BSc in Software Engineering

School of Science, Engineering and Environment

Final Year Project Dissertation

Flutter Mindfulness Application

Author: [redacted]

Supervisors: Julian Bass, Tarek Gaber

2020-2021

Abstract

Due to the pandemic, lots of people faced mental health issues, such as stress and anxiety. The project aims to build a mindfulness application using the Flutter framework to help people stay present with their feelings and emotions. Moreover, since flutter is a new framework and a competitor of the React native, the project also illustrates the positive and negative aspects of the framework. The requirements identified according to the research and common mobile application needs. Moreover, according to the requirements, the application implemented. Furthermore, the flutter_bloc package is used to adopt the bloc state management technique. Overall, using the library is a good start for learning the bloc pattern, and separate the logic from the presentation layer. However, after gaining experience with the flutter_bloc, the technique can be implemented from the scratch using the _bloc package. Furthermore, unit, widget, prototype, integrational and user testing conducted. As a result, all of the tests passed. Moreover, the design and functionality liked by the internal testers. Finally, the app released to the Google Play Store for review before the full release. Most of the applications on the marketplace regarding mindfulness and mental health developed using React native. This project illustrated that, even though the Flutter framework is newer, it can still provide as good results as React Native. However, the lack of developers in the industry must be considered before learning the framework.

Context

Chapter 1

Introduction

The project aims to create a mindfulness native cross-platform mobile application using Flutter UI development kit, Dart language and Firestore authorization and cloud storage. The process has begun by producing the project backlog and designing the pages of the mobile application. Moreover, the implementation of the app has done by using Android Studio and XCode. Widget, User, Integration level and the Unit tests have used to test the application. Finally, the application has published to the google play and App stores.

1.1 Motivation

People started struggling with their mental being during the Coronavirus pandemic. According to the Health Foundation, 69% of adults feels worried about the effects of the pandemic. Moreover, 59% of the people feel anxious about the financial and coping mechanism, such as exercise, social life and hobby loss. (Marshall, 2020) This situation leads people to focus more on their mental wellbeing by using mental-health mobile applications. However, not every application allows people to reflect on their thoughts and feelings. Instead, most of them rely on guided meditations to deal with their mental health problems. (Plaza, Demarzo, Herrera-Mercadal, & García-Campayo, 2013) (Roquet & Sas, 2018)

The motivation behind developing a mindfulness application is creating something that not only will help me learn new technologies but also, I will be able to help people who find it hard to stay mindful. My goal is to help people improve their lives and give them space to be present with their thoughts and feelings, especially in these uncertain times. The app will use the journaling feature, which will encourage users to reflect on the current emotions, accept them, and finish the process with gratitude. Moreover, the user will be able to do unguided meditations using nature sounds. The unguided meditation will give them the ability to be present with themselves without any distractions or guidance.

Flutter is a new framework, which is not used in most mindfulness mobile application. By creating a flutter mindfulness application, I will be able to see the performance of the Flutter Framework compared to React native applications on the marketplace.

1.2 Objectives

The following list contains the objectives of the project,

- Objective 1: Learn Flutter and Dart.
- Objective 2: Design the user interface
- Objective 3: Implement the Mindfulness Application
- Objective 4: Implement the Unit, widget, integrational and user testing
- Objective 5: Deploy the application

The project is going to be started by creating a project backlog for each sprint. Next, the architecture, requirement and user interface design are going to be generated. Following that, the third objective is going to be implemented using Android Studio and XCode. After executing the tests, the app will be deployed to the Play Stores.

1.3 Overview of the Report

The report starts with the literature review to discuss native-cross platform application With the native, hybrid and web mobile applications regarding to their performance. Moreover, it includes a discussion of the react-native and flutter frameworks. The literature review then focuses on the Flutter framework, dart language and the state management techniques, such as Redux and bloc. Moreover, the SQLite and Firebase services compared to find the best database. Finally, mindfulness theme discussed with the comparison of the applications on the marketplace.

This then followed by the methodology of the project. This section includes information about the chosen development environment, methodology, testing, design pattern, programming language and database.

The next chapter explains the requirements of the mobile application and demonstrates the UI and database design.

The report then focuses on the implementation of the requirement, issues encountered, and the results achieved.

Testing and analysis section concentrates on the implementation of the widget, unit, integrational, user and prototype tests with the discussion on the problems faced during the implementation.

Next, the critical evaluation chapter reviews the project plan and includes a reflection on the objectives.

The report concludes with the conclusion to discuss the findings and the suggestions for the future work.

Chapter 2

Literature Review

In this chapter, a literature review conducted. The research includes various type of mobile development technologies, the comparison between a native and cross-platform mobile development and between the cross-platform technologies such as Widget-based, Integrated and hybrid. The next part contains information about Flutter and design patterns Redux and Bloc. Finally, the last section holds research about mindfulness and literature on the mindfulness apps in the industry.

2.1- Mobile Application Development

The first smartphone IBM Simon invented in 1970. It is the first mobile phone that contained a touch screen as well as game and calculator apps. However, until 2007, mobile applications had not been a popular subject in the IT industry. Apple released the iPhone in 2007 and started the modern mobile application development. ((Jabangwe, Edison, & Duc, 2018) Below Figure 2.1 illustrates different types of mobile applications.

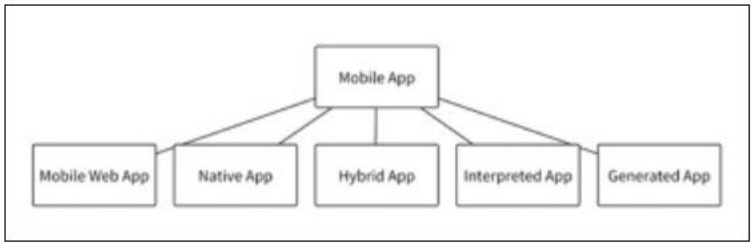


Figure 2.1: Types of mobile applications (Nunkesser, 2018)

2.1.1- Native Mobile Applications

The development process of the mobile application started with the native approach. The native mobile applications developed for a specific operating system Google's Android OS (Operating system) or Apple's IOS (Operating system). (Jabangwe et al., 2018) Because native applications are platform-specific, the usability of the apps is excellent. However, in native

apps, developers lose a large number of users by not deploying the mobile app to both Google and Apple Stores. To deal with this issue, developers need to implement and maintain two different code bases for IOS and OS. (Shah, Sinha, & Mishra, 2019) According to Joorabchi, Mesbaha & Kruchten, developing and maintaining two code bases is challenging, time-consuming and costly. (Joorabchi, Mesbah, & Kruchten, 2013)

2.1.2- Mobile Web Applications

Mobile web applications allow cross-platform development since they are accessible through the browser of the devices. They use technologies such as HTML, CSS and JavaScript. Even though web apps give developers ease while developing multi-platform apps, they cannot provide a user experience as well as native. (Jabangwe et al., 2018) Mobile web apps consist of the client, working application and database tiers. The web server is responsible for the data access. Therefore, internet connection, availability and the reliability of the network are very critical. (Shah et al., 2019) , (Jabangwe et al., 2018).

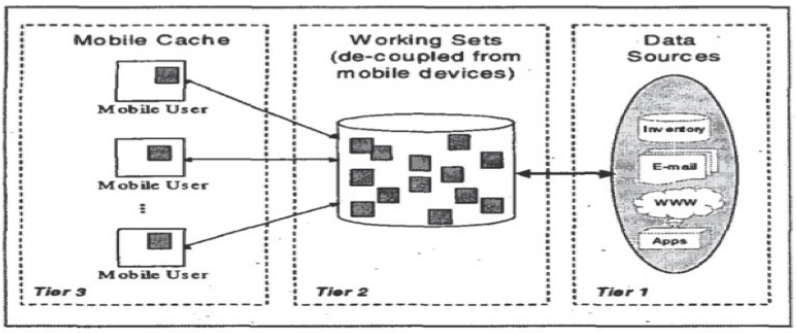


Figure 2.2: Mobile Web Application Architecture (Shah et al., 2019)

2.1.3- Hybrid Mobile Applications

Hybrid apps are useful for developing cross-platform mobile applications. They combine the architecture of the native application with HTML, CSS and JavaScript technologies to make the app accessible for both OS. The developers write a single code base to build the application, and each operating system calls the native portion of the mobile application using API calls. Finally, the WebView part translates the API calls to the specific platform. (Shah et al., 2019) Unlike native and web applications, hybrid apps provide features such as Bluetooth, GPS, and because the user can install these apps, the system does not need an internet connection. (Biørn-Hansen et al., 2020)

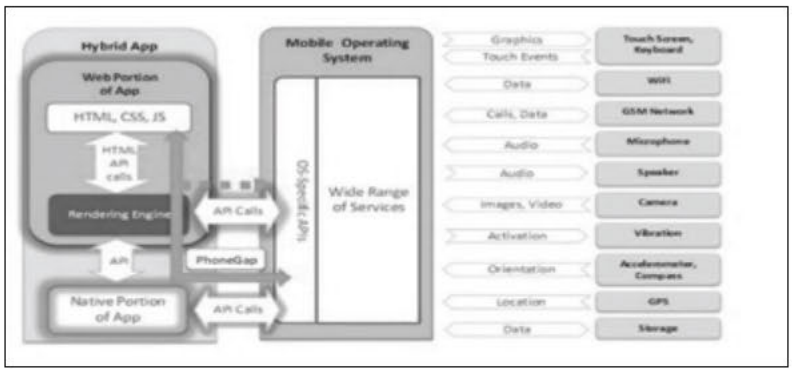


Figure 2.3: Mobile Web Application Architecture (Shah et al., 2019)

2.1.4- Interpreted Mobile Applications

The interpreted mobile application does not contain the WebView part. These apps have their own interpreted runtime component called JavaScript engine. The bridge element helps the application access the different APIs and UIs according to the operating system and process them using the JavaScript engine, as shown in Figure 2.4. On the other hand, the development of these apps highly dependent on their features provided by the specific development environment. (Rahul Raj & Tolety, 2012)

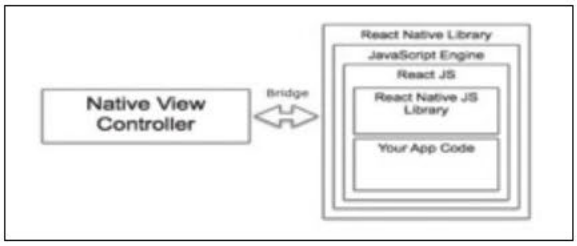


Figure 2.4: Interpreted mobile application architecture (Shah et al., 2019)

2.1.5- Widget Based Mobile Applications (Cross-compiled)

Widget Based Mobile Applications are native cross-platform apps that are made up of stateless and stateful widgets. Moreover, they do not rely on system components, unlike other cross-platform applications. Cross compiled apps use their own rendering Skia Graphics Engine to create a user interface in the quality of the native apps, as shown in Figure 2.5 Widget Based Mobile Application Architecture. (Biørn-Hansen et al., 2020)

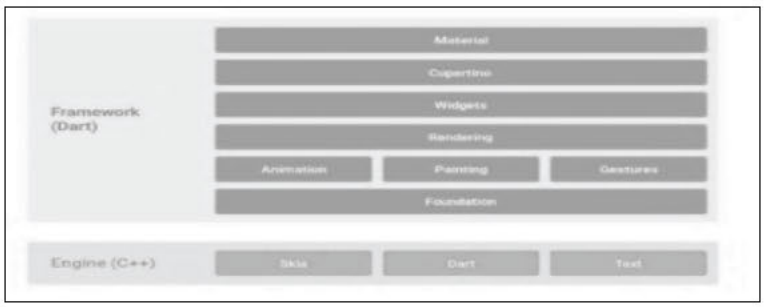


Figure 2.5: Widget Based Mobile Application Architecture (Shah et al., 2019)

2.2- The Comparison Between Native and Cross-platform

2.2.1. Mobile Applications

Native and Cross-platform mobile apps are different types of mobile applications. Native apps are platform-specific. The developers need to choose to develop the app either for the Apple Operating system using Swift or Objective C or Android OS using Kotlin or Java. The main frameworks are Android studio for Android development and XCode for IOS. Cross-platform mobile applications include web, hybrid, interpreted and widget-based. (Shah et al., 2019) Figure 2.6 shows the comparison between a native and cross-platform app. (Kumar, 2014)

	Native	Cross-Platform
Multiple OS Support	No	Yes
User Interface Quality	High	Medium to High
Performance	High	Medium
Cost of Ownership	High	Medium
Application Updates	Native Market	Native Market
Application Maintenance	High	Medium
Development Languages	Java, C, C++, Objective C, Objective C++	Java, HTML, CSS, JavaScript

Figure 2.6: Comparison between native and cross-platform (Kumar, 2014)

2.2.1.1- Multiple OS support

The difference between a native and cross-platform is multi-platform support. A native app is a mobile app developed for a specific Operating system. To achieve cross-platform

support, developers need to build two different code bases. (Shah et al., 2019) However, this approach is costly and time-consuming. (Joorabchi et al., 2013) Moreover, companies need to hire more than one people to develop codebases, and the changes must be implemented in both of them separately. Cross-platform applications, on the other hand, support multi-platform development and are cost-effective and maintainable compared to native applications.

2.2.1.2- User Interface Quality

According to Figure 6, the quality of the user interface of a native app is better than a cross-platform mobile application. A native app can achieve the rendering of the most high-end graphics because they are being built in the device's native language and installed to the device. (Kumar, 2014) This is the advantage of native mobile applications. Contrarily, cross-platform apps use different rendering technology and have various level of user interface quality. A cross-platform app can be very similar to a native but cannot provide the same user experience. (Shah et al., 2019)

2.2.1.3- Performance

Native mobile applications are designed for a specific operating system and perform better than a cross-platform. Figure 2.7 compares the performances of native and cross-platform. (Shah et al., 2019) When comparing the installation and start-up consuming time, CPU, memory usage and network flow, the native app with the same functionality provides better performance than the hybrid. Because the hybrid app uses WebView to convert a JavaScript code to a native, CPU usage is larger. Moreover, the download time of the hybrid app is longer due to its plugins. (Kumar, 2014)

Performance parameter	*Hybrid app*	*Native app*
Installation Consuming Time (Second)	9.37	7.64
Start-up Consuming Time (Second)	1.58	1.11
CPU Occupancy Ratio (%)	11.76	5.54
Mermory Occupancy (MB)	101.66	58.77
Battery Temperature (℃)	45.32	37.54
Network Flow (KB)	330.56	323.89

Figure 2.7: Performance comparison between native and hybrid (Kumar, 2014)

2.3- The Comparison Between the Cross-Platform Mobile Applications

Flutter is a new and the only widget-based UI tool kit in the industry. Interpreted mobile application framework such as Titanium, react-native are more popular and older than Flutter applications. Moreover, Apache Cordova and Adobe PhoneGap are well-known frameworks for Hybrid apps. Figure 2.8 illustrates the comparison between cross-platform application types. (Shah et al., 2019)

Decision Parameters	Native	Web apps	Hybrid apps	Interpreted apps	Widget Based (Flutter)
UI / UX	Excellent	Moderate	Moderate	Fairly Good	Very Good
Potential Users	Limited	Maximum	Large	Large	Limited
Development Cost	High	Low	Low	Moderate	Moderate
App Security	Very High	Very Low	Low	Moderate	High
Ease of Update	Low	High	Varying	Varying	Moderate
Implementation Complexity	High	Low	Moderate	Low to Moderate	Moderate
Access to Native APIs	Yes	Yes, but only with HTML5	Yes, through plugins	Yes	Yes
Preferred Development environments	Android Studio, XCode, Momentics	Visual Studio Code, PhpStorm, WebStorm	Eclipse Hybrid Mobile Tools (THyM), Intel XDK,	Nuclide using Atom IDE, Appcelerator Studio	Android Studio, IntelliJ
License	Android: Open-source, iOS: Closed-source	Open-source	Generally open-source	Open-source	Open-source
Language	Java/Kotlin, Swift/Objective-C, etc.	HTML, CSS, JavaScript, TypeScript, etc.	HTML, CSS, JavaScript, Node.js etc.	JavaScript, JSX, UX Markup, etc.	Dart
Publishing to Marketplace	Yes	No	Yes	Yes	Yes

Figure 2.8: Comparison between cross-platform application types (Shah et al., 2019)

2.3.1- UI/ UX (User interface, User experience)

Widget based apps have a high-level user interface and experience quality compared to other cross-platform apps. These apps use Skia Graphics Engine to render UI element and create a user experience and interface close to the quality of a native app. (Biørn-Hansen et al., 2020) For an interpreted app to access the UI element, it needs to use the bridge element. Even though the bridge can cause a performance loss, it helps the app to have a successful user interface. Because, Hybrid apps use HTML, CSS and JavaScript in their implementation,

the user design of the app is not good as the other cross-platform applications. (Rahul Raj & Tolety, 2012)

2.3.2- Performance

Framework	TTC	CPU	PreRAM	RAM	ComputedRAM	∑
Native	5	4	6	6	3	24
MAML/MD_2	4	5	5	5	4	23
NativeScript	6	6	3	3	2	20
React Native	2	1	4	4	5	16
Flutter	3	3	1	2	6	15
Ionic	1	2	2	1	1	7

Figure 2.9: Performance Comparison between cross-platform application types (Biørn-Hansen et al., 2020)

To investigate the performances in Widget-based, Integrated and Hybrid mobile applications, Biørn-Hansen developed a study and tested the sample app scenario using six different mobile application frameworks. After conducting the study, the research completed with the results in Figure 2.9.. The results of time to completion represented as TTC, and PreRAM is to define the memory usage of the idle state. While memory usage during each benchmarking represented as RAM, the actual memory of the bench is called ComputedRAM. Moreover, the Σ symbol represents the sum of the results. (Biørn-Hansen et al., 2020)

Ionic is a hybrid application framework and has the best performance in TTC, RAM and ComputedRam metrics. However, the CPU usage is higher compared to React Native Framework and has a PreRAM result more than the Flutter app. (Biørn-Hansen et al., 2020)

Flutter has a similar performance to React Native. While widget-based app Flutter is strong at PreRAM and RAM, interpreted app framework React Native is better at TTC, CPU and ComputedRAM performances. On the other hand, while comparing the sum, Flutter provides better performance. Consequently, Flutter needs high memory requirements but executes the task with less memory. (Biørn-Hansen et al., 2020)

NativeScript is another framework for Interpreted mobile application. Comparing to React Native and NativeScript, even though PreRAM and RAM and ComputedRAM usage are

lower than React Native, the high rank in TTC and CPU parts lowers the overall performance of the framework. (Biørn-Hansen et al., 2020)

2.4- Flutter

Flutter is a UI toolkit invented by Google in 2016. As an experiment, the google team try to improve the rendering time by replacing the traditional layout on applications and turn everything into a widget. (Fayzullaev, 2018) Moreover, Flutter uses its rendering engine Skia Graphics rather than relying on WebView utilisation as hybrid apps. Moreover, the Hot- reload feature updates the source code on the emulator rather than making changes in the device inner structure. (Wu, 2018)

2.4.1- Widgets

In Flutter, widgets not only the elements of the user interface but also, provide functionality. (Fayzullaev, 2018) For this reason, they must behave quickly and satisfy users. (Wu, 2018)There are two types of widgets, Stateless and Stateful. A stateless widget does not replace its state with user interaction such as Text and Button widgets. On the other hand, if a widget response to the user interaction, such as changing colour or page, these are called Stateful widgets including, checkbox and Form. Figure 2.10 illustrates the comparison between Stateful and Stateless widgets. (Flutter, n.d.)

	Dynamic Composition	Itself immutable	Sub State object mutable
Stateless Widget	False	True	false
Stateful Widget	True	True	True

Figure 2.10: Comparison between Stateful and Stateless widgets (Wu, 2018)

2.4.2- Dart

Dart is the primary programming language of the Flutter framework, and It also developed by the Google team. The programming language Dart follows Object-oriented Languages (OOP) standards. Because, it is object-oriented, the syntax and logic are similar to other OOP languages such as JavaScript and Java. Moreover, the language also contains GarbageCollector to deal with the unused objects to free up memory. (Wu, 2018) Moreover,

mobile applications with dart language can run in IOS, Android and more operating systems natively. (Fayzullaev, 2018)

2.4.3- State Management

Flutter provides state management with the build method. By using this method, the child objects have the data they need while creating. However, for a grandchild object to access the data of the grandparent is accommodable by using of (context) method. For the page navigation, developers use Navigator. pop () method. ("Flutter Architectural Overview - Flutter," n.d.) However, to provide better architectural structure and state management, there are architectural styles like Redux and Block.

2.4.3.1- Redux Pattern

Redux pattern contains Action, Reducer, Store and View parts in its architecture. The store part holds the State object, which represents the state of the application. Each event in the file represented as Actions. However, these executed in the Reducer part of the architecture. According to the action that the Reducer receives, it updates the state object in the store and changes represented in the View part of the architecture for users to see. Figure 2.11 illustrates the architecture of a Redux pattern (RIGAU, 2018)

Figure 2.11: Redux architectural model (RIGAU, 2018)

2.4.3.1- Bloc Pattern

Bloc pattern has three principal parts UI (presentation), bloc (business logic) and data. Firstly, the data layer consists of two parts, data provider and repository. Data provider part used to establish the database connection, and the Repository layer is responsible for transforming the data by interacting with the provider part. Next, the block party is the bridge between the user interface and the database and help them communicate. Finally, the user interface provides functionalities to handle user actions. (Angelov, 2020)

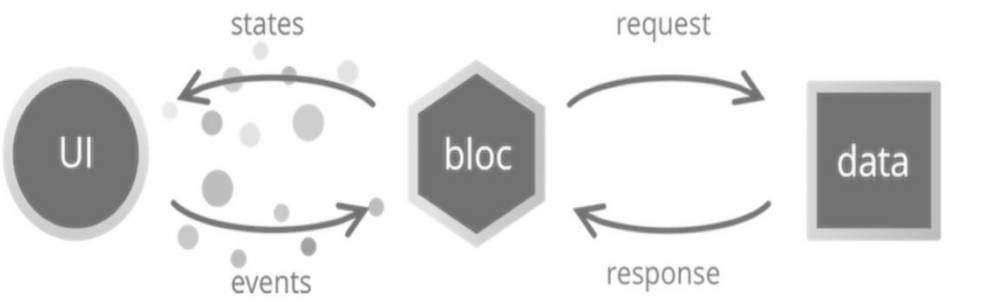


Figure 2.12: Bloc architectural model (Angelov, 2020)

2.5- SQLite and Firebase Database

SQLite is a relational, light weighed, and native database management system written in the C programming language. It has advantages such as faster operations and small code footprint. (Lv, Xu, & Li, 2009) Firebase is a web application platform which helps people to develop mobile and web applications. Moreover, it is a NoSQL database that provides different services such as, Firebase analytics, cloud messaging, authorization, real-time database, a storage test lab, notifications. (Khawas & Shah, 2018)A study to investigate the differences between SQLite relational database and Firebase web application platform took place by Kabakus. (Kabakuş, 2019)

2.5.1- Data Insertion

To see the performance differences while inserting data to the databases, 1 to 100000 data used. As shown in Figure 2.13, the SQLite database provided better functionality. Since

Firebase is a cloud-hosted database it requires more time to operate the action. (Kabakuş, 2019)

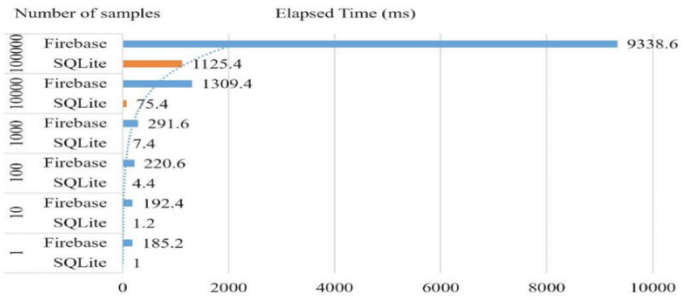


Figure 2,13: Data insertion to SQLite and Firebase (Kabakuş, 2019)

2.5.2- Fetching the Data

The experiment took place to study the fetching the speed of databases, 1 to 1000000 data inserted to the system, the data fetched from the databases to analyse the time and sample relationship. SQLite performed better compared to Firebase while fetching the whole data. For an app to fetch data from the Firebase, it needs to download the data from the cloud. Figure 2.14 illustrates to performance difference while fetching the data from databases. (Kabakuş, 2019)

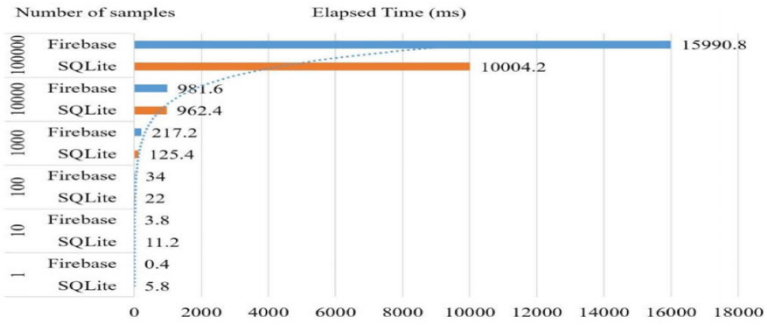


Figure 2.14: Data fetching from SQLite and Firebase (Kabakuş, 2019)

2.6- Mindfulness

Mindfulness defined as bringing the complete attention the present time and accepting the feelings in a non-judgemental way. (Baer, Smith, Hopkins, Krietemeyer, & Toney, 2006) Mindfulness can simply achieve by realizing the surroundings, emotions and thoughts.

