Ahmed Mansour

Building Scalable Web Applications

Researching Frameworks and Design Patterns

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

15 September 2016

Author(s) Title Number of Pages Date	Ahmed Mansour Building Scalable Web Applications, Researching Frameworks and Design Patterns 48 pages 15 September 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Kimmo Sauren, Senior Lecturer Ville Kuusela, (Senior Team leader)

A paradox of choice is a problem in web technologies. The purpose of the project was to research modern technologies to build large-scale web applications. The main goal of the project was to offer practical recommendations of web frameworks and design patterns that help developing a web application that is both scalable and maintainable.

The research was based on the requirements of a large application that was developed in an IT service company. An analysis of the application's requirements was made to understand its main challenges. The analysis led to recommendations to support the application's requirements.

As a result, certain web frameworks and design patterns were described. The project offers technical explanations and guidelines of how these technologies function. In addition, a practical implementation of a web application architecture was developed using some of the technologies described in the project.

Reusability of the code and loosely coupled components are the main factors to build a scalable and maintainable web application. The findings are limited to the technology available at the time this project took place. However, the methods that were applied to find these technologies will be reusable in the future to evaluate new ones.

Keywords	web frameworks, design patterns, Angular 2, React, Module pattern, Observer pattern, web development, architecture

Contents

Abbreviations and Terms

ASP.NET	Server-side web application framework made for web development. It is developed by Microsoft.
Ajax	Stands for Asynchronous JavaScript and XML. It is the use of the XMLHttpRequest object to communicate with server-side scripts. It can send as well as receive information in a variety of formats, including JSON, XML, HTML, and even text files.
Client	Visual user interface portion of an application in the context of web applications.
Closure	Powerful programming technique which is possible by having an inner function that has access to the outer (enclosing) function's variables—scope chain.
ERP system	Enterprise Resource Planning system is a category of business-management software typically a suite of integrated applications that an organization can use to collect, store, manage and interpret data from many business activities.
Git	A version control system that is used for software development and other version control tasks.
HTML5	HyperText Markup Language is Mark-up language that is used to structure and present content on the web.
Java	General purpose programming language.
JavaScript	High level interpreted programming language.
NDA	Non Disclosure Agreement
Material Design	A design-language developed in 2014 by Google. It makes more liberal use of grid-based layouts, responsive animations and transitions, padding, and depth effects such as lighting and shadows.

Open source	Development model that gives universal access to a product and allow its redistribution.
PHP	Server-side scripting language for web development could be also used as a general programming language.
Responsive web design	An approach while building website to provide viewing and interaction experience across a wide range of devices with different sizes.
Routing	The process of using URLs to drive the user interface (UI) in a web application.
Sass	A scripting language that is interpreted into Cascading Style Sheets (CSS)
Server	Device or program that provides functionality to other devices (client)
Static	Refers to a web design that can not adapt to viewer needs.
DOM	The Document Object Model is a cross-platform and language-independent application programming interface that treats an HTML document as a tree structure.
Webpack	An open-source JavaScript module bundler.

1 Introduction

1.1 Overview of the Project

The goal of this project is to study different web frameworks and design patterns commonly used within web frameworks. The case that is taken into consideration is an application that will be developed to one of Tieto's customers. The application has challenging requirements and it is unique to its case. It is necessary to do research first to decide which set of technologies to use and how to implement the building blocks of the application. A major requirement for the application is to handle large amounts of data without dropping the performance and maintain good customer experience. The application is expected to be used from different locations in North America, Europe and China. Issues such as localised languages, time zones and regional laws have to be taken into account.

1.2 Overview of the Company

Tieto is one of the largest IT service providers in the Nordic region. It consists of over 13,000 employees in around 20 different countries. The company is involved in various areas mostly offering their customers IT solutions and services. Tieto is heavily present in the following areas:

- Financial services
- Healthcare and wellbeing
- Retails and wholesalers
- Forestry
- Oil and gas

[1.]

Initially Tieto started its operations in 1968, under a different name at that time, developing IT systems for the Union Bank of Finland. During the late 1990s and the early 2000s the company experienced a huge growth and merged with many other companies to what is known today as Tieto. [1.]

Due to NDA it is not possible to write in detail about the company's own projects or software in the thesis.

2 Fundamentals of Building Web Applications

Building a web application on a large scale is an extremely demanding task. It often requires effort to decide how the application will be built out. Applications on the scale of Facebook, LinkedIn or YouTube have a long list of requirements and features. Questions like “which technologies are best suited for the application” or “how to structure the codebase of the application”, are often difficult questions to answer but also important questions that define the application and its chance to succeed in the market.

With the focus on the client-side, the Frontend, of web applications there are multiple decisions to make. Some of the major ones are choosing which libraries or frameworks will fit best with the requirements of the system, and which design patterns will be used to give a scalable architecture to the application and allow further development with minimum limitations.

There are plenty of different web frameworks available like Angular 1, Ember, Backbone, React, and JQuery. The frameworks vary with many different implementations of different patterns and features. However, when it comes to the client-side framework they are commonly created based on JavaScript.

Design patterns in software development and architecture have been developing over time in order to build more efficient and robust software. There are plenty of patterns available which each solve a specific problem. Often it is important to pick and match the patterns that support with an architecture that offer high performance, decoupled parts to ease changing dependencies of the application and ease code re-usability across the whole application.

2.1 Web Frameworks

In the scope of software development, a framework is a set of libraries or prewritten code that gives certain functionalities for applications. A framework can be considered the building base that engineers can build over. In most cases, the framework provides the application with a certain structure. The reason behind the popularity of the framework concept is that it speeds up the development of the project. Frameworks allow engineers to focus on solving the main problem instead of worrying about rewriting features and structures which are very common in most web applications. Frameworks provide functions and features which are usually well tested and proved to be working. Thus, the code is reliable to be used in production level. Nowadays, most web frameworks are free of charge to use and open sourced allowing anyone to join and contribute in a way or another. [2, 8.] Typically, a developer can be part of the development process of a framework. Contributions can be done by proposing improvements to the core team who is responsible for the framework. Also, reviewing the code and inspecting it for bugs is helpful. Another way of contributing, is getting familiar with the framework and offering assistance to others who have the interest in using this particular framework. [3.]

In 1990 the World Wide Web was created. The nature of the web at that time was very static. Interactivity between the user and the website did not much exist and the process of updating the content of a website on the web took plenty of manual work. First publishers had to edit their pages in their local environment and then to upload to the server. [2; 2.]

All data processing and application logic was done on the server-side. The browser was considered being only able to display content. If the user entered invalid data, it would be sent and checked on the server. Afterwards a message was returned and the page was re-rendered to its initial state. Things improved by allowing small processing chores inside the browser environment. For security reasons, any execution was within controlled area and had no access to the user's private data and was only limited to data within this page. The executable scripts were written in a language called JavaScript that quickly grew in popularity since it improved the experience of the user. [4.]

The years from 1990 onwards it could be considered a transition period in client-side development. The time prepared the web to adapt to changes such as the huge increase in usage of mobile phones. As a result, a concept such as Responsive Web Design was promoted and development of technologies such as HTML5 was accelerated in the period between 2005-2010. Another revolutionary milestone was introducing browsers that had their own full operating system. This innovation softened the line between PCs and browsers. For instance, through Chrome or Firefox and HTML5 web development, quality video game experiences could be effectively created right in the browser. Today the term "web application" has become generic and interactivity is a vital part of most website that has made them more complex but interesting to develop. [5.]

The following points will highlight the main advantages of using frameworks:

- Frameworks reduce the development time by using reusable code that was already written, tested and developed by other skilled engineers. The highest costs in any software development project is the time. Therefore, frameworks are beneficial financial-wise.

- Software development may include different developers throughout its development process. Frameworks often promote a design pattern or specific architecture. These design pattern and best practices make it easier for the developers to comprehend the code and fast start working with it.

- Frameworks make it easier for junior developers to learn design patterns and understand the logic behind different patterns. A junior developer often has a steeper learning curve when they use frameworks compared to learning a regular programming language. This is due to the fact that it is required to know the framework's own programming language as well as the specific syntax that the framework uses.

- The majority of frameworks are often open sourced meaning that the workload is divided among many skilled developers. Frameworks offer a variety of features so developers do not have to start from scratch and worry about all aspects while building the project. A framework's users get support free of charge with common issues such as localization, internationalization or security issues etc.

- By code modularity frameworks support "high level" of programming. For example, carrying out simple tasks like database handling or login can be in the framework but separated from other layers showing the business logic.

[2,11 - 13; 3, 1; 6.]

Frameworks have the positive aspects but they definitely have some negative ones as well. Here are the most common ones:

- Frameworks are built for a general purpose. Their common code is created in a manner that it is able to process as much as possible. On the other hand, it is not optimized specifically for a certain task and that affects the performance in some cases.

- To benefit from all the tools offered by the framework, it requires always the developers to invest significant time in educating themselves and have the appropriate background. Sometimes the learning curve is steeper if the developer is moving from one framework to another (with the assumption that the second framework is similar in size, the use of the programming language, the implementation of design patterns, and the architecture). Therefore, it always depends on the developer's experience.

- Over time, some frameworks tend to be stricter and do not offer flexibility in development. Choosing the right framework is a crucial decision. If the developers decide to use the wrong framework they will find themselves required to do extra work to work around certain challenges just to proceed with their tasks. In these cases, for the most part, it is not the framework's mistake but the developer's choice. Perhaps the framework was created for a different set of tasks other than what the developer had in mind.

- Developers have been surveyed and some have stated that building things from the scratch give them the feeling of higher productivity and along the way they feel more creative since they are not stuck with a certain framework or one way of doing things. It is a personal opinion and it varies from one developer to another.

