Bachelor’s Thesis

Information Technology

2018

Anssi Hautaviita

DEVELOPING A WEB APPLICATION ON THE MEVN STACK

– The architecture of a full-stack JavaScript application

BACHELOR’S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology

2018 | 33

Anssi Hautaviita

DEVELOPING A WEB APPLICATION ON THE MEVN STACK

- The architecture of a full stack JavaScript application

The programming language JavaScript was introduced to the public in 1995 and was initially dedicated purely to the client-side, that is, it was primarily powering small applets dedicated to enhancing user experience by providing more interactability to websites.

However, over the years – especially during the last five years or so – JavaScript and its ecosystem have taken large strides forward and evolved to a fully functional basis of highly advanced and robust web applications, now powering some of the largest services in the world, such as Netflix, Uber and PayPal.

This thesis describes the JavaScript programming language and its libraries in modern Web Development. The abilities of JavaScript and its frameworks are demonstrated by building a full stack web application utilizing the MEVN web stack, consisting of the Node.js runtime, MongoDB database engine, Vue.js frontend framework and the Express.js web framework. The end product is a fully functional working hours tracking application, aimed for small businesses.

KEYWORDS:

JavaScript, NodeJS, Express, Vue, MongoDB

CONTENTS

LIST OF ABBREVIATIONS (OR) SYMBOLS

JS	JavaScript
PHP	PHP: Hypertext Preprocessor
MEVN	MongoDB, Express, Vue, Node.js
VPS	Virtual private server
EJS	Templating language that generates HTML markup with JavaScript
HTML	Hypertext Markup Language
UI	User interface
I/O	Input/Output
Vue	Synonym for Vue.js
Node	Synonym for Node.js
NPM	Node Package Manager
Express	Synonym for Express.js
ODM	Object Data Modeling
GET	GET request: HTTP method used to request data from a specified resource
POST	POST request: HTTP method used to send data to a server to create/update a resource

1 INTRODUCTION

Since its initial public introduction in 1995, the programming language JavaScript has been one of the main building blocks of the world wide web infrastructure. However, for a long time it was dedicated only to serve as an easy-to-approach client-side scripting language, mainly being used to enhance the user experience with small applets – such as counters, banners and animations on otherwise static web pages. [1]

Over the years the language has seen new features emerge, and during the last decade JavaScript has also been gaining traction as a server-side language, as a result of the development of the cross-platform JavaScript runtime Node.js. First introduced in 2009, Node.js allows developers to write and run JavaScript code both on the client-side as well as on the server. It runs “single-threaded, non-blocking, asynchronously programming, which is very memory efficient” [2] – one of the main reasons for its popularity surge is indeed the quickness and scalability of the server environment compared to its competitors, most popular being PHP and ASP.NET [3].

The aim of this thesis is to build a full-stack web application to describe the development process of an application that utilizes the MEVN stack as well as the Node.js infrastructure surrounding the four core components of the software. The MEVN stack consists of a MongoDB NoSQL database, Express web framework, Vue.js frontend framework and the Node.JS JavaScript runtime. The whole application is running on JavaScript code, with the application data stored in JSON-like schemas in a MongoDB database.

The application that is developed to demonstrate and describe the development process and the previously mentioned software libraries is a time-tracking application, aimed for small businesses that have to comply with the working hours tracking laws of Finland and want to achieve that by logging their hours to a simple, easy-to-use web application.

2 APPLICATION INFRASTRUCTURE

This chapter describes the core technologies and frameworks that were utilized to construct the web application. The application frontend is mainly built with the Vue.js framework, and the backend consists of a Node.js runtime, Express web application framework and a NoSQL MongoDB database.

2.1 Frontend

The application frontend consists of the UI elements of the web application that are visible and interactable by the end user via the browser. Main frontend logic was built using the Vue.js JavaScript framework. Vue.js was utilized to construct the main frontend components to provide a seamless and intuitive user experience. This framework allows the visible components to be re-rendered without page reloads when user interaction triggers an action that manipulates the application state, e.g. a database query that returns updated data to the application frontend that is then rendered to the components visible on the screen. This cycle is visualized in Figure 1.

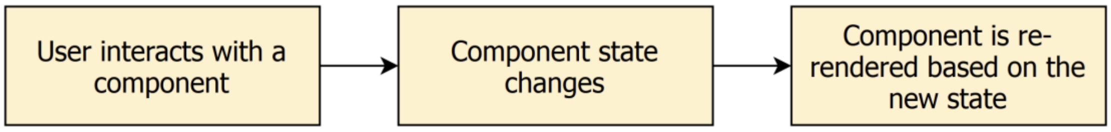

User interacts with a component
Component state changes
Component is re-rendered based on the new state

Figure 1. State cycle of a Vue component after user interaction.

Vue.js is a progressive framework for building user interfaces [4]. It is an open-source project, which means that the source code of the framework is inspectable, modifiable and enhancable to anyone [5]. Vue.js makes application state management simple and intuitive – Vue models are “just plain JavaScript objects” [6] which makes interacting with the application state straightforward. When action triggers a modification on the application state, the view updates automatically. Vue achieves this by converting all data properties of the component to getters and setters, which enable Vue to perform dependency-tracking and change-notification when properties are accessed or modified [7].

Some simpler sections of the application – e.g. the password change form – do not utilize the framework, and were instead composed with static EJS templates.

2.2 Backend

The application backend consists of a Node.js runtime, Express.js web framework and a NoSQL database called MongoDB. These technologies provide a seamless and straightforward development experience and also keep the web application lightweight and fast to use.

2.2.1 Node.js runtime

Node.js is an open-source JavaScript runtime built on Chrome's V8 JavaScript engine. The Node.js project is supported by the Node.js foundation. However, it has various contributors from all around the world, and is one of the most popular repositories in the version control platform GitHub.

JavaScript is a single-threaded language, which means that calling a synchronous method blocks all other actions until that method has been run successfully. An example of a synchronous call is presented in Code snippet 1: in the first line the file system module fs is imported, and in the second line the method readFileSync is called. The system will wait until file.md has been read successfully and the contents have been stored to the constant fileData.

Code snippet 1. Example of a synchronous function.

```
1   const fs = require('fs')
2   const fileData = fs.readFileSync('/file.md')
3   console.log("File read successfully.")
```

Node, however, uses an event-driven, non-blocking I/O model that makes it lightweight and efficient [8]. It leverages the JavaScript event loop to create applications that easily service multiple concurrent events by utilizing asynchronous methods.