However, there are also different meditation types to help people achieve mindfulness. For example, Dispositional mindfulness contains five parts, observing, describing, acting with awareness. It encourages people to pay attention to their feelings without changing them. Meditations like focus attention involve by focusing on a single object and breathing simultaneously. (Cebolla et al., 2017) Moreover, mindfulness techniques are used in chronic pain management, emotional and behavioural disorder treatment, stress and eating disorders. (Bishop et al., 2006)

2.7- Mindfulness Mobile Applications

The research held to compare 203 mobile applications related to the mindfulness theme, including headspace and buddhify, to analyse features of the apps. The research focused on comparison on several downloads, prices, functionalities, networks, content and features and usability. (Plaza et al., 2013)

As a result, most free mobile applications do not have a defined idea of mindfulness. %61 of the application focuses on the meditation, while %13 stress management and 9% contains questionaries. On the other hand, more than half of the mobile applications focused on the meditation aspect (%56), %13 stress management and questionaries. As a result, almost every app is focused on the meditation technique to improve the mindfulness of the users. Moreover, some mindfulness applications give the user opportunity to meditate in different ways such as travelling, walking meditations. Almost every app has resources such as video and reminders as shown in Figure 2.15. (Plaza et al., 2013)

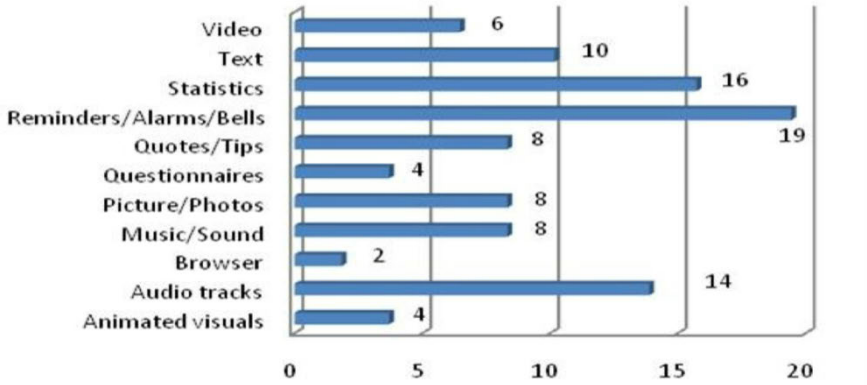


Figure 2.15: Resources and links used in the mindfulness applications (Plaza et al., 2013)

The most ranked free app has features such as the connection to Facebook, Google+ page. Moreover, it also has a journaling activity for the ser to do consciousness self-assessment. Consequently, having a mindfulness app linked to social media pages makes the app more attractive the users. (Plaza et al., 2013)

As a result, Plaza, Demarzo, Herrera-Mercadal, and García-Campayo concluded that the term mindfulness wasn't defined clearly in the applications. Having a defined term of mindfulness and its objectives can make the app meet with the expectations of the users. Moreover, some mindfulness mobile applications lack in usability such as clear buttons, text readability and background colour. Mindfulness global topic, to gain more users, the applications must support different languages. (Plaza et al., 2013)

Moreover, Roquet and Sas investigated 208 mindfulness mobile applications including most popular application such as Headspace, Calm and Breathe. According to the research, the main problem with the most mindfulness application was the lack of meditation techniques. While guided meditations help people to relax, to achieve mindfulness, more effective technique is to provide unguided meditation for people to realise their thoughts, emotions. However, most of the mobile applications doesn't provide unguided meditation. (Roquet & Sas, 2018)

While applications such as Calm and %10 Happier try to keep the user mindful by using breath counting technique, apps such as Headspace asks the user to just be present with the meditation. Most of the application encourages the user to close their eyes during the meditation. However, as expected, visualisation and imagination meditations are less likely to be seen in mindfulness mobile applications. Moreover, relax meditation app recommends the user to meditate at the same time every day to make the meditation habitual. While apps such as Headspace and Relax meditation doesn't contain any sounds, nature sounds are the most popular ones amongst the other applications. However, only Headspace mobile application informs the user about mindfulness by an informative video. Figure 2.16 shows different techniques used in the applications (Roquet & Sas, 2018)

App Name	Meditation Type	Cognitive Strategies	Objects of attention	Closed / open eyes	Static / kinetic	Verbal / non-verbal	Intrinsic / extrinsic	Body Posture	Control of breathing	Avg Duration	Sound Landscape
Headspace	Guided Meditation	Focused attention, Introspection	Body	Closed	Static	Non-verbal	Extrinsic	Sitting on a chair	In nose, out mouth. Focus on rhythm.	10 min	none
Calm	Guided Meditation	Concentration, Introspection, Noting	Body	Closed	Static	*Verbal*	Extrinsic	Sitting on a cushion or chair	Gentle breathing. Count breaths.	3 min - 10 min	nature
Relax Meditation	Guided Meditation	Focused attention, Introspection	Body	Closed	Static	Non-verbal	Extrinsic	Sitting on a chair	In nose, out mouth. Expand belly.	10 min	none
Insight Timer	Self-reliant Meditation	Concentration	*Sound*	-	Static	Non-verbal	*Intrinsic*	-	-	5 min	kangse bell
Digipill	Guided Meditation	Introspection, Passive observation	*Body, Sound*	Closed	Static	Non-verbal	Extrinsic	-	-	13 min	white noise
Relax with Andrew Johnson Free	Guided Meditation	Conscious awareness, Introspection, Body scan	Body	Closed	Static	Non-verbal	Extrinsic	Sitting on a chair	Deep breathing	14 min	music
Mindfulness Daily	Guided Meditation	Introspection, Focused attention	Body	*Any*	Static	Non-verbal	Extrinsic	Sitting, hands on stomach	Breath in deeply, exhale slowly	5 min	none
10% Happier	Guided Meditation	Passive observation, Introspection, Noting	Body	*Any*	Static	*Verbal*	Extrinsic	Sitting still	Inhale straigthen the spine, exhale soften the body	10 min	none
Simple Habit	Guided Meditation	Passive observation	Body	*Any*	Static	Non-verbal	Extrinsic	Sitting on a chair or laying on the ground	In nose, out mouth. Make outbreath sound.	5 min	none
Omvana	Guided Meditation	Focus on the self, Visualization, Body scan	Body	Closed	Static	Non-verbal	Extrinsic	Sitting	Deep breathing	8 min	none
The Mindfulness App	Guided Meditation	Passive observation, Intronspection, Compassion	Body	Closed	Static	Non-verbal	Extrinsic	Posture that reflects dignity.	Expand breathing in, back to center breathing out	10 min	none
Pacifica for Stress & Anxiety	Guided Meditation	Focused attention, Intronspection, Imagination	Body	*Any*	Static	Non-verbal	Extrinsic	Sitting on a cushion or chair	Breathing into stomach.	9 min	nature
Meditation Timer	Self-reliant Meditation	-	-	-	-	-	*Intrinsic*	-	-	10 min	bell
Breethe	Guided Meditation	Focused attention	Body	Closed	Static	Non-verbal	Extrinsic	Sitting on a chair	Deep breathing. In nose, out mouth.	10 min	noise
3 Minute Mindfulness	Guided Meditation	Focused attention, Intronspection, Self-awareness	Body	Closed	Static	Non-verbal	Extrinsic	Comfortable position	Deep breathing. In nose, out mouth.	3 min	none
Tide: Focus, Relax, Meditation	Self-reliant Meditation	-	-	-	-	-	*Intrinsic*	-	-	25 min	nature

Figure 2.16: Mindfulness techniques used in the applications (Roquet & Sas, 2018)

As a conclusion, journaling and meditation features will be used in the app to help people practice mindfulness. One of the most rated application has a journaling feature (Plaza et al., 2013) By using the journaling feature user will be able to identify their feelings, daily mood and thoughts. Moreover, Unguided meditations will be used in the app to create a safe space for the user to be mindful. Even though guided meditations help people to relax, unguided meditation helps the user to recognize what is going on within themselves without any guidance and achieves more mindfulness. (Roquet & Sas, 2018)

Chapter 3

Methodology

This part focuses on methodologies will be used in the application according to the findings from the literature review. The aim of the project is to develop a mindfulness application using Flutter UI toolkit. Following sections explain the best tools, methodologies and environment to achieve this aim.

3.1- Development Environment

The app will be developed using the MacBook Pro laptop (Retina, 15-inch, Mid 2015. It has 2,2 GHz Quad-Core Intel Core i7 CPU processor and runs the macOS Catalina version 10.15.7 which is the latest release of macOS operating system. The primary development environment is going to be Android Studio version 4.0. Android Studio is an IDE for Android mobile application development and it is built on IntelliJ software. ("Meet Android Studio | Android Developers," n.d.)Moreover, Dart and Flutter plugins will be installed to develop the application. However, for testing and deployment purposes, XCode 12.0.01 version will also be used, which is a software development environment to develop software for macOS, IOS products. ("Flutter - Beautiful Native Apps in Record Time," n.d.)

3.2- Methodology

The agile methodology will be used as the methodology of the project. Agile methodology is incremental and iterative, more open for changes during the development period according to customer requirements, and its iterative model will be beneficial for meeting the needs of the customer. Since I don't have lots of experience in Flutter environment as well as the dart language, it will allow me to make changes easier when something goes wrong compared to other development methodologies such as Waterfall.

PXP (Personal Extreme Programming) methodology would not be beneficial for the project since there will not be a pair programming process and the designing of the application is as important as the software functionality.

Kanban methodology is open for changes more than SCRUM. However, it does not contain iterations which will make it hard to get feedback about the application.

By using SCRUM with Agile methodology, the work will be divided into sprints and at the end of each one, receive feedback to produce a high-quality end product. Moreover, sprint planning, prioritising the tasks and sprint retrospectives will be useful on analysing the objectives.

3.3- Testing

IOS emulator that comes with Flutter and Open android Emulator Android Accelerated Oreo and Pixel XL API 30 will be used for testing purposes in the development stage. Emulators create virtual devices on the screen to make the development easier. Moreover, unit, integrated, widget and user testing will take place during the development phase. The unit test is used to prove the functionality of a function or a class under different conditions. Widget testing tests the widgets to prove that UI and functions as expected. Moreover, the integration test used to test the whole app to prove that the UI and features function as expected. ("Testing Flutter Apps - Flutter," n.d.) Finally, prototypes will be used for user testing. Moreover, using questionnaires, users will rate the application and give feedback. The feedbacks taken from the user will help future improvements in the app. The feedbacks and test result will identify whether the app is successful, and objectives achieved, or it must be improved more.

3.4- Design Pattern

Bloc design pattern will be used in the project. Bloc pattern consists of three main parts. UI part helps the user interact with the mobile application. Data provider part used to connect the application with the database and repository layer is used to transform the data returned from the database. Moreover, the block party is the bridge between the user interface and the

database and help them communicate (Angelov, 2020) The bloc design pattern will be used to make the development of the application more maintainable, reusable and testing more easier. Furthermore, it will separate the presentation layer from the logic. In order to implement the design pattern, bloc and flutter_bloc packages will be used (“Flutter_bloc | Flutter Package,” n.d.), (“Flutter_bloc | Flutter Package,” n.d.)

3.5- Programming Language

Dart is a primary language of the Flutter UI kit. It is an Object-oriented language and similar to Java and JavaScript, (Wu, 2018) and it will be used in the development of the application.

3.6- Database

Firebase is a web application platform that contains services such as, cloud messaging, real-time database, authorization. It is a NoSQL database, and everything stores in the cloud system. (Lv et al., 2009) According to Kabakus, SQLite database is a lightweight and performs faster in operations. However, it is not suitable for both web and mobile development and requires storage in the user's device. (Kabakuş, 2019) However, Firebase Authentication system provides simple implementation for user login, signup and sign-out features, and doesn’t take space in users devices. (Khawas & Shah, 2018) The Firebase is the choice of the database due to using a variety of services to provide ease in operations and features to enhance the project further.

Chapter 4

Requirements and Design

4.1- Requirements

All the requirements of the mindfulness mobile application have defined using the Textual use cases and use case diagram. The requirements consist of the most common functionalities of the mobile apps, such as sign-in, sign-up and sign-out features. (Plaza et al., 2013) Moreover, the most desired functions of the mindfulness applications have added as a requirement. According to Plaza, the highly used mindfulness mobile application has a self-reflection future. Moreover, Roquet and Sas state the importance of the un-guided meditations for mindfulness in their research. (Roquet & Sas, 2018) To enhance the capabilities, quotes and mindfulness pages added as a requirement. The UML diagram in Figure 4.1 illustrates the application requirements and system boundaries.

Figure 4.1: Use case diagram

The UML diagram illustrates the Primary actor user and the secondary actors Firebase authentication and Firebase cloud firestore. Moreover, requirements represented in the eclipse shape connected to their actors, and the system boundary represented in a rectangle shape.

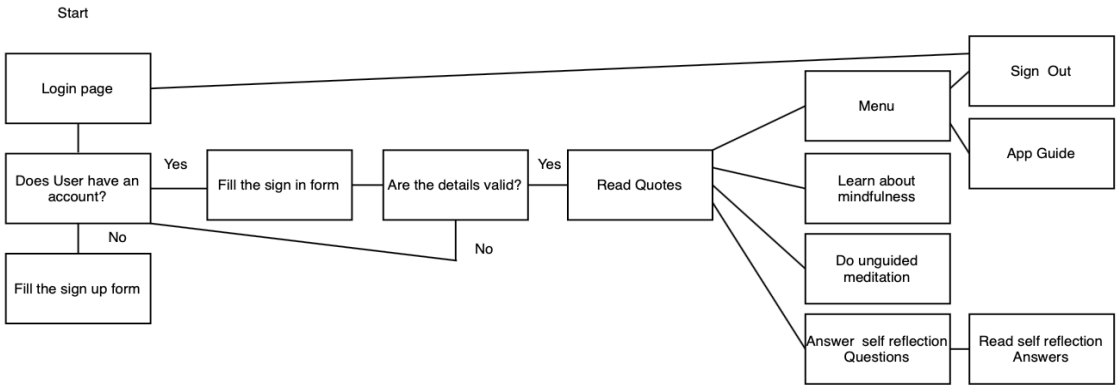


Figure 4.2: Flow diagram

The flow diagram represents the journey of the User within the application. The flow starts by entering the login page, and it completes when the user signs out from the app.

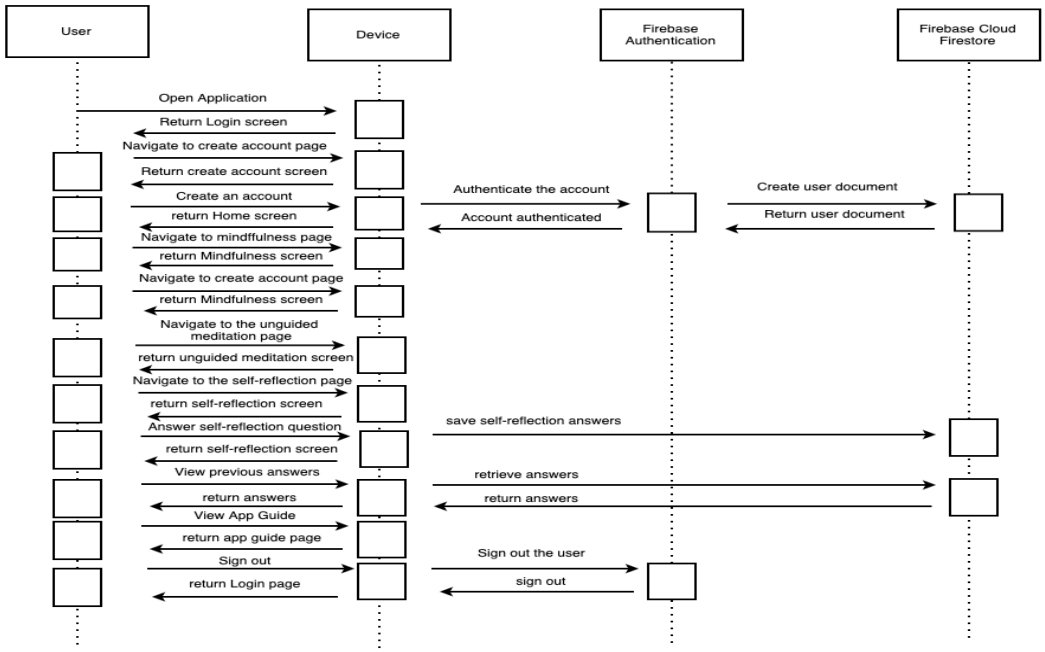


Figure 4.3: Sequence diagram

The sequence diagram on Figure 4.3 represents the interaction between the object in a sequential manner. Mindfulness application includes four objects including User, device, firebase authentication and cloud firestore.

4.1.1- Requirement 1- Sign Into the account

The sign-in requirement is beneficial to limit unauthorized user access to the application. Firebase authentication services play the role of a secondary actor. It handles the sign-in feature with the stored user information and authorizes the user. To sign in to the system, the user has to provide their e-mail address and password. If the user has an account, Firebase authentication authorizes the user. Table 4.1 represents the Textual Use case specification for the Sign-in requirement to explain its purpose.

Use case title	Sign into the account
Primary Actor	User
Secondary Actor	Firebase Authentication
Goal	Sign into the app using e-mail and password
Scope	Mobile application to help user stay mindful
Preconditions	The user has to already have an account
Main success scenario	1- The user enters their e-mail 2- The user enters their password 3- The user clicks the button 4- Firebase authentication confirms the user's account 5- Firebase Cloud Firestorage checks if the user has a document 6- The user signs into the app
Extensions	1a-The user can enter a wrong email address 2a- The user enters wrong password 2b- The user forgets their password 3a- the user doesn't click the button 4a- The user hasn't registered yet. 4b- Firebase fails to authenticate the user's account 5a- Firebase Cloud Firestorage fails to find the document 5b- Firebase Cloud Firestorage finds the wrong document 6a-The user cannot sign in

Table 4.1: Sign in to the account requirement Use case Specification

4.1.2- Requirement 2- Create an account

The second requirement of the project is creating an account. The purpose is to limit the access of anonymous users to the app. Moreover, creating an account helps users to login into the app whenever they want. To implement this requirement, the Firebase authentication system and Cloud Firebase used. Firebase services check if the user has an

account and creates a document when the user registers. Table 4.2 Use case specification illustrates the success scenario, as well as the functionalities.

Use case title	Create account
Primary Actor	User
Secondary Actor	Firebase Cloud Firestorage
Goal	Sign up the app using username, e-mail and password
Scope	Mobile application to help user stay mindful
Preconditions	No preconditions
Main success scenario	1- The user creates username 2- The user enters their e-mail 3- The user enters their password 4- The user clicks the button 5- Firebase authentication creates the account 6- Firebase Cloud firestore creates the document and adds user to the database 7- The user signs up to the app
Extensions	2a- The user enters invalid email address 3a- The user enters invalid password 4a- User forgets to click the button 5a- Firebase Authentication fails to authenticate user 6a- Firebase Cloud Firestorage fails to create the document 6b- Firebase Cloud Firestorage creates document with wrong details 7a- The user cannot sign up to the app

Table 4.2: Create account requirement Use case Specification

4.1.3- Requirement 3- Read the Quotes from the Home page

Read the quotes on the home page requirement is beneficial for motivating and inform the user about mindfulness. According to the research conducted by Plaza, the quotes page is the 4th most wanted mindfulness and meditation app feature, after the reminders and Audios. (Plaza et al., 2013) A list contains 20 quotes has implemented. Furthermore, the quotes are randomly chosen and changes when the user enters the Homepage.

Use case title	Read Quotes
Primary Actor	User
Secondary Actor	None
Goal	Open the Home page and see the quotes
Scope	Mobile application to help user stay mindful
Preconditions	User logged in or sign up to the app
Main success scenario	1- The user enters to the Home page
Extensions	1a- Homepage has a problem loading 1b- The user cannot find the Homepage button

Table 4.3: Read Quotes requirement Use case Specification

4.1.4- Requirement 4- Learn About Mindfulness

While most of the applications on the marketplace use different strategies to make their users more mindful, only the Calm application includes an informative video to give information about mindfulness. (Roquet & Sas, 2018) Requirement 4 helps introduce the mindfulness concept to the user and teach them how to bring their attention to the moment. The technique of seeing and feeling the object used in the application encourages users to slow down and be present with that object while staying mindful.

Use case title	Learn about mindfulness
Primary Actor	User
Secondary Actor	None
Goal	Open the mindfulness page and learn about mindfulness
Scope	Mobile application to help user stay mindful
Preconditions	User logged in or sign up to the app
Main success scenario	1- The user navigates to the Mindfulness page
Extensions	1a- Mindfulness page has a problem loading 1b- The user cannot find the Mindfulness page button 1c- The user fails to swipe to the next part within the page

Table 4.4: Learn about mindfulness requirement Use case Specification

4.1.5- Requirement 5- Do Unguided meditation

Even though the mobile applications on the marketplace contain guided meditation, un-guided meditations are more helpful to stay mindful. (Roquet & Sas, 2018) Requirement 5 focuses on implementing 10-minute unguided meditation to encourage the user to be present and focus on their breath. As a result, the user is going to achieve more mindfulness in their lives. Table 4.5 represents a textual use case diagram for this requirement.

Use case title	Do unguided meditation
Primary Actor	User
Secondary Actor	None
Goal	Open the unguided meditation page and do unguided meditation
Scope	Mobile application to help user stay mindful
Preconditions	User logged in or sign up to the app
Main success scenario	1- The user navigates to the unguided meditation page 2- The user starts the mindfulness timer by clicking button 3- The user meditates 10 minutes 4- The meditation finishes
Extensions	1b- The user cannot find the Unguided page button 2a- The timer doesn't start 2b- The user forgets to click the button 2c- Unguided meditation page has a problem loading 3a- The timer stops working before 10minutes 4a- Meditation continues after 10 minutes

Table 4.5: Do unguided meditation requirement Use case Specification

4.1.6- Requirement 6- Answer Self-Reflection Questions

Requirement 6 helps the user to reflect on their feelings and emotions by a list of questions. According to the research held by Plaza, one of the most rated application has a journaling feature. (Plaza et al., 2013) Moreover, to encourage users to reflect on emotions and thought, twenty questions prepared. The system randomly chooses three questions for users to answer. Furthermore, the answers will be saved to the Cloud Firestore to be retrieved later.

Use case title	Answer self-reflection questions
Primary Actor	User
Secondary Actor	Firebase Cloud Firestore
Goal	Answer the self-reflection questions
Scope	Mobile application to help user stay mindful
Preconditions	User logged in or sign up to the app
Main success scenario	1- The user navigates to the self-reflection page 2- Answer the self-reflection questions 3- Click on the button to save the answers 4- Firebase saves the users to the database
Extensions	1b- The user cannot find the Self-reflection page button 2a- Users cannot find the questions 2b- Self-reflection page has a problem loading 3a- The user forgets to click on the button 3b- The user cannot find the button 4a- Firebase Cloud has a problem while saving the answers 4b- Firebase Cloud saves wrong answers

Table 4.6: Answer Self-Reflection Questions Requirement Use Case Specification

4.1.7- Requirement 7- See the Previous Answers to the Questions

Requirement 7 aims to provide a place that contains previous answers to the reflection questions. Seeing the previous answers allows the user to be inspired, get motivated and reflect on their answers. The answers stored in the Cloud firebase and returned to the user when they open the page.

Use case title	See the previous answers to the self-reflection questions
Primary Actor	User
Secondary Actor	Firebase Cloud Firestore
Goal	Open the self-reflection page and see the previous answers
Scope	Mobile application to help user stay mindful
Preconditions	User logged in or sign up to the app
Main success scenario	1- The user navigates to the self-reflection page 2- Users clicks on the history tab 3- Firebase Cloud system returns the previous answers from the database 4- Users sees the answer
Extensions	1a- Users cannot find the button to navigate to the self-reflection page 2a- The user forgets to click on the History tab 3a- Firebase does not return answers 3b- Firebase return the wrong answers 4a- The page has a problem while loading

Table 4.7: See the Previous Answers to the Questions requirement Use case Specification

4.1.8- Requirement 8- Read the App Guide

To make the application more useful and provide benefits to the user, requirement 8 implemented. The App guide introduces the application to the user and explains each section and its aims. It is a distinctive feature from the apps on the marketplace, which will benefit the user to understand the application.

Use case title	Read the App Guide
Primary Actor	User
Secondary Actor	None
Goal	Open the App Guide page and learn about the App
Scope	Mobile application to help user stay mindful
Preconditions	User logged in or sign up to the app
Main success scenario	1- The user navigates to the App guide page
Extensions	1a- Users cannot find how to enter to the App guide 1b- The system has a trouble while loading the page

Table 4.8: Read the App Guide requirement Use case Specification

4.1.9- Requirement 9- Sign Out

The sign out is another common requirement for the mobile application. It allows users to sign out of their account with the help of the Firebase authentication services.

Use case title	Sign out
Primary Actor	User
Secondary Actor	Firebase Cloud Firestore
Goal	Sign out of the App
Scope	Mobile application to help user stay mindful
Preconditions	User logged in or sign up to the app
Main success scenario	1- The user clicks on the button to sign out 2- Firebase authentication signs out the user 3- User navigates to the Login page
Extensions	1a- Users cannot find the sign-out button 1b- The system has a trouble while loading the page 2a- Firebase authentication fails to sign the user out 3a- User stays in the Homepage, even though they are signed out.

Table 4.9: Sign Out requirement Use case Specification

4.2- Design

4.2.1 Interface Design

4.2.1.1- Wireframes

The wireframes in Figure 29 illustrates the design of the main pages and features. Furthermore, they created as an initial step of the design process to create the style of the application before moving into prototyping and implementation.

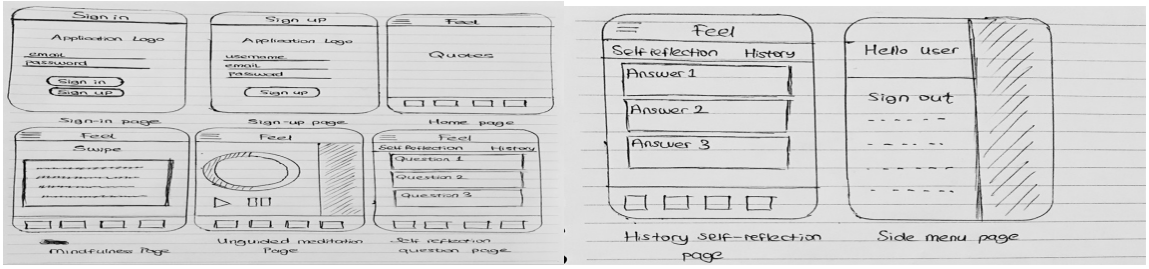


Figure 4.4: Wireframe design

4.2.1.2- Prototype

Before the implementation of the minimum viable product, the prototype has created using the marvel website. The prototype includes sign-in, sign up, home and mindfulness pages. Green and white used as the primary colours of the application. The texts were readable in the prototypes. However, the primary concern of the prototype was the readability of the text sign-in and sign-up pages.

Figure 4.5: The prototype design

After implementing the Minimum viable product and conducting prototype testing, the text was not readable on the emulators, and the colours had problems. This situation led to the creation of a new prototype. The second prototype has blue and white as the primary colours. The settled background picture made the texts readable, and blue and white colours were suitable for the mindfulness theme. The Figure 4.6 represent the new prototype design.

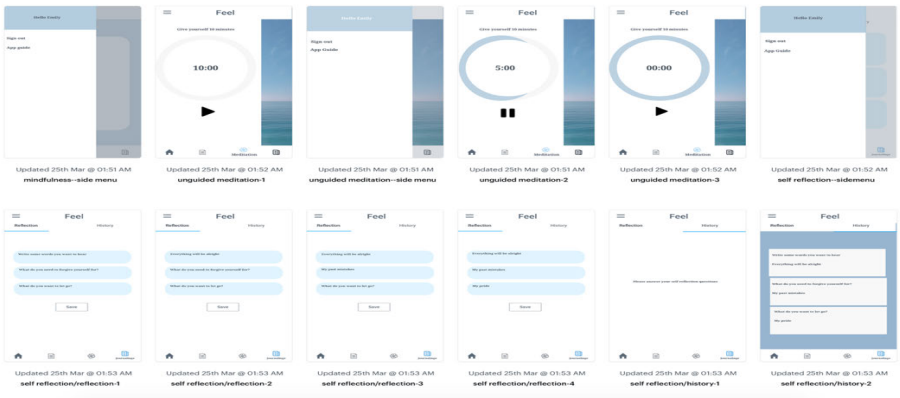


Figure 4.6: The second prototype design

The prototypes designed and tested to be suitable for people with sight problems. Moreover, Figure 4.7 illustrates the colour blindness test result of the login page with the background picture.

