Renetic - An API-Enabled Customer Relationship Management Application for Real Estate

University of Greenwich, Old Royal Naval College
United Kingdom

Luke Austin,
School of Computing and Mathematical Sciences,
Computer Science (Networking)
Email:la9619j@gre.ac.uk,
Supervisors:
Sadiq Sani, Jason Parke
Word Count: 11543

April 24, 2023

Abstract

The purpose of this final year report is to design and implement a landlord and tenant application that can provide a seamless and efficient way for landlords and tenants to manage their rental agreements and properties. The application aims to solve common challenges faced by landlords and tenants such as rent payment management, finding important documents and maintenance requests. The application is built using Apple's programming language 'Swift' and follows a user-centred design approach to ensure a smooth user experience. The application will also include an outside-the-app feature where landlords can set a code for an electronic door lock system for their property, this system will be built using a Raspberry Pi. The app is being developed using modern mobile technologies such as cloud-based databases to ensure scalability and reliability. This report outlines the development process, features, and testing of the app, explores the challenges and limitations faced during the development and provides recommendations for future improvements.

Preface

As the world becomes more digitised, it's no surprise that companies that were traditionally "offline" are starting to embrace the power of technology in the 21st century. One particular industry that is using is using this technology is real estate and property management. With more and more people renting their homes, software applications need to be designed for landlords and tenants that can offer a range of features and tools to help simplify the rental process.

This report focuses on the development of a landlord and tenant application "Renetic", a software application designed to streamline the rental process and improve communication between landlord and tenant. The app provides a suite of tools for landlords, such as collecting rents from tenants, maintenance requests, setting a door security code for their property, messaging their clients and accessing important documents, whilst also offering tenants access to features such as online rent payment, maintenance request and messaging.

As this project is around computer science, the development of this app required the integration of a variety of technologies and programming languages such as the use of Raspberry Pi, Swift and Python programming languages. The report will also include technical details of the development process, including the use of APIs, databases, and cloud services using Google 'Firebase'.

The report also discusses the design of the user interface of the app, with a focus on creating a clean and intuitive user experience for both landlords and tenants. The design of the app has to be simple and easy to navigate, with features that are also easy to access and understandable.

Acknowledgements

I would like to acknowledge and give my thanks to my supervisors Sadiq Sani and Jason Parke for their guidance and feedback on my project and also for helping me write my report.

I would also like to give special thanks to my mother, for giving me knowledge on property management the features required for tenants to have in my application and for giving me feedback on my application. lastly, I would like to thank my father who helped me solder my components together for the door-locking system.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In today's digital age, the real estate industry is in need of more efficient and effective ways to manage customer relationships. The goal of this final-year project is to design and develop an API-enabled customer relationship management (CRM) platform for the real estate industry. The platform will allow real estate agents or landlords to easily manage and track their interactions with clients, access important documents, make a payment system where clients can pay through the platform and include emergency repair notifications in case a client notifies a problem with the property and lastly, an electronic door lock security system where landlords can give tenants a code to enter into their property. By utilising API integration, the platform will also be able to connect with other tools and platforms commonly used in the industry, such as Firebase's database to get information, Google Maps for the property location and even using WeatherAPI to tell the user what the local temperature is in their area. Overall, this project aims to improve the productivity and effectiveness of real estate agents while also providing an improved customer experience for their clients. The primary motivation for this project is that there are currently few applications that provide all the features listed or utilise the full potential of API with the combination of IoT devices in the real estate market. From reading the constant problems clients have with their property in a Reddit post to hearing on the radio, this platform will be beneficial in the real estate industry. Renetic has the potential to be commercialised and to be used by housing associations and private landlords not only in the United Kingdom but internationally.

API stands for Application Programming Interface, when this is applied

to an app, the API then refers to a set of protocols and tools that allow different software applications to communicate with each other. API provides a way for the app to access data or services from another application or server. For example, Apple's IOS weather app uses API provided by 'The Weather Channel's service to retrieve current weather data and display it within the app. The use of an API-enabled CRM platform can provide several benefits for real estate agents and their clients. First, API integration allows the platform to connect with other tools and platforms commonly used in the real estate industry, such as property listing websites and social media. This can greatly improve the efficiency of data management and communication between agents and clients. Additionally, an API-enabled CRM platform can provide real-time data access and updates, allowing agents to easily track and manage interactions with clients. This can greatly improve the effectiveness of customer relationship management and lead to increased sales productivity. However, there can be some disadvantages to using API in a CRM platform, implementing an API CRM platform into an application is a potential data security risk. Hackers can exploit the platform without exploiting the platform itself by bypassing the API server.(Idris et al. 2021)

IoT or Internet of things is a revolutionary technology that has the potential to change the world that we live in. The way it works is by having a connection between a device and the internet, this allows them to communicate with each other and exchange data.(Jesu´s Carretero and J. Daniel Garcı´a, 2013) With security door looks they typically use wireless connectivity, such as WI-FI or Bluetooth, to connect to the internet and allow users to lock and unlock their doors anywhere in the world via an app or website. However, this device will be connecting to the internet via an Ethernet cable with the user unlocking the door with a keypad. IoT door locks offer several advantages over traditional door locks, such as an increase in convenience and security. For example, users of have forgotten their keys or have lost them can get in by via a code and can grant access to contractors or close family members. But, with many IoT devices there as potential risks and vulnerabilities. A physically present attacker can observe what the user is entering into the smart lock, thieves can steal the user's device, relay attack where the attacker is near the door lock, the attacker will retrieve the victim signal and use that to unlock it or electromagnetic damage which can overheat components and permanently damage them. (Ho et al. 2016)(Tomic 2017)

Chapter 2

Literature Review

Customer Relationship Management (CRM) is a vital aspect of any business, particularly in the real estate industry where the management of customer interactions and relationships is crucial to success. In recent years, there has been a significant shift towards digital solutions for CRM, with the use of mobile apps and API integration becoming increasingly popular. This literature review aims to explore the relationship between landlords and clients, the potential benefits of an API-enabled CRM platform for real estate agents and their clients and lastly the benefits of an IoT device such as a smart door lock in the real estate industry.

T**he relationship between landlord and tenant in the real estate industry:** The relationship between landlords and tenants in the real estate industry has been the subject of much research and discussion. The connection between these two started back in the feudal era when landlords owned a large amount of land and rented it to tenants in exchange for rent or labour. The relationship was based on a hierarchical power structure where landlords had complete control over their tenants. (Byres 2009) Since then, tenants have had more rights and protections with the introduction of tenancy laws being included in today's society. However, there are still problems with the relationship in the 21st century. Studies have also shown that the relationship can be complex and multifaceted, with factors such as communication, trust, and power dynamics still playing important roles. According to (Rasila 2010)tenant trust was an important factor affecting landlord and tenant relationship quality. Building and maintaining a positive and respectful relationship can

lead to successful tenancy and long-term retention. Landlords who prioritise creating a positive relationship with their tenants will often experience fewer issues and conflicts, which leads to more efficient and profitable rental property. (Vaughan 1968) Effective communication is widely recognised as one of the most important factors when it comes to building a positive landlord-tenant relationship. A survey which was found by (Alana Stramowski, 2022) found that 48% used text, 28% used email, 13% used phone calls and 10% in personal conversations as a way for landlords to communicate with their tenants.

Landlord and Tenant Responsibilities: Tenant and landlord duties are other important factors in landlord-renter interactions. According to the Gov website, both parties have specific responsibilities under the lease agreement which they have both agreed on (signing of documents). If both tenants and landlords stick to their responsibilities it can lead to a positive renting experience for both tenants and landlords. The landlord's priorities from reading the paper by (Wheeler 2019) were that they were in charge of keeping the property up to date, making repairs as needed, and making sure it was secure and livable. They are also responsible for making a timely rent payment, abiding by the conditions of the contract, and maintaining the tenant's privacy. On the other hand, the tenant is responsible for making timely rent payments, keeping the property in good repair, reporting any concerns that require maintenance or repairs, and abiding by the terms and conditions of the lease agreement. If they do not fulfil this, the landlord has the right to evict them from the property. Both landlords and tenants can benefit by carrying out their respective obligations.

The role of technology in improving landlord-tenant relationships: The relationship between landlords and tenants is a crucial aspect of the real estate sector. Greater pleasure for both parties can result from effective communication and a good relationship, but unhappiness and legal problems might result from ineffective communication and conflict. The landlord-tenant relationship may be improved by technology in several ways, and real estate platform apps have become an important tool for enabling communication and enhancing the entire experience for both landlords and tenants.

By easing communication, technology can help landlord-tenant interactions. Apps for real estate platforms frequently have messaging and chat tools that make it simple and quick for landlords and tenants to communicate. This can assist in minimising misunderstandings and ensuring

that everyone is on the same page. Furthermore, real estate platform apps can give users a central spot to store crucial data and documents like leases and maintenance requests. This can make it less likely that essential documents will be lost or misplaced and make it simpler for both landlords and tenants to obtain that information.

APIs in mobile development: As more and more applications are being built for the user's needs in the App market, nearly all of them have API integrated into them. As explained in the introduction, API or Application Programming Interface allows two applications to talk to each other, which enables developers to build complex, powerful and flexible mobile applications. In this section, we will be exploring the history of APIs, how they are used in mobile app development and their benefit and drawbacks.

History of APIs: The history of APIs can be traced back to the 1940s when British Maurice Wilkes and David Wheeler both worked on modular software libraries for the Electronic Delay Storage Automatic Calculator or EDSAC is known for short. The way Maurice and David designed it to work is by the subroutines in the library were stored on a punched paper tape which was then organised in a filing cabinet. The cabinet would contain a library catalogue which was basically notes containing subroutines and how to integrate them into a program, this process could be considered the first API documentation.(Wilkes et al. 1958) From the 1960s to the 70s, operating systems began to use system calls in their software, system calls are predefined functions that could be called by a program to perform common tasks, an example of this could be opening a file in Windows to just simply asking a printer to print a document. This feature allows programs to interact with the operating system without needing knowledge of the hardware of the system. In the 1980s, as computers started to acquire network communication, there was an area that needed to be sorted to allow computers from one part of the world to communicate with another computer. This was answered by the introduction of RPC, Remote Procedure Call or RPC for short is a communication call that enables a computer program to execute a procedure in a different address space without the programmer coding the details of the remote interaction.(Birrell & Nelson 1984) This performs making a request from the client program to a server for a specific task to be performed. (Client-Server Interaction) Our application would be using this technique because the user in our application

would be requesting data from our Firebase API. In the 1990s, when the World Wide Web was being introduced to the world and many platforms were being made for delivering content and services over the internet. The development of Simple Object Access Protocol (SOAP) and Representational State Transfer (REST) was created. SOAP is a messaging protocol that is used in web services to exchange structured information between applications over the internet while REST is an architectural style for building web services using the standard HTTP methods to access and manipulate resources. This is still used in today's society for web services such as Google Maps and Amazon web services, so integration of this would be beneficial for our application because it would keep up with up-to-date trends. As we move to today's society most applications would have API in their system, Social media apps such as Facebook and YouTube have their own API, according to a survey by (Belkhir et al. 2019) it told us that there were 1,595 mobile apps that used the REST API out of the 9,173 apps that were downloaded from the Google App store, so developers will then use their data for their own applications.

Benefits and drawbacks of using API for real estate applications: As some real estate applications have started to use API, they can provide numerous benefits for the developer, such as real-time data access, integration with third-party services and improve user experience. According to (Castro & Steinberg 2017) MLS or simply known as Multiple Listing Services has its own property listing so that local real estate brokers and agents would add data into the MLS database so that other brokers and agents can use this data in the future. When real estate uses API, they can provide other features such as up-to-date information on property listings and market trends which can greatly improve the functionality of the app and sustains user interest. However they can be some drawbacks when using API, one potential drawback is that an application might be too reliant on a service, if that service went down that application would not be fully functional for the user. Lastly using API can be complex and time-consuming for developers to work on.

IoT devices and door security: A new area that has received a lot of interest recently is the Internet of Things (IoT) door security. This utilises linked devices to the internet to improve the security and safety of residences, workplaces, and other buildings. One of the primary concerns of homeowners or business owners is the security of their prop-

erty. An IoT door security system is designed to address these concerns by providing a range of features and techniques to enhance the security of the building without the use of standard keys. (Sowjanya & Nagaraju 2016) These features include remote access control, where homeowners or business owners can control who enters and exits the building via their smartphone or tablet (Anwar & Kishore 2016), real-time monitoring provides a live feed of the door activity for the user, for example, the current market leader for security cameras "Ring" provides features such as high definition wide-angle view, motion detection, mobile access and night vision. (Ennis et al. 2016). An interview study by (Brush et al. 2013) mentioned that participants were interested in having cameras that recorded videos from outside their home in a rough neighbourhood, while (La Vigne et al. 2011) indicates that the introduction of cameras reduced the crime rate by 30 incidents per month on average, with this evidence its safe to say that cameras are a good crime deterrent for tenants and landlords. Security and privacy consideration also plays a critical role when it comes to IoT door lock or devices in general. The devices that we connect to our houses collect data and store sensitive information, which includes access logs and user credentials, which makes them a primary target for hackers and cybercriminals to expose. To ensure the security of IoT devices, we must implement robust encryption techniques, strong authentication protocols and regular software updates to tackle any vulnerabilities, for example, a technique suggested by (Lastdrager et al. 2020) included a reverse firewall, this technique works by placing a device outside and its job is to sanitise the messages sent to it so that a malicious device does not leak data to the outside world.(Chakraborty et al. 2020) The performance of IoT door locks can be evaluated based on various areas such as reliability, response time, power consumption, and user experience. Reliability is an important factor because door locks must operate consistently and faultlessly to maintain a building's security. Response time is also crucial since the tenants may become less motivated to use the technology if there are delays or slow operations. Another factor to take into account is power consumption because IoT door locks are frequently battery-operated and need to remain effective over a lengthy period of time. Finally, the user experience is an essential component of IoT door lock performance as it influences whether tenants will embrace and keep using the technology. These elements include ease of use, an intuitive design, and general convenience. We can develop a more thorough understanding of how IoT

door locks function and how they might be enhanced to better fulfil user expectations by taking into account these numerous elements.(Park et al. 2009)

In conclusion, integrating a CRM platform app with the use of API and IoT smart door locks has the potential to completely transform how real estate organisations manage their client connections and access control systems. Real estate businesses may connect their CRM platform app with a variety of other services and data sources by utilising the power of APIs, which enables them to obtain a deeper understanding of the wants and needs of their tenants. The integration of IoT smart door locks expands the capability of the CRM platform app by enabling organisations to more effectively and securely manage access to their facilities. These smart door locks provide a level of flexibility and convenience that conventional lock systems purely can't compete with due to their real-time monitoring and remote access features. Overall, the potential for real estate companies to manage their operations and customer contacts has risen significantly with the combination of a CRM platform app with API integration and IoT smart door locks. Real estate businesses could enhance efficiency, boost tenant satisfaction, and improve the security of their facilities for clients by embracing these technologies and utilising them to the fullest extent possible. As a result, it is obvious that any real estate company seeking to remain competitive and responsive to changing markets would benefit from integrating a CRM platform app with API connection and IoT smart door locks.

Chapter 3

Analysis

As more and more applications with the integration of it being cloud-based or use of API technology are being built for the app market. Analysing security, privacy and performance is an essential part of the development of a successful application. Our application ”Renetic” will use Firebase to enhance the security, privacy and performance of our application. In this analysis, we will be exploring various techniques that are used when developing mobile applications that can improve the security, privacy and performance of our application.

3.0.1 Security

Google's 'Firebase' is a mobile and web application development platform that provides developers with a wide range of features and services, such as authentication, database management, storing data and hosting sites. Firebase offers us several security features when protecting our data. As mentioned before, Firebase's 'Authentication' allows users several authentication options, which include email and password which we will be using, phone numbers, Google, Facebook, and Twitter. The option that we choose ”Email and Password” ensures that only authorised users will access our application through that method. However, SQL injections pose a serious threat to our mobile application because it allows attackers to obtain unrestricted access to the databases(Halfond et al. 2006). According to an article by (Mirante & Cappos 2013) a SQL injection attack on a popular business social media platform ”LinkedIn” resulted in 6.5m million passwords being stolen which didn't allow users to access their accounts. An application that uses Firebase is unlikely to be attacked with a SQL injection attack because it utilises a NOSQL

database technique instead, it uses a data structure like JSON to store and retrieve data within the application. We could also implement multi-factor or biometric authentication into our application for additional security. From reading an article by (Sarkar & Singh 2020) one benefit of using this system is that the user does not have to remember their username, password or pin which also helps our targeted audience of landlords (People between the ages of 55-64, commonly having memory issues)(Small 2002). (Sarkar & Singh 2020) noted that the use of biometrics came with drawbacks, one where the biometric information of our users is stored in a database which may be compromised and stolen for other uses and it compromises the user's privacy. Firebase also provides encryption features for mobile apps to help protect sensitive data according to (Tanna & Singh 2018). The way this system works is by encrypting the data both at rest and when it's in transit using the Advanced Encryption Standard encryption algorithm. This feature ensures the data is secured even if it is stolen by hackers. Additionally, Firebase also offers us a secure key management system that enables us to manage encryption keys and access controls. We can implement this feature for the Raspberry Pi(Smart door lock) to retrieve data from our database. Moving on from encryption, conducting vulnerability scanning and penetration testing are critical an aspect when it comes to developing an application. The activities can help us identify potential security weaknesses and vulnerabilities within our application. While researching security vulnerabilities within Apple products using Swift, we found out that there was an issue that involves "Swift for Ubuntu" which allowed attackers to arbitrary code during library loading. (CVE Details Datasource)(Bezemskij 2022) Since we will be using Firebases API packages that do library loading, we will have to consider this when developing the Renetic application.

3.0.2 Performance

When developing applications for Customer Relationship Management like Renetic, using Swift with Firebase can provide several benefits towards performance. Swift is a perfect programming language for developing an app that requires a fast and responsive graphical user interface. The languages were intended to be fully optimised for the iOS ecosystem and have a low-level syntax that can improve our app's performance. With Swift and Xcode, we can create a highly efficient and reliable app

for our intended users that can handle complex workflows, and large data sets, and create a user-friendly app that is easy to navigate within our project. When Apple was developing the IDE (Xcode) it implemented a robust development environment that helps us build, test and deploy the Renetic App (Kelly & Nozzi 2013). An example of the tools used for testing the application within the IDE would be debugging, profiling and performance testing. As mentioned before, performing tests using these sets of tools will help us create an efficient and reliable application that can work with complex workflows and large data sets which we will be using for Renetic. Moving away from Xcode and Swift, we move to the performance of Firebase. This API offers an adaptable backend infrastructure that can handle the demands of a CRM app. Major companies like Alibaba and eToro have millions of customers every month and they benefit from features such as real-time synchronisation, offline support and serverless architecture that Firebase provides. Implementing Firebase can help improve the app's performance and reliability which enables landlords and tenants to access and manage their information more quicker and easier. Since our application handles huge amounts of data such as tenant information, important documents, payment schedules and maintenance request, Firebase can automatically handle increased traffic and demand without us interfering with it. Additionally, when mentioning the offline support that Firebase provides, this means that landlords and tenants can continue using the app even if there is no internet connection. For example, tenants can still submit maintenance requests or view important documents, when they're in areas with very poor or no internet connectivity.

3.0.3 Privacy

Privacy is a critical concern when we're developing a CRM app because we are handling sensitive data such as personal information, lease agreements, and payment details. Unauthorised access to this data can result in identity theft, financial fraud and other privacy violations (Jain et al. 2016). Another concern would be the transparency of collecting data and its use. Users should be informed by us about the types of data collected, how it is used within our application and who it is shared with. Between 2010 to 2016, fifty million Facebook profiles were harvested by Cambridge Analytica, which was then used for the 2016 US presidential campaigns. Because Facebook failed to protect their data, they were

fined $ five billion by the Federal Trade Commission in 2019. (Cadwalladr & Graham-Harrison 2018) We must allow users to have the ability to control their data and opt out of data sharing if they wish to do so. Finally, users who use our application (Landlords and tenants) may have concerns about the storage of their data. Since we will be using the cloud to store their data, it can be vulnerable to attacks such as malware phishing attacks according to (Kaur & Kaur 2021). We can ensure users that their data is safe by implementing strong security measures to protect cloud-stored data.

In conclusion, developing and deploying a CRM platform that is centred around real estate using tools such as Xcode, Swift and Firebase requires a high focus on the security, performance and privacy aspects. Implementing security tools such as user authentication, encryption and executing regular testing and review are necessary to protect sensitive data such as personal information and payment details. Performance can be improved by implementing caching and optimizing network requests within Firebase and our application while utilising Firebase's real-time database and cloud storage capabilities. While privacy concerns should be addressed through privacy policies, data sharing, and secure storage. Overall, introducing these measures into our application can create a reliable and trustworthy app while improving the relationship between landlords and tenants.

Chapter 4

Requirements Specification

4.0.1 Purpose

The purpose of building this CRM platform (Renetic) for landlords and tenants is to provide a solution for managing properties efficiently within our application. Renetic will enable landlords to organise and manage multiple properties, send and handle maintenance requests, set their door codes and track rental income all in one place. While tenants will be able to report a maintenance request, view lease agreements, see the current status of their home and door code set by landlords and lastly be able to pay rent securely. It will also allow communication between landlords and tenants. Renetic will help reduce the time and effort required to manage properties and bring in a new age group to the real estate market, this will result in improved productivity and efficiency for our users.

4.0.2 Intended Users

The intended users of this CRM platform would reach out to landlords and tenants with the potential of people outside our research using it such as property managers, housing associations, full buyers and property agents.

4.0.3 Time-Length of building the application

The time length of building this application (Renetic) for landlords and tenants depends on several factors, Learning and adapting to the programming languages Swift and Python, meeting the requirements needed

for the intended users, the complexity of the project and the development phases which also includes design, development, testing and deployment. According to (Joorabchi et al. 2013) developing applications is not only challenging but time-consuming and costly, so it is important we follow the plan for developing the application. We will be aiming to complete this project within four to five months.

Table 4.1: Time length for building Renetic

Description	**Time length**
Research	1-2 Weeks
Idea Validation	5 days
Product Strategy	2 weeks
Prototyping	2 weeks
Prototyping Validation	1 Week
Design	3 Weeks
Development	2 Months
Deployment	1 Week

4.0.4 Functional Requirements

A set of functional requirements for creating Customer Relationship Management for our landlords and tenants are:

- User registration and login: The application should allow users to register an account with their personal details. It should also allow users to log in to the app using their email and password. The application should fetch or receive from Firebase during this process.
- Rent Payment: Tenants using the application should be apple to pay their rent online. They should be able to see how much they own their tenants and can pay a certain amount if they choose to do so.
- Communication via messages: The app should be able to allow communication between landlords and tenants directly through the app. It should also keep a log of all messages sent and received.
- Maintenance Requests: Tenants and landlords should be able to submit a maintenance request through the app, if the tenant sees something wrong with their homes, they can send a request to the Firebase database which is sent out to the landlord, while the landlord can put in a request for maintenance for the tenant to see.

- Property Management: Landlords should be able to see the information about their properties within the Renetic app with the also additional feature of adding, editing or even removing a property if they wish to do so. While tenants are able to view the property area and show the current status of the household. (WIFI, Heating, Gas and electricity data)

4.0.5 System Requirements

Since we are using Xcode and Swift to develop the application, they will need to have an iPhone Operating System (iOS) to run the required system. Listed below will be the devices that Renetic will be designed for.