An example of an asynchronous call is presented in Code snippet 2: in the first line the file system module fs is imported again, and after that the file file.md is read, this time with the asynchronous method readFile. When the file reading process has completed, a callback function is triggered with the variables err (error) and fileData: if an error has occurred during the file reading process, the variable err will contain information about the error, and the variable fileData contains the contents of the read file.

Code snippet 2. Example of an asynchronous function.

```
const fs = require('fs')
fs.readFile('/file.md', (err, fileData) => {
  if (err) throw err
  // do something with fileData here
  console.log(“File read successfully.”)
})
console.log(“File reading started, continuing ahead while it’s
processing.”)
```

With this method the code execution does not stop at the file read call, and the program can continue to execute other code below this part. This kind of functionality makes Node very agile and scalable.

Node Package Manager

One of the best features of Node is its package manager NPM. With NPM the developer can make use of thousands of modules and packages created by other developers. This provides a way to get applications up and running quickly, since all the features do not have to be designed and programmed from ground up. The beauty of full stack JavaScript programming is that since the same language is used throughout the application, the modules installed from NPM can be used both in the backend and in the frontend.

2.2.2 Express web application framework

“Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications” [9]. It is used to speed up the creation of web applications and services, and it achieves this by offering a range of ready-to-use features for building HTTP server functionality. In its most simplistic form, a web server built with Express could appear as follows:

Code snippet 3. Source code for a simple Express web server.

```
1   var express = require('express')
2   var app = express()
3
4   app.get('/', (req, res) => {
5      res.send('<h1>Welcome to the front page!</h1>')
6   })
7
8   app.listen(8000)
```

Code snippet 3 shows, how quick it is to boot up an Express server. In llines 1 and 2 the module is imported and utilized. In line 4 a GET route to address “/” (the front page, e.g. https://google.com/) is defined. In line 5 a response is established with res.send and in line 8, the app is ordered to listen for calls on port 8000 on the server. If this snippet of code was saved with the filename server.js, an HTTP server could be booted up via the command “node server.js” so one could then navigate to the address localhost:8000 on their browser to see the “Welcome to the front page!” message.

The app.get, res.send and app.listen commands in this snippet are great examples of key functionalities the Express framework provides, that would otherwise need to be constructed from ground up. Express has a wide range of similar tools to support the needs of developers. The framework is also extremely extendable by sub-modules such as the authentication middleware Passport.js and the HTTP header security middleware Helmet [10].

Express is an open source Node.js framework, and it is installable via the Node Package Manager.

2.2.3 MongoDB

MongoDB is an open-source document database that stores data in JSON-like documents, instead of tables and rows as in relational databases (such as MySQL and Microsoft SQL Server). A record in MongoDB is a document which is a data structure composed of field and value pairs – the values of fields may include other documents, arrays, and arrays of documents. This means that the fields can vary from document to document and data structure can be changed over time. [11]

MongoDB is queried and navigated with base commands such as insert, find, update and delete. In addition to these fundamental actions, it supports a wide range of more detailed commands to provide effective tools for data interaction and manipulation.

“MongoDB is essentially built on an architecture of collections and documents. Documents comprise sets of key-value pairs and are the basic unit of data in MongoDB. Collections contain sets of documents and function as the equivalent of relational database tables.” [12]

Mongoose

MongoDB is widely used in full-stack JavaScript applications due to its dynamic and scalable nature. One of the most popular frameworks used with MongoDB and Node.js is the object modeling tool Mongoose. It extends MongoDB and Node.js to provide a seamless method to define models for web applications and services. Mongoose also includes built-in type casting, validation, query building, business logic hooks and other such functions. [13] Mongoose is practically a connecting link between the Node.js application logic and the MongoDB database.

3 DEVELOPMENT AND PLANNING TOOLS

This chapter describes the tools used in the development process, and it gives a brief introduction of the planning phase of the software project itself.

The project mostly utilizes tools and libraries that have become full-stack JavaScript development industry standards for writing understandable, clear code and bundling it for deployment.

3.1 Concepting and visualizing the application

Before starting to actually build and program the visual logic of the application, a raw concept of the end product was drafted. Having a visual baseline as a model for development speeds up the programming process, and also helps in visualizing the backend logic that powers the application behind the scenes. In this case, the application visualization process was intentionally kept as short as possible. Only the fundamental elements were decided beforehand to provide a skeletal structure to be used as baseline.

A wide range of wireframing and prototyping tools exist for web developer and designers. Some of the most popular ones are Balsamiq, Sketch and Wireframe.cc. The purpose of these tools is to quickly create shareable – and in some cases, collaboratable – mockups of the product.

In this project, however, no prototyping tools were applied in the sketching process due to the simple nature of the user interface and the fact that the project was a one-man operation, with no need for collaboration between different parties. Rough drafts were sketched by writing HTML and utilizing a CSS framework called Bootstrap. The advantage of writing HTML mockups is that the files can be viewed instantly on the actual devices that the application itself will be used with, simply by opening the HTML file.

At this point, it was not yet clear how the components itself would look like in their final forms, but chunks of the browser view could be already reserved for different functionalities to serve as a guideline. Two main interfaces that needed to be sketched before starting the actual development process were:

- Calendar view
- Reports view

The calendar view is constructed of four sub-components: the calendar component, the information section, the daily notes field and the clickable bar that the user can interact with to highlight and save periods of time that have been active during that day.

With the help of Bootstrap's grid system – which, in short, divides content to rows that have 12 columns [14] – an HTML mock-up was written to visualize the calendar view consisting of a 3/12 width calendar component, a 5/12 width information area, a 4/12 width daily notes component and a full width component for the time input:

