PROGRAMEZ!
Le magazine des développeurs

CODER MOINS CODER MIEUX

SPÉCIAL PRINTEMPS 2022

M 01642 - 7H - F: 6,99 € - RD

NO CODE
LOW CODE
GITHUB COPILOT
GÉNÉRER DU CODE .NET
CRÉER SON BOT SANS CODE
DELPHI & LE LOW CODE
POWER PLATFORM
SERVERLESS
BUBBLE

Numéro spécial en partenariat avec boomi salesforce Simplicité

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

#HS7-PRINTEMPS2022

Nous n'avons pas attendu le low code / no code pour réduire le code

Si vous nous lisez depuis longtemps, vous savez que le No Code / Low Code n'est pas un sujet qui surgit de nulle part. Nous en parlions avant même que ces outils ne fassent parler d'eux un peu partout. Depuis deux ans, une hystérie semble agiter une partie de l'écosystème et des communautés en voulant opposer le no code / low code aux développeurs. Et que ces outils supplanteront le métier de développeur, car tout le monde pourra développer son app, son site Web.

➜ FLASH-BACK

Comme je le disais en ouverture, la réduction du code et celle de la complexité de création d'un logiciel, sont des composantes de l'informatique. Dans le magazine Technosaures, nous étions revenus sur l'origine du langage Basic. Une des idées fondatrices du langage était de rendre accessible l'usage d'un ordinateur et sa programmation. C'était en 1964 ! À cette époque, l'informatique était un objet élitiste. Le Basic va permettre de démocratiser, timidement, ce curieux objet. Il faudra attendre plus de 10 ans pour que le micro-ordinateur naisse et commence réellement à se répandre même si finalement, cette démocratisation n'arrivera qu'à partir de 1977 avec l'Apple II, le Commodore PET et le TRS-80.

La programmation reste un objet curieux pour la plupart des utilisateurs. Et pourtant que d'efforts et d'inventivités pour réduire la complexité du développement. HyperCard marqua toute une génération par son approche visuelle et un code simplifié en langage naturel. L'autre étape est Visual Basic qui fera découvrir le monde de programmation à des nombreuses personnes.

Les puristes du code parleront même de programmation Lego, à cause des composants graphiques pour créer l'interface et le squelette du code créé automatiquement.
« Quoi du code généré par défaut ? Ça va pas non ? Ce n'est pas de la programmation ! »

© Excelsior, 17 février 2021. Tournage du Project X

➜ RETOUR À MAINTENANT

Revenons à notre premier propos de cet édito. Non, le no code / low code ne va pas remplacer le développeur. Ils sont complémentaires. Pourquoi opposer les deux approches alors qu'elles sont complémentaires. Le low code / no code peut faciliter la création d'apps et de sites Web ne nécessitant pas de mobiliser des développeurs et d'utiliser des outils et langages plus complexes.

D'autre part, le low code / no code peut résoudre, partiellement, le manque de développeurs. Mais attention, ces plateformes ne peuvent prétendre assurer tous les développements. Promettre que tout le monde pourra développer, est aussi une fausse promesse. En Low Code, il y a toujours du code à produire et le No Code nécessite une bonne maîtrise de l'outil pour pouvoir produire une app exploitable et répondant aux attentes (et à condition que les attentes soient clairement exprimées).

Ce numéro spécial est là pour revenir aux fondamentaux du No Code et du Low Code : les comprendre, les utiliser, les usages que l'on peut en faire. Plus largement, nous avons voulu mettre en avant l'idée de « Comment réduire le code ». Immense défi. Certains langages verbeux ont complexifié le développement.
Par cette approche élargie, nous parlerons de No Code, Low Code mais aussi de générateurs de code .Net, de serverless, des outils Delphi pour réduire le code, de Visual Scripting sous Unity, de Web Assembly, de GitHub Copilot, etc.

Nous tenons à remercier tous les dévs et experts qui ont participé à la rédaction de ce numéro et aux échanges que nous avons eus pour préparer le contenu.

François Tonic
Fan de HyperCard & de VB 3.0

LES PROCHAINS NUMÉROS

NUMÉRO SPÉCIAL
100% SÉCURITÉ

Disponible
dès le 2 septembre 2022

PROGRAMMEZ! N°253
NUMÉRO ÉTÉ 2022

Disponible
le 1er juillet 2022

Contenus

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

Les événements Programmez!

Meetups Programmez!

28 juin : Ecriture d'un système bancaire simple et distribué en utilisant Axon

Où : Devoteam 43 bd Barbès, Paris
Métros : Château Rouge (ligne 4)
A partir de 18h30

Informations & inscription : programmez.com

JUIN

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
		1	2	3	4	5
		Web2Days / Nantes				
6	7	8	9	10	11	12
			AlpesCraft / Grenoble			
		FIC / Lille				
			Le camping des speakers Bretagne			
				DevFest Lille		
13	14	15	16	17	18	19
JFTL / Paris				Cloud Ouest / Nantes		
	France API					
20	21	22	23	24	25	26
		JavaDay à Paris, par le ParisJUG	Serverless Day / Paris	Socrates / Rennes		
27	28	29	30	1 juillet		
		Hack in Paris / Paris				
			SunnyTech / Montpellier			

A VENIR

- JUG SummerCamp : 9 septembre / La Rochelle
- Cloud Nord : 29 septembre / Lille
- Paris Web : 6 & 7 octobre / Paris
- Volcamp : 13-14 octobre / Clermont Ferrand
- DevFest Nantes : 20-21 octobre / Nantes
- Open Source Experience : 8-9 novembre / Paris
- ParisTestConf : 15-16 novembre/ Paris
- Agile Tour Toulouse : 15-16 novembre / Toulouse
- Codeurs en Seine : 17 novembre / Rouen
- DevFest Strasbourg : 18 novembre / Strasbourg
- DevOps DDay : 1er décembre / Marseille
- SnowCamp : 25-28 janvier 2023 / Grenoble

Merci à Aurélie Vache pour la liste 2022, consultable sur son GitHub : https://github.com/scraly/developers-conferences-agenda/blob/master/README.md

Les partenaires 2022 de

PROGRAMMEZ!
Le magazine des dévs

Niveau maître Jedi

salesforce

softfluent

Scaleway
The cloud that makes sense

Niveau padawan

OSAXiS

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

Contactez-nous dès maintenant :
ftonic@programmez.com

Low-Code, the Raise of the Citizen (developer)

Jean-Pierre Riehl

Technology Innovation Lead – Avanade

Passionné par les données, fan de Power BI, guidé par l'innovation, mais orienté par le business, artificiellement intelligent, je pilote l'innovation et la "passion-for-tech" chez Avanade.

Je travaille depuis plus de 20 ans sur des projets stimulants, allant du web à l'IoT en passant par beaucoup de données et d'IA.

MVP Data Platform depuis 2008

Bien évidemment, il ne faut pas prendre ce terme au sens littéral. Créer un programme, une solution, une application nécessite d'écrire quelque chose, dans un langage informatique, donc écrire du code. Mais entre une formule Excel, des millions de lignes de C# ou encore des instructions assembleurs, il y a un monde.

On pourrait aussi penser que le low-code fait référence à la quantité nécessaire pour coder une action ou un écran, induisant ainsi qu'en dessous d'un certain nombre de lignes ou d'instructions, on tombe dans cette catégorie.

On pourrait également penser que le no-code fait référence à un outil ou un procédé dans lequel on n'écrit pas de code, mais on utilise une représentation visuelle avec des boîtes, des flèches, etc. Mais on sait très bien que ces outils (j'ai en tête les outils d'ETL par exemple) doivent être complétés par des fonctions, des routines voire même des programmes entiers.

Alors, comment qualifier le low-code ou le no-code ?

Ma conviction est que derrière ce buzzword ne se cache pas une techno particulière, mais un public, une audience, une catégorie d'utilisateurs. D'ailleurs, on a même donné un nom à ces utilisateurs : les *Citizens-x !* En effet, on voit fleurir partout les *Citizens-X* : Citizen-developer, Citizen-data scientist, Citizen-data analyst…

L'idée derrière le Citizen-X, c'est la prise de pouvoir d'une population existante dans l'entreprise sur une partie de la chaîne de valeur autrefois exclusivement réservée aux informaticiens. C'est l'idée que le dernier kilomètre, la dernière ligne de code, soit entre les mains de ceux qui vont utiliser le produit final.

On est entré dans l'ère du DIY (Do IT Yourself). Il n'est donc pas étonnant qu'un contrôleur de gestion ou un responsable des ressources humaines essaie de se fabriquer ses propres outils. Qui est le plus à même de développer une règle de gestion que l'utilisateur lui-même ? D'autant plus qu'il va changer plusieurs fois d'avis pendant la mise en œuvre donc autant qu'il en soit le créateur et le responsable.

D'ailleurs, ce n'est pas si nouveau. Prenez les L4G, les macros Excel, les ETL, la self-service BI… Tous se destinent à un public qui n'est pas spécialement développeur, qui n'est pas puriste de la programmation et du clean-code.

Est-ce destructeur de valeur (ou d'emploi) ? La réponse est évidemment non. C'est même créateur de valeur, car on crée plus de choses pour numériser le quotidien.

D'ailleurs, concrètement, on constate sur le terrain que les réalisations en low-code sont souvent des applications ou des processus qui n'auraient jamais fait l'objet d'un développement traditionnel. Ça a été le cas par le passé.

Le low-code ne remplace pas les développeurs. Au contraire, il leur permet de se focaliser sur les sujets les plus complexes, les applications les plus critiques.

Laissons les *Citizens* créer. Ils sont dans un environnement sécurisé et managé. Encadrés par des administrateurs, des experts, des centres d'excellence qui sont là pour surveiller, aider, supporter, former et aussi reprendre la main quand il le faut.

Soyons lucides, dans un marché du travail totalement pénurique, ce n'est pas étonnant de voir ce phénomène prendre de l'ampleur. On estime à 25 millions le nombre de développeurs dans le monde (source EDC 2018 « Global Developer Population and Demographic Study ») pour une population approchant les 8 milliards, avec un désir de vie numérique (on entre dans le métavers, n'oubliez pas).

Même si on doublait le nombre de développeurs, on n'arriverait pas à résoudre le défi de la transformation numérique du monde.

Alors quelle est la solution la plus simple ? Transformer les non-développeurs en Citizens-dev. Avec juste 0,1% de la population acculturée, voire éduquée, on quadruple le nombre de personnes capables de prendre part au développement numérique.

Et cela concerne tous les domaines technologiques. Il y a du code partout : dans les automates programmables (PLC), dans l'exploitation des données, dans la création de processus métiers (BPM), dans les ERP, dans la programmation des robots (éducateur de robots, un des métiers du futur selon Isabelle Rouhan autrice du livre "Emploi 4.0")…

De plus en plus, les éditeurs pensent les outils, les plates-formes, les solutions avec une place importante pour le Citizen-X qui écrira la ligne de code finale.

Par exemple, Microsoft ne s'y est pas trompé en adressant cette population avec Power Platform qui vient compléter toute son offre M365 et libérer la créativité des utilisateurs en les transformant en développeurs. La firme de Redmond honore ainsi sa mission : *empower every person and every organization on the planet to achieve more*.

En tant que développeur (dans toutes ses définitions), je regarde de très près les initiatives qui promeuvent le Citizen-X. Je cherche toujours à savoir comment je peux les accompagner pour créer plus de valeur, ensemble. C'est aussi ça l'innovation.

Créer une application mobile sans écrire de code en 15 mn avec Microsoft Power Apps

Gérard de Gouzel

Ingénieur de formation, développeur et chef de projet puis Architecte SI depuis 10 ans dans une banque française. J'accompagne les projets en conception d'application, j'anime des formations et je suis passionné de développement

Très intéressé par le développement d'applications, j'ai toujours cherché des outils permettant de faciliter la vie du développeur. Les plateformes No-Code / Low-Code permettent de réduire considérablement le temps de création à partir de composants pré-codés.

Power Apps est une plateforme de développement pour les applications métier. Elle est identifiée comme un leader par le Gartner dans la catégorie Entreprise Low-Code Application Platform. La Power Apps est un logiciel de la Power Plateform de Microsoft qui est composé de 4 outils :

- Power Automate (anciennement Microsoft Flow) pour les workflows
- **Power Apps pour le développement d'applications**
- Power BI permet la consultation de données
- Power Virtual Agent (Chatbot) **Figure 1**

La Power Apps propose deux approches de développement :

- Canevas (Application vide) : Création d'une application (web, tablette, mobile) à partir de zéro, on veut avoir le contrôle complet de l'affichage.
- Pilotée par modèle : Création d'application à partir d'un modèle de données (Dataverse, SharePoint, Excel) ou d'un squelette d'un site web pour créer des interfaces utilisateurs. **Figure 2**

Créer une application mobile pilotée par modèle à partir d'un fichier Excel en 3 click

1. Créer un fichier Excel sous forme de tableau pour stocker les données (exemple une liste d'ordinateur portable avec nom, marque, prix, image…) **Figure 3**
2. Connectez-vous à Power Apps. Pour cela il faut avoir une licence Microsoft.
3. Choisissez « Démarrer à partir d'un fichier Excel pour créer une application sur 3 écrans
4. Sélectionnez le fichier Excel stocké dans One Drive
5. Après quelques instants de patience, la Power Apps génère 3 écrans (consultation, visualisation et modification). **Figure 4**
6. On arrive ensuite dans un éditeur qui permet de faire du design graphique. Le développeur peut prendre la main pour rajouter ou modifier des écrans de l'application (création de la page d'accueil, ergonomie…) par glisser/déposer de nouveaux composants (boutons, images, graphiques…). On peut également rajouter des fonc-

Figure 1

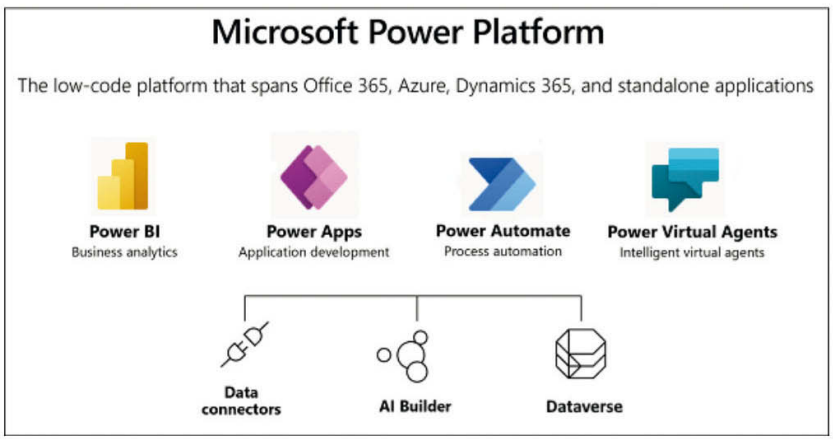


Figure 2

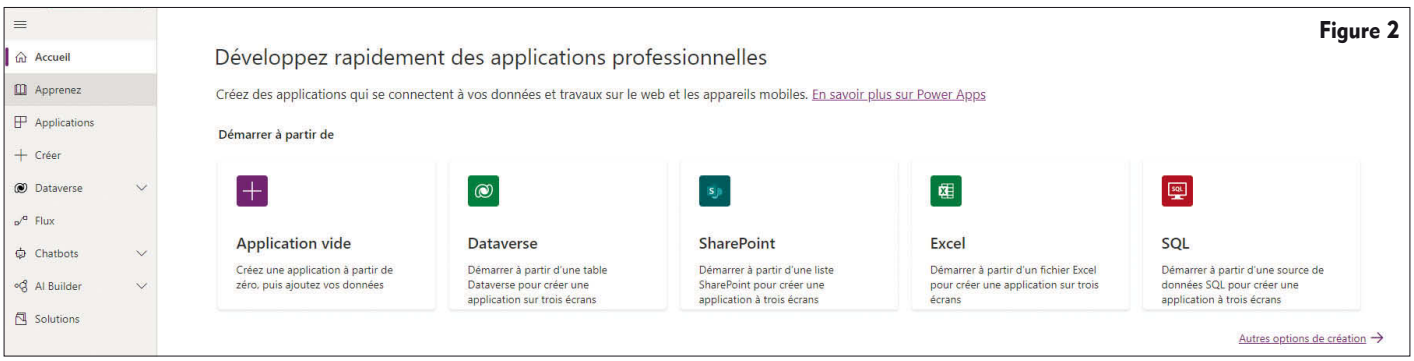


Figure 3

	Name	Category	Price	Image [image]	Overview
2	Aspire U	ACER	$300,00	https://diviceimage.azureedge.net/devices/Acer/Aspire_U.png	Cet Aspire TimeLine U M3 embarque la nouvelle carte Nvidia GeForce GT 640M. Cette puce se révèle très puissante pour sa catégorie et est bien mise en valeur dans cette ultrabook 15,6 pouces signé Acer.
3	Aspire M	ACER	$800,00	https://diviceimage.azureedge.net/devices/Acer/Aspire_M.png	Acer lance deux nouveaux portables ultrabooks baptisés Aspire M5 (M5-581TG-53314G52Mass et M5-581TG-73516G52Mass). Deux 15 pouces, un peu grands et lourds pour des ultrabooks, utilisant des composants à basse consommation et censés offrir 8 heures d'autonomie.
4	Aspire S3	ACER	$698,00	https://diviceimage.azureedge.net/devices/Acer/Aspire_s3.png	Le portable Acer Aspire S3 est un portable 13,3 pouces de seulement 1,3 cm d'épaisseur pour un poids de 1,4 kg. Son autonomie annoncée est de 7 heures.

tionnalités en Low Code avec le langage Power Fx de la plate-forme. Power Fx est le nom du langage de formule pour les applications canevas, c'est un langage à faible code que l'on tape directement dans une barre de formule de type Excel. **Figure 5**

7 Pour lancer un test de simulation de l'application utiliser le bouton play qui exécute l'application au format smartphone.

8 Pour publier l'application cliquer sur « publier cette version » et ensuite sur « partager cette application ». Sur iOS ou Androïd il faut installer « Microsoft Power Apps », s'authentifier avec le compte Microsoft PowerApps et ensuite on trouve la liste des applications disponibles. Il suffit ensuite d'épingler à l'écran d'accueil l'application pour pouvoir la lancer de manière autonome comme n'importe quelle application mobile native.

Conclusion

Power Apps est un outil très puisant qui permet de créer rapidement des applications avec très peu de code. Il existe de nombreux composants sur étagère pour créer des interfaces graphiques. La prise en main est facile, l'outil est riche et complet. Power Apps permet de se connecter à toutes les bases de données du marché grâce à des connecteurs sur mesure (en option). La plate-forme permet de réduire le temps de développement d'une application par 4 et le déploiement est très rapide. C'est un logiciel qui ouvre la création d'application à tous, même sans connaissances techniques. J'ai découvert Power Apps il y a 6 mois, c'est un outil sympathique qui favorise la créativité. J'apprécie l'interface graphique qui reprend l'apparence et la convivialité propres aux outils Microsoft, il y a une similitude avec PowerPoint. Un outil à mettre entre toutes les mains des créateurs d'applications.

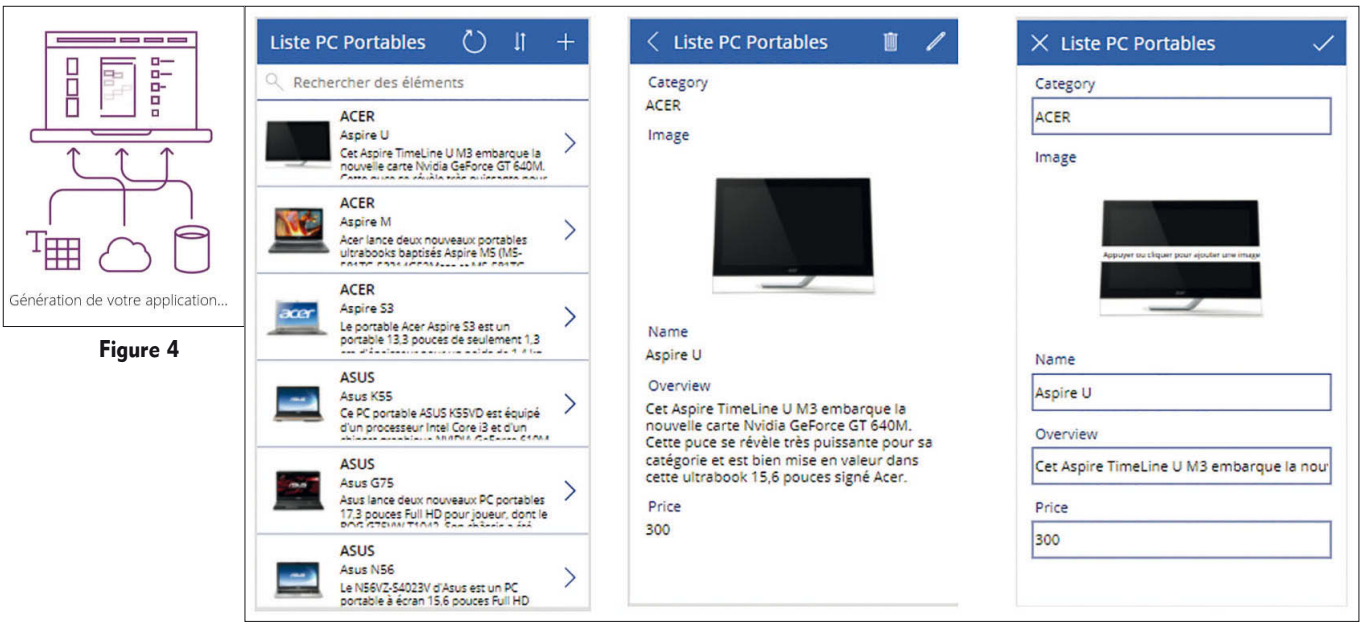
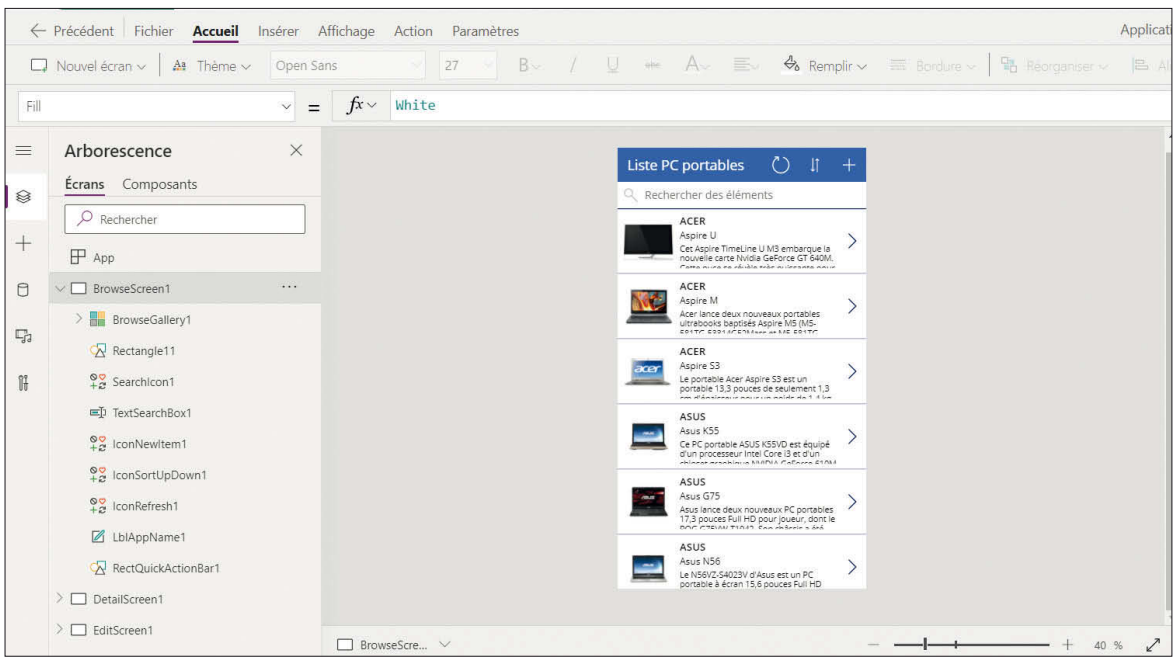


Figure 4

Figure 5



Développer avec peu ou sans code, mais développer quand même

Créer une application nécessite du développement logiciel : que l'on soit avec des outils traditionnels (avec son IDE préféré) ou avec des solutions No/Low-Code le schéma reste similaire. Une personne (ou une équipe) utilise un/des outils pour concevoir jusqu'à son déploiement une application. On peut appeler cela développer/modeler, mais au final le schéma est le même. Il y a un besoin de définir ce que fait l'application, donc de concevoir les IHM, son UX, ses traitements et ses données. La différence réside dans le niveau d'abstraction, de technicité et l'outillage utilisé pour y parvenir.

Alain Faure

Alain Fauré travaille depuis 25 années dans la conception et la réalisation de solutions informatiques. Il a travaillé dans des ESN, une banque, un éditeur de progiciel, en France et à l'étranger. Chez Octo technology, il partage son activité entre missions de conseils et missions d'architecture sur des projets de développement. Ceci dans divers domaines d'activité : banque-assurance, industrie, secteur public, transport. Alain est fortement impliqué dans le mouvement du low-code et no-code. Il était un speaker à la Duck Conf et a écrit de nombreux articles sur le sujet. Il est un des référents low-code/no-code chez Octo technology.

Des compétences de création/conception d'applications informatiques sont absolument nécessaires. Quand bien même le besoin en compétences en "programmation/codage" est beaucoup plus faible avec les outils No/Low-Code (ce sont les outils qui créent, déploient et exécutent le code sous-jacent), la maîtrise des concepts informatiques et en particulier de conception reste incontournable. Seuls des outils très sommaires, simples, mais très normatifs peuvent donner l'illusion d'une non-nécessité de connaissances informatiques. En contrepartie, ces outils n'offrent que des possibilités limitées dans les variations de fabrication d'applications. On peut par exemple citer des outils de création de blog ou de sites webs basiques.

Par analogie (toujours dangereuse, mais tant qu'on se souvient que l'exercice a des limites), si on assimile l'écriture d'un roman à celle d'un programme informatique, selon que les outils proposent de taper des lettres, des mots ou des phrases toutes faites, on sent bien que lorsque l'on gagne en vitesse d'écriture d'un côté, on perd en créativité de l'autre. Enfin, quel que soit l'outil, écrire un roman nécessite d'avoir des compétences dans la construction d'une histoire, en grammaire… Ce qu'aucun outil (on va dire simple) ne permet d'apporter.

Enfin, il faut distinguer le No du Low Code. Dans beaucoup de présentations et discours, ils sont présentés ensemble et plus ou moins assimilés l'un à l'autre. En fait, ils sont bien distincts. Que ce soit sur le "pour quoi" ils sont adaptés, les compétences nécessaires pour les manipuler ou encore les éditeurs qui les supportent, on navigue bien dans deux mondes.

Résumons :

- Le No-Code s'adresse aux "Citizens Developers" dont les compétences en informatique sont modestes, qui ont besoin de développer un concept simple rapidement et dans un budget très contraint. Les éditeurs sont nombreux et en plein foisonnement.
- Le Low-Code lui permet de créer des applications d'entreprises plus pérennes avec des possibilités de développement et de fonctionnalités plus étendues. Les outils généralement plus anciens et matures nécessitent plus de compétences et des formations dédiées au Low-Code et sur leurs plates-formes en particulier. **Figure 1**

Nous y revenons en détail dans le chapitre *"No ou Low-Code comment savoir lequel est adapté"*.

	NO CODE	LOW CODE
Quoi ?	Des outils intuitifs accessibles à tous	Des outils nécessitant des compétences et une formation IT
Pourquoi ?	Prototypes / MVP Bureautique étendue (automatisation)	Permettant de créer et déployer des applications d'entreprise
Pour qui ?	Citizen Developer	Model Developer
Quels produits ?	Glide, Adalo, Appsheet, Weebly, Dropsource, Draftbit, Zapier, Airtable, Caspio, Bubble, BettyBlocks…	Mendix, Outsystems, PowerApps, Appian, Salesforce, Pega…
Le plus	accessible rapide	souple pérenne

Il ne faut pas se méprendre sur le terme de "Citizen developer". Ce terme implique que tout un chacun est en capacité d'utiliser ces outils, mais leur maîtrise va passer par un apprentissage de nombreux concepts informatiques. L'utilisation d'interfaces graphiques pour créer une application dispense d'apprendre un langage de programmation, mais pas de comprendre les concepts sous-jacents. Par exemple les concepts de composant, de format de device, de droits, d'entités et de relations", d'environnements, de versions…

Pour faire l'analogie avec les "mix" qui permettent de faire des gâteaux facilement, il n'y a plus besoin d'avoir beaucoup de connaissances et la boîte vient avec la plupart des ingrédients ainsi que la recette. Il n'en reste pas moins qu'il y a plein de prérequis implicites : lire la recette en avance au cas où il faille acheter des éléments, avoir un minimum de matériel (bol, batteur), savoir mesurer des volumes et des poids, faire fonctionner le four, faire des conversions entre les degrés et des numéros si le four est gradué différemment de la recette, se rendre compte si ça chauffe plus vite ou moins vite que prévu… **Figure 2**

Figure 1

Sylvain Fagnent

Après 22 années d'expérience d'abord en développement informatique puis en architecture, Sylvain Fagnent a acquis une vision étendue des architectures fonctionnelles, applicatives et techniques des SI. Il a accompagné les grandes organisations dans leur transformation digitale et dans des projets d'innovation. De par son expérience de l'IT et des SI, il participe à de nombreuses due diligence technique de startups (secteurs variés) pour le compte de grands groupes ou de VC. Depuis maintenant 3 ans, il est impliqué dans le mouvement des solutions d'abord no-code puis low-code. Avec Alain Fauré il a été un speaker à la Duck Conf et a écrit de nombreux articles sur le sujet. Il est un des référents low-code/no-code chez Octo technology.

Figure 2

Les outils low-code et no-code réduisent le niveau de compétence informatique nécessaire pour construire une application

Figure 3

Les outils de no-code permettent de valider des idées avec des prototypes fonctionnels…jetables !

No ou Low-Code comment savoir lequel est adapté ?

Une des promesses des outils No/Low-Code est de donner des espaces de liberté à tous les employés (voir chapitre suivant) et ainsi leur permettre de développer des applications par eux-mêmes. Oui, mais pour quels usages et avec quels types d'outils ?

De notre expérience, nous sommes arrivés schématiquement au découpage suivant. Pour tester une idée rapidement sans avoir à faire de développement complexe, en mode POC simple (voir MVP si la couverture fonctionnelle du ou des outils est suffisante) le No-Code est bien adapté :

- **Complexité à implémenter** : simple
- **Prérequis** : des employés formés à des outils No-Code sélectionnés
- **Coûts d'investissement** : quelques jours
- **Le petit plus** : les solutions sont ouvertes et on a la possibilité d'en associer plusieurs pour couvrir son besoin. Des extensions sont aussi possibles (voir Extensibilité et développement de plug-in).
- **Le bénéfice** : si le concept ne prend pas, vous avez dérisqué pour pas cher votre idée, si le concept prend, vous avez à peu de chose près les spécifications fonctionnelles de votre prochaine application (à développer en spécifique par exemple enrichies de ses futures évolutions) **Figure 3**

Pour des POC / MVP et le développement d'applications orientées efficacité opérationnelle et avec une complexité fonctionnelle plus importante le Low-Code est l'outil adapté :

- **Complexité à implémenter** : plus importante que ce que permettrait nativement les outils No-Code
- **Prérequis** : des employés formés à une plate-forme Low-Code sélectionnée complétée par du soutien par des experts externes sur la plate-forme
- **Coûts d'investissement** : quelques mois auxquels il faut ajouter les coûts récurrents des licences
- **Le petit plus** : les plates-formes Low-Code sont fonctionnellement plus riches (mais aussi plus complexes), une seule solution doit suffire à couvrir les besoins d'autant qu'ici aussi des extensions sont aussi possibles (voir Extensibilité et développement de plug-in)
- **Le bénéfice** : vous développez des applications indispensables opérationnellement, mais souvent considérées comme non stratégiques par votre DSI (régulièrement dépriorisées dans les roadmaps). Vous sollicitez moins les ressources stratégiques de votre DSI grâce à la mise en place d'une filière Low-Code (voir chapitre suivant).

Note : *Il est important de garder à l'esprit que l'utilisation d'une plate-forme de Low-Code fait changer d'ordre de grandeur par rapport au No-Code. Que ce soit dans les coûts de licences, les profils des concepteurs/développeurs en No/Low-Code ainsi que leur formation et le support nécessaire (voir chapitre suivant).*

Quel outil choisir ?

Les outils de No/Low-Code sont parfaitement assimilables à des progiciels techniques avec leurs apports et accélérateurs technico-fonctionnels, mais aussi leurs limites. Voici 4 axiomes pour vous guider dans le choix de votre outil No/Low-code

Axiome 1 : On ne doit utiliser que ce que le progiciel sait faire uniquement.

Pour le reste, soit on s'en passe, soit on trouve des solutions alternatives en dehors de la solution Voir axiome 2.

De ce premier Axiome, on en déduit qu'il faut absolument vérifier que l'outil possède les fonctionnalités techniques obligatoires à l'application que l'on souhaite créer. Ceci est un point crucial pour les outils "No-Code" dont le spectre est beaucoup moins large que les outils de type "Low-Code", ce qui nous amène au second axiome.

Axiome 2 : Lorsque l'on est VRAIMENT bloqué et de manière exceptionnelle, on doit pouvoir se débloquer en étendant la solution.

Il peut s'agir de composants génériques, que l'utilisateur scripte directement à l'intérieur de la solution ou alors en développant de nouveaux composants sur la base de langages classiques (JavaScript, Java, .Net, …) et intégrés ensuite dans la solution de manière à être utilisés de la même façon que les composants natifs. Cette dernière capacité a l'avantage d'accélérer le développement de nouveaux composants en s'appuyant sur un écosystème lié au produit. Pour que ces développements exceptionnels restent maintenables et maîtrisables, il faut que cette propriété d'extensibilité soit fournie par l'outil. Par exemple par l'exposition d'interfaces publiques, ou par la capacité à intégrer des appels API externes.

Notre expérience nous a appris que sans cette possibilité d'extension il y a un risque majeur à rester bloqué. L'éventualité de faire évoluer à court terme la solution pour vos besoins soit en la tordant ou en la faisant développer en spécifique par l'"éditeur (si toutefois il accepte) est peu recommandée et de toute façon peu probable. Les éditeurs de ces solutions SaaS ne maintiennent et ne veulent maintenir qu'une seule version, et ce afin de rester

dynamique sur un marché en très forte évolution. Votre seul espoir serait éventuellement de lui remonter votre demande qu'il intégrerait alors dans sa future roadmap.

Axiome 3 : Les fonctionnalités liées au cycle de développement sont primordiales

Les éditeurs mettent souvent en avant les fonctionnalités métiers : capacité à afficher des graphes, à générer une application mobile, à stocker les données, à proposer un système de droits… Mais ne s'étendent pas sur leurs capacités, parfois très limitées, à faciliter la vie du développeur/modeleur : capacité à partager le code (si multi-développeurs), à avoir plusieurs versions de l'application, à gérer plusieurs environnements… (https://blog.octo.com/les-dix-commandements-dune-plateforme-no-code-mature/). Certains outils vont proposer de manière native une gestion d'environnements, avec un système de versionning et de bascule du code, voire des données. D'autres éditeurs ne vont rien proposer du tout et il faudra alors accepter que le développement se fasse sur la production… ce qui limite de facto l'étendue des projets potentiels. Un autre contournement est de déclarer deux applications : une de dev et une de prod, puis ensuite trouver un moyen de "promouvoir" l'application de dev… Encore une fois, ce type d'approche limite grandement la taille et la criticité des applications éligibles.

Axiome 4 : La réversibilité n'existe pas

Tout le travail de développement d'une application No/Low-Code est effectué en utilisant des outils de modélisation graphique fournis par l'éditeur de la solution. Sans ces outils point de salut : impossible de modifier les modèles. Exporter ces modèles signifierait qu'il y a une norme de "modélisation No/Low-Code", ce qui n'est pas le cas. On pourrait imaginer aussi reprendre manuellement les programmes générés par la solution de No/Low-Code. Cela n'est pas possible, car déjà une partie des outils ne va pas générer du code, mais repose sur une application de l'éditeur qui "exécute" directement les modèles développés. D'autres solutions vont générer du code qui lui-même doit être déployé sur des serveurs (HTTP, base de données) et utilisent des librairies fournies par l'éditeur. Il est illusoire de penser que l'on pourra maintenir un tel système.

En cas de réversibilité, toute l'application devra être re-développée dans la cible. Il faudra exporter les données pour pouvoir ensuite les importer dans la cible. Suivant les outils No/Low-Code utilisés cet export nécessitera ou non du développement. Quelques éléments comme les feuilles de style pourront aussi être récupérés partiellement dans certains cas.

Quelles applications sont éligibles au No/Low-Code ?

Pour déterminer si un projet va grandement bénéficier d'une approche No/Low-Code nous examinons 4 dimensions :

La dimension Technico/applicative (les usages cibles)

C'est la dimension la plus évidente et qui n'échappe à personne. Les plates-formes No/Low-Code fonctionnent comme des progiciels techniques, il faut se fondre dans ce qu'elles proposent et en particulier en accepter les options techniques qu'elles imposent à de nombreux niveaux : look and feel des IHM, limitations sur de la complexité du modèle de données, des traitements, localisation de l'infrastructure… Ces options sont liées à la plate-forme, les adopter reste le meilleur moyen de tirer un profit optimal de son investissement.

De ce fait l'éligibilité d'une application à un développement Low-Code sera déterminée par le fait que :

- la plate-forme Low/No-code supporte nativement les besoins techniques identifiés
- les parties prenantes sont prêtes à adapter leurs exigences pour se caler sur les capacités, le mode de fonctionnement de la plate-forme

La dimension stratégique

Cette dimension s'applique de manière plus forte pour les solutions Low-Code qui demandent un investissement plus lourd et dans la durée que les plates-formes No-Code.

Comme déjà évoqué, No ou Low-Code, la réversibilité sur ce type d'outils n'existe pas. Le lien fort avec la plate-forme de Low-Code doit être accepté et maîtrisé en termes de risque financier/maintenance/réversibilité (vendor locking).

Partir sur du Low-Code c'est faire un choix produit tout comme pour un progiciel. Utiliser des produits logiciels est quelque chose qui est commun dans les entreprises : SAP, Salesforce, Outlook, suite MS Office. Et aussi pour des développements internes : développement sur les technologies MS (.net), l'utilisation du cloud AWS et ses services managés, la mise en place d'un core banking system pour les acteurs bancaires….

Cela peut très bien se passer si la contractualisation et la relation avec l'éditeur sont soignées. Le marché est en ébullition, il y a donc beaucoup d'acteurs qui ne sont pas encore matures et qui essaient de prendre leur place coûte que coûte.

Pour mitiger les risques, il faut passer par une contractualisation : négociation des licences et du support sur le long terme, obtention des droits d'utilisation du produit en cas d'arrêt, gestion des montées de versions et des migrations éventuelles.

Un facteur clé est de privilégier un éditeur avec un écosystème riche, répandu et ouvert. Outre le gage de maturité de la plate-forme, cela permet de s'appuyer sur un réseau de partenaires et de ne pas dépendre uniquement de l'éditeur de la solution.

Dans tous les cas, il est nécessaire de choisir une solution autorisant une architecture hybride de type API permettant de limiter l'étendue des développements en No/Low-Code (par exemple API et données en spécifique, IHM et logique applicative en No/Low-code)

La dimension financière

Les plates-formes de Low-Code sont des solutions généralement facturées à l'usage. Cela est un avantage important quand l'activité est faible, car on bénéficie d'une plate-forme complète à un prix modique.

Il y a une différence importante entre les outils de No-Code dont la tarification commence en général à quelques dizaines d'euros par mois et les outils de Low-Code où le tarif d'entrée se compte plutôt en milliers d'euros par mois. Toutefois, cette

solution évolue et certains outils de Low-Code ont aussi maintenant une offre plus adaptée à un nombre restreint d'utilisateurs.

Les plans gratuits permettent d'avoir un aperçu de la plate-forme, mais ne sont pas adaptés à un usage réel, même simple. Prendre un plan payant permet de gagner du temps et d'accéder aux fonctionnalités qui font la valeur du produit. Ne pas payer cher au début est assez simple, et ce avec n'importe quelle plate-forme, mais lorsque l'on passe à l'échelle en achetant plus de services pour soutenir la croissance et garantir une qualité de service aux utilisateurs finaux de l'application, la facture peut vite monter (ex. lors d'une augmentation du nombre d'utilisateurs authentifiés ou des exécutions de tâches automatisées…).

Quand l'activité augmente, on bénéficie des mêmes fonctionnalités, mais le prix augmente mécaniquement. L'activité peut être mesurée de différentes manières : au développeur, à l'application, à l'utilisateur, à la donnée, à la CPU, ou un mélange de ces paramètres. Le modèle de tarification et sa progressivité doivent être examinés attentivement au regard du cas d'utilisation. Une plate-forme qui facture à l'utilisateur sera plus adaptée à une application utilisée intensivement par quelques collaborateurs qu'à une application utilisée une fois par mois par l'ensemble des collaborateurs.

Il est recommandé d'anticiper et surtout de monitorer les usages, surtout ceux qui sont susceptibles d'impacter la facture à l'usage. Administrer les applications pour suivre leurs usages en recensant celles qui sont en cours, celles qui sont inactives et celles qui sont potentiellement en train de passer à l'échelle.

Un coût financier trop important peut faire pencher le choix stratégique à plus long terme pour aller porter telle ou telle application vers un développement spécifique interne par exemple. D'où la nécessité de rester dans la maîtrise de ce que propose la plate-forme et ses alentours (cf. développement d'extension). Enfin, ne pas négliger une négociation tarifaire avec l'éditeur adaptée à l'augmentation de vos usages - à date les éditeurs de plates-formes No/Low-Code n'ont pas encore mauvaise réputation sur ce sujet, le marché restant encore relativement ouvert. **Figure 4**

Figure 4

Adéquation d'une solution no-code facturée à l'utilisateur suivant les cas d'usage

	Une seule application	Multiples applications sur la même plateforme à l'échelle de l'entreprise
Utilisateurs anonymes	+ Les utilisateurs anonymes ne sont pas facturés	- Les contraintes liées à une mutualisation ne sont pas justifiées par un gain économique en licence
Quelques utilisateurs authentifiés (~10 à 20)	+	+
Nombreux utilisateurs authentifiés	- Le coût à l'utilisateur rend la solution très onéreuse	+ La répartition du coût de la licence utilisateur sur plusieurs applications abaisse le coût marginal utilisateurs X application

La dimension humaine

La dimension humaine est primordiale en informatique. Dans le domaine du No/Low-code, et particulièrement dans le contexte d'une entreprise, le développement de l'utilisation du No/Low-Code passe par une formation des personnes. Certaines solutions sont simples (en général les moins riche en termes de fonctionnalités/possibilités) comme Wix d'autres comme Bubble où les plates-formes Low-Code demandent plus d'apprentissages. Il existe de nombreux modules/tutoriaux en ligne que les éditeurs mettent à disposition et des organismes dispensent des formations de découverte des outils No-Code.

Les plates-formes Low-Code fournissent aussi des tutoriaux, mais ils s'adressent à des personnes qui ont un minimum de compétences ou à minima une appétence en informatique et nécessitent plusieurs semaines de formation.

Côté DSI, des formations et accompagnements des collaborateurs peuvent être dispensés pour acculturer le reste de l'entreprise ou faire monter en compétence des collaborateurs sur le développement Low-Code en général et sur un outil choisi en particulier. Cela permet ainsi de cadrer les usages qui pourraient en être faits dans le contexte de l'entreprise. La DSI peut aller un cran plus loin et apporter du support et/ou proposer une nouvelle filière de développement au reste de l'entreprise. Cette dernière proposition est celle que nous préconisons. Bien qu'elle introduise une nouvelle dépendance vis-à-vis de la DSI, les ressources que l'on peut déployer sur ce type de solutions sont moins exigeantes en expertise informatique et peuvent ainsi être supportées par des profils plus variés dans l'entreprise. Le préalable étant qu'ils soient acculturés puis formés au Low-Code bien entendu !

Conclusion

La mise en œuvre de solutions No-Code/Low-Code ouvre un nouveau champ des possibles dans le développement d'applications. Ces technologies rendent possible la réalisation d'applications auxquelles les technologies de développement classique ne répondent pas pour des raisons de coût, de délais ou tout simplement de disponibilité de main d'œuvre qualifiée. Ces solutions n'en sont pas moins magiques et nécessitent une expertise adaptée pour leur mise en œuvre réussie, du choix de la solution, des projets à implémenter à leur réalisation.

Références

Blog Octo

https://blog.octo.com/le-low-code-comment-ca-marche/

https://blog.octo.com/no-code-low-code-les-trois-raisons-de-sy-mettre/

https://blog.octo.com/les-competences-it-sont-de-plus-en-plus-rares-et-cheres-et-si-vous-osiez-le-no-code-low-code/

https://blog.octo.com/no-code-ou-low-code-pour-une-application-de-gestion-developpee-avec-airtable-zapier-that-is-the-question/

https://blog.octo.com/les-dix-commandements-dune-plateforme-no-code-mature/

https://blog.octo.com/les-fake-news-du-low-code-compte-rendu-du-talk-alain-faure-et-sylvain-fagnent-a-la-duck-conf-2021/

https://blog.octo.com/culture-innov-osez-le-code-jetable/

https://blog.octo.com/outils-no-code-et-low-code-la-baguette-magique-de-ceux-qui-ne-codent-pas/

https://blog.octo.com/interview-alain-faure-et-dominique-lequepeys-lavantage-principal-du-no-code-low-code-est-de-democratiser-le-developpement-dapplication/

Duck Conf : https://www.youtube.com/watch?v=sQZqyB-EZro

Formations OCTO Academy

https://www.octo.academy/formations/formation/79/developper-une-application-sans-savoir-coder/

Guide autour de l'outillage No-Code : https://quels-outils-nocode.fr/

Podcast : https://podcasts.apple.com/us/podcast/radio-contournement-le-podcast-no-code/id1474485339

De l'usage aux compétences : quelles sont les spécificités du low-code ?

Simon Campano

Après sa formation à l'EPF et un passage en SS2I, Simon rejoint Simplicité en 2016 en tant qu'ingénieur solutions pour accompagner les clients et partenaires dans leur évolution numérique en les aidant à concevoir, construire et renforcer leurs systèmes et leurs équipes à l'aide de la plate-forme low-code Simplicite

Le marché des plates-formes low-code et no-code ne cesse de croître et de nombreuses entreprises ou institutions publiques font le choix de les adopter pour la création de leurs applications métiers. De multiples raisons expliquent cela : la pénurie des compétences techniques couplées avec un besoin croissant de transformation numérique des usages en fait partie. Les pures players historiques (comme Simplicité, Outsystem, Mendix) sont rejoints par des éditeurs dont le cœur de métier est tout autre (Microsoft, ServiceNow, Pega system…).

Au commencement il y avait les cartes perforées, l'assembleur, puis ont suivi les langages 1er, 2 , 3 et 4 générations… En rajoutant des couches d'abstractions à chaque fois. Les plates-formes low-code sont dans la continuité de cette évolution naturelle de l'informatique et de la technologie.

La création d'une application implique généralement 3 axes principaux :

- **La donnée** qui est, *in fine*, le cœur de l'application (quels sont les concepts métiers qui sont manipulés, typiquement pour créer une application de gestion hôtelière, vous allez avoir le concept de client, de chambre, de moyens de paiements, de réservations, etc. Voir Programmez! 251).
- **La logique métier** qui permet d'exploiter cette donnée, d'en tirer sa substantifique moelle, et bien évidemment de faire fonctionner l'application. La logique métier est constituée principalement des règles métiers (de calcul), des règles de sécurité, des droits d'accès, des workflows métier. C'est la logique métier qui va permettre d'orchestrer les actions métier des utilisateurs sur la donnée métier.
- **La partie interface** / IHM, qui va permettre de créer des vues sur la donnée, de s'attacher à fluidifier le travail des uns et des autres en insistant sur l'expérience utilisateur UI/UX, de restituer les données brutes ou agrégées de façon sécurisée et collaborative, de consulter des tableaux de bord, des cartes, des graphiques…

La plate-forme low-code est un outil "tout-en-un" qui permet de combiner ces éléments fondamentaux pour créer l'application métier qui va répondre à l'exact besoin exprimé. Il est utopique de penser qu'une application d'entreprise complexe pourra permettre de couvrir l'ensemble des besoins et contraintes fonctionnelles uniquement par paramétrage ("à la souris"). C'est pourquoi les plates-formes low-code permettent d'adjoindre du code spécifique qui va spécialiser les comportements génériques et atteindre ainsi la parfaite adéquation de l'application au processus métier instrumenté. On a tendance à dire qu'une plate-forme low-code est une feuille blanche au sein de laquelle on "décrit" tout ou partie d'un métier.

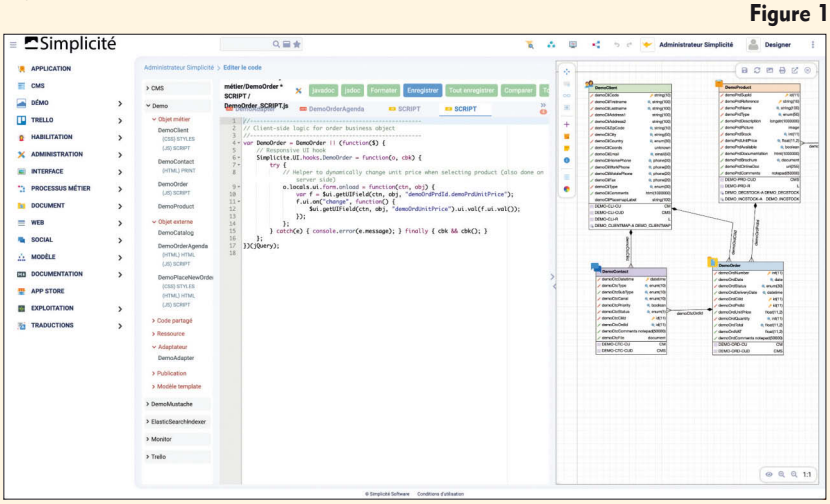
Les plates-formes low-code révolutionnent donc la création des applications métiers tout en s'appuyant sur les techniques, technologies et méthodologies modernes. Elles bénéficient de l'ensemble de l'évolution de l'informatique en général, comme l'avènement du cloud, l'approche DevOps, la conteneurisation (Docker), l'orchestration (Kubernetes) et les méthodologies agiles et CI/CD (intégration et déploiement en continu).

Généralement, le code spécifique ne doit pas dépasser 20 à 30% du périmètre fonctionnel de l'application, pour réaliser par exemple :

- des règles de gestion sur les données pour contrôler, compléter ou calculer,
- des traitements de masse sur les données ou des actions en cascade d'événements IHM ou périodiques / tâches planifiées,
- les connecteurs de données entrantes et sortantes pour adapter les formats avec des systèmes tiers (ex : fichiers CSV, appel d'API tierces, emailing),
- les éditions spécifiques de document (ex : usage d'Excel POI, templating mustache, librairie PDFBox),
- certaines UI et interfaces utilisateurs dédiées à certains usages si les écrans natifs (de liste, formulaire, vue en arbre, kanban, graphique, carte…) ne conviennent pas (ex : afficher un plan 2D avec des bureaux à réserver sur l'étage).

Figure 1

Figure 1



Un bon cas d'usage sera constitué d'une application gérant des objets "assets" (par exemple des dossiers, des personnes, du matériel, des catalogues, factures, etc.) et sur lesquels des utilisateurs vont mener des actions métiers (workflow) en y appliquant des règles métiers spécifiques (sur les données) et des actions en cascade (events ou tâches asynchrones).

Les plates-formes low-code ont vocation à être au cœur du Schéma Directeur de l'entreprise, le référentiel qui décrit comment doivent évoluer les systèmes d'information et aussi toutes les ressources en lien (salariés, logiciels, matériels…) afin d'atteindre les objectifs fixés, et qui permet de supporter :

- Les projets **porteurs d'innovation**.
- La **réduction** de la dette technique,
- Réduire le "Shadow IT" (au passage, il conviendra d'être vigilant à ne pas recréer du shadow IT en laissant les "citizen developers" créer des applications stratégiques avec des outils no-code)
- La réduction de l'hétérogénéité du Système d'Information
- Le contrôle des redondances fonctionnelles
- Outiller une approche "test and learn" itérative et évolutive
- La **réactivité** et **l'agilité d'entreprise** (adapter les solutions numériques aux transformations du métier)

Ici quelques exemples d'applications qui peuvent se faire grâce au low-code : **Figure 2**

Nous parlons bien ici de cas d'usages de création d'applications avec des plates-formes low-code.

Comme dit plus haut, certaines "petites" applications pourraient être réalisées grâce à des outils no-code, mais il y aurait alors un fort risque d'atteindre un plafond de verre infranchissable. En effet, la couverture métier des outils no-code reste par nature restreinte aux fonctionnalités offertes par ladite plate-forme no-code, et il serait alors indispensable de multiplier les apps pour couvrir un besoin complet, ce qui engendrerait une complexité extrême en termes d'interconnexions, de maintenance et d'évolutivité. Le low-code permet d'aller au-delà du plafond de verre, mais nécessite un peu de développement. Étendre avec du code à la main en somme.