- iPhone 11
- iPhone 11 Pro
- iPhone XS
- iPhone XS Max
- iPhone 12
- iPhone 13
- iPhone 14

4.0.6 Security

- Secure user authentication which we can implement using Firebase's authentication
- Firebase security rules that allow us to control access to our data and functions in real-time
- Monitoring security threats, we can do this by using the Firebase Analytics dashboard
- Keeping our application and software used for the creation up to date ensures that we are protected against any vulnerabilities.

Chapter 5

Design

Design is an essential part of creating any product, including applications as well. In fact, the design part is often the most crucial part of building an App. In this chapter we will examine the reasons why design is crucial when creating the app, it will show a user experience flowchart for Renetic, a low-fidelity prototype of the application using Axure software, a diagram of how the security door lock will work and discuss what SDLC (Software Development Life Cycle) is best when creating the app. For us to do the design of the application, research must be done first to evaluate the user needs, identify competitors in the industry and lastly stay up to date with trends and technologies.

5.0.1 Research

Understanding User Needs:
One of the primary reasons why research is essential when creating an application is that it helps to understand the user's needs. Performing research can help identify the user pain points, preferences and expectations, which informs the design and development process.(Calder et al. 1981) Since Renetic is a real estate platform, the target audience would be landlords and tenants so the design needs to be focused on their needs. According to the "English Private Landlord Survey 2021" from the UK government, Most landlords in this country are between the ages of 55 to 64 with the tenants being the same ages. So the design of the application must be easy to use for them. To do this, the Renetic application will use minimal design to prevent cognitive issues in older people, it will be easy to navigate through the app, provide the relevant features needed for the app and will include clear instructions on how to use the

application.

Identifying Competitors:
Another reason why research is crucial when creating an application is to identify potential competitors in the real estate market. Understanding the competition is critical for creating an application that can stand out in a crowded market. Research can also help identify if competitors in the market are doing well, what their weaknesses are, and how they are targeting users.
While searching for other applications that have the same features as what we are trying to implement into our application, we came across 'Happy Tenant', this application allows users to have 'Real-time Analytics', 'Financial Reporting' 'Online Payments' 'E-Approvals' and 'Document storage'. DialMyCalls is an application that landlords use to communicate with tenants by calls or texts. lastly 'Landlord Vision' provides the same features as 'Happy Tenant' but is more in-depth and seems to be a web-based application only. While looking at the reviews of these applications for example for Landlord Vision, users have provided that it is great for bookkeeping and accounting reports, however, users have said that the software is 'Complicated' to use and are not happy with the 136% increase of their services. Renetic will have the combinations of these app features combined with an additional feature of door security whilst being free for landlords and tenants to use so that it allows the app to stand out amongst other applications.

Staying Up-to-Date with Trends and Technologies:
Researching today's trends and technologies helps developers stay up to date with new techniques used within an application. The app market is constantly evolving and new technologies and trends emerge all the time. So adding new technology will keep Renetic relevant in the real estate market.

5.0.2 Landlord Side

This flowchart will show the direction of what landlords will go through in the application. The user will first be greeted with a get started page which would have two buttons on which the user must decide on choosing. The login page will distinguish the user depending on if they are a tenant or a landlord. (Landlord will be taken to the landlord side of

the application while the tenant is taken to the tenant side) The sign-up page will ask the user for their details and save them into Firebase's API database. Once the user has signed in, there's will be five options to choose from, the chat feature where the landlord is able to chat with their tenants, the properties page where landlords can view their own properties, documents page where they can upload and access important documents, track payment page where the landlord can check if tenants are paying rent on time and lastly a set door code where the landlords are able to send a code to the Raspberry pi device and to the tenants so they are able to unlock the door.

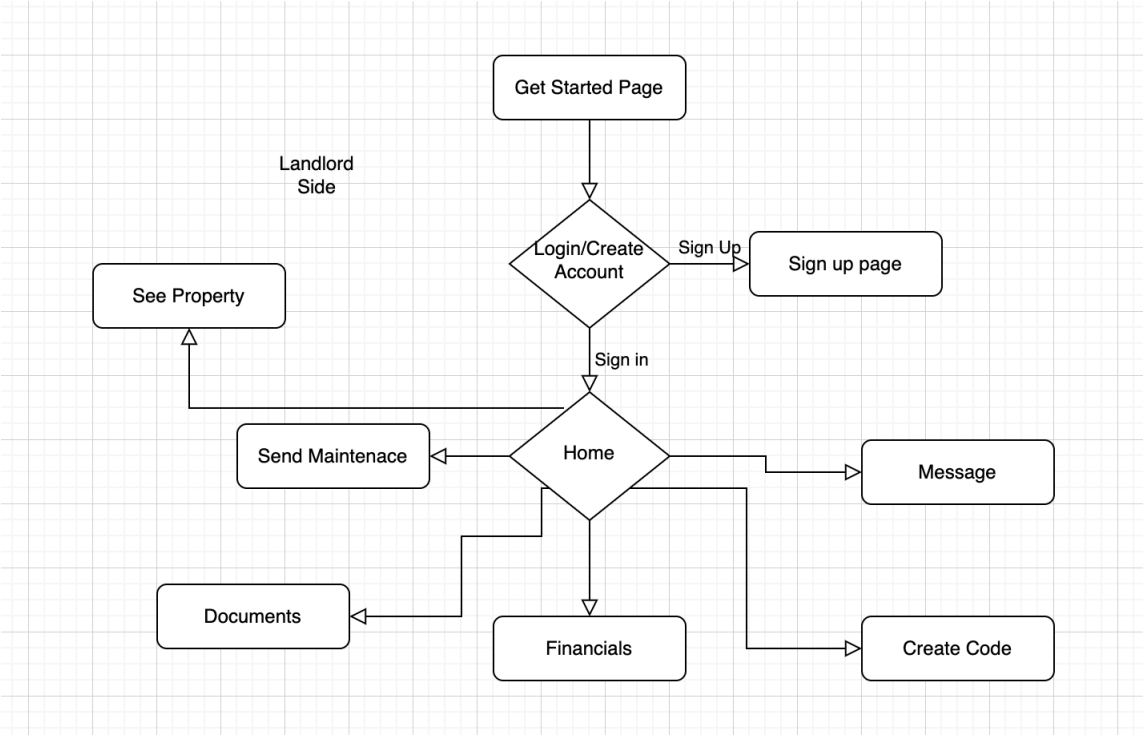

Figure 5.1: Landlord Flowchart

5.0.3 Tenant Side

This flowchart will show the direction in which the tenants will go through the application. Similar to what the landlord will go through, the user will be greeted by the get started page, where the user will have two

options to sign in or create an account. If we want to distinguish what user will be taken to what page within the application, a data set will be created in Firebase which will contain tenant and landlord details, once the user signs in with a particular account, it takes them to the relevant page within the application. The tenant will have four same options as the landlord application but one particular page which is different on the landlord side is the inclusion of the maintenance page where tenants can get vital and important details on repairs to the property.

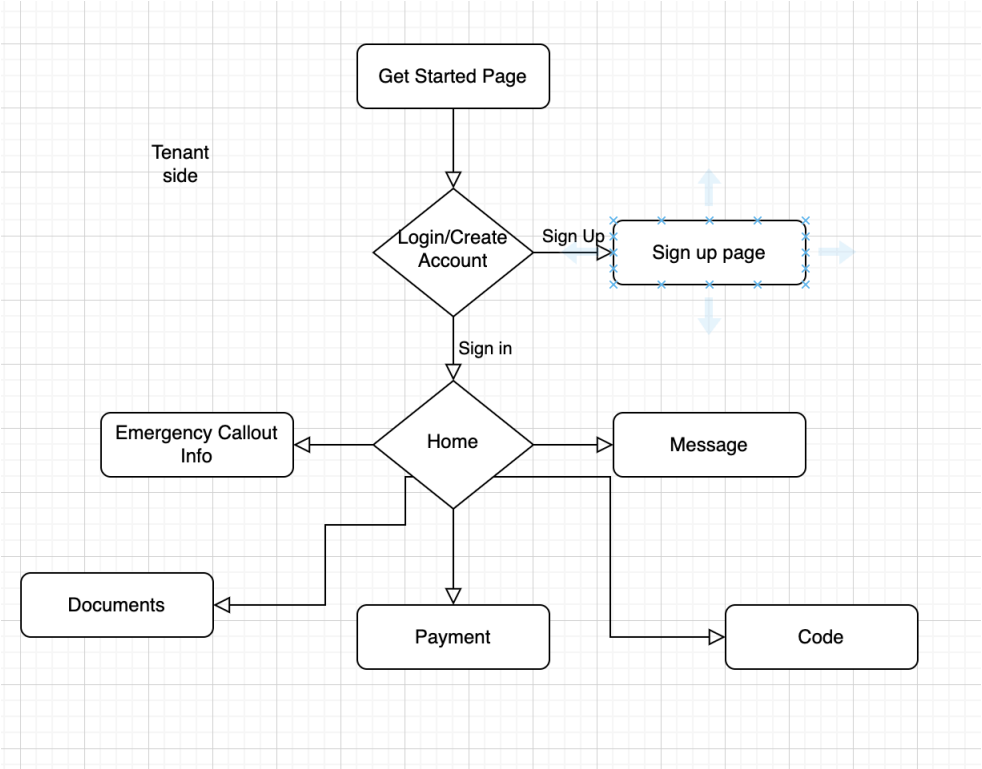

Figure 5.2: Tenant Flowchart

5.0.4 Low-Fidelity Prototype of Mobile App

A Low-fidelity prototype is a vital component of the design of a mobile application. This design phase allows developers and designers quick and cost-effective mock-ups of their mobile applications and gathers user feedback in the early stages of development.(De Sá et al. 2008) By focus-

ing on the core features and functionalities of the app, low-fidelity prototypes enable designers to identify potential design flaws and make the necessary changes before implementing the feature into the main application. To design our low-fidelity prototypes, we will be using the Axure RP10 software, this software allows designers to create interactive prototypes quickly and easily with its drag-and-drop feature and provides user testing because it simulates the user's experience.

Figure 5.3: Sign In Design

This is what the user will see when they choose the sign in button from the first view, as you can see, it will have a label telling the user that they're on the sign in page by the label on the top of the screen displaying the words "Sign In". Then it will have two text fields in the sign in design, one with an Email text box and the other one being a password text box, this is a simple design for older people in mind so that they can enter their details. (The password text box should hide what the user is

entering for privacy) Lastly, we have implemented the sign in button so that the user can sign in.

Figure 5.4: Register Account Design

This design will be similar to the sign in page, however in this design there are more text boxes that will function differently from the sign in page. For the register page, we will want to get the name, email and occupancy of the user. With most register pages the name text box will show first, second will be the email text box and lastly would show the occupancy text box. When the user has filled out these text boxes, the user show and then clicks the ”Register Account” button to submit this data.

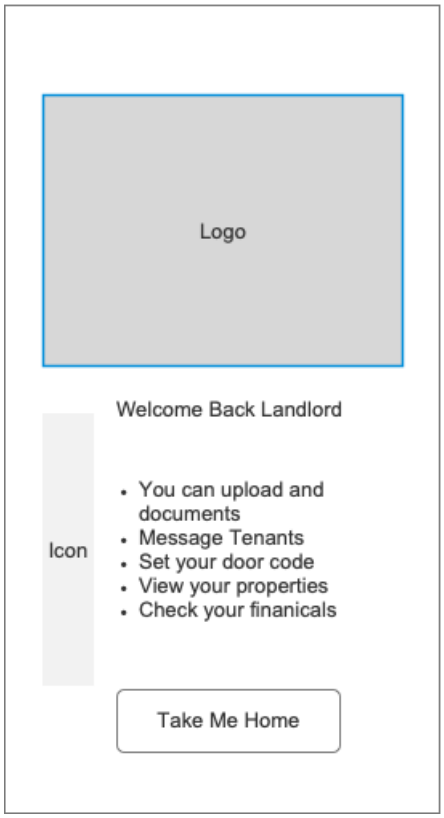

Figure 5.5: Landlord Welcome Page

Depending on what the user logs into(Landlord or Tenant Account) this design will focus on if the user chooses to log into their landlord account. As we can see, we will import the logo at the top of the screen with a welcoming message for the landlord. Below this will indicate what the tenant can do in our application. The icon box to the side will display icons of 'folder' 'message' 'padlock' 'house' and lastly the 'growth chart' for the landlord. The button below will just tell the user if they click on the 'Take Me Home' button, it will take the user to their homepage.

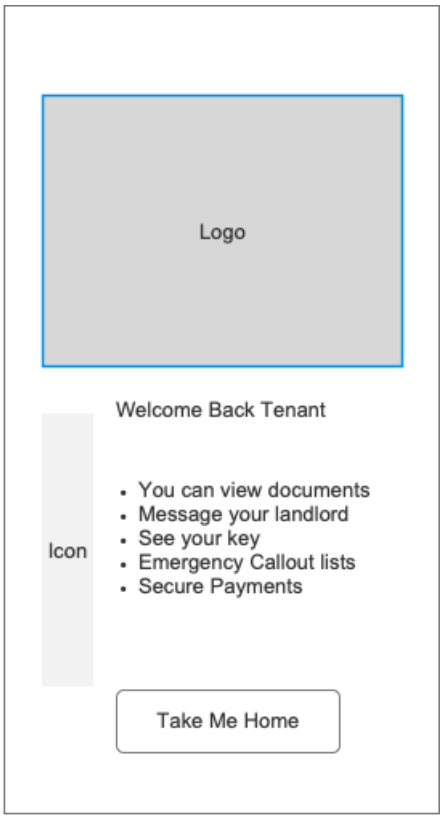

Figure 5.6: Tenant Welcome Page

Same as the Landlord's welcome design, it will have the same design properties however, this design will tell the tenant that they can View the document, message the landlord, see their key, find their emergency callout list and can pay their landlord within our application. The button below this will take the tenant to the tenant homepage. This is an easy and simple design for both landlord and tenant to understand.

5.0.5 The Design for the App

The homepage of our app will be the first point of contact between the user and the application. As such, the design of the homepage determines the success of our app. As mentioned before in the research, most of our users will be between the ages of 55-64, so the design needs to be easy for the user to understand. The landlord's homepage will have seven buttons in total, which are the "Financial" page which shows the

landlord the overview of spending on their properties, ”Set Code” which allows the landlord to set a door code in the text box, ”My Properties” will use the Apple Maps API to show the locations of where their properties are and also provide information about them, ”Maintenance” page will ask the landlord to input details of the tenant and the work to be carried out in the building, ”Documents” is where the landlord is able to create a file and save it into a shared folder, it will also allow them to view the folder as well, ”Messages” allows the communication between user and tenant and lastly the ”Sign out” button will take the user back to the sign in page in our application. Listed below will show all app pages designed for the landlord.

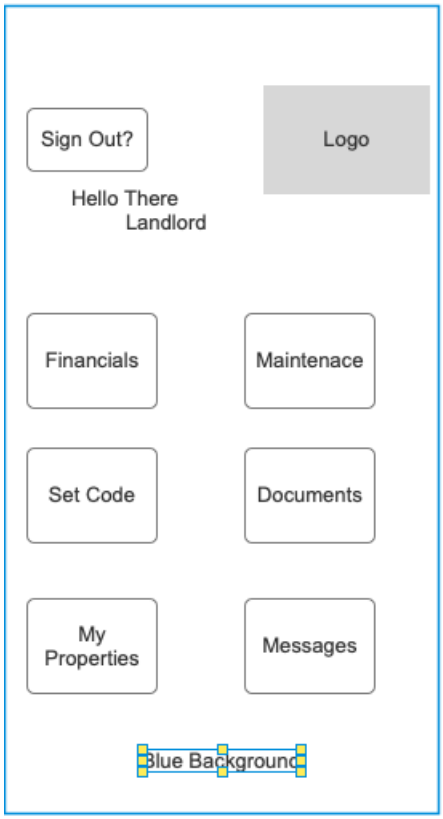

Figure 5.7: Landlord Homepage

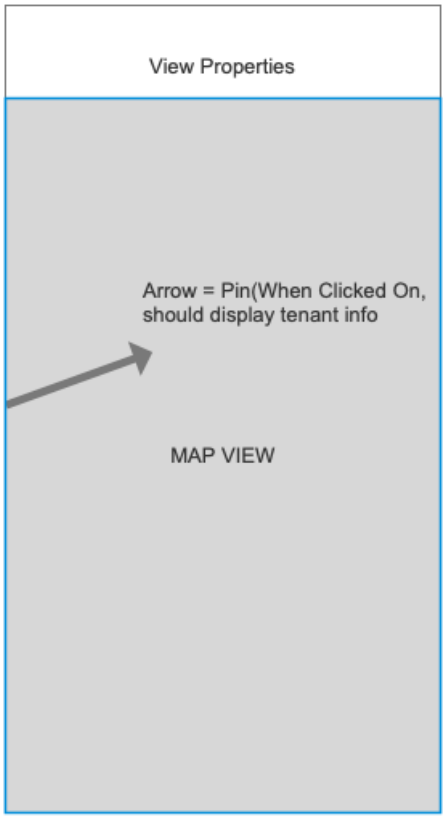

Figure 5.8: The properties page for the landlord

Figure 5.9: Maintenance page for the landlord

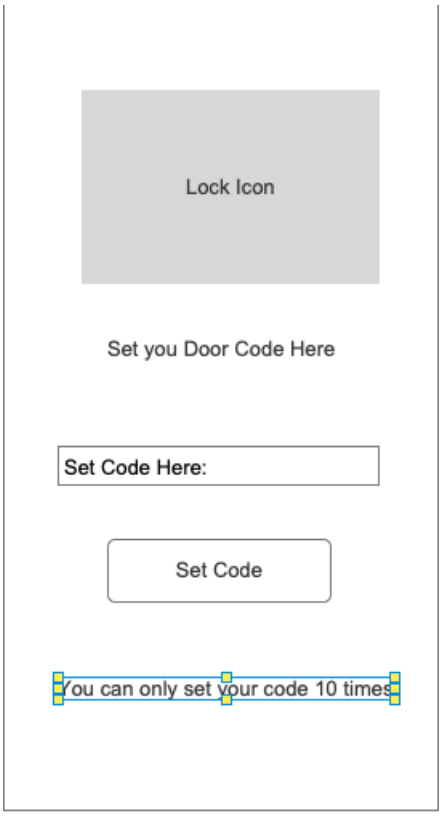

Figure 5.10: Set Door Code Page for the landlord

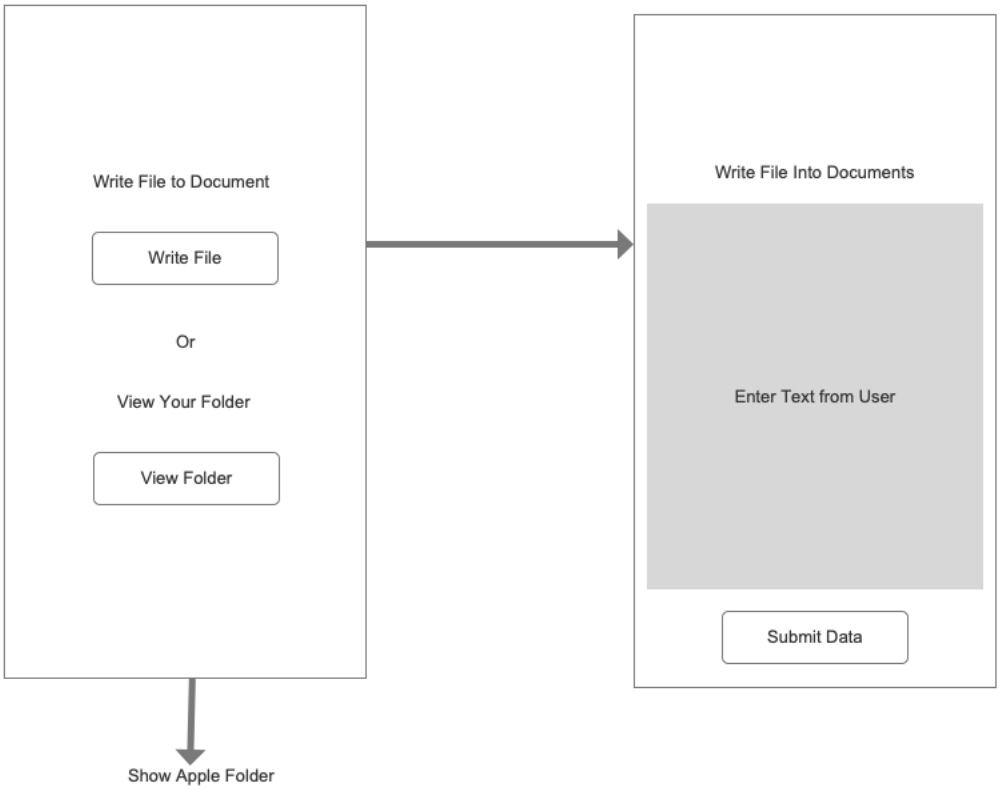

Figure 5.11: Documents page for the landlord

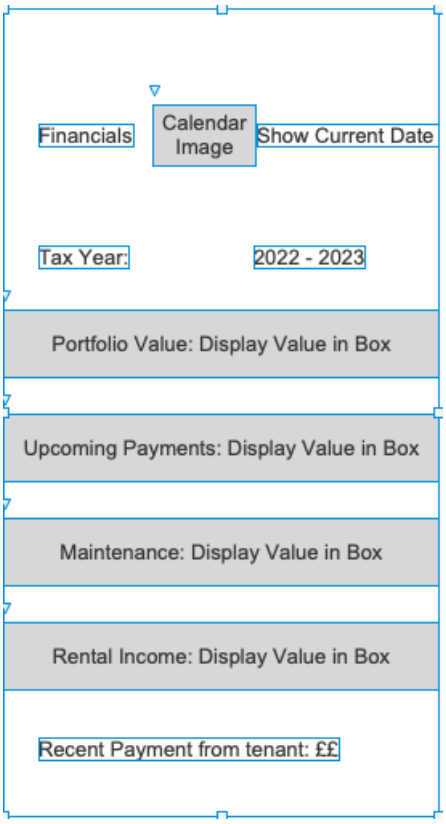

Figure 5.12: Financials page for the landlord

The tenant homepage will also contain seven buttons however, this will be slightly different because the tenant will have different needs from the landlord. The ”Sign out” button will be on the top left of our screen, the same as the landlord design, We introduced the ”Pay” button so that tenants can pay their rent to the landlord, ”See Door Code” is where the tenant is able to see their door code which is set by the landlord, ”My Property” button shows the tenant the location using Apple Maps API with the current status of the household, ”Documents” button will just show the tenant what files are in the folder that the landlord has submitted on the other side, ”Emergency Callout” will display a list of numbers that the tenant can call from with the extra option of sending a request to a landlord and lastly the ”Messages” button allows the communication between the landlord and tenant by text. Listed below will show all app pages designed for the tenant.

