programmez!
#198 - Juillet-Août 2016
le magazine des développeurs

Spécial été 2016

100 PAGES

robot | domotique | voiture | jeux | Raspberry

- Réalité virtuelle avec CardBoard et HoloLens
- Coder son jeu vidéo
- Arduboy : Arduino se transforme en console
- Découvrir les API Haven OnDemand
- Créer son langage de programmation
- Résumé de Devoxx France 2016
- La culture geek au musée

Hacker une Tesla

M 04319 - 198 - F: 6,50 € - RD

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

Devoxx France 2016 :
passion développeur 1ere partie

Que serait le printemps sans une grande conférence développeurs ? Devoxx a une nouvelle fois offert de très nombreuses sessions techniques sur de nombreux sujets. DevOps et Docker ont été deux sujets très présents, tout comme Java (ouf !) et JavaScript. C'est toujours une occasion de découvrir des technologies et des outils que l'on ne connaît pas vraiment, de savoir que telle fonction existe ou tout simplement de croiser les communautés, toujours nombreuses.
Impossible de parler de toutes les sessions, comme chaque année, nous avons dû faire un choix... Bonne lecture !

Modularisez votre JavaScript avec JSPM et SystemJs

En préparant ma venue au Devoxx 2016, une conférence a particulièrement retenu mon attention : « Modularisez votre JavaScript avec JSPM et SystemJs », présentée par Benoit LEMOINE (@benoit_lemoine) et Samuel VAILLANT (@samouss_28).
Ce talk m'a donné un aperçu des possibilités de JSPM et SystemJs.

Emilien MURATON
Développeur chez **PALO IT France**

Emilien développe depuis plus de 2 ans des applications en JavaScript et Java. Il utilise des frameworks tels que ReactJS, Webpack, Spring et Guava. Ses différentes expériences l'ont amené à travailler au sein d'équipes Agiles. Il a récemment développé une application Web s'appuyant sur le langage JavaScript natif dans un contexte international, ce qui lui a permis de renforcer ses connaissances en UX Design.

PALO IT
Innovation & Transformation

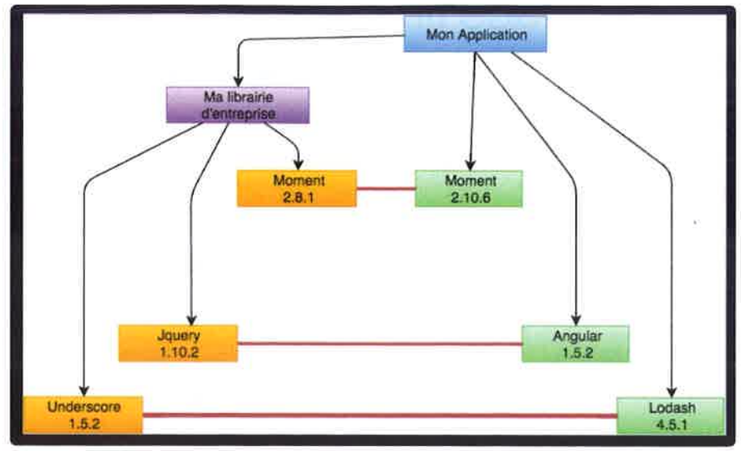
L'approche traditionnelle du JavaScript est d'utiliser beaucoup, beaucoup, mais alors beaucoup de balises scripts pour récupérer des librairies.

```
<script src="vendor/moment.js"></script>
<script src="vendor/lodash.js"></script>
<script src="vendor/query.js"></script>
<script src="src/app.js"></script>
```

LES PROBLEMES RECURRENTS

Cet enchaînement de balises pose de nombreux problèmes :

- Variables globales : prenons un exemple simple, si on importe lodash et underscore, ces 2 librairies définissent une variable globale avec le même nom (le caractère underscore). Mais comment savoir quelle librairie on utilise lorsque l'on se sert de la variable _ ? ;
- Conflits de versions ;
- Ordre d'exécution des scripts (attention ! un mauvais ordre d'appel des scripts casse l'application.) ;
- Dépendances transitives : quand deux librairies ont comme dépendances la même librairie mais avec deux versions différentes. Voici un exemple concret présenté lors de la conférence qui résume ces problématiques :



QUE FAIRE ?

