

**Czech University of Life Sciences Prague**

**Faculty of Economics and Management**

**Department of Information Technologies**



**Bachelor Thesis**

**ReactJS as a tool for component-based web application  
development**

**Adam Peklák**

**© 2019 CULS Prague**

## BACHELOR THESIS ASSIGNMENT

Adam Peklák

Informatics

Thesis title

**ReactJS as a tool for component-based web applications development**

---

### Objectives of thesis

The main objective of the thesis is to compare two of the most popular state management libraries that are used with ReactJS – Redux and MobX.

Partial objectives:

- Describe characteristics of JavaScript, and more generally, of functional programming
- Analyze architecture used by ReactJS
- Analyze data flow and state management

### Methodology

At the beginning, a rather extensive research of the topic is necessary in order to grasp all the aspects of the issue. As a practical part, two identical simple applications will be developed, one using Redux, and the other one using MobX. A comparison of the final results, the speed of development, learning curve and other qualities will be performed. Scientific methods such as comparison, analysis, and deduction will be used. Based on the research from the literature and also on the practical part of the thesis, a final conclusion will be formulated.

## The proposed extent of the thesis

35 – 45 stran

## Keywords

javascript, reactJS, mobX, redux, state management

---

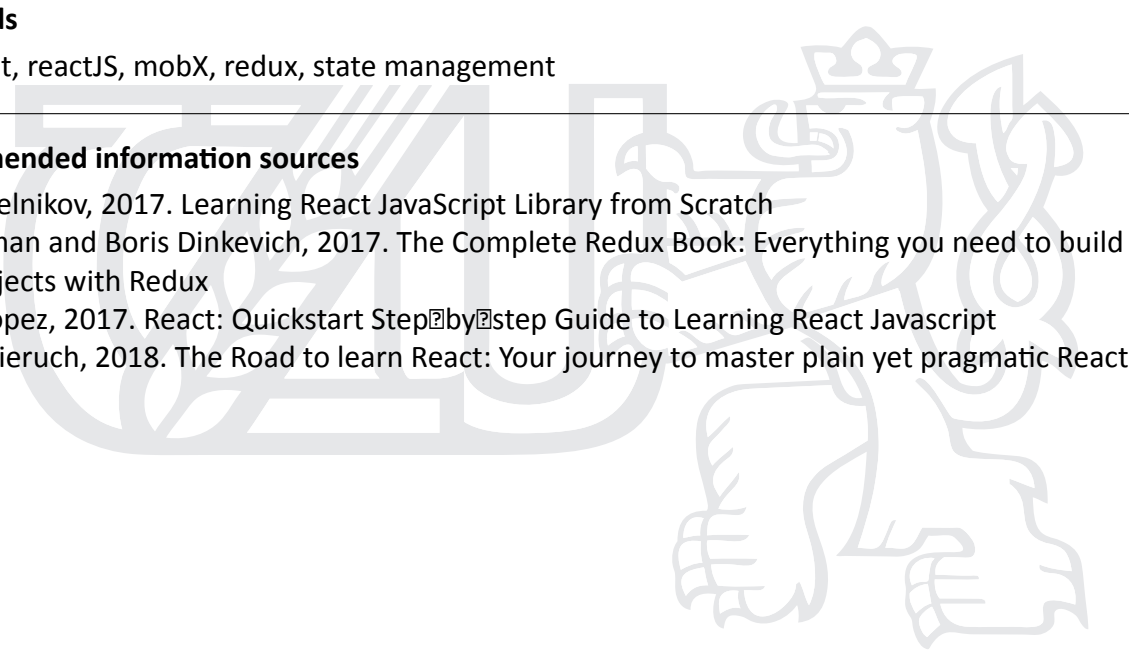
## Recommended information sources

Greg Sidelnikov, 2017. Learning React JavaScript Library from Scratch

Ilya Gelman and Boris Dinkevich, 2017. The Complete Redux Book: Everything you need to build real projects with Redux

Lionel Lopez, 2017. React: Quickstart Step-by-step Guide to Learning React Javascript

Robin Wieruch, 2018. The Road to learn React: Your journey to master plain yet pragmatic React.js



---

## Expected date of thesis defence

2018/19 SS – FEM

## The Bachelor Thesis Supervisor

Ing. Jan Masner, Ph.D.

## Supervising department

Department of Information Technologies

Electronic approval: 11. 9. 2018

**Ing. Jiří Vaněk, Ph.D.**

Head of department

Electronic approval: 19. 10. 2018

**Ing. Martin Pelikán, Ph.D.**

Dean

Prague on 13. 03. 2019

### **Declaration**

I declare that I have worked on my bachelor thesis titled “ReactJS as a tool for component-based web application development” by myself and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the thesis does not break copyrights of any their person.

In Prague on 12. 3. 2019

---

## **Acknowledgement**

I would like to thank Jan Masner and my friends for their advice and support during my work on this thesis.

# **ReactJS as a tool for component-based web application development**

## **Abstract**

ReactJS is a popular open-source JavaScript library used for programming interactive user interfaces for single page web applications. It specifically handles the view layer from the MVC application model. React has component-based architecture, which encapsulates reusable individual pieces of the UI (components) into independent micro-systems.

This thesis examines two of the popular state management libraries for React, MobX and Redux. These libraries keep state object in application store and provide access to it for any components necessary, rather than passing the data through tree structure of the components, as is the native React way. The libraries have been examined and compared to determine advantages and disadvantages of both.

Two web applications, which are in every aspect identical, apart from one using MobX for state management and the other one using Redux, have been compared from multiple points of view such as performance, complexity, code length and learning curve.

It has been found, that even though Redux's popularity far surpasses MobX, there are use-cases, such as this application, where MobX is superior to Redux in performance and speed of development and is overall a better choice.

**Keywords:** JavaScript, React, ReactJS, MobX, Redux, state management, web application

# Table of Content

|          |                                   |           |
|----------|-----------------------------------|-----------|
| <b>1</b> | <b>Introduction</b>               | <b>11</b> |
| <b>2</b> | <b>Objectives and Methodology</b> | <b>12</b> |
| 2.1      | Objectives                        | 12        |
| 2.2      | Methodology                       | 12        |
| <b>3</b> | <b>Literature Review</b>          | <b>13</b> |
| 3.1      | JavaScript                        | 13        |
| 3.1.1    | How does JavaScript work          | 13        |
| 3.2      | What is ReactJS                   | 14        |
| 3.2.1    | Advantages of React               | 15        |
| 3.2.2    | Disadvantages of React            | 16        |
| 3.3      | React Environment Setup           | 17        |
| 3.3.1    | Node.js                           | 17        |
| 3.3.2    | CDN Links                         | 17        |
| 3.3.3    | NPM                               | 17        |
| 3.3.3.1  | Create-react-app                  | 18        |
| 3.3.3.2  | Package.json                      | 18        |
| 3.4      | Architecture                      | 19        |
| 3.4.1    | Components                        | 19        |
| 3.4.2    | JSX                               | 19        |
| 3.4.3    | Virtual DOM                       | 20        |
| 3.4.4    | Flux                              | 20        |
| 3.4.4.1  | Flux vs MVC                       | 20        |
| 3.4.5    | Lifecycle Methods                 | 22        |
| 3.5      | State & Props                     | 22        |
| 3.6      | Redux                             | 23        |
| 3.7      | MobX                              | 25        |
| <b>4</b> | <b>Practical Part</b>             | <b>27</b> |
| 4.1      | React Application                 | 27        |
| 4.1.1    | Introduction                      | 27        |
| 4.1.2    | Application Description           | 27        |
| 4.1.3    | Project Structure                 | 28        |
| 4.1.4    | NPM Packages                      | 29        |
| 4.2      | State management                  | 30        |
| 4.2.1    | Complexity                        | 30        |
| 4.2.1.1  | Redux                             | 30        |

|          |                                    |           |
|----------|------------------------------------|-----------|
| 4.2.1.2  | MobX.....                          | 31        |
| 4.2.2    | Performance .....                  | 32        |
| 4.2.2.1  | Load Time .....                    | 33        |
| 4.2.2.2  | Assigning Task to Sector.....      | 33        |
| 4.2.3    | Learning Curve .....               | 34        |
| 4.2.4    | Code length .....                  | 34        |
| <b>5</b> | <b>Results and Discussion.....</b> | <b>35</b> |
| 5.1      | Results .....                      | 35        |
| 5.1.1    | Complexity.....                    | 35        |
| 5.1.2    | Performance .....                  | 36        |
| 5.1.2.1  | Load Time .....                    | 36        |
| 5.1.2.2  | Modifying state data .....         | 36        |
| 5.1.3    | Code length .....                  | 37        |
| 5.1.4    | Learning Curve .....               | 38        |
| 5.1.5    | Summary .....                      | 38        |
| 5.2      | Discussion .....                   | 39        |
| <b>6</b> | <b>Conclusion.....</b>             | <b>40</b> |
| <b>7</b> | <b>References .....</b>            | <b>41</b> |
| 7.1      | Bibliography.....                  | 41        |
| <b>8</b> | <b>Attachments .....</b>           | <b>42</b> |



