



**HoGent**

*Faculteit Bedrijf en Organisatie*

**ANGULAR, REACT EN EMBER:  
EXPERIMENTELE ANALYSE EN PRESTATIEVERGELIJKING**

Seyed Kavooos Boloorchii

Scriptie voorgedragen tot het bekomen van de graad van

**Bachelor in de toegepaste informatica**

Promotor:  
Lieven Smits  
Promotor:  
Stefaan De Cock

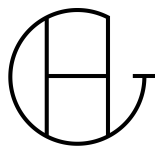
Instelling: -

Academiejaar 2016-2017

Tweede examenperiode

© 2017, Seyed Kavooos Boloorchí. Niets uit deze uitgave mag worden verveelvoudigd, opgeslagen in een geautomatiseerd gegevensbestand, of openbaar gemaakt, in enige vorm of op enige wijze, hetzij elektronisch, mechanisch, door fotokopieën, opnamen of enige andere manier, zonder voorafgaande schriftelijke toestemming van de auteur.  
Het gebruik of de reproductie van bepaalde informatie uit dit werk is enkel toegestaan voor persoonlijk gebruik en mits bronvermelding. Elk gebruik voor commerciële of publicitaire doeleinden is verboden.

Deze bachelorproef/scriptie is gemaakt door Seyed Kavooos Boloorchí, student aan de Hogeschool Gent, ter voltooiing van Bachelor in de toegepaste informatica. De standpunten die in deze bachelorproef/scriptie zijn verwoord, zijn louter het persoonlijke standpunt van de individuele auteur en reflecteren niet noodzakelijkerwijs de mening, het officiële standpunt of het beleid van de Hogeschool Gent.



**HoGent**

Faculteit Bedrijf en Organisatie

Angular, React en Ember: Experimentele analyse en prestatievergelijking

Seyed Kavooos Boloorch

Scriptie voorgedragen tot het bekomen van de graad van  
professionele bachelor in de toegepaste informatica

Promotor:  
Lieven Smits  
Co-promotor:  
Stefaan De Cock

Instelling: —

Academiejaar: 2016-2017

Tweede examenperiode



Faculty of Business and Information Management

Angular, React and Ember: Experimental analysis and performance comparison

Seyed Kavous Boloorch

Thesis submitted in partial fulfillment of the requirements for the degree of  
professional bachelor of applied computer science

Promotor:  
Lieven Smits  
Co-promotor:  
Stefaan De Cock

Institution: —

Academic year: 2016-2017

Second examination period



# Samenvatting

Angular, React en Ember zijn drie jonge<sup>1</sup>, maar krachtige open-source JavaScript frameworks en deze drie frameworks hebben elk een stabiele vrijgegeven versie<sup>2</sup>.

In deze scriptie, als onderzoeksvraag, wordt onderzocht welke aspecten van belang zijn in de keuze tussen Angular, React en Ember. Om deze beter te kunnen beantwoorden wordt de vraag ontleed in enkele deelvragen:

- Wat hebben Angular, Ember en React gemeenschappelijk met elkaar?
- Wat zijn de belangrijkste verschillen tussen Angular, Ember and React?
- Welk framework heeft de hoogste performantie (qua snelheid en memory-footprint)?
- Zijn er situaties waar men best het één of het ander gebruikt?
- Welk framework heeft de hoogste populariteit (gebaseerd op Google Trends, vragen van StackOverflow, hun Github sterren, medewerker, open en opgeloste issues)?

In eerste instantie<sup>3</sup> hebben we kort gekeken naar de fundamentele concepten die JavaScript, TypeScript, framework, single-page application en MV\* design patterns zijn.

In de volgende fase<sup>4</sup> zijn Angular, React en Ember apart bestudeerd om een algemeen beeld te verkrijgen van elk framework zonder ze in detail te vergelijken met elkaar. Vervolgens hebben we een paar testgevallen uitgevoerd om te weten welk framework sneller is en minder memory-footprint heeft<sup>5</sup>.

---

<sup>1</sup>Angular gelanceerd op 2016, React gelanceerd op 2013 en Ember gelanceerd op 2011

<sup>2</sup>De laatste versie van Angular (2017b) is 2.4.9, van React (2017b) is 15.5.4 en Ember (2017b) is 2.13.2.

<sup>3</sup>Hoofdstuk 2

<sup>4</sup>Hoofdstuk 4, 5 and 6

<sup>5</sup>Hoofdstuk 3

Tenslotte hebben we deze frameworks geanalyseerd en vergeleken<sup>6</sup> met elkaar om een antwoord te vinden voor onze onderzoeksvraag.<sup>7</sup>

---

<sup>6</sup>Hoofdstuk 7

<sup>7</sup>Hoofdstuk 8



# Abstract

Angular, React and Ember are three young<sup>8</sup>, but powerful open-source JavaScript frameworks and all them have stable release versions<sup>9</sup>.

This thesis, as its research question, is investigated what aspects are important in the selection between Angular, React and Ember. In order to provide a suitable answer, we break down the research questions into several sub-questions:

- What do Angular, Ember and React have in common?
- What are the most important differences between Angular, Ember and React?
- Which framework has the highest performances (in terms of speed and memory footprint)?
- Is it possible to say if one of these three frameworks is more suitable for a specific project?
- Which framework is more popular (based on Google Trends, StackOverflow questions, their Github stars, contributors, open and closed issues)?

As a first step<sup>10</sup>, we took a brief look at the fundamental concepts which are JavaScript, TypeScript, framework, single-page application and MV\* design patterns.

In the next phase<sup>11</sup>, Angular, React and Ember are studied separately to get a general view of each framework without comparing the three frameworks with each other in detail. Then, we preformed some test cases in order to know which framework is faster and has

---

<sup>8</sup>Angular released in 2016, React released in 2013 and Ember released in 2011

<sup>9</sup>Angular (2017b) latest version is 2.4.9, React (2017b) latest version is 15.5.4 and Ember (2017b) latest version is 2.13.2.

<sup>10</sup>Chapter 2

<sup>11</sup>Chapter 4, 5 and 6

less memory footprint<sup>12</sup>.

And last but not least, we analyzed and compared these frameworks<sup>13</sup> with each other in order find an answer to our research question<sup>14</sup>.

---

<sup>12</sup>Chapter 3

<sup>13</sup>Chapter 7

<sup>14</sup>Chapter 8

# Preface

The bachelor thesis along with the internship are the last parts of the study Applied Computer Science at University College Gent (HoGent). In the last year of this program, I participated in a course, namely Webapp, and I was lucky to have Joeri Van Steen as my teacher. It was then that I realized that I wanted to become an expert in one of the JavaScript frameworks. But I, as a beginner, always struggled with choosing the right framework. Big names like Angular, Ember, React and many more came whenever I tried to search the right one. Therefore, I decided to compare some of them in my bachelor thesis.

First of all I would like to thank my promoter, Lieven Smits, for all the tips and feedback. I would also like to thank my co-promoter, Stefaan De Cock, who was also my first JavaScript teacher and I learned a lot from his courses.

I would like to thank my friends and fellow students – Arno Marchand, Lara Delange, Tom De Witte, Daan Floré and Robin Lodrigo. Thank you all for your kind and friendly behavior which made these years a pleasant experience for me.

I would especially like to thank my parents and my two sisters for the love and support I have gotten over the years.

Finally, I would like to thank and dedicate this thesis to my girlfriend, Sahel, for her constant enthusiasm, encouragement and support. You are the salt of the earth, and I undoubtedly could not have done this without you. I am very grateful to have you on my side.

Kavoos

June 2017, Ghent



# Contents

<b>1</b>	<b>Introduction .....</b>	<b>17</b>
1.1	Initial thesis proposal	18
1.2	Methodology	19
1.3	Problem statement and research questions	19
1.4	Research area	19
1.5	Layout of this bachelor thesis	20
<b>2</b>	<b>Background and theory .....</b>	<b>21</b>
2.1	JavaScript, ECMAScript and TypeScript	21
2.2	Framework	22
2.3	Single-page application	22
2.4	MV* design patterns	22
2.4.1	MVC .....	22

2.4.2	MVP .....	22
2.4.3	MVVM .....	23
<b>3</b>	<b>Benchmarking .....</b>	<b>25</b>
<b>3.1</b>	<b>Technical aspects</b>	<b>25</b>
<b>4</b>	<b>Angular .....</b>	<b>29</b>
<b>4.1</b>	<b>Foundations</b>	<b>30</b>
4.1.1	Modules .....	30
4.1.2	Components .....	31
4.1.3	Templates .....	32
4.1.4	Data binding .....	32
4.1.5	Directives .....	33
4.1.6	Services .....	34
<b>4.2</b>	<b>Features</b>	<b>34</b>
<b>5</b>	<b>React .....</b>	<b>37</b>
<b>5.1</b>	<b>Foundations</b>	<b>37</b>
5.1.1	JSX .....	37
5.1.2	Elements .....	37
5.1.3	Components .....	38
5.1.4	Props .....	38
5.1.5	State .....	38
5.1.6	Events .....	38
5.1.7	Data binding .....	39
<b>5.2</b>	<b>Features</b>	<b>39</b>

<b>6</b>	<b>Ember</b> .....	<b>41</b>
<b>6.1</b>	<b>Foundations</b>	<b>41</b>
6.1.1	Ember Object .....	41
6.1.2	Router and route .....	41
6.1.3	Templates .....	42
6.1.4	Components .....	42
6.1.5	Controllers .....	42
6.1.6	Views .....	42
<b>6.2</b>	<b>Features</b>	<b>42</b>
<b>7</b>	<b>Analysis</b> .....	<b>45</b>
<b>7.1</b>	<b>Overview</b>	<b>45</b>
7.1.1	Size .....	45
7.1.2	Get started and configuration .....	46
<b>7.2</b>	<b>Foundations</b>	<b>47</b>
7.2.1	Creating components and views .....	47
7.2.2	Template syntax .....	47
7.2.3	Data binding .....	47
7.2.4	Routing .....	49
<b>7.3</b>	<b>Features</b>	<b>49</b>
7.3.1	Angular .....	49
7.3.2	React .....	51
7.3.3	Ember .....	52
<b>7.4</b>	<b>Performance</b>	<b>53</b>
7.4.1	Speed .....	53

7.4.2	Memory footprint .....	53
7.5	Popularity, maturity and community	53
8	Conclusion .....	59
	Bibliography .....	65



# Glossary

**AJAX** AJAX (asynchronous JavaScript and XML) is the method of exchanging data with a server, and updating parts of a web page – without reloading the entire page. (Segue-Technologies, 2013). 22

**Dart** Dart is a general purpose programming language. It is a language in the C tradition, designed to be familiar to the vast majority of programmers. Dart is purely object-oriented, class-based, optionally typed and supports mixin-based inheritance and actor-style concurrency. (Bracha, 2015). 29

**data binding** Data binding is a mechanism for coordinating parts of a template with parts of a component. (Angular, 2017a). 32, 47

**decorator** Decorators are functions that modify JavaScript classes. Angular has many decorators that attach metadata to classes so that it knows what those classes mean and how they should work. (Angular, 2017a). 30, 31, 33

**framework** A framework or a software framework is a reusable environment that is part of a larger software platform. They are specifically geared toward facilitating the development of software applications and include components, such as libraries of code, support programs, compilers, tool sets, and specific APIs that facilitate the flow of data. Web application frameworks are software frameworks used to streamline web app and website development, web services, and web resources. (Wodehouse, 2013). 7, 17, 18, 19

**Jasmine** Jasmine is a behavior driven development framework for JavaScript. Jasmine provides functions to help with structuring tests and also making assertions. As the tests grow, keeping them well structured and documented is vital, and Jasmine helps achieve this. (AngularJS, 2017). 17

**Karma** Karma is a JavaScript command line tool that can be used to spawn a web server which loads the application's source code and executes the tests. Karma can be configured to run against a number of browsers, which is useful for being confident that the application works on all browsers that need to support. (AngularJS, 2017).

