Technical Report

BSc in Computing
Software Engineering
Computing Project
2023/2024

Cristina Berlinschi
X20409746@student.ncirl.ie

Table of Contents

Executive Summary

This report outlines the development and implementation of Fusion an innovative business networking application to connect entrepreneurs, business professionals and investors based on their location and business sector. I began this project to address the inefficiencies in traditional networking methods by using technology to bring people together to network.

Within this report I have discussed in detail the requirements of the application, use case diagrams, design, implementation, testing, GUIs and conclusion.

The implementation section goes into detail about the use of Firebase for authentication, ensuring secure and seamless user login, and Firestore for real-time messaging capabilities. The recommendation algorithm, powered by CreateML uses linear regression to analyse user data such as business field and locations.

The geohashing algorithm, converts geographic coordinates into strings. Together, these algorithms ensure that users receive highly relevant and personalized connection recommendations, enhancing the overall effectiveness of Fusion's recommendations.

The results of this project demonstrate Fusion's potential to enhance professional networking by providing a platform to do so. The conclusion highlight's the app's innovative approach to connect. Fusion is determined to make an impact in the networking space and providing valuable connections for its users.

1.0 Introduction

1.1 Background

This project started from a personal passion for business and technology. I observed during my internship and in conversations with mentors and peers, everyone is seeking to connect with like-minded people. Business individuals looking to connect with individuals from the same industry or entrepreneurs are looking for business partners to collaborate. There is power in networking as it can bring numerous benefits that can accelerate business growth, build a supportive network, learn new strategies and many more. This realisation gave inspiration to my app's design, which is based on the idea that technology may be used as an incentive to close the networking gap. The idea is to create a virtual gathering platform for business enthusiasts, investors and entrepreneurs to get together, exchange ideas, and develop mutually beneficial connections that go beyond traditional networking constraints.

This idea began at the end of third year in college. I was aware I had to choose a year-long project and I began brainstorming over summer 2023. I kept a running list of project and app ideas in my notes app constantly adding to it as new thoughts emerged. By September I reviewed and narrowed down these ideas. One idea particularly stood out, I found it to resonate with me the most and it appeared the most innovative. I chose this idea, confident in its potential and thus Fusion was born!

In today’s age networking has evolved a lot over the last few years. Each platform offers unique features tailored to different aspects of networking, from general professional connections to niche start-up investments. To understand how Fusion stands out, I will present below a competitor’s analysis of some of the key players in this space.

App	Target Audience	Key Features	Limitations
LinkedIn	Professionals from all industries	- Profile creation - Networking with other professionals - Content sharing - Learning resources - Promoting businesses and job openings - In app messaging	- Generalized across all industries, which can dilute the relevance of connections for niche sectors
AngelList	Start-ups, job seekers, investors	- Start-up profiles - Investor networking - Tools for pooling funds	- Primarily aimed towards start-ups and investors, lacking broader networking features
Bumble Bizz	General professionals, with a focus on women	- Swipe deck feature - Easy and fun interface - Location-based connections - Chat and messaging	- General professional networking without a specific focus on business sectors or start-ups
Meetup	Individuals looking to join group activities	- In-person and virtual meetups - Communication boards -Event scheduling - Networking	- Primarily event-based, less focused on individual professional connections
Sharpr	Groups looking to organize meetings	- Group creation and management - In-person and virtual meetings - In-app messaging - Discover and search	- Focused on group interactions rather than individual networking
Fusion	Entrepreneurs, investors, business professionals	- Swipe cards feature - Swipe-based connection by location and business sector - Geohashing Algorithm - Recommender Algorithm – Linear Regression - Niche focus on entrepreneurs, business minded individuals and investors - In app messaging feature	- As an new and emerging app, it may initially have a smaller user base compared to established competitors

Fusion is a business networking application that takes an already existing feature and adds a twist to it. My app uses a card swiping feature and allows for business minded individuals to connect. Once an account is created and your personal and business details are inputted you have the ability to swipe left or right on various business users whether or not you want to connect with them. I have implemented two algorithms called Geohashing and Suitable Recommender Algorithm that uses linear regression these algorithms can aid in bringing business users further more. By creating this application, I hope Fusion will become a hub of opportunities, a tool that helps people interact with each other and develops ideas into real business operations.

1.2 Aims

This project's objective is to link people in the business world. It's time for entrepreneurs and business minded people to step outside their comfort zones and take advantage of exciting opportunities. Even those who identify as introverts may be surprised by how easy it is to start networking using Fusion! One of the most important things a business person can learn is to face their fears head-on, network and with the right attitude it can also help create lasting relationships and bring immense benefits. My aim is to facilitate communication between business enthusiasts, investors, and entrepreneurs. The app is designed to connect users with a business background with potential business partners, investors entrepreneurs who have similar business interests and objectives. I believe building a network is key to allowing ideas to develop into lucrative businesses and my app builds the bridge for users to do so. My goals are to create opportunities for collaboration, spark creative conversations, and launch my own business in this technology sphere. Figure 1.2.1 is a visual representation of my aim of connecting business minded individuals with each other and creating valuable networks.

Figure 1.2.1 – Business Networking Connections between users – Collegexpress(2023) How to build a great presence on LinkedIn to grow your network, Available from: https://www.collegexpress.com/articles-and-advice/career-search/articles/career-communication/build-great-linkedin-grow-your-professional-network/ [Accessed 15th of April 2024]

1.3 Technology

The technology behind Fusion is comprised of a few components that work together to create my app.

Swift: Swift is Apple's programming language for iOS, iPadOS, macOS, tvOS, and watchOS. Swift, is powerful and user-friendly, it is known for its efficiency and quickness. The syntax is clean and elegant to code which personally I find it is a joy to work with as it is easier to write and read compared to other coding languages. Swift provides the framework for my project, enabling me to effectively use the most recent developments in software development with features like functional programming ideas, generics, and error handling.

The key reasons I have chosen to code my app in Swift is first of all safety and security. Because of its simple syntax, developers are encouraged to write code that is easy to read and less likely to contain errors. Swift's strong typing system and error-handling features help to catch bugs before the app launches, preventing crashes and increasing its dependability and security. Swift has a clean and expressive syntax which makes Swift a joy to work with, this allows novice and experienced developers to pick up the language quickly and build apps efficiently. Swift is also backed by extensive documentation, a large community, and continuous updates. I like that the ecosystem includes a wealth of libraries and tools that simplify the process of developing, testing, and deploying applications.

Firebase Database: Firebase Database is a reliable and flexible cloud-hosted NoSQL database. I can create collaborative features like instant messaging and live user profile updates thanks to its real-time synchronisation capabilities without having to worry about the hassles of server infrastructure management. The database is accessible directly from client devices via native SDKs, which simplifies the architecture by reducing the need for a middle layer of backend services.

The reason I have chosen this database is because it stands out for its simplicity and efficiency. Firebase has a real-time synchronization feature meaning that Fusion has live updates across the user profiles and instant messaging works seamlessly. Another benefit of using this database is the security and scalability, as my user database grows the database will scale without any manual intervention. This means Fusion will be able to handle increased traffic and data volume as it becomes more popular.

SwiftUI: For creating user interfaces for Apple devices, SwiftUI is a popular tool. It quickly builds complicated screens with few commands. I can work quicker and more effectively because I can observe changes as we create the app in real-time using it. SwiftUI makes my software look good on all Apple devices by automatically modifying the layout to fit various screen sizes, ensuring optimal functionality on both iPhone and iPad and also helping create a flawless UI.

Geohashing Algorithm: Fusion's swiping feature is based of two innovative algorithms and the first one I will be discussing is a clever application of the geohashing algorithm. This algorithm works with two essential components, Core Location and Geohashing. Apple provides a framework called Core Location that can pinpoint the exact location of a device. The framework collects information from all of the device's accessible components, such as the GPS, Bluetooth, WIFI, cellular hardware, among many others. Only if the user has granted permission will this be carried out. The Core Location maps the user's location by latitude and longitude after the user grants permission. The Geohashing then receives these coordinates.

Geohashing algorithm uses Core Location to encode geographic coordinates. These coordinates are represented by a short string of letters and numbers that is generated from geographic coordinates(longitude and latitude). With this data, it constructs a virtual circle with a radius of 500 kilometres centred on the main user. Then, the algorithm starts working, comparing this circle to other users' hashed locations.

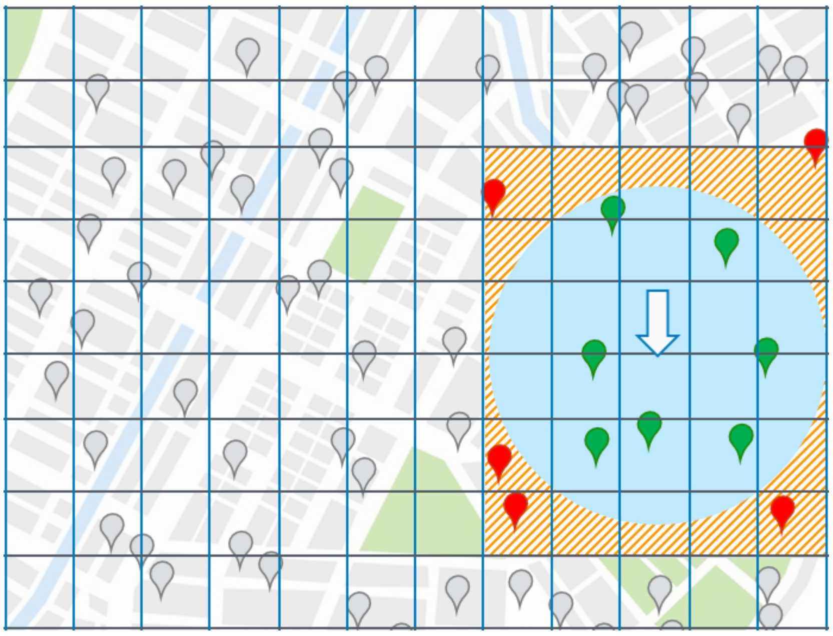


Figure 1.3.1 – Grid view of Geohashing – Amazon(2020) Implementing geohashing at scale in serverless web applications, Available from: https://aws.amazon.com/blogs/compute/implementing-geohashing-at-scale-in-serverless-web-applications/ [Accessed 15th April 2024]

Figure 1.3.1 shows a visual representation of how geohashing works. Geohash corresponds to a grid cell, the search radius overlaps multiple cells. Only the green pins are relevant to the radial search, even though the query returns all location pins inside the cells. Since the grey pins are outside of the geohash cells that overlap, they are removed from the query right away.

Recommender Algorithm – Linear Regression: Fusion's second innovative algorithm is a recommender algorithm specifically using linear regression. This is a well-known algorithm in statistics and machine learning. Based on figure 1.3.2 below you can see that linear regression is a is a model that assumes a linear relationship between the input variables x and the single output variable y. It will calculate y from a linear combination of the input variables x. It calculates the output by fitting a line through the data based on a linear combination of the input variables. In the context of Fusion, these input variables include business fields and geographic location. By analysing these features, the algorithm predicts and recommends highly relevant networking connections.

This machine learning approach makes sure that Fusion's users are matched with others who share similar business interests and are located nearby, enhancing the relevance and value of each connection. Linear regression's ability to identify patterns and correlations within the data allows Fusion to suggest potential connections with a high degree of accuracy. By adding this innovative use of technology, Fusion significantly improves the efficiency and effectiveness of professional networking.

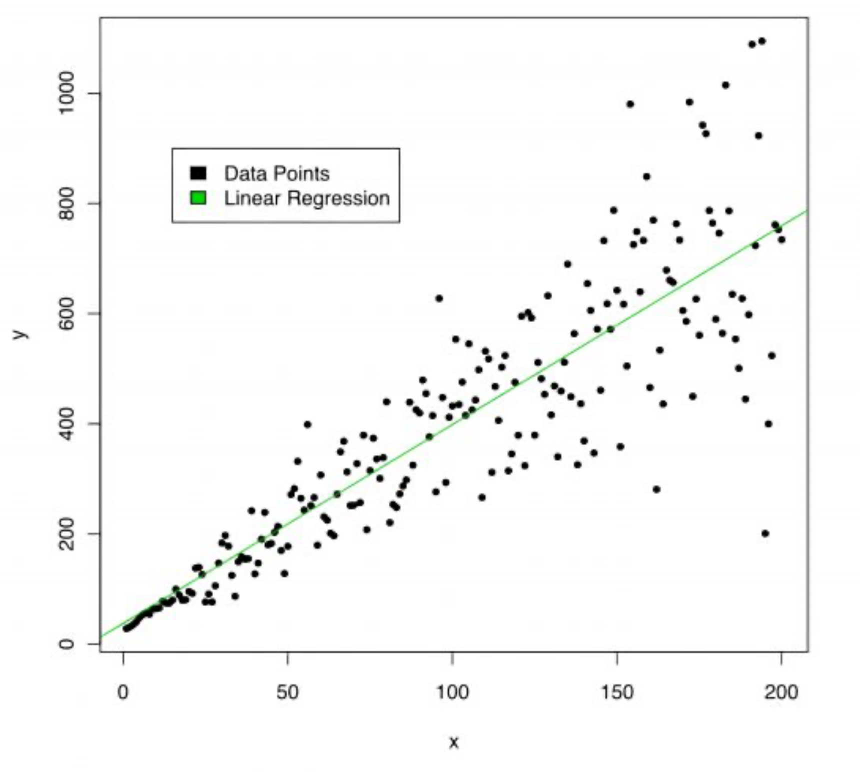


Figure 1.3.2 – Linear Regression Diagram – Kodeco(2022) Building a recommendation app with CreateML in SwiftUI. Available from: https://www.kodeco.com/34652639-building-a-recommendation-app-with-create-ml-in-swiftui [Accessed 2nd April 2024]

1.4 Structure

Introduction - Introduces the purpose, scope, and objectives of the project, setting the stage for the detailed discussion that follows.

Background - Provides context for the project, detailing the initial inspiration and the specific needs it addresses within professional networking.

Aims - Outlines the main objectives of the project, along with the anticipated effects and the particular issues it seeks to resolve.

Technology - Describes the key technologies and frameworks utilised in the project, explaining their relevance and contribution to the project's goals.

System - Discusses the overall system architecture, including both high-level design and detailed descriptions of each component.

Requirements - Breaks down the specific functional, data, user, environmental, and usability requirements, going into detail about how they each align with the project's objectives.

Design & Architecture – This explains the structural design choices, including diagrams and justifications for the architectural approaches used.

Implementation - Covers the practical application of design principles in building the project, noting any significant challenges and solutions.

Graphical User Interface (GUI) - Describes the design and functionality of the user interface, emphasizing usability and user experience.

Testing - Details the testing strategies employed, including unit, UI, and performance testing, and discusses the outcomes and adjustments made.

Evaluation - Reflects on the project's success in meeting its goals, including an assessment of functionality, performance, and user feedback.

Conclusions - Summarizes the project outcomes, lessons learned, and the potential for future work or improvements.

Further Development or Research - Identifies areas for further development or research that could enhance the project or extend its capabilities.

References - Lists all sources I cited in the development and documentation of the project.

Appendices - Provides supplementary material that supports the main text, such as code listings, additional data, or detailed diagrams.

Project Proposal - Includes the original proposal document outlining the intended research methodology, objectives, and expected outcomes.

Reflective Journals - Features personal reflections on the project process, including challenges faced and knowledge gained.

2.0 System

2.1 Requirements

In order to meet the standards of a fully functioning app, these are the requirements I have set:

1. **Login Feature:** Secure login mechanisms will be implemented to protect user data and ensure privacy.
2. **Firebase Database Integration:** I will integrate Firebase for robust and responsive data storage and management.
3. **User Profile Creation and Editing:** User-friendly profile management will enable users to create and customize their profiles with minimal effort.
4. **Design Profile Features:** The app will have intuitively designed profile features to enhance user engagement and make my app easy to use.
5. **Recommender Algorithm – Linear Regression:** I will develop a matching algorithm to enable efficient and relevant connections between users based on details in their profiles such as field of business, and location.
6. **Geohashing Algorithm:** The app will incorporate geohashing algorithms to convert geographic coordinates into short strings of letters and numbers, ensuring precise and privacy-preserving location sharing. This feature will facilitate location-based functionalities while safeguarding user location data.
7. **Design:** To make the app easy to use, transparency and ease of use will be given top priority in the app's design. With a design that avoids complexity and concentrates on a clear interface and a logical user journey, it will be simple to use and enable users to network and communicate without difficulty.
8. **Performance:** The app will be optimized for performance, aiming for quick load times and smooth transitions between different sections, ensuring that it runs smoothly even under heavy user load.
9. **Security:** The app will incorporate security measures to protect user data. This will include encryption for data in transit and secure protocols for storing personal information.

2.1.1. Functional Requirements

Below I have outlined the core functionalities that I expect my application to deliver. Functional requirements are the primary tasks and services that the software must perform. The functionalities are prioritised by importance.

1. **Enable Secure User Registration:** Provide a safe and secure sign-up process for all users.
2. **Firebase Database Integration:** Incorporate Firebase for efficient data management.
3. **Facilitate Profile Customization:** Allow users to personalize their profiles easily.
4. **Swiping feature:** Build a swiping feature allowing users to swipe to connect.
5. **Linear Regression Matching Algorithm:** Integrate the recommender algorithm to match users based on specific criteria such as location and field of business.
6. **Geohashing Algorithm:** Integrate the geohashing algorithm to convert user geographic coordinates into strings. This functionality will be able to provide location based services based on their geohash values while still remaining secure.
7. **Ensure Real-time Messaging:** Enable users to communicate instantly with one another within the app.
8. **Optimize for Performance:** Guarantee that the app operates smoothly under various conditions and loads.
9. **Maintain Data Security:** Protect all user data with encryption and secure storage solutions.
10. **Ensure Accessibility:** Design the app to be accessible to users with different abilities.

2.1.1.1. Use Case Diagram & User Stories

Title: Finding potential business partners	Priority:	Estimate:

User story

As an entrepreneur

I want to swipe through potential business partner’s profiles.

so that I can create a business with individuals I have found with similar business interests and goals.

Acceptance criteria

Given that users can swipe through profiles after completing their on profile.

when profiles are displayed based on certain fields like location and field of business

then app provides a seamless swiping experience

Title: Creating a Profile	Priority:	Estimate:

User story

As a new entrepreneur,

I want to create a fully complete application.

so that I can showcase my personal details, business field, whether I have a business or not, interests, picture(s), field of business that I am in.

Acceptance criteria

Given that users can register and log into my app

when users can add their details, pictures, business interests etc.

then a profile is created, the process is user friendly.

Title: Messaging Feature	Priority:	Estimate:

User story

As a business individual

I want to communicate with other users I have connected with through an in-app messaging feature.

so that I can discuss business ideas, opportunities directly in chat.

Acceptance criteria

Given that messaging feature is available after two users connect.

When messaging interface is user-friendly and allows for easy communication.

Title:	Priority:	Estimate:

User story

As an entrepreneur

I want to access a section dedicated to investors.

so that I can find potential funding opportunities for my startup.

Acceptance criteria

Given that there is a separate section for investors is clearly accessible within my app

when startups can showcase their ideas and business models to attract investors

then Investors can browse through startup profiles and initiate conversations

Title:	Priority:	Estimate:

User story

As an admin
I want to manage user accounts, moderate content on platform, access detailed analytics, provide technical support, regulate, and maintain app.

so that I can view, edit delete profiles, maintain professional app, understand user behaviors and improve app accordingly, ensure smooth experience and address technical issues

Acceptance criteria

Given that Admin can view, edit, deactivate user profiles, receive reported content and take action, view key metrics such as user engagement and user patterns, receive, respond to technical queries, implement and manage updates to app

when admin can respond to user queries and complaints, automatically flags inappropriate content for review, undergo regular maintenance checks.

then the admin dashboard shows my user management tools, admin can communicate with users regarding content issues, data is presented in an easy-to-understand format, users can receive responses to their problems, users are informed about updates.

Title:	Priority:	Estimate:

User story

As an entrepreneur looking for a business partner

I want to connect with individuals based on geographical proximity.

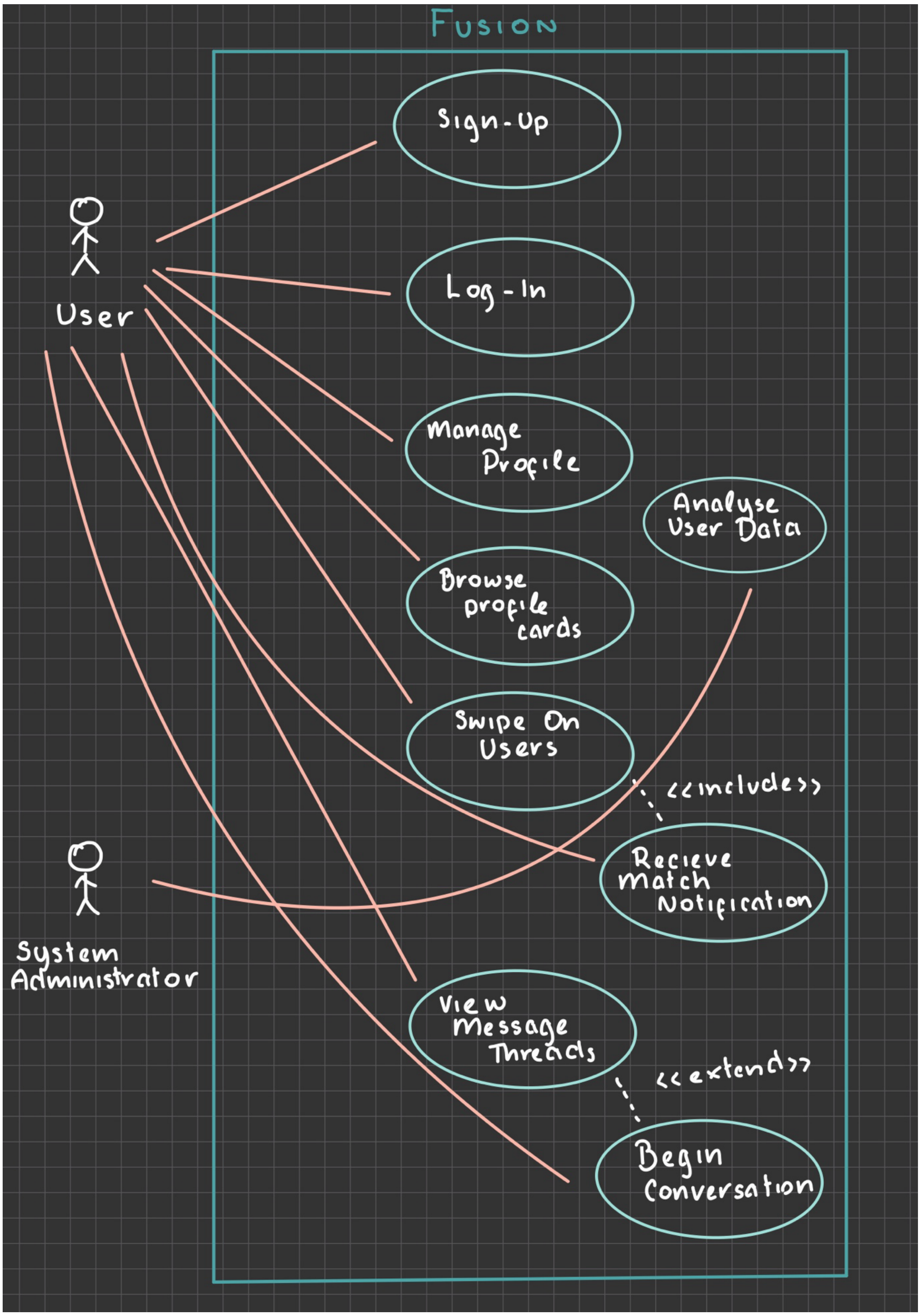
so that I can network locally and find a business partner and expand my business contacts in my area

Acceptance criteria

Given that Users can filter or search for connections based on geographical location

when app provides suggestions for local networking opportunities

then geographical data is accurately and securely used to facilitate connections.


Fusion
Sign-Up
Log-In
Manage Profile
Analyse User Data
Browse profile cards
Swipe On Users
<<includes>>
Recieve match Notification
View Message Threads
<<extend>>
Begin Conversation
User
System Administrator

2.1.1.2. Description & Priority

As the swiping feature is the main way users engage with the app, it is a high-priority requirement. This feature is a simple way for users to show their interest in or disinterest in possible business connections. The app's design is essential to user engagement and retention since it makes the experience both pleasant and functional.

It's fundamental to the app's user interface and core functionality of the app. Swiping is the primary way for users to navigate through potential connections. The swiping function is vital for keeping user interest and remains competitive in the fast paced world of business networking apps.

The swiping feature will incorporate a recommendation algorithm based on linear regression. The recommendation algorithm uses data from three key parameters: 1. Likes and dislikes, 2. Location that is derived from geohashing and 3. Business field. This information is fed into the model to generate personalised recommendations that appear at the top of the screen. The algorithm ensures the most relevant connections are presented to the user, enhancing the swiping feature.

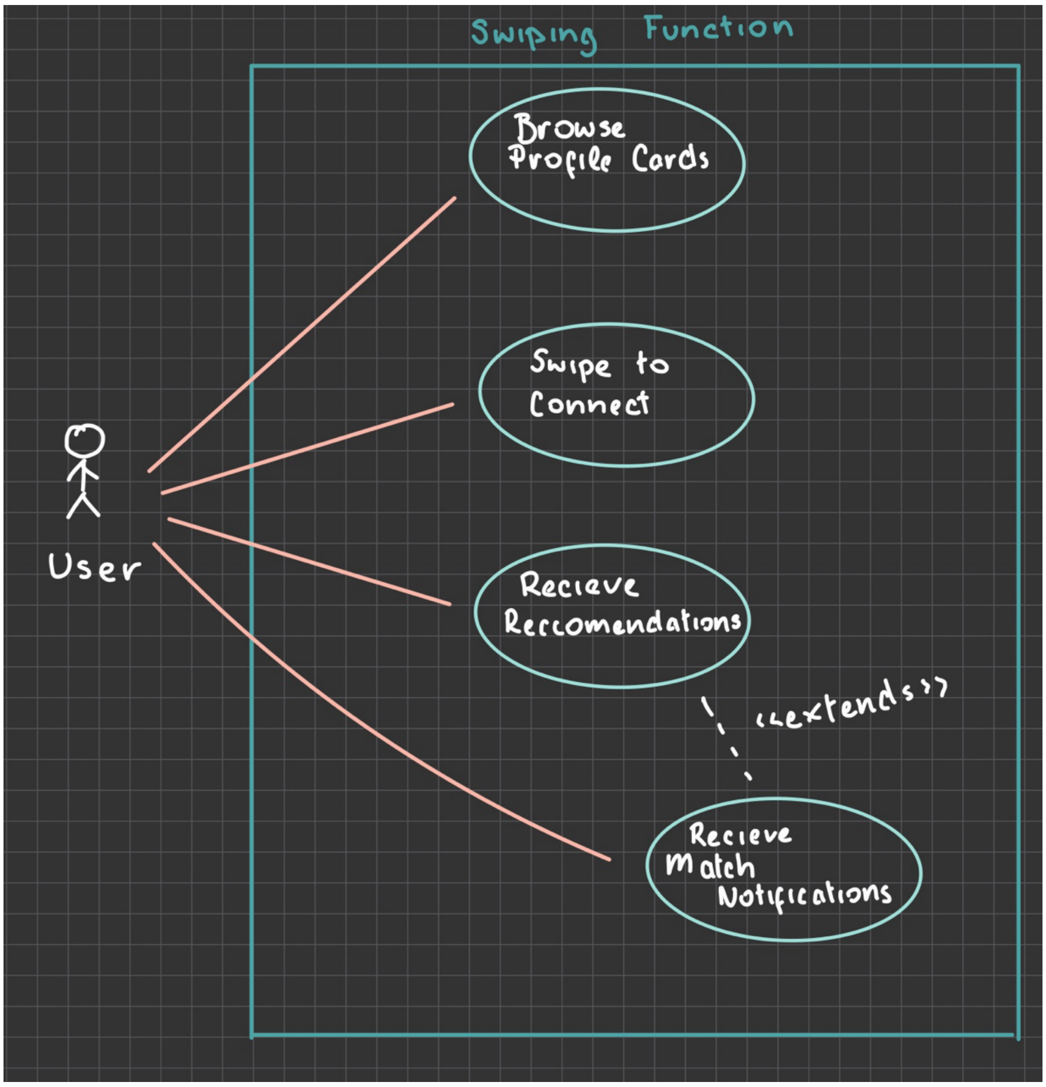
2.1.1.3. Use Case

Use Case ID: UC – 01

Scope: This use case covers the interactions of the user with the swiping mechanism within fusion.

Description: This use case enables users to engage with the core functionality of the app - browsing and connecting with other business profiles by swiping through cards of various users.

Use Case Diagram:



FlowCondition -

Pre-Condition: The user is logged in and shows with potential business matches.

Activation: This use case is activated when a 'User' initiates a swipe action.

Main Flow:

1.The app shows a stack of profile cards to the 'User'.

2.The 'User' swipes right to 'Connect' or left to 'Pass' on a profile.

3.If a swipe right occurs, the system checks for a mutual right swipe and recommends 3 new users (See A1).

4.If mutual interest is confirmed, the system notifies both users of the match (See E1).

5.If the 'User' swipes left, the system discards the current profile and presents the next one.

Alternate Flow:

A1: Mutual Interest Not Found

1.If a mutual right swipe is not detected, the system logs the swipe.

2.The 'User' continues to swipe through other profiles.

Exceptional Flow:

E1: Match Notification

1.Upon detecting a mutual right swipe, the system creates a match.

2.Both 'Users' are alerted of the match via pop up on screen.

3.The use case returns to the main flow at step 5.

Termination: The use case ends when the 'User' exits the swiping interface or logs out.

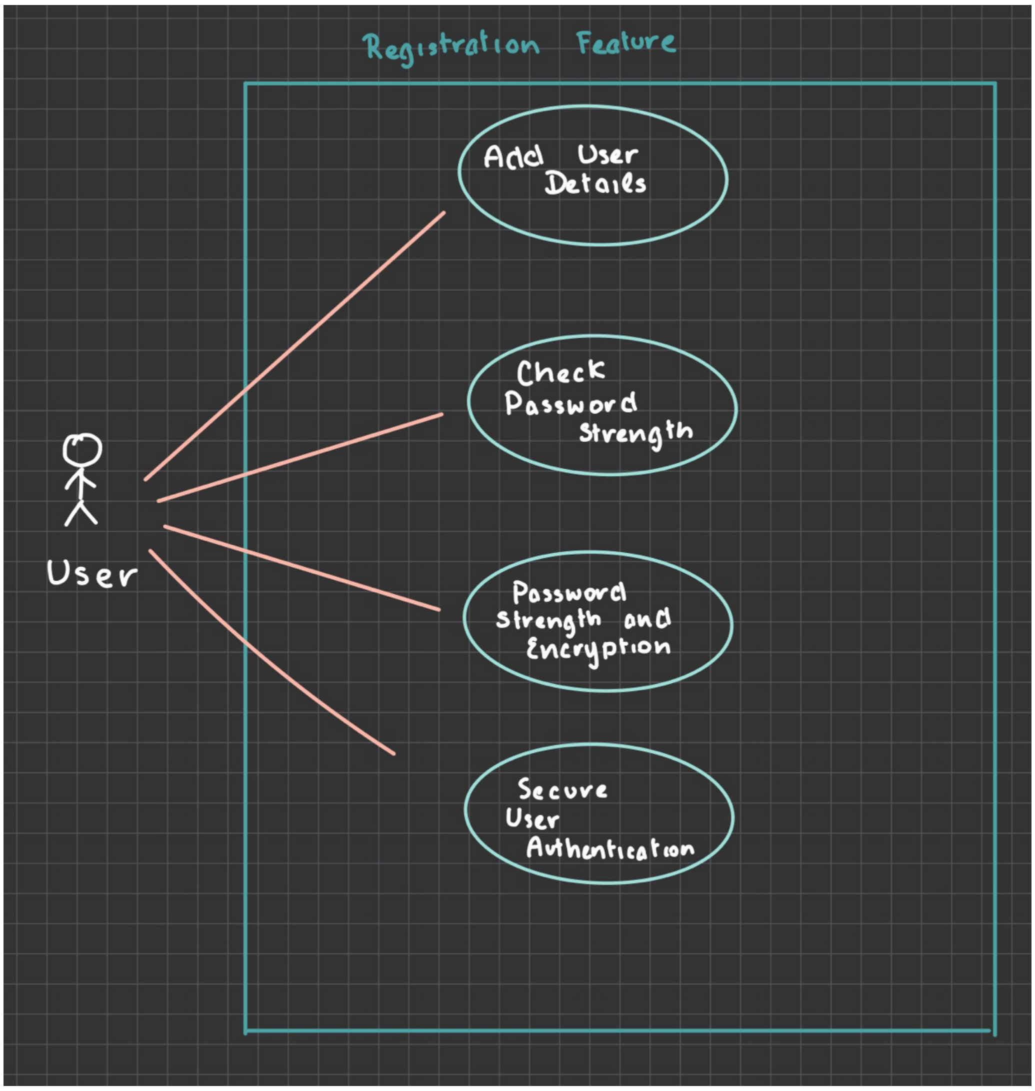
Post Condition: The system records the swipes and updates any matches. The system then goes into a wait state for the next user action.