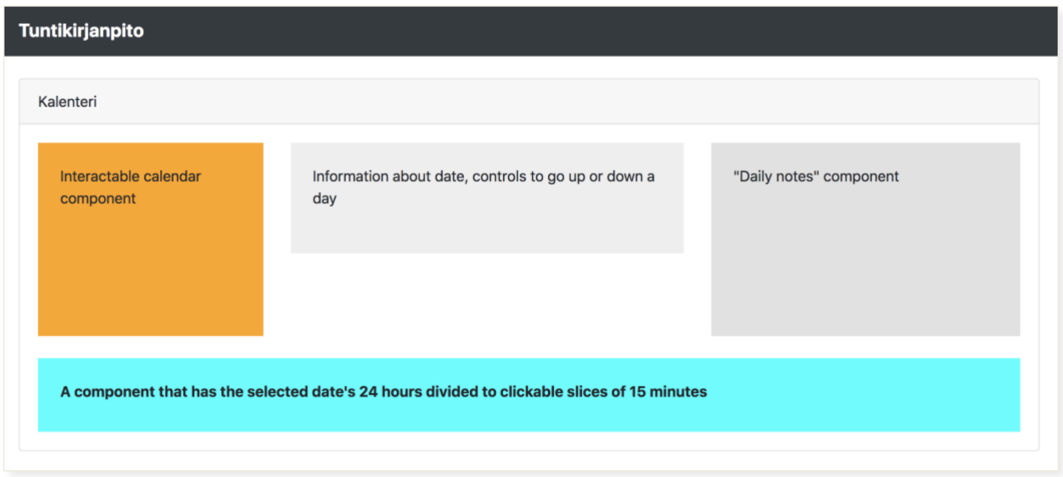

Tuntikirjanpito
Kalenteri
Interactable calendar component
Information about date, controls to go up or down a day
"Daily notes" component
A component that has the selected date's 24 hours divided to clickable slices of 15 minutes

Figure 2. Mockup of the calendar view.

The reports view serves as a page where the user can easily view and export their entries from longer periods of time. It consists of three selectors: the starting date, the ending date and the user that the report should be generated of. Under these selectors, there should be buttons that the user can click to generate reports of the visible data to a format viewable in a spreadsheet program, such as Excel or Google Sheets. Under the buttons should lie an HTML table that has three columns: date, hours worked and notes. This concept was again visualized by writing an HTML mockup based on these ground rules:

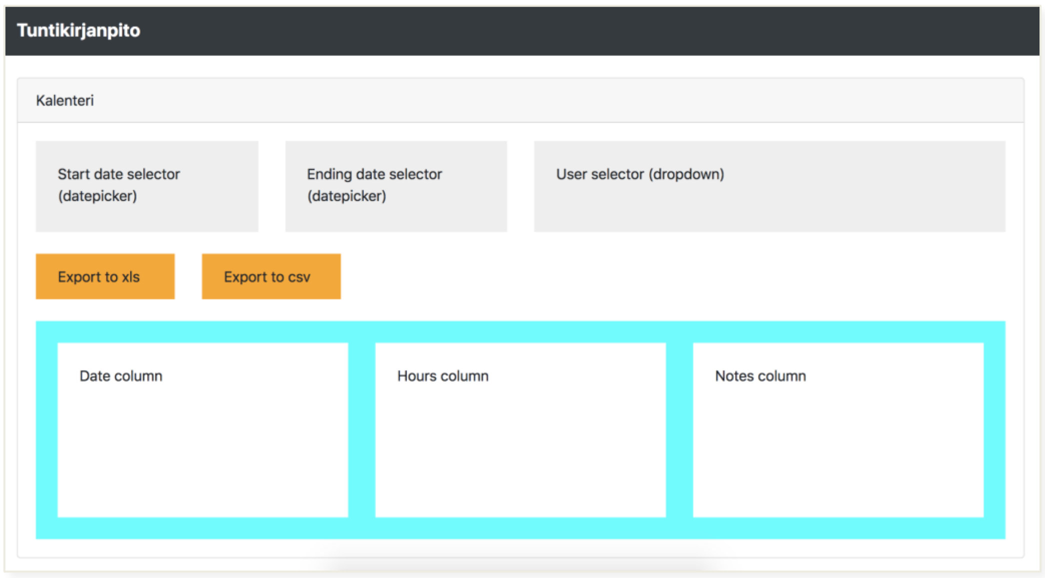


Figure 3. Mockup of the reports view.

3.2 Development tools

To create a full-stack JavaScript web application, one needs to have a set of tools and software to run a development environment. The development environment described below is suitable for macOS devices, since that is the operating system that was used during the development of the application.

3.2.1 Integrated development environment

An IDE (Integrated Development Environment) is an application that facilitates application development. In general, an IDE is a graphical user interface (GUI)-based workbench designed to aid a developer in building software applications with an integrated environment combined with all the required tools at hand. [15]

During this project, a popular IDE Visual Studio Code (often referred to as Visual Code) was used to write the necessary JavaScript code. Visual Code is a desktop application that is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js [16]. The IDE itself is developed by Microsoft, and it is built with an open-source desktop application framework called Electron [17]. “Electron is an open source library developed by GitHub for building cross-platform desktop applications with HTML, CSS, and JavaScript. Electron accomplishes this by combining Chromium and Node.js into a single runtime and apps can be packaged for Mac, Windows, and Linux.” [18]

The installation of Visual Code is very straightforward. An installer can be downloaded from the Visualstudio.com website, and it can be installed with a couple of clicks, without any advanced configuration.

3.2.2 Node.js and Nodemon

The Node.js runtime can be downloaded and installed from the official website nodejs.org. The runtime is installed globally, which means that the program can be then used via the command node. An installation can be verified with the version command. If the Node.js runtime has been installed successfully, it will display the current installed version of the software:

Code snippet 4. Command to verify that Node.js is installed.

```
~ node -v
v8.9.4
```

Nodemon is an open-source Node.js add-on that will watch the files in the directory in which nodemon was started, and if any files change, nodemon will automatically restart the node application. [19]

Nodemon can be installed globally (operating-system-wide) by using npm:

Code snippet 5. Command that installs the Node.js add-on Nodemon.

```
~ npm install -g nodemon
```

When the add-on has been installed, node programs can be launched with the additional functionality via the nodemon command. For example, if the root file of the web application is index.js, it can be launched via the command “nodemon index.js”. After this, if index.js or any files that it references are changed, nodemon will automatically restart the node server.

3.2.3 Webpack

“Webpack is a static module bundler for modern JavaScript applications. When webpack processes your application, it internally builds a dependency graph which maps every module your project needs and generates one or more bundles.” [20]

A modern JavaScript frontend framework like Vue.js consist of so many modules, building blocks and other components that it has become necessary to bundle all of the source code to a single output file, that is run in the end user’s browser. Sometimes the modules are also utilizing features that are not available in some browsers yet, and they need to be transformed to a form that is processable by most, if not all browsers. Webpack allows the developer to reference all the application files in one main file (e.g. index.js) that is then bundled by Webpack to one output file (e.g. app.js) that can be run on most browsers.

The index file of this project’s frontend code can be seen in Code snippet 6. It servers to illustrate the logic of the module bundling standard used in modern JavaScript web development: external modules (lines 2-4) are imported from repositories installed via the Node Package Manager, and the project files (lines 1 and 5-6) are imported from the current project environment. These are then utilized in the code below (lines 8-28).

