CONCOURS : RELEVEZ LE DÉFI DE MOBIOOS. 2250 € À GAGNER !

programmez.com

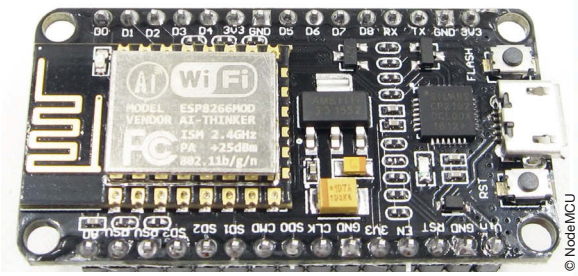
Programmez!

Le magazine des développeurs

231 JUILLET/AOUT 2019

© Amazon

Créer des skills pour Amazon Alexa



© NodeMCU

MicroPython +NodeMCU

TOUT SAVOIR sur C# 8.0

© lloyd

M 04319 - 231 - F: 6,50 € - RD

LE SEUL MAGAZINE ÉCRIT PAR ET POUR LES DEVELOPPEURS

Printed in EU - Imprimé en UE - BELGIQUE 7 € - Canada 9,80 $ CAN - SUISSE 13,10 FS - DOM Surf 7,50 € - TOM 1020 XPF - MAROC 55 DH

Olymp

La formation nouvelle génération

Le logiciel indispensable aux centres de formation pour profiter des avantages de la formation présentielle, et de la formation en ligne.

Edité par

IdeaStudio

EDITO

La techno-dépendance : le cas Huawei

#1

Pour ce numéro d'été, nous proposons beaucoup de bonnes choses : maîtriser Amazon Alexa, créer des jeux pour GameBoy, les API, C# 8, un peu de reverse engineering et du Python sur microcontrôleur ! Et vous pouvez aussi profiter de l'été pour maîtriser une technologie d'avenir : la batterie patate ! Un grand classique de physique-chimie !

#2

Depuis 2018, une véritable guerre commerciale, et technologique, oppose les Etats-Unis et la Chine. Tout tourne autour de Huawei dont les Américains se méfient et opposent la sécurité nationale pour justifier l'interdiction aux entreprises américaines de collaborer, acheter ou vendre des technologies. En Europe, la situation est plus confuse : pas d'interdiction mais « on surveille ».

Le sujet est sensible que ce soit au niveau politique, diplomatique, commercial et technologique. Le constructeur chinois est un acteur important dans les infrastructures télécoms et les réseaux. Ils ne sont pas nombreux, dans le monde, à pouvoir fournir le matériel 5G.

Google a coupé les tuyaux Android et les services maison. L'impact sera important dans les smartphones où Huawei était en forte croissance. Mais l'interdiction de vendre aux USA et les pertes technologiques impacteront le constructeur qui annonce une réduction drastique dans la production des terminaux mobiles. Le Chinois a néanmoins les moyens de contourner une partie du problème : développer son propre OS mobile (il existe une version open source d'Android), se recentrer sur les marchés locaux et régionaux en attendant des jours meilleurs, trouver / développer des alternatives aux technologies américaines. A court terme, Huawei a les moyens de relancer la machine. Et l'ironie de l'Histoire pourrait être inattendue : avec un OS alternatif à Android et de bonnes ventes de smartphones, Huawei pourrait concurrencer l'OS de Google, du moins, sur une partie du marché… Et ce conflit peut aussi inciter des entreprises et des pays à chercher des alternatives et à développer leurs propres technologies pour éviter que les autorités américaines aient la possibilité de couper l'accès aux technologies. Bref, une prise de conscience…

Cette hypothèse reste une hypothèse, mais dans 5 ans ?

#3

N'oubliez pas tous nos liens pour rester informer avec Programmez! :

Github : https://github.com/francoistonic

Podcasts : https://podcast.ausha.co/poddev/

Twitter : @progmag

Et bien entendu : www.programmez.com

Rendez-vous dès le 30 août pour notre 22e saison…

François Tonic
ftonic@programmez.com

SOMMAIRE

Dans le prochain numéro !

Programmez! #232, dès le 30 août 2019

Comment créer des interfaces graphiques en Python ?
Voiture autonome + IA : exemple de programmation
Trop d'écoles d'informatique tuent-elles l'école d'informatique ?

News

Github

À Berlin, Github fait parader son nouveau PDG

Github a fait quelques annonces à l'occasion de sa conférence Satellite qui se tenait à Berlin. L'occasion pour la société rachetée l'année dernière par Microsoft de ramener son nouveau PDG Nat Friedman en Europe. Principale annonce : Github Sponsors, qui permettra de sponsoriser des développeurs en espèces sonnantes et trébuchantes via des dons mensuels. Un peu comme Patreon, mais pour de l'open source en quelques sorte. La plateforme de partage de code a également annoncé plusieurs fonctionnalités sur le front de la sécurité, avec notamment l'acquisition et l'intégration de Dependabot, un outil permettant d'automatiser la mise à jour des dépendances. Github a également affiné son système de permissions avec les rôles « triage » et « maintain » qui viennent s'ajouter aux « read », « write » et « admin » disponibles jusque-là.

Firefox

se laisse tenter par le premium

Mozilla veut proposer une version « premium » de son navigateur aux cotés de la version classique. Cette version demandera à l'utilisateur de payer un abonnement et sera enrichie de plusieurs fonctionnalités intégrées nativement : Chris Beard évoque ainsi des tests portant sur l'intégration d'un VPN ainsi que d'un espace de stockage sécurisé mais précise que l'ensemble des fonctionnalités n'a pas encore été défini. Mozilla espère pouvoir en dire plus au mois d'octobre.

Chrome

et les adblocks

Avec sa nouvelle politique d'extensions, Chrome ne s'est pas fait que des amis. Au mois de janvier, de nombreux développeurs s'inquiétaient ainsi des répercussions de cette nouvelle politique sur l'écosystème des bloqueurs de publicités, qui risquaient de voir leur capacité d'action grandement limitée sous Chrome. On peut également rappeler que Chrome propose son propre bloqueur de publicité « maison ». Dans une série de posts de blog, les équipes de Chrome ont donc tenté de désamorcer la polémique, en assurant sur leur blog que, promis, juré, « Chrome n'essaie pas de tuer les bloqueurs de publicités mais uniquement de les rendre plus sécurisés. » Les ajustements sont pour l'instant toujours en cours. Mais en attendant, les autres navigateurs s'appuyant sur Chromium (Opera, Brave et Vivaldi) ont préféré annoncer qu'ils n'implémenteraient pas les changements sur les extensions dans leurs propres navigateurs. Ambiance au beau fixe dans écosystème Chrome.

Bluekeep

nouvelle source d'angoisse pour sysadmins au bord du gouffre

La nouvelle faille à la mode s'appelle Bluekeep, car oui les bonnes failles ont aujourd'hui un nom, et, parfois même, un site web et un joli logo. Plus sérieusement, cette vulnérabilité immatriculée CVE-2019-0708 affecte le service RDP (Remote Desktop Protocol) sur plusieurs versions de Windows. Identifiée en début d'année, cette faille a été corrigée dans le patch Tuesday du mois de mai. Mais les récents scans montrent encore un peu plus de sept millions de systèmes vulnérables et plusieurs experts s'accordent pour dire que cette vulnérabilité pourrait être exploitée par un mécanisme de type « ver », afin de propager rapidement un malware dans un scenario comparable à Wannacry. Pour l'instant, la faille en question n'a pas encore été exploitée par des groupes malveillants, mais Microsoft et la NSA conseillent vivement d'appliquer les correctifs. En attendant la prochaine faille de la fin du monde qui ne manquera pas de venir prendre la relève.

Gros coup de filet dans les places de

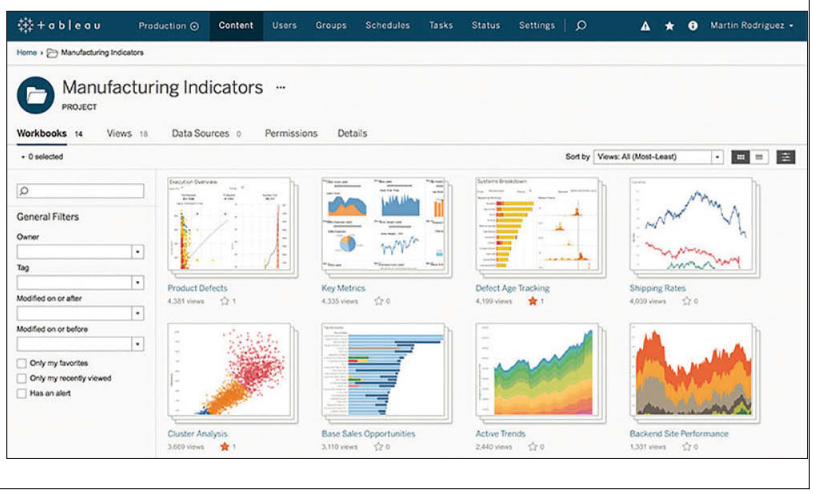
marché illégales

Un an après l'arrestation des administrateurs de la main noire/black hand, les autorités viennent de porter un nouveau coup à l'écosystème francophone des places de marchés illégales dissimulées sur Tor en démantelant French Deep Web. Ce site, en ligne depuis 5 ans, était l'un des principaux forums francophones sur Tor et disposait également d'une place de marché illégale, French Deep Web Market. 3 individus ont été mis en garde à vue suite à l'opération de police du 12 juin, qui a permis de saisir des équipements informatiques ainsi que des cryptomonnaies chez les suspects.

Salesforce

engloutit Tableau pour 15 milliards

Dans la série des rachats pantagruéliques de la tech, c'est au tour de Tableau de passer à la casserole. Salesforce a annoncé son intention de racheter le leader des solutions d'analyse statistique pour la bagatelle de 15,7 milliards de dollars. C'est la plus grosse acquisition menée par Salesforce jusqu'ici mais bon quand on aime, on ne compte pas.



amazon alexa

European Alexa Perks Program

Tu rêves de développer des Skills et d'explorer les multiples possibilités des assistants vocaux ? Viens relever le défi Amazon Alexa !

DE NOMBREUX LOTS SONT À GAGNER !

Amazon Alexa est utilisé sur +100 millions de terminaux dans le monde.

Pour rejoindre ce défi de programmation, c'est très simple :
https://developer.amazon.com/fr/en-gb/alexa-skills-kit/alexa-developer-skill-promotion

1 tu t'inscris

2 tu codes tes Skills

3 tu les publies, avant le 30 novembre

A toi de jouer !

Informations, FAQ, inscription sur : https://developer.amazon.com/fr/en-gb/alexa-skills-kit/alexa-developer-skill-promotion

Juillet

6-7 : leHACK19 / Paris

La nuit du hack change de nom et de dates. Mais le reste c'est comme la Nuit du Hack avec un wargame, des conférences, des ateliers et un grand espace recrutement !
Venez nombreuses et nombreux ! https://lehack.org/fr

Août

7-11 : Apple II Festival France / Auch

Le grand festival dédié à l'Apple II revient pour sa 5e année ! La communauté Apple II se rencontre pour exposer, réparer et discuter autour de leur machine préférée. +20 conférences et ateliers se joueront cette année.
Infos & inscriptions :
https://www.apple2festivalfrance.fr/index.html

30 & 1er septembre : HackTaFerme / région de Toulouse

Venez hacker et innover sur les fermes agricoles ! Le défi est immense. Deux challenges sont proposés :

- 1 réaliser un prototype d'une app dédiée aux producteurs
- 2 travail sur les données pouvant servir aux producteurs et agriculteurs.

Pour en savoir plus : charles.cemay@acta.asso.fr

Septembre

16-17 : Agile en Seine 2019/Paris

Agile en Seine se veut une conférence multipratique, ouverte à tous, dont l'objectif est de mettre en avant toute la diversité de ce que l'on nomme communément « Agilité » : du Scrum à l'agilité à l'échelle, en passant par le Design Thinking, le Kanban, le Lean Startup, le Lean, l'entreprise libérée, l'Holacracy … Pour cette édition 2019, nous vous proposons un nouveau format, sur 1,5 jour, le 16 septembre après-midi et le 17 septembre 2019.
La première demi-journée sera consacrée à l'animation d'ateliers de mise en œuvre des pratiques Agile et la seconde journée à des conférences de partage et des retours d'expérience.
Site : https://www.agileenseine.com

20 : We love speed / Lille

Créé en 2018 par et pour la communauté, We Love Speed est né de l'envie de partager le plus largement possible les connaissances et expériences en matière de webperf. Professionnels du web, de l'e-commerce et experts de la webperf, cet événement est fait pour vous !
Site : https://www.welovespeed.com/2019/

Octobre

3 : DevFest Toulouse 2019/Toulouse

Le DevFest Toulouse aura lieu, pour sa 4e édition, le 03 octobre 2019 au Centre des Congrès Pierre Baudis de Toulouse et réunira 900 développeurs, personnes travaillant dans les métiers techniques de l'informatique et étudiants. En attendant le jour J, le CFP (Call For Papers / Appel à Orateurs) du DevFest Toulouse est désormais ouvert ! Cette année, vous pouvez soumettre du 25 mars jusqu'au 25 mai (2 mois). À mi-CFP, une présélection de nos "early speakers" sera effectuée, ne tardez donc pas à soumettre vos propositions ! La délibération finale se fera ensuite pendant le mois de juin.
Site : https://devfesttoulouse.fr/fr/

21-22 : DevFest / Nantes

Site : https://devfest.gdgnantes.com/fr/

29-31 : Scala IO / Lyon

6e édition de la grande conférence dédiée à Scala et à son écosystème.
Site : https://scala.io

Novembre

6 : DevFest / Strasbourg

Le DevFest, ou "Developers Festival", est une conférence technique destinée aux développeurs. Elle s'adresse aussi bien aux étudiants, aux professionnels ou tout simplement aux curieux, passionnés de technologies.
Site : https://devfest.gdgstrasbourg.fr

Microsoft Experience'19 / Paris

La grande conférence des environnements et technologies Microsoft revient en novembre. Pour le moment, aucune date n'a été annoncée.

CONFÉRENCES ORGANISÉES PAR XEBIA

7-8 Octobre : 4e édition de la FrenchKit

La FrenchKit sera de retour pour une quatrième édition les 7 et 8 octobre prochains. La FrenchKit est la première conférence iOS et macOS en France. Durant ces 2 journées, certains des développeurs les plus reconnus de la communauté internationale traiteront un large éventail de sujets, des API Cocoa à Swift.

28 novembre : XebiCon, Build The Future => https://xebicon.fr/

À travers une soixantaine de conférences, la Xebicon vous donnera les clés pour tirer le meilleur des dernières technologies. Conférences techniques, retours d'experiences clients, hands-on, venez partager et échanger sur les nouveautés technologiques autour du Cloud, de la Data, des nouveaux standards d'Architecture, des nouveaux langages Front-End, de l'IoT, de la culture DevOps, des transformations agiles à l'échelle ou encore de la mobilité.

Girls Can Code !

Les inscriptions pour la 6e édition des stages Girls Can Code! sont ouvertes. Ces stages d'une semaine d'initiation à la programmation pour les collégiennes et lycéennes visent à promouvoir l'informatique auprès des filles dans un cadre agréable et créatif. Les stages Girls Can Code! sont gratuits, sauf pour les éventuels frais de transport et d'hébergement jusqu'au lieu du stage. Encadrés par des étudiants, le niveau est adapté à chacune des participantes et aucun prérequis n'est nécessaire.

Pour l'édition 2019, 3 stages sont prévus :
- EPITA Paris, du 8 au 13 juillet 2019,
- EPITA Paris, du 26 au 31 août 2019,
- ENS Lyon, du 26 au 31 août 2019.

Vous pouvez en apprendre davantage sur Girls Can Code! ainsi que les éditions précédentes sur cette page
https://gcc.prologin.org

Merci à Aurélie Vache pour la liste 2019, consultable sur son GitHub :
https://github.com/scraly/developers-conferences-agenda/blob/master/README.md

Farouk AOUINI
Chief Content Officer Mobioos
farouk.aouini@mobioos.ai

Cet été, créez une application à l'aide de **Mobioos Forge Studio** et tentez de gagner 2250 € !

Mobioos Forge Studio Quezako ?

La solution Mobioos Forge Studio est une extension de Visual Studio 2019 (VSIX) qui permet de générer le code d'une application à partir de modèles dans une multitude de langages. Toutes les couches de l'application sont couvertes (UI, API, Data, etc.), indépendamment des technologies.

Méta-modélisation

Les modèles permettent de définir, entre autres :

- Les fonctionnalités globales de l'application (langue, plateformes supportées, etc.) ;
- Le modèle objet ;
- Les API disponibles (format REST) ;
- Les différents écrans de l'application et les actions associées ;

Les étapes de génération de l'application

1. Création du projet de modélisation
2. Modélisation de l'application via les différents modèles disponibles
3. Validation automatique de la cohérence des modèles
4. Lancement de l'interview avant la génération du code
5. Sélection du ou des langages de génération (en fonction du nombre de plateforme à couvrir)
6. Génération du code source

Un chatbot assistant pour la phase de génération de code

Lors de la génération du code, le développeur est invité à répondre à une rapide interview afin de préciser le langage sélectionné pour chacune des plateformes spécifiées dans le méta-modèle frontend, backend, base de données. Plusieurs Frameworks (frontend et backend) sont d'ores et déjà disponibles (ASPNET Core et Ionic). D'autres langages sont en cours d'intégration (Angular, React Native, Node JS, Python, PHP Laravel, Java, Xamarin). Côté base de données, le développeur a le choix de générer le code SQL (à l'image du modèle objet spécifié dans le modèle data) pour SQL Server, MySQL, PostgreSQL et Oracle.

LE CONCOURS !

Cet été, l'équipe Mobioos Forge Studio organise un concours de développement afin de vous familiariser avec la solution, et pourquoi pas, gagner le premier prix ! Les règles du concours sont simples : développer une application mobile dans au moins une des deux catégories proposées (Productivité et Jeux).

Pour ce faire, rien de plus simple : il suffit de s'inscrire sur le site du concours (https://summerchallenge.mobioos.ai), seul ou en équipe (maximum 3 personnes), puis de télécharger l'extension Mobioos Forge Studio directement via le gestionnaire d'extension de Visual Studio 2019 ou via la Marketplace de Visual Studio.

Pour terminer, vous n'avez plus qu'à créer un compte sur le site https://forge.mobioos.ai afin d'activer la génération en ligne du code.
Pour vous aider à appréhender l'extension, vous trouverez tout ce qu'il vous faut sur l'espace Github de Mobioos.

Une documentation très complète (https://github.com/Mobioos/Forge-Documentation), un QuickStart (https://github.com/Mobioos/Forge-Documentation/wiki/Quickstart) et une application mobile exemple (https://github.com/Mobioos/SmartApp-Sample-Hive) vous guide pas à pas dans l'utilisation de la solution.
Le gagnant de chaque catégorie recevra un chèque de 2 250 €. Les deuxième et troisième recevront respectivement un casque VR Oculus GO et une carte cadeau Amazon d'une valeur de 150€. Pour multiplier vos chances, vous pouvez même concourir à plusieurs catégories.

A vous de jouer !
https://summerchallenge.mobioos.ai

MOBIOOS FORGE STUDIO

1 TÉLÉCHARGER VISUAL STUDIO 2019

2 INSTALLER L'EXTENSION FORGE

3 DÉBUTER AVEC LA MODÉLISATION
Concevez le socle technique et fonctionnel de votre application via les diagrammes Mobioos Forge Studio

4 DÉMARRER L'INTERVIEW DE GÉNÉRATION DE CODE
Répondez aux questions techniques et fonctionnelles du chatbot de génération de code

5 CHOISIR LE(S) LANGUAGE(S) QUE VOUS SOUHAITEZ UTILISER POUR LE CODE DE VOTRE APPLICATION
Sélectionnez le ou les languages de génération (frontend, backend) parmi la liste proposée

6 RÉCUPERER LE CODE SOURCE DE L'APPLICATION
Téléchargez le code source généré

7 ET VOILÀ !
Jusqu'à 60% du code de l'application est généré. Il ne vous reste plus qu'à vous concentrer sur le développement du code métier

Par Mobioos.ai

Roadmap 2019 des langages & des IDE

Chaque mois, Programmez! vous propose un panorama des agendas de sorties des versions des langages, frameworks, etc.

DÉJÀ DISPONIBLE

React 16.8 **Kotlin 1.3.30** **Rust 1.35.0** **TypeScript 3.5** **Angular 8**

PHP 7.1.30 :
mise à jour de sécurité.

PHP 7.2.19 :
mise à jour de sécurité + bug fix

PHP 7.3.6 :
mise à jour de sécurité + bug fix

Symfony 4.3.0

1ER SEMESTRE 2019

TypeScript 3.5

Date de sortie : disponible
Les principales nouveautés attendues / annoncées : la nouvelle version de TypeScript est maintenant disponible. La v3.5 ajoute pas mal de nouveautés et d'améliorations : performances en hausse, une vérification des types plus rapide, --incrémental amélioré, sélection plus intelligente dans le code via la fonction smart select. Tous les détails : https://devblogs.microsoft.com/typescript/announcing-typescript-3-5/
Au-delà : TypeScript 6 est déjà acté pour fin août ! Tous les détails ici : https://github.com/microsoft/TypeScript/issues/31639 et aussi sur https://github.com/Microsoft/TypeScript/wiki/Roadmap

Angular 8

Date de sortie : disponible
Les principales nouveautés attendues / annoncées : Angular arrive très vite et les versions se succèdent à une cadence élevée, tous les 6 mois environ. La v8, qui arrive maintenant, introduit plusieurs améliorations :

- Sur la CLI : possibilité d'utiliser ES5 et ES2015+ avec une amélioration sur le chargement et le build du code JS ;
- Pré-version d'Ivy ;
- Web Worker : amélioration sur les performances et la parallélisation des applications ;
- Mise à jour des dépendances, notamment outils.

Sur la compatibilité, attention : seuls les codes provenant d'Angular 7 sont pris en compte (dans la note officielle). Et aussi : retrait de @angular/http, ngcc -make disponible, diverses corrections de bug. Les équipes préviennent aussi d'un changement provocant une non-compatibilité de code : rajouter impératif de @blazel/typescript dans le package.json
Au-delà : la v9 est prévue pour octobre – novembre.

SWIFT 5.1

Date de sortie : été
Les principales nouveautés attendues / annoncées : ce sera la première mise à jour de SWIFT 5. Cette version doit apporter des corrections de bugs et terminer la stabilité des librairies et des modules. Bien entendu, la 5.1 est rétrocompatible avec la 5.0. On doit s'attendre à quelques changements dans ce que les équipes appellent la « module stability » comme l'apparition du .swiftinterface pour le fichier d'interface à la place de .swiftmodule. Quelques éléments ici : https://swift.org/blog/5-1-release-process/
Attention : la stabilité ABI n'est pas encore totalement disponible sur les plateformes non-macOS.
Au-delà : la prochaine version majeure devrait être la v6.

Ruby on Rails 6

Date de sortie : été
Les principales nouveautés attendues / annoncées : parallèlement au langage, Rails continue à évoluer de son côté. Les grosses nouveautés attendues concernent les actions sur les mailbox, les évolutions sur les fonctions de tests, particulièrement sur l'Action Cable testing et les tests parallèles. Pour la migration, il faudra partir d'une version 5.2 pour faire la transition. La RC1 a été publiée fin avril.

2E SEMESTRE 2019

React 16.9

Date de sortie : sans doute durant l'été.
Les principales nouveautés attendues / annoncées : en attendant la prochaine version « majeure », React 16.8 est régulièrement mis à jour. La version 16.8.6 a été déployée fin mars avec différents bug fix.
Pour suivre les sorties : https://github.com/facebook/react/releases .

Java 13

Date de sortie : septembre - octobre
Les principales nouveautés attendues / annoncées : cette version apportera plusieurs nouveautés et améliorations. Parmi les annonces faites :

- Shenandoah : ramasse-miettes à faible temps de pause ;
- API de constantes JVM ;
- AArch64 : sert à supprimer toutes les sources liées aux arm64port. Le but est de faciliter le portage ARM ;
- archives CDS par défaut ;
- améliorations sur la GC G1.

Kotlin 1.3.40

Date de sortie : été
Les principales nouveautés attendues / annoncées : le développement de cette nouvelle version est en cours. Nous savons d'ores et déjà que les appels trimIndent et trimMargin seront traités par le compilateur et non le runtime. Stabilité de new type inference disponible en pré-version depuis la v1.3.0.

Goland 1.13

Date de sortie : août/septembre.
Les principales nouveautés attendues / annoncées : cette version doit activer le mode module par défaut (le mode par défaut d'auto à activer) tout en déconseillant le mode GOPATH. On aura aussi des nouveautés sur les génériques, la gestion des erreurs.
Au-delà : Go 2 sera LA grosse évolution du langage Go, même si les équipes ne veulent pas faire une version de rupture, mais des évolutions au fil des versions. Un des changements sera la manière de définir les évolutions et comment la gouvernance fonctionne. L'idée est que la communauté soit plus impliquée. La compatibilité avec Go 1.x sera un des aspects cruciaux. Pour le moment, le planning de Go2 reste à préciser.

C# 8.0

Date de sortie : été / automne.
Les principales nouveautés attendues / annoncées : voir dossier du mois.

Python 3.8

Date de sortie : octobre
Les principales nouveautés attendues / annoncées : plusieurs nouveautés devraient plaire aux développeurs. Tout d'abord, on bénéficie du nouveau paramètre pythonpycacheprefix. Il permet d'utiliser le cache bytecode dans une branche séparée du système de fichiers parallèle. Il sera à préférer au __pycache__ présent dans les sous-répertoires des dossiers sources. On disposera aussi de la méthode as_integer_ratio(), dans le type int. Le contrôle-C, SIGINT, sera modifié pour éviter les problèmes liés à l'exception KeyboardInterrupt. Plusieurs modules auront droit à des améliorations comme asyncio avec ProactorEventLoop ou encore dans gc avec de nouveaux paramètres dans le get_objects().
Pour plus de détails : https://docs.python.org/dev/whatsnew/3.8.html .
Au-delà : en toute logique, la prochaine version majeure sera la 3.9. Pour le moment, pas grand-chose n'a été communiqué. Cette version ne devrait pas arriver avant 2021.

Symfony 4.4

Date de sortie : novembre
Les principales nouveautés attendues / annoncées : Symfony 4.4 doit arriver courant novembre. Elle répond à la politique de mise à jour annuelle : 1 en mai, 1 en novembre. Cette version sera LTS.

PHP 7.4

Date de sortie : décembre (?).
Les principales nouveautés attendues / annoncées : la 7.4 doit apporter le préchargement (preloading) au cœur de PHP pour pouvoir améliorer les performances. Ce mécanisme permet de charger l'ensemble des fichiers PHP dès le démarrage et ainsi améliorer les accès pour assurer une disponibilité constante de ceux-ci. Par contre, en cas de changement des fichiers, il faudra redémarrer le serveur. On notera aussi la disponibilité des propriétés typées, l'extension FFI (Foreign Function Interface) pour appeler du code C, le nouvel opérateur Null Coalescing, l'apparition de nouvelles méthodes (__serialize & __unesrialize).
On notera aussi le retrait de PEAR ou la dépréciation de ext/wwdx.
Attention : la 7.4 n'est pas encore totalement figée. Des changements peuvent intervenir.
Au-delà : après la 7.4, difficile d'y voir clair. Il était question d'une v8. Pour le moment, rien de très précis.

Ruby 2.7

Date de sortie : décembre (?).
Les principales nouveautés attendues / annoncées : le langage Ruby continue d'évoluer. La 2.6 est sortie fin 2018, notamment avec un nouveau compilateur JIT. On bénéficie aussi d'un nouveau module RubyVM ::AbstractSyntaxTree. Il analyse une chaîne de caractères et retourne les nœuds d'arbre syntaxique. Un travail d'optimisation a été réalisé.

COURANT 2020

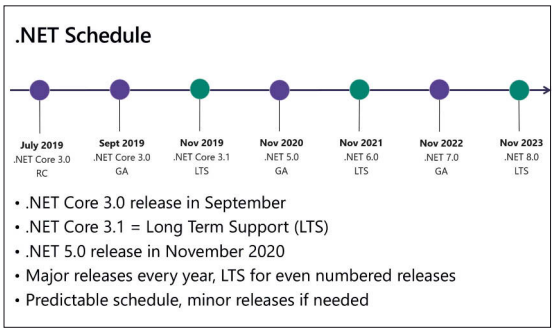
C++20

Date de sortie : 2020.
Les principales nouveautés attendues / annoncées : les spécifications de C++20 sont désormais figées ; un premier document complet sera disponible cet été. Cette future version du langage mettra en avant deux nouveautés : les modules et les coroutines. Les modules constituent une nouvelle alternative aux fichiers d'en-tête qui apportent un certain nombre d'améliorations clés, notamment en isolant les effets des macros et en permettant des compilations évolutives, explique Herb. Il ajoute qu'en 35 ans c'est la première fois que C ++ ajoute une nouvelle fonctionnalité permettant aux utilisateurs de définir une limite d'encapsulation nommée.
Les coroutines sont elles aussi une fonctionnalité à remarquer. Une coroutine est une unité de traitement qui s'apparente à une fonction (ou routine), avec cette différence que si une sortie du corps d'une fonction met fin à l'exécution de celle-ci, la sortie de la coroutine suspend seulement son traitement qui peut ensuite reprendre, l'état du traitement à la sortie étant conservé. Une coroutine peut être vue comme un morceau de programme qui conserve son état, mais qui n'a pas de thread d'exécution. Les coroutines peuvent notamment être utilisées pour des itérateurs et des générateurs.

.Net 5

Date de sortie : novembre 2020.
Les principales nouveautés attendues / annoncées : l'annonce en a été faite à la dernière conférence BUILD. Il s'agit de .Net Core vNext, donc au-delà de .Net Core 3.0. Il s'agit de réunifier les noms. L'ambition est d'être disponible sur Windows, Linux, macOS, iOS, Android, tvOS, webassembly, etc.

Au-delà, Microsoft a annoncé une ambitieuse roadmap :



CÔTÉ IDE

NetBeans 11 : disponible depuis avril. Cette version supporte mieux JDK 11, diverses améliorations sur la partie PHP 7.x, JUnit 5.3.1

Eclipse 2019-06 sera la prochaine version d'Eclipse IDE. Elle sera libérée en juin.

Côté **JetBrains,** on nous annonce :
TeamCity 2019.1 : disponible
- 26 juin : UpSource 2019.1
- 22 juillet : WebStorm 2019.2
- 24 juillet : CLion 2019.1

Android Studio : la version 3.5 est actuellement en bêta. Parmi les améliorations : Project Marble à jour, plug-in Gradle, évolution de la gestion mémoire, rapport de l'usage mémoire généré si besoin. Cette version s'appuie sur IntelliJ IDEA 2019.1.

Spyder : IDE pour Python. Les versions 3.3.x remontent maintenant à août 2018. Pas de nouvelles versions pour le moment.

Qt Creator : la v4.9.0 est disponible depuis avril dernier. IDE pour le C++ et la librairie Qt. Cette version ajoute le support ECMAScript 7. En même temps, l'éditeur a sorti QT Design Studio 1.1.1.

RAD Studio : actuellement, nous en sommes aux sous-versions de la version 10.3 de Rio disponible depuis quelques mois. La 10.3.1 prend en charge iOS 12, utilisation de Ext JS, amélioration des supports Android SDK / NDK, compilation Win32 Clang C++17. Une amélioration de macOS 64 est attendue dans les versions 10.3.x ainsi que diverses améliorations sur Windows et C++. La version 10.4.0 est attendue cet été / automne.

Prix Bernard Novelli 2019-2020

tangente CASIO SIF Société Informatique de France programmez! le magazine des développeurs

Concours de projets informatiques pour collégiens et lycéens.
Participez individuellement, en équipes ou dans le cadre scolaire.

Le trophée d'art numérique offert au vainqueur

- Vous êtes collégien ou lycéen amateur d'informatique ?
- Vous êtes parent d'élève et voulez développer le goût de vos enfants pour l'algorithmique et l'informatique ?
- Vous êtes enseignant et vous voulez donner un objectif à votre enseignement à la rentrée ?

Participez !
Faites participer des jeunes au Prix Bernard Novelli !

Le but des participants :

réaliser un programme informatique autour du jeu et des mathématiques.

• Le calendrier est dorénavant celui de l'année scolaire. Pour 2019-2020 :

- Inscription avant 30 octobre 2019

- Remise du projet avant le 30 avril 2020.

- Palmarès en juin 2020

Un jury composé de représentants des partenaires (Tangente, CASIO, Programmez!, Société Informatique de France) désignera le vainqueur et les mentions liées au niveau scolaire.
Les lauréats gagneront de nombreux prix : œuvre d'art numérique, calculatrice CASIO haut de gamme, déplacement à Paris pour la remise des prix (depuis la métropole).

Réglement sur **www.tropheestangente.com**
Inscription des candidats sur **tropheestangente@yahoo.fr**

LES TROPHÉES tangente

NOUVEAU : organisation dans les établissements scolaires

Depuis cette année, les professeurs de collèges et lycées peuvent organiser le Prix Bernard-Novelli dans leur établissement.
Ils décident des procédures (thème, langage de programmation, participation individuelle ou par équipes...)
Inscription des établissements : **tropheestangente@yahoo.fr**

ETUDE HIRED

Les salaires des développeurs augmentent à Paris. San Francisco de moins en moins compétitif

Si les salaires des développeurs outre-Atlantique se situent donc entre 114 000 et 145 000 dollars dans le secteur des technologies, en France, ce salaire est de 52 000 €, et en progression de 8% par rapport à l'année dernière. Ramené au coût de la vie, la différence n'est toutefois plus aussi frappante.

Pour la quatrième année consécutive, la région de la baie de San Francisco se classe au premier rang des marchés les plus rémunérateurs pour les spécialistes technologiques, avec des salaires qui ont augmenté de 2 % l'an dernier, d'après l'étude States of Salary publiée par Hired, la plateforme de recrutement spécialisée dans les métiers de la technologie, en juin. Ce n'est pas une surprise si l'on considère la montée en flèche des loyers et l'essor du secteur technologique. Cependant, de nombreuses villes - à l'échelle mondiale – pensent être tout aussi compétitives. Le technicien moyen dans d'autres villes aux Etats-Unis a été payé entre 6 à 9 % de plus en 2018 qu'en 2017, et un spécialiste technologique sur trois s'attend à une augmentation de salaire dans les huit mois suivant son entrée en fonction dans une nouvelle entreprise s'il reçoit une évaluation positive de son rendement.

La croissance des salaires reflète également certaines attitudes des spécialistes en technologie à l'égard de leur rémunération et de leur équité. Un employé en technologie sur trois s'attend à une augmentation dans les huit mois suivant son entrée en fonction dans une nouvelle entreprise s'il reçoit une évaluation positive de son rendement, ce qui pourrait exercer des pressions sur les entreprises pour qu'elles augmentent les salaires de façon générale. Et malgré la vague des premières introductions en Bourse de cette année, plus de la moitié des employés du secteur des technologies à l'échelle mondiale (54 %) sont sur le point de renoncer à un salaire plus élevé pour des actions de la société. Mais, 69 % des employés en technologie au Royaume-Uni ne sont pas certains d'accepter des capitaux propres au lieu d'un salaire de base plus élevé.

Les employés du secteur technologique veulent en avoir plus pour leur argent - et ils agiront en conséquence.

Malgré des salaires élevés dans certaines régions de la planète, les demandeurs d'emploi se sentent sous-payés en raison du coût élevé de la vie et des prix de l'immobilier qui atteignent des sommets. Ils ont commencé à remarquer les avantages de la relocalisation, qui comprennent la compensation de certaines dépenses ou l'évitement d'impôts élevés. Pour comparer le coût de la vie dans d'autres villes, Hired a voulu savoir quelle augmentation de salaires serait nécessaire pour maintenir un niveau de vie correct en raison de multiples facteurs qui influent sur le coût de la vie dans une ville tels que le loyer, l'immobilier, les services publics, la nourriture, les impôts locaux et le transport. Les résultats montrent qu'à Paris, il faudrait un salaire de 79 K€, soit entre 20 et 30 K€ de plus en fonction de la spécialité de l'ingénieur. Par exemple, un spécialiste en mobilité est aujourd'hui payé, selon Hired, environ 50 K€ par an. Les « techs » commencent donc à remarquer les avantages qu'il y a à vivre et travailler dans d'autres villes : lorsque Hired a interrogé ses utilisateurs, Seattle et Austin aux Etats-Unis sont arrivées en tête de liste des endroits où ils considéreraient déménager, par rapport à San Francisco qui propose de meilleurs salaires mais où le niveau de vie est bien trop élevé. En parlant de relocalisation, les candidats de Londres disposés à déménager se voient offrir un bon bonus, de près de 17 000 dollars (plus de 12 000 £). Le paysage technologique est en train de changer et, bien que la Silicon Valley continue de montrer la voie, d'autres villes ont beaucoup à offrir.

POPULARITÉ DES LANGAGES

Les classements valent ce qu'ils valent. Si nous regardons les stats GitHub pour le 2e trimestre 2019, nous voyons que JavaScript reste le premier langage des projets déposés sur la plateforme. Mais sur la durée, on constate une baisse régulière depuis 2016 et parallèlement une montée en puissance de Python. En 3e place, Java reste une valeur sûre mais largement derrière. Il est globalement stable.

En 4e, le langage Go montre une belle vitalité avec C++ et, surprise, Ruby. PHP n'est que 7e, C# 9e, Swift 13e, Kotlin 15e mais très loin.

Classement	Langage	Evolution en %
1	JavaScript	19.922% (-2.425%)
2	Python	17.803% (+1.519%)
3	Java	10.482% (+0.591%)
4	Go	7.916% (+0.295%)
5	C++	7.253% (+0.167%)
6	Ruby	6.296% (-0.249%)
7	PHP	5.515% (-0.285%)
8	TypeScript	5.415% (+0.641%)
9	C#	4.001% (+0.659%)
10	C	3.190% (+0.248%)

Si on regarde l'index TIOBE qui se base notamment sur les recherches web, le classement est l'inverse. Mais cela ne signifie pas que le langage soit réellement utilisé. Nous trouvons Java, C et Python. JavaScript est que 7e, C# 6e, C est à la 2e place. Dans TIOBE, Go pointe seulement à 15e place, Swift est 11e et Kotlin seulement 40e !

Juin 2019	Juin 2018	Evolution	Langage	En %
1	1	=	Java	15.004%
2	2	=	C	13.300%
3	4	↗	Python	8.530%
4	3	↘	C++	7.384%
5	6	↗	VB .NET	4.624%
6	5	↘	C#	4.483%
7	8	↗	JavaScript	2.716%
8	7	↘	PHP	2.567%
9	9	=	SQL	2.224%
10	16	↗↗	Assembleur	1.479%

Convergence – divergence des OS : le développeur devra s'adapter !

Pas toujours simple de comprendre la stratégie des éditeurs sur les modèles de développement des applications pour leurs systèmes. Si on prend Apple, Google et Microsoft, il y a des similitudes et pas mal de divergences dans les approches. Et le développeur ?

La rédaction

Faut-il faire converger les OS et si oui, comment ? Faut-il plutôt agir sur les librairies / SDK / frameworks pour les rendre disponibles partout ?

Apple : décliner MacOS X partout et unifier le modèle

Quand on regarde l'ensemble des OS d'Apple, il y a fondamentalement 2 familles :

- macOS : descendant de NeXTStep via MacOS X
- iOS : iOS est un dérivé de macOS en reprenant les fondations du système et en rajoutant les couches nécessaires pour l'adapter. Tous les OS qui suivent (watchOS, tvOS, iPadOS) dérivent de là. Exception de CarPlay qui est un OS particulier.

Sur le modèle de développement, nous avions au départ Objective-C et les librairies / frameworks adaptés à chaque plateforme matérielle (Mac et les matériels iOS et dérivés). Désormais, c'est Swift et les libraires.

Depuis 18 mois, on parle du projet Marzipan pour faire converger les apps iOS vers macOS. Bref, pouvoir porter les apps mobiles sur Mac. Car aujourd'hui, ce sont des univers très différents. Désormais il faut parler du projet Catalyst. Il fera son apparition avec le prochain macOS, Catalina.

Concrètement, Catalyst a un objectif simple : créer des apps Mac depuis des apps iPad. Plusieurs apps Apple le sont déjà. Depuis son XCode, on pourra cocher en plus de iPhone et iPad, Mac pour les builds. Et l'environnement prendra en charge de nombreux supports fonctionnels, mais le dév devrait adapter / modifier un certain nombre de fonctions propres à macOS.

Honnêtement, nous sommes qu'au début du mouvement. Catalina sera disponible cet automne et les détails arriveront rapidement. Et il faudra voir les limites.

« Un code, une équipe » (keynote WWDC)

Au-delà de Catalyst, un important travail est fait pour rapprocher les fondations. Ainsi, traditionnellement, pour construire les interfaces, on disposait de : AppKit (macOS) et de UIKit (iOS). Mais pour permettre la convergence entre tout ce joli monde, il faut une unique couche pour les interfaces. C'est l'ambition de SwiftUI qui remplacent AppKit et UIKit. Il est pur Swift et déclaratif. Avantage : - verbeux et on pourra enfin créer des interfaces plus rapidement et les réutiliser (pas à 100 % mais tout de même). Le dév iOS / mac va apprécier.

Apple annonce que SwiftUI sera natif macOS, iOS, iPad, watchOS et tvOS. Sera-t-il réversé à la communauté comme Swift ? Pas certain.

Google : tout miser sur Flutter avant Fuchsia ?

Depuis 3 ans, on parle d'une nouvelle stratégie système pour unifier Android et ChromeOS. Ce projet a pour nom Fuchsia (ou Andromeda). Pour le moment, rien n'est certain sur cet OS et s'il verra le jour. Pour Google, l'avantage sera d'avoir une plateforme logicielle unifiée. Mais il semble que l'interface sera « adaptée » à chaque terminal. Ce qui est assez logique.

Durant la Google I/O, le nom a été donné en parlant de Flutter et durant la session Android de Fireside Chat. Sans aucune information supplémentaire. Bref, Fuchsia existe et il est en développement.

En attendant, Google mise (très) gros sur Flutter. Flutter est le framework UI pour les terminaux mobiles Android et iOS. Google I/O a révélé les ambitions : proposer Flutter partout ! Ainsi, le framework sera proposé sur desktop, sur les navigateurs, les matériels embarqués. Pour le développer, Google utilise son langage Dart avec du C, C++. Il embarque le moteur de rendu 2D Skia et le moteur textuel, Blink. Flutter pour le Web est une implémentation de Flutter compatible avec le code et le rendu à base de technologies Web standard: HTML, CSS et JavaScript. Les versions desktop et web sont en builds techniques. Nous sommes encore loin d'une version finale.

L'idée générale sera celle vue plus haut avec SwiftUI.

Microsoft : .Net Core 5

Jusqu'à présent, nous avions .Net Framework, .Net Standard, .Net Core et

l'UWP. Sans oublier Xamarin pour le développement mobile et macOS. Par définition, le framework se destine aux machines Windows, .Net Core est multiplateforme, tout comme Xamarin.

Au départ, nous avions les PCL pouvant être déployés sur plusieurs plateformes. Mais pour unifier tout ça et simplifier la vie du développeur, Microsoft propose .Net Standard qui est une couche d'abstraction sur laquelle repose .Net Framework & Core et Xamarin.

.Net Standard définit donc un ensemble d'API indépendant de toute plateforme. Mais cette approche n'est pas universelle et certains éléments restent spécifiques à chaque plateforme. Bref, c'est standard mais pas tout le temps.

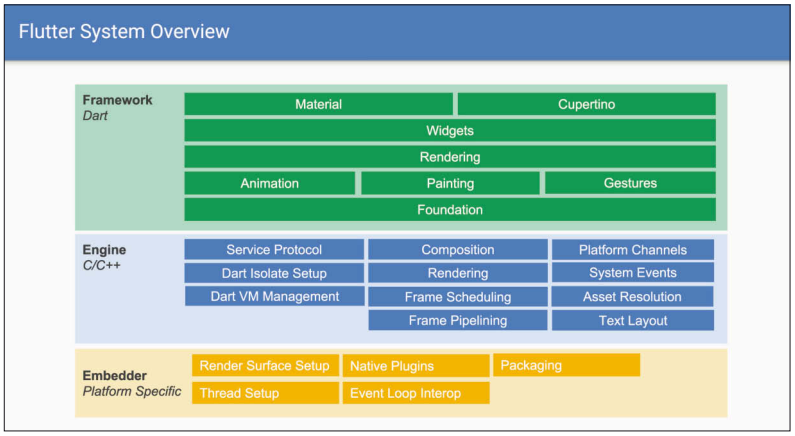
Pour remettre de l'ordre dans l'univers .Net, Microsoft a annoncé .Net 5.0. L'idée directrice de cette version est simple : un unique .Net Core pour toutes les plateformes. On remet l'estampille .Net et on arrête de parler de .Net Core. On évite ainsi de multiplier les noms.

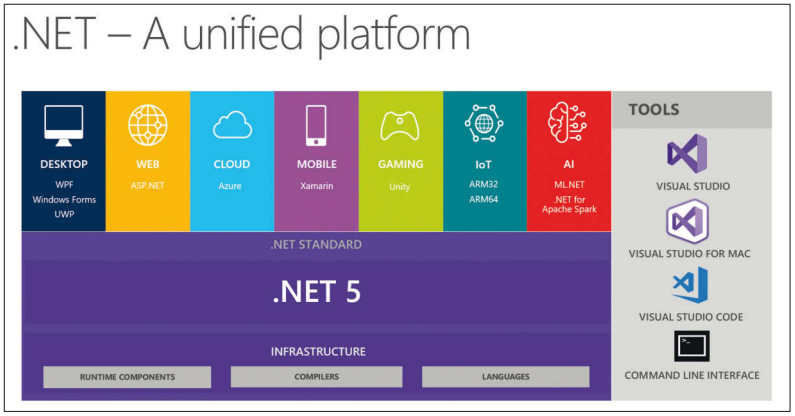
.Net 5 permettra d'avoir un seul runtime et framework pour toutes les plateformes, il reprend le meilleur (selon MS) de .Net Core, Framework, Xamarin et Mono. Mais .Net Core ne disparaît pas réellement. Il continuera à être disponible en open source.

Donc, .Net sera la fondation pour toutes les applications Desktop, web, cloud, mobile, jeux, IoT et IA. WPF, WinForms, UWP, ASP.Net, Azure, Xamarin, Unity, etc. parleront .Net 5 (et accessoirement .Net standard. La v5 est attendue pour l'automne 2020.

Durant plusieurs années, Microsoft avait poussé le modèle des Universal Windows Platform. UWP devait proposer un modèle unique de programmation pour les applications sur tous les environnements Windows et Xbox, en poussant dehors Win32. Rien que sur la partie gaming, cette décision fut très rapidement critiquée et pas question de migrer les jeux Win32 en UWP. Finalement, Win32 reprit sa place. Et réécrire en UWP était loin d'être une évidence même en interne. Et finalement, la contrainte d'utiliser UWP fut jetée.

Conclusion : UWP n'est plus l'avenir des apps Windows et ne sera pas l'unique modèle de programmation. Win32, WPF et WinForms reprennent leur place ! UWP, dans son concept original, est mort. Et on se retrouve avec plusieurs modèles de développement même si .Net 5 va unifier les fondations, avec .Net standard.





« … et ce n'est pas que UWP est mauvais, mais UWP n'est pas une plateforme mature de 35 ans. De très nombreuses apps ont été développées avec (cette plateforme mature). » Joe Belfiore (The Verge, 2017)

Chaque dév peut trouver son modèle de développement mais cela signifie aussi que Microsoft va devoir continuer à supporter x couches et x modèles applicatifs. Ce qui est tout sauf simple.

Pour Microsoft, l'important est au-delà de Windows. D'où cette volonté de proposer les couches .Net à tout le monde. Si Windows 10 est le dernier Windows sous sa forme actuelle (?), quel sera l'avenir de cet OS et quels chemins prendra Microsoft ?

Quelle morale ?

Il est intéressant de voir un certain parallèle entre Apple et Google, l'un avec Flutter, l'autre avec SwiftUI et Catalyst. Et il y a une certaine cohérence de mouvements chez les deux géants. Pour le développeur, même si tout n'est pas clair, il y a l'avantage d'une ligne directrice qui évolue finalement relativement peu (je dis bien relativement peu). Et surtout, on évite de forcer les dévs, tout du moins, pas dans des délais trop courts.

Apple et Google ont su imposer de nouveaux langages, même si la cohabitation avec le langage historique n'est pas simple à gérer. On ne change pas des fondamentaux aussi cruciaux pour des millions de développeurs comme cela.

Microsoft a redéfini les contours de .Net Core et de .Net Framework pour en finir avec les confusions. Mais l'éditeur ne va pour autant exposer un seul modèle applicatif / de programmation. Comme nous l'avons vu, le monde Windows (au sens large) est divers, avec un lourd historique. Microsoft a voulu imposer UWP et en finir avec Win32 (entre autres) mais c'est l'inverse qui s'est passé.

On a encore le douloureux souvenir de Windows Phone qui n'en finissait pas de mourir, faute d'une communication claire sur le sujet.

Comme quoi, on ne peut pas imposer tout et n'importe quoi aux développeurs. Mais ils doivent s'adapter. Peut-être qu'Apple et Google s'y prennent mieux.

