MMUCORD – AN INSTANT MESSAGING PLATFORM WITH EMBEDDED ANALYTICS

YOSUF MOHAMED HASSAN

PROJECT

MASTER OF COMPUTER SCIENCE IN SOFTWARE ENGINEERING AND SOFTWARE ARCHITECTURE

MULTIMEDIA UNIVERSITY

September 2021

COPYRIGHT

DECLARATION

I hereby declare that the work has been done by myself and no portion of the work contained in this Project has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

_____

Yosuf Mohamed Hassan

ACKNOWLEDGEMENT

First and foremost, thanks to Allah SWT for the grace, inspiration and will he gave me to persevere and complete this project. Then, I would like to sincerely thank my supervisor Dr Lim Tek Yong for his kindness, support and guidance during the lifetime of the project. Moreover, I would like to thank my parents for their prayers and endless support during my project, not to forget my brother Omar for his support and contribution to the project with his comments and suggestions regarding the user interface design. Next, I would like to thank my friends Mohammad Badawi and Mohammad Jawdat for their help and contribution during the report's writing by proofreading. In addition to their efforts in the development process of the prototype by suggesting, commenting and participating in the surveys and questionnaires I conducted. Moreover, I would like to thank my peers and friends who participated in the testing and evaluation phase of the prototype. Finally, I would like to thank Multimedia University's faculty and administration for all the support they provide me and my colleagues to accomplish our work and acquire our degree.

DEDICATION

To my role model, Dr Said El Naffar, the professor who believed in me and assumed the role of my mentor and guided me throughout both life and academia. I dedicate this project to you as a way to express my gratitude. I hope when you read this you are proud of what your student has accomplished.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Introduction

The recent COVID-19 pandemic forced people to spend most of their day at home. As the pandemic grew stronger, governments started enforcing strict lock-down policies that forced its residents to stay home full-time. Due to this change in circumstances, businesses had to alter their operations to suit a Work-From-Home environment. Several businesses achieved this by adopting a digital suite that combines several online-based tools to create an ecosystem to accomplish business requirements from a Work-From-Home environment.

The exponential increase in COVID-19 cases led to the Malaysian government introducing a nationwide lock-down. The lockdown policy forced Multimedia University(MMU) to shift their education to the remote learning model. MMU focused on perfecting their technology suite to provide the best coherent and efficient remote learning experience possible. MMU uses video conferencing solutions to deliver classes, a learning management system to distribute learning materials and administer examinations. This change allowed students to pursue education from the comfort of their own homes. Since the shift to online education proved to be successful, MMU dedicated resources to open an online-only version of their existing programs that will continue to operate in parallel to their in-person programs, even if the country opens up later.

1.2 Problem Statement

Prior to the lockdown enforced by the Malaysian government (MCO), when a student needed assistance, he knocked on the door and the student was then greeted by the department representative. If the representative is free, he or she will see to the inquiry of the student right away; otherwise, the student would wait in a queue while the other students ahead in line were served. In such scenarios, the student was able to have a synchronous conversation with the department ambassador; both parties in the conversation were able to ask questions and get the response immediately, which allowed either party to seek further clarification on the topic or to direct the conversation to a different inquiry.

Post Lockdown, the area where poor communication is inimical especially in the administrative department because it is the only communication proxy between the students and the university. Most of the interactions between the administration department and the university students revolve around visa renewal, class registration, and finances. Not only are these tasks and processes time-critical, but these tasks require thorough understanding and effective communication in order to avoid misunderstandings and wasted time.

The administration department of MMU shifted their entire operation to use email for communication. Email, despite being the standard for business communications, is extremely slow and it does not suit the kind or criticality of the tasks that the administration department needs to accomplish.

The switch to email will result in an execrable experience for both students and the administration department of MMU. Conversations via email are slow for various reasons: the nature of emails do not elicit a fast response and people neglect their email inbox either by not checking it regularly or by disabling desktop notifications. This results in a slow response from both parties. A slow response can become detrimental to efficiency in cases where the student has an issue that is complicated and the ensuing conversation would span many emails to clarify since he has to wait between emails that are sent back and forth between him and the administration representative. This standby time can be detrimental for cases that are

time-critical like Visa renewals or class registrations. Moreover, the slow response when using emails induces impatience since the student would think that he is being ignored or his issue is a low priority to the administration department. For a new student who is unfamiliar with university regulation and policy, such a feeling of being a low priority can leave a bad impression and damage the reputation of the university. Finally, another drawback of using emails is that the conversations are cluttered and disorganized, which means revisiting an email conversation for review purposes or looking up information would prove difficult for the students and the administration staff.

1.3 Proposed solution

The goal of this project is to create a platform, which connects MMU' community together, named MMUCord. MMUCord is a platform that is tailored specifically for the usage of the administration department. MMUCord allows the students and the administration department to communicate in real-time using only their name or the university ID as the user identifier, instead of using an email or a phone number like traditional instant messengers. MMUCord will allow both parties in a chat to exchange text messages, documents, emojis and voice notes. In addition, it allows both parties in the chat to react to messages using different reactions that suit a formal communication or environment, instead of the usual reactions used in social media. These reactions include the 'Ok' reaction which signifies agreement or understanding of the message body. The second reaction is the 'Pending' reaction which signifies that the process mentioned in the message body is pending approval. The third reaction is the 'Finished' reaction which signifies that the process or document in the message body has been completed.

MMUCord introduces embedded analytics to provide features that more suit the education environment. Using a combination of both descriptive and semantic analytics MMUCord provides an elevated experience when it comes to the communications between the students and MMU's administration department. On each MMU administration department employee's profile page different descriptive

metrics appear that aid the student in choosing which employee to talk to. In addition, using semantic analytics MMUCart provides features like topic and sentiment analysis where MMUCart analyzes the chat and defines a general topic and sentiment.

1.4 Project Objectives

This project has four main objectives, namely:

1. Identify requirements for an instant messaging application
2. Develop the instant messaging application
3. Evaluate the instant messaging application
4. Implement Embedded analytics features and integrate them in the messaging system

1.5 Project Scope

The MMUCord project has a set of deliverables and constraints that will affect the project progress and the final system.

1.5.1 Deliverables:

a. Interim Project Report

The interim report is the initial version of the project report to be submitted in the first part of the project. The interim report focuses on pitching the project idea, defining the scope and the constraints imposed on the project be it by the author/s or outside factors outside the hands of the author/s. The interim report also discusses the literature review that supports and validates the project purpose. In addition, the interim report outlines the software development methodology used to achieve a final working proof of concept prototype.

b. Initial Prototype

This is an initial prototype to be displayed during the discussion of the first part of the project. This prototype is created to illustrate the base messaging

system that will house the analytics engine/service in the future. The initial prototype illustrates how different parts of the messaging system functions, including, sending/receiving messages, read receipts, registration and status and typing indicators.

c. Final Project Report

The final project report is a continuation of the interim report and serves as a reference and documentation for the project. The final project report outlines all the phases of the project from its objectives, planning, implementation and testing.

d. Final Prototype

The final prototype is a proof of concept prototype illustrating the entire system including the base messaging system with the embedded analytics engine. The goal of the prototype is to show that such a system is possible and is a viable solution to the given problem statement.

1.5.2 Time Constraints:

All the deliverables of this project needs to be completed within six months. This includes the interim report, the final report, any revisions suggested by the moderators, and the proof of concept prototype. Therefore, the project may have some limitations due to lack of time as outlined below.

The allocated time for the project only allows for the evaluation and requirement gathering to only be done in coordination with the Faculty of Computing & Informatics (FCI) administration only. Moreover, due to time constraints the final prototype only includes two out of the four different types of embedded analytics as was originally intended, however, those features that were not implemented in the prototype will be implemented post submitting this project at a later date.

1.5.3 Resource constraints

As a student, I lack the resources and funding a corporation might have access to. This fact does come into play in some aspects of this project.

There shall be an android application for MMUCord. Since I do not have access to a Mac computer or an IOS device for the development, testing or verification of the iOS build.

Due to the specifications of the machine I use for the development of the final proof of concept prototype (Lack of sufficient system memory) I was only able to test the application on two or three emulators simultaneously.

1.6 Conclusion

In the first Introduction chapter, it was illustrated that the COVID-19 circumstances led MMU to shift its operations to suit an online learning experience. In addition, the problems the shift to a fully online operation has given rise to was discussed in detail to provide a complete view of how this shift has affected the students and staff of the university. After discussing the circumstances and its after effects, a solution was provided in the form of the analytics powered instant messaging platform which gave birth to the MMUCord project. Following the birth of the MMUCord project, its objectives, scope and constraints were all defined and outlined in the first chapter.

CHAPTER 2

RELATED WORK

2.1 Instant messaging and its benefits

In this section, instant messaging features will be discussed and how these features will benefit MMU's administration department when an instant messaging solution is deployed throughout the department.

The first implementation of instant messaging was created to provide a better alternative to Short Message Service (SMS). At a basic level, instant messaging is a form of synchronous communication that allows the exchange of text messages between two parties in real-time over the internet. The first modern dedicated online instant messaging platform was ICQ released by Mirabilis in 1996 (ON, 2021).

Modern instant messaging facilitates a more streamlined flow of communication. Today, instant messaging is popular and widely used to connect people across the globe. WhatsApp, the most popular instant messaging platform, has more than 2 billion active users in over 180 countries and 100 billion messages sent everyday (About WhatsApp, 2019). Instant messaging applications today are more versatile and support more than sending and receiving text messages. Modern Text messaging applications support sending and receiving a variety of media including photos, videos, documents, voice notes and location coordinates. In addition, modern

instant messaging applications now support emojis, stickers, reactions as well as voice and video calls.

In a post quarantine world, instant messaging is the perfect replacement for using email for business communications because it has many benefits that more suit the current circumstances. Unlike emails, instant messaging by nature prompts for an instant response. This is because instant messaging applications use notifications to alert the user when a message is received. Moreover, people check their phone more frequently when compared to their email inbox. This is evident when viewing the surveys conducted by reviews.org (Wheelwright, 2021) a popular technology journal and Twigby (Twigby) a popular American telecommunications service provider. The survey conducted by reviews.org stated that Americans check their phone two hundred and sixty-two times a day on average, that is once every five and a half minutes. Another survey, conducted in 2019 by Twigby, stated that smartphone users unlock their phone on average one hundred and fifty times a day. The instant response is crucial in tasks that are time-critical like visa renewal or class registration. In addition, the quick response is beneficial for questions that require rapid answers in order for a task to move forward. This is beneficial in cases where there is a task at hand and all the paperwork is ready but there is a small piece of information missing to proceed with the task, in such case the administration department is able to quickly message the student to retrieve the missing information and proceed with the task.

Other than the instant response time, instant messaging provides more that suits corporate communications. Instant messaging is more organized compared to email. Each chat is organized in a chronological timeline and is on a separate tab, unlike emails where all the conversations are in a single inbox, each including strings of emails attached. Instant messaging conversations being organized, it is easy to go back and for review purposes, information lookup or in case of a dispute. Instant messaging applications provide read receipts where users are notified when the other

party sees their message. Read receipts assurance to the student that the administration department employee saw his inquiry and is either busy or working on it.

The features provided by instant messaging applications help provide a pleasant experience to both the student and the administration department employee. In addition to boosting the efficiency of the administration department, improving the university's image and reputation.

2.2 Instant messaging applications

There are three main players in instant messaging, namely WhatsApp, Telegram and Facebook Messenger. These three main players focus on the western market of the United States and Europe. However, there are big players in the eastern market, Line messenger in the east asian market and WeChat in china. In this section, an outline of the main features of the main players in the instant messaging space will be provided. In addition, a comparison between the three of them to define how these features differ between them.

2.2.1 WhatsApp

WhatsApp was founded in 2009 by Jan Koum and Brian Acton who previously spent 10 years working at Yahoo. WhatsApp was acquired by Facebook in 2014 for 19 Billion dollars, ensuring that WhatsApp's acquisition becomes the largest acquisition for a venture capital backed company in history (About WhatsApp, 2019).

WhatsApp supports both group chats and ordinary chats. Members of a chat are able to send text messages, different media files like videos, pictures, documents and any other file type. In addition to media files, WhatsApp also allows the sharing

of stickers, emojis, GIFs and location, current or live. Also, WhatsApp allows users to record and send voice notes allowing users to adjust playback speed. Moreover, WhatsApp allows users to host group audio or video calls. When it comes to customizing the chat look and feel, WhatsApp is lacking in this department, it only allows users to change the chat background to an image or a solid color.

Stories were popularized by the social platform Snapchat. Users are able to broadcast an image or a short video file to all their contacts for 24 hours. WhatsApp implemented this feature in their application. However, WhatsApp allows users to specify the amount of time the story is kept alive.

WhatsApp allows their users to continue their conversations on the desktop, given that the phone registered to WhatsApp is connected to the same network as the desktop.

WhatsApp keeps track of the status of every user on the platform. The status provides a timestamp of when a user was last online. However, this is optional and can be disabled by the user.

In WhatsApp, read receipts have three different stages: sent, delivered and received. The sent stage is represented with one grey tick, that the message is sent to WhatsApp's server and will be delivered once the other party is connected to the Internet. The delivered stage is represented with two grey ticks, meaning that the message was delivered to the recipient but it was not viewed yet. The received stage is represented with two blue ticks meaning that the recipient saw and read the message. In a group chat, WhatsApp keeps a timestamp for each message revealing when each member received and saw the message. However, in WhatsApp read receipts are optional and can be disabled by the user.

2.1.2 Telegram

Telegram was founded by Pavel Durov who also founded VK Europe's popular social network. This is the reason Durow was given the nickname Mark Zuckerburg of Russia. In 2011, violent protests sparked in Russia, the protestors used VK to orchestrate the protests. Durov created Telegram as he fled Russia after his refusal to cooperate with the Russian authorities.

Telegram supports both group and ordinary chats. Similar to WhatsApp, users are able to exchange text messages. Telegram allows the users to schedule messages for a later time or send them without triggering a notification on the recipient's phone. In addition to text messages, Telegram supports the exchange of different media files, voice notes, location data and unsupported file format. However, Telegram does not compress any sent media files but gives users the choice to either compress the file or send it in its original quality. Moreover, Telegram also allows users to send stickers, emojis and GIFs. However, Telegram does support animated stickers and emojis.

Telegram allows users to host video and audio calls similar to WhatsApp. However, Telegram allows users to share their device screen to facilitate conference calls and team collaboration. Telegram provides a wealth of customization options when it comes to the chat look and feel. Users are allowed to change the text size, change chat background, change the color of chat boxes, chat color theme, animated chat background and chat corner size.

Telegram allows the users to continue their conversations on the desktop. However, Telegram does not require the phone to be connected to the same network. Allowing users to continue their conversations even if their smartphone is off.

Telegram provides a chat archiving functionality named chat folders. Chat folders allow users to put chats into different folders. For example, a user can have two folders work and personal separating chats related to both.

Similar to WhatsApp, Telegram status keeps track of every user and displays a timestamp of when a user was last seen on the platform.

2.2.3 Facebook Messenger A.KA Messenger

Originally developed by Facebook in 2008, after revamping the experience in 2010 and releasing a standalone application on both Android and IOS. Facebook Messenger now has more than 1 Billion active monthly users.

Facebook messenger supports both ordinary and group chats. Similar to both WhatsApp and Telegram, Facebook Messenger allows users to exchange text messages, different multimedia files in addition to stickers, emojis and location data. Facebook messenger allows users to react to different messages using different reactions expressing their feelings towards said message. Facebook messenger allows users to host group audio or video calls with the ability to share their screen similar to Telegram. In terms of customizing the chat look and feel, Facebook messenger allows users to only change the chat color, a step up from WhatsApp however, not as competent as Telegram's customization.

Facebook also implemented the stories feature in their messenger. Facebook allows users to broadcast an image or a short video for a specific amount of time specified by the user.

Facebook messenger was initially a web platform, therefore the user is able to continue his conversations on the desktop using the web platform without requiring a phone to be connected to the internet.

Facebook messenger tracks if a user is online or not. In addition to how many hours have passed since the use was last seen online on the platform.

Facebook messenger supports read receipts similar to Telegram and WhatsApp with three stages: sent, delivered and seen. Similar to WhatsApp and

Telegram, in a group chat Facebook messenger keeps track of when each member in a group chat received and read the message.

2.2.4 MMUCord - Proposed Project

The MMUCord final prototype includes the main features that together construct a real time messaging system. The messaging system of MMUCord enables the user/s to send and receive text messages in real time. Moreover, the messaging system supports online status/indicators, read receipts and typing indicators. In addition to the base messaging system MMUCord has an analytics engine, the analytics engine is responsible for the collection and processing of data going through the system for it to be used later to provide different types of analytics to the end users. The analytics engine collects data pieces like the chats, the messages sent, received and unread. How the data pieces are processed is to be discussed in the coming sections of the report.

2.2.4 Summary

When observing the three main players in the global instant messaging market. There are five main features that appear to be common across all the messaging applications. The first main feature is the ability to chat one on one or in a group chat. A chat allows the parties in it to exchange text messages and various forms of media in addition to stickers and emojis. The second main feature is the ability to engage in audio and video calls with facebook and telegram allowing for screen sharing to more fit the post COVID-19 pandemic era. The third main feature is desktop support, all three main instant messaging applications not only work on mobile but can be accessed via desktop as well; however, WhatsApp does require the user's mobile phone to be connected to the internet for the desktop application to work. The fourth main feature is read receipts, which allows users to track the status of the messages they send. The fifth main feature that all the main players share is status, this feature keeps track of all users in the platform, is the user online, is the user typing or recording audio and when was the user was last seen online on the

platform. After reviewing the three main players in the instant messaging space it is seen that MMUCord's messaging system is not far from WhatsApp, a major player in the instant messaging space. Moreover, MMUCord is a unique instant messaging application because of its embedded analytics engine and the features it provides to the end user. Table 1 below provides a comparison and an overview of how feature rich a platform is.

Table 1 - Comparison between instant messaging applications

Feature	**WhatsApp**	**Telegram**	**Messenger**	**MMCord**
Chats	Yes	Yes	Yes	Yes
Text Messages	Yes	Yes	Yes	Yes
Video,audio calls	Yes	Yes	Yes	No
Animated emoji	No	Yes	No	No
Reactions	No	No	Yes	No
Screen Sharing	No	Yes	Yes	No
Stories	Yes	No	Yes	No
Desktop Support	Yes	Yes	Yes	No
Chat folders	No	Yes	No	No
Silent Messages	No	Yes	No	No
Scheduled messages	No	Yes	No	No
Read Receipts	Yes	Yes	Yes	Yes
Status	Yes	Yes	Yes	Yes
Count unread	No	No	No	Yes
Count Sent	No	No	No	Yes

Count Received	No	No	No	Yes
Count Un-Seen chats	No	No	No	Yes
CountSeen chats	No	No	No	Yes
Charts	No	No	No	Yes

2.3 Analytics

In the past, oil was the most valuable resource on planet earth, and who possessed and controlled the flow of oil influenced the world. Nowadays, data is the most valuable resource on planet earth and whoever possesses it is able to influence the world. Companies today are in a data overflow state, where there exists a large amount of data residing in transactional databases, equipment log files, images, video, sensors or other available data streams. With the exponential increase in the amount of data available, the need to organize and process the data rose to the surface. This is where the field of Big Data comes into play, a field focussed on organizing and processing the large amounts of data available to extract value from it. Analytics is a subset of Big Data focused on using all the information available on a specific product to draw concrete conclusions and meaningful insights using real-time metrics related to the context of the product. The information is collected, segregated and then modeled using different data visualization techniques in an insightful manner to aid product owners to make an informed decision that is backed by data and meaningful insights. Modern analytics software tools provide more than data visualization, modern analytics tools provide real-time reporting, alerts and machine learning aided predictions (Gibson, 2018 and Rawat, 2021).

Traditionally, analytics capabilities are provided in a separate application far from the one being analyzed. The analytics being provided by a separate application means that only the admins responsible for the application can view any analytics

data and use it. This also means that any analytics data and visualizations are obfuscated from the users of the application.

