Online Banking Application Using A MERN Stack

Ultan Kearns

B.Sc.(Hons) in Software Development

Contents

List of Figures

About This Project

Analysis of This Project: This project was designed by me for the module Applied Project & Dissertation with the purpose being to complete an online banking system using a MERN(Mongo, Express, React & Node) stack. The project should allow user to login, view statements, takeout loans, perform transactions and will show the user there monthly and yearly expenditures using graphs. The project will also be secure and utilize user accounts using a mongo Database to ensure that the users account is secure.

The main purpose of this project is to utilize the MERN stack to provide a full and rich user experience and to provide a secure, intuitive and polished online banking system. The project will also utilize Python scripts to perform statistical analysis on user expenditure and income and will provide an estimate of how much money the user should have for the month based upon previous monthly expenditure.

This project was designed to be a stand alone application where a user can perform all their banking needs without any other software. The user should be find the UI intuitive and the features helpful.

This project will link many disparate technologies together for the purpose of providing the user with the features they need. I plan to use this project to show the skills I have attained during my course and to learn a new framework(React). I also plan to improve and cultivate my skills using new technologies such as various Python libraries and React. This project is stored on a GitHub repository the link to which can be found in the appendix.

Authors: My name is **Ultan Kearns**, I am a fourth year student at GMIT. I have never used React or LaTeX before but I plan to learn a lot about these technologies during the course of this project.

Chapter 1

Introduction

1.1 Why A MERN stack?
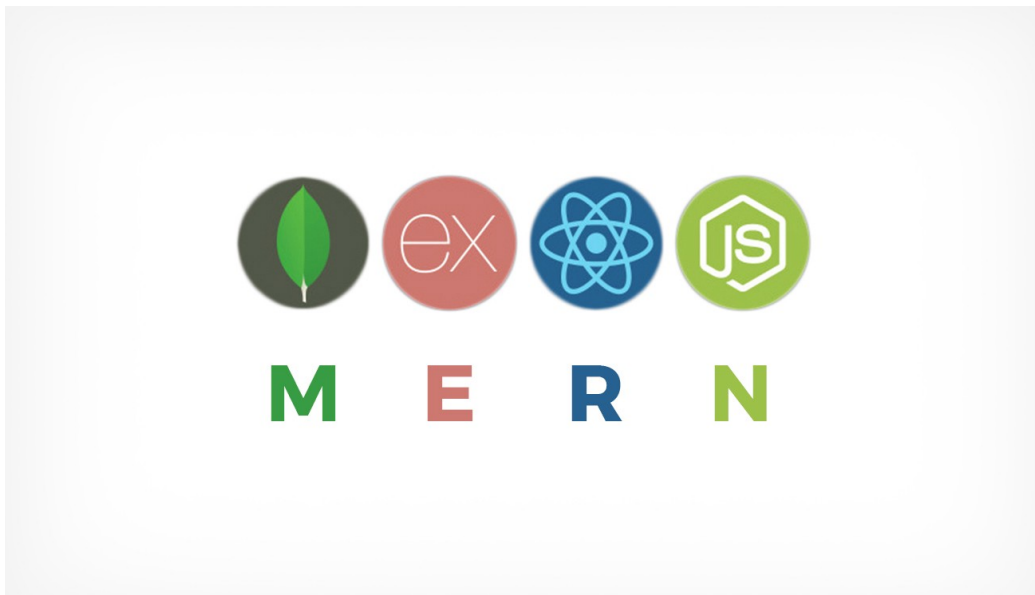
Figure 1.1: MERN Logo

There are many reasons why I have chosen to use a MERN stack for this application the main one is because it provides a framework for a full stack application. MERN stands for **Mongo - For the database backend(Hosted on Mlab, all user info is stored here), Express - A web application framework host a server in a relatively short time compared to**

**other methods , ReactJS for asynchronous JS & Node - For the
server and also includes a package manager to install a variety of
packages to provide certain functionalities.** [1]

1.1.1 Mongo

As you have read above the MERN stack is very powerful in creating a
full-stack application when used correctly. The reason I chose the database
Mongo is because it offers an open source alternative to MySQL and other
proprietary databases [2] and many companies seem to be migrating to it
because it is open source and in some cases may offer better performance
than other databases. [3] Mongo is easy to learn as it stores it's data in

What databases are you using?

1126 respondents - multiple choice answers

Database	Percentage
MongoDB	63.77 %
Redis	36.77%
MySQL / MariaDB	32.95%
Postgre SQL	31.71%
ElasticSearch	17.41%

trace
by RisingStack
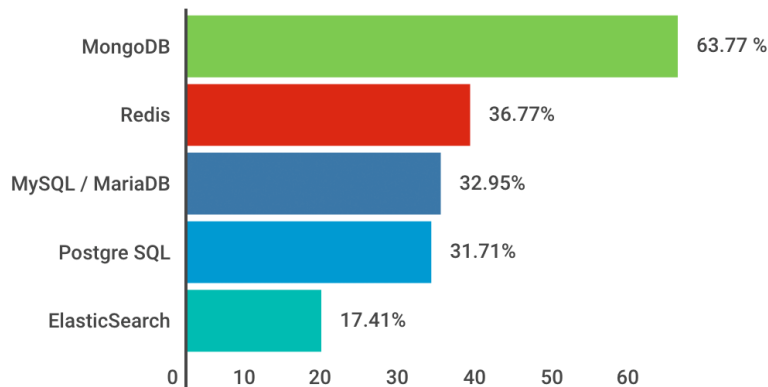
Node.js Survey: survey.risingstack.com

Figure 1.2: Mongo Usage

JSON format and is schemaless, that is the user defines their own schemas. I
have also used it for 2 other projects in Angular and find it an easy database
to use in relation to projects. [4]

Figure 1.3: Mongo Logo

1.1.2 Express

I chose to use Express because it is a very useful framework when creating a server. It is very helpful when pulling data from the server and displaying it to the client it is also very scalable and offers good security when used properly. It is also very easy to setup and can be connected to the MLab database in minutes upon starting the project. I have also used this server before and have a bit of experience with it and found it very helpful in the projects I used it for, which were a messaging forum and an E-commerce application both using MEAN(Mongo,Express,Angular & Node) stacks.