## List of Figures

|  |    |
|--|----|
| Figure 1: JavaScript code execution in HTML ( <a href="https://static.makeuseof.com/wp-content/uploads/2017/09/how-works.png">https://static.makeuseof.com/wp-content/uploads/2017/09/how-works.png</a> )..... | 14 |
| Figure 2: React Advantages (screenshot from <a href="https://2018.stateofjs.com/front-end-frameworks/react/">https://2018.stateofjs.com/front-end-frameworks/react/</a> )<br>.....                             | 16 |
| Figure 3: React Disadvantages (Screenshot from <a href="https://2018.stateofjs.com/front-end-frameworks/react/">https://2018.stateofjs.com/front-end-frameworks/react/</a> ).....                              | 17 |
| Figure 4: MVC Architecture ( <a href="https://cdn-images-1.medium.com/max/800/0*Ift_ZYTPqpLd4AP5.png">https://cdn-images-1.medium.com/max/800/0*Ift_ZYTPqpLd4AP5.png</a> )<br>.....                            | 21 |
| Figure 5: FLUX Architecture ( <a href="https://cdn-images-1.medium.com/max/800/0*M-SY5eww-OW9xbMs.png">https://cdn-images-1.medium.com/max/800/0*M-SY5eww-OW9xbMs.png</a> ) .....                              | 21 |
| Figure 6: Lifecycle Methods ( <a href="https://cdn-images-1.medium.com/max/800/1*0RTMM_pQEpO7kJ-ex80MEA.png">https://cdn-images-1.medium.com/max/800/1*0RTMM_pQEpO7kJ-ex80MEA.png</a> ) .....                  | 22 |
| Figure 7: Redux Store ( <a href="https://mobx.js.org/images/action-state-view.png">https://mobx.js.org/images/action-state-view.png</a> ) .....  | 24 |
| Figure 8: MobX Action State Views .....  | 25 |
| Figure 9: Application User Interface .....   | 28 |
| Figure 10: File structure of src/ folder .....   | 29 |
| Figure 11: App.js Render method.....   | 29 |
| Figure 12: Single Redux action from taskActions.js .....   | 30 |
| Figure 13: Redux's connect method and mapping State to Props.....  | 31 |
| Figure 14: Single MobX action from taskStore.js .....  | 31 |
| Figure 15: MobX Provider wrapped around App.....   | 32 |
| Figure 16: MobX decorators.....  | 32 |
| Figure 17: Benchmark function .....  | 33 |
| Figure 18: Load Time graph .....   | 36 |
| Figure 19: Modifying state data graph.....   | 37 |
| Figure 20: Code length comparison.....   | 38 |

## List of Tables

|  |    |
|--|----|
| Table 1: Speed of modifying data from state..... | 33 |
|--|----|

## Abbreviations and their explanations

|      |                               |
|------|-------------------------------|
| HTML | HyperText Markup Language     |
| URL  | Uniform Resource Locator      |
| API  | Application Program Interface |
| DOM  | Document Object Model         |
| JS   | JavaScript                    |
| XML  | eXtensible Markup Language    |
| JSX  | JavaScript XML                |
| CDN  | Content Delivery Network      |

# 1 Introduction

ReactJS is a popular JavaScript library used for building user interfaces for single-page web applications. It specifically handles the view layer from the MVC application model, and it is component-based, which means, that it encapsulates reusable individual pieces of the UI (components) into independent micro-systems. Component can manage its own state and can consist of arbitrary number of other components. Another feature of ReactJS is Virtual DOM, which is an internal representation of rendered webpage. When a change of the webpage occurs and needs to be rendered, virtual DOM compares the new result to the original, and only performs the necessary modifications without the need of page refresh. This ensures, that Reacts is a fast, scalable and simple JavaScript library.

This thesis first delves deeper into Reacts, it's properties and architecture (FLUX and MVC are both explained and compared), and then focuses on state management libraries Redux and MobX. Both libraries are used for managing application state, which is different from native state in stateful components, because scope of the state is not only one component, but the whole application.

Two web applications have been created as a practical part of the thesis, which are in every aspect identical, apart from the fact, that the first one is using Redux for state management, and the other one uses MobX. Aspects of the libraries, such as performance, complexity, code length and learning curve, are compared to determine the advantages and disadvantages of both.

Measured results are then analysed in order to determine, which of the libraries is a better option for specific use-case, and which is more suitable for different developers.

## **2 Objectives and Methodology**

### **2.1 Objectives**

The main objective is to compare two of the most popular state management libraries, which are used with ReactJS – Redux and MobX. This thesis examines their benefits and shortcomings and determines their suitability for different use-cases. It will be determined, which of the libraries performs better when used for state management in similar ReactJS web application, as well as other important qualities such as the level of complexity and the steepness of learning curve.

#### **Partial objectives are as follows:**

- Describe characteristics of JavaScript, and more generally, of functional programming
- Analyse architecture used by ReactJS
- Analyse data flow and state management

### **2.2 Methodology**

At the beginning, a rather extensive research of the topic was necessary in order to grasp all the aspects of the issue. As a practical part, two identical simple web applications have been developed, one using Redux, and the other one using MobX as a state management library. A comparison of the final results, the speed of development, learning curve and other qualities will be performed. Scientific methods such as comparison and analysis have been used. Based on the research from the literature and also on the practical part of the thesis, a final conclusion has been formulated, which states, which of the two libraries is more suitable for the project, and advantages and disadvantages of both.

## 3 Literature Review

### 3.1 JavaScript

JavaScript is a scripting language, which is used to dynamically manipulate content of a website. Scripting language is characterized by interpretation of the source code, rather than using a compiler to convert it into machine code. Since JavaScript is used in web pages, all web browsers contain built-in engine, which can render JavaScript code. That is the reason, why JavaScript does not require any additional programs to run and inserting the code into a HTML document is sufficient for it to get executed.

#### 3.1.1 How does JavaScript work

When a webpage is loaded by a web browser, it begins by parsing HTML code and thus creating the DOM. Whenever the parser comes across a JavaScript code, it gets send into JavaScript engine provided by the browser. There, the code is executed once HTML (and CSS) parsing is over. The whole process is displayed in Figure 1. Code execution is being done in strict order from top to bottom of the document. When a function is defined and executed, it is certain that DOM has already been built, and therefore can be modified.

JavaScript can be loaded in a website simply by using HTML script tag, where it can either refer to a JavaScript file, or the code can be embedded directly inside of the script tag. Loading the code from external files allows for code separation and reusability, and it is a good practise on any project. The loaded file does not need to be present locally on a machine but can referred to by an URL and fetched from the internet, which is especially useful for loading external JavaScript libraries. Such libraries contain pre-programmed functions and extend or improve the capabilities of the native language.

As JavaScript's main function on the webpage is to dynamically change content, it would not be practical to execute all of the code on page load. When the need arises to execute a function triggered by user action, JavaScript provides event listeners, which can be bounded with specific HTML element. Event listener waits for specific user action, for example mouse click on a button, and then executes appropriate function.

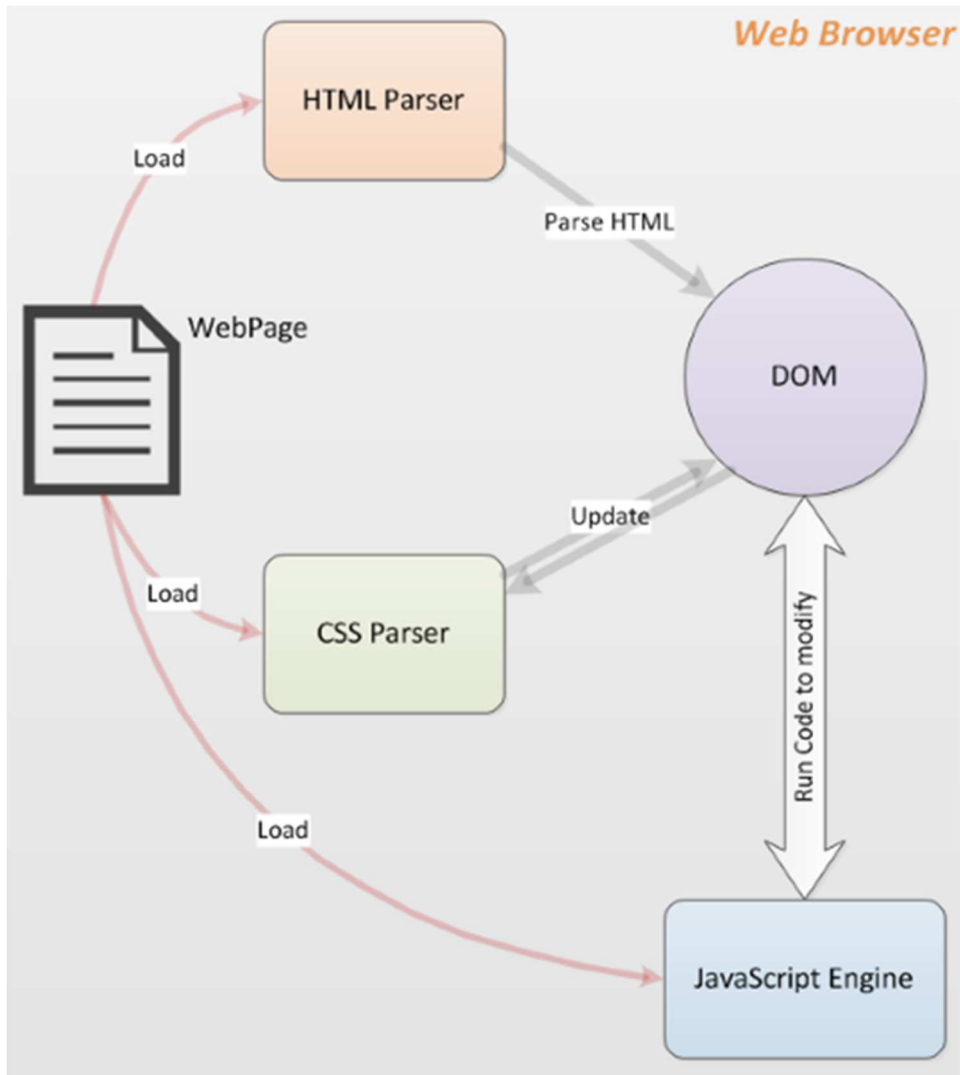


Figure 1: JavaScript code execution in HTML

### 3.2 What is ReactJS

React (also known as React.js or ReactJS) is a flexible JavaScript library responsible for the view layer of the MVC pattern in the application. It is a modern language used for building user interfaces. React uses component-based architecture, which contributes to declarative development, as developers are able to build encapsulated components that manage their own state. React applications are usually divided into logically separated entities - components - which can be nested within other components and thus creating larger structures. A component also may or may not manage its internal state - an object for storing data. State usually stores data that are likely to change, because due to virtual DOM (an in-memory representation of an actual DOM), React is able to detect changes in state, and

swiftly update DOM. This ensures instant change of content of the application without page refresh.