Figure 2

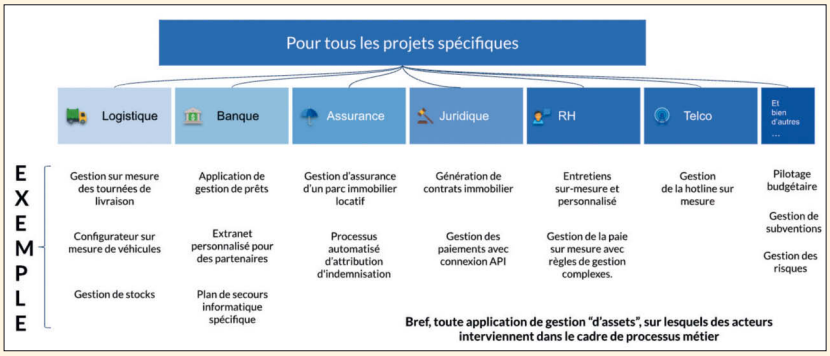
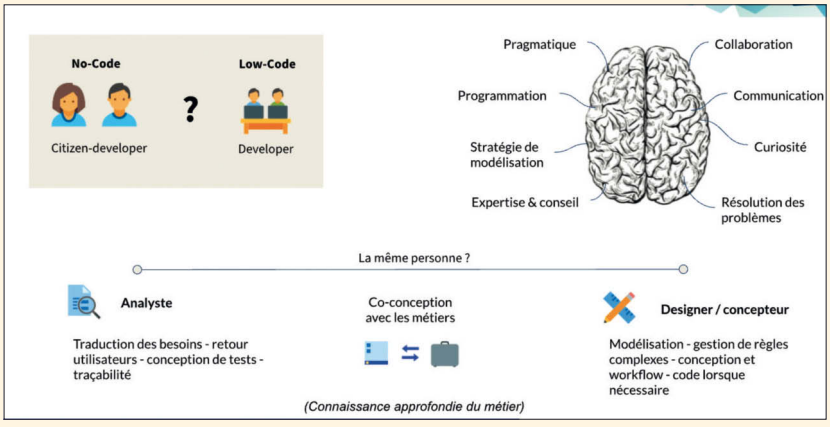


Figure 3



Il y a des cas d'usages où l'utilisation d'une plate-forme low-code n'est pas optimum

En revanche, si votre application peut déjà être achetée "sur étagère" et que cette dernière répond à 100 % aux exigences dont vous avez besoin, alors il est conseillé d'acheter ce produit sur étagère. L'achat d'une telle solution est souvent plus rentable et plus rapide à mettre en œuvre.

En général, ces solutions prêtes à l'emploi sont disponibles pour les applications traditionnelles comme la comptabilité générale ou la paye (métiers qui n'évoluent pas beaucoup), car les processus sont normalisés et bien établis, communs à la plupart des organisations.

Usage approprié pour le low-code*	Peu approprié au low-code*
Tous les workflows d'activité de gestion de données / Listes et formulaires / Tableaux et reporting opérationnel / Publication / Gestion de droits complexes, avec descomposantes non triviales : intégration avec des systèmes tiers / API, authentification, génération de docs, besoin d'outillage spécialisé inexistant sur le marché	Des applications comme le "temps réel", le décisionnel "pur" sur de grosses volumétries, la robotique ou le pilotage de machines-outils, la GED "pure", l'échange de données entre sous-systèmes (type EAI / ETL / ESB), les applications de gestion normalisées / solution sur étagère

**pour la plate-forme Simplicité*

Les bonnes compétences, qui peut vraiment adopter du low-code au quotidien

L'approche low-code nécessite de changer de paradigme. Le développeur tel qu'il était connu il y a quelques années doit (s'il le souhaite) modifier, adapter sa façon de pratiquer son métier. **Figure 3**

Avant toute chose, il convient de bien définir ce qu'est un développeur low-code. Chez Simplicité, la notion de développeur n'est pas très utilisée, car trop partielle. Nous préférons parler de **Designer**.

Un Designer Simplicité est un concepteur qui sait écouter, comprendre et modéliser un besoin métier avec des carrés et des flèches façon UML, le mettre en œuvre par du paramétrage et du développement graphique (par glisser/déposer), et écrire du code à valeur ajoutée pour réaliser les traitements et règles très spécifiques en complément du paramétrage. Enfin, si cette même personne connaît les architectures cloud, la chaîne CI/CD et la sécurité… c'est le mouton à 5 pattes tant espéré ! Vous l'aurez compris, un bon développeur ou un designer au sens Simplicité est un profil multi-compétent capable d'évoluer d'un profil de codeur à celui d'architecte logiciel grâce à une plate-forme qui lui facilite l'apprentissage sur l'ensemble de ces sujets.

Dans la vraie vie, en fonction de la complexité et de l'envergure de l'application métier et de l'équipe projet à mettre en place (Cœur SI par exemple), ces rôles seront

attribués à des profils différents et experts dans leurs domaines. La "clé de voûte" de l'équipe sera alors le binôme business analyste//développeur, le business analyste ayant l'expertise pour modéliser et paramétrer l'application "à la souris" et atteindre ainsi 70 à 80 % de couverture métier ; le développeur concentrant son effort sur le code à valeur ajoutée pour spécialiser les comportements paramétrés au préalable par le Business Analyste et atteindre ainsi 100 % du besoin.

Le profil "pur développeur" va ainsi développer (en langage usuel java/SQL pour la plate-forme Simplicité) la logique métier avec une réelle valeur ajoutée. Nous avons tendance à dire que 80 % d'une application sera du paramétrage ; 20 % du code spécifique.

Mais attention, cela ne veut pas dire que le temps passé par la personne qui développe suivra le même pourcentage. Même si c'est très exagéré, les 20 % du code spécifique d'une application pourront représenter 80 % du temps de création de l'application cible. Par contre, ce qui est à peu près certain, c'est que les 20 % de code spécifique pourront représenter jusqu'à 80 % de la "valeur métier" et qu'il n'y a plus, dans une application low-code, de code sans valeur ajoutée métier (par exemple re-coder systématiquement les paginations, la gestion de droits, les fonctions CRUD, les menus, etc.). **Figure 4**

L'appropriation de l'approche low-code par les équipes IT s'appuie donc sur un certain nombre de compétences élargies et le rôle de designer low-code combine donc des savoir-faire multiples, ce qui permet notamment de réduire de manière drastique la taille des équipes impliquées dans ce type de projets.

Un bon "designer" low-code sait :

- **comprendre** un besoin métier
- le **modéliser** sous forme de "carrés et de flèches" (UML)
- réaliser une programmation visuelle, tant du modèle que des écrans de restitution générique, via un modeleur graphique et des fonctions de drag/drop pour la création des écrans
- surcharger le comportement des objets métier par du code utile
- s'intégrer dans une chaîne CI/CD (ou DevOps)
- posséder des bonnes notions d'architecture IT (cloud, Docker, Git, K8s…)

Il est à noter que les plates-formes low-code qui, comme Simplicité, s'adressent à des professionnels de l'informatique et s'inscrivent bien évidemment dans les standards d'architecture et les cadres de référence techniques des clients. Il ne s'agit pas ici de produire, contrairement aux outillages no-code et/ou à destination des citizen developers, du shadow IT à tout va. La plate-forme se doit d'être maîtrisée par le SI et ainsi ne pas occulter les problématiques de scalabilité, d'opérabilité, d'administration et d'exploitabilité.

Apports de valeur du low-code :

- **Plus vite** : réactivité, productivité.
- **Plus efficace** : maintenabilité, fonctionnalités, mutualisées, fiabilité.
- **Différemment** : agilité, prototypage>implémentation, métiers mieux associés.
- **Maturité et adaptabilité** de la Plate-forme.
- **Donner du sens** : le développeur comprend ce pour quoi il travaille. Il se concentre sur l'essentiel : le développement de composants métiers.
- **Évolutivité** et maintenance simplifiée
- **Qualité** du code par l'incitation aux bonnes pratiques

Failles de sécurité pré-protégées

- **Non-adhérence** aux technologies front à durée de vie éphémère
- **Uniformisation** UX

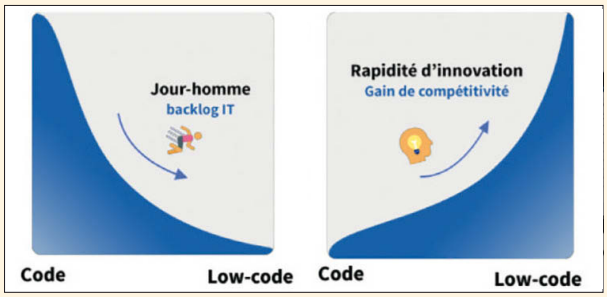


Figure 4

Facteurs clé de la réussite d'un projet low-code :

- Mettre en place une organisation projet adéquate. La **constitution** d'une bonne équipe est primordiale, notamment l'AMOA SI pour « cadrer » le métier (rajout du rôle du Designer, il sait changer de casquette)
- Impliquer **l'ensemble des acteurs** sur tout le cycle de vie du projet (design, build, run, operate et itérer). Les métiers doivent dégager du temps et s'impliquer, notamment pour recetter en "temps réel" l'application en cours de création (méthode agile)
- **Ne pas négliger** la montée en autonomie. Même si elle est rapide, certains concepts de base sont à intégrer et les plates-formes low-code offrent un très grand nombre de fonctionnalités natives qu'il convient d'appréhender au fil de l'eau (par exemple, Simplicité a plus de 15 ans de R&D derrière elle et d'ajout de fonctionnalités dirigés par l'évolution des technologies et les demandes clients)
- Se projeter le plus tôt possible vers **la phase de RUN pour aller vers la cible idéale**
- Mettre en place une organisation **centrée sur la donnée**
- Privilégier au maximum l'usage des standards de la plate-forme : usages génériques. Les IHM proposées nativement par les plates-formes low-code ont fait l'objet de travaux UX/UI et sont donc adaptées à un usage métier. Certaines concessions peuvent être à faire par les métiers afin de ne pas pénaliser le ROI de la création de l'application par des exigences qui impliquent trop de développements en dehors "du standard".

Comment choisir sa plate-forme low-code :

- Maturité de la plate-forme : Une plate-forme "mature" a eu le temps de rassembler une grande communauté d'utilisateurs/développeurs.
- Installation et déploiement rapide
- Capacité à s'adapter à la charge
- Capacité à s'adapter à l'évolution des technologies
- Temps nécessaire à la maîtrise

Exemple de plates-formes en fonction de la complexité			
Type de plate-forme	No-code	low-code for business developer	low-code for IT Professionals
			Simplicité Software
Verbatim	*"Je n'ai pas de culture informatique, je choisis des solutions "prêtes à l'emploi" avec un minimum de paramétrage et disponible en mode SaaS. Il m'en faut une pléiade à assembler pour répondre à mon besoin métier. Je devrais ensuite professionnaliser et basculer à la DSI"*	*"Je maîtrise parfaitement les outils bureautiques, j'ai des compétences en modélisation et je n'ai pas de problème à faire un peu de code, j'ai besoin d'un outil qui me permette d'être autonome pour automatiser des workflows simples ou créer des applications de partage d'informations avec mes collaborateurs ou mes clients internes. J'ai besoin de sécurité et il faut que mes outils soient intégrés avec les annuaires d'entreprise pour gérer les accès. Je travaille en étroite collaboration avec la DSI."*	*"Je n'arrive plus à répondre aux besoins de mes métiers dans des coûts et délais raisonnables avec mes méthodes et outils actuels. J'ai des problèmes de recrutement, car il me faut des spécialistes de chaque domaine (Front, Back, SGBD, BigData, Architecture, Sécurité...). Je travaille sur des technologies en obsolescence et je ne peux plus assumer ma dette technique.J'ai besoin d'une plate-forme industrialisée, fournissant une couche d'APIs générique et qui gère tout le cycle de vie. Je forme facilement mes collaborateurs et/ou je fais appel à mes partenaires. J'inscris cette plate-forme dans ma stratégie SI à moyen/long terme."*
Niveau de compétences techniques	+	++	+++

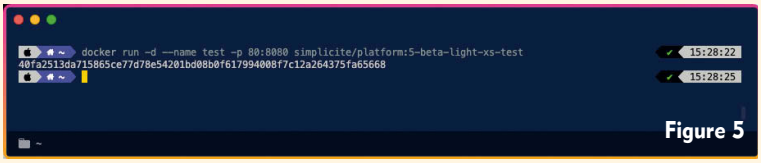


Figure 5

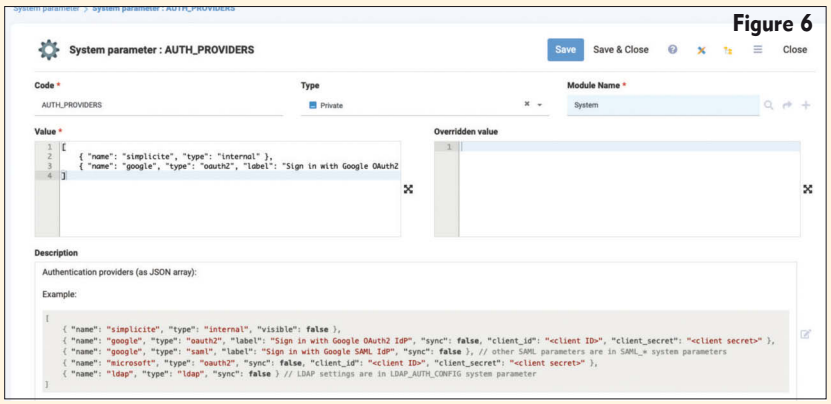


Figure 6

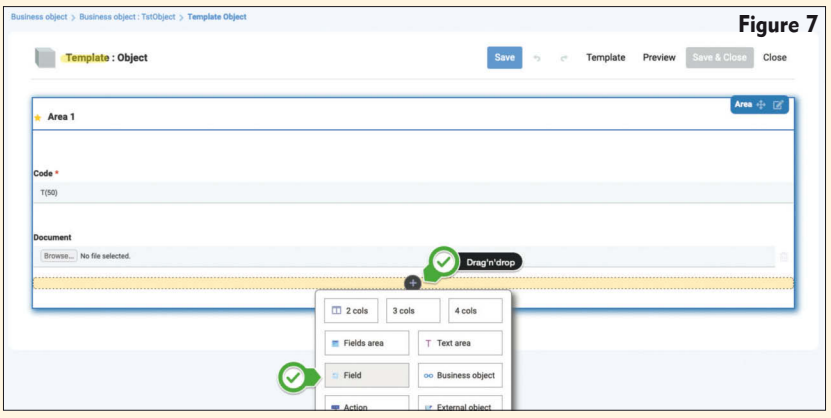


Figure 7

Code vs Low-Code : quelles différences ?

Quelle est la différence en termes d'expérience pour un développeur low-code et un développeur "classique" ? Va-t-on moins coder ? Passer son temps à cliquer partout pour configurer des composants rigides et frustrants ? Pas si sûr ! Le principe est de privilégier la configuration (pérenne, uniforme, standardisée), tout en laissant une grande souplesse, à la fois en extension comme en modification de la plate-forme.

Envisageons un parcours de développement pour une application simple à 1 objet métier avec un champ Code et un champ Document et 1 règle de gestion spécifique au métier : "la valeur du Code doit faire partie d'un référentiel accessible par API REST".

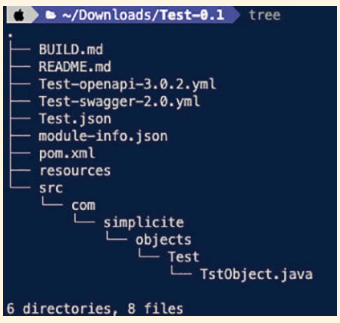
En développement classique, il faudra :

1. choisir les technologies back et front
2. installer et configurer des outils de développement
3. faire les développements :
   - du login/logout via authentification OAuth2 à l'application
   - de la connexion à la BDD + création de la table + indexes
   - du front+back de la vue en liste + pagination + fonction de recherche
   - du front+back de la vue en formulaire + contrôles + boutons d'action
   - de la règle de validation spécifique au métier
4. tester et déployer l'application
5. administrer les droits
6. auditer les usages et les performances

Si l'on compare au parcours d'un développeur low-code pour la même application :

1. déployer un conteneur docker Simplicité. **Figure 5**
2. configurer
   - l'authentification, via saisie d'un JSON en paramètre système. **Figure 6**
   - l'objet métier via le Template Editor. **Figure 7**
3. développer directement dans l'interface web
   - la règle de validation spécifique au métier, en Java **Figure 8**
4. tester et déployer. **Figure 9**
5. Administrer les droits **Figure 10**
6. Auditer les usages et les performances **Figure 11**

Cet exemple fonctionnel (hors fausse URL du webservice) a pris **5 minutes** entre la création de l'instance de développement avec Docker et le code de la règle spécifique. C'est certes un exemple, mais on a déjà une application complètement fonctionnelle, et hautement extensible ! Pour archiver ça dans un GIT, on peut utiliser des mécanismes inclus dans la plate-forme. En l'état, on constate que le contenu suivant a été généré et archivé :



C'est un projet java complet, avec notre fichier java et un **fichier maven "pom.xml"** qui permet par exemple de :

- faire tourner des **tests sonar**,
- **développer dans son IDE** préféré, avec son autocomplétion, son copilote,
- utiliser un **debugger** pour faire du pas-à-pas.

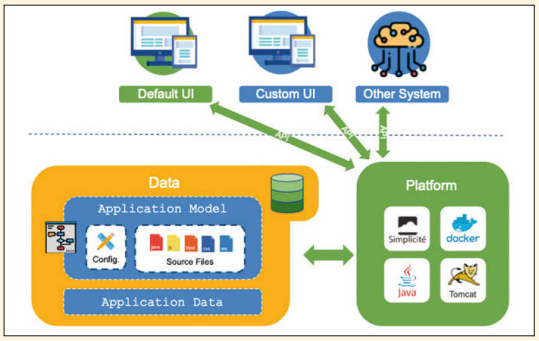
On constate qu'il y a même les descriptions OpenAPI et Swagger des API qui sont par défaut mises à disposition (avec la sécurité qui va bien) pour notre objet métier. À coût 0…

Le fichier "Test.json" est la **configuration** qui contient **tout ce qui a été paramétré** via les wizards et l'éditeur de formulaire :

- les composants du modèle : l'objet métier et ses champs,
- les habilitations : le groupe d'utilisateur et ses droits sur l'objet,
- les définitions graphiques : logo, thème, menu
- les variables d'environnement : l'URL du service à appeler
- la localisation : traductions en français et en anglais de tout ce qui précède

Ce fichier JSON fait ~700 lignes et peut aussi s'exporter sous forme d'une hiérarchie de fichiers de configuration unitaires qui facilite les diff et les merge et qui évite les conflits git.

On passe moins de temps à se poser les questions / développer les interfaces d'UX déjà résolues par les patterns qui ont fait leurs preuves, et plus de temps sur la partie la moins répétitive du développement web : les besoins vraiment spécifiques au métier.



La résultante est une modélisation du métier interprétée dynamiquement par la plate-forme pour :

- exposer les API correspondantes
- servir une UI générique d'accès en fonction des droits
- gérer la base de données et les documents

En conclusion : le low-code est une évolution du métier de développeur, mais il **n'est pas magique**. Il faudra toujours des développeurs !

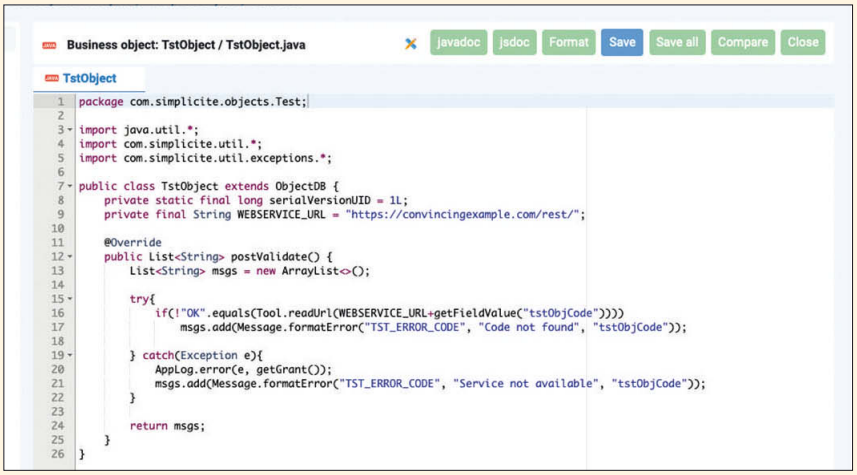


Figure 8

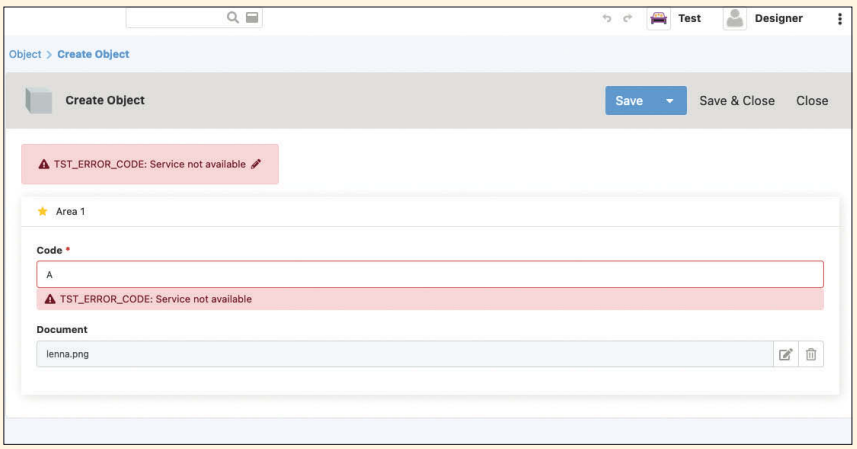


Figure 9

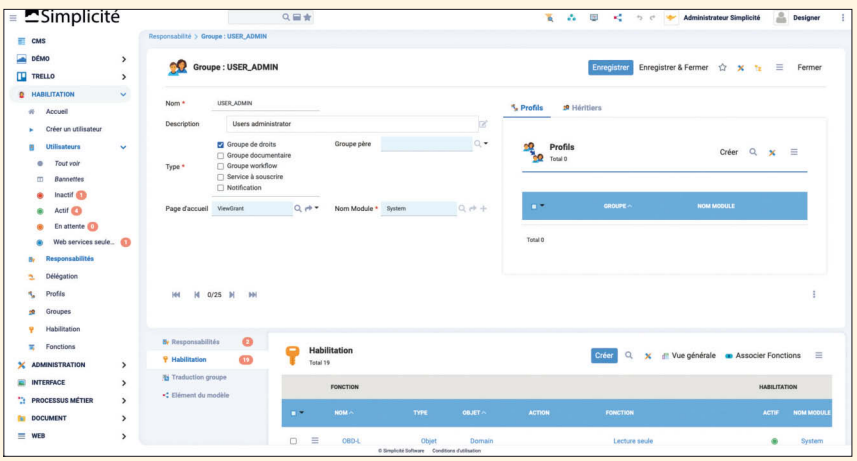
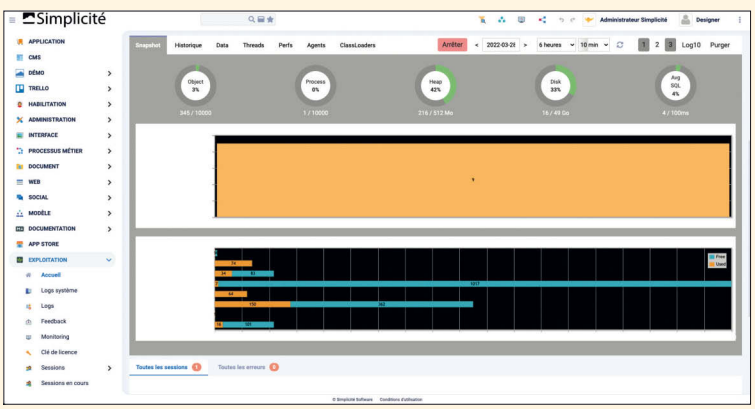


Figure 10

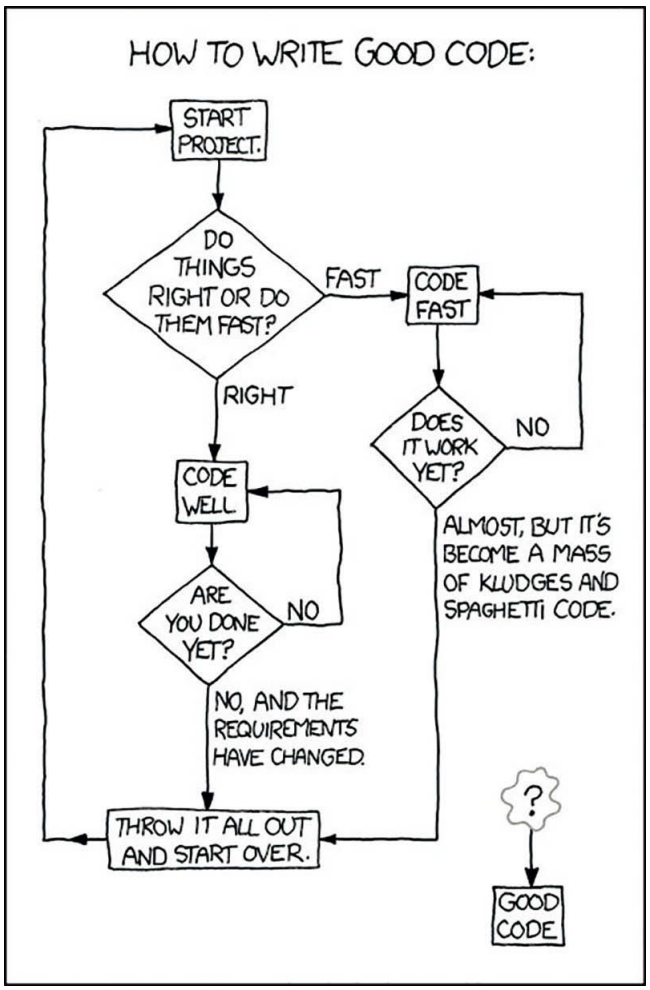
Figure 11



Frédéric Fadel
Technical Angel : More Software Less Code
Aspectize

Coder : passionnément, beaucoup, un peu ?

Quand François m'a demandé d'écrire un article sur le « Low Code », catégorie mal comprise et mal assimilée, je me suis dit qu'au lieu de parler de la technologie d'Aspectize qui peut se classer comme Low Code au sens large (puisqu'on écrit peu de code), il valait mieux élargir le sujet et parler de Low Code sous l'angle de tout outil, approche ou idée visant à réduire le code - qu'ils soient officiellement classés sous la catégorie Low Code ou non.



La fonction de la manivelle avant d'être automatisée par le démarreur.

Il y a beaucoup de sortes de code et de codeurs, chacun ayant une vision de son métier et de son code. Ces métiers et/ou visions peuvent influencer l'appréciation voire le rejet du Low Code.

Je code depuis plus de 40 ans, de la carte perforée au JavaScript en passant par l'assembleur et le débogage au voltmètre. Mon intérêt pour le logiciel n'a cessé de croître en même temps que mon aversion pour le code. Je trouve qu'à résultat égal, on a toujours intérêt à écrire le moins de code possible.

Développer un logiciel est un travail intellectuel de création : c'est comprendre un problème dans un cadre précis, c'est concevoir d'une solution acceptable et finalement c'est coder, c'est-à-dire transcrire le résultat de ce travail sous une forme exécutable par une machine.

Ces trois « activités » sont entremêlées. On ne finit pas la première pour commencer la deuxième et il se peut que la troisième aide à accoucher la première. Souvent, la difficulté principale est d'exprimer le besoin, la douleur qui est censée être calmée par un bout de logiciel. C'est là que le Low Code est utile. S'il faut écrire beaucoup de code, passer beaucoup de temps avant de voir un résultat appréciable, on ratera le tâtonnement nécessaire pour comprendre le besoin et imaginer la solution.

Le développement logiciel est un domaine vaste, c'est probablement le domaine le plus vaste et qui a des applications partout. Il touche d'une manière ou d'une autre quasiment tout et tout le monde. C'est pourquoi c'est aussi un domaine qui est sujet à la mode : il « devrait » être l'un des plus « exacts », mais il évolue avec beaucoup de mythes encombrants qui impactent les pratiques, les choix et les habitudes de développement. C'est un domaine avec des croyances, des camps, des fiertés déplacées, ce qui ajoute à la complexité inhérente au développement logiciel.

Qu'est-ce que le Code ?

On utilise les deux verbes « coder » et « programmer » de façon interchangeable. Ils évoquent néanmoins des aspects différents du job.

Coder c'est numériser, remplacer des actions, des attentes, de l'information par un code, des suites de nombres, des suites d'instructions.

Programmer c'est répéter une action, comme programmer le réveil pour qu'il sonne tous les jours à une heure précise sauf le week-end.

Les deux sens ont à la fois un aspect nécessaire et rébarbatif qu'on aimerait diminuer, voire éliminer.

Le code en informatique ressemble aux frottements en mécanique. Sans frottements aucun mouvement n'est possible, en même temps les frottements empêchent le mouvement et dans tout système mécanique on cherche à éliminer les frottements.

Sans code aucun logiciel n'est possible, mais le code est difficile et coûteux à écrire, il est encore plus difficile et coûteux à maintenir et à faire évoluer, on aimerait en écrire le moins possible.

Quand on a commencé à coder - je parle des premiers qui ont aligné des zéros et des uns dans une mémoire électrique -, on était envoûté par ce qu'on arrivait à accomplir, des additions, des multiplications, des calculs, tous les calculs possibles et imaginables, les mouvements des planètes, les prévisions météo, des tris, des classements, des automatisations de tâches répétitives.

Il se trouve que le processus de codage comporte lui-même beaucoup de tâches répétitives, c'est pourquoi depuis toujours on a développé des outils qui aident à coder moins et mieux. On a automatisé le codage, on a inventé des compilateurs et des langages de plus haut niveau.

On écrit moins de code en C qu'en Assembleur, on écrit moins de code en C# qu'en C, et on se porte très bien, peu de lecteurs de cet article s'offusqueront de ne pas coder en Assembleur. Pour moi le compilateur C# permet - en quelque sorte - de coder « moins » et « mieux » qu'en C ou en Assembleur. D'autres diront – probablement avec raison - qu'on code « moins » et mieux en F# qu'en C#.

Dans une autre catégorie, on peut citer le moteur de base de données relationnelle. Grâce à ce moteur, on peut trouver avec un code SQL ultra réduit une information sur un disque dur, alors que si on devait écrire l'équivalent du code en C# avec des classes d'E/S du système pour trouver la même information, le volume de code serait rédhibitoire et insupportable .

Dans le domaine précis de recherches sur disque d'informations structurées, le moteur SQL et le langage SQL permettent d'écrire beaucoup moins de code. Peu de gens aujourd'hui s'offusqueraient d'utiliser un moteur SQL en prétendant qu'ils feraient mieux en s'occupant des E/S directement. Néanmoins, certains le font et créent régulièrement de nouveaux outils de stockage, de recherche et d'indexation et c'est très bien. C'est peut-être le signe que les outils existants ne couvraient pas tous les besoins, mais ça peut aussi être un signe de l'envoûtement du créateur par sa création, l'envie de créer un truc nouveau juste pour le plaisir.

Un système d'exploitation ou un navigateur Web sont des environnements qui permettent de réutiliser du code écrit par un autre d'une façon transparente, de telle sorte qu'on développe aujourd'hui des logiciels - à résultat égal - avec beaucoup moins de code qu'il y a 50 ans.

Dans un certain contexte, Excel est l'outil de gestion ultime. Sans écrire de code, on obtient avec Excel des résultats extraordinaires, des tableaux, des calculs, des graphiques. Un outil comme PowerBI permet d'aller au-delà d'Excel et de présenter un rapport dynamique sur une page Web en quelques clics - sans coder - en se connectant à une source de données presque quelconque. Officiellement, tous les « outils » que je viens de citer ne sont pas classés dans la catégorie « Low Code ». Ils sont dans des catégories couramment utilisées, néanmoins de fait ils contribuent aussi à écrire moins de code. C'est le sens de l'histoire, le but est de développer des logiciels utiles, pas d'écrire du code.

Il existe des outils qui facilitent le codage et améliorent le confort du codeur, comme un « designer de formulaire » qui génère automatiquement le code - sans valeur ajoutée de comptage des largeurs et hauteurs des éléments en pixels - pour positionner les éléments d'IHM. Ils sont bons à prendre, mais à mon sens un générateur de code est un outil de confort et pas un outil d'élimination de code – puisqu'il en génère.

Un générateur de code – même s'il tombe sous la classification « Low Code » – ne présente pas à mes yeux les avantages requis de la réutilisation, sauf peut-être quand il est garanti que le code généré ne risque pas de se mélanger avec du code écrit à la main, parce que sinon les problèmes de maintenance et d'évolutivité sont aggravés. C'était la principale raison de l'introduction des classes partielles en .net, garantir que le code généré par un robot ne se mélange pas avec le code écrit par le développeur.

Qu'est-ce qui est réutilisable et favoriserait le Low Code ?

C'est difficile de répondre, mais en observant l'évolution de nos environnements et architectures on imagine des pistes de réponses.

Aujourd'hui peu de développeurs ont besoin de connaître les méandres de SMTP, POP3 et d'IMAP pour envoyer un mail puisqu'on a réussi par architecture, normes et standards, de faire en sorte que l'envoie de mail se résume au strict minimum : une appelle d'API.

Un bout de code qui se trouve dans quasiment toute les applications et services c'est le code qui s'occupe de l'authentification et de la sécurité elle est un peu *touchy* mais une fois qu'on sait faire on passe son temps à le refaire, il ne contient pas beaucoup de créativité, on n'a pas besoin de le réécrire N fois. Je peux facilement imaginer que ce code soit éliminé de celui de l'application et soit fourni par un outil ou une technique Low Code, voire No Code (s'il est intégré dans le système d'exploitation – comme un driver d'imprimante)

Dans un contexte applicatif classique, une bonne partie du code lié à l'accès aux données : lecture, écriture, transport peut s'automatiser et être réutilisable pour le codeur.

Au niveau des IHM il y a de la créativité, de l'esthétique et de l'expérience utilisateur à imaginer, mais il y a aussi beaucoup de code lié à l'affichage et la validation des données qui peuvent être complètement automatisées.

Bref tout bout de code pour lequel un best practice est connu, mérite d'être rendu réutilisable pour que tout le monde puisse en profiter.

Dans l'environnement .net, des choses sont réalisées avec cet état d'esprit, par exemple grâce au CTS (Common Type System) un programmeur .net qu'il programme en C#, en F# ou VB.NET réutilise le même code pour trier une liste de chaîne de caractères.

La réutilisation - souvent d'un code écrit par un autre - est la clé principale pour écrire moins de code. Un outil, une architecture, un langage, un environnement, des ruses et astuces de codage qui permettent de mieux réutiliser et d'écrire moins de code sont toujours bons à prendre.

L'interfaçage entre le code écrit à la main et le service rendu par le code réutilisable est un sujet important. On n'aimerait pas perdre en complexité ou contrainte d'interfaçage ce qu'on a gagné en évitant le code. La réutilisation d'un code peut être réalisée soit explicitement soit implicitement. Les ruses et techniques AOP (Aspect Oriented Programming) permettent de favoriser la réutilisation implicite qui je pense contribue à mieux réaliser les promesses du « Low Code ».

Qui est le codeur ? Pourquoi rejetterait-il le Low Code ?

Il y a le codeur qui adopte Wix ou WordPress pour développer un site Web ou une petite application d'e-commerce, son état d'esprit axé sur le résultat final, plus vite il finit, plus vite il sera

> *« Pour bien coder, mieux vaut coder peu. La sobriété du code fait partie de sa qualité.*

payé. Les considérations techniques de codage et de langage, à tort ou à raison, ne font pas partie de ses préoccupations premières.

Il y a aussi le codeur qui met sa fierté dans le code et cela peut se comprendre : on est fier ou simplement content quand on résout un puzzle, quand on résout le Rubik's Cube.

Le codeur aime résoudre des puzzles, coder c'est un peu ça - des mystères abscons, des bugs à trouver, les circonvolutions complexes du code à décoder -, ça amuse au début, mais une fois qu'on sait résoudre le Rubik's Cube, une fois qu'on a atteint le record de la vitesse qu'on souhaite, continuer à jouer au cube devient lassant pour beaucoup.

Un jeune codeur, avant d'être lassé, veut coder et il a raison - il apprend son métier -, il rejette donc l'idée de coder moins. Plus il code, plus le puzzle et sa solution sont satisfaisants.

Plus tard, quand il sera lassé, il cherche un autre métier pas trop éloigné du code, il peut conseiller, former, coacher, gérer, bref il arrête de coder. Il croit avoir fait le tour du sujet et entretient une nostalgie de ses premiers moments d'envoûtement.

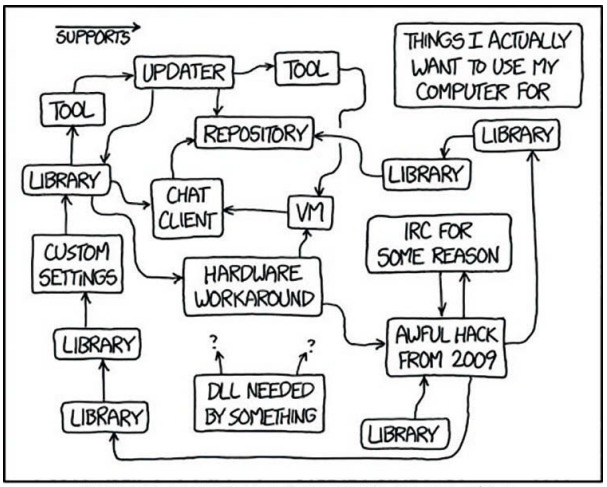
Il peut aussi ne pas se lasser et vouloir coder. Il pourra mettre sa fierté dans une forme de compétence, une expertise pointue.

Par exemple, en .Net si on développe des logiciels dans le domaine des opérations financières, on a besoin de bien maîtriser la mémoire et les mécanismes du Garbage Collector, parce qu'on ne veut pas être interrompu 200ms à un moment critique.

Quand on atteint une telle expertise, ça tombe sous le sens de se méfier d'un outil Low Code puisqu'on n'aura pas la garantie que l'outil gère la mémoire comme on le souhaite, sauf bien sûr si l'outil est justement fait pour le cas précis de la gestion de mémoire.

Le codeur peut vendre son temps et pas un résultat, il peut travailler pour une société qui vend du temps et pas un résultat. L'équation économique devient donc plus complexe, et ça pourrait tomber sous le sens de ne pas vouloir développer plus vite.

Il y a de bonnes raisons de ne pas vouloir utiliser un environnement dit de Low Code : souvent des caractéristiques techniques très pointues et particulières, mais il y a beaucoup de mauvaises raisons aussi : fierté mal placée, piètre calcul économique, décision d'intérêt politique.



Quel codeur, quelle entreprise a intérêt à adopter le Low Code ou le No Code ?

C'est difficile de répondre, je peux parler de mes choix et des choix de mes clients. En écrivant moins de code, en réutilisant un code écrit dans un *framework* spécifique pour un besoin spécifique, on améliore la vitesse de développement, on augmente son agilité puisqu'on s'occupe moins de détails techniques et plus - voire uniquement - des besoins métier. Mais en même temps, on perd un peu le contrôle et on doit faire confiance au *framework* de faire son job correctement.

In fine, je crois que les contraintes économiques et les besoins en matière de réactivité du développeur doivent permettre de répondre à la question.

On peut toujours faire mieux avec plus d'argent, encore faut-il savoir si ce mieux vaut le coup. On peut aussi souhaiter plus de contrôle et de maîtrise du code, encore faut-il savoir si on dispose du temps nécessaire et si ce niveau de contrôle est souhaitable ou pas.

Le contrôle total et le budget infini n'étant pas des options très courantes, on doit souvent opter pour un équilibre entre qualité, prix et fonctionnalité. C'est dans la recherche d'un tel équilibre que le choix d'une technologie Low Code ou No Code devient pertinent.

Si la motivation du développeur est de faire plus de métier et de passer plus de temps dans la compréhension des besoins de son client, plutôt que dans le codage technique, il optera pour un outillage Low Code. Si au contraire sa motivation est de relever des défis techniques ou d'utiliser et de tester la dernière techno à la mode, il n'optera probablement pas pour des outils Low Code.

Si le développeur se trouve dans un environnement où il a accès aux utilisateurs et où il peut démontrer sa réactivité en livrant vite et directement, il profitera grandement d'un outil Low Code. Si au contraire, il évolue dans une hiérarchie qui produit et consomme péniblement des cahiers des charges, l'intérêt du Low Code sera nul.

Pour un développement pérenne qui nécessitera des évolutions et des mises à jour régulières, un outil Low Code apporte un avantage concurrentiel pour un développeur seul ou une petite équipe. Un outil Low Code peut aussi être utile pour un développement jetable, pour faire des POC dans le cadre de développements colossaux.

L'un de nos clients, une petite société de service, a eu l'ambition de s'attaquer au développement d'un SI Achat pour un gros client. Dès les premières phases d'études, il était clair que sans outil d'aide au développement, ils n'entreraient ni dans les temps ni dans le budget. Ils ont fait confiance à la technologie d'Aspectize et ont relevé le défi.

La jeunesse aime inventer, innover, apprendre, et le jeune informaticien code passionnément et produit beaucoup de code. Mais la satisfaction du métier est surtout dans… l'utilité du logiciel qui remplit sa fonction, en temps et en heure. Pour bien coder, mieux vaut coder peu. La sobriété du code fait partie de sa qualité.

Oracle Visual Builder : une app Web en quelques minutes

Marc Gueury

Oracle Domain Architect - AppDev - EMEA

Domain Architect dans l'IT, travaillant et conseillant les entreprises sur les meilleurs moyens de développer ou d'intégrer de nouvelles technologies dans les systèmes existants. Certifié dans un grand nombre de technologies : Dev (Java, Kubernetes, PLSQL, SQL), Cloud (OCI, AWS, Azure), Oracle (DB, SOA, Weblogic), IT (Scrum, TOGAF, Prince2, ITIL), Autres : (CCSK, AI/NLP, ...)

À côté de l'outil Low Code bien connu Oracle Apex (composant de la base de données), Oracle fournit un logiciel de développement rapide Visual Builder basé sur du HTML, du JavaScript et du CSS

Visual Builder est un outil Low Code. Il utilise une interface simplifiée qui permet de créer rapidement des applications et des logiciels à la fois conviviaux et réactifs. Plutôt que d'écrire des lignes de code et de structure de langage complexes, avec Visual Builder, vous pouvez facilement créer des applications comprenant des interfaces utilisateurs, des données, et des intégrations.

Le programme créé peut être intégré à la plupart des systèmes existants :

- Base de données,
- Systèmes de tous types, Oracle et produits tiers via 70+ adaptateurs,
- Ou via REST.

Bien que ce type d'outil nécessite des connaissances inhérentes au développement de logiciels, il peut être utilisé par des non-professionnels.

Visual Builder, c'est quoi ?

Visual Builder utilise simplement HTML5, JavaScript et CSS. Il fournit des outils de développement visuel simples et puissants pour créer des applications Web et mobiles (Progressive Web App, PWA), sans avoir à installer de logiciels supplémentaires. Cet ensemble d'outils visuels aide à concevoir rapidement votre application en faisant glisser des composants et en personnalisant leurs attributs pour définir leurs comportements. Pour les besoins complexes, les développeurs peuvent tout aussi facilement accéder au code sous-jacent, de façon bidirectionnelle.

Accès facile aux données

L'outil facilite l'accès aux données de votre application par le biais de services REST.

Database : vous pouvez aussi accéder ou créer des tables dans une base de données. Visual Builder génère les points d'accès REST pour vous.

Intégration/SaaS : Visual Builder est aussi un composant d'Oracle Integration. L'accès aux données exposées par les applications Oracle SaaS, par des produits tiers via des adaptateurs et par des intégrations personnalisées, se fait simplement via l'interface utilisateur.

REST: Vous pouvez accéder aux données de n'importe quel service REST en quelques clics. **Figure 1**

Plate-forme de développement et d'hébergement

Visual Builder est un service managé. Cela signifie qu'une fois que vous avez provisionné une instance de Visual Builder, il y a très peu de choses à faire en dehors du développement et de la publication de votre application. Tout ce dont l'application a besoin pour fonctionner (y compris un serveur Web pour héberger votre application et sécuriser l'accès aux données) est pris en charge. Ainsi, en tant qu'équipe de développement, vous pouvez faire passer votre application du stade du développement à celui de la publication en un temps très court. **Figure 2**

L'instance de Visual Builder fournit des capacités pour votre application à la fois comme un outil de développement visuel (en haut) et comme une plate-forme d'hébergement d'applications avec un serveur Web intégré (indiqué par les composants côté serveur en bas).

Regarder cette vidéo pour plus d'informations sur son fonctionnement, https://www.youtube.com/watch?v=9DNBAhOUTeY

PASSONS À LA PRATIQUE

Étape 1 : Free Tier

La première chose à faire est de se procurer une installation Visual Builder. Allez sur https://www.oracle.com/cloud/free et créez un compte Cloud Free Tier qui vous donnera accès à Visual

Figure 1

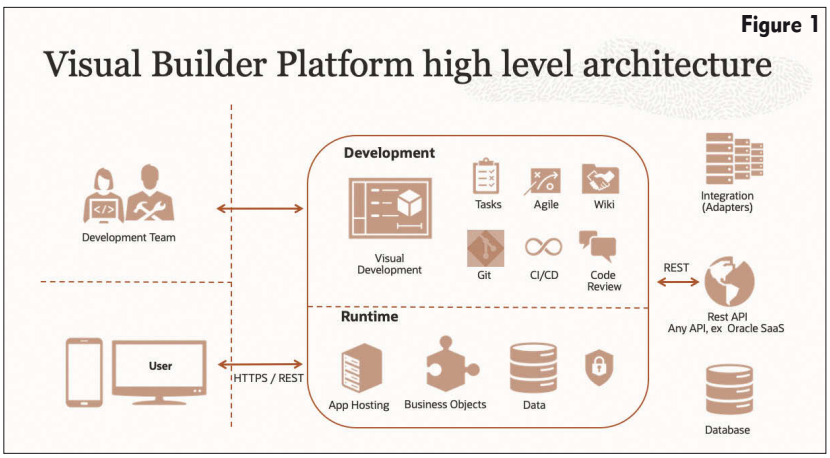
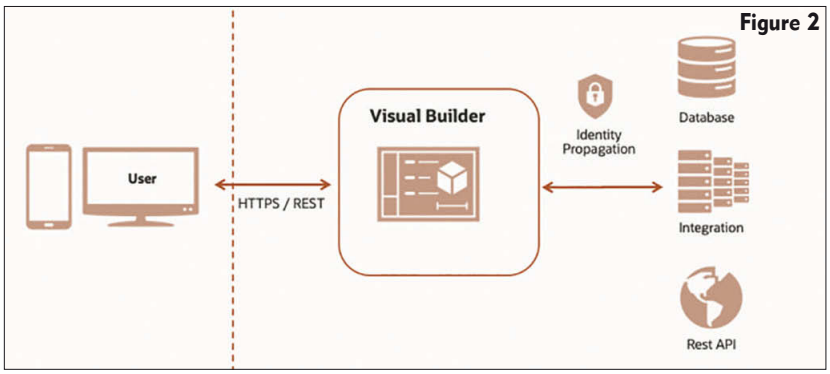


Figure 2



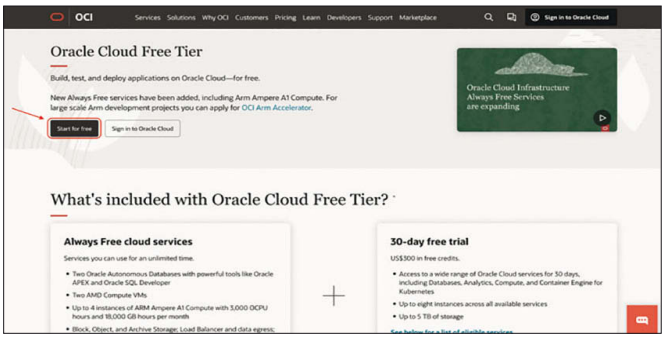


Figure 3

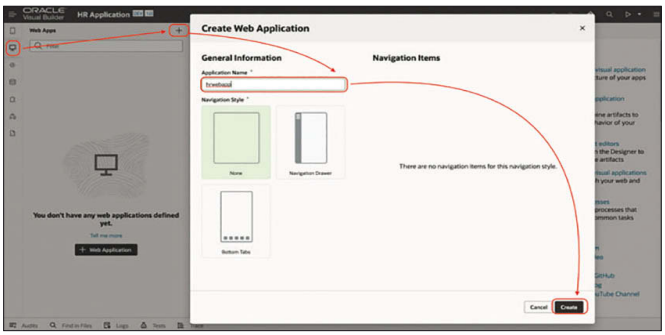


Figure 4

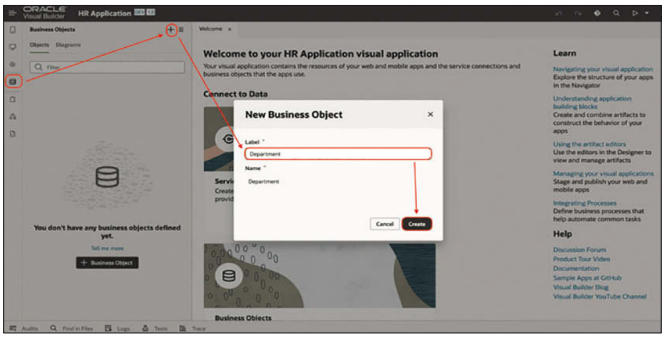


Figure 5

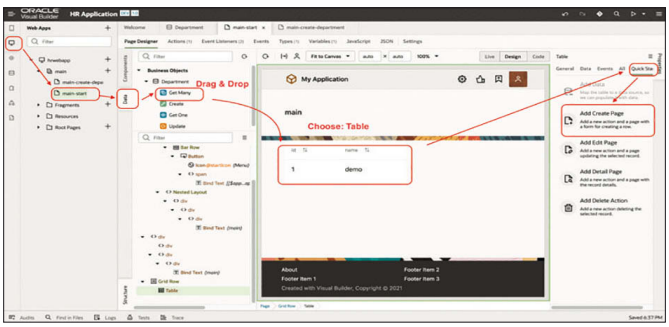


Figure 6

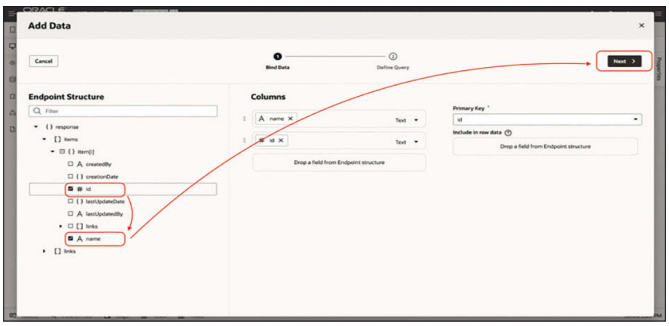


Figure 7

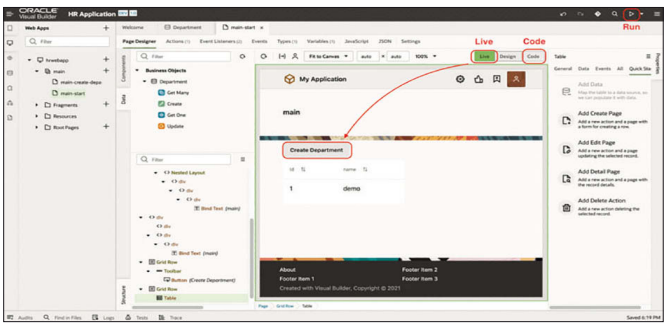


Figure 8

Builder pendant 1 à 2 mois gratuitement. **Figure 3**
Le compte Free Tier comprend aussi des machines Virtuelles, 2 CPU x86 et 4 CPU ARM, 200 Go de stockage…, gratuits et illimités dans le temps ! Pour plus de détails, visitez le lien ci-dessus.

