National College of Ireland

BSHCSD4

Software Development

2022/2023

Joswel Bautista

x19369011

x19369011.student.ncirl.ie

Kohi

Mobile Application

Technical Report

Contents

Executive Summary

This paper describes the goal and functionality of my Final Year Project, a productivity mobile application. Kohi is a productivity mobile application designed to help individuals manage and track their daily tasks, projects, and goals.

Kohi features a notes-taking functionality in a diary that allows users to easily create, organize, and prioritize tasks, as well as set reminders and due dates to stay on track. which makes it easy to break down complex projects into manageable steps. Kohi notes-taking functionality is designed with simplicity and ease-of-use in mind making it accessible to users of all skill levels. With a clean and intuitive interface users can quickly and easily create and manage their tasks with just a few taps. Users can easily search and filter their tasks to find what they need.

The app includes a focus timer to help users in maximizing their productivity. Kohi includes a comprehensive analytics system that tracks the user productivity sessions. This system allows the users to easily identify areas of improvement and set goals to improve their performance. By providing actionable insights Kohi encourages users to make data-driven decisions that can have a significant impact on their productivity and overall success.

Kohi also includes a unique feature that adds a fun and interactive element to the productivity process. By using the focus timer consistently users are rewarded with various virtual items, such as cute backgrounds and timer icons. This makes it similar to a virtual slot machine but in a friendly way. This reward system not only motivates users to stay on task but also adds a layer of excitement and engagement to their productivity journey by collecting all the collectable rewards.

The Gacha system is designed to encourage users to consistently use the focus timer with the reward system acting as a form of positive reinforcement. By rewarding users with virtual items Kohi creates a sense of accomplishment and satisfaction further motivating users to continue using the app and improving their productivity. Additionally, the Gacha system is a creative way to track and reward user progress encouraging users to challenge themselves and set new goals.

Overall, Kohi is a comprehensive productivity mobile application that offers a range of features to help users stay organized, focused, and productive. Whether managing daily tasks or complex projects Kohi is a valuable tool for anyone seeking to improve their productivity and achieve their goals more effectively and its user-friendly interface makes it easy to use for anyone.

1.0 Introduction

1.1. Background

I have undergone this project because I was interested in designing a comprehensive mobile application for my final year project. This presented a challenge since mobile development was not covered in my college courses meaning I had to learn new tools and skills to develop the app. Nonetheless, I was determined to create a robust and feature-rich productivity application that would help users stay organized, prioritize tasks, track their productivity, and motivate them to achieve their goals.

My interest in developing Kohi came from my own struggles with staying organized and productive. I found that I often had trouble prioritizing my tasks and staying focused on my work. I also struggled with managing my time effectively and felt like I could benefit from a tool that would help me improve my productivity.

As I began working on the project, I realized that there were no existing mobile applications that offered all the features I was looking for. I saw an opportunity to create a comprehensive productivity app that could help others like me manage their tasks and goals more effectively.

The project has been challenging but rewarding. I had to learn new tools and technologies to develop the application which included programming languages like Kotlin as well as mobile development frameworks like Android Studio. I also had to conduct user research and design the app's user interface to ensure it was intuitive and easy to use.

1.2. Aims

This is a productivity application. It will allow a user to be more productive when it comes to being productive. When using the app, it will encourage the user to be more productive since they will be rewarded for using the focus timer, At the end of the timer it will reward the user with a random reward which they can collect. The user will also be able to write notes in a diary in order to be organised to know what they need to focus on that time. Lastly an analytics page to show how long the user has studied for through the day or the week to show their progress.

My aim with Kohi is to create a versatile productivity application that could be used by anyone, whether they are a student, professional, or just someone looking to stay on top of their daily tasks and goals. With its user-friendly interface and powerful features, Kohi will be a valuable tool for anyone looking to boost their productivity and accomplish their goals.

My application aim would be to help individuals stay organized and productive, regardless of the task at hand. My goal is to design an app that could be used for studying, work, or even social activities such as exercising.

To achieve this goal, The app includes a focus timer that rewards users for completing tasks, whether it be studying for an exam, completing a work assignment, or exercising for a set period of time. At the end of each task, users are rewarded with a coffee bean that they can use to redeem a reward. This aims to motivate them to continue using the app.

In addition to the focus timer, Kohi includes a notes-taking feature that enables users to create and organize tasks, set reminders, and prioritize their work. Users can categorize their task based on their work for greater organization.

Finally, Kohi includes an analytics page that shows users how long they have spent on tasks throughout the day or week. This page helps users track their progress and identify areas where they may need to improve their productivity.

1.3. Technology

Technology I have decided to use is Android Studio when it comes to creating the application. For my CI/CD I have decided to use GitHub which lets me do commits to different branches when needed. For packages in Android studio, I am deciding between Room Database or Firebase as my database. Since I am still in the production stage more technology will be used along the way such as flutter for the UI designing but for now, I have decided to create a simple UI for the prototype.

I have decided to use Android Studio as the primary technology for developing the mobile application. Android Studio provides an easy-to-use platform with a range of built-in tools and resources for creating high-quality applications.

To manage the codebase and ensure smooth development, I have chosen to use GitHub for Continuous Integration and Continuous Deployment (CI/CD). GitHub enables me to maintain a comprehensive history of my project allowing me to track changes, revert to previous versions, experiment with different features without compromising the integrity of my codebase. By utilizing GitHub's pull and push functionalities I can easily manage my project's progress and integrate new features to ensure a consistency across all versions and to allow me to stay organized throughout the development process.

In my project I have decided to use Firebase as my database. Firebase is a powerful and flexible database system that offers several advantages for mobile application development. One of the main benefits of using Firebase is that it allows for real-time data synchronization, which means that any changes made to the database will be instantly reflected in the app. This feature is particularly useful for my productivity app, as it will allow users to quickly update and view their tasks and progress without having to manually refresh the app.

Another advantage of using Firebase is its scalability. As my app grows and more users begin to use it, Firebase will be able to handle the increased demand without sacrificing performance or reliability. Additionally, Firebase offers a wide range of features, such as user authentication, cloud storage and cloud functions which is used in my application.

1.4. Structure

This document is structured to provide an overview of the development process of my project outlining the key areas of focus and addressing the details of each section. It is divided into several sections which include the system requirements, design and architecture, implementation, graphical user interface (GUI), testing, evaluation, conclusions, further development, references, and appendices. The section on system requirements provides a detailed analysis of the functional, non-functional, data, user, and environmental requirements, as well as the usability requirements. The design and architecture section covers the high-level design of the application, while the implementation section discusses the implementation of the application. The graphical user interface section addresses the application's user interface design. The testing section explains the testing procedures, and the evaluation section covers the assessment of the application. The conclusions section presents the overall findings of the project, while the further development section outlines potential areas for future development. The references section lists the sources used in the project, while the appendices section includes any additional materials that support the project.

2.0 System

2.1. Requirements

In order to ensure the functionality and usability of my application all requirements outlined must be verifiable. It is essential that experienced controllers are able to proficiently utilize all system functions after a total of two hours of training. Following this training the average number of errors made by experienced users should not exceed two per day. Additionally, the main GUI must be fully functional and allow users to use the Focus Timer, which is currently set to 30-minute increments. Following the timer, users should be able to make notes, redeem their prize points, and view their data in the analytics page to track their progress during the 2-hour training period. These requirements are integral to the overall success and efficacy of the application.

2.1.1. Functional Requirements & Non-Functional Requirements

These are the functional requirements for the application ranked in order of importance.

1. The application must be able to run on a mobile device: This is the most important requirement since the application won't be functional if it cannot run on a mobile device.
2. The application must be able to interact with the user: The user must be able to interact with the application to achieve their desired goals.
3. The application must be able to store and retrieve data: Storing and retrieving data is crucial for many mobile applications as they need to remember user settings, preferences, and activities.
4. The user should be able to save, create, delete, and edit their data: This is important for applications that involve creating content or managing information.
5. The user should be able to track their analytics: Analytics tracking can provide valuable insights for users and developers alike.

6. The user should be able to choose whether to save their data or not: While not as important as the other requirements, this still provides flexibility for users.
7. The user should be able to start, pause, and reset the timer.
8. The user should be able to summon content giving them a user interaction.

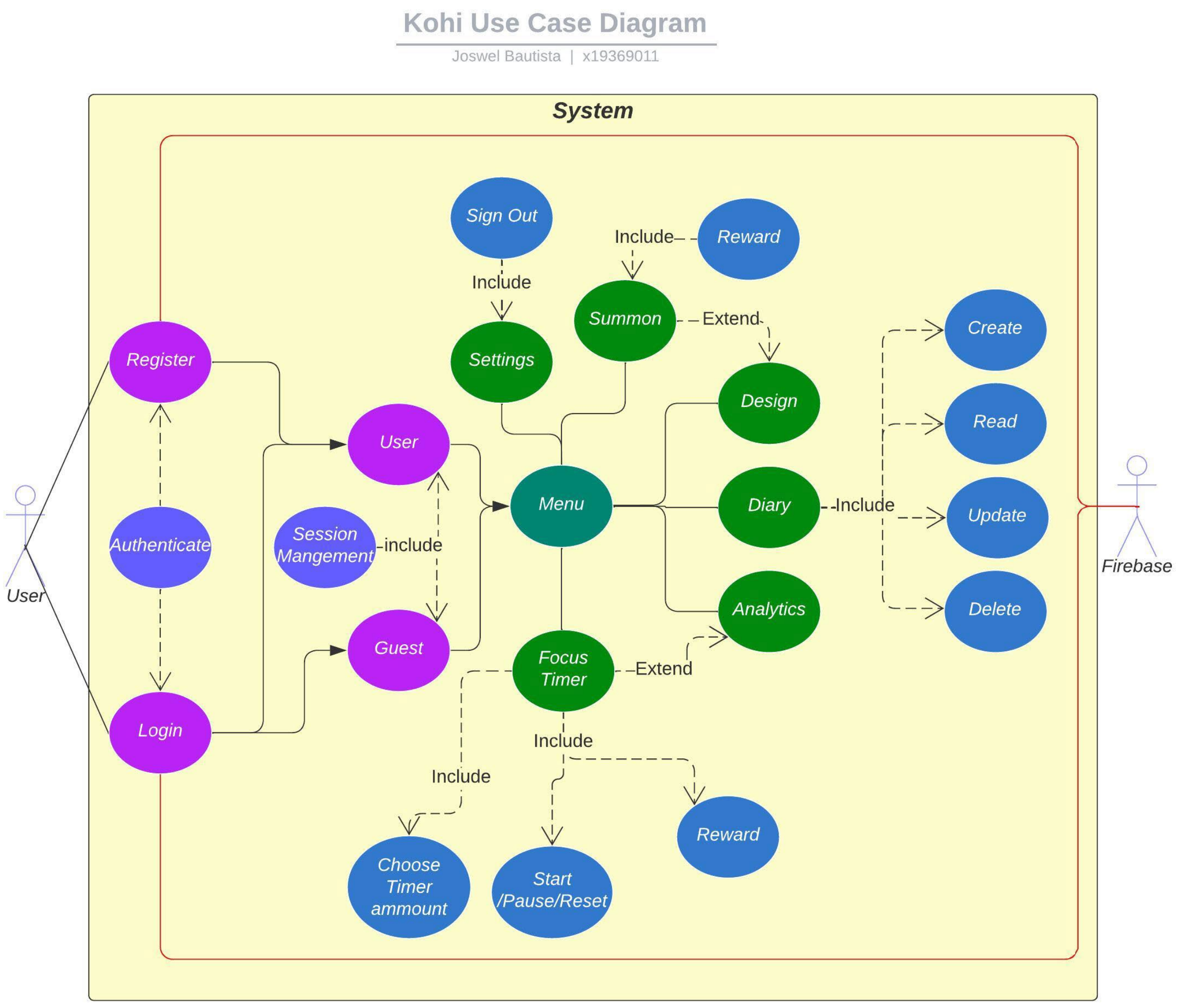
It will run smoothly on an Android SDK. The application has been designed to be compatible with various mobile devices, allowing for consistent functionality and user experience across different platforms. The system stores data locally on the user's device and retrieves it when needed including user preferences, settings, diary notes, and timer data. The application interacts with the user by displaying content and accessing device memory. It provides the ability to start, pause, and reset a timer and enables users to create, save, delete, and edit their diary entries. Users can summon content and track their analytics within the application. Additionally, users have the option to choose whether they want their data saved or not. The application's functional requirements have been met, providing users with a comprehensive and user-friendly experience.

Non-functional requirements are just as crucial as functional requirements in the development of a mobile application. It's important to prioritize the most critical requirements first, such as usability and security followed by performance, scalability, and ease of use. The following is a ranking of the non-functional requirements in order of importance for our application:

1. The system should be usable on a variety of mobile devices, with the same functionality and usability across platforms. This is critical to ensure that all users can access and use the application regardless of their device preference.
2. The system should also be secure, capable of protecting data and preventing unauthorized access. This is important to ensure that user data and information are protected from potential threats or breaches.
3. The application should have a response time of less than 1 second. This is crucial to provide users with a seamless experience and prevent frustration or dissatisfaction with slow loading times.
4. The application should be able to handle large amounts of data without compromising performance. This is important to ensure that the application can store and handle all user data without slowing down or crashing.
5. The application should be scalable to accommodate future updates and features. This is important to allow for future development and improvements to the application.
6. The system should be simple to use, with a simple user interface and simple navigation. While this is still important, it falls lower on the list since it is more subjective and can be improved through user feedback and testing.

2.1.1.1. Use Case Diagram

The use case contains several components such as Login/Register, Menu, focus timer, Diary, Analytics and summon. These components describe how a user will interact with the mobile application, including how they will log in or register, navigate through the menu, use the focus timer, create, read, update, and delete (CRUD) diary entries, track their analytics, and summon for content. Each of these components plays a crucial role in ensuring that the mobile application is functional and user-friendly.



The use case diagram provides a clear understanding of the relationship between the various functionalities of the application. The user begins by accessing the login page, where they have the option to either register or login as a user or guest. Authentication takes place and once the user logs in their session is saved until they choose to sign out of the account. The firebase database is used to retrieve and store the user data based on the unique UID given to the user during registration or login. This ensures that user data is saved and retrieved only to the specific user. This is done once the user registers or logs into their account.

Upon a successful login the user is directed to the Menu which features four main functionalities which includes the Focus Timer, Diary, Analytics, and Summon, along with two side functionalities Settings and Design. The Focus Timer enables the user to choose a session length in increments of 30 minutes. It includes a reward system, start, pause, and reset buttons. Data on successful focus sessions is saved to the analytics section. If the user leaves early or chooses not to work/focus the data is saved. This data is saved to firebase, and it extends to the Analytics class. The Analytics feature displays user data on how long they used the Focus Timer for.

The Diary feature allows the user to create, read, update, and delete notes, helping them stay organized and on track with their tasks. The Diary data is saved based on the user UID, and it is retrieved from the firebase database using the same UID.

The Summon feature extend the rewards received from the Focus Timer. Users can redeem points to receive rewards and design their app based on those rewards. The Settings section provides an overview of the application and includes a sign-out feature for users to end their session.

These Use case diagrams have been drawn using Lucidchart.com.

2.1.1.2. Requirement 1: Login/Register Functionality

This requirement allows the user to create sessions and end sessions in the application.

Description & Priority

Requirement 1: Login/Register Functionality

The Login/Register functionality is an essential aspect of the application which provides users with the ability to create an account and access the main features of the application. This requirement includes a robust authentication system that ensures that only authorized users can access the apps menu. Once authenticated the user's session is saved until they choose to log out and their data is stored securely in the Firebase database. The user's unique ID is used to identify their data and ensure that no other user can access their information. The requirement also includes a guest login option for users who do not wish to create an account which provides a hassle-free experience but when they choose to sign out their data will be deleted.

The Login/Register functionality is of high priority for the overall system as it is essential for the user's privacy and data security. It is a fundamental requirement that must be implemented correctly to ensure the system's properly functioning. Failing to implement this requirement correctly can lead to security breaches, loss of user data, and a negative impact on the user experience. Therefore, it is crucial to prioritize this requirement and ensure that it is implemented correctly.

Use Case

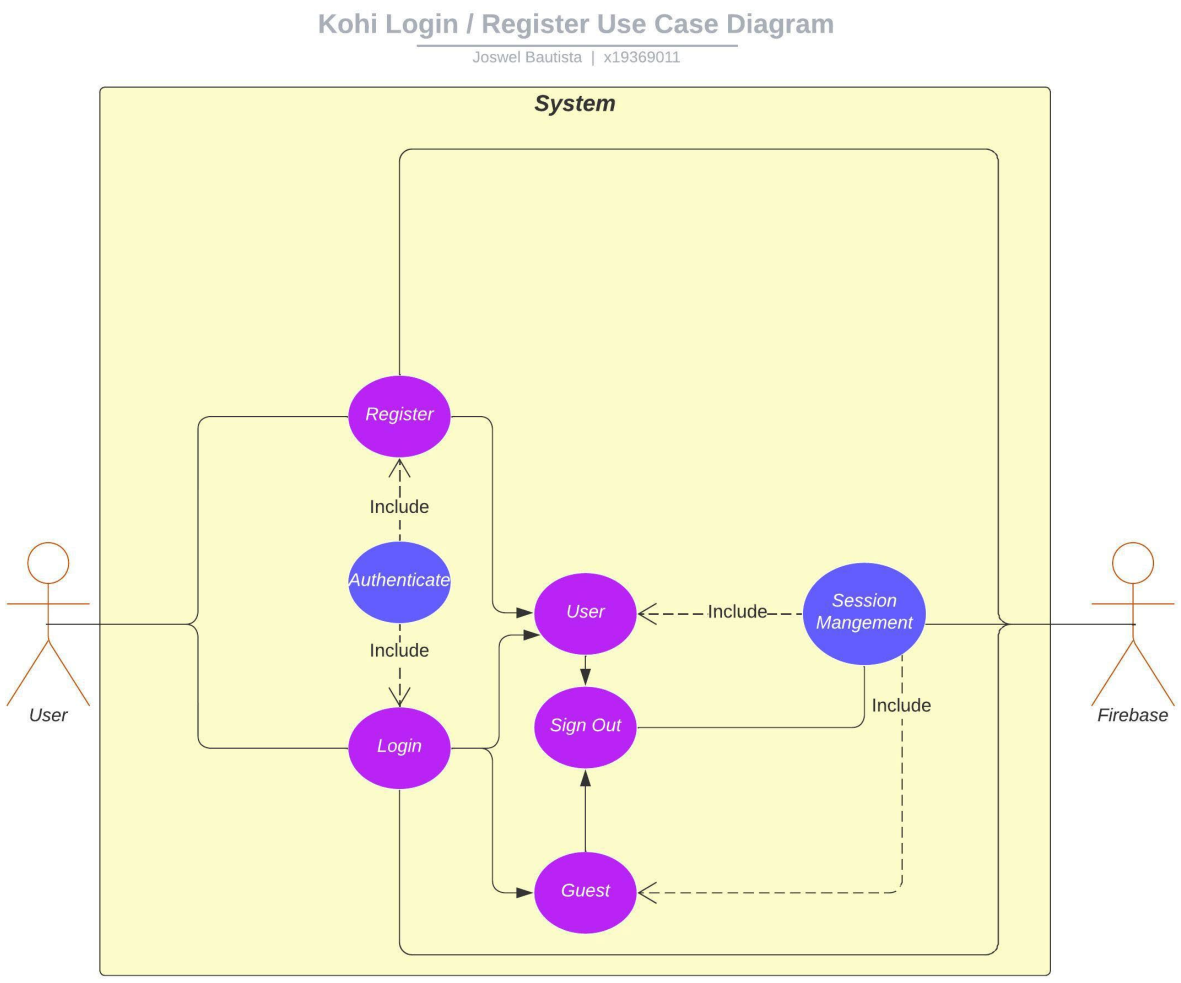
Scope

The scope of this use case is to showcase the Login/Register process within the application.

Description

This use case illustrates how the user interacts with the Login/Register functionality of the application. Initially the user is presented with the Login page where they have the option to either log in as a registered user or log in as a guest user. If the user is a new user, they can register on the registration page which will log them in as a registered user. The login process includes an authentication step where the user's credentials are verified through the Firebase database. Once authenticated the user session is saved allowing them to access the main features of the application. If the user chooses to sign out their session will end.

Login/Register Use Case Diagram UC-v1.1



Use Case Flow Description:

Use Case ID:	UC-v1.1

Use Case Name:	Login/Register
Actors: Primary Secondary	 Primary Actor is the User who enters the System. Secondary Actor is Firebase Database
Activation:	User launches the application
Pre-conditions:	User must have the application installed and running
Post-Conditions:	User is authenticated and logged in as a registered user or guest user or user registration is completed successfully
Main Flow:	1. The user opens the application and is presented with the Login/Register page. 2. User clicks on the "Register" button to create a new account or on the "Login" button to log in as an existing user or as a guest. 3. If the user chooses to register, they are directed to the Registration page. 4. On the Registration page, the user enters their email address and password. 5. The system verifies that the email address is not already registered, and if it is, the user is notified and prompted to enter a different email address. 6. If the email address is not already registered, the system creates a new user account for the user and saves their information to the Firebase database. 7. The user is automatically logged in and redirected to the main menu. 8. If the user chooses to log in as an existing user, they enter their email address and password on the Login page. 9. The system verifies the user's email and password against the Firebase database. 10. If the email and password match, the user is logged in and directed to the main menu. 11. If the email and password do not match, the user is notified and prompted to enter the correct email and password. 12. If the user chooses to log in as a guest, they are directed to the main menu without the need to enter any credentials. 13. If the registered user chooses to sign out their session will end.
Alternate Flows:	1. If the email address is already registered, the user is prompted to log in with their existing account or to enter a different email address.

	2. If the user enters an invalid email address, the system notifies the user and prompts them to enter a valid email address. 3. If the user's email or password is incorrect, the system notifies the user and prompts them to enter the correct email and password. 4. If the user's email is not found in the Firebase database, the system notifies the user and prompts them to enter a valid email address or to register as a new user. 5. If the passwords do not match, the system displays an error message indicating that the passwords must match and prompts the user to re-enter their password.
Exceptional Flows:	• If the user enters incorrect login credentials they will be prompted to try again. • If there are issues with the Firebase database or authentication process an error will be prompted. • If the user encounters technical issues during registration an error will be prompted.
Termination	• User chooses to log out of their session or when the guest user's session ends. • User decides to close the application. • System errors or technical issues, the use case will terminate with an error message displayed to the user
Includes:	Includes Authenticate and Session Management
Extends:	There are no extends in this use case for this application in UC-v1.1
Priority:	High
Special Conditions:	None
Assumptions:	The user has a device with internet connectivity
Notes and Issues:	The Firebase database may have delays or errors in saving and retrieving data due to the network or server issues.

2.1.1.3. Requirement 2: Focus Timer Functionality

This requirement allows the user to use the focus timer in the application.

Description & Priority

Requirement 2: Focus Timer Functionality

The Focus functionality is an essential aspect of the system as its where the user will interact with the application and engage in productive focus sessions. With this functionality, the user is rewarded for every successful focus session they complete with rewards varying based on the length of the session. Additionally, the system records the amount of time users have studied allowing them to track their progress throughout the week or day. This functionality is the foundation upon which other features such as the Summon activity and analytics are built. Without it, the application would be of limited use to users. Therefore, it is vital that the Focus functionality is implemented correctly and is fully functional.

The Focus Timer functionality is a high-priority requirement since it is the central feature of the application. It allows users to track their focus progress and receive rewards. Without the Focus Timer functionality, the application will not serve its purpose and users will not be able to track their focus sessions progress effectively and receive rewards.

Use Case

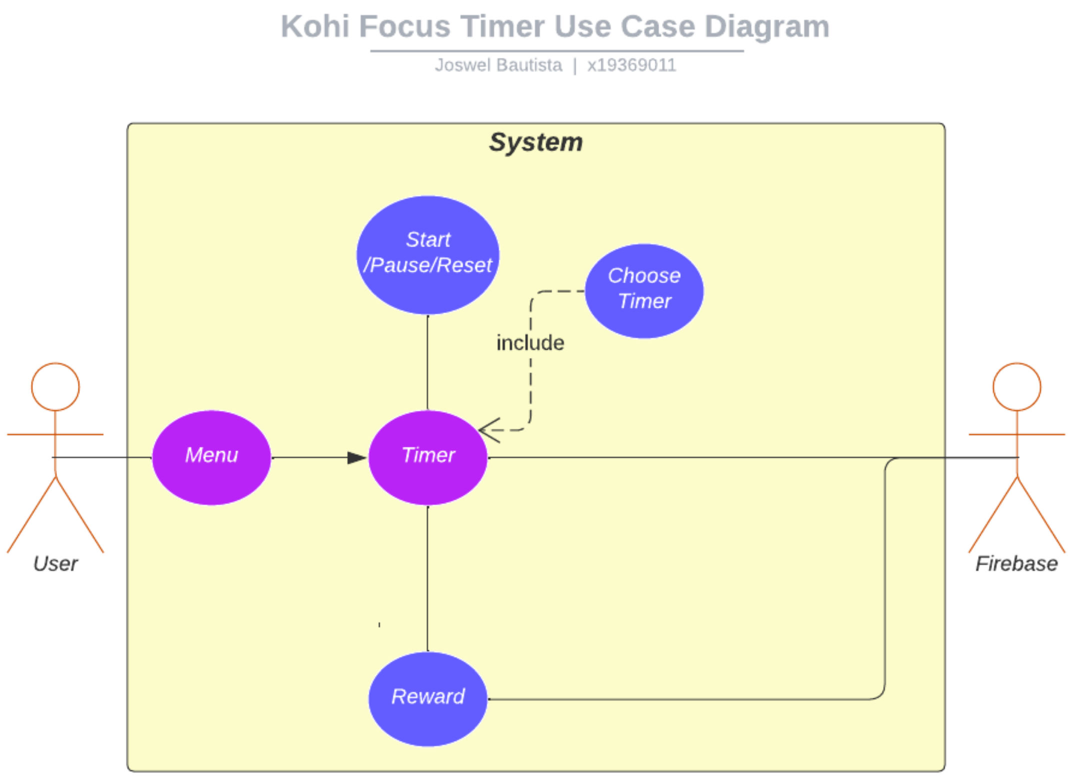
Scope

The scope of this use case is to illustrate the Focus Timer functionality of the application which includes the user's interaction with the Focus timer, and the recording and saving of the user's activity.

Description

This use case shows the user's interaction with the Focus Timer functionality in the application. The user is provided with a menu to select the duration of the timer session they wish to undertake. Upon selecting a duration, the user can initiate, pause, or reset the timer as necessary. Upon completion of the timer, the user will be rewarded with points based on the duration of the session where its then saved to the Firebase database. The timer duration is saved in the firebase database. This includes whether the timer was stopped midway, or the focus session was completed in full.

Focus Timer Use Case Diagram UC-v1.2



Use Case Flow Description:

Use Case ID:	UC-v1.2
Use Case Name:	Focus Timer
Actors: Primary Secondary	Primary Actor is the User who enters the System. Secondary Actor is Firebase Database
Activation:	User selects the Focus Timer option from the main menu
Pre-conditions:	1. The user must have successfully logged into the application. 2. The user must have selected the Focus Timer option from the main menu.
Post-Conditions:	1. The user can start, pause, and reset the focus timer. 2. Upon completion of the focus session, the user is rewarded, and their points are saved to Firebase. 3. The timer duration is saved to the firebase database, along with the session completion status and reward amount.
Main Flow:	1. The user opens the application and selects the "Focus Timer" option from the main menu. 2. The system displays the "Focus Timer" screen which includes a selection of timer durations (such as 30 minutes, 60 minutes, or 90 minutes).

	3. The user selects a timer duration and clicks the "Start" button to begin the focus session. 4. The system starts the timer and displays the remaining time on the screen. 5. The user can pause the timer at any time by clicking the "Pause" button. 6. The system stops the timer and displays the remaining time on the screen when the user pauses the timer. 7. The user can resume the timer by clicking the "Resume" button. 8. The system resumes the timer and displays the remaining time on the screen. 9. When the timer reaches zero, the system displays a notification that the focus session is complete. 10. The system awards the user points based on the length of the focus session and saves the points to the user's profile on Firebase. 11. The system also saves the length of the focus session to the on Firebase. 12. The user can click the "Back" button to return to the main menu.
Alternate Flows:	1. If the user clicks the menu button in the focus timer operation: The system stops the timer and returns the user to the main menu no points awarded, and the timer is saved. 2. If the user completes the focus session: The system displays a "Session Complete" message on the screen. The user is awarded points based on the length of the focus session and the points are saved to the Firebase database. The user is directed to the main menu. 3. If the user quits the focus session before completion: The system displays an "Application Closed" message on the screen. no points will be rewarded, and the session will be recorded in the firebase database as an incomplete session.
Exceptional Flows:	• If the user loses their internet connection while using the timer, the system will check if there is a cached copy of the Firebase data available. • If a cached copy is available, the system will use this data to allow the user to continue using the timer while offline. • When the device regains internet connection, the system will save the cached data to the Firebase database to update any changes that were made while the device was offline.

	• If no cached data is available, the system prompts an error. • If there are issues with the Firebase database or an error will be prompted.
Termination	• The user decides to exit the application by closing it or navigating away from it. • Upon completion of the timer, the user will be granted points and the data will be securely saved to Firebase. The timer will then halt, and the user will receive a prompt that redirects them to the main menu.
Includes:	Includes choosing the timer amount
Extends:	None
Priority:	High
Special Conditions:	The user must have an active internet connection to access Firebase and save data.
Assumptions:	• The user has basic knowledge of how to operate a timer function on a mobile application. • The user has an internet connection to access the Firebase database. • The user is using a device that is compatible with the application.
Notes and Issues:	The Firebase database may have delays or errors in saving data due to the network or server issues.

2.1.1.4. Requirement 3: Analytics Functionality

This requirement allows the user to track their focus session activities within the application.

Description & Priority

Requirement 2: Study Functionality

This requirement is crucial for the Analytics functionality in the Focus Timer feature of the application. It allows the user to track their focus sessions' activity and progress. The user can select a specific date on the calendar view, and the data retrieved from the Firebase database based on the user UID will be displayed for that day.

This feature is of medium priority since other functionalities can run without it, but it is an essential feature as it enables users to monitor and evaluate their progress. Thus, it needs to be implemented with medium priority.

Use Case

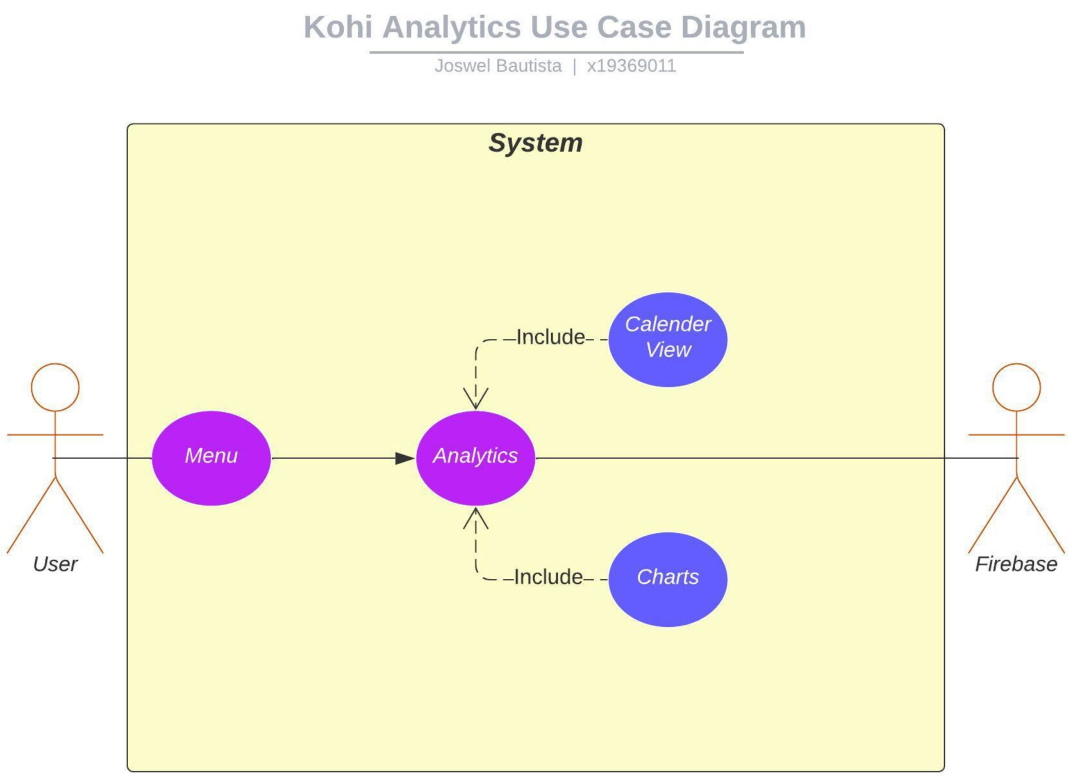
Scope

The scope of this use case is to showcase the Analytics functionality of the application. This includes the user's data from the Focus Timer feature, which allows the user to track their focus sessions activity.

Description

This use case describes how the user interacts with the Analytics functionality in the application. The user is provided with data that is retrieved from the Firebase database to show their activity on a selected date which include using the calendar view. The data being displayed includes charts, where the user is presented with a bar chart of their activity on the selected date. This selection of dates works by doing a query, where it would match the UID and order it by the timestamp in the Firebase database. This feature is essential to the overall system, as it allows the user to keep track of their progress and improve their productivit*y.*

Focus Timer Use Case Diagram UC-v1.3



Use Case Flow Description:

Use Case ID:	UC-v1.3
Use Case Name:	Focus Timer
Actors: Primary Secondary	Primary Actor is the User who enters the System. Secondary Actor is Firebase Database

Activation:	The Analytics functionality is activated when the user selects the Analytics option in the main menu of the application.
Pre-conditions:	• The user must be logged into the application, and there must be data available in the firebase database for the selected date on the calendar view.
Post-Conditions:	• The user is presented with the Analytics screen showing their activity on the selected date, including the bar chart displaying their focus session activity
Main Flow:	1. The user selects the Analytics option from the main menu. 2. The application retrieves the user's data from the Firebase database based on their UID. 3. The user is presented with a calendar view and is able to select a date. 4. The application queries the database to retrieve the user's activity data for the selected date. 5. The user is presented with a bar chart displaying their activity for the selected date. 6. The user can select a different date or return to the main menu.
Alternate Flows:	1. If there is no data available for the selected date, the application displays an empty graph to the user indicating that there is no data available for that date. 2. If there is data available for the selected date, the application displays the user's activity in a chart format.
Exceptional Flows:	• If there is a problem retrieving data from the Firebase database due to a connection error or other technical issue, an error message will be displayed to the user informing them of the problem. • If the user's account is deleted or the Firebase database is reset, all user data including activity on the Focus Timer will be lost and the Analytics functionality will not work as intended. An error message will be displayed to the user informing them of the issue.
Termination	The user exits the Analytics functionality and is redirected back to the main menu.
Includes:	Includes calendar view and charts
Extends:	None
Priority:	Medium
Special Conditions:	• The user must be logged into the application to access the Analytics functionality.

	• The user must have data stored in the firebase database for the Analytics functionality to display. • The data being displayed in the Analytics functionality is based on the selected date from the calendar view.
Assumptions:	• The user has an active internet connection to retrieve data from the firebase database. • The user understands how to use the calendar view to select a date.
Notes and Issues:	The Firebase database may have delays or errors in saving data due to the network or server issues.

2.1.1.5. Requirement 4: Gacha Functionality

This requirement is a part of the application's core feature. The Gacha functionality is implemented to reward the user for their completed focus sessions.

Description & Priority

Requirement 4: Gacha Functionality

The Gacha functionality is an essential component of the application where users are rewarded for completing their study sessions. This feature allows users to interact with the reward system of the application making it a core feature. Upon completing a study session, users are awarded points which can be used to earn different virtual rewards such as additional focus time backgrounds. This requirement is important because it provides a way for users to be rewarded for completing focus sessions, motivating them to use the application more often. It also provides a fun and interactive way for users to obtain different app designs, which adds to the overall user experience.

The Gacha functionality is a high priority requirement since it is a key feature of the application that directly impacts user engagement and motivation. Without this functionality users may lose interest in the application and may not be incentivized to complete their study sessions. Therefore, it’s essential to have this feature in place to ensure users are motivated to continue using the application.

Use Case

Scope

The scope of this use case is to demonstrate the Gacha functionality of the application including the user's ability to receive rewards for completing focus sessions and using those rewards to obtain different app designs.

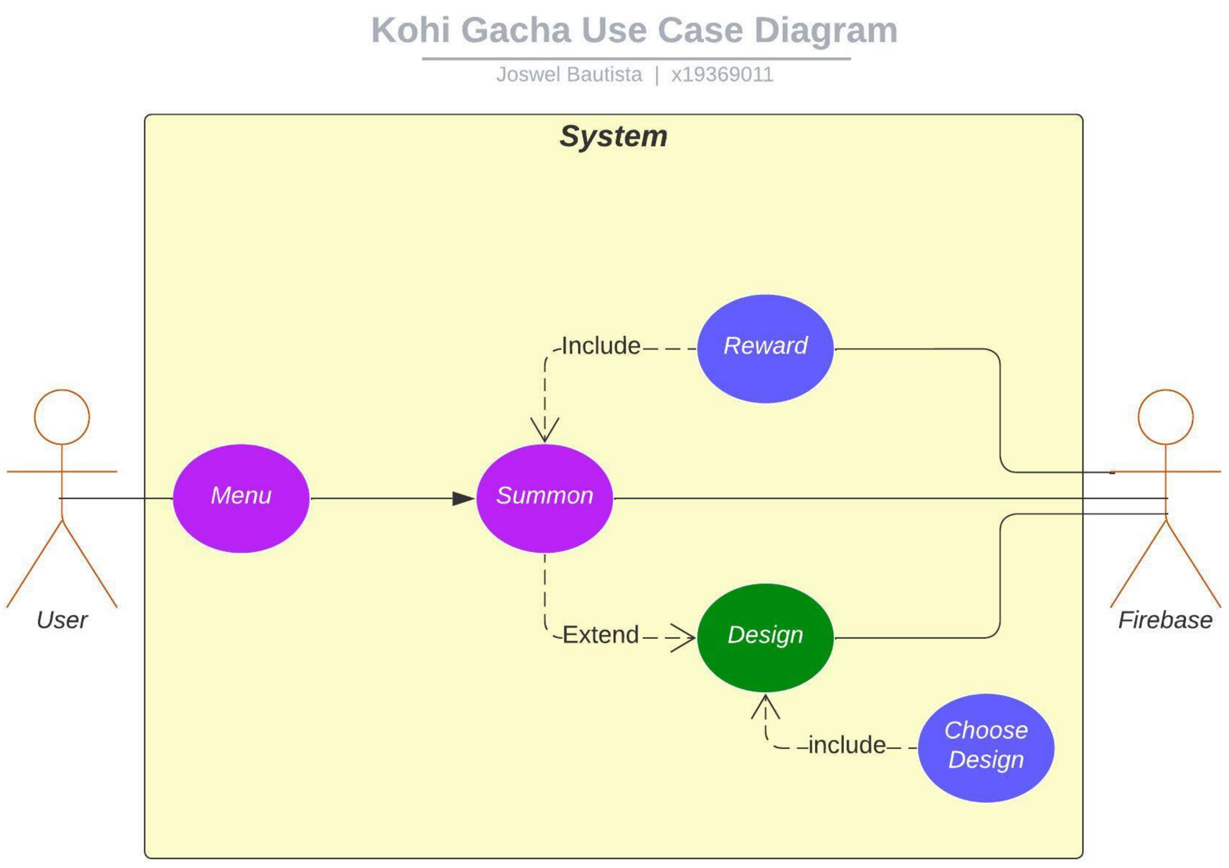
Description

This use case shows the user's interaction with the Gacha activity in the application. The user clicks on the Gacha button from the main menu, which brings up the summon screen. The user can then summon for a random reward by clicking the "Brew" button. The Gacha system is designed with a 30% success rate, meaning the user has a 30% chance of receiving a random reward from the list of rewards.

If the user receives a reward, the reward is added to the existing rewards list in Firebase to prevent the user from receiving the same reward twice. However, the user's points are deducted by 1 for every summon, regardless of whether they receive a reward or not.

The rewards obtained through the Gacha system can be used to obtain different app designs. The user can choose a design from the app design screen, which is accessed by returning to the main menu. The chosen design is saved to Firebase so that it can be retrieved and stored for future use when the application is reopened.

Gacha Use Case Diagram UC-v1.4



Use Case Flow Description:

Use Case ID:	UC-v1.4
Use Case Name:	Gacha
Actors: Primary	Primary Actor is the User who enters the System. Secondary Actor is Firebase Database

Secondary	
Activation:	The user clicks on the Gacha activity from the main menu to activate this use case.
Pre-conditions:	• The user is logged into the application. • The user has completed at least one study session to earn points.
Post-Conditions:	• The user receives a random reward from the Gacha summon. • The user's points are updated and reduced by one for every summon. • The user's reward is added to the existing rewards in Firebase, preventing the user from receiving the same reward twice. • The user can access their rewards and select a design for their application. • The selected design is saved to Firebase and can be retrieved and stored for later use.
Main Flow:	1. User clicks on the Gacha activity from the main menu. 2. User clicks on the "Brew" button to summon a random reward. 3. Application checks user's points and takes 1 point away for every summon. 4. Application generates a random reward from the list of available rewards. 5. If the user is lucky enough to receive a reward, the application updates the user's points and saves the reward to Firebase. 6. If the user is unlucky and doesn't receive a reward, the application simply updates the user's points. 7. User then redirects back to the main menu to use their rewards. 8. User clicks on the Design Activity 9. User can choose a design for their application by selecting a design from the available options. 10. The selected design is saved to Firebase so it can be retrieved and stored later when the application is re-opened.
Alternate Flows:	1. If the user doesn't have enough points, they will be prompted a message “no points”. 2. If there are no more rewards, the system displays a message stating "No rewards left." 3. If there is an error saving the design state to Firebase, the user will receive an error message.
Exceptional Flows:	• If the user clicks on a received reward from the design activity, a message will be displayed indicating that the

	reward has already been received and the user will not be directed to the summon activity. • If there is a connection issue to the firebase database, an error message will be displayed.
Termination	The user successfully receives a reward from the Gacha functionality or if there are no more rewards left. The user can then proceed to use the reward in the Design activity or go back to the main menu.
Includes:	Includes reward and choose design
Extends:	Design
Priority:	High
Special Conditions:	• The Gacha functionality is dependent on the user having completed focus sessions to earn points. • The rewards system is based on chance, with a 30% probability of receiving a reward on each summon. • The rewards list is limited and once all rewards have been received, the user will receive a message that no more rewards are available.
Assumptions:	• The user has a stable internet connection for the application to access the firebase database and update rewards data. • The user is aware of the Gacha system and its probability of reward and understands that it is a game of chance.
Notes and Issues:	The Firebase database may have delays or errors in saving data due to the network or server issues.

2.1.1.6. Requirement 5: Diary Functionality

This requirement is for the Diary functionality which serves as a tool for the user to create and manage their tasks. The Diary feature allows for the user's productivity and organization within the application.

Description & Priority

Requirement 5: Diary Functionality - The Diary functionality is an essential requirement of the application that allows the user to create and view tasks. The user can create a task by entering the task name, task details and due date and time. The user can view their tasks in a list view where they can view, update, or delete it.

The Diary functionality is essential for the user's productivity and organization within the application, and its priority is high. Without this feature, the user will not be able to keep track of their tasks and deadlines leading to decreased productivity and frustration. The Diary functionality enhances the user experience by providing them with a centralized location for their tasks which can be accessed at any time. This feature also allows the user to plan and prioritize their tasks effectively, which is crucial for achieving their goals.

Use Case

Scope

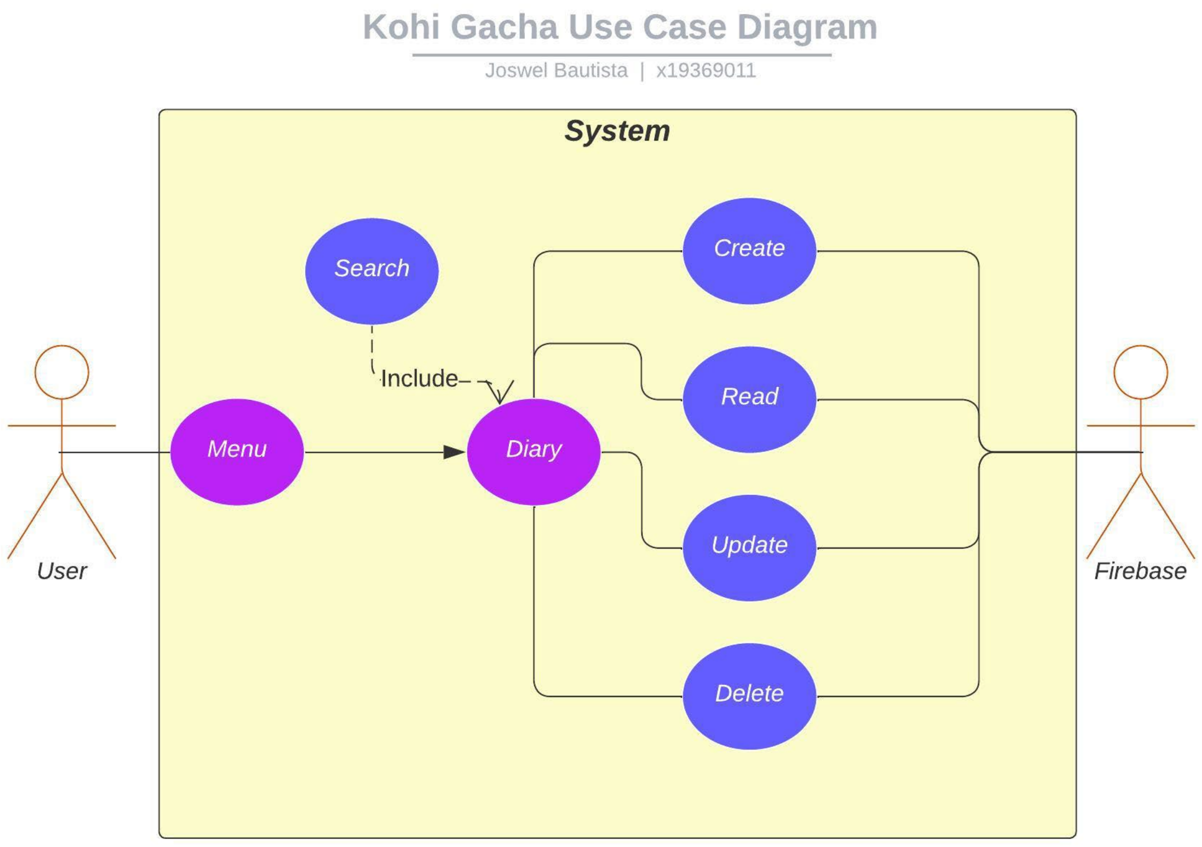
The scope of this use case is to demonstrate the functionality of the Diary feature in the application. This includes creating, viewing, updating, and deleting tasks that the user has assigned to themselves.

Description

The Diary feature in this use case how the the user can be more organized and productive. The user can access the Diary feature by clicking on the Diary button from the main menu which will direct them to the diary activity. In the diary the user can create tasks by providing a title and context to describe the task. This feature helps the user to organize their work and keep track of their progress. The Diary feature also has a view, update, and delete functionality, allowing the user to manage their tasks as they see fit. The user can view their existing tasks, edit them if necessary, and delete them if they are no longer needed. All data entered by the user is saved and retrieved from the firebase database.

Additionally, the Diary feature includes a search functionality, which enables the user to search for a specific task by title or context. This feature is particularly useful if the user has many tasks and needs to find a specific one quickly.