Figure 1.4: Express JS

[5]

1.1.3 React

I also chose this project because I have zero experience in ReactJS and since it's such an up and coming framework I decided on it for this project. It also offers many features and libraries which are desirable when programming a user interface to ensure the user finds the application intuitive and easy to use. ReactJS also utilizes modular programming for components which makes designing web applications far easier than just using HTML, CSS & Javascript. React is also very fast at rendering pages so that the user will not experience a major delay in accessing information.

Figure 1.5: React

[6]

1.1.4 Node JS

The reason I chose Node JS was because it offers a great package manager which can be used to improve productivity on my part and also the applications security, UI, UX and various other aspects of the application. I debated using Yarn for this project but settled on Node because it was already installed on my laptop. Node is a very useful tool when developing full-stack applications and was very helpful in installing tools and libraries for react. Node is also very good for getting a server up and running in a few minutes and handles HTTP requests very well.

[7]

1.2 Why a banking application?

1.2.1 Explanation of Why I Chose This Project

A banking application is broad in scope and offers many questions to the developer such as how can I optimize load times and how can I ensure user

Figure 1.6: Node

data is secure. I think questions such as these offer the potential for growth in the areas of design and problem solving - which are two of the most major skills a software developer can possess. I feel that an application such as a multi-user banking system can be beneficial to my career and help to improve my skills as a developer. Banking applications also have the potential to offer a wide array of features and are ubiquitous in the real world, think major banks such as AIB & Bank of Ireland. Banking systems also offer a broad range of problems such as security, design and usability.

1.2.2 My Contention With Existing Online Banking Systems & How I Plan To Improve Upon Them

The current generation of online banking systems or at least the ones I have used tend to have a variety of problems. These problems are glaringly obvious to most people and the main problems include but are not limited to: usability, appearance, lack of personalization & lack of information given to user. I will how I plan to solve each of these problems below:

- **Usability:** I aim to provide an intuitive and cutting-edge user interface using the latest react libraries to provide the user with an easy to

understand banking application. All features will be easily navigated to using a navigation bar and I will aim to make features as obvious as possible to the user.

- **Appearance:** The UI of the modern day online banking system tends to be absolutely depressing. I aim to reduce this by adding in a responsive UI and to offer the user a vibrant online banking experience.

- **Lack of personalization:** I aim to make this application very personal to the user by adding in personalized expenditure charts and giving the user a unique and inimitable banking experience.

- **Lack of information:** Modern banking applications sometimes display a lack of information given to the user. I plan to solve this by offering the user expenditure charts, reports and also by sending automated emails to them when their account balance falls below a user specified number.

1.3 Requirements

Below I will include the requirements for this application and expand upon them.

- **Multiple Users:** This application must allow multiple users to execute simultaneous banking and user sessions must be independent of each other.

- **Secure:** Database information must be encrypted

- **Accurate:** All statments and user information must be accurate.

- **Login/Logout** The user must be able to login and logout

- **information**: The user must be able to view all information related to them (eg: statments, withdrawl dates etc.)

- **Graphs/Charts**: The banking app must display expenditures and credit in graphs and charts generated from python scriptss

- **Emails** The application must be able to email the user a forgot password if they forgot their password and alos be able to email the user if budget controls are turned on.

- **Register** The user must be able to register new accounts using an email and password also ensure that it cannot be an email that already exists.

- **Delete account** The user must be able to terminate their account and all information that exists about the user must be purged from the server.

- **Change information** The user must be able to update all their personal information in relation to their account except obviously banking statments and balances.

- **User sessions** The application must use cookies to maintain a user session and ensure that the cookies do not last more than a specified timeframe max of a day.

1.4 Outline of chapters

Below I will outline the chapters that my Dissertation is broken up into and give a brief outline of each one.

1.4.1 Context

In Chapter 2 I will discuss the context of my project and how online banking applications have affected the modern age and traditional banking. I will research how online banking came about and how it is useful for the consumer as well as the bank.

1.4.2 Methodology

In Chapter 3 I will discuss the methodology I followed and how it affected my project and productivity. I will also discuss my how I planned to complete the project and discuss the methodologies I utilized in completing this project. I will discuss why I chose these methodologies and give the reader an insight into how this application was developed.

1.4.3 Technical Review

In chapter 4 I will discuss the technical aspects of this project and discuss how they impacted the development of this project and why they were implemented. I will discuss the MERN stack in more detail and how it was utilized to create a full-stack online banking application.

1.4.4 System Design

In Chapter 5 I will explain the architecture and design of this project. I will use graphs and diagrams to explain how the application is designed and how it will function when deployed. I will also present some of the code I used and how it is used to perform various functions of the banking application.

1.4.5 System Evaluation

In chapter 6 I will analyze the finished product. I will test the system and evaluate if it is up to standard and meets all the requirements I have specified. I will test if the scalability, security and the UI to ensure that the user is provided with the features outlined in the requirements section.

1.4.6 Conclusion

In chapter 7 will briefly outline what I learned from this project and highlight all my findings from previous sections. I will also discuss the impact the project had on my skills as a software developer and how it helped me to grow as a developer. I will also discuss what I would do differently if I had to do the project over again.

1.5 Structure of Project

The github project contains two branches feature and master I use the feature branch to test all new code and ensure that it works properly before merging it with the master branch. The git repository contains two folders banking-app and dissertation, the banking app folder contains the main project and the dissertation contains all materials relating to the dissertation. In addition to the two folders I also have a sprints.md file which contains information about all my sprints as I decided to use the agile methodology during this project, I also have a usefulresources.md file which contains a list of useful resources which helped me throughout the course of this project. The final file is the README.md this will contain a brief intro to the project as well as information on running the application.

Chapter 2

Context

2.1   Project Objectives

- To provide safe & secure online banking

- To provide an intuitive UI that can be easily navigated by the user

- To provide user generated statistical analysis of expenditure

- To provide a multi-user server and banking service

- To provide a RESTful API to the banking service(client/server totally independent ,stateless environment, caching)

- To provide a scalable application

- To provide user with security using encryption for the mongo database

- To limit loadtimes of traditional online banking eg. 365 Online Banking and others

2.2   Online Banking

Online banking is very relevant in today's modern world, the ease of access which the digital age allows us has led to a significant amount of banking being done online. Think of how many inconveniences have been shed away by the advent of online banking, no longer do people have to wait in line for ages at the bank to take out loans or to view their statements as all this can be done online or with the help of computers. Every modern bank now has some form of website which allows their customers to do all their daily

tasks related to banking online. In this chapter I will discuss the advantages, disadvantages and overall affect of online banking.