Use Case ID: UC – 02

Scope: This use case covers the interactions of the user registering on Fusion

Description: This use case enables users to engage with another core functionality of the app – registering and creating an account.

Use Case Diagram:



FlowCondition -

Pre-Condition: The user is not logged in and wants to create a new account.

Activation: This use case is activated when a 'User' initiates the sign-up process.

Main Flow:

1. The user initiates the registration process by choosing to sign up.

2. The app prompts the user to enter their details (e.g., name, email, business field etc).

3. The user enters their details.

4. The system prompts the user to create a password.

5. The user creates a password.

6. The system checks the password strength.

7. If the password meets the strength requirements, the system encrypts the password and stores it securely.

8. Fusion integrates with Firebase Authentication to securely manage the user's account.

9. The system confirms successful registration and notifies the user.

Alternate Flow:

A1: Password Does Not Meet Strength Requirements

1. If the password does not meet the strength requirements, the system notifies the user to create a stronger password.

2. The user creates a new password.

3. The system rechecks the password strength.

Exceptional Flow:

E1: Registration Error

1. If there is an error during the registration process, the system notifies the user of the error.

2. The system allows user to retry the registration process

Termination: The use case ends when the user successfully completes the registration process or exits the registration interface.

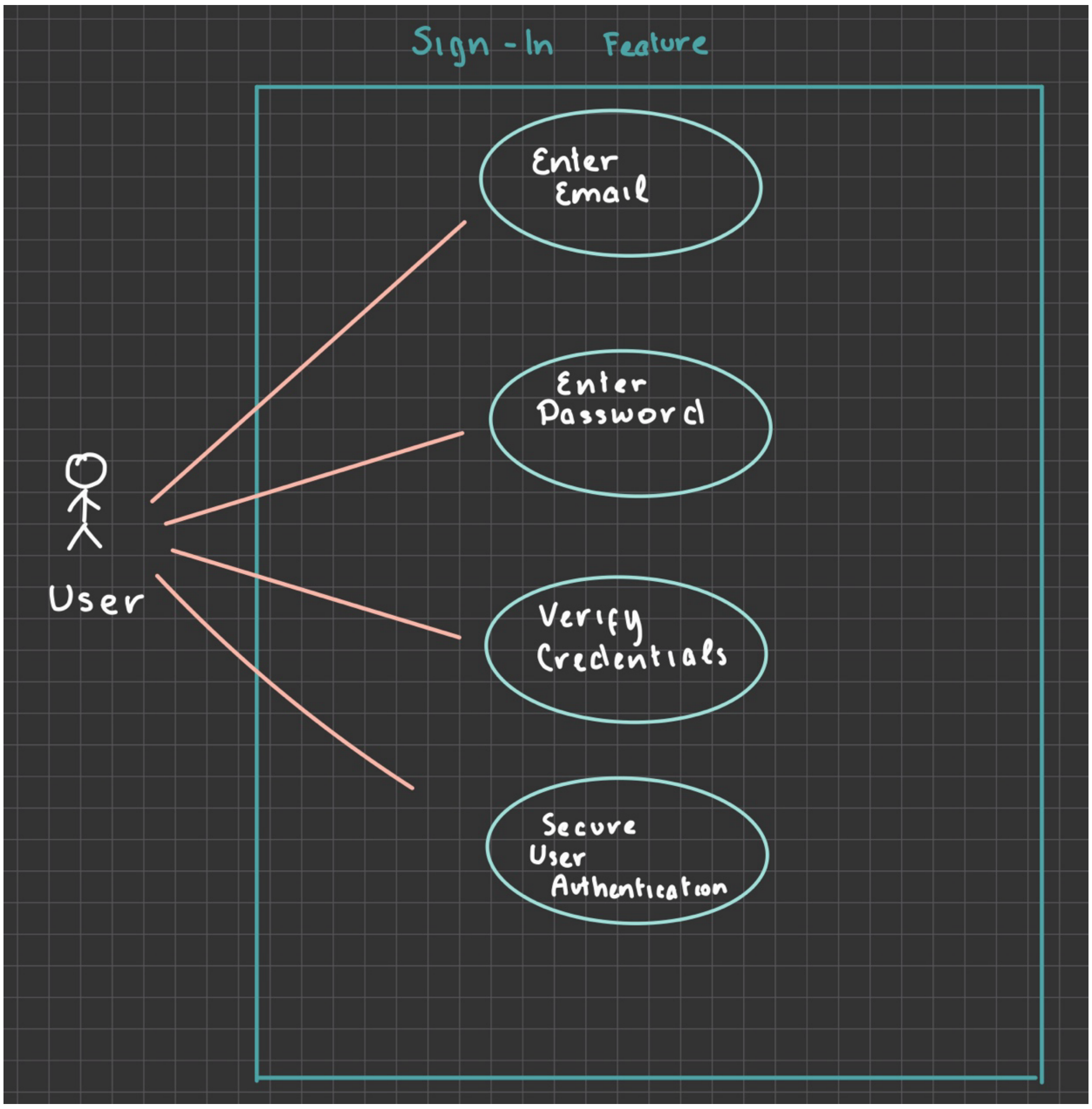
Post Condition: The system securely stores the user's encrypted password and registration details.

The system waits for the next user action.

Use Case ID: UC – 03

Scope: This use case covers the interactions of the user log in on Fusion app.

Description: This use case enables users to engage with another core functionality of the app – logging into an account.



FlowCondition -

Pre-Condition: The user has an existing account and is not currently logged in.

Activation: This use case is activated when a 'User' initiates the sign-in process.

Main Flow:

1. The user initiates the sign-in process by choosing to log in.

2. The app prompts the user to enter their email.

3. The user enters their email.

4. The app prompts the user to enter their password.

5. The user enters their password.

6. The system verifies the entered credentials.

7.If the credentials are correct, the system authenticates the user using Firebase Authentication SDK.

8. The system grants the user access to the app and confirms successful login.

Alternate Flow:

A1: Incorrect Credentials

1. If the entered credentials are incorrect, the system notifies the user of the failed login attempt.

2. The user is prompted to re-enter their credentials.

3. The user re-enters their credentials.

4. The system rechecks the credentials.

Exceptional Flow:

E1: Sign In Error

1. If there is an error during the sign in process, the system notifies the user of the error.

2. The system allows user to retry the sign in process

Termination: The use case ends when the user successfully logs in or exits the login interface.

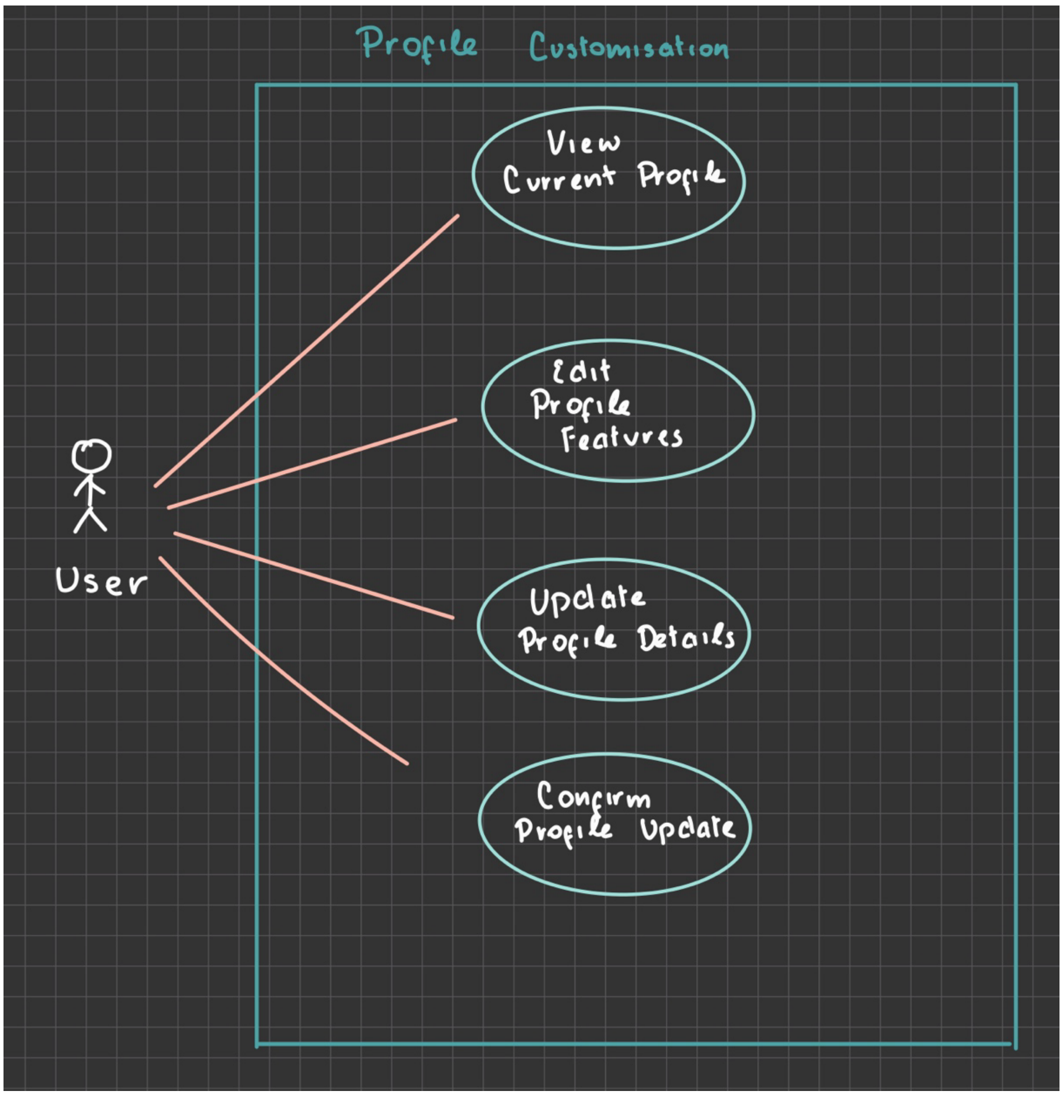
Post Condition: The system securely authenticates the user and grants access to the app.

The system waits for the next user action.

Use Case ID: UC – 04

Scope: This use case covers the interactions of the user log in on Fusion

Description: This use case enables users to engage with another core functionality of the app – updating their profile.



FlowCondition -

Pre-Condition: The user is logged in and accessing their profile.

Activation: This use case is activated when a 'User' initiates the profile customisation process.

Main Flow:

1. The user initiates the profile customisation process.

2. The app displays the current profile details to the user.

3. The user selects an option to edit their profile.

4. The user updates their business interests, avatar, experience, and bio.

5. The system validates the entered information.

6. The system saves the updated profile details.

7. The user reviews the changes and confirms the updates.

8. The system confirms the successful update and displays the updated profile.

Alternate Flow:

A1: Invalid Profile Details

1. If the entered profile details are invalid (e.g., missing required fields), the system notifies the user of the errors.

2. The user corrects the errors.

3. The system revalidates the entered information.

Exceptional Flow:

E1: System Error

If there is a system error during the profile update, the system notifies the user of the error.

The system provides options to retry the update or contact support.

Termination:. The use case ends when the user successfully updates their profile or exits the profile customisation interface.

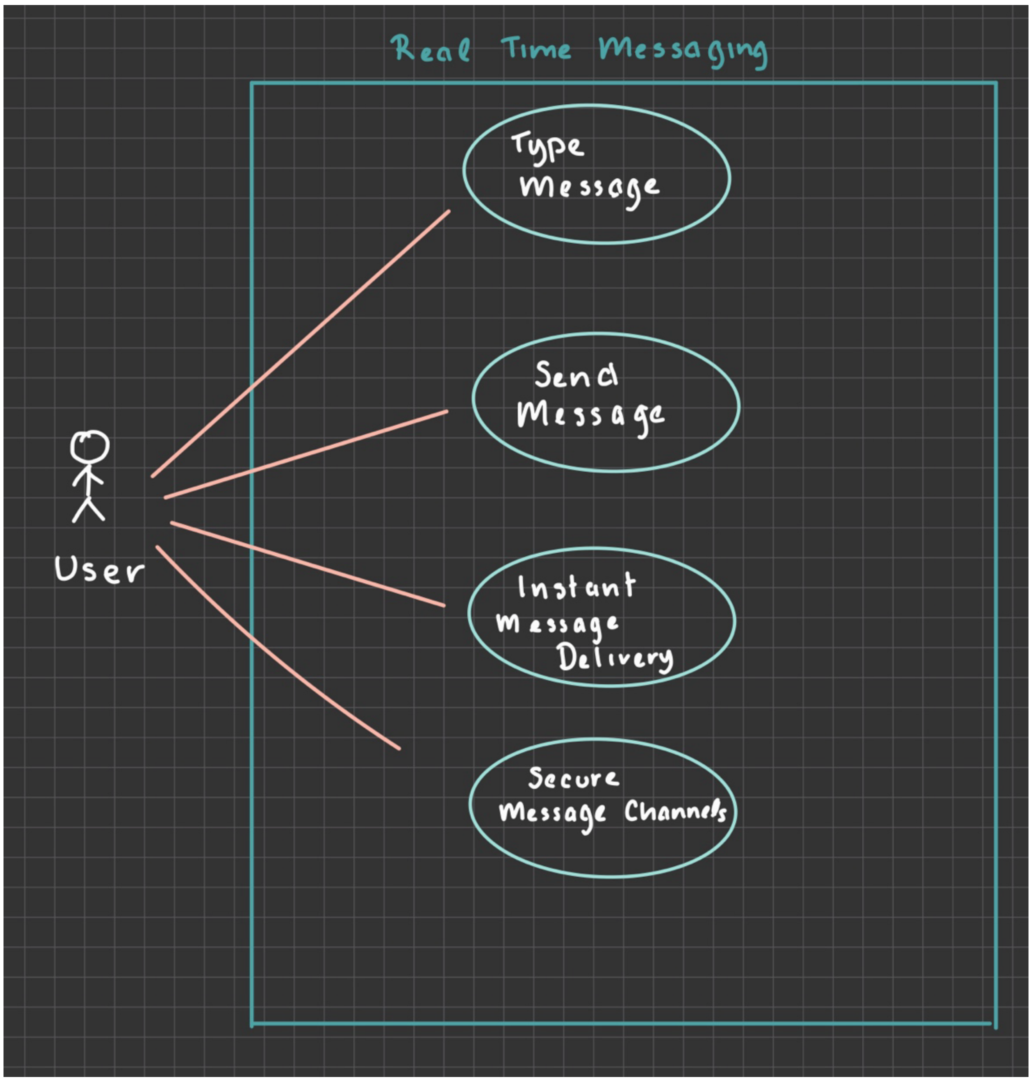
Post Condition: The system securely stores the updated profile details.

The system waits for the next user action.

Use Case ID: UC – 05

Scope: This use case covers the interactions of the user in the messages .

Description: This use case enables users to engage with another core functionality of the app – messaging feature.



FlowCondition -

Pre-Condition: The user is logged in and engaged in a message thread with another user.

Activation: This use case is activated when a 'User' connects with another user and initiates the sending a message.

Main Flow:

1. The user types a message in the chat interface.

2. The user sends the message.

3. The system instantly delivers the message to the recipient.

4. The system secures the message during transmission to protect privacy.

5. The recipient receives the message in real-time.

Alternate Flow:

A1: Message Delivery Failure

1. If the message fails to deliver, the system notifies the sender.

2. The system attempts to resend the message.

3. If the message is successfully delivered after retrying, the system confirms delivery.

Exceptional Flow:

E1: Network Issues

If there are network issues, the system notifies the user of the connectivity problem.

The system queues the message to be sent once the network is restored.

The message is delivered once the network connection is re-established.

Termination: The use case ends when the message is successfully delivered or the user exits the messaging interface.

Post Condition: The system ensures the message is securely stored and accessible to both the sender and recipient.

The system waits for the next user action.

2.1.2. Data Requirements

<u>Data to be stored on users:</u>

User Profile Information: This includes the user's name, email, hashed password, profile picture, business fields, bio, company and experience

Business Data: This includes the user's business ventures, business name, industry sector and business location

Networking Data: Records of user interactions within the app, including connections made, likes and dislikes.

User Preferences: Settings and preferences selected by the user for app notifications, privacy settings, and matching preferences.

<u>Security of data:</u>

Password Encryption: Encrypting user passwords. This means that even if data is compromised, no one will be able to access the actual passwords.

Access Control: Have strict access control policies ensuring only authorised users can access their personal data by using firebase database.

<u>Data Integrity and Validation:</u>

Regulatory Adherence: Comply with relevant data protection regulations such as GDPR.

Audit Logs: Maintain logs of all data accesses and changes to provide an audit trail for compliance purposes.

<u>Data Use for Improvements:</u>

Analytics: Use anonymised data for app performance analytics to generate insights for feature improvements.

Feedback Integration: Collect user feedback to understand how data usage can be improved for a better user experience.

2.1.3. User Requirements

In order to meet the needs and expectations of Fusion's target audience, I need to ensure it has all these requirements from below.

User Demographics and Characteristics:

Entrepreneurs and Business Owners: Individuals looking to expand their business network and find potential partners or investors.

Investors: Those looking for new and promising business ventures to invest in.

Business Enthusiasts: Users interested in exploring the business landscape for opportunities and collaborations.

Functional User Interactions:

Log in and Sign Up Feature: Have a straightforward login and signup process.

Account Management: Users must be able to easily manage their account settings, including privacy controls and being able to change their account details.

User Experience:

Intuitive Navigation: My app should be easy to navigate, and the user should be able to find their way around the app freely.

Responsive Design: Fusion must be responsive and provide a consistent experience across various devices and screen sizes.

Performance Expectations:

Quick Load Times: Fusion should have fast load times, minimizing the wait for users.

Compliance and Privacy:

Data Privacy: Users should be assured that their data is handled with the utmost care and in compliance with privacy laws.

Transparency: Clear communication regarding how user data is used within the app.

2.1.4. Environmental Requirements

Below are some environmental requirements necessary for Fusion to function optimally.

Platform Compatibility: Fusion will be compatible exclusively with iOS devices, ensuring a tailored experience that takes full advantage of Apple's ecosystem.

Operating System: The minimum iOS version supported will be iOS 15 to ensure Fusion's advanced features and security measures are fully supported.

Internet Connectivity: A stable internet connection is required to access the full range of the app's features, including real-time messaging, swiping and profile updates.

Storage Space: At least 100MB of free storage space should be available to install the app and store essential data.

Battery Usage: My app will be optimized to be energy efficient to minimize battery consumption during standard app usage

Environmental Sustainability: Fusion encourages digital networking, reducing the need for physical travel and thus contributing to environmental sustainability efforts.

2.1.5. Usability Requirements

The usability requirements focus on ease of use, accessibility and overall user satisfaction with app's interface and interactions.

Ease of Navigation: Fusion's navigation will be intuitive, allowing users to move between features seamlessly and without being confused.

Learnability: As app is easy to use, users can familiarise themselves easily with the functions

Consistency: The UI will have a consistent design language and interaction patterns across all features to prevent user confusion.

Aesthetic and Minimalist Design: Fusion's design will be aesthetically pleasing and minimalistic, avoiding unnecessary elements that do not support user tasks.

2.2 Design & Architecture

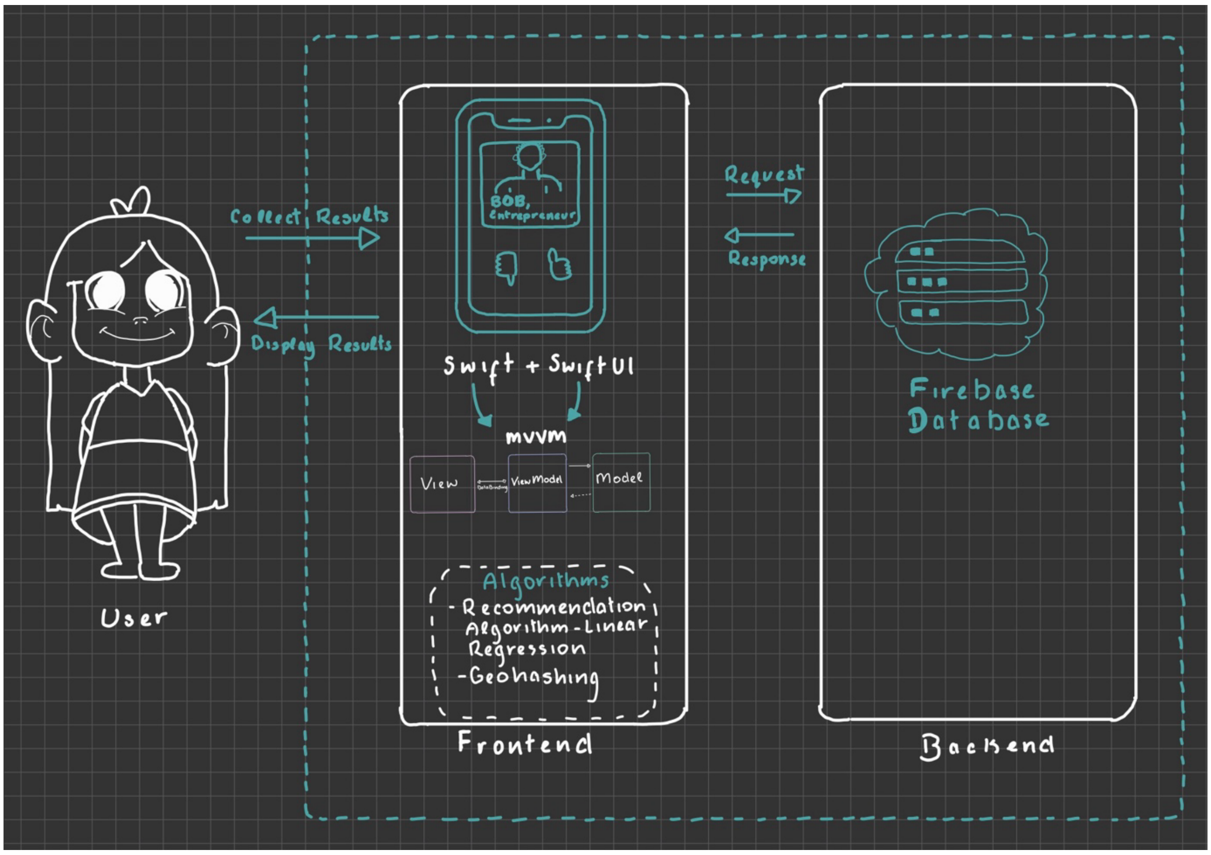


Figure 2.2.1 – Architecture Diagram of Fusion

The architecture of my app is built to support both real-time user interactions and have reliable performance.

<u>The Frontend</u> is built with Swift and SwiftUI. Swift UI is the layer that handles all user interactions and displays the data to users. Business minded people , entrepreneurs, investors etc will be able to effortlessly swipe, match, and message thanks to the GUI's smooth user experience on the front end. For building the app I used MVVM(Model-View-ViewModel) architectural pattern.

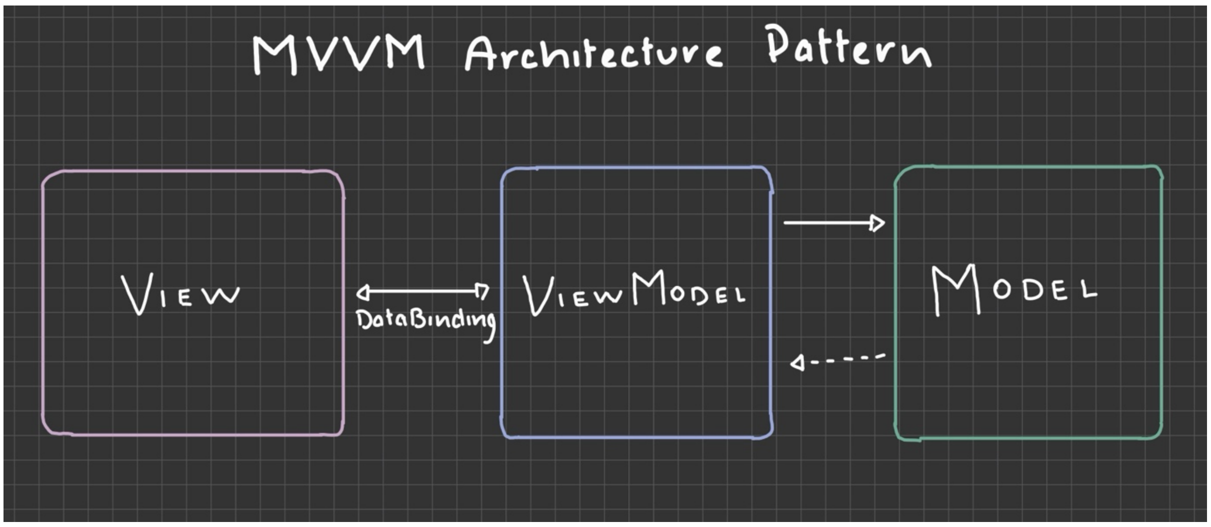


Figure 2.2.2 – Close up of MVVM architecture pattern from figure 2.2.1

MVVM (Model-View-ViewModel) is a design pattern I used in creating Fusion. This design pattern helps me separate my application logic from the user interface. This also makes apps much easier to build and test as It separates different aspects of Fusion so they can be developed individually. A benefit of using MVVM is it makes it easier to upgrade your code in the future as you can make changes in the apps view model and its data binding without having to worry if you will break other parts of the app.

View: What the user sees on the screen is represented by the view. View manages and carries out the application's visual behaviour. It receives keyboard input from the user and responds with feedback.

ViewModel: The ViewModel serves as my application's brain. It is in charge of transferring data to the view (the user interface) from the model, which is a collection of objects, properties, and methods.

Model: The model represents the data in my application. This could include business and validation logic. I only communicates with the ViewModel and lacks awareness of the view.

Google Cloud Storage is used for storing user avatars securely.

Algorithms – The recommendation algorithm processes the user swipes and based on that data uses business fields and location data to generate relevant recommendations.

Geohashing encodes the latitude and longitude from the user into stings for efficiency and privacy.

<u>The backend</u> - Firebase database, which offers a scalable solution for storing user data, managing authentication, user swipes, and overseeing real-time messages. In order to maintain user engagement with the app, Firebase also makes it possible for users to receive rapid notifications when there are new matches or communications which I will all implement within my app.

The diagram illustrates the app's data flow, where a swipe on a user's device sends a request to the cloud server, which then queries the Firebase database for matches and returns the results. When a match is found, A pop up is shown and a message thread is started enabling for both users to begin communication.

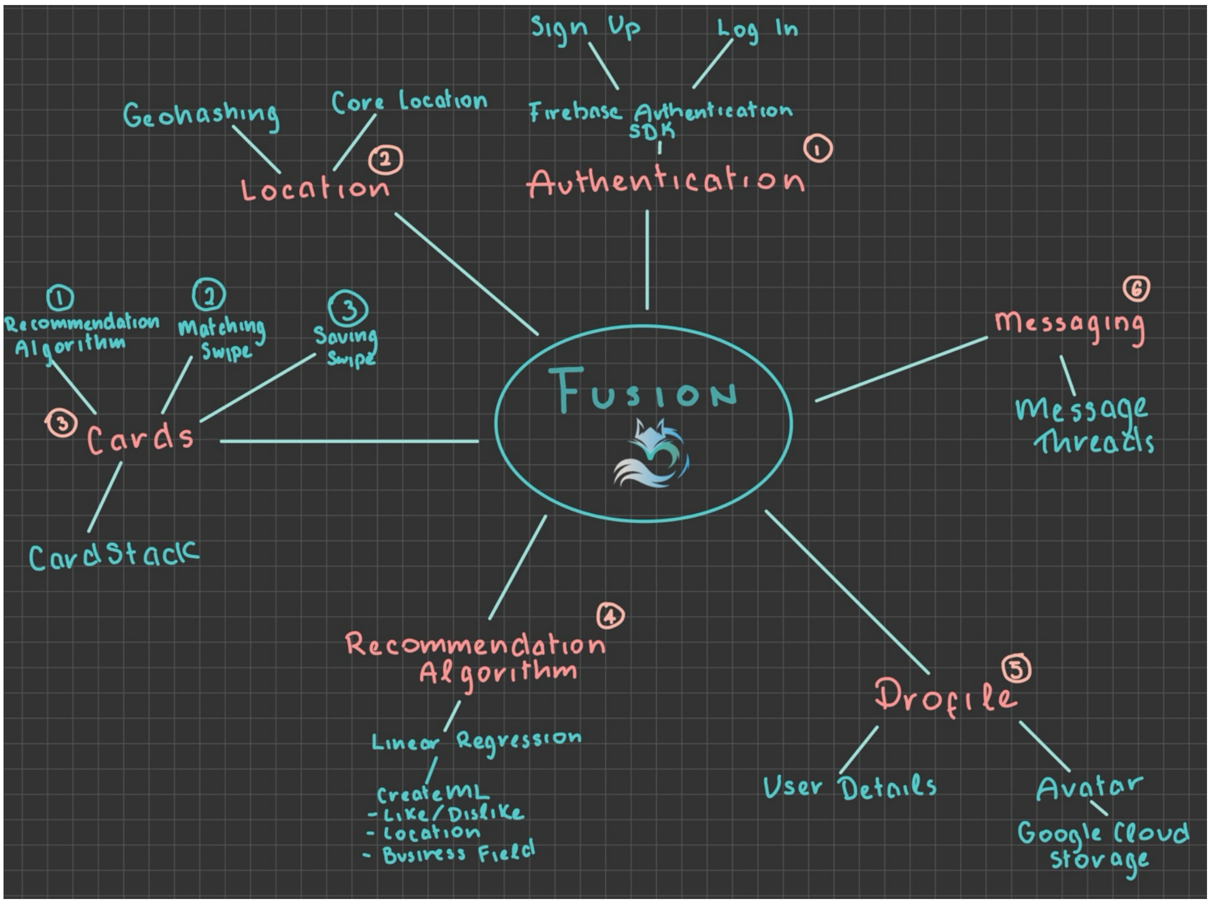


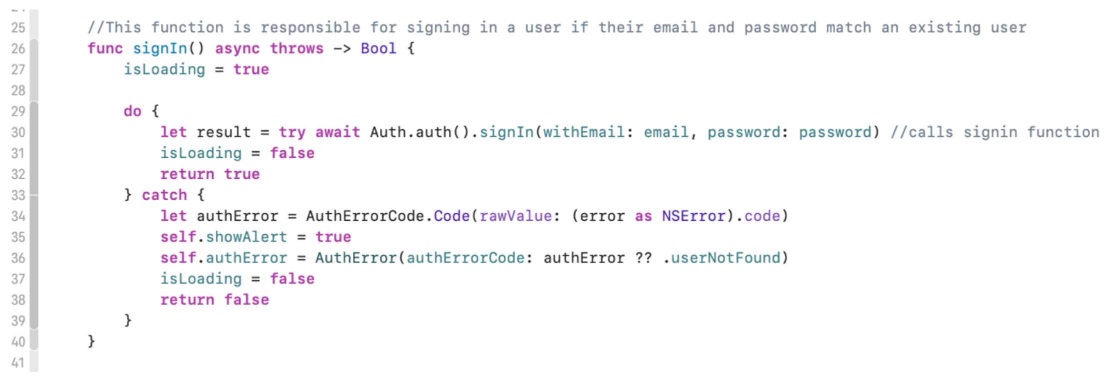
Figure 2.2.3 – Spider Diagram of features in Fusion

In figure 2.2.3 I drew a spider diagram branching out into the different main features of Fusion which are numbered and written in red to distinguish each feature. A detailed explanation about each feature is in the implementation section explaining the features and the components necessary for the feature.

