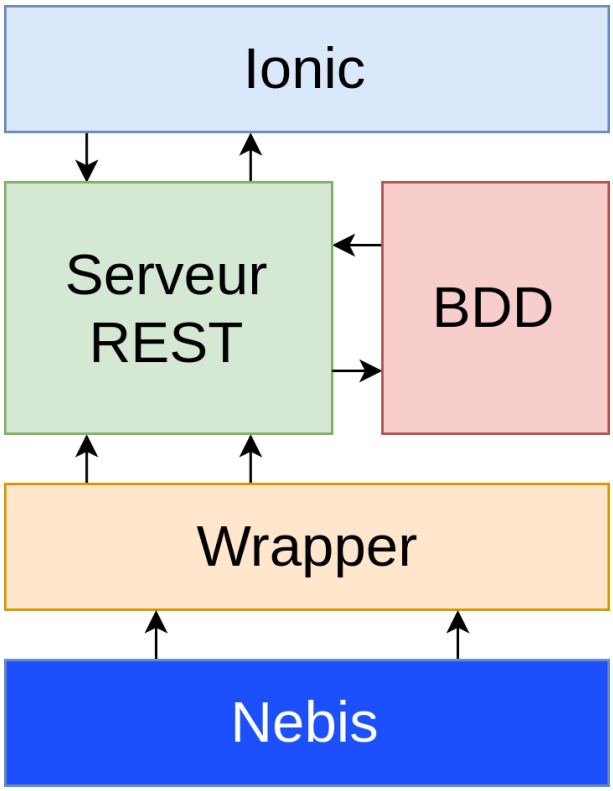
BibApp Hepia

Steven Liatti

Projet de semestre - Prof. Mickaël Hoerdt

Hepia ITI 3ème année

11 mars 2018



h e p i a
Haute école du paysage, d'ingénierie et d'architecture de Genève

Hes·SO GENÈVE
Haute Ecole Spécialisée de Suisse occidentale

Table des matières

Table des figures

Table des listings de code source

1 Introduction

1.1 BibApp V1

1.2 User stories

1.3 Besoins de la bibliothèque

La bibliothèque de l'hepia aimerait attirer plus de monde en créant du contenu personnalisé autour de son catalogue d'ouvrages et revues par le biais d'un site/application mobile. Voici les besoins listés plus précisement :

— Résumé d'un nouveau livre arrivé à la bibliothèque (nouveautés)

— Coups de coeur des bibliothécaires sur les ouvrages présents

— Revues de presse des périodiques

— Ajouter les images des couvertures scannées par la bibliothèque

Toutes ces actions doivent pouvoir être réalisées via le site web en mode administrateur. Les opérations de création, récupération, modification et suppression de contenu (CRUD) sont nécessaires. Un mode utilisateur, ou visiteur, permet de consulter le contenu, depuis un ordinateur ou un appareil mobile. L'idée est de créer une source de contenus basée sur le catalogue Nebis et augmentée par les ajouts des bibliothécaires.

2 Technologies utilisées

Pour un aperçu rapide de Typescript et Angular, voir cet article de Josh Morony [5].

2.1 Typescript

Ionic fait usage de Typescript, une surcouche à Javascript, offrant des types vérifés à la "compilation" (car Typescript est traduit, "transpiled", vers du Javascript conventionnel) et non à l'exécution, signalant les erreurs sur les types et offrant ainsi plus de rigueur à l'écriture du code [6].

```
function add(x : number, y : number) : number {
    return x + y;
}
add('a', 'b'); // compiler error

// Class example :
class Greeter {
    greeting: string;
    constructor (message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}
```

Listing 1 – Syntaxe Typescript

La structure basique des fichiers *composants* Typescript avec Ionic ou Angular est semblable à ceci :

```typescript
import { Component } from '@angular/core';

@Component({
  selector: 'my-app'
})
export class AppComponent {
  title = 'My App';
  private field: string;

  constructor(public param: string) {
    this.field = param;
  }
}
```

Listing 2 – Exemple d'une classe Typescript sous Ionic ou Angular

Explications : on importe le composant `Component`, on définit le sélecteur utilisé dans le HTML pour faire le rendu et on définit notre classe `AppComponent` qui pourra être exportée dans d'autres modules. Dans cette classe nous définissons deux attributs `title` et `field` et un constructeur. Typescript amène également une notion de visibilité des attributs et méthodes d'une classe, comme en Java.

