

RAPPORT DE STAGE DEVELOPPEUR LOGICIEL

Création d'applications mobile avec ZetaPush sur
Ionic 3 et rédaction de tutoriels

Résumé

Réalisation d'applications bout en bout montrant l'utilisation du BaaS ZetaPush sur le
Framework Ionic 3.
Rédaction de tutoriel pour le site « Developpez.com »

Remerciements

Je tiens tout particulièrement à remercier mon maître de stage M. Mikaël Morvan pour son accueil chaleureux et pour m'avoir accepté en tant que stagiaire au sein de ZetaPush.

Je remercie également M. Grégory Houllier pour le soutien technique, M. Pablo Abreu pour la relecture et la correction des tutoriels en anglais et M. Jean-Charles Michel pour les détails commerciaux.

D'une façon plus générale, je remercie l'équipe de ZetaPush, pour l'intérêt qu'ils m'ont porté tout au long de mon stage ainsi que pour leur aide et précision.

Je tiens aussi à remercier l'ensemble des formateurs et personnel de l'administration de l'ENI Ecole Informatique pour ces 6 mois de formations.

Je remercie aussi M. Malick Seck Community manager du site « Developpez.com » pour son aide lors de la rédaction du tutoriel.

Sommaire

ABSTRACT	5
INTRODUCTION	5
LISTE DES COMPETENCES COUVERTES PAR LE STAGE	6
EXPRESSION DES BESOINS DE L'APPLICATION A DEVELOPPER.....	6
PRESENTATION DE L'ENTREPRISE	7
TECHNOLOGIES UTILISEES	8
<i>ZetaPush</i>	<i>8</i>
<i>ZetaPush Macro Script (ZMS)</i>	<i>8</i>
<i>Node.Js - npm</i>	<i>9</i>
<i>Angular 4</i>	<i>9</i>
<i>Ionic 3</i>	<i>9</i>
<i>TypeScript</i>	<i>10</i>
FONCTIONNEMENT DE ZETAPUSH.....	11
EXIGENCES ET CONTRAIRES	12
TRAVAIL EFFECTUE LORS DU STAGE	12
REALISATION D'UNE APPLICATION DE PARIS.....	13
<i>Spécifications fonctionnelles.....</i>	<i>13</i>
<i>Spécifications techniques.....</i>	<i>13</i>
<i>Trouver le thème d'une application.....</i>	<i>13</i>
<i>Création d'un nouveau projet.....</i>	<i>13</i>
<i>Mise en place des écrans</i>	<i>14</i>
<i>Design pattern MVC sur Ionic</i>	<i>15</i>
<i>Provider</i>	<i>15</i>
<i>Modèle</i>	<i>15</i>
<i>Page</i>	<i>16</i>
<i>Conclusion : design pattern avec Ionic</i>	<i>17</i>
<i>Composants Ionic.....</i>	<i>17</i>
<i>Création d'un projet ZMS sous Eclipse</i>	<i>18</i>
<i>Déploiement de services via ZetaPush.....</i>	<i>20</i>
<i>Utilisation d'un service et programmation en ZMS sur Eclipse.....</i>	<i>21</i>
<i>Configuration de l'application pour utiliser ZetaPush</i>	<i>22</i>
<i>Installation du SDK JS.....</i>	<i>22</i>
<i>Création de méthodes appelant les MacroScripts.....</i>	<i>22</i>
<i>Création d'un client de connexion.....</i>	<i>23</i>
<i>Utilisation de ZetaPush dans l'application.....</i>	<i>24</i>
<i>Conclusion.....</i>	<i>26</i>
REALISATION D'UNE APPLICATION MANGATHEQUE	26
<i>Spécifications fonctionnelles.....</i>	<i>26</i>
<i>Spécifications techniques.....</i>	<i>26</i>
<i>Diagramme d'utilisation</i>	<i>27</i>
<i>Mise en place de l'application sous Ionic 3.....</i>	<i>28</i>
<i>Programmation du service et des MacroScripts ZetaPush</i>	<i>28</i>
<i>Déclaration du service</i>	<i>28</i>

Julien Bertacco
RAPPORT DE STAGE DEVELOPPEUR LOGICIEL

Création des tables SQL.....	29
<i>ZetaPush Macro Scripts</i>	<i>30</i>
<i>Création de l'interface</i>	<i>33</i>
Affichage des séries	34
Two-Way Data Biding et formulaire	39
Formulaire d'ajout et d'édition de données	40
REDACTION D'UN TUTORIEL SUR LA MANGATHEQUE POUR DEVELOPPEZ.COM	46
<i>Rédaction au format Developpez.com.....</i>	<i>46</i>
REALISATION DE TUTORIELS EN ANGLAIS SUR DES SERVICES DE ZETAPUSH.....	48
CONCLUSION IONIC 3	48
CONCLUSION PLATEFORME ZETAPUSH	49
CONCLUSION GENERALE	49

Abstract

I did my internship at ZetaPush, which is a start-up that has developed a product that can do without back end and server management.

To facilitate the use of the product ZetaPush has set up a programming language that allows to parameterize and to deploy the services in a few minutes. My role during the internship was to create an application in order to write a tutorial to show that it is possible to use and to take in easily the language and the services proposed the company.

To realize the application I used Ionic which is a framework based on Angular and Cordova to develop mobile applications.

As for writing the tutorial, I used the Developpez.com service to format and publish the article.

Introduction

Ce stage, d'une durée de deux mois, a consisté à mettre en place des tutoriels pour aider les développeurs à utiliser le Framework ZetaPush.

Ce rapport présente le travail que j'ai effectué lors de mon stage au sein de ZetaPush. Il s'est déroulé du 3 avril au 26 mai 2017 à Rennes Atalante Beaulieu.

Le projet réalisé s'est avéré très enrichissant pour mon expérience professionnelle. En effet, ma formation s'inscrit précisément dans ce secteur (développement web). Grâce à ce stage, j'ai travaillé sur des projets qui m'ont permis d'entrevoir en quoi consiste la profession de développeur front end.

Je vous exposerai dans ce rapport en premier lieu une présentation de l'entreprise. Puis les technologies utilisées et ensuite je vous expliquerai les différents aspects de mon travail durant ces quelques mois. Enfin, en conclusion, je résumerai les apports de ce stage.

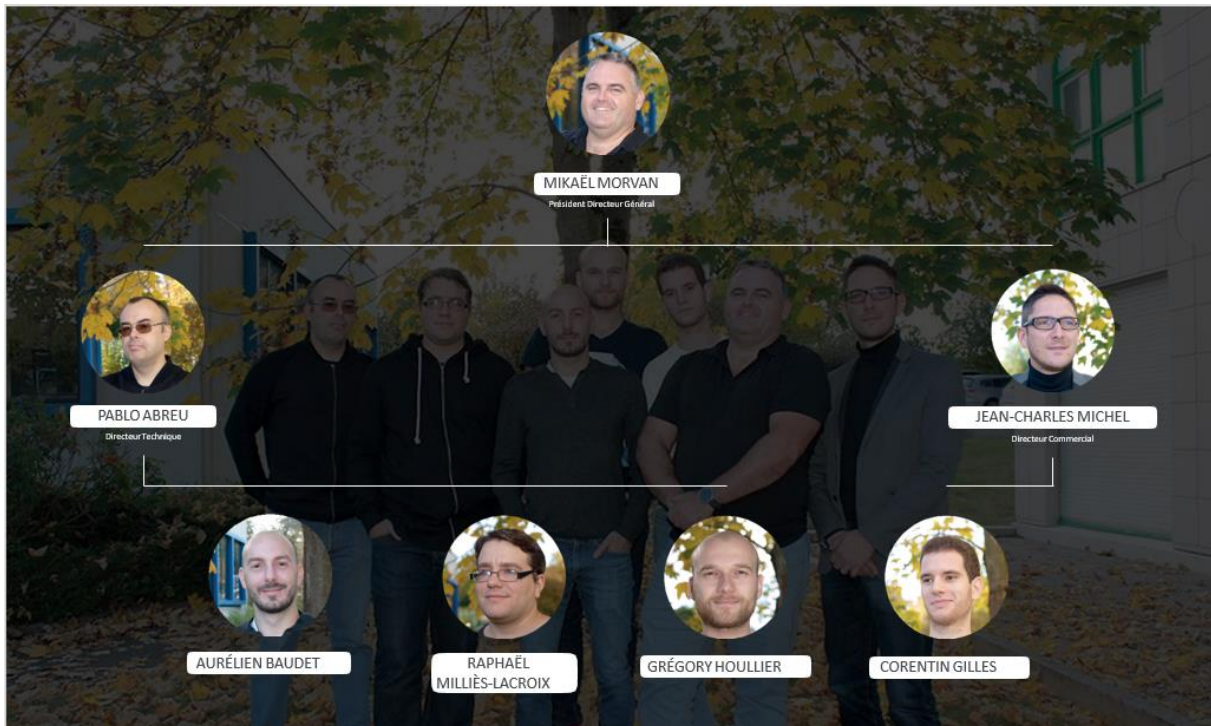
Liste des compétences couvertes par le stage

- Maquetter une application
- Mettre en place une base de données
- Développer une interface utilisateur
- Développer des composants d'accès aux données
- Développer des pages web en lien avec une base de données
- Mettre en œuvre une solution de gestion de contenu ou e-commerce
- Développer une application simple de mobilité
- Utiliser l'anglais dans son activité professionnelle en informatique

Expression des besoins de l'application à développer

Utilisation du langage et des services de ZetaPush ainsi que le Framework Ionic 3 afin de réaliser une application mobile dans le but de rédiger un tutoriel sur l'utilisation de ZetaPush.

Présentation de l'entreprise



Mikaël Morvan - Co-fondateur de ZetaPush

- Directeur général
- Inventeur de ZetaPush
- Développeur du SDK Swift

Pablo Abreu - Co-fondateur de ZetaPush

- Directeur technique de ZetaPush
- Développeur du produit et du Back End
- Responsable Hébergement et exploitation
- Correcteur d'anglais pendant son temps libre !

