PROGRAMMEZ!

PROGRAMMEZ!

Le magazine des développeurs

N°247
07/08
2021

CRÉER SON JEU 2D AVEC DELPHI

SPÉCIAL été 2021

Créer son template Wordpress

Un robot pour son Rubik's Cube

De Java 9 à Java 15 : les nouveautés

LA MACHINE DE TURING

Le seul magazine écrit par et pour les développeurs

M 04319 - 247 - F: 6,99 € - RD

Printed in EU - Imprimé en UE - BELGIQUE 7,50 € - Canada 10,55 $ CAN - SUISSE 14,10 FS - DOM Surf 8,10 € - TOM 1100 XPF - MAROC 59 DH

DÉVELOPPEZ 10 FOIS PLUS VITE

WINDEV Tech Tour 26

926 NOUVEAUTÉS

NOUVELLE VERSION

REGARDEZ LES REPLAYS

CETTE ANNÉE, LE **WINDEV TECH TOUR** S'EST DÉROULÉ
SOUS FORME DE 3 WORKSHOPS EN LIGNE

REGARDEZ LES REPLAYS DES WORKSHOPS SUR
pcsoft.fr/windev/videos.htm

WWW.PCSOFT.FR

WINDEV, AGL DEVOPS LOW CODE

Contenus

Divers

Programmez! est une publication bimestrielle de Nefer-IT.
Adresse : 57, rue de Gisors 95300 Pontoise – France. Pour nous contacter : redaction@programmez.com

EDITO#247*

Le Rust de la colère

**en réalité, nous sommes au numéro 250*

Depuis plusieurs mois, il y a des débats autour du noyau Linux et de l'utilisation du langage Rust. Le premier acte se déroule au printemps : Google veut améliorer la sécurité et la stabilité d'Android. Pour l'éditeur, les problèmes mémoires en C et C++ sont toujours là et difficile à fixer. La décision est d'utiliser Rust, car il apporte une meilleure sécurité mémoire selon les experts Google.

Mais, Google avait tout de même précisé : RUST, oui, mais uniquement pour les nouveaux développements. Tout le code existant restera. Ouf ! Car on ne parle pas de quelques milliers de lignes, mais de millions ! Réécriture un composant aussi gros et critique serait un suicide technique et il faudrait plusieurs années pour tout réécrire et stabiliser le code, sans garantir un fonctionnement identique.

Mi-juin, Google a réaffirmé son ambition : pour moderniser et améliorer la sécurité de Linux et par conséquent des systèmes mobiles et d'Internet. L'idée est la même que celle émise au printemps : écrire les nouveaux composants du noyau en Rust pour fournir une fondation fiable et prévisible. Linus Torvalds, le créateur de Linux, n'était pas contre et avait même fait une sévère critique de C++ (Linus n'est jamais contre une bonne petite critique). Le projet Rust for Linux a été lancé pour ajouter le langage dans le noyau. Cependant, Linus a aussi dit qu'il ne pousserait pas Rust et qu'il attendait de voir comment l'implémentation se ferait (il peut aussi changer d'avis). Une des premières utilisations pourrait être l'écriture de pilotes. Cette utilisation est possible uniquement si Rust est supporté par le noyau.

Google n'a jamais hésité à pousser de nouveaux langages : Java puis Kotlin, Dart avec Flutter ou encore Go. Pour Google, il s'agit donc d'un langage supplémentaire et depuis la mésaventure du procès autour de codes Java, on peut le comprendre.

Reste aussi à convaincre la communauté et les « gardiens » du noyau car nous ne voyons pas de consensus. Et beaucoup attendent de voir pour se positionner.

Miguel Ojeda a été choisi pour mener cette quête. Dans un post, Miguel précise que l'idée d'avoir un autre langage dans le noyau est une bonne chose, même si la tâche n'est pas simple.

Il liste même une série d'arguments, par exemple :

- Le code Rust réduirait les risques mémoires, les data races, les bugs logiques
- La maintenance serait plus efficace
- Écrire plus simplement les nouveaux pilotes et modules
- Encourager les développeurs à participer au noyau avec un langage plus « simple » et surtout plus moderne.

Source : https://lkml.org/lkml/2021/4/14/1023

Reste à voir comment ce projet pourrait aboutir et comment le support du langage se fera. La qualité de l'implémentation sera primordiale. Il y a un an, un développeur de Qemu, Stefan Hajnoczi, avait publié un long post sur : pourquoi Quemu devrait être passé de C à Rust. Il mettait en avant les mêmes arguments sur la sécurité et la gestion mémoire et l'utilité du langage pour coder des outils de monitoring de machines virtuelles.
Qemu s'est plus de 1,5 million de lignes de code. Stefan concluait que migrer un code aussi important est très difficile et qu'il ne le fera pas et se pose le temps de la transition et d'une double maintenance du code.

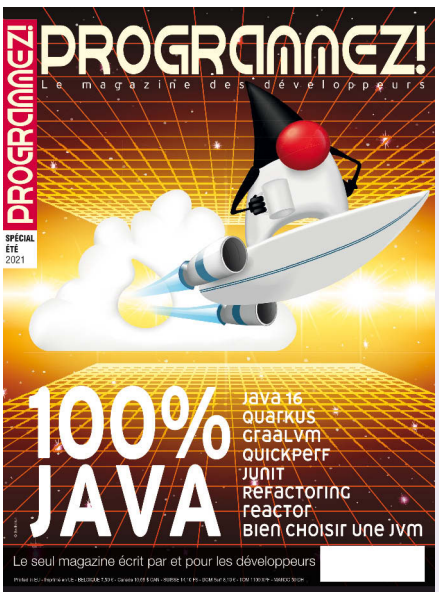
Changer pour changer ?

Ce débat nous en évoque une autre : changer de frameworks et de langages pour une app, un projet, bonne ou mauvaise idée ? On pourrait aussi résumer ainsi : quel framework JS ? Un truc connu, maintenu et qui ne devrait pas disparaître dans les 2-3 ans ? Ou un framework tout jeune, avec une faible communauté mais prometteur ?

Faut-il réécrire du code existant et en production qui fait ce qu'il doit faire (ce qui est une bonne nouvelle) ? Ou réserver les nouvelles technologies aux nouveaux projets ? Dans une autre vie, je l'ai vécu à plusieurs reprises : réécriture totale du progiciel avec une techno plus récente.
Heureusement, ces changements étaient réalisés tous les 4-5 ans avec l'ajout de nouvelles contraintes, par exemple : assurer la portabilité du logiciel en dehors de Windows (ce qui imposait un nouveau langage).
Que peut-on changer dans un projet existant sans risquer de casser son fonctionnement ? Dans un front-end web, on pourrait se dire : « bah on prend le framework JS, on en installe un autre, on change quelques codes et en quelques jours, c'est en prod ! ». Bien entendu, dans la réalité, dans 99 % des cas, les choses ne sont pas aussi simples. Et ces changements doivent avoir une logique, s'inscrire dans une stratégique claire sur x années.

Nous reviendrons sur ce passionnant débat dans les prochains numéros.

François Tonic
Rédacteur en chef depuis longtemps



LES PROCHAINS NUMÉROS

HORS SÉRIE #4 ÉTÉ

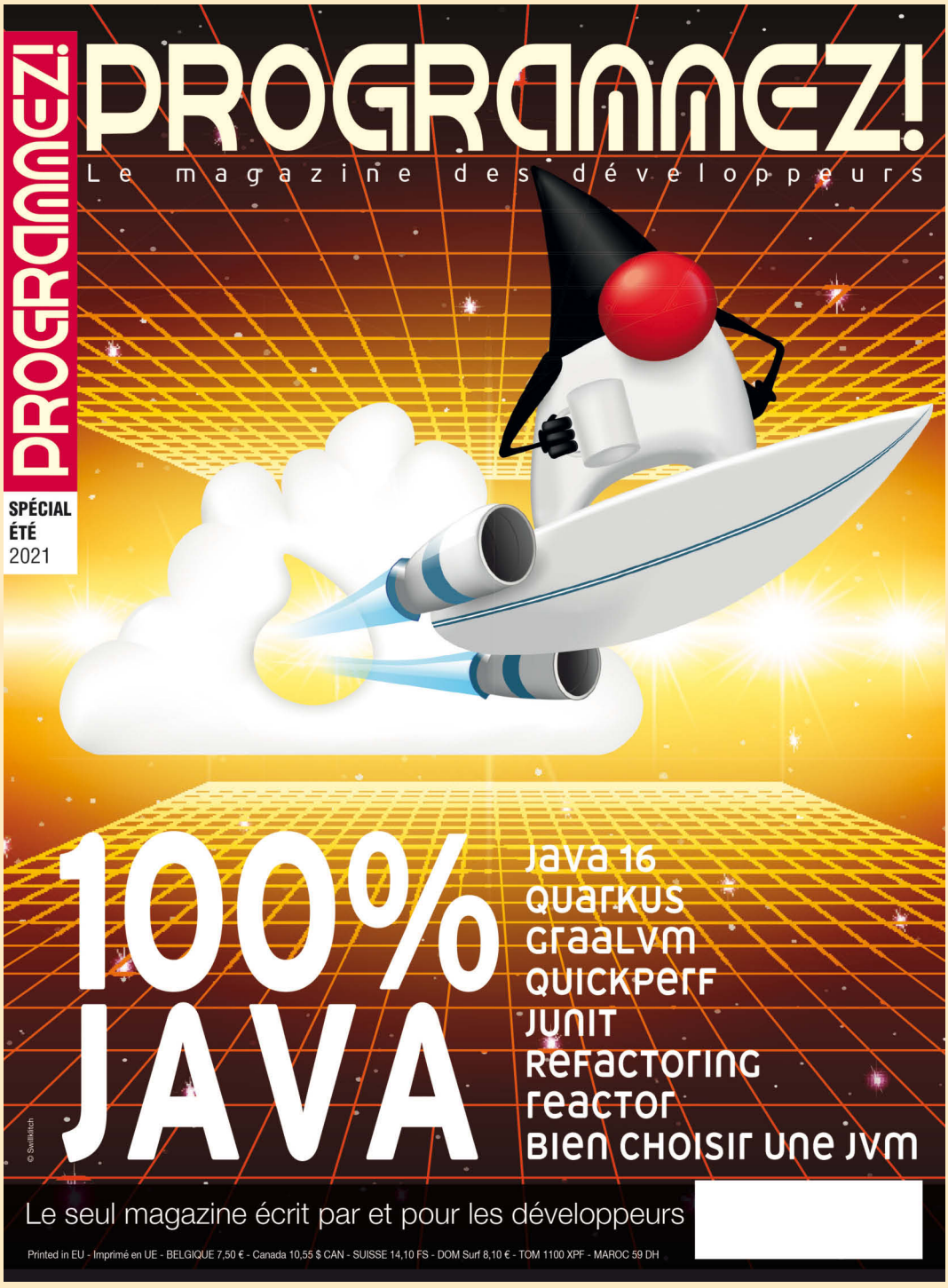
100 % JAVA

Disponible le 9 juillet 2021

Programmez! n°248

Dossier quantique
Node/JS

Disponible le 3 septembre 2021



100% JAVA

disponible dès le 9 juillet 2021

Kiosque / Abonnement - Version papier / Version PDF

Les événements Programmez!

Meetups Programmez!
Nous débuterons la session 2021-2022, le 7 septembre !
19 octobre - 2 novembre - 7 décembre

Où : WeWorks, 33 rue Lafayette / Paris
Métros : Notre-Dame de Lorette (l12), Le Peletier (l7)
A partir de 18h30

DevCon by Programmez !
23 septembre : Spécial .Net 2e édition

Où : Epitech
1 journée - 12 sessions

Informations & inscription : programmez.com

septembre

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
		1	2	3	4	5
6	7	8	9	10	11	12
	Meetup Programmez!			API Platform Conference (Lille + virtuel)	Angers GeekFest / Angers	
				JUG Summer Camp / La Rochelle		
13	14	15	16	17	18	19
JFTL (Montrouge)						
	WAX, 100 % / Marseille					
20	21	22	23	24	25	26
			DevCon .Net 6 / Azure / Windows (Epitech Paris)			
27	28	29	30			
	Big Data Paris					
	Serverless Days Paris	Devoxx France (Paris)				

octobre

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
				1	2	3
				Devoxx France		
4	5	6	7	8	9	10
			Paris Web (Paris)			
			Cloud Bord			
11	12	13	14	15	16	17
18	19	20	21	22	23	24
	Meetup Programmez !		DevFest Nantes			
25	26	27	28	29	30	31

novembre

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
1	2	3	4	5	6	7
	Meetup Programmez!					
8	9	10	11	12	13	14
	Open Source Experience (Paris)					
	DevFest Strasbourg					
15	16	17	18	19	20	21
Hack in Paris (virtuel)			Codeurs en Seine (virtuel)	DevFest Lille		
22	23	24	25	26	27	28
Paris Test Conference 2021						
29	30	31				

RENDEZ-VOUS EN 2022

- DevOps Rex
- Sunny Tech
- NCrafts Paris
- BreizhCamp
- DevFest du bout du monde
- Best of Web
- Flutter Con Paris
- DevFest Toulouse

Merci à Aurélie Vache pour la liste 2021, consultable sur son GitHub : https://github.com/scraly/developers-conferences-agenda/blob/master/README.md

Les partenaires 2021 de

PROGRAMMEZ!

Le magazine des développeurs

Niveau maître Jedi

softfluent

LA MANUFACTURE CACD2

Niveau padawan

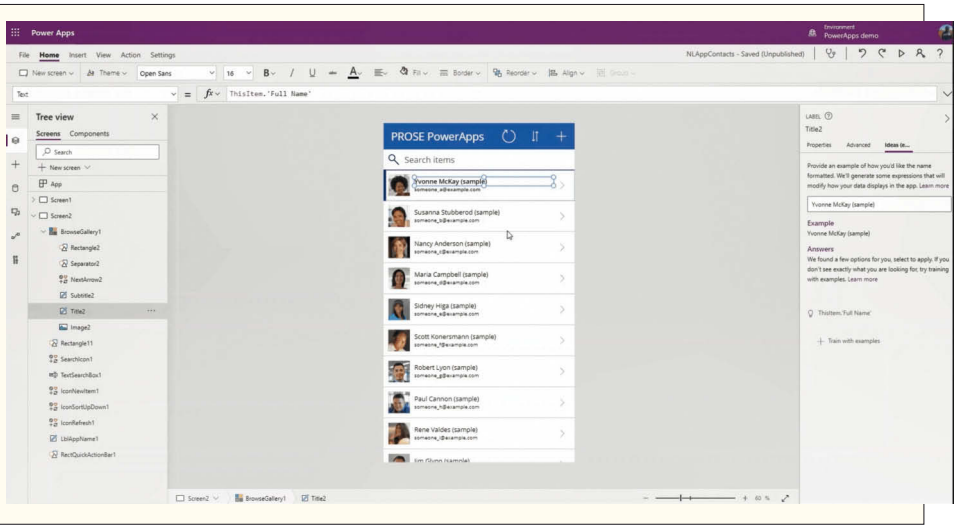
OSAXiS

Vous voulez soutenir activement Programmez! ?
Devenir partenaires de nos dossiers en ligne et de nos événements ?

Contactez-nous dès maintenant :
ftonic@programmez.com

Microsoft a trouvé ce qu'il voulait faire de GPT3

On vous en parlait il n'y a pas si longtemps : à force de généreux financements, Microsoft était parvenue à s'arroger une licence d'exploitation commerciale pour GPT3, le réseau neuronal aux capacités surprenantes en matière de rédaction de textes. À l'occasion de sa conférence BUILD 2021, Microsoft a commencé à dévoiler ce qu'il comptait faire de son nouveau jouet : l'éditeur l'intégrera à son langage de programmation PowerFx. L'objectif est de permettre aux utilisateurs de programmer en utilisant le langage naturel. GPT3 se chargera ensuite de retranscrire en code source compréhensible par la machine.



Ransomware : un pipeline américain paralysé pendant plusieurs jours

Un groupe de ransomware connu sous le nom de Darkside s'est taillé une réputation au mois de mai en s'attaquant à Colonial Pipeline, une société américaine chargée de l'approvisionnement de carburant. Le ransomware déployé sur les systèmes a contraint la société à bloquer pendant plusieurs jours le fonctionnement d'un de ses oléoducs les plus importants, au début du mois de mai. L'opérateur américain est parvenu à rétablir le service quelques jours plus tard, en se délestant au passage d'une rançon de 4,4 millions de dollars pour obtenir la clef de déchiffrement détenue par les cyberattaquants. Dans un retournement de situation de dernière minute, le FBI a annoncé être parvenu à récupérer une partie de la rançon payée par la victime : 63,7 des 75 bitcoins cédés par Colonial Pipeline ont ainsi été rendus à la société. Manque de chance, la valeur de la cryptomonnaie ayant chuté entre temps, cela ne représente plus que 2,3 millions de dollars. C'est toujours ça de pris.

Orange dans la tourmente après une panne

Ce n'est pas facile tous les jours d'être l'opérateur historique : Orange a essuyé une panne affectant les appels d'urgence, qui a empêché de nombreux appels à destination des pompiers ou des urgences d'être correctement rerouté pendant plusieurs heures, mercredi 2 juin, ce qui a valu au dirigeant de l'opérateur de se faire sérieusement rabrouer par le gouvernement. Selon les résultats d'un premier audit mené par Orange, un bug logiciel lors d'une mise à jour des serveurs utilisés pour rerouter les appels en interne est à l'origine du problème. Au moins, ce n'est pas une cyberattaque.

Sur le cloud, le gouvernement veut le beurre et l'argent du beurre

Le gouvernement a donné le coup d'envoi à son label « cloud de confiance », qui vise à identifier les offres de cloud certifiées par l'état. Ce label se repose pour une partie technique sur la qualification SecNumCloud, déjà proposée par l'Anssi, mais en ouvrant cette fois la porte à des acteurs américains. Ceux-ci pourront proposer leurs technologies au travers de partenariat avec des acteurs français, en hébergeant les données et les serveurs en France et en proposant des garanties juridiques pour limiter les risques. Plusieurs annonces en ce sens ont suivi :

VISA DE SÉCURITÉ — ANSSI

Microsoft proposera, par exemple, ses services à travers un partenariat avec Orange et Capgemini. Néanmoins, n'appelez pas ça du « cloud souverain », le souvenir de Numergy et Cloudwatt est encore trop douloureux.

Cookies : Noyb veut accélérer le rythme des autorités

noyb

Fin mai, la CNIL a mis en demeure une vingtaine d'organisations sur le recueil du consentement pour le dépôt de cookies. Il faut dire que la Commission avait donné ses lignes directrices en la matière en octobre 2020 et que, six mois plus tard, il fallait bien montrer un peu les muscles pour convaincre les traînards. Et à ce sujet, l'association Noyb a décidé de donner un coup de pouce aux autorités de protection des données en rappelant à l'ordre environ 500 sites web qui ne sont pas en règle. En l'absence de réponse, l'association indique qu'elle transmettra ses griefs aux autorités compétentes : elle explique avoir développé des outils permettant d'automatiser la procédure, et espère pouvoir « alerter » un peu plus de 10 000 sites sur l'année 2021.

RGPD : trois ans après, peu de satisfaits

Le RGPD fêtait les trois ans de son entrée en application le 25 mai, et force est de reconnaître que le bilan du règlement ne fait pas que des heureux. D'un côté, les entrepreneurs se plaignent d'un texte difficile à interpréter et qui les désavantagerait face à la concurrence. De l'autre, les militants déplorent l'inactivité des autorités de protection des données, qui rechignent à sanctionner les contrevenants, et le rôle ambigu de l'autorité irlandaise de protection des données, qui est chargée de faire appliquer le règlement auprès de nombreux acteurs américains majeurs de la tech. Au final, personne n'est vraiment content et certains anciens commissaires européens évoquent même l'idée d'une « refonte ».

François Tonic

La BUILD 2021 a été une fois de plus très riche en annonces. Sans revenir en détail sur tous les aspects, on pouvait tout de même cerner 2 axes importants :

- Les processeurs ARM
- Java sur Azure

Java est une cible prioritaire pour l'éditeur sur Azure. Plusieurs serveurs Java EE sont officiellement supportés et notamment Jakarta EE, le successeur de Java EE depuis qu'Oracle s'est désengagé de la plateforme. En entreprise, et particulièrement dans les grandes entreprises, Java, EE est toujours utilisé. L'autre annonce est la disponibilité d'OpenJDK. Cette distribution repose naturellement sur OpenJDK avec quelques patchs et rajouts de l'éditeur. Une bêta est disponible pour supporter Java 16.

L'éditeur en a profité pour annoncer que le C++ 20 est totalement supporté par le compilateur et la STL. Détail non négligeable : le runtime LLVM OpenMP supporte x86 et ARM64. D'autre part, Visual Studio 2022, dont une pré-version est attendue cet été, est attendue avec une certaine impatiente : 64 bits, support de CMake, Linux et WSL, .Net 6.

ARM est une cible claire de Microsoft. L'éditeur veut accélérer son support dans les outils de développement, dans Windows. Mais il s'agit aussi de développer les PC et matériels utilisant un processeur ARM. Une des annonces les plus intéressantes est la création du Snapdragon Developer Kit. Il s'agit d'un PC ARM pour faciliter le travail des développeurs. Ce kit rappelle le Mac Mini Arm qu'Apple proposait aux développeurs pour faciliter la transition x86 vers Apple Silicon.

Windows 11 n'a pas été présenté à la BUILD, mais très vite, les rumeurs ont commencé à circuler. La fin de support de Windows 10 avait été annoncée pour 2025 indiquant la disponibilité d'un nouvel OS dans les prochains mois. Après l'annulation, ou le décalage, de Windows 10X, il semblerait que Windows 11 récupère des nouveautés de cet OS. L'interface devrait largement changer et avoir une approche telle qu'elle existe déjà dans macOS, Android, etc. La barre des tâches ressemble au dock de macOS. Et le menu Windows a été assez largement modifié même s'il garde son esprit. Normalement, et sur le leak que nous avons testé, l'OS ne change fondamentalement pas. C'est une évolution de Windows 10 ou du moins, une mise à jour qui se fond dans l'héritage Win10. Les widgets se réinstallent dans l'OS et on pourra utiliser un mode sombre plus efficace. Le logo Windows a été revu. On s'attend à une sérieuse cure de performances et d'optimisation. Il faut dire que Windows 10 a grossi au fur et à mesure des mises à jour. Mais Microsoft n'effacera sans doute pas toutes les ergonomies anciennes remontant parfois à XP.

Sauf surprise, Windows 11 a été annoncé lorsque ce numéro a été imprimé. Le nom de Windows 11 est Sun Valley. Pour en savoir plus : programmez.com

La WWDC 2021 s'est concentré sur les prochains OS et les logiciels. Le matériel a été absent de cette édition. Cet automne, tous les OS seront mis à jour. La grosse nouveauté pour les développeurs est un service de BUILD dans le cloud : XCode Cloud. Il permettra de faire le build, les tests et de déployer son app. XCode rattrape donc son retard sur la chaîne CI/CD. SwiftUI a été mis en avant par Apple sur comment créer des apps macOS avec SwiftUI. Pour la partie documentation, Apple intègre DocC dans Xcode 13 (et +).

La Google I/O 2021 a été l'occasion de dévoiler la bêta de Android 12. Cette version permettra une plus grande personnalisation de l'interface, une meilleure protection des données privées, la possibilité de déverrouiller sa voiture (compatible), le codec H.265 par défaut, nouvelles API Performances. L'autre grosse annonce, un peu surprise, est une fusion en WearOS et Tizen (Samsung). Il faut dire que WearOS était un peu l'OS délaissé. Objectif : créer une plateforme unifiée. On retrouvera aussi des fonctions de Fitbit. La prochaine version permettra aussi de récupérer des apps directement depuis un smartphone.

Autre nouveauté, la modélisation 3D temps réel avec le projet Starline ! Imaginez d'être un avatar 3D sur votre Meet ou Zoom ? C'est l'objectif de Starline. ChromeOS pourrait aussi supporter nativement les apps Linux !

Côté IA, Google a dévoilé LaMDA. Il s'agit d'un nouveau modèle pour comprendre et interpréter le langage naturel et pour interagir avec une personne. Il est conçu pour pouvoir traiter de n'importe quel sujet.

Monitoring : la supervision des applications simplifiée dans Nutanix Karbon

Le monitoring est un véritable enjeu lors du déploiement d'applications dans les environnements Kubernetes. Dans cet article, Nutanix présente comment déployer simplement une infrastructure de pilotage à partir de sa distribution Karbon, de Prometheus et de Grafana.

Lors du déploiement d'application dans un cluster Karbon Kubernetes, ou même un cluster Kubernetes classique, fraîchement déployé, une question se pose très rapidement : comment mettre en place un monitoring ? Karbon, la distribution Kubernetes intégrée à la plateforme Nutanix, un des leaders du cloud computing, embarque tout ce qui est nécessaire pour mettre en place du monitoring. Lors du déploiement d'un nouveau cluster avec Karbon, Prometheus Operator est installé par défaut et une instance Prometheus est également déployée et configurée pour surveiller les ressources du cluster Kubernetes. L'objectif de cet article est de donner un aperçu et un guide de configuration étape par étape pour configurer un cluster Karbon afin de monitorer à la fois le cluster et les applications à travers une interface unifiée basée sur Grafana.

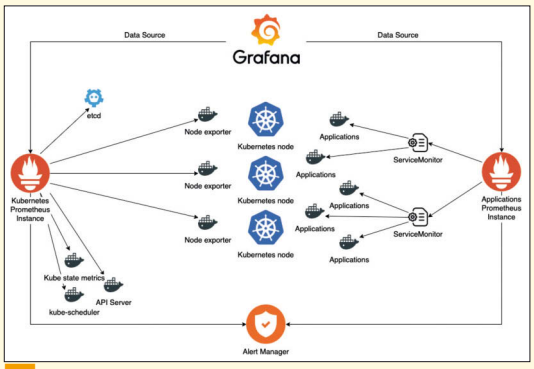
Karbon

Karbon est la solution de gestion de Kubernetes de Nutanix qui permet l'approvisionnement clé en main, les opérations et la gestion du cycle de vie de Kubernetes. Contrairement aux autres solutions Kubernetes, Karbon s'intègre de manière transparente à l'ensemble de la pile native du cloud de Nutanix, et simplifie considérablement Kubernetes sans verrouillage du fournisseur. Pour les clients de Nutanix, Karbon est inclus dans toutes les éditions du logiciel AOS.

Dans Kubernetes, les opérateurs décrivent une classe de logiciels qui aide à automatiser le fonctionnement d'un autre logiciel. Un opérateur Kubernetes est généralement conçu pour gérer tous les aspects opérationnels entourant le cycle de vie d'une application, depuis la configuration initiale et le déploiement jusqu'aux mises à jour et aux correctifs, en passant par la mise à l'échelle ou la simple utilisation. Tous ces éléments peuvent être intégrés dans les opérateurs Kubernetes et appelés selon les besoins. L'idée est qu'en utilisant les opérateurs Kubernetes, l'implication humaine dans le fonctionnement d'une application peut être réduite au strict minimum ou supprimée complètement.

Prometheus

L'opérateur Prometheus permet de faire fonctionner Prometheus au-dessus de Kubernetes de manière évolutive, automatisée et élégante, tout en préservant les options de configuration natives de Kubernetes. Prometheus est un système de surveillance open source, doté d'un modèle de données dimensionnelles, d'un langage d'interrogation souple, d'une base de données de séries chronologiques efficace et d'une approche d'alerte moderne.



1 *Schéma des composants de l'architecture de surveillance.*

Ayant rejoint la Cloud Native Computing Foundation en 2016 en tant que deuxième projet "Graduated", après Kubernetes, Prometheus est le choix de facto pour le reporting et le monitoring dans un environnement Kubernetes.

Grafana

Grafana est la solution open source d'analyse et de surveillance pour toutes les mesures et pour chaque base de données. La solution permet d'interroger, de visualiser, d'alerter et de comprendre les mesures, quel que soit l'endroit où elles sont stockées. Grafana dispose en outre d'une pléthore d'options de visualisation pour vous aider à comprendre les données.

Ces outils permettront de mettre en place du reporting sur le cluster Karbon Kubernetes. À noter que cette manipulation marche également sur un cluster Kubernetes classique avec quelques adaptations.

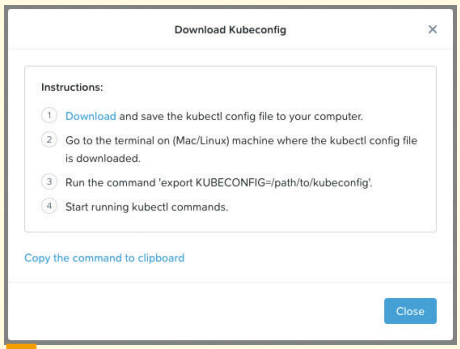
Architecture

Une première instance de Prometheus est déjà déployée avec Karbon et dédiée à la surveillance du cluster Kubernetes. Il faut en déployer une seconde spécifiquement pour la surveillance de nos applications. Une seule instance de Grafana sera également déployée pour offrir une vue unifiée des deux instances de Prometheus. 1

L'architecture repose donc sur deux instances Prometheus qui surveillent différentes ressources :

- Un serveur Prometheus (déployé par défaut par l'installateur de Karbon) va surveiller les composants internes et l'état de Kubernetes.
- Un autre serveur Prometheus surveillera toutes les autres applications déployées dans le cluster.

Il est évidemment possible d'augmenter le nombre d'instances indépendamment dans chacun de ces groupes afin d'apporter de la résilience si nécessaire.



2 *Karbon UI pour télécharger le fichier Kubeconfig*

Les ressources de Kubernetes

Par défaut, l'opérateur Prometheus crée des définitions de ressources personnalisées dans le cluster Kubernetes.Dans cet article, ce sont les ressources customs suivantes qui seront utilisées :

- **Prometheus,** qui définit un déploiement de Prometheus souhaité. L'opérateur s'assure à tout moment qu'un déploiement correspondant à la définition de ressource est en cours.
- **ServiceMonitor,** qui spécifie de manière déclarative comment les groupes de services doivent être surveillés. L'Opérateur génère automatiquement la configuration Prometheus en fonction de la définition.

Guide de configuration

Afin de pouvoir effectuer les étapes suivantes, il est nécessaire d'avoir installé le fichier kubeconfig du cluster à configurer. 2

Vous avez également besoin d'un service LoadBalancer fonctionnel, qui sera MetalLB dans cet exemple.

Instance Kubernetes Prometheus

En premier lieu, il faut reconfigurer l'instance existante de Prometheus afin de restreindre les ServiceMonitors qu'elle utilisera.

Pour cela, nous allons utiliser les étiquettes Kubernetes et nous allons dire à Prometheus de n'utiliser que les ServiceMonitors qui se trouvent dans un espace de noms avec l'étiquette

monitoring=k8s. 3

3

```
#!/bin/bash

# Set label on the system namespace
kubectl label ns/kube-system monitoring=k8s
kubectl label ns/ntnx-system monitoring=k8s
# Patch existing prometheus resource to limit ServiceMonitors used
kubectl -n ntnx-system patch --type merge prometheus/k8s -p '{"spec":{"serviceMonitorNamespaceSel
```

karbon-app-mon-step1.sh hosted with ❤ by GitHub view raw

La prochaine étape consiste à déployer la nouvelle instance Prometheus qui surveillera les paramètres de l'application.
Pour ce faire, il faut d'abord créer un namespace, un compte de service et quelques règles RBAC spécifiques pour permettre à l'instance Prometheus de collecter des mesures sur les applications. 4
Les règles RBAC ci-dessus devront être adaptées en fonction des besoins de sécurité de votre cluster.
Appliquons-les =>

```
kubectl apply -f karbon-app-mon-step2-rbac.yml
```

Ensuite, nous allons créer une nouvelle ressource pour permettre à l'opérateur Prometheus de déployer une nouvelle instance.
Il faut l'appliquer =>

```
kubectl apply -f karbon-app-mon-step2-prom.yml
```

La définition de la ressource sera la suivante :

- Utilisation de tous les ServiceMonitors avec un label monitoring=apps.
- Réutilisation du gestionnaire d'alertes existant.
- Stockage des données pendant 15 jours
- Création d'un volume persistant de 40 Go sur la classe de stockage par défaut

La dernière étape consiste à créer une ressource ServiceMonitor qui configurera automatiquement Prometheus pour collecter les mesures de tous les services avec un label spécifique 5
Il faut l'appliquer =>

```
kubectl apply -f karbon-app-mon-step2-service-monitor.yml
```

Dans ce cas, il faudra utiliser l'étiquette monitoring=apps pour spécifier qu'un service doit être surveillé, et les mesures seront exposées sur le port appelé http-metrics.

Tableau de bord Grafana

Afin de pouvoir construire un tableau de bord visuel, Grafana doit être déployé en utilisant la charte officielle de Helm à partir du CLI Helm v3 installé sur la machine. 6
Il faut ensuite passer par la création un fichier de réponse pour Grafana et l'utiliser avec Helm pour déployer Grafana. 7
Dans la sortie du script, se trouve l'URL de Grafana et le login/mot de passe requis pour s'y connecter.

Déploiement d'un exemple d'application

Maintenant que le système fonctionne, il est temps de déployer un exemple d'application et de montrer comment il est possible d'intégrer automatiquement au nouveau système de surveillance des applications. 8
Il s'agit d'une application de base qui expose les mesures prêtes pour Prometheus sur le port 8080.

```
kubectl apply -f karbon-app-mon-step4-app.yml
```

4

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: monitoring-apps
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus
  namespace: monitoring-apps
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: prometheus-apps
rules:
  - apiGroups: [""]
    resources:
      - nodes
      - services
      - endpoints
      - pods
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources:
      - configmaps
    verbs: ["get"]
  - nonResourceURLs: ["/metrics"]
    verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: prometheus-apps
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus-apps
subjects:
  - kind: ServiceAccount
    name: prometheus
    namespace: monitoring-apps
```

karbon-app-mon-step2-rbac.yml hosted with ♥ by GitHub view raw

5

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: monitoring-apps
  labels:
    monitoring: apps
spec:
  selector:
    matchLabels:
      monitoring: apps
  endpoints:
    - port: http-metrics
```

karbon-app-mon-step2-service-monitor.yml hosted with ♥ by GitHub

6

```yaml
## Expose the grafana service to be accessed from outside the cluster (LoadBalancer service).
## or access it from within the cluster (ClusterIP service). Set the service type and the port t
## ref: http://kubernetes.io/docs/user-guide/services/
##
service:
  type: LoadBalancer

## Enable persistence using Persistent Volume Claims
## ref: http://kubernetes.io/docs/user-guide/persistent-volumes/
##
persistence:
  enabled: true
  size: 10Gi

## Pass the plugins you want installed as a list.
##
plugins:
  - grafana-piechart-panel
  # - digrich-bubblechart-panel
  # - grafana-clock-panel

## Configure grafana datasources
## ref: http://docs.grafana.org/administration/provisioning/#datasources
##
datasources:
  datasources.yaml:
    apiVersion: 1
    datasources:
      - name: Prometheus K8s
        type: prometheus
        url: http://prometheus-k8s.ntnx-system.svc.cluster.local:9090
        access: proxy
        isDefault: true
      - name: Prometheus Apps
        type: prometheus
        url: http://prometheus-apps.monitoring-apps.svc.cluster.local:9090
        access: proxy
```

karbon-app-mon-step3-grafana-values.yml hosted with ♥ by GitHub view raw

7

```
#!/bin/bash

# prerequisite
# helm v3 with stable repo
helm repo add stable https://kubernetes-charts.storage.googleapis.com
helm repo update


# create dedicated namespace
kubectl create ns grafana

# install Grafana helm chart
helm install grafana stable/grafana --namespace grafana -f karbon-app-mon-step3-grafana-values.y

kubectl -n grafana rollout status deploy/grafana
export SERVICE_IP=$(kubectl get svc --namespace grafana grafana -o jsonpath='{.status.loadBalanc
export GF_PASSWORD=$(kubectl get secret --namespace grafana grafana -o jsonpath="{.data.admin-pa
echo "you can connect on Grafana http://$SERVICE_IP with admin/$GF_PASSWORD"
```