2.3 History of Online Banking

2.3.1 Definition of Online Banking

The definition of online banking includes any form of electronic payment systems that allows the customer or business entity to conduct financial transactions via a financial institutions website or application.

2.3.2 The Beginning

The metamorphosis of the old brick and mortar banks to the click and mortar banks of today all started as early as the 1980s [8]. The earliest version of the online prescence of banking took place in none other than New York City, New York, USA. Citibank, Chase Manhattan, Chemical Bank and manufacturers Hanover became the first banks to introduce on online banking system in 1981.[8] Then in 1983 the Bank of Scotland became the first bank in the UK to offer an internet banking service which was called Homelink to UK customers.People had to use their phones and their televisions to manage their online banking as computers where not as ubiquitous at this time [8] It was not until 1994 that a bank in the USA called the Stanford Federal Credit Union began to offer online banking services to all of its customers [8].In the year 2006 80% of US banks had began to offer some variation of online banking [8]. It was a slow process to move from the tried and true brick and mortar banking that the majority of the public were,if not entirely satisfied with, were familiar with and accustomed to through decades of useage. The move from brick and mortar to click and mortar was not all advantages, there were many issues which had to be addressed such as ease of access, security and the education of the public about phishing scams and various other crimes which can occur by storing information online. Some banks offered advice to the elderly and people who may not know so much about computers advice in an attempt to cut down on online banking crimes I have refereneced an example of one such site targeting the elderly to dispense such advice [9]

2.4 Advantages of Online Banking

2.4.1 Convenience

There are many advantages to online banking which I will discuss in this section the most prominent of these advantages is convenience. Bank customers no longer have to drive or walk to the bank or to an ATM to check their balance and with online payments customers will never have to withdraw money unnecessarily. You can now easily transfer money between accounts online and set up standing payments to pay your bills which is a lifesaver for some people. You can also use your mobile to view your balance and statements anywhere using mobile data and online banking applications, this allows users to access their account from anywhere in the world depending on the availability of reception.

2.4.2 Speed

The speed of which a bank customer can access their account information is now determined by the speed of their internet. Before the advent of online banking you would have to wait in line at an ATM to see your statements or to withdraw money to pay your bills, now that online banking has become virtually ubiquitous in the modern world you no longer have to wait in lines to perform activities and tasks relating to banking.

2.4.3 Competition

The rise of online banking means that banks are constantly trying to out do each other in terms of their banking applications. The customer now has much more choice in choosing banks since the advent of online banking. The advent of online banking has led to some banks maintaining only an online prescence.[10]

2.4.4 Creation of Jobs

The switch from physical banking to digital banking has created numerous jobs in the software industry especially in terms of cyber-security as banks increasingly worry about security braches and hacks. Any hacks that occur are viewed as the fault of the bank and allows many customers to migrate to other banks, that is why many banks spend a lot of money to secure data on their servers which creates jobs and stimulates the economy.

2.5 Disadvantages of Online Banking

2.5.1 Security

Storing details in a digital environment is a lot less secure than storing them in a non digital format. This happens because the data is stored on a server and if phishing scams getting a user's password are successful they could find that their bank account has been drained of money. There are many other forms of attacks which could occur such as keylogging, network monitoring and on public networks your data is far less secure. Although banks have taken many counter measures to prevent such attacks from occuring sometimes they slip through the cracks of cyber-security analysts[11]. Most people find that a little less security is a fair trade-off for the convenience online banking provides.

2.5.2 Layoff of Bank Employees

As the migration of banking from physical to digital takes place it leads to a surplus of banking employees which are made redundant. This occurs because of the ease of access of online banking and this limits the need for more tellers in many banks.

2.5.3 Increased Crime

As banks migrate there is an increase of cyber-criminals who try to gain access to the banks. According to one article 25% of all malware hits financial services [12]. This increase of crime leads to an increased number of compromised accounts on a banks server.

2.6 The Overall Effect of Online Banking

The overall effect of online banking has had a stunning impact on our world. The availability of a bank 24/7 has provided countless benefits to both consumers and businesses alike. The advantages in my mind as in the minds of many others far outweigh the disadvantages of online banking.

Chapter 3

Methodology

3.1 Overview of Methodology

In this project I utilized the Agile methodology, I used Sprints to coordinate my work and after each sprint I performed module testing and regression testing. I also used Test Driven Development which involves writing tests and writing just enough code to make them pass and then going back and refactoring the code.

3.2 Agile

3.2.1 What is Agile?

Agile is a software development methodology which according to the agile manifesto website [13] favours: "Individuals and interactions over processes and tools, Working software over comprehensive documentation, Customer collaboration over contract negotiation, Responding to change over following a plan". There are many variations of agile [14] such as Scrum, XP, Kanban and many more. All Agile methodologies follow the same principles outlined in the Agile Manifesto[13] For the purposes of this project I'll be using Kanban. All Agile methodologies stress continuous improvement and incremental delivery.
[15]

3.2.2 Why Kanban?

Kanban [16] allows for a visualized workflow and was also easy to implement into the project. It allows for incremental development and a continuous

Agile Lifecycle Diagram for PowerPoint
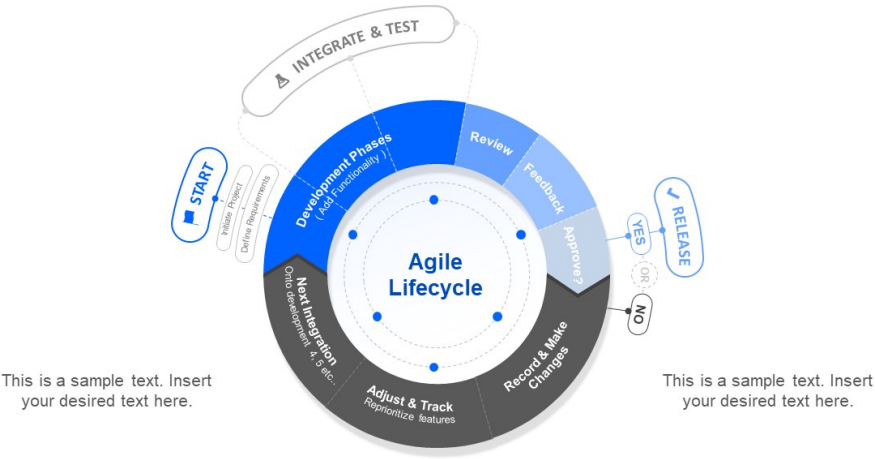
Figure 3.1: Agile

improvement of features. In contrast to waterfall, Agile methodologies also allow the developer to go back and improve upon features. In summary Kanban offered me a visualized workflow and also allowed me to divide my sprints into daily or weekly tasks which I could complete in a relatively short time.