- One of the main technical problems with frameworks is common bugs. Sometime, bug and security issues are detected while the framework is already being used among many applications. As a result, all these applications will suffer from the same problem. This puts the application at risk all the time. Frameworks have to be used with caution and only consider the professional ones with a skilled community around them.

[2,12–13 ; 3,2 .]

Reasons to Use Frameworks

Someone could start thinking of a new web application. However, it is often time-consuming to build web applications, which makes the idea challenging to develop further. In that case, it is smarter to spend less time developing a prototype to verify that the idea is worth the effort. Having a prototype application is the best way to get the reliable opinion if the final application will really succeed. In this scenario, frameworks are typically useful as they shorten the development time. Therefore, the prototype will be developed in a shorter time. [2, 13.] Otherwise, getting the application ready in shorter time without consideration for future development leads to what is referred to in the developer's community as "Spaghetti Code". Later it becomes a mission impossible to maintain the codebase of the application. This is when frameworks can become very beneficial since they offer a predefined structure that guides the developer how things could be done.

2.2 Design Patterns

Often programmers get a feeling that the problem they are facing seems familiar. Actually they have solved it before. However, they usually decide to come up with a new solution without realizing that they actually solved it before. Inventing the same solution is a common problem for developers. That is why design patterns exist, to avoid this same problem of being resolved again and again.

The aim of section 2.2 is to understand what design patterns actually mean, why they are used, and to point out some of the common design patterns that are applicable to JavaScript.

Architectural patterns generally focus on the end result of the development process and the relationships between different classes and objects. Even though architectural patterns are important they are not sufficient enough. It is recommended to have a re-usable design for a system. Therefore, each building block has to be developed in a manner that promotes re-usability and flexibility in interchanging its content. A lower level strategy needs to be implemented. Architectural patterns are not very helpful if the building blocks are not built properly. Thus, optimal design patterns complement a good architectural design.

Architectural patterns are helpful when it comes to separation of concerns between the model and the view. They are described further in section 3.1.1 as part of the criteria to evaluate a web framework.

The challenge to most experienced developers is designing solutions that are problem-specific and also generic enough to handle future problems and requirements. Building object-oriented applications is hard. However, it is even harder to have a loosely coupled reusable object-oriented application. It is always undesired to redesign solutions. Flexibility is needed to minimise redoing the work twice. Developers always try to solve problems by reusing previous solutions again instead of trying to solve every single problem repeatedly. [7, 11.; 7.]

One generic way to define a pattern is the following, according to Raplh Johnson (1994): “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.” [7, 12.]

2.2.1 Fundamental Elements of Patterns

Every pattern has a name, problem, solution and consequences. Each element will be described in this section:

- Name: A pattern’s name is descriptive enough to tell the problem being solved, the proposed solutions and consequences in couple of words. The name is used for discussions between different people and in documentation. [7, 12-13.]
- Problem: The main problem that is being solved. It generally explains what is the issue, its context, and when to apply the pattern. Depending on the pattern, some problems are very specific with a list of conditions that have to be present for that particular pattern to be applicable and other patterns are more generic. [7, 12-13.]

- Solution: A proposal where pattern offers a template that can be applied in different scenarios. The solution is an abstract description of an arrangement of components that solve the problem. [7, 12-13.]
- Consequences: The result and the sacrifices of using the pattern are typically discussed in this part. Since it is not often very clear which pattern is the best when it comes to choosing one. A comparison of different patterns' consequences helps to choose a better fit for the problem present. [7, 12-13.]

2.2.2 Reasons to Use Design Patterns

The major benefit of using design patterns is that they give a standard terminology and get developers accustomed to certain kind of situations. Therefore, design patterns could be considered as a common platform for software engineers. [9, 7-8; 6, 11–16.] For instance, an observer design pattern unifies event handling to a single point per event that notifies the observers once the event occurs.

There has been development in design patterns for a long time. They provide solutions and best programming practises to certain common problems that are faced in the process of developing an application. In that manner, developers who have not experienced a particular problem before could make use of the design patterns that focus on solving their problem.

Design patterns make parts of the codebase reusable and easier to integrate into other systems. They aid developers to choose an option that makes the code reusable instead of an option that is not very flexible. As a result, design patterns support with designing the system in the right manner faster. [8.]

3 Criteria for Research

Web frameworks and design patterns are considered two major tools in web development. These tools boost development time and ease the process of building large web applications. Web frameworks and design patterns are often interconnected to each other. Modern frameworks often adopt different design patterns in its architecture to give best programming practises. Design patterns are applicable solely depending on the nature of the problem encountered in the application. Therefore, experienced developers must always consider the flexibility of web frameworks and which design patterns the framework promotes. Using very strict frameworks is not recommended.

The sizes of frameworks vary just as the nature of problems faced during software development. Questions that are often asked when a new application is being built are “Which framework to use?”, “Is this framework the best solution to my problem?”. In the same manner, different design patterns and architectural structures each have their own benefits only depending on the requirements of the application. This paradox of choice, creates a need for general guidelines to help evaluate a particular framework and design patterns to use while developing the application.

3.1 Criteria for Web Frameworks Research

Frameworks are all marketed to the developers as the best solution with many possibilities. It is extremely difficult to decide which framework is the right one to use. Especially that it sometimes occurs that a framework does not perform as promised.

Often when a framework is being considered for development a long list of features is advertised. However, there is no general criteria to evaluate the frameworks. This makes it difficult to decide if that particular framework will be a good choice for the application. This problem urges to compile a list of features and certain points to evaluate a framework. The list has to contain the common features of web applications and common concerns that most developers have. More specifically, it has to include some of the functionalities that promote rapid development and lower the overall time of development, testing and maintenance. This generic list includes technical and non-technical points which should guide developers to the important parts in any framework. Each point will be discussed in details in the sections below.

Architectural Patterns

Architectural patterns can be defined as solutions which can be reapplied in commonly occurring challenges in software design. They are similar to design patterns but on a higher level. Every complex large system gets deconstructed on different levels to simplify its complexity. The system gets deconstructed on the high level using architectural patterns. In addition, it also gets deconstructed on lower level using design patterns. Design patterns are discussed in section 2.2 to illustrate how they work on the low level code. [10.]

Architectural patterns are tools that are related to larger portions, or components, of an application. Architectural patterns also are related to the global properties of the system. As a result, it is essential for a framework to adopt some architectural pattern. These patterns enforce a form of structure that increase scalability, reusability, and maintainability of the system. Architectural patterns like MVC (Model-View-Controller), MVVM (Model-View-ViewModel) and MVP (Model-View-Presenter) are currently used by many developers for the front-end. It depends on the case of the application which pattern would be the best choice. Therefore, all the three patterns will be briefly explained to illustrate their key differences. [10.]

In MVC, the model part manages the data for an application. They are mainly responsible to represent the data that an application needs. The Views are the visual representations of the model's current state. A view looks after any model changes and updates itself accordingly. The controllers are considered as the middle layer between the models and the views. The controllers are taking care of updating the model if the user manipulates the view. Typically, all the business logic is stored in the controllers. [10.]

In MVP, the P represents a presenter. This component holds the business logic of the user interface for the view. Invocations from the view are taken care of by the presenter which typically passes it to an interface. This gives the chance to mock the view in unit tests. [10.]

Since the MVP pattern increases the testability of the application, it is often used in the enterprise-level applications. MVVM pattern is based on MVC and MVP. It tries to stress clearly the separation of the user-interface from the business logic in an application. [10.]

Frameworks that adopt an architectural pattern are recommended. General criteria to evaluate web frameworks are to have a framework with an architectural pattern that is flexible enough to allow the development of the application. The pattern must allow on a high-level maximum reusability of the code, enforcement of the separation of concern principle, and simplification of the application.

License

Most frameworks are distributed under licenses that give the developer some freedom to use it for commercial applications. But then there are some which are not very generous. The worst scenario that can occur to one is developing the whole application and find out that the license does not allow the developer to distribute it commercially. It is highly advised to read the license terms properly before starting the development. [11.]

Learning Curve

The flexibility offered by frameworks vary tremendously. Some frameworks are loose when it comes to naming conventions, directory structure etc. However, other frameworks could be so strict that they keep throwing errors at the smallest of problems. Programmers generally try to follow the general conventions when a feature gets implemented. However, some may just decide to implement it their way. It is advised to choose a framework that requires the smallest possible learning curve. However, a big learning curve can be justified if the framework is offering a lot of positive features to the application.

Unit Tests

Unit testing is a crucial part of professional development. Unit tests are the way for the programmers to validate their work. Unit tests are used to ensure that the source code of single component/unit is functioning properly in the application. Frameworks that allow unit tests are considered better and some frameworks offer the possibility to write custom tests besides the basic tests to verify the reliability of the vital parts of the application. [12, 7-8.]

Documentation

A well-documented framework will have a plenty of examples, sample code, articles, tutorials and snippets. The documentation itself has to be clear and detailed enough that early users and evangelists understand how to use the framework. Documentation is the main factor for the success of a framework. Choosing a framework with poor documentation will lead to problems with understanding how to use each feature. [13.]

Community

Regardless of having a well-documented framework, problems always appear over time. Seeking assistance from the community behind that particular framework is the usual way to solve these problems. There have been two extremes in the development world. Some communities are very aggressive towards new users and tend to be tough on them while others are cheerfully welcoming new users and guiding them. Therefore, choosing a framework with a friendly community is preferred in the long run. Communities are often determining whether a framework will succeed or fail.