Gregory Houllier - Développeur Front End

- Développeur du SDK JavaScript de ZetaPush
- Développeur d'application web : HTML, CSS, JavaScript et Frameworks
- Développeur Advocate : Représente les développeurs Front End pour améliorer l'expérience utilisateur de ZetaPush
- Formateur ZMS Language : Donne des formations aux entreprises sur le ZMS Language

Jean-Charles Michel - Directeur commercial

- Définit la stratégie commerciale de la société
- Traite tous les actes de prospection et de vente

Aurélien Baudet – Développeur

Raphaël Milliès-Lacroix - Développeur

Corentin Gilles - Chargé de communication & Marketing en alternance

- Réalisation des plaquettes et outils de présentation
- Web Design
- Community Manager

Damien Le Dantec - Développeur Front End en stage

- Développeur d'application web : Angular & Ionic en utilisant la plateforme ZetaPush

Technologies utilisées

ZetaPush

ZetaPush est une plateforme BaaS « Back-end as a Service » qui propose des services hébergés dans le cloud. Les services proposés sont rapides à déployer et à paramétrer grâce au langage ZMS « ZetaPush Macro Script » développé en interne spécifiquement pour l'utilisation des services par les développeurs. ZetaPush dispose de SDK pour différents supports (SDK Swift, JavaScript, Java, Android, etc.).

ZetaPush Macro Script (ZMS)

ZMS est un langage de programmation dédié aux services de ZetaPush.

Ce langage est très similaire à JavaScript. Il a été développé par ZetaPush pour répondre aux besoins des développeurs Front end d'avoir un outil simple et fonctionnel. Les MacroScripts définissent des points d'entrées temps réel depuis internet. Il ne s'agit pas de fonctions, le return n'est pas placé dans le corps de la Macro. Il est d'ailleurs possible de remplacer le return par broadcast ce qui permet d'envoyer les informations de retour à un autre appareil que l'appelant.

Node.Js - npm

J'ai principalement utilisé NPM afin d'installer les Frameworks dont j'avais besoin.

Les Frameworks comme Ionic et Angular disposent de scripts utilisables en lignes de commandes et exécutés par Node.Js afin par exemple de déployer un serveur web.

Exemple : ionic serve

Angular 4

Angular 4 est un Framework open-source développé par Google.

Il est fondé sur l'extension du langage HTML par de nouvelles balises et attributs pour aboutir à une définition déclarative des pages web.

Le code HTML étendu représente alors la partie « vue » du patron MVC.

Depuis la version 2, Angular intègre un transpileur de TypeScript car il apporte de nombreuses facilités de développement, comme par exemple une auto-complétion très poussée, pour peu que l'on utilise un IDE avancé. Par ailleurs, en imposant une syntaxe plus stricte et un typage plus statique, TypeScript rend le code plus lisible et maintenable.

Ionic 3

Ionic est un Framework open-source créé en 2013. Il utilise les outils web comme HTML, CSS et JavaScript pour créer des applications multi plateforme. Dans la version 3, Ionic intègre Angular 4 qui est équipé d'un transpileur TypeScript.

De plus il embarque une brique Cordova ce qui lui permet de créer une application mobile (Android, IOS, Windows Phone) en utilisant le navigateur natif de l'appareil.

Par ailleurs Cordova lui permet également d'utiliser des fonctionnalités natives comme la caméra.

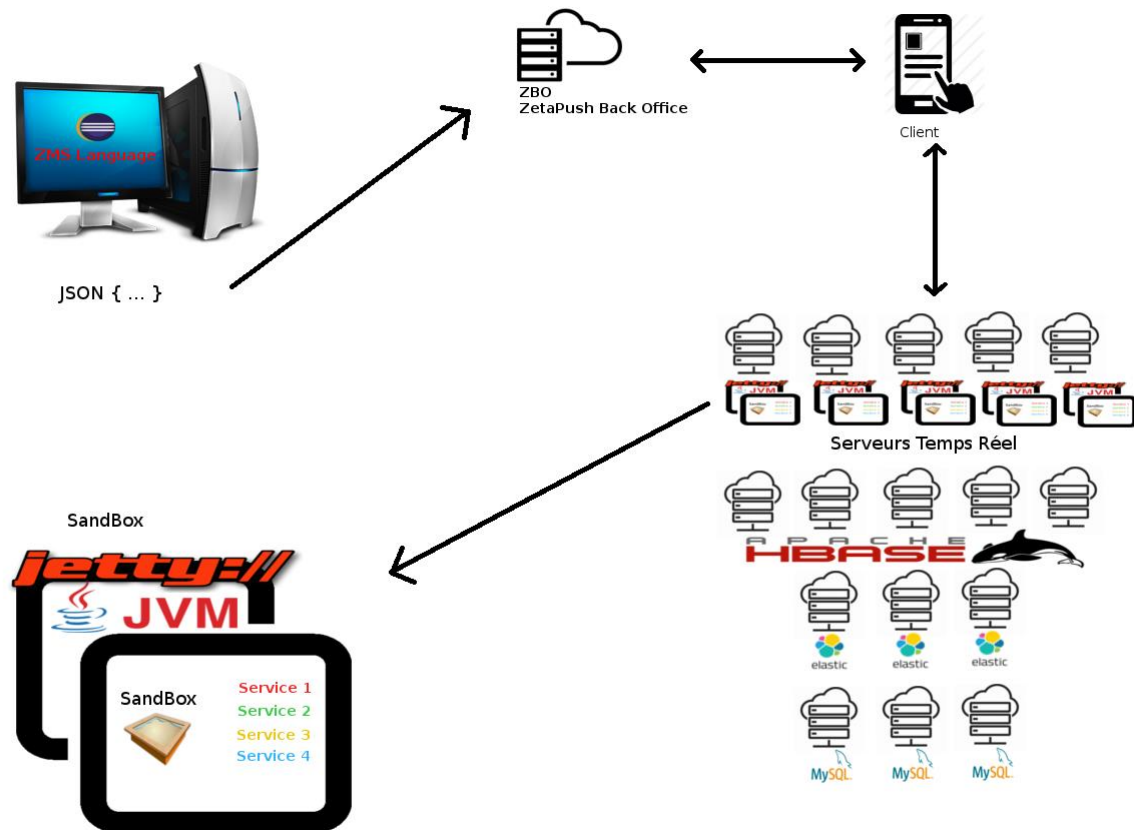
TypeScript

TypeScript est un langage de programmation libre et open-source développé par Microsoft qui a pour but d'améliorer et de sécuriser la production de code JavaScript. C'est un sur-ensemble de JavaScript. Le code TypeScript est transcompilé en JavaScript, pouvant ainsi être interprété par n'importe quel navigateur web ou moteur JavaScript.

Il a été cocréé par Anders Hejlsberg, principal inventeur de C#.

C'est grâce à la formation Développeur Logiciel où nous avons appris C# que j'ai facilement pris en main ce langage.

Fonctionnement de ZetaPush



ZBO dispose des adresses de tous les Serveurs Temps Réel de ZetaPush. Quand une application souhaite utiliser un service lié à une SandBox, ZBO envoie l'adresse d'un Serveur Temps Réel auquel l'application pourra se connecter.

La connexion s'effectue via WebSocket.io.

Tous les STR disposent d'une copie des SandBoxes.

Derrière les STR nous retrouvons des serveurs HBase pour les utilisateurs de NoSQL.

Viennent ensuite des serveurs Elastic Search puis des serveurs MySQL.

Quand un développeur programme en ZMS Language et qu'il déploie son code, le plugin ZetaPush va compiler le ZMS en JSON puis l'envoyer au serveur.

La suite de la procédure étant propre à ZetaPush je ne peux pas communiquer dessus.

Exigences et contraintes

La principale exigence était de faire une application utilisant les services de ZetaPush sur laquelle il serait possible de faire des tutoriels. La seconde exigence était de rédiger un tutoriel sur le site « Developpez.com ».

En ce qui concerne les contraintes techniques, je devais utiliser le Framework Ionic pour la partie front end et la documentation de ZetaPush n'était pas encore en place. De plus, je ne connaissais rien sur le Framework Ionic/Angular ou en JavaScript/TypeScript.

Travail effectué lors du stage

Avant d'entrer au cœur du sujet, j'ai dû installer plusieurs IDE, outils et Framework.

Dans un premier temps il m'a fallu mettre en place Eclipse pour la partie back end réalisé en ZMS avec un plugin dédié, fourni par ZetaPush.

Dans un second temps j'ai mis en place Atom pour la partie TypeScript – JavaScript – HTML – CSS. C'est-à-dire pour la partie Front end

J'ai ensuite installé Node.js afin de pouvoir utiliser NPM et ainsi installer Angular 4 et Ionic 3.

Réalisation d'une application de paris

Spécifications fonctionnelles

- Parier sur la position d'un joueur
- Historique des paris
- Gestion d'utilisateurs

Spécifications techniques

- Utilisation du langage ZMS
- Utilisation des services de ZetaPush
- Utilisation du Framework Ionic

Trouver le thème d'une application

Avant d'entrer en stage, mon tuteur m'a demandé de trouver le thème d'une application qui puisse donner envie aux développeurs d'utiliser ZetaPush.

Sur la base du principe que les développeurs sont aussi des geeks et qu'ils aiment jouer aux jeux vidéo, j'ai cherché à savoir quel jeu est le plus regardé en streaming. C'est de là que j'ai trouvé League of Legends, un MOBA (Multiplayer Online Battle Arena) ou « arène de bataille en ligne multijoueur ». Je ne connaissais pas ce jeu je l'ai donc découvert en parti avec cette application.

Après avoir trouvé le thème de l'application, il fallait encore trouver sur quoi porterait l'application. J'ai choisi une application de paris pour le côté attractif et interactif. L'utilisateur pourra parier sur la position qu'atteindra un joueur au classement dans 48 heures.

Création d'un nouveau projet

Grâce à Ionic il est possible de réaliser des applications mobiles multi plateformes. C'est donc avec ce Framework que j'ai travaillé.

Je n'avais aucune connaissance sur Ionic et cette application m'a permis d'apprendre énormément.