3.2.3 How I Applied Agile To This Project

During this project I used a Kanban board on Github [17] to coordinate my sprints(Backlog) and added the issues that needed to be finished to the Kanban board. When an issue was added it was automatically moved to the "To Do" section of the Kanban board when it was in progress it was moved to the "in progress" section and finally when it was done I moved it to the "Done" section of the Kanban board. The Kanban board helped create a visualized workflow which helped productivity as I could visualize which issues I would be working on that day and also I could coordinate the Kanban board with my sprints to segregate work I needed to accomplish in a fixed time-frame.

Figure 3.2: Image of Kanban board from wikipedia

3.3 Test Driven Development (TDD)

3.3.1 What Is Test Driven Development?

Test Driven Development or TDD is a form of testing during development which focuses on writing the tests of a function before coding it and then writing enough code to make sure that it passes the test and then going back and refactoring it[18].
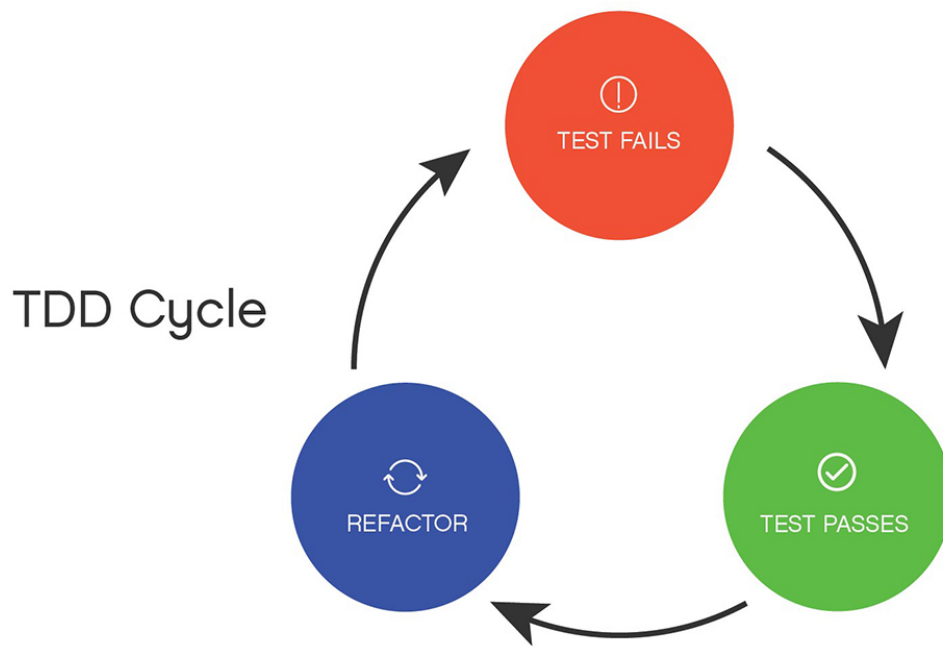

Figure 3.3: TDD

[19]

3.3.2 How I Applied Test Driven Development To This Project

I wrote simple tests as I developed this code, for example for the statements section I wrote a test such as "Output only the statements of the user, output must be in form of location, cost, name, date" I then proceeded to write just enough code to make it pass and then went back and refactored it.

3.4 Time Management

3.4.1 How I Handled Multiple Projects

I was able to handle multiple projects at a time by defining sprints in my final year project and assigning myself daily tasks using the Kanban Board, sometimes I had issues in completing these daily tasks but I always tried to complete them in a timely and effective manner. Sometimes I experienced scope-creep with a task where I had to add more features to the task to make the project viable.

3.4.2 Issues I Faced With Time Management

I faced some issues with time management as the workload was fairly heavy, I had to cut some projects short and leave out some features of various projects to ensure that all projects were completed to an acceptable level.

3.5 Version Control

I used GitHub to manage the versions of this application and to incrementally add code to my repository. I also used GitHub for the Kanban board and to manage any issues that arose during the course of development.

Github also allowed me to have multiple branches, one of which I used for all code that had been tested and was proven to work called Master and another where I implemented and tested desirable features, this branch I called Feature. The compartmentalization of tested and untested code helped me a lot during the course of development as it allowed me to perform accurate regression tests and module tests which made debugging the application a lot easier.

Github also served as a backup of all my code in case of hard drive errors, hardware failures or various other issues that may arise. Github also allowed me to revert back to previous versions of the project just in case I had made a mistake in the code and needed to revert back to a working version. Github also served to provide me with a place were I could log any issues I had with the project and also allowed me to divide my sprints into daily or weekly tasks since the sprints were usually set over the course of a few days.
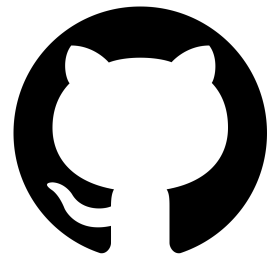
Figure 3.4: Github Logo from github.com

Chapter 4

Technology Review

4.1 Intro

In this chapter I will explore the technologies I used in this project and why I used them. I will also explain how the technologies were implemented and how they were utilized to ensure that the online banking application was fully functional. I will also review the technologies I used in this project and show how each technology provided a desired functionality which would be paramount to achieving a fully fledged robust, secure and overall user-friendly baking system.

4.2 MERN Stack

The MERN stack is a technology which is used to make full-stack web applications. Each component in a MERN stack has a specific function to allow for full-stack development. The MERN stack is widely used in industry and is composed of the following components:

- MongoDB - For storing items in a database

- Express - Provides a web application framework

- React - A JavaScript library for building UI

- NodeJs - For server architecture and backend

4.2.1 About MongoDB

MongoDB is a document database that is scalable and flexible, it provides the user with a litany of various functions to perform actions on data stored

in the database. MongoDB stores data in JSON-like documents which means the fields can vary between documents and that the structure of the data can morph over time. The use of JSON-like objects makes the data very easy to access and to manipulate for example say I had the following object:

```
{
  "person": {
    "name": "ultan"
  }
}
```

Here if I wanted to acess the given persons name the only command I would have to execute would be

```
person.name();
```

I will explore the simplicity of JSON more in the JSON section but it is a rather pertinent point about MongoDB and it is one of the reasons why I chose to use MongoDB in this project.