17

**separation of concerns** Separation of concerns (SoC) is kind of divide and conquer strategy that software engineers use. There are various ways in which separation of concerns can be achieved. In terms of software design and coding, it is found in modularization of code and in in object-oriented design. There may be separation in time; for example, developing a schedule for a collection of periodic computing tasks with different periods. (Laplante, 2007). 22

**single-page application** Single page applications (SPAs) are web applications that load a single HTML page and dynamically update that page as the user interacts with the applications. SPAs use AJAX and HTML5 to create fluid and responsive Web applications, without constant page reloads. (Wasson, 2013). 15, 17, 22, 59

**XSS attack** Cross-site Scripting is a type of attack that can be carried out to compromise users of a website. The exploitation of a XSS flaw enables attackers to inject client-side scripts into web pages viewed by users. (Detectify, 2015). 37

# Acronyms

**API** Application Programming Interface. 13, 22, 34, 39, 50, 51

**CLI** Command-Line Interface. 35, 42, 50, 52, 60, 61

**DOM** Document Object Model. 25, 33, 34, 37, 38, 45, 49

**ECMA** European Computer Manufacturer's Association. 21

**ES** ECMAScript. 29, 38

**HTML** Hypertext Markup Language. 22, 23, 31, 32, 33, 42, 47, 49

**IDE** Integrated Development Environment. 35, 51

**JIT** Just in Time. 21

**MV\*** Model-View-Controller/ViewModel/Presenter. 18, 19, 59

**MVC** Model-View-Controller. 18, 22, 23, 22, 23, 67

**MVP** Model-View-Presenter. 18, 23, 22, 23, 67

**MVVM** Model-View-ViewModel. 18, 23, 67

**OS** Operating System. 34, 45, 50

**SPA** Single-page application. 22

**UI** User Interface. 23, 35, 38, 47, 50

**URL** Uniform Resource Locator. 41, 43, 52, 59

**XSS** Cross-Site-Scripting. 14



# 1. Introduction

Using JavaScript frameworks for developing single-page applications can make web developers tasks much easier than creating their own JavaScript code. However, it may have some disadvantages such as:

- Not learning the deep-down JavaScript code because of using shortcut functions within the framework, or
- JavaScript frameworks are bloated and contain a great amount of code which will never be used, or
- Making users download more than what is actually needed<sup>1</sup>. (Walsh, 2007)

JavaScript frameworks should be used for the most important reasons such as:

- Not reinventing the wheel.
- Doing more with less code. Most JavaScript frameworks provide function "chaining". Chaining allows to do more with less code. Less code means less maintenance time, less download time, and less coding time.
- Saving time. Let the experts do the tough part. you take their work and make what you would like of it.
- Chances are, you are not the expert.
- Speeding thrills. The creators of JavaScript frameworks put a lot of effort into making sure their frameworks are fast.
- Avoiding cryptic JavaScript base code. Why use JavaScript's default functions when you can use a framework's English-named function. Figure 1.1 shows the differences between JavaScript/jQuery code vs. Angular for a sample code. (Walsh, 2007)

---

<sup>1</sup>Packages like Karma and Jasmine, when using Angular

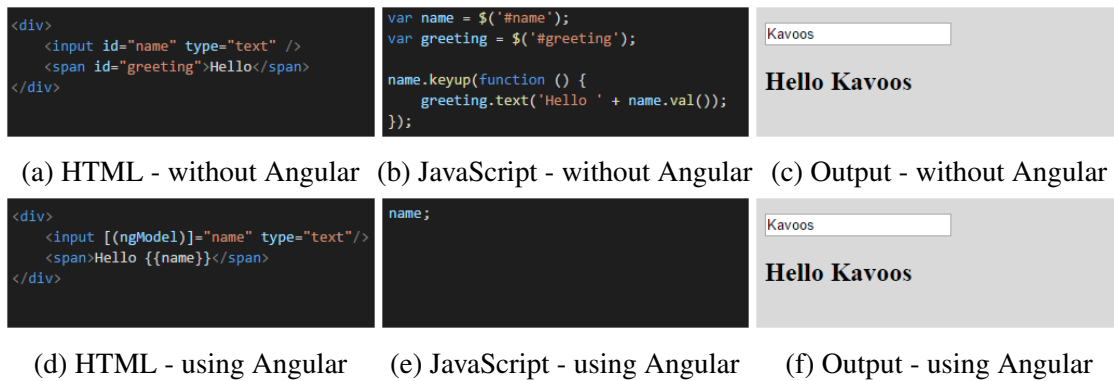


Figure 1.1: JavaScript/jQuery sample code vs. Angular

There exist several JavaScript frameworks – more than 100 (JavaScripting, 2017). Each framework has its own advantages and disadvantages – like speed, memory footprint, size and complexity. Most of the time, it is very difficult to decide which framework is the most suitable one for a specific project. (Walsh, 2007)

## 1.1 Initial thesis proposal

The initial title of this thesis was “*Angular, Backbone, React and Ember: Experimental analysis and performance comparison*” and as it is significant Backbone is dropped from it in the thesis. In this section, the reason of dropping Backbone is discussed.

Backbone does not provide structure. It rather provides some basic tools can be used to create structure, leaving it up to the developer to decide how to structure his or her application, and there are also many blanks to fill. Things such as memory management have to be carefully considered. The lack of view lifecycle management makes route/state changes prone to memory leaks unless you take care of cleaning up everything yourself. (Shaked, 2011)

While it is true that many of the functions not provided by Backbone itself could be filled by third-party plugins, this also means that there are many choices to be made when creating an application, as many functions have several alternative plugins. For example, nested models can be provided by Backbone.DocumentModel, Backbone.NestedTypes, Backbone.Schema, Backbone-Nested, backbone-nestify, just to name a few. Deciding which one is the best for a project requires research, which in turn takes time – and one of the main purposes of using a framework is to save you time. (Shaked, 2011)

Backbone, is not MVC. It’s also not MVP, nor is it MVVM. It takes bits and pieces from different flavors of the MV\* family and it creates a very flexible library of tools that we can use to create amazing websites. (Bailey, 2011)

It absolutely has its own advantages, like being lightweight in terms of the package size, so small and basic but it does not fit completely in the same category as Angular, React

and Ember to be compared with.

## 1.2 Methodology

In this thesis, we performed an experimental analysis and performance comparison among the three selected frameworks. To this end, next to the literature study, each framework is investigated separately.

In order to know which framework is faster and has less memory footprint for a few test cases, it is necessary to have a benchmark. But since the benchmark must be repeated in order to reduce sampling errors, a manual extraction is not a suitable option when running a benchmark. In this thesis, a custom solution from Krause (2016a) is chosen which used the Selenium Webdriver and it can report the raw performance log entries from Chrome's timeline.

## 1.3 Problem statement and research questions

In this thesis, we concentrate on three of the most popular frameworks: Angular, Ember and React.

We investigate whether the benefits of each of the named frameworks outweigh the other ones and could a perfect choice be made between these three.

The research question of this thesis is what aspects are important in the selection between Angular, Ember and React?

In order to provide a suitable answer, we break down the research questions into several sub-questions:

- What do Angular, Ember and React have in common?
- What are the most important differences between Angular, Ember and React?
- Which framework has the highest performances (in terms of speed and memory footprint)?
- Is it possible to say if one of these three frameworks is more suitable for a specific project?
- Which framework is more popular based on Google Trends, StackOverflow questions, their Github stars, contributors, open and closed issues?

## 1.4 Research area

There exist several JavaScript frameworks – more than 100 (JavaScripting, 2017). Each framework has its own advantages and disadvantages – like speed, memory footprint,

size and complexity. This research focuses on three of the MV\* JavaScript frameworks, namely Angular, Ember and React. We have selected these frameworks for comparison because each one of them has been created by powerful IT companies, corporations, and individuals such as Google (Angular), Facebook (React), and Yehuda Katz (2017) (Ember) – a member of the jQuery, Ruby on Rails and SproutCore core teams. Additionally, all these three frameworks have stable release versions which leads to more accurate and reliable conclusions.

In this thesis, we used these latest versions which are 2.4.9 for Angular, 15.5.4 for React and 2.13.2 for Ember. Also note that Angular v1.\* is officially referred as AngularJS.

## 1.5 Layout of this bachelor thesis

Chapter 2 provides the required background concepts related to the topic explored in this thesis.

The overview and technical aspects of the benchmarking is discussed in chapter 3.

In Chapter 4, 5 and 6 each framework is investigated separately. Chapter 7 presents comparison between the three frameworks and an analysis of the developed case study.

And finally, chapter 8 presents a conclusion which provides the answer to the research question stating what aspects are important in the selection between Angular, Ember and React.



## 2. Background and theory

### 2.1 JavaScript, ECMAScript and TypeScript

JavaScript is a JIT-compiled programming language with first-class functions. (Alexiou, 2017)

First-class functions are treated like any other variable. For example, a function can be passed as an argument to other functions, can be returned by another function and can be assigned as a value to a variable. (Cullocà, 2016)

The first big change for JavaScript after its public release came in the form of ECMA standardization. ECMA is an industry association formed in 1961 concerned solely with standardization of information and communications systems. For trademark reasons, the ECMA committee was not able to use JavaScript as the name. The alternatives were not liked by many either, so after some discussion it was decided that the language described by the standard would be called ECMAScript. Today, JavaScript is just the commercial name for ECMAScript. (Peyrott, 2017)

TypeScript starts from the same syntax and semantics that JavaScript developers know today. Use existing JavaScript code, incorporate popular JavaScript libraries, and call TypeScript code from JavaScript. It compiles to clean, simple JavaScript code which runs on any browser that supports ECMAScript 3 (or newer). (TypeScript, 2017)

TypeScript offers support for the latest and evolving JavaScript features, including those from ECMAScript 2015 and future proposals, like async functions and decorators, to help build robust components. These features are available at development time for high-confidence app development, but are compiled into simple JavaScript that targets

ECMAScript 3 (or newer) environments. (TypeScript, 2017)

## 2.2 Framework

A framework is a foundation that a programmer may extend using their own code. It might include a set of software libraries, compilers, interpreters, or an API. In general, it provides an environment that facilitates a specific type of programming for a software development project. (Hope, 2017)

## 2.3 Single-page application

Single-page applications (SPAs) are web applications that load a single HTML page and dynamically update that page as the user interacts with the application. SPAs use AJAX and HTML5 to create fluid and responsive web applications, without constant page reloads. However, this means much of the work happens on the client side, in JavaScript. For the traditional ASP.NET developer, it can be difficult to make the leap. Luckily, there are many open source JavaScript frameworks – like Angular, React and Ember – that make it easier to create SPAs. (Wasson, 2013)

## 2.4 MV\* design patterns

### 2.4.1 MVC

MVC is an architectural design pattern that encourages improved application organization through a separation of concerns. It enforces the isolation of business data (Models) from user interfaces (Views), with a third component (Controllers) traditionally managing logic and user-input. The pattern was originally designed by Trygve Reenskaug during his time working on Smalltalk-80 (1979) where it was initially called Model-View-Controller-Editor. (Osmani, 2012)

As figure 2.1a shows, here is some the characteristics of MVC

- The input is directed at the Controller
- Many-to-one relation between Controller and View
- Controller passes the Model to the View, so View has knowledge about the Controller

### 2.4.2 MVP

Model-view-presenter (MVC) is a derivative of the MVC design pattern which focuses on improving presentation logic. It originated at a company named Taligent in the early 1990s while they were working on a model for a C++ CommonPoint environment. Whilst