2.3 Implementation

Authentication

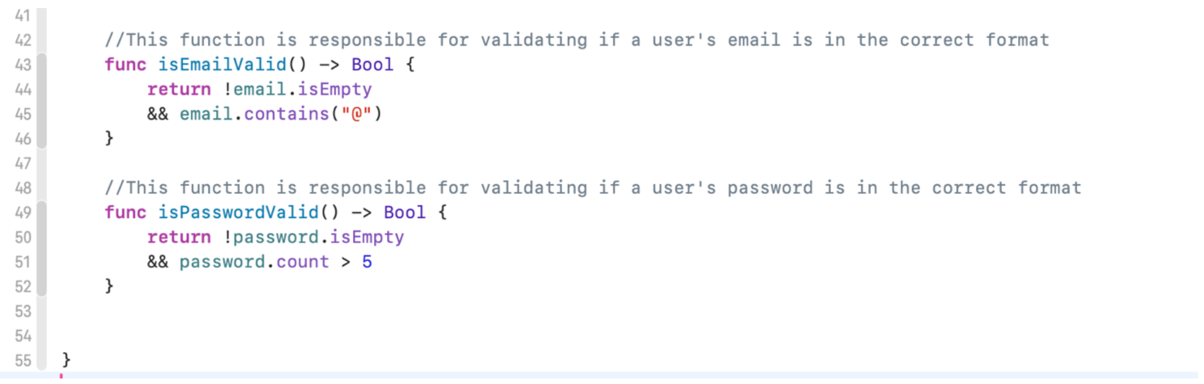
Authentication is a process of signing in the user into the Fusion app. To sign a user into Fusion the credentials used is email and password. These credentials are passed to the Firebase Authentication SDK and then the backend services verify the user's credentials. Authentication is split into two parts 1. Log In and 2. Sign Up. By using firebase authentication I was able to streamline the user login process while also maintaining high security standards.



```
25      //This function is responsible for signing in a user if their email and password match an existing user
26      func signIn() async throws -> Bool {
27          isLoading = true
28
29          do {
30              let result = try await Auth.auth().signIn(withEmail: email, password: password) //calls signin function
31              isLoading = false
32              return true
33          } catch {
34              let authError = AuthErrorCode.Code(rawValue: (error as NSError).code)
35              self.showAlert = true
36              self.authError = AuthError(authErrorCode: authError ?? .userNotFound)
37              isLoading = false
38              return false
39          }
40      }
41
```

Figure 2.2.1 – screenshot of code from LoginViewModel

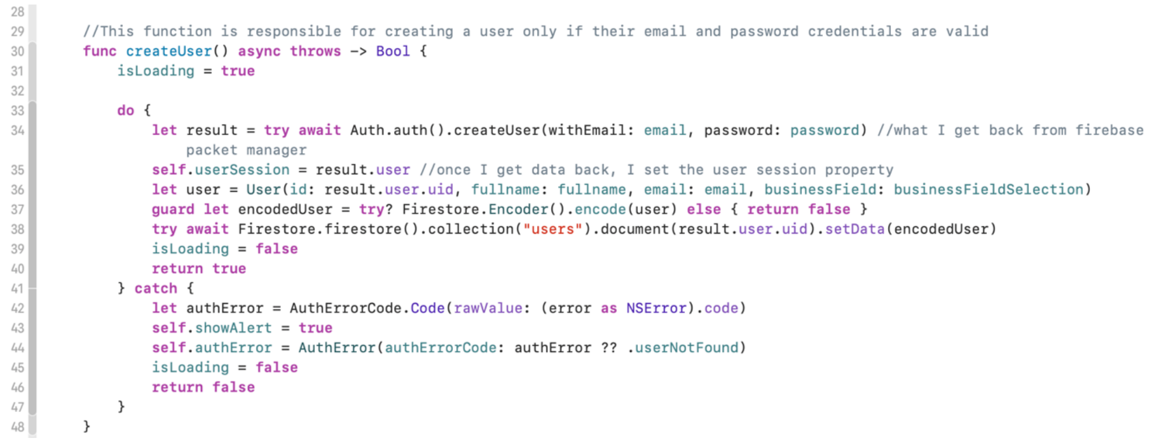
This is a function designed to sign in a user using their email and password credentials. On line 30 I used the Authentication SDK to pass in the email and password that the user's entered into the login form. If the credentials are incorrect, this triggers an error that is captured in the catch block lines 33-39.



```

    //This function is responsible for validating if a user's email is in the correct format
    func isEmailValid() -> Bool {
        return !email.isEmpty
        && email.contains("@")
    }

    //This function is responsible for validating if a user's password is in the correct format
    func isPasswordValid() -> Bool {
        return !password.isEmpty
        && password.count > 5
    }


}
```

Figure 2.2.2 – screenshot of code from LoginViewModel

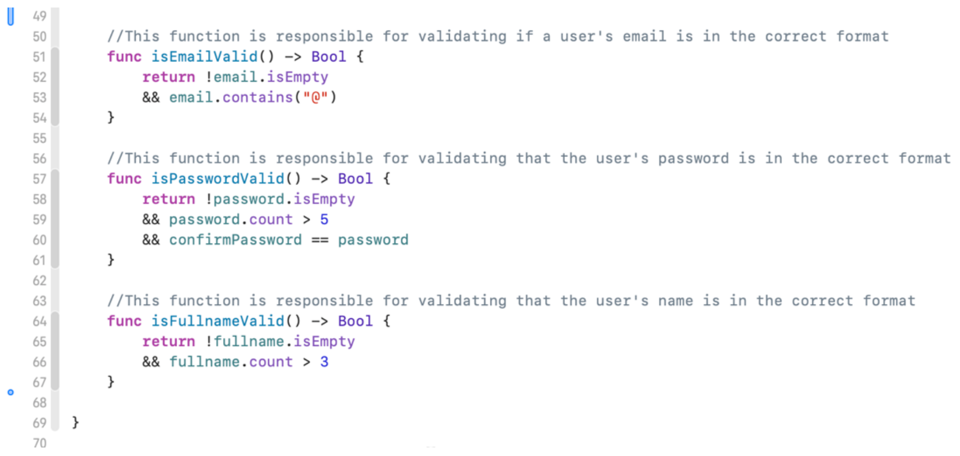
These functions validate the users email and password in the login form. This is the prevent user input errors and to ensure that the users email and password are in the correct format. The functions are enforced by disabling a user from logging in unless their email and password are correct. User Validation is an important part of the concept of GIGO(Garbage in Garbage Out) which refers to the idea that the quality of the user input has a direct impact of the quality of the app's output.



```
28
29      //This function is responsible for creating a user only if their email and password credentials are valid
30      func createUser() async throws -> Bool {
31          isLoading = true
32
33          do {
34              let result = try await Auth.auth().createUser(withEmail: email, password: password) //what I get back from firebase
                    packet manager
35              self.userSession = result.user //once I get data back, I set the user session property
36              let user = User(id: result.user.uid, fullname: fullname, email: email, businessField: businessFieldSelection)
37              guard let encodedUser = try? Firestore.Encoder().encode(user) else { return false }
38              try await Firestore.firestore().collection("users").document(result.user.uid).setData(encodedUser)
39              isLoading = false
40              return true
41          } catch {
42              let authError = AuthErrorCode.Code(rawValue: (error as NSError).code)
43              self.showAlert = true
44              self.authError = AuthError(authErrorCode: authError ?? .userNotFound)
45              isLoading = false
46              return false
47          }
48      }
```

Figure 2.2.3 – screenshot of code from RegistrationViewModel

The following function called createUser is responsible for signing up a user using their email, password, full name, and selected business field. On line 34 I used the Firebase Authentication SDK to sign up a user with their provided username and password. Provided that the email does not already exist and that the email and password are in the correct format it should succeed. If there is an error this is caught by the catch block and handles as an error lines 41-47.



```
    //This function is responsible for validating if a user's email is in the correct format
    func isEmailValid() -> Bool {
        return !email.isEmpty
        && email.contains("@")
    }

    //This function is responsible for validating that the user's password is in the correct format
    func isPasswordValid() -> Bool {
        return !password.isEmpty
        && password.count > 5
        && confirmPassword == password
    }

    //This function is responsible for validating that the user's name is in the correct format
    func isFullnameValid() -> Bool {
        return !fullname.isEmpty
        && fullname.count > 3
    }

}
```

Figure 2.2.4 - screenshot of code from RegistrationViewModel

Within figure 2.2.4 validate the user's email and password and check that they are in the correct format. The user cannot tap the sign up button unless these are correctly formatted. User Validation is an important part of the concept of GIGO(Garbage in Garbage Out).

Geohashing Algorithm

Geohashing and core location are both essential components in location-based services, they both serve two different purposes but they complement each other effectively.

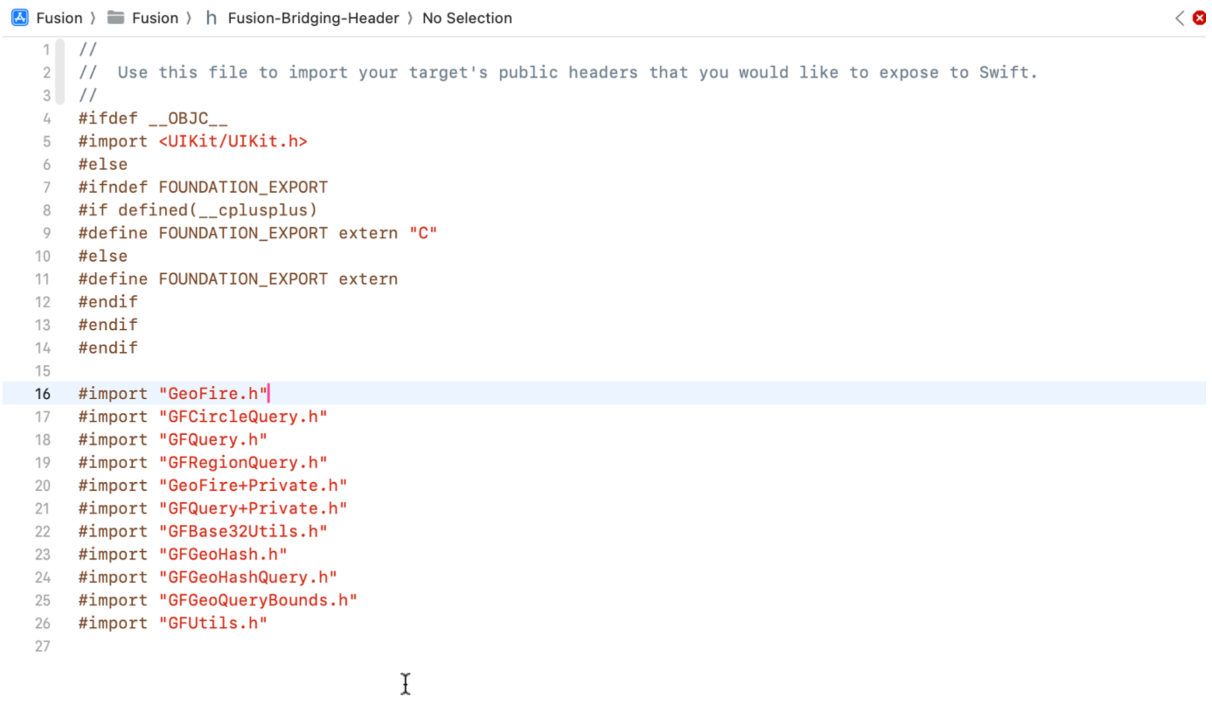
Core Location is a framework from Apple that can determine a device's geographic location. The framework gathers data from available components on the device including GPS, cellular hardware and many more. This is only if the user has given permission to do so. Once the user has granted permission the Core Location then maps the user's location by latitude and longitude. These coordinates are then fed into the Geohashing.

Geohashing is an algorithm that encodes geographic coordinates from Core Location. This encoding allows for efficient and compact representation of geographic locations, which is useful for indexing and querying spatial data. In the Fusion app I use Geohashing to map out people within a predefined radius of 500km. See figure 1.3.1 for a visual representation.

GeoFirestore is a library that I used for the Geohashing algorithm and incorporating it into Fusion required for me to do some creative problem-solving. As this library is written in Objective-C it necessitated me to use CocoaPods to manage the integration into my Swift-based project. However, this approach did not work due to compatibility issues arising from mixing Swift and Objective-C through CocoaPods in a SwiftUI context.

To resolve this issue, I manually copied the necessary GeoFirestore code into Fusion. I also added in a bridging header, this is a file allows Swift to work with Objective-C code within the same project. The bridging header facilitated the import of the Objective-C files and seamlessly translated them into Swift-compatible interfaces.

This solution meant that CocoaPods was no longer needed. By removing CocoaPods and relying on the bridging header, I was happy that I was able to successfully integrate GeoFirestore into my Swift project, ensuring that my app could use the geolocation features for an amazing user experience.

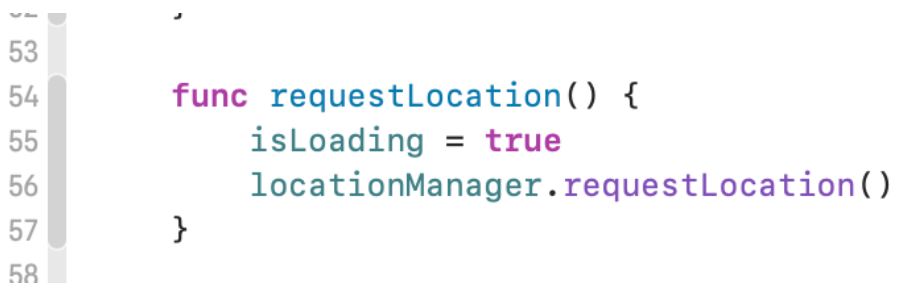


Fusion 〉 Fusion 〉 h Fusion-Bridging-Header 〉 No Selection

```
//
//  Use this file to import your target's public headers that you would like to expose to Swift.
//
#ifdef __OBJC__
#import <UIKit/UIKit.h>
#else
#ifndef FOUNDATION_EXPORT
#if defined(__cplusplus)
#define FOUNDATION_EXPORT extern "C"
#else
#define FOUNDATION_EXPORT extern
#endif
#endif
#endif

#import "GeoFire.h"
#import "GFCircleQuery.h"
#import "GFQuery.h"
#import "GFRegionQuery.h"
#import "GeoFire+Private.h"
#import "GFQuery+Private.h"
#import "GFBase32Utils.h"
#import "GFGeoHash.h"
#import "GFGeoHashQuery.h"
#import "GFGeoQueryBounds.h"
#import "GFUtils.h"

```

Figure 2.3.5

In figure 2.3.5 is a bridging header file within my code.

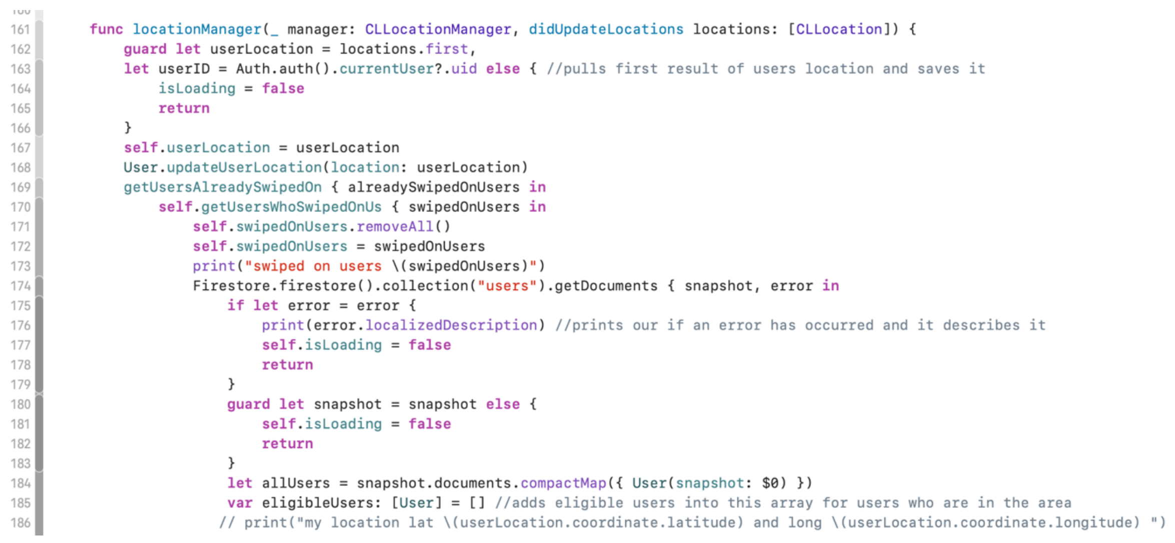


```

    func requestLocation() {
        isLoading = true
        locationManager.requestLocation()
    }

```

Figure 2.3.6

In Figure 2.3.6 you can see the function requestLocation() which obtains the user's geographical location. The 'isLoading' is set to true. This informs the user that a location request is in progress and that the app is working on fetching their current location.

The function then calls locationManager.requestLocation(). This is a method provided by CLLocationManager. When this method is called, the location manager asynchronously attempts to determine the user's current location.



```swift
    func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
        guard let userLocation = locations.first,
        let userID = Auth.auth().currentUser?.uid else { //pulls first result of users location and saves it
            isLoading = false
            return
        }
        self.userLocation = userLocation
        User.updateUserLocation(location: userLocation)
        getUsersAlreadySwipedOn { alreadySwipedOnUsers in
            self.getUsersWhoSwipedOnUs { swipedOnUsers in
                self.swipedOnUsers.removeAll()
                self.swipedOnUsers = swipedOnUsers
                print("swiped on users \(swipedOnUsers)")
                Firestore.firestore().collection("users").getDocuments { snapshot, error in
                    if let error = error {
                        print(error.localizedDescription) //prints our if an error has occurred and it describes it
                        self.isLoading = false
                        return
                    }
                    guard let snapshot = snapshot else {
                        self.isLoading = false
                        return
                    }
                    let allUsers = snapshot.documents.compactMap({ User(snapshot: $0) })
                    var eligibleUsers: [User] = [] //adds eligible users into this array for users who are in the area
                   // print("my location lat \(userLocation.coordinate.latitude) and long \(userLocation.coordinate.longitude) ")
```

Figure 2.3.7

Figure 2.3.7 is a location manager function which is called when CLLocationManager finds the user's location . Capturing the user's location is a pivotal aspect of the Fusion app. The location is used by the geohashing to map users within a 500km radius and is used to present it in the profile cards in the HomeView.

When a location is found three important things happen in the location manager. The first thing is a query is made to the firestore database for users that the logged in user has already swiped on. These users are excluded from showing in the profile cards. The second thing a query is made to the firebase database to the users who have swiped left on the logged in user. This is to prevent ineligible matches from appearing in the profile cards. The last thing is a query is made to firestore database using geohashing algorithm to find all users within a 500km radius excluding user's that have already been swiped on and users that have swiped left on the logged in user. The final result is presented in the HomeView as profile cards.

Profile Cards

Once geohashing has mapped the users around you within the 500km radius the cards are then generated excluding the logged in user has already swiped on. The profile cards are the representation of the users that are near you. The profile cards display information about a user including name, bio and location.

The profile cards use a CardStack library and I have implemented similarly to geohashing by adding the library in manually. This was because I made some adjustments for knowing whether the user tapped like or dislike on the card. Everything else about the library worked well.

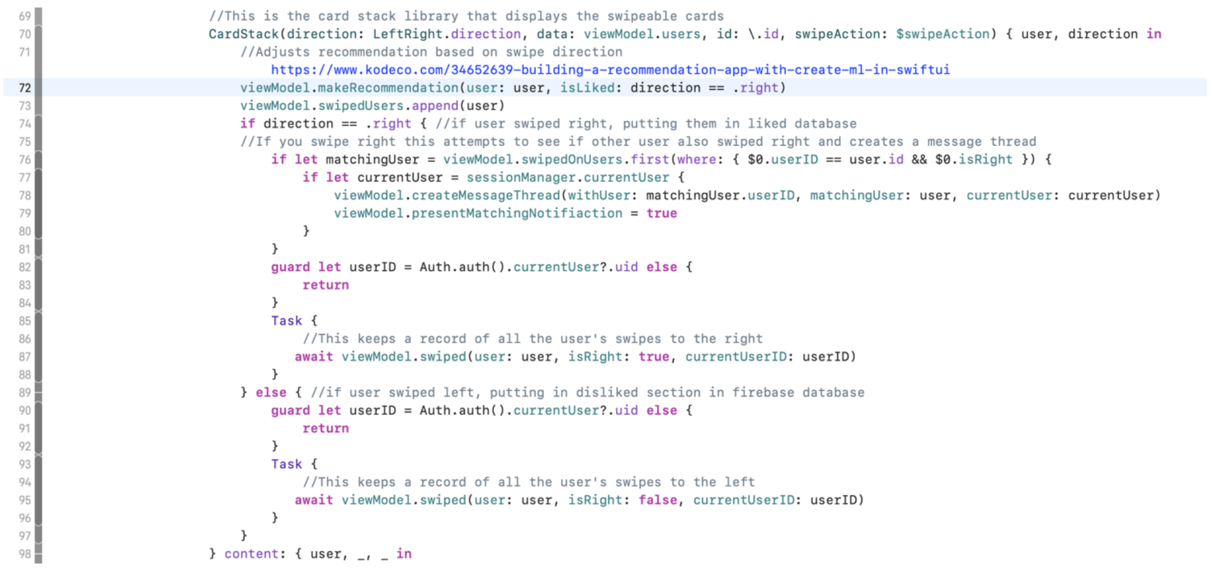
The CardStack library fires off a function every time a card is swiped. When this function is fired off three things happen.

1. This swipe is fed into the recommendation algorithm.

2. I am observing if to see if other users have equally swiped right on that user.

3. It actively stores all the swipes made by the user to avoid the same card appearing again.

Once a user taps onto the profile card they will be brought to the CardDetailView where more information about the user will be displayed.

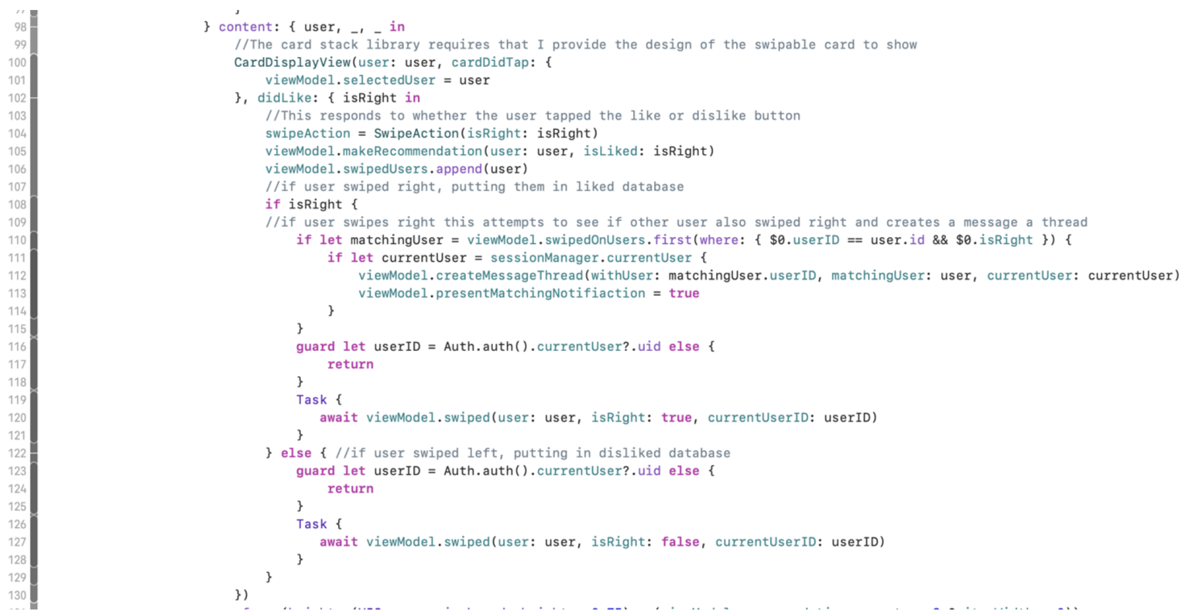


```
69  //This is the card stack library that displays the swipeable cards
70  CardStack(direction: LeftRight.direction, data: viewModel.users, id: \.id, swipeAction: $swipeAction) { user, direction in
71      //Adjusts recommendation based on swipe direction
            https://www.kodeco.com/34652639-building-a-recommendation-app-with-create-ml-in-swiftui
72      viewModel.makeRecommendation(user: user, isLiked: direction == .right)
73      viewModel.swipedUsers.append(user)
74      if direction == .right { //if user swiped right, putting them in liked database
75      //If you swipe right this attempts to see if other user also swiped right and creates a message thread
76          if let matchingUser = viewModel.swipedOnUsers.first(where: { $0.userID == user.id && $0.isRight }) {
77              if let currentUser = sessionManager.currentUser {
78                  viewModel.createMessageThread(withUser: matchingUser.userID, matchingUser: user, currentUser: currentUser)
79                  viewModel.presentMatchingNotifiaction = true
80              }
81          }
82          guard let userID = Auth.auth().currentUser?.uid else {
83              return
84          }
85          Task {
86              //This keeps a record of all the user's swipes to the right
87             await viewModel.swiped(user: user, isRight: true, currentUserID: userID)
88          }
89      } else { //if user swiped left, putting in disliked section in firebase database
90          guard let userID = Auth.auth().currentUser?.uid else {
91              return
92          }
93          Task {
94              //This keeps a record of all the user's swipes to the left
95             await viewModel.swiped(user: user, isRight: false, currentUserID: userID)
96          }
97      }
98  } content: { user, _, _ in
```

Figure 2.3.8

This screenshot displays the implementation of the CardStack in the HomeView. Line 72 calls a function in the ViewModel that passes in information about the swipe and the person that was swiped on into the recommendation algorithm. Lines 76-81 attempts to find if the user that was swiped right on equally swiped on the logged in user. If there is a matching swipe a new message thread is created and a notification is presented.

From lines 82-97 saves the swiped on user into firebase database so that the logged in user doesn’t see them again when they relaunch the app.



```
98  } content: { user, _, _ in
99      //The card stack library requires that I provide the design of the swipable card to show
100     CardDisplayView(user: user, cardDidTap: {
101         viewModel.selectedUser = user
102     }, didLike: { isRight in
103         //This responds to whether the user tapped the like or dislike button
104         swipeAction = SwipeAction(isRight: isRight)
105         viewModel.makeRecommendation(user: user, isLiked: isRight)
106         viewModel.swipedUsers.append(user)
107         //if user swiped right, putting them in liked database
108         if isRight {
109         //if user swipes right this attempts to see if other user also swiped right and creates a message a thread
110             if let matchingUser = viewModel.swipedOnUsers.first(where: { $0.userID == user.id && $0.isRight }) {
111                 if let currentUser = sessionManager.currentUser {
112                     viewModel.createMessageThread(withUser: matchingUser.userID, matchingUser: user, currentUser: currentUser)
113                     viewModel.presentMatchingNotifiaction = true
114                 }
115             }
116             guard let userID = Auth.auth().currentUser?.uid else {
117                 return
118             }
119             Task {
120                await viewModel.swiped(user: user, isRight: true, currentUserID: userID)
121             }
122         } else { //if user swiped left, putting in disliked database
123             guard let userID = Auth.auth().currentUser?.uid else {
124                 return
125             }
126             Task {
127                await viewModel.swiped(user: user, isRight: false, currentUserID: userID)
128             }
129         }
130     })
```

Figure 2.3.9

This is the second part of the CardStack implementation. This displays the card itself. The CardStack implementation lines 98-130 provides a function which is where I provided details of how the cards should look like. I added a view called CardDisplayView which maps the design of the card and also provides the functionality of what happens when you tap on like or dislike button. Lines 102-130 look very similar to the code used in the upper part of the CardStack implementation in figure 2.3.8 which is lines 74-97. The reason for this is the code for like or dislike button simulates the same functionality as swipe right or left.

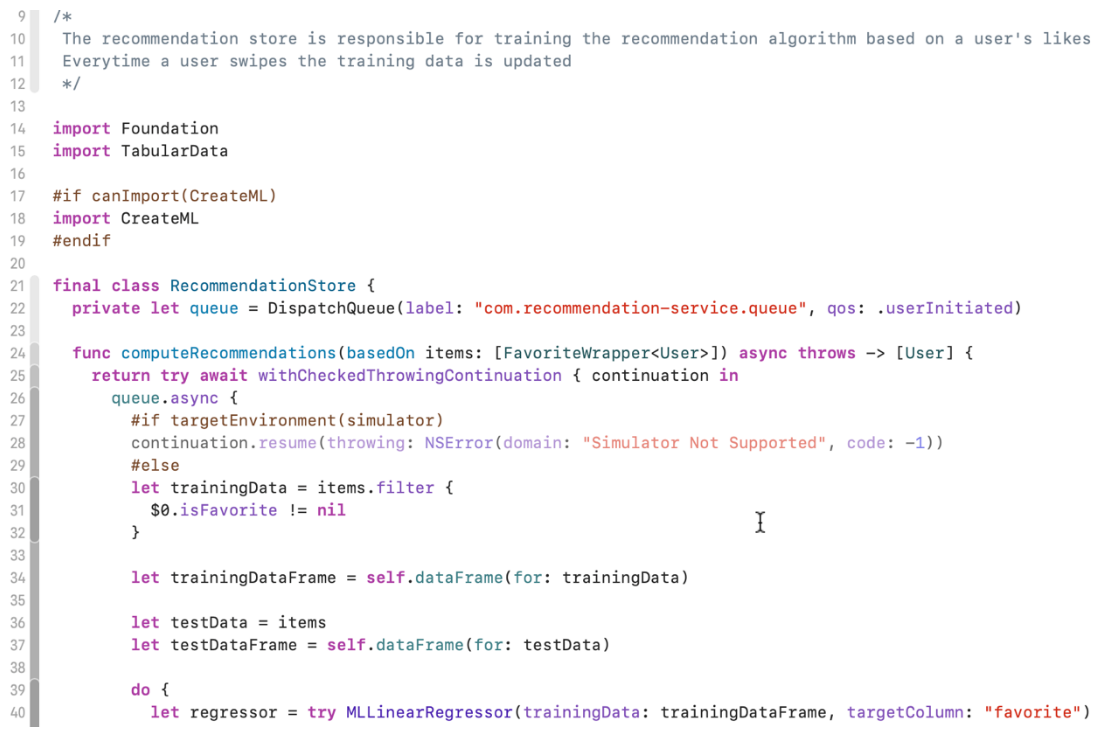
Recommendation Algorithm

A potential concern with the app was that you had to swipe through a lot of users in order to find your match. I wanted to help users find their matches much faster and I decided to implement a recommendation algorithm.

