

Bachelor's thesis

Bachelor of Engineering in Information and Communications Technology

2020

Sujan Pokharel

DEVELOPING A BACKEND WEB APPLICATION AND DOCKERIZING

- A Note Keeping Application



BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Bachelor of Engineering in Information and Communications Technology

2020 | 44

Sujan Pokharel

DEVELOPING A BACKEND WEB APPLICATION AND DOCKERIZING

- A Note Keeping Application

The main objective of the thesis was to study the backend of web development with NodeJS, MongoDB, Express, containerization technology, develop a prototype based on learning and finally package the developed application with Docker

The theoretical part of the thesis introduces the main technologies for the backend, i.e., Node, Express, MongoDB, Mongoose and the concept of containerization is presented through the study of Docker. Since the backend of MERN/MEAN stack is heavily dependent on the knowledge of JavaScript programming language, a brief history of web development and evolution of JavaScript was carried out. The practical part of the thesis guides through the backend development process of a note-taking application and then Docker was implemented to package the app.

As a result, a fully functional backend was developed and was containerized with Docker. The goal of learning about backend development and containerization was fulfilled along with in-depth knowledge of NodeJS in backend development.

KEYWORDS:

Web development, Backend Web Development, Containerization, Node, MongoDB, Docker

CONTENTS

LIST OF ABBREVIATIONS

1 INTRODUCTION	7
2 BACKGROUND	9
2.1 A brief history of web development	9
2.2 JavaScript	10
3 BACKEND DEVELOPMENT	14
3.1 Node and its Working pattern	14
3.2 Express	16
3.3 MongoDB	17
3.4 Mongoose	17
3.5 Frontend Technology (React)	18
4 CONTANERIZATION WITH DOCKER	20
4.1 Docker containers vs Virtual Machine Architecture	20
4.2 Docker Objects	22
5 PROJECT IMPLEMENTATION	23
5.1 Project Introduction	23
5.2 Development environment	26
5.3 Package Installation and database connection	26
5.4 Models	28
5.5 Controller	30
5.6 Authentication	32
5.7 Testing with Postman	33
5.8 Docker Implementation	35
6 RESULTS AND DISCUSSION	38
7 CONCLUSION	41

FIGURES

Figure 1. ES6 important features.	13
Figure 2. Working of I/O in NodeJS (Ofogbu, 2018).	15
Figure 3. Working of Middleware (Ramirez, 2020).....	16
Figure 4. Virtual Dom vs DOM.	19
Figure 5. Docker Architecture vs Virtual Machine Architecture (Yadav, Garg and Mehara, 2019).	21
Figure 6. Overview of the application.	24
Figure 7. MVC architecture (Syromiatnikov and Weyns, 2014).	25
Figure 8. Notes Model.	29
Figure 9. The structure of the project.	31
Figure 10. Sign-up to the application.	32
Figure 11. Login Testing with Postman.	34
Figure 12. Dockerfile of backend.....	35
Figure 13. Docker-compose.yml file of the project.	37
Figure 14. The result from docker-compose.	39

TABLES

Table 1. List of ECMAScript Edition and published date.....	12
---	----

LIST OF ABBREVIATIONS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CI/CD	Continuous Integration/ Continuous Development
CLI	Command Line Interface
CPU	Central Processing Unit
CRUD	Create Read Update Delete
CSS	Cascading Style Sheets
DevOps	Development and Operations
DOM	Document Object Model
ECMA	European Computer Manufacturer's Association
ES6	ECMAScript 6
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IT	Information Technology
JS	JavaScript
JSON	JavaScript Object Notation
JWT	Json Web Token
MEAN	MongoDB Express Angular NodeJS
MERN	MongoDB Express React NodeJS
MVC	Model View Controller
NPM	Node Package Manager
OS	Operating System
SPA	Single Page Application
UI	User Interface
URL	Uniform Resource Locator
VCS	Version Control System

WWW	World Wide Web
XHR	XMLHttpRequest
XML	Extensible Markup Language
YML	YAML Ain't Markup Language
ZSH	Z Shell

1 INTRODUCTION

With the advancement in all technological fields, web development has also seen tremendous changes in recent years. The changes have occurred both in the structure of writing a web program, the way web browser behaves and how the program is packaged before deployment on the server. In the past, web applications were developed with HTML, CSS, and JavaScript. When entering the page, the browser fetched the HTML document in the beginning and upon receiving the HTML page, it would again fetch the CSS and JavaScript file from the server. Modern web applications are Single-page application (SPA) and they do not fetch all the pages separately but instead a single HTML page from the server (fullstackopen, 2020).

This thesis focuses on creating a modern-day backend of a website with the use of practices that is prevalent in modern web applications. Backend is a part of a web application that handles every logic behind the application, database interaction, user authentication, server configuration and so on. Despite having the availability of many frameworks and libraries to create a web app, this thesis is only concerned with the research and development of backend development of MERN (MongoDB Express React, and NodeJS) Stack or MEAN (MongoDB, Express Angular and NodeJS). In other words, the study will be made on NodeJS, Express, different packages, and MongoDB and backend prototype will be developed so that it can be either rendered using React or Angular. The reason behind choosing this stack is the popularity of NodeJS to create Restful APIs. The front-end development is beyond the scope of this thesis but instead, the thesis explains the containerization technology with the use of Docker to wrap the application so that application is ready for DevOps and deployment. DevOps is a set of practices that works to automate and integrate the process between software development and IT teams to build, test, and release software faster and more reliability.

The thesis will commence with an overview of the history of web development, providing brief insight of past web. After this has been achieved, the thesis will elaborate on the purpose and motivation behind the topic and later will go through the environment setup and focus on the technologies such as JavaScript, NodeJS, Express, MongoDB which are used in backend development and then about containerization with Docker.

A prototype will be built using the above-mentioned technologies and a detailed explanation of building a backend and packaging the application with Docker will be

carried out. This will provide a solid base for understanding the development of backend and containerization technology. The backend can be used to make a full-stack development with either MERN or MEAN stack if React and Angular are implemented respectively. Since this thesis is also a research study on Node and its framework that makes backend of MERN/MEAN stack, the reasons for the growing popularity of Node are also presented.

After the discussion of different technologies for backend development and implementation of Docker, the results chapter will then evaluate whether the project achieved its goal and analyze the used technologies.

2 BACKGROUND

2.1 A brief history of web development

After the invention of the World Wide Web (WWW) by Tim Berners-Lee in 1989, the WWW has come a long way. Originally, the main purpose behind the development of the web was “to meet the demand for automated information-sharing between scientists in universities and institutes around the world” (Runestone Academy, 2020). It was publicly accessible in 1991 and is accessible at its original URL. Information was shared between computers over the network with the text documents in the form of static HTML pages (Runestone Academy, 2020).

The web’s popularity grew quickly, and later it became possible to add images, audio files and videos to the web pages but still were static. There was the intention to make HTML pages dynamic, so it resulted in the development of client-side programming language; JavaScript in 1995 (Runestone Academy, 2020).

However, a real shift from static to dynamic web pages did not take place until the introduction of AJAX (Asynchronous JavaScript and XML) in 2005. It was the new approach that enabled who? What? to use responsive web design app. It meant a client-side web application could be built entirely in JavaScript. Web sites were becoming like desktop applications and the browser became the platform for an application.

JavaScript was immature at this point, thus there was a need for modern tools and libraries (Runestone Academy, 2020). So, different libraries and frameworks were introduced. There was the improvement in hardware and networks were solid with greater bandwidth and this also paved the way for constant improvement of the frameworks.

The need for the web application also grew tremendously as web applications are easy to access from any operating system with the help of the browser. Desktop applications, on the other hand, required developers to develop an application for each operating system as one application will not be compatible with both.

2.2 JavaScript

The thesis is primarily focused on the background of the web and project introduction until this point. To proceed further into the development of backend, it is important to introduce the JavaScript, as NodeJS is dependent on knowledge of JS. Thus, this section will introduce JavaScript and its different versions.

Introduction to JavaScript

JavaScript or in shortened form JS is a lightweight, object-oriented language with first-class functions based on objects that can return a value by passing a function itself or other functions as an argument. It is also known as the scripting language and can be used in many non-browser environments. JavaScript was first developed in 1995 by Brendan Eich in a company named Netscape. It was developed to serve as a scripting language for Java (MDN Web Docs, 2020). Although JavaScript was primarily known to be used for client-side of the web, it has evolved significantly to support the server-side scripting as well. JavaScript now is viewed as a programming language that is used for front-end and back-end of web applications, and databases.

Since JavaScript supports both front and backend development, it has several advantages over using different languages in frontend and backend. With the knowledge of a single language, one can be established as a full stack developer. It is a good stack for developing dynamic and high performing applications. Code sharing and reusability help to lessen the number of lines of code. Moreover, using a common language helps in better team efficiency, since all the team member is working on the application. Besides, developers do not need to worry about the syntactical differences as the same language is being used for the application. (MDN Web Docs. 2020).

According to the report (PayPal Engineering, 2020), PayPal after moving from JAVA language to NodeJS for their server-side application found a tremendous increase in performance with NodeJS. It claimed that (PayPal Engineering, 2020), the development of an application with JavaScript required less than half of the time as compared with Java and had fewer people, 33% fewer lines of code and 40% fewer files". Similarly, in terms of performance, "NodeJS application had double the requests per second and there was a 35% decrease in the average response time" (PayPal Engineering, 2020).

JavaScript Versions

Ever since, JavaScript was standardized by Netscape in 1996 with ECMA (Ecma International – European Association for Standardizing Information and Communication Systems), JavaScript has gone through major changes in the syntax of code. ECMA sets the standards and works to make JavaScript modern and relevant to the current technological standard. Different versions were being released in space of considerable time until 2015, but they decided to release new version afterwards each year. Thus, ECMA was changed to ECMAScript (ES6) but popularly it is known as ES6.

