
Online Banking Application Using A MERN Stack

Ultan Kearns

B.Sc.(Hons) in Software Development

FEBRUARY 23, 2020

Final Year Project - Banking Application using MERN stack

Advised by: Dr. Dominic Carr

Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)



Contents

1	Introduction	8
1.1	Why A MERN stack?	8
1.1.1	Mongo	9
1.1.2	Express	10
1.1.3	React	11
1.1.4	Node JS	12
1.2	Why a banking application?	12
1.2.1	Explanation of Why I Chose This Project	12
1.2.2	My Contention With Existing Online Banking Systems & How I Plan To Improve Upon Them	13
1.3	Requirements	14
1.4	Outline of chapters	15
1.4.1	Context	15
1.4.2	Methodology	15
1.4.3	Technical Review	15
1.4.4	System Design	16
1.4.5	System Evaluation	16
1.4.6	Conclusion	16
1.5	Structure of Project	16
2	Context	17
2.1	Project Objectives	17
2.2	Online Banking	17
2.3	History of Online Banking	18
2.3.1	Definition of Online Banking	18
2.3.2	The Beginning	18
2.4	Advantages of Online Banking	19
2.4.1	Convenience	19
2.4.2	Speed	19
2.4.3	Competition	19
2.4.4	Creation of Jobs	19

2.5	Disadvantages of Online Banking	20
2.5.1	Security	20
2.5.2	Layoff of Bank Employees	20
2.5.3	Increased Crime	20
2.6	The Overall Effect of Online Banking	20
3	Methodology	21
3.1	Overview of Methodology	21
3.2	Agile	21
3.2.1	What is Agile?	21
3.2.2	Why Kanban?	21
3.2.3	How I Applied Agile To This Project	22
3.3	Test Driven Development (TDD)	23
3.3.1	What Is Test Driven Development?	23
3.3.2	How I Applied Test Driven Development To This Project	23
3.4	Time Management	24
3.4.1	How I Handled Multiple Projects	24
3.4.2	Issues I Faced With Time Management	24
3.5	Version Control	24
4	Technology Review	25
4.1	Intro	25
4.2	MERN Stack	25
4.2.1	About MongoDB	25
4.2.2	About Express	26
4.2.3	About React	27
4.2.4	About Node	27
4.3	Testing	28
4.3.1	Selenium	28
4.4	React Libraries	28
4.4.1	Axios	28
4.4.2	React Helmet	29
4.4.3	Create-React-App	29
4.5	Third Party APIs	29
4.5.1	NodeMailer	29
4.5.2	News API	29
4.5.3	Reddit	29
4.6	Architecture	30
4.6.1	REST(Representational State Transfer)	30
4.7	Security	30
4.7.1	OpenSSL	30

4.7.2	SHA256	31
4.8	Database	31
4.8.1	MLAB	31
4.9	Languages	32
4.9.1	Javascript	32
4.9.2	HTML(Hyper-Text Markup Language)	32
4.10	Documentation	33
4.10.1	L ^A T _E X	33
4.11	Styles	33
4.11.1	Cascading Style Sheets(CSS)	33
4.11.2	React Bootstrap	34
4.12	Cloud	34
4.12.1	Google Cloud Platform	34
4.12.2	Docker	34
5	System Design	36
5.1	Introduction	36
5.2	Architecture of Project	36
5.2.1	Introduction to System	36
5.2.2	Reasons for System Design	37
5.3	Front End Architecture	37
5.3.1	List of Components	37
5.4	Designing for UX(User Experience)	61
5.4.1	Designing For End-Users As Opposed to Engineers	61
5.5	Backend Architecture	61
5.5.1	Mongoose	61
5.5.2	ExpressJS	61
5.5.3	NodeJS	62
5.6	RESTful Architecture	62
5.6.1	Client / Server Independence	62
5.6.2	Cacheable	62
5.6.3	Stateless	63
5.7	Database Architecture	63
5.7.1	MLAB	63
5.7.2	Schema of Database	63
5.8	Security	63
5.8.1	Overview of Security	63
6	System Evaluation	66
7	Conclusion	67

<i>CONTENTS</i>	5
Appendices	68
A Preamble & Intro	69

List of Figures

1.1	MERN Logo	8
1.2	Mongo Usage	9
1.3	Mongo Logo	10
1.4	Express JS	11
1.5	React	12
1.6	Node	13
3.1	Agile	22
3.2	Image of Kanban board from wikipedia	22
3.3	TDD	23
3.4	Github Logo from github.com	24
4.1	TLS.	31
5.2	Image of Register Component	41
5.1	Image of Login Component	65

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 L^AT_EX 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**

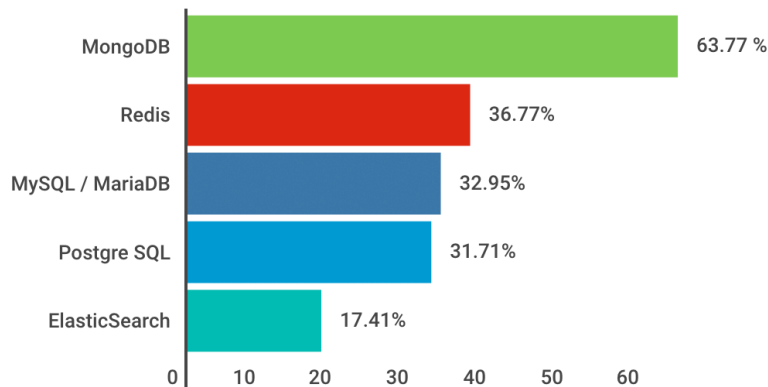
other methods , ReactJS for asynchronous JS & Node - For the server and also includes a package manager to install a variety of packages to provide certain functionalities. [1]

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



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 now 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 statements 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: statements, withdrawal dates etc.)
- **Graphs/Charts:** The banking app must display expenditures and credit in graphs and charts generated from python scripts
- **Emails** The application must be able to email the user a forgot password if they forgot their password and also 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 statements 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 on-line 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 referened 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 presence.^[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 occurring 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

This is a sample text. Insert
your desired text here.

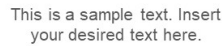


Figure 3.1: Agile

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.

[illegible]

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 Kan-
ban
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].