When researching the different recommendation algorithms, I came across a few of them. Collaborative filtering is a recommendation algorithm that gives recommendations based on the user and other users with similar preferences. I didn’t use this because it requires pooling in other user’s preferences and I had a limited dataset to work with. Content based filtering is based on creating recommendations on the similarity of the items. The reason I didn’t use it because the items are the users themselves and they are fairly homogeneous. Linear Regression is making recommendations on the basis of predicting a continuous quantity. I ended up considering linear regression because I am able to create recommendations on the history of a single user’s swipe preferences.

The ideal way to implement the linear regression algorithm would be to have it to work in real time and adjust as the user makes their swipes. I discovered that Apple provides a machine learning framework called CreateML with the ability to train models using linear regression. CreateML is able to conveniently train these models in real time and improve as more data is fed to it.

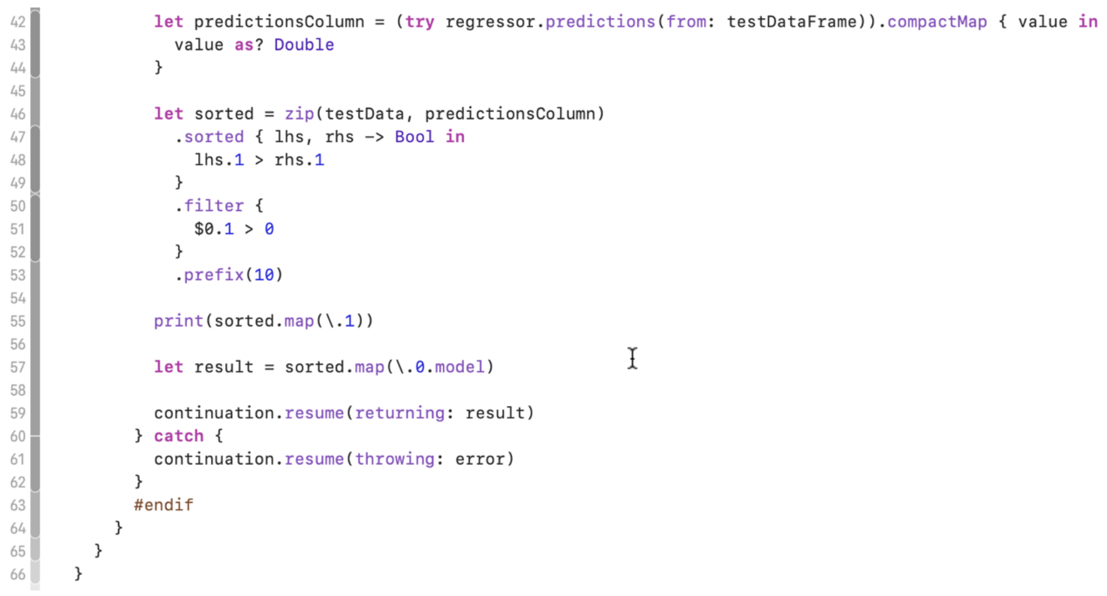
I found a suitable example of how to implement this on a tutorial on Kodeco website called TShirtFinder. The tutorial was about creating a swipe feature on t-shirts which is very similar to my application. I took the example from the tutorial an implemented it to work for my own.



```
 9  /*
10   The recommendation store is responsible for training the recommendation algorithm based on a user's likes
11   Everytime a user swipes the training data is updated
12   */
13
14  import Foundation
15  import TabularData
16
17  #if canImport(CreateML)
18  import CreateML
19  #endif
20
21  final class RecommendationStore {
22    private let queue = DispatchQueue(label: "com.recommendation-service.queue", qos: .userInitiated)
23
24    func computeRecommendations(basedOn items: [FavoriteWrapper<User>]) async throws -> [User] {
25      return try await withCheckedThrowingContinuation { continuation in
26        queue.async {
27          #if targetEnvironment(simulator)
28          continuation.resume(throwing: NSError(domain: "Simulator Not Supported", code: -1))
29          #else
30          let trainingData = items.filter {
31            $0.isFavorite != nil
32          }
33
34          let trainingDataFrame = self.dataFrame(for: trainingData)
35
36          let testData = items
37          let testDataFrame = self.dataFrame(for: testData)
38
39          do {
40            let regressor = try MLLinearRegressor(trainingData: trainingDataFrame, targetColumn: "favorite")
```

Figure 2.3.10

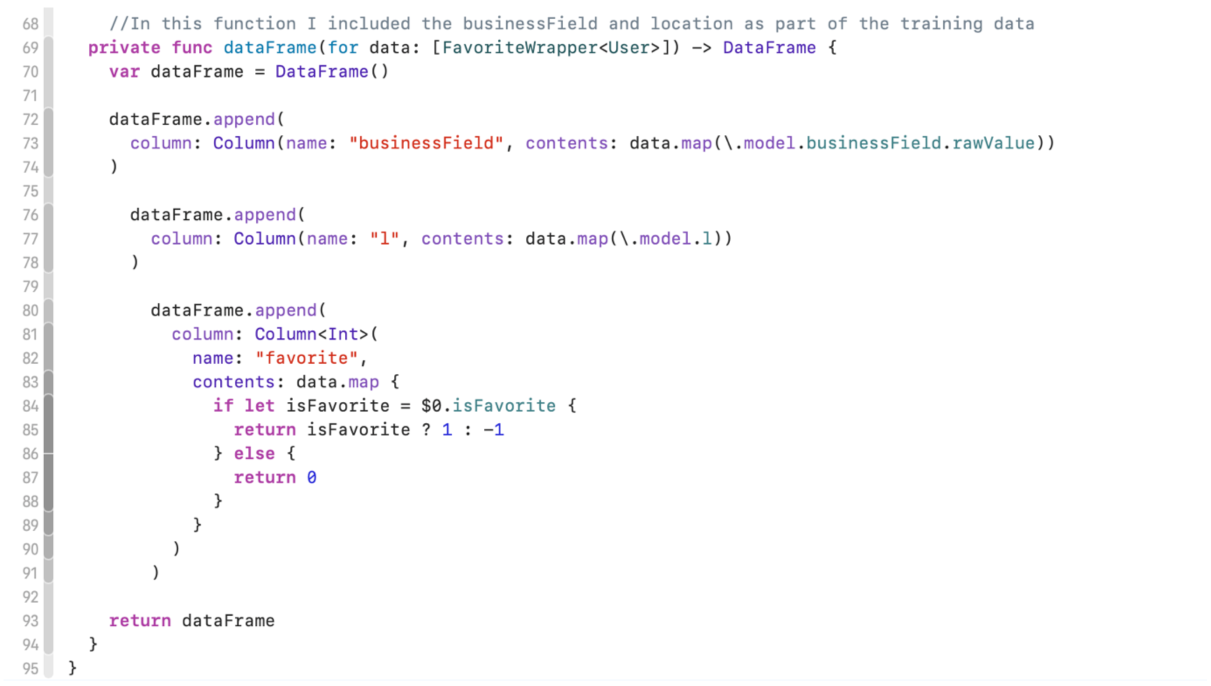
In the figure above I have a class called 'RecommendationStore' where the training of the data happens and the recommendations are made. On lines 30-37 I prepared the training data based on the people the user swiped on. On line 40 I feed that training data into a machine learning regressor provided by CreateML to make predictions on suitable recommendations.



```
42          let predictionsColumn = (try regressor.predictions(from: testDataFrame)).compactMap { value in
43            value as? Double
44          }
45
46          let sorted = zip(testData, predictionsColumn)
47            .sorted { lhs, rhs -> Bool in
48              lhs.1 > rhs.1
49            }
50            .filter {
51              $0.1 > 0
52            }
53            .prefix(10)
54
55          print(sorted.map(\.1))
56
57          let result = sorted.map(\.0.model)
58
59          continuation.resume(returning: result)
60        } catch {
61          continuation.resume(throwing: error)
62        }
63        #endif
64      }
65    }
66  }
```

Figure 2.3.11

On lines 42-66 I make predictions on the outcome of the machine learning algorithm.



```
  //In this function I included the businessField and location as part of the training data
 private func dataFrame(for data: [FavoriteWrapper<User>]) -> DataFrame {
  var dataFrame = DataFrame()

  dataFrame.append(
   column: Column(name: "businessField", contents: data.map(\.model.businessField.rawValue))
  )

   dataFrame.append(
    column: Column(name: "l", contents: data.map(\.model.l))
   )

    dataFrame.append(
     column: Column<Int>(
      name: "favorite",
      contents: data.map {
       if let isFavorite = $0.isFavorite {
        return isFavorite ? 1 : -1
       } else {
        return 0
       }
      }
     )
    )

  return dataFrame
 }
}
```

Figure 2.3.12

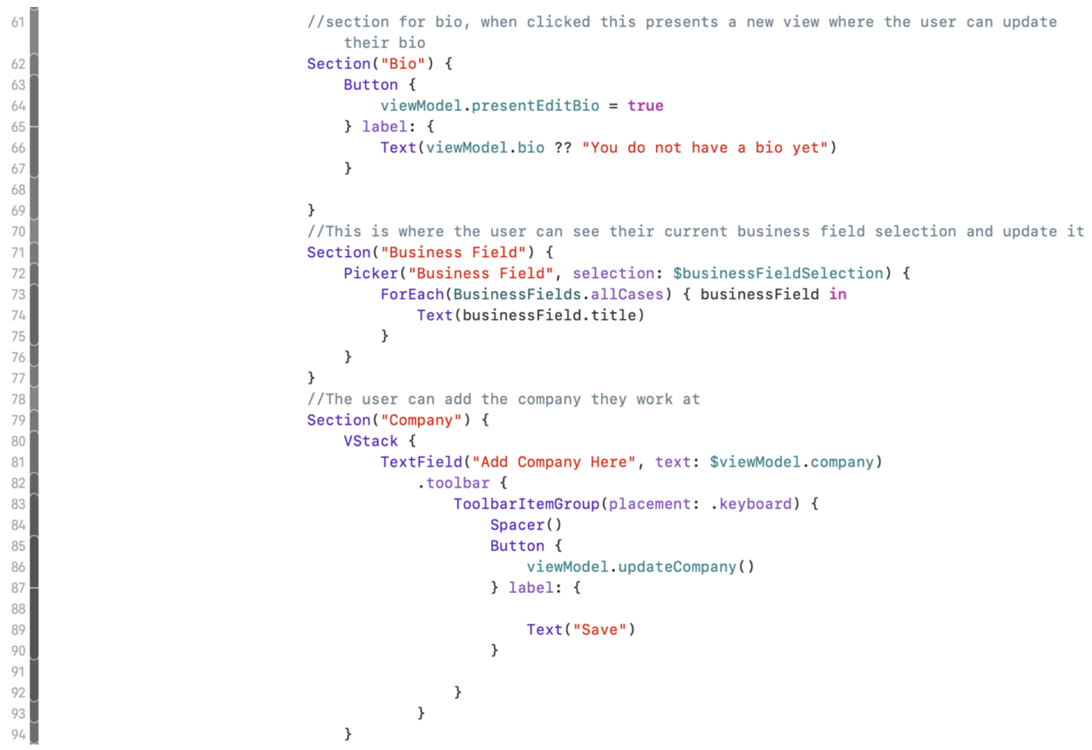
The dataFrame function is used in the earlier function computeRecommendations shown on figure 2.3.10 on line 24. The dataFrame function selects relevant fields to include in the linear regression algorithm. I have included the businessField, location and whether the user has liked or disliked as relevant fields for the linear regression algorithm.

The final result is I have created three boxes at the top of the swipe card of the top three recommendations which are updated each time the user swipes. See figure 2.4.4. A limitation of the CreateML is that it cannot run on a simulator only on a real device as a result I was not able to do testing on a simulator.

Profile

Within the profile section users have the ability to personalize their profile. The profile customisation directly influences how the profile cards are shown within the app. Users can edit and update their avatar, bio, business field, company and experience. The avatar feature utilises Google Cloud Storage, the avatars are uploaded and then viewed from the cloud.

Additionally users have the ability to sign out and delete their accounts. The option to delete their account is important for a number of reasons, firstly it respects the user's right to deletion by giving them control over their personal data and the freedom to remove it if they choose. Secondly, it aligns with regulatory requirements. For example the CCPA (California Consumer Privacy Act) states that you have a right to account deletion. As of June 30th, 2022, Apple mandates that apps have available on their platform that there must be an option for users to delete their accounts. Fusion supports user autonomy and data privacy, which are highlighted by this requirement. By offering these features, the app makes sure that users feel safe and in charge of their personal data.



```swift
//section for bio, when clicked this presents a new view where the user can update
    their bio
Section("Bio") {
    Button {
        viewModel.presentEditBio = true
    } label: {
        Text(viewModel.bio ?? "You do not have a bio yet")
    }

}
//This is where the user can see their current business field selection and update it
Section("Business Field") {
    Picker("Business Field", selection: $businessFieldSelection) {
        ForEach(BusinessFields.allCases) { businessField in
            Text(businessField.title)
        }
    }
}
//The user can add the company they work at
Section("Company") {
    VStack {
        TextField("Add Company Here", text: $viewModel.company)
            .toolbar {
                ToolbarItemGroup(placement: .keyboard) {
                    Spacer()
                    Button {
                        viewModel.updateCompany()
                    } label: {

                        Text("Save")
                    }

                }
            }
    }
```

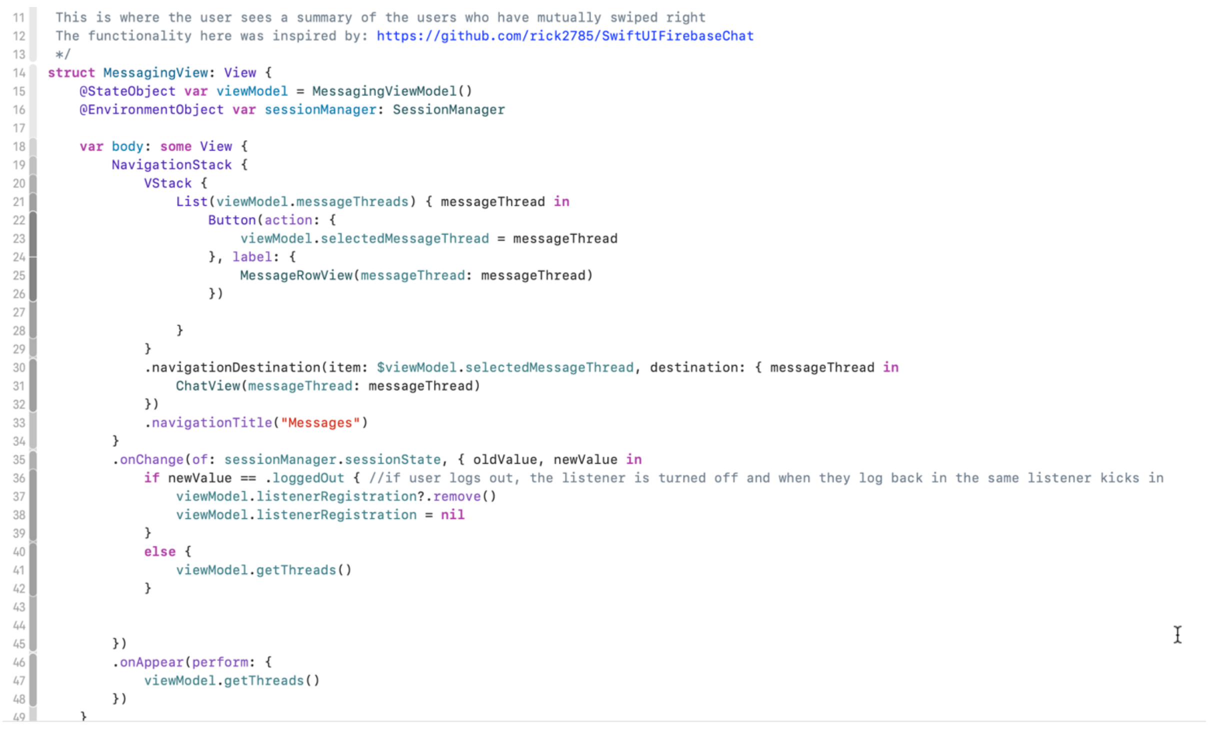
Figure 2.3.13

Figure 2.3.13 is an example of a few of the fields located in the profile section. In this figure you can see the bio, business field and company section.

Messaging

I added in messaging to facilitate communication once a connection is made. When a user swipes right on someone a message thread is automatically created allowing both users to start a conversation. Within the messaging section users can view who they have interacted with and access to their chat history. For inspiration and guidance I referred to the SwiftUIFirebaseChat library.

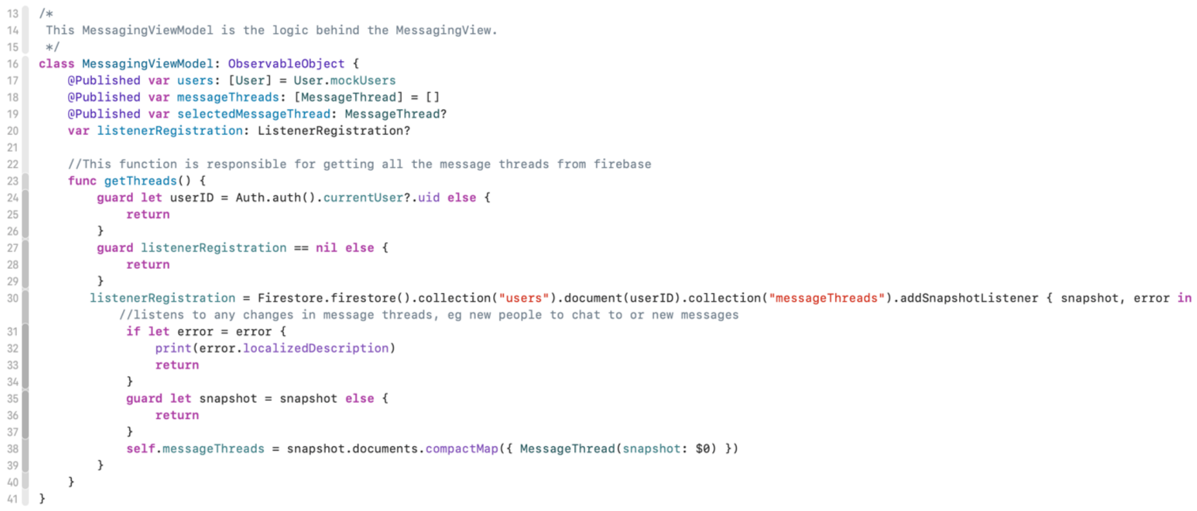
The implementation of messaging in Fusion is powered by Firestore. Firestore allows for real-time updates ensuring that messages are delivered and displayed instantly. This real-rime capability is essential for maintaining responsive communication between users.



```swift
 This is where the user sees a summary of the users who have mutually swiped right
 The functionality here was inspired by: https://github.com/rick2785/SwiftUIFirebaseChat
 */
struct MessagingView: View {
    @StateObject var viewModel = MessagingViewModel()
    @EnvironmentObject var sessionManager: SessionManager

    var body: some View {
        NavigationStack {
            VStack {
                List(viewModel.messageThreads) { messageThread in
                    Button(action: {
                        viewModel.selectedMessageThread = messageThread
                    }, label: {
                        MessageRowView(messageThread: messageThread)
                    })

                }
            }
            .navigationDestination(item: $viewModel.selectedMessageThread, destination: { messageThread in
                ChatView(messageThread: messageThread)
            })
            .navigationTitle("Messages")
        }
        .onChange(of: sessionManager.sessionState, { oldValue, newValue in
            if newValue == .loggedOut { //if user logs out, the listener is turned off and when they log back in the same listener kicks in
                viewModel.listenerRegistration?.remove()
                viewModel.listenerRegistration = nil
            }
            else {
                viewModel.getThreads()
            }


        })
        .onAppear(perform: {
            viewModel.getThreads()
        })
    }
```

Figure 2.3.14

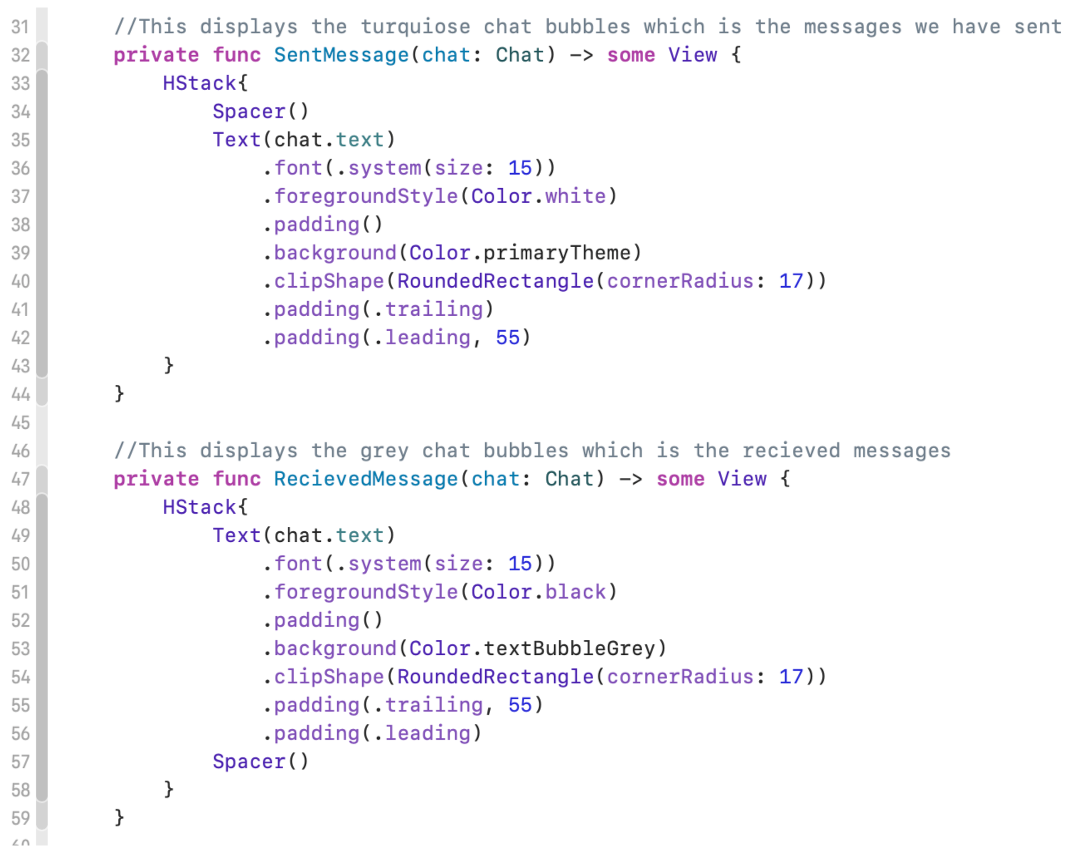
Line 21 lists out all the message threads. The message threads are retrieved from firestore and they each represent a user with whom you have matched with. When tapping on a message thread line 30 is triggered which takes you to the ChatView which contains the chat history.



```
13  /*
14   This MessagingViewModel is the logic behind the MessagingView.
15   */
16  class MessagingViewModel: ObservableObject {
17      @Published var users: [User] = User.mockUsers
18      @Published var messageThreads: [MessageThread] = []
19      @Published var selectedMessageThread: MessageThread?
20      var listenerRegistration: ListenerRegistration?
21  
22      //This function is responsible for getting all the message threads from firebase
23      func getThreads() {
24          guard let userID = Auth.auth().currentUser?.uid else {
25              return
26          }
27          guard listenerRegistration == nil else {
28              return
29          }
30         listenerRegistration = Firestore.firestore().collection("users").document(userID).collection("messageThreads").addSnapshotListener { snapshot, error in
               //listens to any changes in message threads, eg new people to chat to or new messages
31              if let error = error {
32                  print(error.localizedDescription)
33                  return
34              }
35              guard let snapshot = snapshot else {
36                  return
37              }
38              self.messageThreads = snapshot.documents.compactMap({ MessageThread(snapshot: $0) })
39          }
40      }
41  }
```

Figure 2.3.15

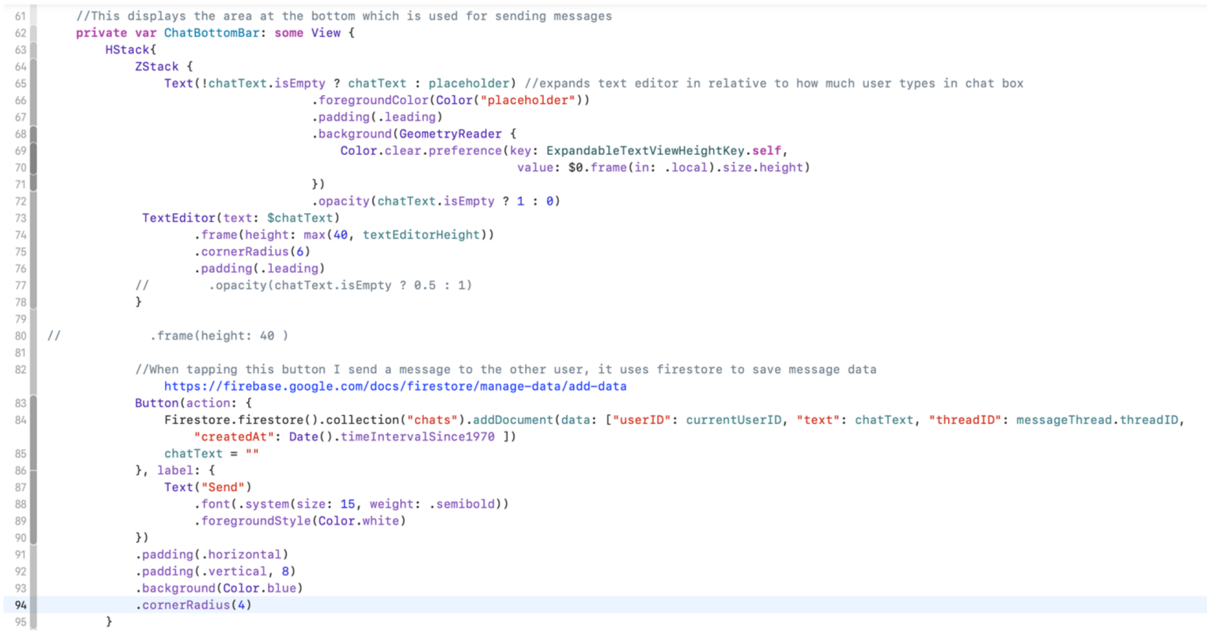
Figure 2.3.15 shows the getThreads function in more detail. The getThreads function is contained in the MessagingViewModel and this is responsible for getting all message threads as mentioned above. Line 30 is the most important line of the function as this makes a call to the firestore database, but it does in a way in which it listens for any new threads that are created and automatically adds them to the MessagingView.



```
31      //This displays the turquiose chat bubbles which is the messages we have sent
32      private func SentMessage(chat: Chat) -> some View {
33          HStack{
34              Spacer()
35              Text(chat.text)
36                  .font(.system(size: 15))
37                  .foregroundStyle(Color.white)
38                  .padding()
39                  .background(Color.primaryTheme)
40                  .clipShape(RoundedRectangle(cornerRadius: 17))
41                  .padding(.trailing)
42                  .padding(.leading, 55)
43          }
44      }
45
46      //This displays the grey chat bubbles which is the recieved messages
47      private func RecievedMessage(chat: Chat) -> some View {
48          HStack{
49              Text(chat.text)
50                  .font(.system(size: 15))
51                  .foregroundStyle(Color.black)
52                  .padding()
53                  .background(Color.textBubbleGrey)
54                  .clipShape(RoundedRectangle(cornerRadius: 17))
55                  .padding(.trailing, 55)
56                  .padding(.leading)
57              Spacer()
58          }
59      }
```

Figure 2.3.16

This screenshot is taken from the ChatView. It shows the two essential functions that are responsible for the chat bubbles. On line 32 the SentMessage function is responsible for creating chat bubbles that the logged in user has sent. Line 47 is responsible for displaying chat bubbles for the messages that have been received.



```
61      //This displays the area at the bottom which is used for sending messages
62      private var ChatBottomBar: some View {
63          HStack{
64              ZStack {
65                  Text(!chatText.isEmpty ? chatText : placeholder) //expands text editor in relative to how much user types in chat box
66                                      .foregroundColor(Color("placeholder"))
67                                      .padding(.leading)
68                                      .background(GeometryReader {
69                                          Color.clear.preference(key: ExpandableTextViewHeightKey.self,
70                                                                 value: $0.frame(in: .local).size.height)
71                                      })
72                                      .opacity(chatText.isEmpty ? 1 : 0)
73               TextEditor(text: $chatText)
74                      .frame(height: max(40, textEditorHeight))
75                      .cornerRadius(6)
76                      .padding(.leading)
77              //        .opacity(chatText.isEmpty ? 0.5 : 1)
78              }
79
80  //            .frame(height: 40 )
81
82              //When tapping this button I send a message to the other user, it uses firestore to save message data
                    https://firebase.google.com/docs/firestore/manage-data/add-data
83              Button(action: {
84                  Firestore.firestore().collection("chats").addDocument(data: ["userID": currentUserID, "text": chatText, "threadID": messageThread.threadID,
                        "createdAt": Date().timeIntervalSince1970 ])
85                  chatText = ""
86              }, label: {
87                  Text("Send")
88                      .font(.system(size: 15, weight: .semibold))
89                      .foregroundStyle(Color.white)
90              })
91              .padding(.horizontal)
92              .padding(.vertical, 8)
93              .background(Color.blue)
94              .cornerRadius(4)
95          }
```

Figure 2.3.17

This screenshot shows the ChatBottomBar variable which is responsible for creating a space at the bottom of the ChatView for the logged in user to create a message. Line 83 creates a button which when tapped sends a message the logged in user wrote to the firestore database.

