5. Software Architecture

5.1 Introduction

This chapter starts off with a general overview of the software architecture, then progresses into a detailed review of the architecture of the mobile app and API.

5.2 Software Architecture Overview

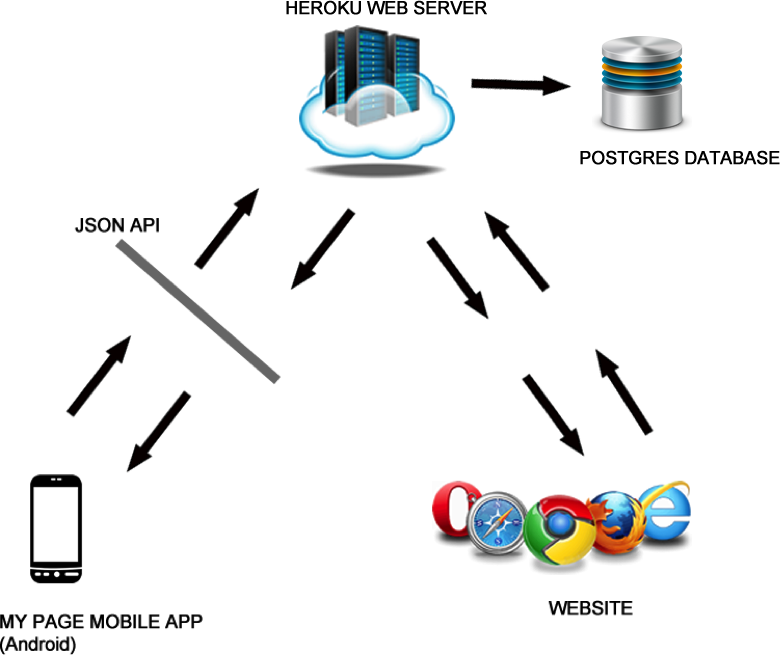


Fig 5.1: Overview of software architecture

As can be seen from the figure above, the mobile app makes Ajax calls to the JSON API, which then fetches the data from the Postgres Database through the Heroku Web Server. All mobile devices will interact with the JSON API to fetch data. This is a good design because it abstracts how data is fetched and how the API is implemented -- as long the mobile devices get data in the format expected, how the API is implemented doesn’t matter. The API does most of the data processing.

The introduction of the API doesn’t affect the operation of the website: the website continues to fetch its data via HTTP requests (JSON and HTML).

5.3 Architecture of Mobile Application

The ionic framework was used to develop the mobile app and the mobile app uses the MVC architecture (Model-View-Controller). MVC is an architectural pattern that splits a software application into three connected but separate parts: model, view and controller. This separation abstracts the internal representations of information from the ways the information is presented to the user (Wikipedia. *Model–View–Controller*).

5.3.1 Model

The model layer of an ionic app provides the data, often from an external web service. The controller requests data from the model; this data is bound to the view. Services and factories make up the model layer of an ionic app.

5.3.2 View

The view contains the HTML tags and directives of your pages. Views structure and present the data to the user. Views are also called templates in ionic. They have a html or htm file extension.

5.3.3 Controller

The controller controls the flow of logic and data manipulation. The controller calls the services and/or factories to obtain data to be presented in the view. A $scope object is used to bind the data to be presented in the view.

Some other components of an ionic app are:

5.3.4 Config

The config component configures the app. Here the routes (states) of the application are defined and hooked up with a controller and/or view.

5.3.5 Directives

Directives are DOM (Document Object Model) markers that can be custom elements, classes or attributes. They trigger custom behaviour on an element.

5.3.6 Physical Structure of “My Page” ionic app

The physical structure of an ionic app can be found below in Fig 5.2. Most of the development is done in the “www” folder. The “www” folder consists of following default folders and file:

* css: contains your custom styles
* img: contains the app’s images
* js: contains the app.js file that contains initialisation scripts. The app.js also contains the routes (states), directives, services and/or factories and controllers – though you can place them in separate files e.g. service.js
* lib: contains third-party libraries
* index.html: contains HTML tags and directives. The views can be separated into different html files and these separated files placed in a folder e.g. templates or pages folder.

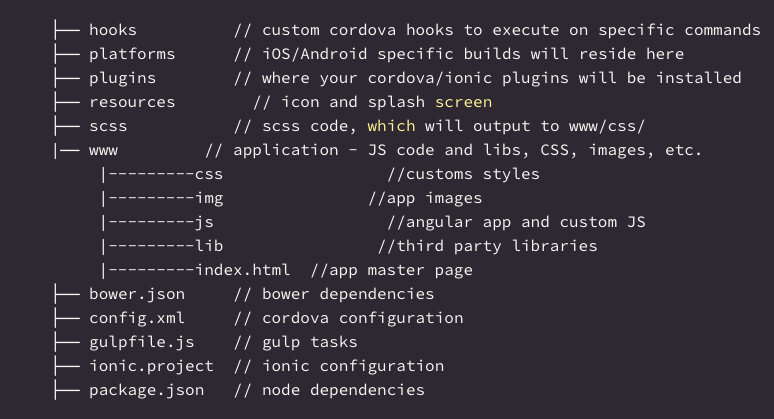


Fig 5.2: Physical Structure of an Ionic App (Nwamba, C., 2015)

5.4 Architecture of JSON API

The JSON API is based on the Rails framework and its architecture can be seen below.

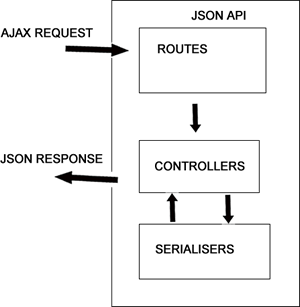


Fig 5.3: Architecture of JSON API

When the JSON API receives an ajax request, the routes file (config/routes.rb) resolves the requested url to the corresponding controller. The controller controls the flow of logic and returns a JSON object. The controller in the processing of JSON objects optionally uses the serialisers.

5.4.1 Physical Structure of JSON API



Fig 5.4: Physical Structure of JSON Rails API

The API reuses some code from the main source code (CiaB’s application), which can’t be submitted for confidential reasons, and is versioned, v1 (version 1), to ensure better control of releases.

5.5 Conclusion

The architecture of the software enables multiple devices to connect and obtain up-to-date data. Obviously the speed is dependent on the speed of the network. The physical structure of the Rails API enables me to work unhindered without worrying about modifying the source code of the main application. The v1 versioning of the API is also a good design.