Étape 2 : Provisionner Visual Builder
Pour provisionner Visual Builder, il faut créer une instance, donner un nom et attendre quelques minutes. Pour les instructions pas-à-pas, allez sur : https://apexapps.oracle.com/pls/apex/dbpm/r/livelabs/view-workshop?wid=763

Étape 3 : Créer une application Web.
Démarrez Visual Builder et cliquez « ***New Application*** »
Donner un nom : ***HR application***
Cliquez : ***Create***.
Cliquez sur l'icône ***Web Applications***.
Cliquez sur l'icône ***(+)***
Entrer : ***hrwebapp*** comme nom.
Puis cliquez ***Create*** **Figure 4**

Étape 4 : Table **Figure 5**
Nous allons créer une table et ses services REST associés.
Cliquez sur l'icône ***Business Objects***
Cliquez sur l'icône ***(+)*** / ***Business Objects***
Entrer : ***Department*** comme nom.
Puis cliquez ***Create***

Étape 5 : Ajoutons un champs.
Cliquez le tab ***Fields***
Cliquez ***(+)*** / ***Field***
Label: ***Name***
Cliquez ***Create***
Pour voir les services REST créés, allez dans le TAB Endpoints. Vous trouverez Get Many / Get One / Post (Create) / Patch (Update) / Delete (Delete)

Étape 6 : Ajoutons la table dans l'écran. **Figure 6**
Cliquez l'icône ***Web Application***
Ouvrez la page ***main-start***
Ouvrez le tab ***Data***
Drag and Drop ***Department*** / ***Get Many***
au milieu de la page.
Choisissez ***Table***
Choisissez les champs : ***id, name***.
Cliquez ***Next, Finish***

Étape 7 : Une page pour créer un département **Figure 7**
De retour dans l'écran d'édition,
Ouvrez la tab ***Quick Start***
Puis cliquez ***Add Create Page***
Choisissez : ***Department***
Puis ***Next***
Le champ ***Name***
Puis ***Finish***
Test: L'application est terminée. Figure 8
Testez-la soit via le mode ***Live***

Happy Coding !
Pour plus d'information :
https://docs.oracle.com/en/cloud/paas/app-builder-cloud

Les anciens numéros de PROGRAMMEZ!

Le magazine des dévs

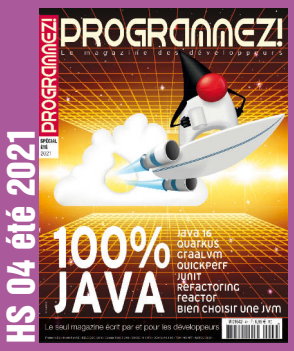
241

242

243

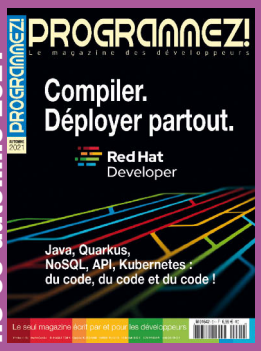
HS 02

246



HS 04 été 2021

249



HS 05 automne 2021

250

251

Complétez votre collection....

Tarif unitaire 6,5 € (frais postaux inclus)

☐ *241*	: __ ex	☐ *246*	: __ ex	☐ *250*	: __ ex
☐ *242*	: __ ex	☐ *HS4 été 2021*	: __ ex	☐ *251*	: __ ex
☐ *243*	: __ ex	☐ *249*	: __ ex		
☐ *HS2 automne 2020*	: __ ex	☐ *HS5 automne 2021*	: __ ex		

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

soit __ exemplaires x 6,50 € = __ € ..... __ € soit au **TOTAL** = __ €

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

Prénom : __ Nom : __

Adresse : __

Code postal : __ Ville : __

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

Offres pouvant s'arrêter à tout moment, sans préavis **PROG spécial 7**

Mike Kiersey
Director / Systems Engineering
United Kingdom, Boomi
https://www.linkedin.com/in/mikekiersey/

“Etre ou ne pas être” ?
La réponse est dans le Low Code / No Code

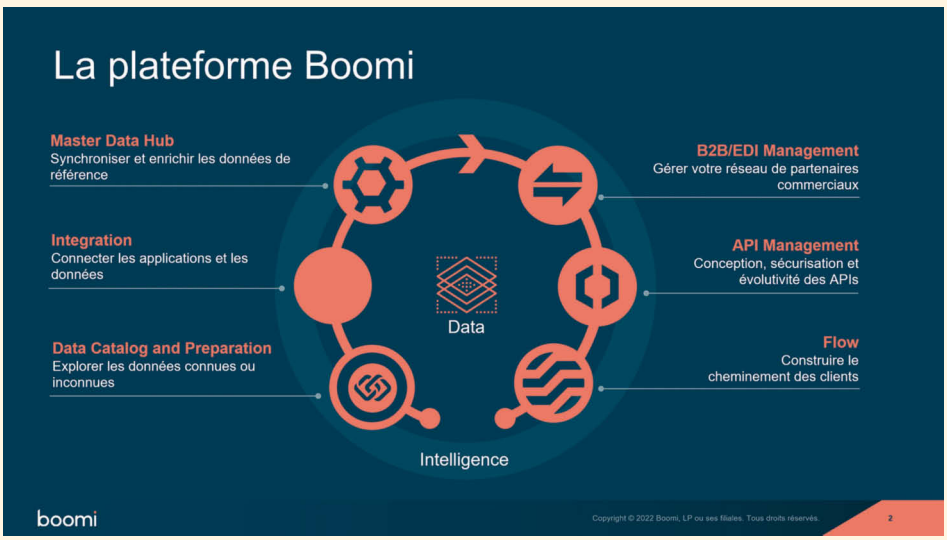
Ce qui lie les Jedi du codage et les guerriers de l'entreprise, c'est d'offrir la meilleure expérience, les meilleurs produits ou d'obtenir rapidement des résultats tangibles. Cette convergence d'intérêt s'est accélérée avec la pandémie et la nécessité de fournir des solutions numériques à un rythme encore plus rapide. La crise sanitaire a encore aujourd'hui des répercussions majeures et durables sur le paysage du développement logiciel, qui souffrait déjà d'une grave pénurie de talents. Dans ce contexte, les organisations voient dans les plates-formes low code une nouvelle approche pour mener de concert leurs projets commerciaux et informatiques.

Un constat

Face à la pénurie de main-d'œuvre et de compétences, les entreprises font pression pour disposer de méthodes plus faciles, plus simples et plus efficaces pour réaliser leurs projets numériques.
Selon une étude Gartner sur la croissance des plates-formes low code, 41 % des employés non techniques personnalisent ou structurent déjà des données ou des solutions technologiques. Ce même rapport prévoit que d'ici fin 2025, la moitié des utilisateurs de plates-formes low code seront extérieurs au secteur informatique. Selon Forrester, l'usage de plates-formes low code pour du développement d'applications s'est même accru de 31 % entre 2020 et 2021.
Pour les développeurs, c'est une opportunité d'acquérir de nouvelles compétences et de rester pertinents sur le marché où les changements s'accélèrent. C'est une situation gagnant-gagnant pour les entreprises, qui peuvent ainsi capitaliser sur tous les aspects d'un low code conçu par et pour des développeurs, en vue de faire évoluer les processus en entreprise, l'intégration, la gestion de données, le développement d'API, etc.
Vous associer, en tant que développeur interne, à l'adoption d'une approche low code pour réaliser le changement numérique de l'entreprise présente de nombreux avantages :

- Répondre aux difficultés de recruter de nouveaux développeurs.
- Aider à construire une fonction libre-service pour apporter en souplesse des changements informatiques.
- Participer à un alignement plus fort entre l'informatique et l'entreprise.
- Réduire le délai de création de valeur à seulement quelques semaines ou jours.
- Aider les autres collaborateurs à développer de nouvelles compétences au sein de l'entreprise et à se recycler.
- Participer à la rapidité d'exécution, et délivrer plus de projets dans le même temps.
- Aider à réduire les coûts d'exploitation.
- Supprimer les coûts et dettes des anciennes plates-formes ou des anciens systèmes.
- Favoriser l'accès à l'informatique dématérialisée comme l'évolutivité, la disponibilité, les performances et la sécurité.

Pourtant, la réalisation de projets plus rapidement qu'un développement traditionnel nécessite une réflexion, une gouvernance et des bonnes pratiques en matière d'architecture d'entreprise. A défaut, vous ne faites que décupler le problème.
Plusieurs rapports de Forrester soulignent les atouts de ces plates-formes cloud natives, ouvertes, distribuées et unifiées.
Le choix d'une plate-forme low code doit prendre en compte plusieurs paramètres. Boomi a investi dans des outils et des modèles pour vous aider à comprendre les principales métriques à mesurer.
L'*Integration platform as a service* (iPaaS) de Boomi est une plate-forme qui connecte vos applications et automatise vos flux de travail, avec plus de 18 000 clients en référence. La plate-forme traite plus de 4,5 milliards d'intégrations tous les mois soit près de 55 milliards d'intégrations et 1 trillion de documents par an.



ROI

Selon une étude de Forrester sur l'impact économique total (TEI - Total Economic Impact), la plate-forme Boomi permet en moyenne un retour sur investissement (ROI) de 410 % sur une période de trois ans, et est rentabilisée en moins de six mois.
Toujours selon Forrester, Boomi AtomSphere de Boomi réduit de 65 % les temps de développement de l'intégration, ce qui permet aux équipes IT d'économiser sur leurs dépenses dédiées à l'efficacité, et dans un même temps, de connecter les données, les applications, les appareils et les personnes en quelques heures ou jours, au lieu de plusieurs semaines ou de mois.

Plate-forme iPaaS : Quels fondamentaux pour commencer votre mouvement vers le Low Code ?

A mesure que les entreprises adoptent une culture de l'utilisation d'applications cloud nécessitant peu de code pour atteindre leurs objectifs numériques, l'adoption de plates-formes purement low code monte en flèche. Et ce mouvement ne peut que s'accélérer. Quels sont les incontournables pour l'équipe de développement ?

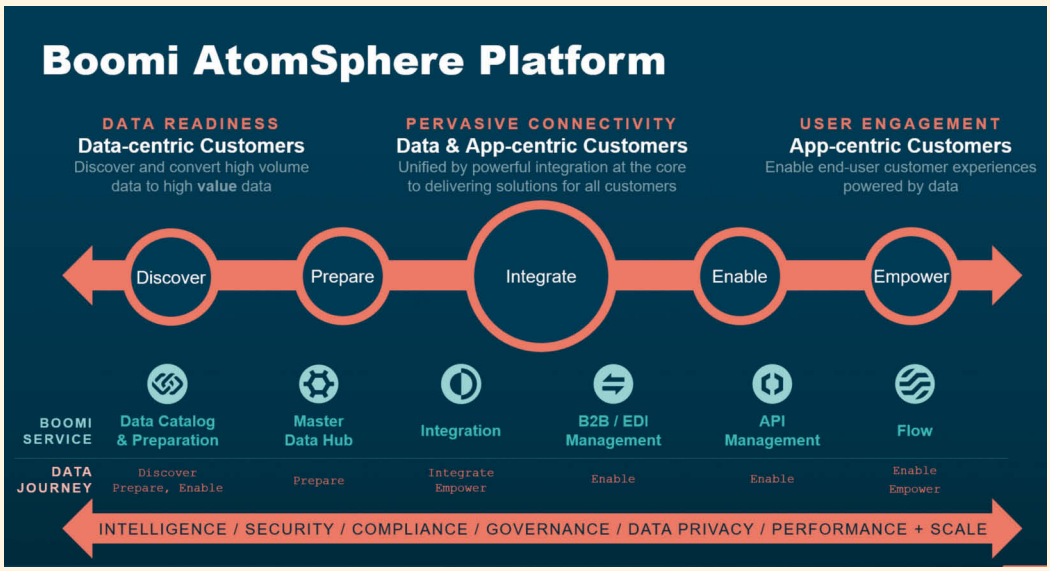
Par Mike Kiersey

Cloud Native – les contraintes commerciales actuelles exigent des équipes informatiques qu'elles en fassent plus, et plus vite, pour répondre aux exigences et besoins croissants des clients. Les approches lourdes en code ne suffisent plus. Les environnements de développement à faible code réduisent considérablement le temps nécessaire à l'intégration des applications et des données, ce qui permet aux équipes informatiques d'aller plus vite et de réaliser des projets rapidement et à moindre coût.

Ouverte - Pour connecter efficacement la myriade d'applications, de sources de données et de dispositifs actuels et à venir, votre iPaaS doit être agnostique en termes d'applications et de points d'extrémité sans vous contraindre à utiliser les applications ou les plates-formes cloud de tel ou tel fournisseur.

Distribuée - L'iPaaS doit tout intégrer, des applications sur site et dans le cloud aux appareils IoT et périphériques, et prendre en charge des volumes de données en croissance exponentielle. Pour y parvenir efficacement - et tenir la promesse d'une faible latence, de hautes performances et de mises à niveau automatiques - un iPaaS doit avoir une architecture distribuée.

Unifiée - L'intégration des applications est au cœur du système. Tout iPaaS doit prendre en charge toute la gamme des modèles d'intégration, tant au sein d'une entreprise qu'au sein de son réseau de partenaires, mais pour accélérer les résultats de l'entreprise, un iPaaS doit également



garantir la synchronisation des données et l'accès à des données de haute qualité à tout moment, en tout lieu et sur tout appareil.

Au-delà des fondamentaux architecturaux, comment maximiser l'efficacité d'une plate-forme low code ?

Qu'il s'agisse de l'intégration d'applications et de données low code, du développement d'API ou de la création d'applications commerciales intuitives, vous aurez besoin de votre partenaire iPaaS pour :

- Une méthodologie de livraison éprouvée pour aider à maximiser votre expérience, basée sur des services professionnels et un réseau de partenaires certifiés.
- Une stratégie IT planifiée et développée pour le low code, afin de réduire le risque de "low code sprawl", c'est-à-dire de se retrouver avec une liste interminable de projets déployés pour solutionner un problème déjà obsolète.
- Un apprentissage structuré, assuré en lien avec le fournisseur iPaaS, avec des engagements de projet co-délivrés (comme la programmation par paire dans un mode de livraison agile).
- Une approche « gestion de projet », éprouvée pour délivrer des sprints agiles en utilisant une plate-forme comme Boomi.
- Des normes architecturales bien définies qui permettent de cerner le modèle d'application commun et les principes à respecter par l'entreprise et l'organisation informatique.
- Des exemples des meilleures pratiques pour assurer aux équipes commerciales qu'elles atteignent les plus hauts niveaux d'agilité, de rentabilité et de stabilité opérationnelle.
- Une architecture de référence pour disposer d'exemples immédiats sur la manière d'utiliser les capacités de la plate-forme et pouvoir se l'approprier rapidement.
- Un CoE ou Compétences d'excellence. Pour les grandes organisations, le développement d'un CoE est essentiel pour établir des normes, une gouvernance et des niveaux élevés de durabilité au sein du service de dev low code.
- Des contrôles de fonctionnement. Extrêmement importants et quelque peu négligés, les contrôles de bon fonctionnement, avec le fournisseur, sont essentiels à la durabilité et permettent aux organisations d'anticiper les lacunes éventuelles à mesure que les meilleures pratiques évoluent et changent.
- Une communauté d'utilisateurs acquise à la cause low code, qui stimule la réflexion et l'adoption de plates-formes low code, en apportant des idées, des recommandations, des solutions et des innovations pour faire évoluer l'entreprise.

Pierre Oudot
Country Manager France, Boomi

Le Low-Code gagne tous les acteurs de l'IT

La pénurie de développeurs menace de ralentir la poursuite de la transformation numérique des entreprises. Elle a conduit de nombreux éditeurs, et autres acteurs de l'IT, à s'intéresser aux plates-formes Low-Code et No-Code. Développement de nouvelles applications, intégration, déploiement de fonctionnalités IA, les cas de figure sont nombreux pour répondre aux attentes des directions métiers.

Effervescence du marché du logiciel autour des solutions Low-Code ou No-Code.

L'engouement se généralise. L'enjeu est de taille puisque selon Gartner les applications ayant recours au Low code devraient représenter 64 % des applications d'ici 2024. Après les « pures players » comme Mendix et OutSystems, c'est au tour des éditeurs de logiciels de revendiquer cette terminologie pour leurs plates-formes de développement ou d'y inclure des fonctions de ce type dans leurs solutions.
Conscients que la pénurie de développeurs pourrait freiner les multiples chantiers de modernisation à l'œuvre dans les entreprises, les éditeurs entendent bien soulager le travail des développeurs et aider les directions métiers à réduire le temps de mise en services de leurs nouvelles applications. Les ERP n'échappent pas à la tendance, à l'image de SAP qui propose de créer des processus automatisés sur S4/HANA avec une plate-forme No-Code du nom de Ruum.
Selon une estimation du cabinet Markets and Markets, le marché mondial du Low-Code No-Code devrait représenter 45,5 milliards de dollars en 2025, et il intéresse particulièrement les leaders du Cloud, lesquels étoffent déjà leurs offres, comme Microsoft avec PowerApps et plus récemment Amazon avec Honeycode.

L'intégration monopolise encore trop de développeurs.

Impossible de parler d'applications sans évoquer le sujet de l'intégration. Le fonctionnement fluide d'un système d'information repose sur une parfaite intégration de multiples applications interconnectées, quelles que soient les évolutions du SI. Les développeurs y consacrent un temps précieux qui ralentit d'autant l'écriture de nouvelles applications, pourtant plus créatrices de valeur. Or, en utilisant des plates-formes iPaaS (plate-forme d'intégration as a service) disposant d'une interface de développement Low-Code les opérations d'intégration peuvent être simplifiées et accélérées. Le Low-Code accélère le travail d'intégration d'un facteur de 5 à 10. Un code d'intégration peut être généré en quelques jours, voire en quelques heures. On obtient ainsi une intégration moderne et flexible mieux adaptée aux contraintes d'évolutions actuelles. Enrichie d'une solution Low Code, une plate-forme d'iPaaS permet la conception rapide de processus métier de bout en bout. Les silos de données disparaissent, les workflows s'automatisent, ce qui réduit considérablement le temps nécessaire à la mise en route effective d'un nouveau produit ou d'offre commerciale (le « time to market »).

Démocratiser l'intelligence artificielle.

Le Machine Learning (ML) change lui aussi les règles de la programmation en profondeur. C'est l'algorithme qui définit ses propres règles après un processus d'apprentissage alimenté par des jeux de données adéquates. Les cycles traditionnels de développement ML peuvent s'étendre sur plusieurs mois et nécessitent une collaboration étroite entre les analystes de données, les datascientists et les métiers.
Pour améliorer cette collaboration, des entreprises comme Alteryx, Dataiku, DataRobot, pour n'en citer que quelques-unes, proposent des plates-formes Low-Code spécialisées pour le ML qui permettent de créer rapidement des Proof of Concept afin de concrétiser des idées et de faire évaluer rapidement leur faisabilité par des datascientists, souvent très sollicités. La promesse de Google, Microsoft Azur et AWS est également alléchante. Leurs offres permettent aux analystes de données de créer à partir d'interfaces visuelles et intuitives des modèles de ML sans écrire une seule ligne de code et obtenir par exemple des algorithmes de prédictions de précision. Le Low-Code et le No-Code font de nombreux adeptes jusqu'aux concepteurs de processeurs accélérés par GPU tel Nvidia qui propose une solution pour créer des applications de reconnaissance visuelle basée sur l'Intelligence Artificielle.

L'amélioration du développement logiciel est un long processus entamé depuis le début de l'ère de la programmation. Le Low-Code et le No-code ne sont pas les seules voies possibles. Il en existe d'autres et de nouvelles vont émerger. Intel a déjà annoncé le logiciel ControlFlag capable de détecter automatiquement, à l'aide de ML, les anomalies dans les codes informatiques et DeepMind AI a dévoilé AlphaCode, un nouveau générateur de code avec de l'IA. Reste que l'association du Low-Code à une stratégie d'intégration hybride et multicloud est le meilleur atout pour intégrer rapidement et de manière fluide l'existant et les applications développées initialement sur site.

Salesforce : les no code / low code sont dans l'ADN de la plateforme

Aujourd'hui, Salesforce n'est plus le « simple » CRM 100 % centré sur le Sales Automation de ses débuts. Cette fonction centrale a permis à l'éditeur de s'étendre sur d'autres aspects de relation clients comme le service client, marketing, le e-commerce ce qui fait de l'entreprise aujourd'hui un des acteurs majeurs de la donnée client, notamment avec plusieurs rachats stratégiques : Heroku, Mulesoft Tableau et le dernier slack. Si nous regardons attentivement la partie développement, deux piliers sont proposés : declarative development et programmatic development. Le premier est proche du no code, alors que le sujet est plus orienté low code. Dès sa création, les services cloud (actuellement regroupés sous le nom Customer 360) sont proposés sur une plateforme conçue pour la réduction du code. Simplifier la conception de processus et d'applications ont été deux priorités.

Par François Tonic

L'une des forces de la plateforme est la possibilité de développer et de déployer des applications (web, mobile, desktop) s'appuyant sur les services Salesforce et une vue unifiée des données "Single Source of Truth" . La donnée doit être injectée, stockée et surtout analysée et affichée pour pouvoir en tirer toute sa valeur. Mais Salesforce est une plateforme très riche fonctionnellement et c'est pour cela que dès l'origine, l'éditeur a voulu proposer des outils pour simplifier l'usage des services et des données.

Aujourd'hui, on estime que 80 % des apps développées sur Salesforce sont en no code, 20 % en low code.

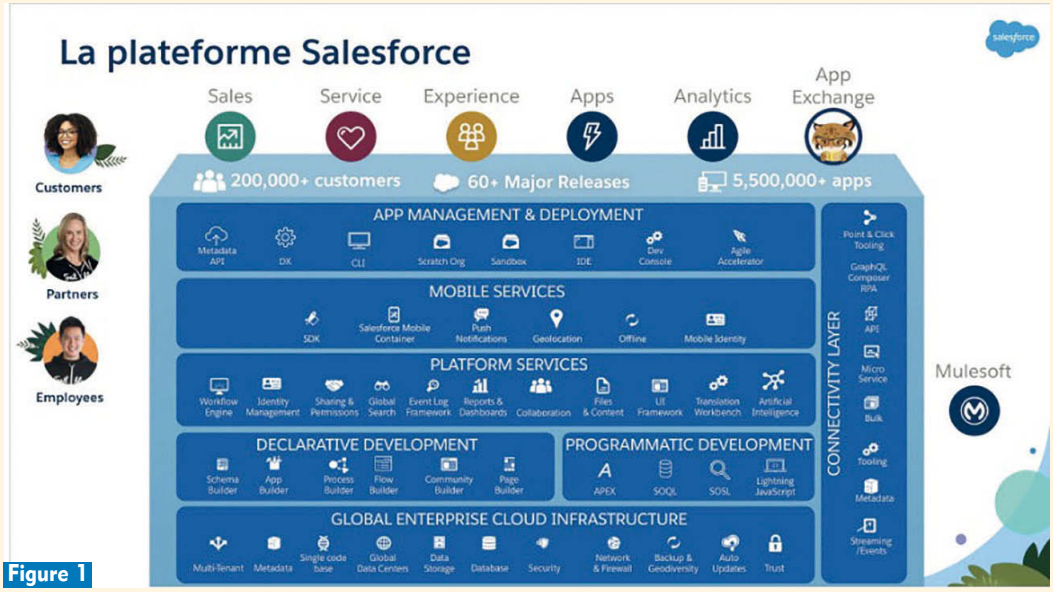


Figure 1

Figure 1

Développement déclaratif : approche no code

Ces outils se dédient avant tout aux business analysts, aux data scientists, ou équipes métiers / marketing / ventes, qui sont les plus moteurs pour accompagner les transformations avec le numérique de leur entreprise. L'idée est simple : aller vite pour créer une app pour visualiser, manipuler les données. Tout se fait par composition de composants et configurations.

Six outils sont proposés : Schema Builder, App Builder, Process Builder, Flow Builder, Community Builder, Page Builder.

APP EXCHANGE : TROUVER LA FONCTIONNALITÉ QUI MANQUE

Pour les développeurs, App Exchange est le store incontournable pour trouver la fonctionnalité non disponible en standard lorsque l'on développe son app. Plus de 8000 applications existent. Ces composants sont prêts à l'emploi. On trouve la bonne fonction, on installe et peut l'intégrer à son projet. Par exemple, pour la signature électronique, DocuSign a développé un composant.

Ce sont des développeurs et partenaires qui développent les fonctionnalités manquantes qui peuvent ensuite être proposées à tous les utilisateurs sur Appexchange.

Développement programmatique : approche low code

L'approche programmation repose sur APEX, SOQL (langage de requêtage Salesforce), SOSL (langage de recherche dans les données Salesforce) et LWC (Lightning Web Components). LWC est l'outil de référence low code de la plateforme. Le composants Lightning sont des éléments de code réutilisables riches fonctionnellement généralement classés par métier ou industrie. Apex est un langage objet et fortement typé pour créer et exécuter les flux et transactions sur la plateforme Lightning. La syntaxe est proche de celle de Java et son comportement ressemble aux procédures stockées des SGBD. L'autre outil de développement incontournable est Lightning Web Components. LWC prend la succession d'un autre outil de l'éditeur : Aura framework. LWC repose sur les standards du web. On peut utiliser Visual Studio Code pour coder des projets LWC. C'est le modèle de base des composants Salesforce.

Typiquement, une app Apex est déployée sur la plateforme. Il faut avoir en tête qu'une App Salesforce est un code XML, des metadonnées. On peut ouvrir et coder le projet directement sur Visual Studio Code. On déploie donc des métadonnées sur une instance. On peut faire du debug via le debug log directement dans la plateforme Salesforce pour monitorer son app et tracer l'utilisateur. On peut débugger du code Apex via le system debug directement dans VS Code. On rajoute system.debug dans son source Apex pour ensuite récupérer les logs dans debug logs. On dispose d'une extension VS Code pour Apex : Apex Replay

Figure 2

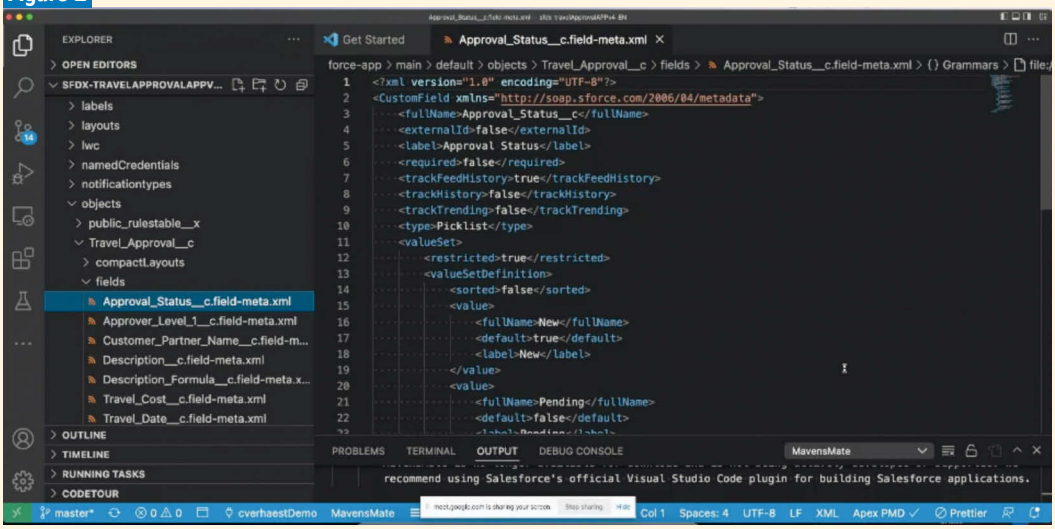
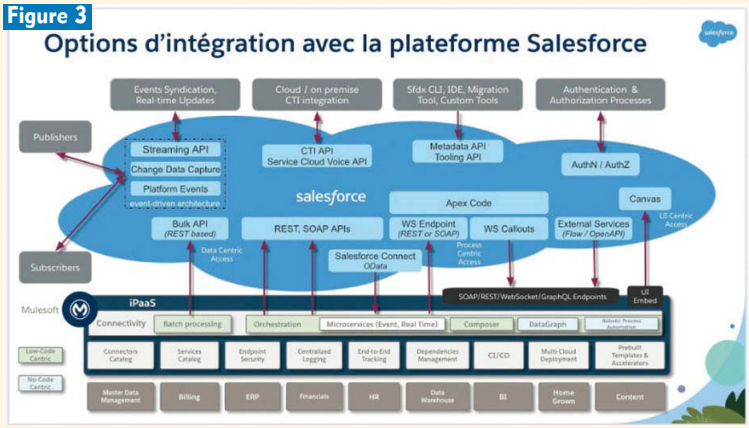


Figure 3



TRAILHEAD : LE CENTRE DE DOCUMENTATION INCONTOURNABLE

Trailhead centralise toute la documentation Salesforce pour les développeurs et les utilisateurs. La plateforme permet de découvrir les services, de la prendre en main plus rapidement, de se former. Il est aussi possible de se préparer et de passer des certifications. La dimension communauté est très importante. Cette rubrique permet d'accéder aux forums, aux événements et aux différents groupes utilisateurs. Pour les entreprises, une section dédiée est disponible.

Trailhead Academy permet de former avec un expert Salesforce. Ces formations se déroulent sur plusieurs jours.
Les parcours sont accessibles gratuitement et permettent de se former sur de nombreux services Salesforce. Chaque trail se divise en module de 10 minutes en moyenne. Chaque trail permet d'acquérir des points. Vous pouvez choisir le niveau technique, le type de produits et votre profil (développeur, architecture, etc.)
https://trailhead.salesforce.com/fr

MULESOFT : LES API D'INTÉGRATION !

Vous l'aurez compris, Salesforce s'appuie fortement sur les API MuleSoft pour pouvoir intégrer et communiquer avec le « monde extérieur » et tout silos de données. Cette intégration se fait aussi bien par code que pour une approche low code / no code. Les dernières annonces renforcent cette logique. Il faut voir MuleSoft comme une couche iPaaS avec du batch, de l'orchestration, du microservice (événements et temps réel), datagraph et du RPA.

Les External Services sont importants pour appeler une API externe. Il supporte Flow et OpenAPI. On configure l'appel à l'API dans la plateforme. On peut se connecter à son environnement MuleSoft pour dire l'API que l'on veut utiliser pour définir l'appel et mettre à disposition. Là encore, en configuration.

LES DERNIÈRES ANNONCES DURANT TRAILBLAZER DX 2022

Durant la conférence technique Trailblazer DX d'avril 2022, Salesforce a dévoilé beaucoup d'outils et de services autour du low code, no code et de l'automatisation sur l'ensemble des plateformes : Salesforce, MuleSoft, Tableau et Slack. L'automatisation se généralise dans l'ensemble des couches Salesforce et des services.

- Salesforce Flow : c'est l'outil majeur de la plateforme pour les développeurs. Ce n'est pas un nouvel outil, mais il s'est enrichi de nouvelles capacités d'intégration avec Slack, par exemple envoyer des messages dans un Slack ou depuis Slack déclencher un Flow Salesforce. Bref, on facilite l'intégration et l'interaction.
- Tableau Embedded : possibilité de mieux intégrer les données et tableaux dans les apps Salesforce
- Salesforce Platform for Slack : nouvelle toolbox pour créer des apps et automatiser Slack
- MuleSoft Anypoint Code Builder : nouvel IDE pour créer, intégrer les API. IDE simplifié pour faciliter l'intégration avec une approche Now Code. Outil hébergé.
- Anypoint Flex Gateway / API Gouvernance : gouvernance des API que l'on crée, que l'on intègre. Ces outils sont là pour sécuriser les API, vérifier la conformité des API et de la sécurité. En complément aux outils Low Code / No Code.

Debuger. **Figure 2**
Côté MuleSoft, que l'on fasse du Datagraph ou du Composer, le debug se fait directement dans la console web (mode test).

Intégration

Salesforce fournit des API pour intégrer des objets Salesforce depuis son app qui n'est pas Salesforce. Typiquement, l'éditeur fournit des API pour pouvoir communiquer avec son instance Salesforce. Il faut, bien entendu, donner les droits d'accès. Depuis Salesforce, on peut bien entendu intégrer / interroger les apps tiers, des apps mainframe, etc. Les API Mulesoft jouent un rôle central pour faire le lien avec la couche de connectivité. Par exemple, pour intégrer SAP. Mulesoft fournit ce que l'on appelle la couche iPaaS avec du traitement batch, microservices, DataGraph, RPA, etc. External Services pour appeler une API tiers (Swagger ou OpenAPI). On peut configurer l'appel au service directement dans la plateforme. Edition Mulesoft, on se connecte à son environnement Mulesoft et on configure l'API directement dedans. On peut aussi appeler une API dans un outil déclaratif comme Flow. **Figure 3**

Merci à Lila Dorato, Guillaume Tourres, Cyril Verhaest et Kiet Yap

Ma première app Lightning

HELLO WORLD : DÉVELOPPEZ VOTRE PREMIÈRE APP SALESFORCE

Après la théorie, la pratique ! Voyons comment développer en 8 étapes une application évolutive, avec une logique itérative, sur la plateforme Salesforce. À chaque étape, vous disposez d'outils de configuration prêts à l'emploi, en mode déclaratif ou au besoin, en programmation. Let's code.

Notre app s'appelle *Demande de Voyage* : les employés d'une entreprise s'en servent pour faire approuver un déplacement professionnel par leur responsable, et pour suivre l'avancée de leurs demandes.

Le flux des demandes repose sur 3 étapes réduites à l'essentiel. Ce flux permet à l'utilisateur d'être guidé dans la saisie de sa demande. Dans une fenêtre flottante, l'utilisateur renseigne toutes les informations liées au voyage : date, motif, destination... En fonction du motif du voyage (réunion interne, rencontre client), des champs additionnels apparaissent (nom du client, projet…) pour préciser la demande. Cette interactivité permet d'alléger l'interface et de rendre l'expérience (plus) pertinente. La seconde étape du flow est de renseigner le coût, et la troisième est un récapitulatif de la demande avant envoi au manager. Enfin, l'utilisateur retrouve l'avancée de toutes ses demandes dans un tableau de bord.

1 Préparer le modèle de données

Nous avons à notre disposition deux outils low code pour établir le modèle de données :
- **Schema Builder** représente visuellement les relations entre les objets (équivalent des tables) qui composent l'app. Ici, un objet custom « Travel Approval » et deux objets standards, « Account », pour lier un client à la demande et « User » pour lier le demandeur et son manager.
- **Object Manager**. Il sert à configurer les objets utilisés par l'app, notamment les champs qui le compose (date, motif, coût...). À cela s'ajoute la possibilité de configurer des règles de saisie (*Validation Rules*) pour s'assurer de la qualité des données.

2 Gérer les droits d'accès

La sécurité des données est au cœur de Salesforce. La plateforme permet de paramétrer en lecture seule ou lecture/écriture chaque donnée, champ, composant ou objet, pour tous les types d'utilisateurs de l'application. Pour les administrateurs, c'est la garantie d'une discrétion absolue dans le partage de l'information. Pour les utilisateurs, c'est une interface qui est simplifiée (les composants inutiles sont masqués).

3 Structurer l'application

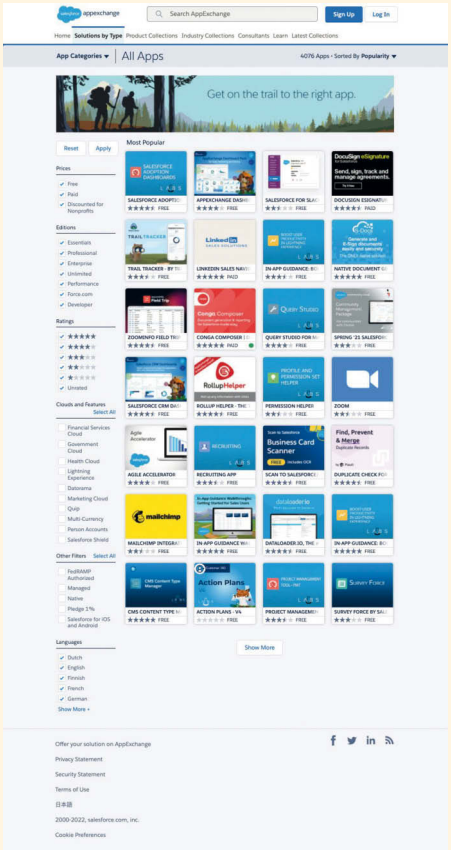
L'outil **App Manager** permet de créer l'ossature de l'application. On y ajoute sous forme d'onglets, tous les objets et fonctionnalités utiles au fonctionnement de l'app. Ainsi, depuis la page d'accueil de l'application, l'utilisateur peut consulter des objets connexes (compte client, projet) sans avoir à ouvrir une autre application ou un autre système.

4 Construire l'interface utilisateur

App Builder permet de créer l'UI par glisser-déposer de composants. Le moindre champ de l'interface est un composant pouvant être déplacé, ou masqué selon le contexte, en fonction du type de device utilisé, du type d'utilisateur, de ses droits et des informations propres à la demande.
Salesforce propose une large librairie de composants standards, afin de limiter le développement spécifique. Il est possible d'ajouter des composants *customs*, appelés "Lightning Web Component", codés en HTML, Javascript et CSS côté client et avec du code Apex côté serveur (langage Salesforce similaire à du Java). Dans notre projet, un composant *custom* a été ajouté pour uploader facilement les notes de frais relatives à une demande de voyage. Un autre composant *custom* affiche les règles de voyages (travel policy), en interrogeant en temps réel une application tierce. Ainsi, toutes les données, mêmes externes, peuvent être affichées dans l'interface Salesforce.

5 Automatiser les processus

Lorsqu'un utilisateur dépose une demande de voyage, il déclenche s'en vraiment s'en apercevoir un flux automatisé réalisé à l'aide de **Flow.** L'outil représente visuellement, sous forme de carte, tous les écrans et actions qui se succéderont pour accomplir le flux (étapes guidées). Il est possible de personnaliser l'interface de chaque action, toujours à l'aide de glisser-déposer de composants. **Flow Builder** sert autant à créer des *screenflows* (visible par l'utilisateur) ou des *auto-launched flows* (en arrière-plan).



6 Mobilité

Toutes les applications conçues sur la plateforme Salesforce sont nativement *"mobile ready"*. Aucune manipulation n'est nécessaire, l'utilisateur accède à ses applications sur *ces devices mobiles*.

7 Reporting

L'analyse des performances est au cœur du business. Salesforce propose deux outils pour créer des rapports et tableaux de bord : **Report Builder** et **Dashboard Builder** Ces rapports sont cliquables et affichent toutes les données internes ou externes à Salesforce. Il est facile de créer des dashs sur l'activité de chaque application.

8 Intégration et développement

Pour peaufiner l'application selon vos souhaits, améliorer l'expérience utilisateur, et dans le cas où les éléments standards ne répondent pas totalement à vos besoins, il est bien évidemment possible de développer vos propres composants, processus automatisés... Pensez néanmoins à explorer les librairies de composants Salesforce ou l'AppExchange, pour voir si un partenaire n'a pas déjà conçu la fonctionnalité dont vous avez besoin. Enfin, il est possible d'intégrer les applications Salesforce avec des solutions externes, via des accélérateurs (ex : APIs standards, Salesforce Connect, External Services for MuleSoft, ...) ou via du développement.
C'est tout. A vous de jouer !

Un écosystème de ressources pour étendre votre plateforme

Il n'est pas toujours facile de se lancer dans l'apprentissage d'un outil aussi vaste que Salesforce, d'autant plus quand le temps manque et que les métiers réclament aux développeurs des applications en quelques semaines. C'est pour cela que Salesforce en propose plusieurs à destination de ses développeurs. Qu'il s'agisse de se former rapidement et gratuitement aux dernières fonctionnalités de la plateforme, ou d'obtenir des applications prêtes à l'usage via ses magasins, tout est à votre disposition pour accélérer votre travail et satisfaire vos clients.

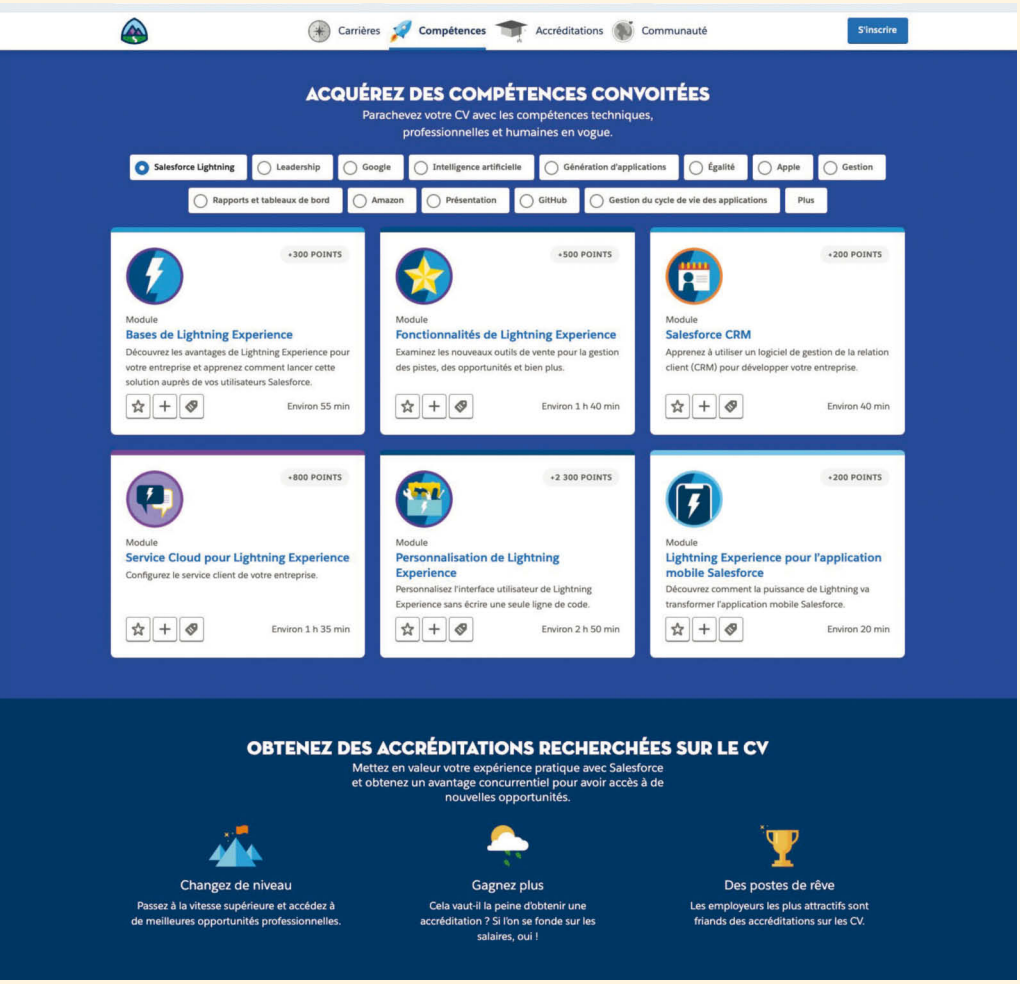
1 Trailhead : se former gratuitement

Trois fois par an, Salesforce apporte des mises à jour majeures à l'ensemble de sa plateforme. Toutes ces nouvelles fonctionnalités sont décrites dans l'aide aux développeurs. Mais pour apprendre à les maîtriser, rien de mieux que **Trailhead**. Lancé en 2014, le site web propose des modules gratuits de formation à sa plateforme. Vous pouvez y suivre des **Trails : des parcours** de modules pour maîtriser de fond en comble les sujets qui vous importent le plus : administrateur d'instance, développement, architecte cloud... Ces trails sont proposés en Français et en Anglais.

Mais Trailhead propose surtout une expérience ludique et gamifiée. Chaque module accompli vous octroie des points et des badges à afficher sur votre profil Trailhead. Votre progression vous fait monter en niveau, jusqu'à atteindre le grade ultime de Ranger, le plus reconnu de la communauté. Ce grade ultime est en train d'ailleurs d'être modifié pour prendre en compte la montée en compétence et le nombre de plus en plus important de Rangers. Car, au-delà du plaisir d'apprendre, le monde professionnel scrute avec attention les développeurs les plus qualifiés sur Salesforce. N'hésitez donc pas à afficher vos badges et votre progression sur LinkedIn, pour booster votre carrière.

2 AppExchange : l'App Store des pros

Bien que Salesforce propose de nombreuses fonctionnalités dans sa version standard, chaque entreprise a des besoins précis, et peut être amenée à personnaliser sa plateforme pour y répondre soit avec des applications développées en interne, soit avec des fonctions tierces. Seulement, le développement *custom* est un processus long et coûteux, incompatible avec le calendrier des métiers. C'est pour cela que Salesforce propose AppExchange. Cette place de marché d'applications B2B, de workflows et de composants lightning permet d'acquérir des add-ons créés par d'autres développeurs, afin d'enrichir votre instance Salesforce, sans avoir à les développer. En mai 2022 plus de 8 000 applications gratuites ou payantes sont disponibles. Depuis son lancement en 2006, 10 millions de téléchargements ont été effectués sur le magasin.



3 Anypoint Exchange : le magasin d'API

Suivant la même logique qu'AppExchange, Anypoint Exchange est le magasin d'API de la solution MuleSoft Anypoint Platform. Ce magasin met à votre disposition une bibliothèque d'API, de connecteurs et de modèles réutilisables, pour connecter vos données, applications et systèmes sans avoir à les développer. Un gain de temps considérable pour vos projets.

4 Tableau Exchange : data-visualisation de haut niveau

Tableau est l'outil de data-visualisation de Salesforce, intégré dans toutes les solutions de la plateforme. Pour étendre les capacités des tableaux de bord et des rapports, le magasin Tableau Exchange propose des extensions de fonctionnalités (nouveaux diagrammes), des connecteurs vers des bases de données externes (Oracle, SAP) et des accélérateurs. Ainsi, la data-visualisation gagne en précision, en profondeur et en richesse, et ce, directement depuis la fenêtre principale des utilisateurs.

Ensemble, ces ressources promettent d'accélérer votre travail de développement, de découvrir de nouvelles fonctionnalités et de livrer plus vite à vos utilisateurs la plateforme idéale pour leur travail.

Histoire amusante

AppExchange devait initialement s'appeler... App Store ! Marc Benioff, le CEO de Salesforce, était allé rencontrer Steve Jobs à Cupertino en 2003. Ce dernier aurait dit à Marc Benioff que pour assurer sa croissance, Salesforce devait devenir un écosystème de software dans le cloud. Ceci a poussé Benioff à imaginer le concept de magasin d'applications, qu'il souhaitait appeler App Store. Un nom de domaine avait été rapidement déposé. Cependant, son entourage lui a déconseillé ce nom, pour préférer AppExchange en 2006. Deux ans plus tard, au lancement de l'App Store par Apple, Marc Benioff a cédé le domaine appstore.com à Steve Jobs, en remerciement des conseils qu'il lui avait prodigués 5 ans plus tôt.

AWS Amplify Studio : Low Code pour le dev full stack

Le low-code, ou no-code, menace-t-il nos jobs de développeurs ? Non. Ces technologies nous permettent d'être plus productifs, de passer moins de temps à coder des coins et recoins des applications qui n'apportent pas ou peu de valeur ajoutée. Par exemple les fonctions d'authentification, d'appels sécurisés d'API, ou encore le codage du lien entre les composants UI et les modèles de données : ce sont des fonctionnalités de commodités et standards. Sans elles, nous pouvons enfin nous consacrer à des fonctions à plus hautes valeurs ajoutées, celles qui feront la différence pour attirer et garder les utilisateurs sur nos apps.

Florian Chazal
Senior Solution Architect, AWS

Sébastien Stormacq
Principal Developer Advocate, AWS

Merci à Jérôme Van Der Linden, Solution Architect, AWS pour la relecture et les suggestions.

AWS Amplify est un framework et une CLI open source pour les développeurs full stack qui permet de créer et de gérer une infrastructure cloud, sans être un spécialiste, et surtout d'écrire du code de haut niveau pour intégrer ce backend. En quelques commandes et lignes de code, vous pouvez ainsi ajouter des fonctions d'authentification, un data model, une API REST ou GraphQL, une base de données, du stockage et de la synchro des données en local, etc. Vous pouvez démarrer avec Amplify sans posséder un compte AWS grâce à sa GUI et sa sandbox gratuite.

Amplify va plus loin grâce à la disponibilité d'AWS Amplify Studio. Ainsi, vous pouvez transformer un design graphique créé dans Figma (un outil communément utilisé par les designers) en un composant React (au pixel près) et connecter ces composants au cloud en quelques minutes. Dans ce didacticiel, on vous explique comment créer une application de blog avec Amplify Studio à partir de zéro et quasiment sans écrire de code.

Voici le résultat final : **Figure 1**

Maintenant, passons au monde réel avec une prise en main.

Créer un backend d'application

Tout d'abord, créons un backend pour notre application. Vous pouvez vous diriger vers Amplify Sandbox pour commencer :

1. Connectez-vous sur https://sandbox.amplifyapp.com/ A ce stade, il n'est pas nécessaire d'avoir un compte AWS.
2. À partir de là, choisissez de construire un modèle vierge (**"Blank data model"** dans la console) avec **"React"**.
3. Cliquez ensuite sur **"Get started"** pour commencer.

Figure 2

Sur la page de modélisation des données, vous pourrez créer un modèle. **Figure 3**

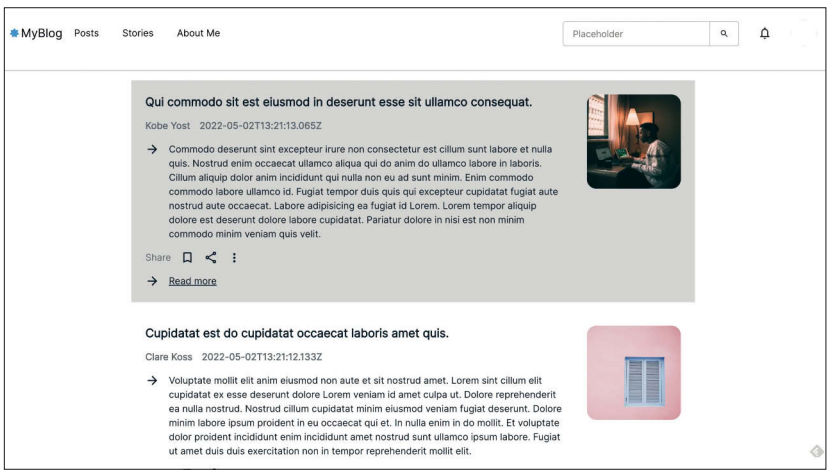
Déployez ensuite sur AWS en cliquant sur **"Deploy to AWS"**

Notez que vous pouvez tester les données localement dans votre application sans compte AWS en cliquant sur « **Next : Test locally in your app** » à la place. Par souci de brièveté, je vais ignorer cette option. Le déploiement sur AWS requiert un compte AWS.

1. Connectez-vous à votre compte AWS ou créez-en un nouveau.
2. Choisissez ensuite une région AWS et un nom pour votre application.
3. Cliquez sur **"Confirm deployment"** pour initier le déploiement.

Le déploiement prend quelques minutes. Sous le capot, pour exposer notre modèle de données, Amplify utilise AWS AppSync pour exposer une API GraphQL et Amazon

Figure 1



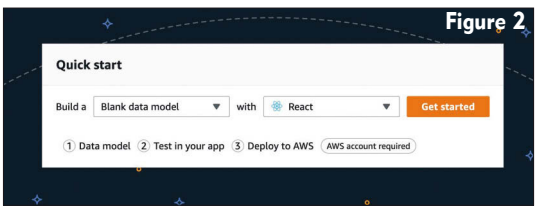


Figure 2

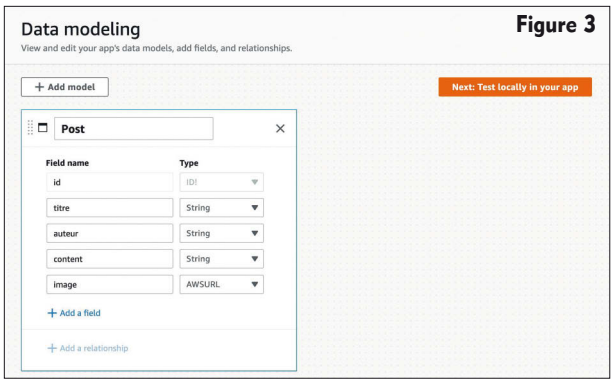


Figure 3

Figure 5

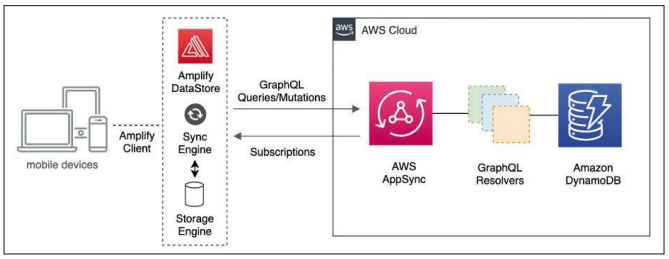


Figure 6

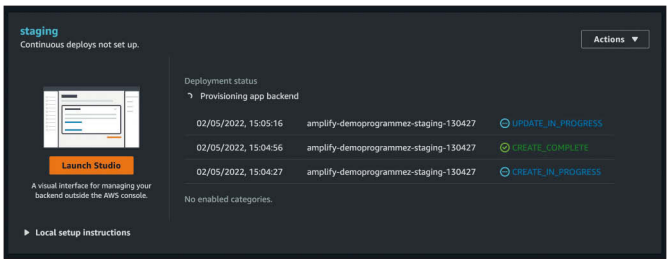


Figure 7

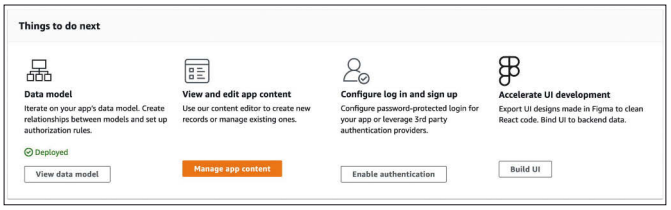


Figure 8

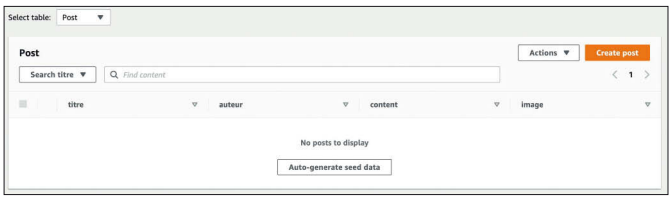
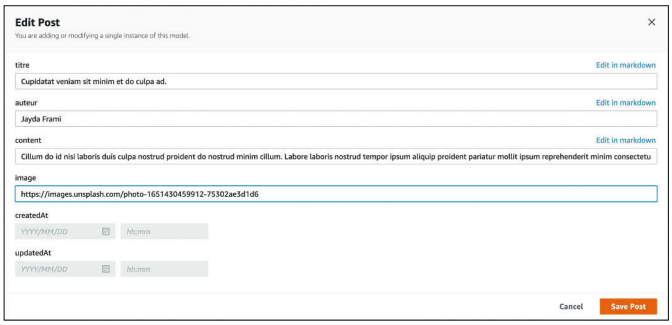


Figure 10



DynamoDB pour la persistance des données en base NoSQL. Voici un schéma montrant ce qui est déployé coté AWS : **Figure 5**

Une fois votre application déployée, cliquez sur "**Launch Studio**" pour lancer Amplify Studio. **Figure 6**

Puis cliquez sur "**Manage app content**" pour gérer le contenu de l'application. **Figure 7**

Vous pouvez cliquer sur "**Auto-generate seed data**", pour crée plusieurs posts aléatoirement. **Figure 8**

Vous voudrez probablement remplacer les URL des images par de vraies images. Je recommande d'utiliser https://unsplash.com pour les trouver. Sélectionnez l'URL de l'image, puis utilisez ce lien dans votre champ image dans votre modèle de données. **Figure 10**

L'interface utilisateur

Maintenant que nous avons un backend d'application, passons à notre interface utilisateur.

