

Jaakko Voutilainen

Evaluation of Front-end JavaScript Frameworks for Master Data Management Application Development

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

8 December 2017

| | |
|---|--|
| Author Title | Jaakko Voutilainen Evaluation of Front-end JavaScript Frameworks for Master Data Management Application Development |
| Number of Pages Date | 50 pages 8 December 2017 |
| Degree | Bachelor of Engineering |
| Degree Programme | Information Technology |
| Specialization option | Embedded Systems |
| Instructors | Olli Hämäläinen, Senior Lecturer Jussi Järveläinen, Director of Research and Development |
| <p>The purpose of this thesis was to evaluate the most significant JavaScript frameworks in terms of a master data management (MDM) application development, and select the most feasible option for use in FCG Prodacapo Group.</p> <p>In the study, modern web application architecture and the differences between multi-page applications and single-page applications were exploited. The structure of JavaScript frameworks and libraries was introduced, and three of the most popular technologies were selected into the evaluation process: React.js, Angular and Vue.js.</p> <p>The selected frameworks were evaluated focusing on relevant factors in the development of an MDM application. These factors included the frameworks' structure, such as components, data binding and state management. Other aspects related to efficient development were considered as well, such as frameworks' popularity and prospects, documentation and developer experience. The final selection was based on aspects seen as the most important considering the whole ensemble, such as frameworks' fulfillment of the general requirements for an MDM application, frameworks' continuation of development and high effectiveness. Considering these aspects, the evaluated factors were placed in order of significance, and on the basis of these aspects, Angular was chosen as the best choice for MDM application development. For verifying the feasibility of the selected framework (Angular), a test application was implemented. The test application included two pages, and functionalities for routing, data table and for server connections. The test application filled its objectives and verified the feasibility of the Angular framework.</p> <p>All evaluated frameworks were found to be suitable for fulfilling the needs. The evaluated factors were not equally important from the evaluation point-of-view, but the evaluation was based on aspects seen as the most important. The evaluation process is suitable on a more global scope as well, as the minimum requirements were not strictly MDM application specific. According to the company, this thesis achieved its objectives comprehensively. It is highly probable that Angular will be used in MDM application development and as the company's main front-end development platform.</p> | |
| Keywords | SPA, MDM, JavaScript, framework, evaluation |

| | |
|---|---|
| Tekijä Otsikko | Jaakko Voutilainen Selainpuolen JavaScript-ohjelmistokehysten soveltuvuusarviointi avaintiedonhallintajärjestelmän kehitykseen |
| Sivumäärä Aika | 50 sivua 8.12.2017 |
| Tutkinto | Insinööri (AMK) |
| Koulutusohjelma | Tietotekniikka |
| Suuntautumisvaihtoehto | Sulautetut järjestelmät |
| Ohjaajat | Lehtori Olli Hämäläinen Tuotekehitysjohtaja Jussi Järveläinen |
| <p>Insinööriyön tarkoituksena oli arvioida merkittävimpiä JavaScript-ohjelmistokehityksiä ja valita niistä soveltuvin vaihtoehto käytettäväksi avaintiedon hallintaan (Master Data Management, MDM) keskittyvässä web-sovelluskehityksessä insinööriyön tilaajayrityksessä.</p> <p>Työssä perehdyttiin moderniin web-sovellusarkkitehtuuriin, sekä monisivuisten ja yksisivuisten web-sovellusten eroavaisuuksiin. Myös JavaScript-ohjelmistokehysten ja kirjastojen rakenteet käytiin läpi, ja kolme tämän hetken suosituinta teknologiaa valittiin arviointiprosessiin: React.js, Angular ja Vue.js.</p> <p>Valitut ohjelmistokehitykset arvioitiin keskittymällä MDM-järjestelmän kehitykseen liittyviin olennaisiin asioihin. Näihin sisältyivät ohjelmistokehityksen rakenne, kuten ohjelmistokehityksen komponentit ja vuorovaikutus, tiedon sidonta sekä tilanhallinta. Myös muita esteettömään sovelluskehitykseen liittyviä asioita otettiin huomioon, kuten ohjelmistokehityksen suosio ja tulevaisuudennäkymät, dokumentaatio ja kehittäjäkokemus. Lopullinen valinta perustettiin kokonaisuuden kannalta tärkeimpinä nähtyihin asiakokonaisuuksiin, jotka olivat yleisen MDM-sovelluksen vähimmäisvaatimusten täyttyminen, ohjelmistokehityksen tuen jatkuminen ja ohjelmistokehityksen tehokkuus. Näiden kokonaisuuksien valossa arvioidut asiat asetettiin merkittävyyssjärjestykseen, ja arvioinnin tuloksena Angular valittiin parhaaksi vaihtoehdoksi MDM-järjestelmän kehitykseen. Ohjelmistokehityksen soveltuvuuden ja MDM-järjestelmälle tyypillisten toimintojen toteutettavuuden varmistamiseksi toteutettiin testisovellus. Testisovelluksessa oli kaksi sivua, ja ominaisuuksina reititin, datataulukko ja palvelinyhteys. Testisovellus täytti tavoitteensa ja varmisti Angularin soveltuvuuden.</p> <p>Työn aikana huomattiin, että kaikki ohjelmistokehitykset täyttävät vähimmäisvaatimukset. Arvioidut aiheet eivät olleet samanarvoisia arvioinnin näkökulmasta, vaan valinta perustui tärkeimpinä pidettyihin asioihin. Työn arviointiprosessi soveltuu käytettäväksi myös yleisemmällä tasolla, sillä vähimmäisvaatimukset eivät olleet MDM-sovelluskohtaisia. Insinööriyön tilannut yritys pitää opinnäytetyötä kattavana ja tarkoitukseen sopivana. On todennäköistä, että Angularia tullaan käyttämään yrityksessä MDM-järjestelmän kehityksessä ja yrityksen pääasiallisena kehitysalustana selainpuolen ohjelmistokehityksessä.</p> | |
| Avainsanat | SPA, MDM, JavaScript, ohjelmistokehitys, soveltuvuusarviointi |

Contents

List of Abbreviations

| | | |
|-------|--|----|
| 1 | Introduction | 1 |
| 1.1 | Background | 1 |
| 1.2 | Problem statement | 2 |
| 1.3 | Objectives | 3 |
| 1.4 | Structure of the report | 3 |
| 2 | Web application architecture | 4 |
| 2.1 | Background | 4 |
| 2.2 | Model-View-Controller pattern | 5 |
| 2.3 | Single-page applications | 6 |
| 2.3.1 | Data transfer and server architecture | 6 |
| 2.3.2 | Routing | 7 |
| 2.3.3 | Single-page applications compared to multi-page applications | 9 |
| 3 | JavaScript frameworks | 9 |
| 3.1 | Background | 9 |
| 3.2 | Definition of a JavaScript framework | 10 |
| 3.3 | Significant options in JavaScript frameworks | 12 |
| 3.4 | Description of the selected frameworks for evaluation | 13 |
| 3.4.1 | React | 13 |
| 3.4.2 | Angular | 13 |
| 3.4.3 | Vue | 14 |
| 4 | Evaluation of the selected JavaScript frameworks | 15 |
| 4.1 | Components | 15 |
| 4.2 | Data binding and state management | 19 |
| 4.3 | Routing | 22 |
| 4.4 | Performance | 22 |
| 4.4.1 | Startup performance and build times | 23 |
| 4.4.2 | Scripting and rendering | 25 |
| 4.4.3 | Platform support | 27 |
| 4.5 | External libraries and user experience components | 27 |
| 4.6 | Localization | 28 |
| 4.7 | Documentation and community support | 29 |

| | | |
|------|---|----|
| 4.8 | Learning curve and developer experience | 31 |
| 4.9 | Popularity and future | 32 |
| 4.10 | Conclusion | 36 |
| 5 | Proof of concept | 39 |
| 5.1 | Introduction | 39 |
| 5.2 | Demonstration | 39 |
| 6 | Conclusion | 42 |
| | References | 44 |

List of abbreviations and key concepts

| | |
|------------|--|
| Angular | JavaScript framework maintained by Google. Successor of Angular.js. |
| DOM | Document Object Model. A standard tree kind of structure or model of an HTML document for browser to access elements. |
| Framework | A reusable software environment to provide a standard way to build applications. |
| JavaScript | A high-level programming language used widely in WWW-content production. |
| JSON | JavaScript Object Notation. Human-readable text based data format, used in JavaScript. |
| MDM | Master Data Management. Processes, strategies and management related to company's master data. |
| MVC | Model-View-Controller. Software architectural pattern, which divides an application to a model, a view and a controller. |
| Node.js | A run-time environment for server-side JavaScript. Node.js NPM is a package manager for Node.js modules. |
| React | JavaScript library maintained by Facebook. |
| SPA | Single-page application. In web architecture, web application which contains technically only one page, and pages are dynamically created and modified programmatically. |
| TypeScript | A superset of JavaScript which adds optional typing to JavaScript. |
| UI | User interface |
| Vue | Progressive JavaScript framework by Evan You, developed and maintained by international core team. |

1 Introduction

1.1 Background

Modern web application technologies have evolved very quickly over the past few years. The traditional web application model is a multi-page application, which has dominated the WWW-world from the beginning. One crucial disadvantage in traditional web applications (multi-page application) is bad responsiveness. When a user changes pages, it takes time for the browser to retrieve a new HTML document from the server. The server's internal processing can take time as well. Nowadays, user devices in the WWW-world continually possess more processing power and larger memory capacity. Due to these facts, a bigger share of application logic and processing is feasible to hand over to the end device, a desktop PC or a mobile phone for instance. This will free the server from using great amounts of resources for each client. A so-called single-page application (SPA) model applies better to this concept. As the data transfer rates have also improved lately, the SPA model offers a significant improvement in the user experience. In SPA, the whole application content is loaded at once and so the initial page load is usually longer, but the latter page changes occur instantaneously.

As single-page applications are getting popular and more processing is centered to the client's side, front-end programming languages (e.g. JavaScript) need to evolve as well. Application framework, defined as big reusable set of libraries for implementing application main structure, have become a trend also in JavaScript development. JavaScript frameworks aim to extend the developer's possibilities and make developing JavaScript easier, providing ready-made, optimized functions for more complex functionalities. In the company point-of-view, it is feasible to use some JavaScript framework on top of native JavaScript, as it makes the front-end development process faster in the longer term.

FCG Prodacapo Group (Prodacapo) is a Nordic software company offering advanced solutions for regional cost accounting, productization and analytics. Prodacapo is a part of FCG Finnish Consulting Group. Prodacapo has a need for deciding which JavaScript framework would be most applicable in their master data management related application development, and more generally in other upcoming projects as well. Master data is

company's central data, which is long-living and only slowly changing. Common examples of master data are e.g. company's product information, customer data, organizational information and code sets. Master data management means processes, strategies and management related to the company's master data.

1.2 Problem statement

Master data management (MDM) application is an application which handles the company's master data, in some form or another. There is no uniform definition for MDM application, and thus any strict requirements do not exist. Neither Prodacapo has provided specifications. For most MDM applications, there are still some similar characteristics: they handle large sets of data and possibly include heavy data tables, there are more than one functional pages, and they communicate with the server. Hence, when no more specific requirements have been set, these are considered as minimum requirements.

Due to continually evolving and changing JavaScript frameworks, it is not a foregone conclusion which framework is the most feasible option for developing MDM application, or to be used as a company's main development platform. The most feasible option depends on several things, including the framework's popularity and prospects, performance and development team's previous experience.

This thesis aims at evaluating the current and the most popular JavaScript frameworks and select the most feasible option for developing a master data management related application. The selected framework will be possibly used as Prodacapo's main development platform in upcoming projects as well. The evaluation will be made focusing on relevant matters in a master data management application development.

Google's Angular is preselected as one of the frameworks to be evaluated, as requested by Prodacapo. The Finnish development team in Prodacapo has some previous familiarity in the Angular framework. Previous experience is not, however, an evaluable aspect in this study, as the evaluation will be affecting other departments as well.

As the requirements can differ among the applications, the selected framework in this thesis is not necessarily the best case in every scenario. A secondary outcome is to

introduce significant areas for consideration when selecting the JavaScript framework, for the reader to be able to make their own conclusions based on them.

