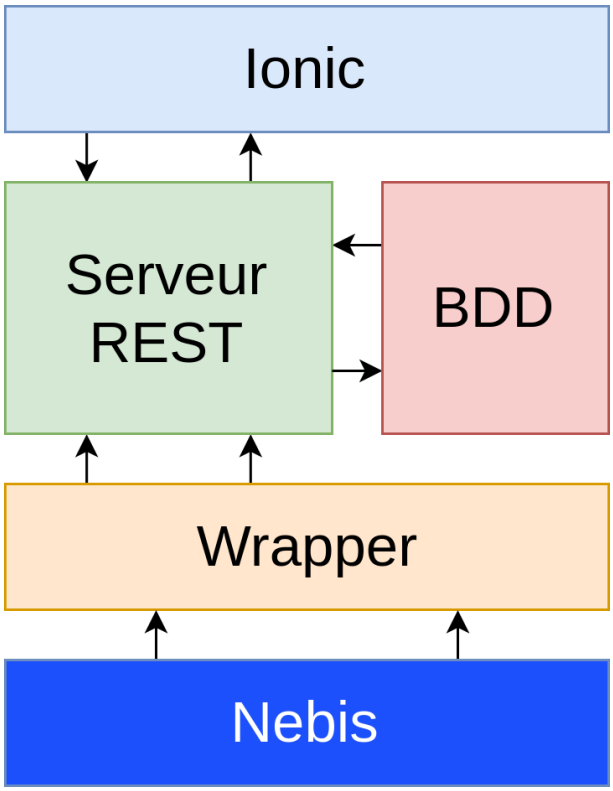
BibApp Hepia

Steven Liatti

Projet de semestre - Prof. Mickaël Hoerdt

Hepia ITI 3ème année

9 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 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

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 [2].

```
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 ([3], [4]) 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".

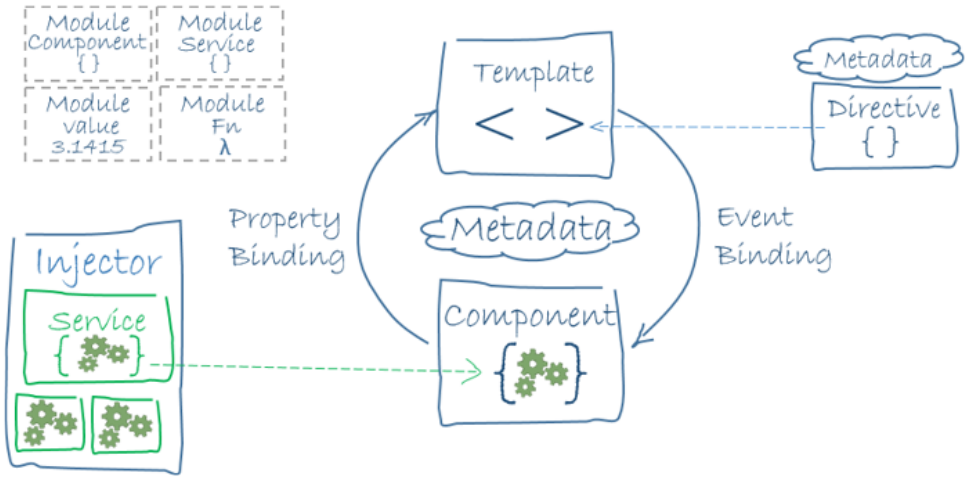


FIGURE 1 – Architecture d'un composant Angular [1]

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 :

```
1 <input [value]="firstName">
2 <button (click)="someFunction(event)">
3 <p>Hi, {{ name }} {{ 1 + 1 }}</p>
4 <input [(ngModel)]="name">
5 <p #myParagraph></p>
6 <section *ngIf="showSection">
7 <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

2.4 Promise Javascript

2.5 Node.js

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. Pour approfondir l'étude, un tutoriel sur le site de Ionic est disponible ([5]).

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 4 – 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 5 – Initialisation d'un projet Ionic

La première commande crée l'arborescence de base d'un projet Ionic se nommant "myapp". 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.

3.1 Arborescence

3.2 Ionic CLI

3.3 Page

3.4 Provider

3.5 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" ([6]).