Avec Amplify Studio, vous pouvez importer des composants à partir de l'outil de conception d'interface utilisateur Figma. Amplify fournit un https://www.figma.com/community/file/1047600760128127424 à utiliser qui correspond à la https://ui.docs.amplify.aws. **Figure 11**

Vous pouvez dupliquer le fichier d'interface utilisateur en cliquant sur "**Duplicate**", puis vous pouvez apporter des modifications stylistiques aux composants que vous souhaitez.

Passez à l'onglet comprenant vos composants en cliquant sur "**My Components**" pour apporter des modifications.

Par exemple, vous pouvez mettre en gras la date et le nom de l'auteur sur le composant "**SocialPost**". **Figure 12**

Copiez le lien de votre fichier Figma depuis votre navigateur, il vous sera utile pour l'importation dans Amplify Studio. **Figure 13**

Ensuite, vous pouvez vous rediriger vers Amplify Studio et l'onglet "**UI Library**" puis cliquer sur "**Get started**" pour commencer à utiliser ces composants graphiques.

Collez le lien vers votre fichier Figma, puis cliquez sur

Figure 11

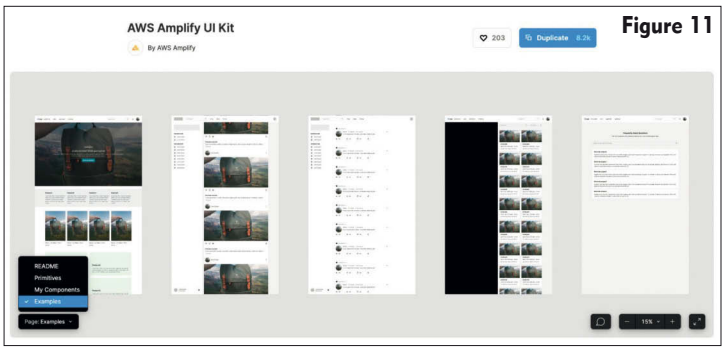


Figure 12

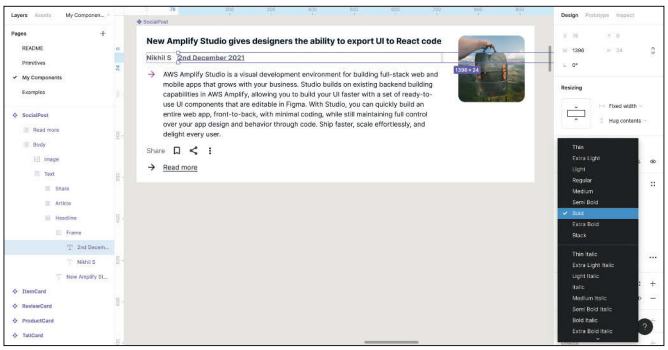


Figure 13

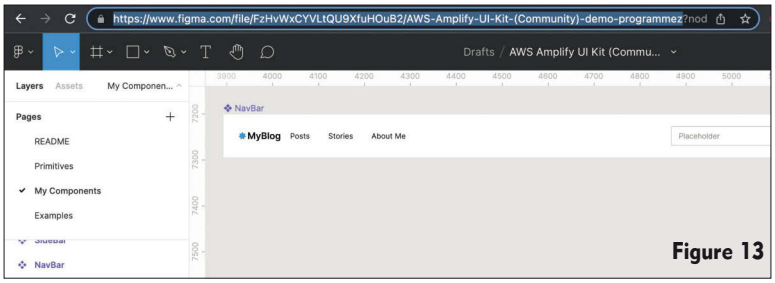
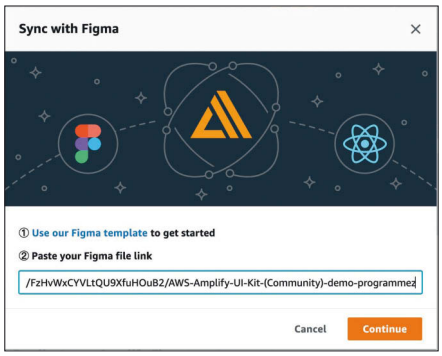


Figure 14



"Continue". **Figure 14**

Synchronisez votre fichier Figma en cliquant sur "**Sync Figma**" en haut à droite et autorisez l'accès à Amplify Studio en cliquant "**Allow access**". Ensuite, vous pouvez afficher vos composants d'interface utilisateur dans Studio ! Vous pouvez accepter tous les changements en cliquant sur "**Accept all changes**" pour importer le thème et "**Accept all**" pour importer tous vos composants depuis Figma vers Amplify Studio.

Lier les données aux composants de l'interface utilisateur

Maintenant, connectons les données et les composants de l'interface utilisateur. Vous pouvez sélectionner le composant "**SocialPost**" dans Studio, puis cliquer sur "**Configure**". **Figure 15**

Ajoutez ensuite une propriété de composant. Vous pouvez l'appeler post, puis mettre le type à Post (celui-ci a été directement exporté de votre modèle défini au départ). **Figure 16**

Ensuite, vous pouvez sélectionner la partie du composant que vous voulez mettre à jour et choisir les données auxquelles vous voulez les lier.

Par exemple, vous pouvez sélectionner l'en-tête de la carte, puis vous pouvez choisir l'attribut label et le lier à post.titre. Vous devriez voir un de vos titres ajoutés lors de la phase de création de contenu apparaître dans l'aperçu. **Figure 17**

Vous pouvez changer les données d'aperçu pour voir différentes données connectées à votre composant en cliquant sur "**Shuffle preview data**"! **Figure 18**

Créer une collection

Au lieu de simplement afficher un article de blog à la fois, vous voudrez probablement en afficher une liste. Vous pouvez alors créer une collection d'articles de blog. En haut à droite, cliquez sur "**Create collection**" pour créer une collection puis choisissez-lui un nom. Par exemple "PostCollection". **Figure 19**

Vous pouvez également ajouter une barre de recherche ou une pagination n'hésitez pas, il suffit d'un clic sur "Search" ou/et "Pagination" !

Vous pouvez ensuite trier ou filtrer vos données afin que seuls les enregistrements que vous souhaitez soient affichés ! Dans un blog du monde réel, vous voudrez peut-être des brouillons et des articles publiés - vous pouvez l'utiliser pour n'afficher que ceux qui ont été publiés !

Ajoutez votre composant à votre application

Maintenant, vous pouvez l'ajouter à notre application ! Tout d'abord, vous devez créer une application React en ouvrant le terminal de votre ordinateur et en exécutant la commande suivante :

```
npx create-react-app amplify-studio-programmez-demo
cd amplify-studio-programmez-demo
```

(Si la première commande utilisant l'outils npx ne réussit pas, il faudra peut-être installer NodeJS en suivant les instructions officielles disponibles https://nodejs.org/fr/)

Ensuite installez la ligne de commande Amplify

```
npm install -g @aws-amplify/cli
```

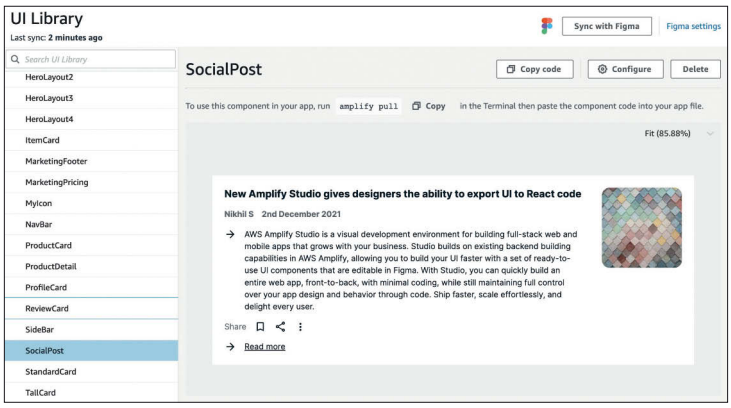


Figure 15

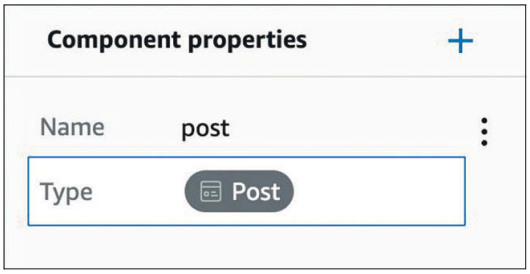


Figure 16

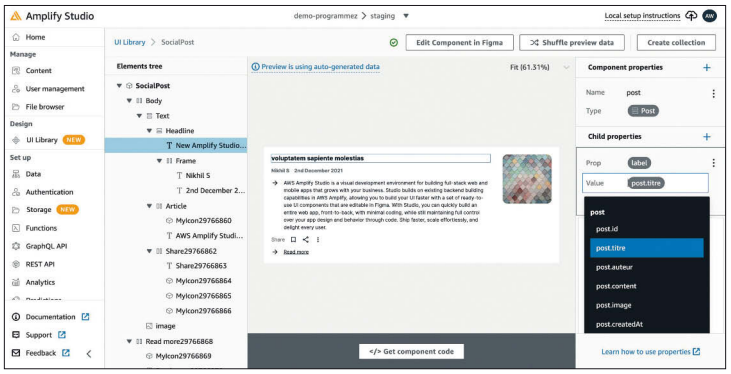


Figure 17

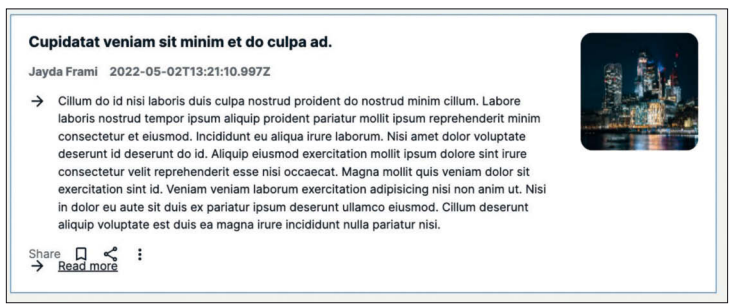
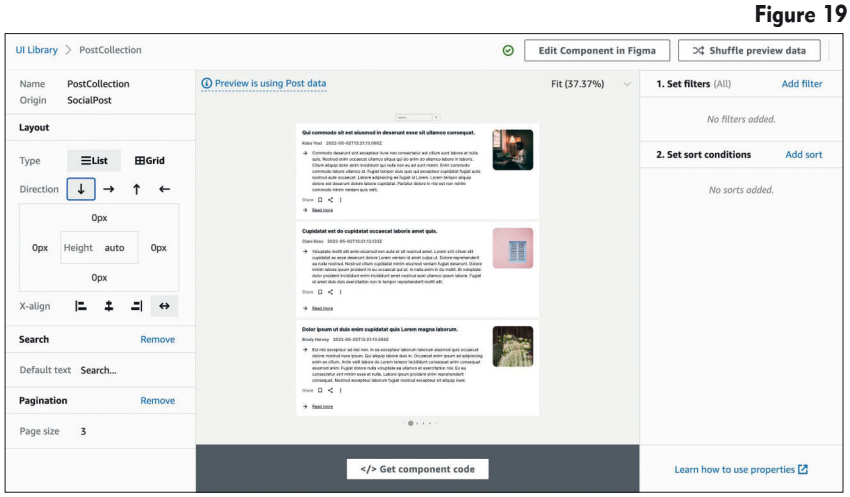


Figure 18

Figure 19



Enfin, en haut à droite de votre page Amplify Studio, cliquez sur le lien "**Local setup instructions**" pour voir les instructions de mise en place locale.
Il y aura une commande amplify pull avec votre ID d'application. Vous pouvez copier et exécuter cette commande.

```
—> amplify-studio-programmez-demo$ amplify pull —appId
d28xha0ek36x0f —envName staging
Opening link: https://eu-west-
3.admin.amplifyapp.com/admin/d28xha0ek36x0f/staging/verify/
```

Répondez ensuite aux questions dans votre CLI, vous devriez pouvoir accepter les valeurs par défaut pour la plupart des questions.

```
✔ Successfully received Amplify Studio tokens.
Amplify AppID found: d28xha0ek36x0f. Amplify App name is: demo-
programmez-demo
Backend environment staging found in Amplify Console app: demo-
programmez-demo
? Choose your default editor: Visual Studio Code
? Choose the type of app that you're building javascript
Please tell us about your project
? What javascript framework are you using react
? Source Directory Path:  src
? Distribution Directory Path: build
? Build Command:  npm run-script build
? Start Command: npm run-script start
```

Figure 21


```
✔ Synced UI components.
✔ GraphQL schema compiled successfully.
```

Ouvrez le projet dans votre éditeur de code préféré. Vous remarquerez qu'un dossier amplify est maintenant présent il contient les détails de votre backend sous la forme de templates CloudFormation, schéma GraphQL et autres fichiers de configuration vous permettant de modifier et garder une trace de l'état de votre infrastructure.
Vous remarquerez aussi le répertoire /src/ui-components qui lui contient tout votre code React ! Vous pouvez inspecter les fichiers et ainsi retrouver le code correspondant aux différents composants affichés dans Amplify Studio / UI Library.
Vous allez maintenant pouvoir faire en sorte que votre application affiche vos articles de blog en utilisant ces composants.
Tout d'abord, installez les composants Amplify React, les fonts et la bibliothèque principale Amplify :

```
npm install @aws-amplify/ui-react @fontsource/inter aws-amplify
```

Maintenant, connectez votre interface à votre application Amplify en ajoutant le code suivant à votre ficher src/index.js.

```
import config from './aws-exports';
import { Amplify } from 'aws-amplify';

Amplify.configure(config);
```

Ensuite, effacez votre composant src/App.js.
Ajoutez le CSS par défaut des composants graphiques d'Amplify en important @aws-amplify/ui-react/styles.css dans src/App.js.
Importez ensuite le composant AmplifyProvider et votre PostCollection dans ce même fichier.

```
import '@aws-amplify/ui-react/styles.css';
import '@fontsource/inter';
import { AmplifyProvider, View, Flex } from '@aws-amplify/ui-react';

import PostCollection from './ui-components/PostCollection';
```

Le AmplifyProvider transmettra le style Amplify à tous ses composants enfants. Enfin, utilisez le composant PostCollection!

```
function App() {
 return (
  <AmplifyProvider>
    <View>
   <Flex direction="row" alignItems="center">
    <PostCollection />
   </Flex>
  </View>
  </AmplifyProvider>
 )
}

export default App
```

Vos publications doivent s'afficher sur la page. **Figure 21**
Vous pouvez ajouter des propriétés à vos composants afin de les modifier. Par exemple, si vous souhaitez ajouter une pagination à votre collection, vous pouvez procéder comme suit :

```
<PostCollection isPaginated itemsPerPage={5} />
```

Figure 22

Désormais, 5 éléments s'afficheront par page ! Vous pouvez lire toutes les options https://ui.docs.amplify.aws/components/collection.

De plus, si vous ouvrez un fichier de composant, par exemple src/ui-components/SocialPost.jsx, vous remarquerez que chaque composant a une propriété overrides. Vous pouvez utiliser cette propriété pour remplacer les propriétés de ce composant individuel.

Quant aux Collections, comme notre src/ui-components/PostCollection.jsx, elles disposent aussi d'une propriété overridesItems permettant de remplacer les propriétés des éléments listés.

Par exemple, faisons en sorte que le fond d'un Post sur deux soit d'une couleur différente.

Vous pouvez attribuer à la propriété overridesItems une fonction fléchée avec en paramètre son élément et son index.

```
<PostCollection overrideItems={
  ({ item, index }) => ({

  })
}}/>
```

Vous pouvez ensuite simplement définir la propriété à redéfinir:

```
<PostCollection overrideItems={
  ({ item, index }) => ({
    backgroundColor: index % 2 === 0 ? 'white' : 'lightgray',
  })
}/>
```

Figure 23

Plus d'informations sur les remplacements dans Amplify : https://docs.amplify.aws/console/uibuilder/override/#save-form-data

Modifier le thème de votre interface utilisateur

Vous pouvez aussi ajouter un thème à votre interface utilisateur afin d'ajouter des couleurs de marque ou d'autres personnalisations. Vous pouvez utiliser https://www.figma.com/community/plugin/1040722185526429545/AWS-Amplify-Theme-Editor pour modifier la palette de couleurs de vos composants Figma. Vous pouvez également utiliser la https://ui.docs.amplify.aws/theming dans votre code via CSS, des jetons de conception ou des objets JavaScript.

Mettre à jour les composants

Vous voudrez peut-être modifier la conception de vos composants à un moment donné. Dans ce cas, vous pouvez les modifier dans Figma, puis cliquez sur "Synchroniser avec Figma" dans la bibliothèque de l'interface utilisateur ("UI Library" dans la console) de Studio. Vous pourrez prévisualiser vos modifications, puis exécuter amplify pull afin d'obtenir les modifications de conception dans votre application locale !

Conclusion

En une heure, à partir de composants Figma existants, vous avez créé une application Web React. Les champs des composants UI sont automatiquement liés à vos données dans le backend. Le backend est déployé sur un compte AWS avec une API GraphQL (via AWS AppSync) et une base de données NoSQL (Amazon DynamoDB).

Figure 22

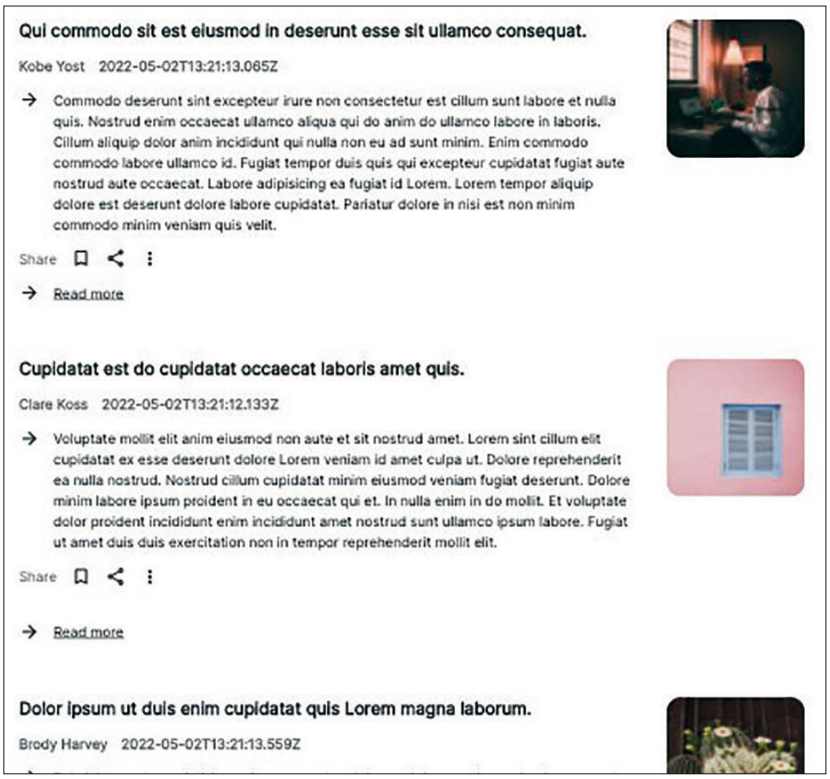
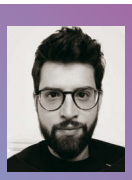


Figure 23

Au-delà d'Amplify Studio, vous pouvez également ajouter un https://docs.aws.amazon.com/amplify/latest/userguide/getting-started.html Web Amplify avec une chaîne de déploiement continu (CI/CD), ou ajouter https://docs.amplify.aws/cli/ telles que des fonctions serverless ou des fonctions de prédictions basées sur l'AIML, et plus encore. Si vous rencontrez des problèmes ou avez des suggestions d'amélioration, n'hésitez pas à soumettre des tickets sur https://github.com/aws-amplify/amplify-adminui !

Bubble : de la théorie à la pratique



Thibault Marty

Co-fondateur et CEO de la société Ottho

Expert Bubble, anciennement formateur pour le Wagon, Epitech, Simplon.

J'ai fait mes armes dans le webmarketing et l'entrepreneuriat avant de plonger à corps perdu dans les outils No-Code comme Bubble.

Créé en 2012 par Emmanuel Strashnov et Joshua Haas, Bubble est l'un des pionniers et leader des outils No-Code. Bubble rend accessible la programmation d'applications et sites Internet complexes sans avoir à coder. Il offre une interface graphique, laquelle permet au plus grand nombre de développer des applications en dessinant l'interface et en définissant des workflows. Avec plus de 2 millions de pratiquants dans le monde, Bubble est devenu incontournable pour quiconque souhaite lancer une application web complète.

Attention cependant, si on ne parle pas d'écrire du code, il y a tout de même une logique de programmation (la base de données, le backend, etc) qui demandera un peu de travail avant de maîtriser l'outil.

Dans le cas de Bubble, on parle plus précisément de programmation visuelle. Elle permet de faire à peu près tout, de manière visuelle. La meilleure analogie que je pourrais donner : ce sont des Lego. Chaque brique représente un élément visuel avec une fonction précise, les blocs sont rassemblés sur des pages, les pages forment l'application.

Dans quels cas utiliser Bubble ?

Sa puissance repose sur son fonctionnement et sa base de données qui lui permettent de couvrir un très large éventail d'usages, autant

- Une plateforme e-commerce à la Amazon,
- Une Marketplace à la Airbnb
- Un Réseau social à la Instagram
- Une Plateforme d'éducation à la Udemy
- Un Système de réservation à la booking
- Un Dashboard à la Google Analytics

NDLR : *Bubble peut quasiment tout faire et s'adresse à des usages très larges.*

Avec son système de connexion API, et les services annexes, il est aussi possible de couvrir des cas particuliers :

- Transformer l'application web en application mobile ou desktop via un système d'encapsulage
- Utiliser Bubble en front en utilisant des bases de données externes
- S'interfacer avec un produit déjà codé via les API

Avantages et fonctionnalités

Outre le fait que ce soit un outil No-Code, Bubble propose un système de programmation visuelle. Comme évoqué plus haut, l'intérêt de ce système est de pouvoir ajouter des briques de fonctionnalités pour gérer des applications complexes. Mais ce n'est qu'un des avantages qui rend Bubble aussi plaisant.

Quels sont les avantages de Bubble ?

- Éditeur drag and drop avec positionnement libre
- Système de Workflow qui permet de faire des boucles de fonctionnalités en fonction des éléments affichés sur la page
- Base de données sous forme visuelle et accessible en temps réel
- Marketplace de plus de 1000 plug-ins gratuits et payants pour ajouter des fonctionnalités
- Éditeur de plug-ins pour que les développeurs puissent ajouter des fonctions en code
- Créateur d'API qui permet de créer facilement des liens avec d'autres applications existantes
- Gestion du Backend permettant des actions au niveau du serveur du Bubble (comme programmer des actions futures)
- Système de géolocalisation (maps)
- Gestion des rôles et permission (Data Privacy)
- Versionning (gérer plusieurs versions de son application)
- Multicompte permettant l'ajout de collaborateur sur son application
- Dashboard de gestion du serveur (avec visualisation des performances)
- Moteur responsive et gestion spécifique du mobile
- Marketplace de template pour l'achat de templates préconstruits

En somme, Bubble permet de faire une passerelle entre le code et le No-Code et donc de pouvoir intégrer des fonctionnalités supplémentaires qui ferait défaut à l'application d'origine.

Comparaison Bubble Vs WordPress Vs Webflow

Fonctionnalités	Bubble	WordPress	Webflow
Éditeur drag and drop	Oui	Plug-ins	Oui (non libre)
Workflow	Oui	Non	Non
Base de données visuelles	Oui	Non	Oui
Marketplace Plugins	Oui	Oui	Oui
Éditeur de plug-ins	Oui	Non	Non
Création d'API	Oui	Non	Non
Backend	Oui	Non	Non
Géolocalisation	Oui	Plug-ins	Non
Hébergement	Oui	Non	Oui
Gestion des rôles et permissions	Oui	Plug-ins	Non
Versionning	Oui	Non	Non
Multicompte	Oui	Oui	Non
Multilingue	Oui	Plug-ins	Non
Dashboard gestion serveur	Oui	Non	Non
Moteur responsive	Oui	Oui	Oui
Marketplace template	Oui	Oui	Oui

Les limites de la plateforme

Bien que Bubble puisse reproduire 90% des applications codées que l'on voit sur Internet, il y a des limites.

Outil propriétaire :

Bubble n'est pas un outil open source. Il n'y a pas d'export de code, ce qui peut constituer un frein si vous souhaitez un contrôle total sur la technologie.

Le scale et la croissance :

Bubble est légèrement plus lent qu'une application native à cause de sa surcouche. Pour des applications comportant moins de 100 000 lignes en base de données, cela reste imperceptible, cependant au-delà, des optimisations et des choix de conception doivent être réfléchis en amont.

Le SEO (le référencement naturel) :

Bubble intègre un certain nombre d'éléments pour disposer d'une visibilité dans les moteurs de recherche (génération de sitemap, ajout de balises dans le header, fichier robots.txt...). Cela demande une attention particulière lors de la création de votre application et des bases en référencement naturel afin d'administrer les éléments essentiels. Le moteur responsive est en question au niveau de la vitesse de chargement des pages, l'un des facteurs de positionnement. Bubble propose depuis fin 2021, un tout nouveau moteur responsive basé sur les flexbox qui permet de gommer les défauts du précédent moteur.

Maîtrise et niveau de complexité :

Nous recommandons chaudement, même aux personnes ayant des compétences en développement, de se former au préalable avant de pouvoir délivrer des applications performantes et sécurisé à de potentiels clients. Cette expertise, bien que plus rapide, peut être atteinte en 3 mois temps plein pour un profil développeur et jusqu'à 6 mois pour un profil totalement novice.

Quels débouchés freelance et métier sur Bubble ?

Avec l'essor du No-Code, c'est tout un pan du marché du développement qui s'ouvre. Aujourd'hui, on compte de plus en plus de missions sur des sites de freelance comme Malt ou codeur.com et cela va en s'intensifiant. C'est d'ailleurs établie comme une tendance par Malt pour 2022.

Cela ne s'arrête pas au freelancing, sur des plateformes tel que Welcome to the Jungle, c'est plus de 300 postes en CDI proposés pour du No-Code dont près de la moitié sont dédiés uniquement à Bubble. Cela s'explique notamment par les levées de fond des startups s'étant intégralement développées à partir d'outils comme Bubble. On peut citer des boîtes comme Cuure (1,8 M€), On Train (1,2M€) et Prello (13M€). Ces boîtes sont actuellement en recherche de Bubblers chevronnés mais aussi de Product Manager No-Code pour assurer le développement de leur produit.

Enfin, un nombre incalculable d'agences se lancent dans la course comme Alegria, TinkSo, Cube ou Evodev, dont les besoins en développeur Bubble sont grandissants.

SE FORMER À BUBBLE

A-t-on besoin de connaissances techniques pour apprendre Bubble ?

S'attaquer à Bubble lorsqu'on n'a aucune connaissance technique peut faire peur, surtout quand on ouvre l'éditeur pour la première fois, mais cela ne doit pas vous rebuter pour autant. Même si la logique peut vous paraître abstraite au début, un apprentissage en douceur est possible.

Comment apprendre gratuitement Bubble ?

Il est possible d'apprendre gratuitement Bubble en vous baladant sur le forum officiel de Bubble, en regardant des vidéos gratuites sur YouTube et en étant très débrouillard. Attention cependant à ne pas vous perdre, car la compréhension des concepts de fonctionnement nécessite des méthodes.

Apprendre rapidement Bubble via une formation

Ottho propose des formations intensives 100% en ligne à travers 2 formats :

- Bootcamp, en temps plein de 4 semaines
- Cohorte, en temps partiel de 6 semaines.

C'est le moyen le plus rapide d'acquérir la maîtrise de Bubble à travers :

- Plus de 10 exercices dont 3 mini projets (Blog, Todo List, panier d'achat)
- Une marketplace à la Airbnb
- La construction d'un projet complet type M.V.P

C'est le meilleur format d'apprentissage que l'on peut vous proposer aujourd'hui et qui a aidé plus de 440 personnes et en moyenne après 3 mois :

- 51% des apprenants créés leur entreprise
- 49% réalisent leur première mission (freelance ou métier)

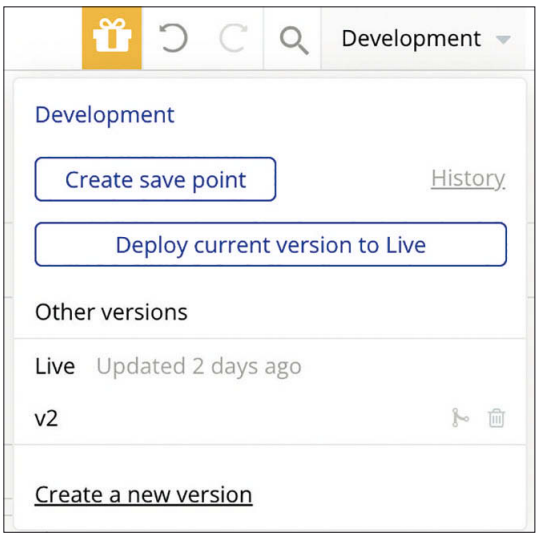


Figure 1

Versionning

La logique de développement se rapproche de la programmation classique avec :

- Une version "development" qui correspond à un environnement de développement avec une base de données spécifique.
- Une version "live" qui correspond à un environnement de production, et la base qui correspond.
- Chaque mise en ligne se fait par une fonction "deploy" qui va mettre à jour l'environnement de production.

Figure 1

L'INTERFACE DE BUBBLE

L'interface de Bubble se compose de 3 menus principaux :

- **Design** : un éditeur WYSIWYG pour construire votre page
- **Workflow** : un système d'événements et d'actions pour gérer les interactions
- **Data** : L'accès à la base de données

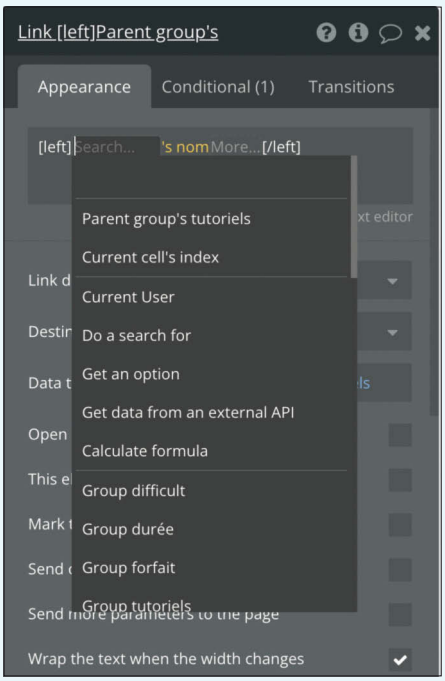


Figure B

Partie Design

Niveau design, le constructeur se compose :

- **D'éléments visuels** pour créer des éléments graphiques type texte ou bouton
- **De containers** pour grouper les éléments et structurer l'affichage
- **D'input forms** pour enregistrer la donnée

Figure A

Chaque élément se dépose par drag&drop. En cliquant sur un élément, vous avez accès à un menu contextuel. **Figure B**

À l'image d'un framework, Bubble vous donne accès à des raccourcis de fonctions vous permettant d'ajouter des fonctionnalités dynamiques à votre élément (par exemple : appeler un item de votre base de données). **Figure C**

Partie Workflow

Les workflows permettent de gérer les actions de votre application.

Ils comportent 2 éléments principaux :

- Les events qui sont les déclencheurs de l'action
- Les actions qui sont les résultantes du déclencheur.

Les actions sont déjà pré-configurées et classées par thématique, vous permettant de couvrir à peu près tout.

L'ajout de plug-ins vous permet d'ajouter des actions supplémentaires. **Figure D**

Partie Data

À l'instar d'autres outils No-Code, Bubble intègre sa base de données. La partie Data permet principalement de créer ses tables et de visualiser les entrées des différentes tables.

Data type

Chaque table comprend ses propres fields qui correspondent à une propriété de la donnée.

Elle est définie par 3 éléments :

- Son nom qui correspond à son appellation.
- Son type, qui va définir la typologie de données attendues.
- Son "caractère" : unique si la donnée attendue est unique, "list" si la donnée comprend plusieurs éléments.

Tips : Il est aussi possible de définir de relation entre les tables à partir de ce menu. **Figure E**

App Data

Dans cet onglet, vous pourrez visualiser les entrées de chaque table de votre base de données, les modifier/supprimer à loisir.

Bubble intègre un système de sauvegarde de l'état de la base de données vous permettant de restaurer une ou plusieurs tables à une date antérieure. **Figure F**

Intégrité de la base de données

Afin de maintenir la base de données toujours fonctionnelle, la suppression d'une table n'entraîne que la suppression de son accès. Ainsi, il est facile de manipuler les différentes tables sans risquer une manipulation irréversible.

Figure A

Figure D

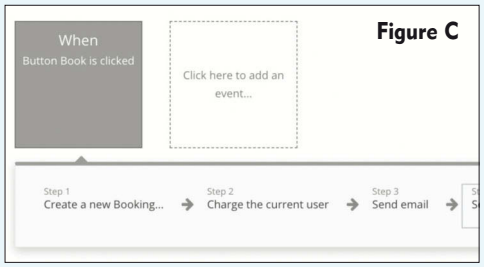


Figure C

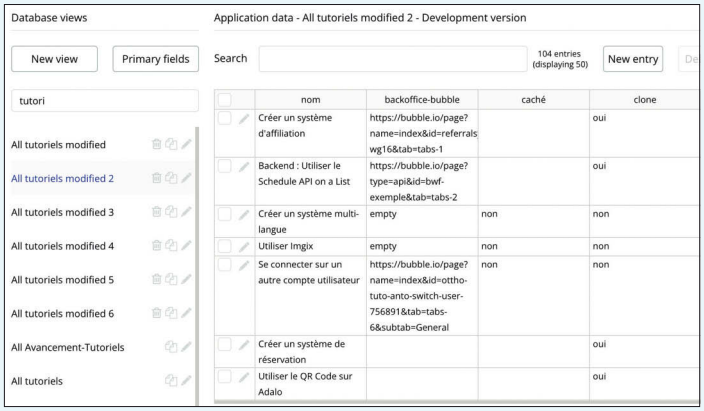


Figure F

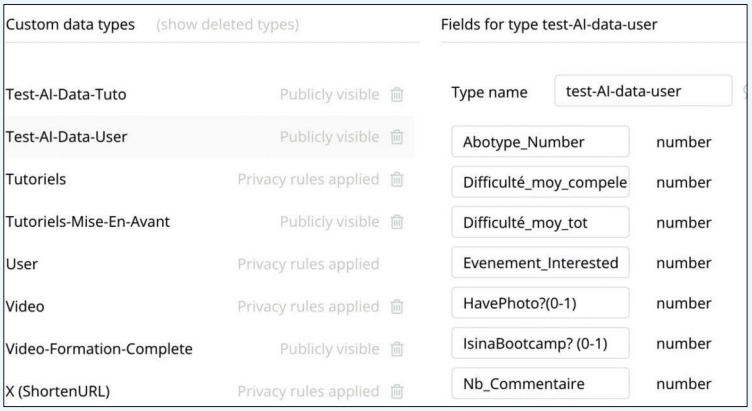


Figure E

La logique de Bubble

Pour utiliser Bubble, il est capital de bien comprendre sa logique :

- La partie visuelle permet d'afficher l'UI
- La partie workflow permet de gérer la logique
- La partie data permet de gérer la donnée.

Enregistrement de la donnée **Figure 2**

Une fois vos tables de données configurées, vous pouvez directement passer sur la partie visuelle afin d'enregistrer vos premières données.

Ici nous allons rentrer les données grâce aux éléments Input Form, et nous allons les enregistrer via un élément Button, sur lequel nous allons construire nos workflows.

L'Event du worklow correspond au clic de l'utilisateur sur le bouton.

Les actions d'enregistrement vont permettre de créer une nouvelle entrée dans la table à partir des informations comprises dans les champs Input.

Tips : chaque input comprend le type (la propriété) de la donnée attendue. **Figure 3**

L'enregistrement pourra se vérifier directement dans la partie App Data.

Afficher la donnée

Une fois enregistrée, la donnée peut être appelée directement sur l'interface visuel, directement à partir des formules disponibles dans le menu contextuel ou via un workflow.

Il est aussi possible d'appeler une liste de données. Dans ce cas, Bubble intègre un conteneur spécifique : le repeating group. Il permet de gérer un tableau permettant d'appeler une liste de données grâce à la fonction "do a search".

Tips : il faut différencier l'appel de la donnée de son affichage. Quand vous chargez la donnée dans un repeating group, vous la rendez disponible, mais pas directement visible. Pour ce faire, il faut ajouter un élément visuel. Dans le cas où l'on souhaiterait afficher le nom de l'élément enregistrer, il faut ajouter un élément de type text afin d'afficher son contenu à travers la fonction "current cell x text". **Figure 4**

Le No-Code, une opportunité pour rassembler des mondes trop longtemps éloignés.

Le terme No-Code peut prêter à confusion car il porte l'idée d'opposer le code et le no-code. En réalité il n'en n'est rien, jamais la communication entre les techs et les non techs n'aura été aussi fluide, car les apprentissages de no-code sont emprunt des logiques du code, à savoir la programmation. Et de fait, ces 2 mondes peuvent enfin se parler sans avoir besoin d'un traducteur. Ils peuvent enfin se rassembler autour du produit.

Certes tout n'est pas parfait, le no-code, mal géré, peut accumuler de la dette technique, mais à de nombreux stades de développement d'une entreprise, cela reste un choix tout à fait louable.

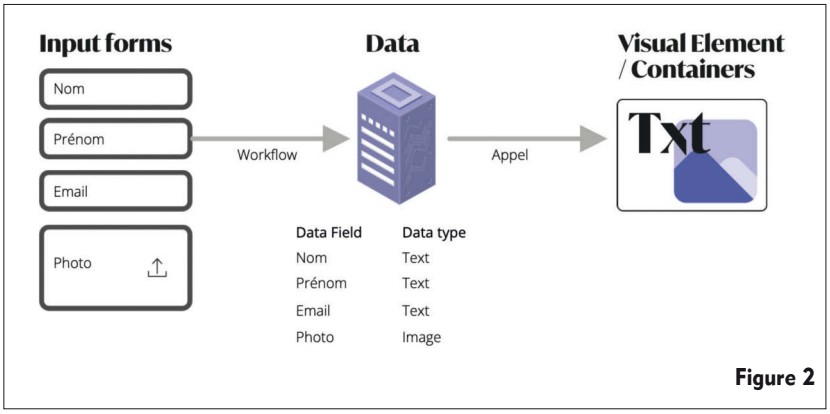


Figure 2

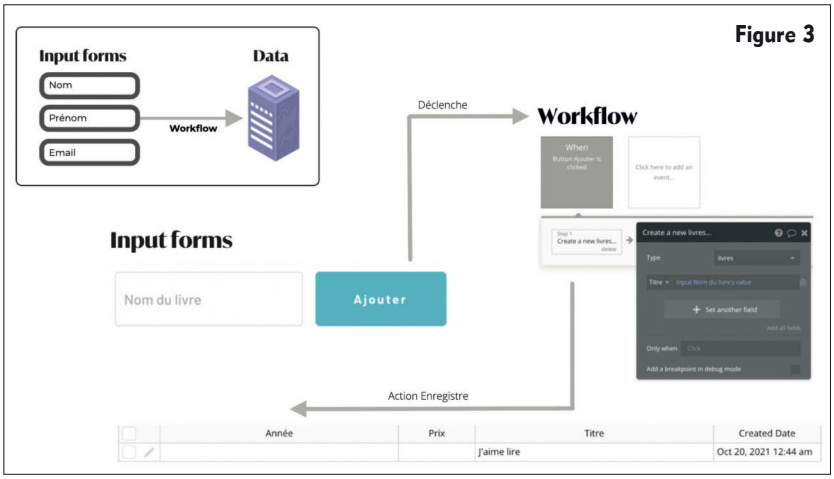


Figure 3

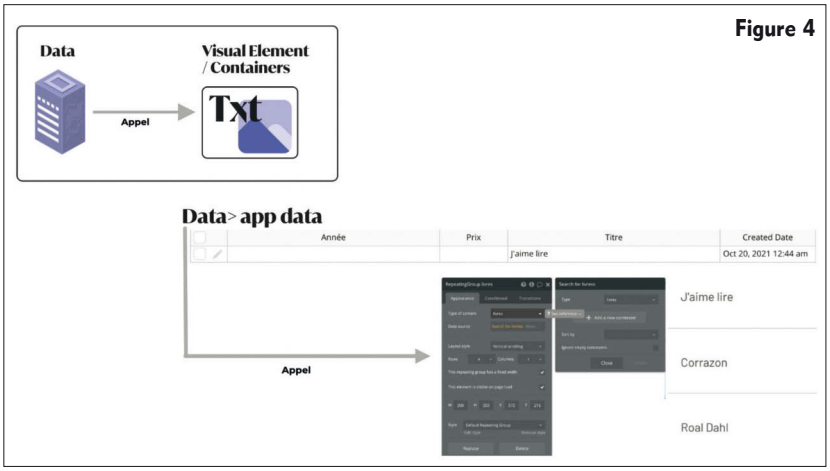


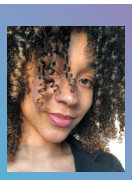
Figure 4

D'un autre côté, cela permet à de nombreuses personnes n'ayant pas choisi la voie du code, de pouvoir créer des applications à leur mesure, sans pour autant devoir investir des sommes folles dans une agence.

Enfin, si nous poussons les limites de la réflexion à d'autres contrées comme l'Afrique. Le No-Code est une formidable opportunité d'émancipation, là où la formation en code, supposée longue (et donc coûteuse à moindre mesure), reste un frein à toute forme de création numérique.

Power Platform et l'approche du développement de fusion

Un Citizen Developer est un utilisateur capable de maîtriser les outils de Power Platform sans pour autant avoir de notions de programmation. Son profil est plus orienté métier, mais il doit aussi posséder des connaissances d'Office 365 (Excel, Access, PowerPoint, etc.). Grâce à cette solution No Code, il a la capacité de créer des applications rapidement et à moindre coût.



Émilie Sanchez
Conceptrice développeuse – SQLI

SQLI DIGITAL EXPERIENCE

Pourtant la maîtrise des outils Power Platform par un *Citizen Developer* a ses limites : selon la complexité du besoin métier, il peut être amené à faire appel à un développeur « old school ». On appelle ça le développement de fusion...

Qu'est-ce que le développement de fusion ?

L'approche du développement de fusion consiste à associer le travail d'un *Citizen Developer* avec celui d'un développeur, mais aussi d'utilisateurs finaux, afin de créer des équipes de fusion numérique et ainsi de mener à bien les projets à réaliser sur Power Apps dans Power Platform.

Power Platform est une solution Microsoft pour permettre de centraliser les données d'une entreprise dans un environnement afin d'effectuer trois actions distinctes :

- Manipuler les données avec Power Apps,
- Automatiser avec Power Automate,
- Et analyser avec Power BI.

L'approche du développement de fusion s'applique généralement avec Power Apps, mais aussi avec Power Automate et Power BI. Dans cet article, nous allons découvrir plusieurs cas qui prouvent la nécessité de constituer une équipe de fusion numérique dans des projets à réaliser sur Power Platform.

Cycle de travail d'une équipe de fusion numérique

Dans une équipe de fusion numérique, chacun à son rôle à jouer durant la réalisation d'un projet dans Power Platform. Dans un premier temps, le *Citizen Developer* réalise une application sur Power Apps selon des besoins métiers spécifiques.

Si les tâches s'avèrent trop compliquées, il pourra faire appel à un développeur qui s'occupera alors des développements complexes.

Par exemple, le développeur se chargera de la création d'API Web et de leur implémentation dans Power Platform avec des connecteurs personnalisés qui récupéreront les données métier. Il sera également amené à mettre en place une approche algorithmique dans la création de flux automatisés. Et avec ses connaissances en programmation, il pourra intégrer du Python dans des rapports Power BI.

Les utilisateurs finaux eux, auront la responsabilité de tester l'application et de remonter des feedbacks directement au *Citizen Developer.* **Figure 1**

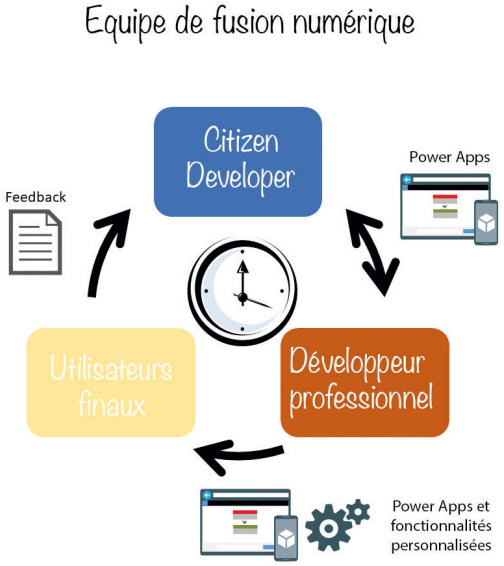


Figure 1 : *Cycle de travail d'une équipe de fusion numérique*

Création des connecteurs personnalisés

Il existe, dans Power Platform, des composants appelés connecteurs et permettant de centraliser les données métiers dans un environnement. Actuellement, il en existe plus de 300, mais il est possible que certains ne répondent pas aux besoins métiers du Citizen Developer.

La tâche du développeur professionnel sera donc de créer un connecteur personnalisé de différentes façons :

- À partir de zéro,
- À partir du service Azure,
- Avec un document OPEN API,
- Avec une collection Postman,
- À partir de GitHub.

Création de connecteurs personnalisés à partir de zéro :

Dans cet exemple, les développeurs ont implémenté un connecteur personnalisé de zéro qui récupère, via une API Web, les données météorologiques du site https://openweathermap.org/. **Figure 2**

Power Fx, utilisé principalement dans les applications canevas Power Apps, est un langage similaire aux formules utilisées dans Excel.

Dans l'exemple ci-dessous, le *Citizen Developer* fait appel au connecteur personnalisé pour récupérer les données de l'API Web *OpenWeatherMap*. **Figure 3**

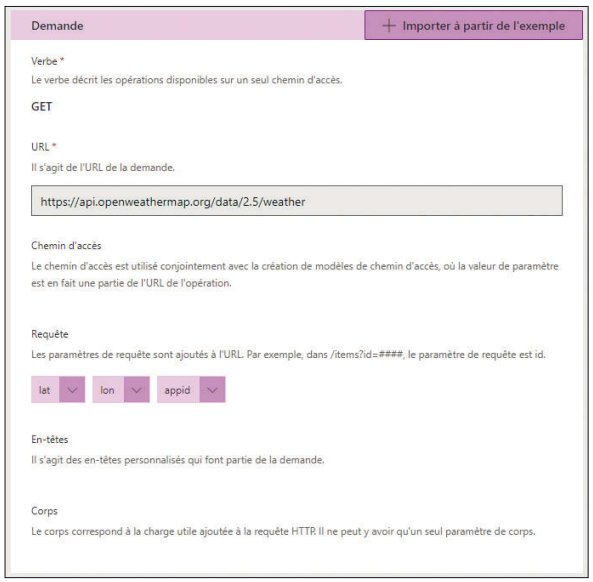


Figure 2 : *Interface de création d'un connecteur personnalisé à partir from scratch*

Figure 3 : *Récupération de données via un connecteur personnalisé dans une application Canevas Power Apps*

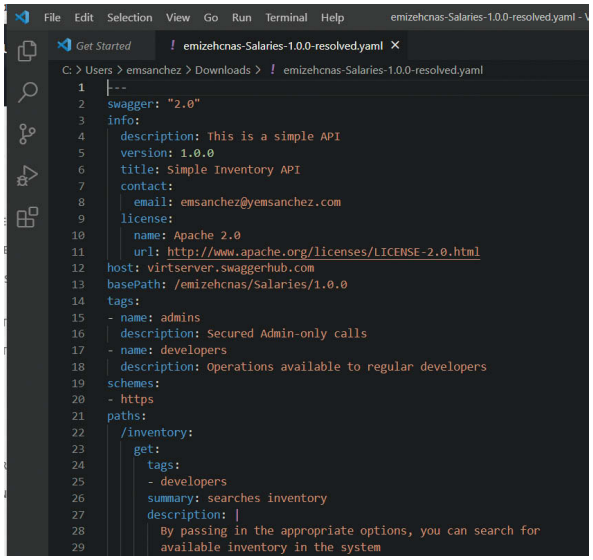


Figure 4 : *Exemple de fichier OPEN API au format YAML*

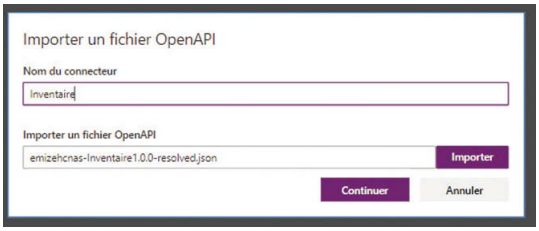


Figure 5 : *Importation d'un document OPEN API dans Power Platform pour la création d'un connecteur personnalité*

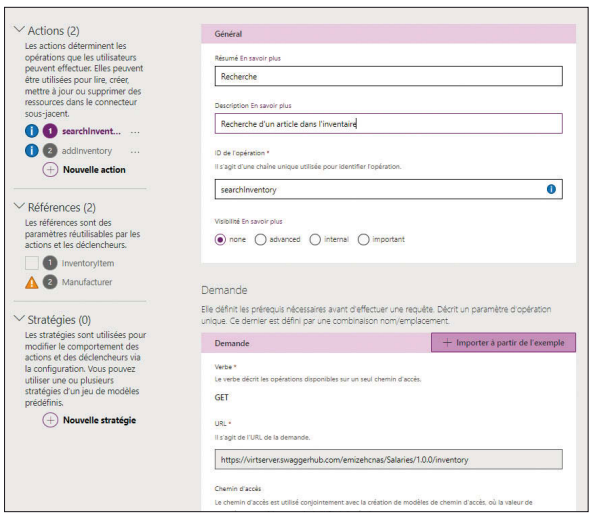


Figure 6 : *Interface de paramétrage d'un connecteur personnalisé avec OPEN API*

SwaggerHub pour générer une documentation OPEN API

Il est aussi possible de créer un connecteur personnalisé avec la plateforme de développement *SwaggerHub*. Cet outil permet de créer, documenter, gérer et déployer des API Web.

La documentation OPEN API est nécessaire à la création d'un connecteur personnalisé dans Power Platform. Il s'agit soit d'un fichier YAML ou JSON contenant les informations principales de l'API Web. **Figure 4**

Dans cet exemple, le document OPEN API est directement importé dans l'environnement Power Platform : **Figure 5**

Lors de l'importation, les paramètres de l'API Web sont automatiquement remplis ainsi que les différentes méthodes de requête : *La requête searchInventory permettra à l'utilisateur de rechercher un article dans l'inventaire (GET) et la requête addInventory pour ajouter un article (POST)* **Figure 6**

De leur côté, les *Citizens Developers* n'auront plus qu'à utiliser le connecteur personnalisé dans Power Automate ou Power Apps. Par exemple, pour rechercher un article ou en ajouter un dans l'inventaire. **Figure 7**

Les approches algorithmiques dans Power Automate.

