Mobile Web Project Report

Laureen Schausberger

PROJECT-REPORT

submitted for

MSc Mobile Application Development

in Canterbury

January 2019

This report was created as part of the course

Mobile Web Development

during

Fall Semester 2018

Advisor:

Dr. Les Walczowski

Contents

Chapter 1

Introduction

The following chapters will give an overview of the ionic application, which was created during the Mobile Web Development module, discuss the overall design, mention integrated features and explain the main implementation details.

The goal of this project was to build a complete mobile web application, which connects to external web services using JavaScript/TypeScript. The developed app should be able to demonstrate the skills learnt in this module.

This project focuses its communication on the New York Times' Books API[1].

The requirements were:

- **Retrieve dynamic data**. Your app should allow data to be obtained from the internet using web services, for example using Discogs or MusicBrainz for a music based mini-project. Although you can select any web services you want, the services should be data rich which will allow you to design a data-driven mobile web app with a range of features. A list of available APIs is available on https://www.programmableweb.com/apis/directory
- **Functionality**. You should consider carefully what functionality you wish to build into the App, but I would suggest you need at least three distinct web pages to be produced, each of which should be data driven.
- **User experience**. Particular care should be taken in designing a web app which is easy to use, looks attractive and provides a rich user experience. The recommended framework for the development of your app is Ionic 4, however, you are free to use any frameworks you find suitable for mobile development such as Mobile jQuery.
- **Additional**. Incorporating plugins/technologies/frameworks beyond those presented in the course and/or provision of social media facilities such as Facebook or Twitter will be recognised when the work is assessed.

Please note that the screenshots presented in this report were made on an iPhoneX.

[1] https://developer.nytimes.com/books_api.json

Chapter 2

Overall Design

The app is split into 4 main tabs:

- Bestseller
- Search
- Favorites
- Randomizer

Those sections will be discussed and shown in this chapter along with the Details-View, while also explaining the overall design of the application with some screenshots.

2.1 Bestseller

The Bestseller Tab shows the user the top 15 bestsellers of the current week. Given that those bestsellers are returned by category, the user can change the default value (which is "Combined Print and E-Book Fiction") by changing it in the selection field on the top of the view. An example can be seen in figure 2.1. The figure also shows the loading indicator, that every view uses for long loading operations.

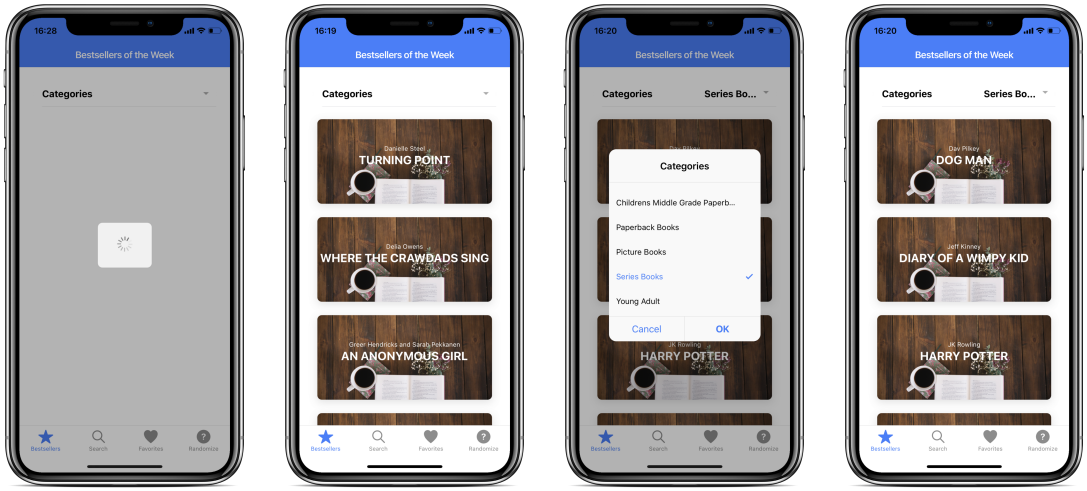

Figure 2.1: Interface Flow of Bestseller Tab when changing the Category

As shown in figure 2.1, all dynamically generated items are presented as **ion-card**s that have a picture in the background, and two white labels in the front, which present the author and the title of the book. The background image[1] stays the same, as the API unfortunately doesn't provide a picture of the books themselves. It does sometimes return an URL to the Amazon website, where they have pictures, but after doing some research about website scraping, it turn's out that it is illegal to show Amazon's product pictures when you haven't signed up for an Amazon Associates account.