Code snippet 6. Index file of the web application frontend.

```
1   import './bootstrap'
2   import Vue from 'vue'
3   import VuejsDialog from 'vuejs-dialog'
4   import Tooltip from 'vue-directive-tooltip'
```

```
import App from './App.vue'
import router from './router'

Vue.use(Tooltip, {
  delay: 0,
  placement: 'bottom',
  class: 'tas-tooltip'
})

Vue.use(VuejsDialog)

window.moment = require('moment')
window.moment.locale('fi')

window.swal = require('sweetalert2')

if (document.querySelector('#app')) {
  new Vue({
    el: '#app',
    router,
    template: '<App/>',
    components: { App }
  })
}
```

Webpack then takes this code, checks for all references made to external sources and fetches all the necessary code to a single file which can then be implemented in the browser. This process is illustrated in Figure 4.

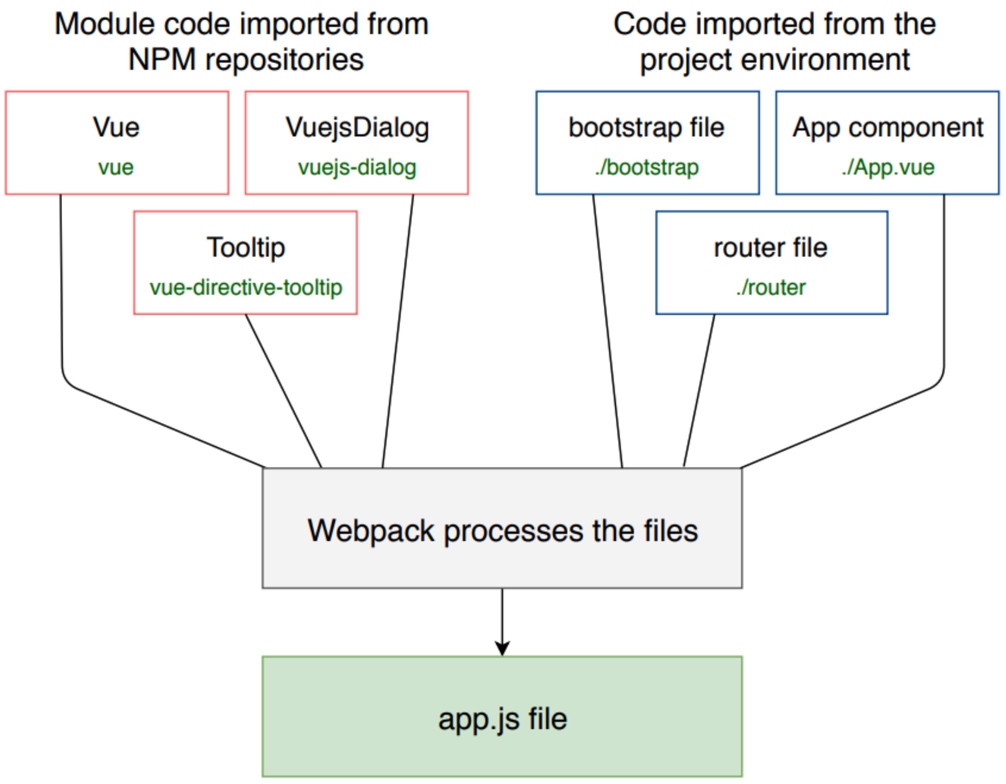


Figure 4. Illustration of Webpack operation logic.

3.2.4 ESLint

“ESLint is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code, with the goal of making code more consistent and avoiding bugs.” [21] With ESLint the developer can set guidelines for the project that all the contributors need to follow, or follow some industry standards theirselves with pre-defined guidelines.

Readable and re-usable code is becoming ever more important in the “modular era” of web development. Projects are often open-source, and they get contributions from all over the world, from parties that have never met each other before. Programmers can have very different styles and concepts that they use in their code, and this can cause frustration. By defining ground rules that all developers need to follow before their code can be used in the project, a software project can and will be easier to manage and plan for.

In this project, ESLint was integrated to the IDE Visual Code. Some of the main rules that were set for this project:

- No semicolons at end of lines, since they are not needed in ES6
- Tab-indent equals two spaces
- No unnecessary empty lines
- Files should have one empty line in the end

3.3 Version control

”Version control systems are a category of software tools that help a software team manage changes to source code over time. Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.” [22]

In this project, the version control system Git was used. Git is a free and open source solution that is used in all sorts of projects, from small practice projects like this one all the way to enterprise software.

Git allows the developer to divide the version control repository to different branches. With branches, the code can be split when deemed necessary – it is often used to separate features from the main timeline until they have been completed. When a feature branch of this kind is finished, all of the new code and the made changes can then be merged to the main branch.

3.3.1 Setting up a Git repository

Setting up a Git repository in a development machine is very straightforward and fast. Git can be installed with the homebrew tool (only in OS X) with the command “brew install git”. After this, Git is available globally in the system.

After the installation is complete, a new repository can be created by navigating to the project folder and entering the command “git init”. The version control system will track all changes made in this folder after the init command has been entered. Code can then be commited to the repository with commands such as “git add .” and “git commit”.

3.3.2 Connecting a Git repository to GitHub

GitHub is a web-based hosting service for version control using Git. By deploying the application code to GitHub, the developer can easily make it accessible to others, and at the same time verify that the changes are backed up in some other location than the local machine.

Connecting a local repository to the GitHub counterpart is done by navigating to the project folder and entering the command “git remote add origin [repository URL]”. Of course, the repository should exist in GitHub before this can be executed.

4 IMPLEMENTING APPLICATION FUNCTIONALITY

This chapter is dedicated to describing the process of programming the functionality of the application. It does not reference every file in the project, or depict each step taken during the development process, but it aims to provide a clear explanation about fundamental parts of the program that employ the MEVN stack. The stack and the main employment of its parts can be broken down in the following manner:

- **M**ongoDB: used as the database engine, that stores the users and the data generated by users. Use case detailed in subchapter 4.2
- **E**xpress: used as the web framework that routes the user requests between the end user and the database. Use cases detailed in subchapter 4.1
- **V**ue: used as the frontend framework to advanced user interface logic in the Calendar view and the Reports view. Use cases detailed in subchapter 4.3
- **N**ode: used as the application engine that powers all functionality between the user and the server. Used application-wide

4.1 URL structure of the application

The URL structure described in Table 1 was defined in the routes.js file, which is the basis for the application routing. Most of the routes rely on the Passport.js middleware, which is explained more in detail in the next subchapter.