A solution to such an issue is to use embedded analytics, the idea behind embedded analytics is to make the analytics data and visualizations available in the application itself, hence the name embedded analytics. Using embedded analytics will allow users of the application to view and visualize all the gathered analytics data effortlessly which will result in a better user experience increasing customer satisfaction.

There are four different types of data analytics that are in use across all industries, they are separated into different categories. However, they are all linked together and used in combination to achieve the goal of analytics. The four types of data analytics are descriptive analytics, diagnostic analytics, predictive analytics and prescriptive analytics (Gibson).

2.3.1 Descriptive analytics

Descriptive analytics is the foundation of all analytics, it provides an overview of an event or phenomenon. Descriptive analytics helps in the description, summarization and visualization of past data in the form of dashboards or charts. For descriptive analytics the data is first collected then sorted to produce a manageable dataset. There are four main measures in descriptive analytics. They are measures of frequency, measures of central tendency, measures of dispersion or variation and measures of position. Such measures help better understand the dataset, since it enables the detection of outliers, differences and similarities between different groups in the dataset (Business Analytics, 2019 and Rawat, 2021). Descriptive analytics can be implemented to the benefit of instant messaging applications in different scenarios.

2.3.1.1 Measures of Frequency

In descriptive analytics, it is essential to know how frequently a certain event or response is likely to occur. The purpose of such a measure is to create a counter or a percentage that keeps track of a certain event or action (Rawat). In an instant messaging application, there are a few measures of frequency that can be implemented to benefit organizations that use instant messaging in a business environment. The measures to be suggested are focused on the usage of users on the platform. A measure is created to count the number of active and inactive users on a platform. For each user on the platform, a measure to count the number of messages sent and the number of messages received each day. Such measures are used by WhatsApp for their business focused instant messaging platform WhatsApp Business. In a business's admin panel, a chart is created using the above mentioned measures. Another measure is the percentage of chats in a group chat. This measure can be used to determine how active a user is in a group chat. KIK, an American instant messaging platform uses such a measure to kick inactive users from public groups. In addition to these measures, a measure can be created to monitor each user's active time on the platform. A chart can be then created to visualize each user's daily active time.

2.3.1.2 Measures of central tendency

In descriptive analytics, it is essential to identify the central tendency, in other words the average. Central tendency is measured with the use of three different averages, mean, median and mode (Rawat). Examples of measures of central tendency that can be implemented to increase the benefits of using instant messaging applications in a business environment are measures that focus on quantifying and analyzing response and active time of a user on the platform. A measure can be created to find the mean, mode and median response time. The mean response time represents the average response time of users on the platform. The mean response time does not represent the actual average because the mean is affected by outliers. In instant messaging applications, there are always outliers, users who either respond instantly or users who always respond a day later. In such cases, the median response

time is more accurate when it comes to evaluating the average response time of users. The median response time is an average that focuses on the midpoint, better representing the average response time of all the users. The mode is the most common value in a dataset. In the case of instant messaging, using the mode average is not practical in the instant messaging use case. Therefore, it is practical to use the median average to compute the average response time. A chart is created to display the average (median) response time for all the users.

2.3.1.3 Measures of dispersion

In descriptive analytics, it is essential to identify how the data is distributed across a range. To measure data distribution different measures of dispersion can be employed, measures like range and standard deviation. In the case of instant messaging, two measures of dispersion can be implemented to improve the benefits of using instant messaging in a business setting. A measure of dispersion could be implemented to compare the response time of a user to the average (median) response time. Such a measure could be beneficial to evaluate the performance of an agent in a customer service center using an instant messaging application to communicate with clients. Another measure could be implemented to compare the active time of a user to the average active time. Another example that is related to the context of MMUCord when it comes to measures of dispersion is to measure and record the time it takes an employee of MMU's administration department to solve a student's issue. Such a measure could be used to evaluate and improve the student's experience.

2.3.1.4 Measures of position

In descriptive analytics, it is important to identify the position of a single data point or its response to other points in the dataset. Measures like percentiles and quartiles can be employed to identify the position of a single value in the dataset (Rawat). An example of such measures in an instant messaging application to find the time of day where the user is most active, this can be very beneficial in

companies that use their instant messaging service for collecting data to serve other products that use the collected data for targeted advertisements or a recommendation engine for other services provided by the same enterprise. This approach is used by LINE, an eastern asian ecommerce social platform. A feature that can be implemented based on the time stot the user is active in is the blue light feature, where a blue light filter is applied to the screen if the user is active from sunset to sunrise to protect the user's eyes.

2.3.2 Diagnostic analytics

Diagnostic analytics is concerned with finding the cases and the reasoning behind the insights interpreted from the finding of the descriptive analytics stage. Diagnostic analytics is usually performed by data analysts in four stages, data discovery, drill-down, data mining, and correlations. In the discovery stage data analysts will find and identify the data sources that will aid them in interpreting the insights found. In the drilling down stage, the focus is directed to one facet of data to interpret what it means and what it represents. Finally, Data mining and correlations, is a process of obtaining information from a massive set of raw data and finding consistent patterns and correlations, identifying patterns of behavior. Sometimes the data analysts have to look for patterns in external sources outside the company's internal database. An example where diagnostic analytics can be used with instant messaging applications in the context of the university would be to use the data generated from the conversations on the platform to identify the reasons of discontent and concern from the students. Identifying those reasons and then acting to eliminate them would improve the university's ranking and student satisfaction (Diagnostic Analytics, n.d and Gibson, 2018).

In the context of instant messaging diagnostic analytics may appear alien and serve no purpose. However, diagnostic analytics can be applied in a variety of ways. One of the scenarios that diagnostic analytics can be applied to is to find out why an individual is not replying on time or in the worst case scenario not replying at all. This can be achieved by analyzing active time patterns to determine the time slots

where the user is mostly active. Another way this can be achieved is to drill through previous messages with other users to find out the reason why he/she is not replying.

2.3.3 Predictive Analytics

Predictive analytics is a step up from descriptive and diagnostic analytics, an advanced branch of analytics that focuses on providing logical predictions about future outcomes using historical data. The analysis and forecasting of future outcomes is done with the aid of machine learning algorithms. These include linear and nonlinear regression, neural networks, support vector machines and decision trees. However, it is important to note that a prediction is only an estimate where its accuracy relies on the quality of the data provided to the machine learning algorithms (Gibson, 2019).

While descriptive and diagnostic analytics are widely used by many companies, predictive analytics is where many organizations show signs of difficulty. This is because predictive analytics require resources and manpower that most organizations simply do not have access to (Predictive Analytics, n.d).

Predictive analytics can be used in many scenarios to solve difficult problems and uncover new opportunities. Predictive models are used to help businesses attract, retain and grow their most profitable customers. Instant messaging platforms owned by large corporations Like Facebook use the data generated by users using their instant messaging platforms (Messages, audio, location and pictures) to train predictive models to be able to correctly identify current user needs and products the user is likely to purchase. The trained models and the data are used to market other services by the same company or serve targeted advertisements.

3.3.4 Prescriptive Analytics

Prescriptive analytics is the most sought after form of data analytics, however few organizations are equipped to employ. Prescriptive analytics provides the best

course of action to achieve business objectives like customer satisfaction, profits and cost savings. Prescriptive analytics systems use artificial intelligence and different optimization techniques to tackle complex business problems with a variety of decision variables, constraints and tradeoffs. An example where this can be used in the context of the university is when the data used from the chats with the administration team members is used to determine the classes to be enrolled for the next trimester. Another example would be to use the data generated to efficiently schedule and prioritize bug fixes for CamSys and MMLS based on the technical support tickets and issues received by students. Another scenario where prescriptive analytics could be applied is to tell the user when to message a certain person based on the collected active time slots. For example a student wants to message an instructor at 11:00 PM at night, when the student opens the chat between him and the instructor a popup appears telling the student that it is best to text the instructor at 8:30 AM since this is when he is usually active (Prescriptive Analytics, n.d and Gibson, 2019).

2.3.5 Semantic Analysis and Analytics

Understanding the meaning of text is a simple task for a human being, however, for a computer it is a daunting task. A computer has to first be trained to understand the human language and make sense of the context in which words are used. An example would be in a customer service center where a customer replies “This is a joke, I have been waiting here for 30 minutes.”, the word joke may be misinterpreted as positive if the context is not well understood (Wolff, 2020).

This is where semantic analysis comes into play, semantic analysis is the process where a computer or a machine analyses a piece of text to draw meaning from it. The process of semantic analysis allows computers to to understand and interpret text by analyzing grammatical structure and identifying relationships between words in the context of the text. Semantic analysis is essential in today’s world and is the driving force behind technologies like search engines, chatbots, and autocorrection(Wolff, 2020).

Lexical semantics is the science that is concerned with the relationships between different lexical items. Lexical semantics play a huge role in the semantic analysis process. There are six main relationships between words in the English language: Hyponyms, Meronomy, Polysemy, Synonyms, Antonyms and Homonyms. These six relationships are outlined below (Wolff, 2020).

- **Hyponyms:** specific lexical items of a generic lexical item (hypernym) e.g. An apple is a hyponym of fruit (hypernym).
- **Meronomy:** a logical arrangement of text and words that denotes a constituent part of or member of something e.g., a piece of an apple
- **Polysemy:** a relationship between the meanings of words or phrases, although slightly different, share a common core meaning e.g. I attended a class, and I registered a class)
- **Synonyms:** words that have the same sense or nearly the same meaning as another, e.g., sad, downhearted, glum, forlorn
- **Antonyms:** words that have close to opposite meanings e.g., happy, sad
- **Homonyms**: two words that are sound the same and are spelled alike but have a different meaning e.g., orange (color), orange (fruit)

There are other tasks within the semantic analysis process other than lexical semantics. Two of the main tasks in semantic analysis are word sense disambiguation and relationship extraction. Word sense disambiguation is the process of identifying in which sense the word is used according to the context of the text. For example, the word orange can refer to the color orange, the fruit orange or a city in Florida named orange. Another example would be the word date, it could refer either to the fruit, a particular date in the month or a meeting. Word sense disambiguation is responsible

for identifying which of these meanings are used in the current context. Relationship extraction is the process of identifying the relationship between different entities in the text. For example, the phrase “Steve Jobs is one of the founders of Apple, which is headquartered in California” contains three different entities, Apple the well known tech giant, California a place in the US and Steve Jobs a well known tech entrepreneur. The sentence also contains two different relationships: the first being that Steve Jobs is the founder of Apple and the second being that Apple’s head office is in California (Wolff, 2020).

There are two different categories of models available in semantic analysis depending on the type of information to be extracted from the text or the way the text needs to be processed. The two model categories are classification and extraction models. Classification models are used to cluster information available in a single text or cluster multiple text pieces into different categories. Classification models are also used to infer information from the given text. Classification models are of three categories, outlined below.

- **Topic classification**: sorting text into predefined categories based on its content. An example, a university technical support team receives support tickets from students through an instant messaging platform. Through semantic analysis, machine learning tools can recognize if a ticket should be classified as a “Visa issue” or a “Registration issue”.
- **Sentiment analysis**: detecting positive, negative, or neutral emotions in a text. In an instant messaging application sentiment analysis can be used on a conversation to identify if the overall tone of the conversation is positive or

negative. This can be used to infer student satisfaction and the quality of overall experience.

- **Intent classification**: Semantic analysis techniques can be used to predict the intention of a person. An example would be classifying text based on what customers want to do next. An example would be using the chat in an instant messaging application to determine a user's intention. For example, in the context of a university intent classification could identify if a student is willing or interested to register in the university.

Extraction models are used to extract information from the given text. This is useful in many scenarios like searching photos by a location or tagging customer support issues. Semantic extraction models are of two categories, outlined below (Wolff, 2020).

- **Keyword extraction:** finding relevant keywords and expressions in a given text. This technique can be used in an instant messaging application to extract keywords that provide insights on what the chat is about. In instant messaging, this can be used to automatically infer what this chat is about. For example, two people are chatting and the algorithm detects the following keywords [Visa, administration, EMGS, passport, letter], from the keywords it is easy to conclude that the chat was about a student that has issues with his/her visa.
- **Entity extraction:** identifying named entities in text, like names of people, companies, places, etc. In instant messaging, it is possible to extract entities mentioned in the chat, for example in the context of a university, entity

extraction can be used to extract entities like professor names, class names, class codes, student IDs, etc.

2.3.6 Table 2 - Compare different analytics approaches

Type of analytics	Explanation	Examples
Descriptive analytics	Provides insights based on previous accumulated information	Average active time for a user
Diagnostic analytics	Determine the cause of previous results	Find out why a person is not replying to the message
Predictive analytics	Provides a look to the future by predicting what is going to happen based on previous collected data	Use the data collected to train predictive models to predict user messaging habits to adapt the experience to them
Semantic analytics	Provides a way to understand and provide context to a piece of text	Tag messages into different groups
Prescriptive analytics	Provides foresight to the future, assists in choosing the best option to achieve the desired goal	When to message a certain individual based on active time

2.3.7 Flow of Analytics

The four main types of analytics usually work in tandem with each other taking the findings of one step as the input for the next to create a complete analytics ecosystem. This approach aims to create a cohesive user experience. To achieve such a flow a scenario is crafted, the scenario defines the goal to be achieved by implementing analytics and the pieces of data that need to be collected for use in the analytics engine.

Post data collection the data collected are used to define the metrics used for Descriptive analytics by this stage Descriptive analytics is implemented since Descriptive analytics is only focused on displaying the metrics to the end user.

In the second phase is concerned with Diagnostic analytics, in this phase the metrics collected are processed to identify cause-effect relationships between the metrics which will in turn identify reasons for certain behavioral patterns. By the end

of this stage the Diagnostic Analytics part is done since Diagnostic analytics is concerned with identifying the cause of behavioral patterns.

In the third phase the extracted relationships and behavioral patterns from the previous stage are used as an input for a machine learning algorithm to predict when these patterns or behaviors may occur next. The machine learning algorithm type is selected based on the goal defined in the scenario. By the end of this stage the Predictive analytics section is complete since Predictive analytics is concerned with predicting when certain behavior or actions may occur.

In the fourth stage the predictions and findings of the machine learning algorithm are analyzed to suggest the best course of action to the user. By the end of this phase the Prescriptive Analytics section is complete. Since Prescriptive analytics is concerned with providing the user with a suggested course of action.

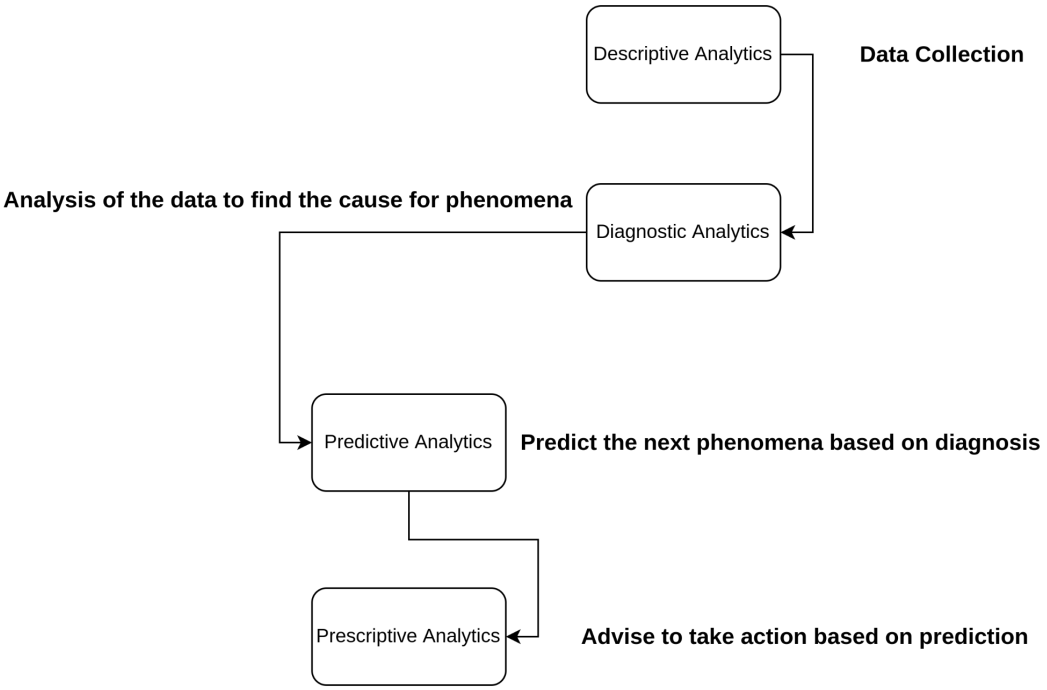

Fig 1 - Analytics Flow

2.3.8 Analytics in MMUCord - Crafting a scenario

The MMUCord system follows the same approach. In this section the scenario and the corresponding data pieces will be defined to outline how embedded analytics will be implemented in the MMUCord system. In addition this section shows how the data collected will be processed to implement the various analytics stages.

The goal of using embedded analytics in a messaging system is to show the student that the MMU administration is currently busy and that is the justification for the late response. Also, the system should inform the student that the admin employee is busy or provide sufficient metrics for the student to intuitively judge how busy the admin employee is. Moreover, the system should predict with fair accuracy when the MMU administration employee will respond. Finally, the system should suggest when to message a user based on the previous activity.

The first step is to identify and collect the data needed to proceed with implementing the embedded analytics. The data MMUCord collects for the processing is as follows, note that the data collection and processing will be discussed in the System Design and Implementation section:

- The number of messages sent by a specific user (student or MMU Admin Employee)
- The number of messages received by a specific user (student or MMU Admin Employee)
- The number of chats for a specific user (student or MMU Admin Employee)
- The number of seen chats (chats with no unread messages) for a specific user (student or MMU Admin Employee)
- The number of un-seen chats (chats with unread messages) for a specific user (student or MMU Admin Employee)
- Active Times for a specific user
- Message timestamps

An example will be used throughout this section to illustrate how MMUCord implements embedded analytics, the example is as follows: The system has three users, **user A**, **user B** and **user C**. **User A** sends **three** messages to **user B** and **four** messages to **user C**. **User B** replies to **user A** with **seven** messages and **user C** replies to **user A** with **six** messages. Moreover, **user A** saw the messages from **user B** and did not see the messages of **user C.**

2.3.8.1 Analytics in MMUCord - Descriptive Analytics

At the core of descriptive embedded analytics is displaying metrics to the user, MMUCord as a system should display metrics that emphasize how a user (MMU Admin employee) is currently busy, in other words the system should show metrics that allow the student to visualize the current load on the MMU admin employee.

The metrics MMUCord will use are divided into two sections, the first section is responsible for tracking the number of messages sent and received by the user (student or MMU Administration Employee) and present them in a way that aids intuitive judgment. The second section is concerned with the chats as a whole where MMUCord tracks the individual chats and records how many of the chats are seen and unseen.

Based on the example illustrated earlier, if the system wants to generate the metrics for **user A** it would be as follows: For the first section related to messages, **user A** sent **seven** messages in total, **user A** received **thirteen** messages, **user A** has **six** unread messages.**user A** has a total of **two** chats, **one** with **user B** which is considered as seen since **user A** saw his messages however the second chat with **user C** is considered to be unseen since **user A** did not see his messages.

Based on the example illustrated earlier, if the system wants to generate the metrics for **user B** it would be as follows: For the first section related to messages, **user B** sent **seven** messages in total, **user B** received **three** messages, since **user B** received **three** messages the number of unread messages is expected to be **three**. **User B** has a total of **one** chat which is a chat with **user A** and the chat is considered **unseen** since **user B** has unread messages.

Based on the example illustrated earlier, if the system wants to generate metrics for **user C** it would be as follows: for the first section related to messages, **user C** sent **six** messages in total, **user C** received **four** messages, since **user C** received **four** messages the number of unread messages is expected to be **four**. **User C** has a total of **one** chat which is a chat with **user A** and the chat is considered **unseen** since **user C** has unread messages.