ES6 was released to support writing complex applications, libraries as well as code generators. More importantly, its main purpose was to keep the versioning system simple and better. As a result, the new version introduced several new features such as modules, classes, arrow functions and ES6 proxies to name a few (Rauschmayer, 2014)

Although ES6 was released in 2015, most of the browsers did not support new features. Therefore, Babel and Google Traceur were used as transpilers to convert the JavaScript code written in ES6 to ES5. Later, all the modern browsers supported new ES6 features and the use of transpilers is obsolete. Table 1 shows the list of different ECMAScript editions, their official name and release date.

Table 1. List of ECMAScript Edition and published date.

Edition	Official name	Published Date
ES10	ES2019	June 2019
ES9	ES2018	June 2018
Es8	ES2017	June 2017
ES7	ES2016	June 2016
ES6	ES2015	June 2015
ES5.1	ES5.1	June 2011
ES5	ES5	December 2009
ES4	ES4	Abandoned
ES3	ES3	December 1999
ES2	ES2	June 1998
ES1	ES1	June 1997

This detailed explanation of each edition is beyond the scope of this thesis but some common features that were introduced in ES6 are shown in Figure 1. Since this thesis is about the development of backend, an explanation of promise and async function is provided.

Promises

Promises were introduced in ES6. Promises make work slightly easier when it comes to writing complicated asynchronous functions. A promise is an object with “then” and “catch” methods. Either of the above methods is called when the promise returns either a value or error but never both. A promise object is created from promise constructor/class and needs a callback function. This callback function receives the “resolve” and “reject” function parameters. These methods are chainable as shown in Figure 1 under Promises.

Async/Await

ES7 introduced a new way to add async behaviour in JavaScript thereby making it easier for developers to work with Promise. With the introduction of the `async` and `await` keywords, the `async` keyword can be used before a function declaration. This keyword makes it asynchronous, which means that when the function is called, a promise is returned, and normal code execution will commence as usual. Similarly, the `await` keyword can be used inside the same function which blocks the execution of JavaScript in that context until the promise it is awaiting is settled. This gives a cleaner syntax to work with promises in asynchronous function.

```

4
5 // Default parameter in Es6
6 var person = function(name = 'sujan', num = 134, url = 'http://sujan134.com') {
7   }
8
9 //Template Literals
10 var name = `Your name is ${first} ${last}.`
11 var url = `http://localhost:3000/api/contact/${id}`
12
13
14 //Destructuring
15 [a, b] = [10, 20];
16 console.log(a); // expected output: 10
17 > console.log(b); // expected output: 20...
18
19 [a, b, ...rest] = [10, 20, 30, 40, 50];
20 console.log(rest); // expected output: Array [30,40,50]
21
22
23 //Arrow Functions
24 var UpperCase = function() {
25   this.string = this.string.toUpperCase()
26   return () => console.log(this.string)
27 }
28 UpperCase.call({ string: 'well that is interesting' })()
29
30
31 //Promises
32 var wait1000 = () => new Promise((resolve, reject) => {setTimeout(resolve, 1000)})
33
34 wait1000()
35   .then(function() {
36     console.log('Yay!')
37     return wait1000()
38   })
39   .then(function() {
40     console.log('Wheeyee!')
41   })
42
43
44 //Let Vs Const
45 const name = 'sujan'
46 let finalPoints = 50;
47 let winner = false;
48 if(finalPoints > 40) {
49   let winner = true
50 }
51

```

Figure 1. ES6 important features.

3 BACKEND DEVELOPMENT

Until now, the background for the web, purpose for the thesis and JavaScript and its different version was discussed. In this section, the focus will be on the introduction of the technologies that are going to be used in the project. Thus, the section will explore on NodeJS, Express, MongoDB, Mongoose and a brief introduction on React as boilerplate was used during Docker use. Even though the focus on frontend development is beyond the scope, it is necessary to introduce the brief concept as it is vital for full-stack development.

3.1 Node and its Working pattern

JavaScript primarily had been a language for browsers until Ryan Dahl created NodeJS in 2009. It is a JavaScript runtime built on Google Chrome's V8 JavaScript engine (The Odin Project, 2020). This approach allowed JavaScript to run on a server. Suddenly, developers did not require to have additional language such as PHP or Python to write server-side code. Instead, they could focus on JavaScript.

NodeJS is a non-blocking, asynchronous and event driven I/O system. Asynchronous means that sequence or order in which the program runs is not put into consideration. Instead, functions that were written will get called when there is an occurrence of events like network request (event-driven) (The Odin Project, 2020). If the code is processed synchronously, then there the steps in which the code is written will matter. When a function is called, the program will wait until the function returns before moving into the next step. This process is considerably slow.

In NodeJS, when a file system is reading the file, it uses the idle time to handle another request. When there is the completion of a file system, it tells NodeJS to take the resources and send to the browser. Event Loop is a program that waits for events and dispatches upon receiving. "Node uses an event loop that facilitates non-blocking I/O combined with event-driven I/O, a scheme where a registered event callback function is invoked when some action happens in the program "(Chettri, 2016).

It is also important to note that there are two types of operations. I/O operations like read/write of file, network as explained above and other CPU intensive operations like

editing an image, compressing and decompressing etc. Latter tasks require a lot of time and would have a negative effect if the process runs on the main thread. Thus, such operations should run on the background thread and upon completion, the result should be sent to the main thread.

Thus, there is a need for “two different thread pools for the background work that the application will do” (Hellman, 2020). As most of the operations are I/O, it should have a larger pool, but threads for CPU-only operations also should be considered (Hellman, 2020)

Figure 2 illustrates the general working process of NodeJS. NodeJS creates an event loop to execute all the requests that comes to the server. As a result, it increases the speed and performance of an application.

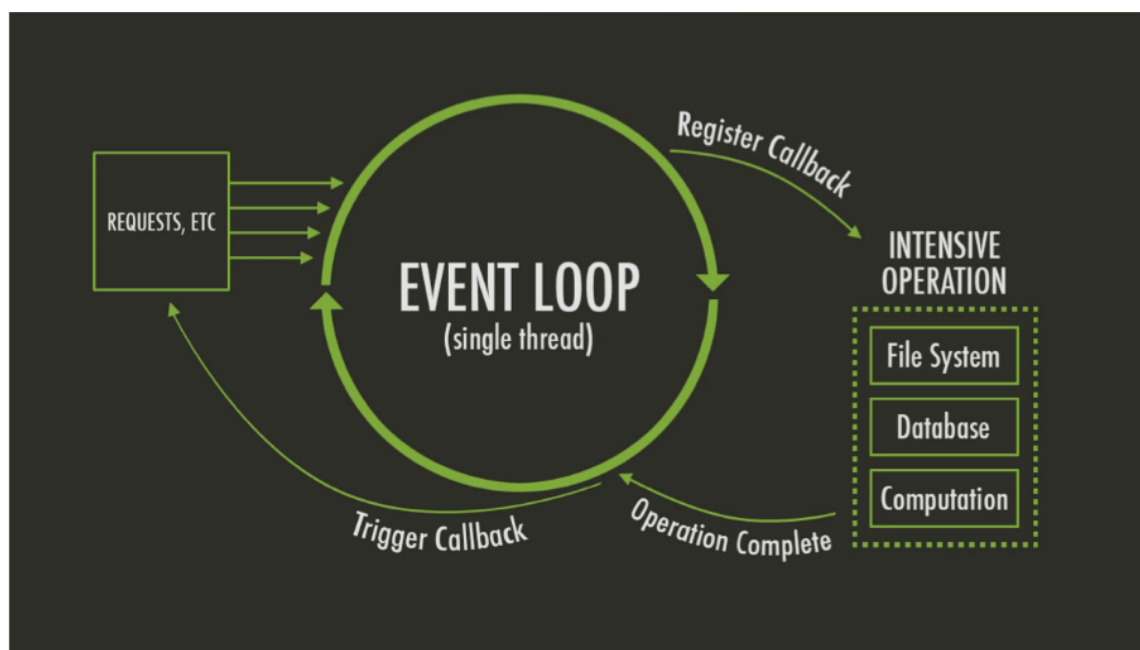


Figure 2. Working of I/O in NodeJS (Ofogbu, 2018).

NodeJS's performance and stability are constantly being improved regularly. The new JavaScript features that are introduced in the yearly release of ECMAScript are integrated into NodeJS in its new releases.

3.2 Express

Express is a web framework written in JavaScript and hosted within NodeJS runtime environment. It is also the most popular NodeJS web framework and provides useful features like writing handlers for requests with different HTTP verbs at different URL routes, adding middleware etc. along with the core node module. With Express, one can speed up the web development process with ease (Express Documentation, 2020).

“Express, is deliberately a very lightweight web application framework, so the main benefit comes from the use of third-party libraries and features” (Express Documentation, 2020). With Express, use of middlewares is possible. Middleware is/are functions that are invoked by the Express routing layer before final request handler is made. Middlewares have access to the request, the response object along with the next function (Express Documentation, 2020). The middleware function that is loaded first will get executed first as shown in Figure 3.

