RE-Agent Dashboard

James Porter

B.Sc.(Hons) in Software Development

May 10, 2021

Final Year Project

Advised by: Dr Gerard Harrison

Department of Computer Science and Applied Physics

Galway-Mayo Institute of Technology (GMIT)

Contents

List of Figures

About this project

Abstract Within the field of real estate agency, agents are expected to manage a plethora of client and personal data and generally this is performed through paper based written formats such as journals or diaries. This often leads to the loss of work-related information and can cause issues for this working class of people who are constantly on the move and dealing with new client's home information regularly. As such many of the current online solutions provided for this issue within the field are expensive or do not work in a uniformed way, such that the real estate agents are often keeping track of different pieces of information on various free to use online services such as Google Calendar and Trello. This segregation of their information is inefficient and does not offer much upon the original paper based format in terms of data loss and confusion.
Hence this project is proposed as a potential solution to this data management issue and will allow real estate agents to store their real estate related data upon a single web application. This application will be easy to use and have support for their desktop and mobile devices and allow them to keep track of their information in an easy to use uniformed format upon a cloud based service.

Authors This project was produced by James Porter , a final year student of the Hon. Bachelor's Degree in Software Development Course provided by the Galway Mayo Institute of Technology's (G.M.I.T) Galway Campus.

Acknowledgements I would like to acknowledge Dr. Gerard Harrison for he's time and advice throughout the production process of the Applied Project and Dissertation.

GitHub Repository Link `https://github.com/JamesP1996/RE-Agent-Dashboard`

Chapter 1

Introduction

1.1 Context

During the initial planning phase of this project, it was ideal to come up with a project idea that would further my skills as a professional software engineer and would be able to offer myself relevant experience in modern full stack technologies, project planning, methodologies and common place organizational practices. This project would need to establish skills which I had acquired over my the course of my education while also offering newly learned experiences and technical skills. It would require to offer these experiences whilst staying within the scope of a Final Year Software Development Major Project.
The idea for the applied project, which is a Real Estate helper application known as **RE-Agent Dashboard**, was decided upon after a conversation with a professional real estate agent named *Carissa Jurina*[1] who works at *Berkshire Hathaway Home Services* within the **United States**. This conversation entailed on the topic of displeasure that her and her colleagues were facing when organizing their information that was coming from multiple sources into different applications or physical formats such as paper written documents and how it would be more plausible to have a single web-based application where they could organize all of this data, such that her and her fellow workers could be more efficient when answering queries from clients or staying on-top of their schedules and organizing their work plans.
It was felt that this project idea would hit a niche market where an application like it was absent for cost effective public use while also offering the experience of trying to solve an industry level problem in a non industry partner based project for the college using the problem-solving and practical software development skills I have accumulated over the course of my 4-year

degree. By having a project based discussion with a professional who works in the Real Estate Field, it would be more ideal to set up the requirements for the application and plan around the concerns of my target audience thus being more tactful in my approach to the issue rather than developing for a market I am not highly educated in. After deciding the project's area of interest, it was commonplace to have weekly discussions with my supervisor *Dr. Gerard Harrison* to ensure that the project remained on the right track and stayed within scope without becoming under or overwhelming and to get advice throughout the development process. Although I would not have the funding to release this project to a mass audience as a commercial product, the skills I have acquired during its development are invaluable and helped solidify the skills I needed to tackle an industry level problem domain.

1.1.1 Objectives of the Project

As previously mentioned, the goal of this project is to make an application that could be use able by Real Estate Agents in their professional field for data management. This application would need to reach certain requirements to be worth the target audience's time while also being worthy of a 15-credit module at a 4th year level within a Software Development degree. The requirements and preferences for this applied project and dissertation can be broken down as follows:

- Investigate the information that real estate agents would like to document in a singular application by doing online research and speaking to a professional or their colleagues.
- Evaluate and investigate the available frameworks and tools for creating an independent web application for data keeping.
- Create and develop an application based on the investigations above in a Create, Read, Update, Delete (CRUD) format.
- Incorporate Security and Validation such that personal data of the users does not get shared amongst others who discover the page and may have malicious intents.
- The application needs to be related to the field of Real Estate Agency and allow the users to manage their data with components of the application such as Listings, Open House entries, To-do's, Notes, Calendar entries and the ability to add the details of people who had attended their open houses.

- The application needs to be easily navigated able by any person who wishes to operate it from the Real Estate Field, this should include people who are not computer-literate along with those who are.
- The application should be mobile friendly in design as the target audience may wish to use it upon a device such as a tablet or a smartphone.
- The documentation of this project and code commentary should be concise and easily understood as the code needs to be understood by any developer familiar with the frameworks or architectures used within the project.
- The dissertation should explain the project in its whole along with the plans, decisions and risks that were encountered throughout.
- By the end of the dissertation it should be made clear why and how the project was made, and no further queries should need to be made into its research or development that is not already disclosed within the dissertation.
- The project should adhere to objectives previously set out as much as possible without over reaching the designated scope or time allotment.

1.2 Metrics for Success and Failure

This section will discuss the metrics for which Success and Failure were established for the project. As the objectives and the metrics for success need to be closely aligned, much of this discussion will be closely related to the overall objectives of the project listed in **Section 1.1.1**. Having such objectives to work towards in the project and defining what makes them successful, or a failure was imperative to the decision-making process during the time creep at later stages during the project as with the nature of the project it could be continually improved and more features could have been added, but it would eventually exceed the scope and run past the deadlines required. Having such metrics established gave a real tangible benefit of what it's like to be a developer working on a industry domain problem within where deadlines are stricter and pressure within the work environment is high.

The metrics for the applied project and dissertation could be described as follows:

1.2.1 Applied Project

The most important metrics for **success** for the **applied project** would be:

- The developed program must have a full stack architecture and make use of modern technologies in an innovative way to accomplish the task at hand, in such a way that it is easy for a user to pick up, understand and navigate when using the web application.
- The project must make use of authentication and validation along with being tested either through unit tests and/or usability tests (Black Box)[2] such that the end application is highly secure and is hard to break during common user use cases which can be established through user stories[3].

The metrics for **failure** within the applied project's development could be explained as follows:

- The program does not suit the purpose originally set out in the objectives and is not easy for a user to navigate or understand.
- The project does not adhere to privacy and security concerns that a user may have and lacks thorough testing such that the application is consistently breaking upon usage.

1.2.2 Dissertation

The major metrics for **success** in the writing of the **dissertation** can be explained as follows:

- The dissertation should be easily comprehended by the end readers and the purpose and importance of the overall project should be made apparent to the readers.
- The dissertation should show evidence of research and thoroughly planned development over the course of the project's conception such that the reader understands the research undertaken by the student in the pursuit of creating an web application for use as a real estate agent aid for data management.

The metrics for **failure** within the realm of the dissertation are:

- The dissertation is not easily digested or comprehended by the end reader's and the purpose of the project is not well understood by the time the reader is finished reading the dissertation.
- The dissertation shows no evidence of research or planned development and the end reader has no understanding of the research performed by the student prior to each phase of the applied projects development.

1.3 Summary of Dissertation Chapters

1.3.1 Methodology

This chapter will explain the research and development phases prior to and during the project. How it was planned for and developed, the different approaches that were considered and the approach used. This section will also give details on the use of a version control system and other platforms used during the applied project to keep track of plans, goals, and issues throughout its creation.

1.3.2 Technology Review

The technology review section of the paper will describe the technical aspects of the project in detail. This section will give insight into the different technology considerations, the technology used, their implementation and purpose within the project. The benefits and faults of the different technologies will also be critically analysed to give the reader a perspective into why the chosen technologies were researched and implemented within the applied project.

1.3.3 System Design

A detailed explanation of the solution architecture of the applied project. This will include the details of the chosen technology's implementation along with code snippets and diagrams of how they were used within the project.

1.3.4 System Evaluation

An evaluation of the overall architecture and solutions against the original objectives set out prior to the project's conception. The final applied project will be reviewed, and critical analysis will be applied such as how robust the

application is and how well does it serve its purpose along with the potential pitfalls and future solutions if development were to continue.

1.3.5 Conclusion

A brief conclusion to the project. Key learning outcomes will be identified and reflected upon and thoughts upon the projects' development life cycle will be explained in the areas of what could have been improved and what had gone well during the applied project and dissertations development.

1.3.6 Appendices

This section will contain any appendices relating to the project such as the demo recording, GitHub URL along with Carissa Jurina's email in the case of queries into the relationship and communication that occurred during this project's conception.

Chapter 2

Methodology

2.1 Initial Thoughts and Research

The project planning began shortly after the first supervisory meeting with Gerard Harrison, where a discussion about ideal project scopes were made. A brainstorming method was commenced on possible different project ideas and technologies that could be used for a final year project. As time would work against those who started late, it was advised to come up with a project idea and report back to Gerard within a week or two maximum with it. The discussions I had with Carissa Jurina about a real estate application had come to mind during this initial planning phase and was decided upon the next week. The major issue with the planning came in terms of what technologies to use, as this required extensive research and came under the following key headings.

- What is currently used in industry?
- What do I as the developer have experience in?
- What technologies would be ideal for a CRUD application that could be created by a single person within a period of 6 to 9 months?
- Are the selected technologies interesting enough in complexity to be considered worthy of the applied projects' scope at a 4th year level?
- What Development Methodology to use?
- How should further research be handled?

2.2 Project Supervisory Meetings

Once the basic project plan was created, the questions outlined above were brought to the attention of the project supervisor as he's experience within the applied project space would be invaluable in making decisions on research methodologies and web technologies to use as he could critically analyse what could be completed in the time and scope given. He's ideas and direction helped complete the foundation from which the project could begin.

Meetings with the supervisor over the direction and development of the project continued over the weeks of the project creation and were performed on each Wednesday from September until May during which the project needed to be submitted.

During such meetings suggestions were often given by the supervisor and ideal goals to have completed each week were given. This system of advice was useful as it gave the project more structure and urgency during times where the development lost focus due to outside matters.

2.3 Research Methodology

In order to have a grounded approach to the problem I discussed the features a real estate agent would like to have in the form of application the project would be based around with Carissa[1] who also approached her co-workers with questions inline with what was asked of her by me. These responses were then researched, and different technologies were considered by their efficiency in creating solutions to the wants and concerns of these real estate agents. As Carissa had previous experience with computer science, she respected the scope and time constraints on the project as best as she could and stuck with basic outlines whilst not asking for features that would be too hard to develop within the projects' timeline.

2.4 Development Methodology Considerations

As the field of computer science is quite vast and has a long history, the number of development methodologies that could be used were immense but despite this vast selection, two major methodologies continually were mentioned by the college lecturers throughout the course of the degree program and code based bulletin boards online such as Stack Overflow and Reddit software development based discussions. Those being that of the Waterfall and Agile Methodology types.

2.5 Determining which methodology to use for developing the application

Extensive research was done into the merits and faults of both the Agile and Waterfall methodology types and their sub categories such as Kanban and Scrum.

2.5.1 Waterfall

The Waterfall based methodology breaks down the project into linear and sequential stages, where each stage has to be completed before the project can progress. This puts waterfall under two distinct attributes.

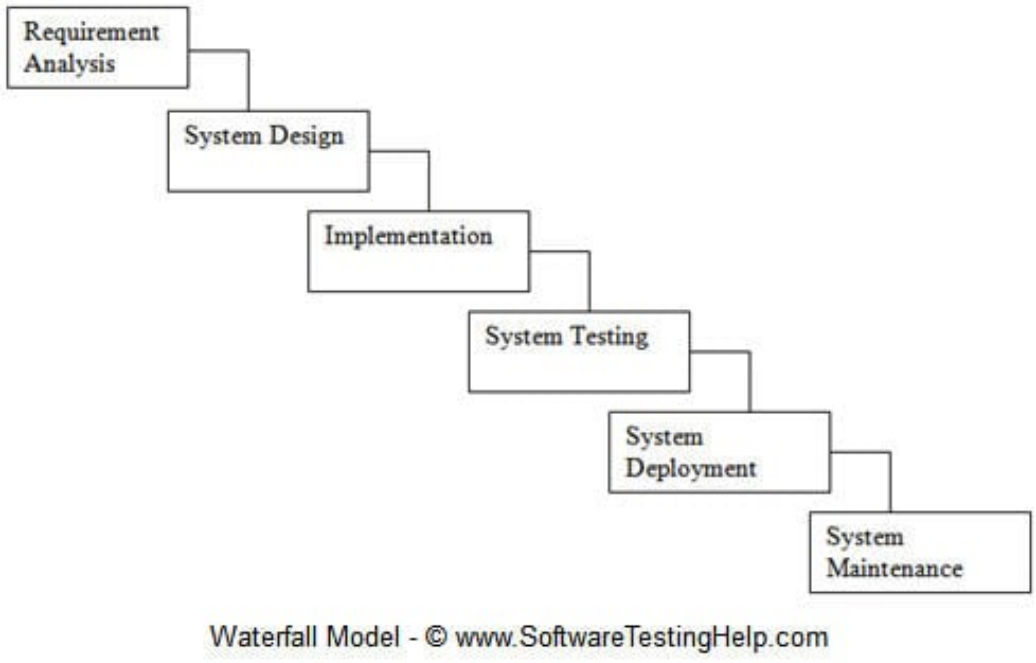

Figure 2.1: Waterfall Methodology

- **Discrete Phases** In the Waterfall methodology, you cannot continue onto a section of the project unless a preceding section is complete first, such that it works in a traditional way with a likeness to that of the workflow within labour or engineering disciplines. For example *you cannot put an engine in a car before the chassis is made as an automotive engineer.* Likewise, you cannot go backwards within the project

when a phase is complete, you can only go forward into its development, this offers a lack of flexibility hence why the second attribute of the Waterfall methodology is very important[4].

- **In Depth Documentation** Because Waterfall discourages back tracing within the project, it requires the developers to do thorough research into the requirements of the project along with each phases end goal. Teams within the software or engineering industry that use the Waterfall model as their main development methodology must continuously document their work in its entirety such that if team members were to be changed throughout the development cycle of the project due to whatever circumstances may occur, the project would not be halted and as such all decisions and design choices would be made apparent to the new members on the team when reading the documentation[4].

2.5.2 Agile

Agile is often the preferred choice within software development work settings as its main focuses on are collaboration, self organization and a cross-functional approach to completing work solutions or requirements. Agile breaks projects into smaller, iterative periods which works well under client orientated work environments where solutions are often user-tested or regular revision occurs. The Agile model encourages the developers to improve and adjust workflow as needed unlike the Waterfall model where the result of the development phases are set in stone and often unchangeable. So the major key point of the Agile Methodology is its ability to compensate for changes needed within the development processes of a project as it adapts to the needs of clients when alterations to the project are needed or suggested[5, 4].

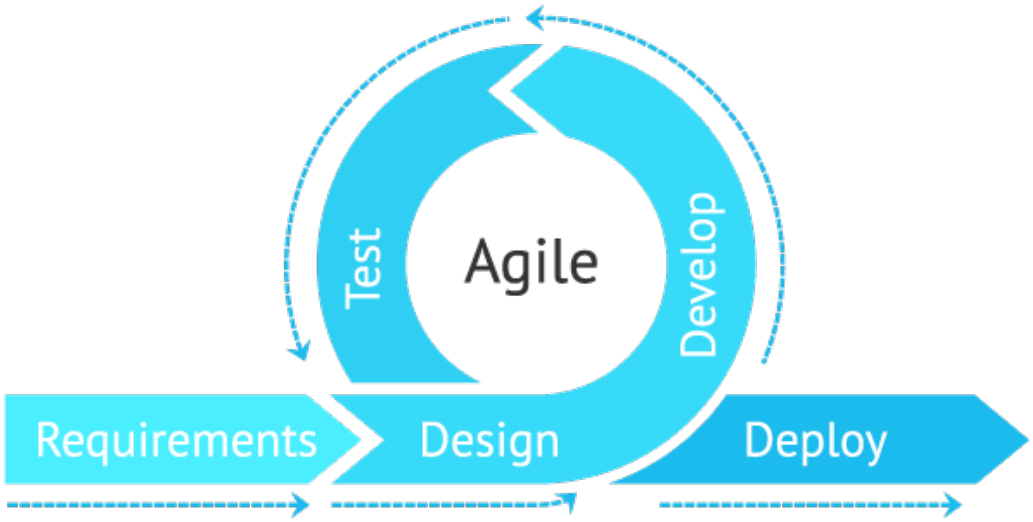

Figure 2.2: Agile Methodology - Image by DevCom.com

As the Agile methodology is more known as a generalized term for a subset of development workflow methodologies. The following two sections will discuss both the Kanban workflow management system and then the Scrum workflow management system (*Section 2.5.4.*) which both are two commonly used types of Agile Development sub methodologies in modern industry level software teams.

2.5.3 Kanban

Kanban focuses on the co-ordination of software teams through using a workflow based board known as a **"Kanban Board"**. This board can be both physically written upon a surface such as a white board or it can be a virtually created board upon a team's communication platform or version control system such as Jira or GitHub [4].

The Kanban board consists of categories such as *"Work to be Done"*, *"Work In Progress*, *"Work Completed"*. Although these categories are most common, it is not unusual to see companies add additional categories such as those needed for bug fixing or review purposes. Each task that needs to be completed is recorded upon a **"Kanban Card"** which will progress through each category depending on what development state it is currently in. The purpose of the Kanban Board is to keep all the team on the same page in a simplified format such that each team member knows what needs to be done and what tickets are available to take, along with what has been completed thus far[6].

The Kanban methodology requires strict limits on progress made at any time such that if a category such as "Work In Progress" is filled with many cards, a developer cannot take another card and add it to the same section until some of the cards in that category progress to the "Work Completed" section. This promotes collaboration among developers as they try to help other members of the team who may be behind catch up and remove that card from the work in progress category and progress it to the work completed category[6, 4].

In terms of communication within Kanban, meetings are often held to discuss changes that need to be made and, these changes are then represented upon the Kanban Board. These regular meeting allow the team to correct and adjust their processes such that it improves workflow without sudden unexpected changes occurring hence Kanban is easily implemented into software development teams and has shown improvements in the software development workflow of teams in the many industries who have decided to implement it such as Pixar, Zara and Spotify [7].

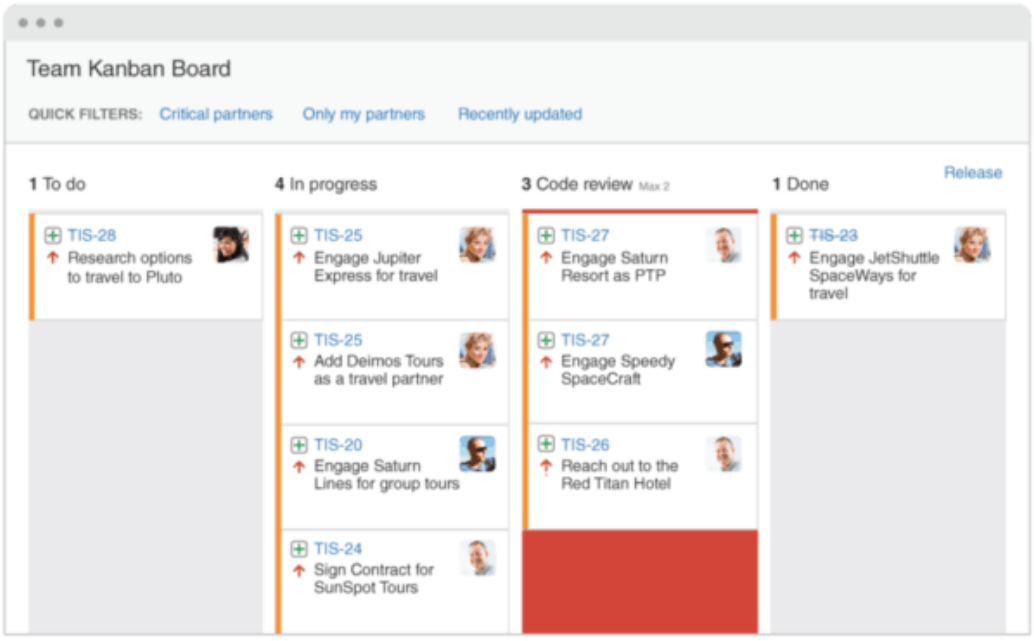

Figure 2.3: Kanban Methodology Example

2.5.4 Scrum

The final methodology that was researched was that of Scrum project management. Scrum commonly tackles knowledge based work such as software

development but, it differs from Kanban in that it is more concerned with outputting results faster whilst Kanban focuses on continually improving workflow processes [8, 4].

One of Scrums' key attributes is the idea of using short Sprints to get work done. These sprints can range from 2 weeks to a month in size and once they are completed, the work performed within them is reviewed. Before each sprint a meeting is held to discuss what is known as a **sprint backlog** which consists of different tasks that need to be managed or completed.[8].

At the end of each day the teams hold a 15-minute scrum meeting where the contributors to the project discuss potential roadblocks that are interfering with the sprints' or projects' success, and they also review the work completed that day and what each member plans to focus on for the next work day. These meetings ensure each of the collaborators stay in sync and nobody is left out of the workflow loop.

Each team has a leadership role member called a **Scrum Master**[9], who links the team to the product owner / client and before the beginning of each project the team commences upon, this member holds a discussion with the client in order to define requirements for the next project. Once a project has commenced and the first sprint has started, this member also takes upon the duty of removing any roadblocks that arise throughout the development life cycle.

A common misconception about the Scrum Master is that they function similarly to a traditional product manager which is simply not the case, as a Scrum Master facilitates the work done within the team rather than managing it since one of the main attributes of the Scrum Methodology is that teams manage their own productivity and a Scrum Master simply helps them reach their project objectives [4].

Unlike in Kanban, Scrum likes to use a system called the **Burndown Chart** although it may be similar in theory, it is much more simplified and only contains notes on what needs to be done, not the actual project workflow from start to finish. This burn down chart needs to be continuously updated by the Scrum Master and the team in order to facilitate the progression of the project as objectives are met overtime. [4, 8].