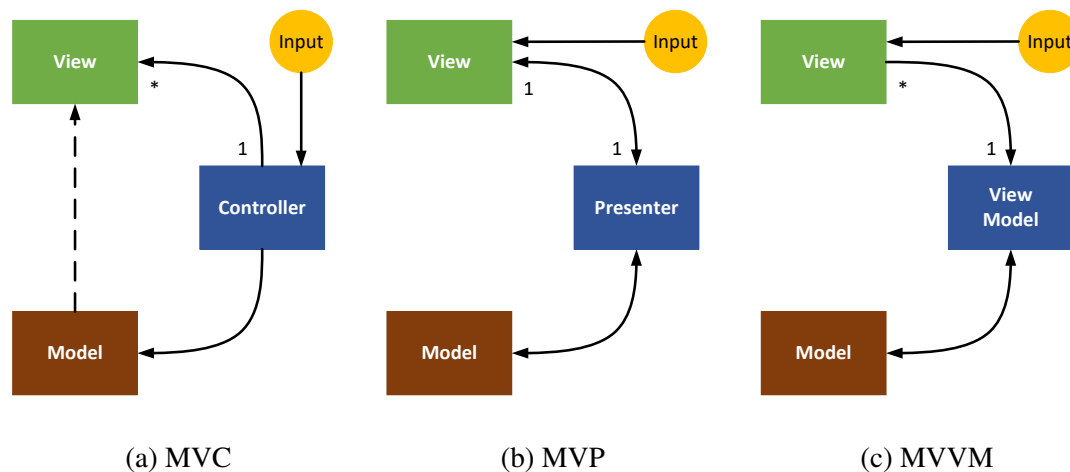


Figure 2.1: MVC vs. MVP vs. MVVM

both MVC and MVP target the separation of concerns across multiple components, there are some fundamental differences between them. (Osmani, 2012)

As figure 2.1b also shows, here is some the characteristics of MVP

- The input is directed at the View
- Many-to-one relation between View and Presenter
- View is not aware of Model

### 2.4.3 MVVM

MVVM (Model View ViewModel) is an architectural pattern based on MVC and MVP, which attempts to more clearly separate the development of user-interfaces (UI) from that of the business logic and behavior in an application. To this end, many implementations of this pattern make use of declarative data bindings<sup>1</sup> to allow a separation of work on Views from other layers. This facilitates UI and development work occurring almost simultaneously within the same codebase. UI developers write bindings to the ViewModel within their document markup (HTML), where the Model and ViewModel are maintained by developers working on the logic for the application. (Osmani, 2012)

As figure 2.1c also shows, here is some the characteristics of MVVM

- The input is directed at the View
- One-to-many relation between View and ViewModel
- View is not aware of model

<sup>1</sup> Binding JavaScript variables to fields



## 3. Benchmarking

In order to know which framework is faster for a few test cases, it is necessary to have a benchmark but first of all we need to have a clear vision what fast actually means.

Figure 3.1 shows a Google Chrome timeline. The timeline includes three relevant parts. The first line – the yellow one – labeled with "Event(click)". Going into deep enough, a method can be found in the controller that preforms the model changes that should be benchmarked – in this case the "run" method of an Angular controller is the very small dark blue line below "r.\$apply".

Right after the event handling, three purple lines show up. Purple is used in Chrome's timeline to indicate rendering.

The third part is the green line which stands for painting in Google Chrome. (Krause, 2016a)

### 3.1 Technical aspects

In this thesis, for the purposes of the benchmark, the duration from the start of the DOM event to the end of the rendering (include painting) is measured.

But since the benchmark must be repeated in order to reduce sampling errors, a manual extraction is not a suitable option when running a benchmark.

There exist different tools that could automate the measurement, like Protractor-Perf and Benchpress. In this thesis, a custom solution from Krause (2016a) is chosen which

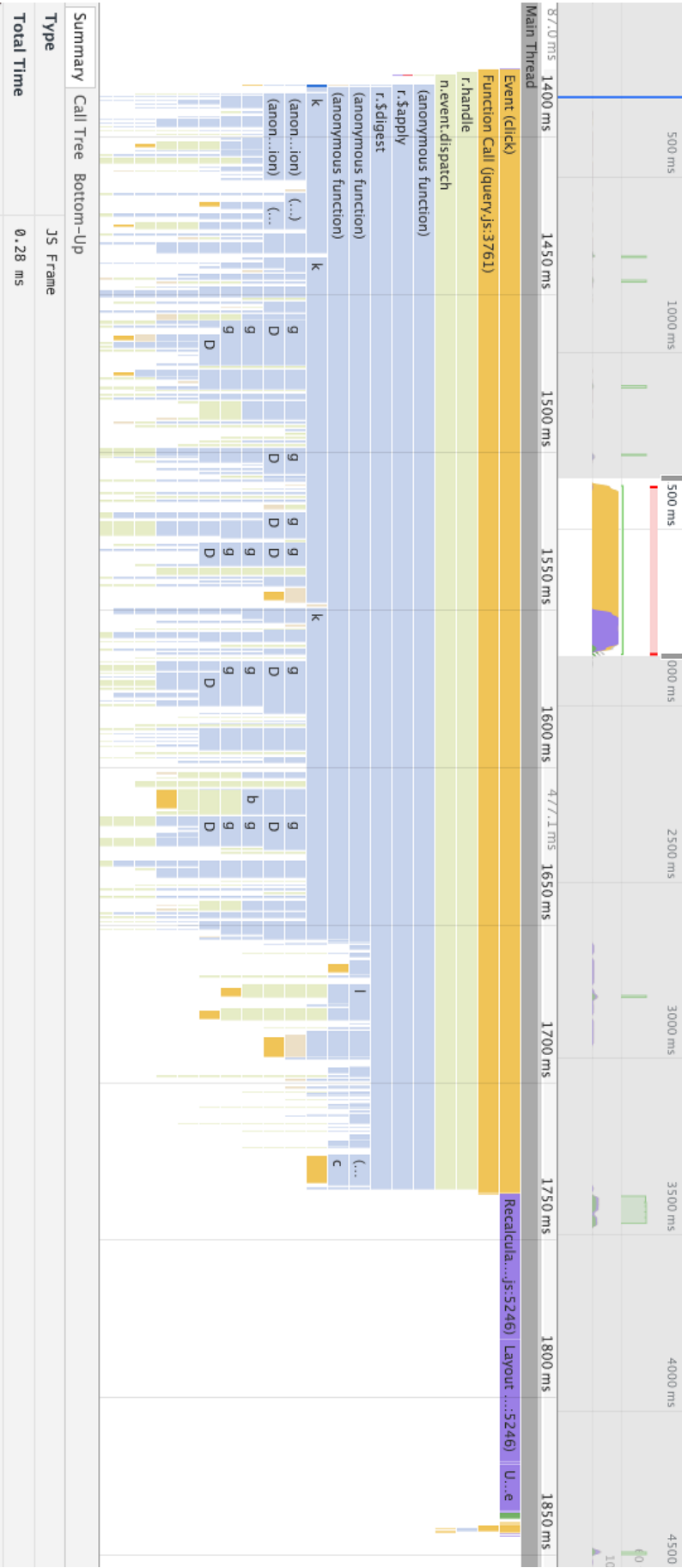


Figure 3.1: Google Chrome timeline (Krause, 2016a)

used the Selenium Webdriver and it can report the raw performance log entries from Chrome's timeline. It measures the duration from the start of a DOM click event to the end of the repainting of the DOM by parsing the performance log. To reduce sampling artifacts it takes the average of running each benchmark 12 times ignoring the two worst results. (Krause, 2016b)

The test cases are list below:

- Create rows: Duration for creating 1000 rows after the page loaded.
- Replace all rows: Duration for updating all 1000 rows of the table (with 5 warm-up iterations).
- Partial update: Time to update the text of every 10th row (with 5 warm-up iterations).
- Select row: Duration to highlight a row in response to a click on the row. (with 5 warm-up iterations).
- Swap rows: Time to swap 2 rows on a 1K table. (with 5 warm-up iterations).
- Remove row: Duration to remove a row. (with 5 warm-up iterations).
- Create many rows: Duration to create 10000 rows.
- Append rows to large table: Duration for adding 1000 rows on a table of 10000 rows.
- Clear rows: Duration to clear the table filled with 10000 rows.
- Startup time: Time for loading, parsing and starting up.





## 4. Angular

In 21st October 2010 , Google and a community of individuals and corporations released the first version of Angular, known as AngularJS. (AngularJS, 2010)

In 2014, the original AngularJS team began working on Angular. Angular, also known as Angular 2, is a TypeScript-based open-source front-end web application platform. Angular is a complete rewrite from the same team that built AngularJS. Its initial release was in 14th September 2014. (Angular, 2017b)

In this thesis, Angular is used to be investigated since it is the upgraded version of AngularJS. The key difference between AngularJS and Angular are listed below (Dotnet, 2016):

- AngularJS was not built with mobile support in mind, where Angular is mobile oriented.
- Angular provides more choice for languages. Any of the language from ES5, ES6, TypeScript or Dart can be used to write Angular code. Where, AngularJS has ES5, ES6 and Dart.
- AngularJS's controllers and scope are gone in Angular and replaced with "components". Angular is component based.
- AngularJS has 2-ways data binding but Angular has one-way data binding.
- Structural directives syntax is changed.
- To filter output in templates in AngularJS, pipe character " | " is used with one or more filters. Where in Angular they are called pipes. The syntax remains same.
- One of the major change in Angular is, that it directly uses the valid HTML DOM element properties and events. Due to this, many of the available built-in directives in AngularJS are now no longer required.

- In AngularJS, ng-bind is used for one way data binding, but with Angular it is replaced with [property], where property is valid HTML DOM element property.
- In AngularJS, ng-model is used for two-way data binding, but with Angular it is replaced with [(ngModel)].
- In AngularJS, we can define a service via 5 different ways – Factory, Service, Provider, Constant and Values – but in Angular 2, class is the only way to define a service.

## 4.1 Foundations

### 4.1.1 Modules

Angular applications are modular and Angular has its own modularity system called Angular modules or NgModules. An Angular module, whether a root or feature, is a class with an @NgModule decorator. NgModule is a decorator function that takes a single metadata object whose properties describe the module. The most important properties are:

- declarations
- exports
- imports
- providers
- bootstrap

#### Declarations

The property declarations expects an array of components, directives and pipes that are part of the module. All of them considered as view classes. (Angular, 2017a)

#### Imports and Exports

The property imports expects an array of modules which are other modules whose exported classes are needed by component templates declared in this module. Exports makes the components, directives, and pipes – which are the subset of declarations – available in modules that add this module to imports. (Angular, 2017a)

#### Providers

Providers are to make services and values known to dependency injection. They are added to the root scope and they are injected to other services or directives that have them as dependency. (Angular, 2017a)

## Bootstrap

The bootstrap property is where we define the root component of our module. Even though this property is also an array, 99% of the time we are going to define only one component which. Only the root module should set this bootstrap property. (Angular, 2017a)

### 4.1.2 Components

One of the core concepts of any Angular application is the component. In fact, each application can be formed as a tree of its components – figure 4.1. Essentially, a component is anything that is visible to the end user and which can be reused within an application. (Angular, 2017a)

In TypeScript, Metadata which tells Angular how to process a class, is attached by using a `@Component` decorator. In the `@Component` decorator, the values of different properties can be used to set the behavior of the component. The most used properties are as follows (Angular, 2017a):

- `template`
- `templateUrl`
- `directives`
- `providers`
- `styles`
- `styleUrls`

#### Template and templateUrl

As it mentioned in section 4.1.3, a template can be defined inline (using `template`) or in a separate file (using `templateUrl`). The complex view should be put in an external HTML file instead of an inline template. Angular allows to use an external HTML file as a template, and that can be linked to the component by setting the value of the `templateUrl` property. (Angular, 2017a)

#### Style and StyleUrls

A component can also have its own styles or it can refer to various other external style sheets beside the global style(s). Creating inline style is possible by setting the value of the `style` property. A component can refer to various external files as well by setting the value of the `styleUrls` property. (Angular, 2017a)

#### Providers

To use a service in a component, which can be an Angular service or a custom service, two steps needs to be followed (Angular, 2017a):

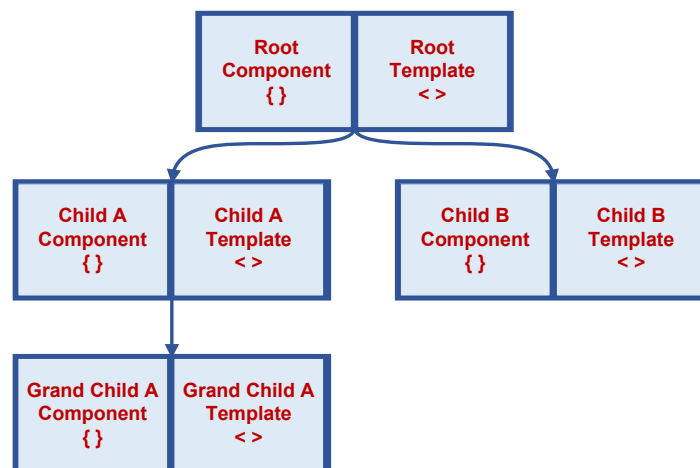


Figure 4.1: Parent/child components in Angular

- Importing the service in the component.
- Setting the value of the providers property. Providers accept an array and a list of services being used in the component can be passed to it.