Diary Use Case Diagram UC-v1.4



Use Case Flow Description:

Use Case ID:	UC-v1.5
Use Case Name:	Diary
Actors: Primary Secondary	Primary Actor is the User who enters the System. Secondary Actor is Firebase Database
Activation:	The user clicks on the Diary button from the main menu, which will direct them to the Diary activity.
Pre-conditions:	• The user is logged into the application. • The Diary feature is accessible from the main menu. • The user has tasks to create, view, update, or delete.
Post-Conditions:	• The user has created, viewed, updated, or deleted a task in the Diary feature. • The user can access their tasks at any time from the Diary feature. • Any changes made by the user are saved and retrieved from the firebase database.
Main Flow:	1. User clicks on the Diary button from the main menu. 2. User is directed to the Diary activity where they can create, view, update, or delete tasks.

	3. To create a task, user clicks on the file icon on the top right and enters the task name and details. 4. The task is saved to the firebase database. 5. To view tasks, user clicks on the task and the list of all tasks is displayed. 6. User can select a task from the list to view the details. 7. To update a task the user selects the task and clicks on the task to make the necessary changes and save it. 8. To delete a task, user selects the task and clicks on the Bin icon button. 9. The task is deleted from the database. 10. User can exit the Diary activity and return to the main menu.
Alternate Flows:	1. User searches for a specific task by entering the task name or context in the search bar. 2. The search function filters the tasks to display only those that match the search criteria. 3. User selects the desired task from the search results and proceeds with viewing, updating, or deleting the task as required. 4. User can exit the Diary activity and return to the main menu.
Exceptional Flows:	• If there is an error connecting to the firebase database, the user will be prompted of an error. • If the user attempts to update or delete a task that does not exist an error message will be prompted.
Termination	The Diary activity will be closed when the user clicks the Menu button or exits the application.
Includes:	Includes reward and choose design
Extends:	None
Priority:	Search
Special Conditions:	• The Diary functionality requires an active internet connection to save and retrieve data from the Firebase database. • The user must have a valid account to use the Diary functionality.
Assumptions:	• The user has basic knowledge of how to use a mobile application. • The user is familiar with creating, reading, updating, and deleting tasks. • The user has an active internet connection to use the Diary functionality.

Notes and Issues:	• The Firebase database may have delays or errors in saving data due to the network or server issues. • The performance of the Diary feature should be optimized to ensure that it can handle large amounts of data without slowing down the application. • It's important to provide proper error handling and error messages to the user in case any issues or errors occur while using the Diary feature.

2.1.2. Data Requirements

The data requirements for the application include the following:

User login credentials: The application shall store and retrieve user login credentials such as email and password.

Security: The application will ensure the security of the users data by using secure authentication protocols and encrypting sensitive data stored in the Firebase database. The application will also comply with relevant data privacy regulations such as GDPR and CCPA.

User profile information: The application shall store and retrieve user profile information such as name. The application shall also use Firebase authentication to verify user credentials during login and registration.

Firebase Query: The application will use Firebase queries to retrieve user data based on the unique user ID assigned to each user during registration or login.

Focus timer data: The application shall store data on user focus timer sessions, including session length, start and end times, and rewards earned.

Diary data: The application shall store data on user diary entries, including the title, content, date, and time.

Analytics data: The application shall store data on user analytics, including the number of focus sessions completed, the total time spent on the focus timer.

Summon data: The application shall store data on user rewards and design choices, including the type of reward redeemed and the design elements chosen, and any rewards earned.

2.1.3. User Requirements

The user requirements for the application include the following:

User registration and login: The application should allow users to register for a new account or log in to an existing account or login as a guest.

Focus timer: The application should provide a focus timer feature that allows users to set a session length and earn rewards for completing successful focus sessions.

Diary: The application should provide a diary feature that allows users to create, read, update, and delete diary entries.

Analytics: The application should provide analytics feature that displays user data on focus sessions completed and rewards earned.

Summon: The application should provide a summon feature that allows users to redeem rewards and design their app based on those rewards.

Application settings: The application should provide users with settings option to sign out.

2.1.4. Environmental Requirements

The environmental requirements for the application include the following:

Mobile devices: The application should be compatible with mobile devices running iOS or Android operating systems.

Internet connectivity: The application requires a stable internet connection to ensure proper functioning and data retrieval.

Battery life: The application should be optimized to minimize battery usage to ensure users can use the application for extended periods without recharging.

Screen size and resolution: The application should be optimized for different screen sizes and resolutions to ensure a consistent user experience across different devices.

2.1.5. Usability Requirements

The usability requirements for the application include the following:

User-friendly interface: The application should have a user-friendly interface that is easy to navigate.

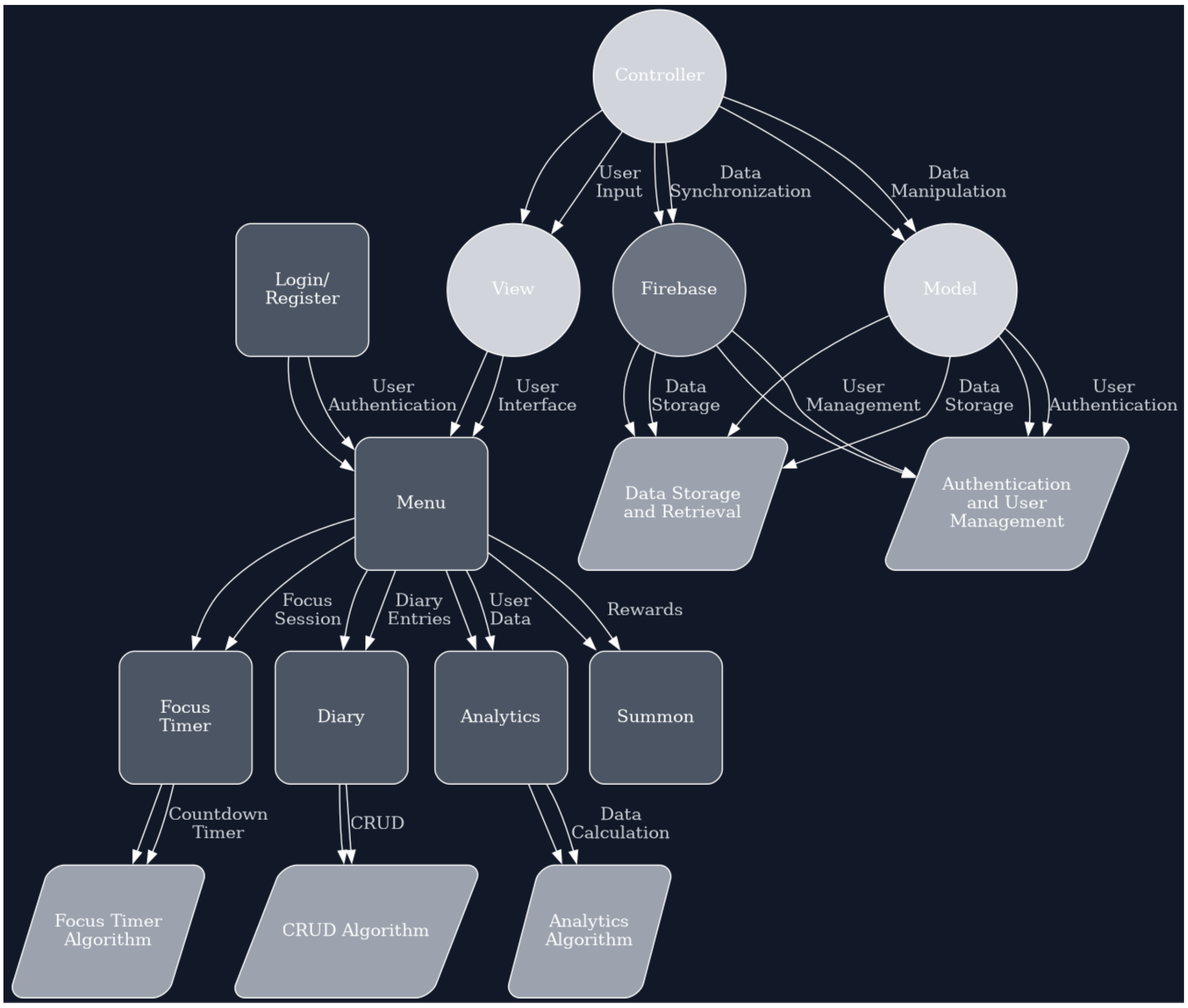
Intuitive design: The application should have an intuitive design that is easy to understand and use.

Performance: The application should have optimal performance, including fast loading times and smooth transitions between pages.

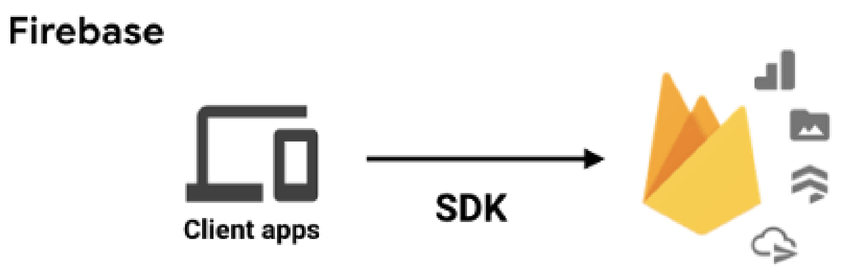
Error handling: The application should handle errors gracefully by providing clear error messages and options for recovery.

2.2. Design & Architecture

Architecture of the application AC-v1.1



The mobile application follows a client-server architecture, where the client runs on the user's mobile device and communicates with the server through Firebase, which provides cloud-based storage and authentication services.



(DEVOPEDIA®, n.d.)

In this architecture, the client (mobile app) communicates with the server (Firebase backend) to retrieve data and perform certain actions. The client sends requests to the server, which processes them and returns the requested information.

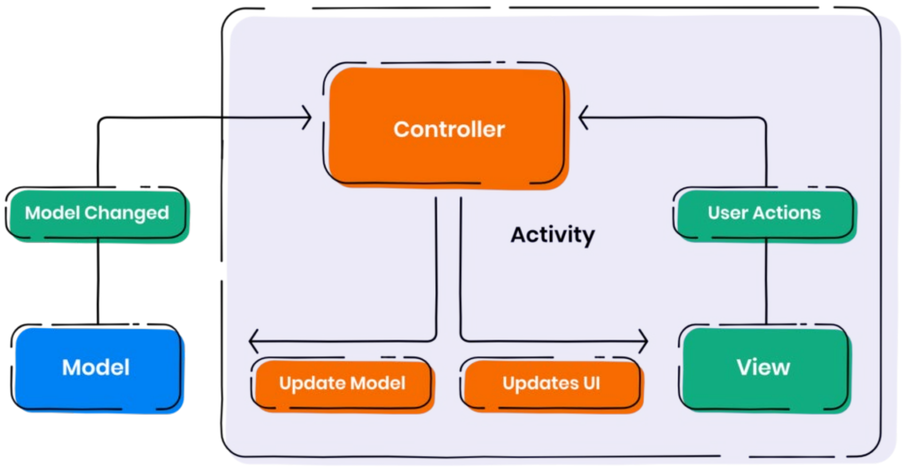
For example, when a user logs in or registers, the client sends a request to the server with the user's credentials. The server then authenticates the user and sends a response indicating whether the login or registration was successful.

Similarly, when a user creates, reads, updates, or deletes diary entries, the client sends a request to the server with the relevant data. The server processes the request and updates the database accordingly. When the user wants to view their diary entries, the client sends a request to the server, which retrieves the relevant data from the database and sends it back to the client.

This client-server architecture allows for efficient management of data and provides users with a seamless experience as they do not need to worry about managing the data themselves. Additionally, the use of a cloud-based database (Firebase) allows for scalability and accessibility since users can access their data from any device with the app installed.

The main components of the application include Login/Register, Menu, Focus Timer, Diary, Analytics, and Summon. Each of these components plays a crucial role in ensuring that the mobile application is functional and user-friendly. The Login/Register component enables users to create an account or log in with an existing account. The Menu component serves as a central hub for accessing the other components of the application. The Focus Timer component enables users to set a session length and focus on a task, while the Diary component allows users to create, read, update, and delete tasks. The Analytics component displays user data on how long they used the Focus Timer for. The Summon component enables users to redeem points and customize the application based on rewards.

MVC Design MVC-v1.0



The mobile application follows a model-view-controller (MVC) architectural pattern. The model represents the data and logic of the application and includes classes for handling user authentication, storing and retrieving user data, and managing rewards and analytics. The view represents the user interface and includes classes for displaying the various components of the application. The controller acts as an intermediary between the model and view and includes classes for handling user input and updating the model and view accordingly.

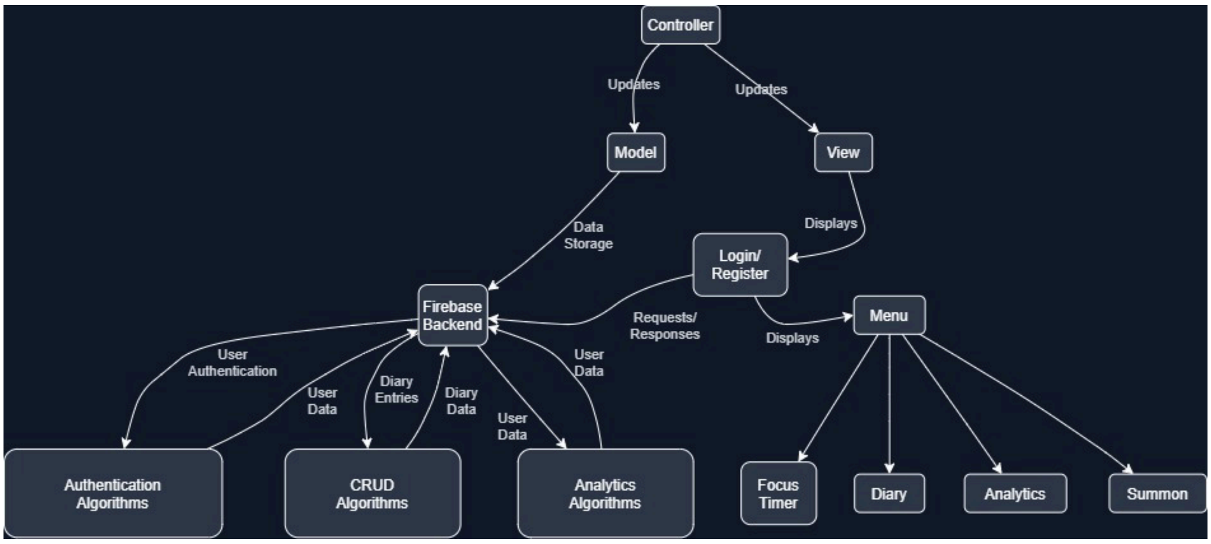
The main algorithms used in the project include authentication algorithms for handling user registration and login, CRUD algorithms for creating, reading, updating, and deleting diary entries, and analytics algorithms for calculating and displaying user data.

Authentication algorithms use Firebase authentication services to handle user registration and login. When a user registers the algorithm checks if the email is not in use and if it isn’t in use, it creates a new account and saves the user data to the Firebase database. When a user logs in the algorithm retrieves the user data from the Firebase database and saves it locally on the device for session persistence.

CRUD algorithms use Firebase's cloud-based storage to create, read, update, and delete diary entries. When a user creates a new diary entry, the algorithm saves the data to the Firebase database using the user's UID as the key. When a user reads a diary entry the algorithm retrieves the data from the Firebase database using the same key. When a user updates or deletes a diary entry the algorithm modifies or removes the data from the Firebase database accordingly.

Analytics algorithms use Firebase's cloud-based storage to calculate and display user data. When a user completes a focus session the algorithm saves the data to the Firebase database using the user's UID as the key. The algorithm then retrieves the data from the Firebase database and calculates the total time spent on focus sessions. The results are then displayed to the user in the Analytics component of the application.

Design of the application DA-v1.1



This is an architecture diagram for my mobile application which was created to visualize the various components and their interactions. The diagram can include the client-server architecture, the MVC pattern, and the algorithms used in the project. This was drawn using Draw.io.

2.3. Implementation

The main algorithms used in the application are authentication algorithms for handling user registration and login, CRUD algorithms for creating, reading, updating, and deleting diary entries, analytics algorithms for calculating and displaying user data and the Gacha algorithm where it selects a random reward from the rewards list.

The main classes used for the main features are the Login/Register, Focus Timer, Diary, Analytics, and Summon classes. The Login/Register class allows users to create an account or log in with an existing account. The Login/Register class enables users to create an account or log in with an existing account. The Focus Timer class allows users to start and end a session. The Diary class allows users to create, read, update, and delete notes. The Analytics class displays the user data on how long they used the Focus Timer for. The Summon class enables users to redeem points and customize the application based on rewards.

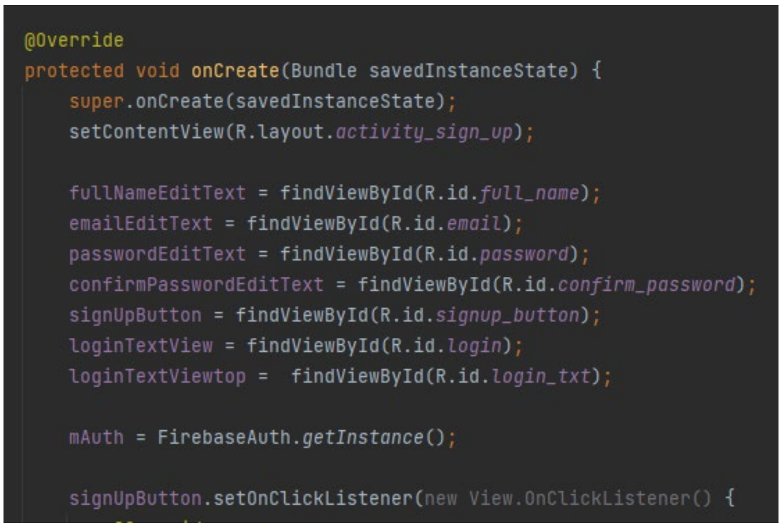
In addition to these primary classes the application features several key classes that work in t such as Settings, Menu, and Splash, which are responsible for managing the user interface and various ancillary functions. The Menu class is responsible for displaying the Hub menu of the application which contains all of the components. The Settings class provides users with the ability to sign out of the application while the Splash class serves as the introduction page displaying splash art with engaging animations.

The main functions and methods used in the application include handling user authentication, storing and retrieving user data, managing rewards and analytics, and updating the user interface based on user input. The functions are divided into different classes based on their functionality and follow the model-view-controller (MVC) architectural pattern. The model represents the data and logic of the application and includes functions and methods for handling the user authentication, storing, and retrieving user data and managing rewards and analytics. The view represents the user interface and includes functions for displaying various components of the application. The controller acts as an intermediary between the model and view and includes methods and functions for handling user input and updating the model and view accordingly.

Authentication Implementation

The authentication algorithms use Firebase authentication services to handle user registration and login. The user's information is stored securely in a database and password hashing is used to ensure the user's information is not easily accessible. The Login/Register class is responsible for handling the user's registration and login requests. When a user logs in the system checks if the entered credentials match with the stored information and if they do the user is granted access to the application. The algorithm then retrieves the user data from the Firebase database and saves it locally on the device for session persistence. When a user registers the algorithm checks if the username is unique and if it is it creates a new account and saves the user data to the Firebase database. (Firebase, n.d.)

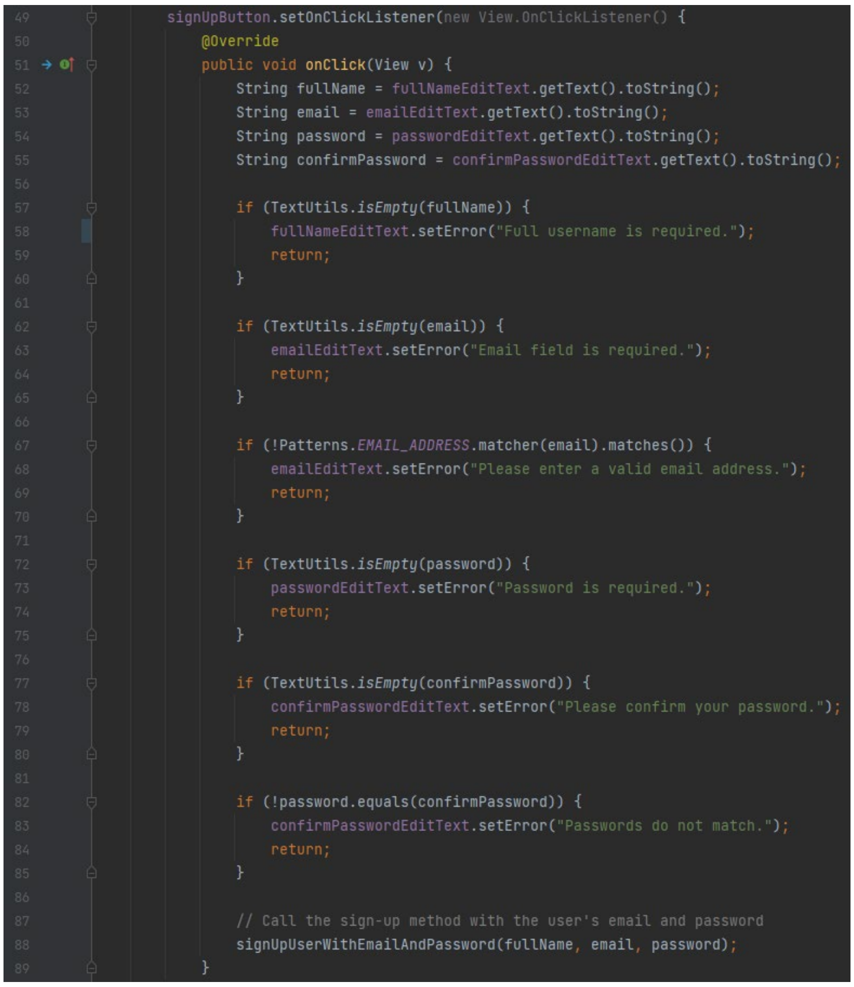
The SignUpActivity class is responsible for handling the sign-up process for new users. It contains EditText fields for the users full name, email address, password, and confirm password, a sign-up button, and TextViews for the login link. It also initializes the FirebaseAuth object to communicate with Firebase Authentication and create a new user account.



```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_sign_up);

    fullNameEditText = findViewById(R.id.full_name);
    emailEditText = findViewById(R.id.email);
    passwordEditText = findViewById(R.id.password);
    confirmPasswordEditText = findViewById(R.id.confirm_password);
    signUpButton = findViewById(R.id.signup_button);
    loginTextView = findViewById(R.id.login);
    loginTextViewtop =  findViewById(R.id.login_txt);

    mAuth = FirebaseAuth.getInstance();

    signUpButton.setOnClickListener(new View.OnClickListener() {
```

Figure c1.0

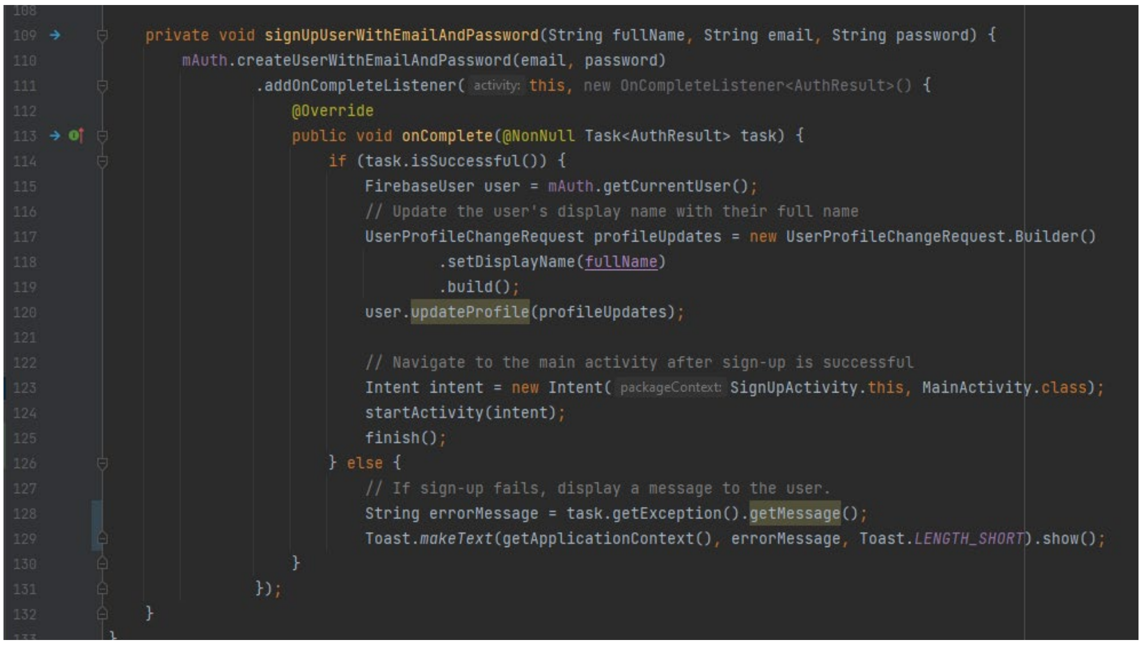
The onCreate() method is called when the activity is created. This initializes the UI elements such as EditTexts, Buttons, and TextViews using findViewById(). It also initializes the FirebaseAuth instance using getInstance(). The onClickListener on the signUpButton to handle the sign-up process. It sets onClickListeners on the loginTextView and loginTextViewtop to navigate to the LoginActivity.



```java
        signUpButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String fullName = fullNameEditText.getText().toString();
                String email = emailEditText.getText().toString();
                String password = passwordEditText.getText().toString();
                String confirmPassword = confirmPasswordEditText.getText().toString();

                if (TextUtils.isEmpty(fullName)) {
                    fullNameEditText.setError("Full username is required.");
                    return;
                }

                if (TextUtils.isEmpty(email)) {
                    emailEditText.setError("Email field is required.");
                    return;
                }

                if (!Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
                    emailEditText.setError("Please enter a valid email address.");
                    return;
                }

                if (TextUtils.isEmpty(password)) {
                    passwordEditText.setError("Password is required.");
                    return;
                }

                if (TextUtils.isEmpty(confirmPassword)) {
                    confirmPasswordEditText.setError("Please confirm your password.");
                    return;
                }

                if (!password.equals(confirmPassword)) {
                    confirmPasswordEditText.setError("Passwords do not match.");
                    return;
                }

                // Call the sign-up method with the user's email and password
                signUpUserWithEmailAndPassword(fullName, email, password);
            }
```

Figure c1.1

The signUpButton onClickListener handles the sign-up process. It first gets the users input from the EditTexts. It then validates the input by checking if the fullName, email, password, and confirmPassword fields are not empty and checking if the email is valid using Patterns.EMAIL_ADDRESS.matcher(). If any of the fields are invalid, it will set an error message on the corresponding EditText and returns it. If the input is valid it then calls the signUpUserWithEmailAndPassword() method with the users fullName, email, and password.



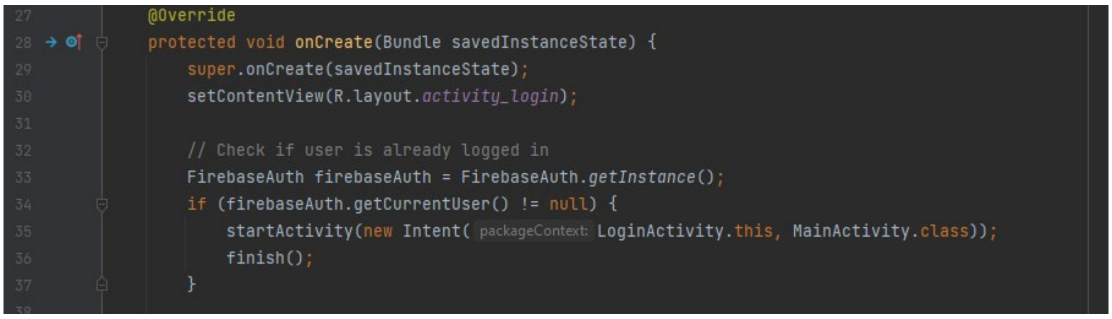
```java
    private void signUpUserWithEmailAndPassword(String fullName, String email, String password) {
        mAuth.createUserWithEmailAndPassword(email, password)
                .addOnCompleteListener( activity: this, new OnCompleteListener<AuthResult>() {
                    @Override
                    public void onComplete(@NonNull Task<AuthResult> task) {
                        if (task.isSuccessful()) {
                            FirebaseUser user = mAuth.getCurrentUser();
                            // Update the user's display name with their full name
                            UserProfileChangeRequest profileUpdates = new UserProfileChangeRequest.Builder()
                                    .setDisplayName(fullName)
                                    .build();
                            user.updateProfile(profileUpdates);

                            // Navigate to the main activity after sign-up is successful
                            Intent intent = new Intent( packageContext: SignUpActivity.this, MainActivity.class);
                            startActivity(intent);
                            finish();
                        } else {
                            // If sign-up fails, display a message to the user.
                            String errorMessage = task.getException().getMessage();
                            Toast.makeText(getApplicationContext(), errorMessage, Toast.LENGTH_SHORT).show();
                        }
                    }
                });
    }
```

Figure c1.2

When a user registers in the application the authentication algorithm checks if the email is not in use. If the email is not used the algorithm it creates a new account for the user using the createUserWithEmailAndPassword() method provided by Firebase authentication services. The signUpUserWithEmailAndPassword() method then calls the createUserWithEmailAndPassword() method of the FirebaseAuth instance with the users email and password. This method takes the users email and password as input and creates a new user account in Firebase authentication services. Once the account is created the user’s data is saved to the Firebase database.

The addOnCompleteListener then handle the result of the sign-up process. If the sign-up is successful it gets the current user using getCurrentUser() and updates the users display name with their fullName using UserProfileChangeRequest.Builder(). It then navigates the user to the MainActivity using an Intent. If the sign-up fails, it displays an error message to the user using Toast.makeText().

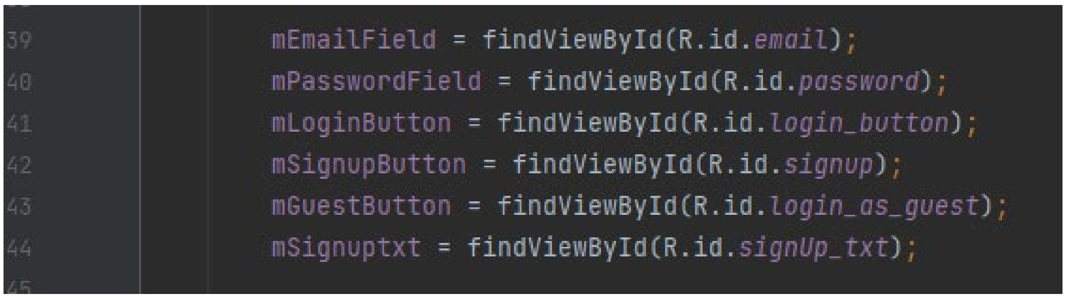
The LoginActivity class then is responsible for implementing the login functionality in the app using Firebase Authentication. It provides UI elements such as EditTexts, Buttons, and TextViews to allow the user to input their email and password. It validates the users input and handles the login process using the FirebaseAuth instance. I have also added a ProgressDialog to provide feedback to the user that the login process is in progress. If the user is already logged in it will navigate them instead to the MainActivity.



```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_login);

    // Check if user is already logged in
    FirebaseAuth firebaseAuth = FirebaseAuth.getInstance();
    if (firebaseAuth.getCurrentUser() != null) {
        startActivity(new Intent( packageContext: LoginActivity.this, MainActivity.class));
        finish();
    }
```

Figure c1.3

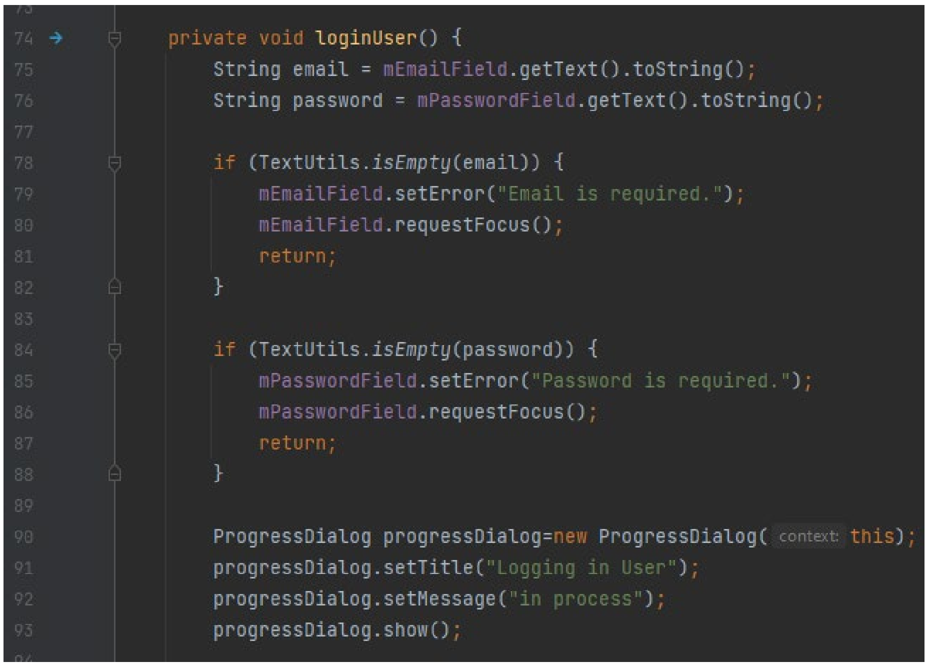
In the onCreate() method It first checks if the user is already logged in using FirebaseAuth.getInstance().getCurrentUser(). If the user is already logged in, it navigates to the MainActivity using an Intent and finishes the LoginActivity.



```java
        mEmailField = findViewById(R.id.email);
        mPasswordField = findViewById(R.id.password);
        mLoginButton = findViewById(R.id.login_button);
        mSignupButton = findViewById(R.id.signup);
        mGuestButton = findViewById(R.id.login_as_guest);
        mSignuptxt = findViewById(R.id.signUp_txt);
```

Figure c1.4

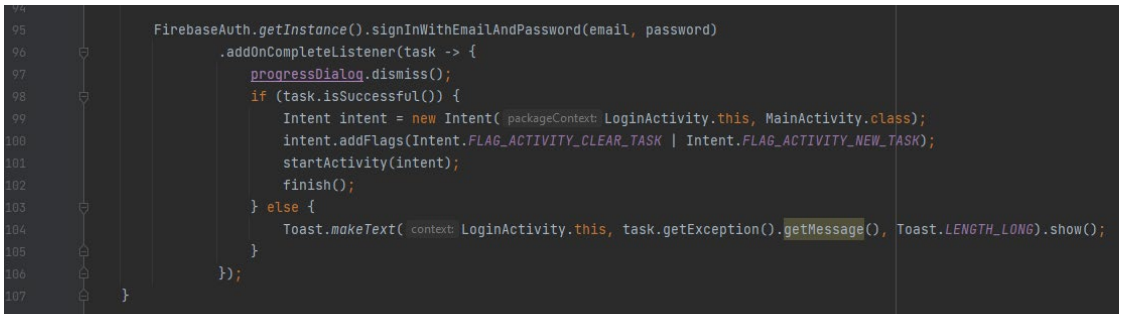
I then initialize the UI elements such as EditTexts, Buttons, and TextViews using findViewById(). It sets onClickListeners on the mLoginButton, mSignupButton, mGuestButton, and mSignuptxt to handle the login, sign-up, login as guest, and navigate to the SignUpActivity processes, respectively.



```java
private void loginUser() {
    String email = mEmailField.getText().toString();
    String password = mPasswordField.getText().toString();

    if (TextUtils.isEmpty(email)) {
        mEmailField.setError("Email is required.");
        mEmailField.requestFocus();
        return;
    }

    if (TextUtils.isEmpty(password)) {
        mPasswordField.setError("Password is required.");
        mPasswordField.requestFocus();
        return;
    }

    ProgressDialog progressDialog=new ProgressDialog( context: this);
    progressDialog.setTitle("Logging in User");
    progressDialog.setMessage("in process");
    progressDialog.show();
```

Figure c1.5

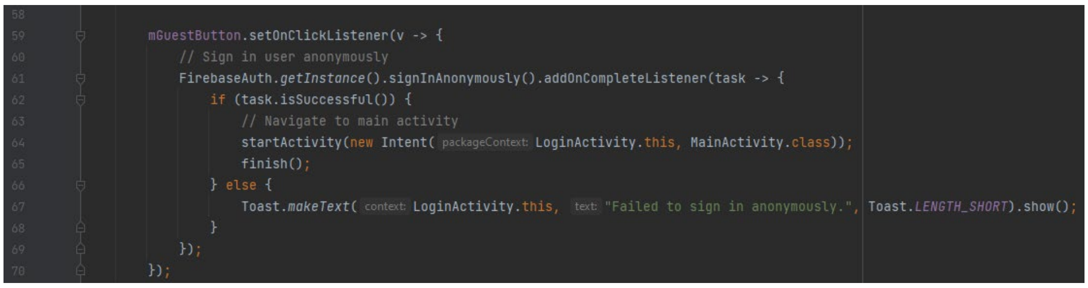
The loginUser() method then handles the login process of the user. The method first gets the users input from the EditTexts. It then validates the input by checking if the email and password fields are not empty. If any of the fields are empty, it sets an error message on the corresponding EditText and return it. If the input is valid it shows a ProgressDialog to indicate that the login process is in progress.



```
        FirebaseAuth.getInstance().signInWithEmailAndPassword(email, password)
                .addOnCompleteListener(task -> {
                    progressDialog.dismiss();
                    if (task.isSuccessful()) {
                        Intent intent = new Intent( packageContext: LoginActivity.this, MainActivity.class);
                        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK | Intent.FLAG_ACTIVITY_NEW_TASK);
                        startActivity(intent);
                        finish();
                    } else {
                        Toast.makeText( context: LoginActivity.this, task.getException().getMessage(), Toast.LENGTH_LONG).show();
                    }
                });
    }
```

Figure c1.6

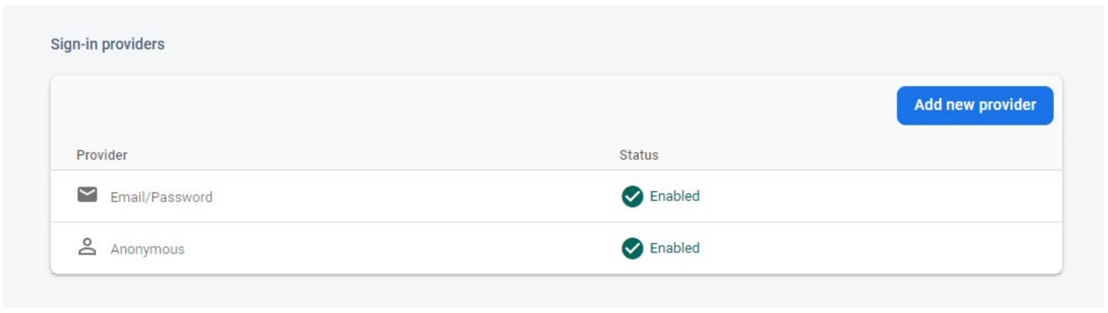
When a user logs in to the application the authentication algorithm retrieves the users data from the Firebase database using the signInWithEmailAndPassword() method provided by Firebase authentication services. Here I then call the signInWithEmailAndPassword() method of the FirebaseAuth instance with the users email and password. It adds an onCompleteListener to handle the result of the login process. If the login is successful, it creates an Intent to navigate to the MainActivity and clears the task stack using addFlags(). The Intent flag FLAG_ACTIVITY_CLEAR_TASK is used to clear the activity stack when the user logs in. This is done to prevent the user from going back to the login activity once they have successfully logged in. FLAG_ACTIVITY_NEW_TASK flag is used to create a new task and add the activity to it, so it becomes the start of the new task. This flag is used here to ensure that the MainActivity is created in a new task to separate from the login activity. This then starts the activity using startActivity() and finishes the LoginActivity. If the login fails it will display an error message to the user using Toast.makeText().



```
        mGuestButton.setOnClickListener(v -> {
            // Sign in user anonymously
            FirebaseAuth.getInstance().signInAnonymously().addOnCompleteListener(task -> {
                if (task.isSuccessful()) {
                    // Navigate to main activity
                    startActivity(new Intent( packageContext: LoginActivity.this, MainActivity.class));
                    finish();
                } else {
                    Toast.makeText( context: LoginActivity.this,  text: "Failed to sign in anonymously.", Toast.LENGTH_SHORT).show();
                }
            });
        });
```

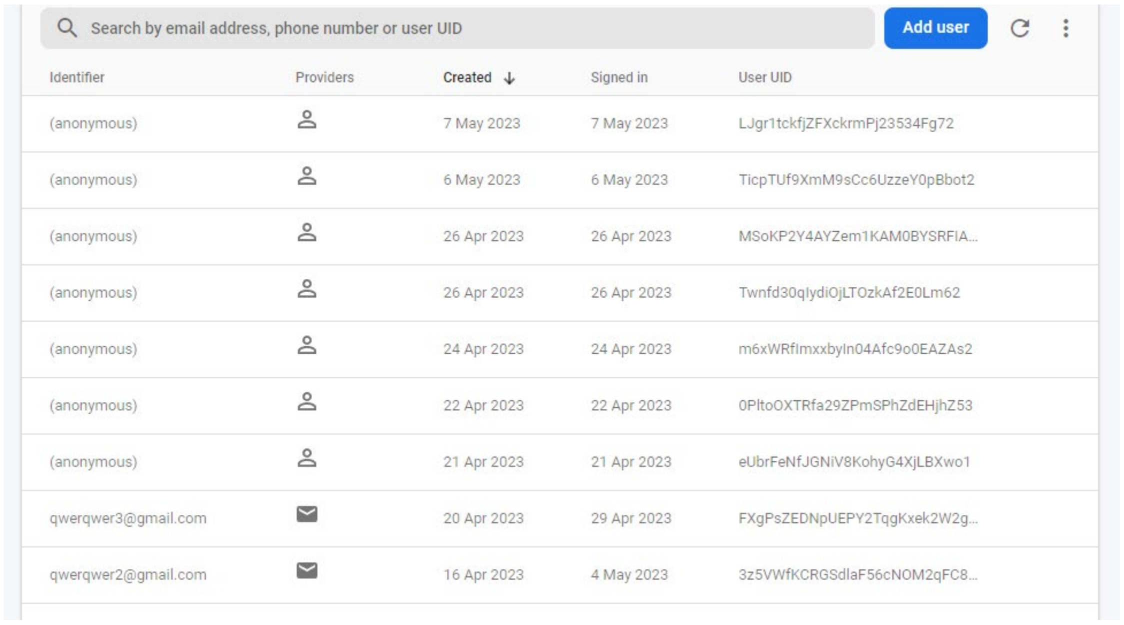
Figure c1.7

The LoginActivity class also includes a Continue as Guest button that allows the user to log in anonymously using the signInAnonymously() method provided by Firebase authentication services. This method creates an anonymous user account in Firebase authentication services and authenticates the user with the anonymous account. This allows the user to use the application without creating a permanent user account.


Sign-in providers

Provider	Status
Email/Password	✓ Enabled
Anonymous	✓ Enabled

Figure c1.8



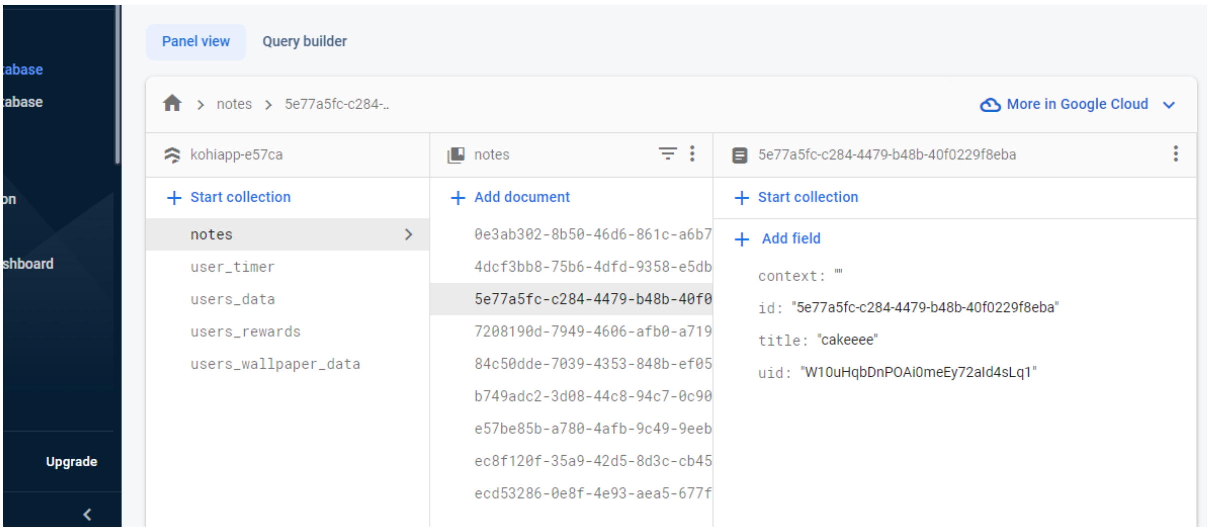
Identifier	Providers	Created ↓	Signed in	User UID
(anonymous)	👤	7 May 2023	7 May 2023	LJgr1tckfjZFXckrmPj23534Fg72
(anonymous)	👤	6 May 2023	6 May 2023	TicpTUf9XmM9sCc6UzzeY0pBbot2
(anonymous)	👤	26 Apr 2023	26 Apr 2023	MSoKP2Y4AYZem1KAM0BYSRFIA...
(anonymous)	👤	26 Apr 2023	26 Apr 2023	Twnfd30qIydiOjLTOzkAf2E0Lm62
(anonymous)	👤	24 Apr 2023	24 Apr 2023	m6xWRfImxxbyIn04Afc9o0EAZAs2
(anonymous)	👤	22 Apr 2023	22 Apr 2023	0PltoOXTRfa29ZPmSPhZdEHjhZ53
(anonymous)	👤	21 Apr 2023	21 Apr 2023	eUbrFeNfJGNiV8KohyG4XjLBXwo1
qwerqwer3@gmail.com	✉	20 Apr 2023	29 Apr 2023	FXgPsZEDNpUEPY2TqgKxek2W2g...
qwerqwer2@gmail.com	✉	16 Apr 2023	4 May 2023	3z5VWfKCRGSdlaF56cNOM2qFC8...

Figure c1.9

The authentication algorithms used in the code provide a secure and easy-to-use way to authenticate users and manage user accounts in the application. These algorithms allow users to log in with their email and password, register for a new account, or log in anonymously. The users are saved as an identifier being anonymous or saved with their email and given a unique user UID by Firebase.

Diary Implementation

The CRUD algorithms use Firebases cloud-based storage to create, read, update, and delete diary entries. When a user creates a new diary entry, the algorithm saves the data to the Firebase database using the users UID as the key. When a user reads a diary entry, the algorithm retrieves the data from the Firebase database using the same key. When a user updates or deletes a diary entry, the algorithm modifies or removes the data from the Firebase database accordingly.