Power Automate est l'outil permettant d'automatiser les tâches dans le but d'améliorer la productivité d'une organisation. Dans un flux, le *Citizen Developer* définira un déclencheur. Ce déclencheur récupérera des données métier et ajoutera ensuite des étapes qui effectueront des actions telles que l'envoi d'un e-mail ou bien l'ajout d'un évènement dans un calendrier Outlook.

Bien que l'outil soit facile d'utilisation, il se cache quelques techniques que seuls les développeurs professionnels connaissent.

Les états de type Control regroupent plusieurs actions :

- Des conditions,
- Des boucles : Appliquer à chacun,
- Des Switch : Basculer,
- Des boucles : Exécuter jusqu'à,
- Des portées. **Figure 8**

Intégration d'une clause Try Catch dans un flux

Le développeur peut intégrer une clause *Try Catch* dans un Flux Power Automate. Lorsqu'un flux s'exécute, si l'une des actions est en erreur, alors les autres actions qui la suivent ne s'exécuteront pas. Admettons que le besoin initial vise à récu-

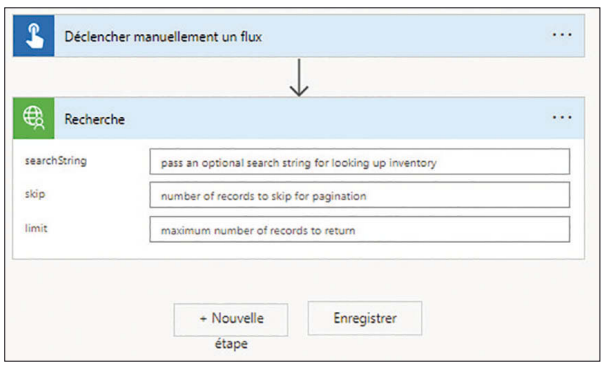


Figure 7 : *Utilisation d'un connecteur personnalisé dans un flux Power Automate*

pérer les propriétés d'un fichier et de les envoyer par e-mail. Si les propriétés du fichier ne sont pas récupérées, alors il faudrait envoyer un e-mail alertant le destinataire que le fichier n'est pas à disposition, dans le cas contraire, le corps du message sera composé du nom du fichier.

Dans cet exemple, les propriétés du fichier n'ont pas été récupérées : l'action « Obtenir les propriétés du fichier » a généré une erreur et l'action « Envoyer un e-mail (v2) » ne s'est donc pas exécutée. **Figure 9**

Nous souhaitons tout de même envoyer un mail en informant le destinataire, même si les propriétés du fichier n'ont pas été récupérées.

Il existe, dans Power Automate, une méthode permettant d'exécuter une action même si la précédente a généré une erreur. **Figure 10**

L'action « Envoyer un e-mail (V2) » s'exécutera même si l'action « Obtenir les propriétés du fichier » génère une erreur.

En sélectionnant le pictogramme et en cliquant sur : « configurer la propriété exécuter après », on peut décider de l'exécution d'une action selon l'état de la précédente :

- Réussie,
- Échouée,
- Ignorée
- Expirée.

Ici, le flux s'est exécuté dans sa totalité et on remarque que l'action « Obtenir les priorités du fichier » génère toujours une erreur. Pourtant, l'exécution des autres actions n'est pas impactée.

Seulement, la logique métier n'est pas bonne : le corps du mail indique que le fichier est à disposition. Or, ce dernier n'a pas été récupéré puisque l'action qui le récupère a généré une erreur. **Figure 11**

C'est là qu'intervient la clause *Try Catch* qui peut être utile pour respecter la logique métier. En algorithme, la clause Try catch fonctionne de cette façon :

- Dans le bloc *Try* : on ajoute le code à exécuter.
- Dans le bloc *Catch*, le code à exécuter si le code dans le bloc Try à générer une erreur.

Dans Power Automate, il est possible de faire la même chose en utilisant l'action ***Etendue***. Cette action permettra d'ajouter plusieurs sous-actions dans cette dernière.

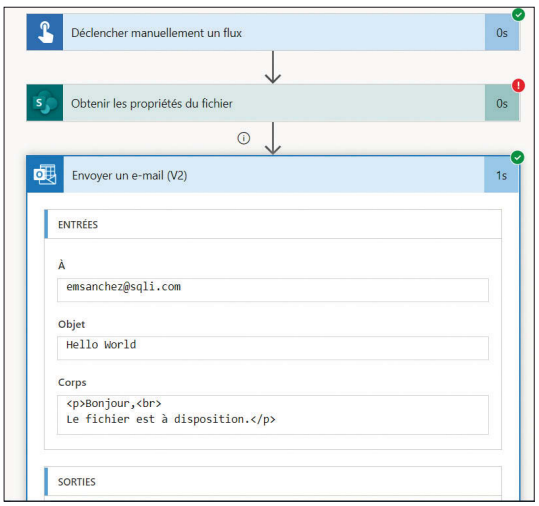


Figure 11 : *Exécution d'un flux où la troisième étape est exécutée bien que la précédente soit en erreur*

Lors de l'exécution du flux, le bloc *Try* ayant échoué, il passe directement au bloc *Catch* afin de récupérer le message indiquant que le fichier n'est pas à disposition. Le développeur a reproduit exactement la même méthode Try Catch comme s'il l'avait codée dans un IDE. **Figure 12**

A l'inverse, si les propriétés du fichier ont été récupérées, alors l'action Catch est ignorée et le corps du message indique que le fichier a bien été récupéré.

On remarque que l'action *Try* ayant réussi, l'action Catch a été ignorée. Le corps du message contient le nom du fichier et s'est bien envoyé. **Figure 13**

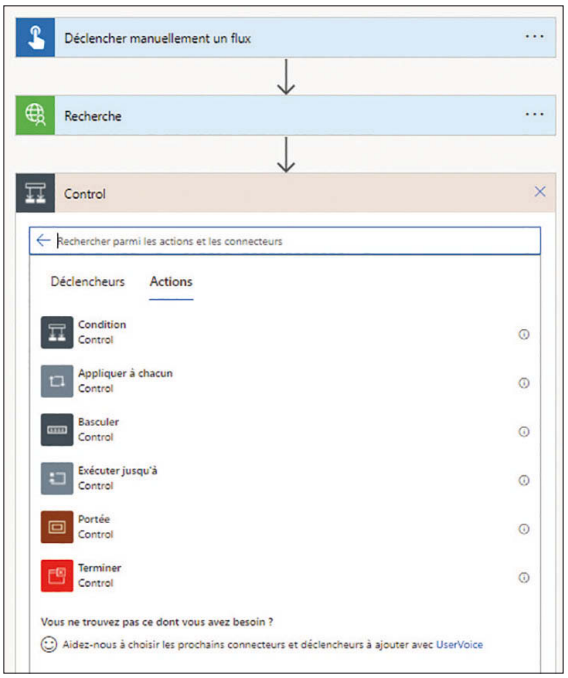


Figure 8 : *Insertion d'une étape de type Control*

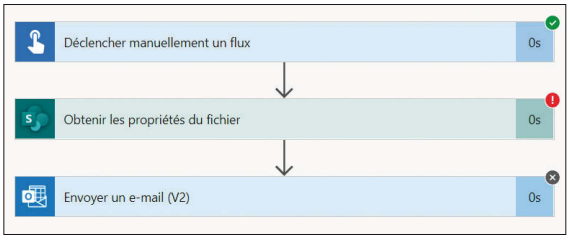


Figure 9 : *Exécution d'un flux en erreur*

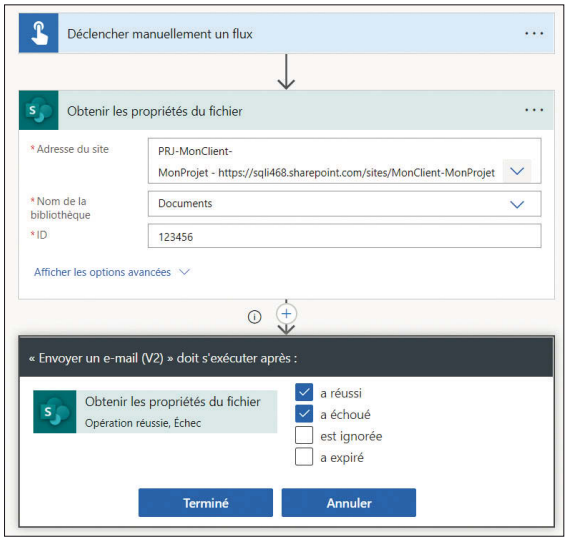


Figure 10 : *Paramétrage de l'exécution d'une étape selon l'état de la précédente*

Figure 12 : *Exécution d'un flux où l'e-mail est envoyé sans les propriétés d'un fichier*

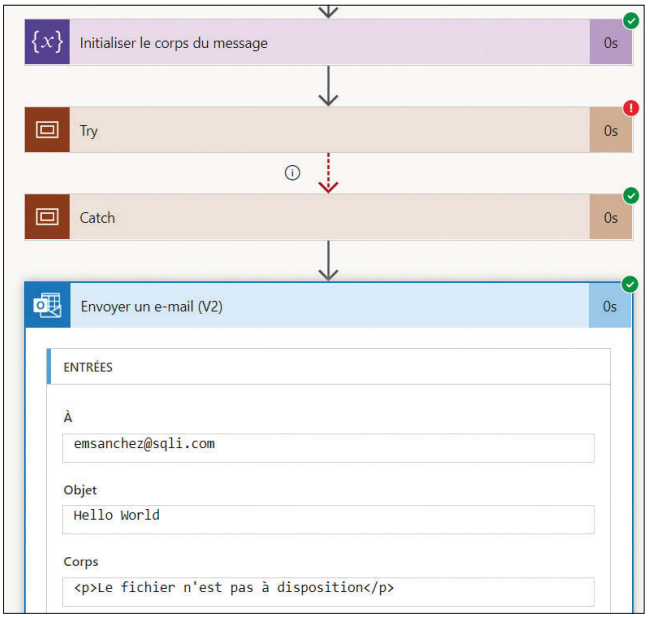


Figure 13 : *Exécution d'un flux où l'e-mail est envoyé avec les propriétés d'un fichier*

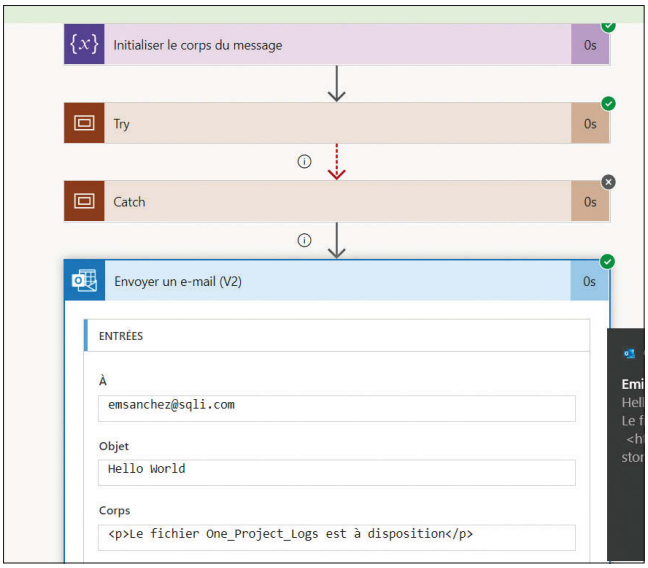
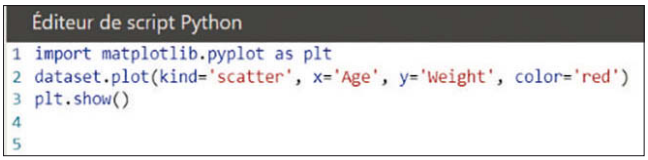


Figure 15 : *Création d'un visuel avec un script Python dans Power BI Desktop*



Python dans Power BI

Power BI est une solution développée pour la création et le partage de rapports et tableaux de bord.

Elle comporte trois principaux composants :

- Power BI Desktop, l'application Bureau permettant de créer des rapports,
- Power BI Service permettant publier et partager des rapports dans l'organisation,
- Les applications accessibles sur les équipements de type tablette et mobile.

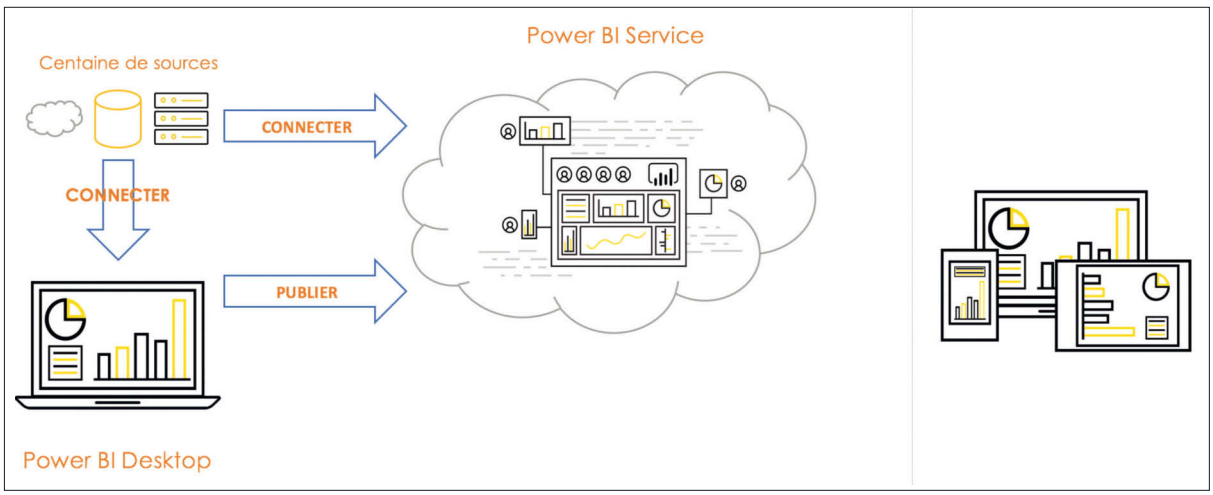
Dans le même principe que Power Apps et Power Automate, Power BI utilise des connecteurs pour centraliser les données métiers dans un modèle et ainsi permettre de créer des rapports. **Figure 14**

Peuvent s'ajouter à l'équipe de fusion numérique, des Data Analystes ou autres métiers travaillant directement dans l'informatique décisionnelle. Power Query est un outil de transformation de données directement intégré dans Power BI Desktop. Il permet entre autres de nettoyer les données métiers qui ont été importées dans le modèle. Les data analystes de l'équipe de fusion numérique ont les compétences de créer des visualisations dans des rapports Power BI Desktop. Elles sont la représentation visuelle des données métiers importés. Cependant, selon leur besoin, ils peuvent faire appel aux développeurs professionnels pour créer des visuels plus complexes à l'aide de scripts Python. **Figure 15**

Conclusion

Souvent, lorsque l'on positionne un développeur de métier sur un projet Power Platform, il peut se retrouver parfois déçu du manque de technique dû au concept du Low code / No Code. Les *Citizen Developers*, quant à eux, risquent de se sentir dépassés par la complexité de la demande. C'est pourquoi, il est important de constituer une équipe de fusion numérique regroupant différents profils aussi bien techniques que métiers. Les *Citizen Developers*, idéalement, devront être à cheval entre la partie technique et la partie métier pour la conception d'applications Power Apps et de flux Power Automate. Les développeurs interviendront uniquement en cas de tâches trop complexes. Dans ce dispositif, n'oublions pas les utilisateurs finaux ! Ces derniers sont voués à tester les applications et à communiquer leurs retours aux *Citizen Developers* dans un objectif d'amélioration des applications.

Figure 14



Power Query : 95% no code, 5% code

D'abord présenté comme outil complémentaire du module PowerPivot d'Excel, Power Query a peu à peu acquis ses lettres de noblesse, jusqu'à devenir un moteur de préparation de données à part entière, moteur que l'on retrouve dans les dernières versions d'Excel et de Power BI Desktop, mais aussi dans les services du cloud Azure (sous le nom de data flows dans Azure Data Factory et Power BI).

Joël CREST
Team Leader chez Exakis Nelite et Microsoft MVP Data Platform

Exakis Nelite
MagellanPartners

L'outil est doté d'une interface permettant de se connecter à des centaines de sources de données différentes, de transformer ces données sans produire une seule ligne de code et de les mixer entre elles avant de les mettre à disposition en mémoire en vue de leur utilisation future. C'est la promesse de Power Query : en quelques clics, l'utilisateur décrit les transformations de données étape par étape. Ces transformations seront ensuite appliquées automatiquement à chaque actualisation des données sources.

Premiers pas en Power Query

Pas une ligne de code, des centaines de transformations

Pour vous guider dans vos premiers pas avec l'outil Power Query, prenons un cas simple : vous souhaitez exploiter des données issues d'une source de données quelconque et les manipuler via Excel.

Dans notre exemple, nous allons nous baser sur un flux OData d'exemple disponible publiquement, mais la démarche peut être adaptée pour n'importe quel fichier source dont vous disposez.

Une fois Excel ouvert, dirigez-vous vers le menu **Données** et cliquez sur le bouton **Obtenir des données**. C'est là que se cache l'entrée dans le monde de Power Query. **Figure 1**

Choisissez le sous-menu **A partir d'autres sources**, puis sélectionnez **A partir d'un flux OData**. Dans la fenêtre qui s'ouvre, saisissez l'URL suivante :

https://services.odata.org/V4/Northwind/Northwind.svc **Figure 2**

Dans la fenêtre d'authentification, conservez le choix **Anonyme** (bien entendu, adaptez les informations d'authentification à votre source de données).

Un premier écran **Navigateur** va vous proposer de sélectionner les tables que vous souhaitez extraire. Pour sélectionner plusieurs tables, veuillez cocher au préalable la case Sélectionnez plusieurs éléments. Dans notre exemple, nous allons sélectionner les tables suivantes : **Order_Details**, **Customers**. **Figure 3**

Deux possibilités s'offrent alors à vous :

- Laisser Power Query faire le job pour vous (démarche on ne peut plus no code !) en cliquant sur le bouton **Chargez**
- Sélectionner **Transformez des données** pour pouvoir éventuellement (toujours en mode no code) modifier les transformations proposées par Power Query.

Nous allons choisir cette seconde option qui va vous mener tout droit vers l'éditeur Power Query. Bienvenue dans le royaume du no code !

Tour d'horizon de l'éditeur Power Query

Une fois l'éditeur Power Query affiché, prenez quelques minutes pour vous familiariser avec les différents éléments qui le composent. À gauche se situe la zone **Requêtes**. Comme son nom l'indique, elle permet d'accéder à chacune des requêtes sources initiales (ici les requêtes de connexion aux tables sélectionnées). Chaque requête est indépendante (du moins en mode no code). **Figure 4**

Figure 4 : *zone Requêtes*

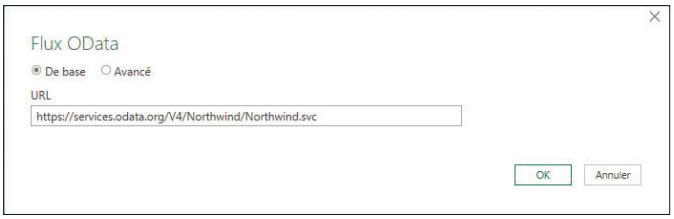


Figure 2 : *paramètres du flux OData*

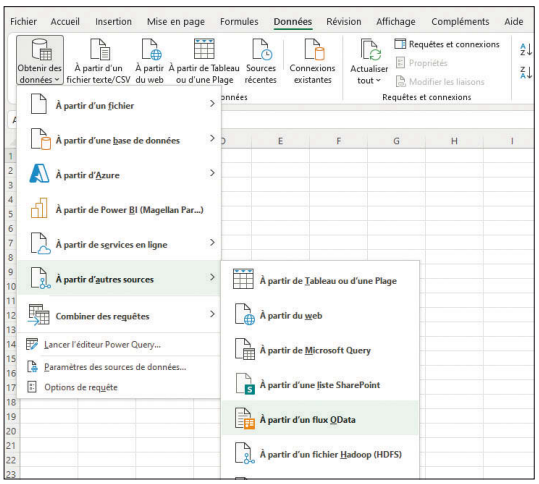


Figure 1 : *obtenir des données*

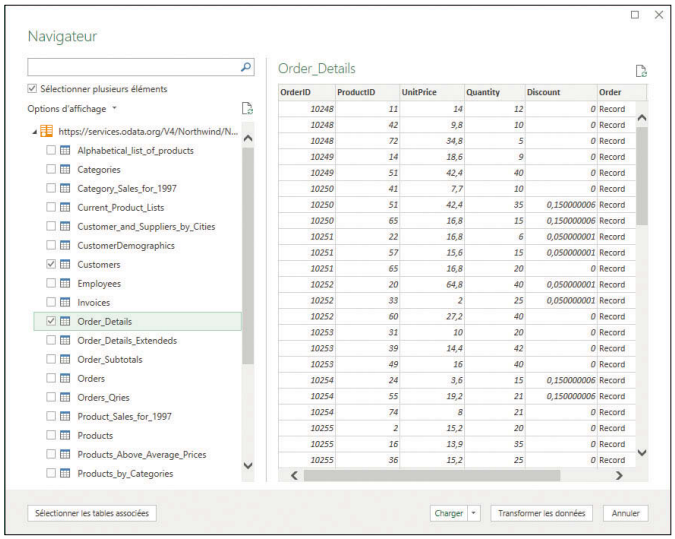


Figure 3 : *fenêtre Navigateur*

Au centre se situe la zone de **prévisualisation des données**. Cette zone affiche uniquement les 1000 premières lignes de la requête sélectionnée, afin d'éviter de faire s'effondrer la mémoire vive de votre poste de travail en cas de connexion à une source exposant plusieurs millions de lignes.

À droite se situe la zone **Paramètres d'une requête**. On y retrouve le nom de la requête sélectionnée, mais surtout une liste d'étapes appliquées. Il s'agit des différentes transformations appliquées sur la source. Il est tout à fait normal de retrouver, pour l'instant, deux étapes nommées **Source** (connexion initiale au flux oData) et **Navigation** (sélection de la table concernée par la requête parmi la liste des tables proposées). Ces étapes sont donc le résultat de nos premières sélections via l'interface avant d'arriver dans l'éditeur Power Query. **Figure 5**

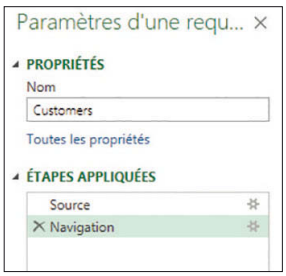


Figure 5 : *zone Paramètres d'une requête*

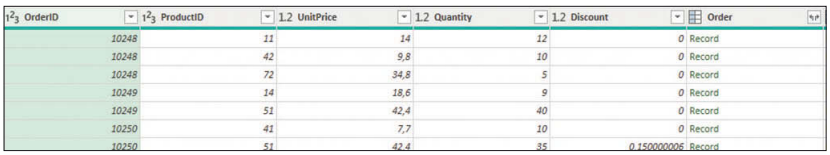
Autres transformations

À partir de ces étapes, je peux appliquer de nouvelles transformations, étape par étape. L'ensemble des transformations sont accessibles de deux façons :

- Via les menus rubans en haut de l'écran (menus **Accueil**, **Transformer** ou **Ajouter une colonne**)
- Via les menus contextuels disponibles en sélectionnant une ou plusieurs colonnes (la touche **CTRL** combinée à la sélection de l'en-tête de plusieurs colonnes permet de faire de la multi-sélection) ou via le menu global positionné en haut à gauche de la zone de prévisualisation des données.

L'adoption de l'un ou l'autre de ces modes d'accès aux transformations est un choix personnel. Comme souvent dans le monde Microsoft, de nombreux choix s'offrent à vous pour atteindre un même but. Parmi les transformations classiques, on trouve le changement de type d'une colonne, la sélection ou l'exclusion de valeurs données, la sélection de colonnes à conserver ou encore la fusion de plusieurs colonnes. Toutes ces transformations peuvent paraître simples, mais il vous est possible d'aller plus loin !

Dans notre exemple, les requêtes englobent les relations entre les tables. Des métadonnées sont donc présentes dans certaines requêtes, matérialisées par une coloration de la valeur de la colonne (en vert dans le monde Excel, en jaune dans le monde Power BI). La colonne **Order** de la table **Order_Details** affiche, par exemple, une valeur **Record** et le pictogramme de transformation, à droite du nom de la colonne propose de « déplier » les colonnes de la table liée. **Figure 6**



OrderID	ProductID	UnitPrice	Quantity	Discount	Order
10248	11	14	12	0	Record
10248	42	9,8	10	0	Record
10248	72	34,8	5	0	Record
10249	14	18,6	9	0	Record
10249	51	42,4	40	0	Record
10250	41	7,7	10	0	Record
10250	51	42,4	35	0,150000006	Record

Figure 6 : *métadonnée Record*

Vous pouvez ainsi joindre les tables **Order_Details** et **Orders** en une seule table à plat en cliquant sur le pictogramme de transformation. Vous êtes alors invité à sélectionner la ou les colonne(s) à inclure dans votre table résultat. Sélectionnez **OrderDate**, **ShipCity** et **ShipCountry**. Au passage, décochez la case **Utiliser le nom de la colonne d'origine comme préfixe**. Cette option préfixe systématiquement chaque nouvelle colonne par le nom de la table d'origine, ce qui au final rend le jeu de données moins lisible. **Figure 7**

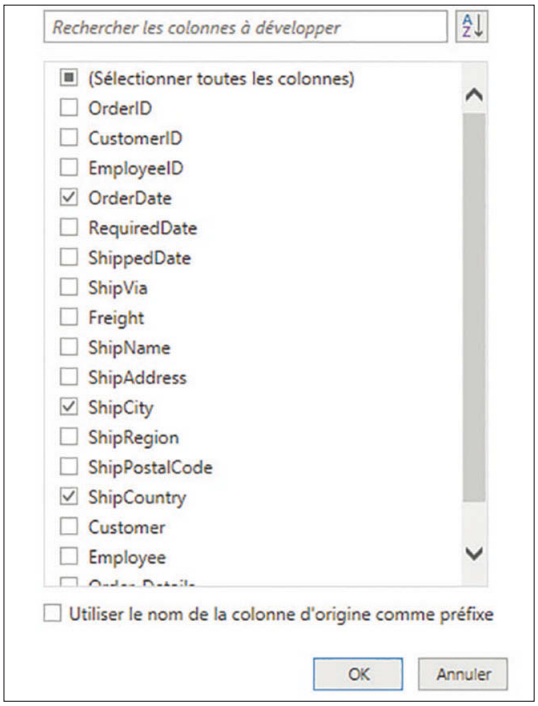


Figure 7 : *sélection des colonnes à étendre*

Colonne par l'exemple

Admettons maintenant que les transformations classiques disponibles via les menus vous semblent trop limitées. Vous souhaitez créer à partir de votre table **Customers** une colonne reprenant l'initiale du prénom du client suivie d'un point, d'un espace et du nom de ce client, suivi entre parenthèses du titre de celui-ci. Pour cela deux solutions : vous plongez dans la syntaxe du langage généré par Power Query, le langage M (démarche pas très no code, il est vrai) ; ou vous apprenez à Power Query les transformations à appliquer par l'exemple.

C'est cette dernière possibilité que nous allons explorer. Sélectionnez tout d'abord la requête **Customers** dans laquelle nous voulons créer notre nouvelle colonne. Sélectionnez également les colonnes servant de sources à l'exemple (ici les colonnes **ContactName** et **ContactTitle**).

Dans le menu ruban **Ajouter une colonne**, vous trouverez facilement le choix **Colonnes à partir d'exemples**. Cliquez sur cette option en prenant soin de sélectionner dans le menu déroulant le choix **à partir de la sélection**.

L'écran vous présente alors une nouvelle colonne à droite de la zone de prévisualisation. C'est dans cette colonne que vous allez proposer des exemples à Power Query. Par exemple, pour la première ligne (**ContactName** = Maria Anders), saisissez dans la cellule de la colonne 1 :

Figure 8 : *saisie d'un premier exemple*

M. Anders (Sales Representative) **Figure 8**

Vous pouvez constater que ce premier essai n'est que partiellement réussi. Power Query vous propose des valeurs pour les autres lignes. L'outil a bien compris que la valeur de la colonne **ContactTitle** doit se trouver entre parenthèses. Néanmoins, il reproduit le même nom dans chacune des lignes. Il va falloir lui proposer un second exemple. Sur la seconde ligne, saisissez le texte suivant :

Trujillo (Owner)

Cette fois-ci, Power Query réussit le challenge et propose les bonnes valeurs pour la nouvelle colonne. **Figure 9**

Repoussons les limites

Nous venons de voir qu'il est possible de laisser Power Query produire le code correspondant à la transformation que nous souhaitons appliquer. Il existe toutefois des transformations simples combinant les valeurs de deux colonnes ou plus et qui ne peuvent s'appliquer directement via l'interface de Power Query.

Pour ces transformations-là, il est possible de saisir une partie du code tout en conservant le bénéfice de l'interface de Power Query (aucune nécessité de réécrire l'intégralité du code déjà généré). Il s'agit alors de produire le code minimal utile pour réaliser notre transformation (souvent limité à l'utilisation d'opérateurs classiques).

Pour illustrer cela, imaginons que nous voulions ajouter une colonne **Total ligne** à notre table **Order_Details**. À cet effet, nous disposons, dans le menu ruban **Ajouter une colonne**, d'une commande **Colonne personnalisée**. **Figure 10**

Lorsque nous sélectionnons cette commande, la fenêtre suivante s'affiche. Une zone permet de saisir la formule de colonne personnalisée. À droite, nous disposons d'une liste des colonnes disponibles.

À titre d'exemple, si nous souhaitons obtenir le total de chaque ligne, il suffit de multiplier la quantité vendue par le prix unitaire de l'article. Un premier double-clic sur la colonne **Quantity** permet d'ajouter cette colonne (entre crochets) dans la formule. Nous pouvons ensuite saisir le symbole de la multiplication (*) puis double-cliquer sur la colonne **UnitPrice**. Nous obtenons alors la formule suivante. **Figure 11**

Langage M : la syntaxe minimale à connaître

Si nous souhaitons utiliser au maximum cette fonction de colonne personnalisée, il suffit de connaître la syntaxe de base des opérateurs Power Query, à savoir :

- Les opérateurs élémentaires (*, /, +, -)
- Les opérateurs logiques (or, and, not)
- L'opérateur de concaténation (&)
- Les opérateurs de comparaison (>, >=, <, <=, =, <>)
- Les éléments conditionnels (if, then, else)

Ce jeu restreint d'opérateurs constitue la trousse de survie du développeur Power Query. La possibilité de produire des transformations avec un code minimal facilite l'adoption de Power Query par un utilisateur non technique. Toutefois, si le démon du code vous démange et que vous souhaitez aller plus loin dans la maîtrise du langage M, il est possible de basculer à tout moment en mode « pro dev ».

Du low code au pro dev : l'éditeur avancé Affichage du code généré

Nous venons de voir que nous pouvons réaliser des transformations, des plus simples au plus complexes, sans produire le moindre code. Pourtant, nous savons que le moteur Power Query utilise le langage M pour identifier les transformations à réaliser. Je vous propose de jeter un œil sur le code que nous venons de produire ! Chaque transformation produit son propre code. Si on reprend la colonne par l'exemple crée précédemment sur notre table **Customers**, nous voyons que l'étape est appelée **Colonne personnalisée ajoutée**. Le code M associé peut être visualisé dans une fenêtre au-dessus de la zone de prévisualisation. Si cette fenêtre ne s'affiche pas, il vous suffit de vous rendre dans le menu ruban **Affichage** et de cocher la case **Barre de formule** pour la faire apparaître. **Figure 12**

Le code de notre colonne par l'exemple est le suivant :

```
= Table.AddColumn(Customers_table, "Colonne1", each let splitContactName = Splitter.SplitTextByDelimiter(" ", QuoteStyle.None)([ContactName]) in Text.Combine({Text.Start(splitContactName{0}?, 1), ". ", List.Last(splitContactName), " (", [ContactTitle], ")"}), type text)
```

Figure 13

Figure 9 : *saisie d'un second exemple*

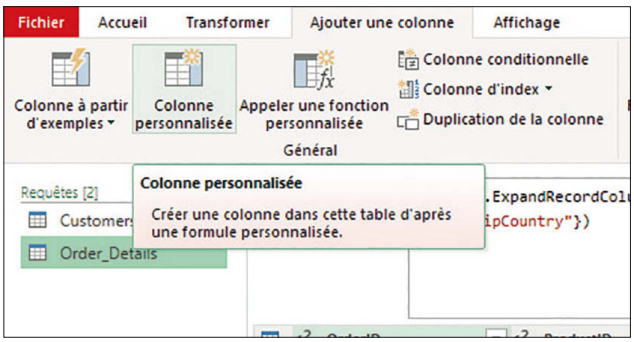


Figure 10 : *colonne personnalisée*

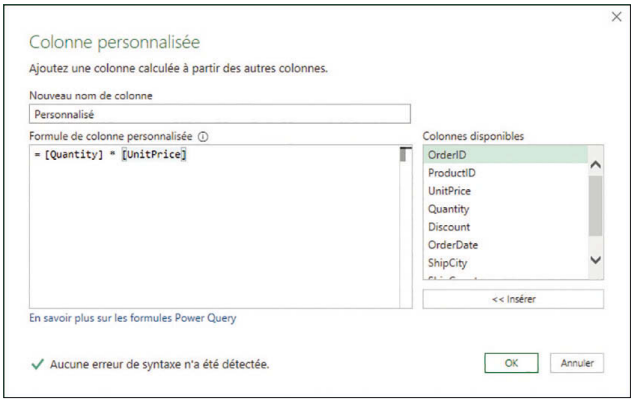


Figure 11 : *formule de colonne personnalisée*

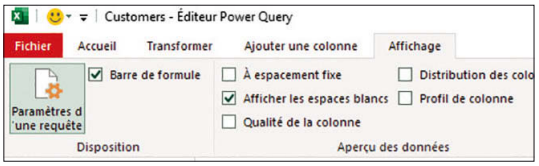


Figure 12 : *affichage de la barre de formule*

Sans aller dans le détail des fonctions utilisées ici, on peut identifier des éléments clefs du langage M :

- Chaque étape est appliquée sur un type d'élément (ici **Table**) représentant les éléments affichés à l'étape précédente. Dans Power Query, les éléments de type **Table** sont représentatifs d'une matrice de lignes et colonnes. Nous retrouverons fréquemment ce préfixe de fonction.
- Séparée du type d'élément par un point, on trouve ensuite la fonction utilisée. Elle décrit la transformation à appliquer (ici **AddColumn** pour ajouter une nouvelle colonne).
- Chaque fonction admet un certain nombre de paramètres entre parenthèses. Le plus souvent, le premier paramètre est le nom de l'étape précédente (les étapes « classiques » de type **Navigation** prennent parfois un autre nom technique, comme **Customers_Table** par exemple).

À titre d'exemple, nous pouvons analyser le code produit :

- Par la dernière étape de la requête **Customers** : pour **AddColumn**, nous aurons donc 3 paramètres : le nom de l'étape précédente, le nom de la nouvelle colonne, la transformation appliquée à chaque ligne de données (symbolisée par le mot clef **each**).
- Par la dernière étape de la requête **Order_Details** : pour **ExpandRecordColumn**, nous aurons 4 paramètres : le nom de l'étape précédente, le nom de la table développée, la liste entre accolades des colonnes à étendre, la liste entre accolades des nouveaux noms des colonnes étendues.

Enfin, il est possible d'avoir une vue de l'ensemble du code généré pour une requête. Il suffit pour cela de cliquer dans le menu ruban **Accueil** sur le choix **Editeur avancé**. L'ensemble des lignes générées est alors affiché à l'écran, ouvrant au développeur désireux d'aller plus loin la possibilité de saisir ou modifier directement son code dans l'éditeur.

Premiers pas dans le langage M

En analysant le code affiché via l'éditeur avancé, nous constatons que chaque requête commence par le mot clef **let** et se termine par le mot clef **in** suivi de l'étape finale de transformation.

Figure 13 : *formule en langage M*

Il en résulte deux constatations :

- Chaque étape / transformation, représentée par une ligne de code, n'est en fait qu'une variable initialisée à partir de la variable précédente. Ainsi, je peux déclarer n'importe quelle variable (en utilisant la syntaxe #''NomDeMaVariable'') pour la réutiliser plus tard dans mon code.
- Je peux remplacer ma dernière variable (celle qui suit le mot clef **in**) par n'importe quelle autre variable/étape utilisée dans mon code. Je peux, en particulier, débuguer mes éventuelles étapes en erreur, en affichant directement un résultat intermédiaire. Le code suivant, bien que fonctionnellement inutile, est syntaxiquement correct.

Figure 14

L'une des forces du langage M est qu'il s'agit d'un « lazy language » : Power Query exécute les étapes minimales permettant d'obtenir le résultat mentionné après le mot clef **in**.

Dans l'exemple précédent, la requête doit produire le résultat de la variable #''MaVariable1''. Power Query va chercher la définition de cette variable (première ligne). Il affecte la valeur 10 à cette variable et retourne le résultat sans réaliser le calcul intermédiaire (utilisation de #''MaVariable2'' pour obtenir la valeur de #''MaVariable3'').

Cette notion de « lazy code » ouvre la voie à des requêtes plus complexes, utilisant des paramètres et des expressions conditionnelles permettant d'optimiser son code en fonction des paramètres fournis en entrée. La souplesse de l'éditeur Power Query permet de générer 95% du code utilisé pour appliquer les transformations de données, tout en donnant la possibilité au développeur expert de coder intégralement les 5% restants.

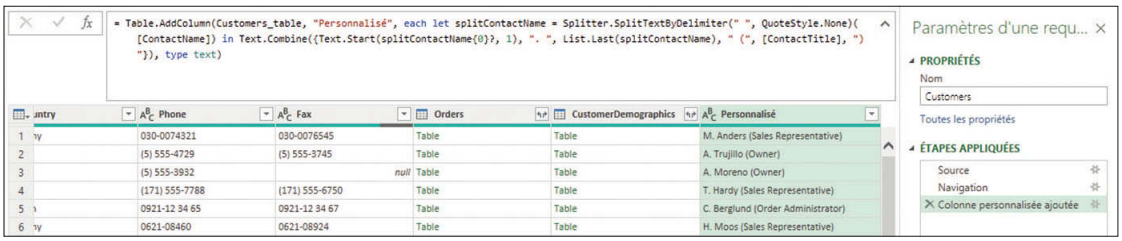
Un éditeur unique pour les gouverner tous

Dans cette brève introduction à Power Query, vous avez pu découvrir une interface simple d'emploi, commune à plusieurs outils Microsoft (Excel, mais aussi Power BI) et qui offre à un utilisateur non technique un ensemble de fonctionnalités permettant de réaliser des transformations avancées de préparation et nettoyage de données.

Bien que guidé, l'utilisateur peut à tout moment (pour une transformation précise ou pour l'ensemble de sa requête) basculer en édition avancée pour avoir la maîtrise de son code. L'éditeur Power Query permet de bénéficier à la fois d'une démarche essentiellement no code (pour la plupart des requêtes) et d'une ouverture vers un langage de script (pour les requêtes les plus complexes).

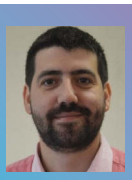
Son adoption au sein de services PaaS du cloud Azure de Microsoft et la constante évolution de l'interface de l'éditeur prouvent la volonté de la firme de Redmond d'investir sur la durée sur cet outil de self-service data prep à la longévité exemplaire.

Figure 14 : *exemple de formule « lazy code »*



Au cœur d'Azure avec les Fonctions Durables

Les Azure Functions font référence à la partie serverless des ressources mises à la disposition des architectes et des développeurs. Elles servent à écrire moins de code et à simplifier la maintenance comme la scalabilité de son infrastructure. Ainsi, elles offrent au développeur d'innombrables possibilités en termes de développement, mais également au niveau de la connectivité avec les autres briques de l'environnement Azure.



Clement Sannier
Leader Technique MS
SQLI DIGITAL EXPERIENCE

De l'utilisation d'un déclencheur basique en fonction d'un minuteur à une tâche planifiée (ou tâche Cron dans Linux), Azure Fonctions s'interconnecte avec du stockage blob ou de l'Azure Cosmos DB, en passant par une gestion événementielle comme Event Grid ou Iot Hub, voire du Kafka. Ce ne sont pas moins d'une vingtaine de connecteurs disponibles pour interagir avec des évènements, comme générer une API, traiter des flux de données métiers, ou encore déclencher des actions de modifications de base de données. Par exemple, vous souhaitez traiter des fichiers issus de flux métier en entrée, puis par la suite, envoyer un message d'information dans votre Event Grid afin de dispatcher l'évènement à différents services ? Vous pouvez utiliser un déclencheur sur votre blob storage, faire appel à votre ressource Event Grid et insérer votre code entre le deux : **figure 1**

Si vous souhaitez aller plus loin, vous pouvez retrouver l'ensemble des déclencheurs et des liaisons en suivant ce lien : https://docs.microsoft.com/fr-fr/azure/azure-functions/functions-triggers-bindings?tabs=csharp#supported-bindings.

Pour finir sur les fonctions Azure, sachez qu'à l'heure actuelle, les langages supportés sont : C#, JavaScript, F#, Java, Powershell, Python et TypeScript (par transpilation en JavaScript). Vous trouverez les renseignements sur les langages supportés en suivant ce lien : https://docs.microsoft.com/fr-fr/azure/azure-functions/supported-languages

Les fonctions durables dans l'univers des fonctions Azure

Les fonctions Azure sont de puissants outils pour se simplifier la tâche dans certains cas d'usage. Ils offrent également une simplification de la scalabilité de l'architecture, sans compter la facilité à déployer les fonctions à partir d'outils d'intégration continue comme GitHub ou Gitlab.

Toutefois, les fonctions Azure ont des limites dans leur exploitation. Par exemple, par défaut, une fonction Azure a un timeout de 30 minutes. Même si la durée du timeout est configurable jusqu'à l'infini, il réside tout de même une limitation « philosophique » sur la gestion du temps de traitement d'une fonction Azure. En effet, à l'origine, la partie serverless est optimisée pour des traitements courts se basant sur des évènements, afin de délivrer de manière optimale les informations, dont l'utilisateur à besoin.

C'est là qu'entre en jeu les « fonctions durables ». Elles sont tout simplement une extension des fonctions Azure offrant aux développeurs la possibilité de penser en mode workflow et non-utilisation unique. Microsoft gère en arrière-plan l'état des processus en cours et le redémarrage des tâches en cas de problème ainsi qu'un framework avec différents types de fonctions. Pour information, les deux principaux types de fonctions sont les fonctions d'orchestration et les fonctions d'activité.

La fonction d'orchestration

En s'appuyant sur des fonctions d'orchestration, il est ainsi possible de gérer un ordonnancement de tâche spécifique, de déléguer des traitements ou d'attendre une action utilisateur. Les fonctions d'orchestration sont les cerveaux de la gestion du traitement que le développeur veut mettre en place. En décrivant les actions et l'ordre dans lequel elles doivent être exécutées, il est envisageable d'ordonnancer des tâches de différentes manières afin de réaliser un workflow complet de validation de facture, par exemple, ou de traitement d'un incident depuis sa détection jusqu'à sa résolution. Par ailleurs, une fonction d'orchestration peut appeler une autre fonction d'orchestration.

Une fonction d'orchestration se présente comme ceci :

```
[FunctionName("ChainingDurableFunction")]
  public static async Task<List<string>> RunOrchestrator(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
  {
```

```
[FunctionName("BlobStorageEventGridFunction")]
0 references | 0 changes | 0 authors, 0 changes
public async Task Run([BlobTrigger(blobPath: "BlobEventGrid/{name}", Connection = "BlobEventGrid")]Stream myBlob, string name, ILogger log,
    [EventGrid(TopicEndpointUri = "EventGridEndpoint", TopicKeySetting = "EventGridKey")] IAsyncCollector<CloudEvent> eventCollector)
{
    log.LogInformation(message: $"C# Blob trigger function Processed blob\n Name:{name} \n Size: {myBlob.Length} Bytes");

    var data :string = ExtractData(myBlob);

    //Insérer votre code ici

    CloudEvent e = new CloudEvent(source: "IncomingRequest", type: "IncomingRequest", data);
    await eventCollector.AddAsync(e);
}
```

Figure 1 : *Implémentation d'une fonction azure*

```
    var commandNumber = context.GetInput<int>();

    var outputs = new List<string>();

    outputs.Add(await context.CallActivityAsync<string>("VerifyCommand", commandNumber));
    outputs.Add(await context.CallActivityAsync<string>("SendEmail", "La commande est acceptée"));
    outputs.Add(await context.CallActivityAsync<string>("PrepareCommand", commandNumber));

    return outputs;
  }
```

Au-delà du nom, elle prend en paramètre un « contexte d'orchestration » définissant les informations nécessaires au bon déroulement de la fonction en cours ainsi que les méthodes ou fonctionnalités à disposition pour orchestrer les différentes tâches. Dans l'exemple, nous pouvons voir que l'orchestrateur appelle grâce à la méthode « CallActivityAsync » une fonction de type activité.

La fonction d'activité

Si la fonction d'orchestration est le cerveau d'un traitement, la fonction d'activité est la base de tous les traitements de la fonction durable. Ce sont les tâches accomplies durant le workflow. Elles ne peuvent être appelées que par la fonction d'orchestration et le développeur a le choix de retourner un résultat afin d'affiner les différentes facettes de son traitement. Concrètement, cela donne :

```
[FunctionName("VerifyCommand")]
  public static bool VerifyCommand([ActivityTrigger] int commandNumber, ILogger log)
  {
    log.LogInformation($"Verification Command number : {commandNumber}.");

    //Verification de la commande

    return true;
  }
```

La notion d'orchestration

Il faut savoir que derrière les fonctions durables, nous retrouvons le framework open source DurableTask (https://github.com/Azure/durabletask) que vous pouvez utiliser dans vos applications afin de réaliser les mêmes workflows que sur Azure. Par conséquent, même si vous n'utilisez pas les fonctions Azure, vous pouvez tout de même utiliser le framework sur vos serveurs.
L'orchestration des fonctions durables repose sur une table d'historique. Au moment de démarrer une nouvelle instance d'un traitement du workflow, l'orchestrateur assigne un Id unique sous forme de Guid. Cet Id est autogénéré par le service, mais vous pouvez le générer vous-même au besoin, toutefois, cela n'est conseillé que si vous souhaitez réellement avoir un lien entre votre Id et une instance métier comme un bon de commande.
Afin de gérer au mieux les étapes du workflow, l'orchestrateur s'appuie sur la notion d'évènements, un peu comme nous le ferions en mode « event sourcing » où chaque évènement est consigné et pour connaître l'avancement d'un workflow, il suffit à l'orchestrateur de regarder les évènements survenus pour une instance donnée.
Par exemple, dans notre fonction d'orchestration, nous aurons pour chaque étape du workflow de commande, des entrées dans l'historique pour « VerifiyCommand », « SendEmail » et « PrepareCommand ». Pour information, chaque appel à une fonction d'activité a au moins deux évènements émis qui correspondent au démarrage et à la complétion de l'activité.
En cas de coupure, l'orchestrateur sait ainsi où il en est et quelle activité doit-il redémarrer pour achever le workflow.
Pour plus de détails, n'hésitez pas à regarder dans la documentation : https://docs.microsoft.com/fr-fr/azure/azure-functions/durable/durable-functions-orchestrations?tabs=csharp

Le déclenchement d'une fonction durable

Le déclenchement d'une fonction durable fait appel aux fonctions classiques avec la particularité d'avoir en paramètre l'interface cliente de l'orchestrateur « IDurableOrchestrationClient ». Vous pouvez ainsi lancer le traitement d'une fonction durable à l'aide d'un appel http ou encore à partir d'une heure spécifique :

```
[FunctionName("DurableTimerTrigger")]
    public async Task Run([TimerTrigger("0 */5 * * * *")]TimerInfo myTimer,
      [DurableClient] IDurableOrchestrationClient starter,
      ILogger log)
    {
     log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");

      string instanceId = await starter.StartNewAsync("DelegateFunction", null);

      log.LogInformation($"Started orchestration with ID = '{instanceId}'.");
    }
```

Les différents cas d'usage

Maintenant que vous y voyez plus clair sur le fonctionnement des fonctions durables, je vous propose de regarder d'un peu plus près quelques cas d'usage qui m'ont été utiles afin de bâtir différents processus de workflow. Vous verrez, ils s'avèrent complémentaires.

Cas simple, le chaînage

Le cas le plus simple est le chaînage des fonctions d'activité. Son modèle réside dans l'enchaînement des tâches les unes après les autres jusqu'à la fin du traitement. Par exemple, le processus d'une commande comme vue précédemment peut faire partie du chaînage des tâches :
Code complet sur programmez.com & github
Nous ne démarrons une tâche que si la précédente est terminée.

La délégation de traitement

Le deuxième cas d'usage concerne la délégation d'un traitement. Nous en avons besoin soit parce que le traitement est trop long soit parce que son résultat n'est pas immédiatement souhaité par l'utilisateur. Dans ce cas, un traitement asynchrone est de rigueur et les fonctions durables vous aide-

raient dans cette tâche. Imaginons que nous souhaitons effectuer un diagnostic précis de l'ensemble des sondes IoT d'une maison ou d'un immeuble. Le superviseur de cette tâche lance son diagnostic depuis sa console d'administration. Dans ce cas, l'objectif est de réaliser une tâche asynchrone et d'attendre que celle-ci soit terminée pour afficher le résultat à l'utilisateur.
Pour cela, il vous faut un déclencheur :

```
[FunctionName("DelegateFunction_HttpStart")]
    public static async Task<HttpResponseMessage> HttpStart(
      [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post")] HttpRequestMessage req,
      [DurableClient] IDurableOrchestrationClient starter,
      ILogger log)
    {
      // Function input comes from the request content.
      string instanceId = await starter.StartNewAsync("DelegateFunction", null);

      log.LogInformation($"Started orchestration with ID = '{instanceId}'.");

      return starter.CreateCheckStatusResponse(req, instanceId);
    }
```

Ici nous prenons un déclencheur de type http qui appellera l'orchestrateur pour réaliser les diagnostics :

```
[FunctionName("DelegateFunction")]
    public static async Task<List<string>> RunOrchestrator(
      [OrchestrationTrigger] IDurableOrchestrationContext context)
    {
      var outputs = new List<string>();

      outputs.Add(await context.CallActivityAsync<string>("Diagnostic", "Sensor1"));
      outputs.Add(await context.CallActivityAsync<string>("Diagnostic", "Sensor2"));
      outputs.Add(await context.CallActivityAsync<string>("Diagnostic", "Sensor3"));

      return outputs;
    }
```

Chaque diagnostic se fera par l'intermédiaire de la fonction d'activité « Diagnostic ».
Lorsque le déclencheur est appelé, le système se met automatiquement en marche et renvoie les éléments suivants dont le développeur peut profiter :

```
{
  "id": "bfd62d52ede6428bb5106548342e0df6",
  "statusQueryGetUri": "http://localhost:7071/runtri": "http://localhost:7071/runtime/webhooks/durabletask/instances/bfd62d52ede6428bb5106548342e0df6?taskHub=TestHubName&connection=Storage&code=HbWNFwiTivwtFIHntG043Drw34rmoy81G8hjInqosa21bPLR9ITtXA==",
  "sendEventPostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/bfd62d52ede6428bb5106548342e0df6/raiseEvent/{eventName}?taskHub=TestHubName&connection=Storage&code=HbWNFwiTivwtFIHntG043Drw34rmoy81G8hjInqosa21bPLR9ITtXA==",
  "terminatePostUri": ["http://localhost:7071/runtime/webhooks/durabletask/instances/bfd62d52ede6428bb5106548342e0df6/terminate?reason={text}&taskHub=TestHubName&connection=Storage&code=HbWNFwiTivwtFIHntG043Drw34rmoy81G8hjInqosa21bPLR9ITtXA==",
  "purgeHistoryDeleteUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/bfd62d52ede6428bb5106548342e0df6?taskHub=TestHubName&connection=Storage&code=HbWNFwiTivwtFIHntG043Drw34rmoy81G8hjInqosa21bPLR9ITtXA==",
  "restartPostri": "http://localhost:7071/runtime/webhooks/durabletask/instances/bfd62d52ede6428bb5106548342e0df6?taskHub=TestHubName&connection=Storage&code=HbWNFwiTivwtFIHntG043Drw34rmoy81G8hjInqosa21bPLR9ITtXA==",
  "restartPostUri": "http://localhost:7071/runtime/webhooks/durabletask/instances/bfd62d52ede6428bb5106548342e0df6/restart?taskHub=TestHubName&connection=Storage&code=HbWNFwiTivwtFIHntG043Drw34rmoy81G8hjInqosa21bPLR9ITtXA=="
}
```

Comme indiqué précédemment, un Id d'instance est transmis permettant de faire référence à l'instance du traitement créé. De plus, le framework est livré avec plusieurs Url permettant, entre autres, de connaître le statut, de renvoyer un event, d'annuler le job ou encore de purger le résultat.
Si nous nous arrêtons sur l'Url de la propriété « statusQueryGetUri » nous pouvons rafraîchir le statut du diagnostic en cours :

```
{
  "name": "DelegateFunction",
  "instanceId": "bfd62d52ede6428bb5106548342e0df6",
  "runtimeStatus": "Running",
  "input": null,
  "customStatus": null,
  "output": null,
  "createdTime": "2022-04-19T15:54:07Z",
  "lastUpdatedTime": "2022-04-19T16:00:52Z"
}
```

Puis en rafraîchissant quelque temps plus tard, le « runtimeStatus » indique que le traitement est « completed » :

```
{
  "name": "DelegateFunction",
  "instanceId": "416a5afd0ece49939420d2b1193d80d5",
  "runtimeStatus": "Completed",
  "input": null,
  "customStatus": null,
  "output": [
    "Sensor1 OK !",
    "Sensor2 OK !",
    "Sensor3 OK !"
  ],
  "createdTime": "2022-04-19T16:06:46Z",
  "lastUpdatedTime": "2022-04-19T16:07:17Z"
}
```

Vous remarquerez que les 3 valeurs dans la propriété « output » concernent les 3 retours de la fonction diagnostic. Nous ajoutons donc les éventuels dysfonctionnements du processus en retour de nos méthodes. Toutefois, retenez bien deux choses importantes : la propriété « output » n'est remplie qu'une fois le processus complet terminé, tandis que l'orchestrateur conserve l'historique tant que celui-ci n'est pas purgé par l'utilisateur.
Dans tous les cas, grâce aux fonctions durables, il est possible de déléguer à une plateforme des traitements plus ou moins longs et récupérer le résultat par la suite.

