JAVA / ZENBOT / ARDUINO / DEVEXTREME / SÉCURITÉ / ALEXA

[Programmez!]

Le magazine des développeurs

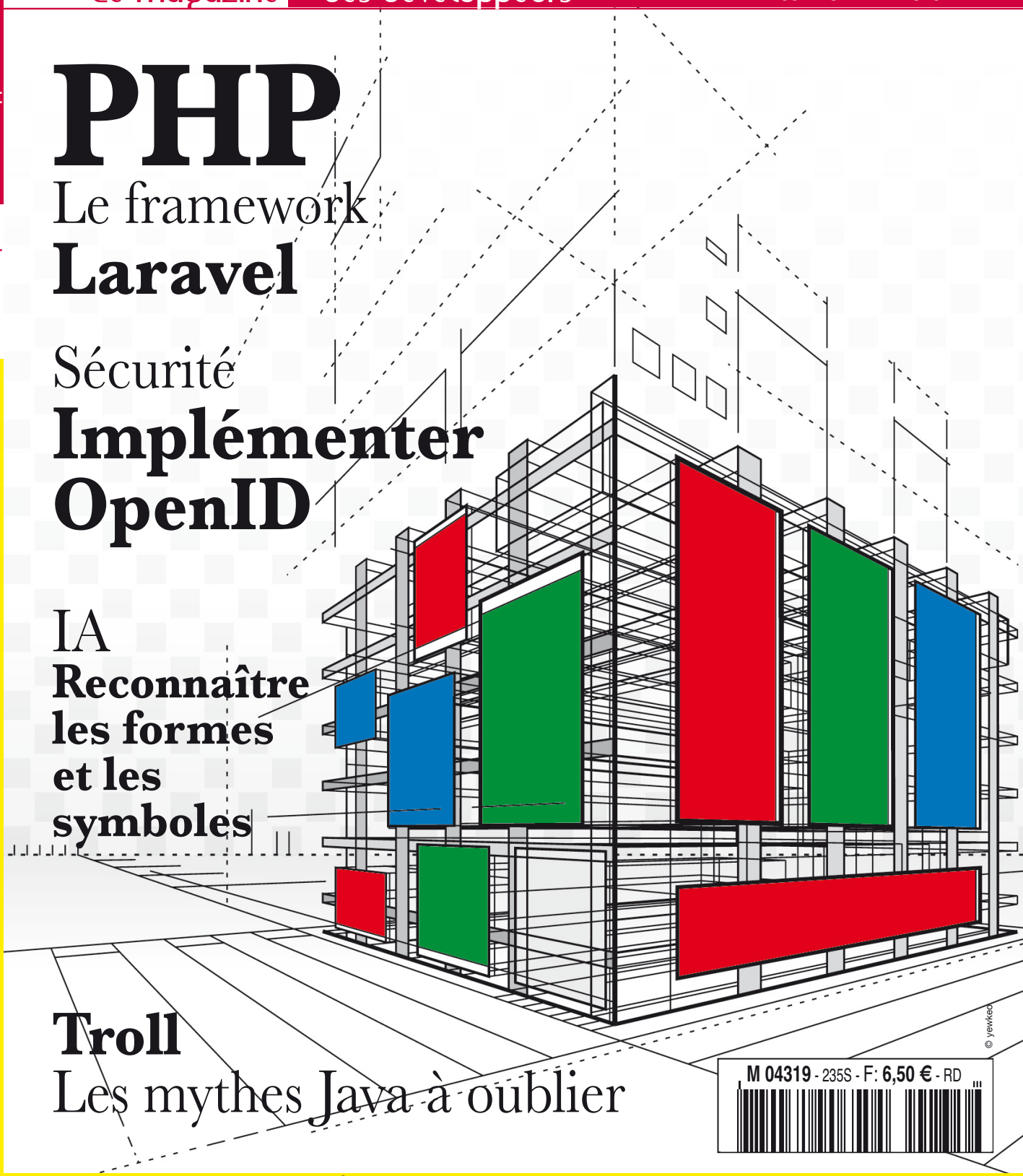
235 DÉCEMBRE 2019

PHP
Le framework
Laravel

Sécurité
Implémenter OpenID

IA
Reconnaître les formes et les symboles

Troll
Les mythes Java à oublier



© yewkeo

M 04319 - 235S - F: 6,50 € - RD

LE SEUL MAGAZINE ÉCRIT PAR ET POUR LES DEVELOPPEURS

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

JAVA / ARDUINO / ALEXA / PHP / C++ / PYTHON

Olymp

La formation nouvelle génération

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

Edité par

IdeaStudio

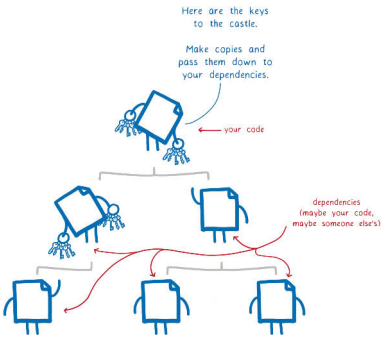
Faut bien s'occuper !

Tu ne sais pas quoi faire ? Trouve quelques camarades de code, tu crées une alliance sympa qui va faire rêver. Et là, tu annonces que tu vas réinventer un truc qui existe déjà et qui fonctionne pas si mal. Mais toi, tu vas créer un truc mieux que l'actuel. Un conseil : change au moins le logo, les dessins et quelques mots du texte de présentation.

Bien entendu, toute similitude avec la toute nouvelle, et top géniale, bytecode alliance serait pure ~~évidence~~ coïncidence. Cette alliance a de quoi te plaire : une communauté open source dédiée à la création de nouvelles fondations sécurisées, utilisant des standards comme WebAssembly et WebAssembly System Interface. Le tout pour exécuter des codes inconnus, non certifiés, dans un joli environnement sécurisé, capable de fonctionner partout. Ce n'est pas comme si WebAssembly faisait déjà un peu ça, que .Net Core, Rust, Java, C++ n'existaient pas…

Mais finalement, WebAssembly n'a jamais réussi à devenir un standard de développement car il se heurte à .Net Core, Kotlin, Swift, etc. De là à y voir un nouveau concurrent sérieux, attendons un peu. On trouve dans les soutiens, Mozilla, Intel et Red Hat. Mais on constate que les grands fournisseurs technologiques sont absents : Microsoft, Google, Apple, IBM, Facebook, etc. Mais c'est pas grave car l'alliance est là pour interdire les méchants codes inconnus, malveillants de s'exécuter. La preuve que la menace est là : +47 % de vulnérabilités pour npm en 2018, et Maven Central et PHP Packagist sont à +27 et 56 % ! Ouf j'suis totalement secure !

Heureusement, de jolis dessins vont te convaincre que la bytecode alliance est l'avenir. Il est pas beau mon dessin ?



Pas convaincu(e) ? Nanoprocess, isolation partout, sandbox, isolation de la mémoire. Toujours pas ? Là franchement tu m'aides pas beaucoup. Voici mon argument ultime : 42. Non ? Toujours pas ?
Bah, je balance mon dernier argument :

Franchement, si la bytecode alliance ne te fait pas kiffer, je vais monter un élevage de balrogs. C'est tellement affectueux.

François Tonic - ftonic@programmez.com

SOMMAIRE

Dans le prochain numéro !

Programmez! #236, dès le 3 janvier 2020

Développement mobile :

les plateformes, app native vs app hybride ? Focus sur Flutter

Carrière :

pourquoi partir au Canada ?

Numéro exceptionnel buildé avec la communauté des Duchess France

Bercy veut garder un œil sur vos réseaux sociaux

Le projet de loi de finances 2020 contient un article qui entend permettre aux agents du ministère des Finances d'étudier les publications des contribuables sur les réseaux sociaux afin de détecter d'éventuelles fraudes. Bien évidemment, ce genre de disposition n'est pas passée inaperçue : la presse s'est interrogée sur la question, la CNIL s'est emparée du dossier et a émis plusieurs mises en garde, ce qui a conduit les législateurs à introduire plusieurs limitations à l'article en question. Il faudra donc réfléchir à deux fois avant de poster des photos de votre nouveau yacht sur Facebook (ou Instagram).

Xerox lorgne sur HP

Le fabricant d'imprimantes Xerox a annoncé son intention de prendre le contrôle de HP Inc., l'entité issue de la scission de HP en 2015 et chargée de poursuivre son activité dans le secteur de l'impression et des PC. Les velléités de Xerox ont été révélées par le New York Times, mais HP Inc. a rejeté l'offre de Xerox : la société proposait un rachat à 22 dollars par action, ce qui aurait valorisé la société à environ 33,5 milliards de dollars. Trop peu selon le conseil d'administration, qui laisse néanmoins ouverte la possibilité d'une consolidation entre les deux groupes.

Web : Chrome veut signaler les traînards

Avec Chrome, Google est devenu un acteur majeur du web. Sa dernière lubie : signaler à ses utilisateurs les sites qui mettent trop de temps à charger sur leur navigateur. Un « name and shame » qui risque de faire grincer des dents, surtout quand on voit la façon dont Google met à profit sa position dominante afin de pousser ses technologies au sein des standards du web. Pour l'instant, Chrome ne donne pas beaucoup de détails sur les critères précis qui seront pris en considération, mais promet de nouvelles informations au fur et à mesure de l'avancement de son projet. À surveiller si vous voulez éviter de voir votre site brocardé par le premier des navigateurs.

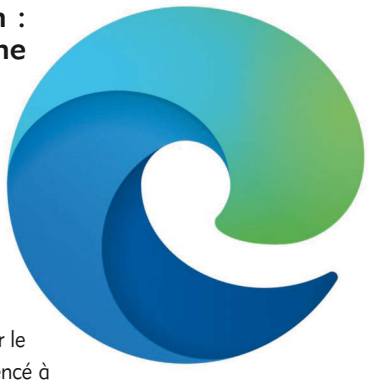
Bluekeep : la faille Windows commence à être exploitée

Au mois de mai, Microsoft alertait sur une faille de sécurité RDP baptisée Bluekeep et invitait les administrateurs à corriger rapidement celle-ci. L'éditeur craignait en effet que l'utilisation de cette faille par les cybercriminels ne conduise à une nouvelle épidémie de logiciels malveillants comparables à ce qu'avait pu provoquer la diffusion de l'exploit EternalBlue. Début novembre, les premières attaques utilisant cette faille ont été détectées : un groupe d'attaquants l'utilise pour infecter des systèmes vulnérables et y installer un logiciel clandestin de minage de cryptomonnaie. Ce n'est pas vraiment le scénario catastrophe que redoutait Microsoft, mais c'est le signe que les attaquants commencent à s'emparer de la faille et qu'il vaudrait mieux songer sérieusement à patcher avant qu'un autre groupe ne fasse plus de dégâts.

Pour Snowden, le RGPD ne suffit pas

Edward Snowden, le lanceur d'alerte à l'origine des fuites de données ayant mis en lumière la collecte de données opérée par la NSA, a récemment donné son avis sur la mise en place en Europe du Règlement General de Protection des Données. Pour l'ancien informaticien de l'agence, le texte part d'une bonne intention, mais ne suffira pas à mettre un coup d'arrêt à la collecte massive opérée par les agences autant que par les entreprises : pour lui, nous ne faisons que légitimer un état de fait, celui d'une société de la surveillance où les données personnelles sont devenues une ressource précieuse pour les entreprises.

Edge sur Chromium : janvier 2020 en ligne de mire



La nouvelle mouture de Edge, basée sur Chromium, sera disponible pour tout le monde à partir du 15 janvier 2020. Microsoft avait annoncé son intention de proposer une nouvelle version de son navigateur Edge s'appuyant sur le moteur Chromium et a commencé à proposer plusieurs versions beta de ce navigateur ces derniers mois. L'objectif pour Microsoft est de proposer un navigateur plus en phase avec le web actuel (comprendre : avec Chrome) et au passage bénéficier de l'écosystème des webextensions, utilisées par Chrome et Firefox.

Github veut cryogéniser l'open source

Github tenait sa conférence Github Universe le 13 et 14 novembre et présentait les nouveautés de sa plateforme. Mais bon, ce qui a vraiment marqué les esprits, c'est le dernier plan un peu fou du Github Archive Program. Github entend en effet créer une grande archive du code open source, conçue pour conserver les données pendant plus de 1 000 ans. Ces données seront stockées en Antarctique, sur l'archipel de Svalbard, et répliquées chez des partenaires multiples afin de s'assurer qu'elles ne soient pas perdues avec le temps. Github prévoit de stocker une première archive de l'ensemble des dépôts Github publics actifs au 2 février 2020. Si à cette date, vous avez laissé des bugs traîner dans votre projet open source, sachez donc que l'humanité pourrait s'en souvenir dans un millénaire.

FIRST EUROPEAN FREE & OPEN SOURCE EVENT

opensourcesummit.paris

#OSSPARIS19

PARIS REGION SMART WEEKS

La FRENCH TECH

PARIS OPEN SOURCE SUMMIT

5e ÉDITION

10 & 11 DECEMBRE 2019

Show and Congress
Dock Pullman

Pour toute demande d'informations complémentaires :
Email : cjardon@weyou-group.com – Tel : 01 41 18 60 52

sponsor diamond

Inria

sponsors platinium

alter way

SMILE I.T IS OPEN

sponsors gold

actency

BlueMind Messagerie & espaces collaboratifs

jahia

Microsoft

Red Hat

SensioLabs Créateur de Symfony

Vorteks

sponsors silver

AXELTI ZABBIX

Capgemini

FACTOR X

Vates

un événement

Systematic Paris Region Deep Tech Ecosystem

weyou group

en partenariat avec

Région îledeFrance

CNLL

Décembre

5 : conférence agilité / Paris

Zenika organise une journée de conférences et d'ateliers autour de l'Agilité sur le thème des interactions et des émotions, le 5 décembre 2019 à la cité de la Roquette à Paris.
Si toi aussi tu penses qu'on n'est pas des robots, inscris-toi : https://zenikagileday.eventbrite.fr

5 & 6 : La nuit de l'info 2019

La nuit de l'informatique revient. Pour rappel, il s'agit d'une grande compétition nationale pour les étudiants, enseignants et entreprises pour développer un projet durant la nuit. Les équipes doivent choisir le défi à relever ! +4 000 participants ont tenté de ne pas dormir en 2018…
Site : https://www.nuitdelinfo.com

10 & 11 : Paris Open Source Summit / Aubervilliers

Le Paris Open Source Summit est le premier évènement en Europe sur l'open source, les logiciels libres et l'innovation ouverte. Sommet international de conférences, salon business et rendez-vous communautaires, OSS Paris met en lumière le rôle moteur des technologies open source dans les transformations numériques actuelles et à venir.
En 2019, OSS Paris devient le rendez-vous TECH de l'écosystème open source, qui réunit pendant 2 jours les contributeurs, décideurs et utilisateurs de briques technologiques et de solutions open source.
Site : https://www.opensourcesummit.paris

13 / Agile Tour Strasbourg / Strasbourg

Dans la lignée des éditions précédentes, l'Agile Tour continue à rassembler une communauté d'agilistes large et variée tout en ayant un focus important sur la communauté et les intervenants locaux. Alors si vous souhaitez découvrir, apprendre, profiter des retours d'expérience de vos pairs, jouer et ouvrir vos horizons en matière d'agilité, inscrivez-vous : https://www.meetup.com/fr-FR/ElsassJUG/events/265130586/

17 décembre

Meetup Programmez!

spécial langage GO !

Venez découvrir le langage Go, les dernières versions, les fonctions clés !

A PARTIR DE 18H30.

Où : Infeeny 5 rue d'Uzès Paris
Métro : station Grands Boulevards (lignes 8 & 9)

Inscription : programmez.com

2020

Janvier

14 : MongoDB Paris / Paris

Venez rencontrer MongoDB durant une journée dédiée à la base de données et à l'écosystème. La plus grande conférence MongoDB en France offre plein d'occasions d'apprendre et rencontrer les meilleurs experts.

Site : https://www.mongodb.com/local/paris

22-25 : SnowCamp / Grenoble

SnowCamp est une grande conférence rassemblant dévs, ops et architectes. On y trouve une journée de formation, 2 jours de conférences et une journée d'échanges sur les pistes.
Site : http://snowcamp.io/fr/

31 : Touraine Tech / Tours

Nouvelle édition de Touraine Tech. Au programme : 18 sessions, des ateliers. On y parlera design, UI, mobile, big data, cloud, jeux, IoT, architectures !
Site : https://touraine.tech

Février

1 & 2 : FOSDEM 2020 / Bruxelles

L'évènement incontournable des développeurs open source. C'est l'occasion de découvrir des projets et surtout d'autres développeurs. On échange, on discute, on oppose les arguments.
Site : https://fosdem.org/2019/

3 : dotSwift / Paris

La conférence Swift revient à Paris. Avec les dernières évolutions du langage, le nouveau framework UI, il y aura de nombreuses choses à découvrir.

14-16 : DevFest Paris / Paris

La DevFest revient à Paris mi-février. L'événement promet de superbes sessions, des animations, et beaucoup de technos et de rencontres ! 36 sessions sont prévues !
Site : https://devfest.gdgparis.com

Avril

15-17 : Devoxx France / Paris
29 & 30 : MixIT / Lyon

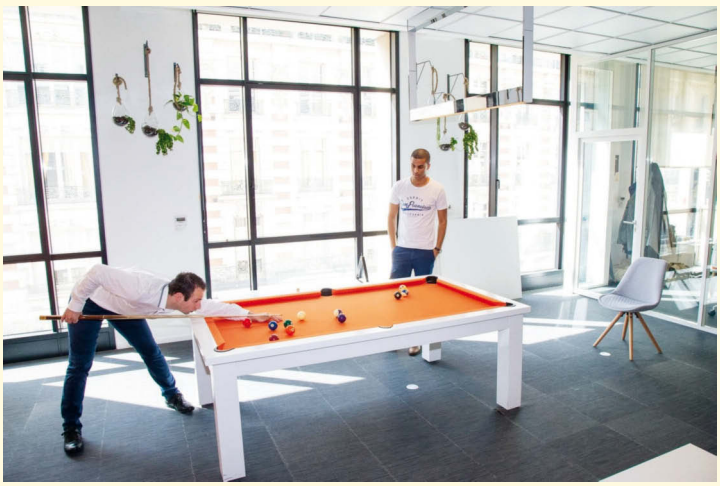
Merci à Aurélie Vache pour la liste 2019/2020, consultable sur son GitHub :
https://github.com/scraly/developers-conferences-agenda/blob/master/README.md

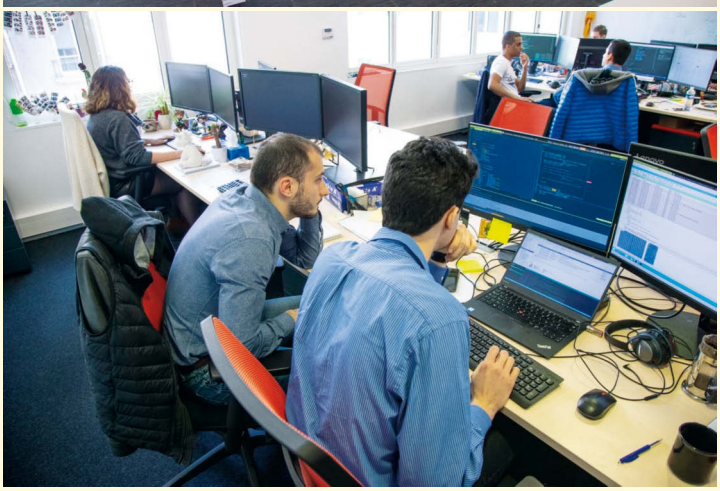
Comment recruter et garder ses développeurs ?

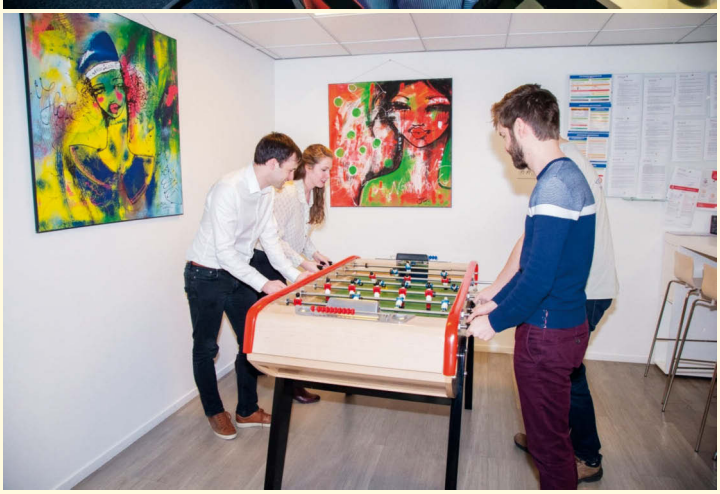
Les entreprises peinent aujourd'hui à recruter et fidéliser leurs développeurs. Delphine Schoffler, Directrice des Ressources Humaines chez WALLIX nous livre son regard et son expérience auprès des développeurs.

WALLIX
CYBERSECURITY SIMPLIFIED

C'est un fait, le monde informatique et notamment la cybersécurité manque cruellement de développeurs. La problématique est aussi simple que cela *« Les développeurs sont très sollicités et reçoivent des offres attractives de tout type d'entreprises, secteurs, projets »* évoque Delphine Schoffler. Son objectif principal étant de « sortir du lot » et de leur donner envie de rester : *« S'ils se sentent bien dans l'entreprise, ils resteront car ils sont foncièrement fidèles et s'investissent dans le long terme »*. Cet enjeu est énorme pour WALLIX, qui cherche à attirer des dizaines de développeurs dans le cadre de son plan Ambition 21 et sa forte croissance.
La méthodologie de WALLIX étant tout d'abord de comprendre les attentes des développeurs en entretien en travaillant sur leurs réponses *« qui s'articulent particulièrement autours du contenu et des challenges techniques des projets, sur l'ambiance et l'environnement au travail ainsi que le salaire et les avantages proposés. »*.







Pour convaincre, WALLIX associe les opérationnels aux entretiens de recrutements pour l'explication des contenus techniques, des challenges mais aussi pour qu'ils aient un aperçu des collègues avec qui ils travailleront. D'autre part, *« nous mettons en avant nos atouts, qu'ils soient tangibles : locaux, billard, bowling, séminaires internes, etc. ou intangibles liés à nos valeurs intrinsèques et sur lesquelles nous essayons de travailler au quotidien (ndlr : esprit d'équipe, audace, engagement). »*. L'importance d'une forte culture d'entreprise semble être la clé pour attirer et garder ce rare spécimen qu'est le développeur en 2019… Sujet sur lequel l'expert du PAM (Privileged Access Management) a plusieurs projets internes en cours afin d'améliorer la qualité de vie au travail et le bien-être de ses salariés.

Challenge, bien-être, dynamisme et innovation seraient donc les mots clés d'une réponse à la problématique initiale. Un sujet que l'entreprise française de Cybersécurité semble avoir compris : *« c'est faire le choix d'une structure à taille humaine, dynamique en croissance, avec de fortes ambitions où chacun a un rôle à jouer et peut faire la différence ! »*
C'est aussi, semble-t-il, parier sur le potentiel d'une *« scale-up »*, avec une approche RH qui vise en retour, à valoriser le potentiel de ses collaborateurs.

La proposition se veut claire pour la Directrice des Ressources Humaines : WALLIX veut avoir les moyens de ses ambitions et atteindre ses objectifs. Pour cela, ils travaillent sur les problématiques et attentes personnelles de ses collaborateurs pour attirer, fidéliser et construire avec eux une belle histoire *« C'est avant tout une aventure humaine et technique. Notre histoire, ses exigences techniques et notre investissement dans l'innovation doivent attirer et retenir les développeurs et les talents de manière générale. »*

Rendez-vous sur
www.wallix.com/carrières

Concours Amazon Alexa : le gagnant est…

Programmez ! et Amazon Alexa avaient organisé un concours de skills. Les résultats ont été annoncés durant la DevCon spéciale Alexa à 42. Reynald, et sa skill monVoyage, est arrivé 1er.

2

Peux-tu te présenter en quelques mots ?

Je m'appelle Reynald Lechapt et j'ai 30 ans. Je suis venu au code en faisant l'école 42. Je travaille actuellement chez Havas Voyages en tant que responsable API. Depuis mes débuts en code je me suis toujours intéressé au conversationnel. J'ai commencé avec un projet : "Chatboté" qui sélectionnait les logements (apparts, hôtels…) pour les courts séjours en fonction du mood de l'utilisateur. Puis j'ai travaillé chez Hello Marcel, l'assistant qui déniche les meilleures idées de sorties en famille.

Tu développes des skills pour Amazon Alexa depuis combien de temps ? Quelles sont les forces de la plateforme ?

J'ai découvert la plateforme Alexa lors de l'Hackathon "Hack The Echo" en 2015. Alexa n'était pas encore sortie en France. La skill monVoyage est la première skill que j'ai développée et publiée.

Pour moi, le vrai plus de la plateforme par rapport à la concurrence c'est la facilité de mise en action. Tout est fait pour faciliter la tâche des développeurs et les aider à commencer :

- Alexa hosted skills se charge d'héberger le code de la skill pour nous + un espace mémoire mis à disposition pour persister les infos entre les sessions sans avoir besoin de configurer une bdd + Amazon S3 pour stocker les fichiers statiques. Tout ça clé en main. C'est du prêt à coder.
- De nombreux exemples pratiques avec le code correspondant sont dispos sur le repo Github Alexa. Ça m'a été particulièrement utile pour implémenter rapidement la demande des permissions d'accès aux infos perso de l'utilisateur (email et numéro de téléphone) depuis la skill.
- Les nombreux évènements organisés par Amazon (live ou webex) pour monter en compétence les développeurs.

Tu as participé au concours Amazon - Programmez!, quelles ont été tes motivations ? Comment l'idée de monVoyage t'es venue ?

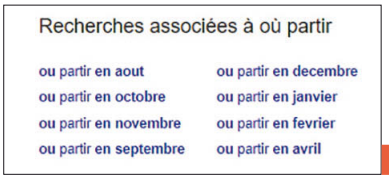
J'ai commencé à développer la skill monVoyage à l'été 2019 sans avoir idée du concours organisé par Programmez! Quand j'ai pris connaissance du concours, je me suis dépêché de sortir la skill. Je voyais en ce concours l'opportunité d'avoir les retours de la communauté des développeurs Alexa et early adopters. Récolter du feedback était ma priorité. Travaillant dans le monde du voyage, ça fait un moment que je réfléchis à comment utiliser la voix dans cette industrie.

Mes premiers constats avec les usages de la voix en général étaient qu'il fallait un use case qui permettait d'apporter une réponse rapide à une question simple de l'utilisateur. C'est pourquoi j'ai choisi de construire la skill autour de la question "Où partir ..?" en essayant d'apporter une réponse courte avec des éléments suffisamment inspirants.

Comment as-tu construit le scénario de ta skill ? Quelles ont été les difficultés techniques ?

La question "Où partir" a l'avantage de définir précisément la fonctionnalité de la skill, mais aussi d'offrir un large terrain de jeu du fait des variantes possibles de la question : "où partir au ski à Noël ?", "où partir au soleil en janvier", "où partir en weekend en amoureux ?".

On peut vite se perdre à vouloir traiter toutes ces possibilités directement ! Pour la première version il m'a fallu réduire les paramètres variables à l'essentiel. Je l'ai fait



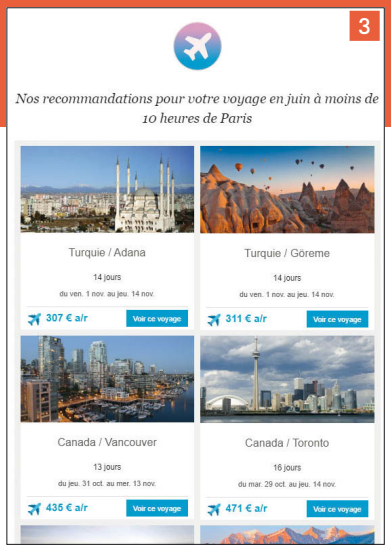
1

après avoir discuté avec quelques potentiels utilisateurs et consulté les recherches Google associées à "où partir". 1

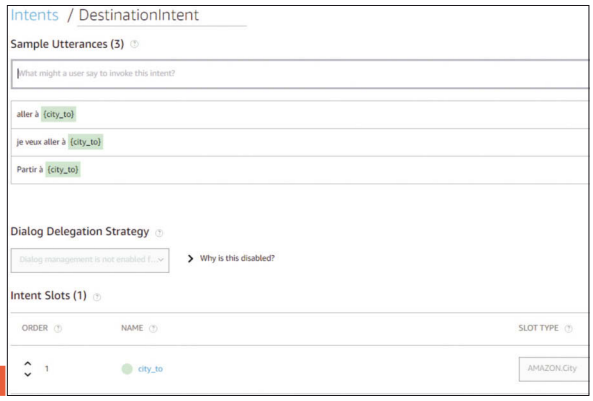
Il ressortait que les deux paramètres les plus importants sont le mois de l'année et la durée du transport. L'enjeu est d'aller le plus rapidement à l'essentiel. Donc, la skill, après le "bienvenue" d'usage, invite l'utilisateur à formuler sa requête en demandant "où partir..." en précisant le mois de l'année puis demande de spécifier la durée du trajet maximum souhaité. "Où partir" est le marqueur d'une nouvelle recherche mais cette fonctionnalité et l'invitation telle qu'elle est posée est mal comprise par la plupart des utilisateurs. Soit ils demandent en effet à partir à une destination précise (fonctionnalité non couverte par la skill pour le moment), soit ils ne donnent que le mois et oublient d'indiquer la durée du trajet souhaité. Une volonté que j'avais pour faire gagner du temps à l'utilisateur est que la demande puisse être formulée en une fois. Si vous dites "Alexa, demande à mon voyage où partir à moins de douze heures de Paris " vous obtiendrez directement le résultat : 2

La skill répond en général avec une quinzaine de résultats. C'est la limite pour la réponse pour ne pas aller au-delà la taille limite de 24kbit fixée par Amazon et risquer d'avoir une erreur.

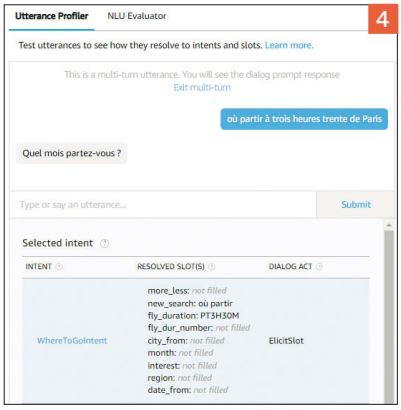
Après le listing de résultats, il est possible pour l'utilisateur de faire varier les para-



3



9



4

mètres individuellement sans recommencer la recherche depuis le début. Par exemple en demandant "A moins de cinq heures" ou "en Avril"... Les nouveaux résultats seront directement communiqués.

Pour le moment La ville de départ ne peut pas être modifiée, la skill comprend une autre ville de départ mais la recherche du prix du vol ne se fait en fait que depuis Paris. Le support d'autres villes de départ arrive avec la prochaine version ;)

A partir de là, le choix est donné à l'utilisateur d'en savoir plus et de recevoir les infos complémentaires des voyages avec le lien de réservation des vols par email ou par sms, ou de découvrir les destinations suivantes. C'est parce que par la voix, les destinations ne sont données que 3 par 3. 3

Les principales difficultés rencontrées avec le modèle de données est de trouver le bon compromis pour détecter certains slots.

Par exemple pour la durée de transport qui peut être donnée naturellement par l'utilisateur "où partir à moins de trois heures ?", on utilise ici le built-in slot AMAZON.DURATION pour détecter la durée de transport. 4 5 6

Si la durée de transport est manquante, la skill va demander à l'utilisateur de préciser la durée du voyage "combien d'heures de transport êtes-vous prêt à faire ?". Il faut prévoir que l'utilisateur puisse répondre simplement par un nombre sans ajouter "heures" derrière. C'est donc le built-in slot AMAZON.NUMBER qui est utilisé ici. 7

Le problème est que dans mon modèle j'ai un intent DestinationIntent qui permet de détecter la destination choisie par l'utilisateur. 8 9

Vous constatez que dans le sample Utterances, je n'ai pas d'utterance avec le slot {city_to} sans rien qui le précède, car en faisant cela, il y avait un conflit avec le slot filling du slot {fly_dur_number} :

A la question "Combien d'heures de transport maximum souhaitez-vous faire ?" Si l'utilisateur répondait par "trois" ou "sept" c'étaient les villes de Troyes et Sète qui étaient détectées

Pour moi la principale difficulté est de trouver les bons ajustements dans le modèle d'interaction pour que celui-ci donne les meilleures résultats en fonction des réelles demandes des utilisateurs.

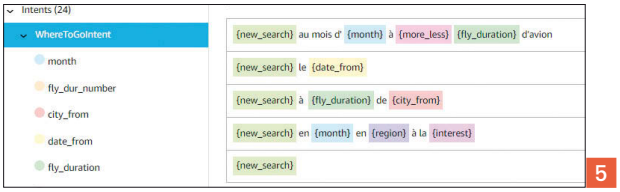
On sent que le backend de monVoyage utilise beaucoup de services tiers notamment pour les notifications de résultats ? Peux-tu nous en parler ?

Toutes les infos relatives aux destinations (meilleur mois pour s'y rendre, durées de vol, thème principal) ont été consolidées en base à partir d'infos trouvées sur internet. Le prix du meilleur vol pour la période donnée est obtenu grâce à l'API de Kiwi qui offre des performances incroyables (les prix sur une période d'un mois sont récupérés en temps réel pour chaque destination). Les sms sont envoyés avec l'API twilio. Les emails sont envoyés avec le package npm nodemailer

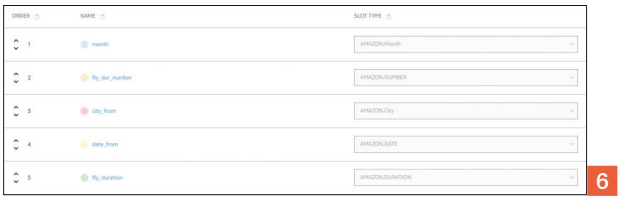
Quel futur pour ta skill ?

J'ai la volonté de mettre à jour la skill en fonction des retours des utilisateurs. La prochaine mise à jour permettra de choisir une autre ville de départ que Paris car c'est quelque chose que les utilisateurs demandent en priorité.

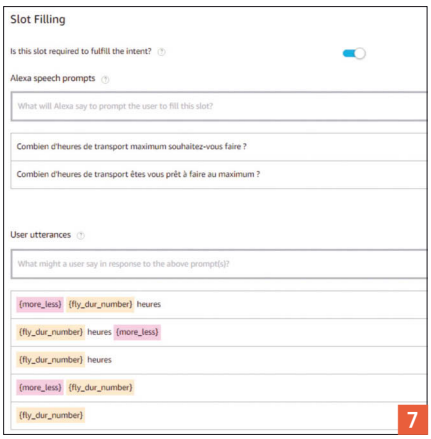
J'aimerais petit à petit que la skill monVoyage soit capable de plus en plus de variantes à la question "où partir", par exemple avec le support des envies (plage, ski, city break, vie nocturne...) en paramètre de recherche ou bien avec des périodes plus précises. Le support d'autres moyens de transport comme le train a aussi été demandé... Après pourquoi pas ajouter des fonctionnalités comme la recherche



5



6



7

8

d'un voyage pour une destination définie...

Ce qui drive mon développement sur monVoyage est de permettre la recherche et la découverte de voyage de la manière la plus simple possible. J'ai en tête en travaillant sur ce projet des utilisateurs dans leur cuisine au petit déjeuner cherchant leur destination du weekend sur echo show ou encore une famille cherchant l'inspiration pour leur prochain voyage. C'est pour ces nouveaux cas d'utilisation où la technologie est ambiante que je travaille. •

Roadmap des langages & des IDE

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

Attention ! Python 2.7 sera déprécié dès le 1er janvier 2020. A partir de cette date, plus aucune mise à jour.
PHP 7.0 arrive en fin de vie en janvier 2020

DÉJÀ DISPONIBLE

Flutter 1.9

Les principales nouveautés attendues / annoncées : Google sort très rapidement les versions de Flutter. La 1.9 a été déployée le 10 septembre dernier. Cette version inclut plus de 1500 corrections et modifications ! Elle supporte le prochain macOS et iOS 13, les nouveautés du langage Dart et les nouveaux composants Material. Grande nouvelle aussi, l'intégration de Flutter web dans le référentiel principal de Flutter ! L'éditeur annonce aussi le support de 24 nouvelles langues.

Dart 2.5

Les principales nouveautés attendues / annoncées : bien que Dart soit plus discret que Go, il évolue lui aussi régulièrement. La v 2.5 inclut une préversion du Foreign Function Interface qui permet d'appeler du code C directement depuis Dart. Cette version comporte pas mal de nouveautés : code complétion en utilisant des modèles de machine learning (preview), amélioration sur les const. Des fonctionnalités à suivre.

SWIFT 5.1

Les principales nouveautés attendues / annoncées : ce sera la première mise à jour de SWIFT 5. Cette version doit apporter des corrections de bugs et terminer la stabilité des librairies et des modules. Bien entendu, la 5.1 est rétrocompatible avec la 5.0. On doit s'attendre à quelques changements dans ce que les équipes appellent la « module stability » comme l'apparition du .swiftinterface pour le fichier d'interface à la place de .swiftmodule.
Quelques éléments ici :
https://swift.org/blog/5-1-release-process/

Kotlin 1.3.50

Les principales nouveautés attendues / annoncées : cette version apporte des améliorations sur le convertisseur Java – Kotlin, plugin de debug Kotlin/Native pour IntelliJ Ultimate, support de la compilation des projets Java multiplateformes, nouveau design de l'API temps et heure (préversion).
La version 1.3.60 est en pré-version.

TypeScript 3.7

Les principales nouveautés attendues / annoncées : disponible depuis début novembre, TypeScript 3.7 propose le chaînage d'optionnels, la coalescence null (opérateur). Attention : cette version introduit aussi des changements dans le langage qui casse la compatibilité.
Pour en savoir plus :
https://devblogs.microsoft.com/typescript/announcing-typescript-3-7/

Ruby on Rails 6

Les principales nouveautés attendues / annoncées : cette version propose la fonctionnalité Action Mailbox permettant de router des mails entrants vers des boîtes mails se comportant comme des contrôleurs. On notera également, grâce à l'arrivée d'une nouvelle API, la prise en charge de connexions à plusieurs bases de données simultanément. L'équipe de Ruby on Rails souligne que ceci offre l'opportunité de segmenter certains enregistrements dans leurs propres bases de données à des fins de dimensionnement ou d'isolation. La v6 propose un nouveau chargeur de code et propose désormais Webpacker comme bundle JavaScript par défaut.
La 6.0.1 a été déployée début novembre.

Angular 9

Les principales nouveautés attendues / annoncées : le framework continue d'évoluer rapidement. La v9 apporte un focus sur les performances, le compilateur Ivy, la localisation. La RC1 a été déployée début novembre. Attention : Angular 9 compile par défaut Ivy. Un guide de compatibilité Ivy est disponible :
https://next.angular.io/guide/ivy-compatibility
Angular 10.0 est attendue pour mai 2020.

Python 3.8

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

FIN 2019

.Net Core 3.1

Date de sortie : décembre
Les principales nouveautés attendues / annoncées : Microsoft a annoncé dès la sortie de la v3 que la 3.1 sera disponible vers le mois de novembre. Cette version sera dite LTS. Elle corrigera les bugs connus et stabilisera les API et librairies. L'éditeur annonce une migration simplifiée depuis la 3.0. C++/CLI fonctionnera avec .Net Core et Visual Studio. Uniquement sur Windows pour le moment. Il s'agit d'une version mineure. On peut s'attendre à des améliorations sur Blazor. Informations :
https://github.com/dotnet/core/tree/master/release-notes/3.1

React 16.x

Date de sortie : en cours
Les principales nouveautés attendues / annoncées : actuellement, React est en version 16.11 (disponible depuis le 22 octobre). Cette version corrige des bugs et retire des éléments de l'environnement comme unstable_createRoot. Les versions mineures sortent en moyenne chaque mois.
Pour suivre les sorties :
https://github.com/facebook/react/releases

Symfony 4.4

Date de sortie : novembre
Les principales nouveautés attendues / annoncées : Symfony 4.4 doit arriver

courant novembre. Elle répond à la politique de mise à jour annuelle : 1 en mai, 1 en novembre. Cette version sera LTS. On bénéficiera d'un thème Bootstrap 4, de la notification des mails (NotificationEmail).

PHP 7.4

Date de sortie : fin novembre
Les principales nouveautés attendues / annoncées : la 7.4 doit apporter le préchargement (preloading) au cœur de PHP pour pouvoir améliorer les performances. Ce mécanisme permet de charger l'ensemble des fichiers PHP dès le démarrage et ainsi améliorer les accès pour assurer une disponibilité constante de ceux-ci. En revanche, en cas de changement des fichiers, il faudra redémarrer le serveur. On notera aussi la disponibilité des propriétés typées, l'extension FFI (Foreign Function Interface) pour appeler du code C, le nouvel opérateur Null Coalescing, l'apparition de nouvelles méthodes (__serialize & __unesrialize).
On notera aussi le retrait de PEAR ou la dépréciation de ext/wwdx.
Au-delà : la prochaine version majeure devrait être la 8.0. Une grosse nouveauté connue sera la présence d'un compilateur JIT. Cette nouveauté permettra de se passer du Zend VM.

Ruby 2.7

Date de sortie : décembre (?).
Les principales nouveautés attendues / annoncées : le langage Ruby continue d'évoluer. La 2.6 est sortie fin 2018, notamment avec un nouveau compilateur JIT. On bénéficie aussi d'un nouveau module RubyVM ::AbstractSyntaxTree. Il analyse une chaîne de caractères et retourne les nœuds d'arbre syntaxique. Un travail d'optimisation a été réalisé. Pour améliorer la gestion de la mémoire, le ramasse-miette introduira Compaction GC qui doit défragmenter un espace mémoire fragmenté.
La preview 2 est sortie fin octobre. Toutes les évolutions : https://www.ruby-lang.org/en/news/2019/10/22/ruby-2-7-0-preview2-released/

COURANT 2020

Swift 5.2

Date de sortie : ?
Les principales nouveautés attendues / annoncées : la prochaine version du langage Swift doit se focaliser la correction de bugs, la stabilité et les performances. Peu de détails pour le moment.

Goland 1.14

Date de sortie : février
Les principales nouveautés attendues / annoncées : parmi les grands changements, l'équipe Go annonce le retrait de SSLv3, TLS 1.3 deviendra opt-out par défaut (opt-in dans la v1.12), static flag pour le build, réécriture du rulegen.

Java 14

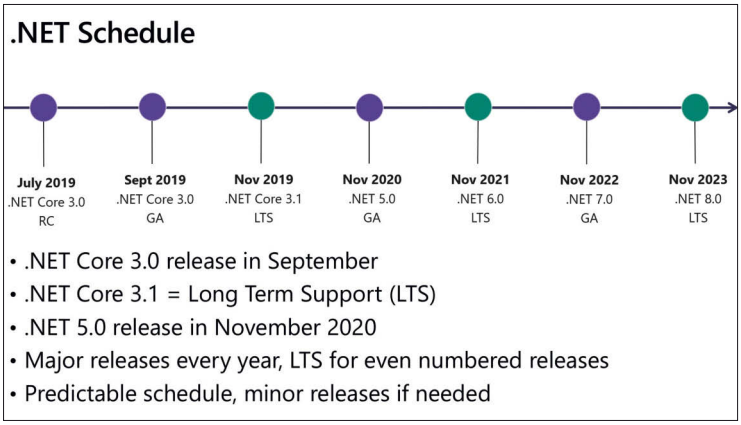
Date de sortie : mars
Les principales nouveautés attendues / annoncées : Java 14 proposera assez peu de choses à en croire les premières builds. On notera le JFR Event Streaming (consommation des données JFR). Il permettra de collecter des données de profilage et d'analyse d'une application Java en exécution. On aura aussi les expressions switch. Elles doivent simplifier le codage.
NullPointerException aura droit à quelques améliorations. Java supporta aussi les mémoires NVM.

C++20

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

.Net 5

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



LTS

On parle souvent de LTS et de non-LTS. LTS signifie support long terme. C'est-à-dire que cette version est supportée officiellement durant x années et recevra les mises à jour et les patchs de sécurité nécessaires. Une version non-LTS a une durée de vie très courte, quelques mois. Seules les versions issues d'une forte communauté, d'une fondation, d'un éditeur sont LTS. Chacun fait un peu ce qu'il veut sur le support. Exemple : Angular est sur un cycle de 6 mois pour les versions majeures. Le support se fait sur une version.

VERSIONING

On parle de versions majeures et de versions mineures. Ces dernières sont souvent des versions de bug fix, de sécurité, introduisant peu ou pas de nouveautés.
React explique aussi la structure des versions avec x.y.z :
X = une version majeure introduisant une rupture ;
Y = nouvelle fonction, version mineure (ex. : 15.6) ;
Z = bug fix. On corrige les bugs, les problèmes importants.
N'oubliez pas de lire les notes de versions (release notes). Elles indiquent les changements, les modifications, les nouveautés, la migration entre les versions.

Nutanix Era : apporter aux bases de données l'agilité nécessaire au DevOps

Comptant parmi les leaders du cloud computing d'entreprise, Nutanix a entrepris depuis 2017 une stratégie de diversification. Ces dernières années, l'éditeur a présenté des solutions de gestion du cycle de vie des applications (Calm), d'orchestration de containers (Karbon), de déploiement d'applications IoT et IA (Xi IoT) ou encore de gestion des bases de données avec Era. Et c'est sur cette dernière que nous allons nous pencher.

Présentée l'année dernière, cette solution permet d'administrer les bases de données en quelques clics avec la possibilité de provisionner, de cloner, de mettre à jour, de sauvegarder et restaurer des bases de données. Sont aujourd'hui prises en charge les solutions Oracle, SQL Server, MySQL, MariaDB, Postgres et d'autres bases de données en développement telles que SAP HANA et MongoDB.

Non content de simplifier la vie des DBA, Era permet d'apporter l'agilité du cloud public pour les bases de données et d'accélérer les logiques DevOps.

Dans les pratiques DevOps, la gestion des bases de données, des moteurs de bases de données et des modifications qui doivent y être appliquées lors des déploiements en production représentent un point de friction important et peuvent réduire les gains espérés dans le cadre de telles pratiques. Dans une étude réalisée cette année, près de 9 DBA sur 10 assurent que le déploiement des modifications de code sur les bases de données prenait plus de temps que la modification de codes sur les applications. Quant au déploiement de nouveaux schémas de bases de données, cela reste une des tâches les plus chronophages. Il faut ajouter à ces problématiques, celles de la gestion des copies et des snapshots ou encore des mises à jour qui prennent autant de temps aux équipes que le déploiement sur l'infrastructure. Era a été pensée pour éliminer ces problématiques. Là où

ces tâches pouvaient s'avérer fastidieuses, elles peuvent maintenant être automatisées sans avoir à maîtriser ou accéder à l'infrastructure.

Grâce aux API Rest, Era peut également être intégrée aux différents pipelines pour faciliter le déploiement d'environnements de développement et de test, ou simplement s'intégrer à un catalogue de services avec par exemple ServiceNow ou VMware vRealize Automation (vRA).

Chose importante, Era permet de mettre rapidement à disposition des développeurs des bases de données identiques à celles utilisées en production, permettant d'éviter ainsi les mauvaises surprises lors des déploiements. La gestion automatisée des copies et des clones permet, en outre, de garder les bases de données laissées à disposition des équipes de développement constamment à jour. Le patching simplifié permet en outre de garder des moteurs de bases de données à jour et de s'assurer de la compatibilité de ces derniers avec les différentes applications en cours de développement.

Era 2.0, présentée lors de l'évènement .Next Europe qui s'est tenu début octobre à Copenhague, simplifie encore plus le déploiement, le patching. De même, la sauvegarde façon Time Machine des bases de données s'étend désormais à SAP HANA ainsi qu'aux bases de données NoSQL en commençant par MongoDB. ERA 2.0 s'enrichit aussi de fonctions basiques de Data Management orienté sécurité des accès et du support de déploiements multiclusters.