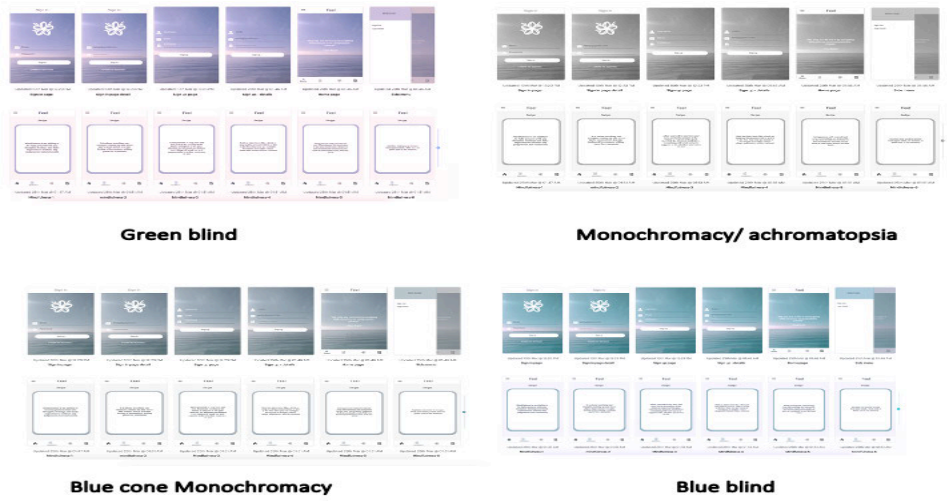


Figure 4.7: The sight difficulty test results

Moreover, the mobile application created to be easily understood and used by the user by bottom navigation bar and abstract design. Moreover, the dark patterns, such as nagging, obstruction, sneaking, interface interference and forced actions defined by Gray, Kou, Battles, Hoggatt, and Toombs, have been avoided in the application (Gray, Kou, Battles, Hoggatt, & Toombs, 2018)

4.2.2 Database Design

Firebase cloud firestore services used to store information about the users. To store the users' documents, the collection called users created. Each user document is associated with a unique user identification number. Moreover, each user document contains three fields called username, register date and reflection. A username field is a type of string and stores the nickname of the user. Register date is a type of timestamp and shows the date the user set up their account. Finally, the reflection field is a type of map and stores three self-reflection

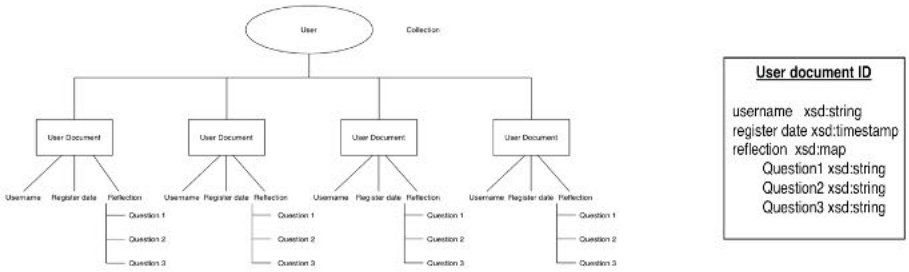


Figure 4.8 The database design

questions and answers. The design of the database illustrated on the Figure 4.8.

4.2.3 System Design

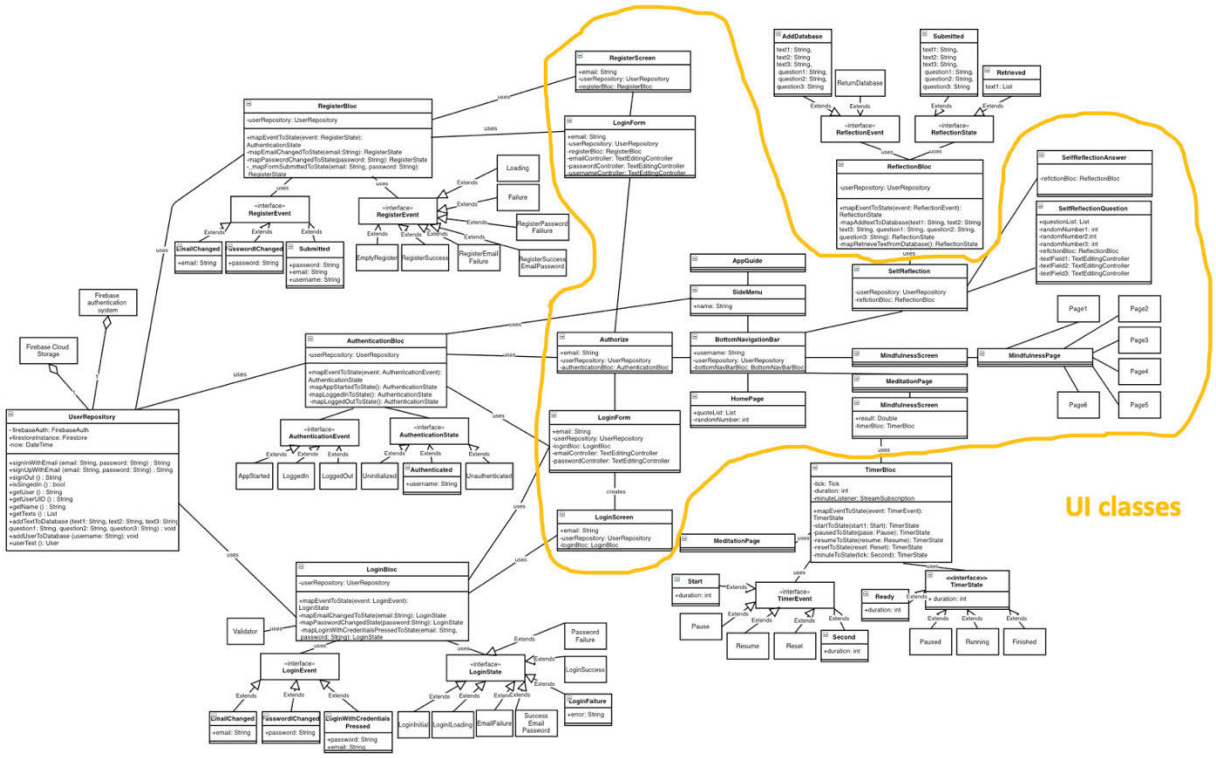


Figure 4.9: UML Diagram

4.2.3.1- User Repository

User repository class used to achieve the connection with the firebase services. It contains methods to help users create their account, sign in and sign out. Moreover, the class includes functions to create user documents on the firebase cloud storage. Each class connected to the systems must call this class to reach the database.

4.2.3.2- Authorize

Authorize class connected to the Authentication Bloc class. The main purpose of this class is to observe the state changes and navigate the user to the responding pages. For instance, if the authentication state is unauthenticated, the class navigates the user to the login page.

4.2.3.3- AuthenticationBloc

AuthenticationBloc receives the Authentication events and converts them into states using the Stream methods. The class contains three private methods to return the AuthenticationState. Moreover, it returns the username of the user from the database using the UserRepository.

4.2.3.4- AuthenticationEvent

AuthenticationEvent used to connect the AuthenticationBloc to the UI. It contains an abstract class called Authentication event and three subclasses of the AuthenticationEvent superclass.

4.2.3.5- AuthenticationState

AuthenticationState class aims to identify the state of the Authorize class. The Authentication state superclass contains three subclasses named Uninitialized, Authenticated and Unauthenticated.

4.2.3.6- RegisterScreen

Register screen class used to build the bloc provider of the register bloc. Bloc provider enables bloc to be introduced on the application once and used in the different classes without building the same bloc again. This class contains the RegisterForm class as its child.

4.2.3.7- RegisterForm

Register form used to build the registration screen for the user to create their account. The text form fields help user to enter their details, and the button element submits the information to the register bloc to set up the account. Moreover, it navigates the user to the home page.

4.2.3.8- RegisterBloc

RegisterBloc takes the Register events and returns them as state using Stream methods. The class contains three different private methods to return the state from the RegisterState. Moreover, this class registers the users to the firebase services by calling the UserRepository class.

4.2.3.9- RegisterEvent

RegisterEvent used to connect the RegisterBloc to the RegisterForm class. It contains an abstract class called RegisterEvent and three subclasses of the RegisterEvent superclass.

4.2.3.10- RegisterState

RegisterState class aims to store the state of the RegisterForm class. The Registerstate superclass contains seven abstract methods to identify the state of the form.

4.2.3.11- LoginScreen

Login screen class aims to build the bloc provider of the login bloc. Bloc provider enables bloc to be introduced on the application once and used in the different classes without building the same bloc. This class contains the LoginForm class as its body.

4.2.3.12- LoginForm

Login form beneficial for building the login form for the user. The text form fields help user to enter their details, and the button element submits the information to the login bloc to enter to the account. Moreover, if the login is successful, the class navigates the user to the home page.

4.2.3.13- LoginBloc

LoginBloc receives the Login events and returns them as state using Stream methods. The class contains three private methods to return LoginState.

4.2.3.14- LoginEvent

The LoginEvent class used to connect the LoginBloc to the LoginForm class (UI). It contains an abstract class called LoginEvent and four subclasses of the LoginEvent superclass.

4.2.3.15- LoginState

LoginState class aims to store the state of the LoginForm class. The Loginstate class contains seven abstract methods to identify the state of the form.

4.2.3.16- BottomNavigation

BottomNavigation class used to create the bottom navigation menu of the application, which users can navigate to different pages. The class contains Streams to observe the state. Moreover, each navigation bar item contains icons and text and associated with the represented class.

4.2.3.17- BottomNavigationBloc

BottomNavigationBloc contains a stream controller to control the state of the bottom bar and sinks the events with the switch statement. Moreover, the class contains an enum that defines the named constant values.

4.2.3.18- AppGuide

The purpose of the AppGuide class is to inform the user about the purpose and the contexts of each section.

4.2.3.19- HomePage

The homepage includes a List to store eleven quotes and a background image. The container element utilized to print the random quotes by the index by the chosen by the random generator. Moreover, the centre element helps to set the text to the middle of the application screen.

4.2.3.20- MindfulnessScreen

MindfulnessScreen class contains a stack to separate the text from the context of the mindfulness page. The text informs the user about the transaction of the page, and the container prepares the pageview of the Mindful Screen.

4.2.3.21- MindfulnessPage

This class contains a Pageview element to create the horizontal scroll effect. Moreover, the element contains six different child widgets. Each child widgets have distinctive texts to inform the user about mindfulness.

4.2.3.22- SelfReflection

SelfReflection class prepares the bloc provider for Reflection Bloc class to use in the SelfReflectionQuestions and Answers pages. Moreover, A TabBar created for the user to navigate between the answers and question while staying on the self-reflection section.

4.2.3.23- SelfReflectionQuestions

The main purpose of the class is to return the self-reflection questions to the screen. The class contains a question list to store seventeen questions. Three random generators choose the index of the question. Text Field and Text Controller elements help users to type their answers. Moreover, the button submits the questions and answers to the reflection bloc to store in the database.

4.2.3.24- SelfReflectionAnswer

SelfReflectionAnswer prints the previous answers of the users to the page. BlocBuilder watches the state of the reflectionbloc, and if the database is not empty, the class returns the list and represents it on the screen with the Text element.

4.2.3.25- SelfRelectionBloc

SelfReflectionBloc sinks the SelfReflection events and returns them as state using Stream methods. The class contains two private methods to return SelfReflectionState. Moreover, it adds the self-reflection answers and questions to the database by calling UserRepository.

4.2.3.26- SelfReflectionState

SelfReflectionState class aims to store the state of the self-reflection class. The ReflectionState superclass contains three subclasses to identify the state of the self-reflection question and answers.

4.2.3.27- SelfReflectionEvent

SelfReflectionEvent used to connect the SelfReflectionBloc to the self-reflection class. It contains an abstract class called SelfReflectionEvent and four subclasses of the selfReflection Event superclass.

4.2.3.28- SideMenu

SideMenu contains ListView to store the ListTiles. The first ListTile is used to sign the user out of the application, and the second class navigates the user to the AppGuide page.

4.2.3.29- UnguidedMeditation

UnguidedMeditation calls the TimerBloc class to get the state of the Timer and returns it to the screen. Moreover, it illustrates the timer process using the CircularProcessIndication. Icon buttons play and pause, helps users to control the timer and the process indication.

4.2.3.30- MeditationPage

MeditationPage contains a Row to separate the image from the timer and locates the image on the right side of the screen.

4.2.3.31- MeditationScreen

MeditationScreen uses BlocProvider to build the TimerBloc, and it contains Unguided Meditation class as a child.

4.2.3.32- TimerBloc

TimerBloc receives the Timer events and returns them as state using Stream methods. The class contains five methods to return the TimerState.

4.2.3.33- TimerState

TimerState class aims to store the state of the Timer class. The TimerState superclass contains four subclasses to identify the state of the timer.

4.2.3.34- TimerEvent

TimerEvent used to connect the TimerBloc to the UnguidedMeditation class. It contains an abstract class called TimerEvent and five subclasses of the TimerEvent superclass.

4.3- Summary

Firstly, this chapter discussed the identified requirements to build the mobile application. Moreover, to represent the system design, use case, flow and sequence diagrams used. The use case specification prepared for each requirement. Furthermore, the design of the application discussed with the wireframes and prototypes. Finally, a system UML diagram built, and each class discussed.

Chapter 5

Development and Implementation

5.1- Sprint 1

Sprint 1 dedicated to building the Minimum viable product and connecting the application with the Firebase authentication services. Firebase_core, firebase_auth and cloud_firestore packages added to the project. During this period, the login form and Bottom Navigation Bar built, the connection between Authentication services and the application achieved. Moreover, challenges and results are discussed.

5.1.1-Sign in Requirement

The implementation of the Login page started by creating the login form using the ListView element and scroll-down effect. Moreover, TextFormField implemented to create the email and password fields. Two different controllers assigned to control the user's input. The raised Button element implemented to send the user email and password to the authentication services. As an initial stage, no state management technique provided, and the call directly done by the LoginFormclass.

5.1.2-Sign Out Requirement

In the attempt to use the sign out feature of the authentication services, the home page with the sign out button developed. The home page contains the BottomNavigationBar element to help the user navigate to different pages. Moreover, each item holds an icon and text to inform the user about the page. Set State () function used to add state management to the application. Finally, to achieve the sign out requirement, a floating action button added to the app bar. Listing 5.1 illustrates the call to the authentication services to sign the user out.

```
floatingActionButton: FloatingActionButton
  (
  onPressed: () {
    FirebaseAuth _firebaseAuth;
    _firebaseAuth.signOut();
  },
), // FloatingActionButton
```

Listing 5.1: Firebase authentications sign out

5.1.3- Issues

The most challenging part of Sprint-1 was establishing the connection between the authentication services and the application. Because the flutter framework and dart language are not familiar, it was hard to get used to the logic. According to different resources and the Firebase website, WidgetsFlutterBinding.ensureInitialized() and Firebase.initializeApp() functions should implemented on the application, as illustrated on the Listing 5.2. After adding these statements to the main method, the problem has resolved.

```
void main() async {
  //main function to internilize the firebase and call the app class
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(Authorize(email: null)); //
```

Listing 5.2: Initializing the Firebase methods

5.1.4- Result

At the end of the first sprint, the login and log out functions added to the system and the bottom navigation bar built. Moreover, the Firebase authentication services achieved to connect to the application for both IOS and Android. Figure 5.1 illustrates the minimum viable product.

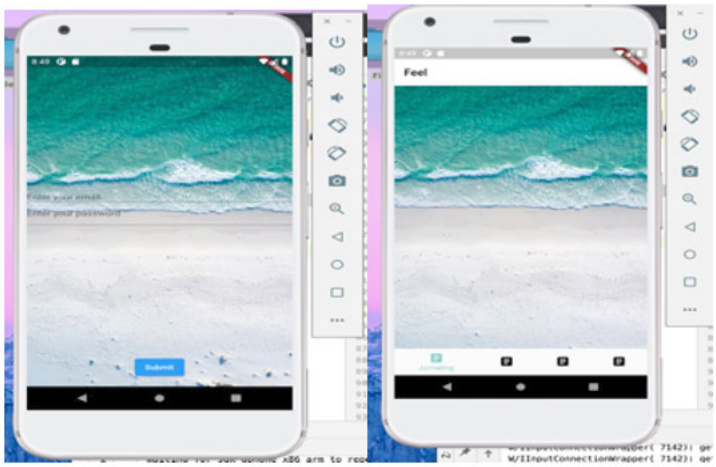


Figure 5.1: Minimum viable product

5.2- Sprint 2

In the second sprint, sign-in and sign-out requirements refactored to apply the bloc stage management technique. To use the bloc state management, the fluter_bloc and bloc packages added to the project as a dependency. Moreover, the sign-up requirement and the Mindfulness and Home pages built.

Bloc package takes a Stream of Events as input and transforms them into a Stream of States. Moreover, the flutter_bloc package is beneficial for building the blocs using BlocProvider and BlocBuilder classes. According to different developers, such as Rešetár and Stool, implementing the bloc pattern from scratch with the _bloc package causes boilerplate and unnecessary code for smaller application. (Rešetár, n.d.)(Stoll, 2019)

5.2.1- Sign in Requirement

To achieve the sign-in requirement with bloc pattern, Login Event, Login state and Login bloc classes implemented. As to achieve the state management, the Event class added to the project. The Event class is beneficial for connecting the User interface and the bloc class. Furthermore, the LoginEvent contains an abstract superclass and three different subclasses to identify the user's interaction with the application. (Listing 5.3)

```dart
@immutable
abstract class LoginEvent extends Equatable {
  LoginEvent([List props = const []]);
}

//when a changes made in the email field
class EmailChanged extends LoginEvent {
  final String email;

  EmailChanged({@required this.email,} ) : super([email]);
  //sending the email to the LoginEvent superclass
  String toString() => 'EmailChanged { email :$email }';
}

//when a changes made in the password field
class PasswordChanged extends LoginEvent {
  final String password;

  PasswordChanged({@required this.password}) : super([password]);
  //sending the password to the LoginEvent superclass
  String toString() => 'PasswordChanged { email :$password }';
}

class LoginWithCredentialsPressed extends LoginEvent {
  final String email;
  final String password;

  LoginWithCredentialsPressed({@required this.email, @required this.password})
      : super([email, password]);
//sending the password and email to the LoginEvent superclass

  @override
  String toString() {
    return 'LoginWithCredentialsPressed { email: $email, password: $password }';
  }
}
```

Listing 5.3: LoginEvent class

Additionally, to implement the Login state class, the tutorial by Felix Angelov followed. Moreover, the state management implemented factory constructors and different values of the fields according to the functions. (Listing 5.4) The factory constructors were beneficial for returning the class from the cache instead of creating a new instance.

```dart
class LoginState {
  final bool isEmailValid;
  final bool isPasswordValid;
  final bool isSubmitting;
  final bool isSuccess;
  final bool isFailure;

  bool get isFormValid => isEmailValid && isPasswordValid;

  LoginState({
    @required this.isEmailValid,
    @required this.isPasswordValid,
    @required this.isSubmitting,
    @required this.isSuccess,
    @required this.isFailure,
  });

  factory LoginState.empty() {
    return LoginState(
      isEmailValid: true,
      isPasswordValid: true,
      isSubmitting: false,
      isSuccess: false,
      isFailure: false,
    );
  }

  factory LoginState.loading() {
    return LoginState(
      isEmailValid: true,
      isPasswordValid: true,
```

Listing 5.4: Previous LoginState Class (Angelov, 2019a)

The factory classes help the application to observe the state of the login operation. (Listing 5.4) However, a more maintainable and testable implementation approach found during sprint 2. (Listing 5.5)

```dart
abstract class LoginState extends Equatable{
  @override
  List<Object> get props => [];
}

class LoginInitial extends LoginState {}

class LoginLoading extends LoginState {}

class EmailFailure extends LoginState {}

class SuccessEmailPassword extends LoginState {}

class PasswordFailure extends LoginState {}

class LoginSuccess extends LoginState {}

class LoginFailure extends LoginState {
  final String error;

  LoginFailure({@required this.error});

  @override
  List<Object> get props => [error];
}
```

Listing 5.5: Current LoginState Class

Moreover, the purpose of the bloc class is to return these actions to states. Two private methods yield a single value which is an instance of a LoginState. Furthermore, the validity of the password and email checked by the validator class and returned a Boolean value. For instance, If the value is true, the SuccessEmailPassword () state is called. The final method used to sign the user into the application by calling the User Repository class function. (Listing 5.5)

```dart
//email that matches the pattern of aaaaaa@aaa.aaaa
//@ and . must be used
class Validators {
  static RegExp _emailCheck = RegExp(
    r'^[a-zA-Z0-9.!#$%&’*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$',
  );

  //minimum 8 character password at least one number
  static RegExp _passwordCheck = RegExp(
    r'^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{8,}$',
  );

  static isValidEmail(String email) {
    return _emailCheck.hasMatch(email);
  }

  static isValidPassword(String password) {
    return _passwordCheck.hasMatch(password);
  }}
```

Listing 5.5: Validators to check the email and password

The UserRepository class implemented to establish the connection between the application and Firebase services. The signInWithEmailAndPassword () built to store the details to authentication services.

The LoginState class built in the Sprint 1 refactored. The LoginScreen class added to build the LoginBloc with BlockProvider functionality of the flutter_bloc. When a bloc built using the BlockProvider function, the child widgets can reach the block anytime with a single call. (Listing 5.6)

```
    body: BlocProvider<LoginBloc>(
      bloc: _loginBloc,
      child: LoginForm(userRepository: _userRepository)
    ), // BlocProvider
  ); // Scaffold
}
```

Listing 5.6: Bloc Provider

To adapt the class to the new approach, the login bloc is called with the corresponding LoginwithCredentialPressed () event. Moreover, by using the BlockBuilder utility of the bloc pattern, the state of the login bloc is reached, and the users navigated to the Authorize class. (Listing 5.7)

```
return BlocBuilder<LoginEvent, LoginState>(
  bloc: _loginBloc,
  builder: (
    BuildContext context,
    LoginState state,
  ) {
    if (state is LoginSuccess) {
```

Listing 5.7: Bloc builder function

5.2.2- Sign up Requirement

The second requirement is to Sign up. The system uses the Firebase authentication system to register the user and the cloud firestore to create the user document. The bloc design pattern implemented to achieve this requirement and control the state of the user register action. Moreover, the event, state and bloc classes added to the project.

The event class is beneficial for observing the user interaction with the presentation layer. The class contains three subclasses EmailChanged, PasswordChanged and Submitted. Furthermore, the EmailChanged and PasswordChanged classes check the validity of the details. Moreover, the Submitted instance gets the username, email, password and passes them to the RegisterBloc class. (Listing 5.8)

```
//submitting the form
class Submitted extends RegisterEvent {
  final String email;
  final String password;
  final String username;

  Submitted({@required this.email, @required this.password, @required  this.username})
      : super([email, password, username]);

  @override
  String toString() {
    return 'Submitted { email: $email, password: $password, username: $username }';
  }
}
```

Listing 5.8: Submitted function

State class includes a superclass and seven subclasses. The subclasses define the state of the Register function. For instance, If the registration fails, the bloc class returns the Failure state, and the user cannot reach the home page.

Furthermore, the Register Bloc class contains three private Stream functions. mapEmailChangedToState () and mapPasswordChangedToState are responsible for verifying the email address of the user using the Validators class whenever the user types into the email and password text field. (Listing 5.10) Finally, the mapSubmittedToState method calls the User repository class to sign the user up to the firebase authentication services, creates a document on Cloud firestore to store the username.

```
class RegisterBloc extends Bloc<RegisterEvent, RegisterState> {
  final UserRepository _userRepository;

  RegisterBloc({@required UserRepository userRepository})
      : assert(userRepository != null),
        _userRepository = userRepository;

  @override
  RegisterState get initialState => EmptyRegister();

//returning events to state
  @override
  Stream<RegisterState> mapEventToState(
      currentstate, RegisterEvent event,
      ) async* {
    if (event is EmailChanged) {  //checking the validity of email
      yield* _mapEmailChangedToState(event.email);
    } else if (event is PasswordChanged) { //checking the validity of password
      yield* _mapPasswordChangedToState(event.password);
    } else if (event is Submitted) { //submitting details
      yield* _mapFormSubmittedToState(event.email, event.password, event.username);
    }
  }
```

Listing 5.9: Converting events to the state

```
  //checking the validity of email
  Stream<RegisterState> _mapEmailChangedToState(String email) async* {
    if (!Validators.isValidEmail(email)) { //email is invalid
      yield RegisterEmailFailure();
    } else if (Validators.isValidEmail(email)){//email is valid
      yield RegisterSuccessEmailPassword();
    }
  }
//checking the validity of password
  Stream<RegisterState> _mapPasswordChangedToState(String password) async* {
    if (!Validators.isValidPassword(password)) { //password invalid
      yield RegisterPasswordFailure();
    } else if (Validators.isValidPassword(password)){ //password valid
      yield RegisterSuccessEmailPassword();
    }
  }
```

Listing 5.10: Using validator to check the email and password

In the User repository class addUsertoDatabase () function creates the document on cloud firestore. The document created using the unique firebase authentication id of the user. Finally, using the firestore set function, the username and register date of the user stored. (Listing 5.11)

```
Future <void> addUserToDatabase (String username) {
  var userUid =  _firebaseAuth.currentUser.uid;
  firestoreInstance.collection('users').doc(userUid).set({
    "username": username,
    "register date": _now,
  });
}
```

Listing 5.11: Adding user to the database

Register Screen class builds the registerbloc by calling the BlocProvider method of the bloc class. Moreover, as a child of the BlocProvider, the RegisterForm class is initialized to set up the registration form.

ListView element utilized to build the register form. As the children of the ListView widget, three TextFormField widgets implemented to receive the user's input. Controllers set to observe the changes in the fields. Each TextEditingController assigned to a method, and these methods are responsible for calling the registerbloc with events to check the validity of the password and emails. (Listing 5.12)

```
final TextEditingController _emailController = TextEditingController();
final TextEditingController _passwordController = TextEditingController();
final TextEditingController _usernameController = TextEditingController();

    void _onEmailChanged() {
      _registerBloc.dispatch(
        EmailChanged(email: _emailController.text),
      );
    }

    void _onPasswordChanged() {
      _registerBloc.dispatch(
        PasswordChanged(password: _passwordController.text),
      );
```

Listing 5.12: Dispatching event to the bloc class to check the validity of the email and password.

The RaisedButton with the text "Sign up" implemented. The functionality of the button is to retrieve the details of the user from controllers and call the registerbloc with the parameters to register the user. (Listing 5.13)

```
        onPressed: () { //calling the registerbloc
          _registerBloc.dispatch(Submitted(
              email: _emailController.text,
              password: _passwordController.text,
              username: _usernameController.text)); // Submitted
        },
        shape: new RoundedRectangleBorder(
```

Listing 5.13: Connection to the bloc with dispatch method

Finally, using the BlocBuilder functionality, the class reaches the state of the registerbloc. (Listing 5.14) For instance, If the state is a Failure, the user cannot navigate to the homepage.

```
return BlocBuilder(
  bloc: _registerBloc,
  builder: (BuildContext context, RegisterState state) {
    if (state is RegisterSuccess) {
      print(_usernameController.text);
      BlocProvider.of<AuthenticationBloc>(context).dispatch(LoggedIn());
      SchedulerBinding.instance.addPostFrameCallback((_) {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (BuildContext context) => Authorize(
              email: _usernameController.text,
            ), // Authorize
          ), // MaterialPageRoute
        );
      });
    }
```

Listing 5.14: Reaching the states through bloc builder

5.2.3- Sign out Requirement

Authentication bloc, event and states are responsible for observing the state of the user when the application starts. For example, if a user is already signed in or registered, the bloc returns LoggedIn state. However, if a user is logged out, the class returns them to the login form, and users cannot enter the home page.

The AuthenticationEvent is an abstract superclass, and it has three subclasses. If the user registers or logs in to the application through the form, the event yields LoggedIn (). On the other hand, if the user logs out from the application, LoggedOut () event is returned from the AuthenticationEvent class.

The authenticationBloc class contains four methods. The mapAppStartedToState private method uses the User Repository class to observe if the user signed in when the application started. (Listing 5.15) Moreover, If the method returns true, the username received from the UserRepository class.

```
//when the application started it gets the username of the user and checks if the
//user signed in or registered.
Stream<AuthenticationState> _mapAppStartedToState() async* {
  try {
    final isSignedIn = await _userRepository.isSignedIn();//user signedin or not
    if (isSignedIn) {
      final name = await _userRepository.getName();//returns the username of user
      yield Authenticated(name);
    } else {
      yield Unauthenticated();
    }
  } catch (_) {
    yield Unauthenticated();
  }
}
```

Listing 5.15: Converting the events to states

Additionally, the User Repository class achieves the required operations between the database and the bloc class with different methods. For instance, the getName method returns the username of the current user by checking their documents. (Listing 5.16)

```
//returns the username of the user from cloud firestore
Future<String> getName() async {
  var userUid =  _firebaseAuth.currentUser.uid;
  DocumentSnapshot variable = await Firestore.instance.collection('users').doc(userUid).get();
  return variable.data()['username'];
}
```

Listing 5.16: Returning the username of the user from Cloud firebase

The Authorize class constructs the authenticationBloc using the BlocProvider element. (Listing 5.17) It enables authentication bloc to be used on the other classes with a call. Moreover, this class navigates the logged-in users to the home page and unauthenticated users to the login form.

```
child: BlocProvider(
  bloc: _authenticationBloc,
  child: MaterialApp(
    debugShowCheckedModeBanner: false,
    home:
    BlocBuilder(
      bloc: _authenticationBloc,
      builder: (BuildContext context, AuthenticationState state) {
        if (state is Unauthenticated || state is Uninitialized) { //if the user is unauthorized navigate to the login page
          return LoginScreen(userRepository: _userRepository);}
        else if (state is Authenticated) {
          return BottomNavigation(username: state.username, userRepository:_userRepository );} //if the user is authorized navigate to the home page
        return Container();},
      ), // BlocBuilder
  ) // MaterialApp
  ), // BlocProvider
); // WillPopScope
}}
```

Listing 5.17: Checking the state of the authentication from the UI

Since the Authenticationbloc defined in the Authorize class with the BlocProvider element, the call was done using the BlockProvider.of () method. Besides, when the user clicks on the List Tile, the request to the LoggedOut event of the Authenticationbloc is made, and the user navigates back to the Authorize page. Finally, if the state is unauthenticated, the Authorize class returns the user to the LoginForm class.

