

ATHENS UNIVERSITY OF ECONOMICS AND
BUSINESS

INTERNSHIP REPORT

Comparison of ReactJs and Angular

Author:
Kleio Fragkedaki

Supervisor:
Prof. Panagiotis Louridas

*Internship report and Thesis submitted as part of
Bachelor degree*

in the

Department of Management Science and Technology

June 27, 2019

Contents

I	Thesis	2
1	Introduction	4
1.1	Definition	4
1.2	Research Goal	4
1.3	Thesis's Structure	4
2	Background	5
2.0.1	Architectural Designs	6
2.1	Three-layer Architecture	7
	Presentation Layer	9
	Business Logic Layer	9
	Data Layer	9
2.2	Rest	9
2.3	Dynamic Pages	10
2.3.1	Data Architecture	11
	Redux	12
3	ReactJS	13
3.1	Definition	13
3.2	React Virtual DOM	13
3.3	Stateful Components	14
3.3.1	State	14
3.4	Lifecycle Methods	15
3.5	JSX	16
3.5.1	Babel	16
3.6	One-way data binding	17
3.6.1	Redux	17
3.7	Summery	18
4	Angular	19
4.1	Definition	19
4.1.1	Angular & AngularJS	19
4.1.2	HTTP	20
4.1.3	Testing	20
4.2	Typescript	20
4.2.1	Types	20
4.2.2	classes	20
4.2.3	Imports	21
4.2.4	Decorators	21
4.2.5	Utilities	21
4.3	Components	21

4.4	Services	22
4.5	Lifecycle Methods	22
4.6	Two or one-way data binding	23
4.7	Summery	24
5	Comparison	25
5.1	Architectural Differences	25
5.1.1	Framework & Library	25
5.1.2	Real & Virtual DOM	26
5.1.3	Templates: HTML & JSX	26
5.1.4	TypeScript & JS	26
5.1.5	Components	27
5.1.6	State Management	27
5.1.7	Data Binding	27
5.2	React Native & IONIC	27
5.3	Testing	28
5.4	Learning Curve	28
5.5	Performance	28
5.6	Popularity	29
5.6.1	What are companies using?	31
5.7	Discussion	31
6	Conclusion	33
6.1	Future Work	33

II	Internship Report	34
1	Introduction	36
1.1	Company Description	36
1.1.1	Beat App	37
1.1.2	Beat Hotels	38
1.1.3	Hive	38
1.2	Internship Goal	39
1.3	Report's Structure	39
2	Basic Characteristics	40
2.1	Department	40
2.1.1	Role in the Company	40
2.1.2	Department's structure	40
2.1.3	Basic Procedures	41
2.1.4	GR Squad	41
2.2	My Role	41
2.2.1	Skills Required	42
3	Projects/Activities	43
3.1	Introduction	43
3.2	Project 1	43
3.3	Project 2	43
3.4	Project 3	43
3.5	Project 4	43
4	Results	44
4.1	Project 1	44
4.1.1	Description	44
4.1.2	Best Practices	44
4.1.3	Schedule	44
4.1.4	Problems occurred	44
4.2	Project 2	44
4.2.1	Description	44
4.2.2	Best Practices	44
4.2.3	Schedule	44
4.2.4	Problems occurred	44
5	Updated Time Management	45
5.1	Time Schedule	45
5.2	Gantt Chart	45
6	Skills	46

List of Abbreviations

CEO	Chief Executive Officer
SMT	Senior Management Team
B2B	Business To Business
B2C	Business To Customer
CX	Customer Experience
DOM	Document Object Model
HTML	Hyper Text Markup Language
JSON	JavaScript Object Notation
RPC	Remote Procedure Call
SPA	Single Page Application
UI	User Interface
URI	Uniform Resource Identifier
REST	Representational State Transfer
MVC	Model View Controller
Ajax	Asynchronous JavaScript And XML
CLI	Command Line Interface
Framework	Reusable software environment to build applications.
JavaScript	High-level programming language.
TypeScript	Superset of JavaScript which adds optional typing to JavaScript.
Angular	JavaScript framework maintained by Google.
React	JavaScript library maintained by Facebook.
Node.js	Run-time environment for server-side JavaScript.
NPM	Package manager for Node.js modules.

Part I

Thesis

Chapter 1

Introduction

1.1 Definition

A lot of companies and individual developers are torn of which framework or library to choose for building client-side applications. ReactJS and Angular are the two most popular choices used for this purpose, and their comparison is one of the most popular topics in the tech community. In this thesis, it is aimed to compare these two technologies based on different metrics, such as popularity, performance, structure and many more.

1.2 Research Goal

The objectives of this thesis are to cite the main concepts of web applications and architectural design layers, to introduce the most popular front-end developing tools, ReactJS and Angular, and concluding with the evaluation of the analyzed frameworks and libraries based on a variety of metrics.

1.3 Thesis's Structure

This thesis is structured into 6 chapters. Chapter 2 presents some basic concepts of web applications and introduces the web's architectural layers, what dynamic pages and Rest are, and how the data structure of an application is conducted. Chapter 3 and 4 describes in detail the structure and component model of ReactJS and Angular. Chapter 5 contains the evaluation of the framework and library analyzed, with each evaluable aspect divided into sections. Chapter 6 concludes this thesis and set future work to be done.

Chapter 2

Background

Web applications have made rapid progress over the last years in the field of both technologies used and the architectural approach. A web application consists of a client and a web server. The client is considered to be, for example, a web browser on a mobile phone or a desktop computer. The client sends a request to the server, where processes are performed depending on the request, and a response is sent back to the client. (Voutilainen, 2017) As the picture below reveals, the client is making a request and the server attempts to fulfill it by referring to a database, performing calculations, controlling or sending another request to other servers.

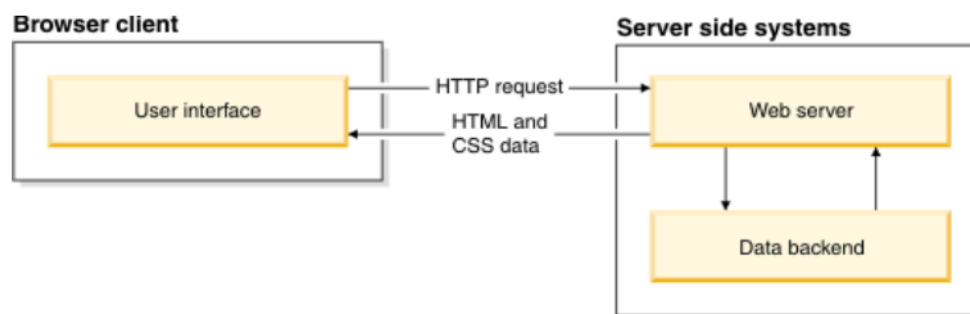


FIGURE 2.1: Web application data transfer model

Source: Voutilainen, 2017

Functionalities can be spotted on the client side as well. The client can customize HTML DOM, Document Object Model structure used for accessing different elements, display interactive visualizations or alerts through the utilization of Javascript programming language. (Voutilainen, 2017)

Architecture, which is a set of defined terms and rules used as instructions to build a web service, influence both software design and engineering. The choice of a web application's architectural design impacts on development time and maintenance costs, every day's transactional performance, response times, continual application's flexibility and scalability. The architecture is selected based on the app's complexity, integration level, number of users and their geographical distribution, network's nature and long-term transactional needs. (Cemus et al., 2015)

2.0.1 Architectural Designs

In the late 1950s, mainframe architecture was introduced. This architecture was designed for mainframe computers that are used for large-scale computing applications, such as data storage or customer statistics. Every kind of program and data was stored in a single machine and users could only reach this centralized computer only through the terminals' usage.

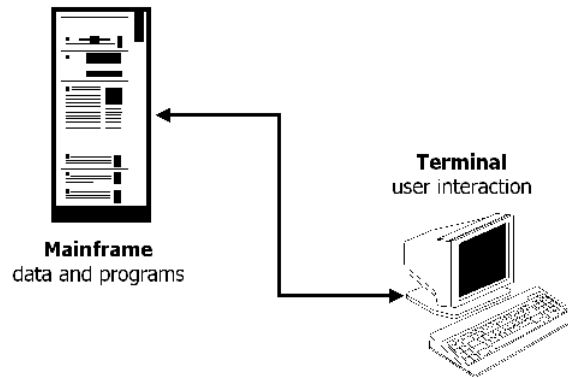


FIGURE 2.2: Mainframe Architecture
Source: Ramirez, 2000

In the 1980s, two-tier architecture was introduced due to the entry of network connected computers. In more detail, this architectural approach consists of an application running in the client and interacting with the server as a database management system. In that perspective, the client contains the presentation logic, navigation of the application, business rules and database access. By changing the business rules in a two-tier architecture, the client had to be modified and tested all over again, even in case the user interface is the same. For minimizing the costs of conversions, presentation logic and business rules had to be separated, a fact that gained the principle of three-layer architecture.

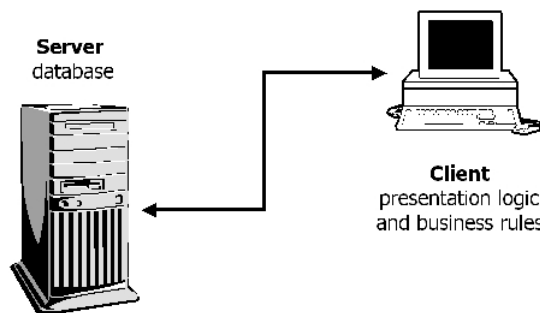


FIGURE 2.3: Two-Tier Architecture
Source: Ramirez, 2000

In a three-tier architecture, also referred to as multi-tier, there are up to three interacting layers. System functionality is thereby distributed with its own of these tiers having a subset of responsibilities.

As Figure 2.4 shows, the first tier refers to the presentation logic, known as client, and includes user interface and input validation. Second tier or, in other words,

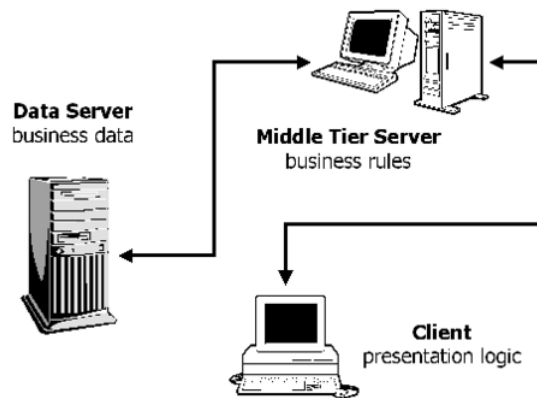


FIGURE 2.4: Three-Tier Architecture
Source: Ramirez, 2000

middle tier or application server, provides data access and the business logic. Finally, the third tier is the data server and provides business data and resources. The pros of a three-layer architectural design are the ease replace or modification of any layer without influencing or changing the others. Another advantage is the load balancing that this separation of layers and the database functionality provides. (Ramirez, 2000)

2.1 Three-layer Architecture

Three-layer Architecture is a software architectural pattern in which the application is separated into three logical layers known as presentation, business logic, and data storage layer. This architecture is adopted by client-server applications that have the frontend, backend, and database. (Chen and Zhang, 2013) Data access and calculations required by the presentation layer, are part of the business logic layer and, for this purpose, calls to middle-tier servers are made. Middle-layer servers can be approached by various clients, which are from different applications as well. Each one of these tiers has specific responsibilities and can be handled independently. (Gallaughier and Rarnanathan, 1996)

Interaction between client and server is succeeded mostly through the Remote Procedure Call, which is a way to describe calling mechanism among procedures and to exchange data via messages. In RPC, clients request data by passing parameters needed and specify data structure to received values for the request to be fulfilled.