## Directives

In order to have a nested component – in other word, using a component inside another component –, it is needed to set the value of the directives property of component metadata. Therefor, three steps must be followed (Angular, 2017a):

- Importing child component in the parent component.
- Using the selector of child component in the parent component's template.
- Setting value of directive property of parent component to the child component.

### 4.1.3 Templates

A component requires to have a view which is a template. A template can be defined inline or in a separate file. To simplify things, a template helps to render HTML with some dynamic parts depending on the data. It allows to express data and property binding, event binding and templating concerns which are explained in section 4.1.4. (Angular, 2017a)

### 4.1.4 Data binding

Data binding is a connection bridge between view and business logic of an application. Angular supports data binding, which is the automatic synchronization between Model and the View. When the Model changes, the Views are automatically updated and vice-versa. There are four forms of data binding in Angular (Angular, 2017a):

- Interpolation

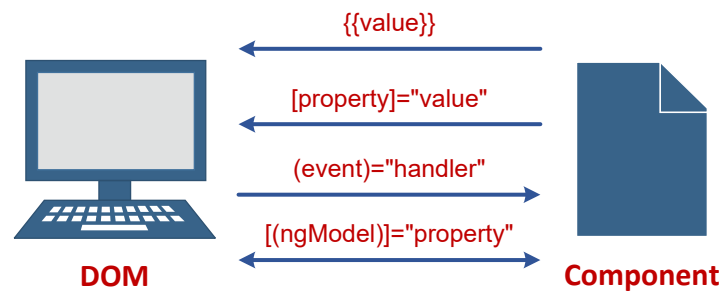


Figure 4.2: Data binding in Angular

- One-way binding
- Two-way binding
- Event binding

### Interpolation

Interpolation is the simplest way of data binding in Angular. In interpolation, as figure 4.2 shows, property name in the template must be enclosed in double curly braces, e.g. `{{value}}`. It is used for one-way binding (component to template only). (Angular, 2017a)

### One-way and two-way binding

Angular uses DOM element property for one-way binding. The square brackets, e.g. `[[property]]="value"`, are used with property name for one-way data binding in Angular. `[(ngModel)]` directive is used for two-way data binding in Angular. The `ngModel` directive is part of a built-in module called "FormsModule". Therefore, this module must be imported into the template module before using the `ngModel` directive. (Angular, 2017a)

Figure 4.2 also shows one-way and two-way data binding in Angular.

### Event binding

Angular directly uses the valid HTML DOM element events. For example, (click) and (keypress).

#### 4.1.5 Directives

Directives add behavior to an existing DOM element and make Angular templates dynamic. A directive is a class with a `@Directive` decorator. Angular components, section 4.1.2, are actually just directives under the hood. What makes them different from the other types of directives is components are directives that have a template. (Angular, 2017a)

The other two directive types do not have templates. Instead, their purpose is specifically tailored to two types of DOM manipulation. These two directive types are (Angular, 2017a):

- Structural directives
- Attribute directives

### Structural directives

Structural directives change the DOM layout by adding and removing DOM elements. They are prefixed with an asterisk "\*" when being used which warns Angular that the structure of the DOM elements within the directive may change depending on certain conditions. Two of the most common structural directives are "ngIf" and "ngFor". (Angular, 2017a)

### Attribute directives

Attribute directives change the appearance or behavior of an element. They are surrounded by brackets [] which signals to Angular that the appearance or behavior of the DOM elements within the attribute directive may change depending on certain conditions. The most common attribute directive is the "hidden" directive. (Angular, 2017a)

## 4.1.6 Services

Service is a broad category encompassing any value, function, or feature that your application needs. Almost anything can be a service. A service is typically a class with a narrow, well-defined purpose. It should do something specific and do it well. There is nothing specifically Angular about services. Angular has no definition of a service. There is no service base class, and no place to register a service. Yet services are fundamental to any Angular application. Components are big consumers of services. Component classes should be lean. They don't fetch data from the server, validate user input, or log directly to the console. They delegate such tasks to services. A component's job is to enable the user experience and nothing more. It mediates between the view (rendered by the template) and the application logic (which often includes some notion of a model). A good component presents properties and methods for data binding. It delegates everything nontrivial to services. However, Angular doesn't enforce these principles. (Angular, 2017a)

## 4.2 Features

In this section, we take a quick look at Angular features and benefits as they mentioned in their official website (Angular, 2017a):

- Progressive web apps: Use modern web platform capabilities to deliver app-like experiences. High performance, offline, and zero-step installation.
- Native: Build native mobile apps with strategies from Ionic Framework, NativeScript, and React Native.
- Desktop: Create desktop-installed apps across Mac, Windows, and Linux using the

same Angular methods you've learned for the web plus the ability to access native OS APIs.

- **Templates:** Quickly create UI views with simple and powerful template syntax.
- **Angular CLI:** Command line tools start building fast, add components and tests, then instantly deploy.
- **IDEs:** Get intelligent code completion, instant errors, and other feedback in popular editors and IDEs.
- **Testing:** With Karma for unit tests, you can know if you've broken things every time you save. And Protractor makes your scenario tests run faster and in a stable manner.

In chapter 7, they are compared with the other two frameworks.





## 5. React

### 5.1 Foundations

#### 5.1.1 JSX

JSX is a syntax extension to JavaScript and it is basically a preprocessor step that adds XML syntax to JavaScript. It is recommended using it with React to describe what the UI should look like. JSX produces React "elements". Any JavaScript expression can be embedded in JSX by wrapping in curly braces and after compilation, JSX expressions become regular JavaScript objects. This means that JSX can be used inside of if statements and for loops, be assigned to variables, be accepted as arguments, and be returned from functions. By default, React DOM escapes any values embedded in JSX before rendering them. Thus it ensures that anything can never be injected that's not explicitly written in the application. Everything is converted to a string before being rendered. This helps prevent XSS attacks. (React, 2017a)

#### 5.1.2 Elements

React elements are objects and React reads and uses them to construct the DOM and keep it up to date. Elements can also represent user-defined components. They are the smallest building blocks of React apps. An element describes what you want to see on the screen. Unlike browser DOM elements, React elements are plain objects, and are cheap to create. React DOM takes care of updating the DOM to match the React elements. React elements are immutable. Once an element is created, its children or attributes cannot be changed. An element is like a single frame in a movie: it represents the UI at a certain point in time. React DOM compares the element and its children to the previous one, and only applies

the DOM updates necessary to bring the DOM to the desired state. (React, 2017a)

### 5.1.3 Components

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation. Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen. There exist two type of components. Functional components are those which are defined by writing a JavaScript function. Class components are defined using ES6 classes. (React, 2017a)

A functional component can be converted to a class in five steps (React, 2017a):

1. Creating an ES6 class with the same name that extends `React.Component`.
2. Adding a single empty method to it called `render()`.
3. Moving the body of the function into the `render()` method.
4. Replacing props with `this.props` in the `render()` body.
5. Deleting the remaining empty function declaration. (React, 2017a)

### 5.1.4 Props

Props are available in the component as `"this.props"` and can be used in the render method to render dynamic data. They are Read-Only. Whether a component is declared as a function or a class, it must never modify its own props. (Wheeler, 2014)

### 5.1.5 State

State is similar to props, but it is private and fully controlled by the component. By converting a function to a class, additional features such as local state and lifecycle hooks can be used. (React, 2017a)

State is set using the `"setState"` method. Calling `"setState"` triggers UI updates and is the bread and butter of React's interactivity. If one want to set an initial state before any interaction occurs the `"getInitialState"` method can be used. (Wheeler, 2014)

### 5.1.6 Events

React also has a built in cross browser events system. The events are attached as properties of components and can trigger methods. Handling events with React elements is very similar to handling events on DOM elements. (Wheeler, 2014)

### 5.1.7 Data binding

In React, application data flows unidirectionally via the state and props objects, as opposed to the two-way binding of libraries like AngularJS. This means that, in a multi component hierarchy, a common parent component should manage the state and pass it down the chain via props. A state should be updated using the "setState" method to ensure that a UI refresh will occur, if necessary. The resulting values should be passed down to child components using attributes that are accessible in said children via this.props. (Wheeler, 2014)

## 5.2 Features

Here below, some of the React's features which are mentioned in their official website are listed (React, 2017a):

- **Composition:** The key feature of React is composition of components. Components written by different people should work well together.
- **Escape Hatches:** React is pragmatic. It is driven by the needs of the products written at Facebook. While it is influenced by some paradigms that are not yet fully mainstream such as functional programming, staying accessible to a wide range of developers with different skills and experience levels is an explicit goal of the project.
- **Stability:** React values API stability. At Facebook, they have more than 20 thousand components using React. Many other companies, including Twitter and Airbnb, are also heavy users of React. This is why they are usually reluctant to change public APIs or behavior.

In chapter 7, they are compared with the other two frameworks.



## 6. Ember

### 6.1 Foundations

#### 6.1.1 Ember Object

Ember has its own object system. The base object is "Ember.Object" and all the other objects in are extensions of "Ember.Object". Mostly extensions of "Ember.Object" – like "Ember.Controller" and "Ember.View" – are being used but using "Ember.Object" itself is also possible. One its usage is for creating service objects that handle some specific logic. (Ember, 2017a)

Objects can have three types of things inside them (Ember, 2017a):

- properties which can be basic or computed. Basic properties accept number, boolean, string, etc. Computed properties actually do some work and call other properties.
- functions
- observers which are functions that fire whenever anythings they observe change.

#### 6.1.2 Router and route

Everything in Ember starts with routes. By default the "hashchange" event is being used in the browser to know when routes are changed. It implements its own "HashLocation" object to handle this. With "HashLocation", an Ember route will be visible after the "#" in the URL. For example, "http://my-ember-app.com/#!/about".