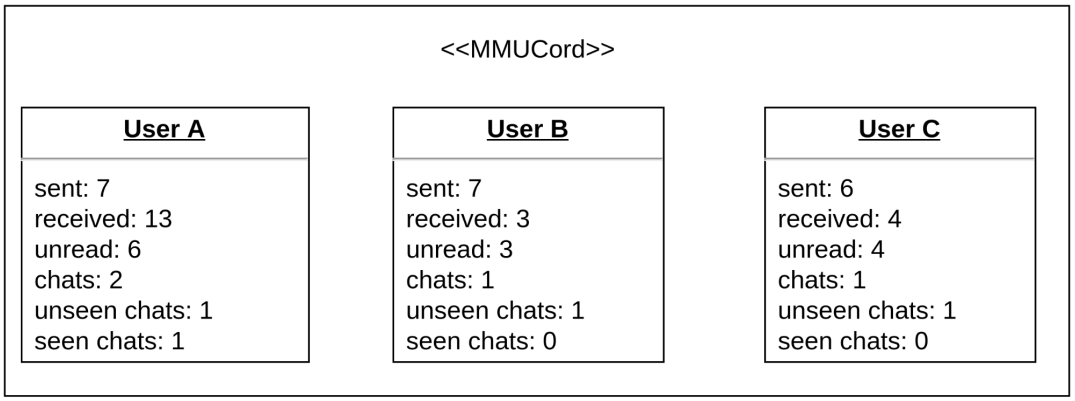

Fig 2 - Analytics Example - Descriptive Analytics

2.3.8.2 Analytics in MMUCord - Diagnostic Analytics

The core of diagnostic embedded analytics is to find out a reason for a particular phenomena and inform the user of the system about the reasons found based on the data collected and processed in the descriptive analytics stage. In MMUCord diagnostic analytics is about telling the student why the MMU Admin employee is busy based on the metrics outlined in the descriptive analytics stage. MMUCord relies on the diagnosis to be done manually by the user based on the provided metrics, this means that the user is left to deduce how much load is on the MMU administration employee based on the number of chats and he/she receives and how much of it he/she is still unseen (pending issues), if an employee has unseen chats meaning that the employee still has issues or enquires from other students to deal with and that is the reason he might not be able to answer the student promptly.

Another way the metrics can be used by the student to deduce the amount of load the MMU administration employee is facing is to use the metrics related to the number of messages sent,received and unread to infer the amount of cases/issues the employee is working on, if the percentage of messages read out of the total messages received is low and the number of sent messages by the administration employee is also low that means that the employee saw the issue of the student and replied with just an acknowledgement of the issue and is currently working on it or added the issue to his/her backlog.

To aid intuitive manual diagnosis MMUCord uses pie charts that visualizes the number of sent,received and unread messages as a percentage of the total number of messages related to the user. Moreover, the pie charts visualize the number of seen and unseen chats as a percentage of the total number of chats.

2.3.8.3 Analytics in MMUCord - Predictive Analytics

At the core of predictive analytics is to predict when a particular phenomena or action might occur. This is usually done by using past data collected previously or data generated in real time. In MMUCord the goal to be achieved from implementing predictive analytics is to predict when the MMU administration employee is expected to be active and when he/she is expected to answer a student’s message.

To predict when the MMU admin employee is expected to be active/online MMUCord follows a series of steps to form an algorithm that can estimate with fair accuracy when a user might be next active/online. First the collected active timestamps are passed to a clustering algorithm (KNN) to group them together, post grouping them together the distance between each point in a group and the average

distance is calculated and recorded for each group. When a student requests the predicted active time for the MMU administration employee, the employee's current time is compared to the recorded groups and then after identifying the group which the employee's time falls in, the average distance is added to the student's time and the result is the predicted active time.

To estimate the average response time for the MMU administration employee, the time difference for each message for each student is calculated and recorded in the system's database. Taking an average response time for each student instead of the average for all students is necessary to accommodate for cases where the relationship of an employee and a student is closer than usual making the employee more responsive. After the average response time is recorded it is added to the current student's time and the result is the predicted response time.

2.3.8.4 Analytics in MMUCord - Prescriptive Analytics

The core of prescriptive analytics is to guide the user by suggesting the best course of action to solve the issue at hand. In MMUCord prescriptive analytics is used to advise the student when to message the MMU administration employee. The advised time is taken from the predictive analytics step which predicts when the employee will be active and this time is the time the system will advise the student to message the MMU Administration employee.

2.3.9 Conclusion

The second chapter of this report, "Related Work" was divided into two sections. The first section was dedicated to instant messaging, the instant messaging section outlined the beginning of instant messaging in 1996 with the introduction of ICQ. In addition, it provided an overview of the instant messaging market, discussing the three main players WhatsApp, Telegram and Facebook Messenger. Providing an overview of each that includes a brief introduction of its founding history in addition to outlining the features present in all three of them and how these features differ across the three applications. The second section of this chapter was dedicated to analytics. The analytics section provides an overview of analytics and how it is used in the instant messaging market. Discussing the four different types of analytics Descriptive, Diagnostic, Predictive and Prescriptive analytics in detail with various examples illustrating each type and how they are linked together to produce the big picture of a business and its customers. In addition to the main four types of

analytics, this section also discussed semantic analytics outlining different processes in semantic analytics and the different models that can be used to extract and infer information from a given text. Moreover, this chapter discussed how the four types of analytics work in tandem together by using the result of one stage as an input for the next in a well confined scenario with set goals in mind. Finally, this chapter described how the four different embedded analytics will be implemented into MMUCord and how each phase of analytics will transition smoothly to the next.

CHAPTER 3

METHODOLOGY

3.1 Overview

This chapter is focussed on exploring, comparing and illustrating the techniques used for requirements acquisition. This chapter will also illustrate the waterfall methodology, the software development methodology used to govern this project.

There are two techniques used in this project to investigate the problem and gather requirements. The two techniques are stakeholder analysis and interviews. This project has two main stakeholders, the students and employees of MMU's administration department.

To govern this project, the waterfall methodology will be used. The waterfall methodology is a sequential development process where progress flows smoothly from one stage to another and the output of the previous stage affects the next. The waterfall method has five different stages each crucial to the software development process.

3.2 Software Development Methodology

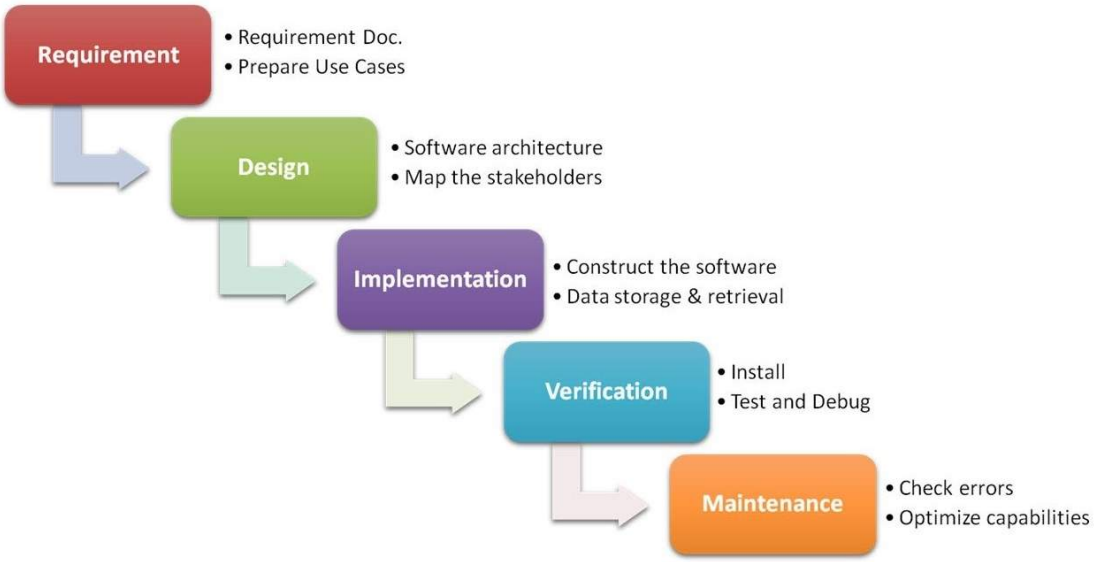

Fig 3 - Waterfall method diagram (Wikipedia Contributors)

The software development methodology used to govern this project is the waterfall method. A traditional linear software development methodology where the requirements are gathered in the beginning and then a sequential plan is created to achieve these requirements. In the waterfall method, the project is delivered in a set of stages where the progress flows from one stage to the next like a waterfall. Since all the stages are dependent on each other it is not possible to advance to the next stage without concluding the stage prior to it. The waterfall method was originally used in manufacturing and construction, this is the reason behind its rigid structure. It was first mentioned in the context of software development in an article written by Winston W. Royce in 1970. However, the term "Waterfall" was not introduced by Royce but instead coined in 1976 by T.E. Bell and T.A Thayer (Ajam, 2017).

The waterfall method has five phases modeling the project lifecycle. When these phases are executed successfully, the project can be marked as successful. The first phase is requirements analysis. This is where the software requirements are gathered. Implementation phase, is the second phase the development of the software will commence. The verification/testing phase is the third phase where the software developed is deployed and tested thoroughly. The Final phase is the maintenance phase, in this phase the application is deployed on MMU Premises, given to students

and MMU's administration department employees to be tested. This section shows the stages of the waterfall method that is applied to guide this project to completion.

3.2.1 Requirements

At this stage, it is important to identify relevant sources of information to aid in the gathering of functional and quality requirements. Based on the information collected, the project specification will be adapted based on the requirements collected from the users. To elicit functional and non-functional(quality) requirements for this project, three requirements elicitation methods are used; these are stakeholder analysis and interviews. Stakeholder analysis is done to identify and classify stakeholders and their interests. After the stakeholder analysis stage is performed, interviews are conducted to understand and sympathize with the stakeholder's perspective to increase the quality of the gathered requirements for the project. Below exists an overview of the methods used for requirements elicitation and the project's approach to applying them.

3.2.1.1 Stakeholders analysis

Stakeholder analysis is all about identifying and understanding the interests, needs and expectations of stakeholders inside and outside the project environment. There are five main steps to be executed for the stakeholder analysis stage to be completed. The five steps are outlined below.

3.2.1.1.1 Identify Project Stakeholders

A stakeholder is a person or a group of people with interest or level of authority or influence over the project. Initially, it is difficult to identify and classify the main stakeholders to focus on in the context of the project. Therefore, at this stage, it is important to consider all possible stakeholders involved in the project. However, some of the stakeholders may be dropped in later stages of the analysis

where it is found that these stakeholders are not of importance or influential to the project.

As an instant messaging service that is focused on serving the MMU community, MMUCord as a project in the beginning has defined five stakeholders. These five stakeholders are students, MMU's management, MMU's administration department employees, the head of the administration department and lecturers.

3.2.1.1.2 Identify stakeholders interest

Continuing on the work of the previous stage, the level of interest each stakeholder holds in the project must be identified. Moreover, individual stakeholders' interests must be identified, the key to identifying a stakeholder's interest must be done by viewing the project from the stakeholder's point of view. Identifying stakeholders interests is a difficult task because interests are usually hidden and sometimes contradict openly stated aims. In addition to contradicting the stated aims, stakeholder interests may change throughout the project. In addition to identifying stakeholders' interests it is important to classify the priority of these interests to the project. It is also useful to outline how the project will be impacted if these are or are not met.

As mentioned above there are five stakeholders that express interest in the MMUCord project. The interests of these stakeholders generally do not conflict with each other on the surface. To discuss stakeholder interests, several points need to be tackled from the stakeholder's perspective to provide the full picture regarding each stakeholder's interests in the project. First, a meaning must be established to the expectations the stakeholder has from this project. Second, it is important to understand how the completion of this project will benefit the stakeholder and finally to identify if any other stakeholder's interests are in conflict with the stakeholder in question.

At the end of the MMUCart project students expect to have an instant messaging platform that is competent and provides a pleasant instant messaging experience that does not feel lacking when compared to other platforms they use daily like WhatsApp. In addition, students expect to have features more suited to the interactions they have in an academic environment with the rest of the MMU Community. MMUCord focuses on providing core messaging features without the bloat provided by other instant messaging services. MMUCord focuses on features like online indicators, last seen, average response time, extensive file sharing capability, group chats, etc… Completing the MMUCart project will improve the quality of the student's interactions with other members of the MMU community, namely MMU's administration department employees, their fellow students and lecturers. The communication will be much faster and more efficient than using emails.

At the end of the MMUCart project the MMU's administration department employees expect to have an instant messaging platform that is competent, user friendly, provides a pleasant user experience and does not feel lacking or alien compared to the platforms they use daily. In addition MMU's administration department employees expect features that aid them in fulfilling their role in the MMU community. Features like topic extraction and classification for different chats, sentiment analysis for the chat body, etc... Completing the MMUCord project successfully will increase the MMU's administration department employees productivity and efficiency since an instant messaging application like MMUCart allows them to communicate instantly with students, navigate through students' inquiries with ease easily, sort students' inquiries with ease, etc…

At the end of the MMUCart project the head of MMU's administration department expects a robust instant messaging system that is robust and capable of sustaining the day to day operations of MMU's administration department, successfully replacing email as the form of communication for MMU's administration team. Completion of the MMUCart project will benefit the head of the administration department because it is now easier to communicate with members of

the department in addition it will boost the administration department employees productivity and efficiency, increasing the student satisfaction as a result.

At the end of the MMUCart project the MMU management expects a competent instant messaging platform that is able to facilitate and adapt to the university's setting and operations aiding different members of the community in accomplishing their daily tasks. In addition, the university's management expects the new messaging platform to adhere to MMU's policies and regulations. Completing the MMUCord project will benefit the MMU management because MMUCord will provide a platform where members of the MMU community are able to communicate and carry on their daily tasks more efficiently in the post pandemic world, increasing students satisfaction in the process while also providing a unique experience for the community when compared to other universities.

At the end of the MMUCart project the lecturers expect an instant messaging application that is competent and does not feel lacking and easy to use compared to the platforms they use daily. Lecturers expect the platform to be more focused on chat features that aid in efficient communications with students without all the bloatware features present on other instant messaging platforms. Completing the MMUCord project, it will benefit the lecturers by providing a platform that provides efficient communications with their students and facilitates their role as instructors.

3.2.1.1.3 Assess stakeholders for importance and influence

In this stage, it is important to classify the main stakeholders in terms of their importance and influence over the project. Influence is measured by the relative power of the stakeholder within the organisation and his ability to control key decisions within the context of the project. A stakeholder with high influence can control key decisions within the project and have strong ability to facilitate implementation of project tasks and cause or sometimes force different parties within the organization to take action. Usually such influence is derived from the individual's hierarchical, economic or political position. On the other hand, a

stakeholder's importance is measured by how important the stakeholder is to project success. An important stakeholder is not necessarily of high influence to the project. However, if the needs and issues this particular stakeholder raises are not addressed, this may be critical to project success.

When classifying stakeholders based on their importance or influence, all stakeholders can fall into four groups, each group defining a certain level of importance and influence within the organization. The first group is called the subjects, these individuals are the ones that show high interest but have low power and influence within the organization. This stakeholder group offers great insights and ideas for the project, however they don't have the power or influence to make decisions. In the context of MMUCord, the subjects group is represented by the students and MMU's administration department employees, they offer great insights but have no power or influence to act on their ideas. However the administration department employees hold more power and influence when compared to the students due to them being near to the head of the department interacting with him/her on a weekly basis. The second group is called the context-setters, the context-settlers group is made up of all the low interest high power stakeholders. The context-settlers group usually has a large influence over the project but have low interest in the project. Therefore, it is best not to involve them in every detail but only to keep them up to date. In the context of MMUCord the context-settlers group is represented by MMU's management. The management may not show interest in MMUCord as a project, however they hold the final decision on what is permitted and what is not. The third group is the players , this group usually have the high power, high interest stakeholders, these are the stakeholders that the project manager wants to prioritize, collaborate with and keep fully engaged. In the context of MMUCord, this group is represented by the head of MMU's administration department. The head of the department has enough interest in MMUCord as a project and enough influence to make decisions in the project. The next group is called the crowd, this group is the one with both the lowest interest and the lowest influence, this stakeholder group requires some form of communication from time to time on the project progress however, they require the least amount of

communication when compared to the other stakeholders. In the context of MMUCart the crowd group is represented by the instructors, the instructors hold the least interest when it comes to the MMUCord project since MMUCord does not directly target them. In addition the instructors in MMU do not have enough influence over the management decisions to contribute in making decisions that benefit the project. However the project may still cater to some of the instructors needs and aid them in communicating with their students. The classification of stakeholders into different groups and the power-influence relationship they hold can be better represented with a power-interest graph. The power interest graph that represents the stakeholders of MMUCord is presented below.

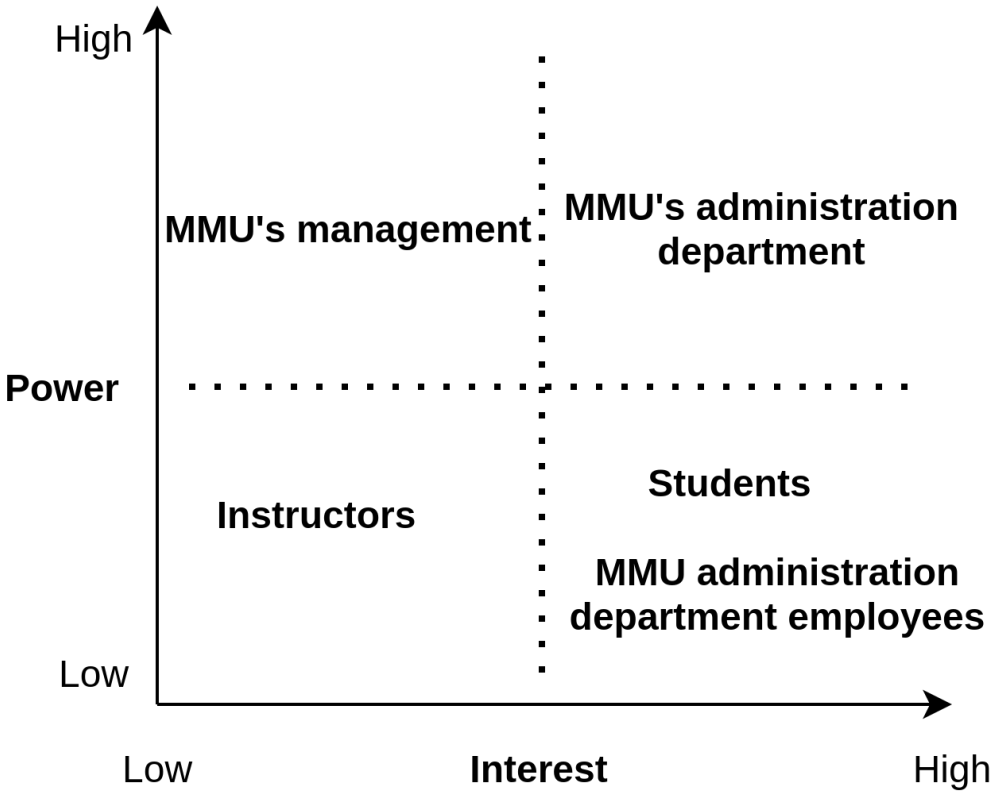

Fig 4 - Power-interest graph

3.2.1.1.4 Define stakeholder participation

In the next steps, an effort is made to assess the level of participation for each stakeholder. It is important to identify which stakeholders participate in each stage of the project. Moreover, not all stakeholders need to be involved in all aspects or every project stage.

The results of stakeholder analysis will provide an overview of who the stakeholders of the project are and their level of interest and influence. In addition to the level of detail and communication they expect.

In the context of MMU it is easy to define the level of communication required by every stakeholder group using the power-interest grid and the classification from the previous section. The first stakeholder group in the top left of the grid, the context setters are to be informed periodically, keeping them up to date with the overall project progress and direction. However, they are not to be involved regularly with minute details. The second stakeholder group in the top right of the grid, the players are to be prioritized, closely managed and kept up to date with the project progress, involving them in any details or changes that may affect the project direction or functionality. The third group in the bottom left of the grid, the crowd, are to be periodically informed about milestones in the project progress but not to be involved in the minute details regarding the project. However, it is not necessary to adhere to their requests. The fourth group of the stakeholders in the bottom right the subjects are to also be periodically informed about the progress milestones and any updates to the project, however similar to the crowd stakeholder group it is not necessary to completely adhere to their requests.