1.3 Objectives

The objectives of this thesis are:

- To introduce a current web application architecture and present the differences between the multi-page application model and the single-page application model.
- To introduce JavaScript frameworks' structure and functionality, and present the most significant options in their field.
- To select a collection of JavaScript frameworks, considered as the best in terms of popularity and suitability for more accurate evaluation. Angular is preselected in the collection, due to request of Prodacapo.
- To evaluate the feasibility of each selected framework based on their performance, stability, and prospects, for building a master data management related single-page application.
- To select and justify the most feasible option for the purpose, primarily to be used in a master data management application development in Prodacapo, and secondarily as Prodacapo's main development platform in front-end software development.
- Verify the feasibility of the selected framework in practice.

1.4 Structure of the report

This thesis is structured into 6 chapters. Chapter 2 presents the current web application architecture and compares differences in multi-page applications and single-page applications. Chapter 3 describes in detail the structure and component model of JavaScript frameworks and contains the selection of the evaluable frameworks. Chapter 4 contains the evaluation of the frameworks, with each evaluable aspect divided into sections. The length of the sections varies depending on the aspect's relevance. Chapter 5 is verifying the outcome in practice (proof of concept), in the form of a test application. Chapter 6 concludes this thesis, and aims to aggregate and consolidate opinions which have emerged during the study.

2 Web application architecture

2.1 Background

A traditional web application architecture consists of a client and a web server. The client can be, for instance, a browser in a desktop PC or a mobile device. The client sends HTTP requests to the web server, which then performs some functionality based on the request and possibly returns some result. In the picture below (Figure 1) is the pattern of a traditional web application data transfer model.

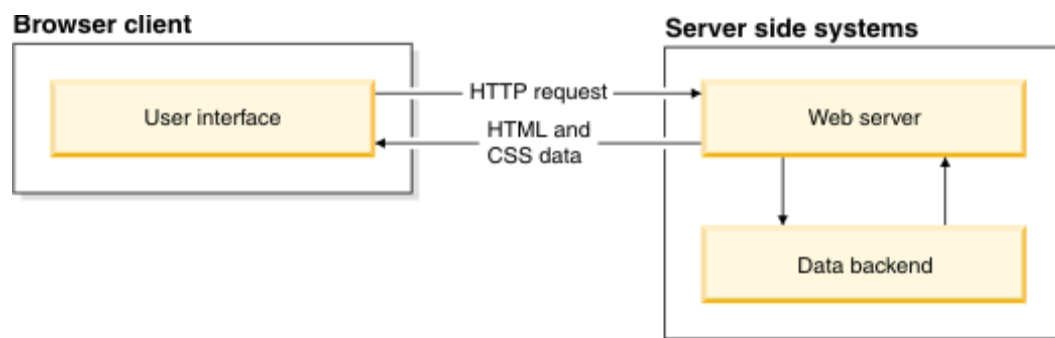


Figure 1. Web application data transfer model [1].

In Figure 1, the browser client asks for a web page from the web server, and the web server in turn asks some data from the database, and finally returns an HTML document along with CSS styles to the user's browser.

A multi-page application usually consists of multiple pages. When the user clicks a link on the page, it redirects the browser to a different HTML file existing on the server. The needed stylesheet files are retrieved as well, if they are not yet saved in the browser's cache. The server traditionally has other tasks as well, in addition to returning HTML documents. The server can authenticate the user, make database changes, or for instance, send emails.

In a web application architecture, routing is a process in which each different URL corresponds to a specific page or content. In a multi-page application model, routing is more of a self-evident part of an application as the pages are separated also in the file system. Using URLs and routes, browsers can bookmark, share a link, and navigate on websites using back and forward buttons.

Some processing can occur also at the client's end. Processing in the browser usually utilizes JavaScript and contains tasks like modifying the DOM, creating interactive visualizations, or displaying alerts on the screen. HTML DOM (Document Object Model) is a standard tree kind of structure, or model, which the browser uses for accessing different elements. There are many kinds of setups, but the very background of the tasks divided is usually the same in traditional multi-page web applications.

2.2 Model-View-Controller pattern

Model-View-Controller (MVC) is an architectural pattern in which the data model, the visible view, and the functional controller are separated. The data model presents all the data content of the application, the view part indicates visible representation of information, and the controller handles the processing of the data and shifts it between the model and the view. Figure 2 illustrates Regis Frey's vision about the MVC pattern [2].

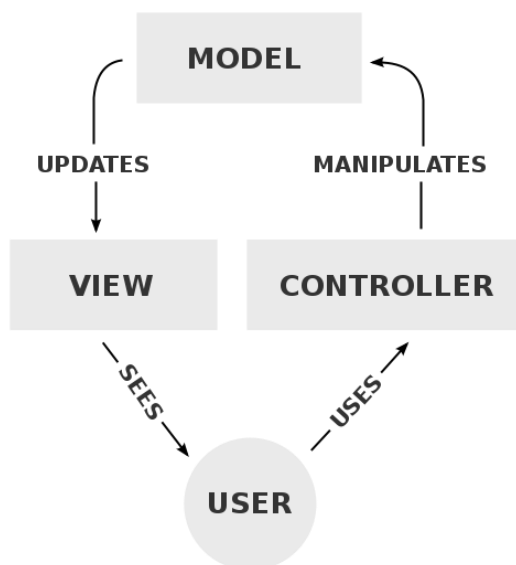


Figure 2. Model-View-Controller model, by Regis Frey [2].

In Figure 2, due to user's actions, the controller is manipulating the model, and thus the view is updated as well. The MVC model defines a standard way to split application in parts, and so understanding the functionality of the application becomes easier regardless of the technology used. The MVC model is used widely in multi-page applications.

2.3 Single-page applications

A single-page application (SPA) operates quite differently in terms of data transfer or routing, compared to a multi-page application. Single-page applications conceptually consist only of one single page. When a client asks for a web page, it receives a full website's code at once. The received content often contains only a partially complete HTML document. It is complemented dynamically at the client's end, usually with JavaScript. In other words, the visible view is created by a series of JavaScript functions, which complements the HTML DOM on run-time. No further requests are needed, but all the visible content is created and modified with JavaScript. Thus, for instance, the page changing in an SPA is not really page changing, but rather replacing the content of the current page with a new content. This content is generated by JavaScript functions.

2.3.1 Data transfer and server architecture

In a SPA, the whole application, including all of its pages, are downloaded in a single, initial page load. First response from the server includes a full website; HTML, JavaScript and stylesheets. Due to this fact, the amount of data in the first request is naturally big, but download times are still tolerable due to the current high speeds in the broadband technology.

Some data are required to be retrieved after the initial page load, however. Some portion of the page may need to be updated, like in social-media feeds or advertisements, because of a user action for example. In single-page applications, this kind of data is retrieved with Ajax requests. Ajax requests are asynchronous methods which allow exchanging the data with the server without reloading the page. In below, Figure 3's upper line depicts the first request, and the lower line presents the latter requests. Generally, the data is transferred as an JSON object. JSON (JavaScript Object Notation) is a common, lightweight, human readable JavaScript object format.

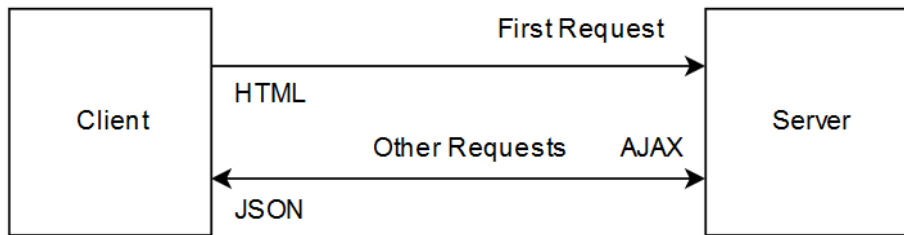


Figure 3. Data transfer in single-page applications [3].

In Figure 3, the first request is for an HTML document, scripts and stylesheets. The lower line represents the latter, asynchronous data exchange in JSON object format in a single-page application model.

Server-side logic is usually much simpler in the single-page application model, and has a smaller amount of processing and logic. The server is often only a pure stateless web service, or state-full in some cases. *RESTful* web services are quite common among single-page applications. RESTful (Representational state transfer) web services is a concept for uniform, stateless operations for accessing the resources. Different HTTP methods are connected to specific resources, and the application asks for specific data individually.

Node.js is a runtime for running server-side JavaScript. Many front-end technologies utilize *Node.js* for instance in requests to the database. Hence, *Node.js* settles in between the web service and the front-end application. *Node.js* also provides NPM, a package manager. Packages in *Node.js* are called *modules*. *Node.js* modules are an easy way to handle libraries in a single-page application project.

2.3.2 Routing

In the case of single-page applications, the whole application has initially only one URL. From the file system point-of-view, this is because the whole application's initial HTML document is included in one file. Routing functionalities, such as bookmarking or navigation, are needed to be implemented programmatically if necessary. Two common types of routing are used: static routing and dynamic routing.

Static routing is most commonly used in single-page applications. In static routing, routes are defined as part of the application's initialization. Before any rendering occurs, routes

are already known to the application. In Figure 4 below is a data flow diagram in static routing model.

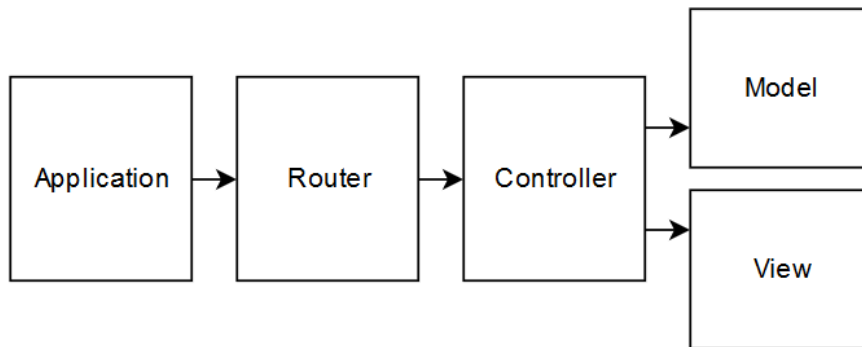


Figure 4. Static routing data flow diagram

In Figure 4, the router is placed in between the application and the controllers. This means that the routes are known to the controller in the initialization phase and cannot be changed afterwards. There are a couple of known disadvantages in using static routing. As routes are static, it is not possible to change routing on run-time.

Dynamic routing is another option in SPA's routing. In dynamic routing, routing can be altered based on prevailing environment on the client's end. Figure 5 below illustrates dynamic routing model, in which the router is placed next to the controller.

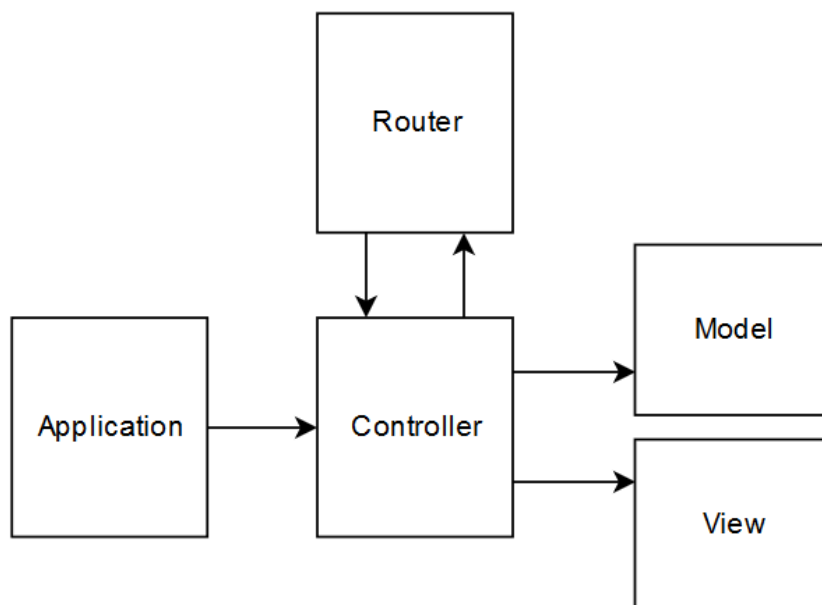


Figure 5. Dynamic routing data flow

Figure 5 illustrates how the controller is constantly working with the router, and so it can transfer information to it and receive a new route information. Thus, dynamic routing allows routes to be changed at any time, based on the client's prevailing environment, e.g. browser window's size.