En outre, Era permet une granularité d'accès aux différentes entités, à l'authentification locale et à l'authentification via l'Active Directory. La prise en charge de RBAC permet de différencier les accès en fonction des rôles. Par exemple, un développeur n'aura accès qu'aux clones fournis par le DBA et non à la base de données master (le système autorisera un contrôle granulaire sur des entités Era, sur les serveurs de bases de données, les bases de données, les clones, les instantanés, les balises, etc.). Il permettra également aux administrateurs de suivre les opérations réalisées par les utilisateurs.

Avec Era, Nutanix veut rendre la consommation et l'utilisation des bases des données plus simple, tout en se réconciliant avec l'infrastructure.

TROIS QUESTIONS À JAMES KARUTTYKARAN,

directeur technique de Nutanix Europe du Sud

James, quelle est l'histoire de Nutanix avec les bases de données ?

Les bases de données sont un sujet important pour nous depuis le début. Elles sont au cœur des applications et ce sont les fondations des entreprises. Nous avons accompagné les entreprises dans leurs problématiques de performances et de disponibilité des bases de données. En outre, avec les applications modernes, les bases de données ont évolué vers le NoSQL. Google et Facebook ont développé de nouvelles façons de penser les bases de données avec, par exemple, Bigtable ou Cassandra, pour adresser la quantité massive de données. On parle de Pétaoctets (Po). Dès l'origine de notre logiciel, nous avons adopté et optimisé Cassandra pour permettre d'avoir un système très évolutif et performant, ce qui en fait encore aujourd'hui un différentiateur important.

Pourquoi avoir lancé Era ?

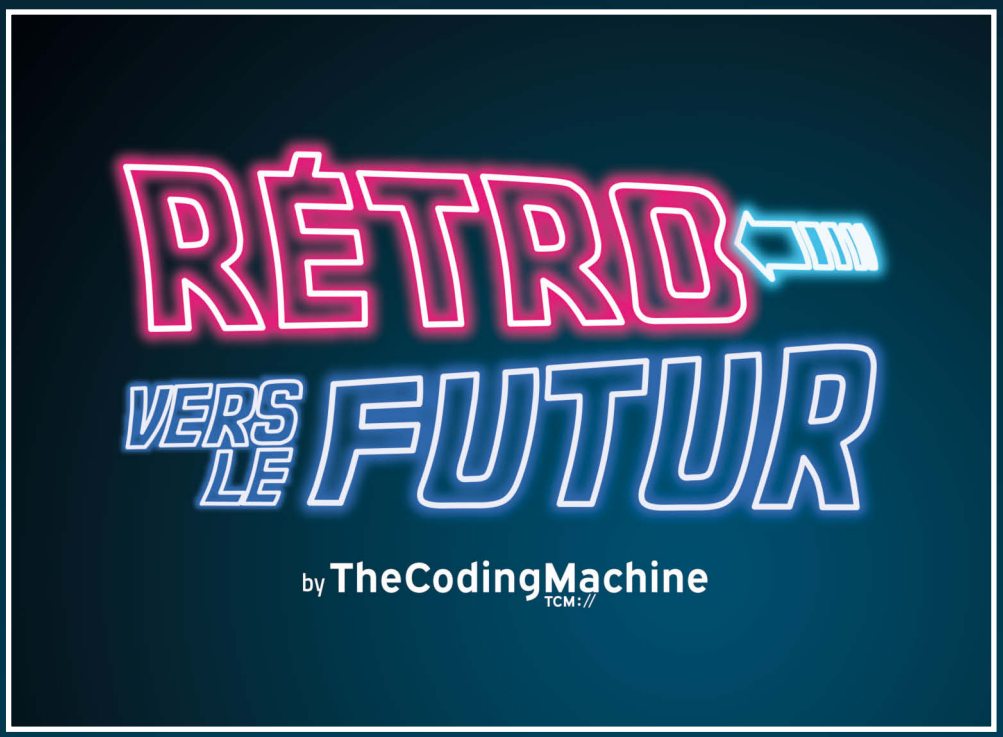
Pour déployer les applications, il faut déployer des bases de données et cela prend du temps, beaucoup de temps. Plus de 2 jours pour une base de données avec de la haute disponibilité (RAC). L'objectif était de comprendre comment mettre facilement à disposition de toute l'entreprise, y compris des développeurs, des bases de données. L'autre problème est la gestion des copies pour du test ou du dev. Cela peut représenter jusqu'à 10 fois plus de données que les données de production. Et ceci malgré les technologies de snapshot qui étaient complexes à gérer pour les entreprises car cela pouvait nécessiter une intervention des administrateurs de bases de données. Être agile et DevOps pour moderniser ses applications ne suffit plus, il faut que toute l'infrastructure et la plateforme soient aussi agiles.

Quels sont concrètement les avantages d'Era dans une logique DevOps ?

Si on revient au DevOps, c'est rapide et facile de déployer des serveurs web, des containers, etc. Mais il ne faut pas oublier les bases de données. Era permet aux administrateurs de bases de données de suivre les rythmes imposés par les logiques DevOps en simplifiant et automatisant bon nombre de tâches. Dans une logique, il faut tester… souvent. Era va permettre d'avoir des copies récentes de la base de données afin d'éliminer des potentiels problèmes. Era permet de mettre en place un référentiel qui peut être consommé de manière identique partout dans l'entreprise. En outre, avec les API Rest, Era peut être intégrée à différents pipelines, par exemple pour la création automatique de bases de données de test.

Rétro vers le futur

Save the date :
François Tonic, rédacteur du magazine Programmez! & TheCodingMachine consacrent une rétrospective inédite sur les micro-ordinateurs. Plus qu'une simple exposition, c'est l'univers du rétro qui est à l'affiche. Un saut dans le temps de la fin des années 70 au début des années 2000, culture pop & geek. Stay tuned !



« Rétro vers le futur », qu'est-ce que c'est ?

En collaboration avec François Tonic, nous avons décidé il y a quelques mois de mettre en scène l'évolution des micro-ordinateurs de la fin des années 70 à aujourd'hui. Agence de développement web et mobile mais avant tout passionnés par les nouvelles technologies, TheCodingMachine accueille cette exposition inédite dans nos locaux. Le sujet nous a tout de suite beaucoup inspiré et est en totale cohérence avec notre identité ! De là est née l'idée de monter une exposition, mettant en scène une douzaine d'ordinateurs dans des univers directement inspirés de nos propres souvenirs, un beau voyage dans le temps pour nous, organisateurs de l'évènement.

Quel est le déroulé de cet évènement inédit ?

L'exposition « Rétro vers le futur » se déroulera la dernière semaine de janvier : du 27 au 31 janvier.

Après une soirée de lancement en interne le lundi, 2 évènements phares seront organisés : un afterwork étudiant le mardi 28 janvier et un cocktail presse et clients le jeudi 30 janvier. Au-delà de la soirée de lancement pour nos collaborateurs, nous avons également prévu une visite virtuelle à distance pour notre communauté Open Source de Coders. Notre objectif est de pouvoir faire découvrir le monde des micro-ordinateurs à une très large cible, en mettant en parallèle notre savoir-faire.

Pour plus d'informations

communication@thecodingmachine.com
01.85.08.34.98
TheCodingMachine
56 rue de Londres 75008 Paris
#tcmverslefutur
Twitter : @coding_machine
Instagram : @thecodingmachine_tcm
Linkedin : TheCodingMachine

POURQUOI CIBLER LES ÉTUDIANTS POUR CET ÉVÈNEMENT ?

Chez TheCodingMachine, nous recrutons tous les ans environ une quinzaine de stagiaires au début de l'année avec à la clé un CDI. Nous formons des étudiants en fin de cycle ingénieur, pour leur proposer un poste de Chef de projet web à la sortie. En 2019, 93% des étudiants ayant effectué leur stage chez nous ont été embauchés par la suite. Cette exposition permet à ces futurs TCMienn-e-s de partager avec nous, le temps d'un après-midi, notre culture d'entreprise. La transmission est une valeur ancrée dans notre identité. C'est l'occasion rêvée de se rencontrer.

Christophe Villeneuve
Consultant IT pour pour Atos, Mozilla Rep, auteur de livres pour les éditions Eyrolles et aux Editions ENI, PHPère des elePHPants PHP, membre des Teams DrupalFR, AFUP, LeMug.fr (MySQL/MariaDB User Group FR), ...

Security by design

niveau 100

La cybercriminalité est en constante augmentation ces dernières années. Les techniques s'appuient sur les développements modernes qui ont des cycles de mises en production de plus en plus courts, mettant parfois en retrait une partie de la validation.

Dans cet article, nous allons revisiter la sécurité au travers de la « security by design » (ou secure by design, NDLR) à destination des développeurs. Les vulnérabilités logicielles ont en effet souvent pour origine des erreurs de programmation ou des omissions qui laissent les applications / serveurs / sites Web exposés. C'est aux développeurs logiciels de créer des applications avec un haut niveau de sécurité pour empêcher ces attaques et réduire la surface d'attaque potentielle.

La security by design est importante pour de nombreuses raisons, car elle a pour but de protéger : tout d'abord contre les malveillances, les erreurs et la malchance. Et de l'autre contre les vols, fraudes, destructions et perturbations.

La sécurité est une activité de gestion des risques. Si vous vous faites pirater, vous perdez du temps, de l'argent, de la vie privée, de la réputation, etc. C'est pourquoi vous devez prévoir un modèle pour équilibrer les coûts et le risque de perte.

Les principes

Il n'existe pas de normes ou de concepts définis par des standards. Mais on trouve des référentiels, des bonnes pratiques, des méthodes pour faire de la security by design. Nous pouvons citer Viega & McGraw, OWASP, nist, NCSC, cliff Berg's, etc.

La « security by design » possède de multiples facettes qui doivent s'adapter à votre projet. Elle s'articule principalement autour de 4 axes :

- La stratégie d'entreprise ;
- Le pilotage des processus métiers : approche fonctionnelle et orientée métier ;
- Urbanisation du SI : fonctionnel orienté applications & données ;
- Architecture technique avec le socle technique de l'infrastructure.

Quel que soit l'axe où vous vous trouvez, le processus technique doit rester identique. C'est-à-dire : la prévention, la détection et la réaction.

L'article se dédie plutôt à l'ensemble des profils techniques pour prévenir des risques.

La security by design

Nous avons choisi 10 bonnes pratiques qui répondent à la sécurité par le design. Cette liste n'est pas exhaustive, mais c'est déjà un bon début.

Minimiser la surface d'attaque

Lorsqu'une nouvelle fonctionnalité débarque dans une application, le risque de vulnérabilités de celle-ci (ou sur une autre partie du code) augmente, pouvant toucher la totalité de l'application.

C'est pourquoi l'objectif d'un développement sécurisé est de réduire le risque global en diminuant la surface d'attaque.

Par exemple, si vous ajoutez une recherche avancée supplémentaire, les risques encourus sont :

- Un risque de vulnérabilité aux attaques par injection SQL (un grand classique) ;
- Une attaque d'inclusion de fichiers.

La solution pour limiter les risques d'attaques serait de l'associer à un compte utilisateur identifié et référencé, comme cela vous pouvez lui proposer de mémoriser ces différentes recherches.

Cependant, si la fonction de recherche a été complètement réécrite en supprimant l'existant et en ajoutant cette évolution, avec une meilleure interface utilisateur, cela élimine une surface de l'attaque.

NDLR : *la gestion de l'authentification reste cruciale pour l'accès aux services et définir les droits à chaque utilisateur. Ne réinventez pas la roue mais faites l'implémentation proprement.*

Établir des valeurs par défaut sécurisées

Lors du déploiement d'une application, celle-ci doit être sécurisée par défaut pour que les utilisateurs puissent obtenir en toute confiance une application 'prête à l'emploi'. Bien entendu, un utilisateur doit pouvoir disposer d'un compte avec certains privilèges (élevés ou pas) et prédéfinis.

Les règles de sécurité doivent être strictes sur les différentes actions que l'utilisateur peut faire et sur la façon dont les enregistrements de celui-ci seront traités.

Par exemple, la garantie passe par différentes implications de l'utilisateur comme :

La mise à jour de son mot de passe à un intervalle régulier ;

- La vérification de la complexité de son mot de passe lors du changement ;
- La détermination d'un vieillissement de son mot de passe pour lui imposer le changement sur une longue période d'absence.

NDLR : *une autre technique est l'utilisation d'une authentification à double facteur.*

Principe du moindre privilège

Une application Web doit avoir un minimum de privilèges comme un accès aux contenus et aux différentes ressources liées aux droits des utilisateurs.

Si une action précise est prévue, celle-ci doit posséder un minimum de privilège comme :

- La définition des droits de l'utilisateur ;
- Les permissions de ressources telles que les limites du CPU, la mémoire, le réseau ;
- Les permissions du système de fichiers.

Par exemple, un site d'actualité avec une équipe de contributeurs, possédant des rôles définis, doit pouvoir accueillir de nouveaux membres qui auront un minimum de droits. Ils peuvent par exemple être classés comme 'auteur', sans obligatoirement avoir la possibilité de publier directement un contenu sans une relecture par une personne habilitée. De plus, il ne doit pas avoir des privilèges supplémentaires comme ceux de d'administrateur qui leur permettrait d'avoir la possibilité d'administrer le site web, de gérer les utilisateurs...

La défense en profondeur

La défense en profondeur privilégie des contrôles de sécurité multiples pour aborder les risques de différentes manières. C'est la meilleure option pour sécuriser une application. Les contrôles, lorsqu'ils sont utilisés en profondeur, peuvent rendre les vulnérabilités très élevées extrêmement difficiles à exploiter. Bref, vous réduisez la surface d'attaques.

Ainsi, au lieu d'avoir un seul contrôle de sécurité pour l'accès des utilisateurs, vous pouvez ajouter :

- Une validation à différents niveaux ;
- Des contrôles de vérification centralisés ;
- Une obligation pour les utilisateurs d'être connectés et identifiés à toutes les pages de navigation ;
- De prévoir des outils d'audit de sécurité supplémentaires ;
- Avec des outils de journalisation (log).

Par exemple, pour un utilisateur qui se connecte avec ses identifiants (login + mot de passe), vous pouvez associer un contrôle IP, une double authentification (2FA), un captcha. Et vous lui imposez des autorisations d'accès à une partie de l'application, tout en gardant un historique des différentes tentatives de connexion.

Échec en toute sécurité

L'échec d'une transaction ou une action venant de l'utilisateur peut provoquer différents problèmes, comme :

- Un problème de traitement d'une transaction :
  Il s'agit d'une action générée par l'utilisateur comme un clic sur un bouton et le résultat retourné est erroné ou inexistant ;
- Un problème de connexion :
  Il s'agit principalement des identifiants saisis (login + mot de passe) qui vont permettre d'accéder à une interface utilisateur ;
- Les données saisies sont incorrectes :
  L'utilisateur va entrer des caractères ou symboles dans un formulaire et celui-ci doit détecter si la saisie dans le champ est au bon format (exemple : un code postal).

Lors d'un échec, le message d'erreur affiché doit être clair et précis, car si ce n'est pas le cas, le message peut fournir de nombreuses informations pouvant être exploitées par un hacker. C'est pourquoi les informations sensibles ne doivent pas être affichées à l'écran mais plutôt dans un journal de logs ou dans une base de données spécifiques.
Dans la réalité, cela se représente de la façon suivante :

```
isAdmin = true;
try {
 codeWhichMayFail();
 isAdmin = isUserInRole( "Administrator" );
}
catch (Exception ex) {
 log.write(ex.toString());
}
```

Si l'appel à la fonction codeWhichMayFail () ou isUserInRole échoue ou déclenche une exception, alors l'utilisateur sera administrateur par défaut.
Il s'agit d'un risque de sécurité important car une personne mal intentionnée devriendrait administrateur du projet.

Attention aux services tiers

Dans un souci de productivité, un projet web s'appuie ou utilise des services tiers pour offrir de nouvelles fonctionnalités. Un développeur n'est pas là pour coder un service qui existe déjà. Mais utiliser un service externe, une API externe, etc., a une conséquence sur la sécurité et peut, potentiellement, élargir la surface d'attaque. Il est important de suivre leurs actualités et de s'assurer qu'ils sont aussi un processus clair et précis de sécurité dans les services que vous utilisez. Quelles sont les nouvelles versions d'API ? Les fonctions obsolètes ? Etc.
Cela signifie que l'application doit toujours vérifier la validité des données envoyées par les services tiers. Comme ceci, vous évitez de ne pas donner à ces services des permissions de haut niveau dans l'application.
Par exemple : l'utilisation d'un coupon de fidélisation fournit des données qui sont utilisées par un tiers. Ces données doivent être vérifiées pour s'assurer qu'elles peuvent être affichées en toute sécurité aux utilisateurs finaux sans que les informations affichées soient erronées, corrompues, etc.

Séparation des fonctions

La séparation des fonctionnalités ou des tâches est importante dans une application pour éviter des agissements frauduleux.
Par exemple : les accès administrateurs sur un projet ne sont pas obligatoirement les mêmes s'il s'agit d'un site marchand ou d'un site d'actualités. Cette distinction est importante, car l'administrateur ne doit pas avoir la possibilité de modifier les informations d'une commande client comme la quantité, le prix ou l'adresse. Mais pour un site d'actualités, il doit pouvoir modifier le contenu si c'est nécessaire même s'il n'en est pas l'auteur.

Éviter la sécurité par l'obscurité

La sécurité par l'obscurité est considérée comme une sécurité faible, voire inexistante, c'est-à-dire que le chemin d'accès n'est pas visible directement par l'interface de l'application. C'est à dire en cacher une url à l'internaute pour éviter qu'il y accède directement.
Après quelques mois, cette partie obscurisée ou cachée, ne bénéficiera pas des évolutions de l'application, et, par conséquent, des dernières implémentations de sécurité par rapport au reste du projet. Cela se résume comme une porte dérobée, surtout si la partie cachée permet d'avoir un profil d'administrateur. Par exemple un attaquant va deviner en modifiant le port de l'application qu'il peut accéder à une interface et récupérer le mot de passe administrateur.
La solution pour éviter cette faille est d'imposer que tous les utilisateurs passent par un seul point d'identification à x facteurs de contrôles et de mots de passe. C'est-à-dire une défense en profondeur avec des limitations fonctionnelles et d'accès.

Gardez la sécurité simple et claire

Les développeurs doivent éviter d'utiliser une architecture très/trop sophistiquée lorsqu'ils développent des contrôles de sécurité pour leurs applications. Cela signifie que des mécanismes très complexes peuvent augmenter le risque d'erreurs dans un projet, et provoquer des failles supplémentaires.
La conception complexe est rarement comprise, ce qui est contraire à une conception simple qui permet d'effectuer une analyse plus rapide.
C'est pourquoi il faut éviter les fonctions inutiles, qui se dédoublent, ou même des fonctions sophistiquées.
Par exemple, vous pouvez vous poser cette question : lors de l'ajout d'une nouvelle fonctionnalité dans votre architecture, celle-ci ne pourrait-elle pas utiliser certaines fonctionnalités déjà existantes ?

Les corrections

Une fois qu'un problème de sécurité a été identifié dans le projet, il est important d'élaborer un test et de comprendre la cause du problème et les conséquences de cette faille.
Toutefois, si le projet utilise le modèle de conception déjà utilisé pour une application, il est probable que le problème de sécurité soit aussi présent sur les autres projets qui utilisent le même modèle. Il est donc essentiel de développer la bonne solution de sécurité sans introduire de régressions.
Par exemple, un utilisateur pouvant accéder aux informations d'un autre utilisateur, juste en modifiant la valeur de son cookie, est une faille et vous risquez une fuite de données, voire, la possibilité de pirater un compte et des accès non autorisés.
La solution semble relativement simple, mais comme le code de gestion des cookies est partagé entre toutes les applications, un changement vers une seule application aura des répercussions sur toutes les autres applications. C'est pourquoi le correctif doit donc être testé sur toutes les applications concernées.
La réactivité à corriger une faille / une vulnérabilité est importante dans votre politique de sécurité.

Conclusion

La 'security by design ' impacte tous les métiers de l'entreprise, car un responsable, un commercial ou une équipe de développement, n'auront pas les mêmes visions de la sécurité (NDLR : ce qui est une erreur malheureusement trop répandue. Le Securty Plan devrait donner une vue globale et unique à tout le monde).
Toutefois, tous les systèmes auront besoin d'utiliser les principes de base de la conception de la sécurité. C'est pourquoi, un socle commun permettra de proposer une architecture modulaire pour répondre aux différentes attentes.
La sécurité dans une application ajoute une couche supplémentaire de sécurité à l'ensemble du SI, mais elle peut aussi impacter les performances globales d'une application, d'une infrastructure. Se rappeler que plus la sécurité intervient tardivement dans un projet, plus son impact sera important. Il faut la définir dès la conception.
Bien entendu, il faut garder à l'esprit les fondamentaux de la sécurité de l'information :

- Confidentialité : n'autoriser l'accès qu'aux données pour lesquelles l'utilisateur est autorisé ;
- Intégrité : s'assurer que les données ne sont pas altérées ou altérées par des utilisateurs non autorisés ;
- Disponibilité : s'assurer que les systèmes et les données sont disponibles pour les utilisateurs autorisés lorsqu'ils en ont besoin.

Du côté des attaques, les plus dangereuses contre lesquelles les développeurs doivent se prémunir sont celles des collaborateurs, car ils ont souvent un niveau élevé d'accès aux systèmes sensibles.
Il n'est jamais trop tard pour consulter et appliquer le TOP 10 de l'OWASP !
N'hésitez pas à lire et relire notre dossier spécial cybersécurité dans Programmez! n°234. •

Marie Viley
IT Recruitment Consultant / RGPD Consultant at **Zenika**

3 techniques faciles de manipulation

Qui ne s'est jamais fait manipuler ? Bien évidemment, on est tous allés à une soirée alors qu'on n'en avait pas envie, on a tous acheté quelque chose sans en avoir besoin et on a tous été influencés d'une façon ou d'une autre par nos proches.

Je vais donc vous exposer ici les mécanismes de prises de décisions qui permettent aux autres de vous manipuler simplement. À la fin de cet article, vous devriez être moins influençable et davantage maître de vos décisions. Et vous pourrez même vous servir de ces éléments pour manipuler les autres.

Tout d'abord qu'est-ce que la manipulation ? On obéit tous à des règles qui nous sont propres, sans même en avoir conscience.
Le but de la manipulation est d'utiliser ces règles instinctives pour faire faire aux gens ce qu'ils n'auraient pas fait naturellement.
Je vais vous présenter les 3 principales règles de la manipulation.

1ere technique

l'engagement et la cohérence

Vous vous êtes sûrement déjà demandé : "Mais qu'est-ce que je fais là ?" "Pourquoi je suis venue ici ?" lors d'un événement ou d'une réunion qui, à la base, ne présentait pas un grand intérêt.

Eh bien c'est probablement à cause de la 1ère technique, celle de l'engagement et de la cohérence !

Afin d'éviter une dissonance interne, nous créons de la cohérence avec nos actes et nos engagements.

C'est pour cette raison qu'une fois engagé(e) on veut rester cohérent(e) avec notre décision de départ et on s'investit encore plus. Pour résumer la première technique, plus on est engagé(e), plus on s'engage.
Pour manipuler quelqu'un, on va amener la personne à prendre une décision ou lui faire croire qu'elle a pris une décision, la personne est engagée et va agir conformément à sa décision.

Les recruteurs de dons pour les associations humanitaires utilisent pour cela la technique de l'entonnoir : ils vont vous poser des questions où vous allez forcément répondre par oui. Vous vous engagez au fur et à mesure... Le fait de dire "oui" à des questions donnant une image positive de vous-même va vous conduire à accepter de donner à la fin, ceci afin de soutenir votre construction mentale : "Je suis une bonne personne, je suis engagée sociétalement, donner est une conclusion logique à la vision que j'ai de moi. »

Pensez à la fois où vous vous êtes retrouvé en train d'aider un collègue à faire son travail juste parce que vous avez eu le malheur de dire que vous connaissiez le sujet. La technique est simple. Il suffit de poser des questions pour que l'autre puisse vanter ses connaissances sur un sujet et de lui demander ensuite de l'aide sur ce sujet précis. La personne s'engage progressivement en expliquant ses réalisations et ses projets sur le sujet et il sera difficile pour lui de refuser de vous aider sur un sujet qu'il maîtrise si bien !

2e technique

la règle de la réciprocité

Les gens sont plus disposés à accéder à vos demandes si vous leur avez fourni quelque chose en premier. La règle de la réciprocité est ancrée. Il est difficile de s'en affranchir, car elle crée un sentiment d'obligation. Nous sommes contraints de nous acquitter de notre dette.

La force de cette règle est qu'il est quasiment impossible d'accepter un cadeau ou un avantage sans offrir quelque chose en retour.

Pour manipuler quelqu'un, on va créer des « dettes ». Cela fonctionne encore mieux quand lesdites « dettes » sont créées devant d'autres personnes. La pression est encore plus grande. Lorsque nous aurons suffisamment de « dettes » en notre faveur, nous pourrons demander quelque chose de plus conséquent que ce que l'on a donné au départ.

Les exemples les plus répandus ce sont les échantillons et les offres d'essai. C'est statistique, les magasins qui offrent des échantillons auront un taux de transformation plus élevé que ceux qui n'en offrent pas. Il n'y a pas de rapport particulier avec la qualité des produits, mais plutôt avec le fait que l'on se sente redevable.

Il existe beaucoup d'arnaques dans la rue qui visent des touristes en utilisant ce principe. Une personne va vous donner une rose ou un gadget gratuitement en insistant. Votre premier réflexe sera de tendre la main et de prendre cet objet. Une fois en main, la personne vous demandera un don ou une signature, il est compliqué alors que vous avez toujours son « cadeau » dans la main de refuser.

Prenez maintenant l'exemple du travail, vous pouvez aider un collègue quelques minutes sur un sujet et lui demander un service plus important le lendemain.

Il existe une petite variante : la méthode du rejet/retrait. Cette méthode consiste à demander quelque chose de tellement coûteux (en termes de temps, d'argent, d'énergie, etc.) que la personne en face n'aura pas d'autre choix que de refuser. Dans un second temps je pourrais alors demander ce que je voulais réellement à la base et qui bien évidemment sera moins coûteux pour la personne. Elle aura donc l'impression que j'ai fait une concession et elle en fera donc une.

Par exemple, vous pouvez demander à un ami de vous déposer chez vous en sachant que c'est bien trop loin ou compliqué, il va refuser et vous allez lui demander ce que vous vouliez à la base : qu'il vous dépose à l'arrêt de tram le plus proche.

3e technique

la preuve sociale

Consciemment ou inconsciemment nous avons tendance à reproduire le comportement des groupes de gens qui nous entourent. Et plus le statut social ou la notoriété de la personne que l'on observe sont élevés, plus on aura tendance à l'imiter.

Il y a de nombreuses études et vidéos sur le sujet qui montrent des gens enfermés dans une pièce qui se lèvent lors d'un signal sonore et qui se rassoient à la fin du signal sonore. Lorsque l'on fait rentrer une nouvelle personne sans la prévenir, elle s'aligne rapidement sur le comportement des autres.

Si dans la rue, dix personnes regardent le ciel, nous aurons tendance à regarder le ciel.

Cette technique est très utilisée pour attirer les clients. Prenons les restaurateurs qui placent les premiers clients près des fenêtres ou de la devanture. Les clients suivants entreront davantage dans un restaurant qui a déjà des clients plutôt que dans un restaurant vide.

On utilise aussi cette technique pour vendre des produits ou des services grâce à des labels comme "élu produit de l'année" ou bien "saveur de l'année", grâce à des avis sur des sites comme tripadvisor.

Pour devenir populaire : plus une personne a d'amis/followers/connexions sur les réseaux sociaux, plus elle aura tendance à en avoir, et plus on aura tendance à lui faire confiance. A contrario, un statut ou un profil qui ne semble intéresser personne ne dégage généralement pas beaucoup d'intérêt.

Ce n'est pas parce qu'un évènement, magasin ou restaurant attire les foules qu'il est de bonne qualité. Et ce n'est pas parce qu'une personne est populaire qu'elle est un exemple à suivre.

Le danger de ce comportement est que face à un événement, nous allons copier la réaction des autres même si cette réaction n'est pas appropriée ou dangereuse.

Nous avons souvent le témoignage de victimes d'agressions physiques qui ne comprennent pas pourquoi personne ne leur est venu en aide. Lorsqu'on est témoin de ce genre de chose, il est important de réfléchir et de se mettre à la place de la personne plutôt que de copier le comportement de ses voisins.

Conclusion

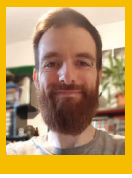
La manipulation est omniprésente et il est assez facile de détecter certaines techniques pour les éviter que ce soit au travail ou dans la vie de tous les jours. C'est un exercice qui devient de plus en plus aisé avec de la pratique et des lectures. À force de pratique il devient même possible d'utiliser certaines techniques subtilement et presque de manière automatique.

Sources

Petit traité de manipulation à l'usage des honnêtes gens de Jean-Léon Beauvois et Robert-Vincent Joule

Influence et manipulation de Robert Cialdini

Petit cours d'autodéfense intellectuelle de Normand Baillargeon



Nicolas Bouteillier
**Artisan développeur* Agile - server admin - fullstack (php/symfony/jquery/bootstrap)*
http://www.kaizendo.fr
https://fr.linkedin.com/in/kaizendo
https://twitter.com/NicolasKaizen

Mon premier projet Laravel

Dans le monde PHP, depuis la chute de Zend Framework, on résume souvent les choses à Symfony. Laravel est clairement un framework en vogue. Sur Github, le projet a presque 56 000 étoiles et +17 000 projets, 129 000 questions sur stackoverflow. Cela montre la vivacité de la communauté Laravel. Plusieurs arguments portent ce framework : une architecture plutôt propre, la communauté croissante et active, la documentation, l'approche MVC. Notre contributeur-expert Nicolas Bouteillier vous plonge dans les entrailles de ce framework.

La rédaction.

Si comme moi vous en avez beaucoup entendu parler mais pas encore eu l'occasion d'essayer, voici un petit article pour débuter.

Principe et contexte

Nous allons quasiment suivre la documentation officielle pas à pas, avec seulement quelques petites variations induites par mes choix de configuration.

Pour commencer, un peu de contexte. J'ai besoin de tester plusieurs API de fournisseurs de solutions dans le tourisme, des « chanel manager », il en existe un bon nombre, tous n'ont pas les mêmes fonctionnalités, et je vais devoir faire un choix.

Sans entrer dans les détails, ils permettent de centraliser la gestion des produits – touristiques – dans un grand nombre de sites et de logiciels spécialisés, réservation, annulation. Pour pouvoir décider, je veux implémenter leurs solutions, quelques fonctionnalités essentielles pour me faire une idée générale du produit et la facilité/difficulté de mise en œuvre.

Parallèlement à ça, un client m'a fait une demande sur un développement basé sur Laravel. Etant développeur Symfony, je pense être à l'aise avec Laravel, celui-ci utilisant plusieurs des bundles de Symfony.

Il a de plus la réputation d'avoir une courbe d'apprentissage (la fameuse « learning curve ») moins importante, il faut comprendre par là qu'il est censé être plus simple à prendre en main.

On parle bien de prise en main, pas de maîtrise.

Il existe des solutions qui peuvent paraître (ou être) plus simples pour faire ce genre de tests, citons Lumen qui est un micro-framework par Laravel, ou bien simplement Symfony 4 qui remplace maintenant Silex qui était le micro-framework basé sur Symfony. Dans mon cas je profite de ce projet pour tester, mais ça n'est pas forcément le bon choix pour le besoin.

On va donc explorer quelques fonctionnalités de base, les premiers pas avec Laravel. Avant de partir sur un projet destiné à être mis en production ou bien un projet plus « sérieux » que des tests je vous encourage vivement à vous renseigner sur les bonnes pratiques : nommage de fichier, arborescence, architecture, etc.

Pré-requis :

- parler le PHP ;
- virtualbox installée et opérationnel ;
- une clef publique dans votre user courant (~.ssh/id_pub.rsa) ;
- composer installé.

L'environnement de développement

Laravel fournit Homestead, un environnement virtuel qui héberge votre projet. Il est géré avec Vagrant. Le tout s'installe en quelques minutes et a beaucoup d'avantages. Parmi les différentes possibilités d'installation, je choisis l'installation de Homestead par projet, car je veux pouvoir partager l'environnement avec d'autres développeurs, ou bien même un client qui pourra tester chez lui dans les mêmes conditions que moi.

Par souci de compatibilité (le côté multiplateforme) et de simplicité, j'ai choisi Virtualbox comme provider pour Vagrant, même s'il y en a d'autres.

J'utilise Phpstorm pour développer, il existe un plugin Laravel que je vous conseille vivement d'installer. Pour ma part, j'ai créé un répertoire « laravel» qui va héberger mes différents projets, chaque projet avec son propre sous-répertoire et son Homestead.

```
/var/www/laravel/poc-consume-api
```

Une fois mon projet créé avec Phpstorm, dans le terminal de lance :

```
composer require laravel/homestead --dev

Using version ^9.0 for laravel/homestead
./composer.json has been created
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 9 installs, 0 updates, 0 removals
  - Installing symfony/polyfill-ctype (v1.11.0): Loading from cache
  - Installing symfony/yaml (v4.3.2): Loading from cache
  - Installing symfony/process (v4.3.2): Loading from cache
  - Installing psr/container (1.0.0): Loading from cache
  - Installing symfony/service-contracts (v1.1.5): Loading from cache
  - Installing symfony/polyfill-php73 (v1.11.0): Loading from cache
  - Installing symfony/polyfill-mbstring (v1.11.0): Loading from cache
  - Installing symfony/console (v4.3.2): Loading from cache
  - Installing laravel/homestead (v9.0.7): Downloading (100%)
symfony/service-contracts suggests installing symfony/service-implementation
symfony/console suggests installing symfony/event-dispatcher
symfony/console suggests installing symfony/lock
symfony/console suggests installing psr/log (For using the console logger)
Writing lock file
Generating autoload files
```

Pour finir :

```
php vendor/bin/homestead make

Homestead Installed !
```

Je change quelques paramètres, j'ai un faible pour MariaDb, dans « Homestead.yaml » (qui se trouve à la racine) :
mes chemins :

```
folders:
    map: /var/www/laravel/consume_channel_manager_api
    to: /home/vagrant/code/

sites:
    map: homestead.test
    to: /home/vagrant/code/laravel/public
```

Quoi que vous choisissiez ici comme architecture, le « to » : doit correspondre au répertoire « public » de Laravel.
Ensuite Mariadb :

```
Features :
  -
    mariadb: false
par :

features:
  -
    mariadb: true
```

(attention au YAML, tous les espaces comptent)

Pour lancer la VM c'est très simple, dans un terminal :

```
vagrant up
```

ATTENTION

j'ai eu un soucis lors du vagrant up pendant la relecture de cet article, une erreur qui est causée par le réseau d'une manière ou d'une autre :

```
OpenSSL SSL_read: SSL_ERROR_SYSCALL, errno 104
```

Une solution consiste à récupérer la box avant en utilisant l'option --insecure, c'est sûr que ce n'est pas le mieux à faire, mais ça évite d'attendre que ça refonctionne.

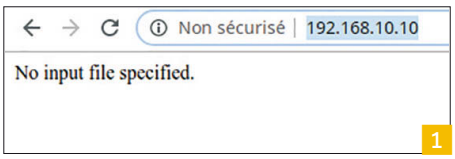
```
vagrant box add --insecure laravel/homestead
```

et après :

```
vagrant up
```

Je vous conseille d'ouvrir un terminal en dehors de votre IDE. Une fois la VM lancée, ce sera plus confortable de s'y connecter que dans le terminal de Phpstorm par exemple. Pour vous y connecter c'est : vagrant ssh depuis le répertoire du projet.
Quand la VM est lancée, elle a une adresse, pour moi c'est :
http://192.168.10.10/ ou http://homestead.test
(configurable dans Homestead.yml)

Elle est accessible, mais pour l'instant rien n'est présent. 1



1

Plus d'infos sur l'utilisation d'Homestead :
https://laravel.com/docs/6.x/homestead#daily-usage

Laravel

Ensuite, on va mettre Laravel en lui-même dans un sous-répertoire. J'ai choisi de le nommer simplement «laravel», ça aurait pu être «code», «src»…
Attention à renseigner correctement Folder et Map dans le fichier Homestead.yml si vous changez ici.
Depuis le terminal de phpstorm :

```
mkdir laravel
```

À partir de cette étape, toutes les commandes seront à lancer depuis la VM. Si vous lancez les commandes depuis votre pc, c'est son environnement qui sera pris en compte. Composer par exemple se basera sur votre version locale de php qui peut différer de votre projet, ou d'un projet sur l'autre. Un des avantages d'Homestead (et de Vagrant plus largement) est justement d'avoir un environnement de développement complètement indépendant de votre environnement local. Réplicable et partageable.
Depuis le répertoire du projet on lance la commande :

```
vagrant ssh
```

Elle va vous connecter à la machine virtuelle, en utilisant un accès ssh avec votre clef publique. Une fois dans la VM :

```
cd /home/vagrant/code/
```

puis :

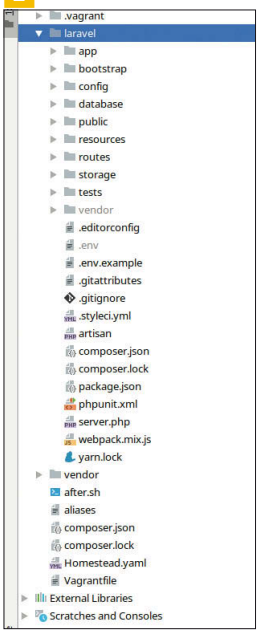
```
laravel new
```

(L'installateur Laravel est déjà installé dans Homestead)
Ci-dessous mon arborescence à la fin de l'installation, avec un petit mot sympa de la part de l'installer :

```
«Application ready! Build something amazing.»
```

Ça fait plaisir:) 2

2



Base de données

Comme la base de données est configurée directement dans Homestead, il y a quelques changements de configuration à faire dans le fichier .env à la racine du projet pour que tout fonctionne correctement :

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=homestead
DB_USERNAME=homestead
DB_PASSWORD=secret
```

Ce sont les paramètres par défaut de «Homestead.»

Je veux des users

J'ai besoin de savoir comment se passe la création des utilisateurs, je vais utiliser la commande de base pour créer une gestion mini-

maliste de ceux-ci, avec la possibilité de s'authentifier. Pour la version 6.0 de Laravel les choses changent et vous devez avoir laravel/ui d'installé :

```
composer require laravel/ui –dev
```

Ensuite cette commande va générer les fichiers nécessaires :

```
php artisan ui vue –auth
```

Le script nous invite à jouer la commande suivante :

```
npm install && npm run dev
```

Pour finir il faut mettre les tables nécessaires dans la base de données :

```
artisan migrate
```

C'est fait ! En retournant voir le fichier de routes, /routes/app.php on voit bien la nouvelle route créée :

```
Auth::routes();
Route::get('/home', 'HomeController@index')->name('home');
```

Et lorsqu'on affiche : http://homestead.test/home
On est correctement redirigé vers : http://homestead.test/login

Ajouter des champs aux utilisateurs

Il est très rare de ne pas avoir besoin de personnaliser les utilisateurs, dans mon optique de test et d'exploration de Laravel, j'ajoute un champ :

```
- last_name


Fichier app/User.php :

/**
 * The attributes that are mass assignable.
 *
 * @var array
 */
protected $fillable = [
    'name', 'email', 'password', 'last_name'
];
```

Ensuite on génère un fichier de migration de base de données en ligne de commande :

```
artisan make:migration add_last_name_to_user_table
Created Migration: 2019_07_24_143949_add_last_name_to_users_table
```

Dans le fichier créé :

```
/database/migrations/2019_07_24_143949_add_last_name_to_users_table.php
```

On va ajouter la colonne 'last_name' dans up() et prévoir sa suppression dans down() pour pouvoir revenir en arrière si on le souhaite :

```
/**
 * Run the migrations.
 *
 * @return void
 */
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->string('last_name');
    });
}
/**
 * Reverse the migrations.
 *
 * @return void
 */
public function down()
{
    Schema::table('users', function (Blueprint $table) {
        $table->dropColumn('last_name');
    });
}
```

Il faut appliquer la migration :

```
php artisan migrate
```

Ensuite il faut l'ajouter aux contrôleurs et aux vues, je vous montre pour register :

```
Fichier /app/Http/Controllers/Auth/RegisterController.php
/**
 * Get a validator for an incoming registration request.
 *
 * @param  array  $data
 * @return \Illuminate\Contracts\Validation\Validator
 */
protected function validator(array $data)
{
    return Validator::make($data, [
        'name' => ['required', 'string', 'max:255'],
        'last_name' => ['required', 'string', 'max:255'],
        'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
        'password' => ['required', 'string', 'min:8', 'confirmed'],
    ]);
}
/**
 * Create a new user instance after a valid registration.
 *
 * @param  array  $data
 * @return \App\User
 */
protected function create(array $data)
{
    return User::create([
        'name' => $data['name'],
        'last_name' => $data['last_name'],
        'email' => $data['email'],
        'password' => Hash::make($data['password']),
    ]);
}
```

Puis dans la vue :

```
/resources/views/auth/register.blade.php
```

J'ajoute en dessous du div de « name » (ligne 26):

```
<div class="form-group row">
  <label for="name" class="col-md-4 col-form-label text-md-right">{{ __('Last name'
) }}</label>
  <div class="col-md-6">
    <input id="last_name" type="text" class="form-control @error('last_name') is
-invalid @enderror" name="last_name" value="{{ old('last_name') }}" required
autocomplete="last_name" autofocus>
    @error('last_name')
      <span class="invalid-feedback" role="alert">
        <strong>{{ $message }}</strong>
      </span>
    @enderror
  </div>
</div>
```

Si on retourne voir sur http://homestead.test/register : 3
C'est tout bon !

CRUD contrôleur pour mes users

La commande «artisan» de Laravel nous facilite la vie pendant les développements, nous allons nous en servir pour générer un contrôleur CRUD (Create, Retreive, Update, Delete) pour nos utilisateurs. Ce qui va nous permettre de voir comment interagir avec les models.

```
php artisan make:controller UserController --resource
Controller created successfully.
```

J'ajoute des routes dans laravel/routes/web.php :

```
Route::resource('users', 'UserController');
```

La doc de la commande :
https://laravel.com/docs/6.x/controllers#resource-controllers
Dans cet exemple je ne gère aucun rôle, pas de sécurité, pas d'accès par niveau, ça n'est pas l'objet de cet article. Pour une utilisation réelle vous devez penser à sécuriser les accès, et gérer des niveaux d'utilisateurs. Au plus simple ça donne un admin et des utilisateurs standards, pour que seul l'admin accède à l'ensemble des informations des utilisateurs, ne puisse accéder qu'à ses propres informations.
Dans la suite de cet article, on va installer une gestion des UUIDs, on reviendra à l'utilisation du contrôleur CRUD juste après, car on va utiliser les UUIDs pour accéder aux utilisateurs.

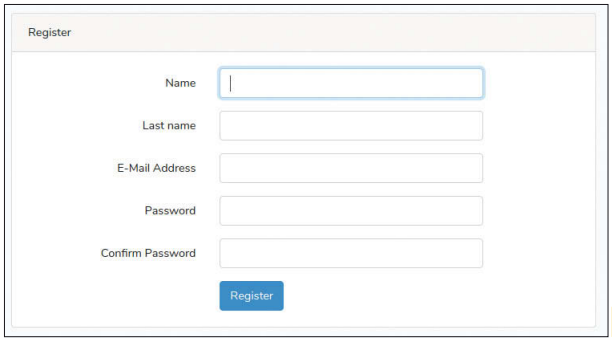
Fonctionnalités additionnelles

Avant de commencer, je vais installer quelques fonctionnalités supplémentaires, pour ça je vais voir sur https://packagist.org/ ce qui est disponible.

Attention : pensez à tout installer depuis la VM Homestead

UUID

Je cherche un moyen d'avoir des UUID au lieu des clefs uniques auto-incrément de la base de données. Je n'aime pas l'idée qu'on puisse simplement changer un ID dans l'URL et deviner des choses.
Ex : je suis l'utilisateur ID=100
https://monsitequejadore.jaime/profile/100/show



3

Je n'ai pas envie que tous les profils soient accessibles simplement en devinant les ID :
https://monsitequejadore.jaime/profile/1/show
https://monsitequejadore.jaime/profile/X/show
Évidemment on doit gérer les droits, l'affichage est autorisé uniquement à celui qui a les droits sur les ressources. Les admins ont les accès à toutes les données et les utilisateurs n'ont accès qu'à leurs informations. Toutefois je trouve les UUID plus propres.
Je cherche sur packagist :
https://packagist.org/?query=laravel%20guid
Dans le haut de la liste, il y a «emadadly/laravel-uuid» :
- 60000 installations ;
- 80 étoiles.

Je prends quelques minutes pour aller vérifier sur le repository, ici sur Github, que le projet est toujours actif. Dernière mise à jour : il y a 2 mois, le projet est maintenu, et l'onglet « issues » sur Github n'est pas inquiétant.
De manière générale je vous conseille de faire ces vérifications, ça vous évitera de commencer un projet avec du code déjà mort, ou en fin de vie. Je l'installe via composer :

```
composer require emadadly/laravel-uuid
```

ATTENTION

J'ai rencontré une erreur lors de l'installation, concernant les prérequis, au lieu d'utiliser le composer require en ligne de commande, j'ai modifié mon composer.json pour y ajouter la ligne :

```
"emadadly/laravel-uuid": "1.*"
```

Dans la section require.
Puis j'ai lancé la commande :

```
composer update
```

J'utilise Laravel 6.0 au moment d'écrire cet article, donc pas besoin de modifier le fichier config/app.php
Cette commande est optionnelle, elle permet de publier les fichiers de conf :

```
php artisan vendor:publish --provider="Emadadly\LaravelUuid\LaravelUuidServiceProvider"
Copied File [/vendor/emadadly/laravel-uuid/config/uuid.php] To [/config/uuid.php]
Publishing complete.
```

On prépare le fichier de migration pour ajouter la colonne UUID à la table users :

```
php artisan make:migration add_uuid_to_users_table --table=users
Created Migration: 2019_08_21_094431_add_uuid_to_users_table
```

Il faut alors modifier ce fichier et ajouter :

```
$table→uuid('uuid');

class AddUuidToUsersTable extends Migration
{
  /**
   * Run the migrations.
   *
   * @return void
   */
  public function up()
  {
    Schema::table('users', function (Blueprint $table) {
      $table->uuid('uuid');
    });
  }
  /**
   * Reverse the migrations.
   *
   * @return void
   */
  public function down()
  {
    Schema::table('users', function (Blueprint $table) {
      $table->dropColumn('uuid');
    });
  }
}
```

Pour finir ne pas oublier d'exécuter la migration :

```
artisan migrate

Migrating: 2019_10_09_193113_add_uuid_to_users_table
Migrated:  2019_10_09_193113_add_uuid_to_users_table (0.02 seconds)
```

Dans le fichier laravel/app/User.php on ajoute l'utilisation du trait UUID :

```
<?php

namespace App;

use Illuminate\Notifications\Notifiable;
use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Emadadly\LaravelUuid\Uuids;

class User extends Authenticatable
{
  use Notifiable;
  use Uuids;
  /**
   * The attributes that are mass assignable.
   *
   * @var array
   */
  protected $fillable = [
    'name', 'email', 'password', 'last_name',
  ];
  /**
   * The attributes that should be hidden for arrays.
   *
   * @var array
   */
  protected $hidden = [
    'password', 'remember_token', 'uuid',
  ];
  /**
   * The attributes that should be cast to native types.
   *
   * @var array
   */
  protected $casts = [
    'email_verified_at' => 'datetime',
  ];
}
```

Maintenant que j'ai mes UUIDs fonctionnels je vais créer quelques utilisateurs.
Pour tester les fonctionnalités rapidement je vais créer les fichiers nécessaires à deux actions du contrôleur, l'index pour afficher la liste des utilisateurs, et une page show pour afficher les détails.
Dans ce tuto, tous les utilisateurs ont accès à toutes les infos, je répète, à ne pas utiliser autrement qu'en local ou que pour vous. Ça n'est pas du tout adapté à un site en production.
L'idée est qu'ensuite on puisse utiliser show pour montrer son profil à l'utilisateur, puis créer un formulaire pour lui permettre de modifier ses infos…

```
laravel/resources/views/users_index.blade.php
@extends('layouts.app')
@section('content')
<div class="container">
  <div class="row justify-content-center">
    <div class="col-md-8">
      <div class="card">
        <div class="card-header">Dashboard</div>
        <div class="card-body">
          @if (session('status'))
            <div class="alert alert-success" role="alert">
              {{ session('status') }}
            </div>
          @endif
          You are logged in as: {{ Auth::user()->name }}
        </div>
        <h2>Users list</h2>
        <ul>
          @foreach ($users as $user)
            <li> <a href="{{route('users_show', ['uuid' => $user->uuid])}}">[{{$
user->uuid}}] {{$user->name}}</a> </li>
          @endforeach
        </ul>
```

```
      </div>
    </div>
  </div>
</div>
@endsection
```

laravel/resources/views/users_show.blade.php

```
@extends('layouts.app')
@section('content')
<div class="container">
  <div class="row justify-content-center">
    <div class="col-md-8">
      <div class="card">
        <div class="card-header">Dashboard</div>
        <div class="card-body">
          @if (session('status'))
            <div class="alert alert-success" role="alert">
              {{ session('status') }}
            </div>
          @endif
          You are logged in as: {{ Auth::user()->name }}
        </div>
        <h2>Users show</h2>
        {{ $user_details->name }} - {{ $user_details->last_name }} - {{ $user_details->email }}
      </div>
    </div>
  </div>
</div>
@endsection
```

Le contrôleur a été créé par la commande artisan que nous avons utilisée plus tôt, maintenant il faut mettre un peu de logique dedans. Je vous montre ici que les deux méthodes que nous allons utiliser.

laravel/app/Http/Controllers/UserController.php

Premièrement ajoutez la ligne suivante en dessous de :use Illuminate\Http\Request;

```
use App\User;
```

Pour dire au contrôleur d'utiliser notre classe User.

