KTH Computer Science
and Communication

Structuring modern web applications

A study of how to structure web clients to achieve modular, maintainable and long lived applications

TIM JOHAN MALMSTRÖM
TJMA@KTH.SE

Master's Thesis at NADA
Supervisor: Olov Engwall
Examiner: Olle Bälter

February 2, 2014

Abstract

This degree project, conducted at Decerno AB, investigates what can be done to create client side web applications that are maintainable for a long time. The focus is on basing the application on an existing framework which both simplifies the development process and helps keeping the application well structured.

Which framework is currently the best is evaluated using a comparison between the currently most popular frameworks. The comparison is done using a set of categories that is defined through discussion with experienced web developers at the principal company for the project: Decerno AB. The alternatives considered the most relevant from the discussion are also implemented and tested for further results to show which framework is the best suited for the solution in the project.

The best solution is a structure that is based on the structure used in Angular JS which is a JavaScript framework developed by Google Inc. The reason to why this framework is the most relevant is its huge community support and that it encourages developers to keep their code well structured. In the solution a set of rules are defined that limit the uses of the framework and at the same time defining a structure that achieves the goal of the project: to create applications that are easy to maintain and long lived.

Referat

Strukturera moderna web-klienter för att få hållbara och långlivade applikationer

Detta examensarbete, utfört på Decerno AB, ämnar att undersöka vad man kan göra för att skapa klientbaserade webapplikationer som går att driva och underhålla under en längre tid. Fokus i arbetet är att basera applikationen på ett existerande ramverk, vilket både underlättar utvecklingsprocessen och hjälper till med att hålla applikationen väl strukturerad.

Vilket ramverk som är det bästa utvärderas genom en jämförelse av olika ramverk som är mest diskuterade i dagens webutveckling. Jämförelsen görs inom en grupp kategorier som definierats genom diskussion med erfarna webutvecklare på uppdragsgivaren för projektet: Decerno AB. Ramverken som anses vara mest relevanta från jämförelsen implementeras även och testas för att få ytterligare resultat att väga in i diskussionen om vilket ramverk som bör användas i lösningen till projektet.

Den bästa lösningen är en struktur baserad på ett ramverk som heter Angular JS, utvecklat av Google Inc. Huvudsakligen för att det är det ramverk som har störst stöd av utvecklare just nu samt att det uppmuntrar utvecklare till att skriva väl strukturerad kod. Lösningen definieras som en lista med regler definierade för att begränsa ramverket till att användas som det är gjort för att användas och samtidigt definiera en struktur som uppnår målet med projektet: att skapa applikationer som är lätta att underhålla och välstrukturerade.

Preface

This degree project is the final task in my Degree of Masters of Science in Engineering. I would like to thank every one that has supported and helped me throughout the project.

A special thanks to:

Olov Engwall for being supervisor with invaluable pieces of advice.
Johan Behrenfeldt, Sven Norman, Leif Pettersson, Mattias Knutsson and **Mats Dahl**, Decerno AB for being available and interested in discussing found results, progress and how to improve the results of the project.
Decerno AB for being the principal company for this thesis providing the necessary tools and environment to work on the project.
Ylva Ersvik for being opponent for my project presentation and helping to improve the report by giving great feedback.
Helen Elemida, Aked Hindi, Joakim Jalap and **Moa Ristner** for reflections on work progress and discussion sessions during the thesis work.

Contents

Chapter 1

Introduction

Chapter 1 is an introduction to the project containing a description and background of the problem this project aims to solve and what approach is used to solve the problem. The approach is followed by a brief explaination of why this approach is scientifically relevant. The chapter also contains a list of abbreviations and concepts commonly mentioned in the report, with a short description of each.

1.1 Background

For many years Decerno AB has been building applications that handle large amounts of information, such as data for case management. The applications have traditionally been built as a server-side application that manages all of the back end data, and a client-side application that displays the data, allowing the user to edit data and return changes to the server.

Lately, many customers have requested web applications instead of having a separate application running on the client, which has provoked a change in the way the applications are being developed. The new structure of the applications has a server that generates HTML pages that are sent to the client for display. This technique in web development is commonly used and the main drawback with it is that the interactions between client and server cannot always be done very efficiently. Knowing this, Decerno AB has previously used some third party javascript frameworks to improve the clients and allow for them to do some of the computations, giving the requested behaviour of the applications.

1.2 Problem description

One major problem that occur from solving problems by importing external libraries during the development process is that the project often gets unnecessarily many external dependencies. This can also lead to the project getting different dependencies in different parts of the project resulting in a unclear structure where noone knows what is used where. Since the imported libraries have been developed separately they are rarely built to handle data in the same way, which also leads to that the data management in the project can be done in different ways in different parts of the project. This becomes a problem when the different parts are integrated and makes the final product less stable since there is no common structure in the project.

Debugging and testing such projects is also very difficult, since the same kind of operations can be done in different places depending on what part of the project you are working with. But it is still doable as long as the members of the development team that built the application knows how the parts that they developed work separately. When such a project is released and needs to be maintained or extended with additional functionality the problem becomes even graver. To add components to these projects someone needs to first understand all of the different structures of the project, find separations between the different structures, and then often add another structure so that the changes to the project does not disrupt the earlier functionality.

1.3 Project Goal

The goal of this project is to find a best-practice solution for the structure of web applications managing large amounts of information. This solution will be found by combining a theoretical analysis in the beginning of the project with practical performance tests done in the later stages of the project. The solution will contain a clearly defined base structure to use from the beginning of a web application project, and possibilities to extend the project without losing the structure.

The solution will be expressed as a list of rules that when followed keeps the application well structured, and keep a clear distinction between the different parts of the application. The questions that the project aims to answer are:

- How should a web application be structured to be maintainable and long lived? (from directory structure to communicational patterns)
- Is there any current framework that can be used as a basis for the solution or does the structure need to be defined from the beginning?
- What factors help making such a structure support being extended and maintained further after release of the application?

The solution found should aim to fulfill as much as possible in a list of important factors listed in section 2.2.

1.4 Approach

There is no reason trying define a structure from the beginning without looking at what has been done previously, therefore the solution to the project aims to be based on an existing framework, so the first step is to see what exists, what solutions might fit the project and how these solutions support being extended later with additional functionality if the framework does not fulfill all requirements of the project by itself. After some different solutions are found they are compared theoretically, practically and tested for performance.

The theoretical evaluation first consists of a brief summary of the considered alternatives, resulting in some alternatives being excluded from further comparison since it becomes apparent that they are not well suited for the project. Then the theoretical comparison is extended further by a more detailed comparison within a set of requirements for well structured web applications.

The practical evaluation of the frameworks is done by implementing an application defined in chapter 2.3, discussing how the frameworks are to work with and whether the frameworks differ between theory and practice.

The performance testing is done to compare how fast and efficiently the frameworks are able to manage data, mostly to confirm that the data management of client side applications is fast enough to provide a good user experience but also to see how the frameworks stand against each other within terms of performance and memory usage.

After this the final solution to the project is determined by whether there are any framework that might fit the solution or not. If there are not any viable frameworks the solution will be defined from the beginning using the method described in 1.6.3, using the determined theoretical requirements and the strengths of the different frameworks to create a solution that fulfills the goal of the project. If there are one or more frameworks that could be used as a basis for the solution, the solution will be based on the most suited framework, extended with rules enforcing the structure and defining how to be able to extend the application further without disrupting the structure.

1.4.1 Scientific methodology

Scientifically the approach used in this thesis is based on a rationalistic methodology. At first there is a comparison between different frameworks, where observations about the frameworks are made using common opinions on the internet and uncontrolled observations. This targets a broad area and many different frameworks to be able to limit down the focus of the thesis and make controlled observations about a smaller group of frameworks. The controlled observations are then used to compare the frameworks further to be able to deduce theoretical conclusions and a hypothesis for what the solution of the project should be based on. After this, further observations are made using experiments with results that hopefully strengthen the hypothesis. The results and observations from the experiments are evaluated using quantitative evaluation, which in layman's terms can be expressed as comparing numerical values. The result and solution to the project are deduced from all of the observations combined, leading to a best practice solution to the problem.

The solution to the project is found using an iterative approach described in section 1.6.3, which will allow for continous improvements in the future.

1.5 Scope & limitations

The project is limited to web applications that need to manage large amounts of data. This does not exclude web applications that mainly show data without managing or changing it, but the discussion in the project will not consider the factors in such applications that do not apply to the applications managing data.

The area is also limited to the client-side of the application. This includes client pages generated on the server and factors of the server that might apply to the clients data management. The rest of the server implementation such as back end database and data management is done by Decerno AB and is not considered in this report.

1.6 History & Related work

1.6.1 JavaScript

JavaScript is a language that was originally created by Brendan Eich in 10 days in May 1995. [63] Originally it was called *Mocha* named by the founder of Netscape. In September the same year the name was changed to *LiveScript* and finally in December changed to *JavaScript* upon receiving a trademark license from Sun. The name in itself was mostly a marketing move since Java was one of the most growing programming languages at the time [6] even though JavaScript did not have anything to do with Java. 1996-1997 JavaScript was converted into a standard called ECMAScript by ECMA [7], with JavaScript being the most popular implementation of the standard. After this different implementations were made and the standard evloved into ECMAScript 2 (1998), 3 (1999), then there were a lot of turmoil with different parties including Microsoft, Mozilla and Yahoo wanting ECMAScript 4 to evolve in different directions. The result of the turmoil was that ECMAScript 4 was delayed and ECMAScript 3.1 was released 2007 (8 years later) wich was mostly based on ActionScript 4, made by Macromedia. During this turmoil many community developed libraries and frameworks made the JavaScript language grow back into the discussion. This eventually led to an agreement of the parties involved and the release of ECMAScript 5 (2009). After the release of ECMAScript 5 there have only been some minor adjustments made, which were released as ECMAScript 5.1 (2011) and is currently the latest version of the standard.

The purpose of ECMAScript based languages is still not defined as providing functionality for web applications but more as an enhancing addition to improve responsiveness and interactions. In the ECMAScript 5.1 language specification it is stated as:

> "ECMAScript is an object-oriented programming language for performing computations and manipulating computational objects within a host environment. ECMAScript as defined here is not intended to be computationally self-sufficient; indeed, there are no provisions in this specification for input of external data or output of computed results. Instead,

> it is expected that the computational environment of an ECMAScript program will provide not only the objects and other facilities described in this specification but also certain environment-specific host objects, whose description and behaviour are beyond the scope of this specification except to indicate that they may provide certain properties that can be accessed and certain functions that can be called from an ECMAScript program." [18]

Summary of this section of the ECMAScript specification shows that the standard that JavaScript is based on is not supposed to be "computationally self-sufficient". The standard is not either made to be able to manage data by itself but is according to the specification expected to get this done by the surrounding computational environment. Part of the aim of this project is to contradict this and show that the language JavaScript can be used for more than what it originally was meant to.

In the ECMAScript specification it is also stated that:

> "ECMAScript was originally designed to be a Web scripting language, providing a mechanism to enliven Web pages in browsers and to perform server computation as part of a Web-based client-server architecture. ECMAScript can provide core scripting capabilities for a variety of host environments, and therefore the core scripting language is specified in this document apart from any particular host environment." [18]

This shows that JavaScript is originally just supposed to be used as a way to "enliven web pages". The main reason for this is that it has not been able to be used for more previously. However, now that more and more frameworks contradict this part of the specification it is probable that the specification will be changed in near future and that the language JavaScript will evolve and be acknowledged to be able to do more than what its current standard states.

1.6.2 Web development

Web development has traditionally been done using a heavy server application that generates pages to be shown in the client. Therefore there is not much work that has been done on the subject of balance between server and client in web development. In 2003 Joseph Williams (Sun professional services) published an article on the called “The web services debate: J2EE vs. .NET” [66]. The big trend in web development back in 2003 was web services, and the discussion about how to structure the projects was only the choice between which server structures to use. JavaScript and client side scripting frameworks did not have the performance to be worth using to a larger extent than a few calculations and UI operations.

With the performance improving, JavaScript grew and started being used for more and more in web pages, enhancing the web technologies for the world. Some small community developed libraries were released and the communifty effort increased rapidly after Jesse James Garret introduced the concept of asynchronous calls to the web clients called *Ajax* in 2005. [9] This made it possible to have truly dynamic content on web pages and made the concept of single page applications (SPAs) possible.

In 2008 Iwan Vosloo and Derrick G. Kourie stated in their research paper “Server-centric web frameworks” [41] that

> "Because of discrepancies between the ECMAScript standard and varying JavaScript implementations provided by individual browsers (and versions of browsers), JavaScript-based frameworks were not really viable in practice for quite some time. As a result, examples of such frameworks are still very immature compared to server-centric frameworks which do not depend on JavaScript."

They also mention that JavaScript can be used in various locations in the code to avoid having to relay all user interactions with a page to the server, and that the most important problem with JavaScript is that it is not always executed the same way in different browsers.

This same year the book “Pro JavaScript Design Patterns” [11] describes the more advanced structures in JavaScript writing object oriented scripts using inheritance between objects and chaining methods. The book was describing some of the core features of today’s JavaScript frameworks. This was where some started using JavaScript to implement processing functionality for the web clients.

So 2008 - 2009 was when the client side processing was starting to be considered for data operations as well as the previous areas of use. This led to more frameworks for client side data management being developed and the release of such frameworks as: Angular JS (2009) [20], Knockout JS (July 5th, 2010) [37], Backbone JS 0.1.0 (Oct 13, 2010) [26]. The first releases were stable and good for handling small amounts of data, but since then both the client computers performance has improved and the frameworks have matured a lot in terms of functionality, stability and performance. Now they might be able to handle applications with large amounts of data as well [47].

1.6.3 Software architecture and design

This section defines what a software architecture is and why it is relevant to have one when developing an application.The section also describes a method that can be used to define a software architecture, which is used in this project.

What is software architecture?