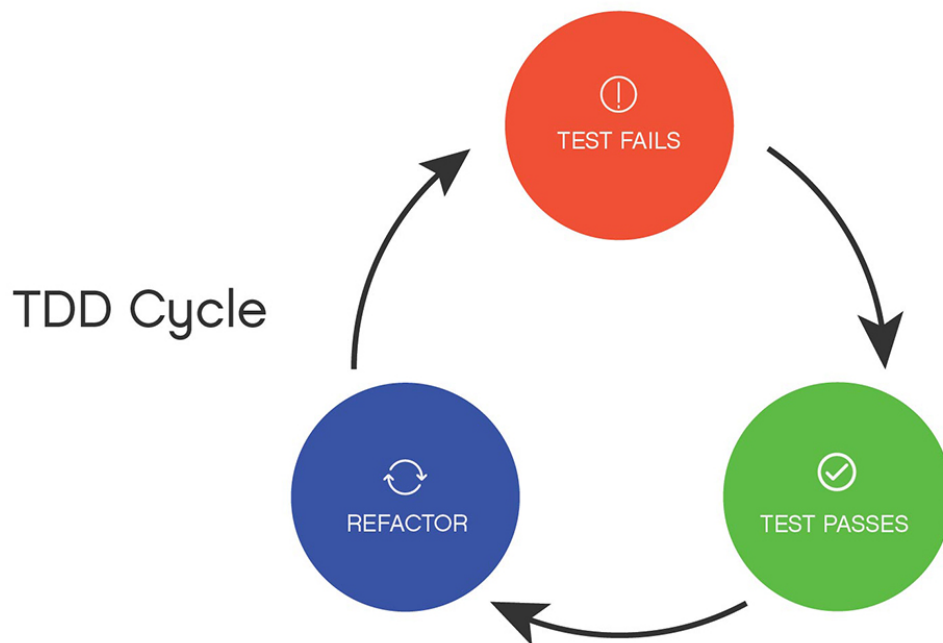


Figure 3.3: TDD

[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 where 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 banking 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 access 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:

```
1  const https = require("https");
2
3  app.get("/", function(req, res) {
4    res.status(200).send("Server is up and running!");
5  });
6  var server = app.listen(8080, function() {
7    var host = server.address().address
8    var port = server.address().port
9    console.log("Example app listening at
    ↪ http://{$server}/{port}:")
10 })
```

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 enhance the applications security. I used OpenSSL to generate a certificate for the Node server to provide an extra layer of security for the user.

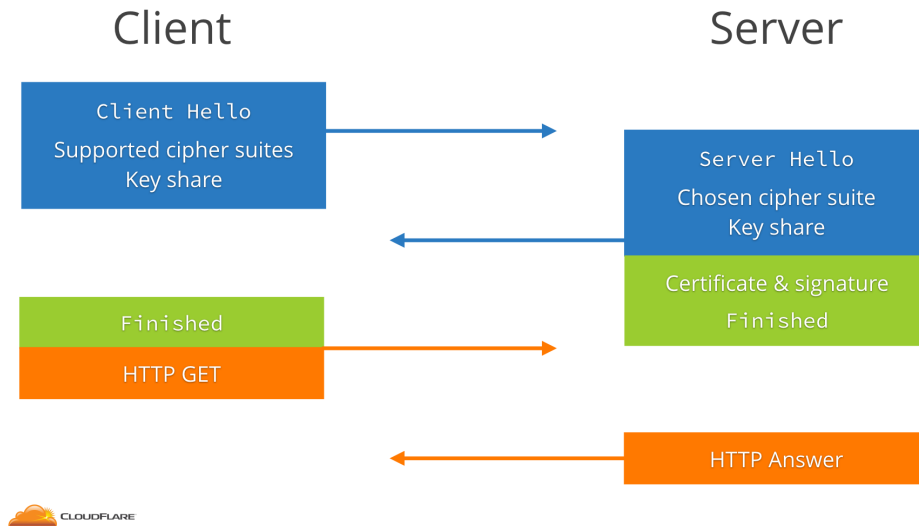


Figure 4.1: TLS.

[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 its 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:

```
1   name = "Ultan"
2   //prints Hello, Ultan
3   print("Hello, " + name);
4   a = 1
5   b = 2
6   //prints 3
7   print(a + b)
8   //prints Hello, Ultan1
9   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:

```
<!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 L^AT_EX

L^AT_EX is a typesetting system that provides various features for designing and writing high quality documentation[33]. L^AT_EX has become the de-facto standard for Academic publications and is open-source. Having never use L^AT_EX 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 L^AT_EX 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.

L^AT_EX 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 there 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 component is the component that the user first sees when starting the application. The component features an introduction which has white text on a black background 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 component" below this paragraph 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 component, 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 component's 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:

```

1  //template was taken from earlier project and refactored
2  //https://github.com/Ultan-Kearns/eCommerceApp/blob/master/BackEnd/Server.js
3  app.get("/api/users/:id/:password", function(req, res) {
4    Users.findById(req.params.id, function(err, data) {
5      if (err) {
6        //send back error 500 to show the server had internal
7        ↳ error
        res.status(500, "INTERNAL SERVER ERROR " + err);

```

```

8     return;
9   } else if (data != null) {
10    //compare user username and password to the username and
    ↪ password in DB
11    if (req.params.id == data._id && data.password ==
    ↪ req.params.password) {
12      res.json(data);
13      res.status(200, "User logged in!");
14    }
15    else {
16      res.json("null");
17      res.status(404, "User not found!");
18    }
19  }
20  });
21 })

```

I had the idea to reuse the code and refactor it after reading Andy Hunt and Dave Thomas' book "The Pragmatic Programmer"[36]. 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:

```

1  handleSubmitForm = event => {
2    const axios = require("axios").default;
3    const sha256 = require('js-sha256');
4    axios
5      .get(
6        "https://localhost:8080/api/users/" +
7        this.state.username.toLowerCase() +
8        "/" +
9        sha256(this.state.password)
10     )
11     .then(function(res) {
12       console.log("TEST LOGIN " + res.data);
13       //need to fix this so it shows error message
14       if (res.data !== "null") {
15         //store the username this will help the bank feel more
        ↪ personal
16         sessionStorage.setItem("username", res.data.name);

```

```

17         sessionStorage.setItem("email", res.data._id);
18         ReactDOM.render(<Home />,
19             ↪ document.getElementById("root"));
19     }
20     else if(res.data == "null"){
21         alert("Wrong username or password")
22     }
23     }).catch(error =>{
24         alert("Error logging in, check internet connection?")
25     });
26
27     console.log("Clicked")
28     event.preventDefault();
29 };

```

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 home-page 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.

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 component 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:

```

1  //check if user exists
2  axios
3    .get("https://localhost:8080/api/users/" +
      ↪   this.state.username)
4    .then(res => {
5      //log res for testing
6      console.log(res.data)
7      if (res.data !== null) {
8        alert("User already exists")
9      }
10   })

```

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:

```

1  this.state.number.length === 10 &&
2  this.state.name.length >= 5 &&
3  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:

```

1  app.post("/api/users", function(req, res) {
2    //check if user with same username exists use findById and
      ↪   change id to username
3    var balance;
4    if(req.body.balance == "")
5    {
6      balance = 0
7    }
8    else{
9      balance = req.body.balance
10   }
11   Users.create({

```



```

12     _id: req.body._id,
13     password: req.body.password,
14     name: req.body.name,
15     number: req.body.number,
16     dob: req.body.dob,
17     balance: balance,
18     iban: "IE",
19     bic: ""
20   });
21   res.status(201, "Resource created");
22 });

```

The above code is used to create a new user object and save it to the database, if successful it returns a status 201 which means that a new user has been created.

Register here by entering information below

Username: ulankesma@gmail.com

Password: ••••••••

Full Name: Name

Phone number: Number

Date of birth: dd / mm / yyyy

Back Register

Figure 5.2: Image of Register Component

Forgot

The forgot component is a simple component which contains a few headings for the user and an input which prompts the user to enter their username. The paragraph above the input field informs the user how to use the component. If the send email button is hit and the username field is not blank then the component sends two axios requests one which is a post that sends a random password of 20 characters to the backend which will then update the user's password, the generated password is of course encrypted using SHA256 for extra security, the get request then sends a request to the backend to email the user their new password in plaintext, here is the code below:

```

1  axios.post("https://localhost:8080/api/users/" +
    ↪ sessionStorage.getItem("email") +
    ↪ "/rand",rand).then(res=>{
2    console.log(res)
3  });
4  axios
5    .get("https://localhost:8080/api/emailuser/" +
    ↪ this.state.username + "/" + plaintext)
6    .then(res => {
7      console.log(res.data);
8    });
9  alert("email sent to " + this.state.username);
10  event.preventDefault();
11 };

```

The random password is generated by using JavaScript's random function to generate random numbers between one and ten as shown below:

```

1  var plaintext = ""
2  for(var i = 0; i < 20; i++)
3  {
4    //generate 20 random numbers for password
5    plaintext += Math.floor(10 * Math.random())
6  }
7  const hashed = sha256(plaintext)
8  alert("HASHED " + hashed)
9  const rand = {password: hashed}

```

once the password is generated I then store the hashed plaintext in a constant variable called rand as I need the plaintext to send to the user, the plaintext is sent over the BackEnd using NodeMailer and a Gmail account created for this project, the code was taken from an earlier project and was coded previously by myselfm here is the code below:

```

1  //template taken from earlier project
2  //https://github.com/Ultan-Kearns/eCommerceApp/blob/master/BackEnd/Server.js
3  //Improved upon in this project
4  app.get("/api/emailuser/:id/:password", function(req, res,
    ↪ next) {
5    Users.findById(req.params.id, function(err, data) {
6      if (data == null)
7        res.status(404, "User does not exist on this server",
    ↪ err);

```

```
8     else if (data._id == req.params.id) {
9         res.json(data);
10        console.log(data)
11        res.status(200, "User logged in!");
12        var nodemailer = require("nodemailer");
13        var transporter = nodemailer.createTransport({
14            host: "smtp.gmail.com", // hostname
15            service: "gmail",
16            auth: {
17                //login to email set up for this project
18                user: "reactproject19@gmail.com",
19                pass: "GMITreact19"
20            },
21            tls: {
22                ciphers: "SSLv3"
23            }
24        });
25        var mailOptions = {
26            //Setting up which account to use for seending emails
27            from: "reactproject19@gmail.com",
28            to: data._id,
29            subject: "Forgot Independent Banking password",
30            text: "Here is your password for Independent Banking: "
31            ↪ + req.params.password
32        };
33        //Send email or log errors if user doesn't exist
34        transporter.sendMail(mailOptions, function(error, info)
35            ↪ {
36            if (error) {
37                console.log(error);
38            } else {
39                console.log("Sent Email to address: " +
40                ↪ req.params.id);
41            }
42        });
43    } else {
44        console.log("EMAIL COULD NOT BE SENT");
45        res.json("error email not sent");
46    }
47 }
```

46 });

The email is encrypted and secured as per security standards

Loans

The loans component greets the user with a header which says "Apply & View Loans", the component is composed of an input box with a button stating "Apply for Loan" which when clicked will retrieve the number the user entered in the input box and send a request from the frontend to the backend and create a new loan if the loan is considered valid by the system. The component also has another header "List of Loans" which will list all the users current loans with a button that says "Pay Back" which when hit will pay back the users balance to the bank and create a statement detailing the paying back of the loan.

The loan if it is to be accepted must conform to the following stipulations:

- User must have 25% of the loan amount in their account for example to take out a loan of €100 the user must have a balance of €25 or greater.
- The user can have a maximum of five loans at any given time.
- The loan value must not exceed €500.
- The loan amount cannot equal €0 as it would be pointless.

If the loan stipulations are met then the frontend executes the following code:

```

1  if (
2    this.state.amount !== "" &&
3    answer === true &&
4    parseInt(this.state.amount * 0.25) <=
5      parseInt(sessionStorage.getItem("balance")) &&
6    sessionStorage.getItem("openLoans") < 5 &&
7    this.state.amount <= 500 &&
8    this.state.amount > 0
9  ) {
10   const newLoan = {
11     email: sessionStorage.getItem("email"),
12     amount: this.state.amount,
13     date: date,
14     owedTo: "Independent Banking",
15     status: "Open"

```

```
16     };
17     const newTransaction = {
18       email: sessionStorage.getItem("email"),
19       cost: this.state.amount,
20       location: "IndependentBanking.com",
21       name: sessionStorage.getItem("username"),
22       date: date
23     };
24     axios
25       .post("https://localhost:8080/api/transactions",
26         ↪ newTransaction)
27       .then(res => {
28         console.log(res);
29       });
30     const newBalance = {
31       balance:
32         parseInt(this.state.amount) +
33         parseInt(sessionStorage.getItem("balance"))
34     };
35     axios.post("https://localhost:8080/api/loans",
36       ↪ newLoan).then(res => {
37       console.log(res);
38     });
39     axios
40       .post(
41         "https://localhost:8080/api/users/" +
42         sessionStorage.getItem("email") +
43         "/balance",
44         newBalance
45       )
46       .then(res => {
47         console.log("TEST " + res);
48         axios
49           .get(
50             "https://localhost:8080/api/users/" +
51             sessionStorage.getItem("email")
52           )
53           .then(res => {
54             sessionStorage.setItem("balance", res.data.balance);
55             alert(
56               "loan approved\n New balance is: " +
```

```

55         sessionStorage.getItem("balance")
56     );
57     })
58     .catch(error => {
59         alert("Could not approve loan");
60     });
61     sessionStorage.setItem("openLoans", getOpenLoans());
62     this.getUserLoans();
63 });
64 } else {
65     alert(
66         "Loan aborted\n Reasons why this may happen:\nLoan cannot
        ↳ be null and user must have at least 25% of loan amount
        ↳ in balance\nUsers can only have 5 open loans at a
        ↳ time\n Loans must also be less than or equal to
        ↳ €500\nLoan must be greater than 0"
67     );
68 }
69 event.preventDefault();
70 };

```

Here I check if the loan stipulations are met, if they are then I create a new loan and a new transaction containing all pertinent information about this loan then via Axios I store this information on the backend. Following this I update the users balance to add the loan amount to it, if the stipulations are not met we send the user an error message.

After this code has been executed the component executes the `getLoans()` function which the component also executes when loaded which is used to get open loans, here is the code used to get the user loans down below:

```

1  getUserLoans() {
2      document.getElementById("loans").innerHTML = "";
3      axios
4          .get(
5              "https://localhost:8080/api/loans/" +
              ↳ sessionStorage.getItem("email")
6          )
7          .then(res => {
8              for (var i = 0; i < res.data.length; i++) {
9                  this.setState({
10                      _id: res.data[i]._id,
11                      amount: res.data[i].amount,

```

```

12         date: res.data[i].date,
13         status: res.data[i].status,
14         owedTo: res.data[i].owedTo
15     });
16     //create LI element then form statment then append to
17     ↪ LI then add to list
18     var node = document.createElement("LI");
19     var text = document.createTextNode(
20         "Amount: €" +
21         this.state.amount +
22         ", \tDate: " +
23         this.state.date +
24         " ,\tStatus: " +
25         this.state.status +
26         " ,\tOwed to: " +
27         this.state.owedTo
28     );
29     var buttonNode = document.createElement("Button");
30     buttonNode.textContent = "Pay Back";
31     buttonNode.id = "payButton";
32     //for repaying loans
33     var loanId = this.state._id;
34     var loanCost = this.state.amount;
35     //create new balance
36     const newBalance = {
37         balance: parseInt(
38             sessionStorage.getItem("balance") -
39             ↪ parseInt(this.state.amount)
40         )
41     };
42     buttonNode.addEventListener("click", function() {
43         //check if balance >= loanpayment
44         if (sessionStorage.getItem("balance") >= loanCost) {
45             axios
46                 .post(
47                     "https://localhost:8080/api/users/" +
48                     sessionStorage.getItem("email") +
49                     "/balance",
50                     newBalance
51                 )
52                 .then(res => {

```