Table 3 - Summary of stakeholder groups and participation

Stakeholder group	Members	Involvement and participation
Context-setters	MMU's Management	Overall project progress
Players	MMU's Administration Department	Keep up to date with the project progress and involve in every detail regarding the project

Crowd	Instructors	Periodically inform about project progress
Subjects	Students and Admin department employees	Periodically inform about milestone project progress

3.2.1.2 Interview

An interview is a dialogue between two people that occurs when they meet to discuss a matter of importance. These two parties are known as the interviewer and the interviewee where questions are asked by the interviewer to obtain information from the interviewee. There are few basic procedures to be followed before an interview to be concluded. First, the interviewer will have to decide on an initial set of interesting questions in general terms and discuss them with the interviewee to make sure that the interviewee has no objections to the questions, and make sure that the questions will cover the topics.

This is usually done verbally. Then, the interviewer just reads off the questions, the interviewee answers without anything written in advance and presumably has not seen the list of questions in advance. An interview may be tightly structured, semi-structured, unstructured, in depth or conversational. In this project, the interview is one on one with an initial set of questions, where the interviewer reads the question and notes the interviewees answers, the interviewer is free to comment on an interviewee's answer, elaborate and ask more questions. This is because interviews are interactive in nature and provide the opportunity to follow up ensuring the stakeholder's complete understanding of the question. However, interviews have drawbacks that may deter from using them, interviews require lots of time to prepare for in addition to commitment from the stakeholders.

For conducting the interviews each stakeholder group a set of starter questions are prepared and are given to each stakeholder to get his/her consent. A meeting spot is agreed on and the interview is then commenced.

The interviewees selected represent all the stakeholder groups. For the group that represents students ten are selected to be interviewed. The ten students selected are all postgraduate students, two are females and the rest are males.

The interviewees selected to represent the Administration department employees five are selected to be interviewed. The five employees selected are all senior employees with five to eight years of experience.

The interviewees selected to be interviewed for the requirements gathering stage are from both groups, students and MMU's administration department employees. There are two students selected, one is a male senior postgraduate student while the other one is a female freshman student in her second trimester. Selecting a freshman student will provide the perspective of students who are new to dealing with the university employees and unfamiliar with the university policy. Selecting a senior student will provide the perspective of students who are well versed in the university, knowledgeable about the policies and already established rapport with the administration department employees. There is one interviewee selected to represent MMU's administration department employees working in the FCI admin office.

3.3 System requirements

This section is concerned with exploring the system requirements for the MMUCord system and the proof of concept prototype to be developed. The requirements section is divided into three subsections, the first subsection is concerned with the requirements for the base messaging system responsible for the registration of a user in addition to the sending and receiving of messages in realtime. The second subsection is concerned with the elicited requirements for the analytics engine of MMUCord where the requirements for all four types of analytics will be discussed. The final subsection is concerned with the requirements of proof of concept prototype.

3.3.1 Base messaging system

This section is concerned with the requirements for the base messaging system. The system responsible for the sending and receiving of messages in realtime. In addition this system is the one responsible for handling the process of user registration. The requirements elicited for the main messaging system is listed below.

1. Users should be able to send and receive text messages in realtime
2. The system should support typing indicators
3. The user should be able to see all the current active users on the platform
4. The user should be able to register to the platform
5. The system should automatically log in the user on application launch
6. The user should be able to choose their username and profile picture
7. The system should support online indicators
8. The system should support read receipts
9. The system should encrypt messages using AES with 128 bits before sending
10. The messages should be stored on the user's device for offline access
11. The system should support both dark and light theme

3.3.1 Embedded Analytics engine

The embedded analytics engine is responsible for the collection, processing and displaying all forms of analytics to the end user be it the student or the MMU Administration employee. This section includes all the elicited requirements related to the four types of embedded analytics.

1. Descriptive Analytics
    a. The system should calculate and display the number of messages sent by the user.
    b. The system should calculate and display the number of received messages by the user.
    c. The system should calculate and display the number of unread messages by the user.
    d. The system should calculate the total number of chats for a particular user.

e. The system should calculate and display the number of seen chats for a particular user.
f. The system should calculate and display the number of unseen chats for a particular user.

2. Diagnostic analytics

a. The system should calculate the percentage of unread messages out of the number of messages received
b. The system should calculate the percentage of sent messages out of the total number of messages for the user (total messages = number of sent + number of received)
c. The system should calculate the percentage of received messages out of the number of total messages for the user
d. The system should calculate the percentage of seen chats out of the total number of chats
e. The system should calculate the percentage of unseen chats out of the total number of chats
f. The system shall present the calculated percentage values related to messages in requirements a-c in pie charts to aid manual diagnostics by the user
g. The system shall present the calculated percentage values related to chats in requirements c and d in pie charts to aid manual diagnostics by the user
h. The system shall tell the user if the MMU Administration employee is busy or not

3. Predictive analytics

a. The system shall predict when the user is next online
b. The system should predict when the user is to respond to a message/query

4. Descriptive analytics

a. The system should tell the user when is the best time to message another user

3.3.2 Proof of concept prototype

This subsection of system requirements is concerned with the set of requirements to be implemented in the proof of concept prototype. As stated previously in the report not all the elicited requirements are implemented due to the circumstances caused by COVID-19 and the travel restrictions in addition to the time constraints there were issues deploying MMUCord to be used by the MMU administration department to gather the data needed to perform predictive and prescriptive analytics. Therefore the proof of concept prototype focuses on the requirements related to descriptive and diagnostic analytics only due to lack of the data needed to implement the algorithms needed in the later analytics stages.

The proof of concept prototype implements the requirements stated under the base messaging system allowing the users of the prototype to send and receive messages in realtime. Moreover the proof of concept prototype implements all the requirements under both Descriptive and Diagnostic analytics.

3.5 Conclusion

The third chapter was concerned with the methodology used to bring the MMUCord platform into life. The Waterfall methodology was used to guide the MMUCord project. The third chapter provides an overview of the Waterfall methodology as a whole and how its different stages were implemented in the context of MMUCord. In the third chapter the first stage of the waterfall method which is requirement analysis was discussed. The chapter discussed the techniques used for requirements gathering and prioritization in the MMUCord project. Finally, the third chapter discussed the requirements of the elicited requirements of MMUCord as a system and the requirements implemented in the proof of concept prototype.

CHAPTER 4

SYSTEM DESIGN

4.1 Introduction

This chapter is concerned with MMUCord's system design. This chapter will discuss the different components that make up MMUCord, exploring each component and its role in the system. At the end of the chapter an overview of how all the different components are related and integrate together to bring MMUCord to life will be discussed.

At this stage MMUCord is in an early stage of development. There only exists a prototype that provides a base text messaging experience in addition to the features related to descriptive and diagnostic analytics where the users are able to send and receive encrypted messages in real time in addition to viewing the different metrics used for descriptive and diagnostic analytics. The prototype has the following features already implemented.

1. **Register**

Users are able to register to the platform, the registering process involves the users choosing a username and choosing a profile picture then the user is automatically logged in and is directed to the home screen.

2. **View active users**

Users are able to view all the active users on the platform.

3. **Typing indicators**

The application supports typing indicators, the users are able to see a typing indicator when the other user is typing.

4. **Send and receive messages**

The users are able to send and receive messages in realtime.

5. **Encrypt messages**

The platform encrypts the messages before sending them to the recipients and decrypts them on the receiver's end.

6. **Message Receipts**

The application supports read receipts to indicate when the message was sent, received and read.

7. **User is saved into the phone local storage**

If a user is already logged into the app before, his/her details are stored in the local storage to allow automatic login.

8. **View Descriptive analytics metrics**

The system allows the user to view the metrics related to descriptive analytics

9. **View the Diagnostic analytics metrics**

The system allows the user to view all the metrics related to descriptive and diagnostic analytics in addition to the charts presenting them

4.2 Use Case diagram

UML use case diagrams describe the interactions between the system and different external components. These external components can be other independent systems or users of the system. The use case diagram consists of three components, these are the system itself, the actors and use cases. The first component is the system itself which represents the system as a whole. The second component of the use case

diagram is the actors. The actors are the users of the systems are referred to as ‘actors’, there are two types of actors reactionary and actionary. Actionary actors are actors that initiate the use of the system and reactionary actors respond to actions by actionary actors. The third component of the use case diagram are the use cases, the use cases are the different processes or actions an actor can perform within the system.

A use case diagram focuses on presenting the functional aspects of a system. It achieves that by capturing the processes actors in the system can perform. It views the system from a user’s point of view. A use case diagram is useful in many ways. It can be used as an overall model to the system to show to different stakeholders in corporate meetings. Another use of use case diagrams can be to identify major components and their interactions to write test scripts to ensure they are working correctly.

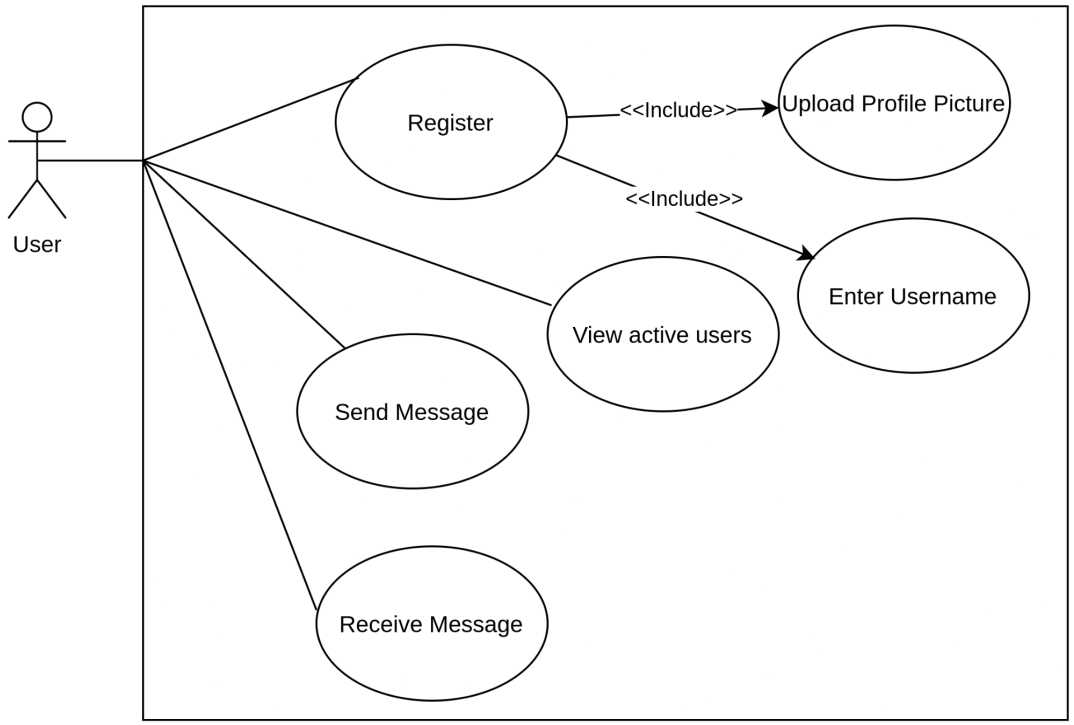

Fig 05 - Use case diagram for the base messaging system

The above diagram explores the different use cases a user is allowed to perform in the current version of MMUCord's prototype. This section provides an explanation of these use cases.

- **Register:** This use case involves the user registering to the MMUCord platform, registering to the platform involves the user entering their username and uploading a profile image.
- **View active users:** This use case involves the user viewing all the active users on the platform.
- **Send Message:** This use case involves the user sending a message to another user on the platform.
- **Receive Message:** This use case involves the user receiving a message from another user on the platform.

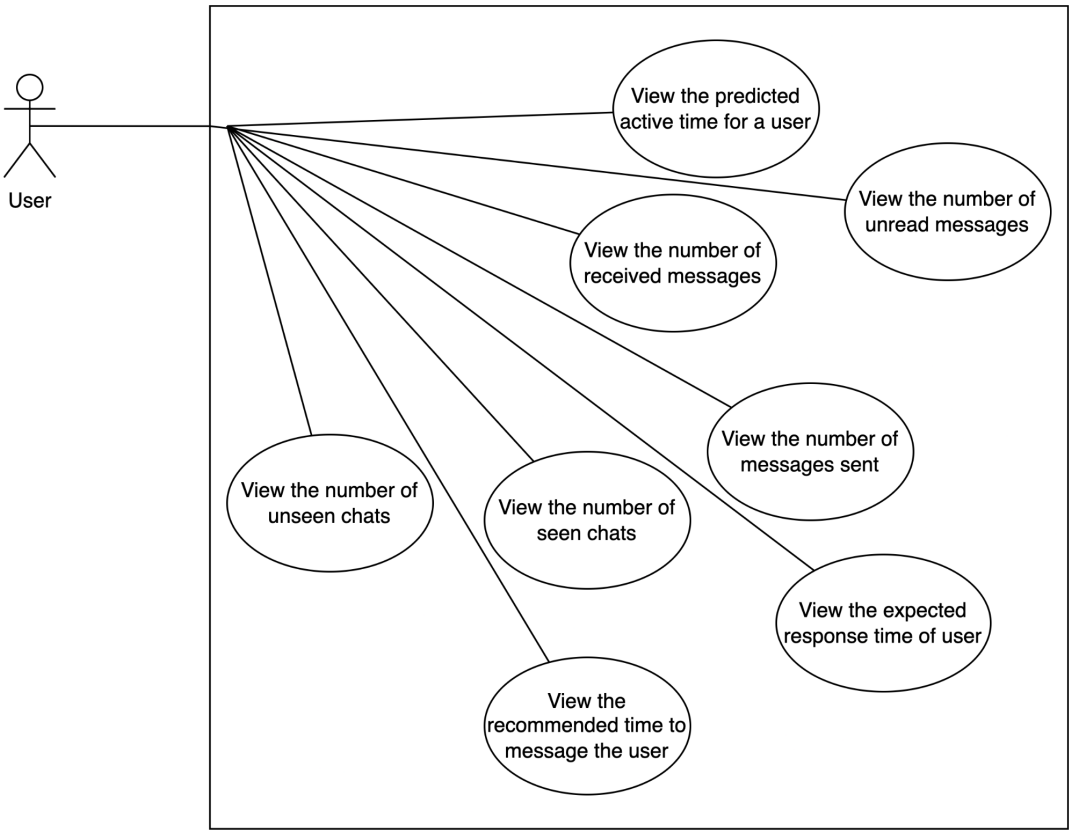

Fig 06 - Use case diagram for the Analytics Engine

4.3 Sequence diagram

The sequence diagram is a part of the UML interaction diagram that shows how different parts of a system interact together, detailing all the operations and their order of execution. A sequence diagram is mainly used to further explain/elaborate on the use cases in the use case diagram, detailing the usage scenario in addition to the logic of any method or service.

4.3.1 Sequence Diagram - User registration

The sequence diagram below is to show the user registration process, first the user using the mobile application provides a username and a profile picture from the pictures stored on the user's device. Post providing a picture and a username the mobile application connects to the **Server** and sends the username and the picture, after the **Server** receives the username and the picture it constructs a user object and then attempts to connect to the database and creates a new record in the users table.

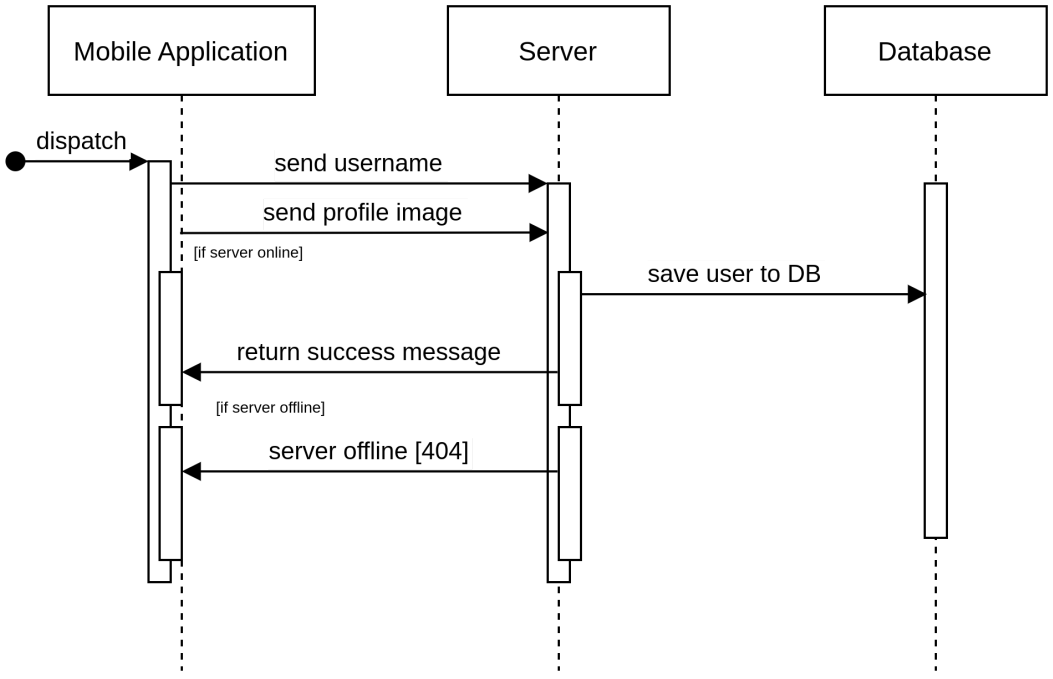

FIg 07 - User registration sequence diagram

4.3.2 Sequence Diagram - Sending and Receiving messages

The sequence diagram below is to show the process of sending and receiving messages in realtime. First the User Service will send the message contents, receiver id and the timestamp to the Message Service. The Message Service then takes all the received parameters and creates a Message object and saves the message to the Database. Finally the Database is queried for the latest messages when the receiver opens the application populating his Messages screen in the process.

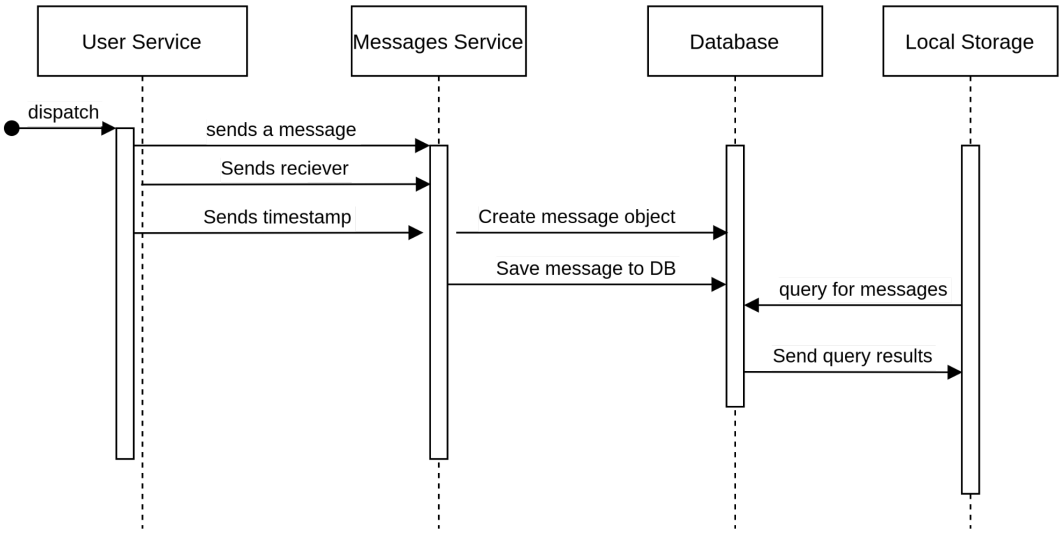

Fig 08 - Sequence Diagram - Sending and Receiving messages

4.4 Components of MMUCord

This section will provide an overview of all the components that make up the MMUCord system and the roles they occupy in achieving the functionality of MMUCord. MMUCord consists of four major components, the first being the Flutter mobile application which acts as the client, the second component being the database which acts as the central storage of most data that make the platform function. The third component is the image server, the image server is responsible for the storage and retrieval of users profile images. The fourth component is the main server which hosts the image server and the database.