React was created by software engineer Jordan Walke at Facebook in 2011 and was initially meant for internal use in the company. It has proven to be very fast and efficient language for building UIs, and found to provide great user experience, so Facebook has decided to make it open-source. Since then, major companies including Instagram, PayPal and Netflix have been using ReactJS for their front-end development.

### **3.2.1 Advantages of React**

React has seen an immense growth in popularity since it was made available to the whole world by Facebook. It has a fair number of competitors, for instance Angular.js, Vue.js or Ember.js, but due to its easy learning curve and great user experience, its popularity is not fading away. New React applications are being developed as we speak, and there are multiple reasons for that.

React is not a complex full-blown framework, but merely a library, which is consistently used with other JavaScript libraries. This allows the learning curve for developing to be short and easier when compared to other, more complex libraries. The official documentation is very clear and thorough, and web is in no short supply of React tutorials and articles.

React application is built by using components, which allows the separation of logic and rendering of each of the parts of the application, and then reuse the code wherever needed.

To ensure lightning-fast content update, React introduced Virtual DOM. This in-memory representation of DOM is where all the modifications at first take place. Virtual DOM is then compared to current DOM, and the difference (and only the difference) is re-rendered.

Data in React is being passed using unidirectional data flow between the states and layers. All data in the application flows in the same direction - from parent components to children components. This allows better control over the data, making the application more predictable and easier to debug, as certain errors such as infinite loops are becoming easier to notice and avoid.

React allows usage of JSX syntax in the code, which is similar to combining HTML and JavaScript. It is certainly not compulsory to use JSX, but it makes the code substantially

shorter and easier to both code and read. Components can be called in the same manner as XML tags, with attributes being passed as properties in XML would.

Overview of React Advantages (according to an online survey) can be found in a form of a graph in Figure 2.

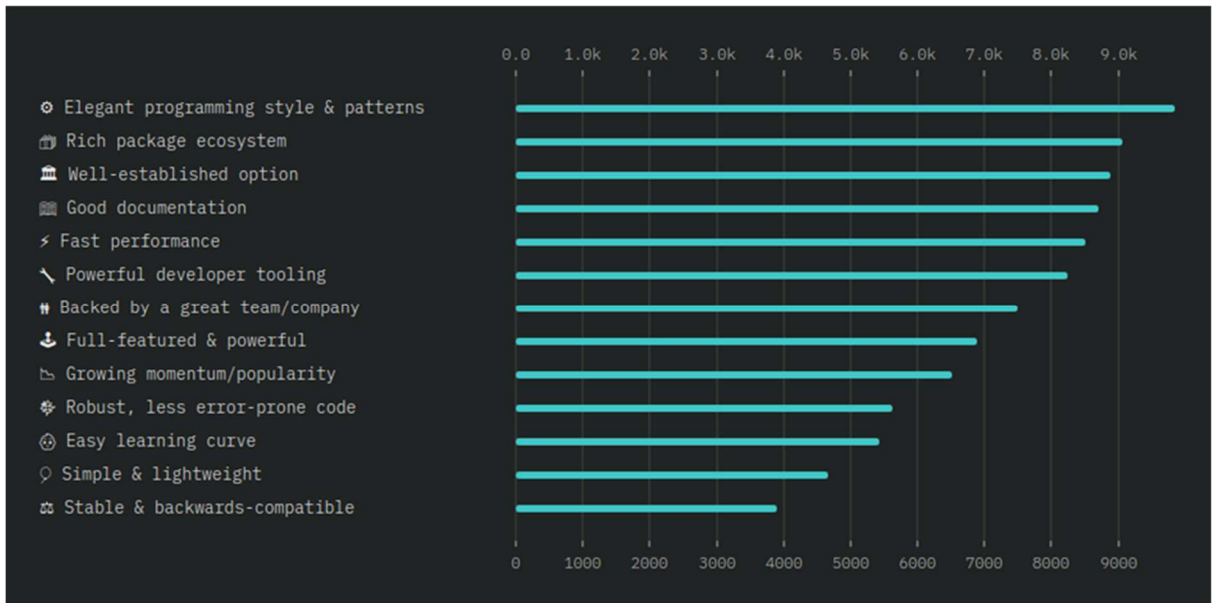


Figure 2: React Advantages

### 3.2.2 Disadvantages of React

ReactJS is a library with the sole purpose of managing UI, which makes it necessary to include additional libraries to handle other parts of the application. React does not force developers to use any specific way to structure the application, which suits the needs of experienced developers, because it allows them to build the structure precisely as they see fit. On the other hand, inexperienced coders may choose inappropriately, which will be costly once the application starts to scale. React also uses lots of technologies in the background, which may confuse beginners and make the learning curve steeper. It is written mostly in JSX and ES6, transpiled using Babel and build & packaged using Webpack & NPM.

Overview of React disadvantages (according to an online survey) can be found in a form of a graph in Figure 3.



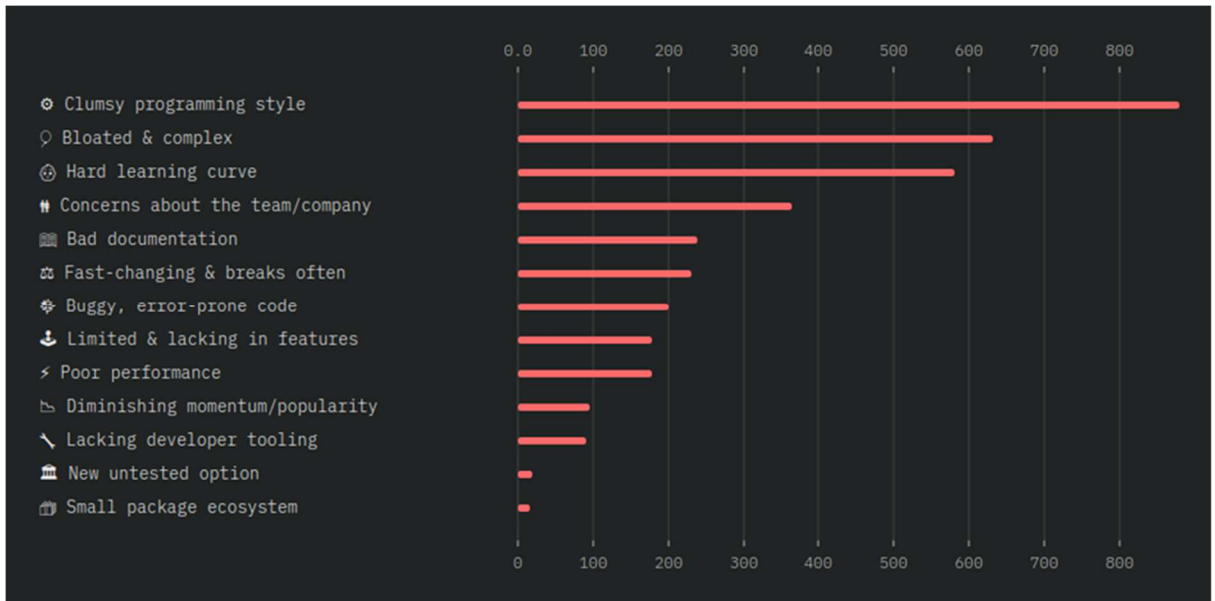


Figure 3: React Disadvantages

### 3.3 React Environment Setup

#### 3.3.1 Node.js

Node.js is an open-source JavaScript runtime environment designed to build scalable network applications. Node is not strictly required to run react code, because if React files are loaded using CDN links, a browser can execute the React code. It is, however, vital for using NPM, which provides developers with a great range of Node packages.

#### 3.3.2 CDN Links

The most basic way to provide necessary JS files is to use CDN links in plain HTML file. Script tags, that handle loading React and ReactDOM files, are provided on official React webpages. Two files are necessary for full functionality - ReactDOM handles all DOM-related methods, while React file exposes all other methods.

To add the ability to process JSX, a transpiler transforming ECMAScript 6 code into browser-compatible ECMAScript 5 is needed.