2.5.5 Chosen Methodology

As the project was being created by a single person. The **Agile Kanban route** was chosen as the development methodology on the basis of its adaptability and organizational qualities. Although I did not have a team to work with, the ability to keep track of the various work categories such as "Work to be done" and "Work finished" within the project came in great use as dur-

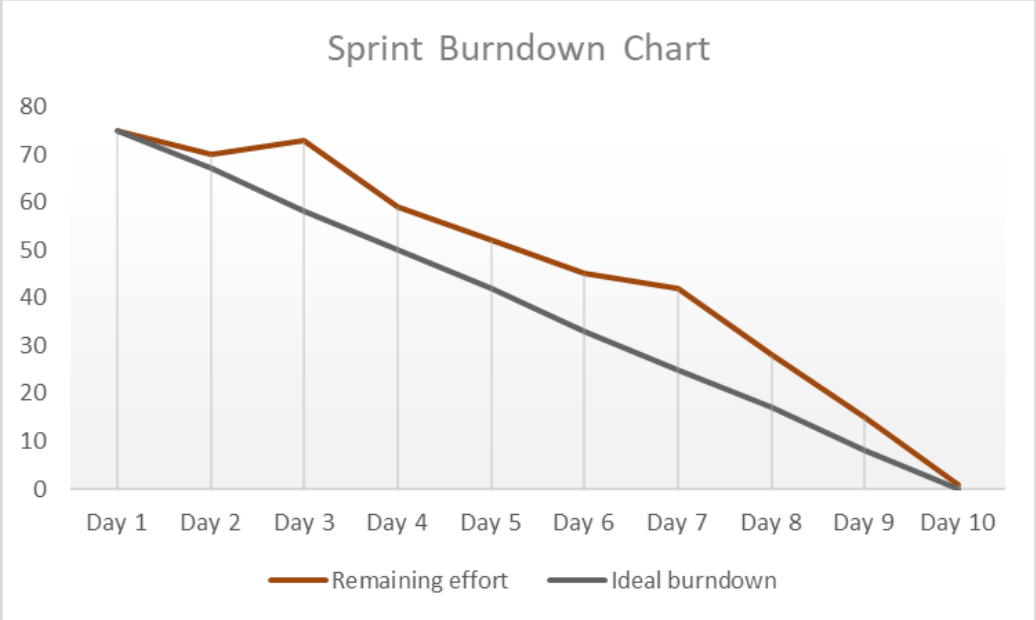

Figure 2.4: Burndown Chart Example

ing certain phases of the projects' life cycle situations outside of the project would become hectic and breaks were often taken from the projects' development in order to facilitate other subjects within the degree year. Having this objective based system kept me on top of where I was within the project after any unwanted breaks in its workflow, even if project phase deadlines were not always met, it made it easier to know what parts of the project needed to be worked upon, when returning to its development.

2.6 Validation and Testing

To ensure that the project was robust and secure, the method of testing was mainly that of black box testing[2] and user stories[3], as the project would be repeatedly ran and attempts to break functionality within it were performed. The reason for focusing mainly on black box testing is that the issues with functionality often presented itself in unpredictable ways such that the only way to make the bugs occur within the code would be to consistently run different sections of the web application and input incorrect values. Validation and Authentication were heavily researched due to this and overall by the end of the project it had become increasingly difficult to break functionality within the project during testing. In terms of tools used for testing, the main libraries and applications used were **Postman Test**

Scripts for the Back-End and **Selenium IDE for the Front-End**, both of which will be discussed in the **System Evaluation** section of this paper in terms of their usage to evaluate the software's robustness.

2.7 Version Control System

The next question in relation to the methodology was the actual coding of the project and the discussion of which version control system or product would be best suited to store the projects updates over time. As the Git library was often used throughout the college degree and is frequently relied upon in the software development industry, I felt it was best to store the project on a git based online platform as it would offer my project a reliable versioning system, security by storage online and give myself relevant industry practice in managing a project through git. The git repository hosting platform to which the project would be stored upon would need to have the following traits:

- The ability to interact with the git terminal and contain git repositories
- The ability to store README markdown files such that any reviewer of the code could be given an explanation on how to deploy it upon their machines and could understand the purpose of the project before downloading its code base locally.
- It had to be secure and be consistent, such that the version control platform would have to have a long history of successful project containment.
- It must hold any commits or versions of the project in the case of major hardware failure or the need to revert the projects progress/version to remove a feature or bug.
- It would preferably have an in-built project management system such as a Kanban Board or Note Cards where I could create project information cards and follow the Kanban Agile methodology without the need for managing the projects' Kanban board upon different platforms separately.

2.7.1 Options Considered

Two major companies, **GitHub** and **Git Labs** offered the ability of hosting git repositories online and mostly followed the trait requirements outlined

above. With both companies long-lasting history within the software development industry and my own personal experience with them, it was felt they could be trusted to host the projects git version system upon and not need to worry about losing progress due to their servers going down or other technical and hardware based problems that may occur on lesser known git based platforms that are less established within the industry.

GitHub

In the following bullet points I will discuss the pros and cons of the *GitHub Software Development Version Control System* [10]:

Pros

- Allows the easy sharing of open source code basis.
- Has a user-friendly UI jam packed with sophisticated features.
- Offers the ability to make pull and merge requests on machines far away from each other.
- Offers the users' ability to comment upon commits or merge requests along with setting up issues for open source products.
- Has a Large Community with extensive forums on any solutions to issues that may occur.
- It's easy to set up using a few command line commands or even using their own GUI Application known as GitHub Desktop.
- Provides project management features such as Automatic Kanban Boards and a detailed Wiki for those who do not enjoy writing extensive README's using Markdown.
- It is the most commonly utilized Git based platform by personal developers, open source developers and industry based software teams according to OpenSource.com[11].

Cons

- It does not offer very good API development features.
- Is not completely free, especially in the use cases of private repositories.
- It is not regularly updated and lacks the features of some other version control system platforms.

- Some personal developers may not feel comfortable keeping their private code bases on GitHub since Microsoft bought it out in 2018 as now it would be considered a commercial platform and often Microsoft has come under scrutiny for it's user privacy practices[12].

Git Labs

Often seen as the main alternative to GitHub, the following pros and cons of Git Labs are detailed as follows: [10, 13]:

Pros

- Consistently updated with new features such as *project group locking, restrict by IP address and audit events.*
- Allows users easier interactions in making private repositories without needing to pay for a premium membership.
- Has a Command Line Interface built in to the user experience.
- Offers package management.
- Supports Continuous Integration and Continuous Development life cycles.
- Easy Maintenance of Code Bases.
- Ability to add weight to issues such that a developer can set one issue as more of a priority over another.
- Ability to move issues across projects and add Burn-up or Burndown Charts.
- In App Messaging without the need for comments upon commits.
- Offers Free Self Hosting and Time Tracking features.

Cons

- It lacks some enterprise level features.
- It has a smaller Community.
- It has a problematic upgrade process as restarts are often required.
- Is not automated testing friendly.

- User Interface is daunting for first time users coming from GitHub or developers with not much experience.
- Kanban Board features are not on the same level of that as GitHub's.

2.7.2 Chosen Platform

In the end after weighing the pros and cons of both platforms. The project version control would seemingly be best hosted upon the GitHub platform which was more familiar to me as the developer. GitHub was easier to navigate and featured a very extensive Project Management and Kanban Board System which helped immensely in the projects' development. It is also more commonly used by users and if the project were to be extended past its submission within the college program and be developed further, the potential need for extra developers to work within future development, would be easier to obtain as more developers are currently using GitHub over Git Labs as their main Git Version Control System platform [10].

Chapter 3

Technology Review

This section of the paper aims to discuss the technologies and tools that were considered along with those that were chosen to create a solution for the project along with giving insight into why they were chosen and the research undertaken to make that choice.

3.1 Initial Technology Considerations

Before the development of the project could take place, it was required to research what technologies were available and what impact they currently had within the software development industry. This was important for figuring out the approach to the project that would be used and uncovering the drawbacks of using each technology.

3.1.1 MERN Stack

The MERN stack stands for the **MongoDB**, **Express.js**, **React** and **Node.js** stack. This stack is a variation of the MEAN Stack which will be discussed in *Section 3.1.2*. This stack is based on a 3-tier architecture in which you have the front-end, back-end and the database. The stacks code base relies mainly upon JavaScript, a programming language mainly used for web development and JSON parsing, a language independent data format. Both JavaScript and JSON are very common within the web development section of the software industry but with modern Node.js, it is also possible to develop desktop and mobile based applications utilizing JavaScript and using frameworks such as React Native and Ionic. As the MERN stack utilizes popular languages within modern web development, one of it's many advantages would be that it's easy to pick up for developers who have a long working

history with JavaScript [14].

MongoDB

MongoDB relies on storing JSON-like documents within a database as according to the MongoDB developers

> *We believe this is the most natural way to think about data, and is much more expressive and powerful than the traditional row/column model.*[15]

The MongoDB architecture is under an umbrella term category known as NoSQL databases which means that it does not have a tabular system with referential integrity in mind like classic SQL databases such as MySQL and PostgreSQL which both rely on the indexing of primary and foreign keys to accomplish their referential integrity. This may sound like NoSQL based databases like MongoDB are not as good but in fact both types of databases come with their own benefits as listed below[16].

SQL

Pros

- Reduced data storage footprint due to **normalization** and other optimizations on SQL platforms.
- Strong and well-understood data integrity semantics using **ACID(Atomicity, Consistency,Isolation, Durability)** properties.[17]
- Standard access to data using SQL commands.
- Flexible Queries that are efficient in large workloads.

Cons

- Rigid Data Models that are not friendly to alteration and require heavy planning.
- Scaling Horizontally is often challenging as it's either unsupported or is supported through ad-hoc and relatively immature technologies.
- As it is hard to distribute the system, it often has a single point of failure that can only be mitigated by replication or fail over techniques which are costly on storage.

NoSQL

Pros

- Supports seamless high horizontal scaling across multiple platforms.
- Flexible data models as no up front commitments to data models need to be made, and schemas that do require change can be updated on the fly.
- High performance by limiting what the database can do, such as relaxing durability guarantees. Many NoSQL systems are able to achieve extremely high levels of performance according to IBM [16].
- High-level data abstractions as moving beyond the values used in data cell model, NoSQL systems can provide high-level APIs for powerful data structures such as Redis and its included native-sorting abstraction.

Cons

- Vague interpretations of ACID constraints, despite widespread claims of ACID support in NoSQL systems, there is a lack of semantics for ACID based constraints such as how to perform isolation when there are no ACID transactions.
- If used in a distributed way, it will incur the normal systemic problems that distributed systems have along with the issue of Cap Theorem where a NoSQL database can only provide two of the three desired attributes of a distributed system, i.e **consistency, availability** and **partition tolerance** which form to make the acronym **CAP**[18].
- Lack of flexibility in relational SQL abstraction which gives the SQL based database engines power to optimize queries on the data without abstraction. Without the prior flexibility sometimes data leaks into application based queries, and it leaves the engine no room to optimize itself.

Within the MERN stack they choose to use MongoDB as it is scale-able and easily distributed with mass amounts of data and is easily picked up by web developers who are used to dealing with JSON like language formats.

ExpressJS

ExpressJS or commonly known as just ”Express”, is a web framework used for creating back-end web applications upon the Node.js platform. It features a minimally semantic design and can help create a robust and flexible web applications within Node.js using common HTTP methods and parameters.[19]

Here is an example of setting up a simple home route within ExpressJS that return a ”Hello World” result.

```
app.get('/', function (req, res) {
  res.send('Hello World!')
})
```

ExpressJS is used to wrap the back end Node.js code within the MERN stack and can be used to create streamlined routes that communicate with the front end. As routes can be created from previously made JavaScript files within a matter of lines of code using Express[20], it is highly efficient and lightweight in design such that it works as a solid middleware to the MERN stack even for developers who are not highly experienced within web technologies.

React

React is a JavaScript library built by Facebook and it was released in May of 2013 for the purposes of creating dynamic user interfaces or components for front-end web applications. It is often used within single-page architectures or mobile based architectures using it’s React Native counterpart. It’s main concern is that of state management and rendering state data onto the Virtual DOM provided on the library through it’s components which can be coded by the developer or downloaded by third parties through package managers such as the Node Package Manager or Yarn [21].

When beginning any application built upon the React library you will need to use the following command within a command line interface:

```
C:\Users\Your Name>npm install -g create-react-app
-OR-
C:\Users\Your Name>npx create-react-app myfirstreact
```

The above command will initialize the library within the folder where the user is typing the command. Once the library has been initialized fully, you can begin making folders for views and components as needed which are all

based on JavaScript classes and/or functional based classes. An example of some react based code could be written as follows:

```
import React from 'react';
import ReactDOM from 'react-dom';

class Test extends React.Component {
  render() {
    return <h1>Hello World!</h1>;
  }
}
ReactDOM.render(<Test />, document.getElementById('root'));
```

Because of Reacts' component based architecture it is very lightweight and does not come with the added bloat that many other technology stacks come with but despite this it is often times hard to find third party components that offer the functionality that some more robust frameworks offer from the start.

NodeJS

NodeJS which is also commonly just known as "Node", is an asynchronous event-driven JavaScript run time that can allow the developer to run their JavaScript code locally or through a web application [22]. NodeJS plans to tackle the issue of modern concurrency models within web development by using a call back system that sleeps when there is no work being fired, such that it is free from the worries of the dead-locking process as there are no locks placed on threads. Due to the previously mentioned attributes, NodeJS is highly scalable even for larger web applications. Although it is not inherently concurrent, it is still possible to create child processes using their inbuilt method[22]:

```
child_process.fork()
```

3.1.2 MEAN Stack

The MEAN stack is very similar to the MERN stack with the main difference being that the Angular Framework is being used as the major front-end technology over the React Library in the MERN stack[23, 24].

Angular

Angular which was developed by Google and currently mainly uses Microsoft's JavaScript inspired language Typescript, is another major web framework. React and Angular are often compared in the front end development landscape, but they are both quite different in purpose as React is only a library which requires extra components to be downloaded such as React-Router-Dom and Axios to perform tasks such as routing and HTTP requests, whilst Angular is a full package within itself and contains numerous built-in packages and libraries that do not need to be downloaded separately to work in areas such as routing and HTTP requests. Below is a comparison list between React and Angular [23, 25].

3.1.3 Bootstrap

Bootstrap is the world's most popular CSS framework and offers the ability to quickly design and customize mobile-responsive web applications and hosts features such as Sass variables, Mixins, a responsive grid system and an extensive amount of prebuilt JavaScript plugins. Bootstrap makes it so that the developer has to write less CSS code and can focus more upon the functional side of developing the front-end of a web application[26].

3.1.4 Hosting Platforms

The initially considered platforms for hosting the application were **Amazon AWS**, **Heroku** and **Firebase** as they seemed popular among developers to host modern JavaScript Frameworks upon.

Amazon AWS

Amazon's web hosting platform AWS hosts many free tier and paid tier services such as machine learning, storage, analytical tools, deployment, virtual machines and various cloud functionality. Due to all the features AWS offers it has a steeper learning curve and this makes it harder to set up, particularly as an inexperienced developer within in its services thus that the AWS platform is best suited for larger, more mature teams and projects rather than fast, lean startups or prototyping [27].

Heroku

Heroku based on the platform as a service architecture (PaaS) which uses a container inspired system with integrated data services for running small

to medium scale web applications. It has features for cloud computing and storage such as PostgreSQL built in along with numerous other features and is fully managed thus giving developers the freedom to focus on their core product.[28].

Firebase

Firebase is Google's own hosting application that allows developers to host full stack web applications upon their platform using features such as Fire Storage, Firebase Authentication, Firebase Real-time Database along with numerous extensions that can be imported into your web application with ease such as the Google Maps API. It is also fully manageable from the Google Cloud platform which it is built on top of [29].

3.1.5 Google Calendar API

The Google Calendar API [30] was originally intended to work alongside the Full Calendar React package[31] to allow the synchronization of both the Calendar within the application and the user's Google account calendar if they were using a Google Email, but this feature was scrapped after privacy concerns arouse due to the Calendar's API requirement of having a public calendar setup on Google which would not be optimal for Real Estate Agents who may deal with sensitive information.

3.2 Chosen Technologies

3.2.1 FERN Stack

The FERN stack is similar to the MERN and MEAN stacks except MongoDB is replaced for Google's Fire Storage, which is a feature of Google Firebase. The added benefits of using such a stack is that it's highly secure behind Google's own infrastructure and is documented exceedingly well upon their website. Google will also offer any support with issues that may occur with the databases you host upon their platform.

The reason Firebase was chosen for the applied project over Amazon AWS or Heroku was because of the projects scale and due to this the utilities and support for large software infrastructures that AWS would offer would not be applicable to the use case of the project and such the added difficulty of using AWS with all it's in built features would only hinder the development speed of the application as AWS is often described as hard to learn efficiently such

that full certification courses on websites such as Udemy and Coursea have been created for the purpose of learning the AWS Web Application Hosting. As this project was not funded by an industry partner, considerations into the financial aspect of each hosting platform were a key part in the research. Firebase offered a very forgiving free tier under its blaze plan such that the application would need to make hundreds of thousands of reads and writes to its cloud functions service before charges would be applied.

3.2.2 Why React was chosen over Angular?

React was chosen over Angular due to its lightweight design which was useful for keeping bandwidths and storage requirements down when hosting the web application's front-end. React was less familiar to me as the project developer and hence would be able to test my ability to pick up new frameworks during a project with tight deadlines. The module system of React also became increasingly useful over the course of the projects design with modules such as Axios, Full Calendar and jwt-decode. These modules mentioned previously allowed aspects of the project to be streamlined as the modules were intuitive and hence allowed the development process to be less cumbersome for the more complicated parts of the front-end.

Research into Angular and React's popularity among developers and within industry use was performed in order to ensure that the technologies were relevant and to check which one would be more beneficial to learn skills for currently as of the time of creating the applied project and dissertation. Through research on GitHub it was found that React was more popular among developers than Angular and had achieved higher ratings. The research showed React was the second most popular GitHub package under the Frontend section of GitHub Topics[32] and had 168,000-star ratings whilst Angular fell behind and held 73,000-star ratings. Based off these ratings it could be discerned that people often preferred to work in React over Angular, but the reasons for such can not be explained through the research as developers individually have their own personal reasons for preferring one front end framework over another.

Another source of the previous research was found from tsh.io[33] where a survey of 4500 developers was performed internationally as shown below

The result of this survey concluded that in terms of the front-end development landscape React was dominating every other JavaScript Web Framework in terms of use and preference as of 2020. Most developers who underwent the survey confirmed that they would either like to work in React professionally or would like to continue to use it in their professional careers. It was also notably higher in "used within the last year" portion of the sur-

Figure 3.1: Tsh.io State of Front-End 2020

vey and often doubled the size of other frameworks graphs such as VueJS, AngularJS and Svellte.

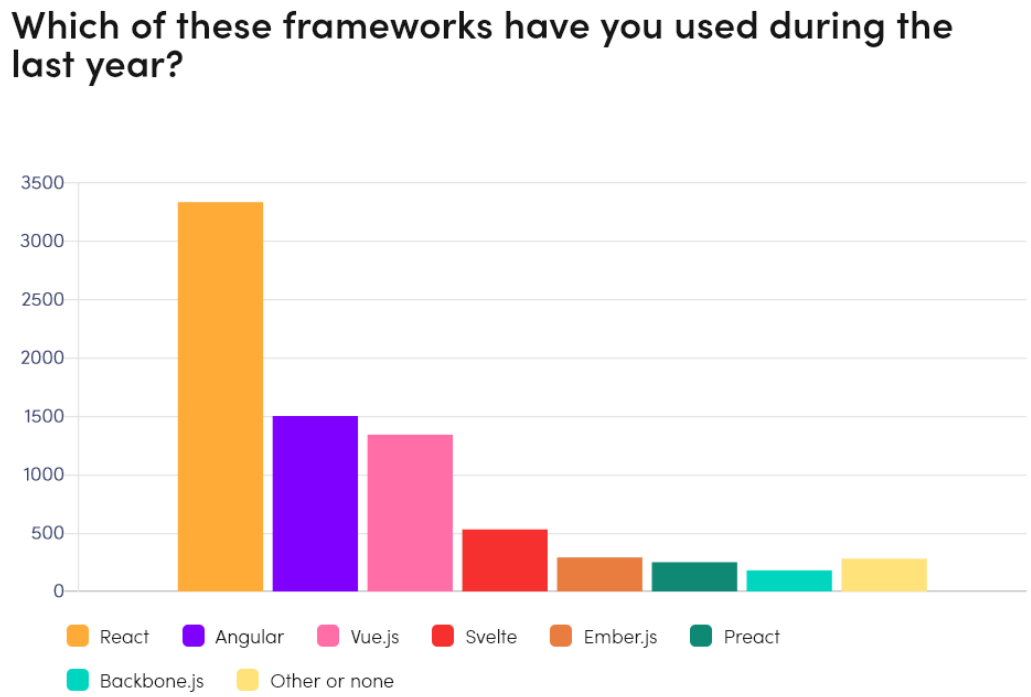

Figure 3.2: Tsh.io JS Frameworks used in past year

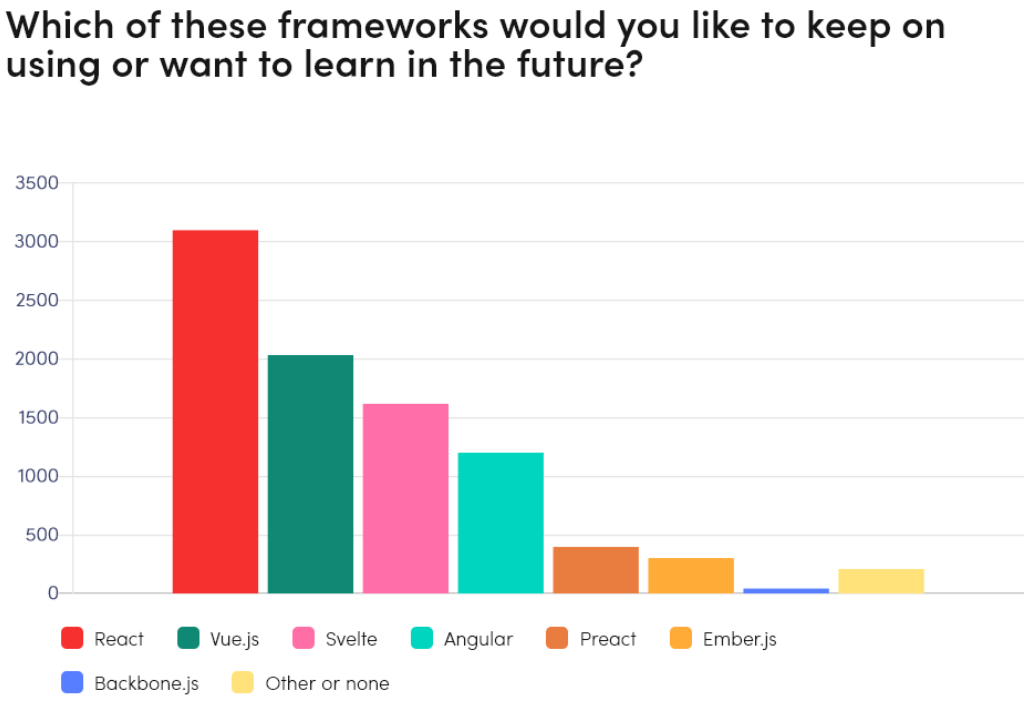

Figure 3.3: Tsh.io JS Preferred Frameworks Survey

Despite the results previously mentioned, the topic of Typescript as a programming language over JavaScript was also researched during this survey and due to the results of this research, the surveyors came to the conclusion that Typescript was a preferable language to use over JavaScript if given the opportunity and thus due to React's more JavaScript orientated nature, it may fall off as the top front-end web framework over the coming years[33].