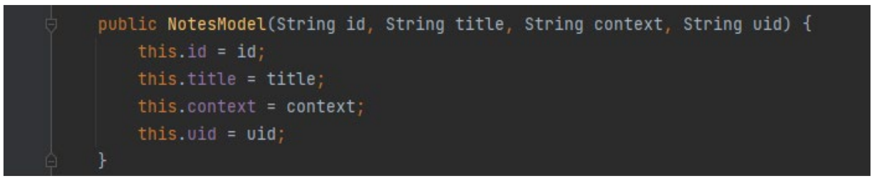


```
notes > 5e77a5fc-c284-..

kohiapp-e57ca          notes                             5e77a5fc-c284-4479-b48b-40f0229f8eba
notes                  0e3ab302-8b50-46d6-861c-a6b7       context: ""
user_timer             4dcf3bb8-75b6-4dfd-9358-e5db       id: "5e77a5fc-c284-4479-b48b-40f0229f8eba"
users_data             5e77a5fc-c284-4479-b48b-40f0       title: "cakeeee"
users_rewards          7208190d-7949-4606-afb0-a719       uid: "W10uHqbDnPOAi0meEy72ald4sLq1"
users_wallpaper_data   84c50dde-7039-4353-848b-ef05
                       b749adc2-3d08-44c8-94c7-0c90
                       e57be85b-a780-4afb-9c49-9eeb
                       ec8f120f-35a9-42d5-8d3c-cb45
                       ecd53286-0e8f-4e93-aea5-677f
```

Figure c2.0

The app uses Firebases cloud-based storage to create, read, update, and delete diary entries which is using the CRUD algorithm. When a user creates a new entry, the algorithm saves it to the Firebase database using the user's unique identifier (UID) as the key. Similarly, when a user reads an entry, the algorithm retrieves the data from the Firebase database using the same key. If a user updates or deletes an entry the algorithm modifies or removes the data from the Firebase database respectively.

The NotesModel class represents a note object. It has four private fields: id, title, context, and uid, which respectively represent the unique identifier, title, content, and user ID of the note.

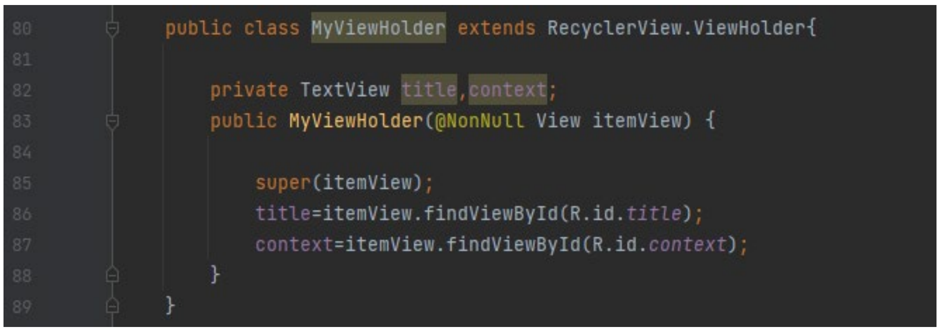


```java
public NotesModel(String id, String title, String context, String uid) {
    this.id = id;
    this.title = title;
    this.context = context;
    this.uid = uid;
}
```

Figure c2.01

The class has a default constructor and a parameterized constructor that initializes the fields with the given values. Additionally, the class has getter and setter methods for each field to retrieve and set their values.

The NoteAdapter class then is a custom adapter that extends the RecyclerView.Adapter class. It has a list of NotesModel objects and provides methods to add, clear, and filter the list. It also provides a method to get the list of NotesModel objects.

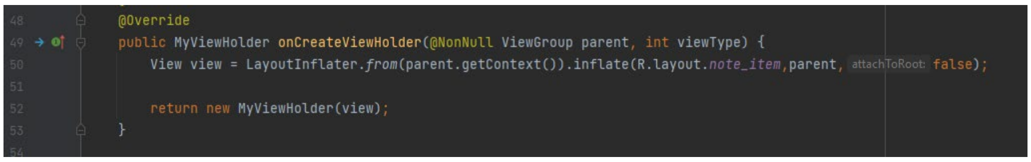


```
public class MyViewHolder extends RecyclerView.ViewHolder{

    private TextView title,context;
    public MyViewHolder(@NonNull View itemView) {

        super(itemView);
        title=itemView.findViewById(R.id.title);
        context=itemView.findViewById(R.id.context);
    }
}
```

Figure c2.1

The MyViewHolder class is a subclass of the RecyclerView.ViewHolder class. It is responsible for holding the references to the UI elements of each item in the RecyclerView. In this case the MyViewHolder class has two TextView members, title and context which are used to display the title and content of each note in the RecyclerView. By using the MyViewHolder class I can efficiently reuse the references to the UI elements of each item in the RecyclerView. This helps to reduce the inflating of new views every time an item needs to be displayed in the RecyclerView.

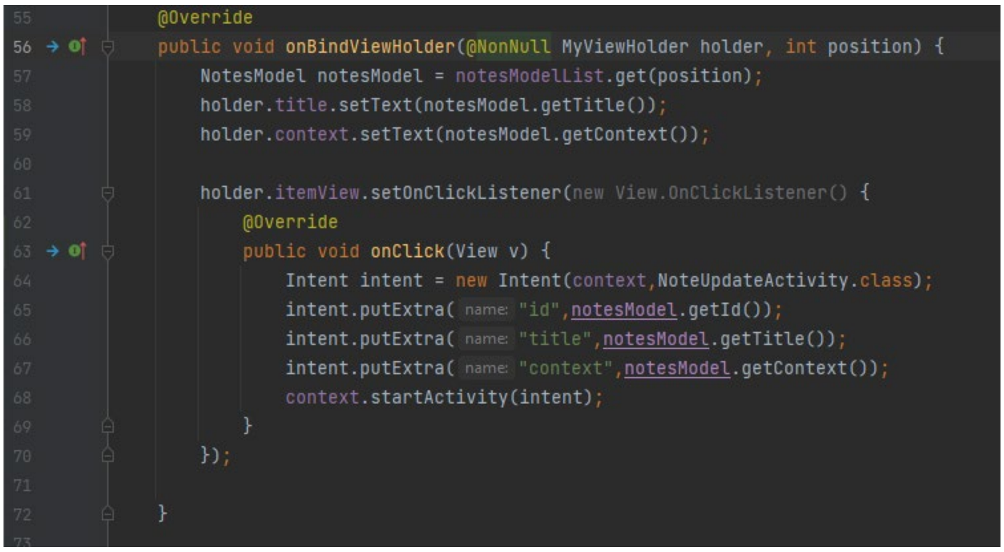
The adapter class has three main methods implemented in the class:



```
@Override
public MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.note_item,parent, attachToRoot false);

    return new MyViewHolder(view);
}
```

Figure c2.1

onCreateViewHolder(): This method is called when the RecyclerView needs a new ViewHolder object to represent an item in the list. This then inflates the layout for the item and returns a new ViewHolder object that holds references to the UI elements in the layout.



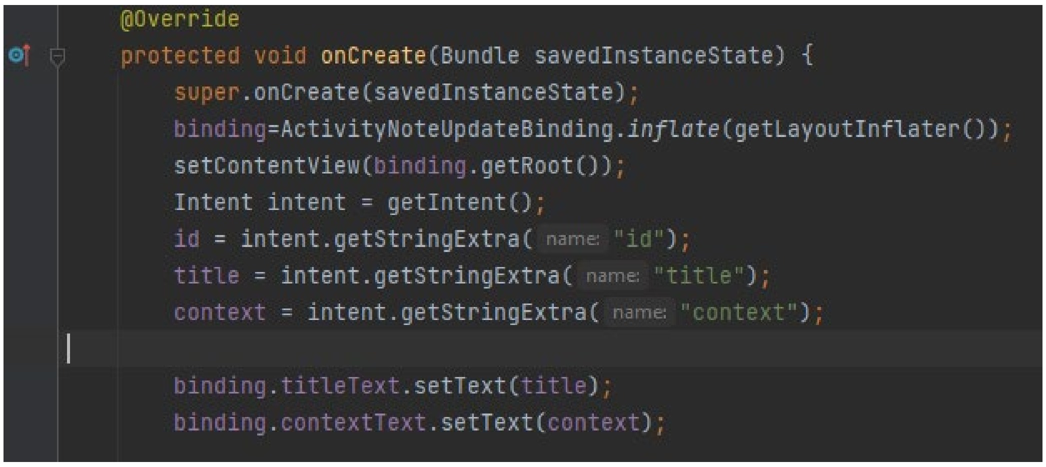
```
@Override
public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {
    NotesModel notesModel = notesModelList.get(position);
    holder.title.setText(notesModel.getTitle());
    holder.context.setText(notesModel.getContext());

    holder.itemView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(context,NoteUpdateActivity.class);
            intent.putExtra( name: "id",notesModel.getId());
            intent.putExtra( name: "title",notesModel.getTitle());
            intent.putExtra( name: "context",notesModel.getContext());
            context.startActivity(intent);
        }
    });

}
```

Figure c2.2

onBindViewHolder(): This method is called when the RecyclerView needs to bind the data to a ViewHolder object (holder). It gets the NotesModel object at the given position in the list and binds its data to the UI elements in the ViewHolder object. I then set an OnClickListener on the itemView of the ViewHolder object. So when the user clicks on the itemView the onClick() method of the OnClickListener is called. In this case the onClick() method creates an Intent object that starts a new activity called NoteUpdateActivity. It also puts extra data in the Intent, including the ID, title, and context of the NotesModel object that was clicked. This then starts the new activity using the context object.This is reason why the user can click on a note in the diary and view its details in a new activity called NoteUpdateActivity.

getItemCount(): This method then just returns the number of items in the list.The NoteActivity class is the main activity of the Diary functionality that displays a list of notes. It uses several classes and algorithms to fetch the notes from the Firestore database and display them in a RecyclerView.

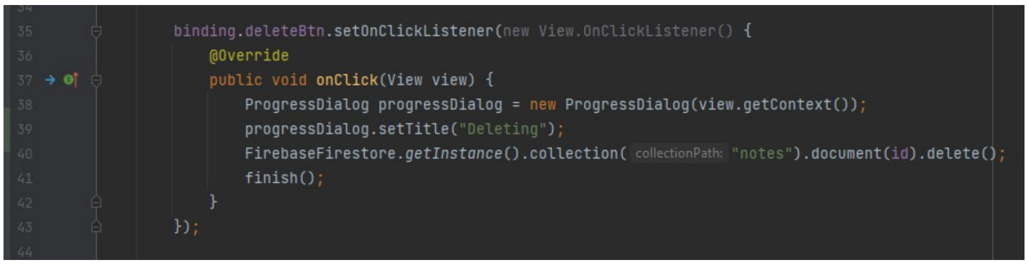
The NoteUpdateActivity class is the class that is responsible for viewing an existing note, updating an existing note, or deleting the note. The class has several private fields including the ID, title, and context of the note, and an instance of the ActivityNoteUpdateBinding class, which provides access to the views in the activity_note_update.xml layout file. (Projects, n.d.)



```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    binding=ActivityNoteUpdateBinding.inflate(getLayoutInflater());
    setContentView(binding.getRoot());
    Intent intent = getIntent();
    id = intent.getStringExtra( name: "id");
    title = intent.getStringExtra( name: "title");
    context = intent.getStringExtra( name: "context");

    binding.titleText.setText(title);
    binding.contextText.setText(context);
```

Figure c2.3

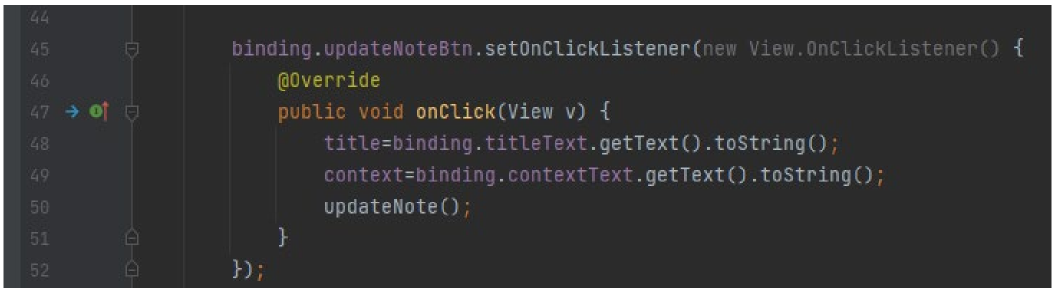
In the onCreate() method the ActivityNoteUpdateBinding object is inflated and set as the content view of the activity. The ID, title, and context of the note are retrieved from the Intent that started the activity and used to set the text of the titleText and contextText views. Click listeners are set up for the delete, update, and back buttons.



```java
        binding.deleteBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                ProgressDialog progressDialog = new ProgressDialog(view.getContext());
                progressDialog.setTitle("Deleting");
                FirebaseFirestore.getInstance().collection( collectionPath: "notes").document(id).delete();
                finish();
            }
        });
```

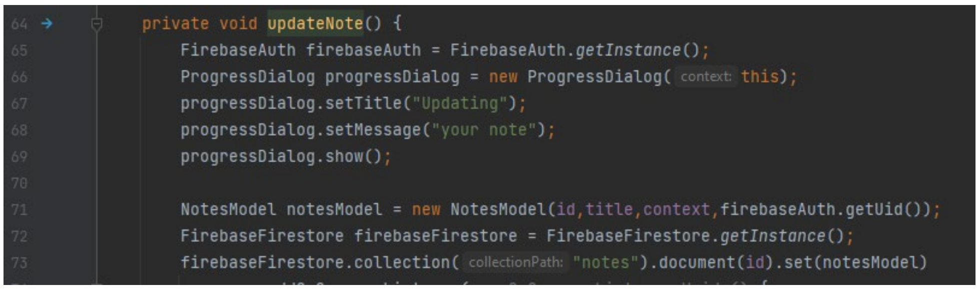
Figure c2.4

The deleteBtn click listener deletes the note from the Firestore database using the ID of the note. A ProgressDialog is displayed to indicate that the note is being deleted.



```java
        binding.updateNoteBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                title=binding.titleText.getText().toString();
                context=binding.contextText.getText().toString();
                updateNote();
            }
        });
```

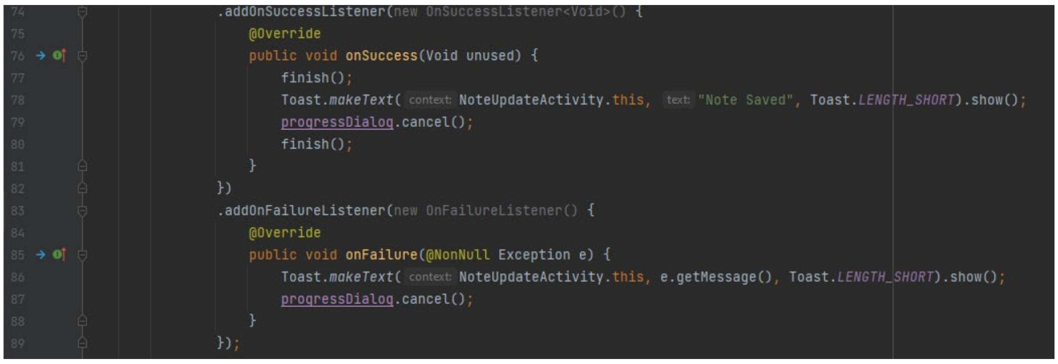
Figure c2.5



```java
    private void updateNote() {
        FirebaseAuth firebaseAuth = FirebaseAuth.getInstance();
        ProgressDialog progressDialog = new ProgressDialog( context: this);
        progressDialog.setTitle("Updating");
        progressDialog.setMessage("your note");
        progressDialog.show();

        NotesModel notesModel = new NotesModel(id,title,context,firebaseAuth.getUid());
        FirebaseFirestore firebaseFirestore = FirebaseFirestore.getInstance();
        firebaseFirestore.collection( collectionPath: "notes").document(id).set(notesModel)
```

Figure c2.6

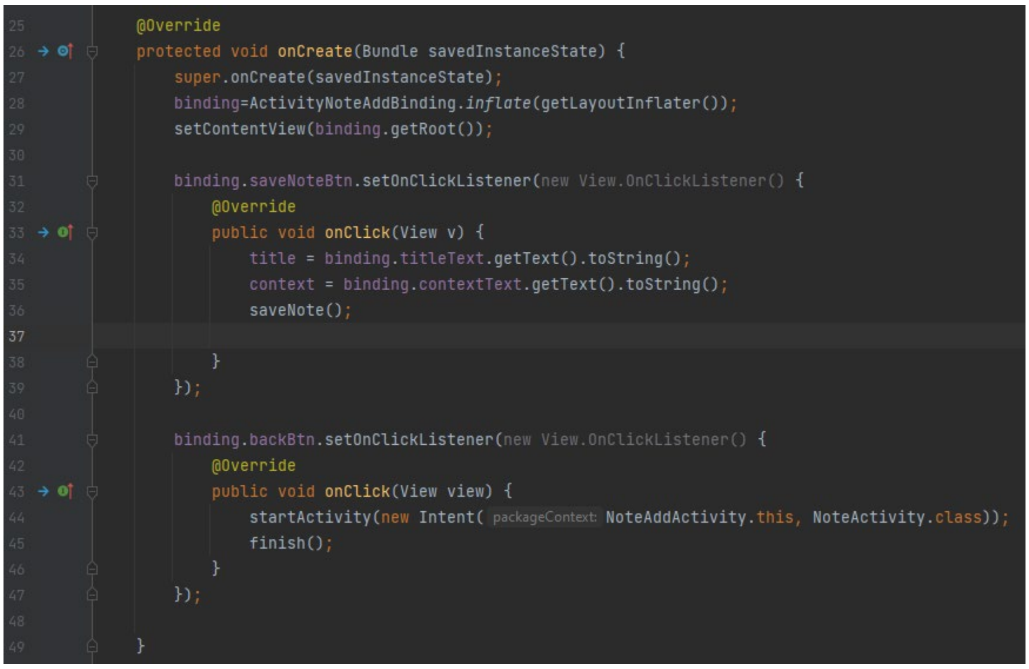
The updateNoteBtn click listener updates the note in the Firestore database with the new title and context entered by the user. A ProgressDialog is displayed to indicate that the note is being updated. The method first gets a reference to the FirebaseAuth instance to get the user ID. It then creates a new NotesModel object with the updated ID, title, and context. The FirebaseFirestore instance is used to update the note in the notes collection in the database with the new NotesModel object.



```java
                .addOnSuccessListener(new OnSuccessListener<Void>() {
                    @Override
                    public void onSuccess(Void unused) {
                        finish();
                        Toast.makeText( context: NoteUpdateActivity.this,  text: "Note Saved", Toast.LENGTH_SHORT).show();
                        progressDialog.cancel();
                        finish();
                    }
                })
                .addOnFailureListener(new OnFailureListener() {
                    @Override
                    public void onFailure(@NonNull Exception e) {
                        Toast.makeText( context: NoteUpdateActivity.this, e.getMessage(), Toast.LENGTH_SHORT).show();
                        progressDialog.cancel();
                    }
                });
```

Figure c2.7

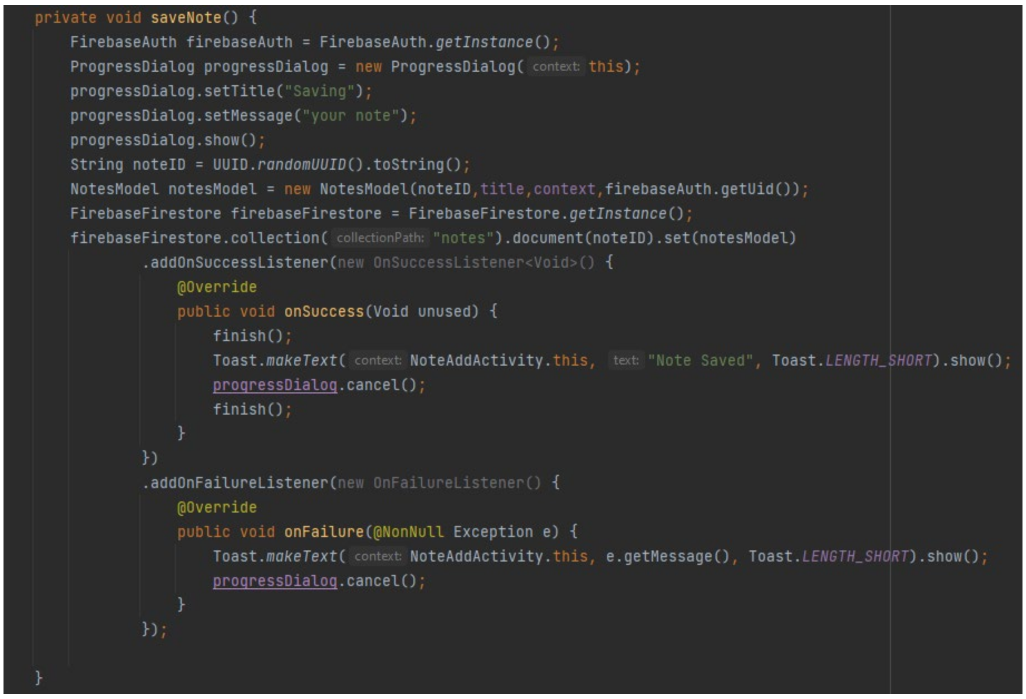
If the update is successful a Toast message is displayed to indicate that the note has been saved and the activity is finished. Else if its not successful it will display the error message to the user.



```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    binding=ActivityNoteAddBinding.inflate(getLayoutInflater());
    setContentView(binding.getRoot());

    binding.saveNoteBtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            title = binding.titleText.getText().toString();
            context = binding.contextText.getText().toString();
            saveNote();

        }
    });

    binding.backBtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            startActivity(new Intent( packageContext: NoteAddActivity.this, NoteActivity.class));
            finish();
        }
    });

}
```

Figure c2.8

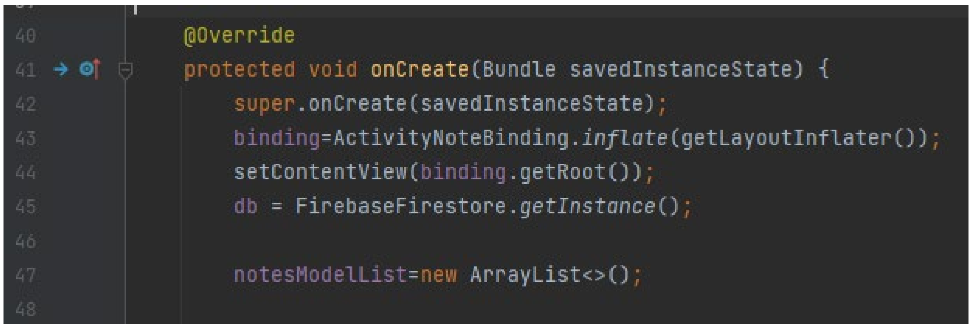
The NoteAddActivity class then is responsible for adding a new note to the Firestore database. In the onCreate() method the ActivityNoteAddBinding object is inflated and set as the content view of the activity.



```java
private void saveNote() {
    FirebaseAuth firebaseAuth = FirebaseAuth.getInstance();
    ProgressDialog progressDialog = new ProgressDialog( context: this);
    progressDialog.setTitle("Saving");
    progressDialog.setMessage("your note");
    progressDialog.show();
    String noteID = UUID.randomUUID().toString();
    NotesModel notesModel = new NotesModel(noteID,title,context,firebaseAuth.getUid());
    FirebaseFirestore firebaseFirestore = FirebaseFirestore.getInstance();
    firebaseFirestore.collection( collectionPath: "notes").document(noteID).set(notesModel)
            .addOnSuccessListener(new OnSuccessListener<Void>() {
                @Override
                public void onSuccess(Void unused) {
                    finish();
                    Toast.makeText( context: NoteAddActivity.this,  text: "Note Saved", Toast.LENGTH_SHORT).show();
                    progressDialog.cancel();
                    finish();
                }
            })
            .addOnFailureListener(new OnFailureListener() {
                @Override
                public void onFailure(@NonNull Exception e) {
                    Toast.makeText( context: NoteAddActivity.this, e.getMessage(), Toast.LENGTH_SHORT).show();
                    progressDialog.cancel();
                }
            });

}
```

Figure c2.9

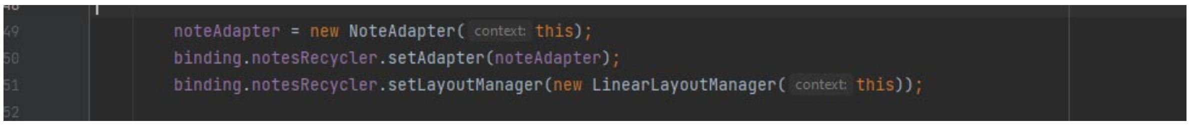
The saveNote() method is called when the user clicks the save button. The method first gets a reference to the FirebaseAuth instance to get the user ID. It then creates a new NotesModel object with a randomly generated ID, the title and context entered by the user, and the user ID. A ProgressDialog is displayed to indicate that the note is being saved. The FirebaseFirestore instance is used to add the new note to the notes collection in the database with the new NotesModel object. If the note is successfully saved a Toast message is displayed to indicate that the note has been saved and the activity is finished. Else it will then display the error message to the user.



```java
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding=ActivityNoteBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());
        db = FirebaseFirestore.getInstance();

        notesModelList=new ArrayList<>();

```

Figure c3.0

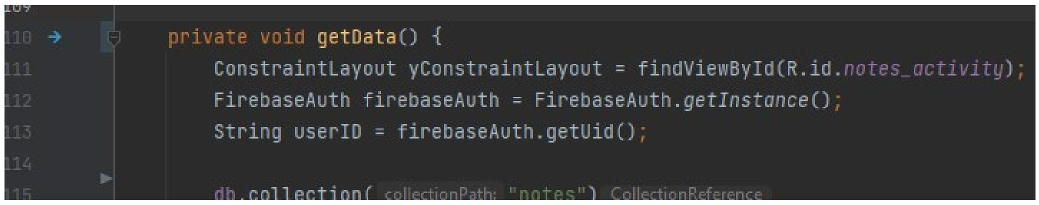
Lastly is the main class of the diary functionality where its responsible for managing the UI and data for the diary feature of the application. In the onCreate method, the layout file for the activity is inflated using the ActivityNoteBinding class which generates a binding class that allows access to the UI elements defined in the XML layout file. A new instance of the FirebaseFirestore class is created to interact with the Firebase Firestore database. A new empty ArrayList of NotesModel objects is created to hold the notes that will be retrieved from the database.



```java
        noteAdapter = new NoteAdapter( context: this);
        binding.notesRecycler.setAdapter(noteAdapter);
        binding.notesRecycler.setLayoutManager(new LinearLayoutManager( context: this));

```

Figure c3.1

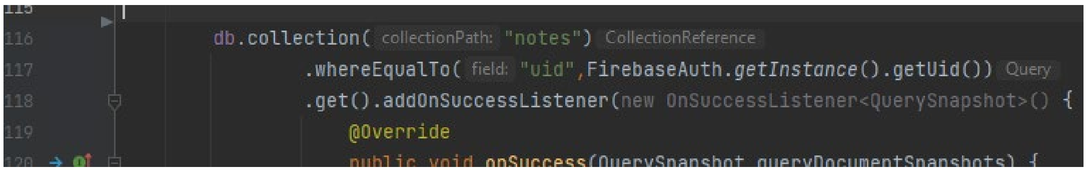
The NoteAdapter class is instantiated, which is a custom RecyclerView adapter that is responsible for displaying the notes in the UI. The adapter is set on the RecyclerView using the notesRecycler element from the layout file. A new LinearLayoutManager is created to set the orientation of the RecyclerView to vertical.



```java
    private void getData() {
        ConstraintLayout yConstraintLayout = findViewById(R.id.notes_activity);
        FirebaseAuth firebaseAuth = FirebaseAuth.getInstance();
        String userID = firebaseAuth.getUid();

        db.collection( collectionPath: "notes") CollectionReference
```

Figure c3.2

To retrieve the diary entries the getData() method in the NoteActivity uses the Firebase Firestore API. It first initializes a ConstraintLayout object to get the ID of the main layout of the activity and then gets the current user ID from the FirebaseAuth object.

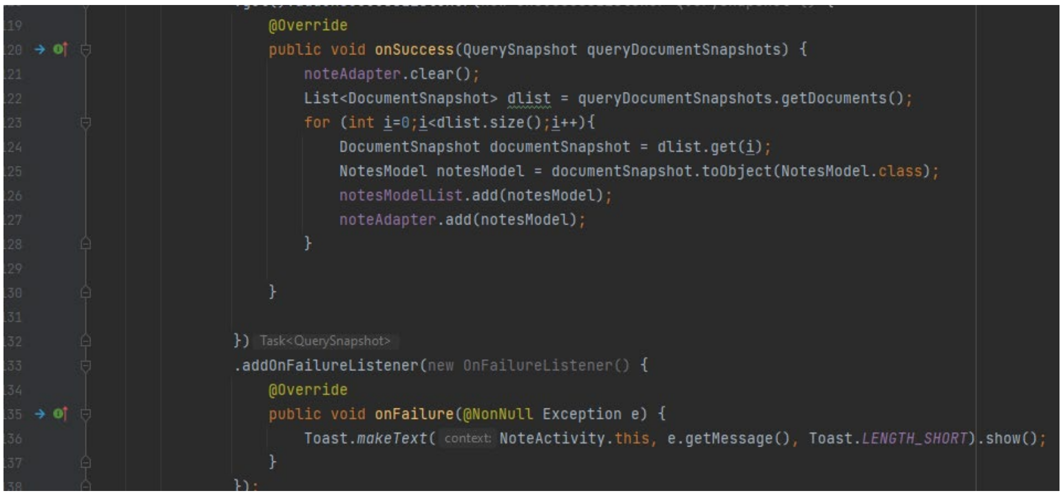


```
        db.collection( collectionPath: "notes") CollectionReference
                .whereEqualTo( field: "uid",FirebaseAuth.getInstance().getUid()) Query
                .get().addOnSuccessListener(new OnSuccessListener<QuerySnapshot>() {
                    @Override
                    public void onSuccess(QuerySnapshot queryDocumentSnapshots) {
```

Figure c3.3

Next the method uses the collection() method to specify the name of the collection from which to retrieve the documents which in this case I have called it notes. It then queries the collection using the whereEqualTo() method to retrieve only the documents that match the user ID of the currently logged-in user. This makes it only the owner of the document is retrieving their own data.

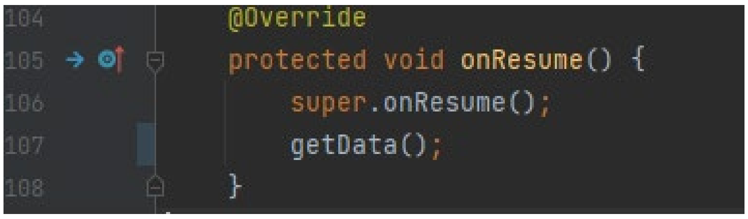
Figure c3.4



```
                    @Override
                    public void onSuccess(QuerySnapshot queryDocumentSnapshots) {
                        noteAdapter.clear();
                        List<DocumentSnapshot> dlist = queryDocumentSnapshots.getDocuments();
                        for (int i=0;i<dlist.size();i++){
                            DocumentSnapshot documentSnapshot = dlist.get(i);
                            NotesModel notesModel = documentSnapshot.toObject(NotesModel.class);
                            notesModelList.add(notesModel);
                            noteAdapter.add(notesModel);
                        }

                    }

                }) Task<QuerySnapshot>
                .addOnFailureListener(new OnFailureListener() {
                    @Override
                    public void onFailure(@NonNull Exception e) {
                        Toast.makeText( context: NoteActivity.this, e.getMessage(), Toast.LENGTH_SHORT).show();
                    }
                });
```

Figure c3.5

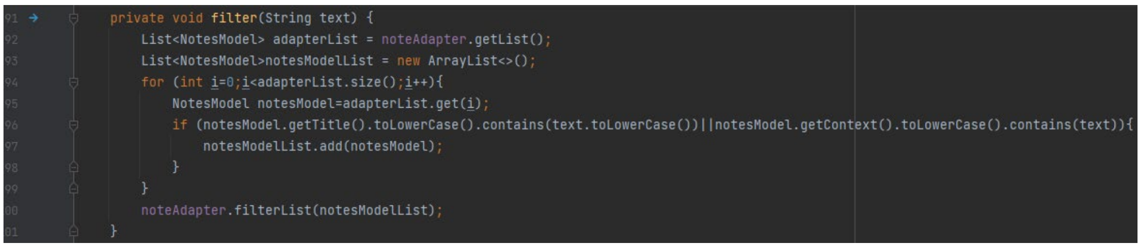
If the query is successful the onSuccess() method clears the noteAdapter and notesModelList lists. It then loops through the retrieved documents using a for loop and converts each document to an object of the NotesModel class using the toObject() method. Finally it adds each NotesModel object to the notesModelList and noteAdapter lists using the add() method. This way the notesModelList and noteAdapter lists contain all the diary entries that match the user ID of the currently logged-in user which are then displayed in the RecyclerView of the NoteActivity class. If the query is unsuccessful the onFailure() method will prompt the error message to the user.



```
    @Override
    protected void onResume() {
        super.onResume();
        getData();
    }
```

Figure c3.6

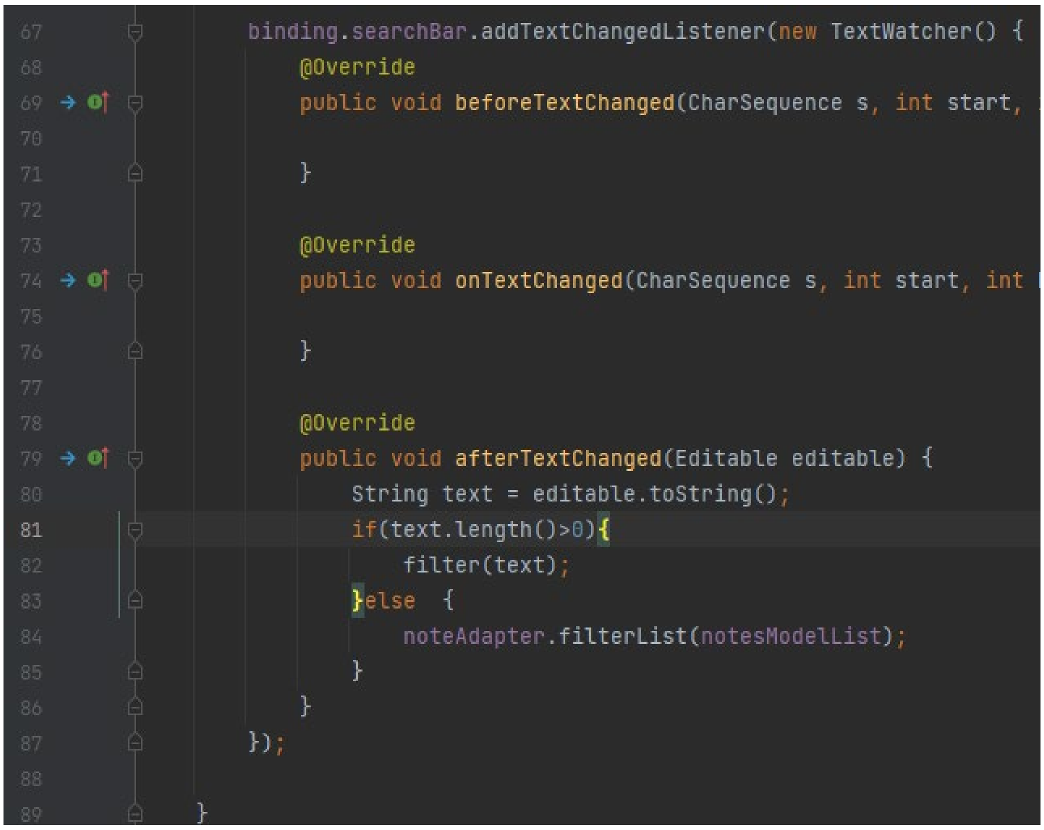
If the application was closed, I have added a resume method where when the application is opened again without being closed it will retrieve the data from the Firestore database using the getData() Method.



```java
    private void filter(String text) {
        List<NotesModel> adapterList = noteAdapter.getList();
        List<NotesModel>notesModelList = new ArrayList<>();
        for (int i=0;i<adapterList.size();i++){
            NotesModel notesModel=adapterList.get(i);
            if (notesModel.getTitle().toLowerCase().contains(text.toLowerCase())||notesModel.getContext().toLowerCase().contains(text)){
                notesModelList.add(notesModel);
            }
        }
        noteAdapter.filterList(notesModelList);
    }
```

Figure c3.7

The NoteActivity class also has the filter() method in the notes which is used to filter the notes based on the search text entered by the user. It first gets the list of notes from the NoteAdapter using the getList() method. It then loops through the list of notes and checks if the title or context of each note contains the search text using the contains() method. If a note contains the search text it is added to a new list called notesModelList. Finally, the filterList() method of the NoteAdapter is called with the notesModelList as the argument to update the RecyclerView with the filtered notes.



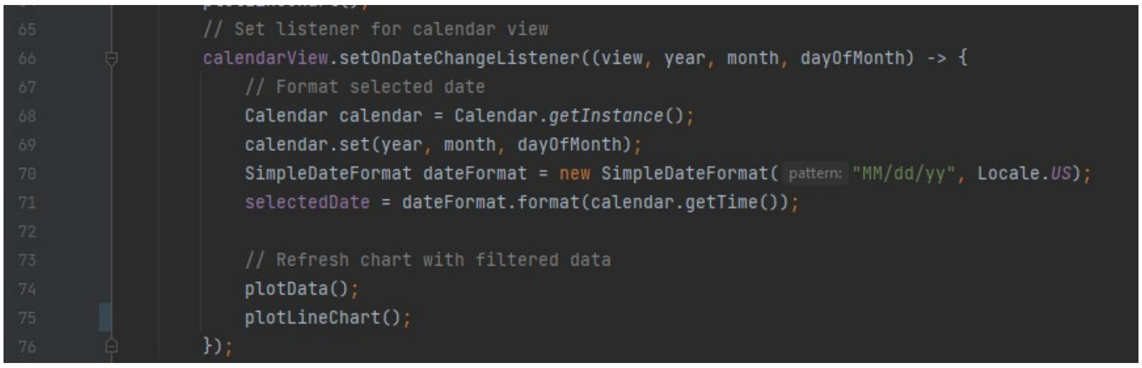
```java
        binding.searchBar.addTextChangedListener(new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start,

            }

            @Override
            public void onTextChanged(CharSequence s, int start, int

            }

            @Override
            public void afterTextChanged(Editable editable) {
                String text = editable.toString();
                if(text.length()>0){
                    filter(text);
                }else  {
                    noteAdapter.filterList(notesModelList);
                }
            }
        });

    }
```

Figure c3.8

The filter() method is called from the afterTextChanged() method of the TextWatcher interface which is set on the searchBar using the addTextChangedListener() method. In the afterTextChanged() method it gets the search text entered by the user using the toString() method of the Editable parameter called editable. If the length of the search text is greater than 0 it calls the filter() method with the search text as the argument. Otherwise it calls the filterList() method in the NoteAdapter with the original notesModelList as the argument to reset the RecyclerView to its original state. (Projects, n.d.)

Analytics Implementation

The analytics algorithms use Firebases cloud-based storage to calculate and display user data. When a user completes a focus session, the algorithm saves the data to the Firebase database using the users UID as the key. The algorithm then retrieves the data from the Firebase database and calculates the total time spent on focus sessions. The results are then displayed to the user in the Analytics component of the application which are the charts.

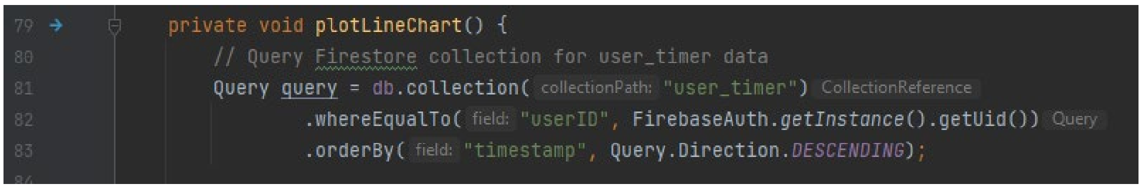


```
        // Set listener for calendar view
        calendarView.setOnDateChangeListener((view, year, month, dayOfMonth) -> {
            // Format selected date
            Calendar calendar = Calendar.getInstance();
            calendar.set(year, month, dayOfMonth);
            SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "MM/dd/yy", Locale.US);
            selectedDate = dateFormat.format(calendar.getTime());

            // Refresh chart with filtered data
            plotData();
            plotLineChart();
        });
```

Figure c3.9

In the onCreate() method it sets a listener for the calendar view. When the user selects a date on the calendar view the listener is triggered and the selected date is formatted using the Calendar and SimpleDateFormat classes. The Calendar.getInstance() method gets a new instance of the Calendar class and the calendar.set(year, month, dayOfMonth) method sets the year, month, and day values of the calendar instance to the selected date. The SimpleDateFormat class is then used to format the date in a specific format, MM/dd/yy, using the dateFormat.format(calendar.getTime()) method. The resulting formatted date is stored in the selectedDate variable.After the selected date is formatted, the plotData() and plotLineChart() methods are called to refresh the chart with filtered data and plot the line chart with the selected date.