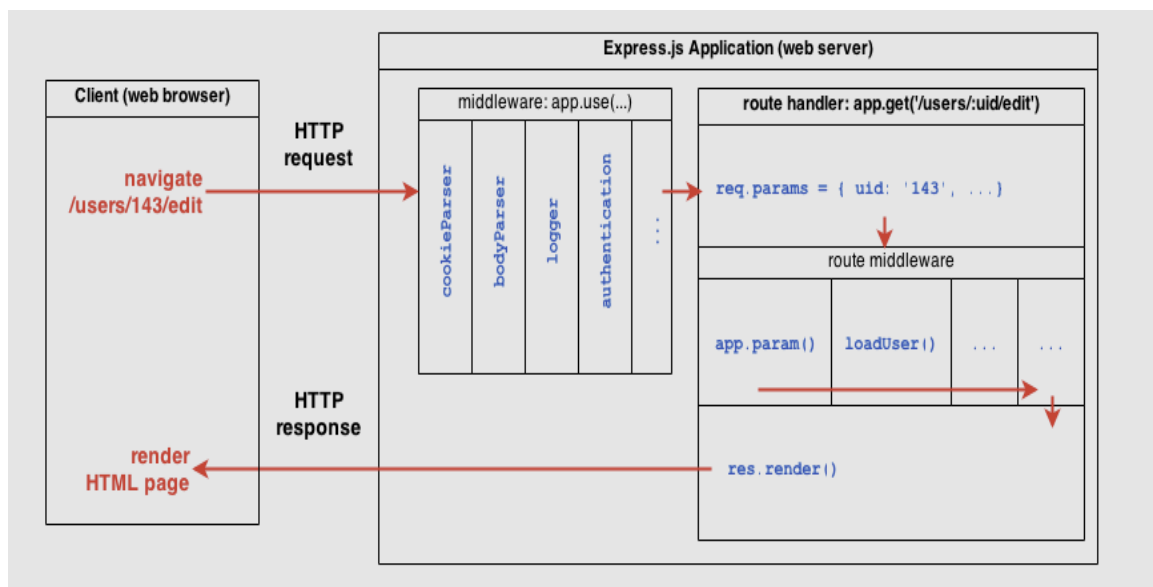


Figure 3. Working of Middleware (Ramirez, 2020).

The middleware takes the request object, executes the code, changes the request as well as response objects and calls the next function. This activates the next middleware if there is any in the queue.

3.3 MongoDB

MongoDB is a document-based database that is designed to make development easier and scalable. Unlike NoSQL which uses traditional “relational” data storing methods, MongoDB stores the data more flexibly as there is no format or structure like in SQL database. In this method, BSON format is used to store the data. This format is also called binary JSON. The document contains different data types like strings, numbers, floats, arrays and objects. MongoDB supports all the CRUD operations like querying, inserting, editing and deleting in the database (Subramanian, 2018).

MongoDB supports storing of dynamic data. Unlike the compulsion for storing similarly structured data in the same group in the SQL database with key-values, the documents can be under same collection. Such an approach provides more flexibility to store non-alike data. Moreover, there is improvement in speed of the database operations. There are also unnecessary hassles to connect different tables like in traditional database (Subramanian, 2018).

3.4 Mongoose

Mongoose is an Object Document Mapper (ODM) meaning that it is possible to define objects with a schema that is mapped against a MongoDB document (Munro, 2020). The data needs to be structured properly thus; Mongoose helps in organizing the data. With the help of Mongoose, schemas can be defined with strongly typed data. With the schemes, models can be created based on the data. It can validate the data. It has also the ability to allow only valid data types to be saved in the database. Later, Mongoose Model is mapped to MongoDB Document in reference to Model’s schema definition. Furthermore, features provided in MongoDB are enhanced with additional features to make the querying easier into the database. Most importantly, connection with the database can be done with the server and along with it can perform similar database CRUD operations (Munro, 2020).

3.5 Frontend Technology (React)

React is one of the modern web libraries for front-end application built by Facebook. It is a JavaScript library used for DOM manipulation to handle navigation through HTML5 push state. In other words, it is a view library that uses components to change contents on the page without refreshing, which is the core principle behind single-page applications. It is used along with react-router to create single-page web applications. This makes websites run fast as a user does not have to wait for different pages to load. It also reduces strain on the server.

The most interesting feature of React is its components. The main ambition behind its development was to make it modular so that it can be reused into different parts of the application.

React uses a syntax called JSX, which is like HTML but uses JavaScript. JSX allows specifying the DOM elements before the components are inside of JavaScript files. This makes logic that is behind the component and visuals all in one place.

The main performance for React comes by utilizing virtual DOM. Unlike other frameworks or libraries, it directly does not operate on browser's DOM. Rather than manipulating the document in the browser, which is quite slow, it uses virtual DOM to compare it with real DOM to tell which elements need to be updated and which are not to be. The Virtual DOM exists entirely in-memory.

Components

React allows users to split any page of the application into independent, reusable pieces called React Components. React components takes in the data and return the desired output through the render method. Each class-based component has several lifecycle methods while functional component does not have a lifecycle method, but later Hooks have been introduced and can function like a class-based component.

The state in the class-based component is a JavaScript object used for recording and reacting to user events. Whenever a component state is changed, the component and all its child components re-renders immediately. States hold values throughout the component and is passed down to other components as props when there is need to pass the state.

Virtual Document Object Model

The HTML DOM was mostly used until the development of React. They were originally intended for static pages and was not optimized for creating dynamic UI. Before, when there was an update in any of the DOM node, HTML DOM needed to update every node and re-render the web page with corresponding CSS and layout. It was common for a single web page application to re-render the whole page when event listeners were attached to them. In dynamic web pages, the HTML DOM must check for every change in each node at a regular time. This considerably reduced application performance. In Figure 4, the difference between DOM and Virtual DOM is presented.

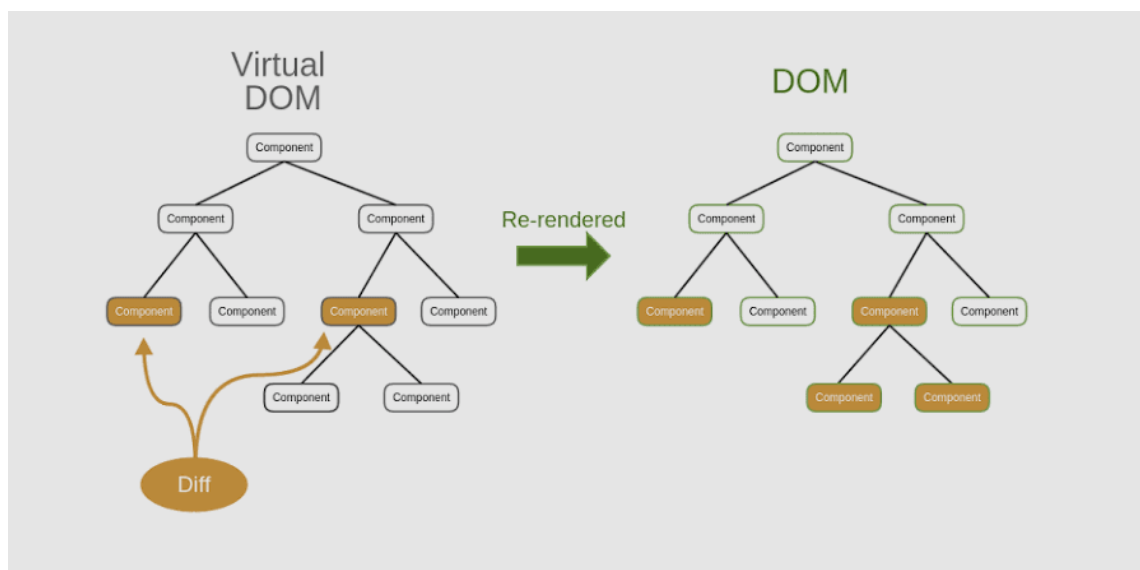


Figure 4. Virtual Dom vs DOM.

The virtual DOM was invented as a solution to this inefficiency. It is lightweight and is detached from the browser. It can be updated without any changes affecting virtual DOM. React has Virtual Dom built-in module called React DOM. When the updates are supplied. React uses a process called reconciliation, using an algorithm that compares changes to know which elements need updating. React then change those elements only without affecting the other elements that don't need change (OnCrawl, 2020).

4 CONTANERIZATION WITH DOCKER

The need for virtualization dates to the 1960s when computers could only perform an individual task due to technical limitations. Bigger projects that required more resource evolved as the result computers needed to be more robust. New hardware and software were requested from different vendors that could support multitasking (Oracle, 2019).

Ever since virtualization has been constantly evolving. The major advantage of virtualization is the isolation at the highest level from the machine that supports the virtualization. This makes it possible to select either the host OS or the virtualized OS. Nevertheless, such implementation has the main drawback regarding the use of computer resource. The amount of resource needed to virtualize is higher for a completely isolated system and does not share anything with the host machine, while containers share the host OS kernel (Docker Documentation, 2020). Virtual machines that are commonly used by individuals and organizations are Oracle's Virtual Box, VMWare's vSphere or Microsoft's Hyper-V (Kleyman, 2012).

A container in Docker is a software that wraps all the code including all the dependencies to make an application to run smoothly from one platform to another (Docker Documentation, 2020). Both containers and virtual machines help to create a self-contained virtual package. Despite being used for a similar purpose, the architecture, advantage and fallbacks of each system are slightly different from each other and explained in Section 6.1

4.1 Docker containers vs Virtual Machine Architecture

From Figure 5, it is seen that the container shares the kernel of the host OS with other containers, and the shared part of the operating system is read-only. Containers share common OS with the help of Docker Daemon which is an engine for Docker. Thus, the containers are lightweight. This enables developers to deploy multiple containers on a single server or virtual machine. There is no need to dedicate the entire server to a single application. As there is only one OS to maintain, scaling becomes easy and fast without the need for more server space.