Table 1. Application URL structure.

Method	URL	Description
GET	/	Utilizes the Passport.js middleware to verify if user is logged in. If they are, displays the Calendar view. Otherwise redirects the user to /login
GET	/reports	Utilizes the Passport.js middleware to verify if user is logged in. If they are, displays the Reports view. Otherwise redirects the user to /login
GET	/login	Utilizes the Passport.js middleware to verify if user is logged in. If they are not, displays the Login form. Otherwise redirects the user to /
POST	/login	Checks the email and password, and logs the user in if a match is found from the database. Otherwise redirects the user to /login
GET	/signup	Utilizes the Passport.js middleware to verify if user is logged in. If they are not, displays the Signup form. Otherwise redirects the user to /
POST	/signup	Checks the email, name and the password and creates a user if the email does not yet exist in the database.

4.2 User authentication

If an application is supposed to have a variation of users, it needs to have something to represent the users and their data in the database. In addition to this, there has to be a method to authenticate the users and the usage sessions reliably. In this application, it was chosen to implement the Mongoose and Passport.js libraries to create a simple and straightforward method to register and authenticate users as well as store their data.

4.2.1 Behind the scenes: user database model

In this application, each user entry in the database has the following three attributes: a name, an email address and a password. To achieve this concept in code, a representative database model for users was created:

Code snippet 7. The User.js model file.

```
1   var mongoose = require('mongoose')
2   var bcrypt = require('bcrypt-nodejs')
3
4   var userSchema = mongoose.Schema({
5    local: {
6      name: String,
7      email: String,
8      password: String
9    }
10  })
11  userSchema.methods.generateHash = function (password) {
12    return bcrypt.hashSync(password, bcrypt.genSaltSync(8), null)
13  }
14
15  userSchema.methods.validPassword = function (password) {
16    return bcrypt.compareSync(password, this.local.password)
17  }
18
19  module.exports = mongoose.model('User', userSchema)
```

On the first line of Code snippet 7, the MongoDB ODM library Mongoose is imported, and the second line the password hashing library bcrypt-nodejs is imported as well. After the imports, the database schema for a user is defined on lines 4 to 10, utilizing the Mongoose library that works as a link between the Schema class and the MongoDB database. After the user schema has been defined, we finalize the user model file by adding functions to hash and verify passwords, and finally export the User model at line 19. By defining models or practically any sort of data in the module.exports of a JavaScript file, it is possible to then import that data in some other file.

4.2.2 User signup and login

The application needs registration and login pages so the users can create accounts and start using the service. To achieve this, a signup form, a login form and the corresponding under-the-hood functionalities were created to allow end users to create new accounts and verify login attempts made to the old ones to and from the database.

Signup

As the User database model contains three fields – name, email, password – the signup form should ask for these values. Since this page does not have any other functionality than the form, it was constructed with simple HTML instead of creating new Vue components, illustrated in Figure 5.

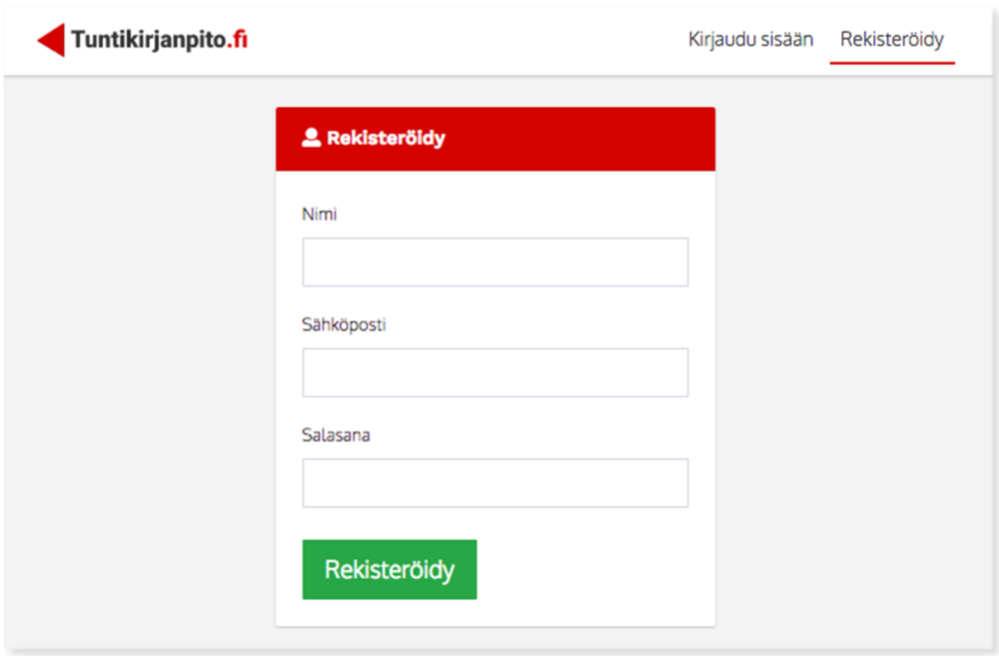


Figure 5. The signup form.

To provide the signup and authentication functionality for the users, the authentication library Passport.js needs to be initialized. This was achieved by creating a passport.js file – allowing it to be utilized later in other parts of the program to verify and store the user sessions.

Code snippet 8. The Passport.js file up to the signup logic.

```
var LocalStrategy = require('passport-local').Strategy
var User = require('../models/user')
..
 passport.use('local-signup', new LocalStrategy({
   usernameField: 'email',
   passReqToCallback: true
 },
 (req, email, password, done) => {
   process.nextTick(function () {
    User.findOne({ 'local.email': email }, function (err, user) {
      if (err) return done(err)
      if (user) return done(null, false,
req.flash('signupMessage', 'Tämä sähköpostiosoite on jo käytössä.'))
        else {
         var newUser = new User()

          newUser.local.name = req.body.name
27          newUser.local.email = email
28          newUser.local.password = newUser.generateHash(password)
29
30          newUser.save(err => {
31            if (err) throw err
32
33            return done(null, newUser)
34          })
35        }
36      })
37    })
38 }))
```

The passport file begins with the necessary imports: the library itself is imported at line 1, and the User database model is imported after that. The signup logic starts at line 14: the server grabs the input from the signup form and queries the database with these values at line 20. If an user does not yet exist with that email, a new user is created – lines 24 to 28 – and saved – lines 30 to 34 – with the parameters from the form.