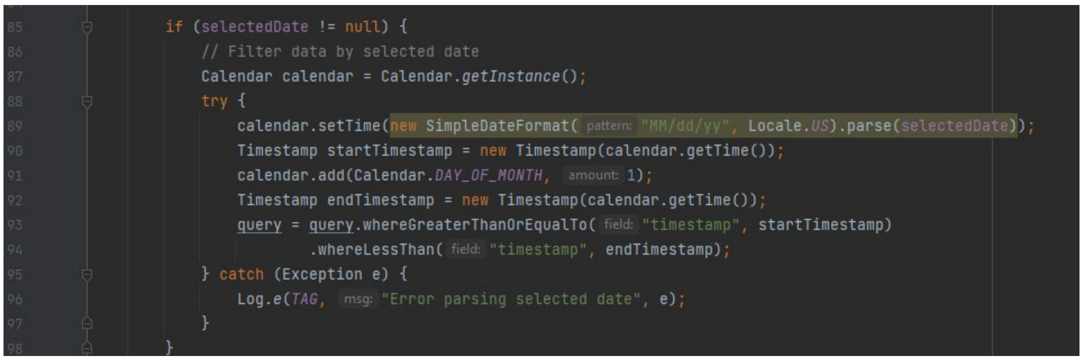
In this class it contains two types of charts for visual representation. These are a line chart and a bar chart. The method plotlinechart() is responsible for displaying the line chart while the method plotdata() is responsible for displaying the bar chart.



```
    private void plotLineChart() {
        // Query Firestore collection for user_timer data
        Query query = db.collection( collectionPath: "user_timer") CollectionReference
                .whereEqualTo( field: "userID", FirebaseAuth.getInstance().getUid()) Query
                .orderBy( field: "timestamp", Query.Direction.DESCENDING);
```

Figure c4.0

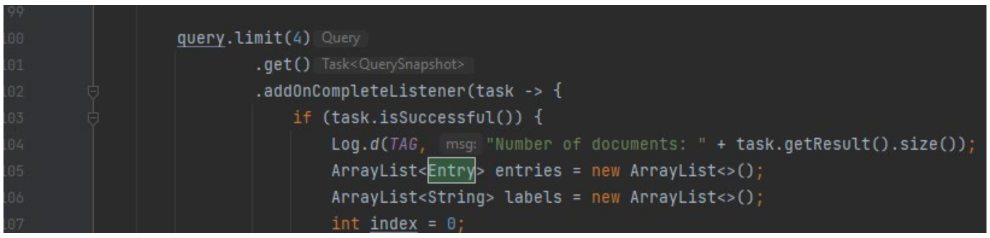
The plotLineChart method performs a query to retrieve data from the Firestore database. Specifically, it queries the user_timer collection for documents with a matching userID field value, which is obtained from the FirebaseAuth instance. The query is executed on the Firestore instance db. The retrieved documents are then ordered based on their timestamp field in a descending order, meaning the most recent documents appear first.



```java
if (selectedDate != null) {
    // Filter data by selected date
    Calendar calendar = Calendar.getInstance();
    try {
        calendar.setTime(new SimpleDateFormat("MM/dd/yy", Locale.US).parse(selectedDate));
        Timestamp startTimestamp = new Timestamp(calendar.getTime());
        calendar.add(Calendar.DAY_OF_MONTH, 1);
        Timestamp endTimestamp = new Timestamp(calendar.getTime());
        query = query.whereGreaterThanOrEqualTo("timestamp", startTimestamp)
                .whereLessThan("timestamp", endTimestamp);
    } catch (Exception e) {
        Log.e(TAG, "Error parsing selected date", e);
    }
}
```

Figure c4.1

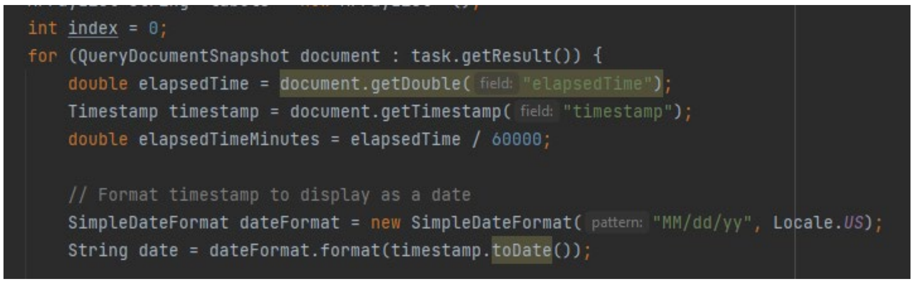
If a selectedDate exists (set by the setOnDateChangeListener listener for the calendarView) the query is filtered to only include documents with timestamp values that fall within a 24-hour period of the selectedDate.



```java
query.limit(4)
        .get()
        .addOnCompleteListener(task -> {
            if (task.isSuccessful()) {
                Log.d(TAG, "Number of documents: " + task.getResult().size());
                ArrayList<Entry> entries = new ArrayList<>();
                ArrayList<String> labels = new ArrayList<>();
                int index = 0;
```

Figure c4.2

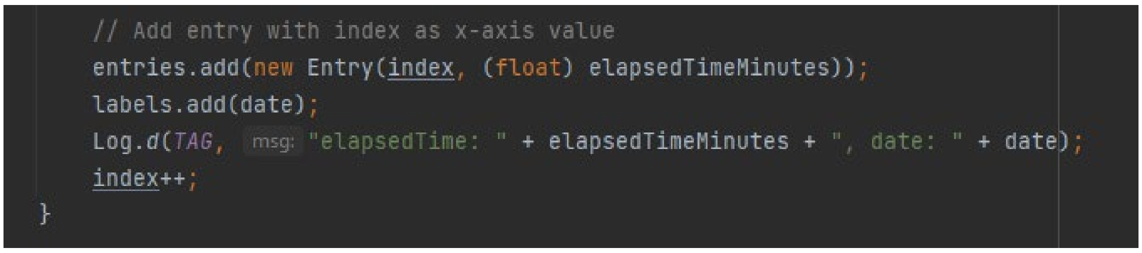
The query is limited to a maximum of four documents. When the query is completed, the tasks completion listener populates an ArrayList of Entry objects with the retrieved data from the documents. Each Entry object represents a data point with an x-value of the index in the ArrayList and a y-value of the elapsed time in minutes. A separate ArrayList of String objects is also populated with the formatted dates corresponding to each Entry object.



```java
int index = 0;
for (QueryDocumentSnapshot document : task.getResult()) {
    double elapsedTime = document.getDouble("elapsedTime");
    Timestamp timestamp = document.getTimestamp("timestamp");
    double elapsedTimeMinutes = elapsedTime / 60000;

    // Format timestamp to display as a date
    SimpleDateFormat dateFormat = new SimpleDateFormat("MM/dd/yy", Locale.US);
    String date = dateFormat.format(timestamp.toDate());
```

Figure c4.3

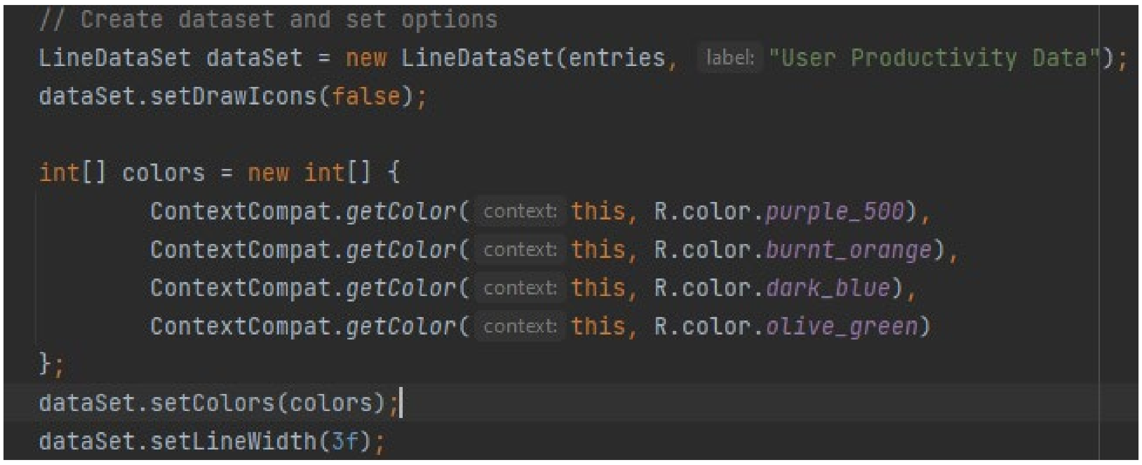
The method then retrieves data from each of the four documents returned by the Firestore database query. For each document we get the elapsedTime and timestamp values using a for loop. The elapsedTime is in milliseconds so its divided by 60000 to convert it to minutes and store it in a new variable called elapsedTimeMinutes. Then the timestamp value is formated as a string in a MM/dd/yy format using a SimpleDateFormat object.



```
    // Add entry with index as x-axis value
    entries.add(new Entry(index, (float) elapsedTimeMinutes));
    labels.add(date);
    Log.d(TAG,  msg: "elapsedTime: " + elapsedTimeMinutes + ", date: " + date);
    index++;
}
```

Figure c4.4

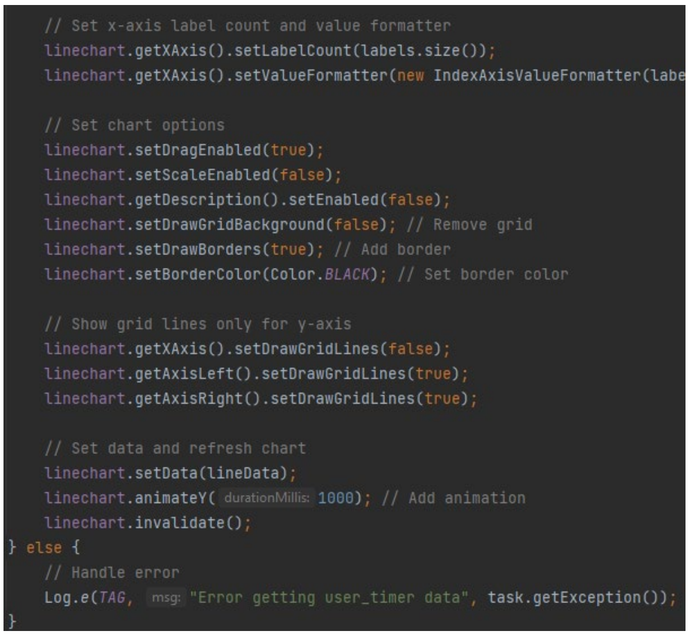
This allows to display the timestamp as a date in the chart. The formatted date is then added to an ArrayList called labels which will be used to label the x-axis of the chart. It then creates a new Entry object using the index as the x-value and the elapsedTimeMinutes as the y-value. This Entry object represents a single data point in the chart. We add this Entry object to an ArrayList called entries. We log the elapsedTimeMinutes and formatted date for debugging purposes and increment the index by 1 to move to the next document. Once all four documents have been processed the ArrayList of Entry objects and an ArrayList of formatted date strings will be used to create the LineDataSet object and populate the chart.



```
// Create dataset and set options
LineDataSet dataSet = new LineDataSet(entries,  label: "User Productivity Data");
dataSet.setDrawIcons(false);

int[] colors = new int[] {
        ContextCompat.getColor( context: this, R.color.purple_500),
        ContextCompat.getColor( context: this, R.color.burnt_orange),
        ContextCompat.getColor( context: this, R.color.dark_blue),
        ContextCompat.getColor( context: this, R.color.olive_green)
};
dataSet.setColors(colors);
dataSet.setLineWidth(3f);
```

Figure c4.5

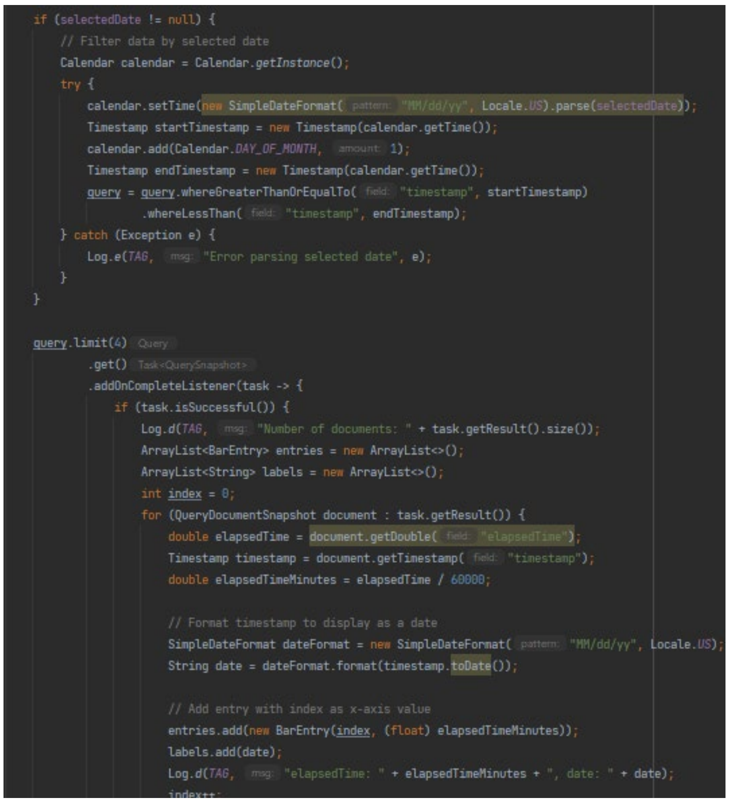
The method then creates a LineDataSet object with the entries and sets its options, such as color, line width, and icons. It then creates a LineData object with the dataSet. The method sets the label count and value formatter for the x-axis enables dragging, disables scaling, removes the grid, adds a border, and sets the border color. The method also shows grid lines only for the y-axis.



```
    // Set x-axis label count and value formatter
    linechart.getXAxis().setLabelCount(labels.size());
    linechart.getXAxis().setValueFormatter(new IndexAxisValueFormatter(labe

    // Set chart options
    linechart.setDragEnabled(true);
    linechart.setScaleEnabled(false);
    linechart.getDescription().setEnabled(false);
    linechart.setDrawGridBackground(false); // Remove grid
    linechart.setDrawBorders(true); // Add border
    linechart.setBorderColor(Color.BLACK); // Set border color

    // Show grid lines only for y-axis
    linechart.getXAxis().setDrawGridLines(false);
    linechart.getAxisLeft().setDrawGridLines(true);
    linechart.getAxisRight().setDrawGridLines(true);

    // Set data and refresh chart
    linechart.setData(lineData);
    linechart.animateY( durationMillis: 1000); // Add animation
    linechart.invalidate();
} else {
    // Handle error
    Log.e(TAG,  msg: "Error getting user_timer data", task.getException());
}
```

Figure c4.6

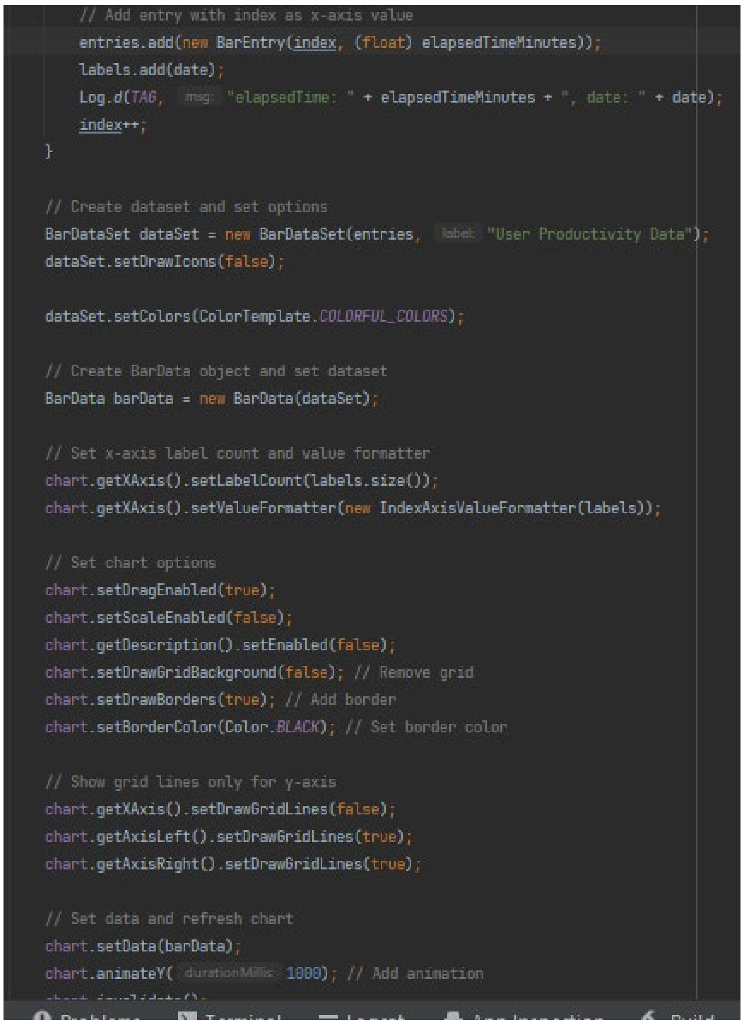
A LineDataSet object is created with the ArrayList of Entry objects and is customized with a title, line colors, and width. A LineData object is created with the LineDataSet object and is used to populate the linechart view. The linechart view is then customized with axis labels, grid lines, and animations.



```
if (selectedDate != null) {
    // Filter data by selected date
    Calendar calendar = Calendar.getInstance();
    try {
        calendar.setTime(new SimpleDateFormat( pattern: "MM/dd/yy", Locale.US).parse(selectedDate));
        Timestamp startTimestamp = new Timestamp(calendar.getTime());
        calendar.add(Calendar.DAY_OF_MONTH,  amount: 1);
        Timestamp endTimestamp = new Timestamp(calendar.getTime());
        query = query.whereGreaterThanOrEqualTo( field: "timestamp", startTimestamp)
                .whereLessThan( field: "timestamp", endTimestamp);
    } catch (Exception e) {
        Log.e(TAG,  msg: "Error parsing selected date", e);
    }
}

query.limit(4) Query
        .get() Task<QuerySnapshot>
        .addOnCompleteListener(task -> {
            if (task.isSuccessful()) {
                Log.d(TAG,  msg: "Number of documents: " + task.getResult().size());
                ArrayList<BarEntry> entries = new ArrayList<>();
                ArrayList<String> labels = new ArrayList<>();
                int index = 0;
                for (QueryDocumentSnapshot document : task.getResult()) {
                    double elapsedTime = document.getDouble( field: "elapsedTime");
                    Timestamp timestamp = document.getTimestamp( field: "timestamp");
                    double elapsedTimeMinutes = elapsedTime / 60000;

                    // Format timestamp to display as a date
                    SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "MM/dd/yy", Locale.US);
                    String date = dateFormat.format(timestamp.toDate());

                    // Add entry with index as x-axis value
                    entries.add(new BarEntry(index, (float) elapsedTimeMinutes));
                    labels.add(date);
                    Log.d(TAG,  msg: "elapsedTime: " + elapsedTimeMinutes + ", date: " + date);
                    index++;
```

Figure c4.8

The plotdata method also performs the same query to retrieve data from the Firestore database as the plotLineChart method.



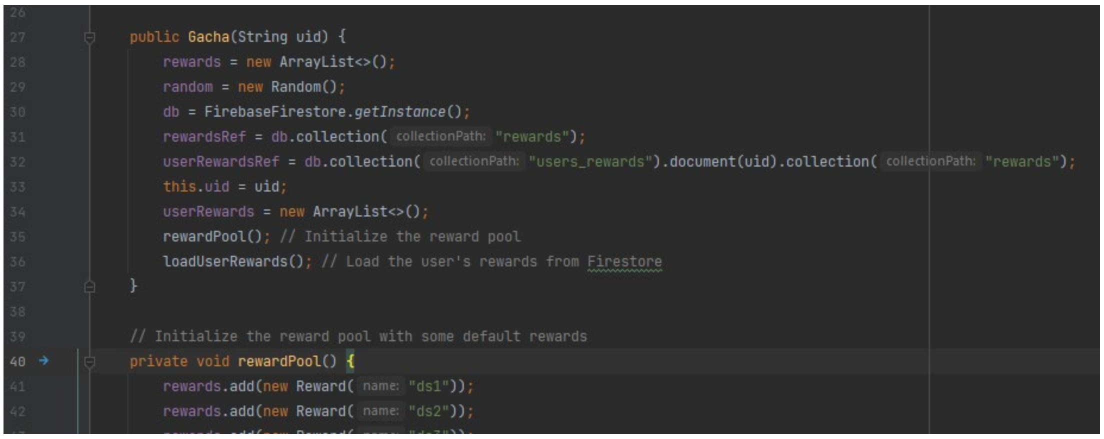
```
        // Add entry with index as x-axis value
        entries.add(new BarEntry(index, (float) elapsedTimeMinutes));
        labels.add(date);
        Log.d(TAG, msg: "elapsedTime: " + elapsedTimeMinutes + ", date: " + date);
        index++;
    }

    // Create dataset and set options
    BarDataSet dataSet = new BarDataSet(entries, label: "User Productivity Data");
    dataSet.setDrawIcons(false);

    dataSet.setColors(ColorTemplate.COLORFUL_COLORS);

    // Create BarData object and set dataset
    BarData barData = new BarData(dataSet);

    // Set x-axis label count and value formatter
    chart.getXAxis().setLabelCount(labels.size());
    chart.getXAxis().setValueFormatter(new IndexAxisValueFormatter(labels));

    // Set chart options
    chart.setDragEnabled(true);
    chart.setScaleEnabled(false);
    chart.getDescription().setEnabled(false);
    chart.setDrawGridBackground(false); // Remove grid
    chart.setDrawBorders(true); // Add border
    chart.setBorderColor(Color.BLACK); // Set border color

    // Show grid lines only for y-axis
    chart.getXAxis().setDrawGridLines(false);
    chart.getAxisLeft().setDrawGridLines(true);
    chart.getAxisRight().setDrawGridLines(true);

    // Set data and refresh chart
    chart.setData(barData);
    chart.animateY( durationMillis: 1000); // Add animation
```

Figure c4.9

The only difference is that a bar chart is being displayed.

Gacha Implementation

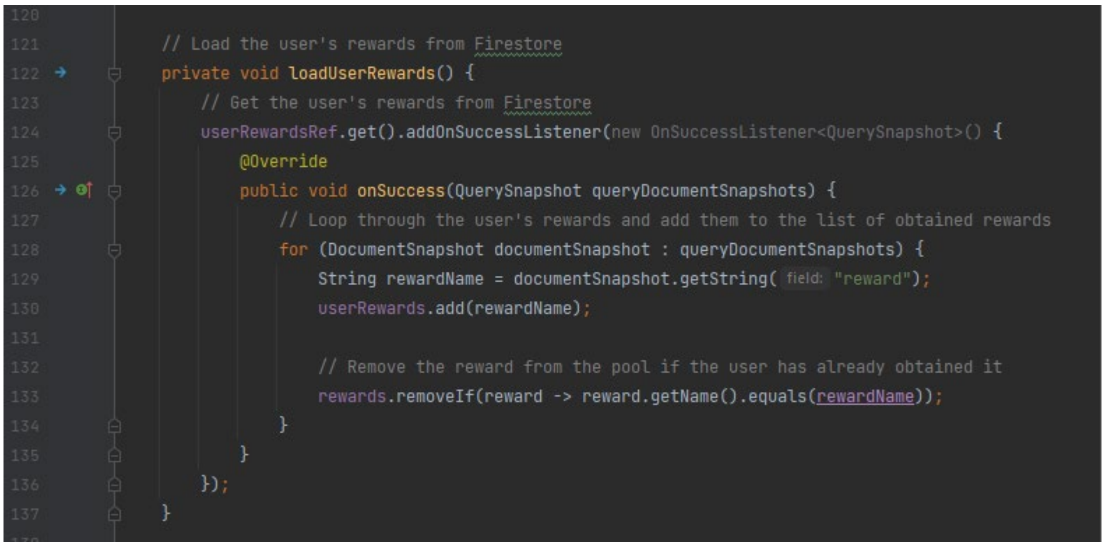
The Gacha class is used to play the gacha. The class has several attributes, including a list of Reward objects representing the available rewards in the game, a Random object used to select a random reward from the reward pool, a FirebaseFirestore object used to interact with the Firestore database and CollectionReference objects representing the "rewards" and "users_rewards" collections in Firestore. The class also has a String representing the user ID and an ArrayList of Strings representing the rewards obtained by the user.



```java
    public Gacha(String uid) {
        rewards = new ArrayList<>();
        random = new Random();
        db = FirebaseFirestore.getInstance();
        rewardsRef = db.collection( collectionPath: "rewards");
        userRewardsRef = db.collection( collectionPath: "users_rewards").document(uid).collection( collectionPath: "rewards");
        this.uid = uid;
        userRewards = new ArrayList<>();
        rewardPool(); // Initialize the reward pool
        loadUserRewards(); // Load the user's rewards from Firestore
    }

    // Initialize the reward pool with some default rewards
    private void rewardPool() {
        rewards.add(new Reward( name: "ds1"));
        rewards.add(new Reward( name: "ds2"));
```

Figure c5.0

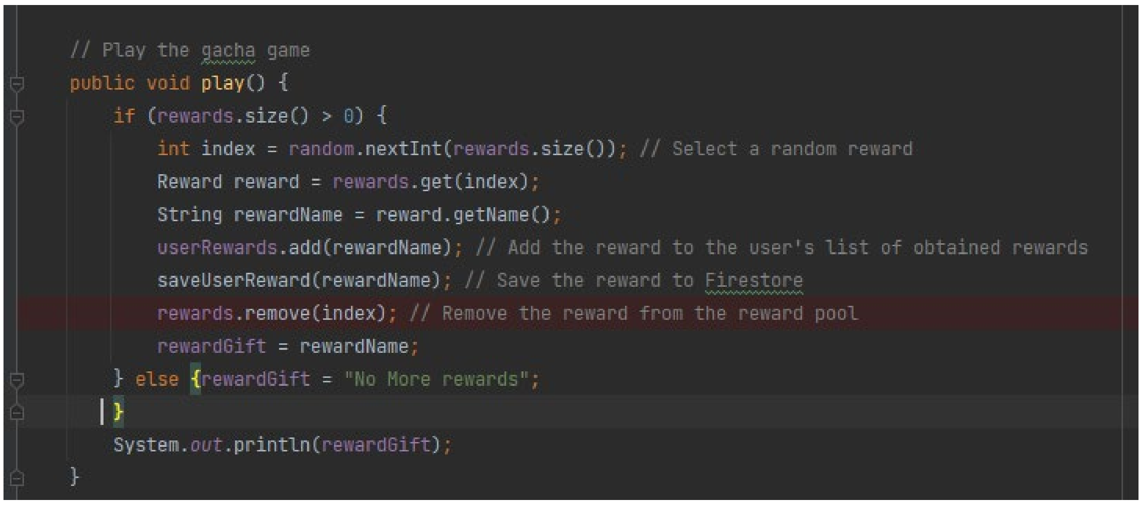
The Gacha constructor initializes the rewards pool and loads the user's rewards from Firestore. The rewardPool() method initializes the reward pool with some default rewards while the loadUserRewards() method loads the user's rewards from Firestore and adds them to the list of obtained rewards.



```java
    // Load the user's rewards from Firestore
    private void loadUserRewards() {
        // Get the user's rewards from Firestore
        userRewardsRef.get().addOnSuccessListener(new OnSuccessListener<QuerySnapshot>() {
            @Override
            public void onSuccess(QuerySnapshot queryDocumentSnapshots) {
                // Loop through the user's rewards and add them to the list of obtained rewards
                for (DocumentSnapshot documentSnapshot : queryDocumentSnapshots) {
                    String rewardName = documentSnapshot.getString( field: "reward");
                    userRewards.add(rewardName);

                    // Remove the reward from the pool if the user has already obtained it
                    rewards.removeIf(reward -> reward.getName().equals(rewardName));
                }
            }
        });
    }
```

Figure c5.1

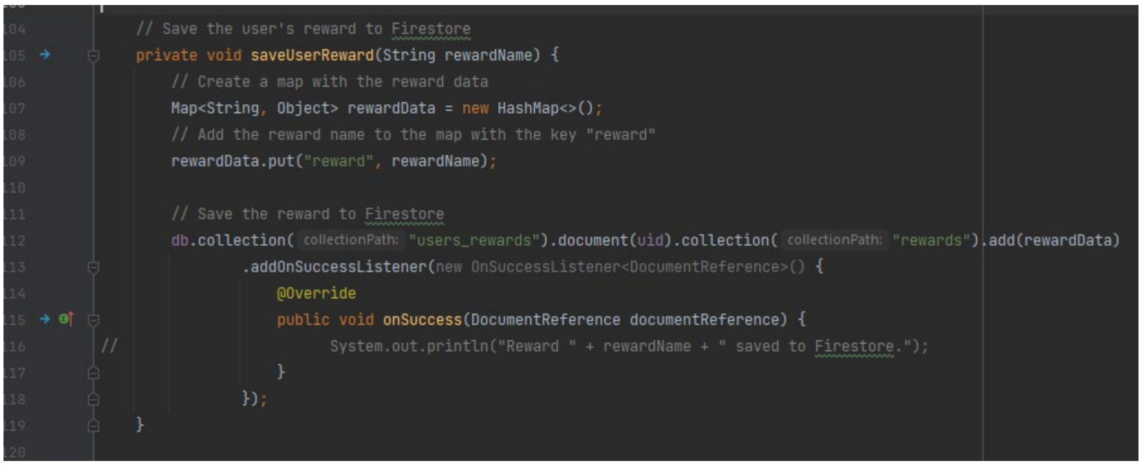
The loadUserRewards() method is a private method in the Gacha class that is called from the constructor. The purpose of this method is to load the user's rewards from Firestore and add them to the list of obtained rewards. The method starts by calling the get() method on the userRewardsRef CollectionReference object. The onSuccess() method is then called on this object which takes a OnSuccessListener<QuerySnapshot> object as a parameter. This listener is called when the query is successful and returns a QuerySnapshot object containing the results of the query. Inside the onSuccess() method it loops through the documents in the QuerySnapshot object using a for loop. For each document the method retrieves the "reward" field using the getString() method adds it to the userRewards ArrayList. This ArrayList represents the rewards obtained by the user. After adding the reward to the ArrayList the method then checks if the reward is already in the rewards pool. If the reward is already in the pool, it is removed using the removeIf() method. This makes sure that the user cannot obtain the same reward twice.



```
// Play the gacha game
public void play() {
    if (rewards.size() > 0) {
        int index = random.nextInt(rewards.size()); // Select a random reward
        Reward reward = rewards.get(index);
        String rewardName = reward.getName();
        userRewards.add(rewardName); // Add the reward to the user's list of obtained rewards
        saveUserReward(rewardName); // Save the reward to Firestore
        rewards.remove(index); // Remove the reward from the reward pool
        rewardGift = rewardName;
    } else {rewardGift = "No More rewards";
    }
    System.out.println(rewardGift);
}
```

Figure c5.2

The play () method then plays the gacha game by selecting a random reward from the reward pool adding it to the user's list of obtained rewards saving it to Firestore and removing it from the reward pool. If there are no more rewards in the pool, the method sets the rewardGift attribute to "No More rewards". The algorithm that’s being used here is the random algorithm which is in the nextInt(). This method generates a pseudorandom distributed integer value between 0 and the specified value which In this case the random.nextInt(rewards.size()) generates a random index between 0 and rewards.size()-1 which is used to select a random reward from the rewards list. The algorithm ensures that each reward has an equal chance of being selected and that the selection is independent of previous selections.

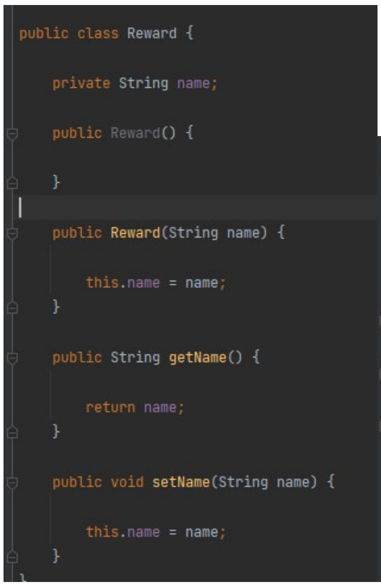


```
    // Save the user's reward to Firestore
    private void saveUserReward(String rewardName) {
        // Create a map with the reward data
        Map<String, Object> rewardData = new HashMap<>();
        // Add the reward name to the map with the key "reward"
        rewardData.put("reward", rewardName);

        // Save the reward to Firestore
        db.collection( collectionPath: "users_rewards").document(uid).collection( collectionPath: "rewards").add(rewardData)
                .addOnSuccessListener(new OnSuccessListener<DocumentReference>() {
                    @Override
                    public void onSuccess(DocumentReference documentReference) {
//                      System.out.println("Reward " + rewardName + " saved to Firestore.");
                    }
                });
    }
```

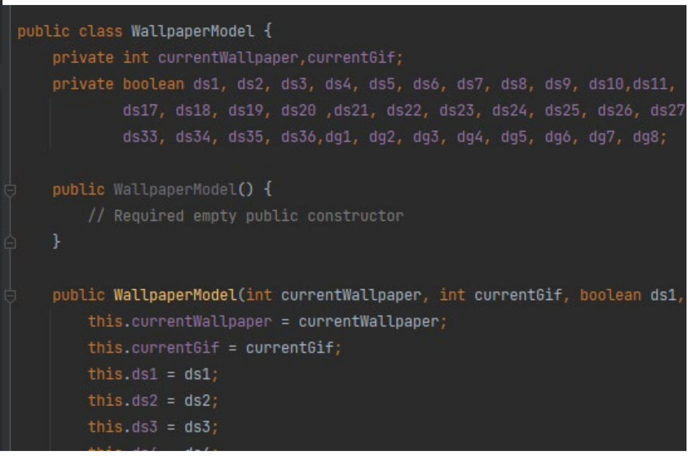
Figure c5.3

The saveUserReward() method then saves the user's reward to Firestore by creating a map with the reward data and adding it to the "users_rewards" collection. The method also logs a message to the console when the reward is successfully saved. The algorithm being used in the saveUserReward() method is a hash map. A HashMap is being used to create a map with the reward data where the reward name is added to the map with the key "reward". The HashMap is a data structure that stores key-value pairs and allows quick access to the values based on their keys. In my code the key "reward" is associated with the value rewardName and the resulting key-value pair is stored in the rewardData map. The HashMap is then used to save the reward data to Firestore.

The getRewardGift() method then returns the name of the reward obtained in the last play() call. This method is used to display the reward to the user.



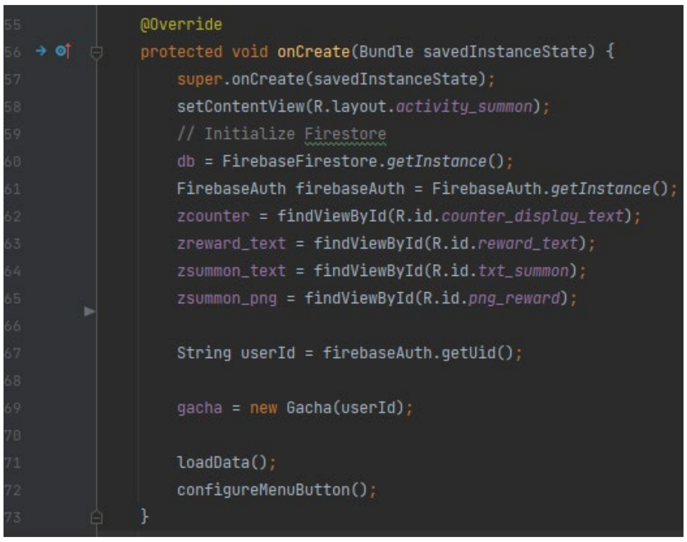
```java
public class Reward {

    private String name;

    public Reward() {

    }

    public Reward(String name) {

        this.name = name;
    }

    public String getName() {

        return name;
    }

    public void setName(String name) {

        this.name = name;
    }
```



```java
public class WallpaperModel {
    private int currentWallpaper,currentGif;
    private boolean ds1, ds2, ds3, ds4, ds5, ds6, ds7, ds8, ds9, ds10,ds11,
            ds17, ds18, ds19, ds20 ,ds21, ds22, ds23, ds24, ds25, ds26, ds27
            ds33, ds34, ds35, ds36,dg1, dg2, dg3, dg4, dg5, dg6, dg7, dg8;

    public WallpaperModel() {
        // Required empty public constructor
    }

    public WallpaperModel(int currentWallpaper, int currentGif, boolean ds1,
        this.currentWallpaper = currentWallpaper;
        this.currentGif = currentGif;
        this.ds1 = ds1;
        this.ds2 = ds2;
        this.ds3 = ds3;
```

Figure c5.4 Figure c5.5

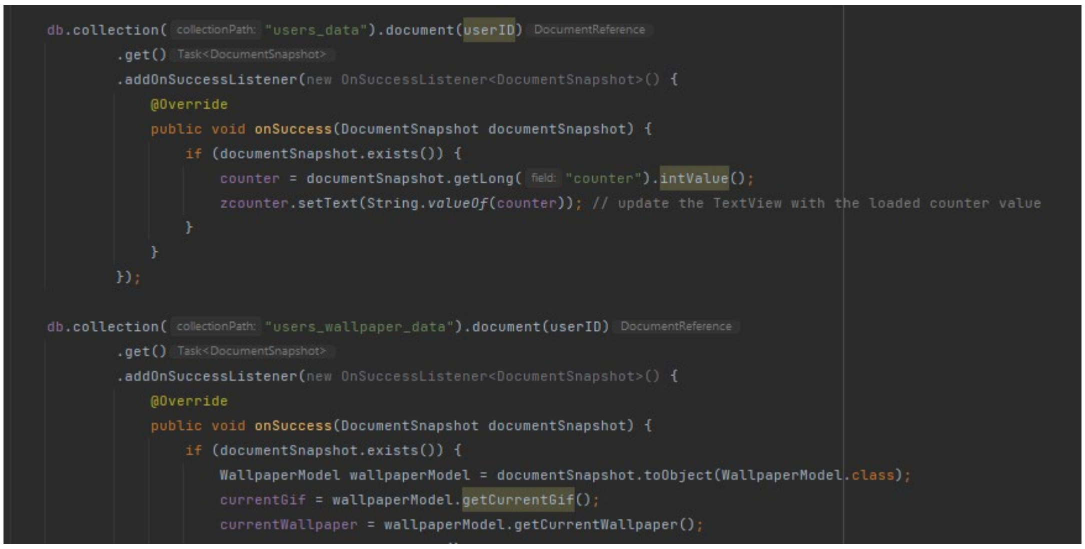
The reward class is used in gacha to represent the rewards that can be obtained by the user. It provides a simple way to store and retrieve the name of the reward which can be used to display the reward to the user or save it to a database. This is the same for the wallpaper model where it represents a wallpaper object with various variables like the current wallpaper, current gif and the Booleans for 36 different attributes. It has a default constructor and a parameterized constructor that initializes all the properties of the wallpaper object. It also has getter methods for all the properties to access them.



```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_summon);
    // Initialize Firestore
    db = FirebaseFirestore.getInstance();
    FirebaseAuth firebaseAuth = FirebaseAuth.getInstance();
    zcounter = findViewById(R.id.counter_display_text);
    zreward_text = findViewById(R.id.reward_text);
    zsummon_text = findViewById(R.id.txt_summon);
    zsummon_png = findViewById(R.id.png_reward);

    String userId = firebaseAuth.getUid();

    gacha = new Gacha(userId);

    loadData();
    configureMenuButton();
}
```

Figure c5.6

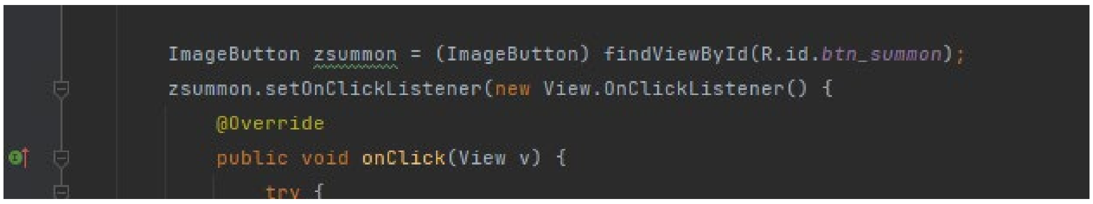
The Summon activity is responsible for handling the logic and behaviour of the Summon feature in the app. In the onCreate() method it initializes the activity and sets the content view to the activity_summon layout. It also initializes the FirebaseFirestore instance and FirebaseAuth instance and sets up the ImageButton and TextView objects. It then creates a new Gacha object with the userID obtained from FirebaseAuth. Finally it loads the data from the Firestore database using the loadData() method and sets up the menu button using the configureMenuButton() method.



```
db.collection( collectionPath: "users_data").document(userID) DocumentReference
        .get() Task<DocumentSnapshot>
        .addOnSuccessListener(new OnSuccessListener<DocumentSnapshot>() {
            @Override
            public void onSuccess(DocumentSnapshot documentSnapshot) {
                if (documentSnapshot.exists()) {
                    counter = documentSnapshot.getLong( field: "counter").intValue();
                    zcounter.setText(String.valueOf(counter)); // update the TextView with the loaded counter value
                }
            }
        });

db.collection( collectionPath: "users_wallpaper_data").document(userID) DocumentReference
        .get() Task<DocumentSnapshot>
        .addOnSuccessListener(new OnSuccessListener<DocumentSnapshot>() {
            @Override
            public void onSuccess(DocumentSnapshot documentSnapshot) {
                if (documentSnapshot.exists()) {
                    WallpaperModel wallpaperModel = documentSnapshot.toObject(WallpaperModel.class);
                    currentGif = wallpaperModel.getCurrentGif();
                    currentWallpaper = wallpaperModel.getCurrentWallpaper();
```

Figure c5.7

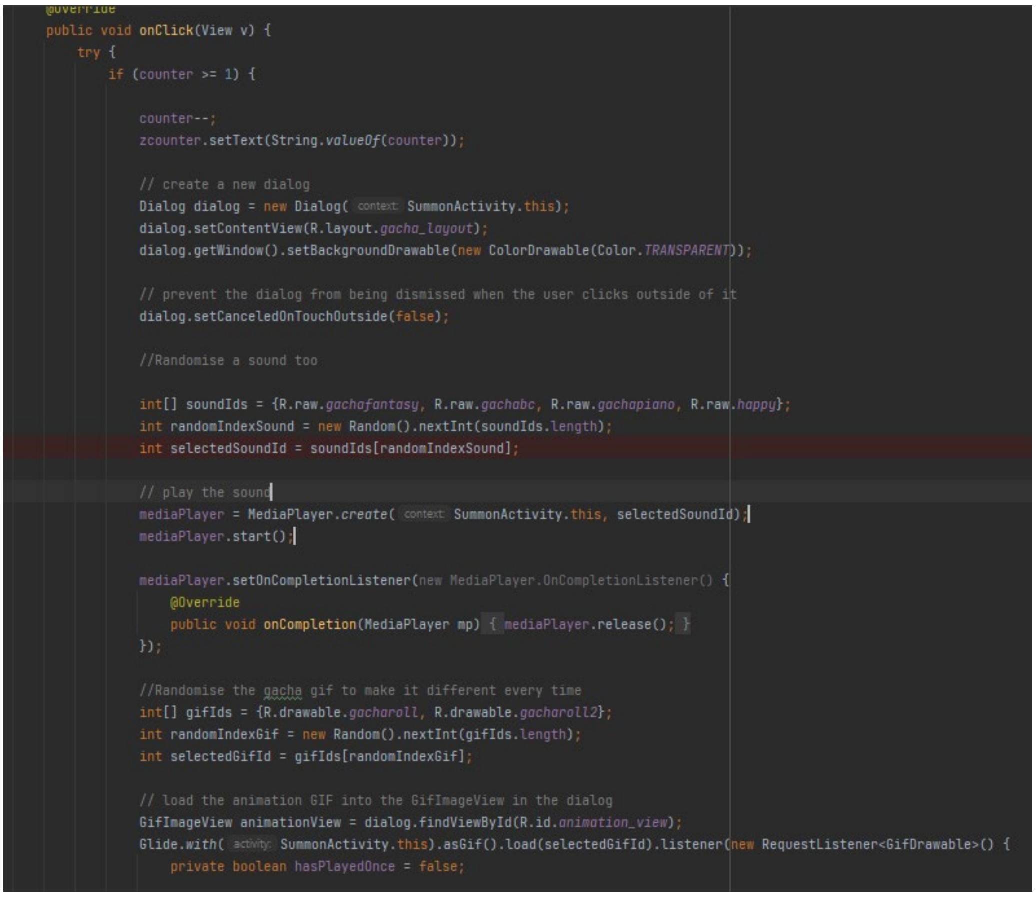
The method then retrieves data from two different Firestore collections: "users_data" and "users_wallpaper_data". For the "users_data" collection, the method retrieves the user's counter value which is stored as a long integer and updates the corresponding TextView to display this value while the "users_wallpaper_data" collection, the method retrieves a WallpaperModel object that contains various fields related to the user's selected wallpaper and related settings. These fields include currentGif, currentWallpaper and boolean values indicating whether various unlockable wallpapers (ds1 through ds35 and dg1 through dg8) have been unlocked. The method retrieves these fields from the WallpaperModel object and assigns the values to corresponding variables in the Summon activity. These variables are then used to display the user's selected wallpaper and unlockable wallpapers in the activity.



```
ImageButton zsummon = (ImageButton) findViewById(R.id.btn_summon);
zsummon.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        try {
```

Figure c5.8

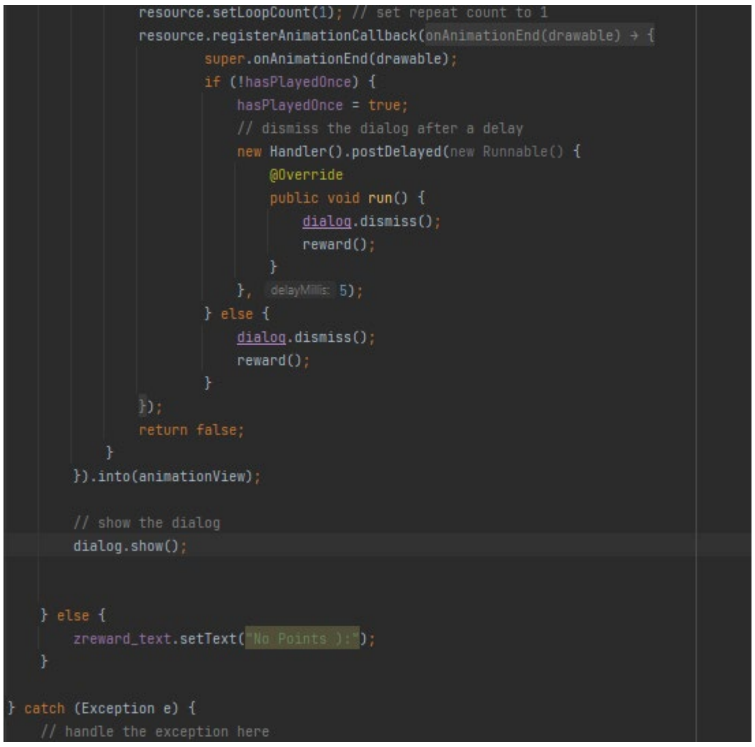
In the configureMenuButton() method it contains the summon button. This is used to initiate a summon in the Gacha. When the user clicks the button the onClick method is triggered which performs several steps to initiate the gacha summon.



```java
@Override
public void onClick(View v) {
    try {
        if (counter >= 1) {

            counter--;
            zcounter.setText(String.valueOf(counter));

            // create a new dialog
            Dialog dialog = new Dialog( context: SummonActivity.this);
            dialog.setContentView(R.layout.gacha_layout);
            dialog.getWindow().setBackgroundDrawable(new ColorDrawable(Color.TRANSPARENT));

            // prevent the dialog from being dismissed when the user clicks outside of it
            dialog.setCanceledOnTouchOutside(false);

            //Randomise a sound too

            int[] soundIds = {R.raw.gachafantasy, R.raw.gachabc, R.raw.gachapiano, R.raw.happy};
            int randomIndexSound = new Random().nextInt(soundIds.length);
            int selectedSoundId = soundIds[randomIndexSound];

            // play the sound
            mediaPlayer = MediaPlayer.create( context: SummonActivity.this, selectedSoundId);
            mediaPlayer.start();

            mediaPlayer.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
                @Override
                public void onCompletion(MediaPlayer mp) { mediaPlayer.release(); }
            });

            //Randomise the gacha gif to make it different every time
            int[] gifIds = {R.drawable.gacharoll, R.drawable.gacharoll2};
            int randomIndexGif = new Random().nextInt(gifIds.length);
            int selectedGifId = gifIds[randomIndexGif];

            // load the animation GIF into the GifImageView in the dialog
            GifImageView animationView = dialog.findViewById(R.id.animation_view);
            Glide.with( activity: SummonActivity.this).asGif().load(selectedGifId).listener(new RequestListener<GifDrawable>() {
                private boolean hasPlayedOnce = false;
```

Figure c5.9

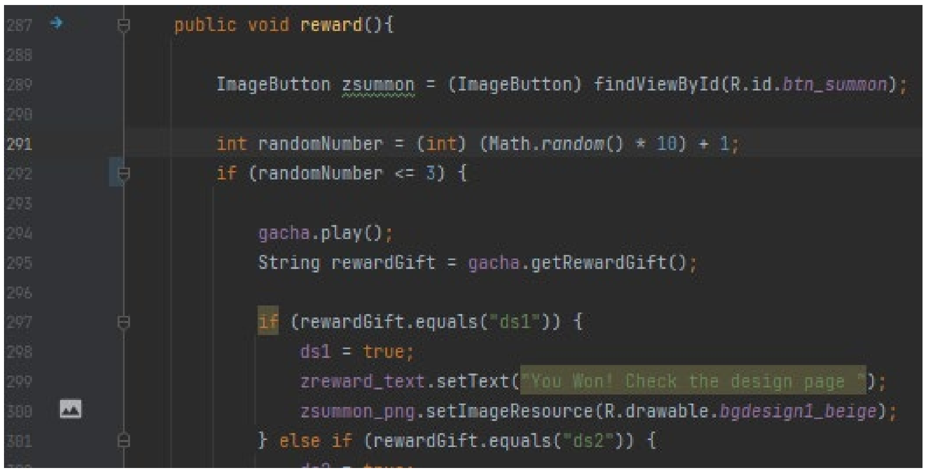
Firstly, the method decreases the counter variable by 1 if the user has more than 1 point. This updates the zcounter text view to reflect the new value. The counter variable represents the number of points the user has available to spend on gacha summons. The method then creates a new dialog window to display the gacha animation. The dialog window is created using the AlertDialog.Builder class which allows me to use my custom dialog windows with custom layouts and content. The dialog window is set to be cancellable which means that the user can dismiss it by clicking outside of the window or pressing the back button. Once the dialog window is created the method randomly selects a sound effect from an array of sound effect resources and plays it using a MediaPlayer object. The sound effect is played to add to the excitement and anticipation of the gacha summon. The method then randomly selects a gacha animation GIF from an array of GIF resources and displays it in a GifImageView using the Glide library. The GifImageView is a custom view that is designed to display animated GIFs and it is included in the app using the Glide library. The GIF is displayed in the dialog window to show the user the gacha summon animation. The GIF is set to play only once using the setLoopCount method of the GifDrawable object which ensures that the animation plays only once and then stops. An animation callback is also registered to dismiss the dialog window and display the gacha reward after the animation has played. The animation callback is triggered when the GIF animation ends, and it dismisses the dialog window and displays the gacha reward to the user.



```java
                resource.setLoopCount(1); // set repeat count to 1
                resource.registerAnimationCallback(onAnimationEnd(drawable) → {
                        super.onAnimationEnd(drawable);
                        if (!hasPlayedOnce) {
                            hasPlayedOnce = true;
                            // dismiss the dialog after a delay
                            new Handler().postDelayed(new Runnable() {
                                @Override
                                public void run() {
                                    dialog.dismiss();
                                    reward();
                                }
                            },  delayMillis: 5);
                        } else {
                            dialog.dismiss();
                            reward();
                        }
                });
                return false;
            }
        }).into(animationView);

        // show the dialog
        dialog.show();


    } else {
        zreward_text.setText("No Points ):");
    }

} catch (Exception e) {
    // handle the exception here
```

Figure c6.0

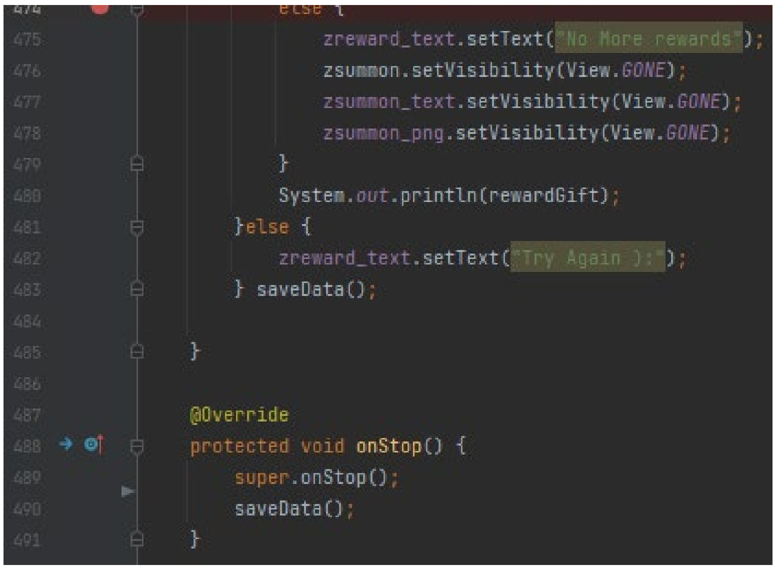
Finally, the dialog window is displayed to the user, and they can watch the gacha summon animation and wait for the reward to be revealed. But if the counter variable is less than 1, the zreward_text text view is updated to display a message indicating that the user has no points to spend on a gacha summon.



```java
    public void reward(){

        ImageButton zsummon = (ImageButton) findViewById(R.id.btn_summon);

        int randomNumber = (int) (Math.random() * 10) + 1;
        if (randomNumber <= 3) {

            gacha.play();
            String rewardGift = gacha.getRewardGift();

            if (rewardGift.equals("ds1")) {
                ds1 = true;
                zreward_text.setText("You Won! Check the design page ");
                zsummon_png.setImageResource(R.drawable.bgdesign1_beige);
            } else if (rewardGift.equals("ds2")) {
```

Figure c6.1

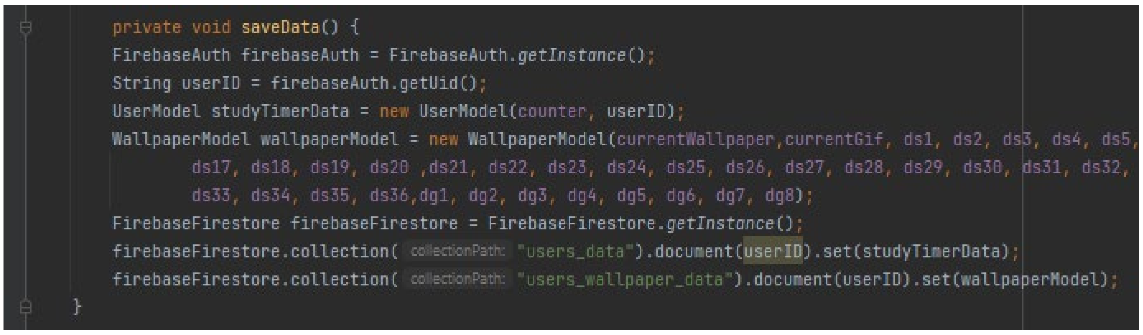
The reward method is responsible for determining whether the user will receive a reward or not. It uses the Math.random() method to generate a random number between 1 and 10. If the number is less than or equal to 3 the user will receive a reward. This makes it a 30% chance of the user to receive a reward. If the user does receive a reward the image displayed will depend on the value of the rewardGift variable which is randomly generated by the getRewardGift() method of the gacha object and the play() method is what gives the user a random reward if they do win. If the rewardGift variable matches one of the values the corresponding image is displayed and a boolean variable is set to true to indicate that the user has won that reward. The zreward_text text view is updated to display a message indicating that the user has won and to check the design page.



```
                else {
                    zreward_text.setText("No More rewards");
                    zsummon.setVisibility(View.GONE);
                    zsummon_text.setVisibility(View.GONE);
                    zsummon_png.setVisibility(View.GONE);
                }
                System.out.println(rewardGift);
            }else {
                zreward_text.setText("Try Again ):");
            } saveData();

        }

        @Override
        protected void onStop() {
            super.onStop();
            saveData();
        }
```

Figure c6.2

If the rewardGift variable does not match any of the predefined values the zreward_text text view is updated to display a message indicating that there are no more rewards available. The zsummon button, zsummon_text text view and zsummon_png image view are hidden from the user. If the user does not receive a reward the zreward_text text view is updated to display a message indicating that they should try again. The saveData() method is called to save the user's progress. If the user closes the application or it stops, the saveData() method is called to make sure that the data is being saved too.