4.4.1 Ubuntu Server

MMUCord relies on a server to store all the data that allows the application to function. The Ubuntu server is used to host and store the database in addition, it hosts the image server that manages the storage and retrieval of a user's profile picture.

The server is an upgraded Dell optiplex 7020 business desktop computer running Ubuntu Server 20.04.3 LTS. The server is powered by an Intel Core I5 4590, a four core Cpu running at 3.70 GHz. Moreover, the server is equipped with 20 gigabytes of DDR3 Ram running at 1600 MHz and a Western Digital Blue 500 GB Mechanical Hard Drive.

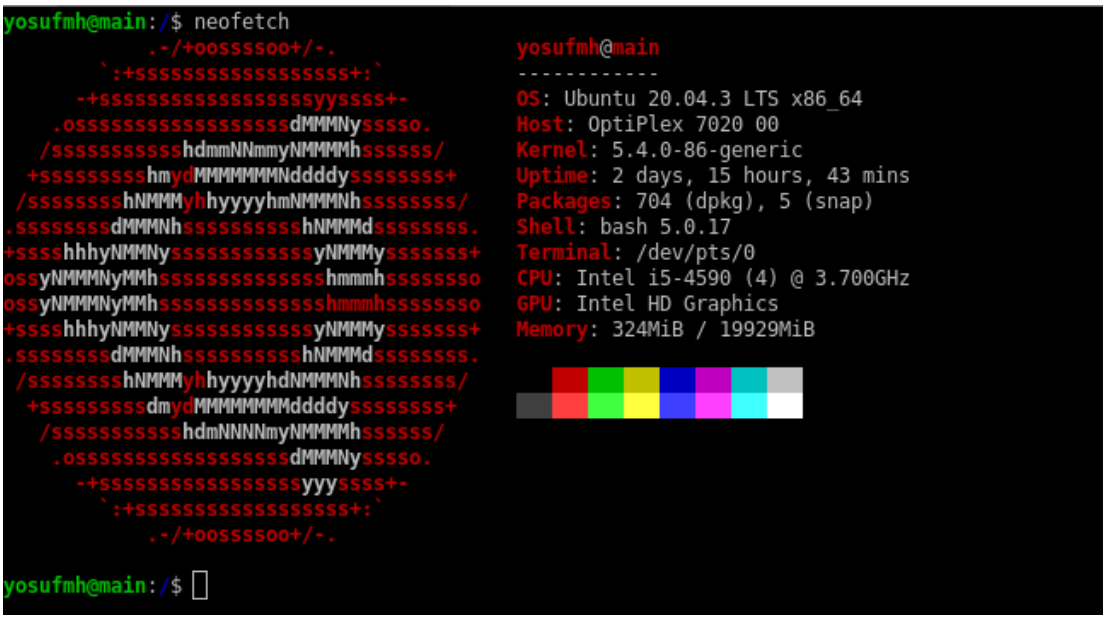

Fig 09 - Server neofetch

4.4.2 Database

The MMUCord application uses a database to store all the data that allows the MMUCord application to function. The database used to prototype the first version of MMUCord is the RethinkDB database, a nosql document based realtime database.

The database is hosted on the ubuntu server and is containerized in an isolated docker container to ensure a static environment, easy deployment and maximum

security. The database has four tables each storing information that relates to one component/service of the mobile application.

4.3.2.1 Database structure

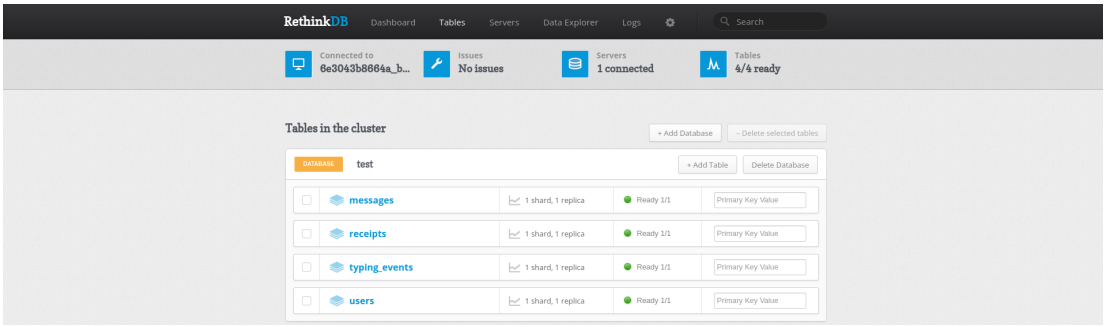

Fig 10 - Database Structure

This section outlines the structure of the database, providing an overview of all the tables present, and the data each store. Information about the tables is found below.

1. Users table

The users table keeps track of all the users registered in the platform, it stores all the user's relevant information. The table below gives an overview of the users table outlining all the fields and their data type respectively.

Table 4 - Database structure - users Table

Field Name	Data Type	Description
username	String	This field stores the user's username
photo_Url	String	This field stores the url of the user's profile image
active	bool	This field stores the status of the user, to keep track of weather he/she is online or offline
last_seen	timestamp	This field stores the date and time that represents the

		time where the user was last active
id	String	The field stores the unique id that identifies each unique record in the table

2. Messages table

The Messages table keeps track of all the messages sent on the platform. The messages table stores the message contents in addition to the timestamp and the sender and receiver information. The table below gives an overview of the messages table outlining all the fields and their data type respectively.

Table 5 - Database structure - messages Table

Field Name	Data Type	Description
from	String	This field stores the sender's user id
to	String	This field stores the receiver's user id
contents	String	This field stores the message contents
timestamp	timestamp	This field stores the date and time that represents the time when the message was sent
id	String	The field stores the unique id that identifies each unique record in the table

3. receipts table

The receipts table keeps track of all the message receipts, the receipts table stores all the information required for functioning message receipts. The table below gives an overview of the receipts table outlining all the fields and their data type respectively.

Table 6 - Database structure - receipts Table

Field Name	Data Type	Description
recipient	String	This field stores the recipient user's id
message_id	String	This field stores the message id
status	ReceiptStatus (enum)	This field stores the status of the message (receipt)
timestamp	timestamp	This field stores the date and time that represents the time when the message was sent, read or delivered
id	String	The field stores the unique id that identifies each unique record in the table

4. typing_events table

The typing_events table stores all the information needed to provide a functioning typing indicator service. The table below gives an overview of the receipts table outlining all the fields and their data type respectively.

Table 7 - Database structure - typing_events Table

Field Name	Data Type	Description
from	String	This field stores the user id of the person who is typing
to	String	This field stores the receiver's user id
event	Typing (enum)	This field stores the status of the typing event
id	String	The field stores the unique id that identifies each unique record in the table

4.4.3 Image server

The image server is responsible for managing the storage and retrieval of users profile images. The image server is a uvicorn server that hosts an API built using FastAPI, a rich web framework used to build performant APIs using python.

The API has three routes, the first route is the Root route, the Root route is a GET route that returns an 200 Ok response once visited. This route is mainly used for testing the connection between the client and the API. The second route is the Upload route, the Upload route is a POST route that takes an image file and uploads it to the server. The Upload route returns a JSON response that contains the filename, filetype, path on the server and a string that contains the encoded image in Base64. The third route is the getimage route, the getimageroute is a GET route that returns the image file as the response. Below exists a table that summarizes all the API routes providing a description for each route alongside the parameters and an example of a successful response .

Table 8 - API route summary

Route	Description	parameters	Response example

/	This is the root route, it is used to test the connection between the client and the API	null	{
			"response": = "200 Ok"
			}

/upload	This is a POST route that is used to upload a picture to the server	file:imagefile	{
			"file_name": "Theresa.jpg",
			"file_type": "image/jpeg",
			"path": "http://192.168.0.139:8000/images/profile/Theresa.jpg",
			"Encoded_image_bytes": "...."
			}

/images/profile/{imagename}	This is a GET route that returns the image file requested	Imagename:String	file:imagefile

4.4.4 Flutter Client Application

MMUCord is a mobile focused instant messaging application. The mobile application acts as the frontend client the users interact with. The mobile application was developed using Flutter, a robust framework developed by Google used to create natively compiled cross platform mobile applications using Google's Dart programming language. The Flutter application was designed using the BLOC pattern, a state management solution that Google recommends instead of the native state management solution.

The Flutter mobile application is divided into several components/services that work together seamlessly to bring a functioning user experience. The Flutter application contains seven different services, each responsible for managing one aspect of the application.

4.4.4.1 Flutter Client Application Client Components

This section is focused on providing an overview of the different services that make up the MMUCord application. As mentioned before there are seven main services that work together to bring MMUCord to life. These services are listed below.

1. **User service:** This service is responsible for managing users on the platform.
2. **Message service:** This service is responsible for managing the sending and receiving of messages on the platform.
3. **Receipt service:** This service is responsible for keeping track and managing the message receipts.
4. **Typing Notifier service:** This service is responsible for managing and keeping track of the typing notifiers.
5. **Encryption service:** This service is responsible for encrypting messages between users in transit to maintain security and privacy.

6. **Datasource service:** This service is responsible for curating and storing the messages that were already received to a database that is locally stored on the user's device.
7. **Image Uploader service:** This service is responsible for uploading the user's profile image to the Image server.
8. **Analytics Service:** This service is responsible for implementing the analytics functions by using the Image server and the Datasource service.

Each of these services are classes implemented in Dart, the class diagram below shows each class with all its accompanying methods and variables. These methods will be further elaborated on in the Implementation chapter.

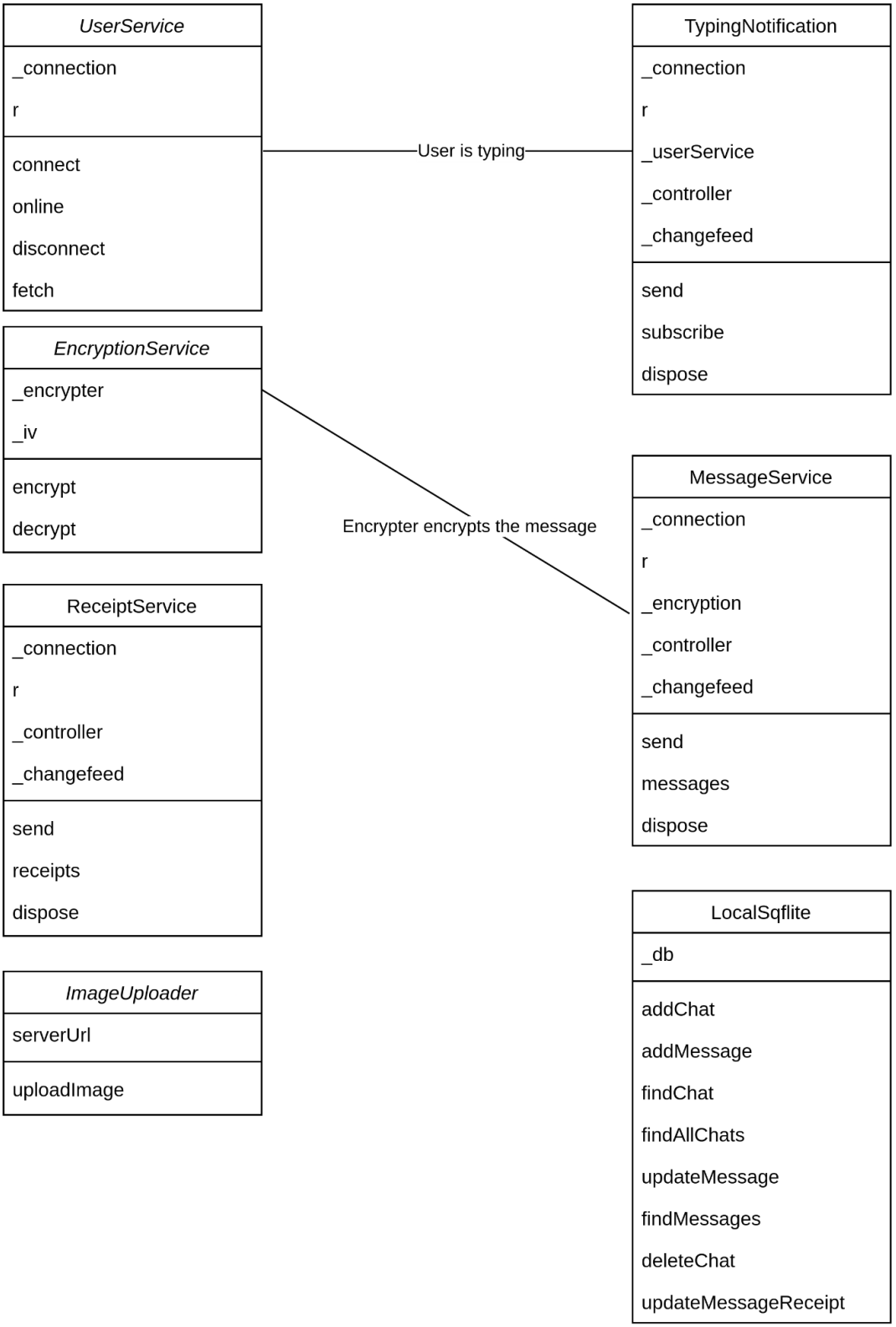

Fig 11 - Services class diagrams

4.5 User interface

This section is focused on demonstrating the Flutter application's user interface. For each screen, a brief description and a few screenshots will be provided.

4.5.1 Boarding Page

The boarding page is the first page a new user is greeted with when opening the application for the first time. This page allows the user to choose a username and a profile picture and register to the platform. After registering the user is directed to the homepage automatically.

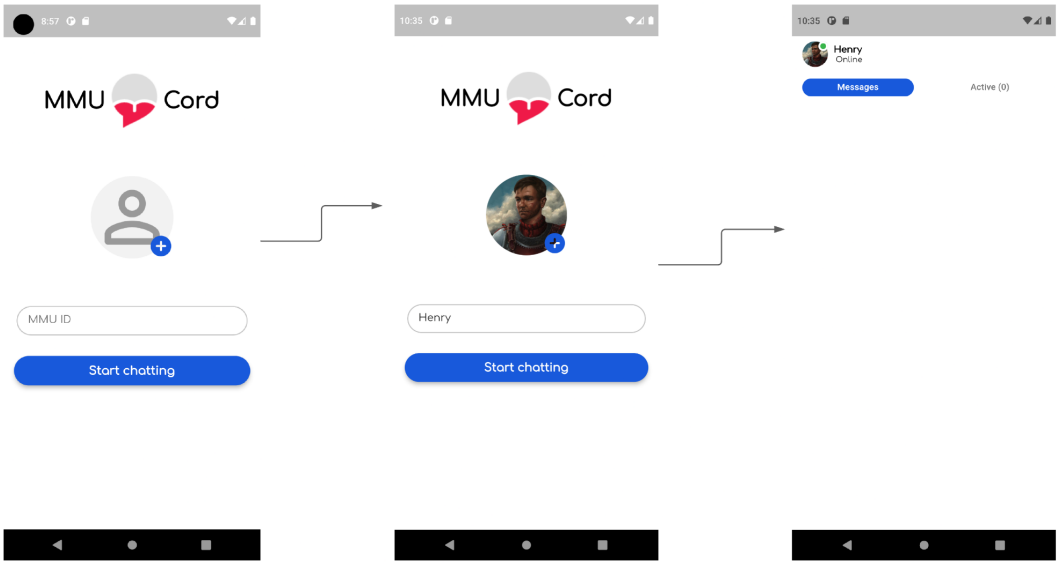

Fig 12 - Boarding Screen flow

4.5.2 Home Page

This is the homepage of the application, for an already registered and logged in user this is the first page he/she is to encounter. This page has two tabs. the first being the messages tab, this tab shows all the chats the active chats for the user. The second tab is the Active tab, this tab lists all the active users on the platform.

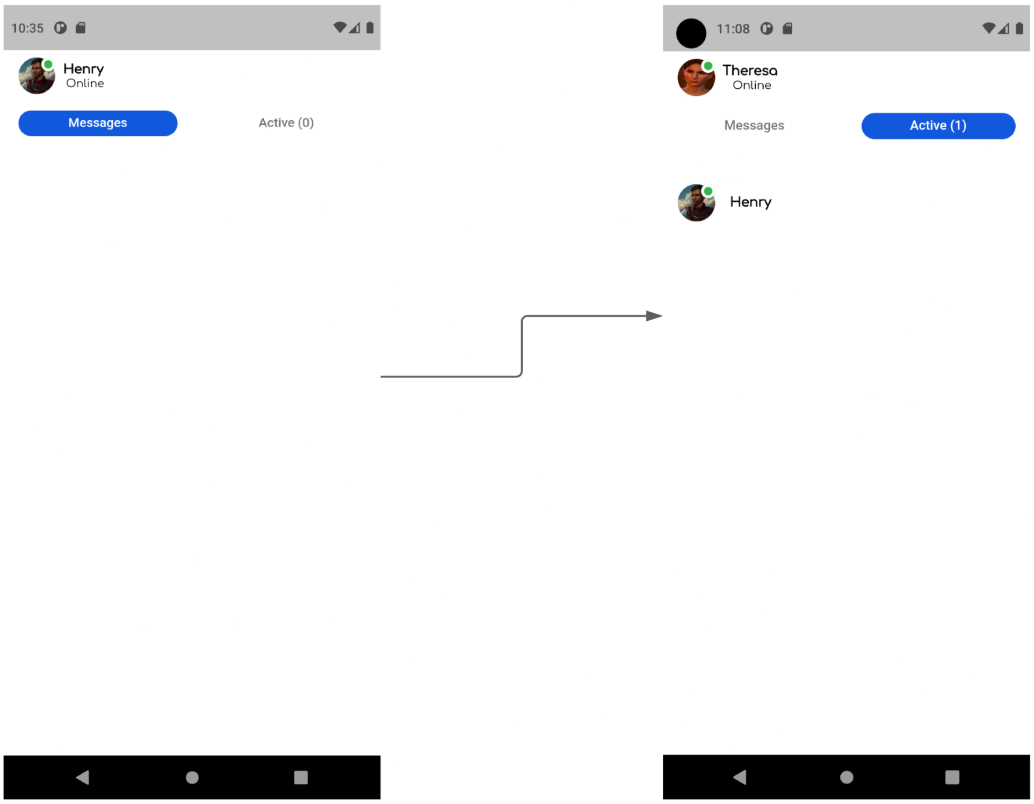

Fig - 13 Home screen

4.5.3 Message screen

The message screen is the screen the users use to write and send messages to each other. The message screen includes a header, the header includes the receiver's profile picture, username and online status/typing indicator. Besides the header the message screen includes all the messages that went back and forth between both parties with read receipts.