Browser Support

It is extremely important to verify that the chosen framework supports the browser requirements given by the project. Sometimes, clients request support to an old version of a particular browser. Internet Explorer 8 is an example. The framework must be checked that its features are compatible with the required browsers.

3.2 Practical Guide to Research Design Patterns

Design patterns are always made to a specific problem with specific circumstances. In that sense, it is difficult to come up with some generic criteria to evaluate different design patterns as it was the case with web frameworks. Every time new code is written it has a different challenge. However, different factors have been taken into consideration while choosing which design patterns are going to be studied further in this thesis.

How applicable a design pattern to JavaScript has been one factor. As stated, JavaScript is the dominant programming language for client-side development. How much a design pattern is actually being used has been considered too. The aim was to find design patterns which are solving the most common problems. Preferably the design patterns would be also as generic as possible. Generally, the design patterns were researched based on the challenges that were encountered while developing the case study application.

The general approach to find an applicable design pattern:

- First, the problem needs to be identified.
- Second. a research is done to find the solution.
- In case of finding different alternatives, a comparison of each solution is needed. The consequences of each solution and its trade-offs have to be considered.
- Third, the solution has to be integrated with the code that had the problem.
- Fourth, a verification that the problem has been resolved is needed to ensure that the solution worked.

[9, 8.]

4 Design Patterns for Maintainable Applications

The design patterns described in this chapter are the Observer Pattern, Module Pattern and revealing module Pattern. Practical examples and code snippets of the chosen design patterns are listed. The aim of this chapter is to give a practical description of how these design patterns are implemented in different circumstances. Reasons to use each pattern and when they are applicable are described as well.

The selection of the design patterns was made based on the most common challenges encountered while developing the case study application in Tieto. However, the selected patterns had to be as generic and as common as possible so that they are applicable to wide variety of large web applications.

The case study application had an interactive nature with large number of events occurring. Often, one event, in the context of web applications, had multiple reactions in the background and the state of the application had to be updated constantly. Many parts of the application were used in multiple areas across the entire application. Therefore, reusable modular components and reactive programming style were necessary to be implemented.

As one criterion, the selected design patterns had to be easily integrated with the frameworks described in chapter 5.

4.1 Observer Pattern

The Observer pattern gives a model where several objects, known as observers, are able to subscribe to a certain event and get a notice every time this particular event takes place. That subscribe-able model is often in a form of an object, and could be referred to as a subject, that maintains a list of all its observers that depend on it. This pattern is one of the building blocks of what is to referred to as event-driven programming in JavaScript and other programming languages. [10.; 13.]

In the case where a subject has to notify its observers about a certain event happening, it just has to broadcast a notification to its observers with information related to the event. [15, 215.] When an observer does not need to be notified of these events by its subject anymore, the subject may delete that particular observer from its list of observers and this way the subject does not notify that observer of any future events. An Observer also may unsubscribe from a subject. [15, 219-220.]

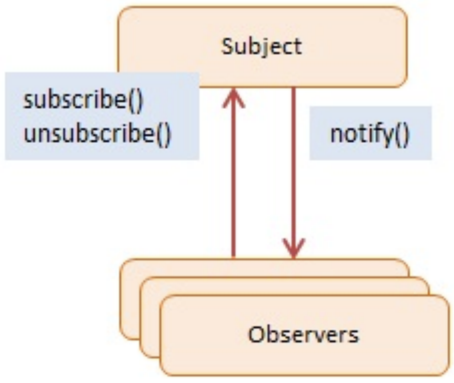

Figure 1. Illustration of the connection between the subject and observers. Copied from Harmes [15, 216.]

In reference to Figure 1, the relationship between a subject and observers is one-to-many in terms of its dependencies. The Observer design pattern comes hand in hand with a good object-oriented design and it gives the opportunity to loose coupling between different objects in the code base that need to interact together. [15, 215.]

In order to be able to get a broader sense of the use cases of that pattern, a general definition of the pattern has been given in a book named, Design patterns: Elements of reusable Object-Oriented Software: “One or more observers are interested in the state of a subject and register their interest with the subject by attaching themselves. When something changes in our subject that the observer may be interested in, a notify message is sent which calls the update method in each observer. When the observer is no longer interested in the subject's state, they can simply detach themselves.” [10.]

It is very common to build web apps that contain multiple event handlers. Event handlers are basically functions that listen to a certain event and wait until it gets notified when that particular event has happened. [10.] The notification that is delivered to the event handlers usually contains details about that particular event. For instance, a “click” event is fired once an element on the page has been clicked by the user. The event may return information like the “id” of that element to the event handler function listening to it.

With reference to the event-handler and the event paradigm in JavaScript, the paradigm could be considered as a manifestation of the Observer pattern. Publication/Subscription or Pub/Sub are used to refer to design patterns that are similar to the Observer pattern. [15, 216.]

In the client-side world where JavaScript is mostly used, the Observer pattern is commonly implemented in form of Publish/Subscribe pattern. Even though this pattern is very similar to Observer pattern, there are few differences among both of these patterns. The Observer pattern works in a manner where the observer object needs to listen to events from the subject. The observer needs to subscribe to that particular subject. In contrast to that, the Publish/Subscribe pattern works in a way where there is a specific topic or event channel that is found between the publisher and subscriber. Publisher fires the events and the subscriber gets the information that the event occurs. This implementation allows custom arguments to be available to the subscriber by giving the possibility to the code to define specific events which are specific to that particular application. [14; 10.] The main difference is that the Publish/Subscribe pattern allows any subscriber that have a valid event handler to listen to a publisher and receive the events broadcasted. This idea removes dependencies between both the subscriber and the publisher therefore making them less coupled.

The code snippet in Listing 1 is a simple example of creating a Publish/Subscribe model:

```
//simple new message handler

//Count storage for the amount of messages received
let messageCounter = 0;

// "newMessage" will be the name to initialise subscribers which will listen
that kind of event

// Render a prevew of new messages
let subsriberA = subscribe("newMessage", function (header, body) {
    //logs to the console
    console.log("A new message was received ", header);
    // Use the data that was passed from our subject
    // to display a message preview to the user
    $(".messageSender").html(body.sender);
    $(".messagePreview").html(body.content);

});

// Here's another subscriber using the same data to perform
// a different task.

// Update the counter displaying the number of new
// messages received via the publisher

let subscriberB = subscribe("newMessage", function (header, body) {

    $('.newMessageCounter').html(++messageCounter);

});

publish("newMessage", [{
    header: "hello@google.com",
    body: "Hey there! How are you doing today?"
}]);

// We could then at a later point unsubscribe our subscribers
// from receiving any new topic notifications as follows:
// unsubscribe( subscriberA );
//or
// unsubscribe( subscriberB );
```

Listing 1. Creating subscribers and publishers [10.]

4.1.1 Advantages

In a scenario where a web application contains a large number of events occurring, following the Observer pattern is a good way to keep the application functioning without cutting the performance. It is typical for any application to have hundreds if not thousands of events happening in one session. Therefore, having observable objects to listen to the actions through one event listener and then send all the data to all its subscribers. As a result, there is no need to have multiple new listeners to that same elements which affect the performance of the application tremendously and makes the application unmaintainable. [15, 223.]

Another benefit for using that pattern is that it encourages developers to consider the relationships between different parts of their application. It gives a chance to find the layers that contain direct relationships that are getting strongly coupled and replace them with a group of subjects and observers. As a result, the application will be broken down into smaller, more maintainable, loosely coupled parts which have the potential to be reused in multiple areas in the codebase. [15, 223; 10.]

4.1.2 Disadvantages

Problems that are associated with the Observer pattern actually are related to its main advantages and the manner it is set up. Since different areas of the application are loosely decoupled, ensuring the quality of different parts of the application becomes difficult as we can not assure separately that each part is working the way it is intended to work. [10.]

A scenario that elaborates the problem is as follows. A publisher with one or more of its subscribers are listening to it. It is safe to always make the assumption that at least one of them is listening. The subscriber that is supposed to perform a function fails or crashes for any reason. At this point, the publisher will have no way of knowing that this has happened since they are decoupled and basically blind to each other.

Another side-effect of applying the Observer pattern to an application, is that setting up these observable objects is costly in terms of application load time. Therefore, having a big number of observable objects with different observers will slow down the application. [15, 223.]

The Observer or Pub/Sub patterns may come with costly drawbacks in some cases. Therefore, they should be used with some caution. One way around the application load time issue is to use a technique called “Lazy loading”. Lazy loading works in a way that it only loads parts of the application when needed. This way only the necessary observables would be initialised and not all of them at the same time.

The Observer pattern may not be the absolute best solution to all problems. However, this pattern definitely could be considered as one of the best tools when it comes to designing decoupled systems. It is recommended for any developer who writes JavaScript application to consider implementing the Observer pattern or some variation of it.

4.2 Module Pattern

The module pattern is a commonly used pattern that promotes state, privacy and code organization which are all possible by using closures. The module pattern gives a platform to wrap public and private methods together with variables. It ensures that no code get leaked into the global scope and run over some other code which could be written by another programmer. The pattern functions in a manner that it provides only a public API and everything else is kept privately in the component using closures. [10.]

This implementation gives a decent solution to protect the core logic of each component while assembling different parts of the application together. Thus, only an interface is exposed to the outside world. There are multiple ways to apply this implementation in practise and those are defined in details in this chapter.

In JavaScript, the privacy of variables does not exist as the language does not have any access modifiers. As a result, variables in JavaScript can not be declared as public or private like in more traditional programming languages like Java. To have this idea implemented in JavaScript, function scope is used so that variables in a function are only accessible within the closing braces of that function. Closure is what makes it possible in JavaScript to implement the module pattern since variables and functions that are declared are only available within that module which acts as a large function. The public interface becomes available by returning objects outside the function which become usable by other modules.

