Mark Piispanen

Modern architecture for large web applications

Bachelor’s Thesis in Information Technology

May 1, 2017

University of Jyväskylä

Department of Mathematical Information Technology

Author: Mark Piispanen

Contact information: mark.piispanen@hotmail.fi

Supervisor: Marjaana Nokka

Title: Modern architecture for large web applications

Työn nimi: Nykyaikainen arkkitehtuuri suurille web sovelluksille

Project: Bachelor's Thesis

Page count: 21+0

Abstract: Nowadays web applications are growing fast. It is important to choose a solid architecture for a large web application to be able to maintain, grow and scale it. In recent years, applications made with architecture using component based approach for user interface and uni-directional data flow became very popular. The aim of this thesis is to research this particular architecture and its parts. It will be shown what problems this architecture solves and what kind of applications will benefit the most from using it.

Keywords: Javascript, Web application, React, Components, Flux, Redux, Unidirectional data flow

Suomenkielinen tiivistelmä: Web sovellukset kasvavat nykyään nopeasti. On tärkeää valita vakaa arkkitehtuuri isolle sovellukselle, jotta sitä voi ylläpitää, suurentaa ja skaalata. Viime vuosina, on tullut suositummaksi sovellukset jotka käyttävät komponentti kohtaisen lähestymisen ja yhdensuuntaisen data kulkun. Tämän tutkielman tarkoituksena on tutkia kyseisen arkkitehtuurin ja sen osia. Näytetään vielä mitkä ongelmat tutkittava arkkitehtuuri ratkaisee ja minkälaiset sovellukset saavat eniten hyötyä sen käyttämisestä.

Avainsanat: Javascript, Web sovellus, React, Komponentit, Flux, Redux, Yhdensuuntainen datan kulku

List of Figures

Contents

1 Introduction

Any web developer will say that making applications for web is a tough task. Developing the front-end of a web application can easily get out of hands. Developer teams must think about application's architecture very carefully. There must be an architecture to structure front-end applications, so that it is easy to add new features, debug and to scale if necessary. In this research a particular architecture which became popular in recent years is examined.

There are many challenges in modern web development. The first challenge is that nowadays, applications have complex user interface and a lot of data must be moved around and changed over time(Facebook a. 2016). The other challenge is that users don't want to wait long for applications to load an because of that applications must be fast. Adding new features should be simple and should not break other parts of an application. Finding and fixing bugs must be straightforward. Applications need to scale if the number of users is growing.

Using the right architecture for a web applications should improve several goals. The first goal is to increase maintainability, so that it is easy to add new features and find bugs. The second goal is to reduce complexity, so that it is easy to reason about different parts of an application and to understand how these parts communicate with each other.

In this study React and Redux were taken as example libraries for this architecture. These particular libraries were chosen for this study because of their popularity and good documentation. There are many other libraries which may be used instead.

There are two parts in this architecture. The first one is a view, which renders data to user and re-renders when data changes. The second part is a state container, which handles application state and data flow.

The architecture examined in this study doesn't have a particular name. This architecture is a few years old, but it started to gain popularity very quickly in recent years. The purpose of this study is to overview this architecture, show its strengths and disadvantages, what problems it solves and who should use it. This material could be used by people, companies or teams who are deciding, which architecture to base their project on, or just need it for educational purposes. It is extremely important to choose the right structure for a project in the beginning.

In chapter 2, there will be told about research background, mentioned libraries and frameworks which work for this architecture. Also, there will be told about history and mentioned about other architectures for web applications.

In chapter 3, the architecture mentioned earlier will be examined. First, React, its principles and basics and after that Redux. Also, it will be shown how React and Redux work together.

In chapter 4, there will be examined pros and cons of this architecture. What problems it can solve, for what applications it is suitable, who should use it and who shouldn't. Also, examples of big web applications will be mentioned.

2 Modern Web development

In this chapter, history of web applications and background research will be reviewed. Also, different libraries which can be used instead of React and Redux will be shown.

2.1 History

First web applications were simple static HTML pages with only CSS. CSS is used for styling HTML into more appealing form. These static pages were slow, because every page, thus all the HTML and CSS, had to be fetched from the server every time user clicked a link.