In addition to all of the above advantages of MongoDB a major factor in choosing it for this project is that it is open-source which means it is free to use, and as typical of any open-source software it has attracted a strong community which provides support to other users of MongoDB and continues to provide feedback to improve Mongo.

4.2.2   About Express

Express is a Node.js web application framework which provides a myriad of desirable features for setting up a web sever. Express helped me set-up a fully functional web application in a timely manner. I also utilized Express to provide routing to multiple paths and to retrieve, delete, create & update various information stored on my Mongo database. I found Express very easy to setup and to use and the developers of Express provided comprehensive and very informative documentation on Express which can be found on their website[20]

Express was able to return data I needed in JSON(JavaScript Object Notation) format and using the returned information I could then output the returned results to the user for purposes such as: to provide news & to show the user their information. I could also pass through parameters in the path and find information on a certain user or piece of data, this was very important for features such as logging the user in to the system as I had to compare the username and password and ensure both were correct.

In conclusion Express provided a thin layer over Node.js to provide the application with features such as routing and also allows the developer to have a server up and running in minutes. Express provided a lot of functionality to the banking application and also allowed me to create, read, update & delete data, all of these are required for a fully fledged banking application in the 21st century.

4.2.3   About React

ReactJS is a JavaScript library for building user interfaces[21] React is maintained by Facebook and numerous other companies and independent developers. React can be used to create user friendly single-page applications which is perfect for an online banking system which is being designed with usability and accessibility in mind. React has numerous libraries which can be used such as Axios[22].
These libraries can be used for various functions such as ,in the case of Axios, performing routing. ReactJS also allows the developer to compartmentalize their code into different components each with it's own set of functions which makes the code far easier to debug as opposed to debugging a god class or god file.
React is also based on a language I have some familiarity with as I have used JavaScript for multiple web-based projects but I have never used React before, this use of JavaScript in React made it easy for me to dive-in and learn the framework. React also has multiple libraries which can be utilized to provide a modern and minimalistic user interface without sacrificing functionality.
React has been growing in popularity along with similar technologies such as Angular and multiple companies are using these technologies for their websites / applications. A combination of all the above factors listed and Reacts growing popularity made it seem ideal for this project.

4.2.4   About Node

Node JS provided server-side architecture for this project. NodeJS is an asynchronous event-driven JavaScript runtime environment which is designed to be scalable and can be utilized to build network applications[23].
Node allows for concurrent connections to the server and can process multiple requests at a time. NodeJS has a strong community and is open source and provides the developer with server-side architecture, it is also very powerful and can be used to perform HTTP requests such as: GET,PUT,POST,DELETE etc.

I utilized NodeJS in this project to retrieve data via HTTP requests either from MLAB or in the case of news stories, Reddit and various other news sites.

Here is an example of how easy it is to start a basic web application with Node:

```
const https = require("https");

app.get("/", function(req, res) {
  res.status(200).send("Server is up and running!");
});
var server = app.listen(8080, function() {
var host = server.address().address
  var port = server.address().port
  console.log("Example app listening at http://{$server}/{$port}:")
})
```

In the above example what will happen when the user searches the URL https://localhost:8080/ they will be greeted with the message "Server is up and running!", the 200 status indicates that the resource was retrieved successfully.

4.3 Testing

4.3.1 Selenium

Selenium is a browser add-on which automates the browser and allows the user to create tests which will monitor the users actions performed and log them so that these tests can be run and re-run with the click of a button. Selenium helped greatly in the development of this application and freed up time I would have used to perform extensive manual testing and allowed me to add a litany of new features to this application.

4.4 React Libraries

4.4.1 Axios

Axios is a promise based HTTP client for node.js[22]. I used Axios in this project to send HTTP requests from the frontend of the application to the

NodeJS server. Axios allowed me to retrieve information from various paths and return that information to the user, and to update, delete and create new information. I used Axios to achieve full CRUD(Create, read, update & delete) functionality.

4.4.2 React Helmet

React-helmet is a reusable react component that can manage the document head of various components, it allows the developer to define HTML tags such as title and outputs them to the browser[24].

4.4.3 Create-React-App

Create-React-App allows the developer to quickstart a React application without having to worry about build-options or anything else, it's as simple as running "create-react-app new app" in the terminal or CLI(Command line interface)[25].

4.5 Third Party APIs

4.5.1 NodeMailer

NodeMailer is a module for NodeJS that allows the developer to set up an automated email system to send the user personalized emails which could be used to inform user of new features, services etc. For this project I will use it to allow the user to reset their password.

4.5.2 News API

News API is a REST API which searches the web for news stories and retrieves headlines from a variety of websites[26]. News API allows the developer to search by keywords and phrases, dates published, source name and various other parameters which may be relevant to the article in questions.

4.5.3 Reddit

I used the Reddit API to retrieve latest financial news and displayed them to the user. The Reddit API allows the developer to read in JSON from reddit and to parse it and output it to the user.

4.6 Architecture

4.6.1 REST(Representational State Transfer)

REST is an architectural design that was first hypothesized in Roy Fielding's dissertation[27]. REST has six constraints which must be met for an application to be considered "RESTful" these are:

- Client/Server must be independent: The idea behind this principle is that by separating the UI from the backend logic we create a more portable application and also it improves scalability by simplifying the server logic.

- Stateless: Every request from the client to the server must contain all pertinent information to understand the user's request, session state is maintained on the client side.

- Cacheable: Data received must be labelled cacheable or non-cacheable. If it is labelled as cacheable the client may reuse the data.

- Uniform Interface: System architecture is simplified and improved by applying the software principle of generality.

- Layered System: The layered system allows for a heirarchial layer of components in which components can interact with certain components. Components are unaware of other components besides the component with which they are interacting.

