HAAGA-HELIA
University of Applied Sciences

JavaScript frameworks: Angular vs React vs Vue

Elar Saks

Bachelor’s Thesis
Business Information Technology
2019

Haaga-Helia
University of Applied Sciences

Abstract

Date

Authors Elar Saks	
Degree programme Business Information Technology	
Report/thesis title JavaScript frameworks: Angular vs React vs Vue	**Number of pages and appendix pages** **42 + 1**

The aim of this study was to compare the three of the most popular JavaScript frameworks in popularity, the difficulty of learning and in performance, so that the reader could make an educated decision when deciding which framework to learn or to use for the project.

To understand the popularity of the frameworks, the data by the three most used software development cooperation platforms were collected and analyzed. React was found to be the most popular out of these three frameworks.

The frameworks learning curve was compared by using its official technical documentation to learn about its syntax, architecture, data management, lifecycle, and in the ease of using third-party libraries. For the writer of this study, Vue seemed to be the easiest to learn.

To compare the framework's performance a simple single page application was built and tested in each framework. Vue proved to be the fastest performing framework.

In the end, it was concluded that: React is the best framework to learn when looking for a job, Vue is the best framework for small scale applications that require fast performance. Angular is most likely best suited for bigger more complex applications. Although, further research is needed to compare frameworks when building large applications.

Key words
JavaScript frameworks, Angular, React, Vue.

Table of contents

1 Introduction

This chapter will tell about thesis background, objectives, JavaScript frameworks to be studied and the domains they are compared in.

1.1 Background

Majority of modern websites use JavaScript in some shape or form, to make the pages more interactive and to handle functionality. The traditional webpages are multi-page applications, where a new HTML document gets loaded every time the user changes page or its content. That is a relatively slow option compared top more modern version of a single-page application (SPA) development model, where only the parts that changed get fetched from the server and updated. Using SPA model reduces application loading speeds and improves user experience.

Single-page applications are gaining popularity, and so do the frameworks they are built-in. Frameworks dictate the application development workflow, reduce the development time and possible errors. Each framework is different with its pros and cons, therefore selecting the correct one for the software project can be a strategic decision for a company and for the junior developer looking to add a new skill to his or her arsenal. As of 2019, Angular, React, and Vue JS are some of the most popular JavaScript frameworks.

1.1.1 Angular

Around 2008 and 2009 Misko Hevery and 2 other Google developers had spent 6 months on building an internal feedback tool for Google, only to realize that it had become too difficult to continue, because it was hard to test the code. Misko and developers were using Googles internal Java framework called Google web toolkit. To write 1 label in HTML, it had to be written in Java, compiled and transformed into HTML and JavaScript to be displayed in browser.

Misko challenged his manager Brad Green that he could do the whole project alone in 2 weeks, using the technology that he and his friend Adam Abrons had built as a side project. It took him Misko 3 weeks. Brad was impressed and asked Misko to finish Google feedback tool with Shyam Seshari and Igor Minor using Angular. After that Angular got open sourced on

GitHub and became available to people outside of Google. It was named Angular because of its use of HTML, that has angular brackets. (Gudelli 2017).

When the second version of Angular was released, the framework was changed so much that it could be considered a new framework. Angular versions 2 and up are backward compatible till the Angular 2, but not with Angular 1. To avoid confusion, Angular 1 is now named Angular JS and Angular versions 2 and higher are named Angular. Angular JS is based on JavaScript while Angular is based on JavaScript superset called TypeScript. (Dziwoki 2017).

Modern day Angular is designed for all platforms: web, mobile web, native mobile and native desktop. It is built for speed on web and enables users to have control over scalability while meeting huge data requirements. It is supported by Googles and has millions of users worldwide (Angular a).

1.1.2 React

React is a JavaScript library for building user interfaces. React is built on 2011 by Jordan Walke, a software engineer at Facebook. It got open sourced on 2013 and has gained additions to it ever since. On 2015, React Native, a library for mobile devices was open-sourced and on 2017, a library for virtual reality development, React360, was released. React is currently developed by Facebook and community of individual users. Facebook uses React in its projects, such as Facebook and Instagram. (Wikipedia).

React is designed to enhance interactive UI development by making it easier to update the view when the data changes. It is done through dividing the view into smaller components, that can be composed to create complex UIs. Components are built in JavaScript instead of templates, enabling easy flow of data. (React a).

1.1.3 Vue

Vue.js is a lightweight JavaScript library, created by Evan You. Before creating Vue, Evan had been working in Google and Meteor. While in Google, Evan felt that using Angular in certain use cases was too heavy and he decided to extract the parts he likes about Angular, to create his light-weight library. On 2014 he shared his work with others on GitHub and that is how Vue.js got started.

Although Evan started Vue while working with Angular, he considers Vue being most similar with the React, as its core idea is data binding and components, as in React. Compared to React, Vue puts more emphasis on the user experience, making it easy to pick up, if user knows the basics: HTML, JavaScript and CSS.

After starting a crowdfunding campaign in Patreon on 2015, Evan has gathered a small amount of private and corporate sponsors. That generates him enough cashflow that for now he is working fulltime on Vue JS. (Cromwell 2017).

1.2 Objectives

Each framework is built different and will perform differently. This study aims to bring out the possible advantages and disadvantages of each framework, enabling a reader to make an educated decision when choosing between them. To achieve that the frameworks will be compared in 3 different categories: popularity, learning curve, and performance.

1.3 Popularity

The framework's popularity will be assessed by collecting and analysing data from major cloud platforms related to software development, such as GitHub, NPM trends, and Stack Overflow.

1.4 Learning curve

The framework learning curve will be evaluated by comparing the framework syntax, architecture, data management, lifecycle, and in the ease of using third-party libraries. The information for that will be collected from each framework's technical documents.

1.5 Performance

To benchmark, the performance of the framework, small application with the same business logic is built in each framework. Then the application's production builds performances will be measured using Google Chromes developer tools. Applications code will be uploaded to the GitHub and working applications will be made accessible over the internet.

1.6 Out of scope

In this study, JavaScript frameworks (Angular, React, Vue) are compared by building single-page applications. Frameworks are not compared by building a multipage application. Frameworks: Angular, React, and Vue are compared only against each other and not against vanilla JavaScript. Under the name of 'Angular' in this thesis is meant JavaScript framework Angular 2 and up.

1.7 Objectives

Each framework is built different and will perform different. The aim of this study is to bring out the possible advantages and disadvantages of each framework, enabling a reader to make an educated decision when choosing between them. To do that, the frameworks will be compared in 3 different categories: popularity, learning curve and performance.

2 Frameworks Popularity

This chapter provides an overview of data sources and data that was used to analyse the popularity of the framework. Data sources are picked so that all 3 of them are used by a large number of software developers around the world in their day to day operations. Data collected from them is rather reflecting the situation as it is, instead of relying on opinions or hype.

2.1 GitHub

GitHub markets itself as a software development platform, used by 31 million developers to host and manage open source and business projects. (GitHub 2019). Hosting the code written by so many different developers for so many different purposes makes a GitHub unique and reliable source to try to gain understanding about the software development market and its trends. Table 1 displays information about Angular, React, and Vue repositories on GitHub.

Table 1. Statistics about frameworks GitHub repositories as of 22.03.2019 (GitHub b; GitHub c; GitHub d.)

	Angular	React	Vue
GitHub Stars	46,572	125,734	133,479
GitHub Forks	12,286	22,848	18,990
GitHub Issues	2,331	453	184

GitHub users can mark repositories they are interested in, with stars, making them easier to find later and letting GitHub know on what topics the user is interested in knowing about. It will also show the appreciation towards the work of a repository maintainer. GitHub repositories are ranked and can be sorted in search based on the number of stars. (GitHub e 2019).

Users can make personal copies of other user repositories and freely make changes on it, without affecting the original project. Copies act as a different path coming from an original project and can be updated by making a pull request to the source. Forks or paths can be recommended to merge with the original repository, making it better for everyone. The number of forks coming out from a repository is a good indicator of how many developers or teams are trying to improve the project as an addition to original authors. (GitHub f 2019).

GitHub has a built-in bug tracker called issues. It makes it easier to keep track of tasks and to share them with a team (GitHub g 2019). The amount of issues could provide some idea about the scale and complexity of a project, as well as community support towards it.

2.2 Npm Trends

Node package manager is the largest software registry in the world. It allows open source developers to share and borrow JavaScript packages. It is also used by many organizations to manage private projects (NPM.) NPM trends is a website that provides information about the amount of JavaScript packages downloaded via NPM throughout time. Figure 1 displays a graph about Angular, React, and Vue packages download during the past two years.

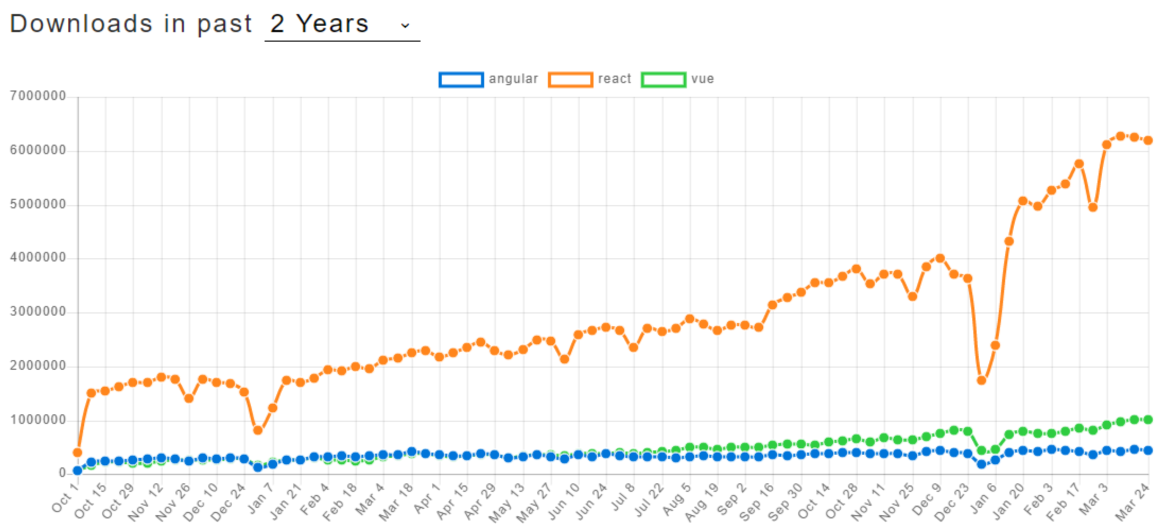


Figure 1. Angular, React and Vue libraries downloads via NPM from 2017 till 2019 (Npm Trends)

2.3 Stack Overflow

Stack Overflow is an online platform where developers can ask and answer questions about programming. According to their homepage, more than 50 million people visit their Stack Overflow every month (Stack Overflow a).

This vast user base made up from developers and tech enthusiasts, discussing different programming languages makes it a perfect place to get an understanding of what the global developer community is using and talking about.

Figure 2 is a graph displaying the percentage of total questions asked about the Angular, React, or Vue during the past ten years on Stack Overflow.