5.2.4- Home Page Requirement

To navigate to the different pages, a bottom navigation bar implemented to the application. Unlike login and signup pages, Streams used to control the state with the help of Bottomnavigation Bloc switch case statement. When the user logs in to the application, the landing page is called Home. The page aims to motivate user with quotes from famous people about mindfulness. Furthermore, the quotes change each time user opens the Home page.

The HomePage class does not need any state management technique. For this reason, it is a Stateless Widget. Firstly, an array list called quoteList initialized to store various quotes. (Listing 5.18) Moreover, the randomNumber variable is the type of integer and returns a random number between 0 to 11. By calling the quoteList Array with the randomNumber variable, the quote with the index randomNumber returns to the screen.

```
class HomePage extends StatelessWidget {
 var quotesList = [
 "\"The present moment is filled with joy and happiness. If you are attentive, you will see it.\" \n \n \n  ~Thich Nhat Hahn",
 "\"Wherever you are, be there totally.\"\n \n \n  ~Eckhart Tolle",
 "\"Be happy in the moment, that's enough. Each moment is all we need, not more.\"\n \n \n ~Mother Theresa",
 "\"Feelings come and go like clouds in a windy sky. Conscious breathing is my anchor.\"\n \n \n ~Thích Nhất Hạnh",
 "\"We have only now, only this single eternal moment opening and unfolding before us, day and night.\"\n \n \n ~Jack Kornfield",
 "\"The only way to live is by accepting each minute as an unrepeatable miracle.\"\n \n \n ~Tara Brach",
 "\"Mindfulness isn't difficult, we just need to remember to do it.\"\n \n \n ~Sharon Salzberg",
 "\"Every time we become aware of a thought, as opposed to being lost in a thought, we experience that opening of the mind.\"\n \n \n ~Joseph Goldstein",
 "\"The only thing that is ultimately real about your journey is the step that you are taking at this moment. That's all there ever is.\"\n \n \n ~Eckhart Tolle
 "\"In today's rush, we all think too much — seek too much — want too much — and forget about the joy of just being.\"\n \n \n ~Eckhart Tolle",
 "\"What would it be like if I could accept life—accept this moment—exactly as it is?\"\n \n \n ~Tara Brach",
 ];

 int randomNumber = Random().nextInt(11) + 0; //returning a random index number

  @override
  Widget build(BuildContext context) {
    return Container(
      decoration: BoxDecoration(
        image: DecorationImage(image: AssetImage("5.JPG"), fit: BoxFit.cover),
      ), // BoxDecoration
      child: Padding(
        padding: const EdgeInsets.all(60.0),
        child: Center(
          key: ValueKey("center1"),
          child: Text(
            quotesList[randomNumber], //showing the random quote on the screen
```

Listing 5.18: Implementing the Quotes list

5.2.5-Mindfulness Requirement

The process of implementing the Mindfulness Page begin with creating the Mindfulness Screen class. The class includes a text element with the “Swipe” value to inform the user about the navigation action. Additionally, with the Container and Box decoration elements, the boundaries of the child element defined. Ultimately, a call to the Mindfulness page executed.

In MindfulnessPage class, the Page View element used. (Listing 5.19) Horizontal scroll and physics achieved using the PageView widget. Moreover, it has six children to return to the screen.

```
class MindfulnessPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return PageView(
      children: <Widget>[
        Page1(),
        Page2(),
        Page3(),
        Page4(),
        Page5(),
        Page6(),

      ], // <Widget>[]
      scrollDirection: Axis.horizontal,
      pageSnapping: false,
      physics: BouncingScrollPhysics(),
      onPageChanged: (number) {
        print("page number is " + number.toString());
      },
    ); // PageView
  }
}
```

Listing 5.19: Page View with swipe horizontal element

Each child contains Container to achieve circular borders using the Box Decoration element. Applying the Column element, the Flexibility implemented. (Listing 5.20) The Mindfulness class intends to inform the user about mindfulness and using a technique to bring their attention to the moment.

```
class Page1 extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.all(8.0),
      child: Container(
        decoration: BoxDecoration(
          border: Border.all(color: Color(0xFF8CADCA), width: 4),
          borderRadius: BorderRadius.all(Radius.circular(60)),
          color: Color(0xFFFFFFFF),
        ), // BoxDecoration
        child: Column(
          children: [
            //Flexible(child: Center(child: SizedBox(height: MediaQuery.of(context).size.height / 20))),
            Flexible(
              child: Padding(
                padding: const EdgeInsets.only(right: 8.0, left: 8.0),
                child: Center(
                  child: Padding(
                    padding: const EdgeInsets.all(8.0),
                    child: Text(
                      "Mindfulness is the ability to become fully present with our thoughts, feelings and c
                      style: TextStyle(
                          fontSize: 18,
                          color: Colors.black54,
                          fontWeight: FontWeight.bold), // TextStyle
                      textAlign: TextAlign.center,
                    ), // Text
                  ), // Padding
                ), // Center
              ), // Padding
            ), // Flexible
            //Flexible(child: SizedBox(height: MediaQuery.of(context).size.height / 1000)),
          ],
        ), // Column
      ), // Container
    ); // Padding
    //color: Colors.red,
```

Listing 5.20: Implementing the child widgets of the pageview

5.2.6- Issues

5.2.6.1- Flutter bloc

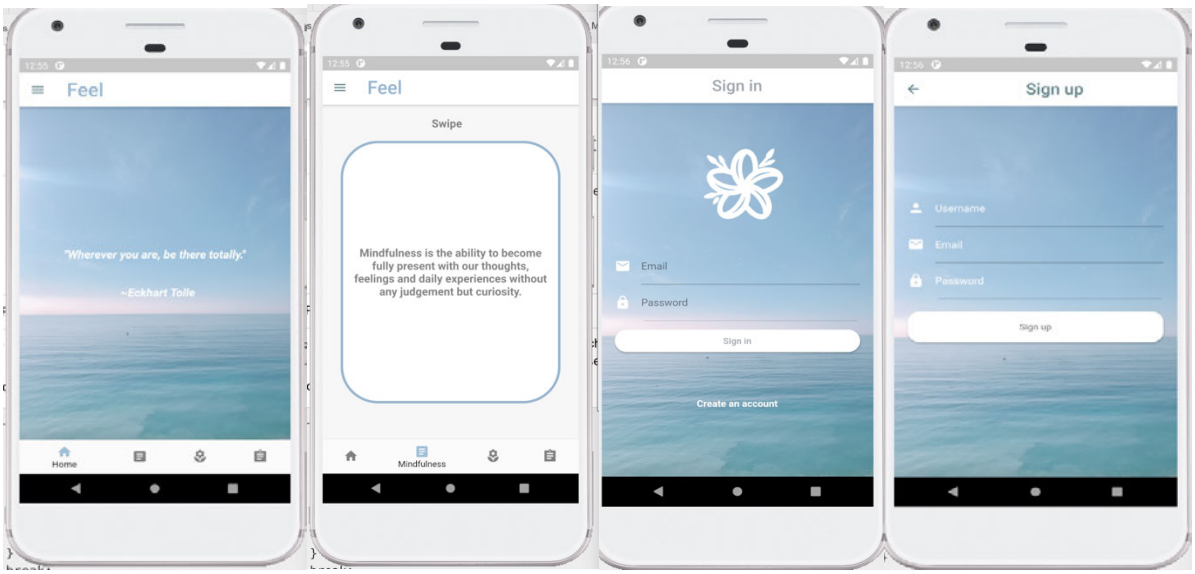
The main issue of the Sprint-2 was implementing the state management technique. The Flutter framework achieves the state management using the setState () function. However, this approach is unmaintainable and not testable. Due to this reason, the flutter bloc state management technique implemented. The Bloc state management approach made the classes more maintainable and testable. The added bloc packages simplified the implementation of the design pattern. However, it was still hard to implement. Due to timing constraints and the difficulty of the pattern, it didn't build from starch. However, the concept of the packages researched and learned from the documentation. ("Flutter_bloc | Flutter Package," n.d.), ("Flutter_bloc | Flutter Package," n.d.)

5.2.6.1- Usage of widgets

On the other hand, since the Flutter framework was unfamiliar, it was difficult to implement the planned design to the application. The flutter website and different resources were helpful while explaining the usage of each widget.

5.2.7- Result

As a result of the second sprint, login and sign-up pages implemented, along with the bloc design pattern and classes, and the bottom navigation bar enforced for the user to navigate to different pages using Stream management. Moreover, the Home page has built with various quotes. Finally, sprint-2 concluded by applying the Mindfulness page. Figure 5.2 represent the UI of the implemented pages.



Home page Mindfulness Page Login Page Sign-up page

Figure 5.2: The implemented pages

5.3- Sprint 3

In the third sprint, Unguided meditation and Self-reflection requirements completed, and the development of the application finalized. Moreover, the errors and problems resolved.

5.3.1- Unguided Meditation Requirement

The unguided meditation requirement sets a circular process indicator to represent each passing second. Moreover, it sets the time to leave to complete the meditation to the screen as a String. The bloc pattern used to implement the requirement, and event, state, bloc and UI classes added to the project.

First of all, the event classes appended to the project. (Listing 5.21) The TimerEvent class is abstract and contains five separate subclasses to define the user actions. The Start event indicates that the timer started, and the Pause event shows that the timer paused by the user.

```dart
abstract class TimerEvent extends Equatable {
  TimerEvent([List props = const []]) : super(props);
}

//the timer starts
class Start extends TimerEvent {
  final int duration;

  Start({@required this.duration}) : super([duration]);

  @override
  String toString() => "Start { duration: $duration }";
}

//the timer pauses
class Pause extends TimerEvent {
  @override
  String toString() => "Pause";
}

//the timer resumes
class Resume extends TimerEvent {
  @override
  String toString() => "Resume";
}

//resetting the timer
class Reset extends TimerEvent {
  @override
  String toString() => "Reset";
}
```

Listing 5.21: Timer Event class

TimerState class includes four child classes called Ready, Paused, Running and Finished. Each of the subclasses defines the state of the timer. Although three subclasses call the superclass with the duration parameter, the Finished class makes the call with 0 as duration to indicate the timer stopped working.

The time class responsible for setting the duration of the timer. Moreover, every time this class called is responsible for decrease the time every second. (Listing 5.22)

For example, assume you have a stream, `counterStream`, that emits an increasing counter every second. Here's how it might be implemented:

```dart
var counterStream =
    Stream<int>.periodic(Duration(seconds: 1), (x) => x).take(15);
```

Listing 5.22: Timer data(Nielsen, 2013)

Moreover, exploring this approach and different resources. The best technique found and implemented to decrease the timing every second. (Listing 5.23)

```dart
class Tick {

  Stream<int> tick({int ticks}) {
    return Stream<int>.periodic(
        const Duration(seconds: 1), (x) => ticks - x % (ticks + 1)); // Stream.periodic
  }
}
```

Listing 5.23: Timer data ("Dart - How to Update the Countdown Timer Repeatedly in Flutter - Stack Overflow," n.d.)

Moreover, TimerBloc class sets the minutes of the timer and converts the events to the states with Streams. The class starts by assigning the duration to ten minutes, and it creates a StreamSubscription to control and listen to the changes on the duration. (Listing 5.24)

```
//to listen the changes in the minutes
StreamSubscription<int> _minuteListener;
```

Listing 5.24: StreamSubscription variable

Next, the private stream function startToState () implemented. The primary purpose of the method is to start the StreamSubscription using the StreamSubscriber and Tick class. (Listing 5.25) The streamsubscription listens to the Tick class. Whenever there is a decrease in the duration, it dispatches the Second event. To implement the method, an article used.

```
//starts the timer
Stream<TimerState> _startToState(Start start1) async* {
  yield Running(start1.duration); //informs the running state
  _minuteListener?.cancel();      //disposes minute listener to start over
  //starts streamsubscription with the tick class
  _minuteListener = _tick.tick(ticks: start1.duration).listen(
        (duration) { dispatch(Second(duration: duration));
    },
  );
}
```

Listing 5.25: Starting the timer (Angelov, 2019b)

Three Stream methods added. The methods are responsible for pausing, resuming and cancelling the streamSubscription and return the corresponding state. Moreover, the subscription disposed of at the end of the duration.

The UnguidedMeditation page contains two variables with the type of Animation Controller and double. The Column element with three children to implement different functionalities. The first child sets the Text element to the screen and style it with the TextStyle widget. The second child builds the timer bloc using the BlocBuilder to receive the state of the duration. The first variable is minutesStr, and it divides the duration to 60 to return one digit value. (Listing 5.26) Moreover, it converts int value to String. It uses the padLeft function and puts 0 to the left of the duration variable to return a two-digit value.

```
final String minutesStr = ((state.duration / 60) % 60)
    .floor()
    .toString()
    .padLeft(2, '0');
final String secondsStr =
```

Listing 5.26: Manipulating the string to receive two-digit minute value

Moreover, the next child contains CircularProcessIndicator to create an indicator on the page. (Listing 5.27) The process indicator value is result/600. The number 600 represents the total duration of the timer, and the result is the minute and second time of the spent time. When the timer starts, it has the colour white, and the process turns the colour white to green. Finally, at the end of the meditation, the timer looks green.

```
Center(
  child: SizedBox(
    child: CircularProgressIndicator(
      backgroundColor:  Color(0xFF9FBDC4),
      valueColor: AlwaysStoppedAnimation<Color>(Colors.white),
      value: result / 600,
      strokeWidth: 20,
    ), // CircularProgressIndicator
    height: 200.0,
    width: 200.0,
  ), // SizedBox
) // Center
]); // <Widget>[], Stack
},
), // BlocBuilder
), // Padding
```

Listing 5.27: Circular Process indicator

Moreover, if the current state is Ready or Paused, it returns the play button and dispatches the Start event. However, if the state is Running, the pause icon set on the page instead of the play. Moreover, with the user action, it dispatches the Reset event. (Listing 5.28)

```
Padding(
  padding: const EdgeInsets.only(top: 0.0, left: 10),
  child: BlocBuilder(
      bloc: _timerBloc,
      builder: (context, state) {
        final TimerState state = _timerBloc.currentState;
        if (state is Ready || state is Paused) {
          return new IconButton(
            icon: new Icon(Icons.play_arrow),
            iconSize: 60,
            onPressed: () => _timerBloc.dispatch(
              Start(duration: state.duration),
            ),
          ); // IconButton
        } else if (state is Running) {
          return new IconButton(
            key: ValueKey("1"),
            icon: new Icon(Icons.pause),
            iconSize: 60,
            onPressed: () => _timerBloc.dispatch(Pause()),
          ); // IconButton
        } else if (state is Finished) {
          _timerBloc.dispatch(Reset());
        }
        return Container();
      }), // BlocBuilder
), // Padding
], // <Widget>[]
); // Column
}
```

Listing 5.28: Checking the state of the timer from UI

5.3.2- Self-Reflection Answer Requirement

The requirement aims to motivate the users to self-reflect by answering three questions. The questions are randomly chosen from a list. Moreover, the system stores the last three answers and the questions to the firebase cloud firestore. ReflectionEvent, ReflectionState and ReflectionBloc implement to achieve the pattern.

ReflectionEvent class contains two diverse public subclasses. The first class is called AddDatabase. Moreover, it has six variables to receive answers and questions. ReturnDatabase returns whenever the user navigates to the history tab. Finally, both classes use the super () function to send the details to the abstract superclass ReflectionEvent.

Furthermore, ReflectionState class is an abstract superclass, which has three subclasses. The purpose of the class is to define the state of the data. The first class returns if the data submitted to the database. Moreover, the next class has a parameter list. The list includes three questions and answers returned from the database.

Moreover, the ReflectionBloc class extends the Bloc class with the help of the bloc package. This class sinks the events and returns them to the states. (Listing 5.29) The initial state of the self-reflection data assigned to the Empty (). Moreover, the RegisterBloc contains three methods, and two of them are private. The mapEventToState calls a function depending on the user interaction of adding or returning the values from the database.

```
//with the help of bloc package
class ReflectionBloc extends Bloc<ReflectionEvent, ReflectionState> {
  final UserRepository _userRepository;

  ReflectionBloc({@required UserRepository userRepository})
      : assert(userRepository != null),
        _userRepository = userRepository;

  ReflectionState get initialState => Empty();

  //convert events to state
  @override
  Stream<ReflectionState> mapEventToState(
    currentState,
    ReflectionEvent event,
  ) async* {
    if (event is AddDatabase) {
      yield* _mapAddtextToDatabase(event.text1, event.text2, event.text3,
          event.question1, event.question2, event.question3);
    } else if (event is ReturnDatabase) {
      yield* _mapRetrieveTextfromDatabase();
    }
  }
```

Listing 5.29: Converting the event to state

To achieve the self-reflection answer requirement, the addTextToDatabase () method implemented in the User Repository class. The function receives variables to store the texts and questions. Update () method of the cloud firestore implemented to store the answers and questions with the date that the answers submitted. (Listing 5.30)

```
//Adding the answers and questions to the database of the user
Future <void> addTextToDatabase(String text1, String text2, String text3,
    String question1, String question2, String question3 ) async{
  var userUid =  _firebaseAuth.currentUser.uid;
  firestoreInstance.collection('users').doc(userUid).update({
    "reflection": {
      "post date": _now,
      question1: text1,
      question2: text2,
      question3: text3
    }
  });
}
```

Listing 5.30: Adding values to the database

The SelfReflection UI class aims to build the BlocProvider to use in the SelfReflectionAnswer and SelfReflectionQuestion classes. Moreover, it creates a TabBar for the user to navigate to the history and self-reflection pages. The TabBar contains two tabs with the String self-reflection and history. (Listing 5.31) Moreover, to navigate to the assigned pages, TabBarView implemented.

```
      bottom: TabBar(
        labelColor: Colors.black,
        tabs: [
          Tab(
            text: 'Self Reflection',
          ), // Tab
          Tab(
            text: 'History',
          ), // Tab
        ],
      ), // TabBar
    ), // AppBar
  ), // PreferredSize
  body: TabBarView(
    children: [
      SelfReflectionQuestion(),
      SelfReflectionAnswer(),
    ],
  ), // TabBarView
), // Scaffold
```

Listing 5.31: TabBar widget

The SelfReflectionQuestion class contains a List that includes seventeen questions and three variables with the type of int. Furthermore, each random generator responsible for creating the random number within the given range. Moreover, the TextEditingControllers help system to receive the input of the users. (Listing 5.32) Since there are three questions on the application, three controller variables implemented.

```
class _SelfReflectionQuestion extends State<SelfReflectionQuestion> {
  var questionList = [
    "Write the words you need to hear",
    "Make a list of everything that inspires you.",
    "What do you love about life?",
    "What do you need to forgive yourself for?",
    "What big changes do you want to make in your life?",
    "Do you take care of yourself today?",

    "What are you proud of?",
    "What do you want to let go of?",
    "What did you learn today?",
    "What are you experiencing emotionally right now in this very moment?",
    "What are you experiencing mentally right now in this very moment?",
    "What worries you most about the future?",

    "What matters most in your life?",
    "When did you last push the boundaries of your comfort zone?",
    "What do you most value in yourself?",
    "What do you want most in life?",
    "What is one thing you wish you did this year but didn't because you were afraid?",
  ];

  int randomNumber = 0 + Random().nextInt(6-0);
  int randomNumber2 = 6 + Random().nextInt(12-6);
  int randomNumber3 = 12 + Random().nextInt(17-12);


  final TextEditingController _textField1 = TextEditingController();
  final TextEditingController _textField2 = TextEditingController();
  final TextEditingController _textField3 = TextEditingController();
```

Listing 5.32: Question list, random generators and text controllers

Furthermore, ListView contains three children with the type of Text Field. Each of the TextField has a random question as a hint text. The questions acquired by calling the list with the index generated by the random generators. (Listing 5.33) Moreover, the controllers assigned to the corresponding TextField elements to receive the input of the user.

```
child: TextField(
  controller: _textField1,
  maxLines: 10,
  minLines: 2,
  autocorrect: false,
  decoration: InputDecoration(
    hintText: questionList[randomNumber],
    filled: true,
    fillColor: Color(0xFFCFFAFF),
```

Listing 5.33: Assigning the questions

The save button connects the presentation class with the reflection Bloc to add the user input to the database. Finally, the AddDatabase event of the bloc class called with the parameters of the questions and answers. (Listing 5.34) Since the Bloc registered on the SelfReflection class using the BlocProvider, the BlocProvider.of () method used.

```
BlocProvider.of<ReflectionBloc>(context)
    .dispatch(AddDatabase(
  question1: questionList[randomNumber],
  question2: questionList[randomNumber2],
  question3: questionList[randomNumber3],
  text1: _textField1.text,
  text2: _textField2.text,
  text3: _textField3.text,
)); // AddDatabase
```

Listing 5.34: Dispatching the add database event from UI

5.3.3- Self-Reflection Question Requirement

This requirement helps the user to see and reflect on their previous answers. The connection to the firestore cloud database achieved to return values, and the flutter bloc pattern built to separate the UI from the logic. To implement this requirement, the same event, state and bloc classes of the self-reflection answer requirement used.

In order to return the values from the Firebase Cloud database, a new function called getTexts () implemented to the User Repository class. The method obtains the document of the user using the userUid variable. Moreover, by calling the data () function with the responding database field, a list of question and answers returned. (Listing 5.35)

```
Future<List> getTexts() async{
  var userUid =  _firebaseAuth.currentUser.uid;
  DocumentSnapshot variable = await Firestore.instance.collection('users').doc(userUid).get();
  List checkin=
  [
    variable.data()['reflection'],
  ];
  return checkin;
}
```

Listing 5.35: Returning the elements from the database

The SelfReflectionAnswer class returns the stored questions and answers. Using the BlocBuilder element, the state of the ReflectionBloc returned. Moreover, if the state is Retrieved and the data is not null, the list returns from the Cloud Firestore. Additionally, the returned values modified using the replaceAll function and split with a comma separator. Moreover, a map is assigned to store the separated values and use them further in the class. (Listing 5.36)

```
return Container(
    color: Color(0xFF8CADCA),
    child: BlocBuilder(
        bloc: _reflectionbloc,
        builder: (BuildContext context, ReflectionState state) {
          BlocProvider.of<ReflectionBloc>(context).dispatch(ReturnDatabase());

          if (state is Retrieved && state.text1.isNotEmpty) {
            String a = state.text1.elementAt(0).toString();
            String b = a.replaceAll(new RegExp("{"), '');
            String c = b.replaceAll(new RegExp(":"), ',');
            final seperate = c.split(',');
            final Map<int, String> values = {
              for (int i = 0; i < seperate.length; i++) i: seperate[i]
            };
```

Listing 5.36: Manipulating the list to return the answers and questions

The column element has four child widgets. The first child sets the Text of the page using the Text () element. Moreover, three widgets use Container to set the background colour of the text elements. (Listing 5.37) Finally, the map with corresponding values called to position the previous question and the answers.

```
Flexible( //second question and answer
  child: Padding(
    padding: const EdgeInsets.only(
        left: 20.0, right: 20.0, top: 5.0, bottom: 5.0), //
    child: Container(
      width: MediaQuery.of(context).size.height / 1,
      height: MediaQuery.of(context).size.height / 9,
      color: Colors.white,
      child: Center(
        child: Text(values[3],
            style: TextStyle(fontSize: 15)), // Text
      ),),),), // Center, Container, Padding, Flexible
```

Listing 5.37: Representing the data on the screen

Although the state is Retrieved, the database can be empty. To prevent empty values on the page, if statement used. The first if statement checks to see if the state is retrieved, but the list is empty. (Listing 5.38) If this situation occurs, an error message returned to the screen.

```
} else if ((state is Retrieved && state.text1.elementAt(0) == null) || state is Submitted) {
  return Container(
    child: Center(
      child: Text(
        "Please answer your self reflection questions",
```

Listing 5.38: Checking if the database empty

5.3.4- App-Guide Requirement

The AppGuide class implemented to inform the user about the context of the application. The Scaffold widget used to build the app bar and the body of the page. While the AppBar element sets the title of the application, the Column widget places the text with the logo of the application. Finally, the centre element used to position the text on the centre of the container.

5.3.5- Issues

5.3.5.1- BlocProvider.of () method error

In this sprint, implementing the flutter bloc pattern was more challenging compared to Sprint 2. The first challenge was dealing with the error with "BlocProvider.of called with a context that does not contain a specific bloc". After researching the solution and learning more about the bloc, I found the error. The BlocProvider.Of () method must be used after the bloc class is instantiated and built using BlocBuilder or BlocProvider. (Listing 5.39) Moreover, using it before instantiating the bloc class was causing an error. After removing the BlockProvider.of () function and implementing the new technique, the error fixed. (Listing 5.40)

```
final TimerBloc _timerBloc = BlocProvider.of<TimerBloc>(context);
```

Listing 5.39: Previous approach

```
final TimerBloc _timerBloc = TimerBloc(ticker: Ticker());
```

Listing 5.40: The solution for the error

5.3.5.2- Not having a return value

The build function returned a null error caused by not providing a return value. BlocBuilder class must return a value. Due to this reason, not implementing a return value caused an error in the system. As a solution, an empty container returned every time the blocBuilder class used.