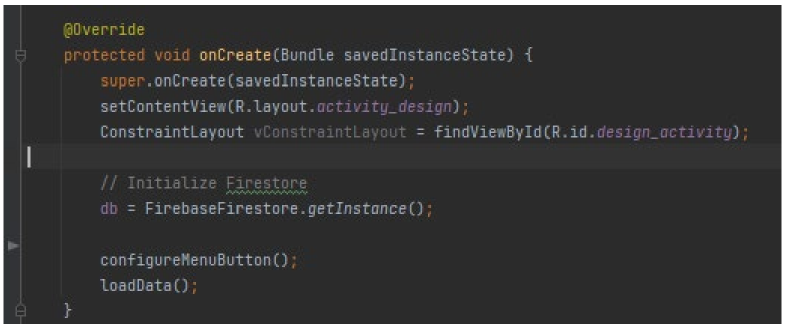
```
    private void saveData() {
    FirebaseAuth firebaseAuth = FirebaseAuth.getInstance();
    String userID = firebaseAuth.getUid();
    UserModel studyTimerData = new UserModel(counter, userID);
    WallpaperModel wallpaperModel = new WallpaperModel(currentWallpaper,currentGif, ds1, ds2, ds3, ds4, ds5,
            ds17, ds18, ds19, ds20 ,ds21, ds22, ds23, ds24, ds25, ds26, ds27, ds28, ds29, ds30, ds31, ds32,
            ds33, ds34, ds35, ds36,dg1, dg2, dg3, dg4, dg5, dg6, dg7, dg8);
    FirebaseFirestore firebaseFirestore = FirebaseFirestore.getInstance();
    firebaseFirestore.collection( collectionPath: "users_data").document(userID).set(studyTimerData);
    firebaseFirestore.collection( collectionPath: "users_wallpaper_data").document(userID).set(wallpaperModel);
}
```

Figure c6.3

The saveData() method gets the current user's ID using the FirebaseAuth class. This ID is used to identify the user in the database and ensure that their data is saved to the correct document. Next the method creates a new UserModel object and a new WallpaperModel object. The UserModel object contains the user's study timer data which includes the counter variable that represents the number of points the user has.

The WallpaperModel object contains the user's wallpaper settings which include the current images and GIF that are used.

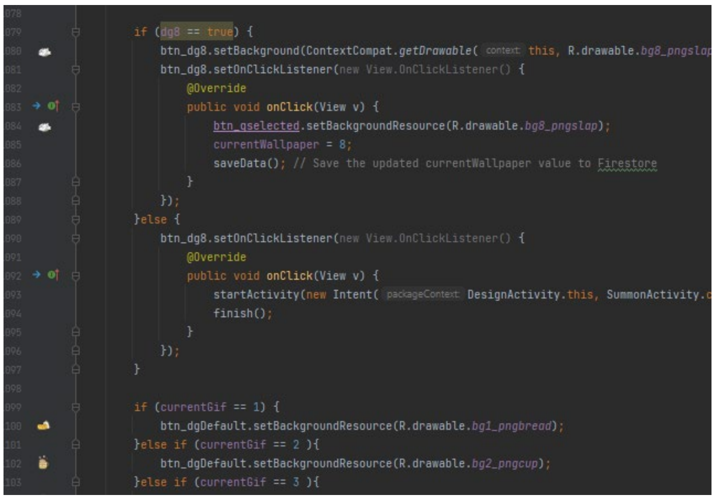
After creating the UserModel and WallpaperModel objects the method gets a reference to the Firebase Firestore database using the FirebaseFirestore class. It then uses this reference to save the UserModel object to a document in the users_data collection and the WallpaperModel object to a document in the users_wallpaper_data collection. The document ID for each document is set to the user's ID which ensures that the data is saved to the correct document.



```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_design);
    ConstraintLayout vConstraintLayout = findViewById(R.id.design_activity);

    // Initialize Firestore
    db = FirebaseFirestore.getInstance();

    configureMenuButton();
    loadData();
}
```

Figure c6.4

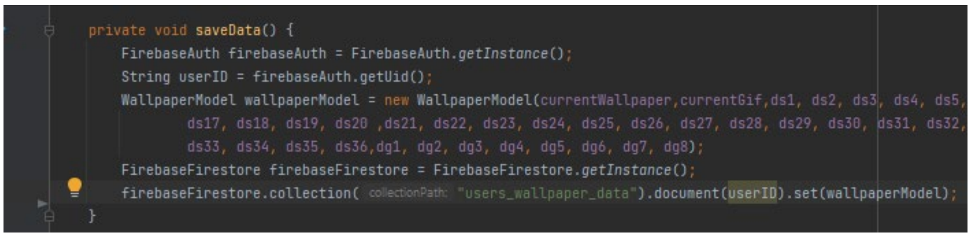
The DesignActivity class then is the main class that is responsible for allowing the user to use the rewards they have received in the SummonActivity class. The class contains boolean variables for each reward that the user can receive such as ds1, ds2, dg1, dg2, etc. These variables are used to determine which reward the user has received and can use in the DesignActivity. In the onCreate method it initializes the Firestore database and loads the data for the user.



```java
if (dg8 == true) {
    btn_dg8.setBackground(ContextCompat.getDrawable( context this, R.drawable.bg8_pngslap
    btn_dg8.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            btn_gselected.setBackgroundResource(R.drawable.bg8_pngslap);
            currentWallpaper = 8;
            saveData(); // Save the updated currentWallpaper value to Firestore
        }
    });
}else {
    btn_dg8.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            startActivity(new Intent( packageContext DesignActivity.this, SummonActivity.c
            finish();
        }
    });
}

if (currentGif == 1) {
    btn_dgDefault.setBackgroundResource(R.drawable.bg1_pngbread);
}else if (currentGif == 2 ){
    btn_dgDefault.setBackgroundResource(R.drawable.bg2_pngcup);
}else if (currentGif == 3 ){
```

Figure c6.5

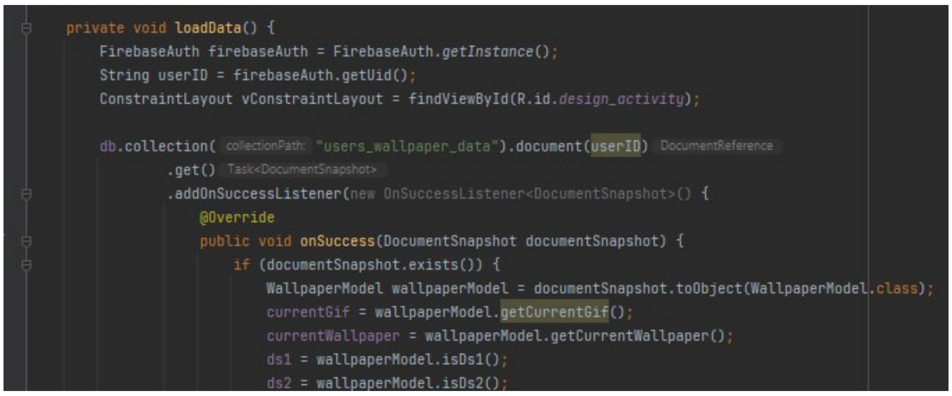
The configureMenuButton method sets up the design buttons for the DesignActivity where it will assign currentgif or currentwallpaper based on what the user has chosen. If the user doesn't have the reward and clicks on the button it will direct them to the summonActivity class.



```
private void saveData() {
    FirebaseAuth firebaseAuth = FirebaseAuth.getInstance();
    String userID = firebaseAuth.getUid();
    WallpaperModel wallpaperModel = new WallpaperModel(currentWallpaper,currentGif,ds1, ds2, ds3, ds4, ds5,
            ds17, ds18, ds19, ds20 ,ds21, ds22, ds23, ds24, ds25, ds26, ds27, ds28, ds29, ds30, ds31, ds32,
            ds33, ds34, ds35, ds36,dg1, dg2, dg3, dg4, dg5, dg6, dg7, dg8);
    FirebaseFirestore firebaseFirestore = FirebaseFirestore.getInstance();
    firebaseFirestore.collection( collectionPath: "users_wallpaper_data").document(userID).set(wallpaperModel);
}
```

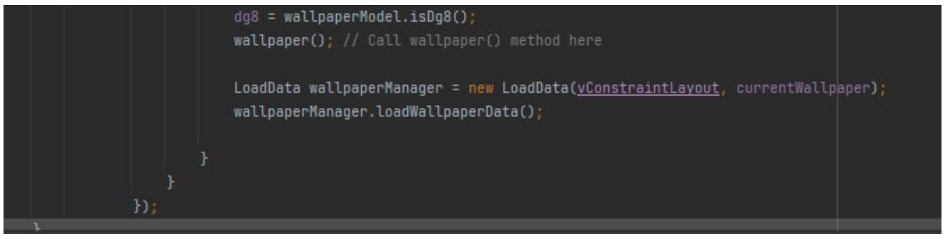
Figure c6.6

When the user chooses a wallpaper or a gif the saveData() method updates the currentWallpaper or currentGif value in the Firestore document for the current user whenever the user selects a new wallpaper or GIF. This ensures that the users selected wallpaper or GIF is saved and can be retrieved the next time the user opens the app.



```
private void loadData() {
    FirebaseAuth firebaseAuth = FirebaseAuth.getInstance();
    String userID = firebaseAuth.getUid();
    ConstraintLayout vConstraintLayout = findViewById(R.id.design_activity);

    db.collection( collectionPath: "users_wallpaper_data").document(userID) DocumentReference
            .get() Task<DocumentSnapshot>
            .addOnSuccessListener(new OnSuccessListener<DocumentSnapshot>() {
                @Override
                public void onSuccess(DocumentSnapshot documentSnapshot) {
                    if (documentSnapshot.exists()) {
                        WallpaperModel wallpaperModel = documentSnapshot.toObject(WallpaperModel.class);
                        currentGif = wallpaperModel.getCurrentGif();
                        currentWallpaper = wallpaperModel.getCurrentWallpaper();
                        ds1 = wallpaperModel.isDs1();
                        ds2 = wallpaperModel.isDs2();
```

Figure c6.7



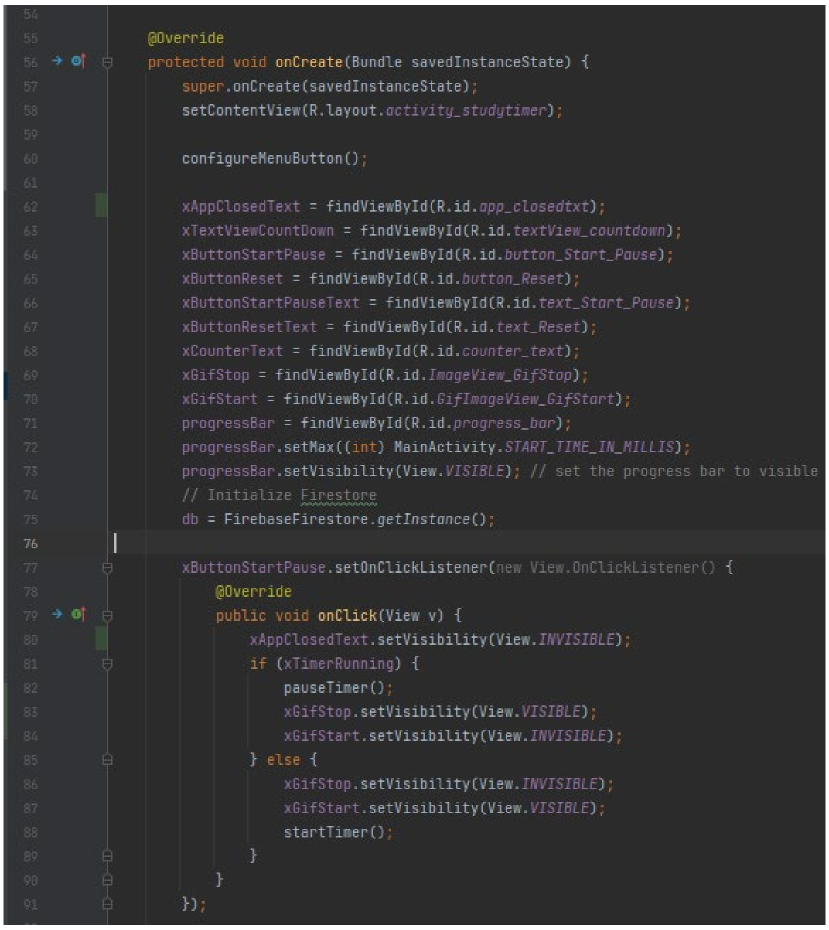
```
                        dg8 = wallpaperModel.isDg8();
                        wallpaper(); // Call wallpaper() method here

                        LoadData wallpaperManager = new LoadData(vConstraintLayout, currentWallpaper);
                        wallpaperManager.loadWallpaperData();

                    }
                }
            });
```

Figure c6.8

The loadData() method is responsible for loading the user's data from the Firestore database. This method retrieves the user's data from the users_wallpaper_data collection in Firestore and sets the values of the ds1 through ds36 and dg1 through dg8 boolean variables and the currentWallpaper and currentGif integer variables. These variables are used to determine which rewards the user has received and which wallpaper and GIF the user is currently using.

Timer Implementation

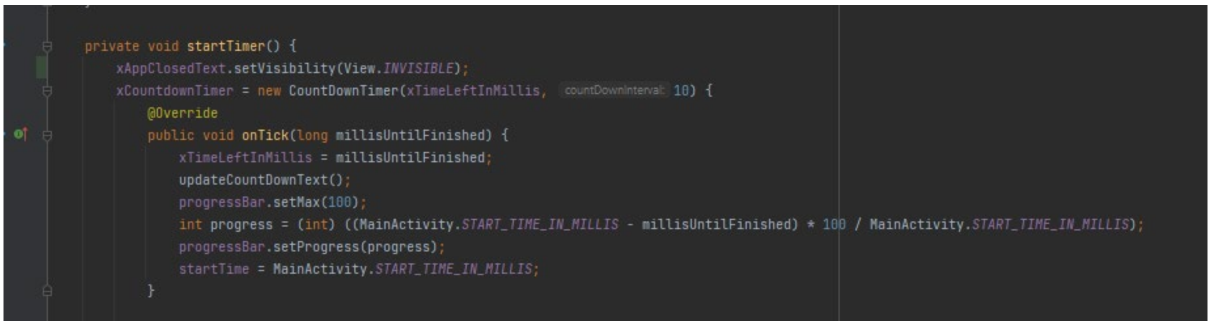
The StudyTimerActivity class is responsible for implementing the functionality of a study timer feature. (code, n.d.)



```java
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_studytimer);

        configureMenuButton();

        xAppClosedText = findViewById(R.id.app_closedtxt);
        xTextViewCountDown = findViewById(R.id.textView_countdown);
        xButtonStartPause = findViewById(R.id.button_Start_Pause);
        xButtonReset = findViewById(R.id.button_Reset);
        xButtonStartPauseText = findViewById(R.id.text_Start_Pause);
        xButtonResetText = findViewById(R.id.text_Reset);
        xCounterText = findViewById(R.id.counter_text);
        xGifStop = findViewById(R.id.ImageView_GifStop);
        xGifStart = findViewById(R.id.GifImageView_GifStart);
        progressBar = findViewById(R.id.progress_bar);
        progressBar.setMax((int) MainActivity.START_TIME_IN_MILLIS);
        progressBar.setVisibility(View.VISIBLE); // set the progress bar to visible
        // Initialize Firestore
        db = FirebaseFirestore.getInstance();

        xButtonStartPause.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                xAppClosedText.setVisibility(View.INVISIBLE);
                if (xTimerRunning) {
                    pauseTimer();
                    xGifStop.setVisibility(View.VISIBLE);
                    xGifStart.setVisibility(View.INVISIBLE);
                } else {
                    xGifStop.setVisibility(View.INVISIBLE);
                    xGifStart.setVisibility(View.VISIBLE);
                    startTimer();
                }
            }
        });
```

Figure c6.9

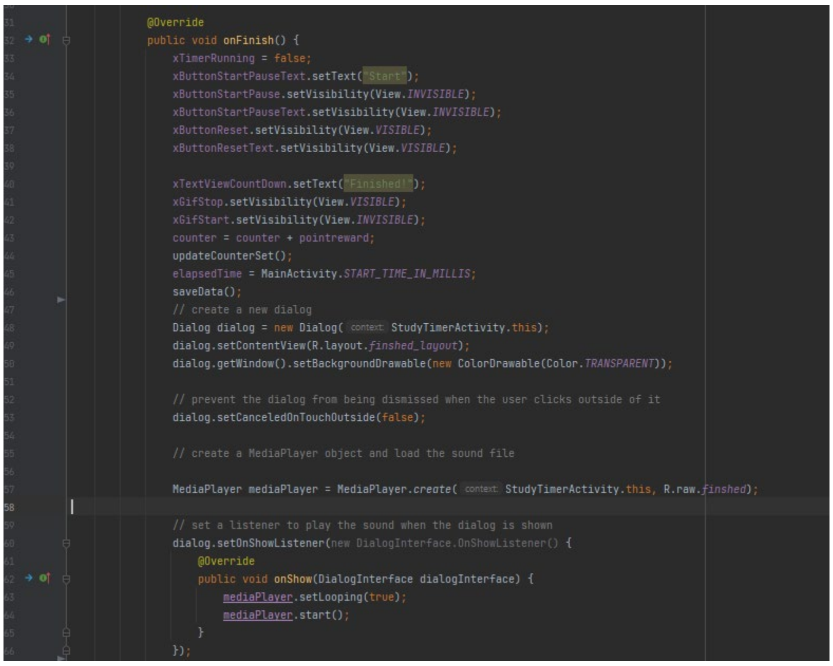
The onCreate() method initializes the UI components of the time It also sets up event listeners for the start/pause and reset buttons. Additionally, it loads user data and updates the UI accordingly.



```java
    private void startTimer() {
        xAppClosedText.setVisibility(View.INVISIBLE);
        xCountdownTimer = new CountDownTimer(xTimeLeftInMillis,  countDownInterval: 10) {
            @Override
            public void onTick(long millisUntilFinished) {
                xTimeLeftInMillis = millisUntilFinished;
                updateCountDownText();
                progressBar.setMax(100);
                int progress = (int) ((MainActivity.START_TIME_IN_MILLIS - millisUntilFinished) * 100 / MainActivity.START_TIME_IN_MILLIS);
                progressBar.setProgress(progress);
                startTime = MainActivity.START_TIME_IN_MILLIS;
            }
```

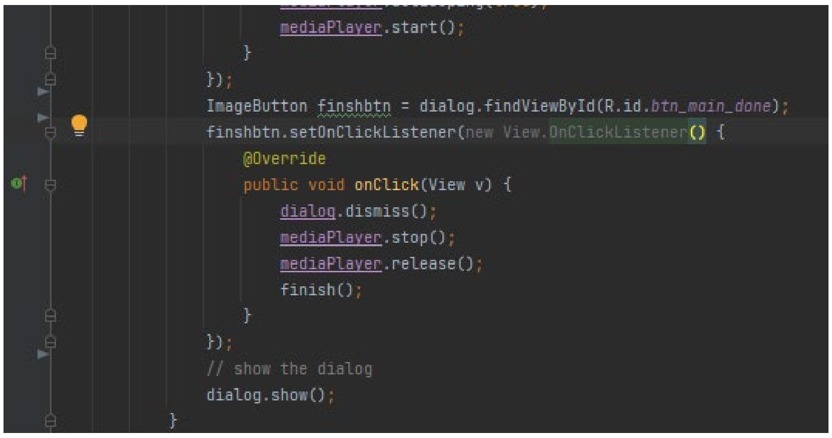
Figure c7.0

The startTimer() method is the main method of the StudyTimer class which initializes and starts the countdown timer. It sets the xTimeLeftInMillis to the default value and updates the UI components such as the countdown text and progress bar and starts the countdown timer using a CountDownTimer object. The onTick() method of the CountDownTimer object is called every time the timer ticks updating the xTimeLeftInMillis and UI components accordingly.



```java
            @Override
            public void onFinish() {
                xTimerRunning = false;
                xButtonStartPauseText.setText("Start");
                xButtonStartPause.setVisibility(View.INVISIBLE);
                xButtonStartPauseText.setVisibility(View.INVISIBLE);
                xButtonReset.setVisibility(View.VISIBLE);
                xButtonResetText.setVisibility(View.VISIBLE);

                xTextViewCountDown.setText("Finished!");
                xGifStop.setVisibility(View.VISIBLE);
                xGifStart.setVisibility(View.INVISIBLE);
                counter = counter + pointreward;
                updateCounterSet();
                elapsedTime = MainActivity.START_TIME_IN_MILLIS;
                saveData();
                // create a new dialog
                Dialog dialog = new Dialog( context: StudyTimerActivity.this);
                dialog.setContentView(R.layout.finshed_layout);
                dialog.getWindow().setBackgroundDrawable(new ColorDrawable(Color.TRANSPARENT));

                // prevent the dialog from being dismissed when the user clicks outside of it
                dialog.setCanceledOnTouchOutside(false);

                // create a MediaPlayer object and load the sound file

                MediaPlayer mediaPlayer = MediaPlayer.create( context: StudyTimerActivity.this, R.raw.finshed);

                // set a listener to play the sound when the dialog is shown
                dialog.setOnShowListener(new DialogInterface.OnShowListener() {
                    @Override
                    public void onShow(DialogInterface dialogInterface) {
                        mediaPlayer.setLooping(true);
                        mediaPlayer.start();
                    }
                });
```

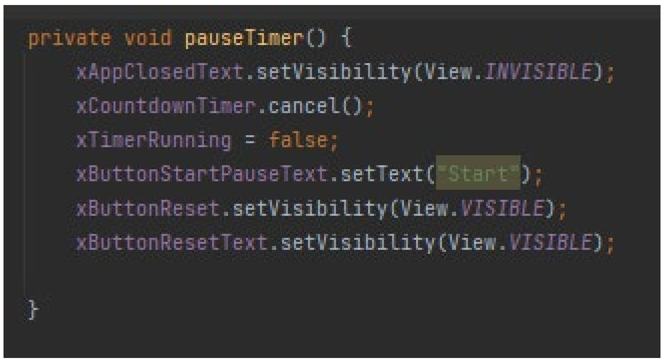
Figure c7.1



```java
                        mediaPlayer.start();
                    }
                });
                ImageButton finshbtn = dialog.findViewById(R.id.btn_main_done);
                finshbtn.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        dialog.dismiss();
                        mediaPlayer.stop();
                        mediaPlayer.release();
                        finish();
                    }
                });
                // show the dialog
                dialog.show();
            }
```

Figure c7.2

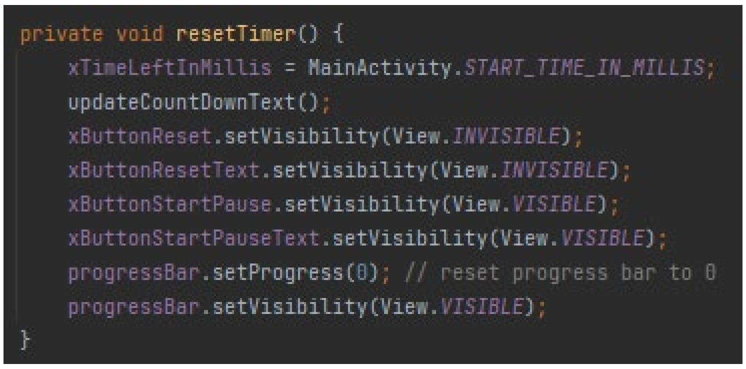
When the timer finishes the onFinish() method of the CountDownTimer is called which handles the event of the timer ending. The UI is updated to display the "Finished!" message. The xGifStop image is made visible while the xGifStart image is made invisible. The counter variable is incremented by the pointreward value and the updateCounterSet() method is called to update the TextView displaying the current point total. A custom Dialog object is created and shown which displays a congratulatory message and a button to finish the activity. A sound effect is played when the dialog is displayed. When the button is clicked the dialog is dismissed. The sound effect is stopped and released and the finish() method is called to close the activity.



```java
private void pauseTimer() {
    xAppClosedText.setVisibility(View.INVISIBLE);
    xCountdownTimer.cancel();
    xTimerRunning = false;
    xButtonStartPauseText.setText("Start");
    xButtonReset.setVisibility(View.VISIBLE);
    xButtonResetText.setVisibility(View.VISIBLE);

}
```

Figure c7.3

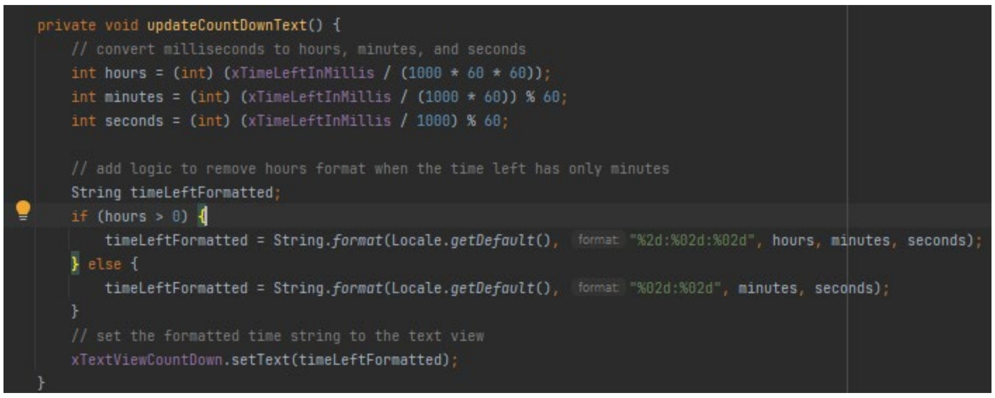
The pauseTimer() method is responsible for pausing the countdown timer by calling the cancel() method on the xCountdownTimer object. This allows the timer to be stopped and resumed at a later point in time. This method updates the UI components accordingly by setting the visibility of certain buttons and text views.



```java
private void resetTimer() {
    xTimeLeftInMillis = MainActivity.START_TIME_IN_MILLIS;
    updateCountDownText();
    xButtonReset.setVisibility(View.INVISIBLE);
    xButtonResetText.setVisibility(View.INVISIBLE);
    xButtonStartPause.setVisibility(View.VISIBLE);
    xButtonStartPauseText.setVisibility(View.VISIBLE);
    progressBar.setProgress(0); // reset progress bar to 0
    progressBar.setVisibility(View.VISIBLE);
}
```

Figure c7.4

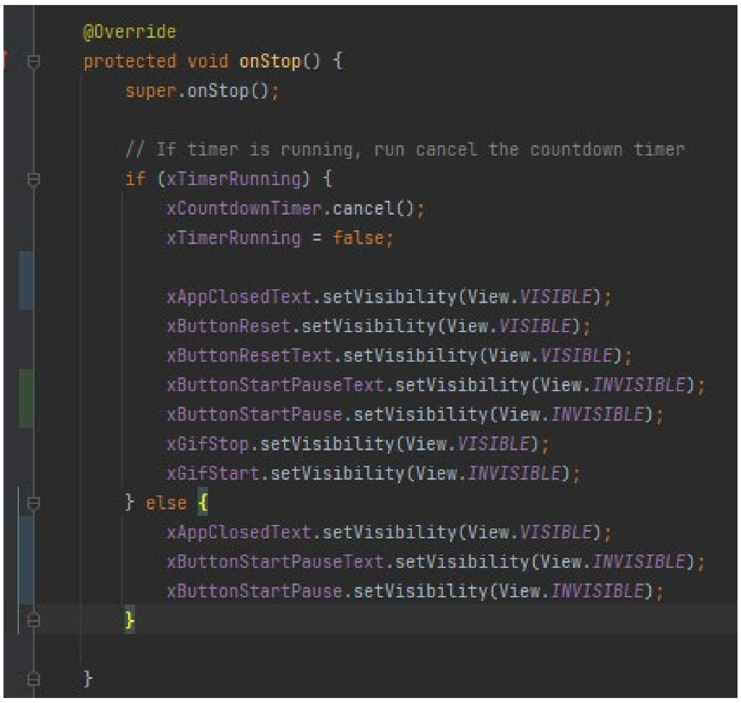
The resetTimer() method is responsible for resetting the countdown timer of the study session to the value of MainActivity.START_TIME_IN_MILLIS which is the initial time set for the study session in the MainActivity class. When this method is called, the remaining time is set to the initial time and the UI components are updated accordingly. The reset button and its text are set to invisible, while the start/pause button and its text are set to visible. The progress bar is also reset to 0 and set to visible.The updateCountDownText() method updates the text view that displays the remaining time on the timer.



```java
private void updateCountDownText() {
    // convert milliseconds to hours, minutes, and seconds
    int hours = (int) (xTimeLeftInMillis / (1000 * 60 * 60));
    int minutes = (int) (xTimeLeftInMillis / (1000 * 60)) % 60;
    int seconds = (int) (xTimeLeftInMillis / 1000) % 60;

    // add logic to remove hours format when the time left has only minutes
    String timeLeftFormatted;
    if (hours > 0) {
        timeLeftFormatted = String.format(Locale.getDefault(), format: "%2d:%02d:%02d", hours, minutes, seconds);
    } else {
        timeLeftFormatted = String.format(Locale.getDefault(), format: "%02d:%02d", minutes, seconds);
    }
    // set the formatted time string to the text view
    xTextViewCountDown.setText(timeLeftFormatted);
}
```

Figure c7.5

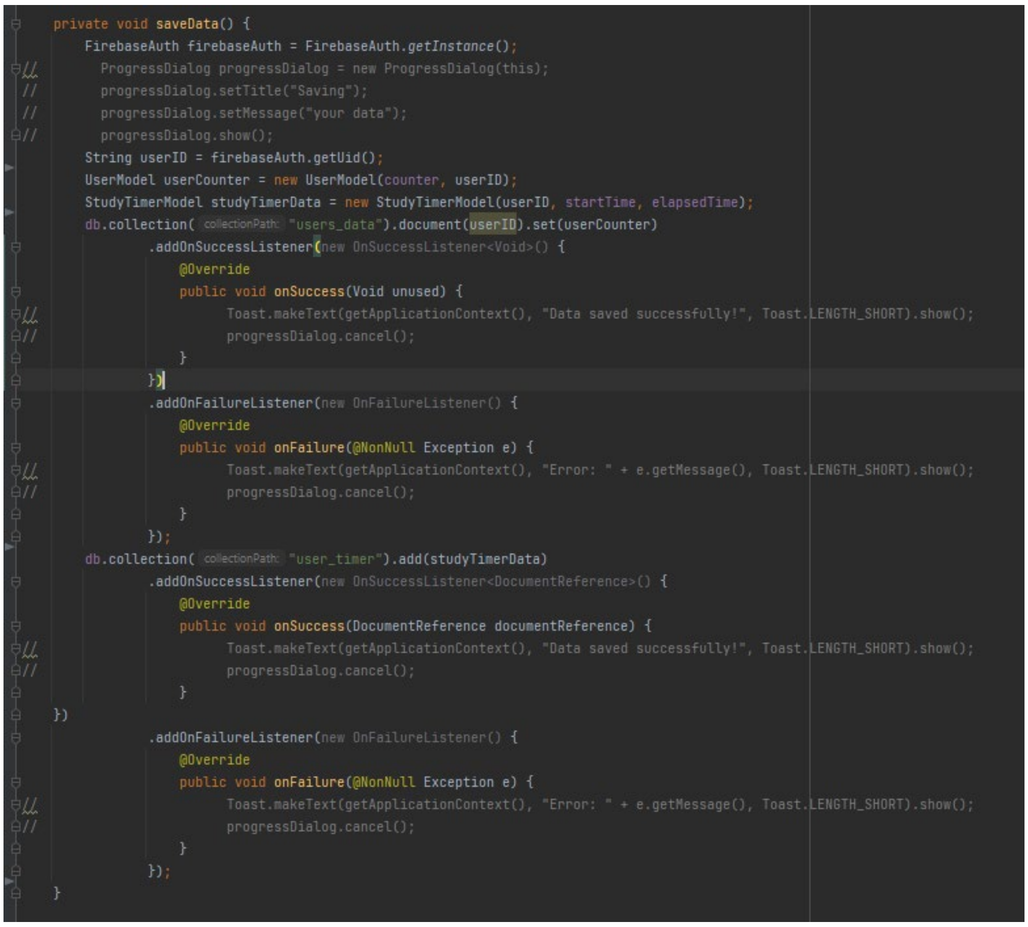
The updateCountDownText() method converts the remaining time in milliseconds to hours, minutes, and seconds and formats them into a user-friendly string. If the remaining time is less than an hour the hours format is removed to show only minutes and seconds. The resulting time string is then set to the TextView for the countdown timer display.



```
@Override
protected void onStop() {
    super.onStop();

    // If timer is running, run cancel the countdown timer
    if (xTimerRunning) {
        xCountdownTimer.cancel();
        xTimerRunning = false;

        xAppClosedText.setVisibility(View.VISIBLE);
        xButtonReset.setVisibility(View.VISIBLE);
        xButtonResetText.setVisibility(View.VISIBLE);
        xButtonStartPauseText.setVisibility(View.INVISIBLE);
        xButtonStartPause.setVisibility(View.INVISIBLE);
        xGifStop.setVisibility(View.VISIBLE);
        xGifStart.setVisibility(View.INVISIBLE);
    } else {
        xAppClosedText.setVisibility(View.VISIBLE);
        xButtonStartPauseText.setVisibility(View.INVISIBLE);
        xButtonStartPause.setVisibility(View.INVISIBLE);
    }

}
```

Figure c7.6

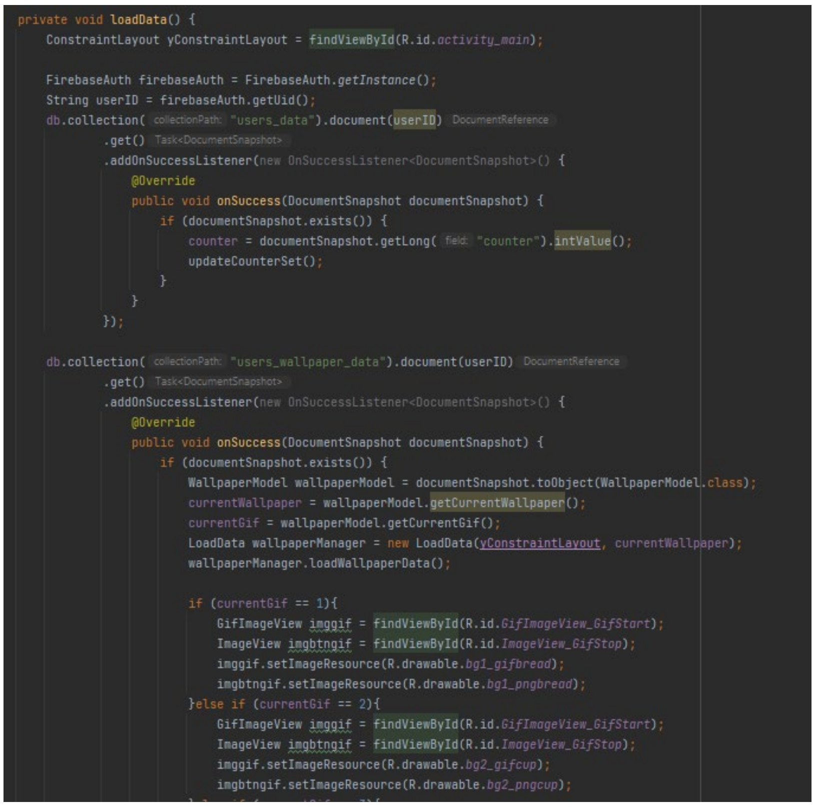
The onStop() method is called when the activity is stopped such as when the user navigates away from the timer screen. It cancels the countdown timer if it is running and updates the UI components accordingly.



```java
private void saveData() {
    FirebaseAuth firebaseAuth = FirebaseAuth.getInstance();
//      ProgressDialog progressDialog = new ProgressDialog(this);
//      progressDialog.setTitle("Saving");
//      progressDialog.setMessage("your data");
//      progressDialog.show();
    String userID = firebaseAuth.getUid();
    UserModel userCounter = new UserModel(counter, userID);
    StudyTimerModel studyTimerData = new StudyTimerModel(userID, startTime, elapsedTime);
    db.collection( collectionPath: "users_data").document(userID).set(userCounter)
            .addOnSuccessListener(new OnSuccessListener<Void>() {
                @Override
                public void onSuccess(Void unused) {
//                      Toast.makeText(getApplicationContext(), "Data saved successfully!", Toast.LENGTH_SHORT).show();
//                      progressDialog.cancel();
                }
            })
            .addOnFailureListener(new OnFailureListener() {
                @Override
                public void onFailure(@NonNull Exception e) {
//                      Toast.makeText(getApplicationContext(), "Error: " + e.getMessage(), Toast.LENGTH_SHORT).show();
//                      progressDialog.cancel();
                }
            });
    db.collection( collectionPath: "user_timer").add(studyTimerData)
            .addOnSuccessListener(new OnSuccessListener<DocumentReference>() {
                @Override
                public void onSuccess(DocumentReference documentReference) {
//                      Toast.makeText(getApplicationContext(), "Data saved successfully!", Toast.LENGTH_SHORT).show();
//                      progressDialog.cancel();
                }
})
            .addOnFailureListener(new OnFailureListener() {
                @Override
                public void onFailure(@NonNull Exception e) {
//                      Toast.makeText(getApplicationContext(), "Error: " + e.getMessage(), Toast.LENGTH_SHORT).show();
//                      progressDialog.cancel();
                }
            });
}
```

Figure c7.7

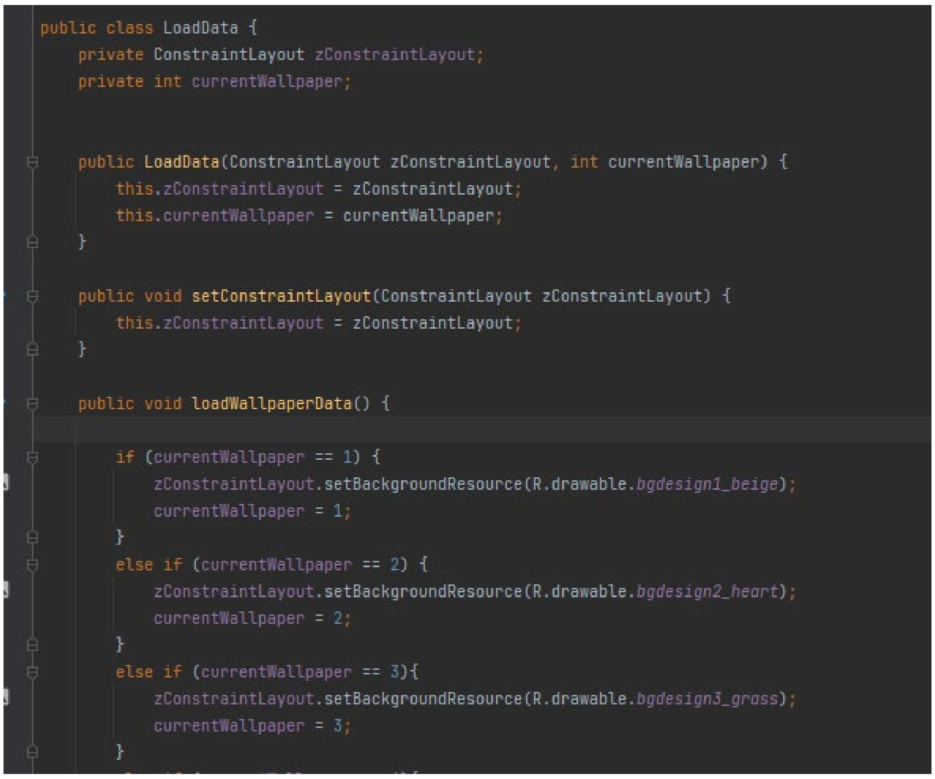
The class also includes methods for updating the counter saving, and loading user data, and configuring the menu button. The saveData() method is responsible for saving the user's data to the Firestore database. The method creates instances of UserModel and StudyTimerModel which hold the user's counter point and study timer data respectively. The Firebase authentication instance is obtained to get the user ID and the two instances are then saved to the Firestore collection "users_data" and "user_timer" respectively. Success and failure listeners are added to the save operations to handle any exceptions that may occur.



```
private void loadData() {
    ConstraintLayout yConstraintLayout = findViewById(R.id.activity_main);

    FirebaseAuth firebaseAuth = FirebaseAuth.getInstance();
    String userID = firebaseAuth.getUid();
    db.collection( collectionPath: "users_data").document(userID) DocumentReference
            .get() Task<DocumentSnapshot>
            .addOnSuccessListener(new OnSuccessListener<DocumentSnapshot>() {
                @Override
                public void onSuccess(DocumentSnapshot documentSnapshot) {
                    if (documentSnapshot.exists()) {
                        counter = documentSnapshot.getLong( field: "counter").intValue();
                        updateCounterSet();
                    }
                }
            });

    db.collection( collectionPath: "users_wallpaper_data").document(userID) DocumentReference
            .get() Task<DocumentSnapshot>
            .addOnSuccessListener(new OnSuccessListener<DocumentSnapshot>() {
                @Override
                public void onSuccess(DocumentSnapshot documentSnapshot) {
                    if (documentSnapshot.exists()) {
                        WallpaperModel wallpaperModel = documentSnapshot.toObject(WallpaperModel.class);
                        currentWallpaper = wallpaperModel.getCurrentWallpaper();
                        currentGif = wallpaperModel.getCurrentGif();
                        LoadData wallpaperManager = new LoadData(yConstraintLayout, currentWallpaper);
                        wallpaperManager.loadWallpaperData();

                        if (currentGif == 1){
                            GifImageView imggif = findViewById(R.id.GifImageView_GifStart);
                            ImageView imgbtngif = findViewById(R.id.ImageView_GifStop);
                            imggif.setImageResource(R.drawable.bg1_gifbread);
                            imgbtngif.setImageResource(R.drawable.bg1_pngbread);
                        }else if (currentGif == 2){
                            GifImageView imggif = findViewById(R.id.GifImageView_GifStart);
                            ImageView imgbtngif = findViewById(R.id.ImageView_GifStop);
                            imggif.setImageResource(R.drawable.bg2_gifcup);
                            imgbtngif.setImageResource(R.drawable.bg2_pngcup);
```

Figure c7.8

The loadData() method here then loads the user data from Firebase Firestore and updates the user interface accordingly. It retrieves the current user's ID from Firebase Authentication then retrieves the user's counter value from the users_data collection and updates the counter variable in the code. It then retrieves the user's wallpaper data from the users_wallpaper_data collection and updates the currentWallpaper and currentGif variables in the code accordingly. The method also loads the wallpaper data using the LoadData function and updates the user interface by setting the appropriate background image and button images based on the currentGif value. The addOnSuccessListener and addOnFailureListener to handle the results of the Firebase Firestore queries and includes conditional statements to handle cases where the queried documents do not exist or where the currentGif value does not match any of the predefined values.

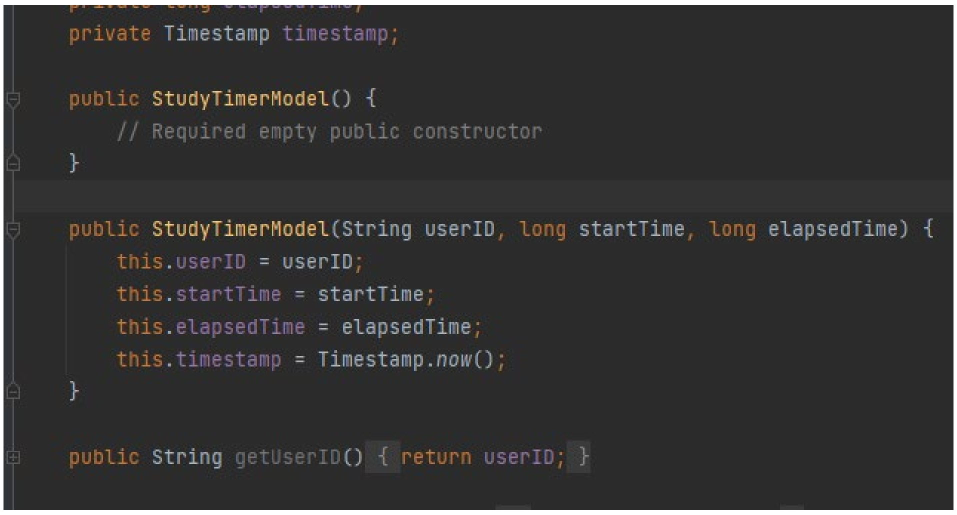


```java
public class LoadData {
    private ConstraintLayout zConstraintLayout;
    private int currentWallpaper;


    public LoadData(ConstraintLayout zConstraintLayout, int currentWallpaper) {
        this.zConstraintLayout = zConstraintLayout;
        this.currentWallpaper = currentWallpaper;
    }

    public void setConstraintLayout(ConstraintLayout zConstraintLayout) {
        this.zConstraintLayout = zConstraintLayout;
    }

    public void loadWallpaperData() {

        if (currentWallpaper == 1) {
            zConstraintLayout.setBackgroundResource(R.drawable.bgdesign1_beige);
            currentWallpaper = 1;
        }
        else if (currentWallpaper == 2) {
            zConstraintLayout.setBackgroundResource(R.drawable.bgdesign2_heart);
            currentWallpaper = 2;
        }
        else if (currentWallpaper == 3){
            zConstraintLayout.setBackgroundResource(R.drawable.bgdesign3_grass);
            currentWallpaper = 3;
        }
```

Figure c7.9

The loadData class is responsible for setting the wallpaper of the application. This method checks the value of the currentWallpaper field and sets the background resource of the ConstraintLayout accordingly using the setBackgroundResource() method. The method has a series of if-else statements to check the value of the currentWallpaper field and set the background resource of the ConstraintLayout accordingly.