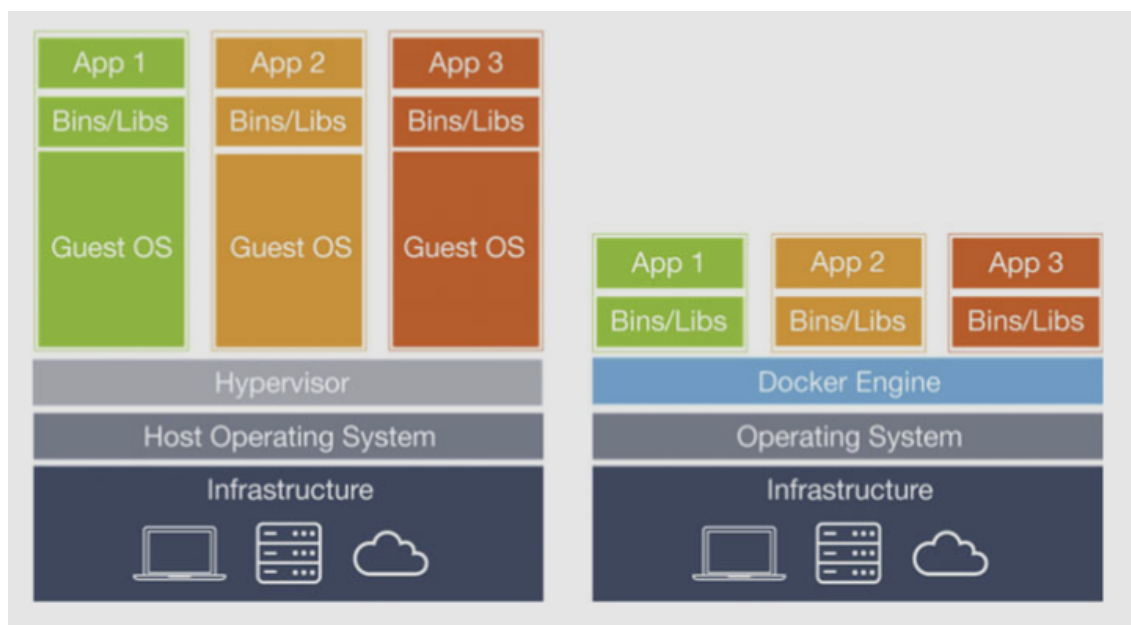


Figure 5. Docker Architecture vs Virtual Machine Architecture (Yadav, Garg and Mehara, 2019).

However, the container has also a few disadvantages. As container shares a kernel with the container host, security vulnerability in the OS kernel is a threat to all containers on the machine. Besides, Containerization is still a new solution with variation in implementation, so adoption could be a challenge for some. Containers are better when developers have to lower the number of servers for multiple applications.

Virtualization, as in Fig 5, enables running of multiple operating systems a server in total isolation. A hypervisor is used to control different running OS. Each OS needs own dependencies and own CPU and memory resource. The main benefit of using virtual machines is that the virtual machine image looks like a data folder. Each can be copied and moved like usual files. This nature helps teams to centralize workloads and run many different OS without on-premises hardware (Burwood Group, 2019). Updating apps and OS would be easy without affecting the end-user experience.

However, virtual machines are not without their disadvantages. As each VM includes an OS and a virtual copy of all the hardware, they require significant RAM and CPU. Besides, moving VMs between public clouds, private clouds and traditional data centres is also challenging (Burwood Group, 2019)

4.2 Docker Objects

During the implementation phase below listed objects are heavily used. This section provides a brief overview of them.

Image

An Image is a read-only template to build containers with instructions in it. It is based on another image with some customization (Docker Documentation, 2020). Docker images are created by writing the steps using *Dockerfile*. Each step creates the layer in the image. During any rebuild or change, only the layers from where the changes have been made are rebuilt. Thus, when writing Dockerfile, attention must be paid on the sequence of steps. Due to this reason, the images are small, lightweight and fast as compared other virtualization technologies (Docker Documentation, 2020)

Containers

Containers are runnable instances of an image that can be created, started, stopped, moved or deleted by using the Docker API or CLI (Docker Documentation, 2020). Containers are isolated from other containers and the host machine.

Storage

Data can be stored within the writable layer of a container in the form of persistent storage. In this case, Docker provides four options like data volumes, data volume container, storage plugins and directory mounts. Data volumes provide the ability to list and rename volumes and list the container associated with the volume.

In a data volume container, container hosts a volume and mount that to other containers. Since the volume container is independent of the application container, it can be shared among the containers.

Storage plugins connect the external storage platforms by mapping from the host to external sources like storage array or an appliance.

5 PROJECT IMPLEMENTATION

5.1 Project Introduction

The project is chosen out of personal choice and different functionalities that are prevalent in modern-day web application was made sure to be included in the prototype. As the author has some daily writing habit, an application that is capable of keeping notes was thought of and development was carried out. The main problem that the author had was the lack of a fixed system to record the notes. Sometimes, there is the use of a paper form like a diary, but other time uses word document or online form to record notes.

The application facilitates daily writing. Also, writing becomes manageable in one single private application. The project has CRUD (Create Read Update Delete) operation which will help to create, read, update and delete the notes. Besides, the system has the authentication system developed so the owner of notes will be able to perform CRUD operations.

Features and Functionalities

After investigating the problems and doing the requirements analysis for the project, the following requirements were listed for a prototype to be built.

- Users shall be able to create an account.
- Users shall be able to login into the system with their credentials.
- Users shall be able to be greeted with welcome mail upon signing up.
- Users shall be able to add notes in the application.
- Users shall be able to edit their individual note in the application.
- Users shall be able to see all their notes in the application.
- Users shall be able to fetch each note in the application.
- Users shall be able to delete each note in the application.
- Users shall be able to add an image in their profile.
- Users shall be able to delete an image in their profile
- Users shall be able to logout of the application.
- Users shall be able to remove their account from the application.
- Users shall be greeted with goodbye message upon deleting of their account.
- Timestamps for the performed operation shall be noted.

Based on the requirements listed and overview of the application as shown in Figure 6, MVC (Model View Controller) architecture is used to design and develop the backend of the project.

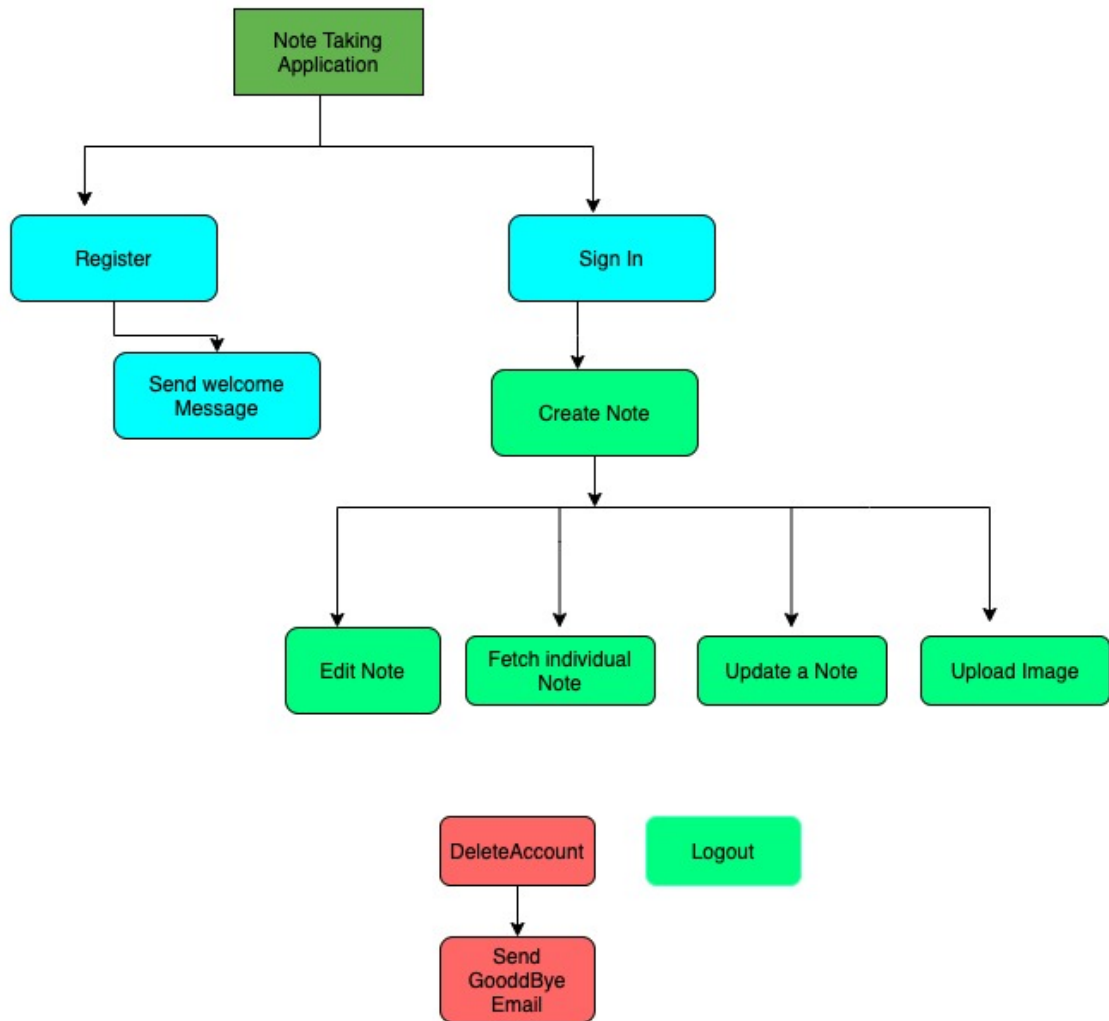


Figure 6. Overview of the application.

Use of MVC architecture is also one of the modern features of the programming paradigm. MVC separates the representation of the application domain (Model) from the display of the application's state (View) and interaction control (Controller) (Syromiatnikov and Weyns, 2014).

Thus, the application is divided into three separate parts, a Model, a Controller and a View which all communicate with each other. Models are an object which contains the

data of the application. They define the data types that live in and receive the data from the controller. They also send the data to the view. The view of an application is the front face of the application where the application user sees and interacts. Users provide the data through the view of the application. Usually, front-end design is the view layer. The controller is the backbone of an application, where all the logic required for the application is kept. CRUD actions such as creating, retrieving, editing and deleting data from models are handled by the controller.

Figure 7 illustrates the scopes of model, view and controller of an MVC application and how they communicate with each other.