"Router" and "Route" are not the same. The "Router" creates named URL routes but a

"Route" object is a specific type of Ember object that helps setting up and managing what happens when someone visit that URL route. (Ember, 2017a)

### 6.1.3 Templates

Ember templates are simply Handlebars files. Handlebars is a logic-less templating engine that dynamically generates HTML page. It is an extension of Mustache with a few additional features. Mustache is fully logic-less but Handlebars adds minimal logic thanks to the use of some helpers (such as "if", "with", "unless", "each" and more). As a matter of fact, Handlebars is a superset of Mustache. (Kumar, 2015)

### 6.1.4 Components

A component is a custom HTML tag whose behavior is implemented using JavaScript and whose appearance is described using Handlebars templates. They allow you to create reusable controls that can simplify your application's templates. (Ember, 2017a)

### 6.1.5 Controllers

Controllers behave like a specialized type of Component that is rendered by the router when entering a Route. Ember provides three types of controllers (Ember, 2017a):

- `ObjectController` which is used whenever that controller's route fetches a single model.
- `ArrayController` which is used when the route fetches an array of models.
- `Controller` which is used when the route is not fetching any models at all.

### 6.1.6 Views

The View is one of the most powerful objects in Ember. A view can be considered as a wrapper for a template. It contains all the JavaScript that might be wanted to execute on the template and manages the logic around attributes and class names. (Ember, 2017a)

## 6.2 Features

Ember provides developers both with many features that are essential to manage complexity in modern web applications, as well as an integrated development toolkit that enables rapid iteration.

Some of these features that mentioned in their official website are (Ember, 2017a):

- **Ember CLI:** A robust development toolkit to create, develop, and build Ember

applications.

- Routing: The central part of an Ember application. Enables developers to drive the application state from the URL.
- Templating engine: Use Handlebars syntax to write your application's templates.

These features are compared with the other two frameworks in chapter 7.





## 7. Analysis

### 7.1 Overview

#### 7.1.1 Size

The JavaScript code is going to have to be downloaded before the application is fully functional. The bigger the code, the longer it takes the application to start up. That affects the perceived responsiveness, which is a big deal. The downloads also affect bandwidth costs. These are important considerations. (Coulman, 2015)

But the size of the underlying framework is not the only thing that matters. It is important to consider the size of the application code as well. For a big enough application, that size should dominate the framework size. (Coulman, 2015)

The framework can have a large impact on the size of the application code. A full-featured framework should provide tools that allow us to write a lot less of our own application code. (Coulman, 2015)

Figure 7.1, is the output from "ls" command. All the files are downloaded from cdnjs.com. This gives the approximations in table 7.1. Note that these files and their sizes are independent from any OS.

It is obvious that React (+ React DOM) is the most lightweight framework and Angular (+ Rx) is the most heavyweight.

```

Kavoos@MSI MINGW32 /d/Documents/HoGent/Chamilo/Bachelorproef/Thesis/Frameworks
$ ls -l
total 1.4M
-rw-r--r-- 1 Kavoos 197121 622K May  1 11:49 angular2.min.js
-rw-r--r-- 1 Kavoos 197121 424K May  1 11:28 ember.min.js
-rw-r--r-- 1 Kavoos 197121 223K May  1 11:50 Rx.min.js
-rw-r--r-- 1 Kavoos 197121 121K May  1 11:32 react-dom.min.js
-rw-r--r-- 1 Kavoos 197121  21K May  1 11:32 react.min.js

```

Figure 7.1: Downloaded frameworks from <https://cdnjs.com>

	Size (KB)
Angular (+ Rx)	845
Ember	424
React (+ React DOM)	142

Table 7.1: Framework sizes (in KB)

### 7.1.2 Get started and configuration

Creating a new application using all three frameworks goes very smooth. As figure 7.2 shows, with just two lines of command, it is possible to get started. Figure 7.3 shows the created application using these commands.

By looking at these applications skeleton, it is obvious that Angular, figure 7.3a, tries to have a folder structure based on the application components. For example, there is "app" folder and inside this folder, we can see all the related files with this component

- app.component.css
- app.component.html
- app.component.ts
- app.module.ts

Ember, figure 7.3c, on the other hand wants to guide the user to have a folder structure based on the functionality of each files. It creates empty folders and tries to have a kind of pre-refactored application. These folder are:

- components
- controllers
- helpers
- models
- routes
- styles
- templates

And React, figure 7.3b, simply does not create any prepared folder structure.

All three applications are already configured and the user does not have to configure anything to run the created applications. For running each application, we can use these

<pre>&gt; npm install -g @angular/cli &gt; ng new my-angular-app  &gt; cd my-angular-app &gt; ng serve</pre>	<pre>&gt; npm install -g create-react-app &gt; create-react-app my-react-app  &gt; cd my-react-app &gt; npm start</pre>	<pre>&gt; npm install -g ember-cli &gt; ember new my-ember-app  &gt; cd my-ember-app &gt; ember server</pre>
(a) Angular	(b) React	(c) Ember

Figure 7.2: Required commands to get an application started

commands in the command prompt:

- For Angular, ng serve – figure 7.2a
- For React, npm start – figure 7.2b
- For Ember, ember server – figure 7.2c

## 7.2 Foundations

### 7.2.1 Creating components and views

Figures 7.4 and 7.5 show a sample component – namely, Home component – and its view (template). Component files are not more than JavaScript/TypeScript classes. In React, figure 7.5b, template takes place in the render() function, inside the JavaScript component class.

However in Angular and Ember, figures 7.5a and s 7.5c, templates take place in separate file. In Angular, it is also possible to have a template inside the component.

### 7.2.2 Template syntax

The template syntax of Angular, React and Ember are completely different from each other. Figure 7.6 shows a simple for-loop example in each framework. React, figure 7.6b, has a syntax which is more like a mix of HTML and JavaScript code. Angular, figure 7.6a, is look less complicated than the other two.

### 7.2.3 Data binding

Data binding is the mechanism that connects a data model to the UI. One-way data binding creates an ongoing connection between the model and the UI. Changes to the model are reflected in the UI through some kind of process controlled by the framework. Two-way includes one-way, but it also facilitates the flow of data from the UI to the model. (Greene, 2016)

Angular, as opposed to AngularJS, does not come with such a (built-in) two-way data binding anymore. However, this does not mean directives that support two-way data binding can not be created. There is one directive in Angular, namely ngModel, that

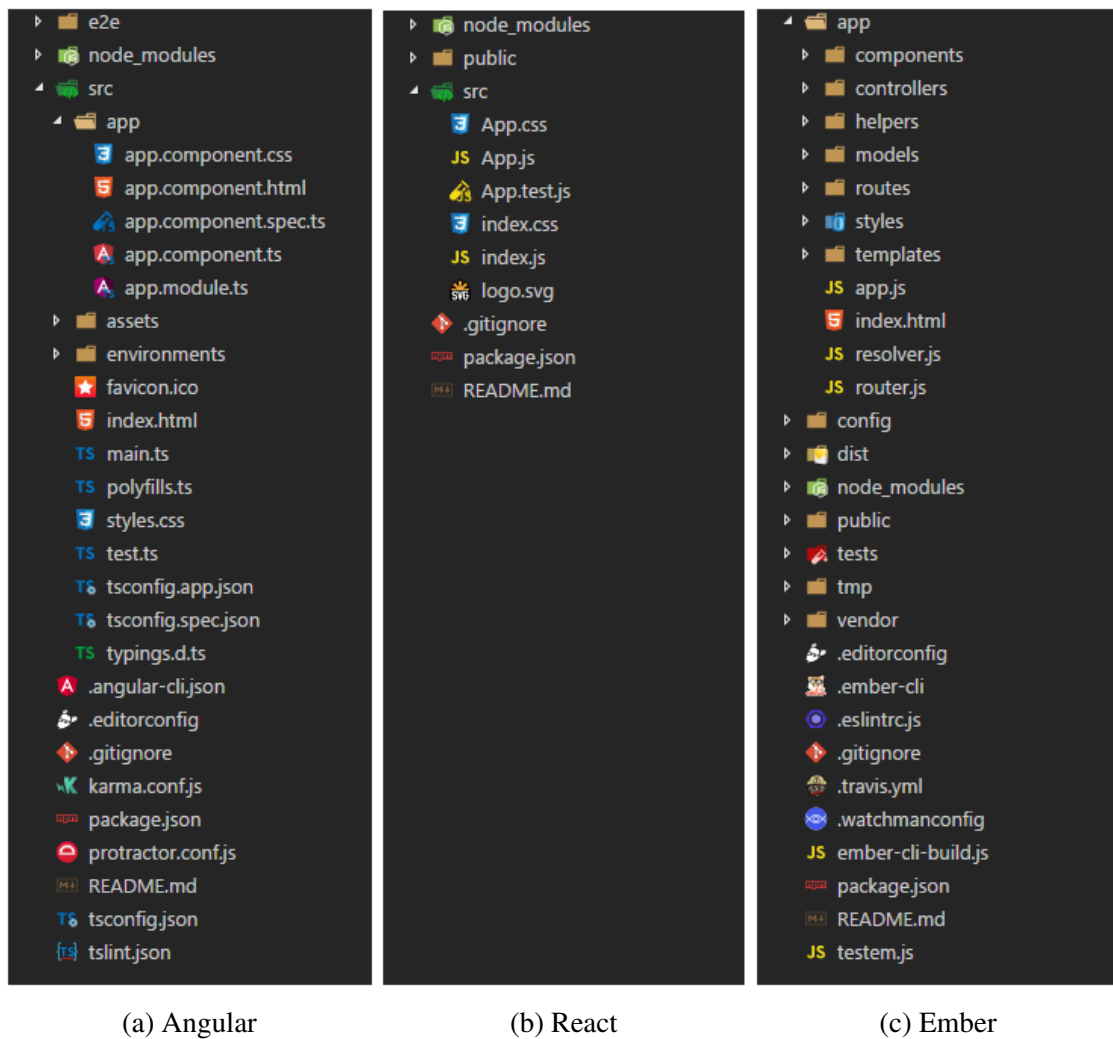


Figure 7.3: Basic application skeleton

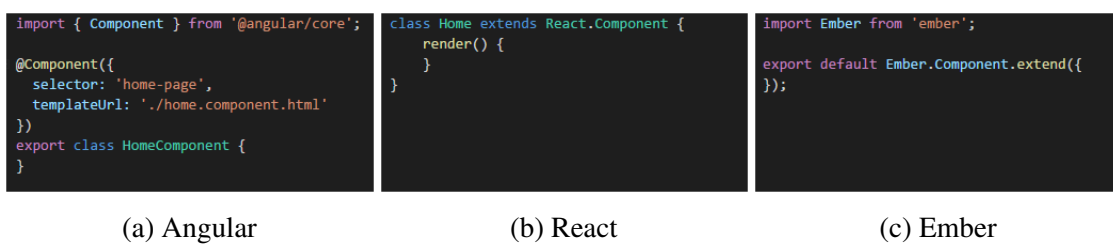
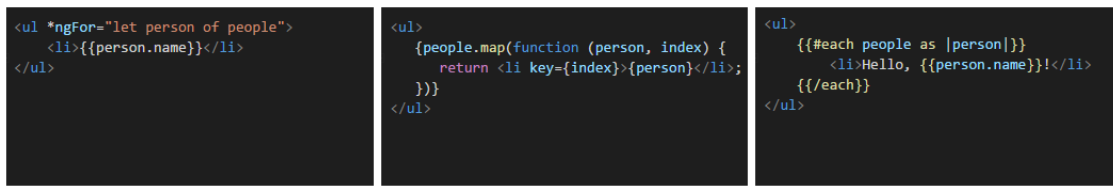


Figure 7.4: A simple component in each framework



Figure 7.5: A simple template in each framework



The figure displays three code snippets side-by-side, each representing a different framework's syntax for iterating over an array. (a) Angular uses the `*ngFor` directive. (b) React uses the `map` function with a callback that returns JSX elements. (c) Ember uses the `each` helper function.