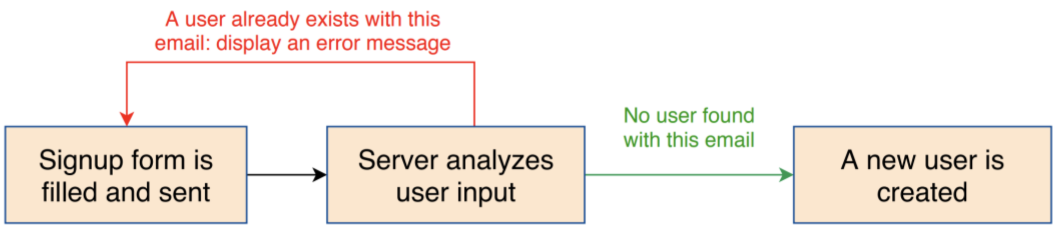
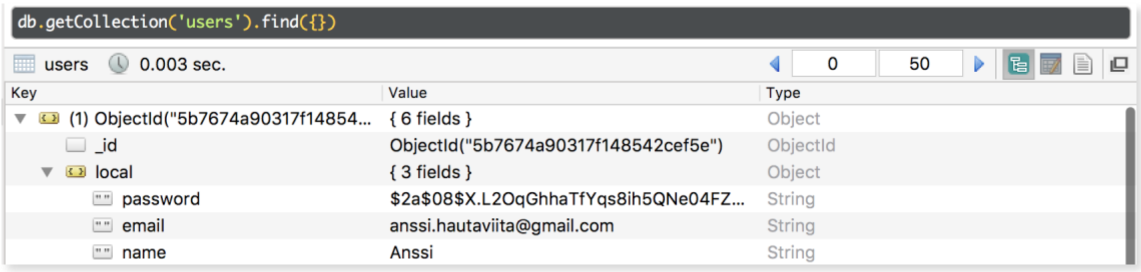

A user already exists with this email: display an error message
Signup form is filled and sent
Server analyzes user input
No user found with this email
A new user is created

Figure 6. Signup process flow.

After this, the User entry is created and therefore visible in the database, as seen in Figure 7. Note that the password is hashed with the function created in Code snippet 7, lines 11 to 13 (userSchema.methods.generateHash). MongoDB automatically creates a unique identifier for each object in the database – unless it is instructed not to – which can be seen here as an _id row in the User object.



```
db.getCollection('users').find({})
```

users 0.003 sec. 0 50

Key	Value	Type
(1) ObjectId("5b7674a90317f14854...	{ 6 fields }	Object
_id	ObjectId("5b7674a90317f148542cef5e")	ObjectId
local	{ 3 fields }	Object
password	$2a$08$X.L2OqGhhaTfYqs8ih5QNe04FZ...	String
email	anssi.hautaviita@gmail.com	String
name	Anssi	String

Figure 7. User entry in the database.

Login

The login page is very similar to the simple signup page, since all of the necessary user input can be inquired via just a couple of form fields (Figure 8).

To provide login functionality to the users, the passport.js file needed to be modified to support login inquiries in addition to the signup calls, demonstrated in Code snippet 9.

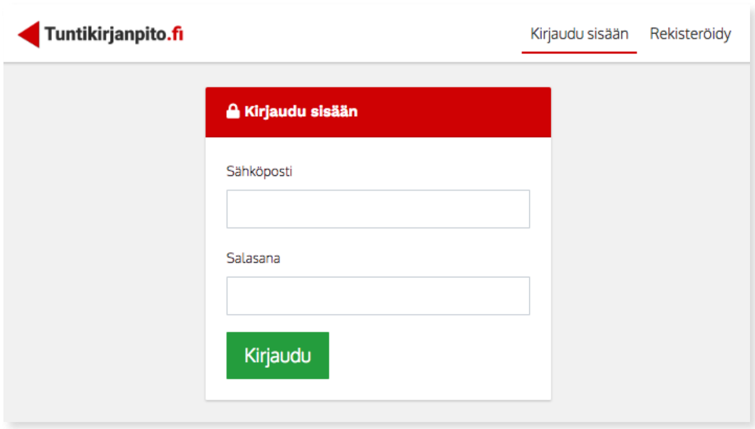


Figure 8. The login form.

Code snippet 9. The login logic of the Passport.js file.

```
passport.use('local-login', new LocalStrategy({
  passwordField: 'password',
  usernameField: 'email',
  passReqToCallback: true
}, (req, email, password, done) => {
  User.findOne({ 'local.email': email }, (err, user) => {
```

```
45      if (err) { return done(err) }
46      if (!user) { return done(null, false,
req.flash('loginMessage', 'Käyttäjää ei löytynyt.')) }
47      if (!user.validPassword(password)) { return done(null, false,
req.flash('loginMessage', 'Väärä salasana.')) }
48
49      return done(null, user)
50    })
51  }))
```

The server grabs the input from the login form and tries to query the database with the email at line 44. If a user with this email is found, the user-submitted password is compared to the password stored in the database at lines 47 to 49. If the password is correct, the user is forwarded to the application and the user details are added to the browser session. If the password is incorrect or some other error occurs, the user is redirected back to the login form.

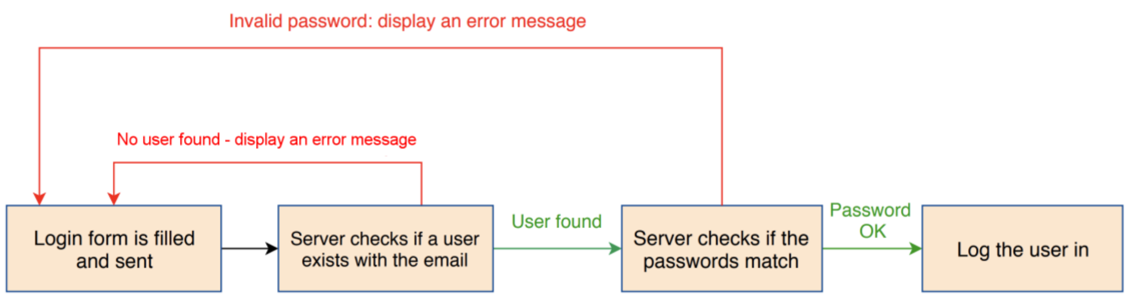

Invalid password: display an error message
No user found - display an error message
Login form is filled and sent
Server checks if a user exists with the email
User found
Server checks if the passwords match
Password OK
Log the user in

Figure 9. Login process flow.