It is essential to understand the theory behind modules to be able to correctly implement the module pattern and benefit from its advantages. It is also essential to understand object literals since the module pattern is partially based on them.

4.2.1 Module Theory

Modules are often used in most current architectures of web applications that are being developed today. Modules in that sense represent integral parts of any application's codebase and usually they have a single purpose in the whole application. Often modules are easily interchangeable. If modules are built in the right manner, they are self contained in that way that could be reused even in another application if they require a similar module. [10.]

A module may have its dependencies defined which allows that module to have its dependencies fetched automatically right away. From scalability point of view, implementing modules in that manner with their dependencies defined makes it much easier to keep track of the different dependencies and avoid manually loading all the modules or including a script tags to manage all these various dependencies. [10 ; 8.]

Facebook is a good example to explain modules. Facebook's news feed, chat room, notifications, and events list are modules which are independent and do not have any dependencies on other parts of the application. However, depending on how demanding the logic of a module is, it may require very complex implementation. It is common to create more of sub-modules that act as a dependency for all other parts of that particular module. The module of Facebook's chatroom has probably a sub-module like emoticons which are shared also with the news feed module.

Within the module architecture, a module typically has a very limited information about what is happening in the rest of the application. Therefore, this responsibility needs to be handled by another part. A good option for responsibilities delegation is using other patterns. A module has to take responsibility of sending announcements to the application when something of interest occurs and the module should not take into consideration the state of the other modules. This way it becomes easy to add or remove modules without the risk of the rest of the application failing.

Loose coupling is a requirement to make the modular design to be functional. It allows the modules to be maintainable by removing some dependencies if they are not needed. Therefore, modules should not be relying on each other in order to work as intended. It becomes relatively easy to track the changes that may occur to the application when one part is changed. [8.]

4.2.2 Object Literals

With reference to the object literal notation, an object could be defined as “a set of comma-separated name/value pairs enclosed in curly braces ({})”. [16.]

```
var person = {
    key: value,
    firstName: "John",
    lastName: "Doe",
    age: 50,
    functionkey: function () {
        //function's code goes here
    },
    eyeColor: "blue"
};
```

Listing 2. Object person

As Listing 2 illustrates, names inside the object could be either identifiers as keys, or strings as values. Each key/value pair is followed by a comma unless it is the last pair to be defined in the object.

Listing 3 is a simple example to show how objects could assist with implementing a module and keep the code private and only allow one public method. The code is written with Ecmascript 6 which is newer version of JavaScript.

```
let myGradesCalculator = ( () => {

    // Keep this variable private inside this closure scope
    let myGrades = [93, 95, 88, 0, 55, 91];

    // Expose these functions via an interface while hiding
    // the implementation of the module within the function() block

    return {
        failing:() => {
            let failingGrades = myGrades.filter( (item) => {
                return item < 70;
            });

            return 'You failed ' + failingGrades.length + ' times.';
        },
        avg: () => {
            let total = myGrades.reduce((accumulator, item) => {
                return accumulator + item;
            }, 0);

            return 'Your average grade is ' + total / myGrades.length + '.';
        }

    }
})();

myGradesCalculator.failing(); // logs 'You failed 2 times.'
```

Listing 3. myGradesCalculator module

Similarities could be found among the usage of object literals and other methods such as “immediately-invoked functions “. Main difference is that modules implemented with object-literals do not return functions but an object is returned instead.

4.2.3 Advantages

With reference to JavaScript’s lack of ability to declare public and private variables and functions, the module pattern assists with organizing the code that grows when an application is scaled up. Whether the module is used with JavaScript or other languages, the pattern promotes self-contained implementations which are highly beneficial. Thus, it makes major parts of the system loosely decoupled. Each module push only one object to the global namespace, or the global state, which keeps the global namespace much cleaner. Modules interact with each other through the “main” method or a facade. Having less clutter in the global namespace reduces the possibilities that other libraries and frameworks collide with the code of the application. [16; 9.]

4.2.4 Disadvantages

The pattern functions in a way that it only exposes a public interface. As a result, the module's behaviour becomes a mystery which is difficult to unit test. The modules often do not offer any testing interface that could allow testing its sub-modules. When modules are used they should just be trusted that they behave the way they are intended to do. Unless the inner code of the module is tested, there is no other way to verify that the module functions in special corner cases of the application. [10.]

The downside of module patterns could partially be overcome with changing the implementation of the regular module pattern by using objects literals or immediately-invoked functions. [17.]

4.3 Revealing Module Pattern

The Revealing Module pattern was created as a response to the frustration of using module pattern. Since programmers were forced to have the name of the main object repeated every time calling public function from another function. Also accessing public variable names was needed. This fact violated one of the basic fundamentals of Software engineering which is "Don't Repeat Yourself" or DRY as some may refer to it.

Programmer Christian Heilmann decided to put effort into updating the module pattern. His implementation worked in a way that all the functions and variables, which are in the private scope, must be defined. Afterwards, an anonymous object is returned with pointers to these private methods and variables that are needed to be revealed to rest of the application. [17.]

Listing 4 shows how to re-implement the same module of "myGradesCalculator" but using the revealing module pattern.

```
let myGradesCalculator = ( ()  => {

  // Keep this variable private inside this closure scope
  let myGrades = [93, 95, 88, 0, 55, 91];


  let failing = () => {
    let failingGrades = myGrades.filter( (item) => {
        return item < 70;
      });

    return 'You failed ' + failingGrades.length + ' times.';
  };


    let avg = () => {
    let total = myGrades.reduce( (accumulator, item) => {
      return accumulator + item;
      }, 0);

    return'Your average grade is ' + total / myGrades.length + '.';
  };

  // Explicitly reveal public pointers to the private functions
  // that we want to reveal publicly

  return {
    average: avg,
    failing: failing
  }
})();

myGradesCalculator.failing(); // will log 'You failed 2 times.'
myGradesCalculator.average(); //  will log 'Your average grade is
70.33333333333333.'
```

Listing 4. Another implementation of myGradesCalculator

The Revealing Module pattern must be used with caution. If where all the methods are exposed, then the module does not become very stable anymore. Methods which are created based on this design may be more fragile in comparison to the other patterns but it also partially solves the drawbacks of the other patterns. Depending on the requirement of the system and the environment, the revealing module pattern may be very helpful. On the other hand, it could be the main cause to not have a very robust system, if it is not implemented right.

5 Scalable JavaScript Web Frameworks

In this chapter, two JavaScript web frameworks will be analysed in detail. These two frameworks are React from Facebook and Angular 2 from Google. The analysis will give a technical overview of the different functionalities and technologies. The major benefits and drawbacks of these frameworks are described as well.

There are multiple reasons why these two frameworks are chosen for this analysis:

- Both frameworks promote solid architecture patterns and provide excellent documentation, have a relatively acceptable learning curve, and are supported by majority of browsers and their older versions. Thus, they fulfil all the criteria specified in section 3.1.
- At the time of conducting this study, these two frameworks were the most widely used by professional developers to build web applications on a big scale. As a result, they have huge communities around them both. Even though Angular 2 is in release candidate stage currently, it has already a reasonably big community and it is expected to grow even bigger once developers upgrade from Angular 1.
- Both of these frameworks have big corporations working on developing them further, thus making the frameworks more reliable since they will be supported for longer period of time by their sponsors.
- Angular 2 and React both promote component-based development. Components design is basically an implementation of the module pattern. Component-based development leads to designing the architecture of the application in a manner that makes the code of the application both reusable and loosely coupled.

The aim of this chapter is to give a practical understanding and an example of using a web framework that is built to develop large-scale web applications. The aim is also to give a general idea what kind of features such frameworks offer and how in practice they could boost development time.

5.1 React

React was released in 2013 by Facebook and it is generally taking a different approach compared to Ember and Angular 1 which often are named MVC client-side frameworks. React does not precisely fit under the MVC category. It is being marketed as more of the V or the view part in MVC. The rest of the needed parts of the pattern are flexible. Often Flux or Redux architectures is used to fill out the needed parts. Examples of applications using React are Facebook, Instagram, BBC, and Netflix. [18, 5-6.]

Developing a web application in the early days followed a certain process which was the only possible way at the time. First a request from the application would be sent to the server and then the server would send back the content of the full page. This was considered a very simple process from a technical point view since it did not include any events occurring in the browser due to interactivity from the user. [18,18]

This have made languages like PHP popular since it allows server-side app development in an easy manner. Creating functional components in a language like PHP is a simple task. Therefore, many developers have written different PHP components that were easy to reuse code and could be implemented in other applications as well. [18, 20.]

However, this process was a fundamental step to create a good experience for the user of the application. Every single time a new action was taken, a new request to the server had to be sent and afterwards get a response sent back. In addition, data processing on the server erased the clientstate that has been created in the application. [19, 16.]

With the goal of making a better experience for the user, developers started to write different libraries to assist with loading applications in the browser using JavaScript. These libraries had different styles of manipulating the DOM from simple HTML templates systems to other systems that controlled the whole application. As a result, this environment created a state of instability as the applications depending on such libraries have scaled up. It became very difficult to control all different events which made it much more difficult to control in comparison to the PHP style of development. [18, 22.]

Development of React started with purpose to take the PHP’s style to load the whole page on the client side applications. Basically React could be considered a “state machine” assisting the developer to deal with the issue of the changing state. It solves this problem by focusing on a very narrow scope. It only cares about updating the DOM and responding to events. [19, 19.]