Everything changed with the arrival of Ajax, which stands for Asynchronous JavaScript and XML. AJAX is a set of techniques which enabled web applications to make quick incremental updates to the user interface without reloading the entire browser page.(Ajax 2017).

Compared to the websites with only static pages which user had to download every time a link was clicked, websites using Ajax were many times faster. In the beginning only some parts of a website used Ajax to get data without refreshing a web browser page. After some time single page applications began their ascend. These applications used asynchronous Javascript to download all the required data so that for the time being on the website user didn't have to refresh a browser page at all. User experience was greatly improved by reducing the time of moving between different sections of application.

Single page applications continued to grow and then need for a better structure of these applications emerged. Developers began to create components to better organize applications. One of the first component-based libraries was React, which was created by Facebook.

Because React was just a library for rendering data, there was a need to manage data in large applications. For this data management problem, an architectural pattern called Flux was created and introduced by Facebook. After the release of Flux, many developer teams started to make their own implementations of Flux. Some of these implementations were resembling Flux, some took a different approach, but the ideas were the same.

2.2 Research background

There aren't many researches on this architecture, because it is relatively new. Books about it are being published right now. Nevertheless, there are a lot of new tutorials, blog posts, videos from Javascript and Web conferences appearing about this architecture on the internet.

There is a bachelor's thesis(Data flow patterns 2016) named 'An Investigation of Data Flow Patterns Impact on Maintainability When Implementing Additional Functionality' which was conducted to test the data flow patterns impact on maintainability. A conclusion could not be made to prove that the data flow patterns does affect maintainability.

2.3 Different frameworks/libraries

Vuejs, Riotjs, Infernojs and React are some of the front-end libraries where the library will work as a view. All of the libraries mentioned are component-based. Some of these libraries are more mature than others. They all vary in library size and syntax. Some of the libraries like Reactjs, Infernojs, Vuejs use Virtual DOM. More information about Virtual DOM will be presented in the chapter about React. Each library has its own advantages and disadvantages, so a library must be chosen based on particular needs. Because React is one of the most popular, matured and developed libraries, it was chosen as an example in this study. React's popularity can be seen from github statistics for having the most stars among other libraries on github(Github 2017).

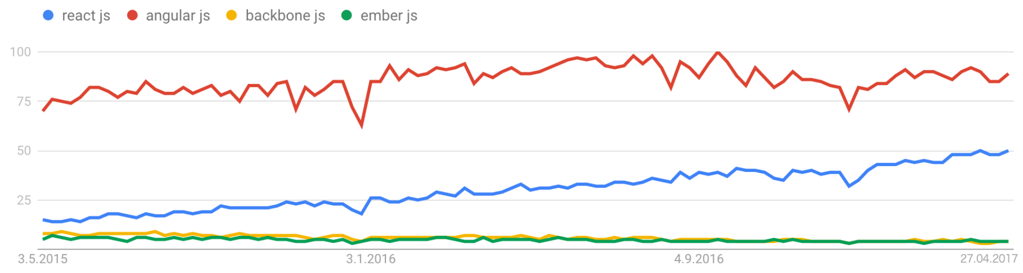

Figure 1. Comparison of React and MVC libraries(Javascript libraries 2017)

For data flow and state management, there are other libraries like Redux. For example Mobx, Reflux, Alt. These libraries vary in implementation in one way or another. Going through all the differences and comparing these libraries would go beyond the scope of this study, so a few differences will be pointed out. For example Redux uses functional programming paradigms and immutable data management(Redux c. 2016). Application state in Redux is immutable. MobX on the other hand does mutate the state of an application. The other thing is that MobX does many things automatically(Mobx 2017), there is a lot happening in the background what developer cannot directly see. Redux on the other hand doesn't hide much, but it tends to be more difficult to learn. Also, some libraries use only one object for application's state while others may use multiple objects.

It can be seen from Figure 1, React is gaining popularity compared to MVC like libraries(Angularjs, Backbonejs, Emberjs) which were released a few years before React. It is hard to exactly define if a particular library is a MVC library or some kind of derivation or variation of MVC.