3.6 Composant HTTP

Angular fournit un composant 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 6 – Requête HTTP avec Ionic

3.7 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 :

```
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 7 – Storage avec Ionic

3.8 Déploiement

3.8.1 Browser

3.8.2 Android

3.8.3 iOS

3.9 Composants

3.10 Thèmes

4 Architecture

Actuellement, le catalogue Nebis 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 :

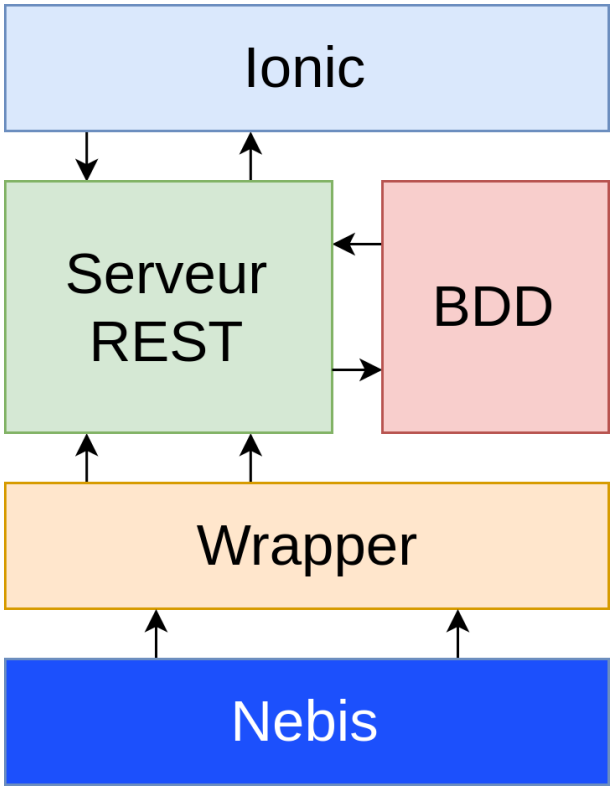


FIGURE 2 – Architecture globale de la web app

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.

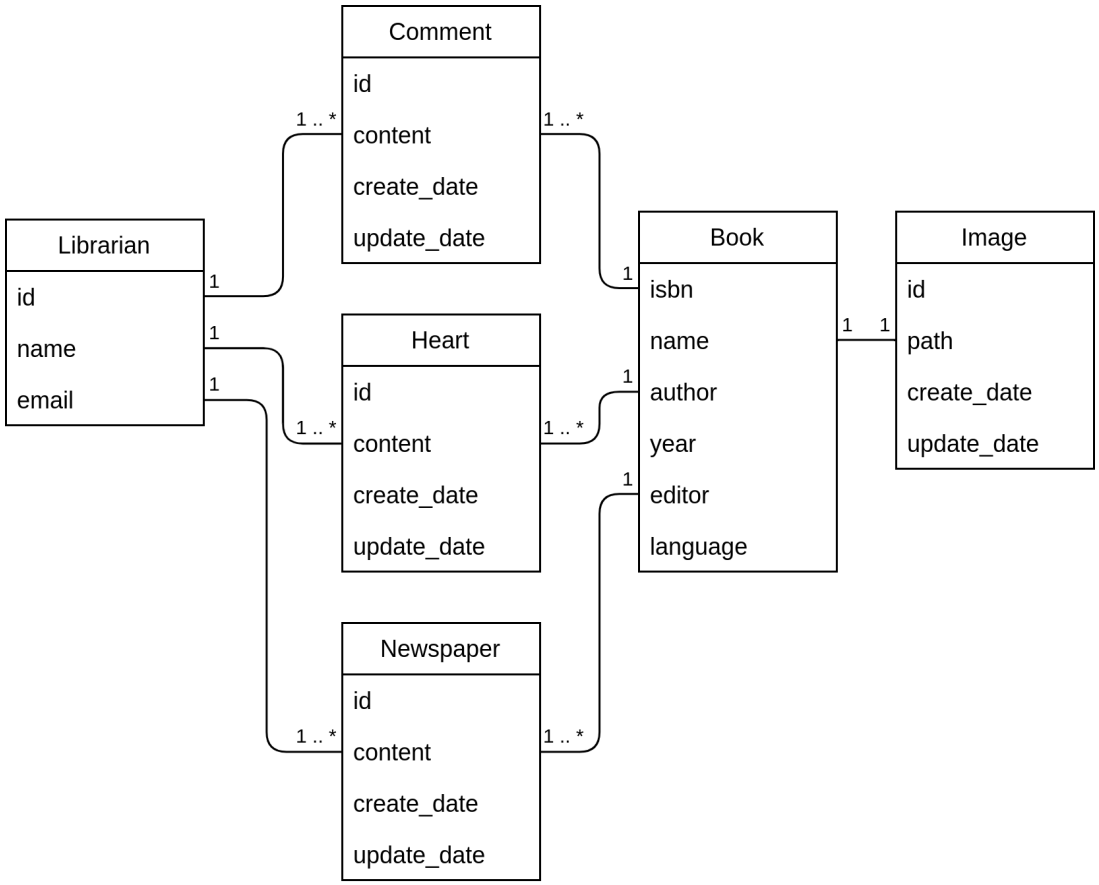


FIGURE 3 – Schéma de la base de données

(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 Conclusion

7 Annexes

8 Références

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

[2] 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.

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

[4] 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.

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

[6] 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.