4.2.3 Authenticating requests

After the user has logged in, they are able to perform actions in the Calendar view and the Reports view. However, we have to be able to authenticate each action in these views, because we don’t want anybody to have unauthorized access to the web application.

This was achieved by defining a isLoggedIn middleware that can be used to verify each request before letting them continue. This middleware, shown in Code snippet 10, is utilized in the routes.js file that contains all of the application URL routes. An example of this behavior is shown in Code snippet 11.

Code snippet 10. The isLoggedIn middleware for authenticating requests.

```
function isLoggedIn (req, res, next) {
  if (req.isAuthenticated()) {
    return next()
  }

  res.redirect('/login')
}
```

Code snippet 11. The isLoggedIn middleware in action.

```
app.get('/ ', isLoggedIn, (req, res) => {
  res.render('index.ejs')
})
```

The middleware employs the built-in utility function isAuthenticated of Passport.js to verify the request before letting it continue. If the request doesn't have a proper authentication cookie set, the user is redirected to the login page.

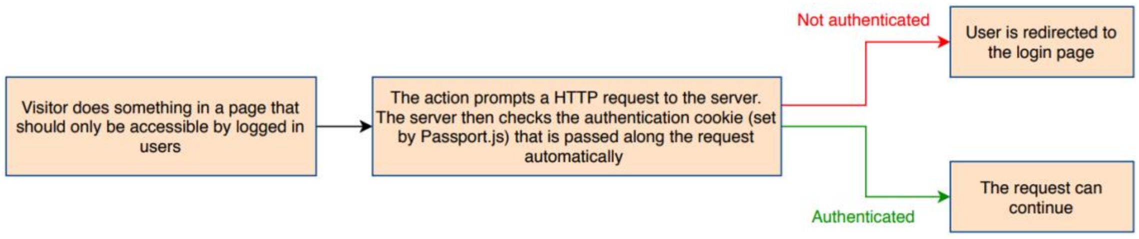

Visitor does something in a page that should only be accessible by logged in users
The action prompts a HTTP request to the server. The server then checks the authentication cookie (set by Passport.js) that is passed along the request automatically
Not authenticated
User is redirected to the login page
Authenticated
The request can continue

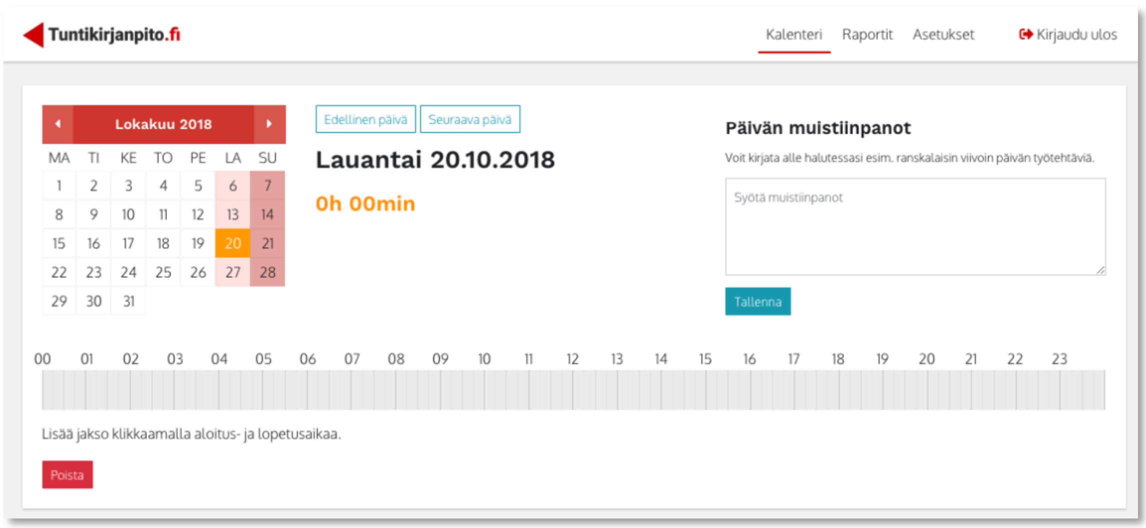
Figure 10. Request authentication logic.

4.3 Inserting daily entries

The main function of this web application, the ability to log working hours to the database, is accomplished with a Vue component ("the Calendar view", see Figure 11) that has the following parts:

- A datepicker field to allow selecting the desired date
- A "date stats" box that displays the selected date and the logged hours for that date
- A field for daily notes

- A field (the “time field”) that contains 96 clickable columns that represent 15 minute time periods, totaling as 24 hours



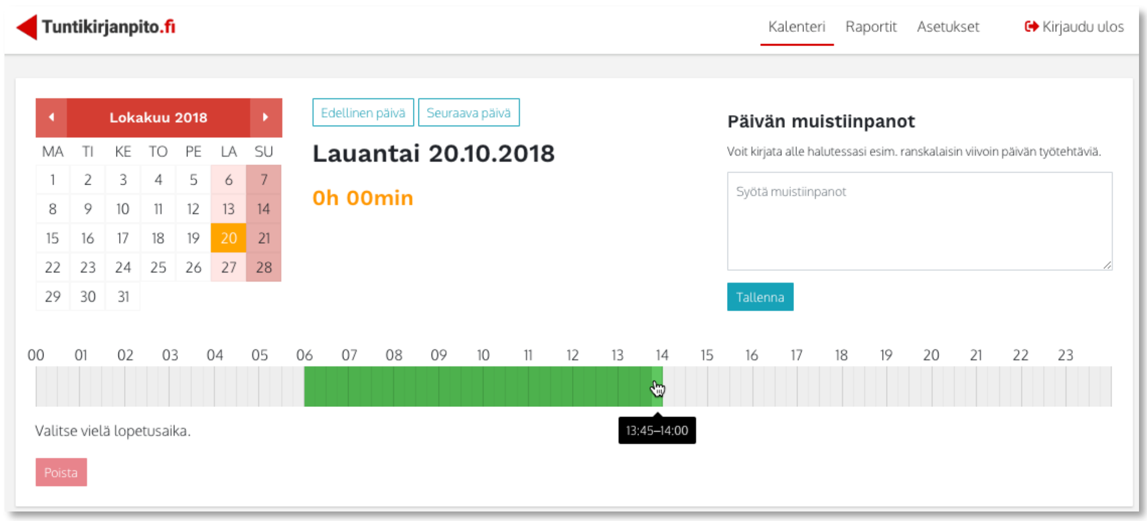
Voit kirjata alle halutessasi esim. ranskalaisin viivoin päivän työtehtäviä.