HTTP is the protocol used for messages' passing. HTTP stands for Hypertext Transfer Protocol through which resources are exposed across distributed systems. An HTTP request includes a body with the representation of resources required by the client. An architectural style that is based on RPC implementation is called Representational State Transfer. REST is implemented in client/server models where the client is gaining data or interacting with resources managed by the server.

There are a lot of advantages in adopting the three-architecture layering design. First and foremost, modularity, having separate software entities allows each layer to be managed independently of the others. (Chen and Zhang, 2013) This has as a result different groups of people to focus on different tiers, so as a parallel development to be succeeded and people to become specialists. It has to be mentioned that

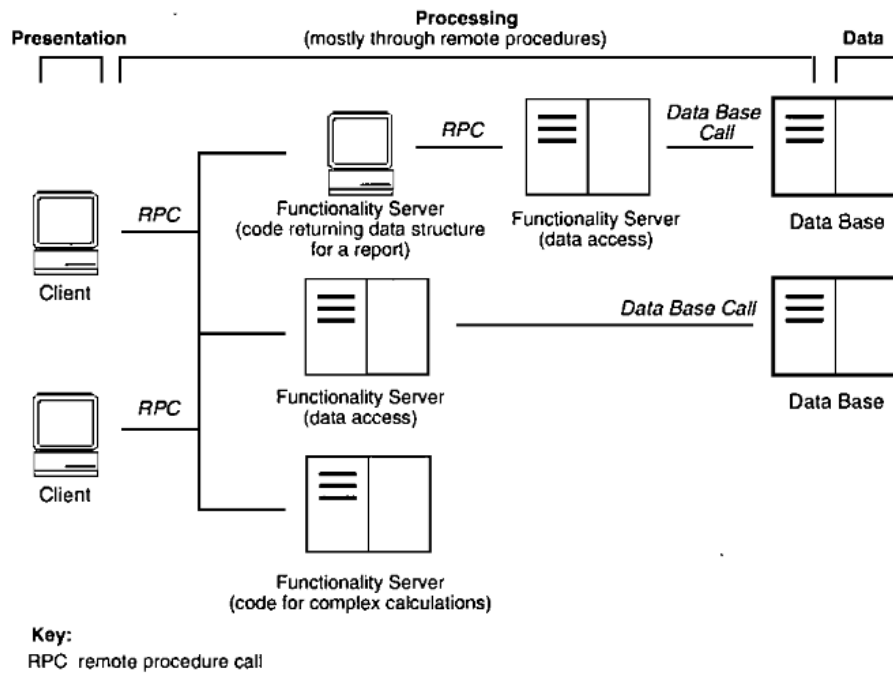


FIGURE 2.5: Three-Tier Architecture
Source: Gallagher and Rarnanathan, 1996

skills needed for application development vary, and having some experts responsible for each tier can improve the general quality of the application. (Gallaughier and Rarnanathan, 1996)

The scalability of the three-tier architectural pattern is another benefit. This architecture provides flexibility in resource distribution. In more detail, servers are portable, and dynamically distributed and changed based on the application's requirements. (Gallaughier and Rarnanathan, 1996) Each of the tiers scales horizontally in order trafficking and request demand to be supported. For example, scalability can be done by adding more Elastic Compute Cloud instances, which is a cloud service providing security and compute capacity, and load balancing to each layer. (Chen and Zhang, 2013)

Another core profit of this distribution is that code units can be reusable. This logic minimizes the maintenance costs, development efforts and the ability to easily switch technologies used. Moreover, various additional features support modularized applications, which means that integrated security, server control, and dynamic fault-tolerance are supported. (Gallaughier and Rarnanathan, 1996)

Security is easier to be performed in three-layer architecture. All of the interactions within the application are made through private Internet protocol addresses. Users access client/server systems via the presentation tier. The other two layers, server, and database are not exposed to the network which offers protection, security, and barriers against malicious users. Fault tolerant is also provided for adaption in any unexpected change. (Chen and Zhang, 2013)

In conclusion, this layering model is the most frequent architectural design and is defined as components' separation into different tiers. Each layer's components are abstract with limited dependences between them, reusable and easy maintained. (Chen and Zhang, 2013)

Presentation Layer

The presentation layer or client is the user interface part of the application. In other words, this layer controls the interaction between user and system and is also responsible for input validation. The presentation tier's infrastructure is exclusively related to interface elements. (Chen and Zhang, 2013)

Business Logic Layer

Business logic layer or server is the body of an application. This is related to how the service works and it is responsible for computations and connection between client and database. (Chen and Zhang, 2013) Several server requests and data access is made through the middle tier instead of the client, thus traffic, memory, and disk storage requirements are minimized. (Gallaughier and Rarnanathan, 1996)

Data Layer

The data access layer or database is responsible for providing business data and resources. Data are stored in this layer and are accessed from other layers when it is needed. (Chen and Zhang, 2013)

2.2 Rest

Representative State Transfer is an architectural style for building networked hypermedia applications that are easy to implement, lightweight, maintainable and scalable. A service based on Rest architectural approach is called Restful. The main purpose of a Restful service is providing data access to the client through a window. By the term of data or resources, we mean everything that can be captured in a computer-based informational system, such as pictures, videos, Web pages or any other business information. (Vaqqas, 2014)

The rest architectural style is based on four main principles. The first one is that resources are identified by URIs, which is a universal addressing space for resources and service exploration. Furthermore, the principle of the uniform interface is the way resources are by using a set of four operations. More specifically, resources can be controlled by PUT, POST, GET, DELETE, which are for the update, add, retrieve and remove actions. The next principle stands for self-descriptive messages. In other words, resources can be represented in a variety of formats depending on the requirements, such as JSON, XML, HTML or even plain text, and meta-data are also available in order caching to be controlled, transmitting errors to be detected and authentication to be performed. Last but not least, every interaction with a resource is stateless. This means that each request message is self-contained and there is no need for state transfer. (Pautasso, Zimmermann, and Leymann, 2008)

HTTP is the protocol that is used in almost every Restful Web service and it is implementing the service with features of a uniform interface and caching. This protocol is giving the ability of communication between client and server through messages. The client sends a request to the server, and the server returns a response to that request. (Vaqqas, 2014) Rest APIs are the bridge between this communication, so as the client to get the resources needed from the server without knowing how the structure of Restful API works and vice versa. (Padmanaban et al., 2018)

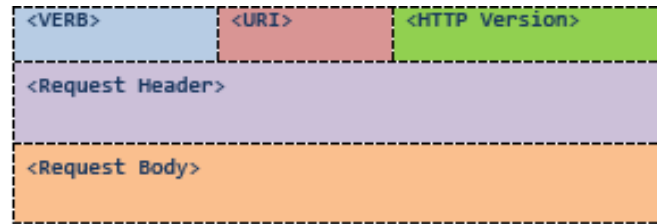


FIGURE 2.6: HTTP request format

Source: Vaqqas, 2014

In conclusion, the Rest architectural approach is a lightweight infrastructure, where services can be built with minimum tooling and effort that leads to inexpensive and low barrier adoption. Restful Web services serve a large number of clients because of the caching and load balancing build of Rest and have the ability to access resources without registration, and the advantage to optimize data in different formats. (Pautasso, Zimmermann, and Leymann, 2008)

2.3 Dynamic Pages

Web application systems are using Web browsers for representing the client. Web browsers interpret HTML, CSS and Javascript code, and communicate with the server side through the usage of URLs and HTTP protocol. In the first place, static web pages were sent to browsers and the server's responsibility was to locate and send files, based on client's requests. Afterward, dynamic pages were generated by servers via running software, and by clients via executable code. Thus, a lot of different software development, such as languages, APIs, libraries or frameworks, were built to support the dynamic pages' approach. Ajax, Javascript, Python, and many other such development tools, are a good example. (Kulesza et al., 2018)

Ajax stands for "Asynchronous JavaScript And XML" and provides asynchronous requests for data transfer. By using Ajax, an application can dynamically update its page's content without the need of reloading the entire DOM all over again. In general, Ajax is a way to get a certain functionality by using web technologies on the client side, to create asynchronous applications. Ajax is different than JS frameworks or libraries which include a lot of capabilities and functionalities along with Ajax. (*Redux Official Website*)

JavaScript is the language created to fulfill the need of client-based dynamic pages. JavaScript is being developed quick and new versions of the language are often released, like the so-called ECMAScript 2015, or namely ES6, and other more recent ones, like ES7. Generally, ECMAScript or ES is a standard formation body of JS scripting language with ES5, named also as ECMAScript 2014, to be the version all browsers understand. (Accomazzo et al., 2016) As regards coding with JavaScript, developers prefer to use ES6 and its upgrades for coding. This is because these new versions mark a new generation of the referring language with a dozen of additional features compared to ES5, the previous version. For using new features implemented in ES6, there are programs like Babel that convert the newer syntax to ES5, which is browser-compatible. (Masiello and Friedmann, 2017)

Moreover, new technologies based on JavaScript are regularly released. So, libraries and frameworks are inclined to make easier JS development and improve their capabilities. JavaScript's libraries exist for a long period and in 2006, they

got popular for the first time with jQuery. AngularJS and Ember, which are other JS frameworks, became known between 2010 and 2011. AngularJS was the first framework that composed routing and data binding in one. Ember followed and provided some improvements on AngularJS, such as better use of routing. Other frameworks and libraries, such as React, Angular, Vue, were announced until 2015. Javascript and its frameworks, CSS and native HTML are more and more powerful today. (Voutilainen, 2017)



FIGURE 2.7: Historical overview of JS frameworks and libraries

Source:

<https://slides.maxpou.fr/vuejs-yet-another-js-framework/index.html#/1>

A framework is defined as a large reusable collection of libraries, functions, and tools in order for the application's main structure to be implemented. Thus, JavaScript frameworks are implementing the whole structure of an application, and make the development of JS painless and faster by offering ready, optimized functions in comparison with native JS. Dependency management, file system structure and routing possibilities are also included. The term framework is used to declare that the execution flow of an application is handled by the framework and not the developer. This is the main difference between JS frameworks and libraries, since a library provides only a set of functionalities, while a framework conducts processing and data flow. (Voutilainen, 2017)

Angular and ReactJS are nowadays widely scoped, full and continually improved JS frameworks or libraries, and their popularity has been significantly increased over time in front-end development. However, there is no framework or library considered to be as the best, a group of great choices exists that do have different functionalities and are selected based on some parameters that will be investigated in this thesis. (Voutilainen, 2017)

2.3.1 Data Architecture

By developing dynamic pages, the need for an application's data management was increased. For this reason, several data architectural patterns have been developed over time. At first, the MVC pattern was used for a long period. MVC stands for Model-View-Controller, where model contains the business logic, the view presents data through the user interface and the controller connects the other two. However, MVC is not compiled into the client side in the best way, fact that leads to the development of other data architectural approaches based on MVC for front-end. More specifically, there is MVW, or in other words two-way data binding, which means Model-View-Whatever. This pattern is used to describe Angular JS's architecture.