2.3.3 Single-page applications compared to multi-page applications

In the single-page application model, loading the whole content at once and creating the page dynamically offers huge improvement in the overall responsiveness of the application. The browser does not have to download more content from the server, when the user changes pages. There is no need for reloading the pages, and so the SPA model also offers offline possibilities. This makes the look and feel experience in single-page applications more like in a desktop application. The overall performance is also better in single-page applications, as the processing resources are divided more efficiently. Client devices usually have a large applicable processing power, which is taken into use in the SPA model.

One factor slowing down a development of single-page application model is search-engine optimization. Search-engines have not been able to index the site, leading to poor visibility of the website. In October 2015, Google brought some improvement into this, by starting to index dynamic pages [4]. The single-page application model also has some issues with security, in terms of Cross Site Scripting (XSS) attacks. Malicious JavaScript code is relatively easy to inject in the SPA website [5]. This can be prevented however, but needs special attention from the developer. One more thing to consider as disadvantage of the single-page applications is that they are also fully dependent of JavaScript support of the browser.

3 JavaScript frameworks

3.1 Background

At the same time as the single-page application concept is growing its popularity, front-side programming languages are also evolving. JavaScript, as well as its libraries are

currently being developed faster than ever, and new technologies are released frequently. Libraries and frameworks tend to ease the JavaScript development, and extend its possibilities.

JavaScript libraries have existed for a long time already. They became popular for the first time with jQuery in 2006 [6]. More robust JavaScript frameworks started to become known to the public somewhere between 2010 and 2011, with AngularJS and Ember. AngularJS was the first real framework which merged data binding, routing and templating in one package [7]. Ember came soon after and offered a few improvements on top of AngularJS, e.g. more optimized routing.

In recent years, a full, comprehensive and more and more optimized JavaScript frameworks have appeared continuously. Popularity of JavaScript frameworks has been raising quickly, hand in hand with the single-page application trend. Currently, there is no framework considered as the best, but there is a handful of good options with slightly different functionalities. Thus, the future direction of JavaScript frameworks is still quite uncertain.

3.2 Definition of a JavaScript framework

A JavaScript framework is a big set of functions and facilitating tools, with its own control flow. As an abstraction, a framework provides a comprehensive development platform, one standard way for building applications. It can provide a dependency management, a file system structure and routing capabilities, for instance.

Many of the biggest frameworks consist of *components*. Components are the main building blocks of the framework. The way components are initialized and processed varies between the frameworks, but the main characteristics are the same: components can take information as parameters, perform actions, and possibly return some result. Components communicate with each other, can use each other's properties and can have parents or child components.

Below, Figure 6 presents various interaction directions between the components.

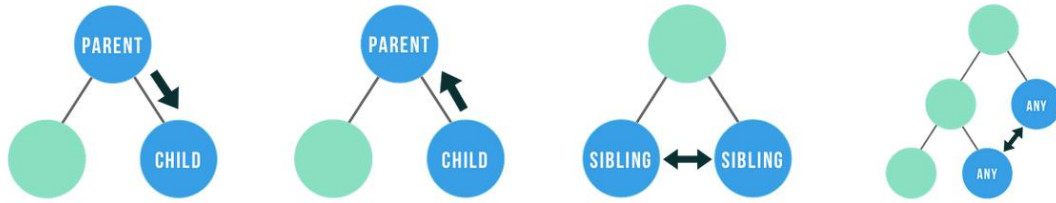


Figure 6. Relations between components [8].

In the above figure, four different relations of the components are visualized: parent to child, child to parent, siblings to one another and from any to any.

Components can be invisible to the user by doing only back-side processing. Or in turn, they can work with the user interface (UI) by producing an HTML document or part of it. Some frameworks utilize templates. Templates are initial HTML document files with an extended syntax of the framework itself. Components can take template as an input, make some changes and additions to it, and later output the completed HTML document. This concept is called component-based architecture, which is a comparable pattern to the MVC model existing in multi-page applications.

Components can exchange data with the server with Ajax requests. As described in section 2.3.1, Ajax provides asynchronous requests for data transfer, so that the application can dynamically update the content of the page without need for reload. Components can for instance, get more rows into a data table because of a user action.

In practice, a JavaScript framework can be only a single JavaScript file included in the web application. Then, the developer can use all the functions of the framework. However, there can be so many functions and tools replacing the native JavaScript versions, that the code does not look or feel like JavaScript code, but entirely new programming language.

The term “framework” is used, when the execution flow of the program is shifted from the responsibility area of the developer, to the responsibility area of the framework. This separates JavaScript frameworks from libraries; a library offers only a set of functions, while a framework manages processing stages and the data flow in whole application.

3.3 Significant options in JavaScript frameworks

Determining the best JavaScript frameworks is challenging, as the options considered as the best are changing continuously. In year 2017, the most popular JavaScript frameworks include at least: AngularJS, Angular, React.js, Vue.js, Ember.js, Meteor.js, Aurelia.js, Polymer.js, Backbone.js, Knockout.js and Mercury.js [9; 10]. These are common JavaScript frameworks or libraries, which are being actively developed and have a large user base.

Due to the request of the FCG Prodacapo Group (Prodacapo), Angular was preselected as one of the evaluated frameworks. AngularJS (predecessor of Angular) is excluded, as it may be outdated by its core [11; 12] and might be starting to fade out in popularity [9]. Nor would it be reasonable to select a preceding version of the framework even though it still has quite high usage statistics.

According to the Hackernoon's article [13], React.js (React) and Vue.js (Vue) are on the top list of 5 best JavaScript frameworks in 2017. React is a popular JavaScript library, developed and used by Facebook. Vue is a rather new framework, but very potential for getting high popularity [14]. In HotFrameworks comparison table [9], React is ranked as the most popular JavaScript framework after AngularJS, followed by Vue sharing the second place with Meteor.js. Sitepoint's article "Best JavaScript Frameworks, Libraries and Tools to use in 2017" by Craig Buckler [15], ranks React and Vue as one of the best frameworks currently existing.

Also, GitHub star ratings of React and Vue are very high. GitHub starring is a popularity measure of repositories; users can give a star to a repository for showing appreciation to the maintainer of the repository [16]. React has 80.963 stars and Vue has 73.543 stars in their GitHub repositories, which is showing a great difference in popularity when compared to Meteor.js with star count of 38.642 or Backbone.js, with 26.829 stars.

Based on the reference data presented, React and Vue are very likely to be the most popular frameworks and thus are selected to the more accurate evaluation process along with Angular.

3.4 Description of the selected frameworks for evaluation

This chapter includes a description of each framework selected for the evaluation process. The selected frameworks are React, Angular and Vue.

3.4.1 React

React is a JavaScript library, for building declarative views. As mentioned in section 3.2, term “library” in this context means that it is not meant to cover all areas of the application, but to provide a platform. A library is a collection of functionalities that the developer uses.

React is designed to create interactive user interfaces. It is declarative, which means that it designs different view for each state, and keeps views updated and rendered [17]. Among the three frameworks to be evaluated, React is probably the most uncontrolled option, as it is only a library and so the needed functionalities are left in the developer’s discretion.

React does not use HTML styled templates, and thus it does not have a separate HTML template file. All HTML document code is defined all the way in JavaScript. React also introduces use of virtual DOM. Virtual DOM is intermediate place before real DOM, where data changes are made first because of faster processing. Virtual DOM is discussed in more detail in the next chapter.

React is maintained by Facebook. It was released first time on March 2013, by Jordan Walke. Facebook uses React heavily on its websites, Facebook itself, and on Instagram and WhatsApp [18]. In this thesis, React version 15.6.1 is used and evaluated.

3.4.2 Angular

Angular is a JavaScript framework, development platform for mobile and desktop applications. It is developed by Google and by a community of individuals. It is the successor of the popular AngularJS framework, and is developed by same team. It was released initially September 2016.

Angular consists of a couple of core packages, definitions to core features. Angular uses templates for representing the view. Angular templates use extended HTML styled syntax, e.g. component's properties can be accessed in the template. [19.]

Angular is built entirely in TypeScript, and using it in the development is recommended, although not obligatory. TypeScript is a superset of JavaScript, which adds optional static typing to JavaScript [21].

Angular has had some naming confusion during its development. It was first labelled as Angular 2, but as it is rather a separate framework than a new version of the AngularJS, the development team announced later that it should be referred as Angular, without the "JS" [20]. Version number 3 was skipped, as it would cause confusion with Angular Router v3.3.0. The newest version of Angular is referenced as "Angular 4", or only "Angular". Hence, Angular 2, Angular 4 and Angular all mean the same framework but different versions, while AngularJS is completely a different framework. In this thesis, Angular version 4.0.0 is referred and evaluated.

3.4.3 Vue

Vue is the newest framework alongside two others. According to its developers, it is a progressive JavaScript framework, meaning its easily approachable, versatile, and very performant. It is easy to adopt, and its learning curve is said to be less steep comparing to React or Angular.

Vue takes into use many good features from React and Angular. Vue utilizes virtual DOM like React, and templating like Angular. Many of these features have been implemented even in a more sophisticated way in Vue.

Vue is an open-source framework, like React and Angular. Initially it was released in February 2014, by Evan You. Nowadays it has 18 official developers in its team and a growing community [22]. Version 2.5.3 of Vue is used in this evaluation.

4 Evaluation of the selected JavaScript frameworks

A single-page application is expected to cover at least all the common functionalities which are in multi-page applications. There are thousands of different functions and many different use cases in each of them. As a responsibility of these functionalities moves to the client side of the application, the application becomes heavily dependent on the client environment. The performance of the client device affects the functionality, as well as the operating system and the browser used. How different environments are supported, is largely a responsibility area of the JavaScript framework used.

Considering the large scale of responsibilities of the JavaScript framework, there are many things for it to take care of. An extensive and exact evaluation is challenging. In this evaluation, the frameworks are evaluated based on their feasibility to be used as a development platform in a master data management related application. The most relevant issues for an MDM application development are covered in detail, such as core features of the framework, feasibility of the process, and continuity of the framework's development. Things like search engine optimization for instance, are irrelevant matters for an MDM application, as the assumed use case is internal company use. Attention is paid also on the need for add-ons or external libraries on top of the library/framework.

A concrete objective is to compare the three selected frameworks, and choose and justify the best framework option for developing a single-page application for general master data management.

4.1 Components

In the development process of the application, the developer is most of the time in interaction with the framework components. Therefore, the evaluation of component syntax, functionality and ease of use in relation to the objective (MDM application) is justified.

In React, there are two possibilities for defining components. One possible style is very similar to writing a function in native JavaScript. In fact, React components can be defined with exactly the same syntax (Listing 1).

```
function AddNumbers(props) {
    return <h1>Addition result is {props.number1 +
props.number2}</h1>;
}
```

Listing 1. AddNumbers component in React (native JavaScript)

The function in the above Listing 1 is a valid syntax in React. The preferred way, however, is to use ECMAScript 6 style classes, as in the below Listing 2.

```
class AddNumbers extends React.Component {
    render() {
        return <h1>Addition result is
{this.props.number1 + this.props.number2}</h1>;
    }
}
```

Listing 2. AddNumbers component as an ES6 class in React

Class example seen in Listing 2 is using ECMAScript 6 (ES6). ES6 is a new JavaScript implementation, which introduces classes and modules [19].

React allows direct rendering of components. This can be achieved by giving an object, including component's name and parameters, to the render function of ReactDOM [23].

Listing 3 in below illustrates the usage of the render function of ReactDOM.

```
function AddNumbers(props) {
    return <h1>Addition result is {props.number1 + props.number2}</h1>;
}

ReactDOM.render(
    <AddNumbers number1={2} number2={3} />,
    document.getElementById('root'));
```

Listing 3. JSX code for rendering a component in React

The previous example, Listing 3, includes JSX syntax for creating an element. JSX is a statically typed programming language, providing faster and easier syntax for developing

JavaScript [24]. React uses JSX for templating, as it provides compiling optimization, it is type-safe, and basically syntactic sugar for creating the elements [25]. However, it is not necessary to use JSX syntax in React application. In React, components take input as properties, called *props*. Using properties can also be seen in above Listing 3.