(a) Angular

(b) React

(c) Ember

Figure 7.6: A for-loop example in each framework

implements two-way data binding. (Precht, 2016)

Ember supports two-way data binding. Data on HTML and JavaScript is automatically updated each other. Ember introduced "`{{}}`" syntax where variables can be written to bind to JavaScript model. (Tsuneo, 2015)

React is a framework specialized for data binding, it also simplify data binding utilizing Virtual DOM. React automatically detect how the DOM is updated without explicitly specifying how to update, and only the difference is applied to DOM. On code, React embed HTML representation on JavaScript code using JSX syntax. Embedded HTML is not directly updated to DOM on HTML, but stored as Virtual DOM. When we need to change state, we use `setState()` function. Then Reacts detect the difference on Virtual DOM and update state difference to HTML. React have smaller performance impact because React does not directly manipulate DOM in HTML but manipulate Virtual DOM and only update the changing. (Tsuneo, 2015)

## 7.2.4 Routing

Angular brings many improved modules to the Angular ecosystem including a new router called the Component Router. The Component Router is a highly configurable and feature packed router. Features included are standard view routing, nested child routes, named routes, and route parameters. (Rylan, 2016)

## 7.3 Features

In this section, we compared features of each frameworks that are mentioned in sections 4.2, 5.2 and 6.2 with the other frameworks.

### 7.3.1 Angular

#### Progressive web apps

*"Uses modern web platform capabilities to deliver app-like experiences. High performance, off-line, and zero-step installation."*

```

import { ModuleWithProviders } from '@angular/core';
import { RouterModule } from '@angular/router';

const articleRouting: ModuleWithProviders = RouterModule.forChild([
  {
    path: 'article/:id',
    component: ArticleComponent
  }
]);

```

```

import { Link } from 'react-router';
import React from 'react';

const ArticleActions = props => {
  return (
    <Link to={`/editor/${article.id}`} className="btn">
      Edit Article
    </Link>
  );
};

```

```

import Ember from 'ember';

const Router = Ember.Router
  .extend({ location: 'hash', rootURL: '/' });

Router.map(function () {
  this.route('article', { path: '/article/:id' });
});

export default Router;

```

(a) Angular

(b) React

(c) Ember

Figure 7.7: Defining routing in each framework

Both React and Ember also have zero-step installations. React performance is almost like Angular, but Ember still needs to be improved in terms of performances. Also both React and Ember can be used off-line thanks to different dependency management tools like Bower and Composer.

## Native

*“Build native mobile apps with strategies from Ionic Framework, NativeScript, and React Native.”*

React has React Native and Ember has Ember-Cordova which support creating native mobile applications.

## Desktop

*“Create desktop-installed apps across Mac, Windows, and Linux using the same Angular methods you’ve learned for the web plus the ability to access native OS APIs.”*

There are also different libraries and adds-on for creating React and Ember desktop applications such as React Desktop and NW.js.

## Templates

*“Quickly create UI views with simple and powerful template syntax.”*

Templates in all three language are almost the same. However since React uses JSX, it takes places in JavaScript files. Ember on the other hand uses separate files for its templates. They all support logics like conditional statements and for loops.

## Angular CLI

*“Command line tools start building fast, add components and tests, then instantly deploy.”*

Ember has a command line tools, like Angular, and it can be used to create different parts of an application like components. It also be used to build an application without any configuration. React uses Create React App which is used to build an application but not more.

## IDEs

*“Get intelligent code completion, instant errors, and other feedback in popular editors and IDEs.”*

Since both React and Ember use JavaScript and it is an interpreted language, they can easily use any text editor like Visual Studio Code, Atom and Sublime.

## Testing

*“With Karma for unit tests, you can know if you’ve broken things every time you save. And Protractor makes your scenario tests run faster and in a stable manner.”*

Both React and Ember are testable. QUnit used for Ember and Jest used for React.

### 7.3.2 React

#### Composition

*“The key feature of React is composition of components. Components written by different people should work well together.”*

Angular and Ember are both composition of components as React.

#### Escape Hatches

*“React is pragmatic. It is driven by the needs of the products written at Facebook. While it is influenced by some paradigms that are not yet fully mainstream such as functional programming, staying accessible to a wide range of developers with different skills and experience levels is an explicit goal of the project.”*

Angular and Ember did not made because of particular company need them and it is one the differences between React and the other two frameworks. However it does not mean that Angular and Ember are not pragmatic.

## Stability

*“React values API stability. At Facebook, they have more than 20 thousand components using React. Many other companies, including Twitter and Airbnb, are also heavy users of React. This is why they are usually reluctant to change public APIs or behavior.”*

All frameworks have a stable release versions. React released in 2013. Ember is even older than React and it released in 2011. Angular on the other hand is quite young, however, Angular team has about 6 years experience with AngularJS since AngularJS release in 2010.

### 7.3.3 Ember

#### Ember CLI

*“A robust development toolkit to create, develop, and build Ember applications.”*

Angular has a command line tools, like Ember, and it can be used to create different parts of an application like components. It also be used to build an application without any configuration. React uses Create React App which is used to build an application but not more.

#### Routing

*“The central part of an Ember application. Enables developers to drive the application state from the URL.”*

Routing being a key aspect of any web applications and could not be left out in React and Angular. All three frameworks have Routing which drive the application state from the URL.

#### Templating engine

*“Use Handlebars syntax to write your application’s templates.”*

Handlebars is a logic-less and forcing its users to separate presentation from logic. Angular is not logic-less which can be considered as an advantage or disadvantage. React uses JSX. It basically has its templates inside the JavaScript files.



## 7.4 Performance

### 7.4.1 Speed

As previously explained in chapter 3, few test cases are executed in order to find out which framework is faster than the other ones. The test cases are list below:

- Create rows: Duration for creating 1000 rows after the page loaded.
- Replace all rows: Duration for updating all 1000 rows of the table (with 5 warm-up iterations).
- Partial update: Time to update the text of every 10th row (with 5 warm-up iterations).
- Select row: Duration to highlight a row in response to a click on the row. (with 5 warm-up iterations).
- Swap rows: Time to swap 2 rows on a 1K table. (with 5 warm-up iterations).
- Remove row: Duration to remove a row. (with 5 warm-up iterations).
- Create many rows: Duration to create 10000 rows.
- Append rows to large table: Duration for adding 1000 rows on a table of 10000 rows.
- Clear rows: Duration to clear the table filled with 10000 rows.
- Startup time: Time for loading, parsing and starting up.

Each benchmark is executed 12 times ignoring the two worst results to reduce sampling artifacts.

Figure 7.8 and table 7.2 provide the results. Obviously, the lower the execution time, the better the performance of the framework.

The results show clearly that Ember is not as fast as Angular and React. Angular and React compete almost neck and neck. Except partial update, selecting, swapping and clearing rows in which Angular is the fastest, React is slightly faster in other test cases.

### 7.4.2 Memory footprint

Figure 7.9 and table 7.3 shows the memory consumption directly after loading the page and when 1000 rows are added to the table.

Here is the competition also between Angular and React and, as a result, React is the winner in both test cases.

## 7.5 Popularity, maturity and community

Figure 7.10 shows Google Trends' interest over time (web search, worldwide, 2011 - 2017). Numbers represent search interest relative to the highest point on the chart for the given region – in this case, worldwide – and time. A value of 100 is the peak popularity for the term. A value of 50 means that the term is half as popular. Likewise a score of 0

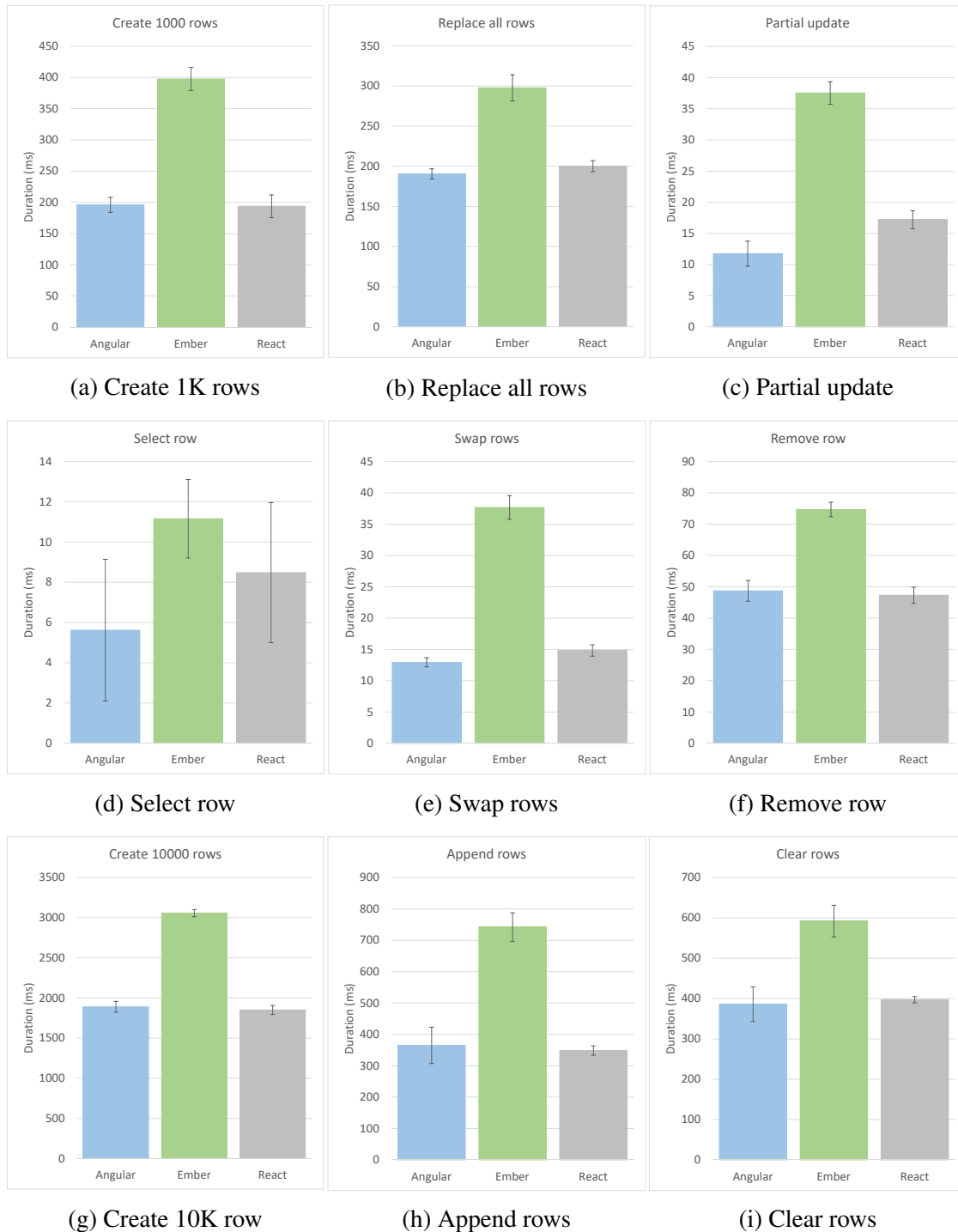


Figure 7.8: Benchmark results for duration tests (in milliseconds)