A software architecture is a fundamental structure defined to build an application on. Before starting a project it is necessary to define an architecture that sets a common ground for the whole project, designed to have the best possible foundation to build on. The design decisions when defining the architecture can have a major impact on the development process, planning from the beginning so that minor features of the project might be more difficult to include but the key features will get the best circumstances possible. The architecture definition involves a series of decisions that have a considerable impact on the result of the project. [53] In *Patterns of Enterprise Application Architecture* by Martin Fowler some common themes when defining software architecture are identified as:

> "The highest-level breakdown of a system into its parts; the decisions that are hard to change; there are multiple architectures in a system; what is architecturally significant can change over a system's lifetime; and, in the end, architecture boils down to whatever the important stuff is." [8]

As stated previously, software architecture is very important for a project to be able to do what it is supposed to. Since the design decisions in the architecture are defining the core components of the project it is also very hard to change and it is an absolute worst case scenario for a project to have to change architecture decisions during the development. With the whole development being based on a core architecture it might even be worth restarting the project if main parts of the architecture needs to change.

Method for defining a software architecture

There are many methods described to define a software architecture. All of them bottom down to some kind of process of specifying requirements for the end result, adapting to them with an architecture and trying to find the weaknesses and strengths with the architecture to be able to improve it until as many requirements as possible are fulfilled. The methods are often used iteratively so that when an architecture is found the designer goes back to revising the specifications trying to find what changes could be made to fulfill more requirements to improve the architecture further.

Example of such a technique is defined in Microsoft Application Architecture Guide, 2nd Edition [54] containing 5 steps shown in figure 1.1

1. **Identify architecture objectives.**
   Specify all objectives for the application, breaking the objectives into smaller objectives helps the designer to focus on the right problems in the design.

2. **Identify key scenarios.**
   Use key scenarios to focus your design even further on the key areas of the application. And break these key scenarios down to what functionalities are required. The key scenarios can often be used to evaluate candidate solutions in the later stages of the design.

3. **Create application overview.**
   Study the application's surroundings, where it will be deployed and architecture of surrounding systems. Connect the architecture design to fit in the real world where the application will operate.

4. **Identify key issues.**
   Identify the largest issues in the application to see what the architecture can do to simplify them.

5. **Define candidate solutions.**
   Create a prototype and evaluate it using all of your previously defined requirements and important factors for the application. After this step return to step one to improve your architecture further by being more and more specific for each iteration of the cycle.

The method that is used to find a solution in this project is based on the method described above.

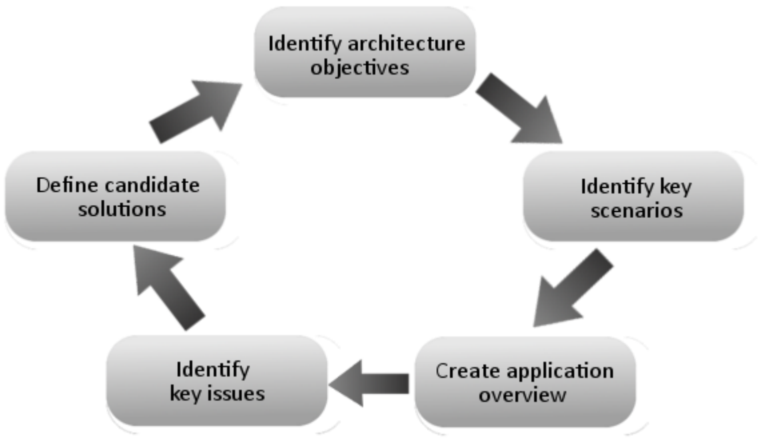
Identify architecture objectives
Identify key scenarios
Create application overview
Identify key issues
Define candidate solutions

Figure 1.1. The iterative steps in creation of the core architecture design

1.7 Definitions

If you are new to the world of web development and interested in learning how to structure your web applications from the beginning, section 1.7 explains some of the basic concepts of web application development that the reader is assumed to have a good understanding of later in this report. If you are already familiar with the concepts and terms below you can skip this section.

1.7.1 Framework

A framework is a set of tools defined to simplify the development of an application. By using frameworks developers can commonly abstract generic levels of functionality using already established solutions instead of having to implement all generic functionality for each application. There is no clear definition for the requirements of a framework other than that it provides a set of functionality and thereby allows developers to use the framework instead of implementing the same functionality on their own.

1.7.2 Web Application

Web application is a generic name for any application software that runs in a web browser. Web applications are divided into a server side and a client side.

Server side

The server side of the application is the basis for the application. This is where a web browser connects to, to request a web application. Then the server sends the client side application to the browser. During an application runtime it is common that the client requests additional data from the server such as new pages or dynamically loaded content.

Client side

The client side of the application is what is received and executed on in the web browser, resulting in what is shown to the user. The client typically consists of HTML pages with JavaScript to add functionality and dynamic content to the application. So to sum up the client side of a web application consists of everything that is sent to and executed by the web browser in the clients computer.

1.7.3 Data bindings

Data bindings are a relatively new concept to something that developers usually implement their own solution to, connecting the view input fields to the data model. A data binding connects these two so that all data changes are done automatically and references to the binding are always updated with the newest content [12]. Data bindings in web applications have recently been implemented as two-way bindings which means that if any side of the binding is updated the other sides will also be updated leading to values that can be updated both programmatically and through user input. Data bindings simplify the development by not needing the developer to create custom bindings and having to copy values between different places.

1.7.4 Code template

A code template is a good way of reusing code. Since applications very often have similar structures in many places, templates avoid duplicating the code by defining it as a template somewhere else and then importing the template wherever that snippet of code is needed [1]. Templates especially simplify the maintainability of the project since changes in a template will follow through the whole project instead of having to check all pages containing the duplicated code and make the same changes in many places.

1.7.5 Pub/Sub

Publish – Subscriber is a pattern that has often been used to notify components of an application about changes in other components. The pattern is built on a structure where all components can publish and subscribe to events so that when a component publishes to an event all subscribers get notified. The benefit of this pattern is that instead of having to keep track of all of the components without knowing which are dependent on which the system only needs to keep track of the events and the subscribers to manage changes in the application correctly. [56].

1.7.6 SPA

SPA or Single Page Application is a deviation from what has been the standard way of web development. The main idea is that instead of navigating between different pages in a web application the browser loads a single page that has the functionality to change the content of the page. Then the same page can show different content without having to reload the root page. The advantage of using this is that a browser is able to maintain a data context between pages that are shown, which in its turn means that all content and parameters of a page does not need to be reloaded repeatedly as it would in the older web applications.

1.7.7 DOM

The Document Object Model (DOM) is the standard way of writing HTML web pages, where the objects in a page are defined as objects using XML syntax.

1.7.8 Architectural pattern

In this report there is some commonly used architectural patterns are mentioned. The patterns are ways to describe how to structure the code by separating what is shown from back end functionality, to separate data management from business logic and so on. This is very necessary for an application to be manageable since problems are much easier to find and fix in a well defined structure. There are many standards for this, however in this report the MVC (Model-View-Controller) and MVVM (Model-View-ViewModel) patterns are the most commonly mentioned. The patterns are explained in further detail later in the report.

1.7.9 Bug

Bug is a term that is very widely used for many different intentions. As Karnow and Curtis A describes in *What's a bug? Whatever the customer says it is.*:

> "What is a 'bug'? The term is used in different ways both in the computer software industry and among users. To some, a bug is an incompatibility or a performance problem. To others, the term can refer to a design flaw or even a misunderstanding about what a product is supposed to do. The problem becomes more significant as systems become more complex and as the need for reliability continues to increase." [40]

Since the term 'bug' is used for so many things this section clarifies that a 'bug' in this report is considered as a non intended behaviour. This excludes incompatibilites, performance issues and design flaws since these are examples of bad planning or bad use of resources.

1.7.10 Business logic

One of the major differences between client and server web applications is where the business logic is placed. Business logic is what encodes real world business rules in an application. This means that it defines the rules that determine how data can be created, displayed, stored and changed. [64] It is often vital for an application to work when needing to interact with a database since the business logic controls data to make sure that it follows used data model and set rules.

1.7.11 Maturity / Stability

When talking about maturity or stability within computer science, this is often a way of describing how likely a component is to change and how likely there are for bugs in the component. Important factors in software maturity are how long the component has existed and how much it has been used. If the component has existed for a long time it is likely that many of the bugs that might have existed at the release have been found and are fixed. A well used component and a component that has existed for long time is also less likely to undergo any major changes, since this could affect the projects using it too much. Maturity/Stability is discussed further in section 2.2.2, and how to assess maturity/stability using a Capability Maturity Model.

1.7.12 JSON

JSON or JavaScript Object Notation is a data format that is commonly used to send data by representing all data as key-value pairs, based on the standard ECMAScript 1999 [39]. A key value pair is defined as *{key : value }*. Example of a JSON object and how it looks in Figure 1.2. Where a person is represented by the fields *firstName*, *lastName* and *age*. The values for each field are easily accessible using the key values.

```
{
    "firstName" : "John",
    "lastName" : "Smith",
    "age" : 25
}
```

Figure 1.2. Example of a JSON object

Chapter 2

Theoretical comparison

There are many different frameworks to consider when using a framework to structure a web application. Chapter 2 first contains a short summary of frameworks that currently are the most relevant for client side web applications. The summary narrows it down to a few frameworks that are discussed further and compared to each other. The comparison then narrows the list down further to decide on only a few frameworks that are the most relevant. These are implemented and tested further in chapter 3 and 4.

2.1 Considered frameworks

This section contains a short summary of the frameworks that are considered to be used in this project. The list comes from research on blogs and popular websites for developers, where developers discuss which frameworks to use for average web applications. The list of frameworks is derived from:

- **InfoQ** Research about which are the most popular JavaScript Frameworks in the beginning of 2013 [17]
- **IBM developer research**
  A research done in 2012 that compared the most relevant frameworks for use in web pages then [51]
- **Blogs**
  Various different blog posts about which frameworks to use have in many cases provoked big community discussions about why the author is wrong and what the correct methods should be, leading to qualitative and often well referenced arguments to what frameworks to use. Example of such is Steven Sandersons blog post*Rich JavaScript Applications - the Seven Frameworks(Throne of JS, 2012)* [55] and other blogs [13] [61] [10] [49] [52].
- **TodoMVC**
  A site which has code examples showing how to implement a to-do list using

different frameworks and discussing some advantages and disadvantages with each. [48]

The initial list contains 6 different frameworks that are the most commonly mentioned and discussed. These are:

- Angular JS
- Knockout JS
- Backbone JS
- Ember JS
- Batman JS
- Meteor

The goal of this section is to use a short summary of each framework to see whether it is possible to narrow the list down to only include the most relevant alternatives further in the project.

2.1.1 Angular JS

Developer

Google Inc. [23]

Description

Angular JS is a framework that focuses on data bindings and dependency injection to create dynamic web applications and link the data model and the view together. Other than the data bindings, Angular also provides much support for the common web application operations, such as form validation and navigation control.

Advantages [+] and Disadvantages [-] with Angular

+ Well used. [23]
+ Is one of the most discussed framework being praised by many in discussions. [49]
+ Extends the HTML language instead of trying to use it for something that it is not made to be. [19]
+ Designed to simplify unit testing, providing support and guidelines to easily get started testing as a developer. [19]
- Is said to be fully extensible with other frameworks, but the general opinion is that Angular is supposed to provide a complete set of operations and is not always compatible with extensions. [23] [62]

- Documentation is known to be quite bad, making the framework difficult to get started with. [19]

Current version

1.1.5 (Stable) (May 23, 2013) [20]

2.1.2 Knockout JS

Developer

Steven Sanderson and Knockout community [38]

Description

Knockout is a data binding framework, creating connections between data layer and view to enable dynamic page creation. The framework is minimal to its purpose (data bindings) and does therefore not interfere in other areas which makes it easy to be extended with other frameworks. Many projects include Sammy JS and JQuery into the solution together with Knockout to achieve a more complete set of operations, covering the areas that other frameworks cover.

Advantages [+] and Disadvantages [-] with Knockout

+ Extensible, Knockout is only supposed to handle the data bindings and is therefore made to be able to coexist with other frameworks. [38]

+ Stable version for over two years and used in many web applications. [37]

+ Uses a common MVVM pattern for structure. [38]

- More of a library than a framework, handles data bindings well, but does not handle other operations. [36]

Current version

2.3.0 (Stable) (July 8, 2013) [37]

2.1.3 Backbone JS

Developer

Jeremy Ashkenas and Community [27]

Description

Backbone JS is a lightweight library that simplifies data management and data based updates in JavaScript applications. The library does not enforce any structure or any specific modeling requirements, but only focuses on adding some commonly needed data management operations to the applications.

Advantages [+] and Disadvantages [-] with Backbone JS

+ Used in well-established web applications such as Sony Entertainment network and Nokia Profiles [28].
+ Very lightweight. [27]
+ Uses a common structure with little hidden functionality, (key-value pairs and event handling). [27]

- Is a library and not a complete framework. [27]
- Is made to work with all other components and frameworks, so does not define or enforce a certain structure. [27]

Current version

1.0.0 (Stable) (March 20, 2013) [26]

2.1.4 Ember JS

Developer

Tilde Inc. [35]

Description

Ember JS is one of the largest frameworks covering all of the commonly used purposes of client side development. The structure is based on the MVC pattern. An opinion in the community is that Ember JS is hard to get into because of its size and it keeps being quite complex even when worked with a lot.

Advantages [+] and Disadvantages [-] with Ember JS

+ Extensive framework containing much functionality and operations. [32]
+ Used in well-established applications and by big companies such as Yahoo and Thoughtbot . [35]

- Extensive framework. Takes up much memory (and much unnecessary memory unless all of the operations actually are used.) [33]

- Has recently reached the first stable release, so is likely to require some updates and bugfixes. [33]

Current version

1.0.0 (Stable)(August 31, 2013) [33]

2.1.5 Batman JS

Developer

Shopify, Nick Small [30]

Description

Batman is framework that manages data bindings in a MVC like pattern. The most important difference from the other frameworks that are mainly JavaScript based is that Batman JS code is written in CoffeScript which is compiled to JavaScript.

Advantages [+] and Disadvantages [-] with Batman JS

+ CoffeScript is compiled before running which supports the development process by avoiding unnecessary bugs in the code since bugs can often be found by the compiler [2].