3 Architecture overview

3.1 React

React is a JavaScript library, developed by Facebook. Initially it was an internal project which was open sourced later in 2013. React's main purpose is the ability to build large-scale user interfaces with data that changes over time(Gackenheimer 2015).

In a React application, everything is made of components, which are self-contained, concern-specific building blocks. These components are reusable and extendable. This reduces complexity and increases maintainability of the user interface. Components are kept small and because they can be combined, it is easy to create complex and more feature-rich components made of smaller components.(Cassio 2015).

Every React component can have an internal state. It is an object which contains properties that belong only to this component. When you update the state of a component, React runs an algorithm which examines, should the component re-render and how to re-render it efficiently.

As Stefanov (2010) states, DOM access is expensive; it's the most common bottleneck when it comes to JavaScript performance. React uses an in-memory, lightweight representation of the DOM called Virtual DOM. Changing and checking Virtual DOM is faster and more efficient than doing the same operations on the real DOM(Cassio 2015). Using Virtual DOM makes React very fast and efficient. React apps can easily run at 60fps, even on mobile devices.(Cassio 2015).

When the state in React component changes, it follows an algorithm presented below and also shown in Figure 2:

1. Re-render everything to a Virtual DOM representation.
2. Compare current Virtual DOM representation computed after the data changed with the previoues one, computed before the data changed.

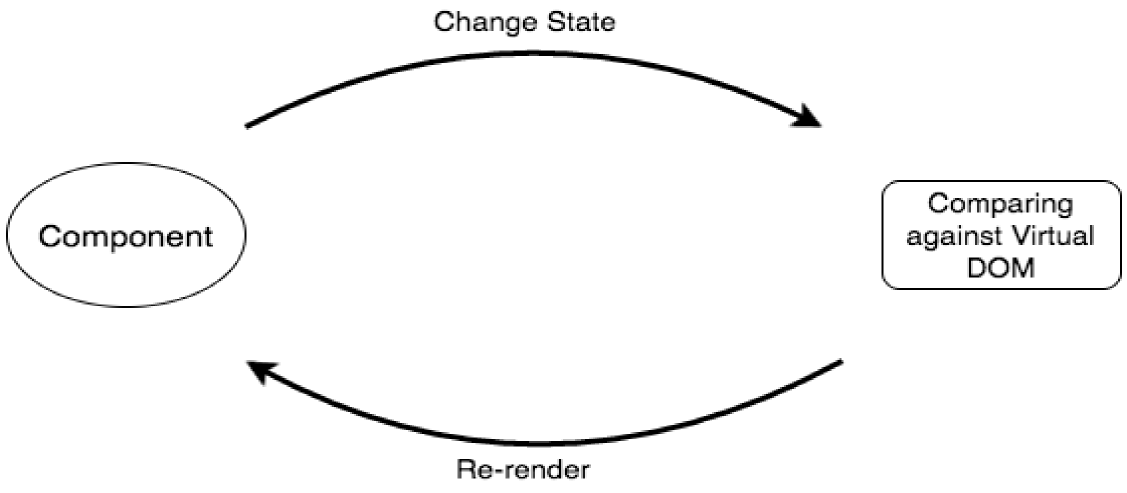

Figure 2. component's state change cicle

3. Compare those two representations, isolate what actually changed in the Virtual DOM and update the real DOM with only what needs to be changed.

React has ability to pass data between components, so data can only flow from parent to child component in one direction. Because React was designed to only render data, it may be a problem if the same data has to be accessed by multiple unrelated components hierarchy in hierarchy. In large application it becomes messy and complex very quickly. In this architecture Redux helps with data flow and management, which will be discussed in the next chapter.

3.2 Redux

As it was pointed out in the previous section, Redux comes from a need to share data between multiple components which reside in different component hierarchies. Before Redux is reviewed, Flux must be mentioned. Flux is an architectural guideline created by Facebook(Cassio 2015). It aims to solve problems related to application state management and data flow in client-side web applications.

Redux is one of the many implementations of Flux. It has been chosen for its popularity and good documentation. At the moment(30.04.17) Redux(Redux github

2017) has almost three times more stars on github than Facebook's implementation of Flux(Flux github 2017).