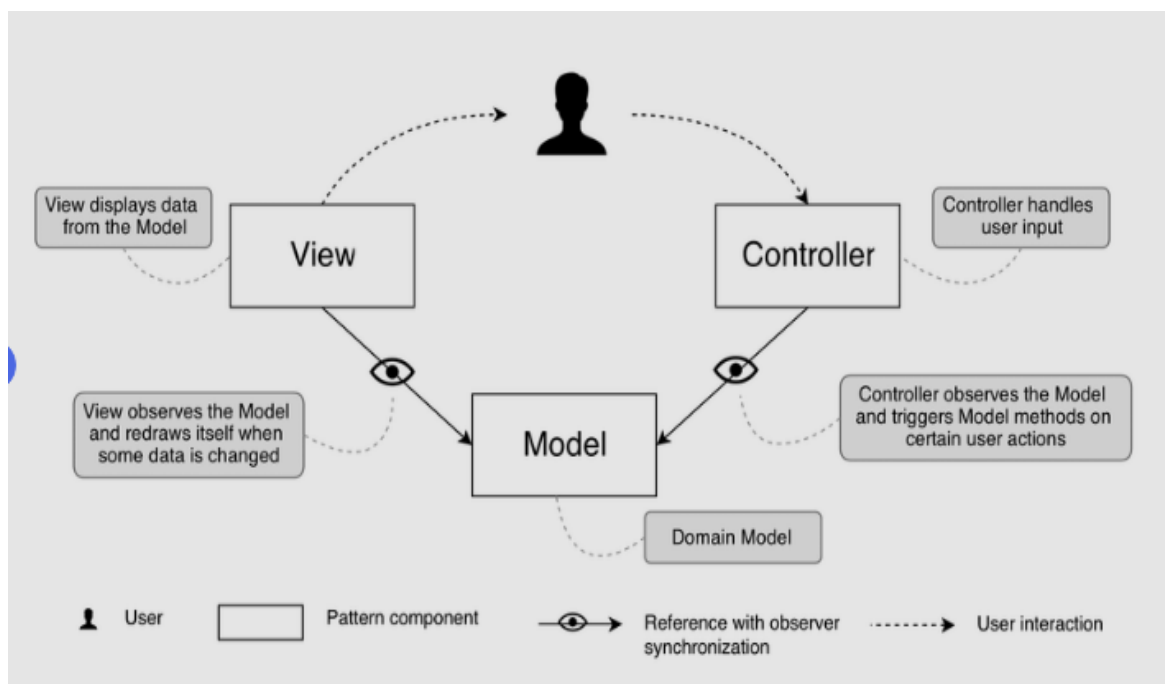


Figure 7. MVC architecture (Syromiatnikov and Weyns, 2014).

As shown, the view displays the data from the Model, observes the Model and redraws itself when some data is changed from the user. The controller handles the user, observes the model and triggers Model methods on user actions. In the end, the view gets the updated data from the model and displays the same data (Syromiatnikov and Weyns, 2014).

5.2 Development environment

Visual Studio Code is a lightweight and powerful source code editor available for Windows, macOS and Linux and comes with built-in support for JavaScript, TypeScript and Node.js (code, 2020). VS Code is a product of Microsoft and was first released in 2015. Since then, it has quickly become leading code editor for JavaScript developer in the market. The version Visual Studio Code used for the development of this application is 1.44.2. Additionally, a package named Node.js and NPM is installed to facilitate the development of Node application.

Git was chosen as the version control system (VCS) which made it easier to track changes made in files. It is a powerful and popular tool for tracking changes. According to a survey conducted by Stack Overflow among the professional software developers in 2018, lion's share 88.4 percentage of respondents used Git as a version control system (Stack Overflow, 2020). Git thinks of its data more like a series of snapshots of a filesystem. With Git, whenever there is commit, or save, Git takes a picture of what all files look like at that particular moment and stores a reference of that snapshot. If files have not been changed, Git doesn't store the file again, just links it to the previous identical file that has already been stored. This way it maintains its efficiency. Git considers its data like a stream of snapshots. (Git, 2020)

The operating system chosen was macOS Mojave 10.14, on a 2017 MacBook Pro 15". This is the primary development machine of the author and it has been heavily customized for the comfort of development. Visual Studio is heavily customized with external extensions for comfort. Unix-based OS is also considered ideal for web development as it offers quick and easy access to the command line and working is similar to the Linux server.

The used terminal was ZSH. It is an extended version of Bourne Shell (sh) with new features. It supports plugins and themes. It has intelligent command completion, spelling correction and improved git integration over alternative software.

5.3 Package Installation and database connection

The development environment was set up with NodeJS installed as explained earlier in the chapter. The tools and nodes packages that are required were installed with Node

Package Manager. (NPM) It is an open-source package manager used for installing JavaScript packages. Node packages that were used were Express, mongoose, Nodemon, SendGrid etc. After installing NodeJS, Express framework was installed with the command `"sudo npm install express"`. Nodemon is used for restarting the application automatically after any changes are made to the server-side code. It was installed with the command `"npm install nodemon -save -dev"`. The use of flag installs the dependencies only in the development environment but not on the production environment.

After installing the required Packages, MongoDB database was setup. The cloud version of MongoDB was used after signing up in the MongoDB website and the free cluster was chosen. The current IP address was added to get access to the and MongoDB driver was installed with command `"npm install mongodb"`

After installing MongoDB, mongoose with installed with the command `"npm install mongoose"` to get the latest version. As stated earlier, mongoose is the query language for MongoDB. The database connection was successful between application and the server with the use of URL that has username and cluster password created during the setup of the cluster. Furthermore, Schemas were created and put inside the model which is discussed in a topic latter.

SendGrid

"SendGrid is a cloud-based SMTP provider that allows to send email without having to maintain email servers" (SendGrid, 2020). SendGrid is used to send email from the application in different cases like, sending a link to the user when resetting a password, purchasing something online, successful registration and so on. Hence, SendGrid is a third-party email server that sends an email securely and reliably.

SendGrid uses two ways to send an email; through SMTP relay or web API. In the project, SendGrid follows a Web API method to send an email. A free account was created on the SendGrid website and the package was installed using the command `'npm install --save SendGrid'`

Multer

Multer is a Middleware for Express and Node.js which is used for handling form data. It adds a body object and a file/files object to the request object. The body object contains the values of the text fields of the form and the file/files object contains the files uploaded through the form (Bcrypt, 2020). The purpose of use of Multer in the project is to help uploading of files into the application. It was installed using the command *“npm install --save Multer”*

Bcrypt

Bcrypt is an encryption library used to hash the password. The password should not be put in a plain text in the database instead it should be hashed and salted. This package was installed into the application using *“npm install bcrypt”*

JSON Web Token

“JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securing transmitting information between parties as a JSON object” (JWT, 2020). It is used in the Authorization of a user during the login. When the user is logged in, each request from the user will include JWT, so that the user can access routes, services and resources. It contains header, payload and signature. It was installed with the command *“npm install jsonwebtoken”*

5.4 Models

Models are like a blueprint for the data types that remain in the system registers in the database. Models were created to define a standard structure for the documents. The application consists of two models namely user Model and Note Model. A schema for the notes object is created in the note model and is exported to be used in other files of applications as shown in Figure 8.

```
const mongoose = require('mongoose')

const noteSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
    trim: true
  },
  description: {
    type: String,
    default: "On this day I did nothing"
  },
  myday: {
    type: Buffer
  },
  owner: {
    type: mongoose.Schema.Types.ObjectId,
    required: true,
    ref: 'User'
  }
}, {
  timestamps: true
})

const Note = mongoose.model('Note', noteSchema)

module.exports = Note
```

Figure 8. Notes Model.

The model only accepts properties that are defined in Schema as shown in figure 8. For instance, the title property is required and must be a string. The trim property is used to sanitize the data that comes in from the user. If the user types “ user ” taking the unnecessary spaces, it trims the unnecessary space. Similarly, the note model also contains the title, description, owner and “myday” property. The owner object stores the reference of the user who created the note. The person who logs in into the system and creates notes is the owner. ‘timestamps’ property is used to record the time of the creation of post and its modification. Finally, the Note model is exported so that this module can be used in other parts of the application. Making such files help to reduce the code so this is also one practice of modern web programming.

Filtering and Sanitizing

The data that the user sends to the server should always be checked. The users that use application could be vulnerable to harming the application by sending the scrips or by trying to steal confidential information from the server. So, any data that a user sends are a security threat, therefore, data needs to be validated and sanitized. Validation can be implemented in both the client-side and server-side, but client-side validation does not prevent the application from attacks thus, server-side is validation is carried out when dealing with user data. To carry out this task, Validator was chosen.

Validator is a third-party package that validates and sanitizes the data which are sent by the user. It is installed using the command `'npm install validator'`. The incoming data is checked in each of the routes and if there are any errors, they are sent back to the user to correct and submit the data again. Unnecessary spaces are automatically removed by the validator. Likewise, the input is sanitized with the validator before keeping

5.5 Controller

The controller has all the server-side logic of the application where the REST API is implemented. The `'users.js'` file contains the logic and the routes for all the user-related activities such as displaying the register and login page, registering a user, uploading the profile image into to the database and checking for the credentials before logging into the system. It also has all the CRUD routes for updating credentials, logging out or deleting the account. The `"note.js"` file has all the logics that are related to updating, creating, reading and deleting the note by the owner. The main logic in controller involves the use of the database. Figure 9 shows the folder structure of the project.

