Bachelor’s thesis

Information Technology

2018

Parshina Maria

JAVASCRIPT BEYOND THE BROWSER

TURKU UNIVERSITY

TURKU AMK

TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR’S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology

2018 | 23

Parshina Maria

JAVASCRIPT BEYOND THE BROWSER

Since 1995 JavaScript has been known as a scripting language. However, nowadays its application in web development is diverse. NodeJS inspired web professionals to create many JS frameworks and libraries. However, JS application is not limited by that. This thesis takes a look at JavaScript’s application in mobile and desktop development. This is made possible by such tools as React Native and ElectronJS, correspondingly. Yet, the opinions of web experts on JS’ readiness for mobile and desktop development vary from positive to rather negative. The main goal of this thesis is to explore how JS functions outside of the browser and assess its efficiency and potential. To achieve the goal, cross-platform application for mobile and desktop is build solely in JS. The development process is supported by recent publications and documentation of used tools. Among the topics discussed in the thesis are JS’ history, server-side JS development, different approaches of mobile and desktop development and modern web application structure. To summarize, the conducted research proved the that JavaScript can be used for a wide range of purposes with high efficiency. Also, the language has solid potential for the future implementation, however, it definitely will not replace all the existing languages.

KEYWORDS:

JavaScript, React Native, Electron JS, mobile development, desktop development, cross-platform, web application

1. CONTENTS

2. FIGURES

LIST OF ABBREVIATIONS (OR) SYMBOLS

CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	HyperText Markup Language
IoT	Internet of Things
JS	JavaScript
NPM	Node Package Manager
VR	Virtual Reality

1. INTRODUCTION

JavaScript (JS) was originally created as an in-browser scripting language, the purpose of which was to bring interactivity to a static web-page. However, recently it has extended beyond its original limitations. In a relatively short period of time, a number of Javascript frameworks has emerged, allowing programmers to build web applications with higher complexity and quality in less time. Furthermore, the programming language has gained popularity not only among frontend specialists. An increasing number of software developers recognize now its usefulness in desktop and mobile development. Consequently, nowadays companies in the information technology and web development business have more options to choose from when it comes to developing different kinds of software. One of these is developing applications for different platforms solely in JS (Neer and Lyle, 2013).

These new tools for software development are, however, welcomed with a certain degree of caution. For example, opinions on JS' applicability in mobile development differ significantly. Some sources affirm JS' readiness for mobile and desktop development or see its solid potential. Abeer Alkhars gives examples of successful desktop application development with JS in his thesis. He uses JS and Java (in this case JavaFX) frameworks for desktop development. In particular, he points out less memory consumption and faster execution of JS, compared to JavaFX application (Alkhars, 2017). Others tend to believe that the idea of using the language for these purposes is far-fetched and, therefore, they are hesitant about it. Matias Martinez has mentioned in his article that the life-cycle and the quality of cross-platform mobile applications built using cross-platform mobile app development frameworks have not been studied in depth yet (Martinez, 2017).

This thesis aims to introduce JS from the outside-of-the-browser perspective. The phrase "outside-of-the-browser" hereafter means "used not as a scripting language".

The main goal of the thesis is to assess JS' potential in mobile and desktop development. For this purpose two sample applications (desktop and mobile) with the same functionality have been built, using two JS frameworks - Electron for the desktop app and React Native for the mobile app. This way we can see the differences between the two frameworks and estimate the timeframe of development in each of them.

This thesis provides answers to the following questions:

- Does JS syntax differ greatly between the frameworks?
- Are documentation and supporting materials for the frameworks available and adequate?

- How much of code base can be reused between the frameworks?
- What are the advantages and disadvantages of using JS based frameworks for cross-platform application development?

In order to answer these questions and conduct the application development successfully the following methods are used. Firstly, the recent literature and publications on the topic are analysed. Secondly, two simple applications are developed to provide a deeper comprehension by learning from practice. Cross-platform mobile application for iOS and Android is built with React Native. Desktop application is built with Electron JS. Both applications utilize the same API. Since server-side development is out of the scope of this thesis, third-party API is utilized for this purpose. Lastly, the official documentation for both frameworks is used and analyzed during the development process.

This thesis consists of three chapters. A brief history of JS' evolution introduces the language in the beginning. The second chapter provides necessary information to understand the influence of JS on the modern software development and how it is used beyond the browser. It goes through the most popular recent technologies and their applications. In addition to this, some technical aspects of JS' cross-platform development are given. The last chapter contains the description of the application development process and its results.

2. SHORT HISTORY OF JAVASCRIPT

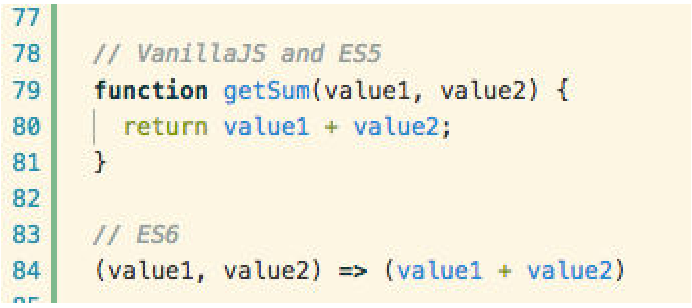
2.1. Evolution of JavaScript’s standards

The history of JS begins in 1995, when developers around the world tried to find a way to make web pages more dynamic. In other words, to create web pages capable of displaying different content each time they are accessed. JS has become an essential part of the classic trinity of web development: HTML, CSS and JS. Since then developers could create animations, handle web page events and manage document object model (DOM) elements. All these functionalities mainly achieve a better usability and accessibility of web applications (Neer and Lyle, 2013).