Nous vous conseillons de suivre de très près tous ces projets et mouvements car ils ont / auront un impact sur votre quotidien de créateurs de codes. •

Benoit Nachawati
Technical Evangelist
Amazon Alexa
@bnachawati

De Alexa à Pierre, Feuille, Ciseaux, Spock, Lézard : l'aventure de la création d'une Skill

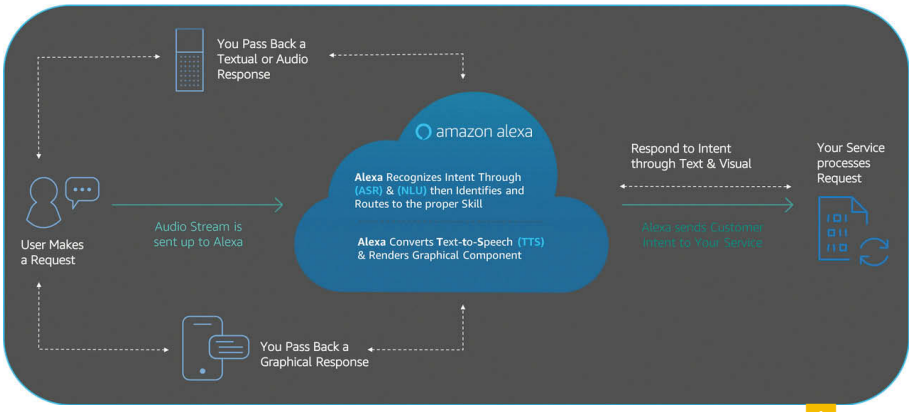
Alexa est le service cloud vocal d'Amazon disponible sur +100 millions d'appareils de la gamme Amazon Echo et de fabricants tiers. Avec Alexa, vous pouvez créer des expériences vocales naturelles offrant aux clients un moyen intuitif d'interagir par la voix. Elle s'intègre davantage dans la vie quotidienne des gens pour répondre à leurs besoins (réveil, minuteur, appel via DropIn, ...). Un simple exemple, elle a chanté "Joyeux Anniversaire" des millions de fois rien que depuis le début de l'année. Dans cet article je vais passer en revue ce qu'est Alexa, son fonctionnement et les possibilités offertes aux développeurs le tout illustré par la création d'une application vocale adaptée du célèbre jeu Pierre-Feuille-Ciseaux-Lézard-Spock. Je vous partagerai également les bonnes pratiques de design et de développement.

Alexa, comment ça marche ?

Pour parler avec Alexa, il faut disposer d'un compte Amazon et d'au moins un appareil Alexa-Enabled que l'on configure sur son compte. L'application mobile Alexa (disponible sur iOS et Android) est aussi un moyen de communiquer avec Alexa.

Alexa vit dans le Cloud Amazon Web Services (AWS [1]) et non dans un appareil Alexa-Enabled. Une interaction avec Alexa commence lorsque le client dit le mot d'activation ("Alexa", "Amazon" ou "Echo" en français) sur son appareil. Une fois le "wake word" détecté, le flux audio de votre demande commence à être diffusé vers le Cloud Alexa. Voici quelques exemples de ce qu'un client peut demander à Alexa : "*donne-moi la météo*", "*donne-moi une recette de gâteau au chocolat*", "*allume le salon*", "*joue du jazz*". C'est ce qu'on appelle une utterance en anglais, ce qui se traduit en français par énoncé ou énonciation.

Alexa va appliquer deux traitements successifs à ce flux pour comprendre quelle était la requête de l'utilisateur. Le premier a pour objectif de transformer cette suite de sons en texte, c'est ce qu'on appelle la reconnaissance automatique de la parole (Automatic Speech Recognition - ASR). Le deuxième traitement a pour objectif de déterminer quelle était l'intention ou l'action principale souhaitée par l'utilisateur, c'est ce qu'on appelle la compréhension naturelle du langage (Natural Language Understanding - NLU). Le résultat de ces deux opérations va produire un document JSON qu'Alexa va envoyer à la fonctionnalité native ou l'app vocale (Skill) capable de répondre à cette demande. Alexa applique ensuite un traitement de Text-to-Speech pour transformer le texte qu'elle reçoit en un stream audio envoyé sur le device de l'utilisateur. Une réponse peut aussi contenir des visuels qui seront affichés sur les devices multimodaux (avec écran) ; certains de ces visuels (les cards) seront aussi visibles sur l'application Alexa. Dans les cas où la réponse contient déjà un flux audio ou vidéo, ce flux sera directement streamé sur le device. 1



1

Créez votre propre appareil Alexa-Enabled

Aujourd'hui, il n'y a pas qu'Amazon qui fabrique des appareils compatibles avec Alexa ! Qu'est-ce que l'enceinte Boulanger Essential B Virtuoz 401 [2], la station connectée Sowee [3], la Freebox Delta [4], la tablette Lenovo Smart Tab M10 [5] et le casque Bose QuietComfort II [6] ont en commun ? Ils intègrent tous une interface vocale capable de communiquer avec Alexa tout comme le ferait un Amazon Echo Show ou Echo Dot. Et c'est le cas de plus de 150 produits intégrant Alexa Voice Service (AVS - alexa.design/get-started). Du point de vue de l'utilisateur, l'expérience sera la même quel que soit l'appareil Alexa-Enabled utilisé. Pour rajouter une interface vocale à votre produit, il vous suffit d'un compte Amazon Developer [7], d'un microphone, d'un processeur pour exécuter le SDK AVS (alexa.design/AVSdeviceSDK), d'une connectivité Internet pour envoyer de l'audio vers le cloud et d'un haut-parleur pour la lecture audio. Un tutoriel en ligne est disponible depuis la documentation AVS [8] pour créer votre propre appareil Alexa-Enabled à partir d'un Raspberry Pi.

Les Skills, c'est quoi ?

Une Skill est une application vocale qui vous permet de proposer vos contenus sur Alexa. Il y a plus de 80 000 Skills disponibles dans le monde et elles sont accessibles aux utilisateurs depuis le Skill Store sur l'application Alexa (alexa.amazon.fr) ou bien le site internet d'Amazon (amazon.fr/skills). Le Store est organisé en catégories (ex. "Maison Connectée", "Jeux et culture générale, "Enfants", ...) pour permettre une découverte plus rapide. Les Skills disponibles

sur le Store sont liées aux langues qu'elles proposent. Pour qu'une Skill soit disponible en France, celle-ci doit inclure une version française. L'activation d'une Skill se fait depuis le Store ou directement depuis un appareil Alexa-Enabled en invoquant la Skill. Une fois activée, une Skill est disponible sur tous les appareils rattachés à ce compte.

Pour créer une Skill, il suffit d'avoir un serveur qui comprend le document JSON envoyé par Alexa et répond avec un autre document JSON. Le plus simple est de déployer une architecture serverless en utilisant AWS Lambda [9]. Mais c'est aussi possible d'utiliser un serveur HTTPS.

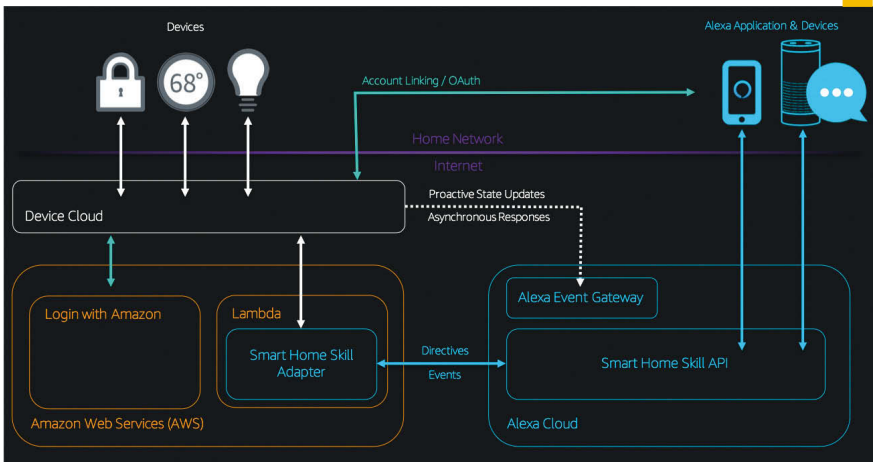
Vous allez définir l'interface vocale qui indique à Alexa comment interpréter les phrases mentionnées par un utilisateur de votre Skill. C'est ce qu'on appelle un modèle d'interaction. Pour traiter les requêtes envoyées par Alexa, vous allez écrire du code correspondant à la logique pour donner une réponse adéquate à l'utilisateur. Le Alexa Skills Kit (ASK) vous proposent différents types pour s'adapter au mieux à vos besoins :

Flash Briefing : diffuser des nouvelles

Lorsque vous souhaitez fournir du contenu informatif court mis à jour régulièrement (exemples : Actualités, Sports, Finances, ...), vous allez créer une Skill Flash Briefing (FB). Le modèle d'interaction est déjà prédéfini par Amazon et votre backend correspond à un flux RSS au format XML ou JSON à héberger sur un endpoint HTTPS publique. Ce flux contiendra les différentes nouvelles (items du flux) qui seront mentionnées par Alexa. Le type de contenu à fournir peut-être soit du texte (Alexa utilisera son TTS pour le transformer en audio avec sa voix) comme pour le FB de Ouest France [10], soit des fichiers audio (mp3) comme pour le FB de L'Équipe [11] ou vidéo (mp4) comme pour le FB LCI [12] qui seront streamés depuis l'appareil Alexa-Enabled.

Typiquement, un FB fait partie de la routine quotidienne d'un utilisateur. Pour y accéder il suffit de dire *"Alexa, quelles sont les infos"* depuis ces devices. Alexa lira alors l'ensemble des FB activés par l'utilisateur selon l'ordre prédéfini par celui-ci depuis l'application Alexa. D'un point de vue usage, pour augmenter votre engagement client, veillez à proposer des nouvelles différentes chaque jour sous forme de source audio d'une durée d'une à deux minutes maximum chacune.

2



SmartHome : faites de la domotique avec Alexa

Lorsque vous souhaitez contrôler des objets connectés par la voix (exemples : ampoules, prises, thermostats, ...), vous allez créer une Skill Smart Home (SH). Tout comme les FB, le modèle d'interaction est déjà prédéfini par Amazon. Votre backend prendra la forme d'une fonction AWS Lambda où votre code sera le proxy entre Alexa et l'infrastructure qui gère les appareils domotiques d'un utilisateur. Concrètement, ce n'est pas Alexa qui allume ou éteint une ampoule mais l'infrastructure du fournisseur de service suite à une requête provenant d'Alexa.

Pour déterminer les objets à contrôler, toute Skill SH doit associer l'identité de l'utilisateur Alexa à une identité dans le système du fournisseur de service. C'est ce qu'on appelle la liaison de compte (Account Linking - alexa.design/accountlinking). Cette association est basée sur OAuth 2.0 [13] et doit prendre en charge l'autorisation via un code (Authorization Code Grant). Cette étape est initiée par l'utilisateur au moment de l'activation de la Skill depuis l'application Alexa.

Les commandes vocales de l'API SH sont disponibles via des directives qui sont organisées en interfaces. Les interfaces décrivent les capacités fonctionnelles de l'objet. Les directives sont les primitives fonctionnelles permettant d'utiliser des commandes génériques pour différents types d'objets. Chaque objet est à décrire par une liste d'interfaces et directives qu'il supporte au moment de la découverte des appareils (*"Alexa, découvre mes appareils"*). Voici quelques exemples de mapping entre utterances et interfaces/directives pour une ampoule connectée pouvant changer de luminosité et de couleur :

```
Alexa.PowerController.TurnOn : "Alexa, allume la cuisine"
Alexa.PowerController.TurnOff : "Alexa, éteins la cuisine"
Alexa.BrightnessController.SetBrightness : "Alexa, règle la luminosité de la cuisine à 40%"
Alexa.ColorController.SetColor : "Alexa, change la lumière de la cuisine en bleu"
```

Une fois les appareils domotiques découverts, ils apparaîtront dans l'application Alexa et un utilisateur pourra les commander soit par la voix, soit via l'application mobile Alexa. Dès qu'un appareil est ajouté, mis à jour ou supprimé par l'utilisateur dans votre système, vous pourrez notifier Alexa de cette modification en envoyant une requête HTTPS POST sur la gateway Alexa (endpoint RESTful) depuis votre infrastructure domotique. 2

Custom Skills : créez votre propre expérience vocale

Lorsque vous souhaitez fournir une expérience vocale complète (exemples : recettes de cuisines, horaires des transports, quizz, ...), vous allez créer une Custom Skill. Vous êtes responsable de la définition du modèle d'interaction et du code à produire pour traiter les requêtes envoyées par Alexa. Le modèle d'interaction est un artefact déployé dans le Cloud Alexa. Concrètement, il s'agit d'un fichier JSON où vous spécifiez l'ensemble des intentions comprises par votre Skill, des exemples de phrases (utterances) pouvant déclencher ces intentions, ainsi que la liste des entités (slots sur Alexa) incluses dans vos intentions avec un catalogue de valeurs associées.

```
{
 "interactionModel": {
```

```
"languageModel": {
  "invocationName": "...",
  "intents": [...],
  "types": [...]
  }
 }
}
```

Le Alexa Skills Kit (ASK) propose une liste d'intentions prédéfinies (built-in intents [14]) pour des actions courantes (AMAZON.YesIntent, AMAZON.NoIntent, AMAZON.StopIntent, ...). Chacune de ces intentions a déjà été entrainée par Alexa et vous n'avez pas besoin de fournir des utterances d'exemples. De la même manière, ASK propose une liste de catalogues de valeurs (built-in slot types [15]) qui définissent la manière dont les données d'un slot sont reconnues et traitées (AMAZON.DATE, AMAZON.NUMBER, AMAZON.Food, ...).

Pour qu'un utilisateur commence à interagir avec votre Custom Skill, il doit l'invoquer explicitement en mentionnant son nom d'appel (invocation name) dans sa demande. Cela peut être soit un nom (comme une marque, ex. : 'Monoprix' [16]) ou bien une suite de mots. L'invocation name est définie par le développeur dans son modèle d'interaction. Par exemple pour accéder à la Skill de l'émission Affaires Sensibles de France Inter [17], l'utilisateur pourra soit faire un lancement modal où seulement le nom d'invocation sera mentionnée "*Alexa, lance Affaires Sensibles*", soit par un lancement "one-shot" où l'utilisateur mentionne le nom d'invocation et sa demande en une seule requête "*Alexa, demande à Affaires Sensibles l'aventure de Neil Armstrong*".

Un modèle d'interaction permet à un utilisateur d'interagir avec la Skill dans une langue donnée (exprimée par une locale). Une Custom Skill peut prendre en charge une ou plusieurs langues. D'un point de vue développeur, il faut créer un fichier JSON par locale supportée (par exemple fr-FR.json). D'un point de vue utilisateur, il communique avec Alexa dans une langue donnée, celle correspondant à la langue configurée sur son appareil Alexa-Enabled. Par exemple, si un utilisateur français configure son appareil en allemand (la locale est alors de-DE), il pourra activer et utiliser toutes les Skills disponibles sur le Skill Store français et ayant un modèle d'interaction pour l'allemand.

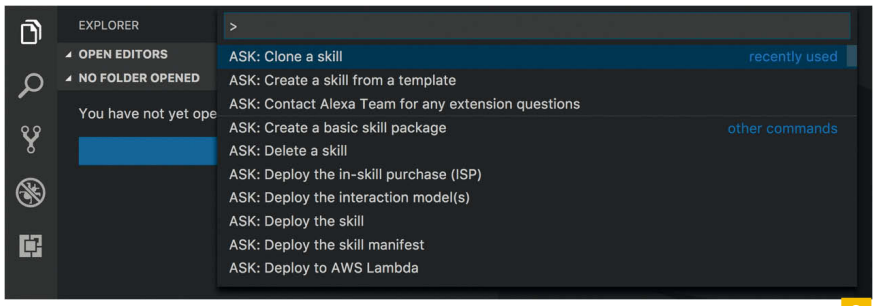
Pour héberger votre web service, vous pouvez soit le déployer sur une fonction AWS Lambda, soit sur un endpoint HTTPS public.

Alexa Skills Kit, il y a quoi dans ta boîte à outils ?

ASK propose un ensemble d'outils et de SDKs accessibles en libre-service pour développer une Skill. Certains services AWS sont aussi compatibles avec les Skills pour vous faciliter leur déploiement. Je vous propose d'en faire un petit tour :

Alexa Developer Console

Le portail Amazon Developer propose une console de développement Alexa (developer.amazon.com/alexa/console/ask) pour créer, gérer et publier vos Skills. Vous pouvez configurer les informations de vos Skills telles que son type (Custom, FB, SH), le endpoint ou flux de contenu et les propriétés de publication (nom, description, icône, keywords). Un dashboard des métriques de vos Skills et un simulateur de test complètent cette console.



3

Command-Line Interface

ASK CLI (alexa.design/cli) permet d'avoir un accès programmatique à la console Alexa pour gérer les Skills et les ressources associées en dehors du portail Amazon Developer. La CLI comporte des fonctionnalités permettant de déployer des modifications et de simuler des appels à des fins de test sur vos Skills. Elle est écrite en Node.js et appellent l'API de gestion des Skills SMAPI (interfaces HTTP RESTful). Pour les développeurs utilisant Visual Studio Code [18], une extension ASK Toolkit [19] est disponible permettant un meilleur usage de ASK dans VS Code. 3

Alexa-Hosted

Disponible pour les Custom Skills, Alexa-Hosted vous permet de développer votre Skill entièrement depuis la console Alexa. Des ressources Cloud AWS sont provisionnées par Alexa pour vous sans avoir besoin de compte AWS. Vous avez accès à une fonction AWS Lambda pour héberger votre code, un bucket Amazon S3 pour gérer la persistance et les fichiers médias ainsi qu'à Amazon CloudWatch pour accéder à vos logs. Un éditeur de code web vous permet de modifier votre code en Node.js et de le déployer directement sur AWS Lambda. Alexa-Hosted est aussi compatible avec la ASK CLI selon une structure de repository Git composée de trois branches : dev (état courant), master (état de la version de dev), prod (état de la version live).

SDK

Pour simplifier le traitement des requêtes dans votre backend, Alexa propose un SDK (alexa.design/sdk) en Node.js, Java ou Python pour implémenter les fonctionnalités d'une Custom Skill (mêmes concepts sur les 3 langages). Le SDK gère nativement le routing des requêtes, la gestion des exceptions ainsi que la sérialisation / désérialisation des requêtes et réponses. Il intègre aussi la gestion des attributs dans les sessions de Skills. Vous avez aussi accès à des utilitaires pour générer les réponses à envoyer à Alexa et différents clients pour appeler les API de services Alexa.

Intégration Services AWS

Le Alexa Skills Kit s'intègre avec trois services AWS différents pour automatiser le déploiement de votre Skill. L'utilisation de ces services est optionnelle pour développer une Skill mais peut accélérer votre développement. Avec AWS CloudFormation [20], vous pourrez décrire et provisionner de manière automatisée les artefacts de votre Skill dans une stack à côté de ressources AWS selon une des-

cription JSON ou YAML. Cela vous permettra de gérer votre infrastructure comme du code.
Avec AWS CodePipeline [21], vous pourrez configurer une chaîne d'intégration et de déploiement continu (CI/CD) sur votre Skill. Tout type de changement dans votre Skill (modèle d'interaction, metadata ou code) seront pris en compte.
Enfin, AWS CodeStar [22] propose un template de projet dédié aux Skills qui fournit une chaîne de CI/CD préconfigurée en utilisant CodePipeline et CloudFormation, un repository git (GitHub ou AWS CodeCommit [23]) et un dashboard de gestion de projet intégrable avec JIRA [24] pour travailler en équipe.

Commencez par l'humain et non l'implémentation

Ne commencez pas immédiatement avec une mise en œuvre technique, pensez au préalable si votre idée est adaptée pour la voix. Le design d'une application vocale est foncièrement différent d'une application web ou mobile. Il n'y a pas de formule magique, c'est d'abord un processus à suivre. Donc, sans plus attendre... Voici les grandes étapes à suivre pour designer votre Skill :

Définir un cas d'usage qui ait du sens

Il s'agit de définir un cas d'usage solide pour lequel vous connaissez votre audience, le type de contenu que vous proposez et la fréquence de mise à jour du contenu. Il faut déterminer comment vous allez vous intégrer dans le quotidien des utilisateurs pour savoir à quel moment de la journée et dans quelle situation votre Skill sera utilisée.

Écrire votre scénario

Une fois que l'on a établi ce que l'utilisateur allait pouvoir faire avec la Skill, il faut réfléchir à comment il va pouvoir le faire. On va commencer par écrire les dialogues entre l'utilisateur et la Skill pour accomplir certaines actions que l'on propose. Il faut en écrire plusieurs (une dizaine est un bon début) pour que vous puissiez avoir une meilleure idée de la structure de votre Skill. Pensez à considérer trois types de formulation pour comprendre les différentes manières dont chacun peut s'exprimer : l'impératif, l'infinitif et la forme conjuguée. Alexa respecte la grammaire et la syntaxe française : prenez du temps pour écrire avec la bonne orthographe et ponctuation ! A la fin de cette étape, vous allez avoir une pièce de théâtre avec plusieurs scènes, c'est votre scénario.

Définir son storyboard

Pour définir votre storyboard, vous allez décomposer votre scénario en situations qui représentent chacune un tour (aller-retour) de conversation entre l'utilisateur et Alexa. Chaque situation sera composée de ce qu'a dit un utilisateur (utterance), du contexte dans lequel il se trouve, de la réponse d'Alexa et comment continuer la conversation (prompt). Cette étape vous permet de déterminer si votre dialogue est intuitif et respecte les quatre principes du design vocal (alexa.design/guide). Des templates de storyboard sont disponibles depuis le GitHub Alexa (alexa.design/situational-design-templates). Si l'on reprend notre cas d'usage, à la fin de cette étape, vous allez avoir des délivrables de ce type : 4

Test, test, test, ...

Vous n'allez pas trouver le bon flow du premier coup, donc testez puis améliorez et testez à nouveau. Il faut tester votre expérience complète pour s'assurer qu'elle soit la plus logique et intuitive possible. C'est essentiel de le faire à l'oral pour que le dialogue soit naturel. Par écrit, on ne se rend pas compte de la longueur du texte, de la lourdeur des mots utilisés : nous ne sommes pas habitués à écrire pour le vocal. C'est mieux de s'enregistrer parce que l'intonation peut changer la compréhension de l'utilisateur. Cette approche vous aidera à mettre la bonne ponctuation dans vos prompts. Depuis la console Alexa, le simulateur de test en mode "Voice & Tone" permet d'entendre le rendu final des prompts dans toutes les langues supportées par Alexa. Utilisez-le en saisissant du SSML (Speech Synthesis Markup Language - alexa.design/ssml) pour impacter la prosodie ou la voix utilisée par Alexa.

Création de la Skill Pierre-Feuille-Ciseaux-Lézard-Spock

Pour réaliser le développement de notre cas d'usage, nous allons créer une Custom Skill en mode Alexa-Hosted depuis la console

4

	1st turn	2nd turn	3rd turn	4th turn	5th turn	6th turn
UTTERANCES	• "Alexa, ouvre pierre feuille ciseaux" • "Alexa, lance pierre feuille ciseaux"	• "Pierre" • "Feuille" • "Ciseaux" • "Spock" • "Lézard"	• "crocodile" • "superman"	• "aide" • "aide-moi"	• "commence une nouvelle partie"	• "stop" • "annule"
SITUATION	Lancement Skill Init de la session de jeu Demande de réponse	Option de jeu reconnu Test vs la valeur de la Skill Demande de réponse	Option de jeu non reconnu Demande de reformulation	Information demandée Demande de réponse	Information demandée Demande de réponse	Arrêt demandé Fin de session
PROMPT	Bievenue sur le jeu Pierre Feuille Ciseaux (version Spock). Nous avons 5 manches pour nous départager. Vous pouvez choisir entre Pierre, Feuille, Ciseaux, Lézard ou Spock pour jouer! J'ai fait mon choix et vous, Quel est votre choix ?	J'ai choisi le lézard et Le lézard empoisonne Spock. Je mène 1 à 0. Que jouez-vous pour la manche suivante ?	Désolé, je n'ai pas compris. Vous pouvez choisir entre Pierre, Feuille, Ciseaux, Lézard ou Spock. Pouvez-vous reformuler?	Ici, les règles classiques du jeu Pierre Feuille Ciseaux s'appliquent, mais il faut ajouter que (...) J'ai fait mon choix et vous, que choisissez-vous ?	Commençons une nouvelle partie. Mon choix est déjà fait et vous, que jouez-vous ?	A la prochaine fois !

Alexa (developer.amazon.com/alexa/console/ask/create-new-skill). Nous allons l'appeler "Pierre Feuille Ciseaux". L'intérêt d'utiliser Alexa-Hosted est d'éviter d'avoir à gérer l'infrastructure cible de déploiement et laisser Alexa le faire pour nous.
Une fois la Skill créée, nous allons utiliser la commande `ask clone` de la ASK CLI pour en obtenir une copie et continuer le développement sur notre machine même en étant déconnecté.
Pour récupérer l'identifiant de la Skill depuis le portail, allez sur la homepage de la console Alexa et cliquez sur '*View Skill ID*' (lien dispo juste sous le nom de la Skill). L'identifiant d'une Skill commence par *amzn1.ask.skill*.

```
$ ask clone --skill-id <votre skill_id>
```

Une fois la commande exécutée, vous obtiendrez un répertoire avec la structure suivante :

```
|--Pierre_Feuille_Ciseaux
 |-- .ask // <- Configuration ASK CLI
 |-- .git // <- Configuration repo AWS CodeCommit
 |-- hooks // <- scripts optionnels pré/post déploiement
  |-- pre_deploy_hook.sh // <- .sh ou .ps1 selon OS
 |-- lambda // <- Code source
  |-- index.js
  |-- package.json
  |-- util.js
 |-- models // <- modèle d'interaction
  |-- fr-FR.json // <- un fichier par locale
 |-- skill.json // <- metadatas (propriétés de publication ...)
```

La Skill complète est disponible sous GitHub dans ce repository : github.com/nachawat/skill-nodejs-rock-scissor-paper
Dans ce tuto, on va se concentrer sur les parties de code spécifique au Alexa Skills Kit à produire, pour les autres parties du code, on ira les récupérer depuis le repo GitHub.

Ajout des nouvelles intentions

Nous allons commencer par rajouter les intentions identifiées pendant l'étape de design en éditant le fichier `fr-FR.json`.
Pour gérer la collecte de choix possible pendant un tour de jeu, on va créer une intention `GameAction` avec un slot `action` qui utilise un catalogue de valeurs custom que l'on va appeler `ListOfActions` :

```
{
 "name": "GameAction",
 "slots": [
 {
  "name": "action",
  "type": "ListOfActions"
 }],
 "samples": [
  "que dis-tu de {action}",
  "je joue {action}",
  "{action}"
 ]
}
```

Il est possible d'associer un identifiant unique (IDs) et des synonymes pour chaque valeur du catalogue `ListOfActions` qui auront la même signification. Alexa utilisera sa résolution d'entité pour corréler chaque valeur avec les synonymes et IDs définis en fonction du contenu mentionné dans l'utterance de l'utilisateur. S'il existe une correspondance, ces informations seront rajoutées dans la requête JSON envoyée par Alexa. Voici la définition des différentes valeurs possibles dans le jeu :

```
{
 "name": "ListOfActions",
 "values": [
 {
  "id": "5",
  "name": {"value": "lézard"}
 },
 {
  "id": "4",
  "name": {"value": "spock"}
 },
 {
  "id": "3",
  "name": {"value": "ciseaux"}
 },
 {
  "id": "2",
  "name": {"value": "feuille"}
 },
 {
  "id": "1",
  "name": {"value": "pierre"}
 }
 ]
}
```

Pour gérer la possibilité de commencer une nouvelle partie, nous utiliserons l'intention prédéfinie `AMAZON.StartOverIntent` qui prend déjà en compte cette notion et nous rajouterons des utterances pour comprendre les mots "jeu", "partie", "nouvelle" :

```
{
 "name": "AMAZON.StartOverIntent",
 "samples": [
  "je veux recommencer",
  "commence une nouvelle partie",
  "commence un nouveau jeu",
  "un nouveau jeu"
 ]
},
```

Dernière étape avant la sauvegarde, on s'assure que le nom d'invocation correspond bien à celui choisi pendant le design :

```
"invocationName": "pierre feuille ciseaux",
```

Note : *une Alexa-Hosted Skill arrive avec une intention HelloWorldIntent par défaut, n'hésitez pas à la supprimer de votre modèle!*

Ajout des handlers pour traiter les nouvelles requêtes d'intention

Chaque handler doit implémenter l'interface RequestHandler qui consiste en deux méthodes : une fonction canHandle() qui renvoie un boolean utilisé dans le routing des requêtes et une fonction handle() qui contient la logique de réponse renvoyant un JSON. En général, vous utiliserez l'objet ResponseBuilder (accessible depuis le paramètre handlerInput) pour construire la réponse. Dans notre modèle d'interaction, nous avons défini 2 intentions supplémentaires (GameIntent et AMAZON.StartOverIntent). Dans le fichier index.js, on va donc créer un handler pour chacune de ces intentions :

```
const NewGameIntentHandler = {
  canHandle(handlerInput) {
    return handlerInput.requestEnvelope.request.type
      === 'IntentRequest'
      && handlerInput.requestEnvelope.request.intent.name
      === 'AMAZON.StartOverIntent';
  },
  handle(handlerInput) {
    const output = prompts.newGame(handlerInput);
    session.resetGame(handlerInput.attributesManager);
    return handlerInput.responseBuilder
      .speak(output.speechText)
      .reprompt(output.repromptText)
      .getResponse();
  }
};
```

Astuce : *lorsque vous posez une question dans un prompt, il faut toujours laisser la session ouverte pour que l'utilisateur puisse répondre. Une bonne pratique est de toujours rajouter un reprompt. Ainsi en cas de non réponse, Alexa attendra 8 secondes avant de mentionner le texte du reprompt. Après un deuxième délai de 8 secondes, si l'utilisateur reste toujours muet, Alexa fermera automatiquement la session. En utilisant le SDK, il suffit d'inclure le reprompt via la méthode* reprompt() *depuis* le ResponseBuilder *ce qui laissera la session ouverte.*

```
const GameActionIntentHandler = {
  canHandle(handlerInput) {
    return handlerInput.requestEnvelope.request.type === 'IntentRequest'
      && handlerInput.requestEnvelope.request.intent.name === 'GameAction';
  },
  handle(handlerInput) {
    const { requestEnvelope, attributesManager, responseBuilder } = handlerInput;
    // Récupération de l'identifiant du slot
    const actionSlotValue = Alexa.getSlot(requestEnvelope, "action");
   const entityResolution = actionSlotValue.resolutions.resolutionsPerAuthority[0];
    const userAction = parseInt(entityResolution.values[0].value.id, 10);
    // Logique de jeu
    const roundResults = logic.play(userAction);
    // ajout des scores en session
    session.updateScores(attributesManager, roundResults);
    // Get des prompts
    const output = prompts.gameAction(handlerInput, roundResults);
    // Ajout du reprompt
    if (output.repromptText) {
      responseBuilder.reprompt(output.repromptText);
    } else {
      // Demande de fermeture explicite d'une session
      responseBuilder.withShouldEndSession(true);
    }
    // Génération du JSON de sortie
    return handlerInput.responseBuilder
      .speak(output.speechText)
      .getResponse();
  }
};
```

Note : *pour le slot* action *de l'intention* GameIntent, *on utilise la résolution d'entités d'Alexa pour récupérer l'id associée à la valeur parlée de l'utilisateur. Pour réaliser cela, on va se servir de l'objet* resolutions *associé au slot. Attention, cet objet est seulement disponible lorsque vous utilisez un type de slot custom ou bien lorsque vous étendez un type prédéfini. Pour vérifier si Alexa a résolu une valeur fournie par l'utilisateur à une valeur de votre catalogue, pensez à utiliser la propriété* resolutions.resolutionPerAuthority[].Status.code.

Définir des handlers ne suffit pas pour qu'ils soient pris en compte par le SDK ! Il faut les référencer sur l'instance du SDK créée. A la fin du fichier index.js, une fois que vous aurez trouvé le builder Alexa.SkillBuilders.custom() rajoutez-lui les deux handlers en utilisant la méthode .addRequestHandlers():

```
Alexa.SkillBuilders.custom()
.addRequestHandlers(
  LaunchRequestHandler,
  GameActionIntentHandler,
  NewGameIntentHandler,
  HelpIntentHandler,
  CancelAndStopIntentHandler,
  SessionEndedRequestHandler,
  IntentReflectorHandler)
```

Note : *nous avons déjà supprimé l'intention* HelloWorldIntent *dans le modèle d'interaction, nous pouvons donc supprimer le handler* HelloWorldIntentHandler *et l'enlever de la chaîne des handlers.*

Sur le handler qui gère les requêtes de lancement de la Skill (LaunchRequestHandler), on va modifier la méthode handle() pour renvoyer les prompts et reprompts souhaités pendant notre étape de design :

```
handle(handlerInput) {
 const output = prompts.welcome(handlerInput);
 return handlerInput.responseBuilder
  .speak(output.speechText)
  .reprompt(output.repromptText)
  .getResponse();
}
```

En cas d'erreurs dans l'un de nos handlers, elles seront gérées par le gestionnaire d'exceptions du SDK qui prend aussi la forme de handlers d'erreurs implémentant l'interface ErrorHandler. Pas besoin de déclarer ce handler dans notre code, il a déjà été inclus au moment de la création de la Skill par Alexa-Hosted et s'appelle ErrorHandler.

Dans les différents morceaux de code ci-dessus, vous avez surement remarqué que l'on fait appel à des fonctions pour la logique du jeu (ex. : logic.play()), les prompts (ex. : prompts.welcome())et la gestion des scores des joueurs (ex. : session.updateScores()) qui n'existent pas encore dans notre code. On va maintenant rajouter ces différents éléments.

Logique de jeu

C'est simple : on va représenter chaque action possible par un entier (Pierre = 1, Feuille = 2, ...). Pour chaque manche, on va calculer la combinaison entre le choix de l'utilisateur et le choix de l'adversaire en utilisant la formule suivante : (userChoice x 10) + alexaChoice. Et pour chaque combinaison, on va dire si l'utilisateur remporte (+1), perd (-1) ou est ex-aequo (0). Le premier à avoir atteint le score de 5 sera déclaré vainqueur. On va créer un nouveau fichier logic.js dans le même répertoire que notre fichier index.js et on va copier-coller le code depuis le repo GitHub de cet article : github.com/nachawat/skill-nodejs-rock-scissor-paper/blob/master/lambda/logic.js

Pour utiliser les fonctions de ce nouveau module dans nos handlers, on va l'importer dans le fichier index.js avec la ligne de code suivante :

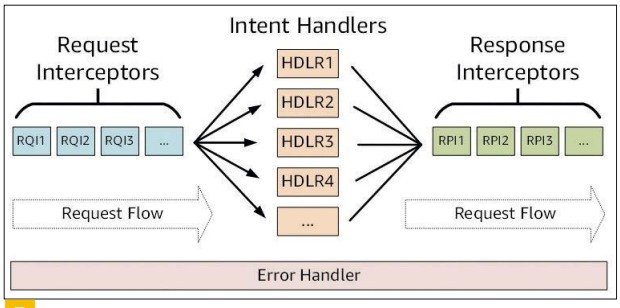
```
const logic = require('./logic');
```

Récupération des prompts

Pour gérer les prompts et reprompts à envoyer à Alexa, vous pouvez choisir de faire appel à une base de données, une API tierce, ou bien les stocker localement dans votre backend. Dans cet article, j'ai choisi de récupérer ces informations depuis un fichier stocké en local disponible depuis ma fonction AWS Lambda en utilisant le concept d'intercepteurs offert par le SDK.

Quel que soit le handler exécuté dans votre code, les intercepteurs de requêtes seront appelés immédiatement avant l'exécution du handler sélectionné et les intercepteurs de réponses seront appelés immédiatement après l'exécution. Il est possible d'en définir plusieurs et ils seront tous invoqués selon leur ordre d'enregistrement. L'objectif est de mutualiser une logique commune qui s'applique à plusieurs requêtes ou réponses selon le principe de développement logiciel DRY [25]. 5

Une bonne pratique est de les utiliser pour logger les requêtes et les réponses de manière systématique ainsi que pour gérer la localisation de nos prompts et reprompts. C'est ce qu'on va faire !



5

Pour l'internationalisation des prompts (bonne pratique pour séparer le code des chaînes de caractères et être prêt à rajouter une autre langue), on va utiliser le framework JS i18next [26] pour récupérer les strings correspondant à la locale de l'utilisateur. Afin d'utiliser cette lib. il faut rajouter la référence à i18next dans le fichier package.json pour qu'elle puisse être importée au moment du déploiement :

```
"dependencies": {
 "i18next": "^10.6.0",
 ...
}
```

Note : *Dans une Skill Alexa-Hosted, pas besoin de lancer* npm install *avant de déployer votre code, cette opération sera réalisée automatiquement par Alexa au moment du déploiement du code sur la fonction AWS Lambda.*

Dans le même répertoire que notre code, l'ensemble des clés-valeurs en français sera stocké dans un nouveau fichier localisation.js. Je vous épargne le fichier complet (dispo sur le GitHub : github.com/nachawat/skill-nodejs-rock-scissor-paper/blob/master/lambda/localisation.js) mais en voici le début :

```
module.exports = {
 fr: {
  translation: {
   SKILL_NAME: 'Pierre Feuille Ciseaux (version Spock)',
   WELCOME_MSG: 'Bienvenue sur le jeu {{skillName}}. Nous avons {{rounds}} ...',
   QUESTION_ACTION_MSG: ['Que choisissez-vous ?', 'Quel est votre choix ?', ...],
   NEW_GAME_MSG: 'Commençons une nouvelle partie. Mon choix est déjà fait ...',
   ROUND_CHOICE: 'J\'ai choisi {{action}} {{connector}} {{explanation}}. ...'
   BUT: ' mais ',
   AND: ' et ',
   ...
  }
 }
}
```

L'intercepteur de localisation a pour objectif de définir une nouvelle fonction dynamique t() sur l'objet handlerInput qui sera utilisée quel que soit le handler ou l'intercepteur suivant appelé. Pour bien séparer les notions, les intercepteurs seront définis dans un nouveau fichier interceptors.js :

```
const localisation = {
  process(handlerInput) {
    const localisationClient = i18n.init({
      lng: handlerInput.requestEnvelope.request.locale,
      resources: languageStrings,
      returnObjects: true
    });
    localisationClient.localise = function localise() {
      const args = arguments;
      const value = i18n.t(...args);
      if (Array.isArray(value)) {
        return value[Math.floor(Math.random() * value.length)];
```

```
      }
      return value;
    };
    handlerInput.t = function translate(...args) {
      return localisationClient.localise(...args);
    }
  }
};
```

N'oubliez pas d'importer les deux modules i18next et localisation dans le fichier interceptors.js :

```
const i18n = require('i18next');
const languageStrings = require('./localisation');
```

On en profite pour rajouter des intercepteurs de logging qui nous seront utiles en cas de débogage :

```
const requestLogging = {
 process(handlerInput) {
 console.log(`Incoming request: ${JSON.stringify(handlerInput.requestEnvelope)}`);
}};
const responseLogging = {
 process(handlerInput, response) {
 console.log(`Outgoing response: ${JSON.stringify(response)}`);
}};
```

Enfin, on rend les 3 intercepteurs accessibles en les exportant dans un module :

```
module.exports = {
  requestLogging,
  responseLogging,
  localisation,
}
```

Tout comme les handlers, il faut référencer les intercepteurs à prendre en compte sur notre instance du SDK. Dans le fichier index.js, on va d'abord importer le nouveau module créé :

```
const interceptors = require('./interceptors');
```

Puis, à la fin du fichier index.js, une fois que vous aurez trouvé le builder Alexa.SkillBuilders.custom() nous allons ajoutez les intercepteurs :

```
Alexa.SkillBuilders.custom()
 .addRequestInterceptors(interceptors.requestLogging, interceptors.localisation)
 .addResponseInterceptors(interceptors.responseLogging)
```

Une fois ces opérations d'ajouts d'intercepteurs terminées, on va écrire des fonctions dédiées à la génération des prompts et reprompts en fonction du contexte de l'utilisateur (accueil, manche en cours, aide, erreur, …). Toujours pour bien séparer les concepts, ces fonctions seront dans un nouveau fichier prompts.js dont le contenu sera copier-coller depuis le repo GitHub de cet article (github.com/nachawat/skill-nodejs-rock-scissor-paper/blob/master/lambda/prompts.js). Pour utiliser les fonctions de ce nouveau module dans nos handlers, on va l'importer dans le fichier index.js :

```
const prompts = require('./prompts);
```

Notez que toutes les fonctions de ce module retournent le même format d'objet contenant seulement le prompt et le reprompt à intégrer. Dans certains cas, le repromptText peut ne pas être présent s'il n'est pas nécessaire de poser une question à l'utilisateur en fonction de son contexte.

```
{
 speechText : '...',
 repromptText : '...'
}
```

Il nous reste juste à remplacer les prompts et reprompts qui sont encore visibles (et en anglais) dans le fichier index.js par les fonctions du nouveau module. Voici la liste des modifications à effectuer :

- Sur la fonction HelpIntentHandler.handle() : utilisez prompts.help()
- Sur la fonction CancelAndStopIntentHandler.handle() : utilisez prompts.goodbye()
- Sur la fonction IntentReflectorHandler.handle() : utilisez prompts.reflector()
- Sur la fonction ErrorHandler.handle() : utilisez prompts.error()

Gérer les attributs de session

Pendant la durée de vie d'une session, il est possible d'utiliser Alexa pour gérer des attributs sous forme de clés/valeurs. A chaque tour de la conversation, votre Skill peut envoyer un objet JSON contenant les attributs qui vous seront renvoyés tels quels au tour suivant. Une fois la session terminée, tous les attributs associés à cette session sont perdus. Si votre Skill souhaite mémoriser des données entre plusieurs sessions, c'est à vous de le persister depuis votre code.

Pour rappel, une session de Skill commence lorsqu'un utilisateur initie l'interaction avec votre Skill en utilisant son nom d'invocation, et se termine lorsque la propriété shouldEndSession du JSON de réponse à la valeur true, ou bien lorsque l'utilisateur ne répond pas à un (re)-prompt. Vous pouvez directement contrôler le statut de la session en utilisant la méthode withShouldEndSession() depuis le ResponseBuilder.

La gestion des attributs de session est prise en charge par l'interface AttributesManager du SDK qui expose les attributs que vous pouvez récupérer (AttributesManager .getSessionAttributes()) et mettre à jour (AttributesManager .setSessionAttributes()) dans les handlers.

Pour la persistance long-terme (en dehors d'une session), le SDK propose deux connecteurs : un pour Amazon S3 (service AWS de stockage objet) - S3PersistenceAdapter et un pour Amazon DynamoDB (service AWS de base de données non-relationnelle) - DynamoDbPersistenceAdapter . Dans les deux cas, la clé primaire utilisée par défaut sera l'identifiant de l'utilisateur fourni par Alexa (UUID anonyme généré à l'activation de la Skill).

Dans notre cas, on va utiliser les attributs de session pour récupérer les scores de l'utilisateur, de l'adversaire et les choix pour chacune des manches du jeu. Les fonctions pour créer, supprimer ou mettre à jour ces attributs sont à rajouter dans un nouveau fichier : session.js

```
function resetGame(attributesManager) {
  const sessionAttributes = attributesManager.getSessionAttributes();
  if (sessionAttributes.game) {
    delete sessionAttributes.game;
  }
  return getOrCreateGame(attributesManager);
}
function getOrCreateGame(attributesManager) {
  const sessionAttributes = attributesManager.getSessionAttributes();
```

```
  if (sessionAttributes.game) {
    return sessionAttributes.game;
  }
  const game = {
    user: {
      score: 0,
      rounds: [],
    },
    alexa: {
      score: 0,
      rounds: []
    }
  };
  sessionAttributes.game = game;
  return game;
}
function updateScores(attributesManager, roundResults) {
  const game = getOrCreateGame(attributesManager);
  // ajout des choix user et Alexa
  game.user.rounds.push(roundResults.userAction);
  game.alexa.rounds.push(roundResults.alexaAction);
  // m.a.j. scores
  switch (roundResults.roundWinner) {
    case 1: game.user.score++; break; // win
    case -1: game.alexa.score++; break; // loose
  }
  return game;
}
/**
 * Module Export
 */
module.exports = {
  resetGame,
  getOrCreateGame,
  updateScores
}
```

Pour s'assurer que pouvoir utiliser les fonctions de ce nouveau module dans nos handlers, on va l'importer dans le fichier index.js :

```
const session = require('./session);
```

Déploiement de la Skill

Nous avons fini le développement de notre Skill et nous pouvons maintenant la déployer !
On commit d'abord les changements sur la branche courante (dev) de notre repo Git :

```
$ git add .
$ git commit -m "add game play"
```

Puis on déploie la Skill avec la ASK CLI qui fera un merge entre les branches dev et master puis un push sur le repository Git distant. Cette opération déploiera le modèle d'interaction ainsi que le code de la branche master sur notre fonction AWS Lambda. Notre Skill est prête pour le test!

```
$ ask deploy
This skill project was cloned from a pre-existing skill. Deploying this project will
Update skill metadata (skill.json)
Update interaction model (models/*.json)
Merge dev branch into master branch
Push dev branch & master branch
Trigger deployment of your skill code?
Do you want to proceed with the above deployments? (Y/n)
```

Note: *ici, on accepte bien sûr de déployer les changements. Il faudra attendre que Alexa-Hosted ait déployé notre code sur la fonction AWS Lambda avant de pouvoir tester. Pour savoir si le déploiement s'est terminé, dans la console Alexa, sur votre Skill, allez dans l'onglet 'Code' pour voir le dernier timestamp de déploiement (visible sous le bouton "Deploy"). Cela ressemble à un texte de la forme* ***"Last Deployed: Jun 11, 2019, 5:46 PM"***.

Testing

Pour illustrer la palette de tests disponibles depuis ASK, j'ai choisi d'utiliser ASK CLI dans cet article, mais notez que les mêmes tests pourront être exécutés depuis SMAPI ou la console Alexa.
La première étape de nos tests sera de vérifier que notre modèle d'interaction réagit correctement en utilisant l'Utterance Profiler. L'objectif est de tester la NLU d'Alexa sans aucun appel à votre code. On utilise la sous-commande ask api nlu-profile en lui donnant une utterance et en retour on obtiendra dans un format JSON l'intention sélectionnée (selectedIntent) mais aussi les intentions considérées mais ignorées (consideredIntents) :

```
$ ask api nlu-profile --locale fr-FR --utterance "Spock"
 --skill-id <votre skill_id>
```

SMAPI : /v1/skills/{skillId}/stages/{stage}/interactionModel/locales/{locale}/profileNlu
Alexa Console : Skill > Build > Utterances Profiler

Dans l'output obtenu, vérifiez bien que selectedIntent contient l'intention GameIntent et que le slot action est rempli avec la valeur spock.
On peut alors passer à l'étape suivante, où l'on va simuler une invocation de notre Skill et récupérer les JSON d'entrée et de sortie traités par notre code. Il est possible de proposer différents inputs au simulateur : soit une expression audio (dispo seulement depuis la console Alexa, une fois le micro autorisé dans mon browser, je pourrais parler à Alexa comme depuis un appareil), soit un texte, ou bien directement en fournissant un JSON pré-formaté.
On va commencer par tester le lancement de notre Skill en fournissant un texte et explorer les documents JSON correspondant au détail de la simulation :

```
$ ask simulate --locale fr-FR --text "ouvre pierre feuille ciseaux"
```

SMAPI : /v2/skills/{skillId}/stages/{stage}/simulations
SMAPI : /v2/skills/{skillId}/stages/{stage}/invocations
Alexa Console : Skill > Test > Alexa Simulator & Manual JSON

On vérifie que la simulation s'est bien passée : la propriété status doit avoir la valeur SUCCESSFUL et les paramètres result.skillExecutionInfo.invocations.invocationRequest et result.skillExecutionInfo.invocations.invocationResponse ne

doivent pas être vides. Point intéressant à noter ici, les intentions considérées mais ignorées sont aussi disponibles dans la simulation sur la propriété result.alexaExecutionInfo.consideredIntents.
On continue nos tests et on va entamer plusieurs sessions complètes d'interactions avec la Skill pour s'assurer de son bon fonctionnement en utilisant la commande ask dialog :

```
$ ask dialog --locale fr-FR
User > ouvre pierre feuille ciseaux
Alexa > Bienvenue sur le jeu Pierre Feuille Ciseaux (version Spock). Nous avons 5 manches pour nous départager. Vous pouvez choisir entre Pierre, Feuille, Ciseaux, Lézard ou Spock pour jouer! J'ai fait mon choix et vous, Quel est votre choix ?
User > Spock
Alexa > J'ai choisi la pierre mais Spock vaporise la pierre. Vous menez 1 à 0. Quel est votre prochain move ?
User >
```

SMAPI : /v2/skills/{skillId}/stages/{stage}/simulations
SMAPI : /v2/skills/{skillId}/stages/{stage}/invocations
Alexa Console : Skill > Test > Alexa Simulator

Les différentes manières de tester présentées ci-dessus permettent de tester la Skill une fois déployée sur votre compte Amazon Developer. Si vous souhaitez tester la partie code en local, vous pouvez utiliser des frameworks de tests dédiés aux applications vocales comme Bespoken [27a] où il vous suffira d'écrire des scénarios de tests en YAML, Besposken se chargera du reste. Voici un exemple de test :

```
- test : Simulation Utilisateur mentionnant Spock
- GameAction action=spock:
 - response.outputSpeech.ssml: "Spock"
 - response.reprompt.outputSpeech.ssml:
  - "prochain move"
  - "prochain choix"
  - "manche suivante"
 - response.shouldEndSession: false
```

Pour exécuter ce scenario de test depuis un terminal, vous allez utiliser la CLI [27b] de Bespoken en lançant la commande bst test pour exécuter vos scénarios :

```
$ bst test
PASS  test/unit/index.test.yml
 fr-FR
  Simulation Utilisateur mentionnant Spock
   ✓GameAction action=spock (551ms)
```

Vous retrouverez sur le GitHub de cet article le fichier test/unit/index.test.yml qui propose des scénarios de tests pour notre Skill. Vous pouvez aussi inclure des tests unitaires (toujours sur la partie code) en utilisant des frameworks de tests classiques en JS comme Mocha [28].
Une fois l'ensemble des tests effectués, je vous recommande de tester sur un appareil Alexa-Enabled pour se mettre en situation réelle et dans les chaussures de l'utilisateur. Si le compte Amazon Developer que vous utilisez est le même que celui utilisé pour configurer vos appareils, alors votre Skill est déjà activée et prête à être testée. On peut maintenant préparer notre Skill à la publication.