3.2.3 Material-UI

Material-UI is a CSS framework based upon the Google Material Design Standard [34] and makes use of multiple CSS based components in react to contain and style your React components within. It was used in the project extensively in order to improve the visuals of the project and add responsive design to the pages[35].

3.3 Languages Involved

3.3.1 HTML

HTML also known as Hyper-Text Markup Language is the standardized markup language that defines the layout of a web page. It has no functional properties on its own but when mixed with languages such as JavaScript or Typescript, it can help create dynamic web pages[36].

3.3.2 CSS

Cascading Style Sheets commonly known as CSS, is the styling language of HTML DOM components such as colours, text styles and animations. In essence, it describes how elements should be displayed visually [37].

3.3.3 Javascript

JavaScript is known as the programming language of the web as it or some subset of it like Typescript adds dynamic functionality to most HTML pages. According to w3techs.com , 97.2% of all websites make use of the JavaScript programming language in one way or another[38].

3.3.4 JSON

JSON stands for JavaScript Object Notation and is a lightweight format for storing and transporting data. JSON should be self describing and easy to understand by developers and the web pages that parse it into the components of the HTML. Every JSON style object should have key value pair separated by commas and curly braces for embedded objects [39].

3.4 Development Environment

Visual Studio Code[40] was chosen as the main development environment as the IDE offered the necessities to develop the application without being too bloated with unneeded additions. If any extensions were ever needed for the development of the project the Visual Studio Code Extension Marketplace offered extensions for them such as code formatting, git version system helpers or various other tools to streamline the process of applications development.

3.4.1 Testing Tools

Postman Testing (JavaScript)

Postman was chosen for the back-end API route testing application, as it was easy to use and fulfilled the basic testing needs of the API [41].*Delete Routes and Update routes were not tested as they work based off a randomized ID and could not be predicted.* The Postman Testing Script suite, is based upon JavaScript and an example of such will be given as follows:

```
pm.test("Response is Status Code 201", function () {
    pm.response.to.have.status(201);
});

pm.test("response must be valid and have a body", function () {
     // assert that the status code is 200
     pm.response.to.be.success;
     // assert that the response has a valid JSON body
     pm.response.to.be.withBody;
     pm.response.to.be.json; // this assertion also checks if a body
     ↪  exists, so the above check is not needed
});
```

Selenium IDE

Selenium was chosen for the front-end testing of the application as it was very quick to set up while offering a plethora of web application testing features. It allows targeted testing of HTML tags by their Class Name, ID or X-Path. It also offers the feature to record the user's movements upon a web page and then allow repetition of the actions the user took to ensure that the application functions perform consistently upon multiple iterations of the same intended actions. Selenium IDE is installed simply through a plugin on multiple browsers such as Firefox or Chrome and within the application you can install further plugins for features that are not inherently built into the Selenium IDE package [42].

3.4.2 Deployment

The project was hosted upon **Firebase's Hosting** feature as to keep the origins of each section of the project upon the same platform, thus allowing

the ability to manage them without switching through multiple platforms as the data is moved through the application. By hosting upon Firebase the metrics on reads, writes,deletions, updates along with the finances were view able in a seamless way, which allowed the project to stay consistently under the free tiers Google offered upon their platform and allowed the Firestore database to be modified as needed through their online Firebase Dashboard if there were major feature changes or if the deletion of test added data was required.

Chapter 4

System Design

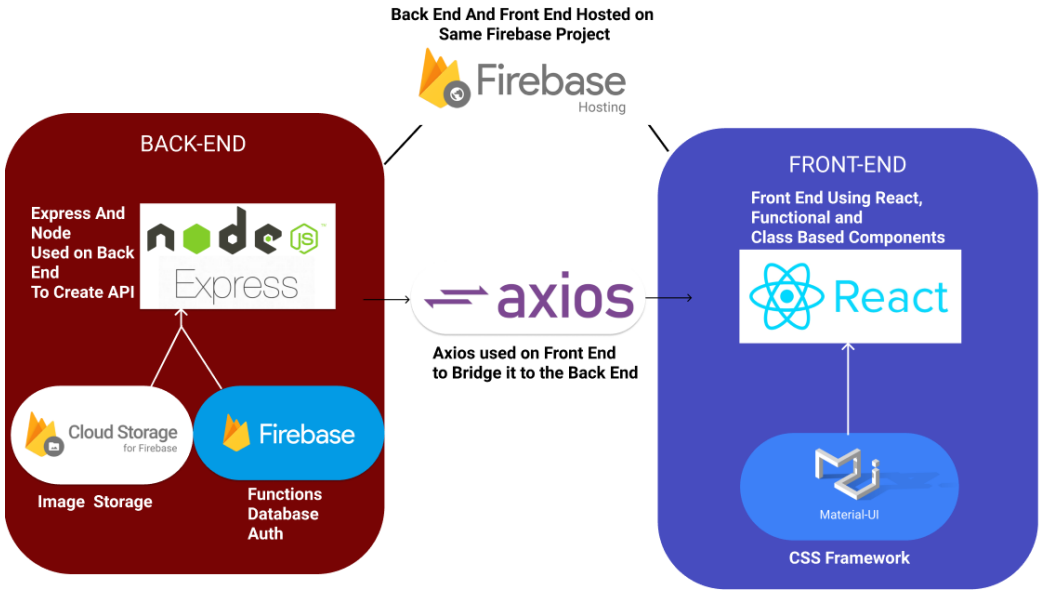

Figure 4.1: Project Final Architecture

4.1 Architecture Brief Overview

The above **Figure 4.1** gives a brief visual outline of the final project architecture. On the Back-End layer Node.js sends requests and receives data from Firebase using their native API and builds functions and databases upon a project setup on their platform prior to the back-end's development. The database encrypts the ID's of all the different documents created within it such as Calendars, Listings, Notes etc. More importantly user accounts

are set up through the back end and registered upon Firebase under their authentication feature, so that they are mildly secured by Google's own infrastructure through conversations with the back-end. A Firebase Storage bucket is referenced within Node.js on image uploads through the Listings or Open Houses section of the Node.js Back-End. All of these features are then deployed using the **"firebase deploy"** command on the back-end folder using a command line interface such as Command Prompt or Bash but during the testing and development phase of this project, **"firebase serve"** was more often used as it does not deploy the back end online and keeps it upon the local development machine such that if major bugs were created, it would not incur consequences such as service downtime or large charges from the Google Firebase platform.
On the Front-End React components are created to create a visual display of any information coming in from the back-end API and a module named BCrypt is used to decode any JWT *(JSON Web Tokens)* Tokens that were created through the authentication functions on the back-end. Axios is used to grab the data from the back-end as it work as a bridging tool to call API routes with front-end parameters and receive the JSON data back from the Node.JS back-end. Most of the visual responsiveness and clean UI was built using the Material UI framework package within React which was installed by running **"npm install @material-ui/core @material-ui/icons"**.
After the full stack application is completed, it was then fully deployed to Firebase by using the firebase deploy command separately upon the back-end and the front-end as they both are stored upon different sections of the Firebase Project, i.e. the Back-End is deployed upon the Functions Feature of Firebase and then the Front-End is deployed upon the Hosting Feature of Firebase. In order to make the front-end deploy-able to fire base, a build folder would need to be created upon the front-end running **"npm run build"** through the node package manager and then firebase would need to be initialized upon the front-end section using the **"firebase init"** command which had a command line interface tool built in to streamline the process of sending the build folder to deployment. After all of this CORS rules would need to be changed upon Firebase using the Google Cloud Command Line Interface and setting the origin to the front-end's URL, this step is important as even though the full stack is hosted upon the same infrastructure and under the same project name on Firebase, they use different IP addresses and the front-end would need to be allowed to access the deployed fire-base back end in order to receive, create or modify files. The next few sections of the paper will cover the code involved in both the back-end and front-end of the application and any other important architectural features.

4.2 Back-End Design

4.2.1 Express JS Index Page

Upon the entry point **index.js** page of the back-end, Express.js is used to set up the routes for the different functions and models within the back-end of the application along with the CORS package being used to help allow cross-origin access into the back-end. A custom package that will be discussed in the Utilities *Section 4.2.5* of this chapter known as **FBAuth** is used to create authenticated express routes within the API so that only a user with a valid authentication token from Firebase can access their features, an example of this code is given as follows:

```
1 // --NOTES ROUTES--
2 app.get("/notes", FBAuth, getAllNotes);
3 app.get("/notes/:noteID", getNote);
4 app.delete("/notes/:noteID", FBAuth, deleteNote);
5 app.post("/notes", FBAuth, postNewNote);
6 app.put("/notes/:noteID", FBAuth, updateNote);
```

As disclosed from the above code snippet, certain routes make use of the custom-made FBAuth (Firebase Authentication) middleware such that they cannot be accessed freely by anyone with the API link, such that this FBAuth functionality sets up the Request User Handle such that the user will only be able to receive documents back that are related to their own user ID.

In order to set up the routes for the functions, all the functions within the back-end need to be imported to index.js such that the bulk of this file is made up of import statements and the rest of the file mostly handles Express routing which are mostly single lines of code. The index.js file holds a reference to the firebase-functions package which is called upon in the end of the file in order to set up the express apps upon a European Firebase server.

```
1 exports.api = functions.region("europe-west2").https.onRequest(app);
```

4.2.2 Data Models

Users

The users class holds functions to **sign up**,**login** and retrieve a **authenticated user**. The **sign up function** begins by creating a *new user object* from the request body fields such as email, password and handle(user name) and then runs the request fields through a custom-made validation function which is from the validators.js class that will be discussed in the Utilities *Section 4.2.5*. If the inputted request data is valid and not empty, then the user will be checked if it exists. After these steps are cleared and the user does not already exist and is validated, then the function will call fire base to create the user and retrieve the authentication token from fire base and log this user in.
The login function called upon Firebase authentication service and tries to log the user in using the request email and password which also need to forgo validation beforehand. If the details are correct, the response will be the authentication token the front-end needs to decode later in order to keep track of the logged-in user.

```
exports.login = (req, res) => {
  const user = {
    email: req.body.email,
    password: req.body.password,
  };

  const { valid, errors } = validateLoginData(user);

  if (!valid) return res.status(400).json(errors);

  firebase
    .auth()
    .signInWithEmailAndPassword(user.email, user.password)
    .then((data) => {
      return data.user.getIdToken();
    })
    .then((token) => {
      return res.json({ token });
    })
    .catch((err) => {
      console.log(err);
```

```
22          return res
23            .status(403)
24            .json({ general: "Wrong credentials, please try again" });
25      });
26      return false;
27  };
```

The final function within the class getAuthenticatedUser() will just grab the details of the currently logged-in user based off the saved token previously set in req.user.handle by logging in or signing up.

Notes and To-do's

The Notes and To-do classes work on the request response system similarly to the login, and both classes hold basic CRUD operations such as Create, Read, Update and Delete and use validation functions from the validators class. As these classes share a very similar model with only a Title and Description being needed by the request object, they are sectioned together within this paper for that purpose as to not add repetition of code from similarly functioning models. It is important to note that the getAll(); functions for both the Note and To-do model only request all the data relevant to the currently logged-in user, this is why it was important to set the req.user.handle in the previous section upon logging in or signing up. Example of the basic CRUD operations within these classes will be given in the following code snippets.

```
1   // Get all the currently logged in User's Notes
2   exports.getAllNotes = (req, res) => {
3     db.collection("notes")
4       .orderBy("createdAt", "desc")
5       .get()
6       .then((data) => {
7         let notes = [];
8         data.forEach((doc) => {
9           if(doc.data().userHandle === req.user.handle)
10          {
11            notes.push({
12              noteID: doc.id,
13              title: doc.data().title,
14              description: doc.data().description,
15              userHandle: doc.data().userHandle,
```

```
16              createdAt: doc.data().createdAt,
17            });
18          }
19        });
20        return res.json(notes);
21      })
22      .catch((err) => console.error(err));
23  };
24  exports.postNewTodo = (req, res) => {
25    const newTodos = {
26      Title: req.body.Title,
27      Description: req.body.Description,
28      userHandle: req.user.handle,
29      createdAt: new Date().toISOString(),
30    };
31    const { valid, errors } = validateTodoData(newTodos);
32    if (!valid) return res.status(400).json(errors);
33
34    db.collection("todos")
35      .add(newTodos)
36      .then((doc) => {
37        res.json({ message: `document ${doc.id} created successfully` });
38        return res.json;
39      })
40      .catch((err) => {
41        res.status(500).json({ error: "Something went wrong." });
42        console.log(err);
43      });
44
45      return false;
46  };
47  // Delete a Todo if the User Owns it
48  exports.deleteTodo = (req, res) => {
49    const document = db.doc(`/todos/${req.params.todoID}`);
50    document
51      .get()
52      .then((doc) => {
53        if (!doc.exists) {
54          return res.status(404).json({ error: "Todo not found" });
55        }
56        if (doc.data().userHandle !== req.user.handle) {
```

```
57          return res.status(403).json({ error: "Unauthorized" });
58        } else {
59          return document.delete();
60        }
61      })
62      .then(() => {
63        res.json({ message: "Todo deleted successfully" });
64        return res.json;
65      })
66      .catch((err) => {
67        console.error(err);
68        return res.status(500).json({ error: err.code });
69      });
70  };
71
72  // Update a note using a noteID
73  exports.updateNote = (req, res) => {
74
75    const document = db.doc(`/notes/${req.params.noteID}`);
76    document.get().then((doc)=>{
77      if(!doc.exists){
78        return res.status(404).json({ error: 'Note not found' });
79      }
80      if(doc.data().userHandle !== req.user.handle){
81        return res.status(403).json({ error: 'Unauthorized' });
82      }
83      else{
84        document.set({
85          title: req.body.title,
86          description: req.body.description,
87          userHandle: req.user.handle,
88          createdAt: new Date().toISOString(),
89        })
90      .then(() =>{
91        res.json("Note updated Successfully");
92        return res;
93      })
94      .catch(err=>{
95        console.log(err);
96        return res.status(500).json({error: err.code});
97      })
```

```
98      return false;
99    }
100
101   })
102
103 };
```

Calendar

Calendars were originally created with different fields in the model but in order to make them work more fluently with the front-end package full-calendar, they were modified to hold the following fields. [**"id", "title", "description", "start", "end", "allDay"**] this is because within full calendar, the fields are labelled the same way and thus the transformation of data through the front-end in order to make it function with this package could be avoided. The calendar model class holds all the basic CRUD operations and works based off of a user ID as well.

```
1  // Get all the Calendar Entries by the current User
2  exports.getAllCalendarEntries = (req, res) => {
3    db.collection("calendars")
4      .orderBy("createdAt", "desc")
5      .get()
6      .then((data) => {
7        let calendars = [];
8
9        data.forEach((doc) => {
10         if (doc.data().userHandle === req.user.handle) {
11           calendars.push({
12             id: doc.id,
13             title: doc.data().title,
14             description: doc.data().description,
15             start: doc.data().start,
16             end: doc.data().end,
17             allDay: doc.data().allDay,
18             userHandle: doc.data().userHandle,
19             createdAt: doc.data().createdAt,
20           });
21         }
22       });
```

```
23
24        return res.json(calendars);
25      })
26      .catch((err) => console.error(err));
27  };
28
29  // Post a New Calendar under the current user
30  exports.postNewCalendar = (req, res) => {
31    const newCalendars = {
32      title: req.body.title,
33      description: req.body.description,
34      start: req.body.start,
35      end: req.body.end,
36      allDay: req.body.allDay,
37      userHandle: req.user.handle,
38      createdAt: new Date().toISOString(),
39    };
40
41    const {valid,errors} = validateCalendarData(newCalendars);
42    if(!valid) return res.status(400).json(errors);
43
44    db.collection("calendars")
45      .add(newCalendars)
46      .then((doc) => {
47        res.json({ message: `document ${doc.id} created successfully` });
48        return res;
49      })
50      .catch((err) => {
51        res.status(500).json({ error: "Something went wrong." });
52        console.log(err);
53      });
54
55    return false;
56  };
```

4.2.3 Listings

Listings encompass the ability for a real estate agent to save details of their various listings for remembrance later on into the Firebase database through the node.js API and the functionality to upload a house image using the

busboy package which parses file data and encrypts it with metadata for the Firebase Storage platform. Details such as the price of a house, the owners names, square feet and more can be added here. When an image is uploaded the mime-type of the requested file upload is added in along with a file path to the storage bucket. As Firebase requires a storage token for file uploading, this is handled here too along with a filename handler that is generated through using a random number function which sets the file name to a number between 1 and 1 million.

```
exports.uploadListingImage = (req, res) => {
  res.header("Access-Control-Allow-Origin", "*");
  const BusBoy = require("busboy");
  const path = require("path");
  const os = require("os");
  const fs = require("fs");

  const busboy = new BusBoy({ headers: req.headers });

  let rand = `${Math.round(Math.random() * 1000000)}`;
  let imageFileName;
  let imageToBeUploaded = {};
  let generatedToken = uuid();

  busboy.on("file", (fieldname, file, filename, encoding, mimetype) => {
    console.log(filename, mimetype);
    if (mimetype !== "image/jpeg" && mimetype !== "image/png") {
      return res
        .status(400)
        .json({
          error: "Wrong File Type Uploaded! -- Only Accepts PNG/JPEG/JPG",
        });
    }
    const imageExtension = filename.split(".")[filename.split(".").length
    ↪  - 1];

    imageFileName = `${rand}.${imageExtension}`;

    const filepath = path.join(os.tmpdir(), imageFileName);
    imageToBeUploaded = { filepath, mimetype };
    file.pipe(fs.createWriteStream(filepath));
```

```
    return false;
  });
  busboy.on("finish", () => {
    admin
      .storage()
      .bucket()
      .upload(imageToBeUploaded.filepath, {
        resumable: false,
        metadata: {
          metadata: {
            contentType: imageToBeUploaded.mimetype,
            firebaseStorageDownloadTokens: generatedToken,
          },
        },
      })
      .then(() => {
        const imageUrl =
        ↪  `https://firebasestorage.googleapis.com/v0/b/${config.storageBucket}
        /o/${imageFileName}?alt=media&token=${generatedToken}`;
        return db.doc(`/listings/${req.params.listingID}`).update({
        ↪  imageUrl });
      })
      .then(() => {
        return res.json({ message: "Image uploaded successfully" });
      })
      .catch((err) => {
        console.error(err);
        return res
          .status(500)
          .json({ error: `something went wrong: ${err.code}` });
      });
  });
  busboy.end(req.rawBody);
};
```

After the above function is ran, the image is uploaded to the listing document which is found by using the listingID provided in the parameters.

4.2.4 Open Houses and Attendees

The Open House section work similarly to the listing section with fewer details of the actual house's features and an added date field for the date when the real estate agent will host the open house. The attendees section of the back end relies on the open house section as attendees are searched by the open house's ID as shown in the below code snippet.

```
// Get all Attendees under the current user
exports.getAllAttendees = (req, res) => {
  db.collection("attendees")
    .orderBy("createdAt", "desc")
    .where('houseID', '==', req.params.houseID)
    .get()
    .then((data) => {
      let attendees = [];
      data.forEach((doc) => {
        if(doc.data().userHandle === req.user.handle)
        {
        attendees.push({
          attendeeID: doc.id,
          full_Name: doc.data().full_Name,
          number: doc.data().number,
          email: doc.data().email,
          contacted: doc.data().contacted,
          interested: doc.data().interested,

          houseID: doc.data.houseID,
          userHandle: doc.data().userHandle,
          createdAt: doc.data().createdAt,
        });
      }
      });
      return res.json(attendees);
    })
    .catch((err) => console.error(err));
};
```

Every attendee requires a house ID on creation which will be gotten through the parameters in the front-end of the applications' dynamic routes.