- Code & the framework itself written in CoffeScript. This is not necessarily a disadvantage but for this project it makes the results less accurate since CoffeScript in itself might affect the results. For keeping a common pattern and structure in a project it is also a major disadvantage to combine different languages so all extensions and other components in the project would need to be written in CoffeScript. [30]

- Has not reached stable release yet. [29]

Current version

0.15.0 (September 27, 2013) [29]

2.1.6 Meteor

Developer

Meteor Development Group [43]

Description

Meteor is mentioned as one of the interesting new alternatives in discussions about client side frameworks [52]. It is described by its developers as:

> "Meteor is a new application platform for this new era. It is built around Smart Packages: little bundles of code that can run on a client, inside a cloud service, or both, and that can manage their lifetime inside the modern distributed environment."
> - Meteor Development Group [45]

Advantages [+] and Disadvantages [-] with Meteor

+ Futuristic, adding new possibilities in web technology [45].
+ Fast to develop in [45].
- Not an established technique [45].
- Not considered stable yet [44].

Current version

0.6.5.1 [44]

2.1.7 Summary

Angular is a framework that has quite recently reached a stable state, and has not yet been used in very many big web applications. However, it is one of the most commonly mentioned frameworks in discussions and has a big community. Another thing that speaks greatly in Angular's favor is that Google Inc. is developing and using it. With the big company backing the development up and using the framework for their projects it is almost guaranteed to keep being maintained and keep being developed further. Therefore Angular is one of the frameworks that are compared further.

Knockout in itself might not be considered as a complete enough framework to be included further, but since it is commonly extended with JQuery and Sammy JS and considered to work well together with these packages it is considered together with these. Knockout is only a part of the framework package, but since it is what determines the structure of the application it is still be referenced to as Knockout. Knockout together with JQuery and Sammy JS is often mentioned as one of the best sets of frameworks. Therefore Knockout is considered further in the comparison.

Backbone is more of a library than a framework. Since it is not able to cover all of the areas that are needed for the project and it does not have any recommended extensions to do so Backbone is not considered further in the discussion and comparison.

Ember is a framework that is already used in many large projects, which improves the credibility for it not to undergo any major changes. The disadvantage of Ember is that it quite recently reached its first stable release. Since Ember is a framework that covers important areas and is considered stable it is included in the later comparison.

Batman is a framework that should be considered by projects working with rails and possibly already with CoffeScript wanting to extend their functionality and get structure to their code. It is however not very interesting for this project since the differences compared to the other frameworks could come from the different back end solution or from CoffeScript itself. Also using a framework written in CoffeScript could make extending the application more difficult since the range of other components written in CoffeScript is smaller than the range of components using JavaScript.

Meteor provides a new way of developing web applications that reduces the work for the developers of the application. However it is not mature enough to be considered using in production applications. So this framework is not considered further.

The frameworks that are included in the comparison with more depth are:

- Angular
- Knockout
- Ember

2.2 Requirements

This section contains a list of important requirements for web applications. The list is derived from discussion with experienced developers at the principal company for the project. Defining what criterias are the most important for web applications. Each requirement is split into one subsection that describes the extent of the requirement and why it is relevant, followed by what and how frameworks support said requirement. The requirements discussed below are

- Maintainability
- Maturity
- Performance (Caching)
- Portability
- Testability
- Modularity
- Popularity

2.2.1 Maintainability

One of the most important aspects for a framework to be easy to maintain in the future is the concept known as “Keep yourself DRY” [16] also known as (Don’t

Repeat Yourself). When maintaining a project and needing to change something it can be a major problem if the same operation is done in multiple places of the code since it requires the change to be done in multiple places. This can often lead to missed changes resulting in problems [57]. It is also much easier to debug problems if the code is kept DRY since the operation that has a problem can be changed without having to be as careful to not affect other functionality. Client side frameworks can support this concept by enabling the developer to create templates to be able to reuse the same code in many places and also by layering the functionality so that specific kinds of operations are implemented separately from the rest of the code.

Another important factor for maintainability is the ability to add additional content to a project after release of the project. For the frameworks this is investigated by checking how they support being integrated with other frameworks that handle other operations and what possibilities they provide to extend the functionality for future requirements.

Angular

From the documentation it does not seem like Angular includes support for writing and reusing templates more than within a page.

Angular is a framework that contain functionality for most important parts of a web application. So if it needs to be extended with additional functionality it is likely to be easier to implement the extra functionality so that it fits the structure defined in the project than to search for a library that fits the structure.

Knockout

Knockout supports the DRY concept using something called *named templates* and through its MVVM structure (described in section 2.2.6). Named templates allow the developer to define HTML paragraphs as templates, giving it names and bind to them later in the code where the template should be displayed. This means that if there are some changes made to the template, the changes will be made all over the project where the template is used. Each instance of a template can have a separate ViewModel applied to it, so the templates can be used with different functionality in different places of the code. However, using a template with different ViewModels is not recommended since it can create a structure that is difficult to understand.

Knockout only handles data bindings so it is simple to combine with frameworks that manage other areas of a web application if an application needs to be extended with additional functionality later on.

Ember

Ember supports templating in almost exactly the same way as *named templates* in Knockout. The main difference is that Ember makes the templating a bit clearer by creating the template with view elements and functionality together and giving it a name. Adding additional functionality that is not included in Ember has the same problem as for Angular since Ember also is a more complete framework and the developer does not want to combine different structures in the same project.

2.2.2 Maturity

The maturity of a framework is basically an evaluation of how it will change in the future and how many bugs/problems are likely to occur from the framework itself. One way to evaluate the maturity of computer software is to use a CMM (Capability Maturity Model), which is what is done in this section. The model used is explained in "Capability Maturity Model for Software" [50] where the maturity stage of software can be expressed in five stages.

1. **Initial** "The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort."

2. **Repeatable** "Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications."

3. **Defined** "The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software."

4. **Managed** "Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled."

5. **Optimizing** "Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies."

For a framework to be used in production applications it is very important that it has reached a high stage of maturity. In the model expressed above the lowest maturity that should be considered in this project is at stage three. In stage three the structure of the framework is set, so that it is possible to be sure what should be done where during the development. There will probably still be changes made to the framework at this stage, which might affect the project in updates, but since the main structure is set, no major changes should be made. It is of course even better if the framework has reached a higher stage of maturity. If a framework is at stage five, implementations using the framework should not have any disadvantages with updates of the framework, but should only get optimizations and improvements.

Another very important factor in maturity is how well established the framework is. If a framework is well established and used in many big projects it is more likely that the development will continue, adding support for future technologies and improvements for existing ones.

Since the area of client side functionality in web applications is a rather new area there are probably not any frameworks in the area that has reached a stage five according to the model. But the frameworks are investigated to see what states they are currently in and how maturity could affect a project using them.

Angular

Angular reached its first release 1.0 June 14th, 2012 with much smaller functionality than what the framework offers today, the main focus then was data bindings, templates and testability. Since then a lot has been added, and Angular is currently at version 1.1.5 (stable) which was released in May 2013 and 1.2.0-rc1 (release candidate 1). The latest blog post from the angular blog states some information about release 1.2.0:

> “AngularJS 1.2.0rc1 spooky-giraffe adds a whole lot of new features that we’ve been working on in the 1.1.x releases: security improvements, a complete rewrite of transitions/animations support, better error messages (including error message minification), compiler additions, and better support for mobile/touchscreen devices.”
> - August 14th, Naomi Black [20]

This clearly states that the improvements being worked on in the development of Angular are adding more functionality and main components of the framework are being changed. So according to the CMM defined previously then Angular is at stage 3, where parts of the framework are already defined and continue to be improved, but the core parts are still being changed and more content is added to the framework.

Knockout

Knockout is currently at version 2.3.0 (Stable), which was released July 8th, 2013. Knockout has been at a stable state for over two years since the first release and from the last year’s commit and update messages on their repository it shows that almost all changes that are currently made to Knockout are minor changes, many of them provoked by pull requests from the community. There are also a few messages about optimizations. So this shows that Knockout is at stage 4 of the CMM, where core components of the framework are unlikely to undergo any major changes and it should be possible to use the current version without having to adapt too much to coming changes in the framework. The fact that the framework has been released and used for so long also speaks for that the overall stability of its components should be good since the framework have been tested and used in many different cases.

Ember

Ember is a framework that started being developed in April 2011, it has since been developed and used in many web applications before its first official stable release August 31th, 2013. Ember is just as Angular a more complete framework than Knockout and are thereby also trying to include more areas and more functionality into the framework. As expressed by the Ember team, they are currently mostly trying to improve documentation of the framework and simplify the process of getting started with ember to make it more accessible for new developers. Ember is a bit between stage 3 and stage 4 in the CMM, where the development team currently is working on stage 4 of improving the product by adding documentation and API improvements. But at the same time more functionality is added to the framework so even though the core components should not have any major changes the framework has not left stage 3 of the CMM.

2.2.3 Performance (caching)

This section does not discuss performance as in speed of computation since the operational performance is tested later on in Chapter 4. This section discusses what the different frameworks do to improve the performance of the application, and what improvements they contain that might provide improved user experience through less loading times and less communication between server and client. After all, performance in an application is important since it makes the application more efficient and more fun to use. Good examples of this are shown in a Microsoft research presentation about the public DNS system and global traffic management. The statistics in the presentation shows that Amazon.com had 1% sales decrease from an additional 100ms latency, and 500 ms delay of google.com caused 20% decrease in traffic [15].

In web applications one of the largest problems is speed in the communication between client and server, so other than improving the performance between the sides separately developers try to reduce the amount of data sent between server and client as much as possible. This can for example be done by caching data. Caching improves responsiveness in the application by not repeating calls, which in turn improves the user experience. This section discusses how caching is done by the different frameworks and check whether the frameworks include any other improvements that might provide similar effects.

Angular

Angular has many community driven hotfixes for performance problems, but does not have any recommended adaptions for when performance is an issue.

For caching, Angular uses a service that they call $cacheFactory. For the developer this component abstracts the problem of implementing a cache to just using some simple functions to achieve the needed caching functionality. $cacheFactory has a default setting that helps the developer add caching to data requests by just setting the "cache" parameter to true for requests. The developer can also add other cached values through a simple key-value operation. So far there is not any functionality for setting timeout for cached values, if e.g. the value that is cached is changed regularly each day, hour etc. So the $cacheFactory adds some functionality that can be used to improve the performancelot, but it is not complete (it could and probably will have some additional functionality added to it).

Knockout

Knockout does not contain any special additions to improve performance, but just relies on its solid base structure. There is no standard for caching defined for Knockout.

Ember

Ember uses a clever change queue when variables or parameters are changed which makes computations asynchronous. The main idea is that when a property is changed this invalidates the current state of the property and queues the actual change to occur later. If another property of the same object is changed before the actual change then the changes coalesce and there is still only one recomputation of the object. The goal with this way of changing values is to improve the user experience, since the user does not get interrupted by the back-end processing. However this leads to some of the common problems with asynchronous operations. Since a developer cannot rely on the synchronous side-effects, the operations that follow on changed values need to be implemented with callbacks and observers.

Ember uses an abstracted module called *store* for all data storage. This module implements an automatic cache for values, so that if you model the data in the store it will be cached automatically. The benefits of this is that the developers do not have to worry about how the values are cached, but only need to manage data in the store and caching is done by Ember. This method has the same problems as the angular cache though with not being able to define different rules for different types of values.

2.2.4 Portability

The users of the application will use different web browsers to use the application. They will use computers with different performances and different settings. This section investigates what the issues with compatibility are for the different frameworks and what the support the frameworks have to handle compatibility issues.

Within portability it is also worth mentioning whether the framework provide some support for using the application on mobile devices. This is not a very important factor for the project since the kind of application discussed is rarely used on mobile devices, but is worth looking into since the demand for web applications being available for mobile devices is growing.

Angular

Angular is 100% JavaScript so it should be compatible with all browsers, on both desktop and mobile devices with JavaScript support. However it needs to import extra files and add some fixes to the pages in the application to be able to support Internet Explorer 8.0 or earlier.

For mobile/touchscreen support Angular is currently adding more support and improved UI support in the coming release (1.2.0). [21]

Knockout

Each release of Knockout is tested by the developers on:

- Mozilla Firefox 2.0+ (latest version tested = 3.6.8)
- Google Chrome (tested on version 5 for Windows and Mac; should work on older versions as well)
- Microsoft Internet Explorer 6, 7, 8, 9, 10
- Apple Safari (tested on Windows Safari version 5, Mac OS X Safari version 3.1.2, and iPhone Safari for iOS 4; should work on newer/older versions too)
- Opera 10 for Windows

and according to the development team it is very likely that it works on older and newer versions of these browsers as well, but they do not guarantee it. [37]

Knockout does not handle the UI components themselves so there are no additions made to support mobile devices or input. But the data bindings and management should be usable on all browsers with JavaScript support.

Ember

As the Ember development team describe it themselves they have support for all available browsers, with some minor issues (not further described) on some of the older browsers [34]. However the Ember development team does not recommend using the older browsers since the performance of older browsers are not the same as for the newer ones, which have a significant risk of causing performance issues when using a heavy framework.

The same goes for mobile devices where the Ember developers describe claim to support all browsers using JavaScript, but again even though the functionality is there, it should be carefully used for older devices since they might lack performance.

2.2.5 Testability

When discussing testing in web application development, there is often a split done into two areas of testing. These are

- Unit testing
- Performance testing

This section covers unit testing since it is an important aspect in the development process. The area of performance testing is covered later in chapter 4.

Unit testing is an important part for a project to be long-lived. The main idea is that each part of the project that contains functionality that is possible to test individually is considered a "unit". Test code should be written to test all units separately. The individual tests are then used to ensure that the functionality of the individual units is not affected by changes in the project.

Angular

Angular is a framework that from the beginning was built with unit testing in mind and it includes many features to help writing tests. One of the main features is the Angular Scenario Runner, which enables the developer to write user scenarios in the testing environment to not only be able to unit test the separate bits of code but also write tests for the navigational functionality. Angular does not however provide any tools for running the tests but recommends using a JavaScript testing framework such as Jasmine or Karma for this.

Knockout

Knockout supports all of the main JavaScript testing environments and libraries, but does not have any standard or recommendations for how unit testing should be done.

Ember

Ember does not have any standard defined for how testing should be done, but relies on the standard unit testing frameworks for JavaScript.