2.2 Angular 4

Angular [7], [8] est un framework front-end Javascript développé par Google. Je ne l'ai pas directement utilisé dans mon travail mais Ionic est construit sur Angular et partage de nombreux concepts et fonctionnalités avec lui. Angular impose une architecture de modules, où, pour chaque module, sont définis des fichiers HTML pour la structure (avec une syntaxe ajoutée, voir plus loin), des fichiers CSS (ou autres préprocesseurs CSS comme Sass) et des fichiers *composant* écrits en Typescript, gérant la logique "métier".

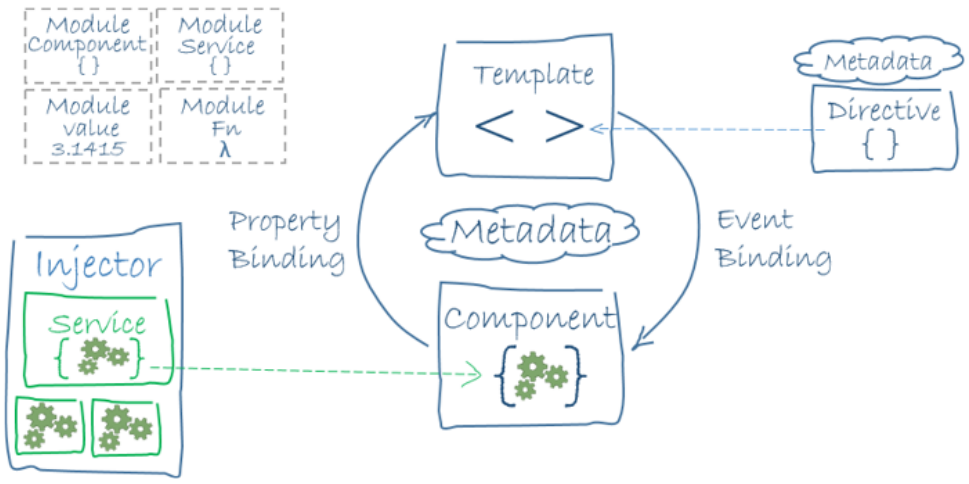


FIGURE 1 – Architecture d'un composant Angular - Angular [1]

Comme on peut apercevoir sur la figure ci-dessus, le template HTML intéragit avec son composant Typescript. Un composant Angular est une simple classe Typescript, possédant attributs et méthodes. Un template représente le combo page HTML et CSS, la vue du module, en interaction avec l'utilisateur. Selon les actions de l'utilisateur, la vue ou le modèle (données) est mis à jour. On peut accéder aux attributs et méthodes du composant depuis le template. Un ou plusieurs services peuvent être utilisés ("injectés") dans un module. On peut voir un service comme un composant réutilisable et ne possédant pas (forcément) de template (exemples : services d'authentification, de gestion d'images, etc.).

Voici une brève description de balises et directives HTML ajoutées par Angular :

```
<input [value]="firstName">
<button (click)="someFunction(event)">
<p>Hi, {{ name }} {{ 1 + 1 }}</p>
<input [(ngModel)]="name">
<p #myParagraph></p>
<section *ngIf="showSection">
<li *ngFor="let item of items">
```

Listing 3 – Syntaxe HTML avec Angular

Descriptif ligne par ligne :

1. Change la propriété `value` en lui attribuant la vraie valeur de l'attribut de classe firstName (définit dans une classe Typescript)
2. Appelle la fonction `someFunction()` en lui passant l'événement `event` au moment du clic sur le bouton
3. Évalue les expressions entre et les affiche, ici en l'occurence un attribut `name` et le calcul de 1 + 1, 2
4. Applique la valeur de `name` à l'input, mais s'il y a changement de la part de l'utilisateur, met à jour l'objet associé
5. Crée une variable locale au template HTML
6. `*ngIf` : supprime l'élément du DOM (ici `section`) si la condition n'est pas remplie
7. `*ngFor` : boucle sur un tableau et répète l'élément du DOM

2.3 Ionic

Ionic est un framework, basé sur Angular, permettant de créer des applications écrites avec les langages du web (HTML, CSS et Javascript/Typescript) afin de générer des applications à destination de multiples plateformes, telles qu'Android, iOS ou Windows Phone. Ces applications ainsi générées sont dites "hybrides" car elles s'exécutent dans une *WebView*, une instance de navigateur du device de destination, à la manière d'un site web classique, mais elles ont néanmoins la possibilité d'accéder aux API du système hôte de manière native, en passant par des "wrapper". Celui utilisé par Ionic est Apache Cordova. Par conséquent, le développeur n'a qu'à écrire le code qu'une seule fois pour les différentes plateformes mobiles visées. En résumé, Ionic est un "Angular++", offrant des composants graphiques prédéfinis, des thèmes CSS et un accès aux API natives via Cordova (voir le tutoriel sur Ionic à la section 3 pour plus d'informations, voir cette vidéo de 1h15 pour apprendre les bases par la pratique [9]).