```
51         sessionStorage.setItem("balance",
52             ↪ newBalance.balance);
53         alert(
54             "Loan repaid new balance is: " +
55             sessionStorage.getItem("balance")
56         );
57     axios
58         .delete(
59             "https://localhost:8080/api/loans/" +
60             sessionStorage.getItem("email") +
61             "/" +
62             loanId
63         )
64         .then(res => {
65             alert("Loan paid");
66         })
67         .catch(error => {
68             alert("error: " + error);
69         });
70     sessionStorage.setItem("openLoans",
71         ↪ getOpenLoans());
72     const newTransaction = {
73         email: sessionStorage.getItem("email"),
74         cost: -loanCost,
75         location: "IndependentBanking.com",
76         name: sessionStorage.getItem("username"),
77         date: date
78     };
79     axios
80         .post("https://localhost:8080/api/transactions",
81             ↪ newTransaction)
82         .then(res => {
83             console.log(res);
84         });
85     } else {
86         alert("Not enough money in account to repay loan");
87     }
88     node.id = "loan";
89     node.append(text);
```



```

89         node.append(buttonNode);
90         document.getElementById("loans").appendChild(node);
91         sessionStorage.setItem("openLoans", getOpenLoans());
92     }
93 },
94 )
95 .catch(error => {
96     alert("Could not get loans");
97 });
98
99 }

```

The `getUserLoans()` function creates a text node listing all pertinent information about the loan, it completes this by sending a get request via Axios to the backend and returns all loans on the user accounts. It also creates a button that is appended to each loan which will offer the user a way to pay back the loan, once clicked the loan is deleted and the user balance is decremented by the loan amount if and only if the user has the loan amount in their balance, if not they are informed that they cannot pay back the loan. The `getLoans()` function is used in two components Loans and UserInfo and is stored in a helper function, I used this to check if the user had open loans which is used when updating information and applying for loans here is the code below:

```

1  export function getOpenLoans() {
2      var openLoan = 0
3      sessionStorage.setItem("openLoans", openLoan)
4      const axios = require("axios").default;
5      //helper function to count open loans
6      axios
7          .get(
8              "https://localhost:8080/api/loans/" +
9              ↪ sessionStorage.getItem("email")
10         )
11         .then(res => {
12             for (var i = 0; i < res.data.length; i++) {
13                 if(res.data[i].status === "Open"){
14                     openLoan++;
15                 }
16                 sessionStorage.setItem("openLoans", openLoan)
17             }
18         }).catch(error => {

```

```

18     alert("error getting loans")
19   })
20   return openLoan
21 }

```

The function goes through all the users loans and checks for the status open then sets a sessionStorage item to the amount of open loans the user has on their account. It uses the user email as a parameter when communicating with the backend to ensure that the user is only judged by their own account information as opposed to anyone elses. It is in this way that the user accounts are kept independent.

Home

The home page is the first page that the user sees when they login to the application, it consists of a section to send money and another section listing the latest headlines from NewsAPI.org. The send money section is used to send money to other independent banking accounts using their username so for example to send money to the user ultankearns@gmail.com I would enter that email into the input box account ID and the amount to send into the box prompting the user to send that amount. The home page also lists the users current balance which is retrieved using an Axios get request to the backend.

The send money feature works as follows, the user inputs the values expected, if the user exists then update the current users value to equal their balance - cost sent then update the other users balance to equal their balance + cost sent and create transactions for both the user sending the money and the user receiving the money. Here is the code below:

```

1  if(this.state.accountId === sessionStorage.getItem("email"))
2  {
3    alert("Cannot send money to yourself ( ;-( )")
4    e.preventDefault()
5    return;
6  }
7  if (this.state.amount === "" || this.state.accountId === "") {
8    alert("The amount / account ID cannot be null, want to donate
    ↪ it to us? >;D");
9  } else {
10   var date = new Date();
11   //payer logic
12   const newTransaction = {

```

```
13     email: sessionStorage.getItem("email"),
14     cost: this.state.amount * -1,
15     location: "Online Banking Transfer to " +
16       ↪ this.state.accountId,
17     name: sessionStorage.getItem("username"),
18     date: date
19   };
20   //create transaction
21   axios
22     .post("https://localhost:8080/api/transactions",
23       ↪ newTransaction)
24     .then(res => {
25       console.log(res);
26     });
27   //update bal
28   const newBalance = {
29     balance:
30       parseInt(sessionStorage.getItem("balance")) -
31       parseInt(this.state.amount)
32   };
33   axios
34     .post(
35       "https://localhost:8080/api/users/" +
36       sessionStorage.getItem("email") +
37       "/balance",
38       newBalance
39     )
40     .then(res => {
41       console.log("TEST " + res);
42       axios
43         .get(
44           "https://localhost:8080/api/users/" +
45           sessionStorage.getItem("email")
46         )
47         .then(res => {
48           sessionStorage.setItem("balance", res.data.balance);
49         })
50         .catch(error => {
51           alert("Could not send money");
52         });
53     });
```

```
52
53 //payee logic
54 const payeeTransaction = {
55   email: this.state.accountId,
56   cost: this.state.amount,
57   location: "Online Banking Transfer from " +
58     ↳ sessionStorage.getItem("email"),
59   name: sessionStorage.getItem("username"),
60   date: date
61 };
62 //create transaction
63 axios
64   .post("https://localhost:8080/api/transactions",
65     ↳ payeeTransaction)
66   .then(res => {
67     console.log(res);
68   });
69 axios
70   .get("https://localhost:8080/api/users/" +
71     ↳ this.state.accountId)
72   .then(res => {
73     this.setState({
74       payeeBalance: parseInt(res.data.balance) +
75         ↳ parseInt(this.state.amount)
76     })
77   }).then(res =>{
78     const newBalance ={
79       balance: this.state.payeeBalance
80     }
81     axios
82       .post(
83         "https://localhost:8080/api/users/" +
84           this.state.accountId +
85           "/balance",
86         newBalance
87       )
88       .then(res => {
89         alert("Money Sent to: " + this.state.accountId + "
90           ↳ Amount sent: " + this.state.amount + " New
91           ↳ Balance: " + newBalance.balance)
92       });
93   });
```

```

87     })
88     .catch(error => {});
89   }
90   e.preventDefault();
91   };

```

The above feature contains the logic both for the payer and the payee.

To get the headlines and the balance was very simple to get the balance I simply performed an axios get request as we have seen before in other components, to get the headlines was slightly harder as I had to format the data and create elements using the DOM(Document object model) as shown below, which is located in a helper file:

```

1  axios
2    .get(
3
4      ↪ "https://newsapi.org/v2/top-headlines?country=us&category=business&apiKey=
5    )
6    .then(res => {
7      for (var i = 0; i < 3; i++) {
8        //create LI element then form statment then append to LI
9        ↪ then add to list
10       var node = document.createElement("LI");
11       node.id = "headline";
12       var text = document.createTextNode(
13         "Headline: " +
14         res.data.articles[i].title +
15         "Description: " +
16         res.data.articles[i].description +
17         "Author: " +
18         res.data.articles[i].author
19       );
20       var image = document.createElement("IMG");
21       image.src = res.data.articles[i].urlToImage;
22       image.alt = "Picture not available";
23       var link = document.createElement("A");
24       link.href = res.data.articles[i].url;
25       link.text = "Link to article";
26       node.append(text);
27       node.append(link);
28       node.append(image);
29       document.getElementById("homeFinance").appendChild(node);

```

```

28     }
29     ReactDOM.render(<App />, document.getElementById("root"));
30   })
31   .catch(error => {});

```

This basically retrieves the information and creates an element on the page which then displays all pertinent headlines to the user.

Statistics

The statistics page shows the user information pertaining to the stock market which is retrieved using the alphavantage API. The alphavantage API has a five calls per minute limitation for free users so if the max call limit is reached the application informs the user by displaying a message. This component also utilizes Chart.JS which is a JavaScript library to create different types of mathematical charts so that the user can see a visual aid to view their spending habits. Every time a loan is taken out or there is any activity on the users account the charts will update.

The information about the stock market is retrieved using an Axios get request and creating text elements via the DOM which we have seen used in previous components here is the code below:

```

1  axios
2    .get(
3
4      ↪ "https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=MSF
5    )
6    .then(res => {
7      var d = res.data["Meta Data"]["3. Last Refreshed"];
8      d = d.split(" ")[0];
9      var node = document.createElement("LI");
10     node.id = "stockresults";
11
12     var text = document.createTextNode(
13       "Stats for today: Open: " +
14       res.data["Time Series (Daily)"][d]["1. open"] +
15       " High: " +
16       res.data["Time Series (Daily)"][d]["2. high"] +
17       " Low: " +
18       res.data["Time Series (Daily)"][d]["3. low"] +
19       " Close: " +
20       res.data["Time Series (Daily)"][d]["4. close"] +

```

```

20         " Volume: " +
21         res.data["Time Series (Daily)"][d]["5. volume"]
22     );
23     node.append(text);
24     document.getElementById("stock").appendChild(node);
25 })
26 .catch(error => {
27 alert("problem getting currency data")
28 });
29 axios
30 .get(
31
32     ↪ "https://www.alphavantage.co/query?function=CURRENCY_EXCHANGE_RATE&from_
33 )
34 .then(res => {
35     var eur_to_btc =
36     "1 Euro = " +
37     res.data["Realtime Currency Exchange Rate"]["5. Exchange
38     ↪ Rate"] +
39     "BTC";
40     var btcNode = document.createElement("p");
41     btcNode.append(eur_to_btc);
42     document.getElementById("eurbtc").appendChild(btcNode);
43 })
44 .catch(function(error) {
45     var node = document.createElement("LI");
46     node.id = "currency";
47     var text = document.createTextNode(
48     "Sorry something went wrong while getting the currency
49     ↪ value data :*(("
50 );
51     node.append(text);
52     document.getElementById("eurbtc").appendChild(node);
53     console.log("error");
54 });
55 axios
56 .get(
57
58     ↪ "https://www.alphavantage.co/query?function=CURRENCY_EXCHANGE_RATE&from_
59 )
60 .then(res => {

```

```

57     var eur_to_sterling =
58         "1 Euro = " +
59         res.data["Realtime Currency Exchange Rate"]["5. Exchange
        ↳ Rate"] +
60         "GBP";
61     var sterlingNode = document.createElement("p");
62     sterlingNode.append(eur_to_sterling);
63
        ↳ document.getElementById("eursterling").appendChild(sterlingNode);
64 })
65 .catch(function(error) {
66     var node = document.createElement("LI");
67     node.id = "currency";
68     var text = document.createTextNode(
69         "Sorry something went wrong while getting the currency
        ↳ value data :*(("
70     );
71     node.append(text);
72     document.getElementById("eursterling").appendChild(node);
73
74     console.log("error");
75 });
76 axios
77 .get(
78
        ↳ "https://www.alphavantage.co/query?function=CURRENCY_EXCHANGE_RATE&from_
79 )
80 .then(res => {
81     var eur_to_dollar =
82         "1 Euro = " +
83         res.data["Realtime Currency Exchange Rate"]["5. Exchange
        ↳ Rate"] +
84         "USD";
85     var dollarNode = document.createElement("p");
86     dollarNode.append(eur_to_dollar);
87
        ↳ document.getElementById("eurdollar").appendChild(dollarNode);
88 })
89 .catch(function(error) {
90     var node = document.createElement("LI");
91     node.id = "currency";

```



```

92     var text = document.createTextNode(
93         "Sorry something went wrong while getting the currency
          ↳ value data :*(("
94     );
95     node.append(text);
96     document.getElementById("eurdollar").appendChild(node);
97
98     console.log("error");
99 });
100 axios
101     .get(
102
          ↳ "https://www.alphavantage.co/query?function=CURRENCY_EXCHANGE_RATE&from_
103     )
104     .then(res => {
105         var eur_to_cny =
106             "1 Euro = " +
107             res.data["Realtime Currency Exchange Rate"]["5. Exchange
          ↳ Rate"] +
108             "CNY";
109         var cnyNode = document.createElement("p");
110         cnyNode.append(eur_to_cny);
111         document.getElementById("eurcny").appendChild(cnyNode);
112     })
113     .catch(error => {
114         var node = document.createElement("LI");
115         node.id = "currency";
116         var text = document.createTextNode(
117             "Sorry something went wrong while getting the currency
          ↳ value data :*(("
118         );
119         node.append(text);
120         document.getElementById("stock").appendChild(node);
121         console.log("error");
122     });
123 } catch (error) {
124     alert("MAX API CALLS FOR FINANCIAL DATA REACHED");
125 }
126 }

```

The above code just retrieves information from the API and formats it in a

way that is readable to the user the code also includes error handling in case of a network error or API error.

The code for the data chart just retrieves information about the user account by using axios get requests to get the transactions and loans on the user account which I will show below:

```
1  async updateLoanData() {
2    await axios
3      .get(
4        "https://localhost:8080/api/loans/" +
        ↪  sessionStorage.getItem("email")
5      )
6      .then(res => {
7        console.log(res.data);
8        var newLoanState = this.state.loanState;
9        for (var i = 0; i < res.data.length; i++) {
10          newLoanState.datasets[0].data[i] = res.data[i].amount;
11          newLoanState.labels[i] = res.data[i].amount;
12        }
13        this.setState({
14          loanState: newLoanState
15        });
16      })
17      .catch(function(error) {
18        alert("Error generating loans " + error);
19      });
20  }
21  async updateTransactionData() {
22    return axios
23      .get(
24        "https://localhost:8080/api/transactions/" +
25          sessionStorage.getItem("email")
26      )
27      .then(res => {
28        var newTransactionState = this.state.transactionState;
29        for (var i = 0; i < res.data.length; i++) {
30          newTransactionState.datasets[0].data[i] =
            ↪  res.data[i].cost;
31          newTransactionState.labels[i] = res.data[i].location;
32        }
33        this.setState({
```