```
/**
 * Display a listing of the resource.
 *
 * @return \Illuminate\Http\Response
 */
public function index()
{
   $users = User::all();

   return view('users_index', ['users' => $users]);
}

/**
 * Display the specified resource.
 * @param string $uuid
 *
 * @return \Illuminate\Http\Response
 */
public function show($uuid)
{
    // $user = User::where('uuid', $uuid)->first();
    // les deux fonctionnent, ci-dessous on utilise la méthode mise à
    // disposition par le package UUID
    $user = User::uuid($uuid);
  return view('users_show', ['user_details' => $user]);
}
```

Pour terminer il faut ajouter les routes :

laravel/routes/web.php

/* pour la méthode index qui liste nos users */

```
Route::get('/users', 'UsersController@index')->name('users_index');
```

/* pour la méthode show qui nous montre les détails */

```
Route::get('/users/show/{uuid}', 'UserController@show')->name('users_show');
```

Avec ces deux méthodes et les deux vues vous avez de quoi voir la liste des utilisateurs et leur profil, accessible via l'uuid.

Webservices

En fonction du type de webservices/api avec lesquels vous allez communiquer, vous aurez besoin de certaines fonctionnalités PHP et/ou certaines librairies qui facilitent la vie.
SOAP : si les webservices que vous voulez tester utilisent SOAP, vous avez besoin de l'extension php-soap.
Pour l'installer (toujours dans la VM) :

```
sudo apt install php-soap
sudo service nginx restart
```

La doc PHP : https://www.php.net/manual/en/class.soapclient.php

Nous l'utiliserons plus tard dans cet article.
Gruzzle : pour faire des requêtes http, je vous conseille vivement la librairie Gruzzle. Pour l'installer dans votre projet (toujours depuis la VM, dans le répertoire du projet) :

```
composer require guzzlehttp/guzzle
```

Test d'un SOAP endpoint

Création d'un contrôleur, avec 2 méthodes

laravel/app/Http/Controllers/EuropePmcController.php

Vous pouvez créer le fichier à la main, via votre IDE par exemple ou utiliser la commande artisan, auquel cas vous n'aurez qu'à copier les méthodes dans la classe :

```
artisan make:controller EuropePmcController


<?php

namespace App\Http\Controllers;
```

```
class EuropePmcController extends Controller
{
  public function showMethodsAvailable()
  {
    $client = new \SoapClient('https://www.ebi.ac.uk/europepmc/webservices/soap?
wsdl', ['trace' => 1]);
    $methods = $client->__getFunctions();
    return view('europepmc_methods', ['methods' => $methods]);
  }

  public function searchPublications()
  {
    $client = new \SoapClient('https://www.ebi.ac.uk/europepmc/webservices/soap?
wsdl', ['trace' => 1]);
    $response = $client->searchPublications([
      'queryString' => 'acouphène',
      'cursorMark' => '*'
    ]);
    return view('europepmc_search_publication', ['response' => $response]);
  }
}
```

Création de deux templates

laravel/resources/views/europepmc_methods.blade.php

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
  <head>
    <meta charset="utf-8">
    <title>Available methods for Europe PMC API</title>
  </head>
  <body>
    <h1>Avalaible methods for Europe PMC API</h1>
    <div class="row">
      <ul>
        @foreach ($methods as $method)
          <li>{{ $method }}</li>
        @endforeach
      </ul>
    </div>
  </body>
</html>
```

laravel/resources/views/europepmc_search_publication.blade.php

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
  <head>
    <meta charset="utf-8">
    <title>Europe PMC API Search</title>
  </head>
  <body>
    <h1>"Accouphènes" on Europe PMC API</h1>
    <div class="row">
      <ul>
        @foreach ($response->return->resultList->result as $result)
          <li>[{{ $result-> id }}] - {{ $result->title }} - {{ $result->authorString }}</li>
        @endforeach
      </ul>
    </div>
  </body>
</html>
```

#6.3 : on ajoute les routes dans laravel/routes/web.php :

```
Route::get('/europepmc', 'EuropePmcController@showMethodsAvailable');
Route::get('/europepmc_search_publication', 'EuropePmcController@searchPublications');
```

Les tests

http://homestead.test/europepmc 4

http://homestead.test/europepmc_search_publication 5

Et voilà, vous avez créé votre premier projet Laravel, avec quelques goodies, pour aller chercher des documentations scientifiques à travers des appels SOAP sur une API publique.

Bravo !

Si vous êtes un expert Laravel et que vous souhaitez ajouter quelque chose, partager une bonne pratique, apporter une précision, écrivez-moi !

Avalaible methods for Europe PMC API 4

- searchPublicationsResponse searchPublications(searchPublications $parameters)
- profilePublicationsResponse profilePublications(profilePublications $parameters)
- getCitationsResponse getCitations(getCitations $parameters)
- getReferencesResponse getReferences(getReferences $parameters)
- getTextMinedTermsResponse getTextMinedTerms(getTextMinedTerms $parameters)
- getDatabaseLinksResponse getDatabaseLinks(getDatabaseLinks $parameters)
- getSupplementaryFilesResponse getSupplementaryFiles(getSupplementaryFiles $parameters)
- getFulltextXMLResponse getFulltextXML(getFulltextXML $parameters)
- getBookXMLResponse getBookXML(getBookXML $parameters)
- listSearchFieldsResponse listSearchFields(listSearchFields $parameters)
- getLabsLinksResponse getLabsLinks(getLabsLinks $parameters)
- getDataLinksResponse getDataLinks(getDataLinks $parameters)

"Accouphènes" on Europe PMC API 5

- [30002038] - Prise en charge de l'acouphène. - Wu V, Cooke B, Eitutis S, Simpson MTW, Beyea JA.
- [30871820] - Standardised profiling for tinnitus research: The European School for Interdisciplinary Tinnitus Research Screening Questionnaire (ESIT-SQ). - Genitsaridi E, Partyka M, Gallus S, Lopez-Escamez JA, Schecklmann M, Mielczarek M, Trpchevska N, Santacruz JL, Schoisswohl S, Riha C, Lourenco M, Biswas R, Liyanage N, Cederroth CR, Perez-Carpena P, Devos J, Fuller T, Edvall NK, Hellberg MP, D'Antonio A, Gerevini S, Sereda M, Rein A, Kypraios T, Hoare DJ, Londero A, Pryss R, Schlee W, Hall DA.
- [29541315] - [Tympanic paraganglioma mimicking cholesteatoma: about a case]. - Rafiq B, Mghari GE.
- [27836443] - [Tinnitus and temporomandibular joint: State of the art]. - Lina-Granade G, Truy E, Ionescu E, Garnier P, Thai Van H.
- [30803389] - One Size Does Not Fit All: Developing Common Standards for Outcomes in Early-Phase Clinical Trials of Sound-, Psychology-, and Pharmacology-Based Interventions for Chronic Subjective Tinnitus in Adults. - Hall DA, Hibbert A, Smith H, Haider HF, Londero A, Mazurek B, Fackrell K, Core Outcome Measures in Tinnitus (COMiT) initiative.
- [28154624] - [External auditory canal cholesteatoma]. - Azeddine L, Aabach A, Chouai M, Elayoubi F, Ghailan MR.
- [28835261] - Core Outcome Domains for early phase clinical trials of sound-, psychology-, and pharmacology-based interventions to manage chronic subjective tinnitus in adults: the COMIT'ID study protocol for using a Delphi process and face-to-face meetings to establish consensus. - Fackrell K, Smith H, Colley V, Thacker B, Horobin A, Haider HF, Londero A, Mazurek B, Hall DA.
- [30488765] - The COMiT'ID Study: Developing Core Outcome Domains Sets for Clinical Trials of Sound-, Psychology-, and Pharmacology-Based Interventions for Chronic Subjective Tinnitus in Adults. - Hall DA, Smith H, Hibbert A, Colley V, Haider HF, Horobin A, Londero A, Mazurek B, Thacker B, Fackrell K, Core Outcome Measures in Tinnitus (COMiT) initiative.
- [28848429] - Exploring Tinnitus-Induced Disablement by Persistent Frustration in Aging Individuals: A Grounded Theory Study. - Dauman N, Erlandsson SI, Albarracin D, Dauman R.
- [PMC4196832] - M'entendez-vous?: Surdité soudaine neurosensorielle à l'urgence. - Cheng AW, Mitchell Z, Foote J.
- [25426200] - [Otosclerosis: retrospective study of 36 cases]. - Bouaity B, Chihani M, Touati M, Darouassi Y, Nadour K, Ammar H.
- [25379111] - [Epidemiological, clinical and therapeutic aspects of otitis externa: about 801 cases]. - Bathokedeou A, Essobozou P, Akouda P, Essohanam B, Eyawelohn K.
- [29981034] - The association between self-reported and objective measures of health and aggregate annoyance scores toward wind turbine installations. - Michaud DS, Marro L, McNamee J.
- [27250987] - Systematic review of outcome domains and instruments used in clinical trials of tinnitus treatments in adults. - Hall DA, Haider H, Szczepek AJ, Lau P, Rabau S, Jones-Diette J, Londero A, Edvall NK, Cederroth CR, Mielczarek M, Fuller T, Batuecas-Caletrio A, Brueggemen P, Thompson DM, Norena A, Cima RF, Mehta RL, Mazurek B.
- [2256008] - [Objective tinnitus and palatal myoclonus. A new therapeutic approach]. - Le Pajolec C, Marion MH, Bobin S.
- [6069859] - [Objective tinnitus. A dura mater arteriovenous fistula] - Robier A, Loustalot B, Lapierre F, Marchal C, Laffont J, Rossazza C, Reynaud J.
- [54545203] - [The phantom hearing (acouphene) after amputation of auditory field by sound trauma] - Pazat P, Grateau P.
- [154553] - [Objective tinnitus] - Morgon A, Muller JP.
- [147922] - [Objective tinnitus caused by venous murmur] - Haguenauer JP, Gaillard J, Duquesnel J, Romanet P, Dubreuil C, Lecuire J.
- [14631455] - Brain modules of hallucination: an analysis of multiple patients with brain lesions. - Braun CM, Dumont M, Duval J, Hamel-Hébert I, Godbout L.
- [PMC4055339] - Prise en charge en cabinet des lésions cérébrales traumatiques légères chez les enfants et les adolescents. - Garcia-Rodriguez JA, Thomas RE.
- [PMC3303665] - Guide de pratique clinique pour les lésions cérébrales traumatiques légères et les symptômes persistants. - Marshall S, Bayley M, McCullagh S, Velikonja D, Berrigan L.
- [21304836] - [Not Available]. - Bertrand RA.

Dorra Bartaguiz
Crafter chez **Arolla**

Les différents tests pour les dévs

Quand on parle des tests pour les développeurs, on pense automatiquement aux tests unitaires. Mais vous savez, comme moi, que ce ne sont pas les seuls tests qu'un développeur peut écrire et réaliser. Avant d'énumérer les différents tests disponibles pour un développeur, rappelons les fondamentaux. A quoi sert un test ? Quand on se réfère au dictionnaire Larousse, un test est un essai d'un produit, d'un appareil pour vérifier son action, son fonctionnement.

niveau 100

Mais nous savons aussi qu'un test vérifie qu'un livrable répond bien au besoin métier. Il instaure donc une confiance au livrable en assurant la non-régression et en réduisant les risques. Il permet aussi de vérifier l'intégration des composants ou systèmes. Sans oublier la capacité d'un test à documenter une fonctionnalité quand il est bien nommé.

Que vous soyez novice ou expert, j'imagine que vous avez déjà entendu parler ou pu assister à des discussions autour des tests unitaires, d'intégration ou des tests systèmes ou de performance. Je vous invite tout au long de cet article à décortiquer ensemble les différents types de tests à disposition d'un développeur à travers une application simple de conversion de montant de devises. L'application et les exemples seront en C#.

Maintenant qu'on sait à quoi sert un test, commençons par énumérer les différents types. Dans les littératures on tombe souvent sur ce qu'on appelle la pyramide des tests.

PYRAMIDE DES TESTS

Présentation

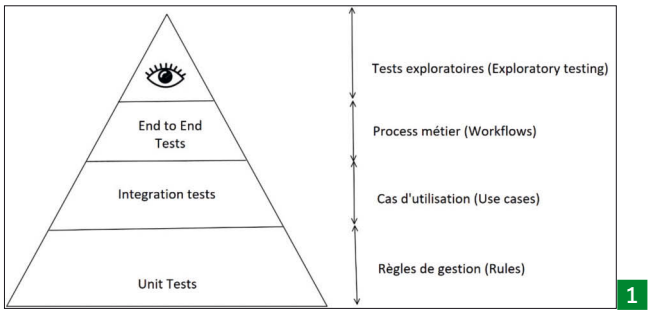
Comme le décrit Martin Fowler (https://martinfowler.com/bliki/TestPyramid.html) dans son article, la pyramide des tests permet de visualiser les familles de tests. Les niveaux ne sont pas exhaustifs, mais cela donne une idée sur les types, le nombre, le coût et le temps d'exécution.

- En effet, plus on est près de la base de la pyramide, plus :
- Le nombre de tests est important ;
- Le temps d'exécution est rapide ;
- Et le coût est faible.

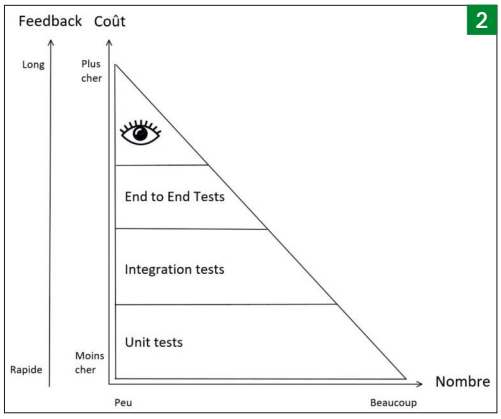
Plus on monte donc et plus le temps d'exécution est long et le coût important. Le coût englobe le coût d'écriture du test, le coût d'investigation quand le test échoue et le coût de correction des tests échoués. Et puis, quand le temps d'exécution d'un test est long, le feed-back est moins efficace puisqu'on ne pourra pas réagir au plus tôt à toute anomalie. Vous allez me dire que dans votre équipe ce n'est pas forcément le cas et que vous vivez bien avec. Peut-être, mais j'espère que vous allez être convaincus de l'inverse à la fin de cet article. Voyons déjà ce que nous propose cette pyramide : 1

De la base vers le sommet, on trouve :

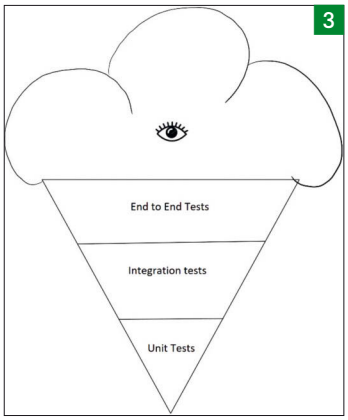
- En premier, les tests unitaires qui représentent généralement les règles de gestion de notre application,
- Puis les tests d'intégration qui représentent les use-cases de notre application,
- Ensuite, les tests end-to-end vont représenter les workflows ou process métier de notre application,



1



2



3

- Et enfin, les tests utilisateurs représentés par l'œil qui sont les tests manuels exploratoires de l'application.

On peut représenter la pyramide des tests sous forme de triangle pour mieux visualiser les axes (coût, temps, nombre). 2

Malheureusement, sont majoritaires les projets qui ont une forme de pyramide inversée appelée aussi « Cornet de glace ». On y trouve donc les tests unitaires en minorité et les tests exploratoires en grand nombre. 3

Pour chaque famille, nous verrons ensemble la définition, comment les mettre en place et les outils qui nous aideront à potentiellement les automatiser.

TESTS UNITAIRES (UNIT TEST, UT)

Définition

C'est un test qui permet de valider le comportement d'une unité de code. La définition d'une unité de code est propre à chacun. Il y a une école qui considère l'unité comme une classe et une autre qui considère que l'unité est une fonctionnalité, et donc, peut correspondre à plusieurs classes.

Quoi qu'il en soit, un test unitaire doit respecter les critères FIRST :

- Fast : il doit être rapide, on peut le lancer à tout moment sans que ça prenne du temps. Ça doit être dans l'ordre des secondes et non pas des minutes.
- Isolated / Independent : il ne faut pas qu'il y ait un ordre d'exécution des tests. Un test doit être indépendant de tous les tests qui l'entourent. Et une seule cause doit être à l'origine de son échec et non plusieurs.
- Repeatable : un test doit être déterministe. Si je le lance plusieurs fois avec une même donnée en entrée, le résultat ne doit jamais changer.
- Self validating : le test doit faire la validation lui-même. Pas besoin donc d'intervention manuelle pour savoir s'il est passé ou a échoué.
- Timely / thorough : le test doit être minutieux, car il doit être précis par rapport à ce qu'il teste. Le test doit être écrit au moment opportun. En appliquant la méthode TDD (Test Driven Development), on va écrire le test avant le code de production et donc l'écriture sera au moment où on en a besoin. Le test peut être écrit après l'écriture du code dans le cadre d'un code legacy (code sans test) par exemple.

Un test doit respecter la règle des 3A (Arrange/Act/Assert) :

- Arrange : correspond à l'initialisation du contexte du test
- Act : correspond à l'appel à la fonctionnalité à tester
- Assert : correspond à la vérification du résultat

Comment mettre en place

Les frameworks de tests unitaires sont multiples sur le marché, pour tous les goûts et pour presque tous les langages. Du Nunit pour C# au Junit pour Java jusqu'au PHPunit pour PHP ou même zunit pour COBOL. Vous n'avez plus d'excuses pour ne pas en créer ;)
La liste est longue et n'est pas exhaustive, car même pour un seul langage on peut avoir plusieurs frameworks. Prenons le cas de C#, nous trouverons Nunit, xUnit, MSTest et d'autres encore.
Via l'IDE (Integrated Development Environment), par simple commande on peut lancer un ou plusieurs tests à la fois.
Voici donc un premier test unitaire avec MSTest :

```
[TestMethod]
public void Should_convert_the_amount_when_given_currency_and_rate()
{
    // Arrange
    var eur = new Currency("EUR");
    var euroAmount = new Amount(10, eur);
    var usd = new Currency("USD");
    var eurUsdRate = new Rate(1.14);

    // Act
    var usdAmount = euroAmount.Convert(usd, eurUsdRate);

    // Assert
    var expectedAmount = new Amount(11.4, usd);
    Check.That(usdAmount).IsEqualTo(expectedAmount);
}
```

TESTS D'INTÉGRATION (INTEGRATION TEST, IT)

Définition

D'après Martin Fowler, un test d'intégration détermine si les composants d'une application continuent à fonctionner correctement quand on les connecte ensemble. Il distingue deux types de tests d'intégration :

- Narrow Integration tests (étroite couverture d'unités) : leur scope s'arrête à quelques composants tout en substituant d'autres dépendances. Si la notion de substitution ou mock vous est étrangère, je vous invite à me suivre, car je vais écrire un autre article là-dessus. Ces tests ressemblent largement à des tests unitaires au niveau de l'utilisation des frameworks et la façon de les écrire. Dans l'exemple suivant, le convertisseur appelle l'implémentation réelle d'Amount. Par contre on crée un substitut au fournisseur des taux (rates) :

```
[TestMethod]
public void Should_convert_the_amount_when_target_currency_is_different()
{
    var usdCurrency = new Currency("USD");
    var eurCurrency = new Currency("EUR");
    var eurAmount = new Amount(10, eurCurrency);
    IRates rates = Substitute.For<IRates>();
    var eurUsdRate = new Rate(1.14);
    rates.GetRateOf(usdCurrency).Returns(eurUsdRate);
    var converter = new Converter(rates);

    var convertedAmount = converter.Convert(eurAmount, usdCurrency);

    var expectedAmount = new Amount(11.4, usdCurrency);
    Check.That(convertedAmount).IsEqualTo(expectedAmount);
}
```

- Broad Integration tests (couverture étendue d'unités) : les tests qui vont avoir besoin d'instances réelles de services, potentiellement d'une base de données... L'idée est d'exécuter un scénario d'un plus large scope que celui utilisant les substituts. Ainsi on s'assure que l'intégration des composants se prépare au mieux pour éviter les surprises aux tests utilisateurs. En voici un exemple :

```
[TestMethod]
public void Should_convert_the_amount_when_target_currency_is_different ()
{
    Currency usdCurrency = new Currency("USD");
    Currency eurCurrency = new Currency("EUR");
    Amount eurAmount = new Amount(10, eurCurrency);
    IRates rates = new Rates();
    Converter converter = new Converter(rates);

    Amount convertedAmount = converter.Convert(eurAmount, usdCurrency);

    Amount expectedAmount = new Amount(11.4, usdCurrency);
    Check.That(convertedAmount).IsEqualTo(expectedAmount);
}
```

Vous remarquerez que la différence réside dans l'utilisation d'un substitut pour le narrow test et l'utilisation de l'implémentation réelle dans le broad test, ce qui fait que le deuxième test est plus profond que le premier.
Un test d'intégration peut très bien être aussi sous format Gherkin. Il s'agit d'un formalisme "given/when/then" qui structure et rapproche le test au langage humain. Ce formalisme facilite l'écriture des tests avec le PO (Product Owner) ou les utilisateurs puisqu'il se rapproche d'une spécification ou d'une documentation. On l'utilise souvent dans le cadre d'une méthodologie BDD (Behavior Driven Development).
Ci-dessous, vous trouverez un exemple de scénario. Chaque ligne correspond à une "step definition" qui sera associée à une méthode (et donc une implémentation). Cette dernière sera exécutée quand l'étape (appelée aussi step) sera atteinte et ainsi elle détermine si le test a réussi ou a échoué.
Scenario: Convert amount if the rate is found

```
Given I have 100 EUR
And I have entered USD as target currency
And the rate is 11.4
When I convert amount
Then the converted amount should be 114 USD
```

Scenario: Convert amount if the rate is not found

```
Given I have 100 EUR
And I have entered TOG as unfound target currency
When I convert amount
Then I should get an error
```

Comment mettre en place

Les tests d'intégration s'écrivent avec les mêmes outils que ceux des tests unitaires. On peut donc utiliser les substituts pour un ou plusieurs composants et utiliser les implémentations de production pour d'autres.
Si on ne veut pas utiliser les substituts alors que notre code doit accéder à une dépendance externe, on a plusieurs pistes.

- Cas où on doit appeler une API REST par exemple. Dans ce cas, il nous faut un environnement prêt ("un environnement up" comme on dit dans le milieu) avec la version à tester du service. Bien sûr ce n'est pas le service de la production, mais ça peut être la même version.
- Cas où on a besoin d'accéder à une base de données. Dans ce cas il nous faut une base de données pour les tests. Cette base va être mise à jour potentiellement tous les soirs ou périodiquement. Il faudra définir la stratégie au niveau de l'équipe et l'importance d'un tel investissement, car il ne faut pas oublier que le schéma de la base évolue sans cesse et les données aussi. Comme le test doit être dépendant d'eux, il faudra penser à insérer les données du test, le faire tourner et puis nettoyer les données afin qu'au lancement suivant le même test ne soit pas altéré.
  Ne vous inquiétez pas, il y a une autre solution pour lancer le test sans devoir déployer une base. On peut utiliser dans ce cas les bases in-memory. Il s'agit d'une base en mémoire qui sera supprimée quand tous les tests auront fini leur exécution. On ne règle pas le problème des schémas de la base qui évoluent, mais on aura un problème en moins.

Pour le format Gherkin, on peut très bien utiliser SpecFlow (https://specflow.org/) pour .Net ou Cucumber (https://cucumber.io/) pour java.

TESTS DE BOUT EN BOUT (END TO END, E2E)

Définition

Ces tests permettent de vérifier que l'ensemble des composants d'une fonctionnalité fonctionnent de bout en bout, de la partie IHM (Interface Homme Machine) aux dépendances externes (base de données ou service externe comme une API REST). On utilise généralement ces tests pour les scénarios critiques d'une application. Leur coût est important comme montré dans la pyramide des tests d'où le fait que leur nombre est inférieur à celui des tests d'intégration et unitaire. Il vaut mieux donc bien les cibler avec le PO ou les utilisateurs pour mieux les identifier et ne pas perdre du temps à créer des tests pour des scénarios non critiques. Gardez en tête que si les choses sont bien faites alors les scénarios non critiques seront couverts par les tests d'intégration et unitaires.

Comment mettre en place

Il y a plusieurs outils permettant d'écrire des tests E2E. Selenium (https://www.seleniumhq.org) fait partie des plus populaires sur le marché. Lors de l'écriture de cet article, j'ai développé une application exemple en dotnet core. Et l'utilisation de Selenium est assez simple. Il suffit de référencer ces packages nugget :

```
<PackageReference Include="Selenium.Support" Version="3.141.0" />
<PackageReference Include="Selenium.WebDriver" Version="3.141.0" />
<PackageReference Include="Selenium.WebDriver.MicrosoftDriver" Version="17.17134.0" />
```

Suivant le navigateur à utiliser pour les tests, on peut rajouter les packages correspondants.
Dans mon cas, j'ai utilisé le EdgeDriver.

```
using (var driver = new EdgeDriver(Path.GetDirectoryName(Assembly.GetExecuting
Assembly().Location)))
```

Ensuite, il faut naviguer vers la page qu'on veut tester avec la méthode :

```
driver.Navigate().GoToUrl(@"https://mysite/Home");
```

Pour paramétrer la valeur des champs, il suffit de récupérer l'élément via son identifiant et insérer la valeur :

```
var element = driver.FindElement(By.Id(elementId));
element.SendKeys(elementValue);
```

Pour récupérer la valeur d'un champ, par exemple, on appelle tout simplement la méthode GetAttribute("value") de l'élément :

```
var convertedAmount = driver.FindElement(By.Id("ConvertedAmount"));
var convertedValue = convertedAmount.GetAttribute("value");
```

Il n'y a plus qu'à vérifier la valeur via une assertion.

TESTS EXPLORATOIRES

Définition

Les tests exploratoires sont des tests exécutés manuellement. Leur rôle principal est de trouver les bugs qui ne sont pas détectés par les autres familles de tests afin d'améliorer la stabilité et l'utilisabilité de l'application. Mais au-delà de la validation, ces tests permettent aussi l'apprentissage et la compréhension du produit. Ça per-

met ainsi d'avoir une idée du produit et de trouver de nouvelles fonctionnalités. Généralement, ils ne sont pas basés sur des documents et donc non dirigés. Leur temps d'exécution peut durer des jours et des jours.

POUR ENCORE PLUS DE QUALITÉ

Quand on veut viser plus de qualité pour notre application, on peut réaliser d'autres types de tests qui pourront se greffer à tous les niveaux de la pyramide. Il s'agit plus d'une caractéristique à mon sens. On trouve donc :

- L'automatisation des tests ;
- Les tests de performance ou de charge ;
- Les tests de vérification en service régulier.

Automatisation des tests

Une fois les tests créés, on peut les laisser dans le projet et les lancer manuellement à chaque modification du code, comme on peut automatiser leur exécution en local en utilisant le live-testing de votre IDE (si l'option existe) ou dans un terminal à côté. Il ne faut pas oublier non plus la CI (Continuous integration comme Jenkins ou TeamCity). Il suffit de configurer une step de tests après la compilation. Ça permettra donc à chaque push de lancer tous les tests et voir ainsi l'état de l'usine au fur et à mesure du développement.

Le live-testing est très utile lors du développement puisqu'à chaque enregistrement ou compilation, tous les tests présents et concernés par la modification vont être lancés. On voit ainsi si une régression est apportée à condition qu'on ait, bien sûr, une bonne couverture de tests. La photo ci-dessous (live-testing de Visual Studio) présente un code couvert par sept tests et que tous les tests passent avec succès.

```
public class Rate
{
    private readonly decimal _rate;

    8 références | 7/7 ayant réussi | Dorra BARTAGUIZ, il y a 4 jours | 1 auteur, 1 modification
    public Rate(decimal rate)
    {
        _rate = rate;
    }

    1 référence | Dorra BARTAGUIZ, il y a 4 jours | 1 auteur, 1 modification
    public decimal Multiply(decimal value)
    {
        return _rate * value;
    }
```

Si on décide de mettre en commentaire une ligne, comme le montre la photo ci-dessous, à l'enregistrement ou à la compilation, tout de suite le statut change en indiquant que certains tests échouent à cause de la dernière modification :

```
public class Rate
{
    private readonly decimal _rate;

    8 références | 3/7 ayant réussi | Dorra BARTAGUIZ, il y a 4 jours | 1 auteur, 1 modification
    public Rate(decimal rate)
    {
        //_rate = rate;
    }

    1 référence | Dorra BARTAGUIZ, il y a 4 jours | 1 auteur, 1 modification
    public decimal Multiply(decimal value)
    {
        return _rate * value;
    }
```

Tests de performance

Ils sont aussi appelés tests de charge ou "stress tests". Ils permettent de s'assurer que l'application supporte bien une certaine charge (de données, d'appels …) en restant réactive et stable. Avant de commencer à mettre en place ce type de test, il faut commencer par définir les objectifs de ces tests et pourquoi on a besoin de mettre en place des tests de performance.

- Quel est le temps de réponse maximal attendu en secondes ou millisecondes (temps de réponse serveur et temps d'affichage) ? Le temps de réponse exigé d'une page web ne sera pas forcément le même que celui d'un écran d'un client lourd.
- Quels sont les modules ou composants qui rentrent dans le scope des tests de performance ?
- Quel est le nombre d'utilisateurs sur tel ou tel module ou composant ? Deux informations sont intéressantes dans ce cas : le cas nominal (activité classique) et le cas critique (pic d'activités).

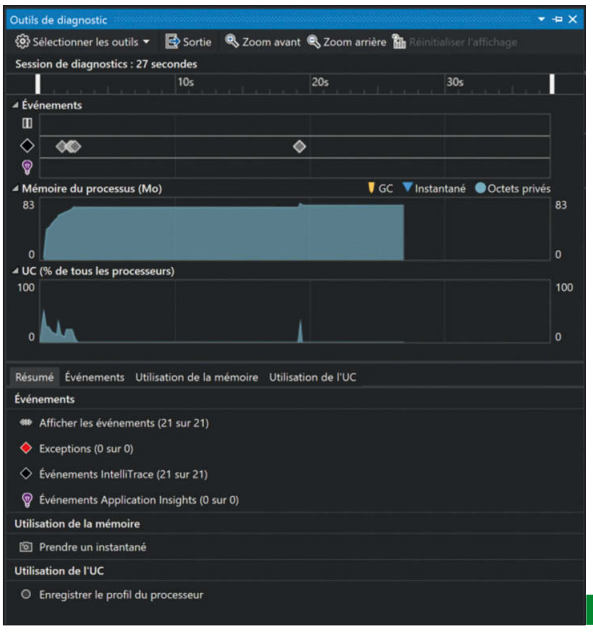
Les tests de performance peuvent, bien sûr, être automatisés et lancés dans l'usine d'intégration continue afin de vérifier la non-régression des performances de l'application.

Par exemple, on peut commencer par utiliser l'outil DevTools du navigateur s'il s'agit d'un service web pour calculer le temps de réponse. La photo 4 montre bien que l'appel l'API Convert a mis 180,51 ms. Ça donne ainsi une idée, par rapport aux objectifs déterminés en amont, s'il faut mener ou non une action d'optimisation du code en question.

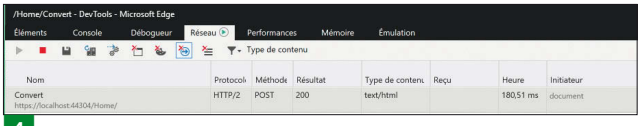
Votre IDE peut aussi vous aider à investiguer, la photo 5 montre l'outil de diagnostic de Visual Studio lors du debug de l'application. Ça affiche les évènements lancés, l'état d'utilisation de la mémoire et du CPU. Ainsi on peut prendre des actions si besoin pour agir au mieux pour l'optimisation de notre application.

Pour des tests plus poussés, on peut aller au-delà en utilisant des outils plus poussés tels qu'Apache Bench (https://httpd.apache.org/docs/2.4/programs/ab.html) ou Siege (https://www.joedog.org/siege-home/) pour la partie serveur.

Côté client web on trouve, par exemple, SiteSpeed (https://www.sitespeed.io/) qui analyse votre site par rapport aux bonnes pratiques en matière de métriques de performance ou WebPageTest



5



4

(https://www.webpagetest.org/) qui est gratuit. Il teste une page sur le navigateur choisi et remonte des informations précises sur les performances de la page. 6

Tests de vérification en service régulier

Une fois que l'application est livrée en production, il est important de vérifier qu'elle est "up" et fonctionnelle. La vérification peut se faire manuellement en lançant, par exemple, tous les matins ou x temps l'application, mais elle peut très bien être automatisée. On peut bien sûr utiliser des tests End To End avec Selenium, par exemple, pour voir si la page se charge bien. En plus, on peut aussi faire des "health checks" de nos APIs. L'idée en tout cas est de vérifier que le service répond bien avec un statut 200.

Dans l'application exemple créée pour cet article, j'ai juste créé un test qui fait appel à notre page d'accueil et vérifie qu'elle répond bien HttpStatusCode.OK qui le code http 200.

```
[TestMethod]
[TestCategory("Health check test")]
public void Check_that_web_site_is_up()
{
    const string url = @"https://mysite/Home";
    bool isServiceUp;
    try
    {
        var myRequest = (HttpWebRequest)WebRequest.Create(url);
        var response = (HttpWebResponse)myRequest.GetResponse();
        isServiceUp = response.StatusCode == HttpStatusCode.OK;
    }
    catch (Exception)
    {
        isServiceUp = false;
    }

    Check.That(isServiceUp).IsTrue();
}
```

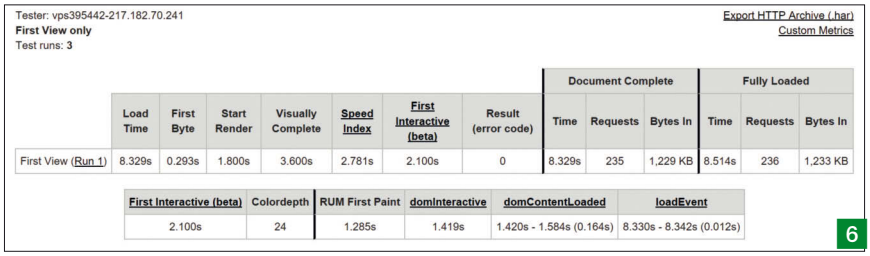
ET PLUS ENCORE DE QUALITÉ

Quadrant test agile

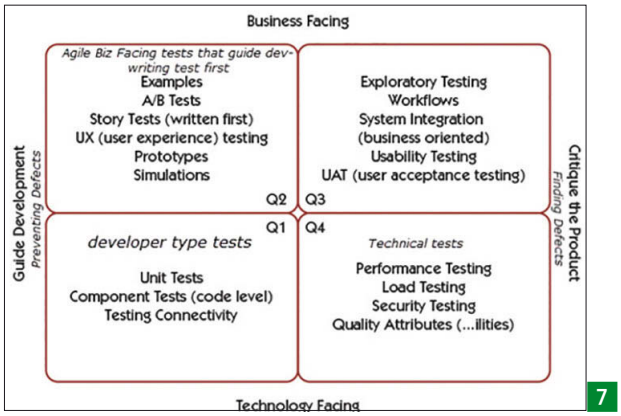
Initié par Brian Marick et repris par Lisa Crispin et Jane Gregory dans "Agile Testing" (https://agiletester.ca/agile-testing/), ce quadrant présente les différents types de tests dont a besoin un projet agile. Ce quadrant peut être visualisé de deux façons différentes, mais complémentaires. De part et d'autre de la ligne horizontale, on trouve une vision métier (Business facing Q2 et Q3) dont les tests sont généralement conçus par les testeurs et une vision technique (Technology facing Q1 et Q4) dont les tests sont implémentés par les développeurs. Et par rapport à la ligne verticale, à gauche (Q1 et Q2) on trouve les tests pour guider l'équipe et prévenir les défauts en vérifiant l'attendu (Check for the expected) et les bugs et à droite (Q3 et Q4) les tests pour critiquer le produit et détecter les éventuels défauts en cherchant l'inattendu (Looking for the unexpected). 7

Property based testing

Quand on crée des tests, généralement, on fige les données d'entrée (inputs) pour vérifier que le résultat est bien calculé. Avec le



6



7

property based testing, on va vérifier les propriétés du code avec des valeurs multiples et aléatoires en entrée. Prenons le cas où on a 10€ qu'on veut convertir en dollars avec un taux de 1.14. On doit avoir en résultat 11.4$. On parle donc d'input et d'output. Ayant ces données d'entrée (10€ et 1.14), on doit avoir le résultat 11.4$. En property based testing, on parle de données d'entrée (multiples et aléatoires) et de comportement (behavior) attendu. L'idée est donc de spécifier le comportement de la règle de gestion au sein du test. Classiquement notre test ressemble à ceci :

```
[TestMethod]
[TestCategory("Unit test")]
public void Should_convert_the_amount_when_having_currency_and_rate()
{
    Currency eur = new Currency("EUR");
    Amount euroAmount = new Amount(10, eur);
    Currency usd = new Currency("USD");
    Rate eurUsdRate = new Rate(1.14);
    Amount usdAmount = euroAmount.Convert(usd, eurUsdRate);

    Amount expectedAmount = new Amount(11.4, usd);
    Check.That(usdAmount).IsEqualTo(expectedAmount);
}
```

Avec le property based testing notre test ressemblera plus à ça :

```
[Property]
public Property Should_convert_the_amount_using_rate(decimal amountValue, decimal rate)
{
    Func<bool> codeToCheck = () =>
    {
        Currency eur = new Currency("EUR");
        Amount euroAmount = new Amount(amountValue, eur);
        Currency usd = new Currency("USD");
        Rate eurUsdRate = new Rate(rate);
```

```
        Amount usdAmount = euroAmount.Convert(usd, eurUsdRate);
        return usdAmount.HasCurrency(usd) && usdAmount.HasValue(amountValue * rate);
    };

    return codeToCheck.When(rate != 0);
}
```

Ce test sera donc lancé plusieurs dizaines, voire centaines de fois en ayant à chaque fois des valeurs différentes pour le montant et le taux. L'image 8 montre que le test est lancé 100 fois avec des valeurs différentes et tous ont réussi.
En cas d'erreur, un message est affiché pour annoncer le nombre de tests passés avant d'atteindre l'échec. 9
Sur le marché et selon le langage, on trouve beaucoup de librairies pour le property based testing. Pour cet exemple, j'ai choisi FsCheck (https://fscheck.github.io/FsCheck/) qui fonctionne aussi bien avec C#, F# ou VB.net, mais pour Java on trouve JUnitQuickcheck (http://pholser.github.io/junit-quickcheck/site/0.8/junit-quickcheck-core/jacoco/com.pholser.junit.quickcheck.runner/JUnitQuickcheck.java.html)

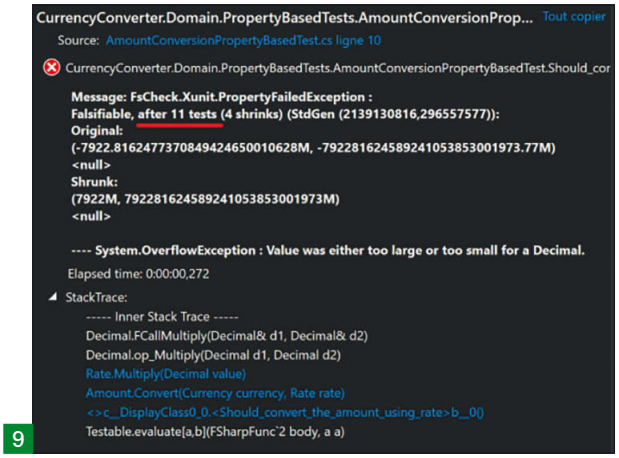
Mutation testing

La technique de « mutation testing » permet de vérifier si nos tests sont de bonne qualité.
Certains sont convaincus qu'une bonne couverture de code (vers les 80%) suffit comme indicateur de la bonne qualité du code. Sauf qu'on oublie que c'est une métrique et on peut très bien jouer avec pour avoir le résultat qui nous arrange. Prenons un exemple :

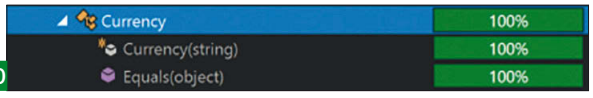
```
[TestMethod]
public void Should_currency_be_equal_when_having_same_name()
{
    Currency eur = new Currency("EUR");
    Currency eurCurrency = new Currency("EUR");
    Check.That(eur).IsEqualTo(eurCurrency);
}
```

8

9



10



Quand on lance le calcul de la couverture de code pour ce test, on a un résultat à 100 % de la classe. 10
Si je remplace le code de mon test par ce qui suit, la couverture de code reste toujours à 100% et pourtant je n'ai pas d'assertion.

```
[TestMethod]
public void Should_currency_be_equal_when_having_same_name()
{
    Currency eur = new Currency("EUR");
    Currency eurCurrency = new Currency("EUR");
    var areEqual = eur.Equals(eurCurrency);
}
```

Pour éviter de tomber dans ce genre de piège, la mutation testing vient pour créer des mutants de notre code. Elle va donc lancer le test avec plusieurs versions (mutants) du même code. Chaque mutant correspond à une modification simple comme changer une addition par une soustraction ou une égalité par une différence ou modifier les valeurs de constantes…
Les outils et librairies de mutation testing sont nombreux sur le marché. On peut donc utiliser Stryker (https://stryker-mutator.io/) ou NinjaTurtles (http://www.mutation-testing.net/) pour .Net ou PI (http://pitest.org/) pour Java et tant d'autres.
On peut le faire aussi manuellement. Rappelons-nous, un mutant c'est un code qui contient une petite modification. Donc on peut juste modifier le code et relancer les tests tout simplement. Ainsi on transforme le code en mutant manuellement. Une fois le code modifié et les tests lancés, s'il n'y a pas de test rouge, cela signifie deux choses :

- Soit le code modifié n'est pas couvert par des tests et donc c'est l'occasion d'en rajouter.
- Soit les tests ne sont pas pertinents et donc il vaut mieux les corriger.

Attention, il ne faut pas oublier de remettre le code de production comme il était bien sûr ;)

AVANT DE FINIR

Vous pouvez trouver la solution avec tous les tests sur mon Github https://github.com/iAmDorra/CurrencyConverter. Pour faciliter le parcours et l'identification des tests, je les ai catégorisés. Vous trouverez donc les unit tests, narrow integration tests, broad integration tests, end to end tests, les health check tests et les property-based tests.

CONCLUSION

J'espère que vous avez une meilleure connaissance des différents tests qu'un développeur peut implémenter. A vous de jouer avec maintenant. N'oubliez pas de créer au minimum autant de tests unitaires que de règles de gestion dans votre application (n'oubliez pas les cas limites). Créez des tests d'intégration pour éviter au plus tôt les problèmes d'incompatibilité de composants, et, surtout choisissez toujours en équipe les tests E2E à écrire, car ils sont plus coûteux à créer et maintenir. Mais avant de vouloir appliquer tous les tests sur votre projet, comme l'a bien présenté Nicolas Fédou lors de sa conférence à Flowcon'18, commencez par réfléchir à la stratégie de tests pour en faire moins, mais mieux. C'est la meilleure façon de montrer qu'une application est de qualité suffisante pour aller en production. Il s'agit donc d'un raisonnement avec des choix et des renoncements pour développer un meilleur produit agile.
See you ;) •

Jonathan Antoine
Développeur passionné
@ Infinite Square
blogs.infinitesquare.com

INFINITE SQUARE

OpenIdConnect

Le protocole Oauth 2.0 commence à être bien connu de nos jours. Il permet de gérer la notion d'autorisations dans vos applications. Cependant, il lui manque la partie identification/authentification. C'est justement ce que normalise OpenIdConnect, aussi appelé OIDC. Tout étant basé sur REST via du HTTPS bien sûr. Dans cet article, je vous présente tout cela ainsi qu'une mise en place rapide sur une API Asp.net Core !

niveau 200

Pour rappel, on distingue 2 notions :

- Identification et **authentification** : qui se connecte à mon application ? dans les faits, quelle est son identité et quel est le processus de vérification de cette identité ?
- Autorisation : qu'est-ce que peut faire un utilisateur identifié ?

Le fonctionnement théorique d'OIDC est le suivant :

- Un **client** (une appli mobile, une fonction Azure, etc.) veut accéder à une ressource.
- L'utilisateur va s'identifier auprès d'un **serveur d'identification et autorisation** (dans une mire de connexion) qui lui retourne un jeton d'échange.
- Le jeton d'échange est utilisé pour obtenir un jeton d'accès.
- Avec ce jeton d'accès, le client accède à cette ressource sur l'application **possédant la ressource** via l'application hébergeant la ressource (**resource server**).

Il est donc tout à fait possible d'avoir votre **serveur d'autorisation OIDC complètement décorrélé de votre serveur hébergeant les ressources**. Cela est très intéressant pour mutualiser l'accès à plusieurs serveurs hébergeant des ressources (plusieurs API) afin de mutualiser la couche d'identification et de permettre des choses sympas comme le SSO. Mais rien ne vous empêche non plus d'avoir les deux (APIs et OIDC) en même temps (c'est ce que nous verrons plus tard dans cet article).

Sous le capot, cela se base sur plusieurs informations techniques :

- **client_id** : identifiant de l'application cliente souhaitant accéder à l'application. Chaque application souhaitant s'intégrer à votre système OIDC est normalement déclarée préalablement et en possède un. Il est aussi possible de mettre en place de l'enregistrement dynamique d'application cliente ou de désactiver cette vérification et d'accéder anonymement à vos ressources (d'un point de vue applicatif).
- **client_secret** : une sorte de "mot de passe" de votre application cliente associée au client Id.
- **access_token** : un jeton donné par le serveur OIDC à votre application cliente pour valider son identité auprès des serveurs fournissant les ressources. Il possède une durée de vie limitée. Ce token est dans le format standard JWT (3 parties séparées par des points, le tout encodé en base64).
- **refresh_token** : un jeton permettant d'obtenir un nouvel access_token lorsque le premier obtenu est périmé. Ce mécanisme n'est pas toujours autorisé.
- **id_token** : un jeton contenant les informations sur l'utilisateur. La documentation officielle (https://openid.net/specs/openid-connect-core-1_0.html#IDToken) sur le sujet décrit son contenu.

Il est à noter que dans certains cas, il est possible de recevoir deux tokens (id_token et access_token) dans la même réponse du serveur OIDC.

Il existe ensuite plusieurs façons de s'identifier auprès de votre serveur OIDC, les plus courantes étant (liste non exhaustive !) :

- **Implicit flow** : dédié aux applications sans backend (SPA, etc.). L'utilisateur se connecte sur une mire de connexion et le client récupère un jeton d'accès. Une grande partie de la sécurité de ce flow réside dans la présence et l'utilisation du navigateur web. Dans ce contexte, il est recommandé de ne pas permettre de rafraîchir le token.
- **Authorization code flow** : dédié aux applications "serveur" (ou natives avec PKCE). L'utilisateur se connecte sur une mire de connexion du serveur OIDC. Le serveur d'authentification appelle une url de callback sur le serveur "client" en fournissant un jeton d'autorisation que celui-ci peut échanger contre un jeton d'accès à l'API.
- **Resource Owner Password Credentials / Password flow** : dédié aux cas où vous avez une relation de confiance très forte avec l'application cliente ou pour des scenarii de migration depuis une authent HTTP Basic. C'est l'application cliente qui présente une mire de connexion et envoi le login/mot de passe de l'utilisateur à votre serveur d'authentification. Cela a beaucoup de désavantages et il faut bien peser le pour et le contre avant de l'utiliser.

Le flow de connexion est le choix le plus important qui sera fait par l'application cliente. Les différents flows font l'objet d'un traitement détaillé plus bas dans cet article.

Finalement, il faut noter que chaque demande de connexion est faite pour **un périmètre donné (des scopes** dans la littérature technique). Le standard OpenIdConnect en définit quelques-uns :

- profile : les infos générales sur l'utilisateur.
- email : email (vérifié ou non) de l'utilisateur.
- phone : téléphone (vérifié ou non) de l'utilisateur.
- openid : **obligatoire** pour identifier un utilisateur :)

Sécuriser une API Asp.Net Core avec OpenId Connect

La mise en place d'un serveur OIDC est, on peut l'imaginer, assez complexe. Pour faciliter cela, j'utilise la librairie OpenIddict maintenue activement par le français Kévin Chalet. Mon serveur OIDC sera aussi mon serveur hébergeant les APIs : tout est au même endroit.

La mise en place que je propose repose sur ces briques techniques :

- Asp.Net Core 2.X,
- Entity Framework Core 2.X

La librairie repose sur le système d'identité / sécurité d'Asp.Net Core et s'intègre très bien avec.
Dans la suite de l'article, nous verrons comment mettre en place le mode d'authentification "Resource Owner Password Credentials".

Configuration du contexte Entity Framework

La première étape consiste à faire dériver votre contexte du type de base proposé par Asp.Net Core Identity : IdentityDbContext. Dans mon cas, je souhaite que les clefs primaires de mes objets soient des long (et pas des strings comme par défaut) et j'utilise donc son pendant générique.