2.4 Promise Javascript

Une Promise ("promesse" en français) est un objet Javascript qui s'utilise dans des opérations asynchrones. Elle représente l'état de cette opération, qui peut réussir ou échouer. Elle peut être dans un de ces 4 états :

— pending (en attente) : état initial, la promesse n'est ni remplie, ni rompue

— fulfilled (tenue) : l'opération a réussi

— rejected (rompue) : l'opération a échoué

— settled (acquittée) : la promesse est tenue ou rompue mais elle n'est plus en attente

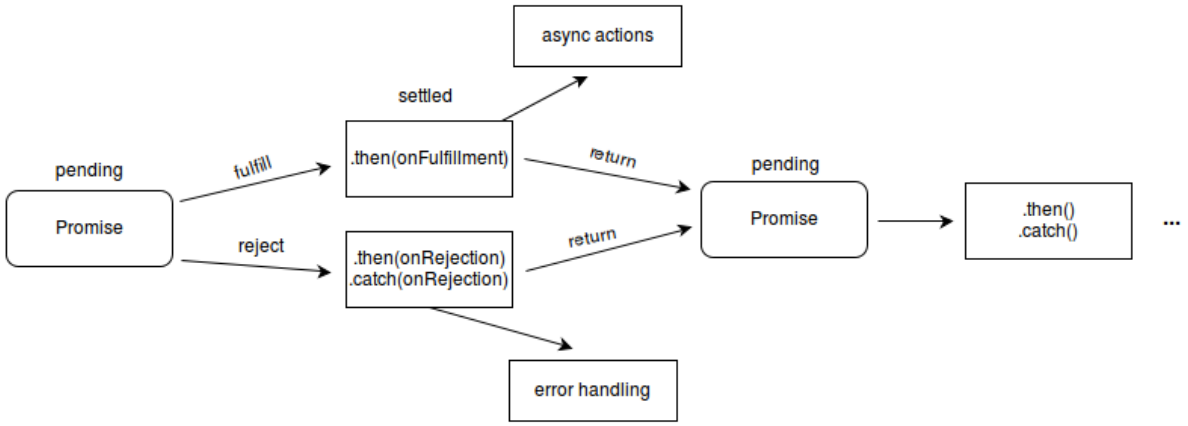


FIGURE 2 – États d'une Promise - MDN Web Docs [2]

Les promesses sont particulièrement utiles dans les cas de requêtes vers des ressources distantes où le temps de réponse et la réussite de l'appel ne sont pas prévisibles. Ainsi, les promesses remplacent les classiques *callbacks* par un moyen élégant d'enchainer les opérations asynchrones :

```
doSomething(function(result) {
  doSomethingElse(result, function(newResult) {
    doThirdThing(newResult, function(finalResult) {
      console.log('Got the final result: ' + finalResult);
    }, failureCallback);
  }, failureCallback);
}, failureCallback);
```

Listing 4 – "Callback hell" - MDN Web Docs [3]

Cet imbriquement de *callbacks* se transforme en chainage de promesses :

```
doSomething().then(function(result) {
  return doSomethingElse(result);
})
.then(function(newResult) {
  return doThirdThing(newResult);
})
.then(function(finalResult) {
  console.log('Got the final result: ' + finalResult);
})
.catch(failureCallback);
```

Listing 5 – Réécriture avec les Promise - MDN Web Docs [3]

Ce mécanisme de programmation va être très utilisé dans le Wrapper et Serveur REST. *Pour approfondir, voir ces deux ressources sur le MDN Web Docs [2] et [3].*

2.5 Node.js

Node.js est un environnement d'exécution de code Javascript côté serveur. Il fait office entre autres de serveur HTTP. Node.js est basé sur une architecture orientée événements. Son moteur d'exécution Javascript est celui de Google Chrome, V8. Le fonctionnement de Node est non bloquant, il délègue les opérations I/O à son OS hôte, pour pouvoir continuer à servir les requêtes entrantes.