2.2.6 Modularity

The goal of the project is to define a structure to base applications on, so modularity in the framework is an important factor. A structure within an application is very often dependent of having separations between where to place certain types of functionality. An example of this is to keep all connection and data transfer in a separate file, so that is can be used from anywhere in the project. This also makes it easier to change parts of the project since separate components can be changed without having to affect other components much. Often having this kind of modularity can help making the structure of the project easier to understand and thereby make the application easier to develop and maintain. Really important for modularity is being able to keep the modules defined, separated and small.

Angular

The main structure of Angular is shaped based on the MVC pattern, even though Angular often describes it as a (Model-View-Whatever). This means that the code is divided into Models, Views and Controllers. The models consists of application data, the controllers consists of business logic and functions, and the views consists of all visualization of the data in the application. The controller's job is to construct the model and publish it into the view along with callback methods so that input from the view can be updated to the model. Other than the MVC structure the code is divided into different files separating views and controllers for the different pages. Generally, each page in the application consists of a separate view and controller that interacts with the model that is common for the application.

Knockout

Knockout uses a pattern called MVVM (Model-View-ViewModel). This pattern differs from MVC by having a more minimal view. In the MVVM pattern the view contains a minimal amount of functionality and does all operations by calling directly to the ViewModel, which performs updates to the Model. So all content changes and all data management is done between the ViewModel and the Model, and the view is just generated to create a visualization of what the ViewModel contains. Knockout has support for applying separate ViewModels to parts of pages.This means that for large pages more than one ViewModel can be applied to be able to separate different parts that manage different data from each other.

Ember

Ember is also a framework based on the MVC pattern, so it has the same benefits from this as Angular. The model in Ember is however more strictly defined since all data management is done through the "store". So the modularity of Ember is about the same as for Angular

2.2.7 Popularity

Since web client development currently is a very community driven area it is important what the community currently is interested in. There are many factors where it is beneficial to have a big community that use the framework, some of which are:

- More bugs and problems are discovered and fixed.
- If there is a problem more people will be engaged in fixing it.
- It is more likely that the framework will be long lived and remain up to date if there are more people using it.

The popularity of the frameworks is measured in a number of units. These are:

- **Google searches**
  Since Google currently is the worlds biggest search engine this shows a result of how much people actually are interested in the frameworks. In addition to this many developers use google to search for solutions to problems, which also improves this measure since it also weighs in current developers. When listing this the discussion also takes into account where in the world the searches are focused.

- **Stackoverflow questions and followers**
  Stackoverflow is one of the worlds biggest communities for developers, where many questions are asked and answered each day. The amount of questions asked and the number of stack overflow tag followers for the different frameworks are measurements of how big the developer community interest is.

- **Subscribers to the frameworks repositories**
  People in the community that work on developing frameworks often subscribe to the framework's repository to get the latest release and development updates as well as to be able to add in own changes to a framework. Therefore it is relevant how many have subscribed to the repositories, it is a measurement of how many that are interested and involved in the development of the frameworks.

Google searches

Figure 2.1 shows a table of Google searches over time between Jun 2010 and Dec 2013
Angular has its searches mostly focused in India but otherwise spread quite globally.

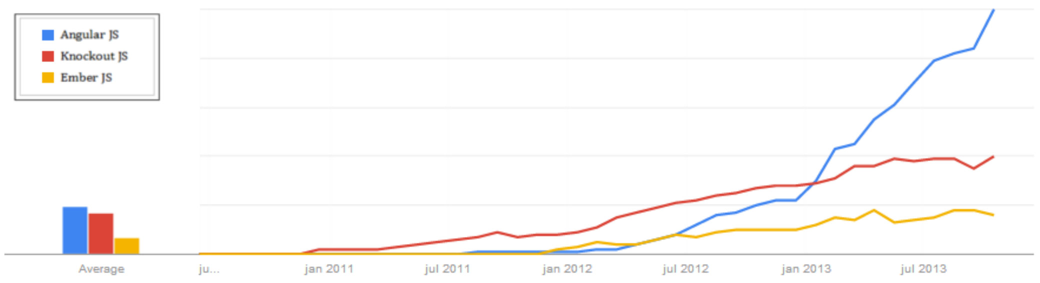

Figure 2.1. Google searches for framework names Jun 2010 - Dec 2013

Knockout also has a quite wide spread of its searches over the world, with the main focus in India.
Ember has not got a very big spread globally but is only focused on USA.

Other popularity measurements

Figure 2.1 shows how the different frameworks measure in popularity over some sites that are commonly used in the web client development community.

Framework	Stackoverflow tag	Stackoverflow questions	Stackoverflow followers	Repository subscribers
Angular	angularjs	15,619	4,366	1,679
Knockout	knockout.js	7,838	1,980	749
Ember	ember.js	6,319	1,286	367

Table 2.1. Measurements of framework popularity

[58] [60] [59] [22] [37] [33]

2.3 Sample Application

A sample application is defined and implemented, this section contains a description of the application and some additional more practical factors used to compare the frameworks. The purpose of the sample application is to represent the behavior of a larger system for case management in a smaller application, so that the application can be implemented as a way of comparing and performance testing different frameworks. The application contains the main functionality of the bigger systems having the same dependencies on a framework, but in a smaller scale.

2.3.1 Application description

The application that is implemented to evaluate the frameworks further is a typical application for case management i.e., handling persons, cases and documents. Generally the views of the application can be divided into object views and search views, where both kinds exist for the management of persons and cases. So in more depth the views are:

- **Create person**
  In this view the user can create a new person to the system by filling in required fields with person data.
- **Create case**
  In this view the user can create a new case to the system by filling in required fields with case data.
- **View Person**
  Shows a person object and its properties. Also shows what cases the person is involved in and what role the person has in said cases. From this view the user can also edit the properties of the person. This view also allows the user to add cases, change roles in cases, remove cases etc.

- **View Case**
  Shows a case object and its properties. Also shows what persons are involved in the case, and what roles said persons have. Just as in the person-view, the user can in this view edit the properties shown, add/remove persons etc. In this view there is also a list of documents that are related to the case that can be edited by adding, removing or changing documents.
- **Search for Person**
  A view where the user can search for a person and through the found results enter its view person page.
- **Search for Case**
  A view where the user can search for a case and through the found results enter its view case page.

The application also has a set of rules that are there to handle special cases since this is often the case in the real applications. The rules that need to be handled are:

- Persons can be individuals or legal persons having different properties depending on which.
- Search results have to be divided up into pages.
- The application remembers last searches.
- Search results can be sorted in different orders by clicking the table headers that they are presented with.
- Navigation between the different views should behave “as expected” (back button, forwards button, links etc.)
- Input fields should be validated.
- A person can only have one role in each case.
- The state of a case can only be completed if a deadline is set and a case’s state cannot go directly from new to completed.

Some of the most important operations that the frameworks need to handle efficiently are forms, data validation and navigation. What follows is a summary of what the frameworks do within these areas and thereby, how well they manage some of the tasks in the sample application.

2.3.2 Form management

To interact with the application the user uses forms, so this is one of the most basic operations of a web application managing data. Since forms are so important, it is important for the framework to be able to manage forms efficiently. This section discusses whether the frameworks contain any functionality that improves form management further than the standard HTML form. Another thing that is discussed is wether the frameworks do anything to simplify the implementation of forms for the developer.

Angular

In angular forms are only needed to be defined with their fields and properties in the DOM using data bindings to a controller. Then the fields and data variables are created automatically in the controller when the fields are filled in. This makes it really easy to define a form since the fields do not need to be named and created in both the view and the controller simplifying changes and providing a low chance of errors since very little code is actually needed.

Knockout

The Knockout data-structure in itself supports creating forms dynamically using knockout observables but other than that Knockout does not include any functionality that extends forms.

Ember

To simplify form management Ember implements a functionality that they call Event Delegation, which allows the developer to reuse event handlers for elements of a form that need similar functionality. Other than that Ember does not include anything that extends the standard way of implementing forms.

2.3.3 Data validation

When showing a form for the user, it is important that the user's input is correct so that the data works with the rest of the application. To ensure this the data must be validated which can be done in various ways. This section covers how the frameworks support data validation in forms, what defined validation templates are included and how easily the framework can be extended with additional form validation defining requirements that are individual for the application at hand.

Angular

Angular has its own system for validating input data which has many standard patterns such as email and url validations predefined. So this makes it easy for the developer since their predefined validations can be used for most of the inputs. If the developer wants to extend the validation with customized values this can be done by either adding a regular expression which is used to validate the input, or by defining a validation function of their own and use it.

Knockout

Knockout does not handle data validation, however they recommend the use of their extend tag to include JQuery validation to the knockout data model. This also allows the developer to add new validation methods for the application.

Ember

Ember does not contain any data validation functionality, neither do they define any recommended way to solve this. In the Ember community it seems like the best approach is to use JQuery validation.

2.3.4 Navigation

Web browsers implement and have for long implemented a behavior of navigation between different pages. For the users this has also added the functionality of being able to navigate forwards and backwards between previously visited pages. This is very important for the user experience since it is something that the users take for granted. This section discusses what the frameworks include to implement the standard web browser behavior, since this is not a trivial task in SPAs.

Angular

Angular implements a service called $location together with config $routeProvider to manage browser navigation. The service is based on the standard (window.location) which makes the browser URL available in the application. Instead of just using the standard raw object properties of the window.location Angular implements getters and setters for the URL. This simplifies implementing navigation by allowing the developer to use different getters and setters for page context and document root.

Knockout

Knockout does not manage navigation at all, but recommends the developer to use an external library called Sammy.js for this. Sammy.js allows the developer to define functions that restore the state of the application using url values, so the developer can determine to what depth urls should determine the state of the application and what should be restored when loading a page.

Ember

Ember contains a component called "Router" which contain functionality for all requested navigational behavior. The routing is implemented through defining a binding between view templates and URL paths so that they are directly linked in the application. This means that a user can both save a URL or type in a URL in the browser to get to the correct page of the application. Since the links are directly between templates and the browsers URL also means that the standard browser navigation features such as forward and backward buttons are implemented automatically through the definition of the linkings.

2.4 Conclusions

This section contains a summarization of the theoretical comparison of the frameworks made in this chapter. Following is a discussion of which frameworks will be used to implement the sample application and tested in practice.

2.4.1 Maintainability

Ember and Knockout both support templating for view elements which allows the reuse of HTML code. The difference between their ways of defining templates is because of their underlying structures, which leads to that knockout allows the use of the same template applying different functionality to it. However, this functionality is an example of entangling views making the code difficult to understand so should not be used in applications trying to achieve good maintainability. Angular did not support these kind of templates.

In extendibility Knockout is clearly the best of the three, this since both Angular and Ember are aiming to be complete frameworks, in the essence that they should not need to be extended with other functionality. An application using knockout is easily extended since knockout only manages the data bindings of the application. This leaves room for the developer to add additional components for all other parts of the application.

In maintainability all of the frameworks should be able to be used to create maintainable applications as long as they are used the way they are meant to. However, the best one in this category is probably Knockout since it only does its part of the application and other parts could be changed or extended independent of the framework, without having to change any basic structure of the application.

2.4.2 Maturity / Stability

Even though all of the frameworks have reached releases that are considered stable the obvious answer to which framework is considered being the most mature is Knockout. Knockout does only manage its own part of the application and therefore it is not extended with additional components which might decrease the stability of the framework. Angular and Ember does not differ very much in stability, both of them cover a lot of areas and both of them are still being extended with additional functionality.

2.4.3 Performance (Caching)

Angular and Ember both implement caching functionality, where Angular has a simple cache that can be used by the developer and Ember implements an automatic caching system. Since the application that is relevant in this project is one managing large amounts of data automatic caching could be risky since it might mean that the wrong data gets cached. So even though Ember is probably the most efficient for a developer the cost of this is by a loss of control over the data. The result of this is that Angular probably is the best framework for caching, with the main reason that it allows the developer to specify how to and when to cache data.

2.4.4 Portability

All of the frameworks have about the same backwards compatibility, with the minor exception to Angular not being compatible with internet explorer versions < 8.0. Other than that all of the frameworks support most devices that can manage JavaScript, so there are no big differences here. In conclusion there is no best framework in this category.

2.4.5 Testability

Angular is definitely the winner in this category since it was built with testing in mind. All of the frameworks support testing in similar ways, by using the more general JavaScript testing frameworks, however Angular supports it further by defining test methodology as well as test examples for the different components in the framework.

2.4.6 Modularity

Angular and Ember are both based on the MVC pattern while Knockout is based on the MVVM pattern. Comparing Ember and Angular Ember has minor advantage since the framework abstracts the data model from the rest though the *store* component which takes care of the data management. In Angular the data model layer has no predetermined structure. Comparing Ember to Knockout then, boils down to the comparison between MVC and MVVM.

They both have their advantages and disadvantages, mainly since MVC splits up the functionality to the view and the controller, keeping the modules smaller than as in the MVVM pattern contain all logic in the ViewModel. But at the same time The ViewModel avoids code duplication that might occur from the split into view and controller.

The conclusion then is that all of the frameworks attempt to achieve a modular structure in different ways, with no definite best option. The modularity is therefore more relevant in the implementation when seeing whether they support keeping the code structured into modular components.

2.4.7 Popularity

Knockout and Ember have about the same increase in popularity, but since Knockout has been released for so much longer it is currently more popular. Also worth mentioning is that Ember is only increasing in popularity in USA, which is a major disadvantage when working from Europe. Angular started becoming popular around its stable release in 2012 and has grown in popularity ever since. In the beginning of 2013 it passed Knockout in the amount of google searches and has more than doubled since.

When looking at the data that is a measure of developer interest and not general interest Angular also has a major advantage. It is by far the framework that have the biggest developer community.

2.4.8 Form management

All of the frameworks are able to bind variables and their fields to forms, making the development more efficient and removing unnecessary code from the application. There is no clear ordering then in how good the frameworks manage forms since they all have about the same functionality.

2.4.9 Data validation

Data validation is defined as a part of Angular, Knockout implements the extend tag which is made for this purpose and Ember does not include this functionality. The end result of this is that Angular and Knockout achieve about the same results with similarly looking code, so considering Knockout extended with JQuery this makes Knockout and Angular too similar in this aspect to be able to determine one that is the best.