Publication

Pour que votre Skill apparaisse sur le Skill Store, elle doit passer par une étape de certification de la part d'Amazon, l'objectif étant de vérifier la cohérence de la Skill. La première étape dans la publication est donc de s'assurer que la Skill est en adéquation avec les tests cases qui seront revus par l'équipe de certification. Lisez bien attentivement les bonnes pratiques disponibles dans la documentation en ligne pour passer une certification la plus douce possible : alexa.design/certification !

Ajoutez les informations de publication

L'étape suivante consiste à renseigner les différentes informations de publication qui apparaîtront sur le Skill Store (Nom, Description, icônes, mots-clés, phrases d'exemples pour utiliser la Skill, pays de distribution, ...). Ces infos sont à renseigner dans le fichier de metadata de la Skill : skill.json (schéma JSON du fichier de manifest dispo en ligne [29]). Vous pouvez aussi utiliser la console Alexa (Skill > Distribution) pour remplir ces différentes données. C'est une étape obligatoire qui est vérifiée par une série de tests automatisés à la fois fonctionnelle et de validation des propriétés du Skill Store avant la publication. Vous aurez accès à un compte-rendu avec la description et le statut de chaque test (assurez-vous pour chaque test que le status est SUCCESSFUL). La commande pour lancer ces tests depuis la ASK CLI est la suivante :

```
$ ask validate --locales fr-FR
```

SMAPI : POST/v1/skills/{skillId}/stages/{stage}/validations
Alexa Console : Skill > Certification > Validation & Functional Tests

Proposez la Skill en beta-test

Après avoir passé la validation automatisée, il est possible de soumettre votre Skill en beta-test jusqu'à 500 utilisateurs pour une période de 90 jours maximum (vous pouvez l'arrêter à tout moment). L'idée est de pouvoir récupérer du feedback utilisateur avant de la soumettre en certification. C'est au développeur de fournir les beta-testeurs via une liste d'adresse emails à uploader sur la console Alexa ou bien via la ASK CLI. Alexa se chargera d'inviter les beta-testeurs par mail à partir du moment où vous activerez le beta-test. C'est une étape optionnelle mais recommandée.

```
# 1- Création d'un beta-test
$ ask api create-beta-test --skill-id <votre skill_id> --feedback-email name@example.com
# 2- Ajout des emails beta-testeurs
$ ask api add-beta-testers --skill-id <votre skill_id> --file listeEmails.csv
# 3- Début du beta-test
$ ask api start-beta-test --skill-id <votre skill_id>
```

SMAPI : /skills/{skillId}/betaTest
SMAPI : /skills/{skillId}/betaTest/testers
Alexa Console : Skill > Distribution > Availability

Soumettre sa Skill en certification

Après toutes ces épreuves, vous êtes prêt à soumettre votre Skill aux équipes de certification Amazon !
A la soumission, vous aurez une estimation du temps de certification et vous pourrez obtenir l'état courant de la certification en

cours. Attention, pendant toute la période de certification, votre Skill sera en lecture seule, aucune modification ne sera possible. Si vous devez absolument faire une modification, vous pouvez retirer votre Skill du processus de certification à tout moment.
Une fois la certification de la Skill terminée, vous serez notifié par mail du résultat de certification. En cas d'échec de certification, vous aurez un compte-rendu mentionnant les différents points à corriger. Bon à savoir, vous pouvez aussi accéder aux comptes-rendus des certifications précédentes.

```
# 1- Soumission de la Skill en certif.
$ ask api submit --skill-id <votre skill_id>
# 2- Retrait de la certif. (si nécessaire)
$ ask api withdraw --skill-id <votre skill_id>
# 3- Récupération l'ID de la dernière certification
$ ask api list-certifications --skill-id <votre skill_id> --max-results 1
# 4- Compte-Rendu de certif
$ ask api get-certification -s <votre skill_id> -c <votre certification_id>
```

SMAPI : /v1/skills/{skillId}/submit
SMAPI : GET /v1/skills/{skillId}/certifications
SMAPI : /v1/skills/{skillId}/withdraw
Alexa Console : Skill > Certification > Submission 6

Si votre Skill est validée en certification, elle apparaîtra sur le Skill Store et sera disponible au grand public. Il ne vous reste plus qu'à vous rendre sur le dashboard d'Analytics de la console Alexa pour voir de quelle manière votre Skill sera utilisée. Restez attentifs aux commentaires et notes laissés par les utilisateurs, cela vous donnera du feedback pour améliorer votre Skill.

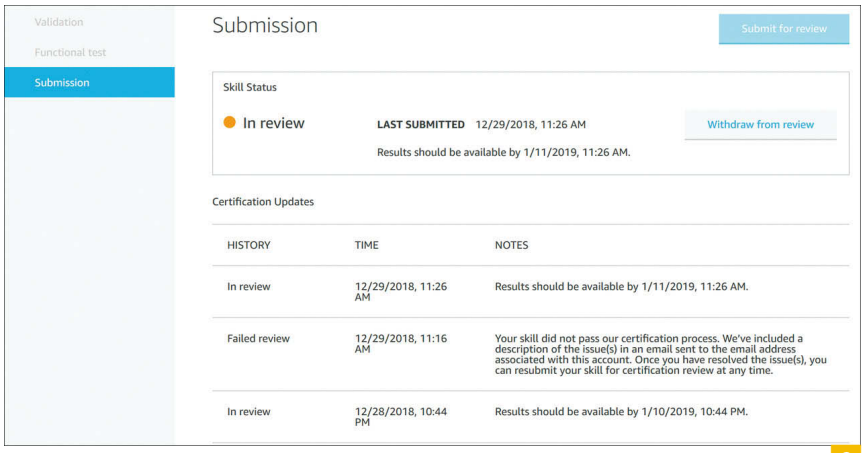
Gérez les hotfix

Vous aurez maintenant deux versions de votre Skill : une version Live et une version In Development. Sur la version Live, il n'est pas possible de modifier ni modèle d'interaction, ni les metadatas. Si vous souhaitez changer l'un des deux, il faut repasser par la case certification. Dans le cas où vous souhaitez faire une modification sur votre code en production (hotfix), sur une Custom Skill Alexa-Hosted, vous pouvez faire un merge sur votre repo Git de la branche master vers la branche prod de cette manière :

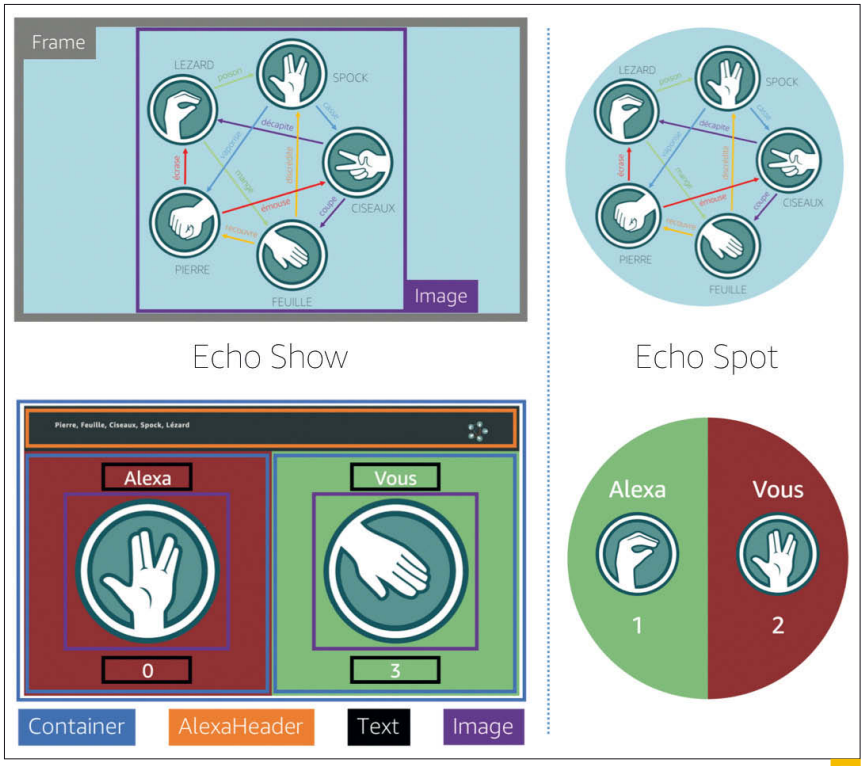
```
$ git add .
$ git commit -m "hotfix"
$ ask deploy --target lambda
$ git checkout prod
$ git merge master
$ git push
```

Rendre la Skill Multimodale

Pour la V2 de notre Skill, on va rajouter la gestion des écrans pour proposer un visuel aux utilisateurs possédant un appareil Alexa-Enabled avec écran (ex. : Echo Show).
Un écran n'est pas présent pour remplacer l'expérience vocale mais pour la compléter en la rendant plus riche et éducative (alexa.design/multimodal), une Skill reste avant tout une expérience vocale !
Pour notre cas d'usage, on va utiliser Alexa Presentation Language



6



7

(alexa.design/apl) pour proposer deux écrans différents : un écran d'accueil montrant les règles du jeu (on l'affichera aussi lorsque l'utilisateur demandera de l'aide, une nouvelle partie ou sur une erreur) ainsi qu'un écran montrant le dernier coup joué par l'utilisateur et son adversaire avec le score de chacun. On utilisera un code couleur (rouge : perdu, vert : gagné, bleu : ex-aequo) pour que l'utilisateur aie un retour visuel rapide sur le gagnant de la manche. Voici les rendus que l'on va inclure dans notre Skill : 7

APL, c'est quoi ?

APL est composée de primitives (container, image, text, scrollview, ...) et de commandes (touch, videoplayer, synchro lecture audio - visuel, ...) permettant de définir un rendu visuel selon votre propre souhait. Pour adapter le contenu en fonction de la taille d'écran, vous avez accès à des clauses conditionnelles pour décider des éléments à inclure. Tout comme en CSS [30], vous avez la possibilité de créer vos propres styles et ressources ou bien d'en utiliser des

styles prédéfinis. Pour vous donner un exemple concret pour mieux comprendre, voici le document APL correspondant à l'écran d'accueil que nous allons ajouter :

```
{
  "type": "APL",
  "version": "1.0",
  "theme": "dark",
  "import": [], // <- Réferencer d'autres documents APL
  "resources": [], // <- Définir des constants
  "styles": {},  // <- Ressources dépendantes d'un état
  "layouts": {}, // <- Composant visuel réutilisable
  "mainTemplate": { // <- mise en page principale
    "parameters": [
      "payload" // <- lien vers la datasource
    ],
    "items": [ // <- tous les composants, ressources, styles, layout utilisés
      {
        "type": "Frame",
        "width": "100vw", // <- vw pour Viewport Width
        "height": "100vh", // <- vh pour Viewport Height
        "backgroundColor": "lightblue",
        "items": [
          {
            "type": "Image",
            "source": "${payload.templateData.image}",
            "width": "95vw",
            "height": "100vh",
            "paddingTop": "5vh",
            "paddingLeft": "10vh"
          }
        ]
      }
    ]
  }
}
```

Il est possible de définir une datasource pouvant être associée à un document APL avec des propriétés provenant de votre code comme l'url d'une image :

```
{
  "templateData": {
    "image": "<URL_IMAGE>"
  }
}
```

Utiliser une datasource avec votre document APL est toujours recommandé pour séparer les données de la partie affichage pur.

Edition d'un document APL

Vous avez à votre disposition un outil d'édition proposé dans la console Alexa (developer.amazon.com/alexa/console/ask/displays) pour écrire un template APL. Cet éditeur vous permettra de construire un écran à partir d'un template existant ou bien de créer le vôtre. La preview du template est immédiate et disponible selon différentes tailles d'écran. Dans ce tutoriel, on va utiliser des documents APL déjà disponibles dans le GitHub de cet article (github.com/nachawat/skill-nodejs-rock-scissor-paper/tree/master/lambda/documents) pour se concentrer sur les étapes nécessaires au rendu de ces templates.

Lorsque vous dessinez vos templates, pensez toujours à adapter le contenu des visuels selon le type d'appareil utilisé. Les informations affichées ne seront pas toutes pertinentes en fonction de la taille de l'écran disponible (écran format radio réveil ? tablette ? tv ?). Pour afficher (ou non), une partie de la mise en page, vous pourrez utiliser les caractéristiques du Viewport avec les clauses conditionnelles **"when"**. Par exemple, sur l'écran montrant les résultats, on ne veut pas montrer le header sur les devices ronds comme sur un Echo Spot. Dans le template gameAction.json, on pourra observer comment utiliser ces clauses :

```
{
 "type": "AlexaHeader",
 "when": "${@viewport.shape != 'round'}",
 "headerTitle": "${payload.templateData.header.title}",
 "headerAttributionImage": "${payload.templateData.header.image}",
 "headerBackgroundColor": "#30393c"
}
```

Gestion des documents APL

Les fichiers JSON de templates et datasources APL sont à inclure dans la réponse de votre Skill soit directement, soit par référence, en utilisant une url HTTP (vous pouvez aussi utiliser un CDN [31]). Dans notre cas, on va inclure le contenu de ces artefacts dans la réponse JSON et on va stocker les documents dans un sous-répertoire de notre Skill : Pierre_Feuille_Ciseaux/lambda/documents/

Chaque datasource sera générée en fonction du contexte de l'utilisateur. Les fonctions pour générer les datasources sont à copier-coller dans un nouveau fichier aplDatasources.js depuis github.com/nachawat/skill-nodejs-rock-scissor-paper/blob/master/lambda/aplDatasources.js.

Cas particulier des images

Pour les images utilisées dans nos visuels, nous avons besoin de les héberger sur un endpoint HTTP public pour qu'elles soient accessibles depuis l'appareil Alexa-Enabled. On va utiliser les capacités offertes par Alexa-Hosted pour stocker et diffuser ces images depuis le bucket Amazon S3 créé pour nous par Alexa.

Pour accéder à cet espace de stockage, il faut se connecter à la console Alexa et aller dans l'onglet 'Code' de notre Skill pour cliquer sur le lien 'Media Storage' disponible en bas à gauche de la page. On sera redirigé sur la console AWS sur la partie ./Media du bucket S3. Une fois sur cette interface web, on va uploader les différentes images disponibles depuis le GitHub de cet article (https://github.com/nachawat/skill-nodejs-rock-scissor-paper/tree/master/images).

Si vous êtes perdu, pas de panique, suivez la documentation expliquant l'upload sur S3 disponible en ligne [32]. Chose importante à savoir : les images déposées sur ce bucket ne sont pas accessibles publiquement. Pour donner un accès externe temporaire, il faudra générer une url présignée [33] depuis notre code en utilisant la fonction getS3PreSignedUrl() fournie dans le fichier util.js.

Activation d'APL

Alexa expose certaines capacités des appareils Alexa-Enabled par des interfaces qui peuvent être activées au sein d'une Skill. Par exemple, Alexa décrit les capacités de streaming audio sur un device par l'interface AudioPlayer (alexa.design/audioplayer). C'est la même chose pour les capacités visuelles. Renvoyer la directive APL depuis la réponse JSON de sa Skill ne suffit pas pour afficher un visuel. Il faut activer l'interface ALEXA_PRESENTATION_APL dans notre Skill. Cette activation peut se faire depuis la console Alexa (dans l'onglet 'Build', cliquez sur 'Interfaces', changer le toggle APL et sauvegardez avec 'Save Interfaces'). Dans notre cas, en utilisant la ASK CLI, on définit cette interface dans le fichier skill.json sur la propriété manifest.apis.custom.interfaces :

```
{
 "manifest": {
  "apis": {
   "custom": {
     "interfaces": [{"type": "ALEXA_PRESENTATION_APL"}],
      ...
   }
  },
  ...
  ...
 }
}
```

Vérifiez la compatibilité du device avec APL

Si l'appareil Alexa-Enabled ne supporte pas l'interface APL, il ne faut pas inclure de directive APL dans la réponse JSON formulée à Alexa. Depuis la requête envoyée par Alexa, la liste des interfaces supportées (context.System.device.supportedInterfaces) et les caractéristiques de l'écran (context.Viewport) de l'appareil Alexa-Enabled utilisé sont accessibles pour nous permettre de tester la compatibilité. Le code gérant la partie APL de notre Skill se trouvera dans un nouveau fichier apl.js. La fonction pour tester la compatibilité de l'appareil à mettre dans ce fichier est la suivante :

```
function supportsAPL(requestEnvelope) {
  const supportedInterfaces = requestEnvelope.context.System.device.supportedInterfaces
  const aplInterface = supportedInterfaces['Alexa.Presentation.APL'];
  return aplInterface !== null && aplInterface !== undefined;
};
```

Ajout des directives APL

Pour afficher un document APL sur la device, la réponse JSON doit inclure la directive Alexa.Presentation.APL.RenderDocument avec le contenu complet du document APL et de sa datasource.

Toujours dans le fichier apl.js, on va rajouter une fonction par écran (les règles du jeu ou bien les résultats du round) pour inclure les éléments nécessaires à chaque visuel seulement si le device est compatible :

```
const aplDatasources = require('./aplDatasources');
function rulesScreen(handlerInput) {
  if (supportsAPL(handlerInput.requestEnvelope)) {
    const document = require('./documents/rules.json');
    const datasources = aplDatasources.getHome();
    handlerInput.responseBuilder.addDirective({
      type: 'Alexa.Presentation.APL.RenderDocument',
      version: '1.0',
      document: document,
      datasources: datasources
    });
  }
};
function gameScreen(handlerInput, roundResults) {
  if (supportsAPL(handlerInput.requestEnvelope)) {
    const document = require('./documents/gameAction.json');
    const datasources = aplDatasources.getGame(handlerInput, roundResults);
    handlerInput.responseBuilder.addDirective({
      type: 'Alexa.Presentation.APL.RenderDocument',
      version: '1.0',
      document: document,
      datasources: datasources
    });
  }
}
module.exports = {
  rulesScreen,
  gameScreen
}
```

La dernière étape consiste à utiliser ces fonctions pour chacune des intentions où l'on souhaite proposer un visuel à l'utilisateur. Dans le fichier index.js, on commence par rajouter l'import du module apl :

```
const apl = require('./apl);
```

Puis, on va identifier les handlers liés à chacun des cas souhaités. L'écran d'accueil sera proposé sur l'invocation de la Skill, une nouvelle partie, le message d'aide. En cas d'erreur, l'écran des scores sera proposé pour chaque round du jeu. Dans les handlers LaunchRequestHandler – NewGameIntentHandler – HelpIntentHandler - ErrorHandler sur la méthode handle(), on va rajouter la ligne de code suivante juste avant l'appel au handlerInput.responseBuilder :

```
apl.rulesScreen(handlerInput)
```

L'écran des scores sera proposé pour chaque round du jeu. Dans le handler GameActionIntentHandler sur la méthode handle(), on va rajouter la ligne de code suivante juste avant l'appel au handlerInput.responseBuilder :

```
apl.gameScreen(handlerInput, roundResults)
```

Le développement de la V2 de notre Skill est terminé.

Testez le rendu des visuels

Il ne nous reste plus qu'à déployer la Skill, jouer les mêmes tests que précédemment (pour s'assurer qu'il n'y ait pas de régressions) puis se rendre sur le simulateur de la console Alexa pour tester les rendus visuels. Pour le déploiement, c'est toujours la même commande avec la ASK CLI :

```
$ ask deploy
```

Le simulateur permet de visualiser le document APL selon différentes tailles d'écrans qui sont classées par profile de viewport : petit rond (équivalent Echo Spot), petit paysage (équivalent Echo Show 5), moyen paysage (équivalent Echo Show), grand paysage (équivalent TV), … N'oubliez pas de tester aussi le cas où il s'agit d'un device sans écran : votre Skill doit toujours fonctionner correctement.

Félicitations pour être allé jusqu'au bout de ce tutoriel, vous venez de créer une Skill complète de jeu multimodal !

Comment monétiser votre Skill ?

Grâce au programme Alexa Developer Rewards [34], chaque mois, Amazon récompense les Skills qui recueillent le plus d'engagement-client. L'engagement des clients est évalué tous les mois. Au fur et à mesure que les clients découvrent et partagent de nouvelles Skills, les Skills les plus engageantes peuvent changer permettant à de nouveaux développeurs de gagner eux aussi de l'argent.

Si vous avez des produits ou services que vous souhaitez rendre disponibles à l'achat sur Alexa comme des billets de train (« *Virgin Trains* » [35] – disponible en Angleterre), des places de cinémas (« *Atom Tickets* » [36] – disponible aux US) ou obtenir des donations (« *Téléthon* » [37] – disponible en France), vous pouvez intégrer Amazon Pay (alexa.design/amazonpay) dans votre Skill. La transaction sera effectuée en utilisant les informations de paiement de l'utilisateur depuis son compte Amazon s'il a activé l'achat par la voix et donné le consentement à votre Skill pour utiliser Amazon Pay. Le flux de paiement sera géré par Amazon Pay. D'un point de vue développeur, vous allez traiter des types de requêtes supplémentaires dans votre code pour communiquer avec Amazon Pay et ajouter des intentions dans votre modèle d'interaction pour gérer la confirmation, l'annulation et le remboursement.

Pour monétiser des contenus digitaux premium et mettre un 'pay-wall' dans votre Skill, vous allez vous tourner vers l'In-Skill-Purchasing (ISP - alexa.design/isp) pour proposer des abonnements (comme la Skill « *Handy Finder* » [38] qui vous aide à retrouver votre téléphone et rajouter jusqu'à trois téléphones avec abonnement – disponible en Allemagne), des achats permanents (comme la Skill « *Deal or No Deal!* » [39] où vous pourrez jouer quotidiennement de manière illimitée après avoir acheté un pack – disponible en Angleterre) ou bien des consommables (comme la Skill de jeu interactif « *Yes Sire!* » [40] qui propose des scénarios ou questions supplémentaires en fonction du gameplay – disponible en Angleterre). La transaction sera aussi réalisée en utilisant les options de paiement du compte Amazon de l'utilisateur. ISP n'est pas encore disponible en France.

Une dernière option est d'utiliser l'Account Linking pour gérer vous-même le flux de transactions avec les informations de paiement que l'utilisateur a choisi d'enregistrer dans votre système. C'est le cas de la Skill « *Domino's* » [41] (disponible en France) qui propose de commander des pizzas parmi l'historique de vos « Commande Express préférées ».

Vers l'infini, et au-delà !

Pour vous tenir toujours informé des nouveautés sur le Alexa Skills Kit, je vous conseille deux très bonnes pratiques : abonnez-vous au flux RSS du blog Alexa (alexa.design/blog) et visitez régulièrement la page des Releases Updates ASK (alexa.design/parity).

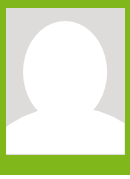
Les équipes Amazon Alexa (alexa.design/contactus) sont aussi là pour vous aider, tout comme la communauté des développeurs (alexa.design/forums), n'hésitez pas à les contacter.

Pour ceux qui n'en n'ont pas eu assez, je vous propose de vous rendre sur les GitHub Alexa (github.com/alexa - github.com/alexa-labs) pour découvrir encore plus d'exemples de Skills.

Et surtout n'oubliez pas : choisissez le vocal quand c'est plus simple, plus naturel et plus rapide pour l'utilisateur ! •

Liens cités

[1] aws.amazon.com
[2] www.boulanger.com/ref/8005231
[3] www.sowee.fr/station-connectee
[4] www.free.fr/freebox/freebox-delta
[5] www.lenovo.com/fr/fr/smart-tab
[6] www.bose.fr/fr_fr/products/headphones/over_ear_headphones/quietcomfort-35-wireless-ii.html
[7] developer.amazon.com
[8] developer.amazon.com/docs/alexa-voice-service/required-hardware.html
[9] aws.amazon.com/lambda
[10] www.amazon.fr/Ouest-France-Ouest-France/dp/B079Z6VRMJ
[11] www.amazon.fr/LEquipe-Le-Flash/dp/B079M1KS6Z
[12] www.amazon.fr/TF1-LCI-LEssentiel-de-lInfo/dp/B07C55CPFQ
[13] fr.wikipedia.org/wiki/OAuth
[14] developer.amazon.com/docs/custom-skills/built-in-intent-library.html
[15] developer.amazon.com/docs/custom-skills/slot-type-reference.html
[16] www.amazon.fr/Monoprix/dp/B07MW3FTBR
[17] www.amazon.fr/Affaires-Sensibles-de-France-Inter/dp/B07QCN1G23
[18] code.visualstudio.com
[19] marketplace.visualstudio.com/items?itemName=ask-toolkit.alexa-skills-kit-toolkit
[20] aws.amazon.com/cloudformation
[21] aws.amazon.com/codepipeline
[22] aws.amazon.com/codestar
[23] aws.amazon.com/codecommit
[24] www.atlassian.com/software/jira
[25] fr.wikipedia.org/wiki/Ne_vous_répétez_pas
[26] www.i18next.com
[27a] github.com/bespoken/bst
[27b] read.bespoken.io/unit-testing/getting-started/#setup
[28] mochajs.org
[29] developer.amazon.com/docs/smapi/skill-manifest.html
[30] fr.wikipedia.org/wiki/Feuilles_de_style_en_cascade
[31] fr.wikipedia.org/wiki/R%C3%A9seau_de_diffusion_de_contenu
[32] docs.aws.amazon.com/fr_fr/AmazonS3/latest/user-guide/upload-objects.html
[33] docs.aws.amazon.com/fr_fr/AmazonS3/latest/dev/ShareObjectPreSignedURL.html
[34] developer.amazon.com/fr/alexa-skills-kit/rewards
[35] https://www.amazon.co.uk/Virgin-Trains/dp/B07798B49K
[36] https://www.amazon.com/Atom-Tickets/dp/B07BMD7M3Z
[37] https://www.amazon.fr/AFM-Telethon-Téléthon/dp/B07L4YNNQB
[38] https://www.amazon.de/Bernhard-%C3%9Cllenberg-Handy-Finder/dp/B06Y5PV98W
[39] https://www.amazon.co.uk/Vocala-co-Deal-Or-No/dp/B07MZ94ZSS
[40] https://www.amazon.co.uk/gp/product/B071ZR5HKR
[41] https://www.amazon.fr/Dominos-Pizza-Enterprises-Limited/dp/B07CBNNYLG



Jean-Bernard Boichat,
Physicien retraité,
auteur chez Eyrolles
jb@boichat.ch
http://www.boichat.ch/wpjrsp/

MicroPython pour le NodeMCU (ESP8266) avec Thonny

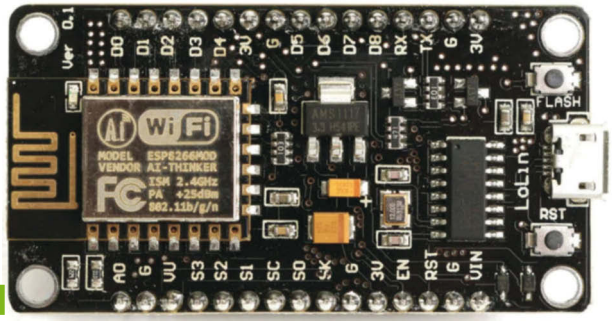
niveau 100/200

La définition dans Wikipedia est parfaite : MicroPython est une implémentation écrite en C du langage Python, adapté au monde des microcontrôleurs. Son site Web officiel se trouve à l'adresse http://micropython.org/, mais https://docs.micropython.org/en/latest/ est plus complet. Thonny est un IDE pour débutant en Python et disponible sous Windows, Mac et Linux. C'est l'outil idéal pour les microcontrôleurs du type ESP8266 ou ESP32 avec un firmware MicroPython que nous allons installer ici pour le premier. Moi-même, sur mes Arduino, ESP8266 et ESP32, je n'ai jamais utilisé le langage de script LUA.

Étant moi-même un fan et un inséparable de programmation Java, donc de programmation objet, je me suis aussi, avec le temps, familiarisé avec Python, le langage que j'ai utilisé pour vérifier mes composants attachés au GPIO du Raspberry Pi 3. Je cherchais aussi quelque chose de ludique simple, orienté éducation pour les débutants, pour mon NodeMCU, un ESP8266. C'est alors que j''ai découvert le MicroPython.

Je ne mentionnerai pas ici la carte piboard (http://micropython.fr/cartes_micropython/pyboard) déjà rencontrée en novembre 2018 dans Programmez. La pyboard possède plus de broches GPIO et analogiques, vient préinstallée avec MicroPython, mais est nettement plus coûteuse. Pour les makers débutants, je recommanderais le NodeMCU avec lequel il est possible de le programmer à la manière Arduino, avec l'IDE de ce dernier, où il y a pléthore d'exemples sur le Web, et avant de passer à l'installation manuelle du firmware MicroPython. Le principe de développement et les outils sont similaires, cependant l'approche ludique de l'apprentissage de Python sur le PC me semble intéressante. De plus une carte PyBoard va bien coûter 10 fois plus cher qu'une solution NodeMCU.

Jouant moi-même depuis longtemps avec l'Arduino, aussi pour vérifier certains capteurs ou composants digitaux dont je disposais sur mes Raspberry Pi, il était naturel de passer au NodeMCU : 1



1

Le modèle présenté et utilisé ici vient du vendeur Ai-Thinker. Si un autre modèle était utilisé, il faudrait consulter le vendeur pour vérifier les différences, voire la version du firmware en cas de réinstallation.

Le NodeMCU est un Arduino simplifié, meilleur marché, possédant une interface Wi-Fi et utilisant le même outil de développement IDE que celui de l'Arduino, donc avec une connexion via un câble USB. Le NodeMCU, en bref, est une carte basée sur le microcontrôleur 8266 avec 10 broches GPIO et une seule analogique (A0).

Lorsqu'on attache au NodeMCU des composants comme des LEDs, des capteurs, voire des relais, l'outil pour les contrôler par logiciel est l'IDE de l'Arduino avec les langages C et C++ (fonctionnalités limitées). Pour les débutants ou enseignants désirant apprendre les rudiments de la programmation, le langage Python est certainement plus approprié.

Un programmeur expérimenté devra aussi avoir des bases du langage C, évidemment, pour ensuite passer à du plus sérieux comme le C++ et améliorer son CV. Personnellement, je pencherais plus pour Java comme premier langage, mais ce dernier est simplement inapplicable sur un Arduino ou un ESP8266. Le langage Python, avec de nombreuses alternatives d'outils, étant disponible sur des plateformes comme Windows, Linux ou encore le Raspberry Pi, ce sera un choix judicieux de remplacer le firmware d'un NodeMCU, donc d'un ESP8266, pour supporter et développer du logiciel en MicroPython, c'est à dire en Python 3. J'indiquerai encore que l'installation et l'utilisation du MicroPython sur un ESP32 est similaire.

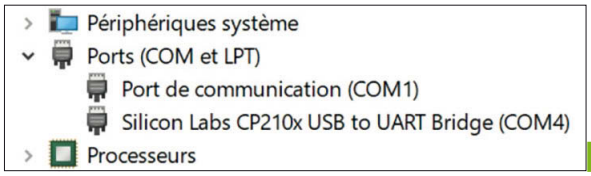
Préparation de l'installation et utilisation des outils

Les procédures décrites ici sont pour Windows, mais c'est assez équivalent sous Linux puisque l'IDE de l'Arduino, le langage Python et d'autres outils comme Thonny existent aussi sous Linux et macOS. Pour pouvoir jouer avec le MicroPython, il faut commencer par remplacer le firmware installé d'usine sur le NodeMCU ESP8266, avant d'utiliser un IDE comme Thonny sur PC.

Le lecteur aura sans doute déjà pianoté avec l'Arduino IDE pour télécharger et tester des sketches. Nous y aurons découvert le COM utilisé ainsi que la dimension de la flash de 4 Moctets (le minimum est de 1 Moctets pour MicroPython, à surveiller, si on utilise d'autres types d'ESP8266).

Lors du branchement du câble USB sur le NodeMCU, il faudra utiliser **Gérer** dans l'explorateur de fichiers de Windows sous « **Ce PC** » pour découvrir quelque chose comme : 2

indiquant que le port COM4 est ici utilisé. Si le port COM n'est pas identifiable, il faudrait envisager d'installer le driver USB depuis Silicon Lab à l'adresse https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers.



2

Installation du firmware

Avant de passer à l'installation de MicroPython, il faut se rendre compte que le firmware original du NodeMCU sera écrasé. Nous ne pourrons donc plus utiliser l'IDE de l'Arduino pour développer des applications avec le firmware original qui sera remplacé par la version MicroPython. Si nous désirons revenir au NodeMCU d'usine, il faudra utiliser par exemple **ESP8266Flasher.exe** pour Windows 64-bit qui se trouve à l'adresse https://github.com/nodemcu/nodemcu-flasher/tree/master/Win64/Release. C'est un exécutable qui ne nécessite aucune modification dans les options : il suffira d'indiquer le COM qui est facilement identifiable après la connexion avec le câble USB.

J'ai fait moi-même la vérification et ensuite un test à partir de l'IDE de l'Arduino avec un sketch rudimentaire contenant juste un **print()** et un **delay()** dans la boucle **loop** traditionnelle. Pour ceux qui veulent faire cet exercice, ils prendront la dernière version binaire de l'IDE sur https://www.arduino.cc/en/Main/Software et il faudra spécifier l'URL de téléchargement https://github.com/esp8266/Arduino/releases/download/2.3.0/package_esp8266com_index.json dans les préférences et ajouter le bon gestionnaire de carte, NodeMCU 1.0 (ESP-12E Module), sous Outils, Type de carte. Pour un autre vendeur d'ESP8266 (ici c'est Ai-Thinker), il faudrait considérer l'outil esptool.py avec *read_flash* pour reprendre le firmware d'origine : il faudra alors donner les bons paramètres.

Pour les makers utilisant Linux ou un autre type d'ESP8266, il faudra se retourner sur l'outil esptool.py et consulter le site https://nodemcu.readthedocs.io/en/master/flash/. Nous y trouverons la description de la commande avec write_flash.

Le téléchargement du firmware se faisant avec un script Python, ainsi que les outils qui suivront, nous aurons besoin d'installer préalablement le langage Python sur le PC. Moi-même j'ai eu quelques difficultés avec mes versions de Python 2 et 3, c'est pourquoi je donnerai ici, à chaque fois, le chemin complet de l'installation lors de l'exécution de scripts.

Pour cette installation, je n'ai pas considéré la version de Python qui vient installée avec Thonny, mais la version standard que j'utilisais déjà avec d'autres outils, comme Eclipse sous PyDev, une extension tiers (http://www.pydev.org/ , et pour mon article dans Programmez de février 2019 (no.226) où j'avais besoin d'une installation traditionnelle.

Nous trouverons sur https://www.python.org/downloads/ le fichier d'installation de Python, la version 3, nommé « Windows x86 executable installer » et nous choisirons l'option « **Customize installation** » avec toutes les « **Features** » pour le positionner par exemple dans le répertoire D:\Python37-32. Il m'est aussi arrivé de devoir faire une réparation (**Repair**), dans le même répertoire, après avoir remarqué que **python.exe** ne voulait plus s'exécuter.

Dans un nouveau répertoire **D:\Python37-32\esptool-master**, nous irons télécharger les outils et les dézipper depuis https://github.com/espressif/esptool (avec le bouton « **Clone or Download** » suivi de « **Download zip** ») ainsi que le firmware de MicroPython, **esp8266-20190125-v1.10.bin**, depuis le site http://micropython.org/download, qui est la version disponible au moment de l'écriture de cet article. Le fichier **esptool-master.zip** n'a pas de date. Attention également de bien prendre la version 8266 du firmware et non celle du 32, c'est à dire de la puce ESP-32.

Pour pouvoir communiquer en Python avec le port série de l'USB, il faudra installer le module **pySerial**. Si nous ne le connaissons pas, pas de soucis. Dans mon cas, dans la console CMD de Windows, en se positionnant sur le disque **D:** entrez les commandes suivantes :

```
D:
D:\>cd Python37-32
D:\Python37-32>python.exe -m pip install pyserial
```

Cela nous indiquera par exemple :

```
D:\Python37-32>python.exe -m pip install pyserial
Requirement already satisfied: pyserial in d:\python37-32\lib\site-packages (3.4)
```

que c'est déjà installé.

Sur certains sites Web, nous pourrions trouver une référence à l'effacement de la flash, voire même en tenant pressé le bouton FLASH sur le NodeMCU :

```
D :
CD D:\Python37-32\esptool-master
D:\Python37-32\python.exe esptool.py --chip esp8266 erase_flash
```

Cette procédure n'a pas été nécessaire sur mon modèle.

Pour l'écriture du firmware, c'est à dire du fichier **esp8266-20190125-v1.10.bin**, certains sites indiquent cette commande :

```
esptool.py --port COM5 write_flash 0x00000 esp8266-20190125-v1.10.bin
```

Dans mon cas, pour ma variante de NodeMCU, j'ai utilisé :

```
esptool.py --port COM5 --baud 115000 write_flash --flash_size=detect -fm dio 0 esp8266
-20190125-v1.10.bin
```

Si notre NodeMCU connecté vient d'être utilisé, il faudra reconnecter le câble USB. J'ai eu un cas, après avoir laissé mes platines de démonstration branchées, de devoir reflasher le firmware en tenant le bouton FLASH pressé durant le transfert.

Donc après avoir identifié le **COM4** depuis Windows, utiliser une vitesse raisonnable (**115000**), voire créer une commande **flash.cmd** avec une PAUSE en dernier :

```
D :
CD D:\Python37-32\esptool-master
D:\Python37-32\esptool-master>D:\Python37-32\python.exe esptool.py --port COM4 --
baud 115000 write_flash --flash_size=detect -fm dio 0 esp8266-20190125-v1.10.bin
esptool.py v2.7-dev
Serial port COM4
Connecting....
Detecting chip type... ESP8266
Chip is ESP8266EX
Features: Wi-Fi
MAC: 5c:cf:7f:00:d6:51
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Flash params set to 0x0240
Compressed 615388 bytes to 399928...
Wrote 615388 bytes (399928 compressed) at 0x00000000 in 35.4 seconds (effective 139.0 kbit/s)...
Hash of data verified.
```

```
Leaving...
Hard resetting via RTS pin...
```

Nous allons à présent indiquer comment vérifier l'installation du firmware.

Vérification de notre NodeMCU avec MicroPython

La première vérification se fera avec **PuTTY** avant d'utiliser **Thonny,** l'outil IDE pour Python et MicroPython.

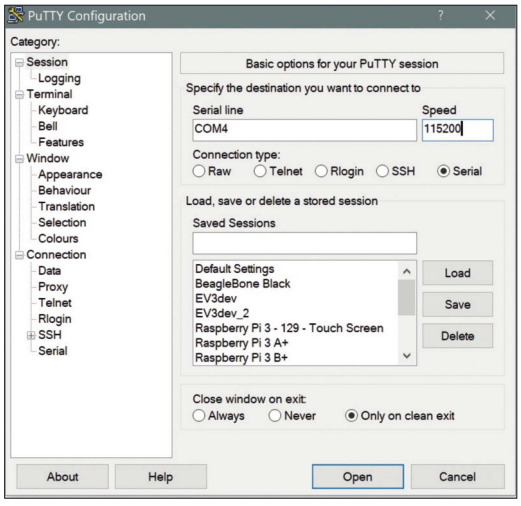
Installation de PuTTY et REPL

Il n'est pas essentiel d'utiliser **PuTTY** pour tester nos premiers scripts. Nous pourrions passer directement à Thonny, l'IDE parfait pour le MicroPython. Pour les bricoleurs comme moi, jouant par exemple avec des Raspberry Pi, un robot Lego EV3 ou d'autres plateformes à microcontrôleur, PuTTY devrait déjà être installé sous Windows. PuTTY est une application client SSH et telnet que l'on peut aussi utiliser sur un port série COM.

Tera Term est par exemple une alternative à PuTTY(https://osdn.net/projects/ttssh2/releases/). Si le lecteur le connaît déjà, il peut l'utiliser sans passer par l'installation de PuTTY. Il ne devra pas oublier de spécifier la vitesse avec le menu **Setup** / **Serial port**… et ceci après la connexion sur le COM du **NodeMCU ESP8266** déjà branché avec le câble USB.

Nous pourrons télécharger PuTTY depuis le site http://www.putty.org/ et nous choisirons la version 64 bits installable avec l'extension **.msi**.

La fenêtre de l'application PuTTY nous permettra d'entrer le port série (bouton à cocher « **Serial** ») et la vitesse de **115200** définie par le firmware : 3



Il est important d'ajuster le **Flow Control** à **None**. Dans cette fenêtre, nous irons sous **Category**, **Connection** et **Serial** pour le modifier. Pour une prochaine utilisation, et c'est vraiment pratique, nous sauverons la session avec un nom prédéfini, comme par exemple NodeMCU COM4.

Avec le bouton « **Open** » nous devrions voir cette fenêtre avec l'invite du MicroPython en ligne de commande **REPL**, c'est à dire « **>>>** ». Elle apparaîtra après l'envoi de la touche **Retour** sur le clavier et positionné dans la fenêtre de PuTTY : 4

REPL signifie Read Evaluate Print Loop. Il s'agit du nom donné à l'invite interactive du MicroPython. Utiliser le **REPL** est de loin le moyen le plus simple de tester du code simple et d'exécuter des commandes. Nous allons commencer par entrer les deux commandes Python suivantes (un Ctrl-C sur une sélection suivi d'un clic du bouton droit de la souris, permet un copier/coller une sélection d'ailleurs dans la fenêtre de PuTTY) :

```
import sys
print(sys.platform+ " " + sys.version)
```

qui seront ainsi interprétés en mode **REPL** :

```
>>> import sys
>>> print(sys.platform+ " " + sys.version)
esp8266 3.4.0
```

La version est bien 3.4.0, celle de notre MicroPython, qui vient d'une réécriture complète de la version 3.4 du langage Python. Pour l'installation et l'utilisation de Thonny qui va suivre, il faudra préalablement fermer la session PuTTY, et ensuite sortir et rebrancher le câble USB.

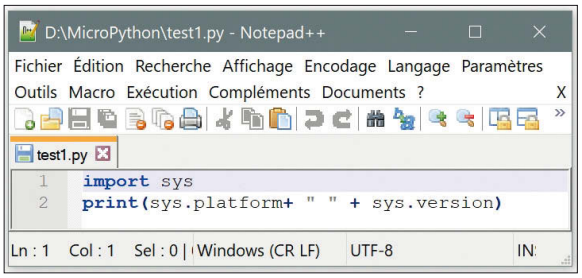
Installation de Thonny

J'ai choisi l'IDE Thonny sous Windows qui permet le développement d'applications en Python et MicroPython depuis le PC et avec un câble USB connecté à notre NodeMCU.

Une alternative à **Thonny**, c'est **uPyCraft**: https://randomnerdtutorials.com/install-upycraft-ide-windows-pc-instructions/. Nous noterons le site https://randomnerdtutorials.com que je conseille au lecteur de suivre régulièrement ou de s'abonner. Rui Santos fait un travail extraordinaire pour de nombreux projets Arduino, ESP8266 ou ESP32. Il a aussi été pour moi une source d'information pour écrire cet article et certains sujets de mon dernier livre dédié au Raspberry Pi 3 avec Python et Java.

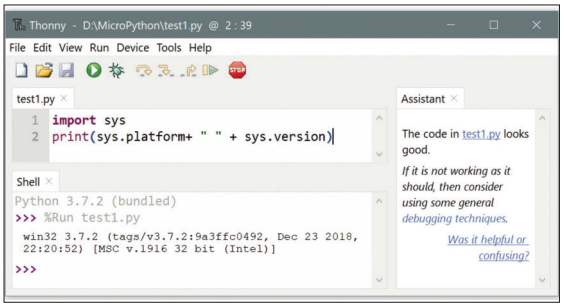
Sur le site Web de Thonny, https://thonny.org/, au moment de l'écriture de cet article, j'y ai trouvé la version 3.1.2 (**thonny-3.1.2.exe**). Comme j'avais déjà installé Thonny, il me demande juste de mettre mon ancienne version à jour sur D:\Thonny (j'utilise le disque D : pour ne pas surcharger mon disque SSD C:).

Nous commencerons par créer un fichier **test1.py** avec les deux instructions que nous connaissons déjà : 5

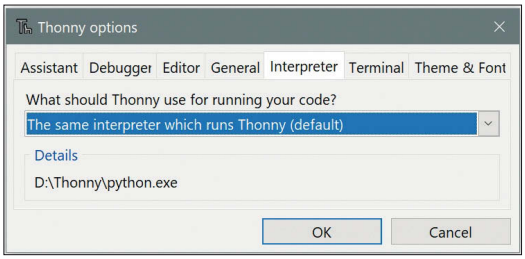


Je l'ai mis ici volontairement dans Notepad++ (https://notepad-plus-plus.org/fr/), car j'adore sa colorisation syntaxique et je l'utilise souvent avant de déposer mes scripts Python sur mes Raspberry Pi, scripts qui ne nécessitent pas souvent de tests sur le PC au travers d'un interpréteur Python à installer.

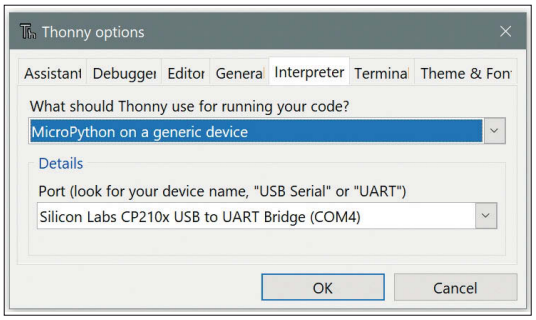
Ayant créé ce premier script Python, nous pouvons à présent l'ouvrir dans Thonny et l'exécuter avec le bouton fléché circulaire **Run**, coloré vert (ou F5) : 6



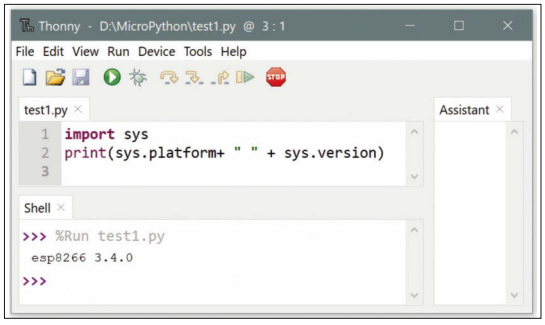
Le premier message indique que l'interpréteur est celui empaqueté avec Thonny. Nous pourrons l'identifier avec le menu **Tools**, **Options** et **Interpreter** : 7



Nous passerons rapidement à notre NodeMCU ESP8266, qui est déjà connecté avec un câble USB, en changeant le mode en « **generic device** » avec le même menu **Tools**, **Options** et **Interpreter** : 8



Nous remarquons que le **COM4** a été assigné par Windows. Comme ci-dessus, nous utiliserons à nouveau le bouton **Run** coloré vert (ou F5) : 9



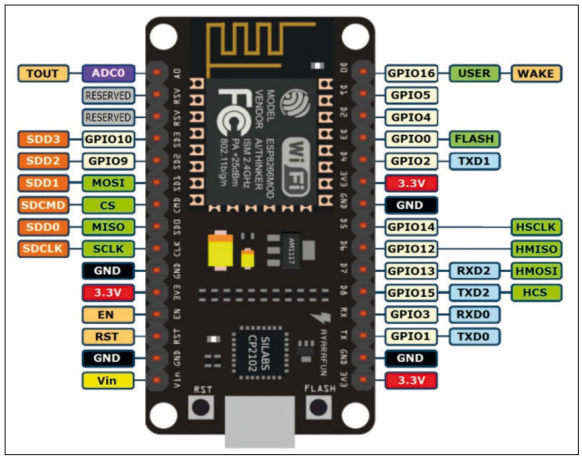
Nous sommes cette fois-ci sur notre NodeMCU ESP8366. Nous avons la même réponse que précédemment en mode **REPL** sous **PuTTY**. Magnifique !