5.1.1 Virtual DOM

Features such as AJAX, routing, storage or data-structures are not included in React. This makes React not a Model-View-Controller framework but more like the V part in the MVC architecture. Therefore, React is very flexible and has the ability to play along a big variety of systems. React is commonly used in other MVC frameworks as well to render the views. Every time the page is reloaded when an event occurs, the process is extremely slow with JavaScript. This is because the DOM is trying to read the page and update its content. However, React offers a very powerful concept called virtual DOM. React only have to update the DOM and check the changes from its virtual DOM. This feature is one of the main reasons why React is very quick when it comes to performance in comparison to other frameworks. [20, 41 ; 18, 20.]

The virtual DOM works in a manner where it takes the state of the DOM and translates it into virtual representation using a render functions. Once a state-change has occurred, these functions are triggered again. The triggered functions find a new virtual representation of the page that is equivalent to the real page after the state-change. Instantly the virtual DOM is updated in alignment with the necessary changes. These changes are applied to the actual DOM showing the new view. This implementation was criticised when it was first introduced.

The criticism was based on the assumption that it would take a longer time in comparison to the usual JavaScript approach of modifying every element on needed basis. Nevertheless, what happens in React behind the scenes is that it does exactly the same thing as it was being criticised for. However, what makes it very efficient is the algorithm which finds the different parts between the current and the previous versions of the virtual DOM. Based on the results coming from this algorithm, React renders the minimum amount of updates needed to the DOM. As a result, React offers a high performance since it avoids unnecessary DOM manipulations which is one of the main reasons behind poor performance for any JavaScript based library/framework. [19, 36 ; 21.]

The main problem that React has been aiming to solve is the chain of state-changes. Having a big scale web application built on the client side gives a high chance that one interaction leads to an update which triggers a different event leading to a third update. Handling these updates and batching them properly must be taken into account. Otherwise, the performance of the whole application starts to drop. React's virtual DOM minimises these problems by taking only necessary actions in one step. As a result, it makes the process of maintaining the application easier. In situations where a state has changed because a user has clicked a button or an external event have occurred, the developer needs to only let React know that the state has experienced a change. At that point React takes control and automatically updates the rest depending states. React uses one event handler for the whole application and passes all the occurring events to that particular event handler. This technique boosts React's performance.

5.1.2 JavaScript XML

JavaScript XML is often referred to as "JSX". The main purpose of the components in React is to implement one of the programming principles called the separation of concerns. The components are not used for creating templates or for displaying logic as components in other libraries and frameworks are used for. A fundamental idea in React is that the mark-up language and the scripting code are tied together. This way while building the mark-up for the application, all the expressive power of JavaScript will be available to support it. [22.]

React gives a high level API to generate the virtual DOM. However, developers might experience problems in events where complex structures are being generated. This is where the use of an intermediate format becomes a smart solution. Facebook's JSX is one common format that is being used in React. JSX is a superset of JavaScript that gives you the possibility to mix syntax similar to XML and JavaScript. It is used to construct the mark-up inside React's components. The API given by React is used to serve this purpose but using JSX instead makes the components more readable and therefore more maintainable. [19, 53-60.]

It is common that JSX syntax is odd to many developers who have never used it before. The fact that the mark-up is being written within JavaScript looks confusing at the first sight. However, there are many benefits of using JSX and it becomes very logical once developers get familiar with it. Some of the benefits are:

- It provides an easy way to show the structure of the application.
- It gives a layer of abstraction to create a React Element.
- It puts all the code and the markup related tightly close to each other.
- It is plain JavaScript so no new syntax need to be learnt.
- It makes the markup easier to read and understand.

[21;20.]

5.1.3 Data Flow

React makes a paradigm shift from the usual way of building applications and puts some interesting concepts under the spotlight. JSX is one of these paradigm shifts. Especially, that the HTML mark-up and the JavaScript code are being used together at the same place. The methods used for data flow in React are another unusual concept to many developers as well. That's why, these methods may not be very clear at the first glance. [18, 69 – 73 ; 19, 65 - 69.]

The data flows only in one direction between all React's components. Data always comes from the parent component to the child component. The fact that data flows in this manner makes the components easy to understand. Therefore, React's components have an easily predictable state. Every component uses props, "properties", as a way to pass data from the parent to the child. A typical scenario will be that there is a component that takes props from its parent and renders the view. When that property changes at a top-level component, it is propagated down the component-tree. Therefore, the component used earlier will be re-rendered since it was using that property. Components can also have internal state which is different from props. It can be considered a presentation of how the data of a component may look like anytime. The state is modified only internally within its component. [19, 68-80; 22.]

5.1.4 Lifecycle of the Component

A React component is a state machine and in all scenarios it should return the same output for a given input. In the lifecycle of the component, there are multiple changes in its props, in its state and even in its DOM representation. React gives lifecycle hooks for its components to be able to react accordingly to different changes throughout its stages starting from the creation and the lifetime to the teardown. [23.]

Examples of different lifecycle hooks are briefly described below:

- Instantiation: First stage in a component lifecycle where a new component is created and rendered for the first time. Different methods are available to control the state of the component when it is created. “getDefaultProps”, “getInitialState”, “componentWillMount” and “render” are all examples of the available methods in this phase. [18, 50-54.; 20, 80-82.]
- Lifetime: Once the component is rendered for the first time, other methods become available to give further flexibility to control the component. Examples are “componentWillReceiveProps” and “componentWillUpdate” that allow manipulation of the state of the component. [18, 56-59.; 20, 123-127.]
- Teardown & clean-up: This life cycle occurs once the component is not needed any more and need to be removed from the view. Method “componentWillUnmount” will run right before the component is being removed and gives the opportunity to clean up or send data if needed. [18, 59-62.]

React’s methods to manage the component’s lifecycle are well-designed. Since components are considered state machines, they are made to give a stable output and predictable mark-up at each stage of their lifecycle. Since the parent components pass on props to their children and these children render their own children components, it is rare to have a component that lives in an isolated environment. Caution is needed when designing how the data will flow through the application.

5.1.5 Final Conclusions

React is relatively easy to learn. Developers do not experience difficult learning curve. React contains less of what is commonly known as “domain-specific language”. [17, 6.] Thus, developers who have prior experience with JavaScript will have an easy time to understand how it works. With a component-based approach, React maximizes the code reusability. Each component represents a single part of the user interface, like a form element, a search button or a page title. [18, 79-81.]

When it comes to performance and speed, React is considered extremely fast. The implementation of a virtual DOM takes credit for React’s speed. The virtual DOM is a local simplified copy of HTML’s DOM that React stores. Instead of doing the “real” DOM operations which often slow down the user experience. React allows developers to do all the operations needed inside this abstract DOM instead of the “real” DOM. [19,81-82.]

React also supports server-side rendering. This contributes to high performance and solves Search Engine Optimization problems which are experienced with frameworks that do not support this feature. The way it is implemented includes React having the first render done on the server. Afterwards, all updates taking place in the UI occur on the client-side. [18,152-158.]

React Native is an exclusive advantage that comes with React. React Native gives possibility to create a mobile application on both iOS and Android using the same components used to create the web application. [18, 28–30.]

5.2 Angular 2

Angular 2 is a newer version of the Angular web framework that is being developed by Google. There are two major versions of Angular. Angular 1 was the first known stable version of Angular and it has been used in production already by big companies such as Vevo and YouTube for PS3. Angular 2, on the other hand, is the newer release from Google. Angular 2 was in Release Candidate stage in the time of writing. Angular 2 has very different features and solely for marketing purposes it was named “Angular 2” but the whole layout of the framework has been reformed. Therefore, Angular 1 and Angular 2 have be addressed separately and considered to be different frameworks. [24.]

The background of Angular started with Angular 1 being first released in 2009. But it is considered more recent since it was more popular in year 2012. In reference to Figure 2, Angular 1 had a huge interest over the recent years. However, the numbers on Figure 2. are not giving the right impression since these are not referring to the percentage of the usage in the market. Instead, the numbers are referring to the amount of times it was searched on Google’s search engine during the time period of 2009-2015. [25, 35-37.]

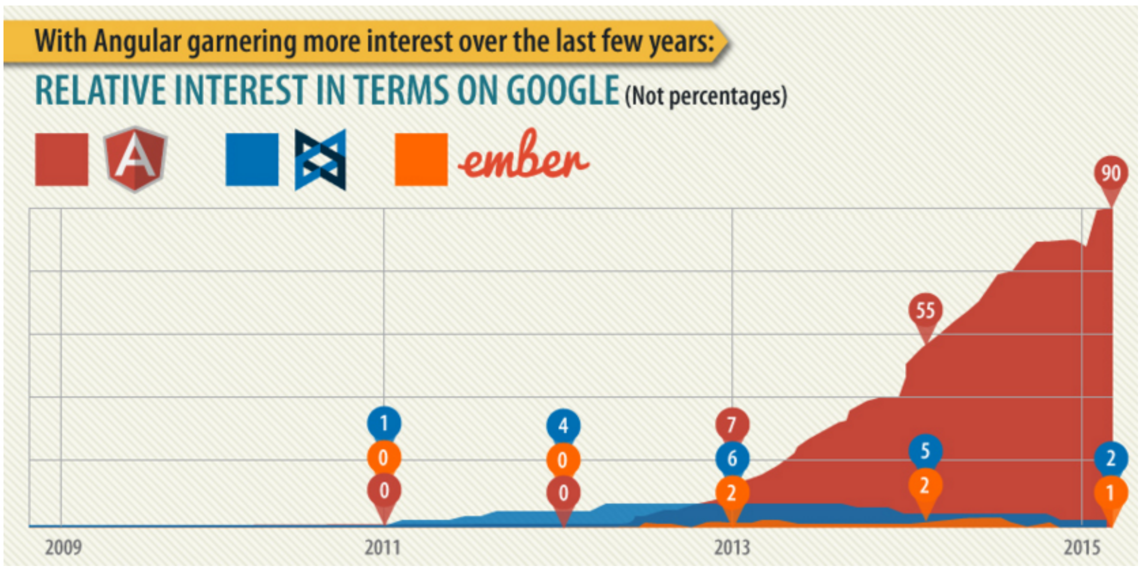

Figure 2. Comparison of search counts between Angular, Ember and React. Copied from Pronschinske [29.]