Pour démarrer le projet j'ai fait l'installation avec une configuration vierge :

```
ionic start ParisLoL blank --v2
```

Voici la ligne de commande qui permet de démarrer un nouveau projet.

Le premier mot indique à Windows que nous allons utiliser le Framework Ionic. Ensuite nous disons à Ionic de démarrer un projet portant le nom de ParisLoL avec un modèle « blank » ou vierge. Pour terminer, j'ajoute « --v2 » pour lui indiquer que je souhaite utiliser la version 2 du Framework.

Une fois l'installation terminée il faut se rendre dans le dossier « créer » portant le nom du projet, ici comme dans l'exemple : ParisLoL il suffit d'utiliser la commande `cd ParisLoL` sous Windows.

Mise en place des écrans

Avant de me lancer dans la programmation, j'ai passé un peu de temps à mettre en forme les écrans sur Mockups afin de gagner du temps par la suite (voir annexe 1).

À partir de là je savais à quoi allait ressembler l'application finale et je me suis mis à travailler sur la partie Ionic.

Afin de prendre en main Ionic, je n'ai pas fait tout de suite de relation avec la base de données. J'ai mis en premier lieu les écrans réalisés sur Mockups.

N'ayant pas une grande expérience, j'ai utilisé la documentation de Ionic pour comprendre le fonctionnement et utiliser les différents composants.

Je souhaitais tout de même intégrer des données dynamiques dans l'application et j'avais besoin de récupérer les informations sur les joueurs de League of Legends pour pouvoir les afficher. C'est à partir de là que j'ai voulu respecter le design pattern MVC que nous avons appris lors de la formation. Pour réaliser cela, je suis monté en compétence sur Ionic grâce à des tutoriels en anglais.

Design pattern MVC sur Ionic

Provider

Le provider est la partie qui va alimenter les modèles en récupérant les données des différentes sources disponibles. Pour commencer l'application j'ai utilisé un provider pour récupérer les informations des joueurs de League of Legends via l'API de Riot Game (Société éditrice de League of Legends).

Cette API prend en compte différents paramètres. À savoir une clé privée que l'on obtient en s'inscrivant sur le site de Riot Game, la région du jeu que l'on souhaite exploiter ici West Européen et pour finir les données que nous souhaitons récupérer, en l'occurrence les informations sur les 100 meilleurs joueurs.

Pour créer un provider sur Ionic il suffit d'entrer une commande :

```
ionic g provider player-infos-provider
```

Ionic va générer un fichier dans le dossier provider portant le nom de notre provider avec un modèle de base.

C'est dans ce modèle de provider que j'ai mis en place la récupération des données via l'URL de l'API de Riot Game et que j'ai passé les données à un modèle.

Modèle

Le modèle est généralement placé dans le dossier ou projet BO sous C# ou dans le dossier model en Java. Ici les modèles sont placés dans le dossier de notre choix car il n'y a pas de commande pour les générer automatiquement.

Afin de rendre le projet plus lisible, j'ai placé tous les modèles dans un dossier nommé model.

Pour les informations des joueurs de League of Legends j'ai créé un fichier nommé player-model.

Player-model a des propriétés privées, des getters et setters public pour toutes les informations et un constructeur public vide afin de pouvoir construire l'objet player via les getters et setters.

Voici un exemple de propriété et de getter, setter :

```
private id :string ;  
public get Id() : string  
{  
    return this.id  
}  
public set Id(id :string)  
{  
    this.id = id ;  
}
```

Il est important de mettre en export la classe pour pouvoir l'utiliser dans d'autre classe.

Exemple :

```
export class PlayerModel { ... }
```

Page

Une page est à la fois la partie Vue et la partie Contrôleur du projet car chaque page dispose d'une partie HTML et d'une partie TypeScript.

Afin d'afficher les joueurs, j'ai créé une page nommé « player-page »

Pour générer une page il suffit d'entrer la commande suivante :

```
ionic g page player-page
```

Avant de pouvoir utiliser les données dans la page il faut effectuer l'importation de la classe. Pour pouvoir faire l'importation il est nécessaire de faire la déclaration dans le fichier app.module.ts

C'est dans ce fichier que toutes les pages ou providers générés doivent être déclarés avant utilisation.

En ce qui concerne la Vue il s'agit d'un fichier HTML portant le nom de la page générée.

Dans le code HTML il est possible d'utiliser des variables mais aussi des balises propres à Angular ou Ionic.

Par exemple, pour effectuer une boucle for, très utile pour générer l’affichage d’une liste, il suffit de faire appel à la balise personnalisée d’Angular ngFor :

```
<p *ngFor = 'let utilisateur in utilisateurs'> {{utilisateur.nom}} </p>
```

Cela va générer autant de balises HTML <p> avec comme contenu le nom de l’utilisateur, qu’il y a d’utilisateurs dans la liste.

Tous les points forts d’Angular se retrouvent dans Ionic. Ainsi dans le cas d’un formulaire il est possible de faire du two way binding qui permet de définir le contenu d’une variable en fonction de la valeur d’un champ de formulaire et inversement, le champ du formulaire équivaut à la valeur de la variable.

Par exemple, si dans mon contrôleur j’ai une variable nommée « Age défini à 18 ans », je peux assigner le contenu de cette variable à un champ input de la façon suivante :

```
[(ngModel)] = 'age'
```

Le champ input portant l’attribut ngModel sera alors relié à la variable du contrôleur. Chaque modification dans la vue aura pour effet de modifier la valeur dans le contrôleur et inversement.

Conclusion : design pattern avec Ionic

Mettre en place le modèle MVC sur Ionic a été très simple et il m’a permis de rendre la structure et le code du projet nettement plus lisible facilitant ainsi le développement de l’application.

Composants Ionic

Les composants que fournit Ionic sont indispensables pour le développement d’une application sur le Framework.

Les applications Ionic sont constituées de blocs de haut niveau appelés composants. Les composants permettent de construire rapidement une interface pour une application. Ionic est livré avec un certain nombre d’entre eux.

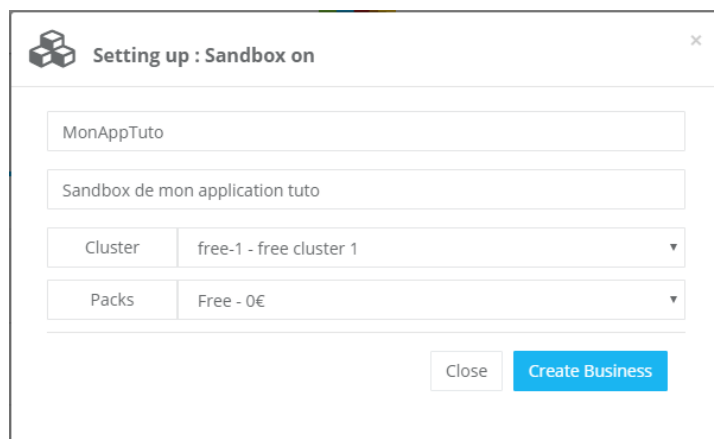
La majeure partie des composants sont des outils pour les formulaires de saisie utilisateur. Le reste permet de rendre l'expérience utilisateur plus attractive en ajoutant des surcouches à l'affichage.

Création d'un projet ZMS sous Eclipse

Pour réaliser l'application de paris, le projet a nécessité l'utilisation d'un service de base de données relationnel MySQL. Pour cela j'ai créé un projet ZMS sous Eclipse grâce au plugin de ZetaPush installé sur l'IDE.

Avant de pouvoir créer un projet sous Eclipse il est important de s'inscrire sur le site de ZetaPush afin de créer des identifiants et de déployer une SandBox.

Une fois l'inscription au service effectué, dans la section AllSandBox il faut ajouter une SandBox à notre compte :



Depuis Eclipse il est très facile d'installer le plugin. Il suffit de se rendre sur le menu supérieur dans Help puis install New Software puis de cliquer sur Add et d'entrer les informations suivantes :

Name : ZetaPush

Location : <https://zms-site.zetapush.com/releases/>

Pour terminer l'installation, il suffit de faire Select All puis Next et Finish.

Une fois le plugin installé, Eclipse redémarrera et la barre d'outils sera disponible en haut de l'interface :



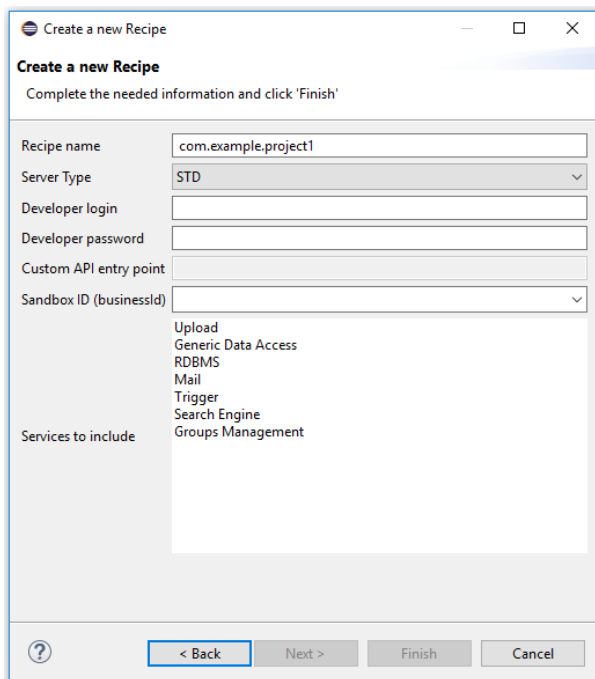
J'ai testé le plugin sous Eclipse de la version Mars.2 à Neon.3.

La barre d'outils fournit divers raccourcis. De gauche à droite, on peut retrouver :

- Déploiement des MacroScripts sur la SandBox
- Déploiement des Services sur la SandBox
- Mode Debug pour les Macros
- Tester les Macros en local
- Configuration du plugin
- Outils de Debug

Une fois le plugin installé, il est possible de créer un nouveau projet depuis le menu File, New, Other, ZMS Recipe.