4.7 Security

4.7.1 OpenSSL

OpenSSL is a toolkit for providing TLS(Transport Layer Security) & SSL(Secure Socket Layer) protocols. It is also a cryptography library which can be used to enchance the applications security. I used OpenSSL to generate a certificate for the Node server to provide an extra layer of security for the user.
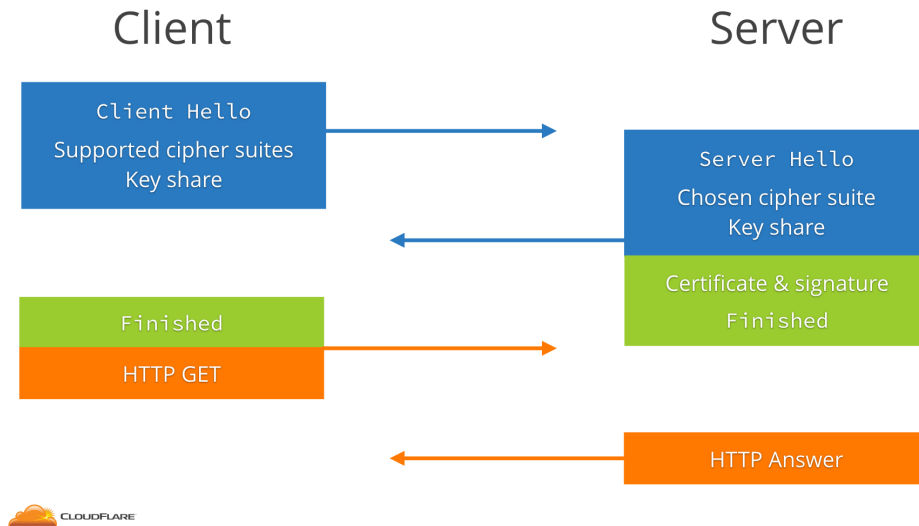
Figure 4.1: TLS.

[28]

4.7.2 SHA256

SHA256 stands for Secure Hashing Algorithm 256 which is used as a one way function which means that data which is hashed using this algorithm is impossible to revert into it's original form[29]. I used the SHA256 algorithm to encrypt and secure user passwords stored in the database and to ensure that the passwords were not stored in plaintext for extra security.

4.8 Database

4.8.1 MLAB

MLAB is a DaaS(Database as a service) platform and provides a free hosting environment for my applications database[30]. The reason I chose MLAB is that it's free and it is easy to setup multiple separate schemas which can then be accessed through connecting to the database via the NodeJS server. All the relevant data to the banking application is stored on MLAB. MLAB utilizes a Mongo database and allows the user to connect to the database using a username and password.

4.9 Languages

4.9.1 Javascript

JavaScript is a lightweight programming language which is interpreted and is compiled just in time[31]. It is a very popular language for web developers as it is Turing Complete and allows the developer to manipulate the DOM(Document Object Model) programmatically.

I used JavaScript for performing various functions which were expected from an online banking application such as: cycling through a users transactions, outputting userdata, comparing inputted values to values retrieved from the server for login, checking if a user already exists etc.

In short JavaScript provided me with a full Turing Complete language to code the logic of the online banking application. Here is an example of some JavaScript code:

```javascript
name = "Ultan"
//prints Hello, Ultan
print("Hello, " + name);
a = 1
b = 2
//prints 3
print(a + b)
//prints Hello, Ultan1
print("Hello, " + name + a)
```

4.9.2 HTML(Hyper-Text Markup Language)

Hyper-Text Markup Language or HTML is a markup language and is the standard for creating web pages[32]. I used HTML to structure my web page and insert various elements into the web pages such as: images, headers, paragraphs etc.

Below I will include a very basic HTML page:

```html
<!DOCTYPE html>
<html>
<head>
<title> You'll find me on your tab!</title>
</head>
```

```
<body>
<h1>This is a header</h1>
<p>Hello world! This is HTML</p>
</body>
</html>
```

All valid HTML pages must have a DOCTYPE html declaration, html tag, head tag, and a body tag. Any tag which is opened must be closed.

4.10   Documentation

4.10.1   LaTeX

LaTeX is a typesetting system that provides various features for designing and writing high quality documentation[33]. LaTeX has become the de-facto standard for Academic publications and is open-source. Having never use LaTeX before I found it very easy to setup and get started, it also made referencing a lot easier and made generating tables of contents & figures very simple.

The reason I chose LaTeX to write this dissertation apart from the above reasons listed, was because it helps to create high quality, Academic style papers with minimal onus placed upon the user.

LaTeX also has extremely useful packages which can be used to provide numerous functionality.

4.11   Styles

4.11.1   Cascading Style Sheets(CSS)

CSS allowed me to style the HTML pages and to make them look how I wanted them to look. CSS describes how the HTML is supposed to look when the browser loads a web page.

Heres an example of CSS:

```
h1{
  //turns color of all h1 elements red
  color:red;
  //centers the text of h1
  text-align:center;
  //changes font-size to 21px
```

```
    font-size:21px;
    //These are just a small sample of things you can change with CSS!
}
```

4.11.2 React Bootstrap

React Bootstrap is a front-end framework for React that provides the developer with a number of components which can be used to create a modern looking and intuitive user-interface.

4.12 Cloud

4.12.1 Google Cloud Platform

Google Cloud Platform is an Infrastructure as a service(IaaS) platform which I will use to host my application and server to allow users to connect to the application remotely[34]. All logic will be performed by the user connection to the Google Cloud Platform server and utilizing the application.
Google Cloud allows the developer to setup a virtual machine in minutes and it is in this way I plan to host my application. I will create a VM(virtual machine) on Google Cloud and open the ports necessary for the end user to communicate with the application. After this is done the user will only need to type in the IP address of the machine and the port they wish to connect to(in our case port:3000) to be able to access the application.
The reason I chose Google Cloud Platform is because I have used AWS(Amazon Web Services) and Azure(Microsoft's cloud platform) before and wanted to try a variety of cloud platforms to gain familiarity and experience with a variety of cloud functions and platforms.

4.12.2 Docker

Docker aims to simplify and accelerate workflows for applications and allows the developer to build and share their applications as well as allowing them to deploy and ship their applications easily[35]. Docker uses containerization to manage the applications which it hosts.
The containers are portable and can run on any host. Docker automatically releases a new build of the project everytime a new commit is pushed to the master branch, it is in this why that docker allows for continuous integration of new components for the application and a continuous delivery of new updates to the application. Docker is extremely useful for Agile development as

continuous integration and delivery are an integral part of the Agile methodology.

I have never used Docker before but I chose to use it for this project as it is rapidly growing in popularity and it is extremely useful in today's industry as companies focus on releasing patches and updates to the user as soon as they are ready.