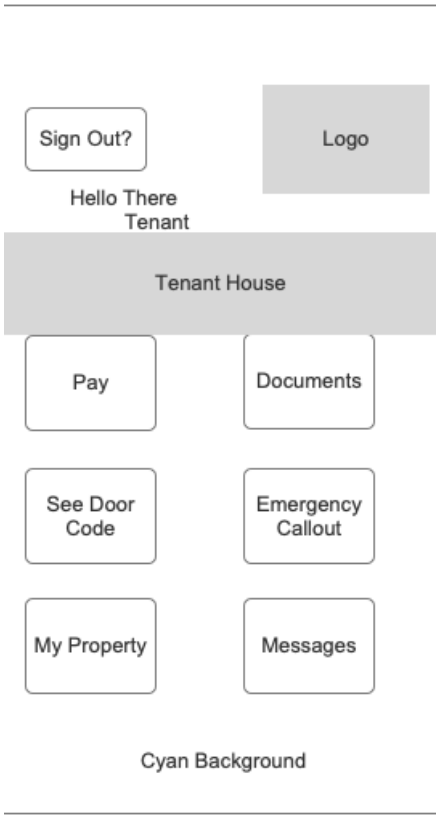

Figure 5.13: Homepage page for tenant

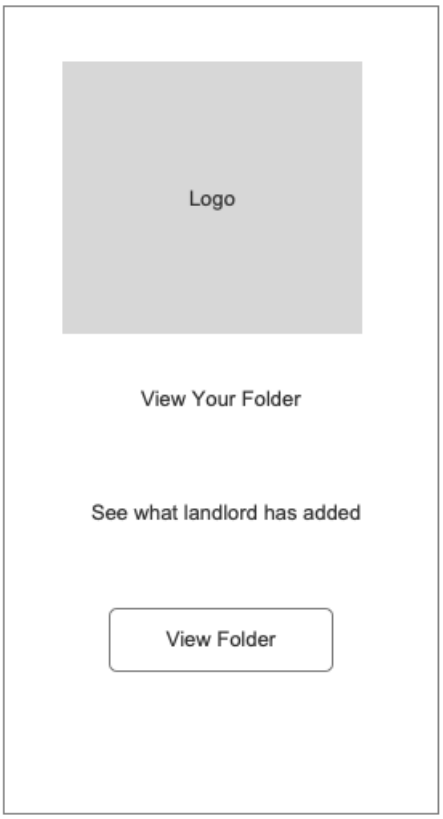

Figure 5.14: Folder page for tenant

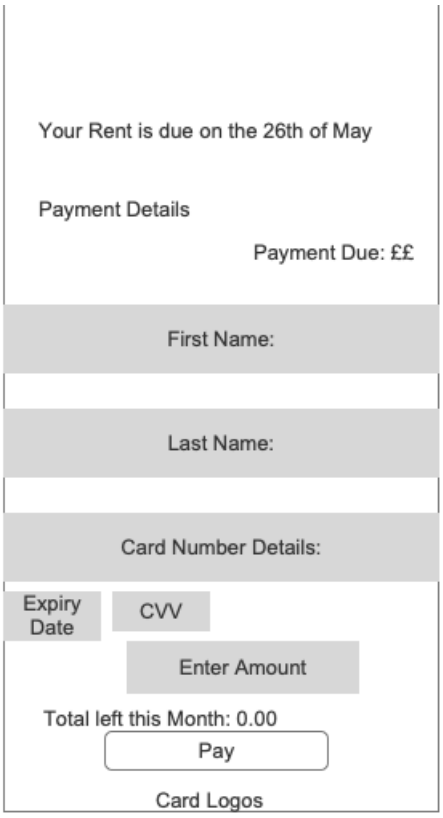

Figure 5.15: Payment page for tenant

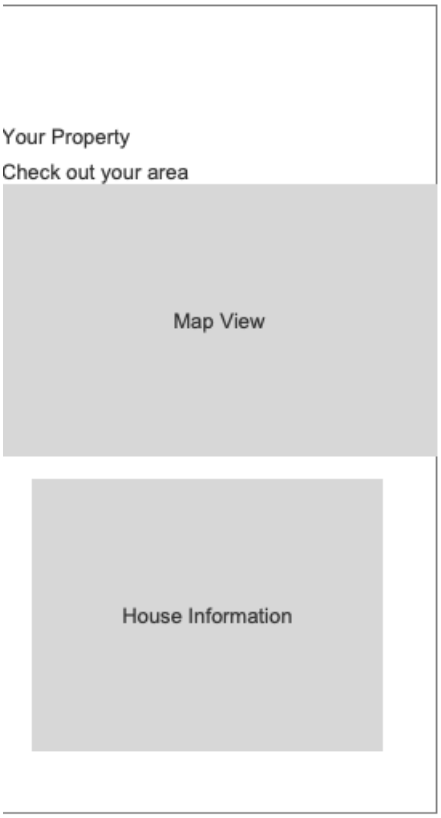

Figure 5.16: Your property page for tenant

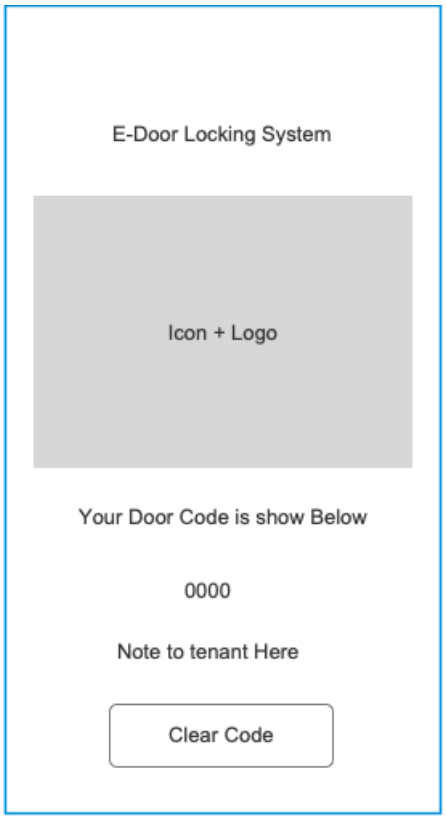

Figure 5.17: Door code page for tenant

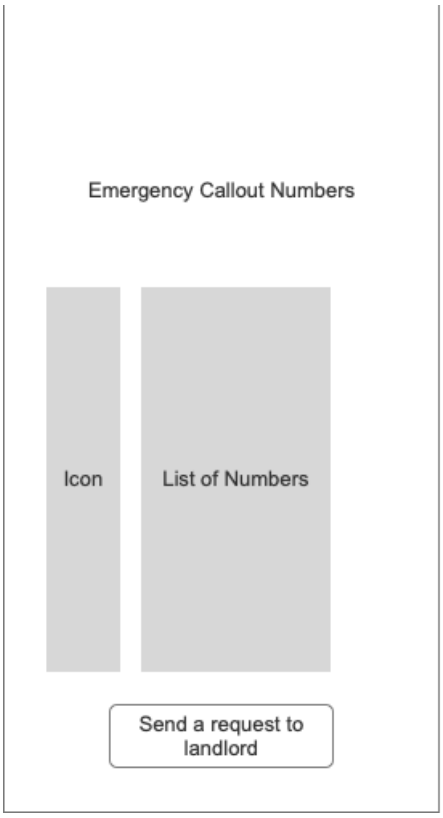

Figure 5.18: Emergency Callout Numbers for tenant

Chat Feature For Both Landlord and Tenant: This last design will show what the chat feature within our app will look like for landlords and tenants. A chat feature or application allows users to communicate from a great distance according to (Henriyan et al. 2016), which is a perfect scenario for landlords if they are not available or in a different country and still want to communicate with their tenants. From researching popular designs for our chat, the most basic but easy for the user to understand would be the simple chat logs (Texts or communication data between sender and receiver) at the top of the user interface while the text box for the user to enter information would be at the bottom left of the screen and the send button would adjacent to that text box. This design is universal and can be seen in popular applications such as WhatsApp, Facebook, iOS Messages, Snapchat and etc. Implementing this design shown below would allow tenants and landlords to have an easy understanding of how to communicate with each other using the Renetic

app.

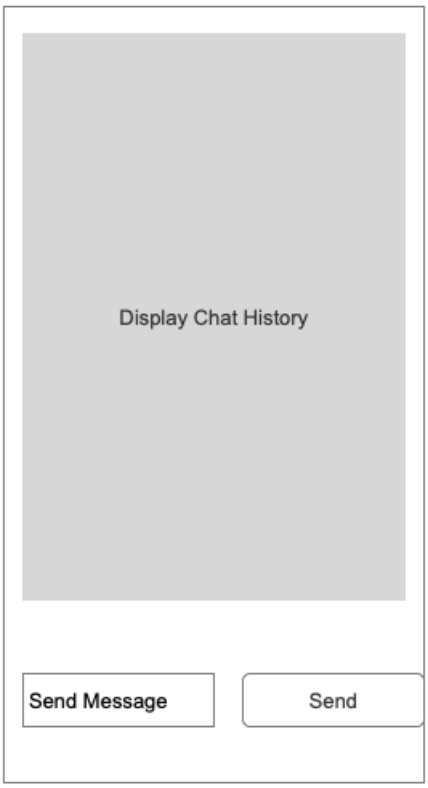

Figure 5.19: Chat Feature for both landlords and tenants

5.0.6 Raspberry Pi Door lock Security

System Design:
In the figure below (Figure5.3) shows the design of how the IoT door security will work. The landlord will send the code using the Renetic App over the Internet. Firebase will then save what the user has entered into their database. The Raspberry Pi will then retrieve this data over the internet and get the newest code that has been entered. A Python script will retrieve this code from the Raspberry Pi and set the new code for the door lock. The tenant will now log in to the app and see their new door code, they will enter this code which was given to them by the app into the keypad which will unlock the door for them.

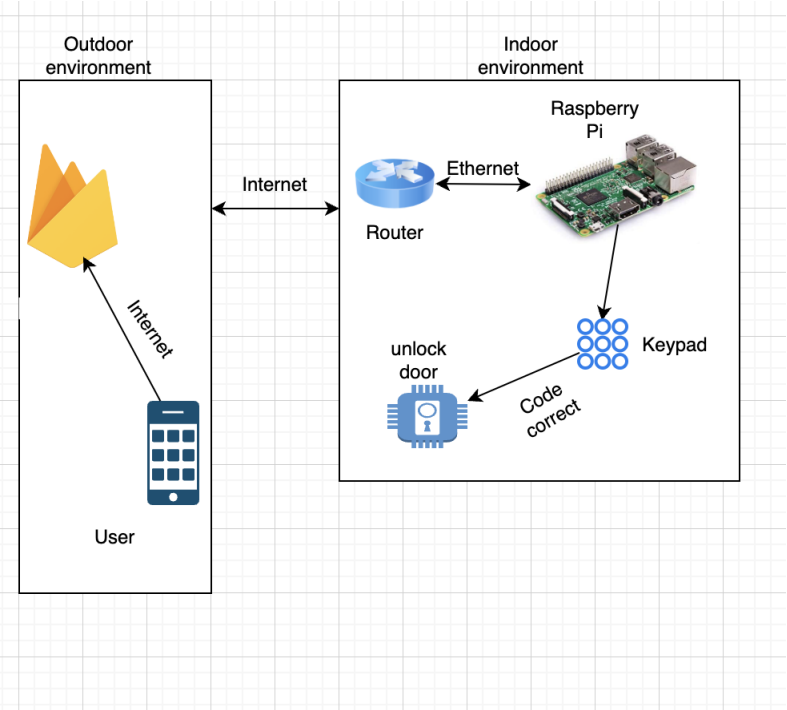

Figure 5.20: The architecture of IoT Renetic Door lock

This design shows what the tenant will see on the LCD(Liquid-crystal display). The tenant will be greeted by ”Enter Renetic Code” on the upper screen, when the user types in a code using the keypad, it will display the number on the bottom part of the screen.

LCD Display

Enter Renetic Code:
1234

Figure 5.21: LCD First Screen View

The second design will show if the tenant enters the right code into the

door lock. It should display a simple message saying ”Code Successful”, with the buzzer beeping once to indicate to the tenant that the code was successful.

Figure 5.22: LCD Code Successful

The last design will show if the tenant has entered the wrong code into the door lock. It should display a message saying ”Wrong Pin”, with the buzzer beeping three times to indicate to the tenant that the code was unsuccessful.

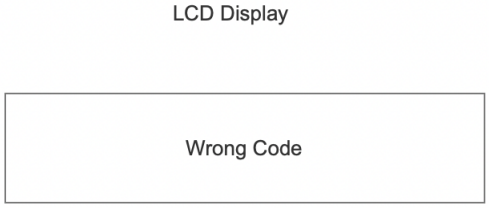

Figure 5.23: LCD Code Unsuccessful

5.0.7 Software Development Life Cycle

Why Is The Software Development Life Cycle Important For Mobile Developers And Their Clients:
The Software Development Life Cycle (SDLC), which offers a methodical approach to the development process, is essential in the creation of mobile apps. The SDLC makes sure that every step of the process, from gathering requirements through deployment, is well-defined and

organized, which enhances the quality of the mobile application. Mobile app developers can avoid costly delays and problems by using a disciplined approach and identifying potential problems early in the development process. In order to ensure that the final result satisfies the client's expectations, the SDLC also aids in creating clear communication channels between the development team and the client. The successful delivery of high-quality mobile applications is largely dependent on the SDLC.((Leau et al. 2012)) For us, we will be following the waterfall software development life cycle which includes the requirements, design, development, testing, deployment and maintenance phases.

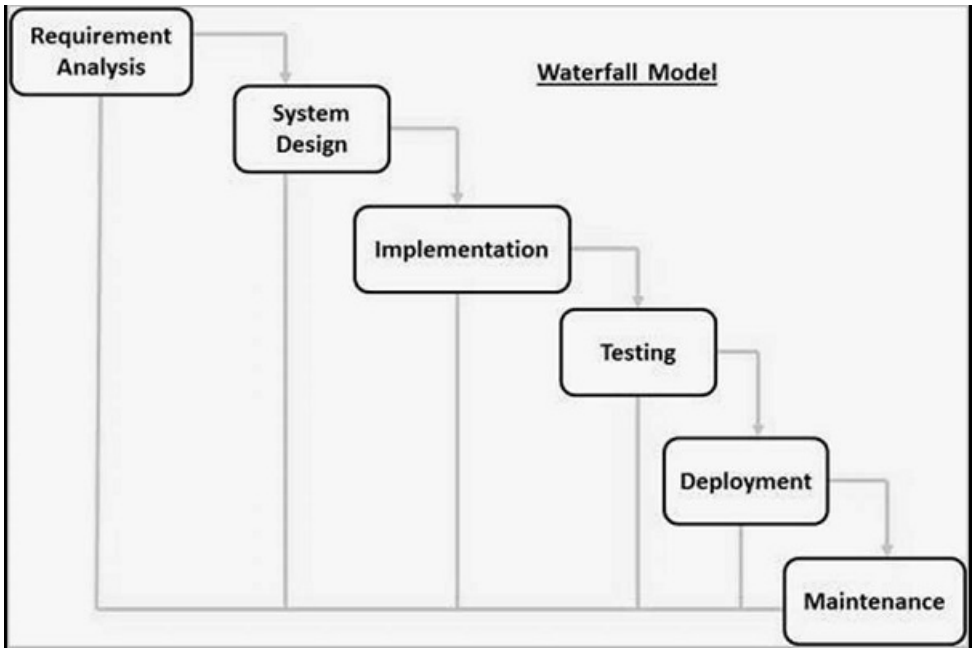

Figure 5.24: Waterfall Diagram

Chapter 6

Implementation

(Note: The code for the application of Renetic and security door look will be in the appendices as well)

6.0.1 Swift vs Other Programming Languages

In 2014, Apple introduced the Swift programming language to the world, it was built to be the successor of another programming language called Objective-C and was aimed at developers to develop IOS apps. Since the release of Swift, it has become a popular programming language for mobile app development according to (Rebouças et al. n.d.) There are several reasons why we opted to choose the Swift programming language for the development of Renetic, this would include the ease of use of the IDE (Integrated Development Environment) and the language itself, the safety and the performance. One of Swift's key benefits, when we were creating the application for Renetic, is how simple it is to use. Swift was designed to be simple for programmers like us to utilise in comparison to other programming languages like Java, Python, and Rust that rule the app market. The compact and expressive qualities of this new language allowed us to create code more rapidly and precisely. A lot of Swift's features also make it simpler to develop mobile apps, such as the Storyboards, which let us drag and drop tools into our application.

6.0.2 Swift Storyboard Login Phases:

Choose Option: When the user firsts open the Renetic App, they are introduced to the splash screen first, A splash screen mostly involves the company logo and some text to indicate what the user is using. This is

needed in our application because when the application is running, we look at a blank white screen for a time of 4 seconds, this is not suitable for developing applications and will deter our landlords and tenants not to use the app because they may think that the app is not well made, So implementing the splash screen was needed. Moving on to the choose option page, this is a simple design that gives the user an option to register an account or sign in. We decided to ask the user to fill out a ”Request account form” because we wanted to phase out unwanted users in our application (provides another level of security for our app) and it loads the application faster because of the fewer data being sent and received to the Firebase database. When coding the ’Choose option’ page, there wasn’t much to it because the storyboard allows us to create buttons that can go to any view controller (Single User Interface) in our app.

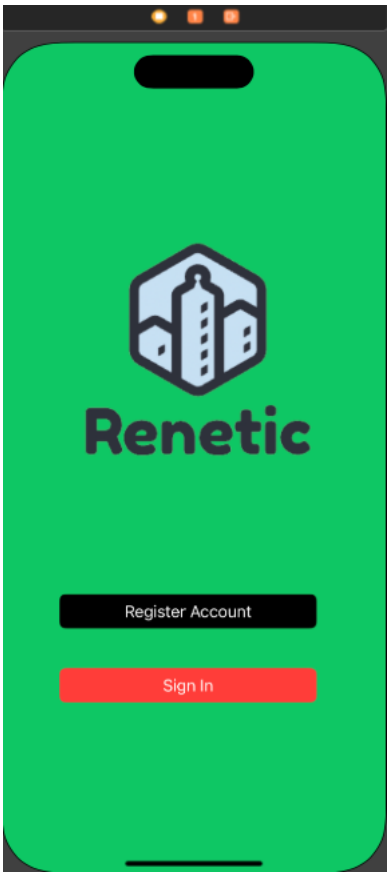

Figure 6.1: Choose Option page

Code for the Options page:

```swift
//
//  ReneticMenuView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 01/04/2023.
//

import UIKit

class ReneticMenuView: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }


    @IBAction func createAccountBtn(_ sender: Any) {
        // This button will take the user to the create Account Page

    }



    @IBAction func logInBtn(_ sender: Any) {

        // This button will take the user to the login page
    }
}
```

Figure 6.2: The back-end for the code option page

Request Account Page: When the user clicks on the register account button from the first screen (Figure 6.1), they're taken to this screen (screen shown below). Here we will be getting information about the user on this screen. Firstly we imported Firebase packages in our package dependencies, doing this allows communication between our app and Firebase. We put in three text boxes which are "Name" "Email: and "Landlord / Tenant", three labels as well that instruct the user what to enter into the text boxes and a button that submits the data into Firebases Firecloud storage to be stored. From stopping the user from entering null data into the storage, we implemented validations which are the UIAlerts that check if all text boxes have data in them, if there is no data it will stop the data from being sent to the storage and tell the user "They must enter all fields". When the user does enter the correct data, it will create a folder within Firebase storage and save what they've entered.

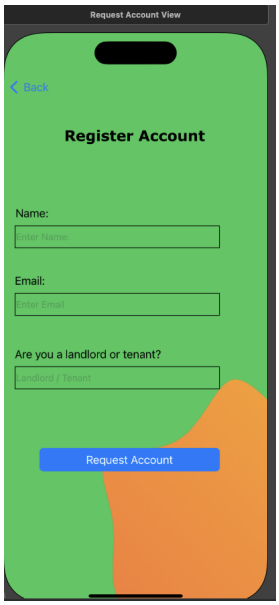

Figure 6.3: Register/Request account details

Code For Register Account Page:

```
//
//  RequestAccountView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 02/04/2023.
//

import UIKit
import Firebase


class RequestAccountView: UIViewController {

@IBOutlet weak var nameTextField: UITextField!
@IBOutlet weak var choiceTextField: UITextField!
@IBOutlet weak var emailTextField: UITextField!

override func viewDidLoad() {
 super.viewDidLoad()


 // Do any additional setup after loading the view.
    }
```

```
@IBAction func requestAccount(_ sender: Any) {
guard let text1 = nameTextField.text,!text1.isEmpty,
 let text2 = choiceTextField.text, !text2.isEmpty,
let text3 = emailTextField.text, !text3.isEmpty
 // Add additional guard statements for any additional text fields
else {
let alertController =UIAlertController(title: "Error", message:"
Please enter information in all text boxes", preferredStyle: .alert)
alertController.addAction(UIAlertAction(title: "OK", style: .default, handler: nil))
present(alertController, animated: true, completion: nil)
            return
        }

//If user does not enter any text into the textfield,
it will display an UIerror message to the user



guard let name = nameTextField.text,
let email = emailTextField.text,
let choice = choiceTextField.text else{
            return
        }

let db = Firestore.firestore()
let data = ["name": name, "email": email, "choice": choice]
db.collection("account_request").addDocument(data: data){error in
if let error = error {
print(error.localizedDescription)
} else{
 // This Code will save the users data into the Firebase Database,
 so that people at renetic can review their request and set them a account
            }
        }

    }
}
```

Sign In Page: If the user has an account with Renetic, they will click on the ”Sign in” button from ”Figure 6.1”. Here the user will see two text boxes which are the email text field and the password text field with the sign in button. The email text field will be a normal text box while the password text field will be a secure entry, this is where the user enters text into this text box but it does not show what they have entered instead it shows ”dots”. This is commonly used in applications and websites for security reasons. Now using Firebases authentication, it checks if there are users in the authentication database, if the user enters an incorrect user or gets their email or password wrong, it will display a Firebase

authentication error saying ”Incorrect password or can’t find the user”. If the user enters the correct email and password, depending on the user account (Landlord or Tenant) it will take them to different parts of our application. This is important because it will direct the right user to the correct features within the application.

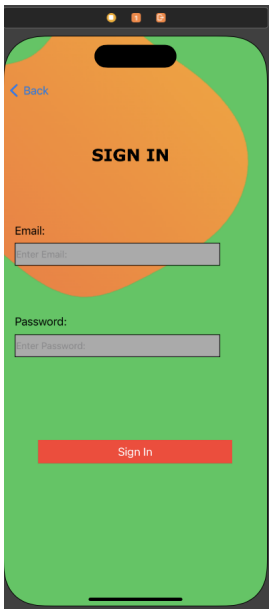

Figure 6.4: Sign in Page

Landlord and Tenant Welcome Page: If the user chooses to sign in to the landlord, it will take them to the landlord side, here the user will see a list of what they can do in the application as the landlord. The ’Take me home’ button will take the user to the landlord’s homepage. The tenant welcome page would be similar to the landlord. However, it will show a different list of what the tenant can do and will take the user to the tenant’s homepage.