```

34         transactionState: newTransactionState
35     });
36 }
37 .catch(function(error) {
38     alert("Error generating transactions " + error);
39 });
40 }

```

The above code just feeds the retrieved data into a variable called `transactionState` and `loanState` depending on which method is called.

Transactions

The transactions component lists transactions made on this users account which means it will list every transfer, loan or loan repayment made on the users account. It accomplishes this by using an axios get request to the backend and utilizing the user email to return all there transactions. Here is the code for requesting the transactions below:

```

1  componentDidMount() {
2      const axios = require("axios").default;
3      //use email instead
4      axios
5          .get(
6              "https://localhost:8080/api/transactions/" +
7              sessionStorage.getItem("email")
8          )
9          .then(res => {
10             for (var i = 0; i < res.data.length; i++) {
11                 this.setState({
12                     location: res.data[i].location,
13                     cost: res.data[i].cost,
14                     name: res.data[i].name,
15                     date: res.data[i].date,
16                     email: res.data[i].email
17                 });
18                 //create LI element then form statment then append
19                 ↪ to LI then add to list
20                 var node = document.createElement("LI");
21                 var text = document.createTextNode(
22                     "Location: " +
23                     this.state.location +

```

```

23         ", Cost: €" +
24         this.state.cost +
25         ", Name: " +
26         this.state.name +
27         ", Date: " +
28         this.state.date
29     );
30     node.id = "transaction"
31     node.append(text);
32
33     ↪ document.getElementById("transactions").appendChild(node);
34 }
35 }).catch(error =>{
36     alert("Can't get transactions issue connecting to
37     ↪ server")
38 });
39 }

```

Once retrieved each transaction is added to the transactions list as a child and is viewable by the user.

headlines

The headlines component gets the latest headlines from reddit and NewsAPI and returns them to the user the headlines component utilizes the same helper method as in home but this component has a lot more stories and also retrieves the latest headlines from Reddit /r/wallstreet which are retrieved utilizing the Reddit API as demonstrated in the code below:

```

1  axios
2    .get(
3      "https://www.reddit.com/r/wallstreet/.json"
4    )
5    .then(res => {
6      for (var i = 0; i < res.data.data.children.length; i++) {
7        var node = document.createElement("LI");
8        node.id = "reddit_headlines";
9        var text = document.createTextNode(
10          "Headline: " + res.data.data.children[i].data.title +
11          ↪ " " + res.data.data.children[i].data.selftext
12        );

```

```
12     var link = document.createElement("A");
13     link.href = res.data.data.children[i].data.url
14     link.text = "Link to article";
15     var image = document.createElement("IMG");
16     image.src = res.data.data.children[i].data.thumbnail
17     image.alt = "Picture not available";
18     image.height =
19     ↪ res.data.data.children[i].data.thumbnail_height
20     image.width =
21     ↪ res.data.data.children[i].data.thumbnail_width
22     node.append(text)
23     node.append(link)
24     node.append(image)
25     document.getElementById("reddit").appendChild(node);
26 }
27 }).catch(error => {
    alert("Having issues getting your news from reddit")
});
```

This is basically the same function as we have seen from the helper method in the home component with the exception that it has different field names as opposed to newsapi.org, but essentially they function the exact same way.

Error

The error page is a simple page which handles errors which may occur when the user tries to access a resource that may not exist and it returns a message stating to the user "Sorry this page cannot be found" and also has a header which says "Error 404", a 404 error is returned when the server cannot find a requested resource.

About

The about section just displays information about the site to the user and features no backend logic or axios requests its sole purpose is to make the bank appear as if it were a real entity instead of an imaginary business which only exists in the mind of a college student. Of course all businesses start as imaginary entities and through hard work and effort become real viable businesses.

UserInfo

5.4 Designing for UX(User Experience)

5.4.1 Designing For End-Users As Opposed to Engineers

5.5 Backend Architecture

5.5.1 Mongoose

Mongoose is a modelling tool for MongoDB and I utilized it on the backend to create schemas, create objects, read data, update data and delete data. Mongoose was very useful in communicating with the database and allowed me to create custom features which could be utilized to search for various objects using their parameters. All requests made to the database were made using Mongoose to achieve a full CRUD(Create, Read, Update & delete) application.

5.5.2 ExpressJS

ExpressJS provided a thin layer which sat on top of NodeJS and provided me with desirable features such as saving me the effort of specifying the full URL of the server on the backend and instead using the relative path to the resources I needed. It also provided a body parser so I could return JSON(JavaScript Object Notation) objects which made manipulating data coming from the server incredibly easy, and in this way I was able to provide the user with the pertinent information they needed and format said information easily and effectively.

5.5.3 NodeJS

NodeJS provided a framework to build the server-side architecture and made creating and designing the BackEnd for this application both easy and enjoyable. NodeJS has various benefits which it provided to the application such as being open source, cross-platform and it utilizes JavaScript which is one of the most popular languages for web-based development, which means various programmers can work and improve upon my original design.

5.5.4 Full Backend Code

```

1  var express = require("express");
2  var app = express();
3  //this ensures i don't have to write full address and can use
   ↪ relative paths for URL
4  var path = require("path");
5  //parses response
6  var bodyParser = require("body-parser");
7  //api to communicate with DB
8  var mongoose = require("mongoose");
9  //mongoDB link to connect
10 //3kCeKdq4iZWqlg00 this is for mongoatlas may migrate
11 var mongoDB =
12
   ↪ "mongodb://ultan:ultanultan1@ds135107.mlab.com:35107/appliedproject";
13 var Schema = mongoose.Schema;
14 var router = express.Router();
15 app.use(bodyParser.json());
16 //need this for some browsers to allow cross origin request
   ↪ to allow app to communicate with server
17 app.use(function(req, res, next) {
18   //to allow cross origin requests
19   res.header("Access-Control-Allow-Origin", "*");
20   res.header("Access-Control-Allow-Methods", "GET, POST, PUT,
   ↪ DELETE, OPTIONS");
21   res.header(
22     "Access-Control-Allow-Headers",
23     "Origin, X-Requested-With, Content-Type, Accept"
24   );
25   next();
26 });
27 const fs = require("fs");
28 //add https support
29 const https = require("https");
30 const keysDirectory = "./";
31 const cors = require("cors");
32 app.use(cors({ credentials: true, origin: true }));
33 const security = {
34   //read files for ssl connection - keys are generated with
   ↪ openssl

```

```

35  //password for decryption - 123456789 Not very secure I
    ↪ know but can't remember harder passwords
36  key: fs.readFileSync(keysDirectory + "ca.key"),
37  cert: fs.readFileSync(keysDirectory + "cert.crt")
38  };
39  //use mongoose API to connect to backend
40  mongoose.connect(mongoDB, { useUnifiedTopology: true,
    ↪ useNewUrlParser: true });
41  //body parser for middleware
42  app.use(
43    bodyParser.urlencoded({
44      extended: false
45    })
46  );
47  app.use(bodyParser.json());
48  //change later
49  var userSchema = new Schema({
50    _id: { type: String },
51    password: { type: String, required: true },
52    name: { type: String, required: true },
53    number: { type: String, required: true },
54    dob: { type: Date, required: true },
55    balance: { type: Number, default: 0, required: true },
56    iban: { type: String },
57    bic: { type: String },
58  });
59  //change this later
60  var transactionSchema = new Schema({
61    location: { type: String, default: "Unknown" },
62    cost: { type: Number, default: 0 },
63    name: { type: String },
64    date: { type: Date },
65    email: { type: String }
66  });
67  transactionSchema.methods.findName = function(username) {
68    //define logic to find statement by name in here
69    return this.model("Transactions").find({ email: this.email
    ↪ }, username);
70  };
71  transactionSchema.methods.findNameAndDelete =
    ↪ function(username) {

```