Une nouvelle fenêtre s'ouvrira pour demander les informations sur le projet :



Recipe name : il s'agit du nom du package

Serveur type : STD (Standalone)

Developer login et password : Il s'agit là des identifiants de connexion à l'administration du site ZetaPush.com.

SandBox ID : l'ID est disponible dans l'administration après déploiement :



Une fois cette étape terminée, plusieurs fichiers sont générés :

- Un répertoire *src* avec un fichier exemple *welcome.zms*
- Un fichier *init.zms* qui sert à initialiser les services une fois qu'ils ont été déployés
- Un fichier *README.md*
- Un fichier *recipe.zms* qui permet de créer les services que vous souhaitez utiliser
- Un fichier *zms.properties* qui sert à gérer les propriétés globales au projet
- Un fichier *zms.user.properties* qui sert à gérer les propriétés du développeur

Déploiement de services via ZetaPush

Afin de déployer des services il est nécessaire de les déclarer dans le fichier *recipe.zms*.

N'ayant pas de documentation à disposition, mais seulement une référence j'ai dû apprendre en faisant.

Dans un projet ZMS il faut utiliser du ZMS Language !

Voici la déclaration type d'un service en prenant pour exemple MySQL :

```
/** a classical SQL database */  
  
service database = rdbms(__default);
```

La déclaration commence par le mot clé `service` suivi du nom que l'on souhaite donner à la variable pour utiliser le service. Vient ensuite le nom technique du service ici : `rdbms` pour Relational Database : SQL storage

Entre les parenthèses il s'agit du nom qui sera attribué au service dans la SandBox. En laissant `__default`, le nom du service sera : `rdbms_0`

Utilisation d'un service et programmation en ZMS sur Eclipse

Dans un fichier *zms* placé dans le dossier *src* du projet, il faut respecter la syntaxe suivante :

```
/**
 * Insertion d'un nouvel utilisateur dans la base de données users
 */
macroscript insertUser(string userKey,
string pseudo,
string dateInscription,
number argent,
number visible,
string server,
string dateConnexion,
string description,
string email,
number groupId)
{
    @@(database) INSERT INTO users
    (userKey,nickname,dateInscription,argent,visible,server,dateConnection,d
escription,email,groupId)
    VALUES (:{userKey},{pseudo}, :{dateInscription}, :{argent}
, :{visible}, :{server}, :{dateConnexion},
:{description},{email},{groupId}
);
} return { message:'insertion ok'}
```

Dans cet exemple, la MacroScript commence obligatoirement par la clé *macroscript* suivi par le nom de la macro. Vient ensuite entre parenthèse les paramètres. Il est possible de typer les paramètres comme dans l'exemple (*string*, *number*, *boolean*) et mettre des annotations exemple :

@NotNull

Cette annotation définit un paramètre obligatoire.

Entre les crochets, dans le corps de la Macro nous retrouvons la requête SQL.

@@(database)

Une des particularités ici est le fait d'utiliser la variable *database* encapsulée exemple :

Cette encapsulation va permettre de faire une requête SQL.

La syntaxe de la requête est identique à une syntaxe SQL classique avec une deuxième particularité, l'encapsulation des variables exemple : *:{variable}*

Dans cet exemple la MacroScript utilise un *return* qui renvoie une variable nommée *message* avec comme contenu *'insertion ok'*

Un *return* renverra les données à l'appareil en faisant la demande.

Il est possible d'utiliser *broadcast* à la place de *return* afin de retourner les données à l'ensemble des appareils connecté d'un utilisateur.

Une fois les MacroScripts terminé, il suffit de cliquer sur l'icône en forme de fusée rouge afin de déployer le service, puis sur l'icône bleu afin de compiler et déployer les MacroScripts.

Configuration de l'application pour utiliser ZetaPush

Installation du SDK JS

Pour utiliser un service il faut tout d'abord installer le package ZetaPush via la commande suivante :

```
npm install zetapush-js
```

npm va se charger d'installer le SDK JavaScript dans le répertoire module de notre application, il faut ensuite le déclarer dans le fichier app.module.ts puis nous pouvons l'importer et l'utiliser dans toute l'application.

Création de méthodes appelant les MacroScripts

Tout d'abord j'ai pour ma part crée un provider :

```
ionic g provider user-provider
```

Dans ce fournisseur j'ai supprimé l'intégralité du contenu afin de créer un provider correspondant à mes besoins.

Il est nécessaire de faire l'importation du SDK JavaScript :

```
import { services } from 'zetapush-js' ;
```

Puis d'étendre la classe Macro :

```
export class UserProvider extends services.Macro { ... }
```

Ensuite il faut créer les méthodes qui utiliseront nos MacroScripts. Pour ma part j'ai utilisé le même nom de fonction que pour les MacroScripts :

```
InsertUser(login, password, email,pseudo) {  
  This.$publish('InsertUser',{login,password,email,pseudo}) ;  
}
```

Dans cet exemple, la fonction *InsertUser* à plusieurs paramètres, ces paramètres doivent porter exactement le même nom qu'à la création des MacroScripts.

Dans le corps de la fonction viens ensuite une méthode provenant du SDK JS : *\$publish*

Cette méthode prend en premier paramètre le nom de la MacroScript puis les variables à envoyer à cette Macro.

Création d'un client de connexion

Pour permettre à l'application de communiquer avec un serveur temps réel de ZetaPush il faut mettre en place et configurer un client.

Pour cela le SDK JS nous facilite le travail.

```
ionic g provider client-provider
```

J'ai créé un provider dans lequel je fais appelle à SmartClient :

```
import { SmartClient } from 'zetapush-js' ;
```

Dans ClientProvider je fais un Singleton afin de toujours avoir une instance de connexion :

```
client : SmartClient ;  
  
public getInstance() {  
  if(typeof this.client === 'undefined' || this.client === null){  
    this.client = new SmartClient({  
      sandboxId : <SANDBOXID>,  
      apiUrl : <APIURL>  
    }) ;  
  }  
  return this.client ; }  

```


Après avoir fait la déclaration du provider ClientProvider dans le fichier app.module.ts il est possible d'utiliser dans le constructeur d'une page le client afin d'établir une connexion.

```
constructor(private client :ClientProvider)
{
    this.client.getInstance.connect() ;
}
```

Utilisation de ZetaPush dans l'application

Il est à présent possible d'utiliser toutes les MacroScripts ou services dans une application web.

Toujours sous Ionic, il faut dans un premier temps importer le ClientProvider et le déclarer dans le constructeur de la page d'entrée de l'application. Puis dans un second temps importer le provider des MacroScripts, UserProvider et déclarer une variable qui nous permettra d'accéder à toutes les fonctions du provider.

```
import { ClientProvider } from '../providers/client-provider';
import { UserProvider } from '../providers/user-provider';

export class HomePage {
    // Déclaration d'une variable pour UserProvider
    private userService =
    this.client.getInstance().createAsyncMacroService({
        Type : UserProvider,
        deploymentId : 'macro_o'
    }) as UserProvider

    // Ajout du client dans le constructeur
    constructor(private client :ClientProvider)
    {
        This.client.getInstance.connect() ;
    }
}
```

Dans cet exemple, j'utilise la méthode *connect()* directement dans le constructeur. Cela aura pour conséquence d'utiliser le service weak authentication de ZetaPush.

Le service Weak Auth n'a pas besoin de login / mot de passe pour établir une connexion.

Dans le cadre d'une page de connexion utilisateur, il faudrait faire appel avant la connexion à *setCredentials(login,password)*. Ce qui aurait pour effet d'utiliser le service Simple Authentication qui lui gère les comptes utilisateurs.

Il est possible d'écouter la connexion via un listener :

```
this.client.getInstance().addConnectionStatusListener({
  onConnectionEstablished : () => {
    this.userService.InserUser(login,
password,email,...).then((result) =>
    {
      if(result.verif)
      {
        console.log('Bienvenue');
      }
    }).catch((error) => {
      console.error(error);
    });
  }
});
```

Dans cet exemple, une fois la connexion établie, j'utilise le provider UserProvider pour insérer un nouvel utilisateur dans la base de données SQL. L'utilisation de then() permet de retourner un objet Promise qui est utilisé pour réaliser des traitements de façon asynchrone. Une promesse représente une valeur qui peut être disponible maintenant, dans le futur, voir jamais.

À partir de là, j'ai effectué des requêtes SQL pour réaliser l'application et ainsi mettre en place l'application de paris comprenant les fonctions suivantes :

- Multi paris
- Historique
- Statistiques
- Progression des joueurs sur 48 heures
- Commentaire sur la fiche d'un joueur
- Forum de discussions
- Gestion de groupes
- Forum de groupe
- Chat de groupe
- Gestion d'amis
- Messagerie privée
- Préférence utilisateur
- Gestion d'utilisateur

Tout ce qui concerne la partie « sociale » de l'application n'était pas prévu au départ, c'est mon tuteur qui a souhaité que j'ajoute ces fonctionnalités, je me suis donc adapté pour intégrer ces fonctions.

Conclusion

L'application était trop lourde pour réaliser un tutoriel dessus. De plus je n'utilisais qu'un seul des services proposé par ZetaPush et je commençais à peine à prendre en main le Framework Ionic.

J'ai tout de même réalisé cette application en 10 jours en autonomie.

En ce qui concerne la partie ZetaPush, manquant d'expérience en développement et du fait qu'il n'y ait pas de documentation j'ai eu un peu de mal à comprendre son mécanisme et voir tout son potentiel.

Réalisation d'une application Mangathèque

Suite au fait que l'application de paris soit trop lourde pour rédiger un tutoriel complet sur ce sujet, mon tuteur, monsieur Mikaël Morvan m'a demandé de créer une application légère qui utiliserait un seul service pour expliquer le fonctionnement de ce dernier dans un tutoriel que je publierais sur le site Developpez.com.

Avant de commencer à développer l'application, j'ai dû répondre à la question suivante posé par mon tuteur : Que possèdent tous les geeks ?

J'ai répondu à cette question par : une collection.

J'ai donc décidé avec l'accord de mon tuteur de créer une application de gestion de mangas.

Cette application utiliserait le service SQL que j'ai pris en main les jours précédents sur la première application.