Déposer des composants sur notre NodeMCU

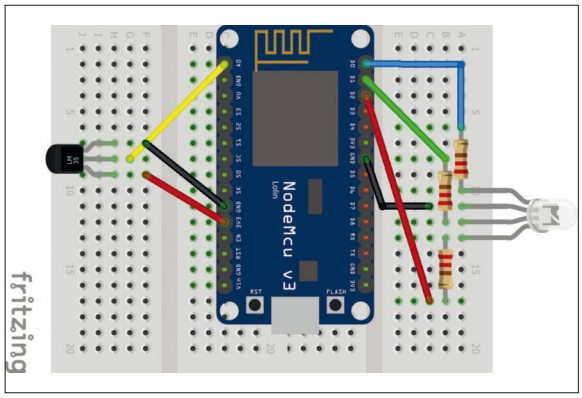
Avant de pouvoir programmer nos premiers scripts Python, il nous faut préparer nos plaques de prototypage avec quelques capteurs et composants.

Une LED à 4 branches et un LM35

Une LED à 4 branches, c'est une LED couleur où les trois couleurs RVB peuvent être activées séparément. Ce sera un excellent sujet pour de premiers exercices en Python. Nous devons tout d'abord présenter la disposition des pins sur le NodeMCU : 10



Regardons également le diagramme **Fritzing** (http://fritzing.org/home/) de notre LED où chaque broche, correspondant à une couleur RGB différente, doit être protégée, pour limiter le courant, par une résistance de 220 Ohm. Valeur que nous trouvons dans les kits et correcte pour cette tension de 3.3Volt et ce composant LED : 11



Nous y avons aussi ajouté un capteur de température **LM35** sur l'unique broche analogique du NodeMCU. En plaçant le NodeMCU juste entre deux petites plaques de prototypage, il tiendra bien, et nous aurons d'un côté la partie analogique et de l'autre la digitale. Pour la couleur des fils de la LED RGB, nous avons utilisé ici sur ce schéma la couleur qui sera activée sur la LED. La broche D0, en haut à droite, va correspondre au GPIO16, c'est à dire à la couleur rouge. J'ai ajouté ces descriptions de couleurs dans les scripts Python qui suivront. Nous mettrons en série trois

résistances de 220 Ohm avant chaque broche correspondante à une couleur. La broche la plus longue est celle associée à la terre.

Test de nos deux composants depuis Thonny

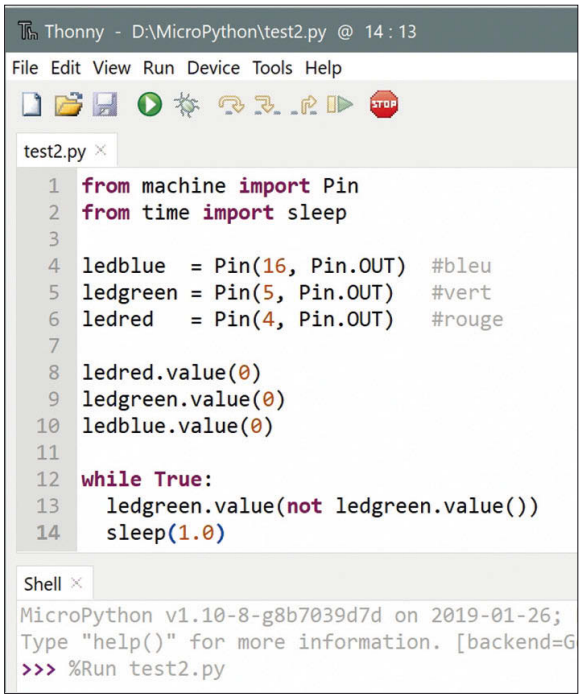
Nous commencerons par tester les deux composants depuis Thonny avec deux scripts des plus dépouillés, mais aussi lisibles que possible.

Le plus facile et direct ici est de démarrer Thonny avec le NodeMCU déjà connecté et avec le mode en « **generic device** » (**Menu** : **Tools**, **Option**, **Interpreter**) sur le COM déjà détecté.

Dans le fichier **test2.py** nous déposerons le script Python suivant :

```
from machine import Pin
from time import sleep

ledblue  = Pin(16, Pin.OUT)  #bleu
ledgreen = Pin(5, Pin.OUT)   #vert
ledred   = Pin(4, Pin.OUT)   #rouge

ledred.value(0)
ledgreen.value(0)
ledblue.value(0)

while True:
 ledgreen.value(not ledgreen.value())
 sleep(1.0)
```

Nous montrons à nouveau l'exécution dans Thonny avec le bouton **Run** coloré vert (ou **F5**) : 12



Ici, il n'y a pas de message dans la partie Shell de Thonny, car aucune instruction **print()** n'est présente dans le script.

Après l'assignation des broches (voir le diagramme ci-dessus, où par exemple la broche D0 correspond au GPIO16 et la couleur bleu) nous déposons 3 valeurs 0 sur chacune. Aucune couleur ne sera donc activée.

Dans la boucle **while** le **not** de l'instruction **value()** est particulier et génial. Chaque seconde, la valeur de la couleur rouge sera inversée. Donc, au premier passage, la LED sera allumée en rouge pour 1 seconde. Avant la boucle **while** nous pourrions activer la partie bleue et la laisser ainsi, donc avec **ledblue.value(1)**. Lorsque la broche rouge sera inactive, chaque seconde après une seconde, la LED sera bleue, sinon le bleu et le rouge se mélangeront.

En ajoutant le code qui suit qui contient quelques pauses en secondes (**sleep(sec)**), et avant les trois **value(0)**, et comme ceci :

```
sleep(1.0)
ledblue.value(0)
sleep(2.0)
ledboard = Pin(2, Pin.OUT)
ledboard.value(0)
sleep(3.0)
```

Nous comprendrons que (nous retirerons et remettrons le câble USB pour s'assurer du bon état des broches au démarrage) :

- la LED couleur sera bleue, car le GPIO16 sera 1 au démarrage par défaut ;
- après une seconde la LED s'éteint, mais une LED bleue sur le NodeMCU s'allume (près du câble USB) (elle est mise en circuit inversé par rapport à D0, le GPIO 16) ;
- après 2 autres secondes, une autre LED bleue sur l'ESP8266, à côté du D0, associée au GPIO02 (D4) s'allume (**ledboard.value(0)** : valeur inversée comme D0. Par défaut au démarrage le D4 (pas utilisé ici) est à 1 ;
- lorsque le code initial est actif avec la LED clignotant en rouge, les deux LEDs bleues sur la carte resteront allumées.

Il est donc possible d'utiliser les deux LEDS de la carte pour indiquer à l'utilisateur des états d'activité du logiciel. Ce dernier devrait donc éviter d'utiliser D0 et D4 pour des composants intégrés sur les platines de développement.

La manipulation du bouton **STOP** dans Thonny, pour arrêter l'exécution n'est pas évidente, comme d'ailleurs le **Save** (**Ctrl-S**) dans le menu qu'il faut parfois cliquer deux fois pour fonctionner après des corrections de code. Suivant les situations, il faudra parfois retirer le câble et le remettre, voire relancer Thonny. Par contre charger un autre script et l'exécuter sans stopper l'actuel semble bien fonctionner. Nous pouvons facilement nous imaginer le nombre d'exercices possibles en Python, voire avec la librairie PWM (modulation de largeur d'impulsion) comme ici :

```
from machine import Pin, PWM
import time

from machine import Pin
from time import sleep

ledblue  = Pin(16, Pin.OUT)  #bleu
ledgreen = Pin(5, Pin.OUT)   #vert
ledred   = Pin(4, Pin.OUT)   #rouge
ledred.value(0)
ledgreen.value(0)
ledblue.value(0)
```

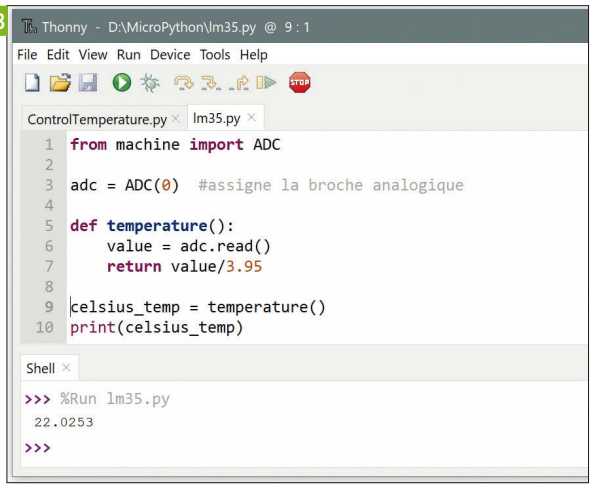
```
pwm = PWM(Pin(4))
for level in range(0, 1023):
  time.sleep(.01)
  pwm.duty(level)
```

Le PWM n'est pas disponible sur la broche D0 (GPIO 16) : il faudrait alors faire passer le bleu de la broche D0 à la broche D3 (GPIO 0) par exemple. Cet exemple nous montre notre LED couleur mise au rouge (1013 pour le maximum) qui se fane rapidement jusqu'à s'éteindre (pwm.duty(0)).
Comme pour le script **test1.py** et les suivants, tous ces scripts ne sont pas téléchargés dans le NodeMCU et il faudra les exécuter à chaque fois. Mais cela permet de les vérifier, d'y ajouter plus de code, d'autres fonctions ou modules, avant une installation pour un démarrage automatique au boot de l'ESP8266, c'est à dire dans le fichier **main.py**. Nous y viendrons rapidement !
Le script ci-dessus sera vraisemblablement déposé dans un fichier et chargé dans Thonny. Mais un copier/coller dans la fenêtre **Shell** de ce dernier est tout à fait possible : le code devrait cependant rester simple et une exécution ligne par ligne est recommandée.
Le prochain composant à vérifier, c'est le capteur de température, un LM35.

```
from machine import ADC

adc = ADC(0)  #assigne la broche analogique

def temperature():
  value = adc.read()
  return value/3.95

celsius_temp = temperature()
print(celsius_temp)
```

Le **def** n'est pas vraiment nécessaire, peut-être juste pour montrer au lecteur que l'auteur de cet article sait un petit peu comment écrire une fonction en Python !
Le 3.95 a été défini, après calibration, en déposant un thermomètre à côté de notre platine de prototypage et en vérifiant le résultat. Avec deux doigts sur le LM35, nous pouvons rapidement vérifier que la température augmente ou diminue, mais évidemment pas déterminer si nous sommes fiévreux ! 13
Le 22 degrés est une température juste confortable pour travailler et les chiffres après la virgule pourraient être ignorés, ou mieux, arrondis à une décimale avec la fonction **round(celsius_temp, 1)**. J'ajouterai encore qu'en déconnectant la platine de démonstration contenant le LM35, j'ai noté une valeur de 2 à la lecture. Un LM35 mal branché pourra aussi être vérifié en le pinçant de deux doigts pour y faire monter la température.

13



Un script plus sérieux et déposé dans le NodeMCU

Nous allons à présent considérer les aspects suivants :

- intégrer le code des scripts précédents, **test2.py** et **lm35.py** pour y faire quelque chose de plus intelligent ;
- ajouter le code nécessaire pour attribuer une adresse IP à notre NodeMCU ;
- y ajouter un serveur Web pour obtenir la température de l'extérieur.

Les trois composants seront vérifiés comme précédemment, et toujours sans télécharger les scripts pour qu'ils soient disponibles au prochain démarrage du NodeMCU.

Trois couleurs différentes suivant la température

Nous présentons le script **ControlTemperature.py**, que nous préparerons sous Thonny, suivi de quelques explications :

```
from machine import Pin
from machine import ADC
from time import sleep

ledblue  = Pin(16, Pin.OUT)  #bleu
ledgreen = Pin(5, Pin.OUT)   #vert
ledred   = Pin(4, Pin.OUT)   #rouge

adc = ADC(0)  #LM35 sur l'analogique 0

ledgreen.value(0)
ledblue.value(0)
ledblue.value(1)
sleep(.3)
ledblue.value(0)
ledgreen.value(1)
sleep(.3)
ledgreen.value(0)
ledred.value(1)
sleep(.3)
ledred.value(0)
sleep(1)

while True:
 reading = adc.read()
 celsius_temp = round(reading/3.95, 1)
 print(celsius_temp)

 ledgreen.value(0)
 ledblue.value(0)
```

```
ledred.value(0)

if (celsius_temp) > 24.0:
 ledred.value(1)
else:
 if (celsius_temp) < 22.0:
  ledblue.value(1)
 else:
  ledgreen.value(1)

sleep(2.0)
```

Rien de bien sorcier. C'est une composition des scripts précédents avec la lecture de la température. Chaque deux secondes nous coupons la tension GPIO de toutes les broches, pour ne pas mélanger les couleurs, et nous attribuons une couleur dépendante de la température :

- bleu si en dessous de 22 degrés ;
- rouge si supérieure à 24 degrés ;
- vert, entre les deux, c'est confortable.

Comme il n'y a ici qu'un seul **print()**, c'est la seule chose que nous verrons dans le shell de Thonny en l'exécutant, c'est à dire la température en degrés Celsius avec une décimale. Nous verrons la LED avec la couleur correcte, couleur qu'il nous est facile de faire passer au rouge avec deux doigts sur le LM35. La mise à jour se fera chaque seconde, c'est suffisant.

Avant de passer au transfert de fichiers MicroPython sur le NodeMCU et en particulier aux **boot.py** et **main.py** qui seront exécutés à chaque démarrage, il nous faut revenir un peu au mode REPL et à la configuration Wi-Fi.

Configuration Wi-Fi

Notre application finale, qui aura besoin d'un accès réseau pour transmettre en particulier la température récupérée par un LM35, va nécessiter préalablement la mise en place de sa configuration réseau.

Cette configuration et sa vérification peut très bien être faite du **Shell** de Thonny voire aussi de **PuTTY**. Nous noterons ici que le Shell de Thonny permet de copier/coller (Ctrl-C/Ctrl-V) depuis une autre fenêtre Windows et aussi de naviguer sur d'anciennes commandes avec les touches curseur.

```
import network
sta_if = network.WLAN(network.STA_IF)
sta_if.active(True)
print('network config:', sta_if.ifconfig())
```

Il faudra un peu patienter avant de voir la réponse avec l'adresse IP activée (entretemps un 0.0.0.0 serait retourné) :

```
>>> print('network config:', sta_if.ifconfig())
network config: ('192.168.1.138', '255.255.255.0', '192.168.1.1', '192.168.1.1')
```

Dans mon cas, j'ai attribué, dans mon routeur Wi-Fi, 192.168.1.138 comme adresse IP statique.

Il serait ensuite possible avec **sta_if.active(False)** de désactiver l'adresse IP et de, par exemple, nous déplacer dans un autre réseau Wi-Fi ou routeur, en refaisant la même procédure.

Notre application main.py

Thonny nous permet de télécharger des fichiers et des scripts Python sur notre NodeMCU installé avec MicroPython. Ceci se fait avec le menu **Device** et un des trois sous-menus **Upload**. Nous commencerons par le premier script décrit ici, le fichier **test1.py** et ensuite avec **main.py** et **boot.py**.

Le **main.py** est le fichier qui contiendra le code de notre application et qui sera lancé lors de la mise sous tension, s'il existe, et après le script **boot.py**. Nous déposerons dans ce dernier le code pour initialiser le réseau Wi-Fi.

Nous allons à présent examiner comment accéder au système de fichiers du NodeMCU voire de le modifier. Ayant déposé, avec le menu correspondant de Thonny, le fichier **test1.py** sur notre NodeMCU, nous pourrons avec le Shell de Python, exécuter les commandes suivantes :

```
import os
os.listdir()
print(open('test1.py').read())
os.remove('test1.py')
os.listdir()
```

Typiquement nous devrions retrouver pour la dernière instruction, et plus tard après l'installation des **main.py** et **boot.py** qui vont suivre, le résultat de la dernière commande :

```
>>> os.listdir()
['boot.py', 'main.py']
```

Précédemment, le **read()** nous aura montré le contenu de **test1.py**. La commande **remove()** effacera le **test1.py**.

Notre application **main.py** présentée maintenant est composée des fonctions discutées précédemment :

- test de notre LED ;
- lecture de la température ;
- couleur de la LED en fonction de cette dernière.

Et nous y ajoutons un serveur Web pour obtenir cette température de l'extérieur.

```
from machine import Pin
from machine import ADC
from time import sleep
import socket

#import machine
html = """<!DOCTYPE html>
<html>
  <head><title>Temperature</title></head>
  <body>
   %s
  </body>
</html>
"""

celsius_temp = 10.0

ledblue  = Pin(16, Pin.OUT)  #bleu
ledgreen = Pin(5, Pin.OUT)   #vert
```

```
ledred  = Pin(4, Pin.OUT)  #rouge

adc = ADC(0)  #LM35 sur l'analogique 0

ledgreen.value(0)
ledblue.value(0)
ledblue.value(1)
sleep(.3)
ledblue.value(0)
ledgreen.value(1)
sleep(.3)
ledgreen.value(0)
ledred.value(1)
sleep(.3)
ledred.value(0)
sleep(1)

addr = socket.getaddrinfo('0.0.0.0', 80)[0][-1]
s = socket.socket()
s.bind(addr)
s.listen(1)

while True:
 #print('Boucle ')
 s.settimeout(2)
 try:
  conn, addr = s.accept()
 except OSError as er:
  pass
  #print(er.args[0] in (110, 'timed out'))
  # 110 is ETIMEDOUT
 else:
  #print('Client connecté de ', addr)
  conn_file = conn.makefile('rwb', 0)
  while True:
    line = conn_file.readline()
    #print('Line ', line)
    if not line or line == b'\r\n':
      break

  response = html % celsius_temp + '\n'
  #print('Client receives ', response)
  conn.send('HTTP/1.1 200 OK\n')
  conn.send('Content-Type: text/html\n')
  conn.send('Connection: close\n\n')
  conn.sendall(response)
  conn.close()
  conn.close()

 reading = adc.read()
 celsius_temp = round(reading/3.95, 1)
 #print(celsius_temp)

 ledgreen.value(0)
 ledblue.value(0)
 ledred.value(0)

 if (celsius_temp) > 24.0:
  ledred.value(1)
 else:
  if (celsius_temp) < 22.0:
   ledblue.value(1)
  else:
   ledgreen.value(1)
```

Comme pour les scripts précédents, nous vérifierons le script de la même manière, depuis Thonny, et avant de le télécharger. Un certain nombre de **print()**, ici en commentaire, nous permettraient de vérifier le bon fonctionnement du script.
La variable **html** est typique d'une page Web vraiment simple, juste pour nous retourner la température où le **%s** sera remplacé par la valeur de **celsius_temp** qui sera de 10.0 degrés au démarrage et immédiatement corrigée après 2 secondes dans la boucle **while True**. Les deux secondes viennent du **s.settimeout(2)** sur le socket du port **80** (Web).
Le MicroPython de l'ESP8266 n'ayant pas de support Thread, c'est une manière de faire un peu simpliste, mais suffisante, pour cette application. Une requête http://192.168.1.138/ devra patienter jusqu'à 2 secondes avant de recevoir la dernière température. Au timeout la température sera mise à jour comme la couleur de la LED.
L'adresse IP 0.0.0.0 est une adresse réservée indiquant ce site et ce réseau. Avec 0.0.0.0 nous écouterons sur tous les réseaux d'interface configurés.
L'instruction **s.accept()** est assez particulière pour les débutants en Python, car elle nous retourne une paire de valeurs, **conn** et **adress**, un tuple : le premier est la référence d'un nouvel objet de la classe **socket()**, qui sera la véritable interface de communication entre le client et le serveur, et le second un autre tuple contenant les coordonnées de ce client (son adresse IP et son numéro de port utilisé).
Dès qu'un client sur le port 80 demande un accès, nous lisons toute la requête en ignorant ici les paramètres. Si nous jouons, avec par exemple http://192.168.1.138/temperature, nous recevrons toujours la même réponse envoyée avec les différents **conn.send()** qui spécifient le protocole et la réponse. Le **conn.sendall()** est identique à **conn.send()** mais attend que tout soit envoyé. La **send('Connection: close\n\n')** est intéressante : elle indiquera que la connexion n'est pas persistante après l'envoi de la réponse : notre température « emballée » dans des balises HTML.
Il y aurait du travail pour implémenter le passage dans une requête Web pour nos deux limites de température, ici fixées à 22 et 24 degrés, pour les intégrer dans le script **main.py** voire stockées dans un fichier de configuration de l'ESP8266.

Notre script de démarrage boot.py

Nous déposerons le code réseau pour le Wi-Fi dans le fichier Python **boot.py** :

```
def do_connect():
 sta_if = network.WLAN(network.STA_IF)
 if not sta_if.isconnected():
  print('connecting to network...')
  sta_if.active(True)
```

```
  sta_if.connect('routeur', 'mot_de_passe')
  while not sta_if.isconnected():
    pass
 print('network config:', sta_if.ifconfig())

do_connect()
```

Nous le vérifierons tout d'abord avec le shell :

```
>>> %Run boot.py
network config: ('192.168.1.138', '255.255.255.0', '192.168.1.1', '192.168.1.1')
```

La fonction **do_connect()** devra être modifiée en y mettant le nom et le mot de passe du routeur. Comme je le fais pour mes composants IOT, après l'attribution de l'adresse IP, ici ce **192.168.1.138**, j'utilise l'accès Web du routeur pour y attribuer le mode statique. Ce sera plus facile pour retrouver ce NodeMCU parmi une pléthore d'objets connectés.
La variable **sta_if** est utilisée avec la classe WLAN pour obtenir une connexion au routeur et en passant à la fonction **connect()** le SSID (le nom du réseau, le routeur) et son mot de passe.
Ensuite nous utiliserons le menu **Device** avec « **Upload current script as boot script** ». Le menu **Device** possède deux sous-menus pour visionner nos deux **boot.py** et **main.py**. Nous débrancherons et reconnecterons le câble USB pour vérifier l'installation, que la LED change de couleur en la touchant de deux doigts, et que la requête http://192.168.1.138/ nous retourne des valeurs différentes.
En cas de difficultés, nous pourrions toujours remettre quelques **print()** dans le code, voire utiliser PuTTY. Pour utiliser ce dernier, il faudra rebrancher le câble pour avoir une session correcte dans PuTTY. Un Ctrl-C dans sa console permettrait d'interrompre le script **main.py**, avec ces conséquences, et de retravailler en mode **REPL**. Le lecteur comprendra vite que notre Thonny est bien agréable à utiliser.

Enfin un peu de Java

Ma passion et mon attachement restant toujours la programmation Java, je n'ai pas hésité à écrire une petite classe Java pour récupérer la température de cet objet connecté qu'est notre NodeMCU :

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;

public class NodeMcuWebRead {
 public static void main(String[] args) throws IOException {
  URL url = new URL("http://192.168.1.138");
  BufferedReader in = new BufferedReader(
  new InputStreamReader(url.openStream()));

  String inputLine;
  float temperature = (float)0.0;
  int lineNb = 0;
  while ((inputLine = in.readLine()) != null) {
                if (lineNb == 4) {
                 inputLine = inputLine.replaceAll("\\s+","");
                 temperature = Float.parseFloat(inputLine);
                 System.out.println(temperature);
                 break;
                }
                lineNb++;
  }
  in.close();
 }
}
```

Ce code pourra être compilé et déposé sur n'importe quel PC ou système embarqué, comme un Raspberry Pi (avec Java 8), voire un NAS et là où une machine virtuelle Java est disponible et installée. En utilisant un explorateur Internet à l'adresse http://192.168.1.138/ et en examinant le code source (bouton droit de la souris sous Chrome et Firefox) nous retrouverons notre température sur la 5e ligne du code HTML. Nous retirerons les espaces avant une conversion en **float**.
Nous pourrions par exemple étendre cette mini-classe en une classe lisant régulièrement la température pour :

- enregistrer la température chaque minute dans une table de base de données SQLite, aussi créée avec sa table dans cette même classe ;
- calculer les moyennes horaires et journalières et les déposer dans deux autres tables SQLite ;
- y ajouter d'autres NodeMCU pour des statistiques de températures mesurées à différents endroits de la maison, voire de l'extérieur ;
- développer un Web serveur en Java pour montrer ces informations sous différentes formes ;
- envoyer chaque jour ou semaine un courriel avec ces informations de statistique ;
- créer un système d'alarme pour indiquer un problème de température lié aux limites de température précédemment discutées et en permettant au NodeMCU d'accepter de nouvelles valeurs limites.

C'est parti pour l'apprentissage de Python

Ce n'est pas le but de cet article, mais nous voyons bien que nous avons ici une infrastructure idéale pour l'apprentissage du langage Python.
Nous pourrions commencer par des variables, qui pourraient par exemple représenter l'intensité d'une ou de plusieurs couleurs de la LED. Des tests de conditions suivant la valeur de la température, ou de petits jeux de variations d'intensité ou de couleurs, sont faciles à imaginer et à développer sous Thonny.
En fonction des thèmes de l'apprentissage de Python, il serait alors facile de mettre en place une série d'exercices évolutifs, comme par exemple, l'implémentation, qui me semble incontournable, d'une classe Python, LedColor, contenant toutes les fonctionnalités sorties de notre imagination.

Apprendre Java et Python en parallèle

Dès que nous parlons de classe, ici de LedColor, il nous viendrait vite à l'esprit d'écrire aussi une jolie classe en Java, sous Eclipse, pour un Raspberry Pi, avec la même LED, mais avec un Dallas DS18B20 digital pour remplacer notre LM35 analogique, car le Pi n'a pas d'interface analogique. •

Fabien LOISON
Développeur / chef de projet chez **Wanadev.** *Passionné par les technologies anciennes et moins anciennes. Bidouilleur et blogueur à ses heures perdues.*
Blog : https://blog.flozz.fr/

Introduction au développement Game Boy

Développer un jeu Game Boy en 2019, c'est possible, mais malgré une communauté qui commence à s'organiser, les ressources restent encore assez éparpillées. Cet article est une introduction au développement Game Boy et a pour but de vous aider à vous lancer dans de bonnes conditions.

niveau 200

La Game Boy a fêté ses 30 ans cette année, et malgré son âge, de nombreux développeurs continuent à s'y intéresser. Ces dernières années, la *homebrew scene* a même été plutôt active autour de cette console, et on a pu voir émerger une poignée de jeux, dont certains peuvent même être trouvés en version physique, sous forme de cartouches. Si je devais n'en retenir qu'un seul, je nommerais *Sheep It Up!*, qui a relancé mon intérêt pour le développement sur la protable de Nintendo, mais je peux également citer *Tobu Tobu Girl* ou *Dangan GB*.

Aujourd'hui, développer un jeu Game Boy est à la portée de toute personne ayant quelques connaissances en C et un poil de motivation. Il faudra certes faire face à quelques contraintes techniques, mais programmer un petit jeu est malgré tout étonnamment simple !

Commençons par faire le tour des principales caractéristiques de la console. La Game Boy dispose :

- d'un écran de **160×144 pixels** capable d'afficher **4 niveaux de gris**,
- d'un processeur cadencé à **4,19 MHz** (environ),
- de **8 ko** de mémoire vive (RAM),
- de **8 ko** de mémoire vidéo (VRAM).
- d'un générateur de son à 4 canaux,
- et d'un port série (câble Link).

Les jeux de la Game Boy sont distribués sous forme de cartouches qui sont composées :

- d'une ROM (mémoire non réinscriptible) dont la capacité varie de **32 ko** à **4 Mo** en fonction des jeux,
- et optionnellement d'une mémoire de sauvegarde (SRAM) afin d'y conserver la progression du joueur.

Les cartouches de plus de 32 ko embarquent également un contrôleur de banque de mémoire (*Memory Bank Controller*), et certaines cartouches peuvent contenir du matériel plus exotique, comme une caméra ou un vibreur, mais on va s'en tenir pour le moment aux cartouches plus standards.

Contrairement aux développeurs de l'époque qui avaient besoin de matériel spécifique pour développer et tester leurs jeux, il suffit aujourd'hui d'une poignée de logiciels pour tout faire. Pour développer un programme pour la Game Boy, nous aurons ainsi besoin :

- de **SDCC** : la chaîne de compilation (compilateur, assembleur, éditeur de liens,...) pour compiler nos programmes et générer nos ROMs,
- de **gbdk-n** : ensemble de bibliothèques permettant de développer pour la console,
- et d'un émulateur (de préférence équipé d'un débogueur) afin de tester nos créations.

Petite précision à propos des outils de développement :

Pour développer en C sur Game Boy, on utilisait généralement un SDK nommé **GBDK** qui date du début des années 2000. Ce SDK n'est plus mis à jour et le compilateur qu'il intègre est malheureusement bogué. C'est pourquoi je vais utiliser dans cet article une version plus récente du compilateur **SDCC**, accompagné de **gbdk-n**, une version modernisée et mise à jour des bibliothèques qui étaient présentes dans GBDK.

Installer la chaîne de compilation

Avant toute chose, il faut commencer par installer une version de SDCC compilée avec le support de la plateforme gbz80, ainsi que

les bibliothèques associées. Si vous êtes sous Linux, vous trouverez généralement tout le nécessaire dans les dépôts de votre distribution. Pour Debian et Ubuntu, vous pouvez tout installer à l'aide de la commande suivante :

```
sudo apt install build-essential sdcc sdcc-libraries
```

Si vous êtes sous Windows, vous devez vous rendre sur le site sdcc.sourceforge.net afin d'y télécharger la dernière version du compilateur. Au moment où j'écris cet article, il s'agit de sdcc-3.9.0-x64-setup.exe. Une fois téléchargé, lancez l'*installer* et faites bien attention aux deux points suivants :

- Si vous ne faites pas une installation « *Full* », veillez à bien cocher la case « SDCC GBZ80 library » lors de la sélection des composants (fig. 1).
- À la fin de l'installation, vérifiez que la case « Add <Chemin de votre installation> to the PATH » soit bien cochée, sinon vous aurez des difficultés à compiler vos jeux (fig. 2).

Obtenir et compiler gbdk-n

Une fois la chaîne de compilation installée, il faut récupérer les bibliothèques que nous allons utiliser pour développer nos jeux Game Boy. Pour ce faire, commencez par cloner le dépôt de **gbdk-n** qui se trouve sur Github :

```
git clone https://github.com/flozz/gbdk-n.git
```

NOTE : si vous n'avez pas Git d'installé, vous pouvez vous rendre à l'URL ci-dessus et télécharger le projet sous la forme d'une archive zip.

Ensuite, il faut compiler les bibliothèques que l'on vient de télécharger. Pour cela, ouvrez un terminal dans le dossier de gbdk-n et tapez-y l'une des commandes suivantes.
Pour Linux ou Windows (via CMD.EXE) :

```
make
```

Pour Windows (via Git Bash) :

```
./make.bat
```

Récupérer un émulateur

Dernière étape avant de se lancer dans le code : récupérer un émulateur, on en aura besoin pour tester nos créations. Si n'importe quel émulateur peut faire l'affaire, tout du moins au début, je vous recommanderais tout de même **BGB**, qui propose une émulation fidèle du matériel et fournit pas mal d'outils pour aider au débogage. BGB n'est par contre disponible que pour Windows, mais il fonctionne heureusement parfaitement avec WINE sous Linux.
Pour télécharger BGB, il faut vous rendre sur le site bgb.bircd.org et télécharger le zip proposé. Il n'y a pas d'installateur : pour utiliser l'émulateur, il suffit de décompresser l'archive et de lancer l'application « bgb.exe ».
Pour ceux qui sont sous Linux, il vous faudra commencer par installer WINE. Le logiciel devrait être disponible dans les dépôts de votre distribution, et peut être installé avec la commande suivante sous Debian / Ubuntu :

```
sudo apt install wine
```

Et pour lancer le logiciel, il vous suffira d'utiliser la commande suivante :

```
wine chemin/vers/bgb.exe
```

3

Écrire notre premier programme

Maintenant que tout est prêt, on va pouvoir écrire notre premier programme pour la Game Boy. Il s'agira bien évidemment d'un « *Hello World* », tradition oblige. On va donc commencer par créer

PETITE ASTUCE

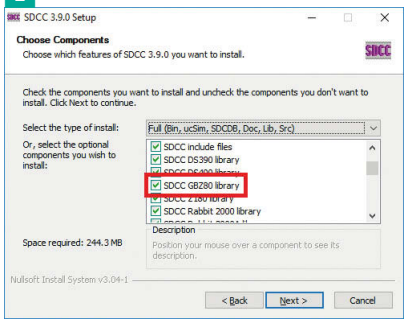
BGB dispose d'une option « *watch* » qui recharge automatiquement la ROM à chaque fois qu'elle est modifiée (donc à chaque fois qu'on la recompile). C'est extrêmement pratique lorsque l'on est en train de développer : ça évite de relancer l'émulateur à chaque fois que l'on veut tester une modification de notre programme. Pour utiliser cette fonctionnalité, il faut utiliser la commande suivante :

```
bgb.exe --watch <rom_a_lancer.gb>
```

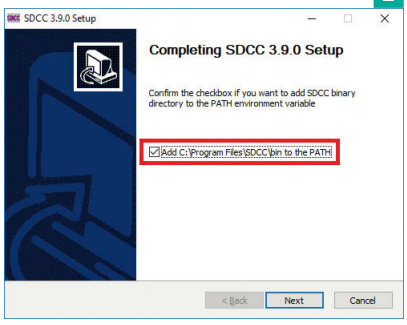
Si vous êtes sous Linux, pensez bien à lancer cette commande avec Wine :

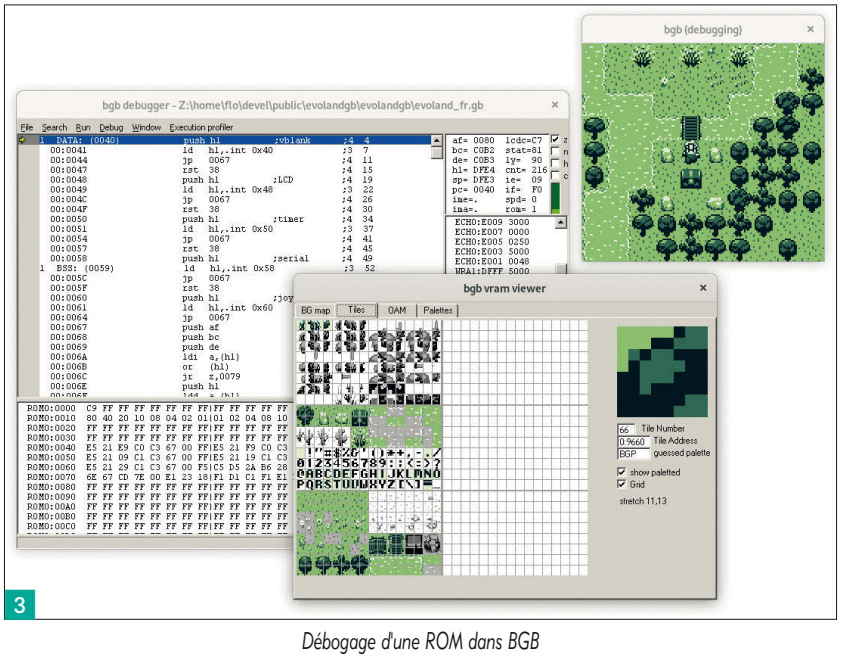
```
wine bgb.exe --watch <rom_a_lancer.gb>
```

1



2





3

Débogage d'une ROM dans BGB

un fichier « hello.c » dans lequel on écrira notre programme, et pour simplifier les commandes pour la suite de cet article, on va placer gbdk-n et bgb à côté. Mon dossier de projet ressemble donc à ceci :

```
HelloProject/
  |
  +-- gbdk-n/
  |
  +-- bgb.exe
  |
  +-- hello.c
```

Il ne nous reste plus qu'à remplir le fichier « hello.c » avec le code suivant :

```
#include <stdio.h>

void main() {
  printf("Hello world!\n");
}
```

Eh oui, surprise, ça n'est pas plus compliqué que ça ! On aurait écrit exactement la même chose si on avait voulu faire le même programme pour un ordinateur actuel. On a simplement inclus la bibliothèque stdio afin de disposer de la fonction printf() et utilisé cette dernière pour écrire le texte souhaité.

Compiler notre programme

Il ne nous reste plus qu'à compiler le programme et à générer la ROM. Cela va se passer en trois étapes :

- compiler le programme,
- faire l'édition des liens,
- et générer la ROM.

Pour compiler le programme, ouvrez un terminal dans le dossier du projet et tapez-y la commande suivante pour Linux ou Windows (via Git Bash) :

```
./gbdk-n/bin/gbdk-n-compile.sh hello.c
```

Si vous êtes sous Windows avec CMD.EXE, la commande devient :

```
gbdk-n\bin\gbdk-n-compile.bat hello.c
```

Cette commande va générer tout un tas de fichiers, mais celui qui nous intéresse est « hello.rel ». Il s'agit de la version compilée mais pas encore *linkée* du code se trouvant dans « hello.c ». Les fichiers .rel correspondent aux fichiers .o que l'on retrouve lorsque l'on compile avec les compilateurs plus « conventionnels ».

Maintenant que le programme est compilé, il faut faire l'édition des liens. Il s'agit donc de donner à l'éditeur de liens tous les fichiers .rel qui composent notre programme pour qu'il puisse faire son travail. Dans notre cas, étant donné qu'on a un seul fichier .c, on a également un seul fichier .rel. On peut donc le compiler avec l'une des commandes suivantes suivant si l'on est sous Linux / Windows avec Git Bash :

```
./gbdk-n/bin/gbdk-n-link.sh hello.rel -o hello.ihx
```

ou Windows avec CMD.EXE :

```
gbdk-n\bin\gbdk-n-link.bat hello.rel -o hello.ihx
```

Cette fois encore, plusieurs fichiers ont été générés, et celui qui nous intéresse est « hello.ihx ». Il s'agit de notre programme, dans le format Intel HEX, que l'on rencontre couramment dans le monde de l'embarqué lorsque l'on programme des microcontrôleurs.

Il ne nous reste plus qu'à générer la ROM depuis ce fichier .ihx. Pour Linux et Windows avec Git Bash, cela se fait avec la commande suivante :

```
./gbdk-n/bin/gbdk-n-make-rom.sh hello.ihx hello.gb
```

Et si vous utilisez Windows avec CMD.EXE, la commande devient :

```
gbdk-n\bin\gbdk-n-make-rom.bat hello.ihx hello.gb
```

Et voilà, en résultat de cette dernière commande, on obtient le fichier « hello.gb » qui est la ROM contenant notre programme !

Tester notre ROM dans un émulateur

Comme on l'a vu un peu plus tôt, n'importe quel émulateur peut faire l'affaire, mais si vous avez choisi BGB, vous pouvez lancer votre ROM avec l'une des commandes suivantes.

Linux :

```
wine bgb.exe hello.gb
```

Windows (Git Bash) :

```
./bgb.exe hello.gb
```

Windows (CMD.EXE) :

```
bgb hello.gb
```

Il ne reste plus qu'à admirer le résultat ! 4

Un peu d'interactivité

Maintenant que l'on sait afficher du texte sur l'écran de la console, voyons comment nous pouvons interagir avec le joueur. La Game Boy dispose d'un *gamepad* composé de **8 boutons** (une croix directionnelle, A, B, START et SELECT) et GBDK nous fournit un ensemble de fonctions et de constantes pour les utiliser.

Nous devons donc commencer par importer la bibliothèque qui contient ces fonctions et constantes :

```
#include <gb/gb.h>
```

GBDK nous fournit trois fonctions pour lire les entrées du joueur :

- deux d'entre elles sont bloquantes,
- et la dernière est non bloquante.

La fonction à utiliser dépend de nos besoins, et c'est ce que l'on va voir tout de suite.

4

Fonctions bloquantes

Si l'on souhaite attendre que le joueur appuie sur un bouton (par exemple START), nous pouvons utiliser la fonction waitpad() :

```
waitpad(J_START);
```

En paramètre de la fonction, nous avons passé la constante J_START. Il s'agit d'un masque nous permettant d'indiquer à la fonction quelle(s) touche(s) on attend. Il existe une constante similaire pour chaque bouton :

- J_UP : bouton haut du D-Pad,
- J_DOWN : bouton bas du D-Pad,
- J_LEFT : bouton gauche du D-Pad,
- J_RIGHT : bouton droit du D-Pad,
- J_A : bouton A,
- J_B : bouton B,
- J_START : bouton START,
- J_SELECT : bouton SELECT.

Il est bien sûr possible d'attendre plusieurs touches en même temps, il suffit pour cela d'ajouter les masques des touches souhaitées à l'aide de l'opérateur binaire **OU** :

```
// Ici on attend l'appui du bouton A ou B
waitpad(J_A | J_B);
```

Et si l'on souhaite savoir laquelle des deux le joueur a pressée, il faut lire la valeur de retour de la fonction et appliquer le masque du bouton à tester à cette valeur en utilisant l'opérateur binaire **ET** :

```
UINT8 keys = waitpad(J_A | J_B);
if (keys & J_A) {
  // Bouton A appuyé
}
if (keys & J_B) {
  // Bouton B appuyé
}
```

Résumons tout ça sous la forme d'un unique programme afin de pouvoir le tester dans un émulateur :

```
#include <stdio.h>
#include <gb/gb.h>

void main(void) {
  printf("Press START\n");
  waitpad(J_START);
  printf("Ok...\n");

  printf("Now press A or B\n");
  UINT8 keys = waitpad(J_A | J_B);
  if (keys & J_A) {
    printf("You pressed A!\n");
  }
  if (keys & J_B) {
    printf("You pressed B!\n");
  }
}
```

Il ne reste plus qu'à compiler le programme comme on l'a vu précédemment, et à le lancer dans l'émulateur. Si tout se passe bien, vous devriez arriver à un résultat similaire à celui-ci : 5

Il existe également une autre fonction permettant d'attendre cette fois-ci que le joueur ait relâché tous les boutons :

```
waitpadup();
```

Ces fonctions sont donc bloquantes : elles bloquent l'exécution du programme jusqu'à ce que le joueur ait effectué les actions demandées. Leur utilisation peut être pratique sur un écran de titre ou dans des menus. Il est également possible de les utiliser pour des jeux très simples, mais la plupart du temps, on ne souhaite pas bloquer le programme en attendant des entrées de l'utilisateur. Imaginez si dans le jeu *Tetris* les tétrominos ne pouvaient pas tomber parce que le programme est en attente d'une entrée de l'utilisateur... ça serait tout de suite moins fun...

Fonction non bloquante

Heureusement, dans les cas où les fonctions bloquantes ne peuvent pas être utilisées, GBDK nous fournit une dernière fonction, joypad(), nous permettant de lire en temps réel l'état des touches du *gamepad*. Voici un court exemple d'utilisation de cette fonction :

```
UINT8 keys;

while (1) {
  keys = joypad();

  if (keys & J_A) {
    // Le bouton A est actuellement pressé
  }
}
```

Si on exécute le programme ci-dessus, on saura quand le bouton A est pressé, mais l'information se répétera à chaque tour de boucle. Parfois, c'est ce que l'on souhaite (lorsque l'on fait avancer le personnage par exemple), et parfois non. Si on souhaite savoir si le bouton vient d'être pressé ou d'être relâché, il faudra conserver l'état des boutons du tour de boucle précédent.

Voici un exemple complet illustrant l'utilisation de la fonction joypad() :

```
#include <stdio.h>
#include <gb/gb.h>

void main(void) {
  UINT8 prev_keys = 0;
  UINT8 keys;

  printf("Press A or B\n");

  while (1) {
    keys = joypad();

    if (keys & J_A && !(prev_keys & J_A)) {
      printf("A pressed\n");
    }
    if (!(keys & J_A) && prev_keys & J_A) {
      printf("A released\n");
```

```
Press START
Ok...
Now press A or B
You pressed A!
```

5

Abonnez-vous à Programmez! Abonnez-vous à Programmez! Abonne

[Programmez!] Le magazine des développeurs

Nos classiques

1 an
11 numéros — 49€*

2 ans
22 numéros — 79€*

Etudiant
1 an - 11 numéros — 39€*

* Tarifs France métropolitaine

Abonnement numérique

PDF — 35€
1 an - 11 numéros

Option : accès aux archives — 10€

Souscription uniquement sur
www.programmez.com

Offres 2019

1 an — 59€
11 numéros
+ 1 vidéo ENI au choix :

- Symfony 3*
  Développer des applications web robustes
  (valeur : 29,99 €)
- Raspberry Pi*
  Apprenez à réaliser et piloter une lumière d'ambiance
  (valeur : 29,99 €)

2 ans — 89€
22 numéros
+ 1 vidéo ENI au choix :

- Symfony 3*
  Développer des applications web robustes
  (valeur : 29,99 €)
- Raspberry Pi*
  Apprenez à réaliser et piloter une lumière d'ambiance
  (valeur : 29,99 €)

* Offre limitée à la France métropolitaine

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

ABONNEMENT à retourner avec votre règlement à :
Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex.

☐ Abonnement 1 an : 49 €
☐ Abonnement 2 ans : 79 €
☐ Abonnement 1 an Etudiant : 39 €
Photocopie de la carte d'étudiant à joindre

☐ Abonnement 1 an : 59 €
11 numéros + 1 vidéo ENI au choix :
☐ Abonnement 2 ans : 89 €
22 numéros + 1 vidéo ENI au choix :

☐ Vidéo : Symfony 3
☐ Vidéo : Raspberry Pi

PROG 231
Valable jusqu'au 30 août 2019

☐ Mme ☐ M. Entreprise : ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| Fonction : ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| |||

Prénom : ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| Nom : ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| |||

Adresse : ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| |||

Code postal : ||| ||| ||| ||| ||| Ville : ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| |||

email indispensable pour l'envoi d'informations relatives à votre abonnement

E-mail : ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| @ ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| ||| |||

☐ Je joins mon règlement par chèque à l'ordre de Programmez !
☐ Je souhaite régler à réception de facture
*** Tarifs France métropolitaine**

A DÉCOUVRIR D'URGENCE

NOUVEAU !

LE CADEAU GEEK IDÉAL

Découvrez l'âge d'or des micro-ordinateurs de 1973 à 2007

9,99 €
(+ 3 € de frais postaux)

116 pages - Format magazine A4

☐ **Découvrez l'âge d'or des micro-ordinateurs de 1973 à 2007** :
9,99 € (+3 € de frais postaux) = 12,99 €

Commande à envoyer à :
Programmez!
57 rue de Gisors - 95300 Pontoise

PROG 231

☐ Mme ☐ M. Entreprise : Fonction :

Prénom : Nom :

Adresse :

Code postal : Ville :

E-mail : @

☐ Je joins mon règlement par chèque à l'ordre de Programmez !

commandez sur www.programmez.com

```
    }

    if (keys & J_B && !(prev_keys & J_B)) {
      printf("B pressed\n");
    }
    if (!(keys & J_B) && prev_keys & J_B) {
      printf("B released\n");
    }

    prev_keys = keys;

    // Attend que l'écran soit rafraîchi
    wait_vbl_done();
  }
}
```

Voici un aperçu du programme exécuté dans l'émulateur : 6