JavaScript follows ECMA-262 standards (ECMAScript specification), which define certain rules and guidelines that are compulsory to comply with. They comprise what is called “Vanilla JS”, in other words, simply plain traditional JS. However, during the recent years JS has seen some significant changes in newer ECMAScript standards. Such major modifications and improvements to the specification were made in 2009 and then in 2015, which are correspondingly the fifth and the sixth edition and thereafter are called ECMAScript 5, or ES5, and ECMAScript 6, or ES6 (Zakas, 2016).

Today most of the modern browsers support ES5, but not all. Support for ES6 is only starting to be added to very few browsers.

While ES5 has much in common with Vanilla JS, ES6 is inspired by functional programming. As such, it includes functions such as .map() for looping through arrays and .reduce() for retrieving a sum of all integers in an array (Elliott, 2014). Function declaration is also different in ES6. Figure 1 presents an example of function declaration in Vanilla JS / ES5 version and an arrow function: a function without declaration, introduced in ES6.



```

// VanillaJS and ES5
function getSum(value1, value2) {
  return value1 + value2;
}

// ES6
(value1, value2) => (value1 + value2)
```

Figure 1. Vanilla JS / ES5 and ES6 function structure.

Most JS frameworks that utilize JS for outside-of-the-browser development use one of these latest standards, that is ES5 or ES6 (and some even support both), with the code being compiled to Vanilla JS. However, the fact that these newer standards have significant differences and cannot be used interchangeably has caused slight confusion among developers. In cases where people have done some development using ES5 and want to switch to newer standards of ES6, they need some tools to migrate their codebase between the two standards. Therefore, ‘transpilers’ like Google Traceur exist. With their help, it is possible to translate ES5 code into ES6. Unfortunately, such translators do not always work flawlessly, leaving space for a grat amount of manual work (Zakas, 2016).

2.2 Appearance of JavaScript frameworks and NodeJS

During the development process, a web developer may face common tasks such as making web page respond to users’ interactions, animate or trigger page elements. VanillaJS does not provide ready solutions to these by default, therefore programming of such common elements becomes too much work. Also, such solutions are prone to errors and not reliable (Chaffer, 2009).

With a growing complexity of web-based applications, developers seek for ways to simplify the development process and sustain product quality at the same time. Nowadays, using pre-written JS code, called a library or a framework, is a common practice. Libraries and frameworks are usually created by professional developers and are often open-source. The main purpose of those is to provide desired functionality without writing unnecessary code (Elliott, 2014).

A wide range of frameworks and libraries were built to expand JS’ in-browser functionality. A JS library is a pre-written JS code which provides developers with a very specific functionality. Nowadays, it is a standard practice to utilize multiple libraries in a single project, as graphic 3D and user interface libraries. A framework in general is an aggregation of libraries, written following certain rules, the purpose of which is a complete application development. Being more sophisticated and intricate than a library, it requires more time for a developer to get used to. (Hartmann, 2011)

One of the most successful JS library releases was the introduction of jQuery library in 2006 (Neer and Lyle, 2013). jQuery allows a developer to create dynamic page content by selecting its elements in the DOM. Image sliders, AJAX, pagination, form validation are few examples of what can be achieved with jQuery. The library gained its popularity due to the fact that it could work the same way with all of the existing web browsers. Before jQuery, developers had to customize web application for each browser separately, which added a huge amount of unnecessary work, hence made the development process tedious. Although web browsers have improved greatly over time, the library still remains the most popular JavaScript library on the Internet (Daly, J., 2016).

Released in 2011, Bootstrap.js is another good example of a popular JS library. In combination with HTML and Bootstrap.css, the library's plugins allow developers to build a responsive and interactive website base. It works by means of providing extensive pre-built website components, which are essential for the majority of modern web applications. For instance carousel, dropdown menu, dialog box and transitions. By implementing Bootstrap in their code, developers avoid repetitions and ensure cross-browser compatibility (Cochran, 2012). Both Bootstrap and jQuery reduce development time significantly and, therefore, allow developers to concentrate on enhancing more important features.

jQuery, Bootstrap.js and other extensions helped JS to become popular and essential in the web development world. However, for a long time they served only the in-browser development purposes.

Everything changed with the introduction of NodeJS. It was released in 2009 as a JS runtime environment. In other words, an environment, where the program can be executed: it makes it possible for a developer to see errors and warnings. NodeJS has become popular in a short period of time for a number of reasons. One of them is the ability to use JavaScript as a server-side language, including the web-server and server-side scripts (Cantelon et al., 2014). JavaScript is the most popular language in the world, therefore, most of developers are at least somewhat familiar with it from the start and do not have to learn a totally new language. Another reason is the above mentioned runtime environment, which in addition to other features delivers the possibility of parallel command execution, making NodeJS fast and, therefore, exceptionally attractive for developers. Moreover, NodeJS is relatively quick and easy to start working with. Developers tend to install it and start programming with it rather quickly, avoiding long and complex procedures (Teixeira, 2012).

NodeJS is open source, consequently, a great number of developers around the world contributes to its repository or to its modules or packages. Nowadays, NodeJS community offers hundreds of Node Package Manager (NPM) packages for a great number of purposes, which has a considerable impact on the speed of web development. In addition, NodeJS has provided a solid ground for developers' inspiration to elaborate on JS development even further (Dayley, 2014). This makes NodeJS a founder of "JavaScript everywhere paradigm", the main point of which is unification of a programming language for web development (Cuomo, 2013).