2.4.10 Navigation

Angular treats this as a core component of an application since it is something that is wanted in all applications. Ember manages this functionality almost exactly as Angular does it, and Knockout using Sammy.js also does this in the same way, except for that Sammy.js allows the developer to write functions for restoring states of a application. So this category is also very similar between the different frameworks and there is no definite answer to which one is the best.

2.5 Summary

From the discussion in this chapter and as can bee seen in figure 2.2 it is clear that the frameworks differ in many ways. However, Angular and Ember are quite similar with Angular having some minor advantages in terms of stability and popularity which are derived from Angular having been at a stable state for a longer time. The fact that Angular also is backed up by Google Inc. is a determining factor between the two since being backed up by one of the biggest companies in the world makes the framework more likely to be long lived. So with this in mind one would probably get about the same behavior from an application implemented using Angular as for an application using Ember, but the Ember application is not as likely to be long lived.Thus, Ember will not be implemented in this project.

Category	**Angular**	**Knockout**	**Ember**
Maintainability	-	X	-
Maturity	-	X	-
Performance	X		-
Portability	-	-	-
Testability	X	-	-
Modularity	-	-	-
Popularity	X	-	-
Form management	-	-	-
Data validation	-	-	
Navigation	-	-	-

Figure 2.2. Summary of the theoretical comparison in chapter 2. (X) being considered the best in a category, (-) being covered but not best, ( ) not covered in the framework

There is a big difference between Knockout and Angular, mostly since Angular covers a larger area of functionality but also in how they do things differently in data bindings and data management. Furthermore these two frameworks will be compared practically through implementation and performance testing.

Concludingly, the frameworks that are compared further are:

- Angular JS
- Knockout JS

Chapter 3

Implementation

Chapter 3 discusses the implementation of the sample application, how the frameworks are to work with in practice and theoretical aspects that can not be discussed without actually working with the frameworks. The chapter also contains some notes on whether there are any differences between how the frameworks are described in chapter 2 and how they are to work with in practice.

3.1 Learning threshold

The learning threshold of a framework can be a vital factor when deciding on a framework since it determines what the cost would be to get developers ready to actually start working on a project and getting it right from the beginning so that components do not need to be redone because of lack of understanding. The learning threshold can also be valuable since it shows how easy the structure is to understand later on when maintaining the project. The DRY concept as discussed earlier in the section 2.2.1 is also difficult to apply to a project unless the understanding of the project structure is at a high level.

3.1.1 Angular

The code structure used in Angular has very strong separations between the different components, and how the components are supposed to interact with each other. The result of this is that even though the learning threshold of the framework as a whole is quite high, it is easy to get the structure right from the beginning and to use the components correctly. As soon as one understands the basics of the framework then it is quite simple to start by using the components understood and when a task is encountered that does not fit the already understood components there are often solutions described online to what component to learn next.

Angular has a component called $scope which is generated from both the controller and the view together building a viewmodel alike layer. All data bindings are done automatically by creating variables in this $scope component, which makes all data operations act as if they were any variables. This way of managing the data bindings layer helps the developer to develop without any major differences from non bound variables. This is a summary of what needs to be understood to get started with Angular.

The learning threshold required to begin working with Angular then is quite low but to learn the whole framework and where to use which component takes a lot of time.

3.1.2 Knockout

Knockout is a very small framework so it is possible to achieve a basic understanding of the whole framework in quite short time. Since Knockout uses the standard HTML attributes this made it even easier with some previous knowledge about HTML.

With the learning threshold for Knockout being so that it is very easy to get started there can still be some problems in the data bindings. Knockout uses something called Knockout *observables* to manage data bindings between, these are implemented as JavaScript functions which is a bit confusing sometimes. This is something that caused some problems in the implementation since the variables can be accessed as both functions and values in different places of the code without a clear separation on where to use which syntax.

3.2 Documentation

Documentation is something that is very important to be able to get started with and work with a framework. This section compares the documentations of the frameworks, how they are structured and how they are to work with.

3.2.1 Angular

The documentation of Angular consists of a short overview of what components exist in the framework and what they do written in a few words. There is also a list with all of the components that link to pages showing details for the components, how they can be used and some short code examples of how the component is included in an application. Other than that the documentation is mostly just an API documentation where attributes are defined and described.

The documentation for Angular is focused on a target group of experienced developers, giving the tools needed to include Angular in an application. The documentation does not however suggest any ways to get started with Angular or contain any more simple guides for people with less previous knowledge of web development.

3.2.2 Knockout

The documentation for Knockout contains about the same kind of information as the Angular documentation, with an overview with short explanations and the more specific API references. Knockout also has a section in the documentation called *Live Examples* that show complete code examples of common uses for the Knockout framework and guides of how to, from simple hello world examples to examples of receiving JSON data into a model to store and manage collections. The Knockout documentation also contains a tutorials section where new developers have an easy way of getting started with Knockout and learning to use Knockout step by step, with assistance and explanations for each step.

The documentation for Knockout is aimed at a broader target group than Angular, with more support for new developers while still maintaining a good technical documentation for the more experienced developers.

3.3 Code comparison

This section contains some examples of how code looks like in the two frameworks, giving a simple example taken from the sample application. The case that is shown is the implementation of searching for a case and listing the search results. How the operation is implemented is explained step by step, from creating the search parameters to defining click events on the search results shown. The section also contains some explaination and discussion about the differences that showed between the frameworks when the application was implemented.

3.3.1 Data bindings

Data bindings in the frameworks are implemented using different techniques. The data bindings are also used in different ways. This section shows a comparison of how the search function is implemented using the different data bindings and some discussion about advantages and disadvantages with the differences in data-binding implementation. The code below contains the functionality for searching for cases in the sample application. Where at first a JSON object with search parameters is defined, values are entered through the DOM bindings and then the parameters are sent using an Ajax call that returns the search results.

Angular

The data bindings in angular are implemented using dirty-checking which means that values are updated by comparing the value of variables with previous value, if there is a change an event is fired to update all references to the value. Using dirty-checking is what allows Angular to use the standard javascript objects and variables as bindings instead of having to as Knockout use their own types to implement the bindings.

The problem with dirty-checking occurs if the amount of bindings are many, since all data values need to be checked to see if they are dirty or not to update one binding. This operation can take a lot of memory and processor power. So there is a risk for Angular to use very much memory and lack performance in an application with many bindings.

```
<form novalidate class="simple-form">
    Diarienummer
    <input type="text" data-ng-model="search.Diarienummer" /
       >
    AerendeTyp
    <select data-ng-model="search.AerendeTyp" data-ng-
       options="AerendeTyp.aerendeTypNamn for AerendeTyp in
       AerendeTyperMedAlla" />
    Aerendemening
    <input type="text" data-ng-model="search.AerendeMening"
       />
    <button data-ng-click="searchAerenden(search)">
        Soek
    </button>
</form>
```

Figure 3.1. Case search HTML implementation using Angular

In the code example in Figure 3.1 a form is defined containing search parameters for a case-search.

The search parameters are defined as fields in an object called search, the different parameters are: *Diarienummer*, *AerendeTyp* and *AerendeMening*. *Diarienummer* and *AerendeMening* are selected using text fields and *AerendeTyp* is selected using a combobox.

To bind a variable or a field to a UI element Angular uses the *data-ng-model* tag. This is an example of how Angular extends the DOM with additional attributes for the Angular components, as can be seen in the two *input* tags and the *select* tag. In the select tag there is an additional extension to the DOM using the *data-ng-options* tag to define what are the possible options to select from in the combobox. The syntax used in the *data-ng-options* tag is:
<field of object> for <object> in <collection of objects>
so in the figure 3.1 the combobox contains the field of name *aerendeTypNamn* in all objects in the array *AerendeTyperMedAlla*. Binding to and displaying collections or arrays is explained further in section: 3.3.2.

To send the form and thereby execute the search query another attribute is used, *data-ng-click* is a binding that takes a function as parameter and runs the function when the object is clicked. So in this case the function *searchAerenden* will be executed when the button is clicked, passing along the search object as parameter to the function.

```
$scope.searchAerenden= function (params) {
    $scope.SearchResults = [];
    AerendeFactory.getAerendeList(params).success(function (
        data) {
         angular.forEach(data, function (aerende) {
             $scope.SearchResults.push(aerende);
         })
    });
}
```

Figure 3.2. Case search JavaScript implementation using Angular

To be accessible to the view the function searchAerenden (Figure 3.2) is defined in the *$scope* object. When called the function initiates *SearchResults* as an empty array, or if the array already exists empties it to be able to fill it with the new results. After that the function calls a function called *getAerendeList* in *AerendeFactory* passing along the search parameters. When *getAerendeList* returns, a list of the found *aerende* objects are added to the *SearchResults* array by iterating over all returned objects and adding them.

Knockout

Knockout uses change listeners to create data bindings. This means that the bindings are implemented as a pub/sub system where all bindings publish and subscribe to change events on the data. When the variable is changed this triggers an event that notifies all subscribers so that they can update the binding to show the new value. The difference between this and the dirty-checking method is that the change listeners does not need to keep track of values that are not changed, but only needs to keep track of the publishers and subscribers. Therefore this also means that change listeners does not take up any performance when no values are changed and are independent on how many variables are bound in the same page.

Even though change tracking is more performance efficient there are a few drawbacks. One common example is when adding elements iteratively to an array, then the array fires a change event for each element that is added to the collection. The same kind of behaviour can be achieved when a variable is changed multiple times, this will trigger multiple change events possibly before the value has been updated to all of its subscribers, which in its turn might cause problems. Knockout manages these cases with no problems detected in the implementation of the sample application as well as in the testing, but this means that the code to manage the bindings has to be very complex. One major problem with change tracking is that the variables cannot use the standard syntax but needs to be implemented differently to make sure to trigger the change events. In Knockout this is done by implementing the variables as functions called Knockout *observables*.

```
self.search = {
    Diarienummer: ko.observable(),
    AerendeTypNamn: ko.observable(),
    Aerendemening: ko.observable()
};
```

Figure 3.3. Initialization of the search parameters in knockout

To implement the case-search in Knockout the first step is to define the search parameters in the viewmodel and make them available to the view, this is done using the code in Figure 3.3. "self" is defined as the viewmodel object, where all functions and variables that need to be accessed by the view are defined. The parameters are defined as empty Knockout *observables*, which is how bindings are implemented in Knockout.

```
<form data-bind="submit: searchAerenden">
    <label>Diarienummer</label>
    <input type="text" data-bind="value: search.Diarienummer
        " />
    <label>Aerendetyp</label>
    <select data-bind=" foreach: AerendeTyperMedAlla, value:
         search.AerendeTypNamn">
        <option data-bind="text: $data.AerendeTypNamn, value
            : $data.AerendeTypNamn"></option>
    </select>
    <label>Aerendemening</label>
    <input type="text" data-bind="value: search.
        Aerendemening" />
    <button type="submit">soek</button>
```

Figure 3.4. Case search HTML implementation using Knockout

After the search parameters are defined in the viewmodel the values of the parameters are bound to the view using the code in Figure 3.4. The search parameters are just as in the Angular example *Diarienummer*, *AerendeTyp* and *AerendeMening* and using the same kind of input types. The input tags are quite similar, but Knockout does not extend the DOM with new attributes but uses the *data-bind* tag for all data bindings and binding operations. So in the text input fields Knockout binds the view to the viewmodel by *data-bind="value: <observable>"* which sets up the bindings to the *value* attribute, the same is done for the value of the combobox.

To populate the combobox in Knockout the *foreach* binding is used, which repeats the child DOM elements for each object in the given array, so as defined above the *<option>* tag is repeated for each object in the array *AerendeTyperMedAlla*, and the value and text of the option is given by the field *AerendeTypNamn* in each object.

The button that executes the search in Knockout only calls the function *searchAerenden* and does not pass any parameters to the function. The parameters are not needed since the values of the bound values already are defined in the viewmodel and accessible from the function that executes the search.

```
self.searchAerenden = function () {
    self.SearchResults(null);
    datacontext.getAerendeList(self.SearchResults, self.
        error, self.search);
};
```

Figure 3.5. Case search JavaScript implementation using Knockout

The function *searchAerenden* is defined in the viewmodel object (*"self."* in the context of figure 3.5) and starts by emptying the SearchResults Array, to make sure that the array is empty. Then the function calls another function that gets the data requested passing along the search parameters, the array object to return the results in and an observable to return possible error messages. When the function returns the *SearchResults* array will be populated with the search results, or the error message passed on to the function will be set with a non empty error message to be able to notify the user that something went wrong.

3.3.2 Collections

The data bindings are also managed in different ways when they are applied to collections. This section continues on the previous example by adding the search results that were gotten from the Ajax call to a collection and using iterative bindings to display the returned results in a table.

Angular

Angular does not use any special implementation for collections but builds their functionality around the standard JavaScript array.

The main feature that Angular contains to manage collections is called filters. A filter is used to limit a collection to a subcollection using some kind of filtering function. The function is applied to the collection which makes the collections automatically update when content of the filtering function is changed. Filters can also be used to change parts of the collection since when applied to a collection Angular applies the function to all elements. Then the function can both change elements and determine whether they should be displayed or not. Angular has some predefined filters for common operations and lets the developer define own filters if the preexisting ones are not enough.

```
angular.forEach(data, function (aerende) {
    $scope.SearchResults.push(aerende);
})
```

Figure 3.6. Storing search results using Angular

Figure 3.6 continues on the example in Figure 3.1 and Figure 3.2 searching for a case. The search results are returned in an array *data*. Using the angular *forEach* too loop through each element in the array data and adding the elements to the search results array. This is just showing that angular uses the standard JavaScript array for storing the collection but extends with functionality such as *foreach* loops.

```
<table data-ng-show="SearchResults.length > 0">
    <thead>
        <tr>
            <th>Diarienummer</th>
            <th>Aerendetyp</th>
            <th>Aerendemening</th>
        </tr>
    </thead>
    <tbody>
        <tr data-ng-repeat="aerende in SearchResults" data-
           ng-click="selectAerende(aerende)">
            <td>{{aerende.diarienummer}}</td>
            <td>{{aerende.aerendeTypNamn}}</td>
            <td>{{aerende.aerendemening}}</td>
        </tr>
    </tbody>
</table>
```

Figure 3.7. Showing search results using Angular

Figure 3.7 shows a HTML table that displays the search results. The first tag *data-ng-show* is used to determine the visibility of the table, with the parameter setting the value as true if the search results contain any element. So the table is hidden if the results list is empty.