Angular 2 is still a work in-progress, as the time of conducting this study. The framework is expected to be released in the last quarter of 2016. The framework is already mature enough to build a stable web application as it is in Release Candidate phase and no major changes are expected. It is expected to have a similar popularity as Angular 1 had. Angular 2 has already a growing community around it. An active community is definitely pushing the framework forward with creating different plugins and libraries to be compatible with it. Angular 2 applications are commonly developed with a programming language called TypeScript but also it is possible to write Angular 2 applications in JavaScript and Dart.

This analysis will focus on the parts which are unique to Angular 2 in comparison to other web frameworks. Certain technologies and patterns that Angular 2 uses will be described in details. The main aim of this section is to give a practical and a technical overview of how Angular 2 works and what to expect as a developer while considering using it.

5.2.1 Data Architecture

Angular 2 does not enforce any particular pattern or a technology stack which makes the architecture of the framework flexible. Angular 2 makes it easier to adopt different architectures without having the performance suffering. This is due to the fact that every application has its own requirements and challenges and therefore one solution that works well in some applications may not work for others. [26, 152.] However, freedom comes with a cost. Making decisions with many options creates a paradox of choice problem to developers, especially the inexperienced ones. Therefore, different alternatives need to be considered before making a choice.

Some of the possible patterns and implementations that could be used together with Angular 2 include the following:

- Flux: Promotes a unidirectional data-flow. It has three major parts. Stores which hold the data together, Views that actually fetches what is rendered in the store, and Actions which change the data that is contained in the store.
- Observables: Each observable gives what is called a stream of data. These streams could be subscribed to and then call different methods to react to the changes announced.
- Redux: Is one implementation that has been inspired by Flux. It is very similar but contains one more major part, a reducer. A reducer is group of pure functions that belong to a specific action.

[26, 152-153.]

5.2.2 TypeScript

TypeScript is the recommended language to use while building an application with Angular 2. The Angular's core team at Google decided to integrate it with the framework and the codebase of Angular 2 is written in TypeScript. The language is primarily developed by Microsoft.

Typescript is considered a superset of JavaScript but a more similar version to programming language Java as it gives the possibility to define new types.

Figure 3 demonstrates how Typescript is a superset of both ES6 and ES5 versions of JavaScript.

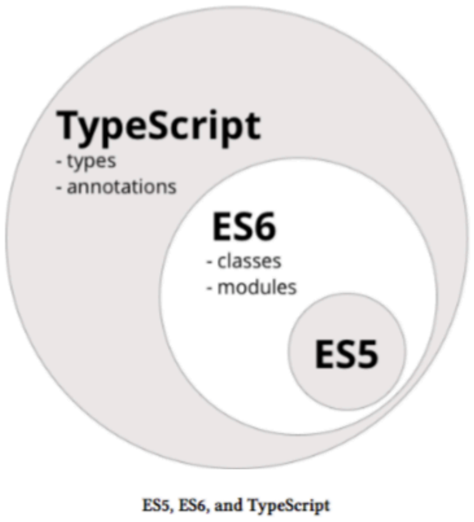

Figure 3. Typescript in relation to different JavaScript versions

The fact that types are available over the generic “var” gives more opportunities to have more support in terms of tools. Developers who are more familiar with loosely typed languages like JavaScript may not prefer the idea at the first glance. However, in practise a typed language increases the productivity of developers. TypeScript offers a static code analyser so while code is being inserted the editor is able to give guidance by suggesting functions parameters, available objects and methods. It also acts as a safety net. In events like inserting a wrong value in the code that does not match its expected type, the editor will be able to highlight the mistake and what is the correct type to insert.

In reference to types, TypeScript allows created custom types. In a scenario where the application is using a third-party library that is built in JavaScript. It is possible to create specific type declarations for that library. The TypeScript community is very active so the declarations for most JavaScript libraries are available. TypeScript is easily compiled back to basic JavaScript and therefore it is supported by all browsers.

5.2.3 Data Flow and Components

Component is a key concept in Angular 2. An application written with Angular 2 is basically one high-level component which holds its parts together in other components. In other words, an application is built by a tree of components. [26, 70.]

To demonstrate the concept in a more practical way, the following scenario could be considered. Building a dashboard that has a basic functionality. The dashboard will have two views, sales report and orders. The dashboard also has a navigation bar which is in a form of an independent component. Sales report have filters componentss and list-items components while orders also has a list-items component and Item-details component.

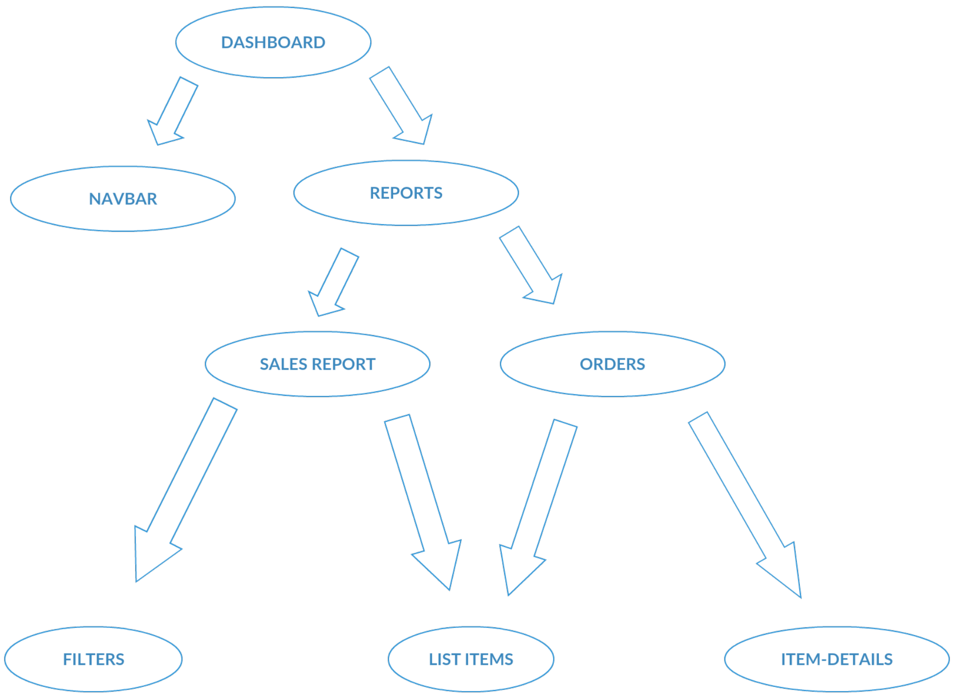

Figure 4. Components tree of a regular dashboard.

Figure 4 shows how an application could consist of different components and demonstrates how components can be used in multiple parts of the application such as “list-items” component. Each of these components contains the logic that belongs only to that particular part of the application.

Listing 5 is a very simple example of a component declaration.

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

@Component({
    selector: 'dashboard',
    templateUrl: 'name.component.html'
})
export class DashboardComponent implements OnInit {
    constructor() { } // Often used to inject dependencies

    ngOnInit() { } // one life cycle method that is trigged once the
component was initialized.
}
```

Listing 5. Dashboard component

The data flows from the top of the hierarchy to the bottom and vice-versa. There are two decorators available for this purpose. @Input inside a child component will allow its parent component to bind to a variable and pass data to it.

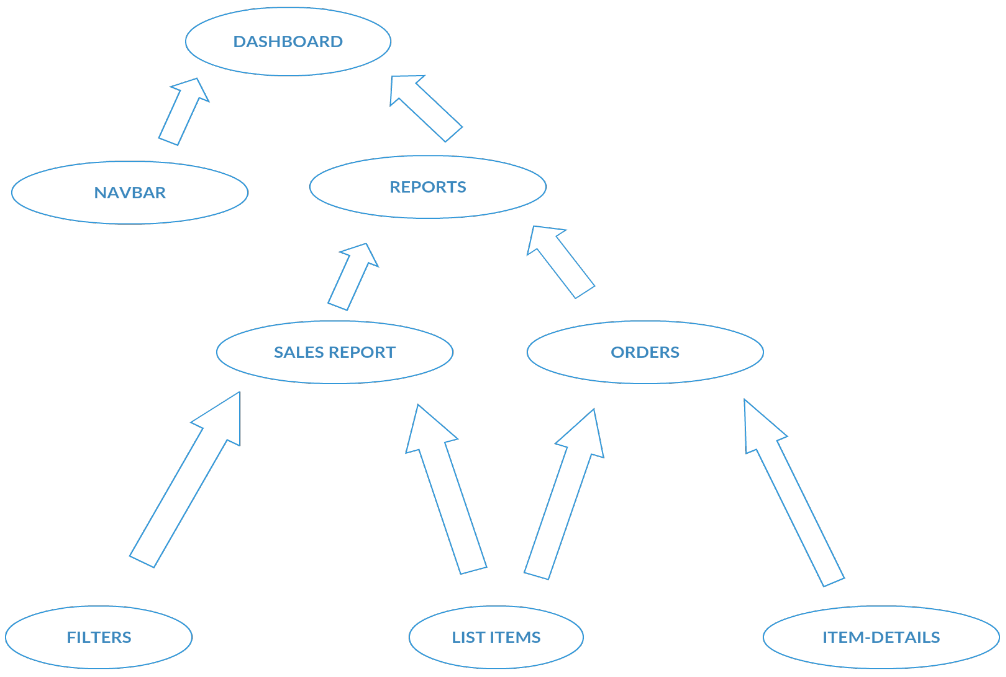

Figure 5. Components tree where the data flows from the bottom children to the top parent.

It is also possible for the children component to send data back to its parents as shown in Figure 5. @Output is another decorator that takes care of outputting the data to its parent component.

5.2.4 Observables and RxJS

This data architecture pattern is a practical implementation of the Observer design pattern. Observables heavily depend on reactive programming paradigm. A popular reactive streams library for JavaScript is RxJS that gives handy methods for calling different functions on streams of data. Observables are often used with RxJS as its relatively easy to work with it and Angular 2. [26, 155 – 156.]

The idea behind Observables has been explained before when the Observer design pattern was described. A code sample implementing a simple observable will be used to show how Observables are implemented in Angular 2.