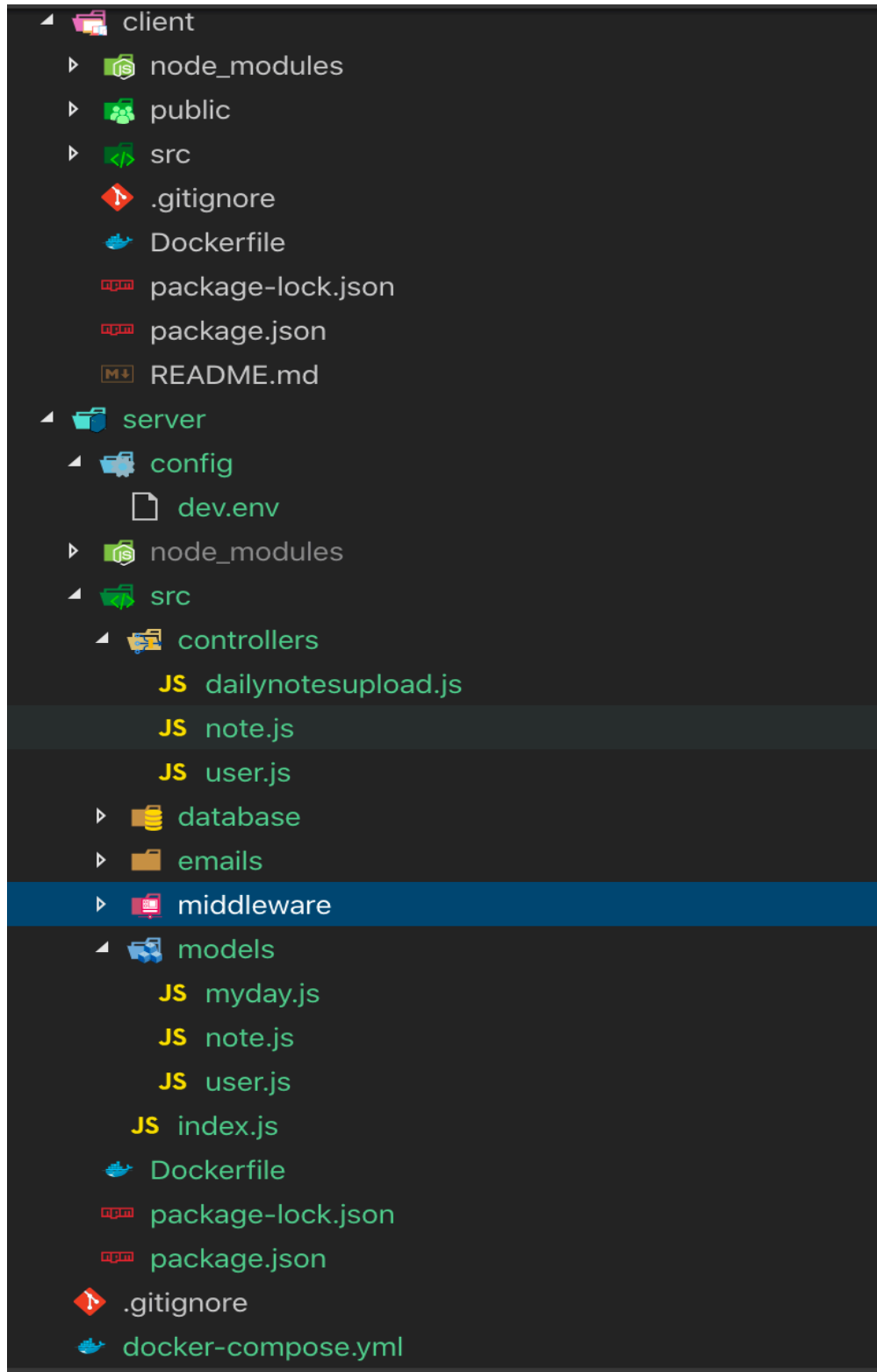


Figure 9. The structure of the project.

5.6 Authentication

Authentication is the process in which a user verifies themselves to have access in the application. There are several ways users can verify themselves to the server. They can verify with the login form, fingerprint, online banking data, voice etc. However, the most common form is the authentication with the login form that has email and password and it was implemented in the application.

A user upon creating an account for themselves with information such as email, name and password. The password is encrypted before being saved in the database. The key concept of handling authentication is the understanding of JWT. When the user tries to log in, the server verifies the credentials of the user by comparing the information that was stored during the creation of the account in the database. If the data match, the server sends the JWT token and a session is created and stored in the user's browser as a cookie. The cookie is sent with every other request to establish the connection. The session is destroyed after the user is logout out from the application and during the closing of the browser.

To access the services of the application such as adding notes, adding profile image, changing profile image, editing notes and so on the user must log in to the application first.

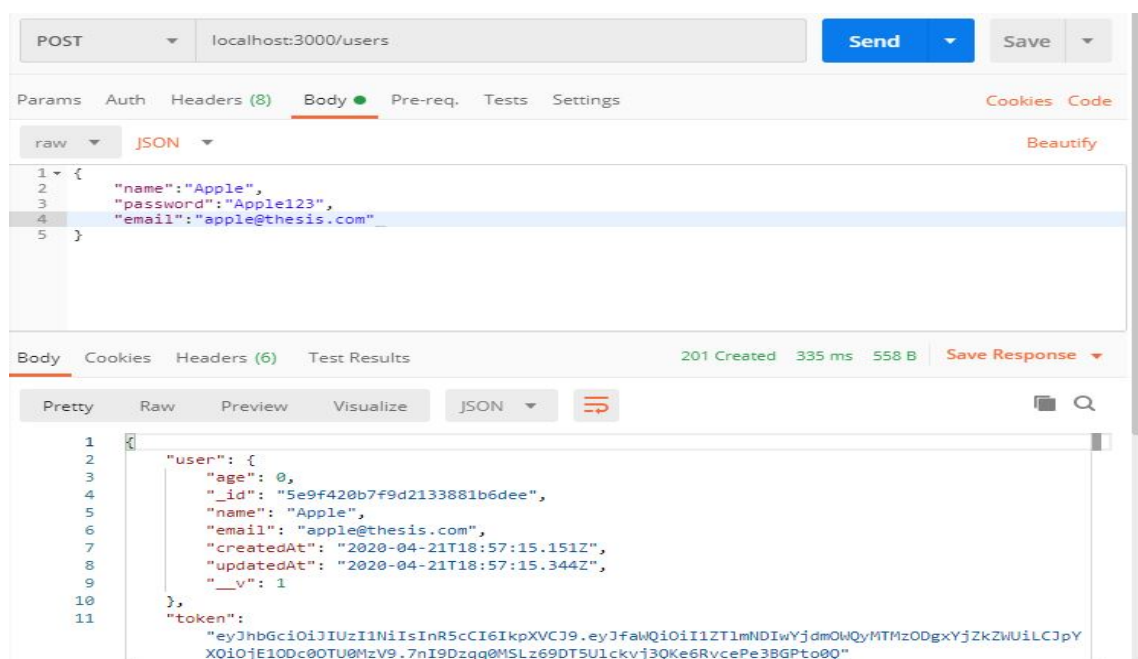


Figure 10. Sign-up to the application.

5.7 Testing with Postman

Postman is a platform for testing APIs, by executing requests and validating responses. API testing is a part of integrating testing that determines whether the APIs meets the users' expectations in terms of application functionality, reliability, performances, and security. Postman returns the response for the test carried out. API testing helps to determine the structure of the output, checks the response based on a request, and checks the time the API takes to retrieve and authorize the data.

All the backend of the application was tested in the Postman by sending a request to the server and getting the response back. In the Postman, users can set up all the headers and cookies the API needs and check the response. It works in the server-side and makes sure that each API endpoint is working as expected. Postman provides a collection of API calls, and one must follow the collection of API calls for testing APIs of the application (Kotecha, 2018). An API response consists of the body, headers, and the status code (Postman, 2020). A request can perform CRUD operations on data and send parameters, authorization details, along with other data that are required. When the request is sent, Postman displays the response received from the API server in a way that lets the user examine, visualize, and troubleshoot if needed. All the responses can be saved and are available whenever the request is loaded. The Postman body tab consists of several tools to help the user to understand the response quickly. User can view the body in one of three views: pretty, raw, and preview. The pretty model formats JSON or XML responses, so the responses are easier to view. The preview model is helpful to restrict the default return of HTML errors. Headers are displayed as key-value pairs and describe the header according to the HTTP specification. Postman breaks down the size of the response into body and headers, the sizes of the responses are approximate.

By default, Postman selects the GET method for a new request. Different request methods are available in Postman to send data to APIs. Below are the four request methods used frequently to test the APIs of the application

- POST Request – Adding new data
- GET Request – Updating the data
- PATCH Request – Updating some existing data fields
- DELETE Request – Deleting existing data

For instance, Figure 11 shows the output during the Login of the User. Postman generated the status code of 200 which meant succeeded in login. Similarly, all the API endpoints were tested with the Postman and the desired response status code was received during the test.