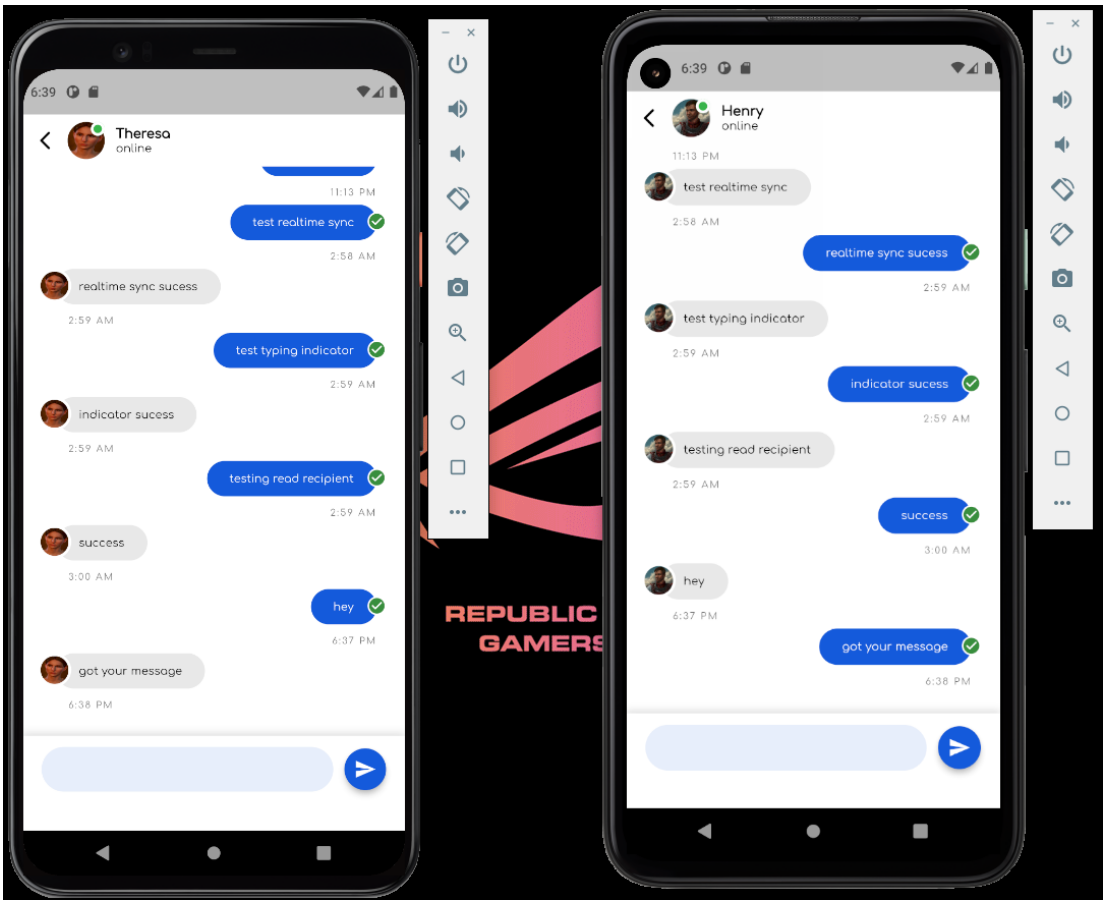

Fig 14 - Message Screen

4.5.4 Analytics screen

The analytics screen allows the user to view all the metrics related to descriptive and diagnostic analytics in addition to the charts presenting them.

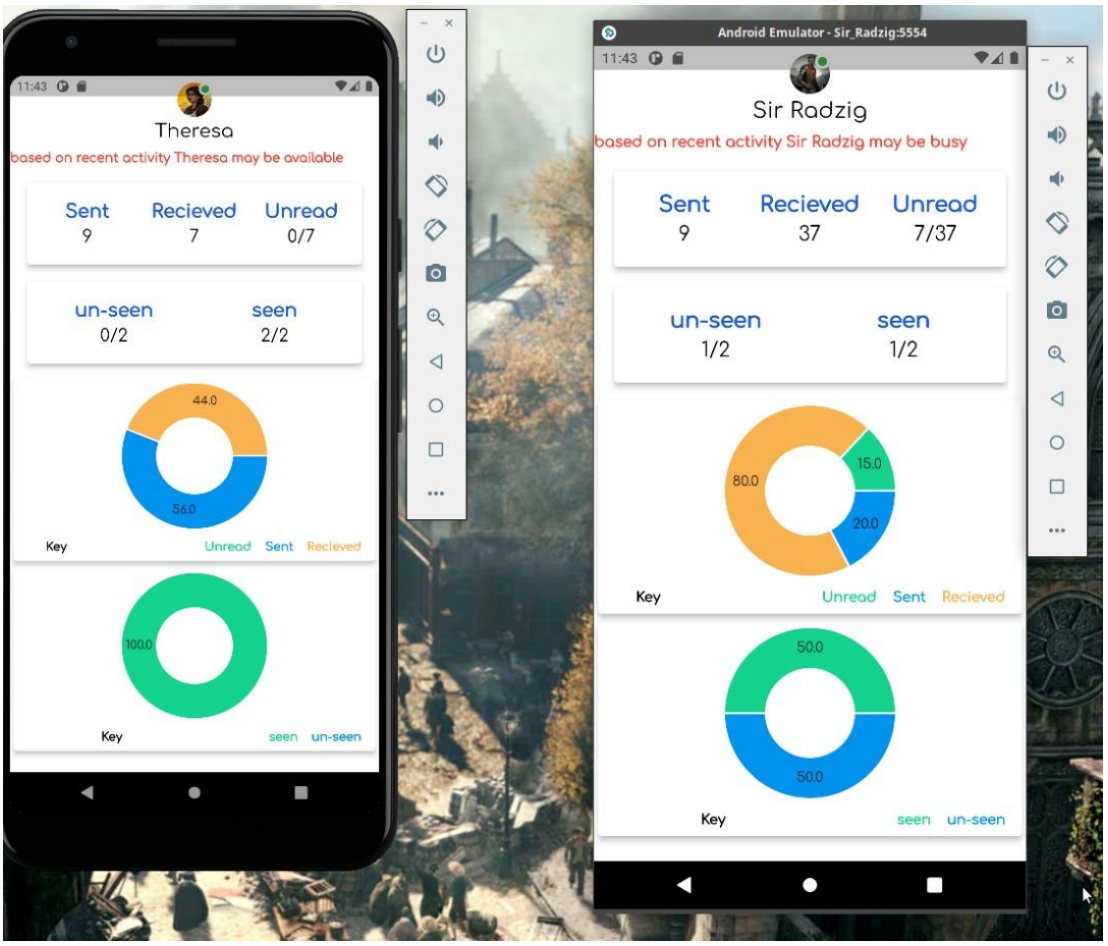

Fig 15 - Analytics Screen

4.6 Conclusion

The fourth chapter focussed on MMUCord's system design. This chapter illustrated how every component is designed. First a use case diagram is provided illustrating all the use cases a user is permitted to do. Moreover, the fourth chapter discussed how the database is deployed and configured in addition to how the tables are structured. Next, the chapter discusses how the image server is configured and deployed, in addition the same section discusses how the API is designed outlining all the available routes. Finally, the fourth chapter demonstrates the Flutter client application including the core services and the user interface.

CHAPTER 5

IMPLEMENTATION

5.1 Introduction

This chapter is concerned with MMUCord's implementation. As mentioned before MMUCord is developed using Flutter, a UI framework developed by Google to develop cross-platform mobile applications. Flutter uses OpenGL to render everything on the screen. This gives the developers complete control over what is rendered on the screen. This results in astonishing user interface designs that are hard if not impossible to replicate using Native SDKs.

MMUCord is mainly developed for Android because I do not have access to a computer running MacOS or an Iphone. Since the application is mainly developed for android it will mostly follow Google's material design principles. Flutter as a framework does not allow updating the UI automatically when components change their state, instead flutter requires the developer to manually manage the state of all the components and update the user interface to different events accordingly.

Flutter comes with its own native state managing solution however, Google recommends to use the BLOC pattern, a state management solution that is more suited for enterprise grade applications. The BLOC pattern is more suited for enterprise applications because it allows the user interface, data layer and business logic into different components. This makes the application easier to test and easier to have multiple developers working on different components concurrently.

MMUCord requires an image server in addition to an API to store and retrieve user profile images. The image server hosts and runs the API. The API was developed using the FastAPI python web framework. Later in the chapter the implementation of the different API routes will be presented.

5.2 Services

As mentioned before, MMUCord has eight different components/services that manage different aspects of the application. In this section, the eight services will be dissected and their implementation will be discussed.

5.2.1 User Service

The user service is responsible for managing the users of the platform. It handles operations like logging in and logging out, fetching all the users and filtering out the ones that are active. The user service has four methods, the four methods each with their respective implementation are found below.

5.2.1.1 Connect method

The connect method is a method that takes the details entered by the user in the boarding page and inserts a new record in the database. In other words, this method is used to register new users.

```
Future<User> connect(User user) async {
  var data = user.toJson();
  if (user.id != null) data['id'] = user.id;

  final result = await r.table('users').insert(data, {
    'conflict': 'update',
    'return_changes': true,
  }).run(_connection);

  return User.fromJson(result['changes'].first['new_val']);
}
```

5.2.1.2 Disconnect method

The disconnect method is a method used to log users out of the platform. It achieves this by updating the last_seen and active fields, setting the active field to false and the last_seen field to the current time.

```
Future<void> disconnect(User user) async {
  await r.table('users').update({
    'id': user.id,
    'active': false,
    'last_seen': DateTime.now()
  }).run(_connection);
  _connection.close();
}
```

5.2.1.3 Online method

The online method is a method that is used to fetch all the active/online users on the platform. It achieves that by filtering the users table by the active field, returning any user that has the active field set to true.

```
Future<List<User>> online() async {
  Cursor users =
      await r.table('users').filter({'active': true}).run(_connection);
  final userList = await users.toList();
  return userList.map((item) => User.fromJson(item)).toList();
}
```

5.2.1.4 Fetch method

The fetch method is used to fetch the data of a particular user. The method achieves this by querying the users table for the particular user's Id and then returns a Json object that has all the user's information.

```
Future<User> fetch(String id) async {
  final user = await r.table('users').get(id).run(_connection);
  return User.fromJson(user);
}
```

5.2.2 Message Service

The message service is responsible for managing the sending and receiving of messages on the platform. The message service has three methods, the three methods each with their respective implementation are found below.

5.2.2.1 Send method

The send method converts the message to a Json object to suit being stored in a nosql database. Then it takes the converted message object and inserts it into the messages table in the database.

```
Future<Message> send(Message message) async {
  var data = message.toJson();
  if (_encryption != null)
    data['contents'] = _encryption.encrypt(message.contents);
  Map record = await r
      .table('messages')
      .insert(data, {'return_changes': true}).run(_connection);
  return Message.fromJson(record['changes'].first['new_val']);
}
```

5.2.2.2 Messages method

The messages method takes a user as a parameter and listens to a stream of messages that returns all the messages sent to the particular user in real time.

```
Stream<Message> messages({User activeUser}) {
  _startReceivingMessages(activeUser);
  return _controller.stream;
}
```

The messages method has a method _startReceivingMessages which contains the code that listens to the real time stream.

```
_startReceivingMessages(User user) {
  _changefeed = r
      .table('messages')
      .filter({'to': user.id})
      .changes({'include_initial': true})
      .run(_connection)
      .asStream()
      .cast<Feed>()
      .listen((event) {
        event
            .forEach((feedData) {
              if (feedData['new_val'] == null) return;

              final message = _messageFromFeed(feedData);
              _controller.sink.add(message);
              _removeDeliverredMessage(message);
            })
            .catchError((err) => print(err))
            .onError((error, stackTrace) => print(error));
      });
}
```

The _startReceivingMessages method contains two methods the first being the _messageFromFeed method and the second being _removeDeliverredMessage method.

The _messageFromFeed method decrypts the message and converts it into an object that abides by the Message model so it can be used in the application.

```
Message _messageFromFeed(feedData) {
  var data = feedData['new_val'];
  if (_encryption != null)
    data['contents'] = _encryption.decrypt(data['contents']);
  return Message.fromJson(data);
}
```

The _removeDeliverredMessage method removes the message from the database table once it is received and stored on the user's device. This is done for two reasons, the first reason is that this ensures the user's privacy and this is important in a business environment specially in MMUCord's use case since students send sensitive personal information such as passport numbers. The second reason this is done is that it saves storage space on the server reducing the cost of operating the application in the long run.

```
_removeDeliverredMessage(Message message) {
  r
      .table('messages')
      .get(message.id)
      .delete({'return_changes': false}).run(_connection);
}
```

5.2.2.3 Dispose method

The dispose method takes care of all dangling dependencies of the messages service. The method cancels the stream and closes the controller.

```
dispose() {
  _changefeed?.cancel();
  _controller?.close();
}
```

5.2.3 Encryption Service

This service handles the encryption of messages on transit, it also handles the decryption of messages before storing it on the device's local database. The encryption service has two methods, one that encrypts the message and the other decrypts the message.

5.2.3.1 Encrypt method

The encrypt method takes the body of the message as a parameter and then encrypts it.

```
String encrypt(String text) {
  return _encrypter.encrypt(text, iv: _iv).base64;
}
```

5.2.3.2 Decrypt method

The decrypt method takes the body of the encrypted message as a parameter and then decrypts it.

```
String decrypt(String encryptedText) {
  final encrypted = Encrypted.fromBase64(encryptedText);
  return _encrypter.decrypt(encrypted, iv: this._iv);
}
```

5.2.4 Typing Notification Service

The typing notification service is responsible for managing and tracking the typing notification. The typing notification service has three methods, the three methods each with their respective implementation are found below.

5.2.4.1 Send method

The send method accepts an event as a parameter, the send method uses the fetch method from the message service to fetch the receiver's info from the user's table. The event is then converted into a Json object, the record is then a record is inserted into the receipts table.

```
Future<bool> send({@required TypingEvent event}) async {
  final receiver = await _userService.fetch(event.to);
  if (!receiver.active) return false;
  Map record = await _r
      .table('typing_events')
      .insert(event.toJson(), {'conflict': 'update'}).run(_connection);
  return record['inserted'] == 1;
}
```

5.2.4.2 Subscribe method

The subscribe method takes a user and a list of users as parameters and listens to a stream of messages that returns all the typing events sent to the particular user in real time.

```
Stream<TypingEvent> subscribe(User user, List<String> userIds) {
  _startReceivingTypingEvents(user, userIds);
  return _controller.stream;
}
```

The subscribe method has a method _startReceivingTypingEvents which contains the code that listens to the real time stream.

```
_startReceivingTypingEvents(User user, List<String> userIds) {
  _changefeed = _r
      .table('typing_events')
      .filter((event) {
        return event('to')
            .eq(user.id)
            .and(_r.expr(userIds).contains(event('from')));
      })
      .changes({'include_initial': true})
      .run(_connection)
      .asStream()
      .cast<Feed>()
      .listen((event) {
        event
            .forEach((feedData) {
              if (feedData['new_val'] == null) return;

              final typing = _eventFromFeed(feedData);
              _controller.sink.add(typing);
              _removeEvent(typing);
            })
            .catchError((err) => print(err))
            .onError((error, stackTrace) => print(error));
      });
}
```

The _startReceivingTypingEvents method contains two methods the first being the _eventFromFeed method and the second being _removeEvent method.

The _eventFromFeed method converts the object returned from the database to a TypingEvent object to suit being used by other components in the application.

```
TypingEvent _eventFromFeed(feedData) {
  return TypingEvent.fromJson(feedData['new_val']);
}
```

The _removeEvent method removes the typing events from the database table once it is received and stored on the user's device. This is done because it saves storage space on the server reducing the cost of operating the application in the long run.

```
_removeEvent(TypingEvent event) {
  _r
      .table('typing_events')
      .get(event.id)
      .delete({'return_changes': false}).run(_connection);
}
```

5.2.4.3 Dispose method

The dispose method takes care of all dangling dependencies of the typing notification service. The method cancels the stream and closes the controller.

```
dispose() {
  _changefeed?.cancel();
  _controller?.close();
}
```

5.2.5 Image Upload Service

The image service is used to upload users profile images to the image server. The dio flutter package is used to send a post request to the upload image route and get the image path from the response to save it as the photo url in the user object.

```
class ImageUploader {
  final String serverUrl;

  ImageUploader(this.serverUrl);

  Future<String> uploadImage(File profileImage) async {
    var dio = Dio();

    dio.options.baseUrl = serverUrl;

    String imageFileName =
        profileImage.path.split('/').last; // extract image filename

    var formData = FormData.fromMap({
      'file': await MultipartFile.fromFile(profileImage.path,
          filename: imageFileName)
    });
    final response = await dio.post('/upload', options: Options(followRedirects: false),data: formData);
    if (response.statusCode != 200) {
      return null;
    }
    return response.data['path'];
  }
}
```

5.2.6 Receipt service

The receipt service is responsible for handling message receipts in the application. A message receipt has three states, delivered, sent and read. The receipt service is responsible for managing and relaying the state of the message receipt to the user. The receipt class has three methods each with their respective implementations outlined below.

5.2.6.1 Dispose method

The dispose method takes care of all dangling dependencies of the receipt service. The method cancels the stream and closes the controller.

```
dispose() {
  _changefeed?.cancel();
  _controller?.close();
}
```

5.2.6.2 Send method

The send method is responsible for converting a receipt object to a Json like format to be later inserted into the receipts table in the database.

```
Future<bool> send(Receipt receipt) async {
  var data = receipt.toJson();
  Map record = await r.table('receipts').insert(data).run(_connection);
  return record['inserted'] == 1;
}
```

5.2.6.3 Receipts method

The receipts method is responsible for listening on the stream that sends the receipts information in real time.

```
Stream<Receipt> receipts(User user) {
  _startReceivingReceipts(user);
  return _controller.stream;
}
```

The receipts method contains the method _startReceiviingReceipts which handles listening to the real time stream in addition to processing the information coming from it.

```
_startReceivingReceipts(User user) {
  _changefeed = r
      .table('receipts')
      .filter({'recipient': user.id})
      .changes({'include_initial': true})
      .run(_connection)
      .asStream()
      .cast<Feed>()
      .listen((event) {
        event
            .forEach((feedData) {
              if (feedData['new_val'] == null) return;

              final receipt = _receiptFromFeed(feedData);
              _removeDeliverredReceipt(receipt);
              _controller.sink.add(receipt);
            })
            .catchError((err) => print(err))
            .onError((error, stackTrace) => print(error));
      });
}
```

The _StartReceivingReceipts method contains two methods the first being the _ReceiptFromFeed method and the second being the _RemoveDeliveredReceipt method.

The _receiptFromFeed method is a method that takes a Json object coming from the receipts stream and then converts it into a receipt object so it can be used with the rest of the application.

```
Receipt _receiptFromFeed(feedData) {
  var data = feedData['new_val'];
  return Receipt.fromJson(data);
}
```

The _removeDeliverredReceipt method removes the record corresponding to the receipt from the database. This is done to ensure the user's privacy and save storage space on the server.

```
_removeDeliverredReceipt(Receipt receipt) {
  r
      .table('receipts')
      .get(receipt.id)
      .delete({'return_changes': false}).run(_connection);
}
```

5.2.7 Datasource service

The Datasource service is responsible for managing the user's device local storage operations. The service is responsible for curating all the messages that arrive to the user and then sort them into separate chats storing them on the user's device local storage. This is required because each message that arrives is stored as a separate record in RethinkDB.

The local database that is used is the SQLITE database, a relational database that is widespread in the android development scene. The Datasource service has eight methods. The eight methods with their respective implementation.