5.3.5.3- Data returned null

Additionally, "flutter/src/widgets/text.dart: Failed assertation 'data! = null'" error outputted from the system. The error was the returned data being null when it shouldn't be. The error was because the new register users do not have self-reflection data on the Cloud Firestore, and for that reason, the data returns null. To fix the error, a conditional statement

added. (Listing 5.40) The conditional statement not only checks the state of the bloc class. But also, it checks the data and returns a message if the data is empty.

```
if (state is Retrieved && state.text1.elementAt(0) == null) {
```

Listing 5.40: The new conditional statement

5.3.5.4- Upgrading the Framework

The final issue was upgrading the Flutter framework in the middle of the development phase. To make a function work, the “flutter upgrade --force” command used. This command caused the Flutter framework to upgrade to the latest version. The new version caused most of the functions to break, including the bloc classes. The emulators stopped working, and the app was not building. (Listing 5.41) To solve this issue, the Flutter framework downgraded using the “flutter downgrade” command. Additionally, a refactoring held to the bloc classes to solve the rest of the errors. Moreover, IOS pods removed and built again.

```
lib/blocks/reflection/reflection_bloc.dart:22:27: Error: The method 'ReflectionBloc.mapEventToState' has fewer positional arguments than those of overridden method 'Bloc
  .mapEventToState'.
  Stream<ReflectionState> mapEventToState(ReflectionEvent event,) async* {
                          ^
../../Developer/flutter/.pub-cache/hosted/pub.dartlang.org/bloc-0.8.4/lib/src/bloc.dart:56:17: Context: This is the overridden method ('mapEventToState').
  Stream<State> mapEventToState(State currentState, Event event);
                ^
lib/blocks/reflection/reflection_bloc.dart:22:59: Error: The parameter 'event' of the method 'ReflectionBloc.mapEventToState' has type 'ReflectionEvent', which does not match the
  corresponding type, 'ReflectionState', in the overridden method, 'Bloc.mapEventToState'.
  - 'ReflectionEvent' is from 'package:flutter_app/blocks/reflection/reflection_event.dart' ('lib/blocks/reflection/reflection_event.dart').
  - 'ReflectionState' is from 'package:flutter_app/blocks/reflection/reflection_state.dart' ('lib/blocks/reflection/reflection_state.dart').
Change to a supertype of 'ReflectionState', or, for a covariant parameter, a subtype.
  Stream<ReflectionState> mapEventToState(ReflectionEvent event,) async* {
                                          ^
../../Developer/flutter/.pub-cache/hosted/pub.dartlang.org/bloc-0.8.4/lib/src/bloc.dart:56:17: Context: This is the overridden method ('mapEventToState').
  Stream<State> mapEventToState(State currentState, Event event);
                ^
lib/blocks/register/register_bloc.dart:23:25: Error: The method 'RegisterBloc.mapEventToState' has fewer positional arguments than those of overridden method 'Bloc
  .mapEventToState'.
  Stream<RegisterState> mapEventToState(
                        ^
../../Developer/flutter/.pub-cache/hosted/pub.dartlang.org/bloc-0.8.4/lib/src/bloc.dart:56:17: Context: This is the overridden method ('mapEventToState').
  Stream<State> mapEventToState(State currentState, Event event);
                ^
lib/blocks/register/register_bloc.dart:24:21: Error: The parameter 'event' of the method 'RegisterBloc.mapEventToState' has type 'RegisterEvent', which does not match the
  corresponding type, 'RegisterState', in the overridden method, 'Bloc.mapEventToState'.
  - 'RegisterEvent' is from 'package:flutter_app/blocks/register/register_event.dart' ('lib/blocks/register/register_event.dart').
  - 'RegisterState' is from 'package:flutter_app/blocks/register/register_state.dart' ('lib/blocks/register/register_state.dart').
Change to a supertype of 'RegisterState', or, for a covariant parameter, a subtype.
```

Listing 5.41: Errors before refactoring the bloc classes.

5.3.6- Result

In this sprint, new classes and functionalities added, and unguided meditation, self-reflection and App Guide classes implemented. Moreover, the connection between the app and the Firestore cloud firebase made.

5.4- Summary

In this chapter, the implementation of the application, results and issues discussed. Moreover, the project divided into three sprints. In each sprint, the added features explained with the source code, and the results provided with images of the application. The sign-in, login, log-out functionalities added. Furthermore, the quotes, mindfulness, un-guided meditation and self-reflection pages implemented to build the application.

Moreover, this section also focused on the main problems encountered while building the application, such as framework up-gradation, flutter bloc pattern implementation, and data returned null error.

Chapter 6

Testing and Analysis

In this chapter, the testing plan, implementation and results are discussed. Moreover, the execution of the tests divided into three sprints.

The first and second sprint contains a widget and prototype testing the implemented features of the application on Sprint-1 and Sprint-2. Furthermore, the last sprint includes information about the implemented unit, widget, integrational, user and performance testing. Finally, the chapter concludes with the issues faced during the testing period and a discussion on the testing results.

6.1- Sprint 1

In this sprint, widget and prototype testing implemented. The widget testing executed to test the login and bottom page widgets. Moreover, the prototype testing performed with two users. The testing plan built and illustrated in the table below as represented on the Table 6.1.

Test No	Test Type	Test name	Purpose of test	Expected result	Actual Result	Outcome and actions required
1	Widget test	Login screen	Test the widgets and actions on the login page	-two input fields are found -the button found -the loginForm widget found -find inserted values as result -return an error on non-existed value	-as expected, -as expected, -as expected, -as expected, -instead of an error nothing found	All elements found and the correct results returned on the widgets. Moreover, non-existed values returned nothing
2	Widget test	Bottom navigation bar	Test the widgets and actions on the bottom navigation bar page	-all the corresponding elements have found -the unexpected variables returned nothing -the expected elements returned as widget -the test passed	-as expected, -as expected, -as expected, -as expected,	All elements found and the correct results returned on the widgets. Moreover, non-existed values returned nothing

Table 6.1: Sprint 1 testing plan

6.1.1- Widget Testing

The aim of widget testing on the first sprint is to test the login page and bottom bar widgets. The testing started with defining the test plan. The testing type, name, expected result defined using a test plan table. According to the test plan, the implementation of the test

started. As expected on the test plan, the inserted values returned as a widget. (Listing 6.1) Moreover, the non-existed values tested, and nothing has returned.

```
testWidgets("login form", (WidgetTester tester) async {

  await tester.pumpWidget(
    MaterialApp(
      home: Scaffold(
        body: BlocProvider(
          bloc: loginBloc,
          child: LoginForm(userRepository: user2,),
        ), // BlocProvider
      ), // Scaffold
    ), // MaterialApp
  );
/find widgets with the corresponding values
  final addusernamefield = find.byKey(ValueKey("emailfield1"));
  final addemailfield = find.byKey(ValueKey("passwordfield1"));
  final addloginbutton = find.byKey(ValueKey("submitloginbutton1"));

/insert values to the text form fields
  await tester.enterText(addusernamefield, "ece");
  await tester.enterText(addemailfield, "ece@gmail.com");
  await tester.tap(addloginbutton);
  await tester.pump();

/expect the inserted values to be returned
  expect(find.text("ece"), findsOneWidget);
  expect(find.text("mark"), findsNothing);
  expect(find.text("ece@gmail.com"), findsOneWidget);
  expect(find.text("mark@gmail.com"), findsNothing);
  expect(find.byKey(ValueKey("submitloginbutton")),findsOneWidget);
 });
```

Listing 6.1: Login form widget test

Listing 6.2: Login form widget test results

After executing the login form test, the result returned as success, and the expected results achieved. (Listing 6.2) Finally, the testing plan has completed with actual results and outcome. Furthermore, the same testing implementation applied for the bottomNavigationBar class and passed.

6.1.2- Prototype Testing

The initial prototype testing accomplished with two users. The purpose of the prototype testing was to observe if the user will understand the action they need to take, understand the flow and readability of the application and navigation bar icons. After the users tested the prototype, certain changes applied, and a new prototype created.

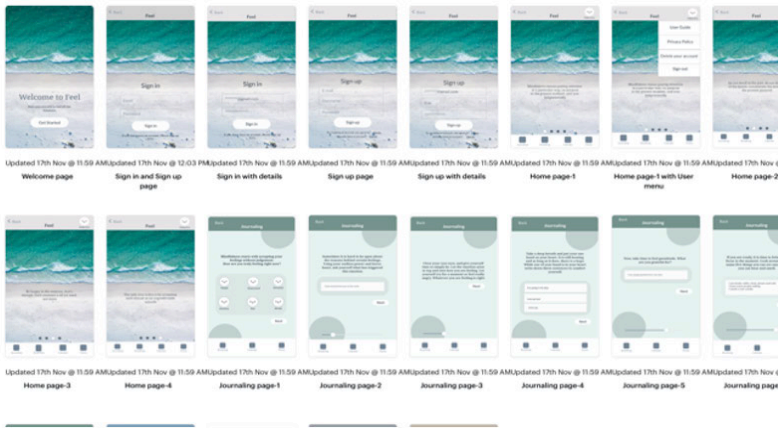


Figure 6.1: Initial Prototype

According to one user, the icons failed to represent their corresponding pages. Moreover, the text on the home screen was not very readable. On the other hand, the sea theme liked by the user. The second user stated that they couldn't understand how to navigate to see the different quotes on the homepage. Figure 6.1 represent the initial prototype of the project

After conducted a user test, the design is changed. The contrast achieved by using a dark background and white text. Moreover, instead of implementing multiple pages to return different quotes, one page built to show them randomly. Even though the sea theme still used on the application, the colours changed to a blue-white spectrum. The change in the colour pattern made the prototype more abstract, settle and suitable for the mindfulness theme. Figure 6.2 demonstrates the changes made on the design.

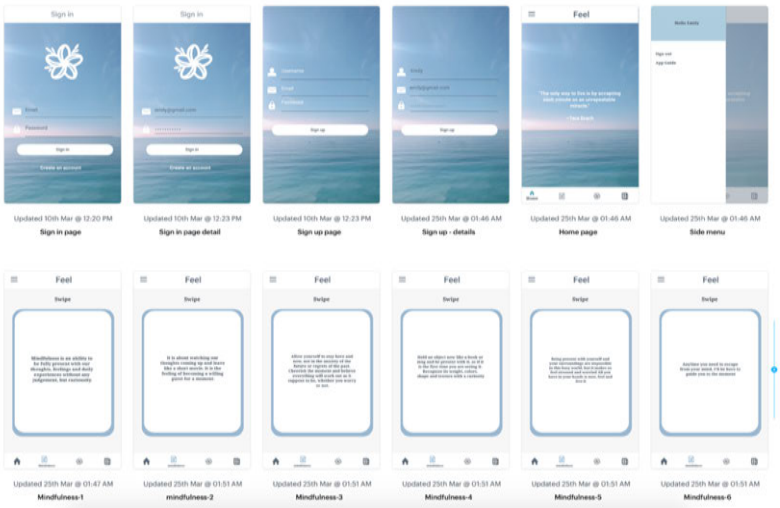


Figure 6.2: Current Prototype

6.2- Sprint 2

In this sprint, widget tests for register, mindfulness and homepage are planned and executed using the flutter test classes.

6.2.1- Widget Testing

The widget testing implemented to test the UI, functionality and action of the elements on the screen. Before starting to implement the widget test, the test plan for sprint-2 planned by using a table. The table contains the name, purpose of the tests, expected and actual results. The Table 6.2 shows the widget testing test plan.

Test No	Test Type	Test name	Purpose of test	Expected result	Actual Result	Outcome and actions required
1	Widget test	Register Form	Test the widgets and actions on the register page	-three input fields are found -the button found -the RegisterForm widget found -find inserted values	-as expected, -as expected, -as expected, -as expected,	All elements found and the correct results returned on the widgets. Moreover, non-existed values returned nothing
2	Widget test	Home page	Test the widgets and actions on the home page	-all the corresponding elements have found -the unexpected variables returned nothing -the expected elements returned as widget -the test passed	-as expected, -as expected, -as expected, -as expected,	All elements found and the correct results returned on the widgets. Moreover, non-existed values returned nothing
3	Widget test	Mindfulness	Test the widgets and actions on the mindfulness page	-pageview element found -children of the pageview element found -text, column, container of the page classes is found -tester return findsNothing for non-existed variables	-as expected, -as expected, -as expected, -as expected,	All elements found and the correct results returned on the widgets. Moreover, non-existed values returned nothing

Table 6.2: Sprint 2 widget testing plan

The same strategy for the sign-in class adopted to implement to test the registration form. Initially, a test plan built, and expected results identified. The variables expected to find by the tester assigned. The tester must not find the non-existed values on widgets. (Listing 6.3) As a result, the register form, home and mindfulness pages passed the test. Moreover, the testing plan completed.

```
testWidgets("register form", (WidgetTester tester) async {
  await tester.pumpWidget(
    MaterialApp(
      home: Scaffold(
        body: BlocProvider(
          bloc: registerBloc,
          child: RegisterForm(),
        ), // BlocProvider
      ), // Scaffold
    ), // MaterialApp
  );

  final addusernamefield = find.byKey(ValueKey("usernamefield"));
  final addemailfield = find.byKey(ValueKey("emailfield"));
  final addpasswordfield = find.byKey(ValueKey("passwordfield"));
  final addregisterbutton = find.byKey(ValueKey("registerbutton"));

  await tester.enterText(addusernamefield, "ece");
  await tester.enterText(addemailfield, "ece@gmail.com");
  await tester.enterText(addpasswordfield, "ece125");
  await tester.tap(addregisterbutton);
  await tester.pump();

  expect(find.text("ece"), findsOneWidget);
  expect(find.text("ece@gmail.com"), findsOneWidget);
  expect(find.text("ece125"), findsOneWidget);
  expect(find.text("ecfsfd@gmdfsfail.com"), findsNothing);//non-existed
  expect(find.text("ecesfdsfsdferf125"), findsNothing);//non-existed
  expect(find.byKey(ValueKey("registerbutton")),findsOneWidget);
});
```

Listing 6.3: Register form widget test

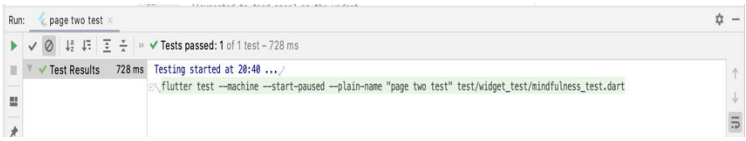
Listing 6.4: Register form widget test results

In the Flutter framework, everything is a widget, and widgets create widget trees. The Pageview class contains a Pageview element with six children. However, only the first children are found and represented on the widget tree and caused an error while trying to reach the second and third children. (Listing 6.5) Due to this reason, the pageview element implemented with different children to test other widgets. (Listing 6.6)

```
testWidgets("page two test", (WidgetTester tester) async {
  await tester.pumpWidget(
      MaterialApp(
          home: Scaffold(
              body: PageView(
                  children: [
                    Page1(),
                    Page2(),
                    Page3(),
                    Page4(),
                    Page5(),
                    Page6(),
                  ],
                ) // PageView
            ) // Scaffold
      ) // MaterialApp
  );

  final text1 = find.byType(Page1);//second page
  final text2 = find.byType(Page2);//second page

  await tester.pump();
  expect(text1, findsOneWidget);
  expect(text2, findsOneWidget);
});
```

Listing 6.5: Page 1 test with Pageview element

```
//should  return the second widget
testWidgets("page two test", (WidgetTester tester) async {
  await tester.pumpWidget(
      MaterialApp(
          home: Scaffold(
              body: PageView(
                  children: [
                    Page2(),
                    Page3(),
                    Page4(),
                    Page5(),
                    Page6(),
                  ],
                ) // PageView
            ) // Scaffold
      ) // MaterialApp
  );

  final text1 = find.byType(Page2);//second page
  final text2 = find.byType(Page3);//third page

  await tester.pump();
  //find the page 2 and cannot find the page 3
  expect(text1, findsOneWidget);
  expect(text2, findsNothing);
});
```



Listing 6.6: Page 2 test with Pageview element

6.3- Sprint 3

In this sprint, unit, integrational, widget, user, prototype and performance testing conducted. Moreover, Flutter_driver and flutter_test dependencies added to implement the integrational testing.

6.3.1- Unit Testing

The bloc class tested using the unit test to see the functionality of each part. Authentication, login, reflection, register, timer and user authentication blocs, event and state classes tested. Moreover, the table below illustrates the unit testing plan for sprint 2. Table 6.3 demonstrates the unit testing plan for Sprint 3.

Test No	Test Type	Test name	Purpose of test	Expected result	Actual Result	Outcome and actions required
1	Unit testing	authentication	To check if the functionalities of the authentication bloc class is working	-the initial state is Uninitialized -when the authentication bloc dispatches the login event, the states are going to be Uninitialized and authenticated -when the authentication bloc dispatches the loggedOut event, the states are going to be Uninitialized and Unauthenticated	-as expected, -as expected, -as expected,	No further action required to taken
2	Unit testing	login	To check if the functionalities of the authentication bloc class and the validator are working	-the initial state is LoginInitial -when the email checked with a valid email, the login bloc states are login initial and login success -when the email checked with an invalid email address, validator returns error, and the state is EmailFailure - when the password checked with a validpassword, the login bloc states are login initial and SuccessEmailPassword - when the password checked with an invalid password, validator returns error, and the state is PasswordFailure	-as expected, -as expected, -as expected, -as expected, -as expected,	No further action required to taken
3	Unit testing	reflection	To check if the functionalities of the reflection bloc class is working	-the initial state is Empty -when the reflection bloc dispatches the add database event, the states are going to be Empty and Retrieved -when the reflection bloc dispatches the return database event, the states are going to be Submitted and retrieved	-as expected, -as expected, -as expected,	No further action required to taken
4	Unit testing	register	To check if the functionalities of the register bloc class and the validator are working	-the initial state is Empty Register -when the email checked with a valid email, the login bloc states are emptyRegister and RegisterSuccessEmailPassword -when the email checked with an invalid email address, validator returns error, and the states are EmptyRegister and RegisterEmailFailure - when the password checked with a validpassword, the login bloc states are emptyRegister and RegisterSuccessEmailPassword - when the password checked with an invalid password, validator returns error, and the states are EmptyRegister and RegisterPasswordFailure	-as expected, -as expected, -as expected, -as expected, -as expected,	No further action required to taken

5	Unit testing	timer	To check if the functionalities of the timer bloc class is working	-the initial state is Ready -when the timer bloc dispatches the Start event, the states are going to be Ready, Running -when the reflection bloc dispatches the Pause event, the states are going to be Ready(), Paused() - when the reflection bloc dispatches the Reset event, the states is going to be Ready()	-as expected, -as expected, -as expected, as expected,	No further action required to taken
6	Unit testing	User authentication	To check if the functionalities of the user repository class is working	-when the loginbloc dispatch with loginwithcredentialpressed event, the user repositorys signup with email is called and returns success - when the loginbloc dispatch with loggedout event, the user repositorys signout function is called and returns success - when the loginbloc dispatch with signinwithemail event, the user repositorys signinwithemailandpassword function is called and returns success	-as expected, -as expected, -as expected,	No further action required to taken

Table 6.3: Sprint 3 unit testing plan

Authentication, reflection, timer classes tested to check if the corresponding bloc class functionalities work as planned. To achieve the aim, the bloc classes dispatched an event, and the tests implemented to check if the state of the function will return the expected result. Moreover, the expected states compared with the actual states. (Listing 6.7) As a result of the Authentication, reflection, timer test classes, all the tests passed, and the test plan achieved. The code illustrates the conducted tests.

```
group('LoggedOut', () {
  test(
      'emits [uninitialized, unauthenticated] when the user logs out',
          () {
        final expectedResponse = [
          Uninitialized(),//first state
          Unauthenticated(),//second state
        ];
        expectLater(
          authenticationBloc.state,
          emitsInOrder(expectedResponse),
        );
        authenticationBloc.dispatch(LoggedOut());
            //athentication bloc is
            //called with the event loggedOut
      });
```

```
Testing started at 01:19 ...
 flutter test --machine --start-paused --plain-name "emits [uninitialized, authenticated] when the user loggs in" test/unit_test/athentication_test.dart
```

Listing 6.7: log-out test

Initially, the first tests for login and register classes implemented. However, the files couldn't pass the tests. (Listing 6.8) (Listing 6.9) The actual and expected results were the same, but an error was returning. After various research and debugging, the issue couldn't resolve. The best strategy was finding a new technique to implement the State class, which will provide

ease in testing. After converting the factory classes into abstract classes, the tests passed, and the unit testing completed. (Listing 6.10) (Listing 6.11)

Listing 6.8: Previous state class

Listing 6.9: error

```
abstract class LoginState extends Equatable{
  @override
  List<Object> get props => [];
}

class LoginInitial extends LoginState {}

class LoginLoading extends LoginState {}

class EmailFailure extends LoginState {}

class SuccessEmailPassword extends LoginState {}

class PasswordFailure extends LoginState {}

class LoginSuccess extends LoginState {}

class LoginFailure extends LoginState {
  final String error;

  LoginFailure({@required this.error});

  @override
  List<Object> get props => [error];
}
```

Listing 6.10: Current state class

Listing 6.11: the test passed

6.3.2- Integrational Testing

Research has made on Integrational testing types. Moreover, the top-down technique was chosen to test the application. The top-bottom approach starts by testing the priority modules first and move to the details. (Jorgensen & Erickson, 1994) Moreover, a testing plan created to implement the integrational testing. The table on the Figure 6.4 shows the testing plan.

Test No	Test Type	Test name	Purpose of test	Expected result	Actual Result	Outcome and actions required
1	integrational testing	App_test	-tap the fields, enter the email and password details and login to the app	-reach the homepage and see the title "Feel"	-as expected,	No further action required to taken
2	integrational testing	App_test	-tap the fields, enter the email, username and password details and sign up to the app	-reach the homepage and see the title "Feel	-as expected,	No further action required to taken
3	integrational testing	App_test	Find the navigation bar and click on the mindfulness page	-enter the mindfulness page and found the text Swipe	-as expected,	No further action required to taken
4	integrational testing	App_test	Click on the meditation page. Start the timer and stop	-the timer started and stopped	-as expected,	No further action required to taken
5	integrational testing	App_test	Click on the journaling page. Answer the tree question, click the button to save it	-the journaling page opened; the questions answered, the button clicked. The results are stored on the database	-as expected,	No further action required to taken

Table 6.4: Sprint 3 integrational testing plan

Flutter_driver dependency added to the project to execute the integrational testing. Furthermore, the testing started with the priority functionality. The login and sign-in tests built to reach the home page. Moreover, after achieving the tasks, the tests scenario prepared to test each of the pages using the bottom navigation bar. Each of the pages opened, and the expected results outputted. (Listing 6.12) Finally, the questions on the journaling page answered and details checked using the database. The code illustrates the approach taken to implement the integrational tests.

```
test('Open meditation page, play the meditation timer and stop', () async {

      await driver.tap(find.text('Meditation'));
      await new Future.delayed(const Duration(seconds: 5));
      //start meditation
      await driver.tap(find.byValueKey("playicon"));
      await new Future.delayed(const Duration(seconds: 5));
      await driver.tap(find.byValueKey("stopicon"));
 });
```

Listing 6.12: Meditation page integrational testing

Finally, the integrational testing wanted to be completed by signing the user out of the application. However, the driver failed to find the first child of the ListView element. Research made on how to get the element from the widget. However, no information found. Due to this reason, the sign-out functionality hasn't tested.

6.3.3- Prototype Testing

The new prototype design conducted with the three users. The users recruited from the slack group, and an invitation paper that explain the purpose of the testing sent to the users. Moreover, each of the user's e-mail addressed, name and surname kept private to protect their information during and after the interactive prototype testing. Furthermore, after the users receive the introduction to testing e-mail, a document shared through google drive to explain the test scenario, feedback questions, and a link to the interactive prototype on the Marvel webpage. The test implemented to make sure the user understood the icons, enjoy the colour schema and the design, and took the right actions to achieve their goals. Furthermore, instead of creating a task, a scenario created to make users more comfortable with the test.

Three users found the application beneficial. However, all of them were confused about the icons. One user stated that the prototype caused problems during the swipe action.

Overall, the design of the application liked by the users, and the main problem was the icons on the bottom navigation bar. The icons changed to make the application better.

6.3.4- Widget Testing

In order to test the classes to check the UI functionality, App guide, Self-reflection, SideMenu and Unguided meditation tests implemented. Moreover, a test plan generated before the execution of the widget tests. Table 6.4 illustrates the widget testing plan for Sprint 3

Test No	Test Type	Test name	Purpose of test	Expected result	Actual Result	Outcome and actions required
1	Widget testing	appguide	To find appbar element, title, column, container and the text	-Appbar, title, column, container and test return as widget -the non-existed test returned as found nothing	-as expected, - as expected,	No further action required to taken
2	Widget testing	Self-reflection	To find text and container	-the zero indexed text and container returned as widget -the searched fist indexed test returned as found nothing	-as expected, - as expected,	No further action required to taken
3	Widget testing	SideMenu	Find the texts, list view, drawer header and list tile elements	-the existed texts, list view, drawer header and list tile elements are found -the non-existed texts are not found	-as expected, -as expected,	No further action required to taken
4	Widget testing	UnguidedMeditation	-finds the icons, column, stack, padding, circluarprocessindicator	-the icons, column, stack, padding, circluarprocessindicator found -the non-existed stop button couldn't find	-as expected, -as expected,	No further action required to taken

Table 6.4: Sprint 3 widget testing plan

According to the testing plan, the widget test implemented. The existed elements on the UI classes found, and the non-existed elements returned as nothing found. Moreover, the plan completed, and the expected results achieved.

6.3.5- User Testing

Initially, the user testing planned to achieve as moderated and through zoom application. However, after researching user testing, the unmoderated approach was more beneficial. With the unmoderated user testing, the users will be more comfortable while achieving the scenario. Google play internal testing services used to conduct the test.

Three users from a slack group volunteered for the internal testing. Moreover, a document to inform users about the purpose of the test and the data privacy sent. The e-mail addresses, name and other information of the user kept private during and after the testing. Moreover, after the introduction document, the users added to the internal testing list, and they received a link to the application. A file shared with the users to explain the scenario and questions.

The design liked by the two users and especially the process indicator effect. However, one user stated that using more colours would make it more attractive in the marketplace. While two of the users registered, navigate to each page and sign-out, one user faced a bug. Even though the user answered the questions and saved their answers, the history page didn't return the correct result and yield different fields of the user's database. After reading the feedback, the application tested with different devices and the other two testers contacted again. The functionality was working on every emulator, real device and the other two testers device. The reason presumed to be the internet connection of the user.

6.4- Issues

The primary issue during the testing phase was to connect the test files to the firebase system. The methods implemented to register the Firebase application. (Listing 6.13) However, even though the methods initialized, the same error returned from the application. (Listing 6.14) Research made on how to fix the issue.

```
//main function to internilize the firebase and call the app class
WidgetsFlutterBinding.ensureInitialized();
await Firebase.initializeApp();
```

Listing 6.13: Initializing the firebase application

```
Failed to load "/Users/ececaliskan/AndroidStudioProjects/flutter_app/test/widget_test.dart                [core/no-app] No Firebase App '[DEFAULT]' has been created - call Firebase.ini
tializeApp()
  package:firebase_core_platform_interface/src/method_channel/method_channel_firebase.dart 118:5          MethodChannelFirebase.app
  package:firebase_core/src/firebase.dart 52:41                                                           Firebase.app
  package:cloud_firestore/src/firestore.dart 43:21                                                        FirebaseFirestore.instance
  package:flutter_app/userRepository/userRepository.dart 7:47                                             new UserRepository
  userRepository_test.dart 24:42                                                                          main
  ===== asynchronous gap ===========================
  package:test_api                                                                                        RemoteListener.start
  /var/folders/cb/1fp1x6q13sqgjms45l4npqc40000gn/T/flutter_tools.zpLcbs/flutter_test_listener.abbmET/listener.dart 16:25  serializeSuite
  /var/folders/cb/1fp1x6q13sqgjms45l4npqc40000gn/T/flutter_tools.zpLcbs/flutter_test_listener.abbmET/listener.dart 41:36  main
```

Listing 6.14: The error of not initialized Firebase app

After the research, various people faced the same issue, and it was due to a bug with the Mock classes and the Firebase application. Moreover, a new file added to the project from GitHub to achieve the connection. (Listing 6.11) As a result, the connection achieved with the method from the file.

```
typedef Callback = void Function(MethodCall call);

void setupFirebaseAuthMocks([Callback customHandlers]) {
  TestWidgetsFlutterBinding.ensureInitialized();

  MethodChannelFirebase.channel.setMockMethodCallHandler((call) async {
    if (call.method == 'Firebase#initializeCore') {
      return [
        {
          'name': defaultFirebaseAppName,
          'options': {
            'apiKey': '123',
            'appId': '123',
            'messagingSenderId': '123',
            'projectId': '123',
          },
          'pluginConstants': {},
        }
      ];
    }

    if (call.method == 'Firebase#initializeApp') {
      return {
        'name': call.arguments['appName'],
        'options': call.arguments['options'],
        'pluginConstants': {},
      };
    }

    if (customHandlers != null) {
      customHandlers(call);
    }

    return null;
```

Listing 6.11: The code from the GitHub ("Flutterfire/Mock.Dart at Master · FirebaseExtended/Flutterfire · GitHub," n.d.)

6.5- Summary

This chapter focused on the widget, prototype, user, integrational and unit testing of the application. Furthermore, the results discussed, and the expected results achieved according to the test plan. Finally, the chapter concludes with a review of the issues during the testing phase.