2.4 Graphical User Interface (GUI)

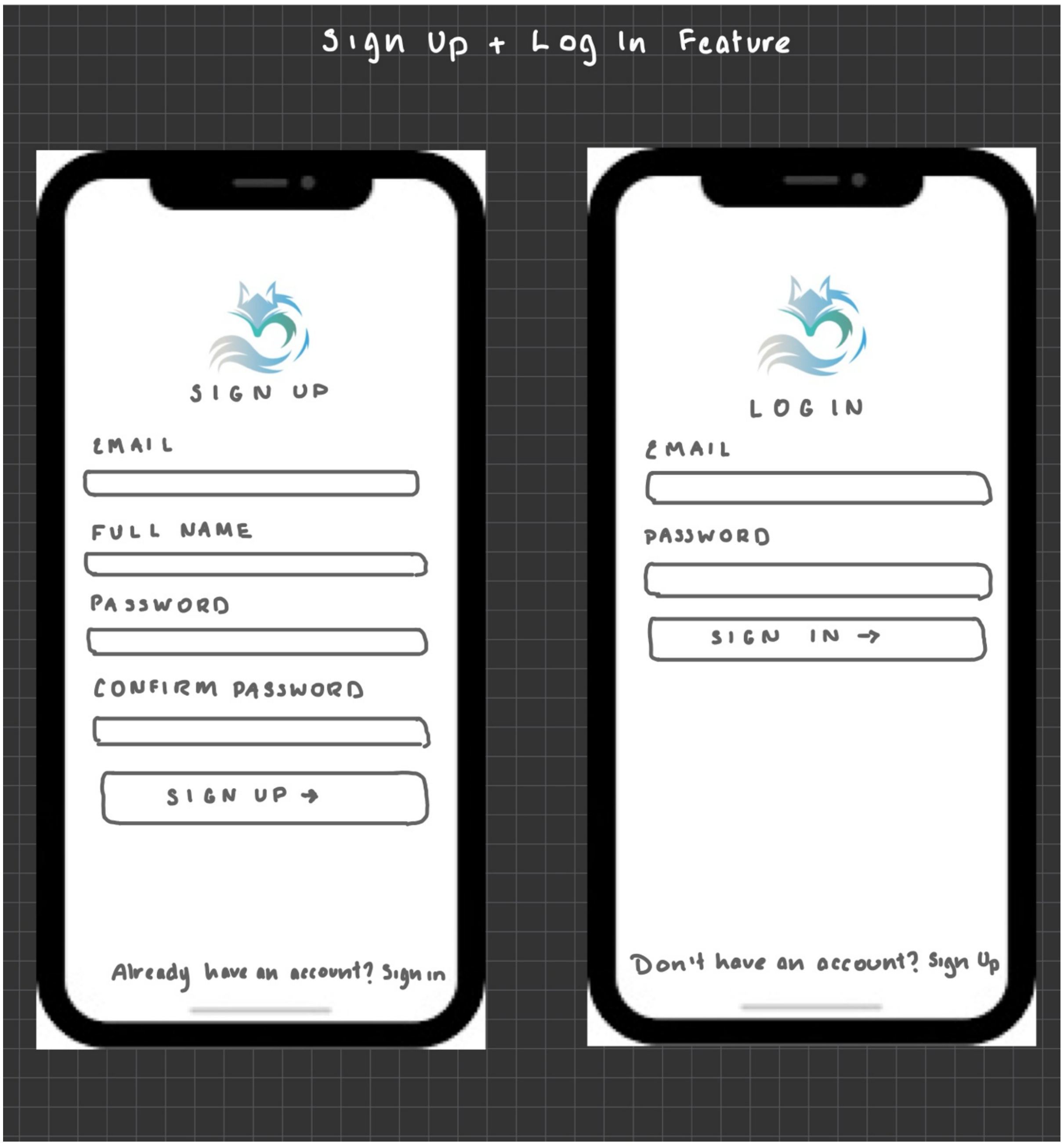
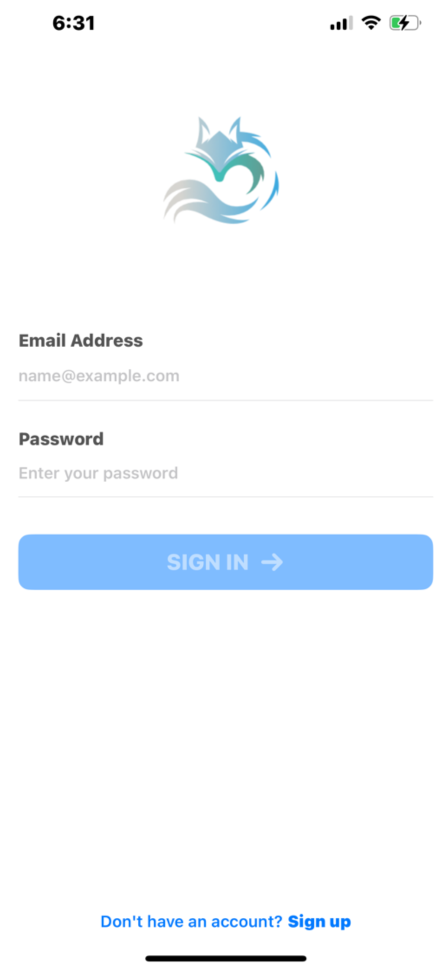


Figure 2.4.1 – Design for Sign Up & Log In feature

Above are the designs for Sign Up and Log In feature, the user will either be able to sign up with their details or log in with their credentials. At the bottom is a button where the user can click to sign in or sign up and they will be brought to the page.



6:31

Email Address

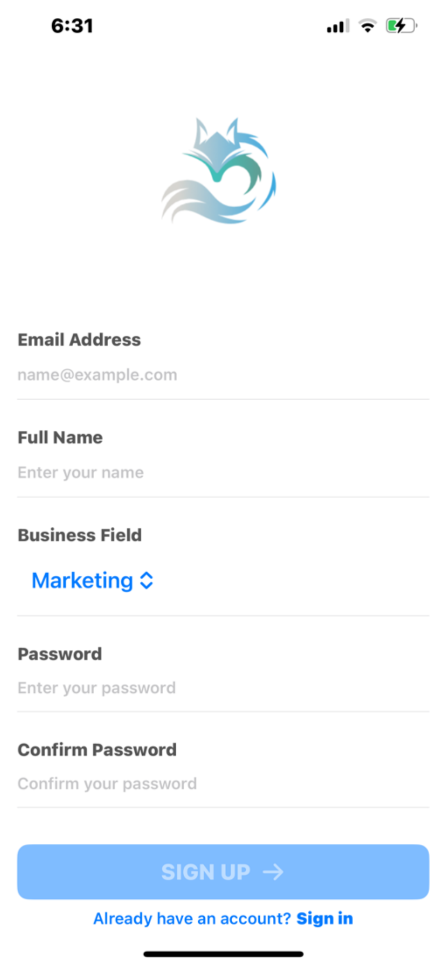
name@example.com

Password

Enter your password

SIGN IN →

Don't have an account? **Sign up**



6:31

Email Address

name@example.com

Full Name

Enter your name

Business Field

Marketing

Password

Enter your password

Confirm Password

Confirm your password

SIGN UP →

Already have an account? **Sign in**

Figure 2.4.2 – Authentication feature, log in and sign up from Fusion

Above in figure 2.4.2 is the authentication feature for Fusion. They correspond to my designs. A user can create an account with their corresponding details and then log in using their email and password.

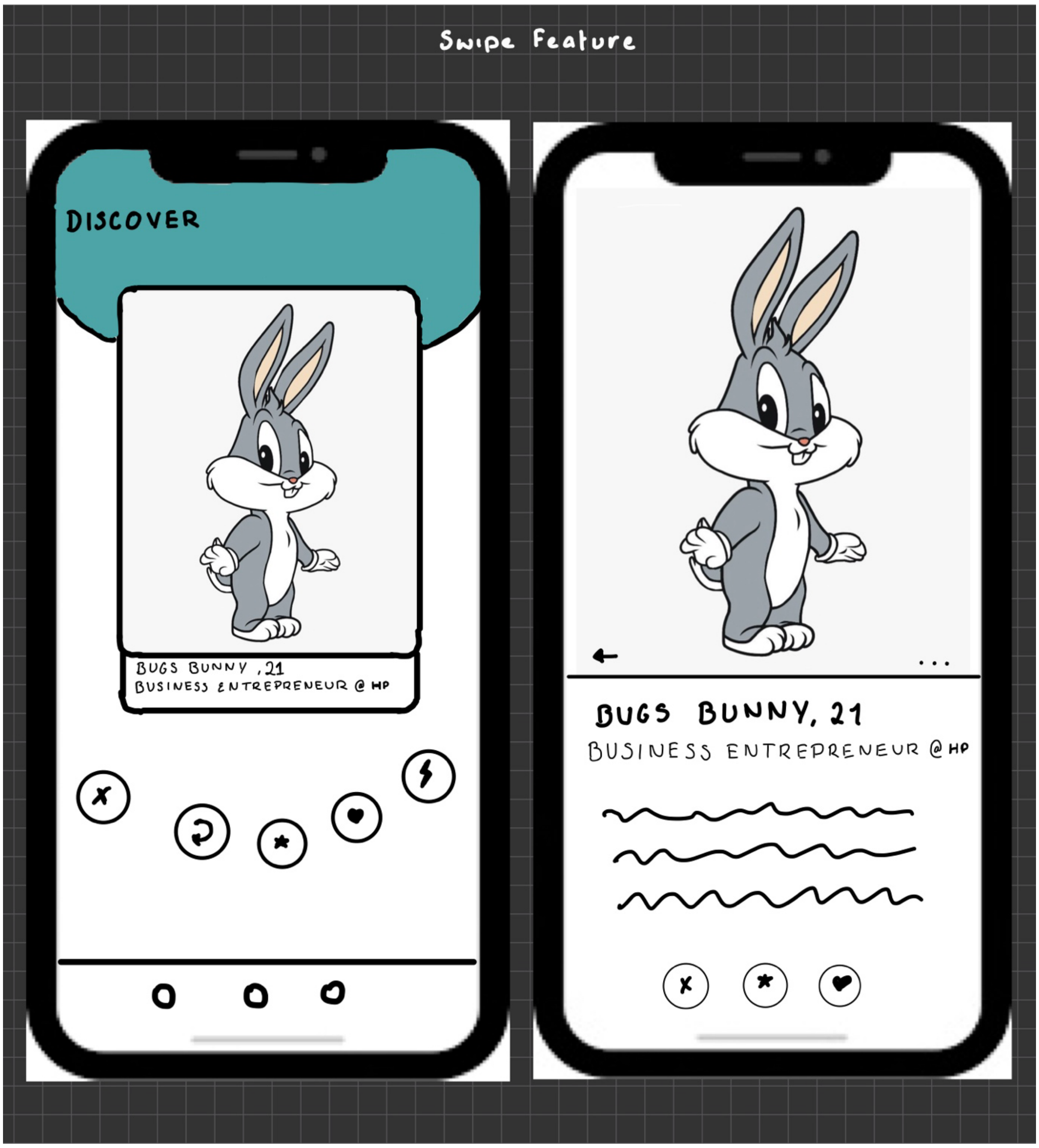
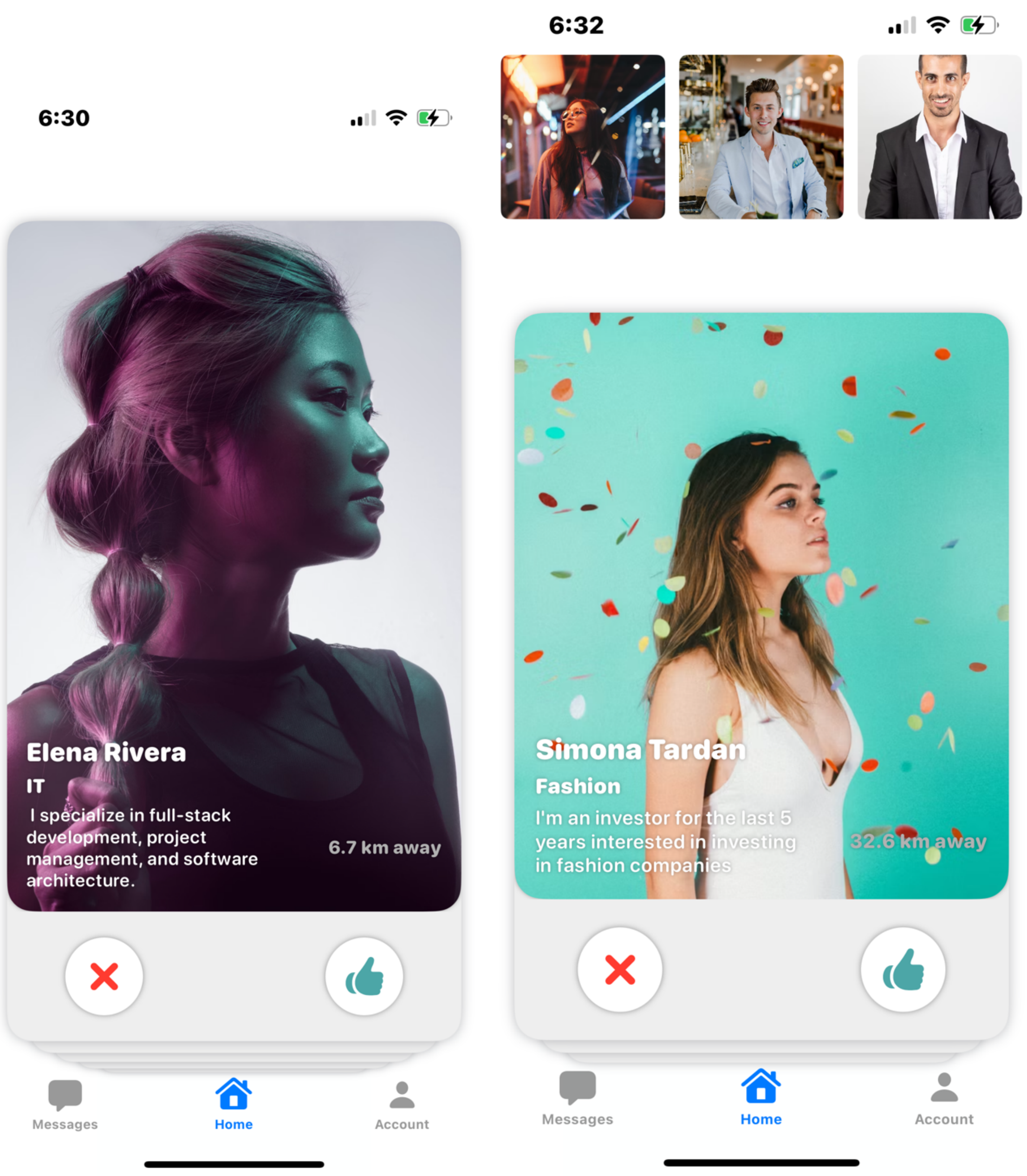


Figure 2.4.3 – Design for Profile Card and CardDetailView

The first picture is a design of the Swipe Feature that a user is presented with. The user will be able to swipe right or left based on their preferences. The user can then click on the profile and more details about the user can be shown such as business interests and experience.


6:30

Elena Rivera

IT

I specialize in full-stack development, project management, and software architecture.

6.7 km away

6:32

Simona Tardan

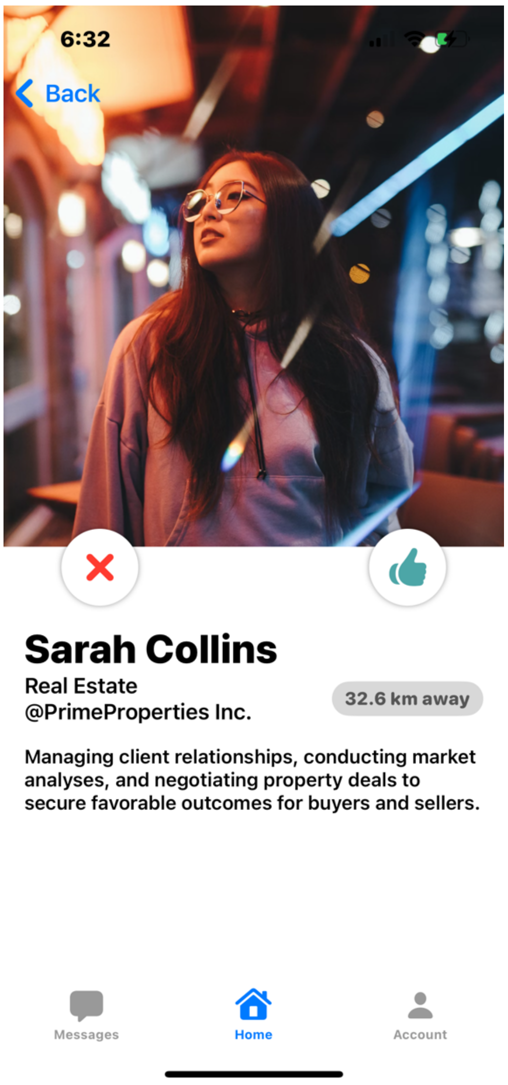
Fashion

I'm an investor for the last 5 years interested in investing in fashion companies

32.6 km away

Figure 2.4.4 – Swiping Feature UI

Figure 2.4.4 shows the swiping feature in the home section. The user is presented with a profile card. Within the profile card the user is able to see key details about the user such as name, business field, location and bio. Once the user starts swiping the user will begin to see recommendations pop up at the top of their screen.



Sarah Collins
Real Estate
@PrimeProperties Inc.
32.6 km away

Managing client relationships, conducting market analyses, and negotiating property deals to secure favorable outcomes for buyers and sellers.

Figure 2.4.5 – CardDetaiView shows more detailed information about user

The next screenshot is from the CardDetailView. The user can view more details about the user once they click on the profile card. Inside the CardDetailView the user is presented with the name, business field, company, location capsule and experience.

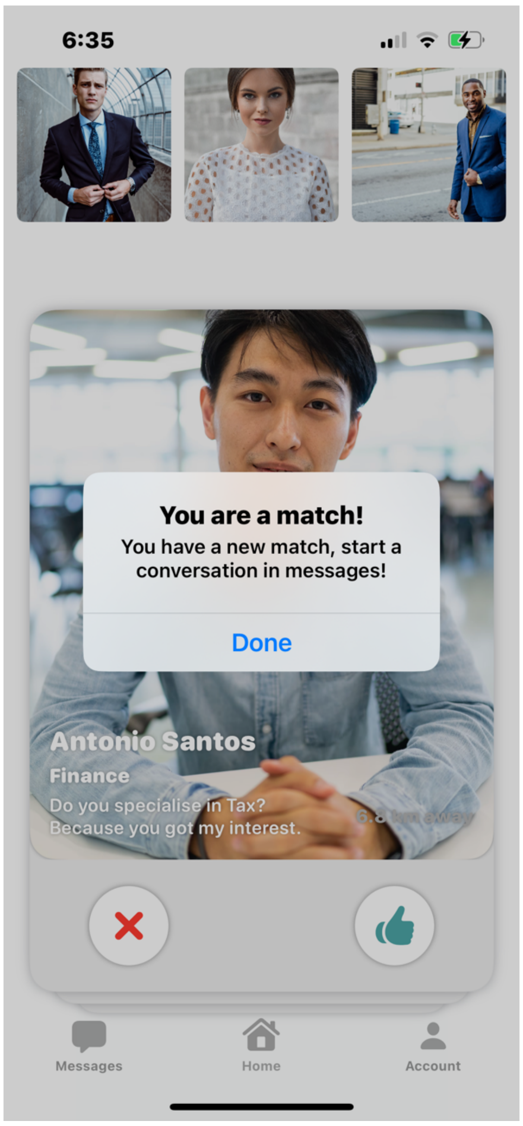
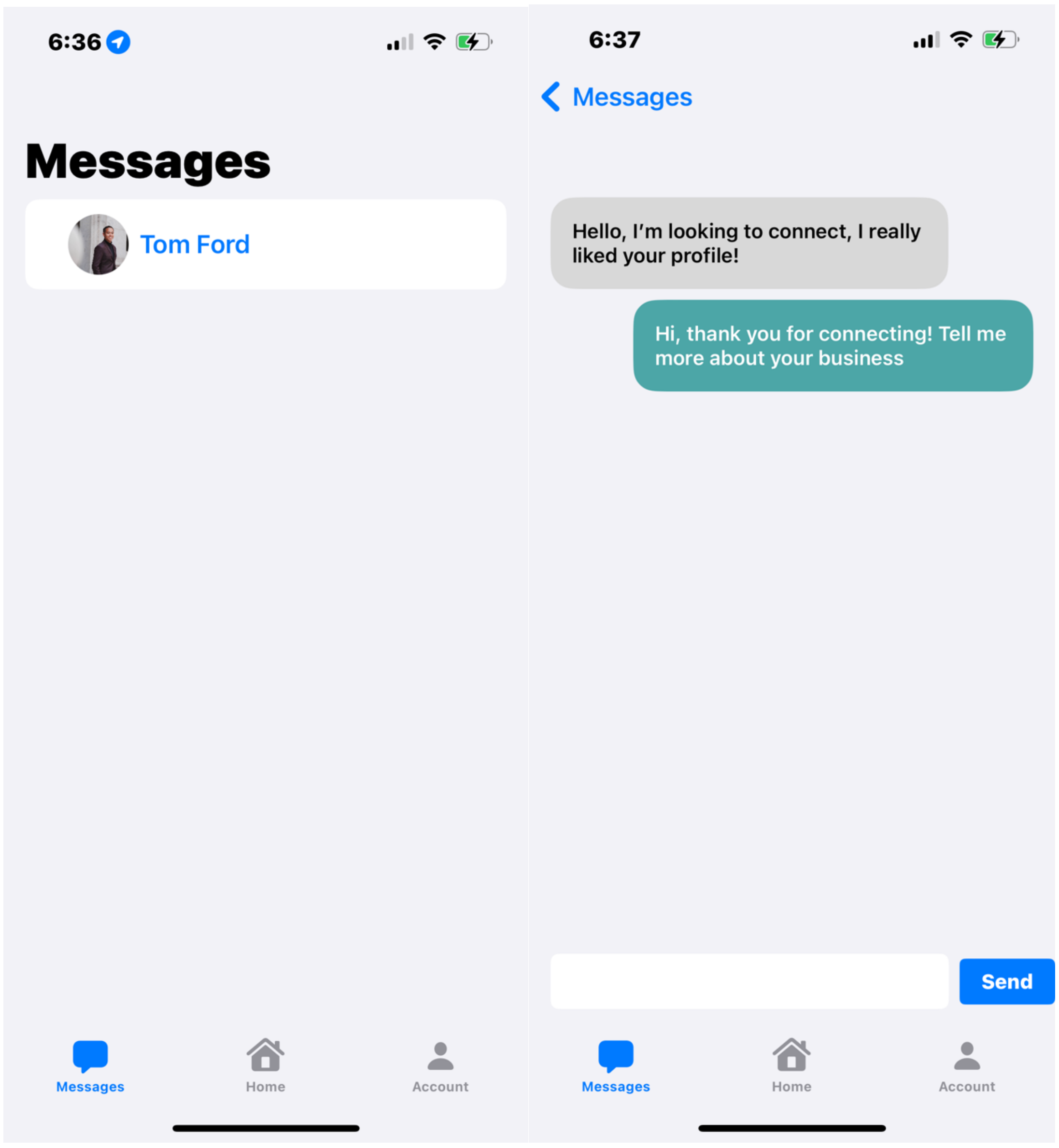

6:35
You are a match!
You have a new match, start a conversation in messages!
Done
Antonio Santos
Finance
Do you specialise in Tax? Because you got my interest.
Messages
Home
Account

Figure 2.4.6 – 'You are a match' pop shown from two users connecting

This is a pop up that comes up one two users have swiped right on each other. They will be notified with that they have a new match and can a new message thread will begin in the messages section.

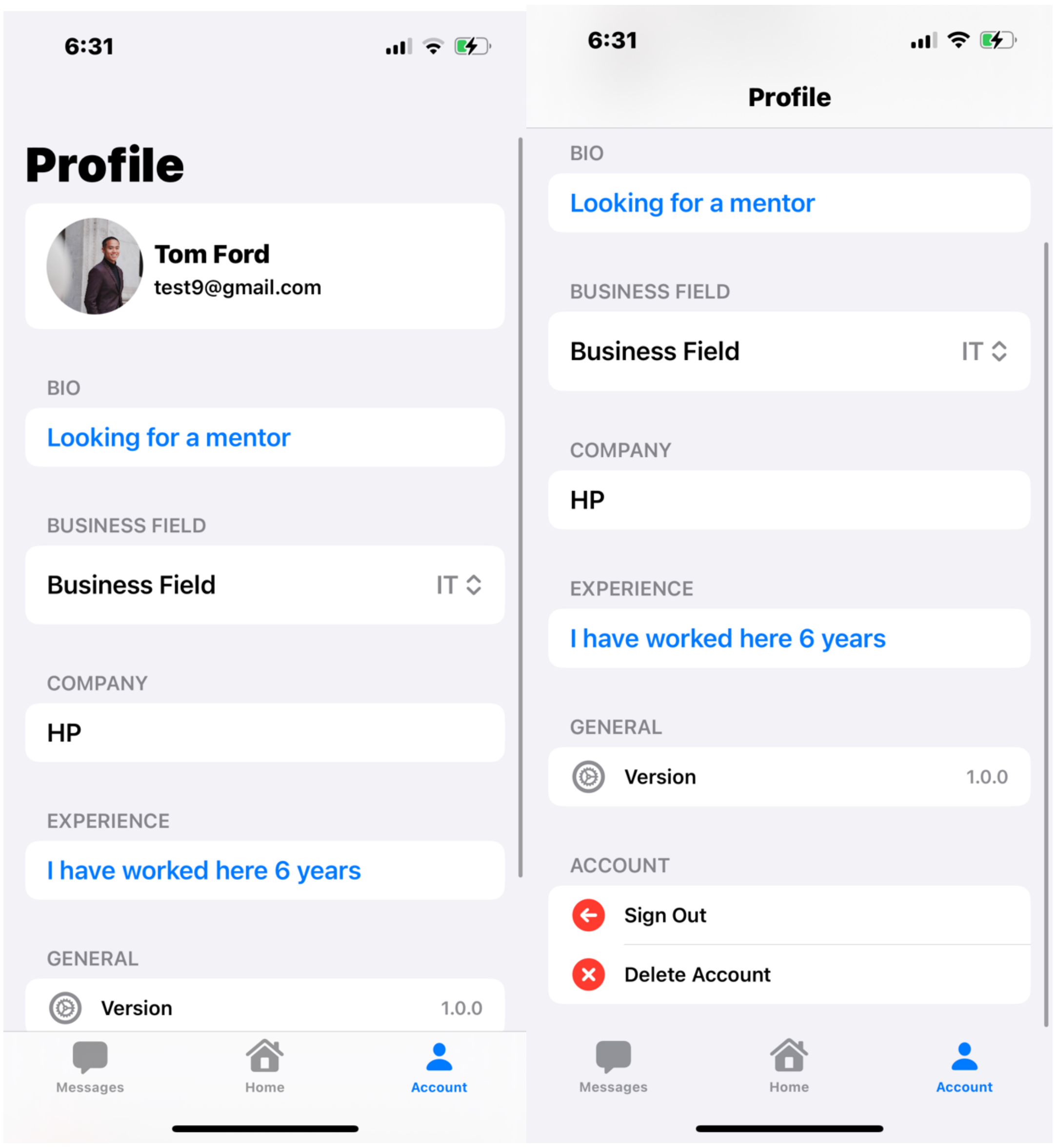


Hello, I'm looking to connect, I really liked your profile!

Hi, thank you for connecting! Tell me more about your business

Figure 2.4.7 – UIMessagingView shows the message thread and a conversation between two users

Figure 2.4.7 is a screenshot of the messages section on the left is the messages thread that has begun with the user matched. On the right is the message thread between two users.



6:31

Profile

Tom Ford
test9@gmail.com

BIO
Looking for a mentor

BUSINESS FIELD
Business Field — IT

COMPANY
HP

EXPERIENCE
I have worked here 6 years

GENERAL
Version — 1.0.0

ACCOUNT
Sign Out
Delete Account

Figure 2.4.8 – Profile View is displayed showing user's details

The image above represents the user's profile, they have the ability to customise it as they desire such as adding in their profile picture, their details and also having the ability to delete their account.

2.5 Testing

Testing is a crucial phase in the development process, ensuring that the app functions as intended and meets the defined requirements. For Fusion, I architected the project using the Model-View-ViewModel (MVVM) pattern, which is a software development approach designed by apple that defines the architecture of the app. MVVM separates the application into three main components: the Model, the View, and the ViewModel.

Model: Represents the data and business logic of the application.

View: Responsible for displaying the user interface and handling user interactions.

ViewModel: Acts as an intermediary between the Model and the View, handling data manipulation and preparing it for the View.

This architectural pattern offers several advantages:

1. Native to SwiftUI: MVVM is well-suited for SwiftUI, making it easier to develop and maintain the application.
2. Principle Architecture: The Apple team has made MVVM the principal architecture for SwiftUI. Apple ensures compatibility and support.
3. Ease of Testing: MVVM facilitates testing by allowing testing targets to focus on the ViewModel, which handles the business logic and data manipulation.
4. Scalability: MVVM is a predictable and well-understood structure this in turn helps with the scalability of the project, making it easier to manage as the Fusion's codebase grows.

Using the MVVM pattern simplifies the testing process, typically involving three types of testing Unit Testing, UI Testing and Performance Testing.

Unit Testing:

The first testing methodology I completed is Unit Testing. This is a process of testing individual source code components, like functions, methods, classes, and modules. Unit testing focuses on a 'unit' of code isolation giving a precise definition of correctness that every unit needs to stick to.

I began by creating corresponding tests at the same time as I developed each app feature. In my head these tests serve as the judges, determining whether the logic embedded in my views and view models works as it should. The view and view model have a key relationship; the view model contains the operation's logic, or "brains," while the view displays what the user sees. Comparing the view to the view model guarantees that changes to the underlying logic will not affect the user interface and vice versa. To choose the units critical for testing I focused on the components that handle core functionalities.

Initially, my testing resembled a rather confused Christmas tree. A flurry of red and green, where green signalled success and red, quite bluntly, meant I had to get back to the drawing board. It was a vivid display of trial and error as I refined my code. Every green light was a small victory, each red light was an opportunity to learn and improve.

UI Testing:

The second testing methodology I completed is UI Testing. The way UI Testing works is that it focuses on verifying that the user interface behaves as expected. UI Testing works by simulating user interactions with the app. Automated scripts are created to mimic the actions a real user would usually take, such as tapping various buttons, entering text, and even navigating through the app. These scripts then check the responses of the UI elements to confirm that they are displaying the correct information and responding appropriately to user input. For Fusion, I used Xcode's built-in UI Testing framework to create and run these automated tests. By doing these specific tests I was able to find any inconsistencies that was in my code, correcting the inconsistencies prevented a negative user experience.

Performance Testing:

The third testing methodology I employed is Performance Testing. This type of testing focuses on evaluating how well the application performs under various conditions.

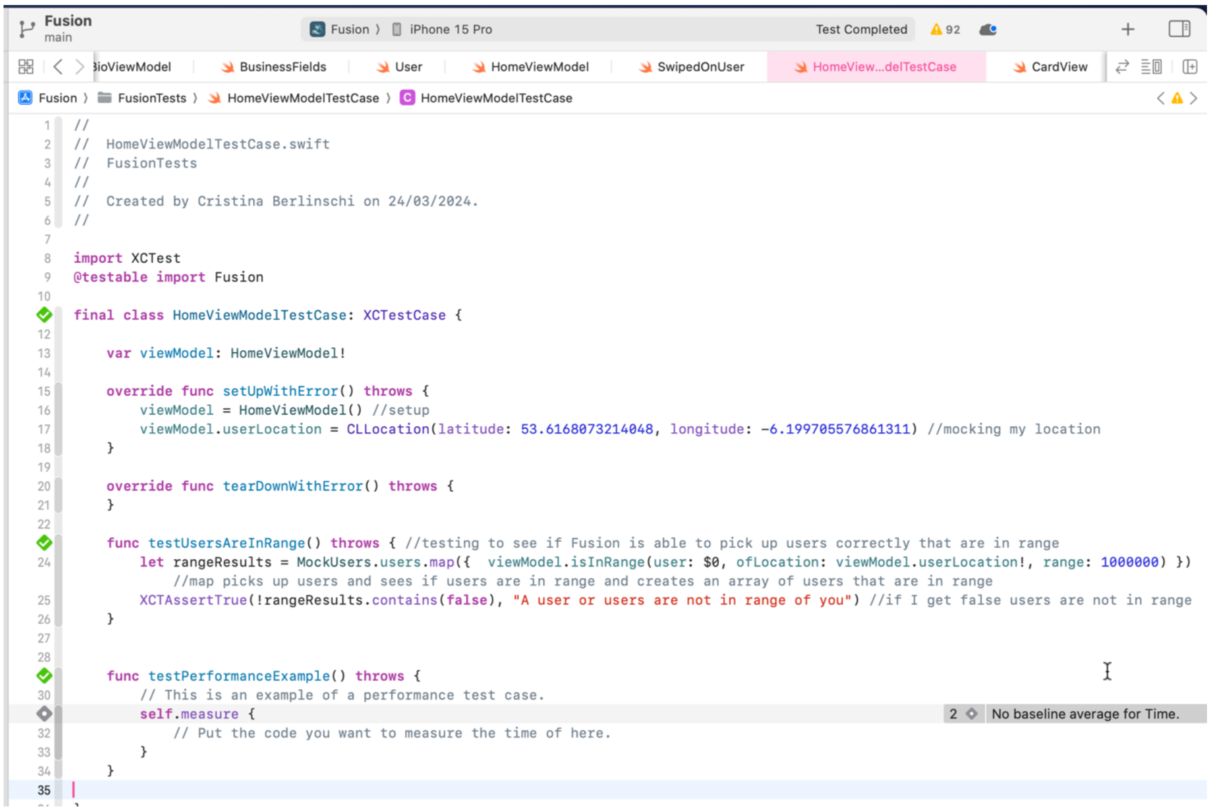
Performance testing encompasses three main key aspects: CPU usage, memory consumption, and speed analysis.