5.2.7.1 addChat method

The addChat method inserts a record into the chats table.

```
Future<void> addChat(Chat chat) async {
  await _db.transaction((txn) async {
    await txn.insert('chats', chat.toMap(),
        conflictAlgorithm: ConflictAlgorithm.rollback);
  });
}
```

5.2.7.2 updateMessageReceipt method

The updateMessageReceipt method is a method that updates the state of the message receipt when the user receives or reads the message.

```
Future<void> updateMessageReceipt(String messageId, ReceiptStatus status) {

  return _db.transaction((txn) async {

    await txn.update('messages', {'receipt': status.value()},
        where: 'id = ?',
        whereArgs: [messageId],
        conflictAlgorithm: ConflictAlgorithm.replace);
  });
}
```

5.2.7.3 findAllChats method

The findAllChats method is a method that returns a list of chats that have unread messages in them.

```
    final chatsWithLatestMessage = await txn.rawQuery('''SELECT messages.* FROM
    (SELECT
      chat_id, MAX(created_at) AS created_at
     FROM messages
     GROUP BY chat_id
    ) AS latest_messages
    INNER JOIN messages
    ON messages.chat_id = latest_messages.chat_id
    AND messages.created_at = latest_messages.created_at
    ORDER BY messages.created_at DESC''');

    if (chatsWithLatestMessage.isEmpty) return [];

    final chatsWithUnreadMessages = await txn.rawQuery('''SELECT chat_id, count(*) as unread
    FROM messages
    WHERE receipt = ?
    GROUP BY chat_id
    ''', ['deliverred']);

    return chatsWithLatestMessage.map<Chat>((row) {

      final int unread = chatsWithUnreadMessages.firstWhere(
          (ele) => row['chat_id'] == ele['chat_id'],
          orElse: () => {'unread': 0})['unread'];

      final chat = Chat.fromMap({"id": row['chat_id']});

      chat.unread = unread;

      chat.mostRecent = LocalMessage.fromMap(row);

      return chat;

    }).toList();
  });
}

@override
Future<Chat> findChat(String chatId) async {
```

5.2.7.4 updateMessage method

The updateMessage method is a method that updates a certain message. This method will be used in the future to implement various features.

```
Future<void> updateMessage(LocalMessage message) async {

  await _db.update('messages', message.toMap(),
      where: 'id = ?',
      whereArgs: [message.message.id],
      conflictAlgorithm: ConflictAlgorithm.replace);
}
```

5.2.7.5 findMessages method

The findMessages method is a method that finds all the messages associated with a chat.

```
Future<List<LocalMessage>> findMessages(String chatId) async {
  final listOfMaps = await _db.query(
    'messages',
    where: 'chat_id = ?',
    whereArgs: [chatId],
  );

  return listOfMaps
      .map<LocalMessage>((map) => LocalMessage.fromMap(map))
      .toList();
}
```

5.2.7.6 addMessage method

This method inserts a message into the messages table

```
Future<void> addMessage(LocalMessage message) async {
  await _db.transaction((txn) async {
    await txn.insert('messages', message.toMap(),
        conflictAlgorithm: ConflictAlgorithm.replace);
  });
}
```

5.2.7.7 deleteChat method

The deleteChat method is a method that is used to delete a chat from the local storage. The method deletes the chat from the chats table and also it deletes all the messages associated with it from the messages table.

```
Future<void> deleteChat(String chatId) async {

  final batch = _db.batch();

  batch.delete('messages', where: 'chat_id = ?', whereArgs: [chatId]);

  batch.delete('chats', where: 'id = ?', whereArgs: [chatId]);

  await batch.commit(noResult: true);
}
```

5.2.7.8 findChat method

The findChat method is a method that takes a chatId as a parameter and returns that particular chat.

```
Future<Chat> findChat(String chatId) async {

  return await _db.transaction((txn) async {

    final listOfChatMaps = await txn.query(
      'chats',
      where: 'id = ?',
      whereArgs: [chatId],
    );

    if (listOfChatMaps.isEmpty) return null;

    final unread = Sqflite.firstIntValue(await txn.rawQuery(
        'SELECT COUNT(*) FROM MESSAGES WHERE chat_id = ? AND receipt = ?',
        [chatId, 'deliverred']));

    final mostRecentMessage = await txn.query('messages',
        where: 'chat_id = ?',
        whereArgs: [chatId],
        orderBy: 'created_at DESC',
        limit: 1);
    final chat = Chat.fromMap(listOfChatMaps.first);
    chat.unread = unread;
    chat.mostRecent = LocalMessage.fromMap(mostRecentMessage.first);
    return chat;
  });
}
```

5.2.7.9 numChats method

The numChats method is responsible for counting the number of chats for each user.

```
Future<int> numChats(){
  return _db.transaction((txn) async {
    final chatsWithLatestMessage =
        await txn.rawQuery('''SELECT messages.* FROM
    (SELECT
      chat_id, MAX(created_at) AS created_at
     FROM messages
     GROUP BY chat_id
    ) AS latest_messages
    INNER JOIN messages
    ON messages.chat_id = latest_messages.chat_id
    AND messages.created_at = latest_messages.created_at
    ORDER BY messages.created_at DESC''');

    if (chatsWithLatestMessage.isEmpty) return 0;

    final chatsWithUnreadMessages =
        await txn.rawQuery('''SELECT chat_id, count(*) as unread
    FROM messages
    WHERE receipt = ?
    GROUP BY chat_id
    ''', ['deliverred']);

    return chatsWithLatestMessage.map<Chat>((row) {
      final int unread = chatsWithUnreadMessages.firstWhere(
          (ele) => row['chat_id'] == ele['chat_id'],
          orElse: () => {'unread': 0})['unread'];

      final chat = Chat.fromMap({"id": row['chat_id']});
      chat.unread = unread;
      chat.mostRecent = LocalMessage.fromMap(row);
      return chat;
    }).toList().length;
  });
}
```

5.2.7.10 activeChat method

The activeChat method is responsible for counting the number of activeChats for a specific user

```
Future<int> activeChat(){
  return _db.transaction((txn) async {
    final chatsWithUnreadMessages =
        await txn.rawQuery('''SELECT chat_id, count(*) as unread
    FROM messages
    WHERE receipt = ?
    GROUP BY chat_id
    ''', ['deliverred']);
    return chatsWithUnreadMessages.length;
  });
}
```

5.2.7.11 unreadCount method

This method is responsible for counting the number of unread messages for a user

```
Future<int> unreadCount() async{
  final unread = await _db.rawQuery('SELECT * FROM messages WHERE receipt = ?',['deliverred']);
  return unread.length;
}
```

5.2.8 Analytics Service

The analytics service is responsible for the implementation of descriptive and diagnostic analytics functions. It takes two parameters: the ServerUrl and the UserId.

5.2.8.1 RecvCount method

The RecvCount method is responsible for counting the number of messages received by the user.

```
RecvCount(this.serverUrl,this.userId);

Future<String> countRecv(String id) async{
  var dio = Dio();
  dio.options.baseUrl = serverUrl;
  var route = '/recvcount/$userId';
  final response = await dio.get(route);
  return response.data;
}
}
```

5.2.8.2 CountUnread method

The CountUnread method is responsible for counting the number of messages unread by the user

```
class Countunread{
 final String serverUrl;
 final String userId;

 Countunread(this.serverUrl,this.userId);

 Future<String> countUnread(String id) async{
   var dio = Dio();
   dio.options.baseUrl = serverUrl;
   var route = '/countunread/$userId';
   final response = await dio.get(route);
   return response.data;
 }
}
```

5.2.8.3 ChatCount method

The ChatCount method is responsible for counting the number of chats for each user

```
class Chatcount{
 final String serverUrl;
 final String userId;

 Chatcount(this.serverUrl,this.userId);

 Future<String> chatCount(String id) async{
   var dio = Dio();
   dio.options.baseUrl = serverUrl;
   var route = '/chatcount/$userId';
   final response = await dio.get(route);
   return response.data;
```

```
}
}
```

5.2.8.4 ChatActive method

The ChatActive method is responsible for counting the number of unseen chats for each user

```
class ChatActive{
 final String serverUrl;
 final String userId;

 ChatActive(this.serverUrl,this.userId);

 Future<String> chatActive(String id) async{
   var dio = Dio();
   dio.options.baseUrl = serverUrl;
   var route = '/chatactive/$userId';
   final response = await dio.get(route);
   return response.data;
 }
```

5.3 Image server

This section is concerned with the implementation of the image server used to store and retrieve the users profile images. As mentioned before the image server is running an API written using the FastAPI web framework. The API has three routes, a description alongside the implementation of each route is provided below.

5.3.1 Root route

The root route is a GET route that returns a 200OK response. This route is mainly used for testing the connection to the API.

```
@app.get('/')
def root():
    return {
        "response":"200 Ok"
    }
```

5.3.2 Upload route

The upload route is a POST route that is responsible for uploading an image file to the server. After the image is uploaded a response is sent to the client detailing information about the uploaded file and the path it is stored in on the server so it can be retrieved later.

```python
@app.post('/upload')
async def upload(file: UploadFile = File(...)):

    content = await file.read()
    filename = file.filename
    mime = file.content_type
    path = f'http://192.168.0.139:8000/images/profile/{filename}'
    encodedMessage = base64.b64encode(content)
    #size = await file.size()

    with open(filename, 'wb') as buffer:
        buffer.write(content)
        buffer.close()
        await file.close()

    if(os.path.exists(f'images/profile/{filename}')):

        os.remove(f'images/profile/{filename}')


    else:

        shutil.move(filename, 'images/profile/')

    return {
        "file_name": filename,
        "file_type": mime,
        "path": path,
        "encoded_image_bytes": encodedMessage
        #"size": size
    }
```

5.3.3 GetImage route

The GetImage route is a GET route that takes an image name as a parameter and then returns the requested image file to the client.

```python
@app.get('/images/profile/{imagename}')
def getImage(imagename:str):
    return FileResponse(f"images/profile/{imagename}")
```

5.3.4 users route

```python
@app.get('/users')
def getUsers():

  r.connect( "localhost", 28015).repl()
  users = []
  cursor = r.table("users").run()
  for document in cursor:
     users.append(document)
  cursor.close()
  return users

@app.get('/users/{id}')
def getUser(id:str):

  r.connect( "localhost", 28015).repl()

  return r.db('test').table('users').get(f'{id}').run()
```

5.3.5 sentCount route

```python
@app.get('/sentcount/{id}')
def countSent(id:str):
  messages = []
  r.connect('localhost',28015).repl()
  cursor = r.table("messages-analytics").filter(r.row["from"] == f"{id}").run()
  for document in cursor:
     messages.append(document)
  count = len(messages)
  cursor.close()
  return str(count)
```

5.3.6 recvCount route

```
@app.get('/recvcount/{id}')
def countSent(id:str):
    messages = []
    r.connect('localhost',28015).repl()
    cursor = r.table("messages-analytics").filter(r.row["to"] == f"{id}").run()
    for document in cursor:
        messages.append(document)
    count = len(messages)
    cursor.close()
    return str(count)
```

5.3.7 updatestatus route

```
@app.post('/updatestatus/{id}')
def updateStatus(id:str):
    r.connect('localhost',28015).repl()
    r.table("users").filter(r.row['id'] == f"{id}").update({"last_seen":
r.expr(datetime.now(r.make_timezone('+00:00')))}).run()
    return {"update": f'active time set to {datetime.now()}'}
```

5.3.8 countunread route

```
@app.get("/countunread/{id}")
def countunread(id:str):
    messages = []
    messagesRead = []
    countReceipts = 0
    countMessages = 0
    countUnread = 0
    r.connect('localhost',28015).repl()
    cursor = r.table("messages-analytics").filter(r.row["to"] == f"{id}").run()

    for document in cursor:
        messages.append(document)

    cursor1 = r.table("receipts").filter(r.row["recipient"] == f"{id}").run()
    for document1 in cursor1:
        messagesRead.append(document1)
```

```
  countMessages = len(messages)
  countReceipts = len(messagesRead)

  if countReceipts == 0:
      return str(countMessages)
  elif countMessages == 0:
      return str(0)
  elif (countMessages-countReceipts) == 0:
      return str(0)
  else:
      return str(countReceipts)
```

5.3.9 chatcount route

```
@app.get("/chatcount/{id}")
def chatcount(id:str):
  messagesTo = []
  messagesFrom = []
  messagesFromId = []
  messagesToId = []
  count = 0

  r.connect('localhost',28015).repl()

  cursor1 = r.table("messages-analytics").filter(r.row["to"] == f"{id}").run()
  for document1 in cursor1:
      messagesTo.append(document1)

  cursor = r.table("messages-analytics").filter(r.row["from"] == f"{id}").run()
  for document in cursor:
      messagesFrom.append(document)

  for i in range(len(messagesTo)):
      messagesToId.append(messagesTo[i]['from'])

  for j in range(len(messagesFrom)):
      messagesFromId.append(messagesFrom[j]['to'])

  count = len(unique(messagesToId))
```

```
return str(count)
```

5.4 Conclusion

The implementation chapter focussed on how different components of MMUCord were implemented. First a discussion regarding the technology stack and the reason behind choosing every part of it. The technology stack used includes Flutter, FastAPI and RethinkDB. In addition to the technology stack overview the implementation of the services that make up MMUCord was discussed and presented. Finally, the implementation of all the routes present in the API were illustrated.

CHAPTER 6

TESTING AND VERIFICATION

6.1 Introduction

This section is focussed on testing the different components of the MMUCord system to ensure they are functioning as expected. Since MMUCord is still in its infancy and only exists the prototype, Ubuntu server and the image server there is no need to write sophisticated unit tests since there are not many testing scenarios therefore testing the components manually is sufficient. The components to be tested are the Ubuntu server, the image server, the API and the different services of MMUCord.

6.2. Ubuntu server

Testing the Ubuntu server involves making sure that the server is booting successfully, connected to the internet and that it accepts connections from other devices on the network.

To confirm that the server has booted successfully and accepts connections from other devices on the network an SSH connection is initiated on port 22 to verify that network operations are normal. The results can be seen from the screenshots below.

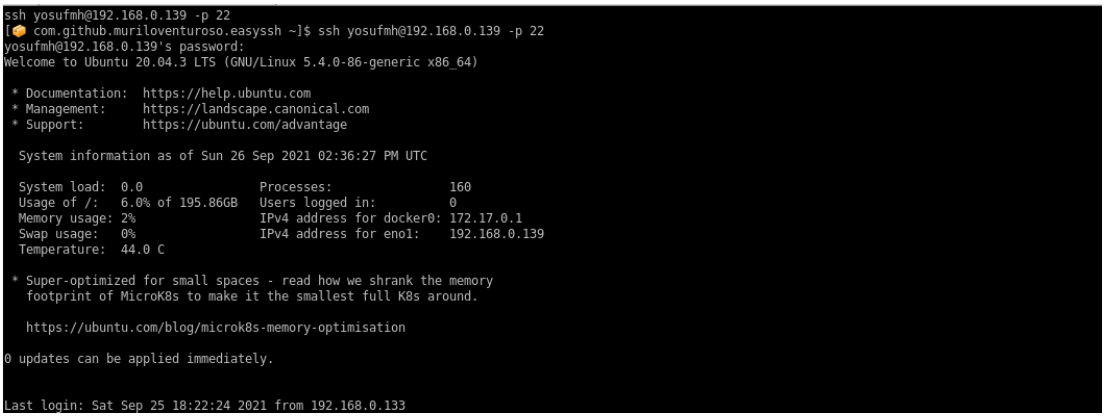

Fig 15 - SSH request

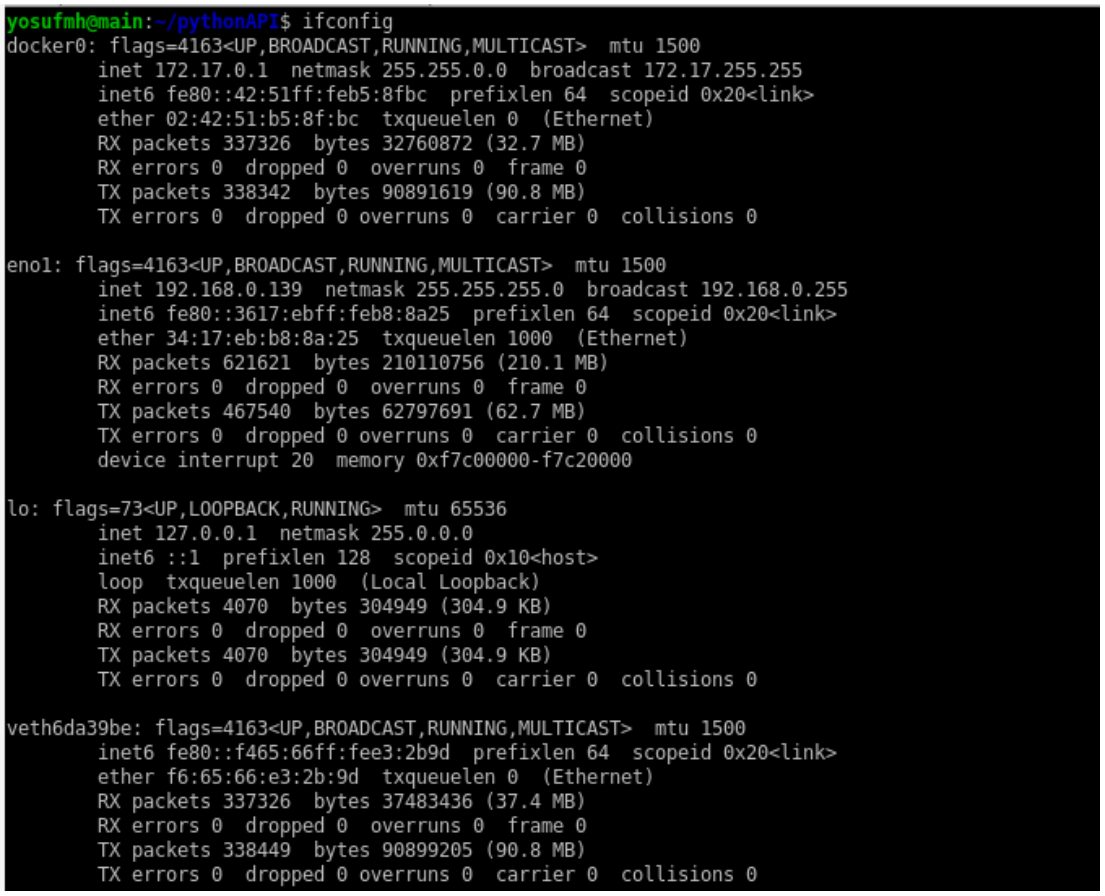

Fig 16 - ifconfig

From both the screenshots it is easy to confirm that the server has access to the internet and was given the ip 192.168.0.139. The screenshots also confirm that the server is accepting connections from other devices on the network hence the successful SSH connection.

6.3 Image Server

Testing the image server involves making sure that the server successfully loads and is able to process different HTTP requests.

Fig 17 - Image server run

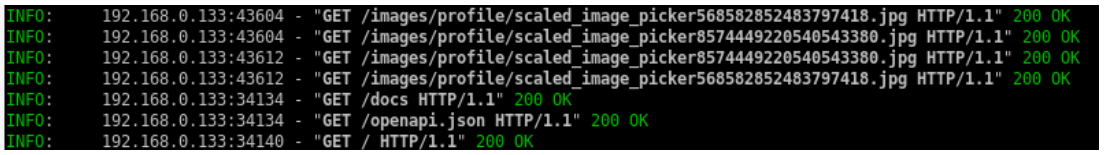

Fig 18 - Processing Requests

From both the screenshots it is confirmed that the server is running with no issues and is able to process GET requests successfully.

6.4 API

Testing the API involves testing all the API routes to make sure they act as expected. For testing the Swagger UI provided by FastAPI is used.

6.4.1 Root Route

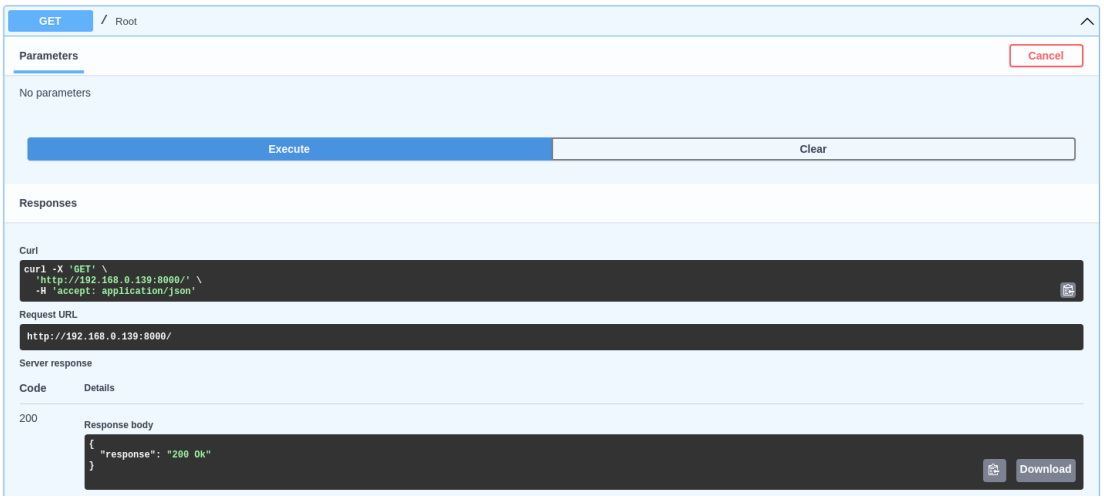

Fig 19 - Root route test

From the screenshot it is confirmed that the root route works as expected since it returns the expected response body.