2.2 Search

The second tab implements the search functionality of the app. It has three filter options:

- Age Group (selection)
- Title (input)
- Author (input)

Figure 2.2: Interface Flow of Search Tab when looking for a specific book

Figure 2.2 shows how a user can use those options to look for a specific title of a specific author. The blue search button must be pressed to trigger the search.

The options age, title and author can be used together or seperately. If, for example, the user only selects an age group, the results will show every book in the age group category. If the user only writes "crime" for the title value but nothing else, the app will show all books that have "crime" in their title.

[1] Image provided/created by Freepik.com

2.3 Favorites

In the DetailView, which is being discussed in a later section, the user can declare books as his/her favorite. Those favorites get saved and are presented in the third tab: the Favorite Tab.

The ion-cards in this view slightly differ from the cards that are used in the Bestseller and Search Tab. They still have the same image, but the title is now displayed underneath it and additionally there is a light-blue button at the end of the card, with which the user can quickly delete the book from his/her favorite-list.

An example is shown in figure 2.3.

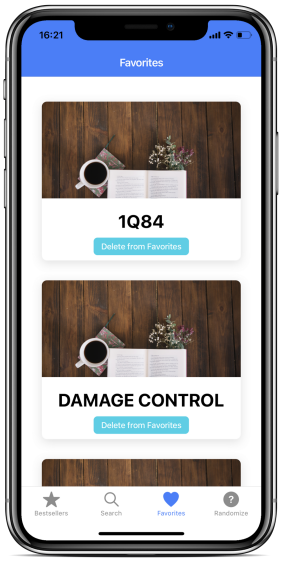

Figure 2.3: Favorites Tab

2.4 Randomizer

The Randomizer Tab generates and displays a random book from the database, based on a random offset and a random selection. Given that the database doesn't always have all entries filled, it sometimes only shows the title and the author (if there is no author entry from the database, a new random book will be selected as those are the minimum requirements for the randomizer).
The Randomizer Tab shows one ion-card, that is very similar to the DetailsView. It shows the title, badges of how often is was already ranked before, the age-group (if it has any) and the prize (0 if there isn't any in the database).

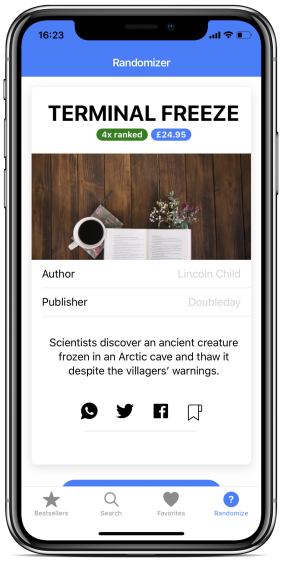

Figure 2.4: Randomizer Tab

After the picture, which when selected opens the web browser to display the owners website freepik.com, the author and the publisher are shown.

The next sections hold the description, sharing buttons and the favorite button. The last section displays reviews if there are any.

At the end of the screen (hidden under the tab controller in figure 2.4) there is a blue button, which can be used to generate a new random book.

2.5 Details View

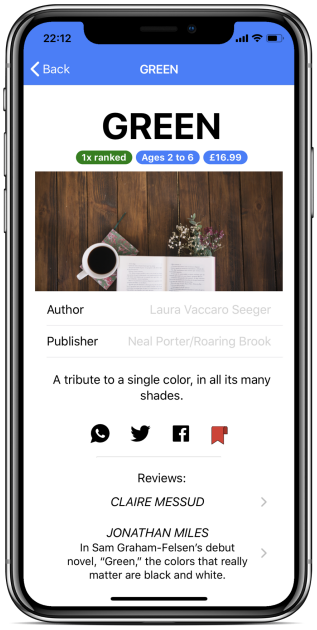

Figure 2.5: Details Page

If the user wants to see more information about the book object in the Bestseller-, Search- or Favorite-Tab, he/she can do that by simply selecting the cell card. This will trigger the Detailspage, which shows:

- book title
- badges for how many times the book was previously ranked (default 0), age groups (optional) and price (default 0)
- default picture
- author
- publisher (optional)
- description (optional)
- social sharing buttons and fav button
- amazon url (optional and only when coming from bestseller tab)
- reviews (optional)

To credit the source from where the picture was taken, it is linked to the freepik.com website. Therefore, if the user selects the image in the Detailspage, the browser will automatically be opened to show the corresponding freepik.com website with reference to the source.