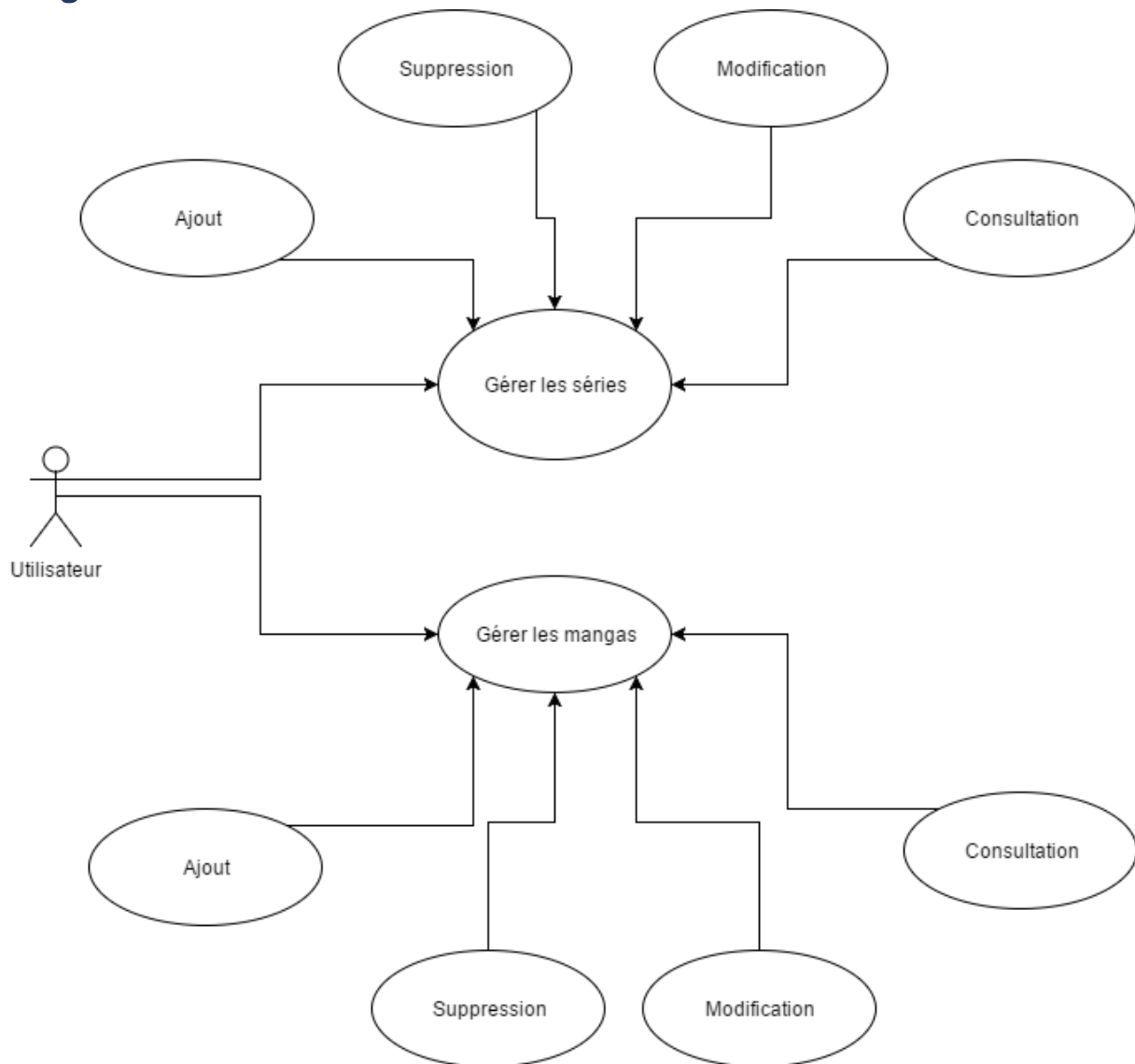
Spécifications fonctionnelles

- Gestion de séries
- Gestion de mangas
- Gestion d'utilisateurs

Spécifications techniques

- Utilisation de ZMS
- Utilisation des services ZetaPush
- Utilisation du Framework Ionic

Diagramme d'utilisation



- 1 – L'utilisateur accède à la page affichant les séries
- 2 – L'utilisateur peut ajouter une série
- 3 – L'utilisateur peut supprimer une série
- 4 – L'utilisateur peut modifier une série
- 5 – L'utilisateur peut consulter une série
- 6 – L'utilisateur peut ajouter un manga
- 7 – L'utilisateur peut supprimer un manga

8 – L'utilisateur peut modifier un manga

9 – L'utilisateur peut consulter un manga

Mise en place de l'application sous Ionic 3

Comme pour l'application de paris, la Mangathèque sera développée sous le Framework Ionic 3.

Comme précédemment j'ai généré un nouveau projet vierge pour y inclure tous mes éléments.

```
ionic start MonAppTuto -v2  
cd MonAppTuto  
npm install zetapush-js --save  
ionic serve --lab
```

Suivant le diagramme d'utilisation, l'utilisateur entre par l'affichage des séries. Ionic génère le projet avec une page Home définie comme page de démarrage, j'ai commencé à coder dans cette page.

Programmation du service et des MacroScripts ZetaPush

Sachant la direction à prendre pour réaliser cette application j'ai pu créer les services et ZMS d'un seul bloc.

Voici les tables de la mangathèque :

Mangas	Séries
Identifiant unique	Identifiant unique
Identifiant de la série	Nom
Titre	Résumé
Tome	
Résumé	
Est-il lu	
Est-il Acquis	

Déclaration du service

Pour commencer, j'ai généré un nouveau projet ZMS sous Eclipse et dans le fichier *recipe.zms* j'ai déclaré le service que j'allais utiliser :

```
/** a classical SQL database */  
service database = rdbms(__default);
```

Création des tables SQL

Le code des tables se rédige de la manière suivante :

```
database.rdbms_ddl({statement : 'NOTRE CODE ICI'}) ;
```

Ensuite dans le fichier *init.zms* j'ai créé mes tables :

```
database.rdbms_ddl({statement : '  
CREATE TABLE IF NOT EXISTS mangas (  
  id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  titre VARCHAR(150) NOT NULL,  
  tome INT NOT NULL,  
  resume TEXT NULL,  
  idSerie INT NOT NULL,  
  isLu TINYINT(1) NOT NULL,  
  isAcquis TINYINT(1) NOT NULL  
) '});  
  
database.rdbms_ddl({statement : '  
CREATE TABLE IF NOT EXISTS series (  
  id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  nom VARCHAR(150) NOT NULL,  
  resume TEXT NULL  
) '});
```

Une fois cette opération terminée, j'ai déployé le code et le service via la barre d'outils du plugin de ZetaPush pour Eclipse.

Dans la console si tout c'est bien passé la console affiche success :

```
[INFO] *****  
[INFO] **          SUCCESS          **  
[INFO] *****
```

ZetaPush Macro Scripts

Pour réaliser les MacroScripts du projet, j'ai splitté le contenu en deux fichiers. Le premier est dédié aux séries et ce nomme Serie.zms.

Serie.zms :

```
/**
 * Ajout d'une nouvelle série dans la base de donnée
 */
macroscript SerieInsert(
    @NotNull string nom,
    @NotNull string userKey,
    string resume
)
{
    @@(database) INSERT INTO series (nom,resume,userKey) VALUES
    (:{nom},{resume},{userKey});
    var id = @@(database) SELECT id FROM series WHERE id= LAST_INSERT_ID();
    } return { id }

/**
 * Récupération de toutes les séries
 */
macroscript SerieGetAll(@NotNull string userKey)
{
    var series = @@(database) SELECT * FROM series WHERE userKey = :{userKey};

    } return { series }

/**
 * Récupération d'une série via son identifiant unique
 */
macroscript SerieGetById(@NotNull number idSerie)
{
    var serie = @@(database) SELECT * FROM series WHERE id = :{idSerie};
    } broadcast { serie } on channel __selfName
```

```
/**
 * Mise à jour d'une série
 */
macroscript SerieUpdateById(@NotNull number idSerie,
                             @NotNull string nom,
                             string resume
)
{
  @@(database) UPDATE series SET nom = :{nom}, resume = :{resume} WHERE id =
  :{idSerie};
}
/**
 * Suppression d'une série et de tous les mangas associés
 */
macroscript SerieDeleteById(@NotNull number idSerie)
{
  @@(database) DELETE FROM mangas WHERE idSerie = :{idSerie};
  @@(database) DELETE FROM series WHERE id = :{idSerie};
}
```

Chaque macro associée à la gestion des séries est préfixée du mot Serie toujours dans un but de lisibilité en cas de reprise future du projet ou de transfert à un tierce développeur ou dans le cadre de la rédaction d'un tutoriel.

Les MacroScripts reprennent le principe d'une DAO ou d'une DAL classique à savoir insertion, récupération de tous les objets, d'un objet par son identifiant, suppression d'un objet et mise à jour d'un objet par son identifiant.

Concernant les Mangas, il s'agit de la même chose.