Modularity brought to JS world by NodeJS led to creation of dozens of JavaScript front-end frameworks. Among them AngularJS, Vue, ReactJS, Ember and many more. Their main purpose is simplification and optimization of front-end development.

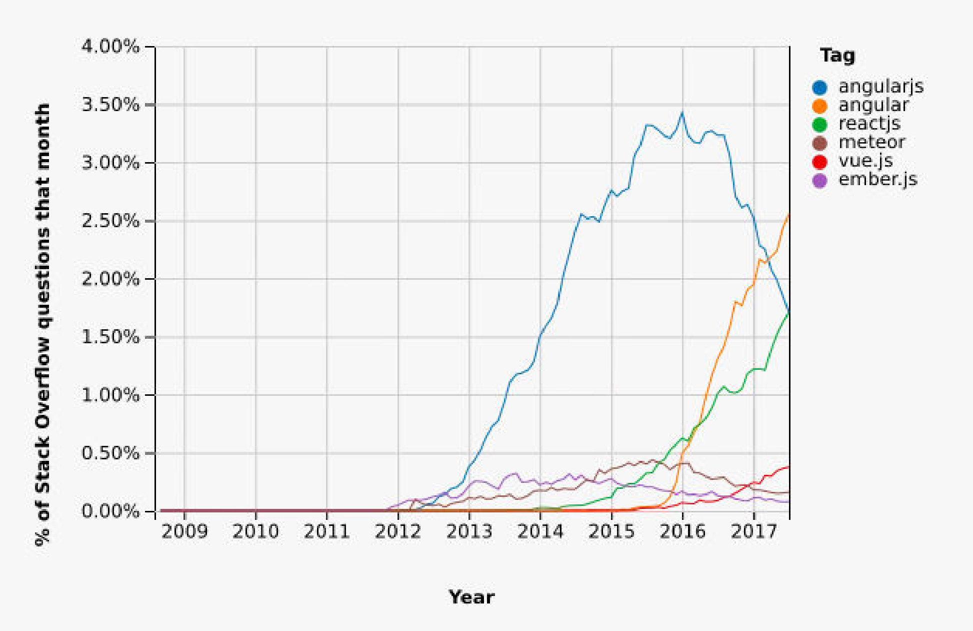


Figure 2. Popularity trends of JavaScript frameworks (source: https://trends.google.com/trends/explore?date=today%205-y&geo=US&q=%2Fm%2F012l1vxv,Vue.js,%2Fm%2F0j45p7w,%2Fg%2F121dcj3r).

Figure 2 presents popularity trends of six most popular frameworks, based on number of questions asked on StackOverflow from 2009. (insights.stackoverflow.com/trends, 07.01.2018) According to the graph, the frameworks start to gain popularity as soon as they appear. It can also be mentioned that among represented on the graph frameworks, the most popular and fast-growing are AngularJS, ReactJS and Vue.

2.3 JavaScript in modern software development

The most important consequence of NodeJS' creation is an opportunity to use JS for server-side tasks, which undoubtedly became a success in a world of web-development. World-famous companies including PayPal, Netflix, Cisco, Uber and IBM have chosen NodeJS for their backend architectures in production. Apparently, they have made this decision based on NodeJS advantages over the platforms they used before. For example, an experiment was conducted in PayPal, during which they compared Java and JavaScript development process and results. NodeJS application has shown better statistics: it took fewer human resources and less time to complete, 33% less code was written and 40% less files were created during development (Harrel, 2013).

Nevertheless, JS application is not limited to web development. Recently, JavaScript has become an instrument for building modern mobile and desktop applications. It is widely known, that there are several operational systems and mobile platforms on the market. Each of them supports particular languages and systems, which others are not compatible with. Consequently, developers undertake an obligation to build several applications with the same functionality, but which have nothing in common code-wise. This compatibility may be facilitated by building desktop and mobile applications with JavaScript. (Serrano, Hernantes, Gallardo, 2013)

One of the noteworthy examples of a recent JavaScript library is Electron. It was created in 2013 by GitHub. Web browser Chromium and NodeJS are utilized by the library in order to build cross-platform desktop applications. It works by means of wrapping the browser window into the frame of a desktop application, using advanced features of NodeJS, such as file system and OS communication to name a few (Herron, 2013). Electron has quickly become popular among affluent IT corporations, such as Slack, Atom, Wordpress and VK messenger as well as startups (electronjs.org/apps, 17.12.2017). Among the main reasons for its growing popularity is the fact that software developers can avoid learning C-based languages to build software, if they already know JavaScript. As already mentioned above, the application built with Electron are cross-platform.

However, Electron is not unique in its purpose. Its main competitor NW.js (Node-Webkit), released in 2011, serves the same functionality in general. However, Electron has seemingly a larger list of applications built on it (electronjs.org/apps, 17.12.2017). The number of contributors also differ greatly: 94 for NW.js (github.com/nwjs/nw.js, 17.12.2017) and 704 for Electron correspondingly (github.com/electron/electron, 17.12.2017).

For the exact same reason of working across platforms, React Native, released in the early 2015, steadily gains popularity. As opposed to Electron, its aim is to facilitate cross-platform mobile development. Based on React web framework, it makes it possible to build applications for both iOS and Android systems, using JS code, which is 87% the same on both, with only 13% of code written for each system specifically (Gackenheimer, 2015). However, a number of drawbacks still exists in the process of native mobile development with React Native. For example, the performance of the application might be worse in comparison with natively developed and optimized applications. Also, the limited amount of libraries may complicate the development of a more complex programs (Eisenman, 2015). However, the pace of React Natives' evolution is incredibly fast: new releases come on monthly basis. This eliminates existing bugs and brings noticeable improvements to the framework.