```
72     //define logic to find statement by name in here
73     return this.model("Transactions").remove({ email: this.email
74         ↪ }, username);
75 };
76 var loanSchema = new Schema({
77     email: { type: String },
78     amount: { type: Number },
79     date: { type: String },
80     owedTo: { type: String },
81     status: { type: String }
82 });
83 loanSchema.methods.findName = function(username) {
84     //define logic to find statement by name in here
85     return this.model("Loans").find({ email: this.email },
86         ↪ username);
87 };
88 loanSchema.methods.findNameAndDelete = function(username) {
89     //define logic to find statement by name in here
90     return this.model("Loans").remove({ email: this.email },
91         ↪ username);
92 };
93 //models for mongoose
94 var Users = mongoose.model("Users", userSchema);
95 var Transactions = mongoose.model("Transactions",
96     ↪ transactionSchema);
97 var Loans = mongoose.model("Loans", loanSchema);
98 var transaction = new Transactions();
99 var loan = new Loans();
100 //Here I will use get requests to retrieve resources
101 app.get("/", function(req, res) {
102     res.status(200).send("Server is up and running!");
103 });
104 app.get("/api/users", function(req, res) {
105     Users.find(function(err, data) {
106         res.json(data);
107         res.status(200, "request completed");
108     });
109 });
110 app.get("/api/transactions", function(req, res) {
111     Transactions.find(function(err, data) {
112         res.json(data);
```

```

109     res.status(200, "request completed");
110   });
111 });
112 app.get("/api/loans", function(req, res) {
113   Loans.find(function(err, data) {
114     res.json(data);
115     res.status(200, "request completed");
116   });
117 });
118 //template was taken from earlier project and refactored
119   ↪ https://github.com/Ultan-Kearns/eCommerceApp/blob/master/BackEnd/Server.js
120 app.get("/api/users/:id/:password", function(req, res) {
121   Users.findById(req.params.id, function(err, data) {
122     if (err) {
123       //send back error 500 to show the server had internal
124       ↪ error
125       res.status(500, "INTERNAL SERVER ERROR " + err);
126       return;
127     } else if (data != null) {
128       //compare user username and password to the username and
129       ↪ password in DB
130       if (req.params.id == data._id && data.password ==
131         ↪ req.params.password) {
132         res.json(data);
133         res.status(200, "User logged in!");
134       }
135       else {
136         res.json("null");
137         res.status(404, "User not found!");
138       }
139     }
140   });
141 });
142 //template taken from earlier project -
143   ↪ https://github.com/Ultan-Kearns/eCommerceApp/blob/master/BackEnd/Server.js
144 //Improved upon in this project
145 app.get("/api/emailuser/:id/:password", function(req, res,
146   ↪ next) {
147   Users.findById(req.params.id, function(err, data) {
148     if (data == null)

```

```
143     res.status(404, "User does not exist on this server",
    ↪     err);
144   else if (data._id == req.params.id) {
145     res.json(data);
146     console.log(data)
147     res.status(200, "User logged in!");
148     var nodemailer = require("nodemailer");
149     var transporter = nodemailer.createTransport({
150       host: "smtp.gmail.com", // hostname
151       service: "gmail",
152       auth: {
153         //login to email set up for this project
154         user: "reactproject19@gmail.com",
155         pass: "GMITreact19"
156       },
157       tls: {
158         ciphers: "SSLv3"
159       }
160     });
161     var mailOptions = {
162       //Setting up which account to use for seending emails
163       from: "reactproject19@gmail.com",
164       to: data._id,
165       subject: "Forgot Independent Banking password",
166       text: "Here is your password for Independent Banking: "
    ↪       + req.params.password
167     };
168
169     //Send email or log errors if user doesn't exist
170     transporter.sendMail(mailOptions, function(error, info)
    ↪     {
171       if (error) {
172         console.log(error);
173       } else {
174         console.log("Sent Email to address: " +
    ↪         req.params.id);
175       }
176     });
177   } else {
178     console.log("EMAIL COULD NOT BE SENT");
179     res.json("error email not sent");
```

```
180     }
181   });
182 });
183 app.get("/api/users/:id/", function(req, res) {
184   Users.findById(req.params.id, function(err, data) {
185     if (err) {
186       //send back error 500 to show the server had internal
187       ↪ error
188       res.status(500, "INTERNAL SERVER ERROR " + err);
189     } else if (data != null) {
190       res.json(data);
191     }
192     else{
193       res.json("null")
194     }
195   });
196 });
197 app.delete("/api/users/:id/", function(req, res) {
198   Users.findByIdAndRemove(req.params.id, function(err, data) {
199     if (err) {
200       //send back error 500 to show the server had internal
201       ↪ error
202       res.status(500, "INTERNAL SERVER ERROR " + err);
203     } else if (data != null) {
204       res.status(200, "Deleted Account")
205     }
206   });
207 });
208 app.delete("/api/loans/:email/:id", function(req, res) {
209   Loans.findByIdAndRemove(req.params.id, function(err, data) {
210     if (err) {
211       //send back error 500 to show the server had internal
212       ↪ error
213       res.status(500, "INTERNAL SERVER ERROR " + err);
214     } else if (data != null) {
215       res.status(200, "Paid loan")
216     }
217   });
218 });
219 app.post("/api/users", function(req, res) {
```

```

217    //check if user with same username exists use findById and
      ↪ change id to username
218    var balance;
219    if(req.body.balance == "")
220    {
221        balance = 0
222    }
223    else{
224        balance = req.body.balance
225    }
226    Users.create({
227        _id: req.body._id,
228        password: req.body.password,
229        name: req.body.name,
230        number: req.body.number,
231        dob: req.body.dob,
232        balance: balance,
233        iban: "IE",
234        bic:""
235    });
236    res.status(201, "Resource created");
237 });
238 app.put("/api/users/:id", function(req, res) {
239     /*
240     Decided to keep this as it may be useful if db is
      ↪ refactored
241
      ↪ Users.findByIdAndUpdate(req.params.id,{name:req.body.name,password:req.body
242     if (err) {
243         //send back error 500 to show the server had internal
      ↪ error
244         res.status(500, "INTERNAL SERVER ERROR " + err);
245     } else if (data != null) {
246         res.status(200,"Updated Account")
247     }
248 })
249 */
250 });
251 app.post("/api/users/:id/rand", function(req, res) {
252     ↪ Users.findByIdAndUpdate(req.params.id,{password:req.body.password},functi

```

```
253     if (err) {
254         //send back error 500 to show the server had internal
           ↪ error
255         res.status(500, "INTERNAL SERVER ERROR " + err);
256     } else if (data != null) {
257         res.status(200, "Updated Password ")
258     }
259 })
260 });
261 app.post("/api/users/:id/balance", function(req, res) {
262     Users.findByIdAndUpdate(req.params.id, {balance:
           ↪ req.body.balance}, function(err, data){
263         if (err) {
264             //send back error 500 to show the server had internal
               ↪ error
265             res.status(500, "INTERNAL SERVER ERROR " + err);
266         } else if (data != null) {
267             res.status(200, "Updated balance")
268             res.json(data);
269         }
270     })
271 });
272 app.post("/api/loans", function(req, res) {
273     Loans.create({
274         email: req.body.email,
275         amount: req.body.amount,
276         date: req.body.date,
277         status: req.body.status,
278         owedTo: req.body.owedTo
279     });
280     res.status(201, "Resource created");
281 });
282 app.post("/api/transactions", function(req, res) {
283     Transactions.create({
284         email: req.body.email,
285         cost: req.body.cost,
286         location: req.body.location,
287         name: req.body.name,
288         date: req.body.date
289     });
290     res.status(201, "Resource created");
```