MVW provides a common data structure in the application, and changes made in any area reflect the whole app. Another architectural approach is **Flux72**, which is a one-way data flow pattern. In this case, there are states that keep data and actions that change these data if needed, while views render what has been stored. Last but not least, there is the so-called Observable pattern which is implemented from the RxJs73 library. This is a publisher-subscriber architecture that provides streams of data. It includes functions for publishing values which are only executed in case subscribers exist. (Murray et al., 2018)

Redux

Redux is a library for data management on the client side and was based on Facebook's Flux architectural approach, which was mentioned previously as a Flux72 pattern. (Masiello and Friedmann, 2017) Managing data can be complicated when it comes to large apps. Intermediate passing of component state, the coupling between parent and child components that makes refactoring inflexible and the mismatching of state and DOM tree, are the reasons Redux was designed. ((2018))

More specifically, Redux is a Model-View-Controller library that handles data and interactions between layers inside an application. For this purpose, it is using reducers which are functions for computing application's state (Geary, 2016) and a global store that wraps all states of the application. In that way, data can be accessed and handled from any component, fact that unbinds UI from state changes, and reduces errors raised from state mismanagement. (Accomazzo et al., 2016)

Redux is better described via three main principles. First of all, there is a single store, which mostly has the form of a JS object, and uses reducer functions for managing the small parts of the store. (Masiello and Friedmann, 2017) In this way, it is easier to design and parse data through the application, and quick the process of debugging and testing. (*Redux Official Website*) The next principle is that the state must not be modified by any component. For this purpose, reducer functions are used to build a new state when an action is provided without affecting the original state. Finally, the last principle is that reducers have to be functions that do not include API calls and do have deterministic results. (Masiello and Friedmann, 2017)

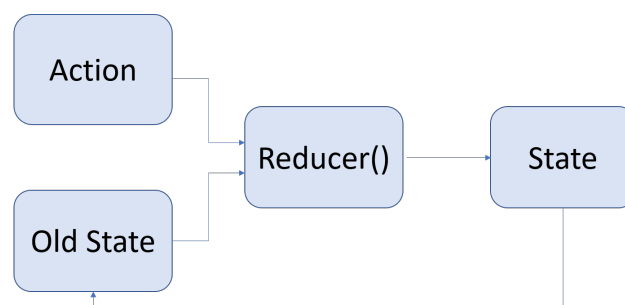


FIGURE 2.8: Redux Structure

In conclusion, Redux is based on an architectural approach that provides scalability and tools for better data management. This library can be used in combination with other libraries or frameworks, like React and Angular, as it will be mentioned in the next Chapters.

Chapter 3

ReactJS

3.1 Definition

ReactJS is a JavaScript library for building web interfaces. The library was created by Jordan Walke, a software engineer on Facebook, and is maintained by Facebook, Instagram and a community of individual developers and organizations. React became open source in March 2013, while it is used by Facebook since 2011, Instagram since 2012. PayPal, Uber, Sberbank, Asana, Khan Academy, HipChat, Flipboard and Atom are a few more applications using React. (Mardan, 2017)

As mentioned in Chapter 2, the library is just a set of functionalities that developers use. It doesn't aim to cover all fields of an application or to provide a complete solution for developing an app, unlike frameworks. React is a UI component library that creates interactive user interfaces. It was developed for complex and large scale web-based applications to manage how views change in response to data modifications. In that way, better user experience is provided with fast and robust development. (Mardan, 2017)

In more detail, React is mostly a view-layer solution, but it includes mechanisms for managing state without providing a specific way for data flow management, server or API interaction, or routing. For this reason, additional libraries will be needed. Its main concern is simplicity and scalability. The most popular features of React library are Virtual Document Object Model, Lifecycle Methods, Stateful components, JSX and One-way data binding that will be analyzed later on this chapter. (Masiello and Friedmann, 2017)

3.2 React Virtual DOM

DOM stands for Document Object Model and represents an XML or HTML document as a tree structure where each node is a different part of the document. The HTML DOM was designed for static pages and not for creating dynamic pages. So, when DOM is updated, every node and page has to be updated as well, the fact that it reduces application performance. For this reason, Virtual DOM was invented. Virtual DOM is an abstraction of HTML DOM and has analogous properties with a real DOM, with the difference that it cannot make any direct change to the screen. It is also lightweight and detached from the browser and can be updated without affecting real DOM which means faster manipulation and updates. (Naim, 2017)

React differs from other front-end JavaScript frameworks since it is not working with the browser's DOM. Alternatively, React keeps a JavaScript representation of the DOM in memory, the Virtual DOM which is a tree of JS objects, and makes all modifications in there. In that way, actual DOM is not directly manipulated and the minimum number of changes required are eventually applied to real DOM. More

specifically, React compares virtual DOM with the actual one, determines which parts have been updated and then updates only those in the browser's DOM. Moreover, Solutions for browser differences, server-side rendering, and implementation of target rendering are provided in this way. (Accomazzo et al., 2016) This approach brings greater performance which can be noticed by end users. (Masiello and Friedmann, 2017)

By updating actual DOM, there are some obstacles. It is difficult to keep track of changes, the current and previous state of DOM, so as to manipulate it based on needs. Furthermore, changing the real DOM is causing low performance and high costs. For these reasons, virtual DOM was introduced which provides an API for DOM's transformation. Developers code as if they are re-creating the whole DOM on each update. This leads to an easier development model which does not require tracking changes of DOM. This implementation uses effective algorithms for rendering a JSX element so as to be aware of what has been changed and update only those elements that are modified, updates at the same time subtrees of DOM and batch the updates. (Accomazzo et al., 2016)

In conclusion, React's Virtual DOM results in an easy-manipulated and improved way of building web applications. Virtual DOM is actually an intermediate area in which data are firstly changed due to faster processing. In fact, if changes in DOM are not frequent, Virtual DOM maybe is not suited, but in complex and dynamically modified ones, re-rendering, whenever needed, is a great feature. (Masiello and Friedmann, 2017)

3.3 Stateful Components

React split the user interface, which is what screen shows, into smaller parts in order to be reusable and easily maintainable. These parts are called React components and provide both data to the view and changes to it. More specifically, React is a set of components that are nested inside one another and there is one rooting component that composes all of them. Components are basically JavaScript functions that take inputs named properties and return as output React UI elements that are what is shown to the user. (Stefanov, 2016) React components include a render method that returns what has to be displayed as well as other lifecycle methods, in order code to be executed in certain times during the component's life. All these methods will be described in detail during the next section. (*React Official Website*)

3.3.1 State

Components receive as inputs the so-called "properties" that are passed externally through their parents. Except for properties, each component also has its internal state that can keep track of. (Masiello and Friedmann, 2017)

The state is a plain JavaScript object that is used to track user events and other variables inside the component. Moreover, it can be passed as properties to its child components. Furthermore, the state is owned by the component, is private to it, and can be updated any time needed. Whenever the component's state or its passed props are changed, the component and all of its child components are re-rendered. (Accomazzo et al., 2016)

3.4 Lifecycle Methods

Each React component rendered into the DOM follows a series of steps. Developers can access each step through component's life-cycle methods, so tasks and specific check conditions to a stage can be performed. Lifecycle methods are used to listen to changes in components. Thus, the component is constant until its parent passes new props or some event causes any change in its state. (Masiello and Friedmann, 2017) There are four main stages for each component's life, the so-called "Mounting" stage which is the phase where an instance of a component is being created and inserted into DOM, the "Updating" which is the phase where component is being re-rendered, the "Unmounting" where component is being deleted from the DOM, and another not that critical, the "Error Handling" for any errors occurred during rendering. Each of the first three previously mentioned stages of component's life include their own methods that will be described in detail. (*React Official Website*)

Before a component has been placed into DOM for the first time, a method named **componentWillMount** is applied so as a component to receive timers and data needed from the server. Method **constructor** belongs to the "Mounting" stage and is called when initializing the component. In this stage **render**, the method is also called and converts JSX elements to HTML which place them into DOM, or in other words mounts component. After mounting is done, method **componentDidMount** is called and used as integration between non-React and React libraries. (Masiello and Friedmann, 2017)

The first method called when either props or state has been changed, or in other words during the "Updating" stage, is **shouldComponentUpdate**. The method receives two arguments a new set of properties and a new state and compares those with the old ones. Moreover, this method is returning a boolean value. (Masiello and Friedmann, 2017) If the value returned is false, then the update is aborted which means that the render method won't be invoked, while if it is true, the update is normally happening. This is useful for performance reasons for the application, only in cases that the changes made are not that necessary to be made. Rendering is generally considered as computationally expensive and can slow down the application. This decision is made based on either the comparison of the new and old state and properties or if the component is static and doesn't need to be re-rendered. (Stefanov, 2016) If the method is not defined, its default value is true. The next method called, if the previously mentioned method returns true, is **render** again. In this case, render gets the new JSX representations, compares it to the old one into the virtual DOM, and creates and applies changed parts to real DOM. Finally, once this process is complete, **componentDidUpdate** which receives the previous state and properties as arguments are executed. This is used to operate on the DOM, like doing network requests, after the component has been updated. (*React Official Website*) After that the update lifecycle, the components remain inactive until a change has occurred. These methods are executed all over again unless the component is unmounted from DOM. (Masiello and Friedmann, 2017)

As regards the "Unmounting" stage, there is only one method called named **componentWillUnmount** which does not receive any arguments. This method is executed right before the component is removed from DOM and is the final stage of the component's life. (Stefanov, 2016) In this phase, any necessary cleanup is performed. Anything that has been created over the component's lifecycle, such as invalidating timers, canceling network requests, or cleaning up subscriptions made, are removed through this method. (*React Official Website*)

Each of the methods mentioned above might be used inside a component, but their usage is optional and based on the component's functionality needs. Furthermore, it is a good practice to use these methods for DOM manipulation and not other kinds of libraries alongside React. This is because React uses a virtual representation of DOM in order to apply and manage changes in the browser's DOM. So, if other libraries are used, there are possibilities of not being synchronized with React expectations so as errors to happen when React tries to match changes. (Masiello and Friedmann, 2017)