```
1  public abstract class DbContext :
2    IdentityDbContext<User, Role, long>
3  {
4  }
```

Mes entités User et Roles sont des classes dérivant de IdentityUser<long> et IdentityRole<long>.
Il faudra enfin **laisser OpenIddict enregistrer ses types dans le DbContext** au moment de l'enregistrement de ce dernier en utilisant la méthode UseOpenIddict :

```
1  services.AddDbContext<DbContext>(options =>
2  {
3     options.UseOpenIddict<long>();
4  });
```

Configuration du pipeline HTTP Asp.Net Core

La suite se situe au niveau de la méthode ConfigureServices de votre classe de Startup. On commence par ajouter la couche Identity utilisant EF :

```
1  services.AddIdentity<User, Role>()
2     .AddEntityFrameworkStores<DbContext>()
3     .AddDefaultTokenProviders();
```

Ensuite, on ajoute la partie authentification en spécifiant que le schéma par défaut est de passer via les token :

```
1  services.AddAuthentication(options =>
2  {
3     options.DefaultScheme = OAuthValidationDefaults.AuthenticationScheme;
4  }).AddOAuthValidation();
```

On configure ensuite la partie Identity selon nos goûts et les besoins du projet. Il est aussi important de remplacer le nom des Claims utilisés par défaut par la partie Identity d'Asp.Net Core (ceux de WS Federation avec des valeurs de ce genre : http://schemas.xmlsoap.org/ws/2005/identity/claims/name) par ceux utilisés par OpenIdConnect (plus simples : name, role, etc.).C'est ce que font les 3 dernières lignes de l'extrait de code :

```
1  services.Configure<IdentityOptions>(options =>
2  {
3     // Password settings
4     options.Password.RequireDigit = false;
5      ....
6
7     options.ClaimsIdentity.UserNameClaimType = OpenIdConnectConstants.Claims.Name;
8     options.ClaimsIdentity.UserIdClaimType = OpenIdConnectConstants.Claims.Subject;
9     options.ClaimsIdentity.RoleClaimType = OpenIdConnectConstants.Claims.Role;
10 });
```

Finalement on doit ajouter la partie OpenIddict en elle-même. C'est ici que l'on configure ce qui est autorisé ou non d'un point de vue OpenIddict et notamment penser à enregistrer les différents scopes existants.

```
1  services.AddOpenIddict()
2  .AddCore(options =>
3  {
4   options.UseEntityFrameworkCore()
5    .UseDbContext<TDbContext>()
6    .ReplaceDefaultEntities<long>();
7  })
8  .AddServer(options =>
9  {
10  // nécessaire pour enregistrer les types OpenIddict
11  options.UseMvc();
12
13  // les scopes définis
14  options.RegisterScopes(
15   OpenIdConnectConstants.Scopes.OpenId,
16   OpenIdConnectConstants.Scopes.OfflineAccess);
17
18  // le endpoint de connexion
19  options.EnableTokenEndpoint("/connect/token");
20
21  // permettre de se connecter en ressource owned grant..
22  options.AllowPasswordFlow();
23
24  // options.AcceptAnonymousClients();
25 });
```

La dernière étape consiste à ajouter le middleware d'authentification. Ajoutez cette ligne au tout début de la méthode Configure de votre Startup :

```
1  public void Configure(IApplicationBuilder app, IHostingEnvironment env)
2  {
3     app.UseAuthentication();
4  }
```

D'accord, c'est un peu verbeux mais on en a terminé pour le moment :)

Mise en place du contrôleur d'authentification

Il faut maintenant **créer un contrôleur (controller) permettant d'échanger les informations de connexion de l'utilisateur** contre un token. Un sample très complet est proposé sur le GitHub OpenIddictConnect. Globalement, on va utiliser les méca-

nismes d'Asp.Net Core Identity pour vérifier les informations de connexion de l'utilisateur, configurer un ticket d'authentification et le donner à la méthode SignInAsync (présente sur tous les controllers Asp.Net) pour le laisser faire la connexion de l'utilisateur sous le capot.

Sécuriser un controller

Pour sécuriser l'accès à un controller, il suffit alors de lui apposer les attributs Authorize *classiques* en spécifiant éventuellement un rôle. Voici un exemple sur une action :

```
1  [Authorize(
2    AuthenticationSchemes = OAuthValidationDefaults.AuthenticationScheme,
3    Roles = "ADMIN")]
4  public ActionResult ActionWithAdminAuthorize(){}
```

Sécuriser vos APIs pour qu'elles soient utilisées par d'autres applications

Nous avons vu comment sécuriser une API en suivant le protocole OpenId Connect. La démarche était orientée pour une connexion faite par des utilisateurs *humains* (ou développeurs). Voyson à présent **comment sécuriser vos APIs pour qu'elles soient utilisées par d'autres applications.**

Ce scénario permet de répondre à des usages assez fréquents :

- Un WebJob utilise votre API à heure régulière pour effectuer des traitements en batch.
- Une AzureFunction utilise votre API en lecture/écriture pour effectuer des opérations.

La notion importante à retenir ici est que **vous ne serez pas dans le contexte d'un utilisateur** : chaque opération sera faite en tant qu'application et pas en tant qu' utilisateur. Attention donc lorsque vous exposez vos APIs de ne pas avoir cette notion d'identité *humaine*.

Définition de l'application

Pour se connecter en tant qu'application, on va utiliser le *grant_type de type* "client_credentials". Celui-ci n'est pas lié à OpenId Connect en lui-même mais au protocole Oauth2.0 dont il dérive et permet d'utiliser une application comme identité (plus d'infos sur https://blogs.infinitesquare.com/posts/web/openid-connect-clientid-et-clientsecret-oidc). ATTENTION, ce flow est uniquement à utiliser dans un contexte où vous pouvez garantir la sécurité du couple client_id/client_secret. Dans ce mode, l'utilisation de refresh_token n'est pas possible car il ne fait pas sens : l'application a déjà tout ce qui est nécessaire pour redemander un jeton d'accès si le premier obtenu expire.

Lors de l'enregistrement d'une application permettant de se connecter à l'API sur votre serveur OIDC il faut **lui créer un client_id/client_secret** et spécifier que celle-ci peut utiliser le grant type susnommé. En reprenant le code de https://blogs.infinitesquare.com/posts/web/securiser-une-api-avec-openid-connect, il faut donc ajouter cette ligne aux permissions :

```
1  OpenIddictConstants.Permissions.GrantTypes.ClientCredentials
```

Le contrôleur d'échange de token

Il s'agit de la partie la plus importante : permettre d'échanger une demande de connexion pour un jeton d'accès. **Pour cela, on repart de la définition de base d'OpenIddict de l'action dédiée à ce processus et on utilise la méthode IsClientCredentialsGrantType pour n'accepter (dans mon cas) que les demande de connexion avec ce grant_type.** Il ne faudrait pas faire ce return si vous souhaitez autoriser d'autres types de connexion.

```
1  public async Task<IActionResult>
2    ExchangeAsync([FromBody] OpenIdConnectRequest request)
3  {
4    if (!request.IsClientCredentialsGrantType()){
5      return;
6    }
7  }
```

Une fois que nous sommes sûr d'être sur une demande de connexion provenant d'une application, il va falloir faire plusieurs choses :

- **Créer une identité (ClaimsIdentity)** : on ne peut pas utiliser le SignInManager d'Asp.Net Core vu qu'il n'y a pas d'utilisateur *humain* dans ce cadre.
- **Ajouter un claim de type *Subject* obligatoire** pour un bon fonctionnement avec la couche sous-jacente de sécurité Asp.Net Core. J'utilise la valeur de client_id que j'ai dans la requête.
- Créer un ticket d'authentification en utilisant l'identité créée.
- **S'assurer de mettre les bons scopes sur ce ticket.** Ici je mets tout ceux passés dans la demande de connexion mais les samples OpenIddict vous donnent un très bon exemple pour ne garder que ceux que vous acceptez réellement ici à base de méthode Intersect.
- Utiliser le ticket pour demander une connexion via la méthode de base *SignIn* des contrôleurs.

Il n'est **pas nécessaire de vérifier la véracité de la demande de connexion** dans le code du contrôleur car l'infrastructure OpenIddict mise en place s'occupe déjà de cela telle que configurée :

- Le client_id passé existe bien et le client_secret correspond bien.
- L'application correspondant à ce client_id a bien le droit d'utiliser ce mode de connexion.
- Les scopes passés en paramètre existent bien.
- L'application correspondant à ce client_id a bien le droit d'utiliser ces scopes.

Le code à produire reste alors assez succinct :

```
1  // on crée une identité connectée via OpenId Connect.
2  var identity =
3    new ClaimsIdentity(OpenIdConnectServerDefaults.AuthenticationScheme);
4  var claimsPrincipal = new ClaimsPrincipal(identity);
5
6  // On ajoute le claim subject obligatoire
7  identity.AddClaim(
8
9    new Claim(OpenIdConnectConstants.Claims.Subject, request.ClientId));
10
11 // Creation du ticket d'authentification
12 var ticket = new AuthenticationTicket(
13   claimsPrincipal,
14   new AuthenticationProperties(),
```

```
15    OpenIdConnectServerDefaults.AuthenticationScheme);
16
17 // Application des scopes
18 ticket.SetScopes(request.GetScopes());
19 return SignIn(
20   ticket.Principal,
21   ticket.Properties,
22   ticket.AuthenticationScheme);
```

Et c'est tout ! L'API demandeuse reçoit un acces_token et peut commencer à l'utiliser pour appeler vos contrôleurs qui sont à protéger de la même manière que si la connexion provenait d'un utilisateur.

ClientId, ClientSecret – qu'est-ce que c'est ?

ClientId

Le ClientId n'est rien d'autre qu'**un identifiant représentant votre application.** Aucun format n'est imposé par la spécification mais il doit être unique : un seul par plateforme. Il est de bon goût de le rendre difficile à deviner pour rendre plus difficile son exploitation par un acteur malveillant mais cela peut aussi être quelque chose décrivant l'application enregistrée.

Quelques exemples :

- Windows Live : 00000000400EDD04
- Facebook : 803375973059704
- GoogleConsole : 30590984682
- Github : 6779ef20e75817b79602
- Inwink : identifiant-comprehensible-par-un-humain_bfa182e9-a09e-49c5-82fc-28905aae8bbf

ClientSecret

Le Client Secret est une sorte de mot de passe applicatif. Il est donc **TRES important de ne pas le stocker à un emplacement visible de tous** tel qu'une application Angular SPA ou dans le code d'une application native. Il est donc réservé à un usage où sa sécurité n'est pas compromise et donc principalement pour des "composants serveurs" (web app, azure functions, post-it sur votre bureau, etc.).

Aussi on distinguera les applications publiques (sans client_secret) et des applications privées (avec un client_secret).

Il s'agit d'un mot de passe et celui-ci doit donc respecter les normes de sécurité qui lui sont attachées. L'organisme Oauth recommande de générer une valeur de 256 caractères à l'aide d'un algorithme de cryptographie et de le convertir sous la forme d'une chaîne de caractères hexadécimaux. Pour ma part je préfère utiliser des mots de passes longs et faciles à retenir (https://www.theverge.com/2017/8/7/16107966/password-tips-bill-burr-regrets-advice-nits-cybersecurity).

Au fait pourquoi on utilise cela ?

Il est tout à fait possible d'autoriser une application sans ClientId de s'authentifier sur votre serveur. L'inconvénient de cette pratique est qu'il est impossible de savoir d'où provient une requête ultérieurement émise avec l'acces_token généré initialement.

En créant une "identité" pour chaque application utilisant votre serveur OIDC, cela permet notamment :

- De révoquer complètement l'accès d'une application aux ressources protégées si elle a été compromise.
- De tracer l'origine des demandes de connexion.
- Et surtout de donner des périmètres d'accès aux ressources différents par app (https://blogs.infinitesquare.com/posts/web/securiser-une-api-avec-openid-connect).

Dynamic Client Registration

Il est possible de proposer un enregistrement à la volée des applications souhaitant utiliser votre serveur d'authentification. Cela est défini dans la spécification même d'OpenId Connect et permet d'exposer une URL sur votre serveur OIDC pour enregistrer une application. Attention cependant car dans ce scénario, vous n'avez pas forcément de contrôle sur qui va utiliser votre serveur pour authentifier des utilisateurs. A ma connaissance, OpenIddict ne propose rien par défaut mais cela reste assez simple à mettre en place (utilisez le code ci-dessous par exemple :)).

Une application Console pour enregistrer une application avec OpenIdConnect

La plupart du temps et faute de temps les différentes applications sont enregistrées à la main dans la base de données avec un script SQL par quelqu'un de l'équipe technique.

Je ne trouve pas cela idéal car :

- cela nécessite de se connecter à une base de données avec un outil permettant d'exécuter du SQL ;
- cela ne permet pas la mise en place de règles automatiques de validation de critères de base que vous instaurez (pertinence du mot de passe, etc.) ;
- cela ne permet pas d'être dans une Release pour du déploiement automatisé ;
- cela ne permet pas de déléguer cette tâche de gestion à une équipe moins *proche* du code mis en place et plus à l'aise avec un exécutable *classique*.

Bref, j'aime bien avoir une application console associée à une documentation que je peux facilement donner à une autre personne dédiée à l'environnement de production.

La documentation d'OpenIddict donne un exemple de code à mettre facilement dans votre API pour permettre l'inscription d'une application sur votre serveur OIDC. Cela permet facilement la mise en place d'une console d'administration dans un site web et je vais réutiliser le même code dans une application console.

Pour cela je vais *tricher* (certains diront que c'est vraiment ... pas très propre) un peu en **hébergeant l'API dans une application console le temps de bénéficier du pipeline d'injection de dépendance pour récupérer les composants nécessaires et faire le travail de création d'une application.**

La technique pour mettre en place cela est très simple :

- On crée une app console .NET CORE ;
- On référence le projet correspondant au site où OIDC est en place ;
- On héberge ce site web à l'aide de WebHost ;
- On récupère l'OpenIddictManager à l'aide du service d'injection de dépendance ;
- On enregistre l'application.

Pour récupérer la configuration depuis les arguments passés en ligne de commande, j'utilise les outils ASP.Net core. La classe MaConfig étant uniquement un POCO définissant les propriétés à lire.

```
internal static async Task Main(string[] args)
{
    var config = new ConfigurationBuilder()
      .AddCommandLine(args)
      .Build();
    var clientIdGeneratorConfig = config.Get<MaConfig>();
      ...
}

public class MaConfig
{
   public string ClientId { get; set; }
   public string ClientSecret { get; set; }
   public string DisplayName { get; set; }
   public string RedirectUrl { get; set; }
   public string SqlConnectionString { get; set; }
}
```

L'hosting de l'API est ensuite réalisé après avoir reproduit une configuration "conforme" :

```
var inMemoryConfig = new Dictionary<string, string>
{
   {"ConnectionStrings:DefaultConnection", connectionString}
};
var configuration = new ConfigurationBuilder()
    .AddInMemoryCollection(inMemoryConfig)
    .Build();

var webHost = WebHost.CreateDefaultBuilder()
   .UseStartup<StartupDeMonApi>()
   .UseConfiguration(configuration)
   .Build();

// démarrage de l'API
await webHost.StartAsync();
```

Une fois cela fait, on peut récupérer le service d'injection de dépendances depuis notre webHost :

```
using (var serviceScope = webHost.Services.CreateScope())
{
}
```

On peut alors utiliser le code proposé en exemple sur le GitHub OpenIddict tel quel. Il sera bien sûr nécessaire de choisir les scopes selon les nécessités du projet.

```
var manager = serviceScope.ServiceProvider
  .GetRequiredService<
  OpenIddictApplicationManager<OpenIddictApplication<long>>>();

var descriptor = new OpenIddictApplicationDescriptor
{
    ClientId = myConfig.ClientId,
    ClientSecret = myConfig.ClientSecret,
    DisplayName = myConfig.DisplayName,
    Permissions =
    {
      OpenIddictConstants.Permissions.Endpoints.Token,
      OpenIddictConstants.Permissions.GrantTypes.Password,
      OpenIddictConstants.Permissions.Scopes.Email,
      OpenIddictConstants.Permissions.Scopes.Profile,
    }
};

await manager.CreateAsync(descriptor);
```

Voici un exemple d'appel avec passage de paramètres :

```
--ClientId=Kikou
--ClientSecret=Lol
--DisplayName="Une application"
--SqlConnectionString="Server=tcp:deded;Initial Catalog=dev;Persist Security Info=False;User ID=inwink-sql-admin;Password=dededed;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=True;Connection Timeout=30;"
```

Comment bien choisir son flow de connexion ?

Dans cette partie, je vais détailler les différents flows possibles de connexion dans le cadre d'OpenId Connect. Nous utiliserons la terminologie suivante :

- L'application cliente (AC) souhaitant accéder à une ressource sécurisée ;
- Le serveur d'Authorization (SA) permettant d'authentifier l'utilisateur ;
- Le serveur possédant la ressource (SR).

Je vais présenter plusieurs concepts dans cet article :

- Le processus de connexion "Authorization code flow" ;
- Le processus de connexion "Implicit flow" ;
- Le processus de connexion "Ressource Owned Password Credential" ;
- Le processus de connexion "Client Credential" ;
- PKCE (à prononcer pixy).

Authorization Code Flow

Processus de connexion

Dans ce mode, l'utilisateur doit se connecter sur une mire de connexion du serveur d'Autorisation qui retourne un code à l'application cliente. Cette dernière pourra alors échanger ce code auprès du serveur d'autorisation afin d'obtenir un jeton d'accès.

Le processus suivant est alors mis en place :

- L'AC affiche à l'utilisateur une page de connexion hébergée sur le SA. Il fournit en paramètre de l'URL les informations obligatoires : client_id, scope demandé, url de callback et un state (aussi appelé nonce). On utilise ici un endpoint spécifique appelé "Authorization endpoint" dans la littérature.
- Le SA affiche alors une page de connexion à l'utilisateur où il lui demande de se connecter et d'autoriser l'accès aux ressources lui appartenant pour le périmètre donné à l'AC.
- Une fois que l'utilisateur accepte, le SA appelle l'url de callback (indiquée lors de l'appel initial) avec comme paramètre (GET) un code d'Autorisation.
- L'AC interprète ce code et appelle le SA en demandant d'échan-

1

ger ce code contre un access_token. S'il s'agit d'une application privée, elle doit absolument s'identifier (utiliser le client_id / client_secret) pour faire cet échange. On utilise ici un endpoint spécifique appelé "Token endpoint" dans la littérature.

- Le SA valide ce code et retourne un jeton d'accès et aussi le nonce initialement passé en paramètre.
- L'AC valide alors que le nonce reçu correspond bien à celui transmis initialement, sinon il faut considérer le jeton comme invalide. 1

Pour quels types d'application ?

Ce processus est souvent recommandé pour les "applications privées" n'étant pas en mesure de stocker de manière sécurisée un client_secret. Cependant, rien dans la spécification n'empêche de l'utiliser pour une application publique (sans utilisation de client_secret donc). Aussi, ce workflow est dédié aux applications ne pouvant pas faire confiance à la couche applicative gérant la partie "callback". Imaginons par exemple que l'url de callback utilisée soit une URL de type "protocole" déclenchant l'ouverture d'une application, rien ne garantit qu'une application espionne ne peut pas intercepter un éventuel jeton d'accès en implémentant elle-même le protocole utilisé.

Voici quelques exemples d'applications pouvant utiliser ce mode de connexion :

- Application serveur : il s'agit du mode préféré de connexion car il est possible de stocker de manière sécurisé le client_secret.
- Application SPA : pas de problème non plus pour ce mode qui est une étape de plus par rapport au mode "implicit" décrit ci-après. Attention, vous le verrez sur l'internet mondial, il n'y a pas de consensus général sur la possibilité d'échanger un code contre un token par une application "publique" et beaucoup de fournisseurs d'identités l'empêchent. L'utilisation de PKCE (plus d'infos plus tard dans cet article) est alors recommandée.
- Application native : même chose que pour une application SPA, à la différence près que le mode 'implicit" n'est pas une option pour application native (du fait de l'absence du navigateur web en guise d'intermédiaire).

Comment gérer la péremption du jeton d'accès ?

Il est possible d'obtenir un jeton de rafraîchissement dans ce mode mais il faudra s'assurer de le conserver de manière sécurisée car il donne un accès à votre API.

Pour tous les cas où cela ne peut pas être garanti, il faudra faire repasser l'utilisateur par le processus complet de connexion. Par contre, si l'utilisateur est encore connecté (par exemple s'il a coché une case "se souvenir de moi" sur la page de connexion) et que le périmètre demandé n'a pas changé, alors cela peut être fait de **manière assez transparente pour lui** en utilisant une iframe / webview cachée qui appellera déclenchera de manière naturelle l'appel de l'url de callback.

À quoi ressemble un appel ?

Dans ce mode, l'appel correspond notamment à l'URL d'affichage de la mire de connexion. On remarquera notamment que l'on demande une réponse de type token :

```
1 GET /authorize?
2   response_type=code
3   &client_id=monClientId
4   &redirect_uri=https%3A%2F%2Furl.callback%2Fcb
5   &scope=openid%20profile
6   &state=kikouLolMdr
```

Je vous invite à lire la spécification pour y trouver une liste complète des paramètres possibles : https://openid.net/specs/openid-connect-core-1_0.html#CodeFlowAuth

Implicit Flow

Processus de connexion

Ce mode est très similaire au précédent, à la différence près que vous pouvez faire confiance à la couche applicative effectuant la partie callback. Cela est par exemple le cas d'une application Web : on peut faire confiance au Browser pour intercepter la demande d'affichage d'une URL utilisant le protocole HTTP (mais si, je vous dis, on peut !).

Attention cependant, le groupe de travail IETF sur Oauth recommande depuis peu de ne pas utiliser ce mode de connexion (https://tools.ietf.org/html/draft-ietf-oauth-security-topics-09#section-2.1.2) :

```
1 In order to avoid these issues, Clients SHOULD NOT use the implicit
2 grant and any other response type causing the authorization server to
3 issue an access token in the authorization response.
```

Dans tous les cas, le processus mis en place est le suivant :

- L'AC affiche à l'utilisateur une page de connexion hébergée sur le SA. Il fournit en paramètre de l'URL les informations obliga-

A DÉCOUVRIR D'URGENCE

LE CADEAU GEEK IDÉAL

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

9,99 €
(+ 3 € de frais postaux)

116 pages - Format magazine A4

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

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

PROG 235

☐ Mme ☐ M. Entreprise : Fonction :

Prénom : Nom :

Adresse :

Code postal : Ville :

E-mail : @

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

commandez sur www.programmez.com

toires : client_id, scope demandés et url de callback.

- Le SA affiche la page de connexion à l'utilisateur où il lui demande de se connecter et d'autoriser l'accès aux ressources lui appartenant pour le périmètre donné à l'AC.
- Une fois que l'utilisateur accepte, le SA appelle l'url de callback (indiquée lors de l'appel initial) avec comme paramètre (GET) le jeton d'accès.
- L'AC interprète l'URL de callback et y trouve avec joie le jeton d'accès. Dans ce mode, le jeton est fourni sous la forme d'un fragment de l'url (derrière un # donc), plutôt que dans un paramètre GET, car dans cette configuration aucun appel réseau n'est fait avec le jeton d'accès visible dans l'URL (les fragments ne sont pas envoyés par les navigateurs web aux serveurs) bien que restant visible/exploitable par l'AC. 2

Pour quels types d'application ?

Principalement pour toutes les applications dont le mécanise de "callback" est garanti comme sûr : les applications SPA principalement.

Comment gérer la péremption du jeton d'accès ?

Exactement de la même manière que pour le code flow.

À quoi ressemble un appel ?

Très similaire à celle vue précédemment mais on demande une réponse de type "token" :

```
1 GET /authorize?
2   response_type=token
3   &client_id=monClientId
4   &redirect_uri=https%3A%2F%2Furl.callback%2Fcb
5   &scope=openid%20profile
6   &state=kikouLolMdr
7 Host: infinitesquare.com
```

2

Resource Owner Password Credentials / Password flow

Ce mode permet à l'AC d'héberger elle-même le formulaire de connexion pour le présenter à l'utilisateur. Cela est très intéressant et permet de proposer à l'utilisateur une expérience plus agréable :

- Le formulaire est complètement intégré (d'un point de vue design) à l'application,
- Il n'y a pas de navigation dans un browser "qui le sort du contexte où il est",
- Il n'a qu'à cliquer sur un bouton pour se connecter.

Processus de connexion

Ce processus est alors mis en place :

- L'AC affiche à l'utilisateur une page de connexion.
- L'AC demande un jeton d'accès en utilisant le login/mot de passe donné par l'utilisateur. **Ici c'est l'endpoint donnant des tokens qui est utilisé en POST : il n'y a donc pas de mot de passe qui transite en clair dans une URL.** 3

Pour quels types d'application ?

En réalité il s'agit surtout d'un processus de connexion mis en place pour migrer des applications utilisant l'ancienne connexion HTTP basic vers OAuth et n'est pas forcément le mode de connexion le plus recommandé.

En effet **il possède plusieurs défauts à prendre en compte** :

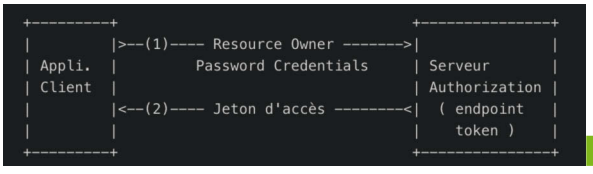
- Le login / mot de passe de l'utilisateur va être dans les mains de l'AC et transiter "en clair" dans les paramètres envoyés au SA.
- Il est facile dans ce contexte de créer une application "fake" qui demande à l'utilisateur son login / mot de passe car il a l'habitude de le faire dans l'AC.
- On fait confiance à l'application cliente pour présenter de manière correcte le périmètre demandé à l'utilisateur.
- On ne peut pas mettre de SSO car la connexion se passe à chaque fois sur l'application client (et non pas sur un seul SA).
- Son acronyme ROPC est quand même très difficile à prononcer.
- etc.

Néanmoins, il reste intéressant à mettre en place dans certains contextes, notamment s'il s'agit d'une application "confidentielle" (pas visible sur l'internet mondial) avec une seule application cliente (pas besoin de SSO), et où le besoin d'expérience utilisateur de haute voltige est très fort.

À quoi ressemble un appel ?

L'appel ici est unique pour demander le jeton d'accès :

```
1 GET /authorize?
2   grant_type=password
3   &client_id=monClientId
```



3

```
4  &scope=openid%20profile
5  &username=jonathan
6  &password=motDePasse
```

Client Credentials

Ce processus est dédié à des applications sans notion d'utilisateur. J'y suis déjà rentré en détail en termes de mise en place avec OpenIddict précédemment dans cet article.

Processus de connexion

Ce processus qui n'a pas besoin de demander à l'utilisateur de se connecter est donc très simple :) :

- L'AC demande un jeton d'accès en utilisant le client_id/client_secret. **Ici encore c'est l'endpoint donnant des tokens qui est utilisé en POST : il n'y a donc pas de mot de passe qui transite en clair dans une URL.** 4

Pour quels types d'application ?

Pour toutes les applications en mesure de stocker de manière sécurisée un client_secret. Cela est donc réservé de manière générale aux applications "serveur".

À quoi ressemble un appel ?

```
1  GET /authorize?
2     grant_type=client_credentials
3     &client_id=monClientId
4     &scope=openid%20profile
5     &client_id=jonathan
6     &client_secret=motDePasse
```

PKCE

PKCE signifie Proof Key for Code Exchange et se prononce "pixy". Ce n'est pas un processus de connexion à proprement parler mais **c'est une "surcouche" à l'Authorization code flow qui permet d'éviter les problèmes liés à l'interception du code. Il est vivement recommandé d'utiliser PKCE avec ce mode de connexion (https://tools.ietf.org/html/draft-ietf-oauth-security-topics-09#section-2.1.1).** En effet, si une application malveillante réussit à intercepter le code, elle peut l'échanger facilement contre un jeton d'accès et utiliser l'API au nom de l'utilisateur. Cela est d'ailleurs très bien schématisé dans la documentation officielle : 5

PKCE propose ainsi deux choses :

- Dans le premier appel de demande de code : indiquer au SA le hash d'un code de vérification et la méthode de hachage utilisé. Le code de vérification doit avoir un minimum de 43 caractères et un maximum de 128 caractères *non réservés*. Il existe deux méthodes de cryptage : plain ou sha256 qui doit être préféré si cela est possible.

4

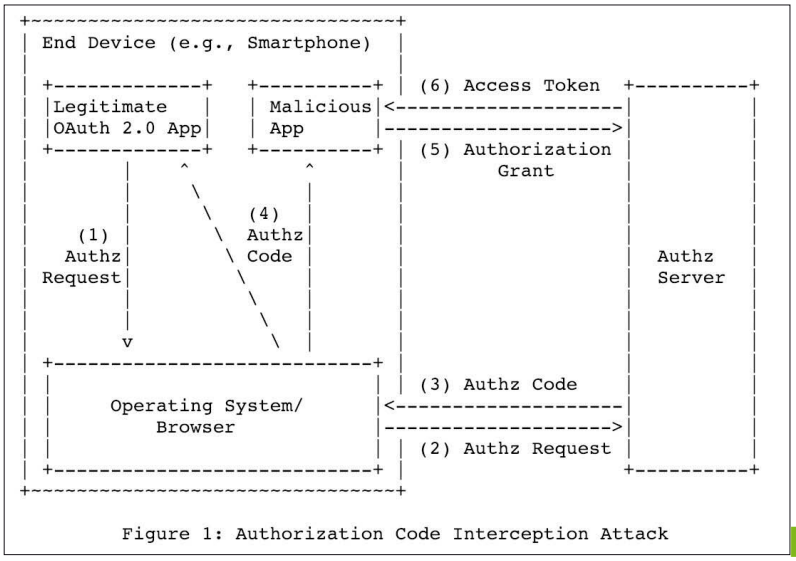


Figure 1: Authorization Code Interception Attack

5

- Dans l'appel permettant d'échanger le code contre un jeton d'accès, indiquer le code de vérification en clair. Le SA vérifie alors que le jeton initial correspond à celui donné au second appel et ne donne un jeton que si cela est le cas.

Ainsi, même en interceptant le code, il devient difficile de l'échanger contre un jeton d'accès.

Un appel pour une demande de code ressemble alors à celui vu pour Authorization Code Flow avec en plus les paramètres code_challenge et code_challenge_method :

```
1  GET /authorize?
2     response_type=code
3     &client_id=monClientId
4     &redirect_uri=https%3A%2F%2Furl.callback%2Fcb
5     &scope=openid%20profile
6     &state=kikouLolMdr
7     &code_challenge=vBcZBGqBEcQdekdkedkJNDLEJjnde678HD
8     &code_challenge_method=S256
```

Plus d'infos : https://tools.ietf.org/html/rfc7636

Certificat de signature de token - déploiement et utilisation sur une webapp Azure

Les jetons d'accès générés selon la norme OpenId Connect peuvent être signés numériquement afin d'avoir une vérification faite par le client demandeur de token. Cela est notamment le cas lors de l'utilisation de jeton au format JWT. Voyons désormais comment générer un certificat, demander son utilisation par OpenIddict et comment l'utiliser au sein d'une web app sur Azure.

Générer un certificat de signature

La première étape consiste à générer un certificat pour permettre la signature de vos jetons JWT. Je vous propose de le faire avec openSSL en utilisant cette commande :

```
1  openssl req
2     -x509
```

```
3    -sha256
4    -nodes
5    -days 365
6    -newkey rsa:2048
7    -keyout votreClefPrivee.key
8    -out votreCertificat.crt
```

Ces paramètres spécifient qu'il s'agit d'un certificat de type X509 valable pour une durée de 365 jours qui va produire une clef privée RSA 2048 bits. L'argument nodes demande de ne pas encrypter la clef privée au format PKCS. Vous aurez ainsi en sortie un fichier "votreCertificat.crt" et la clef privée "votreClefPrive.key".

Configurer OpenIddict

L'étape suivante consiste à demander à OpenIddict d'utiliser votre certificat **en partant du principe qu'il est disponible sur la machine d'exécution de votre serveur. Je vous propose donc de passer par la méthode AddSigningCertificate qui prend en paramètre l'empreinte correspondant à votre certificat nouvellement créé.** L'empreinte du certificat est ici lue dans la configuration ASP.Net Core à la clef OpenIdConnect:Thumbprint :

```
1  services.AddOpenIddict()
2  .AddCore(
3     options =>
4     {
5      // configuration du cœur
6     })
7  .AddServer(
8     options =>
9     {
10   // autres configurations du serveur ici
11   var thumbprint = Configuration["OpenIdConnect:Thumbprint"];
12   options.AddSigningCertificate(thumbprint);
13   });
```

Pour obtenir l'empreinte de votre certificat, openssl vient à votre secours avec la commande ci-dessous. Il faudra prendre soin d'omettre les caractères « :" lorsque vous le renseignerez dans le fichier de configuration.

```
1  openssl
2     x509
3     -noout
4     -fingerprint
5     -sha1
6     -inform pem
7     -in votreCertificat.crt
```

Configurer votre WebApp - téléversement du certificat

Il reste alors à aller configurer votre webapp sur Azure. La première étape consiste à téléverser le certificat au format PFX sur votre AppService. **Il sera nécessaire de choisir un plan Basic B1 au minimum pour avoir accès à la page en question.**

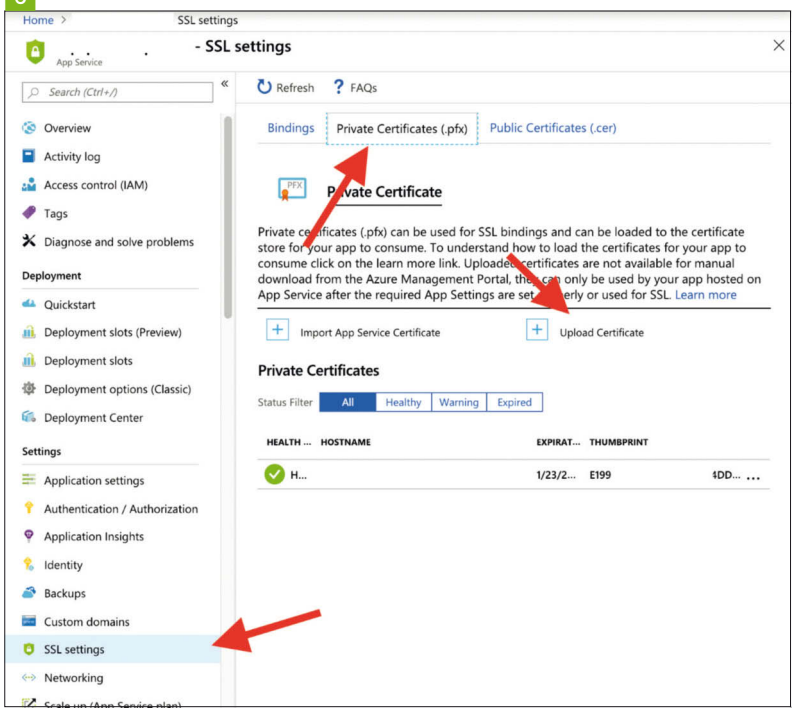
Un PFX est une archive protégée par mot de passe qui contient à la fois votre clef privée et le certificat. OpenSSL va encore une fois vous aider à générer cela avec cette commande :

```
1  openssl
2     pkcs12
3     -export
4     -out votreArchive.pfx
5     -inkey cotreClefPrivee.key
6     -in votreCertificat.crt
```

Pour téléverser le certificat, on suit cette procédure :

- Se connecter au portail Azure,
- Se rendre sur votre WebApp,
- Sélectionner "SSL Settings" dans le menu à gauche,
- Sélectionner l'onglet "Private Certificates (.pfx)" dans la page qui s'ouvre,
- Ajouter le fichier PFX précédemment créé via le bouton d'upload.

6



Configurer votre WebApp - activation du certificat

La dernière étape consiste à activer votre certificat au niveau de la WebApp. Cela se fait simplement en rajoutant un appsettings ayant comme clef "WEBSITE_LOAD_CERTIFICATES" et comme valeur le thumbprint de votre certificat.

Conclusion

J'espère qu'après la lecture de cet article vous avez maintenant une bonne vision de ce qu'est OpenId Connect et que sa mise en place ne vous fait maintenant plus peur ! Je vous invite fortement à aller creuser la documentation officielle, vous plonger dans le code source d'OpenIddict de Kevin Chalet sur GitHub…et à nous rejoindre sur le blog Infinite Square. •

Philippe BOULANGER
Manager des expertises C/C++ et Python
www.invivoo.com

INVIVOO BEYOND TECH

A la recherche de la mémoire perdue : la mémoire, un gruyère ?

niveau 200

Dans une application de calculs de risque chez un client, de nombreux crashs aléatoires survenaient. Le symptôme visible était la mémoire d'un processus qui augmentait de manière régulière jusqu'à 3Go, et là, le processus crashait. Cette application était multiprocessus répartis sur un cluster de 2 à 7 machines : chaque machine exécutant 1,5 fois plus de processus que de nombre de cœurs (une machine de 16 cœurs logiques exécutait donc 24 processus) et chaque processus travaillant avec de 4 à 30 threads. Du fait de l'architecture, plusieurs traders accédaient simultanément aux machines pour exécuter des simulations ce qui faisait que nous n'arrivions pas à reproduire le problème sur les environnements de développement ; d'autant plus que les traders, jaloux de leurs idées, ne nous disait pas vraiment quelles données ils avaient fournies au cluster : on avait au mieux un à peu près. Nous n'avions alors que les fichiers de logs et notre expérience pour comprendre les causes du problème… L'application était écrite en C++ avec un framework (***ICE***) simplifiant la programmation distribuée et la communication entre les processus. Pour des raisons de « performances », les premiers développeurs avaient intégré boost et ***tbb*** (Thread Building Block d'Intel) pour, notamment, avoir accès à des conteneurs efficaces. Il y avait des changements de structures de données importants entre les conteneurs ***STL***/***boost*** et ceux du framework.

Comprendre

C et C++ sont des langages de bas niveau, le résultat de la compilation promet de bonnes performances, mais, de ce fait, les tâches d'allocations et de libérations de la mémoire sont à la charge du développeur… Le développeur est humain et l'erreur est humaine : oublier de libérer de la mémoire est donc malheureusement fréquent. Depuis l'apparition de C++ 11, des outils comme les pointeurs intelligents (***std::unique_ptr***, ***std::shared_ptr*** et ***std::weak_ptr***) permettent de simplifier la gestion mémoire au prix d'un léger overhead. Cependant, dans les codes « legacy » qui existent depuis bien avant le C++ 11 ceux-ci ne sont pas utilisés…

La seconde raison qui peut engendrer des consommations excessives de mémoire est un mauvais choix de types pour les données. Supposons que vous ayez besoin de stocker une valeur entière de 0 à 9, si vous utilisez un '***int***' vous allez utiliser 4 octets alors qu'un '***unsigned char***' (1 octet) suffirait… Si la volumétrie de cette donnée se compte en millions, on perd rapidement des mégaoctets d'espace… Bon nombre de codes ont été conçus pour une volumétrie faible sans penser qu'un jour celle-ci allait augmenter de manière importante et cela est d'autant plus vrai dans l'environnement bancaire…

La troisième raison est la méconnaissance des structures de données provenant de la bibliothèque standard (***STL***) ou des bibliothèques tierces (***boost*** ou ***tbb*** pour le cas cité dans le contexte) … Pour travailler, certaines de ces structures de données allouent plus de mémoire que ce à quoi l'on s'attend…

La quatrième raison est l'optimiseur du compilateur C++ qui va prendre des décisions dont le développeur ne va pas forcément avoir conscience… Et c'est sur ce point que nous allons travailler dans cet article.

La dernière raison, qui est certes peu fréquente en dehors d'un système très multi-threadé, est la pénurie de mémoire disponible, car trop de threads allouent et libèrent de la mémoire en même temps. Car ce qu'il faut savoir c'est que sous Windows et Linux, lorsque vous faîtes '***free***', '***delete***' ou '***delete []***' vous stocker le bloc de mémoire dans une file et un thread ramasse-miette va le rendre au système. Si jamais trop de threads consommateurs travaillent simultanément, le thread ramasse-miette n'a plus un quantum de temps suffisant pour satisfaire tout le monde.

Les types de données simples et leur taille

La première chose à faire est de comprendre quels sont les types de données de base et la taille en octet de chacun d'entre eux. En C/C++ il y a une façon simple de récupérer la taille d'un type de données : ***sizeof***. C'est une fonction qui prend en paramètre le nom du type et qui retourne le nombre d'octets pris par celui-ci. Voici un exemple de programme pour récupérer les tailles :

```
int main()
{
    cout << "Size of bool    : " << sizeof( bool )    << "b" << endl;
    cout << "Size of char    : " << sizeof( char )    << "b" << endl;
    cout << "Size of short   : " << sizeof( short )   << "b" << endl;
    cout << "Size of int     : " << sizeof( int )     << "b" << endl;
    cout << "Size of int16_t : " << sizeof( int16_t ) << "b" << endl;
    cout << "Size of int32_t : " << sizeof( int32_t ) << "b" << endl;
    cout << "Size of int64_t : " << sizeof( int64_t ) << "b" << endl;
    cout << "Size of float   : " << sizeof( float )   << "b" << endl;
    cout << "Size of double  : " << sizeof( double )  << "b" << endl;
    return 0;
}
```

Voici le résultat obtenu :

Type	Information	Taille en octets
bool	true, false	1
char	-128 à 127	1
unsigned char	0 à 255	1
std::int8_t	-128 à 127	1
std::uint8_t	0 à 255	1
short	-32768 à 32767	2
unsigned short	0 à 65535	2
std::int16_t	-32768 à 32767	2
std::uint16_t	0 à 65535	2
int	-2147483648 à 2147483647	4
unsigned int	0 à 4294967295	4
std::int32_t	-2147483648 à 2147483647	4
std::uint32_t	0 à 4294967295	4
std::int64_t	-9223372036854775808 à 9223372036854775807	8
std::uint64_t	0 à 18446744073709551615	8
float	7 digits	4
double	15 digits	8

Pour des raisons pratiques, aujourd'hui et depuis C++ 11, il est préférable d'utiliser les entiers définis dans la STL : std::int8_t, …, std::int64_t.

Les types structurés et leurs tailles

Commençons par un petit exemple

Les types structurés (***struct*** et ***class***) ont eux aussi des tailles que l'on peut obtenir grâce à ***sizeof***… Jusque-là le C/C++ reste cohérent.

Partons d'un exemple simple, je souhaite simuler un intervalle de nombres flottants potentiellement ouvert ([-10, 20] ou]-oo, 50] ou [15, +oo[pour exemple). Afin de savoir si les bornes sont définies ou infinies, nous allons uti-

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

[Programmez!] Le magazine des développeurs

Nos classiques

1 an 11 numéros	49€*
2 ans 22 numéros	79€*
Etudiant 1 an - 11 numéros	39€*

** Tarifs France métropolitaine*

Abonnement numérique

PDF 1 an - 11 numéros	35€
Option : accès aux archives	10€

Souscription uniquement sur
www.programmez.com

Offres 2019

1 an — 49€
11 numéros
+ 1 vidéo ENI au choix :

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

2 ans — 79€
22 numéros
+ 1 vidéo ENI au choix :

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

** Offre limitée à la France métropolitaine*

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

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

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

☐ **Abonnement 1 an :** 49 €
11 numéros + 1 vidéo ENI au choix :
☐ **Abonnement 2 ans :** 79 €
22 numéros + 1 vidéo ENI au choix :

☐ Vidéo : Symfony 3
☐ Vidéo : Raspberry Pi

PROG 235
Valable jusqu'au 3 janvier 2020

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

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

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

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

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

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

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

NOUVEAU ! OFFRES 2020

1 an
11 numéros

+ Histoire de la micro-informatique 1973 à 2007

+ clé USB Programmez!

69€*

1 an
11 numéros

+ 1 an de PHARAON *Magazine* (Histoire / Archéologie) 4 numéros

69€*

2 ans
22 numéros

+ 2 ans de PHARAON *Magazine* (Histoire / Archéologie) 8 numéros

99€*

OFFRES SPÉCIALES D'ABONNEMENT DISPONIBLES SUR WWW.PROGRAMMEZ.COM

*(*Offre limitée à la France métropolitaine. Pour l'étranger : nous consulter)*

liser des booléens ('*bool*'). Pour les valeurs des bornes, nous utiliserons des flottants en double précision ('*double*'). Ce qui nous donnera la déclaration de type suivante :

```
struct Intervalle
{
    bool   _isLowerBoundInfinite;
    double _LowerBound;
    bool   _isUpperBoundInfinite;
    double _UpperBound;
};
```

Ayant utilisé 2 '*bool*' et 2 '*double*', je m'attends à ce que la taille du type Intervalle soit de 2x1+2x8=18 octets… En effectuant un sizeof sur le type, j'obtiens… (suspense… si un petit peu quand même) … J'obtiens 32 octets !!!!! Et là je ne comprends plus rien… Je ne m'y attendais pas du tout… Où sont passés les 14 octets perdus et, surtout, pourquoi ?

Faisons des tests pour comprendre

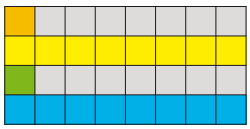
Le C/C++ étant un langage de bas niveau (ouf !), nous avons des outils pour comprendre comment est composé le type : nous avons un opérateur '**&**' pour récupérer l'adresse d'une variable.