Chapter 5

System Design

5.1 Introduction

In this section I will discuss the architecture of the application both in terms of frontend and backend. I will discuss each component individually and the reasoning for its existence. I will also discuss how the frontend communicates with the backend, how I designed the application with a strong focus on UX(User Experience) and how I improved upon existing paradigms in online banking. The system is designed to be robust and to be as error prone as possible, I achieved this to the best of my abilities by using TDD(Test Driven Development) and an Agile methodology.

5.2 Architecture of Project

5.2.1 Introduction to System

This application was made using React which is a web-application framework for building user interfaces, for the purposes of my application I used it to build a SPA(Single Page Application) which has multiple advantages over a standard webpage, I will list the advantages here:

- Single Page Applications load faster and are more responsive

- Single Page Applications cache data very effectively, thus decreasing load times

- Creates a more user friendly experience

- Easier to debug as their are many tools such as React Developer Tools.

The system is made up of 11 components some of which share the same cascading stylesheets for a more uniform look and to create a better user experience.

The system's backend was written using NodeJS and utilizes the ExpressJS library which offers the developer a wide range of additional features.

5.2.2 Reasons for System Design

5.3 Front End Architecture

5.3.1 List of Components

Login

The login page is the page that the user first sees when starting the application. The page features an introduction which has white text on a black backround and prompts the user to "Login to experience next generation banking". In addition to this there is a paragraph which states "Please enter your username and password below and we'll redirect you to your account page" below this paragrah are two input boxes which prompt the user to enter their username and password respectively. When the user enters the correct password they are redirected to their accounts page, however if the password or username is entered incorrectly they are alerted by means of a JavaScript alert that their password or username is wrong. In addition to the above stated there are three buttons one which says "Register" which navigates to the register component another which says "Forgot Password" which will navigate to the Forgot component and one which says "Login". Once the user has entered the correct username and password they can hit the login button to be redirected to the main application page. The user cannot enter in a components name and gain access to the component unless logged in. I accomplished the login feature by utilizing code from a previous project's function which I had coded:

```
//template was taken from earlier project and refactored https://github.com/Uli
app.get("/api/users/:id/:password", function(req, res) {
  Users.findById(req.params.id, function(err, data) {
    if (err) {
      //send back error 500 to show the server had internel error
      res.status(500, "INTERNAL SERVER ERROR " + err);
      return;
    } else if (data != null) {
```

```
    //compare user username and password to the username and password in DB
    if (req.params.id == data._id && data.password == req.params.password) {
      res.json(data);
      res.status(200, "User logged in!");
    }
    else {
      res.json("null");
      res.status(404, "User not found!");
    }
  }
});
})
```

I had the idea to reuse the code and refactor it after reading Robert C Martins book "The Clean Coder"[?]. The above code basically says if there's an error throw a 500 which is a generic server error, if there is no such error and data is retrieved then respond with the data and return a 200 else return null and send back 404 to show that the requested resource was not found. I handle all issues which may arise in the login component's logic:
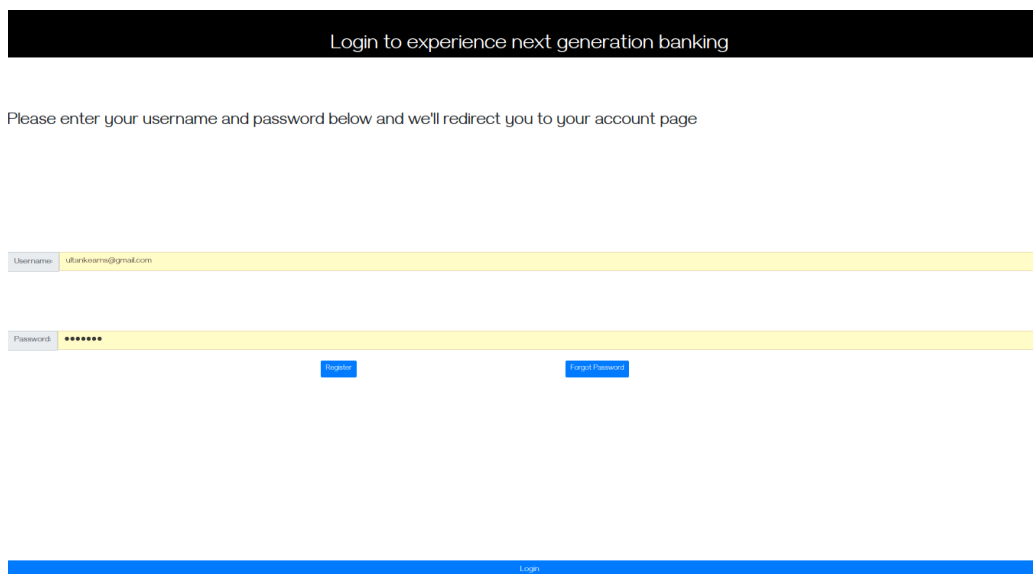
```
handleSubmitForm = event => {
  const axios = require("axios").default;
  const sha256 = require('js-sha256');
   axios
     .get(
       "https://localhost:8080/api/users/" +
         this.state.username.toLowerCase() +
         "/" +
         sha256(this.state.password)
     )
    .then(function(res) {
       console.log("TEST LOGIN " + res.data);
       //need to fix this so it shows error message
        if (res.data !== "null") {
         //store the username this will help the bank feel more personal
         sessionStorage.setItem("username", res.data.name);
         sessionStorage.setItem("email", res.data._id);
         ReactDOM.render(<Home />, document.getElementById("root"));
       }
       else if(res.data == "null"){
         alert("Wrong username or password")
       }
```

```
  }).catch(error =>{
    alert("Error logging in, check internet connection?")
  });

  console.log("Clicked")
event.preventDefault();
};
```

Here I send a get request to the BackEnd URL to retrieve the user using the username and hashed password(password was hashed using SHA256 security standard), next I check if data is null if not login user and render the homepage I also set the sessionStorage items for other logic in the application. If the username or password is wrong I return a message to the user telling them. I also have a message if the client is unable to communicate with the server.