#### 3.3.3 NPM

Whenever React project spans across multiple files (which transpires fast due to the separation of code into components), including files using CDN would be tedious and impractical. That is the reason why using a package manager in a project helps to sustain well-arranged code. NPM (Node.js Package Manager) offers thousands of easily-installable

packages, which can be utilized and customized for the needs of the project. “It's the world's largest software registry, with approximately 3 billion downloads per week. The registry contains over 600,000 packages (building blocks of code). Open-source developers from every continent use NPM to share and borrow packages, and many organizations use NPM to manage private development as well.”<sup>1</sup>

### 3.3.3.1 Create-react-app

Create-react-app is an official boilerplate from Facebook designed to help developers get started on their project fast. It generates a file structure of the project, sets up a web server, which is necessary for running the code, and much more. Generating a ready-to-develop application is remarkably easy and the process is completed within minutes. The structure generated by create-react-app package is easy to understand and ready to be expanded by adding more files. There are three folders by default - node\_modules, public and src. Node\_modules accommodates all packages defined in packages.json and their dependencies. The folder public contains all the files in a project, which can be accessed publicly by anyone including index.html, an html file containing a root div element for rendering React code. The src folder is a home to all of application's components and any additional files such as CSS or JS. Default top-most component App.js can be found here as well.

### 3.3.3.2 Package.json

Package.json file can always be found in the root of a React project and it contains information about the application as well as list of NPM packages, which are being used, and scripts, which can be executed through the NPM command. It is very useful when a project is shared using a VCS (Version Control System) such as Git, because all the necessary packages and their dependencies can be installed using a single command.

---

<sup>1</sup> NPM documentation. 2018. Online Source. Retrieved from <https://docs.npmjs.com/all>

## 3.4 Architecture

### 3.4.1 Components

The concept of components is one of the most important principles in react. Using components is a convenient way to separate application code into logical parts, which can be reused in different parts of the application. There are two types of a component - stateless and stateful, sometimes also referred to as function component and class component. Both of these components can receive props, but only the latter one has the ability to manage its own state. Props stands for properties, a set of attributes which can be passed along when calling a component. They are accessible anywhere inside of the component and their value is immutable.

All components must return no more than one JSX element, with arbitrary number of sub-elements. In case the component needs to return more elements, they must be enclosed inside a parent element.

Stateful components have its own private state object. When the state changes, React re-renders changes in DOM in that component in the browser. This is the biggest difference between state and props - the state object can be modified, while the props cannot. State is utilized for saving values which are prone to changing in the runtime and whose modification will in any way affect the UI.

### 3.4.2 JSX

“JSX is an XML/HTML-like syntax used by React that extends ECMAScript so that XML/HTML-like text can co-exist with JavaScript/React code. The syntax is intended to be used by pre-processors (i.e., transpilers like Babel) to transform HTML-like text found in JavaScript files into standard JavaScript objects that a JavaScript engine will parse.”<sup>2</sup>

Facebook officially recommends using JSX with React, as it combines the simplicity of HTML and the power of JavaScript. What looks like HTML are in fact functions, which generate optimized JavaScript code for creating elements. JSX allows rendering logic and UI logic to blend together, because of the possibility to use the logic inside of views before rendering. This behaviour makes it easier to handle events, to change state according to user interactions with the UI, and to render content conditionally.

---

<sup>2</sup> React Enlightenment. 2018. Online Source. Retrieved from <https://www.reactenlightenment.com/react-jsx/5.1.html>

### 3.4.3 Virtual DOM

Usually, whenever an app, which requires a lot of data updates or feedback from the user, is being developed, DOM manipulations must be done carefully and infrequently due to negative effect on performance. React faces this problem by introducing virtual DOM.

The virtual DOM is a programming concept where a representation of an UI is stored in memory and synchronized with the real DOM using a library ReactDOM. Any new changes are first performed on the virtual in-memory DOM, and an efficient algorithm then determines the modifications that must be done to the real DOM. This provides better performance, and therefore minimum update time.

### 3.4.4 Flux

Flux is an architecture for building client-side web applications used by Facebook together with React. It is responsible for creating data layers in JavaScript applications. Flux complements React's Composable view components through its unidirectional data flow. The concept of unidirectional data flow makes it easier to debug the application, as the data go through strict pipeline. Flux presumes, that data is stored in a central place, and that it only flows through components in the direction from parent element to its children.

The Flux pattern consists of four main components: dispatcher, stores, views and actions. Actions are methods that facilitate passing data to dispatcher. Dispatcher receives these actions and triggers callbacks connected with them. Stores are simply containers for application's state and for logic of said callbacks. Views are components, which upon receiving the state forward it to child components via props. "When a user interacts with a React view, the view propagates an action through a central dispatcher, to the various stores that hold the application's data and business logic, which updates all of the views that are affected. This works especially well with React's declarative programming style, which allows the store to send updates without specifying how to transition views between states."<sup>3</sup>

#### 3.4.4.1 Flux vs MVC

The Model-View-Controller is considered the most spread application design pattern for web application development. Model serves the purpose of handling data independently

---

<sup>3</sup> Flux Github. 2018. Online Source. Retrieved from <https://facebook.github.io/flux/docs/in-depth-overview.html>

of the controller or view. View visually represents the data, and controller connects model and view and handles external inputs from a user. Following this pattern ensures separating the presentation from the model (as shown in diagram in Figure 4), which allows more flexibility and implementation of tests. Separating the controller from the view is most useful for web UI, where extensive routing, event handling and templates are common.

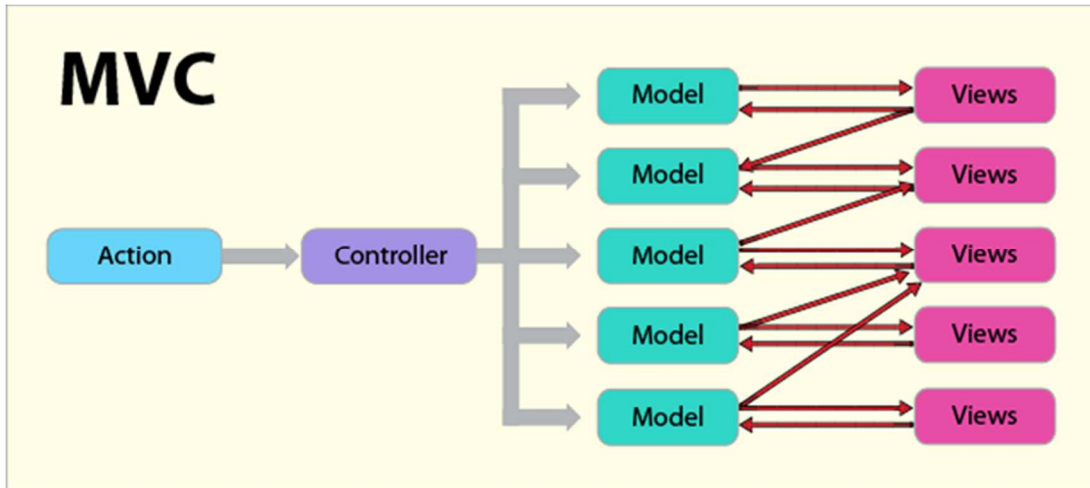


Figure 4: MVC Architecture

Facebook started using Flux over MVC due to its unidirectional data flow and better scalability for their huge codebase. Flux enforces strict rules about data flow, whereas MVC makes no assumptions about whether the data flow should be unidirectional or bidirectional. Flux design pattern can be implemented to be completely asynchronous, which is harder to achieve using MVC. Asynchronous UI provides synchronized data and fast response time to user actions, which contributes positively to great user experience.

In Flux, every state change must be part of an action, which gets executed through dispatcher (as shown in Figure 5). Reducing communication between elements of the application prevents unwanted behaviour, where one change can loop back and have cascading effect on data, which should have been left untouched.

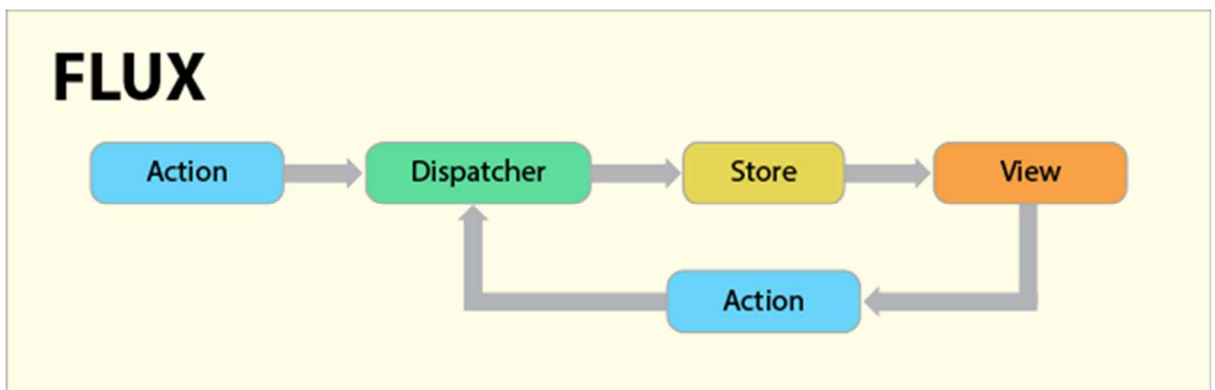


Figure 5: FLUX Architecture

### 3.4.5 Lifecycle Methods

In every ReactJS application, components are rendered into virtual DOM. Before and after rendering the virtual DOM, a component may contain methods to execute at a precise moment throughout component's life. These methods are called Component Lifecycle Methods and can be categorized into three groups, based on whether their execution takes place on creation, update or destruction of a component. Utilization for such behaviour can be found when a need occurs to trigger a method or a function automatically, without a direct impulse from a user. An example can be componentDidMount method, which will be called once a component have been mounted. Few of the many purposes, which this lifecycle method could serve, is fetching data through AJAX call or to add event listeners to desired objects. In Figure 6 are shown existing lifecycle methods to date.