Parallélisme de traitement

Dans le cas d'usage précédent, la fonction est exécutée en mode chaînage comme le premier cas d'usage. Toutefois, quand nous y réfléchissons, paralléliser le traitement serait plus judicieux surtout s'il faut diagnostiquer de nombreuses sondes.

Dans ce cas, le traitement sur la fonction d'orchestration est légèrement différent, mais les développeurs ayant l'habitude de lancer plusieurs « Tasks » en parallèle ne devraient pas être perdus. Ainsi, il nous est nécessaire de changer la manière d'appeler les fonctions d'activité :

```
[FunctionName("ParallelDurableFunction")]
    public static async Task<List<string>> RunOrchestrator(
        [OrchestrationTrigger] IDurableOrchestrationContext context)
    {

        var parallelTasks = new List<Task<string>>();

        for (int i = 1; i <= 3; i++)
        {
         Task<string> task = context.CallActivityAsync<string>("ParallelDiagnostic",
$"Sensor{i}");
            parallelTasks.Add(task);
        }

        await Task.WhenAll(parallelTasks);

        return parallelTasks.Select(parallelTask => parallelTask.Result).ToList();
    }
```

Le résultat étant le même et obtenu plus rapidement :

```
{
  "name": "ParallelDurableFunction",
  "instanceId": "20d542983ba84d7eb2392b0464c630c2",
  "runtimeStatus": "Completed",
  "input": null,
  "customStatus": null,
  "output": [
    "Sensor1 OK !",
    "Sensor2 OK !",
    "Sensor3 OK !"
  ],
  "createdTime": "2022-04-19T16:57:34Z",
  "lastUpdatedTime": "2022-04-19T16:57:45Z"
}
```

L'interaction humaine

Enfin le dernier cas d'usage que je souhaitais partager avec vous est sûrement l'un des plus importants car que serait un workflow sans la validation d'un acteur pendant le processus. Les fonctions durables offrent la possibilité de gérer ce type d'évènement externe grâce à la méthode « WaitForExternalEvent ».

En reprenant notre exemple d'une commande, nous pouvons aisément ajouter une validation dans le workflow, par exemple, au moment de la validation de la commande ou encore au moment de la préparation. Pour ce faire, il nous faut modifier notre workflow et assigner un nom faisant référence à l'évènement comme « ApprovalCommand » :

Code complet sur programmez.com & github

Avec Postman, vous pouvez appeler la fonction durable, grâce à son nom « ApprovalCommand » pour approuver l'évènement :

```
curl —location —request POST 'http://localhost:7071/runtime/webhooks/durabletask/
instances/8e7960063324410bbb0855de3d82ab65/raiseEvent/
ApprovalCommand?taskHub=TestHubName&connection=Storage&code=HbWNFwi
TivwtFIHntG043Drw34rmoy81G8hjlnqosa21bPLR9ITtXA==' \
—header 'Content-Type: application/json' \
—data-raw '"true"'
```

Ainsi, le traitement continue et la commande peut être préparée. Toutefois, dans le cas contraire si dans les 24 heures, comme indiqué dans la variable « limitDateTime », la commande n'est pas approuvée, par conséquent à la fin du « timer » le traitement annulera la commande.

Conclusion et REX

Adepte des fonctions Azure, la découverte et la mise en pratique des fonctions durables ont eu pour moi un regain d'intérêt. Cela m'a conforté dans l'idée que dans certaines situations, notamment la mise en place de workflow ou encore les traitements longs, les actions peuvent être déléguées à une instance en « background » et ainsi offrir une nouvelle expérience aux utilisateurs sans devoir mettre en place toute une usine logicielle pour autant. Intégrées totalement aux Fonctions Azures, les fonctions Durables n'en sont que plus facilement déployables au travers d'une chaîne d'intégration continue.

En un minimum de code, nous avons mis en place des traitements chaînés et grâce aux fonctions durables et à la parallélisation, nous avons réduit le temps de traitement de certaines tâches. Enfin, la délégation de traitement offre à nos clients la possibilité de demander un export de certaines données et être prévenu par mail lorsque le traitement est terminé. Un atout non négligeable !

Un bref aperçu de Unity Bolt

Unity, moteur de création de jeux bien connu, n'échappe pas à la mode du « no code » en proposant gratuitement Bolt un module dédié au Visual Scripting (VS). La promesse étant de vous affranchir du code pour mieux vous concentrer sur le gameplay.

Franck Dubois
Video Game Codeur
Développeur agile
Formateur JavaScript / Unity / GDevelop

Ici, le **VS** n'a pas d'autre objet que de représenter du code sous forme graphique, en l'occurrence ici des nœuds (appelés aussi unit) mis en musique par des liens, un nœud étant un événement ou une instruction. Donc pas de syntaxe complexe à connaître, mais cela n'enlève en rien la nécessité d'être dans une logique de codage. L'autre élément appréciable de **Bolt** est la possibilité de modifier à la volée un graphe en cours d'exécution. Pour le reste pas de changement, les composants de **Unity** restant identiques, l'usage du **VS** nécessite d'être familiarisé avec l'outil et ses concepts : variables, types, rigidbody, animator, et d'autres restent de mise. Concepts réutilisés ici sur lesquels je ne reviendrai pas en détail.

Quelques éléments pour débuter

Une fois le projet créé, pour utiliser Bolt, son installation avec ses dépendances passe par l'asset store et le package manager. Au 1er démarrage, l'assistant de configuration est lancé que vous pouvez aussi relancer depuis le menu **Tools/Bolt/Setup Wizard**. La 1ère question étant relative à l'affichage des nœuds orienté « user friendly » à la syntaxe plus compréhensible pour le débutant (**Human Naming**) ou « code » à la syntaxe plus orientée code (**Programmer Naming**). Pour les phases suivantes de configuration, laissez les options proposées par défaut. **Figure 1**

Le **flow graph** est le pendant de l'éditeur de code : au lieu d'écrire du code en **C#**, on utilise un graphe dans lequel figureront des actions et des événements. On utilise aussi la terminologie de macro pour désigner un graphe.
Pour rattacher un graphe à un objet, il faut au préalable lui rattacher un composant **Flow Machine**. C'est à partir de ce composant que vous pouvez créer un graphe/macro en cliquant sur le bouton **New**. **Figure 2**

Unity vous demandera de le nommer pour le sauvegarder. Un simple clic sur le bouton **Edit Graph** suffit pour l'afficher. Par défaut les événements **Start** et **Update** sont créés, équivalents aux méthodes **Start** et **Update** créées par défaut lors de la création d'un script C#. Retenez que les événements sont semblables aux méthodes proposées par les différents composants **Unity**. **Figure 3**

Il y a les événements et il y a les actions qui peuvent y être rattachées. Elles prennent la forme de nœuds et sont semblables à des instructions comme en C#. Chacun peut avoir un ou plusieurs paramètres en entrée (IN) et une « valeur » de sortie (OUT). L'exemple ci-dessous d'un nœud d'une fonction **Multiply** prend 2 valeurs en paramètres et en renvoie le produit. **Figures 4 et 5**

Pour ajouter un événement ou un nœud (Unit), tout se fait par navigation dans le menu contextuel du graphe ou par saisie de mots clés en ayant au préalable sélectionner l'objet.
Pour relier un nœud à l'autre, il suffit de cliquer une 1ère fois sur une entrée ou sortie et une 2de fois respectivement sur une sortie ou entrée.

Un petit exemple

L'exercice va consister à déplacement un shooter à l'aide des touches **z, q, s et d** du clavier. **Unity** étant par défaut configuré pour les claviers QWERTY, pensez à adapter la configuration au votre depuis les paramètres du projet accessibles depuis le menu **Edit/Project Settings/Input Manager/Axes/Horizontal|Vertical/Alt Negative|Positive Button**.
Au préalable, un simple sprite **shooter** a été intégré au projet et à la scène auquel a été associé un composant **RigidBody 2D** dont la gravité a été fixée à 0, sans quoi c'est la chute assurée. Pour gérer les interactions au clavier, la fonction **Input Get Axis** est appelée une fois pour le déplacement hori-

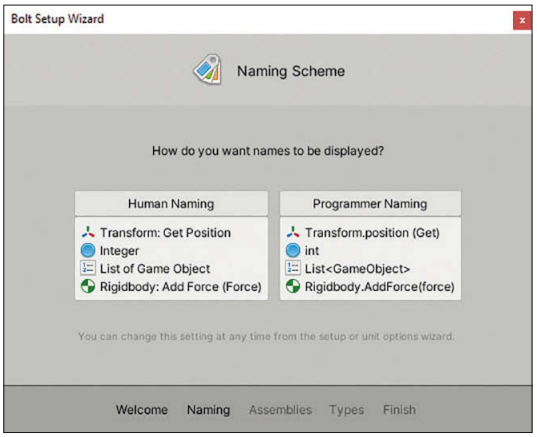


Figure 1

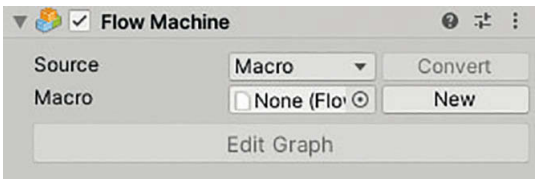
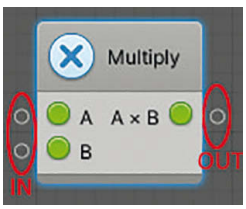


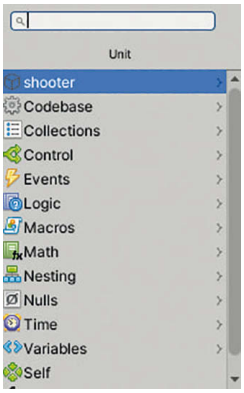
Figure 2

Figure 3

Figures 4 et 5



Entrées IN/Sortie OUT



Menu contextuel Unit

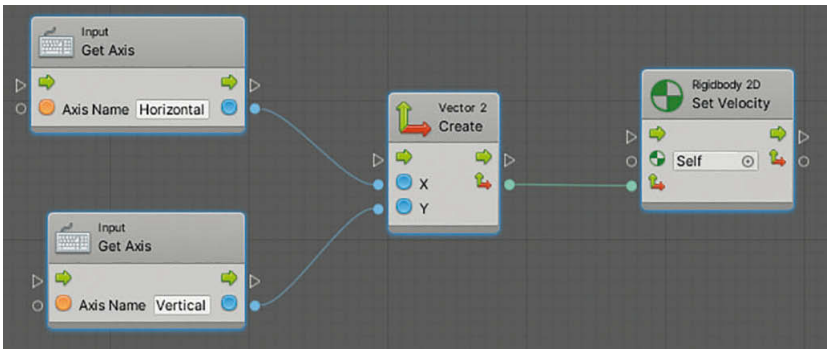
Figure 6

Figure 7

zontal avec comme paramètre **Horizontal** et une 2e fois pour le déplacement avec comme paramètre **Vertical**. Sur chacun des axes, cette fonction renvoie une valeur entre -1 et 1 en fonction de la touche appuyée : vers la gauche ou le bas, une valeur négative et vers la droite ou le haut, une valeur positive. **Figure 6**

Pour déplacer l'objet **shooter**, on modifie sa vélocité (**velocity**) du **Rigidbody 2D** en utilisant un **Unit Rigidbody 2D Set Velocity** qui prend en paramètre un **Vector 2D**. **Figure 7**

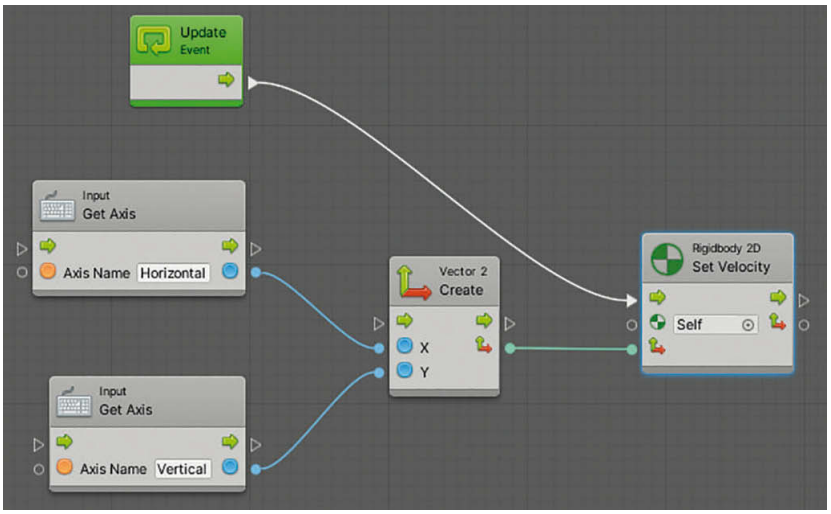
Figure 8



Il ne reste plus qu'à alimenter la vélocité par le biais d'un Vector 2D créé à partir des valeurs renvoyées par la fonction **Input Get Axis**. **Figure 8**

Pour être pleinement opérationnel, ce flux nécessite d'être appelé par l'événement **Update**. La modification de la vélocité étant le chaînon conduisant au déplacement contrôlé par le joueur, l'événement **Update** est rattaché à ce nœud. **Figure 9**

Figure 9

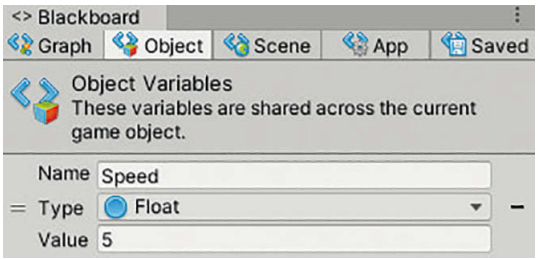


En l'état, le shooter se déplace à une vitesse relativement modeste, mais modulable en multipliant les valeurs renvoyées par les fonctions **Input Get Axis** par un coefficient vitesse. Cette dernière prend la forme d'une variable définie depuis le composant **Blackboard** ajouté avec tout composant **Flow Machine**. Si nécessaire, on y accède par le menu **Window/Variables**. **Blackboard** propose 5 onglets spécifiant le périmètre d'accessibilité des variables :

- Graph : variable locale au graph et uniquement accessible depuis le graph ;
- Object : variable rattachée à l'objet tout comme une propriété ;
- Scene : variable accessible depuis la scène ;
- App : variable accessible depuis toutes les scènes du jeu et initialisée à son lancement ;
- Saved : variable persistante (sauvegardée) même après avoir stopper le jeu.

La vitesse étant celle du shooter, il semble naturel d'utiliser une variable d'objet qu'on appelle **Speed** typée **float** et valorisée arbitrairement à 5. **Figure 10**

Figure 10

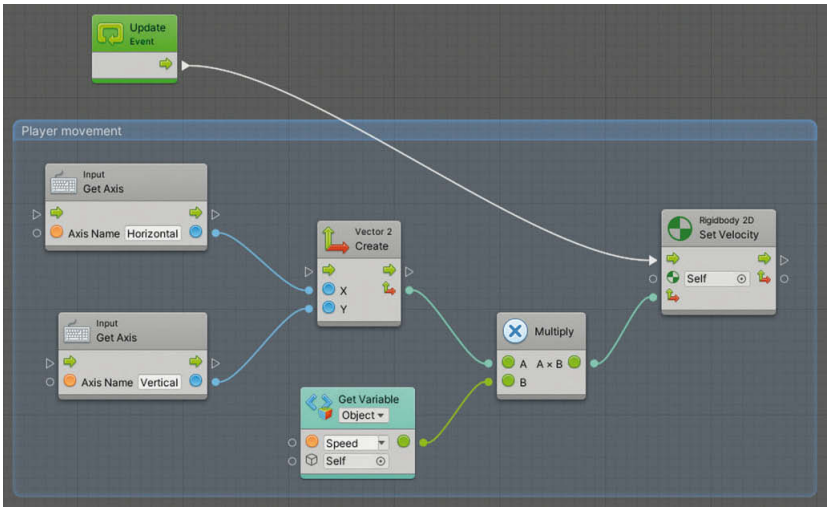


Pour intégrer ce coefficient **Speed** au graphe, 2 **unit** sont créés :

- un nœud **Get Variable Object** qui récupère la valeur de la variable **Speed** ;
- un nœud **Multiply** chargé de multiplier les coordonnées du **Vector 2D** par la variable **Speed**.

La gestion du déplacement terminée et pour une meilleure lisibilité, il est pertinent de placer tout ce petit monde dans un groupe en maintenant la touche Ctrl et en sélectionnant les **unit** concernés. **Figure 11**

Figure 11



Un autre groupe à créer dans la suite de l'exercice qui consiste à lancer un missile depuis le shooter par le biais de la touche **Espace** du clavier à l'aide de la fonction **Input Get Key Down** en prenant en paramètre la touche **Space**. **Figure 12**

Au préalable, on crée un **prefab missile** comportant aussi un composant **RigidBody 2D** avec une gravité a été fixée à 0 permettant de lui donner son mouvement. Ce mouvement est obtenu en appliquant une force sur le **RigidBody 2D**, déterminée par un **Vector 2D**. Le missile se déplaçant sur la verticale vers le haut, le vecteur prend comme coordonnées (0,10). Le paramètre **Mode** permet simplement de donner plus d'intensité à la force. **Figure 13.** Il ne reste plus qu'à créer les **unit** intermédiaires.

Figure 12

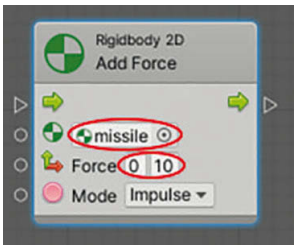


Figure 13

La fonction **Input Get Down** renvoie une valeur booléenne : **true** lors de l'appui sur la touche ou **false** le reste du temps. Cette valeur doit être traitée avec un unit **Branch**, qui, à partir d'une valeur booléenne donnée en entrée, permet de rediriger le flux d'exécution vers d'autres nœuds.
Dans le cas présent, l'appui sur la touche (**true**) déclenche la création du **missile** : on ajoute un **unit** qui instancie un **prefab missile** sans se soucier de la position et de la rotation. **Figure 14**

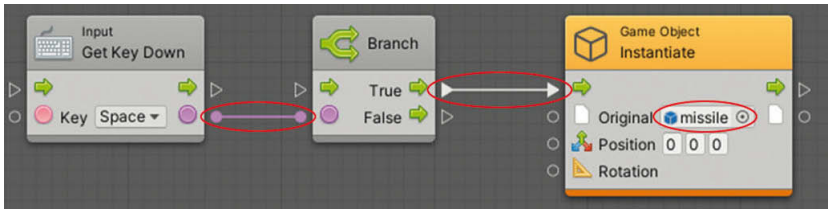


Figure 14

En l'état, le **missile** est créé au milieu de la scène (position 0,0,0). Or on souhaite le créer là où se trouve le shooter. Du coup, on ajoute au graphe un unit **Transform Get Position Self** qui renvoie la position de l'objet auquel est rattaché le script, ici le **shooter** et qui alimente le nœud **Instantiate**. Comme aucune rotation n'est appliquée à l'objet, il est alimenté par un autre unit **Quaternion** nul.
On termine en reliant le nœud **Instantiate** au nœud **Rigibody 2D Add Force** avec 2 liens : le 1er lié au flux et le 2d lié à l'objet instancié sur lequel s'applique la force, en l'occurrence le **missile**.
Il n'y a plus qu'à intégrer ce groupe au flux d'exécution de l'événement **Update** en faisant pointer le nœud **Rigibody 2D Set Velocity** vers le nœud **Branch**.
Alors pourquoi le nœud **Branch** et pas le nœud **Rigibody 2D Set Add Force** ? Ceci tient au mode opératoire de **Bolt** : pour exécuter une action, Bolt remonte le flux des actions liées aux paramètres spécifiés en entrée du nœud.
En faisant pointer le nœud **Rigibody 2D Set Velocity** vers le nœud **Rigibody 2D Set Add Force**. Bolt remonte sur le nœud précédent qui instancie l'objet **missile**. De ce nœud, Bolt remonte dans les nœuds paramètres précédents qui donnent des valeurs valides pour instancier l'objet. On a alors comme résultat, des instanciations de **missiles** en continu et sans action de la part du joueur.
Même si regrouper des nœuds connexes rend un graphe plus lisible, on voit poindre un début de commencement d'un graphe spaghetti, il va falloir donc être très vigilant.
Pour alléger notre graphe et faire disparaître ce lien entre les 2 groupes, il est possible de déporter le lancement de **missiles** à un événement propre à la gestion du clavier et déclenché chaque fois que l'utilisateur appuie sur une touche : **On Keyboard Input** et qui prend 2 paramètres. Le 1er étant la touche pressée et le 2e l'état de la touche : au moment de l'appui (**Down**), au moment de la relâche (**Up**) et en continu (**Hold**)). On crée le nœud événement correspondant à la touche **Espace** sur appui ou relâche. La sortie de cet événement est empruntée uniquement lorsque le joueur appuie sur

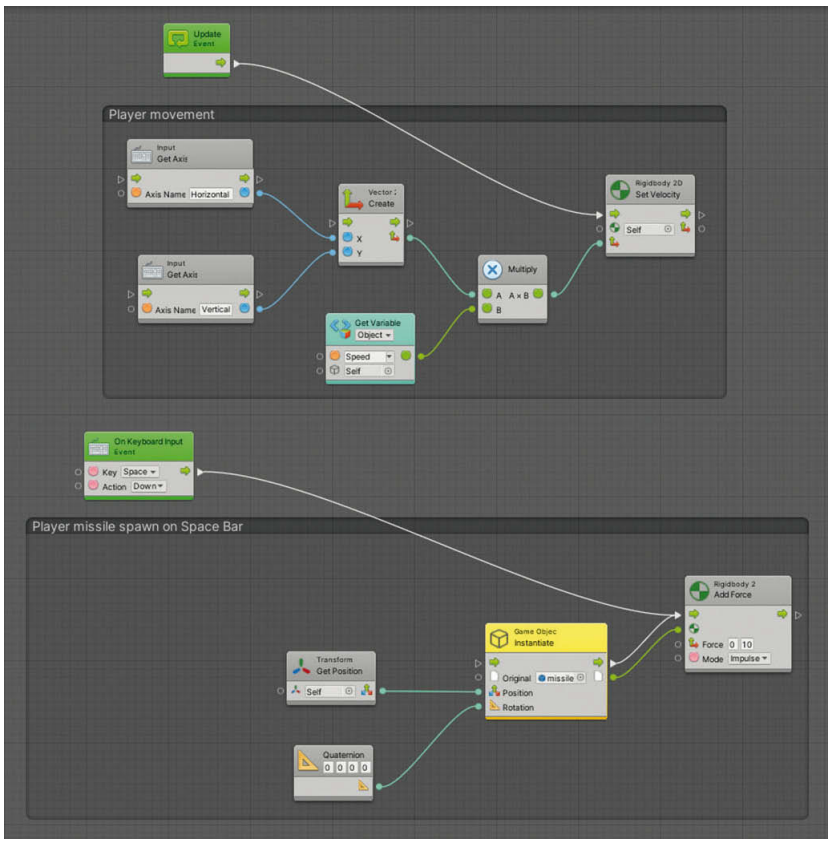


Figure 15

la touche **Espace**. Les nœuds **Input Get Key Down** et **Branch** peuvent donc être supprimés. Il ne reste plus qu'à relier l'événement **On Keyboard Input** au nœud **Rigibody 2D Set Add Force**. **Figure 15**

À première vue, **Bolt** est un outil qui peut séduire le débutant. Mais il n'enlève pas la manière de penser le projet. Et que ce soit avec le **VS** ou en **C#**, le temps d'apprentissage reste le même. Le **C#** a l'avantage d'être utilisé à d'autres fins que **Unity**. Et **Bolt** a le désavantage d'être **Unity** dépendant.

Philippe Charrière
Senior Technical Account Manager Chez GitLab et Explorateur Wasm

Combattre la complexité de WebAssembly avec Reactr

WASI (WebAssembly System Interface) est une spécification qui pose les fondations pour "libérer" Wasm du navigateur (ou plus précisément de la VM JavaScript, on parlera aussi d'hôte JavaScript pour les modules Wasm).

Aujourd'hui, il existe plusieurs "runtime WASI" permettant d'exécuter du code wasm à partir d'autres langages (Rust, Go, C/C++,...), autrement dit : écrire nos propres hôtes wasm. Les plus connus sont actuellement :

- Wasmer: https://wasmer.io/
- Wasmtime: https://wasmtime.dev/
- WasmEdge: https://wasmedge.org/

Selon moi, lorsque vous vous lancez dans un projet où vous allez exécuter du Wasm en dehors du navigateur (ou en dehors de la VM JavaScript de Node.js), vous allez être confrontés à deux malédictions majeures de WebAssembly.

La première malédiction est le système de type.

WebAssembly possède un système de type très (trop) simple : seulement 4 types numériques : integers (32 and 64 bit) et floats (32 and 64 bit). Cela signifie que par exemple, vous ne pouvez pas passer simplement une chaîne de caractères comme paramètre à une fonction **"importée"** d'un module Wasm.

Vous pouvez utiliser les pointeurs pour cela, et je vous conseille la lecture de l'excellente documentation du projet **WasmEdge** qui y consacre une explication ainsi que des exemples à mettre en œuvre : **"Pass complex parameters to Wasm functions"** :

https://wasmedge.org/book/en/embed/go/memory.html.

Dans un future proche (enfin je l'espère), Wasm va bénéficier de nouveaux types : les "interface type", un ensemble de types pour décrire des valeurs comme les strings, records, … Mais ce n'est pas encore pour tout de suite.

La seconde malédiction vient de la sécurité.

Les modules Wasm sont exécutés dans des environnements "sandboxed", donc séparés de leur hôte : pas d'accès au système d'exploitation dans lequel l'hôte s'exécute, ni accès aux autres modules Wasm. Ce qui signifie que par exemple, vous ne pouvez même pas faire une requête HTTP à partir de votre module Wasm. Mais l'avantage, c'est que vous pouvez garantir que le code qui s'exécute ne provoquera pas de vulnérabilité ou de faille de sécurité.

Mais grâce à la spécification **WASI** et les projets de **runtimes WASI*, nous pouvons résoudre ceci avec les **host function**; ce sont des fonctions **exportées** de l'hôte vers le module Wasm (qui lui les "verra" comme des fonctions importées).

Une fois encore, le projet **WasmEdge** propose une explication claire ainsi que des codes d'exemples : **Host Functions** : https://wasmedge.org/book/en/embed/go/ref.html#host-functions.

Mais, je ne vais pas vous le cacher, au départ, je suis plutôt un développeur JavaScript (et Java), et même si je commence à "tomber amoureux" de GoLang et RustLang, je n'aime pas du tout faire des acrobaties compliquées avec les pointeurs et la gestion de mémoire. Heureusement, j'ai trouvé un framework qui permet de grandement améliorer **"l'expérience développeur"** dans le développement de modules Wasm et d'hôtes pour ces modules (un hôte, cela peut être un petit serveur HTTP en Go qui fait des appels à une fonction Rust dans un module Wasm). Et ce framework, c'est **Reactr** de chez **Suborbital** (https://suborbital.dev/).

Donc, Reactr…

Lorsque j'ai commencé mes expérimentations avec Wasm et GoLang, j'ai eu un rapide échange sur Twitter avec **Connor Hicks** (@cohix), le fondateur de Suborbital, et ainsi j'eu le loisir de découvrir deux de ses projets:

- **Atmo** https://github.com/suborbital/atmo
- **Sat** https://github.com/suborbital/sat

Si je devais expliquer simplement et rapidement ce que sont **Atmo** et **Sat**, je dirais que le premier vous aide à créer une application web à base de services Wasm et que le second vous permet de générer des microservices wasm simplement et efficacement.

Atmo et **Sat** utilisent tous les deux le projet **Reactr** (https://github.com/suborbital/reactr). **Reactr** est une librairie pour faciliter le développement d'hôtes Wasm en GoLang (et il peut utiliser aussi bien WasmEdge, Wasmer and Wastime), mais aussi le développement des modules Wasm qui seront utilisés par votre hôte. Si l'hôte est développé en GoLang, néanmoins, l'API **Reactr** utilisée par les modules Wasm est disponible pour TinyGo, Rust, AssemblyScript, Swift.

Dans un projet **Reactr**, les modules Wasm sont nommés des **"Runnables"** (https://docs.suborbital.dev/atmo/runnable-api/introduction/). **Atmo** et **Sat** permettent de servir des **"Runnables"** comme des services HTTP.

Pour résumer, **Reactr** va vous aider à exposer des "host functions" de l'hôte au module Wasm avec des "helpers" déjà existants pour fonctionner avec Redis, PostgreSQL, HTTP requests, etc. Mais aussi **Reactr** simplifiera le passage de paramètre à une fonction d'un module Wasm. J'aime bien dire que **Reactr** donne des "super pouvoirs" à l'hôte WASI/

Remarque : Reactr n'est pas seulement un SDK pour construire des Runnables. Je vous invite à lire la documentation pour en découvrir les possibilités.

Mais il est maintenant temps de passer à la pratique et d'écrire notre premier "Runnable".

Mon premier "Runnable": je veux "passer une String" à une fonction Wasm

Un **Runnable** est une sorte de fonction écrite en Go (avec le compilateur TinyGo, car il suit la norme WASI, ce qui n'est pas le cas du compilateur officiel), Rust, Swift, AssemblyScript et Grain (un langage fonctionnel qui cible essentiellement Wasm). Il est même possible d'écrire ces fonctions JavaScript ou en TypeScript, mais dans ce cas ce sera de l'interpréter au sein du module Wasm.
Le moyen le plus simple de créer un **Runnable** est d'utiliser la CLI de Suborbital (https://github.com/suborbital/subo), de son "petit nom" subo. Si vous êtes sous Mac ou Linux, l'installation est extrêmement simple : (il n'existe pas encore de version pour Windows)

```
brew tap suborbital/subo
brew install subo
```

Ensuite, pour créer notre premier Runnable **en Go**, lancez la commande suivante :

```
subo create runnable hello-world —lang tinygo
```

La **subo** CLI va générer une structure de projet avec tout ce qu'il faut pour bien démarrer : **Figure 1**
Ensuite pour compiler le Runnable, utiliser la commande ci-dessous :

```
subo build hello-world/
```

Cette commande va charger la toolchain de Suborbital (vous aurez besoin de Docker) et ensuite compiler le module Wasm hello-world, et donc vous obtiendrez un nouveau fichier : hello-world.wasm. A partir de maintenant, nous avons besoin d'un autre programme pour tester (exécuter) le module Wasm. Nous allons donc développer un "lanceur de Runnable" avec GoLang et Reactr.

Le lanceur de Runnable

Le code source du "lanceur" est plutôt simple avec 4 étapes principales:

- Créer une instance de runtime Reactr
- Charger le module wasm et enregistrer la fonction
- Appeler la fonction avec ce paramètre : []byte("Bob Morane!") (en fait vous pouvez passer directement une String)
- Récupérer le résultat: string(result.([]byte))

```
package main

import (
  "fmt"
  "github.com/suborbital/reactr/rt"
  "github.com/suborbital/reactr/rwasm"
)

func main() {
  r := rt.New()
  doHelloWasm := r.Register("hello-world", rwasm.NewRunner("./hello-world/hello
-world.wasm"))

  result, err := doHelloWasm("Bob Morane!").Then()
  if err != nil {
    fmt.Println(err)
    return
  }

  fmt.Println(string(result.([]byte)))

}
```

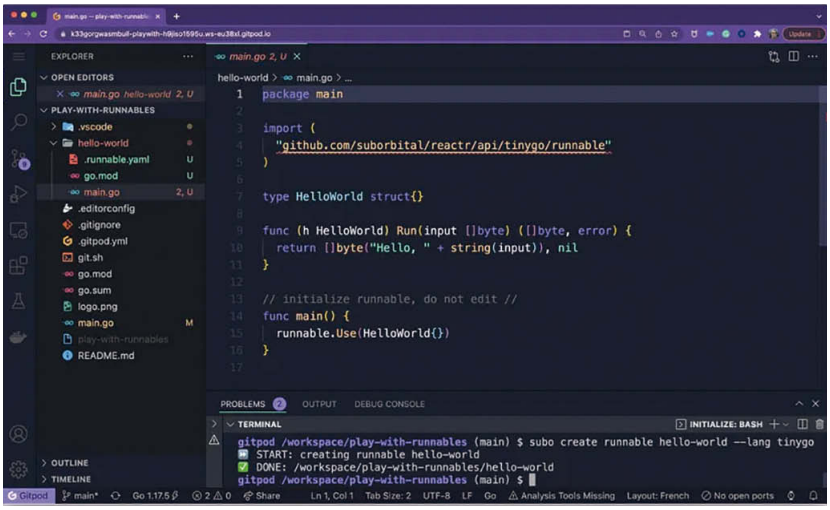


Figure 1
Image Runnable in GitPod

Démarrez le "lanceur" avec la commande suivante : go run main (ou go build, pour compiler et ensuite lancer l'exécutable). Vous allez obtenir un splendide Hello Bob Morane!.
Avec une perspective de développeur JavaScript, finalement ce n'est pas si difficile de passer une chaîne de caractère à un module Wasm. Mais maintenant, il est temps de pousser un peu plus loin : je voudrais appeler un service HTTP externe à partir de mon module Wasm.

Mon premier "Runnable": je veux appeler un service HTTP à partir d'une fonction Wasm

Nous avons de la chance, **Reactr** expose déjà des **host functions** pour les Runnables, comme la possibilité de faire des requêtes HTTP, utiliser du cache (avec Redis) et d'autres possibilités dont vous trouverez la liste ici (pour TinyGo) : https://github.com/suborbital/reactr/tree/main/api/tinygo/runnable.
Nous allons donc utiliser l'API HTTP et mettre à jour le code de notre Runnable "Hello World" de la manière suivante :

```
package main

import (
  "log"

  "github.com/suborbital/reactr/api/tinygo/runnable"
  reactrHttp "github.com/suborbital/reactr/api/tinygo/runnable/http"
)

type HelloWorld struct{}
```

```
func (h HelloWorld) Run(input []byte) ([]byte, error) {
  // call an external web service
  payload, _ := reactrHttp.GET("https://jsonplaceholder.typicode.com/todos/1", nil)

  // display the service response
  log.Println(string(payload))

  return []byte("Hello, " + string(input)), nil
}

// initialize runnable, do not edit //
func main() {
  runnable.Use(HelloWorld{})
}
```

Remarque : **https://jsonplaceholder.typicode.com/** est un service très pratique pour simuler une API Json.
Ensuite, compilez le Runnable : `subo build hello-world`, et relancez à nouveau le "lanceur", vous devriez obtenir une sortie comme celle-ci :

```
2022/03/26 07:02:34 {
 "userId": 1,
 "id": 1,
 "title": "delectus aut autem",
 "completed": false
}
Hello, Bob Morane!
```

Vous pouvez voir par vous-même que c'est d'une simplicité déroutante. **Reactr** est un incroyable SDK qui facilite la vie des développeurs lorsqu'ils souhaitent développer des applications reposant sur WebAssembly. Et Wasm est polyglotte et les Runnables aussi. Vous pourriez imaginer créer des CLI en Go avec des modules Wasm écrit en différents langages.

Les Runnables sont polyglottes

La CLI `subo` permet de générer des squelettes de Runnables pour différentes cibles:

- Rust (par défaut)
- Go (c'est en fait du TinyGo)
- Swift
- AssemblyScript (un sous-ensemble de TypeScript dédié à Wasm) **https://www.assemblyscript.org/**
- Grain (un tout nouveau langage fonctionnel dédié à Wasm) **https://grain-lang.org/**

C'est aussi une opportunité pour apprendre de nouveaux langages.
Il est même possible de coder des Runnables en JavaScript ou en TypeScript, mais dans ce cas-là c'est un interpréteur qui est exécuté par le module Wasm àl'aide du projet **Javy** (**https://github.com/Shopify/javy**).

Modifions d'abord le code du lanceur de Runnable

Nous voulons pouvoir appeler le lanceur de cette façon :

```
./play hello "Bob Morane"
```

Où `play` est le nom de l'exécutable, `hello` est le nom du module wasm et "Bob Morane" est le paramètre que nous voulons passer à la fonction du module.
En Go, pour lire le 1er argument passé à l'exécutable (donc `hello`), nous utiliserons :

```
calledFunction := os.Args[1] pour le nom de la fonction
```

Puis :

```
wasmModule := os.Args[1] + "/" + os.Args[1] + ".wasm" pour construire
 le chemin vers le module Wasm (donc hello/hello.wasm)
parameters := os.Args[2:][0] pour obtenir le 2e paramètre (donc "Bob
Morane")
```

Maintenant, modifions donc le code du lanceur de la façon suivante :

```
package main

import (
  "fmt"
  "os"

  "github.com/suborbital/reactr/rt"
  "github.com/suborbital/reactr/rwasm"
)

func main() {
  calledFunction := os.Args[1]
  wasmModule := os.Args[1] + "/" + os.Args[1] + ".wasm"

  parameters := os.Args[2:][0]

  r := rt.New()

  wasmFunction := r.Register(calledFunction, rwasm.NewRunner(wasmModule))

  result, err := wasmFunction(parameters).Then()
  if err != nil {
    fmt.Println(err)
    return
  }

  fmt.Println(string(result.([]byte)))

}
```

Pour compiler notre exécutable, utilisez la commande suivante :

```
go build -o play
```

Et maintenant créons plusieurs runnables dans différents langages.

Hello Rust

Pour créer un runnable en Rust, utilisez la commande suivante :

```
subo create runnable hello-rust
```

Vous n'avez pas besoin d'utiliser l'option —lang=rust, Rust est le langage par défaut des Runnables.
Vous allez obtenir le code suivant (dans le répertoire hello-rust et le fichier src/lib.rs) :

```
use suborbital::runnable::*;

struct HelloRust{}

impl Runnable for HelloRust {
  fn run(&self, input: Vec<u8>) -> Result<Vec<u8>, RunErr> {
    let in_string = String::from_utf8(input).unwrap();

    Ok(String::from(format!("hello {}", in_string)).as_bytes().to_vec())
  }
}

// initialize the runner, do not edit below //
static RUNNABLE: &HelloRust = &HelloRust{};

#[no_mangle]
pub extern fn _start() {
  use_runnable(RUNNABLE);
}
```

Pour compiler le Runnable, utilisez la commande suivante :

```
subo build hello-rust
```

la CLI va produire un fichier hello-rust.wasm dans le répertoire hello-rust

Hello Assemblyscript

Pour créer un runnable en Assemblyscript, utilisez la commande suivante :

```
subo create runnable hello-assemblyscript —lang=assemblyscript
```

Vous allez obtenir le code suivant (dans le répertoire hello-assemblyscript et le fichier src/lib.ts) :

```
import { logInfo } from "@suborbital/suborbital"

export function run(input: ArrayBuffer): ArrayBuffer {
  let inStr = String.UTF8.decode(input)

  let out = "hello, " + inStr

  logInfo(out)

  return String.UTF8.encode(out)
}
```

Pour compiler le Runnable, utilisez la commande suivante :

```
subo build hello-assemblyscript
```

la CLI va produire un fichier hello-assemblyscript.wasm dans le répertoire hello-assemblyscript

Exécutez les modules

Maintenant, nous pouvons utiliser notre lanceur de la manière suivante :

```
./play hello-rust "Jane Doe"
# output: hello Jane Doe
./play hello-assemblyscript "John Doe"
# output: hello, John Doe
```

Je vous laisse essayer avec les autres langages. Mais avant de vous quitter, une dernière petite chose…

Utilisez les Runnables comme des microservices

Un de mes projets préférés chez **Suborbital** est le projet **Sat** (https://github.com/suborbital/sat) qui est un "mini serveur http" qui permet de servir un Runnable comme microservice (on pourrait dire un lanceur http de runnable). Nous allons donc pouvoir réutiliser tels quels nos Runnables.

Installer Sat

Vous pouvez utiliser la dernière release de **Sat** https://github.com/suborbital/sat/releases/download/v0.1.2/sat-v0.1.2-linux-amd64.tar.gz et copier l'exécutable sat dans /usr/local/bin.
Ou "builder" le projet **Sat** (ce que je fais par exemple, car il n'y a pas de release pour macOS) :

```
git clone https://github.com/suborbital/sat.git
cd sat
make sat
sudo cp .bin/sat /usr/local/bin
cd ..
rm -rf sat
```

"Lancer" les Runnables

Dans un terminal :

```
SAT_HTTP_PORT=8081 sat hello-rust/hello-rust.wasm
```

Dans un autre terminal :

```
SAT_HTTP_PORT=8082 sat hello-assemblyscript/hello-assemblyscript.wasm
```

Puis "interrogez" les services :

```
curl -d 'Jane Doe' http://localhost:8081
# output: hello Jane Doe
curl -d 'John Doe' http://localhost:8082
# output: hello, John Doe
```

Simple, non ?

Il devient très facile grâce à Reactr, les Runnables et Sat de créer des microservices à la fois légers et puissants. Et pour information, j'ai pu compiler sans problème **Sat** pour du arm et donc le faire "tourner" sur un Raspberry Pi, donc à suivre…
Vous pouvez retrouver les codes de cet article dans les projets suivants - https://gitlab.com/k33g_org/wasm.builders/play-with-runnables - https://gitlab.com/k33g_org/wasm.builders/play-with-runnables-next

Génération de code C# avec dotnet tool

Depuis la sortie du SDK .NET Core 2.1 dotnet tool vous permet de packager des applications consoles en C# destinées à tout type d'usage. Dans cet article nous allons voir au travers d'un projet exemple l'un des usages possibles : la génération en ligne de commande de classes C# à partir de fichiers d'échanges.

Sacha BAILLEUL
Développeur full stack axé environnement Microsoft actuellement en poste chez Neos-SDI.

Neos-SDI an open company

Fonctionnalités du projet exemple

Lien du projet : https://github.com/sbailleul/ToCSharp

Dans cet exemple, la fonctionnalité principale est la génération de définitions de classes C# à partir de fichiers contenant des données d'échange : yaml, json etc.

En complément l'interface en ligne de commande permet de :

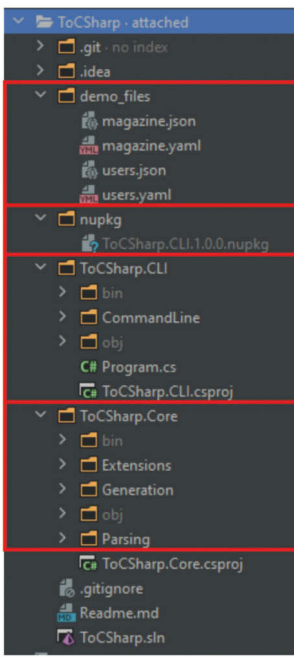
- Visualiser la liste des options disponibles
- Spécifier un fichier source de données
- Spécifier un répertoire de destination dans lequel seront enregistrés les fichiers .cs générés par la conversion du fichier source, si ce répertoire n'est pas spécifié le répertoire utilisé est le répertoire courant.

Vue d'ensemble du projet

Ce projet réalisé avec le SDK .Net 6 a une structure classique qui se décompose en 4 répertoires principaux :

- demo_files, contient la liste des fichiers utilisés pour vérifier le fonctionnement du générateur
- nupkg, contient le package nuget généré lors du build du projet ToCSharp.CLI
- ToCSharp.CLI, contient le projet qui sert de point d'entrée en ligne de commande de l'application
- ToCSharp.Core, contient le projet qui est au cœur du fonctionnement de l'application, c'est-à-dire la conversion de données d'échanges en classes C#. **Figure 1**

Figure 1



CLI

En l'état, vous pouvez faire fonctionner l'application comme toute application CLI réalisée en .Net via la commande **dotnet run.** Par exemple la commande suivante :

```
ToCSharp> dotnet run —project .\ToCSharp.CLI\ — —help
```

Affichera le résultat :

```
ToCSharp.CLI 1.0.0
Copyright (C) 2022 ToCSharp.CLI

-o, —output    Output directory for generated class definitions, default is current directory

-f, —file      Required. Source file for class generation, process e.g : file.yaml

—help          Display this help screen.

—version       Display version information.
```

Pour générer cette sortie, j'ai utilisé la librairie **CommandLineParser v2.8**, les usages de la librairie se trouvent dans le namespace **ToCShar.CLI.CommandLine**.

Fonctionnement du parsing JSON

Choix d'un parser

La classe **ToCSharp.CLI.CommandLine.ArgumentsProcessor** se charge de convertir les options passées en ligne de commande en valeurs interprétables par le programme. C'est via cette classe et sa méthode RunAsync qu'il est possible d'extraire le nom d'un fichier à traiter. Ce nom de fichier est ensuite utilisé dans la méthode de GetParser de la classe **ToCSharp.Core.Parsing.ExtensionMatcher.** Dans le cas d'un fichier avec une extension en .json le JsonParser est retourné par cette méthode.

Générateur de code

Le JsonParser une fois instancié est transmis à la classe **ToCSharp.Core.Generation.CSharpGenerator.** Cette classe se charge de déclencher le parsing et l'écriture des définitions de classe C# dans un répertoire spécifié par l'utilisateur.

JsonParser

La classe **ToCSharp.Core.Parsing.Json.JsonParser** implémente l'interface **ToCSharp.Core.Generation.Contracts.IParser** pour être utilisable en tant que classe de parsing. Sa méthode de Parse est le point d'entrée du parsing. Le mécanisme de parsing s'appuie sur la librairie **Newtonsoft.Json** pour effectuer une analyse récursive d'un arbre représentant les données stockées en JSON.

Une fois le parcours de l'arbre terminé toutes les définitions de classes détectées sont ensuite transformées en records immutables qui seront retournés par la méthode Parse.

Dotnet tool

Nous allons voir ensemble dans cette partie comment j'ai fait pour transformer cette application console en dotnet tool.

Transformation projet CLI en projet dotnet tool

Pour transformer le projet **ToCSharp.CLI** en dotnet tool j'ai modifié son .csproj en ajoutant 3 balises :

- **PackAsTool** indique que le package nuget généré lors du build est utilisable sous forme d'un dotnet tool
- **ToolCommandName** indique quelle sera la commande permettant d'utiliser l'outil, dans notre cas : tocsharp
- **PackageOutputPath** indique l'emplacement du répertoire où sera généré le package nuget.

Code complet sur programmez.com & github

Utilisation du dotnet tool dans votre environnement

Packaging du projet

Packagez l'application ToCSharp.CLI avec la commande suivante :

```
dotnet pack .\ToCSharp.CLI\
```

Ce package nuget correspond au dotnet tool et est stocké dans le chemin spécifié dans la propriété **Package OutputPath** du .csproj, c'est-à-dire le répertoire : **nupkg/.**

Installation globale

Pour installer le dotnet tool sur votre machine faite la commande suivante :

```
dotnet tool install —global —add-source ./nupkg ToCSharp.CLI
```

L'utilisation de l'option –global vous permet d'accéder à votre outil n'importe où sur votre machine.

Installation locale

Si vous souhaitez limiter l'utilisation de cet outil à votre répertoire de solution, vous pouvez suivre ces étapes.
Créez un manifeste avec la commande suivante :

```
dotnet new tool-manifest
```

Le manifeste est généré dans le fichier **.config/dotnet-tools.json**. Ce fichier liste les dotnet tools installés dans le répertoire de la solution. Pour installer le dotnet tool en local faites ensuite la commande :

```
dotnet tool install  —add-source ./nupkg ToCSharp.CLI
```

Cette commande déclarera le tool dans le fichier manifest généré précédemment :
Code complet sur programmez.com & github

Utilisation

Maintenant que vous avez terminé l'installation du dotnet tool vous pouvez tester son fonctionnement en vous basant sur fichier **demo_files/magazine.json** :
Code complet sur programmez.com & github
Si vous avez installé le dotnet tool globalement alors faites la commande suivante :

```
to-csharp —file .\demo_files\magazine.json —output from_magazine_json
```

Sinon si votre installation est locale, faites la commande suivante :

```
dotnet tool run to-csharp — —file .\demo_files\magazine.json —output
from_magazine_json
```

Peu importe le type d'installation, le résultat sera le même vous aurez dans le répertoire **from_magazine_json**/ des classes en C# correspondant avec le fichier passé en paramètre de l'option –file.


```
public class Magazine {
        public int PageCnt {get;set;}
    public bool IsCollector {get;set;}
    public Collection<Authors> Authors {get;set;}
    public DateTime PublicationDate {get;set;}
    public decimal WeightGrm {get;set;}
    public string Name {get;set;}
    public string Subject {get;set;}

}
```

Désinstaller le dotnet tool

Si vous avez installé le dotnet tool globalement faites cette commande :

```
dotnet tool uninstall tocsharp.cli —global
```

Si le dotnet tool est installé localement alors, faites cette commande dans le répertoire ou vous avez initialisé votre manifeste :

```
dotnet tool uninstall tocsharp.cli
```

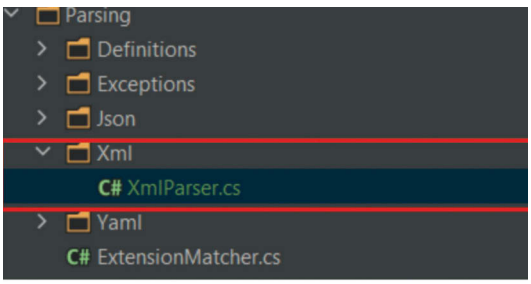
Ajouter votre parser personnalisé

Pour rajouter un parser supplémentaire, vous devez suivre les étapes suivantes :

- Créer une classe qui implémente **ToCSharp.Core.Generation.Contracts.IParser** ;
- Faire correspondre votre parser avec une extension via la méthode SetParser de la classe **ToCSharp.Core.Parsing.ExtensionMatcher.**

Par exemple pour rajouter un parser de fichiers xml suivez les étapes ci-dessous.

Création du parser




```
public class XmlParser: IParser
{
        public IImmutableSet<Class> Parse(TextReader txtReader)
  {
    throw new NotImplementedException();
  }
}
```

Déclaration du parser avec l'extension .xml

Dans la classe **ToCSharp.CLI.Program** remplacez :

```
await ArgumentsProcessor.ProcessAsync(args);
```

Par :

```
ExtensionMatcher.SetParser(".xml", () => new XmlParser());
await ArgumentsProcessor.ProcessAsync(args);
```

Conclusion

Dans cet article nous avons vu ensemble qu'il était possible de mettre en place un outil de génération de code extensible et qui s'intègre parfaitement à l'environnement .Net via l'utilisation de dotnet tools. Cette façon de générer du code permet d'avoir une maîtrise totale du processus de création de code, cependant sa mise en place est plus coûteuse que des outils clés en main. Pour aller plus loin vous pouvez personnaliser le projet présenté précédemment, sachez qu'il existe aussi d'autres projets qui utilisent dotnet tool pour vous permettre de générer du code, tel que dotnet-aspnet-codegenerator qui permet de générer des blocs CRUD de votre application en se basant sur des modèles de classe.

Patrick Prémartin
MVP Embarcadero, développeur freelance, formateur Delphi et web, occasionnellement streameur.

Delphi & C++Builder : low code par conception !

Delphi de Borland a été l'un des premiers environnements à fonctionner avec des composants pour créer les interfaces utilisateurs sous Windows. Quelques années après la sortie de Visual Basic et deux ans avant C++Builder suivis de Kylix, JBuilder, RadPHP, HTML5 Builder...

Delphi.OnBefore

L'ancêtre de Delphi c'est Turbo Pascal et sa librairie Turbo Vision qui permettaient sous MS-DOS de créer des interfaces utilisateurs avec boutons et boîtes de dialogue en mode texte. Quand Windows s'est répandu, Borland a modernisé Turbo Vision pour créer Visual Component Library (VCL) et sortir Delphi.