Because of how the API returns its data, it is only possible to get the Amazon-URL from the Bestseller Tab and not when one requested a specific book. But when the Detailspage is opened from the Bestseller Tab, the user will be able to see an Amazon URL under the social sharing buttons. If selected, this link will again open the app browser to get referred to the products Amazon website.
Some books also have New York Times reviews. If that's the case, the reviews title and discription (optional) will be shown at the end of the screen. Those reviews are also links to their respective websites.

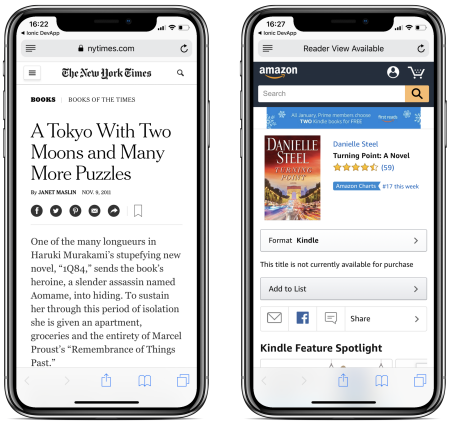

Figure 2.6: Trigger browser to show additional information

With the social sharing buttons in the middle it is possible to share some small details on Whatsapp, Twitter and/or Facebook. In Whatsapp a default text including the title and the Amazon URL (if available) is shared, whereas Twitter shares the Amazon URL with the text in its header, and Facebook only the URL.

Figure 2.7 shows the interface for those features:

Figure 2.7: Sharing Interface Whatsapp, Twitter and Facebook

Next to the social sharing buttons, there is a small bookmark button[2]. If selected, its color changes from transparent (default) to red. This means, that the book has been stored into the favorites and can be found in the Favorite Tab. By selecting the red icon again, it deletes the book from the favorites and changes its color back to transparent.

2.5.1 Original Wireframes

The original wireframes do not differ much from the actual implementation, except that the login page was not implemented. After a quick talk with the advisor, it was decided that for a normal book-lookup application like this, no login is needed as the favorite-information can easily be stored in the ionic storage too, and because it would only downgrade the user experience.
Instead of the login, the randomizer tab was implemented.

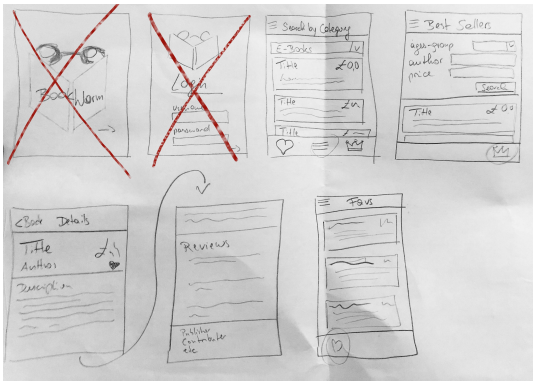

Figure 2.8: Original Scribbles

[2]Icon made by Smashicons from www.flaticon.com

Chapter 3

Implementation Details

This chapter explains how the most interesting and difficult parts of the app were implemented.

3.1 Data Handling

Every data request outside the ionic storage is done by the class **BookDbService** in **book-db.service.ts**. This class holds to variables that get initialized in the constructor: *baseUrl* and *apiKey*. The baseURL is just a convinient variable as the base URL for the requests is to 80% the same as always - it only differs for the reviews. The apiKey is the key that was given to me when I signed up on the NYT Developer platform and is needed in every call as authentication.

Every request then has to append certain information to the baseURL. Sometimes the baseURL has to be extended (for example with */names* or */best-seller/history*), but what always needs to be appended is the format, in which we want the result to be. In our case it is always *.json*. Afterwards we use *'?api-key=' + this.apiKey* for authentication. Everything after this has to be appended by using *&* (for example *'&title=' + title*). With the function call *encodedURIComponent()* we can make sure that the title we send in our request is correctly formatted, so that special symbols won't crash the system.

The API only ever returns up to 20 books at a time, but some requests accept an offset to get the rest of the data in blocks of 20. Therefore, we also add the offset to the http call when needed by adding *'&offset=' + offset* if the offset > 0 and nothing if the offset $== 0$.