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Listing 6 – Hello World avec Node.js - Node.js [4]

2.6 MongoDB

2.7 JSON Web Tokens

3 Tutoriel sur Ionic

Ce tutoriel est réalisé avec la version 3.x.x de Ionic avec une machine sous Linux. Je me suis basé sur le tutoriel sur le site de Ionic [10] et sur les tutoriels de Josh Morony [11], [12].

3.1 Installation

Tout d'abord, Ionic nécessite Node.js et `npm`, son gestionnaire de paquets/dépendances Javascript. Une fois `npm` installé, il faut entrer la commande suivante dans un terminal :

```
npm install -g ionic cordova
```

Listing 7 – Installation de Ionic et Cordova

Cela installe Ionic et Cordova, l'outil permettant de traduire une web app à base de HTML, CSS et Javascript en application hybride (moitié native, moitié web) pour la plateforme choisie (Android, iOS, etc.). Pour créer un nouveau projet, il faut alors faire ceci :

```
ionic start myapp
cd myapp
ionic serve
```

Listing 8 – Initialisation d'un projet Ionic

La première commande crée l'arborescence de base d'un projet Ionic se nommant "myapp". Un menu interactif apparait avec plusieurs choix de template pour notre application : vide, avec des pages déjà intégrées ou un projet complet avec des pages et providers correspondant aux bonnes pratiques Ionic. La dernière commande compile les sources Ionic dans le sous-dossier `www` et lance un serveur web de développement en écoute sur `http://localhost:8100` qui compile à nouveau le projet à chaque changement dans le code.

3.2 Ionic CLI

Ionic fournit une interface en ligne de commande bien pratique. Elle permet de générer un projet avec choix de templates prédéfinis (`ionic start`), de build l'application (`ionic build`), de lancer l'application en mode développement avec `ionic serve` et, commandes les plus pratiques, générer pages et providers entre autres (`ionic g [page|provider] NewGenerate`). Un sous-ensemble de commandes avec le préfix `ionic cordova` permet de build et lancer l'application Ionic sur des devices Android et iOS. Pour retrouver une aide sur toutes les commandes disponibles, entrez simplement `ionic` dans un terminal.

3.3 Arborescence

Voici l'arborescence racine pour un projet Ionic, après avoir ajouté les plateformes désirées :

```
.
|-- node_modules
|-- platforms
|-- plugins
|-- resources
|-- src
|-- www
|-- config.xml
|-- ionic.config.json
|-- package.json
|-- package-lock.json
|-- tsconfig.json
|-- tslint.json
```

Listing 9 – Arborescence racine

On peut apercevoir les dossier suivants :

- `node_modules` : contenant les dépendances Angular + Ionic
- `platforms` : contenant les builds et fichiers nécessaires des plateformes ajoutées avec cordova
- `plugins` : contenant les plugins Cordova
- `resources` : contenant les fichiers statiques pour Android et iOS (les icônes des applications notamment)
- `src` : contenant nos fichiers sources, c'est finalement le seul dossier qu'on utilise en développement
- `www` : contenant les fichiers construits ("build") à partir des sources, destinés à être servis par un serveur web (tel qu'Apache)

Ensuite, le détail du dossier `src` :

```
.
|-- app
  |-- app.component.ts
  |-- app.html
  |-- app.module.ts
  |-- app.scss
  |-- main.ts
|-- assets
  |-- icon
    |-- favicon.ico
  |-- imgs
      |-- logo.png
|-- pages
  |-- home
      |-- home.html
      |-- home.scss
      |-- home.ts
|-- theme
  |-- variables.scss
|-- index.html
|-- manifest.json
|-- service-worker.js
```

Listing 10 – Arborescence du dossier src

On peut apercevoir les dossier et fichiers suivants :

— app : contient `app.component.ts` et `app.module.ts`, le composant principal et la liste des modules de l'application

— `assets` : les fichiers statiques tels que des images

— `pages` : contient toutes les pages générées avec Ionic CLI, c'est ici que s'écrit 90% du code

— `theme` : contient `variable.scss`, un fichier sass avec les variables globales de l'application

Pour approfondir, je conseille de lire ce tutoriel [12].