```
#define ADDR(p) reinterpret_cast<char*>(&(p))
void work_Infinite()
{
    cout << endl;
    cout << "Intervalle est compose de 2 booleens et de deux flottants 64bits" << endl;
    cout << "   2*sizeof(bool)+2*sizeof(double) : " << 2*sizeof(bool)+2*sizeof(double) << endl;
    cout << "   taille du type Intervalle constate : " << sizeof(Intervalle) << endl;

    Intervalle t;
    cout << "   adresse(_isLowerBoundInfinite) : " << ADDR(t._isLowerBoundInfinite) - ADDR(t) << endl;
    cout << "   adresse(_LowerBound) : " << ADDR(t._LowerBound) - ADDR(t) << endl;
    cout << "   adresse(_isUpperBoundInfinite) : " << ADDR(t._isUpperBoundInfinite) - ADDR(t) << endl;
    cout << "   adresse(_UpperBound) : " << ADDR(t._UpperBound) - ADDR(t) << endl;
}
```

En effectuant 'ADDR(t._isLowerBoundInfinite) - ADDR(t)', on obtient la position relative en octet au sein de la structure. En C++, on peut aussi utiliser '*offsetof*(t, x)' qui fournira la même fonctionnalité. En exécutant le code, on va obtenir ceci :

```
Intervalle est compose de 2 booleens et de deux flottants 64bits
   2*sizeof(bool)+2*sizeof(double) : 18
   taille du type Intervalle constate : 32
   adresse(_isLowerBoundInfinite) : 0
   adresse(_LowerBound) : 8
   adresse(_isUpperBoundInfinite) : 16
   adresse(_UpperBound) : 24
```

Pour mieux comprendre, on va utiliser un tableau :



_isLowerBoundInfinite
_LwerBound
_isUpperBoundInfinite
_UpperBound
mémoire perdue

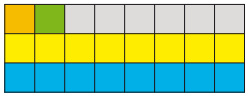
En gris nous avons la mémoire perdue. Il semblerait que le compilateur ait essayé d'aligner les adresses des membres de la structure… Essayons de changer l'organisation de la structure pour voir si cela change quelque chose :

```
struct Intervalle1
{
    bool   _isLowerBoundInfinite;
    bool   _isUpperBoundInfinite;
    double _LowerBound;
    double _UpperBound;
};

void work_Infinite1()
{
    cout << endl;
    cout << "Intervalle1 est compose de 2 booleens et de deux flottants 64bits" << endl;
    cout << "   2*sizeof(bool)+2*sizeof(double) : " << 2*sizeof(bool)+2*sizeof(double) << endl;
    cout << "   taille du type Intervalle1 constate : " << sizeof(Intervalle1) << endl;

    Intervalle1 t;
    cout << "   adresse(_isLowerBoundInfinite) : " << ADDR(t._isLowerBoundInfinite) - ADDR(t) << endl;
    cout << "   adresse(_isUpperBoundInfinite) : " << ADDR(t._isUpperBoundInfinite) - ADDR(t) << endl;
    cout << "   adresse(_LowerBound) : " << ADDR(t._LowerBound) - ADDR(t) << endl;
    cout << "   adresse(_UpperBound) : " << ADDR(t._UpperBound) - ADDR(t) << endl;
}
```

Et en l'exécutant, on obtient :

```
Intervalle1 est compose de 2 booleens et de deux flottants 64bits
   2*sizeof(bool)+2*sizeof(double) : 18
   taille du type Intervalle1 constate : 24
   adresse(_isLowerBoundInfinite) : 0
   adresse(_isUpperBoundInfinite) : 1
   adresse(_LowerBound) : 8
   adresse(_UpperBound) : 16
```

Si on reprend notre organisation sous forme d'un tableau :



_isLowerBoundInfinite
_LowerBound
_isUpperBoundInfinite
_UpperBound
mémoire perdue

On constate que la taille de la structure a diminué de 8 octets et que l'on a plus que 6 octets de perdus… La solution consiste peut-être à mettre les booléens à la fin de la structure… Essayons cette dernière solution :

```
struct Intervalle2
{
    double _LowerBound;
    double _UpperBound;
    bool   _isLowerBoundInfinite;
    bool   _isUpperBoundInfinite;
};

void work_Infinite2()
{
    cout << endl;
    cout << "Intervalle2 est compose de 2 booleens et de deux flottants 64bits" << endl;
    cout << "   2*sizeof(bool)+2*sizeof(double) : " << 2*sizeof(bool)+2*sizeof(double) << endl;
    cout << "   taille du type Intervalle2 constate : " << sizeof(Intervalle2) << endl;

    Intervalle2 t;
    cout << "   adresse(_isLowerBoundInfinite) : " << ADDR(t._isLowerBoundInfinite) - ADDR(t) << endl;
    cout << "   adresse(_isUpperBoundInfinite) : " << ADDR(t._isUpperBoundInfinite) - ADDR(t) << endl;
    cout << "   adresse(_LowerBound) : " << ADDR(t._LowerBound) - ADDR(t) << endl;
    cout << "   adresse(_UpperBound) : " << ADDR(t._UpperBound) - ADDR(t) << endl;
}
```

Et voici le tableau mémoire que nous obtenons :

_isLowerBoundInfinite
_LowerBound
_isUpperBoundInfinite
_UpperBound
mémoire perdue

On a malheureusement de l'espace perdu à la fin de la structure.

Le padding

Cette façon de ranger la mémoire au sein des structures, on appelle ceci le **padding**… C'est une optimisation du compilateur qui part du principe que la vitesse d'exécution est votre principal objectif. Il va donc aligner les adresses sur 4, 8 ou 16 octets pour accélérer les accès à la mémoire, car un processeur accède plus rapidement à la mémoire si les adresses sont des multiples de 4, 8 ou 16 octets !!! Pour utiliser les commandes SSE, il faut (même) obligatoirement aligner les vecteurs sur 16 octets pour profiter des performances optimales… Tous les compilateurs ont un nombre d'octets (ou stratégie) d'alignement par défaut… Visual C++ alignait sur 8 octets, par exemple. Changer cette stratégie est possible soit localement soit globalement. Lorsque l'on travaille en environnement embarqué avec peu de mémoire disponible, il peut être utile de changer globalement les alignements pour garantir que l'on ne consommera que la mémoire utile. Pour se faire, il faudra changer une des options du compilateur (et là, il faudra chercher dans la documentation, car il n'y a pas de normes quant aux paramètres des compilateurs)…

Changement global

Pour gcc, l'option '-fpack-struct=1' permet de forcer le padding à 1 octet au lieu des 4 par défaut. On peut aussi activer l'option '-Wpadded' pour que le compilateur lève un warning sur les structures dans lequel il y a du padding (mais malheureusement le message n'est pas toujours clair)… Sous VC++, l'option du compilateur est '/Zp[1|2|4|8|16]' où le nombre correspond au nombre d'octets de l'alignement.

Changement local

Ou alors, on peut désactiver l'option localement ; par exemple, sous gcc et Visual C++, vous pouvez faire :

```
#pragma pack(push, 1)
struct Intervalle3
{
    bool   _isLowerBoundInfinite;
    double _LowerBound;
    bool   _isUpperBoundInfinite;
    double _UpperBound;
};
#pragma pack(pop)

void work_Infinite3()
{
    cout << endl;
    cout << "Intervalle3 est compose de 2 booleens et de deux flottants 64bits" << endl;
    cout << "   2*sizeof(bool)+2*sizeof(double) : " << 2*sizeof(bool)+2*sizeof(double) << endl;
    cout << "   taille du type Intervalle3 constate : " << sizeof(Intervalle3) << endl;

    Intervalle3 t;
    cout << "   adresse(_isLowerBoundInfinite) : " << ADDR(t._isLowerBoundInfinite) - ADDR(t) << endl;
    cout << "   adresse(_LowerBound) : " << ADDR(t._LowerBound) - ADDR(t) << endl;
    cout << "   adresse(_isUpperBoundInfinite) : " << ADDR(t._isUpperBoundInfinite) - ADDR(t) << endl;
    cout << "   adresse(_UpperBound) : " << ADDR(t._UpperBound) - ADDR(t) << endl;
}
```

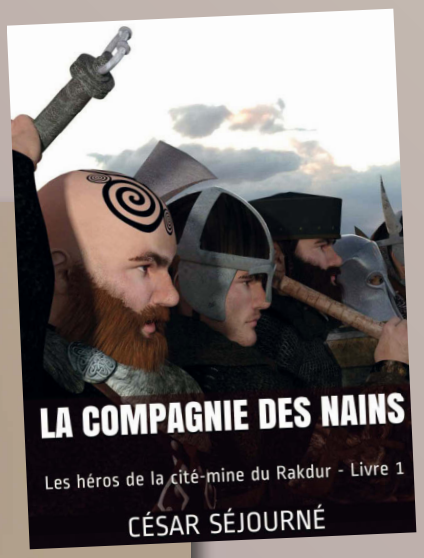
Ce qui en exécutant nous donnera :

```
Intervalle3 est compose de 2 booleens et de deux flottants 64bits
   2*sizeof(bool)+2*sizeof(double) : 18
   taille du type Intervalle3 constate : 18
   adresse(_isLowerBoundInfinite) : 0
   adresse(_LowerBound) : 1
   adresse(_isUpperBoundInfinite) : 9
   adresse(_UpperBound) : 10
```

Nous n'avons plus de mémoire perdue… YOUPI !!! Vous savez ce qu'il vous reste à faire.

La compagnie des nains

« Bramor est un jeune nain. Apprenti forgeron, il était destiné à un bel avenir au sein de la cité-mine du Rakdur. Affronter gobelins, géants et trolls, voilà ce dont il rêve chaque nuit. Sa première mission - assurer la reconnaissance des alentours du Rakdur - devait être des plus tranquilles... »



nouveau

Roman illustré d'heroic-fantasy de César Séjourné

L'ouvrage est disponible sur Amazon.fr en format broché (15,99 €) et kindle (2,99 €) :

https://tinyurl.com/y3lptfog

Samuel Blanchard
Directeur Innovation
SYNERGIZ
@samoteph

niveau 300

Reconnaissance de symbole avec IA

Vous rappelez-vous du jeu Okami de Capcom ressorti il y a quelques mois sur console Switch ? Dans ce jeu, le joueur doit dessiner des symboles avec la manette, ou au doigt, pour déclencher des actions. Les symboles sont parfois simples (point, trait, cercle, boucle, …), parfois plus complexes (bombe).

En traçant un cercle autour de l'arbre, il fleurit !

Le principe est accrocheur et l'on souhaite reprendre cette mécanique de détection de symboles dans un jeu de morpion. On dessinerait les pions du jeu (Croix ou Rond) pour les positionner dans une grille 3×3 et le symbole Bombe pour recommencer la partie. La bombe est choisie en conscience car c'est un symbole proche à la fois de la croix et du rond et donc intéressant en termes de difficulté de détection.

Dessiner un symbole

Dans Okami, pour dessiner, il est nécessaire d'appuyer sur une touche de la manette afin de rentrer dans le mode « Pinceau ». Lorsque l'on sort de ce mode, le dessin réalisé est analysé et, s'il est reconnu, l'action est déclenchée.

Dans le cadre de notre morpion, le mode « Pinceau » est actif en permanence. On doit pouvoir tracer la croix ou le rond sur la grille et l'analyser dès la fin du dessin. Cette fin est relativement simple à déterminer pour le symbole Rond car il ne comporte, normalement, qu'un seul trait mais pour le symbole Croix qui en comporte deux, il va falloir introduire le mécanisme suivant permettant de réaliser des dessins plus complexes à plusieurs traits :

- **Le pointeur (touch ou souris) appuie sur l'écran** : on crée une liste de points qui représente le trait (path).
- **Le pointeur bouge :** la position du pointeur (un point) est ajoutée dans la liste de points.
- **Le pointeur relâche son appui sur l'écran :** on ajoute la liste de points dans une liste chargée de collecter tous les traits (paths). On lance un minuteur (timer) d'une durée d'une seconde.
- **Le timer est déclenché au bout d'une seconde :** c'est la fin du dessin !
- **Le pointeur appuie de nouveau sur l'écran :** on arrête le timer puis on reprend depuis le début (puce 1.).

Lorsque la fin du dessin (puce 4.) est effective, la détection du symbole commence. Le symbole est alors composé d'une collection de traits, elle-même composée d'une liste de points.

Cahier des charges de la détection de symboles

Avant de se lancer dans le développement de la détection, il est préférable de garder en tête les objectifs à atteindre.

Les symboles détectables seront les suivants :

- Croix
- Rond
- Bombe (un rond avec un trait)
- La taille ou la position des symboles ne doivent pas influer sur la détection.
- Les symboles seront détectés quelle que soit la vitesse à laquelle ils sont réalisés.
- Ils pourront être composés d'un ou plusieurs traits.
- On pourra commencer et finir le dessin du symbole où l'on souhaite (un rond peut être commencé par le haut ou par le bas, à gauche ou à droite)

Algorithme et Intelligence Artificielle

L'algorithme utilisé dans Okami pour détecter les symboles a sans doute été très complexe à réaliser et les développeurs ont dû mettre en place un procédé spécifique pour chaque symbole.

Pour réaliser notre jeu de morpion, on souhaite, au contraire, un procédé unique pour chaque symbole, qui doit rester simple et rapide à mettre en place.

L'intelligence artificielle (IA) semble un bon candidat pour résoudre ce problème, de par son adaptabilité et sa simplicité d'intégration. On décide d'orienter notre choix vers une IA de type vision (image) : Custom Vision.

Custom Vision est un moteur d'apprentissage automatique qui fait partie de la gamme Cognitive Services de Microsoft. Il permet de créer son propre modèle très rapidement. Son interface graphique en fait l'un des moteurs les plus faciles à prendre en main du marché. Il propose un accès à ses prédictions par le biais d'une API ou d'un export du modèle sous forme de fichier.

L'API nécessite un accès à internet pour fonctionner, tandis que le fichier peut être utilisé en étant déconnecté. Le format fichier reste néanmoins bien moins performant que la version online.

Transformer le tracé en image

Les moteurs d'apprentissage automatique se nourrissent de jeux de données afin de créer un modèle. Plus le jeu est normalisé et cohérent, meilleurs seront les résultats.

Puisque l'on s'appuie sur une IA de type vision, on va devoir lui fournir des images correspondantes au symbole à analyser.

Chaque symbole tracé sera donc transformé en image de 500×500 pixels. Les dimensions de l'image sont un bon compromis entre la précision du tracé et le temps de traitement.
La création d'un modèle nécessite un nombre d'image important. Compter 500 images pour obtenir un modèle fonctionnel.

Application de saisie

Pour accélérer la création de ces images, il est souvent souhaitable de mettre en place une application de saisie spécifique. Ici, nous développerons une application de saisie de symbole qui reprendra le même mécanisme de dessin que celui utilisé pour notre jeu du morpion (voir Dessiner le Symbole) mais dont l'objectif sera d'enregistrer des images du symbole, voire plusieurs images pour un même symbole.

Implémentation du tracé de symbole

Dans un premier temps on va chercher à obtenir un rectangle contenant tous les points du tracé. C'est relativement simple à obtenir puisqu'il suffit de récupérer les valeurs X et Y les plus hautes et les plus basses des points (le code est en C#) :

```
// on recherche le rectangle contenant le symbole
double? left = null, right = null, top = null, bottom = null;

foreach (var path in paths)
{
  foreach (var point in path)
  {
    if (left == null)
    {
      left = point.X;
      right = point.X;
      top = point.Y;
      bottom = point.Y;
    }

    if (point.X < left.Value)
    {
      left = point.X;
    }
    else if (point.X > right.Value)
    {
      right = point.X;
    }

    if (point.Y < top.Value)
    {
      top = point.Y;
    }
    else if (point.Y > bottom.Value)
    {
      bottom = point.Y;
    }
  }
}

Rect boundSymbol = new Rect(left.Value, top.Value, right.Value - left.Value, bottom.
Value - top.Value);
```

Une fois, ces valeurs obtenues on est capable de déterminer la position et la taille du symbole tracé.
La taille du tracé n'est pratiquement jamais carrée. Mais puisque l'image destination l'est, il est nécessaire de recentrer le tracé (soit verticalement, soit horizontalement).
Pour recentrer le tracé dans un carré, on prend la plus grande taille de ce tracé :

```
double boundSize = Math.Max(boundSymbol.Width, boundSymbol.Height);
```

En retirant la taille du symbole (largeur et hauteur) à la plus grande taille (boundSize) puis en la divisant par 2 on obtient deux marges qui permettent de recentrer les points.
L'une des marges a toujours une valeur égale à 0 car elle représente la taille la plus grande.

```
double marginHorizontal = (boundSize - boundSymbol.Width) / 2;
double marginVertical = (boundSize - boundSymbol.Height) / 2;

foreach (var path in paths)
{
  ...
  foreach (var point in path)
  {
    ...
    var x = point.X + marginHorizontal;
    var y = point.Y + marginVertical;
    ...
  }
}
```

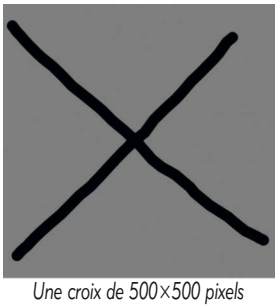
Normalisation

Pour faciliter les calculs on va, normaliser les valeurs des points, c'est à dire les transformer en valeurs comprises entre 0 et 1. Ainsi, que le symbole soit grand ou petit, positionner au centre ou en bas n'aura plus d'importance.
En utilisant le champ boundSymbol qui contient la position et la taille du symbole et le champ boundSize la plus grande taille il est facile de parcourir l'ensemble des points pour les transformer :

```
foreach (var path in paths)
{
  var normalizedPath = new List<Point>();
  // tous les traits normalisés se trouvent ici
  normalizedPaths.Add(normalizedPath);

  foreach (var point in path)
  {
    var normalizedPoint = new Point();

    var x = point.X + marginHorizontal;
    var y = point.Y + marginVertical;

   // les points x,y sont maintenant compris entre 0 et 1
    normalizedPoint.X = ((x - boundSymbol.Left) / boundSize);
    normalizedPoint.Y = ((y - boundSymbol.Top) / boundSize);

    normalizedPath.Add(normalizedPoint);
  }
}
```

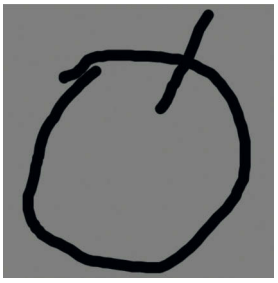
Dessiner le symbole

Maintenant que le tracé est normalisé, on peut multiplier chaque point de chaque trait par la taille de l'image voulue (ici 500×500 pixels). On dessine ensuite chacun des traits. Pour Windows, en UWP par exemple, on utilisera le contrôle Polyline pour dessiner chaque trait. Ces Polylines seront ensuite ajoutées comme enfants d'un contrôle dédié au rendu de symbole. Le symbole est désormais dessiné et est enregistrable au format jpeg en effectuant une capture de l'écran (RenderTargetBitmap).



Une croix de 500×500 pixels

Un rond dessiné à la souris



Une bombe

Configurer CustomVision

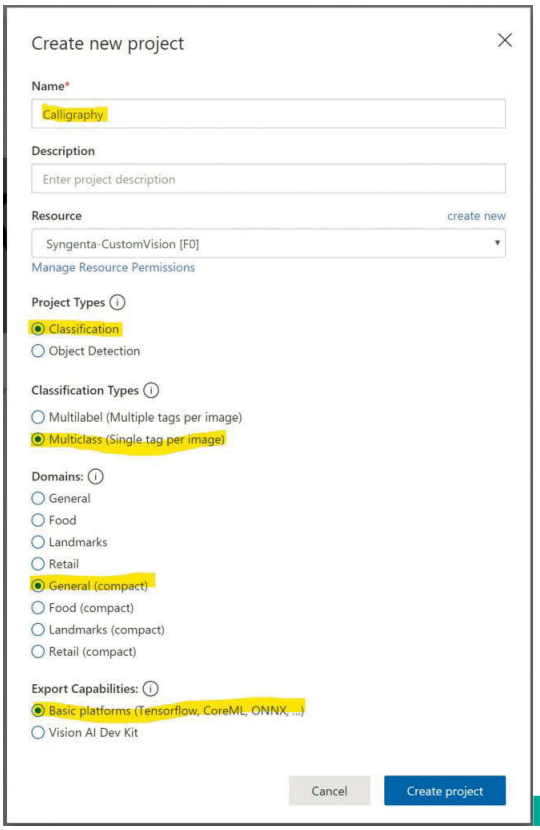
On dispose, à présent, d'un outil capable de tracer un symbole enregistrable sous forme d'image et l'on est prêt à configurer CustomVision. Pour cela, rendez-vous sur :

https://www.customvision.ai/

On se connecte avec un compte Microsoft et l'on ajoute un nouveau projet (new project). Une boîte de dialogue s'affiche : 1

Dans Project Types, on a le choix entre **Classification** et **Object Detection**. Classification est un type de projet qui se donne pour objectif de déterminer la présence ou non d'un objet sur une image. Par exemple y-a-t-il un chien sur la photo ? Object Detection sera enrichi de la position et de la taille du chien sur la photo. Classification est plus simple à mettre en œuvre et plus rapide en termes de traitement que Object Detection.

Pour notre projet, les symboles à détecter étant tous de la même taille, la classification sera le type le plus adapté. Un tag (ou étiquette) correspond à un objet que l'on désire détecter dans une



1

photo. Lorsque l'on choisit un projet de type Classification, il est possible de détecter un ou plusieurs tags dans une même photo. Dans notre projet, l'image ne comportera qu'un seul symbole à détecter ce qui correspond à l'option **Multiclass** dans **Classification Types**.

CustomVision ne part jamais d'un modèle vierge pour créer un projet. En effet les prédictions sont meilleures lorsque le modèle est pré-entraîné. Selon le contexte (**Domain**) de son projet on partira plutôt sur de l'alimentaire (**Food**), des structures repérables comme la tour Eiffel par exemple (**Landmarks**) ou des objets commercialisables (**Retails**). Si vous n'êtes pas sûr ou si aucun des contextes ne vous semble correspondre à vos besoins choisissez **General**. Pour ce projet j'ai opté pour General bien que LandMarks pourrait s'avérer intéressant pour mieux détecter les symboles. Comme ce choix peut être changé par la suite, on peut poursuivre sans trop de pression.

Pour finir la configuration du Domain, on doit également choisir si l'on souhaite un modèle online ou offline (Compact). Comme notre application de Morpion doit fonctionner sans internet, on choisit le modèle **General (Compact)**. Et l'on configure ses capacités d'exportation en **Basic platforms** afin de bénéficier du format d'exportation au format fichier. Même si nous ne l'utilisons pas sur ce projet, gardez à l'esprit que la version en ligne est beaucoup plus performante que la version offline.

Préparez le jeu de données

A l'aide de notre application de saisie on générera automatiquement pour chaque symbole une image que l'on clonera, par code, en effectuant une rotation de 90°,180° et 270°. Cela nous permettra de de minimiser le nombre de symbole que nous devrions pro-

duire manuellement. Ainsi à partir d'une l'image de base on générera les trois suivantes :

Image de base

Rotation 90°

Rotation 180°

Rotation 270°

Pour une image de symbole dessiné, notre application en génèrera donc quatre.

Dans d'autre cas que le nôtre, il pourrait être également intéressant d'ajouter du flou, de changer les contrastes ou toutes autres retouches d'image permettant à l'IA de mieux comprendre ce que l'on souhaite rechercher.

Il est important de comprendre que si vous désirez détecter un chien ou un chat et que dans les images que vous envoyez le chien apparaît souvent sur fond rouge et les chats sur fond vert, CustomVision pourrait penser que c'est la couleur de fond qui est déterminante pour reconnaître ce qu'est un chat ou chien. Dans le cas de nos symboles, il n'y aura pas de problème puisque le tracé sera toujours noir sur fond gris.

Vous trouverez d'autres informations sur la nature des images à envoyer à cette adresse :

https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/cognitive-services/Custom-Vision-Service/includes/choose-training-images.md

Processus d'entraînement de CustomVision

Maintenant que le projet est créé, on doit entraîner notre modèle afin qu'il puisse reconnaître nos symboles. Le processus d'entraînement se déroule comme suit :

- Upload des images dans CustomVision et association à un tag ;
- Entraînement du modèle à partir des images ;
- Evaluation du modèle en effectuant des prédictions.

Création et Upload du jeu de données

En appuyant sur « Add Image » on upload des images en lui associant un tag. Il est possible d'ajouter plusieurs images à la fois ce qui est très pratique pour une importation en masse.

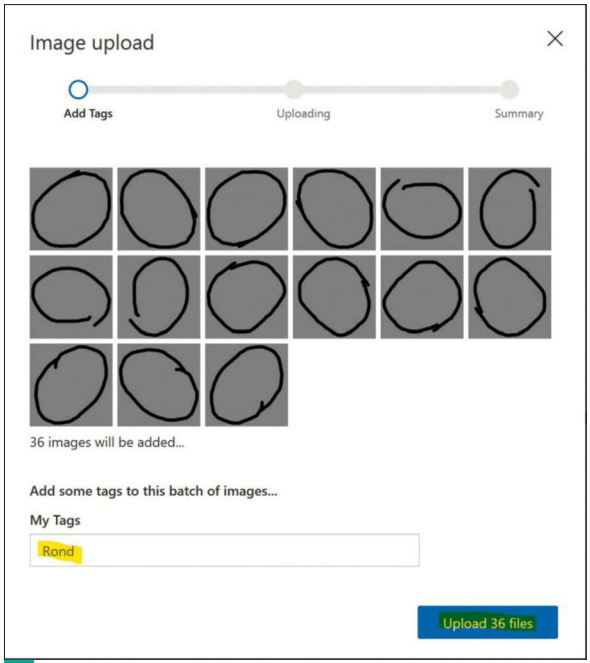
Avec notre application de saisie on commencera à générer les trois types de symboles du jeu (rond, croix, bombe) que l'on répartira dans trois répertoires nommés par type. Comptez 20 images originales par répertoires (soit 20×4 images par répertoire si l'on compte les images crées dynamiquement par l'application de saisie). On ajoutera ensuite ces images dans CustomVision par répertoires en lui appliquant un tag correspondant au type d'images (rond, croix, bombe). 2

Entraîner son modèle

Une fois les images ajoutées vous devriez obtenir la configuration suivante (avec beaucoup moins d'images néanmoins). On voit, également, une catégorie présente dont nous n'avons pas encore parlé « Other » et sur laquelle nous reviendrons. 3

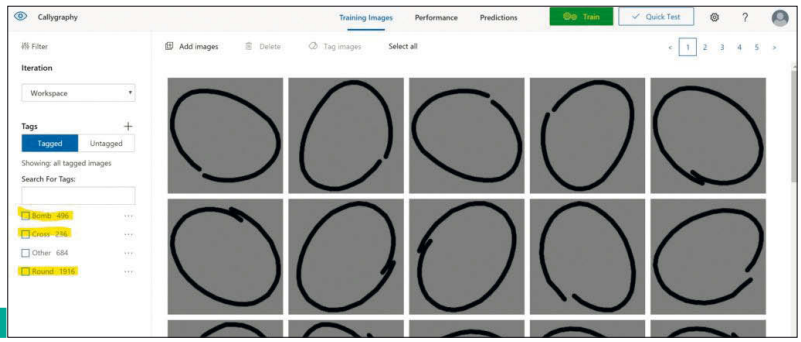
On voit les tags sur la gauche avec le nombre d'images associées. En haut, à droite, en vert, le bouton « Train ».

Il est possible désormais d'entraîner notre modèle en appuyant sur le bouton vert « Train ».

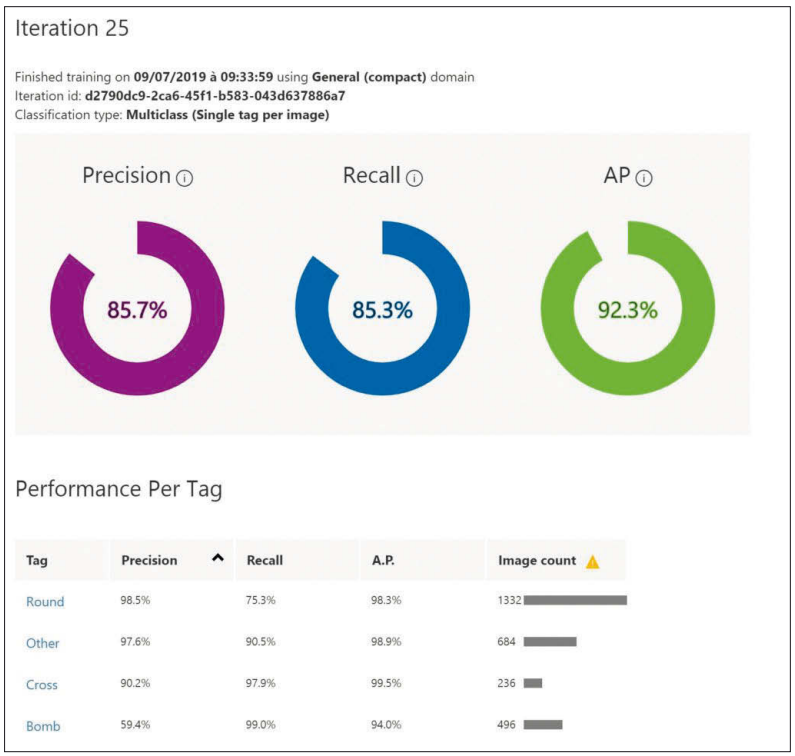


2 *On upload à partir du répertoire « Rond » 36 images qui seront associées au tag « Rond »*

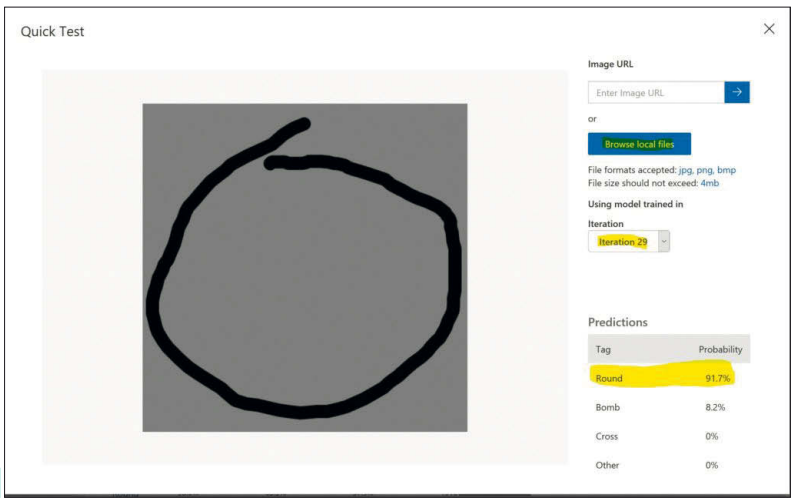
Il y a deux types d'entraînements proposés : Fast Training et Advanced Training. Le premier permet de faire un test rapide sur la qualité du modèle. Dès que vous êtes satisfait de modèle, vous pouvez effectuer un dernier entraînement en mode Advanced Training. Celui-ci vous proposera d'augmenter le temps d'entraînement jusqu'à 24 heures et ainsi d'obtenir un modèle encore plus robuste.



3



4



5

En appuyant sur ce bouton, on va créer une itération, c'est à dire un modèle contenant les images en cours. Pour qu'une nouvelle itération soit créée, il faut de nouvelles images à proposer à CustomVision. Les itérations permettent de voir la progression (ou la régression) dans la prédiction de notre modèle. Une prédiction s'exprime par une probabilité exprimée de 0 à 1. Plus cette probabilité est élevée (proche de 1) plus le modèle est sûr de sa prédiction.

Il est toujours difficile de juger de la qualité de son modèle car CustomVision est, en quelque sorte, une boîte noire. Néanmoins chaque itération comporte un score indiquant la performance de son modèle. 4

Ce score se divise en trois catégories :

- **Precision** correspond au ratio des symboles correctement détectés sur le nombre total de symboles détectés (sur l'ensemble du jeu de donnée). La précision varie selon le seuil de précision de probabilité accepté. Ce seuil (Thresold) est représenté par une jauge dans l'interface de CustomVision.
- **Recall** est le pourcentage de symboles correctement détectés sur le nombre de symboles qu'il y avait à détecter (sur l'ensemble du jeu de donnée). Il varie également selon le seuil de l'interface de CustomVision.
- **AP** est un résumé de la précision et du Recall à différents seuils de probabilité de prédiction. C'est la note globale du modèle. Elle ne varie pas selon un seuil.

Malgré ce score, il n'est pas toujours simple de savoir comment se comporte le modèle. Le mieux est encore de tester chaque itération. Le bouton « Quick Test » permet de se donner une idée rapide de la qualité de l'itération. 5

On sélectionne l'itération si nécessaire (par défaut c'est la dernière) puis on appuie sur le bouton « Browse Local File » pour charger l'image à tester. Il ne reste plus qu'à attendre le résultat. Ici l'image est détectée en tant que symbole Rond avec une probabilité de 97%.

Comme il arrive parfois que le modèle régresse, il peut être intéressant de tenir un journal de test de ses itérations.

Iteration 1 : 100 cross / 100 round : loss = 1 / bonne reconnaissance mais peut se faire duper avec un A par exemple. À partir de l'itération 2, on intégre la Bomb qui est très proche de Round.
Iteration 2 : Intégration de Bomb, un symbole très proche de Round. Round ne sort pas souvent / Beaucoup de faux positifs.
Iteration 3 : Intégration de plus de Round : Round sort plus souvent mais Bomb est encore pris pour Round / Beaucoup de faux positifs.
Iteration 4 : Intégration de plus de Round. Beaucoup de faux positifs.

À partir de l'**itération 5**, on ajoute le tag « **other** » à notre modèle. C'est un tag fourre-tout qui permet d'y intégrer en petits nombres des symboles parasites qui empêchent la bonne détection des symboles. Par exemple 27 dessins de triangles suffisent pour qu'un triangle ne soit plus pris pour un rond.

Iteration 5 : intégration de other. 300 symboles de ligne, gribouillis, triangle, rectangle sont ajoutés. Plus de faux positifs à part croix courbé qui est pris pour une bombe.

Iteration 6 : ajout de croix car lorsqu'il est courbé, il est pris comme une bombe (+40). Plus de faux positif (mais la bombe est moins bien détectée).
Iteration 7 : ajout de croix pour arriver à 236 (+100) + quelques bombes. La bombe est bien détectée à 68%.
Iteration 8 : ajout de carré dans other (+40). la bombe est bien détectée à 90%.
Iteration 9 : Pas de 9 !
Iteration 10 : ajout de bombes + 50 / Round + 50 / Other + 50 (Carrée + croix).
Iteration 11 : intégration de dessin strié dans other + 30 car ils sont pris pour des croix.
Iteration 12 : amélioration de other pour ne pas détecter de bombe (symbole coeur).
Iteration 13 : amélioration de other pour ne pas détecter de croix/bombe (symbole A et B)
Iteration 14 : amélioration de other pour ne pas détecter de croix/bombe (dièse).
A partir de l'itération 14, on possède un modèle qui répond à nos exigences en termes de détection de symbole. Il est temps de penser à l'intégration.

Export du modèle sous forme de fichier

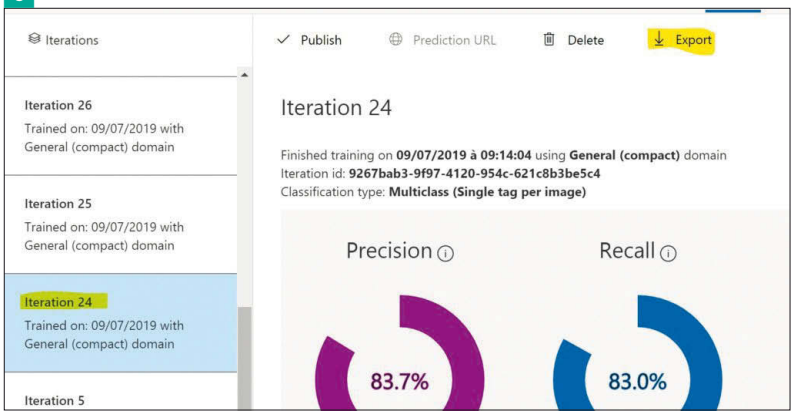
Lorsque le modèle semble satisfaisant, on peut l'exporter sous forme d'un fichier. Cette option n'est disponible que pour les modèles de domaine de type General (Compact) comme vu dans le paragraphe Configurer CustomVision. Il est possible de changer le domaine en cas de problème. L'exportation s'effectue sur la page de l'itération souhaitée en appuyant sur le bouton Export. 6
On choisit ensuite la plateforme d'exportation. La plupart des plateformes du marché sont supportées par Custom Vision (Windows, iOS, Android, ...). Notre application s'exécutant dans un environnement Windows nous privilégierons le format ONNX. 7
Il ne reste plus qu'à exporter le modèle au format ONNX1.2 en appuyant sur le bouton Export puis une fois le modèle généré sur le bouton « Download ». 8
Un fichier ONNX est alors directement téléchargé sur votre ordinateur.

Utilisation du fichier ONNX dans une application UWP (Windows Desktop)

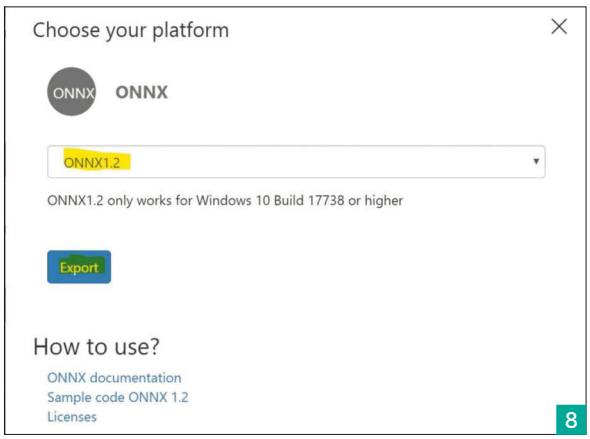
Pour intégrer le fichier ONNX dans une application UWP il suffit de le déposer dans le répertoire Assets et de positionner sa propriété « Build Action » à « Content ».
Dans la version de Visual Studio 2017, il suffisait d'ajouter un fichier ONNX dans le dossier Assets pour qu'il génère une classe permettant de charger le modèle et de lancer une prédiction. Ce n'est plus le cas dans la version 2019.
Heureusement, on peut accéder au Sample du Code ONNX 1.2 grâce au lien contenu dans la section « How to use ? » de la fenêtre d'exportation du fichier ONNX. Mais le plus simple reste encore de vous montrer la classe générée à l'origine par Visual Studio 2017 qui est beaucoup plus pratique :

```
using System;
using System.Collections.Generic;
```

6



7



8

```
using System.Threading.Tasks;
using Windows.AI.MachineLearning;
using Windows.Media;
using Windows.Storage;

namespace Synergiz.AI.Callygraphy.Tools
```

```
{
  public sealed class OnnxModelInput
  {
    public VideoFrame Data { get; set; }
  }

  public sealed class ModelOutput
  {
    public TensorString ClassLabel = TensorString.Create(new long[] { 1, 1 });
    public IList<IDictionary<string, float>> Loss = new List<IDictionary<string, float>>();
  }

  public sealed class OnnxModel
  {
    private LearningModel _learningModel = null;
    private LearningModelSession _session;

    public static async Task<OnnxModel> CreateOnnxModel(StorageFile file)
    {
      LearningModel learningModel = null;

      try
      {
        learningModel = await LearningModel.LoadFromStorageFileAsync(file);
      }
      catch (Exception e)
      {
        var exceptionStr = e.ToString();
        System.Console.WriteLine(exceptionStr);
        throw e;
      }
      return new OnnxModel()
      {
        _learningModel = learningModel,
        _session = new LearningModelSession(learningModel)
      };
    }

    public async Task<ModelOutput> EvaluateAsync(OnnxModelInput input)
    {
      var output = new ModelOutput();
      var binding = new LearningModelBinding(_session);
      binding.Bind("data", input.Data);
      binding.Bind("classLabel", output.ClassLabel);
      binding.Bind("loss", output.Loss);
      LearningModelEvaluationResult evalResult = await _session.EvaluateAsync(binding, "0");

      return output;
    }
  }
}
```

Sans rentrer dans les détails de l'implémentation, il suffit de lancer une fois la méthode CreateOnnxModel pour charger notre modèle, puis un appel à la méthode EvaluateAsync pour effectuer une prédiction.

```
var file = await StorageFile.GetFileFromApplicationUriAsync(new Uri("ms-appx:///Assets/
Callygraphy-14.onnx"));
var predictionModel = await OnnxModel.CreateOnnxModel(file);
// frame est de type VideoFrame c'est à dire un objet en provenance d'une vidéo ou d'une
 //caméra. Une image est facilement transformable en VideoFrame via
 //VideoFrame.CreateWithSoftwareBitmap(bitmap)
var output = await predictionModel.EvaluateAsync(new OnnxModelInput() { Data = frame });
// traitement du résultat
var vector = output.ClassLabel.GetAsVectorView();
var tagName = vector[0];
var probability = output.Loss[0][product];
Debug.WriteLine("Product=" + tagName + " Loss=" + probability);
```

Le résultat nous donne comme résultat le tag « Round », « Cross », « Bomb » ou « Other » et la probabilité selon le modèle que la réponse soit correcte.

Rendu final

Maintenant que nous savons prédire le symbole dessiné dans une image, on a tous les éléments pour réaliser notre jeu du morpion. Nous utiliserons le contrôle permettant de dessiner un symbole sur l'ensemble de la surface du jeu. Celui-ci le normalisera puis générera une image de 500 par 500 pixels qui sera envoyée dans le modèle de prédiction. Selon le résultat on affichera une étoile de mer pour la croix, une coquille saint jacques pour le rond et la bombe servira à effectuer un reset du jeu. La position de chaque pion sera calculée en fonction du centre du symbole dessiné sur le contrôle.

Conclusion

Custom Vision nous a permis, sans algorithme complexe, de détecter des formes diverses bien que relativement proches. Il est facile d'enrichir notre modèle de nouvelles formes si on le désire, et leurs détections restent souples (une ellipse peut être considérée comme un rond). On l'a vu tout au long de cet article, la difficulté technique ne se trouve pas dans l'intégration du modèle mais plutôt dans son entraînement. Il est donc nécessaire de trouver le meilleur moyen de générer beaucoup de données à moindre coût car plus le modèle sera nourri de données originales, plus il sera performant. •

Merci à Vincent Mugnier pour sa relecture attentive

Benoît Prieur
Développeur indépendant pour sa société Soarthec. Il est par ailleurs l'auteur du livre Pygame : Initiez-vous au développement de jeux vidéo en Python publié aux éditions ENI en novembre 2019.

Pygame : les principales notions à connaître pour développer vos jeux vidéo en Python

Pygame est le framework Python dédié au développement de jeux vidéo en deux dimensions même s'il est possible en utilisant celui-ci de fabriquer un jeu 3D. Il permet en effet en quelques lignes de code de fabriquer un casse-briques ou un jeu de labyrinthe, avec tout ce qui est nécessaire pour obtenir un jeu amusant, au niveau graphique comme au niveau sonore.

niveau 100

Basé sur SDL et aucunement sur OpenGL (comme on le lit parfois à tort), Pygame trouve son origine à l'orée des années 2000. Pete Shinners recycle alors un projet existant nommé PySDL pour en faire Pygame dont le code est d'emblée placé sous licence libre. Si le framework n'est pas tant utilisé par les studios de développement de jeux vidéo ou par les professionnels en général, on peut citer tout de même quelques jeux très populaires développés avec Pygame comme *Frets on Fire* (clone de *Guitar Hero*) ou encore *Dangerous High School Girls in Trouble!*
L'un des grands intérêts de Pygame est qu'il permet la réalisation de jeux utilisables sur le système d'exploitation de son choix, Linux, Windows ou macOS. En termes de déploiement, on se retrouve dans le schéma habituel d'un développement en langage Python ; l'usage d'un environnement virtuel est à privilégier et le lancement du jeu se fait en première approche en exécutant en ligne de commande un fichier d'extension .py.
Outre sa portabilité, le framework constitue un point d'entrée fréquent dans l'apprentissage du langage Python entre autres grâce à sa relative facilité de prise en main. Le monde des *makers* se l'approprie également par exemple pour offrir un simulateur graphique à des véhicules connectés conçus avec Raspberry PI ou Arduino.
Son intérêt pédagogique réside enfin dans son approche concrète de la programmation orientée objet. Une fois cette base acquise, il s'agit essentiellement de connaître et de comprendre la notion de Sprite, celle de gestion des collisions et celle de boucle de jeu. Après l'explication de chacune de ces notions, nous réaliserons un petit jeu en Pygame ; il s'agit du jeu mythique du *Snake* (le jeu du serpent).

La boucle de jeu

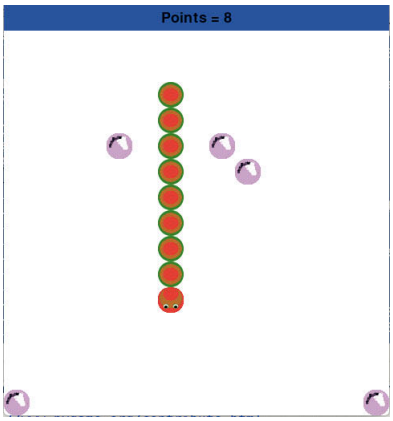
Parfois également appelée boucle d'animation, la boucle de jeu se retrouve dans (presque) tous les jeux vidéo et constitue à ce titre une sorte de colonne vertébrale. Elle correspond en général à une boucle infinie que l'on réalise avec une boucle *while*. On peut identifier les actions suivantes comme étant réalisées à chaque itération de la boucle infinie.

1. Vérifier que les conditions d'arrêt ne sont pas atteintes, et si elles le sont, interrompre la boucle.
2. Mettre à jour les ressources nécessaires pour l'itération courante.
3. Obtenir les entrées, soit issues du système, soit issues de l'interaction avec la personne en train de jouer.
4. Mettre à jour l'ensemble des entités qui caractérisent le jeu.
5. Rafraîchir l'écran de jeu.

On travaille en général avec Pygame avec une fréquence de 30 itérations par seconde.

La notion de *sprite* dans Pygame

On souhaite lorsque l'on développe un jeu vidéo, associer un emplacement de la fenêtre de jeu, avec une représentation graphique et un ensemble de propriétés : dans l'itération courante, le héros ou l'héroïne, se situe à tel endroit et possède telle arme et tel niveau de points de vie (les propriétés). On peut donc envisager un *sprite* comme un objet qui associe un emplacement, une représentation graphique (telle ou telle image par exemple) et un ensemble de propriétés. Ces propriétés peuvent être un nom, un texte, des booléens qui caractérisent l'objet en question (par exemple, l'objet peut se déplacer, ou non). Cette notion est évidemment particulièrement compatible avec le paradigme objet. Pygame propose d'ailleurs une classe *Sprite* dont les objets de notre jeu peuvent hériter. Pygame propose également la notion de groupe (*Group*) de *sprites*. Cette dernière notion est très utile lorsqu'il s'agit de gérer les collisions.



Le jeu en action

La gestion des collisions avec Pygame

On entend par gestion des collisions l'idée de déterminer si un objet est entré en contact avec tel autre. Si le but de notre jeu est de faire rebondir un ballon, il va nous falloir déterminer si le ballon a rencontré le bord inférieur de notre fenêtre de jeu. On se frotte alors rapidement à des calculs géométriques qui peuvent s'avérer compliqués et fastidieux. Quand des dizaines de cibles sont visées par plusieurs tireurs, c'est autant de collisions potentielles à analyser : chaque cible a-t-elle été touchée par un des projectiles ? Pygame propose une gestion simplifiée des collisions basée sur la notion de *sprite* et sur celle de groupe de *sprites*. En effet l'objet *Sprite* possède de plusieurs méthodes permettant de savoir si oui ou non, l'instance courante de *Sprite* collisionne tel autre *sprite* ou tel groupe de *sprites*.

Présentation du jeu du *Snake*

Dans ce jeu datant de 1976, le joueur pilote un serpent grâce aux touches directionnelles et l'oriente vers des éléments de nourriture disposés aléatoirement et qui ne restent affichés à l'écran que

quelques secondes. À chaque fois que le serpent parvient à manger un élément de nourriture, il marque un point et il voit son corps grandir. Cette dernière conséquence pose problème au bout d'un certain temps, car il est de plus en plus difficile pour le serpent d'éviter de collisionner son propre corps ce qui correspond à la fin de la partie.

Définition des objets héritant de la classe Pygame *Sprite*

On commence par définir les images illustrant nos différents types de *sprites* à savoir la tête du serpent à qui nous ferons subir une rotation selon la direction empruntée, un élément de corps du serpent ainsi qu'un élément de nourriture.

On commence par créer la classe NOURRITURE qui hérite de la classe *Sprite* de Pygame. On lui associe l'image représentant la nourriture au sein de son initialiseur. Dans cette méthode est défini l'emplacement aléatoire de cet élément de nourriture. Dans la surcharge de la méthode *update*, on teste si l'élément de nourriture courant doit être supprimé ou non (en fait chaque élément reste affiché durant une durée de quelques secondes définie dans la variable DUREE_NOURRITURE_DISPARITION). Cette méthode *update* est appelée à chaque itération de la boucle de jeu.