CPU Usage: This involves monitoring the amount of processing power the app requires. This is important because high CPU usage can lead to increased battery drain on phones and may cause Fusion to run slowly. For Fusion to have optimal performance, I profiled the app's CPU usage using Xcode’s Time Profiler instrument. This tool provides detailed insights into how the app's code executes, allowing me to identify and optimise any parts of the code that were excessively being used by the CPU.

Memory Consumption: Efficient memory management is essential for preventing the app from crashing or slowing down, especially on devices with limited resources. I used Xcode’s Memory Graph Debugger to monitor the app's memory usage and also identify memory leaks. This provides information that the app releases memory properly when it is no longer needed.

Speed Analysis: Speed analysis calculates how long it takes an application to perform different tasks, like loading screens, processing data, and loading user interface elements. I used a debugging method by writing code and assessed the time by logging the start time of process, end time of the process and these two were taken away from each other and resulted in the speed of the function.

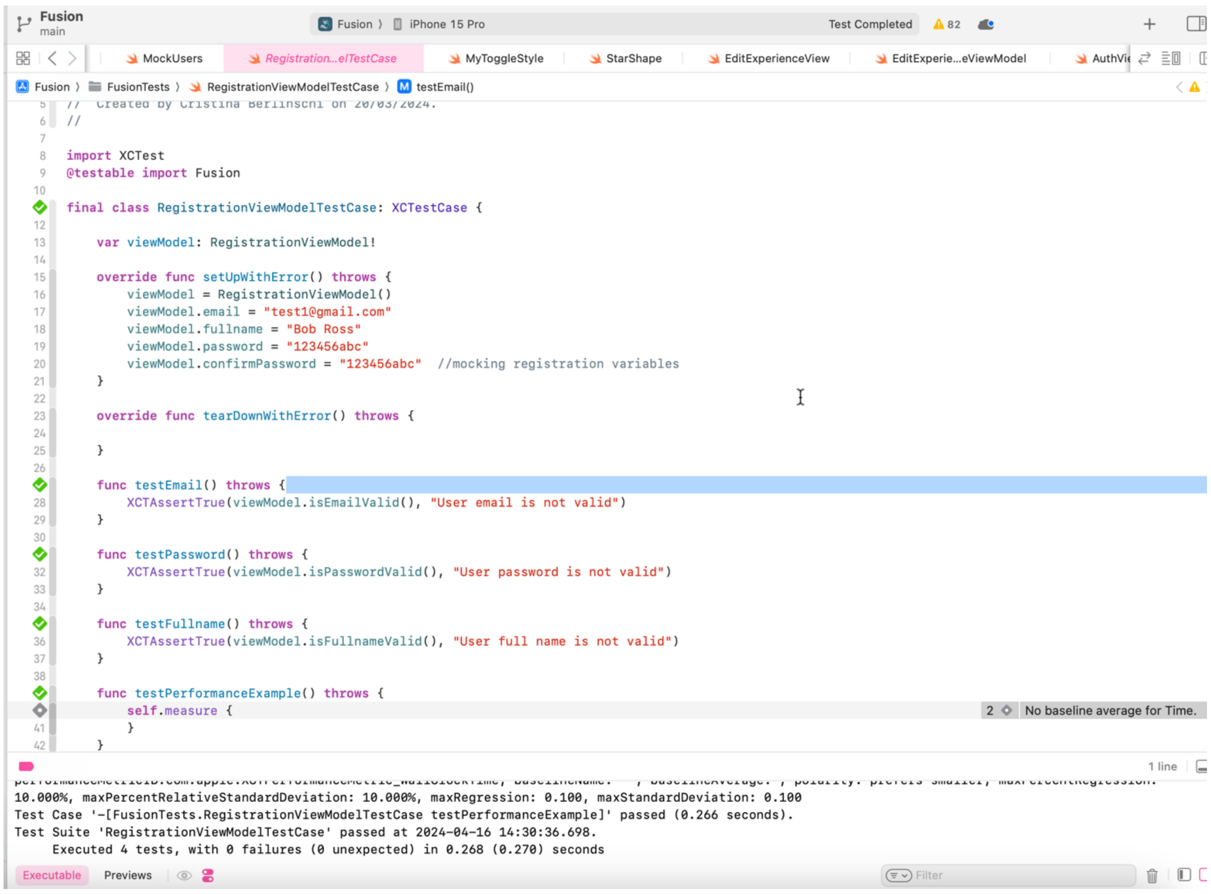
Through the long process of testing CPU usage, memory consumption, and speed profiling, I managed to get a detailed overview of the application's performance. I was able to address issues before they could effect user experience.



```swift
//
//  HomeViewModelTestCase.swift
//  FusionTests
//
//  Created by Cristina Berlinschi on 24/03/2024.
//

import XCTest
@testable import Fusion

final class HomeViewModelTestCase: XCTestCase {

    var viewModel: HomeViewModel!

    override func setUpWithError() throws {
        viewModel = HomeViewModel() //setup
        viewModel.userLocation = CLLocation(latitude: 53.6168073214048, longitude: -6.199705576861311) //mocking my location
    }

    override func tearDownWithError() throws {
    }

    func testUsersAreInRange() throws { //testing to see if Fusion is able to pick up users correctly that are in range
        let rangeResults = MockUsers.users.map({  viewModel.isInRange(user: $0, ofLocation: viewModel.userLocation!, range: 1000000) })
            //map picks up users and sees if users are in range and creates an array of users that are in range
        XCTAssertTrue(!rangeResults.contains(false), "A user or users are not in range of you") //if I get false users are not in range
    }


    func testPerformanceExample() throws {
        // This is an example of a performance test case.
        self.measure {                                                                   2  No baseline average for Time.
            // Put the code you want to measure the time of here.
        }
    }

```

Figure 2.5.1 – Unit Test ‘HomeViewModelTestCase’

In the ‘HomeViewModelTestCase’ figure 2.5.1, I am testing the feature that ensures users are within proximity to be introduced into the array of users that are in range. I mocked my location at the beginning of the test. The function ‘testUsersAreInRange’ uses this and compares it against the MockUsers to see if the users are in range. As you can see on the left-hand side the lights on my Christmas tree shone all green.

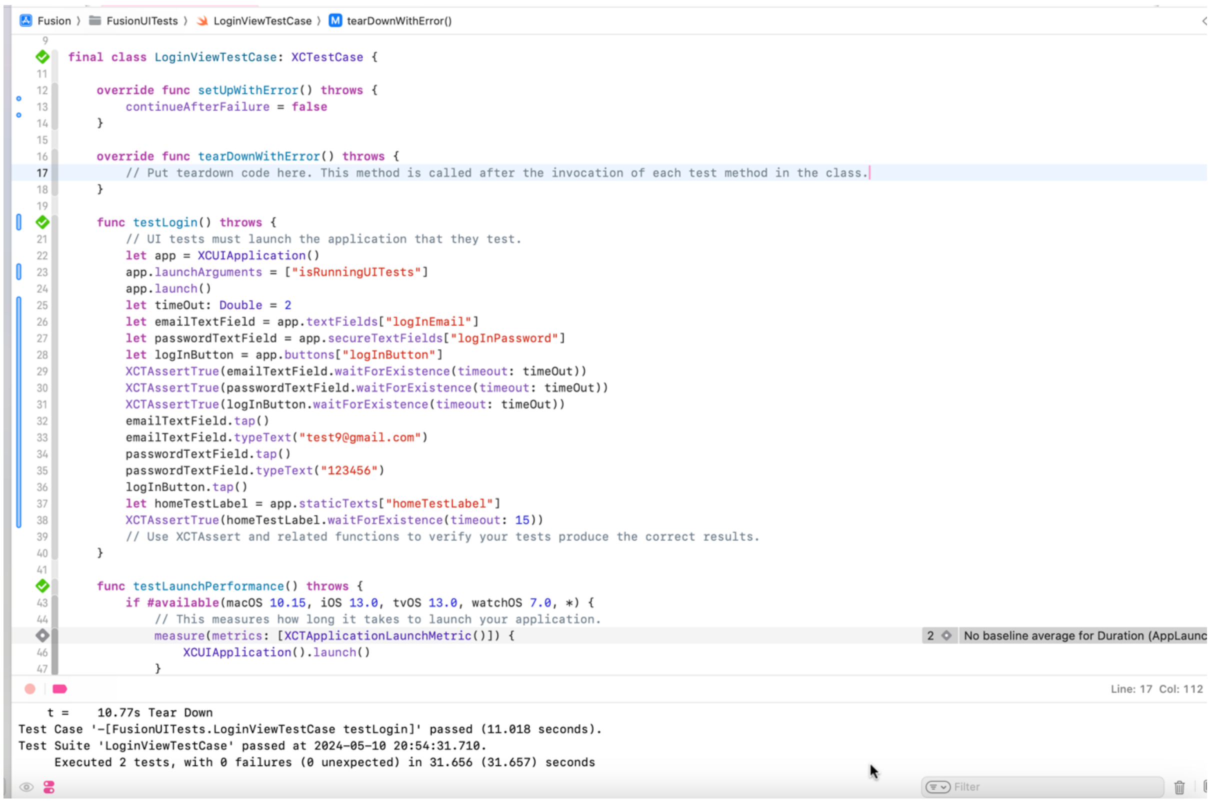


```swift
//  Created by Cristina Berlinschi on 20/03/2024.
//

import XCTest
@testable import Fusion

final class RegistrationViewModelTestCase: XCTestCase {

    var viewModel: RegistrationViewModel!

    override func setUpWithError() throws {
        viewModel = RegistrationViewModel()
        viewModel.email = "test1@gmail.com"
        viewModel.fullname = "Bob Ross"
        viewModel.password = "123456abc"
        viewModel.confirmPassword = "123456abc"  //mocking registration variables
    }

    override func tearDownWithError() throws {

    }

    func testEmail() throws {
        XCTAssertTrue(viewModel.isEmailValid(), "User email is not valid")
    }

    func testPassword() throws {
        XCTAssertTrue(viewModel.isPasswordValid(), "User password is not valid")
    }

    func testFullname() throws {
        XCTAssertTrue(viewModel.isFullnameValid(), "User full name is not valid")
    }

    func testPerformanceExample() throws {
        self.measure {
        }
    }
```

```
10.000%, maxPercentRelativeStandardDeviation: 10.000%, maxRegression: 0.100, maxStandardDeviation: 0.100
Test Case '-[FusionTests.RegistrationViewModelTestCase testPerformanceExample]' passed (0.266 seconds).
Test Suite 'RegistrationViewModelTestCase' passed at 2024-04-16 14:30:36.698.
     Executed 4 tests, with 0 failures (0 unexpected) in 0.268 (0.270) seconds
```

Figure 2.5.2 Unit Test 'RegistrationViewModelTestCase'

In figure 2.5.1 shows the 'RegistrationViewModelTestCase'. This test ensures the data inputted during registration is checked for accuracy. The testEmail(), testPassword(), and testFullName() functions are like the checkpoints at a gate. The screenshot demonstrates a series of green lights, indicating that each input has passed its respective test.


Fusion 〉 FusionUITests 〉 LoginViewTestCase 〉 tearDownWithError()

```
 9
   final class LoginViewTestCase: XCTestCase {
11
12     override func setUpWithError() throws {
13         continueAfterFailure = false
14     }
15
16     override func tearDownWithError() throws {
17         // Put teardown code here. This method is called after the invocation of each test method in the class.
18     }
19
       func testLogin() throws {
21         // UI tests must launch the application that they test.
22         let app = XCUIApplication()
23         app.launchArguments = ["isRunningUITests"]
24         app.launch()
25         let timeOut: Double = 2
26         let emailTextField = app.textFields["logInEmail"]
27         let passwordTextField = app.secureTextFields["logInPassword"]
28         let logInButton = app.buttons["logInButton"]
29         XCTAssertTrue(emailTextField.waitForExistence(timeout: timeOut))
30         XCTAssertTrue(passwordTextField.waitForExistence(timeout: timeOut))
31         XCTAssertTrue(logInButton.waitForExistence(timeout: timeOut))
32         emailTextField.tap()
33         emailTextField.typeText("test9@gmail.com")
34         passwordTextField.tap()
35         passwordTextField.typeText("123456")
36         logInButton.tap()
37         let homeTestLabel = app.staticTexts["homeTestLabel"]
38         XCTAssertTrue(homeTestLabel.waitForExistence(timeout: 15))
39         // Use XCTAssert and related functions to verify your tests produce the correct results.
40     }
41
       func testLaunchPerformance() throws {
43         if #available(macOS 10.15, iOS 13.0, tvOS 13.0, watchOS 7.0, *) {
44             // This measures how long it takes to launch your application.
               measure(metrics: [XCTApplicationLaunchMetric()]) {
46                 XCUIApplication().launch()
47             }
```

2 No baseline average for Duration (AppLaunc

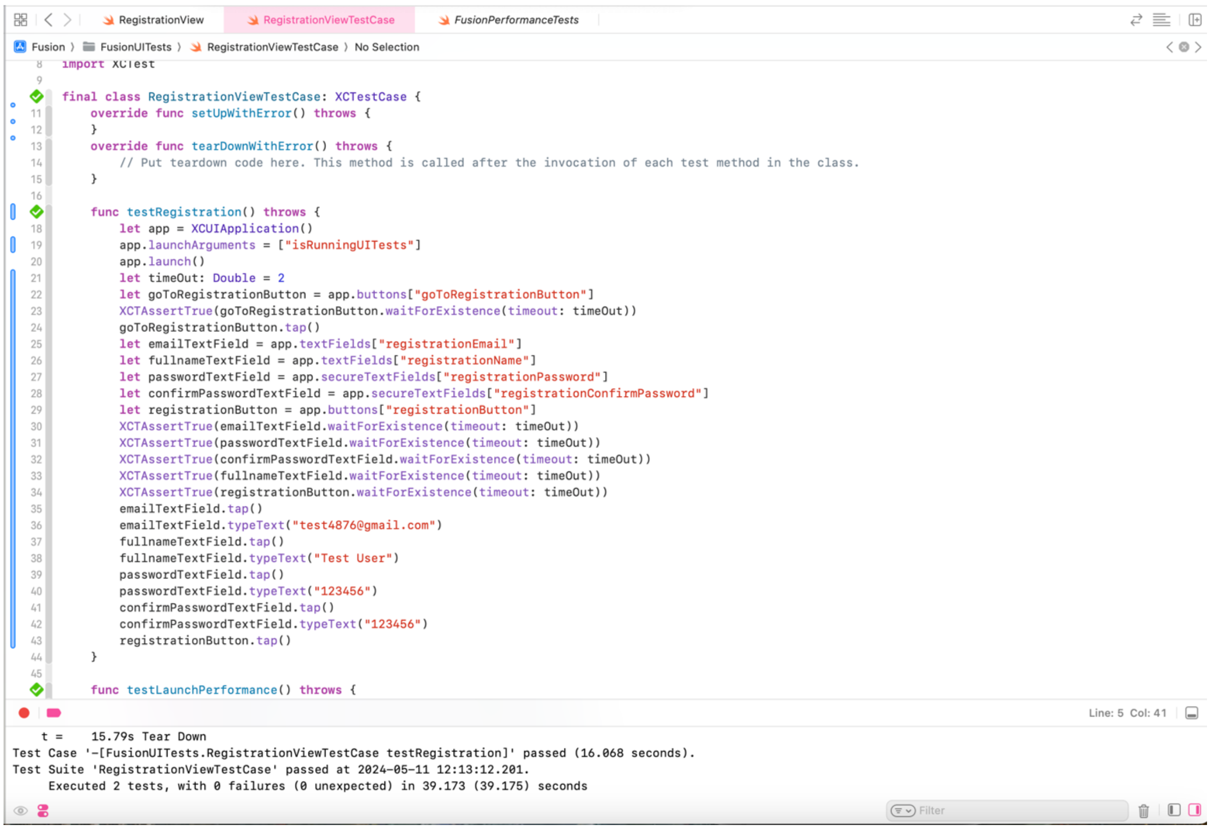
Line: 17 Col: 112

```
    t =    10.77s Tear Down
Test Case '-[FusionUITests.LoginViewTestCase testLogin]' passed (11.018 seconds).
Test Suite 'LoginViewTestCase' passed at 2024-05-10 20:54:31.710.
     Executed 2 tests, with 0 failures (0 unexpected) in 31.656 (31.657) seconds
```

Filter

Figure 2.5.3 – UI Test ‘LoginViewTestCase’

On the screenshot above illustrates the code behind the LoginView UI test. The function testLogin on line 20 holds the code that mimics a log in again. The XCTAssertTrue are the core parts of the UI test as they confirm the existence of certain UI elements during login. If a login succeeds, the user is redirected to the HomeView. The final XCTAssert checks if a certain label that only exists in the HomeView appeared.



RegistrationView | RegistrationViewTestCase | FusionPerformanceTests

Fusion 〉 FusionUITests 〉 RegistrationViewTestCase 〉 No Selection

```swift
import XCTest

final class RegistrationViewTestCase: XCTestCase {
    override func setUpWithError() throws {
    }
    override func tearDownWithError() throws {
        // Put teardown code here. This method is called after the invocation of each test method in the class.
    }

    func testRegistration() throws {
        let app = XCUIApplication()
        app.launchArguments = ["isRunningUITests"]
        app.launch()
        let timeOut: Double = 2
        let goToRegistrationButton = app.buttons["goToRegistrationButton"]
        XCTAssertTrue(goToRegistrationButton.waitForExistence(timeout: timeOut))
        goToRegistrationButton.tap()
        let emailTextField = app.textFields["registrationEmail"]
        let fullnameTextField = app.textFields["registrationName"]
        let passwordTextField = app.secureTextFields["registrationPassword"]
        let confirmPasswordTextField = app.secureTextFields["registrationConfirmPassword"]
        let registrationButton = app.buttons["registrationButton"]
        XCTAssertTrue(emailTextField.waitForExistence(timeout: timeOut))
        XCTAssertTrue(passwordTextField.waitForExistence(timeout: timeOut))
        XCTAssertTrue(confirmPasswordTextField.waitForExistence(timeout: timeOut))
        XCTAssertTrue(fullnameTextField.waitForExistence(timeout: timeOut))
        XCTAssertTrue(registrationButton.waitForExistence(timeout: timeOut))
        emailTextField.tap()
        emailTextField.typeText("test4876@gmail.com")
        fullnameTextField.tap()
        fullnameTextField.typeText("Test User")
        passwordTextField.tap()
        passwordTextField.typeText("123456")
        confirmPasswordTextField.tap()
        confirmPasswordTextField.typeText("123456")
        registrationButton.tap()
    }

    func testLaunchPerformance() throws {
```

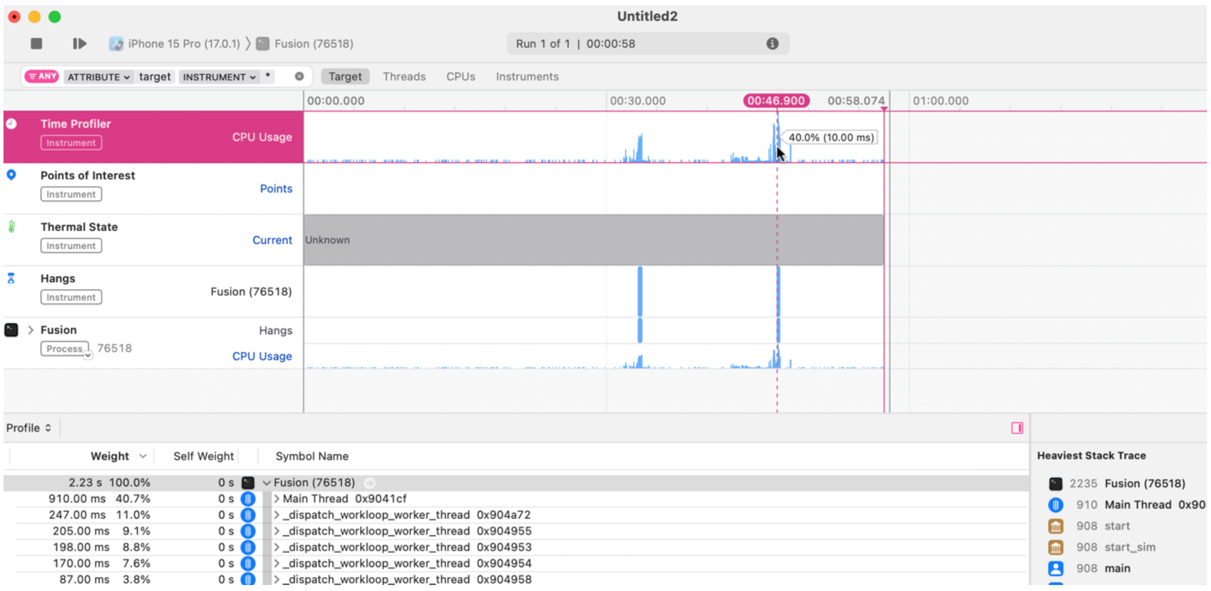
Line: 5 Col: 41

```
    t =    15.79s Tear Down
Test Case '-[FusionUITests.RegistrationViewTestCase testRegistration]' passed (16.068 seconds).
Test Suite 'RegistrationViewTestCase' passed at 2024-05-11 12:13:12.201.
     Executed 2 tests, with 0 failures (0 unexpected) in 39.173 (39.175) seconds
```

Figure 2.5.4 – UI Test 'RegistrationViewTestCase'

The screenshot above shows RegistrationViewTestCase mimics a registration event filling out details like name, email, password and confirm password. It then attempt to perform a registration and provided the Home screen appears, the test will succeed.

2.6 Evaluation


Untitled2
iPhone 15 Pro (17.0.1) 〉 Fusion (76518)
Run 1 of 1 | 00:00:58
ANY ATTRIBUTE target INSTRUMENT
Target Threads CPUs Instruments
00:00.000 00:30.000 00:46.900 00:58.074 01:00.000
Time Profiler Instrument CPU Usage
40.0% (10.00 ms)
Points of Interest Instrument Points
Thermal State Instrument Current
Unknown
Hangs Instrument Fusion (76518)
Fusion Process 76518 Hangs CPU Usage
Profile

Weight		Self Weight	Symbol Name
2.23 s	100.0%	0 s	Fusion (76518)
910.00 ms	40.7%	0 s	Main Thread 0x9041cf
247.00 ms	11.0%	0 s	_dispatch_workloop_worker_thread 0x904a72
205.00 ms	9.1%	0 s	_dispatch_workloop_worker_thread 0x904955
198.00 ms	8.8%	0 s	_dispatch_workloop_worker_thread 0x904953
170.00 ms	7.6%	0 s	_dispatch_workloop_worker_thread 0x904954
87.00 ms	3.8%	0 s	_dispatch_workloop_worker_thread 0x904958

Heaviest Stack Trace

2235	Fusion (76518)
910	Main Thread 0x90
908	start
908	start_sim
908	main

Figure 2.6.1 – CPU usage - Instruments tool

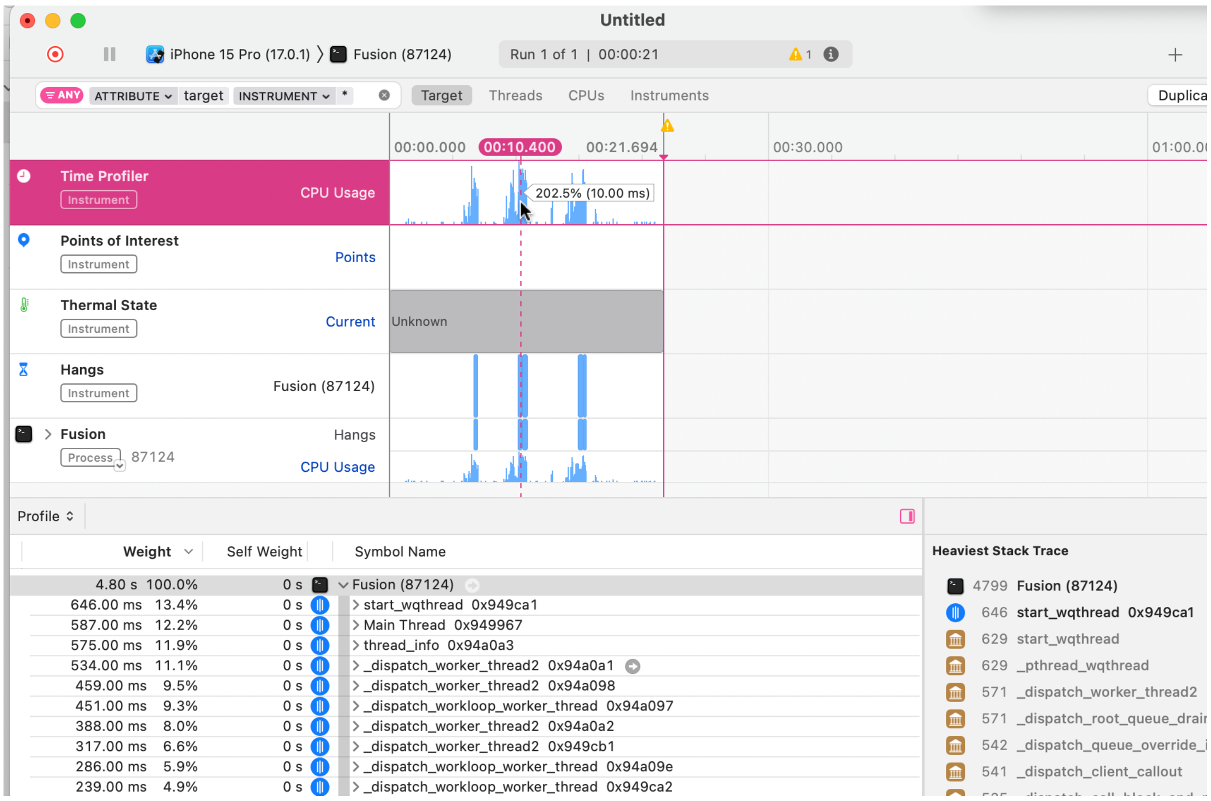
To evaluate the performance of Fusion I used the debugging tool called Instruments which comes included with XCode. This provides details into Fusion's CPU usage and processing efficiency. Using the Instruments tool I measured the critical main point and in this case it is the swiping feature.

The time profiler data indicates that the Main Thread accounts for a significant portion of the CPU usage. Main thread is mostly UI working. The spikes shown in the graph is where a swipe occurred peaking at 40% over a period of 10ms. This is pretty fast considering when completing a swipe three things happen:

1. This swipe is fed into the recommendation algorithm.

2. It is observing if to see if other users have equally swiped right on that user.

3. It actively stores all the swipes made by the user to avoid the same card appearing again.


Untitled
iPhone 15 Pro (17.0.1) 〉 Fusion (87124)
Run 1 of 1 | 00:00:21
ANY ATTRIBUTE target INSTRUMENT *
Target Threads CPUs Instruments
Duplica
00:00.000 00:10.400 00:21.694 00:30.000 01:00.0
Time Profiler Instrument CPU Usage
202.5% (10.00 ms)
Points of Interest Instrument Points
Thermal State Instrument Current
Unknown
Hangs Instrument Fusion (87124)
Fusion Process 87124 Hangs CPU Usage
Profile

Weight		Self Weight	Symbol Name
4.80 s	100.0%	0 s	Fusion (87124)
646.00 ms	13.4%	0 s	start_wqthread 0x949ca1
587.00 ms	12.2%	0 s	Main Thread 0x949967
575.00 ms	11.9%	0 s	thread_info 0x94a0a3
534.00 ms	11.1%	0 s	_dispatch_worker_thread2 0x94a0a1
459.00 ms	9.5%	0 s	_dispatch_worker_thread2 0x94a098
451.00 ms	9.3%	0 s	_dispatch_workloop_worker_thread 0x94a097
388.00 ms	8.0%	0 s	_dispatch_worker_thread2 0x94a0a2
317.00 ms	6.6%	0 s	_dispatch_worker_thread2 0x949cb1
286.00 ms	5.9%	0 s	_dispatch_workloop_worker_thread 0x94a09e
239.00 ms	4.9%	0 s	_dispatch_workloop_worker_thread 0x949ca2

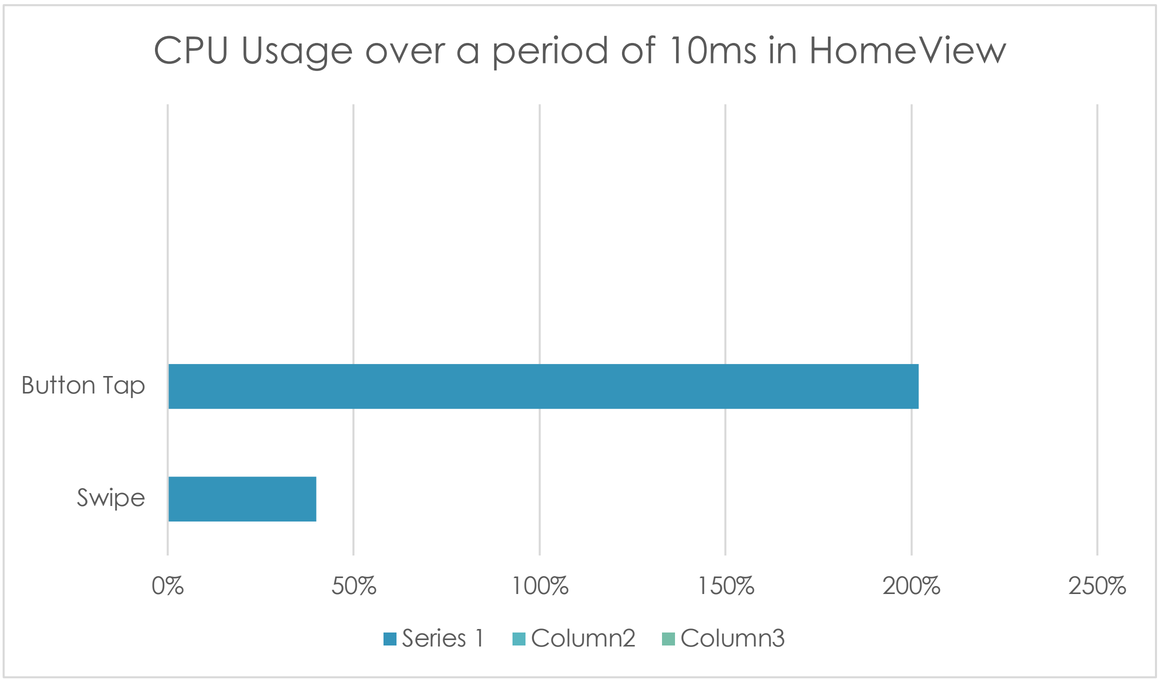
Heaviest Stack Trace

4799	Fusion (87124)
646	start_wqthread 0x949ca1
629	start_wqthread
629	_pthread_wqthread
571	_dispatch_worker_thread2
571	_dispatch_root_queue_drai
542	_dispatch_queue_override_i
541	_dispatch_client_callout

Figure 2.6.2 – CPU Usage - Instruments tool

The time profiler shows spikes where the ‘like’ and ‘dislike’ button were tapped. As shown on the image CPU usage rose to 202% over a period of 10ms. Tapping the like or dislike button and swiping the card achieve the same thing but tapping the button requires more resources as the library I used was designed for swiping cards and I included buttons on the card for ‘like’ and ‘dislike’ as the buttons were not originally in the library.

By analysing figure 2.6.1 and figure 2.6.2, the bigger spike occurred in figure 2.6.2 when tapping the ‘like’ and ‘dislike’ button at 202% for a period of 10ms. Whereas the spike in figure 2.6.1 was only at 40% over a period of 10ms. The time taken was the same but one feature needs more resources. Therefore the percentage rise from 40% to 202% is 405%.The underlying mechanism in the buttons may account for greater spike and lack of efficiency.