After this the code declares the headers of the table as *Diarienummer*, *Aerendetyp* and *Aerendemening*. And the body to show the content of the search results. The body consists of a definition of a table row that contains the *data-ng-repeat* tag, which repeats the table row for each element in the *SearchResults* array. The *data-ng-repeat* tag syntax declares an object *aerende* for child elements which enables the {{}} binding to elements in the object, so {{aerende.diarienummer}} binds to the field *diarienummer* in the object aerende. The *data-ng-click* binding is used to select a certain element that should be marked as selected. Showing which element is selected is done using css and is not covered in this report.

Knockout

In knockout collections are managed using something called Knockout *observableArray*. *observableArray* responds on updates and changes on a collection of objects instead of as the *obsertvable* just watching one object. *observableArray*s can have custom functions attached to them extending them to have filters or other additional functionalities, but as default they support the most common collection functionalities, and can be bound to iteratively.

```
self.SearchResults = ko.observableArray();
ko.utils.arrayForEach(data, function (aerende) {
    aerendeObservable.push(new createAerendeList(aerende));
});
```

Figure 3.8. Storing search results using Knockout

Figure 3.8 continues on the example in Figure 3.3 to Figure 3.5 searching for a case. This function has just received a list of objects in the array *data*. To show how an *observableArray* is initialized the function initializes the *SearchResults* object as a new *observableArray*. Then the function runs a foreach loop through all elements in the list *data*, creates a new object for each element using the function *createAerendeList* and adds the object to the *SearchResults* array.

Figure 3.9 shows how to display the search results in a table using HTML to bind to the elements in the collection. The results are shown in a similar table to the Angular implementation. The first part of the code uses the *data-bind* tag to bind to the standard HTML *visible* attribute, setting it to true if the length of the array containing the results is greater than zero, resulting in that the table is hidden if there are no elements to display and shown if there are. After this the table headers are defined as *Diarienummer*, *Aerendetyp* and *Aerendemening*. Knockout uses a binding similar to Angular's *data-ng-repeat* that is called *foreach*, which is bound to a HTML element and repeats child elements instead of as the Angular binding does it, repeat the object itself and its child elements. The *data-bind="foreach: SearchResults* repeats the table row for each element in the array *SearchResults*. In the table row a click event is declared for the row using the *data-bind="click: "* attribute which calls the function to select a table row. Inside the *data-bind="foreach: "* tag the current element can be accessed as *$data*. However, since the *$data* is very commonly accessed in a foreach binding Knockout also

```
<table data-bind="visible: SearchResults().length > 0>
    <thead>
        <tr>
            <th>Diarienummer</th>
            <th>Aerendetyp</th>
            <th>Aerendemening</th>
        </tr>
    </thead>
    <tbody data-bind="foreach: SearchResults">
        <tr data-bind="click: $root.goToObjectPage">
            <td data-bind="text: Diarienummer"/>
            <td data-bind="text: AerendeTypNamn"/>
            <td data-bind="text: Aerendemening"/>
        </tr>
    </tbody>
</table>
```

Figure 3.9. Showing search results using Knockout

interprets values without any parent object as in the current *$data* element. So for each table column in the row the text of the column is bound to the element using the *data-bind="text: "* binding and the values are given by the previously mentioned search results fields.

3.4 Conclusions

This section contains some concluding words about how the frameworks are to work with.

Angular

Angular is a bit difficult getting started with since the documentation is not very great for beginners. However, when the main components in Angular are understood, the framework really shows its potential and it is easy to extend the application using the components. There were never any greater uncertainties or confusions during the implementation of the sample application since data is managed as JSON objects and Angular does all of the bindings for the developer. It is a bit confusing at first that Angular allows the developer to change what HTML attributes are possible, and assuming that a developer knows HTML but not Angular it might be difficult to understand the code, but this is one of the parts that were really well explained in the documentation so it is not a big problem. The new HTML attributes could become a problem in projects with many developers if the communication between the developers is not done properly.

Knockout

Knockout is simple to get started with using the Knockout tutorials. However, some problems did occur from different sources giving different examples of how the Knockout *observables* should be used. Other than this the implementation of the sample application had no major problems. The problems that occur using Knockout mostly occur from uncertainties of whether the observables should be referenced to as functions or not in different situations since this is something that varies a lot. Overall Knockout is simple to use in an application, and using the "usual" HTML attributes simplifies the learning process.

Chapter 4

Testing

Testing in this report is divided into two different areas, unit testing and performance testing. Chapter 4 covers the later area beginning with a description of methodology for web client performance testing. This is followed by a description of the factors that are tested and why. The chapter is ended with an exact specification of the tests that are run and the device they are run on.

4.1 Testing methodology

This section describes some problems with testing client side web applications and a description of what factors are tested.

4.1.1 Problems with testing client side applications

Normally there are four key types of web application performance tests: [5]

- Performance test
- Load test
- Stress test
- Capacity test

Since these types of tests are built for an older structure of web applications load, stress and capacity tests only test the server side of the application, resulting in that only performance tests are relevant for the results in this project. The server tests might be important for the client in terms of the initial load time and load time for data that needs to be requested from the server. This however depends on the implementation of the server independently and is thereby outside the scope of this project. So the relevant tests are the ones that come down to the computational speed tests and the size of the client.

Another major problem when testing client side frameworks is that the tests should analyze performance of the data bindings. Since the purpose of data bindings is to be updated automatically there is no simple way to check when the values actually are updated, and as a developer this should not be done in an actual application. What this leads to in the end is that the performance testing of the frameworks has to use a method that would not be used in an actual application, resulting in test results that might be inaccurate.

4.1.2 Size of the client

According to research about application usage done by the principal company in this study, systems managing big amounts of data have a standard procedure that contains starting up the application and have it running for a long time while performing operations. Therefore the focus of the application needs to be mainly on the runtime and not on starting the application quickly. The size of the client determines how long it takes to initially load and start the application, which then is not the most important factor, but the test is done to check for any extremes that might cause user experience issues.

4.1.3 Speed in calculations

Speed in calculations and business logic is the reason why client side implemented functionality has not been considered previously, therefore this is one of the most important tests. Since JavaScript has quite recently reached a state where it can be considered to be used for data operations this test could result in that client-sided applications are still not feasible when dealing with large amounts of data. This test does not only show the computational speed of JavaScript but also considers whether the frameworks deal with data efficiently and whether the frameworks require unnecessarily much resources for their own operations. The test also checks whether the frameworks disrupt the JavaScript performance using excessive amounts of metadata or in any other way that might affect the user experience negatively [42]. The operations tested include:

- Sorting big amounts of data
- Loading big amounts of data
- Changing bound data values

4.1.4 Storing large data sets

New web browsers rarely have any problem with loading large amounts of data since they are made to be able to handle images and many kinds of heavy files. The test for data storage loads a number of data entries and benchmark memory usage to see if the frameworks store metadata that might grow out of proportion when loading large amounts of data

4.1.5 Runtime testing

Since web applications are closed when the web browser running them is closed and then reloaded when the browser and application is opened again the standard behavior for web browsers and frameworks is to not be running for a long time consecutively. But since the this is a common behaviour for applications managing data the system needs to be able to manage prolonged sessions.

This can cause some problems if data is not trash collected correctly or if the data changes over time. So this test attempts to reenact the scenario of having the application running for a long time by performing operations that could be used in a real life situation and keeping pauses between operations. The test simultaneously benchmarks the long run and smaller parts during the run to see if the performance of the application is affected by the application running for a long time.

4.2 Test environment & affecting factors

This section contains a description of the tests, what device is used, how the test results are received and an hypothesis for what results are expected.

4.2.1 Benchmarking framework

The framework that is used to benchmark the tests is called *Benchmark.js* [3]. *Benchmark.js* is a framework made for testing different JavaScript approaches and comparing them against each other. *Benchmark.js* does not use the standard approach of running a test a defined number of times and then from the results get results with a certain margin of error, but does instead dynamically adjust the number of times a test is run to always achieve a maximum error of 1% in the results. The framework also makes use of several different clocks to achieve the best possible time measurements when running the tests [4] [14].The output from *Benchmark.js* is the frequency that the test was able to run with given in Hz.

4.2.2 Test device and browser

It is not of grave importance that the testing environment is powerful regarding performance since the testing is not used to benchmark the frameworks against other units but only against each other. So as long as all of the tests are done on the same device the differences in results caused by the surrounding system should be minimal. The device that the tests are run on is a Dell Latitude E6420, partly since this is a very commonly used work computer and partly since this is the device that is available to use for testing in this project.
The system specifications of the device are:

- **Manufacturer** Dell Inc.
- **Processor** Intel(R) Core(TM) i7-2640M CPU @ 2.80GHz (4CPUs), 2.8GHz

- **Memory** 8192MB RAM
- **Operating system** Windows 8 Professional N 64-bit (6.1 ,Build 7601)

The tests are run using *Chrome 31.0.1650.57* since it currently is one of the most used web browsers in the world [65]. Another reason to use Chrome for the tests is its memory tools, which show details of memory usage for each Chrome process, resulting in a way to be able to analyze the frameworks memory usage while running the tests.

4.2.3 Test data

The test data attempts to represent real life data using the Person and Case structures defined in the sample application (section 2.3). Test data is generated using *Mockaroo* [46] which is a free web based tool to generate dummy test data that contain realistic values. *Mockaroo* allows developers to specify a number of settings for the test data to be generated, and allows the use of data such as names that are not randomly generated content but actual human names. There are no real benefits to using real values instead of randomized ones other than that it might show whether there are errors in the bindings, confusing forename with surname or such. The main reason to use this kind of test data is that it is more aestetically pleasing and thereby more fun to work with. The data generated in *Mockaroo* can be extracted to a *sql* or *csv* file, which also simplifies the process of inserting the data into a database table.

4.3 Specific tests

This section contains a short description of each test that is run in the experiment, including exact numbers and options. Also for each test is a short description of the hypothesis for the test.

4.3.1 Data binding updates

Data binding updates are tested in two odifferent ways, both consist of a number of variables that are changed a number of times.

The first test consists of 10 integer values that are bound to the view, the integers are each changed as fast as possible. This shows how the frameworks manage multiple updates on the same variables.

The second test consists of a collection of integer values that are bound to the view, the integers are changed as fast as possible in the test. The size of the collection varies between 10 and 2000 elements to show how the framework bindings update time is depending on collection size. The test cycle is measured in the test contains an iteration of all elements in the collection. This test also shows how the framework manage updates in many variables simultaneously.

Both of the tests are run with and without bound values, to see if the frameworks require the same update procedure even though there are no UI elements bound to the values.

Hypothesis for the test is that the frameworks should show their differences in how the data bindings are implemented with dirty-checking compared to change listening, meaning that Angular is expected to be faster for the small amounts of variables but slower when there are many variables.

4.3.2 Loading data

This test loads an amount of data used into the sample application. Then the test compares how much data is needed by the frameworks for metadata and to apply the data bindings. This, since it is important for web applications not to need to store more data than necessary. The data that is loaded is representing 1000 persons and is generated as described in section 4.2.3. The size of the data actually loaded in the test is 132 kb.

Hypothesis for the test is that Angular is the most memory consuming at start up since the framework contains more components to be loaded. Then when running the tests Knockout is probable to require more memory since it uses more information for each bound value than what is used by the standard JavaScript objects used in Angular.

4.3.3 Sorting data

The data that is tested in this test is the same as for the *loading data* test, generated as described in 4.2.3. The data is sorted using different fields and is benchmarked testing memory allocation and performance by the sorting operations. The frameworks should not differ in this category, however it shows iterative capabilities and check if they contain any previously undetected bugs that only occur when working with large amounts of data.

There is no hypothesis for which framework is the fastest in sorting, but Knockout is probable to require more memory because of it carrying more metadata per variable.

4.3.4 Linked dependencies

Linked dependencies is something that is very commonly used in large systems since business logic should be updated to the interface automatically. This makes sure that users do not enter data that is wrong by disabling alternatives that are not accepted by the system. In the sample application an example of linked dependencies is that a case's status can not be set from "new" to "completed". This means that it is unnecessary to even show the "completed" status as an option in the status combobox if the current status of the case is "new". These dependencies can also be applied between different comboboxes, creating chains of options depending on previously selected values. This test tests the frameworks using three comboboxes, where the value of combobox A limits the possible options of combobox B, and combobox B limits the possible options of combobox C. The data in this test is dummy test data generated using a web application called *json-generator*, generates data in a quite similar faction as described in section 4.2.3. The data in the test is randomized string values with 30 characters.

There is no hypothesis for this test.

Chapter 5

Results

Chapter 5 contains the results of the tests in chapter 4 followed by a brief discussion about the test results. Chapter 5 also contains a larger discussion combining the theoretical conclusions with the conclusions from the implementation and the test results, leading to the solution and structure defined in chapter 6.

5.1 Test results

5.1.1 Clarification

All performance results presented in the tables of this section are presented in hz. Thereby, higher numbers are better. In the figures of this chapter the performance results are shown as time per operation (1/hz) since it visualizes the results better, thereby lower results in are better in the figures. The memory tests are measured in kb, since this achieves a detail level that is currently relevant for web applications.

5.1.2 Results

Performance

As can be seen in tables 5.1, 5.3, and 5.5 Knockout has a better result in all performance tests. This is most likely caused by the previously mentioned difference in the implementation of data bindings.

One of the most important results can be seen in the figure 5.1 and 5.2 which are plots of the data presented in table 5.2 - 5.3. The data plotted is the time of each operation cycle in the test. In Knockout the update time of each variable update is done in the same time independently of how many variables are in the same page, which leads to a linear relation of execution time to the size of a cycle.

Meanwhile a plot of the same values for Angular shows a difference where the time for data updates are dependent on the number of variables in the page. As can be seen in figure 5.1, showing an polynomial increase in cycle execution time as the number of variable updates in each cycle increases.