```
import { Injectable } from '@angular/core';
import { Subject } from 'rxjs/Subject';

@Injectable()

export class EventsService {

    // Observable string sources
    private eventAnnouncedSource = new Subject<string>();
    private eventConfirmedSource = new Subject<string>();
    // Observable string streams
    eventAnnounced$ = this.eventAnnouncedSource.asObservable();
    eventConfirmed$ = this.eventConfirmedSource.asObservable();
    // Service message commands
    announceEvent(event: string) {
        this.eventAnnouncedSource.next(event);
    }
    confirmEvent(sponsor: string) {
        this.eventConfirmedSource.next(sponsor);
    }

}
```

Listing 6. Angular 2 service with two subjects that could be subscribed to.

Listing 6 demonstrates how a simple Observable may look like. Two Subjects were created, “eventAnouncedSource” and “eventConfirmedSourced”. Now these subjects will keep a list of different events and different sponsors as well. Now it becomes easier to subscribe to the Subjects and receive the data streams. The “EventService” can be used by other Angular components. For instance, “EventControlComponent”, in the code snippet found in Listing 7, is using it on a high level.

```
import { Component } from '@angular/core';
import { EventsService } from './';
@Component({
    selector: 'event-control',
    providers: [EventsService]
})
export class EventControlComponent {
    sponsers = ['HS', 'Redbull', 'Helsinki Kaupunki'];
    history: string[] = [];
    events = ['Karaoke night',
        'Midnight run',
        'Art Gallery'];
    nextevent = 0;
    constructor(private eventsService: EventsService) {
        eventsService.eventConfirmed$.subscribe(
            sponsor => {
                this.history.push(`${sponsor} confirmed the event`);
            });
    }
    announce() {
        let event = this.events[this.nextevent++];
        this.eventsService.announceEvent(event);
        this.history.push(`event "${event}" announced`);
        if (this.nextevent >= this.events.length) { this.nextevent =
0; }
    }
}
```

Listing 7. An Angular 2 component to handle events

Using Observables requires some knowledge of RxJS and reactive programming. As RxJS offers multiple methods to handle different situations while dealing with data streams. The library is considered challenging at the first glance to many developers as it works in a different manner in comparison to other libraries that handle data. A detailed analysis of RxJS is not within the scope of this thesis but few points will be highlighted to give an overview of what makes it different.

- "Pushing" data is different in Reactive programming. The usual case in imperative programming is that data is gathered by a method called "pull" but with using Observables "push" is used instead to allow the streams to send the data to its subscribers.
- RxJS is a pure functional paradigm. Methods such as map, filter, and reduce are all regular methods that are often used while using functional programming. Therefore, prior experience with that makes it much easier to use RxJS as all these streams that have been mentioned earlier are some form of lists and functional methods are applicable.
- Promises and streams are different. Promises are a technique to handle async server-calls which is preferred over callbacks. Streams emit many values while Promises emit only one value. Streams could be thought of as the new Promises that will be preferred over Promises as they increase readability and data maintenance.
- Streams are data pipelines. The streams work in a manner where it is possible to have access to whichever part of the stream and have the flexibility to combine different streams and create new ones. The stream concept gives a full control of what to do with the data and which data to select.
  [26, 155 ; 27.]

6 Implementation of Scalable Application Architecture

As a result of the findings discussed in chapter 4 and 5, development of a large web application started. The application had challenging requirements in terms of architecture, business-logic, and performance. The main idea behind the application was to have a business-to-business (B2B) channel allowing a multi-international company to sell their products to other businesses across Europe, North America, and Asia. The application would also be used to by their customers to track their orders. Typically, a company would use this application to get products shipped to them and the application's core value is to be able to place orders and track them accurately. Due to the wide range of the users, the application has to support 6 languages, multiple time zones, multiple unit of measurements and local laws of different countries. The application has to connect to a central ERP system to place orders in the system and fetch data for other functionalities. Functionalities of the applications included:

- Generate orders & transactions reports.
- Search and filter items from multiple records.
- Create new orders.
- Track order in real-time.
- Admin panel to manage and update items in the application.

The front-end was developed using latest technologies available at time. Angular 2 was chosen as the main framework to build the application. Publish/Subscribe & modular patterns were heavily used in the architecture of the application. The main technologies the stack used include the following:

- HTML
- Sass
- Git
- Angular 2
- TypeScript
- Rxjs
- Webpack 2
- Material Design
- Jasmine 2
- Karma

6.1 Angular 2 vs. React

When it comes to React and Angular 2, they are both excellent choices available in 2016. As described in Chapter 5, they promote good architecture patterns, use component-based development, have strong community around them, and offer best programming practises.

Choosing between the two options was not an easy task. The evaluation of the frameworks was not only based on the technical features. The specific case of the application and the development team's competences were taken into consideration as well. In the case of the application being developed in this thesis work, there have been multiple factors in favour of Angular 2 over React.

Out of the box Angular 2 has more functionalities and options in comparison to React. Angular 2 contain functionalities such as routing, server API calls, management of dependencies, code testing, and directional data flows. All these options are available without installing any additional libraries. On the other hand, React has the same functionalities available but they are not included out of the box. In React, additional libraries always have to be researched as there is a large number of options available for each common functionality. As a result, the development team has to make many decisions that it becomes a problem. Angular 2 offered more recommendations out of the box which helped kicking off the development. The application had a tight delivery schedule and it was expected to have more developers starting to work on the same application at a later stage. Therefore, using a consistent framework like Angular 2 was more convenient.

Tools supporting for Angular 2 are richer than the options available for React due to the technologies it is using. React's JSX has less support from code editors as it is not a standard markup language like HTML. React relies solely on JavaScript versions ES5 & ES6. JavaScript is a loosely typed language and therefore there is a limitation to the IDE's support. However, TypeScript the primary language of Angular 2 is a better version of JavaScript. Angular 2 does not require TypeScript but it is definitely recommended to use by the community and the Angular 2 core team. Using a strongly typed language like TypeScript gives a huge productivity enhancement in software development. It gives more control over the code and therefore less chances for errors.

Web components is a group of features that are being added to HTML and the DOM by W3C standard. Web components are intended to create reusable widgets in web applications. They expected to be widely used in the future. Angular 2's design has accounted for the new web component's standard. Angular 2's gives the possibility to convert the code in native web components. Therefore, it will be easier task to convert Angular 2 code into web components in comparison to React. Part of building scalable web applications is thinking ahead of time. The compatibility of the technologies used with new standards is an important issue.

In addition to the technical advantages of Angular 2, the development team had prior experience with Angular 1. Therefore, it was relatively easier to use Angular 2 over React. Since TypeScript is slightly similar to Java which is a strong typed object-oriented language. Some of the development team members had experience with Java which would give them a smaller learning curve while learning TypeScript.

It was more convenient to use Angular 2 over React. With reference to the points described earlier in chapters 4 and 5 and the technical merits that were offered by Angular 2 and the challenging nature of the application's requirements, they support choosing Angular 2. In addition, the time schedule planned for the application to be delivered and the background of the development team were other factors to choose Angular 2.

6.2 Practical Architecture and Structure

Creating a scalable and maintainable application starts with a good architecture that does not limit development and allows maximum code reusability. Angular 2 promotes a structure for the application's parts that follows the best programming practises and is logically linking the code together. An overview of Angular 2's architecture is highlighted in Figure 6. Angular 2 consists of modules which hold together services and components.

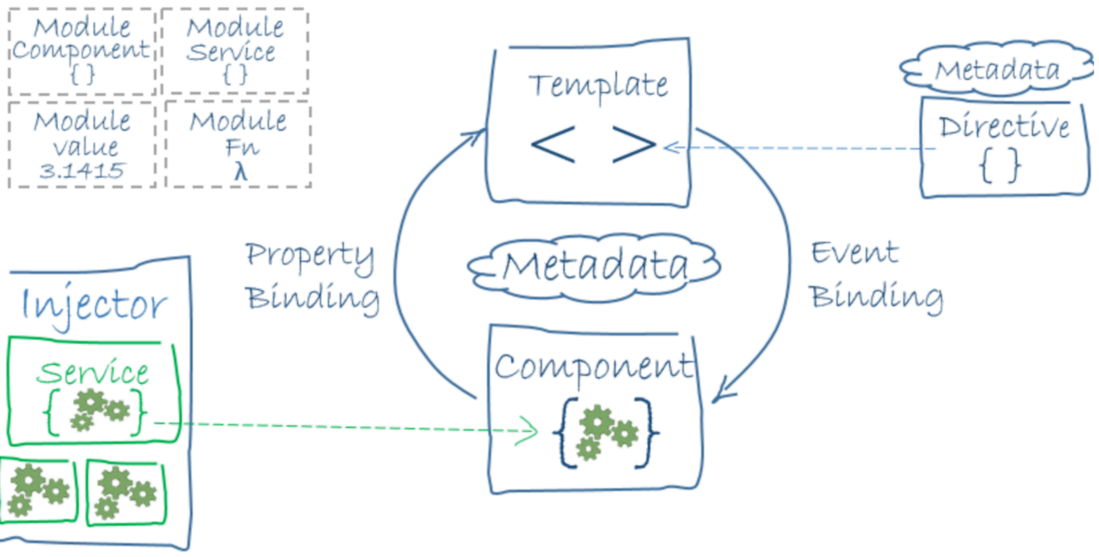

Figure 6. illustration of Angular 2’s architecture. [24.]

The application developed in the thesis had a structure that was built following Angular 2’s architecture and the modular design pattern. Thus, each group of functionalities were grouped together in a module. In reference to Listing 8, the application’s modules can be found under directory “src”. The whole application is bundled together under a root module called app module found in app folder. App folder contains different modules such as order, dashboard, and shared modules. Each of the application’s modules has a folder starting with a “+” sign to indicate that it’s a module. The sign is a naming convention recommended by Angular 2.