The attendee class has functions for marking them as **contacted, interested and uninterested** to allow the real estate agent to keep track of whom they have contacted and how interested were they based on their last conversation with this open house attendee.

```
// Update an Attendee as Interested
exports.MarkAsInterested = (req, res) => {
  const document = db.doc(`/attendees/${req.params.attendeeID}`);
  document.get().then((doc) => {
    if (!doc.exists) {
      return res.status(404).json({ error: "Attendee not found" });
    }
    if (doc.data().userHandle !== req.user.handle) {
      return res.status(403).json({ error: "Unauthorized" });
    } else {
        document
        .update({
          interested: true
        })
        .then(() => {
          res.json("Attendee Marked As Interested");
          return res;
        })
        .catch((err) => {
          console.log(err);
          return res.status(500).json({ error: err.code });
        });
    }
  }).catch((err)=>{
    return console.log(err);
  });
  return false;
};

// Update an Attendee as Uninterested
exports.MarkAsUninterested = (req, res) => {
  const document = db.doc(`/attendees/${req.params.attendeeID}`);
  document.get().then((doc) => {
    if (!doc.exists) {
      return res.status(404).json({ error: "Attendee not found" });
```

```
36      }
37      if (doc.data().userHandle !== req.user.handle) {
38        return res.status(403).json({ error: "Unauthorized" });
39      } else {
40          document
41          .update({
42            interested: false
43          })
44          .then(() => {
45            res.json("Attendee marked as UnInterested");
46            return res;
47          })
48          .catch((err) => {
49            console.log(err);
50            return res.status(500).json({ error: err.code });
51          });
52      }
53    }).catch((err)=>{
54      return console.log(err);
55    });
56    return false;
57  };
58
59
60  // Update an Attendee Contacted
61  exports.MarkAsContacted = (req, res) => {
62    const document = db.doc(`/attendees/${req.params.attendeeID}`);
63    document.get().then((doc) => {
64      if (!doc.exists) {
65        return res.status(404).json({ error: "Attendee not found" });
66      }
67      if (doc.data().userHandle !== req.user.handle) {
68        return res.status(403).json({ error: "Unauthorized" });
69      } else {
70          document
71          .update({
72            contacted: true
73          })
74          .then(() => {
75            res.json("Attendee Marked as Contacted");
76            return res;
```

```
77          })
78          .catch((err) => {
79            console.log(err);
80            return res.status(500).json({ error: err.code });
81          });
82      }
83    }).catch((err)=>{
84      return console.log(err);
85    });
86    return false;
87  };
```

Each attendee document also contains basic details for the real estate agent to contact them such as *email, phone number and their full name.*

4.2.5 Utilities

Admin

Admin.js contains a system for basic connection with the fire-base admin package, which will wake the application and give a reference to the current database stored upon fire base. This is used in all the model classes to allow the node.js API to send data and save it to Firebase.

```
1  // Module to export the admin and database components
2  const admin = require("firebase-admin");
3
4  admin.initializeApp();
5
6  const db = admin.firestore();
7
8  module.exports = { admin, db };
```

Config

The file config.js holds the API keys and various URLs that are instantiated when **"firebase init"** is called on the back-end, without this Firebase would deny all interactions from node.js.

FBAuth.js (Firebase Auth)

This class is in charge of authenticating the user and managing authenticated routes as a middleware. It checks the user's authorization header from the front-end which is later sent through Axios and checks if their Bearer Authentication Token is valid upon the Firebase server. If it is, it will set the request user handle to the ID of the user who owns that token on the Firebase authentication service server, however if the token is expired or illegitimate, it will deny the user and return a 403 status code error to notify the user of this.

```
1  // Checks if user is authorized based off their token
2  module.exports = (req, res, next) => {
3    let idToken;
4    if (
5      req.headers.authorization &&
6      req.headers.authorization.startsWith("Bearer ")
7    ) {
8      idToken = req.headers.authorization.split("Bearer ")[1];
9    } else {
10     console.error("No Token Found");
11     return res.status(403).json({ error: "Unathorized Access" });
12   }
13
14   admin
15     .auth()
16     .verifyIdToken(idToken)
17     .then((decodedToken) => {
18       req.user = decodedToken;
19       return db
20         .collection("users")
21         .where("userID", "==", req.user.uid)
22         .limit(1)
23         .get();
24     })
25     .then((data) => {
26       // If Token on the userID is equal to the current userID then
         ↪  progress them through
27       // the system
28       req.user.handle = data.docs[0].data().handle;
29       return next();
```

```
30      })
31      .catch((err) => {
32        console.error("Error while veryifying token", err);
33        return res.status(403).json(err);
34      });
35      return false;
36  };
```

This class is exceedingly important to add security to the application and prevent malicious intent, as real estate agents are often working with the personal details of customers and will keep personal notes themselves upon the application, it was imperative to have this class play a huge role within the back-end of the project as to work in ensuring privacy for the users.

Validators

The validators class contains a list of validation functions for the project to alert the user if they were to for example try to enter empty fields into the database or weak passwords. It contains a function for each model and gives personalized error messages for each and disallows the continuation of a POST request as long as the validation functions are triggered and errors are returned. It is important to know that the email is valid and password is not weak validators make use of regular expression (Regexp) string matching and as such some of these Regexp strings are exceedingly long and cannot be displayed through the minted package on latex even with line breaks enabled. This is the reason why the show email is valid function has not been given as a code snippet.

```
1   // Checks if a String is Empty
2   const isEmpty = (string) => {
3     if (string.trim() === "") return true;
4     else return false;
5   };
6
7   const isTooWeak = (password) => {
8     //eslint-disable-next-line
9     const regEx =
10
    ↪   /^(?=^.{8,}$)(?=.*\d)(?=.*[!@#$%^&*]+)(?![.\n])(?=.*[A-Z])(?=.*[a-z]).*$/;
```

```
11    if (password.match(regEx)) return true;
12    else return false;
13  };
14
15  // Validates Signup Data Using Email and Password
16  exports.validateSignupData = (data) => {
17    // setup an errors object
18    let errors = {};
19
20    // if the email is empty or is not a email return a error
21    if (isEmpty(data.email)) {
22      errors.email = "Must not be empty!";
23    } else if (!isEmail(data.email)) {
24      errors.email = "Must be a valid email address!";
25    }
26
27    if (isEmpty(data.password)) errors.password = "Must not be empty!";
28    if (data.password !== data.confirmPassword)
29      errors.confirmPassword = "Passwords must match!";
30    if (!isTooWeak(data.password)) {
31        errors.password = "Password is to weak!";
32      }
33    if(isEmpty(data.handle)) errors.handle = "Must not be empty!";
34
35    return {
36      errors,
37      valid: Object.keys(errors).length === 0 ? true : false
38    };
39  };
40
41
42  // Validates Calendar Entries
43  exports.validateCalendarData = (data) => {
44    let errors = {};
45    if(isEmpty(data.title)) errors.title =
      ↪  "Please Enter a Title for the Event";
46    if(isEmpty(data.description)) errors.description =
      ↪  "Please Enter a Description for the Event";
47    if(isEmpty(data.start)) errors.start = "Please Enter a Start Date";
48    return {
49      errors,
```

```
    valid: Object.keys(errors).length === 0 ? true : false,
  };
}
```

4.3 Front-End Design

4.3.1 App (App.js)

App.js is the centrepiece of the single page architecture that React.js incorporates. This class holds the Route Information, Authenticated Route Information and how each page should be accessed through the URL. For example to get to the notes page. A route with /notes is created like so:

```
<Route exact path="/notes" component={Notes} />
```

Authenticated routes which is performed by the AuthRoute.js class under the Util folder on the Front-End can be written as follows on App.js

```
 // Example of Authentication Routes within App.JS,
 // these ensure the user is logged in and authenticated
 // before viewing the routes as their token is required to
 // view the notes associated with the account.
<AuthRoute exact path="/createNote" component={CreateNote} />
<AuthRoute exact path="/notes/edit/:noteID" component={EditNote} />
```

The App.js class is also in charge or providing the overall Material UI Theme to the rest of the application, in this case a deep purple chosen with a secondary style of Red.

```
// Change the default MUI theme
const theme = createMuiTheme({
  palette: {
    primary: {
      main: deepPurple[500],
    },
    secondary: {
```

```
      main: "#f44336",
    },
    typography: {
      fontFamily: 'Quicksand',
    },
    contrastThreshold: 3,
    tonalOffset: 0.2,
  },
});
```

Authentication and User Sign out's have functions within App.js to ensure that if the user wishes to log out or should be logged out due to a token expiration, that the token saved on the user's local storage upon their browser is cleared and the state of authentication is set to false. If a token is expired or invalid, the user will be re-directed to the login page and hence need to re-acquire a new JWT *(JSON Web Token)* token by logging in.

```
// Set the Token and Check it's expire date.
// If the current token is older than the expiry date, log the user out.
const token = localStorage.FBIdToken;
if (token) {
  const decodedToken = jwtDecode(token);
  console.log(decodedToken);
  if (decodedToken.exp * 1000 < Date.now()) {
    window.location.href = "/login";
    authenticated = false;
  } else {
    authenticated = true;
    axios.defaults.headers.common["Authorization"] = token;
  }
}

// Signout the User and Remove their authorization status
const signout = () => {
  try {
    localStorage.clear();
    authenticated = false;
    axios.defaults.headers.common["Authorization"] = null;
  } catch (err) {
    console.log(err);
```

```
24    }
25    window.location.replace("/");
26    return null;
27  };
```

During deployment Axios is set with a **default baseURL** to call the API in a more seamless manner. For example when retrieving notes using Axios, rather than typing the full URL of the API for each request *"https://europe-west2-re-agent-dashboard-22410.cloudfunctions.net/api/notes"*, the setting of the baseURL within Axios makes it so that only the routes after the API part of the URL need to called *"/notes"*.

4.3.2 Home Page

The home page contains some simple filler code with a ¡h1¿ tag to inform the user they are welcome to the website.

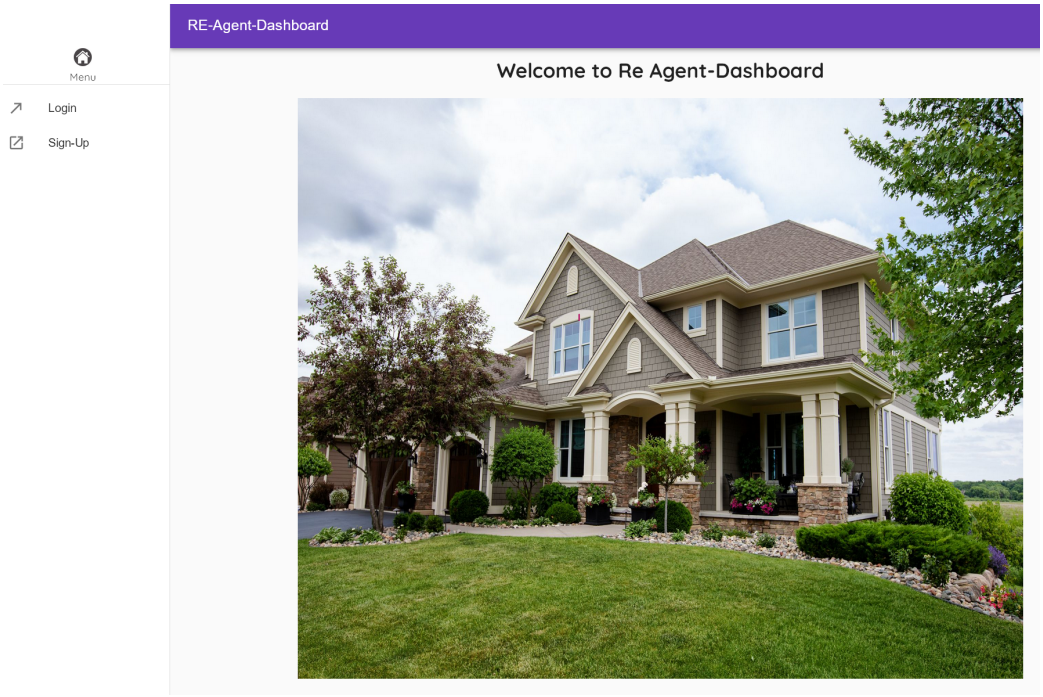

Figure 4.2: Home Page

4.3.3 Responsive Drawer

Displayed on the left of the home page screenshot above, is the responsive drawer. This drawer works using Material UI components and has a state based on the screen size of the device such that it becomes a hamburger menu[43] on mobile devices and remains open to the left on larger screens such as laptops or desktops. Below is an image of the drawer in desktop and mobile states.

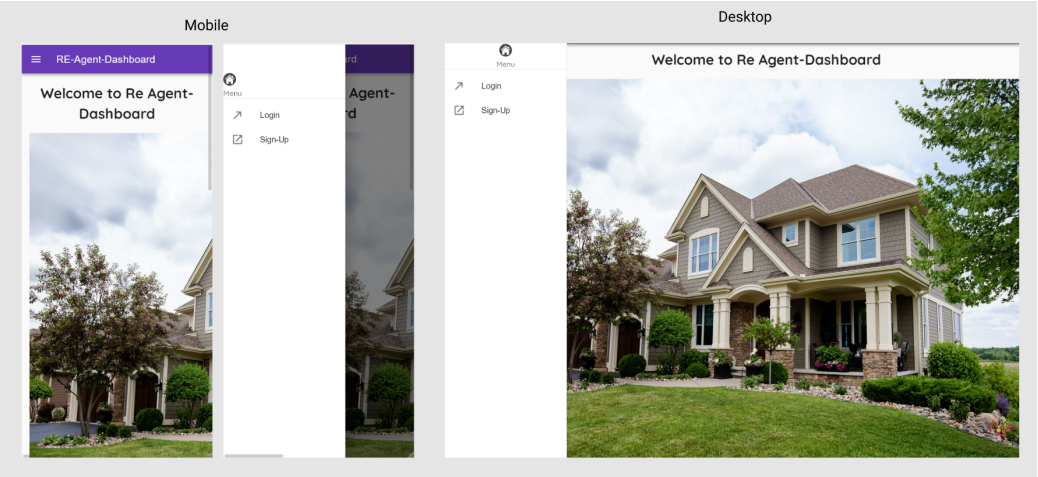

Figure 4.3: Responsive Drawer (Mobile,Desktop)

The list of items on the drawer, which work as links to routes on the application will change based on the user's logged in status. In the prior figures, the user is logged out but when logged in the state of the drawer will change to following:

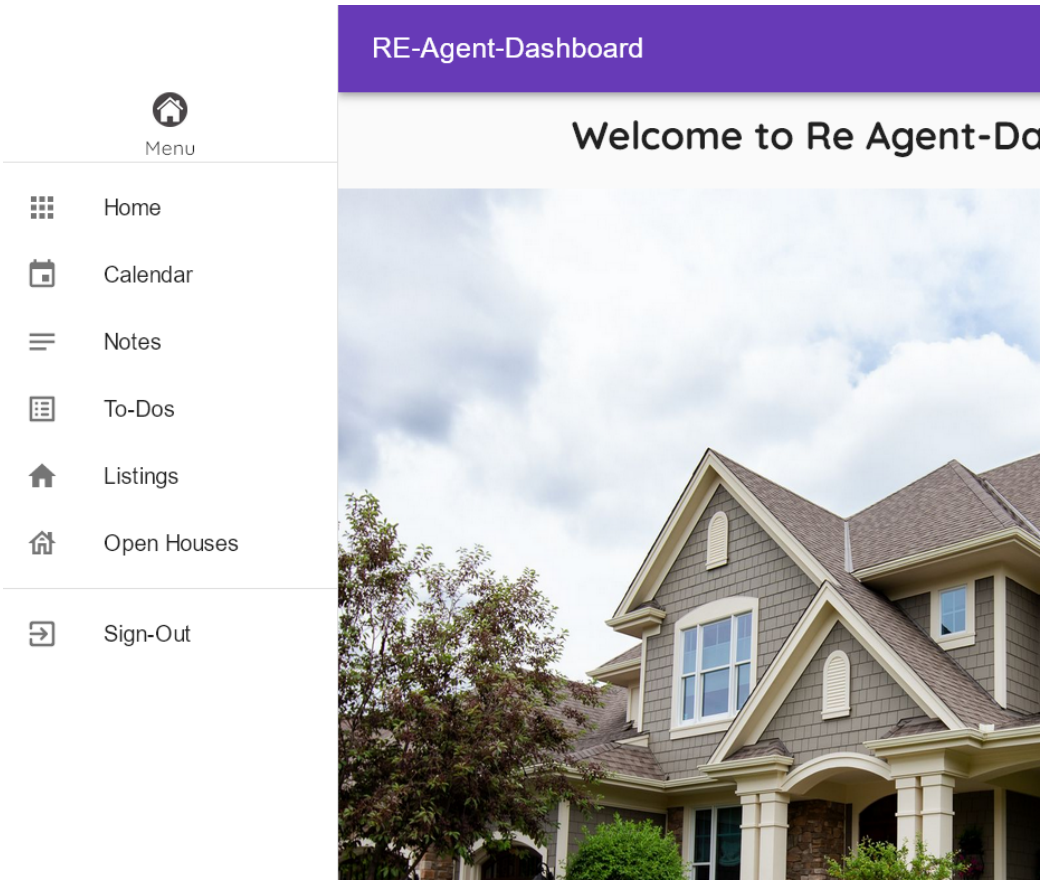

Figure 4.4: Drawer: User Logged In State

4.3.4 Login and Sign-Up

Login

The login page consists of a Username and Password Text Field provided by Material UI. This information is grabbed inside the state of the class on submission and then sent to the back-end API under the login route. The state is dynamically changed as the User types into the field using the handleChange(); function event caller. On a successful login a token is retrieved from the back-end and set within the browser local storage and later on this token is checked within App.js and the user is finally authenticated if they are valid. However, if a user enters the wrong details, the login page's relevant validation function on the back-end is triggered and a warning to tell the user they are using the wrong-credentials is displayed.

```
handleSubmit = (event) => {
    event.preventDefault();
    this.setState({
      loading: true,
    });
    const userData = {
      email: this.state.email,
      password: this.state.password,
    };
    axios
      .post("/login", userData)
      .then((res) => {
        console.log(res.data);
        localStorage.setItem("FBIdToken", `Bearer ${res.data.token}`);
        this.setState({
          loading: false,
        });
        window.location.reload();
      })
      .catch((err) => {
        this.setState({
          errors: err.response.data,
          loading: false,
        });
      });
  };
  handleChange = (event) => {
    this.setState({
      [event.target.name]: event.target.value,
    });
  };
```

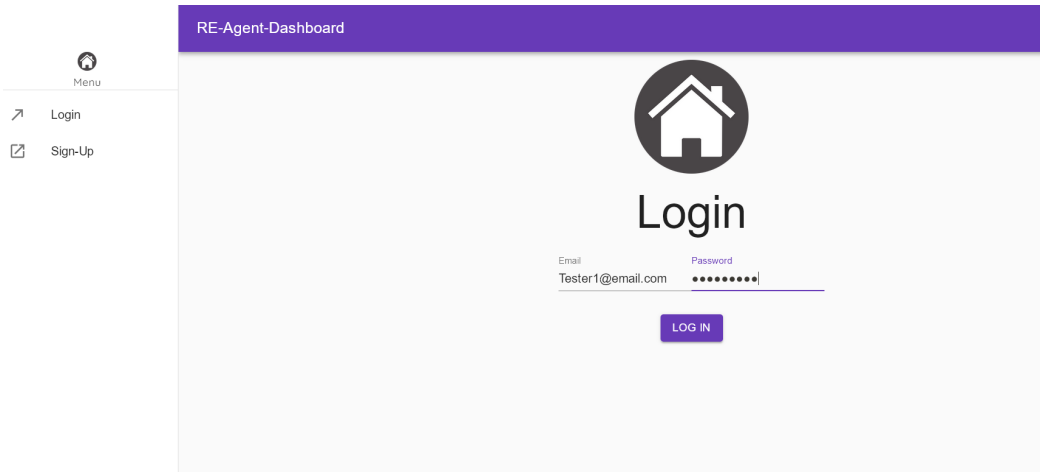

Figure 4.5: Login Page

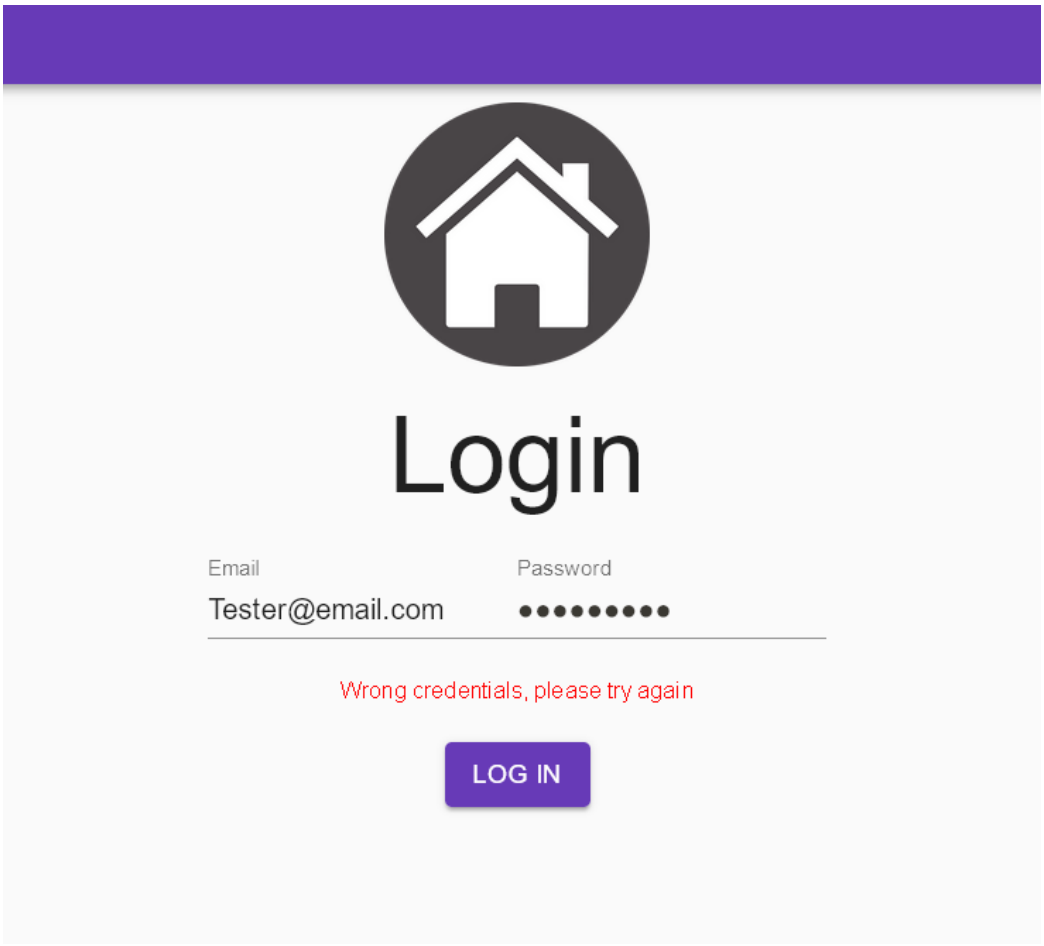

Figure 4.6: Login: Failed "Wrong Credentials"

Sign-Up

The sign-up page consists of further details and validation as the email is checked if it's a valid email type, i.e. contains a @ symbol and top level domain names such as .com, .net, .org. The password is also validated to ensure that it is strong in such that it contains unique symbols, capitalized and lower case characters along with some numbers. If any of the fields are empty a warning will come up for that field to warn the user that it cannot be empty. All of these validation checks are important to ensure that the User makes a secure and correct account that can work well with the back-end server and the Firebase Authentication Service. On a successful sign-up, an alert will be shown to the user to inform them before redirecting them to the login page where they can sign in. The background detail of how the sign-up pages text fields work has similarities to that of the login page such that

it uses Material UI and onChange() event functions with a final submission function that passes the data to the back-end sign up route and receives a response based on the success of the sign-up attempt.