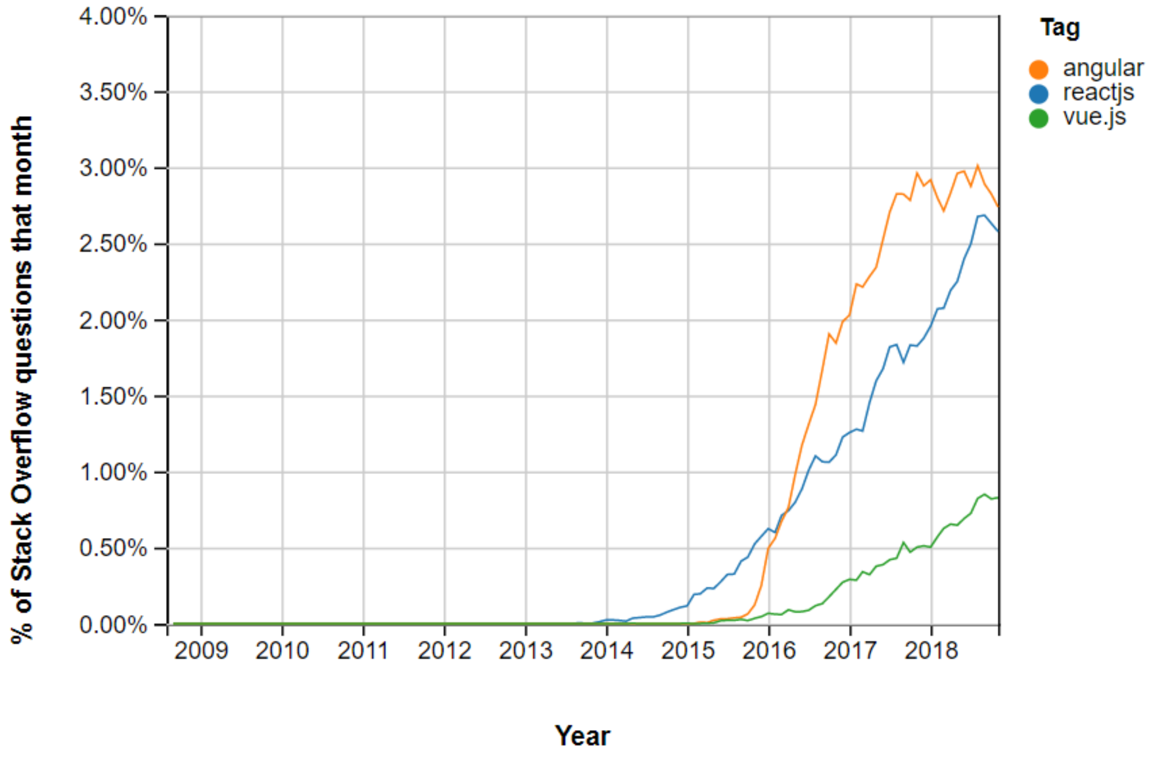


Figure 2. Percentage of questions about Angular, React and Vue from 2009 to 2019 on Stack Overflow (Stack Overflow b)

3 Frameworks learning curve

This chapter provides a high-level technical overview of each framework. Information for this chapter is gathered from each framework's documentation. Since these are all JavaScript frameworks, they are all similar in their structure and workflows. That is why it was possible to gather information about each framework on the five same topics: syntax, architecture, data management, lifecycle, and third-party libraries.

3.1 Angular

Agular's homepage, Angular.io, has extensive documentation explaining the framework. It has tutorials to get new coming developers up to the level and it has documentation for more in-depth topics, for more experienced Angular developers.

3.1.1 Syntax

Angular is written in HTML and TypeScript (Angular b). TypeScript is an open-source programming language developed and maintained by Microsoft, that adds static typing option to JavaScript and compiles into JavaScript (Wikipedia b).

Angular template syntax is HTML, and almost all HTML is the valid Angular syntax, except <*script*> tag, that gets ignored by Angular to avoid script injection attacks. Angular extends regular HTML by making some JavaScript function expressions available in HTML. Double curly braces are used to render JavaScript content into the view (Angular c).

3.1.2 Architecture

Angular applications are composed of NgModules, that are containers for a block of code dedicated to an application domain, workflow, or a closely related set of capabilities. NgModules provide the compilation context for their content. They can contain components, service providers, and other code files. They can export and import functionality from and to other modules.

Every Angular app has a root module, conventionally named AppModule, which provides the bootstrap mechanism that launches the application. Root module can include an unlimited hierarchy of child modules. Organizing code into distinct functional modules helps to manage the development of complex applications (Angular d).

Angular partly utilizes the Model View Controller concept, by having a component as a controller and template as a view. Component controls the view that represents a patch of the screen (Angular e). A component can contain a view hierarchy, made up of views from different modules. Illustration of it can be seen in figure 3., where a root component belongs to module A, and some of its child and grandchild components are imported from module B.

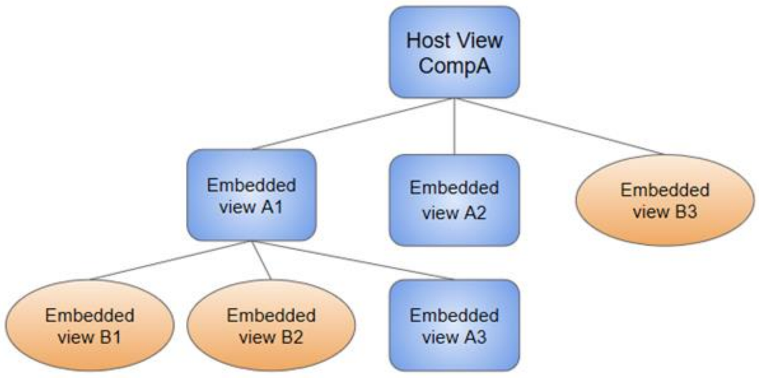


Figure 3. Application view hierarchy (Angular f)

3.1.3 Data management

Angular supports two-way data binding, a mechanism for coordinating the parts of a template with the parts of a component. Data flows into the template from a component with the property binding and back from template to component with event binding, as illustrated in Figure 4.

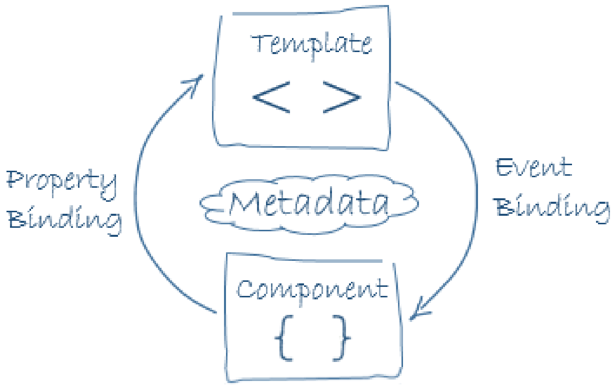


Figure 4. Data flow between component and a template (Angular g)

Angular processes all data bindings once, for each JavaScript event, from a root of an application to all the child components in the component tree (Angular g).

3.1.4 Lifecycle

A component has a lifecycle managed by Angular. It creates the component, renders the component and updates the component when the data changes. Angular offers lifecycle hooks, giving a user an opportunity to interact with the code during the different stages of a component lifecycle.

These are lifecycle hooks explained on the Angular website:

– *ngOnChanges()* gets called whenever one or more data properties changes
– *ngOnInit()* initializes the component when it is first created or when the *NgOnChanges()* is called
– *ngDoCheck()* is called during every change detection run, right after *ngOnChanges()* and *ngOnInit()*. Is used to detect changes that Angular cannot or won't detect itself
– *ngOnChanges()* and *ngOnInit()*. Is used to detect changes that Angular cannot or won't detect itself.
– *ngAfterContentInit()* – is called once after the first *ngDoCheck()*. Respond after Angular projects external content into components view.
– *ngAfterContentChecked()* – is called after the *ngAfterContentInit()*. Let's user to respond after Angular has checked the content projected into the component.
– *ngAfterViewInit()* – is called after *ngAfterContentChecked()*. Let's user to respond after Angular has initialized the component and its children views.
– *ngAfterViewChecked()* – called after *ngAfterViewInit()* and every following *ngAfterContentChecked()*. Let's user to respond after Angular has checked all the rendered component and its children views.
– *ngOnDestroy()* – called just before Angular destroys the component. Cleans up before component destruction by unsubscribing to any observables, to avoid memory leaks.

Other Angular sub-systems or third-party libraries might have their own lifecycle hooks apart from these component hooks (Angular h).

3.1.5 Third party packages

Third-party libraries can be used in Angular by installing them through NPM and importing the provided functionality (Angular i). Third-party libraries extend Angular usability by adding readymade styling and functionality.

One of the most popular styling libraries for Angular is Angular Material. It consists of ready-made Material Design components for Angular. They are fine-tuned for performance and integrate seamlessly with Angular (Angular j).

3.2 React

React has proper official documentation with multiple walk-through tutorials on their homepage reactjs.org, about how to get started. Main concepts are explained clearly and illustratively for newcomers and for experienced users, documents are going in-depth on more advanced topics.

3.2.1 Syntax

React uses JSX syntax, that is JavaScript syntax extension. It is a mixture of JavaScript and HTML and might remind a template language to some, with the exception of having the full power of JavaScript (React b).

3.2.2 Architecture

JSX is used to build elements that makeup components. React elements are plain objects and cheap to create, compared to browser DOM elements. Reacts virtual DOM takes care of updating the browsers DOM (React c). Virtual DOM (VDOM) is a concept where "virtual" representation of UI is kept in memory, and "real" DOM is synced via a library, such as React DOM. That removes the manual attribute and event handling and automates updating the DOM (React d).

3.2.3 Data management

React components are the building blocks of application, that allow UI to be split into independent, reusable parts that can be thought about in isolation. Components are like JavaScript functions that accept inputs (called "props") and return React elements. Components can refer to other components, making them reusable. Usually, React apps have a single "App" component on top of the view hierarchy (React e).

Each React component can, but do not have to have a state. In React, the state is local and encapsulated into a single component. Data can be passed from parent components to children as props. Props are immutable and cannot be passed upward in a component tree hierarchy. A state can be changed, and on state change, the component will be re-rendered. For better control over components, lifecycle React has built-in lifecycle functions providing quick and easy control over events, such as component mounting, unmounting, receiving data, etc (React f).

3.2.4 Application Lifecycle

Figure 3 illustrates components lifecycle. The virtual document object model is kept in the cache. Changes in data are passed down from top to down. React then compares the differences between virtual DOM and Real DOM and makes changes if necessary.

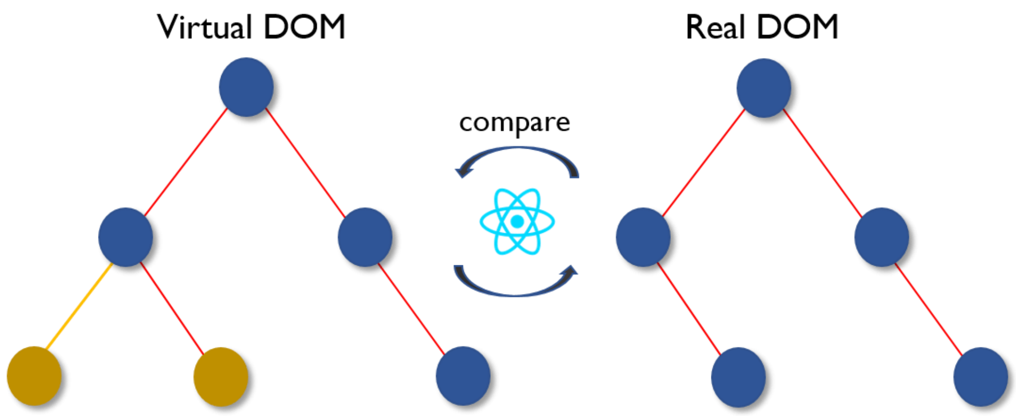


Figure 5. React component tree and virtual DOM (Pngkey)

React official documentation has an explanation of a component lifecycle and few lifecycle methods, but not the full overview of a lifecycle or lifecycle hooks.

3.2.5 Third Party Packages

Third-party packages in React can be installed through NPM (Node Package Manager). Running third party libraries on top of React enhances its usability and adds extra functionality, such as two-way data binding and routing. Probably one of the most used third-party libraries for React is Redux.

Redux is a state container for a JavaScript application and can be used together with React. In Redux, the whole app state is stored in an object tree inside a single store. As the application grows, the Redux store grows and can be divided into a tree-like structure, the same as the app component tree. It might seem like overkill for smaller applications, but it will prove beneficial as the application scales up and gets more complex (Redux a).

Figure 4. displays how Redux store receives data from the lowest component of a component tree and can share it with the nodes higher up in the tree.