6.4.2 Upload route

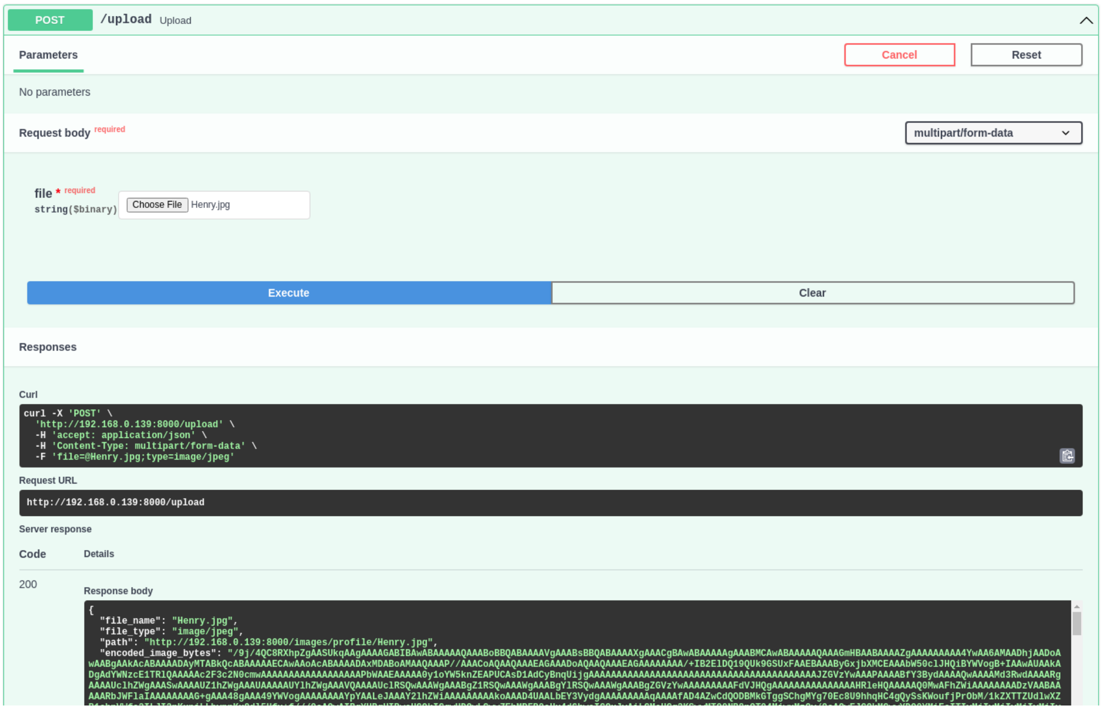

Fig 20 - Upload route test

The screenshot confirms that the upload route works as expected. It accepts a file as a parameter to be uploaded to the server. The screenshot confirms that the file is successfully uploaded and the expected response is returned.

6.4.3 Get Image route

The screenshot below confirms that the Get Image route works as expected. It accepts an image name as a parameter and returns the requested image successfully.

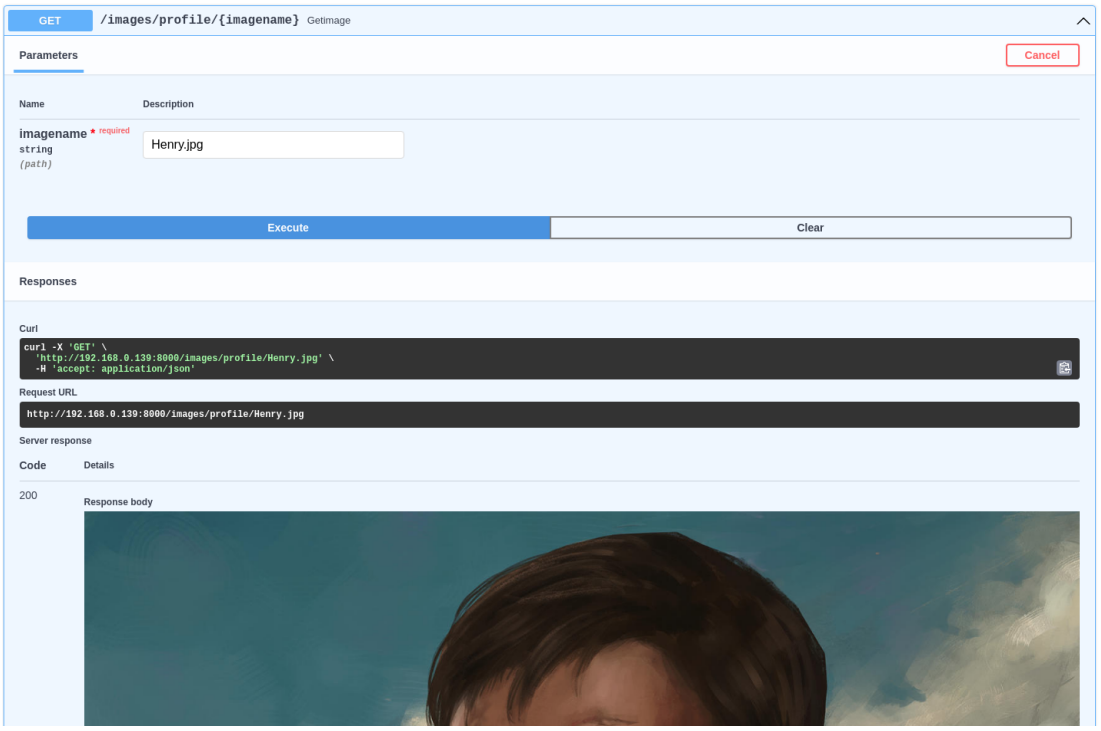

Fig 21 - Get Image route test

6.5 MMUCord services

To test and confirm that all the MMUCord services are working correctly, manual testing of the Flutter application will be used, every function will be tested multiple times in many different scenarios to ensure that everything functions correctly. The behavior of the user interface will be observed to identify changes that indicate that different services are working as they should. The screenshots below are proof that the services work as expected.

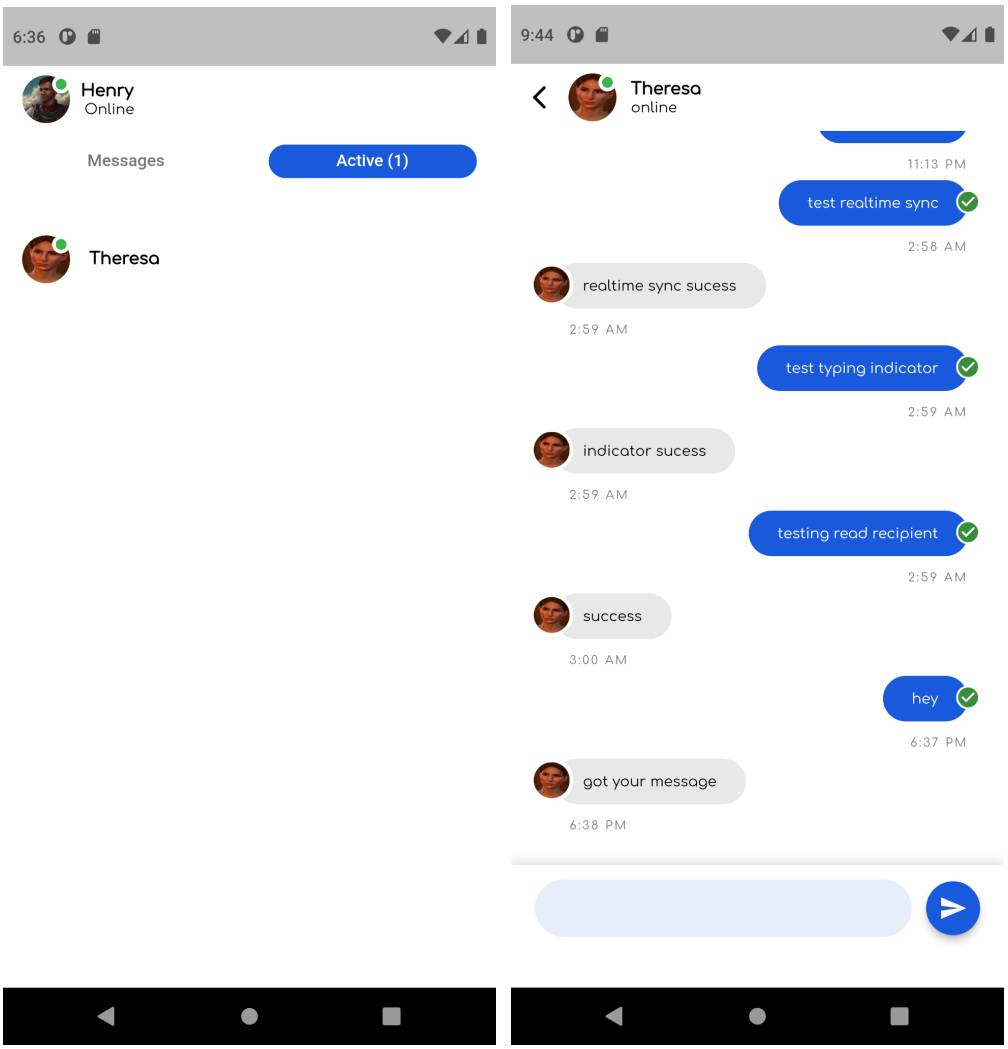

Fig 22 - active and message screens

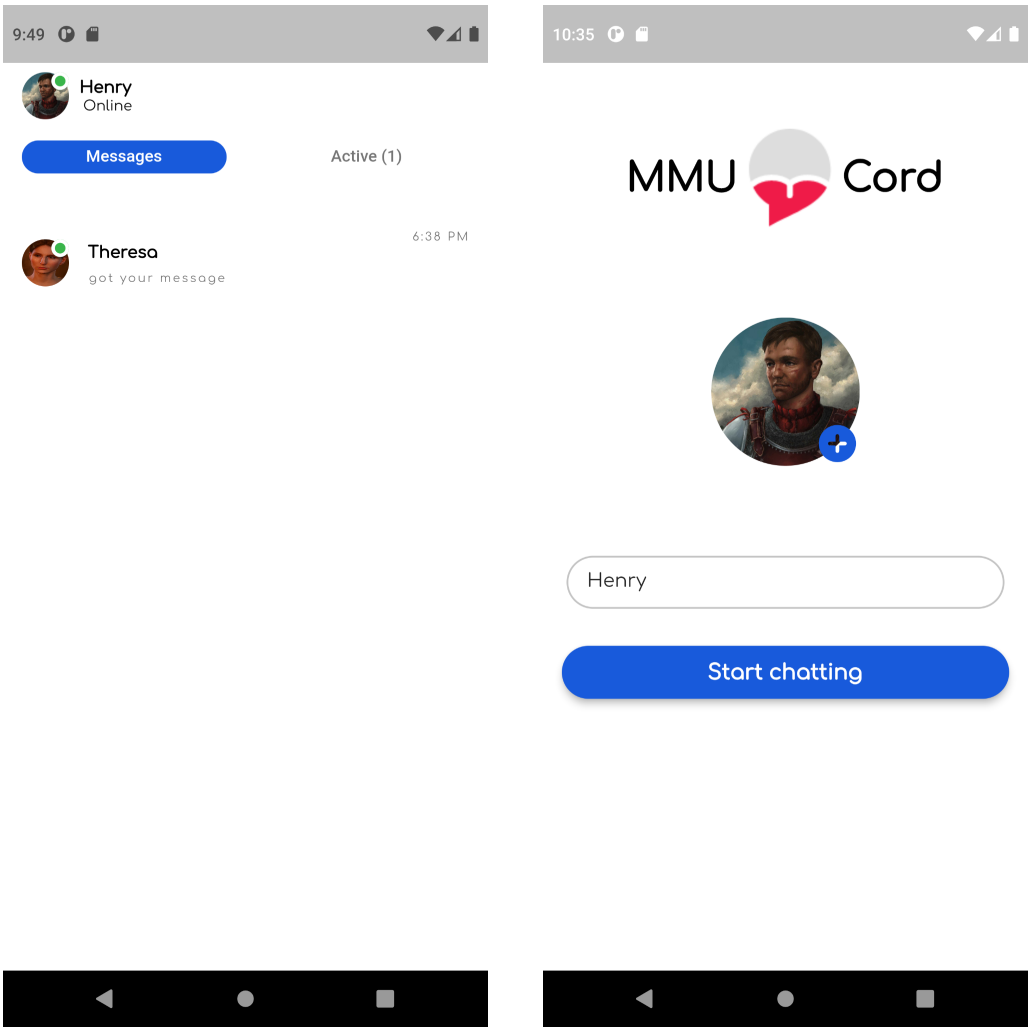

Fig 23 - messages and boarding screens

The first screenshot shows that the user is able to receive new messages and that the chats are correctly stored in their local storage. The second screenshot shows that the user is able to register successfully and that the data is correctly stored.

6.6 Conclusion

This chapter focused on testing the different components of the MMUCord system. The first component that was tested was the Ubuntu main server, the tests confirmed that the server is running and performing as expected. The second component that was tested was the image server, it was confirmed that the image server performs as expected and is able to accept and process HTTP requests successfully. The final component that is tested is the API, the API was tested using the Swagger UI provided by the FastAPI framework. All the routes in the API were tested and confirmed to work as expected.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

In the first Introduction chapter, it was illustrated that the COVID-19 circumstances led MMU to shift its operations to suit an online learning experience. In addition, the problems the shift to a fully online operation has given rise to was discussed in detail to provide a complete view of how this shift has affected the students and staff of the university. After discussing the circumstances and its after effects, a solution was provided in the form of the analytics powered instant messaging platform which gave birth to the MMUCord project. Following the birth of the MMUCord project, its objectives, scope and constraints were all defined and outlined in the first chapter.

The second chapter is more focussed on discussing similar work to MMUCord. First providing an overview of instant messaging as a concept and the benefits it provides. In addition, the second chapter discusses the three relevant applications in the instant messaging market, Facebook messenger, WhatsApp and Telegram. Providing an overview of their features and a comparison to see how the three applications differ from each other. Finally the second chapter dives deep into analytics, discussing the different types of analytics and how each can be used in an instant messaging application to provide benefit to the users. Concluding the second chapter is a table that illustrates how each analytics approach differs with an example.

The third chapter was concerned with the methodology used to bring the MMUCord platform into life. The Waterfall methodology was used to guide the

MMUCord project. The third chapter provides an overview of the Waterfall methodology as a whole and how its different stages were implemented in the context of MMUCord. In the third chapter the first stage of the waterfall method which is requirement analysis was discussed. The chapter discussed the techniques used for requirements gathering and prioritization in the MMUCord project. At the end of the third chapter a list of requirements were put together to start working on the proof of concept prototype.

Chapters four to six were concerned with the prototype developed, they discussed the process of designing, developing and testing the prototype following the stages in the waterfall method.

The fourth chapter was about MMUCord's system design. The fourth chapter provided a detailed view of every component in MMUCord's system. The fourth chapter discussed the main server and how it is deployed and configured to host both the database and the image server. Moreover, MMUCord was divided into 7 main services each responsible for managing one aspect of the application. In addition the fourth chapter discussed both the database and the image server individually. Illustrating how the database is configured and how the tables inside it were structured. Moreover, the fourth chapter went through how the image server accomplishes its functionality using uvicorn and FastAPI. Finally, the fourth chapter illustrated and presented the different screens that make up MMUCord's user interface.

The fifth chapter was more focussed on the implementation phase. First the fifth chapter provided a brief overview about the different technologies used to develop each component of the system. In addition to discussing the technologies used, the fifth chapter also discussed and presented the code related to each of the services, the image server and the API.

The sixth chapter was more focused on testing the different components that make up MMUCord. The chapter discussed how each component was tested to

ensure it was functioning correctly. To make sure that the Ubuntu server was functioning correctly two tests were initiated, the first one was to make sure that the server had internet access and the second was to ensure that the server was accepting connections from other devices on the network. For the testing of the main services the flutter application was manually tested to see if all the services were functioning and integrating together correctly. The testing of the user interface confirmed that all the services are working as expected and that the user interface has no bugs or glitches. To confirm that the image server is working correctly, different requests were sent to it to examine how it would handle such requests. After the testing it was confirmed that the image server was functioning as expected. Finally, the sixth chapter illustrated how the API was tested, the API was tested using the SwaggerUI provided by FastAPI. All the routes were extensively tested and the results show that the routes all work as expected.

7.2 Future Work

In the future, the application will be further improved to more resemble the vision behind MMUCord. The core messaging experience will be improved to add more futures. Moreover, a more secure and robust user authentication system will be added. In addition, the user interface will drastically change to more suit the analytics features to be added in later revisions of the application. Finally, the code will be refined to provide better readability and performance.

References

About Insights for Your WhatsApp Business Account. (n.d.). Facebook Business Help Center; Facebook. Retrieved August 15, 2021, from https://www.facebook.com/business/help/338500813332755?id=2129163877102343

About WhatsApp. (2019). WhatsApp.com; WhatsApp. https://www.whatsapp.com/about/

Ajam, M. (2017). *Project Management beyond Waterfall and Agile (Best Practices in Portfolio, Program, and Project Management)* (First). Auerbach Publications.

Business Analytics. (2019). Ibm.com; IBM. https://www.ibm.com/analytics/business-analytics

Diagnostic Analytics. (n.d.). Sisense; Sisense. Retrieved August 5, 2021, from https://www.sisense.com/glossary/diagnostic-analytics/

Gibson, P. (2018). *Types of Data Analysis*. Chartio; Chartio. https://chartio.com/learn/data-analytics/types-of-data-analysis/

Messaging apps. (2021, August 1). Wikipedia; Wikipedia. https://en.wikipedia.org/wiki/Messaging_apps#:~:text=History%20of%20messaging%20applications

Msc Bucero. (2017). *INFLUENTIAL PROJECT MANAGER : winning over team members and stakeholders.* Auerbach Publications.

ON, M. 'MiMi' A.-T. (2021, January 29). *The Evolution of Instant Messaging*. Medium. https://andermel.medium.com/the-evolution-of-instant-messaging-b77629a662ee

Predictive Analytics. (n.d.). Www.ibm.com; IBM. Retrieved August 6, 2021, from https://www.ibm.com/ae-en/analytics/predictive-analytics

Predictive Analytics: What it is and why it matters. (n.d.). Www.sas.com; SAS Analytics Software & Solutions. Retrieved August 4, 2021, from https://www.sas.com/en_ae/insights/analytics/predictive-analytics.html#dmworld

Prescriptive Analytics. (n.d.). Www.ibm.com; IBM. Retrieved August 6, 2021, from https://www.ibm.com/ae-en/analytics/prescriptive-analytics

Rawat, A. S. (2021, March 31). *What is Descriptive Analysis?- Types and Advantages | Analytics Steps*. Www.analyticssteps.com. https://www.analyticssteps.com/blogs/overview-descriptive-analysis

Tschakert, N., Kokina, J., & Kozlowski, S. (2016). The next frontier in data analytics. *Journal of Accountancy*, *8*(2016).

Twigby. (2020, June 17). *Survey Says: Cell Phone Usage Impacted During COVID-19*. Twigby.com; Twigby. https://www.twigby.com/blog/survey-says-cell-phone-usage-impacted-during-covid-19/?utm_source=pr%20newswire&utm_medium=release&utm_campaign=may_2020_survey

Wheelwright, T. (2021, April 21). *Cell Phone Behavior Survey: Are People Addicted to Their Phones?* Reviews.org. https://www.reviews.org/mobile/cell-phone-addiction/#2021_Cell_Phone_Behavior

Wikipedia Contributors. (2019, March 10). *Waterfall model*. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/Waterfall_model

Wolff, R. (2020, August 12). *Semantic Analysis: What Is It & How Does It Work?* MonkeyLearn Blog; MonkeyLearn. https://monkeylearn.com/blog/semantic-analysis/