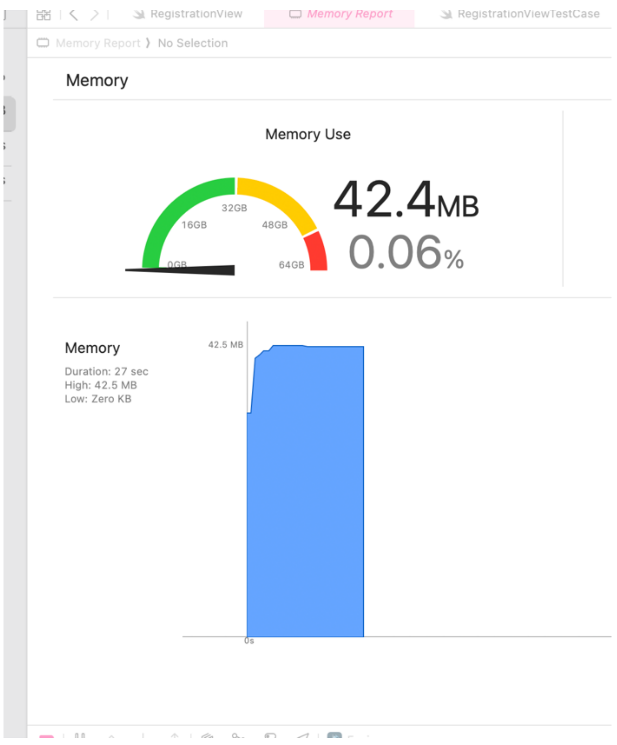


Figure 2.6.3 – Memory Usage of LoginView

In figure 2.6.3 is the lowest usage I have which is in the LoginView. The arrow is in the green section and shows that it is using very little memory. Log in takes only 42.4mb of memory at 0.06%

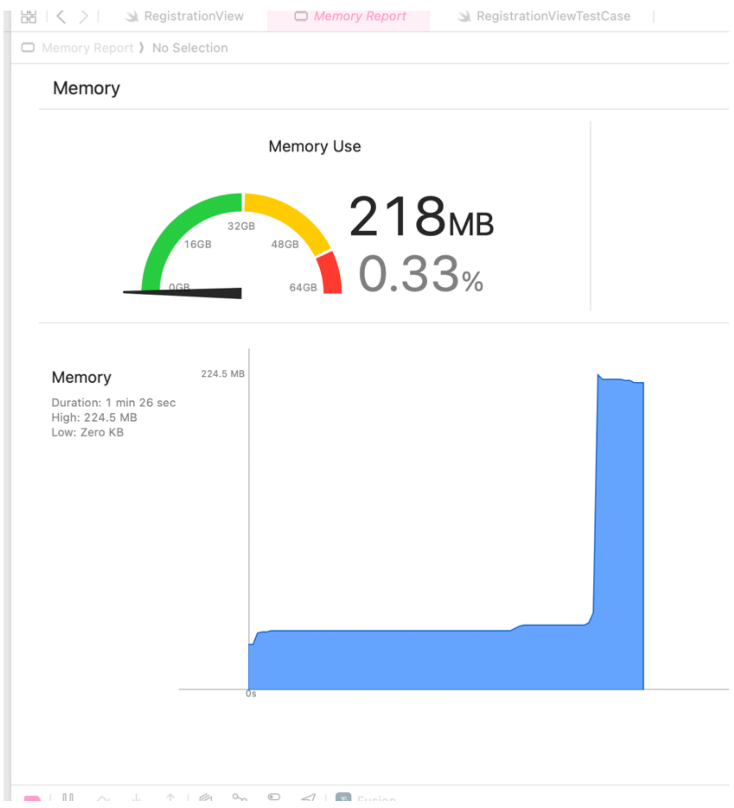
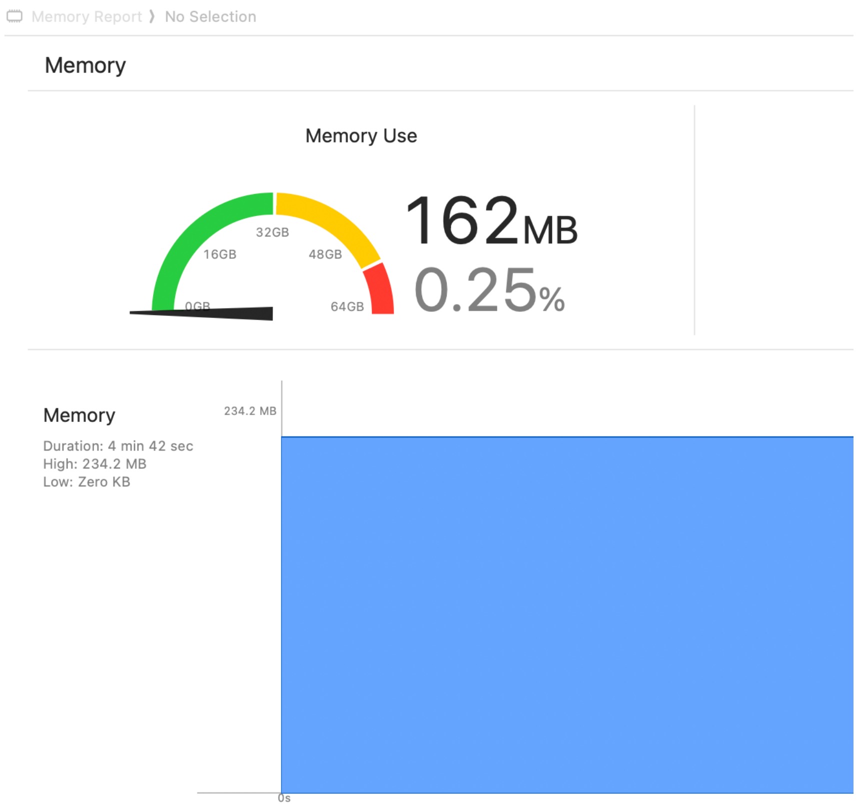


Figure 2.6.4 – Memory Usage of HomeView

When I logged in and HomeView was opened the memory jumped to 218mb at 0.33%. This jump in memory makes sense as more resources are being used but the arrow still points in green area showing that the app continues to be efficient when the HomeView is opened.

By conducting this test I was able to also asses for memory leaks. Memory leaks happen when an application continues to use memory when it is no longer needed. This can lead to performance issues. Memory leaks can occur when an application fails to properly release memory that it has allocated.

The jump in memory between LoginView in figure 2.6.3 and HomeView figure 2.6.4 is a approx. a 414% percentage increase. This is expected.

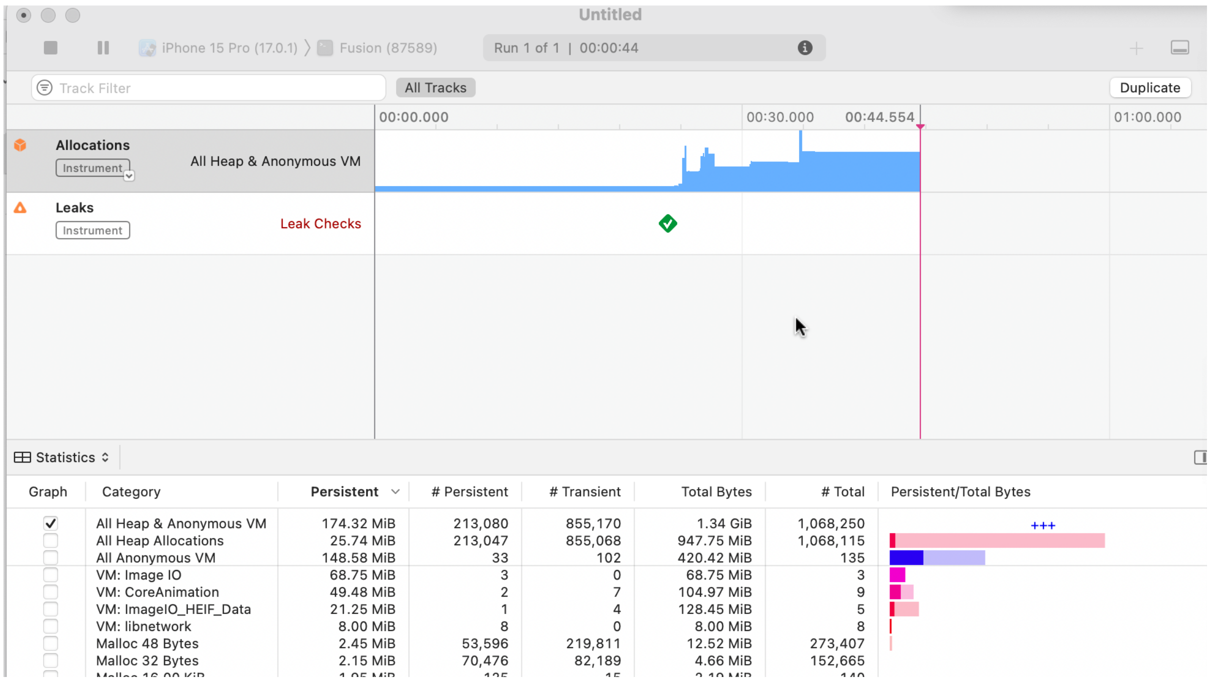


162MB
0.25%

Memory
Duration: 4 min 42 sec
High: 234.2 MB
Low: Zero KB

Figure 2.6.5 – Memory Usage of HomeView when swiping

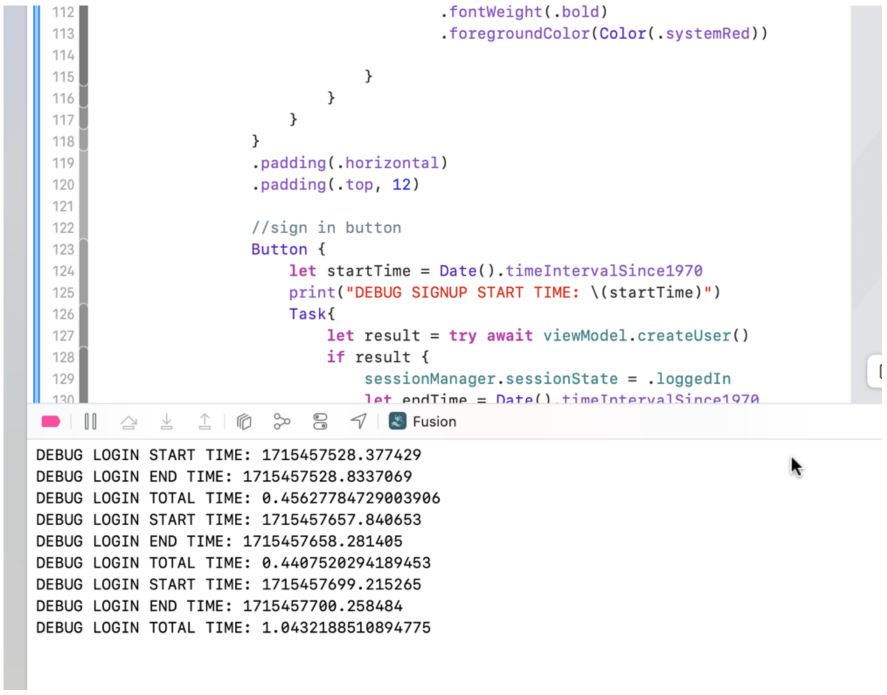
As I began swiping through the profile cards in the HomeView the memory usage decreased to 162mb. The reduction percentage from figure 2.6.4 compared to figure 2.6.5 is approx. 25%.



Graph	Category	Persistent	# Persistent	# Transient	Total Bytes	# Total	Persistent/Total Bytes
✓	All Heap & Anonymous VM	174.32 MiB	213,080	855,170	1.34 GiB	1,068,250	+++
	All Heap Allocations	25.74 MiB	213,047	855,068	947.75 MiB	1,068,115	
	All Anonymous VM	148.58 MiB	33	102	420.42 MiB	135	
	VM: Image IO	68.75 MiB	3	0	68.75 MiB	3	
	VM: CoreAnimation	49.48 MiB	2	7	104.97 MiB	9	
	VM: ImageIO_HEIF_Data	21.25 MiB	1	4	128.45 MiB	5	
	VM: libnetwork	8.00 MiB	8	0	8.00 MiB	8	
	Malloc 48 Bytes	2.45 MiB	53,596	219,811	12.52 MiB	273,407	
	Malloc 32 Bytes	2.15 MiB	70,476	82,189	4.66 MiB	152,665	

Figure 2.6.6 – Investigating if there are any memory leaks when swiping

To further investigate memory leaks I used another tool in Instruments that profiles memory leaks. This tool analyses the memory usage and as shown in the image I have a green check mark while swiping indicating it is being used efficiently.



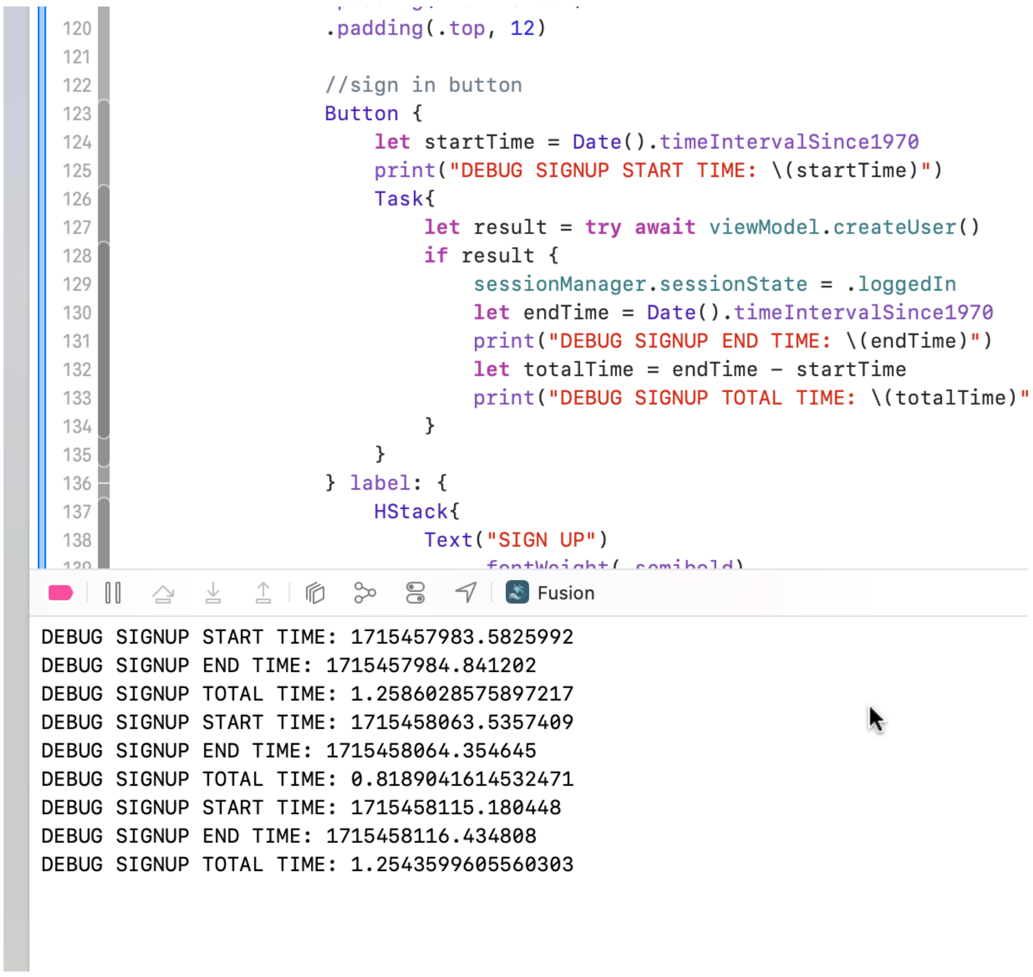
```
                                    .fontWeight(.bold)
                                    .foregroundColor(Color(.systemRed))

                            }
                        }
                    }
                }
                .padding(.horizontal)
                .padding(.top, 12)

                //sign in button
                Button {
                    let startTime = Date().timeIntervalSince1970
                    print("DEBUG SIGNUP START TIME: \(startTime)")
                    Task{
                        let result = try await viewModel.createUser()
                        if result {
                            sessionManager.sessionState = .loggedIn
                            let endTime = Date().timeIntervalSince1970
```

```
DEBUG LOGIN START TIME: 1715457528.377429
DEBUG LOGIN END TIME: 1715457528.8337069
DEBUG LOGIN TOTAL TIME: 0.45627784729003906
DEBUG LOGIN START TIME: 1715457657.840653
DEBUG LOGIN END TIME: 1715457658.281405
DEBUG LOGIN TOTAL TIME: 0.4407520294189453
DEBUG LOGIN START TIME: 1715457699.215265
DEBUG LOGIN END TIME: 1715457700.258484
DEBUG LOGIN TOTAL TIME: 1.0432188510894775
```

Figure 2.6.7 – Speed analysis of logging into Fusion

Above is a speed analysis of logging into Fusion that is printed in the console. I conducted three tests. I used a debugging method by writing code and assessed the time by logging the start time of process, end time of the process and these two were taken away from each other and resulted in the speed of the function. The range of times are 0.456s, 0.440s and 1.043s averaging at around 0.646s with the lowest being 0.440s and the highest is just above a second at 1.043s. The speed of logging in was pretty fast and people would not notice it taking 1 second.

Unfortunately the speed is affected by the speed of the internet or the speed of the server and this is out of the user's control. Factors like using mobile data may slow down the app.

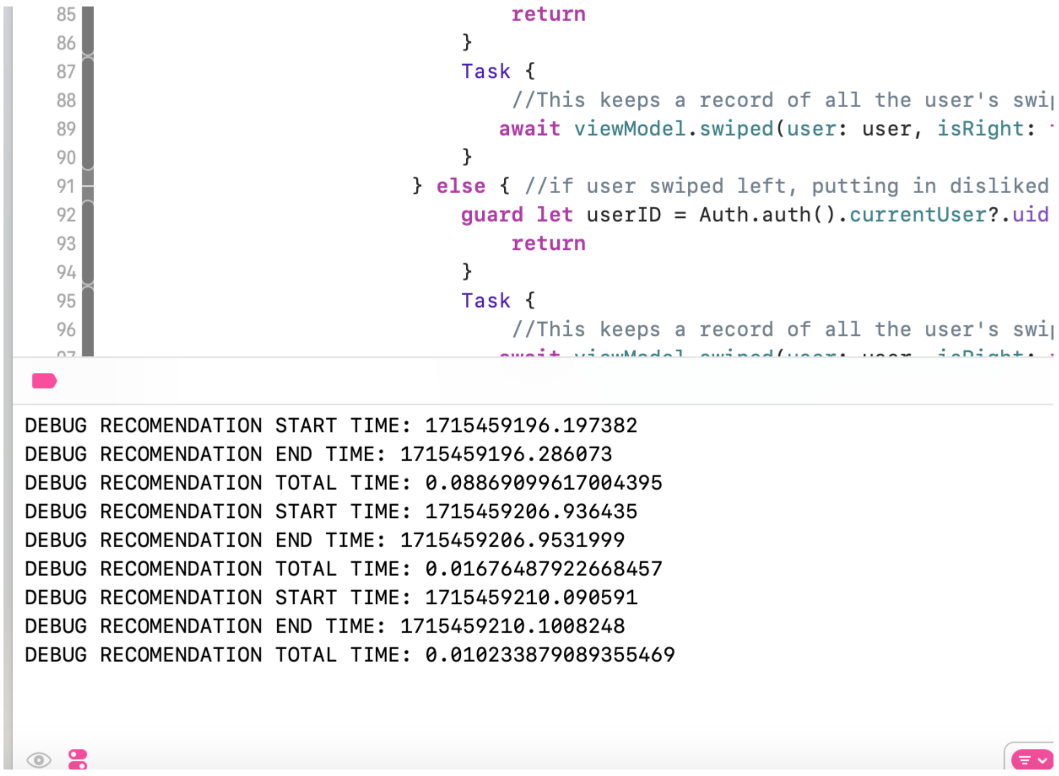


```
                .padding(.top, 12)

                //sign in button
                Button {
                    let startTime = Date().timeIntervalSince1970
                    print("DEBUG SIGNUP START TIME: \(startTime)")
                    Task{
                        let result = try await viewModel.createUser()
                        if result {
                            sessionManager.sessionState = .loggedIn
                            let endTime = Date().timeIntervalSince1970
                            print("DEBUG SIGNUP END TIME: \(endTime)")
                            let totalTime = endTime - startTime
                            print("DEBUG SIGNUP TOTAL TIME: \(totalTime)"
                        }
                    }
                } label: {
                    HStack{
                        Text("SIGN UP")
```

```
DEBUG SIGNUP START TIME: 1715457983.5825992
DEBUG SIGNUP END TIME: 1715457984.841202
DEBUG SIGNUP TOTAL TIME: 1.2586028575897217
DEBUG SIGNUP START TIME: 1715458063.5357409
DEBUG SIGNUP END TIME: 1715458064.354645
DEBUG SIGNUP TOTAL TIME: 0.8189041614532471
DEBUG SIGNUP START TIME: 1715458115.180448
DEBUG SIGNUP END TIME: 1715458116.434808
DEBUG SIGNUP TOTAL TIME: 1.2543599605560303
```

Figure 2.6.8 – Speed analysis of signing up for Fusion

In the console is shown the time in second taken to sign up. Three tests were conducted. The highest amount of time taken is 1.258 s and the lowest is 0.818 s with the average coming out to 1.110 seconds. On average it takes 0.464s longer to log in than sign up.



```
                                    return
                                }
                                Task {
                                    //This keeps a record of all the user's swi
                                   await viewModel.swiped(user: user, isRight:
                                }
                            } else { //if user swiped left, putting in disliked
                                guard let userID = Auth.auth().currentUser?.uid
                                    return
                                }
                                Task {
                                    //This keeps a record of all the user's swi
```

```
DEBUG RECOMENDATION START TIME: 1715459196.197382
DEBUG RECOMENDATION END TIME: 1715459196.286073
DEBUG RECOMENDATION TOTAL TIME: 0.08869099617004395
DEBUG RECOMENDATION START TIME: 1715459206.936435
DEBUG RECOMENDATION END TIME: 1715459206.9531999
DEBUG RECOMENDATION TOTAL TIME: 0.01676487922668457
DEBUG RECOMENDATION START TIME: 1715459210.090591
DEBUG RECOMENDATION END TIME: 1715459210.1008248
DEBUG RECOMENDATION TOTAL TIME: 0.010233879089355469
```

Figure 2.6.9 – Speed analysis of the app providing recommendations

In my last speed analysis test I am measuring to see the time it takes Fusion to provide recommendations. Three tests were completed and the average time it took to recommend users was about 0.068s. This is fast and the reason why it is so fast is because the recommendations are being trained on the phone.

Results Speed Analysis Table

	Test 1	**Test 2**	**Test 3**	**Average**
Log - In	0.456s	0.440s	1.043s	0.646s
Sign - Up	1.258s	1.254s	0.818s	1.110s
Recommendations	0.088s	0.016s	0.010s	0.068s

Do my algorithms perform as expected?

Yes, the algorithms in Fusion work as intended. The recommendation algorithm uses linear regression to predict relevant business connections. It considers user interactions, geohashing location data, and business fields. By training on mock data, it learns patterns and makes accurate recommendations. Testing showed high relevance and user satisfaction, proving its effectiveness.

The geohashing algorithm converts geographic coordinates into alphanumeric strings. This enables location-based recommendations without exposing precise locations. It groups users geographically, ensuring local relevance while maintaining privacy.

3.0 Conclusions

As I reflect on the journey of developing Fusion, I'm proud of what I have accomplished. I followed the actions I have set out at the beginning of the year. I successfully integrated key features such as an authentication feature, real-time messaging. I was determined to add in features that would be innovative the way professionals connect. Fusion combines geohashing algorithm (a geocoding method used to encode coordinates into a string of digits and letters) and a recommendation algorithm using linear regression(a method that streamlined the process of connecting users based on their proximity and business fields) and by adding in a swiping feature this significantly enhances user engagement by simplifying the connection process. The ease of use and the increased opportunities for bringing like-minded professionals and entrepreneurs and this can lead to business growth and collaboration.

During my third year in college, I had a module called Artificial Intelligence and Machine Learning. I completed assignments on Logistic Regression and K-Nearest Neighbour, which equipped me to revisit these methods with confidence when integrating logistic regression into my app. This academic background was crucial in laying the technical foundation for my project. Data analysis and cleaning played a significant role in refining the app, especially for setting up the mock users for the recommender algorithm. My prior internship experience, where I worked extensively with data, was instrumental in making this aspect of the project feel familiar and manageable.

I used Trello to track my progress during the development process, making sure that each feature I added was in line with the main goals of the app. I was able to stay focused on developing a minimum viable product (MVP) and avoid the pitfalls of overcomplicating the app from all my ideas by being disciplined. I didn't want my app to become too complex and wanted to focus on the main features because of this I removed the notifications and swapped it with a in-app popup when the user makes a connection.
The recommender algorithm was greatly influenced by my internship experience, especially with regard to data cleaning and analysis.

By completing this project it has been a rewarding journey, I learned so much about data, efficient matchmaking and also enhanced my own technical proficiency in algorithm design and application. By implementing the algorithms I had many issues and challenges which I had to be resilient and use my problem solving skills. I had to make sure that they were effective, I had issues when adding in geofirestore due to it being written in objective-C and it was necessary for me to use bridging headers in order to make it compatible. Implementing the recommender algorithm had come with it's own difficulties with making the algorithm responsive. The recommender algorithm initially struggled to function effectively until I managed to compile a sufficiently large dataset, which required considerable time and effort to develop and enhance.

Regarding the strengths and limitations, one of the most significant strengths is the app's ability to provide precise and secure connections, which is essential in today's digital age. A limitation of the app is its reliance on continuous internet connectivity. Fusion’s features real-time data updates and location-based services through geohashing, users without a stable and fast internet connection may experience delays or interruptions in service, which could affect the user experience and effectiveness of the networking functionalities.

Another limitation of the app is that if there is no users in the radius of 500km, the app would not be able to display any results in the HomeView. In order for the app to be useful, it is necessary there are enough users to be able to connect.

I chose firebase as the database and it was a calculated decision as it fits with the requirements. The architecture of Firebase supports real-time updates which is important for the swiping feature and messaging feature to show changes, Firebase is scalable, it can accommodate higher loads without compromising performance as the app grows. Firebase also provides crash reporting, analytics, authentication and many things. Firebase can handle increased loads without any loss in performance.

Throughout this project I developed my skills as a software developer and learning how to code in swift. I learned to combine various fields of knowledge including working with algorithms, testing and user experience design. I refined my abilities in project planning and management and learnt to set realistic milestones and manage my resources in order to complete everything in time. I realised that there is many aspects to developing an app and I understand why it can take even a couple of years to release a fully functioning app. Each challenge was a learning opportunity and I feel that I am better equipped for working on future projects.

4.0 Further Development or Research

With additional time and resources, Fusion would broaden and expand its capabilities across multiple key areas:

The linear regression-based recommendation algorithm would be further trained with more user data, incorporating additional factors such as bios and professional experiences. This would not only increase its accuracy but also allow it to suggest more relevant connections, creating a database that grows organically with more users.

I would also introduce specific subcategories within broader business fields, such as “Software” within “Technology” providing more granular networking opportunities tailored to the user’s specialise.

Fusion would integrate within the Apple ecosystem by offering notifications and allowing users to log in seamlessly using their Apple accounts.

To broaden its reach the app would be deployed on the App Store making it accessible to a larger user base across more countries.

A separate section would also be created within the app where entrepreneurs can showcase their start-ups, allowing investors to browse and invest in their desired business. I believe this will simplify bringing start-ups and investors together and also enhancing the funding process, benefiting both parties.

I aim to transition Fusion into a fully-fledged business and have a solid foundation with a team working with me.

A subscription based monetization strategy would be implemented to have a sustainable revenue stream

I would allocate resources to marketing and build a community around the app and encourage engagement and regular networking activities.

The app's design would be further refined to make it more visually appealing, elegant and user friendly.

5.0 References

AppStuff (2023). *COMPLETE User Login / Sign Up App | Swift UI + Firebase | Async / Await*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=QJHmhLGv-_0&ab_channel=AppStuff [Accessed 3 Dec. 2023].

Code With Chris (2023). *How to Make an App - Lesson 1 (2024 / SwiftUI)*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=HJDCXdhQaP0&ab_channel=CodeWithChris [Accessed 5 Nov. 2023].

Code With Chris (2023b). *Intro to Xcode 14 - Lesson 2 (2024 / SwiftUI)*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=8dVLNxs0kTE&ab_channel=CodeWithChris [Accessed 8 Nov. 2023].

Code With Chris (2023a). *How to Build a User Interface in Xcode - Lesson 3 (2024 / SwiftUI)*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=PCvopT3Wnek&list=PLMRqhzcHGw1Y5Cluhf7pKRNZtKaA3Q4kg&index=3&ab_channel=CodeWithChris [Accessed 10 Nov. 2023].

Code With Chris (2023d). *Starting The War Card Game App - Lesson 4 (2024 / SwiftUI)*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=2Zi-looUuIA&list=PLMRqhzcHGw1Y5Cluhf7pKRNZtKaA3Q4kg&index=4&ab_channel=CodeWithChris [Accessed 23 Nov. 2023].

Code With Chris (2023d). *Introduction to Swift Coding - Lesson 5 (2024 / SwiftUI)*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=Hhf5pnWG1II&list=PLMRqhzcHGw1Y5Cluhf7pKRNZtKaA3Q4kg&index=5&ab_channel=CodeWithChris [Accessed 25 Nov. 2023].

Code With Chris (2023). *Swift Functions - Lesson 6 (2024 / SwiftUI)*. *YouTube*. Available at: https://www.youtube.com/watch?v=r-wkDztkB0E&list=PLMRqhzcHGw1Y5Cluhf7pKRNZtKaA3Q4kg&index=6&ab_channel=CodeWithChris [Accessed 25 Nov. 2023].

Code With Chris (2023f). *SwiftUI Buttons and Properties - Lesson 7 (2024 / SwiftUI)*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=q3KRE-MyCO8&list=PLMRqhzcHGw1Y5Cluhf7pKRNZtKaA3Q4kg&index=7&ab_channel=CodeWithChris [Accessed 26 Nov. 2023].

Code With Chris (2023a). *Creating the Logic - Lesson 8 (2024 / SwiftUI)*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=YDtZrQjrp10&list=PLMRqhzcHGw1Y5Cluhf7pKRNZtKaA3Q4kg&index=8&ab_channel=CodeWithChris [Accessed 28 Nov. 2023].

Adalar, D. (2023). *dadalar/SwiftUI-CardStackView*. [online] GitHub. Available at: https://github.com/dadalar/SwiftUI-CardStackView [Accessed 1 Feb. 2024].

Erbay, Y. (2022). *Unit Testing in Swift*. [online] adessoTurkey. Available at: https://medium.com/adessoturkey/unit-testing-in-swift-c1607da61e13 [Accessed 16 Feb. 2024].

Khaliq, K. (2023). *Test-Driven Development (TDD) in Swift*. [online] Khawer Khaliq. Available at: https://khawerkhaliq.com/blog/swift-test-driven-development/ [Accessed 5 Jan. 2024].

Gadhesariya, P. (n.d.). *pratikg29/LightDarkMode-ToggleStyle*. [online] GitHub. Available at: https://github.com/pratikg29/LightDarkMode-ToggleStyle [Accessed 16 Mar. 2024].

GitHub. (n.d.). *imperiumlabs/GeoFirestore*. [online] Available at: https://github.com/imperiumlabs/GeoFirestore [Accessed Feb. 2024].

geofirestore.com. (n.d.). *geofirestore*. [online] Available at: https://geofirestore.com/ [Accessed Feb. 2024].

Hudson, P. (n.d.). *How to create an Objective-C bridging header to use code in Swift*. [online] Available at: https://www.hackingwithswift.com/example-code/language/how-to-create-an-objective-c-bridging-header-to-use-code-in-swift.

Indeed (n.d.). *How to Write a Project Proposal (With Example)*. [online] Indeed Career Guide. Available at: https://www.indeed.com/career-advice/career-development/how-to-write-a-project-proposal [Accessed Apr. 2024].