```
├── config
└─ src
    ├── app                  /**root app module**/
    |    ├── +order          /**order module **/
    |    ├── +dashboard      /**dashboard module **/
    |    └─ +shared  /**shared module **/
    |    ├── app.component.ts
    |    ├── app.config.ts
    |    ├── app.e2e.ts
    |    ├── app.module.ts
    |    ├── app.routes.ts
    |    ├── app.service.ts
    |    ├─── app.spec.ts
    |    ├─── app.style.scss
    |    ├─── app.template.html
 ├─── assets
    └── custom_typings
```

Listing 8. Overview of the application’s general structure.

Every module typically has its own routing configurations. Therefore, each module is responsible for controlling how to load its components. The routes are stored in files with extension “*.routes.ts”. The root app module also contains a routing file that is responsible for controlling the paths to every other module.

In addition, the app module contains the end to end, e2e, tests of the whole application. The e2e tests are performed only on a high level and only test the application’s modules. Typically end to end tests are to ensure that the modules of the application perform together as expected.

Each module consists of multiple components that together form the module. Listing 9 offers more detailed illustration of how a module may look like in terms of structure. The dashboard module includes order reports and stock reports component. Also it contains the necessary files to export the dashboard module such as “dashboard.component.ts” and “dashboard.module.ts“.

```
├─── orders-report
|   └── shared
|         └── order-report.service.ts
├─── stock-report
|   ├─── shared
├─── dashboard.component.ts
├─── dashboard.spec.ts
├─── dashboard.module.ts
├─── dashboard.routes.ts
├─── dashboard.style.scss
```

```
├──── dashboard.template.html
└── index.ts
```

Listing 9. Dashboard module's files

The file with name "dashboard.spec.ts" in Listing 9 is responsible for the unit tests of the dashboard component. Angular 2 recommends writing unit tests with extension " *.spec.ts ". With that naming convention it becomes easy to locate all the unit test files.

Listing 8 contained a folder called assets on the same level as the src folder. Assets is where some of the application's supporting files live. Files such as CSS libraries, font files, and images are all meant to be in the assets. Listing 10 gives an insight of the different files that may be placed in the assets folder. The CSS files should not be confused with component's styling as CSS files in assets folder are only for libraries.

```
└── assets
    ├──── css
    ├──── fonts
    ├──── icon
    ├──── img
    ├──── mock-data
    └── svg
```

Listing10. Illustration of application's assets folder

An important module found in the overview of the application was the shared module. This module is responsible for all the different components which are shared all over the application. For instances, input elements with validations and navigation bar components are used in other modules in the application. The shared module also may contain CSS styles which are shared in multiple other modules. The shared module's implementation allows code to be reused across the application. Listing 11 gives a simple structure of how the shared module could be built considering the scenario that the orders module requires having a navigation bar. The only action required is to import the shared module into the orders module and the code of the nav component is instantly available for the order module.

```
└── shared
      ├─── dropdown-element
      ├─── nav
      ├─── quantity-input-element
      ├─── radio-list
      └── styles
            ├─── modules
            └── partials
```

Listing 11. Shared module

7 Conclusion

Design decisions and use of available technologies are major factors that play a huge role in building a large web application that is functional and maintainable. Frameworks, design patterns, and architectural structures are extremely helpful to boost development time and make complex applications possible.

The final result of this project was practical recommendations and guidelines to design and develop a large-scale maintainable web application. Therefore, the project achieved its goals. All of the design patterns and web frameworks that were researched followed common principles and patterns. These tools are aiming to solve common problems faced by developers to allow them to focus on the real challenges of their case. This is possible only by the code re-usage, application of the ready-made solutions, and the loosely coupled codebase.

The findings are limited to the technology available at the time this project was taking place. However, the logic and the methods that were applied to find the frameworks and the design patters will be reusable in the future to evaluate technologies available at that time. There is no absolute solution to every problem. Every application has its own unique problems. Therefore, identifying these problems and researching solutions is the key element to create a good application.

References

1 Tieto home page. [Online] Accessed 15 September 2016 URL: http://www.tieto.com/

2 MODx Web Development. John, Antano; March 2009:8-13

3 7 reasons why frameworks are the new programming languages. Peter Wayner; Infoworld; March 30 2015. [Online] Accessed 4 May 2016 URL: http://www.infoworld.com/article/2902242/application-development/7-reasons-why-frameworks-are-the-new-programming-languages.html?page=2

4 On the Way to the Web: The Secret History of the Internet and Its Founders. Banks, Michael A.; January 2008: 49

5 Web Teaching. A Guide to Designing Interactive Teaching for the World Wide Web. Brooks, David W. Nolan, Diane E., Gallagher, Susan M.; February 2001; 3

6 Web 2.0 Website Programming with Django: Step Through the Development Of A Complete Social Bookmarking Application with the Python Web Framework That Encourages Clean And Rapid Development.Hourieh, Ayman; April 2008; 5-6

7 Design patterns: Elements of reusable Object-Oriented Software, Gamma , Raplh Johnson, Richard Helm, John Vissides; October 1994

8 Patterns For Large-Scale JavaScript Application Architecture , Addy Osmani [online]. Accessed 28 June 2016 URL: https://addyosmani.com/largescalejavascript/

9 Design Patterns For Dummies, Steve Holzner ; May 8, 2006;

10 Learning JavaScript Design Patterns [Online]. Addy Osmani; 2015. [Online] Accessed August 29 2016 URL: https://addyosmani.com/resources/essentialjsdesignpatterns/book/#detailmvcmvp

11 Software License Unveiled: How Legislation by License Controls Software Access. Phillips, Douglas E.; June 2009; 9-20

12 JavaScript Unit Testing. Salen, Hazem; 2013; 7-8

13 Software Engineering [ONLINE]. St. Andrews University; 2010: chapter 30 - documentation URL:

14 Speaking on the Observer Pattern - Tony Sintes ; May 2001 [online]. URL: http://www.javaworld.com/article/2077444/learn-java/speaking-on-the-observer-pattern.html

15 Pro Javascript Design Patterns , Harmes , Ross ; 2008

16 Show love to the module pattern , Christian Heilmann ; July 2007 [online]. Accessed August 29 2016 URL: http://www.christianheilmann.com/2007/07/24/show-love-to-the-module-pattern/

17 Again with the module pattern - reveal something to the world , Christian Heilmann ; August 2007 [online]. Accessed August 29 2016 URL: http://www.christianheilmann.com/2007/08/22/again-with-the-module-pattern-reveal-something-to-the-world/

18 Developing a React Edge. Frankie Bagnardi, Jonthan Beebe, Richard Feldman,Tom Hallett, Simon Højberg, Karl Mikkelsen ; November 2015

19 Introduction to React , Cory Gackenheimer ; 2015

20 React.js essentials : a fast-paced guide to designing and building scalable and maintainable web apps with React.js , Artemij Fedosejev ; 2016

21 React's JSX: The Other Side of the Coin, Cory House; Aug 13 2015 [Online]. Accessed September 5 2016 URL: https://medium.com/@housecor/react-s-jsx-the-other-side-of-the-coin-2ace7ab62b98#.7zyb84qo9

22 Thinking in React – Official Facebook's React documentation. Accessed September 5 2016 URL: https://facebook.github.io/react/docs/thinking-in-react.html

23 Component Specs and Life Cyclce – Official Facebook's React documentation. Accessed September 5 2016 URL: https://facebook.github.io/react/docs/component-specs.html

24 Architecture Overview [Online]. Google; 2016.[Online] Accessed September 5 2016 URL: https://angular.io/docs/ts/latest/guide/architecture.html

25 Beginning AngularJS. Andrew Grant; 2014; 35-34

26 Ng-book 2: The complete Book on AngularJS 2 , Ari Lerner, Felipe Coury , Nate Murray , Carlos Taborda; 2016

27 ReactiveX - Observable, RxJS official documentation. [Online] URL: http://reactivex.io/documentation/observable.html

28 Observer Design pattern using JavaScript , Salvatore Vetro ; 2006 [online]. Accessed September 5 2016 URL: http://www.codeproject.com/Articles/13914/Observer-Design-Pattern-Using-JavaScript

29 Infographic: AngularJS vs. Ember vs. Backbone.js by Mitch Pronschinske [Online] Accessed 6 Sept 2016. URL: https://dzone.com/articles/infographic-angularjs-vs-ember-vs-backbonejs