Main motivations behind Redux were to make state mutations predictable(Redux d. 2016) and reduce the complexity of data management in applications by using unidirectional data flow. Because in Redux data flows only in one direction and has state in one place, it makes application more predictable(Gackenheimer 2015). If it is known how the data flows in an application, its state and behaviour can be predicted. Redux also uses immutable data management which causes less side effects in the data flow, thus leading to simpler programming and debugging(Redux c. 2016).

The Store in Redux is immutable. It means that when the Store must be modified, new Store is created, instead of mutating the old one. It must be noted that immutability isn't used in every Flux implementation library. Other libraries which don't use immutable data management mutate the Store object directly.

Redux can be described in three fundamental principles(Redux b. 2016):

1. The state of the whole application is stored in a Javascript object tree, a single store.
2. The state is read-only. The only way to change to state is to emit and Action.
3. Changes to the store are made with pure functions. It means that they don't mutate the state, but return new state objects.

Redux mainly consists of three parts. The first one is called the Store. It stores the state of an application. Every single piece of data of the application is stored there. It is a plain Javascript object which may be created in several ways, here is an example state of application which manages Todos:

```
{
  todos: [
    { text: 'Buy groceries', complete: false },
    { ... }
  ]
```

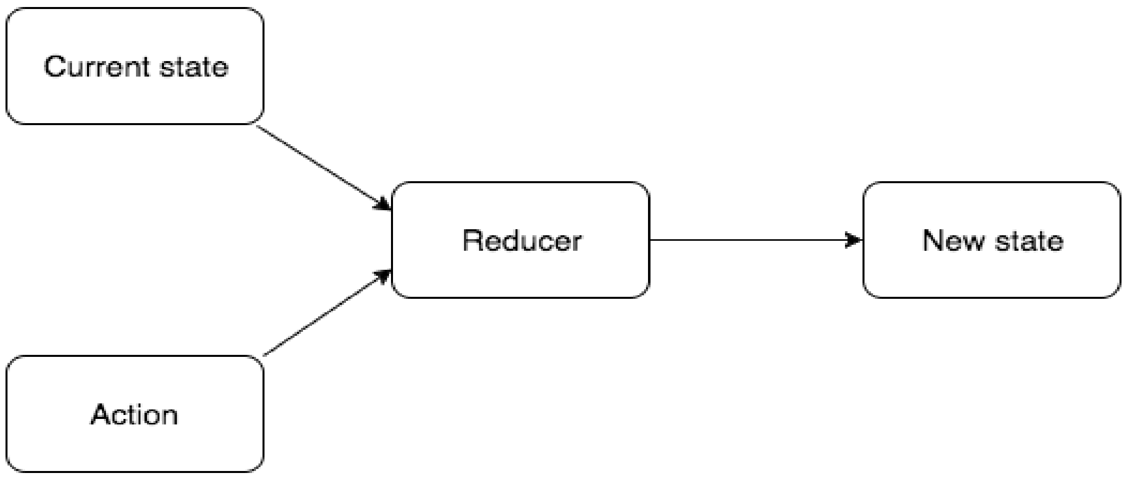

Figure 3. Reducer performing a change to the state

```
        }
```

The next part is Actions. Actions are plain Javascript objects which describe what should happen, but they don't specify how it should be done. Actions contain data that is sent or dispatched to the Store.

For example an Action might be like the following object:

```
        {
         type: 'ADD_TODO',
         text: 'Walk a dog'
        }
```

The last part are Reducers. They tie the Store and Actions together and contain the logic of an application. Reducers are simply functions that make changes to the Store. Reducers are pure functions. It means that they don't mutate data but return a new data every time.

In Figure 3 below, function takes the state of an application and an Action as parameters. Then a Reducer identifies Action's type, makes changes to state and returns new state. If it cannot identify action's type, it returns unmodified state. In the image below it can be seen how a Reducer works.

Redux helps keep the code easier to read, maintain and grow(Gackenheimer 2015). Redux is also scalable(Gackenheimer 2015). There is an easy and clear way on how to expand an application which uses Redux. For example, Actions might be stored in a separate file, so they aren't spread around an application. Also Reducers can be stored in one place for clear structure. When a new feature must be added in Redux, an Action must be created. After that, a function must be added to Reducer, which does the actual change to the Store.