Chapter 7

Critical Evaluation

This chapter contains a personal evaluation of the project. Moreover, the review of the project objectives, plan, evaluation, reflection on the previous deliverables, and the lessons learnt sections discussed. Finally, the chapter concludes with a summary of the chapter.

7.1- Review of Objectives

The review of objectives chapter discusses the objectives identified in the project proposal.

7.1.1 Learn Flutter framework and Dart language

I successfully managed to learn the Flutter framework and dart language from different video tutorials, books and websites. The video tutorials helped start the Flutter framework. I discovered about mobile development and Flutter and used the knowledge during the implementation.

During the development process, I found the learning by doing technique more beneficial than researching everything before the development. Due to this reason, I could have spent less time focusing on this objective.

7.1.2 Design the User Interface

Initially, I planned to use Sketch software to design the mobile application. However, after downloading and using it, the layouts and the procedure of creating a prototype seemed complicated. Because of this reason, I researched various resources to find a better tool. After the research, I discovered the Marvel website and created the design.

However, after receiving comments from the users and exploring text readability problems with the background picture, the design changed during Sprint 1.

7.1.3 Implement the mindfulness application

I managed to implement a mindfulness application using the Flutter framework with the pages in compliance with the requirements and the design prototypes in three sprints. However, some functionalities, such as an alert when the meditation starts, and ends couldn't add to the project. The audio player plugin must be implemented in the project to play audio sounds. Moreover, I built the functionality, and it worked on the emulators. Nevertheless, the package caused a problem while building the apk. I research the solution; however, I couldn't identify the reason for the error. Due to this reason, the package removed, and this functionality couldn't add to the application.

Moreover, I wanted to implement the google, Facebook and Twitter sign-in functions. However, this requirement was not a priority. For this reason, I couldn't implement it and spend time on writing the dissertation and completing other assignments.

The errors I faced during the implementation slowed the implementation process more than expected. Due to this reason, it would have been better to spend less time on the first objective and more time while implementing the extra requirements.

7.1.4 Implement the unit, widget, integrational and user testing

I implemented the unit, widget and integrational testing successfully in three sprints. Firstly, I watched video tutorials to learn how to conduct the test, then started the implementation of the tests. The main problem during unit testing was testing the State class that included the factory classes. Accordingly, the state class refactored, and the tests conducted successfully.

7.1.5 Deploy the application

The application app bundle built and released to the google store. Since it is my first time developing an application, I didn't realize that the example.aaa workplaces are private and cannot deploy to the studio. I built the application with the workplace named com.example.flutter_app, and the google play store returned an error. To deploy the application to the play store, I changed the workplaces of the project, firebase and android to

com.ece.flutter_app. I created the key to sign the application and build the app bundle. Finally, I was able to upload the release to the google store for review and release. Initially, I was planning to release the application on both google play and the apple store and didn't realize the cost of the Apple developer program. Due to this reason, the application only realized on the google play store.

7.2 Review of the Plan

The Agile methodology adopted, and the project divided into three sprints. Moreover, before starting the implementation of the project backlog created and each sprint and product backlog planned. Moreover, a Gantt chart also built to plan the project. The project started by learning about flutter and dart. However, since I didn't write it as an objective, I forget to add it to the Gantt chart on the project proposal. Due to this reason, a new and more accurate Gantt chart created. Figure 7.1 and 7.2 represents the Trello board and Gantt Chart.

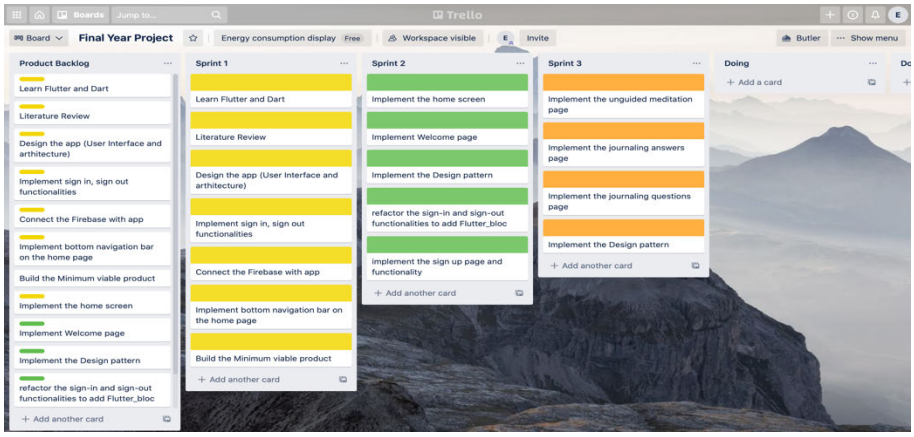


Figure 7.1: Initial Trello board

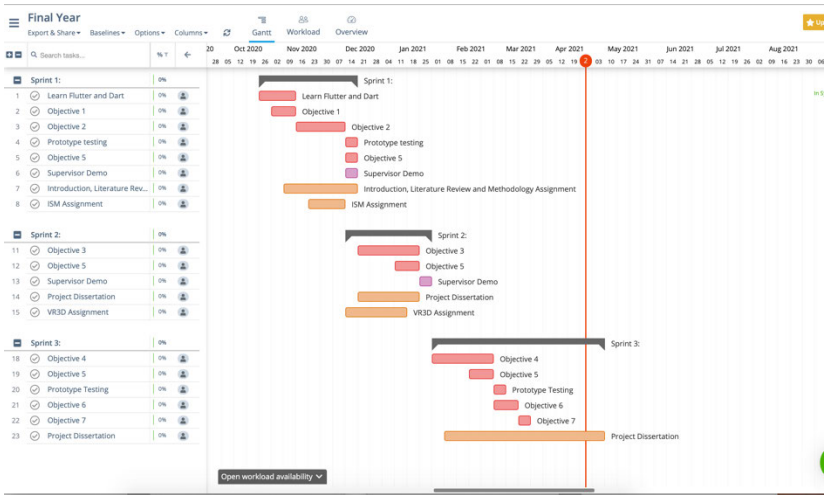


Figure 7.2: Improved Gantt chart

Overall, the project achieved in parallel with the Gantt chart. The objective, sprint start and end dates of the first, second and third sprints as planned. Moreover, even though I built an extra mindfulness page on the second sprints, I completed the project on time.

Moreover, the prototype testing planned to implement on the first sprint and added to the Gantt chart. However, the prototype testing on the second sprint and third sprint did not plan. After receiving feedback from supervisors, I decided to conduct prototype testing on the third iteration before the internal testing. The decision added to the new Gant-chart on the Sprint 3.

Implementing integrational, widget and unit tests took much more time than expected due to some difficulties with the system and other assignments. Releasing the app to the play store postponed one month, but it did not affect other objectives since it was the last task on the Gantt chart. I could have assigned more time to the testing objective on the third sprint.

Moreover, the project dissertation writing must have started on 8 February. Since I wasn't complete with the development, I didn't want to write and change the dissertation when something on the application changes. Due to this reason, the dissertation writing started in the first week of April.

The Gantt chart hasn't been modified since the second sprint because I found the Trello website more beneficial than the Gantt chart. While the agile Trello board used for tasks, the Gantt chart was beneficial for time management. The Trello board was easy to use and better at illustrating the things I need to do and the objectives I achieved.

7.3 Evaluation of the Product

After completing the project and releasing it to the Play Store, I am happy with the application. The most beneficial thing I did during the implementation was only using the UI and state management together on the first sprint and refactoring it on the second sprint easily. If I continued building the application without the state management technique and implement the design pattern at the end, it would have been a very challenging task. Using the state management from the beginning enabled me to separate the app states from the UI easily.

However, the audio functionality on the meditation page couldn't apply to the project. According to different forums, some people received the same error with the package and received an error while building the apk. This situation did not affect the overall quality of the product. However, I would have been happier to have it on my application.

Moreover, I am also happy with the code that I wrote and the organization of the files. I think I achieved to maintain a clean structure and code from the beginning of the project.

7.4 Lessons Learnt

First and foremost, I learned how to deal with the difficulties that occur during the development phase. Especially, upgrading the framework was one of the most impactful experiences. The emulators stopped building, the application returned almost fifty errors, and nothing was working. Initially, I didn't know the reason, and it left me feeling confused. However, dealing with this situation and solving it taught me how to react to difficult situations during programming.

I also learned the importance of changing approaches and refactoring when it is needed. I kept getting an error about a test that the expected and the actual results seem identical. Moreover, I debugged the test file and researched it on the internet but couldn't find a solution. To apply the test, I implemented the classes in another approach and passed the test. This situation helped me to understand how to be open-minded to new implementation approaches and refactoring.

Moreover, I also realized the importance of user testing. After I conducted prototype testing on the first iteration, I wasn't planning to manage prototype testing on the second or third sprint. However, after receiving feedback from the supervisors, I planned prototype testing on the third sprint and made sure that my design liked by the users before the internal test.

7.5 Summary

I achieved the objectives and executed the project plan successfully. Moreover, learning a new programming language and more about mobile development was challenging. However,

I believe the challenges I faced during the project helped me learn more about Flutter, dart and mobile development.

Chapter 8

Conclusion

The project aimed to build a mindfulness application using the Flutter framework and the flutter_bloc library pattern. As a result, the application built and released to the Google Play Store for review. Furthermore, before the deployment, widget, integrational, user, unit and prototype tests conducted to experiment if the functionalities work as expected.

Flutter Framework is a competitor to React Native, which helps developers to build native cross-platform applications. Working with this framework will be easier for the developers who have prior mobile development experience. Since the Flutter framework still considered new, various tutorials can be found on the internet. However, the lack of developers of the framework caused the information to be in the same content. Moreover, it is hard to experience and see the different versions of implementation of the state management techniques or functions.

Furthermore, the bloc state management technique is beneficial for separating the presentation layer from the state. Moreover, the bloc pattern can be implemented from scratch using _bloc library to learn more about the logic, instead of the flutter_bloc. Because there are not enough resources on how to implement the pattern without the library, the beginner Flutter developers must start with the flutter_bloc to learn the basic implementation first.

The application can be improved to the marketplace standards by adding interactivity, where the users can text each other, and follow their mindfulness progress. The firebase instant messaging services can be useful for achieving this objective. Moreover, this feature will allow the user to spend more time on the application, and texting will separate the application from the others on the marketplace. Moreover, to add more social media presence, the application can be connected to the applications like Facebook and Twitter.

Consequently, even though the Flutter framework is new, it can be used to build the native cross-platform applications and produce a high-quality result as React native.

References

Angelov, F. (2019a). Firebase Login with “flutter_bloc” | by Felix Angelov | Flutter Community | Medium. Retrieved May 3, 2021, from https://medium.com/flutter-community/firebase-login-with-flutter-bloc-47455e6047b0

Angelov, F. (2019b). Flutter Timer with “flutter_bloc” | by Felix Angelov | Flutter Community | Medium. Retrieved April 13, 2021, from https://medium.com/flutter-community/flutter-timer-with-flutter-bloc-a464e8332ceb

Angelov, F. (2020). Bloc. Retrieved December 9, 2020, from https://bloclibrary.dev/#/

Baer, R. A., Smith, G. T., Hopkins, J., Krietemeyer, J., & Toney, L. (2006). Using self-report assessment methods to explore facets of mindfulness. *Assessment*, *13*(1), 27–45. https://doi.org/10.1177/1073191105283504

Biørn-Hansen, A., Rieger, C., Tor, ·, Grønli, M., Majchrzak, T. A., & Gheorghita Ghinea, ·. (2020). An empirical investigation of performance overhead in cross-platform mobile development frameworks. *Empirical Software Engineering*, *25*, 2997–3040. https://doi.org/10.1007/s10664-020-09827-6

Bishop, S. R., Lau, M., Shapiro, S., Carlson, L., Anderson, N. D., Carmody, J., … Devins, G. (2006). Mindfulness: A Proposed Operational Definition. *Clinical Psychology: Science and Practice*, *11*(3), 230–241. https://doi.org/10.1093/clipsy.bph077

Cebolla, A., Campos, D., Galiana, L., Oliver, A., Tomás, J. M., Feliu-Soler, A., … Baños, R. M. (2017). Exploring relations among mindfulness facets and various meditation practices: Do they work in different ways? *Consciousness and Cognition*, *49*, 172–180. https://doi.org/10.1016/j.concog.2017.01.012

dart - How to update the countdown timer repeatedly in flutter - Stack Overflow. (n.d.). Retrieved April 13, 2021, from https://stackoverflow.com/questions/60377519/how-to-update-the-countdown-timer-repeatedly-in-flutter

Fayzullaev, F. J. (2018). *Native-like Cross-Platform Mobile Development*.

flutter_bloc | Flutter Package. (n.d.). Retrieved April 15, 2021, from https://pub.dev/packages/flutter_bloc

Flutter. (n.d.). Flutter documentation - Flutter. Retrieved December 4, 2020, from https://flutter.dev/docs

Flutter - Beautiful native apps in record time. (n.d.). Retrieved December 12, 2020, from

https://flutter.dev/

Flutter architectural overview - Flutter. (n.d.). Retrieved December 9, 2020, from https://flutter.dev/docs/resources/architectural-overview

flutterfire/mock.dart at master · FirebaseExtended/flutterfire · GitHub. (n.d.). Retrieved May 3, 2021, from https://github.com/FirebaseExtended/flutterfire/blob/master/packages/firebase_auth/firebase_auth/test/mock.dart

Gray, C. M., Kou, Y., Battles, B., Hoggatt, J., & Toombs, A. L. (2018). The dark (patterns) side of UX design. *Conference on Human Factors in Computing Systems - Proceedings*, *2018-April*, 1–14. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3173574.3174108

Jabangwe, R., Edison, H., & Duc, A. N. (2018). Software engineering process models for mobile app development: A systematic literature review. *Journal of Systems and Software*, *145*, 98–111. https://doi.org/10.1016/j.jss.2018.08.028

Joorabchi, M. E., Mesbah, A., & Kruchten, P. (2013). Real challenges in mobile app development. *International Symposium on Empirical Software Engineering and Measurement*, 15–24. https://doi.org/10.1109/ESEM.2013.9

Jorgensen, P. C., & Erickson, C. (1994). Object-Oriented Integration Testing. *Communications of the ACM*, *37*(9), 30–38. https://doi.org/10.1145/182987.182989

Kabakuş, A. T. (2019). A Performance Comparison of SQLite and Firebase Databases from A Practical Perspective. *Düzce Üniversitesi Bilim ve Teknoloji Dergisi*, *7*(1), 314–325. https://doi.org/10.29130/dubited.441672

Khawas, C., & Shah, P. (2018). Application of Firebase in Android App Development-A Study. In *International Journal of Computer Applications* (Vol. 179). Retrieved from https://www.firebase.com/login/

Kumar, P. (2014). *Analysis of Native and Cross-Platform Methods for Mobile Application Development [Whitepaper]*.

Lv, J., Xu, S., & Li, Y. (2009). Application research of embedded database SQLite. *Proceedings - 2009 International Forum on Information Technology and Applications, IFITA 2009*, *2*, 539–543. https://doi.org/10.1109/IFITA.2009.408

Marshall, L. (2020). Emerging evidence on COVID-19's impact on mental health and health inequalities | The Health Foundation. Retrieved December 17, 2020, from https://www.health.org.uk/news-and-comment/blogs/emerging-evidence-on-covid-19s-impact-on-mental-health-and-health

Meet Android Studio | Android Developers. (n.d.). Retrieved December 12, 2020, from https://developer.android.com/studio/intro

Nielsen, L. (2013). Creating streams in Dart | Dart. Retrieved April 13, 2021, from https://dart.dev/articles/libraries/creating-streams

Nunkesser, R. (2018). Beyond Web/Native/Hybrid: A New Taxonomy for Mobile App Development - IEEE Conference Publication. Retrieved December 2, 2020, from 2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft) website: https://ieeexplore-ieee-org.salford.idm.oclc.org/document/8543456

Plaza, I., Demarzo, M. M. P., Herrera-Mercadal, P., & García-Campayo, J. (2013). Mindfulness-Based Mobile Applications: Literature Review and Analysis of Current Features. *JMIR Mhealth and Uhealth*, *1*(2), e24. https://doi.org/10.2196/mhealth.2733

Rahul Raj, C. P., & Tolety, S. B. (2012). A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. *2012 Annual IEEE India Conference, INDICON 2012*, 625–629. https://doi.org/10.1109/INDCON.2012.6420693

Rešetár, M. (n.d.). Bloc Library – Painless State Management for Flutter - Reso Coder. Retrieved April 15, 2021, from https://resocoder.com/2019/06/12/bloc-library-updated-painless-state-management-for-flutter/

RIGAU, X. (2018). Introduction to Redux in Flutter. Retrieved December 9, 2020, from https://blog.novoda.com/introduction-to-redux-in-flutter/

Roquet, C. D., & Sas, C. (2018). *Evaluating Mindfulness Meditation Apps*. https://doi.org/10.1145/3170427.3188616

Shah, K., Sinha, H., & Mishra, P. (2019). Analysis of Cross-Platform Mobile App Development Tools. *2019 IEEE 5th International Conference for Convergence in Technology, I2CT 2019*. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/I2CT45611.2019.9033872

Stoll, S. (2019). Flutter Tutorial: Pros and Cons of popular State Management Approaches | Codemagic Blog. Retrieved April 15, 2021, from https://blog.codemagic.io/flutter-tutorial-pros-and-cons-of-state-management-approaches/

Testing Flutter apps - Flutter. (n.d.). Retrieved December 13, 2020, from https://flutter.dev/docs/testing

Wu, W. (2018). React Native vs Flutter, cross-platform mobile application frameworks. *Metropolia University*, (March), 28. Retrieved from https://ieeexplore-ieee-org.salford.idm.oclc.org/stamp/stamp.jsp?tp=&arnumber=9033872

Appendices

Appendix A

Project Logbook

Project Logbook

----------SPRINT 1----------

30/09/2020

Lecture
Duration: 50 minutes

Today, we had our first final year project lecture. Dr Murray talked about the importance and content of the logbook about mentioned deadlines. He asked us to pick our final year project grouping by the end of this week.

I don't have a project idea in my mind yet. However, I will try to attend all of the seminars to see different kind of project ideas and supervisors. I believe, this way I will make the best decision.

Seminar
Duration: 50 minutes

Today, Dr Bass and Gaber talked about different project ideas. But they most likely encouraged us to find something we passionate about to be more enthusiastic about the project. We discussed that the project should challenge us, but it should still be possible to achieve within the time limit. They advised us to research the literature to have some project ideas.
One of my passions is mindfulness and being present. I will be so happy to make something related to this topic. I will research the project ideas next week. Another project idea might be a management system or a book exchange application. However, I am not very sure.

01/10/2020

Independent work
Duration: 30 minutes

Today, because I feel more passionate about the mindfulness topic, I brainstormed various project ideas around it. The best option would be to build an application. A web application would not be very useable for the users. Consequently, I chose to develop a mobile application.
Until next week I want to learn more about different kinds of mobile applications to decide which one would be suitable for my project. I decided to research different mobile application types.

03/10/2020

Independent work
Duration: 1 hour 20 minutes

I researched different application ideas. I preferred a hybrid application rather than a native since I want to deploy the app for both the App Store and google store and maintain only one codebase. A couple of months ago, I listened to a talk about the Flutter development kit, and I was interested in learning the dart language and Flutter. I think that would be very beneficial for me in the future. Next week, I want to talk with the supervisors to get their opinion.

Seminar
Duration: 50 minutes

After I talked about my idea to Dr Julian Bass and Dr Tarek Gaber, they both like the topic of mindfulness and found it significant. I gained so much insight from both of the supervisors. Dr Bass advised me to enhance the idea by adding a database server element which will be beneficial for me while authenticating the users. He mentioned some issues around testing it in both android and iPhone.

Dr Gaber advised me to explore existing mindfulness applications and improve them in some ways such as security or privacy.

After the meeting, the project overall starts making sense, and I feel very excited to start. This week, I will do more research about Flutter and React Native to see the best development platform for my application. Moreover, I will download other mindfulness apps to see what they are lacking and how I can improve them.

11/10/2020

Independent work
Duration: 40 minutes

I researched more about the Flutter and React Native app today, and I read some articles about the comparison of both development tools. Most of the writers were talking about the future of the flutter app development and its positive sides compared to React Native. I learned that Flutter provides more testing abilities such as widget and integration level and it is easier to maintain because of the hot reloading feature.

I believe developing a project in Flutter and learning the dart language will be very beneficial for me for my future employment, even though it seems very challenging for me right now.

In the upcoming seminar, I would like to discuss the project proposal and Gantt chart with the supervisors.

14/10/2020

Lecture
Duration: 1 hour

Today, Dr Julian Bass gave us a lecture about different development methodologies and advised us to choose incremental while developing our project. He strongly advised us to have at least three increments and at the end of each one encouraged us to get feedback from the supervisors. Each increment has a plan, design, build, test and deploy phases. While choosing the right development methodology for my project, I will be able to maximise productivity and minimise the risks.

Moreover, He talked about Agile ceremonies that may be useful in our project such as, sprint planning, retrospectives and demo to the supervisors, the requirements, Architecture styles, implementation choices, testing and finally deployment to the available platforms.
This week I am planning to find the best development method which will help me to minimise the risks and start with my project proposal.

Seminar
Duration: 1 hour

This week other people in the seminar talked about their ideas and got feedback from the supervisors. We talked about Project proposals and Gantt Chart. Supervisors advised us to have a sensible number of rows in the Gantt Chart and informed us more about how to present incremental and waterfall model development in it.
Today I e-mailed Dr Bass and Dr Tarek my overall project description and logbook to get feedback from them. Today I will also start my project proposal and research different development methods.

15/10/2020

Independent work
Duration: 2 hours

Today, I downloaded Flutter and set it up on the Android studio. I tested the connection with both IOS and Android devices to make sure it works well with both of them. Moreover, I researched Waterfall and Agile development methodologies and decided to use Agile. It will allow me to make changes, and I will be able to meet the customer requirements.
This week I will finish the Project proposal and start learning Flutter and Dart language.

17/10/2020

Independent work
Duration: 1 hour

Today, I finished the Project Proposal and tried to explore Flutter by looking at the source code and changing various attributes. Moreover, I e-mailed Supervisors my draft project proposal to get their feedback.

21/10/2020

Lecture
Duration: 50 minutes

Today, Dr Chris Hughes gave a presentation about various research methods, research process and motivation. He strongly advised us to review the literature before starting the project to avoid researching or doing something which is already existed in the literature. Moreover, we can expand our projects by reading articles. He informed us how to and where to find information and how to record them.

Seminar
Duration: 1 hour

This morning I received feedback for my draft project proposal from my supervisors. So, I didn't have anything to ask. I listened to other people's project ideas and questions as well as Supervisor's advice on how to write the proposal and objectives.

Independent work
Duration: 1 hour 40 minutes

I reviewed the comments on my draft project proposal from supervisors and made changes to improve it.

24/10/2020

Independent work
Duration: 1 hour

I completed my Risk assessment and Ethical Approval form and got them ready to send on Monday.

28/10/2020

Lecture
Duration: 50 minutes

Dr Chris Hughes talked about various research methods such as experimental, simulation, theoretical and social. Also, he defined a good methodology and stated that it must be robust, reliable and repeatable. He talked about Research strategy and talked about naïve, informed and refined approaches and mentioned the importance of methodology.

Seminar
Duration: 1 hour

After submitting the project proposal, the general idea of the mobile application was clearer. However, the only question I have in mind is how I can add different features and make the app better than the ones on the marketplace in privacy or usability ways. Since I have an idea about the basic functionality of the application, the literature review will give me more ideas about how to enhance the functionalities.

We talked about the literature review in the seminar as well as the next steps. Supervisors advised us to dedicate at least two days a week for the final year project to be consistent and successful. I asked whether I need to research about mindfulness theme for the literature review in addition to the technologies.

This week, I am planning to start learning Flutter through different YouTube tutorial as well as LinkedIn learning.

01/11/2020

Independent Work
Duration: 2 hours

Today, I started a YouTube tutorial for Flutter and Dart. After two hours of studying, I found it very useful. I learned the basics of the Dart language, as well as the technical information about Flutter. Moreover, I discover Stateless widgets, build function, Material App, scaffold, App Bar, title and basic Text widget.

Flutter Crash Course for Beginners 2020 - Build a Flutter App with Google's Flutter & Dart

https://www.youtube.com/watch?v=x0uinJvhNxI&t=5465s

02/11/2020

Independent Work
Duration: 1 hour

I continued to follow the tutorial and learned more about Stateful Widgets, lists, built method buttons and finally created an app.

03/11/2020

Independent Work
Duration: 2 hours 50 minutes

Today, I continued with the tutorial and learned how to create and manage multiple dart files. I discovered how to call other classes, use styling elements, text-align, font size, container width and margins.

I discovered a tutorial on styling and building app. This tutorial will be very beneficial for me while learning.

https://flutter.dev/docs/codelabs

Moreover, I started designing the app using the Marvel website. When I create an account on Fluid UI, I found it hard for the user design and prototype. I started researching better tools and started using Marvel. The design idea prototype of the welcome page.

04/11/2020

Lecture
Duration: 50 minutes

Today, we talked about literature review and technical writing. I learned that there are two types of writing approaches, Top-down and bottom-up. Moreover, Dr Bass advised us to keep our sentences short, simple and positive. He mentioned paragraphs and topic sentences. For the literature review, we need to start with more broad journals, magazines and articles. Then move towards more specific ones such as journal articles.

Seminar
Duration:50 minutes

Today supervisors recommended that I research technical aspects like native cross-platform mobile applications and also the mindfulness theme. They advised me to investigate the apps in the marketplace.

07/11/2020

Independent Study
Duration: 1 hour

Today, I researched the comparison between native cross-platform, native and hybrid. Furthermore, I investigated the mindfulness theme and found the following resources.

- An empirical investigation of performance overhead in cross-platform mobile development frameworks

  https://doi.org/10.1007/s10664-020-09827-6

- Analysis of Cross-Platform development tools

  https://ieeexplore.ieee.org/document/9033872

- React Native vs Flutter, cross-platform mobile application framework

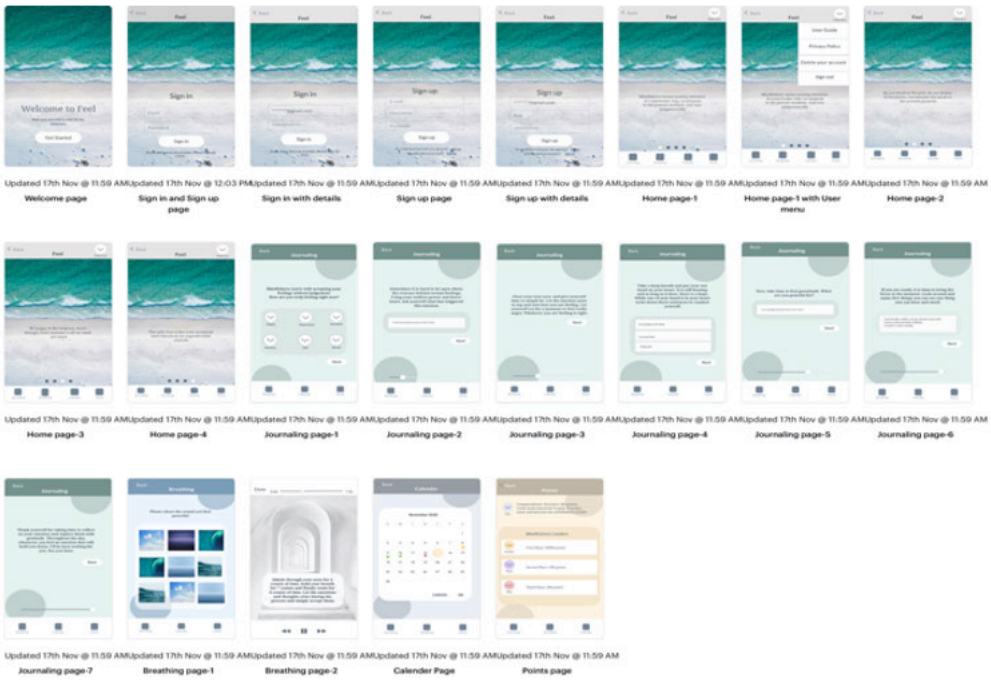
https://ieeexplore-ieee-org.salford.idm.oclc.org/stamp/stamp.jsp?tp=&arnumber=9033872

Moreover, I continued with the Flutter tutorial and learned about HashMap. I also discovered more about creating one more file, buttons with String. Now the buttons look better.