Data	Angular	Knockout
bound values	7,000 hz	9,100 hz
non bound values	7,000 hz	150,000 hz

Table 5.1. Results received from test 1. Described in section: 4.3.1

# variables	Angular	Knockout
10	474 hz	930 hz
50	36 hz	190 hz
100	10 hz	85 hz
250	1.4 hz	35 hz
500	0.30 hz	18 hz
750	0.13 hz	12 hz
1000	0.07 hz	8.9 hz
1500	0.027 hz	6 hz
2000	0.015 hz	4.4 hz

Table 5.2. Results received from test 2 . Described in section: 4.3.1

# variables	Angular	Knockout
10	1,100 hz	79,000 hz
50	94 hz	14,000 hz
100	30 hz	7,400 hz
250	4.4 hz	2,900 hz
500	0.85 hz	1,300 hz
750	0.39 hz	700 hz
1000	0.17 hz	640 hz
1500	0.07 hz	333 hz
2000	0.04 hz	270 hz

Table 5.3. Results received from test 2. Described in section: 4.3.1

	Angular	Knockout
Memory usage before	47,144 kb	36,592 kb
Memory usage after	49,704 kb	45,332 kb
Diff	2,560 kb	8,740 kb
Memory usage sorting	75,132 kb	80,012 kb

Table 5.4. Testing the memory usage storing data and metadata. Memory usage while processing data. Described in sections: 4.3.2 and 4.3.3

	Angular	Knockout
Sorting performance	17 hz	48 hz
Linked dependencies	19.5 hz	31.5 hz

Table 5.5. Testing the data frameworks for performance while processing data. Described in sections: 4.3.3 and 4.3.4

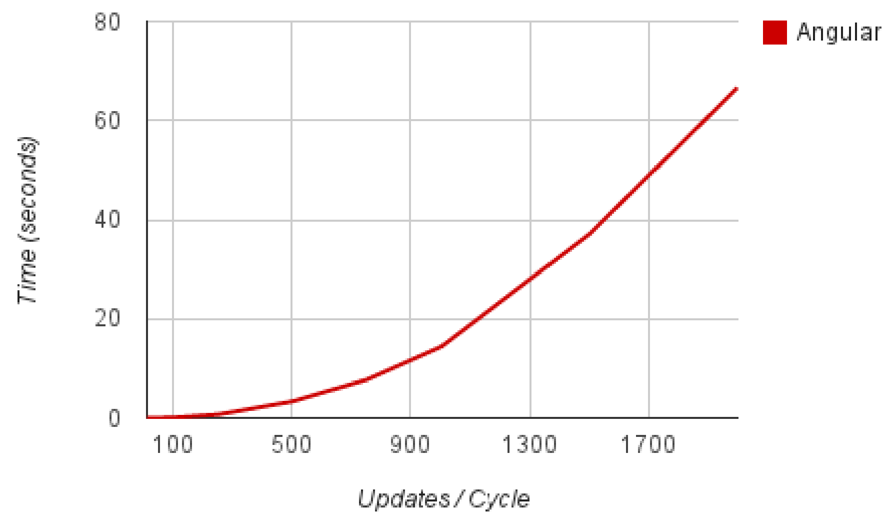

Figure 5.1. Plot Angular time requirement per cycle of updates depending on the cycle size. Data from Test 2. Described in section 4.3.1.

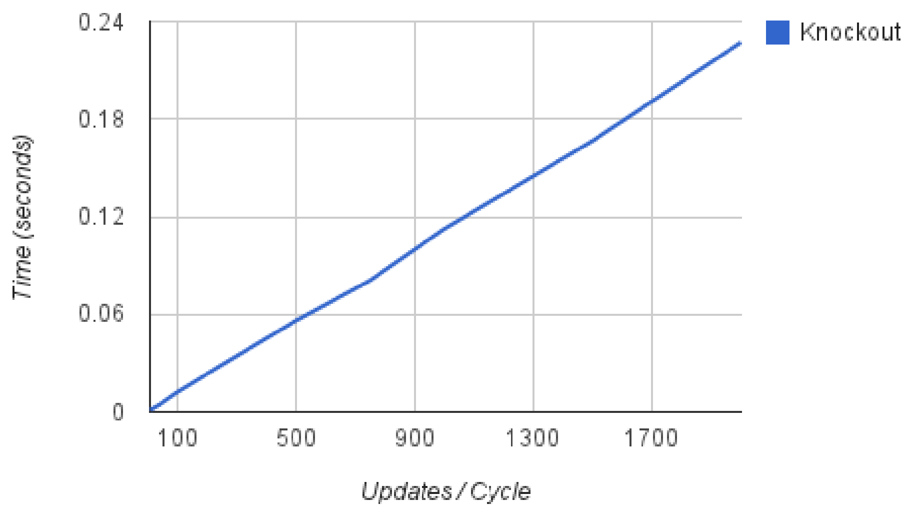

Figure 5.2. Plot of Knockout time requirement per cycle of updates depending on the cycle size. Data from Test 2. Described in section 4.3.1.

In web applications it is not very usual that more than a few values are updated simultaneously or within a short period of time. Therefore, one of the most relevant measurements is the total time needed to update one variables value depending on the number of variables in a page. The time requirement of one variable update depending on the number of variables in the page can be seen plotted in figure 5.3. This also shows a more clear visualization of that the Knockout variable updates are independent on page size, while the time of one value update for Angular is linearly dependent on the number of variables in the same page.

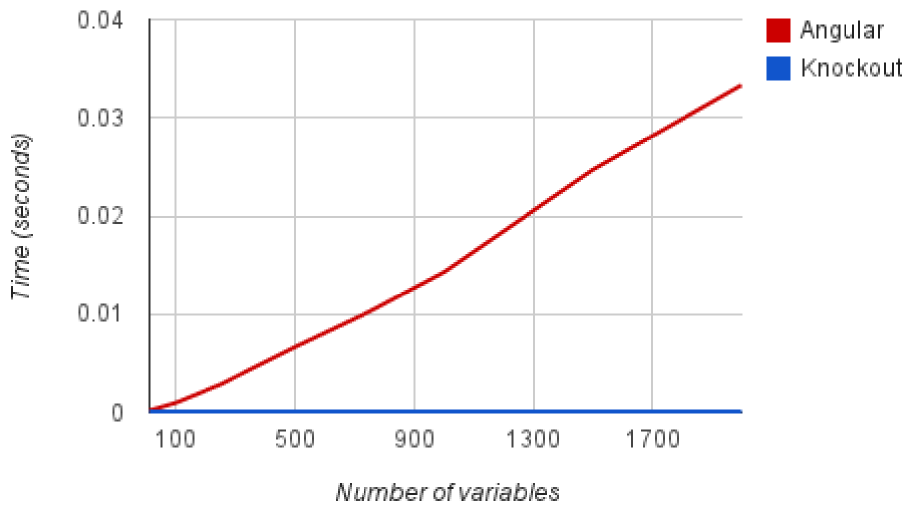

Figure 5.3. Plot of resulting time per variable update depending on the amount of variables in the same page. Data from Test 2. Described in section 4.3.1.

From the test data in table 5.3 it can be seen that the unbound values of Knockout and Angular differ a lot in update speed. The reason for this is that Angular creates bindings for all values in the *$scope* even though the bindings are not bound to any other elements. This is not a big problem for Angular since data that is not supposed to be bound to should not be placed in the *$scope*. However, Knockout does not perform any additional operations on value updates if no elements are bound to the observable, resulting in very good performance on updates of non bound variables.

Memory

The results in memory usage show the combined private and shared memory of the application. What can be seen is that from the beginning the package using Knockout requires less memory to load the initial web page since the package size is smaller. However, when the person data is loaded the amount of memory this requires is larger for Knockout than it is for Angular since there is more metadata to each variable loaded. The memory usages are shown in the bar chart in figure 5.4. What can also be seen is that Knockout requires the use of more memory when processing data, probable reason for this is that the excessive metadata also needs to be managed. In figure 5.4 the bars showing memory requirement while sorting shows the peak value reached during the sorting.

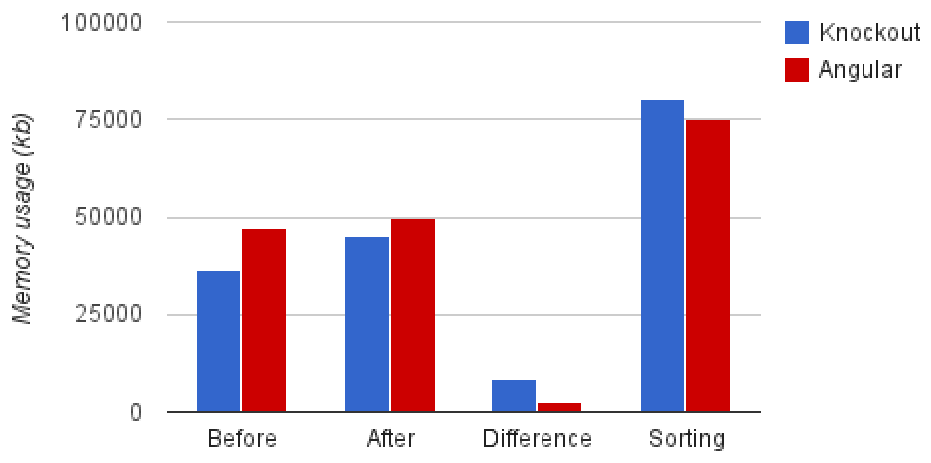

Figure 5.4. Memory usage during the data management tests

5.1.3 Conclusions

The test results show that Knockout is the faster of the two, while using a bit more memory for metadata and Angular is a bit slower, but does not store as much metadata. When looking at the data generated by the tests (figure 5.3), we can see that even though Knockout is much faster that Angular with data binding updates Angular still manages to perform data updates in less than 50 ms when the number of elements in the page are 2000. This is a test of the extreme since web pages rarely have more than 100 elements in a page and above 500 elements in a page should be considered bad design.

In the results from the memory tests it is shown that the difference in memory usage between the two frameworks is quite small. The most important conclusion regarding memory is that since Knockout requires more memory for storing the same amount of data it also has a bigger risk to encounter problems with memory growing during the application runtime. Meanwhile Angular requires more memory initially but shows less tendencies to grow much in memory requirement during the runtime of an application..

The test was mostly to test for if there were any extremes and there were not. The same goes for the test over long duration of time and none of the frameworks showed any problems managing a long runtime.

So concludingly for the test results one could say that even though Knockout was faster than Angular, both were within reasonable bounds within both performance and memory and should therefore not cause any major problems within these areas.

5.2 Conclusions

Combining the conclusions from theory, implementation and results both of the frameworks showed that they were good enough to be used in applications managing large amounts of data. To summarize the results one could say that Knockout is the smaller framework that only manages the data bindings but does it very well, while Angular is the larger framework bringing more components and a more complete solution.

In the implementation section Angular enforced and encouraged the developer to keep the code well organized and structured. The many components with separate uses keeps the code modular and always makes sure that both as a viewer of the code and as developer one would know where to look for a specific functionality. Meanwhile Knockout showed some difficulty since it the *observables* and bindings are parts of the code where there is great risk for errors.

In the performance testing Knockout was faster than Angular in all tests but required more memory to manage data. Since both frameworks showed a performance that was sufficient to provide a good user experience the conclusion to the performance test was that the design of an application using Knockout should make sure that the memory requirement does not become a problem, meanwhile the design of an application using Angular should make sure that each page does not contain too many elements so that the performance does not become a problem.

So the end conclusion is that the structure that the final solution to this project is based on is Angular. Further, the set of rules determining the structure contains some limitations to not use behavior that Angular allows but is not best practice and not things that should be used when the goal is to create maintainable and long lived applications. The final structure is defined and described in chapter 6.

Chapter 6

Structure

Chapter 6 contains a description of the structure defined using the results and conclusions from this project. The section also includes some discussion of weaknesses with the structure, if it needs to be completed with additional components and how to start up a new project working with the structure.

6.1 Defining a structure

The structure defined as solution to the problem is defined in this section as a bulleted list with rules to follow during the development.

- Directory structure should be according to one of the Angular best practice projects. Example of such is *Angular-seed* [25], directory structure shown in section 6.1
- Scripts should be loaded at the end of pages [24].
- All additional HTML attributes and Angular directives need to be properly documented (more than everything else). So that they can be reused and understood by all developers in the project.
- Controllers should never reference the DOM. All such references should be implemented as bindings using Angular *directives*. The controller contains view behaviour and view logic.
- Services/Factories should not reference the DOM (with a few exceptions). All such references should be implemented as bindings using Angular *directives*. Services/Factories should contain view independent logic.
- The $scope object can be edited to from both the view and the controller, which might cause some confusion. Therefore the $scope should be treated as read only in views and write only in controllers. [24]. Important to note is that the $scope is not the model, but contains references to the model.

- Controllers should never depend on other controllers, if interaction between controllers is required then it should be performed through a shared service which both controllers are dependent on, or through event handling via the $scope using *$on()* and *$emit()* .
- All styling should be done in CSS-files. This is web development best practice but can be tempting to skip in some cases. For dynamic style elements adaptions can be done to the CSS using Angular tags.
- In angular the same operations could be done using different tags such as *data-ng-bind="element"*, *ng-bind="element"*, *ng bind="element"*, *data_ng bind="element"* and *{{element}}*. All of these are parsed as the same attribute since Angular parses *’:’, ’_’ and ’-’* the same way and filters *data-* or *x-* tags before the directives. Therefore one syntax should be chosen in the beginning of the project and used in all pages.

- **app** All files that are sent to the client (all non development data)
  - **css** CSS files
  - **img** Image files
  - **js** Javascript files written for this project
  - **lib** Angular and other third party javascript libraries
  - **partials** Angular view partials (html files)
- **config** Config files used in development, setting up testing environment and so on.
- **logs** Log files constructed during testing.
- **scripts** Handy shell/js/ruby scripts used in the development process
- **test** Test source files and libraries

Figure 6.1. Angular-seed directory structure [25]

6.1.1 Comments on the structure

There are always special cases in big projects that does not fit the common structure. This is acceptable as long as the solution to these special cases does not bend the rules defined in the structure. Managing the special cases within the rules might lead to that the special cases are more difficult to implement, but in the end it is worth the extra time since it makes the application more maintainable and stable.