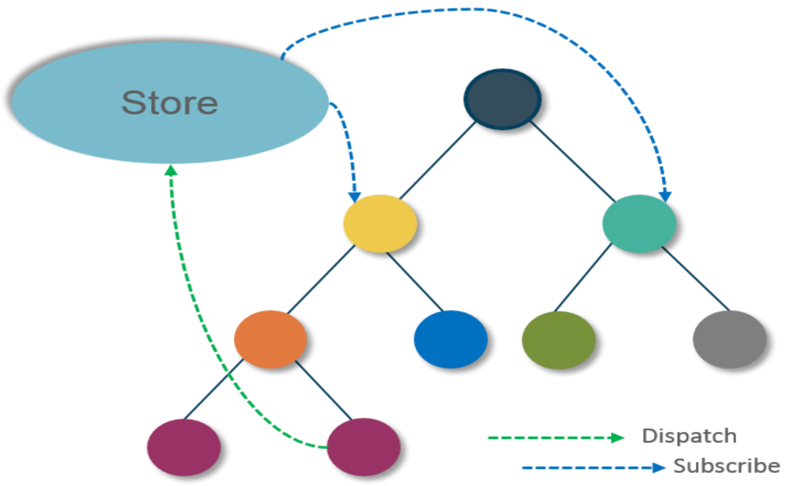


Figure 6. Redux store distributing data (Edureka)

3.3 Vue

Vue JS has in detail documentation and examples on their homepage vuejs.org, on about how to get started. Main concepts are explained thoroughly and illustrated with examples on their homepage and in jsfiddle.

3.3.1 Syntax

Vue.js uses HTML -based template syntax, that gets binded to underlying Vue instance JavaScript. All Vue templates are valid HTML. Data coming from Vue application can be binded into HTML using the "Mustache" syntax (double curly braces), for example:
<p> Message: {{ *msg* }} *</p>*. Everything written insides these double curly braces will be evaluated as JavaScript in the data scope of a Vue instance (VueJS a).

3.3.2 Architecture

Vue applications are made up of reusable components called Vue instances, accepting the same options as a new Vue instance, such as data, methods, and lifecycle hooks. The only exceptions are a few root-specific options. It is common for an app to be organized into a tree of nested components and illustrated in figure 7. To use components in templates, they must be registered so that Vue knows about them. Components can be registered globally and locally. Having registered component globally makes it available to all the subcomponents of that Vue instance component tree
(VueJS b).

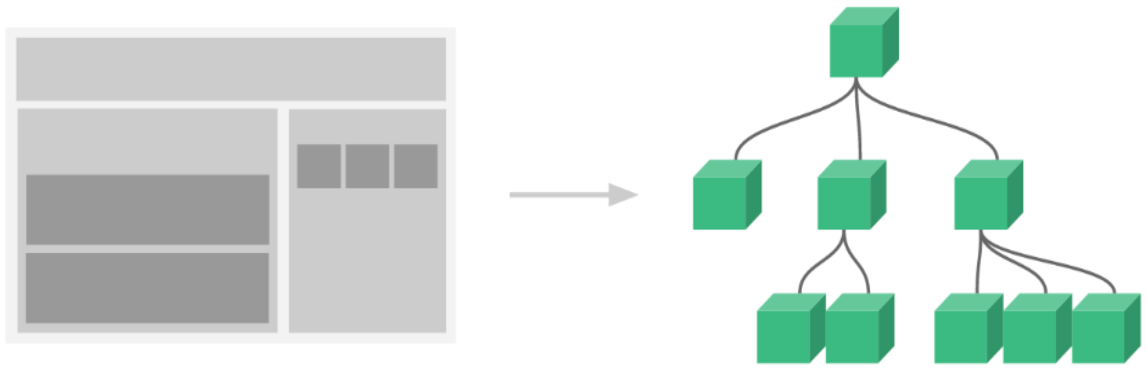


Figure 7. Visual user interface and component tree (VueJS c)

3.3.3 Data management

Application data is stored in the components data function, that acts as a components state. Data can be passed as a prop from parent to child components. When a value is passed as a prop attribute, it becomes the property of that component instance. Prop value can be accessed in the component the same as a data attribute. One component can have an unlimited amount of props tree (VueJS d; VueJS e).

3.3.4 Application lifecycle

Similar to the React, Vue uses Virtual Dom to update the page. Each Vue instance goes through a series of initialization steps when it is created. It needs to set up data observation, compile the template, mount the Vue instance on the DOM, and update the DOM when the data changes. During this process, it also runs the lifecycle hooks, that allow users to add their code at the specific stages.

Appendix 1. illustrates the full lifecycle of a Vue instance. When new Vue instance gets called, Vue initiates the instance creation but gives a user a chance to add their code by calling a beforeCreated lifecycle hook, before finishing the instance creation.

Then Vue checks if the root element has any options, such as a template. If the element property exists, it will be compiled, if not Vue instance parent HTML element will be used as a template.

Before mounting the compiled element, a user is given another lifecycle hook named beforeMount, and after creating a view model, a user gets another chance to affect the component by using lifecycle hook named mounted. After that, the component gets rendered.

While the component is mounted, Vue actively watches for the changes in data. When the data changes, the user is given a lifecycle hook beforeUpdate, to add their code before virtual DOM gets re-rendered.

When the component gets dismounted and the destroy function is called, the user has a chance to call the hook named beforeDestroy, and after all the components methods and children are torn down, there is a final lifecycle hook named destroyed. (Vue JS f.)

3.3.5 Third-party packages

Third-party packages in Vue can be installed through NPM or by importing them as dependencies through script tag. Third-party libraries provide Vue with extra functionality and enhance its usability.
One of the most widely used third-party packages for Vue JS is a VueX.

VueX is a state management pattern and library for Vue applications. It serves as a centralized data store for all the components in the application, ensuring that the state can be mutated only in a predictable manner. VueX is inspired by Flux, Redux, and The Elm Architecture, with the only exception that it is tailored specifically for the Vue.

Figure 8. illustrates how VueX runs together with a Vue. It gets new data from the Vue, initiates the action, that will determine the way the state will be mutated and then continues mutating and passing the new state back to the Vue component. (VueX).

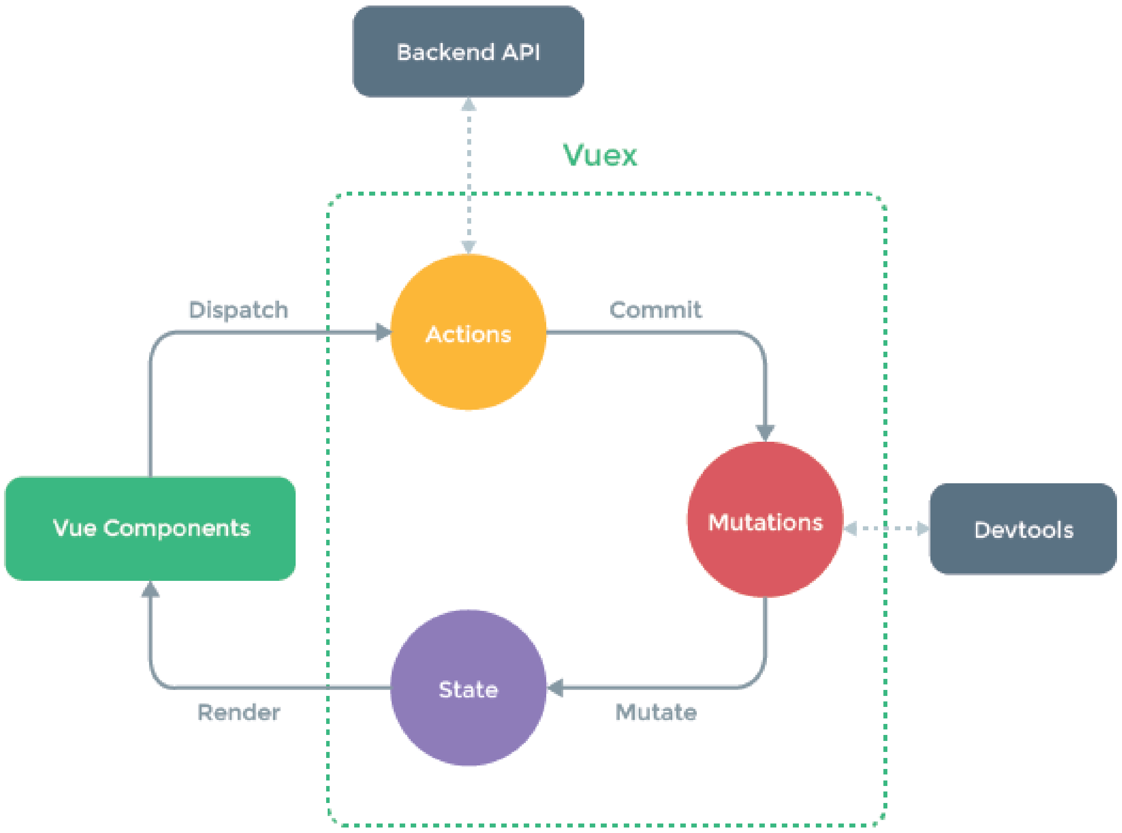


Figure 8. VueX running on top of Vue (VueX)

4 Frameworks performance

To gather data about frameworks performance, small test application is built in each framework to test loading speeds. All test applications have the same business logic and visual design.

4.1 Application design and shared parts

Test applications have a data table and a menu, with five functions:

- Create a table with 1000 rows
- Create a table with 10 000 rows
- Add 1000 rows to the table
- Add 10 000 rows to the table
- Delete all rows

Rows added to the table are made up from two columns: 1 consisting row number and another one consisting, string made up from 3 random words. Applications design can be seen in figure 9.

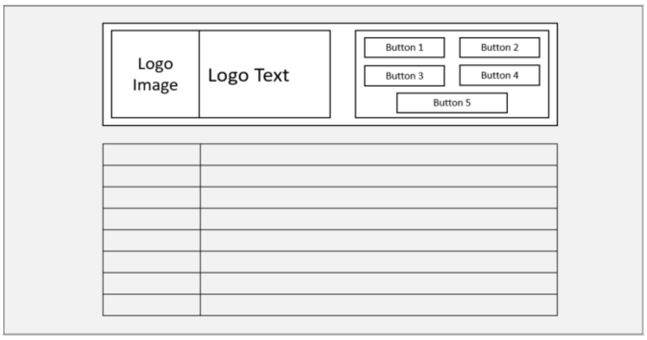


Figure 9. Application design and structure.

Applications are divided into three separate components, with App being the parent component that has two children, as illustrated in figure 10.

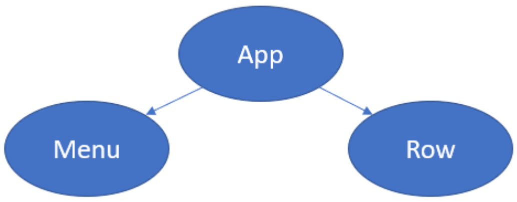
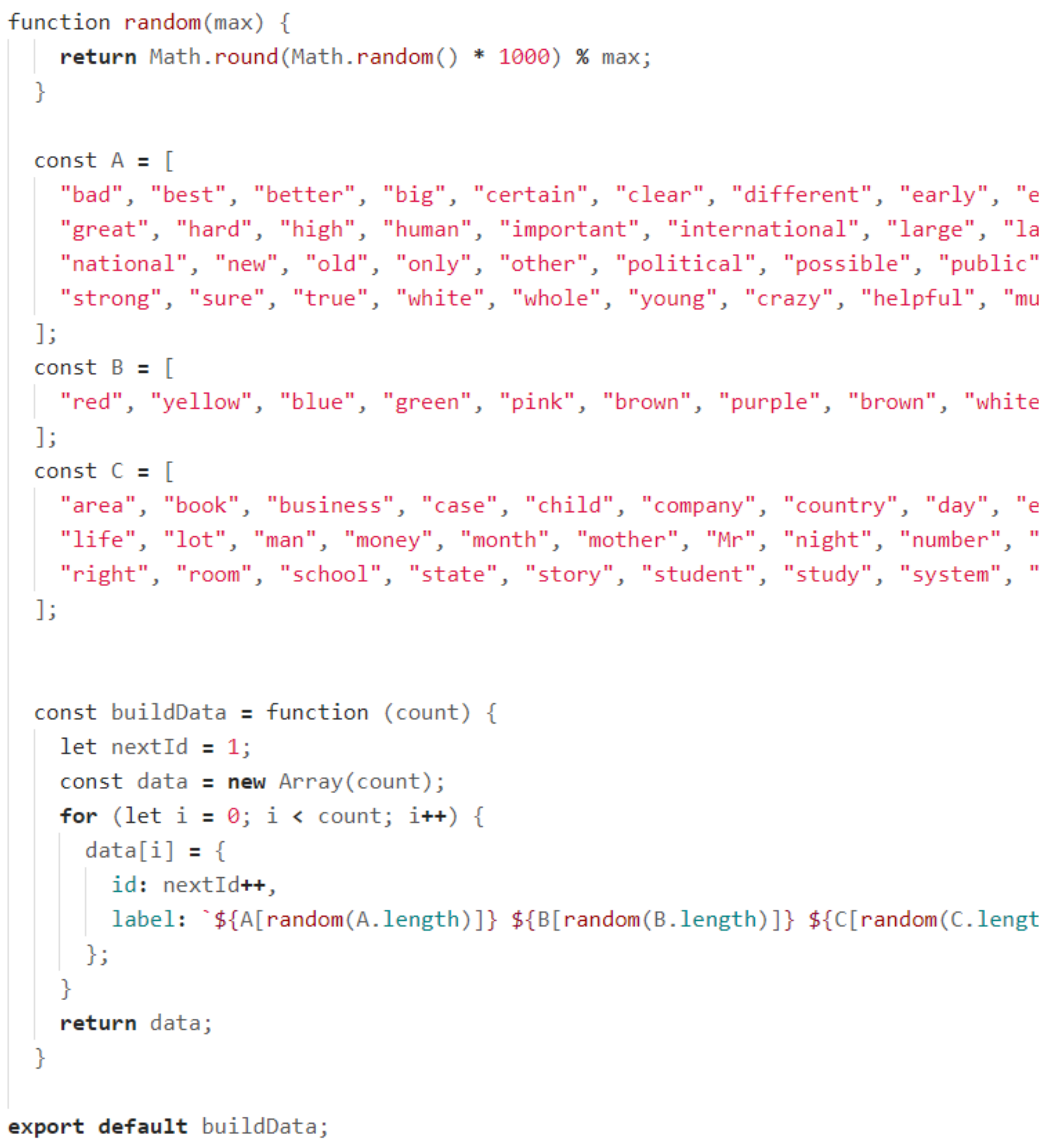