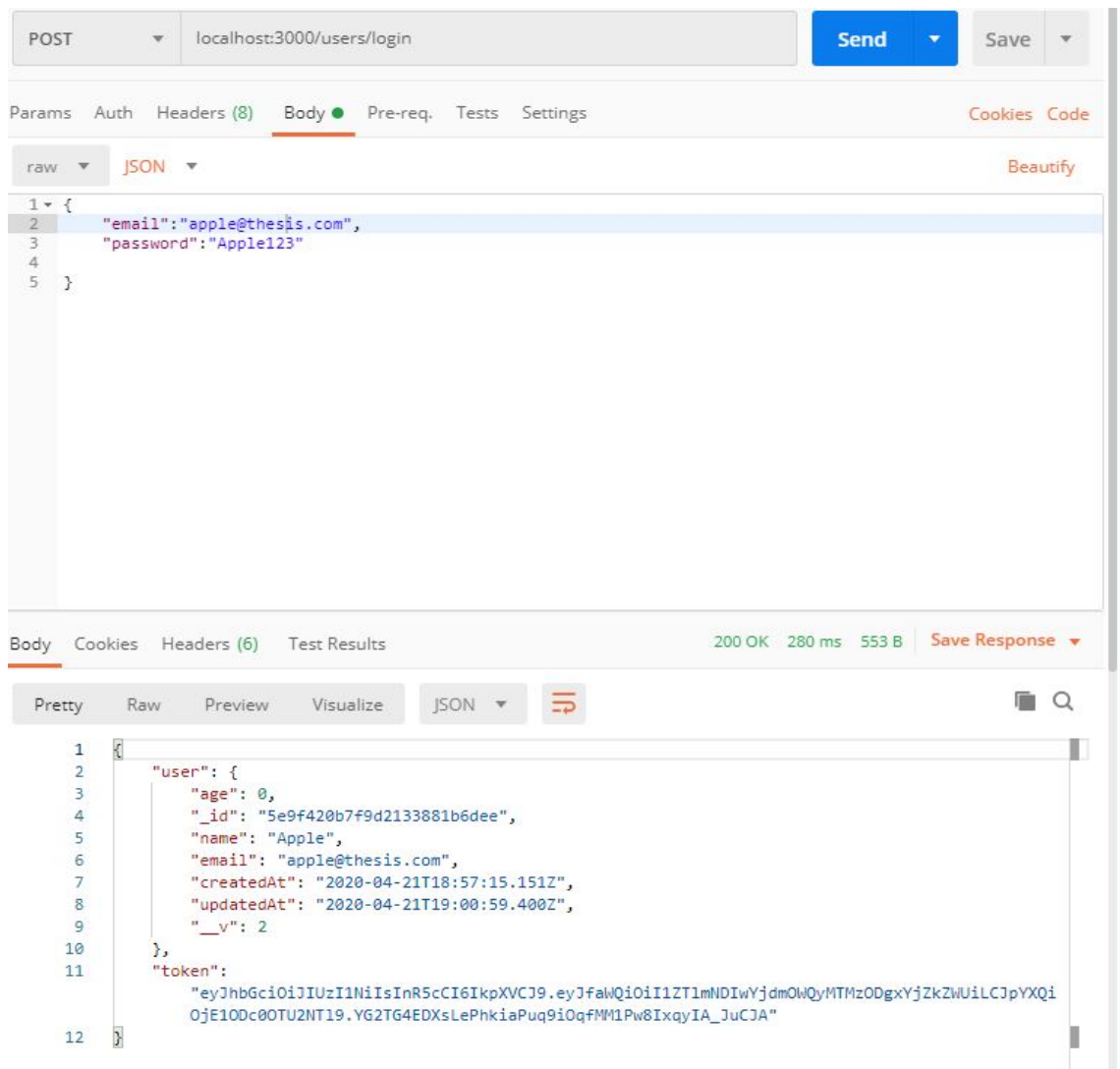


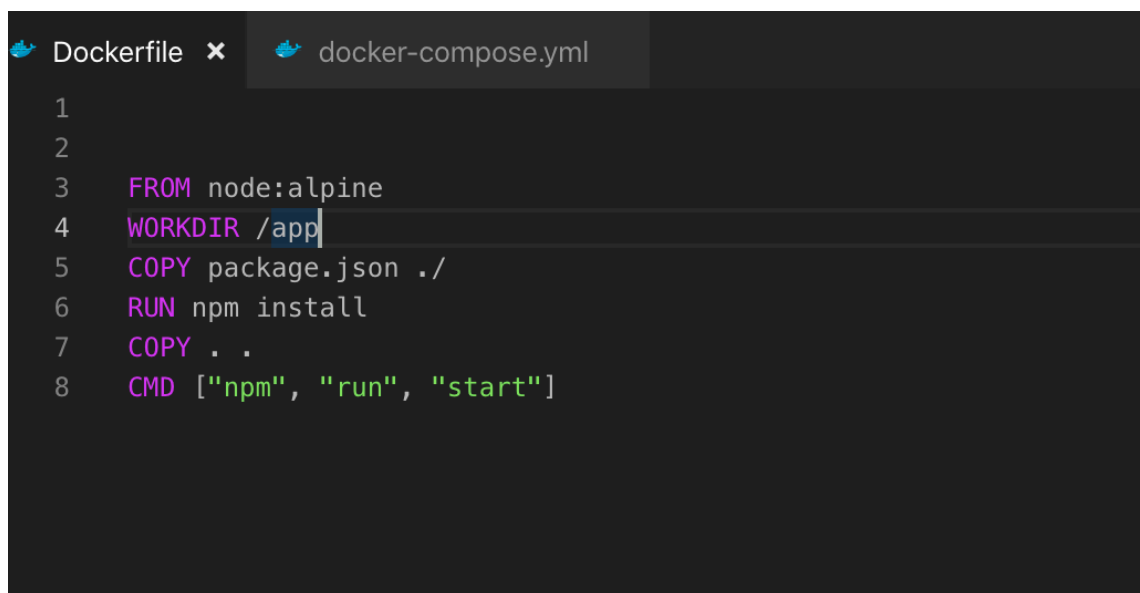
Figure 11. Login Testing with Postman.

5.8 Docker Implementation

As mentioned earlier in the docker objects; section, docker images are built with *Dockerfile*, where all the steps are defined in *Dockerfile*. When creating the *Dockerfile*, three things need to be considered which are mentioned below.

- The base image needs to be specified
- Run some commands to install dependencies
- Command to run on the container on startup

In the project, two *Dockerfile* were created meaning the project has two images, each for the frontend and backend of the application. Although the development of frontend was beyond the scope of this thesis, a project which had frontend was to merge in the containerization process. This would resemble real project and thesis could also explore the ways to manage two containers instead of one with the use of Docker-compose. Figure 12 shows the *Dockerfile* created for backend development.

A screenshot of a code editor showing a Dockerfile. The editor has two tabs: 'Dockerfile' and 'docker-compose.yml'. The Dockerfile content is as follows:

```
1
2
3 FROM node:alpine
4 WORKDIR /app
5 COPY package.json ./
6 RUN npm install
7 COPY . .
8 CMD ["npm", "run", "start"]
```

Figure 12. *Dockerfile* of backend.

The image has the base image of the Node. Alpine is the lightweight and stable version Node. The working directory is defined and *package.json* file will be copied to the working directory and then the dependencies for the project will be added. Finally, the command “npm run start” is executed.

The file should be named as `Dockefile` and “`docker build . -t image-name:tag sh`” command will build the image. The command will give a shell session inside the container to interact with it. “`docker ps`” will list out the running container.

Dockerfile is used to manage a single container but when multiple containers should be considered, it becomes difficult to manage only with *Dockerfile*. The applications in real life have external dependencies database or external service that the application has to rely upon. The communication between the containers becomes difficult to establish so Docker-compose comes into the scene.

Docker Compose

Docker Compose is a tool for defining and running many containers of Docker applications. A YAML file is used to configure full application including its all dependencies in a single file, which makes development easy. The building of Docker-compose file can be divided into six main steps as bellows.

1. Splitting into services
2. Pulling or building image
3. Configuring environment variables
4. configuring networking
5. setting up volumes
6. Building and running

With these above-mentioned steps, Docker-compose file is created as shown in the figure below. Two images that were built previously using the docker file was listed under services. The build will investigate the context directory and run the *Dockerfile*. Creating an image means taking a snapshot of all the source code inside the directory. A docker volume has a reference that points to the local directory from which the mage was made. This is the way to link files and folders in a docker container to the files in the local directory. Environment variables that needed for the projects to run are kept under environment.

Running of the container is done with the command “`docker-compose up -d (detach)`” The “d” flag help to launch the containers in the background. “`docker ps`” command will list the running containers.

```
docker-compose.yml x
1  version: '3'
2  services:
3    client:
4      build:
5        context: ./client
6        dockerfile: Dockerfile
7      volumes:
8        - /app/node_modules
9        - ./client:/app
10     ports:
11       - '3000:3000'
12     restart: on-failure
13     container_name: client
14   backend:
15     build:
16       context: ./server
17       dockerfile: Dockerfile
18     ports:
19       - '5000:5000'
20     restart: on-failure
21     container_name: backend
22     volumes:
23       - /app/node_modules
24       - ./server:/app
25     environment:
26       - SENDGRID_API_KEY= *****
27       - MONGODB_URL= *****
28       - JWT_SECRET= *****
29       - PORT=5001
30
31
```

Figure 13. Docker-compose.yml file of the project.

6 RESULTS AND DISCUSSION

The MEAN and MERN stack are the buzzing stack among the developers. The only difference between them is either the use of React or Angular for the frontend development. However, the backend development of both the stack is the same which the thesis tried to explain and implement.

In the Notes application, when testing the API endpoints, a user was able to create an account, log in into the system with their credentials, was greeted with welcome mail upon signing up in their signing up email. The user was able to add notes, edit, fetch each note or all notes and delete the particular note. Besides, the user was able to update their credentials and upload their Image and logout from the system. Finally, when deleting the account from the system the user was sent a goodbye email. The application was a success as all the routes worked correctly either when performing the CRUD operation or when adding an external package into the app like for uploading the image or sending the email. Thus, the application fulfilled its ambitions in creating the backend.

Although the author was familiar with the JavaScript, the development of backend was comparatively new. The main challenge in the application was learning about the Models methods and how the Models interacted in the NoSQL database. NodeJS supporting different external packages made work easier but understanding the integration into the application took a considerate amount of time. The app in itself has all the basic functionalities of CRUD operations, use of the external library, authentication system but more additional features can be implemented like role-based authentication, testing and frontend implementation etc.

The Use of Node, Express, MongoDB provided much flexibility as many third-party packages can be easily integrated. The flexibility is one of the main reasons for being it so popular. Furthermore, knowledge of JavaScript alone can be enough for developing a full application so, there is not any need to switch between different programming language. Likewise, the use of JSON format and cloud service by MongoDB to store data is useful and easier to transform the work into production mode. Finally, the MVC architecture could be easily applied so, the logic, data and the view are distinct.