karbon-app-mon-step3-grafana.sh hosted with ♥ by GitHub view raw

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: example-app
  template:
    metadata:
      labels:
        app: example-app
    spec:
      containers:
      - name: example-app
        image: fabxc/instrumented_app
        ports:
        - name: http-metrics
          containerPort: 8080
---
kind: Service
apiVersion: v1
metadata:
  name: example-app
  labels:
    app: example-app
    monitoring: apps
spec:
  selector:
    app: example-app
  ports:
  - name: http-metrics
    port: 8080
```

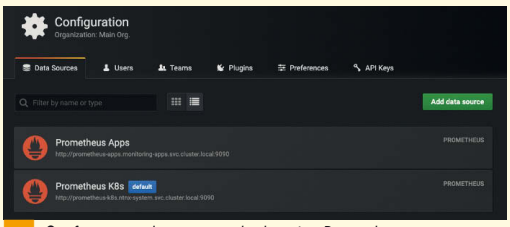
karbon-app-mon-step4-app.yml hosted with ♥ by GitHub

8 *Voici l'application en exemple.*

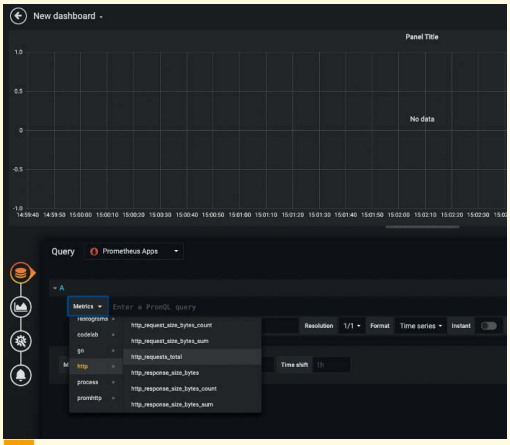
Il faut ensuite ouvrir l'interface de Grafana et vous trouverez deux sources de données déjà configurées :

- Prometheus K8s : l'instance intégrée de Prometheus avec le système Metrics
- Prometheus Apps : l'instance Prometheus fraîchement déployée avec les paramètres des applications 9 et 10

Dans cet article, vous avez pu constater à quel point



9 *Configuration des sources de données Prometheus*



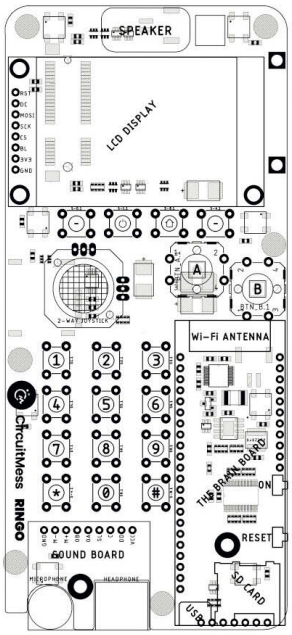
10 *Lors de la création d'un nouveau tableau de bord, il suffit de sélectionner les sources de données des applications Prometheus pour naviguer dans la liste déroulante des mesures des applications.*

il est simple, grâce à l'opérateur intégré Prometheus, de contrôler les paramètres de vos propres applications. Il est maintenant temps de libérer la puissance de Grafana et de Prometheus en construisant votre propre tableau de bord complexe.

François Tonic

Ringo : le téléphone à souder soi-même

Circuitmess conçoit et vend des kits à souder et à monter : assistant vocal, console rétro, mini table de mixage, montre… et un téléphone. Nous allons justement parler de ce kit plutôt original : Ringo. Le kit complet permet de souder et de monter un véritable téléphone. Nous sommes plus proches d'un feature phone que d'un smartphone. L'expérience est originale !



D'aspect, le Ringo est comme un gros Nokia des années 1990 – 2000. Il intègre un clavier numérique, une prise jack, un port micro-USB, des boutons d'action, un joystick, un écran LCD, haut-parleur, des touches de fonctions, un port micro-SD et un port SIM ! Eh oui, le Ringo est un téléphone. Circuitmess utilise un ESP32.

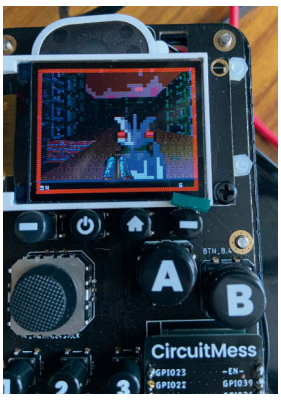
Une électronique efficace

La première étape est de monter le téléphone. Il faut souder les touches, l'écran, quelques ports ici et là et connecter l'ensemble avant d'ajuster les deux socles. Il faut environ 2h à 2h30 pour finaliser le montage. L'audio et le réseau (wifi+téléphone) sont gérés par deux cartes indépendantes

Nos conseils :

- Ajuster bien les touches pour éviter tout blocage avec la façade
- Faites des soudures propres sinon vous allez les vérifier une par une en cas de problème
- Tester le Ringo avant de fixer les deux parties

Ringo fonctionne avec CircuitOS. C'est lui qui permet de configurer et d'utiliser les fonctionnalités par défaut. Ringo est compatible 4G. On peut coder des apps en C, C++, Python, CircuitBlocks.



Qui dit ESP32 dit compatibilité Arduino IDE. Pour rajouter le Ringo dans les cartes de l'outil, il suffit d'ajouter l'URL dans Additional board Manager (panneau préférences) : https://raw.githubusercontent.com/CircuitMess/MAKERphone/boardArduino/package_CircuitMess_Ringo_index.json

On valide. Dans le menu Tools, on affiche la fenêtre boards manager (item board) puis on tape Ringo dans la recherche. Et on installe. On vérifie s'il y a une mise à jour. Et c'est tout ! Pour utiliser Ringo dans l'IDE, il suffit de sélectionner dans Tools -> Board -> Ringo by CircuitMess. Et de choisir le bon port COM. Comme toujours, sous Windows, le gestionnaire de périphérique est notre ami. Pour compléter le support, vous pouvez ajouter la librairie Ringo, depuis le Library Manager.

Attention : vous pouvez à tout moment revenir au firmware usine (très simple depuis Arduino IDE).

CircuitBlocks et la partie OS

Le constructeur propose aussi son propre outil de développement par blocs : CircuitBlocks. Il s'appuie sur MakeCode (Microsoft) et PXT-Blocky (fork de Blocky de Google). Il fonctionne comme Scracht. On compose son projet avec des blocs fonctionnels et le code se construit en temps réel. L'outil est open source. L'intérêt de CircuitBlocks est qu'il génère un code C compatible Arduino. On peut donc reprendre le code généré et le coller sur Arduino IDE.

Attention : si vous utilisez CircuitBlocks et Arduino IDE en même temps, Arduino IDE générera des erreurs pour téléverser les codes et en cas de flashage du firmware. Quittez CircuitBlocks pour utiliser Arduino IDE. L'outil de CircuitMess s'appuie sur des éléments Arduino pour fonctionner…

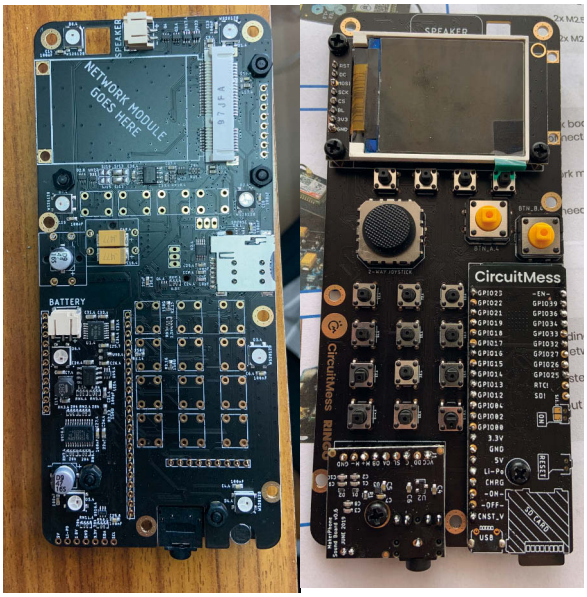
L'OS est appelé CircuitOS. Il se base sur FreeRTOS. L'OS contient toutes les fonctionnalités du téléphone et la configuration. Il s'agit en réalité du firmware. Il est possible de l'adapter, et de le modifier. Pour ce faire, il faut utiliser un IDE de type Visual Studio Code. Notons que la librairie Ringo utilise platformio, un socle technique utilisé par des dizaines de devices IoT / DIY. Il faudra installer les extensions platformio sur VS Code.

Il suffit d'ajouter l'extension platformio IDE à VS Code. Lors de l'installation, les dépendances sont installées. Après un redémarrage de l'IDE, l'icône platformio apparaît à gauche. Quand vous créez un nouveau projet, il faut sélectionner Wemos Lolin32 dans Board. Ce MCU est celui du Ringo. Le template va générer le projet et l'ensemble des composants nécessaires.

Ensuite il faudra récupérer depuis le github CircuitMess : CircuitMess Ringo et firmware. Ajoutez les deux dossiers à l'explorateur platformio. Il faut charger le dossier dans platformio puis de faire le build et le upload.

Notre avis

Voilà un projet DIY comme nous les aimons : l'idée est sympa, le kit bien pensé et de qualité. Oui, le tarif est un peu élevé, mais nous sommes dans des kits haut de gamme. Pour le développeur-maker, la documentation technique nécessiterait d'être complétée et plus d'exemples pour prendre en main les fonctionnalités. La partie développement est le point



faible de la documentation. Il n'existe pas de guide de références de la librairie.

En fouillant un peu dans les sources, on comprend que l'écran utilise le protocole SPI, que les LED (en réalité un NeoPixel) sont gérées avec la librairie FastLED. Regardez les MAKERphone.h et .cpp pour comprendre comment fonctionne Ringo et les possibilités de hack. La communauté est assez active et on y trouva des astuces, conseils et apps.

Pour nous, Ringo est une plateforme plus logicielle que matérielle car les GPIO ne sont pas accessibles si la coque est installée. C'est le gros défaut du kit. Cependant, rien ne vous empêche de changer l'écran, de rajouter des capteurs, de développer des usages avec des capteurs. Pour mapper correctement les touches et les modules, reportez-vous aux déclarations dans MAKERphone.h.

Site officiel : https://circuitmess.com
Pour aller plus loin :
https://github.com/CircuitMess/CircuitMess-Ringo

RINGO, C'EST AUSSI POUR LES JEUX !

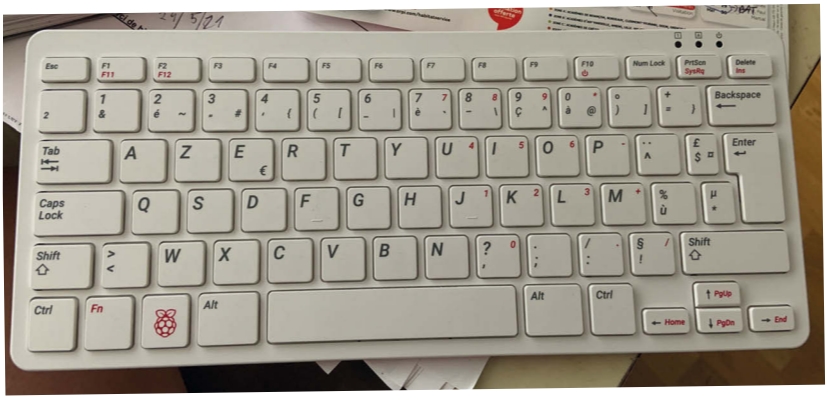
L'installation d'une nouvelle app est très simple : on connecte la carte SD, on récupère le .bin de l'app et on le copie dans la SD ! C'est tout. Ensuite, sur Ringo, on scrolle les icônes d'accueil et on sélectionne l'app souhaitée. Plusieurs jeux sont disponibles.

L'un des plus remarquables est Anarch. Il s'agit d'un jeu à la Doom et Wolf3D ! Le résultat est impressionnant pour les ressources réduites de l'ESP32 et la taille de l'écran.

Anarch exige seulement 200 Ko, 32 Ko de RAM et un processeur à 40 MHz. L'ESP32 du Ringo suffit largement ! Pour en savoir plus : https://drummyfish.gitlab.io/anarch/

Petite vidéo : https://youtu.be/y5m0TIQY73g

Raspberry Pi 400 : retour sur 6 mois d'utilisation



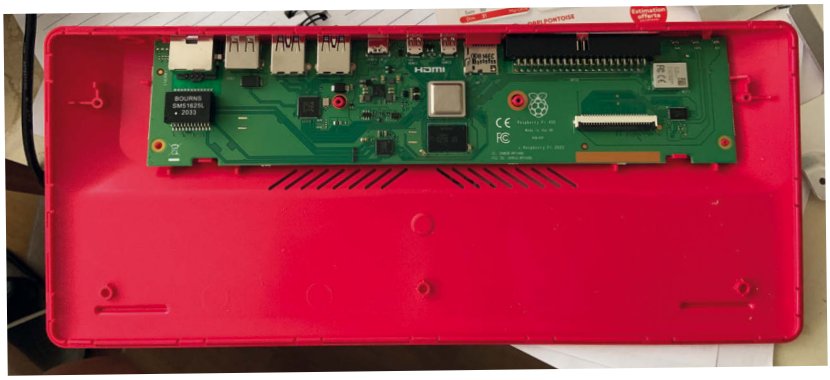
Automne 2020, un ordinateur complet sortait de la fondation Raspberry Pi : le Pi 400. On avait l'impression de retrouver l'esprit des anciens ordinateurs avec toute l'électronique sous le clavier. Nous avons utilisé le Pi 400 à la rédaction durant plusieurs mois. L'idée est bonne, mais le produit final déçoit sur plusieurs points.

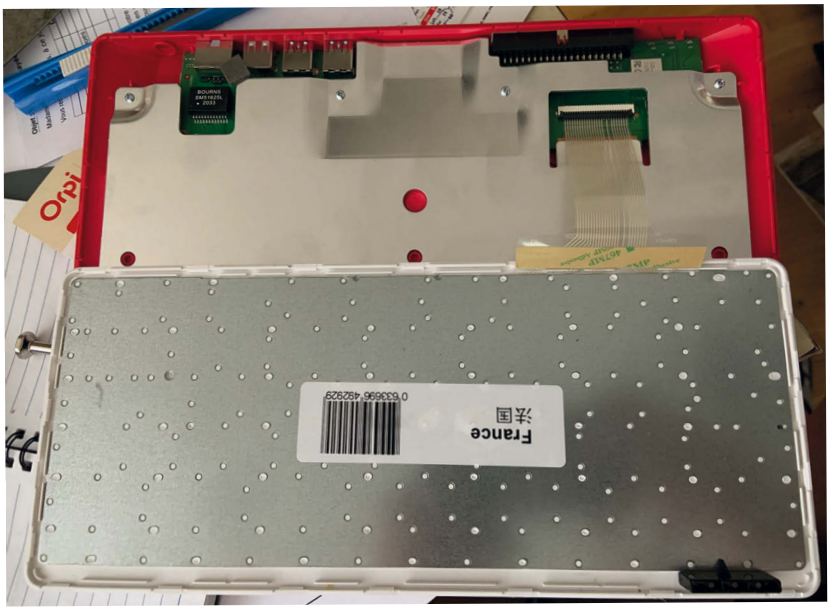
Robe blanche, pas plus grand qu'un clavier sans pavé numérique, le Pi 400 est plutôt élégant. Seules trois LED donnent des informations de fonctionnement, l'ensemble de la connectique est à l'arrière. On connecte un écran, une souris, et nous disposons d'un ordinateur complet !

Connectique et spécifications

Côté connectique, nous disposons de l'USB 2 et 3, USB-C pour l'alimentation, Ehternet 1 Gb/s, port microSD, 2 micro-HDMI, le Wifi + Bluetooth, une encoche sécurité et les GPIO de la Pi. Bref, on retrouve les ports de la Pi 4. Trois points négatifs sur cette connectique : les micro-HDMI qui obligent à utiliser un adaptateur, une espace entre les deux ports HDMI trop juste, la position des GPIO. Pour nous, les GPIO sont trop à l'intérieur du boitier et cela gêne l'insertion des HAT. Il faut utiliser une nappe entre les GPIO et la carte d'extension. Ce point sera à revoir impérativement dans une future version. Pour le reste, c'est suffisant pour un usage courant et le monde maker / DIY.

La Pi 400 intègre un processeur ARM 4 cœurs à 1,8 GHz, 4 Go de RAM. Comme pour la Pi classique, nous n'avons aucun stockage interne ni extension possible interne hormis la SD. Dommage. L'alimentation, si on prend le kit complet, est classique : 5V. La fondation





aurait pu faire un effort sur la grosseur de l'adaptateur ! Le clavier est complet, sans pavé numérique. Concernant la frappe et la qualité des touches, nous sommes mitigés. On sent la qualité de fabrication, mais la frappe n'est pas assez franche. Les touches sont un peu molles et cela peut nuire en usage intensif. La carte SD est prête à l'emploi. Il est très facile de flasher une carte avec les OS compatibles.

Ouverture et réparabilité : proche de 0 !

Nous espérions beaucoup sur la facilité d'ouverture et les possibilités de changer les cartes internes. Nous avons immédiatement déchanté. La fondation a raté une occasion de se démarquer. Tout d'abord, l'ouverture est tout sauf facile. Il faut forcer les deux parties du boîtier pour ouvrir la bête. Le clavier est protégé par une feuille de protection. Il est relié par un fin ruban à la carte mère. L'un des défauts des Pi est le dégagement de chaleur, nous l'avons constaté avec la Pi 4. Malheureusement, la Pi 400 n'améliore pas la situation. Et pour pallier l'absence de ventilateur ou de dissipateur sur le SoC, une épaisse plaque de dissipation cache l'électronique !

La carte du Pi 400 occupe à peine la moitié du boîtier. Et il s'agit d'une carte spécifique. La Pi classique a été entièrement redesignée pour placer tous les ports à l'arrière, ajouter le connecteur spécifique du clavier. On peut espérer de sérieuses évolutions dans une v2.

Actuellement, il n'est pas possible d'acheter les différents modules de la Pi 400 (carte, boîtier vide, clavier, etc.) en cas de panne. Et les accessoires dédiés n'existent toujours pas. Et il manque plusieurs connecteurs pratiques : port caméra (CSI) et pour l'écran Pi (DSI), on notera aussi l'absence d'un connecteur audio. C'est une autre lacune de la Pi 400.

En usage, nous retrouvons la Pi classique, donc aucune surprise à avoir. Mais c'est appréciable d'avoir un ordinateur complet et on évite d'avoir des câbles partout. On apprécie la qualité de fabrication et les couleurs choisies.

On aime les touches spéciales, dont l'option marche / arrêt :
- Fn + F10 : arrêt et démarrage
- CTRL + Fn + F10 : on force l'arrêt complet

La Pi 400 est disponible en deux kits :

Kit standard	Kit complet
Pi 400 seul	Pi 400 Adaptateur alimentation Carte SD Câble HDMI Souris Livre de découverte
74,5 €	106,5 €

Sauf à disposer de tous les accessoires du kit complet, la Pi 400 seule est chère ou alors est-ce le kit complet. Il aurait été appréciable de le proposer à 99 € et d'inclure une nappe pour les GPIO.

Nos idées pour une Pi 400 v2

Plusieurs points pourraient être améliorés : port audio, meilleure ergonomie des GPIO, ouverture plus facile, possibilité d'installer un stockage interne, peut être des ports USB sur le côté et pas uniquement à l'arrière, pourquoi pas un clavier rétroéclairé ou un vrai pavé numérique, un bloc d'alimentation plus petit, plus de coloris. Sans oublier : la possibilité d'avoir les modules et non la Pi 400 entière en cas de panne.


BIGDATA & AI by Xcorp PARIS
Conférence et Exposition
10e Edition • 28 & 29 septembre 2021
Palais des Congrès • PARIS
DevOps
BIG
Reinforcement learning
TIME TO ACCELERATE
Ethique
Robotique
Data for Good
DATA
15 000 PARTICIPANTS
200 PARTENAIRES
350 CONFÉRENCES & ATELIERS
Cybersecurité
Gouvernance et qualité
Cloud
IA Green
Machine learning
&
AI
Automatisation
NLP / NLG
Inscrivez-vous sur www.bigdataparis.com

SwiftUI a-t-il le potentiel pour détrôner UIkit ?

Hugo Extrat,
Ingénieur Concepteur Développeur

Yoann Lathuilière,
Ingénieur Concepteur Développeur

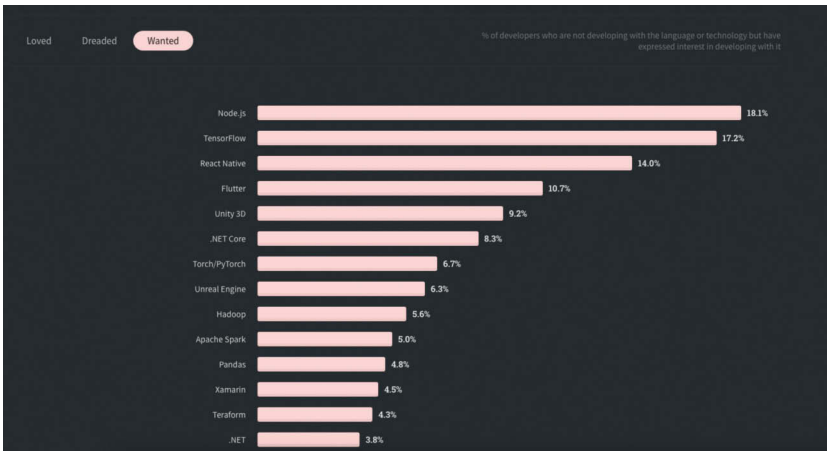
SQLI DIGITAL EXPERIENCE

SwiftUI est le nouveau framework de développement mobile, conçu par Apple. Il est apparu pour la première fois en 2019. Les technologies permettant de créer des applications multi-plates-formes ont le vent en poupe depuis 2015, avec l'arrivée de React Native, suivi de Flutter en 2017. Ces deux technologies novatrices partagent certains points communs, dont la construction d'interfaces utilisateur avec une syntaxe au style dit "déclaratif".

Contrairement au code généré par un story-board très difficile à interpréter à la lecture, un code au style déclaratif se caractérise par une lecture aisée, permettant de se figurer sans peine à quoi ressemblera l'interface. Alors que vaut SwiftUI ? Va-t-il détrôner UIkit, le framework utilisé depuis plus de 10 ans sur iOS, trousseau pour créer des interfaces utilisateurs, animer les éléments et gérer les interactions ?

La popularité de Flutter et React Native ne cesse de grandir : d'après l'étude Stackoverflow Developer Survey 2020, React Native et Flutter se hissent aux 3e et 4e places des technologies avec lesquelles les développeurs aimeraient travailler.

Figure 1



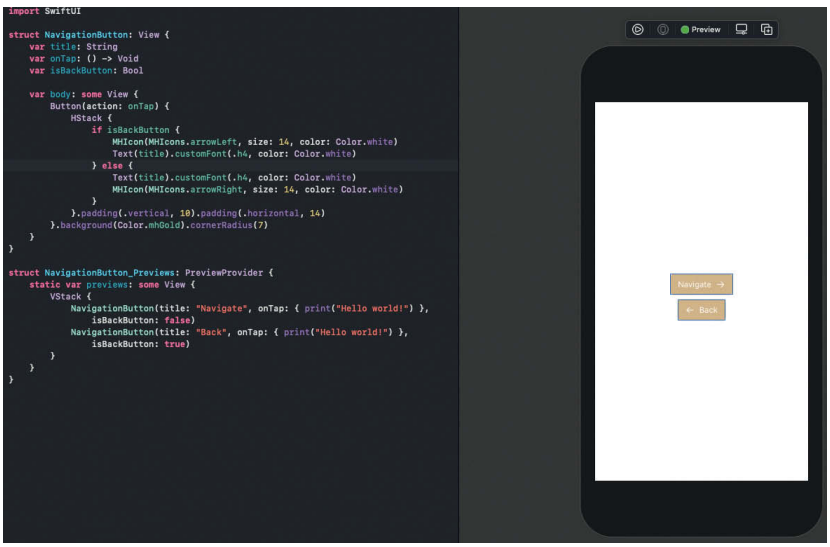


Figure 2 : *Exemple de Preview avec deux états*

C'est certainement la raison qui a poussé Apple à sortir SwiftUI, un nouveau framework permettant lui aussi de construire des interfaces utilisateurs de façon déclarative. Preuve de l'importance de ces technologies, Android tente à son tour de se lancer dans l'aventure avec Jetpack Compose qui est en cours de développement et actuellement en bêta. **Figure 1**

SwiftUI : tour d'horizon et ses avantages

Depuis le lancement du langage Swift, SwiftUI représente probablement le plus gros changement dans l'écosystème des développeurs Apple. Cette nouvelle façon de développer des interfaces apporte un gain de temps assez conséquent par rapport à UIkit, grâce à une écriture plus naturelle et une prise en main simplifiée. Cette syntaxe déclarative offre également un code plus compréhensible, avec la possibilité de créer facilement des composants réutilisables en comparaison à la programmation impérative avec les Storyboards et XIB.

Avant de se lancer avec SwiftUI, un détail reste à ne pas négliger… Prenez bien en compte que ce langage est uniquement disponible à partir d'iOS 13 ! En 2020 lors de la WWDC, Apple a annoncé la sortie de SwiftUI 2.0 et son lot de nouveautés (composants, corrections de bugs…). Toutefois, celui-ci n'est disponible… qu'à partir d'iOS 14. Avant de démarrer un projet avec SwiftUI, assurez-vous impérativement de la version d'iOS concernée : si vous souhaitez cibler des versions plus anciennes qu'iOS 13 ou 14, vous devrez malheureusement vous passer de SwiftUI. Enfin, il existe d'autres prérequis techniques pour commencer une application SwiftUI : Xcode 11 et macOS Catalina (ou supérieur).

SwiftUI propose un système de preview permettant aux développeurs de prévisualiser leurs vues directement dans Xcode. À chaque modification d'un morceau de code, l'aperçu se met à jour et affiche le rendu en conséquence. Une fonctionnalité bienvenue, notamment lors du développement d'une vue possédant différents états : grâce à la preview, il devient désormais possible de visualiser tous les états en un seul coup d'œil. Vous pourrez aussi vous représenter votre interface sur différents devices, ou encore visualiser votre vue avec le dark mode activé, et tout ceci sans même avoir besoin de lancer un simulateur ! **Figure 2**

SwiftUI permet donc de réaliser des interfaces de manière déclarative, et ce genre d'interface s'associe généralement à une bibliothèque de programmation réactive. Grâce à cette bibliothèque, l'interface se rafraîchit automatiquement lorsqu'une donnée change, par exemple. Apple a donc décidé de fournir cette bibliothèque, nommée Combine, assez proche de RxSwift et utilisée par de nombreux développeurs.
Ci-contre un schéma simplifié montrant le flux de données pour une application SwiftUI / Combine : **figure 3**
Lorsqu'on construit une vue SwiftUI, il est possible d'indiquer des dépendances de données pour cette vue. Lorsque les données changent (à cause d'un événement externe ou d'une action utilisateur), SwiftUI va automatiquement mettre à jour les parties de l'interface concernées par les changements.

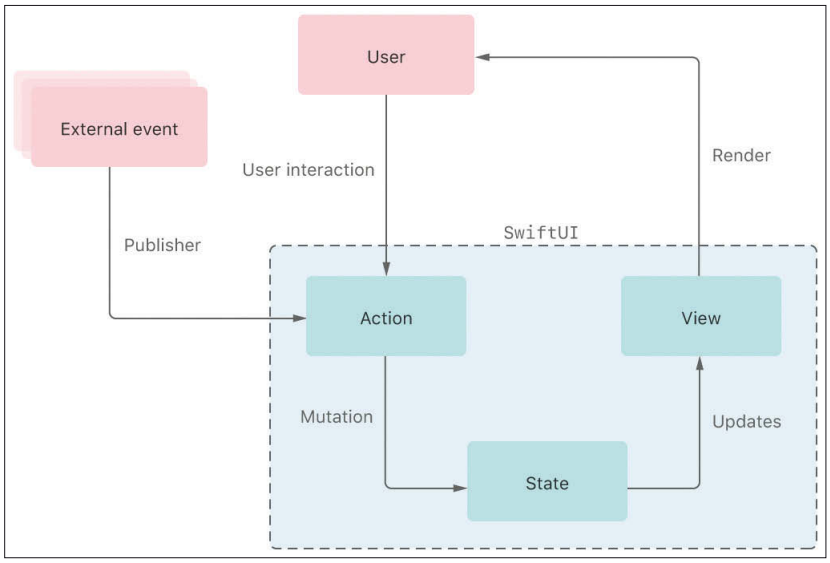


Figure 3

SwiftUI offre la possibilité aux développeurs de construire des applications pour tous les systèmes d'exploitation Apple (iOS / macOS / watchOS / tvOS) à partir d'un seul et même framework. Obtenir une interface utilisateur qui se partage facilement entre ces plateformes facilite grandement la tâche des développeurs, pour ne pas dire qu'elle apporte même une certaine tranquillité d'esprit ! Depuis 2020, avec le lancement d'iOS 14, SwiftUI rend également possible le développement de widgets pour vos applications.

Vous pouvez désormais créer des applications complètes en Swift uniquement, sans avoir besoin de créer des interfaces XML, et ainsi éviter tous les problèmes liés à ce type de fichier… surtout quand deux développeurs travaillent sur la même vue, causant des conflits et par extension, des prises de tête pour les résoudre ! L'un des bénéfices les plus importants de SwiftUI reste sûrement le gain en productivité, améliorée aussi bien par l'interface déclarative, la programmation réactive, Swift, ou encore la preview. **Figure 4**
Enfin, nous avons trouvé l'ajout d'animation relativement aisé. Quelques lignes de code suffisent à créer des animations, comme par exemple avec le modificateur « animation », qui donne automatiquement vie aux propriétés animables d'une vue (couleur, opacité, rotation, taille…).
Figure 5
Avant de se lancer avec SwiftUI sur votre futur projet, il faut bien réfléchir aux différentes problématiques que vous pouvez rencontrer ; certaines API ne sont pas disponibles en SwiftUI et vous allez avoir besoin d'UIkit. L'adoption est encore assez faible, ce qui implique une documentation assez restreinte ou encore se retrouver face à un bug SwiftUI très peu voire non remonté par d'autres développeurs. La montée en compétence sur SwiftUI sera plus aisée si vous avez déjà une connaissance de UIkit et de la programmation réactive.
Si vous avez un petit projet, une application personnelle ou un prototype à faire, vous pouvez commencer avec SwiftUI pour prendre la main sur le framework, vous faire votre avis sur le framework, et enfin si vous en avez l'occasion et que vous pensez que SwiftUI correspond à votre projet, l'utiliser pour des applications à plus grande envergure.
Il est aussi possible de commencer à intégrer SwiftUI sur un projet UIkit existant grâce à l'interopérabilité de ces deux technologies. On pourra par exemple commencer avec une

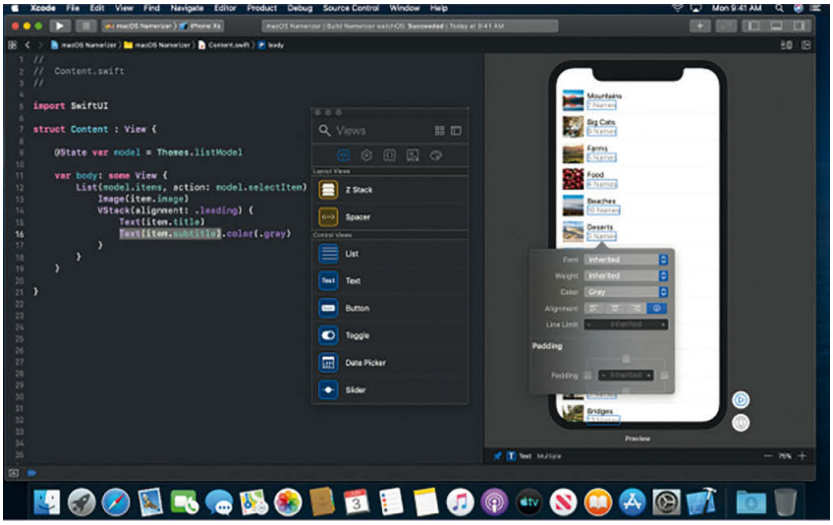


Figure 4 : *Exemple d'interface créé avec SwiftUI*

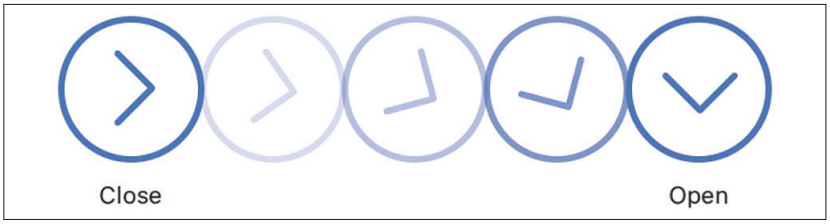


Figure 5 : *Exemple d'animation pour un effet de rotation*

partie d'un écran, ou même un écran en entier, pour ensuite poursuivre le développement de l'application avec SwiftUI si cela convient.

L'accompagnement de Xcode pour initialiser un nouveau projet SwiftUI est assez similaire à UIkit et va vous permettre de lancer votre nouveau projet en quelque seconde seulement avec une vue assez basique pour commencer.

Voici comment afficher un « Hello, World! » sur votre application SwiftUI en quelque ligne seulement. Une simplicité

```swift
import SwiftUI

struct ContentView: View {
    var body: some View {
        VStack {
            Text("Hello, World!")
        }
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

Figure 6

déconcertante par rapport à Uikit, qui aurait nécessité la création d'un story-board ou un XIB (définir les contraintes) pour placer un texte au sein de votre écran et par la suite utiliser ce texte dans un ViewController **Figure 6**

Toute découverte cache son lot de surprises !

Certaines fonctionnalités sont disponibles pour iOS 13, et non pour iOS 14, et inversement, ce qui s'avère contraignant et nécessite la mise en place de hacks pour bien supporter ces deux versions. Nous avons aussi remarqué qu'il existe parfois des différences de comportement de SwiftUI vis-à-vis de certaines versions mineures d'OS, par exemple entre iOS 13.2 et iOS 13.3. C'est un point assez frustrant quand on souhaite cibler les devices à partir d'iOS 13. Si vous voulez mener un projet utilisant la pleine puissance de SwiftUI, il devient alors préférable de cibler seulement la dernière version d'iOS, actuellement iOS 14.

Si vous avez démarré un projet en SwiftUI et que vous voulez intégrer un élément UI qui n'existe pas encore, ou qui n'est pas adapté à ce que vous souhaitez, il reste possible d'utiliser des vues UIkit avec SwiftUI, et inversement. Cet interfaçage peut aussi vous permettre de démarrer la transition d'un projet qui utilise UIkit vers SwiftUI, en douceur.

Des limitations frustrantes

SwiftUI est un framework récent, il est logiquement plus difficile de trouver des ressources comme des formations, des outils ou encore de l'aide sur des forums, en comparaison à des frameworks plus matures. SwiftUI est présenté par Apple comme un framework très simple, et nous ne pouvons que l'affirmer : la montée en compétences est très rapide. Cependant, la principale difficulté réside dans l'utilisation du framework Combine, pour la gestion de l'état de l'application. Ce framework est plus complexe à appréhender, surtout pour un développeur n'ayant jamais pratiqué la programmation réactive. S'il est rapide de s'initier à SwiftUI, il faut beaucoup plus de temps pour maîtriser le framework Combine, et devenir capable de concevoir des applications de qualité.

Les composants SwiftUI sont parfois limités au niveau de la personnalisation, il est par exemple impossible de modifier les séparateurs de listes ou la largeur des UISplitViewController. Cela nécessite dans certains cas d'associer du code UIkit avec du code SwiftUI pour parvenir à certaines personnalisations.

Au cours de nos tests, nous avons remarqué des comportements inégaux de certains composants entres les versions d'iOS. Les séparateurs de listes, par exemple, ne s'affichent pas à l'identique entre iOS 13 et iOS 14. Garder une uniformité entre les différentes versions d'iOS exige alors plus d'efforts au niveau du développement pour trouver des solutions de contournement, et au niveau des tests pour vérifier le comportement des composants sur chaque version d'iOS.

Certains bugs avec Xcode se révèlent très frustrants, il n'est pas rare de rencontrer des problèmes avec les previews qui ne peuvent plus s'afficher, ou encore avec l'autocomplétion qui cesse de fonctionner. L'utilisation de SwiftUI nécessite d'utiliser iOS 13 ou plus, ce qui limite fortement son adoption, certains projets ayant besoin de supporter des versions plus anciennes d'iOS.

Les limitations liées à SwiftUI laissent penser qu'il faudra encore plusieurs années pour voir ce framework s'imposer au niveau du développement iOS. Ce n'est pas sans rappeler le langage Swift, qui a mis plusieurs années et plusieurs versions avant d'atteindre sa popularité actuelle.

SwiftUI est donc un framework très prometteur, mais qui a encore besoin de temps avant d'atteindre son plein potentiel. Les limitations de SwiftUI poussent encore de nombreux projets à démarrer avec UIkit, et même pour des projets SwiftUI, l'association de UIkit et SwiftUI reste parfois nécessaire. UIkit n'est donc pas près de disparaître et a encore de beaux jours devant lui !

Un marché du jeu vidéo prometteur pour tous

Franck Dubois
Video Game Codeur
Développeur agile
Formateur javascript / node.js / Unity / GDevelop

La crise sanitaire a bouleversé nos usages. Les confinements successifs, la fermeture des magasins, la crainte du virus ont modifié nos comportements avec, entre autres, un gros besoin de se divertir, et d'occuper le temps. La conséquence a été un « boom » de la consommation en ligne : les achats en général, la vidéo à la demande et surtout pour ce qui nous intéresse ici : le jeu vidéo.

L'usage du jeu vidéo a donc considérablement augmenté depuis fin 2019 et continue encore aujourd'hui avec environ 2,5 milliards de joueurs dans le monde et près de 36,46 millions de joueurs rien qu'en France, tous types confondus. C'est ce que révèlent deux études relatives à l'utilisation du jeu vidéo : l'une centrée sur le jeu mobile, réalisée à l'international par Facebook en juillet 2020, qui a sondé 13 246 personnes de 9 pays (États-Unis, Royaume-Uni, Allemagne, Corée du sud, France, Canada, Japon, Vietnam et Brésil) et l'autre plus spécifique au marché français, mais aussi plus large en termes de médias, menée par le SELL (syndicat des éditeurs de logiciels de loisirs) sur l'année 2020.

On y distingue 2 grandes classes de joueurs : avec d'un côté les joueurs qui jouent quotidiennement et pour qui l'activité est une passion. De l'autre, des joueurs occasionnels qui jouent entre amis ou pour passer le temps : des joueurs de « salle d'attente » qui jouent dans les transports en commun ou à toute occasion durant laquelle ils n'ont rien d'autre à faire. Ces derniers jouent donc quotidiennement, mais n'ont ni l'envie, ni le temps d'apprendre des mécanismes complexes de gameplay, ni le temps de s'investir dans une histoire. Pas de jeux au long cours ni d'attachement : ils zappent. Il leur faut donc un gameplay simple et intuitif.

Ces données ont un réel impact sur la manière dont on crée un jeu vidéo. Impact d'autant plus important si vous souhaitez vous engager dans la voie du développement indépendant. Que vous soyez jeune padawan ou maître jedi dans la création de jeux vidéo, et à partir de ces études, on vous donne quelques pistes de réflexion pour avancer à petits pas.

L'émergence de nouveaux joueurs

Tous pays confondus, l'étude montre une considérable augmentation du nombre de joueurs mobiles :

- +28% aux Etats Unis ;
- +50% au Royaume-Uni ;
- +34% en Corée du Sud ;
- +25% en Allemagne.

Nouveaux et joueurs historiques jouent aussi souvent les uns que les autres avec une tendance à jouer plus longtemps pour les premiers. Le comportement des joueurs historiques a aussi évolué puisqu'ils déclarent aussi jouer plus longtemps depuis la crise. **Figure 1**

Tout monde joue !

Certes, plus la tranche d'âge est élevée, moins on joue. Mais la proportion de joueurs (52%) au-delà de 54 ans reste élevée. La propension à jouer est donc transgénérationnelle, pas la propension à dépenser.

Le regard sur le jeu vidéo évolue. Celui-ci ne remplit plus seulement un rôle de divertissement pour geek, mais s'impose également comme un média à part entière consommé massivement. **Figure 2**

Ancien ou nouveau joueur, les raisons de jouer sont les mêmes

Les 2 raisons qui ressortent le plus sont une volonté de décompresser et de passer le temps, soit une certaine forme de dilettantisme. En 3e position vient un souhait d'accomplissement. Un top 3 identique en Corée du sud où l'accomplissement y est plus prégnant. En France, les jeux dits « Hyper Casual » représentent plus de 30% des téléchargements.

Le développeur indépendant qui débute dans la création de jeux mobiles devrait donc privilégier ce type de jeux à faibles coûts de production dans un premier temps : avec un gameplay simple et intuitif, des sessions de jeux très courtes de l'ordre de la minute et un visuel facile et rapide à créer.

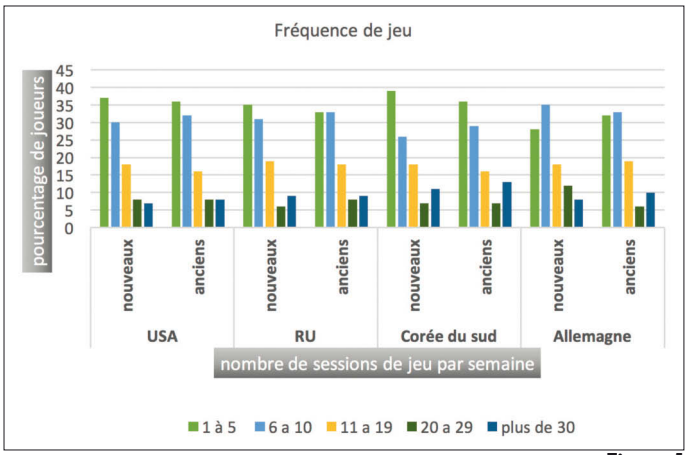


Figure 1

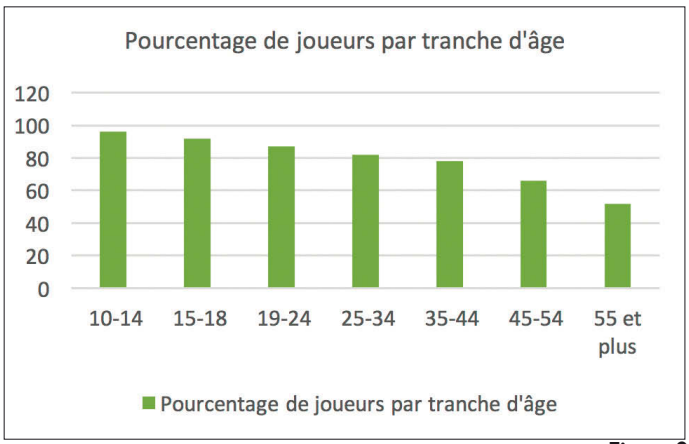


Figure 2

Solo, multijoueur et réseaux sociaux

Pour le mobile, le jeu « individuel » reste majoritaire. Cela n'a rien d'étonnant : le jeu en réseau nécessite un certain investissement incompatible avec le temps consacré à jouer.

Depuis quelques années déjà, les communautés ont le vent en poupe et le jeu vidéo n'échappe pas à ce phénomène. Une importante activité sociale sur le réseau au travers du jeu vidéo pour se faire aider dans leur progression, partager des bonus ou des scores, ou encore participer à des événements connexes. Ainsi de nouveaux réseaux sociaux dédiés comme discord ont émergé au détriment des réseaux sociaux historiques bien que ceux-ci soient encore largement utilisés. Cette tendance s'est aussi fortement accentuée du fait de l'arrivée de nouveaux joueurs.

Le streaming est aussi devenu une nouvelle manière de partager du contenu dédié aux jeux vidéo avec près de 8,4 milliards d'heures de visionnage avec en tête Twitch qui cumule plus de la moitié de la totalité des vues. De nouvelles manières de partager via le streaming dont l'usage s'est accentué depuis la pandémie avec une croissance de 91,8% d'une année sur l'autre (7,26 milliards d'heures de visionnage de juillet 2020 à septembre 2020, toute plate-forme confondue). **Figure 3**

Autant le créateur aurait de bonnes raisons de peaufiner le mode solo de son jeu vidéo, autant il serait aussi pertinent de travailler sa renommée en jouant sur la complémentarité des réseaux sociaux et du streaming en participant à des événements associant ces 2 formats. Parmi ceux-ci, on trouve OneHourGameJam (https://one-hourgamejam.com/), un concours hebdomadaire de création de jeux vidéo qui diffuse chaque semaine sur Twitch un gameplay des jeux participants.

Le mode multijoueur, étant une activité sociale ne concernant qu'une partie de joueurs, devrait être envisagé plus tard si le jeu est plébiscité.

La monétisation

Les nouveaux joueurs ont tendance à dépenser plus d'argent que leurs homologues historiques avec des motifs variés parmi lesquels on retrouve pour les joueurs les plus engagés la personnalisation de l'avatar, ou la possession de véhicules, armes ou outils rendant la progression plus aisée dans le jeu. Ces achats intégrés se font en majorité sur les jeux RPG ou de stratégie (44% en France) qui embarquent souvent un univers support d'histoires. On y trouve notamment en tête des dépenses les jeux Coin Master, Clash of Clans et Brawl Stars.

Le retrait pur et simple de la publicité, ou encore le passage de niveaux difficiles sont aussi d'autres motifs d'achat.

L'acte d'achat est grandement facilité du fait que les sommes engagées sont souvent faibles, de l'ordre de quelques euros. Et le marché ne s'y trompe pas puisque ces petits achats se sont aussi développés dans les mondes PC et consoles pour lesquels il existe aujourd'hui un marché parallèle de revente.

Toutefois, que l'on soit ancien ou nouveau joueur, le modèle gratuit de consommation arrive toujours en tête, prédomine en occident alors qu'en Asie le modèle avec achat intégré est autant privilégié que le modèle gratuit avec publicités. Toutefois, côté nouveaux joueurs, bien que toujours majoritaires, le modèle gratuit perd du terrain avec une dépréciation de l'ordre de 5 à 10 %.

Sur ce terrain-là aussi, les nouveaux joueurs se distinguent des joueurs avec une propension à l'achat plus importante et une dépense moyenne mensuelle de l'ordre de 20 € contre 13,50 € pour les anciens.

Le gratuit reste aujourd'hui le meilleur moyen d'attirer les joueurs à soi, la monétisation doit donc passer par un autre biais que l'achat pur et simple du jeu en privilégiant un modèle mixte comportant à la fois la publicité que les joueurs acceptent en grande majorité et les achats intégrés qui rassemblent à eux seuls plus de la moitié des préférences de monétisation des joueurs.

Quand est-ce qu'un joueur télécharge un jeu ?

On chiffre à plus de 350 000 le nombre de jeux mobiles disponibles en téléchargement aujourd'hui. Un fait intéressant et rassurant pour les indépendants : le quart des joueurs téléchargent des jeux dont ils n'ont jamais entendu parler.

Néanmoins, l'acte de téléchargement reste favorisé par sa popularité : on télécharge un jeu parce qu'on en a entendu parler. Le gros des téléchargements se faisant dans les 3 mois qui suivent cette connaissance (49%).

La promotion n'est donc pas une nécessité, du moins pour débuter puisque si vous faites un jeu à succès sans communication, il bénéficiera du bouche-à-oreille et de la notoriété qui s'en suivent, sources de renforcement de ce même succès.

Ce qu'il faut retenir : réalisme, qualité et notoriété

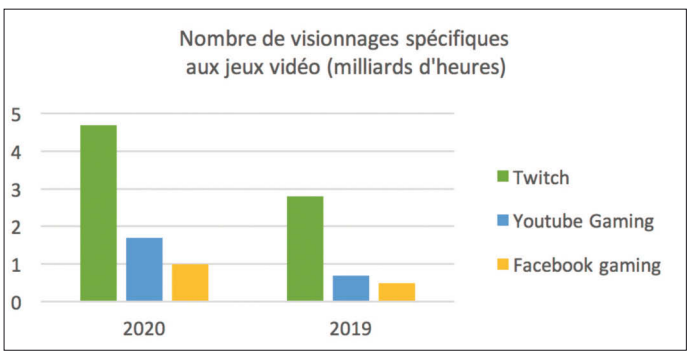
L'arrivée de nouveaux joueurs sur le marché n'est pas neutre sur l'avenir de cette industrie : elle laisse entrevoir encore un fort potentiel de croissance avec un chiffre d'affaires qui a cru d'environ 18% en France passant de 1231 millions à 1426 millions d'euros entre 2019 et 2020. Sur ce même périmètre et tout type d'achat confondu, le jeu mobile représente 27% des parts sur un marché total borné aux jeux vidéo évalué à 5,3 milliards d'euros. La bonne nouvelle ne bénéficiera pas uniquement aux grands studios. Les développeurs indépendants ne seront pas laissés-pour-compte, il y a de la place pour tout le monde.

Toutefois un certain réalisme qui tient compte du modèle de consommation jetable sera de rigueur. L'effort technique devra être efficient en regard de cette audience en misant notamment sur les jeux hyper casual.

Ce n'est pas tout puisque des terrains qui vont au-delà de l'effort technique de création devront être investis. En choisissant à la fois un modèle de rémunération mixte associant publicité et achats intégrés, et un modèle de communication abusant de la mixité des médias les plus plébiscités que sont les réseaux sociaux et sites de streaming spécialisés.

Nous reviendrons très rapidement sur le jeu et comment les développer !

Figure 3



QUELQUES TENDANCES ET POINTS À RETENIR

Le marché du jeu évolue avec les technologies et les nouvelles habitudes de consommation. L'arrivée des App Store sur iOS et Android ont bouleversé notre façon de consommer le jeu mobile. La grande tendance actuelle est incontestablement les services de stream : Google Stadia, Apple Arcade, Xbox Game Pass. Le Français Shadow avait misé sur ce marché mais difficile de tenir la cadence face aux géants (particulièrement Microsoft, Sony et Google). Le service a été racheté par OVH.

Cependant, le succès n'est pas toujours au rendez-vous. Google avait beaucoup misé sur Stadia mais le jeu n'est pas son marché traditionnel. Début juin, Google lançait la version iOS. Pour l'éditeur impossible de ne pas y aller ! Cependant, l'avenir de Stadia reste incertain. En février 2021, Google avait fermé son studio de jeux : Stadia Games and Entertainment.

Les plateformes de réalité virtuelle ou mixte restent toujours à la recherche du bon modèle. Microsoft, Oculus, HTC n'ont pas (encore ?) trouvé le déclic

La rédaction

tangente 200

l'aventure mathématique

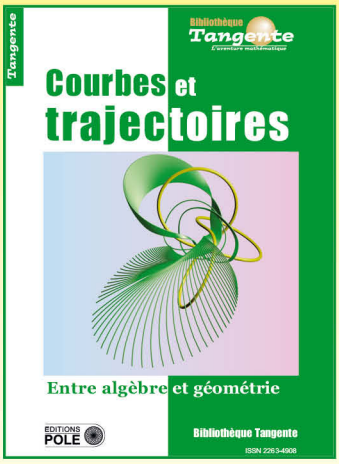
Il vient de paraître : le numéro 200 de *Tangente*

Un numéro « Collector » à ne pas manquer !

Avec son supplément à détacher : un cahier central d'œuvres d'art format A3 à afficher selon vos goûts.

Les derniers numéros parus

Le numéro 200



Bibliothèque Tangente 74

Le hors-série 78

Les numéros de *Tangente* sont disponibles :

– chez votre marchand de journaux

– chez votre libraire pour la bibliothèque Tangente

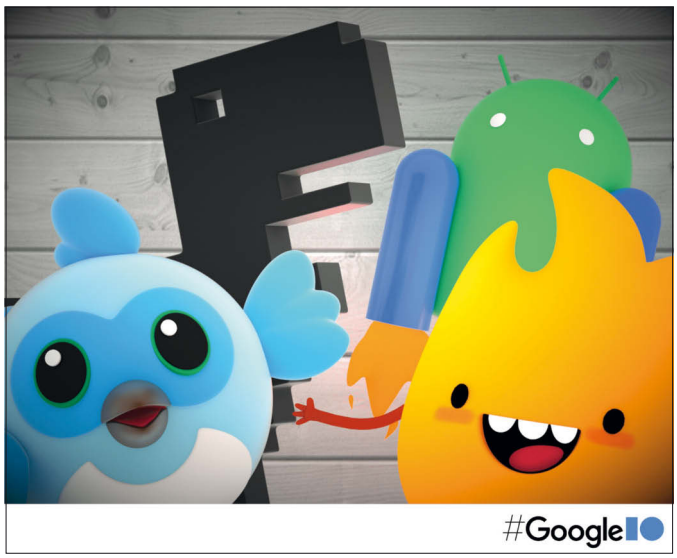
– en les commandant ou s'abonnant sur infinimath.com/librairie

– en ligne sur tangente-mag.com

Roadmap des langages

Déjà disponible

Flutter 2.2



Cette version a été annoncée à la Google I/O. La 2.2 doit apporter plus de stabilité et d'amélioration sur les différentes plateformes. On dispose de nouveaux icônes, des améliorations sur plusieurs composants d'interface. La partie web se complète avec un cache plus performant et des compléments sur l'accessibilité. Sur la partie iOS, Flutter 2.2 booste le rendu des animations et permet les installations incrémentales. Cette version corrige aussi de nombreux bugs.

Angular 12

Ivy est désormais le moteur par défaut en lieu et place de View Engine qui a été déprécié. Ce moteur sera retiré d'Angular à la prochaine version majeure. Les développeurs sont donc invités à migrer vers Ivy et à tester les apps actuels. L'outil de test Protactor est désormais déconseillé. Il est possible qu'il soit entièrement refait ou retiré. Le mode strict est activé par défaut sur Angular CLI. Ce mode doit permettre de voir les erreurs plus rapidement. Webpack 5 est supporté par Angular et TypeScript a été mise à jour avec la dernière version. D'autre part, le support de IE11 a été déprécié. Il sera totalement retiré avec la v13.

Kotlin 1.50.10

Cette version apporte plusieurs nouveautés dont support des records Java, classes Inline, interfaces sealed. Les records sont des classes Java pour le stockage des données immutables. Ce support doit améliorer l'interopérabilité Kotlin – Java. À noter que la cible JVM (compilation) par défaut est désormais la 1.8. La 1.6 a été dépréciée. Les équipes annoncent aussi de meilleures performances sur Kotlin/Native.
Tous les détails sur Kotlin 1.50.x : https://kotlinlang.org/docs/whatsnew15.html#standard-library

TypeScript 4.3

La 4.3 est sortie en mai dernier. Cette version permet notamment de spécifier les types pour lecture et l'écriture dans des propriétés, ajout de override et noImplicigtOverride pour mieux gérer l'extension des classes, ajout de #private (ECMAScript)…
Attention cette version introduit aussi des casses de code sur lib.d.ts, Promise, Union Enums. La 4.4 sortira quand vous lirez ces lignes.
Pour tous les détails : https://devblogs.microsoft.com/typescript/announcing-typescript-4-3-rc/

Cassandra 4

Ce SGBD de type NoSQL est développé par Apache. Java 9, et les versions ultérieures sont supportés. Grâce à ce support, Cassandra supporte le ZGC (ramette-miette disponible en production depuis Java 15. ZGC devrait améliorer les performances de traitement des données.
La liste des nouveautés et amélioration est très longue : https://cassandra.apache.org/doc/latest/new/

V8

V8 est le moteur JavaScript de Google. Pour l'améliorer, les équipes ont intégré un nouveau compilateur : Sparkplug. En réalité, il vient en complément de TurboFan, un compilateur d'optimisation. Sparkplug doit compiler plus rapidement et donc faire gagner du temps à TurboFan. Et pour être très performant, Sparkplug ne génère pas de représentation intermédiaire, il compile directement en code machine !
Pour comprendre toutes les subtilités : https://v8.dev/blog/sparkplug

Rust 1.56 : septembre / octobre

Cette version est aussi appelée Rust 2021. Pour les équipes, il s'agit de poursuivre les évolutions tout en assurant la stabilité du langage. L'ajout de nouvelles fonctions doit donc se faire progressivement sans provoquer de problèmes de compatibilités. Pour assurer ces évolutions, Rust utilise le concept d'éditions. Pour en savoir plus : https://blog.rust-lang.org/2021/05/11/edition-2021.html

Python 3.10 : octobre

La 3.10 devrait sortir en octobre prochain. Parmi les nouveautés prévues : opérateur Union, nouveauté sur le TypeAlias, amélioration sur plusieurs modules (base64, codecs, py_compile). Il n'est pas prévu d'ajouter de nouveaux modules. Les listes des retraits et dépréciations sont assez longues : checker la liste. Plusieurs changements sont indiqués sur les API. Là encore, soyez vigilant(e) sur la migration. Mais, il y a largement le temps pour le portage / la migration des codes actuels.

PHP 8.1 : 25 novembre

La communauté PHP prépare la 8.1. Cette version introduit plusieurs nouveautés : enums, Fibers, amélioration des performances (opcache), type never, fonction array_is_list. Cette version est considérée comme mineure et aucune casse de compatibilité n'est pour le moment annoncée.

Date inconnue

Swift 5.5

Durant la WWDC, Apple a lancé une bêta de Swift 5.5. Cette version doit apporter un grand nombre de nouveautés. Une des plus importantes est le fameux Async / await, tellement pratique pour faire de l'asynchronisme et donc de la programmation concurrente même c'était déjà possible.

React 18

La prochaine version majeure de React apportera des améliorations sur de nombreux composants, de nouvelles API, le support de React.lazy. Les développeurs veulent une migration vers la v18 en douceur sans besoin de réécrire du code. Les détails de cette version seront disponibles au fur et à mesure et la liste des nouveautés peut changer.

Jeu de plateforme multi-plate-forme en Delphi

Delphi permet d'écrire toutes les apps que l'on veut : bureau, mobile, web, front, back… mais également des jeux vidéo. Il est possible d'utiliser des moteurs de jeux existants, mais, en tant que développeur, il est très enrichissant et pas si compliqué que ça d'écrire ses propres jeux de A à Z. C'est ce que nous allons voir ! Un bon coding4fun pour passer un coding summer parfait.

Grégory Bersegeay
MVP Embarcadero (Delphi), développeur fullstack, conférencier.

Pour cet article, j'ai utilisé la dernière version de Delphi (10.4 Sydney update 2) sortie en février 2021. Toutefois, le code source est compatible avec la version 10.3 Rio et donc avec l'édition Community gratuite sous certaines conditions (https://www.embarcadero.com/fr/products/delphi). Nous utilisons uniquement Delphi et son framework multi-plate-forme Firemonkey (FMX) sans utiliser de moteurs ou de librairies dédiées « jeux ». De ce fait, il vous suffit simplement de disposer de Delphi en version 10.3 minimum pour compiler le projet.

Avant de démarrer

N'étant pas graphiste, j'ai choisi d'utiliser un kit graphique gratuit et libre de droits en l'occurrence celui de Szadi (https://szadiart.itch.io/rocky-world-platformer-set). Cet asset est complet et fournit différentes images d'arrière-plan, un tileset et des images de personnages animés (sprites cheets).

Le tileset est une image constituée de différentes tuiles (tiles en anglais). Chaque tuile peut être assemblée à d'autres tuiles afin de générer, tel un jeu de construction, les différents éléments d'un niveau du jeu. Le tileset fourni contient des tuiles de 16 x 16 pixels. Nous nous servirons de ce tileset pour constituer les différentes plateformes. Pour les autres éléments du décor, j'ai pioché dans ce tileset pour créer des images autonomes.

Dans cet article, je ne détaillerai pas entièrement le code. En revanche, nous nous attarderons sur les principaux concepts et les parties importantes du code. Vous trouverez en fin d'article les liens pour télécharger le code source et l'exécutable du jeu pour différentes plateformes (Windows, macOS et Linux).

Le code source est disponible sur programmez.com et le GitHub du magazine.

Principe du jeu

Il s'agira d'un jeu de plateforme en deux dimensions à déplacement (scrolling) horizontal. Le joueur dispose de trois essais pour récupérer les bonus (des diamants) présents dans le niveau. Pour se faire, il pourra sauter sur les plateformes, mais devra faire attention aux ennemis. En effet, s'il en touche un, le joueur perdra un essai et recommencera au début du niveau. Si le joueur ne dispose plus d'essai, la partie se termine.

Sur les plateformes de bureau, le jeu se joue au clavier, sur Android, des boutons sont affichés.

Taille de la fenêtre

Le kit graphique utilisé étant basé sur des tuiles de 16 pixels, j'ai arbitrairement taillé la fenêtre de jeu initiale à une résolution de 480 x 272 soit 30 tuiles affichées par ligne sur 17 lignes. Cela se rapproche également du format 16/9e (480 x 270).

De plus, j'ai prévu dans le code une constante nommée « scale ». Elle est positionnée à 2 par défaut et permet de zoomer l'affichage par 2. Ainsi, la fenêtre de jeu passe en 960 x 544 pixels ce qui permet de bénéficier d'une surface d'affichage suffisante sans trop dégrader les graphismes d'origine. Vous pouvez modifier la valeur de la constante scale dans le code et recompiler pour voir son effet.

Sur les cibles desktop, la taille de la fenêtre est modifiable à l'exécution, mais la « hauteur » du niveau ne sera pas modifié si l'utilisateur augmente la hauteur de la fenêtre. L'affichage restera sur 17 lignes de tuiles.

Sur la cible Android, un layout est affiché en bas de l'écran permettant d'afficher les boutons d'interaction (boutons « gauche », « droite » et « saut »).

À noter : *comme nous allons placer des objets à l'écran en 2D, il est important de bien savoir où se situe l'origine de notre repère (figure 1). Sous Delphi, en 2D, l'origine x = 0 et y = 0 correspond au coin supérieur gauche de la fenêtre. X augmente en se déplaçant vers la droite et Y augmente en se déplaçant vers le bas. La propriété position des composants graphiques correspond également aux coordonnées du coin supérieur gauche du composant.* **Figure 1**

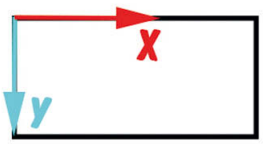


Figure 1

Création du projet sous Delphi

Nous partons d'un nouveau projet de type « Application multi-périphérique » afin d'utiliser le framework FMX.

Nous plaçons quelques composants graphiques comme illustré à la **Figure 2**.

La vue structure (**Figure 3**) montre l'arborescence des éléments graphiques. Sous Firemonkey, n'importe quel composant peut contenir d'autres composants. Cela s'avère très pratique, car les composants enfants héritent des propriétés de leurs parents (position, rotation, mise à l'échelle…).

Pour modifier la « généalogie » d'un composant rien de plus simple : il suffit de le déplacer à la souris via glisser/déposer dans cette vue structure.