3.4 Page

Une page est constituée au minimum d'un fichier html (la vue) et Typescript (la logique métier). Pour personnaliser le design de la page uniquement, il est possible d'ajouter un fichier Sass (.scss) qui sera traduit en CSS. Chaque page doit être ajoutée au fichier `app.module.ts` pour pouvoir être importée et utilisée dans d'autres pages. Dans le fichier `app.component.ts` est définie la page principale (`root page`). Les pages se retrouvent par défaut dans le dossier `src/pages`. Par défaut, chaque page possède son propre dossier à son nom.

3.5 Provider

Un provider est un fournisseur de services aux pages de notre application. Il peut être vu comme une page sans vue, donc uniquement du Typescript, de la logique métier. Les providers

servent généralement à fournir une API interne pour manipuler des données, qu'elles soient distantes ou locales. Il se retrouvent par défaut dans le dossier `src/providers`.

3.6 Navigation

Ionic utilise un système de pile pour la navigation entre ses pages. Selon si la page suivante a une relation semblable à celle "parent-enfant", ça vaut la peine de "push" la nouvelle page sur la première, pour facilement y revenir : par exemple, une liste d'articles, avec pour chaque article la possibilité de naviguer vers lui, il semble naturel de pouvoir facilement revenir à la liste des articles. Au contraire, si on change de section sur notre application ou si les deux pages n'ont pas de lien direct entre elles, il vaut mieux changer la `root page`, autrement dit, la "page racine". Une page parente a la possibilité de passer des paramètres à la page enfant. Tous ces mécanismes sont accessibles depuis le composant `NavController` [13].

3.7 Composants disponibles

Ionic fournit de nombreux composants graphiques prêts à l'emploi pour l'interface utilisateur. La liste complète est disponible sur https://ionicframework.com/docs/components/. On y trouve par exemple des boutons prédéfinis mais personnalisables, des boutons flottants, des gestes (surtout pour le tactile), des listes, des modales, des menus, une barre de recherche, des onglets, etc.

3.8 Design

Ionic offre des feuilles de style Sass et CSS préconçues. Une disposition des éléments selon une grille (à la manière de Bootstrap) est disponible. Toutes les variables Sass sont modifiables dans le fichier `variables.scss`.

3.9 Service HTTP

Angular fournit un service HTTP, utilisable dans Ionic, pour faire des requêtes asynchrones. Exemple ici d'une requête `GET`. La fonction `map()` applique pour chaque élément des données reçues (un tableau par exemple) la fonction passée en argument et retourne un `Observable`. Ici on parse les données JSON reçues. Ensuite, dans `subscribe()`, trois cas de figure :

- On accède aux données lorsqu'elles sont "prêtes"
- Si la requête a échoué, on affiche un message d'erreur (dans le cas présent)
- Enfin, on exécute les instructions dans tous les cas de figure

```
this.http.get('http://example.com/api')
.map(res => res.json())
.subscribe(
  data => {
    // process data
  },
  err => {
    console.log("Error : ", err);
  },
  () => {
    // finally block
  }
);
```

Listing 11 – Requête HTTP avec Ionic

3.10 Service Storage

Ionic fournit une méthode simple pour stocker des données sour forme de paires clé/valeur sur le client local, que ce soit dans le navigateur ou dans l'application mobile. Une utilisation possible est de déclarer une classe qui fait appel à `storage` de la manière suivante [14] :

```
import { Storage } from '@ionic/storage';
import { Injectable } from '@angular/core';

@Injectable()
export class DataProvider {

  constructor(private storage: Storage) {}

  getPairs() {
    let pairs = [];
    this.storage.forEach((v, k) => {
      comments.push({key: k, value: v});
    });
    return pairs;
  }

  get(key: string) {
    return this.storage.get(key);
  }

  set(key: string, data: string) {
    this.storage.set(key, data);
  }

}
```

Listing 12 – Storage avec Ionic

3.11 Déploiement

Grâce une fois de plus à ls CLI Ionic, le déploiement d'une application Ionic est facilité. Pour plus d'informations, voir cette page.

3.11.1 Browser

Pour un déploiement à destination des navigateurs web, il faut se placer dans le dossier de l'application et, dans un terminal, entrer `ionic build`. Ceci va compiler les fichiers Typescript, Sass et HTML dans le dossier `www`. Ce dossier peut ensuite facilement être ajouté à un serveur HTTP tel qu'Apache.