```
Press A or B
A pressed
A released
B pressed
B released
A pressed
B pressed
B released
A released
```

6

Afficher des images

On sait à présent interagir avec l'utilisateur, il ne nous manque plus qu'une chose pour pouvoir commencer à faire un jeu : afficher des images. Mais avant de mettre les mains dans le code, il faut que l'on fasse le point sur les capacités graphiques de la console.

La Game Boy dispose d'un écran de **160×144 pixels**... Mais ces pixels ne sont pas adressables. Dit autrement, il ne nous est pas possible de choisir directement la couleur de chaque pixel de l'écran. Le système vidéo de la console fonctionne avec un système de **tuiles**. Il s'agit de petites images de **8×8 pixels** que l'on va placer dans la mémoire vidéo de la machine, et on lui indiquera dans un second temps où les afficher. Il est donc plus juste de se représenter l'écran comme une mosaïque de **20×18 tuiles**.

Pour disposer ces tuiles à l'écran, on dispose de trois éléments que l'on peut considérer comme des « couches » ou des « calques » :

- La couche **Background** : il s'agit d'une grille de **32×32 tuiles** que l'on peut faire défiler (c'est pour ça qu'elle est plus grande que l'écran). C'est sur cette couche que l'on va afficher la plupart des éléments d'un jeu.
- La couche **Window** : il s'agit là encore d'une grille, que l'on ne peut en revanche pas faire défiler, mais que l'on peut déplacer pour recouvrir partiellement ou en totalité la couche *Background*. On s'en sert généralement pour afficher le nombre de vies du joueur ou des boîtes de dialogue.
- Les **sprites** : ce sont des éléments composés de 1 ou 2 tuiles (8×8 ou 8×16 pixels) que l'on peut positionner librement à l'écran. La Game Boy peut afficher jusqu'à 40 *sprites* simultanément à l'écran, avec toutefois une restriction : on ne peut pas en aligner plus de 10 horizontalement. Les *sprites* sont utiles pour afficher le joueur, les ennemis, bref tous les éléments mobiles à l'écran. 7

Pour les besoins de cet article, nous n'aborderons que la couche *Background*.

Important
À partir de maintenant, il ne faudra plus importer la bibliothèque stdio.h **ni utiliser la fonction** printf()**. Pour afficher du texte, cette bibliothèque remplit une bonne partie de la mémoire vidéo avec des tuiles représentant les caractères affichables. Si l'on souhaite afficher du texte dans notre jeu, il faudra le faire nous-même.**

Tileset

Comme nous venons de le voir, pour afficher des images, il faut commencer par stocker des tuiles dans la mémoire vidéo de la console. Bien évidemment, la Game Boy ne sait pas lire des images au format PNG ou JPEG, il faut les encoder dans un format spécifique. Prenons par exemple le *tileset* suivant que j'avais créé il y a quelque temps pour un jeu que je n'ai jamais terminé (il s'agit d'un *Snake* pour ceux qui se poseraient la question) :

Une fois encodé dans le format de la console, cela donne :

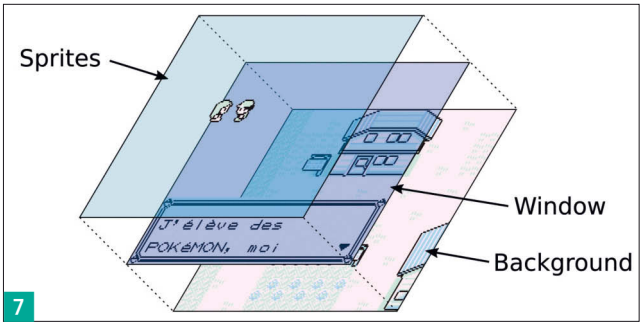
```
const UINT8 TILESET[] = {
  // Tile 00: Blank
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  // Tile 01: Block
  0xff, 0x01, 0x81, 0x7f, 0xbd, 0x7f, 0xa5, 0x7b,
  0xa5, 0x7b, 0xbd, 0x63, 0x81, 0x7f, 0xff, 0xff,
  // Tile 02: Snake Body
  0x7e, 0x00, 0x81, 0x7f, 0x81, 0x7f, 0x81, 0x7f,
  0x81, 0x7f, 0x81, 0x7f, 0x81, 0x7f, 0x7e, 0x7e,
  // Tile 03: Cherry
  0x04, 0x04, 0x04, 0x04, 0x0a, 0x0a, 0x12, 0x12,
  0x66, 0x00, 0x99, 0x77, 0x99, 0x77, 0x66, 0x66,
};
```

Bien sûr, on ne s'amuse pas à faire la conversion à la main, on dispose aujourd'hui d'outils qui font cela automatiquement. J'ai développé l'un d'entre eux pour répondre à mes propres besoins : img2gb que vous trouverez à l'adresse suivante : https://flozz.github.io/img2gb/.

Une fois le *tileset* intégré au code du jeu, il ne reste plus qu'à le charger dans la mémoire vidéo de la console :

```
set_bkg_data(
  0,      // first_tile
  5,      // nb_tiles
  TILESET  // data
);
```

- Le paramètre first_tile indique à partir de quelle case mémoire on va charger nos tuiles,
- nb_tile indique le nombre de tuiles à charger,
- data correspond au tableau contenant les tuiles de notre *tileset*



7

Exemple d'utilisation des différentes couches dans le jeu Pokémon version rouge

Tilemap

Une fois nos tuiles chargées, il faut indiquer à la Game Boy où les afficher. Pour cela on va créer une *tilemap*. Il s'agit tout simplement d'un tableau dans lequel on inscrit dans chaque case le numéro de la tuile à afficher : 8

Ce qui, une fois converti en code, nous donne le tableau suivant :

```
const UINT8 TILEMAP[] = {
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
  1, 0, 2, 2, 2, 0, 0, 3, 0, 1,
  1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
  1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
  1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
  1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
  1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
  1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
};
```

Il ne nous reste plus qu'à charger cette *tilemap* dans la couche *Background* :

```
set_bkg_tiles(
  0,    // x
  0,    // y
  10,   // width
  10,   // height
  TILEMAP // tiles
);
```

- Les paramètres x et y indiquent à quelles coordonnées de la couche *Background* on souhaite coller notre *tilemap*,
- les paramètres width et height correspondent aux dimensions de la *tilemap*,
- enfin, le paramètre tiles est le tableau contenant la *tilemap*.

Afficher la couche *Background*

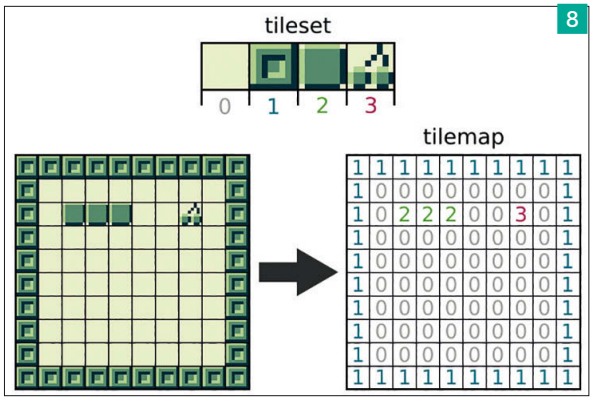
Il ne nous reste plus qu'un dernier détail à régler pour afficher notre image : rendre visible la couche *Background*. Elle est en effet masquée par défaut après le démarrage de la console. Pour ce faire, c'est très simple, il suffit d'utiliser la macro suivante :

```
SHOW_BKG;
```

Exemple complet

Si on recolle tous les morceaux ensemble, on obtient le programme suivant :

```
#include <gb/gb.h>

const UINT8 TILESET[] = {
  // Tile 00: Blank
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  // Tile 01: Block
  0xff, 0x01, 0x81, 0x7f, 0xbd, 0x7f, 0xa5, 0x7b,
  0xa5, 0x7b, 0xbd, 0x63, 0x81, 0x7f, 0xff, 0xff,
  // Tile 02: Snake Body
```

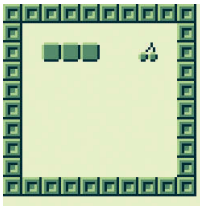


8

```
  0x7e, 0x00, 0x81, 0x7f, 0x81, 0x7f, 0x81, 0x7f,
  0x81, 0x7f, 0x81, 0x7f, 0x81, 0x7f, 0x7e, 0x7e,
  // Tile 03: Cherry
  0x04, 0x04, 0x04, 0x04, 0x0a, 0x0a, 0x12, 0x12,
  0x66, 0x00, 0x99, 0x77, 0x99, 0x77, 0x66, 0x66,
};

const UINT8 TILEMAP[] = {
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
  1, 0, 2, 2, 2, 0, 0, 3, 0, 1,
  1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
  1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
  1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
  1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
  1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
  1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
};

void main(void) {
  set_bkg_data(0, 5, TILESET);
  set_bkg_tiles(0, 0, 10, 10, TILEMAP);
  SHOW_BKG;
}
```

Et voici le résultat une fois compilé et exécuté dans un émulateur : 9

Avec tout ça, vous avez toutes les bases pour développer un petit *puzzle game* relativement simple, comme un morpion ou un démineur.


9

Tester notre ROM sur une vraie Game Boy

Tester notre programme dans un émulateur c'est bien pratique lors de la phase de développement, mais au bout d'un moment, on a envie de voir ce que ça donne sur le matériel réel. Il existe plusieurs solutions pour tester ses ROMs sur une vraie Game Boy, je vais vous en présenter deux :

- les cartouches EverDrive-GB,
- et le duo *flash cart* / *linker*.

Cartouches EverDrive-GB

Les cartouches EverDrive-GB représentent la solution la plus simple pour lancer n'importe quelle ROM, que ce soit un *homebrew* ou un

jeu commercial, sur une Game Boy. Il suffit de placer les ROMs (ainsi que le *firmware* téléchargeable sur le site du constructeur), sur une carte micro SD et de l'insérer dans la cartouche. Au démarrage de la console un menu avec la liste de toutes les ROMs s'affiche, il n'y a plus qu'à sélectionner celle que l'on souhaite lancer.

Ces cartouches sont donc très simples et pratiques d'utilisation, mais coûtent assez cher. Il faut en effet compter entre 70 € et 120 € en fonction du modèle.

Il existe trois modèles de cartouches EverDrive-GB : le X3, le X5 et le X7. Le X3 étant la version la plus basique et le X7 la plus avancée. Si vous optez pour cette solution, je vous recommande personnellement le modèle X5. Il coûte une dizaine d'euros de plus que le modèle X3, mais est bien plus pratique à utiliser. Le modèle X3 dispose en effet d'un petit bouton situé sous le plastique de la cartouche qu'il ne faudra pas oublier de presser avant d'éteindre la console si on souhaite conserver sa progression. Ce bouton fait redémarrer la console logiciellement (*soft reboot*) ce qui permet au *firmware* de récupérer la sauvegarde effectuée par le jeu et de l'inscrire sur la carte SD. Les modèles supérieurs ne souffrent pas de cet inconvénient. 10

Flash cart + linker

Les *flash cart* sont des cartouches réinscriptibles, que l'on programme à l'aide d'un appareil appelé *linker*. Il s'agit d'une solution moins onéreuse que les cartouches EverDrive-GB, mais qui est moins évidente à mettre en œuvre. Cette solution a toufois l'avantage de permettre de créer des cartouches dédiées à un jeu que l'on pourra ensuite distribuer.

Pour créer ce type de cartouches il nous faut donc deux éléments : des cartouches réinscriptibles et un *linker*.

Pour les cartouches, il en existe de plusieurs sortes, de différentes capacités et embarquant du matériel différent. Il faut choisir la cartouche en fonction de la ROM que l'on souhaite y inscrire (toutes ne font pas la même taille et n'ont pas les mêmes besoins). Pour notre cas, le mieux c'est de prendre des cartouches de 32 ko simples (pas de SRAM, pas de contrôleur de banque). On peut trouver ce type de matériel :

- Chez Catskull Electronics (https://catskullelectronics.com/32kcart). Les cartouches sont de bonne facture et coûtent une dizaine de dollars, il faudra cependant faire attention aux frais de port qui sont assez élevés pour une livraison en dehors des États-Unis.
- Ou chez insideGadgets (https://shop.insidegadgets.com/product/gameboy-32kb-flash-cart/). Là encore la qualité est au rendez-vous et le prix est également d'une dizaine de dollars. 11

Pour ce qui est du *linker*, j'utilise personnellement le **GBxCart RW** de chez insideGadgets (https://shop.insidegadgets.com/product/gbxcart-rw/), qui coûte une trentaine de dollars et qui a l'avantage de supporter aussi bien les jeux Game Boy / Game Boy Color que les jeux Game Boy Advance. Il existe également le modèle **GBxCart Mini RW** qui coûte dix dollars de moins, mais qui ne supporte pas les jeux Game Boy Advance. 12

Une fois en possession des deux éléments, il suffit de télécharger sur le site du fabricant un petit logiciel permettant de lire et d'écrire des ROMs sur les cartouches. Je ne vous détaille pas la procédure ici car le manuel fourni par le fabricant documente déjà très bien l'opération. 13

Vers l'infini et au-delà

Cet article n'étant qu'une introduction au développement Game Boy, je n'ai pu qu'effleurer la surface du sujet. Si vous souhaitez aller plus loin, je vous recommande de consulter les ressources suivantes :

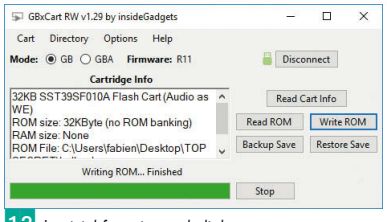
- La série d'articles que je rédige sur mon blog. Il s'agit actuellement de la plus importante ressource en français sur le sujet. Vous trouverez le premier article de la série à l'adresse suivante : https://blog.flozz.fr/2018/10/01/developpement-gameboy-1-hello-world/
- Le dépôt Github **Awesome Game Boy Development**, qui est un effort de la communauté pour centraliser un maximum de ressources autour du développement Game Boy (tutos, documentations, exemples de code, logiciels,...) : https://github.com/gbdev/awesome-gbdev
- Enfin, je vous recommande la chaîne Youtube **Gaming Monsters** qui est remplie de vidéos sur le développement Game Boy : https://www.youtube.com/channel/UCMMhSfBStEti-Lqzs30HYWw/videos

J'espère en tout cas que cet article aura éveillé votre curiosité sur le sujet et qu'il vous aura donné envie de développer vos propres jeux pour la célèbre console portable de Nintendo. •

10

11 *Catskull 32k Flash Cart*

12 *Linker insideGadgets GBxCart RW*



13 *Logiciel fourni avec le linker*

Anne-Charlotte Bournigal
Product Owner Front-End chez Moskitos, éditeur de la plateforme d'intégration de données Crosscut iPaaS. Ingénieur en développement logiciel et spécialiste du développement full-stack (C# .Net, Angular), elle est responsable de toute la partie interface et parcours utilisateur des produits Moskitos.

Mise en place d'une application Angular pour vos APIs

Dans ce tutoriel, nous allons créer une application simple permettant d'afficher la météo d'une ville saisie par l'utilisateur, en utilisant une API existante (https://www.apixu.com/) et gratuite. Nous aurons besoin d'un frontal simple, avec un champ pour saisir un nom de ville, et une zone d'affichage du résultat de la recherche. Nous avons choisi d'utiliser le framework Angular 6 car il permet de créer des applications réactives d'une seule page (Single-page application – SPA) afin d'éviter le chargement à chaque navigation, fluidifiant l'expérience utilisateur.

niveau 100

Présentation de l'environnement du projet

Nous allons avoir besoin d'installer un certain nombre d'éléments qui vont nous être utiles dans le développement de notre application. Pour sa création, nous allons utiliser Visual Studio Code, mais il est possible d'utiliser tout autre environnement de développement.

NodeJS et NPM

NodeJS et NPM (Node Package Manager) sont les indispensables de tout développeur Javascript. NPM est le gestionnaire de *package* (aussi appelé module) officiel de NodeJs, et permet de télécharger tous les *packages* mis à disposition par la communauté.
Aujourd'hui, NPM est directement livré avec NodeJS. Son installation est plutôt simple : direction le site nodejs.org pour télécharger la version nécessaire à votre projet. Une fois le téléchargement et l'installation terminés, vous pouvez vérifier si NodeJs s'est bien installé en tapant cette commande dans votre terminal :

```
node –v
```

Si la version de votre NodeJS s'affiche, c'est qu'il a bien été installé. Vous pouvez aussi vérifier l'installation de NPM via cette commande :

```
npm –v
```

Là aussi, si une version s'est affichée, NPM est bien installé sur votre machine.

```
PS C:\Projects> node -v
v9.5.0
PS C:\Projects> npm -v
5.6.0
```

Figure 1 - Résultats des commandes

Angular

Typescript

Angular est écrit en TypeScript. Il est fortement recommandé d'utiliser du Typescript sur un projet Angular, bien que cela reste facultatif. Dans la mesure où il est possible d'écrire du Javascript dans un fichier TypeScript.
TypeScript est un langage de programmation libre et open source développé par Microsoft en 2012. C'est un sur-ensemble de Javascript et il a pour but d'améliorer la productivité de développement des applications complexes. A savoir que tout code valide en Javascript l'est également en Typescript. Autre particularité du Typescript, il introduit de nouvelles fonctionnalités telles que le typage et la programmation orientée objet. Alors que le Javascript de base est un langage non fortement typé. Dans la pratique, une fois le code écrit en Typescript, il est transpilé (compilation du code source d'un langage vers un autre) en Javascript.

Architecture d'Angular

L'architecture d'Angular repose sur trois briques fondamentales :

- Les **composants** qui définissent les vues de l'application. Chaque écran de l'application est un ensemble de composants imbriqués les uns dans les autres. Par exemple, votre page d'accueil peut avoir un composant pour le menu et un composant pour le header. Un composant nouvellement créé vient avec un fichier HTML, un CSS associé, un fichier TS qui contient une classe permettant de définir la logique de la vue et un fichier de tests pour cette classe.
- Les **services** sont utilisés par les composants et représentent la logique applicative. Ils peuvent être partagés au sein de votre application et permettent d'isoler des briques de codes. Ce qui permet de rendre votre code modulaire, réutilisable et efficace.
- Les **modules** permettent d'organiser votre code en modules fonctionnels distincts. Ils sont composés d'un ensemble de services et de composants. Ils sont très utiles dans le cadre d'application complexe et permettent d'assurer la réutilisation de code. De plus, ils offrent la possibilité de mettre en place du *Lazy Loading*, technique de chargement de module à la demande, ce qui réduit la quantité de code chargée au démarrage de l'application. Toute application Angular a un module de base, appelé généralement *AppModule*, qui permet le démarrage de l'application.

Le framework Angular est très complet; il propose toutes les briques nécessaires à la création et au fonctionnement d'une application. Par exemple :

- Un module de gestion de routes – « Router » : permet de créer les différentes routes de navigation de l'application ;
- Un module http – « HttpClient » : permet de faire des appels à l'API.

Démarrage du projet avec Angular

Angular Cli

Présentation

La génération de base d'une application Angular est un travail assez répétitif. Il est tout à fait possible de le faire manuellement si vous avez du temps et que vous souhaitiez apprendre. Mais pour

une application professionnelle et complexe, il vaut mieux partir sur de bonnes bases.

Ainsi a été mis à disposition des développeurs un outil bien pratique : Angular CLI. C'est un outil d'interface de lignes de commande qui permet d'automatiser les tâches de développement les plus courantes. Quelques exemples :

- Générer l'architecture de base de votre application ;
- Lancer les tests de l'application ;
- Générer des nouveaux composants, services etc. ;
- Compiler l'application et construire une archive qui peut être livrée en production.

Angular CLI apporte un gain de temps considérable et améliore la qualité du code dans le développement d'une application. Il n'est pas obligatoire de l'utiliser, mais il apporte une belle valeur ajoutée. Webpack est un outil puissant, très configurable et connu dans le développement d'application. Il permet, entre autres, de compiler des modules Javascript. Son niveau de configuration le rend flexible mais peut engendrer des difficultés dans sa mise en place. L'équipe d'Angular souhaitait faciliter son utilisation dans un projet Angular, ils l'ont donc embarqué dans Angular CLI. En tant que développeur Angular, nous n'avons donc plus à nous soucier de Webpack.

Mise en place dans notre projet

Une fois que vous avez créé votre nouveau répertoire pour votre projet, nous pouvons, via NPM, installer Angular CLI :

```
npm install –g @angular/cli
```

La commande **ng** (de Angular CLI) est alors globalement installée dans votre système. Vous pouvez vérifier ensuite si Angular CLI a bien été installé via cette commande :

```
ng -v
```

Qui vous retourne la version que vous avez installée :

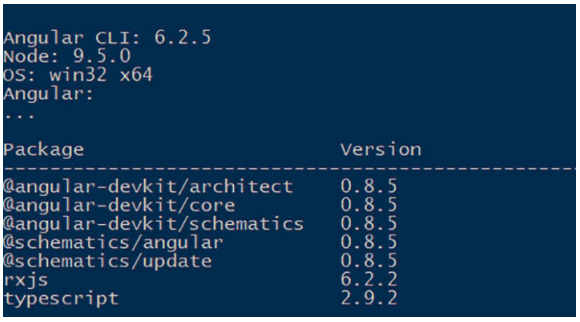


Figure 2 - Version de Angular-CLI installée

Nous pouvons dès lors utiliser la commande qui va nous permettre de créer et initialiser notre application Angular :

```
ng new [nom-application]
```

Dans le répertoire vide, toute une application a été générée et présente la structure suivante :

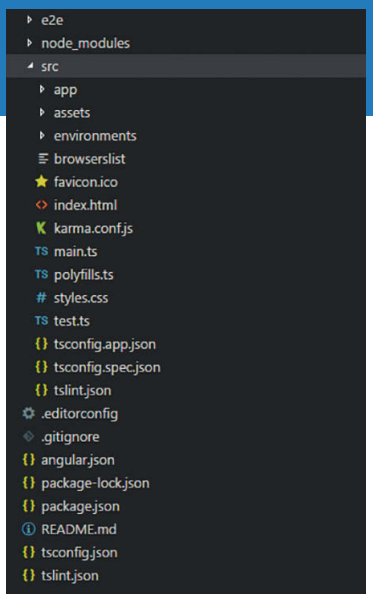


Figure 3 - Génération du projet par défaut

Sur tous les fichiers qui ont été générés, j'attire votre attention sur certains d'entre eux :

Chemin du fichier	Description
angular.json	Permet de configurer Angular CLI.
package.json	Gestion de dépendances de npm, il regroupe les dépendances de l'application.
tsconfig.json	Configuration TypeScript par défaut pour l'application.
Src/karma.conf.js	Configuration des tests. Les tests unitaires en Angular, reposent sur deux outils : Karma : moteur de tests Jasmine : langage d'assertion
src/main.ts	Le point d'entrée principal de l'application. Compile l'application et amorce le module racine de l'application (AppModule) à s'exécuter dans le navigateur.
src/app/	Contient les fichiers de composants dans lesquels la logique et les données de l'application sont définies.
src/assets/	Les images ou d'autres fichiers à copier tels quels lors de la création de votre application.
src/app/app.module.ts	Module racine de l'application, nommé AppModule, qui indique à Angular comment assembler l'application. A la création de l'application, il ne déclare que le composant AppComponent. Tout autre composant ajouté devra être renseigné ici.

Une fois votre application en place, exécutons rapidement l'application Angular. Ouvrez votre terminal à la racine du projet et saisissez la commande suivante :

```
ng serve
```

Figure 4 - Résultat de la commande démarrant l'application

Le serveur web est alors démarré et vous pouvez naviguer sur l'adresse indiquée dans le résultat (ici : http://localhost:4200/). Vous pouvez alors voir la page d'accueil d'une application Angular générée via Angular CLI :

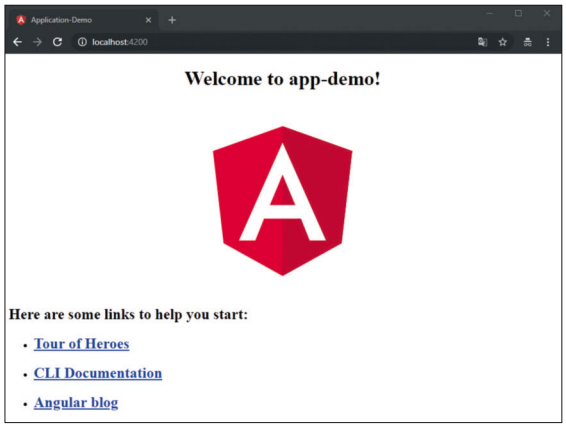


Figure 5 - Page de démarrage de l'application Angular

Structure de l'application

Le dossier de base de notre application (src/app), contient actuellement le module (app.module.ts) et le composant de base de l'application (app.component.ts). Il existe un grand nombre de propositions d'architectures sur le web dont on peut s'inspirer. Pour cette application, nous allons rester sur quelque chose de simple, mais nous allons tout de même mettre en place une architecture un peu poussée.

Le dossier de base va contenir 3 sous-dossiers :

- **Module** : permet de découper les « pages » de son application en plusieurs modules. Nous n'utiliserons pas ce dossier dans notre application, car elle est très simple, mais il est important de le mentionner. En effet, penser son application sous forme de plusieurs modules permet la mise en place du **Lazy Loading**. Pour des applications de grande taille, la page d'accueil sera lente à s'afficher et le Lazy Loading permet de charger des modules à la demande afin de ne pas encombrer le lancement initial de l'application.
- **Shared** : regroupe tous les éléments (composants, services, etc.) qui seront amenés à être utilisés partout dans l'application.
- **Core** : regroupe les éléments de base de l'application (l'authentification, les composants de base tel que la sidebar et la navbar, les services).

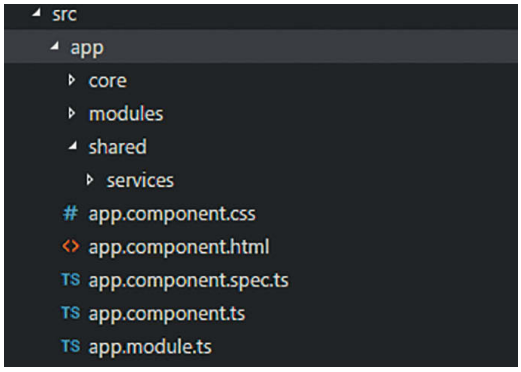


Figure 6 - src/app structuré

Mise en place de la classe de service appelant l'API et de son test unitaire

Nous allons dans un premier temps mettre en place le service qui va s'occuper d'effectuer l'appel à l'API et de récupérer les informations. Comme nous avons pu le voir précédemment, Angular CLI nous permet également de générer le plus simplement du monde nos fichiers de base. Nous n'avons qu'à nous mettre au niveau du dossier où nous souhaitons créer notre service (ici : src/app/shared/services) et taper la commande suivante :

```
ng generate service [nom-du-service]
```

Deux fichiers vont être automatiquement générés : un « .ts » (le service) et un « .spec.ts » (les tests associés à ce service).

Mettre en place @angular/http

Pour utiliser le module Http dans nos composants, nous devons importer @angular/common/http dans la classe de service dans laquelle nous allons faire les appels à l'API. Après, nous pouvons simplement l'injecter via Dependency Injection dans le constructeur du service :

```
import { Injectable, eisObservable } from '@angular/core';
import { Observable } from 'rxjs';
import { map } from 'rxjs/operators';
import { Weather } from '../models/weather'
//Import HTTPCLIENT
import { HttpClient } from '@angular/common/http'

@Injectable({
  providedIn: 'root'
})
export class WeatherService {

  //Inject HTTPCLIENT
  constructor(private http: HttpClient) { }

}
```

Figure 7 -Import et Injection de Angular HTTPClient

Nous devons également importer le module **HttpClientModule** d'Angular dans le module racine de notre application (*app.module.ts*) :

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';

//IMPORT HTTP CLIENT MODULE
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})

export class AppModule { }
```

Figure 8 - Import de Http Module

Création de l'appel à l'API GET

Les injections faites, nous sommes prêts pour faire nos appels à l'API. Pour notre application, nous aurons besoin d'effectuer un appel GET sur l'API. En Angular, nous pouvons utiliser soit les Observables, soit les Promises, dans le cas d'appel d'API. Tout simplement parce que quand nous effectuons un appel à l'API, nous devons attendre la réponse de cette dernière : les Promises comme les Observables permettent d'exécuter du code de façon asynchrone. Le principe est simple, les Observables vont émettre des évènements qui seront interceptés par les observateurs. Il est tout à fait possible de combiner des Observables, de modifier des évènements à la volée, de les filtrer, etc.

Petit point sur les avantages des Observables face aux Promises :

- Un Observable est appelé uniquement lorsque nous souscrivons à ce dernier, il est 'Lazy'. Alors qu'une Promise, dès qu'elle est créée, est exécutée quoi qu'il arrive.

- Il est possible d'annuler un Observable en cours d'exécution. Impossible avec les Promises.
- Un Observable peut être répété : s'il a échoué, il peut réussir lors d'une nouvelle exécution.
- Les Observables peuvent s'utiliser avec tous les opérateurs RxJS (map, filter, reduce etc.).

Avec le design pattern Observable-Observer, nous pouvons parler de programmation réactive, très à la mode dernièrement. Pour faire simple, avec la programmation réactive, il est possible de créer des flux à partir de tout et de n'importe quoi (variables, entrées utilisateurs, propriétés, structures de données, etc.). Et grâce à des outils disponibles (comme RxJS par exemple) nous pouvons combiner, filtrer et créer tous ces flux. Nous allons donc utiliser les Observables dans notre service.

Nous n'avons besoin que d'effectuer un GET sur notre API, qui nous permettra de récupérer la météo d'une ville saisie par l'utilisateur (*ligne 21*) :

```
1  import { Injectable, oisObservable } from '@angular/core';
2  import { Observable } from 'rxjs';
3  import { map } from 'rxjs/operators';
4  import { Weather } from '../models/weather'
5  //Import HTTPCLIENT
6  import { HttpClient } from '@angular/common/http'
7
8  @Injectable({
9    providedIn: 'root'
10 })
11 export class WeatherService {
12
13   //FREE JSON API : https://www.apixu.com/api-explorer.aspx
14   //You need to provide your API KEY here
15   public apiKey: string = "[YOUR_API_KEY]";
16   public weather_api_url: string = "http://api.apixu.com/v1/current.json"
17
18   //Inject HTTPCLIENT
19   constructor(private http: HttpClient) { }
20
21   public getCityWeather(city: string): Observable<Weather> {
22     //We build the URL with the query params : Key (apiKey) and q (= city name)
23     var build_url = `${this.weather_api_url}?key=${this.apiKey}&q=${city}`
24
25     return this.http.get(build_url)
26     .pipe(
27       map(response => (response as Weather))
28     );
29   }
30 }
```

Figure 9 - Implémentation du WeatherService permettant de requêter l'API

Pour appeler l'API, nous avons besoin de récupérer son APIKey (*ligne 15*) directement sur le site après inscription. Dans la documentation de l'API, nous avons récupéré la route nécessaire à la récupération des informations d'une ville.

A la *ligne 25*, notre méthode **getCityWeather** va nous renvoyer un Observable, auquel nous pourrons souscrire depuis nos différents composants.

En amont, nous avons défini une classe **Weather** (*ligne 4*) qui représente la structure de l'objet retourné par l'API. Nous pouvons alors typer le retour de notre méthode **getCityWeather**.

Création du test unitaire associé

Nous pouvons alors tester notre méthode. **HttpClientTestingModule** est l'un des ajouts intéressants apparus avec **HttpClient** dans Angular. Avec ce module, il est possible de tester un service sans utiliser une implémentation de serveur réelle pour traiter les requêtes http de l'application Angular. La méthode **getCityWeather** utilise **HttpClient** pour récupérer de la donnée du serveur. Nous aimerions donc créer un test pour nous assurer que la tâche est gérée correctement. Pour cela, nous devons ouvrir la classe *weather.service.spec.ts*.

```
1  import { TestBed, inject } from '@angular/core/testing';
2  import { WeatherService } from './weather.service';
3  import { HttpClientTestingModule, HttpTestingController } from '@angular/common/http/testing';
4  import { Weather } from '../models/weather'
5
6  describe('WeatherService', () => {
7    let weatherService: WeatherService;
8    let httpMock: HttpTestingController;
9
10   //Before each tests, we store the service and the httpmock in variables
11 ⊞ beforeEach(() => {···
19   });
20
21 ⊞ it('should get weather for a given city', () => {···
73   });
74
75   //Check after each tests if there are not outstanding HTTP calls
76 ⊞ afterEach(() => {···
78   });
79 });
```

Figure 10 - Structure des tests unitaires

Nous avons mis en place les principaux éléments de notre fichier de tests :

- **describe** : définit une suite de tests. Il nous permet aussi de nommer la suite de tests, nous avons donné ici le nom de notre service.
- **beforeEach/afterEach** : exécution de code avant et après chaque test.
- **it** : bloc de description de la fonctionnalité à tester. Dans notre cas, la méthode **getCityWeather** de notre service.

Avant chaque bloc de test (**it**), nous allons définir un module de test pour notre service Angular. Pour ce faire, nous devons configurer le module de tests avec les objets qui vont intervenir dans notre test. Dans notre cas : WeatherService et HttpClientTestingModule. Et nous stockons également le service et le mock http dans des variables pour pouvoir les utiliser dans nos tests.

```
10   //Before each tests, we store the service and the httpmock in variables
11   beforeEach(() => {
12     TestBed.configureTestingModule({
13       providers: [WeatherService],
14       imports: [HttpClientTestingModule]
15     });
16
17     weatherService = TestBed.get(WeatherService);
18     httpMock = TestBed.get(HttpTestingController);
19   });
```

Figure 11 — Détail du bloc BeforeEach

Avec ce bloc, notre module de test sera redéfini à chaque nouveau test **it()**. Nous pouvons maintenant développer notre test. Nous souhaitons donc vérifier que notre méthode **getWeatherService** est bien appelée, et nous retourne un objet **Weather** attendu. Le nom inscrit en premier paramètre du **it** décrit le but du test en détail.

```
21   it('should get weather for a given city', () => {
22     //Fake object matching the result of the HTTP call
23     const resultWeather: Weather = {
24       current: {···
48       },
49       location: {···
58       }
59     };
60
61     //We call the service
62     weatherService.getCityWeather('paris').subscribe(weather => {
63       expect(weather).toBeDefined();
64       expect(weather).toBe(resultWeather);
65     })
66
67     // We set the expectations for the HttpClient mock
68     const req = httpMock.expectOne(`${weatherService.weather_api_url}?key=${weatherService.apiKey}&q=paris`);
69     expect(req.request.method).toBe("GET");
70
71     // Then we set the fake data to be returned by the mock
72     req.flush(resultWeather);
73   });
```

Figure 12 — Détail du bloc It

Notre test se découpe en 4 étapes :

- Nous créons un objet correspondant au résultat attendu de l'appel de notre méthode ;
- Nous appelons la méthode du service et nous souscrivons à l'appel http qu'elle retourne, et nous allons vérifier les assertions suivantes après récupération du résultat : que l'objet retourné existe et qu'il correspond à l'objet attendu ;
- Nous allons paramétrer la requête http qui va avoir lieu : en s'attendant à avoir un GET avec la bonne URL renseignée ;
- Nous allons résoudre cette requête avec **flush()** : ce qui entraîne la fin de notre souscription, retourne le résultat renseigné et donne la main aux **expect()** (*ligne 63*).

Enfin, après chaque test, nous vérifions que toutes les requêtes ont bien été terminées avec le bloc **afterEach** :

```
//Check after each tests if there are not outstanding HTTP calls
afterEach(() => {
  httpMock.verify();
});
```

Figure 13 — Détail du bloc afterEach

Pour lancer les tests de l'application, Angular-Cli nous fournit la commande suivante :

```
ng test
```

Une page web s'ouvre et affiche la sortie de test, nous indiquant que notre test « **should get weather for a given city** » pour le service « **weatherService** » est bien passé, comme ceci :

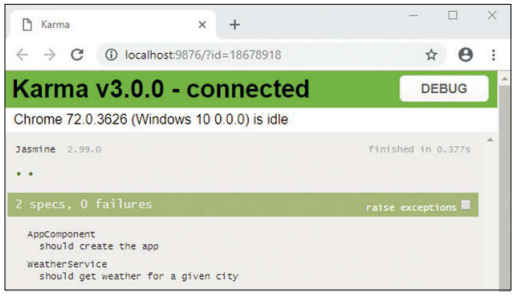


Figure 14 - Jasmine HTML Reporter

Création de l'écran affichant le résultat

Installation de Angular Material

Pour réaliser le design de l'application, nous allons utiliser Angular Material. Il s'agit d'une collection de composants material design pour Angular. Grâce à ces composants, nous pouvons très facilement appliquer un material design à notre application. Angular étant étroitement lié à Angular Material, nous devons installer sur notre projet la version 6 de celui-ci. Nous avons besoin d'installer trois packages à l'aide de la commande **npm install** :

- @angular/material : le package officiel Material pour Angular ;
- @angular/animations : certains composants du package officiel ont besoin des animations, il est donc nécessaire d'avoir ce package ;
- @angular/cdk : Angular Component Development Kit, ajoute de nombreux outils permettant de créer des composants, notamment ceux de Angular Material.

Une fois ces packages installés, il est nécessaire de les importer dans notre application. Le plus simple, est de créer un fichier de module dédié aux imports de modules Angular Material et d'importer ce module dans le app.module.ts. Ce nouveau fichier, nous le mettons dans notre dossier shared, et nous y importons tous les modules Angular Material dont nous avons besoin dans notre application.

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import {
  MatIconModule,
  MatButtonModule,
  MatInputModule,
  MatToolbarModule,
  MatSidenavModule,
  MatCardModule
} from '@angular/material/';
import {FormsModule} from '@angular/forms';

@NgModule({
  imports: [
    BrowserAnimationsModule,
    FormsModule,
    CommonModule,
    MatIconModule,
    MatButtonModule,
    MatInputModule,
    MatToolbarModule,
    MatSidenavModule,
    MatCardModule
  ], exports: [
    BrowserAnimationsModule,
    FormsModule,
    CommonModule,
    MatIconModule,
    MatButtonModule,
    MatInputModule,
    MatToolbarModule,
    MatSidenavModule,
    MatCardModule
  ],
    declarations: []
  })

  export class MaterialModule { }
```

Figure 15 - Fichier d'imports de modules d'Angular Material dédiés

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';

//IMPORT HTTP CLIENT MODULE
import { HttpClientModule } from '@angular/common/http';

//Import material module
import { MaterialModule } from './shared/material.module'

@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    MaterialModule //Import
  ],
  providers: [],
  bootstrap: [AppComponent]
})

export class AppModule { }
```

Figure 16 — Import du module MaterialModule

Récupération des données

Maintenant que nous avons notre service (et qu'il est testé), nous allons pouvoir nous pencher sur l'exploitation du résultat et son affichage sur l'application. Nous avons créé une méthode **getCityWeather** qui va appeler notre service directement dans le composant de base de l'application **app.component.ts**. Comme indiqué précédemment, nous souscrivons à notre requête http grâce à **subscribe()**. Le résultat obtenu est stocké dans un objet (toujours typé) **weatherResult** que nous allons pouvoir exploiter dans la partie HTML.

```
import { Component } from '@angular/core';
import { WeatherService } from './shared/services/weather.service'
import { Weather } from './shared/models/weather'

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public weatherResult: Weather;
  public city: string;

  constructor(private weatherService: WeatherService) {
  }

  public getCityWeather() {
    this.weatherService.getCityWeather(this.city).subscribe(result => this.weatherRe
```

Figure 17 — Composant de base, avec une méthode permettant de déclencher la requête http

Création de notre application et exploitation des données

Pour développer l'application, nous pouvons la démarrer via la commande **ng serve** et visualiser l'application sur le navigateur directement.

Au niveau du fichier HTML app.component.html, nous allons remplacer le contenu existant pour définir une nouvelle structure à notre application. L'affichage du résultat de la requête http va être géré dans un autre composant, que nous pouvons d'ores et déjà générer dans notre dossier shared, via la commande **ng generate component [Nom du composant]**.

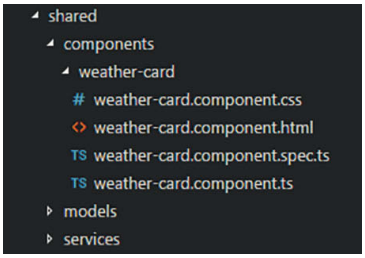


Figure 18 — Nouveau composant weather-card

Dans ce composant, le gros du travail se porte sur le fichier Html et CSS. Dans le fichier ts, nous devons uniquement indiquer que celui-ci prend en entrée un élément (le résultat de notre requête http).

```typescript
import { Component, OnInit, Input } from '@angular/core';
import { Weather } from '../../models/weather'

@Component({
  selector: 'app-weather-card',
  templateUrl: './weather-card.component.html',
  styleUrls: ['./weather-card.component.css']
})
export class WeatherCardComponent implements OnInit {
  @Input() weatherData: Weather; //Input component

  constructor() { }

  ngOnInit() {
  }

}
```

Figure 19 — Déclaration de l'entrée "weatherData" du composant weather-card

À ce stade, notre fichier app.component.html ressemble à ceci :

```html
<mat-sidenav-container class="app-container">
  <!-- Left content -->
  <mat-sidenav class="app-sidenav" mode="side" opened>
    <div class="app-icon">
      <mat-icon [inline]="true">filter_drama</mat-icon>
    </div>
    <div class="app-title">Weather App</div>
  </mat-sidenav>

  <!-- Right content -->
  <mat-sidenav-content>
    <!-- Top toolbar with the input -->
    <form #heroForm="ngForm" (ngSubmit)="getCityWeather()">
      <mat-form-field class="app-form-input">
        <input matInput [(ngModel)]="city" placeholder="Enter city name" name="cityName">
      </mat-form-field>
      <button mat-icon-button type="submit" class="btn btn-success">
        <mat-icon>search</mat-icon>
      </button>
    </form>

    <!-- Show result in a card -->
    <app-weather-card *ngIf="weatherResult" [weatherData]="weatherResult"></app-weather-card>

  </mat-sidenav-content>
</mat-sidenav-container>
```

Figure 20 — HTML du composant de base

Nous avons deux zones bien définies :

- **mat-sidenav (Ligne 3)** : une colonne d'affichage à gauche, permettant d'afficher le nom de l'application et un logo.
- **mat-sidenav-content (Ligne 11)** : le contenu de notre application, à droite. Dans cette partie, nous allons trouver un formulaire contenant un champ de saisie binder sur la propriété city déclarée dans notre **AppComponent**.

Nous obtenons dès lors l'affichage suivant (en ayant fait quelques ajouts de CSS) :

Figure 21 — Apparence de l'application

Dans notre code HTML dans le contenu de droite (Cf. figure 18 – Ligne 23), nous appelons le composant **app-weather-card**. Lorsque l'utilisateur saisit un nom de ville (exemple : Paris) et appui sur Entrée, la méthode dans **app.component.ts** est appelée. Le résultat de la requête http est stocké dans la variable **weatherResult**. Cette variable, nous allons la donner à notre composant **app-weather-card** pour afficher les résultats sous forme de petite carte.

```html
<mat-card class="weather-card">

  <!--Card header -->
  <mat-card-header class="weather-card-header">
    <div class="flex-initial">
      <div mat-card-avatar>
        <img [src]="weatherData.current.condition.icon">
      </div>
    </div>
    <span class="flex-auto"></span>
    <div class="flex-initial">
      <mat-card-title>{{weatherData.location.name}}</mat-card-title>
      <mat-card-subtitle>{{weatherData.location.region}}</mat-card-subtitle>
      <mat-card-subtitle>{{weatherData.location.country}}</mat-card-subtitle>
    </div>
  </mat-card-header>

  <!-- Card Content -->
  <mat-card-content>
   <span class="weather-card-content"> Humidity : {{weatherData.current.humidity}} % </span>
   <span class="weather-card-content"> Cloud cover : {{weatherData.current.cloud}} % </span>
   <span class="weather-card-content">  Wind Speed : {{weatherData.current.wind_kph}} km/h</span>
  </mat-card-content>

  <!-- Card footer -->
  <mat-card-footer class="weather-card-footer">
      <span class="flex-initial">{{weatherData.current.temp_c}} °C</span>
      <span class="flex-auto"></span>
      <span class="flex-initial">{{weatherData.current.condition.text}}</span>
  </mat-card-footer>

</mat-card>
```

Figure 22 - Code Html de app-weather-card, exploitant le résultat de la requête

Et avec un peu de CSS en plus, voici le résultat de l'affichage :

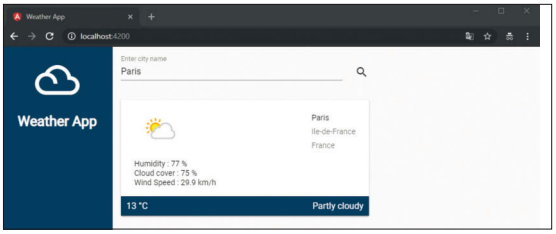


Figure 23 — Affichage de la météo de Paris

Et voilà, nous avons très simplement, depuis une API, réalisé une application Angular 6. Vous pouvez retrouver les sources ici : https://github.com/annecb/angular-demo-application

Conclusion

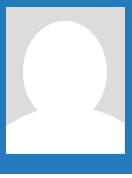
Nous avons pu voir au travers de cet exercice qu'il est très simple de développer une application Angular pour une API. Sans être un expert du design, il existe de très nombreux outils nous permettant de réaliser une interface très rapidement et simplement.

Nous nous sommes basés sur Angular, mais beaucoup d'autres frameworks permettent d'arriver au même résultat. À titre d'exemple, nous pouvons citer ReactJS. La grande différence entre les deux est que Angular est très complet et embarque directement tous les éléments nécessaires à la création d'une application, là où ReactJS ne propose que des composants et du templating et laisse libre cours au développeur pour amener les briques nécessaires au projet (Routing, http, etc.).

Quoi qu'il en soit, il n'y a pas de framework idéal. Le choix de l'un ou l'autre dépend entièrement du projet, du développeur et du langage avec lequel il a le plus de facilité.

À savoir qu'il est possible, pour des applications ayant été développées en AngularJS, d'effectuer une migration petit à petit vers Angular. Utile quand une totale interruption de service n'est pas envisageable le temps d'une migration. En effet, Angular offrant l'opportunité de faire cohabiter les deux versions sur une seule et même application, on parle alors d'Application hybride. Mais il s'agit d'un tout autre sujet.

Les API sont partout de nos jours, et sont au cœur même d'une application. Publiques ou non, elles sont là pour offrir à des développeurs une interface vers les fonctionnalités d'un programme. Il existe de très nombreux catalogues regroupant des API, vous permettant de vous y abonner pour les exploiter. C'est ce que Moskitos propose avec son portail Community. Cette externalisation et simplification du portail d'API permet aux métiers non techniques de l'entreprise, comme les chefs de projet ou les consultants, de gagner en autonomie sur leur utilisation des catalogues d'API et de connecteurs, afin de démocratiser toujours davantage l'utilisation des API dans l'entreprise. •



Virgile Caron
INVIVOO

Dorra Dhouib
Consultante Sénior Python
INVIVOO

Designer des APIs REST avec Flask-RESTPlus

niveau 200

Partie 1

Flask est un microservice web qui permet, entre autres, d'implémenter des API REST. Mais lorsqu'il s'agit de les documenter, visualiser, contrôler et valider les schémas de données d'entrées et de sorties, **Flask** atteint vite ses limites.

C'est ici qu'intervient son extension **Flask-RESTplus** qui - comme son nom l'indique - est dédiée au design des api REST.

Flask-RESTPlus prend en charge la création rapide d'API REST. Il encourage les meilleures pratiques avec une configuration minimale. Et fournis également une collection de décorateurs et d'outils pour décrire votre API et afficher correctement sa documentation (à l'aide de Swagger).

Le but de cet article que je déroulerai comme un tuto est d'implémenter une API REST avec **Flask-RESTPlus**. Cette api sera connectée à une base de données Mongodb, et nous permettra de récupérer, ajouter, modifier et supprimer des données.

Tout au long de ce tuto, nous verrons les points forts de **Flask-RESTPlus** :

- Structuration,
- Documentation,
- Validation des données d'entrées et des réponses.

Cette api permettra d'ajouter, récupérer et supprimer des données dans notre base Mongodb.

Ces données décriront des livres. Un livre est défini par :

- Titre : titre du livre,
- Auteur : auteur du livre,
- Langue : français ou anglais. 1

Configuration et installation

Pour le tuto de cet article, j'ai utilisé Python 3.

Nous avons besoin d'installer Flask et Flask-RESTPlus et de désigner notre api.

```
pip install Flask
pip install Flask-RestPlus
```

Nous installons également Pymongo pour gérer notre base Mongodb.

```
pip install pymongo
```

Création base mongodb

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")

mydb = myclient["tuto"]
```

Notre api effectuera des opérations sur cette base mydb.

Implémentation

Avant de passer à la pratique, listons les opérations qu'un utilisateur pourra effectuer sur notre base via notre api.

Il pourra :

1

- Récupérer la liste de tous les livres présents en base,
- Ajouter un nouveau livre,
- Récupérer un livre à partir de son titre,
- Supprimer un livre.

Il y a évidemment plusieurs autres opérations possibles, mais pour le reste de l'article, je me concentrerai sur ces fonctionnalités.

Nous commençons donc par créer une application Flask qui portera notre API REST.

```
from flask import Flask, request
from flask_restplus import Api, Resource

app = Flask(__name__)

api = Api(app=app, version='0.1', title='Books Api', description='', validate=True)
```

Nous passons maintenant à l'implémentation du contenu de notre api. N'ayez pas peur si vous ne comprenez pas tout de suite le code ci-dessous, les explications vont suivre.