Manga.zms


```
/**
 * Ajout d'un nouveau manga dans la base de données
 */
macroscript MangaInsert(
    @NotNull string titre,
    @NotNull string tome,
    string resume,
    @NotNull number idSerie,
    @NotNull boolean isLu,
    @NotNull boolean isAcquis,
    string couverture
)
{
    @@(database) INSERT INTO mangas
    (titre,tome,resume,idSerie,isLu,isAcquis,couverture) VALUES
    (:{titre},{tome},{resume},{idSerie},{isLu},{isAcquis},{couverture});
    var id = @@(database) SELECT id FROM mangas WHERE id= LAST_INSERT_ID();
    } return { id }

/**
 * Récupération de tous les mangas d'une série
 */
macroscript MangaGetAll(@NotNull number idSerie)
{
    var mangas = @@(database) SELECT * FROM mangas WHERE idSerie = :{idSerie};
    } return { mangas }
```

```
/**
 * Récupération d'un manga via son identifiant unique
 */
macroscript MangaGetById(@NotNull number idManga)
{
    var manga = @@(database) SELECT * FROM mangas WHERE id = :{idManga};
    return { manga }
}

/**
 * Mise à jour d'un manga
 */
macroscript MangaUpdateById(@NotNull number idManga,
    @NotNull string titre,
    @NotNull string tome,
    string resume,
    @NotNull number idSerie,
    @NotNull boolean isLu,
    @NotNull boolean isAcquis,
    string couverture
)
{
    @@(database) UPDATE mangas SET titre = :{titre}, tome = :{tome}, resume =
    :{resume}
                                ,idSerie=:{idSerie},isLu = :{isLu},
    isAcquis=:{isAcquis},
                                couverture=:{couverture}
                                WHERE id = :{idManga};
}

/**
 * Suppression d'un manga
 */
macroscript MangaDeleteById(@NotNull number idManga)
{
    @@(database) DELETE FROM mangas WHERE id = :{idManga};
}
```

Je tiens à préciser que les MacroScripts que j'ai réalisés sont minimes à côté de celles d'un développeur expérimenté. Une documentation de référence sur le ZMS et les services permet d'en apprendre davantage sur ce langage une fois que l'on a pris en main les bases.

Une fois les MacroScripts faites, j'ai déployé le code via le plugin pour Eclipse de ZetaPush pouvant ainsi les utiliser dans la partie Ionic.

Création de l'interface

Pour utiliser les macros dans l'application j'ai répété les opérations comme pour l'application de paris. Dans un premier temps j'ai généré 3 providers, un pour le client de connexion, un pour les séries et un pour les mangas.

Puis j'ai paramétré le client de connexion avec l'ID de ma sandbox que j'ai créé avant de faire les MacroScripts et j'ai remplacé le contenu des providers pour les séries et pour les mangas avec mon code.

J'ai créé deux modèles. Un premier pour les séries et un second pour les mangas. Chaque modèle dispose de propriété et de getter / setter.

```
/**
 * Properties
 */
private titre : string ;

/**
 * Getter
 */
get Titre() :string {
    return this.titre ;
}
set Titre(titre :string){
    this.titre = titre ;
}
constructor(titre :string,.....)
{
    this.Titre = titre ;
}
```

Affichage des séries

La partie contrôleur de la page Home importe le provider des séries afin de récupérer le contenu de la base de données et pouvoir l'affiché. À l'appel des données, j'utilise une promesse, pour pouvoir déterminer les actions à faire après récupération ou en cas d'échec.

En cas de réussite :

```
this.serieService.SerieGetAll(this.user.UserKey).then((result) => {

    this.series = new Array<SerieModel>();

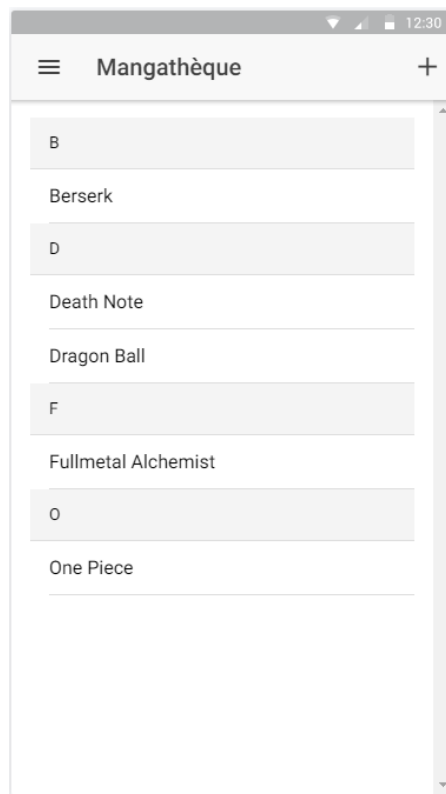
    for (let serie of result.series) {
        let newSerie = new SerieModel();
        newSerie.SetId(serie.id);
        newSerie.SetNom(serie.nom);
        newSerie.SetResume(serie.resume);
        this.series.push(newSerie);
    }
    this.groupSerie(this.series);
```

Tout d'abord je déclare une nouvelle variable série qui sera un tableau de SerieModel.

Ensuite j'effectue une boucle sur le résultat obtenu.

Dans cette boucle je crée une variable utilisant le modèle des séries à la volé afin de l'ajouter aux tableaux des séries.

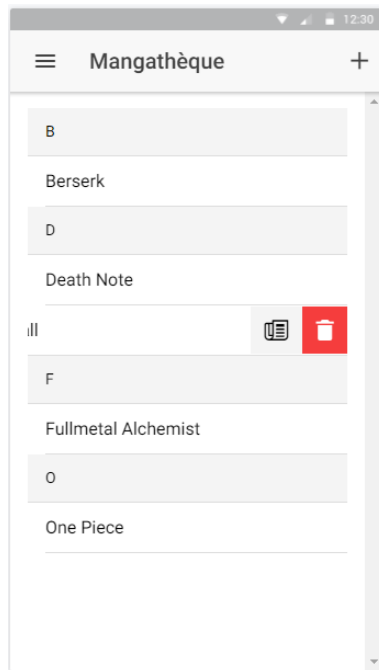
Pour terminer je fais appel à une fonction que j'ai créé pour grouper les séries par ordre alphabétique afin de les afficher trié.



Une fois les données récupérées je peux les afficher dans la partie HTML.

Ionic utilisant HTML 5 et des balises personnalisées il est possible de mettre en place des éléments très avancés. Ici j'ai choisi d'utiliser ion-item-options combiné avec ion-item-sliding. Les deux éléments couplés permettent d'afficher des boutons qu'il est possible de rendre visible en faisant glisser un item.

Ainsi j'ai placé le bouton pour modifier et le bouton pour supprimer dans ces balises :



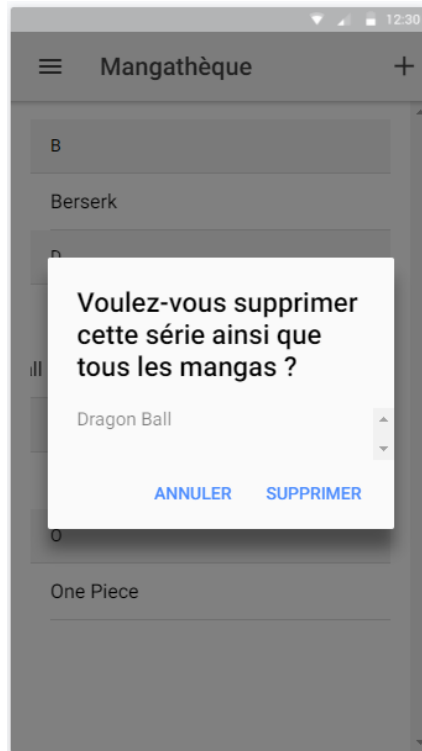
```
<ion-item-options>
    <button ion-button icon-only (click)="editSerie(serie)"
color="light">
    <ion-icon name="paper"></ion-icon>
</button>

    <button ion-button icon-only (click)="showConfirm(serie)"
color="danger">
    <ion-icon name="trash"></ion-icon>
</button>

</ion-item-options>
```

La modification d'une série affiche une surcouche à l'écran en cours via le composant Modal de Ionic. En ce qui concerne la suppression, un message de confirmation est affiché pour être sûr de l'action de l'utilisateur.

Pour afficher le message de confirmation j'ai utilisé un composant alertController :



```
showConfirm(serie) {  
    let confirm = this.alertCtrl.create({  
        title: 'Voulez-vous supprimer cette série ainsi que tous les  
mangas ?',  
        message: serie.GetNom(),  
        buttons: [  
            {  
                text: 'Annuler',  
                handler: () => {  
                    }  
            },  
            {  
                text: 'Supprimer',  
                handler: () => {  
                    this.deleteSerie(serie)  
                }  
            }  
        ]  
    });  
    confirm.present();  
}
```

Si l'utilisateur choisit de supprimer la série, la fonction deleteSerie est appelée.

deleteSerie s'occupe dans un premier temps de retirer la série du tableau les affichants pour actualiser l'écran. Dans un second temps le tableau est repassé à la fonction qui s'occupe de grouper les séries et pour terminer, j'utilise le provider des séries pour effectuer la suppression en base de données.

Voici le code pour regrouper les séries :

```
groupSerie(series) {
  this.groupedSeries = [];
  let sortedSeries = series.sort((a, b) => {
    if (a.GetNom() < b.GetNom()) {
      return -1;
    }
    if (a.GetNom() > b.GetNom()) {
      return 1;
    }
    return 0;
  });
  let currentLetter;
  let currentSeries = [];

  sortedSeries.forEach((value, index) => {
    if (value.GetNom().charAt(0) !== currentLetter) {
      currentLetter = value.GetNom().charAt(0);
      let newGroup = {
        letter: currentLetter,
        series: []
      };
      currentSeries = newGroup.series;
      this.groupedSeries.push(newGroup);
    }
    currentSeries.push(value);
  });
}
```

Cette fonction ordonne les séries par nom pour commencer.

Puis va vérifier que l'on n'ait pas déjà enregistré le groupe de série commençant par la lettre en cours et va récupérer la première lettre d'une série pour ensuite dans une nouvelle variable de type objet assigner la lettre du groupe à la propriété letter et par la même occasion déclarer une propriété series qui sera un tableau.

Enfin pour terminer, une variable currentSeries est assignée au tableau des séries d'un groupe de lettre.

Dans cette variable on ajoutera toutes les séries commençant par la même lettre, puis chaque groupe de série est ajouté à groupedSeries qui garde en mémoire les tableaux de série groupé pour les afficher.

L’affichage en HTML est moins complexe car il s’agit ici de faire deux boucles :

```
<ion-item-group *ngFor="let group of groupedSeries">  
  <ion-item-divider color="light">{{group.letter}}</ion-item-divider>  
  <ion-item-sliding *ngFor="let serie of group.series">
```

Toujours grâce à *ngFor on boucle sur groupedSeries afin d’en extraire les groupes et d’afficher la lettre de chaque groupe. Ensuite c’est sur les groupes qu’est effectué une boucle afin d’en extraire les séries et pouvoir les afficher.