```
class NOURRITURE(pygame.sprite.Sprite):
 def __init__(self):
  pygame.sprite.Sprite.__init__(self)

  self.image = pygame.image.load("NOURRITURE.png").convert_alpha()

  self.rect = self.image.get_rect()
  self.rect.y = random.randint(0, TUILE_NOMBRE - 1) * TUILE_TAILLE + SCORE_HAUTEUR
  self.rect.x = random.randint(0, TUILE_NOMBRE - 1) * TUILE_TAILLE

  self.time = pygame.time.get_ticks()

 def update(self):
  if pygame.time.get_ticks() - self.time > DUREE_NOURRITURE_DISPARITION:
   self.kill()
```

On poursuit en définissant la classe CORPS sachant que le serpent sera composé de plusieurs instances de CORPS et qu'à chaque fois que le serpent mangera de la nourriture, une instance de CORPS sera ajoutée au serpent. Rien de très compliqué pour cette classe. Une fois encore on l'a fait hériter de la classe Pygame *Sprite*. On associe l'image adéquate et on adjoint à la classe des accesseurs à même de mettre à jour les coordonnées de l'élément de corps courant.

```
class CORPS(pygame.sprite.Sprite):
 def __init__(self, x, y):
  pygame.sprite.Sprite.__init__(self)

  self.image = pygame.image.load("CORPS.png").convert_alpha()
  self.rect = self.image.get_rect()
  self.rect.x = x
  self.rect.y = y

 def GET_X(self):
  return self.rect.x

 def GET_Y(self):
  return self.rect.y

 def set_xy(self, x, y):
  self.rect.x = x
  self.rect.y = y
```

Enfin, on définit la classe relative au serpent lui-même ; on précise l'image de la tête du serpent. On positionne celle-ci au démarrage du jeu au centre de la fenêtre de jeu. Une méthode est dédiée à l'ajout d'un nouveau corps, méthode appelée lorsque le serpent mange de la nourriture. Quand cela se produit on ajoute ce nouveau *sprite* à un groupe global de *sprites* ainsi qu'à un groupe représentant l'ensemble des *sprites* formant le corps du serpent. On définit également une méthode *update* en charge d'une part de gérer la rotation de la tête du serpent selon la touche directionnelle appuyée (respectivement le code «D» pour la flèche droite, «G» pour la flèche gauche, «H» pour la flèche vers le haut et «B» pour la flèche vers le bas). On met ensuite à jour les coordonnées de chaque élément composant le serpent puis on vérifie certaines collisions éventuelles :

- La collision entre la tête du serpent et le corps du serpent, auquel cas la partie s'arrête. Cette vérification utilise la méthode *spritecollide* qui évalue la collision entre un *sprite* (l'instance courante) et un groupe (celui relatif au serpent).
- La collision entre la tête du serpent et un élément de nourriture, auquel cas on gagne un point, on instancie un nouvel élément de corps et on supprime la nourriture absorbée. Cette vérification utilise également la méthode spritecollide qui évalue la collision entre un sprite (l'instance courante) et un groupe (celui relatif à l'ensemble des différents éléments de nourriture disposés sur la fenêtre de jeu).

```
class SERPENT(pygame.sprite.Sprite):
 def __init__(self):
  pygame.sprite.Sprite.__init__(self)

  self.TETE = pygame.image.load("TETE.png").convert_alpha()
  self.image = self.TETE

  self.rect = self.image.get_rect()
  self.rect.y = (TUILE_NOMBRE / 2) * TUILE_TAILLE
  self.rect.x = (TUILE_NOMBRE / 2) * TUILE_TAILLE

  self.POINTS = 0
  self.DIRECTION = 'G'
  self.TERMINE = False

 def AJOUTER_NOUVEAU_CORPS(self):
  nouveau_corps = CORPS(self.rect.x, self.rect.y)
```

```
 LISTE_SERPENT.add(nouveau_corps)
 LISTE_GLOBALE_SPRITES.add(nouveau_corps)

def update(self):
 COORD_COURANTE_X = self.rect.x
 COORD_COURANTE_Y = self.rect.y

 if self.DIRECTION == 'G':
  self.image = pygame.transform.rotate(self.TETE, 90)
  self.rect.x -= TUILE_TAILLE
 elif self.DIRECTION == 'D':
  self.image = pygame.transform.rotate(self.TETE, -90)
  self.rect.x += TUILE_TAILLE
 elif self.DIRECTION == 'H':
  self.image = pygame.transform.rotate(self.TETE, 0)
  self.rect.y -= TUILE_TAILLE
 elif self.DIRECTION == 'B':
  self.image = pygame.transform.rotate(self.TETE, 180)
  self.rect.y += TUILE_TAILLE

 for ELT in LISTE_SERPENT:
  x = ELT.GET_X()
  y = ELT.GET_Y()
  ELT.set_xy(COORD_COURANTE_X, COORD_COURANTE_Y)
  COORD_COURANTE_X = x
  COORD_COURANTE_Y = y

 if self.rect.x >= LARGEUR:
  self.rect.x = 0
 elif self.rect.x < 0:
  self.rect.x = LARGEUR - TUILE_TAILLE
 elif self.rect.y >= HAUTEUR:
  self.rect.y = SCORE_HAUTEUR
 elif self.rect.y < SCORE_HAUTEUR:
  self.rect.y = HAUTEUR - SCORE_HAUTEUR

LISTE_COLLISION_SERPENT = pygame.sprite.spritecollide(self, LISTE_SERPENT, False)
 if len(LISTE_COLLISION_SERPENT):
  print("Perdu")
  self.TERMINE = True

 LISTE_COLLISION_NOURRITURE = pygame.sprite.spritecollide(self, LISTE_NOURRITURE, False)
 for nourriture in LISTE_COLLISION_NOURRITURE:
  nourriture.kill()
  self.AJOUTER_NOUVEAU_CORPS()
  self.POINTS += POINT_UNITE
```

Le programme lui-même et ses différentes variables

Il est bien sûr nécessaire de référencer le module Pygame et de l'initialiser. On en profite également pour définir un certain nombre de variables comme les couleurs, les tailles de *sprites*, les dimensions de la fenêtre de jeu ainsi que combien de points sont gagnés à chaque absorption de nourriture et la durée d'affichage d'un élément de nourriture (5,5 secondes). On crée alors la fenêtre de jeu.

```
import pygame, random, sys

NOIR = (0, 0, 0)
BLANC = (255, 255, 255)

TUILE_TAILLE = 32
TUILE_NOMBRE = 15
SCORE_HAUTEUR = 32

LARGEUR = TUILE_TAILLE * TUILE_NOMBRE
HAUTEUR = TUILE_TAILLE * TUILE_NOMBRE + SCORE_HAUTEUR

POINT_UNITE = 1
DUREE_NOURRITURE_DISPARITION = 5500

pygame.init()

screen = pygame.display.set_mode([LARGEUR, HAUTEUR])
pygame.display.set_caption('Le jeu du serpent')
```

Les groupes de *sprites* utilisés dans la gestion des collisions

Nous avons en effet besoin de définir plusieurs groupes de *sprites* utilisés pour tester les différentes éventuelles collisions. Nous avons besoin des groupes suivants, par ailleurs mentionnés et utilisés précédemment :

- Le groupe des éléments de corps composant le serpent : *LISTE_SERPENT*
- Le groupe des éléments de nourriture affichés à l'écran : *LISTE_NOURRITURE*
- La liste de tous les *sprites* utilisés dans le jeu : *LISTE_GLOBALE_SPRITES*

On instancie le serpent de notre jeu (instance unique nommée *_serpent*) ce qui est matérialisé par l'affichage de la tête du serpent au centre de la fenêtre de jeu. On ajoute évidemment cette instance au groupe de tous les *sprites*.

```
LISTE_SERPENT = pygame.sprite.Group()
LISTE_NOURRITURE = pygame.sprite.Group()
LISTE_GLOBALE_SPRITES = pygame.sprite.Group()

_serpent = SERPENT()
LISTE_GLOBALE_SPRITES.add(_serpent)
```

La boucle de jeu de notre jeu du serpent

On définit notre boucle de jeu grâce à une boucle while infinie. Elle s'interrompt si on appuie sur la touche ESC qui correspond à un type d'évènement *pygame.QUIT*. On capture également les évènements clavier relatifs à l'appui sur les touches de flèches. On met alors à jour l'attribut *DIRECTION*. On affiche éventuellement un nouvel élément de nourriture (un tirage aléatoire le décide ou non). Si c'est le cas, on vérifie que ses coordonnées (elles aussi définies aléatoirement) ne correspondent pas à un autre *sprite* du jeu. On utilise donc à nouveau la méthode *spritecollide* pour prévenir cette collision problématique. Si tel n'est pas le cas, on ajoute ce nouvel élément aux différents groupes. On appelle alors la méthode *update* sur le groupe global (*LISTE_GLOBALE_SPRITES.update*) ce qui

permet d'appeler chaque méthode update particulière de chaque *sprite* stocké dans le groupe. Et on redessine chaque sprite à l'écran (*LISTE_GLOBALE_SPRITES.draw*). Enfin on actualise la fenêtre de jeu, ceci à chaque itération, en utilisant la fonction *flip* (*pygame.display.flip*).

```
while not TERMINE:
 for event in pygame.event.get():
  if event.type == pygame.QUIT:
   TERMINE = True
  elif event.type == pygame.KEYDOWN:
   if event.key == pygame.K_LEFT and _serpent.DIRECTION != 'D':
    _serpent.DIRECTION = 'G'
    break
   elif event.key == pygame.K_RIGHT and _serpent.DIRECTION != 'G':
    _serpent.DIRECTION = 'D'
    break
   elif event.key == pygame.K_UP and _serpent.DIRECTION != 'B':
    _serpent.DIRECTION = 'H'
    break
   elif event.key == pygame.K_DOWN and _serpent.DIRECTION != 'H':
    _serpent.DIRECTION = 'B'
    break

 if random.randint(0, 18) == 0:
  _nourriture = NOURRITURE()

 LISTE_CONFLIT = pygame.sprite.spritecollide(_nourriture, LISTE_GLOBALE_SPRITES, False)
 if len(LISTE_CONFLIT) == 0:
  LISTE_GLOBALE_SPRITES.add(_nourriture)
  LISTE_NOURRITURE.add(_nourriture)

 LISTE_GLOBALE_SPRITES.update()
 screen.fill(BLANC)

 LISTE_GLOBALE_SPRITES.draw(screen)
 AFFICHER_SCORE()

 if _serpent.TERMINE:
  pygame.time.wait(5000)
  TERMINE = True

 pygame.display.flip()
 clock.tick(7)

print("Votre score : %d points" % _serpent.POINTS)
pygame.quit()
```

A vous de jouer !

Guillaume Saint Romas
Expert technique
SQLI DIGITAL EXPERIENCE

Partie 2

DevExtreme : une suite de composants à adopter de toute urgence pour développer des web apps

Lors d'un nouveau projet de développement logiciel sur client léger, il est fréquent de recourir à des librairies d'éléments complets, évolutifs et personnalisables afin d'accroître la rentabilité en accélérant le temps en recherche et développement. Nous ne sommes jamais à l'abri de choisir un outil qui ne sera pas pérenne face aux évolutions et qui se fera oublier s'il n'évolue pas au rythme des exigences métiers. Je vous propose de découvrir DevExtreme, une suite de composants JavaScript et HTML5, plusieurs fois élue meilleure suite de composants multiplateformes. Passons à la pratique.

niveau 200

Configuration du « CustomStoreOptions »

Dans l'exemple ci-dessus (voir n°234) de la Datagrid simple, nous avons pu constater combien **il est facile de lier une source de données au composant** au moyen de sa variable « DataSource ». Avec DevExtreme, il est tout aussi rapide de mettre en œuvre sa propre logique d'accès aux données au moyen du « CustomStore » et plus précisément du « CustomStoreOptions ». En effet, le composant « Datagrid » (ainsi que les autres fournis par la solution) peut prendre comme source de données une « DataSource » mais également un « CustomStore » … qui peuvent prendre tous deux comme paramètre optionnel un « CustomStoreOptions ». Nous définissons donc ces options de communication pour les mettre en lien avec notre API.

```
// URL of express server
let url: string = `http://localhost:3000/`;

//#region DataSource | CRUD Actions
// Define CustomStoreOptions
let customStoreOpt : CustomStoreOptions = new CustomStoreOptions({
    key: "id",
    load: () => this.sendRequest(`${url}all`),
    insert: values =>
      this.sendRequest(`${url}add`, "POST", values),
    update: (key, values) =>
      this.sendRequest(`${url}upd/${key}`, "PUT", values),
    remove: key =>
      this.sendRequest(`${url}del/${key}`, "DELETE", {
        key: key
      })
});
// Define DataSource
this.dataSource = new DataSource(customStoreOpt);
//#endregion CustomStoreOptions | CRUD Actions
```

Figure 24 : Déclaration du « CustonStoreOptions ».

Dans cet extrait de code, nous pouvons voir que l'objet passé en paramètre de notre Datasource définit les méthodes à appeler lors des actions de CRUD tels que load, insert, update et remove, ainsi que la clé key, ici « id », permettant d'utiliser la valeur d'identification de l'objet de données remonté par le composant. Utilisons notre méthode générique « sendRequest » afin d'appeler notre API.

```
/**
 * Send Request
 * @param url
 * @param method
 * @param httpParams
 */
public sendRequest(url: string, method: string = "GET", httpParams: any = {}): any {
  let httpOptions = { withCredentials: false, body: httpParams };
  let result;
  switch (method) {
    case "GET":
      result = this.http.get(url, httpOptions);
      break;
    case "PUT":
      result = this.http.put(url, httpParams, httpOptions);
      break;
    case "POST":
      result = this.http.post(url, httpParams, httpOptions);
      break;
    case "DELETE":
      result = this.http.delete(url, httpOptions);
      break;
  }
  return result.toPromise().then((result: any) => {
      return  (method === "GET")
        ? { data: result.exampleData, totalCount: result.totalCount };
        : result;
    }).catch(e => { throw e && e.error && e.error.Message; });
}
```

Figure 25 : Méthode « sendRequest() » envoyant les requêtes d'action à l'API.

Tests et modifications de la vue HTML

Maintenant que nous avons câblé la source de données à notre API, il est temps d'implémenter le tout dans notre composant et de tester. C'est ici que la suite DevExtreme exprime toute son efficacité et tout son intérêt. En quelques lignes HTML, on peut définir toute la mécanique de cet outil « Datagrid ».

Tout d'abord, renseignons à notre composant que nous souhaitons activer le mode d'édition et que seuls les champs « name » et « status » seront soumis à celle-ci.

```
<!—Editing -->
<dxo-editing
    mode="cell"
    [allowAdding]="true"
    [allowUpdating]="true"
    [allowDeleting]="true"
></dxo-editing>
<!-- Columns -->
<dxi-column dataField="id" [allowEditing]="false" caption="ID#"
    [visible]="true" width="50" alignment="center">
</dxi-column>
<dxi-column dataField="name" caption="Name"
    [visible]="true" width="150">
</dxi-column>
<dxi-column dataField="status" caption="Status"
    [visible]="true" width="100" alignment="center">
</dxi-column>
<dxi-column dataField="custom" [allowEditing]="false" caption="Custom"
    cellTemplate="cellTemplate" [allowFiltering]="false" [allowSorting]="false"
    alignment="center" width="150">
</dxi-column>
```

Figure 26 : dx-datagrid: ajout du mode « Editing » et des autorisations d'édition.

Le Mode édition « dxo-editing> » permet ici de préciser quelles actions seront autorisées pour le composant et comment.

- Nous autorisons l'ajout, la modification et la suppression.
- Le mode d'édition est spécifié à « cell » pour cellule (voir figure 27).Dans la définition de nos colonnes nous renseignons par l'attribut « allowEditing » (qui prend un boolean comme valeur) si oui ou non nous souhaitons autoriser la modification.

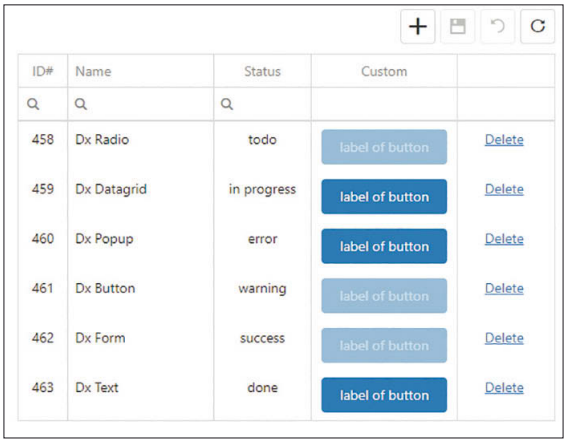


Figure 27 : Aperçu de la "Datagrid" avec le mode édition activé.

À la suite du changement de notre datagrid pour un mode édition, nous pouvons constater que des boutons d'action sont apparus dans la toolbar et qu'une colonne d'action tout à droite a également été rajoutée automatiquement par le composant.

- Ici les boutons d'action « save » et « revert » de la toolbar sont grisés (désactivés) car aucune modification utilisateur n'a été relevée par le composant.

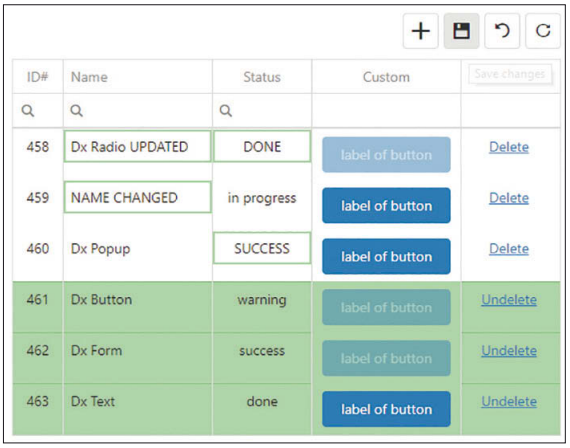


Figure 28 : Edition de la datagrid en mode « Batch » qui est un mode d'édition par lot de cellules.

Dans notre configuration nous sommes en mode « batch», qui traite les modifications par lot de cellules. Nous pouvons donc modifier dans la « datagrid » les valeurs cellule par cellule, avant de sauvegarder ou d'annuler le tout au moyen des boutons d'action présents dans la toolbar.

- Visuellement, le composant encadre en vert les cellules modifiées et surligne les lignes qui vont être effacées lors de la sauvegarde.

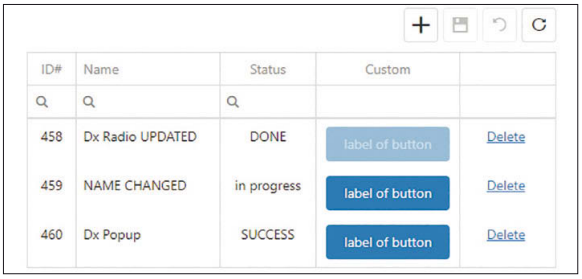


Figure 29 : Aperçu de la Datagrid après modification.

Après sauvegarde, nous nous retrouvons donc avec les données modifiées et supprimées comme préalablement décidé par les actions utilisateur ci-dessus. A noter que l'ajout se fera aussi par cellule dans une nouvelle ligne de Datagrid une fois l'action d'ajout appelée depuis le bouton + de la toolbar.

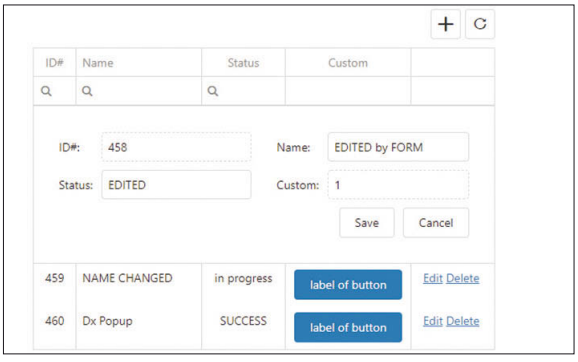


Figure 30 : Datagrid en mode "form" formulaire.

- Ci-dessus, un exemple de la modification en mode « form » pour formulaire. Nous sommes devant un formulaire regroupant tous les champs d'une entrée « ExampleData », avec en bordure pleine les champs modifiables et en pointillés ceux qui ne le sont pas.

En mode formulaire, les boutons d'action « save » et « revert » de

la toolbar ont disparu, et une action « edit » dans la colonne action rajoutée tout à droite de la « datagrid » a fait son apparition.

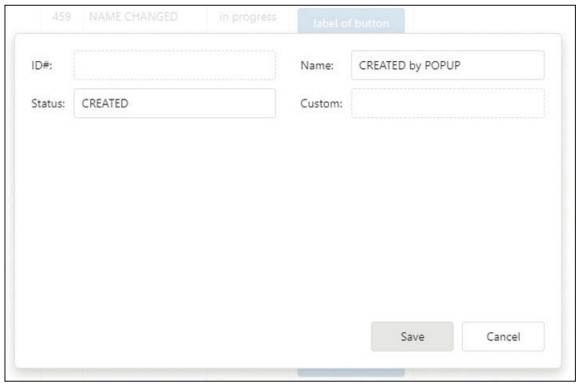


Figure 31 : Datagrid en mode "popup".

- Le mode popup permet les actions de modification au travers d'une popup dédiée.
- Ci-dessus, un exemple de création en mode popup.

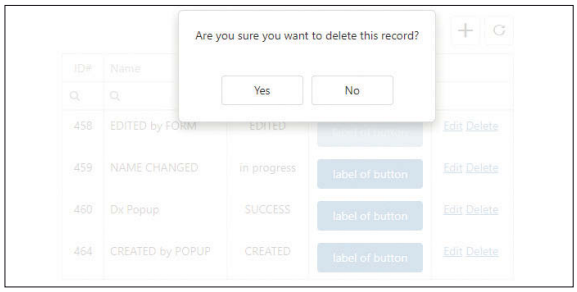


Figure 32 : Popup de confirmation pour la suppression de données.

- Dans tous les modes un popup de confirmation est ouvert avant suppression.

Il ne s'agit que de configurations simples dans cet exemple. Le composant de Datagrid est vraiment abouti et propose tout **un panel d'outils qui se révèleront vraiment très utiles et appréciés des utilisateurs**, tels que les actions de sélection des données, le groupement de données, leur export, la possibilité d'insérer des widgets DevExtreme comme un graphique, une liste déroulante… La liste est longue. Vous vous en doutez, **la plupart des outils proposés sont personnalisables**, des templates html des composants (nous pourrions avoir par exemple des icônes à la place des liens d'action dans la colonne du mode édition), à la position horizontale des éléments rajoutés, comme la colonne d'action ou les boutons de la toolbar.

Personnalisation et ajout des actions d'export et suppression par sélection

Pour illustrer l'intérêt de ce genre d'outil, finissons cet exemple pratique par l'ajout des fonctionnalités d'export et de multi-sélection (pour suppression et choix des données à exporter) à notre Datagrid. Imaginez que votre applicatif utilisant une « Datagrid » est finalisé, entièrement fonctionnel, câblé comme il se doit à votre API et que vos besoins évoluent. Aujourd'hui, il vous serait très utile, d'exporter et/ou de supprimer les données remontées par votre grille de données (qui, comme vu ci-dessus, offre les fonctionnalités pour les ordonner, les filtrer…) et tout ceci par lot d'enregistrement au moyen d'une multi-sélection. Avec ce composant de « Datagrid » proposé par DevExtreme, rien de plus facile, et surtout de plus rapide. **C'est sur cet aspect entièrement évolutif et personnalisable que l'utilisation de cette librairie se révèle un énorme gain de temps dans les développements**.

Sélection multiple et exportation des données

Dans notre déclaration de « Datagrid » côté html, nous ajoutons simplement les options d'export et de sélection comme suit :

```
<!-- Export -->
<dxo-export [enabled]="true" fileName="ExampleData" [allowExport
SelectedData]="true"></dxo-export>

<!-- Selection -->
<dxo-selection mode="multiple"></dxo-selection>
```

Figure 33 : Configuration des modes export et sélection.

- Dxo-selection est ajouté en mode multiple, l'utilisateur peut alors sélectionner une, plusieurs ou toutes les lignes de données de la « Datagrid».
- Dxo-export est activé, nous précisons le nom du fichier exporté et nous activons la possibilité d'exporter les données par ensemble de lignes sélectionnées.

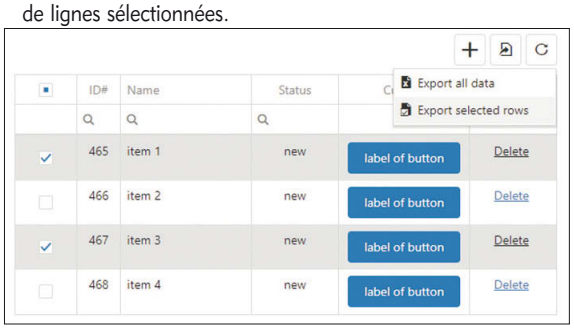


Figure 34 : Aperçu de la Datagrid avec bouton d'action pour l'export de données et les checkbox de sélection multiple.

- Nous utilisons les « checkbox » afin de sélectionner la ligne de données sur laquelle nous souhaitons agir (cette séléction respecte les comportements utilisateurs standards, à savoir la séléction de plusieurs lignes depuis la dernière séléctionnée en un clic au moyen du bouton « shift » associé à celui-ci et bien d'autres).
- Dans la toolbar, l'icône d'exportation est apparue suite à l'ajout de cette fonctionnalité. Le choix d'un export par ligne sélectionnée est possible car l'attribut « allowExportSelectedData » a été passé à vrai.

	A	B	C	D
1	ID#	Name	Status	Custom
2	465	item 1	new	true
3	466	item 2	new	true
4	467	item 3	new	true
5	468	item 4	new	true

Figure 35 : Aperçu du fichier téléchargé des données exportées de la « Datagrid ».

- L'exportation des données retourne un fichier au format .xlsx listant en en-tête les colonnes de la « datagrid », puis la liste de tout les enregistrements, ou de ceux sélectionnés.
  Note : *le résultat de l'export est personnalisable, plus d'info sur le site de l'éditeur* https://js.devexpress.com/Documentation/ApiReference/UI_Widgets/dxDataGrid/Configuration/export/

Personnalisation des éléments et suppression par lot

Maintenant que nous pouvons au travers de notre « Datagrid » chercher, filtrer, ordonner, ajouter, modifier, supprimer, sélectionner et exporter en lot ou entièrement nos données, il serait parfait de pouvoir rajouter notre dernier besoin : la suppression par lot. Nous allons utiliser tout ce que nous avons vu précédemment afin de rendre notre dernière tâche comme l'évidence même. Pour cela, il suffit de rajouter un bouton d'action dans la toolbar et en profiter pour l'organiser, en personnalisant nos boutons « refresh » et « delete » à droite de ma toolbar, et laisser ceux du mode « Editing » à gauche.

```
// DxButton widget with composant configuration in options
 public deleteRowBtnToolbar: DxToolbarComponent["items"] = {
  location: "after",
  widget: "dxButton",
  locateInMenu: "auto",
  options: {
   disabled: !this.deleteBtnValid,
   icon: "trash",
   hint: "Delete Selected Rows",
   onInitialized: (args: any) => {
    // We  recover the instance of DxButton
    this.deleteBtnToolbarInstance = args.component;
   },
   onClick: (e: Event): void => this.deletedSelectedRow(e)
  }
 };
```

Figure 36 : Déclaration de notre bouton de suppression par lot de lignes sélectionnées.

- Nous déclarons un bouton de type items du composant DxToolbarComponent.
- Sa position est définie à « after », soit à droite de la toolbar après les boutons du mode édition, et nous lui passons des options très utiles telles que :
  - disabled : cette option permet de désactiver ce bouton de suppression. Vu qu'il s'agit d'une action par lot, nous décidons de déclarer une variable booléenne deleteBtnValid qui nous informera par data-binding de l'état de sélection de lignes de la datagrid.
  - deleteBtnToolbarInstance : cette option nous permet de récupérer la référence à l'instance de ce widget bouton afin d'interagir avec
- A l'évènement « click » de ce bouton, nous appelons la méthode de suppression **deletedSelectedRow()** qui s'occupe d'envoyer l'ordre de suppression pour chaque enregistrement sélectionné.

```
/**
 * Delected Selected Rows
 */
private deletedSelectedRow(event: Event) {
  let promiseList: Array<any> = [];
  let rows: Array<ExampleData> = this._datagrid.instance.getSelected
RowsData();
   // 1 - for each row selected
  rows.forEach(row => {
   // 2 - we add a promise for every call to our API
   promiseList.push(this.sendRequest(`${this.url}del/${row.id}`, "DELETE",
{ key: row.id }));
  });
  Promise.all(promiseList).then(values => {
   // 3 - we send a notification after the good progress of our actions of
 suppression.
   const type = "success";
   this._notifyConfig.message = "All selected rows deleted!";
   notify(this._notifyConfig, type, 1500);
   // 4 - we deselect the selected rows
   this._datagrid.instance.clearSelection();
  // 5 - we pass the button status to false in order to disable delete button
   this.deleteBtnValid = false;
   if (this.deleteBtnToolbarInstance !== null) {
    this.deleteBtnToolbarInstance.option({
     disabled: !this.deleteBtnValid
    });
   }
   // 6 - we reload the Datagrid
   this._datagrid.instance.getDataSource().reload();
  });
 }
```

Figure 37 : Méthode de suppression des lignes sélectionnées.

La méthode deletedSelectedRow() supprime toutes les lignes qui ont été sélectionnées au sein de la « Datagrid ».

- Pour chaque ligne sélectionnée, récupérée dans l'instance de la datagrid grâce à la méthode getSelectedRowsData() ;
- Nous appelons la méthode sendRequest() avec le paramètre « DELETE » et enregistrons la promesse de cet appel dans un tableau de promesses ;
- Nous utiliserons la méthode all() de l'objet Promise afin d'effectuer un traitement global survenant après la résolution de toutes nos promesses précédemment enregistrées ;
- Une fois la notification envoyée informant du bon déroulement de la suppression de nos données, nous effaçons le tableau de sélection retourné par la datagrid ;
- Nous désactivons le bouton d'action de la toolbar en passant la variable booléenne de son statut à faux ;
- Enfin nous rechargeons la « Datagrid » pour qu'elle mette à jour ses données.

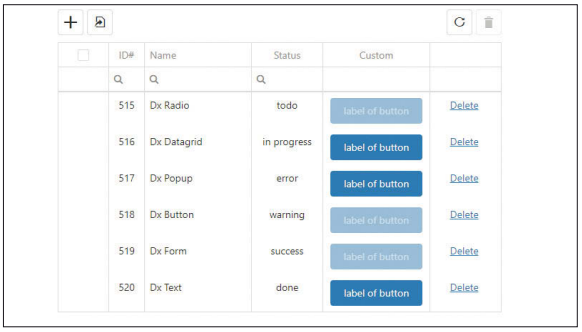


Figure 38 : Aperçu de la « Datagrid » avec personnalisation des boutons de la toolbar.

- Dans la figure 38, les boutons d'actions « add » et « export » du mode édition se retrouvent bien à gauche de la toolbar, et nos boutons « refresh » et « delete » se positionnent bien après « after » à droite. Le bouton de suppression est grisé (ou désactivé car aucune ligne de données n'est sélectionnée).

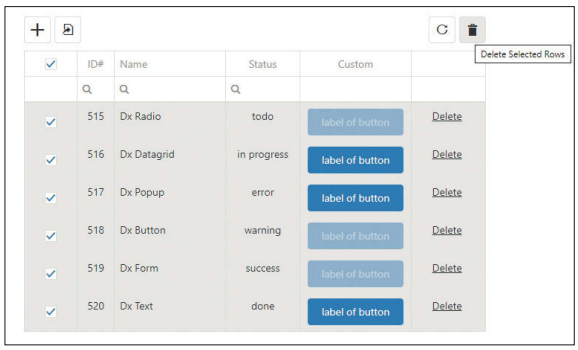


Figure 39 : Aperçu de la "Datagrid" avec toutes ces lignes sélectionnées et le bouton de suppression activé.

Une fois une ou plusieurs lignes selectionnées, le bouton d'action de la toolbar s'active.

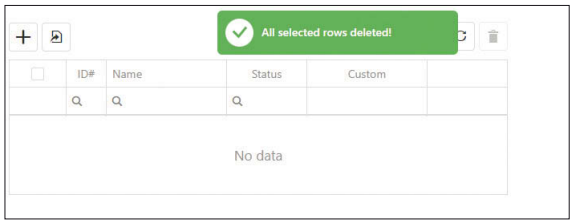


Figure 40 : Aperçu de la "Datagrid" vide une fois ces données supprimées et avec la notification d'action.

Notre « Datagrid » rafraîchit ses données en appelant sa méthode « load » qui requête notre API par la méthode « GET », et nous renvoie un jeu de données actuelles. A noter que des options de pagination, de tri et autres peuvent accompagner les requêtes à notre API, et qu'il est tout à fait possible de requêter par exemple un jeu de 20 enregistrements à partir de la 40e entrée en base de données.
Dans notre cas pratique, toutes les lignes ont bien été supprimées, et une notification nous informe du bon déroulement de cette action. Notre « Datagrid » nous renseigne bien qu'aucune donnée n'ait été remontée, ce qui est le résultat que nous attendions.

Conclusion

Très complète et bien documentée pour un large éventail de technologies, nous venons de parcourir succinctement quelques applications de DevExtreme. C'est une solution facile à prendre en main qui, une fois sa logique assimilée, permet une très grande personnalisation et une large évolutivité sur une multitude de développements, et ce, en un minimum de code. Que ce soit au niveau de ses composants, de sa mise en forme, de ses grilles de données, ou des formulaires et graphiques, en passant par l'éditeur de contenu, **cette suite accroît véritablement la productivité d'un projet**.
Sa licence libre de droit permet de s'amuser à manipuler les composants pour des applications non-commerciales. Quant à sa version payante (nécessaire à des fins commerciales pour 12 mois, et pour chaque développeur du projet), les tarifs réduits pour l'achat de lots et les réabonnements permettent une utilisation illimitée.
Toutefois, à la vue des offres commerciales, de la qualité des composants de cette suite, de la communauté, ainsi que de la documentation qui ne cesse de s'étoffer, **il est vraiment intéressant au sein d'une entreprise de se positionner sur l'acquisition de ce genre de librairie, surtout au sein des projets utilisant des frameworks exploitant le data-binding tel qu'Angular**. •

David MOUTON
Architecte Applicatif chez
Onepoint

Haxe 4
L'isomorphisme cross-plateforme

Partie 2

Le monde du développement connaît depuis quelques années une profonde mutation qui ne semble pas vouloir se terminer. Nous avons pu assister à la mort de certaines technologies, précipitée par leurs éditeurs, comme Adobe avec Flash, Google avec AngularJS, ou encore Apple avec Objective C. D'un autre côté, certains écosystèmes en ébullition voient naître et mourir des frameworks à une vitesse toujours plus grande. Focus sur Haxe 4.

niveau 100

Les nouveautés du compilateur en Haxe 4

• JVM bytecode

Et oui, le compilateur de Haxe est maintenant capable de générer du bytecode pour la JVM. L'intérêt par rapport à javac, d'autant que Java est supporté depuis un moment, c'est qu'il compile beaucoup plus vite (petite pensée à nos amis javaïstes), sans dépendance au JDK, et produit de meilleurs performances au runtime.

• PHP7

Si cela fait longtemps que Haxe supporte PHP7, c'est maintenant la seule version qui est supportée dans cette version 4. En effet, le vénérable et respectable PHP5 a été retiré.

• ES6

Il ne s'agit pas d'une nouvelle target à proprement parler, mais d'un paramètre de compilation pour préciser le standard à utiliser pour la génération du Javascript. C'est devenu utile surtout parce que certaines fonctionnalités HTML5 comme les Web Components, ne sont utilisables qu'en ES6.

• Eval, le nouvel interpréteur

L'interpréteur a été complètement refait pour offrir de bien meilleurs performances comme le montre ce tableau.

what	neko	interp	eval
dox	85.2	257.5	30.5
mandelbrot class	93.6	225.3	29.1
mandelbrot anon	59.7	97.8	24.6
array benchmark	33.4	202.1	19.4

• Hashlink

Hashlink est la nouvelle VM cross-plateforme et haute performance de Haxe. Nous allons y revenir.

• Unicode

Il est supporté sur toutes les targets :

```
"λ".length == 1
```

Hashlink

C'est la nouvelle machine virtuelle de Haxe, successeur de NekoVM. Elle a deux modes de fonctionnements. Le premier est basée sur un JIT, destiné à une utilisation quotidienne car elle est très rapide à compiler. La seconde est une compilation vers du C pour des performances optimales. Pour vous donner une idée de la rapidité de la compilation, celle d'un jeu comme Northgard c'est 5,8 secondes avec un vieux Q6600. Coté performances du runtime, Hashlink surpasse Adobe AIR et Node/V8, et ce n'est qu'un début. Pour compiler vers HL/JIT il suffit d'utiliser la target -hl est de spécifier une sortie en .hl

```
haxe -hl output.hl -main MyApp
```

On peut ensuite démarrer Hashlink ainsi :

```
hl output.hl
```

Cibler HL/C et compiler avec GCC ressemble à ça :

```
haxe -hl out/main.c -main MyApp
gcc -03 -o myapp -std=c11 -I out out/main.c -Lhl [/path/to/required.hdll]
```

Le runtime de HL supporte la Std de Haxe mais pas seulement. On retrouve la fameuse lib SDL pour tout ce qui est OpenGL 3.2+ API, Windows creation, game controllers, etc.
Il y a une UI Library assez basique à laquelle on préférera celle de du framework Heaps.io, DomKit pour concevoir des IHM plus complexes. Enfin FMT pour tout ce qui est ZLIB compression, JPG decoding, image scaling et resampling.

Ma première webapp

Il est temps de s'essayer à Haxe. Pourquoi ne pas commencer par une application cliente en JS et un backend en PHP ? L'idée est d'illustrer ses capacités en matière d'isomorphisme en partageant un même modèle entre le client et le serveur.

Prérequis

Nous allons avoir besoin d'un serveur web en local, un apache par exemple, avec l'extension mbstring activée. Pour coder cette application nous allons utiliser Intellij IDEA. Notez que n'importe quelle distribution est aussi valable, comme WebStorm ou PHPStorm, il faudra juste y ajouter le plugin Haxe.

Installation

Il existe de nombreuses façon d'installer Haxe. Sous Linux on peut utiliser le gestionnaire de paquets. Avec une Ubuntu ça donne ça :

```
sudo add-apt-repository ppa:haxe/releases -y
sudo apt-get update
sudo apt-get install haxe -y
mkdir ~/haxelib && haxelib setup ~/haxelib
```

Cependant, Haxe 4 étant encore en RC il faudra passer par une ins-

tallation manuelle des binaires. Sous Windows il faut télécharger le .msi et le .pkg pour macOS, depuis haxe.org
Il existe encore d'autres façon d'installer Haxe, comme utiliser NPM ou Chocolatey. Pour vérifier que Haxe est correctement installé, ouvrez un terminal et tapez :

```
haxe -version && haxelib version
```

La commande « haxe » invoque le compilateur quand « haxelib » appelle le gestionnaire de paquets officiel. Dans le monde Java Script, ça serait les équivalents des commandes « node » et « npm ».

Architecture

Nous allons commencer par créer la structure du projet que nous allons appeler « demohx ». Créons un dossier demohx ainsi que 2 sous dossiers « src » et « www ». Le dossier src va contenir les sources Haxe de notre projet, alors que le dossier www va contenir le résultat de la compilation. On va ensuite ajouter 2 fichiers vides, build.hxml à la racine du projet et index.html à la racine de www.
Nous devrions avoir une arborescence qui ressemble à ça :

```
- demohx
   build.hxml
   → src
   → www
      index.html
```

Configurez votre serveur web pour qu'il affiche notre index.html via une url du type http://localhost/index.html

Configuration

Il est temps de lancer notre IDE, Intellij IDEA. On commence un nouveau projet de type Haxe en choisissant le SDK (notre version de haxe) que nous avons installé. Comme c'est notre première installation, nous allons devoir le configurer. La procédure est très simple et demande seulement de sélectionner l'emplacement de Haxe sur notre file system. Ensuite on fait pointer notre projet sur le dossier « demohx ». A la création du projet Intellij va créer une classe Main.hx à la racine de src. C'est gentil mais nous n'en avons pas besoin ici. Nous allons avoir besoin de compiler 2 classes « Main », celle du client JavaScript et celle du serveur PHP. On va donc la supprimer et créer 2 packages « front » et « back » dans le dossier src.
Ajoutons la classe Client.hx

```
package front;

class Client {

  public static function main(){
    trace('front');
  }

}
```

Comme il s'agit de notre classe principale pour notre client JS, elle doit avoir une fonction main(). Ensuite on fait la même chose dans le package back, avec une classe Server.hx. Maintenant nous allons configurer la compilation du projet dans build.hxml

```
-cp src

--each

-main front.Client
-js www/js/client.js

--next

-main back.Server
-php www/api
```

Ici on indique au début que toutes les compilations vont partager le même dossier de source. Ensuite nous compilons front.Client en Javascript dans www/js/client.js. Enfin, nous compilons back.Server en PHP dans www/api. A ce stade nous pouvons tester la compilation en ligne de commande depuis la racine du projet via :

```
haxe build.hxml
```

Normalement Intellij a détecté la présence du fichier build.hxml et s'est correctement configuré. Vous devriez pouvoir faire un build du projet (Maj+F9). Il ne nous reste qu'a modifier le fichier index.html pour charger notre javascript.

```
<script src="js/client.js"></script>
```

Maintenant on peut tester http://localhost et voir un message dans la console. nhttp://localhost/api devrait afficher aussi un message.

Le code

Les messages en console c'est quand même moins pratique que dans le DOM. On va ajouter une DIV dans notre index.html comme ceci :

```
<div id="out"></div>
```

Et on va remplacer notre « trace » par :

```
var outDiv = Browser.document.getElementById("out");
outDiv.innerHTML = "front";
```

Normalement à la complétion, Intellij vous a automatiquement ajouté l'import de Browser. Maintenant on aller plus loin en affichant ce que dit le serveur. Pour ça on va utiliser fetch sur l'url de notre api.

```
  public static function main(){

    var outDiv = Browser.document.getElementById("out");

    Browser.window.fetch("http://localhost/demohx/api")
    .then(response -> response.text())
    .then(message -> outDiv.innerHTML = message);
  }
```

C'est le moment de soigner les données échangées. Nous allons donc créer un nouveau package « model » et y ajouter une nouvelle classe « User » package model;

```
class User {
  public var name:String;
  public var firstname:String;

  public function new(name:String, firstname:String) {
    this.name = name;
```

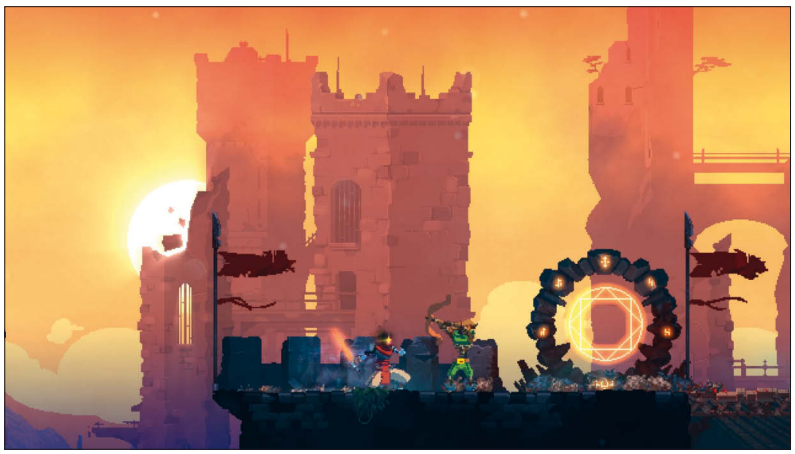
```
    this.firstname = firstname;
  }

}
```

Modifions notre API pour qu'elle nous retourne un user de cette façon :

```
public static function main(){
    var user = new User("Bombadil","Tom"); // le user qu'on va retourner
    Web.setReturnCode(HttpStatus.OK); // Le code http 200
    Web.setHeader('Content-type',Mime.ApplicationJson); // le content-type pour
indiquer qu'on retourne du json
    Lib.print(Json.stringify(user));
  }
```

Nous pouvons maintenant afficher côté client notre user.