```
@api.route("/books/")
class BooksList(Resource):
  def get(self):
    """
    returns a list of books
    """

    cursor = mydb.books.find({}, {"_id": 0})
    data = []
    for book in cursor:
      data.append(book)

    return {"response": data}

  def post(self):
    """
    Add a new book to the list
    """

    data = request.get_json()
    if not data:
      data = {"response": "ERROR"}
      return data, 404
    else:
      title = data.get('title')
```

```
        if title:
            if mydb.books.find_one({"title": title}):
                return {"response": "book already exists."}, 403
            else:
                mydb.insert(data)


@api.route("/books/<string:title>")
class Book(Resource):

    def put(self, title):
        """
        Edits a selected book
        """

        data = request.get_json()

        mydb.books.update({'title': title}, {'set': data})

    def delete(self, title):
        """
    delete a selected book
    """

        mydb.books.delete({'title': title})
```

Ressource

Première particularité de RESTPlus, ce sont les classes « **Resource** ». Elles permettent de regrouper les méthodes http par type de données.

Quand vous créez une api avec **Flask-RESTPlus**, toutes les méthodes http doivent être déclarées dans des classes **Resource**.

Et si vous avez besoin d'implémenter plusieurs méthodes GET pour votre API. Il vous suffit de créer

plusieurs classes **Resource** et de placer chaque méthode dans la classe ressource correspondante.

Le décorateur *@api.route()* est utilisé pour spécifier les URL qui seront associées à une ressource donnée.

Vous pouvez spécifier les paramètres de chemin à l'aide de crochets, comme ici :

@api.route ('/<int: id>')

Vous pouvez éventuellement spécifier le type de paramètre à l'aide du nom d'un convertisseur et de deux points. Les convertisseurs disponibles sont :

- **string:** (valeur par défaut)
- **path:**
- **int :**
- **float:**
- **uuid :**

Documentation swagger

Voyons à quoi ressemblera notre API. Nous lançons notre application Flask :

```
app.run(port= 8887, host= 'localhost')
```

D'autres options sont à paramétrer dans la fonction run, mais pour ce tuto je ne vais pas m'attarder dessus.

Nous allons sur le lien **http:/localhost :8888/** et nous obtenons l'interface Swagger suivante : 2

Cette interface vous permet de visualiser votre API :

- Titre, version, description,
- Différentes méthodes et leurs routes,
- Paramètre des méthodes et leurs types.

Elle vous permet également de tester vos méthodes.

Autre avantage, si vous avez ajouté une docstring dans une de vos méthodes, elle sera automatiquement affichée dans l'interface Swagger.

Namespace

Notre api contient uniquement quatre méthodes. Nous pouvons donc facilement retrouver la méthode que nous souhaitons tester dans l'interface Swagger.

Imaginons que dans le même api, nous voulons effectuer des opérations sur une base de livres, mais également une base de films. Il est alors plus clair et plus lisible de les séparer en deux sections différentes : une section pour les opérations sur les livres et une autre pour les opérations sur les films.

Ceci est facilement réalisable grâce aux namespaces.

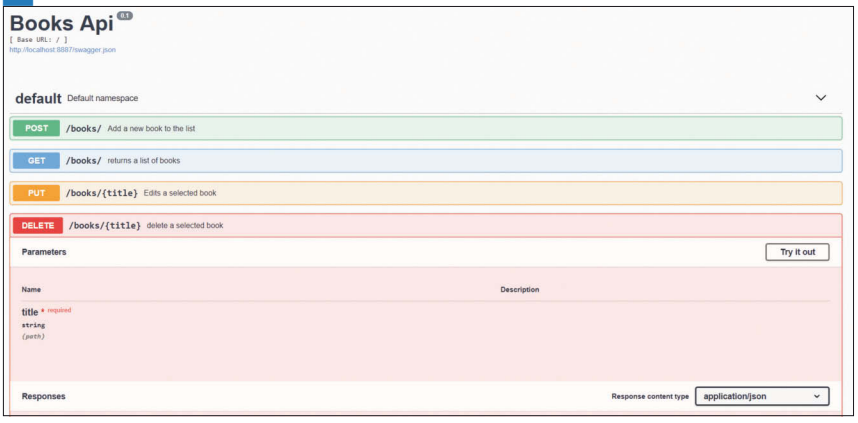
Les *namespaces*, autre option de **Flask-RESTPlus**, permettent d'organiser notre api en regroupant des ressources connexes sous une racine commune.

Chaque section est alors portée par son *namespace*.

On peut aussi ajouter une description à chaque *namespace* qui sera affichée dans l'interface Swagger.

```
app = Flask(__name__)
api = Api(app=app)

ns_books = api.namespace('books', description = "Books operations")
ns_movies = api.namespace('movies', description = "Movies operations")

@ns_books.route("/")
class BooksList(Resource):
    def get(self):
        """
        returns a list of books
        """

    def post(self):
        """
```

2



```
    Add a new book to the list
    """


@ns_books.route("/<string:title>")
class Book(Resource):

  def put(self, title):
    """
    Edits a selected book
    """


  def delete(self, title):
    """
  delete a selected book
  """


@ns_movies.route("/")
class MoviesList(Resource):
  def get(self):
    """
    returns a list of movies
    """

  def post(self):
    """
    Add a new movie to the list
    """


@ns_movies.route("/<string:title>")
class Movie(Resource):

  def put(self, title):
    """
    Edits a selected movie
    """


  def delete(self, title):
    """
  delete a selected movie
  """
```

Pour amener certaines ressources dans un espace de nom donné, il vous suffit de remplacer **@api** par **@ns_books**. La fonction **api.namespace()** crée un nouvel espace de nom avec un préfixe d'URL. Le champ de description sera affiché dans l'interface utilisateur Swagger pour décrire cet ensemble de méthodes. 3

Les namespaces permettent également de mieux organiser notre code : nous pouvons créer un fichier par namespace avec les classes **Resource** requises, et ensuite ajouter ces namespaces à notre api via la méthode add_namespace.

@api.response()

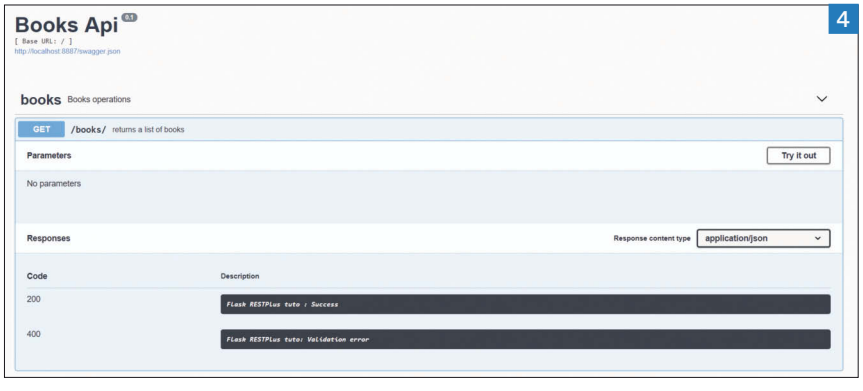
Vous pouvez utiliser le décorateur @api.response() pour répertorier les codes de statut HTTP que chaque méthode est censée renvoyer.

```
@ns_books.route("/")
class BooksList(Resource):
  @api.response(200, 'Flask RESTPlus tuto : Success')
  @api.response(400, 'Flask RESTPlus tuto: Validation error')
  def get(self):
    """
    returns a list of books
    """

    cursor = mydb.books.find({}, {"_id": 0})
    data = []
    for book in cursor:
      data.append(book)

    return {"response": data}, 200
```

4

Le mois prochain nous verrons la validation.

3



4

Retrouvez tous les codes sources de Programmez!
sur https://github.com/francoistonic/ et sur programmez.com

Christophe PICHAUD
Architecte Microsoft chez 'Modern Applications by Devoteam'
christophepichaud@hotmail.com
www.windowscpp.com

Reverse Engineering et analyse de virus

La programmation système est un domaine où les connaissances vont du système d'exploitation, aux API, aux outils et aux techniques de fonctionnement des logiciels. Le reverse engineering (RE), c'est le fait de découvrir, sans le code source, comment fonctionne une application ou un système à l'aide de ses connaissances sur le système.

niveau 200

Exemple d'applications

Le meilleur exemple d'utilisation du RE est l'analyse des virus. On veut savoir quels sont les effets nocifs que va provoquer un module. On le désassemble et on analyse son code assembleur pour savoir ce qu'il va faire. On va noter les appels systèmes et déduire les différentes techniques et algorithmes utilisés. Pour déduire les appels systèmes, on dispose aussi d'autres outils qui sont capables de nous donner des informations pertinentes. Avant, un bref rappel.

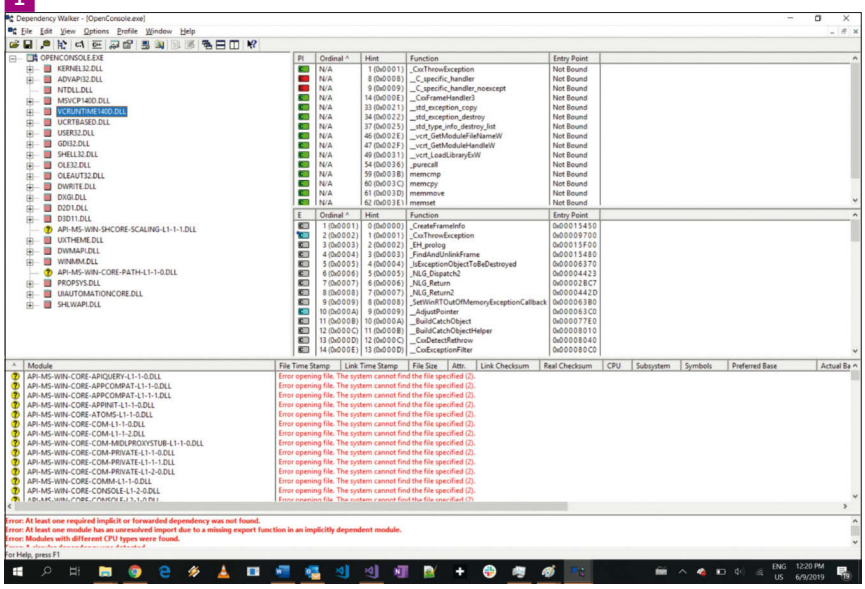
Construction d'un système d'exploitation

Simplifions les choses : l'OS est bâti avec une couche système et un mode user. Dans le mode kernel, il y a le noyau, les sous-systèmes et les drivers et dans le mode user, les applications. Sous Windows, les modules kernel sont linkés (liés à l'édition de liens) à kernel32.lib. En mode user, les modules sont linkés à kernel32.lib et user32.lib et généralement gdi32.lib.
Le système d'exploitation est fait en C donc s'y retrouve aussi le runtime du C à savoir MSVCRTxxx.dll. Donc un module user lambda se voit linké à ces 4 Dlls. C'est valable dans 90% des cas. La question est la suivante : comment puis-je le savoir?
Il y a plusieurs possibilités. 1ère option : télécharger les Windows Server Support Tools dans lequel il y a depends.exe. 2ème option : utiliser un environnement Visual Studio ou un Windows SDK qui fournit dumpbin.exe.

Depends.exe

Vous lancez depends.exe et vous lui faites un drag-and-drop d'un module dedans. Il va vous donner la liste des modules dont dépendent votre module. Pour chaque module, il y a aussi la liste des fonctions utilisées (importées). 1

1



Dumpbin.exe

Vous lancez dumipbin.exe /xxx et il vous donne le plein d'informations sur votre module.
Exemple : dumpbin /imports OpenConsole.exe
Cette commande vous affiche en texte, l'équivalent de depends.

Autre exemple :
dumpbin /dependents OpenConsole.exe
Cette commande donne la liste des DLLs qui seront automatiquement chargées.

```
Microsoft (R) COFF/PE Dumper Version 14.16.27030.1
Copyright (C) Microsoft Corporation.  All rights reserved.
Dump of file OpenConsole.exe
File Type: EXECUTABLE IMAGE

  Image has the following dependencies:

    KERNEL32.dll
    ADVAPI32.dll
    ntdll.dll
    MSVCP140D.dll
    VCRUNTIME140D.dll
    ucrtbased.dll
    USER32.dll
    GDI32.dll
    SHELL32.dll
    ole32.dll
    OLEAUT32.dll
    DWrite.dll
    dxgi.dll
    d2d1.dll
    d3d11.dll
    api-ms-win-shcore-scaling-l1-1-1.dll
    UxTheme.dll
    dwmapi.dll
    WINMM.dll
    api-ms-win-core-path-l1-1-0.dll
    PROPSYS.dll
    UIAutomationCore.DLL
    SHLWAPI.dll

  Summary
```

```
   7000 .data
   2000 .msvcjmc
  32000 .pdata
 143000 .rdata
   4000 .reloc
   9000 .rsrc
 296000 .text
```

A partir de là, on a le tableau des modules qui seront chargés. Cela permet de faire des hypothèses.

Environnement de simulation

Le meilleur moyen de visualiser ce que fait un module est de le lancer et d'observer, d'analyser et de prendre des « traces ». L'outil regmon de SysInternals permet d'enregister la liste des opérations faites sur la base de registres. C'est très intéressant car ces outils essaient pour la plupart de se rendre invisibles sur le système. Voir l'article de Janvier et Février sur les rootkits pour plus d'informations. Attention, pour tester ces modules sensibles, il faut se mettre en mode bac à sable dans une machine virtuelle sinon c'est la catastrophe… Je pense que vous vous en doutiez.
Souvent les scénarios des outils malveillants vont de la dissimulation vis à vis des anti-virus, le dépôt de backdoor ou de keylogger (donc enregistrement de drivers avec de faux certificats), l'ouverture de portes dans le firewall, jusqu'à l'enregistrement en autorun de certaines applications.
ProcessExplorer de SysInternals va nous permettre de voir, comme TaskManager, la liste des applications qui tournent sur notre OS et surtout le chemin depuis lequel ils sont lancés. Les logiciels malveillants se cachent dans les profiles, dans system32 ou dans programfiles avec des noms passe-partout.

L'art du Reverse Engineering

Le reverse engineering, c'est comme un travail de lutte anti-terroriste. On étudie comment il se distribue, comment il se lance, comment il enregistre ses sous modules et ce qu'il projette de faire dans différents scénarios. L'art se distingue, car il faut connaître les modules de Windows pour savoir quel type d'opération va être réalisé. Exemple : si la fonction CreateNamedPipes est utilisée, on va chercher à savoir quel est le nom du pipe. On va parier que le nom va être caché via l'ajout de $ en préfix. L'utilisation de ProcessExplorer de SysInternals peut les lister. On utilise des méthodes d'investigations. On se doit d'être patient car parfois, le module principal charge des modules annexes, des extensions, des plugins. En programmation système, le couple LoadLibrary/Get Proces Address est terrible. Ces sont les API systèmes qui permettent de charger un module et d'avoir accès à une fonction pour l'exécuter. L'avantage, c'est que cela est réalisé de façon dynamique et non statique. C'est invisible pour depends ou dumpbin.

Appels dynamiques

Imaginez que le module reçoive une instruction sur un port tcp/ip et se voie transférer un module avec un nom de fonction. Il enregistre le flow de data sur disque, le renomme en dll et appelle une fonction. Indétectable !
C'est comme ça que sont faits les virus modernes. Par étages.

Win32, l'API ultime

Pour maîtriser la conception d'un virus, il faut maîtriser le host, c'est à dire le système d'exploitation. Il en va de la connaissance des API systèmes. Dans le monde Windows, on appelle cela les API Win32. On y retrouve toutes les API fournies par l'OS. La documentation Microsoft de développement a été complètement refondue (une fois de plus?) et le lien valide est : https://docs.microsoft.com/en-us/windows/desktop/api/
Pour les DLL, voici le lien : **https://docs.microsoft.com/en-us/windows/desktop/Dlls/dynamic-link-library-functions**

Une DLL c'est quoi ?

Une Dynamic Loaded Library ou DLL est un module dynamique. Il se charge à la volée car un autre module en a besoin ; dans ce cas c'est une dépendance et l'OS la charge automatiquement car la liaison est statique. Sinon, le chargement est dynamique, c'est à dire qu'un code fait appelle à LoadLibrary pour charger explicitement une DLL.

```
HMODULE LoadLibraryA( LPCSTR lpLibFileName );
FARPROC GetProcAddress( HMODULE hModule, LPCSTR lpProcName );


LPCSTR est le type Windows pour const char *.
HMODULE est un handle système. FARPROC est un epointeur de fonction (void *)().

typedef void (WINAPI *PGNSI)(LPSYSTEM_INFO);

// Call GetNativeSystemInfo if supported or GetSystemInfo otherwise.
PGNSI pGNSI;
SYSTEM_INFO si;
ZeroMemory(&si, sizeof(SYSTEM_INFO));

pGNSI = (PGNSI) GetProcAddress(
   GetModuleHandle(TEXT("kernel32.dll")), "GetNativeSystemInfo");
if(NULL != pGNSI)
{
   pGNSI(&si);
}
```

Les virus

Les virus sont fabriqués en C/C++ car le C/C++ est le langage des systèmes d'exploitation. Après tout, un virus est un programme comme un autre. OK, il essaie de dissimuler les informations et d'être discret mais il n'y a rien d'illégal à écrire sur disque, faire des fichiers cachés, crypter les communications, enregistrer des drivers, etc. Ce qui est illégal, c'est d'essayer de pénétrer dans le système de quelqu'un d'autre et de faire du vol de données ou de l'utilisation à des fins illicites. Nuance !
Dans une entreprise, j'ai le droit de mettre un keylogger pour surveiller mes employés, oui ou non ? J'ai le doit de filtrer toutes les communications qui sortent sur la passerelle internet, oui ou non ? J'ai le droit de dissimuler des fichiers dans des endroits non surveillés, oui ou non ? J'ai le droit de stocker des données dans des JPEG, oui ou non ?
En tant que chercheur en informatique, vous avez droit à tout tant

que c'est de la recherche. Vous allez certainement installer plusieurs fois des VM à force de casser vos environnements… ou de rebooter à force de provoquer des blue screen si vous touchez au mode kernel:) !

Les débogueurs

Sous Windows, le must c'est WinDBG. Il permet de lancer les applications en pas à pas et c'est un outil gratuit disponible dans le Windows Sdk. Si les symboles de debogage sont installés, WinDBG va afficher le nom des API systèmes au lieu de leur offset dans la liste des fonctions.

Les désassembleurs

Il existe des applications qui transforment le code binaire en code assembleur. On peut découvrir des trucs extraordinaires. Exemple : au chargement du programme, on s'aperçoit que le programme « décrypte des pages de texte » et les transforme en pages qui peuvent s'exécuter c.a.d en code. Cela permet d'être invisible via à vis des anti-virus. Sous Windows le must est Hex-Rays IDA Pro. C'est un outil payant.

Conclusion

Un virus n'est qu'un programme qui fait des opérations à votre insu. Il vole vos données, les envoie via votre box internet à un serveur host ; il est contrôlé à distance pour recevoir des instructions ou des modules complémentaires. Le RE sur ces modules n'est qu'un moyen scientifique de connaître leur fonctionnement et de limiter leurs dégâts car nous n'avons pas le code source… •

Jean-Christophe QUETIN
Passionné depuis toujours par l'électronique et la programmation, je suis l'auteur du livre : Arduino - Apprivoisez l'électronique et le codage (Editions ENI) qui est sorti en 2018. Puis, j'ai eu envie de continuer à écrire, j'ai donc créé mon blog début 2019.
arduiblog.com - twitter.com/jcquetin
www.linkedin.com/in/jean-christophe-quetin-29682a164/

La pile patate : une batterie biodégradable en mode DIY !

niveau 100

Vous utilisez tous les jours des piles pour alimenter les appareils électriques, mais vous ne vous êtes jamais demandé comment ça marche ? Ça tombe bien, aujourd'hui nous allons en fabriquer une.

Au lieu de créer une pile au sens traditionnel du terme, nous allons utiliser des fruits et légumes. :-)
Pour réaliser ce montage, nous avons besoin de quelques pommes de terre, de fils à pinces crocodiles et de fines plaques constituées de deux métaux différents (idéalement du cuivre et du zinc). 1
Le premier test est très simple :

1 on fixe 2 lamelles métalliques (de 2 métaux différents) dans une pomme de terre : la plaque cuivre est le ⊕, la plaque de zinc est le ⊖
2 vérifiez que les lamelles soient assez proches, environ 2 cm, sans se toucher.
3 C'est tout !

Voilà notre pile est prête ! Si on mesure la tension avec un voltmètre, on obtiendra une tension de 0,8 / 0,9 volts. C'est mieux que rien !

Pourquoi ?

Il s'agit d'un phénomène chimique permettant un échange d'électrons (autrement dit de l'électricité) entre les métaux. Il est donc indispensable d'utiliser deux métaux différents (ne possédant pas le même nombre d'électrons). 2
Pour augmenter la tension, il suffit de rajouter des pommes de terre. Il faut les connecter en série en reliant la borne ⊕ de la 1ere pomme de terre à la borne ⊖ de la suivante, et ainsi de suite… 3
Eh miracle : les tensions s'additionnent :
1 pomme de terre = 0,8v
2 pommes de terre : 1,6v
3 pommes de terre : 2,4v
Etc.

4 pommes de terre pour 1 LED !

Vous allez me dire que ce n'est pas beaucoup. Vous n'avez pas alimenté une carte Arduino comme cela. Pour pouvoir allumer une LED, il faut 3V, l'équivalent de 4 pommes de terre. Mais comme l'intensité est faible, la luminosité de la LED sera faible. Dans une pile, la tension dépend des métaux utilisés, mais l'intensité dépend de la surface de métal en contact avec la pomme de terre.

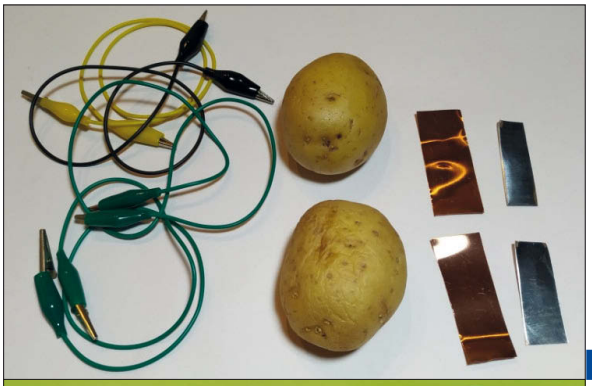
Petit rappel :

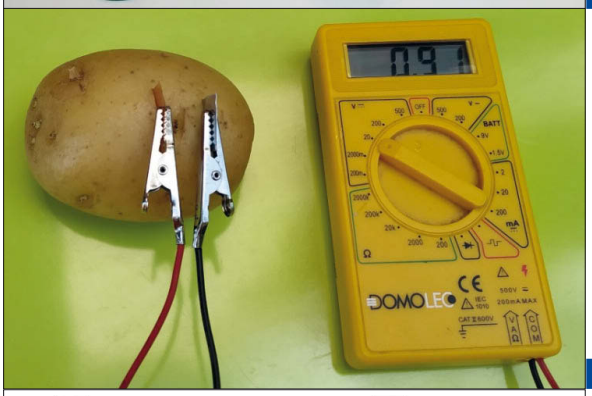
- La tension (mesurée en Volts) correspond à la différence de potentiel entre la borne ⊕ et la borne ⊖ (si l'on compare l'électricité à de l'eau, on pourrait dire que c'est la pression).
- L'intensité (mesurée en Ampères) correspond à la quantité d'électricité (on pourrait dire que c'est le diamètre du tuyau). 4

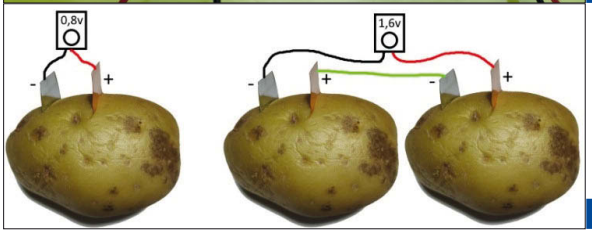
Pour augmenter l'intensité, il faut soit augmenter la taille des plaques de métal, soit doubler le nombre de piles connectées en parallèle (relier les bornes ⊕ ensemble et les bornes ⊖ ensemble). 5
Avec 8 patates, la LED éclaire un peu plus…

Si vous n'aimez pas la pomme de terre, vous pouvez la remplacer par du citron. En réalité tout fruit et légume un peu acide fera l'affaire. Vous pouvez faire la même chose avec un liquide un peu acide…

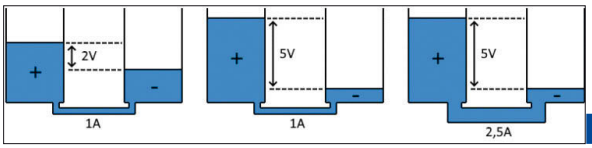
Attention : ne mangez pas les fruits et légumes après les avoir utilisés comme piles !


1

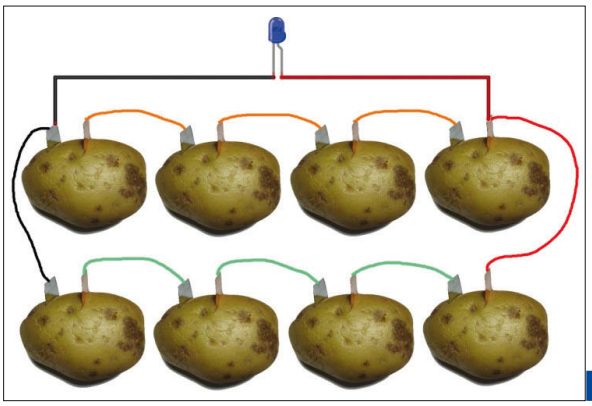

2



3



4


5

Cloud & développeur : Impossible de se passer du cloud

Partie 2

Dans le 230, nous avions abordé plusieurs thématiques pour comprendre les enjeux du cloud computing pour le développeur : au-delà du buzz, l'impact du cloud sur le code (impact ou non impact), comment moderniser une app, la notion de cloud native et enfin un premier aperçu du serverless.

Ce mois-ci, nous poursuivons notre dossier autour du serverless, de l'observabilité appliquée au serverless et enfin du replatforming d'un environnement mainframe / cobol sur un cloud privé (en attendant de migrer sur des conteneurs et du cloud public ?).

Bonne lecture

Jean-Baptiste Bron
Architect | Presales
www.capgemini.com

RANGERS

Cloud & mainframe / cobol : le replatforming est possible mais difficile

L'informatique est toujours tiraillée entre deux horizons : « oui faisons des projets innovants avec toutes les dernières technologies ! » et le fameux « bon, y'a un peu de code (= des tonnes) legacy vieux de 15 ans (= 30 ans) tu verras c'est une question de quelques jours (= disons plutôt quelques années et sans docs) ». Bref : que faire des applications qui peuvent être en production depuis 40 ans sans que personne n'ose y toucher… Au point que trois mots annoncent le cauchemar de beaucoup de développeurs juniors : mainframe, z/OS, cobol !

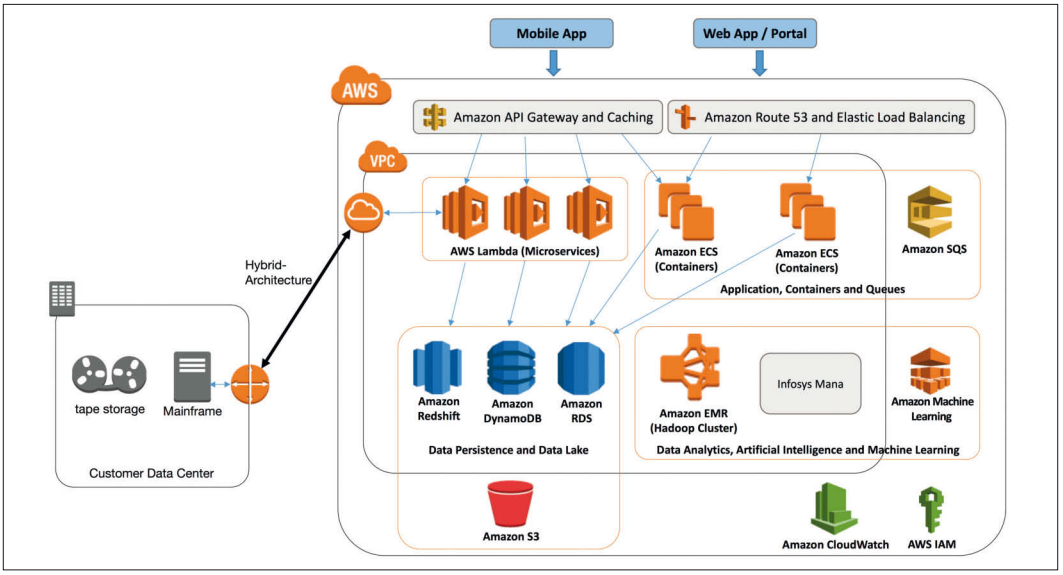
Il y a des projets de transformations (nous y reviendrons) qui méritent notre respect : des apps qui fonctionnent depuis 45 ans sans recompilation, toutes écrites en Cobol, 120 000 mips de puissances mainframe à basculer (là ça calme), +100 personnes impliquées, 1 projet qui s'étale sur 5 ans ! Actuellement, les équipes mènent le POC sur une durée de +6 mois après plusieurs mois de préparation. Dans ce projet hors nom de replatforming, rien n'a été simple.

Un replatforming d'envergure ou comment réconcilier les Ops et les Dévs

Comment définir le replatforming ? L'idée est relativement simple : il s'agit de migrer les applications, les codes, les données sur de nouveaux environnements plus modernes et de pouvoir en tirer parti. Ce dernier point est important ; un replatforming sans optimisation ni réarchitecture sera « juste » une migration sur une nouvelle infrastructure mais avec un périmètre fonctionnel très proche de l'original.

Le replatforming permet donc, théoriquement, de changer l'infrastructure d'exécution (= infra physique et infra logique) et de moderniser le code, l'application ainsi que l'environnement d'exécution (dans la limite du projet défini).

Dans le cadre de cet important projet bancaire, le replatforming vise à :



Il existe plusieurs approches de replatforming et chaque fournisseur aura sa propre approche. Le replatforming sur environnement cloud est celui qui est en vogue.

- Réduire la dépendance au mainframe et donc réduire le coût de ce dernier ;
- Déployer une nouvelle infrastructure logicielle plus moderne ;
- Moderniser les applications historiques (quand c'est possible).

Dans le replatforming, la réduction des coûts de l'informatique historique est une des priorités mais attention à ne pas être aveuglé par cette seule idée. Même si ce projet est hautement stratégique pour les banques concernées, cela n'a pas empêché de lancer un important recrutement, des centaines de personnes, avec des compétences mainframe / cobol. Eh oui ! Il faut continuer à faire vivre le mainframe et de nombreuses applications…

Dans ce projet, le replatforming est un défi car il est impératif que le mainframe continue à fonctionner et à exécuter les apps legacy, tout en migrant ces apps sur la nouvelle infrastructure. Il est impératif que les modifications se fassent sur les deux infrastructures pour éviter tout problème. Ce qui est loin d'être simple. D'autre part, durant la durée du projet, les réglementations continuent d'évoluer alors que le bancaire est déjà un environnement extrêmement réglementé ; ceci rajoute une couche de complexité supplémentaire.

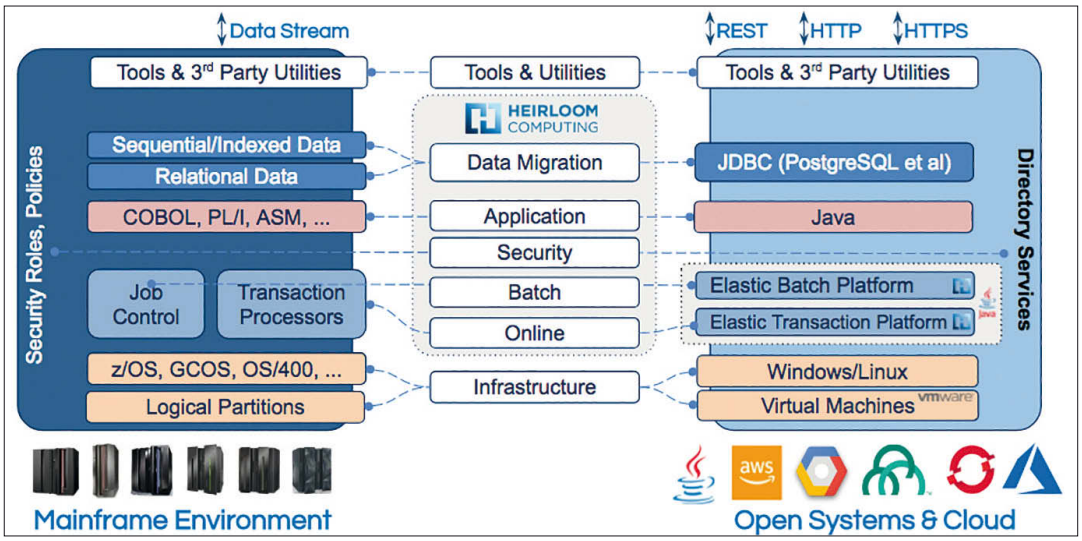
Ce qu'il faut bien comprendre dans ce type de replatforming, qui va bien plus loin que le classique web-to-host des années 2000 et remis au goût du jour, c'est la complexité des applications et de l'environnement mainframe.

Une des difficultés de ce projet est l'architecture des infrastructures. Dans le legacy, le mainframe est une plateforme monolithique : tout tourne dessus. Dans la nouvelle approche moderne, il s'agit d'un environnement distribué. Et c'est là que les problèmes commencent !

Comment ré-architecturer des workloads, des applications par définition monolithiques ?

Comment passer d'une infrastructure monolithique sur 1 machine à x machines ?

Un exemple parmi d'autres illustre ce projet : aujourd'hui le mainframe exécute jusqu'à 300 millions de requêtes SQL par jour ! Le mainframe, qui est tout de même une machine particulièrement robuste, encaisse sans problème mais quid de la nouvelle infrastructure ? Car qui dit environnement distribué dit aussi réseau et



Autre solution chez Heirloom Computing.

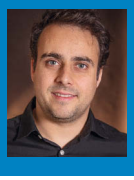
sous-réseau, latence entre les machines et les services. Bref, comment éviter cette latence qui n'existait pas, et comment supporter une telle quantité de requêtes sans risquer un overflow ou tout simplement un effondrement des performances ?
Des effets de bords sont apparus durant la migration des premières applications (dans le POC), notamment à cause des accès aux bases de données qu'il a fallu redéfinir (une DAL mainframe / zOS n'est pas identique à une DAL cloud par exemple). Ceci avec la nouvelle architecture réseau définie. Il faut donc mettre en place du caching et du routage pour minimiser le réseau.
L'autre difficulté est de cartographier l'ensemble des assets logiciels et de comprendre les interactions entre eux : une application peut avoir jusqu'à +40 interactions avec d'autres applications et surtout plusieurs apps peuvent modifier le même jeu de données ! ce qui complexifie d'autant le replatforming. Car dans ce type de projet : on doit garder, a minima, le même comportement, le même fonctionnement. Sur le papier c'est facile, dans la vraie vie, eh bien c'est moins facile…

La nouvelle infrastructure

Pour la nouvelle plateforme, le choix s'est porté sur un cloud privé et non un cloud public. Une des raisons est la contrainte légale et réglementaire propre au système bancaire. Pour éviter de multiplier les latences et les sous-réseaux, un châssis unique a été monté (baie) contenant x serveurs. Les moteurs d'exécutions (notamment bases de données) sont déployés dessus. Les équipes ont opté pour une approche n-tiers assez classiques avec une architecture dite stateless. La partie cloud privé fonctionne sur VMware et Red Hat Linux. Pourquoi une approche par VM et non conteneurs ou véritable cloud public déployée localement comme on peut l'avoir avec Azure Stack, Cloud Foundry, OpenShift ? Le client a imposé une solution qui peut fonctionner aussi en conteneurs mais les applications n'en auraient pas tiré parti sans une profonde réécriture / réarchitecture. Le conteneur n'est pas retenu pour le moment. Et le choix a été fait de ne pas mélanger les différentes architectures comme on peut le faire en conteneur et notamment en infra as code.
Donc, les VM permettent de déployer les workloads legacy (apps + données + services mainframe), les moteurs d'exécution, etc. Mais les VM ne possèdent aucune configuration en interne. Tout est déporté dans une base Redis contenant les configurations. Un des objectifs est de pouvoir gérer de manière centralisée les VM et les paramètres. Pour les bases de données, le choix s'est porté pour du in-memory. Avantage : on accélère les traitements car tout est en mémoire vive.
Côté legacy et déploiement de celui-ci dans la nouvelle architecture, l'éditeur MicroFocus a été retenu pour toute la partie moteur d'exécution et outillage Cobol. Le compilateur Cobol est celui de MicroFocus. Il ne s'agit pas seulement de prendre du code et de le déployer tel quel. Il est nécessaire de le (re)compiler en apportant une certaine optimisation x86 pour tirer parti des nouvelles machines et des technologies modernes. Et grâce à cet environnement, il est possible d'utiliser des API inconnues sur mainframe. Quid de la réécriture du code Cobol ? Aujourd'hui, il est minime car le but n'est pas de réécrire dans un nouveau langage. Mais le nouvel environnement oblige effectivement à ajuster, voire à réécrire certains codes Cobol.
Il faut dire que certains codes n'avaient pas été recompilé depuis 1972 ! Et personne ne savait si le code allait ou non pouvoir être recompilé… Quand on a ce genre de legacy, les équipes naviguent à vue.

Là, on comprend la nécessité de se parler !

Vu l'ampleur du projet et la criticité des applications et données, le DevOps et l'agilité prennent tout leur sens. Une approche agile a été mise en place pour pouvoir faire des itérations régulières : 2 semaines et 4 semaines en phase de POC. Ce qui a été un changement radical pour la direction métier qui n'est pas habituée à ce mode de fonctionnement. Et il fallait aussi que les développeurs comprennent l'infrastructure matérielle mainframe. Car même si le développeur travaille dessus, il ne voit rien.
Le défi n'a pas été tant l'infrastructure matérielle (x86) que l'infrastructure logicielle / applicative à déployer et à provisionner. L'approche DevOps prend alors tout son sens. Toutes les équipes doivent se parler.
La communication inter-équipes / département est cruciale pour la réussite d'un tel projet de replatforming. C'est un facteur de succès (ou d'échec).
Aujourd'hui, le projet est dans sa phase de prototypage réel pour éprouver les choix. Cette phase se déroulera jusqu'à fin de l'année. Mais le projet peut être arrêté à tout moment.
Loin des buzz et des grandes annonces, ce projet de replatforming ne provoque pas de big bang. Pour compléter le projet, une suite ELK sera déployée. Un ELK est un environnement d'analyses de logs permettant de créer un véritable cockpit d'administration où les ops trouveront l'ensemble des données importantes des infrastructures et des applications. Les logs et sysout seront stockés dans un service Big Data. •



Steve Houël
Architecte Cloud & DevOps chez Ippon Technologies, évangéliste Serverless et contributeur sur des projets Open Source comme JHipster et daSWAG. Il est fier d'être un #GeekEnthusiast.

IPPON
Discovery to Delivery

L'observabilité appliquée au Serverless

Partie 1

Aujourd'hui, de nouveaux services font parler d'eux dans le monde de l'IT et principalement lorsque l'on parle de fournisseur de services Cloud; c'est le Serverless. Nous nous attaquerons à un sujet bien précis : l'observabilité. Nous verrons ensemble les impacts sur votre conception lorsque l'on applique ces principes à ce type d'architecture Serverless.

niveau 100

L'observabilité est un terme qui possède de nombreuses définitions selon les personnes et les profils. Pour faire simple, cela vous permettra, au travers de ses 3 piliers, de comprendre et d'avoir une meilleure visibilité de l'état interne de votre système.

Si nous partons des vérités suivantes:

- votre système n'est jamais totalement sain,
- un système distribué est imprévisible,
- l'échec doit être pris en compte dans chacune de nos phases de développements ,de la conception à l'exploitation en passant par les phases de tests et de déploiement,
- votre capacité d'identifier un problème (debug) est un facteur clé de réussite pour la maintenance et l'évolution de votre système.

Vous obtiendrez alors une définition assez représentative de ce que l'observabilité doit apporter et les problématiques qu'elle doit permettre de résoudre.

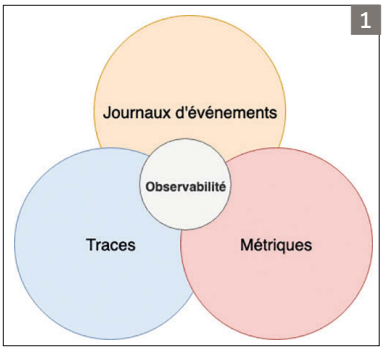
Étudions plus en détail ces 3 piliers : 1

Si vous voulez en savoir plus sur la définition de l'observabilité, je vous invite à regarder le talk de Charity Majors (https://www.youtube.com/watch?v=1wjovFSCGhE) ou de lire l'article de Cindy Sridharan (https://medium.com/@copyconstruct/monitoring-and-observability-8417d1952e1c) sur le sujet.

Les journaux d'évènements (logs)

Un journal des évènements est un enregistrement immuable et horodaté d'évènements qui se sont produits au fil du temps au sein de votre application. Ils se présentent généralement sous différents formats (texte en clair, structurés via l'utilisation du JSON, binaire).

Les données des journaux d'évènements ne sont pas utilisées exclusivement pour de l'évaluation de performance ou dans le cadre d'un débogage. Elles peuvent aussi faire l'objet d'une source d'informations pour vos analyses business comme de la veille stratégique ou pour connaître l'utilisabilité de votre application.



1

Les métriques

Les métriques sont une représentation numérique des données mesurées sur des intervalles de temps. Elles peuvent exploiter la puissance de la modélisation mathématique et de la prédiction pour obtenir des informations sur le comportement d'un système au fil du temps, dans le présent et le futur.

Étant donné que les nombres sont optimisés pour le stockage, le traitement, la compression et la récupération, ces données peuvent être conservées plus longtemps et peuvent être exploitées via des requêtes spécialisées (eg. médianes, moyennes, quantiles, dérivées). Cela les rend aussi parfaitement adaptées à la création de tableaux de bord reflétant les tendances historiques et peuvent être stockées dans des bases de données dédiées de type Timeseries. Nous voyons même l'émergence d'algorithmes de type prédictif pour prédire leur comportement.

Avantage des métriques par rapport aux journaux d'événements

Globalement, le principal avantage de la surveillance basée sur des métriques par rapport aux journaux est que, contrairement à la génération et au stockage de journaux, le transfert et le stockage de métriques entraînent une surcharge constante. Contrairement aux journaux, le coût des métriques n'augmente pas selon le trafic utilisateur ou à toute autre activité du système susceptible d'entraîner une nette augmentation des données.

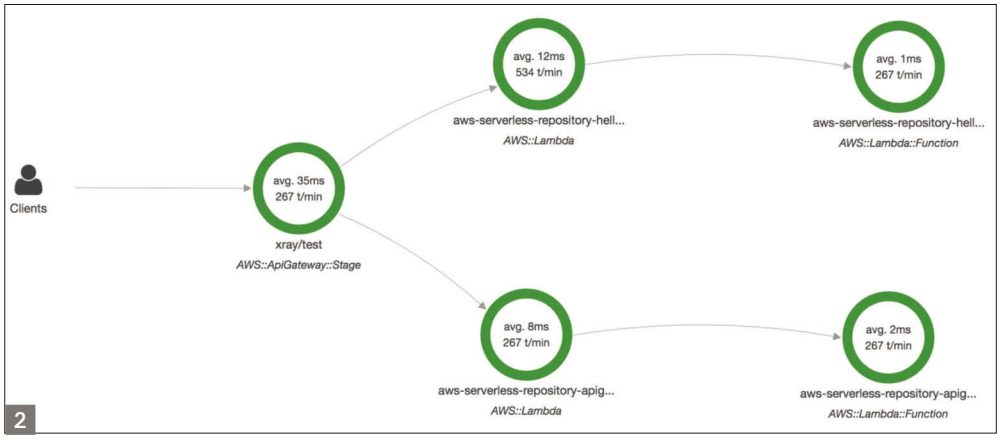
Ainsi avec les métriques, même si votre système supporte une charge croissante, cela n'entraînera pas d'augmentation de la complexité du traitement, de la vitesse de visualisation et des coûts opérationnels (license, stockage, ...), contrairement aux journaux.

Les métriques sont également mieux adaptées pour déclencher des alertes, car exécuter des requêtes sur une base de données Timeseries en mémoire est beaucoup plus efficace et plus fiable, que d'exécuter une requête sur un système distribué comme Elasticsearch, puis d'agréger les résultats avant de décider si une alerte doit être déclenchée ou non. D'autant plus, si l'on se base sur les journaux d'évènements comme source de métriques systèmes, nous pourrions avoir des erreurs d'interprétations, du simple fait que les valeurs sont agrégées sous forme de moyenne et peuvent ainsi masquer des évènements.

Les traces

Une trace est définie comme la représentation d'une série d'évènements distribués, liés de manière causale, qui permet de suivre le flux de requêtes de bout en bout sur un système.

Une seule trace peut fournir une visibilité à la fois sur le chemin parcouru par une requête ainsi que sur la structure d'une requête. Ce chemin permet aux développeurs et aux opérationnels de comprendre les différents services impliqués et d'identifier possiblement des points de blocage. 2



2

Lorsqu'une requête commence, un ID unique global lui est attribué que l'on nomme souvent ID de corrélation. Celui-ci est ensuite propagé dans tout le chemin de la requête afin que chaque instrumentation puisse insérer ou enrichir les informations avant de transmettre l'ID au saut suivant. Chacun des sauts enregistrera son évènement et permettra par la suite de reconstituer le chemin global de la requête à partir de l'identifiant fourni sur l'ensemble des messages.

Nos nouveaux challenges avec le Serverless

Avec les technologies Serverless comme AWS Lambda, nous sommes confrontés à un certain nombre de nouveaux défis pour appliquer ces principes, vu la jeunesse de la technologie et l'inadéquation de certains de nos outils historiques. Le principal impact de ces services "sans serveur" est justement l'impossibilité d'accès à notre infrastructure sous-jacente et ainsi notre incapacité à déployer notre agent de supervision. Nous ne pourrons plus utiliser nos briques standard pour récupérer, compresser et envoyer nos informations sur notre système d'observabilité. Dorénavant ces fonctionnalités seront incluses directement au sein de chacune de nos fonctions et ne pourront être exploitées qu'au travers des services dédiés fournis par le fournisseur de services Cloud. Un autre aspect à prendre en compte est justement cet aspect d'exécution unitaire. Sur un système traditionnel, nous devions gérer une quantité d'évènements en lien avec le volume de requêtes que ce système (instance) pouvait traiter. Aujourd'hui, vu que nous avons une exécution par requête, cette gestion de la concurrence sera nativement embarquée et gérée par le service lui-même. Nous n'aurons plus à nous soucier de ce type de problématique et c'est une bonne nouvelle. En revanche, vous devrez garder en tête que cela risque d'impacter votre système d'observabilité. Au lieu d'avoir un ensemble d'évènements envoyé par paquets, nous aurons des événements unitaires envoyés par chacune de nos fonctions et cette nouvelle charge devra être traitée par votre système et engendrera de nouveaux coûts, ou du moins une réévaluation de votre infrastructure.

La suite de cet article dans le prochain numéro.

Serverless : de quoi parle-t-on ?

Laurent Grangeau

RANGERS

niveau 100

Depuis plusieurs années maintenant, de plus en plus de blog-posts, de conférences, mais aussi de cloud providers eux-mêmes, parlent de Serverless Computing.

La promesse du cloud est que les entreprises ne vont plus passer de temps à acheter du matériel, créer des datacenters et les maintenir à jour, aux dernières évolutions technologiques et les alimenter en énergie renouvelable. Tout ceci est pris en compte par le Cloud provider, qui bénéficie des économies d'échelle en achetant ses serveurs par palettes entières. Les entreprises n'ont plus qu'à commander des serveurs et environnements chez eux.

Mais la maintenance de ces serveurs est toujours présente. Les équipes passent encore énormément de temps à maintenir à jour les serveurs, les faire évoluer et s'assurer que ceux-ci délivrent le service en continu, passant beaucoup de temps en astreintes.

Et c'est là qu'est apparu le Serverless. La promesse ? Se concentrer sur le métier et ne plus s'occuper de la plomberie.

Mais qu'est exactement le Serverless et quelles sont les offres chez les différents Cloud providers ? C'est ce que nous allons essayer de voir dans ce dossier consacré au Serverless.

Qu'est-ce que le Serverless ?

Le marketing faisant, quand le mot Serverless est évoqué, très souvent ce mot est associé à Function-as-a-Service. Oui ! Mais pas que…

Le Serverless est un concept d'architecture où le Cloud provider s'occupe de gérer et d'allouer dynamiquement les ressources sous-jacentes. L'équipe produit n'a donc plus à se soucier de maintenir à jour une liste de serveurs, et de la faire grossir au fur et à mesure des sollicitations.

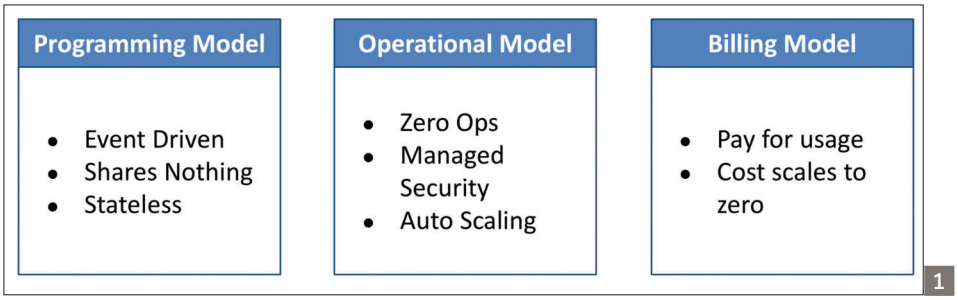
Partant de cette définition, le Serverless peut effectivement être du Function-as-a-Service, mais aussi ce qu'on appelle du Backend-as-a-Service.

Backend-as-a-Service

Le Backend-as-a-Service regroupe tout ce dont a besoin une application pour fonctionner.

Cela peut être des bus de messages, de l'authentification, du stockage, des bases de données, etc.

De cette manière, le développeur peut se concentrer à écrire uniquement du code et faire appel à ce backend via des APIs.



1

De plus en plus de Cloud providers proposent maintenant des services managés comme des bases NoSQL, des bus de messages, mais aussi des bases de données relationnelles qui s'adaptent automatiquement selon la charge.

Function-as-a-Service

Le Function-as-a-Service est le service le plus connu du Serverless car c'est grâce à lui que le terme a été popularisé. Le FaaS apporte plusieurs avantages, mais aussi plusieurs contraintes.

Tout d'abord, comme précisé juste avant, le FaaS est éphémère et ne dure que le temps de la requête qui l'invoque, ce qui induit des réductions de coûts car un service long comme un serveur web n'a pas besoin de rester provisionné en attendant une requête.

Le modèle financier change aussi. Au lieu de payer au mois pour un service qui tourne 24/24h, le billing se fait à l'invocation de la fonction. Tous les cloud providers offrent aujourd'hui a minima le premier million d'appels gratuit, certains cloud providers offrant jusqu'à deux millions d'appels gratuits !

Evidemment, tout n'est pas aussi rose et il y a quelques inconvénients. Tout d'abord, une fonction ne peut pas gérer d'état, c'est-à-dire pas de cache dans une fonction, ou de contexte de données. Il faut donc en tenir compte lors de l'architecture, et passer par exemple par des mécanismes externes.

Ensuite, comme l'infrastructure sous-jacente peut s'arrêter s'il n'y a aucune sollicitation, le redémarrage peut être long. C'est ce qu'on appelle le "cold start", et il faut en tenir compte dans le design de l'application.

Enfin, la contrainte la plus importante est le "vendor lock-in", ou la capacité d'être dépendant du cloud provider. En effet, chaque cloud provider a sa propre implémentation et un code écrit pour AWS ne sera pas transposable facilement vers GCP. 1

Le Serverless est alors une composition de tous ces services, que ce soit le FaaS, ou le BaaS, qui discutent entre eux via des APIs. De plus, l'une des principales caractéristiques du Serverless est que toutes les interactions entre les composants sont pilotées par des évènements. Le plus souvent ces évènements sont des messages JSON, mais nous verrons par la suite que cela peut aussi être d'autres types de message.

Pour résumer, le Serverless n'est pas une nouvelle offre de cloud computing, mais un concept d'architecture, où le design de l'application tire parti de services managés offerts par le cloud provider, où la communication entre composants passe par des APIs et où les interactions entre ces composants sont pilotées par évènements.

Serverless vs PaaS ?

Avec tout ce que je vous ai raconté, vous êtes en droit de me demander en quoi le Serverless se différencie du PaaS. Eh oui! Car le PaaS apportait déjà cette abstraction des serveurs, et les cloud providers géraient déjà toute l'infrastructure de ce modèle PaaS.

Effectivement, le Serverless se rapproche énormément du PaaS, dans le sens où tout est géré par le cloud provider. Mais le Serverless apporte d'autres bénéfices que le PaaS n'offre pas.

L'une des caractéristiques du Serverless est le Scale-to-zero, ou la capacité de détruire l'environnement si celui-ci n'est pas sollicité. Cela induit une réduction de coût car vous n'êtes, dans ce cas-là, facturé que si l'environnement est sollicité.

L'infrastructure s'adapte aussi selon la charge, à la hausse si la demande augmente, mais aussi à la

baisse, avec une possibilité d'éteindre les environnements si ceux-ci ne sont pas sollicités.
C'est là la plus grosse différence entre le Serverless et le PaaS. Le Serverless force aussi l'utilisation d'architecture microservices, car le découpage entre les composants est beaucoup plus fin qu'avec l'utilisation du PaaS.
Enfin, comme le dit Adrien Cockcroft : *"Si votre PaaS peut démarrer efficacement des instances en 20 ms qui s'exécutent pendant une demi-seconde, appelez-le Serverless."*

Serverless vs CaaS ?

De plus en plus d'entreprises basent leur stratégie IT sur les containers, et notamment Docker et Kubernetes. Dans cette approche, il existe aussi une équipe qui met à disposition des images de containers contenant plusieurs frameworks ou middlewares utilisés par l'entreprise. De plus, Kubernetes faisant office d'orchestrateur, il est à sa charge de démarrer d'autres containers par rapport à la demande, et de les éteindre une fois le pic de charge passé.
Dans ce modèle, le CaaS s'approche aussi de la définition du Serverless. Mais il subit les désavantages des autres modèles. Une équipe est toujours dédiée au maintien des images, au patching de ces images, à la correction de failles de sécurité, etc.
De plus, les plateformes d'orchestration coûtent de l'argent, même si elles ne sont pas sollicitées car elles reposent sur des modèles IaaS.

Les principes de l'architecture Serverless

Maintenant que nous avons défini ce qu'est le Serverless, et comment celui-ci se distingue d'autres modèles comme le CaaS ou le PaaS, nous allons voir comment architecturer et implémenter le Serverless dans une application.
Traditionnellement, les applications sont découpées en couches (tier en anglais) qui correspondent chacune à une responsabilité : le front, le serveur d'application et la base de données. Ce découpage a amené à transformer les organisations afin de gérer ce genre d'approche, en ayant une façon de penser et un découpage technique qui reflètent ce besoin.
L'apparition des microservices a bousculé les choses, chaque microservice ayant son propre datastore, et pouvant s'adapter de manière horizontale. Les entreprises ont donc eu à se mettre au diapason afin de pouvoir avoir des équipes sachant gérer ces problématiques.
Avec les architectures Serverless, l'approche est identique de celle des microservices. Cependant, contrairement aux microservices où le découpage n'intervient que peu, il est nécessaire de changer la façon de penser lorsque l'équipe imagine une application Serverless.
Toutes les interactions se font par évènements et il est nécessaire de savoir capter ces évènements, de générer d'autres évènements basés sur ces interactions et de les traiter. 2

Les modèles d'invocations

Différents modèles d'invocations sont possibles lorsqu'il s'agit d'implémenter des architectures Serverless. 3
Comme cité précédemment dans cet article, les architectures Serverless sont pilotées par les évènements. Ces évènements peuvent venir de plusieurs sources différentes :

- La première source, et la plus évidente est l'appel HTTP. Généralement, celui-ci provient d'une HTTP Gateway qui appelle la fonction et qui attend son retour. C'est un appel bloquant.
- La deuxième source est l'appel asynchrone. Celui-ci intervient lorsqu'il y a plusieurs centaines de messages à traiter et que les interactions ne doivent pas être bloquantes. L'utilisation d'un bus est alors nécessaire, sans garantie d'ordre d'arrivée des messages.
- La troisième source est utilisée lorsque l'ordre des messages est primordial, ou lorsqu'un stream est utilisé.
- Enfin, la quatrième source intervient lors de l'utilisation de batchs de processing qui doivent être parallélisés afin d'avoir la meilleure performance possible.

Les différents cas d'utilisations

Avec toutes ces invocations possibles, le Serverless commence à avoir des applications concrètes, et des cas d'utilisation tout trouvés. Cela peut être par exemple :

- Traitement de stream de data ;
- Analyse des messages IoT ;
- Modélisation de processus d'ETL (Extract, Transform, Load) qui requiert beaucoup de traitement dans un temps court ;
- Intégration avec des chatbots pour traitement des informations (Cognitive) ;
- Traitements batch ou cron ;
- Intégration continue ;
- Site web.

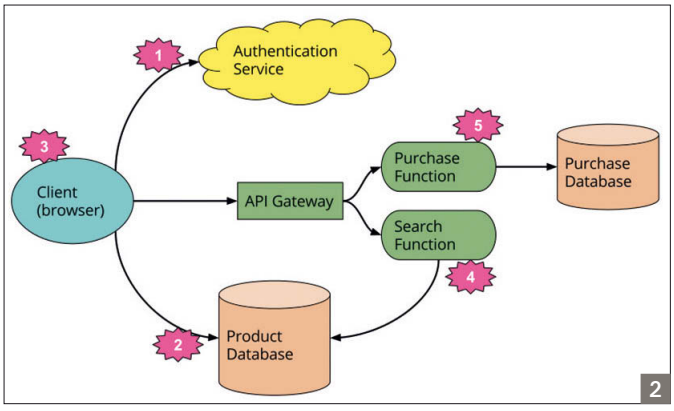
La liste n'est pas exhaustive, mais elle donne un certain nombre de cas d'utilisations les plus fréquents. De plus en plus d'entreprises utilisent les capacités des architectures Serverless afin d'accélérer leur Time-to-Market, de réduire les coûts et donc de se concentrer sur les développements à valeur ajoutée.

Les questions à se poser

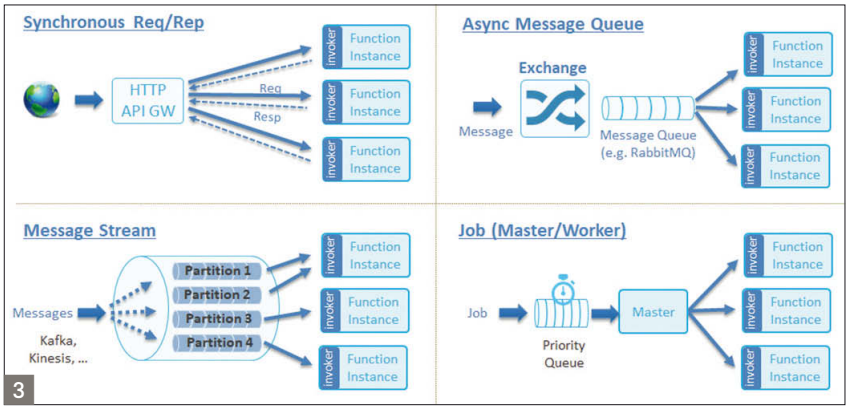
Les offres des CSP

Maintenant que nous connaissons parfaitement le Serverless, intéressons-nous aux différentes offres managées des cloud providers.
Les offres peuvent être découpées en plusieurs catégories, chacune offrant des services associés :

- Compute ;
- API ;
- Storage ;
- Datastore ;
- Messaging ;
- Orchestration ;
- Analytics ;



2



3

2

	Openwhisk	Fission	Kubeless	OpenFaaS	Knative
Création	Février 2016	Août 2016	Novembre 2016	Décembre 2016	Juillet 2018
Vendor	IBM / Apache Foundation	Platform 9	Bitnami	VMWare	Google
Repository	https://github.com/apache/incubator-openwhisk	https://github.com/fission/fission	https://github.com/kubeless/kubeless	https://github.com/openfaas/faas	https://github.com/knative/serving

- AI/Cognitive ;
- Service Mesh.

En plus de ces catégories, il est nécessaire de pouvoir déployer toute cette architecture et l'opérer, il faut alors ajouter à celles-ci quatre catégories supplémentaires :

- CI/CD ;
- Monitoring ;
- Logging ;
- Tracing.

Ces catégories ont l'avantage d'être partagées par tous les cloud providers. D'autres catégories propres à chaque cloud provider apparaissent aussi, comme :

- Developer tools ;
- Containers ;
- Kubernetes.

Nous avons maintenant tout le nécessaire à la bonne implémentation du Serverless chez les CSP. Essayons de faire un listing complet des offres que proposent chacun de ces cloud providers(123) : voir tableau 1.

1

	AWS	Azure	GCP
Compute	Lambda / Lambda@Edge	Azure functions	Cloud functions / App Engine
API	API Gateway	API Management	Apigee
Storage	S3 / EFS	Azure File	Cloud Storage
Datastore	DynamoDB / Aurora Serverless	CosmosDB	Cloud Firestore / BigQuery
Messaging	SNS / SQS	EventGrid / ServiceBus	Cloud Pub/Sub
Orchestration	Step functions	Logic Apps	
CI/CD	CodeStar / CodePipeline / CodeBuild / CodeDeploy	Azure DevOps	Cloud Build
Developer tools	Cloud9 / Eclipse	Visual Studio / Visual Studio Code	Cloud Code
Monitoring	Cloudwatch	Azure Monitor	Stackdriver Monitoring
Logging	Cloudwatch Logs	Azure Log Analytics	Stackdriver Logging
Tracing	X-Ray	Azure AppInsight	Stackdriver Trace
AI / Cognitive	Rekognition	Bot Service / Cognitive Services	Cloud ML Engine
Containers	Fargate	Azure Container Instances	
Kubernetes		Virtual Kubelet	Serverless for Kubernetes Engine (alpha)
Analytics	Kinesis / Athena	Azure Stream Analytics	Cloud Dataflow
Service Mesh	App Mesh	Service Fabric Mesh	Istio

Les offres On-Prem'

Evidemment, avec toutes ces offres et le développement de Kubernetes chez les clients, de plus en plus de projets open source ont vu le jour afin d'essayer d'offrir les mêmes caractéristiques nécessaires au Serverless. La plupart de ces projets ont comme support Kubernetes.

Nous allons en lister quelques-uns ici. Généralement, ces projets sont principalement basés sur l'approche FaaS. Les développeurs peuvent néanmoins ajouter d'autres projets open source afin de compléter la vision du Serverless. 2

Nous voyons ici que l'offre des frameworks FaaS on-premise est riche et qu'elle se développe à grande vitesse afin de proposer à la communauté les briques de base afin de pouvoir répliquer ce modèle d'architecture dans les datacenters traditionnels.

En conclusion

J'ai, dans ce dossier, fait un état des lieux du Serverless, ainsi que des différentes offres des cloud providers. Cet état des lieux nous a permis d'avoir un regard sur les différentes briques possibles à utiliser lors du design d'une solution Serverless. Malheureusement, l'espace m'est restreint pour pouvoir rentrer en détail dans les implémentations techniques. Tout le monde sait que le diable se cache dans les détails et que les offres des cloud providers ne sont pas forcément équivalentes en termes de complétude.

Il faudrait alors tester quelles sont les performances de chaque offre, regarder le coût de celles-ci et comment l'appliquer dans un contexte d'entreprise. Le Serverless est alors bien pour commencer sur une nouvelle application, mais peut-être pas pour migrer l'essentiel d'un système d'entreprise.

Le Serverless reste relativement jeune (il n'a que trois ans d'existence, même si le concept remonte au début des années 2000). Malgré cela, nous voyons de plus en plus de briques sortir chez AWS, Azure ou GCP afin de compléter la vision et les besoins de maintien en conditions opérationnelles. Je constate aussi que de plus en plus d'entreprise ne rechignent plus à migrer vers ces nouvelles architectures, poussées par le besoin de refactoring des applications en microservices et ce, malgré la puissance des containers dans l'organisation.

Enfin, il ne faut pas, avec cette approche Serverless, penser que le développeur peut se passer complètement d'un exploitant. Cette réflexion est notamment poussée par le mouvement NoOps, où l'abstraction et l'automatisation sont telles qu'il n'y a plus besoin de gestion de l'infrastructure. Les développeurs peuvent alors maintenir en service l'application tout seuls. Il ne faut cependant pas oublier que le métier d'exploitant est de maintenir en conditions opérationnelles l'application, et que les contraintes d'exploitation ne sont pas les mêmes que celles du développement. •

(1) "Serverless Computing | Amazon Web Services." https://aws.amazon.com/serverless/

(2) "Serverless Computing | Google Cloud." https://cloud.google.com/serverless/

(3) "Serverless Computing | Microsoft Azure." https://azure.microsoft.com/en-us/overview/serverless-computing/

devoteam | M Applications

RANGERS

Hamza Azzouz
Consultant Microsoft C#.NET/SQL chez Devoteam MA.
Email: hamza_azzouz@outlook.com

Introduction à .Net Core 3 & C# 8

.NET Core 3 est la prochaine nouvelle version de .NET Core, le Framework open source de Microsoft qui va prendre en charge les fonctionnalités suivantes : prise en charge de WinForm, prise en charge de WPF, EntityFramework 6, création d'applications web côté client avec Razor, C# 8, Internet des objets IOT (Internet-Of-Things) et le support ML.NET (machine learning).

niveau 100

C# 8 est la nouvelle version majeure du langage C# que Microsoft prévoit de livrer en même temps que .Net Core3. Cette version fournira plusieurs fonctionnalités, dont :

- Types de référence nullables ;
- Les types Range et Index ;
- Les Flux asynchrones ;
- Implémentation par défaut des membres d'interfaces ;
- …

Bien évidemment je reviendrai juste après en détail sur chacune d'elles.

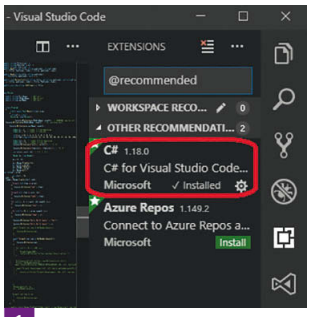
Installation et configuration de C# 8 :

On peut télécharger la version "Preview" de .Net Core 3, et travailler avec Visual Studio Code, ou bien avec Visual Studio 2019.

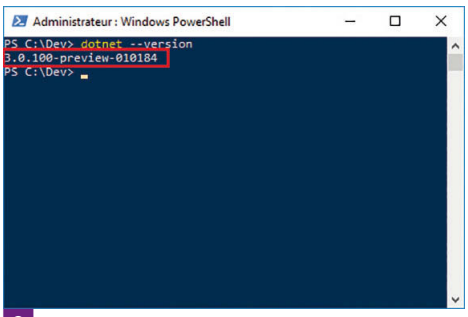
Les étapes d'installation et configuration avec VS CODE :

- Installer VSCODE sur l'adresse : https://code.visualstudio.com/ ;
- Installer la version « Preview » de .NET Core SDK qui se trouve sur : https://dotnet.microsoft.com/download/dotnet-core/3.0 ;
- Installer l'extension C# de Visual studio Code (OmniSharp), depuis le menu « Install From VSIX » de vs Code, ou bien directement via VS CODE extensions. 1

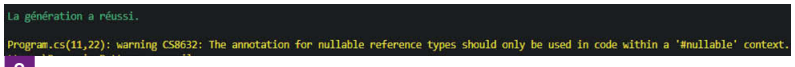
À la fin de l'installation vous pouvez vérifier si elle est bien faite en utilisant la commande suivante sous Ms-Dos ou PowerShell :



1



2



3

dotnet --version 2

Quand vous créez votre premier projet en C# 8, les fonctionnalités de cette version ne sont pas activées. Pour ce faire, il faut modifier le fichier projet de votre applications c'est à dire le « .csproj » pour ajouter les deux lignes suivantes :

- <LangVersion>8.0</LangVersion> : pour l'activation de C# 8 ;
- <NullableContextOptions>enable</NullableContextOptions> : cette ligne permet d'activer l'une des fonctionnalités phare de cette nouvelle version de C# qui est les références nullables.

« NullableContextOptions » , une fois activé, si vous avez des variables références dans votre code elles deviendront non nullables, cette option va nous servir l'ors des migrations de l'ancien code vers le nouveau, j'en parlerai un peu plus tard.

Et de même, si vous créez des variables de type référence null, exemple string ? Chaîne, et si la fonctionnalité est désactivée alors vous aurez un Warning de ce genre : *The annotation for nullable reference types should only be used in code within a '#nullable' context.* 3

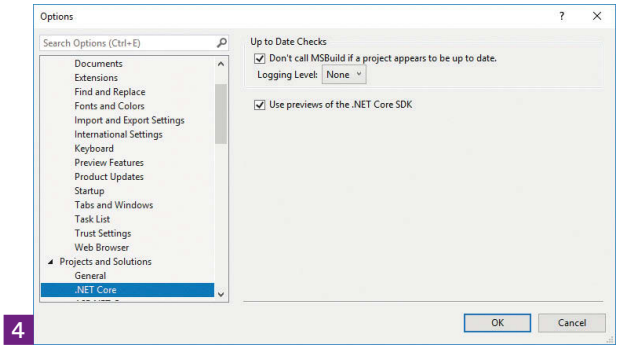
Les étapes d'installation et configuration avec VS 2019 :

- Installer Visual Studio 2019 qui est sur le lien suivant : https://visualstudio.microsoft.com/ ;
- Installer la version .Net Core 3 comme dit auparavant ;
- Lancer l'installation de Visual studio 2019, vous pouvez prendre la version community ;
- Activer le .Net Core 3 dans Tools->Options->Projects and Solutions->.Net Core :

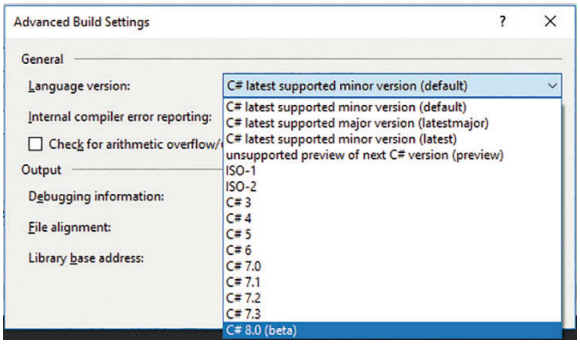
et cochez la case : use previews of the .NET Core SDK et redémarrez Visual studio. 4

Maintenant il faut activer le C# 8 dans les propriétés de projet; dans l'onglet build cliquez sur le bouton « Advanced Build Settings »: il faut choisir C#8.0 (beta). 5

Et voilà! Le tour est joué! Vous pouvez maintenant commencer à travailler avec le nouveau C#8.



4



5

Types référence Nullable

Qui se souvient bien de la fameuse exception : « la référence d'objet n'est pas définie à une instance d'un objet ». Je pense que chacun de nous a eu ce genre d'erreur durant son expérience. C'est souvent difficile de trouver la cause de cette exception. Toutefois, elle est généralement reliée aux types références. Avec le fait d'essayer d'accéder à un membre d'un objet qui est Null, le système lève cette exception : « system.nullreferenceexception ».

Dans C# 8, Microsoft veux aider les concepteurs et développeurs à diminuer cette exception sous réserve bien sûr que les concepteurs fassent un effort supplémentaire pour distinguer d'une manière explicite et responsable des objets qui pourront être null ou pas. Il s'agit aussi pour les développeurs d'implémenter correctement cette conception en faisant en sorte que les types référence n'accepteront plus de valeurs null, sauf si on les déclare nullables explicitement avec la même syntaxe que les types nullables :

Exemple : int? valeur = null : déclaration d'un type int nullable, désormais pour déclarer une string nullable, on fait pareil : string? chaine = null.

Et par conséquent ça ne sera plus possible de déclarer comme ceci : string chaine = null; si on le fait on aura le Warning suivant :