3.5 JSX

JSX, which stands for "JavaScript XML", was created by the Facebook community for React library and it is an extension to ECMAScript syntax. ECMAScript or ES is a standard formation body of JS scripting language, as referred to Chapter 2, and the ES6 version is mostly used among JavaScript developers. More specifically, JSX has a syntax analogous to HTML and is a markup language compiled to ES6 that aims to define the component's layout. It is not necessary to be used alongside with React but is considered as good practice, otherwise React is becoming much more complex and with lower performance. Furthermore, JSX includes tools for converting ES6 to ES5, the browser-compatible syntax. (Masiello and Friedmann, 2017)

JSX describes the way the user interface should look like by producing React elements. Events handler, state changes and data represented are closely related to UI logic. For this reason, JSX's components contain both markup and logic-based technologies in the same file, fact that differentiates React from other frameworks or libraries. (*React Official Website*) In that way, a lot of user errors and warnings are being caught because of the debugging made during the compilation process. (Accomazzo et al., 2016) Another benefit of JSX is speed. Even if JSX is compiled to JS, its output is optimized in a better way compared to the same code written directly in JavaScript. In mobile applications has been found that JSX is 12 % faster in IOS and 29 % in Androids than pure JS code. (Stefanov, 2016) To sum up, JSX is a representation of the component's HTML in JS and also close to object-oriented languages, like Java. (Masiello and Friedmann, 2017)

3.5.1 Babel

As referred in Chapter 2, nowadays the majority of browsers do not support ES6 or other languages, except ES5. This is happening because it takes time to update JavaScript engines of browsers and much more time for users to upgrade to their latest version. So, any language used on the client side needs to be converted to ES5 so as this gap to be closed and code to be executed in any browser. (Accomazzo et al., 2016) For this purpose, there are two main compilers, traceur32 by Google and babel33 by the JavaScript community. These two are not used for Typescript transition to browser-compatible code, but pure ES6 code. (Murray et al., 2018)

As regards, Babel, or babel133, is a JavaScript transpiler that changes ES6 to ES5 code as it has been pointed out. However, this transpiler includes the ability to understand JSX. In this way, JSX can be compiled to conventional ES5 and browser to normally execute code. (Accomazzo et al., 2016)

3.6 One-way data binding

Data binding is the way the connection between the user interface and business logic is succeeded. When binding of these two layers is properly built, components, that are connected to recently modified data, are re-rendered. (*Microsoft Official Website*) There are three main types of binding, one-way binding, two-way binding, and Observables, as it is analyzed in Chapter 2.

React follows a one-way data flow which means the state can change the view, but not vice versa. One-way flow or binding is a one-way direction from state to view that keeps under control state and models and makes the application's architecture less complex and more predictable. Data pass only from parent to child components, so in case of changes are made, event handlers need to be emitted. Even if it is needed extra code for setting data through event handlers to state in order view to be rendered, complex user interface and the number of views and states needed are reduced to a large extent. (Mardan, 2017)

Generally, This way of binding is suitable for applications that do not need to control the view's changes. (*Microsoft Official Website*) React combined with Redux so as one-way binding to be provided.

3.6.1 Redux

React is well combined with Redux since a one-way data flow pattern is adopted, as figure 3.2 also reveals. Redux is providing data management and a global store to the application which leads to scalability and interactivity. Any data can be updated from any view at any time and, in this way, debugging is easier. A lot of services are using Redux alongside with React for faster responses and better user experience, such as Facebook, Instagram, and Airbnb (Naim, 2017)

The process when using Redux is shown in figure 3.1 below. When a user interacts with the application, an action related to this interaction is dispatched. In case an HTTP request is required to the server side, the action is returned and waits for the response. Action and current state are sent to reducer when the request to the server is fulfilled, and reducer creates the new state and substitutes the old one. Finally, any component, that its state has been changed based on the updates in store, is re-rendered. (Murray et al., 2018)

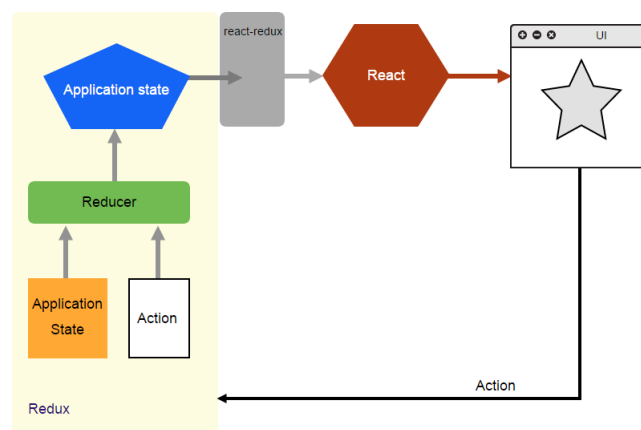


FIGURE 3.1: React with Redux

Source: Naim, 2017

3.7 Summery

React library has changed the way the user interface is developed by building stateful components in JSX syntax. React is one of the most popular libraries between front-end developers and well-known companies for the client side layer. In this chapter, was covered the definition and conceptual fundamentals of React, as well as some external libraries used for its better usage.

Chapter 4

Angular

4.1 Definition

Angular is a JavaScript framework for developing client-side applications. This framework was created in October 2010 by Google. It is maintained by Google and a community of individual developers since September 2016 that its code became open source. (Voutilainen, 2017) Angular as a framework is providing a large set of libraries, functions, and tools, and it is implementing the entire structure of a client-side application. Moreover, Angular is using templates that extend HTML style syntax for view's representation and is built in TypeScript, a JavaScript-like language. (*Angular Official Website*)

Angular's advantage is that provides the ability to create and managing project through terminal. Due to its complicated setup, Google created a command line interface, or CLI, that makes Angular's usage much easier. Angular CLI is responsible for creating and maintaining a common pattern inside an app. (Seshadri, 2018) Creating projects, adding new controllers and many other tasks, are automated and easily implemented with just one command. Last but not least, Angular CLI is based on Webpack, a tool that groups TypeScript, JS, CSS, HTML and any files used in the application, and makes deployment, building an app for production, simpler. (Murray et al., 2018)

4.1.1 Angular & AngularJS

AngularJS, or sometimes referred to as Angular 1, was firstly released in order to provide quick, scalable and maintainable web applications. As years passed, the way browsers, and general web used to work changed dramatically, so AngularJS stopped solving problems relevant to the new updates of the web. Thus, Angular was created in 2014 which is basically a totally newly written framework. As "Angular" are referred to all the newest versions, starting from Angular2 and above. (Seshadri, 2018)

More specifically, Angular and AngularJS are two different frameworks made from the same team. As regards AngularJS, it is using directives, controllers, scopes, services, and dependency injections, while Angular is using components, instead of directives, and services. Comparing these two frameworks, modules were replaced by web components and existing features of AngularJS were improved, such as dependency injection and templating. (Seshadri, 2018) Scope, which stands for two-way binding, and directive definition objects, controllers and angular.module were removed from the new versions of Angular. (Murray et al., 2018)

4.1.2 HTTP

Angular is providing an impended HTTP library through which external APIs can be called. The HTTP requests made are asynchronous that enhance pages performance. This means that the user can continue interacting without waiting for the HTTP's response. In general, there are three ways of dealing with asynchronous code in Angular, Callbacks, Promises and Observables. (Murray et al., 2018)

4.1.3 Testing

When using Angular CLI for component's creation, its main infrastructure is set up and includes four files. One file is for writing CSS, one for HTML, one for TypeScript code and one extra file for testing TypeScript code with a spec skeleton. There are some basic testing when initializing the component that can be improved and contain all test's needed for checking code's validation. (Seshadri, 2018)

4.2 Typescript

Angular is using a JavaScript-based language named TypeScript, which was the result of Microsoft's and Google's collaboration. TypeScript is a superset of ES6 with the additional features of types and annotations, as it is shown in the diagram below. It is compiled to ES5 in order code to be compatible with browsers, but it has five major upgrades compared to the ES5 code that will be described later on this section. (Murray et al., 2018)

It is not obligatory to write TypeScript in Angular apps. However, it is a good practice due to the features provided, which also make coding easier to be maintained and understood. (Seshadri, 2018)

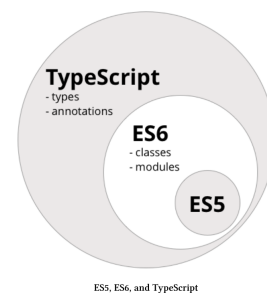


FIGURE 4.1: Typescript
Source: Seshadri, 2018

4.2.1 Types

TypeScript's name was inspired by its typing system, its most important addition. Type checking can prevent developers from bugs and make code much readable due to its obligatory clarification. (Murray et al., 2018)

Regarding types provided, they are the same as in JavaScript, booleans (true or false), numbers, strings, arrays, enums (similar to arrays but has numeric values as names), any (any type of value), void (no type or returned value expected). Types are not necessarily used in TypeScript, they can be omitted in case of writing quick code. (Murray et al., 2018)

decorator

4.2.2 classes

ES5 JavaScript is using functions and prototype-based objects, through which an object-oriented approach is succeeded. In this version, prototypes are used instead of classes. (Murray et al., 2018) In contrast, the ES6 version is using an object-oriented approach based on classes, that inherit functionality and built objects. (*TypeScript Official Website*)

In more detail, classes include properties, methods, and constructors. Properties are the data passed as arguments to a class's instance and can also refer to a type. As regards methods, they are functions included in an object and are called through the instances of them. Lastly, every object owns at least one constructor, which is a method executed at the creation of a class's instance. It can take parameters, but it does not return any values except the instance of the object. As object-oriented, classes inherit their behavior from their parent which can be then modified. (Murray et al., 2018)

4.2.3 Imports

By using import, library modules and other Angular applications are imported in components, services and other parts of the application. In this way, functionalities included in these imports can be used, and extra both code and effort by the developer's side can be reduced. (Seshadri, 2018)

4.2.4 Decorators

With the introduction of Classes in TypeScript and ES6, there now exist certain scenarios that require additional features to support annotating or modifying classes and class members. Decorators provide a way to add both annotations and a meta-programming syntax for class declarations and members. Decorators are a stage 2 proposal for JavaScript and are available as an experimental feature of TypeScript. (*TypeScript Official Website*)

A Decorator is a special kind of declaration that can be attached to a class declaration, method, accessor, property, or parameter. Decorators use the form `@expression`, where expression must evaluate to a function that will be called at runtime with information about the decorated declaration. (*TypeScript Official Website*)

4.2.5 Utilities

Due to TypeScript is a superset of ES6 JavaScript, the extended syntax feature of ES6 is also included in TypeScript and make programming easier. First of all, arrow functions is an important addition. Generally, the arrow-syntax functions are sharing the same with the rest of their components or service. In contrast, when a function is applied in traditional JS, it gives its own this, fact that leads to difficulties using global variables and functions. Moreover, new template string was added, thus variables can be included in strings, as long as backticks are used instead of double or single quotes, and multi-line strings are allowed. (Murray et al., 2018)