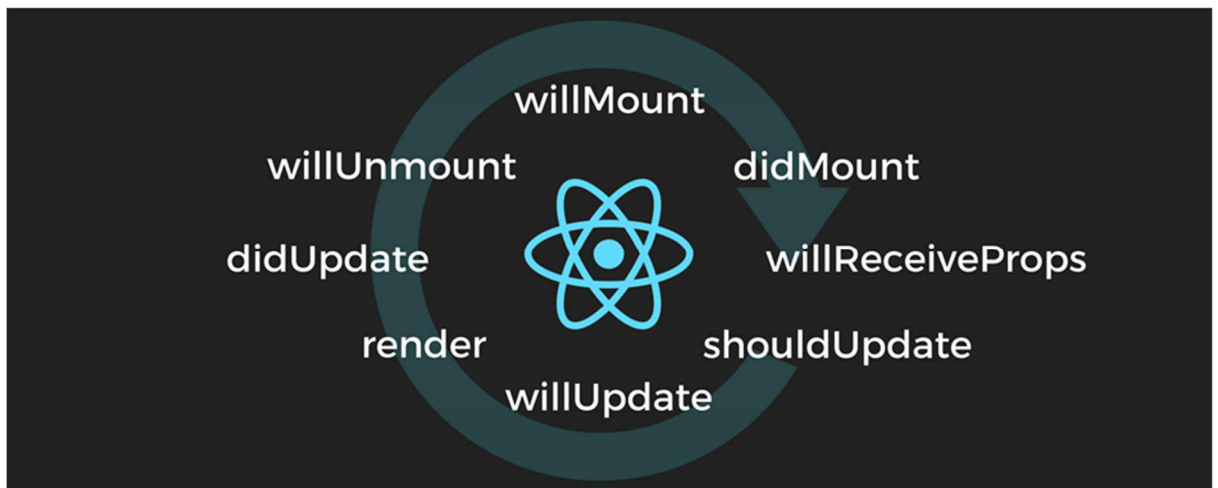


Figure 6: Lifecycle Methods

### 3.5 State & Props

“State is referred to the local state of the component which cannot be accessed and modified outside of the component and only can be used & modified inside the component. Props, on the other hand, make components reusable by giving components the ability to receive data from the parent component in the form of props.”<sup>4</sup> Props can be assigned a value when they are sent into a component but become immutable thereafter. On the other hand, local state is initialized in a stateful component and can only be manipulated with inside said

---

<sup>4</sup> Code Burst. 2018. Online Source. Retrieved from <https://codeburst.io/react-state-vs-props-explained-51beebd73b21>

component. Values of any property of the state object can be modified, and the changes will be quickly propagated in DOM.

Both Redux and MobX libraries bring concept of a global state separated from components. Such state can be connected to “Container Components”, and accessed directly from them, rather than sending necessary state values in props from component to component. This is most convenient when working with medium to large project, where passing values down through all components in the tree structure becomes tedious.

Redux and MobX create state in dedicated place, a store, where all state data is stored. These libraries also manage all methods, which in any way modify data in state, which makes it easier to keep the application well-organized. Such methods are then made available by importing them into components, where they are needed.

### **3.6 Redux**

“Redux is a predictable state container for JavaScript apps. It helps you write applications that behave consistently, run in different environments (client, server, and native), and are easy to test. On top of that, it provides a great developer experience, such as live code editing combined with a time traveling debugger.”<sup>5</sup> It is rather to challenging to keep state properly managed once the application grows into multiple components and subcomponents, since React alone does not strictly specify, how to deal with application’s state. Redux provides a global state, which is stored in a single store and can be connected to any component, no matter how deeply nested within other components (Figure 7)

---

<sup>5</sup> Medium. 2018. Online Source. Retrieved from <https://medium.com/@tkssharma/understanding-redux-react-in-easiest-way-part-1-81f3209fc0e5>

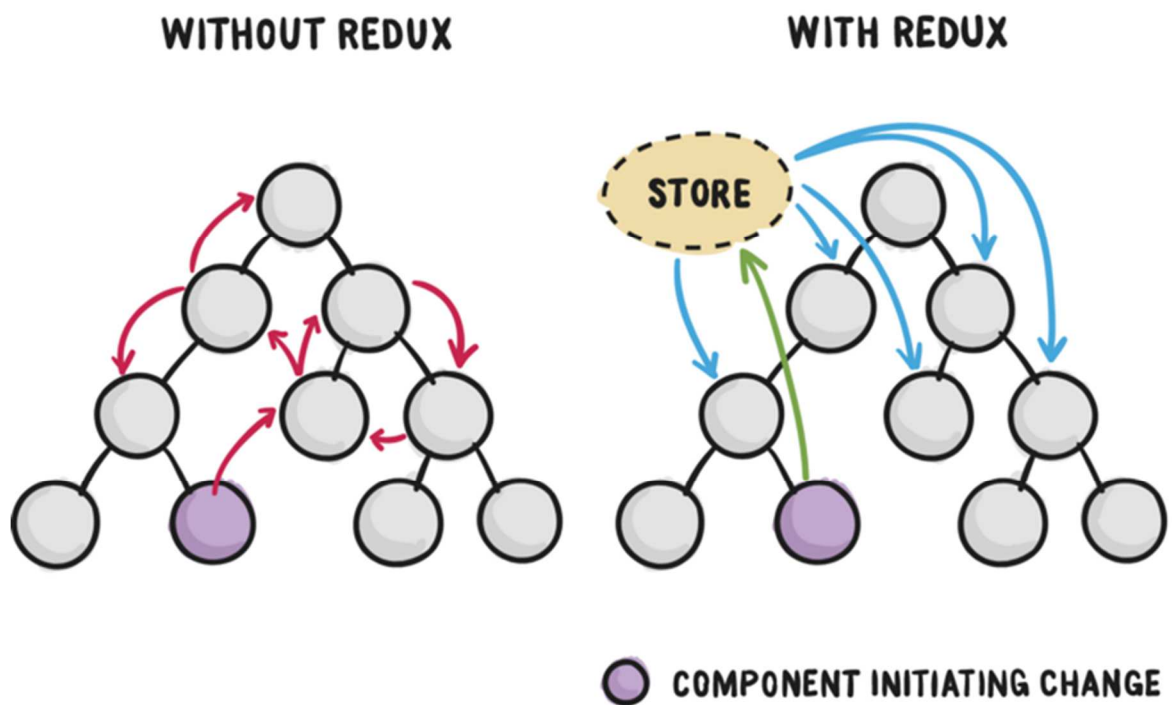


Figure 7: Redux Store

**There are three fundamental principles on which Redux is founded.**

- Application state is stored in a single object. Redux stores state in a single JavaScript object to make it easier to map out and pass data throughout the entire application. Centralizing state in a single object also makes the process of testing and debugging faster
- Application state is immutable, i.e. it cannot be directly modified. The only way to change the state is through an action, and JavaScript object, which describes changes to the state. Because all actions are centralized and happen one by one in a strict order, the danger of race conditions is eliminated.
- Reducers specify how the action transform the state. Reducers are JavaScript functions that create a new state with the given current state and action. They centralize data mutations and can act on all or part of the state. Reducers can also be combined and reused.

This architecture significantly increases scalability for large and complex applications, because it is based on Flux architecture, which demands unidirectional data flow. It also enables the usage of powerful tools for debugging such as Redux DevTools, which track state object, as it was in the past and present, and its actions.



Redux is influenced by functional programming principles, and as a result, it uses pure functions. A pure function always outputs the same result with the same input and does not have any side-effects.

Redux state is always immutable, which means that instead of changing values in the state, a new state with appropriately modified data is necessary. The state object is saved as a normalized structure and the entities reference each other using IDs.

### 3.7 MobX

“MobX is a battle tested library that makes state management simple and scalable by transparently applying functional reactive programming (TFRP). The philosophy behind MobX is very simple: Anything that can be derived from the application state, should be derived. Automatically. That includes the UI, data serialization, server communication, etc. React and MobX together are a powerful combination. React renders the application state by providing mechanisms to translate it into a tree of renderable components. MobX provides the mechanism to store and update the application state that React then uses.”<sup>6</sup>

One of the core concepts of MobX are observables, which observe existing data structures and synchronize them with the state object. Using observable is similar to transforming a property of an object into a spreadsheet cell. But values in the spreadsheets are not just simple values, but references, objects or arrays.

Any modification to state object is induced by actions. Actions are a part of store and can modify state directly, since the state is mutable. Once state changes, view updates accordingly (Figure 8).

MobX also supports, even though not necessarily require, unidirectional data flow, where the state object (or objects - MobX typically contains more logically separated stores) is being changed using actions.



Figure 8: MobX Action State Views

---

<sup>6</sup> MobX Documentation. 2018. Online Source. Retrieved from <https://mobx.js.org/index.html>

MobX strives to derive as much from state data as possible, so that no redundant data, which could be deduced from other data, is saved in state. It distinguishes two kinds of derivations:

- Computed values

These are values that can always be derived from the current observable state using a pure function.

- Reactions

Reactions happen automatically if the state changes. Their purpose is to trigger a side effect, which can be for example displaying re-calculated value on screen.

In MobX, the state is mutable, and more than one state object may exist. The State itself stays denormalized, and the data are saved as a nested structure in relation to each other.