Lisää jakso klikkaamalla aloitus- ja lopetusaikaa.

Figure 11. The Calendar view.

The main user interaction logic is the following:

1. **Select a date** – or use the current date, which is loaded by default. The date can be selected by clicking the calendar in the top left-hand part of the component.
2. **Click the starting and ending period of the desired entry**. This action is illustrated in Figure 12 – the user has clicked the starting period of 6:00 and is hovering on the ending period of 14:00. After clicking the ending period, the component will update to show “8h 00min” in the middle display area, while a database call is made to update the corresponding records for this user.
3. Optional: **Write notes about the selected date** in the textarea, located in the top right-hand part of the component.



Voit kirjata alle halutessasi esim. ranskalaisin viivoin päivän työtehtäviä.

Valitse vielä lopetusaika.

Figure 12. Inserting an entry.

Since Vue allows the developer to break their code into components and subcomponents in a modular fashion, the time field – that is constructed of 96 columns that each represent a 15 minute slice of a day – was broken into subcomponents (the "Quarter" subcomponents). Each Quarter is a child of the Calendar parent component, and has the following attributes:

- **Hovered-in:** triggered when the user hovers over the Quarter with their mouse
- **Hovered-out**: triggered when the users' mouse leaves the Quarter
- **Clicked**: triggered when the user clicks the Quarter

When any of these triggers fire, the Quarter subcomponent informs the Calendar parent component about the action. The Calendar parent component then processes this information and proceeds with the correct actions – updating the UI, making database queries, etc. An example of this queue is presented in Figure 13.

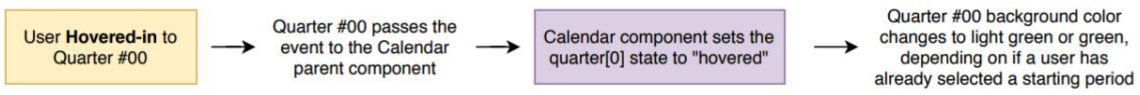

User **Hovered-in** to Quarter #00
Quarter #00 passes the event to the Calendar parent component
Calendar component sets the quarter[0] state to "hovered"
Quarter #00 background color changes to light green or green, depending on if a user has already selected a starting period

Figure 13. The under-the-hood actions when a user hovers over a Quarter.

5 CONCLUSION

JavaScript has established itself as one of the most popular programming languages in the world with a growing ecosystem for a ever-growing range of use cases and a rising population of developers utilizing it in their projects.

This thesis described the building blocks of a full stack web application on the JavaScript-based Node.js platform, utilizing Express.js as the web server, Vue.js as the frontend framework and MongoDB as the database engine – also known as the MEVN stack.

These technologies were applied while building a time-tracking web application aimed for small businesses and individual users. The end product of this software project is a working web application that can be deployed for public use. However, it should probably be thought more as of a prototype for testing the concept of a simple working hours tracking service instead of a full-fledged piece of software, ready for commercial launch.

The application can now be used as a basis for future development and brainstorming. Future development paths could, for example, include refining the code base by including more automated tests, or developing additional features for the end users. In the long term, the application could be launched as a subscription-based service and marketed to small and middle-sized businesses in Finland, and possibly also to companies and private users outside the country.

REFERENCES

[1] Peyrott, S. 2017. *A Brief History of JavaScript*. Available at https://auth0.com/blog/a-brief-history-of-javascript/, accessed May 6, 2018

[2] W3Schools. *Node.js Introduction*. Available at https://www.w3schools.com/nodejs/nodejs_intro.asp, accessed May 6, 2018

[3] W3Techs. 2018. *Usage of server-side programming languages for websites*. Available at https://w3techs.com/technologies/overview/programming_language/all, accessed May 6, 2018

[4] Vuejs.org. 2018. *Vue.js Introduction.* Available at https://vuejs.org/v2/guide/, accessed May 10, 2018

[5] Opensource.com. *What is open source?* Available at https://opensource.com/resources/what-open-source, accessed May 10, 2018

[6] Vuejs.org. 2018. *Reactivity in Depth.* Available at https://vuejs.org/v2/guide/reactivity.html, accessed May 21, 2018

[7] Ibid.

[8] Nodejs.org. 2018. *Node.js.* Available at https://nodejs.org/en/, accessed May 28, 2018

[9] Expressjs.com. 2018. *Express – Node.js web application framework.* Available at https://expressjs.com/, accessed May 30, 2018

[10] Expressjs.com. 2018. *Express middleware.* Available at http://expressjs.com/en/resources/middleware.html, accessed May 30, 2018

[11] Mongodb.com. 2018. *What is MongoDB?* Available at https://www.mongodb.com/what-is-mongodb, accessed May 30, 2018

[12] Techtarget.com. *What is MongoDB?* Available at https://searchdatamanagement.techtarget.com/definition/MongoDB, accessed May 30, 2018

[13] Mongoosejs.com. *Mongoose ODM v5.1.3.* Available at http://mongoosejs.com/, accessed May 30, 2018

[14] Getbootstrap.com. 2018. *Grid system.* Available at https://getbootstrap.com/docs/4.0/layout/grid/, accessed June 2, 2018

[15] Techopedia.com. *What is an Integrated Development Environment?* Available at https://www.techopedia.com/definition/26860/integrated-development-environment-ide, accessed June 6, 2018

[16] Visualstudio.com. *Visual Studio Code documentation.* Available at https://code.visualstudio.com/docs, accessed June 6, 2018

[17] Visualstudio.com. *Why Visual Studio Code?* Available at https://code.visualstudio.com/docs/editor/whyvscode, accessed June 6, 2018

[18] Electronjs.org. *About Electron.* Available at https://electronjs.org/docs/tutorial/about, accessed June 6, 2018

[19] Github.com/remy/nodemon. 2018. *Nodemon (git repository page)*. Available at https://github.com/remy/nodemon, accessed June 6, 2018

[20] Webpack.js.org. 2018. *Concepts.* Available at https://webpack.js.org/concepts/, accessed June 16, 2018

[21] Eslint.org. *Getting Started with ESLint.* Available at https://eslint.org/docs/user-guide/getting-started, accessed June 16, 2018

[22] Atlassian.com. *What is version control?* Available at https://www.atlassian.com/git/tutorials/what-is-version-control, accessed June 16, 2018

[23] Passportjs.org. *Documentation.* Available at http://www.passportjs.org/docs/, accessed October 6, 2018