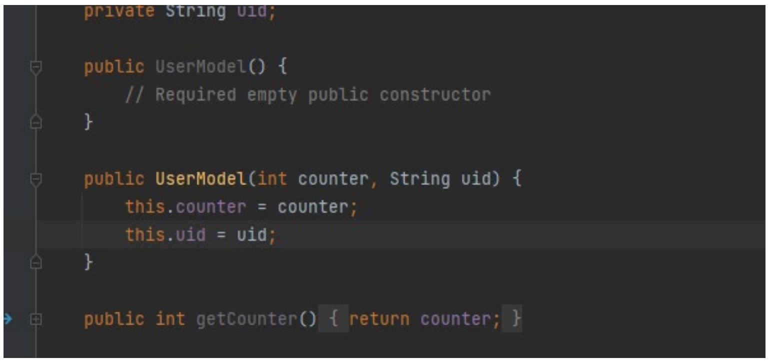
The StudyTimerModel and UserModel classes are used to store and retrieve data related to a user's study activity.



```java
    private Timestamp timestamp;

    public StudyTimerModel() {
        // Required empty public constructor
    }

    public StudyTimerModel(String userID, long startTime, long elapsedTime) {
        this.userID = userID;
        this.startTime = startTime;
        this.elapsedTime = elapsedTime;
        this.timestamp = Timestamp.now();
    }

    public String getUserID() { return userID; }
```

Figure c8.0

The StudyTimerModel class has four instance variables: userID, startTime, elapsedTime, and timestamp. The userID variable is a String that represents the unique ID of the user. The StudyTimerModel class has two constructors. The default constructor is empty and the other constructor takes in three parameters: userID, startTime, and elapsedTime. The constructor initializes the instance variables with the provided values and sets the timestamp to the current time using the Timestamp.now() method.The StudyTimerModel class also has getter and setter methods for each of the instance variables.



```java
    private String uid;

    public UserModel() {
        // Required empty public constructor
    }

    public UserModel(int counter, String uid) {
        this.counter = counter;
        this.uid = uid;
    }

    public int getCounter() { return counter; }
```

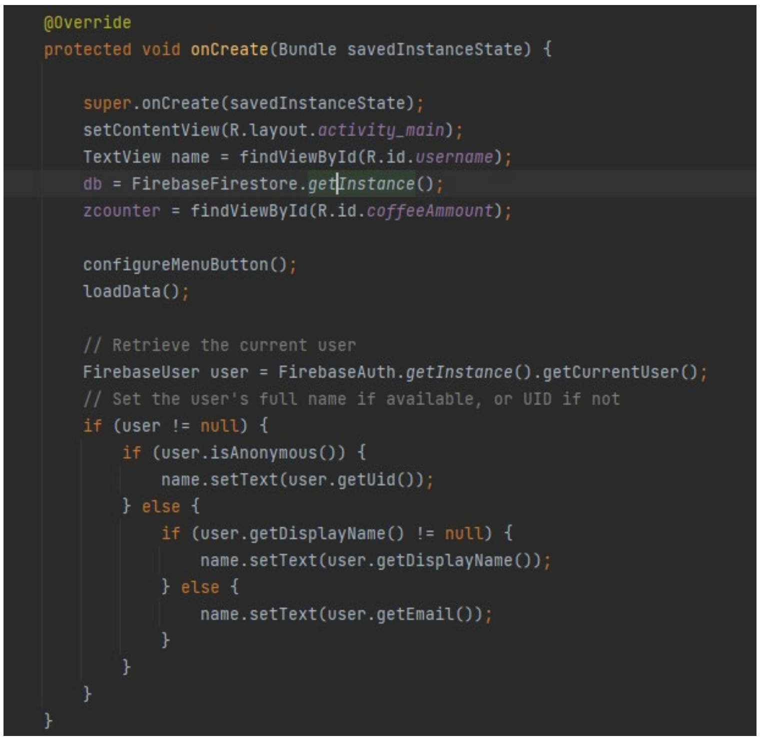
Figure c8.1

The UserModel class has two instance variables: counter and uid. The UserModel class has two constructors. The default constructor is empty, and the other constructor takes in two parameters which are the counter and uid. The constructor initializes the instance variables with the provided values.The UserModel class also has getter and setter methods for each of the instance variables.

Other Implementation

The implementations here are how the UI have been implemented. These are the settings Menu and the splash activity.

The MainActivity class serves as the central component and hub of the application. It provides a means for the authenticated user to interact with the application.

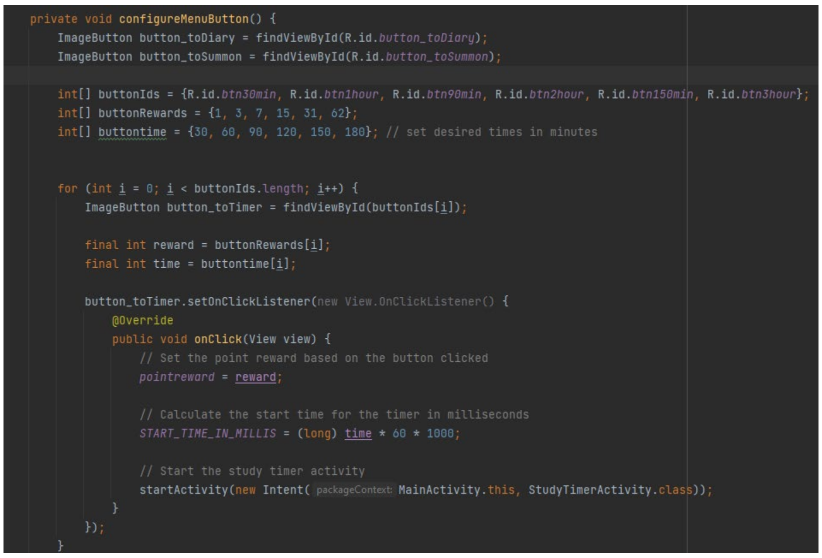


```java
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    TextView name = findViewById(R.id.username);
    db = FirebaseFirestore.getInstance();
    zcounter = findViewById(R.id.coffeeAmmount);

    configureMenuButton();
    loadData();

    // Retrieve the current user
    FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
    // Set the user's full name if available, or UID if not
    if (user != null) {
        if (user.isAnonymous()) {
            name.setText(user.getUid());
        } else {
            if (user.getDisplayName() != null) {
                name.setText(user.getDisplayName());
            } else {
                name.setText(user.getEmail());
            }
        }
    }
}
```

Figure c8.2

The onCreate() method is called when the activity is created. It sets the layout for the activity with the setContentView() method, and then sets up the menu buttons with the configureMenuButton() method. It also loads the user's data and retrieves the user's full name or UID and sets it in a TextView.

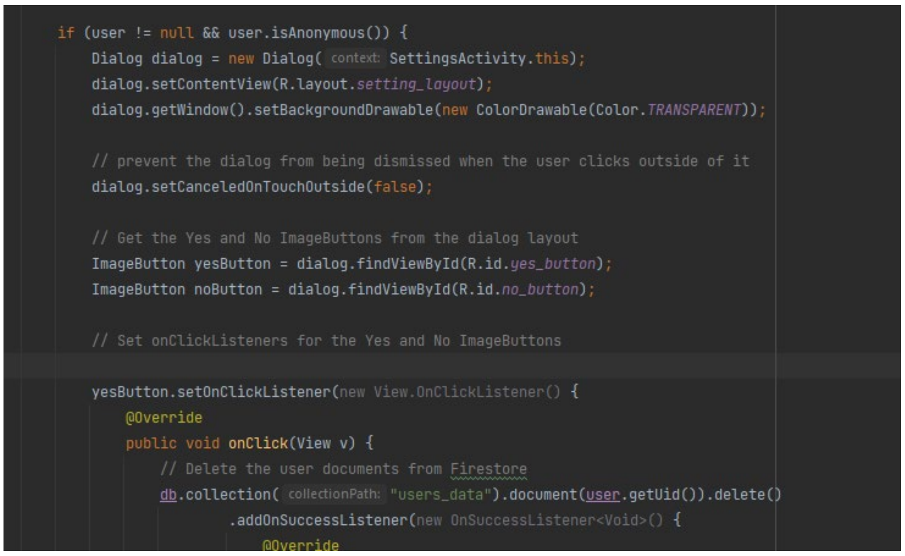
The loadData() method retrieves the user's data from the Firebase database and updates the TextView with the loaded counter value. It also retrieves the current wallpaper and loads it using the LoadData class.



```java
private void configureMenuButton() {
    ImageButton button_toDiary = findViewById(R.id.button_toDiary);
    ImageButton button_toSummon = findViewById(R.id.button_toSummon);

    int[] buttonIds = {R.id.btn30min, R.id.btn1hour, R.id.btn90min, R.id.btn2hour, R.id.btn150min, R.id.btn3hour};
    int[] buttonRewards = {1, 3, 7, 15, 31, 62};
    int[] buttontime = {30, 60, 90, 120, 150, 180}; // set desired times in minutes


    for (int i = 0; i < buttonIds.length; i++) {
        ImageButton button_toTimer = findViewById(buttonIds[i]);

        final int reward = buttonRewards[i];
        final int time = buttontime[i];

        button_toTimer.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Set the point reward based on the button clicked
                pointreward = reward;

                // Calculate the start time for the timer in milliseconds
                START_TIME_IN_MILLIS = (long) time * 60 * 1000;

                // Start the study timer activity
                startActivity(new Intent( packageContext: MainActivity.this, StudyTimerActivity.class));
            }
        });
    }
```

Figure c8.3

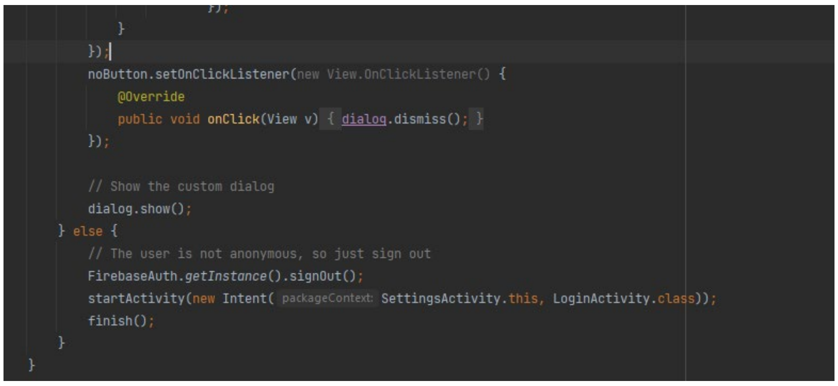
The configureMenuButton() method sets up the OnClickListener for each menu button. It sets the other menu buttons to start the corresponding activities, point reward and start time for the study timer activity based on the button clicked, and then starts the activity. Inside the click listener the value of the pointreward variable is set to the reward where it can be obtained from the buttonRewards array. The value of the START_TIME_IN_MILLIS variable is set to time * 60 * 1000 which is the desired time in milliseconds calculated from the value in the buttontime array. This will set the timer on how long each button is which in this case it has been made to calculate it be incremented in 30 minutes.



```
if (user != null && user.isAnonymous()) {
    Dialog dialog = new Dialog( context: SettingsActivity.this);
    dialog.setContentView(R.layout.setting_layout);
    dialog.getWindow().setBackgroundDrawable(new ColorDrawable(Color.TRANSPARENT));

    // prevent the dialog from being dismissed when the user clicks outside of it
    dialog.setCanceledOnTouchOutside(false);

    // Get the Yes and No ImageButtons from the dialog layout
    ImageButton yesButton = dialog.findViewById(R.id.yes_button);
    ImageButton noButton = dialog.findViewById(R.id.no_button);

    // Set onClickListeners for the Yes and No ImageButtons

    yesButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // Delete the user documents from Firestore
            db.collection( collectionPath: "users_data").document(user.getUid()).delete()
                    .addOnSuccessListener(new OnSuccessListener<Void>() {
                        @Override
```

Figure c8.4

The settings class then consists of the code where the user can sign out. When the sign out button is clicked the code checks if the current user is anonymous or not by getting the current user from Firebase Authentication using FirebaseAuth.getInstance().getCurrentUser(). If the user is anonymous a custom dialog is displayed,prompting the user to confirm if they want to delete their account. If the user confirms that they want to delete their account the code proceeds to delete various user data from Firebase Firestore. This includes deleting the user document from the users_data collection, deleting all documents from the user_timer collection where the userID field matches the current user's UID, deleting all documents from the notes collection where the uid field matches the current user's UID, deleting the user document from the users_wallpaper_data collection and deleting all documents from the rewards subcollection of the users_rewards document where the parent document ID matches the current users UID. Finally, the code deletes the user account from Firebase Authentication using user.delete() and redirects the user to the login screen.

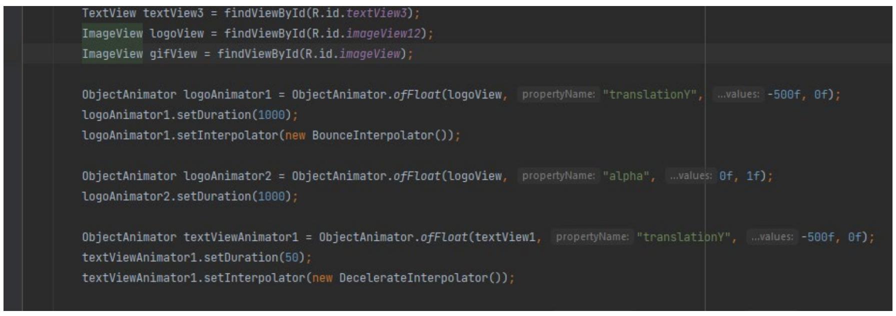


```
            }
        });
        noButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) { dialog.dismiss(); }
        });

        // Show the custom dialog
        dialog.show();
    } else {
        // The user is not anonymous, so just sign out
        FirebaseAuth.getInstance().signOut();
        startActivity(new Intent( packageContext: SettingsActivity.this, LoginActivity.class));
        finish();
    }
}
```

Figure c8.5

If the user is not anonymous, then the code simply signs the user out of the app and redirects them to the login screen.

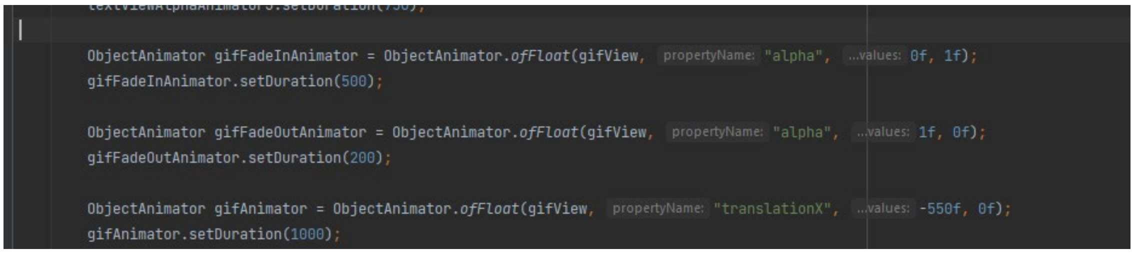
This code creates a splash screen for an Android application, which is the first screen displayed when the application starts. The splash screen consists of an image, some text, and an animation that shows the logo and text moving into place.



```
TextView textView3 = findViewById(R.id.textView3);
ImageView logoView = findViewById(R.id.imageView12);
ImageView gifView = findViewById(R.id.imageView);

ObjectAnimator logoAnimator1 = ObjectAnimator.ofFloat(logoView,  propertyName: "translationY",  ...values: -500f, 0f);
logoAnimator1.setDuration(1000);
logoAnimator1.setInterpolator(new BounceInterpolator());

ObjectAnimator logoAnimator2 = ObjectAnimator.ofFloat(logoView,  propertyName: "alpha",  ...values: 0f, 1f);
logoAnimator2.setDuration(1000);

ObjectAnimator textViewAnimator1 = ObjectAnimator.ofFloat(textView1,  propertyName: "translationY",  ...values: -500f, 0f);
textViewAnimator1.setDuration(50);
textViewAnimator1.setInterpolator(new DecelerateInterpolator());
```

Figure c8.6

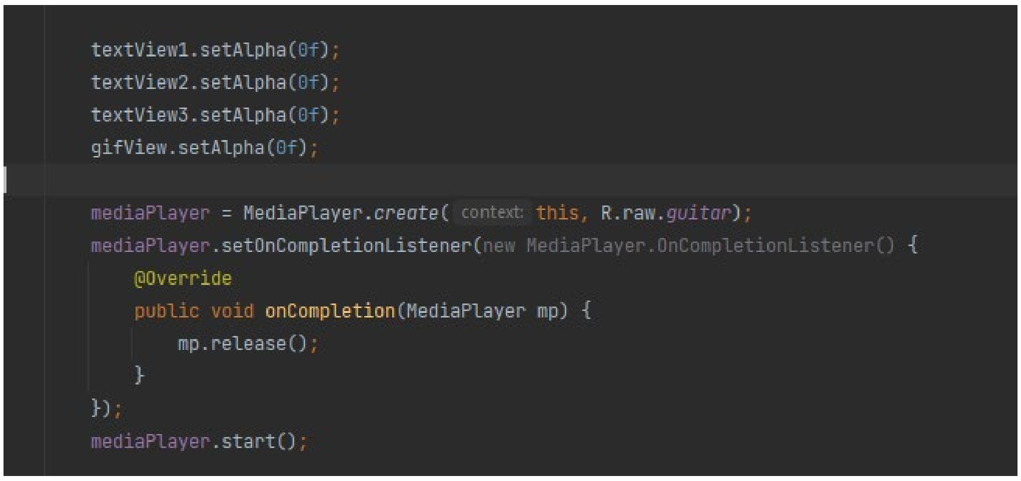
The onCreate() method has several ObjectAnimator objects which animate the activity. For example, logoAnimator1 animates the logoView object by changing its "translationY" property from -500f to 0f over the course of 1000 milliseconds. It also sets the animation's interpolator to a BounceInterpolator which causes the logo to bounce when it reaches its final position.



```
ObjectAnimator gifFadeInAnimator = ObjectAnimator.ofFloat(gifView,  propertyName: "alpha",  ...values: 0f, 1f);
gifFadeInAnimator.setDuration(500);

ObjectAnimator gifFadeOutAnimator = ObjectAnimator.ofFloat(gifView,  propertyName: "alpha",  ...values: 1f, 0f);
gifFadeOutAnimator.setDuration(200);

ObjectAnimator gifAnimator = ObjectAnimator.ofFloat(gifView,  propertyName: "translationX",  ...values: -550f, 0f);
gifAnimator.setDuration(1000);
```

Figure c8.7

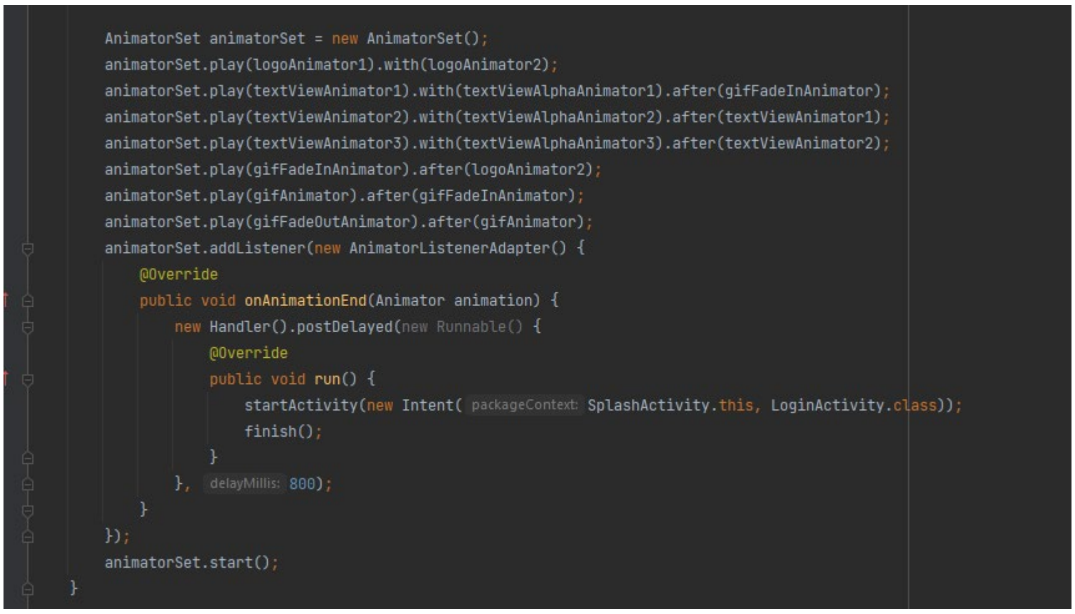
The ObjectAnimator objects changes the alpha property of the TextView and ImageView objects to create a fade-in effect. The gifFadeInAnimator for instance changes the alpha of the gifView from 0f to 1f over the course of 500 milliseconds which gives the fade effect.



```java
textView1.setAlpha(0f);
textView2.setAlpha(0f);
textView3.setAlpha(0f);
gifView.setAlpha(0f);

mediaPlayer = MediaPlayer.create( context: this, R.raw.guitar);
mediaPlayer.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
    @Override
    public void onCompletion(MediaPlayer mp) {
        mp.release();
    }
});
mediaPlayer.start();
```

Figure c8.8

The initial alpha of the TextView and ImageView objects to 0f so that they are not visible at the start of the animation.The MediaPlayer object and sets it to play an audio file.



```java
        AnimatorSet animatorSet = new AnimatorSet();
        animatorSet.play(logoAnimator1).with(logoAnimator2);
        animatorSet.play(textViewAnimator1).with(textViewAlphaAnimator1).after(gifFadeInAnimator);
        animatorSet.play(textViewAnimator2).with(textViewAlphaAnimator2).after(textViewAnimator1);
        animatorSet.play(textViewAnimator3).with(textViewAlphaAnimator3).after(textViewAnimator2);
        animatorSet.play(gifFadeInAnimator).after(logoAnimator2);
        animatorSet.play(gifAnimator).after(gifFadeInAnimator);
        animatorSet.play(gifFadeOutAnimator).after(gifAnimator);
        animatorSet.addListener(new AnimatorListenerAdapter() {
            @Override
            public void onAnimationEnd(Animator animation) {
                new Handler().postDelayed(new Runnable() {
                    @Override
                    public void run() {
                        startActivity(new Intent( packageContext: SplashActivity.this, LoginActivity.class));
                        finish();
                    }
                },  delayMillis: 800);
            }
        });
        animatorSet.start();
    }
```

Figure c8.9

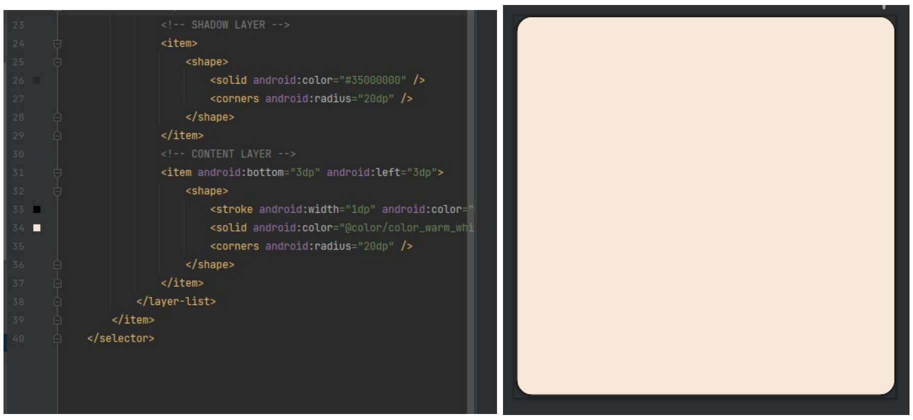
Then I created a AnimatorSet object where it defines the order and timing of the animations. The animations are played together using the "with" method or after each other using the "after" method. The last animation, gifFadeOutAnimator,is played after gifAnimator and causes the gifView to fade out.

Finally, an animation listener is added to the animatorSet which launches the LoginActivity activity after the animation completes. The onDestroy method is also overridden to release the MediaPlayer object when the activity is destroyed.

Figure c9.0

2.4. Graphical User Interface (GUI)

In this section, I am pleased to present the XML layout screenshots, which provide a visual representation of the graphical user interface (GUI) that will be integrated into the application. These screenshots showcase the layout, design, and components that will be utilized to enhance the user experience and facilitate interaction with the application's features.



```
                <!-- SHADOW LAYER -->
                <item>
                    <shape>
                        <solid android:color="#35000000" />
                        <corners android:radius="20dp" />
                    </shape>
                </item>
                <!-- CONTENT LAYER -->
                <item android:bottom="3dp" android:left="3dp">
                    <shape>
                        <stroke android:width="1dp" android:color="
                        <solid android:color="@color/color_warm_whi
                        <corners android:radius="20dp" />
                    </shape>
                </item>
            </layer-list>
        </item>
    </selector>
```

Figure 1.1 xml-shadow-btn

To enhance the user interface of an application, developers often use XML layouts to create customised buttons, icons, and other visual elements. These layouts allow them to design the application's appearance, including creating customised buttons that add shadows and other visual effects. These customisations improve the application's aesthetics, making it more appealing and engaging to users.

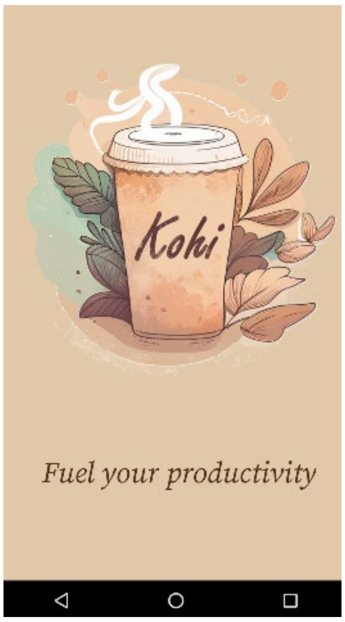


Figure 1.2 splash-screen

This is the splash screen that appears when the user launches the application.

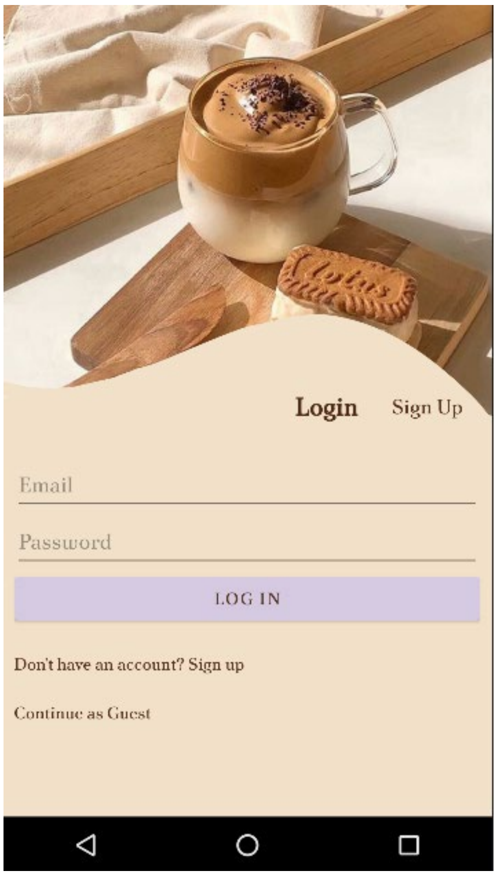


Figure 1.3 login

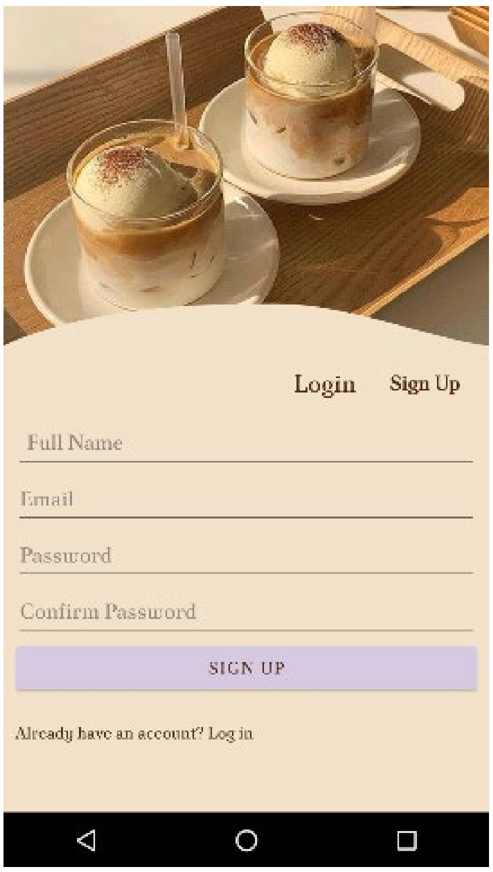
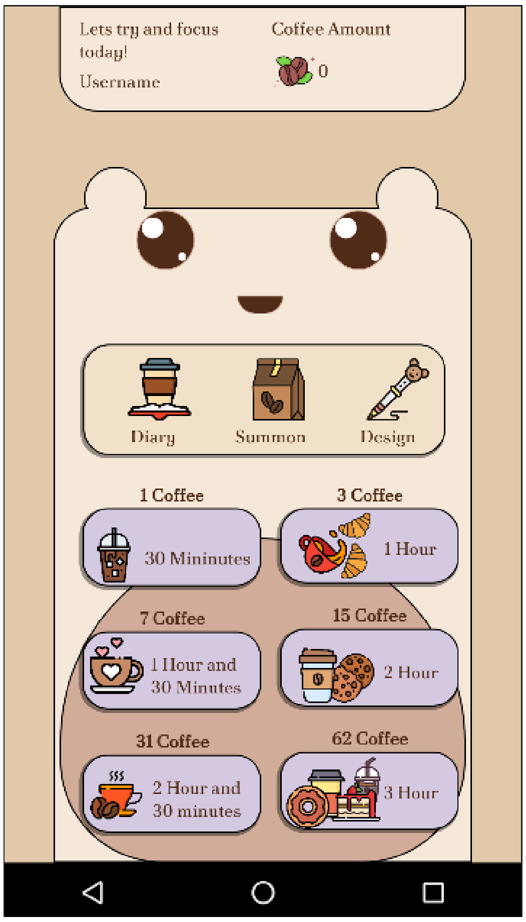


Figure 1.4.Signup

Upon launching the application, the user is directed to the login page where they are presented with a user-friendly login interface. Here the user can choose to log in as a guest or as a registered user. Additionally for new users the application provides an option to sign up which redirects them to the signup page where they can effortlessly input their details and register as a new user. The signup process is straightforward and ensures that the user's personal data is kept secure and confidential.



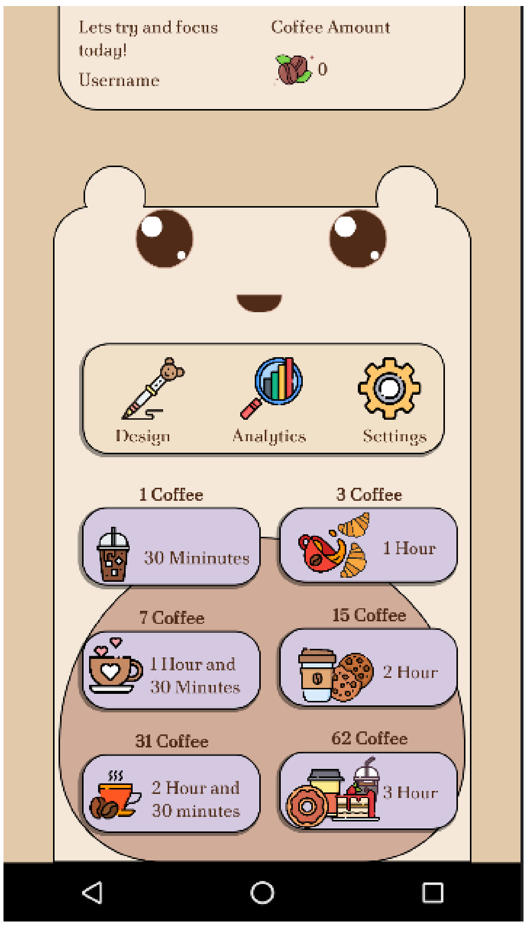
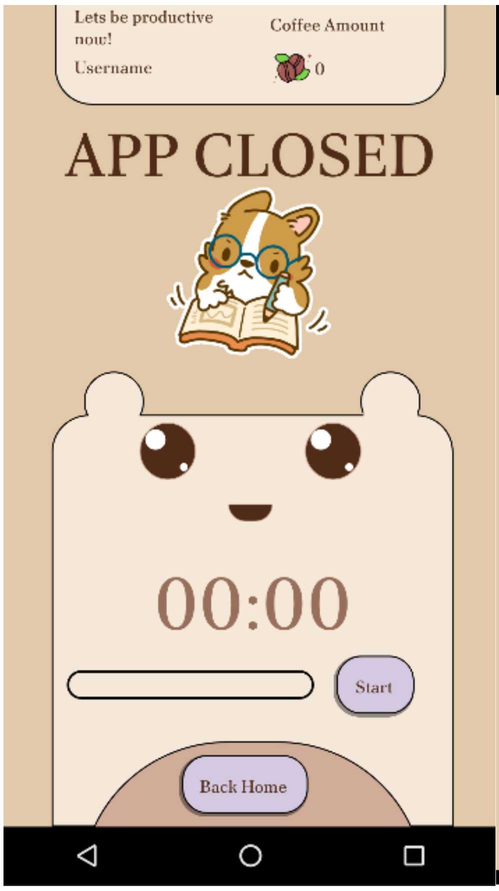


Figure 1.5 menu

Figure 1.6 menu-scroll

The user is directed to the main menu page where they can access the applications main features. The page is designed with a scrollable body where the user can easily navigate through the components which are the diary, summon, design, settings and analytics. The top section of the page displays the user's name and a coffee beans icon which represents the number of points they have earned through their focus sessions. The menu page also features six timer buttons where the user can select the desired length of their session and the corresponding reward.



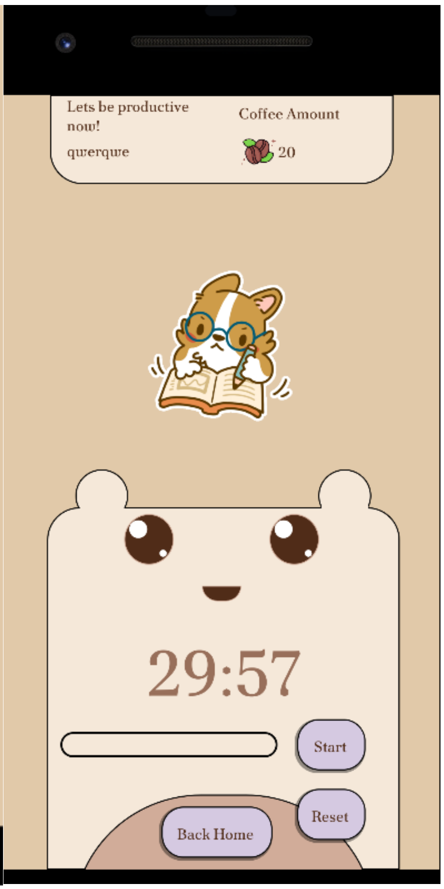


Figure 1.7 timer-start Figure 1.8 pause

When the user decides to initiate a focused session they will be directed to the Timer UI which allows them to start, pause, and reset the timer.

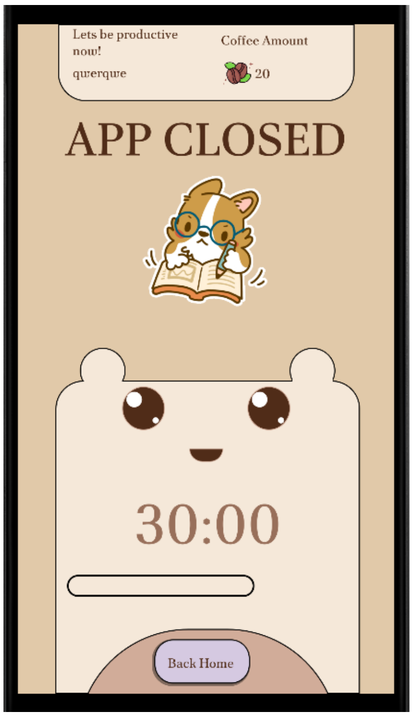


Figure 1.8 App-Closed

In case the user decides to exit the application, a message notifying them of the app's closure will appear, and they will be redirected back to the menu, where they will have to reset their session.

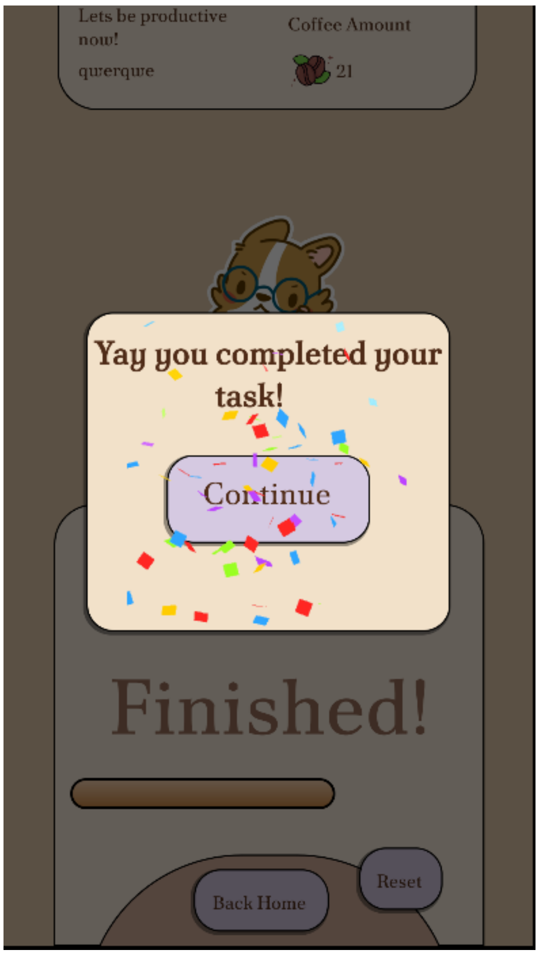
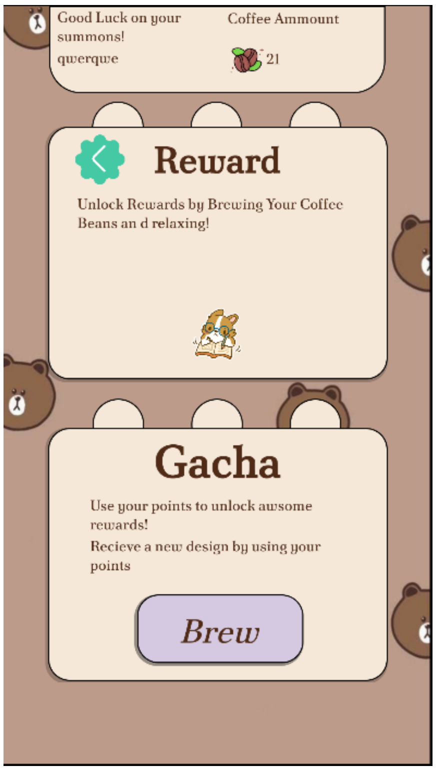


Figure 1.9 session-complete

When the user completes a focus session the user is presented with a celebratory dialog displaying the message "you completed your task". We can see that the user's point balance increased by 1 point as indicated by the UI. The user is then redirected back to the menu page to continue using the application.


Good Luck on your summons!

querqwe

Coffee Ammount

21

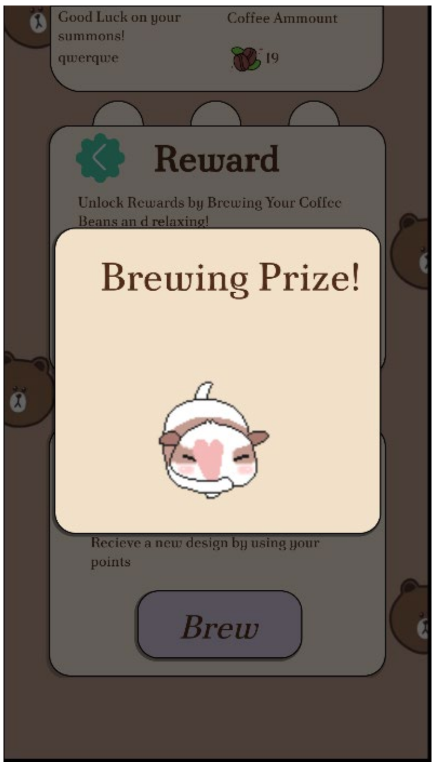
Reward

Unlock Rewards by Brewing Your Coffee Beans an d relaxing!

Gacha

Use your points to unlock awsome rewards!

Recieve a new design by using your points


Good Luck on your summons!

querqwe

Coffee Ammount

19

Reward

Unlock Rewards by Brewing Your Coffee Beans an d relaxing!

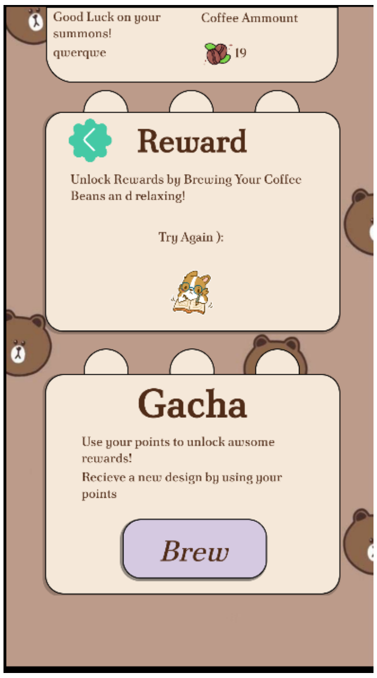
Brewing Prize!

Recieve a new design by using your points

Figure 2.0 gacha-UI

Figure 2.1 gacha-Animation

The user can then navigate back to the menu page where they have the option to use their points in the summon page. By clicking the Brew button a dialog will appear displaying a gacha animation offering the user a chance to win a reward.


Good Luck on your summons!

querqwe

Coffee Ammount

19

Reward

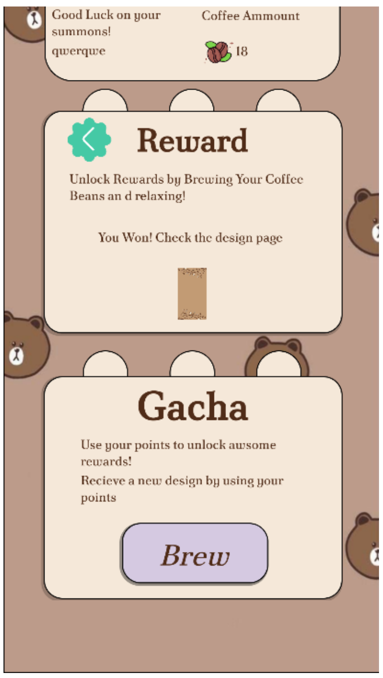
Unlock Rewards by Brewing Your Coffee Beans an d relaxing!

Try Again):

Gacha

Use your points to unlock awsome rewards!

Recieve a new design by using your points


Good Luck on your summons!

querqwe

Coffee Ammount

18

Reward

Unlock Rewards by Brewing Your Coffee Beans an d relaxing!

You Won! Check the design page

Gacha

Use your points to unlock awsome rewards!

Recieve a new design by using your points

Figure 2.3 gacha-reward-fail

Figure 2.4 gacha-reward-pass

In the event that the user does not win any reward a message will be displayed to inform them to try again. But if the user is a winner a reward icon will be displayed with a message saying they won. However, if the user's points are insufficient, they will be prompted with a notification stating that they do not have enough points to proceed. The user can then navigate back to the menu using the green button with an arrow icon.

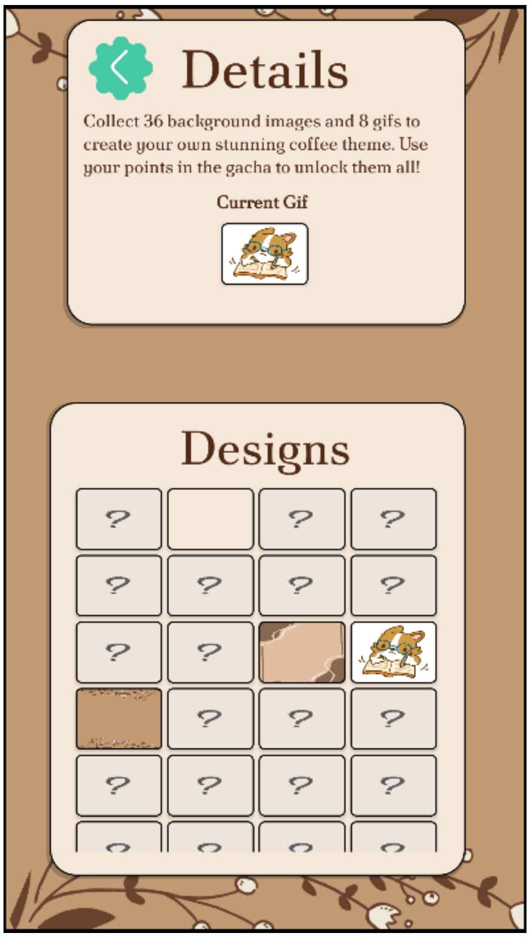
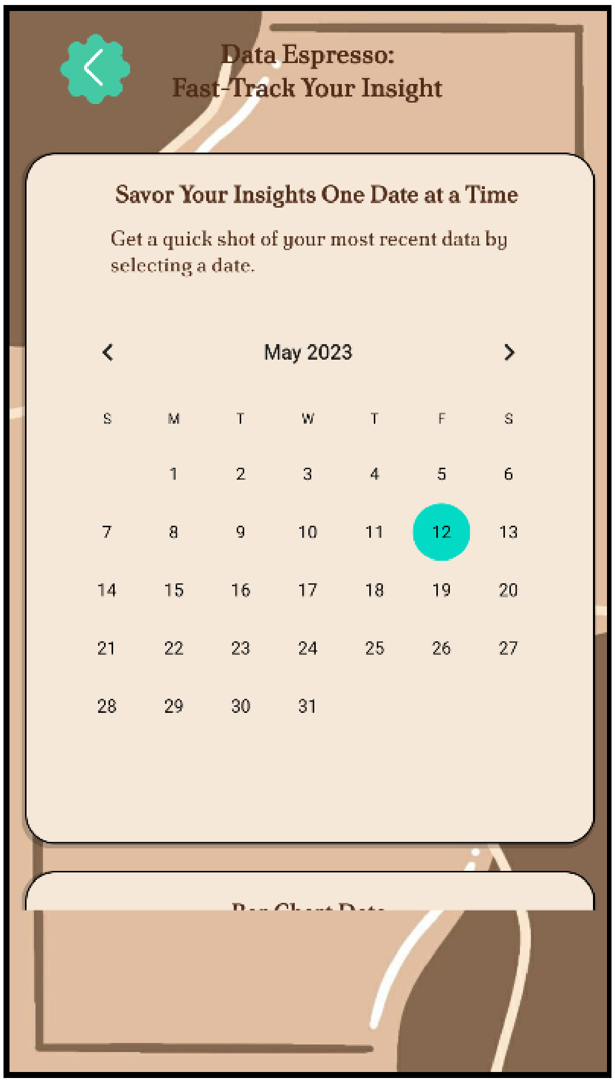

Details
Collect 36 background images and 8 gifs to create your own stunning coffee theme. Use your points in the gacha to unlock them all!
Current Gif
Designs

Figure 2.5 design-page

After winning a reward in the summon page the user can proceed to choose the design they won. The UI allows the user to preview how the design changes with different backgrounds. Additionally the selected GIF will be displayed at the bottom of the text "current gif." If the user attempts to select a locked reward they will be redirected back to the summon page.