3.2 Ionic Storage

The app uses Ionic's storage to save the favorites, and some additional information that needs to be passed from one page to another. To be able to use Storage, those two lines need to be executed in the terminal first:

```
1 ionic cordova plugin add cordova-sqlite-storage
2 npm install --save @ionic/storage
3
```

Listing 3.1: terminal installation of storage

Afterwards we can import Storage by writing *import Storage from '@ionic/storage';* in our .ts file. Then the constructor needs to define *private storage: Storage* in its parameters. Afterwards it can be used like this:

```
this.storage.set('author', author); // set information
let author = this.storage.get('author'); // get information
```

Listing 3.2: Usage of Storage

Those simple lines work perfectly when it is used to pass small amount of data between two pages. But it got tricky when it came to storing the favorites:

The API can return books by requesting a title, which would make it easy to just save the title of the favs in the storage and then request those from the database when needed. The problem is however, that the API will return **all** items that somehow match the given title. Therefore, if we save the book "LUX" into our storage, and then request it from the API, the API will return a large amount of results because "lux" is also contained in i.e. "luxurious" and "luxury".

To find a way to match the correct result to the stored book, we also have to store the author. The storage does, however, not take touples and the compiler has a lot of problems when storing and retrieving dictionaries. Therefore, we have to tell the program to put the information into a *Map<string, string>* and convert it into an array and then into a JSON object to store it properly. When retrieving the object, it then has to be decoded again.

```
// set information
this.storage.set('favoriteBooks', JSON.stringify(Array.from(this.favArray)));

 // get information
 this.storage.get('favoriteBooks').then((bookJSON) => {
   if (bookJSON == "" || bookJSON == null) {
    return;
   }
   var books: Map<string, string> = new Map(JSON.parse(bookJSON));
    // do something
 });
```

Listing 3.3: En- and decode Map-Object

3.3 Loading Controller

Every view is able to present a loading indicator in case that the data requests take too long. For it to work, the *LoadingController* needs to be imported from *'@ionic/angular'*. And then the constructor needs to define *public loadingCtrl: LoadingController* in its parameters. Afterwards it can be used like this:

```
const loading = await this.loadingCtrl.create({}); // create loading view
loading.present().then(() => { // present it while doing:
  this.api.getBooksByCategory(list_name_encoded, 0).subscribe( response => {
    // do data handling
    loading.dismiss(); //dismiss loading indicator when done
  });
});
```

Listing 3.4: Usage of LoadingController

3.4 Social Sharing

The app uses Ionic's social sharing to share information about the books to Whatsapp, Twitter and/or Facebook. To be able to use Social Sharing, this line needs to be executed in the terminal first:

```
npm install --save @ionic-native/social-sharing@beta
```

Listing 3.5: terminal installation of social sharing

Afterwards we can import Social Sharing by writing *import SocialSharing from '@ionic-native/social-sharing/ngx;'* in our .ts file. Then the constructor needs to define *private socialSharing: SocialSharing* in its parameters. Afterwards it can be used like :

```
this.socialSharing.shareViaWhatsApp(this.message, null, this.amazonUrl).then(() => {
  // do something
}).catch((e) => {
  console.log('Error while sharing on WA: ' + e);
});
```

Listing 3.6: Usage of Social Sharing

For Facebook we have to call *.shareViaFacebook()* and for Twitter it's *.shareViaTwitter()*. Unfortunately, those pre-defined methods are a bit buggy, because as seen in figure 2.7, Twitter and Facebook only let you share an URL - which aren't there in some cases.

3.5 Web Browser

When selecting a review, the Amazon URL or the image in the Detailpage, the web browser will be automatically opened to show the specific url. To be able to do this, we have to call *window.open(link,'_system');*. The first parameter is the URL we want to open, the second parameter defines which browser to open (in our case the system's default browser).

3.6 Reload Favs Tab

When coming back from the Detailspage, the Favorite Tab must be refreshed in case that a book was deleted from the favorites. This can be done by using *ionViewWillEnter()*, which will be called every time the page will become active again.