Détaillons rapidement les composants placés :

- **FormMain** : est le composant représentant la fenêtre de

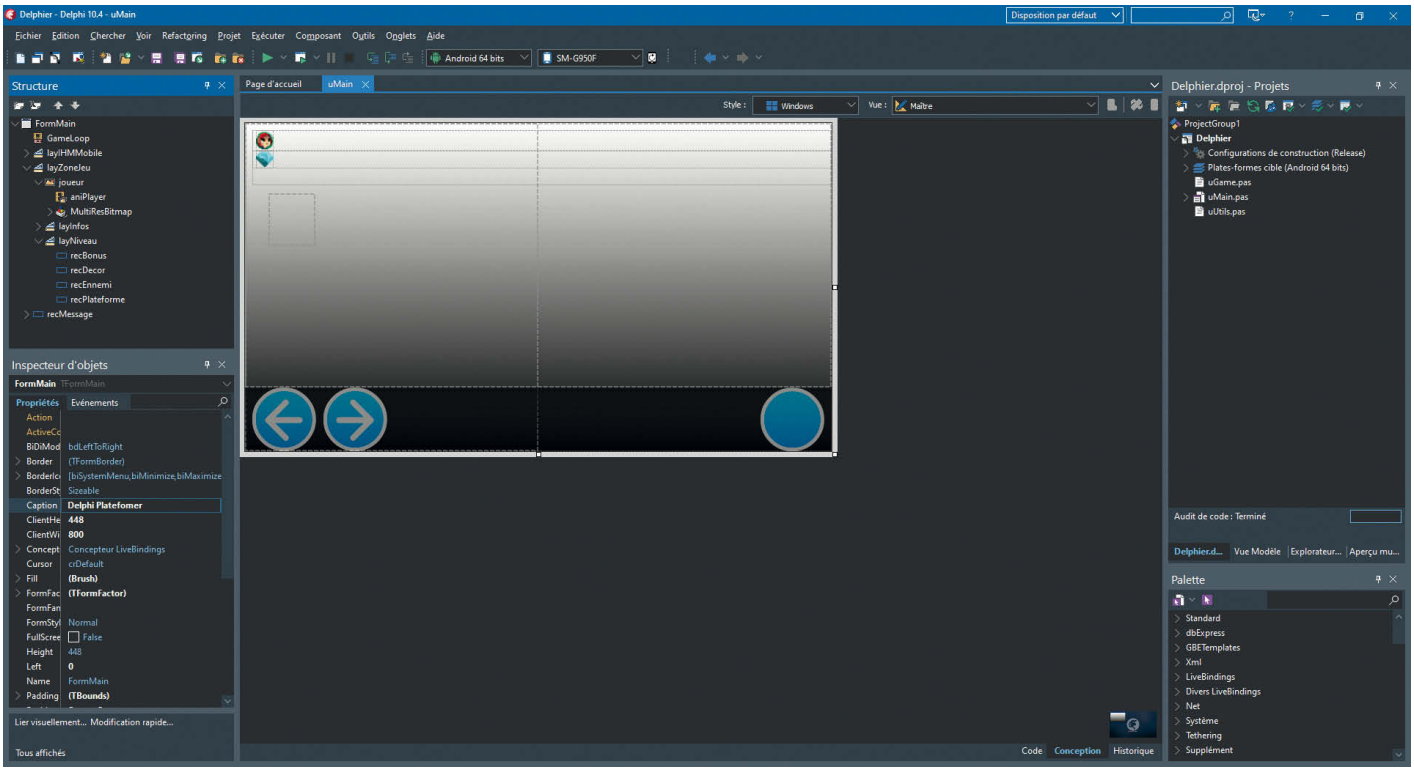


Figure 2

Figure 4

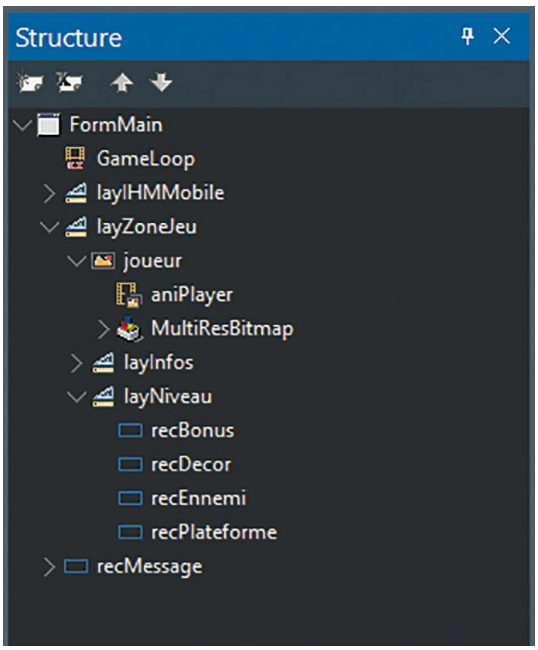


Figure 3

l'application. C'est le parent de tous les autres ;

- **GameLoop** est un TFloatAnimation qui sert de boucle principale du jeu (nous détaillerons ce point plus loin) ;
- **layZoneJeu** est un TLayout qui représente toute la zone de jeu. Il occupe toute la partie cliente de FormMain. La constante **scale** agit sur ce composant pour zoomer sur la zone de jeu ;
- **joueur** est un TImage qui représente le sprite du joueur. Le sprite est un acteur du jeu. Il s'agit d'une image qui peut être animée et qui se déplace à l'écran. Pour animer notre sprite, nous affectons un objet enfant de type TBitmapListAnimation. Ce dernier permet de gérer une image de type sprite cheet (c'est-à-dire une image qui contient les différentes étapes d'une animation voir l'exemple **figure 3**). Ce TBitmapListAnimation est nommé **aniPlayer**.**Figure 4**
- **layInfos** : ce TLayout permet d'afficher les informations sur le nombre de vies disponibles et le nombre de diamants trouvés ;
- **layNiveau** : ce TLayout correspond au plan du niveau. Il contient donc les plateformes, les ennemis, les bonus et les éléments additionnels du décor. Le sprite du joueur ne fait pas partie de ce plan. En effet, aux extrémités gauche et droite du niveau, le sprite du joueur peut se déplacer jusqu'au bord de la fenêtre (notion de bornage que l'on verra plus tard). Le reste du temps, le sprite du joueur est centré dans la zone de jeu de la fenêtre.
  Dans le layNiveau, j'ai placé quatre TRectangle. Ils serviront de conteneurs :
  - **recBonus** : contiendra les bonus ;
  - **recDecor** : contiendra les décors additionnels ;
  - **recEnnemis** : contiendra les ennemis ;
  - **recPlateforme** : contiendra les plateformes.
- **layIHMMobile** : Ce TLayout n'est affiché que sur cible Android. Il contient les boutons permettant à l'utilisateur d'interagir : reculer, avancer et sauter.
- **recMessage** : est un TRectangle qui sera utilisé pour afficher le message de victoire ou de défaite le moment venu.

Le projet se compose de la fiche principale **uMain.pas** et de deux unités supplémentaires **uGame.pas** et **uUtils.pas**.

J'ai placé dans uGame.pas des éléments propres au jeu (constantes, déclaration de types et les méthodes de détection des collisions). Dans uUtils.pas, on y retrouve des méthodes utilitaires qui permettent par exemple de charger une image, récupérer une tuile depuis l'image tileSet en fonction son indice, etc.

Fichier décrivant le niveau

Le niveau du jeu est décrit dans un fichier JSON. Ce fichier contient tous les éléments constituant le niveau : plateformes, ennemis, décors, bonus, plans, animations, etc.

Le fait d'externaliser la description du niveau dans un fichier permet de rendre le projet plus souple et générique. Il est ainsi possible de créer des niveaux supplémentaires ou de modifier le niveau sans avoir à toucher au code. De plus, il est possible (et fortement conseillé) de créer un éditeur de niveau qui permettra de créer des niveaux facilement et rapidement.

Détaillons la structure de ce fichier. Tout d'abord, nous avons des paramètres globaux :

- width : permet de définir la largeur du niveau en nombre de tuile ;
- tileSize : permet de définir la taille des tuiles (comme indiqué précédemment, l'asset graphique utilisé pour ce tutoriel utilise des tuiles de 16x16 pixels) ;
- tileImage : indique le nom de l'image tileset ;
- playerStartPosX : position de départ du joueur en abscisse ;
- gravity : la gravité à appliquer (influence les sauts).

Ensuite, nous trouvons un tableau nommé « scrollings ». Il contient la liste des plans qui seront utilisés. Pour chaque plan, nous avons les informations suivantes :

- image : l'image utilisée pour ce plan ;
- vitesse : la vitesse de déplacement du plan relative à la vitesse du déplacement du plan contenant le niveau de jeu ;
- posY : position du plan sur l'axe des ordonnées ;

Nous détaillerons un peu plus loin la gestion de ces plans.

Un autre tableau nommé « plateformElement » permet de définir les plateformes. Les plateformes seront des rectangles constitués d'une à plusieurs tuiles du tileset. Chaque plateforme est définie par :

- image : une liste de nombres séparés par le caractère « : ». Chaque nombre correspond à l'indice de la tuile dans l'image tileSet. L'image tileSet est lue par bloc de « tileSize » pixels en partant du coin supérieur gauche jusqu'au coin inférieur droit du tileSet ;
- animation : permet de définir une éventuelle animation à appliquer à la plateforme. Il s'agit d'une chaîne de caractères devant respecter une certaine syntaxe. Nous détaillerons cette syntaxe lorsque nous aborderons les animations plus tard dans cet article. En mettant « none », il n'y aura pas d'animation particulière.

Ensuite, le tableau « decorElement » contient les images que j'ai créées à partir de l'image tileSet afin de disposer de blocs de décor à placer où l'on souhaite dans le niveau.

Le tableau « bonusElement » définit les bonus. Un bonus est constitué d'une image pouvant être une sprite cheet. Comme pour le sprite du joueur, nous utiliserons également un objet TBitmapListAnimation. Il suffit de lui indiquer le nombre d'étapes de l'animation (propriété « count ») et sa durée en seconde (propriété « duration »). Il se charge alors d'afficher l'image, partie par partie.

Le tableau « ennemiElement » définit les ennemis. Nous utiliserons à nouveau un TBitmapListAnimation auquel nous pouvons ajouter une autre animation (déplacement : même syntaxe que les animations d'une plateforme). Il peut y avoir deux fichiers images à renseigner : par exemple dans le cas où l'ennemi doit faire demi-tour.

Enfin, le dernier tableau nommé « niveau » permet d'assembler les différents éléments définis précédemment pour constituer le niveau. Ce tableau contient les sous-éléments suivants :

- decor : permet de déclarer les éléments de décor à afficher dans le niveau. Chaque élément de décor est défini par trois valeurs séparées par des « : » : son indice dans decorElement, sa position sur l'axe Y en multiple de tileSize, sa position sur l'axe X en multiple de tileSize.
  Pour délimiter les éléments de décors, il faut les séparer par le caractère « ; ».
- plateformes : même chose que pour les éléments de décor, mais pour placer les plateformes.
- bonus : même principe pour placer les bonus.
- ennemis : même chose pour les ennemis.

Gestion des plans

Bien qu'en 2D, nous utilisons la technique du défilement parallaxe pour donner un effet de profondeur au jeu. Nous allons donc gérer plusieurs plans qui se superposent et défilent chacun à une vitesse donnée. Les plus « éloignés » (en arrière-plan) ont une vitesse moins élevée que les plans plus proches. La **figure 5** illustre cette superposition des plans.

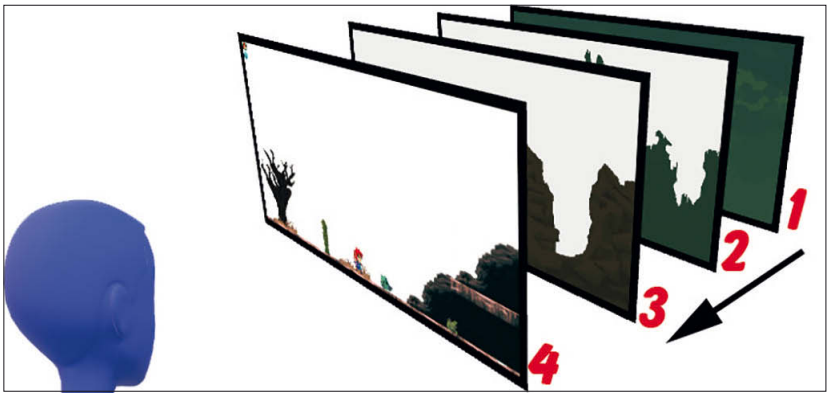
Les plans 1, 2 et 3 correspondent aux plans définis dans la rubrique « scrollings » du fichier JSON. Ils sont affichés dans l'ordre de déclaration dans le fichier, comme indiqué par la flèche sur la **figure 5**.

Le plan 4 est le plan du niveau de jeu. Celui-ci contient les plateformes, les éléments de décor, les bonus, les ennemis et le joueur.

Pour gérer les plans, nous créons la classe TScrolling qui hérite de la classe TLayout de la manière suivante :

```
TScrolling = class(TLayout)
private
  fRectangle: TRectangle;
  fVitesse: single;
```

Figure 5



```
public
  constructor Create(AOwner: TComponent); override;
  destructor Destroy; override;

  property Rectangle: TRectangle read fRectangle write fRectangle;
  property Vitesse: single read fVitesse write fVitesse;
end;
```

Le TScrolling contient ainsi un TRectangle et un attribut de vitesse. L'avantage d'utiliser un composant TRectangle plutôt qu'un TImage pour les plans est qu'il dispose d'une propriété Fill. Cette propriété permet de définir le fond du rectangle de plusieurs manières. Celle qui va nous intéresser est sa propriété Bitmap. En effet, on peut ainsi choisir la manière dont l'image va remplir le fond du rectangle. Nous choisissons le mode duplication qui répétera autant de fois que nécessaire l'image d'origine pour remplir le fond du rectangle.
La largeur du rectangle est calculée en fonction de la largeur de la zone de jeu et de la largeur de l'image utilisée. Pour que le déplacement donne l'impression d'un décor perpétuel, la largeur du rectangle doit être :
largeur de l'image * ((largeur de la fenêtre / largeur de l'image) + 1)
La **figure 6** illustre cela. Le cadre bleu symbolise la fenêtre d'affichage. L'image de fond (montagne noire) est dupliquée 3 fois pour remplir le fond de la fenêtre, mais il faut avoir une 4e fois l'image pour pouvoir décaler l'image de fond vers la gauche (et donc simuler un déplacement de gauche à droite). L'ensemble des 4 images de fond correspond à notre rectangle.
Une fois le décalage du rectangle vers la gauche effectué et arrivé à la situation de la **figure 7**, on replace le rectangle à sa position d'origine pour se retrouver à nouveau dans la situation de la **figure 6**.
Le code suivant est extrait de la méthode qui parse le fichier JSON, plus particulièrement la rubrique « scrollings », et qui instancie un objet TScrolling. Pour chaque scrolling, nous indiquons que son parent est layNiveau. Nous chargeons l'image dans le fond du rectangle et nous taillons la largeur du rectangle avec la formule vue précédemment :

```
var scrollingArr := JSONValue.GetValue<TJSONArray>('scrollings');
for var I := 0 to scrollingArr.Count-1 do begin
```

Figure 6

Figure 7

```
var unScrolling := TScrolling.Create(nil);
unScrolling.Parent := layZoneJeu;
unScrolling.Position.Y := scrollingArr[i].GetValue<integer>('posY');
chargerImage(unScrolling.rectangle.Fill.Bitmap.Bitmap, TPath.Combine(GetAppResources
Path,scrollingArr[i].GetValue<string>('image')));
unScrolling.rectangle.Width := unScrolling.rectangle.Fill.Bitmap.Bitmap.width *
((formMain.ClientWidth / unScrolling.rectangle.Fill.Bitmap.Bitmap.width) + 1);
unScrolling.rectangle.Height := unScrolling.rectangle.Fill.Bitmap.Bitmap.Height;
unScrolling.Height := unScrolling.rectangle.Fill.Bitmap.Bitmap.Height;
unScrolling.vitesse := scrollingArr[i].GetValue<single>('vitesse');
listeScrollings.Add(unScrolling);
end;
```

La gestion du scrolling perpétuel est faite dans la méthode deplacerScrollings dont voici le code :

```
procedure TFormMain.deplacerScrollings(vitesse: single);
begin
 for var scrolling in listeScrollings do begin
  scrolling.position.X := scrolling.position.X + (vitesse * scrolling.vitesse);
  if direction = TDirection.droite then begin
   if scrolling.position.X <= - scrolling.Rectangle.Fill.Bitmap.Bitmap.Width then
scrolling.position.X := 0;
  end else begin
   if scrolling.position.X >= 0 then scrolling.position.X := - scrolling.Rectangle.Fill.
Bitmap.Bitmap.Width;
  end;
 end;
 layNiveau.Position.X := layNiveau.Position.X + vitesse;
end;
```

Cette méthode est appelée à chaque itération de la boucle principale du jeu. La vitesse de déplacement d'un scrolling est déterminée en fonction de la vitesse de déplacement du layNiveau et de l'attribut vitesse propre au scrolling.

Constitution du niveau

Le niveau est créé dynamiquement en fonction des informations contenues dans le fichier JSON. La lecture du fichier est faite dans la méthode chargerNiveau. Celle-ci va récupérer les informations du fichier puis générer le niveau en invoquant les méthodes genererDecor, genererPlateformes, genererBonus et genererEnnemis.
Ces méthodes instancient des objets TImage en leur affectant les images, les positions, éventuellement les animations et vont les affecter au conteneur correspondant recDecor, recPlateforme, recBonus ou recEnnemi.

Boucle principale

Dans le développement d'un jeu vidéo, on retrouve systématiquement une boucle principale dont on ne sort que lorsque le joueur a gagné ou perdu. À chaque itération dans cette boucle, nous devons gérer l'affichage, les collisions, l'interface utilisateur, etc.
Les traitements réalisés dans cette boucle doivent être les plus rapides possibles afin de rendre le jeu fluide. La persistance rétinienne de l'œil humain est de 1/25 seconde. Il faut donc itérer au minimum 25 fois par seconde dans cette boucle pour que le jeu paraisse fluide. Toutefois, la percep-

tion de la fluidité d'un jeu mannettes en main est tout autre et, souvent, un jeu est considéré fluide à partir de 60 images par seconde.
Cette boucle peut être gérée de plusieurs manières. J'ai choisi d'utiliser un objet TFloatAnimation, car cet objet est de lui-même géré dans un thread à part ce qui permet de ne pas bloquer le thread principal du programme.
Ce TFloatAnimation est l'objet nommé **GameLoop** pour lequel nous implémentons son événement **OnProcess** comme suit :

```
procedure TFormMain.GameLoopProcess(Sender: TObject);
begin
 lblNbVie.text := 'x '+nbVie.ToString;
 lblNbPierre.Text := nbBonus.ToString + '/'+nbBonusMax.ToString ;
 vitesseY := vitesseY + gravite;
 joueur.Position.Y := joueur.Position.Y + vitesseY;
 gererTouches;
 testerFinDeJeu;
 testerCollisions;
 if layNiveau.Position.X >= 0 then begin
  bornerAGauche;
  exit;
 end;
 if layNiveau.Position.X <= -(tailleNiveau) then begin
  bornerADroite;
  exit;
 end;
 deplacerScrollings(vitesseX);
end;
```

Dans cet événement, nous mettons à jour l'affichage des compteurs de vie et de bonus trouvés. Nous vérifions également l'état des touches afin de réagir aux actions de l'utilisateur.
Ensuite, nous mettons à jour la vitesse de déplacement verticale du joueur (vitesseY) en l'incrémentant de la gravité.
Nous vérifions si nous atteignons une condition de fin de jeu (victoire ou défaite), nous testons les collisions (détails un peu plus loin), nous bornons l'affichage à droite ou à gauche et nous appelons la méthode deplacerScrollings vue précédemment.
Nous bornons l'affichage afin d'empêcher le joueur d'aller vers la gauche lorsque l'on est au début du niveau et d'aller vers la droite lorsque l'on est à la fin du niveau. De plus, lorsque le joueur atteint ces bornes, le sprite du joueur se déplacera sur le bord gauche ou droit de la fenêtre et ne sera plus au centre de la fenêtre. Ce petit effet permet de mieux se rendre compte du début et de la fin du niveau.

Gestion des touches

Pour jouer, l'utilisateur doit utiliser les touches suivantes du clavier pour jouer :

- Flèche droite ou touche D pour aller vers la droite ;
- Flèche gauche ou touche Q (ou A sur un clavier Qwerty) pour aller vers la gauche ;
- Flèche haut, barre espace ou touche Z (ou W sur un clavier Qwerty) pour sauter.

Pour gérer cela, nous utilisons les événements OnKeyDown (appuie sur une touche) et OnKeyUp (l'utilisateur relâche la touche) de la fiche FormMain.
Petite astuce : dans ces événements nous nous contentons de positionner des flags de type boolean en fonction des touches appuyées. Les actions correspondantes seront réalisées dans la boucle principale via l'appel à la méthode **gererTouches** dont voici le code :

```
procedure TFormMain.gererTouches;
begin
 if toucheAvancer then avancer;
 if toucheReculer then reculer;
 if toucheSauter then sauter;
end;
```

Cette astuce permet de ne pas remplir le buffer clavier en cas de traitement long à réaliser dans les actions.
De plus, le fait de passer par des booléens lors d'appuis sur les touches permet très facilement d'autoriser d'autres moyens d'interagir. Par exemple, sur Android ou iOS, il n'y a pas de clavier. Nous avons prévu d'afficher pour les plateformes mobiles le TLayout layHMMobile qui contient des boutons. Il suffit alors de positionner ces mêmes booléens lorsque l'utilisateur clique ou appuie sur les boutons.

Déplacements du joueur

Nous avons déjà vu brièvement dans la boucle principale qu'une vitesse verticale était gérée. Celle-ci fait tomber en permanence le sprite du joueur tant que celui-ci n'entre pas en collision avec une plateforme ou un ennemi.
Une vitesse horizontale (vitesseX) est également gérée. Celle-ci permet de se déplacer latéralement. Elle est positionnée à une valeur constante (vitesseXMax) lorsque l'utilisateur appuie sur une des touches permettant de se déplacer vers la droite ou la gauche. Dès que l'utilisateur relâche cette touche, viteseX est remise à zéro.
Les sauts sont gérés par la méthode **sauter** dont voici le code :

```
procedure TFormMain.sauter;
begin
 if enCollisionAvecPlateforme then begin
  sautEnCours := true;
  orienterSaut;
  vitesseY := - puissanceSaut;
 end;
end;
```

Nous partons du principe que le joueur ne peut sauter que s'il est en collision avec une plateforme. En effet, pour sauter, il faut pouvoir prendre appui !
Si tel est le cas, alors nous mettons la variable sautEnCours à true. Nous appelons la méthode **orienterSaut** qui permet de sélectionner l'animation du saut au sprite du joueur en fonction de sa direction.
Enfin, nous forçons la vitesse verticale (vitesseY) à moins la valeur constante puissanceSaut. Moins, car, pour rappel, l'axe Y est orienté vers le bas et que le saut doit faire monter le sprite du joueur…

Et c'est tout pour le saut ! Ce n'est pas bien compliqué et nous n'avons pas eu besoin de compétences mathématiques particulières ! Le simple fait de modifier la vitesseY avec la gravité dans la boucle principale fait ensuite le travail.

Les animations

Pour les animations, nous utilisons les animations standards fournies avec Delphi. Nous avons déjà vu l'objet TBitmapListAnimation pour animer les sprites et le TFloatAnimation pour la boucle principale. Il en existe bien d'autres (pour jouer sur les couleurs, les paths, les dégradés…).

Pour déplacer les ennemis et les plateformes, nous allons utiliser à nouveau le TFloatAnimation. Cet objet permet de faire évoluer un nombre flottant d'une valeur de départ jusqu'à une valeur d'arrivée dans un laps de temps défini. De nombreuses possibilités sont offertes comme par exemple, boucler indéfiniment, autoreverse, choisir le mode d'interpolation…). Dans les faits, on attache comme parent de l'objet TFloatAnimation le composant sur lequel doit agir l'animation. La propriété PropertyName de l'animation permet alors d'associer une propriété compatible du composant parent.

Par exemple, pour le TFloatAnimation que l'on rattache à un TImage (une plateforme par exemple), on va pouvoir jouer sur une propriété du TImage du type flottant comme par exemple sa position en abscisse, en ordonnée, son opacité…

Lors de la création du niveau, nous avons créé les ennemis et les plateformes à partir des informations lues dans le fichier JSON. Nous avions vu que ces éléments pouvaient être animés. La syntaxe des animations est liée aux options que l'on va donner à notre animation. Cette syntaxe est particulière et les paramètres de l'animation définis dans le fichier JSON doivent être séparés par des « : ».

Le premier paramètre correspond à la propriété de l'animation (« x » l'animation impactera la position sur l'axe X, « y » l'animation se fera sur la propriété position sur l'axe Y, « o » l'animation se fera sur l'opacité de la plateforme).Le second paramètre est la durée de l'animation en seconde.

Le troisième paramètre correspond à la valeur de fin de l'animation (la valeur de départ étant la valeur à la création de l'objet). Cette valeur sera un multiple de tileSize.

Le quatrième paramètre est le type d'interpolation. Plusieurs interpolations sont possibles (linéaire, sinusoïdale, quadratic, bounce, elastic…).

Le cinquième paramètre est le type d'animation et permet de faire jouer l'interpolation au début de l'animation (« in »), à la fin (« out ») ou aux deux moments (« InOut »).

Enfin, les deux derniers paramètres sont des flags permettant d'indiquer si l'animation boucle ou non (« loop » ou « noloop »), et si l'animation s'inverse automatiquement ou non (« autoreverse » ou « noautoreverse »).

Exemple : pour déplacer une plateforme indéfiniment sur l'axe X de 5 tuiles (5x16 pixels dans notre exemple) en faisant des allers/retours en 10 secondes avec une interpolation sinusoïdale, on définira l'animation :

"x:10:5: sinusoidal:InOut:loop:autoreverse"

Gestion des collisions

Dernier point important pour un jeu de plateforme, il s'agit de la gestion des collisions. Dans notre exemple, nous utilisons les coordonnées et la taille des TImage. Cette méthode n'est pas précise, mais elle est simple et suffisante pour cet exemple.

Nous avons vu que la méthode **testerCollisions** est appelée dans la boucle principale. Voici le code de cette méthode :

```
procedure TFormMain.testerCollisions;
begin
 var position := getPositionJoueur;
 if joueur.Position.Y <= (joueur.Position.Y + vitesseY) then begin
  var resultat := gererCollisionsPlateformes(position, joueur, recPlateforme);
  enCollisionAvecPlateforme := resultat.enCollision;
  if enCollisionAvecPlateforme then begin
   setAnimationPrecedente;
   sautEnCours := false;
   joueur.Position.Y := resultat.objet.Position.Y - joueur.Height;
   if resultat.objet.ChildrenCount = 1 then begin
    if (resultat.objet.Children[0] as TFloatAnimation).PropertyName.ToLower =
'position.x' then begin
     if oldPlateformePosX = 0 then oldPlateformePosX := resultat.objet.Position.X;
      deplacerScrollings(oldPlateformePosX - resultat.objet.Position.X);
      oldPlateformePosX := resultat.objet.Position.X;
     end;
   end;
   vitesseY := 0;
  end;
 end else enCollisionAvecPlateforme := false;
 if not(enCollisionAvecPlateforme) then oldPlateformePosX := 0;

 if gererCollisions(position, joueur, recBonus, true).enCollision then inc(nbBonus);
 if gererCollisions(position, joueur, recEnnemi, false).enCollision then toucher;
end;
```

On récupère tout d'abord la position du joueur. En effet, le TImage joueur n'étant pas parent du layNiveau, sa position en abscisse ne prend pas en compte la position en abscisse du niveau. Pour obtenir la position du joueur dans le niveau, il faut donc ajouter la position du layNiveau à la position du rectangle du joueur. Plus exactement, on ajoute la valeur absolue de la position du layNiveau, car celui-ci se décale vers la gauche (donc sa position en X est négative).

Les collisions avec les plateformes ne seront à calculer que si le joueur est en train de descendre. C'est-à-dire lorsque sa position en ordonnée est inférieure à sa position en ordonnée + vitesseY.

Pour tester s'il y a collision avec une plateforme, nous invoquons la méthode **gererCollisionsPlateformes** définie dans uGame.pas. Nous lui passons 3 paramètres : la position du joueur, le TImage du joueur et le rectangle recPlateforme contenant les plateformes. S'il y a collision, alors on force la position en Y du joueur en fonction de la limite haute de la plateforme (sa position sur l'axe Y). De plus, si la plateforme est en mouvement latéral, on doit appliquer en plus son déplacement au joueur. Si la plateforme est en déplacement vertical, la gestion des collisions combinée à la gestion de la gravité gèrent ce déplacement du joueur sur l'axe Y.

Voyons maintenant la fonction **gererCollisionsPlateformes** :

```
function gererCollisionsPlateformes(positionJoueur : TPoint; joueur: TImage; un
Rectangle : TRectangle):TCollision;
begin
 result.enCollision := false;
 for var recChild in unRectangle.Children do begin
  var unEnfant : TImage := recChild as TImage;
  if gererCollisionsJoueurAvecUnePlateforme(positionJoueur, joueur, unEnfant) then begin
    result.enCollision := true;
    result.objet := unEnfant;
    break;
  end;
 end;
end;
```

Cette fonction va parcourir tous les enfants du recPlateforme. Pour chaque enfant, on teste s'il y a collision avec le joueur en appelant la méthode **gererCollisionsJoueurAvecUne Plateforme**. S'il y a collision, on arrête la boucle de parcours des enfants du recPlateforme et on retourne un objet de type TCollision (un boolean indiquant s'il y a collision ou non, et l'éventuelle plateforme avec laquelle il y a eu collision).
La fonction **gererCollisionsJoueurAvecUnePlateforme** est la suivante :

```
function gererCollisionsJoueurAvecUnePlateforme(positionJoueur : TPoint; joueur,
 plateforme : TImage): boolean;
begin
 Result := ((positionJoueur.X + joueur.Width * 0.75) > plateforme.Position.x) and
       ((positionJoueur.X + joueur.Width * 0.25)< plateforme.Position.X + plateforme
.width) and
       (positionJoueur.Y + joueur.Height > plateforme.Position.Y) and
       (positionJoueur.Y < plateforme.Position.Y) and
       (plateforme.Opacity > 0.3);
end;
```

Cette fonction renvoie un boolean indiquant s'il y a collision ou non entre le joueur et l'objet passé en paramètre. La collision est détectée en fonction de la position en X et Y du joueur, de sa largeur, de sa hauteur et de la position en X et Y de la plateforme. On ne contrôle que le haut de la plateforme (sa position en Y) et pas le bas de la plateforme (position en Y + sa hauteur), car on souhaite que le joueur soit bloqué par la plateforme uniquement si le joueur arrive par le dessus de la plateforme. Dans le code, nous testons également l'opacité de la plateforme. Cela permet de gérer des plateformes qui apparaissent ou disparaissent progressivement.
La **figure 8** montre le contour du joueur (en jaune) et une plateforme (son contour rouge). Ce sont donc les positions de ces TImage qui déterminent s'il y a collision ou non. Les positions sont les coins supérieurs gauches des images.

À noter : *pour les contrôles sur les positions X, nous appliquons un pourcentage de la largeur du joueur. En effet, le dessin du personnage n'occupe pas les 32 pixels de large de l'image. L'utilisation d'un pourcentage de la largeur du joueur permet d'avoir un meilleur rendu lorsque le joueur se trouve au bord d'une plateforme. En effet, les pixels transparents de l'image du joueur comptent dans la méthode de détection des collisions.*

La collision avec les bonus et les ennemis est plus simple. Dans les deux cas, nous passons par la fonction **gererCollisions** qui prend en paramètre la position du joueur, les TImage du joueur et du rectangle contenant les éléments (soit les bonus, soit les ennemis) et un boolean indiquant si l'objet collisionné doit être détruit ou non.
La fonction **gererCollisions** va parcourir tous les enfants du rectangle conteneur passé en paramètre et tester pour chacun d'eux s'il est en collision avec le joueur. Le code est le suivant :

```
function gererCollisions(position : TPoint; joueur : TImage; unRectangle : TRectangle;
 detruireObjetTouche: boolean):TCollision;
begin
 var enCollision := false;
 for var recChild in unRectangle.Children do begin
  var unEnfant : TImage := recChild as TImage;
  if (position.X > unEnfant.Position.X) and (position.X < unEnfant.Position.X + un
Enfant.Width) and
    (position.y + joueur.Height > unEnfant.Position.Y) and (position.y < unEnfant.
Position.Y + unEnfant.Height) then begin
     enCollision := true;
     result.objet := unEnfant;
     if detruireObjetTouche then unEnfant.free;
     break;
  end;
 end;
 result.enCollision := enCollision;
end;
```

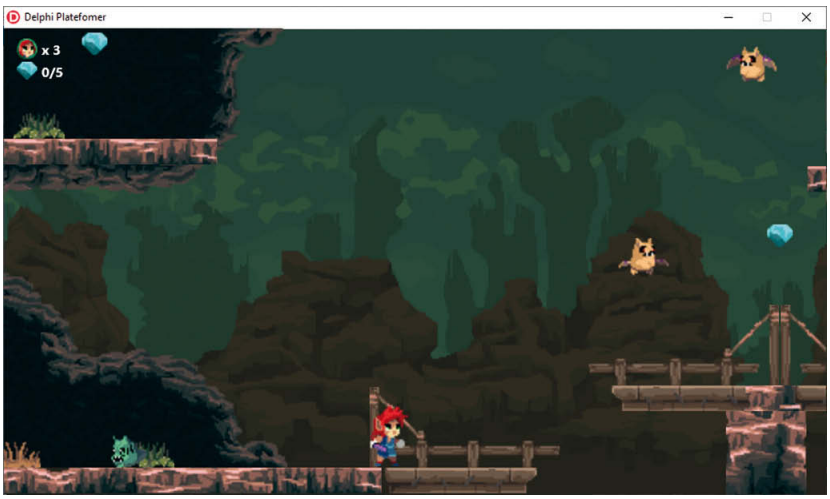
Pour ces collisions, nous contrôlons simplement si le TImage du joueur entre en collision avec le TImage de l'objet.

Résultat

La **figure 9** montre une capture d'écran du jeu tournant sous Windows.
Le **figure 10** montre le jeu sous Android.
La **figure 11** montre le jeu sous Linux.
Et enfin, la **figure 12** qui montre le jeu tournant sous macOS.

Figure 8

Figure 9



Vous trouverez les sources de ce projet sur mon GitHub : https://github.com/gbegreg/Delpier
Vous trouverez également les binaires pour Windows, macOS, Linux et Android : https://github.com/gbegreg/Delpier/bin

Le jeu a été testé sous :

- Windows 10 64 bits : FMX utilise alors DirectX ;
- macOS Big Sur 64 bits processeur Intel : avec Delphi en version 10.4, il est possible d'indiquer à FMX d'utiliser Metal au lieu d'OpenGL (déprécié par Apple). C'est ce qui est fait dans le code du projet via les lignes suivantes :

```
{$IFDEF MACOS}
  GlobalUseMetal := True;
{$ENDIF}
```

- Ubuntu 20.04 64 bits : FMX utilise alors OpenGL ;
- Android 9 64 bits : FMX utilise alors OpenGL ES.

Pour aller plus loin

Cet exemple n'est évidemment qu'une ébauche. De multiples améliorations et ajouts sont possibles pour transformer cette démonstration en véritable jeu : ajouter des sons, des niveaux, des effets, de nouvelles actions possibles par le joueur, gérer un nombre de points, un chronomètre pour limiter le temps de jeu…
Une autre amélioration envisageable serait d'utiliser le moteur physique Box 2D : il est en effet fourni en standard avec Delphi. Le jeu disposerait ainsi d'une véritable physique et d'une gestion des collisions bien plus fine. Une autre source d'inspiration possible est de regarder ce que proposent les moteurs de jeux. À ce propos, sachez que le célèbre GameMaker Studio (https://www.yoyogames.com/fr/gamemaker) est développé avec Delphi.

Au final, la seule limite sera votre imagination !

Conclusion

Au terme de cet article, nous avons vu les principes de base d'un jeu de plateforme en 2D. Nous les avons implémentés sans utiliser de moteur de jeu et pour un total d'environ 800 lignes de code seulement ! D'autant plus que sur ces 800 lignes, 200 lignes sont dédiées au parsing du fichier JSON et à la génération dynamique du niveau.
Bon amusement !

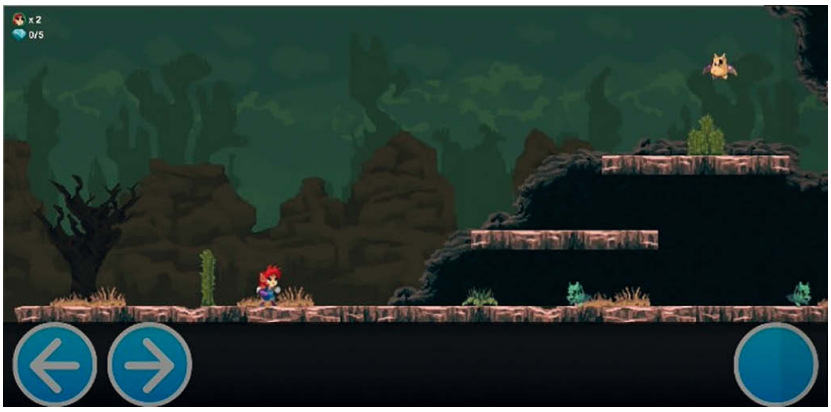


Figure 10

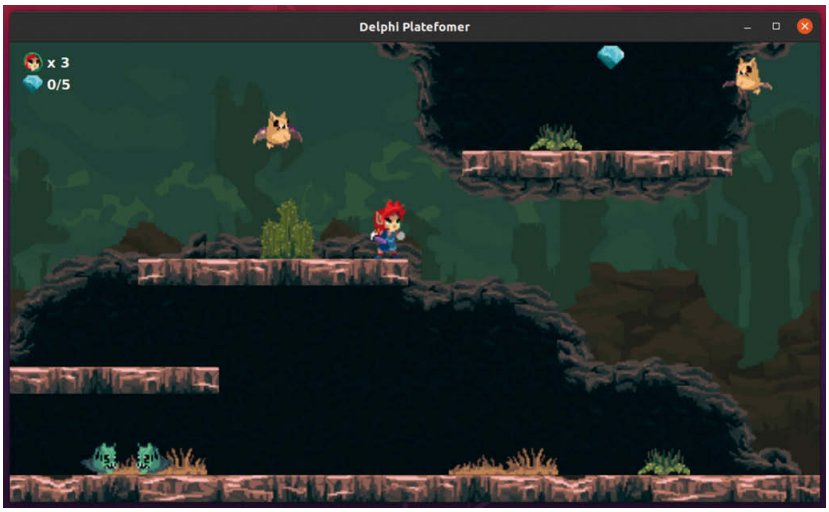


Figure 11

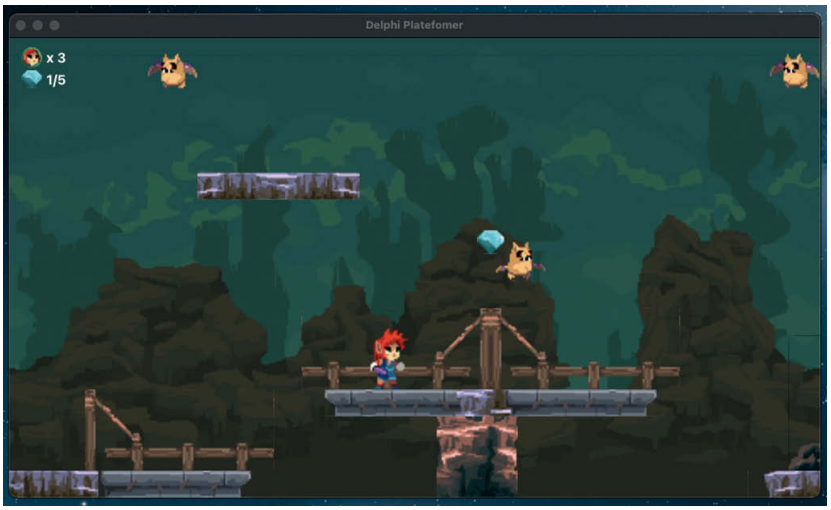


Figure 12

Regex : pourquoi il faut aimer les expressions régulières ? PARTIE 2

Marwa THLITHI
Ingénieur R&D chez INVIVOO

Utilisée à la manière des outils de recherche de texte dans un document, une expression régulière, fournit un moyen concis et flexible pour la correspondance de chaînes de texte, tel que des caractères particuliers, mots ou motifs de caractères.

Regroupement

L'utilisation des parenthèses permet d'avoir une sorte de regroupement qui est très utile quand nous faisons de l'extraction de plusieurs informations. Prenons comme exemple d'utilisation l'extraction de l'adresse mail de l'expéditeur et l'adresse mail du destinataire de la chaîne de caractère de l'exemple précédent. Dans ce cas, nous allons écrire une regex qui comporte un groupe correspondant à l'adresse mail de l'expéditeur et un autre groupe correspondant à l'adresse mail du destinataire. Voici une proposition de regex :

```
z = re.findall('^From.*? (\S+@\S+) .* to (\S+@\S+)', chr)
print(z)
```

Le résultat est une liste de tuple contenant les deux groupes : l'adresse mail de l'expéditeur et l'adresse mail du destinataire :

```
[('marwa.thlithi@invivoo.com', 'admin@invivoo.com')]
```

Ces groupes sont numérotés et nous pouvons y accéder un par un avec la fonction **group()** :

```
adresses_mail = re.match('^From.*? (\S+@\S+).*to (\S+@\S+)', chr)
chaine_complete = adresses_mail.group(0)
print(chaine_complete)
expediteur = adresses_mail.group(1)
print(expediteur)
destinataire = adresses_mail.group(2)
print(destinataire)
```

Le résultat est :

```
From marwa.thlithi@invivoo.com Sat Jan 5 09:14:16 2020 to admin@invivoo.com
marwa.thlithi@invivoo.com
admin@invivoo.com
```

L'indice 0 est un argument par défaut qui correspond à l'ensemble de la regex. Nous pouvons aussi utiliser la fonction **groups()** qui permet d'obtenir un tuple contenant les chaînes pour tous les sous-groupes :

```
adresses_mail = re.match('^From.*? (\S+@\S+).*to (\S+@\S+)', chr)
adresses_mail.groups()
```

le résultat est affiché dans l'ordre en commençant par le premier sous-groupe jusqu'au dernier :

```
('marwa.thlithi@invivoo.com', 'admin@invivoo.com')
```

NB : *les fonctions* **group()** *et* **groups()** *sont applicables sur des objets de type* **MatchObject**. *Pour cela, j'ai utilisé la fonction* **re.match()** *dans l'exemple ci-dessus. Nous pouvons aussi utiliser la fonction* **re.search()** *qui retourne un* **MatchObject**.

Groupes nommés

Le regroupement par des ensembles numérotés permet de capturer des sous-chaînes intéressantes et y accéder en utilisant les numéros de groupes, ce qui permet d'avoir des expressions régulières structurées. Néanmoins, quand il s'agit des regex complexes, le suivi des numéros de groupes devient très difficile. Dans ce cadre, une fonctionnalité de nommage de groupes a été proposée afin de référencer directement les groupes par leurs noms, ce qui permet d'avoir des regex encore plus structurées et plus claires. La syntaxe du nommage des groupe est **(?P<nom>** ***expr***) où ***nom*** est le nom du groupe et ***expr*** est la regex de la sous-chaîne à extraire.
Reprenons l'exemple précédent et réalisons-le maintenant avec des groupes nommés :

```
chr = 'From marwa.thlithi@invivoo.com Sat Jan 5 09:14:16 20 to admin@invivoo.com'
adresses_mail = re.match('^From.*? (?P<expediteur>\S+@\S+).*to (?P<destinataire
>\S+@\S+)', chr)
expediteur = adresses_mail.group('expediteur')
print(expediteur)
destinataire = adresses_mail.group('destinataire')
print(destinataire)
```

Le résultat d'affichage est le même que celui avec les groupes numérotés, sauf qu'en termes de lisibilité de la regex, cette version est beaucoup plus claire et significative grâce au nommage des groupes :

```
marwa.thlithi@invivoo.com
admin@invivoo.com
```

De plus, nous pouvons récupérer directement les groupes nommés dans un dictionnaire dont les clés correspondent aux noms des groupes indiqués dans la regex. Cela est réalisable avec la méthode **groupdict()** :

```
adresses_mail.groupdict()
```

Le dictionnaire obtenu est :

```
{'expediteur':'marwa.thlithi@invivoo.com','destinataire':'admin@invivoo.com'}
```

Prenons un autre cas d'utilisation des groupes nommés qui consiste à extraire la date et l'heure de la chaîne de caractères utilisée ci-dessus :

```
date_heure = re.search('^From.*? \S+@\S+ (?P<jour>[a-zA-Z]+)\s(?P<mois>[a-
zA-Z]+)\s(?P<jour_mois>\d+)\s(?P<heure>\d+:\d+:\d+)\s(?P<annee>\d+)',chr)
date_heure.groupdict()
```

Voici le résultat :

```
{'jour': 'Sat',
'mois': 'Jan',
'jour_mois': '5',
'heure': '09:14:16',
'annee': '2020'}
```

Sans le nommage des groupes, la regex utilisée ne sera pas compréhensible et la récupération de chaque sous-chaîne ne sera pas aussi simple qu'avec les groupes nommés.

Compilation des expressions régulières

La création d'une expression régulière revient à la création d'un graphe appelé « automate fini », qui est considéré comme une machine à états. Donc, si vous êtes amenés à utiliser une expression régulière plusieurs fois, vous allez créer la même machine à états correspondante plusieurs fois. Cela possède un coût non négligeable. Afin de minimiser ce coût et de gagner en performances, les expressions régulières peuvent être compilées en objets motifs possédants des méthodes diverses telles que la recherche et la substitution. La fonction permettant de compiler une regex du module « **re** » est **re.compile()**. Compilons l'expression régulière de l'exemple d'extraction d'adresse mail de l'expéditeur :

```
import re
compiled_regex = re.compile('^From.*? (?P<expediteur>\S+@\S+)')
print(compiled_regex)
```

Voici l'objet retourné :

```
re.compile('^From.*? (?P<expediteur>\\S+@\\S+)')
```

Cette fonction fournit un champ optionnel qui permet de modifier le comportement des expressions régulières. Ces options de compilation sont accessibles dans le module « **re** » avec deux types de nom : un nom long et un nom court en une seule lettre. Parmi les options les plus utilisées, nous citons les options « **IGNORECASE** » et « **VERBOSE** » dont les noms courts correspondants sont « **I** » et « **X** », respectivement.
L'option « **I** » permet d'avoir une correspondance insensible à la casse, ce qui facilite l'écriture des regex pour certaines recherches. Par exemple, le motif [A-Z] correspond aussi aux lettres minuscules. Voici un exemple :

```
regex = re.compile('spam', re.I)
regex.search('Spam')
```

Son résultat est :

```
<_sre.SRE_Match object; span=(0, 4), match='Spam'>
```

L'utilisation de l'option « **I** » avec la regex « spam » dans cet exemple facilite la recherche et elle correspond ainsi à « Spam », « SPAM », « sPam »…
L'option « **X** » permet d'avoir des expressions régulières plus claires et lisibles en les écrivant en multilignes avec des commentaires. L'activation de cette option implique l'ignorance des blancs (les espaces) dans la regex, sauf lorsque le blanc se trouve dans une classe de caractères ou est précédé d'une barre oblique inversée. Voici une version plus compréhensible de l'expression régulière que nous avons utilisée dans la section précédente pour l'extraction de l'adresse mail de l'expéditeur avec des commentaires que nous avons ajoutés grâce à l'option « **X** » :

```
regex = re.compile('''^From # commence par le mot From
                      .*?  # n'importe quel caractère plusieurs fois non vorace
                       \s   # espace
                       (    # commence l'extraction
                      \S+ # n'importe quel caractère sauf l'espace
                      @
                      \S+
                      )''', re.X)
```

Nous pouvons spécifier plusieurs options de compilation en appliquant l'opérateur OR. Par exemple, « re.I | re.X » active à la fois les options « I » et « X ». Reprenons l'exemple de la partie précédente qui consiste à extraire la date et l'heure et écrivons-le maintenant de façon plus lisible avec ces options :

```
regexp = re.compile('''^From    # commence par le mot From
          .*?     # n'importe quel caractère plusieurs fois
                  # non vorace
          \s      # espace
          \S+@\S+\s # le caractère @ précédé et suivi par n'importe
                  # quel caractère sauf l'espace
          (?P<jour>[a-z]+) # au moins une lettre de la classe
                    # [a-zA-Z] en utilisant l'option
                     # Ignorecase
          \s
          (?P<mois>[a-z]+)\s
          (?P<jour_mois>\d+)\s
          (?P<heure>\d+:\d+:\d+)\s
          (?P<annee>\d+)''', re.I|re.X)

date_heure = regexp.search(chr)
date_heure.groupdict()
```

Nous avons pu commenter notre expression régulière complexe en utilisant l'option « X », ce qui la rend plus lisible. Nous l'avons aussi optimisée en spécifiant pour les groupes *jour* et *mois*, qui contiennent des lettres minuscules et majuscules, uniquement la classe [a-z] grâce à l'option « I ».

Caractères et séquences d'échappement

Si vous souhaitez qu'un caractère spécial d'expression régulière fonctionne normalement, il faut ajouter comme préfixe la barre oblique inversée (« \ ») avant le caractère ou bien le mettre entre deux crochets. Voici un exemple d'utilisation :

```
import re

chaine = 'We just received $10.00 for cookies'
y = re.findall('\$[0-9.]+', chaine)
print(y)
```

Dans le code suivant, nous avons utilisé cette regex :

\$[0-9.]+	Le symbole réel du caractère « $ » suivi par un chiffre ou un point au moins une fois

Nous pouvons aussi utiliser cette regex :

[$][0-9.]+	Le symbole réel du caractère « $ » suivi par un chiffre ou un point au moins une fois

Le résultat est :

```
['$10.00']
```

À ce caractère d'échappement s'ajoutent la plupart des séquences d'échappement standards utilisées pour les chaînes littérales telles que « \u », « \U » et « \N ». La séquence « \N » a été ajoutée dans la version 3.8 de Python et elle s'utilise de la façon suivante : « \N{***nom***}» avec ***nom*** est le nom du caractère

Unicode. L'avantage de cette séquence d'échappement est d'avoir une version de regex encore plus claire et compréhensible. Prenons un exemple d'utilisation qui consiste à extraire le nom d'une marque déposée. En voici une première proposition du code avec une regex utilisant la séquence « \u » :

```
import re

produit = 'Nutri-Bio® céréales Blé & Avoine'
marque_deposee_regex = re.compile('(?P<marque>\S.+)\u00AE\s')
nom_marque = marque_deposee_regex.search(produit).group('marque')
print(nom_marque)
```

Dans cette version d'expression régulière, nous avons utilisé le code du caractère Unicode marque déposée en mettant cette séquence « \u00AE ». Cela nous permet d'avoir le résultat attendu qui est le nom de la marque :

```
Nutri-Bio
```

Néanmoins, pour bien comprendre cette regex, il faut revenir à la table de caractères Unicodes pour déchiffrer le symbole correspondant au code que nous avons utilisé. Afin de rendre cette regex plus compréhensible et claire, nous pouvons utiliser la séquence d'échappement « \N » suivie du nom du caractère Unicode qui est « *registered sign* » et qui est plus significatif que le code « \u00AE ». Et voici une version plus cool de notre regex :

```
marque_deposee_regex = re.compile('(?P<marque>\S.+)\N{registered sign}\s')
```

Barre oblique inversée

Comme nous venons de le voir, la barre oblique inversée « \ » est un caractère d'échappement utilisé pour intégrer des caractères spéciaux sans que leur signification spéciale ne soit invoquée. Mais dans le cas où la barre oblique inversée fait partie de la phrase telle que la phrase « C:\Users\Default », la regex devra contenir « \U » et elle doit échapper le caractère « \ » pour qu'il ne soit pas considéré comme un caractère spécial. Afin que ce caractère soit échappé, l'expression régulière devra être alors « '\\U' », mais pour échapper encore le caractère « '\' » dans un littéral chaîne de caractères Python, on devra écrire la regex « '\\\U' ».