React's component has a specific lifecycle. The lifecycle has methods, which will run before or after something happens. These include *render*, which is called when component is about to render (See Listing 4 from last page), *componentWillReceiveProps*, which is invoked before component receives new props, and *componentDidUpdate*, which runs after any updating occurs.

Angular's components work a bit differently as React's. Components are defined as a class with a *@Component* decorator, and a metadata of the component are given at once. The metadata of the component contain information how Angular initializes and processes the component [26]. For example, HTML code can be given as metadata to the component as a template.

The example below (Listing 4) illustrates how the Angular component can be created with a pre-defined HTML template. The template is given with other metadata (marked with *@Component* decorator).

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

@Component({
  selector: 'app-root',
  template: '<h1>Addition result is {{ number1+number2 }}</h1>',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app';
  number1=2;
  number2=3;
}
```

Listing 4. Creating a component with a template in Angular

First, in Listing 4, the `Component` library is imported from *Angular core* package. Then a component is being defined in `@Component` decorator. Templates can be given as a plain HTML as in Listing 4, or in a separate file. A metadata given to component in Listing 4 includes also selector, which gives an unique name for the component. Other possible information which can be passed includes e.g. change detection strategy, animations and styles.

Components in Angular have a certain lifecycle like in React. The lifecycle of an Angular component consists of: *OnChanges*, *OnInit*, *DoCheck*, *AfterContentInit*, *AfterContentChecked*, *AfterViewInit*, *AfterViewChecked* and *OnDestroy*. As in React, each of these lifecycle “hooks” have specific moment when it executes, e.g. *OnChanges* occurs when Angular detects changes in data-bound input fields, *AfterViewInit* occurs when component’s views are initialized, and *OnDestroy* is called when a component is destroyed (which can occur, for instance, on router page change). Lifecycle hooks are defined as part of the class definition. [27.]

Angular supports also dynamic component loader. It allows dynamic modifying of components on run-time via directives. With it application can load new components, and so forth the visible view, anytime during execution. This can be essential for instance in frequently changing advertisements, or some similar behavior needed in an MDM application. [28.]

Vue’s components are very close to the Angular’s implementation. Vue’s syntax is straightforward and easy to understand. As in React and Angular, components in Vue have their own isolated scope, which can have properties. In Listing 5, Vue component is registered with a message property, and used in the template.


```
import Vue from 'vue'

new Vue({
  el: '#app',
  template: '<h1>Addition result is {{ number1+number2 }}</h1>',
  data: {
    number1: 2,
    number2: 3
  }
})
```

Listing 5. Register of a Vue component [29].

As can be noted in Listing 5, Vue component's registration is rather straightforward, but has a quite different syntax comparing to React or Angular.

Different stages of processing can be accessed via lifecycle hooks also in Vue. As a difference to React and Angular, hooks are attached to Vue instance itself, and not to components. Lifecycle hooks in Vue are: *beforeCreated*, *created*, *beforeMount*, *mounted*, *beforeUpdate*, *updated*, *beforeDestroy* and *destroyed*.

As a conclusion, React's style of components are most closely to native JavaScript, and therefore it is presumably the easiest syntax to adapt into for previous JavaScript developer. In the other hand, Angular's and Vue's components offers a bit more functionalities. There are no great obstacles in any of the framework's component implementation considering MDM application, however. As the most efficient style for using components depends largely on the developer's previous knowledge and habits, there is no reason to set any framework above others based on the component implementation.

4.2 Data binding and state management

There may be a major difference in terms of performance, how the communication between a component and a view can be implemented, and how it *should* be implemented. This also applies to component's data binding. This section creates an overview for data binding options and a state management in React, Angular and Vue, and compares differences on each case.

In React, the data is meant to be flowing downwards. This is called as one-way data flow. Any parent component's data can be forwarded to a child component, but no other way around. Hence, the input fields of the page have no direct access to the component's state. In other words; HTML cannot change the component. This model has advantages in terms of simplicity, but usually, it is required for application to be able to change the state of its parent component on some point. This is where inverse data flow is needed. Inverse data flow means moving the data e.g. from an HTML input field to a component. In React, it can be achieved with callback functions; attaching an event handler to DOM events, and getting the value of the HTML input field from the event object. [30; 31.]

In below Listing 6, the input field value is first set from the component's state, and then a method is needed to move the data to another direction.

```
render() {
  return <input value={this.state.value} onChange={this.handleChange} />
}
handleChange(e) {
  this.setState({value: e.target.value});
}
```

Listing 6. Input field and onChange event in React [31].

In the above Listing 6, moving the data to opposite direction is implemented so that *onChange* event handler is set to *handleChange* function. *handleChange* function takes an event object as a parameter. When value of the input field is changed, the component's state is changing according to the event's target value.

As mentioned, React provides only one-way data binding. In addition to this, Angular and Vue both provide also two-way data binding option. Two-way data binding makes it possible to change the component's state as a consequence of changing the view, e.g. when the input field's value changes, the component's addressed property changes too. In this case, event handlers are not needed. Basically, event handlers do exist, but they are managed by the framework, and hidden from the developer.

Listing 7 below shows Angular's two-way data binding syntax.

```
<input [(ngModel)]="name" >
```

Listing 7. Angular's two-way data binding

As can be seen, two-way data binding can be implemented rather easily in Angular. Listing 8 presents Vue's version of the binding.