## **4 Practical Part**

### **4.1 React Application**

#### **4.1.1 Introduction**

Two ReactJS applications have been built for analysing development with external state management libraries. These applications have the same architecture and work in the same way, but one of them is using MobX library, and the other one relies on Redux. This approach allows for a comparison of advantages and disadvantages of both regarding complexness, performance, learning curve and code length. This approach serves well the purpose of highlighting key characteristics of the libraries. It also proves, that Redux, while being by far the most popular state management library used with React, is not the best option for all purposes and that further consideration is important when deciding, what state management library to choose for a project according to its specifications.

#### **4.1.2 Application Description**

The purpose of this application is short-term management of user's tasks. If a person has an overwhelming number of tasks to do, it is useful to prioritize and to divide them into categories, so that it is better arranged and filtered. The application has been designed according to the principle of Eisenhower's Matrix (also known as the Urgent-Important Matrix), which can help prioritize tasks by sorting them into four categories according to their urgency and importance. The top-left section of this grid is for tasks that are both urgent and important, while the bottom-right section symbolizes little urgency and importance. The matrix has been invented by US president Dwight D. Eisenhower in 1950s as a tool to decide, on which tasks he should focus each day.

Task are automatically retrieved from user's Google Calendar through an API using constant in-code credentials. Fetching method is asynchronous, which ensures, that the data are loaded and displayed without the need of page refresh. Tasks are then displayed in a left panel in a list, from where they can be sorted into the four sectors as shown in Figure 9. After sorting has been finished, user has the opportunity to download a jpg image displaying the matrix and all tasks inside.

The application is responsive and the number of tasks per sector is not limited. The front-end is written in ReactJS and HTML and styled with SCSS and CSS3 feature such as grid and transitions. It is fully supported by all major modern web browsers.

## Eisenhower Matrix Task Organizer

This app can help you organize your tasks into 4 categories depending on their urgency & importance.

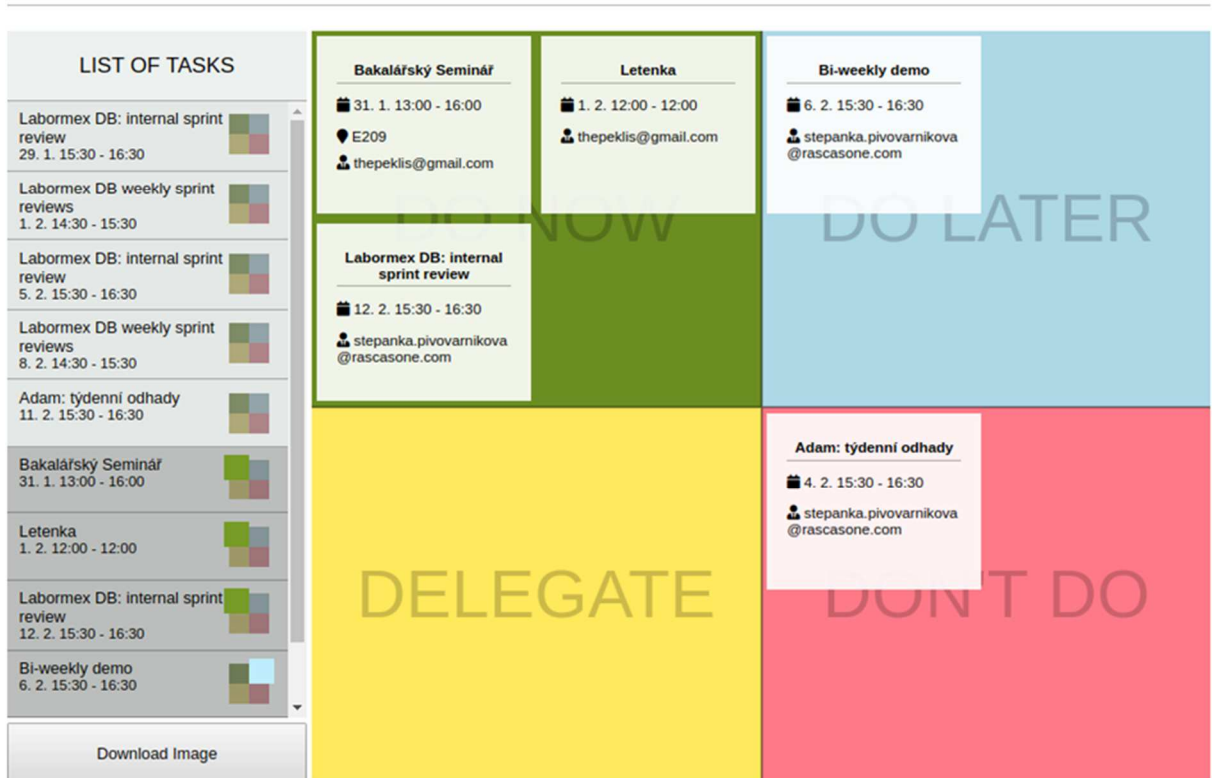


Figure 9: Application User Interface

### 4.1.3 Project Structure

Source code of the application is located in src folder, where following structure has been created to accommodate needs of the project and store files in a logical manner. Contents of the folder are displayed in a tree structure in Figure 10.

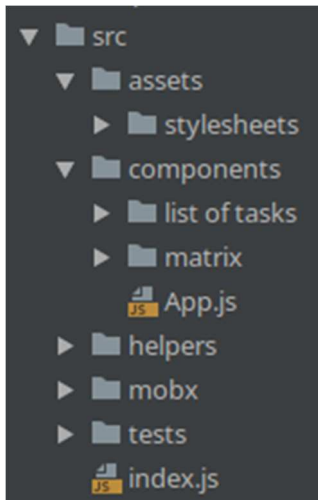


Figure 10: File structure of src/ folder

Assets contains all SCSS files, which are necessary for styling the UI. Helpers hold general reusable functions, which are exported, so that multiple components can make use of them without code duplicity. In case of MobX application, MobX folder is where the store holding application's state is located. Redux application has Redux folder, which holds the state and all Redux-related files. And components folder contains all of React logically structured components, with App.js being the root of the structure tree. Components can be called with JSX within render method and pass data as props from parent to child. Example of a render method from App.js is shown in Figure 11.

```

render() {
  return (
    <main id="grid">
      <header>
        <h1>Eisenhower Matrix Task Organizer</h1>
        <p>This app can help you organize your tasks into 4 categories depending on their urgency & importance.</p>
      </header>
      <TaskList />
      <button id="download" onClick={this.downloadImage}>Download Image</button>
      <Matrix />
      <footer>
        <span>Adam Peklák - Practical part of Bachelor Thesis</span>
      </footer>
    </main>
  )
}

```

Figure 11: App.js Render method

#### 4.1.4 NPM Packages

The project is based on NPM package create-react-app, which offers the advantage of having tools such as Babel or Webpack already installed and configured, and therefore saves developer's time. Create-react-app is widely used for starting React projects, because

it is highly customisable and serves the needs of majority of typical projects, while saving time that would otherwise have to be spent of boilerplate.

The project utilizes Google Calendar API and package react-google-calendar-api, through which the task data is fetched. In order to save & download final image of the matrix, dom-to-image package captures specified div element determined by its ID and file-saver manages download of a snapshot of the div container.

## 4.2 State management

### 4.2.1 Complexity

#### 4.2.1.1 Redux

Redux is a complex library, which is very convenient over large codebase, but it can be too bulky for small to medium sized projects, because it requires relatively lot of boilerplate to set up. After installing necessary NPM packages, a store must be created with at least one reducer imported. It is necessary to insert a specific clause, so that the DevTools extension, which is used for debugging, would be working with the application.

It has three important separate parts: Actions, Reducers and Store. Actions are payloads of information sending data to the store. They are the only way a store can use to obtain information due to React's strict unidirectional data flow rule. This makes it more tedious to create methods, that are modifying state date, but it allows for features such as time travel, which is a debugging feature, which makes it possible to view state data history from the initial load. Sending data to the store can be invoked using store.dispatch(). Tasks are placed in separate dedicated file and are only imported into files where they are necessary. Redux action to assign task to a sector can be seen in Figure 12.

```
export const putTaskInSector = (task_id, sector) => dispatch => {
  const state = store.getState()
  let tasks_copy = Object.assign( target: [], state.tasks.tasks)

  if (task_id !== undefined && tasks_copy.length > 0) {
    tasks_copy.find(t => t.id === task_id).sector = sector
    dispatch({
      type: 'UPDATE_TASKS',
      payload: tasks_copy
    })
  }
}
```

Figure 12: Single Redux action from taskActions.js

“Reducers specify how the application's state changes in response to actions sent to the store.”<sup>7</sup> It must define the type, which an action has dispatched, and determine, how the state will be affected.

Store connects both Actions and Reducers and actually holds the state object of the application. It can be created with Redux method `createStore` from one or many Reducers.

The state object and actions can only be used in a view after it is wrapped in a `connect` method, as shown in Figure 13 below.

```
const mapStateToProps = state => ({
  tasks: state.tasks.tasks
})

export default connect(mapStateToProps, { fetchTasks, putTaskInSector })(TaskList)
```

Figure 13: Redux's `connect` method and mapping State to Props

Because state object must remain immutable, Redux maps the state object to props, which can be accessed within a view like any ordinary props propagated from a parent component. Actions are mapped to props in the same way state is.