Mais dans une regex complexe et/ou contenant plusieurs barres obliques inversées, cette syntaxe devient rapidement illisible et difficile à comprendre. La solution consiste à utiliser des r-strings. En effet, l'utilisation du modificateur « r » comme préfixe à la chaîne de la regex implique que Python ne va tout simplement pas parcourir la chaîne à la recherche de caractères spéciaux et il va l'utiliser littéralement. Donc, la regex de la phrase exemple sera « r'\\U' ».

Mise en pratique

Pour finir, abordons deux exemples d'utilisation: le premier affiche les différentes adresses URL d'un fichier et le deuxième extrait toutes les versions de release de Python ainsi que les informations correspondantes. Le fichier que nous utilisons pour ces deux exemples est LICENSE_PYTHON.txt.

Voici le code du premier exemple :

```
import re
regex_url = re.compile('http[s]?://www.[a-z./]*', re.I)

with open('C:\\Users\\Tools\\Anaconda3\\LICENSE_PYTHON.txt', 'r') as file:
  print('URLs :\n')
  for line in file:
    url_info = regex_url.search(line)
    if url_info:
      print(url_info.group(0))
```

Dans cet exemple, nous avons compilé notre regex, car elle est utilisée plusieurs fois dans la boucle *for* et nous avons utilisé l'option de compilation « I » afin de rendre la correspondance insensible à la casse puisque l'URL peut contenir des lettres en minuscules et en majuscules. Nous avons utilisé la notion de groupe en faisant appel à la fonction **group()** avec l'indice 0 et donc à la totalité de la sous-chaîne correspondante à la regex, qui est l'adresse URL. Et voici le résultat :

```
http://www.cwi.nl
http://www.cnri.reston.va.us
https://www.python.org/psf/
http://www.opensource.org
http://www.pythonlabs.com/logos.html
```

Dans le deuxième exemple, nous cherchons à afficher toutes les releases de Python à partir de la version 1.6 et pour chaque version, nous affichons l'année de sa sortie et la version de laquelle elle dérive. Voici une proposition de code :

Code complet sur programmez.com & github

Dans cet exemple, nous avons aussi commencé par compiler notre expression régulière et nous avons utilisé l'option « X » afin d'avoir une regex plus lisible en l'écrivant en multilignes et plus compréhensible avec les commentaires que nous avons ajoutés. Nous avons nommé les groupes avec des noms significatifs ce qui rend cette regex complexe encore plus compréhensible et nous avons utilisé la fonction **groupdict()** afin d'obtenir directement en sortie un dictionnaire dont les clés sont les noms des groupes. D'où ce résultat :

```
Releases de python à partir de la version 1.6 :
{'Release': '1.6 ', 'Derived_from': '1.5.2', 'Year': '2000'}
{'Release': '2.0 ', 'Derived_from': '1.6', 'Year': '2000'}
{'Release': '1.6.1 ', 'Derived_from': '1.6', 'Year': '2001'}
{'Release': '2.1.2 ', 'Derived_from': '2.1.1', 'Year': '2002'}
{'Release': '2.1.3 ', 'Derived_from': '2.1.2', 'Year': '2002'}
{'Release': '2.2 and above', 'Derived_from': '2.1.1', 'Year': '2001-now'}
```

Conclusion

Nous avons vu que les regex sont un moyen très puissant, énigmatique et amusant une fois que vous les comprenez. Les regex lières sont un langage en soi qui permet de faire, de manière très efficace, des recherches, des validations, des extractions de données, du filtrage… Dans cet article, nous avons parlé du module « **re** » de Python qui permet de manipuler les regex et de ses fonctions principales qui sont :

- Les fonctions de recherches et de validation : **search()** et **match()**
- La fonction d'extraction : **findall()**
- La fonction de sauvegarde et de compilation des regex : **compile()**

Nous avons aussi parlé de la notion de regroupement qui permet d'avoir des regex plus claires et compréhensibles grâce au nommage des groupes et d'extraire les données plus facilement en faisant appel aux fonctions magiques **group()**, **groups()** et **groupdict()**.

A vous de jouer !

Coder son thème Wordpress de 0

Benoît SAKOTE
Développeur Web
Free-lance
adam050986@yahoo.fr
https://github.com/BenoitAdam94

Wordpress est un des CMS les plus utilisés, je ne vous apprends rien. Une des principales critiques qui lui est faite est qu'il est difficile d'en faire un site à l'apparence unique. Effectivement, l'aspect front-end et la customisation est une part importante de l'identité de votre site et de sa navigation. Mais vous allez voir qu'il est finalement assez simple de coder son propre thème. Avec ce dossier, fini d'avoir x modules pour personnaliser l'interface que vous souhaitez !

Commençons par rappeler les solutions qu'offrent Wordpress par défaut :
- Utiliser un thème gratuit (~5000 sont proposés par défaut)
- Utiliser un thème payant ou une version payante d'un thème gratuit
- Utiliser un plug-in pour créer un thème
- Créer un thème Wordpress « from scratch ».

C'est cette dernière option que nous allons voir dans ce numéro. Avant de commencer, petit avertissement : les normes WP évoluent constamment, et il se peut que certaines informations soient incomplètes ou ne répondent pas exactement aux cahiers de charge attendue lors de la soumission d'un thème sur Wordpress.org. **Nous allons aborder ici les grandes lignes afin d'avoir un thème Wordpress fonctionnel simple avec un minimum de fonctions.** Libre à vous de l'améliorer et de le configurer selon votre niveau d'exigence.

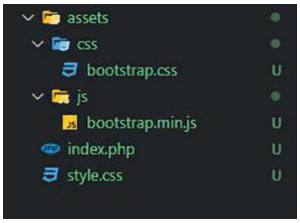


Figure 1

Wordpress.com ou Wordpress.org ?

Wordpress.com est la version commerciale, limitée (et cher !) de Wordpress. Nous utiliserons ici Wordpress.org qui est open source !

À titre d'exemple, la version « Business » de Wordpress.com vous coûtera 25 €, alors qu'un hébergeur accompagné d'une installation Wordpress.org vous coûtera 6 €/mois.

J'utiliserais ici un serveur PHP en local (**Wamp** dans mon cas avec **PHP 7.2**), **Wordpress 5.7** (que vous sur https://wordpress.org/download/) ainsi que la librairie CSS **bootstrap 4.x** (on l'utilisera essentiellement pour le système de grille). Nous supposons que vous connaissez un minimum ces outils.

La première page

Nous aurons besoin de 3 choses pour un premier thème fonctionnel (vous trouverez un fichier ZIP pour démarrer rapidement sur le site Programmez! ou sur mon GitHub : https://github.com/BenoitAdam94/programmez-wordpress-theme):
- Un dossier qui porte le nom du thème (ici : « programmez »), placé dans **wp-content\themes**
- Un fichier **index.php**
- Un fichier **style.css**

À cela nous rajoutons **Bootstrap** comme ceci :
- assets/css/bootstrap.css
- assets/js/bootstrap.min.js

Le contenu de votre dossier de travail devrait ressembler à ça **Figure 1**.

Le fichier **index.php** aura ce contenu :

```
<!DOCTYPE html>
<html lang="fr">

<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- CSS -->
    <link rel="stylesheet" href="assets/css/bootstrap.css">
    <link rel="stylesheet" href="style.css">

    <title>Mon Blog</title>
</head>

<body>
    <header class="container">
        <div class="row">
            <div class="col-12 border">LOGO</div>
        </div>
    </header>

    <main>
        <div class="container">
            <div class="row">
                <nav class="col-3 border">NAV</nav>
                <div class="col-6 border">CONTENU</div>
                <div class="col-3 border">SIDEBAR</div>
            </div>
        </div>
    </main>

    <footer class="container">
        <div class="row">
            <div class="col-4 border">FOOTER 1</div>
            <div class="col-4 border">FOOTER 2</div>
            <div class="col-4 border">FOOTER 3</div>
        </div>
    </footer>
```

```
    <!-- Javascript-->
    <script src="assets/js/bootstrap.min.js"></script>
</body>

</html>
```

Le fichier **style.css** contiendra pour le moment les informations essentielles en en-tête (header) :

```
/*
Theme Name : Programmez !
Text Domain: programmez
Version 1.0
Description : Fancy left sidebar Theme
Tags: programmez, bootstrap
Author: Benoit SAKOTE
Author URI: https://github.com/BenoitAdam94
*/
```

Vous pouvez dès à présent tester votre thème en allant dans votre back-end, puis « Apparence » puis « Thèmes » pour choisir « Programmez ! ». Si vous voulez une image d'illustration, il vous faudra un fichier nommé « screenshot.jpg » (PNG marche aussi) à la racine du dossier /programmez/
Une fois votre thème activé, rendez-vous sur la page principale de votre Wordpress. Le résultat attendu ici est une page blanche avec des colonnes (**Figure 2** (sans la barre d'admin))

Nos premières fonctions

Le contenu que nous avons collé est du HTML pur. Nous allons modifier certains attributs afin qu'ils puissent changer dynamiquement selon les paramètres du blog Wordpress.

HTML Attribute

Ce sont les attributs basiques de la balise <html> (notamment la langue). On va donc remplacer :

```
<html lang="fr">
```

Par :

```
<html <?php language_attributes(); ?>>
```

META

Idem pour la balise <meta>, mais la syntaxe est foncièrement différente :

```
<meta charset="utf-8">
```

Devient :

```
<meta charset="<?php bloginfo('charset'); ?>">
```

Liens relatif pour le CSS

Il existe une autre méthode pour activer le CSS (via **functions.php**), mais dans notre cas nous ajouterons simplement à l'URL la fonction qui permet de retrouver le chemin relatif :

```
<link rel="stylesheet" href="assets/css/bootstrap.css">
<link rel="stylesheet" href="style.css">
```

Devient :

```
<link rel="stylesheet" href="<?php echo esc_url(get_template_directory_uri()); ?>/assets/css/bootstrap.css">
<link rel="stylesheet" href="<?php echo esc_url(get_template_directory_uri()); ?>/style.css">
```

WP_Head() ;

Wp_head est une fonction qui insère des fonctions cruciales (Scripts, styles et Meta) et également la **barre d'admin (figure 02)**. C'est elle qui peut également gérer le Title et le CSS, à condition que les fonctions de support soient activées dans **functions.php** (nous y reviendrons plus tard).
Ajoutez simplement la fonction avant la balise </head> :

```
<?php wp_head(); ?>
```

BODY CLASS

La balise <body> peut également faire appel à des attributs Wordpress, il suffit de la coder comme ceci :

```
<body <?php body_class(); ?>>
```

Liens relatif pour le JavaScript

Idem que pour le CSS, nous ajouterons simplement un lien relatif :

```
<script src="<?php echo esc_url(get_template_directory_uri()); ?>
/assets/js/bootstrap.min.js"></script>
```

Wp_footer() ;

Idem que wp_head, mais pour les éléments à être appelé en bas de page (typiquement le JS et les librairies comme jQuery). À ajouter avant </body>

```
<?php wp_footer(); ?>
```

Avant d'aller plus loin : vérifications !

Enregistrez le fichier **index.php** et testez votre thème. Si vous avez tout bien fait, vous devriez voir la barre d'administration **(figure n°02)**, et le titre de l'onglet de votre navigateur devrait être celui qui se trouve dans Réglages => Général => Titre du site ou dans « Personnaliser » puis « identité du site » **(Figure 4)**

Le Template Hierarchy

Avant d'aller plus loin, il faut bien comprendre le système de hiérarchie des pages. Wordpress utilise une chaîne de requête pour générer la page finale qui sera affichée. Vous pouvez ainsi choisir par exemple d'avoir une page 404 unique (**404.php**). Si vous ne le faites pas, la page affichée sera calée sur **index.php**. Je vous suggère donc immédiatement de chercher **Template Hierarchy** sur **wordpress.org** afin de vous l'afficher en plein-écran **(Figure 3)**
Je vous recommande également d'installer le plugin Wordpress « Show Current Template « (dans Extensions -> Ajouter), qui vous indique la page sur laquelle vous vous trouvez actuellement **(Figure 2)**

Figure 2

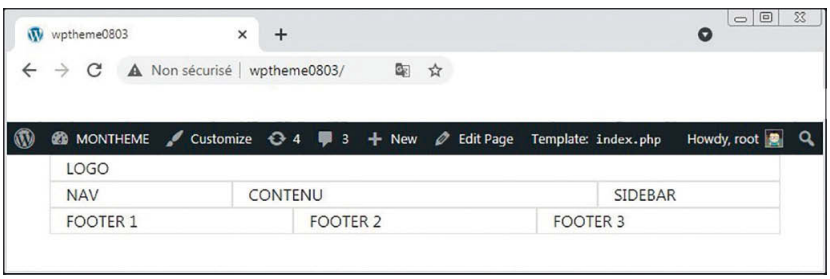


Figure 4

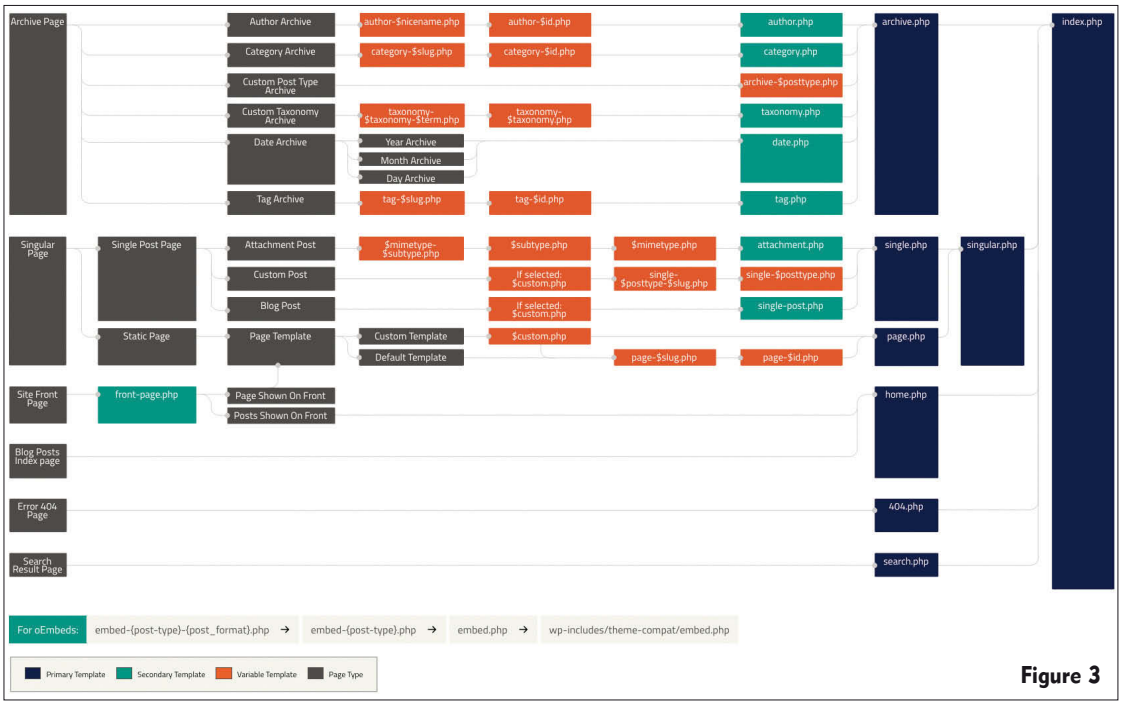


Figure 3

Title et functions.php

Le fichier **functions.php** vous permet de définir les fonctions propres à votre thème. Le nom des fonctions doit TOUJOURS avoir le nom du thème comme préfixe, ceci afin de prévenir d'éventuels conflits. Le fichier **functions.php** permet également de déclarer les éléments qui vont être utilisés dans votre thème (fonctions Wordpress, menu, widgets, CSS, scripts...)

Pour notre premier exemple, dans **index.php** nous allons supprimer entièrement cette balise :

```
<title>Mon Blog</title>
```

Et nous allons ajouter ces lignes dans un nouveau fichier **functions.php** :

```
function programmez_theme_support(){
    add_theme_support('title-tag'); // Titre du site
}

add_action('after_setup_theme','programmez_theme_support');
```

Le <title> sera définissable par l'utilisateur **(figure 4)**, et sera appelé par la fonction **WP_Head()**

Subdivision du header et du footer

Première opération, nous allons fractionner notre **index.php**. Nous allons prendre toute la partie d'en-tête (à savoir le logo et la barre de navigation latérale). Couper/Coller le code de la partie haute (jusqu'à la balise </nav) que nous mettrons dans un nouveau fichier qui s'appellera **header.php**

Rajouter ensuite la ligne suivante au tout début de **index.php** :

```
<?php get_header(); ?>
```

Même principe pour la partie basse, nous prendre tout ce qui est après CONTENU , que nous mettrons dans **footer.php**

Rajouter ensuite la ligne suivante à la toute fin de **index.php** :

```
<?php get_footer(); ?>
```

Si vous avez bien suivi, votre fichier **index.php** devrait donc ressembler à ça :

```
<?php get_header(); ?>
<div class="col-6 border">CONTENU</div>
<?php get_footer(); ?>
```

Vérifier encore que tout est bon. Si vous actualisez la page, tout devrait encore être identique à ce stade **(Figure 2)**.

Ajout du logo et du nom du site

Pour rajouter le logo, dans **header.php**, nous allons remplacer LOGO par cette fonction :

```
<?php the_custom_logo(); ?>
```

Cette fonction ne suffit pas à faire apparaître le logo, il faut ajouter le support dans **functions.php**. Ajoutez là à la fonction programmez_theme_support() en plus du logo :

```
function programmez_theme_support(){
add_theme_support('custom-logo'); // Logo Custom
add_theme_support('title-tag'); // Titre du site
}
```

Ainsi, vous pouvez changer de logo en allant dans Personnaliser (dans la barre d'admin) puis « identité du site ». N'oubliez pas de cliquer sur Publier **(Figure 4)**

Le nom du site

Plus simple à faire, mais nous allons anticiper le style en le plaçant dans un span qui contiendra une classe que nous ajouterons plus tard. Allez donc dans **header.php** pour mettre la fonction qui appelle le nom du site, entouré d'une balise contenant notre classe :

```
<span class="programmez-titre"><?php bloginfo('name'); ?></span>
```

On peut en profiter pour rajouter aussi le Slogan :

```
<em class="programmez-description"><?php bloginfo('description'); ?></em>
```

N'oubliez pas que deux informations doivent être rentrées manuellement. Allez dans Personnaliser > Identité du site. **(Figure 4)**.

Un peu de contenu

Avant d'aller plus loin, il vous faudra créer des pages et des articles de blog. Vous pouvez créer quelques pages, articles et commentaire, mais sachez qu'il existe des fichiers XML avec des articles pré-écrits et prêts à l'emploi (https://github.com/WPTT/theme-unit-test).

Pour les importer, rendez-vous dans Outils > Importer > Wordpress, choisissez le fichier XML puis cliquez sur « Téléverser et importer le fichier » **(Figure 5)**

Afficher les pages et les articles

Idéalement, nous séparons les différents types de publication (via **page.php** / **single.php**), pour l'instant, ils seront fusionnés dans la page **index.php**.

Nous allons donc initier ce qu'on appelle la « boucle » des articles. À la place de CONTENU, mettez ce code :

```
<?php
    if (have_posts()) { // Si il y a des posts à afficher
        while (have_posts()) { // Tant qu'il y a des posts à afficher
            the_post(); // Iteration de l'index
            ?>
            <h2><?php the_title(); ?></h2> <!-- Le Titre -->
            <?php
            the_excerpt(); // L'extrait
        }
    }
?>
```

Nous retrouverons régulièrement ce genre de boucle **while**, pour les commentaires par exemple (tant qu'il y a des commentaires => Les afficher)

Un menu basique

La création de menus dans un thème Wordpress peut rapidement devenir très compliquée. Pour le moment, allez dans **functions.php**, et ajoutez ceci :

```
function programmez_menus(){
    $location = array(
        'primary' => "Desktop primary left sidebar",
        'footer' => "footer Menu Items"
    );
    register_nav_menus($location);
}

add_action('init','programmez_menus');
```

Comme vous le voyez, je vous ai fait profiter de l'offre « 1 menu acheté, 1 menu offert ! ».

Rendez-vous dans **header.php** pour copier ce code à la place de NAV :

```
<?
wp_nav_menu(
    array(
        'menu' => 'primary',
        'theme_location' => 'primary',
        )
);
?>
```

Vous pouvez maintenant aller dans « Personnaliser » pour créer votre Menu à l'aide de quelques pages précédemment créer. Vous pouvez ensuite associer le menu nouvellement créer **(Figure 6)**

Le menu doit maintenant s'afficher sous forme de liste <ul> et <li> **(Figure 7)**. Il est possible de personnaliser le style du menu. On peut utiliser des options de array (n'oubliez pas https://developer.wordpress.org), mais ici on va simplement ajouter une classe à <nav> :

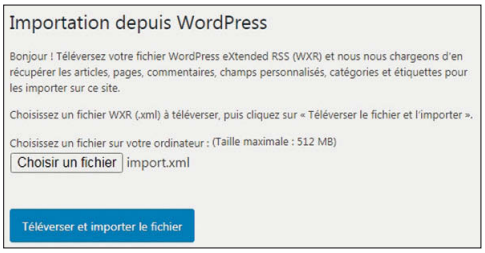
```
<nav class="col-3 border programmez-menu-class">
```

Class que nous définirons dans **styles.css** (voir plus bas dans l'article)

Bon, maintenant que nous avons ajouté un menu « primary », je vous donne un petit exercice : créer un deuxième menu dans « FOOTER 3 ». Celui-ci sera contenu de liens sociaux **(figure 8)** Si vous êtes à l'aise en CSS, vous pouvez même aller plus loin en associant un icône selon l'url ! **(Figure 9)**

Il reste encore beaucoup de possibilités de style et d'organisations. Sachez au passage que dans les recommandations, un menu doit être capable de gérer 3 sous-niveaux minimum. On peut également utiliser un paramètre **walker** qui fera appel à un autre fichier PHP (encore une fois : se référer à la doc !)

Figure 5



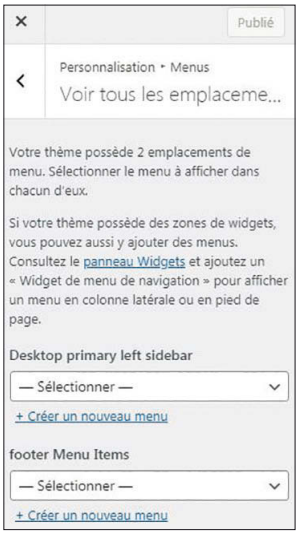


Figure 6

Figure 7

Figure 8

Figure 9

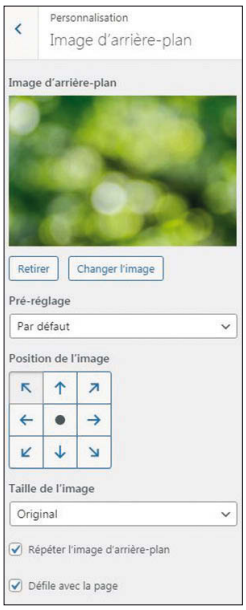


Figure 10

Un peu de style !

Nous avons encore beaucoup de choses à coder pour notre blog, mais à ce stade le site est un peu triste… alors un peu de style ne fait pas de mal pour se donner du baume au cœur !

Ajouter une couleur de fond d'écran par défaut

Nous allons créer la classe programmez-background dans **functions.php** et lui donner une couleur bleu clair :

```
.programmez-background {
    background-color:#9cadbf;
}
```

Nous devons ensuite l'ajouter la classe au <body> dans **header.php**. Nous pouvons procéder comme ceci :

```
<body <?php body_class('programmez-background'); ?>>
```

N'oubliez pas de rafraîchir le CSS en maintenant SHIFT tout en cliquant sur Actualiser !

Un fond d'écran personnalisable par l'utilisateur

On va ajouter ces quelques lignes dans la fonction "programmez_theme_support" que nous avions déjà créer auparavant :

```
    add_theme_support('custom-background');
    add_theme_support('editor-color-palette');
```

La première option permet de donner à l'utilisateur la possibilité d'utiliser une image de son choix. Cette image pourra être en mosaïque ou étirée, et autres paramètres **(Figure 10)**. La deuxième option laisse le choix d'une couleur unie à l'utilisateur via une palette de couleur.

Changer l'apparence du nom du blog

Nous pourrions mettre le nom du site en <h1>, ce n'est toutefois pas recommandé en termes de SEO (Search Engine Optimization), en particulier pour un blog qui cherche à attirer du monde avec ses articles. Bref, ici nous allons styliser le titre en ajoutant ceci dans styles.css :

```
.programmez-titre{
    font-size: 2em;
}
```

Si vos notions de CSS datent un peu, « 2em » signifie grosso modo que le texte affiché sera 2 fois plus grand.

Du style pour le menu

Il est temps de personnaliser notre menu. Commençons par le plus simple :

```
.programmez-menu-class{
    font-size: 1.25rem; /* Taille x1.25 */
    text-transform: uppercase; /* Tout passer en majuscule */
    text-align :center; /* Alignement au centre */
}
```

On peut ensuite retirer les points de la liste <ul> :

```
.programmez-menu-class ul{
    list-style: none; // Enlever les points de la liste
}
```

Ajouter un effet lorsque l'on passe la souris dessus :

```
.programmez-menu-class a:hover{
    background-color:lightgreen;
    padding-left:10px;
    padding-right:10px;
    text-decoration: none; /* Enlever le trait */
    transition-duration:1s;
}
```

Bref, les possibilités sont quasiment infinies ! En codant votre thème de 0, vous n'aurez aucune barrière, et vous pourrez même faire douter les développeurs chevronnés que c'est un WordPress qui tourne derrière votre site !

Des icônes

Vous aurez rapidement besoin d'icônes pour agrémenter votre thème. Je vous recommande les F**ont-Awesome**, car leur licence est compatible GPL, ainsi vous n'aurez pas de problème à la soumission du thème. Vous pouvez aller chercher le CDN et l'ajouter directement au header, ce qui donnait quelque chose comme ça :

```
<link rel="stylesheet" href="https://pro.fontawesome.com/releases/v5.10.0/css/all.
css" integrity="sha384-AYmEC3Yw5cVb3ZcuHtOA93w35dYTsvhLPVnYs9eStHfGJvOvKx
VfELGroGkvsg+p" crossorigin="anonymous"/>
```

Sinon, téléchargez spécifiquement le fichier CSS et le placer dans le dossier /assets/css et l'appeler comme nous avons vu précédemment avec le chemin relatif. Ceci dit c'est le bon moment pour vous montrer comment on charge un style avec **function.php** :

```
function programmez_register_styles(){
    wp_enqueue_style('programmez-fontawesome', get_template_directory_uri().
"/assets/css/all.min.css", '5.13', 'all');
}
```

N'oubliez pas de consulter la documentation pour mieux comprendre la fonction wp_enqueue_style ! N'oubliez pas également d'ajouter l'action après la fonction :

```
add_action('wp_enqueue_scripts', 'programmez_register_styles');
```

Vous pourrez ensuite tester si cela fonctionne en testant avec un icône sur l'une de vos pages :

```
<i class="fa fa-tag"></i>
```

Une page 404

Pour le coup, c'est très simple, créez une page 404.php à la racine du dossier et copiez ceci :

```
<?php get_header(); ?>
<h1>404 Page Not Found</h1>
<?php get_footer(); ?>
```

L'idéal dans ce cas est de proposer des articles de manière aléatoire. Nous allons dans notre cas simplement ajouter une barre de recherche avec la fonction suivante :

```
<?php get_search_form() ?>
```

Le formulaire de recherche redirigera vers **search.php**, mais dans le cas présent on va laisser la page index.php s'en charger (souvenez-vous du template-hierarchy !).

Des widgets !

Les widgets fonctionnent grosso modo comme les menus. On les déclare donc dans **functions.php** :

```
function programmez_widget_area(){
    register_sidebar(
        array(
            'before_widget' => '<ul class="list-inline py-3 mx-auto">',
            'after_widget' => '</ul>',
            'name' => 'Sidebar 1',
            'id' => 'sidebar-1',
            'description'  => 'Add widgets here to appear in your region.',
        )
    );
    register_sidebar(
        array(
            'before_widget' => '<ul class="list-inline py-3 mx-auto">',
            'after_widget' => '</ul>',
            'name' => 'Footer Area',
            'id' => 'footerbar-1',
            'description'  => 'Add widgets here to appear in your region.',
        )
    );
}
```

```
add_action('widgets_init', 'programmez_widget_area');
```

Il suffit ensuite de les appeler via cette fonction, à placer dans **footer.php** dans les endroits adéquats :

```
<?php dynamic_sidebar('sidebar-1'); ?>
<?php dynamic_sidebar('footerbar-1'); ?>
```

Il suffit ensuite d'aller dans « Personnaliser » puis « widgets ». Vous pouvez mettre une barre de recherche sur le Sidebar1, et un calendrier ou une image dans le footer. N'hésitez pas à rajouter plus d'emplacements des widgets !
Dans certains rares cas, il se peut que les widgets ne s'affichent pas dans la barre « personnaliser » **(Figure 11)**. Dans ce cas, rendez-vous directement dans le Back-End puis dans Apparence > Widgets.

Pages, blog, Front-page

Nous rentrons à présent dans le contenu pur. Nous allons donc développer 3 nouveaux fichiers PHP ayant chacune une spécificité :
- single.php : Un article de blog
- page.php : Une page
- **front-page.php** : Une page statique » qui est définie comme page d'accueil, indépendamment ou non de Wordpress.
Index.php sera à ce moment-là relégué à sa fonction primaire : afficher la liste des derniers articles de blog publiée.
Avant d'aller plus loin, nous aurons besoin d'ajouter le support des miniatures dans **functions.php**. Pour cela, ajoutez-le avec les autres dans la fonction « programmez_theme_support » :

```
add_theme_support('post-thumbnails');
```

Bien, améliorons maintenant notre **index.php** en faisant apparaître ces miniatures. Placez ce code avant la fonction **the_excerpt()** :

```
the_post_thumbnail('medium'); // La miniature
```

Nous allons maintenant nous occuper des articles de blog. Copier-coller entièrement le code contenu dans **index.php** dans **single.php**. Une fois cela fait, dans **single.php** remplacez :

```
the_excerpt(); // L'extrait
```

par

```
the_content(); // Le contenu
```

Copiez ensuite le contenu de **single.php** dans **page.php**
À ce stade, nous pouvons accéder aux pages de notre Wordpress via le menu, mais nous avons oublié un point important : permettre aux utilisateurs de cliquer sur un article de la page d'accueil pour le consulter dans son intégralité ! Retournons donc dans **index.php** pour ajouter un lien dynamique à notre titre, comme ceci :

```
<h2>
<a href="<?php the_permalink(); ?>">
<?php the_title(); ?>
</a>
</h2>
```

Vous pouvez faire de même avec la miniature et/ou l'extrait, tout ceci est une question de choix et de design.
Si vous voulez que la page d'accueil n'affiche ni article ni post de blog, créez donc un fichier **front-page.php** recopiez ce qui se trouve dans **page.php,** mais supprimez simplement le permalink (il n'a pas d'utilité ici). Libre à vous ensuite de personnaliser la page comme bon vous semble. Je vous conseille pour la suite de renommer ce fichier en **front-page2.php**, sans quoi nous ne pourrons accéder à un article de blog (à moins de les ajouter dans le menu ou dans un widget).

Date, Tags et commentaires

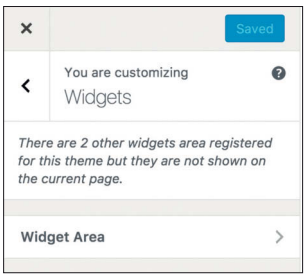
Tout d'abord, nous n'avons plus besoin de la « boucle », et nous pouvons donc commencer notre fichier **single.php** de cette manière (Repartez de 0 en supprimant tout) :

```
<?php get_header(); ?>
<div class="col-6 border">
    <h2><?php the_title(); ?></h2> <!-- Le Titre du post-->
```

Nous allons ensuite afficher la date et la liste des étiquettes (« tags » en anglais). Si votre article n'en a pas, c'est le moment de le modifier en conséquence dans le back-end.
Nous allons utiliser les **font-awesome** que nous avons défini plus tôt histoire de faire quelque chose de joli. Écrivez ensuite le code suivant :

```
<div>
    <span class="date"><?php the_date(); ?></span>
    <?php
    the_tags(
        '<span class="tag"><i class="fa fa-tag"></i> ', // Debut
        '</span><span class="tag"><i class="fa fa-tag"></i> ', // Entre 2 tags
        '</span>' // Fin
    );
    ?>
</div>
```

Figure 11



Les thèmes enfants

Pour chaque thème Wordpress, il est possible de créer un « thème-enfant ». Celui-ci est utilisé pour donner une variante de style au « thème parent ». On peut imaginer un thème avec un aspect « Magazine », avec des variantes de couleur et de typographie selon la thématique du site : actualités, cinéma, sport, musique…

Plus généralement, les thèmes enfants sont souvent utilisés pour modifier un thème téléchargé. En cas de mise à jour de celui-ci par son auteur, vos modifications seront conservées.

C'est une utilisation typique des paramètres de fonctions Wordpress qui nous permet d'ajouter du code avant et après les données qui sont reçue. Remettons ensuite la miniature et le contenu en place :

```
<?php
the_post_thumbnail('medium'); // La miniature du post
the_content(); // Le contenu
?>
```

On peut ensuite ajouter le décompte de commentaire avec encore une fois une icône font-awesome :

```
<a href="#comments"><i class='fa fa-comment'></i><?php comments_number
(); ?></a>
```

On appelle ensuite le template de la boucle commentaires :

```
<?php
comments_template(); // Appeller comments.php
?>
```

Si vous testez, vous verrez qu'une boîte affichant les commentaires apparaît. Écrivez donc un commentaire pour essayer ! **(Figure 12)**
Idéalement, il vous faut votre propre fichier **comments.php** (sinon vous aurez erreur PHP avec le Debug activé. C'est comme ça depuis quelques versions de Wordpress…).
Vous pourrez en avoir un modèle sur mon GitHub : https://github.com/BenoitAdam94/programmez-wordpress-theme

Vérifier son thème

Vérifications usuelles :
Pensez avant tout à vérifier que le site s'affiche correctement sur différents navigateurs. Vous pouvez aussi vous aider du **Validator W3C** (https://validator.w3.org/) : copiez le code HTML généré par PHP (via clic droit puis « afficher le code source de la page »). Puis entrez-le directement dans « Direct Input »). Notez que Wordpress par défaut vous générera quelques warning (il ajoute encore par défaut les « types » pour les ressources CSS/JavaScript).
Le responsive-design étant une grande part de notre thème, n'oubliez pas d'utiliser les outils du navigateur pour simuler le rendu sur un écran de tablette ou de Smartphone.

Figure 13

Vérifications spécifiques :
Avant toute chose, il faut activer le mode DEBUG en allant dans le fichier **wp_config.php** situé à la racine du site :

```
define('WP_DEBUG', false);
```

Changez **false** en **true** puis sauvegardez le fichier.
Nous allons ensuite installer et activer le plugin « **Theme Check** » **(Figure 13)**. Il sera installé dans « Apparence > Theme Check ». Choisissez le thème et cliquez sur « Check-it ». Un ensemble de messages s'afficheront en bas de page. **(Figure 14)**.
Comme vous le voyez, il reste encore pas mal de travail pour rentre le thème conforme ! À vous de lire soigneusement les messages d'erreur pour voir ce que vous pouvez corriger.

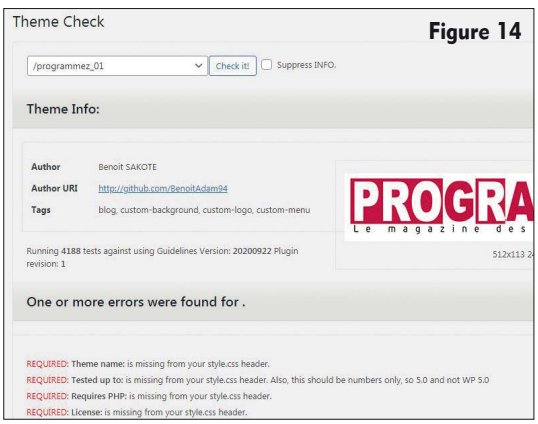
Conclusion

À ce stade, nous avons une ébauche d'un thème fonctionnel. Vous devrez encore creuser pour personnaliser le design et le rendre conforme aux normes et à votre niveau d'exigence (ou celui de votre client). À ce propos, vous pouvez vous référer à la documentation officielle (https://developer.wordpress.org/themes/). C'est parfois un peu indigeste, mais obligatoire si vous voulez que votre thème soit proposé dans les dépôts officiels.
Attention également : les recherches sur Google vous emmèneront parfois sur un site similaire : CODEX (https://codex.wordpress.org/), qui n'est selon les dires de l'équipe sur Slack, plus à jour, ni la documentation de référence.
Si vous voulez aller plus loin, je vous suggère dans un premier temps de faire d'autres tutoriels (on en trouve des très bien sur YouTube), qui vont donnerons des manières de faire différentes et vous apprendrons de nouvelles fonctions ou façon de faire. Également, vous retrouverez la communauté Wordpress française sur https://wpfr.net
Enfin, sachez que vous trouverez le thème sur mon GitHub (https://github.com/BenoitAdam94/programmez-wordpress-theme), vous pouvez le forker ou récupérer le template si vous ne le trouvez pas sur le site programmez !

N'hésitez pas à me contacter directement si vous bloquez sur une étape !

Figure 12

Figure 14



Alan Turing

La machine de Turing

En 1936, le mathématicien anglais Alan Turing a formulé la base conceptuelle d'un dispositif programmable dont nous sommes aujourd'hui tous héritiers : l'ordinateur. Vous allez découvrir dans cet article la puissance de ce concept pourtant très simple dénommée « Machine de Turing ».

Thierry Delattre

Ancien professeur de physique appliquée, Thierry s'intéresse aux sciences et techniques, notamment à l'électronique, ainsi qu'aux mathématiques. Diplômé d'une Maîtrise EEA, il a enseigné aussi dans le supérieur l'électronique et le traitement du signal. En 2012, il a créé thaM thaM et a remporté une médaille d'or du concours Lépine européen en 2013 pour son invention : le thaMographe (outil de géométrie 4 en 1 sans pointe). Il est aussi le concepteur d'une machine de Turing électronique à vocation pédagogique qui est disponible depuis 2020.

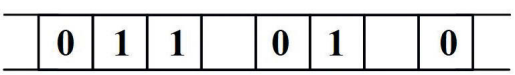
Qui peut le moins peut le plus !

Et non, il n'y a pas d'erreur ! Le concept génial imaginé par Alan Turing est étonnant de simplicité, mais est capable de grandes performances. C'est peu de le dire : toute machine de Turing, aussi rudimentaire soit-elle, est capable de faire tout ce qui peut être fait par un ordinateur moderne, même le plus puissant ordinateur du monde ! Bien sûr, il faudrait s'armer de patience, car la machine de Turing est extraordinairement moins puissante que tout processeur actuel, mais tous les spécialistes des sciences informatiques sont catégoriques : elle peut le faire ! Par ailleurs, tout ordinateur actuel est une machine de Turing dite « universelle », c'est-à-dire, un dispositif dont le programme est semblable à une donnée. Il « suffit » donc de modifier cette donnée spéciale qu'est le programme pour que la machine accomplisse une autre tâche.

Autre façon d'interpréter les choses : chaque instruction d'un langage de programmation peut être réalisée avec une machine de Turing comme celle qui sera présentée dans cet article !

Principe de fonctionnement d'une machine de Turing

Dans son célèbre article de 1936 : « On computable numbers, with an application to the Entscheidungsproblem » (à propos des nombres calculables, avec une application au problème de l'arrêt), Turing décrit son concept de la façon suivante : imaginez un ruban de longueur infinie (il s'agit donc bien d'un concept !) sur lequel est dessiné une infinité de cases. Dans chaque case est écrit un symbole ou alors la case est vide. La machine possède une tête de lecture / enregistrement lui permettant de lire ou d'écrire un symbole dans chaque case. Cette tête de lecture / enregistrement peut se déplacer de case en case tout le long du ruban :



À chaque instant la machine est dans un certain état. Arrêtons-nous un instant pour expliquer cette notion d'état, fondamental et qui est une idée absolument géniale.

Avec cet « état », la machine passe d'un système dit combinatoire (c'est comme une équation booléenne ou numérique dont la sortie ne dépend, à chaque instant, que de la valeur de la ou des variables d'entrées), à un système dit séquentiel : l'état d'une sortie dépend certes de la ou des variables d'entrées, mais aussi, soit de la valeur de la sortie elle-même soit de la valeur d'un nombre, ou l'état d'une mémoire.

Faisons cette analogie : imaginez que vous ayez soif et qu'on vous présente un verre d'eau. Comme vous êtes dans l'état « j'ai soif », face à cette situation, vous exécutez l'action suivante : vous prenez ce verre d'eau et vous le buvez. Après avoir terminé cette action, vous n'êtes plus dans l'état « j'ai soif », mais, désormais, vous êtes dans l'état « je n'ai pas soif ». Si maintenant on vous présente à nouveau un autre verre d'eau, c'est-à-dire, la même situation que précédemment, vous ne ferez plus l'action de boire. L'action de boire ayant modifié votre état interne, votre comportement change face à une même situation. Il y a là l'équivalent d'une mémoire à 2 états (« j'ai soif » et « je n'ai pas soif ») qui influence votre comportement, suite à une action passée. On dit que vous avez transité d'un état « j'ai soif » à un état « je n'ai pas soif ».

Dans une machine de Turing, l'état est simplement un nombre.

Ceci étant dit, quel genre d'instruction est capable d'exécuter la machine de Turing ? Et bien par exemple : si l'état est égal à 12 et que la case lue contient « 0 », alors : écrire « 1 » dans cette case, aller à la case de droite, puis passer à l'état 17. Un algorithme est constitué de plusieurs instructions de ce type. Les premiers algorithmes permettaient de faire des calculs mathématiques (les 4 opérations de base), des suites de nombres (par exemple, la suite de Fibonacci), la détermination du PPCM et du PGCD de 2 nombres, etc.

Puissance d'une machine de Turing

Depuis des décennies, de nombreuses réalisations concrètes ont vu le jour : machine de Turing avec un mécanisme en bois, avec des lego, etc. Ces réalisations n'ont pas toute la même puissance, tout comme les ordinateurs que nous connaissons de nos jours n'ont pas tous la même puissance. Pour une machine de Turing, il y a 3 paramètres principaux de comparaison : 1) la taille de l'alphabet : c'est le nombre de symboles différents que peut contenir chaque case. Chaque symbole peut être un chiffre, ou une lettre, ou un dessin, peu importe. Les machines binaires n'ont que 2 symboles (souvent « 0 » et « 1 »). Notre machine est à 3 ou 8 symboles, selon le numéro de programme choisi. 2) La taille du ruban : c'est tout simplement le nombre de « case » du ruban. 3) la vitesse d'exécution de l'algorithme. Celle-ci est réglable par potentiomètre sur notre machine de Turing décrite. Ensuite, les machines de Turing diffèrent par ce qui fait office de case de ruban, par la manière d'introduire un algorithme dans la machine, et par le nombre d'algorithmes qui peuvent être stockés dans sa mémoire de programme, pour

Les anciens numéros de PROGRAMMEZ!

Le magazine des développeurs

Complétez votre collection....

Tarif unitaire 6,5 € (frais postaux inclus)

TECHNOSAURES

Le magazine à remonter le temps !

Prix unitaire : **7,66 €** (frais postaux inclus)

N°4 Standard 10 €

N°4 Deluxe 15 €

☐ 234	: __ ex	☐ 240	: __ ex
☐ 235	: __ ex	☐ 241	: __ ex
☐ 236	: __ ex	☐ HS1 été 2020	: __ ex
☐ 238	: __ ex	☐ 242	: __ ex
☐ 239	: __ ex	☐ 246	: __ ex

soit __ exemplaires x 6,50 € = ____ €

Technosaures ☐ N°1 ☐ N°2 ☐ N°3 ☐ N°5 ☐ N°6

soit __ exemplaires x 7,66 € = ____ €

☐ N°4 Deluxe ..... 15 €

☐ N°4 Standard ..... 10 €

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

..... ____ € soit au **TOTAL** = ____ €

☐ M. ☐ Mme ☐ Mlle Entreprise : ____ Fonction : ____

Prénom : ____ Nom : ____

Adresse : ____

Code postal : ____ Ville : ____

Règlement par chèque à l'ordre de Programmez ! I Disponible sur www.programmez.com

Offres pouvant s'arrêter à tout moment, sans préavis PROG 247

les machines qui en sont munies. Certaines machines utilisent des cartes perforées : une carte perforée = un algorithme. L'utilisateur perce les trous aux bons endroits. En cas d'erreur de « codage », la carte perforée est à jeter !

Notre machine de Turing électronique

Dans la partie supérieure, le ruban est matérialisé par 16 colonnes. Dans chaque colonne il y a 3 LED : Orange, Verte, et Rouge. Les programmes 1 à 20 (choix du programme en bas à droite) gèrent un alphabet à 3 symboles : 1 des 3 LED de chaque colonne est allumée, et une seule LED par colonne. Les programmes 21 à 30 gèrent un alphabet à 8 symboles : dans chaque colonne, on peut allumer 0, ou 1, ou 3 LED. Il y a donc 2 puissance 3 = 8 combinaisons par colonne (entre « 000 » et « 111 »). Une petite LED bleue représente la tête de lecture / enregistrement qui se déplace de colonne en colonne.

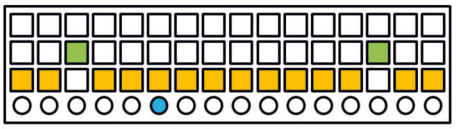
Nombre d'états : pour les programmes 1 à 10, chaque algorithme possède jusqu'à 6 états, ce qui est déjà suffisant pour faire beaucoup d'algorithmes, comme l'addition ou la soustraction de 2 nombres binaires signés par exemple. Pour les programmes 11 à 20 : 12 états maximum par algorithme. Enfin, pour les programmes 21 à 30 : 22 états maximum par algorithme. C'est le compromis qui a été choisi pour pouvoir stocker beaucoup d'algorithmes en EEPROM, tout en disposant d'une puissance intéressante pour ce genre de machine : 8 symboles et jusqu'à 22 états par algorithme. Il est en effet possible de stocker en mémoire un algorithme en appuyant sur le bouton Enter pendant 2 secondes : l'algorithme ainsi sauvegardé ne s'efface pas quand on débranche la machine.

Dans la partie inférieure, il y a les BP (= Boutons Poussoirs) et afficheurs 7 segments qui servent à saisir l'algorithme (commutateur sur « Prog »), puis à suivre son déroulement (commutateur sur « RUN »), à vitesse réglable de très lent (débogage) à très rapide, grâce au potentiomètre « Vitesse ». Les 4 BP blancs des 16 colonnes permettent d'initialiser celles-ci aux couleurs souhaitées lorsque le commutateur est sur « Prog » : à faire avant d'exécuter le programme.

Algorithme 1 = « Ping Pong »

Le programme 1 contient un algorithme d'animation lumineuse en 2 états que nous appelons « Ping Pong » et qui est codé en dur. Chaque machine est livrée avec un dossier pédagogique qui contient, entre autres, le code d'une dizaine d'algorithmes. Pour « Ping Pong », le code est le suivant :

Etat initial de l'afficheur



Programme de l'algorithme

Etat Actuel		Etat Futur		
N° d'état	Couleur	Couleur	Flèche	N° d'état
1			←	2
			←	2
			→	1
2			→	1
			→	1
			←	2

Les vidéos des algorithmes sont à la page « Algorithmes » du site www.machine-de-turing.fr, ainsi que les fichiers PDF, téléchargeables, de chaque « code d'algorithme ». Le plus stimulant est d'essayer de trouver le code par soi-même !

Voici ce qu'il faut comprendre de l'algorithme ci-dessus :

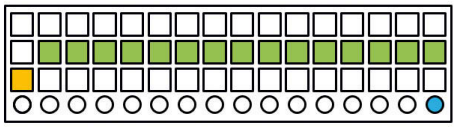
1. Etat 1 : si la couleur (de la colonne en cours de traitement) est Orange, alors : laisser la couleur sur Orange, puis déplacer la tête de lecture / enregistrement d'une case à droite, puis rester à l'état 1 : tant que la tête « voit » une colonne Orange, elle ne fait rien, à part aller à droite.
2. Etat 1 : si la couleur est Verte, alors : mettre la couleur de cette colonne sur Rouge, puis aller à gauche, puis aller à l'état 2 qui est l'état des actions effectuées par la tête de lecture / enregistrement lorsqu'elle se déplace vers la gauche.
3. Etat 1 : si la couleur est Rouge, alors : mettre la couleur de cette colonne sur Vert, puis aller à gauche, puis aller à l'état 2.
4. Etat 2 : si la couleur est Orange, alors : laisser la couleur sur Orange, puis déplacer la tête de lecture / enregistrement d'une case à gauche, puis rester à l'état 2 : tant que la tête « voit » une colonne Orange, elle ne fait rien, à part aller à gauche.
5. Etat 2 : si la couleur est Verte, alors : mettre la couleur de cette colonne sur Rouge, puis aller à droite, puis aller à l'état 1 qui est l'état des actions effectuées par la tête de lecture / enregistrement lorsqu'elle se déplace vers la droite.
6. Etat 2 : si la couleur est Rouge, alors : mettre la couleur de cette colonne sur Vert, puis aller à droite, puis aller à l'état 1.