Delphi.OnCreate

Les programmes Delphi développés en Pascal contiennent des fiches qui représentent des fenêtres Windows (modales, non modales, MDI, SDI,...). Pour concevoir les interfaces utilisateurs, il suffit d'y déposer des composants. Ces composants correspondent aux éléments d'interface liés à Windows (champs de saisie, textes, dessins...).

Le gros avantage de Delphi a été de proposer dès le départ des composants pour accéder facilement aux bases de données du moment : Oracle, SQL Server, Paradox, dBase, MS Access et quelques autres. Ces éléments non visuels se rattachent à des éléments visuels permettant de développer des programmes de gestion de bases de données en ne faisant quasiment pas de code. Le tout se compile sous forme d'un exécutable autonome sans runtime. Ça constituait une vraie différence avec les programmes Visual Basic et les outils du marché.

Delphi.OnActivate

Comme vous avez pu le constater dans de récents numéros de Programmez!, Delphi a su évoluer durant ses 27 années, mais n'a rien perdu de son ADN : la simplicité de concevoir des écrans sans coder. L'approche RAD du départ est toujours présente.

En plus de Windows on peut compiler pour macOS, iOS, Android et Linux. Avec la librairie externe TMS Web Core et le transpileur Pas2JS on peut également faire des applications Web à partir de programmes en Pascal qui sont convertis en HTML/CSS/JavaScript.

C++Builder, « petit » frère de Delphi destiné aux développeurs C++, partage une grande partie de ses évolutions même s'il s'est recentré sur les développements Windows pour le moment. L'environnement de C++Builder est le même que celui de Delphi. Il bénéficie du même concepteur de fiches (VCL ou FMX), mais d'un éditeur de code adapté à la saisie de sources en C++.

Sur le même principe que Delphi, mais en open source, je me dois de citer l'IDE Lazarus. Cet environnement de développement s'est inspiré du fonctionnement de Delphi 7. Il utilise les compilateurs et une partie des librairies de Free Pascal Compiler (FPC). Son ensemble de composants pour interfaces utilisateurs s'appelle la LCL (Lazarus Component Library). Ils fonctionnent globalement comme les composants VCL. La différence principale c'est que Lazarus est lui-même utilisable sur Windows, Mac ou Linux alors que Delphi et C++Builder ne peuvent être utilisés en développement que sous Windows même s'ils proposent des compilateurs performants pour d'autres plateformes.

Delphi.OnShow

Depuis le début Delphi est à la limite d'un environnement no code. On peut créer des logiciels de gestion de bases de données simples juste avec les nombreux composants fournis (interface utilisateur et accès aux bases de données). Pour passer à du low code et des logiciels multifenêtres il suffit de connaître quelques commandes de base: Show, ShowModal, Close, ShowMessage et MessageDlg. Un « if » pour tester le retour des boîtes de dialogue et on sait traiter les actions des utilisateurs sur les écrans.

Bien entendu, si on veut faire des choses complexes, comme partout, on ne coupe pas à programmer un peu ou chercher dans le vaste écosystème disponible en ligne. L'avantage du Pascal étant qu'il est facile à apprendre même dans sa déclinaison objets.

À la différence de WinDev, Delphi ne propose pas en standard de composants métiers. Embarcadero a fait le choix de suivre la politique de Borland sur ce point en fournissant les éléments d'interface de base, ce qu'il faut pour les assembler, les personnaliser ou en créer sans aucune limite. Les codes sources des librairies et composants intégrés sont fournis avec les éditions payantes. Ce n'est pas le cas avec la Community Edition gratuite (usage commercial limité).

Les projets en Delphi et C++Builder

On peut créer tous types de projets, mais l'essentiel reste les applications avec UI. Pour cela nous avons la VCL basée sur les API de Windows et qui en suit les évolutions ou le framework FireMonkey (FMX) pour les applications multi-plates-formes. Deux frameworks, 2 approches différentes, mais globalement le même fonctionnement : un programme est composé de fiches, composants visuels ou non visuels.

Difficile de faire une présentation écrite d'un truc purement visuel donc je vais vous donner des pistes pour savoir quoi utiliser et comment. En cas de questions, n'hésitez pas à me contacter ou regarder mes vidéos en ligne.

Les fiches

C'est la base de tout projet. Il y a une fiche principale qui sert de première fenêtre de l'application et d'éventuelles fiches secondaires que l'on appelle par des options de menu ou des boutons dans l'interface.

Les composants

Les composants sont des classes spéciales référencées dans l'environnement de développement. Ils ont des propriétés et des événements (qu'on va oublier ici puisque ça voudrait dire coder des trucs).

Il y a des composants visuels (champs de saisie, listes, cases à cocher, images, boutons, etc.) et des composants non visuels (horloges, menus, listes d'actions, listes d'images, connexions aux bases de données, connexions à des API REST, boîtes de dialogues standards, etc.).

Le concepteur de fiches

Dans un projet Delphi ou C++Builder, chaque fiche a un fichier de code (Pascal ou C++) et un fichier ressource décrivant les éléments d'interface. Le concepteur de fiches est l'outil de l'IDE qui permet de gérer les écrans, leur aspect et leur contenu. Il donne accès à l'écran, ses composants, leurs propriétés et leurs événements.

Depuis le concepteur de fiches, on peut associer des propriétés de composants avec d'autres lorsque c'est prévu par les composants. Il est aussi possible d'utiliser l'assistant visuel Live Bindings pour associer des choses qui n'étaient pas prévues de l'être. Par exemple afficher le contenu d'un champ de saisie dans un champ texte lorsqu'il est modifié.

Live Bindings sert aussi à associer les tables et les résultats de requêtes de bases de données avec n'importe quel composant visuel.

Web services

Nous pouvons utiliser des composants REST pour interroger des API, récupérer les données sous forme de table en mémoire et afficher le tout comme n'importe quelle source de données.

Les listes d'actions

TActionList est un peu spécial. Il permet de centraliser des actions avec des éléments visuels d'interface communs par exemple des boutons de barres d'outils et des options de menus.

Dans ses entrailles se cachent des actions standards utilisées notamment pour passer d'un onglet à un autre d'un classeur à onglets ou accéder à des services liés à une API (comme lancer un appel téléphonique sur smartphone).

En pratique **Figure 1**

Pour l'exemple je vous propose de voir comment faire un CRUD sur un carnet d'adresses, sans code ni base de données. Vous pouvez télécharger la version Community Edition (usage personnel ou commercial limité) ou une version d'évaluation de Delphi ou C++Builder pour vous y mettre en testant ce projet.

Prenons une gestion pour des nom/prénom/numéro de téléphone. Un truc simple, mais qui nécessitait des heures de codage avant Delphi. **Figure 2**

Créez un nouveau projet en choisissant « Application VCL pour Windows » ou « Application multipériphérique » (FMX). Sur la fiche qui apparaît à l'écran, déposez un composant TFDMemTable. **Figure 3**

Faites un clic droit dessus et choisissez « éditeur de champs » pour ajouter 3 champs par Ctrl+N ou clic droit puis « nouveaux champs » : nom, prenom, telephone de type string en tant que champ de données. **Figure 4**

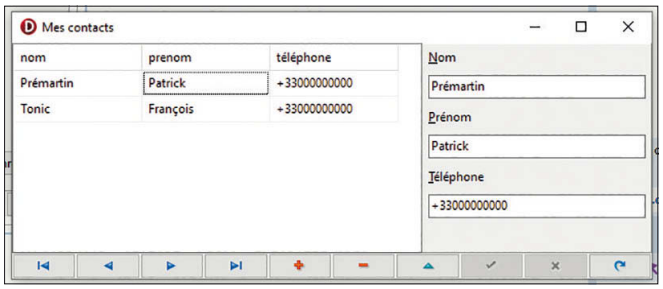


Figure 1

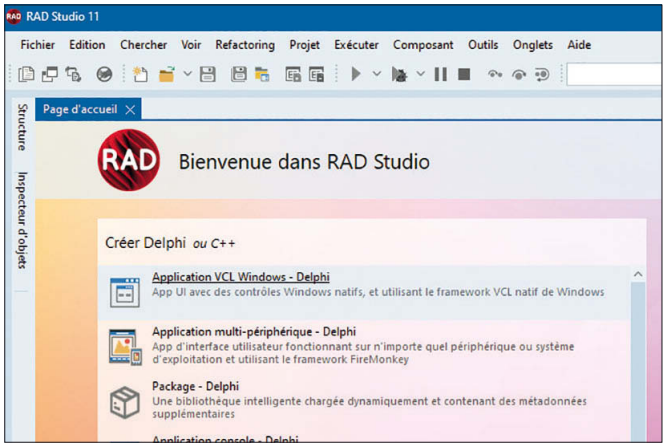


Figure 2

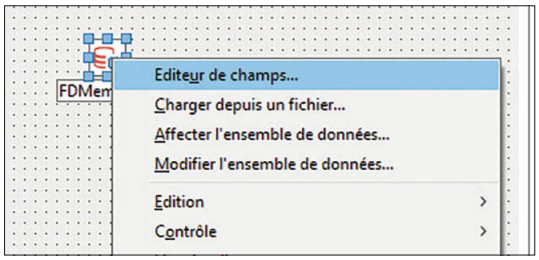


Figure 3

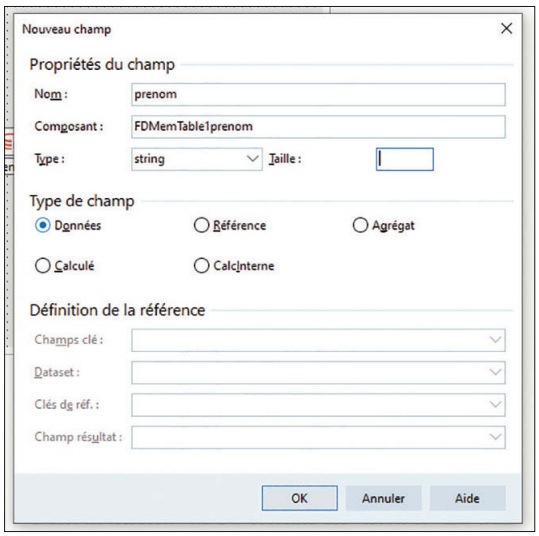


Figure 4

Fermez cet assistant et rendez-vous dans l'inspecteur d'objets pour FDMemTable1 à la rubrique « ResourceOptions ». Activez « Persistent » et mettez un chemin et un nom de fichier qui contiendra vos données dans « PersistentFileName ». **Figure 5**

Attention : c'est un chemin absolu. Vous ne pourrez utiliser ce programme que sur votre ordinateur ou sur un ordinateur ayant la même arborescence de fichiers. Pour choisir un chemin relatif, vous pouvez le faire dans l'événement onCreate de la fiche par exemple avec :

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 FDMemTable1.ResourceOptions.PersistentFileName :=
  'moncheminrelatifverslefichierdestockage';
 FDMemTable1.Open;
end;
```

Si vous ne choisissez pas l'option low code et préférez rester sur un mode no code passez la propriété « Active » du FDMemTable1 à « true » pour déclencher le mécanisme de chargement / enregistrement des données lors de l'ouverture du programme et sa fermeture. **Figure 6**

Ajoutez le composant TFDStanStorageXXXLink correspondant à l'extension choisie (XML, JSON ou BIN). Ce composant permettra de faire les déclarations nécessaires au travail avec ce format de fichier.

Vous avez ainsi configuré votre table de base de données. Passons à l'interface utilisateur.

Posez un composant TStringGrid sur l'écran. Modifiez sa propriété « Align » à « Client » ou « alClient » selon que vous êtes sur de la VCL ou FMX. **Figure 7**

Faites un clic droit dessus puis « lier visuellement ». Faites un drag and drop de l'astérisque de FDMemTable1 vers le même sur StringGrid1. L'affichage de la grille propose désormais les trois colonnes définies sur la table en mémoire. **Figure 8**

Ajoutez un TBindNavigator. Changez sa propriété « align » à « Bottom » ou « alBottom ». Retournez dans l'assistant LiveBindings et attachez son astérisque à celle de FDMemTable1.

Ajoutez un TPanel sur la fiche. Mettez son alignement à « Right » ou « alRight ». Videz sa propriété « Caption » (VCL) ou « Text » (FMX). **Figure 9**

Sur Panel1 ajoutez 3 TLabel et 3 TEdit. Dans la structure, sélectionnez les 6 composants et modifiez leurs propriétés « Align » à « Top » ou « alTop ». Modifiez leurs « Margins » en mettant 5 sur chaque côté. Si vous êtes en VCL activez leur « AlignWithMargins ».

Côté fiche, cliquez sur le panel, puis déplacez les TEdit et TLabel de telle sorte à en avoir un sur deux, puis associez les TLabel à « leur » TEdit en changeant leur propriété « FocusControl » puis leur « Caption » ou « Text » en le rem-

Figure 9

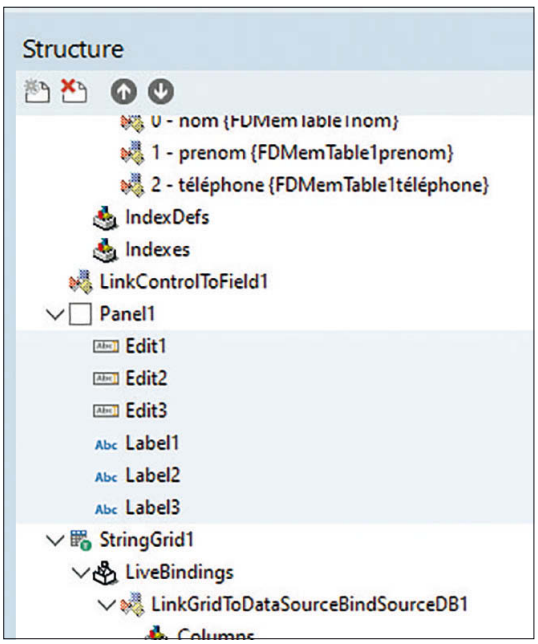


Figure 5

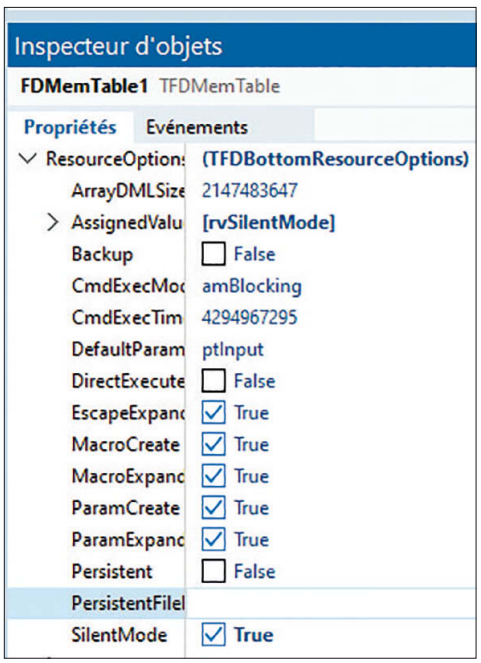


Figure 6

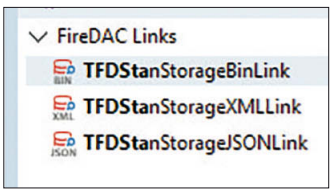


Figure 7

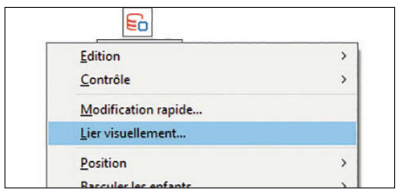
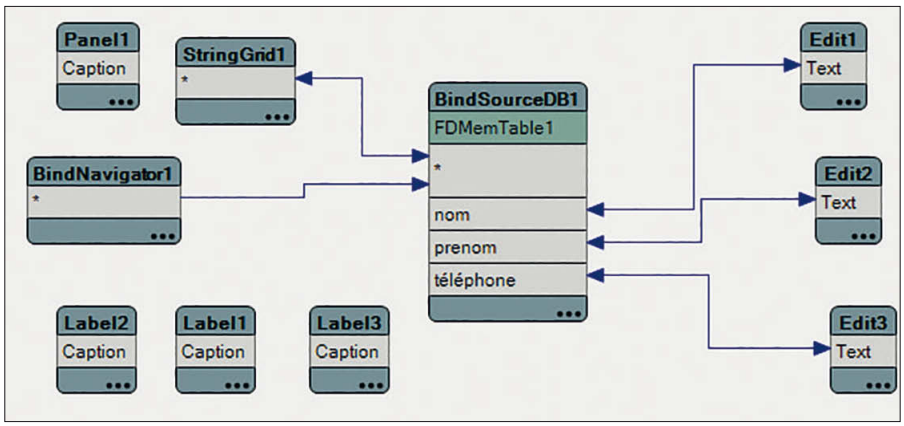


Figure 8



plaçant par le nom du champ à saisir (nom, prénom, téléphone). Pour activer un raccourci Alt+Lettre, préfixez la lettre désirée par « & » dans le texte.
Retournez dans l'assistant LiveBindings pour attacher la propriété « Text » de chaque TEdit avec l'un des champs de FDMemTable1.
Enfin, dans l'inspecteur d'objets, sur la fiche elle-même, changez la propriété « Caption » qui correspond au titre de la fenêtre. Vous avez terminé. Sauvegardez puis exécutez votre programme par F9 ou le menu « Exécuter / Exécuter ». **Figure 10**
Les applications mobiles ne ferment jamais proprement les programmes et par conséquent les données ne sont pas archivées automatiquement, car les connexions ne sont pas fermées, mais plantées.
Si vous êtes partis sur un projet FireMonkey pour Mac, iOS ou Android, vous devrez ajouter un bouton pour fermer puis ouvrir par code le FDMemTable1 afin de déclencher le backup de son contenu. Sur son événement onClick, mettez simplement ce code :

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 FDMemTable1.Close;
 FDMemTable1.Open;
end;
```

Ce n'est pas très compliqué d'aller plus loin avec ou sans code pour de la gestion de bases de données. Il suffit d'ajouter un TFDConnexion, le driver du moteur de base de données et des TFDTable ou TFDQuery.

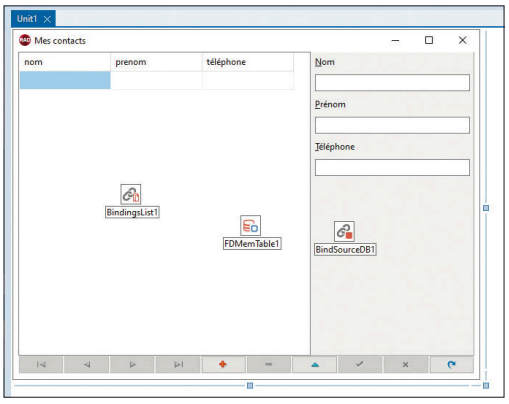
Delphi.OnCloseQuery

Delphi et C++Builder sont livrés avec des centaines de composants. Des exemples et des templates sont disponibles sur GetIt, son espace intégré de téléchargement d'extensions et produits complémentaires. Il y a aussi pas mal de choses sur GitHub et les autres plateformes d'hébergement de projets libres.
Sur GetIt on trouve également un assistant de création d'applications mobiles en Delphi. **Figure 11**
Une fois installé, « FireMonkey Low Code App Wizard » enchaîne les questions pour créer un projet multi-plate-forme à personnaliser. **Figure 12**
Paradoxalement il génère beaucoup de code pour faire fonctionner l'application mobile ainsi créée.
L'utilisateur peut se contenter de relooker les écrans et d'y mettre les données qu'il veut traiter (en mode CRUD). Ce n'est pas forcément très simple à faire dans la version actuelle du projet généré selon ce qu'on veut traiter. **Figures 13, 14 et 15**
Les écrans qu'il propose ont une version autonome en plusieurs déclinaisons directement téléchargeables comme exemples depuis GetIt.
Cet assistant de création de projets low code a encore du chemin à faire avant d'être réellement exploitable pour des applications en production, mais c'est un très bon début et ce qu'il génère peut servir d'exemple pour un vrai projet.

Delphi.onClose

Comme nous l'avons vu, il est assez simple de faire des choses simples sans ou avec très peu de code. Personnellement je persiste à penser que savoir coder est une bonne chose. Après tout, pour avoir des outils low ou no code il faudra toujours des développeurs capables de penser les systèmes, les composants et les connecteurs à mettre en place.

Figure 10



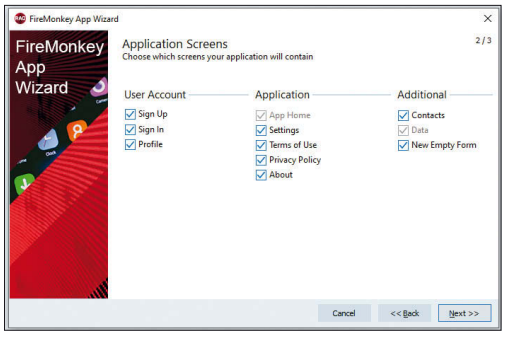
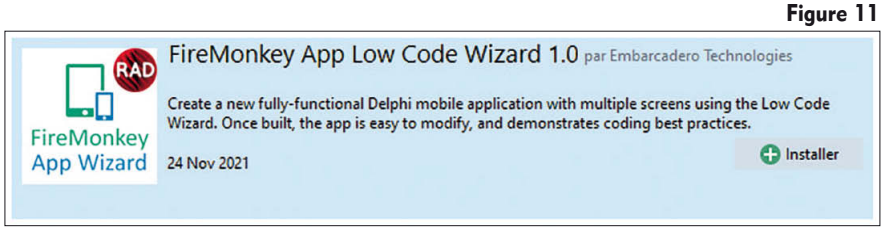
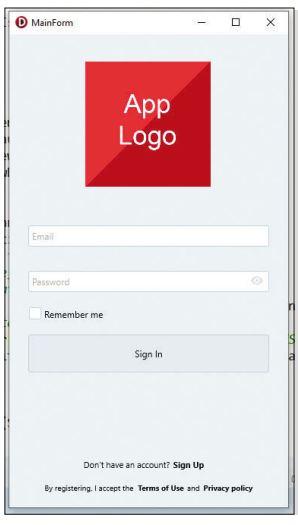


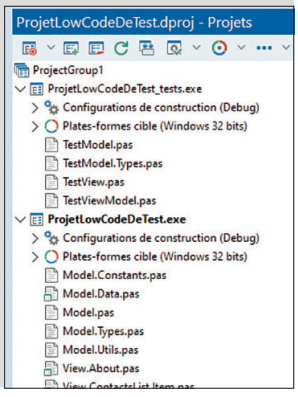
Figure 12

Figure 11





Figures 13, 14 et 15



Tiffany Souterre

Developer Relations à Microsoft. Après une thèse en ingénierie génétique, je me suis intéressée à l'intelligence artificielle. J'ai travaillé pendant 3 ans en tant que Data Engineer et je suis speakeuse en conférence depuis 2018 sur des sujets de Deep learning, tools in actions et Cloud Computing.

GitHub CoPilot : l'art de générer du code

Fin juin 2021, la technical preview de Github Copilot a été lancée. Développée par OpenAI et GitHub, Copilot est une intelligence artificielle de pair programming intégrée à l'environnement de développement dont la promesse est de suggérer des lignes voire des blocs de code à la volée en fonction du contexte. Le but n'est certainement pas de remplacer les développeur·euse·s, mais l'ambition affichée est d'améliorer leur productivité. Cet article est mon retour d'expérience après 10 mois d'utilisation de Copilot sur VSCode en Python. Alors, Copilot répond-il bien à ses attentes ? On vous dit tout.

La technologie derrière GitHub Copilot se nomme OpenAI Codex, une intelligence artificielle capable de traduire du langage naturel en code. Codex est basé sur un modèle de NLP (Natural Language Processing) appelé GPT-3 (Generative Pre-trained Transformer-3) dont l'entraînement a été affiné sur 54 millions de repositories publiques sur GitHub. Une fois installé dans l'environnement de développement, Copilot envoie le contexte, c'est-à-dire les lignes de codes et les commentaires, au service de Github Copilot par le biais d'une API. Ce contexte est analysé par Codex qui génère des suggestions alors proposées aux développeur·euse·s. **Figure 1**

Installation et prise en main

Copilot est disponible en extension de Neovim, JetBrains, VSCode et Visual Studio. Cet article est mon retour d'expérience de Github Copilot sur VSCode. Pour tester Copilot pendant la technical preview, il faudra au préalable vous inscrire sur le site officiel(1) et attendre d'être accepté.e pour y avoir accès. Une fois accepté.e, il vous suffira d'installer l'extension et en quelques secondes vous vous retrouvez en pair-programming avec Copilot.

Figure 1

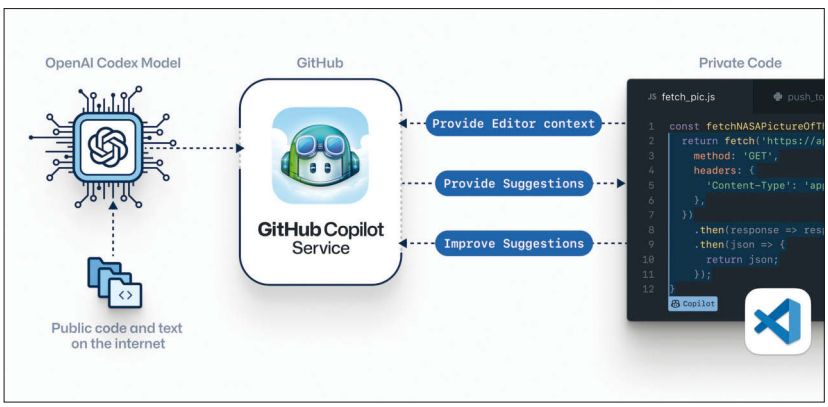


Figure 2

```typescript
#!/usr/bin/env ts-node

import { fetch } from "fetch-h2";

// Determine whether the sentiment of text is positive
// Use a web service
async function isPositive(text: string): Promise<boolean> {
  const response = await fetch(`http://text-processing.com/api/sentiment/`, {
    method: "POST",
    body: `text=${text}`,
    headers: {
      "Content-Type": "application/x-www-form-urlencoded",
    },
  });
  const json = await response.json();
  return json.label === "pos";
}
```

Un des aspects les plus importants dans la démocratisation d'outils d'IA, c'est l'expérience utilisateur. De ce point de vue, Copilot n'est pas en reste avec une interaction très simple et intuitive. Moyennant un accès à internet, Copilot est capable de vous faire des suggestions au fur et à mesure que vous codez, à la manière d'un outil d'autocomplétion. Libre à vous d'ignorer ces suggestions en continuant à écrire ou bien d'accepter en appuyant sur la touche Tab. Si vous n'êtes pas satisfait.e de la première proposition de Copilot, il est possible d'explorer d'autres propositions en cliquant sur "prev" ou "next" ou en appuyant sur CTRL + ENTRÉE ou Command + ENTRÉE sur macOS. Une fenêtre s'ouvre alors avec d'autres propositions que vous pouvez accepter en un clic. Si jamais vous ne souhaitez plus recevoir de recommandations, il vous suffit de cliquer sur l'icône de Copilot en bas à droite dans VSCode pour le désactiver. Personnellement, je trouve que l'expérience est très fluide et non intrusive. On pourrait redouter que les suggestions soient lentes à arriver, mais elles s'affichent réellement en quelques fractions de seconde ce qui est assez rapide pour ne pas freiner le rythme. Par contre, si vous êtes dans un endroit avec une connexion limitée, il est possible que Copilot ne fonctionne tout simplement pas.

Création de fonctions

Sur la page de présentation de GitHub Copilot, on peut voir des exemples à couper le souffle comme la fonction d'analyse de sentiments générée en Typescript à partir d'un simple commentaire. **Figure 2**

Un exemple tellement bluffant que l'on se demande si ce n'est pas que de la poudre aux yeux. Une étude de juillet 2021 démontre que OpenAI Codex est capable de générer des fonctions qui passent tous leurs tests unitaires à partir de docstrings dans 28.8% des cas et ce résultat peut monter jusqu'à 77.5% lorsqu'on laisse à Codex la possibilité de proposer jusqu'à 100 solutions(2). Après 10 mois d'utilisation de Copilot dans VSCode sur du Python, mon ressenti va dans le

sens de ces chiffres. Copilot a été, à ma grande surprise, très souvent capable de me proposer des solutions fonctionnelles ou presque. Un bon exemple qui m'a frappé reste celui du Sudoku Solver, car il est facile d'appréhender la complexité de ce genre d'algorithme. En donnant seulement un peu de contexte, Copilot est capable de proposer une solution qui passe tous les tests unitaires.
Voici le contexte écrit dans l'éditeur de code :

```
"""
A function that solves a Sudoku puzzle by filling the empty cells.
It doesn't return anything, just modifies the board in-place instead.
A Sudoku solution must satisfy all of the following rules:
Each of the digits 1-9 must occur exactly once in each row.
Each of the digits 1-9 must occur exactly once in each column.
Each of the digits 1-9 must occur exactly once in each of the 9 3x3 sub-boxes of the grid.
The '.' character indicates empty cells.
Input: board = [
    ["5","3",".",".","7",".",".",".","."],
    ["6",".",".","1","9","5",".",".","."],
    [".","9","8",".",".",".",".","6","."],
    ["8",".",".",".","6",".",".",".","3"],
    ["4",".",".","8",".","3",".",".","1"],
    ["7",".",".",".","2",".",".",".","6"],
    [".","6",".",".",".",".","2","8","."],
    [".",".",".","4","1","9",".",".","5"],
    [".",".",".",".","8",".",".","7","9"]
    ]
Output: [
    ["5","3","4","6","7","8","9","1","2"],
    ["6","7","2","1","9","5","3","4","8"],
    ["1","9","8","3","4","2","5","6","7"],
    ["8","5","9","7","6","1","4","2","3"],
    ["4","2","6","8","5","3","7","9","1"],
    ["7","1","3","9","2","4","8","5","6"],
    ["9","6","1","5","3","7","2","8","4"],
    ["2","8","7","4","1","9","6","3","5"],
    ["3","4","5","2","8","6","1","7","9"]
    ]
"""
```

Et voici la proposition de Copilot :

```
from typing import List

def solveSudoku(board: List[List[str]]) -> None:
    def isValid(board, row, col, num):
        for i in range(9):
            if board[row][i] == num:
                return False
            if board[i][col] == num:
                return False
        for i in range(3):
            for j in range(3):
                if board[row//3*3+i][col//3*3+j] == num:
                    return False
        return True

    def dfs(board, row, col):
        if row == 9:
            return True
        if col == 9:
            return dfs(board, row+1, 0)
        if board[row][col] != '.':
            return dfs(board, row, col+1)
        for i in range(1, 10):
            if isValid(board, row, col, str(i)):
                board[row][col] = str(i)
                if dfs(board, row, col+1):
                    return True
        board[row][col] = '.'
        return False

    dfs(board, 0, 0)

```

Le Sudoku Solver reste un cas générique, mais cela démontre que Copilot est capable de résoudre des algos d'une certaine complexité. Concrètement, à quoi cela peut-il bien servir dans la vie de tous les jours ? Il est évident que Copilot ne va pas développer une application entière à partir d'un paragraphe, il n'est pas fait pour cela. Par contre, un.e développeur·euse qui a un projet en tête va pouvoir penser à l'architecture de son code et ensuite la fractionner en plusieurs blocs eux-mêmes fractionnés en d'autres blocs. Et c'est dans l'aide à la création de ces petits blocs que Copilot exprime tout son potentiel. En effet, les projets ont beau être différents, les blocs eux ont souvent une logique commune. Bien que Copilot n'ai pas toujours raison, cela peut représenter un gain de temps conséquent s'il a pu à certains moments proposer des solutions correctes ou qui ont mené à des solutions correctes. C'est dans cette configuration que le tandem de pair programming humain/AI prend tout son sens. L'humain est là pour superviser et rester au contrôle de la création de son projet, mais Copilot peut ponctuellement apporter un coup de pouce sous réserve de garder un œil critique sur chacune des suggestions qui sont faites. Cela peut sembler aller de soi, mais dans la pratique c'est plus facile à dire qu'à faire. Il est très facile de tomber dans l'excès de confiance que l'on donne à Copilot et d'accepter toutes ses propositions. Il est important de rester vigilant·e, car cela pourrait porter préjudice à votre projet (i.e failles de sécurités, introductions de bugs…).

Tests

Un code robuste est un code bien testé. Seulement, ces tests peuvent parfois être négligés. Copilot peut rendre cette tâche moins rébarbative en suggérant des blocs de tests. Dans le cas du Sudoku Solver voici ce que Copilot a proposé avec très peu de contexte. **Figure 3**

On remarquera que Copilot a su analyser qu'il y avait déjà un exemple d'input et d'output dans le commentaire de la fonction `solveSudoku` qui se trouve dans le fichier "main.py" alors

main.py | test_main.py ×

test_main.py > TestMain > test_solveSudoku

```
import unittest
from main import solveSudoku

class TestMain(unittest.TestCase):
    def test_solveSudoku(self):
        """a test for the solveSudoku function"""
        board = [
            ["5","3",".",".","7",".",".",".","."],
            ["6",".",".","1","9","5",".",".","."],
            [".","9","8",".",".",".",".","6","."],
            ["8",".",".",".","6",".",".",".","3"],
            ["4",".",".","8",".","3",".",".","1"],
            ["7",".",".",".","2",".",".",".","6"],
            [".","6",".",".",".",".","2","8","."],
            [".",".",".","4","1","9",".",".","5"],
            [".",".",".",".","8",".",".","7","9"]
        ]
        solveSudoku(board)
        assert board == [
            ["5","3","4","6","7","8","9","1","2"],
            ["6","7","2","1","9","5","3","4","8"],
            ["1","9","8","3","4","2","5","6","7"],
            ["8","5","9","7","6","1","4","2","3"],
            ["4","2","6","8","5","3","7","9","1"],
            ["7","1","3","9","2","4","8","5","6"],
            ["9","6","1","5","3","7","2","8","4"],
            ["2","8","7","4","1","9","6","3","5"],
            ["3","4","5","2","8","6","1","7","9"]
        ]

if __name__ == '__main__':
    unittest.main()

```

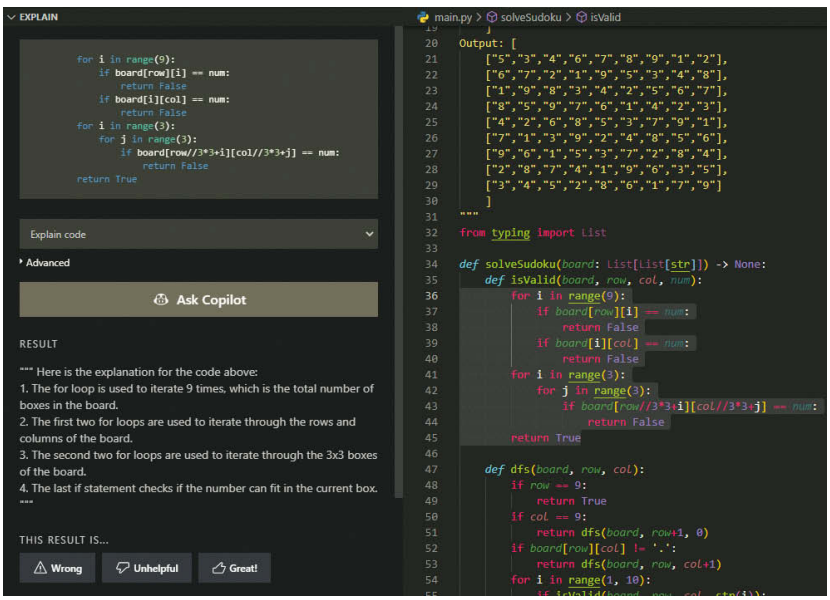
Figure 3

que mon test test_solveSudoku se trouve dans le fichier "test_main.py" et qu'il a su réutiliser cette information pour générer le test unitaire. Ce test proposé reste simple, car il n'y avait pas un fort contexte de tests unitaires dans mon projet sur lequel s'inspirer, mais il a tout de même été capable de proposer quelque chose de cohérent. Il faut aussi garder à l'esprit que le·la développeur·euse a toujours la responsabilité de bien vérifier que tous les edges cases sont bien testés et ne pas partir du principe que Copilot les trouvera pour vous.

Github Copilot labs

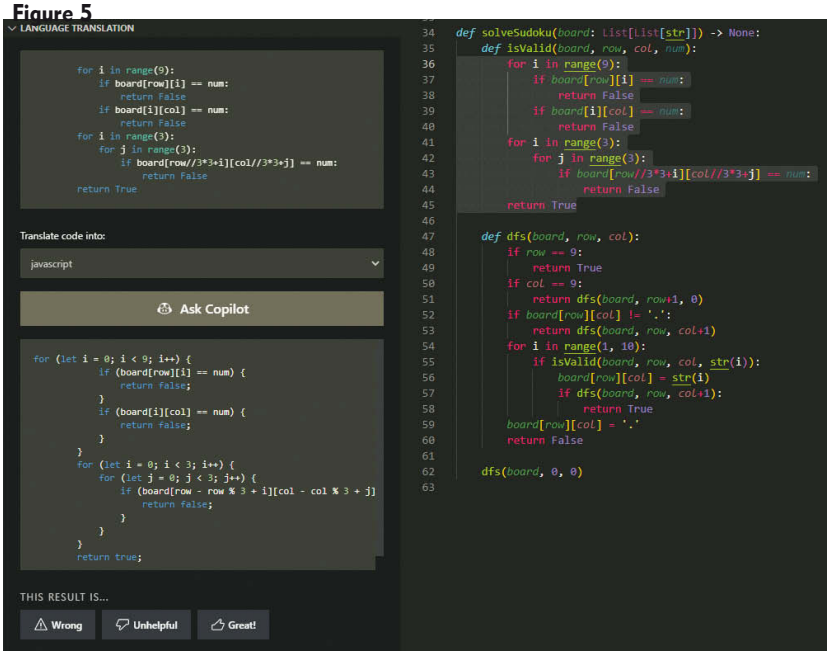
Une extension supplémentaire, https://marketplace.visualstudio.com/items?itemName=GitHub.copilot-labs, est maintenant disponible et ajoute deux nouvelles fonctionnalités. Une première, appelée "Explain", permet de sélectionner une partie du code et d'en demander une explication. Par exemple, dans l'image ci-dessous, j'ai sélectionné le contenu de la fonction isValid qui a été analysé par Copilot. Ce dernier a généré un texte dans l'onglet de gauche avec toutes les étapes logiques de la fonction isValid. Cette fonctionnalité peut s'avérer très utile pour apprendre à coder ou lorsque l'on travaille sur du code legacy et/ou mal documenté. **Figure 4**

Figure 4



Une seconde fonctionnalité, appelée "language translation", permet de traduire une partie de code d'un langage vers un autre. Par exemple ici, nous demandons une traduction de Python vers JavaScript de la fonction isValid. Je ne pense pas que cette fonctionnalité soit utilisée quotidiennement par les développeur·euse·s, mais elle peut être très intéressante dans un cadre pédagogique pour apprendre un nouveau langage. **Figure 5**

Figure 5



Création de documentation

Tout comme les tests, l'écriture de documentation et de readme est critique pour partager un projet. Cependant cette partie demande aussi beaucoup de travail et peut parfois être boudée des développeur·euse·s. Copilot peut là aussi vous rendre la tâche plus facile en vous proposant cette fois-ci des blocs de texte. À plusieurs reprises dans mes markdowns, Copilot a été capable de me proposer des choses très pertinentes. Bien sûr cette fonctionnalité marche mieux en anglais, mais j'ai testé Copilot en français et j'ai tout de même été surprise par ses propositions.

Conclusion

Copilot réussit son tour de force de fournir une AI de pair programming et répond parfaitement à l'ambition d'améliorer la productivité des développeur·euse·s. Copilot est disponible sur Neovim, JetBrains, VSCode et Visual Studio. J'utilise VSCode et son installation est simple et l'expérience utilisateur est très fluide sans lag à condition d'avoir une bonne connexion internet. Certes les suggestions ne sont pas toujours parfaites, mais elles peuvent par moment offrir une vraie valeur ajoutée lors de l'écriture de fonctions, de tests unitaires et de documentation. Attention toutefois à ne pas tomber dans l'excès de confiance. Une bonne utilisation de Copilot demande de garder un esprit critique sur ce qui est proposé sous peine d'intégrer des failles de sécurité ou des bugs dans votre code.

Copilot s'avère aussi être un allié de taille d'un point de vue pédagogique et peut aider à accélérer l'apprentissage de nouveaux frameworks ou de nouveaux langages.

Déjà en technical preview, Copilot est un outil étonnant qui n'a pas manqué de me surprendre. C'est devenu un incontournable de mes extensions VSCode. Hâte de voir la version finale !

Ressources

(1) https://copilot.github.com/
(2) https://arxiv.org/pdf/2107.03374.pdf

JHipster, la boîte à outils sauce *low code* ?

Si vous êtes passé à côté de Programmez! **#223** et **#225,** je vous propose de reprendre l'histoire à zéro en démarrant par une brève introduction de JHipster. JHipster a été créé par Julien Dubois - Java Champion - en 2014 et compte à ce jour plus de 650 contributeurs, 19 400 étoiles sur GitHub et 130 000 téléchargements par mois.

Anthony Viard

Développeur Java et JavaScript depuis 2010, c'est en 2018 qu'Anthony découvre JHipster au cours d'un projet chez un client. Cela l'amènera à contribuer à ce projet Open Source, puis à y intégrer la core-team. En 2020, Anthony rejoint la société Entando, sponsor de JHipster, afin de contribuer à la personnalisation de son Blueprint. Il y occupe depuis le rôle de "Developer Advocate" et participe à la médiatisation des Composable Apps, incluant microservices, micro frontends ou encore fonctionnalités low code pour composer des applications.

L'idée de départ était simple : proposer un outil efficace pour coder rapidement, en utilisant les solutions techniques les plus "branchées" (hipster) et à l'état de l'art. Il fallait ainsi optimiser le temps de démarrage des projets et aider les équipes de développement à se concentrer sur l'essentiel, le code métier. Dans le même temps, des frameworks comme Spring Boot ou AngularJS s'étaient imposés comme des références dans leur domaine, laissant l'opportunité de combiner ces technologies pour générer des applications "fullstack". Avec le temps, les contributions augmentaient et permettaient à JHipster d'enrichir son éventail d'options en incorporant d'autres frameworks ou faisant évoluer les existantes.

L'arrivée de la génération d'entités, à partir d'un DSL (*Domain Specific Language*) inspiré de l'UML, le *JHipster Domain Language* (JDL), permettait de couvrir des aspects métiers plus importants et renforçait l'utilisation de JHipster tout au long du cycle de vie du projet. **Figure 1**

Désormais, JHipster ne se contentait pas seulement de produire une stack technique prête à l'emploi, mais permettait aussi de générer un socle métier, au sein notre application, du domaine à l'interface graphique en passant par les API. Les objets métiers (entités), leurs propriétés, leurs relations et même certaines règles de gestion (taille, contraintes, RegExp…) pouvaient être gérés afin d'obtenir une application aussi proche que possible de sa version finale. L'IHM générée couvrait les actions de CRUD (Create, Read, Update, Delete) en utilisant les frameworks Angular, React ou Vue.js.

JHipster n'était donc plus un "starter" d'application, mais un véritable outil de scaffolding, couvrant la totalité des couches des applications modernes. Il ne restait qu'à y ajouter le code métier, d'y personnaliser l'aspect graphique et de gérer le cycle de vie de l'application.

Et le low code ?

À en croire les leaders du marché, le *low code* est une approche de développement logiciel qui facilite la phase de développement au profit de l'utilisation de solutions clé en main, le plus souvent au travers d'une plateforme dédiée, par exemple en construisant des applications depuis une IHM.

Ainsi, si la majeure partie des fonctionnalités d'une application peuvent être couvertes par la solution utilisée, les équipes de développement doivent couvrir la partie restante par du code, quand cela est nécessaire.

La question que je souhaite me poser est de savoir si cela est applicable à d'autres solutions que les plateformes applicatives, par exemple pour des outils de développement comme JHipster.

Peut-on réduire, pour le développeur, le reste à charge à coder au point de considérer cet outil comme "*low code*" ? Ou bien est-ce quelque chose de totalement différent ?

Dans le paragraphe précédent, nous avons retracé l'histoire de JHipster, compris comment nous sommes arrivés à un outil plus abouti que prévu et ainsi proposer une expérience de génération de code plus complète.

De ce fait, si JHipster n'a pas été pensé comme un outil *low code*, au premier abord, les contributions successives lui ont donné à la fois une qualité de *starter* d'application puis de *builder* de domaine métier.

Bien que ces notions ne soient pas antinomiques, on imagine bien que l'expérience développeur ne soit pas la même quand il s'agit de construire la base technique, ou de se concentrer sur la définition du domaine métier.

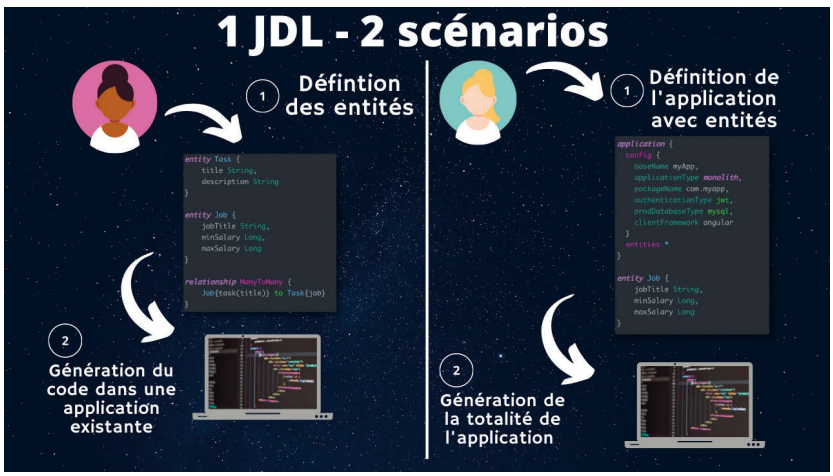
En générant la totalité du socle applicatif, en définissant votre modèle métier avec le JDL, puis en générant votre domaine, vos services, vos API et enfin l'IHM idoine, on peut alors légitimement se demander où se place JHipster sur l'échiquier du *low code*. C'est la question à laquelle nous allons tenter de répondre.

JHipster la boîte à outils du développeur ?

Au-delà d'un générateur accessible depuis une CLI ou depuis une IHM (voir https://start.jhipster.tech), JHipster reste une solution au niveau développement et ne propose pas de solution applicative de haut niveau.

Je me suis longtemps interrogé : "comment définir la méthodologie qui consiste à baser une partie de son application sur le résultat d'un générateur ?", cela a-t-il une influence sur la définition de mon propre travail ?".

Figure 1



En tant qu'utilisateur de JHipster, étais-je devenu un développeur "*low code*" ?
Il s'agit d'abord de comprendre le contexte du projet, l'expérience acquise par l'équipe ou encore la complexité du besoin métier.
C'est en tenant compte de tous ces paramètres que nous serons à même de saisir notre capacité à tirer partie de JHipster et de comprendre pourquoi nous devrions le faire.
En effet, je suis intimement convaincu que l'utilisation de JHipster a du sens pour réduire le temps de mise en production, aider le développeur à se concentrer sur le besoin métier et obtenir un livrable de meilleure qualité.
De la simple boîte à outils à une méthode de travail, l'impact d'un générateur de code peut-il être au point de devoir repenser notre manière de définir le travail ?
Mais alors, JHipster serait-il un outil *low code* au service des développeurs ?

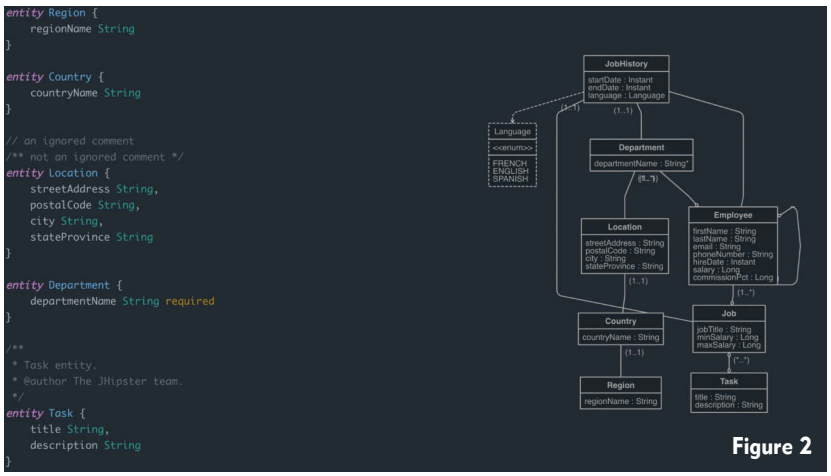


Figure 2

Figure 3

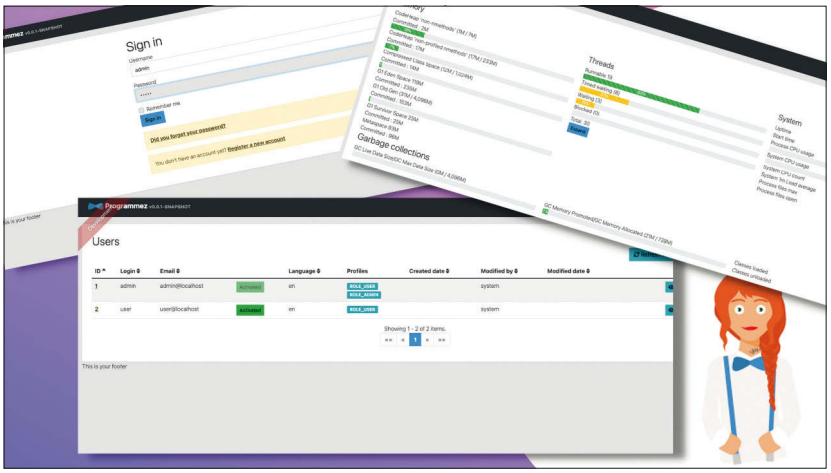


Figure 4

C'est ce que nous allons essayer d'appréhender dans le paragraphe suivant.

Générer une application sans "coder"

Afin d'aider les développeurs dans la création de leur JDL (voir premier paragraphe), la communauté met à disposition un éditeur en ligne appelé *JDL Studio* qui permet à la fois de définir son schéma, de visualiser les entités et leurs relations, et de pouvoir le télécharger.
Par défaut, et à titre de démonstration, le studio propose un exemple définissant l'organisation interne d'une société. Il décrit la gestion des employés, les tâches attribuées, le département rattaché ou encore l'historique des emplois occupés. C'est ce JDL que nous allons utiliser par la suite.
Figure 2. Retrouvez le *JDL Studio* à cette adresse: https://start.jhipster.tech/jdl-studio.
Pour débuter, je m'assure d'avoir JHipster installé (dans le cas contraire je lance la commande *npm i -g generator-jhipster*) puis j'exécute la commande *jhipster* dans mon dossier cible.
Figure 3

Je choisis de générer une application monolithique et je choisis toutes les valeurs par défaut. Enfin, une fois le projet généré, je m'assure de le pousser sur un référentiel Git afin d'en conserver l'historique.
Je peux, dès lors, démarrer l'application avec la commande *./mvnw*. Disponible sur le port 8080, je constate que l'application dispose d'une interface graphique complète, d'un système d'authentification fonctionnel (les comptes par défaut étant *admin/admin* et *user/user*) et de quelques écrans d'administration me permettant, par exemple, de gérer les utilisateurs ou d'accéder aux métriques de mon application.
Figure 4

Après cette première phase de découverte sur cette version "nue" de mon application (comprenez par-là : sans mon socle métier), je me connecte au *JDL Studio* et je télécharge le JDL d'exemple que je place dans le dossier du projet.
J'importe mon JDL via cette commande : *jhipster import-jdl jhipster-jdl.jdl*.
Notez qu'au passage, il est possible que vous deviez valider la surcharge de certains fichiers déjà présents ; acceptez.
Une fois cette seconde phase de génération terminée, je peux redémarrer mon application et constater que le menu *entités* contient désormais l'ensemble de mes objets définis dans mon JDL : *Employee*, *Job*, *Job History*…
Chaque élément de ce menu déroulant nous conduit à un écran qui liste les données de mon entité, m'autorise à en ajouter, en supprimer ou en modifier. Lorsque l'application est démarrée avec le profil *dev*, profil par défaut, JHipster va intelligemment remplir notre base de données, grâce à la librairie FakerJS, très utile pour tester et valider rapidement un comportement. La fonctionnalité n'est pas active en production.
Côté code, on trouvera une couche d'API, de services et DTO, de repositories et, bien sûr, d'entités JPA. Le JDL accorde une certaine flexibilité en permettant de générer ou non les services et les DTO, d'utiliser des interfaces pour les services ou encore d'activer certaines fonctionnalités, comme la pagination. **Figure 5**

Que ce soit pour le backend ou le frontend, un ensemble de tests unitaires et d'intégrations sont aussi mis à disposition afin de bénéficier d'une bonne couverture pour un tel projet. Finalement, on peut voir que cette approche JHipster, couplée au JDL, tire parti de la rapidité d'un générateur tout en étant guidé par le métier. Dans mon étude de cas, j'ai effectué l'étape en deux temps afin de mettre en avant l'apport du JDL, cependant il est tout à fait possible d'effectuer cela en une seule fois.