Redux has its own downsides. It involves writing more boilerplate code than with standard applications. Also it is relatively hard to migrate existing project to using Redux because it uses radically different approach for managing application state and data flow.

3.3 React-Redux

When a component is created, it can subscribe to the Redux store, so that this component will know when the data in the Store changes. Actions can be dispatched from React components when something happens and the Store needs to change. It can be done by calling a certain function from React component which dispatches and Action. After calling, it goes to the right reducer and the Store changes.

Figure 4 shows, how data flows in React-Redux application. First, the View(in this scenario, React component), subscribes to the Store, and if data in the Store changes, React automatically re-renders data if necessary. Events like submitting a form, on the other hand will cause React component to emit an Action, which through Reducer will change the Store.

One of the downsides of this architecture is that it has a steep learning curve. It is hard to quickly begin developing applications with this architecture. It takes some time to get used to for a person who never programmed like this before.

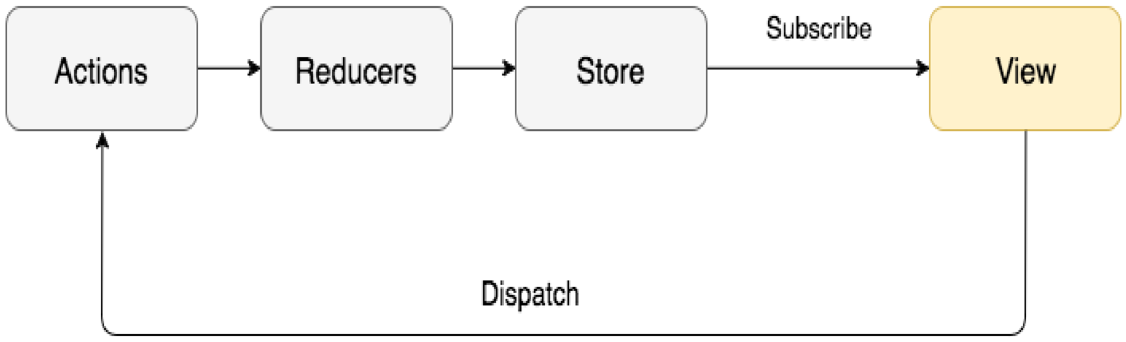

Figure 4. React and Redux data flow

3.4 Comparison with other architecture

There are many numbers of different architectural patterns which can be used in developing Web applications. MVC is a very popular design pattern, which is widely used nowadays and has been used for a few dozens of years. MVC consists of three parts: The Model, The View and The Controller.

The Model stores the data needed for an application. The View is displaying necessary data to the user. The Controller acts as a glue between models and views. It requests data from the Model and listens to the event in the View.(MacCaw 2011). None of the three parts are concerned about how others do their job.

It must be noted that there are many MVC-like architectures, MVP(Model-View-Presenter), MVVM(Model-View-ViewModel) or HMVC(Hierarchical Model–View–Controller), which are all either a variation or a derivation of MVC pattern. For that reason the simplest version of MVC is shown in Figure 5.

If React-Redux compared to MVC(Model-View-Controller) architecture, React can be thought of as the View and Redux as the Model and the Controller. But one of the main differences between Redux and MVC is the data flow. MVC allows bidirectional data flow, while Redux is strictly using unidirectional data flow. For example, in MVC, data may flow from the View to the Controller and back the same way, while in Redux, change must be made through dispatching an Action.