```
public static function main(){

    var outDiv = Browser.document.getElementById("out");

    Browser.window.fetch("http://localhost/demohx/api")
    .then(response -> response.json())
    .then(function(user:User){
      outDiv.innerHTML = 'Hello ' +user.firstname + ' ' + user.name;
    });
  }
```



HAXE EN 2019

Haxe vient du monde du jeu vidéo, il n'est donc pas surprenant de voir cette technologie s'y épanouir avec des success story impressionnantes. Le jeu Northgard du studio indépendant Shirogame a ainsi dépassé le million sur Steam. MotionTwin fait encore mieux avec plus de 2 millions de copies pour Dead Cells, sur Mac, PC, Linux, PS4, XBox, et Nintendo Switch. Haxe n'est pas réservé aux studios indépendants et trouve sa place également chez les grands éditeurs. InnoGames a choisi ce langage pour Forge of Empire, qui avec ses 200 millions de joueurs en fait un des jeux en ligne les plus populaires au monde. Les domaines dans lesquels s'illustre Haxe sont bien plus larges que le seul jeu vidéo. De l'autre côté de l'atlantique, TIVo commercialise une box TV vendue à 10 millions d'exemplaires avec des milliers de lignes de code en Haxe au coeur de son IHM. Dans le Web, les équipes de Docler utilisent cette technologie sur un site qui fait 35 millions de visites par jour !
Et si on regarde du côté du logiciel, on ne peut pas ne pas citer Armory 3D, un clone Open Source de Unity, basé sur Blender.

Conclusion

Ce petit exemple montre à quel point Haxe se révèle efficace dans les approches de type Contract-First, le Domain Driven Design, et les architectures hexagonales. Un seul modèle partagé entre frontend et backend. Et Haxe peut aller encore beaucoup plus loin grâce à la compilation conditionnelle. Dans notre exemple nous avons ciblé un backend en PHP mais nous aurions pu isoler la partie métier de la partie système et supporter plusieurs backends.

Pour bien commencer

Intellij Idea et Visual Studio Code offrent un très bon niveau de support pour les développements basés sur Haxe. Voici une liste de quelques sites bien utiles :

try.haxe.org : pour essayer du code Haxe directement depuis son navigateur,
community.haxe.org : pour échanger et trouver de l'aide auprès d'autres développeurs,
code.haxe.org : des exemples et tutoriels de tous niveaux.

Anthony Giretti
MVP, MCSD
Developer, Blogger, Speaker
anthony.giretti@gmail.com
http://anthonygiretti.com

equisoft
BUSINESS-DRIVEN INNOVATION

Microsoft® MVP Most Valuable Professional

Apprendre GIT en lignes de commandes dans Visual Studio 2019 et GitHub

Partie 1

niveau 100

Lorsque l'on travaille à plusieurs sur un même projet, on est vite confronté à un problème majeur : comment sauvegarder le travail de chacun ? Surtout lorsque des fichiers en commun sont modifiés. Afin de répondre à cette problématique, des outils de gestion de versions, tel que Git, ont été créés.

Mais que permet de faire GIT exactement ? Finalement, beaucoup de choses :

- Créer des dépôts (repository) : Il est possible d'initier un repository localement sur sa machine si le repository n'existe pas déjà sur le serveur de contrôle de source ou le récupérer depuis ce même serveur si le repository existe déjà.
- Effectuer des changements sur les fichiers (ajout, suppression, modification et renommage de ces derniers).
- Créer des branches de travail à partir d'une autre branche.
- Regrouper des changements provenant de diverses branches de travail (copies de travail).
- Mettre en suspend des modifications sans les perdre.
- Vérifier l'historique des versions.
- Synchroniser les changements.
- Exclure des fichiers pour le suivi de version.
- Et bien d'autres fonctionnalités que nous verrons dans la partie 2 (avancé).

Installation des outils pour utiliser GIT en ligne de commande

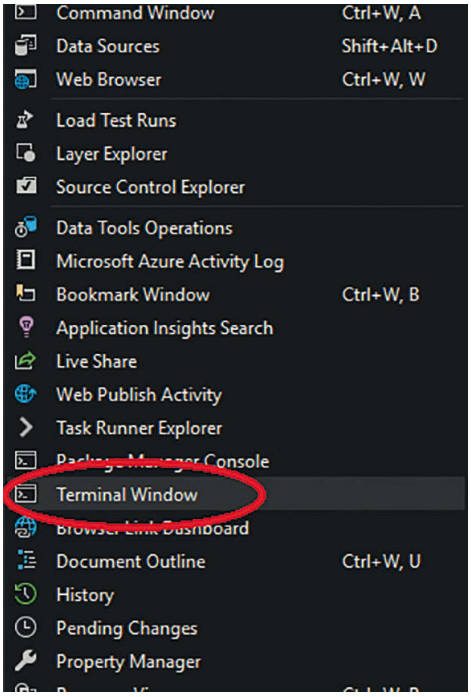
Tout d'abord, il est bon de rappeler que GIT est déjà installé avec Visual Studio 2019. En effet, VS 2019 (tout comme VS 2017) supporte GIT mais à travers l'interface VS.

Il est également possible d'utiliser GIT en ligne de commandes avec le Package Manager Console (qui est finalement une invite de commande PowerShell) mais je ne vais pas utiliser cet outil.

Vous pouvez cependant, pour profiter des dernières nouveautés, installer une mise à jour de GIT. Visual Studio installe la version 2.17 pour Windows, tandis que la dernière version sortie est la 2.23, elle apporte par exemple un alias de *git checkout* avec la commande *git switch*

Je vous propose une alternative assez sympathique (colorée également). A l'heure où j'écris ces lignes le Windows terminal de Visual Studio 2019 n'est pas encore sorti, c'est pourquoi je vais vous proposer d'installer l'extension pour Visual Studio (compatible 2017 et 2019) « Whack Whack » disponible en téléchargement ici : https://marketplace.visualstudio.com/items?itemName=DanielGriffen.WhackWhackTerminal

Une fois installé vous devez activer le Windows Terminal dans le menu *View -> Other Windows*

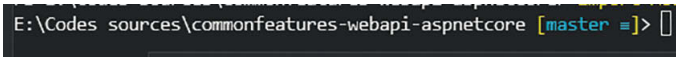


Je recommande également une autre extension nommée posh-git, permettant d'indiquer à tout moment les modification courantes (fichiers modifiés, conflits, etc.) à coté du nom de la branche sur laquelle vous travaillez. Il est disponible en téléchargement ici : https://git-scm.com/book/en/v2/Appendix-A%3A-Git-in-Other-Environments-Git-in-PowerShell

Si vous obtenez un message d'erreur indiquant que l'installation de **posh-git** est bloquée car le script PowerShell n'est pas signé, veuillez exécuter cette commande qui lèvera les restrictions d'exécution de scripts PowerShell :

```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
```

Si vous êtes déjà dans un repository GIT vous devriez obtenir quelque chose comme ça :



Configuration des outils

Nous allons commencer par configurer les informations d'utilisateur pour tous les dépôts GIT locaux.
C'est nécessaire pour identifier l'auteur des changements sur des fichiers, mais aussi pour pousser vos modifications / ajouts / suppression sur le serveur.
Cette étape n'est pas nécessaire si vous avez déjà utilisé GIT en clonant un dépôt distant (setting automatique)
Nous avons deux informations essentielles à configurer : le nom d'utilisateur et l'email. Voici comment procéder :

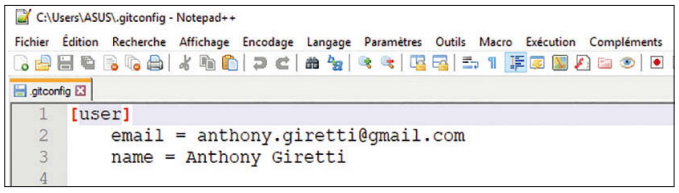
• Le nom d'utilisateur :

```
git config --global user.name "Anthony Giretti"
```

• L'email :

```
git config --global user.email anthony.giretti@gmail.com
```

Une fois ces informations de configurations saisies vous pouvez vérifier le contenu de votre fichier **.gitconfig** dans le répertoire « C:\Users », dans mon cas le fichier se trouvait dans ce répertoire : « C:\Users\ASUS »

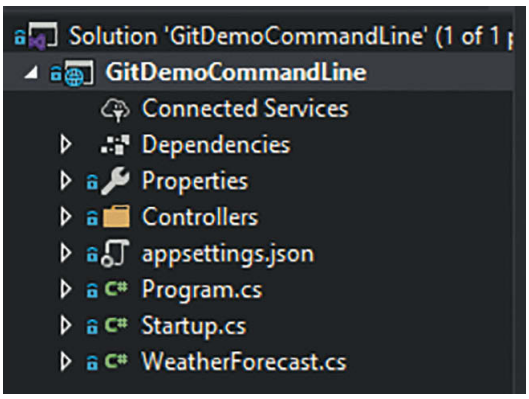


Il vous est possible également d'activer la coloration de la sortie en ligne de commande ainsi :

```
git config --global color.ui auto
```

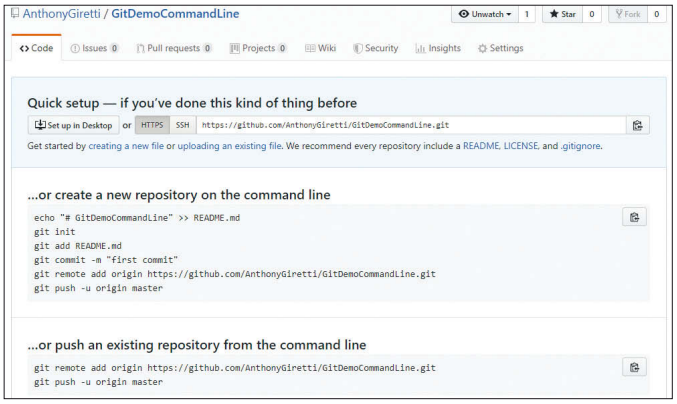
Création d'un dépôt

Pour illustrer mes exemples, je vais utiliser comme dépôt distant GITHUB. Si vous le souhaitez, vous pouvez vous créer un compte ici : https://github.com/
Nous allons dans cet exemple créer le squelette d'une WebAPI Asp.Net Core 3



Il y a 2 façons de créer un nouveau dépôt. S'il existe déjà sur le serveur distant, il va falloir le **cloner,** sinon le **créer localement et le pousser** sur le serveur.
Pour créer localement un dépôt et le pousser sur le serveur :

• Il faut d'abord, avec GitHub, créer le dépôt sur le serveur distant (même url que pour la création du compte).
• Il faut ensuite suivre les étapes mentionnées sur GitHub
• Puis, après avoir créé le dépôt, vous aurez les instructions qui suivent :



Ensuite retournez dans Visual Studio dans votre fenêtre de commande :

```
git init GitDemoCommandLine
```

Création d'un répertoire GitDemoCommandLine contrôlé par GIT.

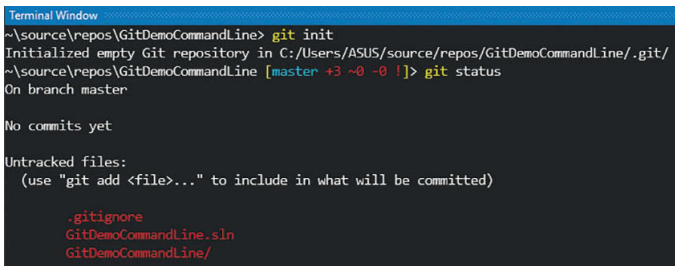
Si le dossier existe déjà et que vous êtes positionné dedans : *git init* suffira. Etant donné que nous sommes dans un contexte .NET dans Visual Studio, je vous suggère de télécharger le fichier gitignore ici : https://github.com/github/gitignore/blob/master/VisualStudio.gitignore.
Ce fichier permet tout simplement de signifier à GIT que l'on souhaite ignorer certains types de fichier dans notre contrôle de source.
Puis créez votre solution .NET, toujours dans ce même répertoire.

```
git add . ou git add –all
git commit -m « Initialisation »
git remote add origin https://github.com/AnthonyGiretti/GitDemoCommandLine.git
git push -u origin master
```

Qu'avons-nous fait?

Nous avons créé un répertoire contrôlé par GIT, ajouté un fichier permettant d'ignorer les fichiers que nous ne souhaitons pas versionner, nous avons ensuite ajouté notre solution .NET dans l'index de l'arbre de travail avec la commande *add*, enregistré le tout de manière permanente dans l'historique de version avec la commande *commit* et enfin envoyé au serveur avec la commande *push*. Par convention la branche par défaut initialement créée est nommée la branche **master.**
Notez que la commande *git add remote* permet de préciser sur quel dépôt distant nous allons nous synchroniser.

Note : l'url de mon dépôt distant est prédéterminée par GitHub, et elle se compose ainsi :
https://github.com/{nomdecompteGitHub}/{nomdudépot}
Voici en image la reproduction des étapes précédentes :



Notez que vous pouvez utiliser la commande *status* pour connaître

le statut des fichiers de votre dépôt local. Ici GIT détecte qu'un fichier **.sln** , un fichier **.gitignore** et un répertoire **GitDemoCommandLine** sont présents dans le répertoire mais ils ne sont pas encore trackés.

```
Terminal Window
~\source\repos\GitDemoCommandLine [master +3 ~0 -0 !]> git add .
~\source\repos\GitDemoCommandLine [master +9 ~0 -0 ~]> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   .gitignore
        new file:   GitDemoCommandLine.sln
        new file:   GitDemoCommandLine/Controllers/WeatherForecastController.cs
        new file:   GitDemoCommandLine/GitDemoCommandLine.csproj
        new file:   GitDemoCommandLine/Program.cs
        new file:   GitDemoCommandLine/Startup.cs
        new file:   GitDemoCommandLine/WeatherForecast.cs
        new file:   GitDemoCommandLine/appsettings.Development.json
        new file:   GitDemoCommandLine/appsettings.json
```

A cette étape, après l'ajout de tous nos fichiers dans le suivi de version que GIT détecte désormais (en vert), nous pouvons voir nos fichiers « éligibles » à l'enregistrement des fichiers de façon permanente dans l'historique des versions.

```
Terminal Window
~\source\repos\GitDemoCommandLine [master +9 ~0 -0 ~]> git commit -m  "Initialisation"
[master (root-commit) d0be3df] Initialisation
 9 files changed, 515 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 GitDemoCommandLine.sln
 create mode 100644 GitDemoCommandLine/Controllers/WeatherForecastController.cs
 create mode 100644 GitDemoCommandLine/GitDemoCommandLine.csproj
 create mode 100644 GitDemoCommandLine/Program.cs
 create mode 100644 GitDemoCommandLine/Startup.cs
 create mode 100644 GitDemoCommandLine/WeatherForecast.cs
 create mode 100644 GitDemoCommandLine/appsettings.Development.json
 create mode 100644 GitDemoCommandLine/appsettings.json
```

```
~\source\repos\GitDemoCommandLine [master]> git remote add origin https://github.com/AnthonyGiretti/GitDemoCommandLine.git
~\source\repos\GitDemoCommandLine [master]> git push -u origin master
Counting objects: 13, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (13/13), done.
Writing objects: 100% (13/13), 5.33 KiB | 1.07 MiB/s, done.
Total 13 (delta 0), reused 0 (delta 0)
To https://github.com/AnthonyGiretti/GitDemoCommandLine.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
~\source\repos\GitDemoCommandLine [master ≡]>
```

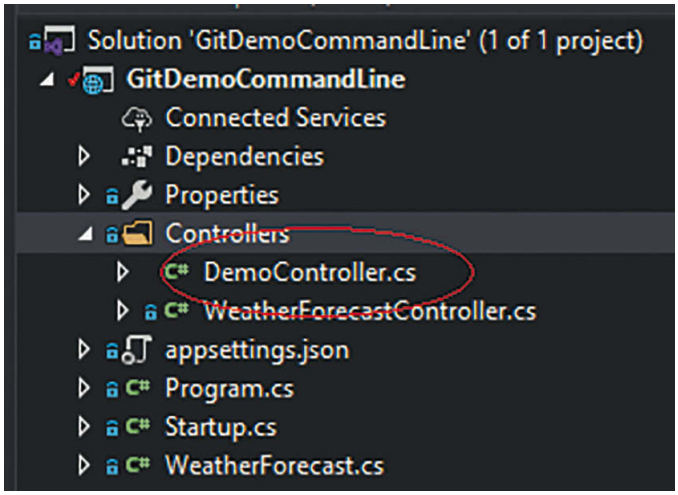
Cloner un dépôt existant déjà sur le serveur distant : la commande est ici très simple, positionnez-vous dans un répertoire que vous aurez créé pour l'opération qui vient, puis exécutez la commande suivante :

```
git clone https://github.com/AnthonyGiretti/GitDemoCommandLine.git
```

```
~\source\repos> git clone https://github.com/AnthonyGiretti/GitDemoCommandLine.git
Cloning into 'GitDemoCommandLine'...
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 13 (delta 0), reused 13 (delta 0), pack-reused 0
Unpacking objects: 100% (13/13), done.
~\source\repos> cd GitDemoCommandLine
```

Effectuer des changements et les synchroniser

Dans le scénario suivant, nous allons créer un contrôleur de WebAPI puis le synchroniser avec GitHub.
Pour ce faire créons un contrôleur dans Visual Studio et appelons le **DemoController** :



Ajoutons maintenant notre contrôleur dans l'index de kl'arbre de travail avec la commande *git add filename*.
Il est possible d'ajouter plusieurs fichiers à la fois en séparant les noms par un espace, ou si vous souhaitez tous les ajouter en même temps, vous pouvez ajouter l'option *–all* a la commande.

```
git add filename1 filename2
git add –all
```

```
~\source\repos\GitDemoCommandLine [master =]> git add GitDemoCommandLine\Controllers\DemoController.cs
~\source\repos\GitDemoCommandLine [master ≡ +1 ~0 -0 | +0 ~1 -0 !]>
```

Je n'ai pas parlé de l'importance de **posh-git** jusqu'ici. Il s'avère vraiment pratique pour s'apercevoir de ce qui se passe.
Comme vous le savez quand vous ajoutez un nouveau fichier à votre solution, votre fichier **.csproj** est également modifié. **Posh-git** va vous indiquer qu'il a été modifié mais qu'il n'a pas été ajouté à l'index. Concernant sa modification, vous allez donc encore une fois saisir la commande git status pour voir le détail de la situation :

```
~\source\repos\GitDemoCommandLine [master ≡ +1 ~0 -0 | +0 ~1 -0 !]> git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   GitDemoCommandLine/Controllers/DemoController.cs

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   GitDemoCommandLine/GitDemoCommandLine.csproj

~\source\repos\GitDemoCommandLine [master ≡ +1 ~0 -0 | +0 ~1 -0 !]>
```

GIT va tout simplement vous confirmer que dans ce cas précis, j'ai oublié d'ajouter le fichier **GitDemoCommandLine/GitDemoCommandLine.csproj** au suivi de version !
Ajoutons donc notre fichier **GitDemoCommandLine/GitDemoCommandLine.csproj** à l'index :

```
~\source\repos\GitDemoCommandLine [master ≡ +1 ~0 -0 | +0 ~1 -0 !]> git add GitDemoCommandLine/GitDemoCommandLine.csproj
~\source\repos\GitDemoCommandLine [master ≡ +1 ~1 -0 ~]>
```

Cette fois-ci GIT nous indique que nous avons un fichier ajouté et un fichier modifié dans l'index en attente d'ajout à l'historique de version, pour ce faire nous allons utiliser une commande que nous avons déjà utilisée :

git commit -m « message »

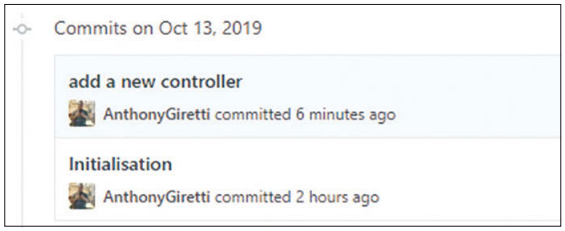
l'option *-m* permet simplement d'ajouter un message au commit, un commit étant **un instantané de fichiers dans l'historique des versions**

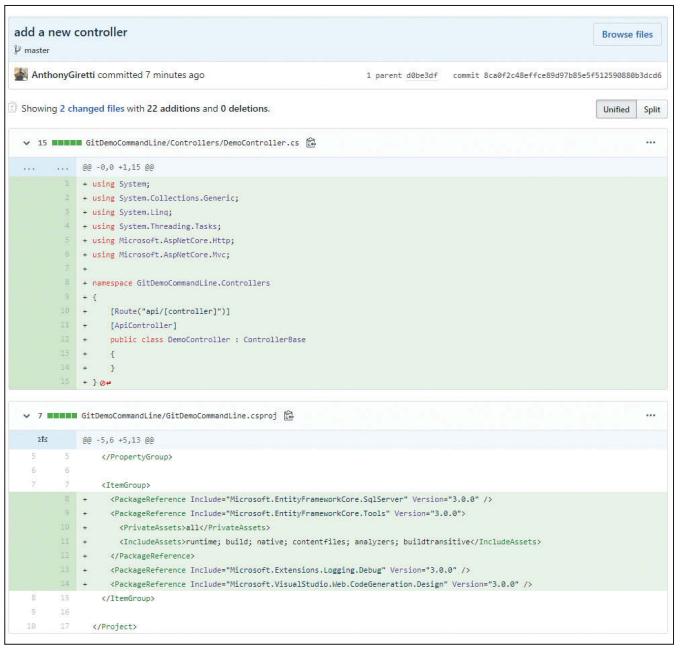
```
~\source\repos\GitDemoCommandLine [master ≡ +1 ~1 -0 ~]> git commit -m "add a new controller"
[master 8ca0f2c] add a new controller
 2 files changed, 22 insertions(+)
 create mode 100644 GitDemoCommandLine/Controllers/DemoController.cs
~\source\repos\GitDemoCommandLine [master ↑1]>
```

posh-git nous indique ensuite que nous pouvons synchroniser le commit (ici 1, il est bien sûr possible de synchroniser plusieurs commits dans la même commande *push*).
Une fois synchronisé avec la commande *git push*, le serveur renvoie un statut de la synchronisation :

```
~\source\repos\GitDemoCommandLine [master ↑1]> git push
Counting objects: 6, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 958 bytes | 958.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/AnthonyGiretti/GitDemoCommandLine.git
   d0be3df..8ca0f2c  master -> master
~\source\repos\GitDemoCommandLine [master ≡]>
```

Regardons désormais sur GitHub l'historique de version :





Il est évidemment possible de faire des erreurs. Par exemple, vous avez créé un fichier, mais votre code n'est pas correct. Vous voulez le modifier avant de l'ajouter au suivi de version, mais hélas vous l'avez ajouté trop vite à l'index avec *git add*… Comment faire pour éviter de l'ajouter à l'historique avec le code défectueux qu'il contient ? Ceci sans pour autant le supprimer physiquement ? C'est possible avec les commandes suivantes :

git reset filename ou git rm –cached filename

```
~\source\repos\GitDemoCommandLine [master ≡ +1 ~0 -0 ~]> git reset GitDemoCommandLine/notwantedfile.cs
~\source\repos\GitDemoCommandLine [master ≡ +1 ~0 -0 !]> git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        GitDemoCommandLine/notwantedfile.cs

nothing added to commit but untracked files present (use "git add" to track)
~\source\repos\GitDemoCommandLine [master ≡ +1 ~0 -0 !]>
```

```
~\source\repos\GitDemoCommandLine [master ≡ +1 ~0 -0 ~]> git rm --cached GitDemoCommandLine/unwantedfile.cs
rm 'GitDemoCommandLine/unwantedfile.cs'
~\source\repos\GitDemoCommandLine [master ≡ +1 ~0 -0 !]> git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        GitDemoCommandLine/unwantedfile.cs

nothing added to commit but untracked files present (use "git add" to track)
```

Vous pouvez maintenant corriger votre fichier puis le renvoyer dans l'index à nouveau ! Finalement vous vous êtes encore trompé. Vous n'avez plus besoin de ce fichier et vous désirez le supprimer complètement alors qu'il a déjà été ajouté à l'index. Pour cela utilisez la commande suivante :

git rm -f filename

L'option *-f* permet de supprimer le fichier physiquement même s'il a été ajouté à l'index.

```
~\source\repos\GitDemoCommandLine [master ≡ +1 ~0 -0 ~]> git rm GitDemoCommandLine/notwantedfile.cs
error: the following file has changes staged in the index:
    GitDemoCommandLine/notwantedfile.cs
(use --cached to keep the file, or -f to force removal)
~\source\repos\GitDemoCommandLine [master ≡ +1 ~0 -0 ~]> git rm -f GitDemoCommandLine/notwantedfile.cs
rm 'GitDemoCommandLine/notwantedfile.cs'
~\source\repos\GitDemoCommandLine [master ≡]>
```

Si vous avez déjà ajouté votre fichier dans le suivi de version avec un commit, il n'est pas non plus trop tard grâce à cette commande pour le supprimer du suivi de version et physiquement :
git rm -f filename puis *git commit* et *git push* pour synchroniser le tout.
Comme vous pouvez le constater, nous avons généré un commit pour l'ajout et un second pour la suppression :

```
~\source\repos\GitDemoCommandLine [master ≡]> git add --all
~\source\repos\GitDemoCommandLine [master ≡ +1 ~0 -0 ~]> git commit -m "add unwanted file"
[master 9b80e6e] add unwanted file
 1 file changed, 11 insertions(+)
 create mode 100644 GitDemoCommandLine/unwantedfile.cs
~\source\repos\GitDemoCommandLine [master ↑1]> git rm -f GitDemoCommandLine/unwantedfile.cs
rm 'GitDemoCommandLine/unwantedfile.cs'
~\source\repos\GitDemoCommandLine [master ↑1 +0 ~0 -1 ~]> git commit -m "remove unwanted file"
[master b1285cd] remove unwanted file
 1 file changed, 11 deletions(-)
 delete mode 100644 GitDemoCommandLine/unwantedfile.cs
~\source\repos\GitDemoCommandLine [master ↑2]> git push
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 632 bytes | 632.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/AnthonyGiretti/GitDemoCommandLine.git
   80168ec..b1285cd  master -> master
~\source\repos\GitDemoCommandLine [master ≡]>
```

Conclusion

Nous venons de voir comment créer des dépôts GIT et comment modifier / ajouter / supprimer des fichiers en lignes de commandes avec des outils pratiques, le tout sans se préoccuper du GUI Visual Studio.
Dans cette partie 1, nous avons seulement vu les commandes de base. Nous verrons dans une partie 2, comment créer des branches, comment fusionner nos branches et comment sauvegarder notre travail en cours sans le perdre. Nous verrons également la notion de Flux avec GIT Flow et comment interroger l'historique de notre dépôt. Enfin nous verrons en quoi il est parfois important de repartir sur une nouvelle base de travail, ou revenir à un commit bien précis, ou encore prendre un commit bien précis sur le serveur. •

Sylvain Saurel
sylvain.saurel@gmail.com
Développeur Java / Android
https://www.ssaurel.com

Les 5 plus grands mythes entourant Java enfin démystifiés

niveau 100

Nous sommes en 2019 et bien qu'il ait souvent été donné pour moribond, voire qualifié de nouveau COBOL par certains, le langage Java reste le leader incontesté des langages de programmation en entreprise. 1

Créé en 1995 par James Gosling et Patrick Naughton, Java a connu un succès fantastique qui perdure. L'acquisition par Oracle en 2009 aura suscité des doutes sur son avenir, mais les récentes décisions de la société de Larry Ellison de raccourcir les délais entre chaque version majeure de Java sont un pas dans la bonne direction puisqu'ils vont permettre d'offrir de nouvelles fonctionnalités à un rythme plus soutenu.

En outre, les dernières versions de Java ont permis des avancées significatives pour le langage et la plateforme, avec l'introduction des Lambdas et des Streams dans Java 8 ou la modularisation du JDK avec le projet Jigsaw tant attendu qui aura finalement été lancé avec Java 9. Alors que la version 13 du JDK vient tout juste de sortir, on peut dire que Java a fait d'énormes progrès pour se maintenir au niveau de ses concurrents avec des syntaxes souvent citées comme plus modernes.

Malgré tous les progrès réalisés, un certain nombre de mythes persistants subsistent autour de Java. Je vous propose donc de démystifier les 5 plus importants.

1 Java est mort

De 1995 à 2006, le langage Java a évolué à un rythme rapide avec une nouvelle version majeure tous les deux ans. L'introduction des Generics dans Java 5 a constitué une véritable révolution en 2004. Publiée en 2006 avec des modifications mineures, la version 6 du JDK sera la dernière avant de nombreuses années du fait de divergences au sein de la communauté et surtout de l'acquisition de Sun par Oracle en 2009. 2

Les doutes entourant l'avenir de Java sont nombreux et certains parient sur sa mort sous la direction d'Oracle. Si Java 7 sort en 2011, c'est avec des nouveautés limitées. Ainsi, le projet Jigsaw a été reporté devant l'absence de consensus au sein de la communauté. Pour beaucoup, Java est en route pour devenir le nouveau COBOL. Il faut encore attendre 3 ans pour que Java 8 ne sorte. Les Lambdas et les Streams sont une véritable révolution ouvrant la voie à la programmation fonctionnelle en Java. Une bouffée d'oxygène pour un langage qui commençait à être critiqué de toute part du fait de sa syntaxe trop lourde.

Néanmoins, l'absence du projet de modularisation du JDK dans Java 8 déçoit une nouvelle fois. Fort heureusement, ce n'est que partie remise puisque le projet Jigsaw est finalement intégré à Java avec la version 9 publiée en septembre 2017.

Conscient de ces problèmes, Oracle a décidé de changer son fusil d'épaule à partir de Java 10 publié en mars 2018. Désormais, Oracle publiera une nouvelle version majeure du JDK tous les 6 mois. L'objectif est de publier de nouvelles fonctionnalités à un rythme plus soutenu afin d'éviter que les développeurs ne considèrent Java comme un langage mort.

C'est finalement un succès, car 18 mois après la mise en œuvre de cette nouvelle politique, Oracle vient déjà de publier Java 13 en septembre 2019. Cette rapide démonstration prouve que Java est loin d'être un langage mort et que les comparaisons le qualifiant de nouveau COBOL sont loin d'être justifiées.

3 *Java est-il lent ?*

1

Les 5 plus grands mythes de Java enfin démystifiés

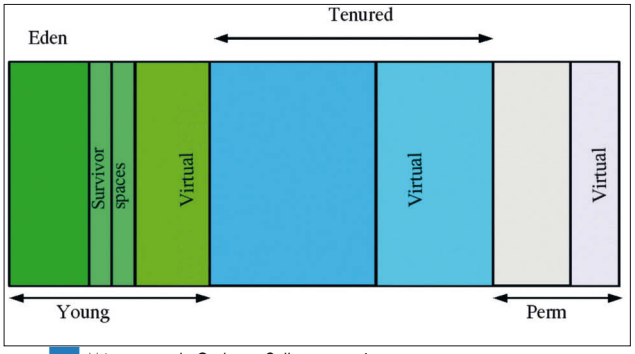
2 *Beaucoup ont pensé que Java allait mourir suite à l'acquisition par Oracle*

2 Java est lent

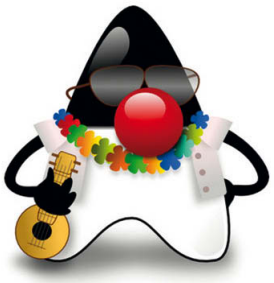
La prétendue lenteur de Java est le mythe suivant. Je dois avouer que ce mythe repose sur certaines vérités puisque les premières versions de Java étaient en effet plus lentes que des langages compilés tels que C ou C++. Afin de tenir sa promesse "Write Once, Run anywhere", Java a dû faire des concessions et sa machine virtuelle rajoutait une couche logicielle ce qui nuisait aux performances dans certains cas spécifiques. 3

Outre ce problème de couche supplémentaire, certaines implémentations de la JVM étaient très lentes au tout début. Néanmoins, ce mythe a vécu, et les aficionados du C/C++ doivent faire face à la réalité : la JVM est maintenant ultra rapide. De plus, le matériel a tellement évolué depuis les premières versions de Java que cette surcouche logicielle n'a plus d'incidence sur les performances de la grande majorité des applications d'entreprise.

4 *Java est-il devenu un service payant ?*



5 *Mécanisme de Garbage Collection en Java*



6 *Java est-il le seul langage que vous devez apprendre ?*

Enfin, le compilateur Just-In-Time (JIT) est une pure merveille qui améliore nettement les performances des applications Java. Le niveau de performance de la JVM est aujourd'hui tel qu'il constitue une plateforme d'exécution pour un grand nombre de langages tels que Kotlin, Groovy ou encore Scala. On parle désormais bien d'un écosystème complet autour de Java.

La prochaine fois que quelqu'un vous dira que Java est lent, répondez-lui simplement que ce mythe a vécu et que les performances des applications Java sont désormais comparables à celles de programmes C/C++, dans la majorité des cas.

3 Java est devenu payant

En 2018, Oracle a annoncé de manière déroutante que le JDK serait désormais facturé pour des utilisations en production. De toute évidence, cela a engendré de la confusion chez certains qui ont commencé à affirmer que Java était devenu payant ! Cependant, ce n'est pas le cas. 4

En réalité, Oracle distribue maintenant deux versions du JDK :

- JDK Oracle
- Oracle OpenJDK

Le JDK d'Oracle est gratuit en environnement de développement et de test, mais il devient payant en production et s'accompagne d'un support long terme d'Oracle. De l'autre côté, nous avons Oracle OpenJDK qui reste gratuit pour tous les environnements.

Ainsi, Java reste libre si vous optez pour Oracle OpenJDK. En outre, il existe d'autres implémentations du JDK, ce qui garantit que Java restera libre. Vous pouvez utiliser le JDK proposé par la communauté AdoptOpenJDK. Plus récemment, Amazon vient de mettre à disposition ses propres versions du JDK sous le nom Amazon Corretto avec une assistance à long terme entièrement gratuite.

En bref, Java reste libre et l'idée selon laquelle Java serait désormais payant est fausse.

4 Le développeur Java n'a pas à se soucier de la mémoire

En Java, la gestion mémoire est déléguée à un Garbage Collector. Ainsi, contrairement à C ou C++, le développeur n'a pas à gérer directement la mémoire dans le but de limiter les fuites potentielles de mémoire. De fait, de nombreuses personnes pensent qu'il n'est pas possible d'avoir des fuites de mémoire en Java. 5

C'est là encore un mythe à démystifier. En effet, en ne libérant jamais de références sur des objets et des variables devenus inutiles, le développeur empêchera le Garbage Collector de fonctionner correctement. Cela entraînera une application plus lente et des problèmes de mémoire. En outre, des fuites de mémoire sont possibles si les ressources d'entrées / sorties ne sont pas libérées ou lorsque les codes natifs embarqués via JNI sont mal utilisés.

Il faut garder à l'esprit que langage à mémoire managée ne signifie pas que le développeur ne doit pas prendre en compte les problématiques mémoire lorsqu'il programme. En effet, une vraie prise en compte est nécessaire même si le raisonnement est différent de ce qu'il peut être en C/C++.

Tout reste une affaire de bon sens comme souvent en matière de programmation.

5 Java est le seul langage que vous devez apprendre

Java reste le langage le plus utilisé en entreprise. En outre, le succès d'Android dans le monde mobile le rend parfaitement utilisable pour créer des applications mobiles. Java EE vous permet toujours de créer des applications Web avec le langage Java. Du côté du desktop, Java est également à l'honneur avec Swing ou Java FX. Ceci en dépit du fait que Swing soit vieillissant et que le support d'Oracle pour Java FX semble assez léger désormais. 6

Java vous permet donc de pratiquement tout faire. Pour certains, il est tentant de considérer que Java soit suffisant et qu'il ne faille pas aller voir ailleurs de temps à autre. C'est une grave erreur ! En effet, bien que Java permette de tout faire, cela ne signifie que Java soit la meilleure solution dans tous les cas. Ainsi, chaque langage de programmation présente des avantages et des inconvénients en fonction du type d'applications à implémenter et du domaine cible.

En tant que développeur, vous devez toujours être ouvert aux nouveaux langages et technologies. Rappelez-vous que tout va très vite en informatique. Ainsi, vous ne devez jamais perdre votre capacité à apprendre de nouveaux langages. Cela vous permettra d'avoir une carrière encore plus réussie.

Il faut donc encore une fois oublier ce dernier mythe persistant autour de Java. Java permet de tout faire mais ce n'est pas forcément la meilleure solution dans tous les cas. En tant que programmeur, votre rôle est d'être pragmatique avant tout !

Conclusion

Java reste le langage de programmation le plus utilisé, et, de fait, le leader que tous les concurrents souhaitent déloger. Tout ceci explique pourquoi les mythes entourant Java sont si nombreux et surtout si persistants. Dans cet article, je vous ai présenté les 5 plus grands mythes tout en vous expliquant pourquoi ils étaient faux. J'espère avoir réussi à vous convaincre du fait qu'ils n'ont plus de raison d'être aujourd'hui.

En conclusion, n'oubliez jamais que Java est là pour durer, mais que vous devez toujours rester pragmatique lorsque vous choisissez une technologie pour un projet informatique. •

Joan Anagbla
Développeur Fullstack JS
à **Zenika** *Paris*

Zenbot : Un chatbot qui répond aux questions en consultant l'API Search de Stack Overflow.

Stack Overflow est la communauté en ligne la plus importante qui permet aux développeurs d'apprendre et de partager leurs connaissances en programmation. Zenbot est un chatbot qui permet de répondre aux questions en effectuant des recherches depuis l'Api Search de Stack Overflow. Dans cet article, je vous propose d'intégrer Zenbot, en 3 étapes, aux plateformes Workplace et Slack. Enjoy !

niveau 200

ÉTAPE 1 :
la configuration d'une application

L'étape 1 passe par la configuration d'une application qui représentera le bot et contrôlera ses actions sur la plateforme concernée. Cette configuration se fait manuellement au niveau de chaque plateforme. Elle permet de définir un ensemble d'informations sur le bot telles que son nom, une description, les différentes permissions qui lui sont accordées, etc.

Workplace

Sur Workplace, le réseau social d'entreprise créé par Facebook, il s'agit de créer une *Custom Integration*. Lorsqu'on crée une Custom Integration, 2 objets sont en fait créés :

- Une application (avec des autorisations qui lui sont spécifiques).
- Une page de type *Bot* (uniquement visible au sein de votre communauté Workplace). Cette page servira, entre autres, de point d'entrée et de découverte de votre bot sur workplace.

Pendant la configuration, il vous sera demandé plusieurs informations sur votre bot dont l'URL sur laquelle le contacter. Nous verrons comment obtenir cette URL à l'étape 3.

Il vous sera aussi demandé de définir un token de vérification *verify token*. Ce token permet de vérifier l'authenticité des échanges entre la plateforme Messenger et le *webhook* déployé sur votre serveur.

À l'issue de cette configuration, un *token (Page Access Token)* est généré. Ce token servira par la suite à légitimer toutes les actions de votre webhook en tant que bot associé à l'application que vous venez de créer. Conservez-le précieusement et ne le divulguez qu'aux personnes de confiance (ex : l'équipe de développement). Nous verrons dans la suite de ce readme, comment utiliser ce token.

Vous trouverez plus de détails sur la création d'une application Workplace ici : https://developers.facebook.com/docs/workplace/integrations/custom-integrations/apps.

Slack

Pour ce qui est de l'intégration de Zenbot dans Slack, j'ai fait le choix d'utiliser les *Slash Commands*. Les Slash Commands permettent à l'utilisateur d'effectuer des actions (dans notre cas des recherches) en tapant des commandes depuis slack. Par exemple, pour consulter Stack Overflow au sujet des « **react hooks** », l'utilisateur pourra taper la commande "/***stack* react hooks**" depuis slack. Il verra alors s'afficher une liste de résultats correspondant à sa recherche.

La page Mes Applications (https://api.slack.com/apps) liste l'ensemble des applications que vous possédez sur Slack. Pour en créer une nouvelle. Il suffit de cliquer sur le bouton *Create New App* depuis cette page, puis de renseigner le nom de l'application et l'espace de travail (*Development Slack Workspace)* auquel elle sera associée.

Une fois l'application créée, il va falloir la configurer. Pour ce faire, il faut se rendre sur la page de configuration de l'application en cliquant sur son nom dans la liste des applications. La page de configuration contient nombre d'informations sur l'application dont son identifiant (*App Id*), le token de vérification (*Verification Token*), etc. Cette page permet également de gérer les permissions ainsi que les différentes features (*Bot, Slash Commands*, etc.) dont vous aurez besoin pour faire fonctionner votre application.

Pour intégrer Zenbot à la messagerie Slack, j'ai eu besoin d'activer 2 features :

- *Incoming Webhooks* : permet de poster des messages sur Slack depuis une source externe.
- *Slash Commands* : permet aux utilisateurs d'effectuer des actions en tapant des commandes. Cette caractéristique nécessite d'être configurée en renseignant :
  - un nom de commande (ex : /*stack*) ;
  - une URL de requête (l'URL que Slack contactera à chaque fois qu'un utilisateur entrera la commande /*stack*). Nous verrons à l'étape 3 comment obtenir cette URL ;
  - une courte description de la commande ;
  - une instruction d'utilisation (court message expliquant comment utiliser la commande) .

Une fois la configuration de l'application et de ses commandes terminées, il faudra installer l'application depuis le volet "*Install your app to your workspace*". Vous pourrez également choisir de distribuer votre application sur Slack, au-delà de votre espace de travail.

Pour de plus amples précisions sur la création d'une application, vous pouvez consulter cette page : Managing Slack apps.

Et voilà ! Vous savez désormais configurer une application Slack ou Workplace. Nous allons maintenant voir comment coder un webhook pour répondre aux requêtes des utilisateurs.

ÉTAPE 2 :
la création de Webhooks

Un webhook est une fonction de rappel HTTP (user-defined HTTP callback) généralement déclenchée lors d'un évènement (dans notre cas l'envoi d'un message à notre bot). Pour faire simple, notre webhook jouera le rôle d'intermédiaire entre notre chatbot et l'API Search de Stack Overflow. Il nous permettra de recevoir, gérer et envoyer des messages. À chaque fois qu'un utilisateur écrira un message à notre bot, il sera envoyé au webhook qui effectuera une recherche auprès de l'API Search de Stack Overflow, puis retournera une réponse à l'utilisateur.

La création de notre webhook consiste à ajouter quelques points de terminaison (endpoints) à un serveur HTTP comme Express par exemple.

Workplace

La configuration du webhook sur Workplace se fait en 2 étapes :

1 - L'ajout du endpoint de vérification du webhook : sur ce endpoint seront envoyées des requêtes de type GET servant à contrôler le token de vérification Facebook (*Verify Token)* défini lors de la configuration de la *Custom Integration* vue à l'étape 1. Cette étape est requise par la plateforme Messenger pour garantir l'authenticité de notre webhook.

```
// Adds support for GET requests to our webhook
app.get('/webhook', (req, res) => {
// Your verify token: should be a random string. let VERIFY_TOKEN = "<YOUR_VERIFY
_TOKEN>"
// Parse the query params let mode = req.query['hub.mode']; let token = req.query['hub.
verify_token']; let challenge = req.query['hub.challenge'];
// Checks if a token and mode were in the query string of the request
if (mode && token) {
// Checks if the mode and token sent are correct if (mode === 'subscribe' && token =
== VERIFY_TOKEN) {
// Responds with the challenge token from the request res.status(200).send(challenge);
} else {
// Responds with '403 Forbidden' if tokens do not match res.sendStatus(403); } } });
```

2 - L'ajout du endpoint principal : sur ce endpoint seront envoyées des requêtes de type POST correspondant aux messages envoyés par les utilisateurs.

```
// Creates the endpoint for our webhook
app.post('/webhook', (req, res) => {
let body = req.body;
// Checks if this is an event from a page subscription if (body.object === 'page') {
// Iterates over each entry - there may be multiple if batched body.entry.forEach(function
(entry) {
// Gets the message: entry.messaging is an array, but // will only ever contain one message,
 so we get index 0 let webhook_event = entry.messaging[0];
// Get the sender PSID let sender_psid = webhook_event.sender.id;
// Check if the event is a message or postback and // pass the event to the appropriate
handler function if (webhook_event.message) {
handleMessage(sender_psid, webhook_event.message); } else if (webhook_event.postback) {
handlePostback(sender_psid, webhook_event.postback); } });
// Returns a '200 OK' response to all requests res.status(200).send('EVENT_RECEIVED'); }
else { // Returns a '404 Not Found' if event is not from a page subscription
res.sendStatus(404); } });
```

Vous trouverez plus de détails sur la configuration du webhook ici : https://developers.facebook.com/docs/messenger-platform/getting-started/webhook-setup/.

Messenger définit 2 types d'évènements entrants : les *messages* et les *postbacks*. Les *messages* représentent les messages textuels écrits par l'utilisateur (textos) tandis que les *postbacks* sont des actions (clic sur un bouton par exemple). Une fois notre endpoint principal configuré, nous aurons besoin de lui ajouter des fonctions de gestion d'évènements :

- une fonction *handleMessage* pour gérer les textos ;
- une fonction *handlePostback* pour gérer les actions (clic boutons, sélections, etc.) ;
- une fonction *callSendAPI* permettant d'envoyer des messages à l'utilisateur via *l'API Send* de Messenger.