3.11.2 Android

Pour déployer sur un Android device, il faut au préalable avoir installé Java JDK et Android Studio avec le SDK à jour. Il faut ensuite, dans un terminal, se positionner dans le dossier Ionic de l'application et lancer les commandes suivantes, pour build ou pour run l'application sur un device connecté :

```
ionic cordova build android --device
# ou
ionic cordova run android --device
```

Listing 13 – Déploiement sur Android

Astuce : s'assurer que le path du SDK Android est bien configuré dans Cordova [15].

3.11.3 iOS

Bien que non testée, la procédure est similaire à Android. Il faut tout d'abord disposer d'un Mac avec Xcode 7 ou supérieur, un device avec iOS 9 ou plus et un Apple ID. Ensuite dans un terminal, se positionner dans le dossier Ionic de l'application et lancer les commandes suivantes, pour build ou pour run l'application sur un device connecté :

```
ionic cordova build ios --device
# ou
ionic cordova run ios --device
```

Listing 14 – Déploiement sur iOS

4 Architecture

Actuellement, le catalogue Nebis [16] est utilisé par la bibliothèque. Cependant, il sera abandonné dans quelques années pour un nouveau catalogue, encore inconnu à ce jour. L'objectif étant de réaliser une web app pérenne, il faut pouvoir s'adapter à ce changement. Voici ci-dessous l'architecture globale de l'application :

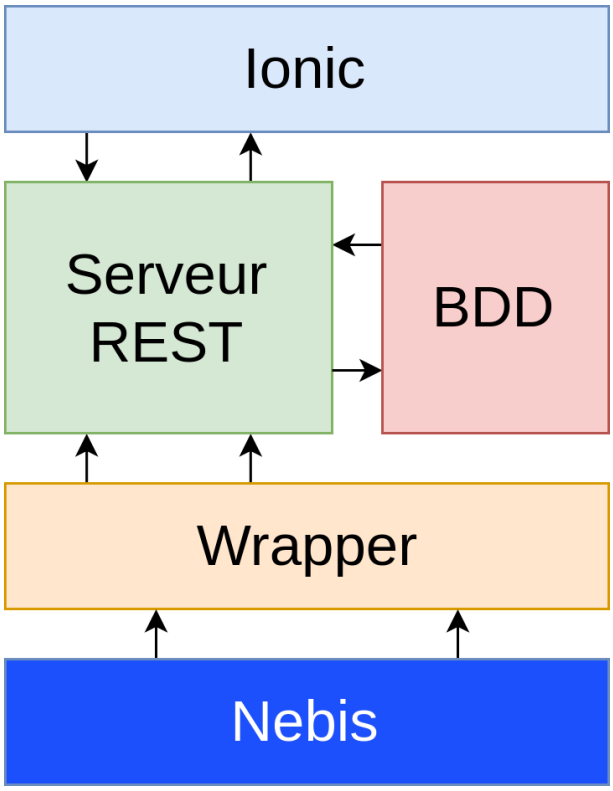


FIGURE 3 – Architecture globale de la web app

Le Wrapper et le serveur REST seront faits en Node.js. La base de données sera faite avec MongoDB.

4.1 Wrapper

Ce module fait le pont entre les données issues d'un catalogue et le serveur REST. Son utilité principale est de s'adapter au catalogue utilisé : si le catalogue est amené à changer ou à disparaître au profit d'un autre, il suffira de modifier ce Wrapper pour continuer à faire fonctionner l'application. Il devra au minimum fournir :

— La liste des nouveautés

— Les informations de base sur les ouvrages

— La recherche d'une oeuvre par ISBN et/ou d'autres critères

4.2 Serveur REST

Ce serveur offre les services suivants :

— CRUD pour les infos de base des oeuvres (Nebis)

— CRUD pour les résumés/commentaires des livres

— CRUD pour les coups de coeur des bibliothécaires

— CRUD pour les revues de presse

— CRUD pour les images scannées par les bibliothécaires

— Authentification et autorisations des utilisateurs

4.3 Base de données augmentée

La base de données sera liée au serveur REST, elle enregistrera le contenu produit par la bibliothèque.