#### 4.2.1.2 MobX

MobX is a relatively simple, lightweight state management library, which requires very little boilerplate. It is typically used with small to medium sized projects, because it offers lesser scalability over huge codebase compared to Redux. MobX offers simpler and more straight-forward approach to state management than Redux without compromising functionality. Its incomplexity allows the code to remain brief and improve its readability.

```
@action putTaskInSector = (task_id, sector = 0) => {
  if (task_id !== undefined && this.tasks.length > 0) {
    this.tasks.find(t => t.id === task_id).sector = sector
  }
}
```

Figure 14: Single MobX action from `taskStore.js`

Action from Figure 14 has the exact same purpose as the action written in Redux in Figure 12, namely, to assign task to one of the four sectors, but it only takes up 5 lines instead of 11, which is the case for Redux. It is the result of mutable state, because new state object does not have to be created due to every state modification.

---

<sup>7</sup> Redux Documentation. 2018. Retrieved from <https://redux.js.org/basics/reducers>

Redux's actions and reducers are also not applicable in case of MobX, because state can be accessed and mutated directly and actions are defined in store class, together with state object. This means, that in order to set a store for this project, only one comprehensible file had to be added to the application. MobX also automatically maps the whole store including data and methods into props, so there is no need for manual `mapStateToProps` method of Redux.

```
const Root = (  
  <Provider TaskStore={TaskStore}>  
    <App />  
  </Provider>  
)
```

Figure 15: MobX Provider wrapped around App

The state is provided using a `Provider` element, which wraps the root element of the application and accepts store object as an attribute (Figure 15). This wrapper is used by both MobX and Redux, but out of these two libraries, only MobX uses decorators. `Provider`, together with `inject` decorator, propagates state object and all actions from the store to child components.

```
@inject("TaskStore")  
@observer  
class TaskList extends Component {
```

Figure 16: MobX decorators

Decorators are an important part MobX. `Inject` decorator serves the purpose of directly providing state to a deep-nested component without the need of relying on passing it through all middle components, as demonstrated in Figure 16. It is similar to Redux's `connect` method, but MobX offers simpler approach with minimum necessary code.

`Observer` is what makes react component reactive. If the component contains any observable value, and that value was to change, observer will call the `render` method of the component, and thus project the changes in the UI. Redux does not have these decorators, because any modification to state must be invoked through an action.

#### 4.2.2 Performance

An identical test for Redux and MobX application has been performed, in order to determine, which application loads faster, and which modifies data in the state object faster. All measurements have been performed 3 times, and an average result has been used as the final result.



#### 4.2.2.1 Load Time

This performance test is measuring the time it takes the two applications to become available for the user from the time it is accessed in browser.

Window.performance JavaScript function has been used in the constructor and lifecycle method of the root component, in order to start and stop the timer. It has been started in constructor of the component and stopped in ComponentDidMount lifecycle method, which gets executed right after component is done loading.

It has been discovered that it takes 462.99ms for Redux application to load, whereas MobX application performs the same in 430.72ms.

#### 4.2.2.2 Assigning Task to Sector

A simple benchmark function has been created for testing, how much time is necessary in order to move a task from once sector of the Eisenhower Matrix to another. This test has been conducted with three different number of state modification - 1000, 10000 and 100000. In every single one, MobX was significantly faster than Redux, and the more data was modified, the more significant the difference became. With thousand changes, MobX was faster by 28 milliseconds, but with hundred thousand changes, the difference was almost 10 seconds. Complete results are listed in Table 1 below.

| Number of changes | MobX (ms) | Redux (ms) | Difference (ms) | Percentage difference |
|-------------------|-----------|------------|-----------------|-----------------------|
| 1 000             | 12,03     | 40,12      | 28,09           | 233,49                |
| 10 000            | 75,19     | 254,74     | 179,55          | 238,79                |
| 100 000           | 187       | 1154       | 967             | 517,11                |

Table 1: Speed of modifying data from state

These results were measured using Google Chrome performance developer tool. The modifications to state object were induced using benchmark function shown in Figure 17, which keeps continuously assigning a task to sectors until given number of times.

```
benchmark(id) {  
  for (let i = 0; i < 10000; i++) {  
    this.props.TaskStore.putTaskInSector(id, i%5)  
  }  
}
```

Figure 17: Benchmark function

### 4.2.3 Learning Curve

MobX is much easier to learn and has a steady learning curve compared to Redux. It is based on Object-oriented programming and uses a lot of abstraction, which leads to shorter code. Direct access to store data increases simplicity and makes the framework easier to understand. Setup of MobX is fast and well documented, and therefore will not discourage people from the very beginning.

Redux has steeper learning curve, because it is made to be scalable and accommodate huge codebase. Lengthy setup is necessary when adding Redux into a project, and Redux library itself is not enough. Additional middleware such as Redux Thunk is necessary for asynchronous data fetching and it makes the learning curve even steeper and can be a breaking point for many inexperienced developers.

Both libraries are well-documented using helpful examples and descriptions. Redux, however, is the most popular state management library used with Redux to date, with over 4 million weekly downloads from NPM. MobX only has 257 thousand weekly downloads from NPM, which is a huge difference, not to mention a significant one. More people using a library means more content and solutions on the internet. Therefore, using a popular library is advantageous, because there is a high probability, that if there is a bug or a problem during development, someone on the internet has already encountered it before and posted a solution.

### 4.2.4 Code length

Code length is an important aspect for developers as it is related to the speed of development and, although not directly, simplicity. All files regarding state management have been taken in account, and the summary of the number of lines of code calculated. For Redux, it means taking store itself, reducer and actions, whereas for MobX, single store file contains everything necessary.

MobX takes up 30 lines of code, whereas number of lines of Redux code reaches 64, which is more than twice the amount it took MobX.

## **5 Results and Discussion**

### **5.1 Results**

State management libraries Redux and MobX have been compared in an identical ReactJS project for the purpose of demonstrating the benefits and shortcomings of both. Comparisons were made of important aspects of a data management library including code length, performance, complexity and learning curve. The results are significant, as they provide specific and definite comparison of the two libraries in the same scenario, and can help with decision, on which library should a developer rely when starting a project. The results also demonstrate the possible advantages of migrating from one library to another on an existing project.

#### **5.1.1 Complexity**

When comparing source code from both Redux and MobX application, it became apparent, that the libraries were designed for different use-cases.

Redux required deeper understanding of its inner workings, and its architecture will become advantageous once the project reaches certain complexness and size. It has stricter rules concerning how data are being manipulated, so it takes away some of developer's freedom to manage data in the best way, that is suitable for the project. On the other hand, enforcing guidelines on unidirectional data flow and on state immutability makes the code clearer, if the project scales into a large application.

MobX allows for more freedom when handling data in state, from which small to medium sized applications benefit, because it allows for the code to be shorter and development faster. The possibility to mutate state makes it simpler to implement methods, that are meant to modify state, but it makes it harder to debug, as the debug tools are not as extensive, and feature like time travel is not possible due to state's mutability. MobX saved a lot of time in the setup stage, where very little is necessary in order to start developing, and the code is clear and concise. A single file with store was created, where methods are defined using convenient decorators.

When using state data or methods in React components with Redux, it is necessary import them and then to map them manually into props. MobX's inject decorator makes both

state data and methods available through props automatically, so that developer can use it right away.

### 5.1.2 Performance

An important aspect of any state management library is, how fast can it load and modify data. Two different tests have been conducted to discover, which one has a faster load time, and which one modifies state data faster.

#### 5.1.2.1 Load Time

After putting JavaScript performance methods inside of constructor and a lifecycle method of a root component, it has been discovered, that Redux application loaded on average in 462.99 ms and MobX application in 430.72 ms, which makes MobX slightly faster, as demonstrated in the graph in Figure 18. Both measurements have been taken 3 times, and an average result has been used.

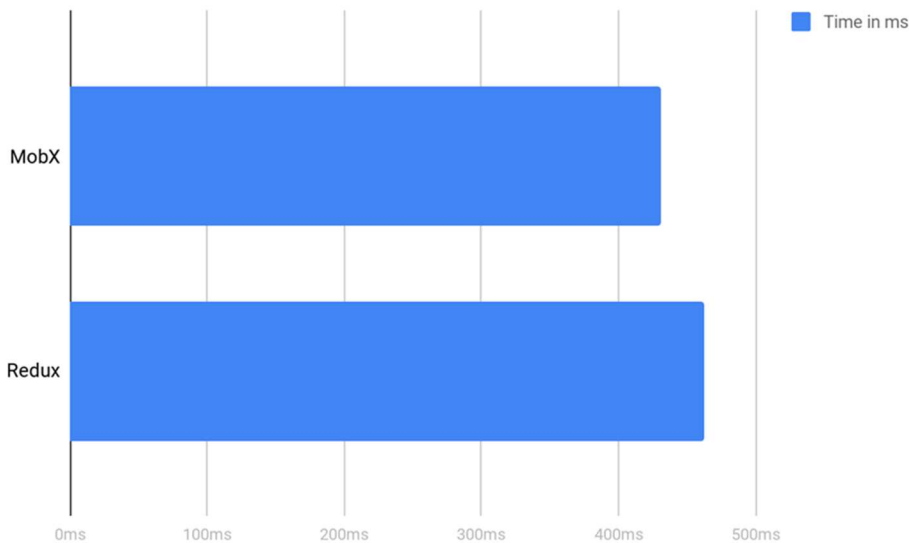


Figure 18: Load Time graph

#### 5.1.2.2 Modifying state data

Using a benchmark function, a test has been conducted to measure, how much time it will take to assign task to different sectors of the Eisenhower Matrix. It has been found, that MobX outperforms Redux in any number of data changes, and that the difference gradually gets larger as the amount of changes increases. The test has been conducted with thousand, ten thousand and hundred thousand modifications to the state object, and the graph

comparison of the results is displayed in Figure 19. It proves, that for simple state object manipulation, MobX is out of the two libraries faster.