Figure 10. Applications component tree

All three applications are using the same CSS file for styling and the same Vanilla JavaScript function to produce data. Data for table rows is produced by buildData function in a separate file and imported into projects as a dependency.

Figure 11 shows buildData function code. Function buildData takes a row amount as a parameter named count and creates an array with that length. Then a for loop is called to pass an object for each position in the array. Each object is made up of two keys: id and label. It is an integer growing with each iteration of a for a loop. The label is a string made up of three words, that are produced by calling a random number and using it as an index to pick a word from constant arrays named A, B, and C.

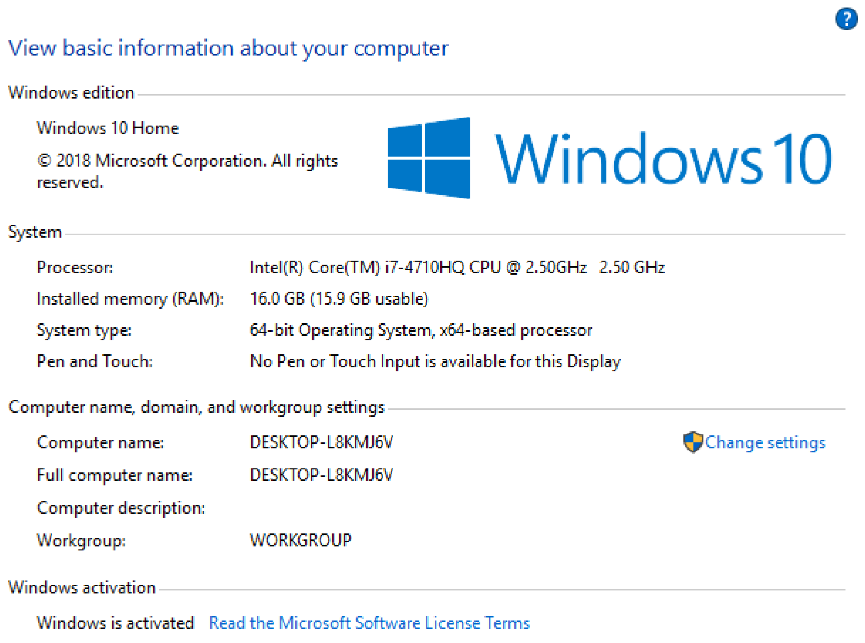


```
function random(max) {
    return Math.round(Math.random() * 1000) % max;
  }

  const A = [
    "bad", "best", "better", "big", "certain", "clear", "different", "early", "e
    "great", "hard", "high", "human", "important", "international", "large", "la
    "national", "new", "old", "only", "other", "political", "possible", "public"
    "strong", "sure", "true", "white", "whole", "young", "crazy", "helpful", "mu
  ];
  const B = [
    "red", "yellow", "blue", "green", "pink", "brown", "purple", "brown", "white
  ];
  const C = [
    "area", "book", "business", "case", "child", "company", "country", "day", "e
    "life", "lot", "man", "money", "month", "mother", "Mr", "night", "number", "
    "right", "room", "school", "state", "story", "student", "study", "system", "
  ];


  const buildData = function (count) {
    let nextId = 1;
    const data = new Array(count);
    for (let i = 0; i < count; i++) {
      data[i] = {
        id: nextId++,
        label: `${A[random(A.length)]} ${B[random(B.length)]} ${C[random(C.lengt
      };
    }
    return data;
  }

export default buildData;
```

Figure 11. Vanilla JavaScript function producing dummy data

4.2 Testing environment

Application speeds were tested and measured in the Google Chrome browser, running locally on the Windows 10 operating system mounted on Asus G73JH laptop. The laptop has a 2.5 GHz i7 processor and 16 GB of RAM, as it can be seen in figure 12.



View basic information about your computer

Windows edition

Windows 10 Home

© 2018 Microsoft Corporation. All rights reserved.

System

Processor:	Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz 2.50 GHz
Installed memory (RAM):	16.0 GB (15.9 GB usable)
System type:	64-bit Operating System, x64-based processor
Pen and Touch:	No Pen or Touch Input is available for this Display

Computer name, domain, and workgroup settings

Computer name:	DESKTOP-L8KMJ6V
Full computer name:	DESKTOP-L8KMJ6V
Computer description:	
Workgroup:	WORKGROUP

Windows activation

Windows is activated Read the Microsoft Software License Terms

Figure 12. Technical properties of a computer used for testing

4.2.1 Testing

Applications were tested locally on a laptop to avoid any anomalies that might have resulted from internet speeds. In total, three tests were done. Each test measured the time to load the home page, and the time it took to perform the functions called by buttons. Buttons “Create 1000” and “Create 10 000” were clicked twice, to see if there were any differences when re-running the same function. Before clicking the “Create 10 000”, the page was cleared by clicking clear button, so that the new 10 000 rows would be created on the empty page. Speeds were measured in milliseconds.

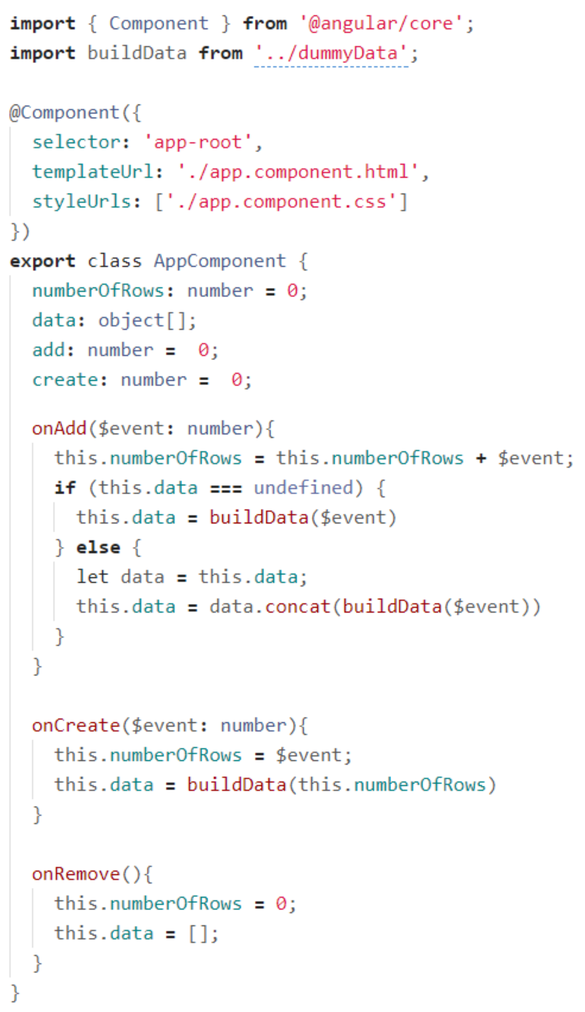
Applications were also made available for further testing, on Amazon Web Services S3 buckets as static websites, configured to be physically located in Stockholm Sweden (AWS a; AWS b; AWS c). Also, applications code for reviewing can be found on GitHub: https://github.com/elarsaks/Thesis.

4.3 Angular

Angular application production build was 15.7MB and is hosted on AWS for testing and on writers GitHub for the code review (AWS a; Elar Saks 2019). Application components are stored in multiple files, having a separate file for HTML template, CSS styling, and for JavaS-cript.

4.3.1 Root component: App

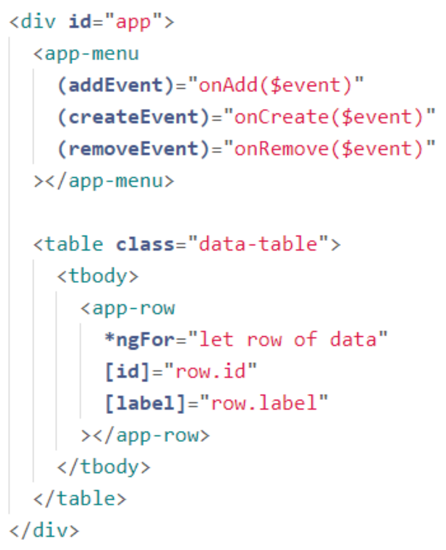
Figure 13 shows the content of a TypeScript file that defines the *App* component. The file starts with importing necessary dependencies, such as Component class from Angular core and a function for a test data creation from *dummyData*. Next, the component properties such as component identifier and the support files locations are set. Then the *App* component is exported with a few data properties serving as a components state, and with a few methods that can be called to update the state.



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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  numberOfRows: number = 0;
  data: object[];
  add: number =  0;
  create: number =  0;

  onAdd($event: number){
    this.numberOfRows = this.numberOfRows + $event;
    if (this.data === undefined) {
      this.data = buildData($event)
    } else {
      let data = this.data;
      this.data = data.concat(buildData($event))
    }
  }

  onCreate($event: number){
    this.numberOfRows = $event;
    this.data = buildData(this.numberOfRows)
  }

  onRemove(){
    this.numberOfRows = 0;
    this.data = [];
  }
}
```

Figure 13. App component declaration

Figure 14 shows App components template or a view. The template is made of a div with an id of *app*, that serves as a container for two child elements. The first child element is a component named *app-menu*, and it receives multiple functions as a prop. The second element is an HTML table that gets *app-row* components looped into it by using Angular directive **ngFor*.



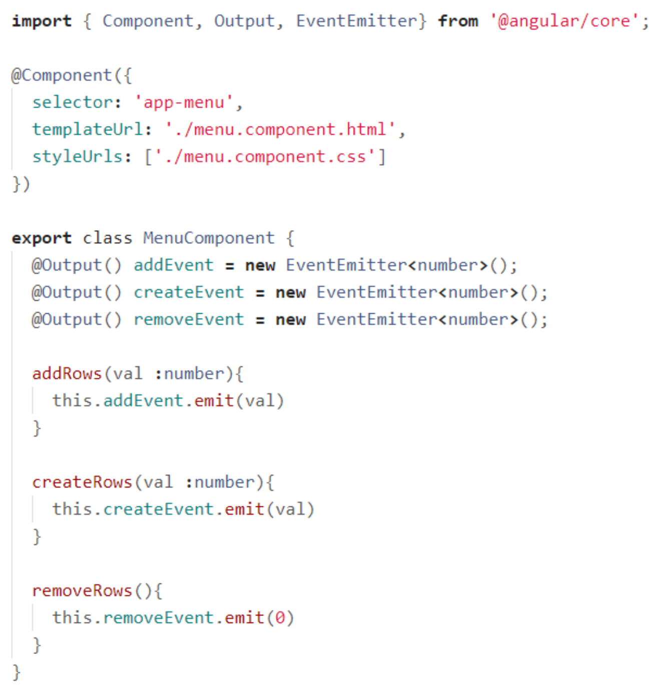
```
<div id="app">
  <app-menu
    (addEvent)="onAdd($event)"
    (createEvent)="onCreate($event)"
    (removeEvent)="onRemove($event)"
  ></app-menu>