On comprend rapidement que sans cette notion d'état, il aurait été impossible de réaliser cette animation lumineuse. En effet, un moment donné la tête de lecture / enregistrement doit poursuivre son chemin vers la droite tant que les colonnes sont Oranges, et à un autre moment, elle doit poursuivre sa route vers la gauche, alors qu'elle rencontre encore toute une série de colonnes Oranges. La notion d'état résout ce problème : à l'état 1, la tête va vers la droite lorsqu'elle lit une colonne Orange, alors qu'à l'état 2 elle file vers la gauche lorsqu'elle lit une colonne Orange.

Compteur binaire

Avec 3 états seulement, il est possible de réaliser un compteur binaire (vert = 0 et rouge =1) :

Etat initial de l'afficheur



Programme de l'algorithme

Etat Actuel		Etat Futur		
N° d'état	Couleur	Couleur	Flèche	N° d'état
1			←	1
			←	2
			←	2
2			←	2
			←	2
			←	3
3				1
				1
			←	3

En électronique, les compteurs binaires ont beaucoup d'applications : division d'une fréquence d'horloge par 2 puissance n, générateur pseudo-aléatoire, synthèse de fréquence, prescaler = pré diviseur dans les processeurs pour générer une interruption cyclique, adressage d'une liste d'adresses successives en mémoire RAM, etc. L'algorithme du compteur binaire met à profit l'aspect cyclique du « ruban ». Comme celui-ci n'a pas une taille infinie, mais seulement 16 cases, la tête de lecture / enregistrement passe de la colonne 1 à la colonne 16. Dans l'autre sens, elle passerait de la colonne 16 à la colonne 1. Cet algorithme a une particularité : il ne s'arrête jamais ! Alors qu'ici cela ne pose aucun problème, c'est tout le contraire dans le cas de l'exécution d'un programme ou d'un sous-programme informatique. C'est le cas par exemple d'une boucle infinie qui fait planter un système ! Turing avait dans son article de 1936 mis en lumière ce problème de l'arrêt des algorithmes et a démontré grâce à un résonnement par l'absurde qu'il n'existe aucun moyen de savoir à l'avance si un algorithme s'arrêtera ou non. Encore de nos jours, aucun logiciel d'analyse de programme n'existe pour déterminer si celui-ci s'arrêtera ou non, d'autant que cela dépend de la valeur des données à traiter.

Addition de 2 nombres binaires

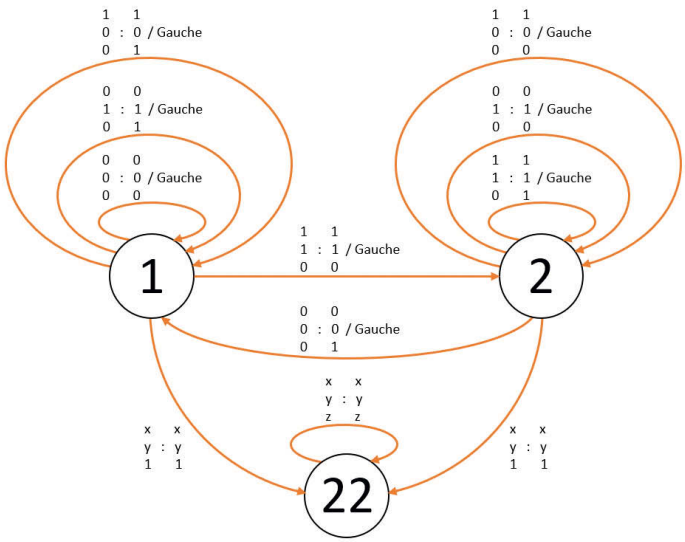
Avec une machine à 8 états, il est facile de réaliser un additionneur binaire qui opère de la même façon que nous lorsque nous effectuons une addition, en commençant par les unités, puis en additionnant les dizaines, les centaines, et ainsi de suite. Bien sûr il faut aussi calculer la valeur de la retenue et l'utiliser à la colonne de rang supérieur, ce que fait la machine de Turing si le code est correct !
Voici un exemple :

Mise en marche en basculant le commutateur sur « RUN » :

En binaire :	En décimal :
0101001001011001	21081
+ 0100011101101010	+ 18282
= 1001100111000011	= 39363

Imaginez l'émotion ressentie par celles et ceux qui ont assisté, pour la toute première fois dans le monde, à la réalisation de l'addition de deux nombres binaires par une machine et non par un être humain !

C'est impressionnant et pourtant, l'algorithme ne possède que 2 états, plus un état d'arrêt dans lequel la machine arrive lorsqu'elle lit une colonne ayant sa LED Orange allumée. Etat 1 : retenue = 0, et état 2 : retenue = 1. Il y a plusieurs façons de représenter un algorithme. Par exemple, ci-dessous, voici le graphe de fluence de l'algorithme d'addition binaire :



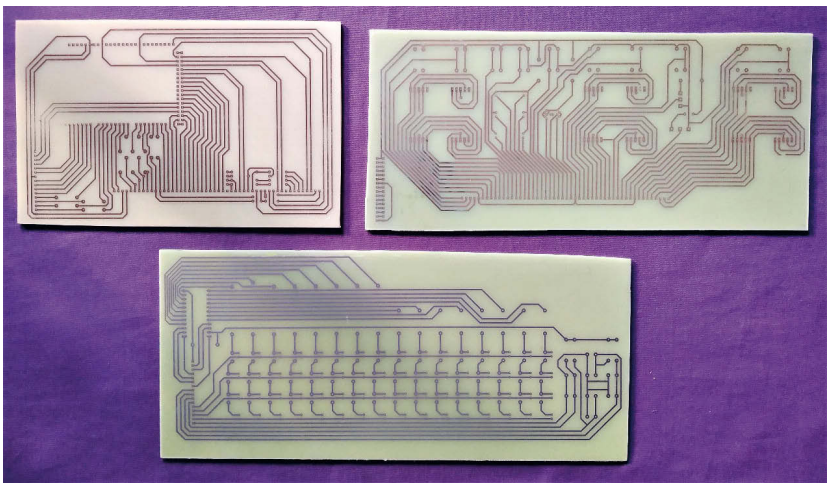


Figure 1 : *Les circuits imprimés. En haut, ceux du module de programmation. En bas, le module d'affichage.*

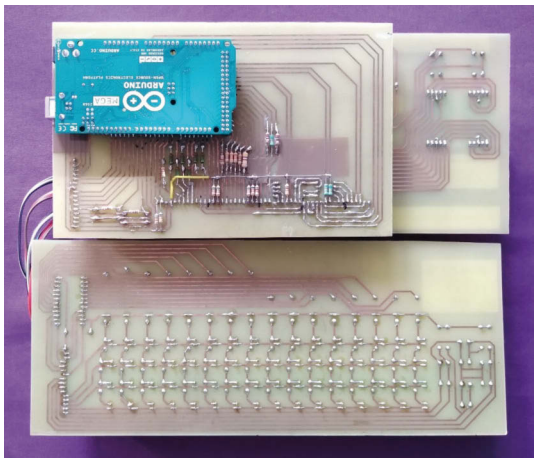


Figure 2 : *Vue côté soudures (et des résistances qui avaient été oubliées...)*

Électronique interne

Le cœur du montage est une Arduino Mega 2560, programmé en C avec l'IDE Arduino. Ce module a été choisi pour plusieurs raisons : son coût relativement faible, le nombre important d'entrées / sorties, et la présence d'une EEPROM interne de 4 ko permettant la sauvegarde des algorithmes.

Prototype puis réalisation industrielle

Nous avons réalisé un prototype avec les faibles moyens à notre disposition : routage manuel des circuits imprimés avec le logiciel TCI 4, réalisation « à l'ancienne » de ceux-ci : impression sur calques, insolation sous lampe actinique, gravure dans un banc de perchlorure de fer chauffé, perçage à la mini perceuse, soudage des composants. Il ne restait plus qu'à déboguer et améliorer l'application.

Figure 1,2 et 3

Nous avons confié le prototype et les schémas électroniques à une entreprise spécialisée pour retravailler le circuit et réaliser un schéma professionnel. Un boîtier a été créé à partir du cahier des charges. Après un premier puis un second prototype industriel, une série de 100 exemplaires a été fabriquée. Il aura fallu un peu plus de 3 ans pour sortir la machine.

De la machine de Turing à l'ordinateur

Sans vouloir retracer les nombreuses évolutions techniques entre les années 1930 à nos jours, notons tout de même qu'Alan Turing a travaillé avec de nombreux mathématiciens et précurseurs des sciences informatiques, par exemple John Von Neumann. Ils ont élaboré l'architecture de Von Neumann, qui devrait d'ailleurs être appelée architecture de Turing-Von Neumann ! En 1945, Turing a utilisé son principe algorithmique et a dressé les plans détaillés du premier ordinateur anglais : le Pilot ACE, en service dès 1950. Dans les années 1970 / 1980, beaucoup de processeurs étaient des processeurs 8 bits (Z80, 6809, 8080). Ils disposaient de registres 8 bits et adressaient des mémoires RAM et ROM qui contenaient des octets. Ces mémoires étaient la version concrète d'un ruban de Turing de longueur finie. Chaque case du ruban possédait une adresse et contenait une valeur 8 bits qui avaient donc 2 puissance 8 = 256 combinaisons différentes entre 00000000 et 11111111. C'était donc un

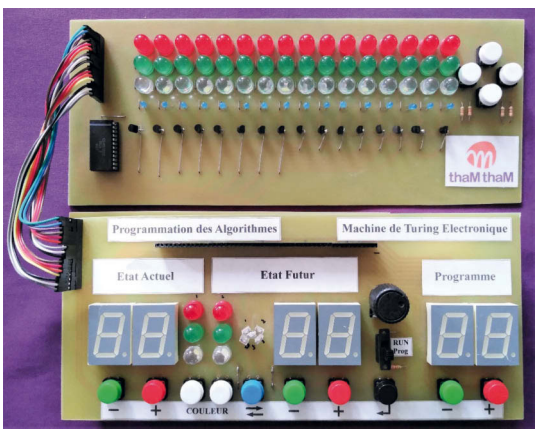


Figure 3 : *Vue de face. En bas, module de programmation. En haut, module d'affichage.*

alphabet à 256 symboles. Nos ordinateurs sont des machines de Turing universelles. Dans une machine universelle, le code de l'algorithme est situé sur le ruban lui-même, dans une zone spécialement prévue pour cela. La machine universelle démarre toujours sur la même case du ruban, celle qui contient le début d'un algorithme, qu'on appelle aussi « programme » et est conçue pour pouvoir interpréter la signification de ces cases programme du ruban. On peut aussi prévoir plusieurs zones du ruban, pour y loger plusieurs programmes, ce qui est le cas dans nos ordinateurs multitâches. **Figure 4**

Un concept obsolète ?

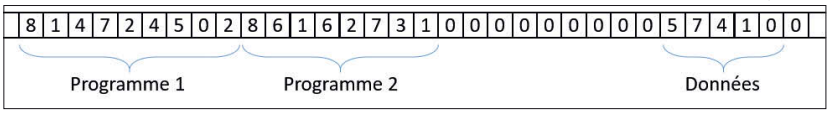
Publié en 1936, l'article d'Alan Turing pourrait sembler complètement dépassé et désuet d'intérêt de nos jours. Pourtant, il n'en est rien ! En effet, le principe dit de « machine de Turing » est encore largement utilisé de nos jours, notamment en cryptographie. Ce concept a été formalisé de façon mathématique et sert de base de recherche en sciences informatiques. Il a fait émerger une discipline de recherche appelée « théorie de la calculabilité ». Parmi les travaux de recherche récents, citons par exemple ceux du hongrois Tibor Radó sur les fonctions incalculables comme la fonction ∑(n), qu'il a introduite en 1962, qui est une fonction non calculable : n est le nombre d'états d'un algorithme, qu'on appelle aussi « classe » et ∑est l'activité opérationnelle maximale associée à ce nombre d'états.

Pour comprendre, il faut s'intéresser à une catégorie particulière d'algorithmes appelée « Castor Affairé ». Imaginez un ruban dont, initialement, toutes les cases contiennent le symbole « 0 ». Le but du jeu est de trouver le « Castor Affairé » (ou l'un d'eux s'il y en a plusieurs), c'est-à-dire l'algorithme qui, pour un nombre donné d'états, arrive à produire le maximum de cases « 1 » sur le ruban, avant de s'arrêter (oui, parce que sinon, c'est trop facile !). Le score est le nombre maximal possible de « 1 » que peut produire un algorithme, et justement, ∑ = ce score maximal. Par exemple, pour un algorithme ayant n = 3 états, on peut produire au maximum ∑ = 6 cases « 1 » en partant d'un ruban ne contenant initialement que des cases « 0 ». Voici l'algorithme en question, dont la vidéo est visible sur la page Facebook @machinedeturing ou sur Twitter à @machinedeturing :

Etat Actuel		Etat Futur		
N° d'état	Symbole	Symbole	Flèche	N° d'état
1	1	1	→	Stop
	0	1	→	2
2	1	1	→	2
	0	0	→	3
3	1	1	←	1
	0	1	←	3

Vous trouverez aussi la vidéo d'un castor affairé ayant n = 4 états : il produits 13 cases « 1 » en 107 pas. Ces 13 cases « 1 » ne sont pas consécutives. Il y a un « trou », c'est un dire une case « 0 » entre 2 cases « 1 », mais ce n'est pas grave, ça ne compte pas. Voici l'algorithme correspondant :

Figure 4



Etat Actuel		Etat Futur		
N° d'état	Symbole	Symbole	Flèche	N° d'état
1	1	1	←	2
	0	1	→	2
2	1	0	←	3
	0	1	←	1
3	1	1	←	4
	0	1	→	Stop
4	1	0	→	1
	0	1	→	4

Vous pouvez essayer de battre ces records, mais bon, sachez d'avance que toutes les combinaisons d'algorithmes ont été testées avec n = 2 à 5 états… Mais si vraiment vous aimez les challenges ou si vous voulez devenir célèbre ou du moins, rentrer dans l'histoire, vous pouvez essayer de trouver le castor affairé pour un plus grand nombre d'états. Sachez toutefois que vous devrez être patient… Car on ne connaît les castors affairés que pour n = 2 à 5, et c'est tout ! À partir de n = 6, cela se complique énormément : en 2010, quelqu'un a découvert un algorithme ayant n = 6 états et qui s'arrête après avoir produit environ 3,515 dix puissance 18267 cases « 1 ». On ne sait toujours pas si on peut battre ce record et si cet algorithme est un castor affairé !

Si vous voulez tester toutes les combinaisons d'algorithmes possibles ayant n états (comme d'autres personnes, un programme en Python qui teste toutes les combinaisons une par une), sachez que pour une machine à 2 symboles, le nombre de combinaisons est donné par la formule :

$$2^{4n} \times (n+1)^{2n}$$

Voici ce que cela donne pour n compris entre 2 et 8 :

n	Nombre de combinaisons
2	20 736
3	16 777 216
4	25 600 000 000
5	$6{,}34.10^{13}$
6	$2{,}32.10^{17}$
7	$1{,}18.10^{21}$
8	$7{,}96.10^{24}$

Pour en savoir plus : https://machine-de-turing.fr
Chaîne YouTube "Machine de Turing pédagogique" :
https://www.youtube.com/channel/UCrSK4Q1s0F85bjBw4bcR_zg

Florian Chazal
Solution Architect
Modern App

Julien Lepine
Head of Solutions
Architect Specialist
EMEA

Le B-A-BA pour démarrer en douceur dans le monde des API

En quelques années, l'utilisation d'API est devenue le standard pour le développement d'applications, tant pour leur publication et leur utilisation par des tiers qu'au sein des entreprises. Cet article propose un tour d'horizon de plusieurs types d'API, leurs avantages et contraintes, ainsi que des retours d'expériences.

La norme financière PSD2 (Payment Services Directive 2) est un exemple concret de l'essor des API. Elle structure les échanges entre les fournisseurs de services financiers, permettant une meilleure sécurité pour les paiements en ligne, et l'extension des services bancaires à des acteurs tiers. Avant la norme PSD2, il était nécessaire de passer par son établissement financier pour avoir une application de consultation des soldes, payer pour un service d'accès aux comptes et effectuer des opérations. Depuis la mise en place de cette norme, les banques ont travaillé sur un projet « open banking » afin d'ouvrir les accès à la consultation de comptes ou encore aux opérations de virement aux partenaires identifiés et autorisés.

Toute cette communication passe par les API pour réaliser les appels d'une application d'agrégation de comptes, par exemple, et obtenir les informations de la banque sur le solde du compte ou les dernières opérations effectuées. À titre d'exemple, un projet complet a été mis en œuvre pour aider les banques à développer leurs services sous forme d'API. https://www.openbankproject.com/

Une API, oui, mais de quel type ?

Bien qu'il existe plusieurs types d'API (librairies logicielles, SDK ...), cet article se concentre sur les API de type "service web", qui ont pour but premier la mise à disposition de services à des clients distants. Plus précisément seront abordées les API : REST, fournissant un modèle requête/réponse sur HTTP ; GraphQL, un langage orienté données qui fournit une abstraction des API traditionnelles ; et finalement gRPC, optimisé pour la communication entre microservices, et développée sur HTTP/2.

Chacun de ces types possède de bonnes pratiques dont certaines seront décrites ici. Pour illustrer nos propos, nous prendrons l'exemple souvent utilisé par la communauté : l'API d'un magasin d'animaux (le fameux "petStore").

Ces principes s'appliquent à tous les niveaux, et permettent à des entreprises comme Netflix d'opérer à une échelle planétaire leur plateforme, et de supporter des pics de charge très importants.

Job 0 : la sécurité

Commun à toutes les implémentations, la sécurité des points d'accès est la priorité et se base sur les mêmes principes et solutions. Une API est, comme décrit précédemment, mise à disposition sur un réseau voire directement sur Internet, et cette exposition rend primordiale la sécurisation de ces points d'accès.

Cette sécurisation passe par de l'authentification qui peut se faire communément soit par clé d'API (un secret partagé entre le client et le serveur), soit par un jeton JWT (JSON web Tokens), ou les deux conjointement. JWT est un standard d'échange sécurisé de données entre différentes applications via des objets JSON chiffrés, très utilisés pour la gestion des connexions à des services web, tels qu'OpenID. Ces données d'authentification sont injectées dans l'en-tête de la requête HTTP et vérifiées par le serveur.

Ces éléments d'authentification peuvent aussi contenir ou être liés à des politiques d'accès qui vont permettre de vérifier aussi les autorisations du client associé.

Un autre élément important qui rend cette exposition vers l'extérieur particulièrement attractive pour des personnes malveillantes c'est la nature même d'une API : elle est programmable ! Il sera donc assez simple d'écrire un code automatisant des attaques de types force brute ou déni de service. C'est pour cela qu'il sera critique d'ajouter une notion de limitation des demandes (API throttling) permettant de limiter les pressions de ce type d'attaque sur les dépendances sous-jacentes à l'API exposée.

Les plateformes cloud, de par leur élasticité, leur capacité à réagir en temps réel aux demandes des utilisateurs, et leurs fonctionnalités de sécurité, permettent d'atteindre de très hauts niveaux de sécurité. Un exemple peut être PayPlug, entreprise française spécialisée dans le paiement sur Internet, qui a pu bénéficier de l'expertise d'AWS et de sa politique de sécurité pour obtenir l'agrément auprès de l'ACPR (autorité publique en charge de l'agrément des plateformes de paiement), ainsi que la certification PCI-DSS (norme internationale pour la gestion des coordonnées bancaires).

Bonnes pratiques pour une API REST

REST est un style d'architecture d'API sur HTTP 1.1. Les lignes directrices communément adoptées ne définissent que des principes de haut niveau et non pas un protocole en soit. Cette liberté d'implémentation (et d'interprétation) des contraintes architecturales en font à la fois sa force, car elle est source de flexibilité, et sa faiblesse, car elle ne permet pas l'énonciation de lignes très strictes pour le développement de ce type d'API. De ce fait, la suite de cette section est une interprétation possible, bien que d'autres existent.

Structure

Une API REST s'appuie directement sur HTTP1.1 qui définit, entre autres, 3 éléments importants pour la conception d'une API REST: les méthodes (GET, POST, PUT, PATCH, DELETE),

l'URL (<protocol>://<nom de domaine>:<port>/<chemin> comme https://example.aws/ma-resource) et le type de média utilisé (application/json).
Comme dit précédemment il n'y a pas de règle stricte, mais voici quelques éléments communément acceptés comme étant un bon point de départ dans la conception d'une API :

- Le chemin de l'URL suit le schéma suivant : /<nom de la resource>/<identifiant d'une ressource>, ex. /animal/12345,
- La notion de version est importante au chemin en préfixant tous les chemins de celui-ci (ex. v1/animal/12345), ceci, car les API sont éternelles
- Les méthodes HTTP doivent être utilisées en accord avec HTTP1.1. (GET pour récupérer un objet, POST pour en créer, PUT pour le modifier). Par exemple, POST est parfois utilisé pour faire des modifications sur un objec existant, ou GET utilisé avec un paramètre du type "?action=terminer" pour changer son état ce qui ne sont pas de bonnes pratiques.
- Les requêtes effectuées en utilisant les méthodes GET, PUT, DELETE devraient être idempotente (POST ne l'est pas).

Plusieurs outils et spécifications, comme Swagger et Open API v3, permettent de représenter cette structure. Voici un exemple de ce que pourrait être l'API REST d'un magasin d'animaux :

```
POST /pet
  body: {
    "name": String,
    "age": Int,
    "status": Enum,
    ...
  }
GET /pet
GET /pet/{petId}
PUT /pet/{petId}
  body: {
    "name": String,
    "age": Int,
    "status": Enum,
    ...
  }
DELETE /pet/{petId}
```

Dans cet exemple, tout changement d'état d'un animal devra être effectué par un PUT sur /pet/{petId} avec un corps de requêtes contenant les détails de l'animal dans un format JSON respectant le modèle défini (non détaillé ici) et on recevra l'objet complet après modification.
Les API REST supportent de manière transparente les appels HTTP/2, car ce protocole respecte la sémantique proposée par les versions précédentes et donc la compatibilité ascendante. Certaines optimisations peuvent être prévues telles que la gestion avancée du cache, mais nécessitent des développements spécifiques sur les applications clientes et serveur pour les activer, et de ce fait sont moins déployées.

Sécurité

En plus des éléments précisés dans la partie Job 0, le respect de la structure énoncée précédemment permettra d'ajouter des règles d'accès aux ressources fines. La plupart des services d'exposition d'API vous permettront d'associer des politiques de sécurité précises liées aux méthodes HTTP utilisées et/ou le chemin associé.
En reprenant l'exemple précédent, il est possible, par exemple, de restreindre les droits d'ajout de nouveaux animaux au magasin (store) en rajoutant une politique de sécurité autorisant seulement les personnes authentifiées à pouvoir effectuer la méthode POST sur /pet et PUT sur /pet/*.
Les plateformes de gestion d'API, telles qu'Amazon API Gateway, supportent la définition de politiques complètes de sécurité pour tous vos utilisateurs, ainsi que la gestion de l'identification et de l'authentification de vos utilisateurs. De grands groupes, tels qu'Euler Hermes (spécialiste de l'assurance-crédit), utilisent des plateformes d'API et le cloud comme éléments clés de leur transformation.

Performance

Lors de requêtes de type GET pour obtenir une liste de ressources, il sera important de proposer au client la possibilité de contrôler le nombre d'éléments retournés par requêtes (en ajoutant un paramètre de requête de type "limits"), l'ordre dans lequel ils sont retournés (avec un paramètre de type "sort_by") et un pointeur pour récupérer les autres "pages" de réponse (avec le retour d'un jeton stocké dans un champs de type "nextToken").
Les principes de conception énoncés précédemment, par l'homogénéité de la structure, ce qu'elle représente ainsi que l'idempotence de certaines méthodes, vont permettre de garder en cache certaines réponses et ainsi éviter des allers-retours entre le client et le serveur.
On va pouvoir, par exemple, garder en cache pour quelques secondes toutes requêtes GET, PUT ou DELETE si elles sont effectuées avec les mêmes paramètres. Ceci permettra de soulager la partie serveur et de répondre beaucoup plus vite au client.
Il sera aussi possible de faire ce type d'optimisation côté client en introduisant un cache local et un mécanisme de rejeu avec un intervalle de temps exponentiel et aléatoire (méthodes appelées "exponential backoff" and "jitters") entre les rejeux.

API GraphQL

Un des retours principaux sur les API REST est le manque de contrôle côté client sur la quantité de données retournées par le serveur. C'est une contrainte qui était particulièrement problématique pour de grands acteurs de l'Internet. Par exemple, Netflix nécessitait, pour leur plateforme de publicité et de marketing, l'utilisation de données de plusieurs sources, afin de personnaliser le contenu au maximum pour leurs utilisateurs. Les API REST nécessitaient l'envoi de gros volumes de données à tous les clients, dont beaucoup de données inutiles, et tout ajout de données impliquait une modification des API pour tous les autres clients. GraphQL, un langage Open Source de requêtage client/serveur permettant au client de contrôler précisément le contenu reçu par le serveur permet de simplifier ces cas d'usage.
En utilisant GraphQL, les concepteurs d'API ne définissent pas des chemins associés à des verbes HTTP, comme pour

les API REST, mais un modèle de données sous une forme précise appelé, "GraphQL Schema". Ce schéma définit les types représentant les ressources gérées par l'API, les fonctions permettant les modifications de celles-ci (les "Mutation"), celles permettant de les récupérer (les "Query") et enfin celles permettant d'observer leur changement en s'abonnant (les "Subscription").
Par exemple, un équivalent de l'exemple REST précédent serait le suivant :

```
type Pet {
 id: Int!
 name: String!
 photoUrl: String!
 tags: [Tag]
 status: PetStatusEnum
}

input AddPetInput {
  id: Int
  name: String!
  photoUrl: String!
  tags: [TagInput]
  status: PetStatusEnum
}

...

type Mutation {
 AddPet(input: AddPetInput!): Boolean
}

type Query {
 listPets(): [Pet]
 GetPet(id: Int!): Pet
}

type Subscription {
  onCreatePet: Pet
  onUpdatePet: Pet
  onDeletePet: Pet
}
```

En suivant cet exemple, il est possible de récupérer exclusivement l'ensemble des noms (name) des animaux (pet) en appelant simplement :

```
query ListPets {
  listPets() {
    name
  }
}
```

Grâce à cette flexibilité, les clients (web, mobiles, tiers) des API ont la capacité de prendre la décision directement des données utiles à leur traitement, limitant ainsi les échanges. Cela offre aussi de la flexibilité aux développeurs, qui n'ont plus à prévoir tous les cas d'usage, car les clients ont plus de contrôle.
Une nouvelle fonctionnalité importante par rapport aux API REST est la notion de « Subscriptions ». Cela permet de recevoir une notification (mode push) lors d'un changement à la création, à la mise à jour ou la suppression d'un animal, réduisant grandement le nombre de requêtes faites de façon synchrone au serveur. La plupart des implémentations utilisent des canaux de type WebSocket pour effectuer ce genre d'échange.

Structure

Dans l'exemple précédent, il n'est pas possible de faire certaines requêtes pourtant réalisables avec l'API REST, telles que de filtrer les valeurs requêtées par statut ou tag par exemple. Pour pallier cela, une bonne pratique est d'ajouter un objet "filter" aux requêtes de listing :

```
input ModelStringInput {
  ne: String
  eq: String
  contains: String
  notContains: String
  between: [String]
  beginsWith: String
  attributeExists: Boolean
}

input ModelPetFilterInput {
  name: ModelStringInput
  tags: ModelStringInput
  and: [ModelPetFilterInput]
  or: [ModelPetFilterInput]
  not: ModelPetFilterInput
}

type Query {
 listPets(filter: ModelPetFilterInput): [Pet]
...
```

Par cet ajout, l'ensemble du langage de filtrage est décrit et disponible pour les clients. Il est donc possible de récupérer les noms des animaux dont le statut est `available` :

```
query ListPets {
  listPets(filter: { status: { eq: "available" } }) {
    name
  }
}
```

Ensuite, de la même façon qu'il est grandement conseillé d'implémenter un système de pagination pour une API REST, l'équivalent s'applique aux API GraphQL. Le schéma peut évoluer pour supporter deux paramètres supplémentaires : `limit` permettra au client de définir le nombre d'éléments maximum retourné par appel, et `nextToken` représentera un pointeur vers la suite des éléments à potentiellement récupérer.

```
type ModelPetConnection {
  items: [Pet]
  nextToken: String
}

type Query {
 listPets(filter: ModelPetFilterInput, limit: Int, nextToken: String): Model
```

```
PetConnection
...
```

Au lieu de retourner directement le tableau d'éléments, l'API retourne le tableau respectant la limite indiquée par le client accompagné d'un jeton permettant de récupérer la suite des données.

Sécurité

Dans le cadre de GraphQL, l'autorisation ne se fait plus directement au niveau du point d'accès, mais au niveau de la logique métier/business pour permettre une granularité maximum. Des frameworks comme AWS Amplify facilitent grandement ce type d'implémentation en l'intégrant directement au schéma graphQL avec une génération automatique de la logique métier liée à l'authentification.

Performance

Côté client, il sera important de profiter au mieux de cette faculté à contrôler les données retournées en ne demandant uniquement que les données nécessaires, limitant ainsi la taille des échanges.

Certains clients, comme AWS Amplify ou Apollo, implémentent des optimisations en étant par exemple capable de gérer un cache local ou une resynchronisation en cas de perte de réseau. Ces clients fournissent aussi par défaut un mécanisme de rejeu, ainsi que présenté dans les API REST, supportant les mécanismes de type "exponential backoff" et "jitters").

La spécification de GraphQL définit aussi la façon dont le serveur peut être construit pour répondre aux exigences du client. Il est donc important de comprendre que l'implémentation d'un serveur GraphQL peut être plus complexe qu'un serveur en charge d'API REST et que donc, même s'il existe aujourd'hui des dizaines d'implémentations open source, il est grandement conseillé d'utiliser un service managé comme Prisma Cloud ou AWS AppSync pour bénéficier de leur optimisation. Dans le cas contraire, il sera important de faire attention, contrairement à un service REST, à exposer un unique chemin pour toutes les requêtes (le plus souvent /graphql) et compresser les données échangées pour améliorer encore plus les performances de transfert comme spécifié dans la documentation officielle.

Concernant les capacités de caching, dans le cadre de GraphQL, les mécanismes vus précédemment dans les API REST ne seront pas directement applicables du fait du chemin et de la méthode HTTP unique (POST sur /graphql). Il faut donc introduire une unicité d'identifiants à travers toutes les ressources par une génération contrôlée de ceux-ci ou par l'utilisation d'une clé composite (combinaison de plusieurs champs comme l'id et le resource_type). Cela permettra au serveur de reconnaître si la requête concerne la même ressource ou non et donc, potentiellement d'envoyer une réponse venant du cache plutôt que de la base.

GraphQL est souvent utilisé pour consommer des ressources existantes, par exemple des API REST, de nombreuses optimisations, parfois complexes, peuvent être appliquées à l'exécution des requêtes sous-jacentes ("fetch") permettant de construire la réponse suivant le schéma GraphQL demandé. Pour donner un exemple concret et simple, dans une requête demandant une liste de Pet, un attribut, par exemple status, peut provenir d'une autre donnée source. Si on exécute les requêtes sans optimisation, on risque d'appeler unitairement l'API fournissant le status autant de fois que le nombre d'occurrences de Pet dans la liste. Un concept classique consiste à faire du batch de requêtes, c'est à dire, dans ce cas, à faire une seule requête demandant le status pour l'ensemble des Pets de la liste. La demande GraphQL étant générée par le client, le nombre de combinaisons pour trouver toutes ces optimisations est souvent très important. C'est la raison pour laquelle, des frameworks ont été créés pour faciliter et optimiser la consommation de ressources sous-jacentes, par exemple Dataloader, fait partie de la suite GraphQL.

Communications entre microservices

Les clients des API présentées précédemment peuvent être soit des clients finaux (interface graphique, mobile, ligne de commande, équipement connecté) soit des services. Ce type de communication « service-to-service » est très présent dans les architectures en microservices, pour lesquels des systèmes dit de « circuit-breaker », limitant l'impact d'une panne d'un service sous-jacent, sont importants.

Un système de « circuit-breaker » va se positionner entre un service A qui communique avec un service B et décider de laisser, ou pas, passer les requêtes entre ces services. Ce choix se fera en fonction du nombre de requêtes ciblant le service B réussies versus celles échouées dans un temps donné. Dans le cas d'un trop grand nombre d'échecs, le circuit-breaker bloquera les requêtes au service B et retournera directement une erreur sans même essayer de contacter ce service. Après un certain temps, le statut de la connexion sera passé en autorisation partielle afin d'évaluer le nouvel état du service B. De façon progressive le circuit breaker pourra refaire passer toutes les requêtes vers le service B.

Ce type de mécanisme, ainsi que la gestion automatique des reprises sur pannes, et du routage dynamique des services, permet d'éviter une propagation des anomalies due à des pannes de services en évitant de surcharger un service déjà dégradé et donc permettre sa potentielle stabilisation. Ceci est particulièrement utile quand les clients des API sont implémentés avec un mécanisme de rejeu en cas d'échec. Ces rejeux peuvent rapidement créer un grand nombre de requêtes après une dégradation du service appelé « thundering herd ».

Un autre élément souvent présent dans l'implémentation de communication service-to-service est l'utilisation de gRPC. gRPC est un framework basé sur HTTP/2 pour le transport et sur Protocol Buffer pour la description des interfaces. Le choix de HTTP/2 permet une communication bidirectionnelle sans avoir besoin d'intégrer un autre protocole comme WebSocket. Le choix de Protocol Buffer impose un typage fort dans la description des messages échangés. Ses performances et ses capacités d'intégrations avec la plupart des langages en font un framework de choix pour l'implémentation d'API de micro-service.

Il est très fréquent d'ailleurs de combiner une API GraphQL publiée à l'extérieur et une interaction en gRPC entre les

microservices sous-jacents. Des plateformes comme AWS App Mesh permettent de gérer la connectivité, l'analyse et les métriques entre les microservices sur plusieurs protocoles.

Testez vos API

Décrire les API sous ces types de formats (OpenAPIv3 ou GraphQL Schema) permet de valider, en amont de leur implémentation, les opérations attendues par les clients ou partenaires, leur comportement, leurs formats d'entrée/retour, les erreurs possibles, etc. Cette approche de définition par la description des API est généralement appelée "Design-first".

Cela permet donc de fournir aux développeurs des solutions de bouchon auto-généré accélérant considérablement le développement des parties clientes en optimisant le cycle de tests avec possibilité d'essai sur le poste local du développeur.

Il existe de nombreux outils permettant de créer des serveurs de tests d'API basé sur OpenAPI comme swagger CodeGen ou GraphQL Schema comme Amplify Mock ou Apollo Server.

Par contre, il est évident que les tests effectués sur ces implémentations sont efficaces pour une implémentation de la partie serveur, mais, ne remplace pas des tests d'intégrations faisant appel à l'implémentation réelle des services sous-jacents.

Partagez vos API

Une API est implémentée pour être partagée et utilisée par des développeurs donc toute exposition d'API doit au moins être accompagnée d'une documentation. Pour les API REST la distribution d'un schéma OpenAPIv3 ou Swagger est une bonne pratique. Pour GraphQL la définition et le partage même du schéma est souvent suffisants, car facile à lire et celui-ci est déjà exhaustif.

Il est souvent préférable d'exposer aussi une interface web (appelé communément un Portail Développeur) permettant de lire et tester ces API. Des implémentations comme Swagger UI pour REST, et GraphiQL pour GraphQL sont disponibles.

Même si par définition, les API sont basées sur HTTP et peuvent être utilisées grâce à une simple commande cURL, on pourra aller plus loin en proposant sur ce portail, la récupération d'un SDK. Le SDK permet aux développeurs d'éviter d'aller chercher les détails d'implémentation de l'API et de simplifier la gestion de la sécurité, du rejeu, de l'authentification.

Les portails développeur sont accompagnés de fonctionnalités permettant à un développeur ou une organisation entière d'enregistrer son application, représentant l'ensemble des API utilisables, et de générer les éléments d'accès nécessaires (clé de sécurité, client_id, ou toute autre méthode d'authentification). Dans le cadre d'une monétisation, le modèle le plus classique est le paiement "à l'usage", c'est-à-dire l'exposition d'un système de souscription permettant le paiement à l'utilisation. Généralement, ceci est accompagné de la capacité à suivre les quotas offerts, la consommation en cours ou l'historique et les limites associées.

Des solutions comme AWS Marketplace permettent aux développeurs de de mettre à disposition leurs solutions Software-as-a-Service et d'atteindre simplement tous les clients AWS.

Débridez vos API

Pour conclure, comme tout service exposé sur le web, une solution d'hébergement d'API répondant à certaines exigences, en termes de disponibilité et de qualité de service, devra être déployée de façon hautement disponible, voire globalisée. Pour ce faire, un déploiement multi-zone de disponibilité et multi-région sera peut-être nécessaire ainsi que faire appel à un réseau de diffusion de contenu (CDN). De plus, une solution de pare-feu applicatif permettant de stopper les attaques par déni de service ou autres menaces comme celles listées par l'OWASP sera hautement recommandée.

Les différents aspects listés pour le développement d'un service d'exposition d'API et l'agilité nécessaire au métier pour les mettre à jour montrent qu'il est préférable de déléguer au maximum la gestion et l'implémentation à des services gérés et hautement disponibles par exemple sur la plateforme AWS. Amazon API Gateway, AWS App Sync ou AWS App Mesh permettent d'améliorer la productivité des équipes en prenant en charge entièrement l'opération du service exposant l'API cliente.

Conclusion

Dans cet article nous avons vu plusieurs types d'API ayant chacun leurs avantages et inconvénients. Que ce soit pour une API REST ou GraphQL, il sera important d'insister sur sa sécurisation, son design, son optimisation côté client, son déploiement et son maintien en opération en gardant en tête que tout ceci peut être grandement facilité par la délégation de la plupart de ses tâches à un fournisseur de Cloud.

Resources

Adoption GraphQL à Netflix :
https://netflixtechblog.com/our-learnings-from-adopting-graphql-f099de39ae5f
PayPlug que AWS :
https://aws.amazon.com/fr/solutions/case-studies/payplug/
Euler Hermes et AWS :
https://www.youtube.com/watch?v=be_fnW13C1I

Base de données convergée Oracle : Relationnel et JSON, mais aussi Graph, Spatial, XML, Texte… PARTIE 1

Après plus de 4 décennies d'existence, la base de données Oracle continue de se renouveler. Depuis plus de 7 ans, le support des données JSON n'a fait que s'enrichir. Avec la version 19c et sa mise à disposition via des services managés, notamment Autonomous JSON Database, les développeurs ont désormais accès à la base de données convergée par excellence. Elle sait traiter de nombreux types de données (JSON, relationnelles, graph, spatial, XML, texte…) pour des charges de travail multiples (OLTP, OLAP, IoT, blockchain, streaming d'événements, machine learning…), avec des APIs simples (SODA, REST, SQL, PGQL…) et ce pour tous les langages modernes du marché. Et 2021 vient avec son lot de nouveautés…

Loïc Lefèvre

Développeur depuis l'âge de 8 ans, je multiplie mes expériences professionnelles depuis plus de 20 ans : startups, grands comptes de la finance et actuellement chez Oracle. Fullstack developer, architecte cloud et DBA applicatif, j'adore les challenges liés au développement d'applications modernes et à la gestion de l'information. Désormais Product Manager pour la base de données Oracle, je participe à l'élaboration des nouvelles fonctionnalités pour tous les développeurs : frontend, backend, low-code, data engineer, business analyst et même data scientist !

@Loic_Lefevre

Historique

Le format de données JSON (JavaScript Object Notation) est de plus en plus populaire pour développer des applications. Initialement utilisé comme format d'échange de données entre les navigateurs et les serveurs web, il a su trouver son chemin jusqu'au cœur des bases de données, y compris pour des cas d'usage de type entrepôt de données.

C'est en 2014 que la base de données Oracle a officiellement supporté ce type de données. L'API Simple Oracle Document Access (SODA), sortie en même temps, permet de s'affranchir du langage SQL pour toutes les opérations CRUD sur les collections de documents JSON et sur lesdits documents. Disponible pour les langages JavaScript/TypeScript, Python, Java... l'API SODA peut également être utilisée au travers d'APIs REST ; on pourra alors se servir des outils ou frameworks Axios, Requests, Spring, Helidon, Micronaut... pour l'intégration avec un frontend. À noter que l'on peut également invoquer des requêtes SQL au travers de ces API REST, ceci offrant de grandes similitudes avec GraphQL (exemple en fin d'article).

Néanmoins, en 2013, Oracle et IBM menant le comité de standardisation SQL/JSON, les avancées ont également été nombreuses pour aboutir au standard SQL:2016. Ainsi, pour faciliter l'adoption du format de données JSON, il a été décidé que la persistance des documents serait faite dans des champs de type chaîne de caractère (VARCHAR, CLOB ou BLOB). De plus, en l'absence de langage de requêtage standard pour les données JSON, il a été défini un langage pour naviguer à l'intérieur de ces documents en se servant de "JSON Path Expressions" ainsi que de nombreuses fonctions SQL (JSON_EXISTS, JSON_VALUE, JSON_QUERY, JSON_TABLE…) et un prédicat pour valider la structure des champs : IS JSON.

Avec la version sortie en 2017 de la base de données Oracle, le format JSON est supporté par le Column Store : une zone mémoire où les données sont représentées en colonnes et compressées pour offrir des performances singulières pour les charges de travail de type reporting, data warehouse, data mart... Un nouvel outil arrive également : le JSON Data Guide pour découvrir les métadonnées des documents JSON d'une collection (liste des champs, types...).

En 2018, l'attention s'est portée sur l'expérience utilisateur : simplification de la syntaxe des opérateurs SQL, génération de documents JSON depuis des données relationnelles, SODA pour C, C++ et PL/SQL, nouveaux types de données spatiales supportés pour les champs GeoJSON (2D, 3D, solides, surfaces...). C'est également l'année de la sortie de la nouvelle version gratuite d'Oracle XE (eXpress Edition).

L'année 2019 a été très riche puisque la version 19c devient la nouvelle version LTS (support long terme jusqu'en 2027). Avec celle-ci viennent les vues matérialisées sur documents JSON imbriqués (y compris des tableaux) avec l'opérateur JSON_TABLE, la mise à jour de documents simplifiée avec l'opérateur JSON_MERGEPATCH.

L'année dernière est né le nouveau service managé Autonomous JSON Database : une base de données Oracle tournant sur des infrastructures Exadata (partagées ou dédiées) hébergée sur Oracle Cloud Infrastructure (aussi connu sous le nom OCI). Offrant souplesse, sécurité, haute disponibilité et performance, c'est le moteur idéal pour le développement d'applications : disponible gratuitement (offre Always Free de 2 bases de données de 20 Go), moins de 2 minutes pour provisionner une base de données, maintenance automatisée, paiement à la seconde, intègre une multitude d'outils web très simples d'utilisation, mises à jour mensuelles, accessibles depuis les autres services cloud d'OCI (Kubernetes, Functions, Streaming, Analytics…). Avec ce service, c'est également la première fois que l'on voit les gains apportés par le nouveau format binaire de persistance de données JSON : le format OSON.

Enfin nous voici en 2021, et la version 21c arrive donc *très prochainement* en téléchargement : Enterprise, Standard et eXpress éditions ! Cette version contient notamment le nouveau type de données **JSON** utilisant le format binaire opti-

misé OSON. Ce nouveau type de données a été propagé à tous les outils et fonctionnalités de la base Oracle : driver JDBC, Transactional Event Queues, SQL loader, GoldenGate, XStream API, PL/SQL, Application Express (APEX), SQL Developer, SQLcl... Enfin, un nouveau type d'index a également vu le jour : le Multi-Value Index permettant l'indexation de champs JSON contenus dans des tableaux ou des tableaux de scalaires (nombres, booléens, chaînes de caractères, dates...) et le nouvel opérateur JSON_TRANSFORM permet quant à lui de réaliser le fameux "partial update" très recherché pour ses performances de mise à jour.

Pourquoi JSON dans les bases de données ?

Le stockage de données JSON a été rendu populaire par les bases de données NoSQL telles que MongoDB. Aujourd'hui, la plupart des bases de données relationnelles prennent en charge le format JSON pour les raisons suivantes.