```
<input v-model="name" >
```

Listing 8. Vue's two-way data binding

Clearly, Angular's and Vue's two-way data binding makes the code prettier and simpler than in the React's solution. But whatever it wins in simplicity, it loses in manageability and performance. Two-way data binding requires for the framework to wire up *watchers* for each element. These watchers are for checking whether the input element's value has been changed. A big number of watchers consume resources from the browser, and in some cases, it might be difficult to keep up with all the different models and views which use same data. For instance, an infinite loop problem may occur with two-way data binding. The problem is visualized below in Figure 7.

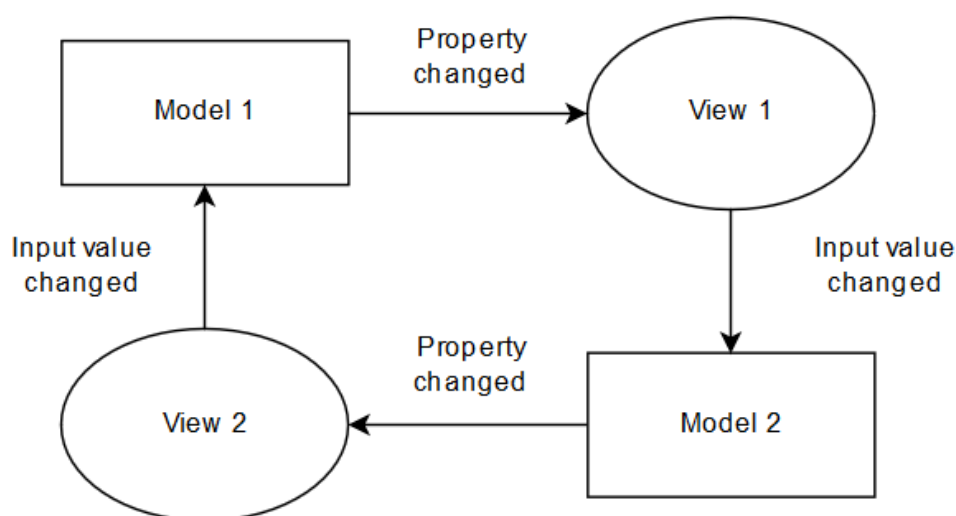


Figure 7. Infinite loop problem in two-way data binding

In Figure 7, the infinite loop problem takes place as follows: View 1 is using a property of Model 1, and when the property changes, the input value changes too. Model 2 uses data from View 1, and so it changes its state as well. View 2 is again using the Model 2's data, and it changes the input field's value accordingly, and Model 1's data changes along with it. Continuing the same way, the loop progresses endlessly.

Both one-way data binding and two-way data binding, have pros and cons. Due to the poor manageability and cons mentioned above in two-way data binding, it should be well justified if taken into use in large and complex applications. Unidirectional data is easier to handle and maintain in one-way data flow, as the event handlers are set by hand. When the state of the component is always known to the developer, data flow has a smaller chance of causing unexpected behavior or performance gaps [32; 33]. It is optional, however, whether to use two-way data binding in Angular and Vue.

4.3 Routing

Entry-level routing capabilities can be considered as a necessary feature in MDM applications, and therefore the framework's routing options should be examined. In this section, possible ways for implementing a router in each framework are presented and evaluated.

React does not include Router, or similar kind of functionality. For this purpose, React needs a router library, and the most popular option is possibly React Router. The newest version of React Router supports also dynamic routing. As mentioned in section 2.3.2, dynamic routing allows responsive routes, so that the content of the page can be adjusted to different screen sizes instantaneously, without a need for refreshing the page.

While React needs an external library, Angular provides a router library as a part of the base installation. Routes are configured in a local object, with path string and specific component where the route redirects [34]. The object is then given to the Angular Router. Angular router does not provide dynamic routing as is, but external libraries are available for achieving this, e.g. *angular-dynamic-routing* library [35].

As well as Angular, Vue also provides an official router in its base installation. Its features include nested routes, modular route configuration, route parameters and a view transition effects [36]. It is possible to implement dynamic routing also in Vue.

4.4 Performance

In single-page applications where JavaScript framework is used, the responsiveness and the user experience in client's end relies a lot on the framework's performance. Hence,

the performance of the framework is also a major aspect to be assessed in the evaluation.

Comparing various areas in comprehensive tests would give reliable evidence about the performance. The evaluation is not, however, feasible to perform only with strict comparison of the performance making the test application bigger and bigger, as there are numerous different ways for the application to grow. A big enough number of test cases would lead to an enormous workload, and a result of a smaller set of test cases would not be reliable for the evaluation. A more relevant matter is assessing what options and tools the framework gives for inspecting the performance gaps and correcting them.

The performance of the framework consists of several small technical factors on the application structure, which can be divided into the program's execution phases. The most relevant of these are a startup time and a creating and modifying DOM elements afterwards. Both matters are affected also by the user's environmental differences in the browser: its JavaScript engine, cache utilization and possible need of polyfills.

The user's browser or version is not generally known in advance by the application developer. In some cases, it might be, e.g. the company's internally used browser, but in terms of the framework's overall evaluation, the support of different browsers and performance in them needs to be considered as well. In the following sub-chapters, a couple of different performance-relevant matters are introduced and evaluated.

4.4.1 Startup performance and build times

A startup time is the time elapsed for a single-page application to load and initialize for the first time. In addition to the application's content size which gets downloaded, also processing and rendering takes time.

Figure 8 below shows JavaScript frameworks benchmark table by Stefan Krause [37]. It gives an indicative reference of startup times for each framework. The frameworks are in the following order from left to right: Angular v4.1.2, React v15.5.4, native JavaScript and Vue v2.3.3. The cells in the charts in the first row are startup time duration in milliseconds +- standard deviation for each framework. In the second row are the slowdown ratios of the frameworks versus native JavaScript. Darker color in the cells means faster processing.

| <u>Name</u> | angular-v4 .1.2-keyed | react-v15.5 .4-keyed | vanillajs-k eyed | vue-v2.3.3- keyed |
|---|--------------------------|-------------------------|---------------------|----------------------|
| startup time Time for loading, parsing and starting up | 84.3 ±2.6 (2.1) | 70.0 ±2.9 (1.7) | 40.5 ±9.5 (1.0) | 56.6 ±2.5 (1.4) |
| slowdown geometric mean | 2.08 | 1.73 | 1.00 | 1.39 |

Figure 8. Keyed results for frameworks in startup time [37].

As can be seen in the above Figure 8, startup times are tolerable. They reflect the fact that at least in an application as small as this, startup times are not so divergent that the framework should be evaluated based on it. What should be taken into consideration is how startup times changes when the application grows, which are the biggest factors in each framework regarding this, and what tools it provides.

In React's case, the startup time is usually heavily dependent on what libraries are included in the application, and how these libraries handle the initialization. Very popular *React-router* library, for instance, allows separating the JavaScript content into chunks, to be retrieved from the server separately with each page change. This reduces the initial loading time, as only a chunk at a time will be downloaded.

Angular provides a few ways to configure the startup initialization. Like React-router, Angular provides a tool as part of its official router package, for the code to be chunked and loaded asynchronously on-demand [38; 39]. For the TypeScript compilation, one great advantage is to be able to choose between Just-in-time compilation and Ahead-of-time compilation.

In Just-in-time (JIT) compilation, HTML templates and components are compiled in the client's browser. This gives an advantage to the development point-of-view, as the application is not needed to be built after every change, and the outcome can be immediately seen in the browser.

Ahead-of-time (AOT) compilation means compiling the HTML templates and components in the server before it is downloaded and used by the client. The client gets a pre-compiled application, so it can be rendered immediately [40]. The package size reduces, and startup time will be essentially shorter in this case.

Vue is referred to as the fastest JavaScript framework available. In the newest versions of Vue, it generally has a small but still distinctive vantage in the startup time (See Figure 8). Vue's startup performance is very optimized by default, but it gives also some tools for customizing deployment, such as pre-compiling the templates [41].

4.4.2 Scripting and rendering

React has a unique way of creating the DOM, called a virtual DOM. Simply put, virtual DOM is React's own version of DOM tree, which can be accessed and modified much faster than the regular one in a client's browser memory. This has many benefits in terms of scripting performance.

In practice, virtual DOM works so that React keeps two instances of the same element, one for the real DOM and one for the virtual one. All the changes which are needed to make, are first made into the virtual version. When the element's content is not changing anymore and the difference of the initial and the changed content is known, the last and lengthy operation is performed: inserting it into the real DOM. The advantage of virtual DOM is its much faster performance in DOM operations, which can be also observed later in this chapter, in Figure 9. [42.]

Angular does not have similar functionality as virtual DOM in React, but uses advanced change detection to determine the changes in the model. Angular creates a change detector for each component on runtime. Each change detector is responsible for detecting the changes in its own sections in the DOM, and updating the parts which needs to be updated. Angular's change detection is fast as is, but for making it even faster, Angular defines *Immutables* and *Observables*. Simply put, Immutables limits the change detection to only *reference* of an object, making it unnecessary to check each property of that object. Observables in turn, are for triggering an event for a specific object, and so the change detection can be limited only to the needed properties. [43; 44.]

Vue's runtime performance is much optimized also in this case. Vue has a virtual DOM implementation, like in React, and even slightly more advanced. A minor difference is that when the component state changes in React, it re-renders the entire component sub-tree. Vue handles the dependencies a bit smarter, allowing it to re-render precisely only the needed components. [45.]

Figure 9 below shows the results for JavaScript web frameworks benchmark by Stefan Krause [37]. The chart includes the measured durations of various tasks related to DOM manipulation, for Angular, React, native JavaScript, and Vue. The numbers are duration in milliseconds. A geometric mean value is visible on the bottom row. Darker color means faster processing.

| <u>Name</u> | angular-v4. 1.2-keyed | react-v15. 5.4-keyed | vanillajs-k eyed | vue-v2.3.3- keyed |
|---|--------------------------|-------------------------|------------------------|------------------------|
| create rows Duration for creating 1000 rows after the page loaded. | 193.1 ± 7.9 (1.4) | 188.9 ± 10.9 (1.4) | 138.5 ± 5.8 (1.0) | 166.7 ± 8.6 (1.2) |
| replace all rows Duration for updating all 1000 rows of the table (with 5 warmup iterations). | 197.4 ± 5.3 (1.3) | 201.0 ± 6.4 (1.4) | 148.0 ± 4.5 (1.0) | 168.5 ± 5.0 (1.1) |
| remove row Duration to remove a row. (with 5 warmup iterations). | 46.1 ± 3.2 (1.1) | 47.2 ± 3.2 (1.1) | 42.8 ± 1.9 (1.0) | 52.6 ± 2.7 (1.2) |
| create many rows Duration to create 10,000 rows | 1946.0 ± 41.8 (1.5) | 1852.4 ± 29.0 (1.4) | 1331.1 ± 22.2 (1.0) | 1587.5 ± 33.9 (1.2) |
| append rows to large table Duration for adding 1000 rows on a table of 10,000 rows. | 324.6 ± 10.1 (1.1) | 345.6 ± 10.4 (1.2) | 295.3 ± 12.8 (1.0) | 399.5 ± 11.0 (1.4) |
| clear rows Duration to clear the table filled with 10.000 rows. | 379.9 ± 11.3 (2.2) | 398.4 ± 8.2 (2.3) | 174.8 ± 4.2 (1.0) | 254.5 ± 5.0 (1.5) |
| slowdown geometric mean | 1.38 | 1.40 | 1.00 | 1.26 |

Figure 9. Keyed results for frameworks in various tasks of DOM manipulation [37].

In the fourth column of Figure 9, values are the smallest in total. This states that Vue's performance is, in fact, the fastest compared to two other frameworks. However, also in this case the differences between framework performances are quite small. Even in the case of creating 10,000 rows, the difference between the slowest framework, Angular, and fastest, Vue, is only less than half a second.

A few other things may also hugely affect the scripting and rendering performance of the application. For instance, in many MDM applications, external libraries must be taken into use at some point of development for the needed functionality. These libraries can be partly or fully in charge of scripting or rendering sections in page. External libraries are evaluated in more detail in section 4.5.

There are a few different styles of handling the DOM among the frameworks. React and Vue utilize virtual DOM, and Angular has its own change detection strategy. There are rather small differences in performance, but each one has their own winning areas. A strict comparison of the framework's performances in scripting and rendering does provide some guidelines, but differences become significant only in the case of rendering very huge data structures. Hence, performance differences should not be emphasized in the overall evaluation.

4.4.3 Platform support

For each of the frameworks, there are fully supported browsers, browsers which are not recommended or supported at all, and browsers which works somehow but are needed to supplement with polyfill scripts. Polyfill scripts, or polyfills, are small compatibility packages which implement the missing features in JavaScript for a specific browser. Usually this means fixing the old browser so that it can run newer features, at least in theory. In practice, the performance of the polyfills varies a lot. Support can also vary a lot between the browser's different versions.