Facebook (company-originator) has a development team with around 20 developers working on improving React Native framework (which includes the addition of new mobile platforms, new features, and bug fixing). Surprisingly, the external community also participates in the evolution of the framework: as for the beginning of

2016, the number of commits from external contributors was more than twice as high as one from company's employees. (Martinez, 2017)

Needless to say, that all of the platforms for mobile and desktop application development, such as Electron and React Native, have their own advantages and drawbacks. Therefore, it is useful to understand the mechanics and principles of potentially chosen platforms to decide between alternatives beforehand. The next chapter introduces different methods and tools for modern mobile and desktop development.

3. DIFFERENT APPROACHES TO MOBILE AND DESKTOP DEVELOPMENT

The mobile phone has become essential in the contemporary world. People cannot imagine a smartphone without a photo camera, music player or Internet connection. These features are provided by several Operating Systems (OS). The ones in the scope of this thesis are Android and iPhone Operating System (iOS). They are the main competitors on the market and both strive achieve the best feasibility and quality in order to satisfy their customers. Each OS possesses features, which the other does not provide. The differences, demanding certain level of expertise, are defined as OS fragmentation. (Halidovic, 2014) The process of creation of an application designed to execute on one OS only is called native mobile development.

3.1 Native approach

The main tool of traditional native mobile development is Software Development Kit (SDK), which provides a development environment tied to a specific OS. For iOS such a SDK is represented by XCode IDE, and for Android by Android Studio. Both of them utilize Model View Controller (MVC) design pattern, introduced on the schema below (Leff, 2001).

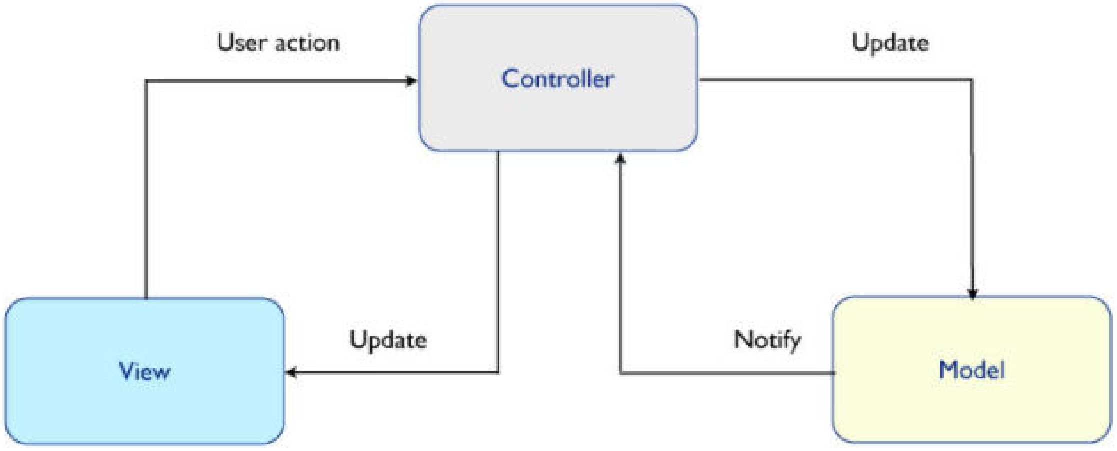


Figure 3. MVC design pattern (Zakas, 2016).

Each OS requires officially-supported programming languages. For example, Swift is required for iOS and Java for Android. The main complication of this approach is in consistency of user experience across platforms. Eventually, software development companies spend almost twice as much timing budget and financial resources to build single mobile applications functioning on both OSs. Even though building application with this method assures high quality of the features and user experience, a native development approach is not considered to be an optimal solution (Amatya, 2014).

3.2 Hybrid approach

Fortunately, other approaches than native to solving the problem of OS fragmentation exist. For example, hybrid mobile development is used widely. It can be described as a combination of a native functionality with modern web-technologies.

Native functionality is provided by means of API, allowing developers to use platform-specific features (Heitkotter, 2013). Although hybrid mobile development is considered to be a good option and provide better performance than mobile web browser, it has its disadvantages too. For instance, an application can still be slower than the one built using native approach. Also, user experience is plagued by the lack of access to some native features. (Delía, 2015) An example of hybrid mobile development platform is Xamarin, based on C# programming language.

3.3 Web Application development

The most recent approach of mobile and desktop development is Web Application development. The main aim of it is to achieve native experience by utilising web-technologies, such as HTML5, CSS and JavaScript. The component, which makes it possible is WebView. This technology allows to render web-pages inside a native application. Therefore, it is not possible to access the application without Internet connection. (Adinugroho, 2015)

3.3.1 Conventional JS application structure

Modern web applications serve various purposes, however, the architecture is often similar (Figure 4).