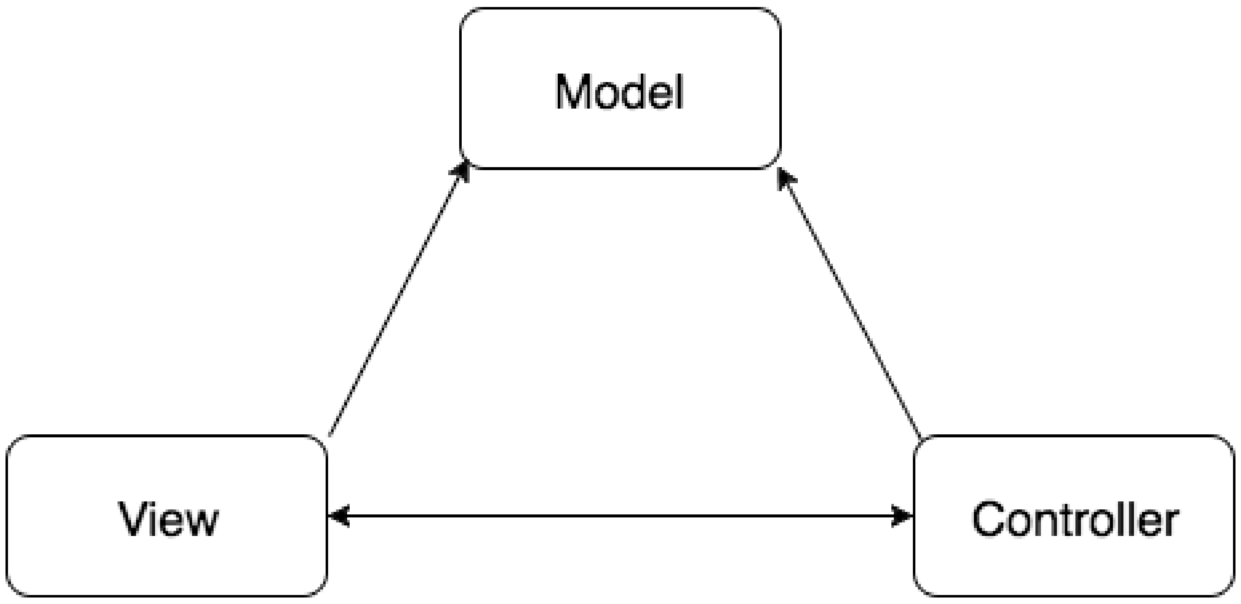

Figure 5. MVC architecture data flow(Chadwick et al. 2012)

The architecture presented in this study is not as mature as MVC architecture. Only after a few years it will be seen how these two really compare to each other.

4 Applications

It will be shown in this chapter what types of web applications will benefit the most from this particular architecture. Also, there will be demonstrated what problems this architecture solves and why it is a good choice for particular web applications. It can be noted that this architecture isn't dependent on any particular backend server. So any server can be used to host or to communicate with web applications based on this architecture.

4.1 Suitable web applications

Because Redux requires a lot of boilerplate code and initially was needed when React application was becoming complex and had a lot of components, data management with Redux is not necessary in small or static applications. It will just add an extra layer of complexity to a small web application. From previous chapter it is known that adding functionality to application via Redux, requires adding data to Actions, Reducers and possibly the Store, so it will require unnecessary extra code and effort if an application just needs to download a piece of data from the server and show it to the user.

On the other hand, web application might have a complex UI with large component hierarchy and a lot of nested components which have to pass data via props to each other. In this situation Redux will help to reduce complexity by providing one place for common data and functions which have to be shared between components. Also, if the project is maintained by a big team of developers, it is easier for everyone to have a one place to add new features.

Some huge companies such as Twitter, Instagram, Netflix and Airbnb are using this architecture in their web applications at the moment(27.04.2017). For example, Instagram web application is a big React component with many components inside of it.

4.2 Web applications related problems

On of the problems is state handling in web application. How to store, mutate, and share the state between components which need it. Redux aims to solve these problems as described in the previous chapter.

The other problem is application's scaling and maintaining. If the application isn't structured properly, later when more and more features are added, codebase expanded, it can become hard to add new features, refactor or understand how the application works. Redux gives a way to structure a web application, so that it is easily scalable and maintainable, as for example adding new features or finding bugs. If new functionality is required, it can be created easily with Redux by writing necessary Actions, Reducers and possibly adding new data to the Store. Maintainability also is achieved through React modular design. If a new user interface feature is needed, new React component can easily be written and plugged in where necessary.

One problem can be application's unpredictability. This architecture solves it by introducing a one way data flow, as described in the previous chapter. It makes the application more predictable and easier to debug.

5 Conclusion

The right architecture for a large Web application is essential. It will affect how the application will scale and grow, will it be maintainable. A lot of money can be saved on maintaining, refactoring.

The architecture presented in this study is a good choice for a large application, which promises easy scaling and maintaining. It has proved its efficiency and reliability in huge projects such as Netflix, Instagram, Twitter, Airbnb.