4.3 Components

Angular components are TypeScript classes decorated with extra attributes and metadata. Classes basically include any data and functionality needed, whereas decorators define the way of translating to HTML. (Seshadri, 2018)

Angular applications are in fact a tree of components, where its root is the application and that's what it will be rendered when any updates exist. As regarding components, they can be built from smaller components that consist of the parent-child structure. When a component is rendered in Angular, its dependent components are updated as well. (Murray et al., 2018)

More specifically, components have three main parts, which are the component's decorator or `@Component`, a view and a controller. Each component is responsible for a small part of the screen and controls this part through templates. Component's decorator is responsible for defining to other parts of the application how interaction with the component is succeeded. For this purpose, it uses templates, which is the visual part of a component or as called the view, and selectors, which indicates how components will be recognized from templates and what parts of HTML match the component. As regards controllers, they accept the server's requests and run comprehensive actions based on paths and parameters sent by the server. These actions are responsible for rendering components when needed. (Murray et al., 2018)

4.4 Services

In Angular, components are responsible for data representation and user interface's updates. Data are passed from components to the screen and, through events, from screen to component's methods where changes for user interface will be handled. Basically, components is the presentation layer, while servers are the ones responsible for fetching real data and business logic to the application. (Murray et al., 2018)

More specifically, Angular services are a common layer through the application that is used in three cases. Firstly, services are created when data are needed from or sent to the server. Secondly, services can be applied for writing application logic that can be reused across components. Additionally, services are used in case data sharing via components, particularly when components do not have access to one another. (Seshadri, 2018)

Services have their own store and can access all components in the application. Generally, they are the layer that handles how data and application logic works, so as components to be only bound to what will be represented on the screen. (Seshadri, 2018)

4.5 Lifecycle Methods

Each component and directive has its life-cycle that is handled by Angular. Angular's lifecycle hooks were created in order developers to write code that acts at specific moments of directive's or component's life. Life-cycle hooks are provided through an interface in each part of the app needed. There are three main stages of life-cycle, creation, update, and destruction, that each includes a list of hooks. In order to add these hooks in code, it is needed declaration of interface's implementation inside directive or component and also of hook's methods wanted. Hook methods are formed by the name of hook plus the prefix "ng". In this way, Angular will call the component or directive, and code is written in methods at the right time. (Murray et al., 2018)

In figure 4.2, it is shown the process followed in the lifecycle of any Angular-based app.

Let's dive into hook methods represented in the diagram above. Constructor is the first method executed and includes any initializations needed. Afterwards, when values of properties or data are modified, `ngOnChanges` method is triggered. Method `ngOnInit` is the following and is used to perform complicated initializations that cannot be made in the constructor. (Angular Official Website) Every time data change all related components or directives are updated, fact that affects performance. So, in case that a change is not important to be executed or an update

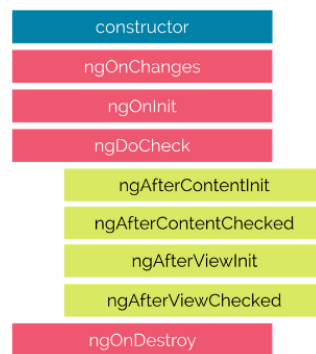


FIGURE 4.2: Life-cycle hooks in Angular

Source: <https://codecraft.tv/courses/angular/components/lifecycle-hooks/>

is needed when a specific property has been modified, `onDoCheck` method can be used. This method overrides `ngOnChanges` which is ignored. (Murray et al., 2018) Moving forward, `ngAfterContentInit` is triggered when content is extended into the component's view, while `ngAfterContentChecked` when this content is checked after its projection. Furthermore, `ngAfterViewInit` is relevant only to components and called after the initialization of component's and its child's views. The same with `ngAfterViewChecked` method which is executed after the views are checked by Angular. Finally, `ngOnDestroy` is the method that clears up any data remained in directives or components before they are removed. (Angular Official Website)

It is not obligatory hooks to be implemented in a class, but it is considered as a good practice. Performance can be increased and actions in particular moments can be executed. (Murray et al., 2018)

4.6 Two or one-way data binding

Two-way data binding is an architecture in which information flows from state to view and vice versa. In Angular JS, the default data flow is the two-way binding or MVW as noticed in 2, which means that when the model is modified, the view is also changed and the other way around. (Murray et al., 2018) This type of binding is easy to start with and is also suitable for totally interactive user interfaces since data are changed in two different directions. (Microsoft Official Website)

On the other hand, two direction data binding is possible to cause a flow of unpredictable updates and a difficulty to follow data flow as the application goes bigger. Moreover, another problem with this architectural approach is that it binds data flow with the DOM tree. For these reasons, the new released Angular is flexible to change between one or two-way data structure if it is needed. As described in Chapter 2, two-direction binding pass data down via components and event handlers needed to reflect state based on the view layer's changes. For adapting a one-way structure, Angular can use either Observables-based, such as Reactive Extensions Library for JavaScript (RxJS), or Flux-based architecture, such as Redux. When using Observables as main data architecture in Angular, it is named Reacting Programming which is a way to use asynchronous data streams. (Murray et al., 2018)

4.7 Summery

Angular is a framework that includes most of the libraries and functionalities needed to build a client-side application. Moreover, it is well combined with TypeScript language and is widely used by both individual developers and companies. In this chapter, was covered the definition and main functionality of Angular.

Chapter 5

Comparison

A lot of companies and individual developers are torn of which framework or library to choose for developing client-side applications. ReactJS and Angular are the two most popular choices used for this purpose. In this chapter, these two technologies will be compared based on both personal experiences and what has already been analyzed in previous Chapters.

There are no official references or scientific papers citing the comparison of ReactJS and Angular so far. For this reason, personal experience in both technologies will be the guide. As regards my experience, I have created a complete web application in Angular7 which is called **Hom-e** and is a home aggregator, while I was coding in React during my internship.

Chapter 5 includes six main sections that will form the key points of comparison. Firstly, architectural differences based on previous Chapters are pointed out. Moreover, mobile application choices, testing capabilities, learning curve, performance and popularity of each technology will be cited. In the end, a discussion about the topic and a recap are included.

5.1 Architectural Differences

Based on Chapters 2, 3 and 4, architectural differences of ReactJS and Angular will be pointed out. This section is separated into seven main parts.

5.1.1 Framework & Library

One of the main differences of ReactJS and Angular is that the one is a JavaScript library that provides only the view layer, while the other is a complete MVC framework that provides all functionality needed to the app.

As regards Angular, it is a framework. As mentioned in Chapter 2 and 4, frameworks provide a collection of libraries and functions to support most features needed in an application. Angular apps have a defined way of how their structure is. Developers do not need to spend time deciding which extended libraries to use, such as routing libraries, but they can directly start coding. However, there is less flexibility since they need to use what Angular provides.

On the other hand, ReactJS is a user interface component library. This means that it does not aim to provide a complete solution for developing a client-side application, as it is also referred to Chapter 3. The React library provides the view implementation, while external libraries are needed for adding functionality in the application. This leads to freedom of choice comparing to Angular, but also many more dependencies. When using ReactJS, developers need to upgrade and migrate dependencies. Furthermore, the architecture and folder hierarchy of each React project

differs. Thus, the app's performance and maintainability can be affected to a large extent.

5.1.2 Real & Virtual DOM

In Chapter 3, we mentioned that ReactJS is using virtual DOM instead of the actual one. This is because virtual DOM can be accessed and changed faster than real DOM which is on the browser's memory. In more detail, React's DOM is used as another instance of DOM and all changes needed are first made in that. When all changes are applied, the differences between previous and current HTML are checked, and only the updated parts of the tree are applied in real DOM. This results in higher performance and speed of React apps. (Voutilainen, 2017)

Contrarily, Angular is applying changes directly to Real DOM. More specifically, Angular is first detecting changes made for each component, and then it rules how those will enter to DOM. A change detector is applied for each component of the application. Detectors are responsible for recognizing updates made inside the component and then modifying DOM only at its changed parts. By using detectors, Angular is fast, but for higher speed, Immutables and Observations are used. Through Immutables, checks to each property are reduced, while through Observables, events are triggered and change detection is limited to specific properties. (Voutilainen, 2017)

5.1.3 Templates: HTML & JSX

React is using JSX syntax as shown in Chapter 3, which is a combination of UI templates and inline JS logic. In this way, components used in React include both markup and logic in the same file. For this reason, a lot of developers think that having everything in one place results in code completion and compiling checks. Regarding Angular, as referred to Chapter 4, it uses templates that are HTML embellished with directives, such as "ng-if" for attaching custom behavior to DOM.

5.1.4 TypeScript & JS

In Chapter 3, it was mentioned that React developers mostly write in JavaScript to add functionality in their application. JS is a dynamically-typed language, meaning that variables do not need to be defined. JavaScript also includes lots of useful features to its early versions (up to ES6) that lead to stating coding and a class-based structure. React favored by Flow, static checker developed by Facebook, supports checking of types as well.

Controversially, based on Chapter 4, Angular is mostly used alongside with TypeScript, a typed JavaScript-based language that transpiles into JS code. TypeScript's advantage is its typing system that can prevent developers from bugs and unreadable code. It was firstly developed for providing an Object-oriented structure to JavaScript developers before the induction of ES6.

Based on my experience, lots of developers prefer writing in JavaScript, since it is widely used and doesn't limit quick developers with a typing system, like TypeScript. TypeScript has its own rules and quick coding is not an easy choice, but it prevents from compiling-time errors.

5.1.5 Components

Both React and Angular are using components to their structure. Components are JavaScript functions that take inputs named properties, revise those inputs and return as output UI elements, which are what a user sees. It is a good practice to work in components since they are small reused parts of the code that reduce complexity and enhance the maintainability of code.

5.1.6 State Management

In an application, every component has its state. More specifically, the state is an interval store in which variables of any type can be saved and used inside the component.

In React, Redux is usually adopted for the app's state management, as referred in Chapter 3. Redux is providing global storage that any component can access if needed. This is how the interaction between components is succeeded.

In Angular, state management is handled by the framework. However, as the application goes larger, the state is not easily managed, and Redux can be used for components' connection as well.

5.1.7 Data Binding

One main difference between ReactJS and Angular is how they manage data. In Angular's Chapter (4), it is mentioned that Angular used to have two-way data binding, meaning that data flow from state to view and vice versa. Even if this approach seems to be simple, it is possible to cause unpredictable updates and difficulties of following data flow, as the application goes bigger. By adding more and more logic, different views update the same models which can also update other views, a fact that can cause an infinite loop problem. (Mardan, 2017) In this way of binding data, event handlers are not needed, but they are managed by Angular. Nowadays, in new versions of React developers have also the choice of one-way data binding by adding either Observables or libraries like Redux.

In React, there is a one-way data flow, which means that only the state can update user interface elements and not the other way around. As mentioned in Chapter 3, React is well combined with Redux, a state management library following a one-way data flow. Thus, there is better data overview, a fact that leads to easier debugging.