Flexibilité du schéma

Les modifications apportées lors du développement d'une application peuvent être facilement implémentées avec le format JSON. Prenons le document JSON suivant à titre d'exemple représentant une personne :

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": { "street": "21 2nd Street",
               "city":"New York",
               "state":"NY",
               "postalCode": "10021",
               "isBusiness":false },
  "phoneNumbers": [
    { "type": "home",
      "number":"212 555-1234" },
    { "type": "mobile",
      "number": "646 555-4567" } ],
  "lastUpdated": "2019-05-13T13:03:35+0000"
}
```

On peut y ajouter le champ "middleName" sans aucun impact pour la base de données. Dans une base de données strictement relationnelle, une nouvelle colonne aurait été ajoutée via une commande ALTER TABLE ADD… opération très facile à réaliser, mais souvent un tel changement nécessite la coordination avec un DBA.
D'autre part, le format JSON permet aussi de modifier la cardinalité. Ainsi une personne peut avoir plusieurs adresses (bureau, résidence secondaire…) et le champ "address" peut alors être modifié en tableau et contenir autant d'adresses que nécessaire. Avec une base de données relationnelle, une nouvelle table aurait été créée, les opérations de mises à jour modifiées…
Le format JSON permet ainsi de dissocier les modifications apportées à l'application de celles apportées au schéma de la base de données ; intérêt d'autant plus grand en début de projet ou lors de phases de maquettage.

Absence de normalisation

Les développeurs ont tendance à penser plutôt en terme d'objets et de hiérarchies : clients, contrats, commandes… La représentation de ces hiérarchies est native au format JSON. Le modèle relationnel quant à lui requiert souvent plusieurs tables normalisées (3-NF…) ce qui se traduit par des commandes SQL que les (plus jeunes) développeurs considèrent souvent comme complexes à cause des jointures par exemple.
Les ORM (Object-Relational Mapping) tels que Hibernate peuvent simplifier ce mapping Objets / Tables, mais ils rajoutent un niveau de complexité supplémentaire et la flexibilité n'est pas toujours au rendez-vous : permettent-ils d'exécuter une commande SQL native ?
Enfin, le format JSON peut également améliorer la performance lorsque la hiérarchie d'objets JSON évite de *très nombreuses* jointures.

API simplifiée

La simplicité d'utilisation des API attire beaucoup de développeurs : exécuter des opérations CRUD directement depuis le langage de programmation au lieu de construire des requêtes SQL via de multiples concaténations… Ces API provenant du monde NoSQL ne supportaient pas initialement le langage SQL (ceci a beaucoup évolué depuis).
Néanmoins, cette souplesse, attrayante au premier abord, ne vient pas sans un certain nombre de restrictions qui peuvent apparaître en cours de développement.

Schéma de données dispersé dans le code applicatif

La flexibilité de schéma du format JSON ne signifie pas qu'il n'y a pas de schéma, mais plutôt que le schéma est implicitement distribué dans le code de l'application. Les modifications apportées au schéma ne sont plus "contrôlées" de manière centralisée par la base de données ; au lieu de cela, le code de l'application peut devoir être adapté, ce qui n'évite pas la maintenance, mais la reporte seulement.
C'est ainsi que les fameux ODM (Object-Document Mappers) sont nés afin de permettre la création d'objets en se référant à un schéma fortement typé. On pourra citer en exemple Mongoose (JavaScript) ou encore Morphia (Java). Il est à noter qu'à l'heure actuelle, les schémas de documents JSON ne sont pas encore standardisés, mais cela arrive… (rendez-vous sur https://json-schema.org pour les curieux).
La conséquence néanmoins de cette dispersion implique un travail potentiellement conséquent le jour où l'application change de langage, de framework ou de base de données !

Absence de références

Le format JSON ne permet pas de renforcer l'intégrité des données comme dans une base de données relationnelle, car il n'y a pas de contraintes de clé primaire / étrangère.
Ceci peut être contourné, par exemple en utilisant des colonnes virtuelles avec la base de données Oracle. Dans une base de données NoSQL, vous n'avez pas cette solution et vous devez essayer de garantir l'intégrité des données dans le code de l'application.

Absence de fonctionnalités basiques pour traiter les données JSON

Enfin, les bases de données NoSQL ont souvent des restrictions supplémentaires : les transactions, les jointures, la sécurité, la gestion des ressources…, et c'est encore une fois à l'application de venir compenser ces limitations.

Combinaison de données JSON et relationnelles

L'une des limitations les plus importantes reste la complexité inhérente à combiner plusieurs modèles de données entre eux. Ainsi, même si le format de données JSON peut se suffire à lui seul, aucune entreprise ne peut se passer de données relationnelles ne serait-ce qu'à cause des outils nécessaires à son fonctionnement : ERP, entrepôts de données, comptabilité…

La capacité à pouvoir réaliser ceci efficacement : sans réinventer la roue (réimplémenter les jointures en JavaScript dans le navigateur, dans le backend ou dans des Fonctions…), sans dupliquer les données vers des systèmes différents, de manière performante et sécurisée et sans faire de concession sur l'intégrité de ces données parfois critiques pour l'entreprise est un véritable challenge.

L'approche d'Oracle n'est pas d'opposer les données JSON comme une alternative aux données relationnelles, mais de permettre de combiner les avantages des deux modèles en fonction des besoins. Plus précisément, cela signifie :

- que les données JSON sont stockées dans des collections qui sont au final des tables normales, mais que l'on pourra gérer très simplement à l'aide de l'API SODA sans requérir la connaissance du SQL,
- que les colonnes relationnelles et les colonnes JSON peuvent être combinées comme vous le souhaitez,
- que les données JSON peuvent être converties en un modèle relationnel,
- que les données relationnelles peuvent être converties en données JSON.

Démarrez avec une base de données autonome

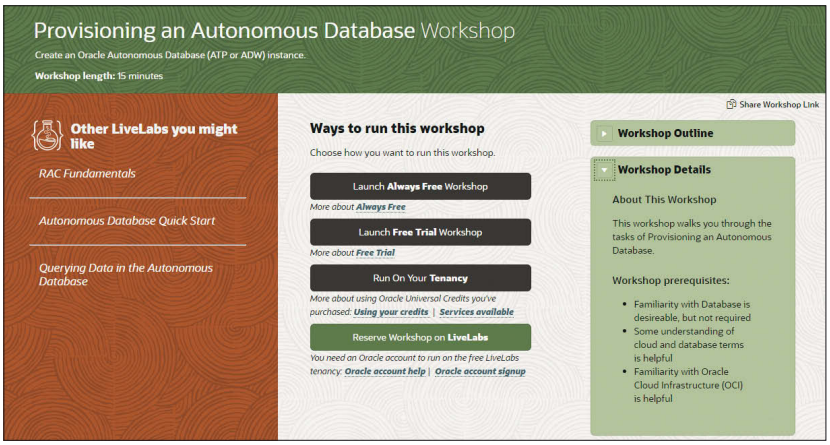
Avant toute chose, il vous faudra une base de données. Plusieurs options sont disponibles :

- Base de données gratuite Oracle 18c XE (eXpress Edition) en attendant la sortie de la version 21c XE (cette année),
- Base de données Oracle 19c SE ou EE, si possible avec la release update 19.11 (avril 2021),
- Base de données gratuite Autonomous Transaction Processing (ATP) ou Autonomous JSON Database (AJD) ; le choix entre les 2 dépendra de votre volonté à mettre la base à niveau vers la version payante le cas échéant : plus de stockage et de puissance CPU avec AJD pour un quart du prix d'ATP.

La dernière possibilité est de loin la plus intéressante, car Oracle Cloud Infrastructure via son offre "Always Free Cloud Services" fournit un ensemble de ressources cloud gratuites sans expiration :

- 2 bases de données Autonomous avec 20 Go et toutes les fonctionnalités,
- 1 base de données NoSQL avec 3 tables et 25 Go de stockage par table,

Figure 1



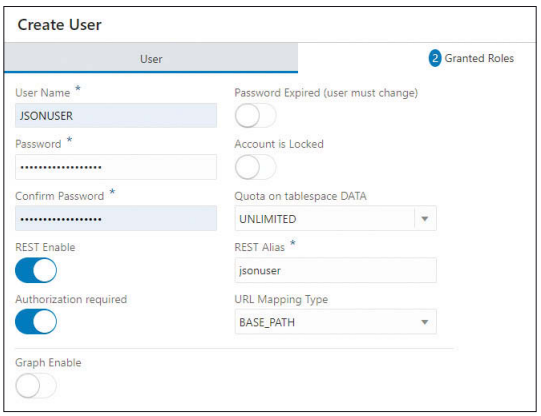


Figure 4

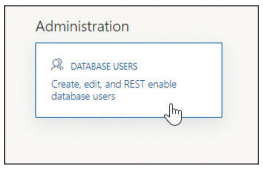


Figure 3

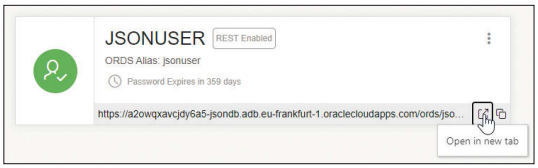


Figure 5

- 2 machines virtuelles avec 1 Go de RAM, 2 volumes de 100 Go de stockage block, 10 Go de stockage objet et 10 Go de stockage de type archive,
- 1 load balancer

Ces ressources pourront être déployées par exemple dans la région de Francfort en Allemagne en attendant l'ouverture de la région française cette année. Pour plus de détails : https://www.oracle.com/cloud/free

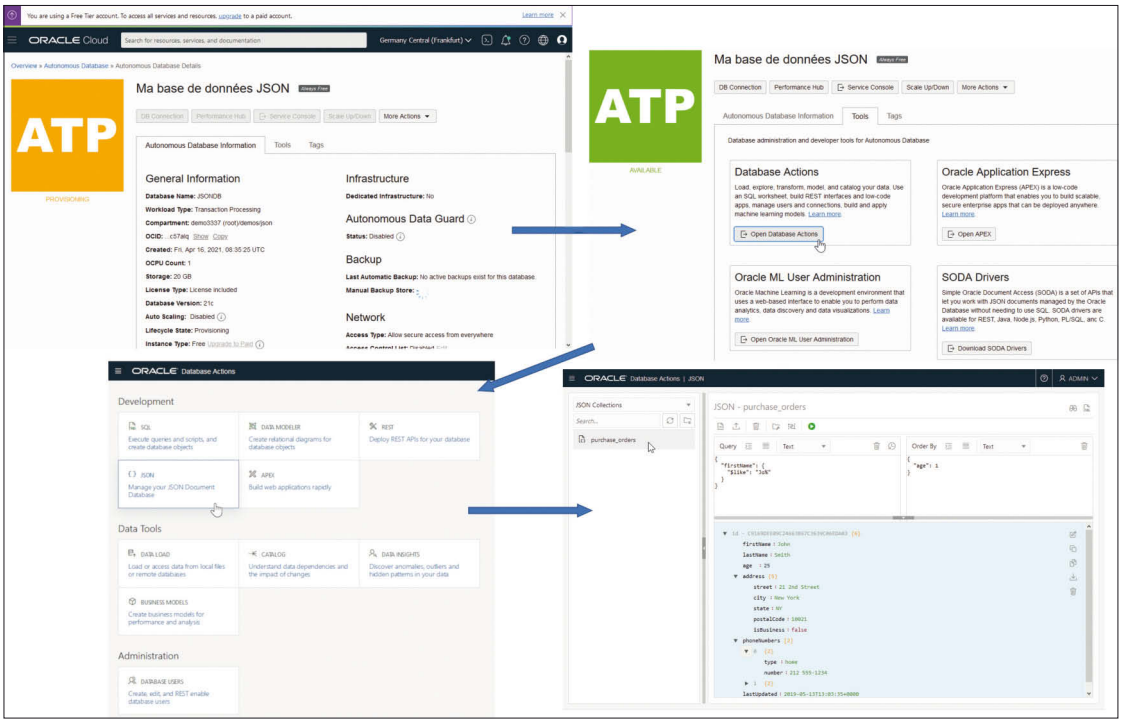
Pour les développeurs qui préfèrent travailler sur leur ordinateur, la version XE (qui ne contient pas encore toutes les fonctionnalités présentées ci-dessous) est disponible à l'adresse suivante : https://www.oracle.com/fr/database/technologies/application-development.html

Pour provisionner votre première base de données, vous pouvez consulter ce tutoriel en ligne http://bit.ly/launchADB ; en sélectionnant bien comme type : Autonomous Transaction Processing. **Figure 1**

Bien sûr d'autres tutoriels sont disponibles sur ce portail pour vous faire découvrir de nombreux autres services clouds et technologies. Une fois toutes les étapes réalisées, vous pourrez vous connecter via votre navigateur : **Figure 2**

Première opération à effectuer : se créer un utilisateur et un schéma avec des droits limités pour tout de suite s'habituer aux bonnes pratiques de développement en matière de sécu-

Figure 2



Collections	Tables
JSON Document	Row
Fields (metadata in *each* document)	Columns (common metadata for all rows)
SODA API	SQL
Standard* JSON data types (number, string, boolean, null) * Oracle JSON binary format extends the list	All data types (date, timestamp, intervals, spatial, graph…)
Index	Index
Embedding	Joins
Arrays	Child tables
Relationships: 0-n, 1-n	Relationships: 0-n, 1-n and n-m
Schema-on-Read	*Schema-on-Write*

Figure 6

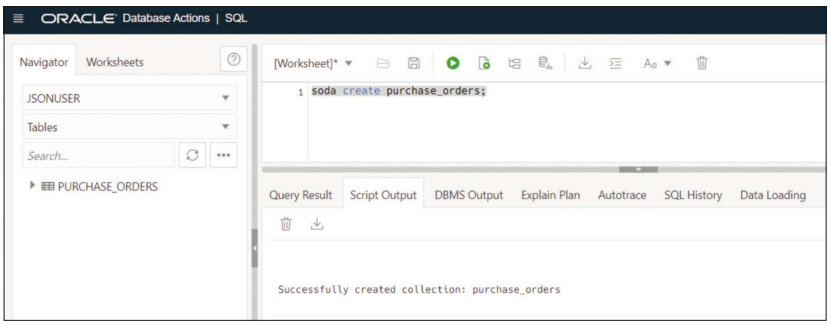


Figure 7

rité. **Figure 3**. Rendez-vous dans le panneau de gestion des utilisateurs et cliquez sur le bouton "Create User". Ensuite renseignez les principaux champs sans oublier d'activer le radio bouton "REST enable". Ceci permettra de se connecter via SQL Developer web et vous aurez la possibilité de créer des services REST sur vos données en quelques clics. **Figure 4**
Pour les rôles à donner, il faudra à minima cocher CONNECT, RESOURCE (contenant le rôle SODA_APP, ne pas l'oublier pour la version XE) et DWROLE (réservé aux bases de données autonomes). Cliquez sur le bouton "Create User".
Cliquez maintenant sur le bouton pour ouvrir un nouvel onglet et se connecter directement. **Figure 5**
La base de données autonome devrait être créée en moins de 2 minutes. Bien sûr, il existe d'autres manières pour le provisionnement :

- Script Terraform,
- API REST ciblant la plateforme cloud directement,
- Des SDK pour de très nombreux langages : Python, JavaScript, Java, .Net, Go, Ruby et PL/SQL,
- OCI cli, un outil en ligne de commande

Simple Oracle Document Access API

Composant essentiel du développement applicatif, l'API SODA est disponible pour tous les langages modernes. Nous allons au travers de quelques exemples découvrir celle-ci.
Mais tout d'abord un tableau comparatif des terminologies JSON à gauche et relationnelles à droite : **Figure 6**
Comme nous allons le voir dans les exemples qui vont suivre, l'API SODA vous permet de manipuler collections et documents sans connaître le langage SQL.

Outillage

Le nouvel outil SQLcl (successeur de sqlplus) permet de lancer des commandes SODA très simplement dans un terminal. Ces fonctionnalités sont également disponibles via SQL Developer web, panneau SQL : **Figure 7**
Les commandes soda permettent de réaliser toutes les opérations sur les collections :

- Création d'une collection (soda create),
- Lister les collections existantes (soda list),
- Supprimer une collection (soda drop)

Elles permettent également de manipuler les documents JSON :

- Insertion,

- Lecture tous les documents,
- Lecture d'un document par son ID (identifiant unique),
- Lecture de documents filtrés par une "Query By Example" (QBE) : on indique via un document JSON les valeurs des champs recherchés (de nombreux opérateurs existent) **Figure 8**

À noter que l'onglet "Data Loading" vous permettra de charger des données dans votre base de données.
Mais le plus intéressant reste à venir : gérer vos données JSON avec votre langage préféré.

Wallet

Pour pouvoir se connecter à votre base de données autonome, il faudra télécharger le wallet de celle-ci. Un wallet, est un fichier zip contenant notamment le certificat pour pouvoir établir une connexion sécurisée entre votre application et la base de données. Pour le récupérer, deux possibilités, vous pouvez :

- Le télécharger depuis le portail de votre base de données (le mot de passe est celui utilisé lors de la phase de provisionnement) : **Figure 9 et 10**
- Le récupérer via l'outil en ligne de commande OCI CLI (Oracle cloud Infrastructure Command Line Interpreter depuis le cloud shell) : **Figure 11**
  - il faudra ensuite le dézipper,
  - et modifier le chemin d'accès du fichier sqlnet.ora pour utiliser la variable d'environnement TNS_ADMIN que l'on positionnera au répertoire où le contenu du fichier zip a été extrait (mon répertoire home ici sous Linux) :

```
$ oci db autonomous-database generate-wallet \
--autonomous-database-id <ocid1.autonomousdatabase....> \
--file wallet.zip --password <votre mot de passe>
$ unzip wallet.zip
$ pwd
$ export TNS_ADMIN=/home/loic_lefev
$ sed -i 's/?\/network\/admin/$TNS_ADMIN/' sqlnet.ora
```

Figure 12

Ceci fait, vous pouvez même utiliser SQLcl, car il est déjà installé dans le cloud shell. La commande suivante va vous connecter à votre base de données en vous servant du service name TP (pour Transaction Processing): sql jsonuser/<votre mot de passe>@jsondb_tp
Un service name est un point d'accès à votre base de données qui vous alloue notamment des ressources une fois connecté. J'aborde les différents service names disponibles dans la section "Base de données Autonome".

Drivers et client Oracle

Afin de pouvoir se connecter à une base de données Autonome, il va falloir récupérer le driver SODA pour votre langage. Pour Java, il peut être téléchargé depuis GitHub (https://git.io/JOz5q) ou plus simplement via Maven :

```
<dependency>
  <groupId>com.oracle.database.soda</groupId>
  <artifactId>orajsoda</artifactId>
  <version>1.1.7.1</version>
</dependency>
```

Si vous êtes développeur Python ou JavaScript, il faudra prendre connaissance des étapes ci-dessous avant de pouvoir

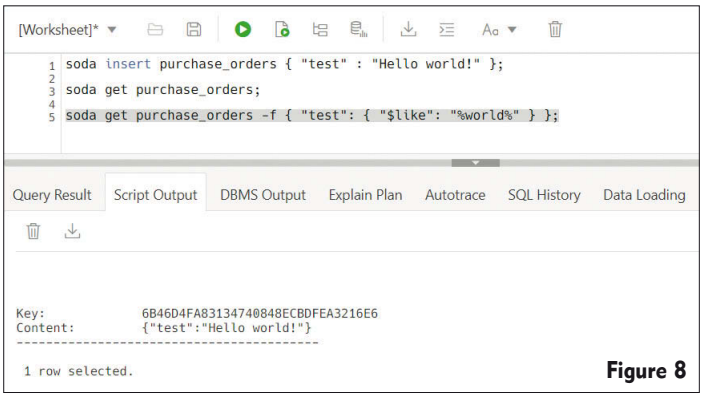


Figure 8

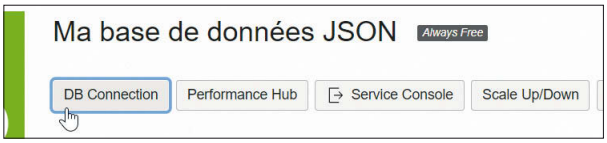


Figure 9

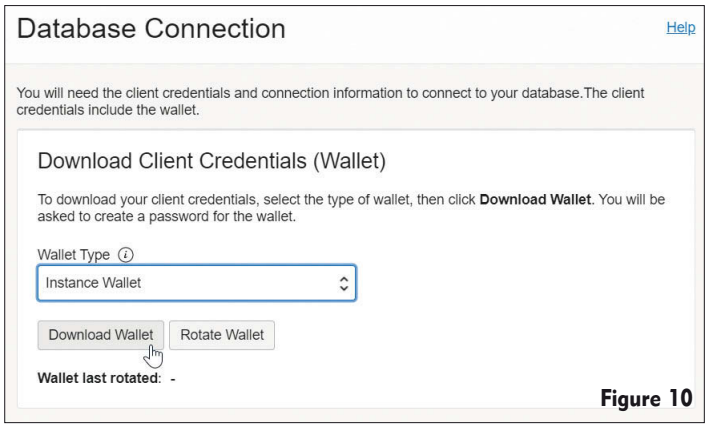


Figure 10

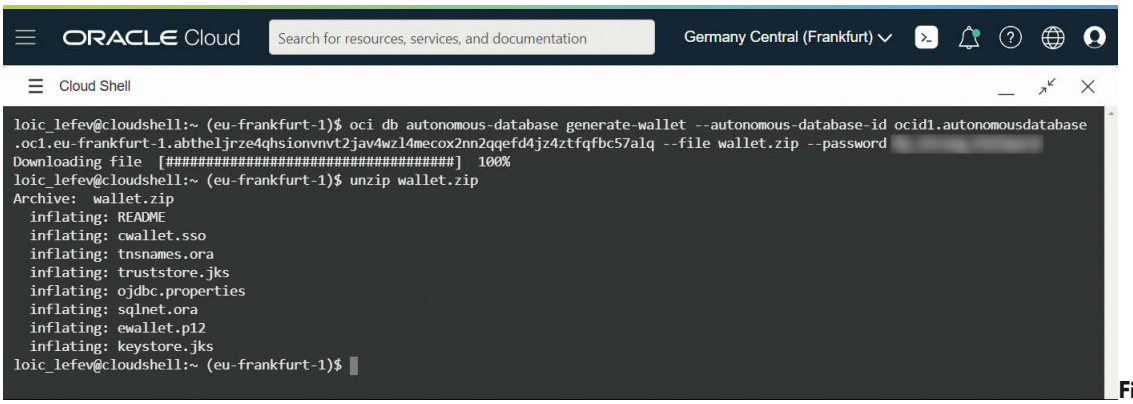


Figure 12

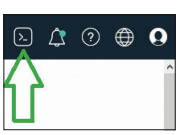


Figure 11

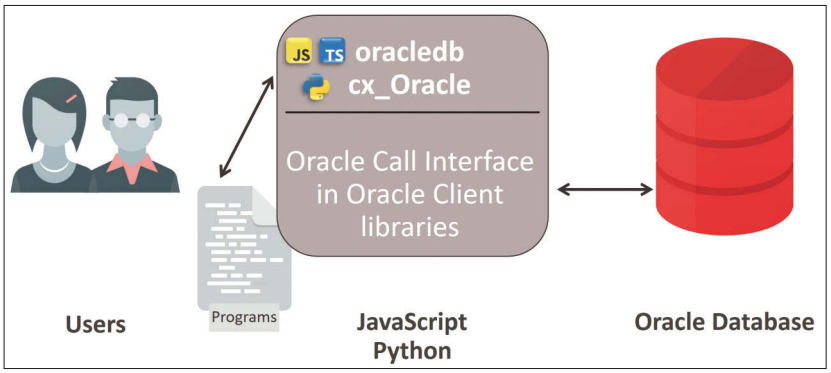


Figure 13

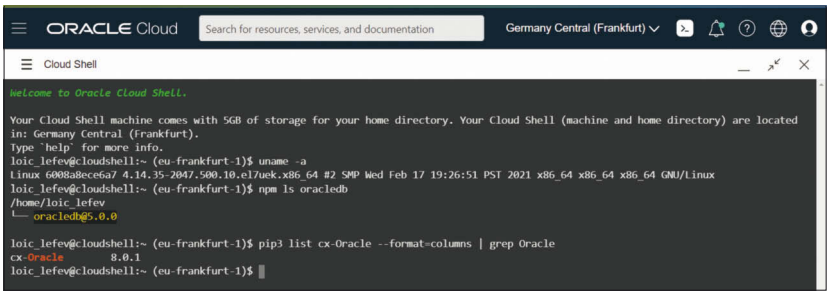


Figure 14

installer les packages ou modules. En effet, ces langages nécessitent qu'un client Oracle soit installé :

- Pour une base de données XE, vous pourrez suivre les guides de démarrage :
  - JavaScript/TypeScript :
  https://oracle.github.io/node-oracledb/INSTALL.html#quickstart
  - Python :
  https://cx-oracle.readthedocs.io/en/latest/user_guide/installation.html

Figure 13

- Pour un compte Always Free Tier, vous pourrez utiliser le Cloud Shell qui possède déjà toutes les librairies pré-installées pour une version 19c de la base de données autonome :

Figure 14

- Dans le cas où de nouvelles versions seraient disponibles, vous pourrez les obtenir comme suit :
  - Client Oracle 21.1 pour une base de données 21c :

```
$ wget https://download.oracle.com/otn_software/linux/instantclient
/211000/instantclient-basic-linux.x64-21.1.0.0.0.zip
$ unzip instantclient-basic-linux.x64-21.1.0.0.0.zip
$ export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/home/loic_lefev/
instantclient_21_1
```

  - JavaScript/TypeScript/Node.js : vous pourrez utiliser npm (pour installer le package oracledb 5.1.0 par exemple) : npm install oracledb
  - Python, il vous faudra utiliser pip3 pour installer le module cx_Oracle : pip3 install --user cx_Oracle

Dernière remarque concernant l'environnement pour le langage JavaScript : différentes versions de Node.js sont disponibles. Il faudra utiliser l'outil nvm pour choisir la version désirée: v14.16.1 (LTS), v15.14.0…

Quelques exemples de lignes de commandes :

- Liste les versions LTS disponibles uniquement :
  nvm ls-remote --lts
- Installe la version 15.14.0 en mettant à jour le gestionnaire de packages npm : nvm install 15.14.0 --latest-npm

Vous êtes maintenant prêt à développer en JavaScript, Python ou Java… Les programmes en exemple sont identiques, ils vont consister à :

1. créer une collection nommée purchase_orders (à noter que l'exemple pour le langage JavaScript utilise le champs metadata pour indiquer le type de la colonne stockant le document),
2. créer un index sur le champs requestor,
3. insérer un document JSON et récupérer la clé unique générée,
4. récupérer ce document JSON et l'afficher

SODA pour JavaScript

```
'use strict';
const oracledb = require('oracledb');
oracledb.autoCommit = true; // OK for simple operations

async function run() {
  let connection, collection;

  try {
    connection = await oracledb.getConnection({
      user: process.env.ORACLEDB_USER || "USERNAME",
      password: process.env.ORACLEDB_PASSWORD || "PASSWORD",
      connectString: process.env.ORACLEDB_CONNECTION_STRING ||
"CONNECTION_STRING"
    });

    console.log("Oracle client version: ", oracledb.oracleClientVersion);
    console.log("Oracle database version: ", connection.oracleServerVersion);

    const soda = connection.getSodaDatabase(); // Create the parent
object for SODA

    // sqlType: "BLOB" for 19c, "JSON" for 21c
    const metadata = { keyColumn: { name: "ID" },
                contentColumn: { name: "JSON_DOCUMENT", sqlType: "JSON" },
                versionColumn: { name: "VERSION", method: "UUID" },
                lastModifiedColumn: { name: "LAST_MODIFIED" },
                creationTimeColumn: { name: "CREATED_ON" }
              };

    collection = await soda.createCollection("purchase_orders", metadata);

    const indexSpec = { name: "REQUESTOR_IDX",
                fields: [{
                  path: "requestor",
                  datatype: "string",
                  order: "asc"
                }]
              };

    try { await collection.createIndex(indexSpec); } catch (err) { console.
error(err); }

    // insert a document
    let doc = await collection.insertOneAndGet({ requestor: "Loïc Lefèvre",
                                address: { city: "Tours" } });
    console.log("Document key/Id is: ", doc.key);
```

```
    doc = await collection.find().key(doc.key).getOne();          // a SODA
 document
    console.log("JSON document: ", doc.getContent());             // a JSON
 document
    console.log("JSON document as string: ", doc.getContentAsString());
 // a string
  }
  catch (err) { console.error(err); }
  finally { if (connection) { try { await connection.close(); }
                    catch (err) { console.error(err); } } }
}

run();
```

SODA pour Python

```
import os
import cx_Oracle

def connect():
  connection = cx_Oracle.connect(os.environ.get("ORACLEDB_USER"),
                    os.environ.get("ORACLEDB_PASSWORD"),
                    os.environ.get("ORACLEDB_CONNECTION_STRING"))
  connection.autocommit = True # OK for simple operations
  print( "Oracle client version: ", cx_Oracle.clientversion()[0] )
  print( "Oracle database version: ", connection.version )
  return connection

def run(soda):
  collection = soda.createCollection("purchase_orders")

  index_spec = { "name": "REQUESTOR_IDX",
          "fields": [{"path": "requestor",
                "datatype": "string",
                "order": "asc"
               }] }
  collection.createIndex(index_spec)

  doc = collection.insertOneAndGet({ "requestor": "Loïc Lefèvre", "address":
{ "city": "Tours" } })
  print( "Document key/Id is: ", doc.key )

  doc = collection.find().key(doc.key).getOne()          # a SODA document
  print( "JSON document: ", doc.getContent() )           # a JSON document
  print( "JSON document as string: ", doc.getContentAsString()) # a string

if __name__ == '__main__':
  run(connect().getSodaDatabase())
```

SODA pour Java (avec Text Block : JDK 15+)

```
import oracle.soda.OracleCollection;
import oracle.soda.OracleDatabase;
import oracle.soda.OracleDocument;
import oracle.soda.rdbms.OracleRDBMSClient;

import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Properties;
```

```
public class App1 {
  public static void main(String[] args) throws Throwable {
    try (Connection connection = DriverManager.getConnection(
        String.format("jdbc:oracle:thin:@%sTNS_ADMIN=/home/loic_
lefev",
            System.getenv("ORACLEDB_CONNECTION_STRING")),
        System.getenv("ORACLEDB_USER"),
        System.getenv("ORACLEDB_PASSWORD"))) {
      connection.setAutoCommit(false);

      Properties props = new Properties();
      props.put("oracle.soda.sharedMetadataCache", "true");

      OracleDatabase soda = new OracleRDBMSClient(props).getData
base(connection);
      OracleCollection collection = soda.openCollection("purchase
_orders");

      String indexSpec = """
          { "name": "REQUESTOR_IDX",
            "fields": [{"path": "requestor",
                "datatype": "string",
                "order": "asc" }] }""";
      collection.admin().createIndex(soda.createDocumentFromString
(indexSpec));

      OracleDocument doc = collection.insertAndGet(soda.createDocument
FromString("""
          { "requestor": "Loïc Lefèvre", "address": { "city": "Tours" } }"""));
      System.out.printf("Document key/Id is: %s", doc.getKey())
.println();

      doc = collection.find().key(doc.getLastModified()).getOne();
 // a SODA document
      System.out.printf("JSON document as string: %s", doc.getContent
AsString()).println(); // a String
    }
  }
}
```

Les exemples auront besoin que différentes variables d'environnement soient configurées :

```
#!/bin/sh
export TNS_ADMIN=/home/loic_lefev
export ORACLEDB_USER=jsonuser
export ORACLEDB_PASSWORD=<mon mot de passe>
export ORACLEDB_CONNECTION_STRING=jsondb_tp
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/home/loic_lefev/
instantclient_21_1
```

D'autres exemples sont disponibles sur GitHub :

- JavaScript : https://git.io/JOwMC
- Python : https://git.io/JOwMO
- Java : https://git.io/JOwM9 et https://git.io/J3JmK (JSON-B…)

Base de données Autonome

Nous venons de voir quelques exemples pour accéder à une base de données à l'aide de l'API SODA pour 3 langages dif-

férents. J'ai pour ma part utilisé une base de données autonome sans pour autant expliquer de quoi il s'agissait. Il est maintenant temps de corriger cette lacune. **Figure 15**

La base de données Oracle fait partie de ces logiciels qui défient le temps et qui au fil des décennies s'adaptent aux dernières tendances dans un but précis : protéger vos données. Avec ses plus de 40 années d'existence, elle a évolué :

- pour faire face à toutes les situations, voir les meilleures pratiques :
  - Maximum Availability Architecture (MAA),
  - Maximum Security Architecture (MSA)
- pour gérer des cas d'usages extrêmes (recherche de nouvelles particules, gouvernement, smart-cities, banques, réseaux sociaux, services SaaS…),
- pour intégrer les dernières avancées en matière de hardware (Persistent Memory ou PMEM/NVRAM, RDMA over Converged Ethernet ou RoCE…),
- pour s'adapter aux différents types de données (JSON, Graph, Spatial, Avro, Parquet, ORC, XML, Object, externes sur stockage objet ou HDFS…),
- pour s'adapter aux dernières pratiques de développement (microservices, transactional queues, blockchain tables, machine learning models, sharding, indexation full-text…).

Mais la dernière avancée, la plus marquante, est le niveau d'autonomie délivré au travers de services cloud managés. Autonomous Database est une nouvelle classe de services cloud inégalée : **l'automatisation poussée à son paroxysme**. En mars 2021, une nouvelle suite d'outils très simples d'utilisation (depuis votre navigateur) a été mise à disposition pour la plus grande joie des *développeurs* ! **Figure 16**

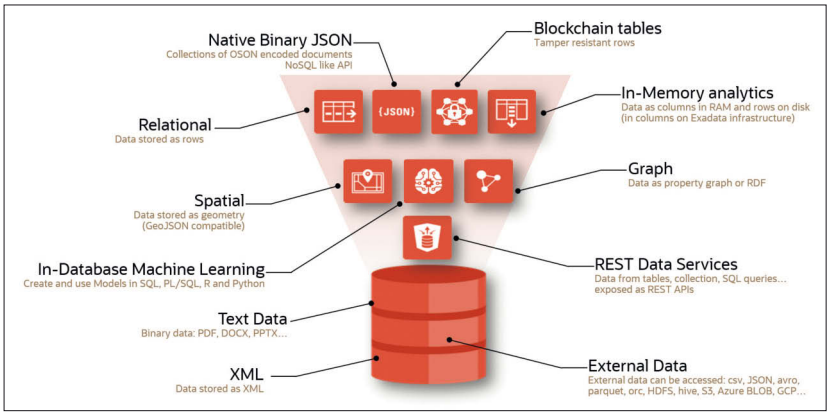


Figure 15

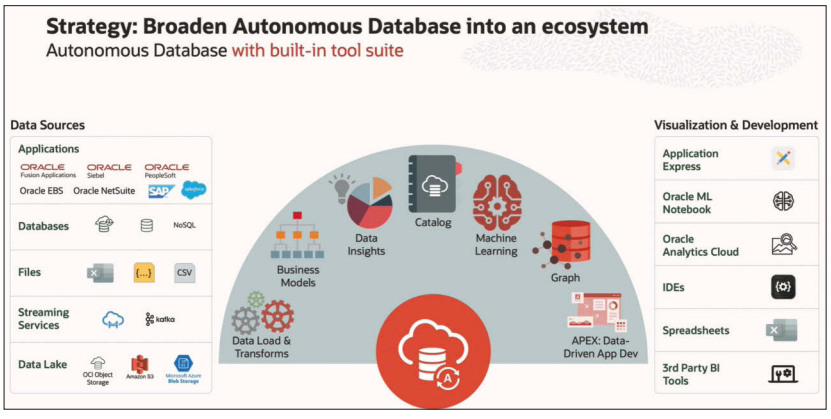


Figure 16

Comme pour un service cloud de niveau SaaS, complètement natif et automatisé, les bénéfices sont très tangibles et on s'y habitue très vite (croyez-moi) :

- vous n'avez plus à patcher la base de données,
- les données sont sécurisées (sur disque, sur le réseau),
- les backups sont automatiques (rétention de 60 jours inclus dans le coût, restauration à la seconde près),
- les ressources CPU s'adaptent à chaud et en temps réel aux charges de travail,
- le prix dépend des ressources CPU et disques consommés,
- éteinte, vous ne payez plus les CPU, uniquement le stockage,
- la haute disponibilité est gérée (failover, disaster recovery, reprise des transactions après bascule : moyennant un peu de configuration),
- le tuning est automatisé : index automatiquement créés (désactivé par défaut), compression colonne automatique pour les entrepôts de données (ADW), parallélisme automatique des requêtes SQL (selon le service name utilisé), vues matérialisées créées et maintenues automatiquement, priorisation automatique des ressources I/O…
- l'infrastructure sous-jacente est Exadata, ce qui se fait de mieux pour les bases de données Oracle : protection des données contre les corruptions avec réparation automatique, optimisations additionnelles pour tous les types de charges : index de stockage, exécution des fonctions SQL dans le stockage… afin de décharger les CPU au niveau base de données…,
- de nombreux outils sont mis à disposition au travers de votre navigateur : requêtage SQL, modélisation, dashboard pour données JSON, gestionnaire de comptes utilisateur, création de services REST, chargement de données, catalogues de métadonnées, modèles métiers, dashboard Application Express (pour développer des frontends), notebooks pour créer des modèles (Python, R, SQL),
- deux modes disponibles : infrastructure partagée et dédiée (plus souple sur la gestion des patchs…),
- enfin les capacités à traiter de multiples modèles de données évitent la duplication des données, la duplication des technologies…

Ce service cloud est fourni par Oracle Cloud Infrastructure (OCI) dans de nombreuses régions (ensemble de centre de données, exemple : Francfort, Amsterdam, Londres… et bientôt la France, l'Italie…). Il est également disponible dans n'importe quel centre de données avec le modèle de déploiement nommé "Cloud at Customer" ou en cas de besoin (sécurité), une région complète peut être installée (OCI Dedicated Region).

La famille Autonomous Database est disponible selon 4 modèles : **Figure 17**

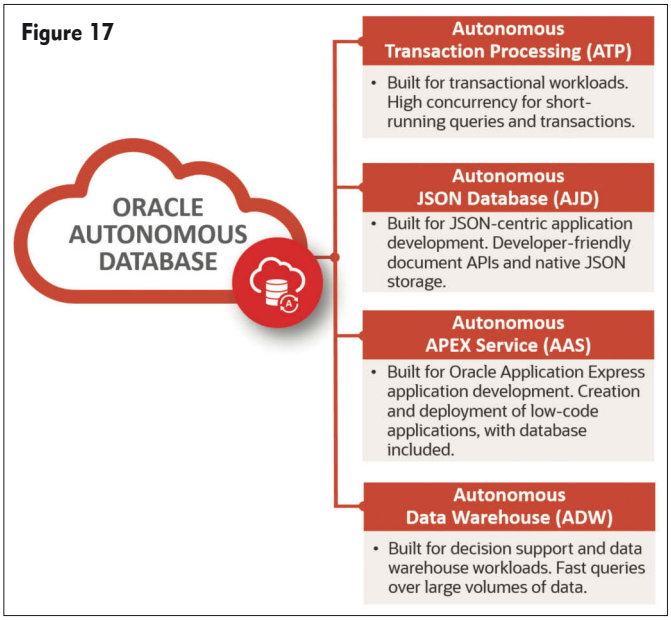
Le service ATP inclut toutes les capacités d'AJD et *n'est pas limité à 20 Go de données relationnelles*, d'AAS et d'ADW. ADW est quant à lui orienté entrepôt de données, analytique et data-science, car il permet de créer gérer tout le cycle de vie des modèles de Machine Learning (y compris sur des données JSON).

Petite particularité très utile à connaître, les services names.

Ce sont des points d'accès aux bases de données autonomes pour lesquels des ressources ont été allouées : **Figure 18** Ainsi si l'on cible une charge de travail OLTP avec beaucoup de transactions concurrentes, l'on pourra utiliser le service TP. Si l'on souhaite générer un rapport via une requête SQL qui va traiter des millions ou des milliards de lignes, le service MEDIUM ou HIGH sera préférable. Chaque service possède ses variantes en lecture/écriture ou lecture seule, et avec ou sans connexion sécurisée via TLS (la sécurité des accès peut se gérer à d'autres niveaux tels que les adresses IP autorisées ou encore les masques de sous-réseaux virtuels par exemple). Le nombre dans la colonne "fraction de ressources garanties" indique la priorité d'un service à garantir son accès aux ressources (CPU, I/O…) lorsque l'une d'entre elles est totalement utilisée. Ainsi, une connexion à un service TPURGENT sera prioritaire sur une connexion à un service LOW. Bien entendu, si aucune contention n'existe, alors une connexion pourra utiliser plus de ressources. Enfin, les colonnes "Fast Application Notification" et "Transparent Application Continuity" indiquent respectivement si les événements de type arrêt, redémarrage… sont communiqués aux programmes clients et si la reprise des transactions en cours et non validées (COMMIT) est activée par exemple lors d'une bascule sur le site de secours (switchover) ou en cas de crash (failover).

La suite de ce dossier dans le n°248

Figure 17



Service Names	Degré de parallélisme (SQL)	Fraction de ressources garanties	Ordres SQL concurrents	Fast Application Notification activé	Transparent Application Continuity activé
tpurgent	Manuel	12	300 x CPUs	Oui	Oui
tp	1	8	300 x CPUs	Oui	Oui
high	Nombre de CPUs	4	3	Oui	Non
medium	4	2	1.25 × CPUs	Oui	Non
low	1	1	300 x CPUs	Oui	Non

Figure 18

De JAVA 9 à JAVA 15 : Évolutions et nouveautés

Houyeme SOUISSI
Consultante Java
www.invivoo.com

Diplômée de Grenoble INP avec un Master de l'université technique de Darmstadt en Allemagne, Houyeme rejoint INVIVOO en mars 2019 après avoir travaillé avec Accenture à Sophia Antipolis. Passionnée par les nouvelles technologies et intéressée par la découverte de la finance de marché, elle intervient aujourd'hui au sein de la Société Générale, en tant que développeuse Java. Elle fait de la veille active en participant au partage de connaissance au sein du programme d'évolution Grow Together, afin de développer ses compétences en continu, et d'apporter les solutions les plus pertinentes aux problématiques toujours plus complexes des entreprises.

Utilisée par des millions de développeurs, Java reste un des langages le plus populaire et le plus utilisé dans le monde d'entreprise. En effet, aujourd'hui, selon ORACLE, on compte 9 millions de développeurs Java (dans le monde et près de 51 milliards de JVM actives).

Pour conserver cette popularité et pour élargir la communauté Java, ORACLE à accélérer le rythme des releases Java depuis la sortie de Java 9 en septembre 2017. Au lieu de fournir des dizaines de milliers de correctifs et de nouvelles fonctionnalités dans une seule release tous les trois ans, des améliorations sont maintenant livrées dans des releases tous les 6 mois et une release LTS (long time support) est disponible tous les trois ans.

L'objectif est d'assurer une meilleure productivité aux développeurs et de leur permettre de s'adapter aux nouvelles pratiques du marché, tout en offrant une prévisibilité et une stabilité continue. Cela facilitera également la migration vers des nouvelles versions et élargira la communauté Java et cet enjeu est déjà en œuvre avec Java 15 où plus de 20% des correctifs sont apportés par des développeurs non ORACLE.

Ce changement de rythme après Java 8 a été assez soudain et beaucoup d'entreprises n'ont pas encore emboîté le pas de ce nouveau rythme, par crainte ou par méconnaissance des bénéfices apportées par chacune des nouvelles versions. Cet article a pour but de donner une vue d'ensemble sur les différentes évolutions qui ont été amenées par les versions de Java 9 à Java 15 afin qu'elles se préparent au mieux à la migration de leur SDK.

Dans une première partie, nous allons nous intéresser aux évolutions techniques du langage qui ont simplifié la vie du développeur et amélioré son quotidien. Ensuite, nous allons étudier les différentes améliorations apportées à la JVM, notamment les « Garbage Collectors » récemment disponibles. Enfin, nous allons finir avec une troisième partie dédiée aux modules et aux API supprimés ou à supprimer du JDK. Nous conclurons avec quelques recommandations pour que vous puissiez aborder votre migration en toute sérénité.

Améliorations techniques

Les blocs de texte (MODE Standard depuis Java 15)

Les chaînes littérales dans la programmation Java ne sont pas limitées à des chaînes courtes, mais peuvent aussi correspondre à des descriptions en XML, des requêtes SQL, des pages web en HTML, etc.

Elles peuvent contenir alors plusieurs séquences d'échappements, des retours en lignes, des caractères spéciaux…

Prenons l'exemple d'une page web en HTML :

```
String html = "<html>\n" +
              "   <body>\n" +
              "      <p> Java, technical Enhancement </p>\n" +
              "   </body>\n" +
              "</html>\n";
```

Ces écritures sont lourdes et peuvent également être une source d'erreur pour le développeur.

Pour résoudre ce problème, Java 13 a apporté la nouvelle notion de « blocs de texte » bidimensionnels en mode préview et cette notion a été standardisée avec Java 15. Voici à quoi ressemblera l'exemple précédent :

```
String html = """  //Retournez à la ligne
        <html>
          <body>
            <p> Java, technical Enhancement </p>
          </body>
        </html>
        """;
```

Dans le même contexte, de nouvelles méthodes sont introduites par java 13 pour la classe String :

- String::translateEscape : retire les séquences d'échappement dans une chaîne de caractères
- String::stripIndent : supprime l'indentation accidentelle au début de chaque ligne.
- String::formatted : permet de formater un string selon les paramètres passés.

On trouve aussi d'autres méthodes pratiques introduites précédemment par Java 11 :

- String::isBlank, String::repeat,
- String::lines : retourne une stream à partir des lignes
- String::strip, String::stripLeading , String::stripTrailing : permettent une meilleure gestion des espaces.

Les switchs (MODE Standard depuis Java 14)

Les blocs de textes ne sont pas les seuls à être assez verbeux en Java, le « switch » est aussi une instruction qui s'écrit historiquement sur beaucoup de lignes.

Voilà à quoi ressemble un « switch » avant Java 12 :

```
switch (day) {
   case MONDAY:
   case TUESDAY:
   case THURSDAY:
      planning = "Training";
```

```
        break;
    case SUNDAY:
    case SATURDAY:
        planning = "Work";
        break;
    case WEDNESDAY:
    case FRIDAY:
        planning = "Weekend";
        break;
    default:
        throw new IllegalStateException(day+ "does not exist");
}
```

Après Java 12, on a quelque chose de beaucoup plus concis :

```
int planning = switch (day) {
  case MONDAY, TUESDAY, THURSDAY -> "Training"; // “case” accepte plusieurs valeurs séparées par des virgules
  case WEDNESDAY, FRIDAY -> "Work"; // il est possible d'utiliser l'operateur “arrow”
  case SATURDAY, SUNDAY -> "Weekend"; // pas besoin de mettre “default” pour les enums si on couvre tous les cas
  };
```

Il est possible également de mettre un bloc de code dans la close case. Java 13 a apporté à son tour un nouveau mot clé « yield » qui vient remplacer break et qui permet de sortir du switch. Ces nouveautés sont passées en mode standard avec Java 14.

VAR pour les variables locales (MODE Standard depuis java 10)

Prenons l'exemple suivant :

```
BufferedReader reader = Files.newBufferedReader(...);
List<String> programmingLanguage = List.of("java", "python", "c++", "javaScript");
Map<String, List<String>> mapping = retrieveWritersBooksMap();
```

Après Java 10, il pourra s'écrire comme ceci :

```
var reader = Files.newBufferedReader(...);
var stringList = List.of("java", "python", "c++","javaScript");
Map<String, List<String>> mapping = retrieveWritersBooksMap();
```

L'inférence des types des variables locales est la plus grande nouveauté de Java 10. Elle permet d'éviter la redondance vue dans les exemples précédents et simplifie l'affichage pour les types compliqués.
Attention, l'utilisation excessive de var peut créer une confusion au niveau du compilateur. Pour cela Java a imposé quelques restrictions pour son utilisation. Voilà quelques exemples de code non autorisé :

```
var value ;  //il faut obligatoirement initialiser les variables
var object = null ; //il faut obligatoirement initialiser les variables
var a=1,b=2 ; // impossible de déclarer plusieurs variable sur la même ligne
var words = {“word1”, ”word2”}; // l'initialisation d'un tableau nécessite un type explicite
var addition = {a, b} -> a+b; // les lamdas expressions nécessitent un type explicite, mais il est possible de caster
var compareString = String ::compareTo ; // les méthodes référence nécessitent un type explicite, mais il est possible de caster (avec Comparator<String> dans l'exemple)
var value = 10 ; value = ”hello “ ; // impossible de transformer une var déjà typée
```

Avec Java 11, Il est possible d'utiliser var à l'intérieur des fonctions lambdas. Cette utilisation présente un avantage majeur, car ça permet d'annoter les paramètres.

```
(@Nonnull var x, @Nullable var y) -> x.process(y)
```

Les streams

La nouvelle API stream introduite par Java 8 a modifié fondamentalement la façon de traiter les collections en proposant une alternative plus simple et plus performante pour le pattern « Iterator », relativement lourd à mettre en place.
Java 9 a fourni à son tour de nouvelles méthodes à cette API, voici quelques exemples :

- Stream::takeWhile : permet de parcourir une collection en utilisant un « prédicat ». On parcourt la collection tant que la condition est validée.
- Stream::dropWhile : suit le fonctionnement inverse de takeWhile. On ne commence à parcourir la collection que si la condition est validée.
- Stream::Iterate : permet d'itérer sur une collection en précisant la valeur de départ, un prédicat et une fonction d'incrémentation.
- Stream::ofNullable : permet d'éviter les NPE en renvoyant une interface “Optional” quand la valeur est nulle.

Java 11 a introduit les Null/InputStream, OutputStream, Reader, Writer qui permettent d'initier un stream avec Null sans générer de NPE. Il est donc possible de traiter d'une manière transparente des input/output même s'ils représentent des flux d'entrée/ sortie nulles.
Pour finir avec les « streams », Java 12 a introduit la méthode Stream::teeing sur l'interface java.util.stream.Collectors. Elle prend en entrée deux collections indépendantes et permet de les combiner en utilisant une bi-fonction :

```
Double average = numbers.stream()
                .collect(teeing(
                SummingDouble(i -> i),
                counting(),
                (sum, n) -> sum / n));
```

Les classes « Sealed » (MODE PREVIEW)

Le mot clé sealed est l'une des plus importantes nouveautés de java 15. Il a pour objectif de restreindre l'implémentation ou l'héritage d'une classe/interface à une liste de classes définie par le mot clé « permits » :

```
public abstract sealed class Shape
  permits Circle, Rectangle, Square {...}
```

Si les classes sont déclarées dans le même fichier source, la close « permits » peut être retirée. Il faut garder à l'esprit que cette nouveauté reste toujours en mode « preview ». Elle sera éventuellement standardisée avec les versions suivantes.

Les classes records (MODE PREVIEW)

Les records sont des POJO moins verbeux. En effet, ils permettent de réduire la quantité de code requise pour créer une classe (constructeur, accesseurs getter/ setter, les méthodes equals(), hashcode()) et le temps nécessaire pour la mainte-

nir à chaque fois qu'un nouvel attribut est créé.
Voici un exemple :

```
public record Person (String firstName, String lastName) {}
```

Ils étaient initialement introduits par Java 14, mais restent toujours en mode « preview » en s'adaptant aux améliorations apportées par Java 15. En effet, un record pourra implémenter une sealed interface, et peut être déclaré localement à l'intérieur d'une méthode.

Null Pointer Exception (MODE STANDARD depuis Java 15)

```
String emailAddress = employee.getPersonalDetails().getEmailAddress()
.toLowerCase();
```

Comme dans cet exemple, un développeur peut souvent écrire un code qui enchaîne plusieurs méthodes. Mais lorsqu'une NPE est générée, il peut devenir difficile de savoir d'où provient l'erreur.

```
Exception in thread "main" java.lang.NullPointerException
at com.enhancement.DemoNullPointerException.main(HelpfulNullPointer
Exception.java:10)
```

Avec Java 14, la JVM fournit un message plus explicite indiquant exactement d'où vient l'erreur.

```
Exception in thread "main" java.lang.NullPointerException:
Cannot invoke "String.toLowerCase()" because the return value of
"com.enhancement.DemoNullPointerException$PersonalDetails.getEm
ailAddress()" is null
at com.enhancement.DemoNullPointerException.main(HelpfulNullPointer
Exception.java:10)
```

Cette amélioration passe en mode standard avec Java 15.

JShell REPL « Read Evaluate Print Loop » (MODE STANDARD depuis Java 9)

« JShell », introduit par Java 9, est un outil de commande en ligne qui permet d'évaluer le code Java.
En effet, plus besoin de créer tout un programme (importer des bibliothèques, définir une classe avec une méthode main(), etc.) pour tester une simple expression. Cela paraît utile surtout pour les développeurs qui débutent avec le langage Java.
Voici un exemple de code exécuté avec Jshell :

```
.. >jshell
|  Welcome to JShell -- Version 9
|  For an introduction type: /help intro

jshell>public long multiply(long n , long m ){
  ...>    return m * n ;
  ...> }
|  created method multiply(long ,long)
jshell>long result = multiply(20 , 10 )
result ==> 200
|  created variable result : long
```

« Jshell » fournit une liste de commandes parmi lesquelles, on trouve :

/set feedback verbose : permet d'obtenir plus d'informations sur les commandes exécutées.
/list -start : liste toutes les commandes Jshell.
/drop [nom_variable] : permet de supprimer la variable créée.
/vars : permet d'afficher la liste de toutes les variables actives dans la session en cours.
/vars [Nom_variable] : permet d'afficher la variable [Nom_variable] et sa valeur.
/vars – all : permet d'afficher la liste de toutes les variables actives, inactives et chargées au démarrage.

/types : permet de lister l'ensemble des types (Class, Interface, Enum) actifs créés dans JShell.
/types [Nom_Type] : permet d'afficher le type correspondant à [Nom_Type].
/types -all : permet de lister l'ensemble des types de la session en cours (actifs, inactifs et chargés au démarrage de JShell).

/edit : permet de modifier les constructeurs dans la session en cours.
/edit 1 : permet de modifier le premier constructeur dans la session en cours.
/edit [Nom-constructeur] : permet de modifier un constructeur déterminé dans la session en cours.
/exit: permet de quitter JShell.

Pour finir, voici quelques autres améliorations qui peuvent vous intéresser :

Les méthodes « private » dans les interfaces : Java 9
Cela Permet de faciliter l'encapsulation et éviter de dupliquer certaines parties du code et d'exposer uniquement les méthodes souhaitées.

Des fabriques pour des collections immutables : Java 9
List.of(),Set.of(),Map.of().