Slite (n.d.). *How to Write a Perfect Project Proposal in 2021*. [online] slite.com. Available at: https://slite.com/learn/how-to-write-project-proposal [Accessed Apr. 2024].

sdtaheri (n.d.). *Building a Recommendation App With Create ML in SwiftUI*. [online] kodeco.com. Available at: https://www.kodeco.com/34652639-building-a-recommendation-app-with-create-ml-in-swiftui [Accessed Apr. 2024].

S, R. (2024). *Recommender Systems: Collaborative Filtering and Content-Based Filtering*. [online] Medium. Available at: https://ogre51.medium.com/recommender-systems-collaborative-filtering-and-content-based-filtering-20785d50d56f#:~:text=Recommendation%20Systems%3A%20An%20Introduction&text=Collaborative%20Filtering%20uses%20user%20interaction [Accessed Apr. 2024].

deepanshi (2021). *Linear Regression | Introduction to Linear Regression for Data Science*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/05/all-you-need-to-know-about-your-first-machine-learning-model-linear-regression/ [Accessed Apr. 2024].

Aws (2020). *Implementing geohashing at scale in serverless web applications | AWS Compute Blog*. [online] aws.amazon.com. Available at: https://aws.amazon.com/blogs/compute/implementing-geohashing-at-scale-in-serverless-web-applications/ [Accessed 1 Apr. 2024].

Unsplash (n.d.). *Beautiful Free Images & Pictures*. [online] Unsplash. Available at: https://unsplash.com/ [Accessed 2024].

ProductPlan (2022). *What is a Minimum Viable Product (MVP)? | A Product Mgmt Definition*. [online] www.productplan.com. Available at: https://www.productplan.com/glossary/minimum-viable-product/.

www.youtube.com. (n.d.). *iOS Tutorial: Bridging Swift And Objective C*. [online] Available at: https://www.youtube.com/watch?v=Wp_-8tE85hE&ab_channel=CodePro [Accessed Mar. 2024].

Anon, (2023). *The Power of Networking for Entrepreneurs*. [online] Available at: https://www.osmoscloud.com/blog/the-power-of-networking-for-entrepreneurs/ [Accessed April 2024].

SwiftAnytime (n.d.). *AsyncImage in SwiftUI - Swift Anytime*. [online] www.swiftanytime.com. Available at: https://www.swiftanytime.com/blog/asyncimage-in-swiftui [Accessed April 2024].

developer.apple.com. (n.d.). *Apple Developer Documentation*. [online] Available at: https://developer.apple.com/documentation/swiftui/geometryreader.

Hrabowskie, R. (2023). *rick2785/SwiftUIFirebaseChat*. [online] GitHub. Available at: https://github.com/rick2785/SwiftUIFirebaseChat

developer.apple.com. (n.d.). *Account deletion requirement starts June 30 - Latest News - Apple Developer*. [online] Available at: https://developer.apple.com/news/?id=12m75xbj

Malik, O.I. (2021). *Right to delete under California Consumer Privacy Act (CCPA)*. [online] Securiti. Available at: https://securiti.ai/blog/ccpa-right-to-delete/

ico.org.uk. (2024). *Your right to get your data deleted*. [online] Available at: https://ico.org.uk/for-the-public/your-right-to-get-your-data-deleted/#:~:text=How%20do%20I%20ask%20for [Accessed 2024].

Firebase. (n.d.). *Upload Files on iOS*. [online] Available at: https://firebase.google.com/docs/storage/ios/upload-files.

Apple Developer. (n.d.). *Create ML - Machine Learning*. [online] Available at: https://developer.apple.com/machine-learning/create-ml/.

Apple Developer Documentation. (n.d.). *Requesting authorization to use location services*. [online] Available at: https://developer.apple.com/documentation/corelocation/requesting_authorization_to_use_location_services [Accessed 2024].

Apple.com. (2020). *Apple Developer Documentation*. [online] Available at: https://developer.apple.com/documentation/corelocation.

Hossain, J. (2023). *Access the Camera and Photo Library — iOS SwiftUI*. [online] Medium. Available at: https://medium.com/@jakir/access-the-camera-and-photo-library-in-swiftui-0351a3c280f5 [Accessed 2024].

www.youtube.com. (n.d.). *Intro to Unit Testing in Swift*. [online] Available at: https://www.youtube.com/watch?v=opkU2UuPk0A&ab_channel=SeanAllen [Accessed 2024].

tundsdev (2022). *What is Unit Testing, Integration, UI Testing & Benefits*. *YouTube*. Available at: https://www.youtube.com/watch?v=_wLdNSvGPB8&list=PLvUWi5tdh92zaJIMa65f8G2aWTytY3rN3&ab_channel=tundsdev [Accessed Mar. 2024].

leamars (n.d.). *Instruments Tutorial with Swift: Getting Started*. [online] kodeco.com. Available at: https://www.kodeco.com/16126261-instruments-tutorial-with-swift-getting-started [Accessed 17 Apr. 2024].

Toptal Engineering Blog. (n.d.). *Swift Tutorial: An Introduction to the MVVM Design Pattern | Toptal®*. [online] Available at: https://www.toptal.com/ios/swift-tutorial-introduction-to-mvvm [Accessed 11 Dec. 2023].

Apple (n.d.). *Writing and running performance tests*. [online] Apple Developer Documentation. Available at: https://developer.apple.com/documentation/xcode/writing-and-running-performance-tests [Accessed 16 Apr. 2024].

Gallardo, E.G. (2023). *What Is MVVM Architecture? (Definition, Advantages) | Built In*. [online] builtin.com. Available at: https://builtin.com/software-engineering-perspectives/mvvm-architecture [Accessed 4 Jan. 2024].

Mparticle.com. (2023). *iOS Session Management*. [online] Available at: https://docs.mparticle.com/developers/sdk/ios/session-management/#:~:text=The%20mParticle%20platform%20tracks%20user [Accessed 2 Apr. 2024].

6.0 Appendices

6.1 Project Proposal

National College of Ireland

FUSION

Business Networking Application

BSc in Computing
Software Engineering
Computing Project
2023/2024

Table of Contents

Objectives

The objective of this project is to create a networking IOS application to seamlessly connect entrepreneurs, investors and business-minded individuals based on the field of business, interests, geographical proximity and work experience. I aim to provide value in several ways:

- Creating a user-friendly platform where users can connect, share business ideas and network.
- The possibility of finding a potential business partner. Many entrepreneurs struggle to find the appropriate partner for business, it's easier said than done. Once the profile is completed potential entrepreneurs can swipe to find another potential entrepreneur based on their profiles and the field of business they are interested in.
- The app will have a communication feature where once two users have connected they will be able to chat.

To ensure I have a high user engagement I will implement the following strategies:

- Each user will be able to create a profile. The user will be able to add pictures, their personal details, bio, interests and also highlight the field of business they are interested in or have a business in.
- Once a profile is created, the user will be able to swipe and view other users' profile cards which will be an intuitive and engaging way for users to connect.
- Within the app there will be a messaging feature simplifying communication between users, making it easier to discuss potential business opportunities.
- Have a separate section for investors to discover start-ups.

In terms of having a positive UX, I plan to have the following features:

- I plan to design the app to have a clean, user-friendly interface. The user will have the ability to access the main features with minimal effort.
- The app will have seamless navigations and will be visually appealing.
- The app will be responsive to meet user expectations with minimal loading times.
- The main colour in the application will be a turquoise–blueish colour. The hex code is #4CA6A7

Background

My journey as a student, my passion for business and my interest in bringing individuals together aided in providing me with valuable insights. Through my interactions with students, mentors at my internship, professors and various individuals, I noticed a common theme – there is *great* power in networking and connections. Regardless of individual background, location or job, we are social animals and seek avenues to network, collaborate and engage with like-minded people. This realisation has led me to the idea of connecting entrepreneurs and business-minded individuals together using technology to bridge this gap.

The objectives that I have set above, I plan to achieve them by:

- Creating mock-ups
- Using a user-friendly sign-up process to gather information and set up a profile.
- Implementing real-time messaging feature for users to have conversations.
- Creating a mind map/road map for building my app
- Designing user profile cards that will display essential information such as name, city and business interests. The cards will serve as a basis for the swipe function.
- Implementing left and right swipe gestures for users to indicate which profile they would like to connect with. A right swipe will mean interested and a left swipe would mean disinterest.
- Designing the app simple and accessible with a user-friendly design ensures that every user can navigate my app.
- Watching plenty of YouTube tutorials and Swift courses.

Stare of the Art

Through the research that I conducted, I discovered several networking and business-related applications that exist on the market. The names of the application are as follows: LinkedIn, AngelList, Bumble Bizz and Sharpr.

To differentiate and stand out from the applications I listed I will focus on the following aspects:

- In terms of UX and UI design, my app will be visually appealing and user-friendly.
- My app will appeal to individuals of all generations as it's simple to use especially with the swipe feature that I will incorporate.
- My app is niche and focuses on entrepreneurs, investors and individuals who share common business interests.
- Having more sections within the app related to business for example dedicating a section to business start-ups. Investors would be able to click into this section and discover already existing start-ups and choose the one they would like to invest in.

Technical Approach

To begin developing my application I will use this Project Proposal as a guide to define my vision and goals for this application. The approach I will take for app development is as follows:

- I will begin to create prototypes on the Mockplus website. This will visualise my app's UI and UX.
- Next, I will design my app's architecture and organise the various layers and components required in order for my app to run functionally.
- To successfully build my application I will break down the development process into sprints using Azure boards, I used sprints and Azure boards in my internship and I learned how to use them.
- When beginning coding I will start with the core functionalities, these will be the user profile creations and the swipe features.
- The database structure will then be tackled, looking into making my database work seamlessly with my application
- Add a live chat feature
- Then, integrate various APIs and libraries that are necessary for my project.
- I will also perform testing at various stages of the app development including unit testing, integration testing, security testing and user acceptance testing, ensuring my app's bugs will be fixed and the functions work as expected and are of quality standard.

To break down my requirements I will:

- Prioritise the features with the most importance such as user profiles, swipe feature and chat components.
- I will break down requirements into user stories and epics which will aid in describing functionalities from a user's perspective.
- I will then focus on dividing these user stories and epics into smaller development tasks.
- As I will be using agile development methodology, I will organise my development tasks into sprints for timeframes and objectives.
- I will also define the project milestones corresponding with the Moodle milestones. My project's milestones will represent my key phases of development.

Technical Details

For my iOS app development, I will be coding in Swift. Although Python may be more attractive on my CV for potential employers in the future, I always had an interest in iOS applications and how they were built. With Python, I only know the basic syntax and it will necessitate anyways to learn from scratch to code an application for Python or Swift. Adding onto this I have looked at using Python for iOS and it is more difficult as I will have to figure out how to do iOS development in a language that isn't Swift or Objective-C and outsource many third-party libraries and APIs that Apple already have incorporated within XCode. There is also a big gap in resources on Stack Overflow, for iOS and Swift tags there are over 184,000 questions asked whereas on iOS and Python tags there are only 850 questions asked. It made more sense for me to choose Swift over Python though if it was easier to use Python for iOS development and didn't have to jump over so many hurdles I would have chosen Python.

Algorithms, APIs, principal libraries and approaches for my app:

- SwiftUI is Apple's framework for building UI. It simplifies the UI development and allows it to be more interactive and user-friendly.
- As my database, I will be using Firebase Realtime Database because it works well with unstructured data and real-time updates.
- I will also put in a search algorithm to help users find profiles within my app.
- For push notifications, I will use Apple Push Notification Service to send out notifications to iPhones.
- To connect entrepreneurs, investors and business-minded people I want to implement recommendation algorithms. Collaborative filtering, and content-based filtering which identifies users who share similar business interests.
- I will use WebSocket/Firebase to ensure messages are delivered instantly within my chat feature.
- I will use Apple sign-in to allow users to sign in with their Apple ID.

Special Resources Required

Special resources I believe will be necessary for my app-building:

- XCode will be the IDE that I will be working with over the coming year to develop my application.
- Resources where I can learn Swift. An example would be Swift Playground where you learn how to code by moving around an animated character and completing various tasks using code.
- Plenty of YouTube tutorials.
- Red Bull and plenty of finger exercises for all the code I will be writing ☺

Project Plan

My project plan with key implementation steps and timelines is below and split into phases.

Phase 1: The Beginning (October 30th - November 10th)

- ☐ Define my project scope and go into detail when planning my project.
- ☐ Learning about Swift in detail
- ☐ Gather up the requirements that are necessary.
- ☐ Developing a high-level design and feature list.
- ☐ Create my sprints on Azure

Phase 2: Design and Prototyping (November 11th – November 15th)

- ☐ Create mock-ups for my application using the Mockplus website
- ☐ Define the app's flow and architecture

Phase 3: App Development (November 16th – April 15th)

Frontend Development (November 16th – February 28th)

- ☐ Begin my coding journey using Swift
- ☐ Implement User Registration and User Profile creation
- ☐ Implement Log-in Feature
- ☐ Develop the Swipe function
- ☐ Use prototypes as a guide to designing app
- ☐ Implement testing

Backend Development (January 1st – March 31st)

- ☐ Develop server-side components and APIs
- ☐ Implement database and data storage
- ☐ Work on real-time communication features
- ☐ Conduct Testing

Algorithms and Recommendation Systems (February 1st – March 31st)

- ☐ Develop algorithms
- ☐ Develop the recommendation systems for swipe user recommendations

Final User Testing and Quality Assurance (April 1st – April 20th)

- ☐ Continuously testing and debugging my app
- ☐ Address user feedback and bug reports
- ☐ Optimise app performance and user experience

Testing

In order for my application to run correctly and meet user expectations I will do the following:

System Testing:

- Functional Testing - Ensure my app's features work as intended. The registration, login, profile creation, swiping, real-time chat and recommendation algorithm. Correct gaps or errors.
- Performance Testing – Evaluate the app's performance under various conditions such as network speeds. Measure response times and make sure the app runs smoothly.
- Security Testing – Identifying security vulnerabilities, user authentications and protection from unauthorised access.

Integration Testing:

- Backend Integration - Test the front end and back end of the app. Make sure the data is correctly transmitted, stored and retrieved within the database. Test the real-time chat functionality and recommendation algorithms to confirm seamless integration.
- API Integration – Ensure the APIs are correctly integrated into my app without any issues.

End-User Testing:

- To conduct end-user testing I will share my app with my close family and friends and they will be entrepreneurs for the sake of my app.
- They will then be given realistic scenarios and tasks to complete in my app.
- I will observe and then gather feedback on usability, user profiles, swiping feature, navigation, design and overall experience.
- The users will also report if any bugs have been discovered.
- With their feedback, I will improve the app.

6.1 Ethics Approval Application (only if required)
6.2 Reflective Journals

Supervision & Reflection Template

Student Name	Cristina Berlinschi
Student Number	x20409746
Course	BSc in Computing
Supervisor	William Clifford

Month:

What?

Reflect on what has happened in your project this month?

This month I worked on my application proposal and came up with the objectives, features, technical approach and so on. I also have perfected my idea for my application and submitted a video proposal of me describing the app in detail. I believe in my app and I can see its potential.

I have also decided on an app name after rigorous searching and thinking for a unique simple app name.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

Completing this app proposal gave me a clear and concise idea of what my application is set out to achieve and also a plan for getting to the finish line. Now I know what I have to do in the next few months in order to achieve a successful application. The clearer the path is, the easier it is to build my app.

My current challenge at the moment is creating an effective UI for my application. I have looked at a myriad of examples and I am confused

about how to design my app. I want my app to have an effective UI that focuses on enhancing the user experience through user-friendly layouts, intuitive navigation and seamless integration of colours, shapes, fonts and imagery. I understand the importance of a good quality UI for the user as this can increase the number of users engaged and overall satisfaction with my business app. Knowing all of this I'm struggling to incorporate this into a UI. I have designed a few sample designs but nothing is up to my standard yet.	
Now What? What can you do to address outstanding challenges? To address my challenges I will keep working on the UI until I am satisfied with my results. I will watch more YouTube tutorials in detail and keep looking up other apps that have an effective use of the UI and apply similar techniques to my app. Another way to address my challenge is to reach out to people with experience in building apps on LinkedIn and ask for advice.	
Student Signature	Cristina Berlinschi

Supervision & Reflection Template

Student Name	Cristina Berlinschi
Student Number	x20409746
Course	BSc in Computing
Supervisor	William Clifford

Month:

What?

Reflect on what has happened in your project this month?

This month within my project I encountered delays in terms of coding due to assignments and a quiz to study for. I have broken up my project plan into sprints, making my project seem a little less intimidating and the hurdles a little bit smaller that I have to jump over.

Due to not going into enough detail about my competitors in my project proposal, I created a word document followed by a table showcasing my competitors features within their business networking application and then compared to mine I displayed my features and how my app is innovative.

I have also written up User Stories for each of my features within my app. These are short simple descriptions of a feature from a perspective of the person using an app. From my perspective the users using my app is investors, business minded people and entrepreneurs. I enjoyed this part as I could actually be in the users shoes and view the features differently as a software engineer.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

In terms of project progress I must admit I still have a long way to go but I will not let this discourage me and continue pushing forward. I have my sprints recorded and have a clearer vision of what way to code my application. I believe in the month of December I will complete a significant part of the project.

My main challenge this month was not finding the time to begin coding and prioritising my other assignments instead of prioritising them all equally and creating a study plan. I see now how I can improve for the month of December.	
Now What? What can you do to address outstanding challenges? This month I will work towards my mid-point proposal, having a functioning prototype showing how a user will interact with my app, having at least one core element implemented, and last but not least have a video presentation. I will continue to follow my sprints. The main core element I would like to focus on is having the app skeleton UI coded and focusing on the sign up and log in feature.	
Student Signature Cristina Berlinschi	

Supervision & Reflection Template

Student Name	Cristina Berlinschi
Student Number	x20409746
Course	BSc in Computing
Supervisor	William Clifford

Month:

What?

Reflect on what has happened in your project this month?

My project has kept me incredibly busy this month, even with a full schedule of college obligations. Although the time limits, I was able to include important functionality into my software, such as the ability to sign in and log in, as well as a simple user profile that showed names and initials. An important step that improved the app's backend functionality was integrating the Firebase database. I managed my tasks using Trello the entire time, which helped me keep the project on schedule.

However the month came to an end on an upsetting note. My app unexpectedly crashed right before I finished a showcase video, and functionalities that had been working before stopped working. There was not enough time for debugging because this issue surfaced too close to the submission deadline. It was an annoying time, particularly when the software had only been available for a few hours previously. Though I'm worried about how this might impact my grade, I'm still optimistic that my prior progress will be acknowledged.

As I think back on these encounters, I've discovered how unpredictable the field of software development can be. The trip this month served as a reminder of the value of resilience, extensive testing, and backup plans. I'm proud of my accomplishments and the learning curve I've climbed despite the setbacks.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

I had great success this month in the features I implemented. A significant achievement was the seamless integration of essential features like the login and sign-in functions. These are essential elements that improve the app's security and user interaction. I created a basic profile functionality where the name, email, initials, log out and delete account is shown on the page. Another success was integrating Firebase onto the app's backend. This enhanced the application's capacity to handle data and gave me real-world experience interacting with a popular database system. I found it was quite successful to use Trello for task management. Despite a demanding academic schedule, it assisted me in staying organised and on track with the project's progress, ensuring that important objectives were reached.

The software unexpectedly crashing right before the demo video was the biggest setback, I found it quite upsetting. This problem, which surfaced at a crucial moment, made clear the necessity for more thorough testing and debugging procedures. It's still difficult to juggle project development with other academic obligations. There was barely enough time for in-depth project work because of the strict deadlines for undergraduate project. Although I've made progress, I still have a learning curve, particularly when it comes to understanding Firebase's nuances and making sure the app remains stable in a variety of scenarios.

In conclusion, the difficulties faced have highlighted the significance of ongoing learning, rigorous testing, and efficient time management, yet I am happy with the progress made, particularly in terms of feature development and learning new technologies. These events have had a major impact on the way I approach software development and will dictate my work in the project's future stages.

Now What?

What can you do to address outstanding challenges?

Enhanced testing, better time management, and continuous learning are necessary to address the unresolved issues in my project and develop my skills in swift. To more efficiently find and fix errors, I intend to put in place a strict testing regimen that includes both unit and integration tests. It will also be essential to use the debugging tools in XCode to identify the underlying reasons of any problems.

Setting priorities and dividing up my workload into smaller, more doable portions will be essential to managing it. Even with a hectic academic schedule, setting out dedicated time blocks for the assignment will guarantee steady progress. Furthermore, it is imperative that I improve my technological abilities, especially in areas such as Firebase. Time must be set up for studying using internet resources and interacting with developer communities to gain knowledge and solutions.

Lastly, seeking feedback from my mentor, and being open to adapting my approach based on this feedback, will provide fresh perspectives and insights. This can be balanced with required breaks and self-care to help sustain productivity and avoid burnout. I want to raise the standard of my project and advance my abilities as a software developer and my skills in swift by approaching these problems in a sensible way.

Student Signature	Cristina Berlinschi

Supervision & Reflection Template

Student Name	Cristina Berlinschi
Student Number	x20409746
Course	BSc in Computing
Supervisor	William Clifford

Month:

What?

Reflect on what has happened in your project this month?

This past month I have been primarily debugging and fixing my issue that I had in December. My app previously crashed and I was unable to open my XCode properly and also the emulator would not run. It felt as if I was a car stuck in the mud and no matter how much I tried to drive the car, the wheels would spin but the car would not move. This is how it has felt with my issues with my app, no matter how much I tried to fix, the app would not be fixed and remain in the same place as the car even though there was work being done in the background similar to the car wheels. Debugging has been a significant and time consuming part of January trying to ensure my functionalities work as I expect. Due to the holiday season I did not have the opportunity to discuss this with my mentor and this time it meant I had to rely on myself on my problem-solving skills and resources.
I also reviewed my Trello board and my plan for completion of this project and have added some tasks.

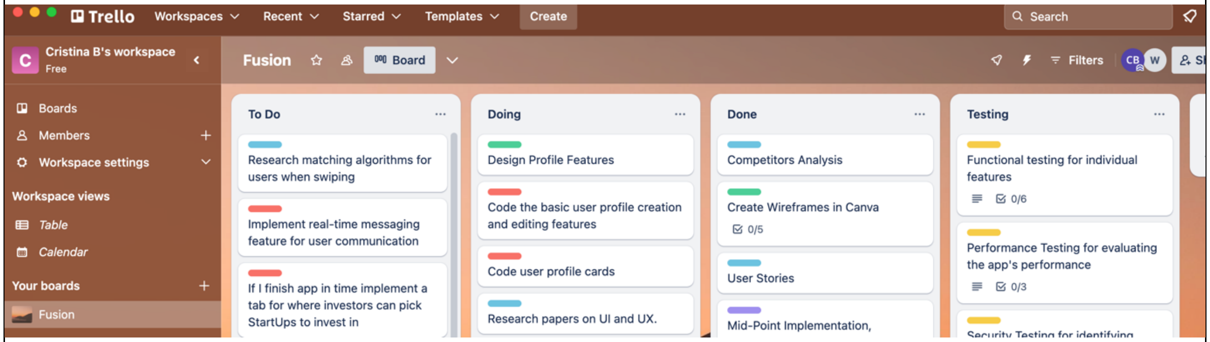
So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

While this month has been dedicated to debugging and troubleshooting, it is crucial to Fusion being reliable and performant. From March onwards I will focus on continuing on my plan I have set out and my tasks on Trello.

Moving forward I will implement new features and functionalities. I am working on coding the basic user profile creating and also the editing aspect. I am planning on implementing the profile cards along with this as this is the last piece of the user's profile.

Below is a sample of some of my tasks written in Trello. I like that on Trello I can create various tasks, put them in various boards such as to do, doing and so on, I can colour coordinate them and also write further details once



To Do
- Research matching algorithms for users when swiping
- Implement real-time messaging feature for user communication
- If I finish app in time implement a tab for where investors can pick StartUps to invest in

Doing
- Design Profile Features
- Code the basic user profile creation and editing features
- Code user profile cards
- Research papers on UI and UX.

Done
- Competitors Analysis
- Create Wireframes in Canva 0/5
- User Stories
- Mid-Point Implementation,

Testing
- Functional testing for individual features 0/6
- Performance Testing for evaluating the app's performance 0/3
- Security Testing for identifying

you click into the tasks. It aids in keeping my mind in order and the obstacles less big.

As for challenges I may have a classic case of leavingthingstolastminute syndrome and I am constantly reminding myself that this is not like most CAs that can be completed last minute and I have to continue to push to make significant additions to my application. Below I have explained how I have addressed and will continue to address this situation.

Now What?

What can you do to address outstanding challenges?

I believe by setting priorities and dividing up my workload into smaller, more doable portions will be essential to managing Fusion, I can see how Trello is helping me and will continue to do so and also continue creating designated time blocks within my week to focus on building Fusion. I believe in having my weekly calls with my mentor is also helping me in keeping myself accountable alongside my tasks I have set out.

The difficulties I faced over the holidays, such as my mentor and I not being able to communicate as much, has allowed me to understand how important it is to be flexible and independent when faced with unforeseen situations. It emphasised how crucial it is to have the ability to recognise and address problems inside the project on your own.

Supervision & Reflection Template

Student Name	Cristina Berlinschi
Student Number	x20409746
Course	BSc in Computing
Supervisor	William Clifford

Month:

What?

Reflect on what has happened in your project this month?

The month of February has been an exciting and fruitful for me. I have added more aspects within the profile, such as the ability to add a profile picture by camera or library, the ability to pick a business field, to write a bio, to add the company you work. Then I added in the home page where by using a library I added profile cards that can swipe. I adjusted the code till I had the profile card that looked like my desired image in my head. The profile card takes up most of the page and has buttons at the bottom of an x and a thumbs up. The ability to swipe is also there but I have yet to finish the backend in March as the buttons and swipe feature don't have a function.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

I consider this month successful compared to the previous months as I made a lot of progress within my app and looking back I am proud of my work that I have completed so far, it gives me the confidence of what else I could do in the remaining 2 months or so. This means I can focus on the algorithm, on the messaging part of the app and many more.

A challenge that still remain is figuring out how and which algorithm to implement for helping users for my app to recommend suitable users for each business individual. I also have yet to conduct proper testing within my app, I have conducted basic testing but I have yet to conduct more testing based on the features I added. Some testing I want to conduct is Unit Testing, Integration Testing, Security Testing, White box testing, Black box testing and so lots of other testing.

Now What? What can you do to address outstanding challenges? To address the algorithm challenge, I have to continue my journey of putting on my investigator hat, investigating various algorithms and which would be best to apply for my app. Once I find one suited I will learn how to code the algorithm and implement it for the swiping feature. In order to conduct to conduct sufficient and proper testing I will come up with a test strategy and test plan and follow it till I submit my app. I have many different tests I want to accomplish for Fusion and with a test strategy and plan I will be able to achieve this and hopefully submit an app error and code free.	
Student Signature	Cristina Berlinschi

Supervision & Reflection Template

Student Name	Cristina Berlinschi
Student Number	x20409746
Course	BSc in Computing
Supervisor	William Clifford

Month:

What?

Reflect on what has happened in your project this month?

March has been full of coding, bugs and mini accomplishments. This month I have done a substantial amount of work for Fusion. I added in my first algorithm which was confusing at first but with a lot of trial and error, I added in Geohashing. I used a library from GitHub. The way the Geohashing algorithm works is that the app Fusion gets your geographic coordinates (longitude and latitude) and turns them into a short string of digits and letters. To get the user's data location I edited the privacy permissions in Fusion in order for the app to allow me to do this, and the user gets a popup on their phone asking to share their location. Using the user's coordinates Fusion creates a 1000km radius from you and checks what users are in range from you recommends people within the swiping feature that are also in within your radius. I have also set this up for Fusion to grab the user's location every time the app is opened. I done this because I know business individuals, entrepreneurs, investors etc travel a lot and if you are in a new country and would like to connect with new individuals you can be rest assured that you can connect with new users everywhere you go. I created mock users in order to test this and they show up within my swiping feature. I also created a little capsule to the right-hand side of the profile card that shows how many km away the user is. I also added in a neat little feature where the user can switch between dark and light mode app in the settings. This wasn't too difficult as iOS has this built-in but I had to change some colours that would better suit the app than what was being shown within my simulator. The last thing I added in was a SessionState file, which I quickly learnt is necessary when building apps that use accounts. A SessionState file tracks the user's user session, whether or not they are logged in and if they are, their account should remain logged in.

Near the end of the month I began focusing on testing more, just as Frances says ‘grabbing those low-hanging fruits’ and this is what I did. So far I created a few unit tests for the home view model, registration view model and also login view model. At first, some of the tests failed but after I changed around some code, I had to think in units and structure some of my code that way the unit tests ran with success. This is where TDD (tested driven development) shows how important it is to be aware that you will be testing and structure your code in such a way that it will be easier to test.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

My main success was my perseverance to code almost every single day for a few hours, out of all months I consider I done the most progress and I am honoured of my accomplishments so far. It’s a success getting so much done and having around 60-70 percent of the app done. Another big success is my algorithm working successfully as at the start I had a few doubts as at the start I was getting a lot of errors and bugs within my code until one day it worked.

My next few challenges that remain ahead is to persevere with Fusion and continue with what I put in plan. I want to further tweak my swiping feature, and add in a button for filtering desired business fields, I want to add in a messaging feature, notifications and further continue on testing, conduct more unit testing, performance testing, security testing and so forth.

Now What?

What can you do to address outstanding challenges?

To address my outstanding challenges, I will continue putting in the hard work. I follow my test strategy and further continue researching and coding to add in the last 30 per cent of Fusion. I have lots of YouTube tutorials and various websites that I can follow. I believe having a good attitude and sticking to a plan is important and that is what I will do to pass the finish line.

Student Signature	Cristina Berlinschi

Supervision & Reflection Template

Student Name	Cristina Berlinschi
Student Number	x20409746
Course	BSc in Computing
Supervisor	William Clifford

Month:

What?

Reflect on what has happened in your project this month?

The month of April felt as passed by so quickly, so much to do and deadline quickly rising above the horizon. The stress was a lot this month, but that isn't necessarily a bad thing, I used it to fuel me to work on this project and making significant developments within my app. April started off with creating a messaging feature. The messages are stored in Firebase's Database and uses chat threads to send and receive messages but also keep track and store. The amazing thing about firebase is it provides real-time synchronization meaning that when a user sends a message the other user can see it immediately.

I also began creating and growing my mock users data and implementing my recommender algorithm. There is a lot of repetitive tasks involved but I am aiming to reach grow with more users. The reason for this is I have implemented a second algorithm. The second algorithm is a recommender algorithm that uses linear regression. In order for my algorithm to work well I need a big dataset because as I train the model and give it more users, the more training data there this and the more realistic the outcomes are. The input is the user swiping on the users they like and the output is the recommender algorithm. Every swipe the user does the algorithm is improved and recommends more users. The algorithm recommends based on users business field preferences and also location.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

My main success was my perseverance to code almost every single day has continued into the month of April. April ticked a lot of my boxes and I have made steady progression towards almost completing my project. My success also include my ability in being able to implement the features I have set out and following my plan.

My next few challenges that remain ahead is to finish the rest of the tasks I have set out by the 12thof May. The deadline is tight but I believe it is manageable and I am confident I will get it finished in time. I will like focus on further refining my algorithm. I will also work on adding some UI features to improve the appearance of my app, and conduct more testing such as integration testing and system testing.

Now What?

What can you do to address outstanding challenges?

In order to address my remaining challenges I have to continue coding everyday conduct more testing and finish off my report. My good attitude and mindset is what has got me so far and it is what will get me past the finish line. For my algorithm I will create more users and grow my database even more to improve algorithm's performance. I also want to add in some UI changes onto the home view page where the algorithm will be shown. For the UI I will follow my wireframes and establish a attractive UI within Fusion. With integration testing , once I implement it, I will test how the various components of my app interact with each other helping identify and resolve issues with interactions between UI, database and my two algorithms. I will also address implementing system testing, this will evaluate my testing as a whole, simulate real-world usage and help in identifying potential issues.

Student Signature	Cristina Berlinschi

6.3 Invention Disclosure Form

6.4 Other Materials Used