Data Espresso:
Fast-Track Your Insight

Savor Your Insights One Date at a Time

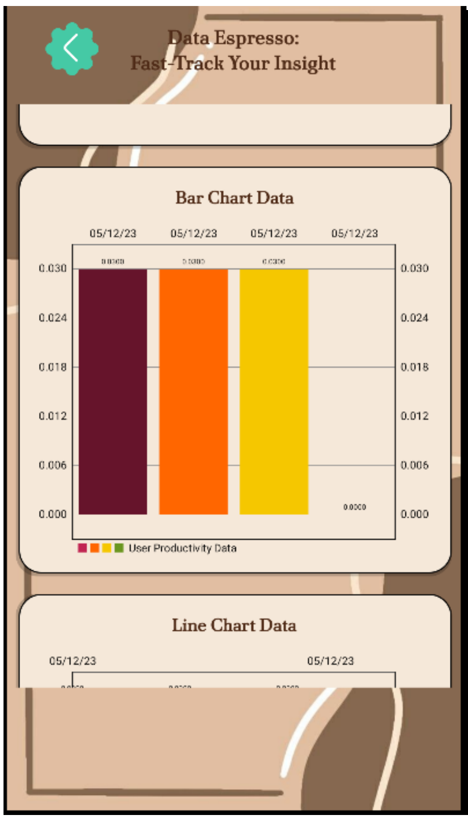
Get a quick shot of your most recent data by selecting a date.

May 2023

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Figure 2.6 anlytics-date

After desining the user can go back to th menu to navigate to the analytics page. The user is presented with two graphs and a calendar to view their focus session data. The user can select a specific date on the calendar to view their focus time for that day and the graphs will update accordingly.



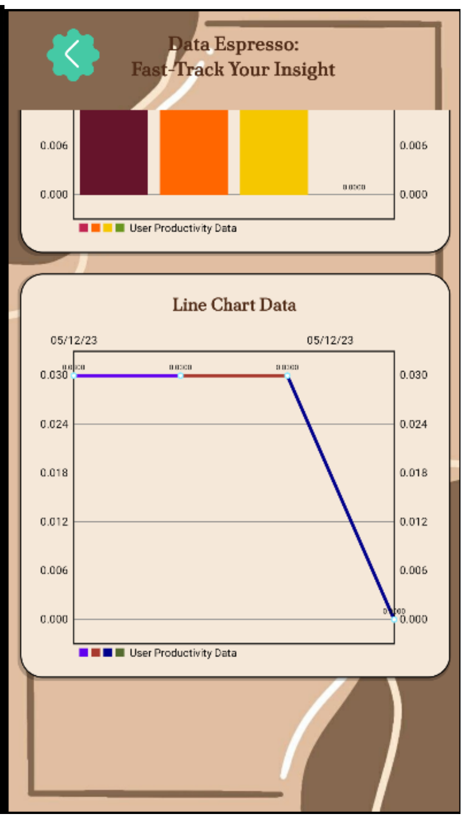


Figure 2.7 anlytics-bar Figure 2.8 anlytics-line

The first graph displays the user's focus sessions in a bar while the second is a line graph shows the latest 4 focus time sessions of the selected day.

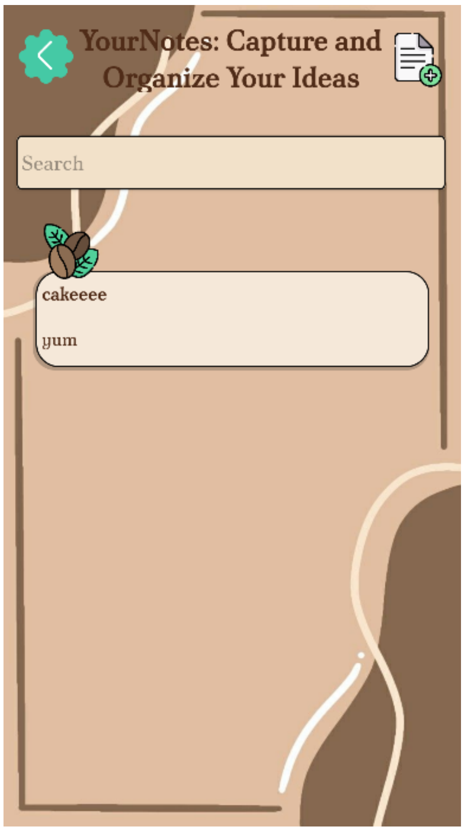
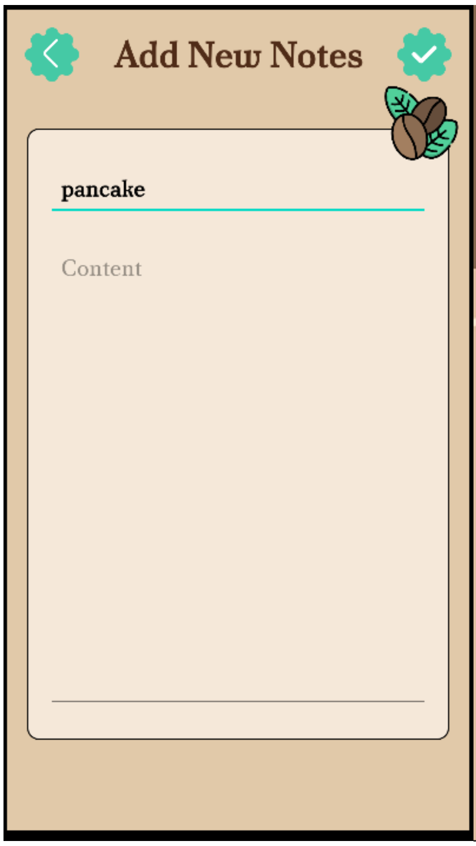


Figure 2.9 diary-main

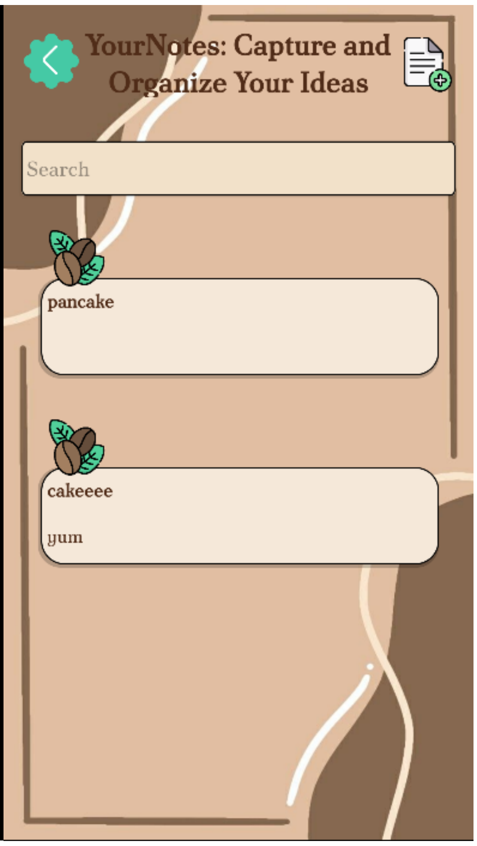
Here the user can view, create and add notes in the diary.

The user can click the green back button where the user can go back to the main menu. Here they can go to the Diary feature where it allows the user to create, search and view notes with ease. The file icon enables the user to create new notes and the notes are displayed as a RecyclerView where a preview of the note can be seen. From there the user can choose to update or view the complete note by clicking on it.



pancake

Figure 3.0 diary-add



pancake

cakeeee

yum

Figure 3.1 diary-added-view

When the user adds a note, they can input the title and content of the note. They can then save it using the tick mark icon located at the top right corner of the screen. Once saved the user will be directed back to the diary menu where they can view the newly added note.

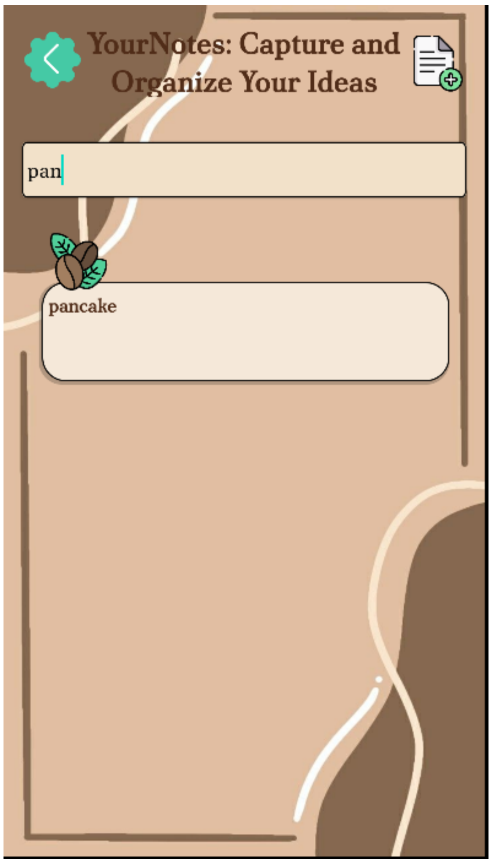


Figure 3.2 diary-search

The user can use the search bar to filter through their notes based on the text they have inputted. This feature allows the user to easily locate specific notes without having to manually scroll through all of their notes.

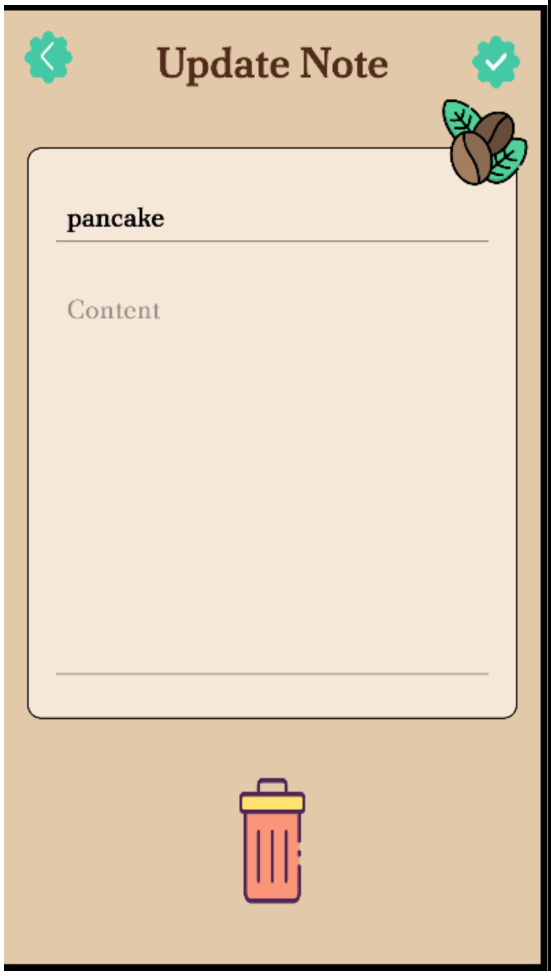


Figure 3.3 diary-display-crud

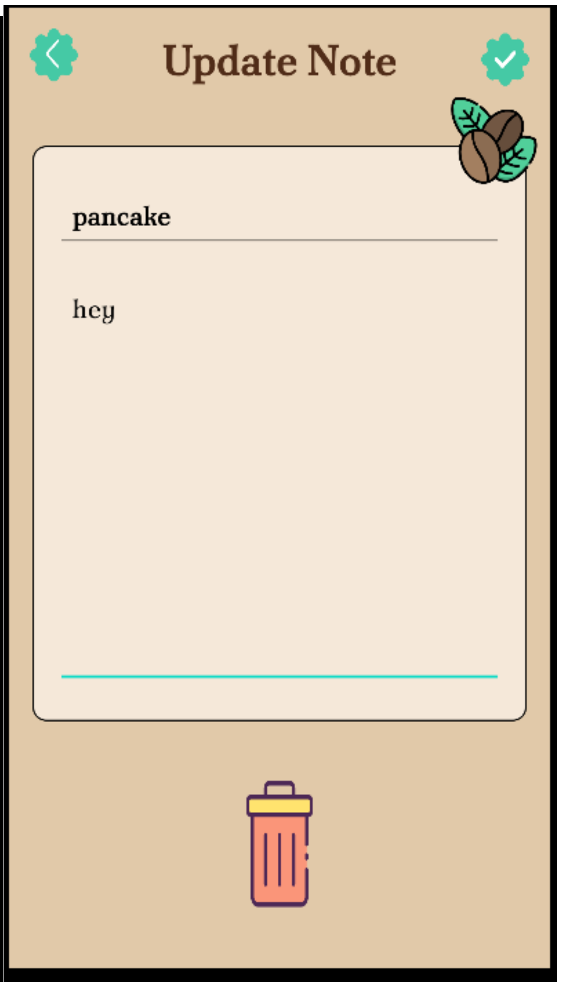
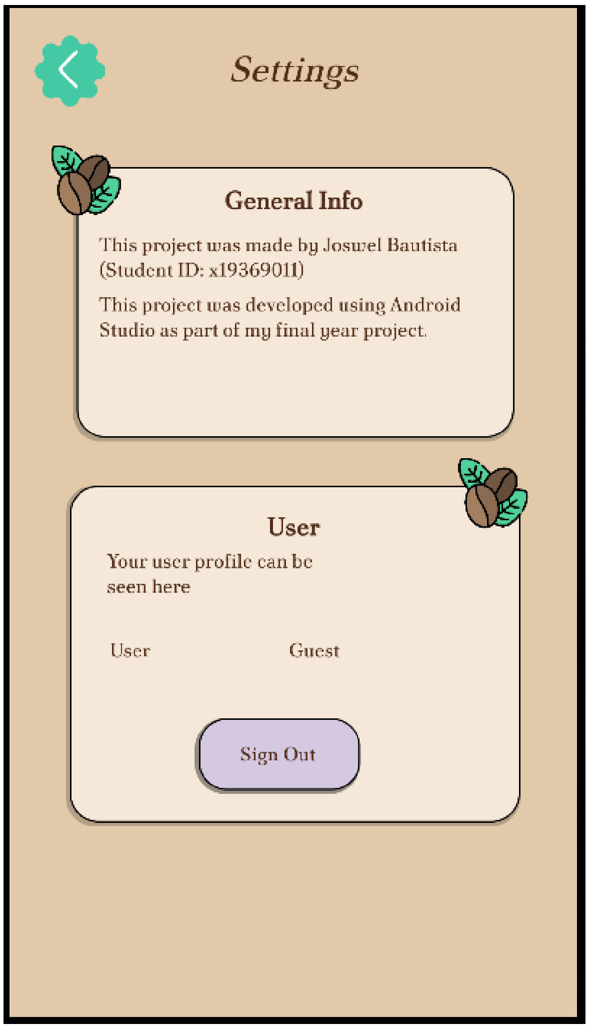


Figure 3.4 diary-edit

Once the user clicks on a note they are directed to a note view page where they can view, edit, and delete the note. The user can edit the title or content of the note by clicking on it. Once they have finished editing they can save their changes by clicking the tick mark icon located on the top right of the page. If the user wishes to go back they can click on the back icon located on the top left of the page. Alternatively if the user wishes to delete the note they can click on the bin icon located at the bottom of the page.


Settings

General Info

This project was made by Joswel Bautista (Student ID: x19369011)

This project was developed using Android Studio as part of my final year project.

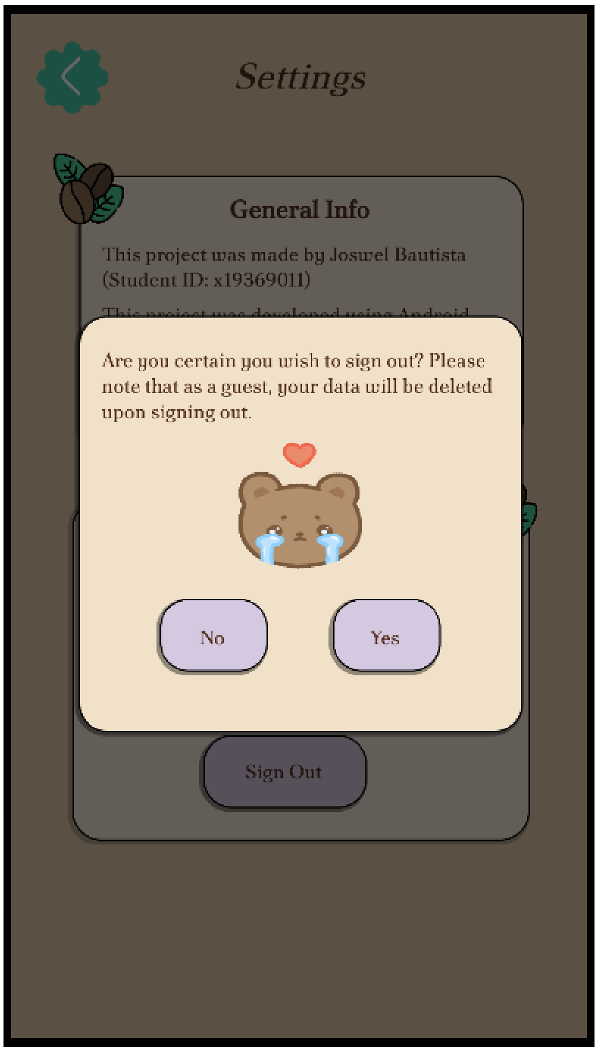
User

Your user profile can be seen here

User Guest

Figure 3.5 settings

If the user wishes to sign out they can navigate to the settings page from the main menu. Here they can see the sign out button which will log them out of their current session. This ensures that the next time they open the app they will need to log in again to access their data. If the user does not wish to sign out and wants to keep their session active they can simply close the app or click the back button to navigate to the previous page.



This project was made by Joswel Bautista (Student ID: x19369011)

Are you certain you wish to sign out? Please note that as a guest, your data will be deleted upon signing out.

Figure 3.5 settings-guest-signout

If the user is a registered user and chooses to sign out their session will be ended. However, if the user is a guest they will be prompted with a message informing them that their data will be deleted upon signing out. The guest user will have the option to either keep their session by clicking "no" or end their session by clicking "yes," which will result in the deletion of their data.

2.5. Testing

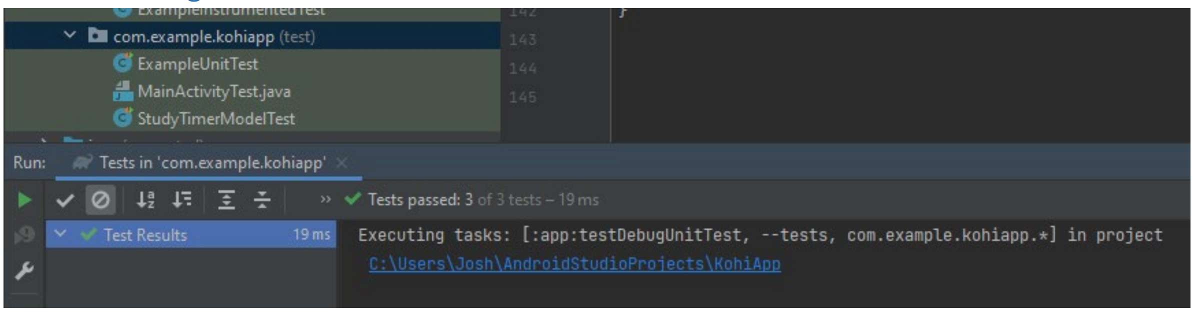
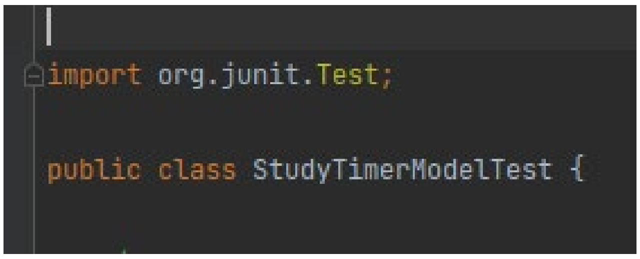

```
com.example.kohiapp (test)
    ExampleUnitTest
    MainActivityTest.java
    StudyTimerModelTest

Run:  Tests in 'com.example.kohiapp'

Tests passed: 3 of 3 tests – 19 ms

Test Results  19 ms
Executing tasks: [:app:testDebugUnitTest, --tests, com.example.kohiapp.*] in project
C:\Users\Josh\AndroidStudioProjects\KohiApp
```

Figure t1.0 testing



```
import org.junit.Test;

public class StudyTimerModelTest {
```

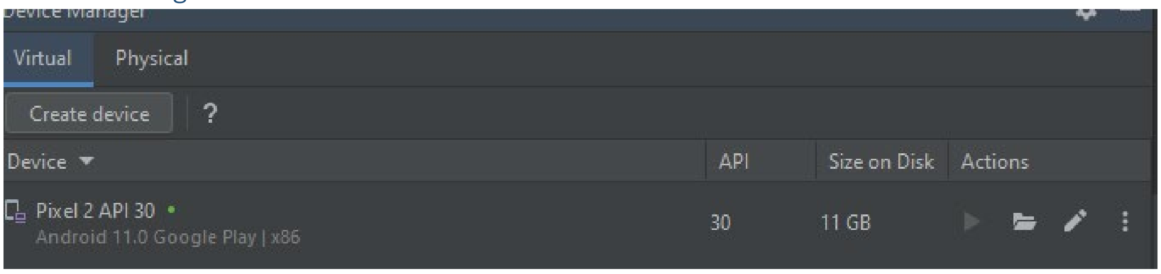
Figure t1.1 junit

As part of my testing plan, I used Junit which is a popular testing framework for Java applications to create unit tests for my Android application. In addition, I used Android Studio built in testing capabilities to execute various types of tests including integration and acceptance tests. Throughout the development process I regularly monitored the logcat in Android Studio to diagnose issues and ensure the proper functioning of my application.

In Scope: All the components in Kohi which were defined in the requirement specs were tested which means that all the features and functionalities specified in the requirements were included in the testing process. This includes user registration and login, focus timer, diary, analytics, summon, and application settings. The testing covered all the expected behaviours functionalities and features of the application as specified in the requirements. Log.

Out of Scope: Any additional components features or functionalities that were not specified in the requirements were not tested as they were not part of the scope. Testing for external factors such as Wi-Fi connection stability, battery life optimization, and network bandwidth utilization were not tested as they fall outside the application's scope.

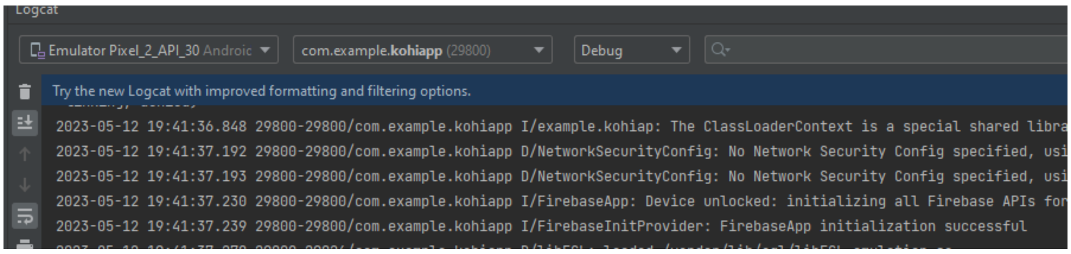
Blackbox testing:



Device	API	Size on Disk	Actions
Pixel 2 API 30 Android 11.0 Google Play | x86	30	11 GB	

Figure t1.1 emulator

Integration testing: This type of testing falls under the category of black box testing which is focused on testing the interaction and integration between different modules or components of the software. The purpose of integration testing is to ensure that the software system as a whole is functioning as expected. I used integration testing to see how different components of a software system function together. I made an emulator or physical device such as my mobile phone to execute tests and validate that the application operates as intended. By conducting integration testing it was aimed to detect any flaws or glitches that may arise when multiple components interact with each other. This ensures that the application delivers the desired functionalities and performance.


Logcat
Emulator Pixel_2_API_30 Androic
com.example.kohiapp (29800)
Debug
Try the new Logcat with improved formatting and filtering options.

```
2023-05-12 19:41:36.848 29800-29800/com.example.kohiapp I/example.kohiap: The ClassLoaderContext is a special shared libra
2023-05-12 19:41:37.192 29800-29800/com.example.kohiapp D/NetworkSecurityConfig: No Network Security Config specified, usi
2023-05-12 19:41:37.193 29800-29800/com.example.kohiapp D/NetworkSecurityConfig: No Network Security Config specified, usi
2023-05-12 19:41:37.230 29800-29800/com.example.kohiapp I/FirebaseApp: Device unlocked: initializing all Firebase APIs for
2023-05-12 19:41:37.239 29800-29800/com.example.kohiapp I/FirebaseInitProvider: FirebaseApp initialization successful
```

Figure t1.1.5 emulator-logcat

Logcat is a crucial tool that is commonly used for debugging and finding errors in Android applications. It is an important part of the Android Debug Bridge (ADB) and provides a real time stream of system messages, including application-specific messages, system logs, and debug messages. By analysing the messages in Logcat I'm able to effectively trace the source of errors in my application and take the necessary steps to fix them. This makes Logcat an essential tool for testing Android applications and ensuring that they are running as expected.

Test Cases

Test Case Type	Description	Test Step	Expected Results	Actual Results	Test Result
Register	Verify user can successfully register for an account	Enter valid user information Click "Register" button	User is directed to login page	Directed to the main page	Pass
Login	Verify user can successfully login to their account	Enter valid login credentials Click "Login" button	User is directed to the menu. With their username	Navigated to the main page. Showing username	Pass
Rewards	Verify user can redeem rewards points	Click "Brew" button for reward	User receives has a chance for a reward. Take a point away	User randomly gets a reward. User loses 1 point.	Pass
Sign Out Sync	Test synchronization of sign out for guest users	Log in as guest user then Sign	User should have an option to sync data	There is no option sync data	Fail
Timer	User receives a reward after study	User completes a focus session	User should gain a point. Save and update user point to firebase	User gets a point. Point saved and updated to firebase	pass
Input	Verify form inputs are validated correctly on signup/login.	Enter invalid input in each field then Submit form	User receives error messages for each invalid field	Message prompt when inputted wrong form input	Pass

Buttons	Verify buttons function correctly	Navigate to Activity with button Click button	Buttons should redirect or submit	Buttons works as intended	Pass

Manual testing was used with the test cases to verify the functionality of the system. This process involved a thorough and structured approach to ensure that each component of the system was examined and tested manually. This enabled the detection and resolution of issues that may not have been identified through automated testing methods.

Whitebox testing:

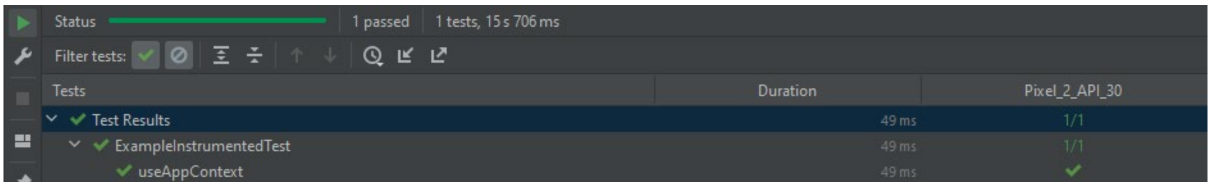
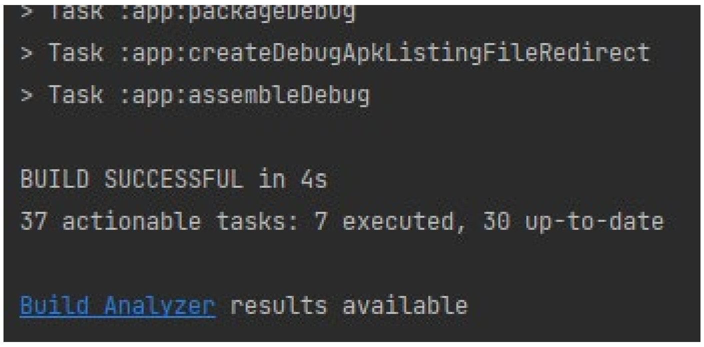

```
Status   1 passed   1 tests, 15 s 706 ms
Filter tests:
Tests                                   Duration   Pixel_2_API_30
  Test Results                           49 ms      1/1
    ExampleInstrumentedTest              49 ms      1/1
      useAppContext                      49 ms      ✓
```

Figure t1.2 test-pass

Unit testing: This type of testing falls under the category of white box testing which is focused on testing individual units or components of the software code in isolation to ensure that they are working as expected.


```
> Task :app:packageDebug
> Task :app:createDebugApkListingFileRedirect
> Task :app:assembleDebug

BUILD SUCCESSFUL in 4s
37 actionable tasks: 7 executed, 30 up-to-date

Build Analyzer results available
```

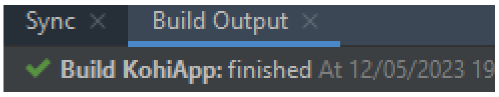
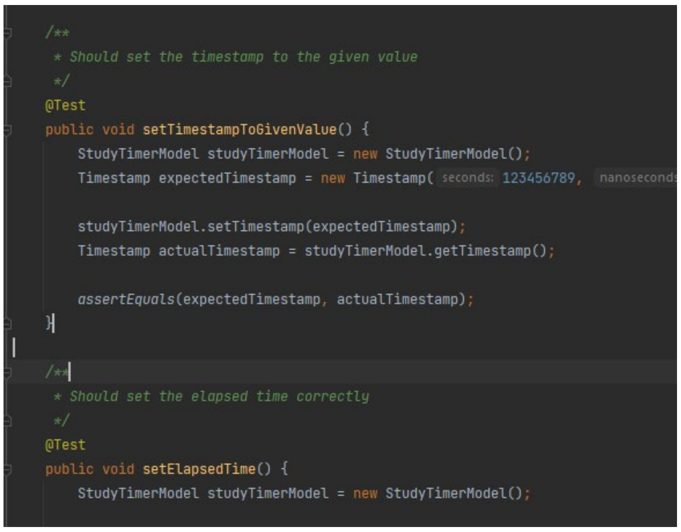

```
Sync ×   Build Output ×
✓ Build KohiApp: finished At 12/05/2023 19
```

Figure t1.2 build-pass

Figure t1.3 build-pass-output

During my testing I executed a series of unit tests to evaluate the behaviour of individual components of my system. These tests were designed to isolate and verify the functionality of each component ensuring that it performs as intended and meets the requirements of the system and allow the application to run and build. These components tested separately is the Timer where I check if the elapsed time is set correctly.



```
/**
 * Should set the timestamp to the given value
 */
@Test
public void setTimestampToGivenValue() {
    StudyTimerModel studyTimerModel = new StudyTimerModel();
    Timestamp expectedTimestamp = new Timestamp( seconds: 123456789,  nanosecond

    studyTimerModel.setTimestamp(expectedTimestamp);
    Timestamp actualTimestamp = studyTimerModel.getTimestamp();

    assertEquals(expectedTimestamp, actualTimestamp);
}

/**
 * Should set the elapsed time correctly
 */
@Test
public void setElapsedTime() {
    StudyTimerModel studyTimerModel = new StudyTimerModel();
```

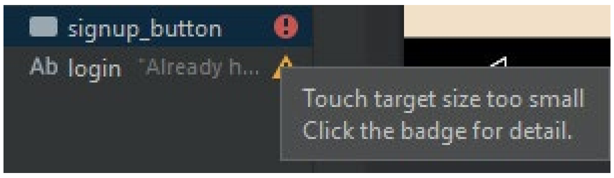
Figure t1.4 timer-test

One positive outcome from these tests was the successful build of the components. This indicates that the code for each component is correct and has been properly integrated with the other components. Additionally, it suggests that any potential issues or errors have been identified and resolved providing greater confidence in the overall quality and reliability of the software system.

Acceptance Testing:

End-user testing: This type of testing falls under the category of acceptance testing which is focused on testing the software from the end user perspective to ensure that it meets their requirements and is fit for purpose. The purpose of end-user testing is to validate that the software system satisfies user requirements.

This testing approach aimed to validate that the software met the user's requirements. I approached this testing as if I were an actual user of the application just like the manual testing which allowed me to gain a genuine understanding of how users would interact with the software. Through this approach I was able to identify potential areas for improvement in the user interface and overall user experience. This provided me with valuable insights that could be used to enhance the software systems usability and overall performance.



signup_button
Ab login "Already h...
Touch target size too small
Click the badge for detail.

Figure t1.5 btn-size

For example, I noticed that some buttons were too small to be clicked and some icons were out of bounds on the screen depending on the phone screen size. This application will still work with this error which is why End-user testing is important since it helps to spot these types of errors.

2.6. Evaluation

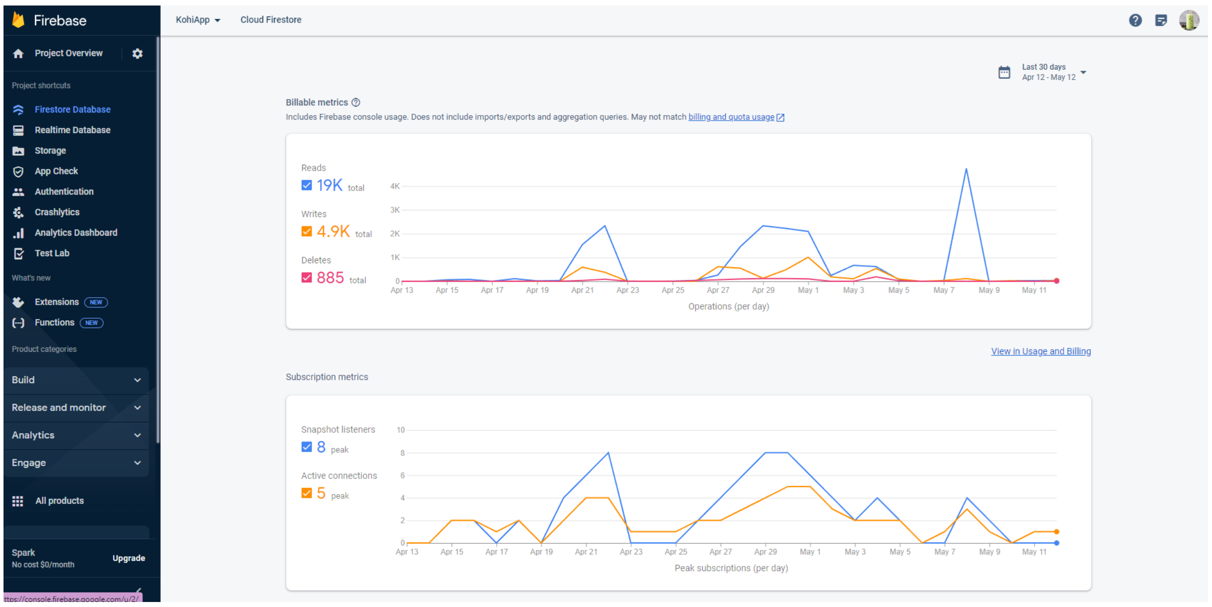


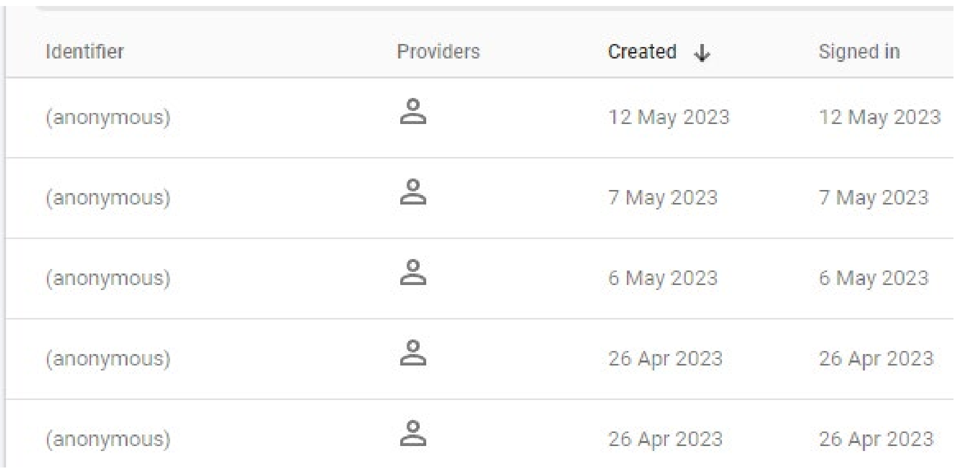
Figure e1 data

The evaluation of the software system is crucial to ensure that it performs optimally and meets the desired requirements. To assess the performance of the system I utilized the Firebase database to monitor the usage data of different users who were granted access to the application. Based on the data collected over the past 30 days the peak number of connected users was five which indicates that the system is capable of handling multiple users concurrently without significant performance issues.

Metric	Value
Peak Connected Users	5
Total Deletes	885
Total Writes	4.9k
Total Reads	19k

Snapshot listeners allow a client to receive updates from the database in real-time whenever there is a change in the data. This is achieved by registering a listener with the database, which will then be notified of changes as they happen.

While the active connections on the other hand, refer to the number of clients that are currently connected to the database. This includes both clients that are actively reading or writing data as well as those that have established a connection but are not currently interacting with the database.



Identifier	Providers	Created ↓	Signed in
(anonymous)	👤	12 May 2023	12 May 2023
(anonymous)	👤	7 May 2023	7 May 2023
(anonymous)	👤	6 May 2023	6 May 2023
(anonymous)	👤	26 Apr 2023	26 Apr 2023
(anonymous)	👤	26 Apr 2023	26 Apr 2023

Figure e1.1 guest-data

Here within the past 30 days, there have been a few instances of anonymous users utilizing the application. Specifically, the most recent anonymous user activity was noted in the month of May. With this I was able to get user feedback in the process of enhancing and improving the application. As a result, I took the initiative to request feedback from these anonymous users which proved to be valuable in terms of improving the functionality and overall user experience of the application. Through this process I was able to identify areas of improvement and implement necessary changes that positively impacted the performance of the application.

In addition to these usage data the system was also evaluated for its scalability and correctness. Performance evaluations were conducted using various load testing tools to simulate heavy user traffic and assess the systems response times under high loads. The system was able to handle increased loads without any significant delays or crashes which indicates good scalability.

3.0 Conclusions

The project has its advantages, disadvantages, strengths, and limitations. Starting with the Advantages and strengths the application is designed with a user-friendly interface that is easy to navigate making it accessible to users of different levels of technological proficiency. Additionally, the project provides customization features that enable users to earn rewards and personalize the interface to their liking. The focus timer feature which is aimed at enhancing productivity which is a strong point of the application. By allowing users to set a session length and focus on a task without interruptions the feature encourages productivity and helps users to manage their time effectively. With so many distractions in today world the focus timer feature offers an effective solution for users who want to increase their productivity and stay focused on their goals. Furthermore, the diary feature enables users to organize their thoughts, activities, and ideas. The analytics feature provides users with insightful data to track their productivity and time management skills. Lastly, the Gacha algorithm provides users with a fun and rewarding way to interact with the application.

However, the application has some disadvantages and limitations that could potentially hinder its overall effectiveness. One of the limitations is the application's limited features which are restricted to time management, diary keeping, and reward-based customization. The lack of other features such as goal setting, habit tracking, or task management could limit the application's appeal to some users. Additionally, the application's lack of real-time synchronization across multiple devices could pose a challenge for users who switch between devices frequently. Another limitation is the limited storage of the diary feature, which could affect users who want to keep detailed notes or have a lot of entries. The absence of collaboration features is another limitation, which could hinder users who wish to share notes or collaborate on tasks with others. Moreover, the customization options are limited which could be a disadvantage for users who prefer to have more control over the interface. The applications security measures are not specified which could be a limitation for users who prioritize privacy and data security. Lastly the absence of user support such as password recovery and the ability to contact support for assistance is a disadvantage.

In conclusion the application presents both strengths and limitations that need to be considered when evaluating its effectiveness. The project user friendly interface, customization features, productivity and organization tools, analytics, and Gacha algorithm provide positive aspects of the application. However, limitations such as limited features lack of real time synchronization, limited storage, lack of collaboration features, limited customization options, lack of security measures and absence of user support could pose a challenge for some users.

4.0 Further Development or Research

One direction the project could take is to expand its features beyond time management, diary keeping, and reward-based customization. For example, adding goal setting, habit tracking, and task management features could make the application more comprehensive and attractive to users who want to manage multiple aspects of their lives in one place.

Another direction the project could take is to improve its user interface and customization options. Providing more customization options and allowing users to change the layout of the application could make it more appealing and personalized to individual users.

Additionally, real-time synchronization between multiple devices could be added to improve user experience and make it more convenient for users who switch between different devices frequently.

Another potential direction the project could take is to add collaboration features. This could include the ability for users to share notes or work on tasks with others making it more suitable for team projects or collaborations.

Finally, enhancing the application's security measures for protecting user data could be another potential direction. Implementing better encryption or providing more information about how user data is protected could help reassure users who prioritize privacy and security.

These are just a few potential directions the project could take with additional time and resources. Generally, the direction the project takes would depend on the goals and priorities it the needs and desires of the target user base.

5.0 References

Please include references throughout your document where appropriate. See here for a guide on referencing from the NCI library.

Bibliography

code, d., n.d. *Timer App in Android Studio and Java.* [Online]
Available at: https://www.youtube.com/watch?v=4aYhSOyAYQQ&t=12s&ab_channel=doctorcode

Developers, A., n.d. *Platform Architecture.* [Online]
Available at:
https://developer.android.com/guide/platform#:~:text=The%20foundation%20of%20the%20Android,and%20low%2Dlevel%20memory%20management.

DEVOPEDIA®, n.d. *Firebase Tutorial.* [Online]
Available at: https://devopedia.org/firebase

Firebase, n.d. *Firebase Authentication.* [Online]
Available at: https://firebase.google.com/docs/auth

Firebase, n.d. *Firebase Documentation.* [Online]
Available at: https://firebase.google.com/docs

Geeks, G. F., n.d. *Android Architecture Patterns.* [Online]
Available at: https://www.geeksforgeeks.org/android-architecture-patterns/

Projects, T., n.d. *Notes app in android studio | Notes app in android.* [Online]
Available at:
https://www.youtube.com/watch?v=Y37PsbMiZ0k&list=PL1tIj6UC0gcut51pibe8VRw4wauQRkEJJ&ab_channel=TechProjects

6.0 Appendices

GitHub Project: https://github.com/NyanNekos/KohiApp

Ethics Approval Application (only if required)

National College of Ireland
DECLARATION OF ETHICS
CONSIDERATION
School of Computing

Student Name:	Joswel Bautista		
Student ID:	X19369011		
Programme	BSHCSD4	**Year:**	2022
Module:	Computing Project Software Development		
Project Title:	Kohi		

Please circle (or highlight) as appropriate

This project involves human participants	[✓ Yes] / No

Introduction

Secondary data refers to data that is collected by someone other than the current researcher. Common sources of secondary data for social science include censuses, information collected by government departments, organizational records and data originally collected for other research purposes. Primary data, by contrast, is collected by the investigator conducting the research.

A project that does not involve human participants requires ONLY completion of Declaration of Ethics Consideration Form and submission of the form on module's Moodle page

A project that involves human participants requires ethical clearance and an Ethics Application Form must be submitted through the module's Moodle page. Please refer to and ensure compliance with the ethical principles stated in NCI Ethics Form available on the Moodle page.

The following decision table will assist you in deciding if you have to complete the Declaration of Ethics Consideration Form or/and the Ethics Application Form.

Public Data	Y	Y	Y	Y	N	N	N	N
Private Data	Y	Y	N	N	Y	Y	N	N
Human Participants	Y	N	Y	N	Y	N	Y	N
Declaration of Ethics Consideration Form	x	X	x	X	X	X	x	
Ethics Application Form	X		X		X		X	

Please circle (or highlight) as appropriate

The project makes use of secondary dataset(s) created by the researcher	Yes / [✓ No]
The project makes use of public secondary dataset(s)	Yes / [✓ No]
The project makes use of non-public secondary dataset(s)	Yes / [✓ No]
Approval letter from non-public secondary dataset(s) owner received	Yes / [✓ No]

Sources of Data:

It is students' responsibility to ensure that they have the correct permissions/authorizations to use any data in a study. Projects that make use of data that does not have authorization to be used, will not be graded for that portion of the study that makes use of such data.

Public Data

A project that makes use of public secondary dataset(s) ***does not need ethics permission****, but* ***needs a letter/email from the copyright holder*** *regarding potential use.*

Some websites and data sources allow their data sets to be used under certain conditions. In these cases, a letter/email from the copyright holder is NOT necessary, but the researcher should cite the source of this permission and indicate under what conditions the data are allowed to be used. See Appendix I for examples of permissions granted by Fingal Open Data, and Eurostat website.

Where websites or data sources indicate that they do not grant permission for data to be used, you will still need a letter/email from the copyright holder. For example, see Appendix II for an example from the Journal of Statistics Education.

Private Data

A project that makes use of non-public (private) secondary dataset(s) must receive data usage permission from School of Computing.

An approval letter/email from the owner *(e.g. institution, company, etc.)* ***of the non-public secondary dataset must be attached to the Declaration of Ethics Consideration****. The letter/email must confirm that the dataset is anonymised and permission for data processing, analysis and public dissemination is granted.*

Evidence for use of secondary dataset(s)

Include dataset(s) owner letter/email or cite the source for usage permission

CHECKLIST

Non-public/private secondary dataset(s) -Owner letter/email is attached to this form ***OR*** Citation and link to the web site where permission is granted – provided in this form	Yes / No Yes / No

ETHICS CLEARANCE GUIDELINES WHEN HUMAN PARTICIPANTS ARE INVOLVED

The Ethics Application Form must be submitted on Moodle for approval prior to conducting the work.

Considerations in data collection

- Participants will not be identified, directly or through identifiers linked to the subjects in any reports produced by the study
- Responses will not place the participants at risk of professional liability or be damaging to the participants' financial standing, employability or reputation
- No confidential data will be used for personal advantage or that of a third party

Informed consent

- Consent to participate in the study has been given freely by the participants
- participants have the capacity to understand the project goals.
- Participants have been given information sheets that are understandable
- Likely benefits of the project itself have been explained to potential participants

- Risks and benefits of the project have been explained to potential participants
- Participants have been assured they will not suffer physical stress or discomfort or psychological or mental stress
- The participant has been assured s/he may withdraw at any time from the study without loss of benefit or penalty
- Special care has been taken where participants are unable to consent for themselves (e.g children under the age of 18, elders with age 85+, people with intellectual or learning disability, individuals or groups receiving help through the voluntary sector, those in a subordinate position to the researcher, groups who do not understand the consent and research process)
- Participants have been informed of potential conflict of interest issues
- The onus is on the researcher to inform participants if deception methods have to be used in a line of research

I have read, understood, and will adhere to the ethical principles described above in the conduct of the project work.

Signature: [illegible]

Date: 29/10/2022

Appendix I

1) Fingal Open Data: http://data.fingal.ie/About

Licence

Citizens are free to access and use this data as they wish, free of charge, in accordance with the Creative Commons Attribution 4.0 International License (CC-BY).

Note: From November 2010 to July 2015, data on Fingal Open Data was published in accordance with the PSI general licence.

Use of any published data is subject to Data Protection legislation.