```
List<String> availableGC= List.of("GC1", "ZGC", "EPSILON");
```

Les variables finales dans Final variables in « try-with-resources » Java 9 : Il est possible de déclarer les ressources « Closeable » à l'extérieur du bloc « try » si elles sont finales. Il est également possible pour faciliter la lisibilité de créer des méthodes utilitaires pour l'instanciation des ressources.

Predicate ::not Java11 : fournit un moyen facile d'inverser la valeur d'un « Predicate » exprimé sous la forme de lambdas ou de références de méthodes, ce qui réduit la complexité du code.

Amélioration de la JVM

Quand on rencontre des problèmes de performance (une application qui met beaucoup de temps pour démarrer, des erreurs de type Java OutOfMemoryError ou autre), on pense souvent au code (améliorer sa qualité, optimiser les algorithmes, choisir les bonnes collections, etc.), mais on oublie souvent le choix du « garbage collector ».
Dans cette partie, on va se concentrer sur les améliorations de performances apportées par les dernières versions de Java, notamment les nouveaux «garbage collectors» et l'archivage des classes en Java.

1. Garbage Collector

Au cours de son cycle de vie, une application crée un certain nombre d'objets, dont la durée de vie varie selon son rôle au sein du programme. Cette durée de vie est définie par un "compteur de référence". Un objet dont le compteur de référence est à zéro est un objet non utilisé.

Un « garbage collector » permet d'identifier puis de supprimer ces objets non utilisés (ou déchets). Historiquement, il divise la mémoire en deux zones : la « YoungGen », qui stocke les objets récents, et l'«OldGen », qui stocke ceux à durée de vie longue. La libération de ces zones (collecte ou GC pour Garbage Collection) se fait ensuite suivant des algorithmes comme le Comptage de références, algorithme « Mark and Sweep », algorithme « Stop and Copy », etc.

On va parler des nouveaux GC (Garbage collectors) implémentés depuis Java 9.

EPSILON No-Op Garbage Collector

Introduit par Java 11, Epsilon est un GC no-op (passif). Il gère seulement les allocations mémoires, mais ne permet pas le nettoyage des objets non utilisés. Quand le tas (« heap ») alloué par l'application est épuisé, la JVM s'arrête. Epsilon est utilisé dans le cas des applications à courte durée, sans déchet ou quand on sait que la mémoire allouée (taille de heap) est largement suffisante pour l'application en cours. Il peut être également utile pour réaliser des tests de performance (test des nouveaux algorithmes de GC, test de pression de la mémoire, etc.).

G1 Garbage First

Le garbage collector « Garbage-first », utilisé par défaut à partir de Java 9, fonctionne principalement avec les threads d'application (comme le CMS), mais il permet d'offrir des temps de pause plus courts et plus prévisibles.

En effet, au lieu de diviser le tas en 2 grandes régions, il le divise en petit lots de taille égale. Les données utilisées dans chaque lot sont tracées. Lorsqu'une collecte est déclenchée, le G1GC effacera en "premier" les lots qui contiennent le plus de « déchets » - d'où son nom « first ».

Mais le G1 n'est pas optimal dans toutes les situations. En effet, s'il n'arrive pas à récupérer rapidement la mémoire non utilisée, il arrête les threads d'application pour faire un full GC. Avec Java 10, au lieu d'utiliser un seul thread lors du full GC, il est devenu possible de lancer plusieurs threads en parallèle (Parallel full GC)

On peut alors personnaliser le nombre de threads utilisés avec l'option « -XX : ParallelGCThreads »

Il existe également 2 améliorations importantes apportées par Java 12 à ce GC :

- G1 commence par définir le temps nécessaire pour faire une collection ("collection set"). Une fois démarrée, G1 doit collecter tous les objets utilisés sans s'arrêter. Mais cela peut prendre beaucoup de temps si la "collection set" est trop grande. Pour résoudre ce problème, avec Java 12, G1 divise la collection set en deux parties : une partie obligatoire et une partie optionnelle qui ne sera effectuée que si le GC ne dépasse pas le temps de pause prévue.
- Java 12 a fourni également une deuxième amélioration. Le G1, retourne automatiquement la mémoire non utilisée aux systèmes d'exploitation (non seulement lors du full GC comme avec les anciennes versions de Java).

Avec Java 14, G1 est devenu « NUMA-Aware » (NUMA : Non Uniform Memory Access) en utilisant l'option « +UseNUMA » Cette fonctionnalité cible principalement les machines ayant plusieurs sockets ou un grand nombre de cœurs.

ZGC (Concurrent Garbage Collector)

ZGC est un GC évolutif et à faible latence. En effet, ses temps de pause n'excèdent pas les 10 millisecondes et son débit de réduction d'application est inférieur à 15% par rapport au G1. Lors de son lancement avec Java 11, ZGC ne renvoyait pas la mémoire au système d'exploitation même si elle n'a pas été utilisée depuis longtemps. Java 13 a fourni cette nouvelle fonctionnalité. Cela est utile dans le cas des applications où l'empreinte mémoire pose problème, ou dans le cadre d'un système avec plusieurs applications actives.

ZGC se base sur des pointeurs de couleurs pour stocker des informations relatives au marquage et à la relocation de mémoire. Ça permet de garder tout type d'information et donc d'agir selon ces données (cela n'est possible qu'avec un processeur 64 bit).

Il faut noter aussi qu'à partir de Java 14, il est disponible avec « macOS » et « Windows ».

ZGC est aujourd'hui stable, performant, à faible latence et prêt à être utilisé en production à partir de Java 15.

Il pourrait rivaliser avec « Shenandoah », également destiné aux applications à grand segment de mémoire.

Shenandoah (Concurrent Garbage Collector)

Java 12 a introduit « Shenandoah » (il peut être configuré avec Java 8). « Shenandoah » permet de diminuer le temps de pause du collecteur. En effet, ses tâches (marquage, libération de mémoire, compactage) sont exécutées dans des threads concurrents aux threads applicatifs utilisés par le programme en cours.

Il est surtout utile dans le cas des applications nécessitant un temps de réponse rapide et prédictible. Il faut noter aussi que la taille des « heaps » n'affecte pas le temps de pause. Ce GC est maintenant fourni en mode standard avec Java 15.

CMS Concurrent Mark and Sweep

Après avoir été déprécié avec Java 9, CMS est finalement supprimé avec Java 14. Cette décision a pour but de réduire la charge de maintenance de la base du code GC et accélérer le développement des nouveaux algorithmes.

Comparaison des nouveaux GC disponibles

Pour choisir le meilleur GC pour une application Java, trois aspects importants doivent être pris en compte :

- Taille de la « heap » : taille de mémoire nécessaire pour faire exécuter une application
- Le temps de pauses de l'application : le temps nécessaire pour le GC pour exécuter ses tâches (principalement le full GC).
- « Throughput » ou débit de l'application : la vitesse à laquelle une application Java s'exécute (Considérer le temps nécessaire pour effectuer les tâches du GC et le temps consacré pour exécuter le code). (voir tableau page suivante).

2. Système d'archivage

En Java, pour exécuter le « bytecode » d'une classe donnée, la JVM effectue quelques étapes préparatoires. En effet, en se basant sur le nom de la classe, la JVM la cherche sur le disque, la charge, vérifie son « bytecode » et puis la met dans une structure de données interne. Cela peut prendre beaucoup de temps lors du lancement de l'application (chargement d'une centaine de classes, voir plus) ou l'exécution d'une nouvelle fonctionnalité pour une première fois. Ces opérations sont inutiles tant que les jars de l'application ne changent pas. En effet, la JVM exécute les mêmes étapes pour finir avec les mêmes résultats à chaque fois que l'application est lancée.

La notion de CDS archive «class data-sharing », introduite par Java 8, vient résoudre ce problème.

Default CDS archives

L'idée des CDS consiste à créer des jars une seule fois, les enregistrer dans une archive, et les réutiliser ultérieurement lors des prochains lancements de l'application.

Cette archive peut être partagée par les instances de JVM exécutées simultanément. Cela permet d'économiser la mémoire autrement gaspillée dans la réplication des données pour chaque instance de JVM.

Pour bénéficier des default CDS, il faut suivre ces trois étapes :

- Créer une liste de classe pour les mettre dans l'archive : -XX:DumpLoadedClassList
- Créer une archive : -Xshare:dump et -XX:SharedArchiveFile
- L'utiliser avec l'option : -Xshare:on et -XX:SharedArchiveFile

JDK 10 vient avec une liste de classes à archiver. La première étape peut donc être ignorée à partir de Java 10.

L'archivage des classes JDK (defaults CDS) est appliqué automatiquement à partir de Java 12.

AppCDS : Application class data sharing

« AppCDS » concerne l'archivage des classes de l'application et non seulement celles du JDK.

Avant Java 13, la création et l'utilisation d'une archive pour les classes de l'application devait suivre la même logique que les classes JDK.

Cependant, avec cette version, il est devenu possible de combiner les deux premières étapes et faire un archivage dynamique des classes après l'exécution de l'application (« Dynamic CDS »).

En ajoutant l'option -XX: ArchiveClassesAtExit, la JVM enregistrera toutes les classes chargées de l'application et les placera dans une archive dédiée.

Les classes archivées incluront donc les classes d'application et les classes de bibliothèque qui ne sont pas présentes dans l'archive CDS par défaut.

L'« AppCDS » fournit des avantages supplémentaires en matière de temps de démarrage et de mémoire par rapport à l'archive CDS par défaut.

Les métadonnées des expressions « lambdas » sont rajoutées à l'archive « AppCDS » à partir de Java 15.

GC	CPU « throughput»	Taille de la « heap »	Temps de pause	Type d'application
G first	- Elevé - Seulement les tâches (Mark & Sweep) du marquage et de la suppression des objets sont concurrents aux threads de l'application	La performance dépend de la taille de la « heap »	Les temps de pauses sont prédictibles et peuvent être définis, mais sont proportionnels à la taille de la « heap »	Les applications ayant une petite « heap », ou acceptant des temps de pauses longs
Shenandoah	Les tâches « MARK Sweep et Compact » sont concurrents aux threads de l'application. Cela peut baisser le « throughput»	- Utilise plus de mémoire que le G1, car il a besoin des pointeurs pour le marquage - La taille de la « heap » n'impacte pas la durée de pause	Temps de pause réduits par rapport à G first.	- Les sites web interactifs - Les applications n'acceptant qu'un temps de pause réduit (Low-latency-system) - Pas besoin de migrer vers une version récente de Java, car Il peut être utilisé avec Java 8
ZGC	Les tâches « MARK Sweep et Compact » sont concurrents aux threads de l'application. Cela peut baisser le « throughput».	La taille de la « heap » n'impacte pas la performance	Meilleur temps de pause (inférieur à 10 ms)	- Les applications ayant besoin d'une grande mémoire (big data) - Les applications n'acceptant qu'un temps de pause réduit (Low-latency-system)
Epsilon	Elevé	Utilise la mémoire allouée sans recyclage	Pas de temps de pause	- Utilisé pour des tests de performance - Les applications dont la « heap » nécessaire est toujours prédictible - Les applications où aucun temps de pause n'est permis (ultra-latency-sensitive-application)

3. Compact String

Avant Java 9, les strings étaient stockés dans un tableau de char où chaque caractère prenait deux octets (1 char = 2 octets).

Avec Java 9, il y'a eu un changement d'implémentation des strings. En effets, elles sont maintenant stockées dans un tableau d'octets et chaque caractère a besoin seulement d'un seul octet.

Mais cela n'est possible que si elles compatibles ISO-8859-1, ce qui est souvent le cas.

Sinon, les caractères seront quand même stockés dans un tableau d'octets, mais il faudra deux octets pour chaque caractère. Un flag est ajouté à la classe String pour connaître le codage de celle-ci.

La compacité des strings a permis une diminution de presque 50% de la taille nécessaire pour stocker les strings. Les strings sont la principale source d'utilisation de la mémoire (25% des objets sont des String). Cela a permis donc de réduire la taille de la « heap » utilisée (Gain de 5% à 15%) et donc d'améliorer la performance de la JVM.

Ce changement d'implémentation disponible avec Java 9 était fait sans modification des interfaces publiques existantes ni la création d'une nouvelle API ou autre.

4. Java Flight Recorder et Java Mission Control

Java Flight Recorder (JFR) collecte les données de diagnostic et de profilage d'une application Java en cours d'exécution. JFR a un impact minime sur une application Java en cours d'exécution.

Java Mission Control (JMC) permet d'analyser les données collectées par JFR. Il les affiche sous forme graphique et permet donc de les explorer en détail.

JFR et JMC étaient des fonctionnalités commerciales dans Java 8, et passent en mode open source avec Java 11. Avec ces outils, on peut diagnostiquer les problèmes de « runtime » comme la surcharge de GC, la fuite mémoire, etc.

Modules et classes supprimés

Avec les dernières versions de Java, plusieurs modules ou API étaient dépréciés et même supprimés.

En effet, La dépréciation encourage les applications à s'éloigner de ces API et donc d'éviter d'y rajouter toute dépendance. L'outil « jdeprscan », fourni par Java 11, permet aux développeurs d'effectuer une analyse statique de leurs fichiers jar (ou autre agrégation de fichiers de classe) pour identifier les utilisations des API obsolètes, leur permettant ainsi de se préparer à l'avance pour une éventuelle migration vers une nouvelle version de Java.

Java 11 a introduit également un nouvel outil « Jdep» qui permet d'analyser les dépendances de l'application y compris les dépendances vers des API internes. Il fournit également une suggestion sur ce qu'il faut utiliser en remplacement.

Voilà quelques modules supprimés par les dernières versions de Java :

Java 11

- Les modules contenant Java EE et Corba sont dépréciés avec Java 9 et supprimés avec Java 11 :
- java.xml.ws (JAX-WS, Java API pour XML-Based Web Services)
- java.xml.bind (JAXB, Java Architecture pour XML Binding)
- java.activation (JAF, JavaBeans Activation Framework)
- java.xml.ws.annotation
- java.corba (CORBA)
- java.transaction (JTA)
- java.se.ee (module agrégateur pour les six précédents modules)
- jdk.xml.ws (outil pour JAX-WS)
- jdk.xml.bind (outil pour JAXB)

Java 14

- La combinaison des deux algorithmes « Parallel Scavenge » et « Serial Old garbage collection » est dépréciée (très peu utilisée, mais nécessitant un effort de maintenance important)
- CMS est supprimé et remplacé par G1
- « pack200 », « unpack200 » et « Pack200 API » (dans java.util.jar package) utilisés par les développeurs pour compresser et décompresser les fichiers jar

Java 15

- Le mécanisme d'activation RMI (Remote Method Invocation) est déprécié et il sera supprimé dans une version future
- Le moteur de script et les API JavaScript « Nashorn », ainsi que l'outil jjs sont supprimés
- Le code source et la prise en charge de build pour les ports « Solaris / SPARC », « Solaris / x64 » et « Linux / SPARC » sont supprimés. Ces ports ont été marqués comme obsolète dans JDK 14

Conclusion

Cet article a exploré les nouvelles fonctionnalités fournies par Java pendant les trois dernières années, ainsi que les nouveaux outils et les améliorations de performance, en particulier les «Garbage Collectors» et l'archivage des classes.

Afin de bénéficier de ces nouveautés techniques et de ces optimisations de performance et de sécurité, la migration vers une version récente de Java paraît nécessaire. Pour ce faire, plusieurs aspects doivent être étudiés. Premièrement, comme expliqué dans la dernière partie de l'article, plusieurs modules ont été supprimés du JDK (en effet, la dépréciation est prise beaucoup plus au sérieux dans les dernières versions de Java). Les dépendances vers ces modules dépréciés ou vers des bibliothèques tierces doivent alors être étudiées avant tout changement. De plus, le «garbage collector» utilisé par défaut dans les versions Java n'est pas toujours le même. Il faut bien penser à faire une étude de performance de l'application avant toute migration. Par ailleurs, Java a mis à la disposition des développeurs une multitude d'outils permettant de faciliter et d'alléger ce changement comme «jdeprscan», et «jdep» qui permettent d'analyser les dépendances de l'application,

«jLink» qui permet de packager et de déployer les éléments nécessaires du JDK dont on a vraiment besoin, «JEnv» qui facilite la gestion de plusieurs installations de version de JDK, etc.

Mais comment choisir sa version de Java ? Pour pouvoir suivre cette cadence de releases et être à jour avec les dernières nouveautés, les entreprises doivent trouver un compromis entre la gratuité, la stabilité et la sécurité. Les versions LTS sont gratuites et plus stables, mais elles sont fournies chaque trois ans et les mises à jour de sécurités sont payantes. Les versions non-LTS (sorties tous les six mois) fournissent plus de nouveautés techniques, ainsi que des améliorations de performance et de sécurité. Cependant, elles sont jeunes, moins stables et leurs fonctionnalités, notamment celles en « Preview », peuvent être modifiées, voire supprimées.

Il faudra alors définir clairement vos priorités en terme de « gratuité-stabilité-sécurité » et mettre en place les garde-fous nécessaires (par exemple ne pas utiliser de fonctionnalité en mode Preview) et un comité en charge du suivi et de la communication sur ces versions pour vous lancer la mise à jour de vos SDK. En effet, ce sera un incontournable si vous ne voulez pas refaire votre parc applicatif. À vous de jouer

Linky : affichage déporté et auto-alimenté compteur linky – DIY

Olivier Bersot

Ingénieur en informatique et électronique avec une spécialisation en IA.

Actuellement responsable de projets informatiques dans la finance. Il passe une grande majorité

de son temps libre à partager ses tutoriels dans l'univers passionnant de l'électronique embarquée sur son site

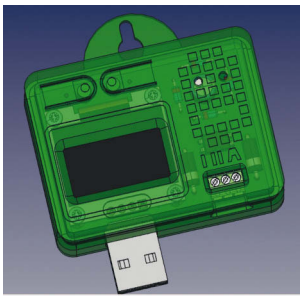
https://www.papsdroid.fr

Ce projet permet de déporter l'affichage d'un compteur Linky sur un OLED piloté par un microcontrôleur PYBStick26 Lite. Ce MCU est peu gourmand en énergie et il peut-être autoalimenté par la prise TIC du compteur Linky sans aucune source 220v. Cela permet de voir en temps réel la consommation en W, les indexes WH, les informations de contrat, etc. sans avoir à descendre dans son garage ou dans le local dans lequel le compteur est installé.

Les sources (programmes micropython, circuit imprimé et fichiers d'impression 3D STL) sont disponibles dans le GitHub du projet : https://github.com/papsdroidfr/Linky

Avertisssement : Programmez!, la rédaction et les contributeurs dégagent toute responsabilité. Vous devez respecter les règles de sécurité. La manipulation d'électricité et de tout outil peut être dangereuse.

Prise TIC du compteur



Il faut enlever le cadre vert clipsable (vous en avez parfaitement le droit) : la prise TIC se situe en bas à droite. Il s'agit d'un bornier sans vis composé de 3 sorties : I1, I2 et A. On peut y loger des fils rigides de section 1 à 1,5mm dedans.

Figure 1

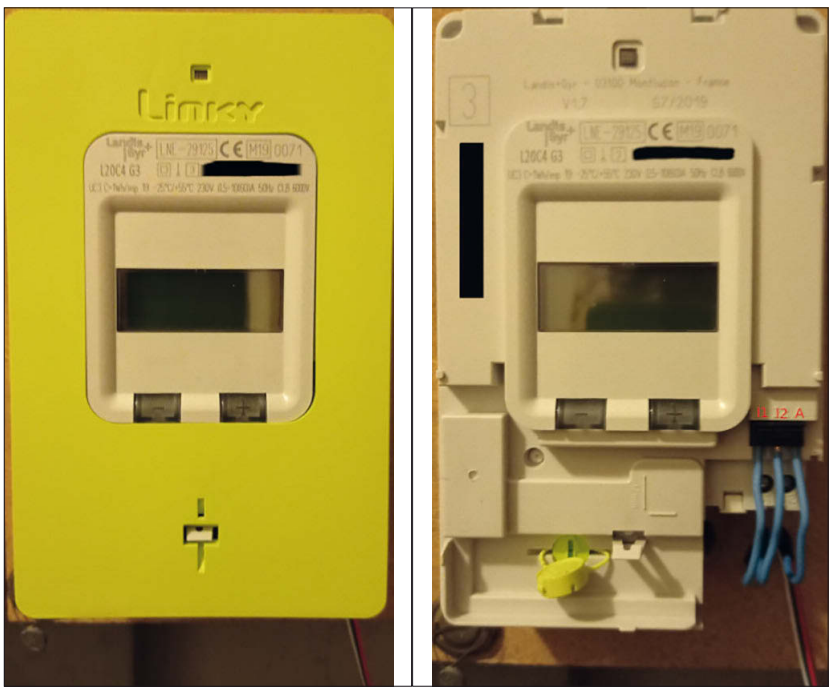
Les informations du compteur sont transmises via les prises I1 et I2 sous la forme d'un protocole UART 1200 bauds sur une porteuse 50khz. Une trame est composée d'informations de 10 bits dont 1 bit start, 7 bits qui codent un caractère ASCII, 1 bit de parité et 1 bit stop. Le décodage UART va consister à récupérer les 7 bits composant un caractère ASCII et de stocker le tout dans un tampon que l'on va lire et décoder.

Entre les prises I1 et A, on peut récupérer une alimentation sous forme de courant alternatif 50khz qui oscille entre -16 et +16v (avec des pics à 20V, selon la longueur du câble que vous allez utiliser). Pour alimenter un microcontrôleur qui exige du courant continu il faudra redresser, filtrer, et réguler cette alimentation, mais elle est limité à 130 mw : il sera impossible par exemple d'espérer faire démarrer un Raspberry Pi (même une Pi Zero).

Plus d'information technique sur la prise TIC: https://www.enedis.fr/sites/default/files/Enedis-NOI-CPT_54E.pdf

Concernant le câble, soit vous utilisez un câble monobrin (fils durs) de section 1mm et coloré, par exemple des câbles bon marché téléphoniques ou d'alarme, soit vous utilisez une nappe souple, mais il faudra souder les extrémités sur des câbles durs (ce que j'ai fait): j'ai utilisé 5m de nappe souple bon marché utilisée pour alimenter des rubans de leds par exemple et ça fonctionne très bien.

Figure 1



Circuit électronique

L'électronique se compose de 4 blocs principaux : **Figure 2**

- **Opto-couplage du signal I1, I2**: un optocoupleur vishay K814P va permettre de récupérer le signal I1, I2 avec un jeu de résistances pour ne pas saturer le transistor. Nous ajoutons des diodes transil bidirectionnelles (16v DO-15) entre I1, I2 puis entre I1,A afin d'y limiter la tension à 16V. On peut faire sans, mais elles sont là pour protéger des surtensions (j'ai déjà cramé une PYBStick26 à cause de pics de tensions assassins qui peuvent grimper à 20v parfois). Il vaut mieux les mettre...
- **PYBStick26**: un modèle Lite est suffisant. On récupère le signal pull-up de l'opto-coupleur sur la PIN S10 qui gère l'entrée Rx UART
- **Pont-redresseur**: 4 diodes 1N4148 vont redresser le signal entre I1 et A, puis une capacité 100uF va filtrer et lisser le signal. Avec en plus la diode Transil entre I1 et A on obtient ainsi un signal propre lissé à 16v max. La PYBStick26 étant équipée de son propre régulateur de tension, elle accepte en entrée Vin jusqu'à 18v : ce pont redresseur filtré est parfait pour l'alimenter.

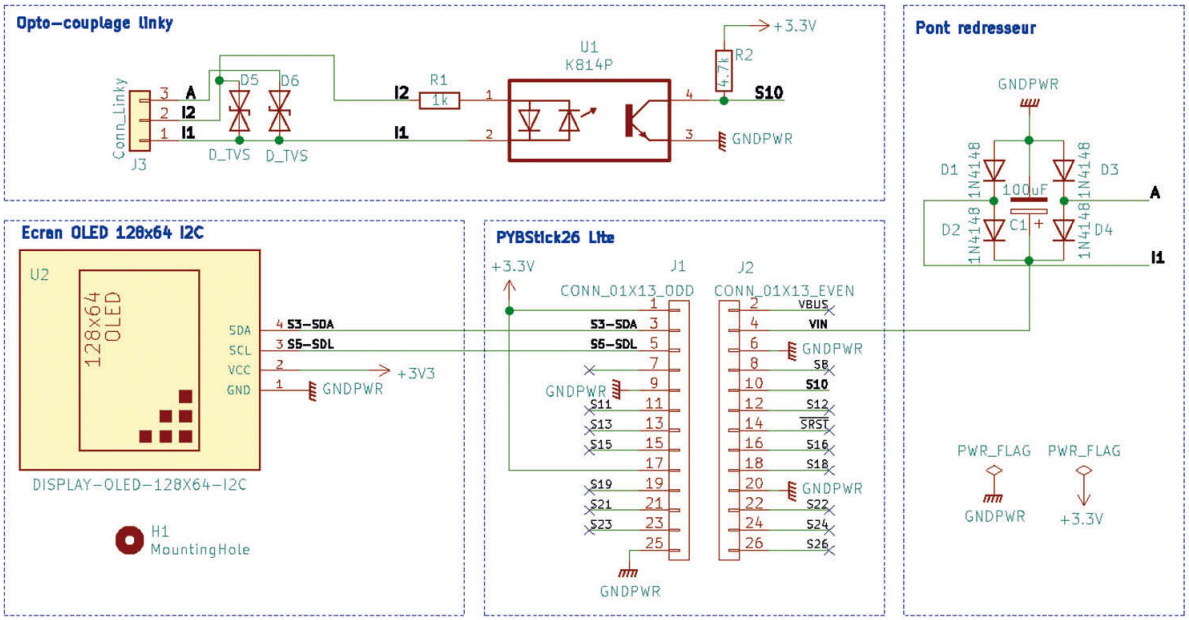


Figure 2

- **OLED** : un écran OLED I2C 0.96" 128*64 avec 4 broches est relié en I2C à la PYBStick26. Ces écrans consomment très peu de puissance, bien moins qu'un LCD. Ils sont parfaits pour notre utilisation à puissance limitée.

La **PYBStick26** est bien adaptée pour ce projet. Tout d'abord cocorico je rappelle qu'il s'agit d'un MCU **made in France** que l'on peut trouver pour quelques €. Un modèle Lite est suffisant pour ce projet. Il se programme en micropython et est équipé de connecteurs UART et I2C qui vont permettre de décoder le signal UART du compteur Linky et de communiquer avec un écran OLED I2C. Il intègre aussi son propre régulateur interne et accepte d'être alimenté aussi bien en 5v qu'en 18v max, donc pas besoin de rajouter un étage de régulation. La PYBStick26 est aussi équipée en interne de deux petits boutons poussoirs que l'on peut programmer pour son usage personnel. Enfin cerise sur le gâteau : il consomme à peine 20mA sous 5V on reste en dessous des 130mw max. Petite présentation de la PYBStick26 :
https://www.papsdroid.fr/post/pybstick

Matériel nécessaire

- 1 **microcontrôleur PYBStick26**, un modèle Lite est suffisant. C'est le cœur de notre système, programmé en microPython, capable de décoder le signal UART 1200 bauds fourni par la prise TIC du compteur Linky, alimenté de son propre système de régulateur de tension et très peu gourmand en énergie (l'alimentation par la prise TIC étant limitée à 130mw)
- 1 **câble 3 fils monobrin** (sections 1 à 1.5 mm) bon marché, à relier à la prise TIC du compteur Linky. Vous pouvez y aller avec la distance, j'ai mis 5m ça fonctionne sans aucun problème.
- 2 **pins header mâles 13 pins 2,54mm** (ne pas utiliser les header femelles qui sont fournies avec la PYBStick26).
- 1 **bornier à vis à souder type phœnix, 3 pins 2,54mm** pour raccorder les 3 fils de la prise TIC (I1, I2, A)
- 2 **diodes transil** bidirectionnelles 16v DO-15
- 1 **opto-coupleur** Vishay K814P
- 1 **résistance 1kΩ** (ou entre 1kΩ et 1.2 kΩ)
- 1 **résistance 4.7kΩ** (ou entre 2.8kΩ et 4.7kΩ)
- 4 **diodes** 1N4148
- 1 **condensateur** électrolyte polarisé 100uF 25V
- 1 **écran OLED I2C 0.96" 128*64 pixels**, équipé de 4 pins mâles 2,54mm
- 1 **carte PCB** à faire fabriquer à partir des fichiers GERBER zippés fournis dans le GitHub du projet (dossier GERBER)
- 1 **boîtier à imprimer avec une imprimante 3D** en PLA, composé de **3 parties** : rehausse boutons, coque basse et coque haute. Les fichiers STL sont à disposition dans le dossier FreeCad du GitHub du projet.
- Optionnel : 4 vis M.2.5 (le boîtier et toute l'électronique sont fermement maintenus sans les vis).

Le circuit est facilement réalisable sur une platine d'essai sans soudure de type breadboard. Mais attention : pour ce projet il ne faut pas souder les header femelles fournies avec la PYBStick26, et si vous soudez les pins header mâles sur la PYBStick26 vous serrez bien gênés après pour souder l'écran OLED sur le circuit imprimé prévu (ça reste possible, mais il faut des doigts de fée). Usage à réserver en mode prototype. Pour faire des tests et écrire le programme, j'ai relié ma PYBStick26 sur le port USB d'un Raspberry Pi3b+ que j'ai laissé dans ma cuisine. Je me suis ensuite connecté en SSH depuis mon bureau sur ce Rapsberry Pi pour pouvoir écrire et tester les programmes micropython de la PYBStick26. N'oubliez pas d'activer le SSH sur la Pi pour faire du remote. Ceci m'a évité d'avoir à tirer un long câble depuis mon Linky (proche de ma cuisine en fait) jusqu'à mon bureau : très pratique... Bien entendu pour l'utilisation finale, la Pi n'est pas utile.

Programme de test pour récupérer la trame communiquée par prise TIC

Votre PYBStick26 doit être reliée à un ordinateur (ou Raspberry Pi) via sa prise USB pour voir sur un écran la trame communiquée par la prise TIC. Dans ce cas il n'est pas nécessaire d'avoir le pont-redresseur, ni l'écran OLED, ni les diodes Transil pour cet essai, puisque la PYBStick26 sera alimentée

par la prise USB du Pi auquel la PYBStick26 est reliée.
Ce programme est disponible dans le dossier /tests du GitHub du projet :

```
#main.py -- put your code here!

import pyb
from time import sleep

print('Linky start')

# pybstick lite à 48MHz, config UART
pyb.freq(48000000)
buffer_size = 128
info = pyb.UART(2, 1200, bits=7, parity=0, stop=1, timeout=0)

def lecture_linky():
  while True:
    tampon=info.read(buffer_size)
    print('tampon lu: ', tampon)
    sleep(0.5)

try:
  lecture_linky()
except KeyboardInterrupt:
  print('Bye')
```

En vous connectant sur l'interpréteur REPL (screen /dev/ttyACM0 sur un Linux) depuis votre PC auquel la PYBStick26 est reliée, vous devez voir apparaître les caractères ASCII de la trame TIC lue dans le tampon.
Si vous ne récoltez que du "None" dans le tampon c'est que l'opto-couplage ne fonctionne pas : vérifiez bien les branchements et l'ensemble du montage. Si vous utilisez un autre opto-coupleur que le Vishay K814P il se peut qu'il soit plus ou moins vite saturé, dans ce cas il faut y aller à tâtons avec les résistances : essayez par exemple 1.2k au lieu de 1k et 3.3k ou 2.2k au lieu des 4.7k pour le pull-up en sortie du transistor. Vous pouvez inverser I1 et I2 dans le montage ça n'a aucune importance. **Figure 2**

Programme pour affichage sur écran OLED

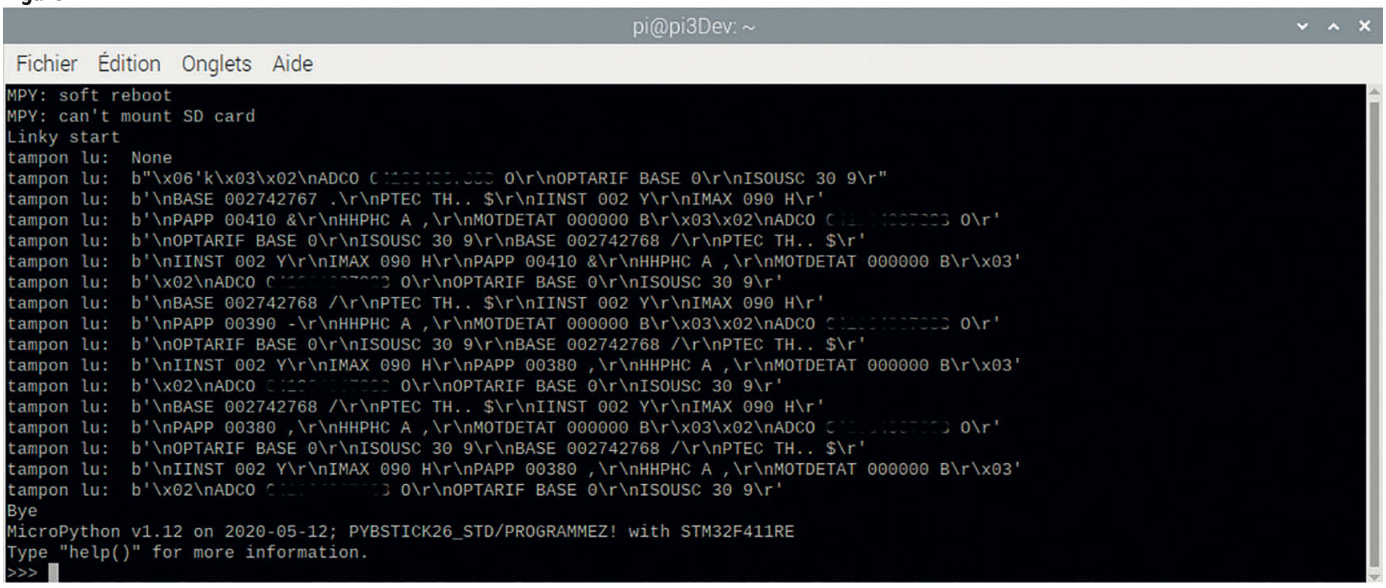
Pour utiliser le dispositif complet autoalimenté par la prise TIC, il faut d'abord réaliser le montage électronique complet, et ne surtout pas oublier les diodes transil qui vont protéger des surtensions (il y a des pics tueurs à 20v !). Bien entendu ne reliez rien d'autre que la prise TIC (I1, I2, et A) sur votre circuit : **n'allez surtout pas y mettre du 220V !** Les diodes claqueraient fort, mais vous serez surtout en danger mortel d'électrocution !
Pour communiquer avec l'écran OLED il faut installer toutes les dépendances : déposez bien tous les fichiers fournis dans /**microPython** à la racine de votre PYBStick26 (inutile de recréer le répertoire /microPython, le MCU exécute les fichiers de la racine).
Le script va lire en boucle la trame TIC stockée dans un tampon. Il faut lire la doc de la prise TIC pour comprendre comment décoder cette trame, mais ce n'est pas compliqué. Chaque index est fourni après un saut de ligne ('\n'), ensuite on a la valeur de l'index après un espace, et ainsi de suite. Chaque index a une taille qui lui est propre. Il suffit de stocker dans un dictionnaire les index que l'on veut rechercher dans le tampon, avec leur taille on sait où récupérer les valeurs lorsque l'index est repéré dans le tampon.
Les deux boutons de la PYBStick26 sont programmés pour :

- modifier le contraste (100%, 50%, 25%, Off) si vous ne voulez pas qu'il reste allumé à 100% 7J/7 24h/24.
- faire défiler deux écrans: l'un qui affiche les données de puissance en temps réel, et un autre qui affiche les données contractuelles.

Circuit imprimé (PCB)

Les composants peuvent être soudés sur une petite platine d'essai, mais un PCB a été conçu sous KiCad. Il est à utiliser avec le boîtier imprimé 3D. **Figure 3**
Les fichiers GERBER nécessaires à la fabrication de ce circuit imprimé sont disponibles dans /**GERBER/GERBER_linky.zip** du GitHub du projet.
Il faut d'abord souder les composants du pont redresseur (attention au sens des diodes 1N4148 et de la capacité), l'op-

Figure 2



to-coupleur (attention au sens). Je conseille de ne pas souder les résistances si vous n'avez pas fait d'essai avant : positionnez-les fermement en tordant les pattes par dessous (valeur sérigraphiée sur le PCB): Il faudra d'abord vous assurer d'avoir le bon jeu de résistances avant de les souder.
Soudez les 2 diodes transil (peu importe le sens, car elles sont bidirectionnelles), le connecteur phoenix (dans le bon sens : on doit pouvoir y brancher les fils de la prise TIC depuis l'extérieur de la carte). Attention à veiller à **souder d'abord** les deux pin header mâle à positionner **sous** la carte (soudures au-dessus donc), avant de souder l'écran OLED (sinon une fois l'écran soudé vous ne pouvez plus souder les 2 rangées de pin header...). Vous devez vous aider d'une breadboard pour les souder bien droits et perpendiculaires à la carte.
Dernière étape : positionnez les rehausses de bouton poussoir à travers les deux petits trous «off» et «next» (voir la section impression 3D) et vous pouvez **souder dernier la PYBStick26** sur les pinheader déjà soudés à la carte, et dans le bon sens surtout : les deux boutons poussoirs doivent être orientés vers les deux trous "Off" et "Next".
Une notice de montage pas à pas avec photos peut être consultée sur : https://www.papsdroid.fr/post/linky

Impression 3D du boîtier

Ce boîtier réalisé sous FreeCad est composé de deux parties clipsables haute et basse, sur laquelle viennent se fixer le PCB et les rehausses de bouton poussoir.
Modélisation du PCB avec les composants **(Figure 3).**
Partie basse du boîtier, dans laquelle se loge la carte PCB. **Figure 4**
Partie haute du boîtier **Figure 5**
Pour imprimer, il faut utiliser les 3 fichiers STL dans le dossier /FreeCad du GitHub du projet :

- **FreeCad/linky_display_coque_basse.stl**: partie basse du boîtier
- **FreeCad/linky_display_coque_haute.stl**: partie haute
- **FreeCad/linky_display_rehausse_boutons.stl**: rehausses de bouton poussoir.

L'assemblage est très simple, pas besoin d'explication ;-)
Quand vous branchez les fils I1, I2 et A de la prise TIC au système : **veillez à ne pas vous tromper de sens**, les indications sont sérigraphiées sur le circuit imprimé et apparaissent en relief sur le boîtier.

Usage ludique et instructif !

L'utilisation est très ludique est instructive. Allumez une lumière et vous verrez l'impact immédiat en Watts : une led va consommer 10W, tandis qu'une ampoule à filament va brûler 50 à 80W ! Allumer votre télé, un ordinateur et plusieurs centaines de W se rajoutent, voire un bon millier s'il y a un qui s'amuse dans sa chambre en full HD 4k en faisant hurler tous les ventilateurs de sa carte graphique. Le four électrique, bouilloire, micro-onde en marche, et le compteur s'affole avec plusieurs milliers de W, et retombe aussitôt les appareils éteints. Je trouve ça super rigolo est instructif. C'est un très bon moyen de mieux consommer si on se rend compte en temps réel de sa consommation.

Enjoy !

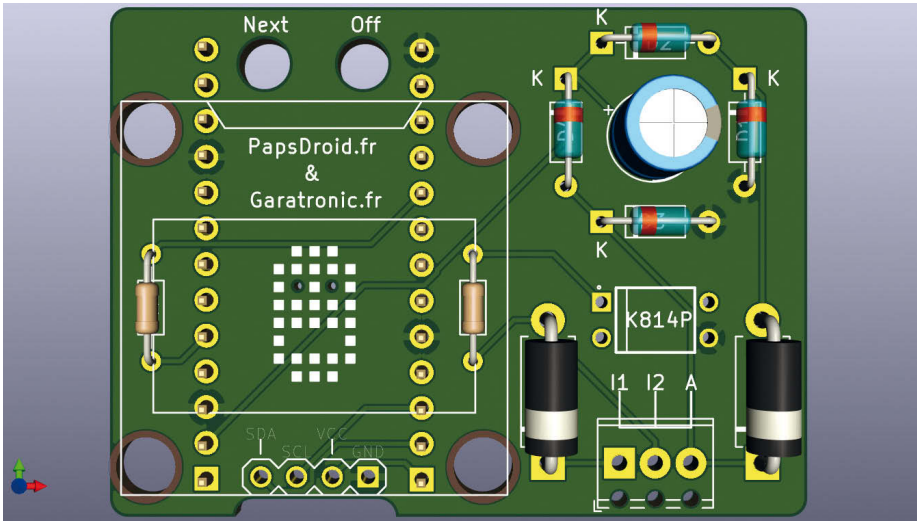


Figure 3

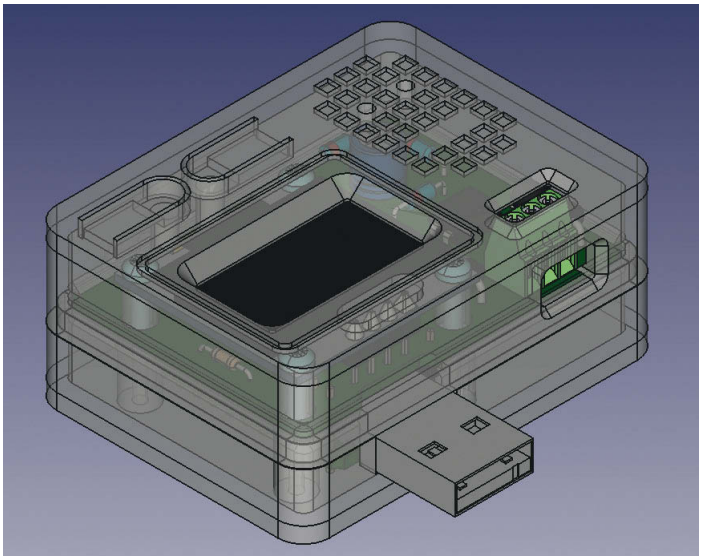


Figure 4

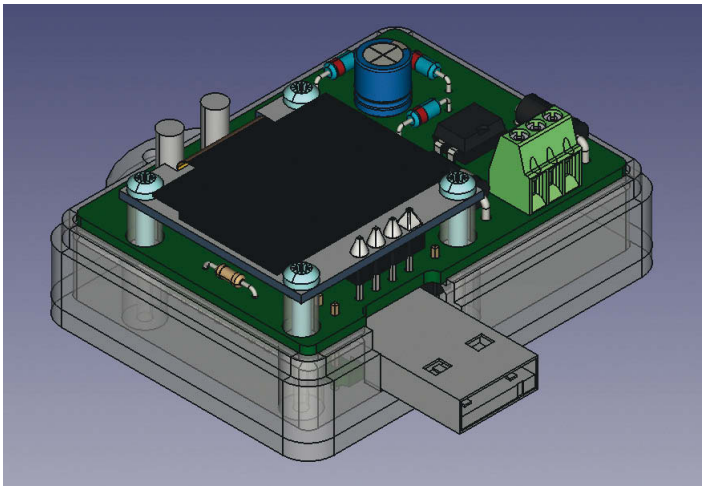
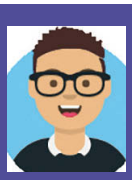


Figure 5

DIY : un robot pour résoudre votre Rubik's Cube



Sébastien Colas

Je suis formateur en informatique depuis bientôt 20 ans. Au cours de ma carrière j'ai pu dispenser des cours sur de nombreuses technologies : Serveur d'applications JavaEE, SOA, Services Web, Linux, Virtualisation, API led Connectivty et Application Network. Sans oublier les langages : Java, PHP, Python, JavaScript... Je suis aussi auteur pour « Linux Pratique » et « Hackable » autour des sujets Raspberry Pi, Arduino, IoT, électronique digitale et bien sûr Linux.

http://colas.sebastien.free.fr/

Vous possédez peut-être un Rubik's cube que vous n'avez jamais réussi à résoudre. Dans cet article nous allons construire un robot à base de Lego et de Raspberry Pi. Il va résoudre le problème pour nous. Nous commencerons par l'expression des besoins en rédigeant un mini cahier des charges, nous aborderons ensuite l'aspect matériel et nous finirons bien sûr par la programmation.

Cahier des Charges

Nous allons créer un robot Lego qui va résoudre automatiquement un Rubik's Cube. Il nous faudra donc :

- des Legos Technic
- des Legos PoweredUp
- un écran LCD
- une caméra
- un Rapsberry Pi

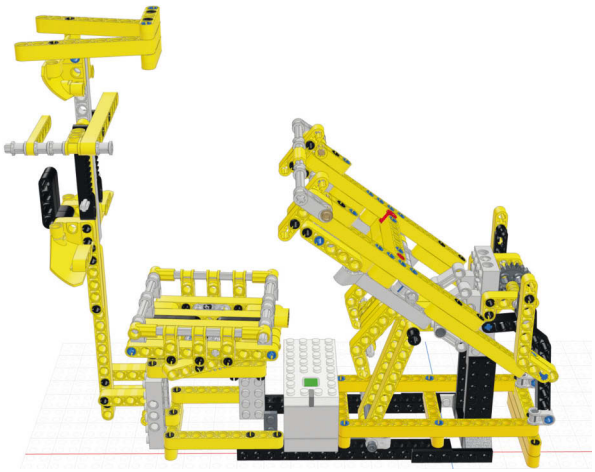
Pour la programmation du robot, nous coderons en Python 3, un langage fréquemment utilisé sur le Raspberry Pi. Nous développerons sur VSCode sur PC fonctionnant sur Ubuntu ou Windows. Le plug-in Visual Studio Code Remote – SSH, nous permettra d'éditer et d'exécuter le programme se trouvant sur notre Raspberry Pi. Toutes les informations concernant l'installation et le fonctionnement du plug-in se trouvent sur le site officiel :

https://code.visualstudio.com/docs/remote/ssh

Étapes du programme

Voici les différentes étapes du programme :

1. Connexion au Lego Powered Hub
2. Étalonnage du panier permettant la rotation du cube
3. Lecture des 6 faces du cube et remise à la position de départ du robot
4. Décodage et résolution du cube via des programmes externes et bibliothèques
5. Résolution du cube grâce au robot



Étapes du robot

Figure 1

Voici les différentes positions du bras du robot permettant le retournement du cube ainsi que le blocage pour effectuer un mouvement de résolution du cube :

- Position de départ/fin (photo de l'article)
- Position 1 : cube débloqué
- Position 2 : cube bloqué pour effectuer un mouvement de rotation
- Position 3 : cube à 3/4 retourné
- Position 4 : cube retourné complètement

Figure 1

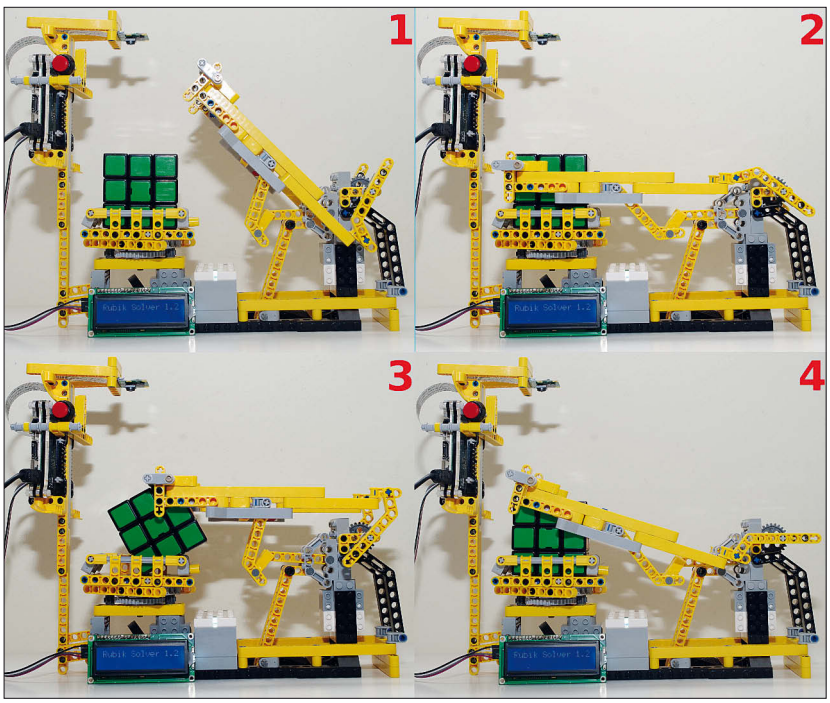
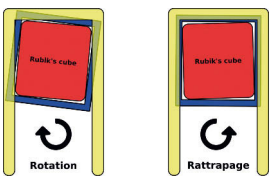
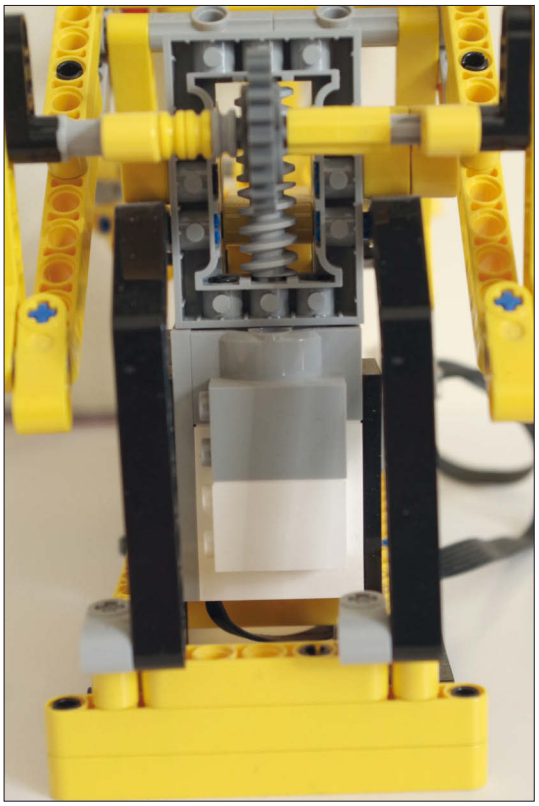


Figure 2

Pour une manipulation du cube, il faudra donc le bloquer (position 2 du bras) et effectuer une rotation du panier. Malheureusement le panier étant plus grand que le cube pour permettre son retournement, il va donc falloir faire 2 rotations :



- Rotation 1 : manipulation le cube (environ 45 °)
- Rotation 2 : remise du panier dans l'axe (pour un futur retournement du cube)



Le matériel (hardware)

Voici le matériel qui a été retenu.

Un Rubik's Cube

Le choix du Rubik's Cube s'est porté sur un modèle d'entraînement où la manipulation du cube n'implique que peu de contraintes, ceci dans le but de ne pas trop solliciter les moteurs Lego.

Des Lego **Figures 3 et 4**

Tout d'abord il va falloir un certain nombre de Lego pour construire notre robot. Inutile de réinventer la roue, un certain nombre de modèles existe déjà sur le site https://www.mindcuber.com/. Malheureusement, rien concernant les Lego PoweredUp. Notre robot sera donc librement inspiré des modèles de MindCuber.

Pour les Legos PoweredUP voici les composants retenus :

- Le Hub 88009 permettant le pilotage de moteurs via Bluetooth
- 2 moteurs moyens 88008 offrant un pilotage précis du moteur. Ces moteurs sont comparables à des servomoteurs.

Sur MecaBricks vous pourrez trouver la modélisation du robot : https://mecabricks.com/en/models/9P2k1Aoevon

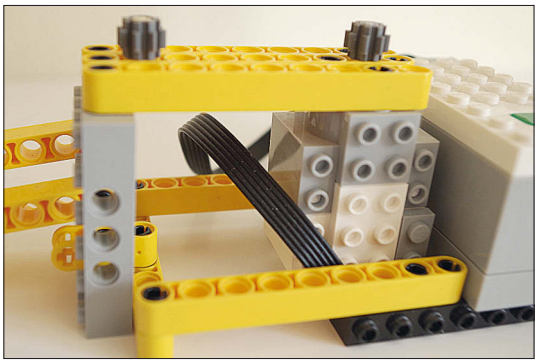
Malheureusement les moteurs 88008 ne sont pas encore disponibles sur le site MecaBricks. Vous pouvez vous référer aux photos pour les mettre en place.

Raspberry Pi

N'importe quel modèle fera l'affaire du moment que le modèle possède le Bluetooth ainsi que le connecteur pour la PiCamera. Nous utilisons un Raspberry Pi 3 Model A+.

Pi Camera

La version 1 est beaucoup trop sensible aux variations de



Figures 3 et 4

lumière, rendant la détection des faces de notre Rubik's Cube difficile à réaliser. Nous partirons donc sur une Pi Camera version 2.

Un afficheur LCD piloté par I2C

Pour avoir un affichage de la progression, nous pouvons rajouter un écran LCD pilotable par I2C. Nous utiliserons ici le matériel suivant :

- LCD IIC Interface Converter Board
- Écran LCD 16x2 rétroéclairé bleu

La programmation : fonctions de base

Nous supposerons que vous avez téléchargé et installé le système Raspberry Pi OS (anciennement Raspbian). Pensez à faire un update avant de passer à la suite.

Activez la PiCamera et l'I2C via raspi-config.

Le programme complet est disponible ici :

https://github.com/colas-sebastien/rpi-rubik

Dans cet article nous allons commenter les points principaux.

Prérequis

Un certain nombre de logiciels et de bibliothèques sont nécessaires à la réalisation de notre projet : OpenCV, BrickNil, LCDDriver, PiCamera, Kociemba, rubiks-cube-tracker, rubiks-color-resolver.

Les 2 projets ***rubiks-cube-tracker*** et ***rubiks-color-resolver*** nous permettront de décoder notre cube à l'aide des photos des 6 faces. L'algorithme Kociemba nous permettra de calculer les mouvements à effectuer pour résoudre le cube.

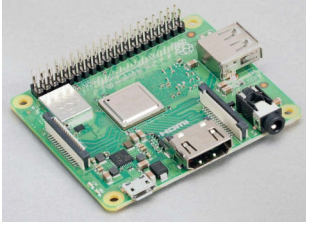
Pour installer l'ensemble des outils, en SSH ou directement sur le Terminal, tapez les commandes suivantes :

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python3-opencv python3-pip git python3-smbus python3-picamera
 python-pip python-opencv
sudo pip3 install bricknil kociemba
sudo pip install git+https://github.com/dwalton76/rubiks-cube-tracker.git
sudo pip3 install git+https://github.com/dwalton76/rubiks-color-resolver.git
git clone https://github.com/the-raspberry-pi-guy/lcd
sudo lcd/install.sh
```

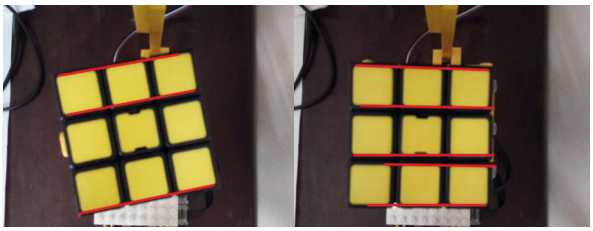
Initialisation des composants

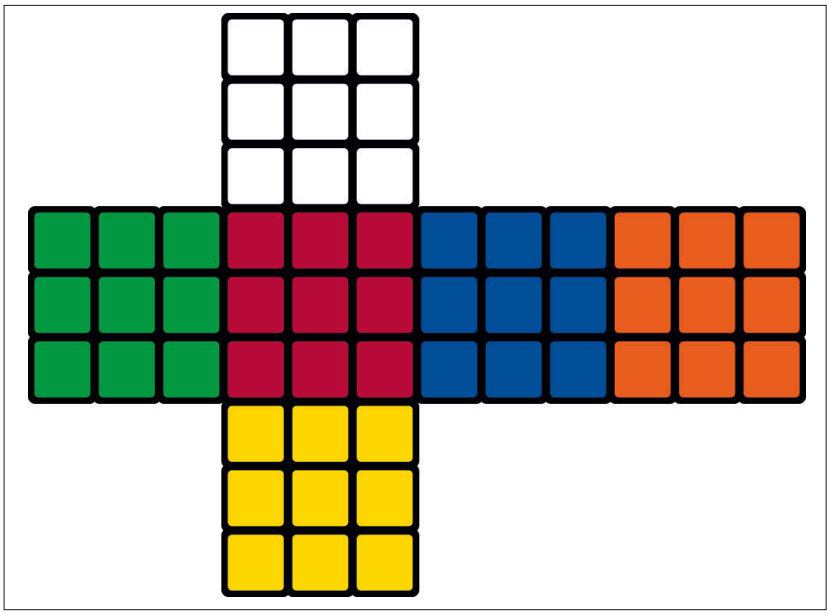
```
display = drivers.Lcd()
camera = PiCamera()
```

Ces deux commandes nous permettent d'initialiser la PiCamera ainsi que le LCD.