### Modifying state data

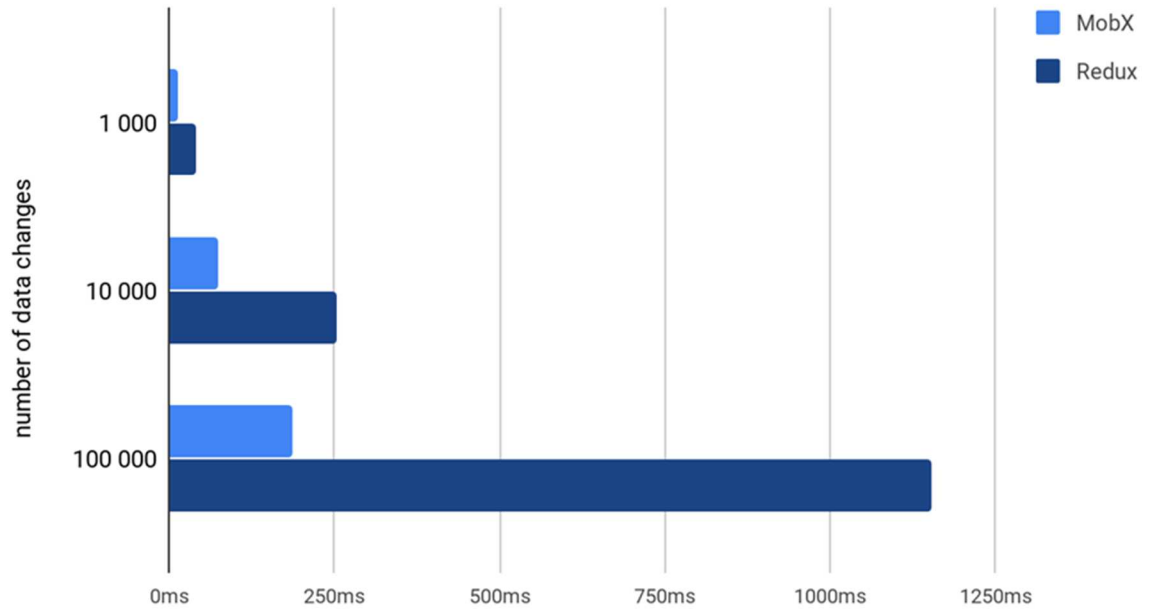


Figure 19: Modifying state data graph

### 5.1.3 Code length

A comparison of code length has been conducted between the Redux and MobX applications. All files from folders MobX and Redux, which contain all state data and methods, have been considered and their lines of code counted. The result is, as visualized in Figure 20, that the number of lines of MobX code was 30, while number of lines of Redux code reached 64.

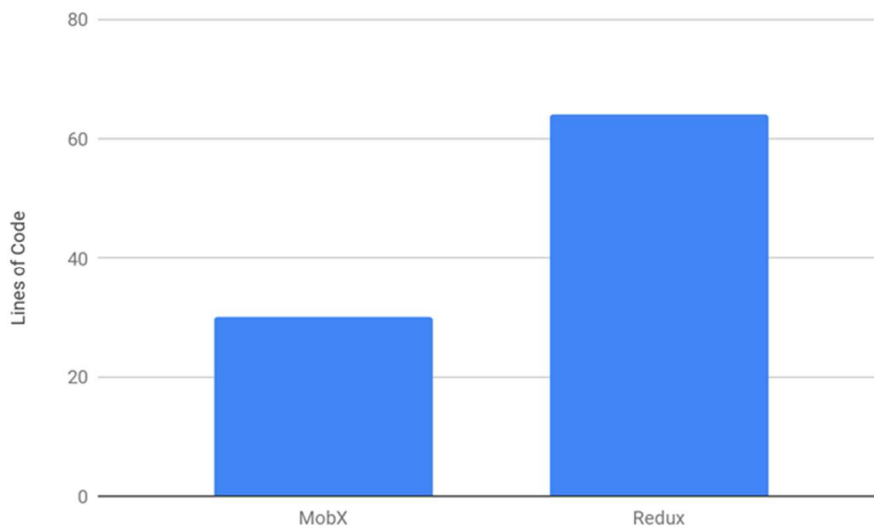


Figure 20: Code length comparison

#### 5.1.4 Learning Curve

Due to Redux's complexity and lengthy setup, its learning curve is steep and unfriendly for beginners. It has been found that additional middleware such as Redux Thunk is necessary for asynchronous calls. It has been proven on this application, that using Redux on a small project is disadvantageous, because it will cause the code to be lengthier and it will take time to set up, without providing any significant benefit in return.

MobX is intended for small to medium sized projects, which makes this application suitable use-case, and it makes it easy for beginners to try it on a simple application. The learning curve of MobX is significantly mellower and steady, because the setup is fast and well documented and because the code is shorter and clearer. Store can be accessed and modified directly, and it is available with all data and actions in injected components automatically through props.

#### 5.1.5 Summary

After reviewing the results, it became clear, that MobX is better suited for this particular use-case than Redux. It is less complex, which ensures fast development and simple implementation. Because this application is meant to be very small, there is no danger of the code becoming messy and hard to maintain, and extensive debugging options are not important. MobX has been proven to be faster in both loading and modifying state data, which is an undeniable benefit. It also has easier learning curve and its code is shorter, which makes it a great state management library for inexperienced programmers.

Redux has been proven to be better fitted for bigger and more complicated projects. Its complexity and longer code become beneficial, once project scales into huge application. If used correctly, it will help keep the project from becoming chaotic due to the vast number of files in a project. Its learning curve is steep, but thanks to extensive documentation, a skilled developer should not have any problems setting up a ReactJS project with Redux state management.

## **5.2 Discussion**

The results obtained in this thesis have been measured precisely and multiple times, but the values, that are being compared, are always coming from the same simple task management application. That is both positive and negative. The advantage of this approach is, that the two applications are identical in every aspect apart from the state management library, and it is therefore possible to compare implementations of the same methods in both. That makes it convenient to compare specific aspects such as code length or performance. It highlights the differences in libraries, because source codes of both have the same purpose, but the implementation of the solution is different. The disadvantage of demonstrating the difference on similar applications is, that only this specific use-case can be tested. This project did not use full capabilities of either library, so the limitations of Redux and MobX have not been tested.

The libraries have been compared in the learning curve aspect, where it was determined, that the learning curve of Redux is steeper than that of MobX. This has been based on a personal experience from writing the application, which makes it a subjective result. Some properties, such as how easy something is to learn, are difficult to measure objectively, because it depends on many factors, for instance beforehand knowledge or simply personal preference. My subjective result is backed up by literature about MobX and Redux and is therefore not misleading.

## 6 Conclusion

The main objective of this thesis was to compare Redux and MobX - two of the most popular state management libraries with ReactJS. The objective was fulfilled in the practical part by creating two ReactJS applications for task management, which are similar in every aspect apart from the state management library, and then comparing them.

The first partial objective was to describe characteristics of JavaScript, and more generally, of functional programming. The first chapter of the literature review has been dedicated to this topic, and it provided a fundamental understanding of the problematics, which all following chapters were expanding.

The second partial objective was to analyse the architecture used by ReactJS, which is a part of literature review, as well as the practical part. Important concepts of React have been defined and specific principles explained. The topic of React architecture has been analysed extensively to provide broad understanding of the inner workings of the language.

The third partial objective was to analyse data flow and state management, which is closely connected with the primary objective, and was also a part of both literature review and practical part. Data flow of native React, as well as state management libraries, has been analysed and compared. State management, being the single most important principle for the practical part, has been elucidated comprehensively and in depth.

It has been proven, that using MobX on a React project of limited dimensions is beneficial in multiple aspects. It offers significantly better performance than Redux, achieves the same results with less code, and takes less time to learn and setup in a project, which contributes to greater speed of development

This result can be used by ReactJS developers to help them with the decision, which state management library to use with their projects, or to learn more about React and state management in general. It also proves, that MobX, while being significantly less popular, can in specific use-cases outperform Redux.



## 7 References

### 7.1 Bibliography

Wieruch, R. (2018). *The Road to learn React: Your journey to master plain yet pragmatic React.js*.

Lopez, L. (2017). *React: Quickstart Step-by-step Guide to Learning React Javascript*.

Gelman, I., & Dinkevich, B. (2017). *The Complete Redux Book: Everything you need to build real projects with Redux*.

Sidelnikov, G. (2017). *Learning React JavaScript Library from Scratch*.

International, E. (2009). ECMA-262 ECMAScript Language Specification. *JavaScript Specification*.

Academind. (2016). *ReactJS Basics - #1 What is React?*

O'Shaughnessy, K. (2016). *Choosing a JavaScript Framework in 2017 – Medium*.

Aggarwal, S. (2018). Modern Web-Development using ReactJS. *International Journal of Recent Research Aspects*.

MacCaw, A. (2011). *JavaScript Web Applications. Presentation*.

<http://doi.org/10.1017/CBO9781107415324.004>

Gagliardi, V. (2018). *React Redux Tutorial for Beginners: The Definitive Guide (2018)*. Retrieved August 20, 2018, from <https://www.valentinog.com/blog/react-redux-tutorial-beginners/>

## **8 Attachments**

File *ReactJS\_applications.zip* has been attached to this bachelor thesis, which contains both Redux and MobX version of the React application source code.