Il faut utiliser un système de modules. Historiquement, on retrouve AMD (Asynchronous Module Definition), une spécification qui permet d'organiser le code en modules, afin de pouvoir les charger à la demande. Cela évite d'avoir un seul gros fichier ou plusieurs fichiers à inclure dans le HTML dans un ordre précis. Son implémentation la plus connue est RequireJS.
Une autre spécification de module est CommonJs, plus orientée serveur (contrairement à AMD qui s'intègre très bien dans les navigateurs) et notamment utilisée dans Node JS. CommonJS charge les modules de façon synchrone. Lors de ce talk, les speakers nous ont plutôt conseillé d'oublier ces deux systèmes de modules afin de privilégier les imports d'ECMAScript2015 (es6), un format créé pour réunir les utilisateurs de CommonJS et d'AMD. Malheureusement, les imports es6 ne sont pas encore implémentés nativement par les navigateurs. C'est pourquoi les speakers nous ont orienté vers SystemJS, un polyfill ou shim des imports es6, qui comprend aussi les modules CommonJS et AMD.

Exemple system JS

```
var System = require('systemjs');
System.import('lodash');
System.import('https://code.jquery.com/jquery.js');
```

Nous voyons clairement ici que SystemJS permet de répondre de manière élégante aux problématiques d'imports successifs couramment rencontrés sur les projets legacy, via un système de modules. Pour aller un peu plus loin, et avoir un workflow plus sympa, on peut utiliser JSPM. Mais qu'est-ce que JSPM ? JSPM (JavaScript Package Manager) est un gestionnaire de paquets qui utilise SystemJS pour gérer les packages et leurs dépendances. Il a pour objectif de devenir LE module manager front-end et supporte le HTTP/2. Pour une installation facile de votre projet, jspm fournit un registry (lien : http://kasperlewau.github.io/registry/#/) des librairies les plus communément utilisées.

COMMENT L'UTILISER ?

Importons jspm :

```
npm install jspm –g
```

Créons la configuration de notre projet.

```
jspm init
Package.json file does not exist, create it? [yes]:
Would you like jspm to prefix the jspm package.json properties under jspm? [yes]:
Enter server baseURL (public folder path) [.]:
Enter jspm packages folder [./jspm_packages]:
Enter config file path [./config.js]:
Configuration file config.js doesn't exist, create it? [yes]:
Enter client baseURL (public folder URL) [/]:
Which ES6 transpiler would you like to use, Traceur or Babel? [traceur]:
```

Cela va créer un fichier nommé config.js qui contient les chemins des dépendances et sera mis à jour lors de l'utilisation des commandes avec jspm. Comme avec npm, on peut choisir d'enregistrer les paquets en tant que devDependencies (uniquement pour l'environnement de dev).
Il ne nous reste plus qu'à éditer ce fichier ou utiliser la commande jspm install pour gérer les dépendances Javascript de notre application.

ATTENTION !

Par défaut, jspm transpile l'es6 en es5 au runtime avec Babel, ralentissant énormément l'application. Ainsi il est conseillé de créer un bundle avec le CLI de JSPM :

```
jspm bundle clock-component --watch
```

Voici une comparaison des temps d'exécution entre une exécution avec transpilation au runtime, et une exécution avec notre bundle. Sans surprise on voit une énorme différence de temps entre les deux exécutions.

CONCLUSION

La combinaison de JSPM et SystemJS procurent une façon unifiée d'installer et de charger simplement et rapidement les dépendances. Elle permet aussi d'utiliser le futur des systèmes de gestion de modules dès aujourd'hui. Enfin JSPM est capable d'installer des paquets non seulement de NPM, mais aussi de Bower, ou directement de GitHub.
Source : Présentation Devoxx France 2016 : http://blemoine.github.io/jspm-slides/.

La Blockchain dans tous ses états

J'ai choisi d'assister à l'intervention de François Galilee et de Heykel Jelassi intitulée « Blockchain as a Trust Machine ? Vers une économie programmable » mais également au quicky de Philippe Antoine intitulé « Développez votre première application décentralisée sur la blockchain avec Ethereum et Embark ». Comme vous l'aurez compris, la Blockchain était le sujet de prédilection cette année.