```

291 });
292 transaction.findName(function(name) {
293     console.log(name);
294 });
295
296 app.get("/api/transactions/:email", function(req, res) {
297     //custom method to find name on transactions
298     transaction = new Transactions({ email: req.params.email });
299     transaction.findName(function(err, data) {
300         res.json(data);
301     });
302 });
303 app.delete("/api/transactions/:email", function(req, res) {
304     Transactions.remove({email: req.params.email}, function(err,
305         ↪ data) {
306         res.json(data);
307     });
308 });
309 app.delete("/api/loans/:email", function(req, res) {
310     Loans.remove({email: req.params.email},function(err, data) {
311         res.json(data);
312     })
313 });
314 app.post("/api/transactions/:email/:newemail", function(req,
315     ↪ res) {
316     ↪ Transactions.updateMany({email:req.params.email},{ $set:{email:req.params
317         res.json(doc)
318     });
319     ↪ });
320     ↪ app.post("/api/loans/:email/:newemail", function(req, res) {
321     ↪ ↪ Loans.updateMany({email:req.params.email},{ $set:{email:req.params.newemail
322         res.json(doc + " " + req.params.newemail)
323     })
324     ↪ });
325     ↪ app.get("/api/loans/:email", function(req, res) {
326     ↪ ↪ //custom method to find name on transactions
327     ↪ ↪ loan = new Loans({ email: req.params.email });
328     ↪ ↪ loan.findName(function(err, data) {

```

```
328     res.json(data);
329   });
330 });
331 //have server listening at port 8080 and have it take
   ↪ keycert to secure server
332 //uses Secure Socket Layer
333 https.createServer(security, app).listen(8080);
```

5.6 RESTful Architecture

5.6.1 Client / Server Independence

To ensure that RESTful architecture standards were met I had to keep the client and the server independent. I did this by making use of the AXIOS HTTP client which I used to send requests from the client side to the server side, in this way I ensured that the server and the client were completely independent of each other. Axios acted as a middle layer which was used whenever the client needed to communicate with the server. Axios returned responses, sent GET, POST, PUT, DELETE and other HTTP requests to the server and returned results from numerous APIS which I used in this project.

5.6.2 Cacheable

I made use of sessionStorage on the client machine to perform various back-end logic and used this stored information to pass parameters to Axios so that it could retrieve user information, perform authentication, send information about the user to the server, update information and create new resources. In addition to caching user information on the client machine the entire website is also cacheable, as it is a single page application and once loaded the user does not have to wait for pages to load as thanks to AJAX(Asynchronous JavaScript and XML) the requested resources are loaded almost instantaneously.

5.6.3 Stateless

As mentioned in the cacheable section I made great use of sessionStorage to store information on the client machine, this information was then used in axios requests when communicating with the server, the server did not maintain a user session as the server and client were entirely independent of

each other. If the server went down the users application would still load but the user would not be able to login or perform any backend functions such as sending HTTP requests. Statelessness is an important part of RESTful architecture and I did my utmost to achieve and implement it into this project so as to create a fully RESTful application.

5.7 Database Architecture

5.7.1 MLAB

MLAB is a DAAS or Database As A Service application which I used to store all the information about users or any other information generated by the application. The Database was accessed using Mongoose on the backend. The database is encrypted and secure and is used by various companies such as Toyota, Facebook & Lyft just to name a few. I chose MLAB for this project in order to make the database more secure and to imitate a real-life application as very few companies own their own servers to host their data, many companies off for IAAS & DAAS as they are far more scalable and inexpensive both in terms of time & cost.

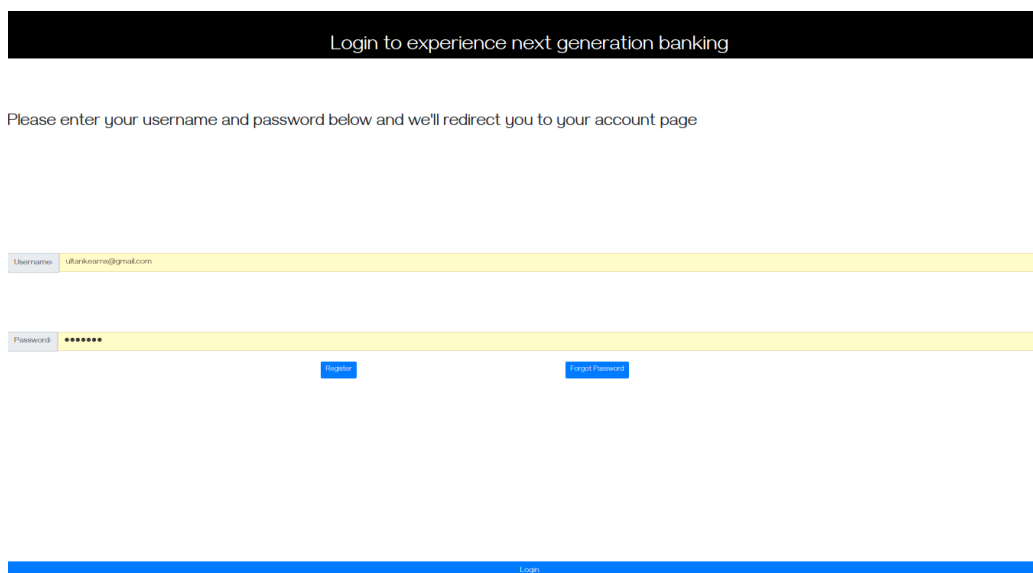
5.7.2 Schema of Database

5.8 Security

5.8.1 Overview of Security

The application is secured using HTTPS(HyperText Transfer Protocol Secure) which encrypts connections to the application and the server is secured using SHA256(Secure Hashing Algorithm) and the users passwords are encrypted with SHA256 as well. The application complies with the most up to date cyber-security principles and offers the user protection from hackers and implements security features expected of a modern online banking application.

The certificates I used on the server were generated using OpenSSL, both the application and the server use self-signed certificates which will generate security warnings in Chrome and Firefox as the browser doesn't know if the certificates were verified by a reputable source, in a real world deployment I would have used GoDaddy or another service to verify the certificates.



The image shows a login component with a black header bar containing the text "Login to experience next generation banking". Below the header, a message reads "Please enter your username and password below and we'll redirect you to your account page". There are two input fields: "Username" with the value "ulankar@gmail.com" and "Password" with masked characters "*****". Below the password field are two buttons: "Register" and "Forgot Password". At the bottom of the form is a blue bar with the text "Login".

Figure 5.1: Image of Login Component

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 opportunities 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-200/#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. Aejmelaesus, “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/>.
- [36] A. H. . D. Thomas, “Pragmatic programmer.” https://www.goodreads.com/book/show/4099.The_Pragmatic_Programmer.