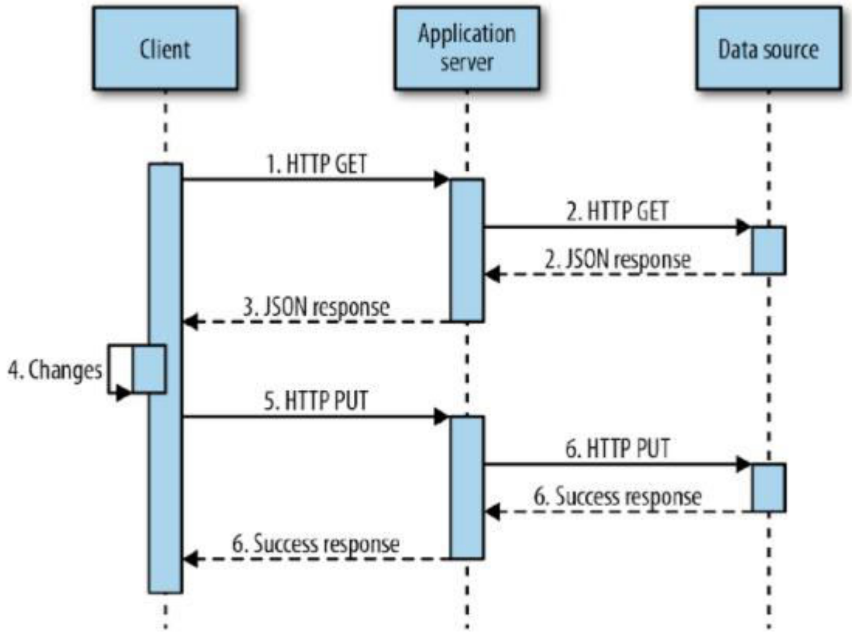


Figure 4. Data flow between database and a browser in RESTful application (Zakas, 2016).

In most of the cases, the application consists of the following components:

1. A datastore. It is a database with the data for the application. It can be either SQL or NoSQL database (Zakas, 2016).
2. Service layer (JSON RESTful Web Service Layer). Representational State Transfer (REST) is an architectural style of client-server communication. REST is designed to sustain application state according to a client and a server. In order to achieve that, RESTful web services utilize HTTP methods GET, POST, PUT and DELETE (Cowan, 2005). As can be seen from Figure 4, GET method returns data in JS Object Notation (JSON) format. The data can be then parsed and displayed on the client side. Alternatively, in step 5 and 6 of the diagram the data is sent to the database in PUT request with intent to make changes to existing data. In case the data is changed successfully, a success response is sent from the data server, otherwise, an error message is sent (Zakas, 2016).
3. Application server. In case with JS application, it is a server, written in JS. Its main purpose is to route requests and serve content to a browser.
4. Content Delivery Network (CDN), which is all the static files of an application, including JS, CSS, HTML templates, icons or images.
5. A browser

Thanks to the formed web application development structure, a number of frameworks has appeared. React Native and ElectornJS are good examples of them.

3.3.2 ElectronJS

Desktop application development has approximately the same approach as mobile development. More specifically, there is a need to build an application for each OS. Normally, .Net (based on C#) is used for Windows systems and Swift, Objective-C for OSX. For non-Windows platforms it is possible to use Java programming language. (Litayem, 2015)

However, the most recent approach is to build cross-platform applications with ElectronJS, utilizing JS only. Electron JS, previously known as Atom Shell works by means of Chromium embedded framework (CEF) for frontend and NodeJS for backend. The application runs in a browser, produced by CEF and controlled by code (ElectronJS documentation, 2018).

An access to native APIs is granted to Electron application, whereas, as a general rule, web-applications are restricted to access of such kind. Moreover, ElectronJS application has an unlimited access to all OSs events. For example, it can control files and send OS notifications. At the same time, ElectronJS can utilize any NodeJS modules, including third side modules. (Jasim, 2017)

When started, ElectronJS launches main process (main.js) and renderer processes, which introduce different windows, representing different views/pages. The main process can communicate with renderer processes by means of a tool called inter-process communication. However, communication between renderer processes is possible only through the main process (ElectronJS official documentation, 21.01.2018).

As indicated in the Figure 5, the main process (Browser process) creates application windows and controls them, while the renderer process loads webpage, supported by a JS script and its CSS styles. So, technically, ElectronJS application is not a desktop application, but a browser window, which has access to some native functionality and styles.

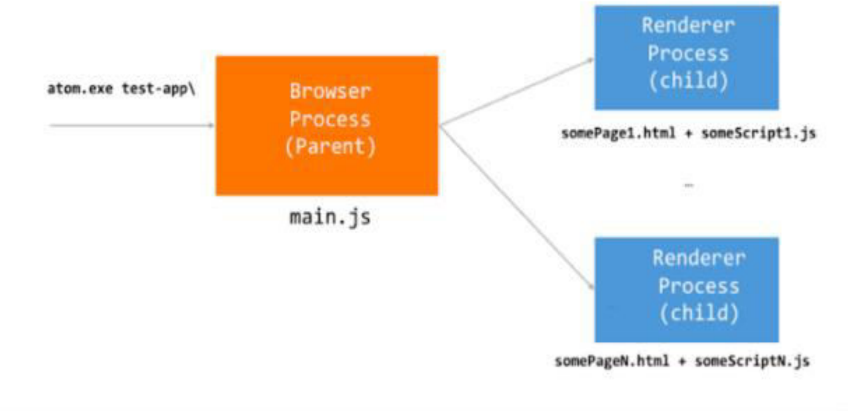


Figure 5. ElectronJS processes and communication between them (Jasim, 2017).

Among requirements to build ElectronJS application successfully is a knowledge of JavaScript, CSS, HTML and NodeJS.

3.3.3 React Native

React Native, based on similar principles as React JS, allows to develop cross-platform mobile applications. Instead of rendering everything in HTML, as ReactJS does, it renders to iOS/Andorid components. So, <*View*>, illustrated in Figure 6, would be rendered to UIView, which is iOS specific.