Both approaches have advantages and disadvantages. Though, when it comes to large and complex projects, two-way data binding causes slow manageability and performance. Controversially, one-way data binding leads to easier handling and maintainable data flow, since event handlers are managed by developers and thus each phase of components' state is known.

5.2 React Native & IONIC

Both React and Angular can be used for building mobile applications. More specifically, React uses the so-called React Native. React Native is a platform developed by Facebook and uses JavaScript, declarative components and the same design as React. React and React Native have small differences in syntax since the second one is using the same user interface blocks as iOS and Android apps. (*React Official Website*)

On the other hand, the Ionic Framework can be integrated with Angular for the development of mobile applications. Ionic Framework is an open source UI tool for

building hybrid applications based on web technologies, such as HTML, CSS, and JS. Ionic has official incorporation with Angular but it also supports other libraries like React and Vue. (*[Ionic Official Website](#)*).

It has to be mentioned that Ionic-based mobile apps are just web applications inside a native view container, and not truly native as React Native is. Thus, comparing to React Native, Ionic apps are slower and with lower performance.

5.3 Testing

As regards testing, Angular is providing Angular CLI that is a command-line interface for creating apps and adding files, tests and deployment capabilities, like it was analyzed in Chapter 4. Moreover, Angular can be integrated with other end-to-end testing and debugging tools, such as Jasmine, Karma, and Protractor.

On the other hand, React is using Jest, a library created by Facebook for JavaScript code testing. Jest is included in every React app and is well integrated with Enzyme for component testing. A drawback of React is that it uses lots of separate tools for different types of testing, such as Jest for JS code, Enzyme for components, a react-testing library for Virtual DOM testing, react-unit for unit testing and skin-deep for render testing.

5.4 Learning Curve

When it comes to choosing new technology for building web apps, an important parameter is its learning curve. The learning curve depends on the developer's previous knowledge and familiarity with the discussed technology. In this section, we will point out what it is needed to be known when developing in React and Angular.

For building React applications, it is needed to learn how to code in JSX, create components and manage state. It is also needed to write JavaScript for adding logic in the app. If a developer knows how to code in JS, React's development becomes much easier. Furthermore, React doesn't include any routing capabilities, which may be both pro and con in terms of learning. Developers can choose their preferred routing library, but if they are not familiar with any, this is a plus in learning features. Redux is also another library that most React users need to learn since it provides state management. In general, React is not the most simple choice, because of the additional libraries and JSX. However, due to its flexibility, it is considered to be simpler than a framework (Mardan, 2017).

Angular is a framework which means that has its ways of handling things, such as installation, error messages and compilation. Directives, modules, state management, decorators, services, components, dependency injection, template, and pipes are some additional features of Angular that need developers' attention. Angular may have the biggest learning curve because of TypeScript, specifically for those who are not familiar with JavaScript frameworks (Voutilainen, 2017).

5.5 Performance

Application's performance is an important criterion of choosing between ReactJS and Angular, because of responsiveness and user experience. To compare these two technologies' performance, technical factors that are based on the application's execution phases will be mentioned. More specifically, these factors are the startup

time of the app and the DOM's creation and modification. Even though these two are depending on the user's browser, and its JavaScript engine, supporting different types of browsers and application's performance in each of them is also considered.

In React applications, startup time is relevant to what libraries are added in the app. React do not have many dependencies, unlike a framework, cause most of the features needed are included as dependencies. Thus, if there are not many external libraries, the package size is smaller and the performance is higher. Moreover, React can also include routing libraries that are dividing JavaScript content into parts. In this way, these parts are fetched from servers based on page changes, a fact that limits loading time as well. Furthermore, React's virtual DOM, as mentioned before, is designed for improving performance and user experience. (Voutilainen, 2017)

As regards Angular, as being a framework includes lots of libraries that may not be needed, so performance is reduced comparing to React. Though, its impeded routing library, separate code into parts that are loaded asynchronously when it is needed. One more thing that improves the performance of Angular apps is TypeScript's compilation. TypeScript can select between Just-in-time compilation, in which HTML elements are compiled in the browser and output can be seen directly, and Ahead-of-time compilation, in which HTML elements are compiled in the server and in this way, rendering is faster in a browser. (Voutilainen, 2017)

5.6 Popularity

Popularity is one of the key ways to choose across new technology since it considered to be the measure of community support and so to the effectiveness of the application's development. This section's purpose is to examine ReactJS and Angular's popularity according to GitHub, Google trends, npm trends, and what well-known companies have chosen to use.

GitHub is a company and provides hosting for software development version control through Git. Both ReactJS and Angular are open-source and their code can be found in their respectively public repositories. Stars on GitHub is a way for GitHub's users to favor the most useful and used repositories. [Angular's repository](#) has, at the time this thesis was written, 49,250 stars, 13,411 forks, and 954 contributors, while there are 2,496 issues due to framework's variety of functionalities. On the other hand, [React's repository](#) has 131,625 stars, 24,298 forks, and 1,298 contributors, and as a library only 556 issues. React seems to be much more popular based on GitHub stars and contributors, but we need to take into account that Angular (up to version 2) was released 3 years later than React.

Another good metric is the number of downloads made. For this purpose, we used npm trends for comparing downloads during the past two years in both ReactJS and Angular. As figure 5.1 reveals, two years ago React and Angular had approximately the same number of downloads, while now react has 3,870,000 more.

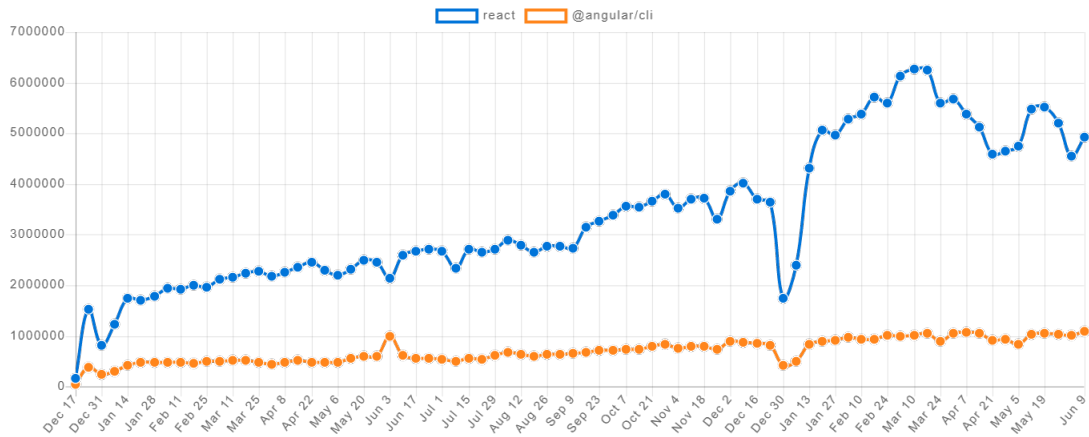


FIGURE 5.1: Npm trends of ReactJS and Angular CLI

Source: <https://www.npmtrends.com/react-vs-@angular/cli>

Furthermore, the Google Trends search hit is another way of measuring popularity. According to Google Trends's chart, Angular, which is the red line, used to be more searched worldwide in Google's search engine two years ago. Nowadays, React seems to win with 30 more hits on average every day. In figure 5.2 (B), it is shown that in China and North America is mostly used to React library, while in Latin America and some European countries Angular framework.

(A) Trends during
16/06/2015-2019

(B) Countries' preferences

FIGURE 5.2: ReactJS VS Angular in Google Trends

Source: <https://trends.google.com/trends/explore?date=2015-05-16%202019-06-16&q=%2Fm%2F01211vxv,%2Fg%2F11c6w0ddw9>

Based on the previous metrics, React seems to grow faster than Angular does. However, both technologies are pretty popular among professional developers, as indicated in Stackoverflow's survey in 2019.

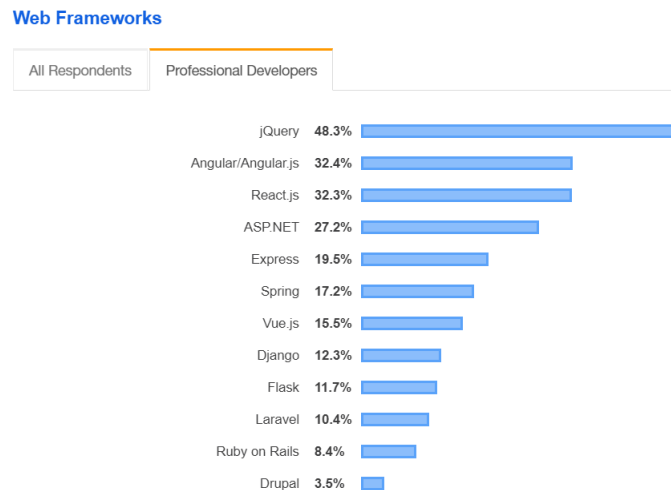


FIGURE 5.3: ReactJS-VS-Angular Survey Stackoverflow 2019
 Source: <https://insights.stackoverflow.com/survey/2019>

5.6.1 What are companies using?

There are a lot of companies that utilize both ReactJS and Angular to their applications. Some of the most famous ones in React development are Facebook, Airbnb, Uber, Netflix, Instagram, WhatsApp, and Dropbox. Other known companies that are using Angular instead are Google, Nike, Forbes, Upwork, General Motors, HBO and Sony.

5.7 Discussion

During my internship, I experienced some good practices and problems as a new developer to React that can be compared to Angular. First of all, it is good practice to use Redux. State mismanagement can lead to complex apps and unresolved bugs. Redux is using a global state, through which interaction between components is succeeded, and it leads to one-way data flow via selectors and actions usage.

Furthermore, in many cases, I had to use React's life-cycle methods to complete and improve the functionality of features added in BeatHotel's web application. Through these methods, it is possible to execute code and update state in specific phases of component.

Additionally, another good practice is the usage of ES6 instead of ES5 JavaScript. ES6 is providing many more features, such as arrow functions, stating code and an object-oriented approach that improves coding. In my point of view, Angular's TypeScript is not that easy to be handled but prevents developers from bugs and compile-errors.

On the other hand, a great problem of React apps is the difficulty of upgrading the project, due to the variety of dependencies added. A lot of dependencies are using other libraries. This means that when a dependency is upgraded, the project may crash if its required libraries are not updated as well. Controversially, Angular handles automatically its upgrades, since most of the libraries used are embedded in the framework.

Both Angular and ReactJS have advantages and disadvantages. When it comes to choosing between the two, there have to be considered metrics regarding the development team's previous experience, library/framework's popularity, performance and other capabilities like mobile application development or testing.

Chapter 6

Conclusion

During this thesis, there were analyzed both fundamental concepts of web applications, including the different architectural approaches and historical references, and the two most popular front-end tools for building web projects, React and AngularJS. The main purpose was to compare ReactJS and Angular, a frequently defined question among newcomer companies and developers. This comparison took place based on different metrics that cite each framework/library's advantages and disadvantages.