Figures 5 et 6





Utilisation de BrickNil

```
@attach(ExternalMotor, name='retourne',port=0, capabilities=['sense_speed'])
@attach(ExternalMotor, name='rotation',port=1, capabilities=['sense_speed'])
class Rubik(PoweredUpHub) :
```

Le programme principal est la classe python **Rubik** de type **PoweredUp**. Les ports 0 et 1 correspondent respectivement aux ports A et B sur le Hub Lego. Sur le port 0 le moteur sera nommé **retourne**. Sur le port 1, le moteur sera nommé **rotation**. On notera bien qu'il s'agit de moteur de type **ExternalMotor** et qu'il s'agit de moteurs intelligents avec capteurs de vitesse : **sense_speed**.
BrickNil nous permet de contrôler les moteurs de façon asynchrone, de manière à pouvoir donner des ordres en même temps aux différents moteurs. Malheureusement dans notre cas, nous souhaitons effectuer les mouvements en séquence, c'est-à-dire de façon synchrone. Il va donc falloir créer des fonctions pour attendre que les moteurs finissent leurs mouvements.

```
async def retourne_change(self) :
 self.retourne_vitesse=self.retourne.value[ExternalMotor.capability.sense_speed]

async def attente_retourne(self) :
 await sleep(0.5)
 while (self.retourne_vitesse != 0) :
  await sleep(0.1)
```

La première fonction est une fonction de CallBack (fonction automatiquement appelée par BrickNil quand la vitesse du moteur change). Cette fonction positionnera la variable **retourne_vitesse** à la vitesse courante. Il suffira d'attendre que la vitesse soit à zéro pour détecter la fin du mouvement, c'est l'objet de la fonction **attente_retourne**.

Fonctions de mouvement

À l'aide des fonctions précédentes, nous pouvons maintenant créer des fonctions pour les mouvements permettant de manipuler le cube. Nous allons en profiter pour leur donner un code. Fonctions de rotation du panier :

Code	Nom de la fonction
0	tourne_gauche
1	manipule_gauche
2	manipule_gauche_2
3	tourne_droite
4	manipule_droite
5	tourne_gauche_2

Fonctions de manipulation du bras :

Code	Nom de la fonction
a	position_debut
b	position_bloque
B	position_debloque
c	position_retourne
C	position_fin_retourne

La programmation : algorithme

Nous pouvons maintenant nous attaquer à l'intelligence de notre robot.

Étalonnage du panier

Notre robot dispose donc de 2 mouvements : le déplacement du bras et la rotation du panier. Le déplacement du bras ne nécessite pas un calibrage fin. On peut donc supposer que la position de départ est toujours bonne. Il ne faudra pas oublier de repositionner le bras à la fin du programme.
La position du panier est par contre critique pour pouvoir convenablement manipuler le cube. Nous procéderons donc à un étalonnage pour nous assurer que le panier se trouve dans la bonne position avant toute manipulation.

Figures 5 et 6

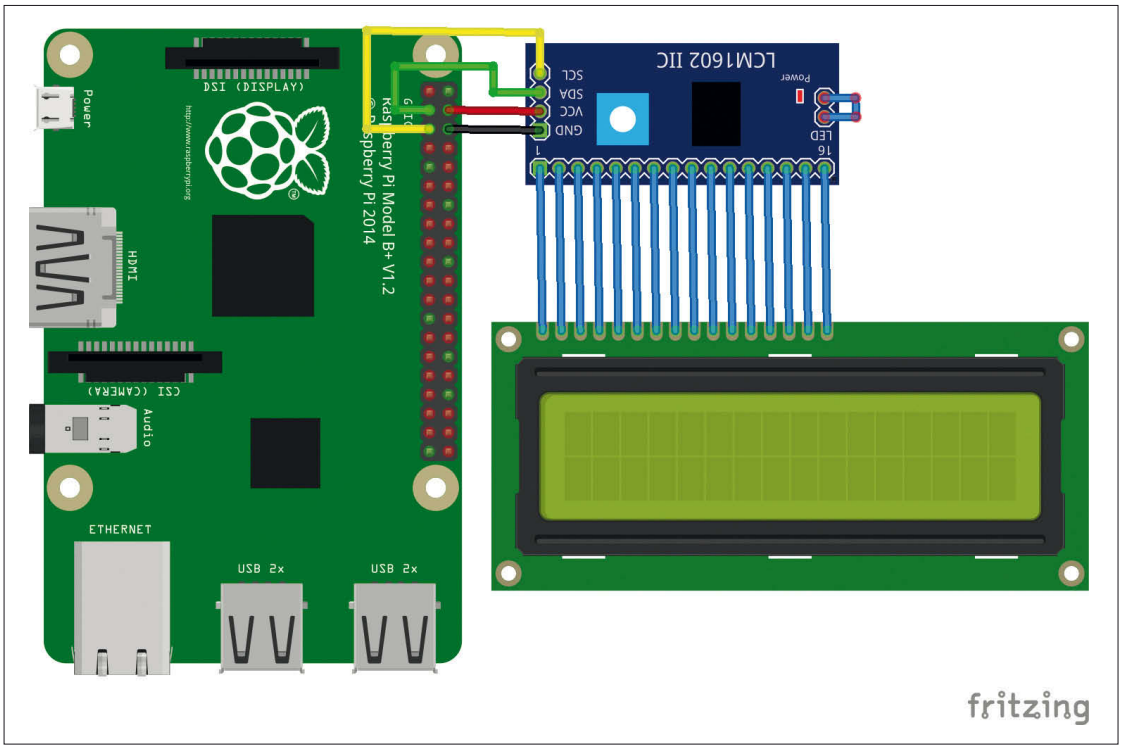
La fonction **correction_rotation** nous permettra de calculer l'angle de correction pour l'étalonnage. Voici les grandes étapes :

- Capture de l'image et détection des segments de plus d'une certaine longueur à l'aide d'OpenCV
- Calcul de l'angle de chacun des segments à l'aide de la fonction arc tangente
- Si l'angle des segments est inférieur à 45° (PI/4), on l'ajoute à la liste des segments retenus
- à l'aide d'un tableau associatif, on cherche l'angle qui apparaît le plus
- on retourne l'angle ainsi trouvé

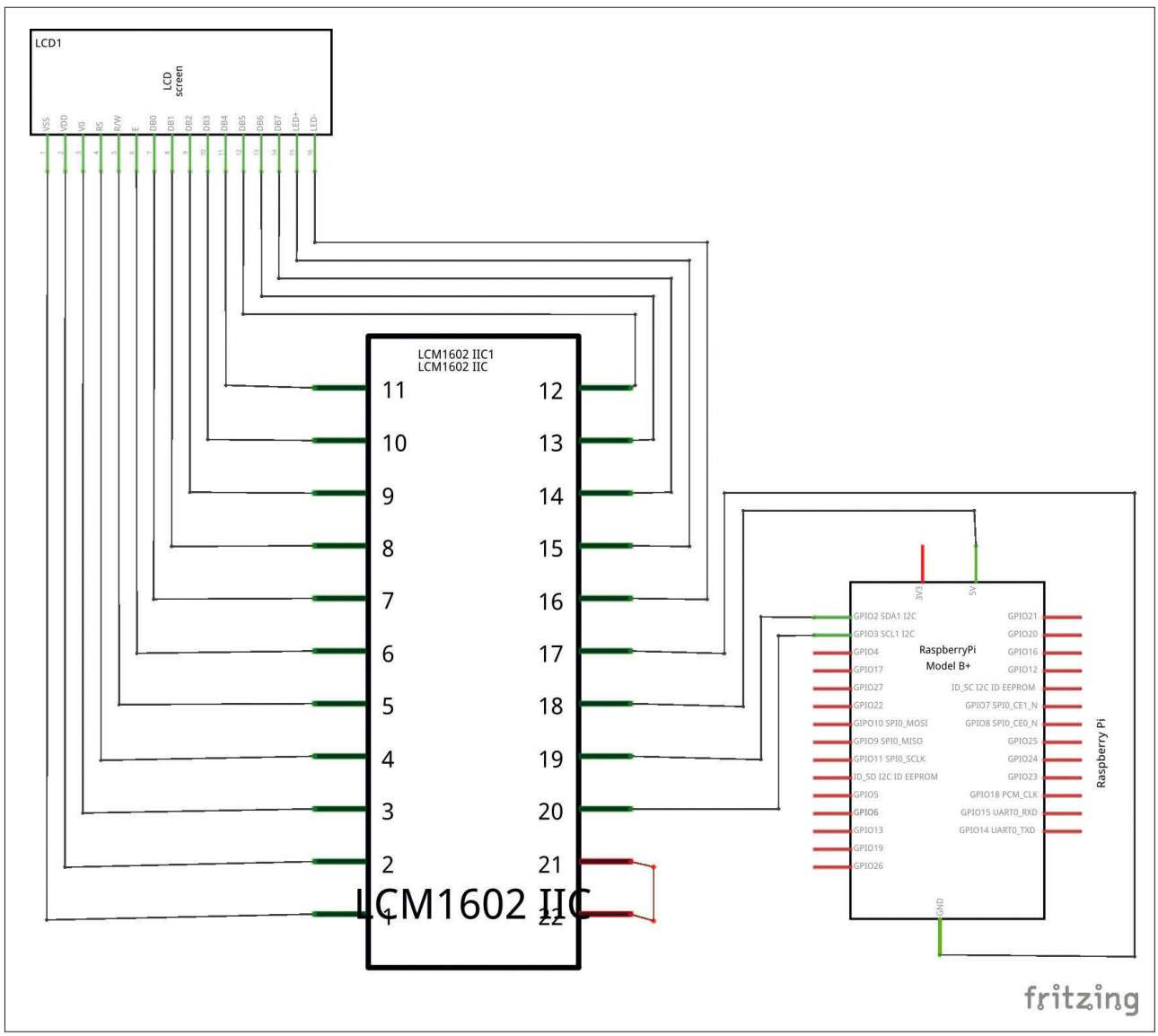
Lecture du cube

```
async def lecture_cube(self) :
 # FACE 1 (devant)
 display.lcd_display_string("Photo 1/6 F", 2)
 rawCapture=PiRGBArray(camera)
```

Le montage



Le schéma



```
camera.capture(rawCapture,format='bgr')
image=rawCapture.array
cv2.imwrite('images/rubiks-side-F.png',image)
await self.retourne_rubik()
```

Regardons les étapes de la lecture de la première face de notre cube (appelée F pour Front) :

1. Affichage LCD
2. Photo à l'aide de PiCamera
3. Écriture de la photo sur disque
4. Mouvement pour obtenir la face suivante

Il faudra répéter l'opération pour les différentes faces : F,U,R,B,D,L (Front, Up, Right, Back, Down, Left)

Décodage et résolution du cube

Le décodage du cube sera assuré par des programmes externes. Le but du décodage étant de trouver les 9 couleurs que comportent les 6 faces de notre cube. Nous utiliserons 2 fonctions :

- **decode_cube** : fonction créant un fichier **tracker.json** à l'aide de photos
- **resolution_cube** : fonction créant une représentation informatique de notre cube à l'aide du fichier **tracker.json**

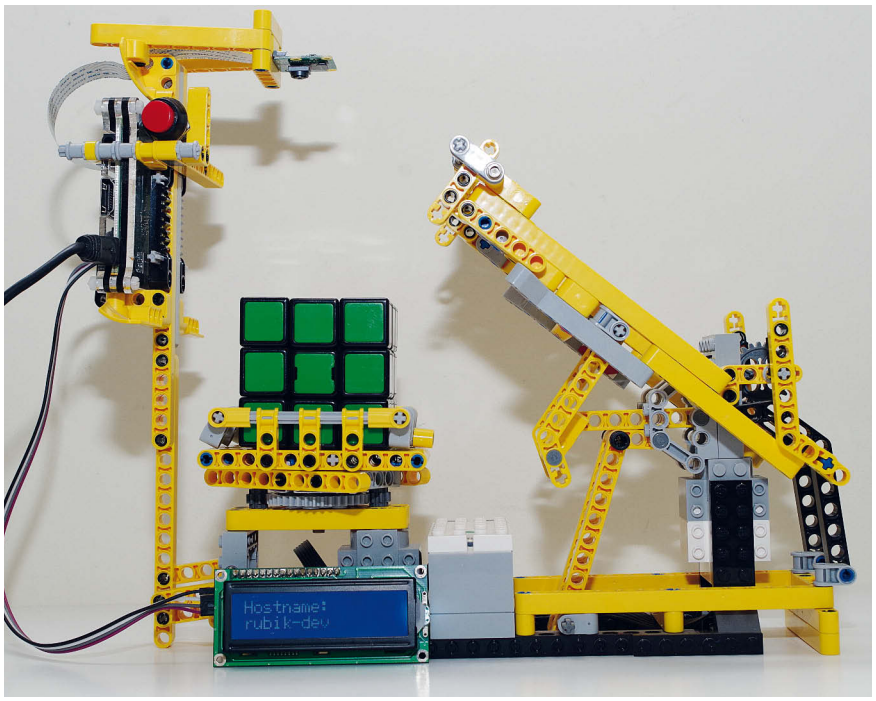
À la fin de la fonction resolution cube on appelle l'algorithme de résolution Kociemba :

```
self.mouvements=kociemba.solve(self.cube)
```

Les mouvements de résolution sont composés d'une lettre F,U,R,B,D,L (Front, Up, Right, Back, Down, Left) ainsi qu'un paramètre optionnel : l'apostrophe pour une rotation en sens inverse, 2 pour une double rotation.

Prise en compte des contraintes techniques

La manière dont a été créé notre robot ne nous permet que les actions B. Il va donc falloir convertir toutes les autres actions pour obtenir des actions B. Transformons nos actions Rubik's cube en actions réalisables par nos fonctions précédemment développées :



Mouvement	Fonctions associées
B	b1B
B'	b4B
B2	b2B
D	cCb1cCcCcC
D'	cCb4cCcCcC
D2	cCb2cCcCcC
F	cCcCb1cCcC
F'	cCcCb4cCcC
F2	cCcCb2cCcC
R	B3cCb1cCcCcCB0
R'	B3cCb4cCcCcCB0
R2	B3cCb2cCcCcCB0
L	B0cCb1cCcCcCB3
L'	B0cCb4cCcCcCB3
L2	B0cCb2cCcCcCB3
U	cCcCcCb1cC
U'	cCcCcCb4cC
U2	cCcCcCb2cC

Optimisations

Si pour résoudre un cube on doit faire les mouvements F puis U (il s'agit ici d'un exemple), cela se traduira à l'aide de nos fonctions : cCcCb1cCcCcCcCcCb1cC, hors cCcCcCcC consiste à retourner 4 fois le cube, donc ne rien faire. On peut alors simplifier cCcCb1cCcCcCcCcCb1cC en cCcCb1cCb1cC. Ainsi, nous réduirons grandement le nombre de mouvements de notre robot.

De plus il est plus rapide de faire 2 rotations et 1 retournement plutôt que 3 retournements on peut donc remplacer : cCcCcC par B5cB5

Conclusion

Nous disposons désormais d'un robot résolvant à notre place un Rubik's Cube. De nombreux tests ont bien sûr été réalisés avant d'aboutir à un résultat. Lors de votre propre réalisation, il faudra très probablement revoir toutes les constantes de positions avant d'obtenir un robot fonctionnel.

Pour aller plus loin, on pourra :

- améliorer la partie robotique pour accélérer les mouvements
- voir si d'autres optimisations de mouvement sont possibles
- ajouter un bouton pour démarrer automatiquement la robotique
- lancer automatiquement le programme au démarrage du Raspberry.

Une petite vidéo du résultat final est disponible si dessous :
https://youtu.be/aUNgpjnsz0o
Enjoy !

Créer des notifications lumineuses avec Home Assistant, wled et Zigbee

Laurent Ellerbach
Principal Engineer Manager Microsoft
laurelle@microsoft.com
https://github.com/Ellerbach

Cela fait des années que j'ai mis en place de l'automatisation à la maison. Depuis un an, je me suis mis à utiliser Home Assistant, https://github.com/home-assistant, en plus de ma solution maison. J'y intègre de nombreuses technologies, du Zwave, Zigbee, beaucoup de développement de capteurs fait maisons à base d'ESP, mais aussi de Raspberry Pi. Dans mes nombreux scénarios, je voulais pouvoir avoir une notification visuelle pour savoir quand et quelle poubelle sortir, quand est-ce qu'un colis a été livré dans la boîte aux lettres et quand est-ce qu'une de mes machines à laver ou sécher le linge a terminé.

Pourquoi une notification visuelle ? Tout simplement parce que je trouve que les notifications vocales ne servent absolument à rien. Vous la manquez, vous avez tout raté. Une notification visuelle reste et est disponible au simple coup d'œil. Et voici à quoi ressemble mon périphérique de notification : **Figure1.** Ici, il y a en vert clair la notification d'une machine terminée, qu'il y a un colis dans la boîte aux lettres en rouge clignotant et qu'il faut que je sorte la poubelle jaune, de recyclage.

Regardons maintenant dans le détail comment créer un tel système de notification.

Figure1

Home Assistant : l'assistant de la maison

Ce que j'aime particulièrement dans Home Assistant, c'est la partie interface graphique pour afficher les informations, créer facilement des graphiques. Hass, pour les intimes, s'installe également très simplement sur un Raspberry Pi ou tourne dans une image docker. De mon côté, je suis passé par un scénario non supporté, mais qui fonctionnement parfaitement bien : le faire fonctionner nativement sur mon Windows Server. Hass est écrit en Python, fonctionne avec des plug-ins écrits en Python également. Ceci dit, pas besoin d'être expert Python pour l'utiliser, la distribution dédiée Raspberry Pi ou l'image docker sont les 2 approches à privilégier. La documentation pour les premiers pas est plutôt propre, pour créer ses propres automatisations, ça paraît un peu compliqué au début dès que l'on veut des scénarios un peu compliqués et puis on finit rapidement par trouver une logique.

Hass intègrent aussi un moteur de règle que nous utiliserons pour les différentes notifications. Sans rentrer dans le détail, au-delà des intégrations qui vont automatiquement détecter certains éléments que vous avez chez vous comme vos assistants vocaux, Sonos, les box internet, imprimantes, il est possible d'intégrer ces propres éléments, mais aussi d'écrire ses propres services basés sur des REST API ou des messages MQTT. Nous utiliserons également les deux.

WLED sur ESP8266 et NeoPixel : créer sa boîte de nuit à la maison

Sans forcément aller jusqu'à créer sa boîte de nuit à la maison, c'est vraiment LE projet à utiliser pour créer ses propres lumières pilotables à base d'ESP8266 : https://github.com/Aircoookie/WLED. Ça fonctionne parfaitement bien, il y a des API, c'est paramétrable, ça s'intègre en MQTT, c'est rapide et c'est fiable !

Les NeoPixel sont ces bandes de LED qui se vendent au mètre pour quelques euros avec des doux noms tels que WS2812B. Le prix augmente suivant la densité de LED. Je ne voulais pas beaucoup de LED, j'ai opté pour une version économique. Cela ne change rien d'en avoir plus, il faudra juste gérer proprement le nombre de LED par groupe. Mesurez la longueur dont vous disposez pour les LED, coupez et souder sur l'ESP8266 D1 Mini WeMos les câbles.

Côté esthétique, impression 3D à partir du modèle trouvé sur Thingiverse : https://www.thingiverse.com/thing:2363013. J'ai imprimé le haut et le bas en blanc, la partie externe qui contient les LED en plastique transparent. L'imprimante 3D que j'utilise est une Prusa i3. J'en suis ravi ! J'ai acheté pour ce projet une bobine de plastique transparent, j'aurais certainement l'occasion de la réutiliser pour d'autres projets !

Suivez les instructions du site pour flasher ou recréez votre propre image. J'ai recréé ma propre image, car j'ai enlevé les éléments dont je n'avais pas besoin et c'est beaucoup plus fun ! Une fois flashé, il propose son propre wifi, connectez-vous

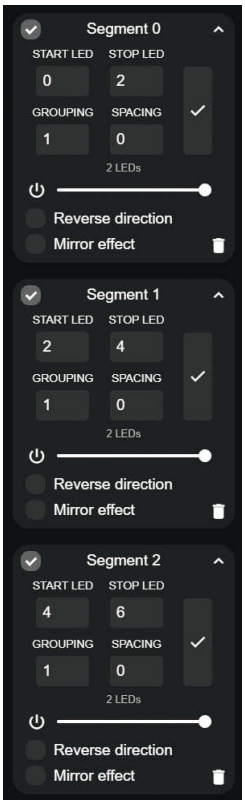


Figure 2

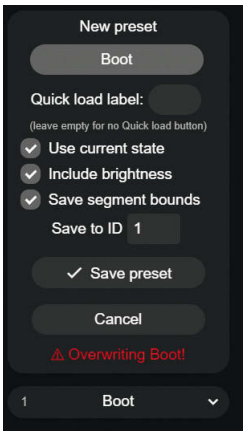


Figure 3

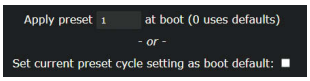


Figure 4

depuis votre PC ou téléphone, puis comme indiqué dans la doc, allez sur http://4.3.2.1 et paramétrez votre propre wifi. Et voilà, vous êtes prêts au paramétrage des groupes.
Comme indiqué dans mon scénario, j'ai besoin de 3 groupes pour les notifications, je vais donc les créer dans l'interface de mon WLED. Réglez bien les couleurs sur noir de façon à ce qu'il n'y ait rien qui s'allume quand vous les bootez : **figure 2**
À ce moment-là, il est vraiment important de bien sauvegarder ce paramétrage dans un preset (ici il existe déjà, n'hésitez pas à le remplacer si vous avez fait une erreur) : **figure 3**
Ensuite, il faut s'assurer que le preset sera bien chargé, il faut aller dans le menu *Config* puis *Led Preference* et bien sélectionner le 1 comme étant celui à appliquer. **figure 4**

Coûts

- ESP8266 D1 Mini WeMos = 2 €
- 1 mètre de NeoPixel = 2 €
- Plastic utilisé = quelques centimes !
- Câble USB et chargeur 5V : tout chargeur de téléphone USB que vous avez

Pour moins de 4 € vous avez un superbe périphérique de notification lumineuse. Il ne reste plus qu'à l'intégrer.

Papa, c'est quelle poubelle qu'il faut sortir aujourd'hui ?

C'était la question que j'ai essayé de résoudre durant des années. En notant sur un mur les dates et périodicités des poubelles, en faisant apprendre cela par cœur à mes enfants, en leur rappelant sur WhatsApp et autres apps. Rien n'y a fait. Ils oubliaient toujours de sortir la poubelle de la bonne couleur. Et quand on sait que la poubelle de verre ne passe qu'une fois tous les 15 jours, il vaut mieux éviter d'oublier de la sortir.
Et c'est donc pour résoudre ce problème que j'ai eu l'idée d'une notification lumineuse ! Hass propose de créer facilement des automatisations, c'est donc ce que je vais utiliser. Le plus simple est de les faire par l'interface graphique, je vais juste partager le code Yaml généré, car cela sera plus simple d'expliquer ce qui se passe avec :

```yaml
- id: '1613383198456'
  alias: Notification poubelle normale on
  description: ''
  trigger:
  - hours: /1
    minute: '1'
    platform: time_pattern
  condition:
  - condition: or
    conditions:
    - after: '16:00:00'
      before: '23:59:00'
      condition: time
      weekday:
      - mon
      - fri
    - after: 00:00:01
      before: '15:59:00'
      condition: time
      weekday:
      - tue
      - sat
  action:
  - data: {}
    service: rest_command.poubelle_normale_on
```

La notion de « trigger », déclencheur dans l'interface en français est ce qui va lancer cette automatisation. Dans mon cas, je la lance toutes les heures et 1 minute passée. Nous verrons plus loin un exemple avec d'autres types de trigger.
Les conditions sont celles qui vont dire si la commande doit être exécutée ou pas. Dans ce cas, je vérifie que je suis soit lundi ou vendredi entre 16h et minuit soit jeudi ou samedi matin. Mes poubelles passant vers 16h, ces deux jours-là.
Si la condition est remplie, la commande rest pour piloter les LED ci-dessous est appelée :

```yaml
poubelle_normale_on:
  url: http://192.168.1.32/json
  method: POST
  payload: '{"on":true,"bri":200,"seg":[{"col":[[100,100,255]]}]}'
  content_type: 'application/json'
```

Ces commandes rest sont à ajouter dans le fichier de configuration global *configuration.yaml* de Hass. Il est possible de les mettre toutes dans un fichier séparé pour faciliter la gestion ensuite. À noter qu'il est important de toujours vérifier que ses fichiers de configuration sont intègres. Une excellente extension VS Code existe pour cela. **Figure 5**
L'UI le permet aussi : **Figure 6**
Malheureusement, il faut redémarrer complètement Hass à chaque changement sur ces API rest, les capteurs ne faisant pas partie de l'intégration, et suivant le nombre d'intégrations que vous avez Hass met un certain temps pour redémarrer. Les automatisations peuvent être rechargées indépendamment. Je supprime les notifications des poubelles systématiquement à minuit passé :

```yaml
- id: '1613384717434'
  alias: Notification poubelle off
  description: ''
  trigger:
  - at: 00:00:30
    platform: time
  condition: []
  action:
  - data: {}
    service: rest_command.poubelle_off
```

La commande rest est similaire à la précédente :

```yaml
poubelle_off:
  url: http://192.168.1.32/json
  method: POST
```

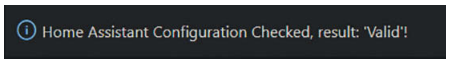


Figure 5

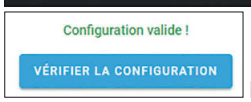


Figure 6

```
payload: '{"on":true,"bri":200,"seg":[{"col":[[0,0,0]],"fx":0}]}'
content_type: 'application/json'
```

L'API est bien documentée, postman est ensuite votre meilleur ami pour tester tout cela. Le *fx* représente ici un des effets qui est le solide. Ma poubelle de verre clignotant vert pour être sûr d'attirer l'attention, il est important de remettre le mode solide par défaut.

Le cas des autres poubelles est similaire, elles passent toutes les semaines, j'ajuste les conditions et les couleurs. La poubelle de verre est plus compliquée, car ne passe qu'une fois toutes les 2 semaines à dates précises. Et ça, par défaut Hass ne sait pas faire. J'utilise une ruse et créer un capteur de type fichier json.

```
#poubelle verre
- platform: file
 name: 'Poubelle verre'
 file_path: 'J:\ServerFolders\HomeAssistant\poubelle_verre.json'
 value_template: '{% for date in value_json.dates -%}{%- if date == now().strftime
("%d/%m/%Y") %}yes{% endif -%}{%- endfor %}'
```

Pour chaque date, si on est à cette date (la veille de quand passe la poubelle, donc des dimanches), j'associe la valeur « yes » au capteur poubelle_verre. Le json doit impérativement se trouver sur 1 seule ligne et ressemble donc à ça :

```
{"dates": ["28/02/2021","14/02/2021","14/03/2021"]}
```

Quant à l'automatisation, le trigger n'est que 3 fois dans la journée (autre façon de faire que les templates de temps) et la condition est que la valeur du capteur soit « yes » :

```
- id: '1613406673648'
 alias: Notification poubelle verre on
 description: ''
 trigger:
 - at: '12:01:00'
  platform: time
 - at: '16:01:00'
  platform: time
 - at: '20:01:00'
  platform: time
 condition:
 - condition: state
  entity_id: sensor.poubelle_verre
  state: 'yes'
 action:
 - data: {}
  service: rest_command.poubelle_verre_on
```

Et l'effet 2 est légèrement clignotant en vert :

```
poubelle_verre_on:
 url: http://192.168.1.32/json
 method: POST
 payload: '{"on":true,"bri":200,"seg":[{"col":[[100,255,100]],"fx":2}]}'
 content_type: 'application/json'
```

Et voilà qui d'un coup d'œil règle le problème des poubelles ! Depuis, les poubelles sont sorties (et rentrées) régulièrement et la poubelle de verre n'est plus oubliée. Même pour moi, c'est un gain de temps.

La boîte aux lettres et Zigbee

Mon autre scénario est de savoir quand quelque chose a été livré dans la boîte aux lettres. J'ai une boîte à deux ouvertures, une côté rue, une côté jardin.

Et pour cela, je me suis dit que mettre un petit capteur d'ouverture sur chacune des portes me permettra de savoir quand un colis est livré et de remettre à zéro la notification quand il est récupéré.

Côté technologie, il me fallait quelque chose de pas cher, facilement intégrable. Je me suis penché sur Zigbee. C'est la technologie qui est utilisée par les lampes Philips Hue par exemple. Sonoff et quelques autres acteurs en Asie l'ont démocratisé récemment. Je me suis donc dit que c'était le moment de me lancer.

Sur les conseils d'un collègue, je suis parti sur une clé Conbee II de Phoscon : https://phoscon.de/conbee2 et j'ai acheté quelques capteurs sur Aliexpress. J'installe le tout très fièrement.

Figure 7

Et là, et bien ma boîte aux lettres est trop loin de mon stick Conbee, je suis obligé de le mettre dehors, sur un Raspberry PI, et en wifi et cela ne fonctionne pas comme je le souhaitais, car le stick n'est pas assez puissant. Au final, je vais réutiliser ce stick, cette fois pour les notifications des machines à laver, j'expliquerai comment plus loin.

Retour à la case réflexion. J'ai essayé en flashant des sticks basés sur un des chips utilisés par Zigbee le CC2530 achetés quelques euros pour jouer. Zigbee est un réseau maillé ou chaque appareil alimenté en permanence permet d'augmenter la taille du réseau. Mais cela a mal fonctionné. Et c'est là que je repense aux kits Sonoff. Ils sont « facilement » flashable avec des firmware sans cloud, connectables à un serveur MQTT local. Le bridge Sonoff a l'avantage de se connecter en wifi et est basés sur un ESP8266 qui est très fiable pour se reconnecter au wifi. Il ne reste plus qu'à commander, patienter le temps de la livraison et se renseigner entre temps.

Figure 7

Tasmota mon amour

Un des systèmes les plus utilisés pour flasher les sonoff et matériels équivalents c'est Tasmota : https://tasmota.github.io/docs/. Il est spécialement fait pour les ESP8266 et les chips Zigbee CC2530 et CC2531. Ensuite, le site indispensable pour savoir ce qui fonctionne avec quoi : https://zigbee.blakadder.com/

Parce que même si Zigbee est censé être un standard, tout n'est pas compatible avec n'importe quoi. En gros, lors de l'achat du matériel il faut voir avec quoi on l'utilisera plus tard. Et les plateformes comme Hass également !

Dans le cas du Conbee II, il y a une intégration directe avec Hass qui rend complètement transparent et simple l'utilisation de périphériques Zigbee compatibles avec le Conbee II. Alors que l'utilisation de Tasmota va s'avérer plus compliqué et qu'il faudra créer les capteurs à la main dans Hass. Mais avant cela, il faut flasher le sonoff. Et là, en suivant scrupuleusement la documentation, impossible de faire fonctionner mon bridge Zigbee flashé avec la version courante Tasmota 9.3.0. Rien ni faisait. J'y ai passé des heures. Désespéré, je propose à mon collègue un call de vendredi « geek » en fin d'après-midi pour essayer de le faire marcher. Lui non plus n'y comprenait rien. Tout semblait se passer correctement à chaque étape, mais le périphérique refuse d'allumer le Zigbee. Au final, il faut flasher l'application minimale. Une fois flashée, il faut y flasher la version tasmota-zbbridge.bin. Suivez toutes les instructions, c'est assez simple, il faut juste avoir de petits câbles et un adaptateur série qui permette d'avoir du 3.3V pour alimenter l'ESP8266. En cas de problème, n'hésitez pas à refaire la manipulation plusieurs fois. Attendez quelques minutes entre chaque étape. **Figure 8**

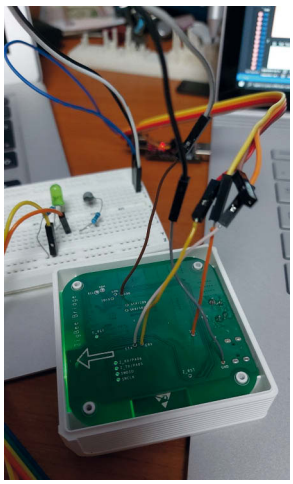


Figure 8

Ensuite le sonoff, une fois flashé, crée son propre wifi (Tasmota-xxxxx), vous vous y connectez, et l'IP est http://192.168.4.1. Cela vous permet de rentrer les informations pour vous connecter à votre propre wifi et vous pouvez ensuite accéder au sonoff flashé à partir de votre réseau.

Après quelques tests, je capte parfaitement bien le wifi du côté de mon portail. Je décide donc d'enlever la coque, récupérer une alim 5V, enlever tout le plastique autour et mettre chaque élément dans un sac antistatique, le tout dans une boîte étanche avec l'arrivée électrique et le dispatch de mon portail électrique. Le tout fonctionne parfaitement bien, les capteurs sont bien reconnus, la portée est excellente d'ailleurs. **Figure 9**

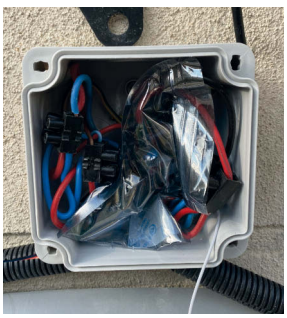


Figure 9

La récupération des données se fait sur le bus MQTT. Tasmota bavarde beaucoup sur le bus, le paramétrage est simple. Côté documentation, peut mieux faire. Mais il est simple de comprendre comment récupérer ces informations.

MQTT et Hass

Me voilà donc avec des données de capteur que je peux exploiter. Dans Hass, je déclare les capteurs en tant que binary_sensor :

```
- platform: mqtt
  unique_id: bal_externe
  name: "BAL externe"
  state_topic: "tele/tasmota/SENSOR"
  value_template: "{{ value_json['ZbReceived']['0xA8A2'].Contact }}"
  payload_on: "1"
  payload_off: "0"
  device_class: "opening"
- platform: mqtt
  unique_id: bal_interne
  name: "BAL interne"
  state_topic: "tele/tasmota/SENSOR"
  value_template: "{{ value_json['ZbReceived']['0x8C95'].Contact }}"
  payload_on: "1"
  payload_off: "0"
  device_class: "opening"
```

Chaque capteur a un nom unique qui par défaut est le début de son adresse MAC. Le payload MQTT envoyé par le sonoff est en JSON. Il est un peu bavard et ce qui va nous intéresser ici c'est uniquement le statut du contact ouvert ou fermé.

Côté de l'automatisation, c'est simple et rapide aussi :

```
- id: '1617029460225'
  alias: Notification BAL on
  description: ''
  trigger:
  - entity_id: binary_sensor.bal_externe
    for: '0'
    from: 'off'
    platform: state
    to: 'on'
  condition: []
  action:
  - data: {}
    service: rest_command.mailbox_on
  - data:
      message: Livraison boite aux lettres {{now().strftime("%Y-%m-%d %H:%M")}}
      title: Livraison boite aux lettres {{now().strftime("%Y-%m-%d %H:%M")}}
    service: notify.laurent
- id: '1617029573467'
  alias: Notification BAL off
  description: ''
  trigger:
  - entity_id: binary_sensor.bal_interne
    for: '0'
    from: 'off'
    platform: state
    to: 'on'
  condition: []
  action:
  - data: {}
    service: rest_command.mailbox_off
```

Le trigger se fait simplement sur le passage de *off* (fermé) à *on* (ouvert) de la porte externe. Il n'y a pas de condition ni de minimum de temps à respecter. Je fais appelle ici à 2 actions, une notification REST sur le WLED, exactement comme précédemment :

```
mailbox_on:
  url: http://192.168.1.32/json
  method: POST
  payload: '{"on":true,"bri":200,"seg":[{},{"col":[[255,100,100]],"fx":1}]}'
  content_type: 'application/json'
```

Ici, j'active seulement le deuxième groupe de LED. Et en plus je m'envoie un mail. Hass possède aussi la faculté de pouvoir notifier par email.
La réinitialisation de la notification se fait par l'ouverture de la porte intérieur, sans condition non plus et l'appelle au service REST cette fois-ci avec les valeurs 0,0,0 pour la couleur noire.

Et mes machines dans tout cela ?

J'ai maintenant les notifications pour mes poubelles, ma boîte aux lettres, il me reste mes machines à laver. Mon idée est de pouvoir détecter quand les machines s'arrêtent. Il me faut donc mesurer la consommation électrique et pouvoir déterminer quand cette consommation passe sous un certain seuil. Je me dirige donc vers des prises avec mesure de consommation BlitWolf SHP13 qui supportent 16 ampères maximum. Ce qui est a priori largement suffisant pour mes machines. Cette fois-ci je décide de réutiliser la Conbee II et l'installe sur mon serveur Windows en natif. Pas très compliqué mis à part que le certificat des drivers avait expiré, ce qui m'a obligé à changer l'heure de mon serveur le temps d'installer les drivers. J'utilise ensuite NSSM (Non-Sucking Service Manager) https://nssm.cc/ pour créer un service Windows qui se lance au démarrage. NSSM est pratique dans ce type de cas. Conbee II propose en natif de tourner sur un PI ou en service Docker, mais il a besoin de l'accès port série. Sous Windows, c'est l'application graphique quoi doit être lancée pour que se lance le reste.
L'association des SHP13 se fait rapidement et simplement, il faut les ajouter en tant que « light ». Il y a une intégration Hass et cela permet d'avoir un accès simple et direct à tous les périphériques sans avoir besoin de passer par la case ymal. **Figure 10**
De là, je crée 2 automatisations, une pour chaque machine :

```
- id: '1617208057524'
alias: Notification lave-linge terminé on
description: ''
trigger:
- below: '7'
  entity_id: sensor.power_lave_linge
  for: 00:05:00
  platform: numeric_state
condition: []
action:
- data: {}
  service: rest_command.machine_terminee_on
```

Cette fois-ci le déclenchement se fait lorsque la consommation passe sous la barre des 7 watts pendant 5 minutes. Et comme pour tout le reste, j'appelle une commande REST pour les LED.
La question qui se pose maintenant est pour la remise à zéro de la notification. Plusieurs options s'offrent à moi : un capteur sur la porte de chaque machine, similaire à la boîte aux lettres. Un simple bouton (switch) qu'il suffit de presser pour remettre à zéro les notifications. J'ai opté pour ce second choix de façon à pouvoir jouer avec plus d'éléments Zigbee.

Une fois ajouté à Conbee II, le switch est utilisable en tant que déclencheur :

```
- id: '1617207891125'
alias: Notification machine terminée off
description: ''
trigger:
- device_id: 2f7e1bb456064024b917684b0078ba9e
  domain: deconz
  platform: device
  subtype: turn_on
  type: remote_button_short_press
condition: []
action:
- data: {}
  service: rest_command.machine_terminee_off
```

Les boutons ont plusieurs fonctions, du simple clic au clic long permettant des scénarios plus avancés.

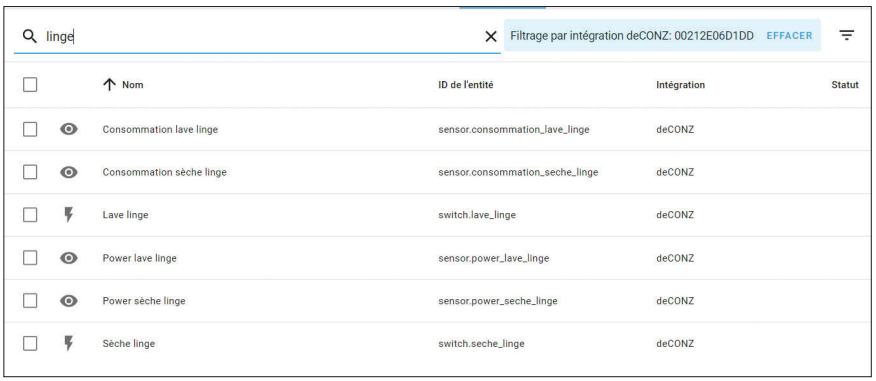
Coût matériel Zigbee

- Phoscon Combee II : environ 30€ sur Amazon
- Contacteur : 7€ par 5, soit 35€
- Prises intelligentes BlitWolf SHP13 avec mesure de la consommation Zigbee : 12€ par prise
- Kit Sonoff Zigbee : 50€ sur Aliexpress
- Total : une bonne centaine d'euros, mais avec des éléments indispensables pour continuer d'étendre le réseau Zigbee.

Que la lumière soit

Mon service de notification fonctionne parfaitement me permettant d'avoir d'un simple coup d'œil les informations concernant mes poubelles, mes livraisons et mes machines de linge. Cela m'a vraiment beaucoup apporté en confort et sérénité. Tout me monde semble d'ailleurs content à la maison. Le système se trouve dans l'entrée, mais aussi dans mon bureau. Les WLED pouvant être associées les unes aux autres et synchronisées en elle. Le coût de périphérique de notification en tant que tel est de moins de 5 €. Le coût des différents périphériques Zigbee de l'ordre de 100 €, mais qui peut être optimisé si une seule passerelle est suffisante. Un projet vraiment amusant qui ouvre la porte à d'autres notifications. Il est facile d'ajouter des capteurs Zigbee. Ils sont à bas coûts et une fois bien maîtriser les différentes passerelles notamment celles basées sur Tasmota, seule votre imagination va limiter ce que vous pouvez en faire.

Figure 10



Piège ou pas ?

LES PROCHAINS NUMÉROS

Programmez! n°248

Dossier quantique
Node/JS

Disponible le 3 septembre 2021

Hors Série #4 ÉTÉ

100 % JAVA

Disponible le 9 juillet 2021

PROGRAMMEZ!
Programmez! n°247
Juillet - Août 2021

Directeur de la publication & rédacteur en chef
François Tonic
ftonic@programmez.com

Contacter la rédaction
redaction@programmez.com

Ont collaboré à ce numéro
La rédaction de ZDnet

Les contributeurs techniques
H. Extrat, Y. Lathuilière, F. Dubois, G. Bersegeay, M. Thlithi, B. Sakote, T. Delattre, F. Chazal, J. Lepine, L. Lefèvre, H. Souissi, O. Bersot, S. Colas, L. Ellerbach

Couverture
Fond page : © vectorplusb
Picto jeux : © Tetiana Lazunova

Maquette
Pierre Sandré

Marketing – promotion des ventes
Agence BOCONSEIL - Analyse Media Etude
Directeur : Otto BORSCHA
oborscha@boconseilame.fr
Responsable titre : Terry MATTARD
Téléphone : 09 67 32 09 34

Publicité
Nefer-IT
Tél. : 09 86 73 61 08
ftonic@programmez.com

Impression
SIB Imprimerie, France

Dépôt légal
A parution

Commission paritaire
1225K78366

ISSN
1627-0908

Abonnement
Abonnement (tarifs France) : 49 € pour 1 an, 79 € pour 2 ans. Etudiants : 39 €. Europe et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € - Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 €.
Autres pays : consultez les tarifs sur www.programmez.com.
Pour toute question sur l'abonnement :
abonnements@programmez.com

Abonnement PDF
monde entier : 39 € pour 1 an.
Accès aux archives : 19 €.

Nefer-IT
57 rue de Gisors, 95300 Pontoise France
redaction@programmez.com
Tél. : 09 86 73 61 08

MARDI 14 SEPTEMBRE
À MARSEILLE

WAX

CONFÉRENCE 100% REX

EN PLEIN AIR

PRÉSENTIEL & STREAMING

CLOUD | DEVOPS | SÉCURITÉ

EARLY BIRDS

WWW.WAXCONF.FR

Evénement imaginé et organisé par Les Filles & Les Garçons DE LA TECH

NOUVELLE VERSION • WINDEV 26, AGL DEVOPS LOW-CODE

DÉVELOPPEZ UNE FOIS, PUIS COMPILEZ AU CHOIX POUR
WINDOWS, LINUX, MAC, INTERNET, IOS, ANDROID

DÉVELOPPEZ 10 FOIS PLUS VITE

N°1 EN FRANCE

926 NOUVEAUTÉS INCONTOURNABLES

• DEMANDEZ LE DOSSIER COMPLET GRATUIT • TÉLÉCHARGEZ LA VERSION EXPRESS GRATUITE

PCSOFT.FR

TÉL: 04 67 032 032

Logiciel professionnel. Document non contractuel. WINDEV permet de compiler pour Windows et Linux. WEBDEV compile pour Internet. WINDEV Mobile compile pour Mac, iOS et Android.