In addition to the development of backend, packaging it with Docker was also other main ambition and it was carried out successfully as shown in Figure 14.

```

mypc :: Desktop/sujan_thesis/Final thesis structure <master> % docker-compose up --build
Creating network "finalthesisstructure_default" with the default driver
Building client
Step 1/6 : FROM node:alpine
--> ea308280893e
Step 2/6 : WORKDIR /app
--> Using cache
--> e6d1d3f5baae
Step 3/6 : COPY package*.json ./
--> Using cache
--> a84967eb156b
Step 4/6 : RUN npm install
--> Using cache
--> 50e976648032
Step 5/6 : COPY . .
--> Using cache
--> 0c298c7ec896
Step 6/6 : CMD ["npm", "run", "start"]
--> Using cache
--> 4a3e3a7acf73
Successfully built 4a3e3a7acf73
Successfully tagged finalthesisstructure_client:latest
Building backend
Step 1/6 : FROM node:alpine
--> ea308280893e
Step 2/6 : WORKDIR /app
--> Using cache
--> e6d1d3f5baae
Step 3/6 : COPY package.json ./
--> Using cache
--> 408fb502979c
Step 4/6 : RUN npm install
--> Using cache
--> 57c4551fa1b7
Step 5/6 : COPY . .
--> f4b9f8029f6b
Step 6/6 : CMD ["npm", "run", "start"]
--> Running in 04b3142e20fb
Removing intermediate container 04b3142e20fb
--> 37ff31f84577
Successfully built 37ff31f84577
Successfully tagged finalthesisstructure_backend:latest
Creating client ... done
Creating backend ... done
Attaching to backend, client

```

Figure 14. The result from docker-compose.

Two *Dockerfile* were written to create the image for the frontend and backend and listed as services in the Docker-compose file. Instructions that were listed in the file were carried out to run each image in their own container. The main benefit of docker is that the whole project is summarized on a single file and when other user needs to run the same project, the user does not need to take care of the installation of individual component and dependencies. This would save time and becomes handy in CI/CD (Continuous Integration/Continuous Delivery) environment where each small change involves a workflow that implies running the stack each time to verify that it is stable.

Since this process happens several times a day and sometimes concurrently when changes are made from different persons simultaneously, docker helps to get faster software iterations without the need to spend on better hardware (Docker Documentation, 2020).

7 CONCLUSION

This thesis aimed to study the backend of web development with NodeJS, MongoDB, Express, containerization technology, develop a prototype based on learning and finally package the developed application with Docker. A significant amount of time was invested in learning each technology and knowledge was later used to develop the application and packaged with docker. The key concepts that were understood are discussed in the theoretical part of this thesis followed by an in-depth guide to build the application.

The prototype built was a web application for keeping daily notes, the idea of an online personal diary that manages the notes or memos written in it. Users were able to register into the application with an email address and password and logging-in using the credentials. The registered users were able to use different features such as creating, editing, updating, deleting, uploading images, getting emails etc. The application uses a modern paradigm of writing code e.g. making the code modular, separation of logic.

The developed prototype is wrapped into a container using Docker which enables the application to be portable, meaning that any other developer who wants to run the application does not need to take care of the installation of dependencies. The DevOps process is carried out fast, resulting in the faster lifecycle of software development.

Lastly, the thesis is the documentation of developing a backend for MEAN/MERN stack and implementing Docker. With the growth of internet user around the world increasing, businesses whether small or large are modelling into an online business, so the need for web technology is more than ever. Consequently, NodeJS offering client-server development integration, aiding code reusability in web applications, is a perfect tool for developing fast, scalable network applications (Chaniotis, and Tselikas, 2014).

REFERENCES

- Bcrypt, G., 2020. *Kelektiv/Node.Bcrypt.Js*. [online] GitHub. Available at: <<https://github.com/kelektiv/node.bcrypt.js/>> [Accessed 26 April 2020].
- Burwood Group. 2019. *Containerization Vs. Virtualization: What's The Difference?* [online] Available at: <<https://www.burwood.com/blog-archive/containerization-vs-virtualization>> [Accessed 16 April 2020].
- Chaniotis, I., Ioannis, K. and Tselikas, N., 2014. Is Node.js a viable option for building modern web applications? A performance evaluation study.
- Chettri, N., 2016. *A Comparative Analysis of Node.Js (Server-Side Javascript)*. [online] Repository.stcloudstate.edu. Available at: <https://repository.stcloudstate.edu/csit_etds> [Accessed 1 May 2020].
- Code, V., 2020. *Documentation for Visual Studio Code*. [online] Code.visualstudio.com. Available at: <<https://code.visualstudio.com/docs>> [Accessed 26 March 2020].
- Docker Documentation. 2020a. *Docker Documentation*. [online] Available at: <<https://docs.docker.com/>> [Accessed 27 April 2020].
- Docker Documentation. 2020b. *Docker Overview*. [online] Available at: <<https://docs.docker.com/get-started/overview/#docker-architecture>> [Accessed 29 April 2020].
- Oracle. 2012. *1.1.1. Brief History of Virtualization*. [online] Available at: <https://docs.oracle.com/cd/E26996_01/E18549/html/VMUSG1010.html> [Accessed 25 April 2020].
- Express Documentation. 2020. *Using Express Middleware*. [online] Available at: <<https://expressjs.com/en/guide/using-middlewares.html>> [Accessed 19 April 2020].
- Fullstackopen.com. 2020. *Fullstack Part0* |. [online] Available at: <https://fullstackopen.com/en/part0/fundamentals_of_web_apps> [Accessed 25 March 2020] Heading of Appendix
- Git-scm.com. 2020. *Git - What Is Git?*. [online] Available at: <<https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F>> [Accessed 26 March 2020].
- Hellman, E., 2020. *Understanding CPU And I/O Bound For Asynchronous Operations*. [online] Hellsoft. Available at: <<https://www.hellsoft.se/understanding-cpu-and-i-o-bound-for-asynchronous-operations/>> [Accessed 6 May 2020].
- JWT, D., 2020. *JWT.IO - JSON Web Tokens Introduction*. [online] Jwt.io. Available at: <<https://jwt.io/introduction/>> [Accessed 20 April 2020].
- Kleyman, B., 2012. *Hypervisor 101: Understanding the Virtualization Market*. Available at: <http://www.datacenterknowledge.com/archives/2012/08/01/hypervisor-101-a-look-hypervisormarket/> [Accessed 28 March 2020].

MDN Web Docs. 2020a. *About Javascript*. [online] Available at: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript> [Accessed 10 March 2020].

MDN Web Docs. 2020b. *Express/Node Introduction*. [online] Available at: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction> [Accessed 13 April 2020].

PayPal Engineering. 2020. *Node.Js At Paypal*. [online] Available at: <<https://medium.com/paypal-engineering/node-js-at-paypal-4e2d1d08ce4f>> [Accessed 26 March 2020].

Munro, J., 2020. *An Introduction to MongooseFor MongoDB and Node.Js*. [online] Code Envato Tuts+. Available at: <<https://code.tutsplus.com/articles/an-introduction-to-mongoose-for-mongodb-and-nodejs--cms-29527>> [Accessed 18 April 2020].

Ofogbu, V., 2018. *The Only Nodejs Introduction You'LI Ever Need*. [online] codeburst.io. Available at: <<https://codeburst.io/the-only-nodejs-introduction-youll-ever-need-d969a47ef219>> [Accessed 11 April 2020].

OnCrawl. 2020. *How Does A Browser Create A Web Page? - Oncrawl*. [online] Available at: <<https://www.oncrawl.com/technical-seo/how-does-a-browser-create-a-web-page/>> [Accessed 18 April 2020].

Postman,. 2020. *Responses*. [online] Postman Learning Center. Available at: <<https://learning.postman.com/docs/postman/sending-api-requests/responses/>> [Accessed 30 April 2020].

Ramirez, J., 2020. *Jorgeramirez/Se2019-Node-Express*. [online] GitHub. Available at: <<https://github.com/jorgeramirez/se2019-node-express/blob/master/day3/README.md>> [Accessed 29 April 2020].

Rauschmayer, A., 2014. *Speaking Javascript*. Sebastopol, CA: O'Reilly Media, Inc.

React Kung Fu. 2020. *The Difference Between Virtual DOM And DOM*. [online] Available at: <<https://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/>> [Accessed 2 April 2020].

Runestone.academy. 2020. *1.2. History — Fundamentals Of Web Programming*. [online] Available at: <<https://runestone.academy/runestone/books/published/webfundamentals/WWW/history.html>> [Accessed 26 March 2020].

Sendgrid, d., 2020. [online] Sendgrid.com. Available at: <<https://sendgrid.com/wp-content/uploads/2016/09/SendGrid-Implementation-Review.pdf>> [Accessed 26 April 2020].

Stack Overflow. 2020. *Stack Overflow Developer Survey 2018*. [online] Available at: <<https://insights.stackoverflow.com/survey/2018#work-version-control>> [Accessed 26 March 2020].

Subramanian, V., n.d. *Pro MERN Stack: Full Stack Web App Development With Mongo, Express, React, And Node*. Apress, p.93.

Syromiatnikov, A. and Weyns, D., 2014. A Journey through the Land of Model-View-Design Patterns. *Working IEEE/IFIP Conference on Software Architecture 2014*.

Theodinproject.com. 2020. *Introduction: What Is Nodejs | The Odin Project*. [online] Available at: <<https://www.theodinproject.com/courses/nodejs/lessons/introduction-what-is-nodejs>> [Accessed 2 May 2020].

Yadav, A., Garg, M. and Mehara, R., 2019. Docker Containers Versus Virtual Machine-Based Virtualization: Proceedings of IEMIS 2018. *Emerging Technologies in Data Mining and Information Security*, (10.1007/978-981-13-1501-5_12).