	Angular	Ember	React
Create rows Duration for creating 1000 rows after the page loaded	195.99 ±12.00 1.01	397.56 ±18.42 2.05	193.77 ±18.18 1.00
Replace all rows Duration for updating all 1000 rows of the table (with 5 warm-up iterations)	190.46 ±6.39 1.00	297.72 ±16.38 1.56	200.21 ±6.83 1.05
Partial update Time to update the text of every 10th row (with 5 warm-up iterations)	11.76 ±2.01 1.00	37.54 ±1.81 3.19	17.21 ±1.45 1.46
Select row Duration to highlight a row in response to a click on the row (with 5 warm-up iterations)	5.62 ±3.52 1.00	11.16 ±1.95 1.99	8.48 ±3.48 1.51
Swap rows Time to swap 2 rows on a 1K table (with 5 warm-up iterations)	12.94 ±0.73 1.00	37.67 ±1.90 2.91	14.84 ±0.91 1.15
Remove row Duration to remove a row (with 5 warm-up iterations)	48.70 ±3.32 1.03	74.70 ±2.31 1.58	47.29 ±2.57 1.00
Create 10000 rows Duration to create 10,000 rows	1889.79 ±68.02 1.02	3056.59 ±45.07 1.65	1850.82 ±56.20 1.00
Append rows to large table Duration for adding 1K rows on a table of 10K rows	365.08 ±57.92 1.05	741.82 ±45.75 2.13	348.60 ±14.84 1.00
Clear rows Duration to clear the table filled with 10.000 rows	386.15 ±42.85 1.00	592.33 ±39.19 1.53	397.17 ±7.70 1.03
Startup time Time for loading, parsing and starting up	102.21 ±2.41 1.09	214.52 ±3.27 2.29	93.88 ±3.34 1.00
Slowdown geometric means	1.02	2.02	1.11

Table 7.2: Benchmark results for duration tests (in milliseconds)

	Angular	Ember	React
Ready memory Memory usage after page load	6.88 ±0.23 1.20	10.21 ±0.24 1.78	5.73 ±0.15 1.00
Run memory Memory usage after adding 1000 rows	14.32 ±0.5 1.10	28.02 ±0.66 2.16	12.98 ±0.75 1.00

Table 7.3: Benchmark results for memory allocation (in MB)

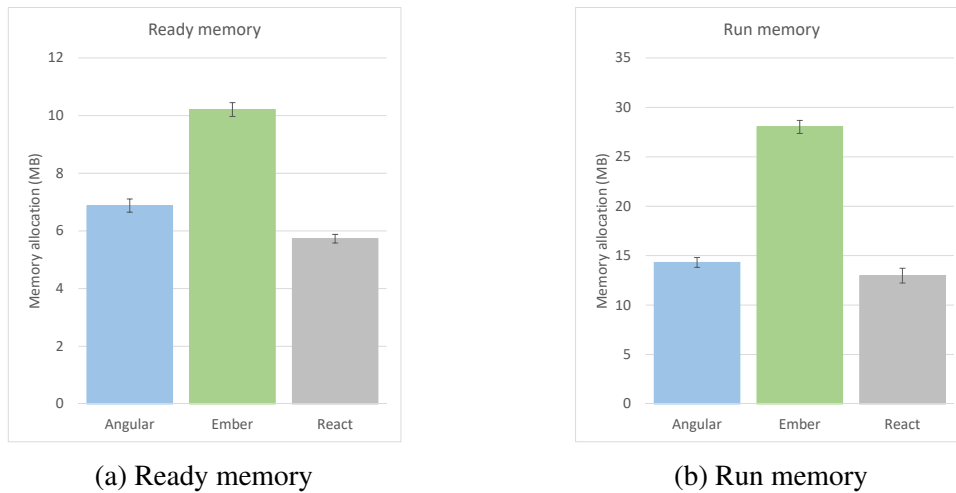


Figure 7.9: Benchmark results for memory allocation (in MB)

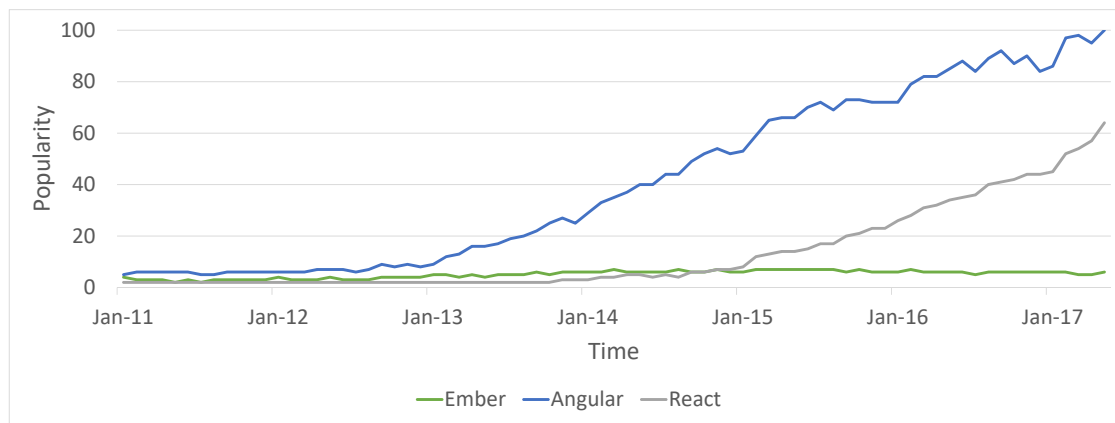


Figure 7.10: Google Trends' interest over time

means the term was less than 1% as popular as the peak. Note that it is not possible to divide AngularJS and Angular search terms. Therefore result should be considered for both Angular and AngularJS.

As table 7.4 indicates, Ember is older among these three frameworks and Angular is the youngest one.

Table 7.4 also shows the number of Github stars, contributors, open and closed issues.

Many of GitHub's repository rankings depend on the number of stars a repository has. Starring a repository allows to keep track of projects which are interesting, even if someone is not associated with the project and it performing two distinct actions (Github, 2017):

- Creating a bookmark for easier access.
- Showing appreciation to the repository maintainer for their work

Issues in Github are suggested improvements, tasks or questions related to the repository.

	Angular	React	Ember
First release	2016	2013	2011
Github stars	24261	67201	17870
Github contributors	441	1005	669
Github open issues	1286	592	235
Github closed issues	9142	3842	5006
StackOverflow questions	51627	42575	21222

Table 7.4: Frameworks in Github and StackOverflow

Issues can be created by anyone (for public repositories), and are moderated by repository collaborators. Each issue contains its own discussion forum, can be labeled and assigned to a user. A contributor is someone who has contributed to a project by having a pull request merged but does not have collaborator access. (Github, 2017)



## 8. Conclusion

This thesis, as its research question, is investigated what aspects are important in the selection between Angular, React and Ember. In order to provide a suitable answer, we broke down the research questions into several sub-questions:

- What do Angular, Ember and React have in common?
- What are the most important differences between Angular, Ember and React?
- Which framework has the highest performances (in terms of speed and memory footprint)?
- Is it possible to say if one of these three frameworks is more suitable for a specific project?
- Which framework is more popular (based on Google Trends, StackOverflow questions, their Github stars, contributors, open and closed issues)?

First, we provide an answer for each sub-question and after that we are going to have a conclusion which is the answer to our research question.

### Similarities

*“What do Angular, Ember and React have in common?”*

Angular, Ember and React are MV\* JavaScript frameworks. They are all open-source and component based. They do not need any configuration after getting started with them.

Navigating to different pages in a single-page application, with no page reloading, is possible via routing and the URL. All these frameworks support routing.

## Differences

*“What are the most important differences between Angular, Ember and React?”*

Ember uses Handlebars syntax for its templates which is logic-less. It also has its own object system, namely "Ember.Object". React introduced JSX which is a syntax extension to JavaScript and it is basically a preprocessor step that adds XML syntax to JavaScript. Angular uses TypeScript which is a superset of JavaScript. Both Angular and Ember have powerful CLIs but React does not have any yet.

## Performances

*“Which framework has the highest performances (in terms of speed and memory foot-print)?”*

React and Angular are in the same level in terms of speed. However Ember is around two times slower than the other ones. In terms of memory foot-print, again Angular and React have the same level and Ember consumes more memory.

## Expertise

*“Is it possible to say if one of these three frameworks is more suitable for a specific project?”*

None of these frameworks claim that they are suitable for a specific kind of project. Even though, they have some differences with each other, they have the ability of covering everything that needed to be done in a project.

## Popularity, maturity and community

*“Which framework is more popular (based on Google Trends, StackOverflow questions, their Github stars, contributors, open and closed issues)?”*

React has the more than 67000 Github stars, which is almost three times than the other ones. So it more popular. React has also more contributors than Angular and Ember. However, Angular in its one year lifetime, earned more than 24000 Github stars and it shows that Angular is becoming a serious competitor of React in the next years.

Open and closed issues of Angular is more than React and Ember. It shows that its improvement is growing very fast. But it also shows that Angular is not as mature as React and Ember.

Besides, figure 7.10 shows that Angular is more searched in Google. But we should note that it is not possible to divide AngularJS and Angular search terms. Therefor we consider this result for both Angular and AngularJS.



**Research question**

*“What aspects are important in the selection between Angular, React and Ember”*

The first aspect which is important in selection between these three frameworks is templating. The rendered view of all these framework could be exactly the same, but a developer should know that React uses JSX which is completely different from templating in Angular and Ember. Even though Handlebars in Ember is close to templates in Angular, but it is almost logic-less and the developer could not bring his or her logics into the templates.

The second aspect is performance. Ember is not in the same level as Angular and React. It is almost two times slower and it takes more memory.

CLI is the next important aspect, specially for beginners. Angular and Ember, both have powerful CLIs which could build a project with a build-in folder structure.

And final import aspect is having a big and active community. Finding good tutorials, solutions to problems in StackOverflow are not things that a developer wants to avoid. React and specially Angular are very successful in this part.



# Bibliography

- Agarwal, R. (2016). Choosing Right Javascript Development Framework: Angular vs React vs Ember. **retrieved** 30 April 2017, **from** <http://www.algoworks.com/blog/choosing-right-javascript-framework-angularjs-vs-react-vs-ember/>
- Alexiou, A. (2017). JavaScript. **retrieved** 28 May 2017, **from** <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- Angular. (2017a). Architecture Overview. **retrieved** 17 April 2017, **from** <https://angular.io/docs/ts/latest/guide/architecture.html>
- Angular. (2017b). Releases. **retrieved** 28 May 2017, **from** <https://github.com/angular/angular/releases?after=4.0.3>
- AngularJS. (2010). Releases. **retrieved** 30 May 2017, **from** <https://github.com/angular/angular.js/releases>
- AngularJS. (2017). Unit Testing. **retrieved** 28 May 2017, **from** <https://docs.angularjs.org/guide/unit-testing>
- Bailey, D. (2011). Backbone.js Is Not An MVC Framework. **retrieved** 28 April 2017, **from** <https://lostechies.com/derickbailey/2011/12/23/backbone-js-is-not-an-mvc-framework/>
- Bracha, G. (2015). *The Dart Programming Language*.
- Coulman, R. (2015). JavaScript Framework Size. **retrieved** 13 May 2017, **from** <http://randycoulman.com/blog/2015/02/17/javascript-framework-size/>
- Cullocca, F. (2016). First-class Function. **retrieved** 28 May 2017, **from** [https://developer.mozilla.org/en-US/docs/Glossary/First-class\\_Function](https://developer.mozilla.org/en-US/docs/Glossary/First-class_Function)
- Detectify. (2015). What is Cross-site Scripting and how can you fix it? **retrieved** 20 April 2017, **from** <https://blog.detectify.com/2015/12/16/what-is-cross-site-scripting-and-how-can-you-fix-it/>
- Dotnet, T. (2016). Difference between Angular 1.x and Angular 2. **retrieved** 15 April 2017, **from** <http://www.talkingdotnet.com/difference-between-angular-1-x-and-angular-2/>