09/11/2020

Independent Study
Duration: 2 hours

Today, I finished the User interface design of the app and completed the prototype. Moreover, I planned the first increment of the project logbook using Trello. The end product of the first increment will be the welcome, user sign-in and sign-up pages. Moreover, I also want to connect the app to the database.



Sprint 1 prototype

11/11/2020

Seminar

Duration: 40 minutes

Today, we discussed our progress in the project. I talked about finishing the user interface of the app. This week, I will start researching some architectures and design patterns for the Flutter app while continuing my literature review. Supervisors have decided to turn seminars into a system where everyone gives a short talk about what they did during the week and what they will do the next. This system will be convenient for everyone, I believe.

14/11/2020

Independent Study

Duration: 50 minutes

Today I started writing my introduction part of the First dissertation. So far, I have done the motivation and objectives. Moreover, I researched some design patterns such as MVC, MVCC and BLOC. I don't know which one to choose for now. However, since Block is very popular and more complex than MVC, I am leaning towards choosing it to challenge myself and learn new things.

18/11/2020

Lecture
Duration: 50 minutes

This week, we learned about different software architectural styles such as Data Abstraction and object organisation, pipes and Filters, Layered systems, client-server, publish-subscribe, peer to peer style and REST. I am planning to use Block as the software architecture method on my project.

Seminar
Duration: 50 minutes

Today we attended our first stand-up meeting. We talked about what have we done so far, what are we going to do the next week and any blockages. Moreover, we discussed the literature review part of the project. Next week, I will work on the architectural design of my project, start the introduction part of the report and learn more about MVP.

19/11/2020

Independent Study
Duration: 30 minutes

Today, I completed writing the introduction part of the Literature review and researched the minimum viable product. Furthermore, I learned its difference from the prototype. Next week, I will start working on the literature review part of the dissertation.

22/11/2020

Independent Study
Duration: 30 minutes

Today, I started writing the literature review part of the assignment and researching more about the differences between mobile development technologies. Moreover, I will continue working on the literature review.

25/11/2020

Lecture
Duration: 50 minutes

Today, Lee Griffiths gave a presentation about the requirement engineering process. He talked about the importance of understanding the requirements to develop a successful project. The requirements can be functional or non-functional. Functional ones illustrated with

Use case diagrams, and they describe what the system must do. On the other hand, response times and security are non-functional requirements.

Seminar
Duration: 50 minutes

In the seminar, we talked about what we did stand up meetings. Due to other assignments, most people feel pressure. However, time management is a lesson we must all learn. Next week, I will continue on the literature review, and I will research the differences between the cross-platform and native applications.

26/11/2020

Independent study
Duration: 40 minutes

Today, I completed writing the comparison between the native and cross-platform technologies. I will write the comparison between the cross-platform technologies such as Flutter, React native and hybrid applications.

30/11/2020

Independent study
Duration: 1 hour

After finishing writing the comparison of the cross-platform technologies, today, I focus more on Flutter, widgets and state management techniques.

03/12/2020

Independent study
Duration: 1 hour

I finished the flutter, widget and state management sections. Moreover, I started writing about databases.

05/12/2020

Independent study
Duration: 3 hours

I completed the literature review after writing about mindfulness and mindfulness application comparison. I will continue on the methodology part after the Dependable software engineering assignment.

11/12/2020

Independent study
Duration: 3 hours

Today I completed the literature review and the methodology part of the assignment and sent it to supervisors to get feedback. I also finished a Use-case diagram to start the development process.

13/12/2020

Independent study
Duration: 3 hours

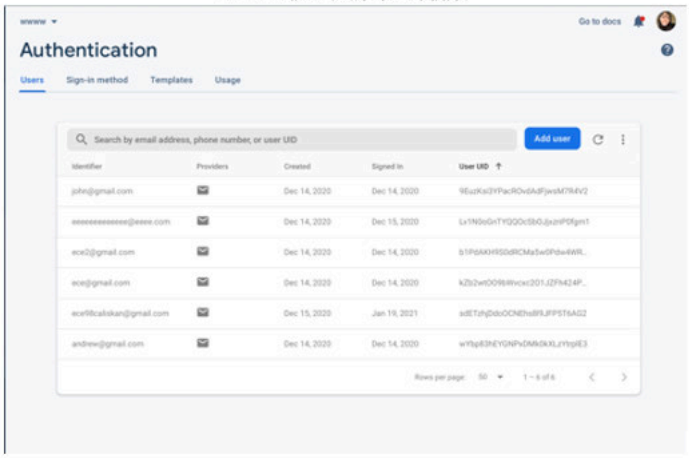
Today, I started developing the minimum viable product. According to the supervisors, it is essential to make sure that the data can be acquired and pass to different pages. I decided to start with storing the user details and calling authentication services to accomplish sign in, sign up and sign out functions. After researching the databases, I discovered the SQLite database requires much effort. On the other hand, the Firebase authentication service provides user authentication operation with a minimum effort. So, I decided to use the Firebase service to store the user. The next step of the project is to connect the application with the Firebase auth and achieve the operations.

14/12/2020 - 15/12/2020

Independent study
Duration: 10 hours

I connected the app with the firebase authentication system without any design patterns. The first two attempts were not successful. Moreover, I was getting error for not initialising a Firebase instance. For this reason, I questioned the database choice. I finally accomplished the connection by defining a Firebase app on the Main function. I added buttons to sign-in, sign-up and sign-out the users and implemented the bottom navigator bar to navigate different pages. Right now, the Minimum viable product is complete for Wednesday.

https://www.youtube.com/watch?v=lhZGf0vcG7Y => Firebase authorization



Firebase authentication page

15/12/2020 - 18/12/2020

Independent study

Duration: 5 hours

Today, I started to widget test the bottom navigation bar and the login form. However, I kept getting an error with the message saying Missing plugging exception.

```
  Failed to load "/Users/ececaliskan/AndroidStudioProjects/flutter_app/test/userRepository_test.dart": MissingPluginException(No implementation f
ound for method Firebase#initializeCore on channel plugins.flutter.io/firebase_core)
  package:flutter/src/services/platform_channel.dart 157:7                    MethodChannel._invokeMethod
  ===== asynchronous gap ===========================
  package:flutter/src/services/platform_channel.dart 332:12                   MethodChannel.invokeMethod
  package:flutter/src/services/platform_channel.dart 345:41                   MethodChannel.invokeListMethod
  package:firebase_core_platform_interface/src/method_channel/method_channel_firebase.dart 30:36  MethodChannelFirebase._initializeCore
  package:firebase_core_platform_interface/src/method_channel/method_channel_firebase.dart 75:13  MethodChannelFirebase.initializeApp
  package:firebase_core/src/firebase.dart 43:25                               Firebase.initializeApp
  userRepository_test.dart 33:18                                              main

00:02 +2 -1: Some tests failed.
ECE-MacBook-Pro:flutter_app ececaliskan$
```

MissingPluginException Error

```
await Firebase.initializeApp();
```

The solution to the MissingPluginException Error

After researching on the internet, I found the solution to fix the error. However, after solving it, I came across a new problem.

No firebase App has been created an error

After researching how to fix the second error, I found out it was a common problem. I discovered this solution on the Stack Overflow.

https://stackoverflow.com/questions/63662031/how-to-mock-the-firebaseapp-in-flutter

I had the same problem. Using this answer I found the solution.

1. Copy the contents of: https://github.com/FirebaseExtended/flutterfire/blob/master/packages/firebase_auth/firebase_auth/test/mock.dart into a file, that you can import into your tests, where you need to initialize a Firebase app.
2. Call `setupFirebaseAuthMocks();` at the top of your `main` function for all your tests.
3. In your `setUpAll` function, call `await Firebase.initializeApp();` (You can also put this in your `main` function under `setupFirebaseAuthMocks();` will still work).

Now you should have a mocked Firebase app.

Here is a full example:

The solution on the StackOverflow website

I added the source code to my test directory to mock the firebase

```
typedef Callback = void Function(MethodCall call);

void setupFirebaseAuthMocks([Callback customHandlers]) {
  TestWidgetsFlutterBinding.ensureInitialized();

  MethodChannelFirebase.channel.setMockMethodCallHandler((call) async {
    if (call.method == 'Firebase#initializeCore') {
      return [
        {
          'name': defaultFirebaseAppName,
          'options': {
            'apiKey': '123',
            'appId': '123',
            'messagingSenderId': '123',
            'projectId': '123',
          },
          'pluginConstants': {},
        }
      ];
    }

    if (call.method == 'Firebase#initializeApp') {
      return {
        'name': call.arguments['appName'],
        'options': call.arguments['options'],
        'pluginConstants': {},
      };
    }

    if (customHandlers != null) {
      customHandlers(call);
    }

    return null;
```

```
    if (call.method == 'Firebase#initializeApp') {
      return {
        'name': call.arguments['appName'],
        'options': call.arguments['options'],
        'pluginConstants': {},
      };
    }

    if (customHandlers != null) {
      customHandlers(call);
    }

    return null;
  });
}

Future<T> neverEndingFuture<T>() async {
  // ignore: literal_only_boolean_expressions
  while (true) {
    await Future.delayed(const Duration(minutes: 5));
  }
}
```

Added file to connect the firebase services to the application

Github
https://github.com/FirebaseExtended/flutterfire/blob/master/packages/firebase_auth/firebase_auth/test/mock.dart

```
Future<void> main() async {
  setupFirebaseAuthMocks();
  await Firebase.initializeApp();
```

Calling the added file to setup the firebase instance

After resolving the issue, I started to implement the widget test for login and bottom navigation pages.

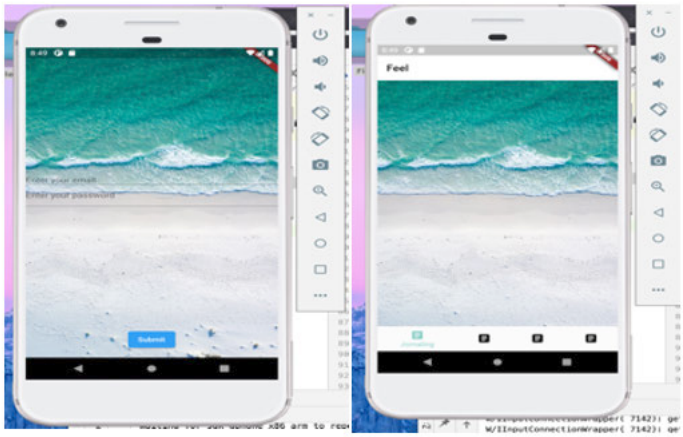
```
testWidgets("bottom navigationbar", (WidgetTester tester) async {
  await tester.pumpWidget(
    MaterialApp(
      home: Scaffold(

        //body: BlocProvider(
        // bloc: loginBloc,
          body: BottomNavigation(userRepository: user2, username: "ece",
            //adding the class bottomnavigation to the test
          ), // BottomNavigation
        ), // Scaffold
      ), // MaterialApp
    //),
  );

  final findAppBar = find.byType(AppBar);
  final findTitle = find.text("Feel");
  final findItem1 = find.text("Home");
  final findIcon1 = find.byIcon(Icons.home);
  final findItem2 = find.text("Mindfulness");
  final findIcon2 = find.byIcon(Icons.article_rounded);
  final findItem3 = find.text("Meditation");
  final findIcon3 = find.byIcon(Icons.local_florist);
  final findItem4 = find.text("Journaling");
  final findIcon4 = find.byIcon(Icons.assignment_sharp);
  final bottombar = find.byKey(ValueKey("bottomnavigation"));
  final findItem5 = find.text("Nothing");
  await tester.pump();

  expect(findTitle, findsOneWidget);
  expect(findAppBar,findsOneWidget);
  expect(findItem1,findsOneWidget);
  expect(findIcon1,findsOneWidget);
  expect(findItem2,findsOneWidget);
  expect(findIcon2,findsOneWidget);
  expect(findItem3,findsOneWidget);
  expect(findIcon3,findsOneWidget);
  expect(findItem4,findsOneWidget);
  expect(findIcon4,findsOneWidget);
  expect(findItem5,findsNothing);
```

Bottom navigation test

The test finds the corresponding texts, types and icons inside the class and returns a widget if they exist.

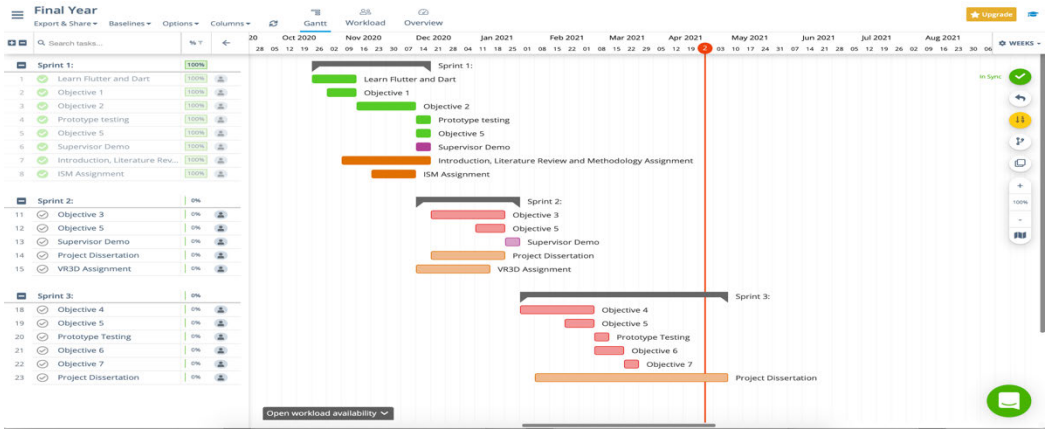
```dart
Future<void> main() async {
  setupFirebaseAuthMocks();
  await Firebase.initializeApp();

  final UserRepository user2 = UserRepository();
  final loginBloc = LoginBloc(userRepository: user2);

  testWidgets("login form", (WidgetTester tester) async {

    final addusernamefield = find.byKey(ValueKey("emailfield"));
    final addemailfield = find.byKey(ValueKey("passwordfield"));
    final addloginbutton = find.byKey(ValueKey("submitloginbutton"));

    await tester.pumpWidget(
      MaterialApp(
        home: Scaffold(
          body: BlocProvider(
            bloc: loginBloc,
            child: LoginForm(userRepository: user2,),
          ), // BlocProvider
        ), // Scaffold
      ), // MaterialApp
    );
    await tester.enterText(addusernamefield, "ece");
    await tester.enterText(addemailfield, "ece@gmail.com");
    await tester.tap(addloginbutton);
    await tester.pump();

    expect(find.text("ece"), findsOneWidget);
    expect(find.text("mark"), findsNothing);
    expect(find.text("ece@gmail.com"), findsOneWidget);
    expect(find.text("mark@gmail.com"), findsNothing);
    expect(find.byKey(ValueKey("submitloginbutton")),findsOneWidget);
  });
}
```

Widget testing the login page

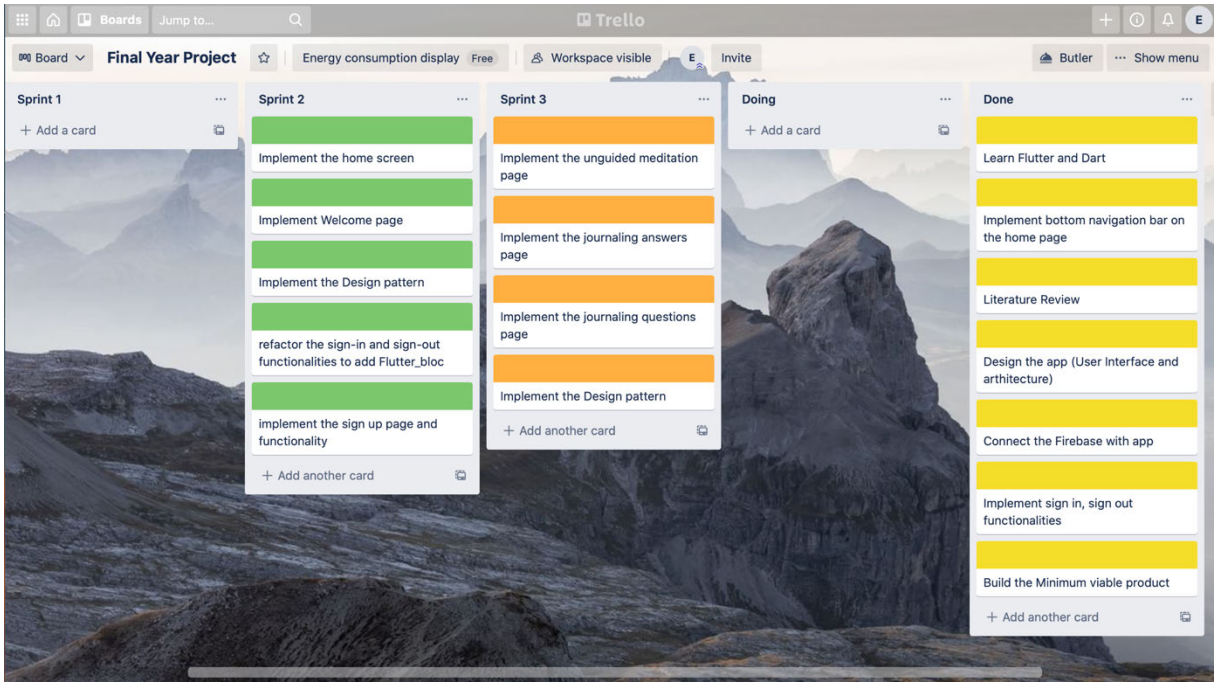


Minimum Viable Product

Moreover, the Gantt cart and Trello board updated according to the progress



Gantt chart- sprint 1



Trello board- sprint 1

----------SPRINT 2----------

16/12/2020

Seminar

Duration: 20 minutes

In the seminar, we had one on one meetings. I was able to get feedback for the dissertation and demo the minimum viable product.

22/12/2020

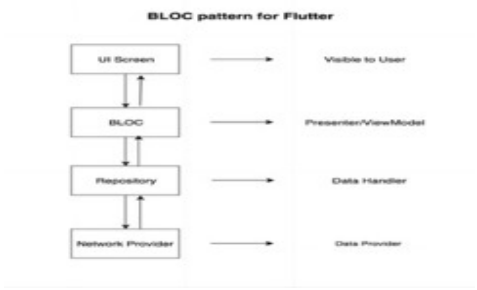
Independent Study

Duration: 50 minutes

Flutter is a great UI toolkit for Cross-platform development. However, the more I work with it, the more I see the downsides. For instance, the UI and logic part are on the same dart file. To have more maintainable and testable code, I have to use state management patterns like Bloc, Provider or Redux. The most used and recommended technique is called bloc.

```
RaisedButton(
  child: Text("Sign up"),
  color: Colors.white,
  onPressed: () {
    auth.createUserWithEmailAndPassword(
        email: _email, password: _password);
  }), // RaisedButton
```

UI and Backend together



Bloc design pattern to separate UI and backend

25/12/2020

Independent Study

Duration: 30 minutes

I found these resources after researching about Bloc pattern usage in Flutter

https://www.youtube.com/watch?v=tKET5s_Vu-c

https://medium.com/firebase-developers/dive-into-firebase-auth-on-flutter-email-and-link-sign-in-e51603eb08f8

https://medium.com/flutter-community/firebase-login-with-flutter-bloc-47455e6047b0

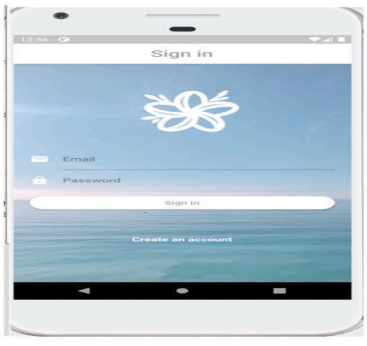
The more I research the bloc pattern, the more I question my development tool choice. Even though developing UI and basic operators are easy and hot reload functionality makes the development faster, the bloc state management is confusing. It will be a real challenge for me.

02/01/2021

Independent Study

Duration: 40 minutes

I started implementing the Login page. After conducting a user prototype testing, I collect the recommendations and changed the colour scheme to blue, white, which made the texts more readable and illustrated the app close to the mindfulness theme.



Welcome page of the new prototype

The new design looks more professional and provides more readability. So far, I connected the app with the firebase and implemented the sign-up, sign in and sign out features. Moreover, I created the bottom navigation bar, which the users will use to navigate to other pages.

6/01/2021-7/01/2021

Independent Study

Duration: 5 hours

I implemented the welcome/login page UI. It is abstract and readable for people. The next step is to implement the Bloc design pattern to the login page and continue with the home page implementation.

8/01/2021-9/01/2021

Independent Study

Duration: 4 hours

I completed the implementation of the home page and the bottom navigation bar. Because the backend and frontend are in the same file, I have to set the logic apart from the UI with the Bloc pattern.

This article will be beneficial for me to get started with the implementation.
https://medium.com/@lovnicki.sandro/flutter-bottomnavigationbar-with-bloc-pattern-bba6f13d49f3

```dart
void main() => runApp(MyApp1());

/// This is the main application widget.
class MyApp1 extends StatelessWidget {...}

/// This is the stateful widget that the main application instantiates.
class MyStatefulWidget extends StatefulWidget {
  MyStatefulWidget({Key key}) : super(key: key);

  @override
  _MyStatefulWidgetState createState() => _MyStatefulWidgetState();
}

/// This is the private State class that goes with MyStatefulWidget.
class _MyStatefulWidgetState extends State<MyStatefulWidget> {
  final auth = FirebaseAuth.instance;

  int _selectedIndex = 0;
  static const TextStyle optionStyle =
      TextStyle(fontSize: 40, fontWeight: FontWeight.bold);
  static List<Widget> _widgetOptions = <Widget>[
    //HomeScreen(),//this class must hold the quotes with background

    Journaling(),
    Text(
      'Index 2: Meditation',
      style: optionStyle,
    ), // Text
    Text(
      'Index 3: Calender',
      style: optionStyle,
    ), // Text
    Text(
      'Index 4: Points',
      style: optionStyle,
    ), // Text
  ]; // <Widget>[]

  void _onItemTapped(int index) {
    setState(() {
      _selectedIndex = index;
    });
```

As shown in the file set state function is in the same file as the UI. The next step is to implement the Bloc design pattern to the home page and the login page.

12/01/2021-13/01/2021

Independent Study
Duration: 4 hours

I started implementing the bloc design pattern and created the states, events and bloc pages following https://medium.com/@lovnicki.sandro/flutter-bottomnavigationbar-with-bloc-pattern-bba6f13d49f3 tutorial. Moreover, I could implement the same functionality with a simpler code. I need to research to make the implementation of the design pattern simpler.

17/01/2021-19/01/2021

Independent Study
Duration: 8 hours

I started the implementation of the bloc pattern by creating the bloc directory and subfolders. First, the user Repository class implemented to connect the app to the firebase. By following https://medium.com/flutter-community/firebase-login-with-flutter-bloc-47455e6047b0 tutorial, I implemented the authentication state, authentication event, authentication bloc classes. The authentication files are responsible for identifying whether the user is already logged in or not. However, the bloc state management technique is high level. I find it quite challenging to understand it without any research.

20/01/2021

Independent Study
Duration: 2 hours

I implemented the user interface of the Register page

21/01/2021-24/01/2021

Independent Study
Duration: 10 hours

I implemented the bloc design pattern for the sign-in function by creating login state, login event and login bloc classes. These classes used and are beneficial for checking whether the user can signed-in, or there were some problems after submitting the Login form. Moreover, if the user didn't sign in, they cannot reach the home page. After the sign-in event

is successful, it notifies the authorization bloc to authenticate the user and navigate to the home page.

25/01/2021

Independent Study
Duration: 8 hours

I implemented the bloc design pattern for the register function by creating register state, register event and register bloc classes. With the same logic of the sign-in, Bloc Listener listens to the state changes and handles the changes by converting events to the state. The state management technique helps identify if the user was being signed up or not. Moreover, if the user didn't sign-up, they cannot reach the home page.

26/01/2021

Independent Study
Duration: 5 hours

I completed the sign-up, sign in and sign out pages with the design pattern. Now, I need to start working on the main pages. Today I found a tutorial to implement the bloc pattern with less code and files for bottom navigation. https://medium.com/@madhu.sambangi/flutter-bottom-navigation-bar-navigation-using-dart-streams-bloc-pattern-61ac0e916804 bloc pages send the chosen index to the stream builder and navigates to the corresponding pages. I learn new things by changing the architecture from the bloc to streams. While bloc is a great design pattern, it is not suitable for every situation. It is important to know different techniques and implement them in the code. Streams with bloc separated the UI from logic (the state management) without lots of different pages and codes like the classic bloc state management technique.

28/01/2021

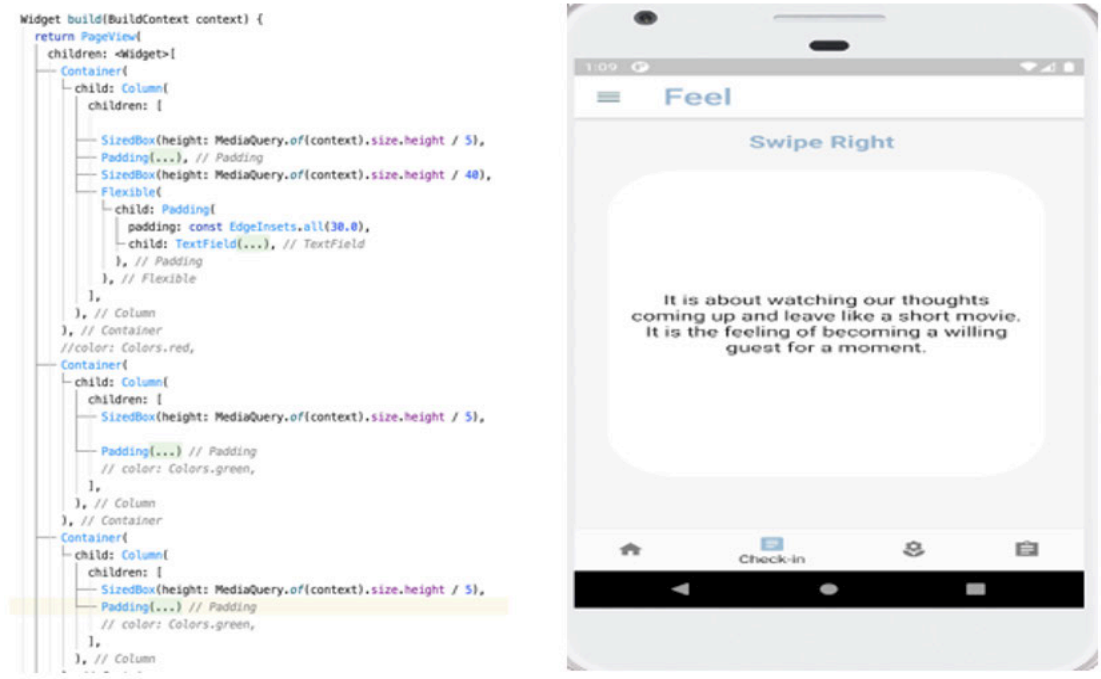
Independent Study
Duration: 3 hours

I implemented the home page today. I planned to add multiple pages to show various quotes about mindfulness. After thinking about it, I realized that having a list of quotes and randomly choosing them is better than implementing multiple pages with a single quote.