3.7 Change Button on Click

When selecting the favButton in the Detailspage or Randomizer, it not only saves the book to the favorites, but also changes its button color. This was implemented by having two same icons - one transparent, the other one filled in red. The HTML looks like this:

```
<button ion-button icon-only (click)="favButtonTapped()" ... >
  <ion-icon *ngIf="icon" [src]="icon" ... ></ion-icon>
</button>
```

Listing 3.7: HTML of favButton

The *[src]="icon"* refers to the icon variable inside the .ts file, which is a string (either 'assets/icons/bookmark.svg' or 'assets/icons/bookmark-full.svg'). This links the ion-icon to its png file. Therefore, a simple if-else clause in the method *favButtonTapped()* decides when to switch to which png file.

3.8 Randomize

The Tab4Page has a method called *getRandomBook*, which selects a random book to be displayed in the Randomizer Tab. To get a completely random book, we need two random numbers: The first number must be between 0 and 31823/20 and will define the offset for the API call. 31823 is the total number of books stored in the database. As the API only takes offsets in 20-steps, we divide the 31823 by 20 to generate a random offset, to be multiplied by 20 later on again. The second number must be between 0 and 19. The API will return 20 results, so this second number is to randomly select one of them.

3.9 API Limitations

Unfortunately, the NYT Book's API has some very strict limitations: **4,000 requests per day and 10 requests per minute**. In the beginning, this sounds very generous and enough to get sufficient data. But sadly, this is not the case, because when the API only returns 20 books per call, but the user wants to see more than that (which I limited to 100 max), those 10 calls per minute get exhausted quickly. After some e-mail communication with the nyt-developers, they have thankfully increased my calls per minute to 20 for a month's time. Sometimes this is still too low, and the loading indicator in the app won't stop animating until the page is reloaded (after waiting for at least one minute to reset the call limit).

3.10 What I wanted to use ..

Unfortunately, Ionic 4 has many bugs and the community is so small that even online you sometimes can't find the answers. I wanted to implement an In-App-Browser, which in theory should have been very easy using those lines in the terminal:

```
1 ionic cordova plugin add cordova-plugin-inappbrowser
2 npm install --save @ionic-native/in-app-browser
```

Listing 3.8: terminal installation of in app browser

But because of other dependencies, the installation was not possible. Even after doing some extensive research, I wasn't able to in-/decrease other libraries and therefore couldn't use the In App Browser feature. The same happened with Toast messages. According to Ionic's Main Website, there is an easy way to use Toast messages by calling following lines in the terminal, which too weren't able to be fully executed:

```
1 ionic cordova plugin add cordova-plugin-x-toast
2 npm install --save @ionic-native/toast
```

Listing 3.9: terminal installation of toast messages

Chapter 4

Acknowledgements

4.1 Ionic Framework

Most of the additional components and frameworks that I used, were found on Ionic Framework. The website also provides examples and sometimes refer you to github for an example code. Furthermore, it has an app called Ionic DevApp which allows you to start your app on your mobile device.
https://ionicframework.com/docs/components/
https://ionicframework.com/docs/appflow/devapp/

4.2 Ionic - on Github

I found a template on how to design ion-cards on this github project:
https://github.com/ionic-team/ionic-preview-app/blob/master/src/pages/cards/background/template.html

4.3 Ionicacademy

With the information provided by Ionicacademy I was able to implement the sharig feature for Whatsapp, Twitter and Facebook.
https://ionicacademy.com/ionic-social-sharing/

4.4 Stackoverflow

Of course, some bugs were not fixed on my own, but with the help of the community. Looking into Stackoverflow from time to time if stuck on a problem, is quite helpful. Unfortunately, there is not much to be found on Ionic specific bugs, but I also used Stackoverflow for hints in HTML, CSS and Typescript. For example, when I had to convert my Map<string, string> into a JSON to be able to store it.

Therefore, big shoutout to the community - thanks!
https://stackoverflow.com
https://stackoverflow.com/questions/46066343/convert-typescript-mapstring-string-to-json-string-representation

Chapter 5

Conclusion

Overall, developing in Ionic 4 and Angular, was quite a challenge. Not only because I had no prior knowledge except for the coursework, but because even the online resources weren't as helpful as I hoped they would be. The community is still very small, and most of the problems weren't made by myself, but were serious mistakes in the versions and dependencies, which I had no control over and in the end kept me from using them.

I can say that I have learned a lot about HTML, CSS and JS/Typescript, and that it is great to see how an app can "quickly" be developed for both platforms (android and ios), But given the small amount of resources and online-documentation/-help, I don't think that this is something that I would like to use in my future.