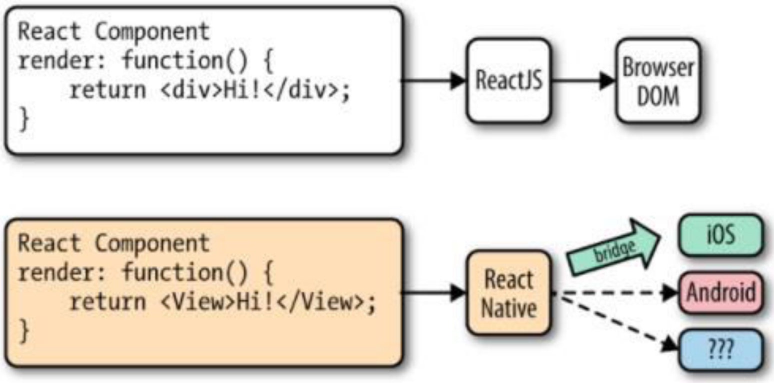
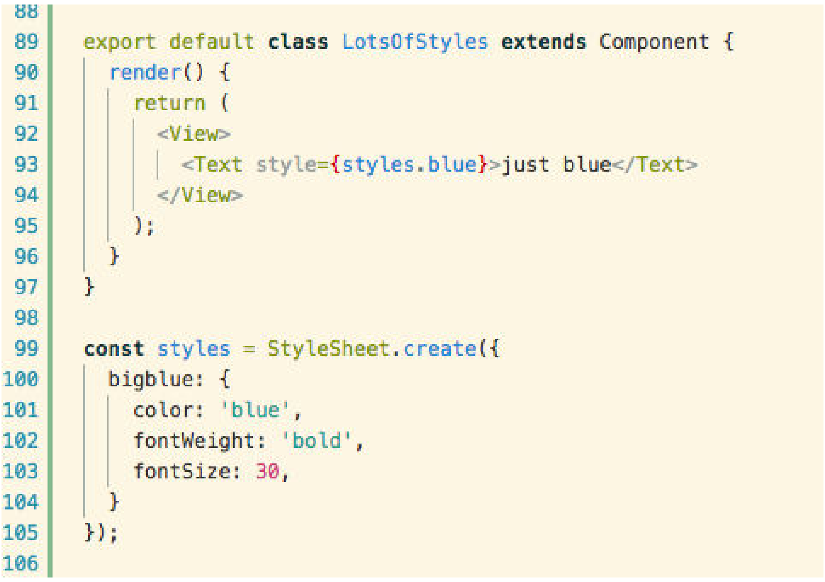


Figure 6. React Native compared to ReactJS (React Native documentation, 2018).

Although standard React Native supports only Android and iOS, it is also possible to develop mobile applications for almost any other platform. The connection between the application and a particular OS is called a bridge. Nowadays, any interested developer is capable of writing their own bridge and make a React Native application function on any OS.

Another noteworthy trait of React Native is the use of JSX, which is a combination of a markup language and JavaScript. Also, instead of regular CSS React Native utilizes CSS-like syntax. (Eisenman, 2015) A snippet of React Native page styling is presented in Figure 7.



```
export default class LotsOfStyles extends Component {
  render() {
    return (
      <View>
        <Text style={styles.blue}>just blue</Text>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  bigblue: {
    color: 'blue',
    fontWeight: 'bold',
    fontSize: 30,
  }
});
```

Figure 7. React Native styles (React Native documentation, 2018).

So, for React Native application development it would be useful to have a knowledge for JavaScript, HTML, CSS.

4. PRACTICAL EXAMPLES OF DEVELOPING JS APPLICATION OUTSIDE OF THE BROWSER

In order to demonstrate the capabilities of JS language beyond the browser, two basic applications (desktop and mobile) with the same functionality are built. The mobile application is cross-platform and supports iOS and Android. The desktop application is also cross-platform and can be run on any OS.

The applications represent a weather forecast application for the City of Turku, Finland. For this purpose, they implement the same API, provided by Yahoo Weather. The information about the current weather conditions is fetched by the applications by sending a GET request.

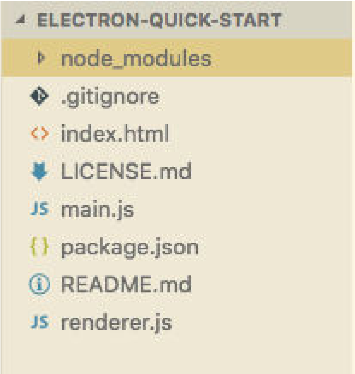
The code editor used is Visual Studio Code, provided by Microsoft and built with ElectronJS itself. For React Native simulators XCode is used. Development of both applications requires Node installed globally on the computer.

4.1 Desktop application development with ElectronJS

Installation and application structure

ElectronJS creators offer to install a quick start application with the ready structure. After the installation of the project folder by cloning it from git repository, the structure looks like the one presented in Figure 8.

The most important of these files for now is package.json. This file consists of important information about the application, such as the list of dependencies, application version, etc. It is fairly common among npm-based applications and prevents many compatibility issues.



ELECTRON-QUICK-START
- node_modules
- .gitignore
- index.html
- LICENSE.md
- main.js
- package.json
- README.md
- renderer.js