  <table class="data-table">
    <tbody>
      <app-row
        *ngFor="let row of data"
        [id]="row.id"
        [label]="row.label"
      ></app-row>
    </tbody>
  </table>
</div>
```

Figure 14. App components template

4.3.2 Menu component

Figure 15. shows the menu component logic or a controller. Menu component starts with necessary imports, such as *Component* class, *Output*, and *EventEmiter* from Angular core. Imports are followed by setting the component settings, such as selectors and supporting files: the view template and its styling.

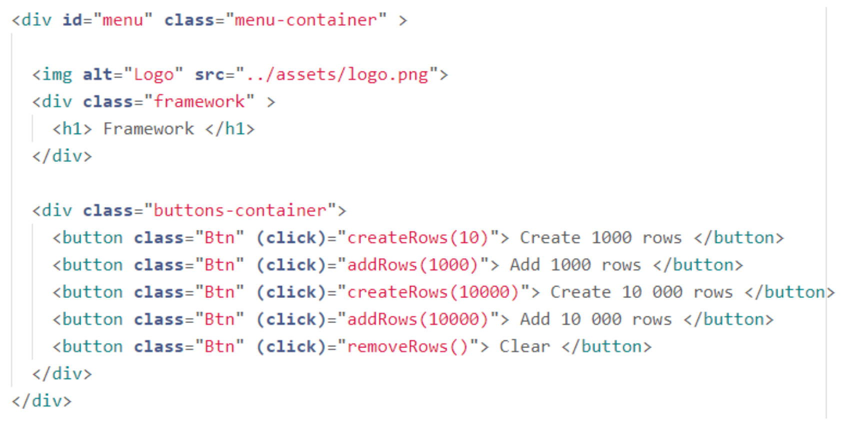
Next comes the *MenuComponent* class declaration, that starts with listing the methods that get returned from the component. File ends with defining the functions to be returned.



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

@Component({
  selector: 'app-menu',
  templateUrl: './menu.component.html',
  styleUrls: ['./menu.component.css']
})

export class MenuComponent {
  @Output() addEvent = new EventEmitter<number>();
  @Output() createEvent = new EventEmitter<number>();
  @Output() removeEvent = new EventEmitter<number>();

  addRows(val :number){
    this.addEvent.emit(val)
  }

  createRows(val :number){
    this.createEvent.emit(val)
  }

  removeRows(){
    this.removeEvent.emit(0)
  }
}
```

Figure 15. Menu component logic

On figure 16. is menu component template or a view. It is made up of a few design elements, such as a header and logo and menu buttons container. Each button has assigned a method defined by the user in the Menu typescript file.

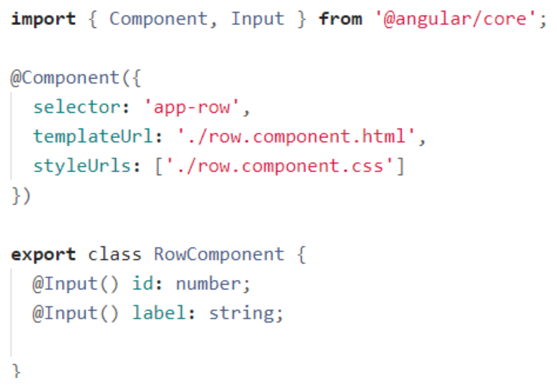


```
<div id="menu" class="menu-container" >

  <img alt="Logo" src="../assets/logo.png">
  <div class="framework" >
    <h1> Framework </h1>
  </div>

  <div class="buttons-container">
    <button class="Btn" (click)="createRows(10)"> Create 1000 rows </button>
    <button class="Btn" (click)="addRows(1000)"> Add 1000 rows </button>
    <button class="Btn" (click)="createRows(10000)"> Create 10 000 rows </button>
    <button class="Btn" (click)="addRows(10000)"> Add 10 000 rows </button>
    <button class="Btn" (click)="removeRows()"> Clear </button>
  </div>
</div>
```

Figure 16. Menu component view

4.3.3 Row component

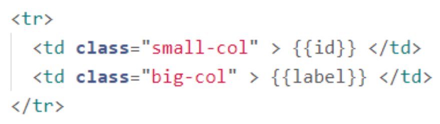
Row component logic part can be seen in figure 17. As it does not have much functionality, it is quite short compared to others. It starts off with some dependency imports and continues with component settings, where the component selector is declared with supporting file locations. It ends with a *RowComponent* class being exported with two incoming data properties: id and label.



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

@Component({
  selector: 'app-row',
  templateUrl: './row.component.html',
  styleUrls: ['./row.component.css']
})

export class RowComponent {
  @Input() id: number;
  @Input() label: string;

}
```

Figure 17. Row component logic

Row components template is made up from only one table row element, made up of 2 columns. Table columns have classes assigned to them, and the content passed in it by Angular expression, using double curly braces.



```
<tr>
  <td class="small-col" > {{id}} </td>
  <td class="big-col" > {{label}} </td>
</tr>
```

Figure 18. Row component template

4.3.4 Test results

Table 2. shows Angular application production build speed tests results in milliseconds. The last column of the table shows the average of the three tests run and is rounded down to full number with no decimal places. The last column will be later used in the analysis part of this thesis.

Table 2. Angular speed test results

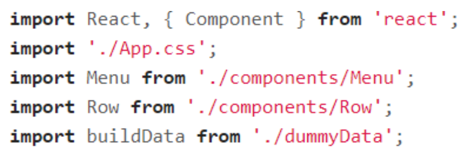
Angular	Test 1.	Test 2.	Test 3.	Average
Load Page	389.6	393.9	425.7	403
Create 1000	404.4	439.5	306.6	384
Re-create 1000	420.2	287.3	284.4	331
Add 1000	288.5	237.9	234.8	254
Create 10 000	6508.6	6268	6306.6	6361
Re-create 10 000	7726.3	7335	7218.5	7427
Add 10 000	7823	7243.2	7804.2	7623
Remove all	2315.9	2205.9	2064.9	2196

4.4 React

React application production build was 587KB and is hosted on AWS for testing and on writers GitHub for the code review (Elar Saks 2019 b).

4.4.1 Root component: App

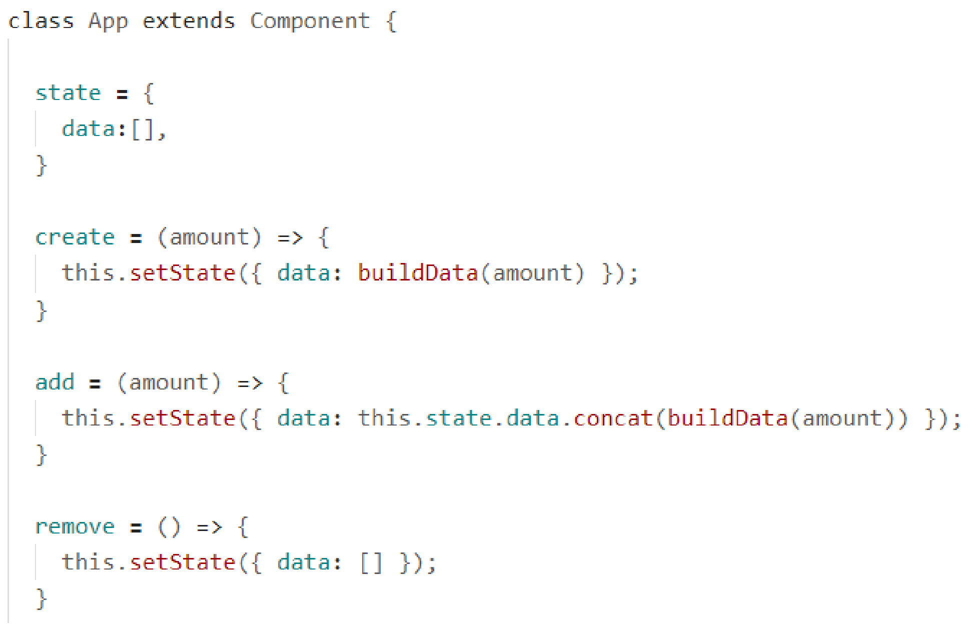
The *App* is the highest component in the view hierarchy and is the clue that brings the rest of the application together. App.js file starts with importing the dependencies, such as *Component* class from React library, CSS file, and child components. Figure 19. displays the *App* component imports.



```
import React, { Component } from 'react';
import './App.css';
import Menu from './components/Menu';
import Row from './components/Row';
import buildData from './dummyData';
```

Figure 19. Importing dependencies in App.js

After imports, the *App* component is declared as a JavaScript class and is extending React library. React components can be divided into two parts, component logic, and component view. The *App* is a stateful component and starts with state declaration, followed by functions that make up the component logic, that can be seen in figure 20.



```
class App extends Component {

  state = {
    data:[],
  }

  create = (amount) => {
    this.setState({ data: buildData(amount) });
  }

  add = (amount) => {
    this.setState({ data: this.state.data.concat(buildData(amount)) });
  }

  remove = () => {
    this.setState({ data: [] });
  }
```

Figure 20. First half of App component

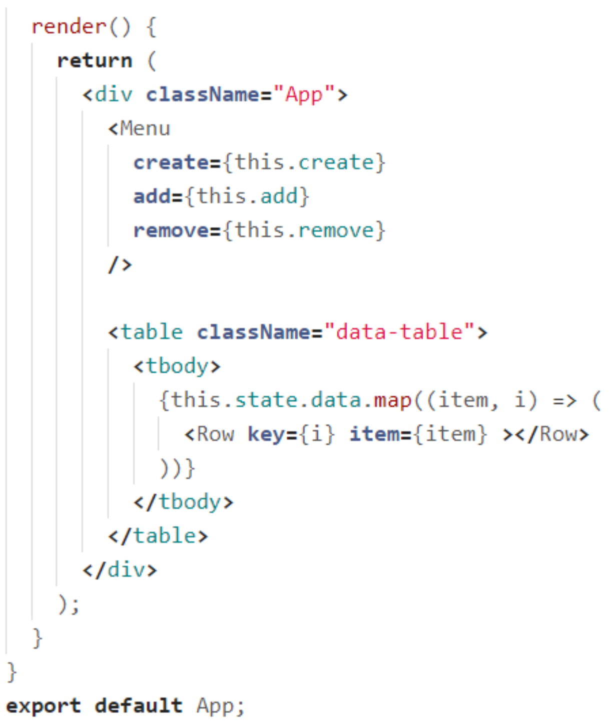
The second part of the *App* component is displayed in figure 21. and it is a render function that returns the *App* components view.

The *App* components view is made up of a div consisting of a *Menu* component and an HTML table. HTML table has a *Row* component that gets rendered into it, based on the amount specified in the application state.

The application state can be changed via functions declared in the first half of the *App* component. These functions are passed as props to the *Menu* component.

Although data flows only from top to down and cannot be passed from children to parent components, it is still possible to update the *App* component state from the Menu component.

Only the function call gets passed from the parent to the child component. Functions are called in the *Menu* component but declared in the *App* component, and will use *App* component as an execution environment, changing the *App* components state.

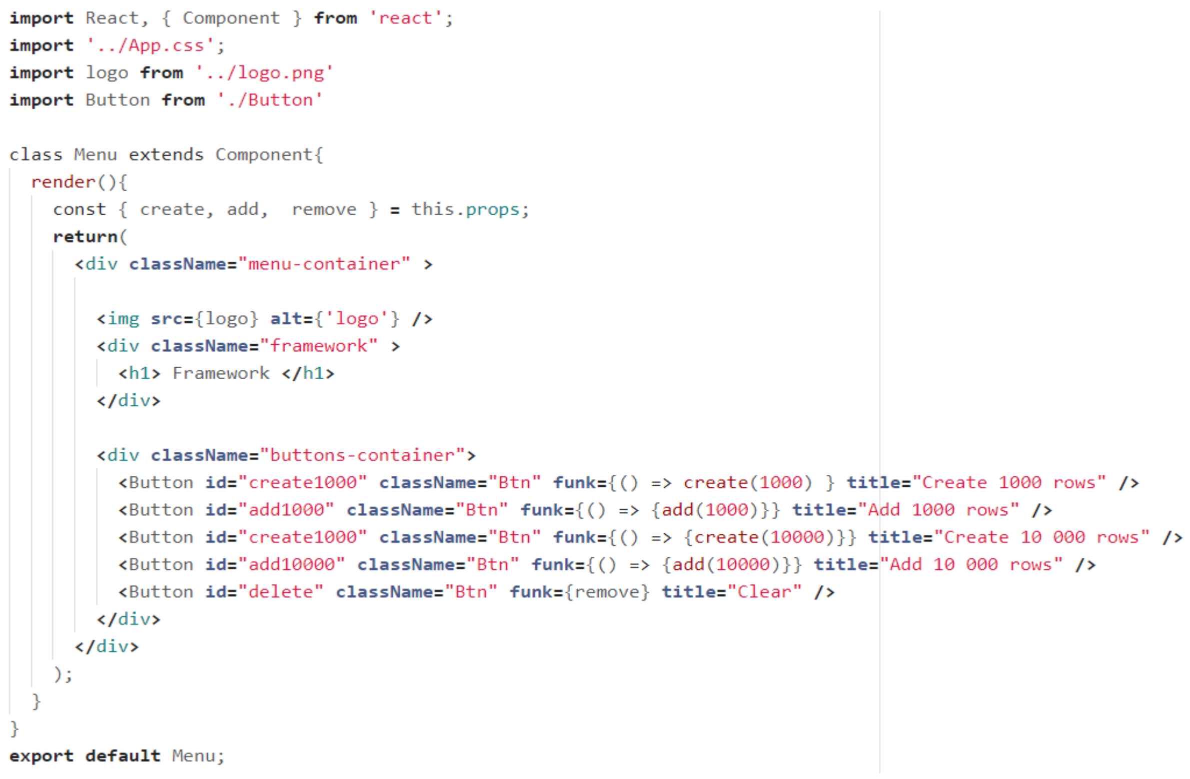