Robert DEGRET
Architecte chez PALO IT France

PALO IT Innovation & Transformation

Robert travaille depuis plus de 8 ans sur le développement d'applications requérant performance et haute disponibilité. Son expérience dans le domaine bancaire lui a permis d'appréhender des flux temps réel comportant des millions de données. Il a également développé en méthode Agile une application Web. Après avoir travaillé chez PALO IT Singapour pendant 4 ans, il est désormais basé au bureau parisien.

Blockchain

Bitcoin a apporté une solution concrète au « problème des généraux byzantins ». Ce problème classique de l'informatique distribuée dans un environnement non fiable peut s'expliquer par la métaphore suivante : dans le schéma, l'armée A fait le siège de l'armée B et C. B et C doivent attaquer ensemble pour gagner contre A. Pour cela, ils doivent échanger des messages : d'abord de B vers C pour proposer le moment et l'heure de l'attaque, mais C peut douter du message qu'il recevra de B (A en a eu connaissance, et peut-être l'a-t-il modifié, etc.). De même B attend le message de confirmation de C pour ne pas se retrouver à attaquer seul. Mais quand il reçoit ce message, il peut lui aussi douter de son origine et de son contenu.

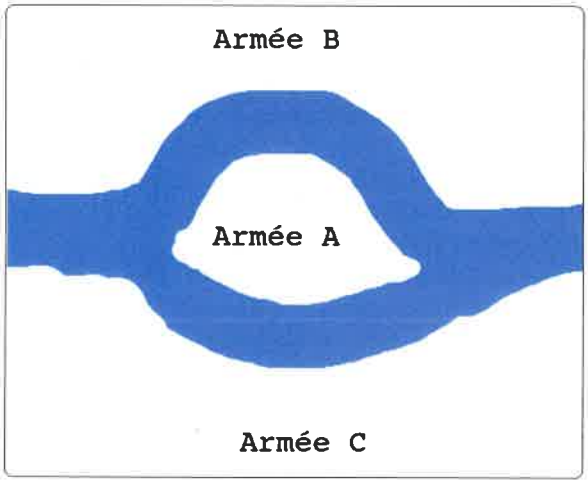
Satoshi Nakamoto [1] donne les principes de sa solution pour résoudre les problèmes du double paiement et des généraux byzantins :

> *« Nous proposons une solution au problème de double-dépense en utilisant un serveur horodaté distribué en pair-à-pair pour générer des preuves informatiques de l'ordre chronologique des transactions. Le système est sécurisé tant que des nœuds honnêtes contrôlent ensemble plus de puissance de calcul qu'un groupe de nœuds qui coopéreraient pour réaliser une attaque. »*

La blockchain représente simultanément un stockage d'information distribué en pair à pair et le mécanisme enrichit toutes les 10 minutes ce livre de compte ouvert à tous. Elle permet l'automatisation de la transaction en supprimant les tiers.

Contrats

Jusque-là nous n'avons parlé que de faits mais une blockchain peut aussi exécuter des programmes. Ils sont stockés avec les faits : chaque nœud l'exécute lorsqu'il reçoit le fait. Dans Bitcoin, c'est le mécanisme utilisé pour rendre certaines transactions conditionnelles. Par exemple, une transaction ne sera valide qu'à une certaine date.



D'autres blockchains sont d'ailleurs conçues autour de contrats bien plus élaborés. Chaque transaction Ethereum est accompagnée d'une mini-base de données avec des méthodes pour modifier les données. Comme les contrats sont diffusés sur tous les nœuds, les bases de données le sont également. A chaque modification des données par un utilisateur, le changement est répliqué et rejoué sur tout le réseau.

La notion de conditions préprogrammées, reliées au monde réel, diffusées à tous, constitue un "smart contract" [3]. Un contrat est une promesse qui lie 2 parties de manière légale. Un "smart contract" est la même chose, en remplaçant le mot légal par "technique". Il n'y a plus besoin de juge.