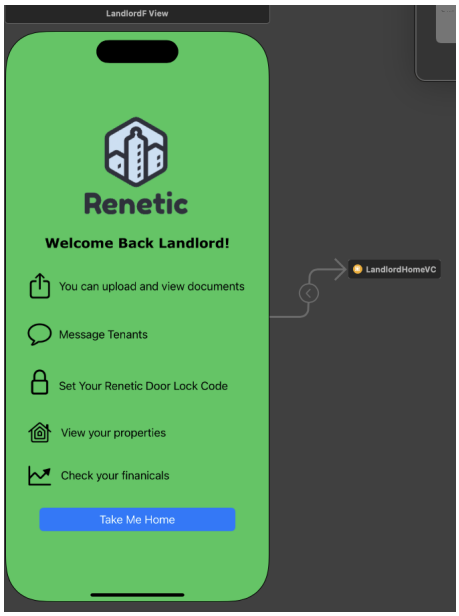

Figure 6.5: Landlord welcome page

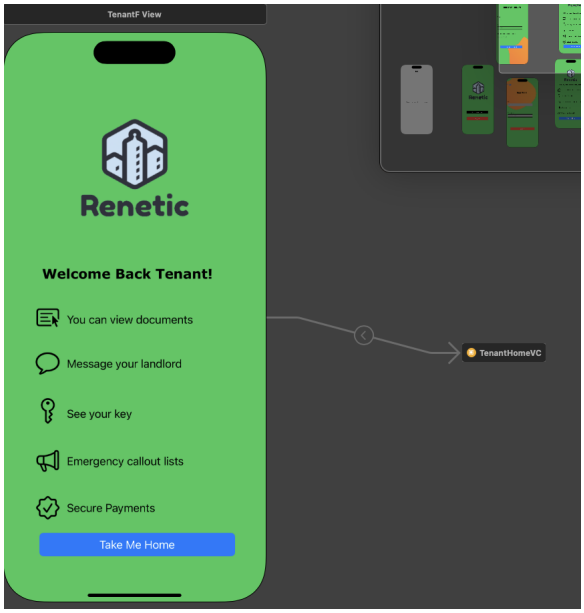

Figure 6.6: Tenant welcome page

Code for sign-in page:

```
//
//  SignInView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 03/04/2023.
//

import UIKit
import Firebase
import FirebaseAuth

class SignInView: UIViewController {

@IBOutlet weak var emailTextField: UITextField!
@IBOutlet weak var passwordTextField: UITextField!
override func viewDidLoad() {
    super.viewDidLoad()

        // Do any additional setup after loading the view.
    }


@IBAction func signIn(_ sender: Any) {
guard let email = emailTextField.text,
let password = passwordTextField.text else {
            return
        }

        Auth.auth().signIn(withEmail: email, password: password) { [weak self]
        authResult, error in
            guard let strongSelf = self else {
                return
            }

            if let error = error {
                let alertController = UIAlertController(title: "Error",
                message: error.localizedDescription, preferredStyle: .alert)
                alertController.addAction(UIAlertAction(title: "OK", style: .default,
                handler: nil))
                strongSelf.present(alertController, animated: true, completion: nil)
                return
            }
            // When the user sign in with their account

            if let email = authResult?.user.email {
                if email.contains("landlord123@renetic.com") {
                    let storyboard = UIStoryboard(name: "Main", bundle: nil)
                    let secondController = storyboard.instantiateViewController
                    (withIdentifier: "LandlordHome")
                    self?.present(secondController, animated: true, completion: nil)
```

```
                } else {

                    let storyboard = UIStoryboard(name: "Main", bundle: nil)
                    let viewController = storyboard.instantiateViewController
                    (withIdentifier: "tenantVC")
                    self?.present
                    (viewController, animated: true, completion: nil)
                }
            }
        }
    }
}
```

6.0.3 Swift Storyboard Landlord Phases

(Code for Landlord phases is in the appendices) In this section, we will show what the user will go through if they decide to log in using the landlord's credentials and explain each view controller. Firstly for the user to navigate through the application, we implemented a 'Navigation Controller". A navigation controller controls the view controllers(User Interfaces) if they have one or more child views. (This can be seen at the top left side, a 'Back ' button will appear, meaning that the application is connected). To send or receive data from the Firebase database or the real-time database in all our view controllers, we have to import the Firebase package into the Swift files, this can be seen in the appendices with (Import Firebase), (Firestore.firestore() = data will be saved in the firestore database), (Database.database() = data will be saved in the real-time database) Using our low-fidelity prototypes designs in the design section At the **homepage** we're able to see six buttons which are the 'Sign Out',' Financials',' Maintenance', 'Set Door Code',' Documents', and lastly the 'My Properties' button. Since storyboards minimise the coding process, were able to connect each button to different view controllers by simply dragging the button to each controller in the landlord storyboard file. However, the sign-out button needed coding. We decided to add a UIAlert that tells the user if they want to sign out of the application, if yes the function will take them back to the sign-in page, if they click 'no' the user will stay on the homepage. On the **financials** page, we implemented time functions into the application, this function will get the current date (Day) off the user's phone and displays it in the user interface. we produced labels that tell the landlord about their portfolio. Lastly, we called the Database.database function in this

user interface to retrieve the tenant payment and display that value in a label. The **set door code** page, has labels, text boxes and logos implemented on this page. Here we also used the Firestore.firestore function to store what the landlord has entered from that text box when the landlord clicks the 'Set Code' button, this sends the data to the storage. for the '**my properties'** page, we implemented another API which is Apple Maps, landlords would be able to see their properties and property information with the use of this API. The '**Documents**' page is where landlords can either write a file to the folder or view their folder, we used the FileManger function that Xcode provides to allow the landlord to write a file directly into the folder and also we can view the files in the folder within the application. Lastly the **'Maintenance'** would consist of five text boxes with one 'Send Maintenance' button, we used the Firestore.firestore function again to submit the data to our storage. There is validation throughout this storyboard to stop the user from entering incorrect or NULL data into Firebase.

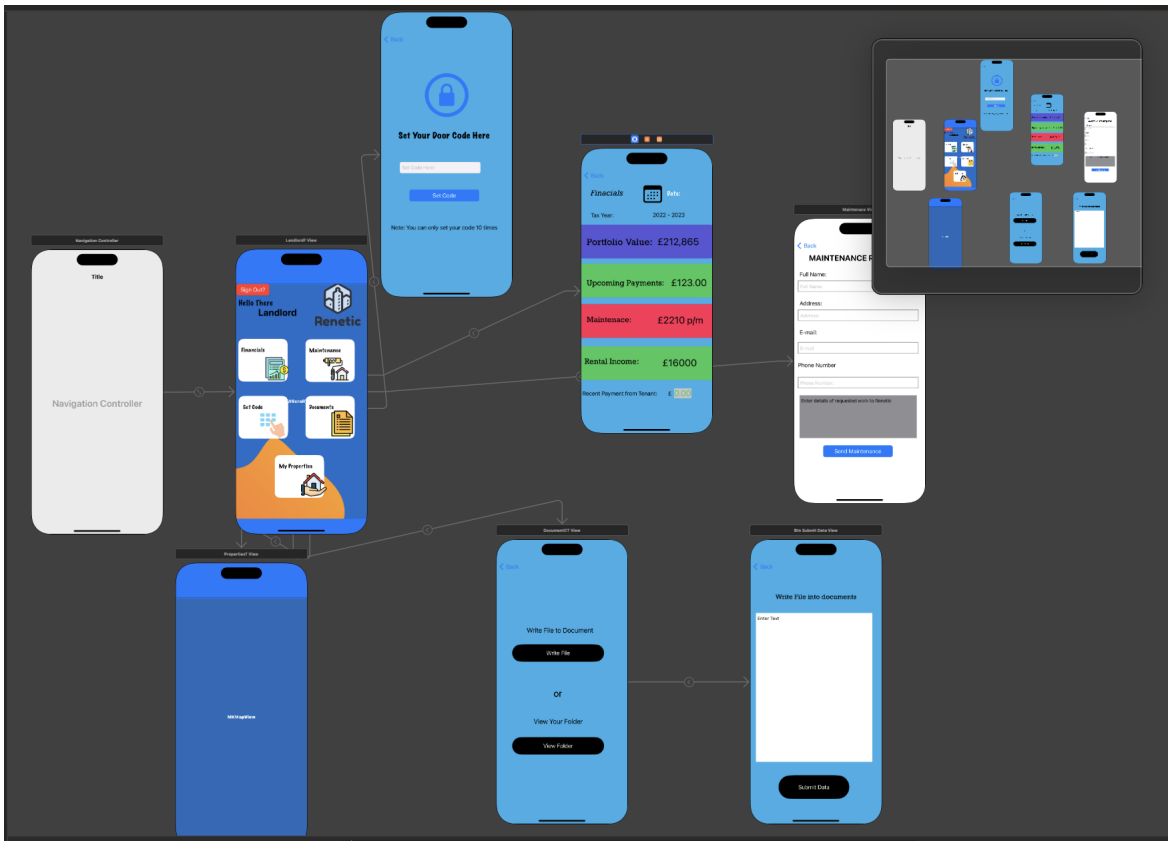

Figure 6.7: Implementation of the Landlord application

6.0.4 Swift Storyboard Tenant Phases

(Code for the tenant application will be in the Appendices)We used the same functions from the landlord storyboard phase in our tenant application. However, there are some differences between each other. Firstly starting with the **homepage**, we have the same six buttons as the landlord application for the homepage but we implemented two new features for the view controller. we imposed placing the image of the tenant's house on the homepage and using the Date/Time feature, we can either display a greeting message saying "Good Morning" or "Good Evening" depending on the time of day on the user's phone. Now moving on to the features of each view controller, the '**pay**' button allows tenants to enter their card details and pay a certain amount if they choose to do so. We used validation in this view controller because as mentioned before,

we do not want NULL data in our database. This UIAlert will say ”Payment Declined” telling the user that the payment did not go through, if the user enters the correct details, it will say ”Payment Succeeded” and saves how much the user paid in the real-time database using the Database.database function which is then used in the landlord side of the application. Now when the user has successfully done this, we used a maths function in this view controller to calculate how much that tenant has paid and show the result in the UIAlert. The **'documents '** will be similar to the landlord's page but they will not have the function to write files into the folder. **'see door code'** page is where tenants are able to see their door code for the electronic door lock, here we implemented a button where when it is clicked, the code disappears for the tenant, this provides security in our application. **Emergency Callout Numbers'** displays a list of numbers of which the tenant can call for an emergency callout to their property, underneath this is a button that when it is clicked sends a message to the firebase storage. Lastly, the **'my properties'** page uses Apple Maps API to get the current location of the tenants. This implementation of this feature is quite useful for tenants because it will show the whole local area, so they can do a whole range of activities using this.

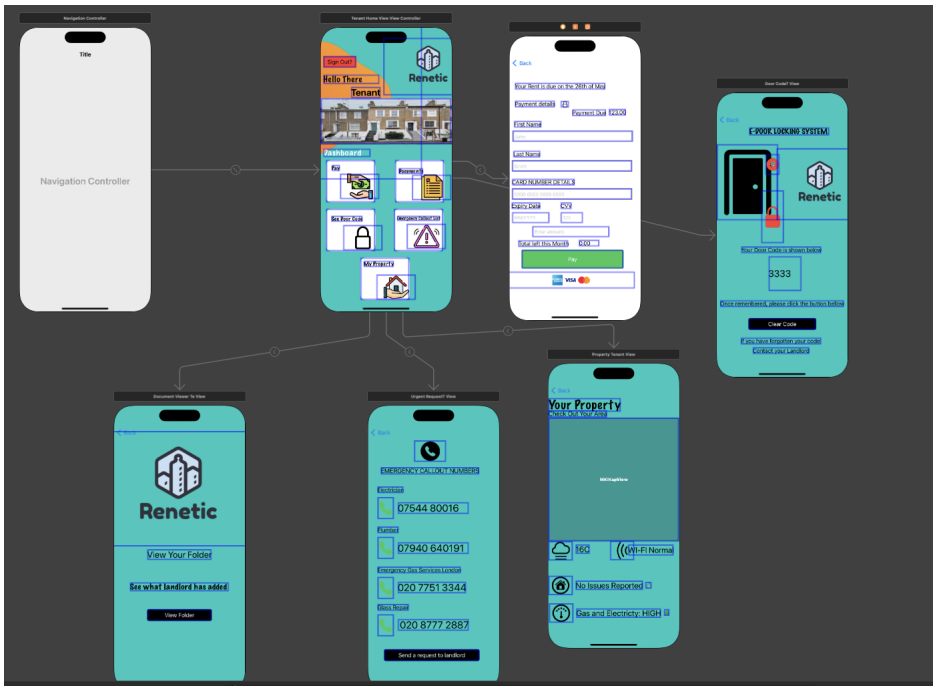

Figure 6.8: Implementation of the Tenant application

6.0.5 SwiftUI Chat Feature

When trying to implement the chat feature into the storyboards using a SwiftUI file, we were unable to do this because Xcode doesn't allow us to integrate them both together. We decided to create another application that is purely for communication. When the user launches the application from their home screen, it will prompt the user to enter their account details for Renetic(Emails and passwords will be the same as the main application) This will use the same functions as the main application for the sign-in page. When the user has signed in with either account, their able to see past conversations at the top of the screen. To make this work, we used the Firestore.firestore function to retrieve the chat data from our database. For users to actually communicate with each other, the text box at the bottom of the screen will get the user's input and the button adjacent to it will submit this data to the storage. The implementation of this feature allows communication between landlords and tenants.

6.0.6 An electronic Security door lock using a Raspberry Pi device

System Structure: The type of system provides the tenant with a convenient and secure way to control access to a door by using a numeric code. The system is built using a device called a Raspberry Pi which is a small single-board computer. The Raspberry Pi serves as the central processing unit (CPU) and interfaces with the other components such as the number pad, power supply, relay, LCD and buzzer which are connected via GPIO pins. ((A.s et al. 2021)) The embedded programming language will be implemented in the Raspberry Pi using the Python language. The smart door lock system operates over an Ethernet connection to connect to our Firebase database and a mobile app where the user can set or read the pin. When the system is powered on, the modules that are connected to the Raspberry Pi are enabled such as the keypad, relay and buzzer modules.

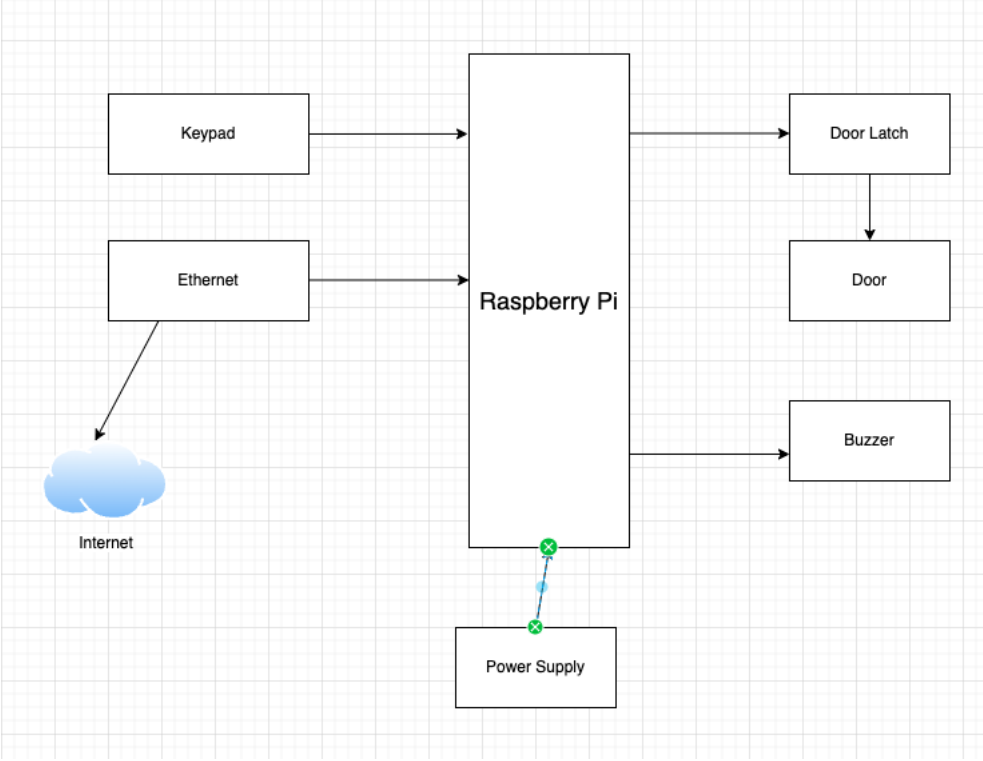

Figure 6.9: IoT Smart Door Lock System Diagram

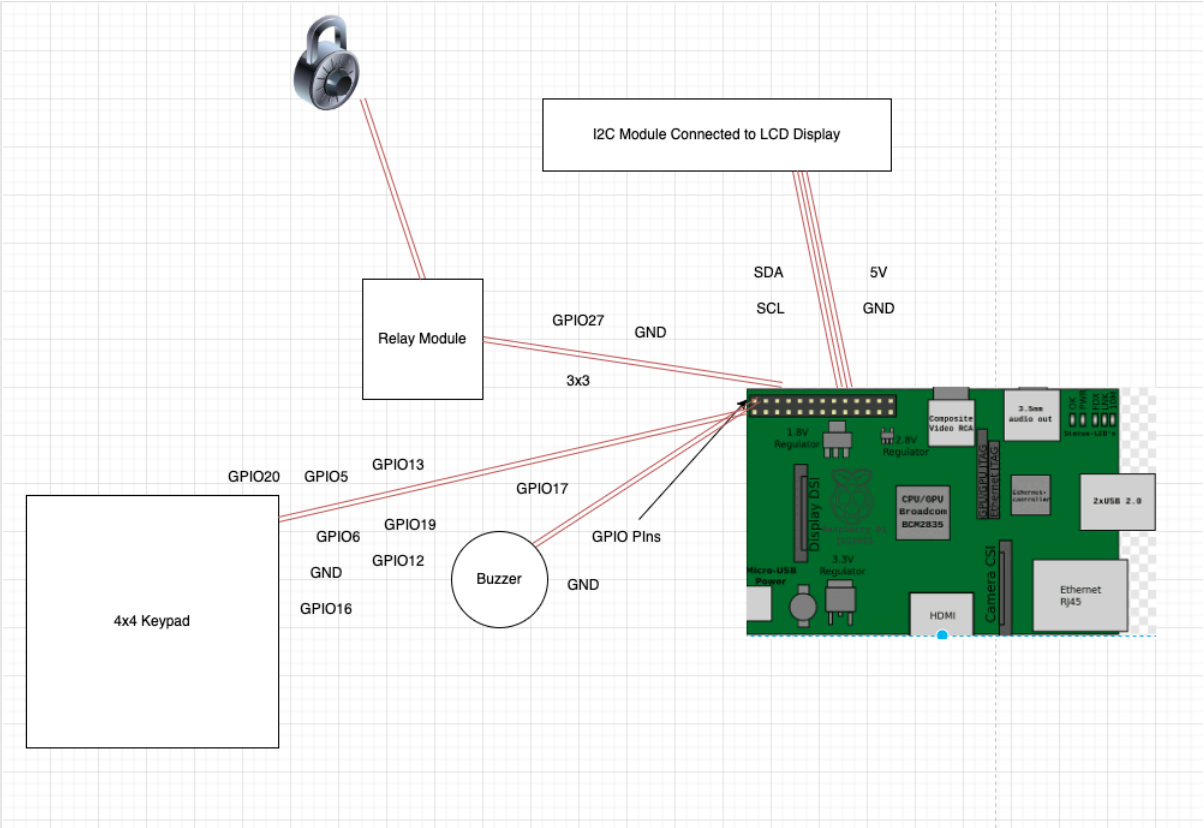

Figure 6.10: Internal Design for Raspberry Pi and Components Used

A book about Raspberry Pi layout used in implementing the Door Lock: (Gay 2018)
Keys for the internal design:

- 3x3: Is a standard size for a circuit board or module, which measures 3 centimetres by 3 centimetres
- SCL: SCL stands for Serial Clock Line. This is for a unidirectional communication line in the I2C protocol, also used for synchronising data transfer between two devices. We connect the I2C module to the LCD display.
- SDA: SDA stands for Serial Data Line. Its the bidirectional communication line used in the I2C protocol, similar to the Serial Clock Line
- 5V: This stands for 5 volts, commonly used in electronic devices we will be using this voltage level for our door lock

- GPIO: General Purpose Input Output. is a set of pins on the raspberry pi that can be configured to function as input or output, depending on the needs of the application. Instead of manually attaching each jumper cable to a pin, we will be using a GPIO ribbon cable that connects all pins at once.
- GND: Stands for Ground, and returns a path for the electrical current.

Photo of Raspberry Pi setup:

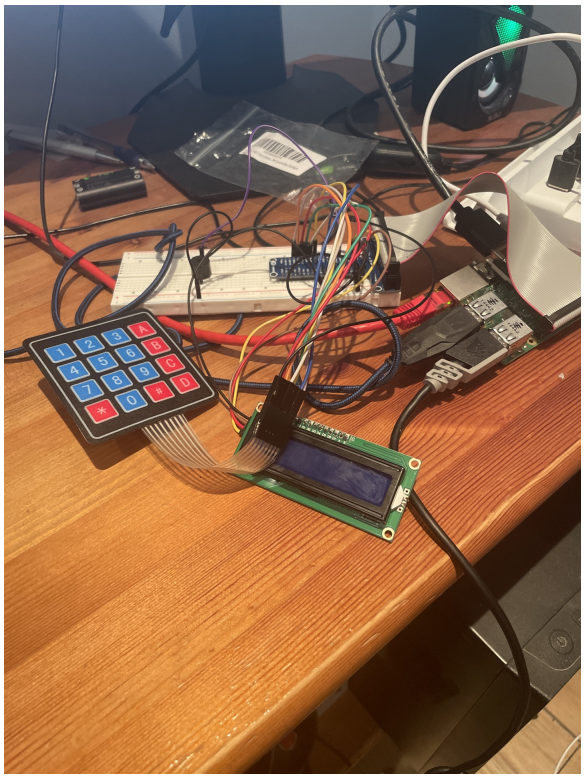

Figure 6.11: Photo of Raspberry Pi and components

With this, we were able to retrieve security door codes by using the Ethernet cable (Which provides a connection to the web, which is required to connect to our Firebase) We followed the design by correctly connecting each GPIO pin with a jumper and ribbon cable. We were also allowed to enter codes by the keypad(shown above) and the LCD displays the code that we have entered, it also checks if the code has been entered correctly and displays a message saying ”Code has been

entered correctly" in the LCD display if the code has not been entered correctly, it displays the"Wrong Code" message.

In conclusion, the implementation process is a vital component for developers to follow when creating an application. The quality of the implementation process can impact the final outcome of the application, its functionality and overall performance. If we had poor implementations for the development of Renetic, it can result in bugs, crashes, not meeting user requirements, and can produce several security vulnerabilities that can damage the app's reputation. On the other hand, if we produce a well-executed implementation process, this can make Renetic a reliable, fast and secure app. Therefore, it is important for us the developer, to pay close attention to the implementation process for the development of the app.

Chapter 7

Testing and Final Product

Testing mobile applications is a critical aspect of the software development phase, as new features are added to the application, there are times when the application has many bugs and affects the performance of the app.(Muccini et al. 2012) Mobile application testing is the process of evaluating the functionality, performance, usability, and security of an app on different mobile platforms and devices to ensure that it meets the user's expectations and requirements. In this section, we will be running the Renetic application using Xcode's simulation of an iPhone and testing what the user will go through in the application. We will be representing it by using a table to show the tests. For the testing phase, we will be doing what is called "usability testing". Usability testing is a technique that is used to develop software in the app market. This usually involves focusing on the interactive features of the app.