```
  render() {
    return (
      <div className="App">
        <Menu
          create={this.create}
          add={this.add}
          remove={this.remove}
        />

        <table className="data-table">
          <tbody>
            {this.state.data.map((item, i) => (
              <Row key={i} item={item} ></Row>
            ))}
          </tbody>
        </table>
      </div>
    );
  }
}
export default App;
```

Figure 21. Second half of an App component

4.4.2 Menu component

Menu component is a stateless component, but it is declared as a class, to illustrate the possibilities.

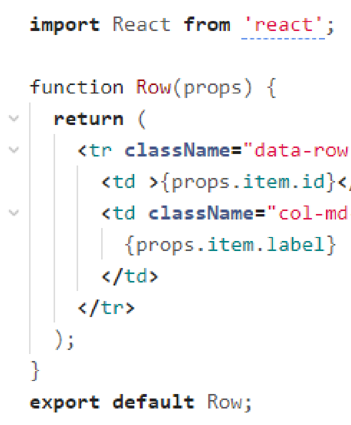


```
import React, { Component } from 'react';
import '../App.css';
import logo from '../logo.png'
import Button from './Button'

class Menu extends Component{
  render(){
    const { create, add,  remove } = this.props;
    return(
      <div className="menu-container" >

        <img src={logo} alt={'logo'} />
        <div className="framework" >
          <h1> Framework </h1>
        </div>

        <div className="buttons-container">
          <Button id="create1000" className="Btn" funk={() => create(1000) } title="Create 1000 rows" />
          <Button id="add1000" className="Btn" funk={() => {add(1000)}} title="Add 1000 rows" />
          <Button id="create1000" className="Btn" funk={() => {create(10000)}} title="Create 10 000 rows" />
          <Button id="add10000" className="Btn" funk={() => {add(10000)}} title="Add 10 000 rows" />
          <Button id="delete" className="Btn" funk={remove} title="Clear" />
        </div>
      </div>
    );
  }
}
export default Menu;
```

Figure 22. Menu component

4.4.3 Row component

Row component is a stateless functional component that takes props as an input and returns a view. There is no need for a class component if the component does not have a state.



```
import React from 'react';

function Row(props) {
  return (
    <tr className="data-row ">
      <td >{props.item.id}</td>
      <td className="col-md-4">
        {props.item.label}
      </td>
    </tr>
  );
}
export default Row;
```

Figure 23. Row component

4.4.4 Test results

Table 3. shows the React application production build speed tests results in milliseconds. The last column of the table shows the average of the three tests run and is rounded down to full number with no decimal places. The last column will be later used in the analysis part of this thesis.

Table 3. React speed test results

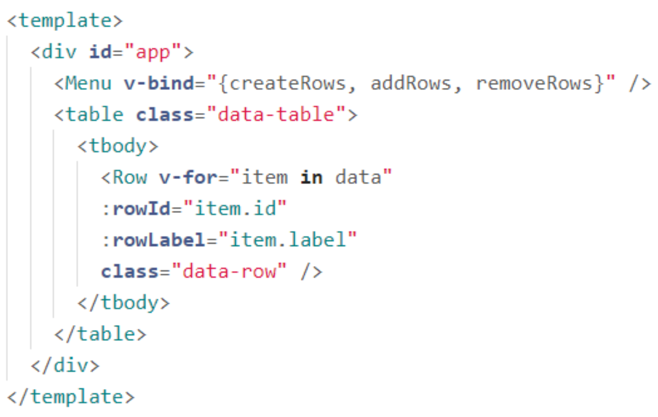
React	Test 1.	Test 2.	Test 3.	Average
Load Page	259.4	268.8	279.4	269
Create 1000	400.9	400.6	459.1	420
Re-create 1000	174.2	206	191.2	190
Add 1000	364.2	364.5	380.5	370
Create 10 000	2787.1	2978.9	2831.7	2866
Re-create 10 000	1181.7	1201.3	1145.7	1176
Add 10 000	3888.7	3906.7	3766.1	3854
Remove all	589.4	619.9	611.3	607

4.5 Vue

Vue application is divided into four components. The *App* is the main and the root component having two children components: *Menu* and *Row*. Vue application production build was 624KB and is hosted on AWS for testing and on writers GitHub for the code review (AWS c; Elar Saks 2019).

4.5.1 Root component: App

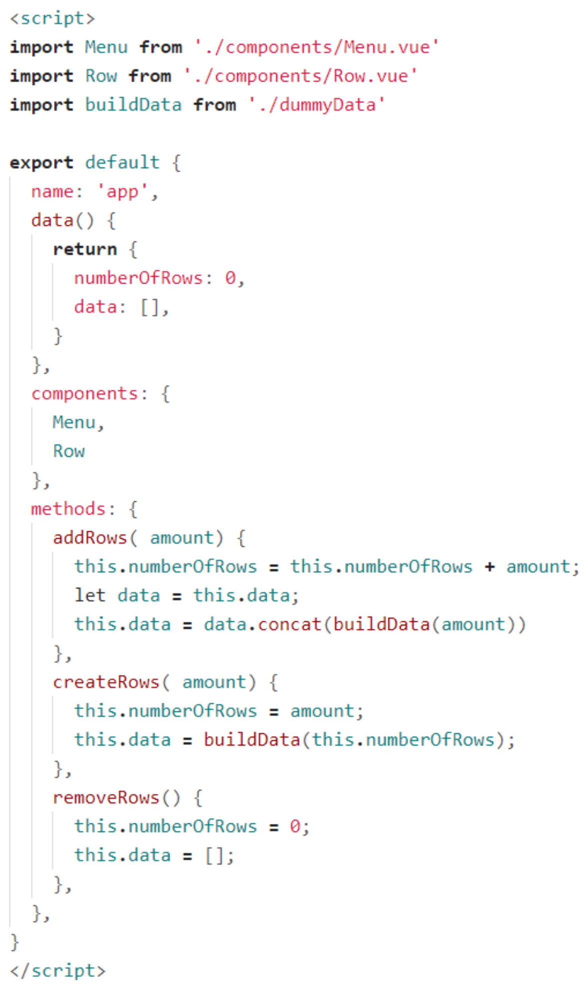
The *App* is a root component of the Vue application and is a parent to the rest of the components. It starts with a *view* template part, as seen in figure 24. The template is made up from *App* div, containing *Menu* component and a table, that gets *Row* components rendered into it based on the data property of an *App* component.



```
<template>
  <div id="app">
    <Menu v-bind="{createRows, addRows, removeRows}" />
    <table class="data-table">
      <tbody>
        <Row v-for="item in data"
        :rowId="item.id"
        :rowLabel="item.label"
        class="data-row" />
      </tbody>
    </table>
  </div>
</template>
```

Figure 24. App.vue template part

The second half of an *App* component is made up from a script tag consisting JavaScript that makes up a Vue instance and its properties. Code from the script tag can be seen in figure 25. It starts with importing the child components and the *buildData* function from. Then the *App* component is exported as an object consisting of a *data* function, *components* object, and *methods* object.

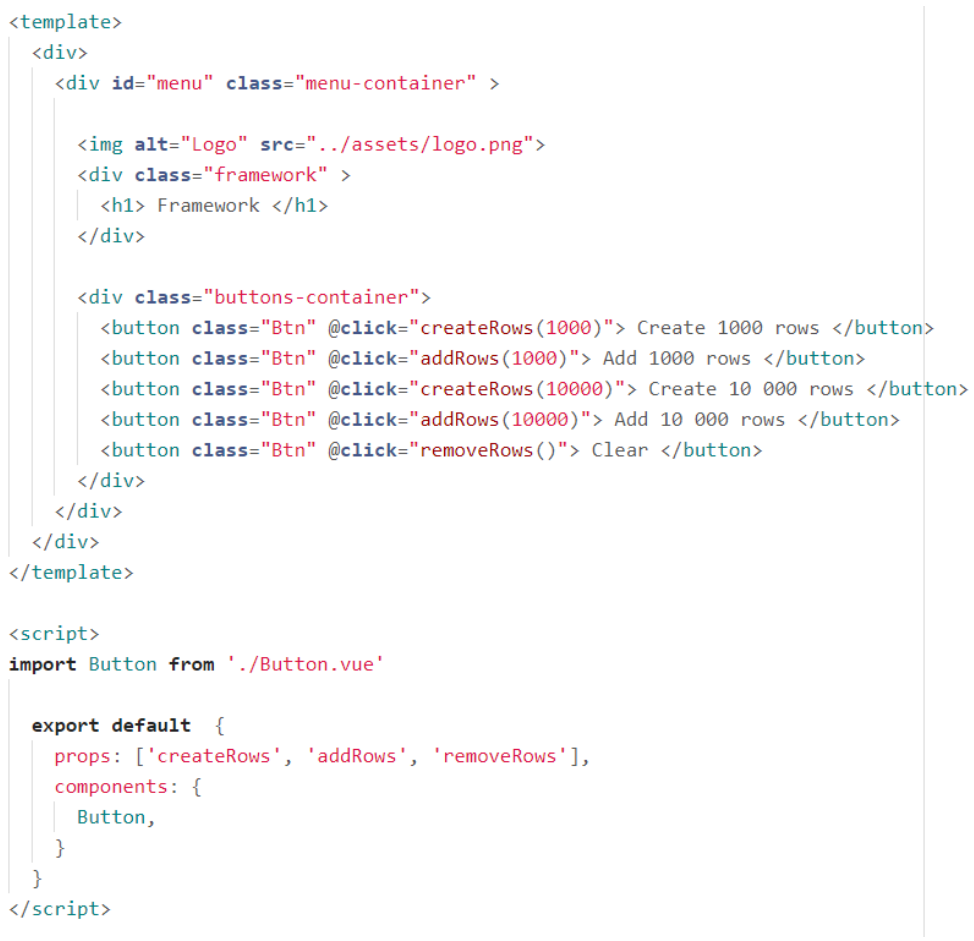