Effet de masse

Le mouvement est lancé et prend de l'ampleur. La communauté grandit et bouillonne d'idées. L'accès à la blockchain est de plus en plus simple, la blockchain en C++ de bitcoin a été portée dans de nombreux langages. Embark [2][4] permet par exemple de créer sa propre blockchain privée en moins de deux minutes. Des sites répertorient les dizaines d'applications "smart contract". Vitalik Buterin, le jeune créateur d'Ethereum, saute d'avion en avion pour donner des conférences à travers le monde. 40 grandes banques se sont ralliées autour de la startup R3 afin de créer une blockchain car elles sentent le danger.

Limites

Bitcoin a déjà rencontré quelques embûches : les fortes fluctuations de valeur, des délais dans les transactions, la fermeture de certaines plateformes d'échanges sous la menace des autorités monétaires dans le monde, etc. La blockchain sera testée lorsqu'il y aura besoin d'arbitrer des contrats réels. Une solution [5] proposée par Vitalik Buterin est d'avoir une cour décentralisée avec des "juges" payés par le réseau.

Liens :

[1] http://www.bitcoin.fr/pages/Bitcoin-expliqu%C3%A9-par-son-inventeur
[2] http://cfp.devoxx.fr/2016/talk/VZJ-1808/Developpez_votre_premiere_application_decentralisee_sur_la_blockchain_avec_Ethereum_et_Embark
[3] http://cfp.devoxx.fr/2016/talk/JGT-7450/Blockchain_as_a_Trust_Machine_%3F_Vers_une_economie_programmable_
[4] http://iurimatias.github.io/embark-framework/
[5] https://www.reddit.com/r/ethereum/comments/4gigyd/decentralized_court/?ref=readnext

précédent appel dès lors que cette fonction a été appelée avec les mêmes arguments. Cela engendre un gain de performances à l'aide d'une seule annotation @memoize sur la fonction.

Etape n°4 : Angular 2 dans Angular 1

Pour commencer, il faut installer Angular 2 à l'aide du package manager choisi. Pour que votre application fonctionne, il faut récupérer au même moment rxjs et reflect-metadata.

Pour utiliser Angular 2 dans une application Angular 1, peu importe l'ordre et la position des modules Angular 2 dans l'arbre définissant l'application hybride, celle-ci fonctionnera correctement tant que la racine restera un module Angular 1. Ensuite, pour ugrader des composants et services Angular 1 ou downgrader des composants et services Angular 2, nous devons utiliser *UpgradeAdapter.* Dans le premier cas, nous souhaitons intégrer une directive Angular 1 dans un composant Angular 2. Voici comment l'utiliser : Fig.3.

Dans le second cas, nous souhaitons utiliser un composant Angular 2 dans notre application Angular 1. Pour cela, la méthode de downgrade de l'UpgradeAdapter va permettre de créer une directive Angular 1 à partir du composant Angular 2. Fig.4. Une autre solution qui s'appelle Ng-forward consiste à écrire du code avec les conventions Angular 2 avec du Angular 1.3+. Cette solution peut être une première étape avant d'écrire du vrai code Angular 2.

En conclusion, pour réaliser une bonne migration d'une application Angular 1 vers Angular 2, il faut suivre les bonnes pratiques Angular, utiliser un système de modules avec Typescript, migrer les « feuilles » de votre arbre de composants en finissant par le routage et le composant le plus haut.

Domain Driven Design

J'ai pu découvrir cette année le Devoxx France. C'était une grande première pour moi, et je dois dire que je n'ai pas été déçu. Un planning qui ne connaît aucun temps mort et des conférences plus intéressantes les unes que les autres : je décrirais le Devoxx comme le « Disneyland des Développeurs » ! J'ai choisi de donner mon retour sur une conférence intitulée « DDD : et si on reprenait l'histoire par le bon bout ? » par Thomas Pierrain et Jérémie Grodziski.

Stéphane LOPES NEVES
Développeur chez **PALO IT France**
Au cours de ses études, Stéphane a travaillé en tant que Développeur au sein d'un cabinet de conseil, mais également chez différents leaders de marché comme Airbus Helicopters, où il a conçu, entre autres des applications de gestion en .NET et Java JEE. Il a rejoint PALO IT.