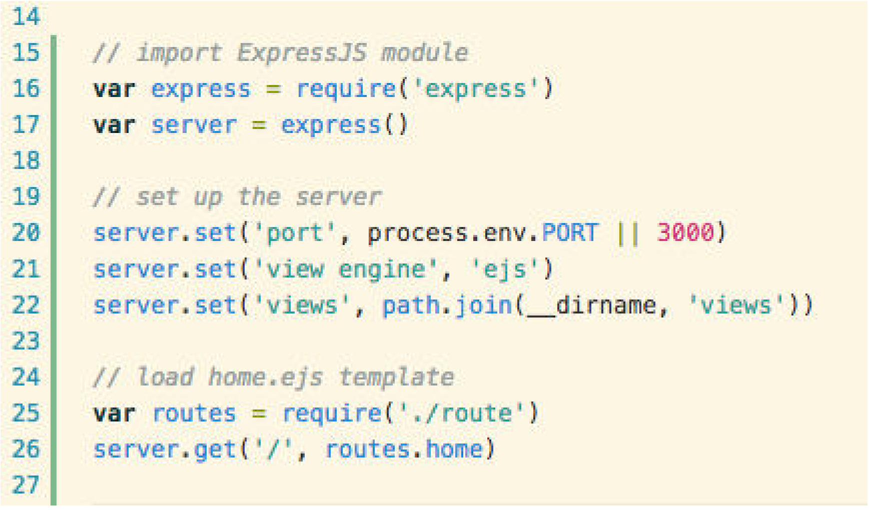
Figure 8. Initial structure of ElectronJS quick-start application.

The Node_modules folder contains modules which are needed for the application lifecycle. These can be installed by running the *npm install* command in the projects' root directory. All the modules are open-source and can be installed on demand. *main.js* contains application server settings and configurations. The native browser window is being called from this file as well.

When developing with Electron JS, there are several alternatives for templating. In this case, EJS templating is used, which is a mixture of HTML and JS. It is a common practice to split the code into several directories and files. In this case, */views* directory stores all the EJS templates and file *routes.js* holds all the applications URLs mapped to the templates.

Application server

For the application server, ExpressJS server framework for NodeJS is used. It is installed as simply as any other npm module. Then, the module is imported to main.js file. After adding some settings (Figure 9), the server is ready to handle requests. At this point the application can already be launched by running *npm start* and the contents of *home.ejs* template are displayed.



```
// import ExpressJS module
var express = require('express')
var server = express()

// set up the server
server.set('port', process.env.PORT || 3000)
server.set('view engine', 'ejs')
server.set('views', path.join(__dirname, 'views'))

// load home.ejs template
var routes = require('./route')
server.get('/', routes.home)
```

Figure 9. Application server settings for ElectronJS application.

Handling API requests

For request sending, ElectronJS has several options. The one chosen for the current application is the Request library. It is installed as any other npm module and provides a functionality of sending HTTP requests. In this case, we send a GET request to Yahoo Weather API endpoint. The request URL already contains the City of Turku as a static parameter.

The data returned is in the format of the string even though it looks similar to JSON. In order to be used properly, it is converted to JSON by using a common JS function. After that, the data can be passed to EJS templates and used there.

4.2 Mobile application development with React Native

Installation and application structure

React Native documentation contains clear steps for the quick project launch. Firstly, we install *create-react-native-app* command line utility globally on the computer. The purpose of it is the creation of the ready-to-launch project directory and files in it. Secondly, we run *create-react-native-app WeatherInTurku* command to create the root project directory and its structure.

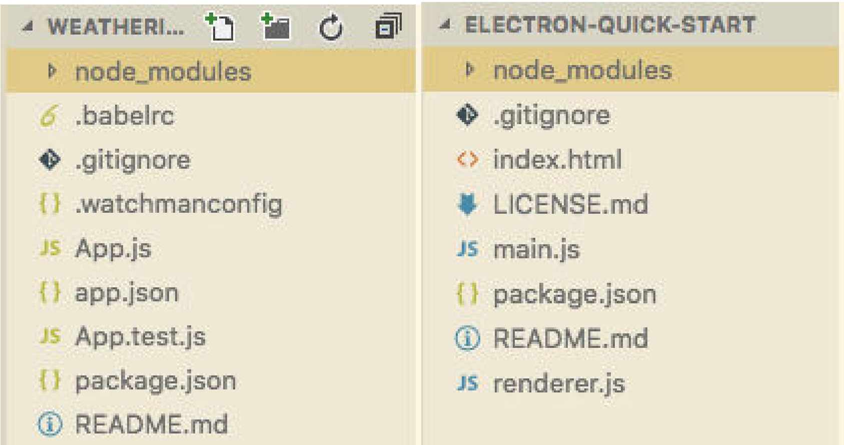


Figure 10. React Native and ElectronJS application folder structure.

As can be seen from Figure 10, the React Native folder structure resembles the ElectronJS structure: it also contains *node_modules* folder and *package.json* file. However, the rest of the files are different: App.js is the starting point of the application. This file contains the component we use to display the weather forecast and corresponding styles.

It worth noting that React Native code structure is very similar to React JS: it uses components and *render* function to display them. However, the components are mobile specific. Instead of regular <*div/*> and <*p/*> tags React Native utilizes <*View/*> and <*Text/*>, for example. Also, components might vary from one OS to another.

Server setup

The application server is run identical to ElectronJS: by running *npm start*. However, to run the application itself on the simulator, we should open a new Terminal/Command line window and execute *npm run ios* or *npm run android,* depending on what device the application is developed. After running the command, we can see a working application in a mobile phone simulator.

One of the noteworthy React Native features is hot reload. This means that the application a programmer is developing reloads automatically when changes are saved in the code. This makes development faster, compared to ElectronJS, by eliminating unnecessary server restarts.

Handling API requests

In the React Native API, requests are typically handled by the *fetch* function. This function sends the GET request in this case and returns data in JSON format. Even though the function is different from what was used in Electron JS application, the fact that it is still JS causes it to look alike. After the data is received, it can be displayed in a View tag. The final result looks exactly the same as Electron JS application and uses the same data.