```
<script>
import Menu from './components/Menu.vue'
import Row from './components/Row.vue'
import buildData from './dummyData'

export default {
  name: 'app',
  data() {
    return {
      numberOfRows: 0,
      data: [],
    }
  },
  components: {
    Menu,
    Row
  },
  methods: {
    addRows( amount) {
      this.numberOfRows = this.numberOfRows + amount;
      let data = this.data;
      this.data = data.concat(buildData(amount))
    },
    createRows( amount) {
      this.numberOfRows = amount;
      this.data = buildData(this.numberOfRows);
    },
    removeRows() {
      this.numberOfRows = 0;
      this.data = [];
    },
  },
}
</script>
```

Figure 25. Second half of an App.vue

4.5.2 Menu component

Similar to the *App* component, the *Menu* component is made up of two parts: the template part consisting of HTML and the script tag consisting JavaScript.

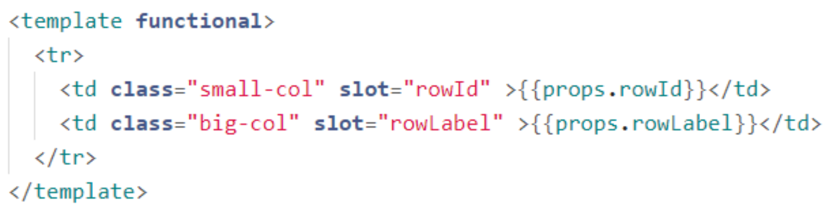


```
<template>
  <div>
    <div id="menu" class="menu-container" >

      <img alt="Logo" src="../assets/logo.png">
      <div class="framework" >
        <h1> Framework </h1>
      </div>

      <div class="buttons-container">
        <button class="Btn" @click="createRows(1000)"> Create 1000 rows </button>
        <button class="Btn" @click="addRows(1000)"> Add 1000 rows </button>
        <button class="Btn" @click="createRows(10000)"> Create 10 000 rows </button>
        <button class="Btn" @click="addRows(10000)"> Add 10 000 rows </button>
        <button class="Btn" @click="removeRows()"> Clear </button>
      </div>
    </div>
  </div>
</template>

<script>
import Button from './Button.vue'

  export default  {
    props: ['createRows', 'addRows', 'removeRows'],
    components: {
      Button,
    }
  }
</script>
```

Figure 26. Menu component

4.5.3 Row component.

Row component doesn't have any application logic and therefore, is written as a functional component that only renders the view. This makes the code much shorter and cleaner, as it can be seen in Figure 27.