PALO IT
Innovation & Transformation

"Le DDD, mais qu'est-ce que cela peut être ?" Les paris sont lancés ! Que peuvent bien signifier ces fameux « D » ? « Driven Development » ? Framework, pattern, techno ?

Le gagnant de l'acronyme du jour n'est autre que le "Domain Driven Design". Il y avait bien "Driven", je n'étais pas si loin de la vérité ! Cachée derrière ces trois mots, une vraie philosophie de développement plus qu'un pattern.

L'objectif est de faire de l'objet en restant le plus proche possible du langage des utilisateurs. Concrètement, il m'arrive de faire face à ce genre de situation où l'on me demande : « Pourquoi certaines factures en attente ne sont pas valides ? Les spécifications fonctionnelles de cette demande sont-elles facilement accessibles? » Si c'était le cas on ne viendrait pas me poser la question. Résultat, je lance l'application que j'exécute pas à pas jusqu'à tomber sur la méthode suivante :

```
public Boolean validate(Invoice invoice) {
    return "14".equals(invoice.getStatus())
        && LocalDate.now().getDayOfMonth() <= 25
        && invoice.getPrice().compareTo(new BigDecimal(500)) < 0;
}
```

Je réponds en plissant les yeux dans un français que je pense correct : *"Pour être valide, une facture ne doit pas avoir le statut 14, être validée jusqu'au 25 et avoir un prix inférieur à 500."*

Si on me répond "Aaah, merci !", dans ce cas pas de problème ! Mais je peux être également confronté à ce type de questions "c'est quoi le statut 14 ?", "pourquoi le 25 ?" ou encore "pourquoi 500 ?". Toute la logique fonctionnelle est masquée par des nombres magiques qui n'ont pas forcément de sens pour le métier. Cela a pour conséquence de complexifier le dialogue entre développeurs et utilisateurs. Je reprends cette même méthode écrite de la manière suivante :

```
public Boolean validate(Invoice invoice) {
    return !InvoiceStatus.ABANDONNEE.equals(invoice.getStatus())
        && invoice.beforeLastDayOfMonthToValidate(invoice)
        && invoice.lowerThanTheMaximumPriceAllowed();
}
```

Simplement en lisant, je sais qu'une facture est valide si :

- Elle n'a pas le statut abandonnée ;
- La validation doit être faite avant un jour précis ;
- Qu'elle doit avoir un montant inférieur au prix autorisé.

Derrière ces différentes méthodes se cachent le même code. Je connais juste la signification de l'utilisation de données et je suis capable de fournir une information à la personne qui me pose la question. C'est simple à mettre en place, non ? Il existe un ouvrage que les adeptes appellent le "Blue Book" d'Eric J. Evans, considéré comme la référence pour comprendre le DDD. Sur le papier, ça n'a vraiment pas l'air d'être un livre auquel on s'attaque pendant les trajets de bus. C'est d'ailleurs un point qui a été souligné par les conférenciers. Ils nous ont également expliqué que le meilleur moyen de faire du DDD, c'est de ne pas dire que l'on en fait, car c'est souvent complexe et difficile d'accès. Afin de démocratiser cette pratique, ils optent plutôt pour une vulgarisation des différents concepts. Ils préconisent que l'on procède par des améliorations successives du code en fonction de ses connaissances du DDD, plutôt que d'essayer de tout assimiler et de tout remettre en question de manière radicale.

Conclusion

L'adoption de cette philosophie amène les développeurs à comprendre les besoins fonctionnels auxquels ils doivent répondre quotidiennement au travers des applications sur lesquelles ils travaillent. Cela a pour conséquence de faciliter leur communication avec les utilisateurs qui se sentent alors compris et seront plus enclins à partager des informations. C'est un exercice qui peut être fait sur le code dans lequel vous travaillez sans compromettre le reste de l'application. Bien sûr, le sujet est bien plus large que ce que j'ai présenté mais cette conférence m'a donné envie de creuser le sujet pour comprendre les différents concepts cachés derrière ces trois lettres.