By default, all three frameworks support all popular browsers. Internet Explorer versions 8 and below are only partly supported and might need polyfills. Hence, all of the frameworks are quite even in this case. What needs to be taken into account, however, is React's need for external libraries. React usually requires external libraries for various functionalities which have their own requirements for the browser, and so the need of polyfills might alter.

4.5 External libraries and user experience components

External libraries provide features which the library or the framework does not provide as out-of-the-box features. External libraries are used for implementing features, adding

developer tools, improving the performance or enhancing the user experience. The user experience (UX) is a broad characteristic of an application, and so using an external library can offer more targeted and comprehensive solutions for different use cases. For instance, a high-performance, broadly compatible and scalable chart may be challenging to implement. Some libraries focus only on it. This section creates a short overview to the frameworks' need for libraries, and presents the current supply.

React has a huge number of libraries available, including user interface libraries, user-experience libraries, routing libraries, and even libraries for making new libraries. React's community is highly active in this sense, and usually there are always a group of developers focusing on some specific functionality. These solutions might be more optimized compared to Angular or Vue, which are trying to cover all the functionalities at once.

For Angular, there are less external libraries, but usually all commonly needed functionalities are covered with viable option. UX libraries are especially popular in Angular, and some of these are referenced even in the official Angular website.

Vue has a relatively good variety of external libraries. These libraries include options for UI, responsiveness, state management, typescript decorators, lazy loading, animations, localization, requests, styling, and developing tools. Vue has a couple of companion libraries, such as *vuex* and *vue-router*, which are officially supported and always updated to meet the core library requirements [46].

As Angular and Vue are referenced as *frameworks*, they aim to support more areas, preferably acting completely independently. React is labelled as library, and so it does not aim to provide all the features, but works more as a platform. Most preferably, the React application also requires external libraries, while Angular and Vue works as their own.

4.6 Localization

Being able to change language in the application is mandatory in many cases. Localization is also a big factor in the application's user experience.

Characteristically, React does not provide localization features as such, but one option is to use external libraries. One of the most popular solutions is a simple *react-localization* library [47]. In *react-localization*, localized strings are defined in arrays, for each language separately, and API methods are used for setting or changing the current language. Formatted strings can be then used in a component.

Angular has the official implementation for localization as part of the framework. It provides a wide range of configurable options, such as support for pluralization and nested expressions [48]. There are also alternatives for localization (e.g. *ngx-translate*, *angular-l10n*) as external libraries [49].

Vue does not offer an official localization library, but there are many options for this functionality, e.g. *vuex-l10n* and *vue-localize*.

For all three frameworks, there are many possibilities for localization. Angular is the only one to provide fully working and extensive solution by default, and so it may reduce the developer's amount of work.

4.7 Documentation and community support

The documentation of the framework is the main source of information for a developer and thus it should be considered when evaluating the feasibility of the framework, as well as how huge and active the framework's community is. For the new frameworks especially, the community support can be missing or scattered for some structures, and for that reason the documentation is an even bigger aspect. Insufficient documentation can cause a slow progress in application development or even an interruption.

In React, documentation is comprehensive and describes all the common functionalities of the framework [50]. It has a structure that is easy to read and understand and it is consistent. There is also a separate tutorial section [51] with hand in hand progression, which allows the developer to follow along. The developer can proceed with either writing the code straight to the browser, or by making his own project and writing to editor. The tutorial starts with installing React, and ends with a fully working example of tic-tac-toe game. It includes also well-adapted theory sections every here and there.

React has one of the biggest community support currently among all JavaScript frameworks. React has own discussion forums and chat, and users can follow the latest news and participate in the discussion on Facebook and Twitter.

Angular has also separated the tutorial section from the fundamentals and theoretic part. The tutorial section is highly detailed and includes step-by-step instructions in each section. The documentation theory part starts with an architecture section, which provides an overview of the Angular modules and components and relationship between them, as well as the templates, data binding, directives and services. After these, the documentation focuses on templating, bootstrapping, and forms. As characteristic to a full framework, there are many features in Angular, but the documentation still covers each of them quite well. Angular has also an official forum in Google Groups. [52.]

The newest version of Angular has already achieved big community support. It is noteworthy that also AngularJS (prior version to Angular) still has huge community support [9]. This may indicate that large community support is moving towards Angular (the newest version) at some point.

Vue's documentation is coherent and has united theory and tutorials giving accurate examples for each section as the reader proceeds. Writing is more from the theory point-of-view, but is also giving practical tasks and hints along the way for the developer to test in his own environment and keeping on track in that way. The documentation covers all the essential parts of the framework: Vue instances handling, template syntax, directives (Vue's special attributes), properties, watchers, classes, styles, conditional and list rendering, form bindings, and more advanced features such as routing and state management. There is also plenty of helpful information for the developer about the deployment, testing, and migration from previous versions of Vue. Vue's documentation includes also a section for comparing it to the other frameworks. [53.]

Vue has growing community support, and official and well-maintained resources for asking questions, getting answers and contributing other ways: forum, real time chat and GitHub repositories.

Community support and popularity are usually tightly correlated. Popularity is handled in more detail in the section Popularity and future (4.9).

4.8 Learning curve and developer experience

Presumably, a developer who is starting the JavaScript development for the first time with a framework has some previous experience in JavaScript programming, but not long-time maturity in frameworks. JavaScript frameworks field is quite new. Getting into a new programming technology always requires time, but the learning curve can differ among the frameworks. The developer experience, including installation, error messages and compilation time, is also a thing to pay attention to in the overall evaluation.

React is not the easiest option in terms of the learning curve, but settles in somewhere between Angular and Vue. The biggest impact on the curve might be the use of JSX (defined in section 4.1).

Angular might have the steepest learning curve, partly because it essentially requires TypeScript [54; 55; 56]. Especially for newcomers in the JavaScript frameworks, other two options (React and Vue), may be easier or faster to approach.

Vue has the lowest learning curve, at least according to its own development team. Only the HTML and ES5 JavaScript familiarity is needed at first, and the syntax is simple and rather easy to adopt. Vue uses ESLint as a default linter. ESLint is a linter tool, which has the responsibility of analysis of the code, identifying and reporting errors. With default config, Vue's ESLint rules are rather strict. For instance, semicolons and unused variables are not allowed at all. Some developers might find this disruptive. However, the configuration can be customized.

Each framework can be installed from Node.js NPM package manager. All three frameworks are also available in Bower package manager, and each of them can also be installed easily by downloading a single script file to include. All three also have their own Command-line-interface (CLI) for scaffolding new projects or components. One more thing affecting the developer experience is the compile time of the framework. Compile times are, however, praiseworthy in every case. React's and Angular's sample applications both compile in 150-400ms, and Vue is even faster, 60-150ms.

All frameworks provide a developer tools addon for browsers for an improved debugging experience. For instance, the developer tools addon allows the developer to see and edit the current component's properties and see detailed information about the component.

React and Vue have developer tools addons for Chrome and Firefox and Angular only for Chrome.

4.9 Popularity and future

Major questions in JavaScript framework's feasibility for companies' main development direction are how popular it is and what kind of prospects it has. Selecting a dying framework could be fatal for a company, and would most likely lead to losing a great deal of money and time. The popularity affects greatly on community support, and thus to the efficiency of application development. The purpose of this section is to examine the current popularity of the frameworks, and estimate how guaranteed the continuity of the development process in each case is.

One measurement method used in this section is HotFrameworks online measurement service [9], and its collected data. HotFrameworks is a website which gives relative scores to frameworks, based on their GitHub stars, and Stack Overflow number of questions tagged to the framework. The scores are normalized to scale of 0-100. [57.]

Figure 10 below shows the popularity of React, Angular, and Vue, based on HotFrameworks' scores. Dots in the chart show a specific value for each date, and trendlines are for visualizing the overall growth rate of popularity over time. The chart presents the change over the period from November 7, 2016, to November 12, 2017.

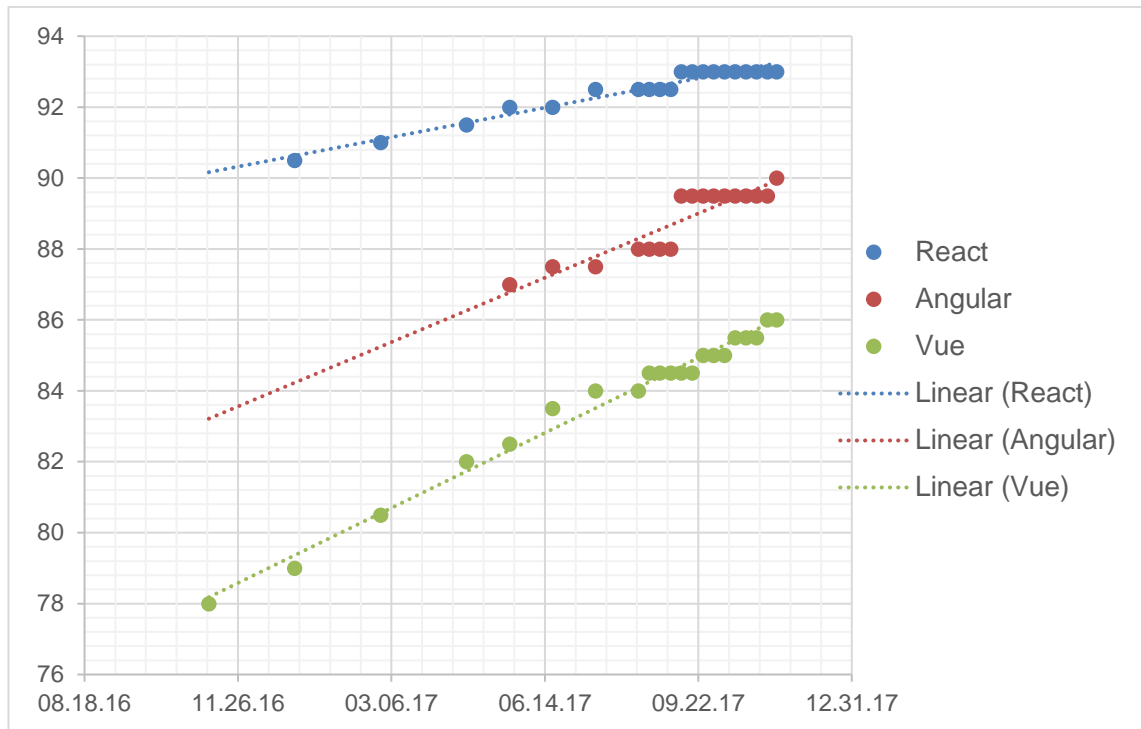


Figure 10. Popularity of React, Angular, and Vue according to HotFrameworks relative scores (based on GitHub stars and Stack Overflow questions). Data retrieved from HotFrameworks service. Full data, November 07, 2016 – November 12, 2017.

According to the measurements illustrated in above Figure 10, React (blue line) has had the flattest and slowest growth during this year. Angular's popularity (red line) has incremented rapidly over the year, but have measurements only after last May. Vue (green line) has an evenly steep rise with Angular, but in addition, has the most measurements for the longest period. As is obvious based on GitHub stars and Stack Overflow questions, React has currently the biggest share of popularity, but Angular and Vue are both rising rapidly.

For evaluating the change in popularity in the future, one method is to take a sample of measurements over a shorter period. In Figure 11 below, there are the latest measurements over the period from May 22, 2017, to November 12, 2017, and the extended trend line for 360 days after the last measurement.

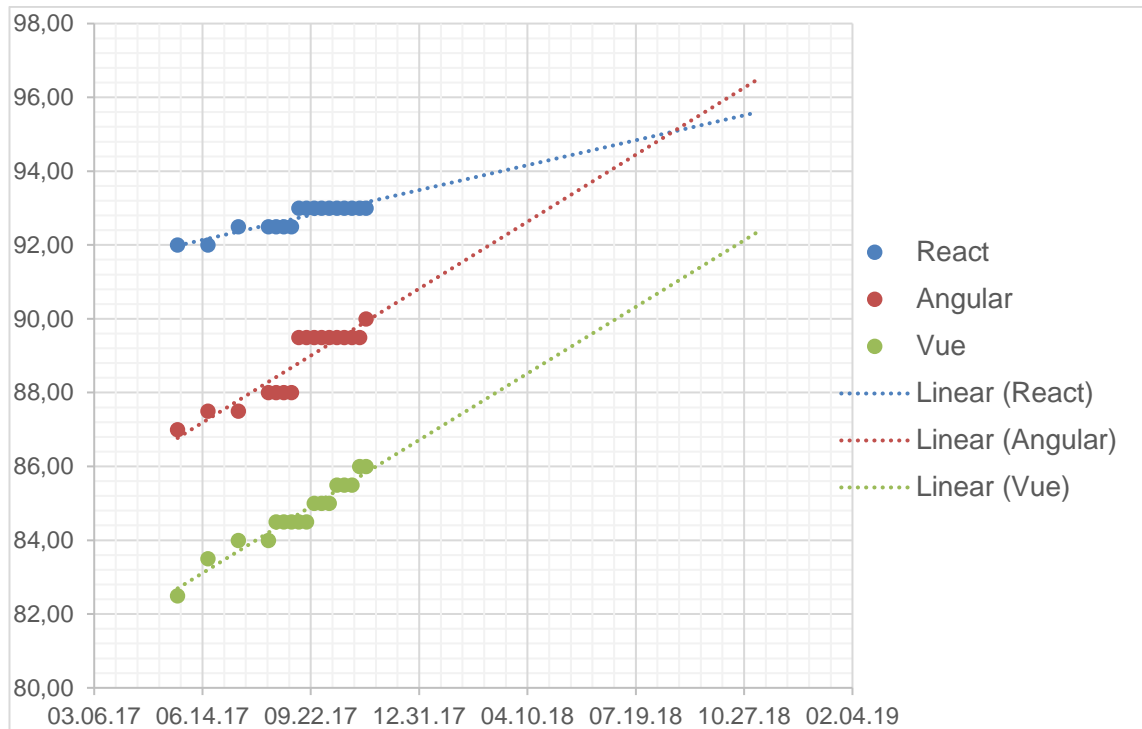


Figure 11. Popularity of React, Angular, and Vue according to HotFrameworks' relative scores (based on GitHub stars and Stack Overflow questions). Data retrieved from HotFrameworks service. Period May 22, 2017 to November 12, 2017 (+360 days).

As can be seen in Figure 11, with the current growth rate, Angular could reach React's popularity during next year, based on HotFrameworks service's data.

Things affecting the HotFrameworks score are GitHub stars and Stack Overflow questions. GitHub star ratings of the frameworks are currently (13 November 2017):

- React, 80.963 stars
- Angular, 29.968 stars
- Vue, 73.543 stars.

Stack Overflow questions asked based on tag are currently (13 November 2017):

- React, 63.653
- Angular, 81.544
- Vue, 10.963

These ratings reflect developers' interest in the technologies. Combining the figures, React and Angular are taking the lead.

Indicative popularity of the frameworks can be also measured with the number of current jobs. Business and employment-oriented social network LinkedIn return worldwide:

- 75.172 jobs as a result for keywords:
React OR ReactJS OR React.js