En effet, le JDL définit directement notre application, les options choisies, les frameworks, la base de données, etc. Ceci remplace donc la phase questions-réponses du CLI. Il est possible, à partir d'un fichier de description (le JDL), d'obtenir une application prête pour la production.

On peut alors affirmer que JHipster peut être approché de deux manières différentes. La première scinde la partie génération (choix des solutions techniques) et la définition des entités métiers. La seconde *JDL centric*, avec une approche *as code*, où le JDL dirige à la fois la définition métier et technique.

Quelle que soit l'approche adoptée, le résultat restera identique et l'on bénéficiera de la même expérience en développement et en production.

Pour aller plus loin, je pense que l'approche *API-First* est aussi à considérer. En sélectionnant l'option *API first development using OpenAPI-generator* de JHipster, on va tirer parti d'OpenAPI par le plug-in Maven afin de définir une d'API depuis un fichier de description. **Figure 6**

Ce fichier contient l'ensemble des points de terminaison, les objets, les règles, les codes HTTP, les messages d'erreur (et bien plus) que l'application va exposer. C'est cette nouvelle couche que l'on va étendre afin d'implémenter notre code et faire le lien avec la partie obtenue par JHipster. Dans ce cas de figure, nous bénéficions de l'apport de JHipster pour définir la structure applicative, le domaine métier et d'Open API pour notre API métier. C'est par ces différents exemples que l'on peut établir une cartographie de ce que promet le développement par générateur. L'approche JDL centric, va couvrir des actions CRUD, une intégration homogène des outils et réduire le Time To Market. L'ajout d'OpenAPI permettra de définir une API plus précise, plus orientée "fonctionnalités". Cependant, il sera rapidement nécessaire de couvrir une partie du besoin par du code et la limite de l'outil risque d'être facilement atteinte, peu nombreuses sont les applications qui se contentent d'un simple CRUD.

Aussi, plus le projet avancera dans son cycle de vie, plus le développeur aura pour charge de maîtriser la base de code. Pour la maintenir ou l'améliorer.

Conclusion

Cet article se termine déjà, et j'ai l'impression d'avoir à peine effleuré le sujet, tant il est passionnant et complexe à la fois. Nous aurions pu aborder la génération de pipelines CI/CD, de fichiers de déploiement vers le Cloud, de l'internationalisation ou encore de la configuration des conteneurs.

Nous en conviendrons, la nature même du développement fait qu'il est très difficile d'imaginer un outil qui permettrait de couvrir tous nos besoins, tout le temps.

C'est pour cela que le rôle de développeur reste prépondérant.

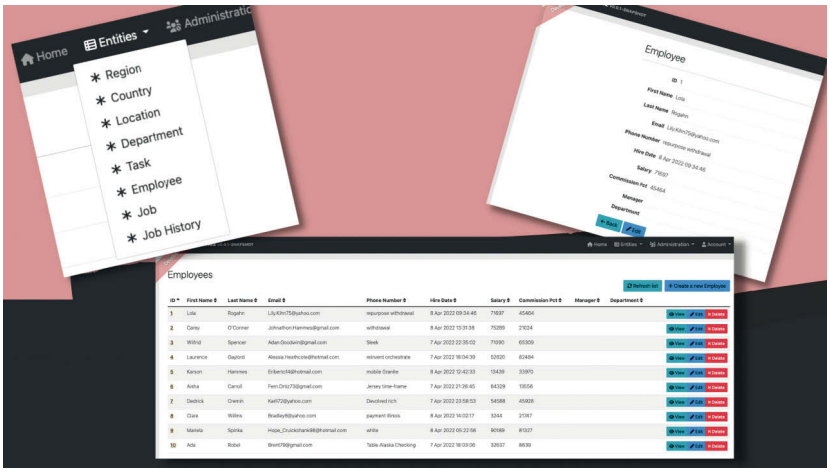


Figure 5

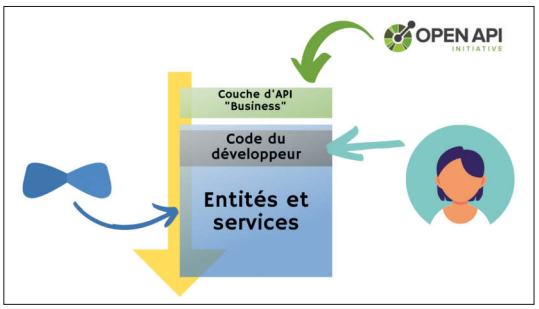


Figure 6

Nous l'avons vu, on peut facilement, à partir d'une description, obtenir du code fonctionnel. Il n'en reste pas moins vrai qu'un travail d'intégration et de développement est à effectuer, que ce soit par l'ajout de règles de gestion, par la personnalisation de l'IHM, par l'intégration dans un système d'information, etc.

Il serait peu concevable d'abandonner totalement cet aspect technique au profit d'outils dont nous ne maîtrisons pas toujours le potentiel.

Derrière toute chose simple se cachent des choses complexes, et nous avons la responsabilité de comprendre cela.

Si l'on considère l'approche JDL comme une approche *low code*, il faut prendre conscience qu'une limite peut être vite atteinte. Selon la complexité de nos besoins, la promesse du *low code* peut vite s'envoler.

Tout ceci m'amène à penser que JHipster est à la fois très proche de cette promesse et est à la fois au-delà, nécessitant indubitablement une maitrise technique élevée.

Selon notre vision, le contexte dans lequel nous évoluons et nos attentes, nous aurons probablement tous une réponse différente à apporter.

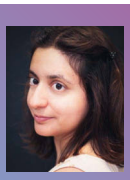
Je vous assure que mes origines Normandes n'y sont pour rien dans cette conclusion, peut-être bien que ce soit du *low code* ou peut-être bien que non.

Finalement, ce qui importe est notre capacité à tirer le meilleur des outils à notre disposition, en obtenir la quintessence, qu'ils soient *low code* ou pas.

J'espère que cet article mettra en lumière une pratique de développement que j'apprécie et que j'ai appris à maîtriser au fil des années. Que cela vous aidera à vous faire votre propre avis sur la question. Aujourd'hui, c'est pour moi un réflexe de penser "générateur", le fait de coder moins me permet probablement de coder mieux.

Si le meilleur moyen de maîtriser un outil est de le connaître, quoi de mieux que d'y contribuer ?

Un bot conversationnel en low code



Aurore de Amaral

Senior backend developer à Meetic.

Développeuse backend depuis 2015 et intéressée par toutes les nouvelles technologies autour de l'interface conversationnelle et le traitement du langage naturel (NLP)

Ces dernières années, les solutions d'automatisation de type IFTTT(1) sont plus attractives que la création de chatbot pour les tâches répétitives. Cependant, la limitation technique des solutions d'automatisation tend à se pencher encore sur les solutions low code des bots conversationnels. Nous allons aller plus loin dans cet article et voir où les bots ont encore leur atout avec l'outil de Google Actions(2).

Rappel : *cet article n'est toujours pas sponsorisé par Google ou Todoist ;)*

Cet article est une suite à l'article paru dans le n°250 sous le titre "Un bot #(quasi)nocode sur la Google Home".

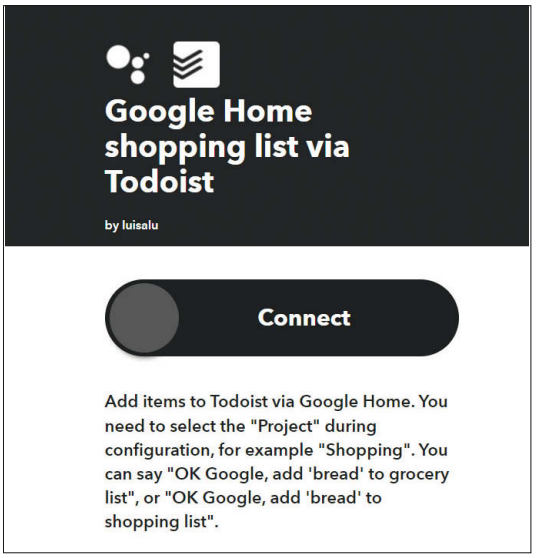
L'idée du premier article était d'avoir un bot qui liste nos tâches du jour avec Todoist(3) et la Google Home. Mais qu'en est-il pour la création de tâches, qui peut se faire aisément avec IFTTT ? Ne serait-ce pas plus facile de faire toutes ces nouvelles fonctionnalités dans le même bot ? Voyons comment faire avec Google Actions !

Pré-requis : account linking

Une documentation(4) en ligne (ou le précédent article) explique comment faire communiquer Todoist et Google Actions. C'est un prérequis indispensable pour faire fonctionner le bot.

Scénario conversationnel

Voici le scénario que nous allons mettre en place (aka conversation design(5)) :

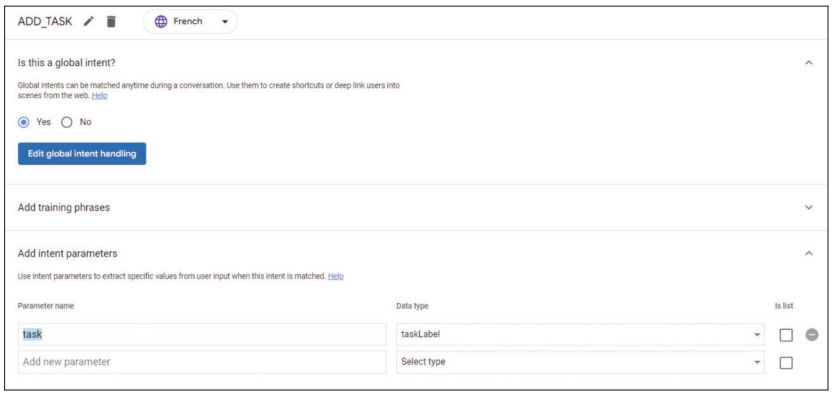


- *OK Google, parler à programme Todoist*
- Bonjour je suis Todoist pour vous aider à créer des tâches [1]
- *Liste moi mes tâches*
- (liste les tâches d'aujourd'hui ou aucune tâche). Voulez-vous en créer une ? [2]
- *Oui*
- Quelle est votre tâche ? [3]
- *Aller voir mamie*
- Bien, la tâche aller voirie va être créé [4]
- *Ah non, je voulais dire aller voir mamie*
- Bien, la tâche aller voir mamie va être créée [5]
- *Ok*
- La tâche a été créée. Au revoir ! [6]

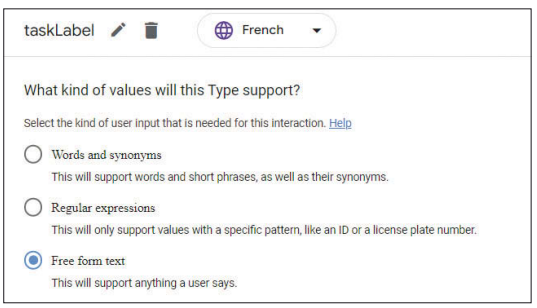
Il est aussi prévu d'appeler directement la liste des tâches et la création d'une tâche à Google.

1. Définir les intentions

On définit tout d'abord les intentions (NLP pour Natural Language Processing avec DialogFlow(6)) qui vont nous servir à comprendre ce que dit notre utilisateur en donnant des phrases d'exemples. Il faut créer une intention AGREE qui comprend des phrases d'entraînement comme oui, ok, ADD_TASK avec aller voir mamie, et contient une entité **task** avec un nouveau datatype taskLabel en free form text), DISAGREE avec ah non, c'est pas ça, … et la même entité **task** et LIST_TASKS avec Tâches à faire, liste moi mes tâches. **Figures 1 et 2**



Figures 1 et 2



(1) https://ifttt.com/

(2) https://developers.google.com/assistant/conversational

(3) https://todoist.com/fr/home

(4) Todoist : https://developer.todoist.com/guides/#authorization et Account linking: https://developers.google.com/assistant/identity/oauth2?hl=en

(5) https://developers.google.com/assistant/conversation-design/write-sample-dialogs et pour aller plus loin : https://researcher.watson.ibm.com/researcher/view_group.php?id=8426

(6) https://dialogflow.cloud.google.com/

2 Créer le *conversation flow* (scènes)

On définit ensuite les scènes qui utilisent les intentions : START (créé par défaut), START_AccountLinking (créé par l'account linking), LIST et ADD.

Une scène représente une étape du dialogue. Techniquement parlant, c'est l'équivalent d'un automate qui possède cinq étapes bien définies et reboucle sur celles-ci quand l'input utilisateur change. Chaque étape peut avoir une transition vers une autre scène ce qui interrompt la scène courante. Voici une description succincte de chacune des étapes :

- *On enter* : se produit uniquement la première fois qu'on rentre dans la scène
- *Condition* : toute vérification validée fait ce qui est demandé ou passe au :
- *Slot Filling*: boucle qui va demander à l'utilisateur des informations supplémentaires, si elles sont remplies, passe au :
- *User Intent Handling*: toute vérification d'intention validée fait ce qui est demandé ou passe au :
- *Error and status handling* : toute vérification validée fait ce qui est demandé ou passe aux intentions système et arrête la conversation.

START a un *On enter* qui donne la première phrase [1] et un *User Intent Handling* de l'intention LIST_TASKS vers la scène LIST, ADD_TASK vers ADD. C'est l'orchestrateur global de toutes nos scènes. **Figure 3**

LIST a un *On enter* qui appellera le webhook (handler PROMPT_LIST_SCENE) pour lister les tâches. Un webhook ou le *Slot Filling* peuvent enregistrer des informations dans la session de dialogue. Dans notre cas, on va enregistrer le nombre de tâches trouvées et vérifier la valeur dans *Condition*, pour avoir deux réponses différentes. On demandera si l'utilisateur veut créer une nouvelle tâche à ce moment-là avec [2]. Enfin, le *User Intent Handling* de l'intention AGREE vers ADD, et DISAGREE finit la conversation. Si aucun des deux n'est validé, le NO_MATCH retournera à la scène START pour recommencer le dialogue de zéro. **Figure 4**

ADD a un *Slot Filling* pour récupérer la tâche qu'on veut créer, qui correspond au même label que le datatype de l'intention ADD_TASKS [3] et une *Condition* pour vérifier que cette tâche a bien été créée dans la session utilisateur pour répondre [4]. Enfin, le *User Intent Handling* sur AGREE appelle le webhook (ADD_TASK_AGREE) pour créer la tâche, répondre [5] et finir la conversation [6]. DISAGREE modifie la tâche en appelant le webhook (DISAGREE_DELETE_TASK_ SESSION_PARAM) et reprend la scène ADD. Le *Status Handling* final est le même que celui de la scène LIST, sauf dans le NO_INPUT où il appellera le webhook pour créer la tâche (ADD_TASK_NO_INPUT) avant de clôturer la conversation. **Figure 5**

3 Un peu de code avec Google Cloud Inline Functions

Voici les handlers en nodejs du webhook, liste et création de tâche. Il ne faut pas oublier d'importer *"todoist-rest-api"*:

Figure 3

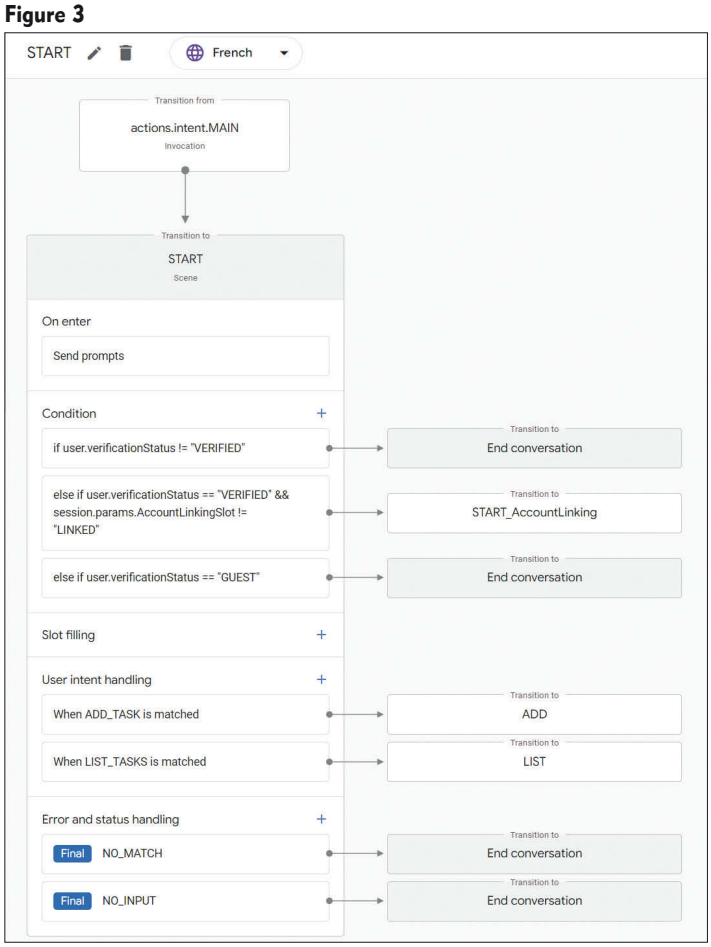
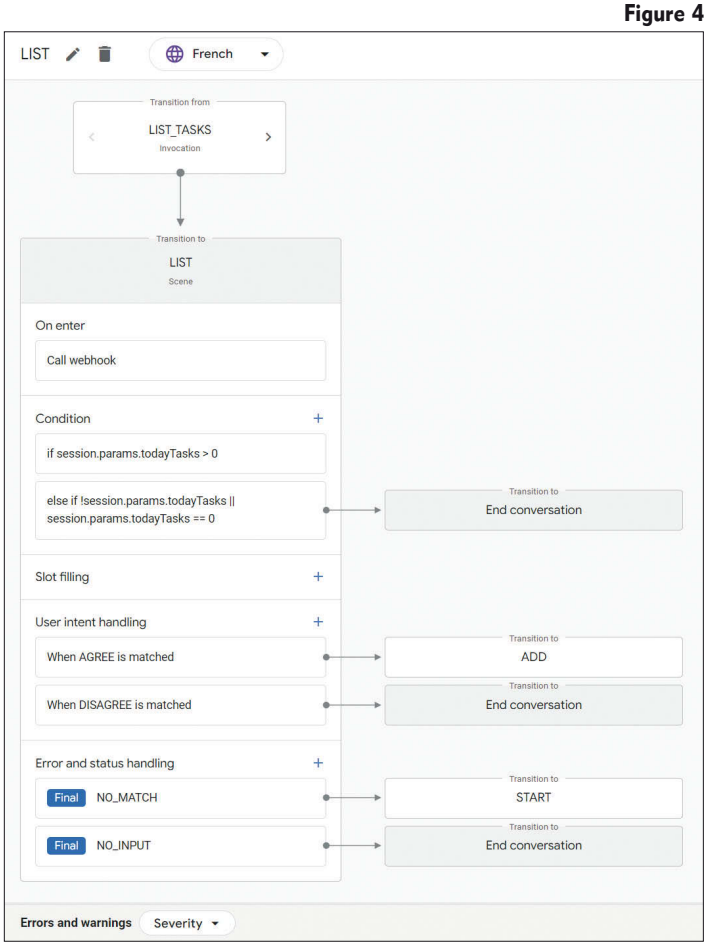


Figure 4



"^1.3.4"(7) dans le *package.json* et à bien mettre les mêmes noms de handlers que ceux appelés dans les scènes. Il y a deux types de formats pour la liste de tâches, celle qui sera dite par la voix, et l'autre affichée sur un terminal adapté (le téléphone ou une Google Nest par exemple).

```
const { conversation, Simple } = require('@assistant/conversation');
const functions = require('firebase-functions');
const todoist = require('todoist-rest-api').default;

const app = conversation();

app.handle('PROMPT_LIST_SCENE', async conv => {
 functions.logger.info("PROMPT_LIST_SCENE");
 const api = todoist(conv.user.params.bearerToken);
 const tasks = await api.v1.task.findAll({ filter: "aujourd'hui"});
 conv.session.params.todayTasks = tasks.length;

 if(tasks.length == 0) {
  return conv.add("Aujourd'hui, vous n'avez pas encore de tâches.");
 }
 else {
  const voiceTasks = tasks.map(({ content }) => { return content; });
  const textTasks = tasks.map(({ content }) => { return `  ${content}`; });
  return conv.add(new Simple({
   speech: `Vos tâches d'aujourd'hui sont: ${voiceTasks.join(', ')}.`,
   text: `Vos tâches d'aujourd'hui sont\n ${textTasks.join('\n')}\n`,
  }));
```

(7) https://www.npmjs.com/package/todoist-rest-api

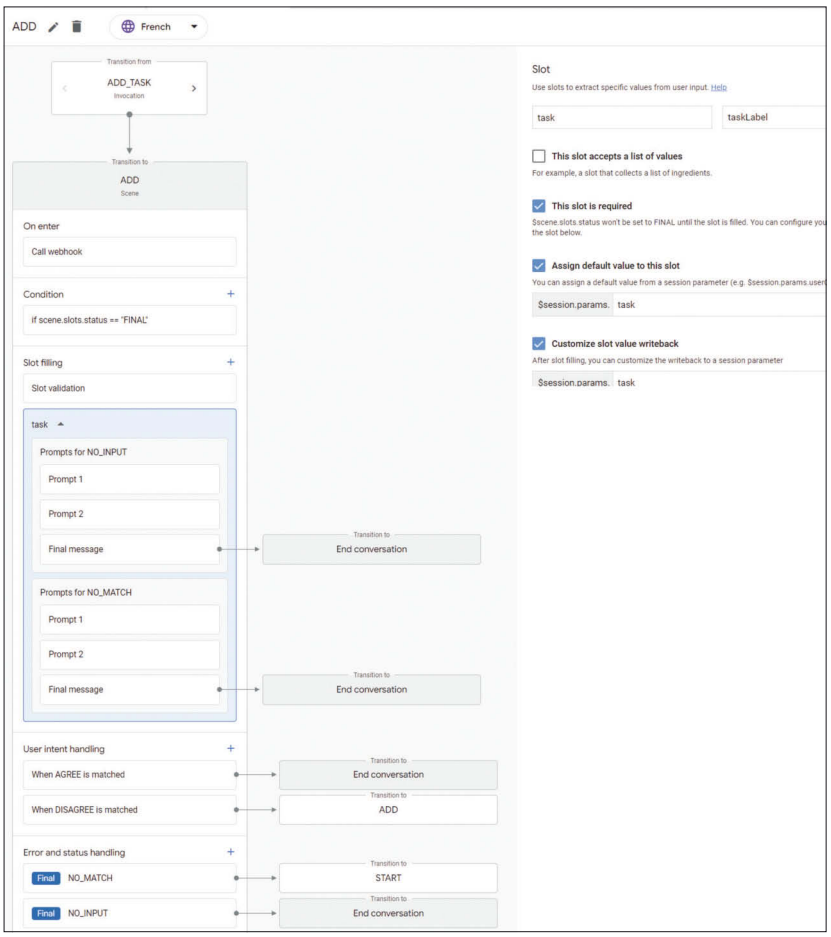
```
 }
});

app.handle(['ADD_TASK_AGREE', 'ADD_TASK_NO_INPUT'], async conv => {
 const label = conv.session.params.task;
 const api = todoist(conv.user.params.bearerToken);
 try {
  const projects = await api.v1.project.findAll({});
  const inbox = projects.filter((project) => project.inbox_project == true)[0];
  const createdTask = await api.v1.task.create({
   project_id: inbox.id,
   content: label
  });
  return conv.add('Bien, la tâche a été créee. Au revoir.');
 }
 catch (error) {
  return conv.add('Désolé, la tâche n\'a pas pu être créée. Au revoir.');
 }
});

app.handle(['DISAGREE_DELETE_TASK_SESSION_PARAM'], async conv => {
 const intentTask = conv.intent.params.task;
 if(intentTask) {
  conv.session.params.task = intentTask.resolved;
 }
 else conv.session.params.task = null;
 return conv;
});

exports.ActionsOnGoogleFulfillment = functions.https.onRequest(app);
```

Figure 5



4 Tester avec Google Assistant

L'onglet test permet de tester son flow conversationnel à n'importe quelle étape de la création du bot. Si le compte utilisé pour créer le bot est le même que celui d'un device android, l'assistant Google peut lancer le test sur le device directement en disant "parler à programme Todoist". Enfin, les deux intentions ADD_TASK et LIST_TASK peuvent être définies en Global Intent ce qui permettra d'invoquer directement la scène ADD et LIST, avec une phrase comme "demande à Programme Todoist de créer une tâche"

Conclusion

Les solutions apportées aujourd'hui tendent de plus en plus à faire du low code, voire du no code. Cela permet de mettre en place des prototypes rapidement. L'interface de Google Actions permet de créer rapidement et facilement des interfaces conversationnelles qui répondent efficacement à toutes les possibilités d'interaction, dès aujourd'hui. Et pour aller plus loin en mettant les mains dans le code, il est possible d'utiliser Google SDK Builder pour utiliser ou coder ses scènes et ses handlers dans la même base de code.

Comprendre et utiliser Serverless pour optimiser votre infrastructure

Vous souvenez-vous de l'époque où nous créions et déployions nos applications Web sur un serveur ? Nous étions responsables de la gestion des ressources, ce qui causait des problèmes. Par exemple, les serveurs étaient facturés (même quand ils n'étaient pas en cours d'exécution ou ne répondaient à aucune requête). De même, l'entretien des serveurs et de leurs dépendances prenait un temps considérable, tout comme les mises à jour de sécurité nécessaires… sans oublier les ressources à ajuster pour faire face au trafic imprévu.

Toutes ces lignes de configuration à écrire pour construire et maintenir une infrastructure nous distrayaient de l'application en elle-même. Les petites équipes d'ingénieurs étaient obligées de s'écarter de leur mission principale pour absorber ces tâches. Les équipes plus importantes, quant à elles, devaient les répartir entre différents postes, ralentissant les processus de développement et de livraison du logiciel ou de l'application.

Alors que les développeurs cherchaient une solution à ces problèmes, serverless est arrivé.

Qu'est-ce que le « Serverless » ?

Serverless est un modèle de développement « natif » qui permet aux développeurs de créer et d'exécuter leurs applications sans avoir à s'occuper de la gestion des serveurs. Le fournisseur de services cloud exécute un morceau de code en allouant les ressources de manière dynamique.
Serverless offre une couche d'abstraction de l'infrastructure pour faciliter l'exécution d'un programme : vous envoyez votre code sous la forme d'une fonction à votre fournisseur cloud et il s'occupe du reste.

Le modèle serverless se distingue des autres modèles d'architecture portés par des produits clouds puisque dans ce cas, votre fournisseur cloud est également responsable de la mise à l'échelle de l'application (ou « scaling » - la faculté d'un système à pouvoir changer de taille ou de volume selon les besoins des utilisateurs), en plus de la gestion de l'infrastructure en elle-même.

Vous pouvez par exemple configurer vos conteneurs pour qu'ils démarrent automatiquement lors d'un pic de charge. Le lancement de l'application ne se fera alors qu'en cas de besoin. En plus des avantages en termes de coût et d'efficacité, cela permet également aux développeurs de se libérer des tâches routinières associées à la mise à l'échelle des applications.

Cependant, les économies promises par les modèles serverless dépendent de votre cas d'utilisation. Serverless est recommandé pour les tâches asynchrones, qui peuvent être parallélisées ou avoir des charges imprévisibles. Cela rend le Serverless très attractif pour les nouveaux services lancés, en particulier pour les petites équipes qui n'ont pas les moyens ou l'envie de gérer leur propre infrastructure. L'investissement financier pour mettre Serverless en œuvre devient un gain - tant financier que de temps. Ce modèle est moins pertinent pour les entreprises plus expérimentées qui peuvent se permettre d'investir dans leur propre système (en utilisant Kubernetes par exemple).
Il est porté par deux produits principaux : les fonctions et les conteneurs.

Serverless Functions

Serverless Functions est un service permettant de créer simplement des applications Web sans avoir à gérer de serveurs. Le code est exécuté à la demande et est invoqué par une requête HTTP sur l'URL fournie par Scaleway. Ce code peut correspondre à une application entière, mais l'utilisation la plus courante de Serverless Functions consiste à intégrer des unités fonctionnelles (traitements simples) de l'application. Cette décomposition vous permet d'utiliser la même fonction dans plusieurs applications et ainsi éviter de dupliquer le code associé.

Une application front-end a besoin de tourner en permanence, il n'est donc pas intéressant d'utiliser Serverless Functions dans ce cas - sauf pour des sites dont la charge est presque nulle. En revanche, le service de souscription à une newsletter par exemple n'est utilisé qu'occasionnellement et serait un bon candidat pour Serverless Functions.
Il peut également être efficace pour intégrer les services tiers.

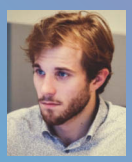
Serverless Containers

Serverless Containers est un service permettant de déployer rapidement des applications Web dans le cloud sans avoir à gérer l'infrastructure sous-jacente. À la différence de Serverless Functions qui est limité par l'environnement de développement proposé, vous pouvez construire votre application en utilisant le langage de programmation et les librairies que vous souhaitez. Scaleway s'occupera toujours de la disponibilité et la mise à l'échelle de ce conteneur. Les équipes de développement pourront ainsi se concentrer sur la création de l'application. C'est un bon point de départ pour déployer dans le cloud une application basée sur des micro-services conteneurisés. **Figure 1**

Hana Khelifa

Content Manager à Scaleway

Hana travaille depuis 10 ans dans l'informatique, d'abord développeuse après des études à 42, elle s'occupe aujourd'hui de créer le contenu nécessaire à la compréhension du cloud chez Scaleway.



Lucas MERDINIAN

Product Manager Serverless à Scaleway

Lucas est Product Manager en charge des produits Serverless Functions, Serverless Containers et Messaging qui permettent aux développeurs de déployer et de gérer simplement des applications à base de micro service dans le cloud.

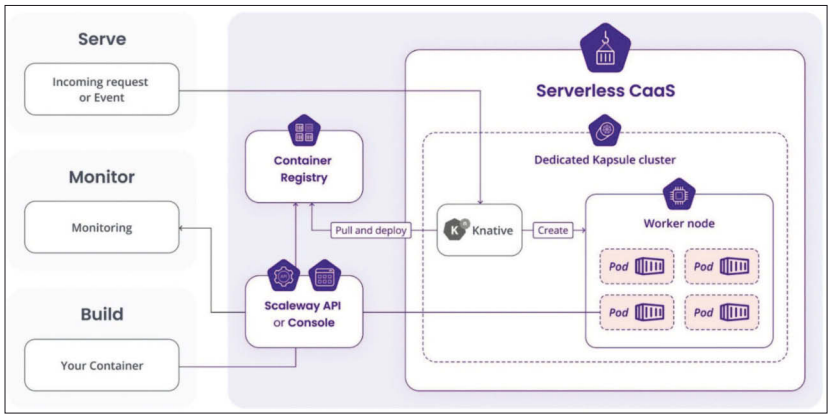


Figure 1 - *Schéma Serverless Containers*

Figure 2

Le serverless de Scaleway est construit à partir de notre solution de Kubernetes managé, Kapsule, à laquelle nous avons ajouté des ressources basées sur Knative dédiées à la gestion des services serverless.

Cas pratique : déployez votre premier « Hello World » sur Serverless

Prérequis : posséder un compte Scaleway

Le plus simple pour déployer une fonction serverless est d'utiliser la console Scaleway. Dans ce premier exemple, nous allons déployer un simple Hello World. L'avantage de Serverless Functions ici est qu'il n'est pas nécessaire de configurer un serveur Web pour générer un service Web accessible via une URL sécurisée.

- Connectez-vous à la console https://console.scaleway.com/
- Créez un namespace. Les namespaces sont des ensembles de fonctions qui partagent des informations communes (variables d'environnement, secrets, token d'accès) ;
  - Entrez un nom et une description ;
  - Il est possible d'ajouter des variables d'environnement ou des secrets, mais cela n'est pas utile pour le moment (ces éléments peuvent être modifiés une fois le namespace créé). **Figure 2**

Une fois le namespace créé, il est possible de créer une fonction :

1. Sélectionnez le Runtime (langage de programmation et version) parmi Python, NodeJS et Go ;
2. le gestionnaire (Handler) est la fonction de votre code qui va être exécutée à l'appel de votre Serverless Functions pour cet exemple vous pouvez laisser 'handler.handle' ; l'éditeur de code en ligne permet d'écrire des fonctions simples utilisant les librairies de base offertes par les runtimes (la section aller plus loin de cet article vous donnera un exemple d'intégration de librairies tierces) ;
3. l'éditeur de code en ligne propose un exemple de « Hello World » qui log le paramètre 'event' qui contient toutes les informations fournies par la requête HTTP (headers, mot HTTP, corps du message, etc.) ;
4. définissez le nom de votre fonction (par exemple 'test') ;
5. choisissez les ressources à allouer à votre fonction (pour cet exemple, le plus petit tiers '128Mo - 70mCPU' suffit) ;
6. Serverless Functions se met automatiquement à l'échelle afin de répondre dans les meilleurs délais aux requêtes entrantes. Le paramètre 'Mise à l'échelle' permet de définir :
   - le nombre minimum d'Instances de fonctions qui seront en permanence prêtes à répondre à une demande (attention si ce paramètre est supérieur à 0 une facturation sera appliquée, ce paramètre est à modifier uniquement si votre application nécessite des temps de réponse très faibles) ;
   - le nombre maximum d'Instances de fonctions qui seront déployées pour traiter les requêtes.
7. il n'est pas nécessaire pour cet exemple d'ajouter des variables d'environnement ;
8. pour simplifier les tests, laissez votre fonction en mode 'public'. Si vous souhaitez restreindre l'accès, définissez-la comme 'privée' : les appels non authentifiés seront rejetés par le système. L'authentification se base sur un système de JWT généré via notre API ;
9. les options avancées vous permettront de configurer le délai d'expiration de vos requêtes (au-delà de ce délai, le système termine l'exécution de votre fonction et renvoie une erreur) ;
10. les *Triggers* permettent de déclencher votre fonction en utilisant des événements. Pour le moment, seuls les CRON (déclenchement temporel) sont disponibles, mais nous prévoyons d'ajouter d'autres sources d'événements (Message Queue, Object Storage, etc.) ;
11. vous pouvez enfin évaluer le coût de votre fonction en vous basant sur le nombre de requêtes qui seront reçues par votre fonction ;
12. cliquez sur 'Créer une fonction' pour lancer le déploiement de votre fonction.
13. L'onglet 'Paramètres des Functions' vous permet de vérifier le statut de votre fonction, de connaître ou supprimer son URL, ou encore d'ajouter une URL personnalisée via l'onglet Endpoints (via un enregistrement CNAME chez votre hébergeur de nom de domaine) ;
14. une fois la fonction déployée (voyant vert), entrez l'URL de la fonction dans la barre d'adresse de votre navigateur. Le résultat suivant devrait s'afficher {"message":"Hello, world"} ;
15. dans l'onglet 'Consulter les journaux' vous pourrez observer les logs de la fonction. Ici les informations de la requête HTTP sont affichées.

Pour aller plus loin

Déployer un « Hello World » depuis la console est simple, mais cela ne correspond pas vraiment à une utilisation réelle de Serverless Functions.

Au travers d'un cas d'usage simple (transformation de fichiers stockés dans des bucket Objects Storage), nous allons voir comment déployer simplement des fonctions en utilisant Serverless Framework

Pré requis :

- un compte Scaleway ;
- des clés API Scaleway (à générer depuis la console) ;
- un bucket source ;
- un bucket destination ;
- NodeJS et npm ;
- Serverless.com CLI (npm install serverless -g). **Figure 3**

Ce projet contient deux fonctions :

- la première se connecte au bucket source, récupère les URL de tous les fichiers images, puis appelle la seconde fonction pour redimensionner chaque image ;
- la seconde récupère une image spécifique (dont le nom est passé par un appel HTTP), la redimensionne et pousse l'image redimensionnée vers un nouveau bucket.

Configurer l'environnement projet

Le framework de Serverless.com est utilisé par de nombreux développeurs pour déployer leurs architectures serverless. Les équipes Scaleway ont construit un plug-in vous permettant d'utiliser ce service pour déployer vos fonctions en toute simplicité.

- Pour créer un nouveau projet, entrez la commande suivante dans votre invite de commande :

```
serverless create —template-url
https://github.com/scaleway/serverless-scaleway-
functions/tree/master/examples/nodejs —path ImageTransformation
```

Un dossier avec l'architecture suivante devrait être créé :

```
- ImageTransformation
— handler.js
— package.json
— serverless.yml
```

- supprimez le fichier handler.js (c'est un « hello World ») ;
- installez les dépendances nécessaires à l'utilisation du plugin Scaleway.

```
npm i
```

Nous allons maintenant créer les deux fonctions (dont vous pouvez retrouver le code sur le GitHub de Programmez!) (Lien temporaire : https://drive.google.com/file/d/1-kw377EZp6XypJ2bhlg3UtMuQsaJAqM3/view?usp=sharing)

Créer la fonction bucketscan.js

1 Les variables d'environnement suivantes seront utilisées :

- SOURCE_BUCKET: clé du bucket source ;
- S3_ENDPOINT_URL: URL Object Storage de Scaleway (http://s3.{region}.scw.cloud) ;
- ACCESS_KEY_ID: l'identifiant de votre clé API Scaleway ;
- ACCESS_KEY: votre clé API Scaleway ;
- TRANSFORM_URL: l'URL de la fonction ImageTransform.

2 Installez le sdk AWS

```
npm i aws-sdk
```

3 Créez la Fonction Bucket Scan :

- Connectez-vous à Object Storage ;

```
// Create S3 service object
const s3 = new AWS.S3({
endpoint: S3_ENDPOINT_URL,
credentials: {
  accessKeyId: ID,
  secretAccessKey: SECRET,
}
});
//Configure parameters for the listObjectsV2 method
const params = {
Bucket: SOURCE_BUCKET
};
```

- récupérez tous les fichiers du bucket source ;

```
s3.listObjectsV2(params, function (err, data) {
if (err) {
 console.log(err, err.stack); // an error occurred
} else {
 let counter = 0;
 const contents = data.Contents;
 const total = contents.length;
```

- connectez-vous à Object Storage ;

```
// Create S3 service object
const s3 = new AWS.S3({
endpoint: S3_ENDPOINT_URL,
```

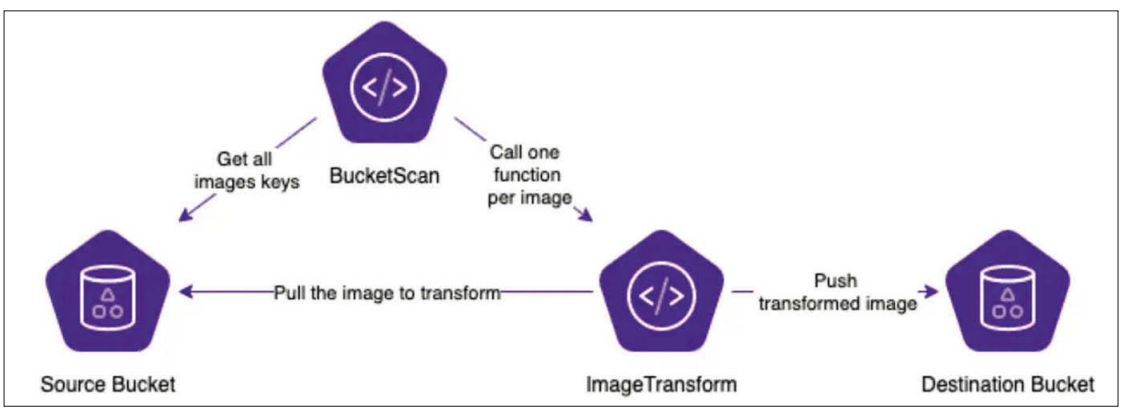


Figure 3 - *Schéma de principe de l'application développée*

```
credentials: {
  accessKeyId: ID,
  secretAccessKey: SECRET,
}
});
//Configure parameters for the listObjectsV2 method
const params = {
Bucket: SOURCE_BUCKET
};
```

- appelez de manière asynchrone la fonction ImageTransform avec en paramètre la clé des **Images** contenues dans le Bucket Source.

```
contents.forEach(function (content) {
 // Infer the image type from the file suffix.
 srcKey = content.Key;
 const typeMatch = srcKey.match(/\.([^.]*)$/);
 if (!typeMatch) {
  console.log("Could not determine the image type.");
 } else {
  const imageType = typeMatch[1].toLowerCase();
  // Check that the image type is supported
  if (["jpeg", "jpg", "png"].includes(imageType) !== true) {
   return console.log(`Unsupported image type: ${imageType}`);
  } else {
   try {
    console.log(TRANSFORM_URL + "?key=" + srcKey);
    https.get(TRANSFORM_URL + "?key=" + srcKey);
    counter += 1;
   } catch (error) {
    console.log(error);
   }
  };
 };
});
```

Construire la fonction imagetransform.js

1 Installez sharp en utilisant spécifiant linuxmusl comme environnement d'exécution afin d'assurer la compatibilité avec le Runtime Scaleway (basé sur Alpine Linux)

```
npm install —platform=linuxmusl —arch=x64 sharp —ignore-script=false —save
```

2 Les variables d'environnement suivantes seront utilisées

- SOURCE_BUCKET: clé du Bucket source ;
- S3_ENDPOINT_URL: URL Object Storage de Scaleway (**http://s3.{region}.scw.cloud**) ;
- ACCESS_KEY_ID: l'identifiant de votre clé API Scaleway ;
- ACCESS_KEY: votre clé API Scaleway ;
- DESTINATION_BUCKET: clé du bucket de destination des images transformées ;
- RESIZED_WIDTH: la taille d'image cible.

3 Écrivez ensuite la fonction ImageTransformation :

- Connectez-vous à Object Storage ;

```
const s3 = new AWS.S3({
endpoint: S3_ENDPOINT_URL,
credentials: {
  accessKeyId: ID,
  secretAccessKey: SECRET,
}
});
```

- récupérez la clé de l'image information grâce aux paramètres de requête de l'objet event :

```
exports.handle = async (event, context, callback) => {
// Read options from the event parameter.
console.log("Reading options from event");
// Object keys may have spaces or unicode non-ASCII characters.
const srcKey    = event.queryStringParameters.key;
```

- récupérez l'image à partir du Bucket source :

```
try {
const params = {
  Bucket: srcBucket,
  Key: srcKey,
};
var origimage = await s3.getObject(params).promise();
} catch (error) {
console.log(error);
return;
}
```

- redimensionnez l'image :

```
try {
var buffer = await sharp(origimage.Body)
  .resize({ width, withoutEnlargement: true })
  .toBuffer();
} catch (error) {
console.log(error);
return;
}
```

- téléchargez l'image sur le Bucket de Destination :

```
try {
  const destparams = {
    Bucket: dstBucket,
    Key: dstKey,
    Body: buffer,
    ContentType: "image"
  };
  const putResult = await s3.putObject(destparams).promise();
} catch (error) {
  console.log(error);
  return;
}
```

Créer vos Serverless Functions Scaleway à l'aide de Serverless Framework

Éditer le fichier serverless.yml avec les informations nécessaires à l'exécution des fonctions :

- les variables d'environnements à utiliser (ici sous forme de secrets) ;
- la clé API de votre projet Scaleway ;
- l'identifiant de votre projet Scaleway.

```
service: imagetransformation
configValidationMode: off

provider:
name: scaleway
runtime: node14
# Global Secret Environment variables - used in every function, you can
use local environment variable to prevent writing the
secret:
 ACCESS_KEY: {Scaleway API key related to source and destination buckets}
 ACCESS_KEY_ID: {Scaleway API key ID}
 SOURCE_BUCKET: {source bucket key}
 DESTINATION_BUCKET: {destination bucket key}
 S3_ENDPOINT_URL: http://s3.fr-par.scw.cloud
 TRANSFORM_URL: https://{your function}.functions.fnc.fr-par.scw.cloud
# the path to the credentials file needs to be absolute
scwToken: {Scaleway API token related to the Project ID}
scwProject: {Scaleway Project ID}

plugins:
- serverless-scaleway-functions

patterns:
 - '!.gitignore'
 - '!.git/**'
```

- Décrivez ensuite les fonctions que vous souhaitez importer en précisant les ressources (mémoire) à attribuer :
  - imagetransform.handle permet de créer une fonction qui appellera la fonction handle() du fichier imagetransform.js ;
  - bucketscan.handle permet de créer une fonction qui appellera la fonction handle() du fichier bucketscan.js ;
  - On ajoute aussi la description d'un CRONJob s'exécutant toutes les heures pour récupérer les images.

```
functions:
 imagetransform:
  handler: imagetransform.handle
  memoryLimit: 1024  env:    RESIZED_WIDTH: '300'
 bucketscan:
  handler: bucketscan.handle
  memoryLimit: 1024
  minScale: 0
  events:
   - schedule:
     rate: '1 * * * *'
```

Pour lancer la création des fonctions, exécutez la commande :

```
serverless deploy
```

L'outil réalise toutes les opérations (zip des fonctions et de leurs dépendances, appels aux API Scaleways) pour créer les namespace et les fonctions.
Une fois la commande terminée, les URL des fonctions seront affichées. Vous pourrez alors copier l'URL de la fonction "imagetransform" dans le secret TRANSFORM_URL:.

- Relancez ensuite le chargement des fonctions avec serverless deploy.
- Pour tester le fonctionnement de l'application vous pouvez à tout moment appeler la fonction bucketscan via son URL et vérifier vos logs dans la Console Scaleway ou via la commande :

```
serverless logs —function bucketscan
```

À votre tour d'essayer

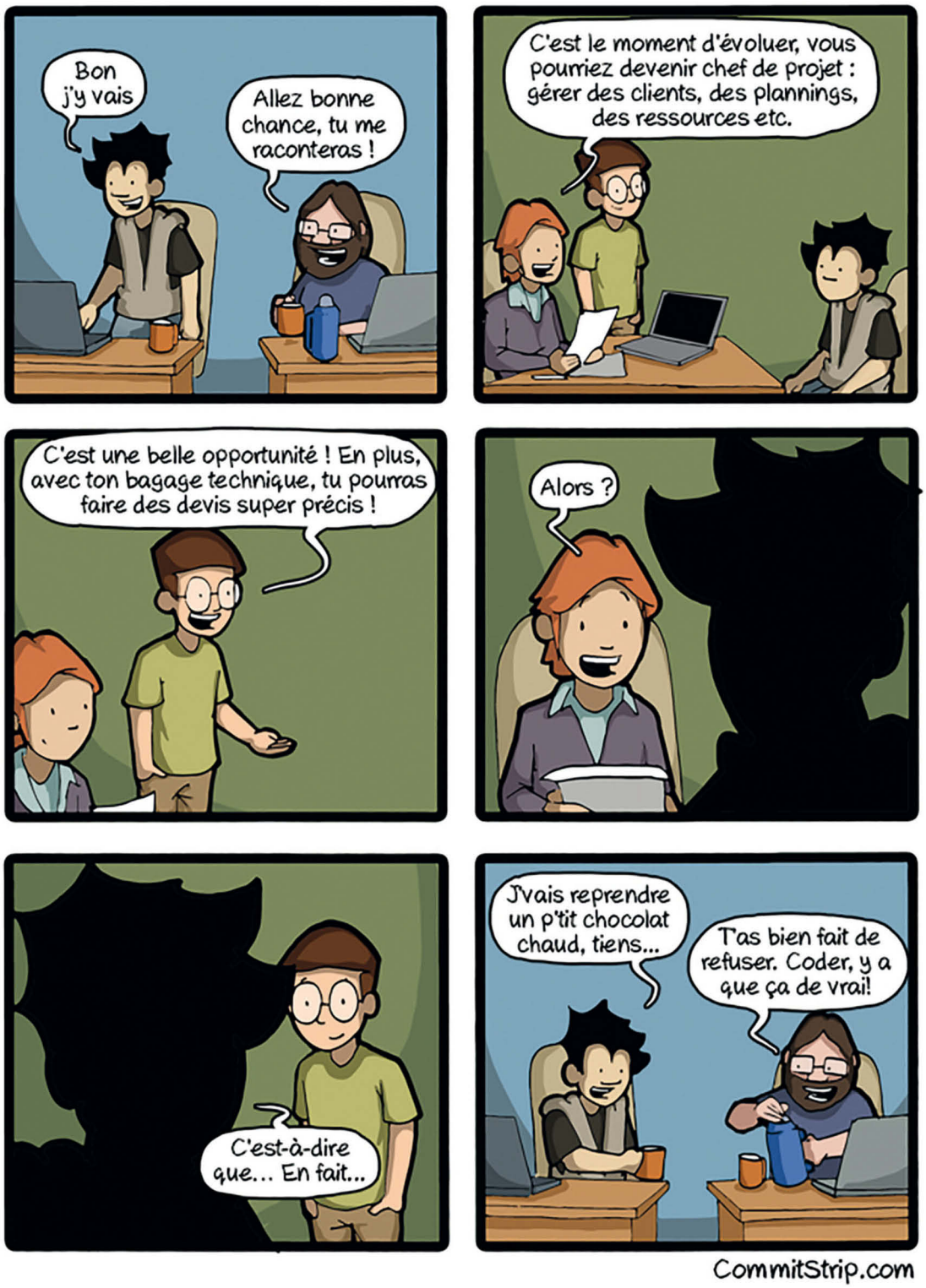
Vous avez maintenant toutes les connaissances et les tutoriels nécessaires pour faire vos premiers pas dans l'écosystème serverless. Et pour vous accompagner dans votre aventure, **Scaleway vous offre 100 euros de crédits** qui vous permettront notamment de mettre en place votre premier conteneur Serverless.
Pour les utiliser, créez votre compte puis rentrez le code **"PROGRAMMEZ"**. Si vous avez des questions sur Serverless, n'hésitez pas à rejoindre notre Slack ouvert à notre communauté à cette adresse: **slack.scaleway.com**

À PROPOS DE SCALEWAY

Scaleway, premier fournisseur multi-cloud alternatif pour start-ups et développeurs répond aux besoins du marché mondial avec une offre de ressources de calcul qui est flexible, fiable, sécurisée, durable et au juste prix. Scaleway est l'un des rares acteurs en France et en Europe à posséder une maîtrise d'ouvrage et sans dépendances à trois niveaux : conception et opération de datacenters, infrastructure matérielle / infrastructure logicielle, IaaS et PaaS. Scaleway s'appuie sur sept datacenters situés à Paris (en France), à Amsterdam (aux Pays-Bas) et à Varsovie (en Pologne).

Actuellement en pleine expansion, Scaleway va recruter quelques 300 profils en 2022, dont de nombreux développeurs.
Pour en savoir plus : https://careers.scaleway.com

Dev un jour, dev toujours !



PROGRAMMEZ!
Programmez! hors-série n°7
PRINTEMPS 2022
Directeur de la publication
& rédacteur en chef
François Tonic
ftonic@programmez.com

Code review :
Dorra Bartaguiz

Contacter la rédaction
redaction@programmez.com

Les contributeurs techniques
Jean-Pierre Riehl, Gérard Gouzel, Alain Faure, Sylvain Fagnent, Simon Campano, Frédéric Fadel, Marc Gueury, Mike Kiersey, Pierre Oudot, Florian Chazal, Sébastien Stormacq, Thibault Marty, Emilie Sanchez, Joël Crest, Clément Sannier, Franck Dubois, Philippe Charrière, Sacha Bailleul, Patrick Prémartin, Tiffany Souterre, Anthony Viard, Aurore de Amaral, Hana Khelifa, Lucas Merdinian, Commitstrip

Couverture : © **bawanch**

Maquette : **Pierre Sandré**

Marketing – promotion des ventes
Agence BOCONSEIL
Analyse Media Etude
Directeur : Otto BORSCHA
oborscha@boconseilame.fr
Responsable titre : Terry MATTARD
Téléphone : 09 67 32 09 34

Publicité
Nefer-IT - Tél. : 09 86 73 61 08
ftonic@programmez.com

Impression : **SIB Imprimerie, France**
Dépôt légal : **A parution**

Commission paritaire
1225K78366

ISSN : **2279-5001**

Abonnement
Abonnement (tarifs France) : 55 € pour 1 an, 90 € pour 2 ans. Etudiants : 45 €. Europe et Suisse : 65 € - Algérie, Maroc, Tunisie : 64 € - Canada : 80 € - Tom - Dom : voir www.programmez.com. Autres pays : consultez les tarifs sur www.programmez.com.

Pour toute question sur l'abonnement :
abonnements@programmez.com

Abonnement PDF
monde entier : 45 € pour 1 an.
Accès aux archives : 20 €.

Nefer-IT
57 rue de Gisors, 95300 Pontoise France
redaction@programmez.com
Tél. : 09 86 73 61 08

softfluent

Rejoignez l'ESN créée par des développeurs pour les Développeurs

Vous possédez une ou plusieurs de ces compétences ?

- Azure DevOps
- Cloud
- SQL Server
- Angular
- React
- C#
- CI/CD
- .NET Core
- ASP.NET MVC
- Microservices

Contactez nous !

Retrouvez-nous sur notre page carrière: softfluent.fr/nous-rejoindre

Ou contactez **Sylvie**, en charge du recrutement chez SoftFluent :

06 67 14 01 39

sylvie.benjelloun@softfluent.com

- Conseil
- Expertise
- Partage

softfluent.fr