Two-Way Data Biding et formulaire

Le two-way biding est une fonctionnalité puissante d’Angular gardé par Ionic.

Grâce à cette fonctionnalité, il est d’autant plus facile de valider des formulaires ou de gérer des événements.

Dans mon cas j’ai principalement utilisé cette fonctionnalité pour enregistrer les données saisies par l’utilisateur.

Avant d’en dire plus, un rappel sur le two-way biding :

Dans la partie Controller nous avons une variable nommée hello et dans la partie Vue sur un champ input une directive ngModel utilisé de la façon suivante : [(ngModel)]

Cet encapsulage est surnommé en anglais « Banana in a box », « Box of bananas » ou encore « banana-box ».

Il suffit d’indiquer le nom de la variable à assigner à cette directive :

```
[(ngModel)]= 'hello'
```

Ainsi, chaque modification dans la Vue entrainera immédiatement la modification dans le Controller.

Il n’est plus forcément nécessaire de passer des paramètres à une fonction, il suffirait de déclarer toutes les propriétés puis de les utiliser dans les actions du Controller.

Cette introduction à la directive ngModel me permet de passer à la suite :

Formulaire d'ajout et d'édition de données

L'application utilise plusieurs formulaires : - ajoute et modification de séries – ajout et modification de mangas.

Tous les formulaires sont affichés grâce à un composant Modal, ce dernier affiche une surcouche à l'écran actuel.

Pour afficher cette surcouche, le composant a besoin de connaître la page et optionnellement les données à envoyer à cette page.

Exemple d'utilisation du composant Modal :

```
editSerie(serie) {  
    let modal = this.modalCtrl.create(UpdateSeriePage, {  
    serie: serie});  
    modal.present();  
}
```

Dans cet exemple je crée un modal avec comme destination la page d'édition d'une série (UpdateSeriePage) puis je passe en paramètre la série que je souhaite envoyer à cette page (serie : serie). Pour terminer j'affiche la sur couche en utilisant la fonction present().

Les données sont ensuite récupérées dans la page de destination via le composant navParameter fournit par Ionic :

```
this.serie = this.navParams.get('serie');
```

À partir de là et grâce au two-way data biding, je peux déclarer des propriétés globales à la page d'édition pour chaque information que l'utilisateur pourra modifier.

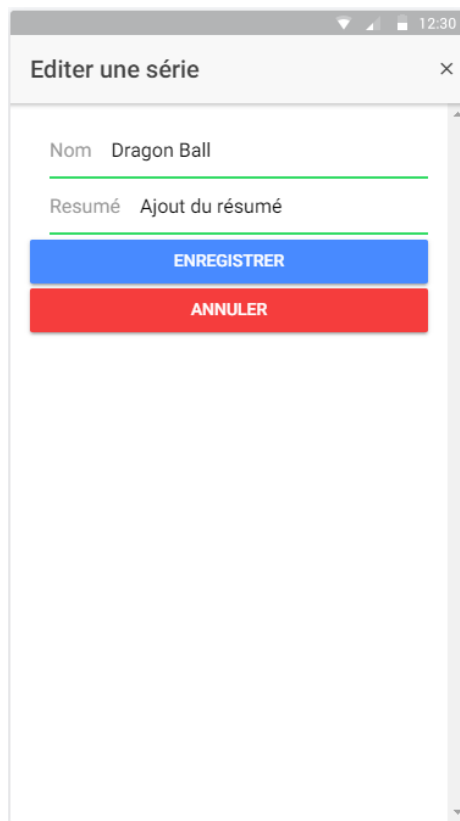
```
private nom: string = "";  
private resume: string = "";
```

Ainsi dans le constructeur du Controller je peux définir les valeurs de ces propriétés :

```
this.nom = this.serie.GetNom();  
this.resume = this.serie.GetResume();
```

De la automatiquement les informations seront affichées dans la partie Vue de l'écran via la directive ngModel :

```
<ion-item>
  <ion-label>Nom</ion-label>
  <ion-input type="text" [(ngModel)]="nom" name="nom"></ion-input>
</ion-item>
<ion-item>
  <ion-label>Resumé</ion-label>
  <ion-input type="text" [(ngModel)]="resume" name="resume"></ion-
input>
</ion-item>
```

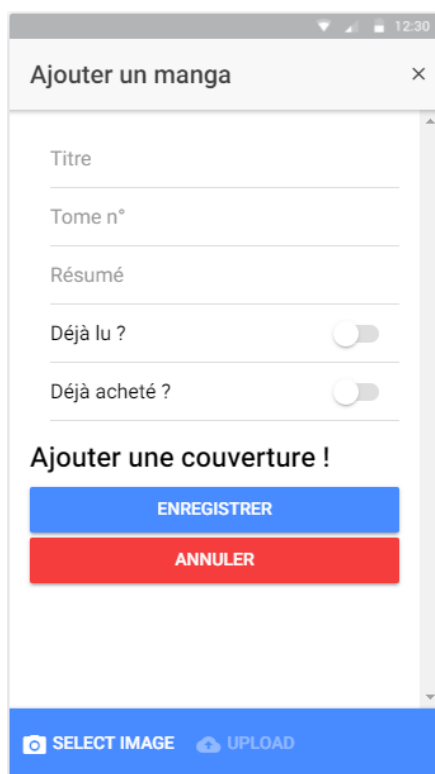


Chaque surcouche affichée via Modal dispose d'un bouton pour être refermé. Ainsi sur cette capture d'écran, le bouton situé en haut à droite en forme de croix fait appel à la fonction dismiss de Modal :

```
dismiss() {
  this.viewCtrl.dismiss();
}
```

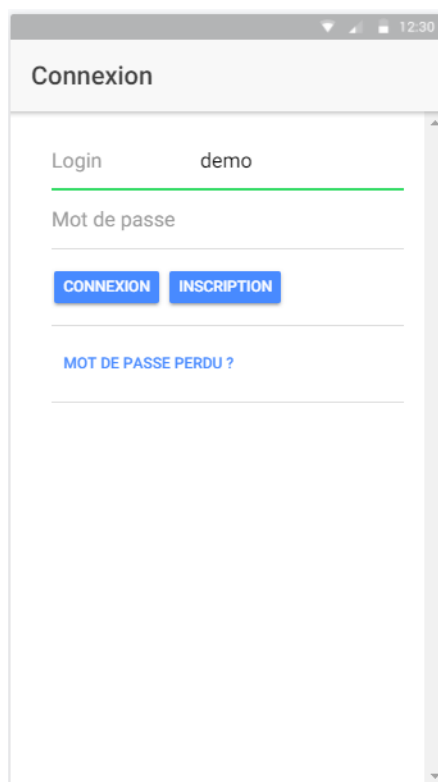
L'utilisateur se retrouve suite à la fermeture de la sur couche sur la page précédente.

Voici quelques écrans de l'application :



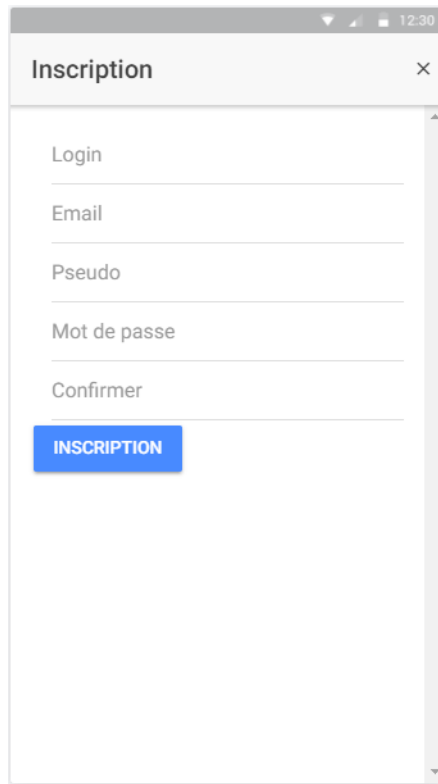
The screenshot shows a mobile application interface for adding a manga. The title bar at the top is 'Ajouter un manga' with a close button (X) on the right. Below the title bar, there are several input fields: 'Titre', 'Tome n°', and 'Résumé'. Below these fields are two toggle switches: 'Déjà lu ?' and 'Déjà acheté ?'. Below the toggles, there is a section titled 'Ajouter une couverture !' followed by two buttons: 'ENREGISTRER' (blue) and 'ANNULER' (red). At the bottom of the screen, there is a blue bar with two icons: a camera icon labeled 'SELECT IMAGE' and a cloud upload icon labeled 'UPLOAD'.

Possibilité d'ajout une couverture aux mangas depuis l'album du téléphone ou depuis la caméra de l'appareil.



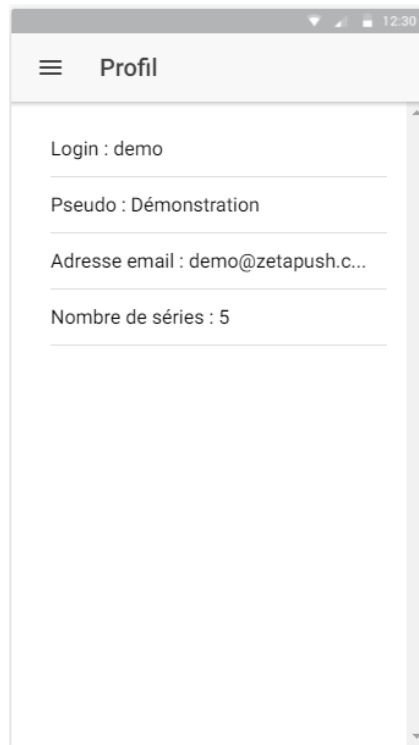
The screenshot shows a mobile application interface for the login screen. The title bar at the top is 'Connexion'. Below the title bar, there are two input fields: 'Login' and 'Mot de passe'. Below the 'Login' field, there is a green line and the text 'demo'. Below the 'Mot de passe' field, there are two buttons: 'CONNEXION' (blue) and 'INSCRIPTION' (blue). Below the buttons, there is a link labeled 'MOT DE PASSE PERDU ?'.