- 21.267 jobs as a result for keywords:
Angular4 OR "Angular4" OR Angular2 OR "Angular 2" OR Angular NOT AngularJS NOT Angular.JS
(NOT keyword is for excluding AngularJS jobs)
- 17.082 jobs as a result for keywords:
Vue OR VueJS OR Vue.js

These results indicate companies' current interest in the different frameworks worldwide. Based on the results, React seems to have the best job possibilities. What needs to be noticed is that the most recent Angular's version is quite new (initial release in September 2016), so companies might be unwilling to take it into use immediately. Still, it passes Vue, which was released in early 2014.

Google Trends is a Google service which keeps track of current trends, based on Google searches. In Figure 12 below are Google Trends' trend lines for:

- React, blue line, using keywords:
React.js + ReactJS + React -"react to" -"react to that"
(Some irrelevant but popular keywords were cut off with minus (-) character)
- Angular, red line, using keywords:
"Angular 2" + "Angular 4" + Angular2 + Angular4
- Vue, yellow line, using keywords:
Vue.js + VueJS



Figure 12. Google Trends trend lines for React, Angular and Vue, 1 Jan. 2013 – 13 November 2017 [58].

According to trend lines in Figure 12, Angular has been far more popular search worldwide than the two other frameworks, for the last two years.

One notable thing is also that React is maintained and backed by the large company Facebook, and Angular, in turn, by Google. Vue is not supported by big companies, but only a relatively small core development team with 18 developers and 17 community partners who are more or less participating in the development.

4.10 Conclusion

As mentioned in the beginning of the chapter, some aspects are more relevant in the evaluation of the frameworks, while some others have less impact on the overall evaluation. In this kind of application development, major aspects to consider are:

1. Fulfillment of the requirements. It must be possible to implement the necessary features using the framework. In general, only transferring information with the server, data grid possibility with or without external libraries, and routing capabilities can be considered as requirements.
2. Framework's continuation of development. Framework's future development must not stop as it would be fatal for companies utilizing the framework.
3. High effectiveness. The development must be undertaken efficiently, taking into account ready-made features in the framework for purpose, team's previous familiarity with the framework and documentation and community support.

All three evaluated frameworks, either directly or with external libraries, cover the general requirements for an MDM application. Considering the last two aspects, the most important evaluated factors are popularity and future (4.9), component's feasibility (4.1), data binding and state management (4.1, 4.2), and documentation (4.7).

In Table 1 below is a summary of feasibilities of evaluation factors, estimated on a scale of one to three. The impact on the outcome is shown in the second column, followed by an estimated grade for each framework. The most important factors are marked in green. Grand totals are visible on the bottom row.

Table 1. Summary of the feasibilities of evaluation factors and their impact on the outcome. Estimated grade for each framework on a scale of one to three.

| Evaluated factor | Impact | React | Angular | Vue |
|--|--------|-----------|-----------|-----------|
| Components | 3 | 2 | 2 | 2 |
| Data binding and state management | 3 | 2 | 3 | 3 |
| Routing | 1 | 2 | 2 | 2 |
| Startup performance and build times | 1 | 2 | 3 | 2 |
| Scripting and rendering | 1 | 2 | 3 | 3 |
| Platform support | 2 | 2 | 2 | 2 |
| External libraries and user interface components | 2 | 3 | 2 | 2 |
| Localization | 1 | 2 | 3 | 2 |
| Documentation and community support | 3 | 2 | 2 | 3 |
| Learning curve and developer experience | 2 | 2 | 1 | 3 |
| Popularity and future | 3 | 2 | 3 | 2 |
| Grand total | | 23 | 26 | 26 |

As can be noticed from the grand totals in the bottom row of Table 1, a direct comparison of all grades sets Angular and Vue above React in terms of feasibility. When considering only the factors with the biggest impact, the result is still the same. The grades are also very even in both cases between Angular and Vue.

Vue has the smallest popularity currently, and it is the newest framework from the three (when taking Angular's previous versions into account for Angular), and it does not have big company support behind it. As clarified in section 4.9, it has very high potential in its features and in GitHub stars, but still relatively small interest among people based on the number of LinkedIn jobs and Stack Overflow number of questions asked. When compared to React or Angular, there are relatively high risks of development stalling or other unexpected obstacles, e.g. decreasing community support. Due to these reasons, Vue

is not yet suitable for the company's main development platform and thus is not a reasonable choice for an MDM application development currently either.

The popularity of React and Angular is comparatively equal, or Angular's popularity is slightly ahead. Angular has a high number of questions asked in Stack Overflow, while React takes the lead in number of stars in Github. Angular has still a significantly higher number of Google searches. As mentioned in section 4.7, one more aspect to consider is the current popularity of AngularJS, predecessor of Angular. This might indicate a big increase in its successor's (Angular) popularity in near future, when AngularJS is eventually fading out. The number of available jobs is very high for React versus Angular, but also in this case, AngularJS jobs are not considered in this count.

React offers only one-way data binding, while Angular provides both options, one-way and two-way. Two-way data binding has some problems in terms of manageability especially in big applications (described in section 4.2), but Angular application can also be implemented using only unidirectional data, and so performing equally well with React. In React, two-way data binding is possible to implement with external libraries. At some point in the development of the application, two-way data binding might become essential or turn out much more productive, and in this case, Angular would offer a more straightforward approach.

Documentations of React and Angular are rather equally comprehensive as both provide excellent tutorials and more advanced information. In addition, both provide also resources for community support, including forums and chat. There is no reason to favor one over the other in this matter.

An article by Nikolas LeBlanc [59] states that Angular 4 is the ideal choice for enterprises' front-end development due to standard conventions, familiar design patterns, strongly typed code, and server-side rendering. According to the article, front-end development with Angular enables high performance and scalable applications. Furthermore, Justin Goodhew writes in LinkedIn [60] that in his company Angular was chosen for software development work due to its technology consistency, community support, clear best practices and minimal unknowns. One aspect considered as a downside of React, which is considered also in this study, is React's high reliance on others. Angular does not rely on others than Google and their development team. Also, many other composition's evaluable factors are the same as in this evaluation.

Angular has a big advantage regarding options of data binding. Multiple data binding options might be essential in the long term in application development. Angular has a notable advantage also in popularity according to Google Trends and HotFrameworks, and based on Stack Overflow number of questions. Angular is also a highly recommended choice for enterprises according to the articles referenced earlier [59; 60], partly because of its technology consistency and low reliance on others, and partly due to big community support, which is also confirmed during this evaluation process. Because of these reasons, the Google's Angular framework is considered to be the best option for serving the needs of master data management and the most suitable for Prodacapo's front-end development platform.

5 Proof of concept

5.1 Introduction

This proof of concept is a demonstration the purpose of which is to verify that the selected framework has potential and is feasible for its purpose. Hence, this chapter's objective is to verify that an MDM related application is possible to implement with the Angular framework, filling the commonly used functionalities in MDM applications. This chapter is also to ensure that future development is not hindered due to unexpected obstacles. The application to be implemented does not represent any real outcome, but works as a proof of concept only.

5.2 Demonstration

In this section, a test application is implemented and tested with the Angular framework. The test application implements at least basic data table and grid functionality, has routing functionality and a main menu and can transfer data with the server.

A realistic PC environment and configuration are used in the implementation, taking into account an assumed use case in Prodacapo. The assumed use case is a computer with Windows 10 operating system with a performance slightly better than mid-level. Angular version 5.0.1 was selected into the test application implementation, even though the

evaluation chapter concerned version 4.0.0. The framework's newest version should perform at least equally well as the previous one, and the development should not depend on the specific version of the framework. Angular quick start documentation [59] was used in the test application project.

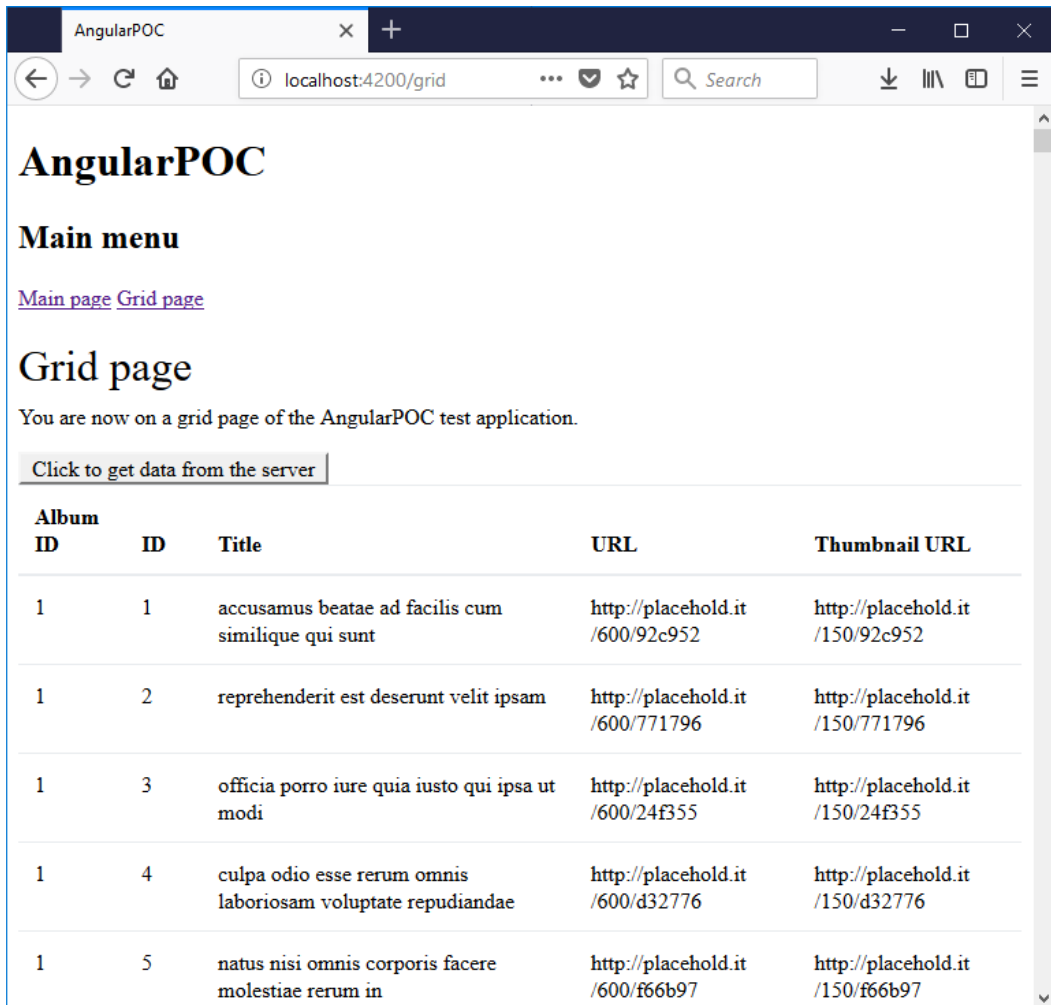
First, Node.JS and NPM package manager were installed. Node.js NPM was used to install the latest version of Angular CLI (Command Line Interface). Angular CLI is a command line tool for making new projects and components in Angular. A new Angular project was created with the tool, and tested to confirm its functionality.

Next, to verify router capabilities, *angular-router* library was taken into use. It is the router library included in Angular, and needed to be configured. Angular-router was configured so that the application included two different pages, a main page and a grid page. Each page had its own specific template file, a stylesheet file, and a component file written in TypeScript, as natural to Angular application. Angular router was configured to be able to change between the pages dynamically, and to provide a main menu.

In the grid page, the other two requirements of an MDM application were then to be verified. First, a button for getting data from a web server was added to the page. The component of the grid page was equipped with a function which creates HTTP GET requests to JSONPlaceholder web server [62]. JSONPlaceholder is an online RESTful web service, testing tool meant for developers. A response from JSONPlaceholder is a JSON formatted array which consists of integer columns "albumId" and "id" and string columns "title", "url", and "thumbnailUrl". The data returned from the request is then saved as a local object, so it can be used by Angular. The data of the response includes 5000 rows of sample data.

Finally, a click event handler was attached to the created button, so that the component's freshly created function executes when the button is clicked. An HTML table element was then defined in the template, the place for storing the results from the web server. The results were bound in the view with Angular *ngFor* command. NgFor executes a loop through the data collection, and creates a row for each item in the results.

The test application was then started and tested. The picture below (Figure 13) shows a screenshot of the running test application.



AngularPOC

Main menu

[Main page](#) [Grid page](#)

Grid page

You are now on a grid page of the AngularPOC test application.

[Click to get data from the server](#)

| Album ID | ID | Title | URL | Thumbnail URL |
|----------|----|--|---|---|
| 1 | 1 | accusamus beatae ad facilis cum similique qui sunt | http://placeholder.it/600/92c952 | http://placeholder.it/150/92c952 |
| 1 | 2 | reprehenderit est deserunt velit ipsam | http://placeholder.it/600/771796 | http://placeholder.it/150/771796 |
| 1 | 3 | officia porro iure quia iusto qui ipsa ut modi | http://placeholder.it/600/24f355 | http://placeholder.it/150/24f355 |
| 1 | 4 | culpa odio esse rerum omnis laboriosam voluptate repudiandae | http://placeholder.it/600/d32776 | http://placeholder.it/150/d32776 |
| 1 | 5 | natus nisi omnis corporis facere molestiae rerum in | http://placeholder.it/600/f66b97 | http://placeholder.it/150/f66b97 |

Figure 13. AngularPOC test application grid page, with retrieved sample data results from JSON-Placeholder web server [62].

In Figure 13, the test application is running, and opened to its grid page. The data has been retrieved by clicking the button. A table of 5,000 rows is displayed correctly, in the table.

The test application implementation was done with ease, and no obstacles occurred. The test application functionality was verified in Internet Explorer, Google Chrome, and Mozilla Firefox browser. The test application fills its purpose for verifying the framework's capabilities for starting MDM application development.

6 Conclusion