Table 7.1: Renetic Sign-In Feature for both the main application and Chat Testing

Test No	**Test Description**	**Expected Outcome**	**Actual Outcome**	**Pass or Fail**
1	Page loaded	Should simulate home page	Page has loaded	Pass
2	Register button takes user to different page	Takes user to another page	button shows other screen	Pass
3	Sign in button takes user to sign in	Takes user to another page	button shows other screen	Pass
4	If the user enters nothing into text boxes, the error message should display	display error message	error message displays	Pass
5	Can user enter text in the email text box in register account view	allows user to enter text	user can enter text	Pass
6	Can user enter text in the password text box in register account view	allows user to enter text	user can enter text	Pass
7	Can user enter text in landlord or tenant text box in register account view	allows user to enter text	user can enter text	Pass
8	Can we enter the Renetic Chat App using the same username and password	Sign in user	user can log in	Pass

Table 7.2: Renetic Landlord and Tenant Feature Testing

Test No	Test Description	Expected Outcome	Actual Outcome	Pass or Fail
1	Testing buttons on the homepage take the user to the correct pages	should go to the correct page	Pages have loaded	Pass
2	Testing functionalities on the set door code page work	functionalities on this page should work	All functionalities work	Pass
3	Testing if all textbox entries goes to firebase	data should be sent to firebase	data is stored in the database	Pass
4	Writing and viewing files	Files can be stored and user can view the document	Folders are written and can access files	Pass
5	Can the user pay in the payment page	allows user to pay a certain amount	User can pay	Pass
6	Is the Apple Maps API in both landlord and tenants scroll-able	Apple Maps API loads	Apple maps loads and is easy to scroll	Pass
7	Can we communicate between the landlord and tenant using the chat app	Communicate to another user	We can communicate with the other user	Pass
8	Can we clear the code in the tenant application (See Door Code)	Code should clear	Code clear	Pass
9	Can we sign out of the application	User should be brought back to the login page	User is brought back to the page	Pass
10	Do all label fields update in the view controllers	labels should update when called a function	Labels do update	Pass

Table 7.3: Door Lock System Testing

Test No	Test Description	Expected Outcome	Actual Outcome	Pass or Fail
1	Display all messages to the LCD	Messages should appear	Messages do appear	Pass
2	Test they keypad for door lock	Keypad records what the user enters	Records what we enter	Pass
3	Test if code gets data from database	Gets code from database and display to the user if they have entered correctly or incorrectly	Entire door locking system works	Pass

7.0.1 Final Product of Renetic

When testing the overall application for Renetic, The user interface was very easy to navigate throughout the app without any bugs or crashes occurring. All functionalities within the app work and there were no crashes in both Firebase or Apple Maps API's. This section will show the results of the Xcode simulator here.

Landlord Application:

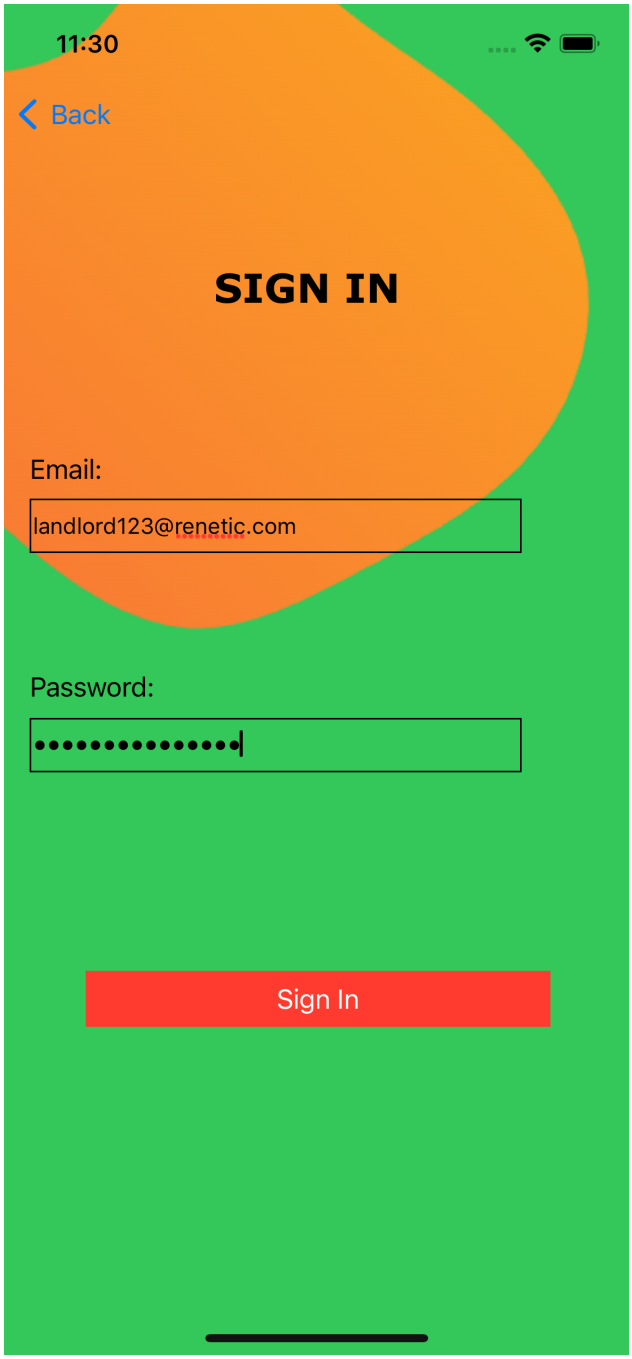

Figure 7.1: Landlord Sign In

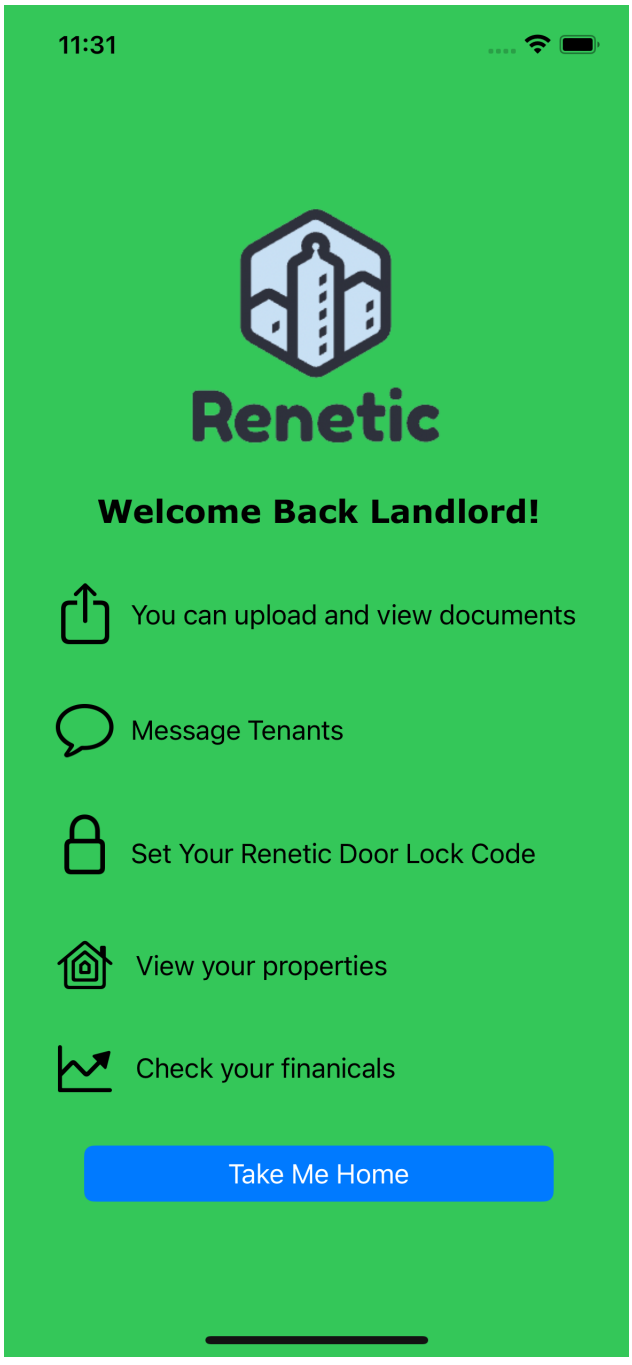

Figure 7.2: Landlord Welcome page

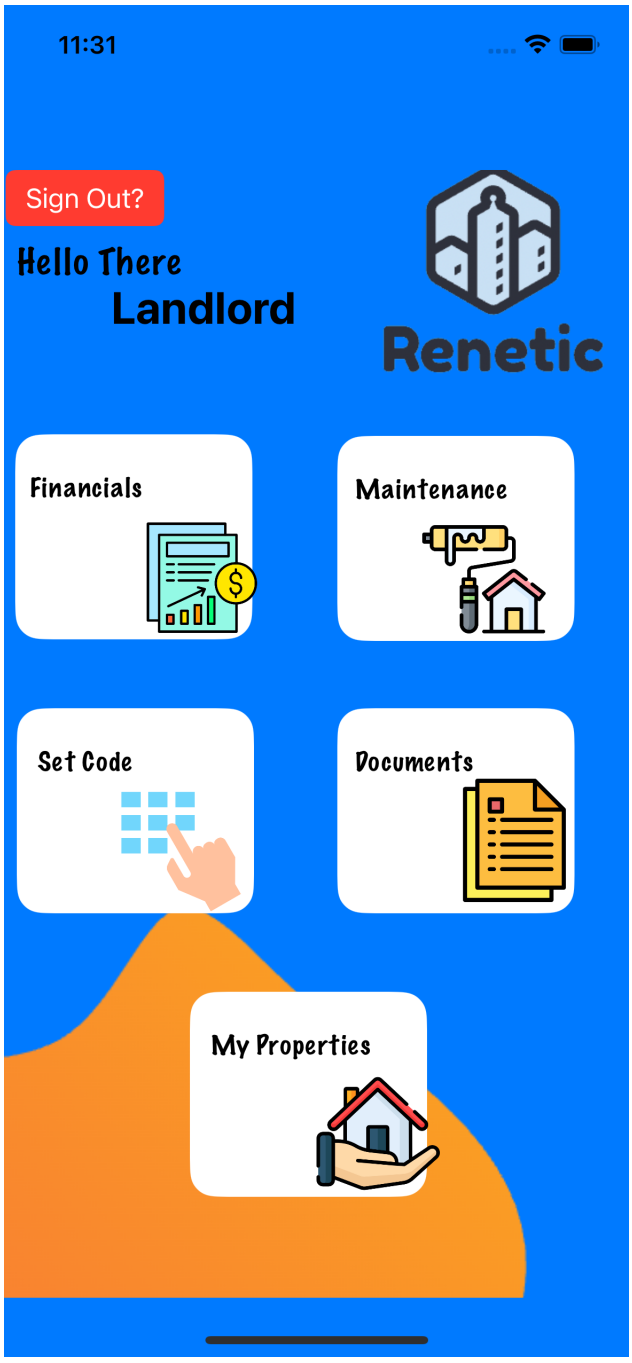

Figure 7.3: Landlord homepage

Figure 7.4: Landlord Finacials

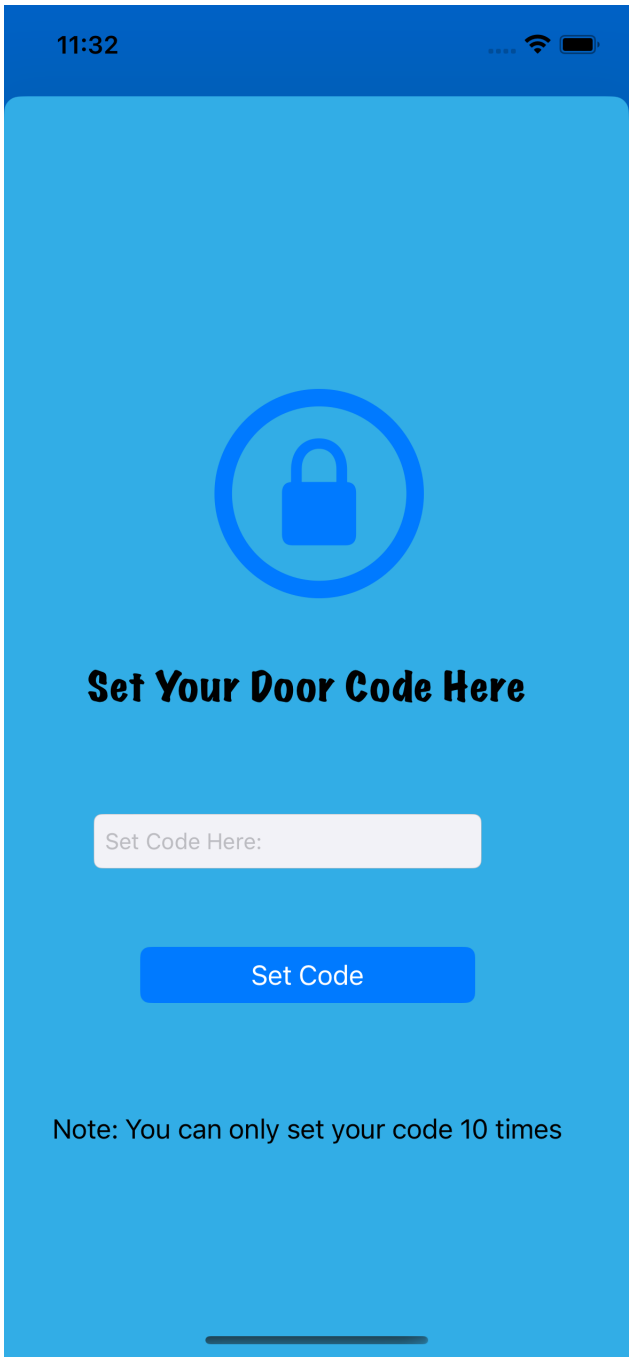

Figure 7.5: Landlord Set Door

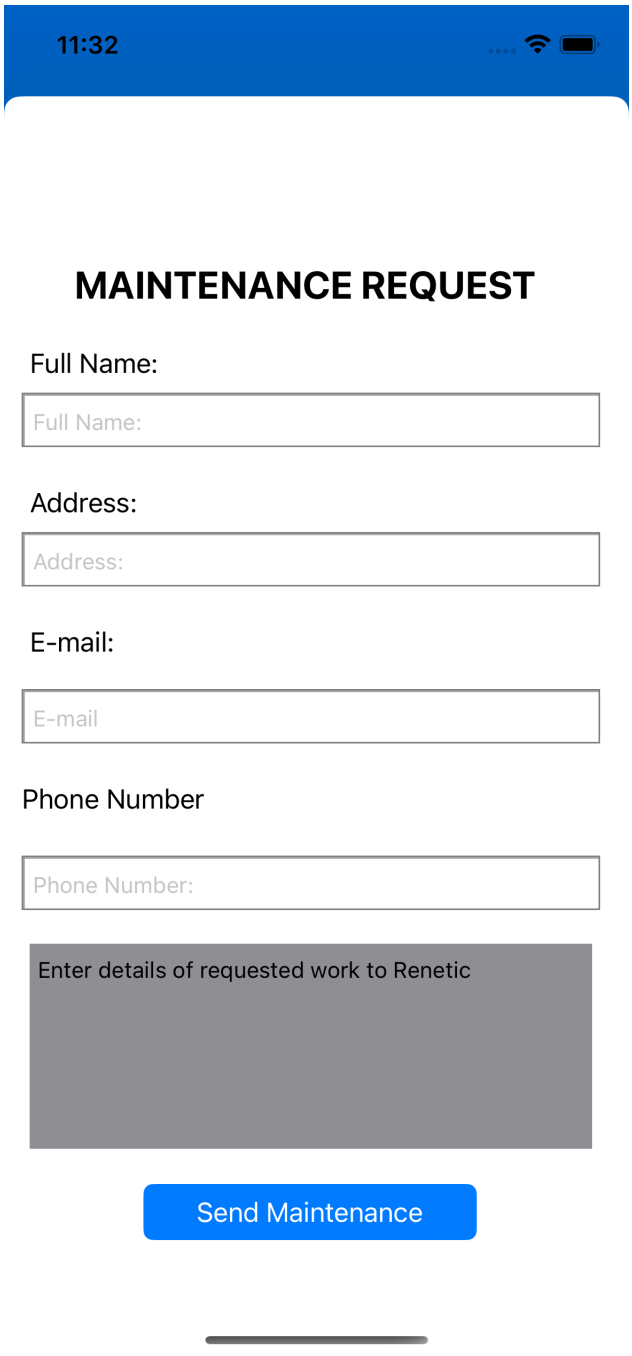

Figure 7.6: Landlord Maintenace

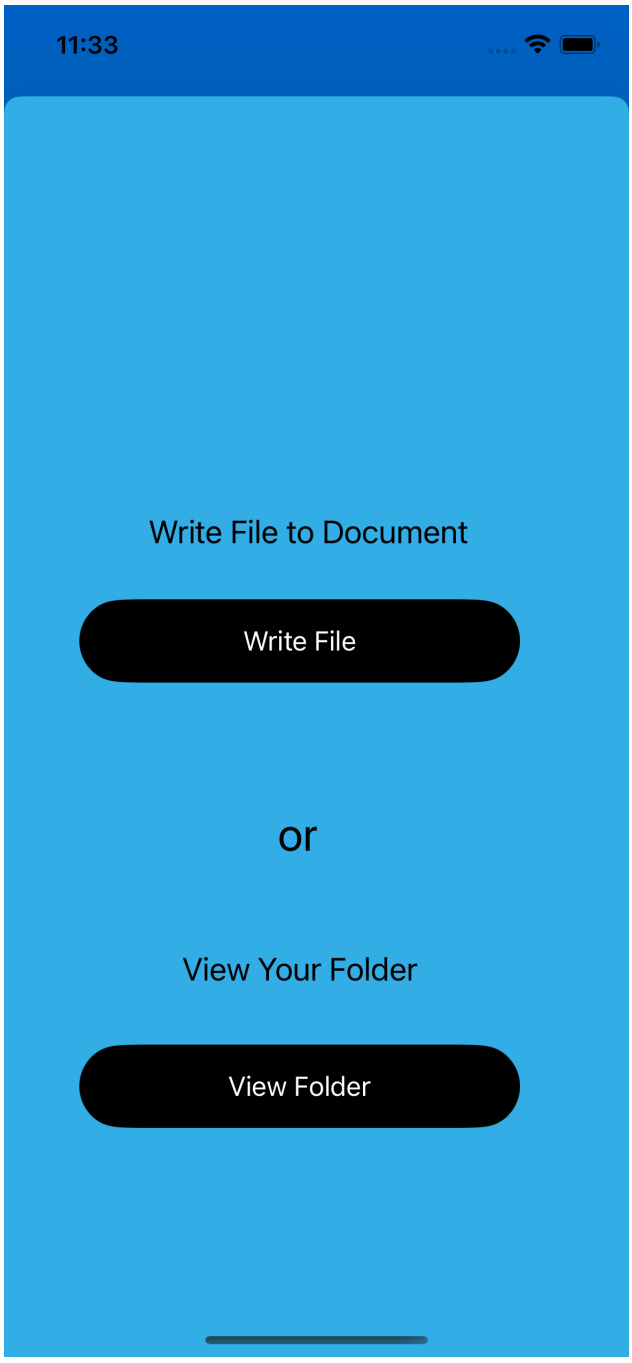

Figure 7.7: Landlord View Documents

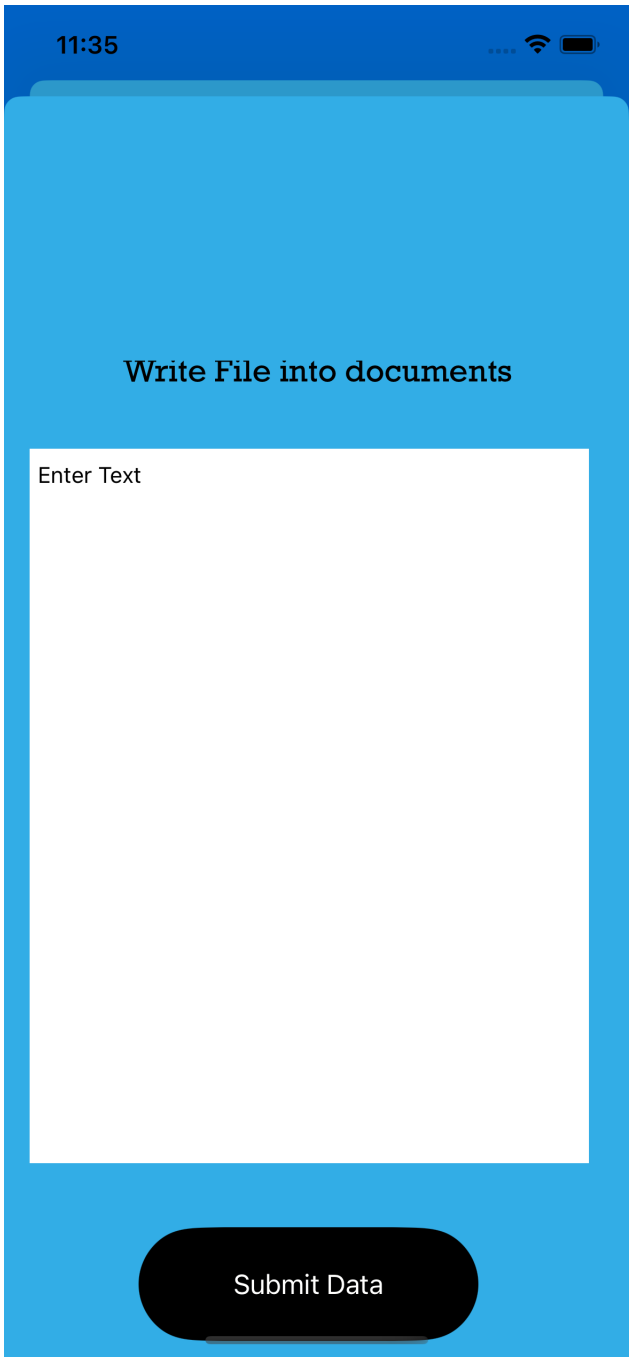

Figure 7.8: Landlord Write File

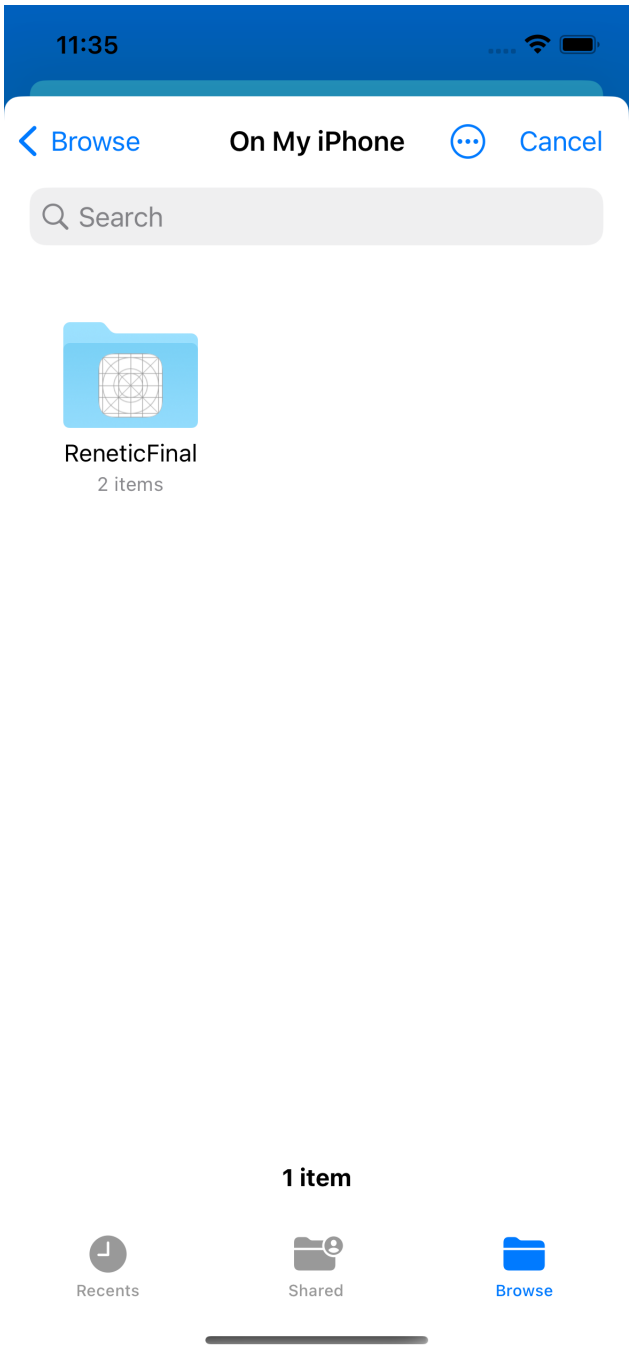

Figure 7.9: Landlord View Folder

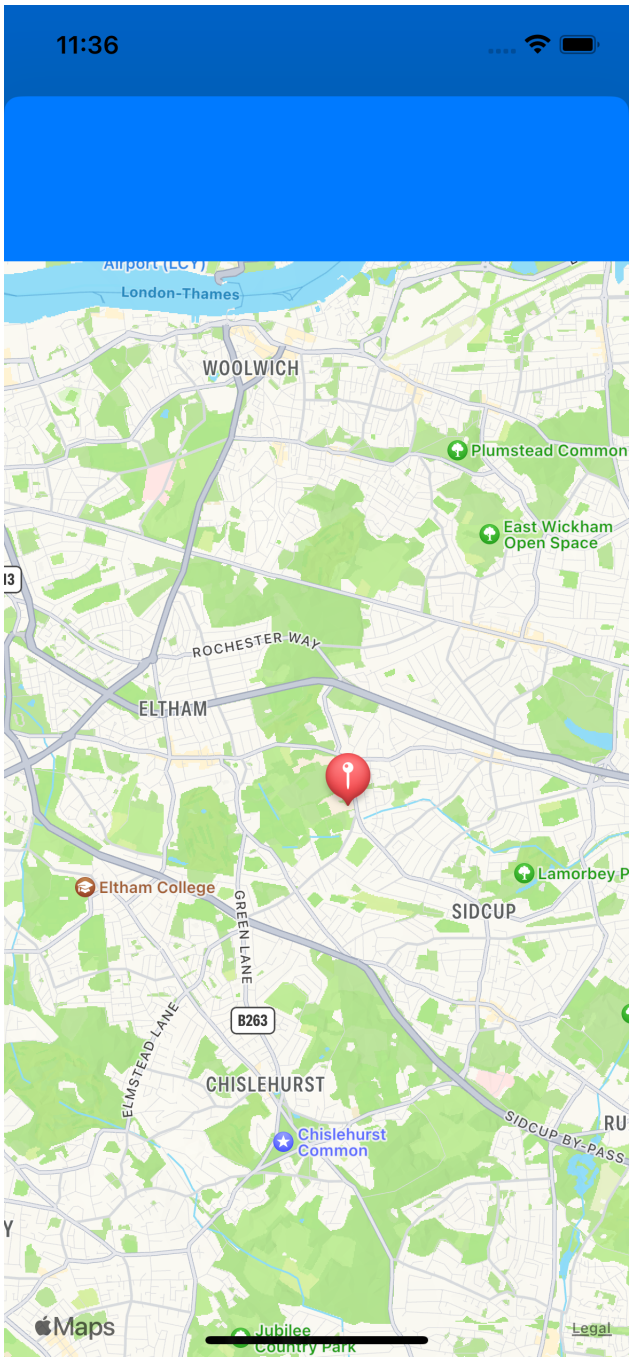

Figure 7.10: Landlord Properties

Tenant Application:

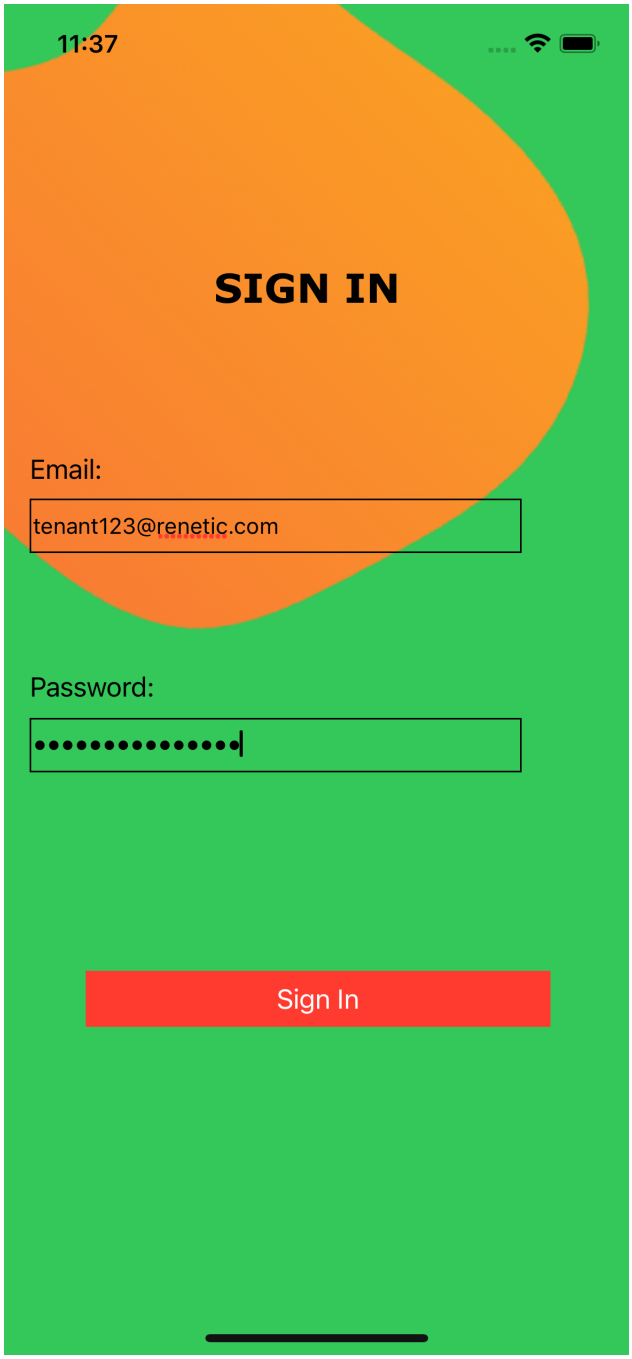

Figure 7.11: Tenant Sign In

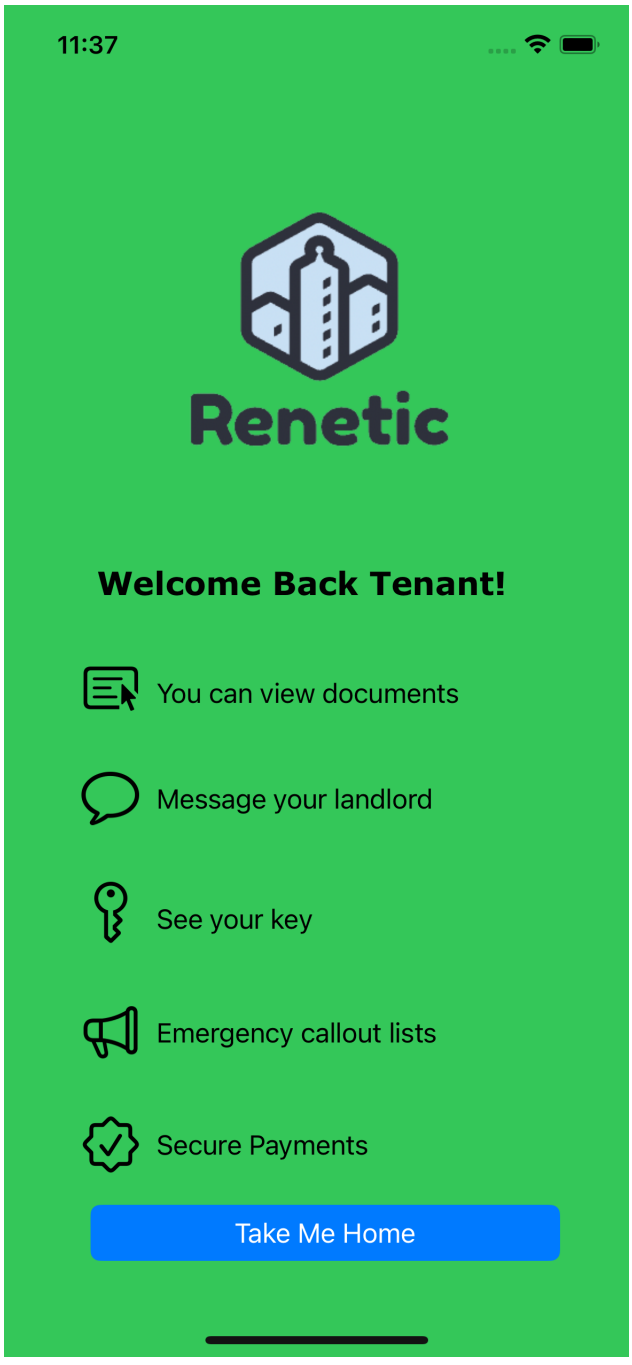

Figure 7.12: Tenant Welcome Page

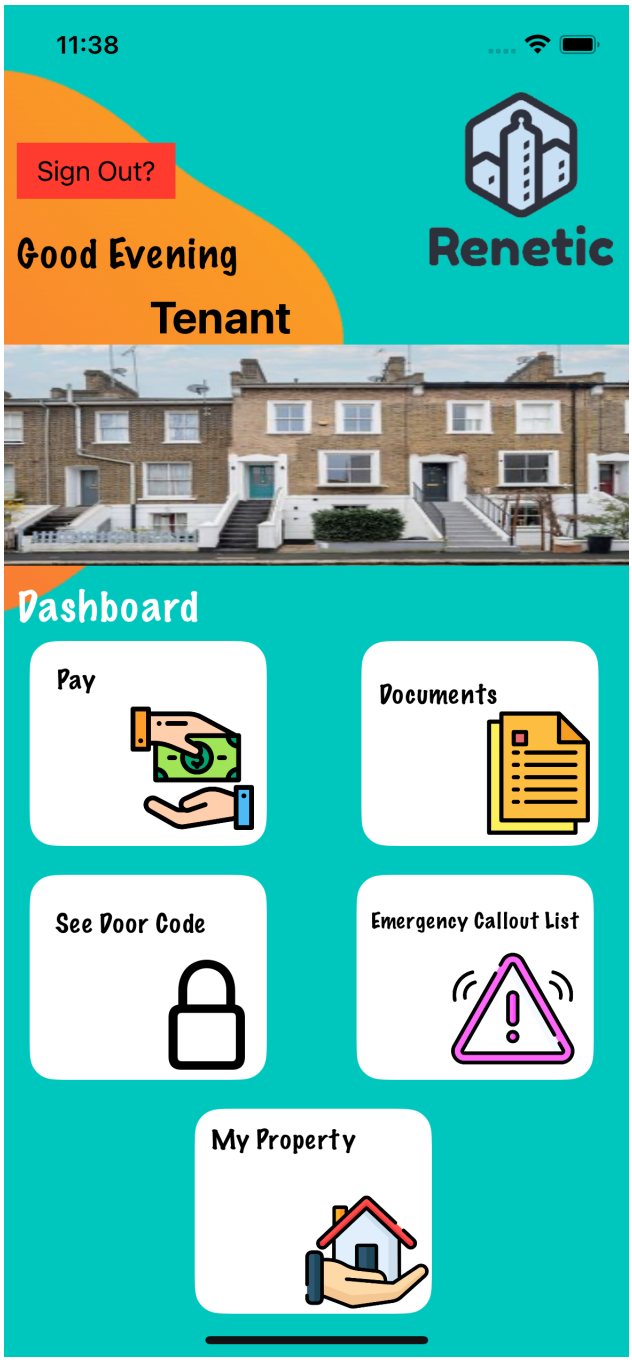

Figure 7.13: Tenant Homepage

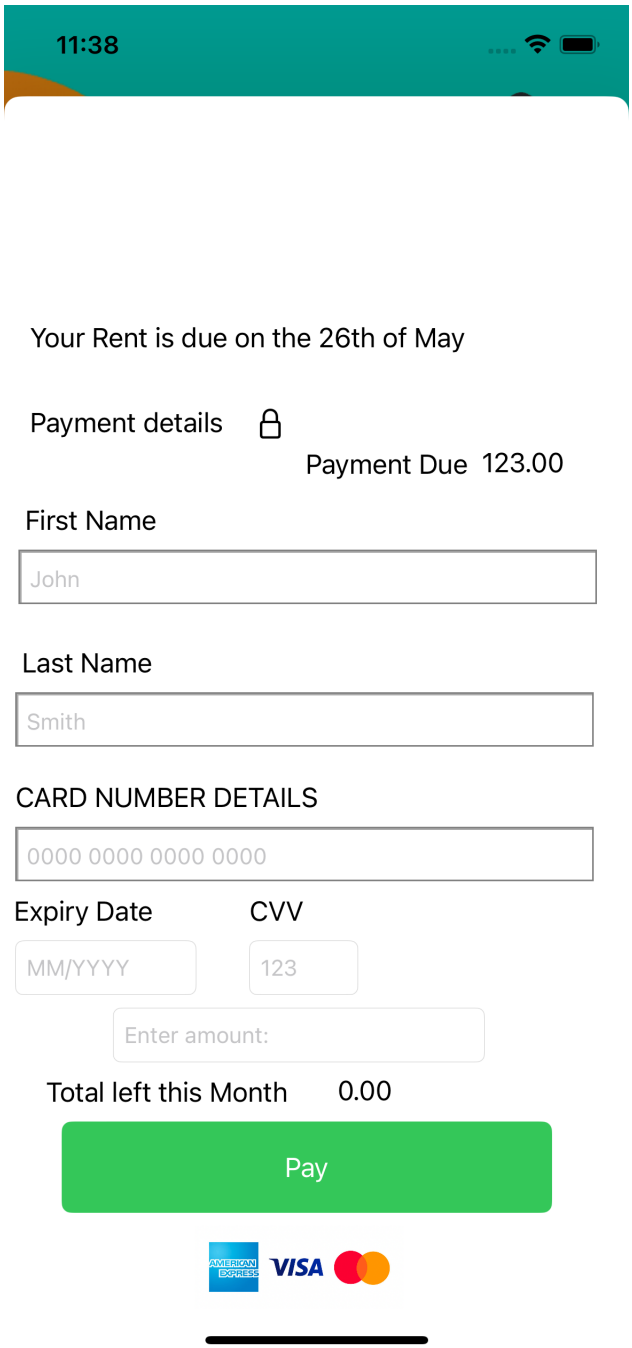

Figure 7.14: Tenant Payment Page

Figure 7.15: Tenant View Folder

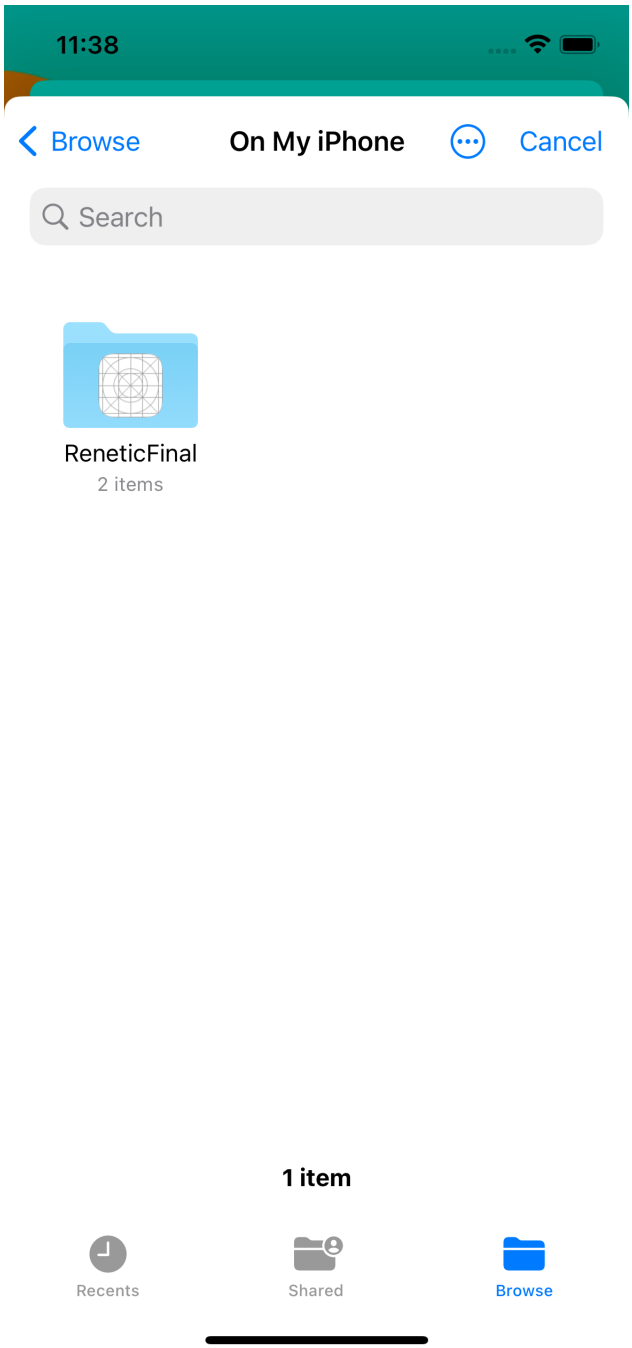

Figure 7.16: Tenant iOS Folders

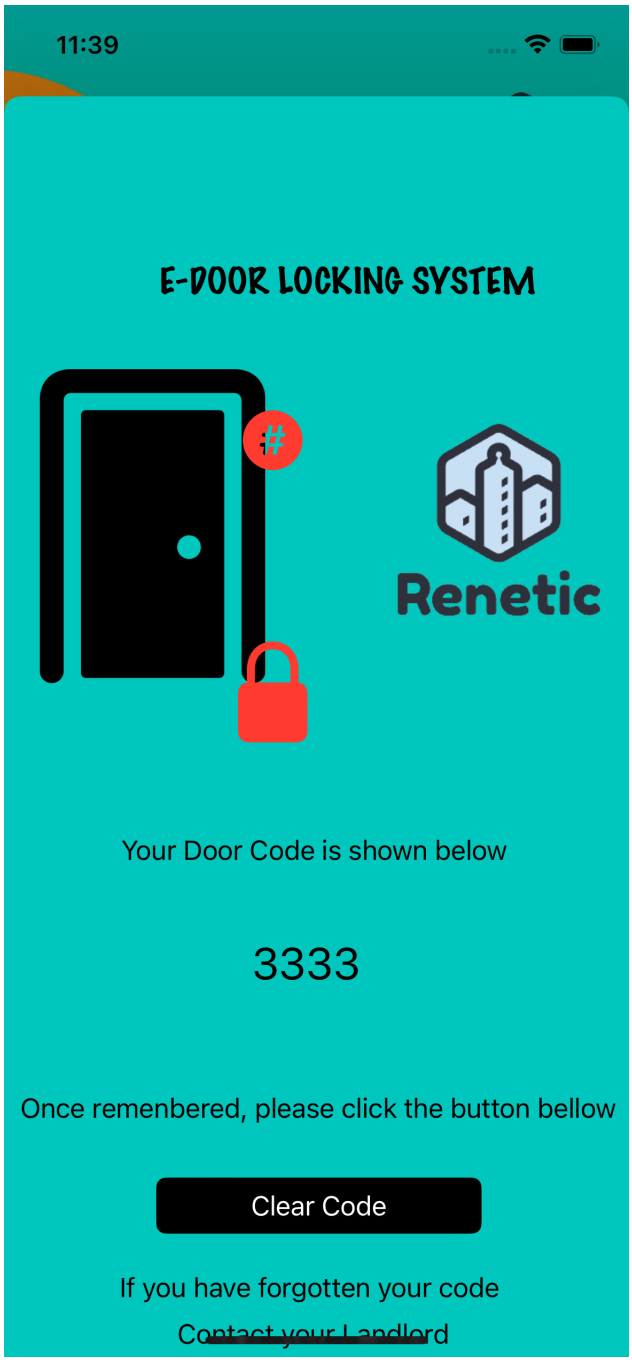

Figure 7.17: Tenant Door Lock Code

Figure 7.18: Tenant Emergency Callout Numbers

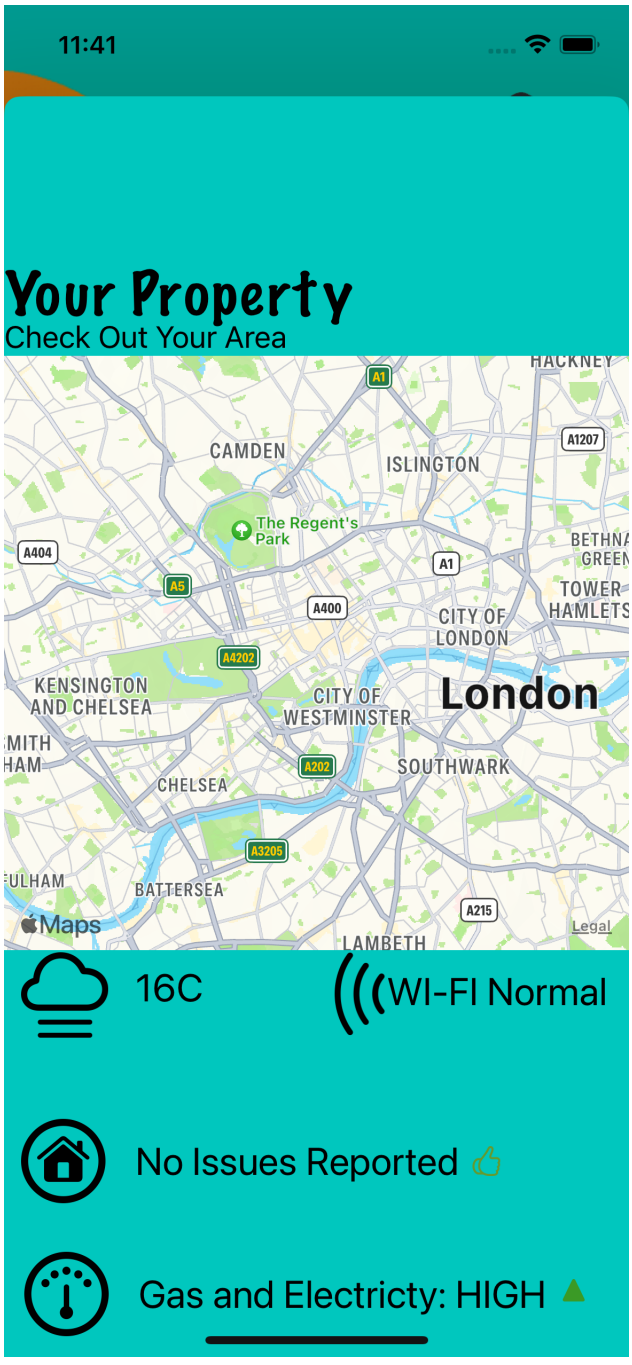

Figure 7.19: Tenant Your Property

Renetic Chat

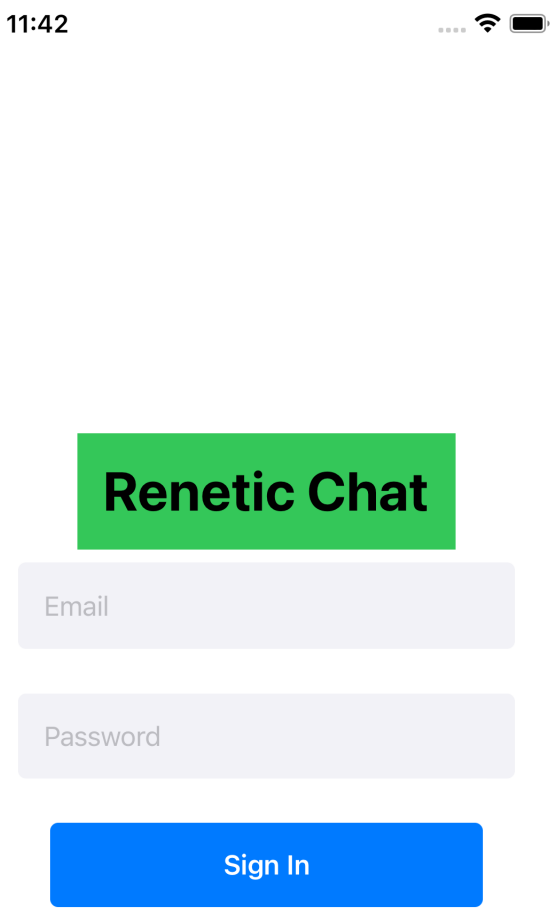

Figure 7.20: Renetic Chat Login

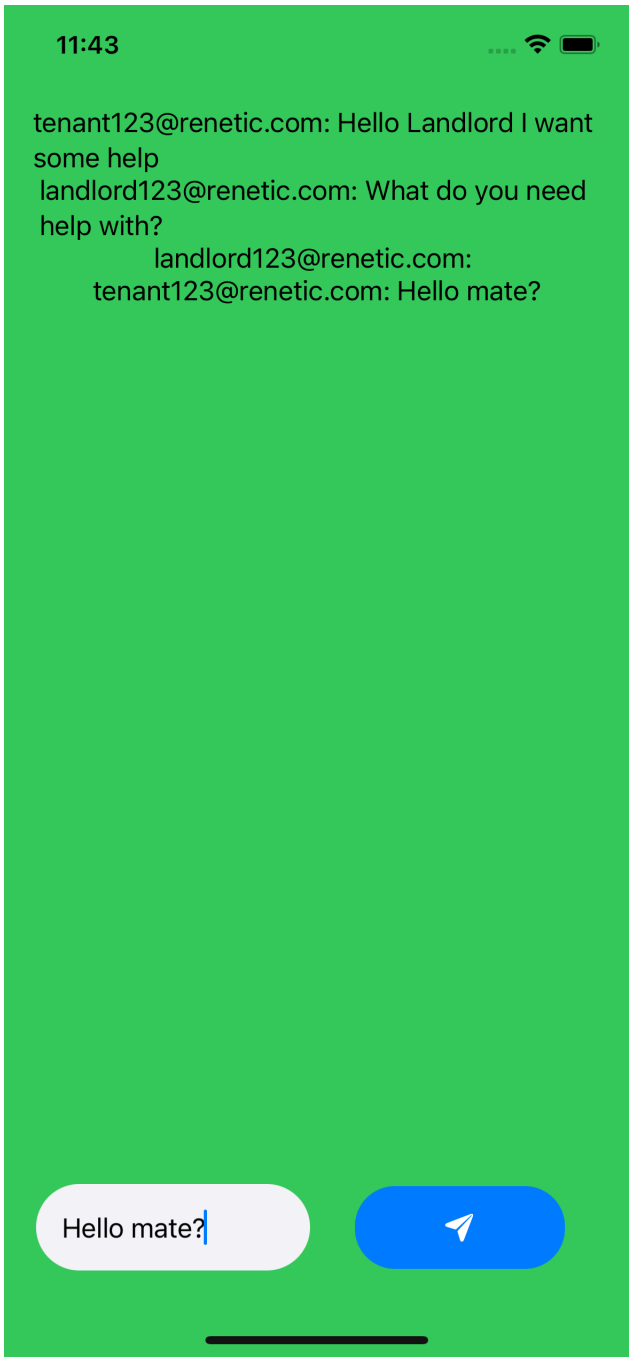

Figure 7.21: Renetic Chat Feature

Door Code

```
Gathering Code from Renetic Database
Enter Your Code: 2222
Code unsuccessfull: Try again
>
```

Figure 7.22: Code unsuccessful

```
Gathering Code from Renetic Database
Enter Your Code: 1111
Code successfull: Welcome Home
>
```

Figure 7.23: Code Successful

Chapter 8

Product Evaluation

Usability, Features and User-Experience: The usability and user experience of the application is one of the main aspects when it came to developing the Renetic application. We focused on making the user interface (UI) easy and informative, which helps users navigate throughout the app. To see if the application was easy to use, we let 40-60 year olds try our application. They reported to us that the application provided all the features needed to manage a property and was easy to navigate using the iOS ecosystem (iPhone devices). However, they commented that additional features such as "Allowing the landlord to add more clients" into the app and "Providing welfare checks on tenants" would make the Renetic app an even better application compared to other CRM apps. We also could improve the application by expanding to more users by developing the application for Android users as well. With the door locking system that we implemented with the application, we are able to use this system quite easily because of its simple design, we can see that the LCD display gives us easy instructions to follow for opening the door for tenants, which improves the usability and user experience for tenants.

Performance and Technical Stability The performance of the application seems to load more quicker, is responsive and can handle a large amount of data within the app. The Firebase website seems to be configured correctly with the Renetic app because we can send huge amounts of data without Firebase even crashing. The application still keeps a consistent theme when users use different devices in apples iOS ecosystem(iPhone 11, 12, XR and etc) Lastly, we can further improve the performance of the app by regularly monitoring the app using Firebase's

analytics dashboard and giving questionnaires to users.

Security: Renetic provides good security features to keep unwanted visitors from using the app. Using Firebases API database and real-time storage instead of the old SQL database was a good implementation because as mentioned before it uses JSON files to store data and can't be hacked using the SQL injection attack. we also benefited from the encryption keys that Firebase has given us.

Business: The Renetic app could start a business in the real estate market. The application was designed to be a middleman between the landlord and the tenant. (Informing the landlords about occurring problems from tenants that they have missed or helping landlords with their properties) If it was to be a business, we encourage future investors, to hire software engineers and customer services because it will improve customer relationship, maintain the application(Software Development Life Cycle) and keep up with new trends in mobile development. Overall the development of Renetic depends on the success of its usability, user experience and performance to beat other CRM applications in the real estate market.

Chapter 9

Conclusion

The development of Renetic which is aimed at landlords and tenants will present a promising solution to the challenges that are faced by both parties in the real estate industry. This app meets the requirements needed to solve this by implementing real-time communication via text messages between landlords and tenants, which helps improve the relationship between each other, providing a way for tenants to pay their rent to landlords on time, write and view important documents which landlords and tenants can both access via the app, view their properties with an additional API of Apple Maps that tells landlords where all their properties are, while tenants can see their local area and lastly both landlords and tenants can both send out maintenance request in the application, reducing the need to visit the landlord's office for tenants which save the landlord and tenants time. One other key benefit of this app that makes it stand out from all CRM platforms in the real estate industry is the introduction of an IoT door lock that utilises Internet of Things technology, The implementation of this feature provides numerous ways to help both landlords and tenants. (If landlords have new tenants moving in into their property, they're able to change the door code which not only saves them money but provides security while if the tenant forgot their key, they're able to see a code within the app that allows them into the property). We have also provided the performance, security and privacy aspects within our application and designed the application, helping us create a user-friendly user interface and help with the development phase of the project.

The limitations that we faced when creating the project was trying to

implement the SwiftUI chat with the Storyboard Files in our application. If we had got this to work in our application, our users wouldn't have to go on a different application to communicate with each other but instead use the same application. We could have also provided more testing in the application such as usability testing and implementing more data security rules to protect the data in our database. Also, we encountered problems with integrating Firebase into the Swift project where the packages which we imported would sometimes disappear entirely from the project and when data would not be submitted to our database, these problems affected the development process and wasted time on fixing these problems.

Potential future work of our work would be adding additional features to our application, one where we provide a more in-depth page for the landlord's financial page, integrating facial recognition for the door locking system which provides more security than a normal keypad passcode entry, introducing a predictive analysis for rental pricing and the current market trends, use blockchain technology to create a digital currency within the application on which tenants can use to pay their rent, shifting the application towards an aggregator business model like AirBnb where users can short rental a property in the app, this would attract more users into the application, creating a group chat feature within the chat app for Renetic that allows users to communicate with their neighbours and lastly another potential feature we could implement would be to allow users to search, view and buy properties using the app, similar to what Rightmove and Zoopla have that allows users to do these exact features. In conclusion, this report has documented the history of the real estate market, benefits, design, and technologies used when creating Renetic. The creation of Renetic seems to be a good idea because as the demand for rental properties continues to grow in the 21st century, the development of the Renetic app for landlords and tenants is an area of great potential for the future of the real estate market.

References

Anwar, S. & Kishore, D. (2016), 'Iot based smart home security system with alert and door access control using smart phone', *International Journal of Engineering Research & Technology (IJERT)* **5**(12), 504–509.

A.s, F., Makinde, B., Adegbola, O., Akin-Olayemi, T., Adeyege, A., Adekunle Adeosun, E. & Akande, B. (2021), 'Design and construction of a smart door lock with an embedded spy-camera', **8**, 2458–9403.