```
<template functional>
  <tr>
    <td class="small-col" slot="rowId" >{{props.rowId}}</td>
    <td class="big-col" slot="rowLabel" >{{props.rowLabel}}</td>
  </tr>
</template>
```

Figure 27. Row component

4.5.4 Test results

Table 4. shows Vue application production build speed tests results in milliseconds. The last column of the table shows the average of the three tests run and is rounded down to full number with no decimal places. The last column will be later used in the analysis part of this study.

Table 4. Vue speed test results

Vue	Test 1.	Test 2.	Test 3.	Average
Load Page	233.3	264.6	240.44	246
Create 1000	248.1	213.5	234.2	232
Re-create 1000	125.6	111.8	112.6	117
Add 1000	193.7	180.7	180.2	185
Create 10 000	1446.8	1448.1	1446.7	1447
Re-create 10 000	553.2	533.9	553.9	547
Add 10 000	1423.2	1452.3	1433.1	1436
Remove all	567.6	334.8	613.9	505

5 Evaluation

Comparing software tools, especially frameworks, is difficult, as they have a wide variety of use cases and domains, some are better in one, and some are better in others. In this chapter, Angular, React, and Vue are compared in three categories: popularity amongst users, frameworks learning curve, and application loading speeds.

5.1 Popularity

In order to compare the popularity, the data from industry-leading platforms, such as GitHub, NPM & Stack Overflow, was collected and gets analysed here.

5.1.1 GitHub data

GitHub gives a unique insight into the JavaScript market, as it provides information about what users have "liked" the most, what framework have most developers working on it, and what framework has the most issues opened about it. This data can be somewhat biased, as older frameworks have had more time to collect followers and more complex frameworks have had time to get more issues opened about them. Keeping that in mind, one could look at GitHub data in table 1. and notice the following.

Vue has slightly more stars than React, with 133 thousand over 125 thousand stars. That is about 6% more and not so big difference. Angular, on the other hand, sits as an outlier, with 46 thousand stars. That is more than two times less than its competitors.

React is the most forked framework out of these three, with 22 thousand forks vs. Vue, 18 thousand forks, and Angular 12 thousand forks. Although the differences between numbers of forks are not as drastic as the difference in the number of stars, it is big enough to be reliable.

Based on the issues opened about it, Angular is either the most popular out of these three or the most problematic one. Angular had 2331 issues opened about it, vs. React 453 issues and vs. Vue 184 issues.

Angular is the oldest of these frameworks, followed by the React and then the Vue. That could explain a large number of issues opened about it, but then looking at the rest of the data, it is safe to say that Angular is the least popular framework on GitHub, out of these 3. Vue is more liked than React, but React more people working on it, so they both are about equally popular.

5.1.2 NPM Trends

Data from NPM Trends shows how many packages were downloaded for each framework, through time. This data might be a bit misleading as some frameworks are bigger and need fewer supporting packages by third parties. For example, Angular is much bigger and has much more features than React and Vue, therefore getting fewer downloads in NPM trends. Yet looking at figure 1., it is evident that React has multiple times more downloads than the other two frameworks. Vue has slightly more downloads than Angular and is showing steady growth, but the difference is not as noticeable as with React.

5.1.3 Stack Overflow trends

Stack Overflow is a cloud platform connecting the global software developer community, by enabling developers to share code and as help from each other. Looking at Stack Overflow data gives a good indication about what gets used and talked about amongst developers all around the world. Although, some more difficult frameworks might get more questions asked about than the easier ones, and therefore look more popular than they are.

Figure 2. displays the percentage of questions that were asked about each framework between 2009 and 2019. We can see that Angular popularity has peaked in 2017, with 3% out of all questions on Stack Overload being about it. It has stayed about the same, since then. Angular is closely followed by the React, that peaked in 2018 with 2,5% out of all questions. Vues growth has not been as aggressive and has started out later, with its highest being 0.8% in 2019.

Although it seems, as Angular and the React have achieved their potential on Stack Overflow and that Vue is still growing, the gap is still large enough that it is safe to say that Angular and the React are the more popular ones on that platform. As the difference between them, two is so small; it is hard to say which one is more popular.

5.2 Learning Curve

In order to understand the difficulty of learning each framework, frameworks are compared in syntax, architecture, data management, lifecycle, and in third-party libraries. In this chapter, data collected about learning each framework gets analysed.

5.2.1 Syntax

Between those three frameworks, syntax wise, Angular is the most difficult to learn, as it uses TypeScript. New user not only has to learn a new framework, but also a new superset of JavaScript, to be even able to use this framework.

React, and Vue are much closer to each other, as far as learning curve goes. Although Vue syntax could be considered easier than React, as its syntax is valid HTML and looks much more like a traditional webpage built-in HTML, CSS, and JavaScript. React uses JSX syntax, that is easy to learn for someone having a solid foundation in HTML and JavaScript, yet it might feel a bit harder to get used than Vue syntax.

5.2.2 Architecture

All three frameworks of architectures are quite similar. The application can be imagined as a component tree where on the top is the root component, that can have an unlimited hierarchy of child components. The only exception being Angular, whose has everything same, but its components are also divided into modules, that contain supporting code files related to components belonging to a certain module.

So, it is safe to say that learning the architecture of React and Vue is equally challenging, while Angular architecture is a little bit more complicated.

5.2.3 Data management

In all three frameworks, data flows downward from component to component in the component tree. Data passed by the parent component becomes a property of a child component, that cannot be mutilated.

Angular differs from React and Vue by having two-way data binding. In Angular, child components are able to pass data to parent components. So, data coming down from component tree can be received, changed, and passed back in from a child component, making building the dynamic website much more comfortable.

In React and Vue, child components cannot pass data to parents, whit out using third party libraries.

5.2.4 Lifecycle

All three frameworks have a similar lifecycle with similar lifecycle hooks. The easiest of them to learn would be Vue, as Vue official technical documentation provides a user with a full process map of a component lifecycle, with all the lifecycle methods included.

The second easiest to learn would be Angular as its official documentation has a full list of lifecycle hooks from beginning till the end.

The hardest of them to learn in lifecycle wise would be React. Although its official documentation explains clearly the way the component lifecycle works and learning it is relatively easy, it is hard to find the full map or list of all the lifecycle hooks.

5.2.5 Third party packages

All three frameworks take advantage of Node Package Manager, and it can be used to install third-party libraries. That makes installing the extensions to each framework equally comfortable. It is worth noting that it is more beneficial for React and Vue than to Angular. Installing third-party libraries allows users to add some vital functionality to React and Vue, that already exists in Angular. Good examples are router and data storage outside of components.

5.3 Performance

All frameworks are not built the same, and they might perform differently for different type and size of applications. Yet the small applications built for this study should provide some hint about the performance of the frameworks in question.

As can be seen from code examples, the application architecture is quite similar for all three frameworks, as well as the amount of code written by users is quite the same. That might change in case scope and scale of applications would grow, but for these types of small applications, there is no remarkable difference.

There was a big difference in application production build sizes, though, as can be seen from table 5. While React and Vue production builds were a quite similar size, the Angular final application was a lot bigger than the rest. The angular distro was 15.7 Mega bites, that is 15 700 Kilobytes, making it 25 times larger than Vue and about 27 times larger than React production build.

It might be so because Angular is built for much bigger and more complex applications and comes with a bigger codebase. The application production builds might even out between frameworks, when building larger applications. But it is definitely something to keep in mind when building small applications in regions and domains where internet speed and bandwidth counts.

Table 5. Production build sizes

React	587 KB
Vue	624 KB
Angular	15.7 MB

For each application built in each framework, three performance tests were run. Speed test results were collected into table 2., table 3., and table 4. The averages of three test results for each framework were combined into table 6.

Table 6. most left column states what function speed is measured. The results are displayed in the columns on the right in milliseconds. Columns are color-coded, with the green marking the fastest function running speed, yellow the second fastest and red, the slowest. Two last rows are used for the analysis.

The row named “Total” calculates the total time used to run all the functions per framework. The last row, named “multiplier” is used to illustrate the total running speed related to the fastest framework (Vue). Multiplier takes total time spent by Vue and divides other frameworks total times with it, illustrating how many times was any given framework slower than Vue.

With the Angular having tens of times larger production build, it feels safe to assume that it would be the slowest of the three, but looking at table 6. it can be seen that it is not always the case. Angular was the slowest when loading the page, but the difference in loading speed was not as huge as the difference in the size of the application. It took Angular around 400 milli-seconds to load the landing page, while Vue and React both stayed into the middle of two hundreds of milliseconds. Angular also managed to beat the React in creating and adding 1000 rows. Rest of the speed test went for the Angular as expected, it being more than 2 times slower than the closest competitor React.

Although Vue had slightly bigger production build than React, it beat both Angular and React in all the tests, in total being five times faster than Angular and two times faster than React. React was around two and a half times faster than Angular.

Table 6. Frameworks loading speeds

	Angular	**React**	**Vue**
Load Page	403	269	246
Create 1000	384	420	232
Re-create 1000	331	190	117
Add 1000	254	370	185
Create 10 000	6361	2866	1447
Re-create 10 000	7427	1176	547
Add 10 000	7623	3854	1436
Remove all	2196	607	505
Total	24978	9752	4715
Multiplier	5.30	2.07	1.00

6 Conclusion

The aim of this study was to compare the three of the most popular JavaScript frameworks in popularity, the difficulty of learning and in performance, so that the reader could make an educated decision when choosing which framework to learn or to use for the project.

6.1 Popularity

During this study, the information about frameworks popularity was collected from the major cloud service providers, such as GitHub, NPM, and Stack Overflow.

On the GitHub, Vue proved to be the most popular one, having just a minor lead ahead of React, with a few more stars and fewer issues. The Angular was the definite loser on GitHub with half fewer stars and multiple times more issues than others.

React gets multiple times more packages downloaded via NPM than other two frameworks combined. It might be that Angular just don't need so many third-party libraries as React and Vue is still a new and growing framework, but what is sure, is that React is the most popular framework on the NPM, by far.

It seems that Angular's popularity on that Stack Overflow has peaked and is about to be overtaken by React. React and Angular share the lead, with both of them individually making up between 2.5% and 3% of total questions asked. Vue, while in steady growth, makes up only 8% of total questions asked.

With React totally dominating the NPM and sharing the lead, both on Github and on Stack Overflow, it is safe to say that as for 2019, React is the most popular JavaScript library.

6.2 Learning curve

Learning a new framework is subjective to the individual learner and cannot be measured quantitatively. Results of this study are the sole opinion of the writer and should be taken with a healthy dose of criticism. The frameworks learning curve was compared in syntax, architecture, data management, lifecycle, and in the ease of using third-party libraries.

Vue had the easiest syntax, as it is the only one using pure HTML, CSS, and vanilla JavaScript. Architecture wise, Vue, and React are the easier than Angular, that also uses modules in addition to the components. Angular also has the built-in two-way data binding, that can only be achieved by other frameworks, through the use of third-party libraries. All three frameworks have a similar lifecycle and are capable of installing third-party libraries via NPM.

Vue is the easiest one to learn for the newcomer, followed by React with little more complex syntax and worse documentation. Angular is by far the biggest and hardest to learn out of these three.

6.3 Performance

To compare the framework's performance a simple single page application was built in each framework. Applications were then tested in loading speeds.

Vue was the fastest performing framework in all tests. In total, Vue was five times faster than Angular and over two times faster than React. React was faster than Angular on six tests out of eight. React also had the smallest production build 587KB, Vue production build was 624KB and Angular was by far the largest with 15.7MB.

It is worth noting that frameworks performance in speed might change as the application size and complexity changes. Therefore, more testing would be required with more complex applications.

6.4 Final conclusion

For getting a job, React being the most popular one, is probably the best framework to learn. The easiest to learn and the fastest performing is the Vue. Angular is the hardest to learn and the slowest performing framework. But what Angular lacks in speed and in ease of learning, it makes up in its usability and might outperform other frameworks in building large scale applications. More research would be needed to compare frameworks in building large, more complex multipage applications.

References

Angular a. Home. URL: https://angular.io/. Accessed: 21.03.2019.

Angular b. Guide/architecture. URL: https://angular.io/guide/architecture. Accessed: 13.07.2019.

Angular c. Guide/template-syntax. URL: https://angular.io/guide/template-syntax. Accessed: 13.07.2019.

Angular d. Guide/architecture-modules. URL: https://angular.io/guide/architecture-modules. Accessed: 13.07.2019.

Angular e. Guide/template-syntax. URL: https://angular.io/guide/template-syntax. Accessed: 13.07.2019.

Angular f. Guide/architecture-modules. URL: https://angular.io/guide/architecture-modules. Accessed: 13.07.2019.

Angular g. Guide/architecture-components. URL: https://angular.io/guide/architecture-components#data-binding. Accessed: 13.07.2019.

Angular h. Guide/lifecycle-hooks. URL: https://angular.io/guide/lifecycle-hooks. Accessed: 13.07.2019.

Angular i. Guide/using-libraries. URL: https://angular.io/guide/using-libraries. Accessed: 13.07.2019.

Angular j. Material. URL: http://material.angular.io. Accessed: 13.07.2019.

AWS a. Angular. URL: http://thesis-angular.s3-website.eu-north-1.amazonaws.com. Accessed: 4.09.2019

AWS b. React. URL: http://thesis-react.s3-website.eu-north-1.amazonaws.com. Accessed: 4.09.2019

AWS c. Angular. URL: http://thesis-vue.s3-website.eu-north-1.amazonaws.com/. Accessed: 4.09.2019

Cromwell, V. 2017. Between the Wires: An interview with Vue.js creator Evan You. URL: https://medium.freecodecamp.org/between-the-wires-an-interview-with-vue-js-creator-evan-you-e383cbf57cc4. Accessed: 21.03.2019.

Dziwoki, M. 2017. URL: https://gorrion.io/blog/angularjs-vs-angular. Accessed: 22.03.2019.

Edureka. React-redux tutorial. URL: https://www.edureka.co/blog/react-redux-tutorial/. Accessed: 22.03.2019.

Elar Saks. 2019. Thesis. URL: https://github.com/elarsaks/Thesis/tree/master/react_test. Accessed: 4.09.2019.

GitHub a. Home. URL: https://github.com/. Accessed: 22.03.2019.

GitHub b. Angular. URL: https://github.com/angular/angular. Accessed: 22.03.2019.

GitHub c. React. URL: https://github.com/facebook/react. Accessed: 22.03.2019.

GitHub d. Vue. URL: https://github.com/vuejs/vue. Accessed: 22.03.2019.

GitHub e. About stars. URL: https://help.github.com/en/articles/about-stars. Accessed: 22.03.2019.

GitHub f. About forks. URL: https://help.github.com/en/articles/about-forks. Accessed: 22.03.2019.

GitHub g. About issues. URL: https://help.github.com/en/articles/about-issues. Accessed: 22.03.2019.

GitHub g. About issues. URL: https://help.github.com/en/articles/about-issues. Accessed: 22.03.2019

Gudelli, A. 2017. History of Angular. URL: https://www.angularjswiki.com/angularjs/history-of-angularjs/. Accessed: 21.03.2019.

Npm. About npm. URL: https://docs.npmjs.com/about-npm/. Accessed: 21.03.2019.

Npm trends. Angular vs React vs Vue. URL: https://www.npmtrends.com/angular-vs-react-vs-vue. Accessed: 21.03.2019.

PngKey. Virtual Dom – React Virtual Dom. URL: https://www.pngkey.com/maxpic/u2e6t4u2o0o0o0o0/. Accessed: 21.03.2019.

React a. Home. URL: https://reactjs.org/. Accessed: 21.03.2019.

React b. Introducing JSX. URL: https://reactjs.org/docs/introducing-jsx.HTML. Accessed: 07.04.2019.

React c. Rendering Elements. URL: https://reactjs.org/docs/rendering-elements.HTML. Accessed: 07.04.2019

React d. Virtual DOM and internals. URL: https://reactjs.org/docs/faq-internals.HTML. Accessed: 07.04.2019.

React e. Components and Props. URL: https://reactjs.org/docs/components-and-props.HTML. Accessed: 07.04.2019.

React f. State and Lifecycle. URL: https://reactjs.org/docs/state-and-lifecycle.HTML. Accessed: 07.04.2019.

Redux a. Introduction. URL: https://redux.js.org/introduction/getting-started. Accessed: 29.06.2019.

Stack Overflow a. Home. URL: https://insights.stackoverflow.com/. Accessed: 21.03.2019

Stack Overflow b. Stack Overflow trends. URL: https://insights.stackoverflow.com/trends?tags=r%2Cstatistics. Accessed: 21.03.2019

VueJS a. Guide/Syntax. URL: https://vuejs.org/v2/guide/syntax.HTML. Accessed: 30.06.2019

VueJS b. Guide/components. URL: https://vuejs.org/v2/guide/components.HTML. Accessed: 30.06.2019

VueJS c. Images/components. URL: https://vuejs.org/images/components.png. Accessed: 30.06.2019

VueJS d. Guide/components/Data must be function. URL: https://vuejs.org/v2/guide/components.HTML#data-Must-Be-a-Function. Accessed: 30.06.2019

VueJS e. Guide/components/Passing data to child components with props. URL: https://vuejs.org/v2/guide/components.HTML#Passing-Data-to-Child-Components-with-Props. Accessed: 30.06.2019

VueJS f. Guide/guide/instance. URL: https://vuejs.org/v2/guide/instance.HTML#Creating-a-Vue-Instance. Accessed: 30.06.2019

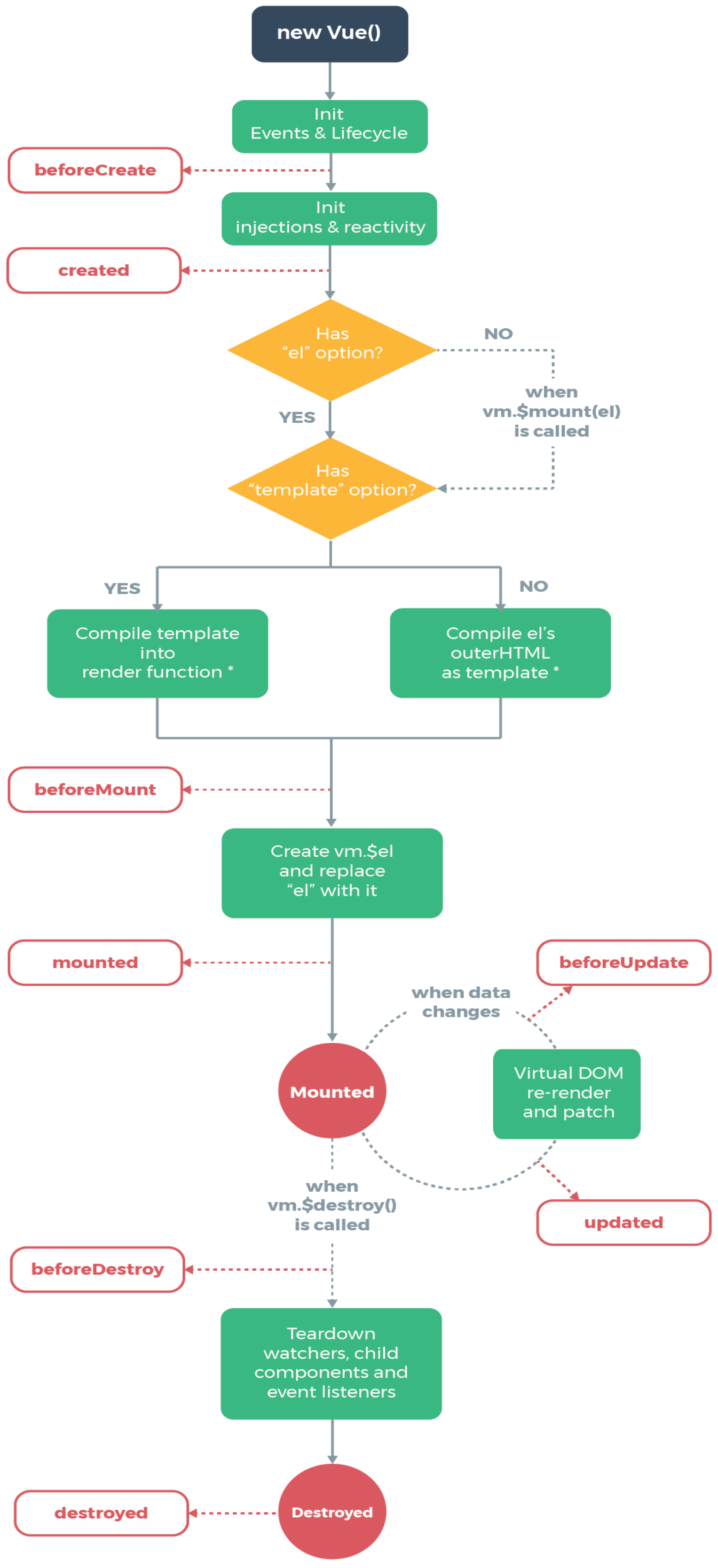
VueX a. What is VueX? URL: https://vuex.vuejs.org/. Accessed: 30.06.2019

Wikipedia a. React (JavaScript library). URL: https://en.wikipedia.org/wiki/React_(JavaScript_library) Accessed: 21.03.2019.

Wikipedia b. Microsoft TypeScript. URL: https://en.wikipedia.org/wiki/Microsoft_TypeScript Accessed: 13.07.2019

Appendices

Appendix 1. Vue Application Lifecycle (VueJS f).


new Vue()
Init Events & Lifecycle
beforeCreate
Init injections & reactivity
created
Has "el" option?
NO
when vm.$mount(el) is called
YES
Has "template" option?
YES
NO
Compile template into render function *
Compile el's outerHTML as template *
beforeMount
Create vm.$el and replace "el" with it
mounted
beforeUpdate
when data changes
Mounted
Virtual DOM re-render and patch
when vm.$destroy() is called
updated
beforeDestroy
Teardown watchers, child components and event listeners
destroyed
Destroyed