- Elliott, E. (2016). Angular 2 vs React: The Ultimate Dance Off. **retrieved** 1 May 2017, **from** <https://medium.com/javascript-scene/angular-2-vs-react-the-ultimate-dance-off-60e7dfbc379c>
- Ember. (2017a). Guides and Tutorials. **retrieved** 29 April 2017, **from** <https://guides.emberjs.com/v2.12.0/>
- Ember. (2017b). Releases. **retrieved** 28 May 2017, **from** <https://github.com/emberjs/ember.js/releases>
- Github. (2017). About stars. **retrieved** 31 May 2017, **from** <https://help.github.com/articles/>
- Greene, E. (2016). Two-Way Data Binding: Angular 2 and React. **retrieved** 14 May 2017, **from** <https://www.accelebrate.com/blog/two-way-data-binding-angular-2-and-react/>
- Hope, C. (2017). Framework. **retrieved** 29 May 2017, **from** <https://www.computerhope.com/jargon/f/framework.htm>
- JavaScripting. (2017). The definitive source of the best JavaScript libraries, frameworks, and plugins. **retrieved** 20 May 2017, **from** <https://www.javascripting.com/application-tools/frameworks/?sort=rating>
- Katz, Y. (2017). About. **retrieved** 28 May 2017, **from** <http://yehudakatz.com/about/>
- Krause, S. (2016a). Benchmarking JS-Frontend Frameworks. **retrieved** 28 April 2017, **from** <http://www.stefankrause.net/wp/?p=218>
- Krause, S. (2016b). JS web frameworks benchmark – Round 1. **retrieved** 28 April 2017, **from** <http://www.stefankrause.net/wp/?p=191>
- Kuhn, Z. (2015). Choosing a Front End Framework: Angular vs. Ember vs. React. **retrieved** 30 April 2017, **from** <https://smashingboxes.com/blog/choosing-a-front-end-framework-angular-ember-react/>
- Kumar, R. (2015). A Beginner's Guide to Handlebars. **retrieved** 31 May 2017, **from** <https://www.sitepoint.com/a-beginners-guide-to-handlebars/>
- Laplante, P. A. (2007). *What Every Engineer Should Know about Software Engineering*.
- Laverdet, M. (2010). XHP: A New Way to Write PHP. **retrieved** 30 April 2017, **from** <https://www.facebook.com/notes/facebook-engineering/xhp-a-new-way-to-write-php/294003943919/>
- Muhammad. (2017). React vs Angular 2: Comparison Guide for Beginners. **retrieved** 1 May 2017, **from** <https://www.codementor.io/codementorteam/react-vs-angular-2-comparison-beginners-guide-lvz5710ha>
- Osmani, A. (2012). *Learning JavaScript Design Patterns*.
- Peyrott, S. (2017). A Brief History of JavaScript. **retrieved** 28 May 2017, **from** <https://auth0.com/blog/a-brief-history-of-javascript/>
- Precht, P. (2016). Two-way data binding in Angular. **retrieved** 14 May 2017, **from** <https://blog.thoughttram.io/angular/2016/10/13/two-way-data-binding-in-angular-2.html>
- React. (2017a). React Docs. **retrieved** 20 April 2017, **from** <https://facebook.github.io/react/>
- React. (2017b). Releases. **retrieved** 28 May 2017, **from** <https://github.com/facebook/react/releases>
- Rylan, C. (2016). Introduction to Angular Routing. **retrieved** 19 May 2017, **from** <https://coryrylan.com/blog/introduction-to-angular-routing>
- Segue-Technologies. (2013). What is Ajax and Where is it Used in Technology? **retrieved** 29 May 2017, **from** <http://www.seguetech.com/ajax-technology/>

- Shaked, U. (2011). AngularJS vs. Backbone.js vs. Ember.js. **retrieved** 28 April 2017, **from** <https://www.airpair.com/js/javascript-framework-comparison>
- Tsuneo, Y. (2015). Data binding code in 9 JavaScript frameworks. **retrieved from** <http://engineering.paiza.io/entry/2015/03/12/145216>
- TypeScript. (2017). TypeScript: JavaScript that scales. **retrieved** 28 May 2017, **from** <https://www.typescriptlang.org/>
- Walsh, D. (2007). 6 Reasons To Use JavaScript Libraries & Frameworks. **retrieved** 17 April 2017, **from** <https://davidwalsh.name/6-reasons-to-use-javascript-libraries-frameworks>
- Wasson, M. (2013). ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET. **retrieved** 17 April 2017, **from** <https://msdn.microsoft.com/en-us/magazine/dn463786.aspx>
- Wheeler, K. (2014). Learning React.js: Getting Started and Concepts. **retrieved** 30 April 2017, **from** <https://scotch.io/tutorials/learning-react-getting-started-and-concepts>
- Wodehouse, C. (2013). What Is a Framework? **retrieved** 28 May 2017, **from** <https://www.upwork.com/hiring/development/understanding-software-frameworks/>



## List of Figures

1.1	JavaScript/jQuery sample code vs. Angular .....	18
2.1	MVC vs. MVP vs. MVVM .....	23
3.1	Google Chrome timeline (Krause, 2016a) .....	26
4.1	Parent/child components in Angular .....	32
4.2	Data binding in Angular .....	33
7.1	Downloaded frameworks from <a href="https://cdnjs.com">https://cdnjs.com</a> .....	46
7.2	Required commands to get an application started .....	47
7.3	Basic application skeleton .....	48
7.4	A simple component in each framework .....	48
7.5	A simple template in each framework .....	48
7.6	A for-loop example in each framework .....	49
7.7	Defining routing in each framework .....	50
7.8	Benchmark results for duration tests (in milliseconds) .....	54
7.9	Benchmark results for memory allocation (in MB) .....	56
7.10	Google Trends' interest over time .....	56





## List of Tables

7.1	Framework sizes (in KB)	46
7.2	Benchmark results for duration tests (in milliseconds)	55
7.3	Benchmark results for memory allocation (in MB)	55
7.4	Frameworks in Github and StackOverflow	57

# Angular, Backbone, React and Ember: Experimental analysis and performance comparison

## Bachelor thesis research proposal

Seyed Kavooos Boloorchí<sup>1</sup>

### Abstract

JavaScript frameworks have evolved over the last few years. Four of the most popular frameworks are Angular, Backbone, React and Ember. Each of them has its own pros and cons which sometimes make it quite difficult to select one. In this bachelor thesis, these frameworks and their technology choices are investigated in details, and their pros and cons are identified. To this end, same projects are developed in each framework, and different aspects such as their performance, costs and community are compared. The outcome of this thesis clarifies where each framework excels and what types of projects would benefit the most from each option.

### Keywords

Web application development. JavaScript — Angular — Backbone — React — Ember

Contact: <sup>1</sup> seyedkavooos.boloorchí.u1660@student.hogent.be

## Contents

1	Introduction	1
2	State-of-the-art	1
3	Methodology	1
4	Expected results	2
5	Expected conclusions	2
	References	2

## 1. Introduction

It was not long ago when JavaScript frameworks were raising eyebrows on their own. Now they have grown and are used in complex scenarios, poaching no small number of developers. But how exactly do these newfangled frameworks up against each other? When is it better to choose which framework?

In spite of the existence of numerous articles, comparing different frameworks, almost all of the comparisons are subjective and lack the implementation of a unique project in all the compared frameworks. Additionally, they do not provide a clear conclusion that which framework is suitable for what kind of project. Even if there is such a conclusion, it is only a suggestion without implementing the same projects in the compared frameworks to give readers a clear vision. Therefore, a beginner web developer cannot decide which framework is the best option for his or her project. This is the target of this study to analyze and compare different JavaScript frameworks experimentally by implementing several projects in different frameworks. Such implementations enable a fair comparison among the frameworks and simplify the decision making process.

In this thesis, we concentrate on four of the most popular frameworks: Angular, Backbone, React and Ember. We investigate whether the benefits of each of the named frameworks outweigh the other ones and could a clear choice between these four be made.

## 2. State-of-the-art

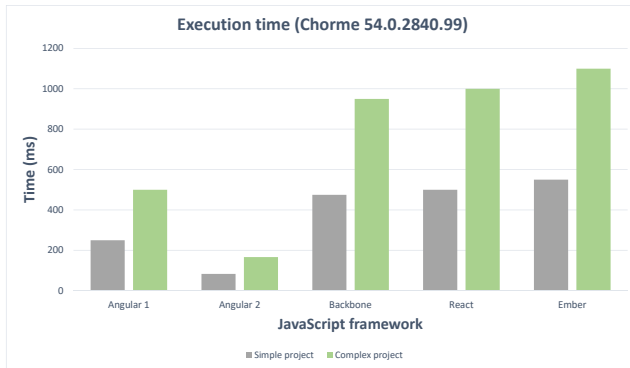
There are hundreds of websites, blogs and articles comparing different JavaScript frameworks, including the ones considered in this study. Articles such as “*AngularJS vs. Backbone.js vs. Ember.js*” [1] has explained Angular, Backbone and Ember frameworks and compared their pros and cons. In another article “*Choosing a Front End Framework: Angular vs. Ember vs. React*” [2] authors identified the strengths and weaknesses of Angular, Ember and React frameworks.

This thesis differs the existing works in the sense that the same projects are implemented in different frameworks. Based on the experimental results, the advantages and disadvantages of each framework are identified. The outcome of this study provides a guideline to select the most suitable framework depending on the complexity of the projects.

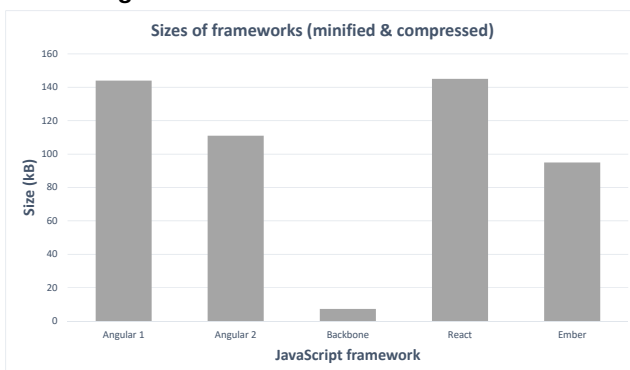
## 3. Methodology

In order to have experimental analysis and a fair performance comparison among the four selected frameworks, several projects with different characteristics – simple or complex – are developed. These projects have different complexities so that we can compare different aspects such as performance, cost, execution time, memory consumption and difficulty to identify the suitability of each framework for different scenarios.

**Figure 1.** Execution time of different JS frameworks.



**Figure 2.** Size of different JS frameworks.



It is worth mentioning that Angular 1 and Angular 2 are counted as one framework in this study but they will be investigated separately and their results will be given separately as well.

#### 4. Expected results

Based on the complexity of the projects, different execution times for different frameworks are expected. These execution times are evaluated in this study which can be compared in a graph as depicted in Fig. 1.

We also compare frameworks size as illustrated in Fig. 2.

Since a framework community is an important aspect to consider before choosing that framework, in this research we also provide a comparison of these different frameworks community size (see Table 1).

#### 5. Expected conclusions

Among the four selected frameworks, Angular 1 is expected to be the best general choice. This prediction is based on: (i) its growth rate over the last few years, (ii) performance and (iii) its extensive community. One of the expected disadvantages of using Angular 1 could be its size. On the contrary, Backbone is quite lightweight and large projects can use this advantage of Backbone.

**Table 1.** Communities of different JS frameworks.

	Angular	Backbone	React	Ember
YouTube results	~90k	~10k	~10k	~9k
StackOverflow questions	~100k	~18k	~15k	~15k
Github contributors	90	250	250	500
Third-party modules	1500	250	1000	1000

The short lifetime of Angular 2 causes a doubt in most of developers decision for using it as their first choice. React and Ember are expected to be in the same level as each other but still the performance of Angular 1 is higher than these two. Besides, learning Angular 1 for beginners seems to be much more easier than all the other frameworks due to the existence of thousands of related tutorials and resources.

#### References

- [1] Uri Shaked. Angularjs vs. backbone.js vs. ember.js, 2014.
- [2] Zack Kuhn. Choosing a front end framework: Angular vs. ember vs. react, 2015.