Belkhir, A., Abdellatif, M., Tighilt, R., Moha, N., Guéhéneuc, Y.-G. & Beaudry, (2019), An observational study on the state of rest api uses in android mobile applications, *in* '2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)', pp. 66–75.

Bezemskij, A. (2022), 'Comp1671 penetration testing and ethical vulnerability scanning, vulnerability types', *Greenwich University* pp. 4–9.

Birrell, A. D. & Nelson, B. J. (1984), 'Implementing remote procedure calls', *ACM Transactions on Computer Systems (TOCS)* **2**(1), 39–59.

Brush, A. B., Jung, J., Mahajan, R. & Martinez, F. (2013), Digital neighborhood watch: Investigating the sharing of camera data amongst neighbors, *in* 'Proceedings of the 2013 Conference on Computer Supported Cooperative Work', CSCW '13, Association for Computing Machinery, New York, NY, USA, p. 693–700.
URL: *https://doi.org/10.1145/2441776.2441853*

Byres, T. J. (2009), 'The landlord class, peasant differentiation, class struggle and the transition to capitalism: England, france and prussia compared', *The Journal of Peasant Studies* **36**(1), 33–54.

Cadwalladr, C. & Graham-Harrison, E. (2018), 'Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach', *The guardian* **17**(1), 22.

Calder, B. J., Phillips, L. W. & Tybout, A. M. (1981), 'Designing research for application?'journal of consumer research 8 (september): 197-207.,–, and. 1982', *The Concept of External Validity* pp. 240–244.

Castro, D. & Steinberg, M. (2017), 'Blocked: why some companies restrict data access to reduce competition and how open apis can help', *Available at SSRN 3108763* .

Chakraborty, S., Dziembowski, S. & Nielsen, J. B. (2020), Reverse firewalls for actively secure mpcs, *in* 'Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II', Springer, pp. 732–762.

De Sá, M., Carriço, L., Duarte, L. & Reis, T. (2008), A mixed-fidelity prototyping tool for mobile devices, *in* 'Proceedings of the Working Conference on Advanced Visual Interfaces', AVI '08, Association for Computing Machinery, New York, NY, USA, p. 225–232.
URL: *https://doi.org/10.1145/1385569.1385606*

Ennis, A., Cleland, I., Patterson, T., Nugent, C. D., Cruciani, F., Paggetti, C., Morrison, G. & Taylor, R. (2016), Doorstep: A doorbell security system for the prevention of doorstep crime, *in* '2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)', pp. 5360–5363.

Gay, W. (2018), *Advanced raspberry pi: raspbian linux and GPIO integration*, Apress.

Halfond, W. G., Viegas, J., Orso, A. et al. (2006), A classification of sql-injection attacks and countermeasures, *in* 'Proceedings of the IEEE international symposium on secure software engineering', Vol. 1, IEEE, pp. 13–15.

Henriyan, D., Subiyanti, D. P., Fauzian, R., Anggraini, D., Aziz, M. V. G. & Prihatmanto, A. S. (2016), Design and implementation of web based real time chat interfacing server, *in* '2016 6th International Conference on System Engineering and Technology (ICSET)', IEEE, pp. 83–87.

Ho, G., Leung, D., Mishra, P., Hosseini, A., Song, D. & Wagner, D. (2016), Smart locks: Lessons for securing commodity internet of things devices, *in* 'Proceedings of the 11th ACM on Asia conference on computer and communications security', pp. 461–472.

Idris, M., Syarif, I. & Winarno, I. (2021), Development of vulnerable web application based on owasp api security risks, *in* '2021 International Electronics Symposium (IES)', IEEE, pp. 190–194.

Jain, P., Gyanchandani, M. & Khare, N. (2016), 'Big data privacy: a technological perspective and review', *Journal of Big Data* **3**, 1–25.

Joorabchi, M. E., Mesbah, A. & Kruchten, P. (2013), Real challenges in mobile app development, *in* '2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement', IEEE, pp. 15–24.

Kaur, S. & Kaur, G. (2021), Threat and vulnerability analysis of cloud platform: a user perspective, *in* '2021 8th International Conference on Computing for Sustainable Global Development (INDIACom)', IEEE, pp. 533–539.

Kelly, M. & Nozzi, J. (2013), *Mastering Xcode: Develop and Design*, Peachpit Press.

La Vigne, N. G., Lowry, S. S., Markman, J. A. & Dwyer, A. M. (2011), 'Evaluating the use of public surveillance cameras for crime control and prevention', *Washington, DC: US Department of Justice, Office of Community Oriented Policing Services. Urban Institute, Justice Policy Center* pp. 1–152.

Lastdrager, E., Hesselman, C., Jansen, J. & Davids, M. (2020), Protecting home networks from insecure iot devices, *in* 'NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium', pp. 1–6.

Leau, Y. B., Loo, W. K., Tham, W. Y. & Tan, S. F. (2012), Software development life cycle agile vs traditional approaches, *in* 'International Conference on Information and Network Technology', Vol. 37, pp. 162–167.

Mirante, D. & Cappos, J. (2013), 'Understanding password database compromises', *Dept. of Computer Science and Engineering Polytechnic Inst. of NYU, Tech. Rep. TR-CSE-2013-02* .

Muccini, H., Di Francesco, A. & Esposito, P. (2012), Software testing of mobile applications: Challenges and future research directions, *in* '2012 7th International Workshop on Automation of Software Test (AST)', IEEE, pp. 29–35.

Park, Y. T., Sthapit, P. & Pyun, J.-Y. (2009), Smart digital door lock for the home automation, *in* 'TENCON 2009-2009 IEEE Region 10 Conference', IEEE, pp. 1–6.

Rasila, H. (2010), 'Customer relationship quality in landlord-tenant relationship', *Property Management* **28**(2), 80–92.

Rebouças, M., Pinto, G., Ebert, F., Torres, W., Serebrenik, A. & Castor, F. (n.d.), An empirical study on the usage of the swift programming language, *in* '2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)', Vol. 1.

Sarkar, A. & Singh, B. K. (2020), 'A review on performance, security and various biometric template protection schemes for biometric authentication systems', *Multimedia Tools and Applications* **79**, 27721–27776.

Small, G. W. (2002), 'What we need to know about age related memory loss', *Bmj* **324**(7352), 1502–1505.

Sowjanya, G. & Nagaraju, S. (2016), Design and implementation of door access control and security system based on iot, *in* '2016 International Conference on Inventive Computation Technologies (ICICT)', Vol. 2, pp. 1–4.

Tanna, M. & Singh, H. (2018), *Serverless Web Applications with React and Firebase: Develop real-time applications for web and mobile platforms*, Packt Publishing Ltd.

Tomic, I. (2017), 'Comp1829 network security lecture 5: Part 2/2 cyber physical attacks 2', *Greenwich University* pp. 4–9.

Vaughan, T. R. (1968), 'The landlord-tenant relation in a low-income area', *Social Problems* **16**(2), 208–218.

Wheeler, H. (2019), 'Landlord and tenant rights and responsibilities in the private rented sector'.

URL: *https://www.gov.uk/government/publications/landlord-and-tenant-rights-and-responsibilities-in-the-private-rented-sector/landlord-and-tenant-rights-and-responsibilities-in-the-private-rented-sector*

Wilkes, M. V., Wheeler, D. J., Gill, S. & Corbató, F. (1958), 'The preparation of programs for an electronic digital computer', *Physics Today* **11**(8), 28.

Chapter 10

Appendices

10.0.1 Password and use for Renetic

Users Email and Passwords for Renetic: To log into the application, the email for the landlord is (Email: landlord123@renetic.com) and the password is (Password: PasswordTest123). With the tenant, the email is (tenant123@renetic.com) and the password is (Password: PasswordTest1234) This will be the same for the user when logging in to the chat app. Note the password is cap sensitive. You can now freely browse the Renetic application as you please.(Note 2: When submitting data in this app, you must complete all fields required to send to the database)

Renetic Door Lock

- Turn on the door lock
- Run Python Script with the door lock system
- Ask the landlord for a code using the Renetic Chat
- Retrieve this code within the application
- Enter the code correctly into the system
- The system should indicate to you that the code was correct

Renetic Chat

- Log in to the system using the same email and password as the main application
- Click the sign-in button to enter the Renetic Chat
- If you want to send a message to another user, type into the textbox at the bottom left
- To send the message, click the blue button with the paper aeroplane to send