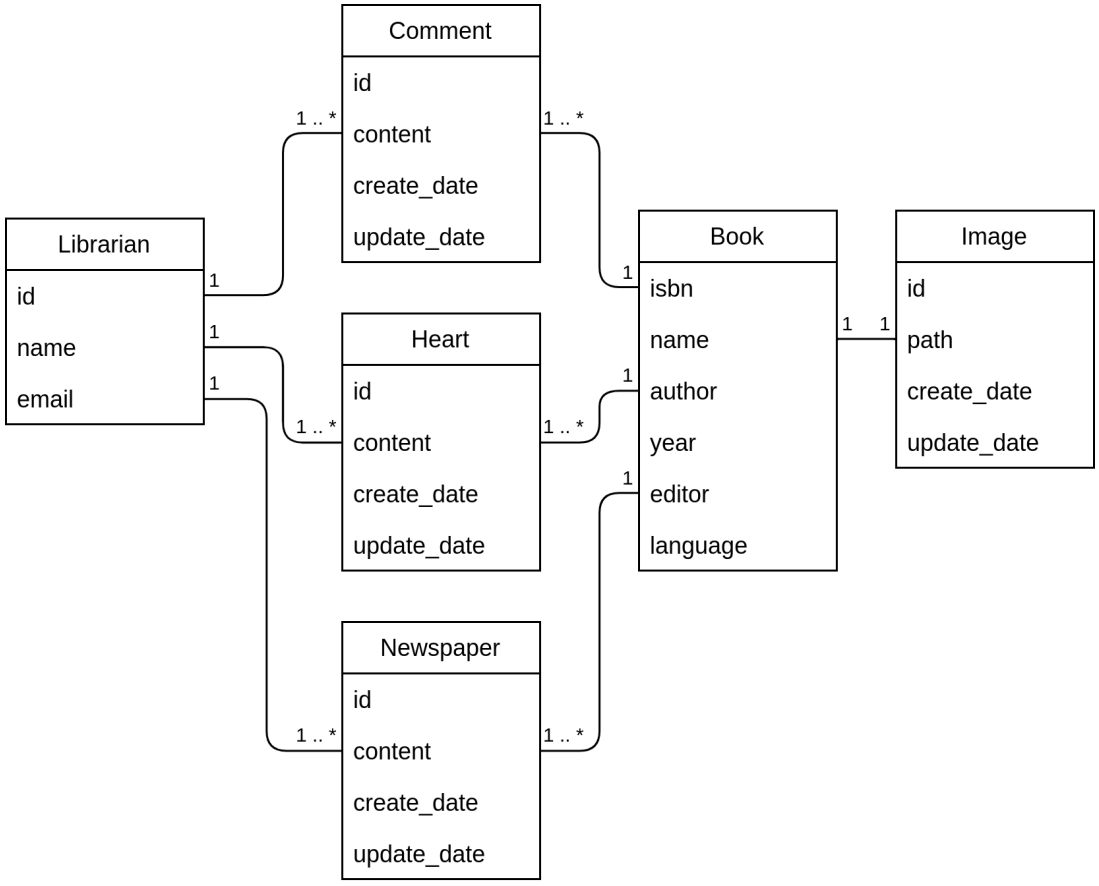


FIGURE 4 – Schéma de la base de données

(Ce schéma est amené à évoluer)

4.4 Ionic

Partie front-end de l'application, elle offrira côté utilisateur :

- Page d'accueil, avec les sections "Nouveautés", "Coups de coeur" et "Revues de presse"
- Pour chaque section, une page listant les ouvrages ou périodiques avec infos de base (Titre, auteur, etc. et image si fournie)
- Pour chaque entrée, la possibilité de cliquer dessus et consulter les infos Nebis et le contenu enrichi

Côté administrateur (ou rédacteur), les bibliothécaires pourront s'authentifier et auront une section supplémentaire, "Images", où ils pourront ajouter les scans des livres aux entrées existantes. Pour les autres sections, ils pourront ajouter le contenu correspondant aux entrées voulues.

5 Réalisation

5.1 Captures d'écran

6 Guide de déploiement

Pour commencer, il faut cloner le repository du projet et inscrire la machine (son IP) qui hébergera le Wrapper auprès de la RIB API [16]. Ensuite, il faut installer Node.js et éventuellement MongoDB si la base de données est destinée à être locale. L'installation de ces deux outils est décrite sur leurs sites respectifs. Il faut changer la configuration des fichiers `server/config/auth.js` et `server/config/db.js`. Le premier fichier contient la clé nécessaire à PassportJS pour générer les JWT. Le deuxième fichier contient l'URL de la base MongoDB. Ensuite, dans un terminal, dans les dossiers `wrapper` et `server`, entrez les commandes suivantes pour installer et lancer les deux serveurs Node.js :

```
npm install
npm start
```