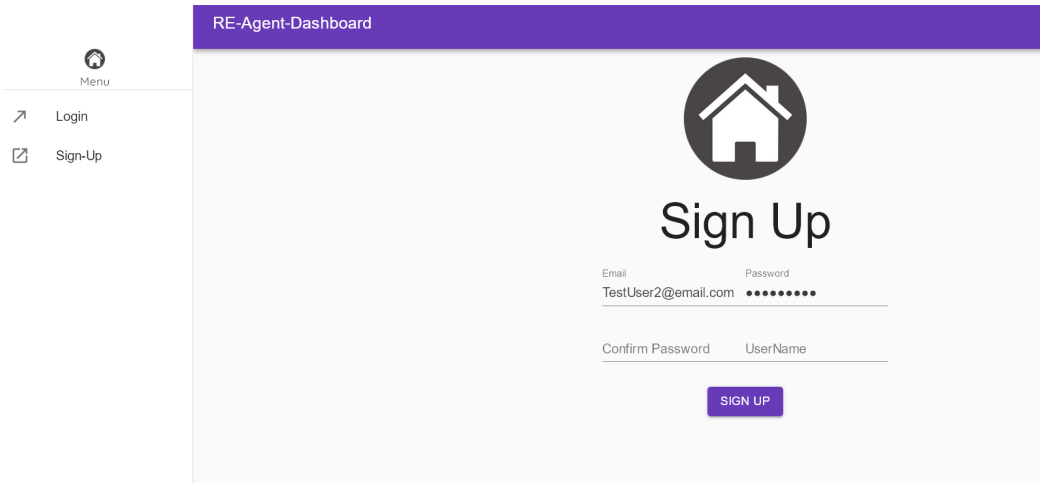

Figure 4.7: Sign-up Page

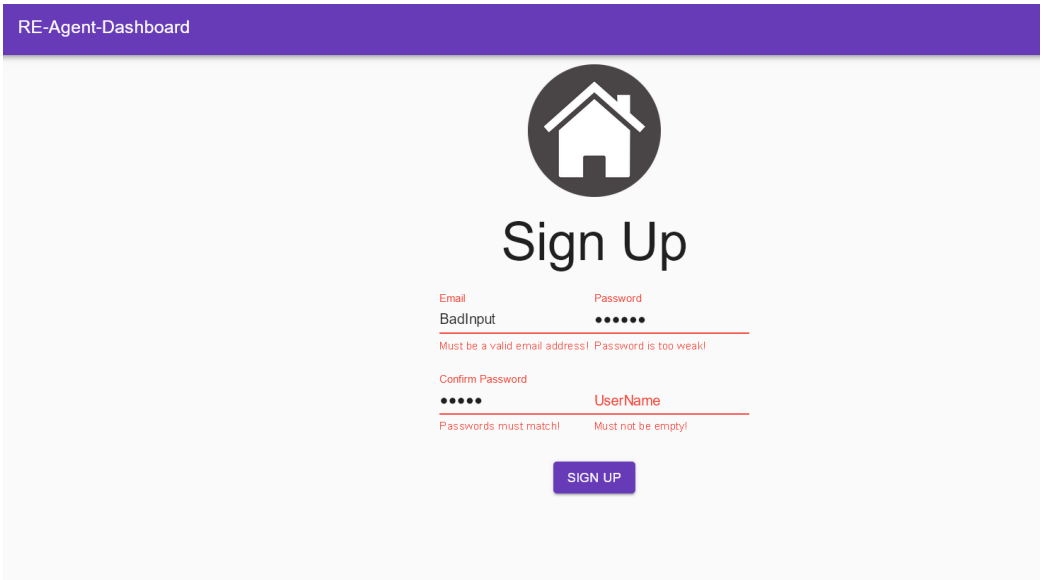

Figure 4.8: Sign-up Page: Incorrect or Weak Inputs

4.3.5 Notes and Todo Components

Notes

The notes section of the front-end contains four classes.

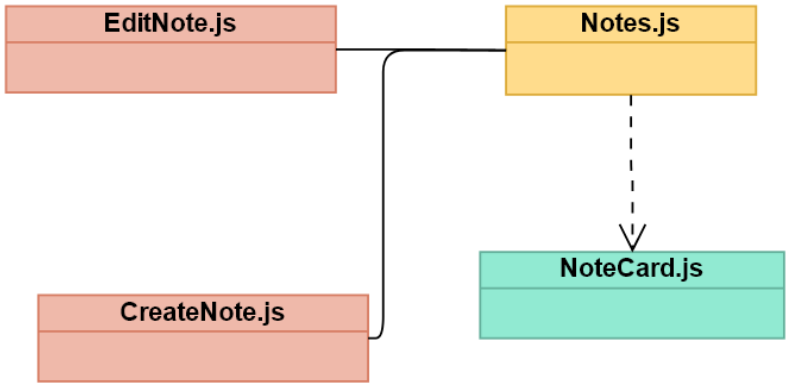

Figure 4.9: Notes Class Diagram

- 1. Notes
- 2. Note Cards
- 3. Create Note
- 4. Edit Note

On a user's entry to the notes page, the Notes.js functional class will be rendered, this class contains a state for the notes which is set through the useEffect(); clause provided by React. This useEffect(); method uses Axios to call a GET request to the /notes path on the back-end which is then provided to the state if successful. There is also a method for handling deletion of a note, which sends a delete request with the currently selected note's ID to the DELETE route on the node.js server and then filters out the deleted note such that the user does not need to refresh the page before seeing the change. In order to add a new note, there is a Material UI Fab Plus Icon

at the bottom of the page to redirect the user to the /createNotes route on the front-end which will be discussed after the Note Cards subsection. All the note data from the state is sent into the ¡NoteCard¿ HTML tag through props which is actually a class that holds a state of its own and will be discussed in the next subsection. The reason for using a card based system for the notes, to-do's, listings and open houses is because it is common in Google Material Design layouts and looks visually appeasing to users.

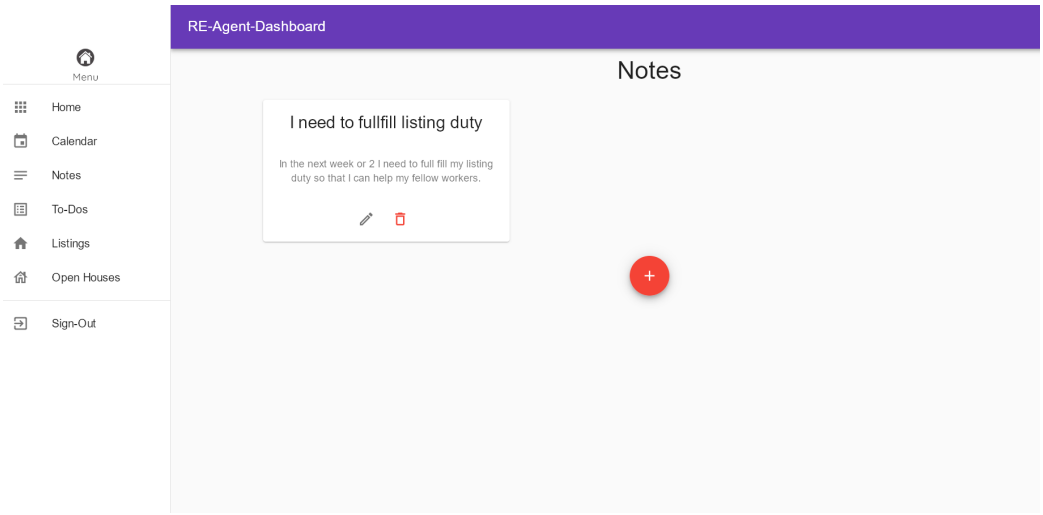

Figure 4.10: Notes Page

Note Cards are created using Material UI components and the props of the note card functional class takes in a Note object and a handleDelete object as properties, both of which are passed in by the previous section's Note.js class. The details of the individual note's state are rendered onto the card layout along with the delete button which has a confirmation dialog when it is pressed to ensure that the user does not accidentally delete an important note. Refer to *Figure 4.10* to see the note card's visual layout.

The **Create Note** and **Edit Note** pages are related in design and both have a state containing the note description and title, along with any errors that may occur during their POST/PUT requests. The key difference between them being that Edit Note retrieves the noteID from the URL parameters and requests the note related to that ID's data from the back-end using Axios and then sets the text fields accordingly to the data retrieved.

```
1 {
2     // The Submit Handler within the Note Creation Class
```

```
3       // Similar in design within the Edit Note Class
4       handleSubmit = (event) => {
5           event.preventDefault();
6           this.setState({
7             loading: true,
8           });
9           const newNote = {
10            title: this.state.title,
11            description: this.state.description,
12          };
13          axios
14            .post("/notes", newNote)
15            .then((res) => {
16              console.log(res.data);
17              this.setState({
18                loading: false,
19              });
20              this.props.history.push("/notes");
21            })
22            .catch((err) => {
23              this.setState({
24                errors: err.response.data,
25                loading: false,
26              });
27            });
28        };
29
30      // The Component did mount within Edit Note that retrieves the note
        ↪  details
31      // By the ID provided in the URL parameters and sets the state
        ↪  accordingly.
32    componentDidMount() {
33          document.title = "Edit Note";
34          axios
35            .get("/notes/"+this.props.match.params.noteID)
36            .then((response) => {
37              this.setState({ title: response.data.title,
38                              description: response.data.description });
39            })
40            .catch((error) => {
41              console.log(error);
```

```
42              });
43          }
```

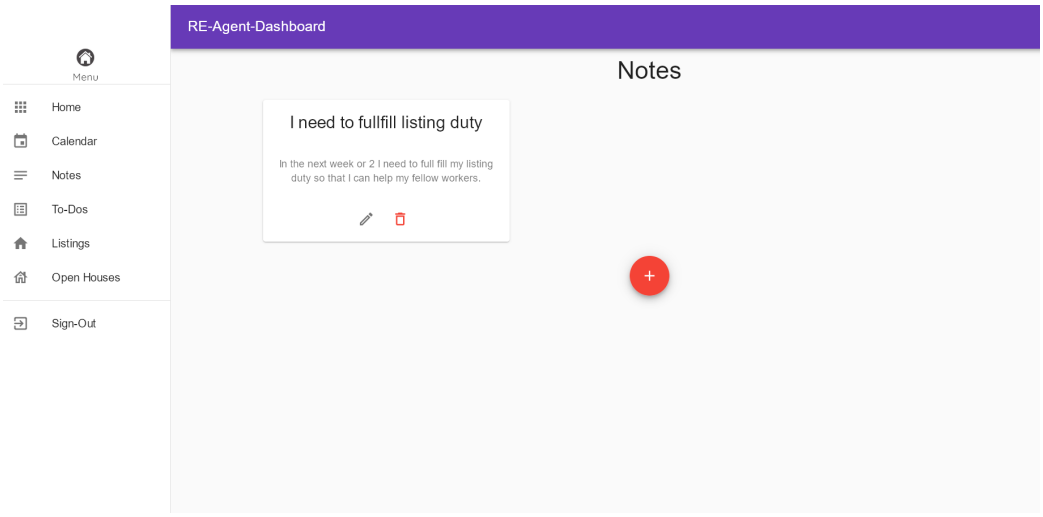

Figure 4.11: Create Note Page

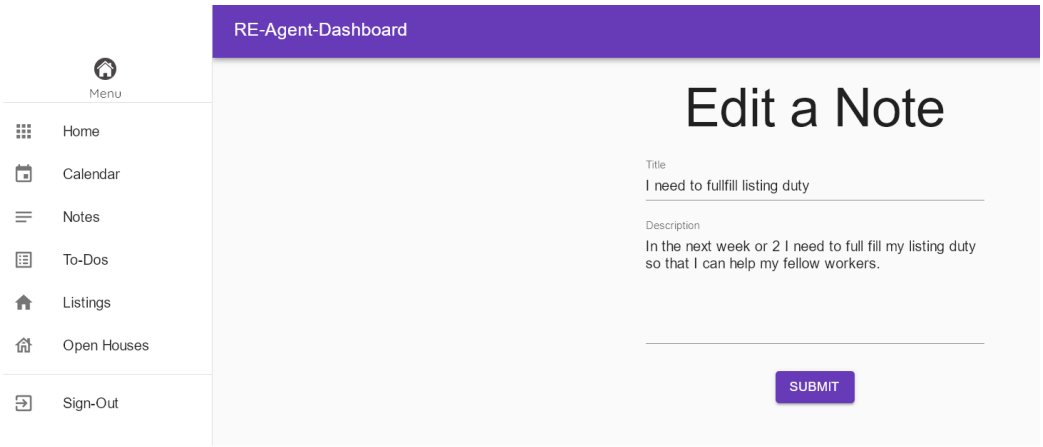

Figure 4.12: Edit Note Page

Todos

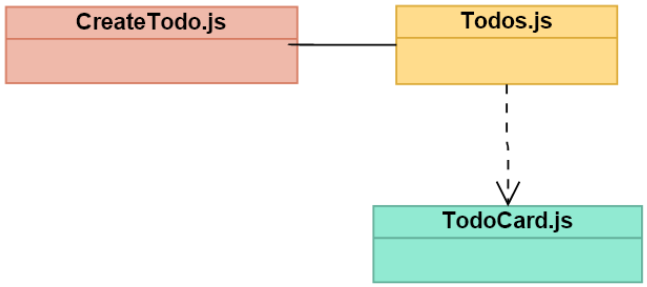

Figure 4.13: To-do's Class Diagram

As stated in the back-end section, To-do's share a familiar data model to notes, due to this extensive details will not be given on the class's code. The To-do section of the Front End allows the user to *Create a To-do*, *View a To-do in a Card List Format* and *Set a To-do as Completed (delete it)*.

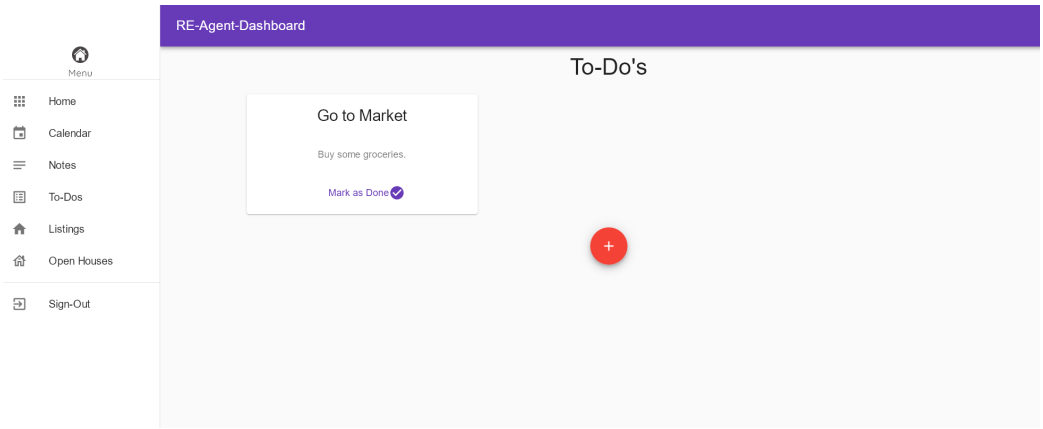

Figure 4.14: To-do's Page

4.3.6 Calendar

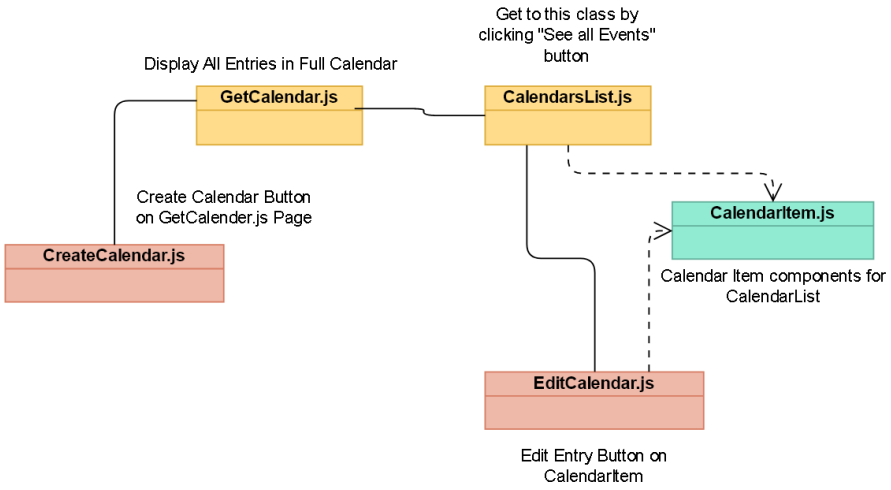

Figure 4.15: Calendar Class Diagram

Calendar Main Page

The Calendar page takes in data from the node.js back end and as described earlier these back-end data fields are modeled in a way to work with the Full Calendar Package for React.js. A state for the events and the if the device is mobile is set up and then an EventListener which is based on window resize(). Then the component is checked if it was mounted and the calendar data is retrieved from the server after this the Full Calendar module is called, and the events are passed in through the events' property on FullCalendar, these are then rendered based on the view type currently used by the Full Calendar. There are four view types that were added into the application which were provided from the Full-Calendar Package **"dayGridMonth"**, **"timeGridWeek"**, **"timeGridDay"** and **"listWeek"**. The default view being "listWeek" which is the best for both desktop and mobile formats along with readability, as some of the other views are not fully responsive on very small screens and hence event details are too condensed to be viewed correctly.