```
Program.cs(18,22): warning CS8600: Converting null literal or possible null value to non-nullable type.
cationdemo.csproj]
    1 Avertissement(s)
    0 Erreur(s)
```

Pour l'instant Microsoft prévoit un moyen pour pouvoir activer ou pas cette fonctionnalité dans le but de ne pas impacter le code existant. Toutefois, pour l'instant, même l'impact n'est pas vraiment grand, car, au lieu d'afficher des erreurs, le compilateur affichera des Warnings (avertissements). Cette nouvelle fonctionnalité reconnaîtra les différentes méthodes existantes de vérification de null, et donc pas de panique, car y a pas d'obligation de modification, par contre prévoir une migration du code c'est mieux.

Pour comprendre un peu la philosophie de Microsoft sur le choix de nullable pour les types références, regardons cet exemple défini dans le blog dev de Microsoft :

```
class Person
{
  public string FirstName;   // non null
  public string? MiddleName; // il peut etre  null
  public string LastName;    // non  null
}
```

Logiquement il est très proche de la réalité qu'une personne doit obligatoirement avoir un nom, un prénom, et ne pas avoir forcément un deuxième prénom "Middlename". Par conséquent le développeur doit déclarer MiddleName comme étant Nullable.

Ranges and indices

Les Index : les index sont utilisés pour l'indexation d'un tableau. Il y a deux sortes d'index. Un index qui commence en début de tableau avec la position 0, et l'index qui commence par la fin de tableau avec la position 1.

Cela signifie :

En admettant qu'on a un tableau d'entiers : int[] Tab = new int[10]{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

On déclare un index comme suit :

Index ibegin = 4; // index qui commence le comptage de début

Index iend = ^6; // index qui commence le comptage de la fin de tableau en commençant par 1, le signe ^ positionne l'index à partir de la fin du tableau

Tab[ibegin] affichera 4, le 4eme élément de tableau en commençant le comptage de 0 de début de tableau.

Tab[iend] affichera 5, le 6 ème élément de tableau en commençant le comptage de 1 de la fin de tableau.

Les Ranges : un range c'est un nouveau type structure qui contient un index début et un index fin. On déclare un range comme suit :

```
char[] charArray = new char[] { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H' };
Range r = new Range(3, ^2);
Range r1 = 3..^2;
```

Les deux instructions affichent "DEF". On commence à partir de la position 3 de début de tableau qui est D et à partir de la fin du tableau on compte 2 positions, donc le F.

Autres exemples:

charArray[3..3] de la position 3 à partir de début jusqu'à la position 3 à partir de début , renvoie un tableau vide.

charArray[^ 3, ^ 3] de la position 3 à partir de la fin jusqu'à la position à 3 à partir de la fin, renvoie un tableau vide.

charArray[4.. ^ 8] de la position 4 de début jusqu'à la position 8 à partir de la fin, renvoie une exception.

charArray[..8] de début de tableau jusqu'à la position 8 à partir de la fin, renvoie tout le tableau

charArray[4..] de la position 4 à partir de début de tableau jusqu'à la fin du tableau, renvoie EFGH

Attention : quand vous donnez des index début et fin qui se chevauchent vous aurez une exception de type : ***System.OverflowException*** : *'Arithmetic operation resulted in an overflow*

Fonctions et propriétés Ranges:

Range.StartAt(N) initialise l'index de début à N, => [N.. ^ 0]

Range.EndAt(N) initialise l'index de la fin à N => [0..N]

Range.All initialise de début et la fin à 0 => [0.. ^ 0] donc tout le tableau.

Les Itérations

```
foreach (int item in a[ibegin..iend]){ }
for (int i = 0; i < a[ibegin..iend].Length; i++) {}
```

Utilisation avec le type Span, ReadOnlySpan, Memory, ReadOnlyMemory:

Vous avez sûrement entendu parler de nouvelles structures, Span, ReadOnlySpan, Memory, ReadOnlyMemory qui permettent de faciliter la manipulation des tableaux et sous-tableaux, d'une manière plus performante et plus fiable.

On peut désormais initialiser ces structures avec le nouveau type Range: la nouvelle méthode d'extension « AsSpan » accepte en paramètre un type Range :

```
Span<int> span = a.AsSpan(Range.All);  // tout le tableau
Range rg = new Range(3, ^2);
Span<int> span = a.AsSpan(rg);  // une partie du tableau
ReadOnlySpan<int> readOnlySpan = a.AsSpan(rg);
Memory<int> memory = a.AsMemory(rg);
ReadOnlyMemory<int> rmemory = a.AsMemory(rg);
```

La méthode SubString()

Vous pouvez également utiliser le type range pour extraire des sous-chaînes de caractères d'une manière rapide.

Anciennement, on utilise la fonction SubString() et on paramètre. On lui passe l'index de début et le nombre de caractères à prendre :

```
"Hello World".Substring(6, 5) ; // World
```

En revanche avec les Ranges, une légère modification de conception. Cette fois-ci, on lui passe un index début et un index fin. Vous pouvez également passer un range en paramètre de la fonction SubString comme vous pouvez utiliser la syntaxe [x..y] directement.

```
"Hello World".Substring(new Range(6, ^0)); // World
"Hello World"[6..^0]; // World
```

Les Flux asynchrones

L'introduction des flux asynchrones dans le C# 8, est très utile lorsque vous voulez consommer des données dans un flux continu qui provient d'une source asynchrone. Par exemple les données d'un capteur qui analyse les paramètres du climat, ou des bases de données dans le cloud, etc.

Microsoft a mis en place à travers C# 8 un ensemble de mécanismes permettant de facilité d'implémentation de la création et la consommation des flux asynchrones en ajoutant le mot clé Async pour les méthodes qui retournent un flux asynchrone, et de définir leur type de retour comme IAsyncEnumerable<T>, la version asynchrone de IEnumerable<T>, et de retourner les résultats à la demande avec le mot clé Yield return.

Et en ce qui concerne la consommation de ce flux, on doit itérer avec la boucle foreach sur notre résultat qui est de type IAsyncEnumerable<T>, et cela en utilisant le mot clé await avant le foreach.

Voici un exemple supposant qu'on veut consommer les données d'un capteur météorologique qui nous donne la variation de température. On est dans du pur asynchrone à chaque changement de température… Par conséquent, l'utilisation des opérations asynchrones (Asych, await) est ici indispensable. Sinon, on aura un blocage d'application à chaque fois qu'on attend une réponse du service température. La création de flux asynchrone :

```
private static async IAsyncEnumerable<string> GetTemperature()
    {
      for (int i = 0; i < 10; i++)
      {
        temperature = temp.Next(10, 16);
        delay = waiting.Next(1000, 60000);
        await Task.Delay(delay);
        if (oldtemp != temperature)
        {
          oldtemp = temperature;
          yield return $"la température actuelle est de {temperature} °C";
        }
      }
    }
La consommation de flux :
await foreach (var num in GetTemperature())
{
  Console.WriteLine(num);
}
```

Implémentation par défaut des membres d'interfaces

On connaît tous un des principes des interfaces qui dit qu'une interface ne peut pas avoir des implémentations de méthodes, mais ça c'était avant ! Avec C# 8, il est désormais possible d'offrir une implémentation par défaut à des méthodes et des propriétés uniquement ! Cela peut sembler dégoûtant, mais examinons pourquoi Microsoft a procédé à ce changement.

Aujourd'hui pour créer un code propre maintenable et scalable, on doit utiliser des interfaces, ce concept qui permet de définir un contrat entre les classes qui l'implémentent.

Imaginant qu'une usine produit des robots avec la fonctionnalité suivante (Move), et qu'une année après l'entreprise ait décidé de créer un module d'intelligence artificielle qui permette à tous les robots de parler.

L'entreprise fabrique plusieurs modèles de robots et chaque modèle est fabriqué dans une sous-usine avec une équipe dédiée. Chaque sous usine a accès au module commun de gestion des robots dont l'interface IRobot

```
public interface IRobot
  {
    void Move();
    void Speak()
    {          Console.WriteLine("Parler...");
    }
  }
class RobotModele1 : IRobot
  {
    public void Move()
    {        Console.WriteLine("Bouger...");
    }
  }
```

```
class RobotModele2 : IRobot
  {
    public void Move()
    {
      Console.WriteLine("Bouger...");
    }
  }
```

Donc vous imaginez les conséquences qu'engendre l'ajout de la méthode « Speak() » dans l'interface. Cela produira effectivement beaucoup de changement, car toutes les équipes qui utilisent cette interface devront, corriger toutes les erreurs de compilateurs en modifiant leurs classes pour implémenter cette nouvelle fonctionnalité, et, ensuite, coder la fonctionnalité pour chaque classe, alors qu'elle est sensée être la même dans tous les modèles de robots et par conséquent elle devra être mutualisée.

C'est pour ça qu'avec C# 8 on peut rajouter la méthode speak() dans l'interface et lui offrir une implémentation commune par défaut à tous les modèles de robots. Le résultat sera juste magnifique car les équipes n'auront même pas de changement à faire dans leurs codes… et c'est un gain de temps très important, on est d'accord ! Même s'il y en a qui vont dire qu'on peut faire ça avec une classe abstraite, je dis oui. Mais dans un grand projet, le temps

qu'il faut pour reconstruire avec une classe abstraite est beaucoup plus important que de faire une implémentation par défaut. En revanche, je vous conseille de ne pas utiliser cette fonctionnalité à tout va. Il s'agit plutôt de l'exploiter dans des cas où vous n'auriez pas beaucoup d'autres choix; dès qu'il y a une autre solution possible, n'utilisez jamais l'implémentation par défaut.

Note : *pour utiliser l'implémentation par défaut, il faut déclarer la variable comme étant de type Interface.*

Exemple :

```
RobotModele1 robot1 = new RobotModele1();
    IRobot robot2 = new RobotModele1();
    robot1.Parler() : ça ne marche pas
    robot2.Parler() : marche
```

Les patterns Récursifs

En C# 7, il y a eu l'apparition des patterns matching, une technique qui vient de la programmation fonctionnelle comme le F#. Elle consiste à donner en entrée une expression et de l'évaluer à travers des motifs (des filtres) pour déduire le modèle ou l'expression correspondante en sortie comme résultat.
Dans la programmation C#, on peut définir cela comme étant des instructions de sélections (Switch ..case, ou bien plusieurs if else imbriqués).
Le code ci-dessous nous montre un exemple d'une syntaxe générique d'un pattern matching : qui veut dire "Matcher l'expression « Expression » avec les motifs (Motif 1, ..., Motif N) pour produire un résultat (Expression 1, ... Expression N).

```
matcher <Expression> avec
| <Motif1> -> <Expression1>
| <Motif2> -> <Expression2>
...
| <MotifN> -> <ExpressionN>
```

Avant, on pouvait écrire des patterns simples comme le montre l'exemple C# 7 suivant :

```
if (liquid is Water wt){ Console.WriteLine($"The name is {wt.Name}"); }
```

Mais en C# 8 on peut même introduire un autre pattern à l'intérieur du premier pattern, admettant qu'on veut afficher quelques caractéristiques du liquide uniquement si c'est de l'eau et que sa source se trouve dans les Alpes.
On va appliquer l'expression suivante qui va contenir un pattern dans un autre, le premier qui vérifie est ce que le liquide est bien de l'eau et le deuxième vérifie sa source :
C# 8 : (Pattern de propriétés)

```
if (liquide is Water { Source:"Les Alpes", Name:string name, Temperature:double temperature })
{
  Console.WriteLine($"Water name is {name}, his temperature is {temperature}");
}
```

Et voilà! L'application des patterns sur une arborescence d'objets est simplifiée et récursive, et c'est très performant !!
Voyons maintenant de côté les expressions Switch; C# 8 a encore simplifié et réduit la verbosité de ces expressions.

Switch expressions

Switch est généralement utilisé pour remplacer une construction trop répétitive des instructions conditionnelles de If ..Else. Avant C# 7, le Switch était limité à des types string et int, long, char, bool, enum, ... Mais à partir de C# 7, avec l'apparition des Patterns Matchings, l'instruction Switch pouvait manipuler d'autres types d'objets, ce qui s'avère très utile ...
Pareil pour les expressions Switch, elles existaient aussi dans C#7, mais elles sont très verbeuses, fastidieuses à écrire. Et tout ça pour ne retourner qu'une seule valeur à la fin ! Du coup, C# 8 a donné une version légèrement simplifiée pour l'utilisation des expressions Switch, et casser cette complexité en essayant de faire le moins de code possible.
Voici une petite comparaison entre un Switch case avant et après C# 8 :

```
private static string SwichV1(Matter matter)
    {
      switch (matter)
      {
        case Water eau when eau.Drinkable is true:
          return $"L'eau: {eau.Name} qui vient de : {eau.Source} est potable";
        case Water eau when eau.Drinkable is false:
          return $"L'eau: {eau.Name} qui vient de : {eau.Source} est non potable";
        case Water eau when eau is default(Water):
          return $"Instance vide !";
        case null:
          return $"L'objet est null !";
        case Petroleum petrole when petrole.Molecule.Name == "H":
          return $" Ce pétrole : {petrole.Name} , Contient la molécule : {petrole.Molecule.
Name} ";
        case Petroleum petrole when petrole.Temperature == 50 && petrole.Flammable:
         return $"Attention la température de ce liquide : {petrole.Name} est trop elevé plus
il est inflammable";
        case Petroleum petrole when petrole.Temperature > 70:
          return $"Attention ce liquide {petrole.Name} , très très chaud !";
        case Petroleum matiere when matiere is Liquid liquid && liquid is Petroleum petrole:
          return $" Cette matière est un liquide ({petrole.Name}), il est produit en : {petrole.
ProducingCountry}";
        default:
          return $"Matière non reconnue !";
      }
    }
```

Avec C# 8

```
private static string SwichV2(Matter matter) =>
    matter switch
    {
      Water { Drinkable: true, Name: string name, Source: string source } eau => $"L'eau:
{name} qui vient de : {source} est potable",
      Water { Drinkable: false, Name: string name, Source: string source } eau => $"L'eau:
{name} qui vient de : {source} est non potable",
      Water { } eau => $"Instance vide !",
      null => $"L'objet est null !",
      Petroleum { Molecule: { Name: "H", Name: string moleculename }, Name: string liquidname }
petrole => $" Ce pétrole : {liquidname} , Contient la molécule : {moleculename} ",
```

```
        Petroleum { Temperature: 50, Flammable: true, Name: string name } petrole =>
$"Attention la température de ce liquide : {name} est trop elevé plus il est inflammable",
        Petroleum { Temperature: int temp, Name: string name } petrole when temp > 70 =>
$"Attention ce liquide {name} , très très chaud !",
        Matter matiere when matiere is Liquid liquid && liquid is Petroleum { Name: string
name, ProducingCountry: string pays } petrole => $" Cette matière est un liquide ({name}),
il est produit en : {pays}",
        _ => $"Matière non reconnue !"
    };
```

Comme vous l'avez constaté, on remarque que le Switch est désormais seul. Il ne prend pas de valeurs entre parenthèses. Il y a la disparition des Cases qui sont remplacés par =>, et le mot clé default remplacé par _, et l'introduction des patterns récursifs qui inspectent les propriétés de l'objet (pattern de propriétés). Tout cela rend la construction plus lisible et élégante, avec moins de code , n'est-ce pas ?
Et comme vous le remarquez aussi, vous pouvez utiliser des patterns vides ou null de cette manière :

```
Water {} eau => $"Instance vide !",
null => $"L'objet est null !",
```

C'est très pratique lorsqu'on veut éviter un maximum d'erreurs à l'exécution des patterns.

Patterns Positional

A partir de C# 7, vous pouvez écrire des méthodes « Déconstructeur » (Deconstruct) dans des classes. Elles permettent de déconstruire un objet. Dans les variables tuple, leurs syntaxe est définie comme suit :

```
public void Deconstruct(out string name, out string? color, out bool flammable) => (name,
color, flammable) = (Name, Color, Flammable);
```

En C# 8, les patterns bénéficient de cette méthode pour construire ce qu'on appelle des patterns positional en s'appuyant sur les méthodes « Deconstruct ». Dans l'exemple ci-dessous, on instancie des objets liquides qu'on passe dans notre méthode « Show » qui permet de deviner le type de liquide en le déconstruisant :

```
static string Show(Liquid liquid) => liquid switch
    {
      ("Eau",null,false)  => "Is Water !",
      (var x, var y, var z) when z is true => $"Is Flammable !",
      _ => "unknown"
    };
```

Patterns Tuples

Ils sont très similaire aux patterns positional. Ils permettent de tester plusieurs éléments en même temps en se basant sur la syntaxe d'un tuple. Voici une illustration dans l'exemple ci-dessous qui permet d'afficher un message d'erreur suivant un tuple (erreur, code d'erreur) :

```
static string ShowError(string erreur, int code) =>
  (erreur, code) switch
    {
      ("File Not Found", 1) => "Error with code 1, filesystem",
      ("Index Out Of Range", 2) => "Error index out, array",
      (null, 0) => "No Error",
      (_, _) => "unknown"
    };
```

Déclarations using

Vous en avez marre des accolades à chaque fois que vous codez un bloc Using ? Et surtout lorsque ces blocs sont imbriqués ?

```
using (ClassA a = new ClassA())
      {
        using (ClassB b = new ClassB())
        {
          using (ClassC c = new ClassC())
          {
            // instructions
          }
        }
      }
```

La multiplicité des accolades devient alors trop importante, ce qui rend le code moins lisible.
En C#8 l'écriture des blocs Usings a été simplifiée en enlevant les accolades, mais je sais que beaucoup d'entre vous se posent la question suivante : dans les blocs usings, avec l'ancienne écriture on sait que les objets seront disposés à la fin de l'accolade de bloc correspondant… Mais là, avec cette écriture simplifiée, on ne voit pas trop comment les objets seront disposés…
Une bonne question effectivement, mais au lieu de disposer l'objet à la fin de l'accolade de Using correspondant, on le dispose à la fin du contexte dans lequel l'objet a été instancié.
Exemple : si on a une fonction qui écrit un flux dans un fichier texte comme suit :

```
private static void WriteInFile()
{
  string filepath = @"C:\_DEV\UsingsDeclaration\file.txt";
  using StreamWriter stream = new StreamWriter(filepath, true);
}
```

comme l'objet stream est instancié à l'intérieur de la fonction WriteInFile() alors il sera disposé à la fin de l'accolade de cette fonction.

Fonctions local (statiques)

Les fonctions locales introduites dans C# 7 sont un outils très puissant pour écrire des mini fonctionnalités à l'intérieur d'une fonction mère. Ceci tout en pensant bien sûr à respecter les principes solides, et de ne pas tomber dans une fonction couteau suisse avec plusieurs fonctions locales divergentes. Par contre, faire des fonctions à usage local en respectant l'orientation de la fonction globale est une bonne pratique et cela évite également qu'un autre développeur puisse appeler une fonction locale par erreur, car elle est encapsulée.
Parmi les contraintes d'utilisation des fonctions statiques, vous ne pouvez pas ajouter un modificateur d'accès car elles sont privées par défaut, et aussi vous ne pouvez pas appliquer le mot clé Static.
Avec C# 8 vous pouvez désormais utiliser le mot clé Static sur les fonctions locales, par contre, elles, ne peuvent pas accéder à des

variables qui sont dans la portée de la fonction globale; cela est un avantage, car par souci de performances, une fonction locale doit utiliser ses propres variables.
Voir l'exemple ci-dessous :

```
private static (int, int, int) CalculateV1(int a, int b)
{
 int resultAdd, resultSub, resultIncrem;
 int Add() { return a + b; }
 int Increment() { b = b + 1; return b; }
 int Sub() { return a - b; }
 resultAdd = Add();
 resultIncrem =Increment();
 resultSub = Sub();
 return (resultAdd, resultSub, resultIncrem);
}
```

```
private static (int, int, int) CalculateV2(int a, int b)
{
 int resultAdd, resultSub, resultIncrem;
 static int Add(int x, int y) { return x + y; }
 static int Increment(int x) {x = x + 1; return x; }
 static int Sub(int x, int y) { return x - y; }
 resultAdd = Add(a,b);
 resultIncrem =Increment(b);
 resultSub = Sub(a,b);
 return (resultAdd, resultSub, resultIncrem);
}
```

Résultat : pour (2,2)
Add = 4 , Sub = -1, second param increment = 3 // fonction non statique => résultat non correct.
Add = 4 , Sub = 0, second param increment = 3 // fonction statique => résultat correct

Je trouve que les fonctions statiques locales sont un bon moyen pour éviter que les développeurs puissent manipuler les variables de la fonction globale, justement dans le corps de la fonction locale. Ceci en raison des performances, comme on vient de le voir dans l'exemple précédent. Et puisque les méthodes statiques ne peuvent pas accéder à des champs non statiques, ni à une variable d'instance d'un objet quelconque dans leur type contenant, cela nous oblige à passer les variables globales en paramètres.
Vous allez me dire qu'on peut aussi dans le cadre des fonctions non statiques utiliser des paramètres. Oui, sauf que ce n'est pas obligatoire. En revanche, les fonctions statiques sont une obligation si vous voulez manipuler les variables globales, sinon le compilateur vous génère l'erreur suivante « *a static local function cannot contain a reference to <variable>* » donc vous n'avez pas le choix , et c'est tant mieux !

Autres fonctionnalités

Certaines de ces fonctionnalités que je citerai ci-dessous ne sont pas encore finalisées. Elles devront être publiées dans des versions ultérieures de C#8 c'est-à-dire C#8.X

Target-typed new-expressions (les types ciblés)

Target-typed new-expressions : c'est une nouvelle fonctionnalité dans C#8 qui vous permet d'omettre le type de l'objet au mot clé New lorsque vous instanciez ce dernier, c'est à dire dans ma classe Liquid , je peux instancier un liquide de cette façon :

```
Liquid liquid = new { Name = "Lat", Color = "White", Flammable = false };
```

Alors qu'anciennement, on le fait comme ça :

```
Liquid liquid = new Liquid { Name = "Lat", Color = "White", Flammable = false };
```

Le compilateur est capable d'inférer le type de la partie gauche de la déclaration de l'objet, par conséquent on aura moins de code à écrire.

Ref Struct

Pour accéder à la mémoire non managée , C# 7 à introduit les ref struct qui est une structure qui sera allouée uniquement sur la pile. Span<T> est un exemple de ce type qui fournit des performances très hautes car il gère automatiquement les accès à la mémoire managée et non managée et à la pile. Par conséquent, lorsqu'on manipule des chaînes de caractères ou des tableaux de grands volumes de données avec Span ou ReadOnlySpan, on garantira plus de performances dans la gestion de la mémoire. Toutefois, une petite limitation qui s'impose, c'est que ces types ref struct ne peuvent pas être disposés en utilisant le « Using » car ils ne peuvent pas implémenter des interfaces. C'est la raison pour laquelle C# 8 permet aux ref struct d'implémenter l'interface IDisposable. Voici un exemple :

```
ref struct Liquid : IDisposable
{
  public void Dispose()
  {
  }
}
using (var liquid = new Liquid ()) {.. } // liberation de resources...
```

L'attribut CallerArgumentExpression

A des fins de diagnostiques, C#8 veut ajouter l'attribut « CallerArgumentExpression » dans le namespace System.Runtime.CompilerServices. Ceci afin qu'une méthode appelée puisse recevoir plus d'informations sur celui qui l' a appelée (appelant). Pour rappel : dans la version C# actuelle, il y a déjà les attributs suivants : (CallerMemberName, CallerFilePath et CallerLineNumber).

Déconstruction du littéral « default »

Pour initialiser une variable à une valeur par défaut, on utilise le mot clé « default », le mécanisme est le même pour les Tuples également.
Avant C# 7, on peut initialiser un tuple de cette façon : on donne à chaque type sa valeur par défaut.

```
(int x1, string y1) = (default(int), default(string));
```

Ensuite en C# 7, il y avait une amélioration pour supprimer les types et laisser juste les mots clés default :

```
(int x2, string y2) = (default, default);
```

Et désormais en c# 8, un seul default peut initialiser un tuple.

```
(int x3, string y3) = default;
```

Operateur ??=

Si vous voulez initialiser une variable lorsqu'elle est null avant C# 8 vous testiez avec le mot clé if

```
if (value == null){value = "Hello World!"; }
```

Avec le mot clé ??= le code est plus simple. Exemple : value ??= "Hello World!";

Changement de l'ordre dans les opérateurs $,@

Pour la mise en forme des chaînes de caractères C# 6 a introduit l'opérateur $ utilisé pour remplacer string.Format. Mais dans une

chaîne dans laquelle on veut rajouter l'opérateur @ pour interpréter textuellement les séquences d'échappement, on aura les 2 opérateurs cote à cote : $@
En C# 7 : la syntaxe était d'écrire le $ suivi de @

```
var filePath = $@"c:\MyFolder\{file}";
```

En C# 8, c'est l'inverse, le @ est suivi de $

```
var filePath = @$"c:\MyFolder\{file}";
```

Utiliser des attributs sur des classes génériques :

Les attributs sont des moyens plus efficaces pour ajouter des métadonnées à nos classes, cependant on ne pouvait pas créer des attributs génériques, et le but de cette nouvelle fonctionnalité est de remédier à cela :

```
public class GenericValidateAttribute<T> : Attribute { }
[GenericValidate<int>]
public class ClassToValidate { }
```

Les Records (enregistrement)

C'est un nouveau format de déclaration de classe et des structures simplifiées, qui permet aussi d'intégrer un certain nombre de fonctionnalités. A partir d'une déclaration de base, le compilateur est donc sensé travailler avec une classe plus détaillée générée par lui, voici un exemple :

```
public class Liquid(string Name, string Color);
```

Le compilateur génère pour lui une classe beaucoup plus grande qui implémente également IEquatable et qui contient des méthodes telles que GetHashCode(), Equals(), Deconstruct(), etc.

```
public class Liquid: IEquatable<Liquid>
{
  public string Name { get; }
  public string Color { get; }
  public Liquid(string Name, string Color)
  {
    this.Name = Name;
    this.Surname = Color;
  }
  public bool Equals(Liquid other) { return Equals(Name, other.Name) && Equals(Color, other
.Color); }
  public override bool Equals(object other) { return (other as Liquid)?.Equals(this) == true; }
  public override int GetHashCode() { return (Name.GetHashCode() * 17 + Color.GetHashCode()); }
  public void Deconstruct(out int Name, out int Color) { Name= this.Name; Color= this.Color; }
  public Liquid With(int Name = this.Name, int Color= this.Color) => new Liquid(Name, Color);
}
```

Extension de Tout (extension everything)

Aujourd'hui vous pouvez faires des extensions des objets on utilisant une classe statique et une méthode statique. Par exemple faire une extension à la classe string pour ajouter une méthode donne cela :

```
public static class MyString
{
  Public static int NewMethode(this string){.. do somthigs..}
}
```

Pour pallier les limitations présentes dans cette technique, C# 8 veut introduire une nouvelle manière d'étendre les objets avec une autre syntaxe complètement différente qui facilitera aux développeurs l'extension des méthodes et également des propriétés et autres choses …

```
public extension MyString extends String
{
  public int NewMethode(){.. do somthigs..}
}
```

Avec l'utilisation de mot clé extends qui veut dire étendre quelque chose.

2.11 Dépendances de plateforme

La plupart des nouvelles fonctionnalités de C#8 sont introduites dans la nouvelle version de .NetStandart2.1. .NET Core 3.0 ainsi que Xamarin, Unity et Mono implémenteront .NET Standard 2.1, mais pas le .NET Framework 4.8, ce qui explique que de nombreuses de ces nouvelles fonctionnalités ne fonctionneront pas sur le .NET Framework 4.8

Migration

L'une des fonctionnalités de C# 8, qui marque un pas majeur par rapport aux versions précédentes, c'est évidemment les références null, car au point de vue de la migration de code, c'est elle qui va générer plus d'efforts de migration que d'autres fonctionnalités. Cet effort de migration, on peut le mesurer par le nombre de Warnings générés suite à l'activation de cette fonctionnalité sur un ancien code. Par conséquent, il y a une équation exponentielle; plus le projet est volumineux plus le nombre de warnings générés est volumineux. Il est donc clair que le travail de migration qui sera fourni dans les cas des petits projets est minime, mais il peut s'avérer de grande envergure en ce qui concerne les grands projets d'entreprises et surtout les projets qui ont demandé moins d'efforts de conception. C'est pour cela que la manière la plus adaptée pour la migration de grands projets consiste à migrer par sous-projets et dans chaque sous-projet migrer modules par modules.

Exemple : on va essayer de migrer un code très simple. Ci-dessous une classe Liquid :

```
public class Liquid
 {
   public Liquid(string name, string color, bool flamable)
   {
     Name = name;
     Color = color;
     Flammable = flamable;
   }
   public string Name { get; set; }
   public string Color { get; set; }
   public bool Flammable { get; set; }
 }
```

Et une méthode qui calcule l'index d'une couleur avec une formule.