5. CONCLUSIONS

Web development is progressing fast. Only 10 years ago, JS was used as a scripting language and today we observe world-known companies migrating their services to server-side JS. Four years ago, desktop application with JS was unusual, and now, according to *npm-stat*, ElectronJS is downloaded more than 600 thousand times per month (NPM-stat, access date 17.05.2018). These examples point to the fact that JS gains its popularity in the web development world and its application expands.

As famous companies start to use JS for their own services, more and more web developers all over the world shift their attention towards JS. Many of them not only use numerous libraries and frameworks, which are already provided, but they actively contribute. As a result, the community grows and JS develops even further.

Application development carried out in this thesis with ElectronJS and React Native shows that it is possible to build a cross-platform application from scratch within a week. As both desktop and mobile applications are written in JS the code reusability is fairly high. The developed application is fully cross-platform, but it uses different frameworks for mobile and desktop development. Hence, implementation differences may be present and need to be accounted for. In case of React Native, some of the code needs to be changed to comply with Android or iOS.

Research shows that JS is successfully used in many areas nowadays, however, it is not always the best option. In some cases, using the native counterparts would be more reliable and timely.

To conclude, JS holds a specific place in web, mobile and desktop development nowadays. Considering the rapid evolution pace, JS will probably continue to expand in different areas outside of the browser: IoT, VR or other environments and systems in the future. However, JS will never be used for all the possible development, due to the fact that there are better suited and more sophisticated languages for different purposes and needs.

REFERENCES

Adinugroho, T.; Bernadi Gautama, J. Review of multi-platform mobile application development using WebView: Learning management system on mobile platform. Procedia Computer Science 59, 2015.

Alkhars, A. Cross-Platform Desktop Development (JavaFX Vs. Electron), 2017.

Amatya S.; Arianit K. Cross-platform mobile development: challenges and opportunities. ICT Innovations 2013, pp. 219-229. Springer, Heidelberg, 2014.

Bootstrap official documentation. Available at: http://getbootstrap.com/. Access date: 18.11.2017.

Chaffer, J. Learning JQuery 1.3: Better Interaction and Web Development with Simple JavaScript Techniques. Packt Publishing Ltd, 2009.

Charland, A.; Leroux B. Mobile application development: web vs. native. Communications of the ACM54, no.5 2011.

Cochran, D. Twitter bootstrap web development how-to. Packt Publishing Ltd, 2012.

Cowan, J. RESTful Web Services 2005. Available at:
http://miageprojet2.unice.fr/@api/deki/files/724/=restws.pdf. Access date: 29.05.2018.

Cuomo, J. JavaScript Everywhere and the Three Amigos, IBM Developer Works, 2013
Daly, Joshua. jQuery Essentials 2016.

Dayley, B. Node. js, MongoDB, and AngularJS Web Development. Addison-Wesley Professional, 2014.

Delía L.; Galdámez N.; Thomas P.; Corbalán L.; Pesado P. Multi-Platform Mobile Application Development Analysis. Research Challenges in Information Science (RCIS), 2015.

Eisenman, B. Learning React Native: Building Native Mobile Apps with JavaScript. O'Reilly Media, Inc., 2015.

ElectronJS official documentation.
https://github.com/electron/electron/tree/master/docs, Access date: 21.01.2018.

Elliott, E. Programming JavaScript applications: Robust web architecture with node, HTML5, and modern JS libraries. O'Reilly Media, Inc., 2014.

Grover, D. ES6 for Humans: The Latest Standard of JavaScript: ES2015 and Beyond. APress, 2017.

Halidovic, R.; Karli G. Cross-Platform Mobile App Development using HTML5 and JavaScript while leveraging the Cloud. IOSR Journal of Engineering, Vol. 04, Issue 0 2, 2014.

Hartmann, G., Stead Asi DeGani, G. Cross-platform mobile development. Available at: https://wss.apan.org/jko/mole/Shared%20Documents/Cross-Platform%20Mobile%20Development.pdf. Access date: 29.05.2018.

Henning, H.; Hanschke, S.; Majchrzak, T.. Evaluating Cross-Platform Development Approaches for Mobile Applications. Lecture Notes in Business Information Processing 2013.

Jasim, M. Building Cross-Platform Desktop Applications with Electron. Packt Publishing Ltd, 2017.

Konicek, M. React native: A year in review. Available at: https://code.facebook.com/posts/597378980427792/reactnative-a-year-in-review/. Access date: 29.05.2018.

Leff, A.; Rayfield, J.T. Web-application development using the model/view/controller design pattern. In Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International (pp. 118-127). IEEE.

Litayem, N.; Bhawna, D.; Sadia, R. Review of cross-platforms for mobile learning application development. International Journal of Advanced Computer Science and Applications 6. IEEE software 30, no.1 2015.

Martinez, M. Towards the Quality Improvement of Cross-platform Mobile Applications. Mobile Software Engineering and Systems, 2017.

Neer, K., F. Lyle III, W. History of JavaScript. History 2013.

Harrell, J., NodeJS at PayPal, Paypal Engeneering 22.11.2013. Available at https://www.paypal-engineering.com/2013/11/22/node-js-at-paypal/. Access date: 29.05.2018.

NPM statistics. Available at: https://npm-stat.com/charts.html?package=electron. Access date: 17.05.2018.

React Native documentation. Available at: https://facebook.github.io/react-native/docs/style.html. Access date: 18.04.2018.

Serrano, N.; Josune H.; Gallardo, G. Mobile web apps. IEEE software 30, no.5 2013.

Teixeira, P. Professional Node.js: Building Javascript based scalable software. John Wiley & Sons, 2012.

Zakas, N. Understanding ECMAScript 6: the definitive guide for JavaScript developers. No Starch Press, 2016.