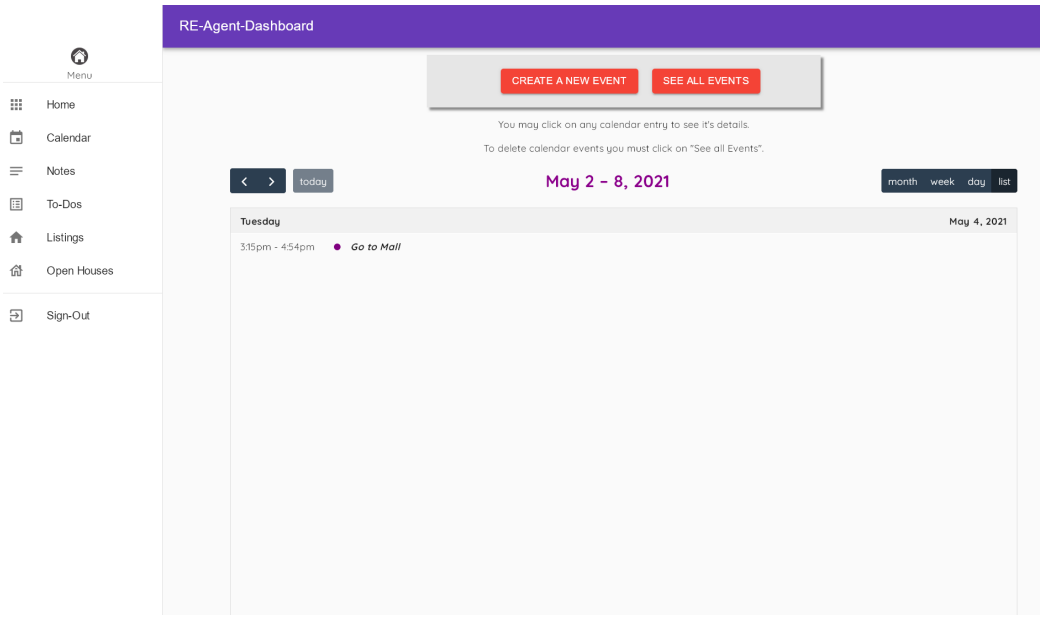

Figure 4.16: Calendar: List View

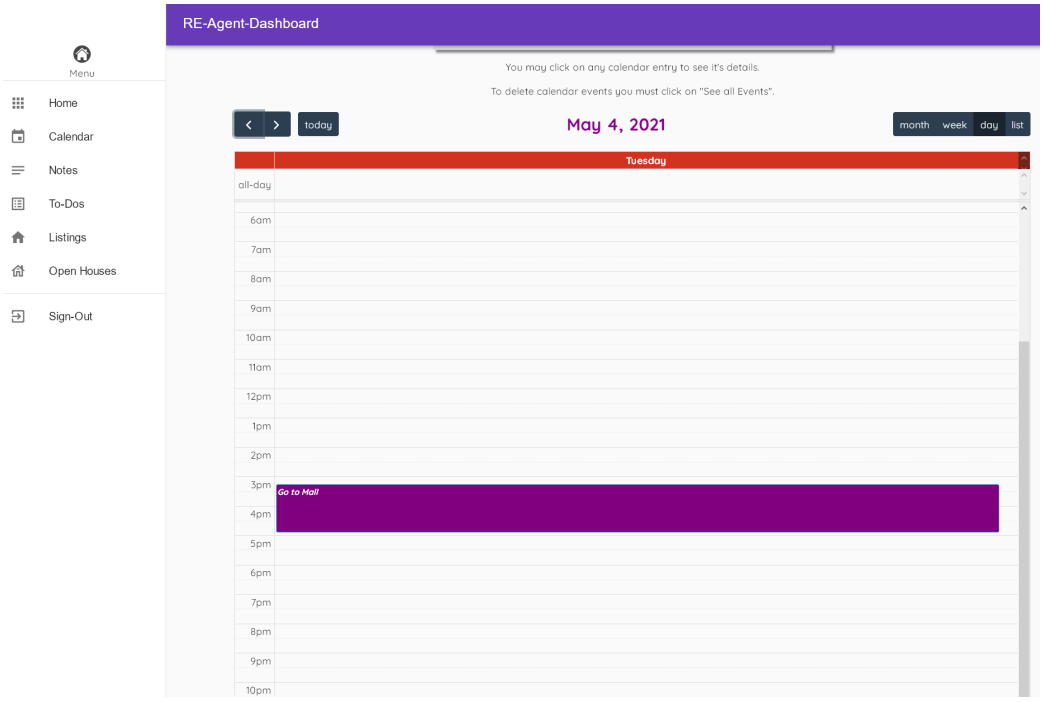

Figure 4.17: Calendar: Day View

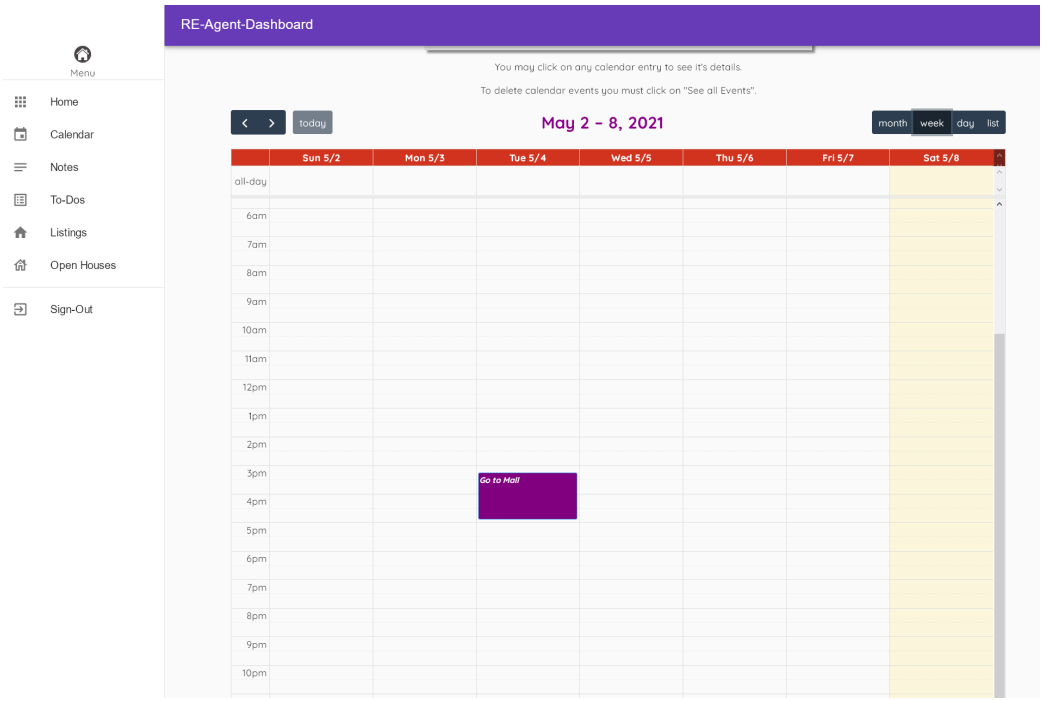

Figure 4.18: Calendar: Week View

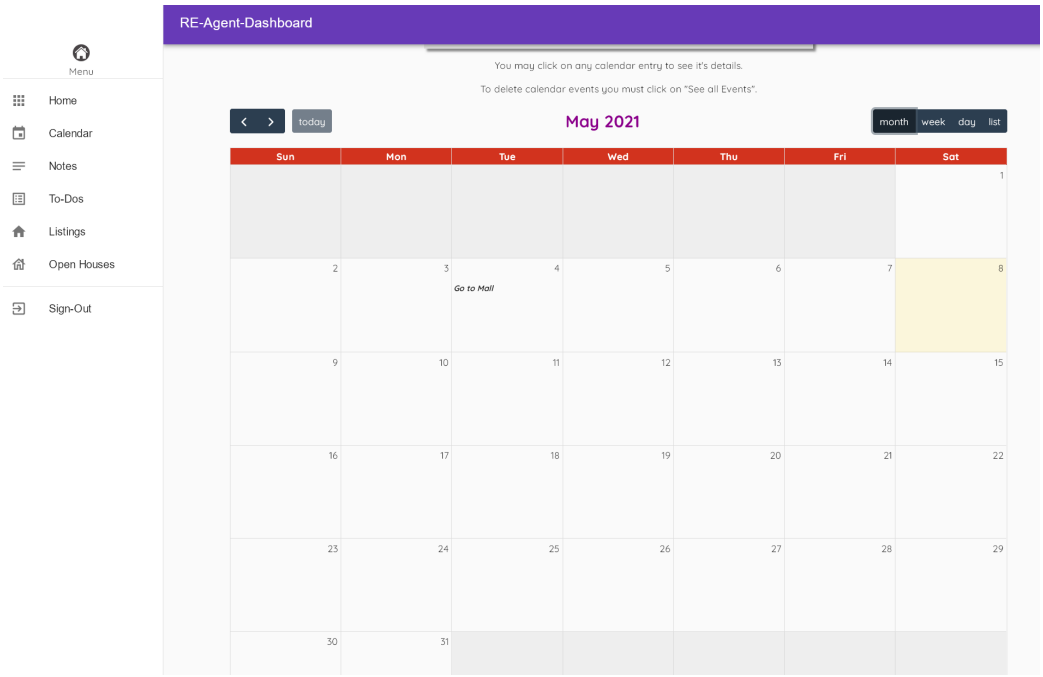

Figure 4.19: Calendar: Month View

Calendar: See All Events

Upon clicking the "See all Events" Button on the main calendar page, the user is redirected to a page that contain all of their calendar events and options to go to each events respective edit and deletion routes. This class works by using the GET route from the API within the Calendars List JavaScript file and assigning each element to a functional class called Calendar Item, these calendar items will then span the width of the page and show each events details.

Code Snippet from Calendar Item:

```
function CheckMark(input) {
    if (input) {
      return 'U+2705';
    } else return 'U+274C';
  }
  return (
    <div>
      <Card elevation={3}>
        <CardHeader
          title={calendar.title}
          style={{ borderBottom: "2px solid lightgray" }}
        />
        <CardContent>
          <Typography variant="body1" color="textSecondary">
            <b>Start:</b> {parseDate(calendar.start)}
            <br />
            <b>End:</b> {parseDate(calendar.end)}
            <br />
            <b>All Day:</b> {CheckMark(calendar.allDay)}
            <br />
            <br />
            <b style={{textDecoration:
            ↪  "underline"}}>Description:</b> <br />
            {calendar.description}
          </Typography>
        </CardContent>
        <CardActions style={{ paddingLeft: "45%" }}>
          <IconButton
            title="Edit Calendar"
            aria-label="Edit Calendar"
```

```
            href={"/calendars/edit/" + calendar.id}
          >
            <CreateIcon />
          </IconButton>
          <IconButton
            color="secondary"
            title="Delete"
            aria-label="Delete"
            onClick={() => {
              if
              ↪  (window.confirm("Are you sure you wish to delete this item?"))
                handleDelete(calendar.id);
            }}
          >
            <DeleteOutlinedIcon />
          </IconButton>
        </CardActions>
      </Card>
    </div>
  );
    );
```

Code Snippet from CalendarList.js (”Show all Entries”):

```
export default function CalendarsList() {
  const [calendars,setCalendars] = useState([])

  useEffect(() => {
    document.title = "Calendar List";
    axios
      .get("/calendars")
      .then((response) => {
        setCalendars(response.data);
      })
      .catch((error) => {
        console.log(error);
      });
  },[])

  const handleDelete = async (id) => {
```

```
17      await axios.delete("/calendars/" + id)
18      .then(res =>{
19        console.log(res);
20        const newCalendars = calendars.filter(calendar => calendar.id !==
          ↪  id)
21        setCalendars(newCalendars);
22      })
23    }
24
25    // List the Listings in a Card Grid with Options included
26    return (
27      <Container>
28       <Typography variant="h4">Calendar List of Events</Typography>
29        <Grid container spacing={3} style={{paddingTop:"20px"}}
          ↪  alignItems={"center"} alignContent={"center"}>
30          {calendars.map(calendar =>(
31            <Grid item key={calendar.id} xs={12} md={12} lg={12}>
32              <CalendarItem calendar={calendar}
                ↪  handleDelete={handleDelete}/>
33            </Grid>
34          ))}
35        </Grid>
36
37        <Button
38          component={Link}
39          to={`/calendars/`}
40          variant="contained"
41          color="secondary"
42          style={{ marginTop: "20px" }}
43        >
44          Go Back
45        </Button>
46      </Container>
47    )
48  }
49  export default CalendarList;
```

Calendar: Create and Edit

To create a calendar a user may do it through the main calendar page or if they wish to update an already created calendar they can do it through the ”See all events view”. These classes work in the same way as previous create and update classes for other models except the calendar model data is prevalent within them. The Material UI Pickers - DateTimePicker Module is used within the creation and editing of calendar events to allow the user to select a date and time for their events in a UI that should be familiar for users who have previously used android based devices.

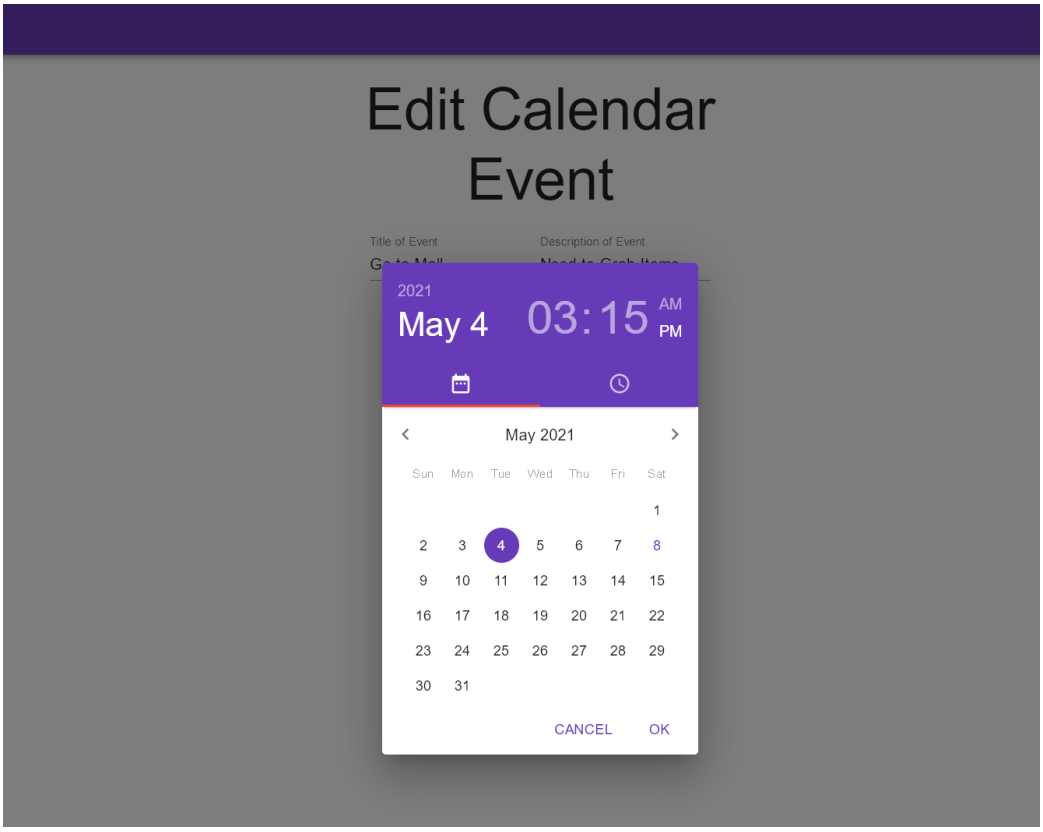

Figure 4.20: Calendar: DateTimePicker within Create/Edit

4.3.7 Open Houses

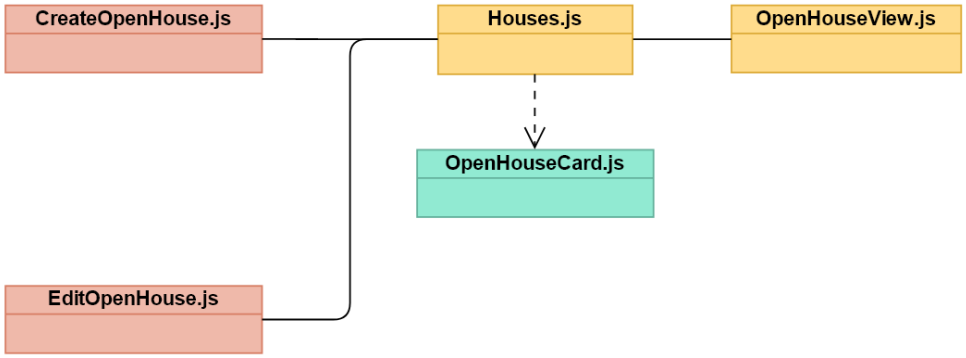

Figure 4.21: Open Houses Class Diagram

Open Houses Page (Houses.js)

The Open Houses are rendered in a card format just like within the notes and To-do's section except contain an image and Material-UI icon buttons to allow the user to **View**, **Edit**, **See Attendees at that Open House** and **Delete** the open house entries relevant to that user along with a figure plus icon to add further open houses.

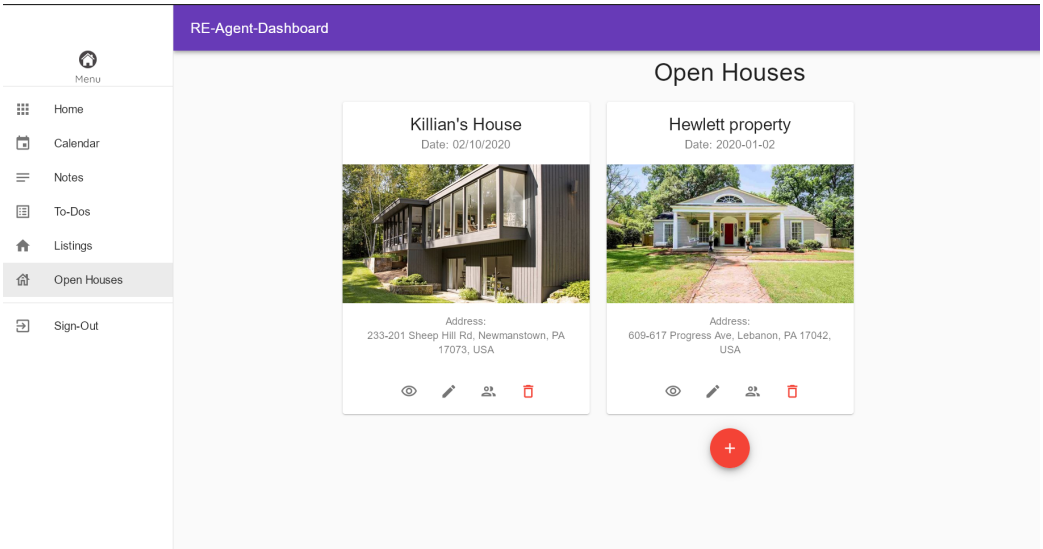

Figure 4.22: Open House's Main Page

Open House Cards

The Open House cards take in details from the Houses.js class and use the same properties are the other card classes, being that of the model and a handleDelete() function.

```
export default function OpenHouseCard({ house, handleDelete }) {
  document.title = "House Cards";
  return (
    <div>
      <Card elevation={1} >
        <CardHeader title={house.property_Name}
        ↪  subheader={"Date: "+house.date}/>
        <CardMedia style = {{ height: 0, paddingTop: '56%'}}
        ↪  image={house.imageUrl}/>
        <CardContent>
          <Typography variant="body2" color="textSecondary">
            Address: <br />{house.address}
          </Typography>
        </CardContent>
        <CardActions style={{paddingLeft:"20%"}}>
        <IconButton title="View Open House Details"
        ↪  aria-label="View Open House Details"
        ↪  href={"/open_houses/"+house.houseID}>
          <VisibilityOutlinedIcon/>
        </IconButton>
        <IconButton title="Edit Open House" aria-label="Edit Open House"
        ↪  href={"/open_houses/edit/"+house.houseID}>
          <CreateIcon/>
        </IconButton>
        <IconButton title="View Attendees" aria-label="View Attendees"
        ↪  component={Link}
          to={`/attendees/${house.houseID}`}>
          <PeopleOutlineOutlinedIcon/>
        </IconButton>
        <IconButton color="secondary" title="Delete" aria-label="Delete"
        ↪  onClick={() =>
        ↪  {if(window.confirm('Are you sure you wish to delete this item?'))
        ↪  handleDelete(house.houseID) }}>
          <DeleteOutlinedIcon />
        </IconButton>
```

```
27              </CardActions>
28            </Card>
29          </div>
30      );
31  }
```

Open House View

Upon clicking into the view of an open house. The user will be greeted by the house image along with all the open house details that were saved into that entry. Such as the property name and address, date and seller's names. To the bottom right of the page there is a map showing the location of where the open house is located on a geographical map, this was performed using the Google Embedded Maps API and passing in the house address into the API's URL query parameters as shown below.

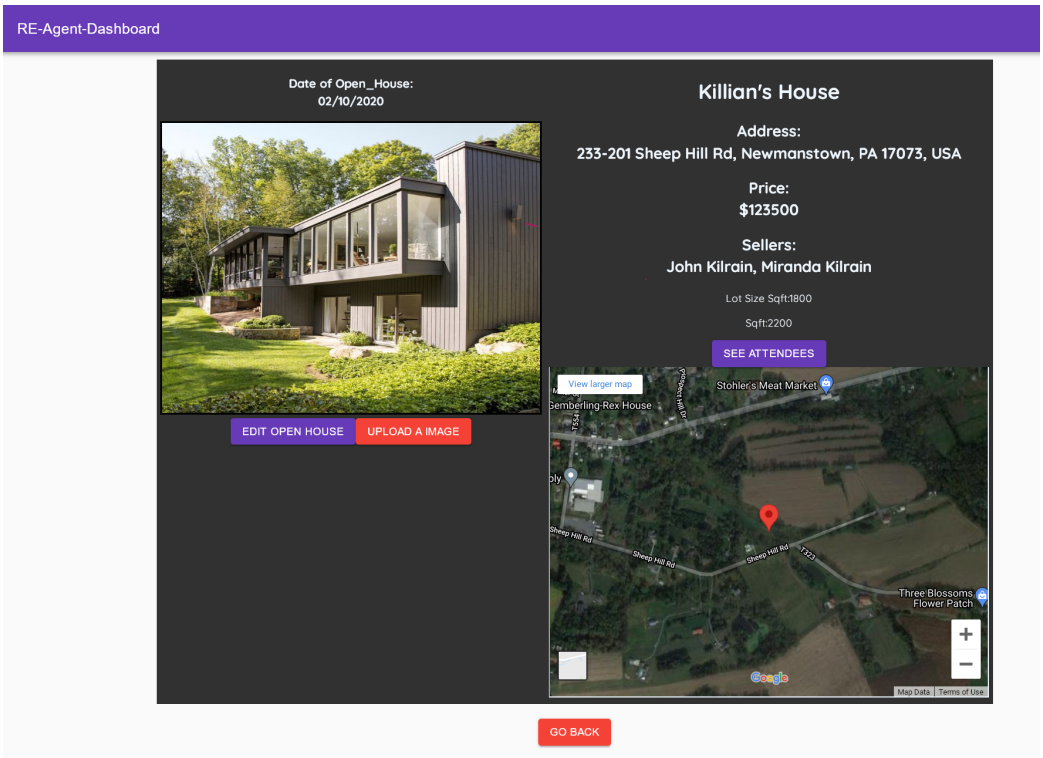

Figure 4.23: Open House View: Google Maps Integration

```
// Google Maps Embedded API
 <iframe
            width="600"
            height="450"
            style={{ borderTop: "30px" }}
            loading="lazy"
            allowfullscreen
            title="Open House on Map"
            src={
              `https://www.google.com/maps/embed/v1/place?
              key=AIzaSyA9XbXYjAlgalSifIKaEzqaPNunDjrVfYw&q=${house.address}`
            }
          ></iframe>
```

A useful function part of this API is that the user can choose to click on the ”View Larger Map” button and be brought to the Google Maps Application online and get directions to the location. This is increasingly useful for real estate agents who may have many clients for whom they have forgotten the directions to some of the clients homes.