- To read the messages, the chat logs will be displayed at the top of the screen

10.0.2 Swift Code

```
//
//  ReneticMenuView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 01/04/2023.
//

import UIKit

class ReneticMenuView: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }


    @IBAction func createAccountBtn(_ sender: Any) {
        // This button will take the user to the create Account Page

    }



    @IBAction func logInBtn(_ sender: Any) {

        // This button will take the user to the login page
    }
}


//
//  RequestAccountView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 02/04/2023.
//

import UIKit
import Firebase


class RequestAccountView: UIViewController {
```

```
@IBOutlet weak var nameTextField: UITextField!
@IBOutlet weak var choiceTextField: UITextField!
@IBOutlet weak var emailTextField: UITextField!

override func viewDidLoad() {
 super.viewDidLoad()


 // Do any additional setup after loading the view.
    }


@IBAction func requestAccount(_ sender: Any) {
guard let text1 = nameTextField.text,!text1.isEmpty,
 let text2 = choiceTextField.text, !text2.isEmpty,
let text3 = emailTextField.text, !text3.isEmpty
 // Add additional guard statements for any additional text fields
else {
let alertController =UIAlertController(title: "Error", message:"
Please enter information in all text boxes", preferredStyle: .alert)
alertController.addAction(UIAlertAction(title: "OK", style: .default, handler: nil))
present(alertController, animated: true, completion: nil)
            return
        }

//If user does not enter any text into the textfield,
it will display an UIerror message to the user



guard let name = nameTextField.text,
let email = emailTextField.text,
let choice = choiceTextField.text else{
            return
        }

let db = Firestore.firestore()
let data = ["name": name, "email": email, "choice": choice]
db.collection("account_request").addDocument(data: data){error in
if let error = error {
print(error.localizedDescription)
} else{
 // This Code will save the users data into the Firebase Database,
 so that people at renetic can review their request and set them a account
            }
        }

    }
}
```

```
//
//  SignInView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 03/04/2023.
//

import UIKit
import Firebase
import FirebaseAuth

class SignInView: UIViewController {

@IBOutlet weak var emailTextField: UITextField!
@IBOutlet weak var passwordTextField: UITextField!
override func viewDidLoad() {
    super.viewDidLoad()

        // Do any additional setup after loading the view.
    }


@IBAction func signIn(_ sender: Any) {
guard let email = emailTextField.text,
let password = passwordTextField.text else {
            return
        }

        Auth.auth().signIn(withEmail: email, password: password) { [weak self]
        authResult, error in
            guard let strongSelf = self else {
                return
            }

            if let error = error {
                let alertController = UIAlertController(title: "Error",
                message: error.localizedDescription, preferredStyle: .alert)
                alertController.addAction(UIAlertAction(title: "OK", style: .default,
                handler: nil))
                strongSelf.present(alertController, animated: true, completion: nil)
                return
            }
            // When the user sign in with their account

            if let email = authResult?.user.email {
                if email.contains("landlord123@renetic.com") {
                    let storyboard = UIStoryboard(name: "Main", bundle: nil)
                    let secondController = storyboard.instantiateViewController
                    (withIdentifier: "LandlordHome")
                    self?.present(secondController, animated: true, completion: nil)
```

```
                } else {

                    let storyboard = UIStoryboard(name: "Main", bundle: nil)
                    let viewController = storyboard.instantiateViewController
                    (withIdentifier: "tenantVC")
                    self?.present
                    (viewController, animated: true, completion: nil)
                }
            }
        }
    }
}


    //
//  LandlordFView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 06/04/2023.
//

import UIKit

class LandlordFView: UIViewController {

@IBOutlet weak var meetingMsg: UILabel!

override func viewDidLoad() {
        super.viewDidLoad()
        }

@IBAction func takeMeHomeBtn
(_ sender: Any) {
        performSegue(
        withIdentifier: "goToLandlordStoryboard", sender: self)
    }

   @IBAction func SignOut(_ sender: Any) {
        let alertController = UIAlertController
        (title: "Renetic Sign Out", message: "Are you sure you want to sign out?", prefer
        .alert)

            let cancelAction =
            UIAlertAction(title: "Cancel", style: .cancel, handler: nil)
            alertController.addAction(cancelAction)
            //Sign Out button When user clicks, Give an UIAlert to User
            let signOutAction = UIAlertAction
            (title: "Sign Out", style: .destructive) { _ in
                // Navigate to the new storyboard
```

```
                let storyboard = UIStoryboard(name: "Main", bundle: nil)
                let viewController = storyboard.instantiateViewController(withIdentifier:
                UIApplication.shared.windows.first?.rootViewController = viewController
                UIApplication.shared.windows.first?.makeKeyAndVisible()
                // This Page goes to the first storyboard
            }
            alertController.addAction(signOutAction)

            present(alertController, animated: true, completion: nil)
    }

}

    //
//  TenantFView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 07/04/2023.
//

import UIKit

class TenantFView: UIViewController {

    @IBOutlet weak var landlordLbl: UILabel!
    @IBOutlet weak var paymentLbl: UILabel!
    @IBOutlet weak var keyLbl: UILabel!
    @IBOutlet weak var docLbl: UILabel!
    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }



    @IBAction func takeMeHomeF(_ sender: Any) {
        performSegue(withIdentifier: "goToTenantStoryboard", sender: self)
    }


}

    //
//  SetCodeTView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 08/04/2023.
//
```

```
import UIKit
import Firebase


class SetCodeTView: UIViewController {

    @IBOutlet weak var CodeTextField: UITextField!
    @IBOutlet weak var TitleLabel: UILabel!
    @IBOutlet weak var labelInstruction: UILabel!
    @IBOutlet weak var lockImage: UIImageView!
    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }


    @IBAction func setCodeBtn(_ sender: Any) {
        guard let text1 = CodeTextField.text, !text1.isEmpty
        else {
            let alertController = UIAlertController(title: "Error", message:
            "No Code has been entered", preferredStyle: .alert)
            alertController.addAction(UIAlertAction
            (title: "OK", style: .default, handler: nil))
            present(alertController, animated: true, completion: nil)
            return
        }
        guard let code = CodeTextField.text else{
            return
        }
        let db = Firestore.firestore()
        let data =  ["Code":code]
        db.collection("DCodes").addDocument(data: data){error in
            if let error = error{
                print(error.localizedDescription)
            } else{
                //Store Door Code into my Firebase Firestore Database
            }
        }
    }

}

    //
//  WriteFileLView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 11/04/2023.
//
```

```
import UIKit
import CoreData


class WriteFileLView: UIViewController {



    @IBOutlet weak var textview: UITextView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }



    @IBAction func submitFileBtn(_ sender: Any) {
        let file = "\(UUID().uuidString).txt"
        let contents = textview.text!
        //Write Files into the Folder when button is Clicked
        let dir = FileManager.default.urls(for:
        .documentDirectory, in: .userDomainMask).first!
        let fileUrl = dir.appendingPathComponent(file)

        do{
            try contents.write(to: fileUrl, atomically: false, encoding: .utf8)
        }
        catch{
            print("Error:\(error)")
        }
    }

}

    //
//  BtnSubmitDataView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 11/04/2023.
//

import UIKit

class BtnSubmitDataView: UIViewController {


    @IBOutlet weak var textView: UITextView!
```

```
    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }


    @IBAction func submitBtn(_ sender: Any) {
        let file = "\(UUID().uuidString).txt"
        let contents = textView.text!

        let dir = FileManager.default.urls
        (for: .documentDirectory, in: .userDomainMask).first!
        let fileUrl = dir.appendingPathComponent(file)

        do{
            try contents.write(to: fileUrl, atomically: false, encoding: .utf8)
        }
        catch{
            print("Error:\(error)")
        }

    }

}

    //
//  DocumentCTView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 11/04/2023.
//

import UIKit
import MobileCoreServices

class DocumentCTView: UIViewController, UIDocumentPickerDelegate{

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }



    @IBAction func importFileBtn(_ sender: Any) {
        let documentPicker =
        UIDocumentPickerViewController(documentTypes: ["public.item"], in: .import)
           documentPicker.delegate = self
```

```
            present(documentPicker, animated: true, completion: nil)
    }
}

    //
//  DocumentViewerTeView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 11/04/2023.
//

import UIKit

class DocumentViewerTeView: UIViewController,  UIDocumentPickerDelegate {

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }



    @IBAction func ViewFolderBtn(_ sender: Any) {
        let documentPicker =
        UIDocumentPickerViewController(documentTypes: ["public.item"], in: .import)
           documentPicker.delegate = self
           present(documentPicker, animated: true, completion: nil)

    }

}

    //
//  MaintenaceView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 11/04/2023.
//

import UIKit
import Firebase

class MaintenaceView: UIViewController {


    @IBOutlet weak var FNameTextField: UITextField!
    @IBOutlet weak var PhoneTextField: UITextField!
    @IBOutlet weak var EmailTextField: UITextField!
    @IBOutlet weak var AddressTextField: UITextField!
```

```
    @IBOutlet weak var DetalsTextView: UITextView!

    override func viewDidLoad() {
        super.viewDidLoad()


    }


    @IBAction func sendMaintenaceBtn(_ sender: Any) {
        guard let text1 = FNameTextField.text, !text1.isEmpty,
              let text2 = PhoneTextField.text, !text2.isEmpty,
              let text3 = EmailTextField.text, !text3.isEmpty,
              let text4 = AddressTextField.text, !text4.isEmpty,
              let text5 = DetalsTextView.text, !text5.isEmpty

        else {
            let alertController =
            UIAlertController(title: "Error",
            message: "Maintenace Request will not be sent to Renetic ",
            preferredStyle: .alert)
            alertController.addAction
            (UIAlertAction(title: "OK", style: .default, handler: nil))
            present(alertController, animated: true, completion: nil)
            return
        }

        guard let name =
        FNameTextField.text, let phone = PhoneTextField.text,
        let email = EmailTextField.text,
        let address = AddressTextField.text, let details = DetalsTextView.text else{
            return
        }
        let db = Firestore.firestore()
        let data =
        ["name": name, "email": email, "Phone": phone, "Address": address,
        "Details": details]
        db.collection("Maintenace_Request").addDocument(data: data){error in
            if let error = error {
                print(error.localizedDescription)
            } else{

            }
        }
    }


}

    //
```

```
//  PayCheckView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 13/04/2023.
//

import UIKit
import Firebase

class PayCheckView: UIViewController {

    @IBOutlet weak var LNameTextField: UITextField!

    @IBOutlet weak var CVVNumberTextField: UITextField!
    @IBOutlet weak var xDate: UITextField!
    @IBOutlet weak var CNumberTextField: UITextField!
    @IBOutlet weak var FNameTextField: UITextField!
    @IBOutlet weak var amountDueLabel: UILabel!
    @IBOutlet weak var amountLeft: UILabel!
    @IBOutlet weak var amountPaidTextField: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()


    }


    @IBAction func calulatePayBtn(_ sender: Any) {
        guard let text1 = FNameTextField.text, !text1.isEmpty,
              let text2 = LNameTextField.text, !text2.isEmpty,
              let text3 = CNumberTextField.text, !text3.isEmpty,
              let text4 = xDate.text, !text4.isEmpty,
              let text5 = CVVNumberTextField.text, !text5.isEmpty,
              let text6 = amountPaidTextField.text, !text6.isEmpty

else {
            let alertController = UIAlertController(
            title: "Payment Declined", message:
            "Has all fields been entered?", preferredStyle: .alert)
            alertController.addAction
            (UIAlertAction(title: "OK", style: .default, handler: nil))
            present(alertController, animated: true, completion: nil)
            return
        }

        if let userInput = amountPaidTextField.text,
           let userInputAsNumber = Double(userInput),
           let numberAsNumber = Double(amountDueLabel.text ?? "0"){
```

```
            let result = numberAsNumber - userInputAsNumber


            //Show the user how much is left when calculated




            let alert = UIAlertController
            (title: "Payment Successful", message:
            "The difference is $\(result)", preferredStyle: .alert)
            alert.addAction(UIAlertAction
            (title: "OK", style: .default, handler: nil))
            present(alert, animated: true, completion: nil)

            amountLeft.text = "\(result)"

            guard let text = amountPaidTextField.text else {return}
            let ref = Database.database().reference()
            let textreF = ref.child("Paid")
            textreF.setValue(text)
            //This records the value of what was
            entered from the amount paid
            textfield and save it in our realtime database Firebase

        }



    }
}

    //
//  TenantHomeViewViewController.swift
//  ReneticFinal
//
//  Created by Luke Austin on 13/04/2023.
//

import UIKit

class TenantHomeViewViewController: UIViewController {

    @IBOutlet weak var GreetingMsgLabel: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()

        let date = Date()
```

```
                let calendar = Calendar.current
                let hour = calendar.component(.hour, from: date)

                if hour < 12 {
                    GreetingMsgLabel.text = "Good Morning"
                } else {
                    GreetingMsgLabel.text = "Good Evening"
                }
    }

    // Depening on the time of day, it should display a greeting message to the user



    @IBAction func signOutBtn(_ sender: Any) {
        let alertController = UIAlertController
        (title: "Renetic Sign Out", message: "Are you sure you want to sign out?",
        preferredStyle: .alert)

            let cancelAction =
            UIAlertAction(title: "Cancel", style: .cancel, handler: nil)
            alertController.addAction(cancelAction)

            let signOutAction = UIAlertAction
            (title: "Sign Out", style: .destructive) { _ in
                // Navigate to the new storyboard
                let storyboard =
                UIStoryboard(name: "Main", bundle: nil)
                let viewController = storyboard.instantiateViewController
                (withIdentifier: "signIn")
                UIApplication.shared.windows.first?.rootViewController = viewController
                UIApplication.shared.windows.first?.makeKeyAndVisible()
                // This Page goes to the first storyboard
            }
            alertController.addAction(signOutAction)

            present(alertController, animated: true, completion: nil)

        //This Code will take the user back to the Sign In page in the "Main" storyboard
    }

}

    //
//  FinanceLView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 13/04/2023.
//
```

```
import UIKit
import Firebase

class FinanceLView: UIViewController {

    @IBOutlet weak var dateLabel: UILabel!
    @IBOutlet weak var price: UILabel!
    override func viewDidLoad() {
        super.viewDidLoad()

        let dateFormatx = DateFormatter()
        dateFormatx.dateFormat = "dd-MM-yyyy"

        let currentDate = Date()
        let dateString = dateFormatx.string(from: currentDate)

        dateLabel.text = dateString

        let ref = Database.database().reference()
        let textRef = ref.child("Paid")
        textRef.observe(.value) { snapshot in
            if let text = snapshot.value as? String {
                self.price.text = text
            }
            //Get reference from Database and display it in the label
        }


    }



}

    //
//  PropertyTenantView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 15/04/2023.
//
import CoreLocation
import MapKit
import UIKit

class PropertyTenantView: UIViewController, CLLocationManagerDelegate {


    @IBOutlet var mapView: MKMapView!
    @IBOutlet weak var tempertureLabel: UILabel!
```

```
    let managerT = CLLocationManager()

    override func viewDidLoad() {
        super.viewDidLoad()

    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        managerT.desiredAccuracy = kCLLocationAccuracyBest //
        The more accurate the location is the more battery is used on the users device
        managerT.delegate = self
        managerT.requestWhenInUseAuthorization()
        managerT.startUpdatingLocation()
    }


    func locationManager
    (_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
        if let location = locations.first{
            manager.stopUpdatingLocation()

            render(location)
        }
        func render(_ location: CLLocation) {

            let coordinate =
            CLLocationCoordinate2D
            (latitude:
            location.coordinate.latitude, longitude: location.coordinate.longitude)

            let span =
            MKCoordinateSpan(latitudeDelta: 0.1, longitudeDelta: 0.1)
            // Birds eye view of map

            let region = MKCoordinateRegion(center: coordinate, span: span)

            mapView.setRegion(region,
                              animated: true)
        }


    }




}

    //
```

```
//  PropertiesTView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 15/04/2023.
//
import CoreLocation
import MapKit
import UIKit


class PropertiesTView: UIViewController, CLLocationManagerDelegate, MKMapViewDelegate {


    @IBOutlet weak var mapView3: MKMapView!

    let managerT = CLLocationManager()

    override func viewDidLoad() {
        super.viewDidLoad()

    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        managerT.desiredAccuracy = kCLLocationAccuracyBest
        //
        The more accurate the location is the more battery is used on the users device
        managerT.delegate = self
        managerT.requestWhenInUseAuthorization()
        managerT.startUpdatingLocation()
    }


    func locationManager(_ manager:
    CLLocationManager,
    didUpdateLocations locations: [CLLocation]) {
        if let location = locations.first{
            manager.stopUpdatingLocation()

            render(location)
        }
        func render(_ location: CLLocation) {

            let coordinate =
            CLLocationCoordinate2D
            (latitude:
            location.coordinate.latitude, longitude: location.coordinate.longitude)

            let span =
            MKCoordinateSpan(latitudeDelta: 0.1, longitudeDelta: 0.1)
```

```
            // Birds eye view of map

            let region = MKCoordinateRegion(center: coordinate, span: span)

            mapView3.setRegion(region,
                               animated: true)

            let pin = MKPointAnnotation()
            pin.coordinate = coordinate
            mapView3.addAnnotation(pin)

            let pin2 = MKPointAnnotation()
            pin2.title = "Greenwich University"
            pin.coordinate = CLLocationCoordinate2D
            (latitude: 51.481846, longitude: -0.006199)
            mapView3.addAnnotation(pin2)

            let pin3 = MKPointAnnotation()
            pin3.title = "Avery Hill"
            pin.coordinate = CLLocationCoordinate2D
            (latitude: 51.445990, longitude: 0.080470)
            mapView3.addAnnotation(pin3)

            mapView3.delegate = self
        }


    }




}

    //
//  UrgentRequestTView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 16/04/2023.
//

import UIKit
import Firebase
import MessageUI

class UrgentRequestTView: UIViewController, MFMailComposeViewControllerDelegate{

    override func viewDidLoad() {
        super.viewDidLoad()
```

```
        // Do any additional setup after loading the view.
    }

    var count = 0

    @IBAction func UrgentRequest(_ sender: Any) {
        count += 1
               let db = Firestore.firestore()
               let collection = db.collection("requests")
               collection.addDocument
               (data: ["message": "Request needed \(count)"]) { error in
                   if let error = error {
                       print("Error adding document: \(error)")
                   } else {
                       print("Document added")
                   }
               }// Adding Documents to Documents Database
           }
}
```

```
    //
//  DoorCodeTView.swift
//  ReneticFinal
//
//  Created by Luke Austin on 18/04/2023.
//

import UIKit

class DoorCodeTView: UIViewController {

    @IBOutlet weak var DoorImage: UIImageView!
    @IBOutlet weak var doorCode: UILabel!
    @IBOutlet weak var lockImage: UIImageView!
    @IBOutlet weak var CodeImage: UIImageView!
    override func viewDidLoad() {
        super.viewDidLoad()

        //Label and Button within the storyboard
    }


    @IBAction func clearCodeBtn(_ sender: Any) {
        doorCode.text = ""
    }
```

```
}

    //
//  AppDelegate.swift
//  ReneticFinal
//
//  Created by Luke Austin on 01/04/2023.
//

import UIKit
import Firebase
@main

class AppDelegate: UIResponder, UIApplicationDelegate {


    override init(){
        FirebaseApp.configure()

        func application
        (_ application: UIApplication,
        didFinishLaunchingWithOptions launchOptions:
        [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
            // Override point for customization after application launch.
            return true
        }

        // MARK: UISceneSession Lifecycle

        func application
        (_ application:
        UIApplication, configurationForConnecting connectingSceneSession:
        UISceneSession, options: UIScene.ConnectionOptions) -> UISceneConfiguration {
            // Called when a new scene session is being created.
            // Use this method to select a configuration to create the new scene with.
            return UISceneConfiguration
            (name: "Default Configuration", sessionRole: connectingSceneSession.role)
        }

        func application
        (_ application: UIApplication, d
        idDiscardSceneSessions sceneSessions:
        Set<UISceneSession>) {
            // Called when the user discards a scene session.
            // If any sessions were discarded
            while the application was not running, this
    will be called shortly after application:didFinishLaunchingWithOptions.
```

```
            // Use this method to release any
            resources that were specific to the discarded
    scenes, as they will not return.
        }


    }

}


    //
//  ContentView.swift
//  ReneticChat2
//
//  Created by Luke Austin on 8/04/2023.
//

import SwiftUI
import Firebase


struct ChatApp: App {


    var body: some Scene {
        WindowGroup {
            LoginView()
        }//Display LoginView
    }
}

struct LoginView: View {
    @State private var email = ""
    @State private var password = ""
    @State private var error = ""

    var body: some View {

        VStack(spacing: 10){
            Text("Renetic Chat")
                .font(.largeTitle.bold())
                .padding()
                .background(Color.green)
        }

        VStack {
            TextField("Email", text: $email)
                .padding()
                .background(Color(.systemGray6))
```

```
                .cornerRadius(5.0)
                .padding(.bottom, 20)

            SecureField("Password", text: $password)
                .padding()
                .background(Color(.systemGray6))
                .cornerRadius(5.0)
                .padding(.bottom, 20)

            Button(action: signIn) {
                Text("Sign In")
                    .font(.headline)
                    .foregroundColor(.white)
                    .padding()
                    .frame(maxWidth: .infinity)
                    .background(Color.blue)
                    .cornerRadius(5.0)
            }
            .padding(.horizontal, 20)

            if !error.isEmpty {
                Text(error)
                    .foregroundColor(.red)
                    .padding()
            }
        }
        .padding(.horizontal, 40)
    }

    func signIn() {
        Auth.auth().signIn(withEmail: email, password: password) { result, error in
            if let error = error {
                self.error = error.localizedDescription
            } else {
                UIApplication.shared.windows.first?.rootViewController =
 UIHostingController(
                rootView: ChatView())
                UIApplication.shared.windows.first?.makeKeyAndVisible()
            }
        }
    }
}

struct ChatView: View {
    @State private var message = ""
    @State private var messages: [Message] = []

    var body: some View {// Graphical User Interface for thechat
        VStack {
            ScrollView {
```

```
            ForEach(messages, id: \.self) { message in
                Text("\(message.sender): \(message.text)")
            }
        }
        .padding()

        HStack {
            TextField("Message", text: $message)
                .padding()
                .background(Color(.systemGray6))
                .cornerRadius(30.0)

            Button(action: sendMessage) {
                Image(systemName: "paperplane.fill")
                    .foregroundColor(.white)
                    .padding()
                    .frame(maxWidth: .infinity)
                    .background(Color.blue)
                    .cornerRadius(25.0)
            }
            .padding(.horizontal, 20)
        }
        .padding(.horizontal, 20)
        .padding(.bottom, 20)
    }
    .navigationTitle("Renetic Chat")
    .onAppear(perform: listenForMessages)
    .background(Color.green)
}

func listenForMessages() {//Retreive message from firecloud
    let db = Firestore.firestore()
    db.collection("messages")
        .order(by: "timestamp")
        .addSnapshotListener { snapshot, error in
            if let error = error {
                print("Error getting messages: \(error)")
            } else {
                guard let documents = snapshot?.documents else { return }

                self.messages = documents.map { document -> Message in
                    guard let sender = document.data()["sender"] as? String,
                          let text = document.data()["text"] as? String else {
                        fatalError("Error parsing message data")
                    }

                    return Message(sender: sender, text: text)
                }
            }
        }
```

```
    }

    func sendMessage() {
        guard let user = Auth.auth().currentUser
        else { return }

                let db = Firestore.firestore()
                db.collection("messages").addDocument(data: [
                    "sender": user.email ?? "Anonymous",
                    "text": message,
                    "timestamp": Date().timeIntervalSince1970
                ]) { error in
                    if let error = error {
                        print("Error sending message: \(error)")
                    } else {
                        message = ""
                    }
                }
            // When user sends message, store the message in firestore
               }
           }

           struct Message: Hashable {
               var sender: String
               var text: String
           }

    //
//  ReneticChat2App.swift
//  ReneticChat2
//
//  Created by Luke Austin on 14/04/2023.
//

import SwiftUI
import Firebase

@main
struct ReneticChat2App: App {

    init() {
        FirebaseApp.configure()
    }

    var body: some Scene {
        WindowGroup {
            LoginView()
        }
    }
```

```
}


    # Credit to SriTu Hobby for Helping me with this project
import firebase_admin
from firebase_admin import credentials
from firebase_admin import firestore
import I2C_LCD_driver
import RPi.GPIO as GPIO
from time import sleep

cred = credentials.Certificate('/home/claud/Downloads/reneticfinal-firebase-adminsdk-chw4
firebase_admin.initialize_app(cred)
# We use firebase encryption key to get access to our database

db = firestore.client()

docs_ref = db.collection('DCodes').limit(10)
docs = docs_ref.get()

for doc in docs:
    code = doc.to_dict().get('Code')
    print(f"Code: {code}")
    ##Print the codes from our database

# Enter column pins from GPIO pins
C1 = 5
C2 = 6
C3 = 13
C4 = 19

# Enter row pins
R1 = 12
R2 = 16
R3 = 20
R4 = 21

# Enter buzzer pin
buzzer = 17

# Enter LED pin from Breadboard
Relay = 27
relayState = True

# Create a object for the LCD
lcd = I2C_LCD_driver.lcd()

#Starting text
lcd.lcd_display_string("Renetic Door Security",1,1)
for a in range (0,16):
```

```
    lcd.lcd_display_string(".",2,a)
    sleep(0.1)

lcd.lcd_clear()


keypadPressed = -1

# Enter your PIN
secretCode = "Code" # Gets data from database
input = ""

#GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(buzzer,GPIO.OUT)
GPIO.setup(Relay,GPIO.OUT)
GPIO.output(Relay,GPIO.HIGH)



GPIO.setup(C1, GPIO.OUT)
GPIO.setup(C2, GPIO.OUT)
GPIO.setup(C3, GPIO.OUT)
GPIO.setup(C4, GPIO.OUT)


GPIO.setup(R1, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(R2, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(R3, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(R4, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)



def keypadCallback(channel):
    global keypadPressed
    if keypadPressed == -1:
        keypadPressed = channel


GPIO.add_event_detect(R1, GPIO.RISING, callback=keypadCallback)
GPIO.add_event_detect(R2, GPIO.RISING, callback=keypadCallback)
GPIO.add_event_detect(R3, GPIO.RISING, callback=keypadCallback)
GPIO.add_event_detect(R4, GPIO.RISING, callback=keypadCallback)


def setAllRows(state):
    GPIO.output(C1, state)
    GPIO.output(C2, state)
    GPIO.output(C3, state)
```

```
    GPIO.output(C4, state)


def commands():
    global relayState
    global input
    pressed = False

    GPIO.output(C1, GPIO.HIGH)



    if (GPIO.input(R1) == 1):
        print("Input reset!");
        lcd.lcd_clear()
        lcd.lcd_display_string("Clear",1,5)
        sleep(1)
        pressed = True

    GPIO.output(C1, GPIO.HIGH)


    if (not pressed and GPIO.input(R2) == 1):
        if input == secretCode:
            print("Code correct! Welcome Home")
            lcd.lcd_clear()
            lcd.lcd_display_string("Successful",1,3)

            if relayState:
                GPIO.output(Relay,GPIO.LOW)
                GPIO.output(buzzer,GPIO.HIGH)
                sleep(0.3)
                GPIO.output(buzzer,GPIO.LOW)
                sleep(1)
                relayState = False

            elif relayState == False:
                GPIO.output(Relay,GPIO.HIGH)
                GPIO.output(buzzer,GPIO.HIGH)
                sleep(0.3)
                GPIO.output(buzzer,GPIO.LOW)
                sleep(1)
                relayState = True


        else:
            print("Incorrect code!")
            lcd.lcd_clear()
            lcd.lcd_display_string("Wrong PIN! Try again",1,3)
            GPIO.output(buzzer,GPIO.HIGH)
```

```
            sleep(0.3)
            GPIO.output(buzzer,GPIO.LOW)
            sleep(0.3)
            GPIO.output(buzzer,GPIO.HIGH)
            sleep(0.3)
            GPIO.output(buzzer,GPIO.LOW)
            sleep(0.3)
            GPIO.output(buzzer,GPIO.HIGH)
            sleep(0.3)
            GPIO.output(buzzer,GPIO.LOW)
        pressed = True

    GPIO.output(C1, GPIO.LOW)

    if pressed:
        input = ""

    return pressed


def read(column, characters):
    global input

    GPIO.output(column, GPIO.HIGH)
    if(GPIO.input(R1) == 1):
        input = input + characters[0]
        print(input)
        lcd.lcd_display_string(str(input),2,0)
    if(GPIO.input(R2) == 1):
        input = input + characters[1]
        print(input)
        lcd.lcd_display_string(str(input),2,0)
    if(GPIO.input(R3) == 1):
        input = input + characters[2]
        print(input)
        lcd.lcd_display_string(str(input),2,0)
    if(GPIO.input(R4) == 1):
        input = input + characters[3]
        print(input)
        lcd.lcd_display_string(str(input),2,0)
    GPIO.output(column, GPIO.LOW)

try:
    while True:
        lcd.lcd_display_string("Enter your door code:",1,0)



        if keypadPressed != -1:
            setAllRows(GPIO.HIGH)
```

```
            if GPIO.input(keypadPressed) == 0:
                keypadPressed = -1
            else:
                sleep(0.1)

        else:
            if not commands():
                read(C1, ["D","C","B","A"])
                read(C2, ["#","9","6","3"])
                read(C3, ["0","8","5","2"])
                read(C4, ["*","7","4","1"])
                sleep(0.1)
            else:
                sleep(0.1)
except KeyboardInterrupt:
    print("Stopped Door Feature")
```