During the research work of this thesis, it became very clear that all three evaluated options are excellent choices, when compared to older frameworks for instance. The differences between the top three were all rather marginal. If a development team of a company has long experience in some of the biggest frameworks, then that framework is most likely the best option for the team.

In the conclusion section of the evaluation, only a handful of matters affected the selection. These were, however, considered much bigger aspects, as there were no real obstacles in any of the evaluated items. If the differences in these most significant factors were only minor, more attention would have been paid on other evaluated aspects. Angular was selected as the most feasible framework when considering Prodacapo's stage and needs. Even though Prodacapo's team members had some previous experience of Angular, for the most part it was chosen due to its prospects, data binding options, and big company (Google) support. These were considered as the most important factors in the framework's suitability.

In this study, the feasibility of the frameworks was evaluated based on elaborate and reasonable aspects, and a distinction was made between the relevant and the irrelevant matters considering the target application to be developed. There is quite a large scale of MDM applications, and thus, any strict requirements could not be set for the framework. The necessary features in an MDM application are not significantly different from any other web application, and thus the evaluation is applicable in a more global scope as well.

One objective of the study was to verify the feasibility of the selected framework. For this purpose, a test application was developed with Angular which implemented some basic functionalities characteristic to an MDM application; routing, data table possibility, and web server communication. As mentioned in the beginning, there were no exact requirements for the application to be implemented with the selected framework, but still, the framework needed to be verified. Hence, the functionalities considered as most appropriate were selected so that the feasibility could be confirmed. The test application implementation was simple, but secured two important things: an MDM application is feasible to implement with Angular, and the development can be started immediately.

This study provides thorough research regarding current single-page application technology and a detailed evaluation of JavaScript frameworks. According to Prodacapo, this thesis achieved its objectives comprehensively, by the precise evaluation process and verified outcome. The proposed framework has good potential to be used in FCG Prodacapo Group. Aspects introduced in this thesis can also be used as references for the reader to be able to make their own conclusions in similar cases.

References

1. What is Ajax? [online]. IBM Knowledge Center. IBM; [cited 30 October 2017]. Available from: https://www.ibm.com/support/knowledge-center/en/SSD28V_8.5.5/com.ibm.websphere.wdt.doc/topics/cajax.htm
2. Frey R. Model-view-controller [online]. Wikipedia. Wikimedia Foundation; 2017 [cited 19 October 2017]. Available from: <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>
3. Bajaj P. Overview of Single Page Application (SPA) [online]. C# Corner. [cited 10 October 2017]. Available from: <http://www.c-sharpcorner.com/blogs/overview-of-single-page-application-spa1>
4. Nagayama K. Deprecating our AJAX crawling scheme [online]. Official Google Webmaster Central Blog. Google; 2015 [cited 20 October 2017]. Available from: <https://webmasters.googleblog.com/2015/10/deprecating-our-ajax-crawling-scheme.html>
5. Silverman M. The Problem with Securing Single Page Applications [online]. Stormpath User Identity API. Stormpath; 2016 [cited 30 October 2017]. Available from: <https://stormpath.com/blog/secure-single-page-app-problem>
6. John R. History of jQuery [online]. SlideShare; 2007 [10 October 2017]. Available from: <https://www.slideshare.net/jeresig/history-of-jquery>
7. Lundiak A. History'n'Evolution of JS MV* frameworks [online]. Work'n'Me; 2015 [cited 21 October 2017]. Available from: <https://worknme.wordpress.com/2014/09/25/history-and-evolution-of-js-mvc-mvv-frameworks/>
8. Andrew F. 8 no-Flux strategies for React component communication [online]. Modern JavaScript with React; 2015 [cited 25 October 2017]. Available from: <http://andrewfarmer.com/component-communication/#the-8-strategies>
9. Find your new favorite web framework [online]. HotFrameworks; 2017 [cited 1 October 2017]. Available from: <https://hotframeworks.com/>

10. Alex I. Top 23 Best Free JavaScript Frameworks for Web Developers 2017 [online]. Colorlib; 2017 [cited 1 October 2017]. Available from: <https://colorlib.com/wp/javascript-frameworks/>
11. Why you should not use AngularJS [online]. Medium; 2015 [cited 1 October 2017]. Available from: <https://medium.com/@mnemon1ck/why-you-should-not-use-angularjs-1df5ddf6fc99>
12. The Death of AngularJS [online]. I Like Kill Nerds; 2016 [cited 21 October 2017]. Available from: <https://ilikekillnerds.com/2015/05/the-death-of-angularjs/>
13. Eugeniya K. 5 Best JavaScript Frameworks in 2017 [online]. Hackernoon; 2017 [cited 1 October 2017]. Available from: <https://hackernoon.com/5-best-javascript-frameworks-in-2017-7a63b3870282>
14. Vue.js Is Good, But Is It Better Than Angular Or React? [online]. ValueCoders; 2017 [cited 1 October 2017]. Available from: <https://www.valuecoders.com/blog/technology-and-apps/vue-js-comparison-angular-react/>
15. Craig B. Best JavaScript Frameworks, Libraries and Tools to use in 2017 [online]. Sitepoint; 2017 [cited 1 October 2017]. Available from: <https://www.sitepoint.com/top-javascript-frameworks-libraries-tools-use/>
16. About stars [online]. Github; 2017 [cited 21 October 2017]. Available from: <https://help.github.com/articles/about-stars/>
17. Daniel D. Understanding MVC Architecture with React [online]. Medium.com; 2017 [cited 25 October 2017]. Available from: <https://medium.com/of-all-things-tech-progress/understanding-mvc-architecture-with-react-6cd38e91fed>
18. Bill F. How was the idea to develop React conceived and how many people worked on developing it and implementing it at Facebook? [online]. Quora.com; 2015 [cited 28 October 2017]. Available from: <https://www.quora.com/React-JS-Library/How-was-the-idea-to-develop-React->

conceived-and-how-many-people-worked-on-developing-it-and-implementing-it-at-Facebook/answer/Bill-Fisher-17

19. Template syntax [online]. Angular; 2017 [cited 26 October 2017]. Available from: <https://angular.io/guide/template-syntax>
20. Stephen F. Branding Guidelines for Angular and AngularJS [online]. Angularjs.blogspot.fi; 2017 [cited 25 October 2017]. Available from: <http://angularjs.blogspot.fi/2017/01/branding-guidelines-for-angular-and.html>
21. Mary J. Microsoft takes the wraps off TypeScript, a superset of JavaScript [online]. ZDNet; 2012 [cited 22 November 2017]. Available from: <http://www.zdnet.com/article/microsoft-takes-the-wraps-off-typescript-a-superset-of-javascript/>
22. Meet the Team [online]. Vue.js; 2017 [cited 26 October 2017]. Available from: <https://vuejs.org/v2/guide/team.html>
23. Components and Props [online]. ReactJS; 2017 [cited 20 October 2017]. Available from: <https://reactjs.org/docs/components-and-props.html>
24. What is JSX? [online]. JSX; 2013 [cited 12 October 2017]. Available from: <https://jsx.github.io/>
25. ReactJS – JSX [online]. Tutorialspoint; 2017 [cited 12 October 2017]. Available from: https://www.tutorialspoint.com/reactjs/reactjs_jsx.htm
26. Component [online]. Angular.io; 2017 [cited 14 October 2017]. Available from: <https://angular.io/api/core/Component>
27. Lifecycle Hooks [online]. Angular.io; 2017 [cited 14 October 2017]. Available from: <https://angular.io/guide/lifecycle-hooks>
28. Dynamic Component Loader [online]. Angular.io; 2017 [cited 14 October 2017]. Available from: <https://angular.io/guide/dynamic-component-loader>

29. Components [online]. Vue.js; 2017 [cited 1 October 2017]. Available from: <https://vuejs.org/v2/guide/components.html>
30. Thinking in React [online]. ReactJS; 2017 [cited 1 October 2017]. Available from: <https://facebook.github.io/react/docs/thinking-in-react.html>
31. Can anyone explain the difference between Reacts one-way data binding and Angular's two-way data binding [online]. Stack Overflow; 2015 [cited 1 October 2017]. Available from: <https://stackoverflow.com/questions/34519889/can-anyone-explain-the-difference-between-reacts-one-way-data-binding-and-angular#answer-34520204>
32. What are the exact demerits of two-way data binding? [online]. Hashnode; 2015 [cited 1 October 2017]. Available from: <https://hashnode.com/post/what-are-the-exact-demerits-of-two-way-data-binding-ciibz8fnq01f8j3xthmjjs6di>
33. Two-Way Data Binding: Angular 2 and React [online]. Accelebrate; 2016 [cited 1 October 2017]. Available from: <https://www.accelebrate.com/blog/two-way-data-binding-angular-2-and-react>
34. Routing & Navigation [online]. Angular.io; 2017 [cited 30 September 2017]. Available from: <https://angular.io/guide/router>
35. Angular Dynamic Routing [online]. NPM; 2016 [cited 30 September 2017]. Available from: <https://www.npmjs.com/package/angular-dynamic-routing>
36. Vue-router [online]. Github – vue-router; 2017 [cited 30 September 2017]. Available from: <https://github.com/vuejs/vue-router>
37. Stefan K. Results for js web frameworks benchmark -round 6 [online]. Stefankrause.net; 2017 [cited 30 September 2017]. Available from: <http://www.stefankrause.net/js-frameworks-benchmark6/webdriver-ts-results/table.html>

38. NgModules [online]. Angular.io; 2017 [cited 6 October 2017]. Available from: <https://angular.io/guide/ngmodule>
39. Lazy Loading a Module [online]. Angular 2 Training Book; 2017 [cited 6 October 2017]. Available from: <https://angular-2-training-book.rangle.io/handout/modules/lazy-loading-module.html>
40. The Ahead-Of-Time (AOT) Compiler [online]. Angular.io; 2017 [cited 10 October 2017]. Available from: <https://angular.io/guide/aot-compiler>
41. Pre-Compiling Templates [online]. Vue.js; 2017 [cited 10 October 2017]. Available from: <https://vuejs.org/v2/guide/deployment.html#Pre-Compiling-Templates>
42. The difference between Virtual DOM and DOM [online]. React Kung Fu; 2015 [cited 10 October 2017]. Available from: <http://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/>
43. Change detection with Observable vs Immutable [online]. Stack Overflow; 2016 [cited 10 October 2017]. Available from: <https://stackoverflow.com/questions/34313661/change-detection-with-observable-vs-immutable#answer-34815684>
44. Angular change detection explained [online]. Thoughttram; 2016 [cited 13 October 2017]. Available from: <https://blog.thoughttram.io/angular/2016/02/22/angular-2-change-detection-explained.html>
45. Comparison with Other Frameworks [online]. Vue.js; 2017 [cited 13 October 2017]. Available from: <https://vuejs.org/v2/guide/comparison.html>
46. Comparison with Other Frameworks - Scale [online]. Vue.js; 2017 [cited 13 October 2017]. Available from: <https://vuejs.org/v2/guide/comparison.html#Scale>
47. React-localization [online]. Github – react-localization; 2017 [cited 16 October 2017]. Available from: <https://github.com/stefalda/react-localization>

48. Internationalization [online]. Angular.io; 2017 [cited 16 October 2017]. Available from: <https://angular.io/guide/i18n>
49. Angular-I10n [online]. Github – angular-I10n; 2017 [cited 16 October 2017]. Available from: <https://github.com/robisim74/angular-I10n>
50. Hello World [online]. React Docs; 2017 [cited 1 October 2017]. Available from: <https://reactjs.org/docs/hello-world.html>
51. Tutorial: Intro To React [online]. React Docs; 2017 [cited 1 October 2017]. Available from: <https://reactjs.org/tutorial/tutorial.html>
52. What is Angular? [online]. Angular Docs; 2017 [cited 1 October 2017]. Available from: <https://angular.io/docs>
53. Introduction [online]. Vue Guide; 2017 [cited 1 October 2017]. Available from: <https://vuejs.org/v2/guide/>
54. Learning Curve [online]. Vue.js; 2017 [cited 21 October 2017]. Available from: <https://vuejs.org/v2/guide/comparison.html#Learning-Curve>
55. Eric M. Angular 2's Learning Curve [online]. Edm00se.io; 2017 [cited 21 October 2017]. Available from: <https://edm00se.io/web/angular-2-learning-curve/>
56. Abou K. Angular2: A couple of months in [online]. Medium; 2017 [cited 21 October 2017]. Available from: <https://medium.com/@abookone/angular2-a-couple-of-months-in-aa75dc2684d9>
57. Frequently Asked Questions [online]. HotFrameworks; 2017 [cited 21 October 2017]. Available from: <https://hotframeworks.com/faq>
58. Google Trends React.js, Angular2 + Angular 4, Vue.js [online]. Google Trends; 2017 [cited 1 October 2017]. Available from: <https://trends.google.fi/trends/explore?date=2013-01-01%202017-11-13&q=React.js%20%2B%20ReactJS%20%2B%20React%20-%22react%20to%22%20-%22re->

act%20to%20that%22,%22Angular%202%22%20%2B%20%22Angu-
lar%204%22%20%2B%20Angular2%20%2B%20Angu-
lar4,Vue.js%20%2B%20VueJS

59. Nikolas L. Angular4: Front End for the Enterprise [online]. Blog Rangle.io; 2017 [cited 15 November 2017]. Available from: <http://blog.rangle.io/angular-4-front-end-for-the-enterprise/>
60. Justin G. Why we chose Angular 2 over React for our enterprise software development work [online]. LinkedIn Pulse; 2016 [cited 15 November 2017]. Available from: <https://www.linkedin.com/pulse/why-we-chose-angular-2-over-react-our-enterprise-software-goodhew/>
61. QuickStart [online]. Angular.io; 2017 [cited 15 November 2017]. Available from: <https://angular.io/guide/quickstart>
62. JSONPlaceholder [online]. Typecode [cited 16 November 2017]. Available from: <https://jsonplaceholder.typicode.com/>