It is also possible to upload an image file type through the open house view to change the default image for open houses. This is done using a form dialog provided by Material UI and some custom JavaScript classes. There were difficulties with implementing this functionality as the Storage Bucket section of Firebase requires its own cross-region based CORS rules which can only be applied through the Google Cloud platform and the Google Cloud Command Line Interface linked to the project.

```
 const fileSelectedHandler = (event) => {
    let file_size = event.target.files[0].size;
    // Give warning if user tries to upload file larger then 5Mb
    if (file_size > 5242880) {
      alert("File Size to Large (Limit 5MB)");
      setFile("");
    } else {
      setFile(event.target.files[0]);
    }
  };

  const fileUploadHandler = () => {
```

```
13      const fd = new FormData();
14      fd.append("image", File);
15      axios
16        .put(
17          `/open_houses/image/${props.id}`,
18          fd
19        )
20        .then((res) => {
21          console.log(res);
22          alert("File Uploaded Successfully");
23          setOpen(false);
24          window.location.reload();
25        })
26        .catch((res) => {
27          alert(res + "\n File needs to be in JPEG/PNG Format");
28          window.location.reload();
29        });
30    };
31
32    <Dialog
33              open={open}
34              onClose={handleClose}
35              aria-labelledby="form-dialog-title"
36            >
37              <DialogTitle id="form-dialog-title">Upload a
                ↪  Image</DialogTitle>
38              <DialogContent>
39                <DialogContentText>
40                  <p>Upload a 5Mb or less image(png/jpeg)</p>
41                </DialogContentText>
42                <RaisedButton
43                  containerElement="label"
44                  label="My Label"
45                  for="image_uploads"
46                >
47                  <input
48                    type="file"
49                    title=" "
50                    onChange={fileSelectedHandler}
51                    accept="image/png, image/jpeg"
52                    id="image_uploads"
```

```
53                    name="image_uploads"
54                  />
55                </RaisedButton>
56              </DialogContent>
57              <DialogActions>
58                <Button onClick={handleClose} color="primary">
59                  Cancel
60                </Button>
61                <Button onClick={fileUploadHandler} color="primary">
62                  Upload
63                </Button>
64              </DialogActions>
65            </Dialog>
```

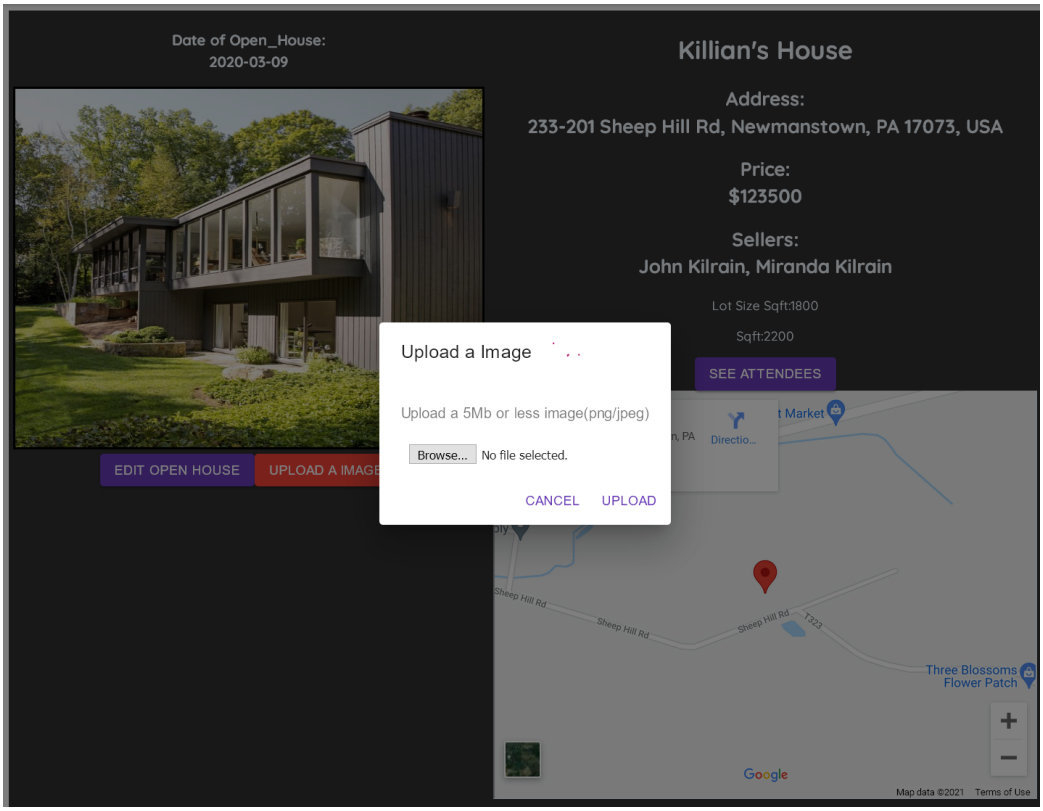

Figure 4.24: Image Upload Dialog: (Same for Open Houses and Listings)

Adding and Editing Open Houses

In order to add or edit an open house the user must go into the respective route of the action they wish to perform and then add the details of the open house along with the date of when it is going to happen. It is important to note that the date entry is in American format i.e. Month-Day-Year and when the open houses are rendered upon the house view page they are in Year-Month-Day Format, this is due to the way JavaScript and Firebase are handling the date objects.

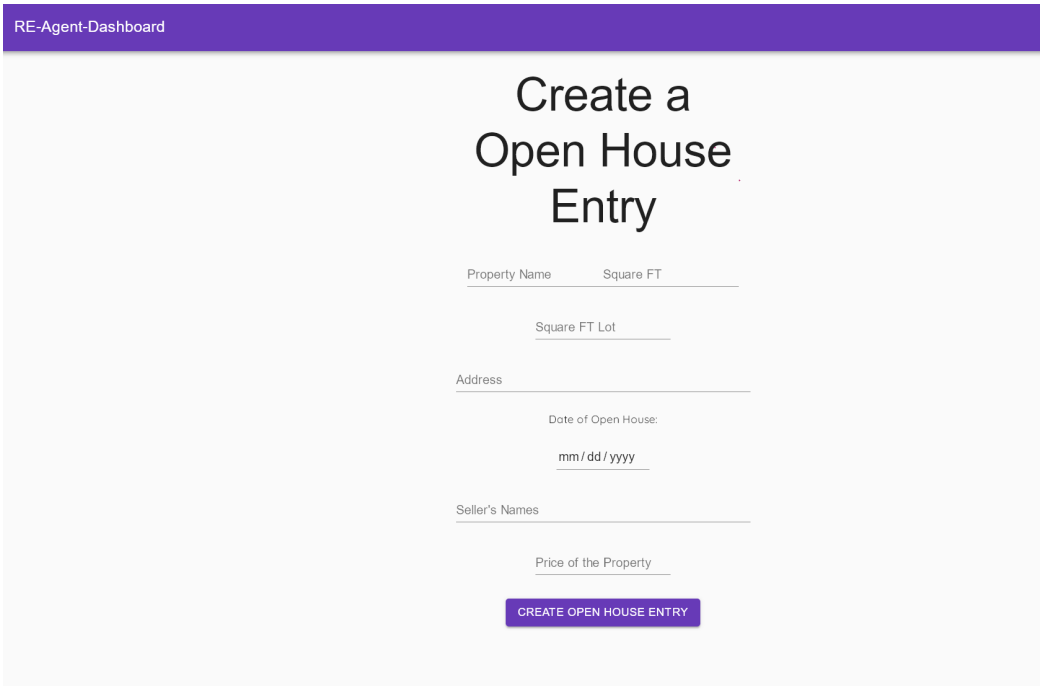

Figure 4.25: Open House: Create

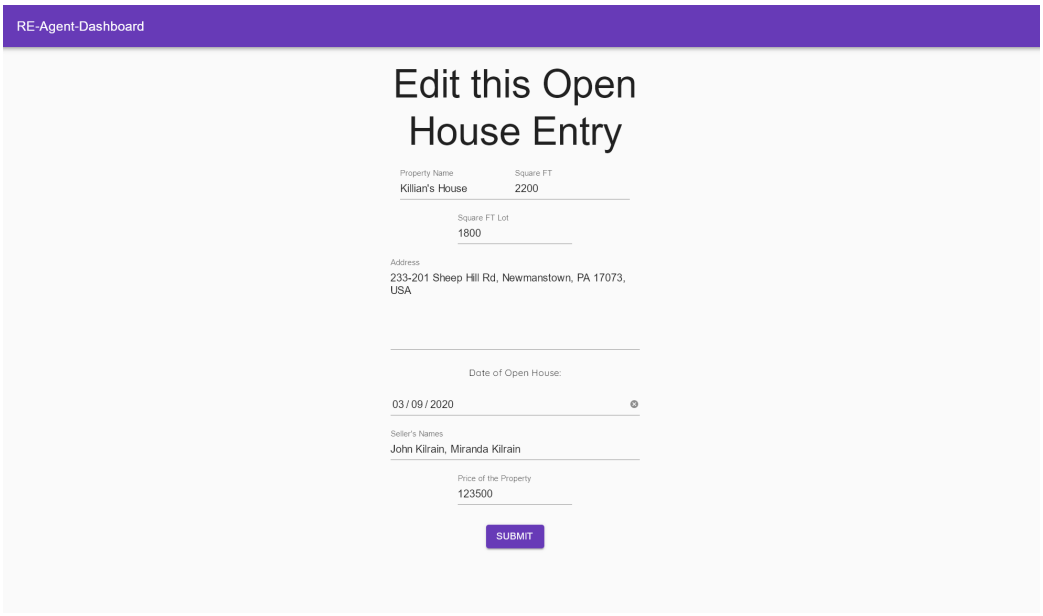

Figure 4.26: Open House: Edit

4.3.8 Attendees

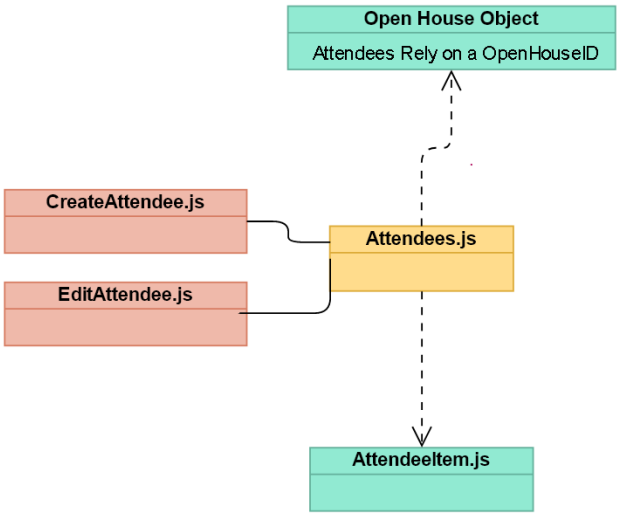

Figure 4.27: Attendees Class Diagram

This part of the application is bound to the open house entries as with each open house there may be unique and different attendees. To access the attendees portion of the application, you either need to be on the **Open House View** or **Open Houses Main** page as discussed previously and shown in previous figures. Once the user enters this section, they are presented with a button to create an attendee which will prompt them for the attendee's full name and contact details.

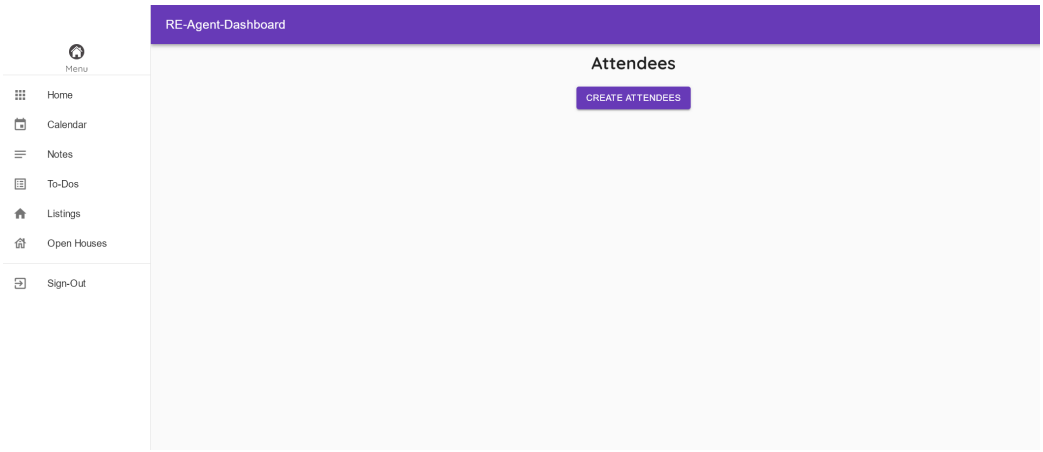

Figure 4.28: Empty Attendee Page

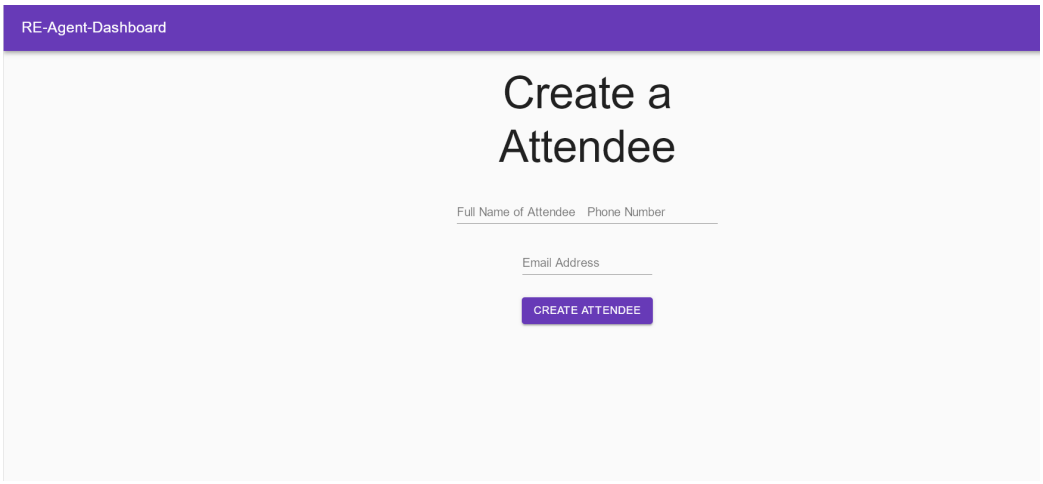

Figure 4.29: Create Attendee Page

Once attendees are created, they will show up on the Attendees page for the open house that the attendee had gone to. This is when optional buttons that an agent can use later on are given for that attendee. These options include **"Mark as Contacted"** and **"Mark as Interested"**. Once an attendee is marked as contacted that button will disappear, but the button for Mark as Interested has a duality of states where if the state is set as interested, the button will change to **"Mark as Uninterested"** which can be used if an attendee changes their mind on whether they are interested in purchasing the property.

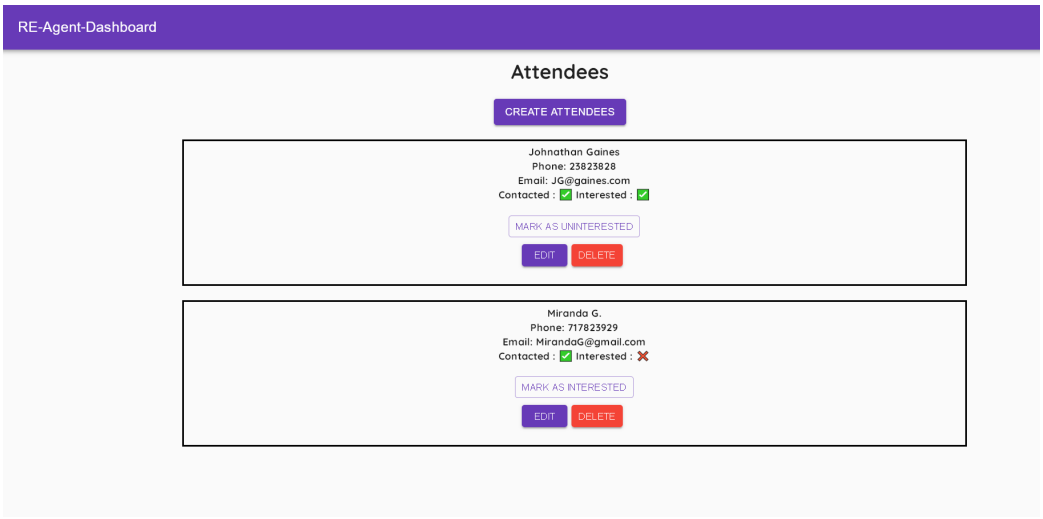

Figure 4.30: Attendee Page with Entries

Code for Marking as Interested/Contacted:

```
// If input is true, return a check mark else a X
    function CheckMark(input){
      if(input){
        return 'U+2705'
      }
      else return 'U+274C'
    }

    // Function to remove the contacted button if the attendee in
    ↪  question
    // has Been Contacted.
    function ContactedButton(input,button){
      if(input === false){
        return(
      <Button variant="outlined" color="primary" onClick={button}
      ↪  size="small" style={{margin:"5px"}}>
        Mark as Contacted
      </Button>
      )
      }else{
        return <p></p>;
      }
    }
```

```
22
23      // Function to change the state of the Interested Button
24      // Based on what the current value is.
25      function InterestedButton(input,InterestFunc,UninterestedFunc){
26        if(input === false){
27          return(
28         <Button variant="outlined" color="primary" onClick={InterestFunc}
          ↪  size="small" style={{margin:"5px"}}>
29          Mark as Interested
30        </Button>
31          )
32        }
33        else{
34          return(
35          <Button variant="outlined" color="primary"
          ↪  onClick={UninterestedFunc} size="small"
          ↪  style={{margin:"5px"}}>
36          Mark as Uninterested
37        </Button>
38          )
39        }
40      }
```

4.3.9 Listings

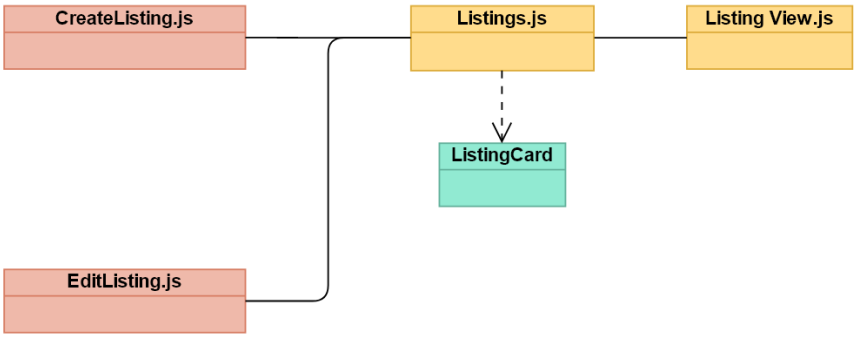

Figure 4.31: Listings Class Diagram

The Listings page system works similarly to the Open House System except that there are no attendees and the information on the cards shows the date of the listing's creation rather than an inputted date the real estate agent provided when creating them. As mentioned in the back end the listing model contains more fields relevant to the actual homes features and structure. Most of these details cannot be displayed on the card format alone, so the listings have their own listing view pages which can be accessed through the card's view icon.This view page contains all the details for the house along with its geographical location on the Google Maps Embedded API.