6.1 Future Work

To complete ReactJS and Angular's comparison in all possible perspectives, performance needs to be examined and proven. In future work, creating the same application in both client-side tools can provide an equal and more objective correlation. In this sense, I aim to build my app, Hom-e, which is developed in Angular, in React as well and then check the performance and user experience of both apps.

Part II

Internship Report

Chapter 1

Introduction

As part of my Bachelor degree, I did an internship for three months in Beat, a company that started as a Greek startup about 5 years ago. From March 18th 2019 until June 18th 2019, I contributed as a Software Developer intern in several projects that was referred to a product named "BeatHotels".

1.1 Company Description

Beat is a company that is developing a mobile application for taxi cab and peer-to-peer-ridesharing. The app is based on the idea of establishing a direct connection between drivers and passengers by offering both sides a modern alternative to conventional booking processes. First known as a greek startup named Taxibeat, the company was founded in 2011 by Nikos Drandakis in collaboration with associates Kostis Sakkas, Nikos Damilakis and Michael Sfictos. Taxibeat was acquired in February 2017 by MyTaxi, a subsidiary of the automotive manufacturer Daimler AG, and renamed to Beat. Nowadays, Beat is part of the FreeNow group and its CEO is Nikos Drandakis. The FreeNow group is the ride hailing joint venture from Daimler and BMW, and consists of the services MyTaxi, Beat, Kapten, Clever and Hive, the e-Scooter service.

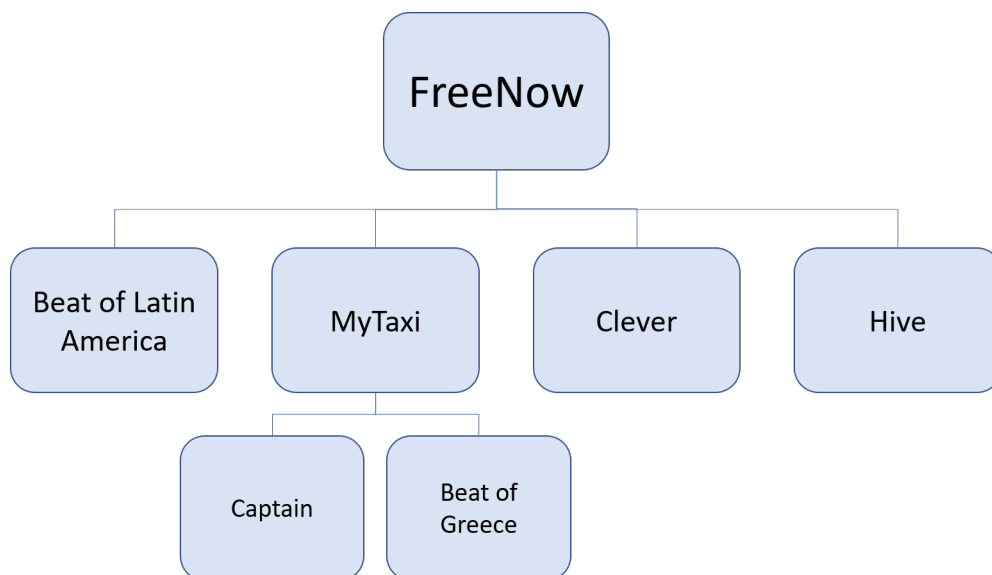


FIGURE 1.1: FreeNow's structure

Beat headquarters in Athens, Greece, while additional development and operation offices are located in Lima, Santiago, Cali, Medellín, Bogotá and Mexico City. It currently has more than 580 employees all over the world, with approximately 400 of them being in Greece. Company's teams work in small, autonomous groups of people following agile methodologies.

As regards Beat's structure, it is slightly different after its buyout. More specifically, the firm is separated in two parts based on its market targets, Beat of Latin America and Beat of Greece. This separation is because Greek market is basically part of MyTaxi group, while Latin America Market is only part of FreeNow, as shown at the diagram above. So, Beat has eight main departments, Business Operations, Finance, Engineering, Greek Market, People Operations, Marketing, Operating Office and Senior Management Team. The last department mentioned is conducted by Nikos Drandakis and his team as picture below reveals.

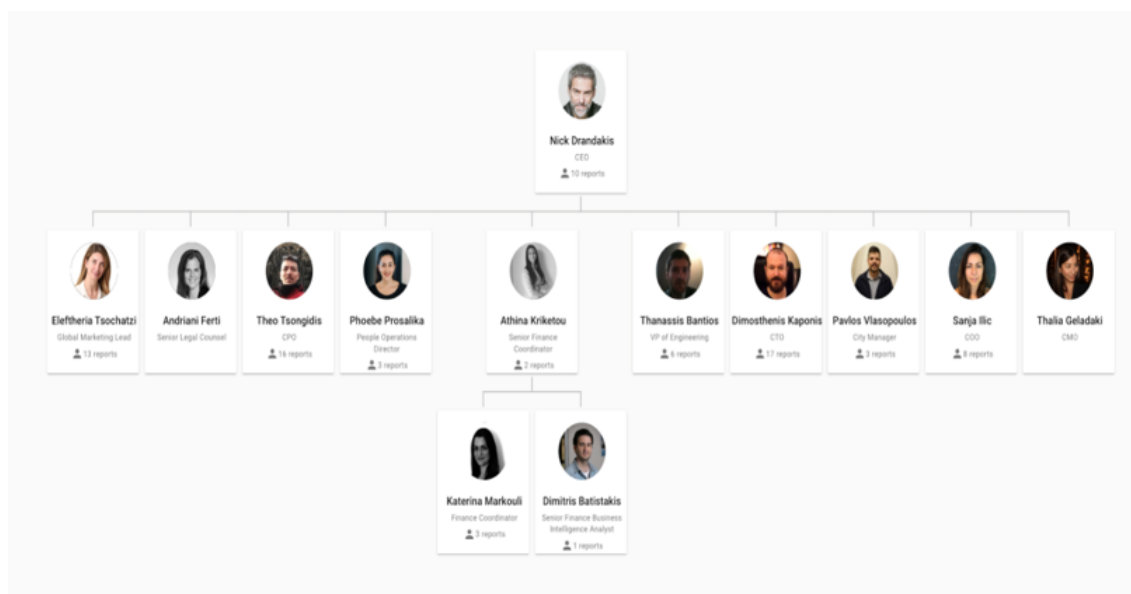


FIGURE 1.2: Beat's SMT Department

Products provided by Beat are mainly three in number. Beat App, with the extended service of peer-to-peer-ridesharing in Latin America Marketplace, Beat Hotels, B2B service only provided in Greece, and Hive, e-scooter services in Greek Market.

1.1.1 Beat App

Beat App is a B2C service provided in both Androids and IOS operating systems. This service is a connection between drivers and candidate passengers.

In other words, someone looking for a cup can find the closest one without any extra costs by using passenger's Beat app. Anyone that has downloaded the app can call a cup from any place, be able to see driver's rating from other Beat passengers, their personal data such as name, plates and more car details or services provided. In addition, before ride starts, an estimated price is given,



FIGURE 1.3: Beat App

list of the closest drivers is shown, and the candidate passenger has the ability to choose one of them based on the details and services provided, and to rate when ride is completed.

On the other hand, driver use another app through which the connection between two-sides is succeed. Available drivers can be located and the closest ones are shown to the candidate passenger. Driver can also accept or reject a ride, see the recommended route and see where passenger is before departure.

1.1.2 Beat Hotels

Beat Hotels is a B2B service that has the same aim with Beat App, the connection between candidate passengers, in this case people staying in a Hotel and request for a taxi, and drivers. The difference between these two services is, except from the aimed passengers, the virtual queues of drivers created in each Hotel.

There is a driver app, different from Beat driver's app, created in React Native and Node.js. This app is used from the driver in order to check the available Hotel queues and how much complete they are, get in a queue and start a ride.

On the other side, Hotel's have a customized dashboard, written in ReactJs and Node.js. Through this dashboard, they can call a taxi for a customer, see where the taxi is any time and get statistics of rides and revenue they have from rides completed.

Finally, a similar dashboard is developed for the agents of Beat with all the informations of each Hotel that Beat's corporate. The agents have the permissions also to change the amount of a ride if it is needed, check all completed rides, or block a driver with inappropriate behavior.

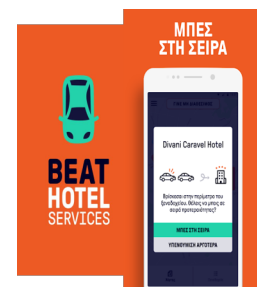


FIGURE 1.4: Beat Hotels

1.1.3 Hive

Hive is an e-scooter sharing system service. Scooters are made available to use for short-term rentals and can be dropped off or picked up from arbitrary locations in the service area. Hive has an app through which a candidate user, can find where e-scooters are in the map, how much battery they have and their cost, scan the barcode of the e-scooter, unlock and rent it.

However, as regards Beat's part in Hive service is limited. Beat only provide customer experience services by receiving e-mails from users and resolving their issues occurred either from e-scooters or the app. Beat also is responsible of placing the e-scooters in the right places and charge them.



FIGURE 1.5: Hive

1.2 Internship Goal

As regards internship's goal, is to gain experience as a Software Developer, learning tools like React, Redux and Node.js, understanding how an application works in production. Learning how to code in Javascript and improving a web site for BeatHotel's agents by completing tasks given, and creating npm packages, were my main responsibilities as an intern.

1.3 Report's Structure

The internship's report is an overview of what I have been interacted with during my internship, analyzing the projects and results, skills that I have gained or used, and my role as an intern in general.

Chapter 2

Basic Characteristics

In the following section I will describe the basic characteristics of my Department's structure and my role as an intern.

2.1 Department

The Department that I am part of is the Greek Market. Greek Market is managing the marketplace of Greece and is also referred as the Beat of Greece as mentioned in the previous chapter.

2.1.1 Role in the Company

The Greek Market is the only department focused on Greece. Its role inside the company is to manage any demands referred to this marketplace. Demands on marketing, finance, business analysis, customer experience and the development of product BeatHotels that is provided only in Greece, are all responsibility of this department.

The Beat App is developed by other departments and any changes required for the greek marketplace are forwarded from the Greek Market to the Engineering Department of Beat.

2.1.2 Department's structure

Greek Market is constituted by five teams, Operations, Customer Experience, the Engineer's of Beat Hotels named as GR Squad team, Marketing and Finance team. The General Manager of this Department is Vasilis Danias and the total number of people working in it is 37.