This is also the reason for the structure being defined in the first place, to avoid these exceptions that might make the project difficult to manage.

6.2 Weaknesses

6.2.1 Performance

As discussed previously there are some possible performance issues that could occur from using the dirty checking bindings in Angular. This is very unlikely to be a problem in an actual application but is still something that a developer and designer should be aware of to know where performance problems come from if they occur.

6.2.2 Data model

The structure defined in the solution does not define what data model to use since this is something that varies a lot between different projects and applications, this might be considered a weakness since it is something that must be defined in the beginning of a project to keep the well structured applications that are the goal of this project. One recommendation would be to look at frameworks managing this and what they could do to improve the application at hand.

Example of such, that would fit the application that this project is aimed at is Breeze JS [31]. This is a framework that simplifies data management in web clients and is commonly known to work well with applications using Angular to manage bindings and logic.

6.2.3 Security

The biggest problem with security in client web applications is that all of the code to execute is sent to the client. The result of this is that the code can both be read by the users and edited by the users. Therefore security is not worth spending much time on when only working with the client. Something that is very important is to implement good security on the server side of the application so that changes in the client cannot affect the underlying data. This can often be done using validation tokens and cookies, and should be considered and researched further for all applications.

The same problem exists in the solutions for data validation, since the code at the client side can be altered by the users one can not as a developer count on that data sent by the client is validated correctly, but should validate the data again at a server level to make sure that invalid data that might affect the database in a bad way is not allowed to pass.

Chapter 7

Conclusion

Chapter 7 contains a short summary of the project's results and the final solution, followed by some words about the affects of this project for society and the world. This is followed by what could be researched further within the area.

7.1 Summary

Creating a client-sided web application there are a lot of different frameworks that can be used to simplify the development process and help in making the application easier to maintain and longer lived. Some of the currently most relevant frameworks are Angular JS, Knockout JS and Ember JS. Comparing these in a set group of requirements shows that Angular supports keeping applications well structured better than the others and would therefore be the best one to use as a basis for structuring a web client.

JavaScript is a very forgiving language which allows many operations that are bad for the maintainability and structure of the application. But by following the rules defined in the solution to this project together with not using the Angular components in ways they are not supposed to be used, one can construct advanced web applications that are well structured and long lived.

7.1.1 Socio-ethical effects

The socio ethical effects that the results from this project have are mostly caused by the transition from server-sided to client sided applications. This transition allows for smaller operators and companies to create more advanced and larger web application projects to a smaller cost. Mostly caused by the fact that performance requirements on a server running a client sided application are much smaller and thereby also the costs of running said server are less. With more operators being able to build advanced web applications to a smaller cost this results in more advanced web applications for the world to use and improvements to the worlds largest means of communication, the Internet.

7.2 Future work

Security is a major issue with client-sided web application, therefore this is something that will be necessary to work on in the future. With web application clients growing, more is managed and more risks are added regarding security. The problem is just as described in section 6.2.3 that all of the client side code is sent to the client where it can be read and altered. A solution could possibly be done using checksums and encryptions to check if the client code has been changed, but this is something that is probable to become more interesting in the future. As trends go security rarely becomes the focus of research and development until there has been a major breach, and it is probably only a matter of time until that happens and the focus of web development changes to more security focused applications.

Client side web application frameworks is an area where a lot of development is done, and as seen in sections 2.1 and 2.2.7 the frameworks that are interesting right now has only existed for a few years and were not at all as popular a year ago. So as trends change the recommendation in the area covered by this report might also change, so using these technologies it is constantly relevant to compare the existing solutions with new alternatives.

List of Figures

List of Tables

Bibliography

[1] Jeroen Arnoldus, Mark van den Brand, A. Serebrenik, and J.J. Brunekreef. Code generation with templates. http://link.springer.com.focus.lib.kth.se/book/10.2991/978-94-91216-56-5/page/1, 2012. Atlantis Press.

[2] Jeremy Ashkenas and Community. Coffeescript is a little language that compiles into javascript. http://coffeescript.org/, January 2014.

[3] Benchmark.js. Website. http://benchmarkjs.com/. [Online; accessed October 31, 2013].

[4] Mathias Bynens and John-David Dalton. Bulletproof javascript benchmarks. http://calendar.perfplanet.com/2010/bulletproof-javascript-benchmarks/, December 2010.

[5] Microsoft Corporation. Performance testing guidance for web applications. http://msdn.microsoft.com/en-us/library/bb924375.aspx, September 2007.

[6] Oracle Corporation. Java history timeline. http://oracle.com.edgesuite.net/timeline/java/, 2013.

[7] ECMA. Tc39 - ecmascript (formerly tc39-tg1). http://www.ecma-international.org/memento/TC39.htm, 2013. [Online; accessed October, 2013].

[8] Martin Fowler. Patterns of enterprise application architecture. Addison-Wesley Professional, May 2002. ISBN-10: 0321127420, ISBN-13: 9780321127426.

[9] Jesse James Garrett. Ajax: A new approach to web applications. http://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php, February 2005.

[10] Paul Hammant. Client-side mvc frameworks compare. http://paulhammant.com/2012/02/13/client-side-mvc-frameworks-compared/, February 2012.

[11] Ross Harmes and Dustin Diaz. Pro javascript design patterns. http://link.springer.com.focus.lib.kth.se/book/10.1007/978-1-4302-0496-1/page/1, 2008. Apress.

[12] Matthias Heinrich and Martin Gaedke. Data binding for standard-based web applications. http://dl.acm.org.focus.lib.kth.se/citation.cfm?doid=2245276.2245402, 2012. SAC, 2012 Proceedings of the 27th Anual ACM Symposium on Applied Computing, pp 652-657.

[13] Gordon L. Hempton. The top 10 javascript mvc frameworks reviewed. http://codebrief.com/2012/01/the-top-10-javascript-mvc-frameworks-reviewed/, January 2012.

[14] Monsur Hossain. benchmark.js: how it works. http://hossa.in/2012/12/11/benchmarkjs.html, December 2012.

[15] Cheng Huang, David A. Maltz, Albert Greenberg, and Jin Li. Public dns system and global traffic management. http://research.microsoft.com/en-us/um/people/chengh/slides/pubdns11.pptx.pdf, 2011. Microsoft Research, INFOCOM 2011.

[16] Andy Hunt and Dave Thomas. Oo in one sentence: keep it dry, shy, and tell the other guy. http://ieeexplore.ieee.org.focus.lib.kth.se/stamp/stamp.jsp?tp=\&arnumber=1293081, May - June 2004. Software, IEEE (Volume:21 , Issue: 3).

[17] InfoQ. Top javascript mvc frameworks. http://www.infoq.com/research/top-javascript-mvc-frameworks, February 2013.

[18] Ecma International. Ecmascript® language specification. http://www.ecma-international.org/ecma-262/5.1/, June 2011. 5.1 Edition.

[19] Angular JS. Api. http://docs.angularjs.org/api/. [Online; accessed September-October, 2013].

[20] Angular JS. Blog. http://blog.angularjs.org/. [Online; accessed September-October, 2013].

[21] Angular JS. Faq. http://docs.angularjs.org/misc/faq. [Online; accessed September-October, 2013].

[22] Angular JS. Repository. https://github.com/angular/angular.js. [Online; accessed September-October, 2013].

[23] Angular JS. Webpage. http://angularjs.org/. [Online; accessed September-October, 2013].

[24] Angular JS. Angularjs mtv meetup: Best practices. http://www.youtube.com/watch?v=ZhfUv0spHCY, December 2012.

[25] Angular JS. Angular-seed - the seed for angularjs apps. https://github.com/angular/angular-seed, November 2013.

[26] Backbone JS. Changelog. http://backbonejs.org/#changelog. [Online; accessed September-October, 2013].

[27] Backbone JS. Webpage. http://backbonejs.org/. [Online; accessed September-October, 2013].

[28] Backbone JS. Projects and companies using backbone. https://github.com/jashkenas/backbone/wiki/Projects-and-Companies-using-Backbone, January 2014.

[29] Batman JS. Download page. http://batmanjs.org/download.html. [Online; accessed September-October, 2013].

[30] Batman JS. Webpage. http://http://batmanjs.org/. [Online; accessed September-October, 2013].

[31] Breeze JS. Rich data for javascript apps is a breeze. http://www.breezejs.com/, 2013.

[32] Ember JS. Api. http://emberjs.com/api/. [Online; accessed September-October, 2013].

[33] Ember JS. Repository. https://github.com/emberjs/ember.js. [Online; accessed September-October, 2013].

[34] Ember JS. The post-1.0 release cycle. http://emberjs.com/blog/2013/09/06/new-ember-release-process.html. [Online; accessed September-October, 2013].

[35] Ember JS. Webpage. http://emberjs.com/. [Online; accessed September-October, 2013].

[36] Knockout JS. Api. http://knockoutjs.com/documentation/introduction.html. [Online; accessed September-October, 2013].

[37] Knockout JS. Repository. https://github.com/knockout/knockout/commits/master. [Online; accessed September-October, 2013].

[38] Knockout JS. Webpage. http://knockoutjs.com/index.html. [Online; accessed September-October, 2013].

[39] JSON. Introducing json. http://www.json.org/. [Online; accessed November 12th, 2013].

[40] Karnow and Curtis A. What's a bug? whatever the customer says it is., September 1992. Computerworld, Vol.26(37), p.33(1).

[41] Iwan Vosloo (Reahl Software Services (Pty) Ltd) and Derrick G. Kourie (University of Pretoria South Africa). Server-centric web frameworks: An overview. http://dl.acm.org.focus.lib.kth.se/citation.cfm?doid=1348246.1348247, April 2008. ACM Computing Surveys (CSUR) Volume 40 Issue 2,.

[42] Giuseppe A. Di Luccaa and Anna Rita Fasolinob. Testing web-based applications: The state of the art and future trends. http://www.sciencedirect.com.focus.lib.kth.se/science/article/pii/S0950584906000851?np=y, December 2006. Information and Software Technology, Volume 48, Issue 12, Pages 1172–1186.

[43] Meteor. People page. http://www.meteor.com/about/people. [Online; accessed September-October, 2013].

[44] Meteor. Repository. https://github.com/meteor/meteorl. [Online; accessed September-October, 2013].

[45] Meteor. Webpage. http://www.meteor.com/. [Online; accessed September-October, 2013].

[46] Mockaroo. Mockaroo, realistic test data generator. http://http://www.mockaroo.com/. [Online; accessed November 13th, 2013].

[47] Josip Maras (University of Split), Jan Carlson (Mälardalen University), and Ivica Crnkovi (Mälardalen University). Extracting client-side web application code. http://dl.acm.org.focus.lib.kth.se/citation.cfm?doid=2187836.2187947, 2012. WWW '12 Proceedings of the 21st international conference on World Wide Web.

[48] Addy Osmani, Sindre Sorhus, Pascal Hartig, Stephen Sawchuk, and Colin Eberhardt. Todomvc, helping you select an mv* framework. http://todomvc.com/, September 2013.

[49] Manuel Palacio. Angularjs rocks. http://manuel-palacio.blogspot.se/2013/01/angularjs-rocks.html, January 2013.

[50] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber. Capability maturity model for software, version 1.1. http://www.sei.cmu.edu/reports/93tr024.pdf, February 1993. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213.

[51] IBM) Peter Bell, (Senior VP of Engineering. A survey of client mvc frameworks. http://www.ibm.com/developerworks/web/library/wa-clientmvc/index.html?ca=dat-, June 2012.

[52] readerspark. Top javascript frameworks (client & server side). http://www.readerspark.com/threads/top-javascript-frameworks-client-server-side.54/, September 2013.

[53] Microsoft research. Microsoft application architecture guide, 2nd edition. http://msdn.microsoft.com/en-us/library/ff650706.aspx, October 2009. Chapter 1: What is Software Architecture?1.

[54] Microsoft research. Microsoft application architecture guide, 2nd edition. http://msdn.microsoft.com/en-us/library/ee658084.aspx, October 2009. Chapter 4: A Technique for Architecture and Design.

[55] Steven Sanderson. Rich javascript applications – the seven frameworks (throne of js, 2012). http://blog.stevensanderson.com/2012/08/01/rich-javascript-applications-the-seven-frameworks-throne-of-js-2012/, August 2012.

[56] Douglas C Schmidt and Carlos O'Ryan. Patterns and performance of distributed real-time and embedded publisher/subscriber architectures. http://www.sciencedirect.com.focus.lib.kth.se/science/article/pii/S016412120200078X, June 2003. Journal of Systems and Software, Volume 66, Issue 3, pp 213-223.

[57] Rubala Sivakumar and Kodhai.E. Code clones detection in websites using hybrid approach. http://research.ijcaonline.org/volume48/number13/pxc3880402.pdf, 2012. International Journal of Computer Applications (0975 – 888), Volume 48– No.13.

[58] Stackoverflow. Tag: angularjs. http://stackoverflow.com/questions/tagged/angularjs/info. [Online; accessed October 28, 2013].

[59] Stackoverflow. Tag: ember.js. http://stackoverflow.com/tags/ember.js/info. [Online; accessed October 28, 2013].

[60] Stackoverflow. Tag: knockout.js. http://stackoverflow.com/tags/knockout.js/info. [Online; accessed October 28, 2013].

[61] SUSE Studio. Client-side js mv* framework roundup. http://blog.susestudio.com/2013/03/client-side-js-mv-framework-roundup.html, March 2012.

[62] thinktecture. Combining angularjs with existing components. http://henriquat.re/directives/advanced-directives-combining-angular-with-existing-components-and-jquery/angularAndJquery.html, 2013. [Online; accessed February 2, 2014].

[63] World Wide Web Consortium (W3C). A short history of javascript. http://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript, June 2012.

[64] Wikipedia. Business logic. http://en.wikipedia.org/wiki/Business_logic. Last modified September 2013, accessed November 2013.

[65] Wikipedia. Usage share of web browsers. `http://en.wikipedia.org/wiki/Usage_share_of_web_browsers`. [Online; accessed November 13th, 2013].

[66] Joseph Williams. The web services debate: J2ee vs. .net. `http://dl.acm.org.focus.lib.kth.se/citation.cfm?doid=777313.777342`, 2003. Sun Professional Services, Fort Collins, CO.