Only the architecture which meets one's needs must be used. React-Redux isn't suitable for any project. In fact, if the application is too small, it will add too much complexity and there will be more obstacles than benefits. In some cases it maybe more appropriate and benefitable to just use React.

Architecture and ideas presented in this research aren't revolutionary. Almost all the ideas behind React and Flux aren't new, but presented and implemented in a new way. These architectures and patterns are evolving the whole time. Because this architecture isn't as old as MVC, some time must pass before it can be said with certainty how much benefits this it really has.

Bibliography

Javascript libraries. 2017. *Javascript libraries comparison*. Saatavilla WWW-muodossa<URL: `https://trends.google.fi/trends/explore?date=2015-04-30%202017-04-27&q=react%20js,angular%20js,backbone%20js,ember%20js`>. Cited 30.04.2017.

Redux github. 2017. *Redux github*. Saatavilla WWW-muodossa<URL: `https://github.com/reactjs/redux`>. Cited 30.04.2017.

Flux github. 2017. *Flux github*. Saatavilla WWW-muodossa<URL: `https://github.com/facebook/flux`>. Cited 30.04.2017.

React a. 2016. *Components*. Saatavilla WWW-muodossa<URL: `https://facebook.github.io/react/docs/components-and-props.html`>. Cited 12.02.2017.

React b. 2016. *State and Lifecycle*. Saatavilla WWW-muodossa<URL: `https://facebook.github.io/react/docs/state-and-lifecycle.html`>. Cited 12.02.2017.

React c. 2016. *Performance*. Saatavilla WWW-muodossa<URL: `https://facebook.github.io/react/docs/optimizing-performance.html`>. Cited 12.02.2017.

Redux a. 2016. *Redux and React*. Saatavilla WWW-muodossa<URL: `http://redux.js.org/docs/basics/UsageWithReact.html`>. Cited 12.02.2017.

Redux b. 2016. *Redux principles*. Saatavilla WWW-muodossa<URL: `http://redux.js.org/docs/introduction/ThreePrinciples.html`>. Cited 12.02.2017.

Redux c. 2016. *Redux immutability*. Saatavilla WWW-muodossa<URL: `http://redux.js.org/docs/faq/ImmutableData.html`>. Cited 27.04.2017.

Redux d. 2016. *Redux motivations*. Saatavilla WWW-muodossa<URL: `http://redux.js.org/docs/introduction/Motivation.html`>. Cited 30.04.2017.

Mobx 2016. *Mobx*. Saatavilla WWW-muodossa<URL: `https://mobx.js.org/index.html`>. Cited 30.04.2017.

Gackenheimer, C. 2015. *Introduction to React*. New York: APRESS.

Front-end Frameworks. 2017. *Top front-end frameworks*. Saatavilla WWW-muodossa<URL: `https://github.com/showcases/front-end-javascript-frameworks`>. Cited 12.02.2017.

Ajax 2017. *Ajax*. Saatavilla WWW-muodossa<URL: `https://developer.mozilla.org/en-US/docs/AJAX`>. Cited 26.04.2017.

Cassio, A. 2015. *Pro React*. New York: APRESS.

Chadwick, J., Snyder, T. Panda, H. 2012. *Programming ASP.NET MVC 4*. Sebastopol: O'REILLY.

MacCaw, A. 2011. *JavaScript Web Applications*. Sebastopol: O'REILLY.

Flux and React *Facebook Flux React presentation*. Saatavilla WWW-muodossa<URL: `https://www.youtube.com/watch?v=nYkdrAPrdcw`>. Cited 27.02.2017.

Stefanov, S. 2010. *JavaScript Patterns*. Sebastopol: O'REILLY.

Zakas, N. 2012. *Maintainable JavaScript: Writing Readable Code*. Sebastopol: O'REILLY.

Mikowski, M. Powell, J. 2014. *Single Page Web Applications*. Shelter Island: MANNING.

Data flow patterns 2016. *Ajax*. Saatavilla WWW-muodossa<URL: `http://lnu.diva-portal.org/smash/record.jsf?pid=diva2%3A1015014&dswid=8532`>. Cited 30.04.2017.