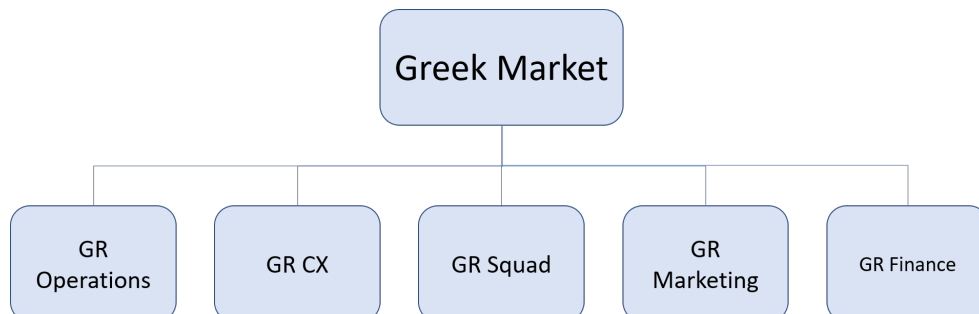


FIGURE 2.1: Greek Market's structure

2.1.3 Basic Procedures

Department's basic procedures are based on the management of three products in the borders of greek market, Hive, Beat App and Beat Hotels.

In more details, each team has different responsibilities. CX team is responsible for training drivers, resolving tickets and detecting any problems regarding these three products. The term tickets is any calls or visits made, or emails sent by either a passenger or driver.

Operations team is responsible for designing and controlling the process of production and redesigning business operations in terms of using as few resources as needed and meeting customer requirements. Marketing is creating, communicating, delivering, and exchanging offerings that have value for both customers and society in total. Competitions, sponsorships, banners, products or videos created for advertisement and social media management are made by this team.

Finance is responsible for beat driver's payments, while GR Squad is the team responsible for developing BeatHotels service.

2.1.4 GR Squad

GR Squad is a newly conducted team which is responsible for the development of BeatHotel. Team is consisted by eight people, three front-end developers, including myself, three back-end, one Product Owner and one Scrum Master following the agile culture as the other Beat teams.

BeatHotel is a service provided only in Greece and started about seven months ago. The technologies that are used for the development of BeatHotel's driver app, dashboards for each Hotel and Agents' dashboard, are React Native, ReactJS, Node.js and for databases Firebase and Redis.

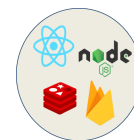


FIGURE 2.2: Technologies Used for BeatHotels

2.2 My Role

As a Software Developer Intern in GR Squad team, my role is releasing code that have immediate impact on BeatHotel service's users.

At the first one and a half month of my internship, I was a full stack developer and had the opportunity to work on both front and back-end elements of BeatHotel system. During this period, I was responsible to deliver npm packages, code in Node.js and ReactJS in order to complete requested projects for Agent's and Hotels' Dashboards, improve and extend tests and code coverage.

After this period of coding in Javascript for both back and front-end, getting familiar with BeatHotel service, my team and the way things flow, I had to choose between front-end and back-end developer. So, as a front-end I started to deliver tasks only referred to client-side development in order to maintain and extend existing web-site dashboards in ReactJS.

2.2.1 Skills Required

The skills required for this internship are enumerated below.

- Ability to produce high quality, maintainable and reusable Javascript code in React and Node.js
- Ability to build a three-layer web application
- Good knowledge of Unix based Systems
- Basic understanding of both Sql and N0-Sql databases
- Ability of problem solving and understanding algorithms complexity
- Familiar with GitHub Usage
- Ability to work in a team, communicate ideas, be an active member and deliver on time

Chapter 3

Projects/Activities

3.1 Introduction

3.2 Project 1

3.3 Project 2

3.4 Project 3

3.5 Project 4

Chapter 4

Results

4.1 Project 1

4.1.1 Description

4.1.2 Best Practices

4.1.3 Schedule

4.1.4 Problems occurred

4.2 Project 2

4.2.1 Description

4.2.2 Best Practices

4.2.3 Schedule

4.2.4 Problems occurred

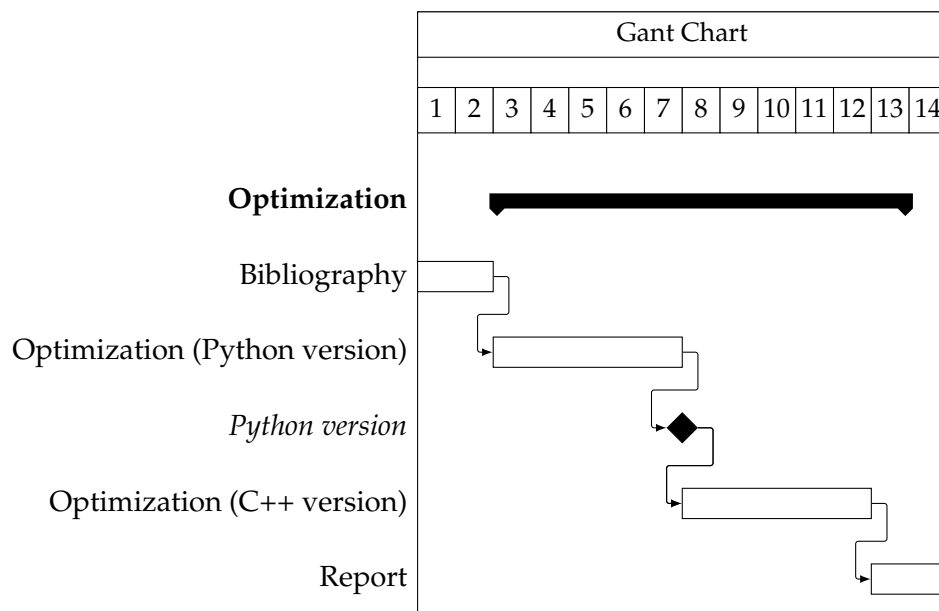
Chapter 5

Updated Time Management

5.1 Time Schedule

Activity	Duration
----------	----------

5.2 Gantt Chart



Chapter 6

Skills

Skills	Related Methodologies	Examples	Summary
Monday	11C	22C	A clear day with lots of sunshine. However, the strong breeze will bring down the temperatures.
Tuesday	9C	19C	Cloudy with rain, across many northern regions. Clear spells across most of Scotland and Northern Ireland, but rain reaching the far northwest.
Wednesday	10C	21C	Rain will still linger for the morning. Conditions will improve by early afternoon and continue throughout the evening.

References

- Abramov, Dan and the Redux documentation authors. *Redux Official Website*. URL: <https://redux.js.org/>.
- Accomazzo, Anthony et al. (2016). *Fullstack React*. URL: <https://www.fullstackreact.com/>.
- Cemus, Karel et al. (Oct. 2015). "Enterprise Information Systems : Comparison of Aspect-driven and MVC-like Approaches". In: *Proceedings of the 2015 Conference on research in adaptive and convergent systems (RACS)*. ACM, pp. 330–336. URL: <https://dl.acm.org/citation.cfm?doid=2811411.2811477>.
- Chen, Xuebin and Shufen Zhang (Apr. 2013). "Experiment teaching management system based on three-layer architecture". In: *8th International Conference on Computer Science and Education Computer Science and Education (ICCSE)*, pp. 1298–1302. URL: <http://eds.a.ebscohost.com/eds/detail/detail?vid=0&sid=1e19995b-4b66-4809-a426-e0a4ac683f32%40sessionmgr4008&bdata=JnNpdGU9ZWRzLWxpdmU%3d#AN=edseee.6554122&db=edseee>.
- Gallaughier, John M. and Suresh C. Rarnanathan (1996). "Choosing a client/server architecture: A comparison of Two-and Three-Tier systems". In: *Information Systems Management* 13, pp. 7–13. URL: <http://eds.a.ebscohost.com/eds/detail/detail?vid=1&sid=64174bb1-2f01-4844-b4a0-8eb1916d5792%40sessionmgr4008&bdata=JnNpdGU9ZWRzLWxpdmU%3d#AN=edselc.2-52.0-0642312740&db=edselc>.
- Geary, David (July 2016). *Introducing Redux*. URL: <https://developer.ibm.com/tutorials/wa-manage-state-with-redux-pl-david-geary/>.
- Google. *Angular Official Website*. URL: <https://angular.io/>.
- Kulesza, Raoni et al. (Apr. 2018). "Evolution of software architectures: From Web 1.0 to Web 3.0 systems". In: *WebMedia 2018 - Proceedings of the 24th Brazilian Symposium on Multimedia and the Web*, pp. 11–13.
- Mardan, Azat (2017). *React Quickly*. Manning. ISBN: 9781617293344. URL: <http://reactquickly.co/>.
- Masiello, Eric and Jacob Friedmann (Jan. 2017). *Mastering React Native*. URL: <https://www.packtpub.com/web-development/mastering-react-native>.
- Microsoft. *Microsoft Official Website*. URL: <https://docs.microsoft.com/en-us/dotnet/framework/wpf/data/data-binding-overview>.
- *TypeScript Official Website*. URL: <https://www.typescriptlang.org/docs/handbook/decorators.html>.
- Murray, N. et al. (2018). *Ng-Book: The Complete Guide to Angular 5*. Fullstack.io. ISBN: 9780991344642. URL: <https://books.google.gr/books?id=6\AqtAEACAAJ>.
- Naim, Naimul Islam (May 2017). "ReactJS : An Open Source JavaScript Library for Front-end Developement". In: URL: <http://www.theseus.fi/handle/10024/130495>.
- Padmanaban, R. et al. (2018). "Computability evaluation of RESTful API using Primitive Recursive Function". In: *Journal of King Saud University - Computer and Information Sciences*. URL: <https://doi.org/10.1016/j.jksuci.2018.11.014>.

- Pautasso, Cesare, Olaf Zimmermann, and Frank Leymann (2008). "RESTful Web Services vs. " Big " Web Services : Making the Right Architectural Decision Categories and Subject Descriptors". In: *Proceedings of the 17th international conference on World Wide Web*, pp. 805–814. URL: <http://wwwconference.org/www2008/papers/pdf/p805-pautassoA.pdf>.
- Ramirez, Ariel Ortiz (July 2000). "Three-Tier Architecture". In: *Linux Journal*. URL: <https://www.linuxjournal.com/article/3508>.
- Seshadri, Shyam (June 2018). *Angular: Up and Running*. O'Reilly Media. ISBN: 9781491999837. URL: <https://learning.oreilly.com/library/view/angular-up-and-9781491999820>.
- Source, Facebook Open. *React Official Website*. URL: <https://reactjs.org/>.
- Source, Ionic Open. *Ionic Official Website*. URL: <https://ionicframework.com/docs/intro>.
- Stefanov, Stoyan (July 2016). *React: Up and Running*. O'Reilly Media. URL: <http://shop.oreilly.com/product/0636920042266.do>.
- Vaqqas, M. (Sept. 2014). "RESTful Web Services: A Tutorial". In: *Dr. Dobb's Journal*. URL: <http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069>.
- Voutilainen, Jaakko (Dec. 2017). "Evaluation of Front-end JavaScript Frameworks for Master Data Management Application Development". In: *Metropolia University of Applied Sciences*, pp. 1–50. URL: https://www.theseus.fi/bitstream/handle/10024/138668/Voutilainen_Jaakko.pdf?sequence=1&isAllowed=y.