Figure 5.1: Image of Login Component

Register

The register component is accessible from the login component and is used when the user wishes to create an account. The register component features five inputs which are: Username for the users email, password for the users

password which is needed to login and is stored as a hash on the server, full name which stores the users full name on the server, phone number which will be used for 2 factor authentication, and date of birth which when clicked will present the user with a calender so that they can select their date of birth. The register component also allows the user to return to the login page via a button whose value is "back", alternatively the user can chose the button that says register which will send a POST request to the backend to create a new user account. The username in this application is a primary key which assures that the key must be unique, if the user choses an email which already exists on the server they will be greeted with a message saying "User already exists", this is accomplished by the code below:

```
//check if user exists
axios
  .get("https://localhost:8080/api/users/" + this.state.username)
  .then(res => {
    //log res for testing
    console.log(res.data)
    if (res.data !== null) {
      alert("User already exists")
    }
  })
```

which sends a get request to the backend and if that request brings back data we know that a user with that key already exists otherwise it will allow the user to register and not return such a message.
The registration form also has form validation which are the following:

```
this.state.number.length === 10 &&
this.state.name.length >= 5 &&
this.state.password.length >= 6 && this.username !== null && dob != null
```

which state that the number length must be equal to ten, the name length must be greater than or equal to five, the password length must be greater than or equal to 6 and the username cannot be null if these requirements are not met the user will be met with a message saying "Form invalid, password length must be greater than 6 and number must have 10 digits and name must have 5 or more characters and dob cannot be null".
The backend code for the register function is listed below:

```
app.post("/api/users", function(req, res) {
  //check if user with same username exists use findById and change id to usern
  var balance;
```

```
if(req.body.balance == "")
{
  balance = 0
}
else{
  balance = req.body.balance
}
Users.create({
  _id: req.body._id,
  password: req.body.password,
  name: req.body.name,
  number: req.body.number,
  dob: req.body.dob,
  balance: balance,
  iban: "IE",
  bic:""
});
res.status(201, "Resource created");
});
```


Figure 5.2: Image of Register Component

Forgot

Loans

Home

Statistics

Transactions

headlines

Error

About

UserInfo

5.4 Backend Architecture

5.5 Database Architecture

5.5.1 MLAB

5.5.2 Mongoose

5.5.3 Schema of Database

5.6 Security

5.6.1 Overview of Security

Chapter 6

System Evaluation

As many pages as needed.

- Prove that your software is robust. How? Testing etc.

- Use performance benchmarks (space and time) if algorithmic.

- Measure the outcomes / outputs of your system / software against the objectives from the Introduction.

- Highlight any limitations or opportuni-ties in your approach or technologies used.

Chapter 7

Conclusion

About three pages.

Appendices

Appendix A

Preamble & Intro

- Link to my github

Bibliography

[1] A. Morgan, "The modern application stack – part 1: Introducing the mean stack." https://www.mongodb.com/blog/post/the-modern-application-stack-part-1-introducing-the-mean-stack.

[2] Educba, "Is mongodb open source?." https://www.educba.com/mongodb-open-source/.

[3] F. Hámori, "How developers use node.js - survey results." https://blog.risingstack.com/node-js-developer-survey-results-2016/.

[4] J. Rocheleau, "Mongodb for beginners: Introduction and installation (part 1/3)." https://www.hongkiat.com/blog/webdev-with-mongodb-part1/.

[5] CodeCondo, "Mean stack vs lamp stack." https://codecondo.com/mean-stack-vs-lamp-stack/.

[6] Dashlane, "The 3 simple reasons why dashlane uses react." https://blog.dashlane.com/the-3-simple-reasons-why-dashlane-uses-react/.

[7] W. various, "Node.js." https://en.wikipedia.org/wiki/Node.js.

[8] R. Sarreal, "History of online banking: How internet banking went mainstream." https://www.gobankingrates.com/banking/banks/history-online-banking/.

[9] B. O. Ireland, "Help for older people." https://www.bankofireland.com/security-zone/help-for-older-people/.

[10] J. Pilcher, "25 digital-only banks to watch." https://thefinancialbrand.com/69560/25-direct-online-digital-banks/.

[11] G. Blog, "Cyber bank robberies contribute to \$1 trillion in cybercrime losses." https://www.globalsign.com/en/blog/cyber-bank-robberies-contribute-to-1-trillion-in-cybercrime-losses/.

[12] Z. Doffman, "Cybercrime: 25% of all malware targets financial services, credit card fraud up 200%." https://www.forbes.com/sites/zakdoffman/2019/04/29/new-cyber-report-25-of-all-malware-hits-financial-services-card-fraud-up-20#44d8cd4b7a47s.

[13] various, "Manifesto for agile software development." https://agilemanifesto.org/.

[14] S. K. Magdalena Maneva, Natasa Koceska, "Measuring agility in agile methodologies." https://core.ac.uk/reader/149219742.

[15] SlideModel, "Agile process lifecycle diagram for powerpoint." https://slidemodel.com/templates/agile-process-lifecycle-diagram-powerpoint/.

[16] atlassian, "Kanban how the kanban methodology applies to software development." https://www.atlassian.com/agile/kanban.

[17] U. Kearns, "Kanban board on github." https://github.com/Ultan-Kearns/AppliedProject/projects/3.

[18] W. Aejmelaeus, "Test-driven development." https://core.ac.uk/reader/37986089.

[19] M. Warcholinski, "Test-driven development (tdd) – quick guide." https://brainhub.eu/blog/test-driven-development-tdd/.

[20] I. . S. . expressjs.com contributors, "Express documentation." https://expressjs.com/.

[21] Unknown/Multiple, "Reactjs." https://reactjs.org//.

[22] Numerous, "Axios github." https://github.com/axios.

[23] Unknown/multiple, "Node js." https://nodejs.org/en/about/.

[24] Numerous, "React helmet github." https://github.com/nfl/react-helmet.

[25] Numerous, "Create-react-app github." https://github.com/facebook/create-react-app.

[26] https://newsapi.org/, "Newsapi documentation." `https://newsapi.org/`.

[27] R. Fielding, "Roy fielding dissertation - rest." `https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm/`.

[28] Cloudflare.com, "Tls image." `https://blog.cloudflare.com/content/images/2016/09/TLS-1.3.007.png`.

[29] https://csrc.nist.gov/, "Descriptions of sha-256,sha-384,and sha-512." `https://csrc.nist.gov/csrc/media/publications/fips/180/2/archive/2002-08-01/documents/fips180-2.pdf`.

[30] https://mlab.com/, "Mlab website." `https://mlab.com/`.

[31] Mozilla, "Javascript documentation." `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference`.

[32] Mozilla, "Html documentation." `https://developer.mozilla.org/en-US/docs/Web/HTML`.

[33] Numerous, "Latex documentation." `https://www.latex-project.org/`.

[34] Google, "Google cloud platform." `https://cloud.google.com/`.

[35] Docker, "Docker website." `https://www.docker.com/`.