Ecran de connexion. Si l'utilisateur c'est déjà connecté, l'application enregistre via LocalStorage le login afin de remplir le champ login automatiquement.

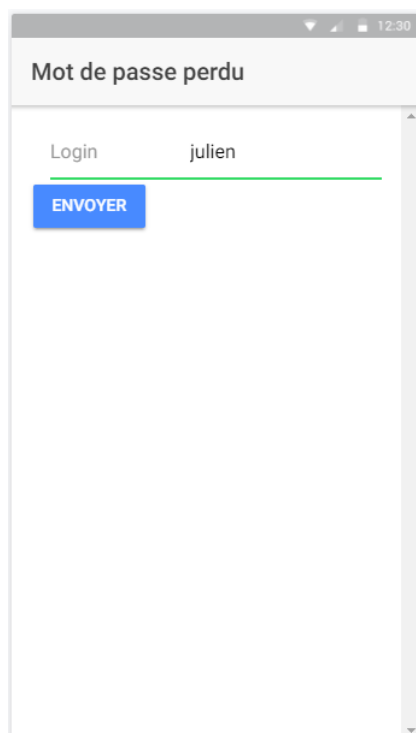


The screenshot shows a mobile application interface for registration. At the top, there is a header bar with the title 'Inscription' and a close button 'X'. Below the header, there are five input fields with labels: 'Login', 'Email', 'Pseudo', 'Mot de passe', and 'Confirmer'. At the bottom of the form, there is a blue button with the text 'INSCRIPTION' in white capital letters. The background of the screen is white, and the input fields have a light gray border.

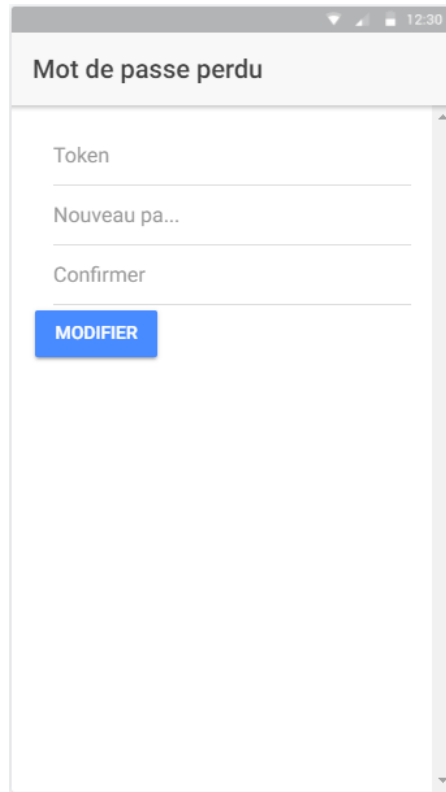
Ecran d'Inscription avec les informations requise.



Ecran Profil utilisateur.



En cas de mot de passe perdu, l'utilisateur peut grâce à son login effectuer un changement de mot de passe.



The screenshot displays a mobile application interface for password recovery. At the top, a status bar shows the time as 12:30. Below it, a header bar contains the title 'Mot de passe perdu'. The main content area features three input fields: 'Token', 'Nouveau pa...' (New password), and 'Confirmer' (Confirm). A blue button labeled 'MODIFIER' (Modify) is positioned below the 'Confirmer' field. The interface is clean and uses a light gray color scheme.

Une fois la demande de changement de mot de passe validé, un email est envoyé à l'utilisateur contenant un token. En possession du token l'utilisateur peut changer le mot de passe du compte.

Rédaction d'un tutoriel sur la Mangathèque pour Developpez.com

La rédaction d'un tutoriel c'est déroulé en plusieurs étapes.

Il m'a fallu dans un premier temps rédiger le tutoriel. En attendant de disposer des accès aux forums de rédaction de developpez.com et voulant un support de rédaction se rapprochant un maximum du rendu final, j'ai opté pour la rédaction sur un blog WordPress que j'ai mis en place avec plusieurs plugins :

- Sommaire automatique
- Highlight du code

Dans un second temps, j'ai dû faire la demande d'accès aux forums privés de rédaction au responsable de la section développement web. Grâce à cet accès j'ai pu télécharger et installer les différents outils de rédactions de tutoriel de developpez.com.

Puis une fois le tutoriel rédigé, le déposer en relecture technique et orthographique pour pouvoir ensuite le publier.

Rédaction au format Developpez.com

Developpez.com utilise un plugin disponible sur les principaux éditeurs de texte libre ou payant afin de donner accès à différents outils lors de la rédaction.

De base, les tutoriels étaient rédigés au format XML, ces outils permettent de passer du format docx, odt, etc. au format XML.

Le plugin donne accès à un modèle sur l'éditeur de texte, modèle qu'il faudra scrupuleusement respecter pour ne pas avoir d'erreurs lors de la génération du document XML.

Une fois la rédaction du tutoriel effectué, il faut le déposer en relecture technique.

Cette étape a demandé du temps, car le seul expert en ZMS que connaît le site n'est autre que mon tuteur M. Mikaël Morvan. C'est d'ailleurs M. Malick Seck, Community manager qui l'a contacté pour lui demander de faire la relecture technique.

M. Morvan a pris le travail très au sérieux et a respecté la procédure à la lettre. C'est ainsi qu'après deux corrections et relectures, j'ai déposé le tutoriel sur le forum de relecture orthographique.

Entre temps j'ai reçu les accès à mon compte FTP pour pouvoir gérer mon espace professionnel sur le site.

J'ai mis en place une page d'accueil afin de présenter les différents tutoriels.

Une fois la relecture orthographique validée, j'ai utilisé le kit de génération de developpez.com pour passer du format odt à une page HTML, des fichiers pour liseuse, fichier ZIP ou encore PDF.

J'ai pu transférer tous les fichiers sur mon espace personnel et le rendre disponible en ligne à l'adresse suivante :

<http://julien-bertacco.developpez.com/>

Afin de rendre plus attractif le tutoriel, j'ai réalisé un vidéo de l'application sur son utilisation pour la mettre en introduction.

Réalisation de tutoriels en anglais sur des services de ZetaPush

En parallèle des tutoriels sur l'application Mangathèque, j'ai rédigé une série de tutoriels en anglais sur le fonctionnement de services.

Pour se faire, chaque service serait mis en place sur une application Ionic, sans pour autant entrer dans les détails de la partie Ionic. N'ayant pas d'accès aux supports interne, les tutoriels seraient disponibles avec les sources sur mon compte GitHub en attendant de savoir ou les mettre à disposition.

Partant de ce principe, j'ai réalisé un tutoriel sur l'utilisation de :

- NoSQL
- Upload de fichier avec HDFS
- Real time chat avec Data Stack

Tous, sont disponibles à l'adresse suivante :

<https://github.com/high54>

Conclusion Ionic 3

Ionic 3 est un formidable Framework pour réaliser des applications mobiles en utilisant les dernières technologies du web.

Les composants et son patrimoine hérités d'Angular le rendent très puissant.

J'ai adoré travailler sur ce Framework et d'ailleurs mon regard se porte vers Electron qui permet de réaliser des applications desktop de la même manière en utilisant les technologies web.

De plus cette expérience m'a permis de redécouvrir JavaScript à travers TypeScript et j'ai aimé le fait de pouvoir retrouver une syntaxe orientée objet. Il faut cependant faire attention, car le Framework a encore quelques défauts. L'affichage en direct de l'application peut parfois utiliser une ancienne version de la Vue quand l'application commence à être lourde.

Certaines fonctionnalités ne sont pas encore au point, comme par exemple le sidemenu. Si le sidemenu est désactivé sur l'écran de login, il ne sera pas activé si l'utilisateur s'inscrit. Il faudra relancer l'application.

Conclusion plateforme ZetaPush

N'étant pas un développeur chevronné j'ai mis du temps pour me rendre compte de tout le potentiel que peut offrir ZetaPush. En effet, je n'avais pas de documentations, et même une recherche Google ne m'aurait rien rapporté.

Cependant, j'ai trouvé une oreille à l'écoute, M. Gregory Houllier, toujours disponible pour venir clarifier certains points et apporter de l'aide sur l'utilisation des services ou des MacroScripts.

À force d'utiliser les services et les macros, j'ai fini par prendre en main et voir à quel point il est facile de déployer et configurer un service et la flexibilité qu'offre le langage ZMS

Pour conclure je dirais que c'est un outil de développement que j'utiliserais volontiers. La possibilité de réaliser des applications sans se soucier du back-end, de serveurs, de la montée en charge et de plus le nombre de services mis à disposition font de la plateforme ZetaPush un outil prometteur.

Conclusion générale

Le stage fut pour moi une grande réussite sur le plan personnel. Je remercie vivement l'ENI de l'avoir mis en place. C'est une plongée dans le monde professionnel qui m'a permis de connaître mes limites, mes faiblesses mais aussi mes points fort. J'ai intégré une entreprise en plein développement et j'ai pu voir les besoins en temps réel.

L'expérience du stage m'a permis de conforter mon choix de reconversion professionnel et m'a d'autant plus motivé à poursuivre des études dans ce domaine qui évolue chaque jour.

L'enseignement de l'ENI m'a permis de tirer mon épingle du jeu grâce aux cours de C# car TypeScript lui est très similaire. Certainement du fait qu'ils soient tout deux développés par Microsoft.

Quand j'ai développé l'application de paris, je restais dans mon espace de confort. À savoir utiliser une base de données en faisant des requêtes SQL et afficher des informations.

C'est après, lors de la réalisation de la Mangathèque et grâce à mon maître de stage, qui souhaitait faire un tutoriel par service que j'ai commencé à maîtriser l'ensemble des services.

La plateforme ZetaPush ma tellement plu, que j'ai réalisé hors stage deux autres tutoriels sur la Mangathèque. Le premier incluant le service HDFS et le second le service Simple Authentication et Mail Sender. Et je pense continuer les tutoriels pour inclure Elastic Search et styliser l'application jusqu'à ce qu'elle soit à mon gout déployable sur le store d'Android.