Licence Statement

Under the CC-BY Licence, users must acknowledge the source of the Information in their product or application by including or linking to this attribution statement: "Contains Fingal County Council Data licensed under a Creative Commons Attribution 4.0 International (CC BY 4.0) licence".

Multiple Attributions

If using data from several Information Providers and listing multiple attributions is not practical in a product or application, users may include a URI or hyperlink to a resource that contains the required attribution statements.

2) Eurostat: https://ec.europa.eu/eurostat/about/policies/copyright

COPYRIGHT NOTICE AND FREE RE-USE OF DATA

Eurostat has a policy of encouraging free re-use of its data, both for non-commercial and commercial purposes. All statistical data, metadata, content of web pages or other dissemination tools, official publications and other documents published on its website, with the exceptions listed below, can be reused without any payment or written licence provided that:

- the source is indicated as Eurostat

- when re-use involves modifications to the data or text, this must be stated clearly to the end user of the information

Appendix II

Journal of Statistics Education: http://jse.amstat.org/jse_users.htm

JSE Copyright and Usage Policy

Unlike other American Statistical Association journals, the Journal of Statistics Education (JSE) does not require authors to transfer copyright for the published material to JSE. Authors maintain copyright of published material. Because copyright is not transferred from the author, permission to use materials published by JSE remains with the author. Therefore, to use published material from a JSE article the requesting person must get approval from the author.

National College of Ireland

Ethical Guidelines and Procedures for Research involving Human Participants

SEPTEMBER 2017

1. Introduction

All research involving human participants that is conducted by students or staff at the National College of Ireland should be done so in an ethical manner. The college has therefore developed an Ethics Committee, which acts as a sub-committee of the Research Committee, to ensure that ethical principles pertaining to research involving human participants are upheld and adhered to. All researchers intending to use human participants as part of their projects are thus required to reflect upon any potential ethical issues and submit their research proposals for ethical review before commencing data collection.

This document gives an overview of the core ethical principles guiding research in NCI, while also documenting the procedures required for seeking ethical approval of research involving human participants.

Am I conducting research?

Research is defined as "the attempt to derive generalisable new knowledge by addressing clearly-defined questions with systematic and rigorous methods" (NHS Health Research Authority). Sometimes, we collect data in order to evaluate a service or practice we are engaged in ("service evaluation"). The main difference between research and service evaluation is in the aim: research is trying to create new generalisable knowledge, and service evaluation is trying to evaluate whether a delivered service/practice is working well. One project may have both aims included in it. It can be confusing if a service or intervention is involved, whether or not research is being conducted. If new or competing interventions are being evaluated, then it is likely to be research, whereas if an existing service is being conducted anyway, with an evaluative component, then it is likely to be a service evaluation. Research requires consideration of the below guiding principles, whereas service evaluation does not require approval from an ethics committee.

2. Guiding Principles

In line with other research institutions, there are three core guiding principles governing the ethical conductance of research involving human participants at NCI. These principles stem from the *Belmont Report* (1979) published by the National Commission for the Protection of Human Subjects of Biomedical and Behavioural Research. While it is recognised that these principles may be operationalised differently depending on the specific research discipline, it is recommended that these are consulted as a starting point for any research involving human participants.

2.1 Principle 1: Respect for Persons

This principle entails recognition that participants should be treated as autonomous individuals and hence should never be coerced or swayed into participating in a research project against their will. The participant's right to withdraw from a research study at any time should be respected, as well as their right to dignity and protection from harm.

Respect for individuals can often be implemented in practice via the process of informed consent, whereby potential participants are made fully aware of the requirements involved in participation. While it is recognised that in certain cases deception (i.e. the withholding of certain information from participants) may take place, this should only occur when it is robustly justified for the validity of the research. In cases where deception is justified, researchers should ensure that any potential risk resulting from this measure is minimised. Participants should also be fully debriefed on the nature of the research after it has taken place.

The principle of respect also requires researchers to protect individuals from vulnerable groups who may have diminished autonomy (see section 4.2 for more detail as to what constitutes vulnerable groups). Where full informed consent is not possible for such population groups, consent may instead be sought from their guardians. In all cases however clear assent, or willingness to participate, should be demonstrated from participants.

2.2 Principle 2: Beneficence and non-maleficence

This principle specifically focuses on the need to protect the well-being of participants. Any potential risk to participants should be minimised, whether that be risk of physical discomfort or of any psychological, emotional or social distress, while possible benefits should be maximised. Researchers adhering to this principle should thus ensure that any potential benefits derived from carrying out the study (e.g. in terms of knowledge gained) should outweigh potential risks. Even in cases where there is only a slight potential risk of harm, participants should be provided with appropriate support to alleviate this.

2.3 Principle 3: Justice

This principle emphasises the need to employ fairness in the distribution of benefits and risks to participants. The way in which participants are selected to take part in research should relate to the purpose of the study, as opposed to other factors such as availability or manipulability of participants. The exploitation of vulnerable populations should be avoided.

Where applicable, researchers are encouraged to consult guidelines stemming from their own professional bodies (e.g. The Psychological Society of Ireland) in addition to the general guiding principles above when planning their research. Researchers should also be sensitive to those issues which are specific to the population under investigation and the methodology that is employed in the project (e.g. qualitative methodologies involving the recording of data may raise issues relating to participants' right to anonymity, as well as the ethical management and use of data). Detailed consideration should be given to all these issues when planning research and when completing the Ethical Review Application form.

3. Ethics Committee

The NCI Ethics Committee was established by the Academic Council in 2012. Acting as a sub-committee to the Research Committee, its role is to oversee ethical issues arising from all research involving human participants that is conducted by students and staff of the college. The key purpose of this committee is to safeguard against any potential harm to participants, and to ensure that their rights are recognised in line with the guiding principles outlined above.

The Ethics Committee reviews all research proposals posing ethical risk to the participants involved, however the decision as to whether projects pose ethical risk is firstly made via the appropriate Filter Committee which operates at School level (see organisational structure in Figure 1 below). The Filter

Committees may review and approve research proposals which are of low ethical risk, while referring those of high ethical risk to be considered by the Ethics Committee (see categories of ethical risk in section 4.1).

While the Filter Committees are made up of staff members with subject-specific knowledge, membership of the Ethics Committee should comprise of no less than five representatives from both the School of Computing and the School of Business, including representatives from the Research Committee.

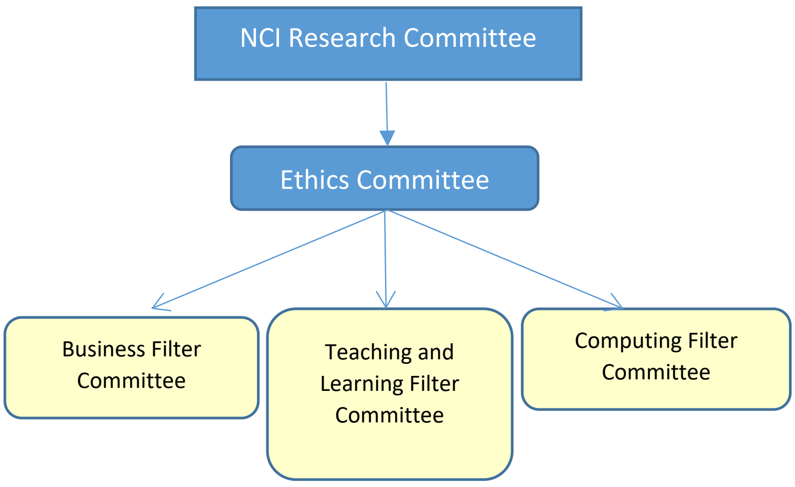

NCI Research Committee
Ethics Committee
Business Filter Committee
Teaching and Learning Filter Committee
Computing Filter Committee

Figure 1: Committee Structures.

4. Review Process

Any staff or student of NCI wishing to conduct a study involving human participants should first submit the Ethical Review Application Form (included at the end of this document), to the relevant School Filter Committee at proposal stage. This initial review will result in a graded categorisation of ethical risk, as outlined below.

4.1 Categorisation of Ethical Risk

Research category A

Research in this category poses little ethical risk to the participants involved. Specifically, it refers to research involving human volunteers, but **excluding** studies involving:

- therapeutic interventions
- new research methodologies
- vulnerable populations (see section 4.2)
- deception of the participants
- any other significant physical, social or psychological risk to participants

Research category B

Research in this category involves human volunteers **including** studies involving:

- therapeutic interventions
- new research methodologies
- vulnerable populations (see section 4.2)
- deception of the participants
- any potentially significant risk to participants

Research Category C

This specifically refers to research involving human volunteers who are service users, patients, staff, records, etc., within the sphere of the HSE or similar setting (but not including clinical trials of investigative medicinal products).

4.2 Vulnerable groups

There are a number of participant populations that may fall under the heading of 'vulnerable groups'. These groups require consideration of unique ethical challenges regardless of the nature of the project. Research involving such populations should therefore always be reviewed by the Ethics Committee.

Groups that may be classed as vulnerable include, but are not limited to:

- Children (under 18 years of age)
- The older old (aged 85+)
- People with an intellectual or learning disability
- Individuals or groups receiving help through the voluntary sector
- Those in a subordinate position to the researcher (e.g. employees)
- Any other groups who might not understand the research and consent process

Note: in addition to the Ethical Review process, any researchers intending to work directly with children will be required to undergo Garda Vetting in advance of the proposed research.

4.3 Exemption from Full Ethical Review

In certain limited cases, researchers can apply for an exemption from full ethical review. In such cases, the Ethical Review Exemption form should be completed, explicitly detailing why the exemption is sought.

In completing this form, researchers must declare that the research does not involve any of the following:

- Vulnerable groups
- Sensitive topics
- Risk of psychological or mental distress
- Risk of physical stress or discomfort
- Any other risk to participants

- Use of drugs or invasive procedures (e.g. blood sampling)
- Deception or withholding of information from participants
- Conflict of interest issues
- Access to data by individuals or organisations other than the researchers
- Any other ethical dilemmas

4.4 Outcomes of Review Process

Following consideration of research projects submitted for Ethical Review, each Filter Committee will submit a report to the Ethics Committee summarising the applications considered and the decisions made.

For research that is deemed to fall under Research Category A (low ethical risk), a favourable outcome at the relevant Filter Committee will be sufficient to secure ethical approval. Research falling under the other two categories must however be considered by the Ethics Committee before approval may be granted.

On the basis of this review, four key outcomes may arise:

1. Research proposal approved (no recommendations)
2. Research proposal approved pending minor revisions (to be accepted by the Chair and Research Supervisor)
3. Research proposal approved pending major revisions (to be resubmitted and approved by the Ethics Committee)
4. Research proposal rejected (resubmission necessary)

A summary of the processes involved in applying for ethical approval can be seen in Figure 2.

Appeals

Appeals against the Committee's decision may be made within ten working days. In this case, at least three members of the Ethics Committee, none of whom will have reviewed the initial application, may review this along with any additional information submitted by the applicant.

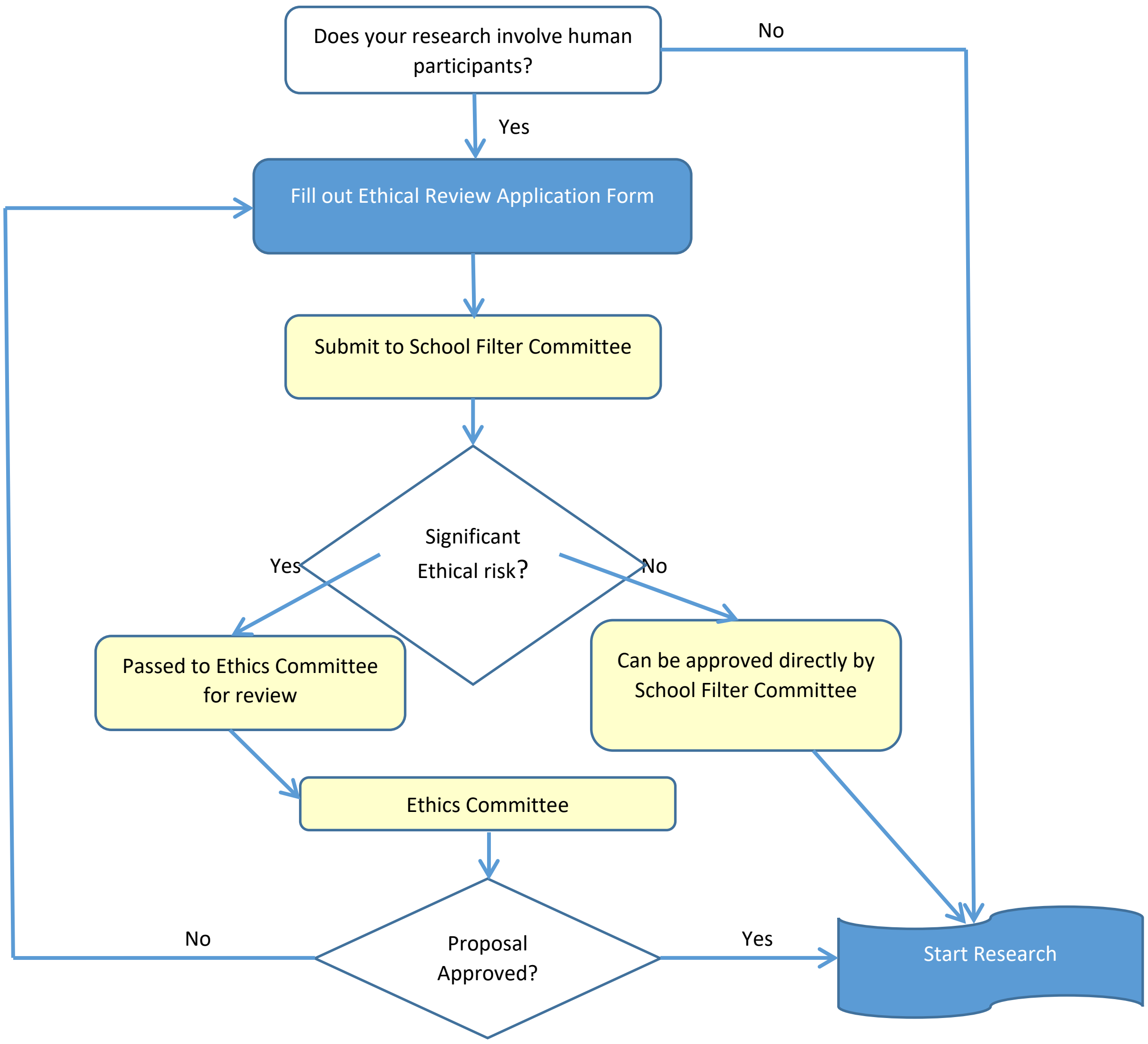

Does your research involve human participants?
No
Yes
Fill out Ethical Review Application Form
Submit to School Filter Committee
Significant Ethical risk?
Yes
No
Passed to Ethics Committee for review
Can be approved directly by School Filter Committee
Ethics Committee
Proposal Approved?
No
Yes
Start Research

Figure 2: Process chart for seeking Ethical Approval

Ethics Application Checklist

To be submitted alongside the ethics application.

Please complete the below checklist, ticking each item to confirm that it has been addressed.

1. I agree to obtain informed written consent from all human participants aged over 18	☐
who are involved in this research (or if circulating digitally, I will ensure that informed	
consent is completed, and will have the participants indicate their informed consent	☐
by continuing with their study engagement).	
2. I agree to obtain informed written consent from the parents of anyone aged under 18	
in this research (or from the schools if appropriate), and informed written assent from	
those under 18 in this research.	
3. I include a letter of agreement from a clinically responsible individual agreeing to	☐
(where appropriate) help me recruit/provide clinical support in the event that	
participants become distressed/host the study data collection.	
4. I append a letter of agreement from an external institution or organisation agreeing to	☐
host the study.	
5. I agree to comply with NCI's Data Retention Policy.	
6. I have appended a) information sheet, b) consent form/assent form, c) debriefing	
sheet.	☐
7. I have provided details of how non-anonymised data will be stored, in a safe and	
encrypted manner.	☐
8. I have included my contact details and those of my supervisor (where appropriate). I	☐
have only included my NCI email address and not included any personal contact	
information.	☐
9. I have given sufficient details on the proposed study design, methodology, and data	
collection procedures, to allow a full ethical review, and I understand that my failure	☐
to give sufficient detail may result in a resubmission being required.	
10. I understand that if I make changes to my study following ethical approval, it is my	
responsibility to seek an ethics amendment if the change merits ethical consideration.	☐
	☐

National College of Ireland

Human Participants Ethical Review Application Form

All parts of the below form must be completed. However in certain cases where sections are not relevant to the proposed study, clearly mark NA in the box provided.

Part A: Title of Project and Contact Information

Name

Joswel Bautista

Student Number (if applicable)

X19369011

Email

X19369011@student.ncirl.ie

Status:

Undergraduate **X**

Postgraduate ☐

Staff ☐

Supervisor (if applicable)

Lisa Murphy

Title of Research Project

Kohi For Productivity

Category into which the proposed research falls (see guidelines)

Research Category A **X**

Research Category B ☐

Research Category C ☐

Have you read the NCI Ethical Guidelines for Research with Human Participants?

Yes **X**

No ☐

Please indicate any other ethical guidelines or codes of conduct you have consulted

N/A

Has this research been submitted to any other research ethics committee?

Yes ☐

No **X**

If yes please provide details, and the outcomes of this process, if applicable:

N/A

Is this research supported by any form of research funding?

Yes ☐

No **X**

If yes please provide details, and indicate whether any restrictions exist on the freedom of the researcher to publish the results:

N/A

Part B: Research Proposal

Briefly outline the following information (not more than 200 words in any section).

Proposed starting date and duration of project

October 2022 – May 2023

The rationale for the project

Covid started in 2019 it has now been 3 years since it has passed. As we are stuck at home for those few years some have been less productive and been emotionally drained being isolated for a very long period of time to prevent the spread of the virus. Everything has now been slowly shifting to in person rather than being remote. For example, due to the shift from remote classes to classes being in person caused by covid peoples schedule can now shift which some people may not be used to. There was also a 25% increase in anxiety and depression during covid which is why it is also important to know your mental state.

Which is why I want to create a mobile app that can encourage users to be more productive while being able to take care of there mental health which this app aims be a self-care productivity app.

The research aims and objectives

My application is a mobile application that is designed to set and notify the user per upcoming deadlines on a task. A study timer, after the timer has finished it will give the user some kind of reward to encourage them to study more. There will be a diary to record small notes quickly when needed. A mood tracker will be implemented to keep track of how they are currently feeling, in order to do this, it will ask the user a short question about their mental health and record this. Lastly, I will have the analytics which will show the data of everything for example how long they used the study timer, current mood through the week and the tasks completed on the timetable. The core features to make this unique from other productivity apps out there are the rewards system and the mood tracker.

The aim for this project is to design, implement evaluate and test the mobile application.

The research design

Kohi will be designed around using Figma which help me design the layouts of the application.

The application will be tested and checked for errors using Unit Testing and QA Testing.

The evaluation will involve human participants (over 18 years old) which will provide feedback about the functionality of the mobile application.

The participants permission will be asked when gathering the data used for the project for research purposes.

A survey will be done online to collect the feedback from the users. The survey will be done anonymously and therefore no personal data will be collected.

The collected feedback will be analysed and used to improve the application. An overall description of answers received in the survey will be presented in a report.

The research sample and sample size

Please indicate the sample size and your justification of this sample size. Describe the age range of participants, and whether they belong to medical groups (those currently receiving medical treatment, those not in remission from previous medical treatment, those recruited because of a previous medical condition, healthy controls recruited for a medical study) or clinical groups (those undergoing non-medical treatment such as counselling, psychoanalysis, in treatment centres, rehabilitation centres, or similar, or those with a DSM disorder diagnosis).

Kohi will be available for download to anyone who wishes to participate in the study and he/she is over the age of 18.

A target of 20 participants that use the application and answer the survey is aimed.

Social media and email will be used to invite people to take part in the study. The participants are selected on a voluntary basis.

If the study involves a MEDICAL or CLINICAL group, the following details are required:

a) **Do you have approval from a hospital/medical/specialist ethics committee?**
   If YES, please append the letter of approval. Also required is a letter from a clinically responsible authority at the host institution, supporting the study, detailing the support mechanisms in place for individuals who may become distressed as a result of participating in the study, and the potential risk to participants.
   If NO, please detail why this approval cannot or has not been saught.
b) **Does the study impact on participant's medical condition, wellbeing, or health?**
   If YES, please append a letter of approval from a specialist ethics committee.
   If NO, please give a detailed explanation about why you do not expect there to be an impact on medical condition, wellbeing, or health.

The nature of any proposed pilot study. Pilot studies are usually required if a) a new intervention is being used, b) a new questionnaire, scale or item is being used, or c) established interventions or questionnaires, scales or items are being used on a new population. If no such study is planned, explain why it is not necessary.

N/A

The methods of data analysis. Give details here of the analytic process (e.g. the statistical procedures planned if quantitative, and the approach taken if qualitative. It is not sufficient to name the software to be used).

The survey will consists of yes / no questions with answers on a scale.

The reporting of the results will be dfone after the data is aggregated across all the participents.

None of the data or quotes will be attributed to any individual participant.

Study Procedure

Please give as detailed an account as possible of a participant's likely experience in engaging with the study, from point of first learning about the study, to study completion. State how long project participation is likely to take, and whether participants will be offered breaks. Please attach all questionnaires, interview schedules, scales, surveys, and demographic questions, etc. in the Appendix.

A brief description of the purpose of the study is provided to the participants at the beginning of the study. Their consent to take part in the study is requested and collected.

The participants will be asked to download the application and use it on their mobile phone. The participants are given three tasks to perform that involves the use of the application.

Once the tasks are completed, they are asked to answer the survey.

It is envisaged that the entire evaluation process of the application will not take more than 20 minutes.

The survey to be answered by the participants and the consent request are provided in A and Appendix B respectively.

Part C: Ethical Risk

Please identify any ethical issues or risks of harm or distress which may arise during the proposed research, and how you will address this risk. Here you need to consider the potential for physical risk, social risk (i.e. loss of social status, privacy, or reputation), outside of that expected in everyday life, and whether the participant is likely to feel distress as a result of taking part in the study. Debriefing sheets must be included in the appendix if required. These should detail the participant's right to withdraw from the study, the statutory limits upon confidentiality, and the obligations of the researcher in relation to Freedom of Information legislation. Debriefing sheets should also include details of helplines and avenues for receiving support in the event that participants become distressed as a result of their involvement in this study.

It is not envisaged that there is any risk of harm or distress to participants.

Participants are allowed to withdraw from this study anytime if they wish to do so.

Do the participants belong to any of the following vulnerable groups?

(Please tick all those involved).

- ☐ Children;
- ☐ The older old (85+)
- ☐ People with an intellectual or learning disability
- ☐ Individuals or groups receiving help through the voluntary sector
- ☐ Those in a subordinate position to the researchers such as employees
- ☐ Other groups who might not understand the research and consent process
- ☐ Other vulnerable groups

How will the research participants in this study be selected, approached and recruited? From where will participants be recruited? If recruiting via an institution or organisation other than NCI please attach a letter of agreement from the host institution agreeing to host the study and circulate recruitment advertisements/email etc.

Emails and social media will be used to invite people to take part in the study. The participants are selected on a voluntary basis.

What inclusion or exclusion criteria will be used?

Participants must be 18 years of age or above

How will participants be informed of the nature of the study and participation?

An invitation will be sent through an email or a text describing the study purpose and a procedure will be sent to the invited participants.

Does the study involve deception or the withholding of information? If so, provide justification for this decision.

N/A

What procedures will be used to document the participants' consent to participate?

The online survey will include question requesting consent to take part in the study

Can study participants withdraw at any time without penalty? If so, how will this be communicated to participants?

Yes / Will be communicated in the text/email

If vulnerable groups are participating, what special arrangements will be made to deal with issues of informed consent/assent?

N/A

Please include copies of any information letters, debriefing sheets, and consent forms with the application.

Part D: Confidentiality and Data Protection

Please indicate the form in which the data will be collected.

□ Identified □ Potentially Identifiable **X De-Identified**

What arrangements are in place to ensure that the identity of participants is protected?

No persnal data will be collected in the survey

Will any information about illegal behaviours be collected as part of the research process? If so, detail your consideration of how this information will be treated.

No such information will be collected

Please indicate any recording devices being used to collect data (e.g. audio/video).

No devices are used

Please describe the procedures for securing specific permission for the use of these recording devices in advance.

N/A

Please indicate the form in which the data will be stored.

□ Identified □ Potentially Identifiable □ **De-Identified**

Who will have responsibility for the data generated by the research?

The applicant for the Ethics approval

Is there a possibility that the data will be archived for secondary data analysis? If so, has this been included in the informed consent process? Also include information on how and where the data will be stored for secondary analytic purposes.

No

If not to be stored for secondary data analysis, will the data be stored for 5 years and then destroyed, in accordance with NCI policy?

X Yes □ No

Dissemination and Reporting

Please describe how the participants will be informed of dissemination and reporting (e.g. submission for examination, reporting, publications, presentations)?

N/A

If any dissemination entails the use of audio, video and/or photographic records (including direct quotes), please describe how participants will be informed of this in advance.

N/A

Part E: Signed Declaration

I confirm that I have read the NCI Ethical Guidelines for Research with Human Participants, and agree to abide by them in conducting this research. I also confirm that the information provided on this form is correct (Electronic signature is acceptable).

Signature of Applicant Joswel Bautista

Date 29/10/2022

Signature of Supervisor (where appropriate):

Date 29/10/2022

Any other information the committee should be aware of?

N/A

Reflective Journals

October

Supervision & Reflection Template

Student Name	Joswel Bautista
Student Number	X19369011
Course	BSHCSD4
Supervisor	Lisa Murphy

Month: Ocotber

What? Reflect on what has happened in your project this month? This month I have done some research and UI Design planning for my project.	
So What? Consider what that meant for your project progress. What were your successes? What challenges still remain? This progress allowed me to get approval on to do this project. The challenge that remains is starting the project.	
Now What? What can you do to address outstanding challenges? To start the project, I have made a timeline to do	
Student Signature	Joswel Bautista

November

Supervision & Reflection Template

Student Name	Joswel Bautista
Student Number	X19369011
Course	BSHCSD4
Supervisor	Lisa Murphy

Month: Ocotber

What? Reflect on what has happened in your project this month? This month I have done some research on how to use some of the tools	
So What? Consider what that meant for your project progress. What were your successes? What challenges still remain? This progress allowed me to get an idea of how to use these tools to see which is a best fit for my project	
Now What? What can you do to address outstanding challenges? The challenges is learning these tools that I have chosen which is android studio.	
Student Signature	Joswel Bautista

December

Supervision & Reflection Template

Student Name	Joswel Bautista
Student Number	X19369011
Course	BSHCSD4
Supervisor	Lisa Murphy

Month: December

What? Reflect on what has happened in your project this month? This month I have done two of the main functionalities which is the Timer and the Gacha feature.	
So What? Consider what that meant for your project progress. What were your successes? What challenges still remain? This progress allowed me to get the basic functionality for the application and I can now proceed on the reward functionality. The challenge I will face is trying to store these rewards and how the user.	
Now What? What can you do to address outstanding challenges? I will research what kind on databases I can use. So far I'm thinking of using Firebase as my database.	
Student Signature	Joswel Bautista

January

Supervision & Reflection Template

Student Name	Joswel Bautista
Student Number	X19369011
Course	BSHCSD4
Supervisor	Lisa Murphy

Month: January

What? Reflect on what has happened in your project this month? This month I have researched on being able to save and load data on my application. I have used Shared preferences to save data locally on the phone.	
So What? Consider what that meant for your project progress. What were your successes? What challenges still remain? This progress will now allow me to work on the summon feature since it is now linked with the study timer. Both activities are now communicating to each other passing the value. A challenge I will now face is doing the Gacha feature on the application	
Now What? What can you do to address outstanding challenges? Here I will research some algorithms that I can use for it . So far I will be using the random algorithm to develop this feature	
Student Signature	Joswel Bautista

February

Supervision & Reflection Template

Student Name	Joswel Bautista
Student Number	X19369011
Course	BSHCSD4
Supervisor	Lisa Murphy

Month: February

What? Reflect on what has happened in your project this month? This month I have implemented the Gacha feature into my application. The basics of being able to randomly get a reward and save the rewards of what you have gotten.
So What? Consider what that meant for your project progress. What were your successes? What challenges still remain? This means I can implement in all the other rewards in the future. There are still some challenges in this which is the Gacha rate of the rewards. I Need to implement non duplicates which is if the reward has already been given It needs to be removed so its not given again. But with this I plan to now move on how to track the user’s engagement using analytics. This will show how long the user has been productive through the day. This will be a challenge because I will need to find a library that supports this which will require some research to find out which fits best for my project.
Now What? What can you do to address outstanding challenges? To address this problem I plan to look through online for some libraries that will fit my project.

Student Signature	Joswel Bautista

March

Supervision & Reflection Template

Student Name	Joswel Bautista
Student Number	X19369011
Course	BSHCSD4
Supervisor	Lisa Murphy

Month: March

What? Reflect on what has happened in your project this month? This month have met up with my supervisor gave me back feedback. This month I have fixed some on going errors along with updating my final report on the project. Features that I have been able to add is the diary feature where users can log in anonymously. A CRUD feature has been implemented using the firebase storage.

So What? Consider what that meant for your project progress. What were your successes? What challenges still remain? Having this CRUD feature I am now able to add a feature where user will also be able to sign up as a user so they can save their data across different devices. This feature also works offline so when the user connects back to the internet it is automatically synced up. This allows me to now save their data and be able to display it the challenge is now being able to make the analytics page using their data. Right now, I need to save the data on cloud now and display it	
Now What? What can you do to address outstanding challenges? To address this challenge, I plan to To address this problem, I plan to implement the cloud storage to connect to the account so I can display them to the user using firebase analytics.	
Student Signature	Joswel Bautista

April

Supervision & Reflection Template

Student Name	Joswel Bautista
Student Number	X19369011
Course	BSHCSD4
Supervisor	Lisa Murphy

Month: April

What? Reflect on what has happened in your project this month? This month I have improved the application GUI and functionality to the application making it more seamless.	
So What? Consider what that meant for your project progress. What were your successes? What challenges still remain? This allows me to go and do bug fixes and clean and optimise my code. The challenges that still remain are testing my application using black and white box testing.	
Now What? What can you do to address outstanding challenges? To address this I plan on using test cases in my application and testing it manually myself after my application has been updated.	
Student Signature	Joswel Bautista

National College of Ireland

Project Proposal

21/10/2022

BSHCSD4

Software Development

2019/2023

Joswel Bautista

x19369011

x19369011@student.ncirl.ie

Contents

Objectives

This project is a self-care productivity app on mobile created for Android users that encourages people to be productive with their time while being able to take care of themselves mentally as things can be very difficult when trying to be productive. The objective is to help people such as students returning to college due to covid or those who struggle to take care of themselves and manage themselves being less productive through the day. My Objective in this project is to create a visually friendly app that people will use to be more productive while taking care of their mental state. To make this project my objective is to use many tools to create this app.

The objective of this project is that it will allow users to set and notify the user per upcoming deadlines on a task that's needed to be done to encourage the user to be more productive. It will have a reminder to ask about your mood everyday throughout the week. This can be after you finish a deadline or a task that's been set. There will be a feature where it will communicate to the user by asking you questions through the day about your mood which will then keep track of this through the week. By the end of the day, it will show how you are feeling by giving a quick notification of how you felt during the day. There will be a feature where you can write in a diary about your day or just to keep notes of things that has happened during the day. There will also be a Study Timer to help you be more productive for your work. When the timer is over it will reward the user with a prize. These prizes can be collected to encourage the user to study more and collect all the collectable prizes. These prizes can be an achievement of some sort which will encourage the user to collect all of them. Lastly which is one of the most important is the analytics on everything. This will show your data on everything through the week, this includes how your mood was through the week giving you an average mood through that week, how long you studied for using the timer, the amount of time spent on certain tasks and things you have completed and not completed through the week.

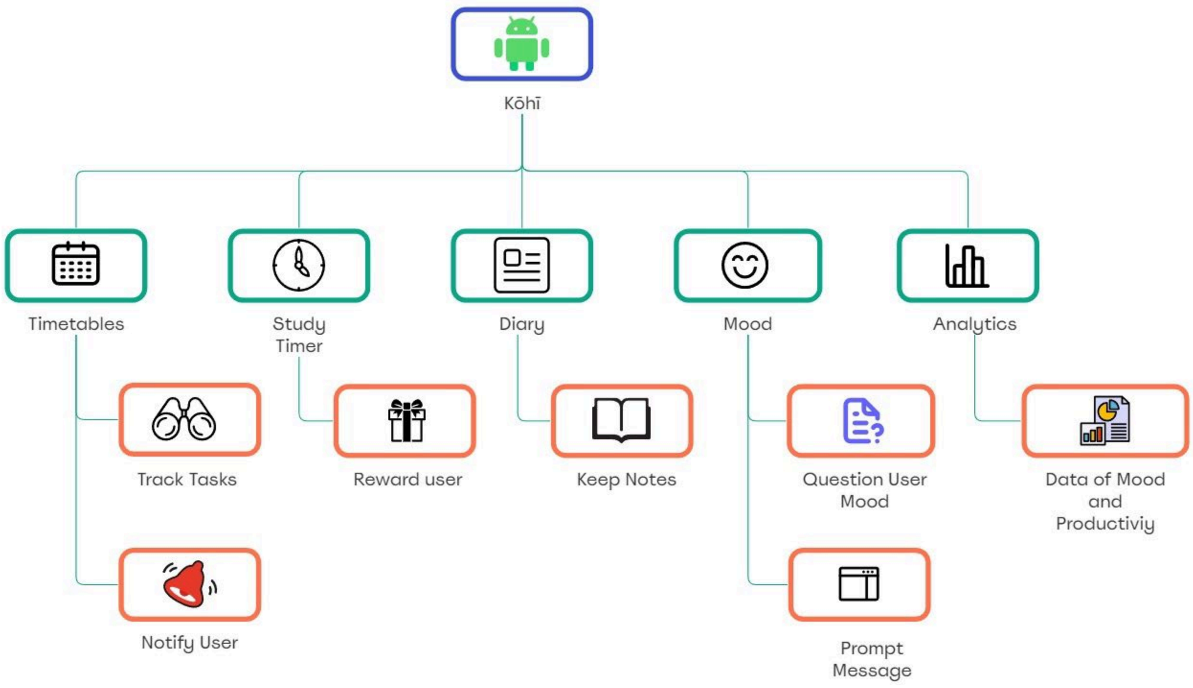
Background

I undertook this project because I wanted to try something new that has not been thought in college which is UI/UX designing and mobile application development. This will make it challenging to do since its my very first mobile application I will be creating. I wanted to choose something that I will enjoy creating. The tools I plan on using for developing this android application is Flutter, Android Studio or Cordova. To meet the objective set for creating a visually appealing mobile app I will have to research UI/UX designing. From my internship I have attended the UI/UX Design meeting which they have set up for the people working on the front end of the project I was a part of. Using the things that I've learned from this I plan to do more research to make my app visually appealing for the experience of the user. Things I have learned is to set my style guide of my project, this helps me make sure to set the standards of how my application will look like which is very important. To meet the objective when creating notifications to prompt the user about there task nearing the deadline, asking for their mood, study timer and the diary I will use android libraries online that I have researched to achieve these such as Koin which is a navigation library.

State of the Art

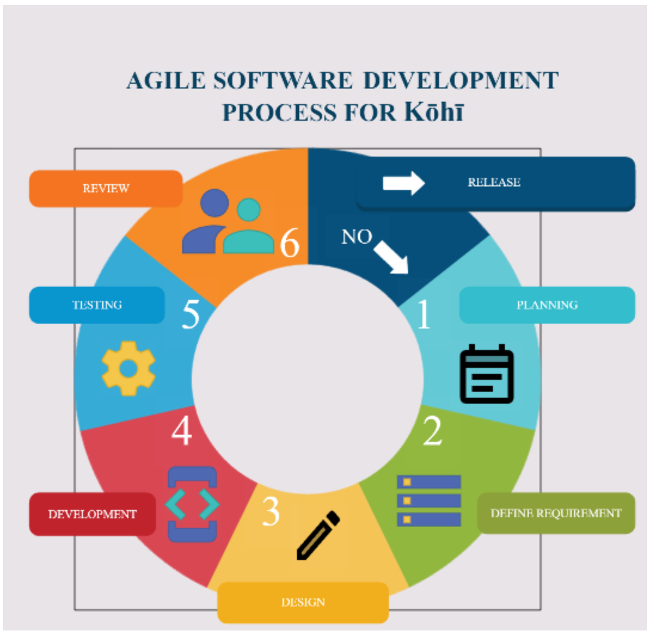
Similar apps out there exist out there which is why I have combined different functionality to my app to make it unique to the ones that already exists. Such as asking questions to its user, this has inspired me by the app called Officevibe which is an application used in my internship used to find how the employees feel about their work. The timetable functionality is inspired by Jira which is a project tracking software this helped organise the work between sprints. Now similar apps exist out there as a mobile application such as Forest which is one of my favourite productivity app. It is a popular productivity app that helps people beat their addiction on the phone and manage their time. The reason its my favourite is due to it rewards the user when being productive. What makes mine stand out to this and other existing productivity app is that mine will communicate to the user asking about their mental state after the task has been done. This will make the user aware of their mental state at the end of there day and the week. For example, the user will use the study timer and feel unhappy after a session, it will prompt back to recommend them to take a break. This project is different from productivity apps because it will also allow users to be self-aware of their own mental state after a task hence the self-care part of the project. The productivity part is the timetable, the study timer and the diary encouraging the user to be more productive to do there work and be aware of their tasks. This sitemap that I had created will show the functionality I plan to have in the application. Sitemap created with

Miro.



Technical Approach

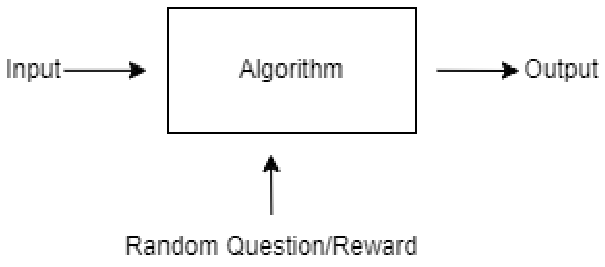
I will take the Software Development Life Cycle (SDLC) approach in order to do this project. The reason for this is that it covers a lot of the factors that's needed in the development of a mobile application. The reason of choosing an Agile methodology is because it will give me room for error down the line. How I will identify the requirements is by knowing, what is needed to be implemented in the application, how long will it take to design, how long will it take to complete the functionality, how long will it take to test the functionality, how much will the application change after the testing and the deadlines due for the project. To break down all these requirements for the project I will use an Agile methodology for my project's requirements. This methodology covers how I can break my project tasks, activities, and milestones. The Agile Process was created with Creately.



Stage one is **Planning** which I will determine the scope and objective of the project making an appropriate schedule for the functionalities breaking down the requirements that may be needed. The 2nd stage, which is **Defining the Requirements** needed, in this stage I will outline what are the requirements needed for this application. I will research what tools and resources I may need in order to meet the requirements of the project. After planning and outlining the requirements, I will then start **Designing** the application which is the 3rd stage. It will give me an idea of how the application will visually look. This will give me a guide on how to start developing the application. After getting a general design and layout of the application, the 4th stage would then be the **Development** of the functionality of the application. I will start to develop the requirements that's needed for the app to function. This stage will take the longest since it carries all the heavy work such as creating the timetable manager. Once the functionality is developed, the 5th stage which is **Testing,** Here I will then test it for bugs, glitches, and any other user experience issues that may be found. After which is the final stage called **Review**, Here I will review if the testing went smoothly for the functionalities that was developed. If the functionality did not go well, I repeat the cycle until I am satisfied with the release. The reason I choose agile is because it will give me room for error along the way when developing my application.

Technical Details

The core feature of the app are the mood tracking and the rewards system. The reason for this is because its what makes it unique to the apps currently out there on the market. The Timetables and the study timer are there to make it that the core features can be used. Since productivity apps are about calendaring, to-do list and reminder apps adding these makes it unique to the others. When implementing these features, I plan on using Java. From research online the tools that mainly use java and kotlin are software's such as Android Studio. I plan on using libraries that will help me implement UI designs, databases, file storage, push messages, analytics and many more. Some important algorithms I plan on using are some sorting algorithms for the timetables and maybe randomized algorithms in order to do the rewards and questions to be given to the user about their mood



Depending on the answer the user gives to the algorithm, it will prompt a random message based on there current mood after completing a task on the timetable or studying. When the user uses the study timer it can also randomise a reward based on how long they have studied for.

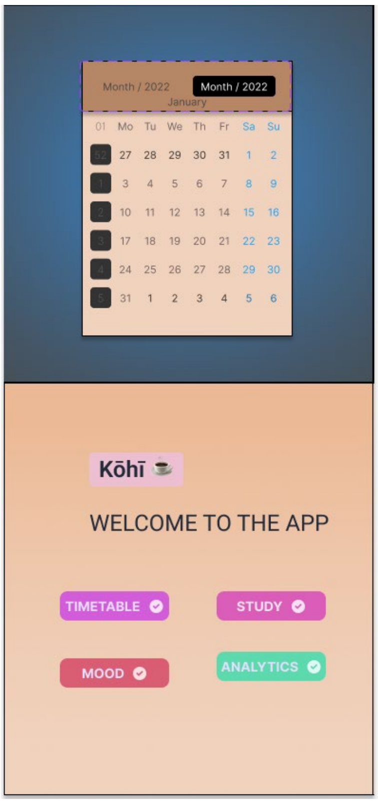
When storing the data about the user I plan on using the cloud to store the data or store the data internally on the phone. Storing this data will then help me create my analytics part for the app. When researching this I came across Mixpanel, Google analytics and Localytics which will help me achieve this objective. As for the Diary and study timer I plan on researching ways to achieve this. As for the reward system to achieve this objective I plan on using the to store the achievements and fetch these achievements later on when the user completes an achievement. I plan on researching further on all these to improve my application.

Special Resources Required

Before starting the application, itself I plan on making a design layout. This will help me map out and have a blueprint ready of the app. is one of the tools I plan on using when creating the design of the application. Special online websites such as colorhunt.co will help me get a colour theme for the application.

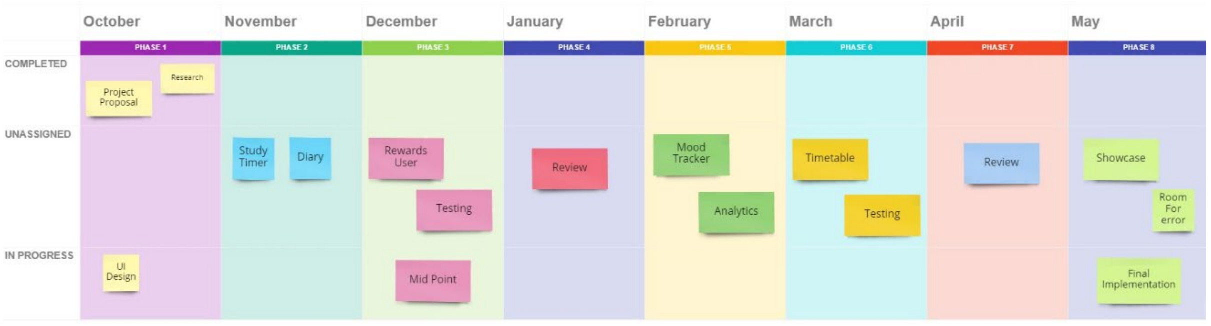
I will be using Krita which is an art software I plan on using to design the application. I also plan on using free tools online such as Pixlr which is an alternative to photoshop. Here is a sample layout I plan. The color scehme is not finalised as I need to research more about UI designing. For now I have went with a matcha/matcha blue coffee color pallete which can change in the future.

<#151618>
<#556b2f>
<#a9a454>
<#efe4d4>
<#b07946>



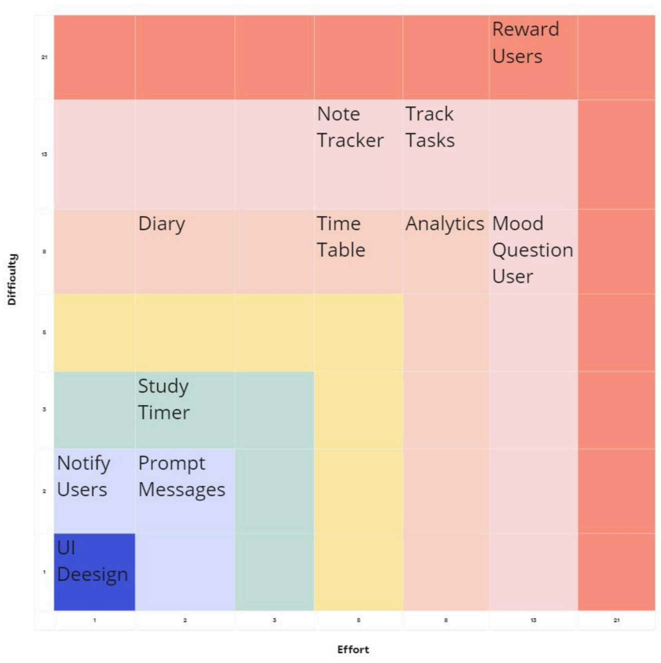
Project Plan

Since there is only 30 weeks left, I have split my tasks into months. These consists of the implementation of the application and testing. Every third month I will have a Review month based on the progress and work done from the past two months. As we can see on the timeline, I have split the tasks into sprints called phases to help me organise myself. These sticky notes can be dragged anytime when the task is complete or is in progress.



The tasks have been chosen mostly based on the difficulty. I had decided to do the easy implementation for the app first then work towards the difficult part in order to get some prototype ready for the midpoint presentation. As the mid-point is finished, I plan on working on the hard part of the application which is to develop the mood tracker and the analytics for it. I have left some room for the final phase which is in may leaving a room of error along my timeline in case there are some tasks that may need more time in order to develop the features.

The image below shows the difficulty and effort it will take in order to develop the functions of the application. Some can change once I start developing the application itself.



Steps of implementation:

Phase 1: This sprint consisted of research, project proposals and UI designing.

Phase 2: In this sprint I plan on starting to develop the Study timer and diary. I started with these tasks so I can have something on my application. I plan on spending half a month on developing the study timer and the diary.

Phase 3: During this phase I plan to implement one of the core features of the application which is the reward system. I also plan on doing some testing when completing this task along with the midpoint presentation.

Phase 4: In this phase I plan to review the application so far to see if there's any flaws or implementations that's needed in order to improve the application.

Phase 5: This phase is where it starts to pick up the difficulties as I plan to implement the Mood tracker and analytics of the application. This may take more than a month, but I plan on spending half a month on each of the tasks.

Phase 6: During this sprint I plan on developing the Timetable along with testing the mood tracker and analytics functions.

Phase 7: During this month I plan on reviewing the whole application and see if there are any changes needed to be made.

Phase 8: This is the final sprint which is the final implementation of the application and the showcase. Some room of error has been made in case some of the sprints may take longer than it needed to be giving me space to work around with.

Testing

In this I plan on doing two types of testing which is Unit Testing and QA Testing. Unit testing will be done in order to test if code is working as its intended while QA testing will be done for Quality Assurance finding and fixing errors in the code while documenting the process and what has caused the error.

The system will be evaluated by comparing data to my system using system testing and integration testing. System testing will be done once the application is developed completely. Integration testing will be done on the individual modules that have multiple functions. These testing are done so that the system can be evaluated and meet the specified requirements needed.

I will evaluate the system by having an end user use the application for a certain period of time getting their permission to record the data when they use the application, I will let them use the functions in the application and see if they work.

Other materials used

Any other reference material used in the project for example evaluation surveys etc.

Signature

Joswel Bautista