```
static void Main(string[] args)
```

```
    {
        Liquid eau = new Liquid("eau", null, false);
        int index = GetColorIndex(eau);
        Console.WriteLine("Hello World!");
    }
private static int GetColorIndex(Liquid liquid)
    {
        return liquid.Color.Length * 10;
    }
```

1- on doit commencer notre migration par l'activation des référence null dans notre projet. Pour cela il faut aller dans le fichier csproj et rajouter la ligne suivante : <NullableContextOptions>enable</NullableContextOptions>. Cette ligne permet d'activer le contrôle des références null dans notre projet. Et si vous voulez le désactiver, mettez disable :<NullableContextOptions>disable</NullableContextOptions>. Voilà! C'est plutôt pas mal! Il faut juste savoir qu'en faisant cela on activera la fonctionnalité sur l'ensemble de notre projet, donc, comme j'ai dit auparavant, si vous avez un grand projet vous risquez de voir beaucoup de Warnings à la fois ! Ce qui rend la migration complexe. Je vous conseille dans ce cas d'utiliser plutôt les directives suivantes #nullable enable et #nullable restore pour entourer le module ou la classe que vous voulez migrer ...c'est plus simple…

```
#nullable enable

// notre code à migrer...

#nullable restore
```

Et vous déclenchez des Warnings juste sur cette partie du code ; poursuivant notre migration :

2- Liquid eau = new Liquid("eau", null, false); on instancie un objet eau avec une couleur null, (l'eau n'a pas de couleur).

3- le compilateur n'est pas content car la propriété Color n'est pas nullable, il génère un Warning (***cannot convert null literal to non-nullable reference or unconstrained type prameter***) sur le null.

4- Donc on corrige la propriété Color à nullable : public string? Color { get; set; } par conséquent on exprime notre intention qu'il existe des liquides sans couleurs comme l'eau .

5- On corrige également le paramètre color de constructeur pour le mettre aussi à nullable.

6- C'est bien, mais encore une fois le compilateur nous prévient, avec un Warning (***possible dereference of null reference***). Si on ne corrige pas ce Warning, il y a une possibilité d'avoir une exception à l'exécution de GetColorIndex(), car on a déclaré la « Color » nullable.

7- On peut corriger cela de plusieurs façons, notamment en vérifiant si la couleur est non null, ou bien en rajoutant tout simplement l'opérateur ? (liquid.Color?.Length).

8- On peut utiliser l'opérateur « ! » pour corriger le Warning, mais dans ce cas vous dites au compilateur "ne t'inquiètes pas on est sûr de nous, on ne va pas recevoir une valeur null !!" Mais je ne vous conseille pas de le faire comme ça, sauf dans le cas où vous implémentez vous-même une fonction de tests de non null par exemple : if(!IsNull(liquid.Color)) return liquid.Color.Length * 10;
Dans ce cas là vous êtes sûr que vous vérifiez bien les null dans la fonction IsNull(). Par conséquent, le compilateur n'est pas capable de deviner ce que vous faites et là vous pouvez le rassurer en ajoutant l'opérateur ! (liquid.Color!.Length) et vous prenez votre responsabilité en cas d'exceptions. Une fois migré, vous allez diminuer énormément les exceptions Reference Null, c'est génial n'est-ce pas ?

Conclusion

C # 8.0 est enfin disponible en « Preview » dans Visual Studio 2019, ou VSCODE. La version finale sera publiée avec .NET Core 3.0. Malheureusement toutes les fonctionnalités, ne seront pas disponibles dans le Framework .NET, comme les flux asynchrones, les ranges et Index qui dépendent des types qui ne seront ajoutés qu'aux plateformes .NET compatibles avec .NET Standard 2.1.
Je pense que les fonctionnalités les plus importantes sont les types de référence nuls et les améliorations des patterns matchings, car ils vont aider les développeurs et concepteurs à créer des applications plus fiables avec un code plus lisible, donc plus maintenable. L'enjeu de migration reste cependant majeur notamment avec les types références null. A noter aussi que parmi les fonctionnalités citées et autres, il y en a beaucoup qui vont attendre les versions ultérieures de C#8. •

Erratum Programmez n°229 / article programmation Atari ST

"Au début de cette zone, il initialise une BASEPAGE (page de base) d'une taille de 512 octets » : en réalité, la taille de la BASEPAGE est de 256 octets.

Cédric Michel
Senior .Net Consultant chez Satellit et Scrum Master. Satellit est une entreprise de 70 consultants spécialisés dans la transformation business, la transformation digitale et dans les technologies Microsoft. Pour en savoir davantage, n'hésitez pas à aller consulter le site : **www.satellit.be**

Entity Framework (EF) Core 3.0

Faut-il encore présenter notre ORM Entity Framework(EF). Selon Microsoft EF Core est une version légère, extensible, open source et multi-plateforme de la très connue technologie d'accès aux données, Entity Framework.

niveau 200

Donc EF Core peut servir de mappeur relationnel/objet (O/RM), permettant aux développeurs .NET de travailler avec une base de données à l'aide d'objets .NET, et éliminant la nécessité de la plupart du code d'accès aux données qu'ils doivent généralement écrire. Mais avec la future nouvelle version 3.0, vous ne serez plus limité à un schéma relationnel. Comme nous le verrons plus loin, un nouveau fournisseur de base de données, Cosmos Db (NoSQL), rejoint la grande famille.

Quand vous utilisez des bases de données NoSQL pour votre couche de données d'infrastructure, vous n'utilisez généralement pas un ORM comme Entity Framework Core. À la place, vous utilisez l'API fournie par le moteur NoSQL, comme Azure Cosmos DB, MongoDB, Cassandra, RavenDB, CouchDB ou des tables Stockage Azure.

EF Core prend en charge de nombreux moteurs de bases de données SQL Server, SQLite,EF InMemory (conseillé pour les tests), PostgreSQL, MySQL, MariaDB, Serveur MyCAT, SQL Server Compact, Firebird, Db2, Informix, Access, Progress OpenEdge.

L'équipe Oracle .NET a publié une version bêta du fournisseur Oracle pour EF Core. Le nuget package Oracle.EntityFrameworkCore est disponible.

Support de C# 8.0.

Comme vous le savez peut-être déjà, la version C# 8.0 a été annoncée plus tôt.

Avec EF Core 3.0, nous pourrons utiliser certaines fonctionnalités impressionnantes de C# 8.0 telles que les flux asynchrones, les types de références Nullable, … Ainsi, EF Core 3.0 sera la première version à inclure les fonctionnalités de C# 8.0.

LINQ being more smart

Pour interroger vos bases de données, vous devez écrire des requêtes avec LINQ. Lorsque vous écrivez une requête complexe dans certains cas celle-ci peut échouer. Car lorsque vous faites cela, votre requête est traduite en SQL. Dans les premières versions d'EF Core, ce moteur est capable de découper votre requête en deux parties. La première partie sera toujours traduite en SQL, mais la seconde partie de la requête sera exécuté en mémoire côté client. Ceci risque peut avoir dans certains cas un effet pervers. Effectivement la partie exécutée côté client pourrait provoquer des problèmes.

Ce mécanisme est moins efficace et va dépendre énormément de la quantité de données et donc pourrait révéler des problèmes en production. Microsoft travaille donc à améliorer ce mécanisme et à traduire davantage l'expression LINQ directement en SQL. Ceci pourra donc générer des requêtes plus efficaces.

Entity Framework Core ne fait plus partie intégrante d'ASP.NET Core

Depuis la preview 1, EF Core 3.0 est devenu complètement indépendant. Avant ASP.NET Core 3.0, quand vous ajoutiez une référence de package à Microsoft.AspNetCore.App ou Microsoft.AspNetCore.All, ceci incluait directement EF Core. Cela ajoutait également certains fournisseurs de données EF Core tels que le fournisseur SQL Server. Maintenant, cela n'est plus le cas, il vous faudra ajouter les packages dont vous avez besoin. Une des raisons de ce changement est de pouvoir laisser le développeur faire évoluer la version d'ASP.NET Core indépendamment de la version d'EF Core. Si vous savez quel fournisseur vous allez utiliser, installer directement le package du provider. Celui-ci installera directement les dépendances dont il a besoin.

Les packages nécessaires.

Nous allons voir quels sont les packages nécessaires pour démarrer avec EF Core 3.0.

```
Install-Package Microsoft.EntityFrameworkCore
```

Ce premier package apporte principalement les objets DbContext et DbSet qui vont vous permettre de démarrer. 1*

Dans un second temps, nous avons besoin de définir quel fournisseur nous allons utiliser (exemple avec SqlServer.

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

Au niveau de votre Context (SchoolContext) vous pouvez surcharger la méthode *OnConfiguring*. Grâce au package, vous avez accès à une méthode d'extension *UseSqlServer*. Cela vous permettra d'y définir votre connexion string.

1*

```
public class SchoolContext : DbContext
  {     public DbSet<Student> Students { get; set; }
    public SchoolContext()
    {
      Database.EnsureCreated();
    }
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
     var localDbConnectionString = @"Data Source=(localdb)\ProjectsV13;Initial Catalog=
Student;Integrated Security=True;Connect Timeout=30;Encrypt=False;TrustServerCertificate
=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False";
        optionsBuilder.UseSqlServer(localDbConnectionString);

    }
  }
```

Les outils

Il existe deux ensembles d'outils utiles pour, entre autres, créer et appliquer des migrations de bases de données, ou créer un modèle EF Core basé sur une base de données existante.
Le premier installe des commandes utilisables depuis la console du gestionnaire de package (Add-Migration, Drop-Database, Get-DbContext, …)

```
Install-Package Microsoft.EntityFrameworkCore.Tools
```

1

Le second installe les outils Net Core CLI(Command-Line Interface). Il s'agit de commandes qui peuvent être utilisées sur Windows, Linux ou macOS. Jusqu'à maintenant ces outils, dotnet ef, faisaient partie du Sdk .Net Core. Comme elles ne font plus partie du SDK, il faudra les installer grâce à la commande suivante :

```
dotnet tool install --global dotnet-ef
```

Elles seront également utilisables depuis la console du gestionnaire de package. Cependant si vous utilisez Visual Studio, Microsoft préconise d'utiliser les commandes du gestionnaire de package car elles apportent principalement deux avantages :

- Elles fonctionnent automatiquement avec le projet sélectionné dans la Console du gestionnaire de package sans nécessiter de basculer manuellement vers le répertoire du projet.
- Elles ouvrent automatiquement les fichiers générés une fois la commande terminée.

Il faudra également installer le package suivant qui est un prérequis pour que le CLI puisse fonctionner.

```
Install-Package Microsoft.EntityFrameworkCore.Design
```

J'ai rencontré plusieurs problèmes pour installer et utiliser ce CLI.
Lorsque je voulais l'installer, j'avais le message d'erreur suivant «The application already exists». Mais lorsque je voulais utiliser une commande exemple : 'dotnet ef migrations add init', j'avais le message suivant : «The application to execute does not exist».
Je me suis rendu compte que le CLI installait un .exe ici : %userprofile%\.dotnet\tools\dotnet-ef.exe et également un répertoire %userprofile%\.dotnet\tools\.store\dotnet-ef.
Après avoir supprimé le fichier dotnet-ef.exe, le répertoire dotnet-ef et relancé l'installation, les commande dotnet ef fonctionnaient correctement.

Support de Cosmos Db

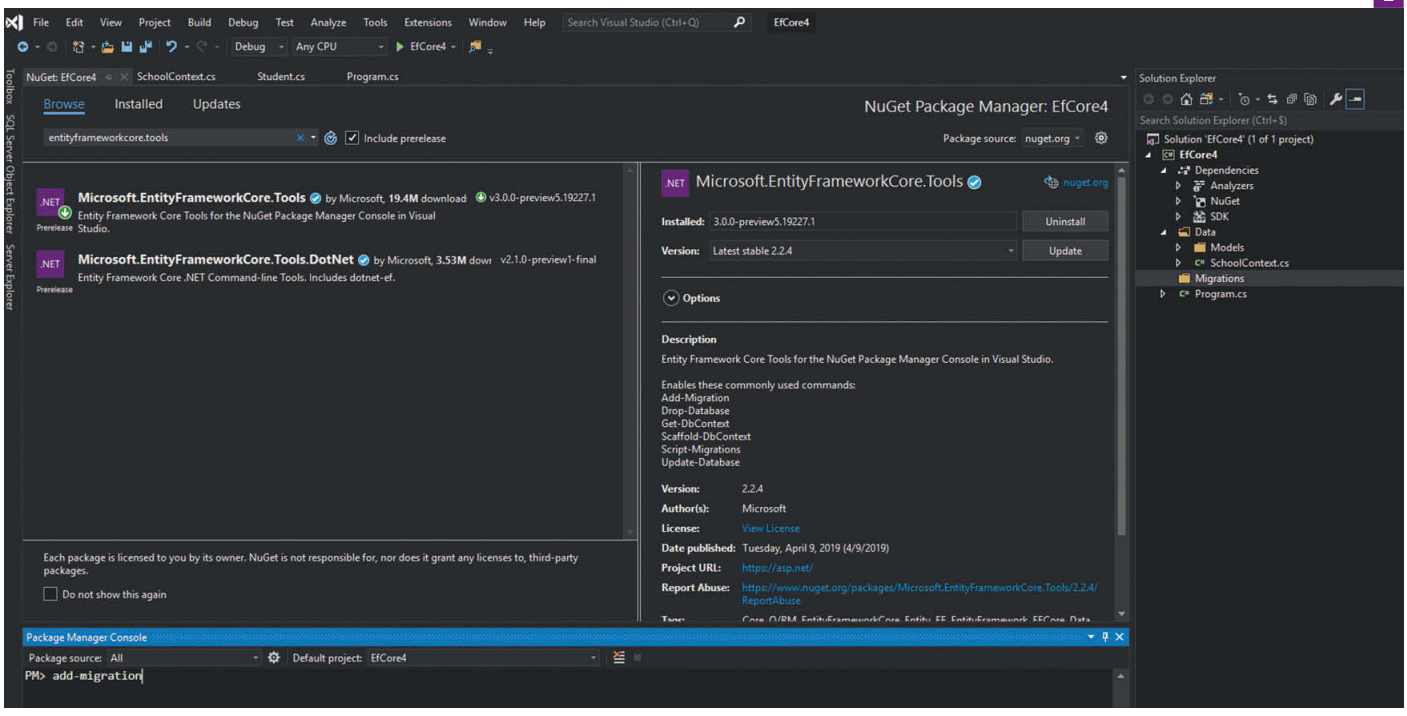
Microsoft travaille sur cette fonctionnalité depuis EF Core 2.2 en preview. D'ici la version RTM le support complet de Cosmos Db devrait être terminé.

```
Install-Package Microsoft.EntityFrameworkCore.Cosmos
```

Ce package permettra aux développeurs familiarisés avec le modèle de programmation EF de cibler facilement Azure Cosmos DB en tant que base de données d'application. Le fournisseur activera la plupart des fonctionnalités d'EF Core, telles que le suivi automatique des modifications, LINQ et les conversions de valeurs. La base de données Cosmos Db sera interrogée au travers de l'API SQL de la base de données Cosmos Db. Il s'agit d'une API qui permet de faire des requêtes presque similaires au SQL comme on le ferait pour SQL Server.
Azure Cosmos DB est un système propriétaire de Microsoft, qui fournit différents type de bases de données. 2
Il est assez facile de changer votre fournisseur de base de données. Dans la surcharge de la méthode *OnConfiguring*, remplacez *UseSqlServer* par *UseCosmos*.

```
public class SchoolContext : DbContext
 {
   public DbSet<Student> Students { get; set; }
```

1



```
    public SchoolContext()
    {
      Database.EnsureCreated();
    }
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
     optionsBuilder.UseCosmos("https://localhost:8081", "C2y6yDjf5/R+ob0N8A7Cgv30VRD
JIWEHLM+4QDU5DE2nQ9nDuVTqobD4b8mGGyPMbIZnqyMsEcaGQy67XIw/Jw==", "Student");
    }
  }
```

Pour tester ce fournisseur vous pouvez installer Azure Cosmos DB Emulator. 3

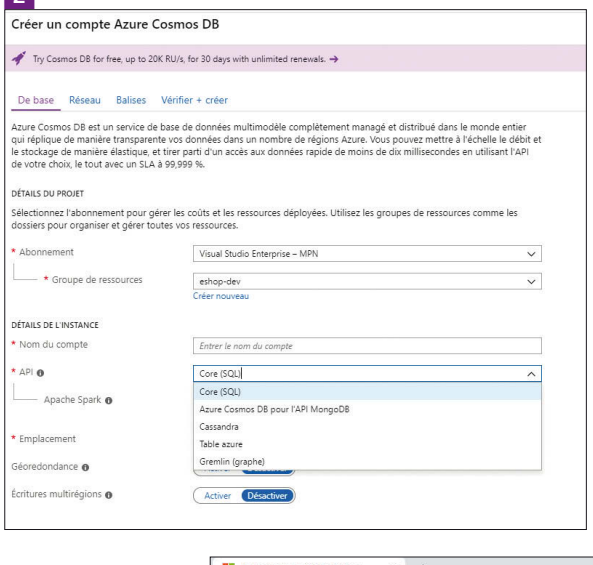
Voici un exemple d'utilisation du Context avec l'ajout d'un Student.

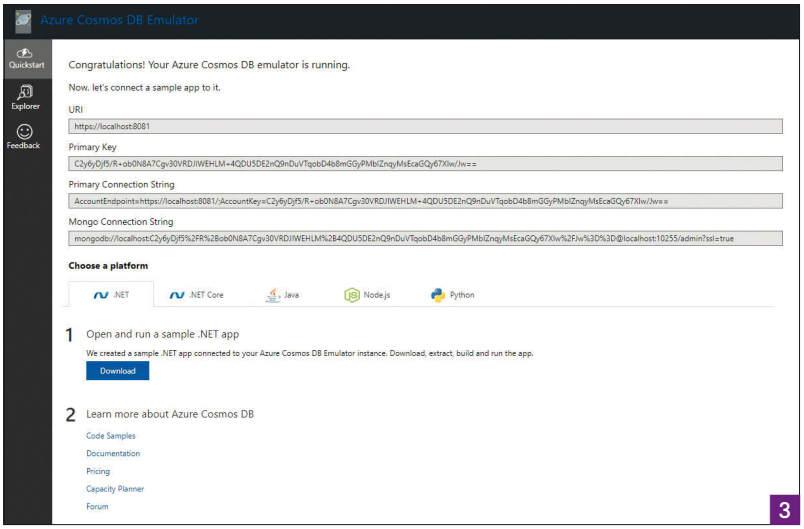
```
class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Hello World. EF Core!");

      using (var context = new SchoolContext())
      {
        var newStudent = new Student()
        {
          Name = "Cédric"
        };

        context.Students.Add(newStudent);
        context.SaveChanges();

      }

      Console.ReadKey();
    }
  }
```

Voici le résultat. 4

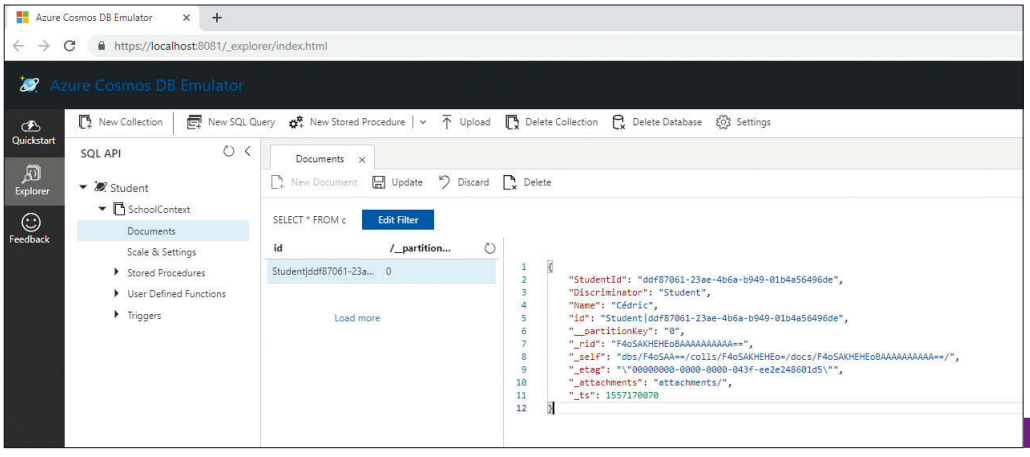
A l'heure où je finis d'écrire cet article, la version preview 5 est disponible. Mais cependant la majorité des fonctionnalités ne sont que partiellement terminées. Exemple, pour Cosmos Db le support n'est pas encore complet et tous les types de requêtes plus avancées ne sont pas encore supportés. Ce qui n'empêche pas que la majorité des fonctionnalités de base soient déjà stables. •

2





3



4

Vincent Rivière
Développeur à l'Université Paris 1 Panthéon-Sorbonne
http://vincent.riviere.free.fr/
https://www.youtube.com/c/Vretrocomputing

niveau 200

Gestion de la mémoire sur Atari ST

Partie 4

Dans les épisodes précédents, je vous ai donné un aperçu du TOS, le système d'exploitation de l'Atari ST. Aujourd'hui, nous allons étudier en détail une question essentielle : la gestion de la mémoire.

En 1985, les premiers Atari ST disposaient de 512 Ko de RAM. Cela peut sembler dérisoire aujourd'hui, mais à l'époque, c'était énorme. Et la machine était extensible jusqu'à 4 Mo de RAM, une quantité qui semblait alors gigantesque. Par la suite, l'Atari TT s'est positionné en modèle haut de gamme : de 2 à 10 Mo de RAM polyvalente, renommée ST-RAM pour l'occasion. A cela pouvaient s'ajouter plusieurs dizaines de Mo de TT-RAM, une mémoire rapide mais accessible uniquement par le processeur. Quelle que soit la quantité de mémoire disponible, celle-ci se gère toujours de la même manière depuis vos programmes. Avec toutefois quelques précautions à prendre pour pouvoir profiter de la TT-RAM.

Il y a 2 mois, j'ai détaillé la manière dont le TOS chargeait les programmes en mémoire. Pour rappel, voici les points essentiels. Le système alloue le plus gros bloc de mémoire libre et le réserve pour le nouveau processus. Cet espace est appelé **TPA** (Transient Program Area) 1. Au début, on trouve une structure de 256 octets appelée **BASEPAGE** (ou parfois PD, comme *Process Descriptor*) qui contient les attributs du processus. Viennent ensuite les segments **TEXT**, **DATA** et **BSS**. Après cela, on trouve une grande zone inutilisée appelée **tas** (heap), et le pointeur de **pile** (stack) est positionné tout à la fin. Notez que sur 68000, comme sur la plupart des processeurs, la pile croît vers les adresses basses.

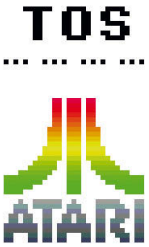
Un tas inutilisé ?

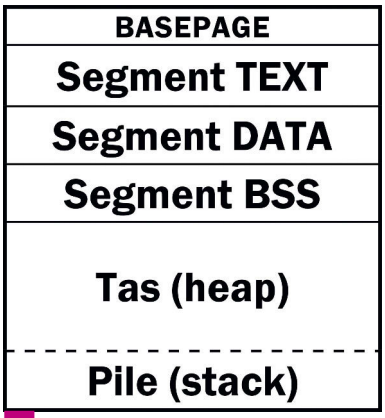
Eh oui, même si de nos jours cela paraît être une hérésie, le système alloue le plus gros bloc de RAM disponible pour l'affecter au nouveau processus. Il réserve un tas immense… mais ne l'utilise pas. Et pire encore, il ne fournit aucune fonction pour l'exploiter. Je suppose que c'est un vestige des anciens systèmes d'exploitation 8 bits, où un seul programme résidait en mémoire à un moment donné. Mais ce principe n'est pas du tout adapté aux systèmes modernes où la règle est d'utiliser la mémoire le plus parcimonieusement possible. Comme ce tas appartient au processus, on peut l'utiliser comme on le souhaite, de la fin du segment BSS jusqu'au début de la pile. Cela représente un espace de plusieurs centaines de Ko, voire plus, selon la quantité de mémoire installée sur la machine. Notez que la limite entre le tas et la pile n'est pas clairement définie. On pourrait même considérer que le tas n'est qu'une vue de l'esprit, et qu'en fait, le processus dispose simplement d'une immense pile. Alors qu'en pratique, une pile de quelques Ko suffirait amplement.

Mais revenons à notre programme. Le système lui a alloué un bloc de mémoire, et a positionné le pointeur de pile tout à la fin. Même si conceptuellement ce n'est pas très élégant, cette configuration reste parfaitement valide. C'est notamment le cas du programme « Hello, World! » que je vous ai présenté le mois dernier ; il ne se préoccupe aucunement de son tas, de sa pile, ou de la place qu'il prend en mémoire. D'ailleurs, combien de mémoire un tel programme occupe-t-il ? Je vous l'ai déjà dit : le système alloue **le plus gros bloc de mémoire libre**. Mais comme le TOS fait peu d'allocations dynamiques, la mémoire n'est généralement pas fragmentée. Donc le plus gros bloc de mémoire libre recouvre en fait… l'intégralité de la RAM disponible.

ADRESSAGE LINÉAIRE

Le processeur Motorola 68000 qui équipe l'Atari ST dispose d'un bus d'adresses sur 24 bits (et même sur 32 bits à partir du 68020). Cela lui permet d'accéder à l'intégralité de la mémoire en un seul bloc contigu. C'est très pratique et intuitif à utiliser. Souvenez-vous qu'à la même époque, les programmeurs PC s'arrachaient les cheveux à cause des segments de 64 Ko des processeurs Intel.





1 *La TPA d'un processus GEMDOS.*

Toute la RAM ?

Oui, vous avez bien lu. Le simple fait de lancer un programme quelconque suffit en pratique à réserver toute la RAM du système. En conséquence, si on appelle la fonction *Malloc()* (GEMDOS $48) pour allouer ne serait-ce qu'un seul octet supplémentaire, elle échouera irrémédiablement en renvoyant un pointeur NULL. Vous pourriez me rétorquer que, puisque vous disposez déjà de toute la mémoire du système, vous n'en avez pas besoin de plus. Mais cette restriction s'applique aussi au système lui-même. Si vous essayez de lancer un processus fils avec la fonction *Pexec()* (GEMDOS $4B), cela va lamentablement échouer. Et vous savez déjà pourquoi : le système va tenter d'allouer le plus gros bloc de mémoire libre pour le nouveau programme. Mais comme il ne reste plus rien, ça ne peut pas aboutir. Idem si vous tentez d'appeler la fonction *fsel_input()* (AES 90) pour afficher le sélecteur de fichier : au lieu de cela, vous obtiendrez une boîte d'alerte indiquant que le système n'a plus assez de mémoire.

Que faire ?

Rassurez-vous, la solution est facile à mettre en œuvre. Il s'agit de la fonction **Mshrink()** (GEMDOS $4A). Elle permet de **réduire la taille d'un bloc de mémoire** précédemment alloué par *Malloc()*. Et ça tombe bien, car le système a justement alloué notre bloc de

mémoire, la TPA, avec cette fonction avant de nous donner la main. Mais pour pouvoir appeler *Mshrink()*, nous avons besoin de deux informations : l'adresse de début du bloc, et la nouvelle taille souhaitée.

Je vous l'ai dit la dernière fois : juste avant d'exécuter un programme, le système empile l'adresse de la BASEPAGE, puis une fausse adresse de retour sous la forme d'un pointeur NULL. Donc au début du programme, l'expression **4(sp)** nous fournit… l'adresse de notre BASEPAGE. Et rappelez-vous, la BASEPAGE est toujours située au début de notre bloc TPA. Bingo ! C'est donc la fameuse adresse dont nous avons besoin.

Ensuite, comment calculer la taille idéale de notre TPA ? D'abord, il va falloir s'occuper de la pile. Je vous rappelle qu'elle est initialement située après le tas, tout à la fin du bloc TPA. Si on réduit ce dernier sans précaution, la pile va se retrouver en dehors de l'espace de notre processus, et dès que l'on empilera quoi que ce soit, on risquera d'écraser des données qui ne nous appartiennent pas. Il nous faut donc adopter une stratégie fiable et robuste. Le plus simple, c'est de réserver un espace pour la pile dans notre segment BSS. Au début du programme, il suffit d'initialiser le registre SP avec l'adresse de fin de notre nouvelle pile. Mais attention ! Avant d'abandonner la pile dont le système nous a doté, il faut récupérer toutes les données utiles qui sont dessus. En l'occurrence : l'adresse de notre BASEPAGE.

Donc le début de notre programme va ressembler à ceci :

```
    move.l   4(sp),a0        ;BASEPAGE
    move.l   #finpile,sp     ;Nouvelle pile
    ; La suite viendra s'insérer ici
    BSS
pile:  ds.b  4096            ;4 Ko
finpile:
```

Nous avons récupéré l'adresse de la BASEPAGE sur l'ancienne pile, et nous l'avons sauvegardée dans le registre A0. Par ailleurs, nous avons réservé de l'espace pour notre nouvelle pile dans le segment BSS. Il suffit donc d'initialiser le registre SP avec l'adresse de fin de notre pile, et le tour est joué. Du moins, pour cette première phase.

A présent, la situation est la suivante. Notre nouvelle pile réside dans le segment BSS. Ensuite, il y a notre tas, que nous n'avons pas l'intention d'utiliser. Et à la fin, il y a notre ancienne pile, que nous avons abandonnée. Nous pouvons donc sans risque réduire notre bloc TPA jusqu'à la fin du segment BSS, puisque ce qui est situé au-delà ne nous intéresse plus.

Nouvel objectif : déterminer la taille idéale de notre bloc TPA pour qu'il se termine juste à la fin du segment BSS. On se place dans le cas général où il y a potentiellement d'autres variables définies ailleurs dans le segment BSS. Toutes les informations relatives aux segments se trouvent dans la BASEPAGE. Et ça tombe bien, car nous venons justement d'en récupérer l'adresse.

Je vous conseille d'aller consulter la documentation de la BASEPAGE sur « **tos.hyp** » [1]. Section *GEMDOS*, puis *GEMDOS structures*, puis *Process Descriptor (PD) resp. BASEPAGE*. Cette structure est décrite en langage C, et malheureusement les offsets nécessaires à son utilisation en assembleur ne sont pas indiqués. On peut toutefois les calculer pour chaque membre en additionnant la taille des champs qui précèdent. Voici les champs qui nous intéressent. Ce sont tous des longs mots.

- offset 12 : p_tlen, taille du segment TEXT
- offset 20 : p_dlen, taille du segment DATA
- offset 28 : p_blen, taille du segment BSS

La seule information qui n'est pas décrite ici, c'est la taille de la BASEPAGE. Mais comme je vous l'ai déjà dit, elle a toujours une taille fixe de 256 octets. Donc c'est bon, on a tous les éléments pour calculer la taille optimale de notre TPA ! Souvenez-vous, nous avons stocké l'adresse de la BASEPAGE dans le registre A0. Nous pouvons donc faire la somme de toutes les tailles de cette manière :

```
move.l   #256,d0     ;  taille de la BASEPAGE
add.l    12(a0),d0   ;+ taille du segment TEXT
add.l    20(a0),d0   ;+ taille du segment DATA
add.l    28(a0),d0   ;+ taille du segment BSS
```

On obtient la taille souhaitée de notre TPA dans le registre D0. Comme l'adresse de la BASEPAGE est aussi celle du début de la TPA, nous avons tous les éléments nécessaires pour faire appel à *Mshrink()*.

```
move.l   d0,-(sp)    ;Nouvelle taille de la TPA
move.l   a0,-(sp)    ;Début du bloc TPA
clr.w    -(sp)       ;Paramètre inutilisé, toujours 0
move.w   #$4a,-(sp)  ;Mshrink()
trap     #1          ;GEMDOS
add.l    #12,sp      ;Dépiler les paramètres
```

Et voilà ! Nous avons réduit la taille de notre TPA au strict minimum, et l'excédent a été rendu au système. Nous pouvons maintenant faire appel à toutes les fonctions qui nécessitent de la RAM supplémentaire telles que *Malloc()*, *Pexec()*, *fsel_input()*… Cette petite gymnastique peut vous paraître fastidieuse, mais sachez qu'en pratique la quasi-totalité des programmes pour Atari ST débutent de cette manière. Ainsi, chaque processus ne consomme que la mémoire qui lui est nécessaire. Seul défaut mineur de cette solution : comme notre nouvelle pile réside dans le segment BSS, son contenu est initialisé à zéro, alors que ce n'est pas strictement nécessaire. Mais comme cette pile fait seulement quelques Ko, le temps passé par le système pour la remplir de zéros est vraiment négligeable. Compte tenu de ses avantages, cette solution reste ma favorite.

QUELLE EST LA TAILLE IDÉALE DE LA PILE ?

Si vous restez en mode utilisateur, votre pile restera strictement privée pour votre processus. En théorie, quelques dizaines d'octets devraient suffire : assez d'espace pour les quelques paramètres que vous allez empiler, plus les adresses de retour des fonctions appelantes. Il pourrait aussi y avoir quelques variables locales, mais contrairement au C, elles sont rarement utilisées en assembleur. Donc pour un usage basique, une pile de 4 Ko est largement suffisante. En revanche, si vous passez en mode superviseur, la situation est différente : dans ce cas, les interruptions s'exécutent sur votre pile, ainsi que les traps : prévoyez large ! Si votre pile est trop petite, elle risque de déborder en écrasant votre segment BSS et peut-être même plus : c'est le fameux « stack overflow ». Plantage assuré, avec des erreurs incompréhensibles à la clé.

Solution alternative

On l'a vu, juste après le segment BSS il y a un tas immense et inutilisé. Donc au lieu de mettre notre nouvelle pile dans le segment BSS, on peut simplement la placer au début du tas. Elle sera hors de tout segment, mais comme ce tas nous appartient, cela reste valide.

```
move.l   4(sp),a0      ;BASEPAGE

move.l   #256,d0       ;  taille de la BASEPAGE
add.l    12(a0),d0     ;+ taille du segment TEXT
add.l    20(a0),d0     ;+ taille du segment DATA
add.l    28(a0),d0     ;+ taille du segment BSS
add.l    #4096,d0      ;+ taille de la nouvelle pile

move.l   a0,a1         ;  BASEPAGE
add.l    d0,a1         ;+ nouvelle taille TPA
move.l   a1,sp         ;= nouvelle pile

move.l   d0,-(sp)      ;Nouvelle taille de la TPA
move.l   a0,-(sp)      ;Début du bloc TPA
clr.w    -(sp)         ;Paramètre inutilisé, toujours 0
move.w   #$4a,-(sp)    ;Mshrink()
trap     #1            ;GEMDOS
add.l    #12,sp        ;Dépiler les paramètres
```

Cette solution est très utilisée, mais personnellement je ne l'aime pas car elle contient une faille. On part du principe que le tas est suffisamment grand pour contenir la nouvelle pile. Mais rien n'est moins sûr. Nous savons juste que le système a alloué un bloc de mémoire suffisamment gros pour y faire tenir les segments de notre programme, et peut-être un peu plus. Mais rien ne nous indique a priori la taille de l'espace supplémentaire tas+pile. Donc si l'on veut être rigoureux, après avoir calculé l'adresse de notre nouvelle pile dans A1, il faut s'assurer qu'elle ne dépasse pas la valeur du précédent SP ! Car si c'est le cas, cela signifie que la TPA n'est pas suffisamment grande pour nos besoins. Il faut alors afficher un message d'erreur et quitter le programme prématurément. Mais soyons clairs, il y a très peu de chances pour que ce cas défavorable se produise en pratique. Il faudrait que le programme en cours de chargement soit à peine plus petit que le plus gros bloc de mémoire libre. Vu la taille des programmes sur Atari (plutôt petite) et la quantité de mémoire disponible (plutôt grande), ce cas est assez improbable. Quoi qu'il en soit, avec l'autre solution (pile dans le segment BSS), ce cas limite ne peut jamais se produire car s'il n'y a pas assez de mémoire, le programme n'arrive même pas à se charger.

Fastload

Croyez-le ou non : plus il y a de RAM installée sur un Atari, et plus les programmes sont longs à se charger. Du moins, par défaut. Sur un ST avec 4 Mo de RAM, ça commence à se sentir : il y a une pause d'environ une seconde entre le moment où le programme a fini de se charger et le moment où il démarre réellement. Vous voulez savoir pourquoi ? Alors, tenez-vous bien. Vous vous souvenez de ce fameux tas immense et inutilisé ? Eh bien sachez qu'au démarrage d'un programme, le TOS initialise l'intégralité de ce tas **à zéro**. Et puisqu'en pratique ce tas occupe tout l'espace libre, plus il y a de RAM dans le système, plus ça prend du temps. Et avec un simple 68000 à 8 MHz, ce n'est pas négligeable. Heureusement, à partir du TOS 1.4, il existe un remède : **le flag Fastload**.

Il y a deux mois, je vous ai dit que les exécutables TOS commençaient toujours par un en-tête de 28 octets. A l'offset $16 se trouve un long mot appelé **PRGFLAGS**. Par défaut, sa valeur est 0, mais chaque bit a une signification particulière. Le bit 0 s'appelle Fastload. S'il vaut 1, alors le TOS 1.4 (et supérieurs) ne remplira pas le tas du programme avec des zéros. Le tas nous sera fourni tel quel, non-initialisé. Seul le segment BSS sera initialisé à zéro, comme il se doit. Atari a fourni un utilitaire appelé **MAKEFAST.PRG** qui permet de facilement modifier le bit Fastload d'un programme existant. En pratique, quasiment tous les programmes fonctionnent correctement avec le bit Fastload à 1. Ainsi, ils démarrent immédiatement, sans délai supplémentaire, quelle que soit la quantité de RAM installée sur la machine. Donc je ne peux que vous recommander d'activer le bit Fastload sur tous vos programmes, et de ne revenir en arrière que dans les très rares cas où cela provoque un dysfonctionnement.

Sachez qu'avec Devpac 2, on peut facilement définir la valeur du champ PRGFLAGS dans un programme assembleur. Il faut utiliser la directive **COMMENT HEAD** dans l'un des sources. Par exemple, pour activer le bit Fastload, la syntaxe est : « COMMENT HEAD=1 ». Facile et efficace. De son côté, GCC active automatiquement le bit Fastload par défaut. On peut le modifier à tout moment avec la commande « **flags** » fournie avec le paquet MiNT-Bin natif, ou son équivalent « m68k-atari-mint-flags » fourni avec les cross-tools. L'option --mfastload (ou -l) permet d'activer le bit, alors que l'option --mno-fastload permet de le désactiver.

Et la TT-RAM ?

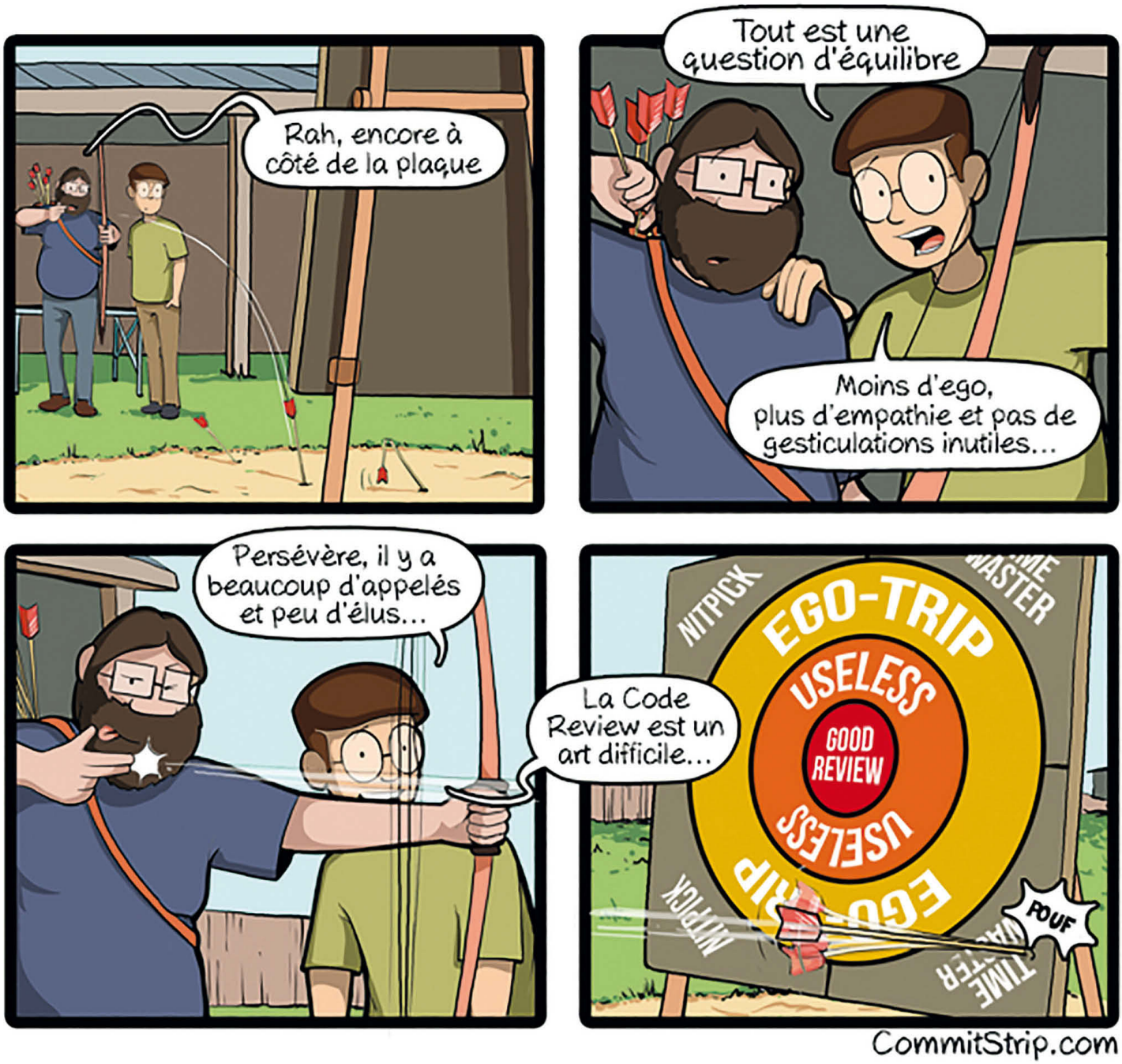
C'est un sujet que nous pourrons aborder en détail une autre fois, mais pour ne pas vous faire languir, voici l'essentiel. Par défaut, pour éviter tout problème de compatibilité, les programmes n'utilisent pas la TT-RAM. On peut la rendre accessible en activant des bits dans le champ PRGFLAGS de l'en-tête des programmes, de la même manière que le Fastload. Le bit 1 autorise le programme lui-même à être chargé en TT-RAM, et le bit 2 permet à *Malloc()* de faire des allocations dans la TT-RAM. Pour résumer, **l'idéal est d'initialiser les PRGFLAGS avec la valeur 7**, afin de mettre à 1 tous les bits dont nous venons de parler. C'est d'ailleurs ce que fait GCC par défaut. Et si un programme souhaite choisir finement quel type de RAM utiliser, il peut utiliser la fonction *Mxalloc()* (GEMDOS $44) prévue à cet effet.

Conclusion

Nous venons de passer en revue les principes fondamentaux de la gestion de la mémoire sur Atari ST. Il est essentiel de les avoir en tête pour pouvoir écrire des programmes propres qui se comportent correctement dans tous les cas de figure, quelle que soit la version du TOS ou même de FreeMiNT. Rassurez-vous, avec les langages de haut niveau tels que BASIC et C, vous n'avez pas à vous préoccuper de tout ça : ces détails techniques sont gérés automatiquement. Mais en assembleur, vous avez le plaisir de maîtriser complètement tous les aspects de vos programmes. •

[1] https://freemint.github.io/tos.hyp/

un art difficile...



Une publication Nefer-IT, 57 rue de Gisors, 95300 Pontoise - redaction@programmez.com
Tél. : 09 86 73 61 08 - Directeur de la publication & Rédacteur en chef : François Tonic
Secrétaire de rédaction : Olivier Pavie
Ont collaboré à ce numéro : ZDNet, Mobioos, Hired
Nos experts techniques : B. Nachawati, J-B Boichat, F. Loison, A-C. Bournigal, V. Caron, D. Dhouib, C. Pichaud, J-B. Bron, S. Houel, L. Grangeau, H. Azzouz, C. Michel, V. Rivière, Commitstrip
Couverture : Iloyd, Amazon, NodeMCU - **Maquette** : Pierre Sandré.
Publicité : François Tonic / Nefer-IT - Tél. : 09 86 73 61 08 - ftonic@programmez.com.
Imprimeur : S.A. Corelio Nevada Printing, 30 allée de la recherche, 1070 Bruxelles, Belgique.
Marketing et promotion des ventes : Agence BOCONSEIL - Analyse Media Etude - Directeur : Otto BORSCHA oborscha@boconseilame.fr
Responsable titre : Terry MATTARD Téléphone : 09 67 32 09 34

Contacts : Rédacteur en chef : ftonic@programmez.com - Rédaction : redaction@programmez.com - Webmaster : webmaster@programmez.com
Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1220K78366 - ISSN : 1627-0908 - © NEFER-IT / Programmez, juin 2019

Abonnement :
Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex. - Tél. : 01 55 56 70 55 - *abonnements.programmez @groupe-gli.com*
Fax : 01 55 56 70 91 - du lundi au jeudi de 9h30 à 12h30 et de 13h30 à 17h00, le vendredi de 9h00 à 12h00 et de 14h00 à 16h30.

Tarifs
Abonnement (magazine seul) : 1 an - 11 numéros France métropolitaine : 49 € - Etudiant : 39 € CEE et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 € - Autres pays : nous consulter.

PDF
35 € (monde entier) souscription sur www.programmez.com

INFORMER
pour transformer l'entreprise

La dématérialisation, le Cloud, les communications unifiées, les nécessités de la cybersécurité transforment le travail et toute l'entreprise, les services publics. Le magazine, le site, ses newsletters vous informent sur cette actualité mouvante et vous aident à décoder les tendances

Abonnez-vous

www.solutions-numeriques.com/abonnement/

4 sites thématiques, pour répondre à vos besoins d'information

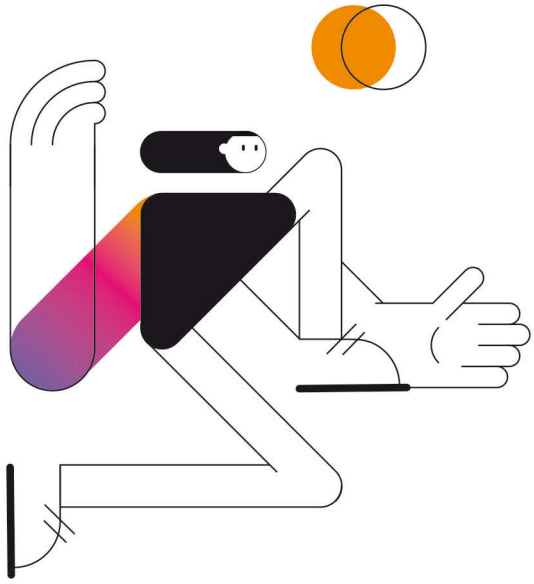
- Vous êtes responsable informatique ou bien dirigeant ou cadre d'entreprise ? *2 sites répondent à votre profil*
- La cybersécurité vous concerne ? *Cliquez sur l'onglet. Vous trouverez les infos, l'annuaire, le lexique, etc*
- L'emploi, les salaires, les formations, les offres vous intéressent ? *Le site sur l'Emploi dans le numérique est à votre disposition*

www.solutions-numeriques.com

Programmes de Support à la Communauté

Chez JetBrains nous créons des outils de développement intelligents, parmi lesquels IntelliJ IDEA, PhpStorm, PyCharm et ReSharper, plébiscités par des millions de développeurs dans le monde.

Nous pensons qu'il est essentiel de soutenir activement la communauté des développeurs et l'open source.



Licences gratuites pour :

- Les groupes d'utilisateurs.
- Les projets open source.
- Les événements de la communauté.
- Les développeurs experts.

3,000+
Événements de la communauté et groupes d'utilisateurs ont bénéficié de licences JetBrains gratuites.

10,000+
Projets open source ont reçu des licences produits JetBrains gratuites.

1,300+
Microsoft MVPs, Google Developers Experts et Java Champions ont été récompensés par un abonnement gratuit à notre offre All Products Pack.

Plus d'informations sur
jetbrains.com/community/support/

Contactez-nous
community-support@jetbrains.com