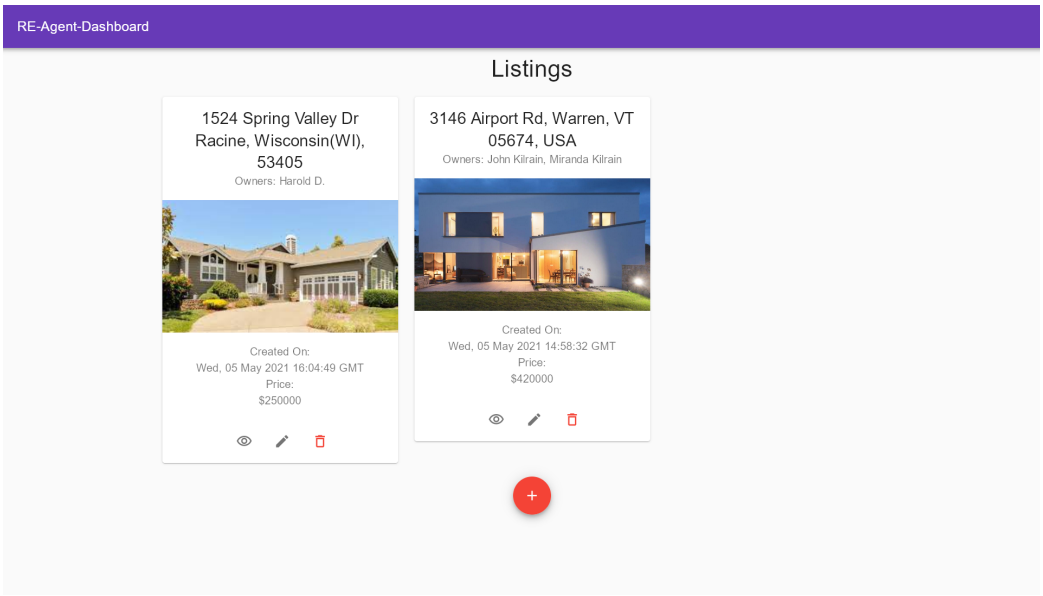

Figure 4.32: Listings: Main Page

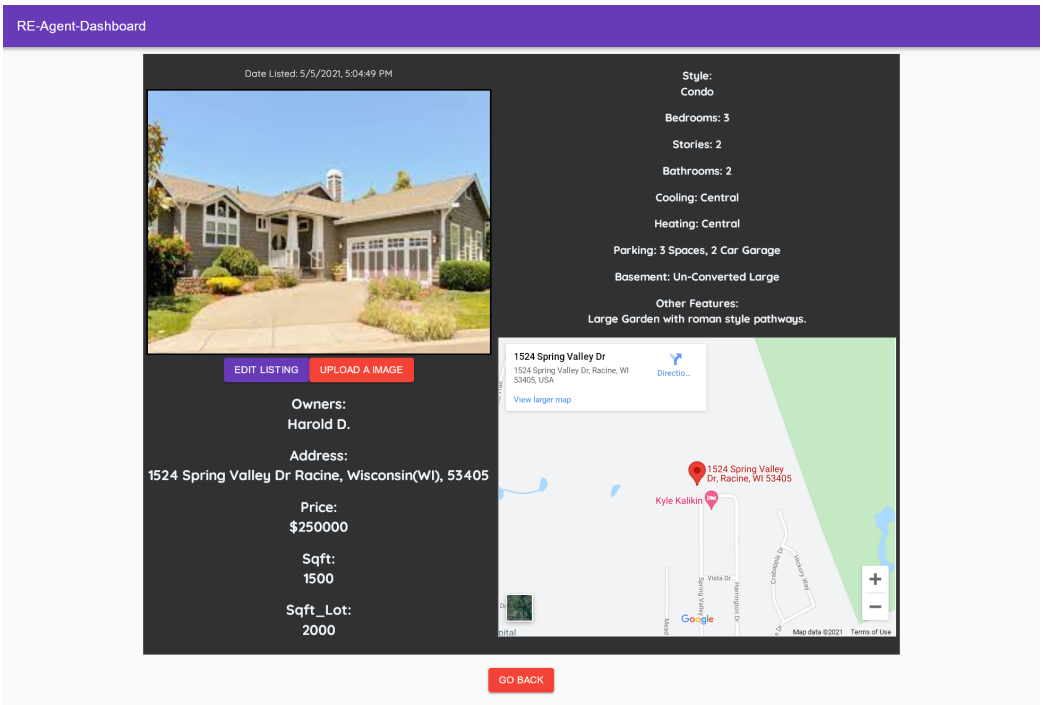

Figure 4.33: Listings: View Individual Listing Page

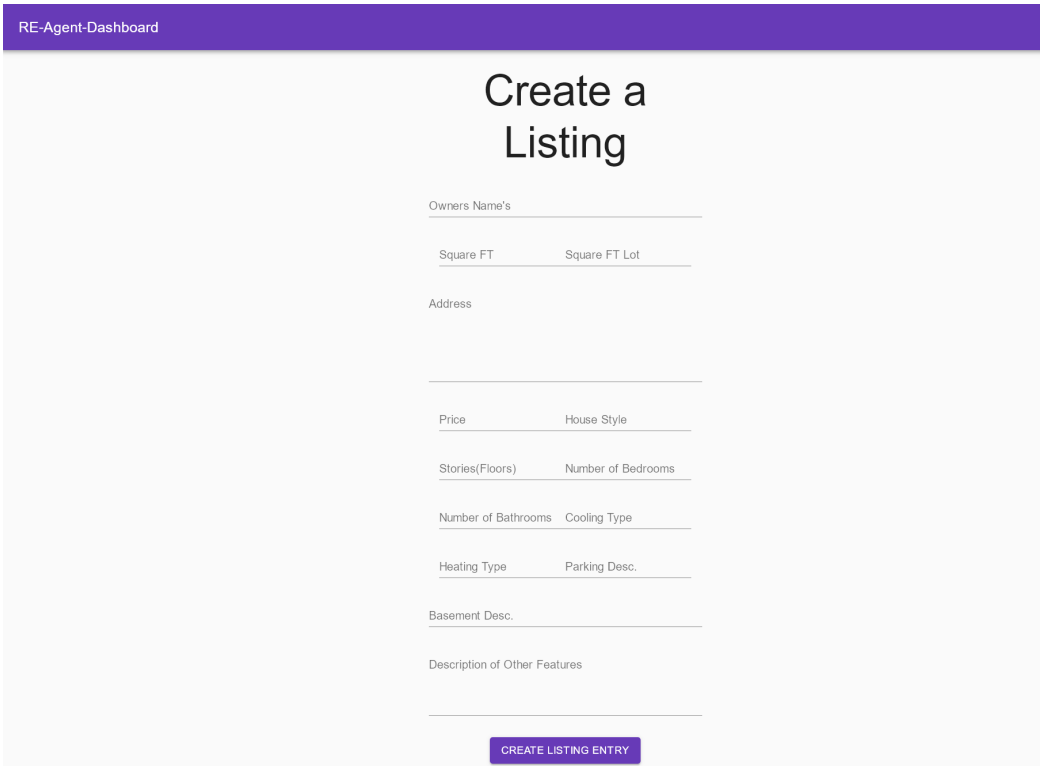

Figure 4.34: Listings: Create a Listing

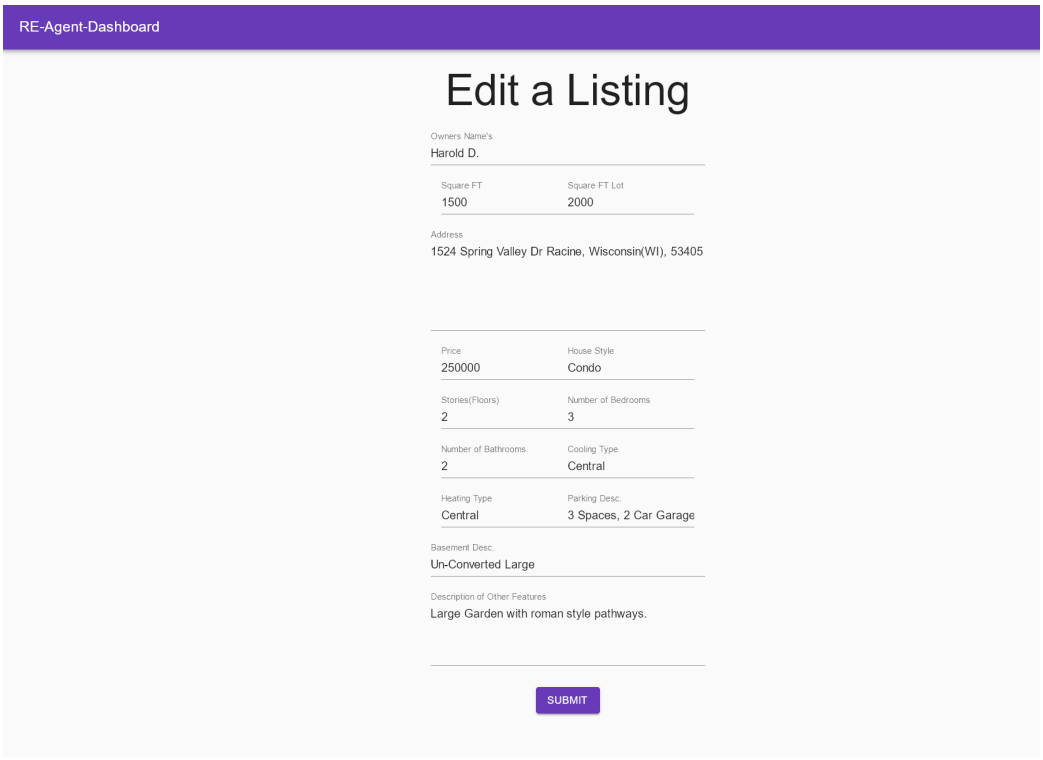

Figure 4.35: Listings: Edit a Listing

Chapter 5

System Evaluation

This chapter will give insight into the evaluation of the application's architecture in a critical sense and topics such robustness testing and an evaluation of the finished project's merits vs the original objectives that were set prior to its creation.

5.1 Testing for Robustness

5.1.1 Back-End Testing

Postman Automated Test scripts were used to test the back-end of the application and verify that the routes functioned as intended although through this method the deletion routes, image uploading route and update routes were not tested as my knowledge within Postman testing was limited, and these tests were done late into the application's life cycle. A figure of the tests performed within Postman is shown below:

Figure 5.1: Back-End Postman Testing

The above tests encompass testing of the User Login, Sign-up, Authenticated Route functionality and the GET and POST functionality of all the back-end data model classes.

5.1.2 Front-End Testing

Selenium IDE was used for the Front-End testing of the application as it worked on a user level basis (black-box testing) and allowed the application to be tested under realistic scenarios that could occur every day for users. Not all aspects of the application were tested through the Selenium IDE plugin as the application was used and black box tested extensively during its development and such only the major components were tested. A figure of the Selenium Test Suite is give as follows:

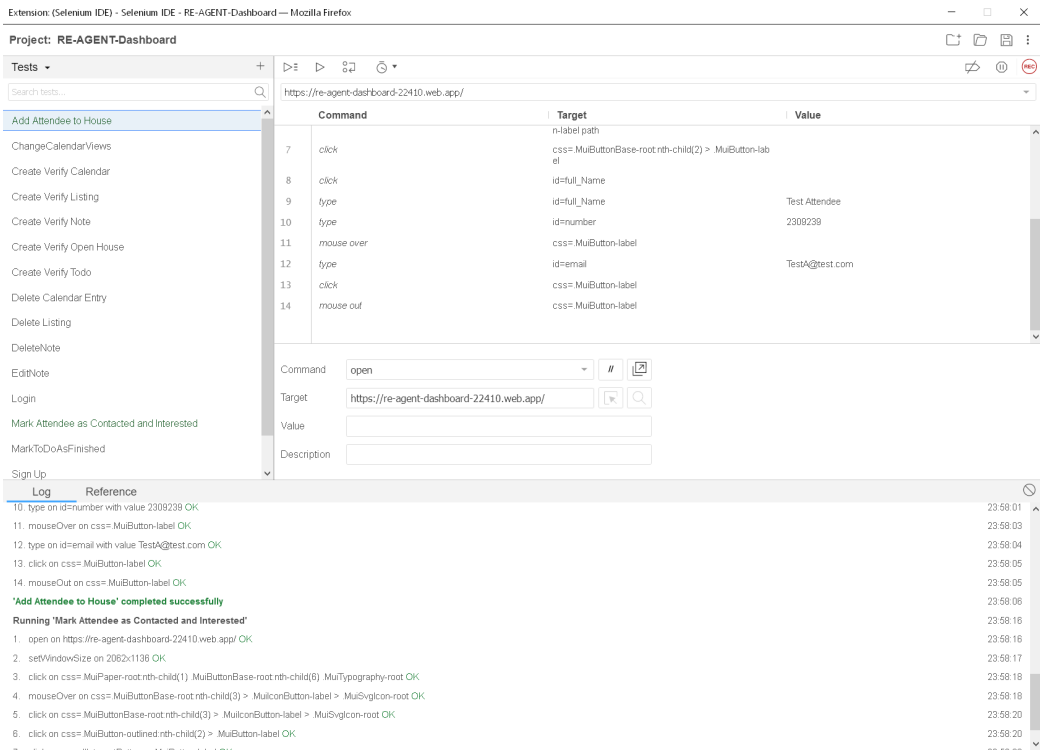

Figure 5.2: Front-End Selenium Testing

5.2 Limitations of the Project and Opportunities for Improvement

Although the project covered the objectives originally set out either fully or to a high degree, there is still issues within the application's features that could be built upon. Examples of such features and limitations will be given below.

Linking the Components Further

It would be ideal to be able to forward listings entries into an open house entry such that the agent does not need to make the same entry twice and thus making it more efficient for the agent and this same sort of intuitive entry forwarding could be applied between To-do entries and the calendar, as a To-Do and Calendar event can often times be the same task to complete.

Mobile Responsiveness

Most of the application is responsive on mobile devices or contains methods to make it responsive such as the different views on the calendar but the full calendar API does not have a responsive month based view built in, and such it would be more visually appeasing and easier to use for an agent if the calendar was fully responsive on any device in any view format they choose, thus this feature could either be tampered with or replaced to fix month view responsiveness issues.

Load Times

The application is fast in most sections based off mine and Carissa Jurina's usage experience and testing, but there is a slight delay in load times especially in the calendar retrieval due to Firebase's own server latency and this often brought up the question during development on whether this would improve over continuous usage like on other platforms such as Heroku where functions are often put to sleep until woken by an HTTP call or would the web application's load time issue get worse over time as more users sign-up to it. In the latter scenario it may be optimal to switch hosting platforms into the Amazon AWS as the application becomes mid-sized in user scale and thus replace the Firebase database format for the MongoDB format. Although this would transformation of database architecture would require the transferring of prior created information within Firebase and transforming it into a MongoDB friendly format manually.

5.3 Overall Evaluation of the Applied Project

Overall the systems works as intended by the original objectives established during the early stages of the projects' composition. It allows real estate agents to save personal notes to help them in their work field and is both easy to use and most of the website works upon mobile devices. It also provides good security through the JWT token authentication system and most input fields for the creation of new documents are validated. Even though the application is basic in theory, it was very difficult to create and offered a plethora of learning opportunities throughout it's conception which will be discussed in the *Conclusion Chapter*. The contact with a real estate agent who worked in the field of my target audience helped me find key points of improvement in the applications' user interface and functionality without it becoming over whelming in most areas. The use of Node.js allowed for a very consistent workflow when creating the API in which the front-end

gathers data from and after some time with the framework it had begun to become easier to create new functionality as I felt the application required. During the post deployment conversation with the Realtor named Carissa Jurina that inspired the idea for this project and helped give insights into it's creation, she had expressed that she personally felt there was a great deal of potential in the future of the application if it were to be developed further for a larger audience in her field and that even in its current state it functioned well as a helpful data management application for her work field but with this conversation she also gave ideas that could be implemented in the future of the application and such those features will be discussed in the *Potential Features in Future Development* section of this paper.

Chapter 6

Conclusion

Re-Agent Dashboard helps fill the gap within the software application market for real estate agents who wish to manage their data in a uniformed way on an online web service where all their data is kept private through security using JWT authentication tokens from the Google Firebase Platform. The application is intuitive for both older and younger users and thus works well for even non computer literate agents.

The application is the construct of many hours of deliberation and hard work and with that many learning outcomes were achieved in relevance to software development and industry standards within modern full stack technologies.

6.1 Learning Outcomes

6.1.1 Consultation with Potential Clients

Being able to talk to a real estate agent about the project created for their work field and about the different issues that could be solved with modern software technologies was a new experience to me as a developer as I had never previously created an application with a real tangible target audience in mind and so the peer reviewing process was both exciting and stressful but had offered me new insights in how to approach a potential client's wishes whilst staying within time restraints and scope. The conversational process became difficult at times as my technical experience as a potential software development graduate would sometimes cause the misuse of technical terminology towards the client during conversation and thus learning to speak to a potential user in a way that they could understand the projects ongoing direction in layman's terms was an invaluable skill that could be used in

future endeavours within the software development industry.

6.1.2 Working with a Full Stack Architecture

Creating the project's full stack architecture from scratch was a difficult process, and it was often times challenging to link the various components in a way that was functional and efficient. Learning how to integrate the various technologies used together was a test of my research capabilities and technical abilities as a developer. Although I am happy with the projects resulting architecture and performance, I had learned about so many aspects of why modern technologies like React, Firebase, JWT Authentication Tokens and CSS frameworks such as Material UI are used over the conventional way of web development a decade ago when simply JavaScript, HTML and CSS were used or other technologies such as PHP. These modern stacks are created to allow the developers to efficiently emulate the modern architectural patterns of today such as the Model View Controller (MVC) pattern[44]. Node.js asynchronous and modulated system made it very efficient to write re-use-able code that was easily updated and React allowed me to separate the concerns of my JavaScript classes in a way that I could manage each component individually without the worry of having bloated files with too many lines of code. I could directly get to the cause of any issues or bugs that occurred during development by simply going to the component that caused the problem and fix it on individual basis without the rest of the application breaking.

6.1.3 The Importance of Research and Testing

Many issues were encountered through the project's development life cycle which were either through normal usage during development or found later through the Postman or Selenium based tests. These issues may not of been found without the importance put upon testing later on into the project and such the application would have performed poorly during user usage after deployment. Many hours were spent trying to fix code issues and a large amount of research was undertaken, through websites like Stack Overflow and GitHub. Solutions and reasoning's to the problems I was having were made apparent through these websites by other developers. For example there was a lot of issues with the user authentication and keeping reference to the logged-in user so that the application could retrieve documents under their user ID, I had very little knowledge on how authentication tokens worked or how to keep them within the request headers, eventually after sifting through many different answers, I came to a solution that combined

the different aspects of the research and worked for my use case. Without such research the application would not of been secure or would not of functioned as intended by the original objectives. Another key issue that was found during development was that of the CORS cross-region error, when trying to upload images to the projects' Firebase storage bucket through the Back-End code, eventually this issue was solved by doing research through Google Cloud's documentation and learning that I needed to set up CORS headers on the storage bucket separately using the Command Line Interface on the Google Cloud Console for the project. Most of the answers I had found on stack overflow for this issue were outdated, as Firebase is constantly updated and so is the Node.js architecture, thus I had learned key skills in being able to sift through the platform documentation provided by the source rather than always relying on finding the solution to an issue through forum based websites like stack overflow from which I had got answers that were deprecated or no longer relevant in the year of the applied projects' creation.

6.2 Potential Features in Future Development

Although the project has the basic requirements needed to function as a real estate agent helper, there are many features that could be added to improve the ease of use and intuitive nature of the project. These features were both realized after much thought and conversations with the Realtor Carissa Jurina who had tested the application for me.

6.2.1 A Better Calendar Solution

The main page with issues in mobile responsiveness is that of the Calendar display, as the *month based view* is too condensed upon mobile sized screens. This issue is because of how the Full Calendar API is designed and is something that either needs to be fixed by tampering with the CSS and other features within the Full Calendar implementation within the application or replacing the package out right for a custom-made calendar based component. The reason for which a custom component was not made in the project during its development is due to time constraints with other projects within the College year. As the project could be developed further post the completion of the college year, the time needed to make a complex calendar component without the use of separate packages could be fulfilled. This custom component would ideally be responsive and allow the exportation of the calendar entries into a .csv (Excel) format that could be uploaded to other calendar based platforms such as Microsoft Outlook or Google Calendar.

6.2.2 Web Scrapping / API usage for Realtor Listings

As of currently the details of an agent's listings have to be entered in manually and thus this is time-consuming and not intuitive. In the future it would be ideal to use an API such as Zillow and search the agents name within the API and parse the results back into the listings within the Re-Agent Dashboard application. Another way of tackling this issue would be to develop another service using Python with beautiful soup or some other web scrapping library and scrapping data off of popular real estate websites to gather all the details for the logged in agents listings. This feature would also require the addition of a full name and email to the user database on the Firebase back-end and such would require large overhauls within the application to perform. This was an oversight of my own and the idea of this feature was relayed to me by the real estate agent I was in contact with late into the application's development life cycle, in such there was no time left to implement such a feature.

6.2.3 The Linking of Components Further

As previously stated within the *System Evaluation Section*, it would be ideal to further link the various components of the system's architecture to make the application more feature packed and intuitive for the user and remove the issues that currently plague the application in terms of information repetition, as users do not often want to input the same information twice into two different areas. The further linking of components would be essential to improve the experience for any real estate agents who use the application.

6.3 Final Thoughts

This project was created to try to solve a industry level problem for real estate agents wishing to manage their work related data, as currently within the industry there was no real estate based web application that could contain the various categories of information that a real estate agent may wish to keep track of online. The applications according to the few real estate agents I had contacted, only solved parts of this problem such as Listing tracking and Note tracking applications, but they often costed a large annual or monthly fee that was not within the budget of real estate agents who were not earning a high commission and such a cheaper alternative was needed. Over many hours of hard work and deliberation this project was created to make a solution for the issue on a decent scale that is cheap and

intuitive to use, efficient and secure.
Through creating this application many hurdles were faced on both the back-end development and front-end along with any research and project management undertaken thus my mindset during development was not always the most enthusiastic, but on a personal note I felt that it really helped me learn stress and time management, and gave me life lessons in how to manage a large project and what pitfalls to avoid such as procrastination or relying on the first few sources I may see for a software development based problem.
I am thankful for the time that everyone I had been in contact with had given to me to create this project, such as my supervisor Dr. Gerard Harrison, the real estate agent for Berkshire Hathaway Carissa Jurina and the various contacts she had provided for me from her workplace when I wanted more feedback on the application. I felt these experiences helped me become more efficient at dealing with potential customer's wants, and although I was not working on a team for this project, I felt that the consistent contact with these peers improved my team work skills as the application would not of been creating as successfully without the time and direction they had given to me. As such I am now more confident in my full stack development and research skills and would potentially like to pursue this project further and make it into a commercial application in the future.

Bibliography

[1] B. Hathaway, "Carissa jurina information."

[2] Y. W. Jerry Gao, H.-S. J. Tsao, *Testing and Quality Assurance for Component-based Software Chapter 6.* Artech House, 2003.

[3] Visual-Paradigm, "What-is-user-story."

[4] lucidchart.com, "Agile-vs-waterfall-vs-kanban."

[5] DevCom, "Agile advantages for software development and your business."

[6] Inflectra.com, "What is agile kanban methodology?."

[7] L. J. Dennis, "Lean tool of 'kanban' embraced at pixar, zara and spotify."

[8] Scrum.org, "What is scrum?."

[9] Scrum.org, "What is a scrum master?."

[10] R. V. Upgrad, "Github v.s gitlabs."

[11] J. W. Huger, "6 places to host your git repository."

[12] I. Times, "Microsoft moves to calm privacy concerns."

[13] GitLabs, "Github-vs-gitlabs."

[14] mongoDB, "Mern stack."

[15] MongoDB, "Mongodb - the database for modern applications."

[16] IBM, "Sql vs. nosql databases: What's the difference?."

[17] essentialSQL, "Sql acid database properties explained."

[18] IBM, "Cap theorem."

[19] ExpressJS, "Expressjs."

[20] ExpressJS, "Expressjs getting started."

[21] ReactJS.org, "React - getting started."

[22] NodeJS, "About node.js."

[23] Angular.io, "Angular - getting started."

[24] IBM, "Mean stack."

[25] Y. Xing, J. Huang, and Y. Lai, "Research and analysis of the front-end frameworks and libraries in e-business development," pp. 68–72, 02 2019.

[26] Bootstrap, "Build fast, responsive sites with bootstrap."

[27] S. AppWorks, "Comparing aws vs. firebase."

[28] Heroku, "About heroku."

[29] G. Firebase, "Firebase."

[30] Google, "Calendar api."

[31] FullCalendar.io, "Fullcalendar: React component."

[32] GitHub, "Frontend - github topics."

[33] Tsh.io, "State of frontend 2020 report."

[34] Google-MaterialIO, "Material.io."

[35] Material-UI, "Learn material-ui."

[36] F. MDN, "Html: Hypertext markup language."

[37] M. MDN, "Css: Cascading style sheets."

[38] w3techs.com, "Usage statistics of javascript as client-side programming language on websites."

[39] JSON.org, "Json."

[40] Microsoft, "Visual studio code."

[41] Postman, "Automated api testing with postman."

[42] S. IDE, "Open source record and playback test automation for the web."

[43] www.computerhope.com, "What is a hamburger menu."

[44] Visual-Paradigm, "What is model-view and control."

6.4 Appendices

GitHub Repository:
`https://github.com/JamesP1996/RE-Agent-Dashboard`
Demo of Applied Project:
`https://youtu.be/OyztJUNIovU`
Web Application Deployment Link:
`https://re-agent-dashboard-22410.web.app/`

Below are the details of the Real Estate Agent I was in Contact with for the Applied Project, who is open to any queries into our relation throughout its development:
Carissa Jurina (Real Estate Agent) Email: `cjurina@homesale.com`
Berkshire Hathaway Home Services Page: `https://www.bhhs.com/homesale-realty-pa305/lebanon/carissa-jurina/cid-1123670`