Listing 15 – Déploiement du Wrapper et du Serveur

Le Wrapper et Serveur écoutent respectivement sur les ports 8081 et 8082. Après, il faut installer Ionic et Cordova : `npm install -g ionic cordova`. La prochaine étape consiste à déployer Ionic sur un serveur HTTP. Remplacez tous les appels à 'http ://bi-bapp2.infolibre.ch' par votre nom de domaine. Puis, déplacez-vous dans le dossier `ionic` et entrez `ionic build`. Entrez 'Y' lorsque demandé pour installer les `node_modules`. Cette dernière commande va générer le dossier `www` qui contiendra tous les fichiers nécessaires pour déployer la web app. Il faut ensuite placer ce dossier `www` sur son serveur HTTP. L'étape suivante consiste à ajouter un utilisateur admin à MongoDB qui pourra ajouter d'autres utilisateurs. Il faut tout d'abord créer le hash de son mot de passe avec le fichier `server/app/pass.js` et entrer `node pass.js yourPassword` dans un terminal. Connectez-vous ensuite à votre shell MongoDB (`mongo bibapp`) et ajoutez un utilisateur comme ceci :

```
db.users.insert({ email: 'testadmin@mail.com', password:
'$2a$10$swh1PhPP6vm2K7g/1KHOTeBCyOncIFgB2doubyPpQHKc8zcddUjV6', role:
'admin' })
```

Listing 16 – Ajout d'un utilisateur à MongoDB

7 Conclusion

8 Annexes

9 Références

[1] Angular. Architecture overview. https://angular.io/guide/architecture. Consulté le 10.10.2017.

[2] Mozilla Developer Network Web Docs. Promise. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise, février 2018. Consulté le 15.02.2018.

[3] Mozilla Developer Network Web Docs. Using promises. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises, mars 2018. Consulté le 15.02.2018.

[4] Node.js. About node.js. https://nodejs.org/en/about/, 2018. Consulté le 09.03.2018.

[5] Josh Morony. Intro to ecmascript 6 and angular 2 for ionic 1.x developers. https://www.joshmorony.com/intro-to-ecmascript-6-and-angular-2-for-ionic-developers/, juin 2017. Consulté le 11.10.2017.

[6] Paul Dixon. What is typescript and why would i use it in place of javascript ? https://stackoverflow.com/questions/12694530/what-is-typescript-and-why-would-i-use-it-in-place-of-javascript/12694578#12694578, octobre 2012. Consulté le 10.10.2017.

[7] Traversy Media. Angular 4 in 60 minutes. https://www.youtube.com/watch?v=KhzGSHNhnbI, juillet 2017. Consulté le 10.10.2017.

[8] Josh Morony. Ionic first look series : Angular concepts & syntax. https://www.joshmorony.com/ionic-2-first-look-series-new-angular-2-concepts-syntax/, juillet 2017. Consulté le 11.10.2017.

[9] Traversy Media. Ionic 3 mobile weather app build. https://www.youtube.com/watch?v=qs2n_poLarc, août 2017. Consulté le 10.10.2017.

[10] Ionic. Ionic tutorial. https://ionicframework.com/docs/intro/tutorial/. Consulté le 03.10.2017.

[11] Josh Morony. Beginners guide to getting started with ionic 2. https://www.joshmorony.com/beginners-guide-to-getting-started-with-ionic-2/, juillet 2017. Consulté le 05.10.2017.

[12] Josh Morony. Ionic 2 first look series : Your first ionic 2 app explained. https://www.joshmorony.com/ionic-2-first-look-series-your-first-ionic-2-app-explained/, juillet 2017. Consulté le 11.10.2017.

[13] Josh Morony. A simple guide to navigation in ionic 2. https://www.joshmorony.com/a-simple-guide-to-navigation-in-ionic-2/, juin 2017. Consulté le 17.10.2017.

[14] Ionic. Storage. https://ionicframework.com/docs/storage/. Consulté le 20.10.2017.

[15] Android platform guide. https://cordova.apache.org/docs/en/latest/guide/platforms/android/index.html. Consulté le 20.10.2017.

[16] Giuliani Germano. Resource information bus (rib) api. https://dinkum.ethbib.ethz.ch/display/RIB/Home, février 2015. Consulté le 29.12.2017.