```
// Handles messages events function handleMessage(sender_psid, received_message) {
let response = {}; //... callSendAPI(sender_psid, response); }// Handles messaging_
postbacks events function handlePostback(sender_psid, received_postback) {
let response = {}; //... callSendAPI(sender_psid, response); }// Sends response messages
via the Send API function callSendAPI(sender_psid, response) { // Construct the message
body let request_body = {
"recipient": {
"id": sender_psid }, "message": response }// Send the HTTP request to the Messenger
Platform request({
"uri": "https://graph.facebook.com/v2.6/me/messages", "qs": { "access_token": PAGE_
ACCESS_TOKEN }, //do not forget to specify the Page Access Token
"method": "POST", "json": request_body }, (err, res, body) => {
if (!err) {
console.log('message sent!') } else {
console.error("Unable to send message:" + err); } }); }
```

Il y a ici 2 choses importantes à retenir :

- On appelle toujours la fonction *callSendAPI* pour envoyer une réponse lors de la réception d'un texto ou d'une action ;
- Pour que la requête de réponse soit acceptée par la plateforme Messenger, il faut **obligatoirement** ajouter dans le paramètre *qs (query string)*, le *Page Access Token* généré à l'issue de l'étape 1.

Pour finir, il ne nous reste plus qu'à définir la structure de nos réponses. Celles-ci sont généralement au format JSON. Messenger dispose d'une grande variété de templates prédéfinis pour nous aider à construire nos messages de réponse. On peut ainsi, envoyer un simple texto :

```
response = { "text": `Hi! I'm Zenbot and I can help you improve your skills on Stack
Overflow.` } ou bien un message riche composé d'un titre, d'une image et de boutons :
response = {
"attachment": {
"type": "template", "payload": {
"template_type": "generic", "elements": [{
"title": "How to send request on click React Hooks way ?",
"subtitle": "https://stackoverflow.com/questions/ 55647287/how-to-send-request-on-
click-react-hooks-way",
"image_url": attachment_url, "buttons": [
{
"type": "postback", "title": "It helped!", "payload": "useful", }, {
```

```
"type": "postback", "title": "Not very useful.", "payload": "useless", } ], }] } } }
```

Vous connaissez maintenant les grandes lignes de la création d'un webhook pour la plateforme Messenger. Vous trouverez sur https://developers.facebook.com/docs/messenger-platform/getting-started/quick-start, un tutoriel complet sur la conception d'un bot Messenger.

Slack

Pour rappel, j'ai fait le choix d'utiliser les slash-commands pour intégrer Zenbot à Slack. Les slash- commands sont envoyées de la même manière qu'un message classique depuis la barre d'envoi des messages. Cependant les slash-commands ne sont pas à proprement parler des messages dans la manière dont slack les interprète. La soumission d'une slash-command entraînera l'envoi à notre webhook d'une requête POST contenant en son corps une charge utile de données (payload).
Sur Slack, la création du webhook fonctionne à peu près de la même manière que sur Messenger. Il s'agit toujours de créer un ou plusieurs endpoints sur un serveur HTTP. Ces endpoints recevront les charges utiles envoyées lors des soumissions de la commande */stack* puis retourneront une réponse au format JSON.

> **À noter que sur slack,** la vérification du webhook est possible mais pas obligatoire.

Pour tout savoir du fonctionnement d'une slash-command et de l'implémentation de son webhook associé, ça se passe ici : https://api.slack.com/slash-commands.
Tout comme Messenger, Slack dispose d'un système de templating pour les réponses au format JSON. Ce système de templating va nous aider à structurer et enrichir nos messages. En plus de la structuration des messages, slack offre une grande variété d'outils de formatage des messages incluant le formatage des dates, l'ajout de fragments de code, etc. La page Messaging for Slack apps (https://api.slack.com/messaging) explique en détail comment composer un message structuré. Il est également possible de trouver sur le compte Github de Slack un memento des templates les plus couramment utilisés.
Vous trouverez plus de détails sur les intégrations Slack en général ici : https://api.slack.com/internal-integrations.
Nous avons fini de configurer notre webhook, il faut maintenant le déployer pour le rendre disponible sur web.

ÉTAPE 3 : le déploiement

Pour déployer notre webhook, j'ai choisi d'utiliser la solution d'hébergement Clever Cloud. Elle fournit aux développeurs une plateforme de déploiement avec une infrastructure robuste et une mise à l'échelle automatique. L'avantage d'utiliser Clever Cloud réside dans l'automatisation du déploiement de chaque nouvelle version de notre bot. Dans cette étape, nous expliquerons comment déployer notre webhook sur Clever Cloud.
Depuis le tableau de bord de Clever Cloud, il est possible de créer une nouvelle application. Pour ce faire, il faut :

- cliquer sur le bouton : "*create*" - puis choisir "*an application*"
- et enfin, sélectionner le repository du projet à partir du menu déroulant "*Select your Github repository*"
- Définir le type d'application que représente le projet en choisissant *Node* parmi la liste proposée
- Choisir le nombre d'instances nécessaires
- On peut ensuite ajouter une description et une région pour l'hébergement, puis cliquer sur "*CREATE*" pour lancer la création de notre application sur Clever Cloud- Nous n'avons pas besoin d'*add-on*, nous pouvons donc passer l'étape correspondante et cliquer directement sur *next* - Il nous est demandé de définir un certain nombre de variables d'environnement. C'est le parfait endroit pour renseigner toutes les valeurs en dur de notre bot, par exemple, le *token de vérification* qui doit rester confidentiel.

Il faut finalement cliquer sur *Next* pour lancer le déploiement de notre application sur le web.
La vidéo *NodeJS Mongo demo* résume bien ces différentes étapes de création d'une application sur Clever Cloud :
https://static-assets.cellar.services.clever-cloud.com/website/home/powerful-features-videos/ deploy.mp4
Si tout s'est bien passé, une notification nous avertit que le déploiement de notre application a été un succès.

Yay ! Notre bot est en ligne

Attention ce n'est pas fini. Nous devons encore récupérer l'URL sur laquelle notre bot a été déployé et la renseigner dans la configuration de la plateforme d'intégration de notre bot (Messenger/Slack) comme vu à l'étape 1. L'URL de déploiement est disponible et configurable à partir du menu "*Domain names*" de notre application sur le tableau de bord Clever Cloud.
Petite cerise sur le gâteau : il est possible de configurer un message de bienvenue sur Messenger en suivant les instructions de la page Welcome screen. https://developers.facebook.com/docs/messenger-platform/discovery/welcome-screen/
C'est terminé ! Nous pouvons maintenant tester que tout fonctionne correctement en écrivant quelques messages à notre bot depuis Messenger ou bien en utilisant la commande */stack* sur Slack !

Les liens

https://stackoverflow.com/
https://api.stackexchange.com/docs/advanced-search
https://www.facebook.com/workplace
https://slack.com/intl/fr-fr/
https://en.wikipedia.org/wiki/Webhook
https://developers.facebook.com/docs/workplace/integrations/custom-integrations/apps
https://api.slack.com/apps
https://api.slack.com/slack-apps
https://expressjs.com/fr/
https://developers.facebook.com/docs/messenger-platform/getting-started/webhook-setup/
https://developers.facebook.com/docs/messenger-platform/send-messages/templates
https://developers.facebook.com/docs/messenger-platform/getting-started/quick-start
https://api.slack.com/docs/verifying-requests-from-slack
https://api.slack.com/slash-commands
https://api.slack.com/messaging
https://github.com/slackapi/slack-platform-assets
https://api.slack.com/internal-integrations
https://www.clever-cloud.com/en/
https://developers.facebook.com/docs/messenger-platform/discovery/welcome-screen/

Renaud Pinon
Développeur
Créateur de Tiny Code Studio

niveau 300

Arduino : on crée un mini-debugger !

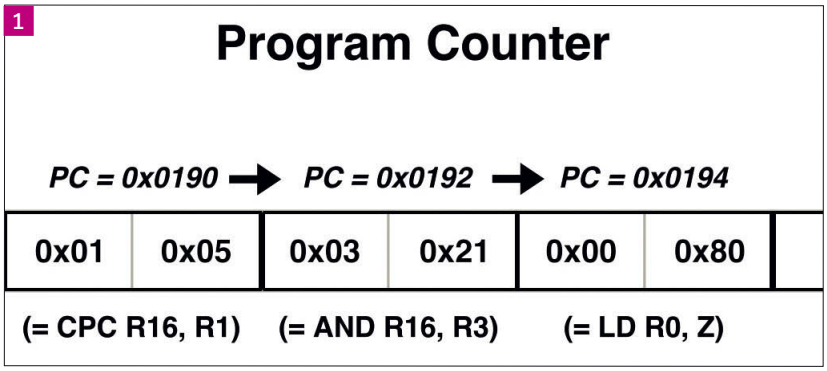
Après avoir vu le fonctionnement de la SRAM dans le précédent article (Programmez! 234), nous allons aujourd'hui nous concentrer sur les instructions stockées en mémoire Flash. Nous tenterons d'obtenir le numéro de l'instruction en cours du programme ou encore celui de l'instruction de retour d'une fonction. Nous créerons enfin un mini-debugger nous permettant d'arrêter un programme à une instruction donnée et d'afficher le contenu de la SRAM pour la fonction en cours.

Les explications données ici ont avant tout un but académique et devraient permettre de vous faire une idée du fonctionnement d'un programme du point de vue de la machine elle-même (ce qui est facilement transposable à d'autres hardwares).
Codes sources et images : https://github.com/renaudpinon/Arduino_miniDebugger

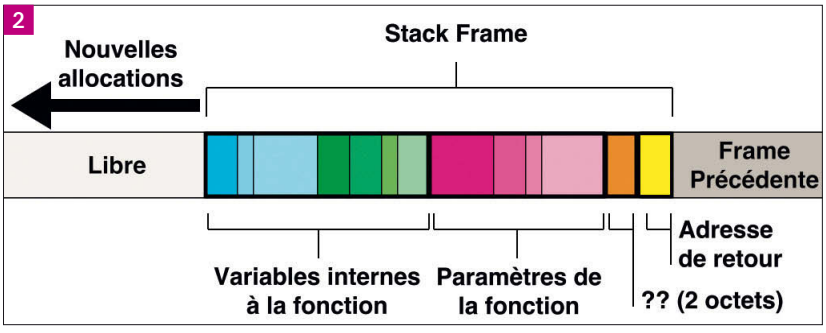
Rappels

Le programme, c'est à dire les instructions que vous avez écrites dans votre éditeur de code et envoyées à la carte, se situent dans la mémoire FLASH. Cette mémoire est inscriptible uniquement depuis votre ordinateur et est persistante même après extinction.
La SRAM quant à elle est la mémoire vive de la carte : elle contient les variables créées lors de l'exécution du programme qui sont stockées dans la "pile" (ou stack). Son début se trouve en fin de SRAM et est allouée "à l'envers", c'est-à-dire qu'à chaque nouvelle allocation, l'adresse de fin de pile est de moins en moins élevée et se rapproche du début de la SRAM.

1



2



Program Counter

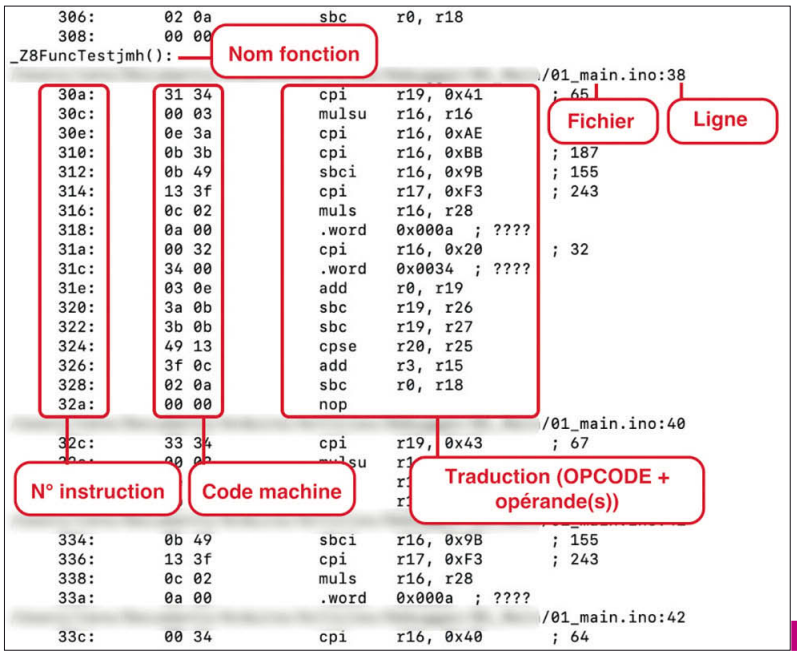
Dans la mémoire Flash, chaque instruction de votre programme a un numéro unique qui lui est attribué de façon incrémentale. Le Program Counter (compteur de programme en français, abrégé PC) est une simple variable contenant le numéro de l'instruction en cours. Elle est toutefois interne au processeur et ne peut donc être obtenue de façon directe (nous verrons un contournement possible tout à l'heure). Lorsque le processeur a fini d'exécuter une instruction en langage machine, il ajoute au PC le nombre d'octets constituant cette instruction (2 à 4 octets pour les instructions AVR), puis va lire l'instruction en mémoire FLASH ayant la nouvelle valeur de PC pour l'exécuter.
Voici une représentation schématique : 1
Dans ce schéma, les numéros d'instructions sont inscrits sur la ligne du haut (PC = 0x019X). Dans les cases se trouvent les octets de l'instruction en langage machine, et, en dessous, leur traduction en Assembleur. Dans cet exemple toutes les instructions font 2 octets : le program counter augmente donc ici de 2 à chaque fois. Evidemment les instructions d'un programme ne sont pas forcément à la suite les unes des autres : les branchements (appels de fonctions, conditions, for / while, …) changent la valeur du PC vers un numéro défini dans le branchement.
Nous utiliserons plus tard le program counter pour savoir à quel endroit du programme nous sommes et arrêter l'exécution s'il se trouve dans une plage bien précise.

Stack Frame : la théorie

Une Stack Frame correspond à la plage mémoire utilisée par une seule fonction dans la pile. La stack Frame contient donc les variables locales, les arguments passés à la fonction, les données générées par les instructions "PUSH" en Assembleur mais aussi le numéro de l'instruction de retour de la fonction.
Le format d'une stack frame dépend totalement du matériel exécutant le code, mais en ce qui concerne l'Arduino, on pourrait la représenter comme ceci : 2
N'oublions pas que les variables sont empilées de manière "descendantes", il faut donc lire le schéma de droite à gauche. On trouve tout d'abord, en jaune, l'adresse de retour de fonction (on y reviendra), d'une taille de 2 ou 3 octets suivant le modèle de carte (2 pour les Arduino UNO, 3 pour les MEGA). Puis se trouvent 2 octets "magiques" (je n'ai jamais réussi à trouver à quoi ils correspondent), puis les paramètres de la fonction et enfin les variables locales internes à la fonctions (variables + données de registres poussées lors de l'exécution de la fonction).

Adresse de retour : la théorie

Comme dit plus haut, l'adresse de retour de la fonction n'est pas une adresse en SRAM, mais un numéro d'instruction dans le programme, c'est-à-dire dans la mémoire Flash de l'Arduino. La suite d'outils AVR-GCC est capable, avec le programme *avr-objdump*, de désassembler le code compilé pour en sortir une représentation textuelle. En voici un exemple : 3



Les numéros d'instruction (les valeurs que peut prendre le Program Counter) se situent à gauche du listing. On voit bien qu'ils s'incrémentent à chaque ligne du nombre d'octets du code machine (+2 ici). Une "traduction" en Assembleur est ensuite présentée à droite, avec parfois à la fin de la ligne des commentaires précédés de ";". Le numéro de retour d'une fonction présent au début de la stack frame correspond donc à un numéro de la colonne de gauche. Pour obtenir ce numéro, on récupèrera les 2 (ou 3 suivant modèle) derniers octets de la stack frame. Cependant, la valeur qu'on obtiendra devra au préalable être transformée : premièrement, et contrairement à tout ce que j'ai pu vous dire jusqu'à présent, ses octets ne doivent PAS être inversés quand on les lit à la suite (c'est à dire que dans la réalité, il faut les inverser. Vous suivez ?).

Exemple: si on lit 01 suivi de 2B, cela donne d'habitude le nombre 0x2B01. Mais ici, l'octet de poids faible est le deuxième, pas le premier, ce qui donne donc 0x012B. De plus, les bits doivent ensuite être décalés de 1 vers la gauche (ce qui change vraiment tout : dans notre cas cela devient 0x0256 !). Voici un exemple de fonction permettant d'obtenir la valeur correcte de l'adresse de retour :

```
uint16_t translatePC(uint16_t address)
{
  // On doit inverser les octets pour
  // Obtenir le PC (ex: 0x2B01 devient 0x012B):
  uint8_t buff[2];
  buff[0] = (address >> 8);
  buff[1] = ((address << 8) >> 8);
  memcpy(&address, buff, sizeof(address));

  // Enfin on décale les bits de 1 vers la gauche:
  return (address << 1);
}
```

Enfin, je précise que l'adresse de retour ne renvoie pas à l'instruction qui a appelé la fonction, mais à celle qui la suit. Si la fonction est de type void, alors l'instruction sera à la ligne suivante. En revanche si la fonction retourne une valeur qui est affectée à une variable, l'instruction suivante sera l'affectation et donc la même ligne (exemple : pour *int x = Sum(11, 12);* : l'instruction qui suit l'appel de fonction est l'affectation du résultat à x).

La (douloureuse) pratique

Malheureusement la pratique pour obtenir la stack frame ou l'adresse de retour se révèle bien plus complexe que la théorie !

Premièrement les listings donnés ici ne fonctionneront pas de base dans Arduino IDE. La faute en revient à l'optimisation du compilateur qui détruit complément la structure des programmes : certaines fonctions, courtes ou utilisées une seule fois, seront transformées en instructions "inline", c'est à dire directement intégrées à la suite sans branchement. Cela rend impossible l'obtention du numéro de l'instruction de retour car… il n'y a plus de fonction ! Pareil pour les paramètres de fonctions ou variables locales : chaque variable potentiellement inutile sera purement et simplement supprimée à la compilation si elles ne sont pas utilisées, voire remplacées par une constante (notamment si vous mettez des valeurs en "dur" dans les appels de fonction, du genre int x = Sum(12, 13); : les paramètres de la fonction ne seront jamais dans la pile).

Deuxièmement, il est possible de modifier Arduino IDE pour compiler avec un niveau d'optimisation beaucoup moins agressif et propice au debug, mais ce ne sera pas de tout repos ! Je vous renvoie au fichier *ArduinoIDE_procedure.rtf* du GitHub qui explique comment faire cette modification pour Windows ou macOS. En substance, il s'agit premièrement de modifier le fichier platform.txt d'Arduino IDE afin de changer les options de compilation d'AVR-GCC pour ajouter -fno-lto, -fno-inline et -g -Og. Mais il faut aussi remplacer les exécutables d'AVR-GCC fournis dans Arduino IDE par les officiels, ceux d'Arduino IDE n'acceptant pas de baisser le niveau d'optimisation ! (et la raison reste un mystère pour moi…).

Je vous invite donc à faire les modifications d'Arduino IDE avant d'essayer le moindre listing de cet article, sans quoi il est peu probable que vous obteniez le résultat escompté…

Passés ces obstacles, et pour en revenir à la Stack Frame, on peut penser qu'il est très simple d'obtenir son contenu quand on voit le schéma précédent : on récupère le contenu de la SRAM à l'adresse SP (pour "Stack Pointer", correspondant au bas de la pile) sur une longueur égale à celle de la Stack Frame… Oui mais comment obtenir cette longueur ? Existe-il une fonction toute prête ? Eh non! Peut-être a-t-on un indicateur de la longueur à l'adresse SP - 2 ou à l'adresse de retour - 2 ? Nope !

Alors peut-être existe-t-il une fonction pour obtenir un pointeur vers le premier octet de la stack frame (c'est à dire les 2 octets entre l'adresse de retour et les paramètres de fonction) ? Eh bien oui ! Mais en fait non.

La suite GCC donne accès à la fonction ***void* __builtin_frame***

_address(int level) qui est censée faire juste cela : elle crée un pointeur vers le tout premier octet créé pour une stack frame. Le paramètre "level" permet de demander une stack frame précise : 0 pour la fonction en cours, 1 pour la fonction parente, 2 pour la fonction parente de la parente, etc. Si la fonction n'existe pas, elle est censée renvoyer 0 ou un nombre aléatoire. Malheureusement l'implémentation est totalement libre suivant l'architecture et en avr elle renvoie simplement la valeur de SP (c'est-à-dire la valeur la plus en bas de la pile). Et si on change le level à 1 ou plus, le résultat est incorrect ! Dommage.

Il va donc falloir ruser. Une possibilité pour obtenir la longueur est de stocker dans une variable globale la valeur de SP juste avant l'appel d'une fonction (donc avant que la stack frame ne soit construite), puis de retrancher la valeur de SP une fois qu'on est dans la fonction (donc après que la stack frame soit créée). Ce n'est pas très pratique, et, en plus, quand on revient dans la fonction parente, notre variable globale n'a plus la bonne valeur de SP. De plus, impossible d'obtenir la longueur de la stack frame dans la fonction loop() ou setup() (à moins de modifier le fichier cores/main.cpp des sources d'Arduino, mais je ne le conseille évidemment pas).

Nous allons cependant voir une autre technique, qui, même si elle présente quelques inconvénients, reste bien meilleure. Elle ne vient pas de moi, je l'ai trouvée à l'adresse suivante :

https://www.stderr.nl/static/files/Hardware/Electronics/Arduino/debug.cpp

Elle consiste à récupérer la variable Assembleur "__stack_usage" créée par avr-gcc à chaque "prologue" (phase de création d'une fonction en langage machine). Voici comment récupérer cette valeur :

```
uint8_t stack_usage;
__asm__ __volatile__("ldi %0, .L__stack_usage" : "=r"(stack_usage));
```

Si vous exécutez ce code dans n'importe quelle fonction, stack_usage contiendra la longueur de la stack frame. J'insiste sur le fait qu'il ne faut pas créer une fonction de debug qui contiendrait ces instructions et l'appeler à chaque fois qu'on en a besoin : dans ce cas la longueur retournée ne serait évidemment pas celle de la fonction qui nous intéresse mais celle de notre fonction de debug, ce qui n'a rien à voir ! On aurait pu en temps normal faire une fonction ayant l'attribut "inline" - ce qui a pour effet de copier le contenu de la fonction tel quel à chaque fois qu'elle est appelée - mais comme nous avons été obligé de modifier les arguments de compilation avec fno-inline, cela ne fonctionnera pas. Une solution viable serait alors de faire une macro qui affecterait la valeur de retour à une variable globale au lieu de l'affecter à stack_usage (les macros sont exécutées de façon "inline").

Attention : stack_usage contient la taille des données "prévisibles". Si vous déclarez un tableau de taille variable en plein milieu d'une fonction (disons : *int monTableau[variable];*), alors la taille renvoyée ne contiendra pas les octets du tableau ! L'idée pour résoudre cela est de calculer et stocker l'adresse du haut de la stack frame au tout début de la fonction : ainsi même si de nouveaux octets sont poussés dans la pile et même si on crée des tableaux de longueur variable, cela ne changera jamais cette valeur.

Voici un exemple :

```
#define DBG_RET_SIZE    2
#ifdef __AVR_3_BYTE_PC__
 #define  DBG_RET_SIZE    3
#elif defined(__AVR_2_BYTE_PC__)
 #define  DBG_RET_SIZE    2
#endif

int FuncTest(int a, int b)
{
  // début : on stocke l'adresse de départ :
  uint8_t stack_usage;
  __asm__ __volatile__("ldi %0, .L__stack_usage" : "=r"(stack_usage));
  uint16_t topSP = SP + stack_usage + DBG_RET_SIZE;

  int ret = a + b;
  // TODO: actions...
  PrintVariable(SP, topSP - SP + 1);

  return ret;
}

void PrintVariable(uint16_t address, int len)
{
  // On place un pointeur à l'adresse :
  uint8_t* pointer = (uint8_t*)address;
  for (int i = 0 ; i < len ; i+= 16)
  {
    for (int j = 0 ; j < 16 && i + j < len ; j++)
    {
      uint8_t val = *(pointer+i+j); // valeur d'1 octet à imprimer.
      Serial.print( (val < 16 ? "0" : "") + String(val, HEX) + " ");
    }
    Serial.println(""); // CRLF;
  }
  Serial.println(""); // CRLF
}
```

__AVR_3_BYTE_PC__ et __AVR_2_BYTE_PC__ sont déclarées à la compilation et permettent de savoir si on a une carte avec un program counter sur 2 ou 3 octets (dans la pratique : toute carte ayant une mémoire Flash > 128 ko a un PC de 3 octets). En tout début de FuncTest() on obtient la taille de la stack frame et on calcule son adresse de début. Cette adresse ne bougera pas quoiqu'on déclare par la suite.

Pour réaliser l'impression de la stack frame, j'ai repris la fonction PrintVariable() de mon précédent article puisqu'elle fait juste ce qu'il faut : imprimer les données à partir d'une adresse mémoire et pour une longueur donnée.

Obtenir l'adresse de retour d'une fonction

Maintenant que nous avons la stack frame, nous pouvons obtenir l'adresse de retour d'une fonction. Cela est utile pour obtenir ce qu'on appelle la "call stack", c'est à dire la pile des appels de fonctions : par exemple loop() à la ligne 40 appelle func1() à la ligne 80, qui à la ligne 83 appelle etc… Pour obtenir la valeur, on va

chercher les 2 derniers octets de la stack frame et on effectue la transformation avec la fonction TranslatePC() un peu plus haut.
Même si cela fonctionne très bien, il faut savoir qu'il existe une méthode plus simple pour obtenir cette adresse : la fonction intégrée ***void* __builtin_return_address(int level)***. Ouf, contrairement à ***__builtin_frame_address()***, celle-ci fonctionne ! Et comme précédemment, level est la fonction appelante : 0 pour la fonction courante, 1 pour la fonction parente, etc. Cependant seul le level 0 semble fonctionner en AVR. Attention : elle renvoie *void** et on serait tenté de croire qu'elle renvoie un pointeur, mais en fait elle renvoie une variable de type int ou long suivant la machine. Cette dernière représente l'adresse de retour de la fonction **avant** la transformation. A noter qu'il existe aussi une fonction censée donner cette valeur sans que l'on ait besoin de la transformer : ***void* __builtin_extract_return_addr(void* address)***. On est censé lui passer en paramètre le résultat de ***__builtin_return_address()***. Mais malheureusement, il semble bien qu'elle soit inopérante pour les AVR puisqu'elle ne renvoie jamais la bonne valeur, on se contentera donc de ***__builtin_return_address()***.
Voici un exemple d'utilisation :

```
uint16_t retAddress = (uint16_t)__builtin_return_address(0);
// TODO: transformer le résultat.
```

Obtenir le PC

On a obtenu l'adresse de retour d'une fonction. Mais comment obtient-on, en plein milieu d'une fonction, le numéro d'instruction exact où nous sommes ? C'est en fait possible et même très simple ! Après tout, en créant une fonction utilisée juste pour le debug et à laquelle on demande d'obtenir sa propre adresse de retour, si on l'appelle dans le programme "principal" alors elle nous renverra bien le numéro de l'instruction en cours. Bien sûr, en réalité, on n'obtient pas le PC réel puisque celui-ci est actuellement irrémédiablement dans notre fonction de debug, mais cela ne change rien pour nous.

```
uint16_t GetPC(uint16_t spValue)
{
  // StackFrame:
  uint16_t sp = SP;
  uint16_t diff = spValue - sp;

  uint16_t tmpAddress = 0;
  uint8_t* pointer = (uint8_t*)(sp + diff - 1);

  memcpy(&tmpAddress, pointer, sizeof(tmpAddress));

  uint8_t buff[2];
  buff[0] = (tmpAddress >> 8);
  buff[1] = ((tmpAddress << 8) >> 8);
  memcpy(&tmpAddress, buff, 2);

  return (tmpAddress << 1);
}
```

J'aurais pu utiliser ***__builtin_extract_return_addr()*** mais j'ai préféré rester sur les valeurs de SP. Par simplicité je renvoie une valeur sur 2 octets, mais le programme que vous trouverez sur le GitHub permet bien d'obtenir un numéro d'instruction sur 2 ou 3 octets. On appelle la fonction en la plaçant juste avant l'instruction qui nous intéresse :

```
int pc = GetPC(SP); digitalWrite(13, HIGH);
```

Désassemblage

Maintenant que nous avons le numéro d'instruction, voyons comment désassembler un programme pour ensuite corréler l'instruction à une ligne. Pour cela, il suffit juste de savoir où se trouve le fichier .elf créé par AVR-GCC. Avec Arduino IDE on peut l'obtenir très simplement : il suffit de passer l'éditeur en mode "verbose". Pour cela, ouvrez les préférences du logiciel et cochez les cases "Compilation" et "Téléversement" du champ "Afficher les résultats détaillés pendant :". Ensuite, si vous compilez un programme, vous verrez dans la console certaines commandes (par exemple avr-size) qui utilisent et affichent le chemin du fichier .elf (pour moi sur MacOs :

```
"/var/folders/zg/ldrv10ws4t113vw7hnf_32hh0000gn/T/arduino_build_219222/test.ino.elf").
```

Copiez le chemin dans le presse-papier, puis ouvrez une commande DOS ou Terminal. Notre but est d'exécuter le fichier avr-objdump qui se trouve dans le dossier hardware/tools/avr/bin d'Arduino IDE (sur macOS : il faudra faire un clic droit sur l'application puis "Afficher le contenu du paquet" et enfin aller dans le dossier Contents/java).
On exécute le programme dans notre Terminal / commande DOS :

```
chemin/applications/avr/avr-objdump -D -C -l chemin/fichier/arduino.ino.elf
```

(attention : -l est un L minuscule, pas un i !)

Si tout va bien, un listing de désassemblage du programme, qui ressemble à la capture présentée en début d'article, apparaît. Attention : à la moindre modification du programme et recompilation, il faudra penser à refaire cette commande !

Mini Debugger

Je vous propose maintenant de passer aux travaux pratiques avec la création d'un mini-debugger. Il devra être capable de :

- Communiquer avec l'utilisateur via le port série pour arrêter le programme à l'instruction en cours ;
- Demander au programme de s'arrêter tout seul quand il arrive dans une plage d'instructions ;
- Demander au programme de s'arrêter dès que possible ;
- Obtenir l'état de l'exécution (en cours, arrêté, arrêt prochain à l'instruction XXXX, etc.) ;
- Obtenir l'adresse de retour de la fonction en cours ;
- Reprendre l'exécution du programme ;
- Afficher dans le moniteur série le contenu de la stack frame lorsque le programme s'arrête.

Les étapes que devra suivre l'utilisateur seront les suivantes :

- Insérer dans son programme, à chaque début de fonction, un code permettant de calculer l'adresse de départ de la stack frame ;
- Insérer à chaque instruction un appel à une fonction de debug créant un "point d'arrêt" ;
- Compiler le code dans Arduino IDE ;

- Désassembler si besoin le programme avec l'utilitaire avr-objdump (pour vérifier que les numéros de lignes correspondent bien aux instructions et adresses de retour ;)) ;
- Ouvrir le moniteur série et taper :
- STOP pour arrêter le programme à l'instruction en cours,
- STOP XXXX YYYY pour arrêter le programme lorsqu'il arrive dans une plage d'instructions précise (XXXX étant l'adresse de départ et YYYY l'adresse de fin en hexa),
- NEXT pour relancer le programme s'il est stoppé et l'arrêter à l'instruction suivante,
- PLAY pour reprendre l'exécution,
- RETURN pour afficher l'adresse de retour de la fonction actuelle,
- STATE pour obtenir l'état actuel (arrêté, en cours, etc.).

Le code est disponible sur le GitHub donc je n'imprimerai pas la totalité ici. J'ai créé deux fichiers s'appelant Debugger.h et Debugger.cpp qu'il suffit d'ajouter à n'importe quel projet (copiez-les simplement dans le même dossier que votre fichier .ino et ils seront considérés lors de la compilation par Arduino IDE).
Pour résumer, Debugger.cpp contient une fonction pour interpréter les commandes reçues sur le port série, une fonction pour obtenir l'adresse de retour d'une stack frame, une fonction pour imprimer le contenu d'une variable en hexa sur une longueur de X octets (vue dans l'article précédent) et une fonction DebugBP() pour "créer" un point d'arrêt.
Cette dernière va vérifier l'adresse de l'instruction actuelle et sortir directement si les conditions d'arrêt ne sont pas réunies. Dans le cas contraire (on est à l'instruction voulue ou on doit s'arrêter à la prochaine instruction), elle rentre dans une boucle while() infinie, ce qui va "stopper" l'exécution du programme. Le programme vérifiera tout de même à chaque itération de cette boucle s'il y a des données envoyées par l'utilisateur sur le port série.

```
void DebugBP(uint16_t spValue, uint16_t previousSpValue, String file, uint16_t line)
{
  uint32_t pc = DebugGetPC(SP, spValue);

  do {
    if (_dbgState == DebugStateStopNext
        || (pc >= _dbgStopMin && pc <= _dbgStopMax))
    {
      if (_dbgState != DebugStatePause)
      {
        _dbgState = DebugStatePause;

        Serial.println(String(F("Stop fichier ["))
            + file + String(F("] : "))
            + String(line));
        Serial.println(String(F("PC = 0x"))
            + String(pc, HEX));

        // Adresse retour:
        _dbgReturnAddress = DebugGetPC(spValue, previousSpValue);

        Serial.println(F("Stack frame: "));
        int stackSize = (int)(previousSpValue - spValue + 1);
        DebugPrintVariable(spValue, stackSize);
      }
    }
    DebugSerialRead();
  } while (_dbgState == DebugStatePause);
}
```

La fonction accepte 4 arguments : spValue, previousSPValue, file et line. Les deux premiers sont la plage de mémoire de la stack frame, file est le nom du fichier actuel (macro __FILE__), et line est le numéro de ligne dans le fichier (macro __LINE__). Voici un exemple d'appel juste avant une instruction quelconque (DBGStackStart est une variable locale contenant l'adresse de départ de la stack frame) :

```
DebugBP(SP, DBGStackStart, __FILE__, __LINE__); delay(1000);
```

Ici un "point d'arrêt" est créé avant l'appel à delay(1000). Si l'utilisateur a demandé de s'arrêter à cet endroit (par une commande STOP, STOP XXX YYY ou NEXT), alors le nom du fichier et la ligne s'afficheront, suivis du contenu de la stack frame.
Pour plus de simplicité j'ai créé une macro au nom un peu plus court qui appellera cette fonction avec les arguments voulus :

```
#define BP()    (DebugBP(SP, DBGStackStart, __FILE__, __LINE__))
```

Du coup la ligne précédente devient :

```
BP(); delay(1000);
```

Ce qui est plus lisible et surtout moins fatigant s'il faut l'écrire à chaque ligne ;)
Il ne faut pas oublier à chaque début de fonction d'obtenir le début de stack frame. Il est un peu plus compliqué de créer une seule macro puisque tout doit rester "inline". Et une macro ne peut faire qu'une ligne. J'ai donc opté donc pour deux macros :

```
#define STACKLEN() __asm__ __volatile__("ldi %0, .L__stack_usage" : "=r"(_dbg
CurrentFrameSize))
#define STACKGETSTART() uint16_t DBGStackStart = SP + _dbgCurrentFrameSize + DBG
_RET_SIZE
```

STACKLEN() stocke la longueur de stack frame actuelle dans la variable globale *_dbgCurrentFrameSize*. *STACKGETSTART()* crée une variable **locale** *DBGStackStart* qui est l'adresse du haut de la stack frame. A noter qu'après cela il ne faut plus utiliser_dbgCurrentFrameSize de toute la fonction car sa valeur pourrait être incorrecte.
On appellera ces deux macros au tout début de chaque fonction :

```
void MaFonction()
{
  STACKLEN();
  STACKGETSTART();
  // TODO: instructions suivantes...
}
```

Un programme exemple très simple (allumage et extinction de la led de la carte) se trouve sur le GitHub. Voici le résultat dans le moniteur série après lui avoir demandé de s'arrêter entre deux numéros d'instructions :

```
Bienvenue dans le mini-debugger.
Tapez:
- STATE pour connaître l'état actuel,
- STOP pour arrêter le programme à l'instruction actuelle,
- STOP XXXX YYYY pour arrêter le programme lorsqu'il arrive dans une plage d'adresse,
- PLAY pour reprendre l'exécution,
- RETURN pour obtenir l'adresse de retour de la fonction actuelle (état stop),
- NEXT pour continuer jusqu'a la prochaine instruction.
Ceci est un test d'addition : 255
Ceci est un test d'addition : 255
executed: stop 1034 1060
Ceci est un test d'addition : 255
Stop fichier [/XX/XXXXX/XXXXXXX/XXXXXXX/test.ino] : 48
PC = 0x1052
Stack frame:
29 af 03 43 00 43 00 21 c5 00 08 cf
```

Les lignes s'affichent dans la sortie, mais si vous voulez savoir à quoi correspondent les numéros d'instructions, il faudra désassembler le programme. Il suffira de rechercher ".ino:[numero de ligne]" pour trouver les instructions de la ligne qui vous intéressent, et de rechercher "XXXX:" pour trouver l'adresse hexadécimale de l'instruction que vous désirez.

Mais où sont mes variables ?

Effectivement vous vous rendrez peut-être compte que les variables ne sont pas toujours présentes dans la stack frame d'une fonction. Pourtant vous les utilisez avec succès, mais elles ne sont pas là. Pourquoi ? Encore une fois, la faute en revient aux optimisations de compilation. Si on a une fonction int Sum(int a, int b) que l'on appelle avec deux paramètres en dur (exemple : *int value = Sum(10, 15);*), alors vous pouvez être sûr que le compilateur va créer des constantes qui n'iront pas dans la pile. Le seul véritable moyen est d'utiliser d'une façon ou d'une autre leur adresse. On peut par exemple écrire dans la fonction *Sum()* :

```
DebugPrintVariable((uint16_t)&param1, sizeof(param1));
DebugPrintVariable((uint16_t)&param2, sizeof(param2));
```

Ce qui aura pour effet d'imprimer le contenu des variables dans le moniteur. Vous pourrez constater ensuite que leurs valeurs se trouvent bien dans la stack frame.

Conclusion

Nous avons vu beaucoup de choses aujourd'hui. La théorie est très simple, mais la pratique se révèle plus difficile du fait des nombreuses optimisations à la compilation qui dénaturent totalement le programme. Une grande partie des concepts que j'ai exposés ici ne serviront jamais dans la vie d'un développeur lambda, mais j'ose espérer avoir donné les clés nécessaires à ceux qui aiment bidouiller et auront envie de tester le fonctionnement interne de leurs machines préférées (ce que j'ai expliqué valant pour de nombreux hardwares différents).

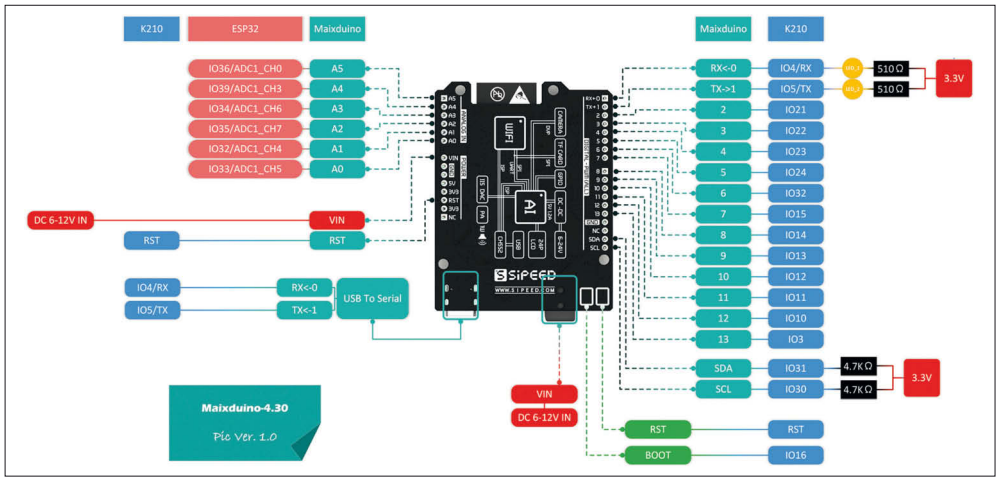
Ces concepts sont grosso-modo ceux que j'utilise dans mon application Tiny Code Studio pour obtenir la pile des appels des fonctions, arrêter l'exécution du code aux points d'arrêts définis par l'utilisateur et obtenir la valeur des variables. Bien-sûr, l'utilisation est bien plus poussée puisqu'il n'y a pas qu'un seul point d'arrêt, et le logiciel affiche une représentation des valeurs des variables suivant leur type. Mais les principes restent les mêmes.

Ceux qui le veulent pourront essayer d'améliorer ce mini-debugger et l'utiliser dans leurs projets pour se simplifier le déboggage (adieu Serial.print() !). On pourrait par exemple faire une macro qui, écrite après la déclaration d'une variable, afficherait l'adresse de cette dernière et sa taille dans le moniteur série, permettant ainsi à l'utilisateur, lorsque le code est arrêté, de demander sa valeur en hexadécimal. •

François Tonic

Maixduino de Sipeed

RISC-V (R-V) est un microprocesseur de plus en plus populaire. Les soutiens se multiplient ainsi que les plateformes de développeurs dans l'IA, l'IoT, le monde maker. R-V fut initié par l'université de Berkeley et il est open source. Maixduino est une des solutions R-V du marché. Visite guidée.



	Maixduino AI-IoT	Longan Nano	Wio Lite Risc-V
CPU	K210 (R-V 2 cœurs 64 bits 400 MHz + FPU)	GD32VF103CBT6 32 bits, mono cœur,	GD32VF103CBT6
Port USB	USB-C	USB-C	USB-C
Réseau	WiFi + Bluetooth (module ESP)	-	WiFi (module ESP)
Compatibilité Arduino	Oui (GPIO + développement) Support Grove	Oui	Support Grove
Connecteurs	Ecran - Caméra - LiPo - SD	SD	SD - LiPo
Mémoire	8 Mo de SRAM 8/16/128 Mo de stockage Flash*	32 Ko de SRAM 128 Ko de stockage	32 Ko de SRAM 128 Ko de stockage
Tarif	-30 €**	– 5 $	– 7 $

**selon modèle*
*** attention les prix sont très variables selon le commerçant*

Maixduino est souvent vendue en kit : carte de prototypage, caméra et écran LCD. La carte proprement dite est de la taille d'une Arduino Uno. Il dispose des mêmes GPIO/headers. Ce qui saute aux yeux : le connecteur USB-C, les connecteurs caméra et écran, les deux grosses puces : un SoC Sipeed K210 et ESP32 (modèle WROOM-32) !

La fabrication est très propre, la carte est plutôt bien agencée, excepté pour le port SD pas forcément le plus pratique. Les éléments en plastique des ports caméra et écran sont très pratiques, éviter d'appuyer trop fort dessus. Les boutons reset et boot sont accolés, attention aux gros doigts.

Au dos de la carte, le petit schéma gravé est très pratique.

Un modèle de développement plutôt ouvert

Comme le précise le constructeur, il y a 4 modèles de développement :

- Maxduino : la carte est compatible Arduino ;
- MaixPy : portage de MicroPython sur les SoC et les cartes Sipeed ;
- Kendryte SDK : SDK en langage C. SDK dérivant de FreeRTOS ;
- RT_Thread : OS open source chinois dédié aux IoT.

Au premier boot, comme toujours, on pensera à mettre à jour le firmware avec la dernière version. L'outil kflash vous facilitera le travail.

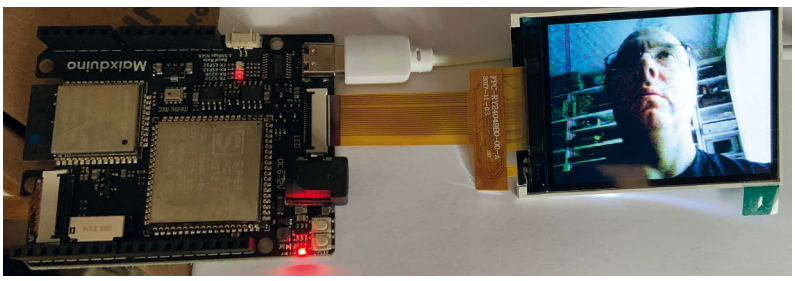
Pour le développement, le plus rapide est d'utiliser l'IDE maison : MaixPy IDE. On codera alors en micropython. L'outil est important, car il possède l'éditeur de code et aussi le serial terminal pour voir ce qui se passe sur la carte. L'accès terminal est parfois aléatoire. Il faudra connecter l'IDE et la carte pour pouvoir envoyer les codes et recevoir les informations de la Maixduino.

Le code Python proposé par défaut permet de vérifier le fonctionnement de la caméra et de l'écran. Ce qui est intéressant, c'est de visualiser, sur l'IDE, les courbes de données reçues de la caméra. Le micropython est le langage par défaut. Mais vous pouvez aussi utiliser Arduino IDE [installer la carte préalablement comme pour chaque carte non Arduino].

Les usages peuvent très divers :

- Reconnaissance faciale/objet avec traitement machine learning ;
- Émulateur rétrogaming [NES par exemple] ;
- Console portable [DOOM, Quake…] ;
- 3D.

De nombreuses librairies sont proposées par le Sipeed : caméra, LittlevGL, reconnaissance vocale, accéléromètre, SPI, etc.



Quelques ressources

Github RT_Thread :
https://github.com/RT-Thread/rt-thread
Github Kendryte :
https://github.com/kendryte/kendryte-standalone-sdk
Documentation : https://github.com/sipeed/MaixPy_DOC

NOUVEAU

Retour vers le passé :

redécouvrez les ordinosaures et les technologies des années 1970 à 2000 !

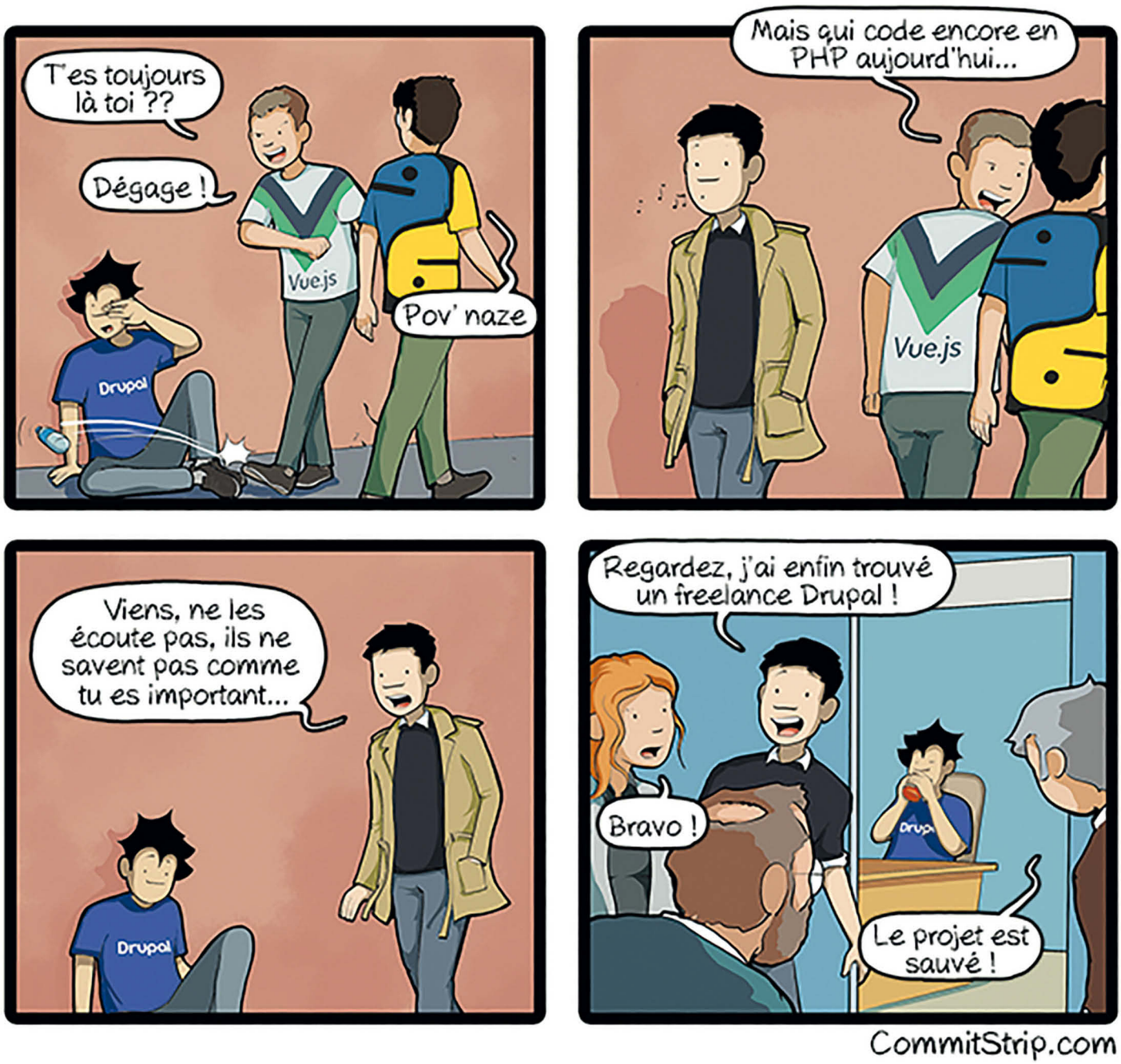
DISPONIBLE À PARTIR DU 9 DÉCEMBRE

Réservez & commandez directement sur **programmez.com**

6,66 € *(+frais de port*)* **36 pages**

*Revue trimestrielle. Editée par Nefer-IT. *Avec frais de port : 7,66 €*

Tout le monde t'aime



Une publication Nefer-IT, 57 rue de Gisors, 95300 Pontoise - redaction@programmez.com
Tél. : 09 86 73 61 08 - Directeur de la publication & Rédacteur en chef : François Tonic
Secrétaire de rédaction : Olivier Pavie
Ont collaboré à ce numéro : la rédaction de ZDNet, S. Saurel
Nos experts techniques : R. Lechapt, C. Villeneuve, M. Viley, N. Bouteillier, D. Bartaguiz, J. Antoine, P. Boulanger, S. Blanchard, B. Prieur, G. Saint Romas, D. Mouton, A. Giretti, J. Anagla, R. Pinon
Couverture : yewkeo - **Maquette** : Pierre Sandré.
Publicité : François Tonic / Nefer-IT - Tél. : 09 86 73 61 08 - ftonic@programmez.com.
Imprimeur : SIB Imprimerie (France)
Marketing et promotion des ventes : Agence BOCONSEIL - Analyse Media Etude - Directeur : Otto BORSCHA oborscha@boconseilame.fr
Responsable titre : Terry MATTARD Téléphone : 09 67 32 09 34

Contacts : Rédacteur en chef : ftonic@programmez.com - Rédaction : redaction@programmez.com - Webmaster : webmaster@programmez.com
Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1220K78366 - ISSN : 1627-0908 - © NEFER-IT / Programmez, novembre 2019

Abonnement :
Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex. - Tél. : 01 55 56 70 55 - *abonnements.programmez @groupe-gli.com*
Fax : 01 55 56 70 91 - du lundi au jeudi de 9h30 à 12h30 et de 13h30 à 17h00, le vendredi de 9h00 à 12h00 et de 14h00 à 16h30.

Tarifs
Abonnement (magazine seul) : 1 an - 11 numéros France métropolitaine : 49 € - Etudiant : 39 € CEE et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 € - Autres pays : nous consulter.

PDF
35 € (monde entier) souscription sur www.programmez.com

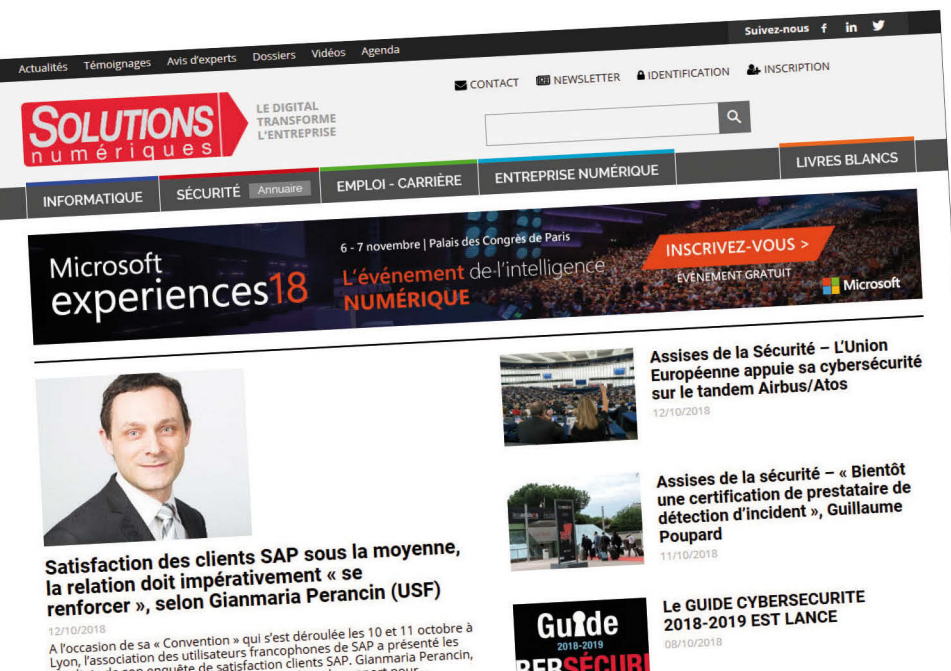
INFORMER
pour transformer l'entreprise

La dématérialisation, le Cloud,
les communications unifiées,
les nécessités de la cybersécurité
transforment le travail et toute l'entreprise,
les services publics.
Le magazine, le site, ses newsletters
vous informent sur cette actualité mouvante
et vous aident à décoder les tendances

Abonnez-vous

www.solutions-numeriques.com/abonnement/

4 sites thématiques, pour répondre à vos besoins d'information



- Vous êtes responsable informatique ou bien dirigeant ou cadre d'entreprise ? *2 sites répondent à votre profil*
- La cybersécurité vous concerne ? *Cliquez sur l'onglet. Vous trouverez les infos, l'annuaire, le lexique, etc*
- L'emploi, les salaires, les formations, les offres vous intéressent ? *Le site sur l'Emploi dans le numérique est à votre disposition*

www.solutions-numeriques.com

G·ECHO
cybersécurité

Solutions pour le c,der sécurité

MICRO FOCUS
Gold Partner
Security

Sécurité des développements : Fortify
Protection des données : Voltage
Evènements de sécurité : ArcSight

beyond SECURITY

Trouvez vos 0-days : bestorm
Scan de vulnérabilités : besecure

I onseilvser´ icesvaé´ eloppement

- Audits de code, test d'intrusion,
- Conseil, analyse de risque, PSSI,
- R&D, développement de solutions. . .

borme tio s et recrutement en SSy

www.g-echo.fr
contact@g-echo.fr
Toulouse - Paris