29/01/2021

Independent Study
Duration: 2 hours

I implemented the Introduction page where the user is going to learn about mindfulness. To implement the pages, I used vertically scrolling Page View. This approach allowed me to implement different text and input boxes without creating various pages, files or widgets. Now the next step is to implement the unguided meditation page. The only concern I have is the usability of the swiping effect



The mindfulness page and the pageview element

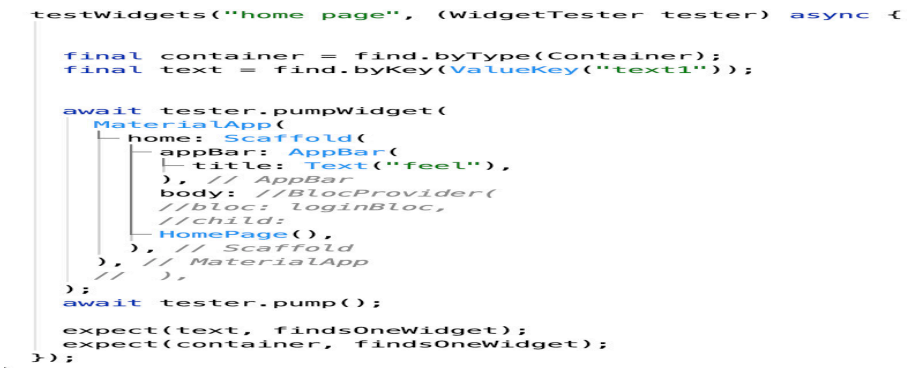
30/01/2021

Independent Study

Duration: 2,5 hours

Today, I focused on the implementation of the widget tests. The implemented pages sign-up, home page and mindfulness tests built after planning the test scenarios. One of the problems faced during the tests is finding the classes on the bottom of the widget tree. In order to solve this issue, multiple tests added to the test class.

Widget test of the mindfulness class



```
testWidgets("home page", (WidgetTester tester) async {

  final container = find.byType(Container);
  final text = find.byKey(ValueKey("text1"));

  await tester.pumpWidget(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text("feel"),
        ), // AppBar
        body: //BlocProvider(
        //bloc: loginBloc,
        //child:
        HomePage(),
      ), // Scaffold
    ), // MaterialApp
    //  ),
  );
  await tester.pump();

  expect(text, findsOneWidget);
  expect(container, findsOneWidget);
});
```

Widget test of the homepage class

----------SPRINT 3----------

-

01/02/2021- 03/02/2021

Independent Study

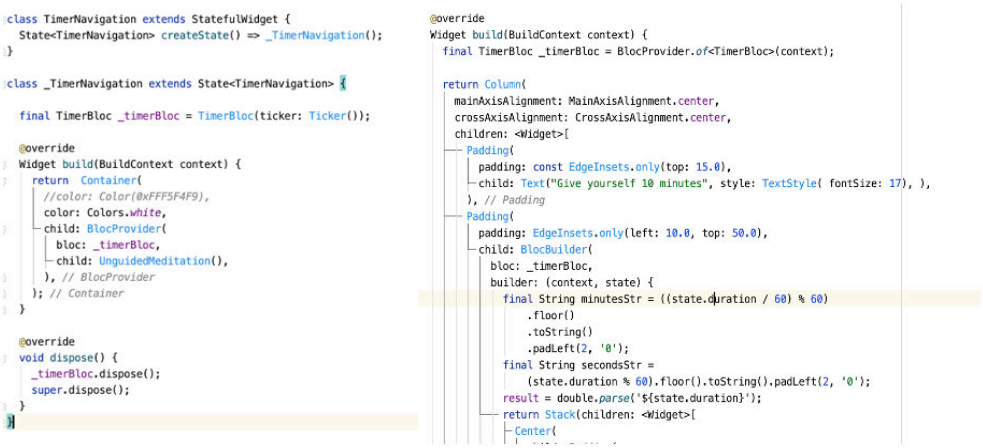
Duration: 10 hours

I have implemented the unguided meditation page in three days. Creating state, event and bloc classes was straightforward, and I had no difficulty implementing them. However, calling bloc pattern from the UI was challenging. From the sign-in page, I knew that I had to have the Bloc provider method first and call the class with the bloc builder method. However, every time I applied this logic, I get an error saying bloc provider called with a context doesn't include Timer Bloc and Unimplemented method error. These errors were confusing, and I couldn't find the problem to create the solution. Finally, I looked at resources to find out about the error and see how other people implement the timer clock on their Flutter project.

BlocProvider.of called with a different context error

I realized my mistake after the research. I was trying to get the bloc context from the previous class. Since it didn't contain Timer Bloc, it was giving me an error. Instead, I had to create an instance of it by calling the source Tick class. Moreover, in the next class, I had the receive the context.

After making this mistake, I feel more comfortable with the stage management technique and Flutter.



The solution to the problem

06/02/2021

Independent Study

Duration: 1 hour

I implemented an alarm for unguided meditation to help user notice when meditation is started and ended. I used the flutter audio players package to add this functionality to the app.

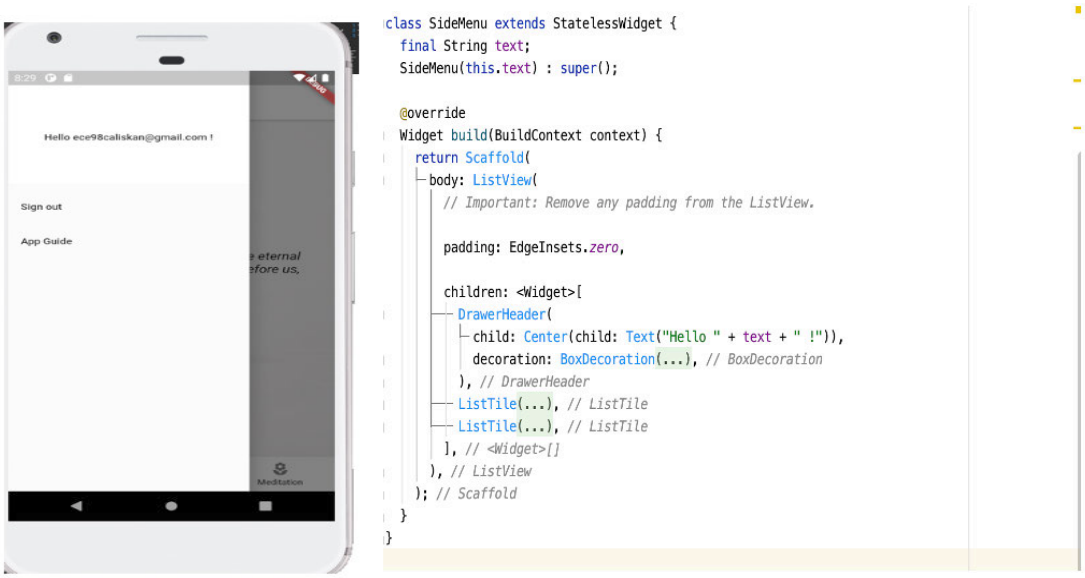
Source of the sound => https://orangefreesounds.com/shining-sound-effect/

08/02/2021

Independent Study

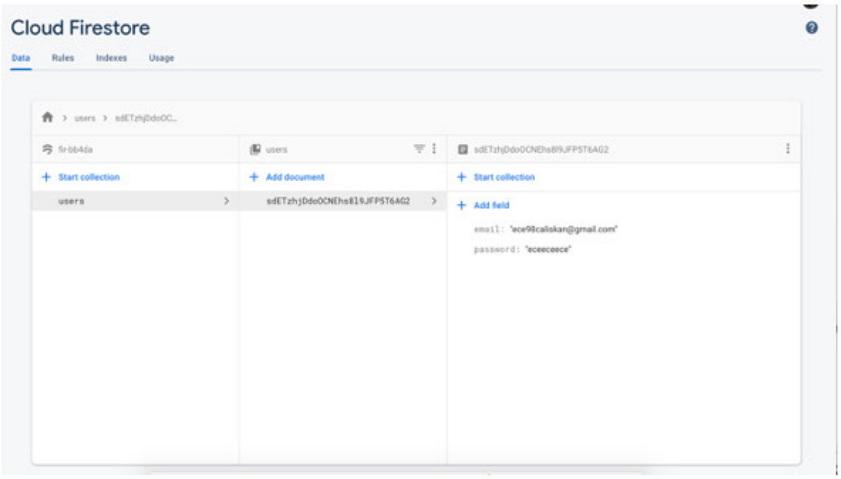
Duration: 1 hour

Today, I implemented the side menu feature for the application using the ListView widget.



ListView widget illustration on the emulator and implementation on the source code

Moreover, I connected the app with the firestore cloud database to provide backend features. However, because the passwords are not encrypted, they are stored in the database as strings. I either have to remove the passwords from the database or encrypt the passwords and store the encrypted password with the keys.



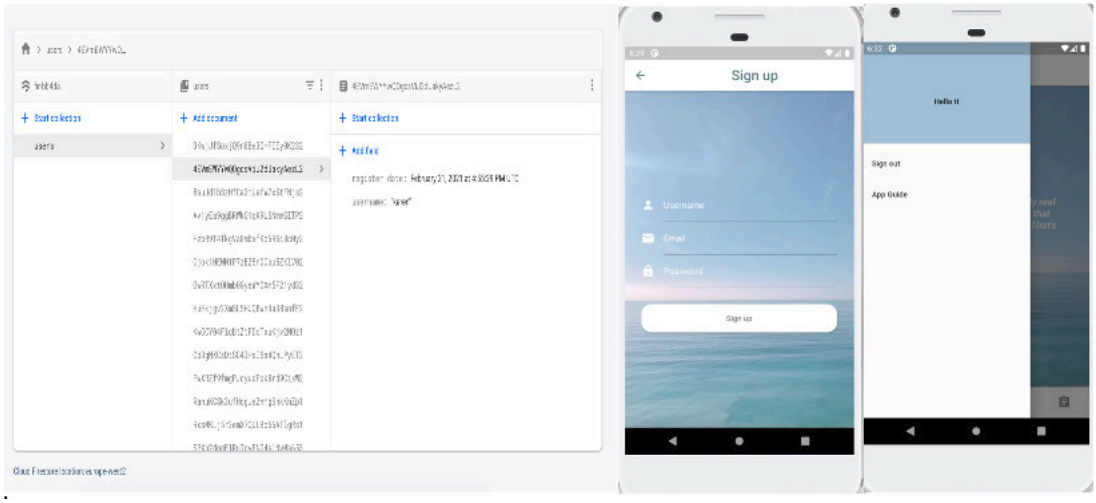
Firebase cloud firestore

12/02/2021

Independent Study

Duration: 1 hour

I implemented a username field to identify the users. The users are going to set the username when creating their accounts. Moreover, instead of including the email of the user on the side menu, I include the usernames. Moreover, the passwords and the emails of the users will not be on the database.



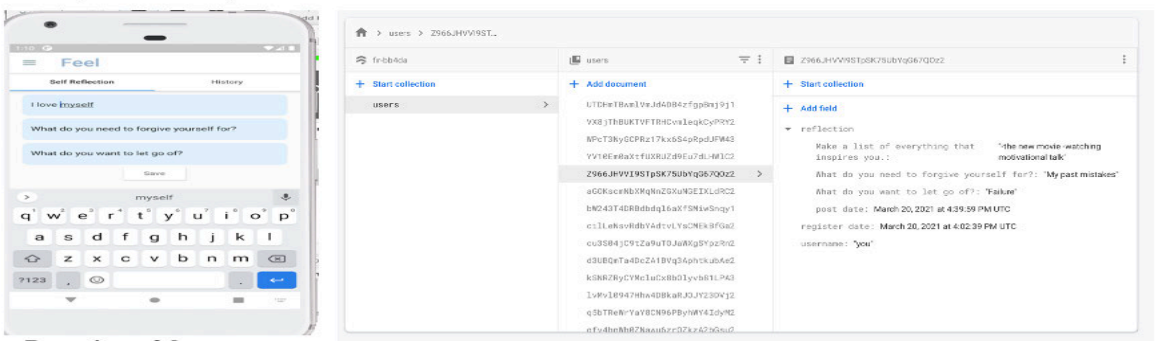
Adding username field to the application and storing it on the database

13/02/2021

Independent Study

Duration: 30 minutes

In order to improve the app, I decided to create a self-reflection page. The page will contain a tab menu, and the first page will ask user questions to make them reflect on themselves. Moreover, the second tab called history will contain the users answers to the previous questions.



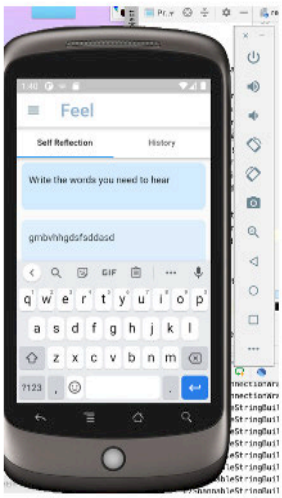
Implementing the self-reflection questions page

19/02/2021

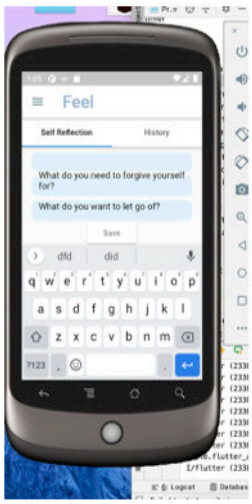
Independent Study

Duration: 2 hours

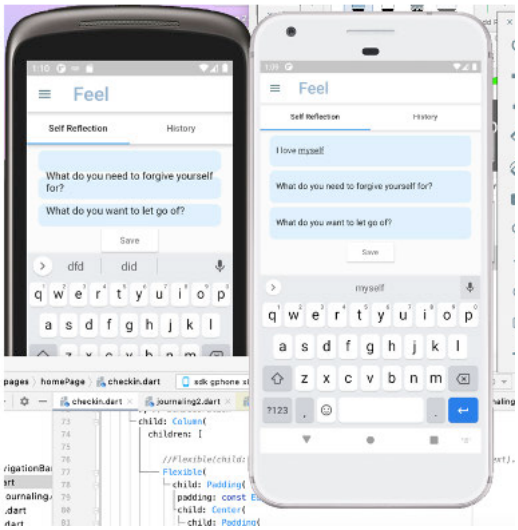
I implemented the self-reflection questions. However, when I test the application with a smaller screen, the scrolling-down effect was not working. To deal with this issue, I decided to use the Flex widget, which will make the text fields responsive. Moreover, the responsive and smaller test fields enable the user to see the save button. However, using smaller text fields prevent the users to see the texts on the text field.



Before



After



The responsiveness problems

20/02/2021

Independent Study

Duration: 1 hour

To fix the issue with the Flex widget, I did some research. Moreover, I decided to use a ListView instead of the Flex and container. The ListView will allow the user to scroll down even though the keyboard is opened.

List View element for the scroll-down effect

24/02/2021

Independent Study

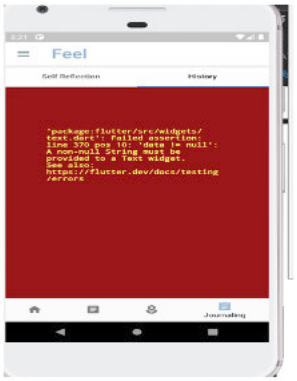
Duration: 5 hours

I built the bloc pattern on the self-reflection page by implementing state, bloc and event classes. However, while I was connecting these with the UI, I faced with build function returned null error. After researching on the internet, I returned Container rather than leaving it empty.

error

The source code to fix the error.

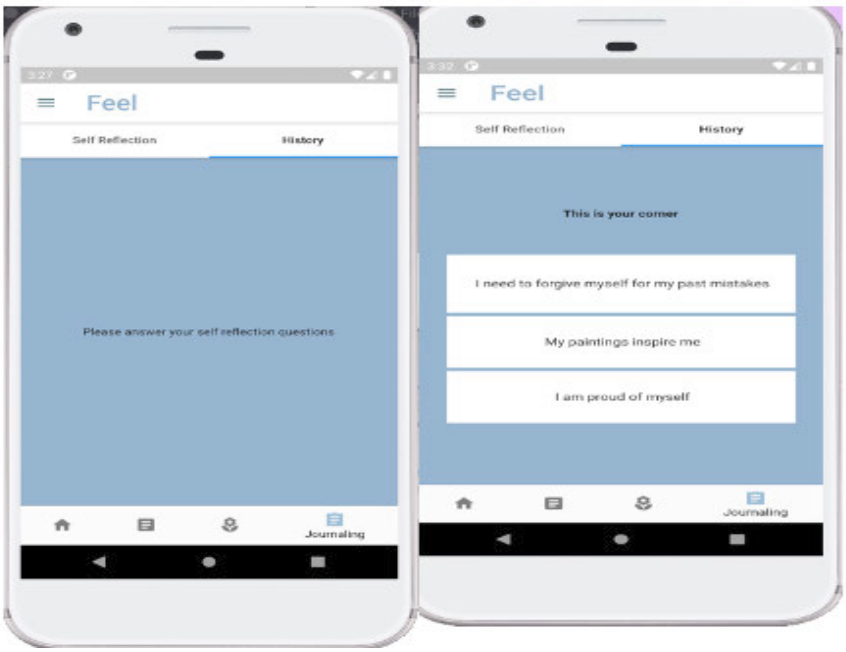
I started to implement the tab menu and the history of the answers of the users by using the state, event and bloc classes. However, for the people whose database values are null, the system gave a null error.




```
child: BlocBuilder(
  bloc: _reflectionbloc,
  builder: (BuildContext context, ReflectionState state) {
   BlocProvider.of<ReflectionBloc>(context).dispatch(Retrievedw());

   if (state is Retrieved && state.text1.elementAt(0) == null) {
    return Container(
     child: Center(
      child: Text("Please answer your self reflection questions"),
      ), // Center
    ); // Container
   } else if (state is Submitted) {
    return Container();
   } else if (state is Retrieved && state.text1.isNotEmpty) {
    String a = state.text1.elementAt(0).toString();
    String b = a.replaceAll(new RegExp("{"), '');
    String c = b.replaceAll(new RegExp(":"), ',');
    final split = c.split(',');
```

data====null error

I fixed this issue by not only checking the state. But by checking whether the data is null or not. If it is null, the user will see a message.



The message for the users if they haven't. After users save the message saved answers to the database

I built the bloc pattern for the self-reflection page by implementing state, bloc and event classes. However, while I was connecting bloc with the UI, I faced data returned null error for the newly registered users. I fixed this issue by checking the state of the database and whether the data is null. If it is null, the system will warn the user to save the answers first.

26/02/2021

Independent Study

Duration: 2 hours

Today, testing the app on smaller screens, I realized that the mindfulness part wasn't responsive for smaller screens.

```
Flexible(
  child: SizedBox(
      height: MediaQuery.of(context).size.height / 5)), // SizedBox, Flexible
Text(
  "Hold an object now like a book or mug and be present with it, as if it is the first time yo
  "Recognize its weight, colors, shape and texture with a curiosity",
  style: TextStyle(fontSize: 18),
  textAlign: TextAlign.center,
), // Text
SizedBox(height: MediaQuery.of(context).size.height / 50),
//  Flexible(
//  child: Padding(
```

Previous code

To make the app more responsive, I removed the sized box and implemented the centre widget. Moreover, today I found three users to conduct prototype testing. Moreover, I send them the introduction and testing scenario documents that contained the link to the prototype.

```
child: Column(
  children: [
    Flexible(
      child: Padding(
        padding: const EdgeInsets.only(right: 8.0, left: 8.0),
        child: Center(
          child: Padding(
            padding: const EdgeInsets.only(right: 8.0, left: 8.0),
            child: Text(
              "Hold an object now like a book or mug and be present with it, as if it is the first time you are seeing it."
              "Recognize its weight, colors, shape and texture with a curiosity",
              style: TextStyle(
                  fontSize: 18,
                  color: Colors.black54,
                  fontWeight: FontWeight.bold), // TextStyle
              textAlign: TextAlign.center,
            ), // Text
```

New code with centre widget

01/03/2021

Independent Study

Duration: 3 hours

Today, I started to build the main titles of the report and research about testing in Flutter. Furthermore, today I received feedback from the users and created a table to store their feedback anonymously.

03/03/2021

Independent Study

Duration: 1 hour

Today, I started preparing the UML diagram and collect the sketches, prototypes for the app.

7/03/2021

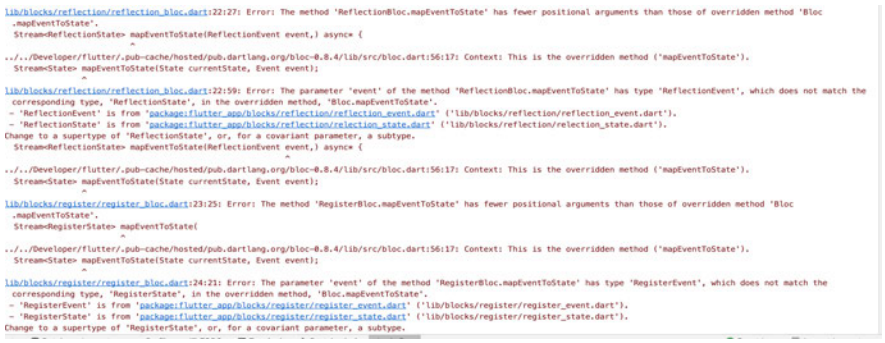
Independent Study
Duration: 1 hour

One of the comments from the previous reports were about my objectives. Today, I worked to improve them and found resources to implement unit testing on the app.

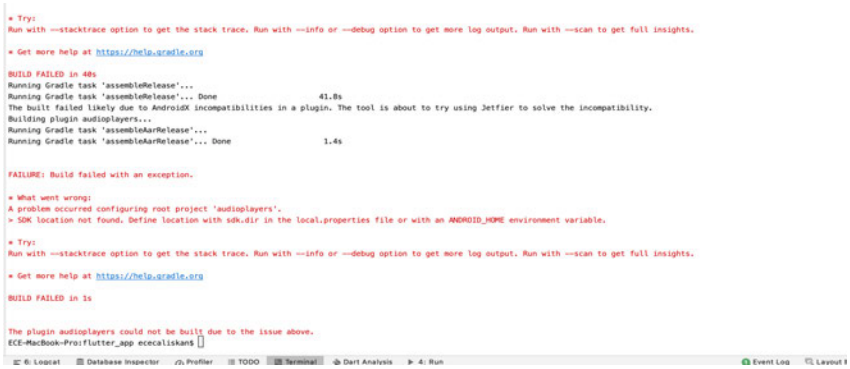
18/03/2021

Independent Study
Duration: 6 hours

Today I wanted to build the API to experience it. In the build stage, I had a problem with the version of the audio player. I started to research how to fix it. To make the versions compatible, I had to upgrade the flutter and dart to the last version, which will cause issues with bloc patterns. While I was researching, I come across a command “flutter upgrade --force”. I run the command and face a disaster.



errors



errors

The build and the emulator weren’t working anymore. I panicked and researched about how to fix all of the issues but couldn’t find a way. I came to the point where I was about to

give up. After some time, I came to my computer and downgraded the flutter version using the command “flutter downgrade”. Now the error was less, but Bloc listener and bloc classes were not working. I tried to change the version of them, but it didn’t work. So, I decided to approach the problem in a different way, which turned out to be a better approach compared to the previous one.

```
@override
Widget build(BuildContext context) {
  return BlocListener(
  bloc: _registerBloc,
  builder: (BuildContext context, RegisterState state) {

    if (state.isSuccess) {
      print(_usernameController.text);
     BlocProvider.of<AuthenticationBloc>(context).dispatch(LoggedIn());
      SchedulerBinding.instance.addPostFrameCallback((_) {
     Navigator.push(
        context,
        MaterialPageRoute(
          builder: (BuildContext context) => Authorize(email: _usernameController.text,),
        ), // MaterialPageRoute
      );
    }
      );}

      return Container(
        height:  MediaQuery.of(context).size.height,

        decoration: BoxDecoration(
          image: DecorationImage(
              image: AssetImage("5.JPG"), fit: BoxFit.cover), // DecorationImage
        ), // BoxDecoration
        child: Form(
          child: Padding(

            padding: const EdgeInsets.only(right: 20.0, left: 20.0, ),
            child: ListView(
              children: <Widget>[
                SizedBox(
                  height: MediaQuery.of(context).size.height/5
```

Errors on the files

```
return BlocBuilder<LoginEvent, LoginState>(
  bloc: _loginBloc,
  builder: (
    BuildContext context,
    LoginState state,
    ) {
    if (state.isSuccess) {
    BlocProvider.of<AuthenticationBloc>(
        context
    )
        .dispatch(LoggedIn());
    SchedulerBinding.instance.addPostFrameCallback((_) {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (BuildContext context) => Authorize(email: _emailController.text),
      ), // MaterialPageRoute
    );
  });}
  return Container(

    decoration: BoxDecoration(
      image: DecorationImage(
          image: AssetImage("5.JPG"), fit: BoxFit.cover), // DecorationImage
    ), // BoxDecoration
    child: Form(
      child: Padding(
        padding: const EdgeInsets.all(20.0),
        child: ListView(
```

Errors on the files

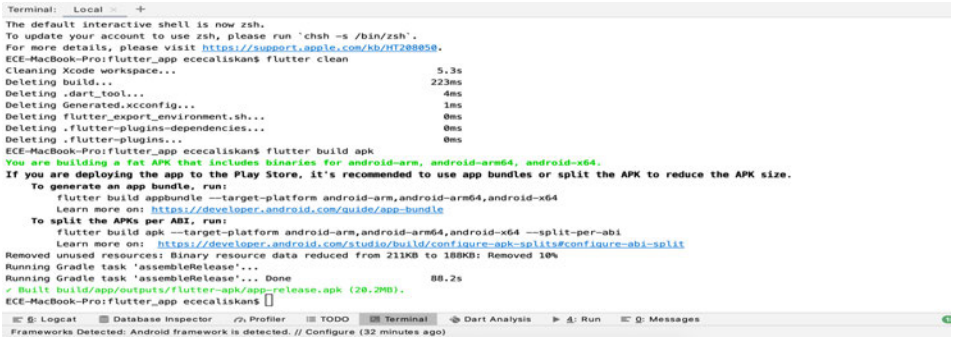
Instead of using Bloc listener and get an error, I now use Bloc builder, which led me to use less code to do the same operation.

```
@override
AuthenticationState get initialState => Uninitialized();

@override
Stream<AuthenticationState> mapEventToState(
    AuthenticationEvent event,
    ) async* {
  if (event is AppStarted) {
    yield* _mapAppStartedToState();
  } else if (event is LoggedIn) {
    print("fine1");
    yield* _mapLoggedInToState();
  } else if (event is LoggedOut) {
    yield* _mapLoggedOutToState();
  }
}
```

```dart
@override
Stream<AuthenticationState> mapEventToState(
   currentstate, AuthenticationEvent event,
    ) async* {
  if (event is AppStarted) {
    yield* _mapAppStartedToState();
  } else if (event is LoggedIn) {
    print("fine1");
    yield* _mapLoggedInToState();
  } else if (event is LoggedOut) {
    yield* _mapLoggedOutToState();
  }
}
```

Errors on the files

Next, instead of the previous version, the current state keyword is required.



Fixing errors and building the apk

After making these changes, everything worked smoothly, and I learned the biggest lesson of my life. I shouldn't have run the command without knowing all of its details. However, since Flutter is still a new Framework, there is not a lot of information on the internet.

20/03/2021

Independent Study

Duration: 2 hours

Because of the errors that I received yesterday, I decided to test the app on iOS, which ended up very bad. It was because of the pods' folder of iOS. After doing the research, this solution from GitHub worked.

```
[!] Oh no, an error occurred.

Search for existing GitHub issues similar to yours:
https://github.com/CocoaPods/CocoaPods/search?q=%5BBug%5D+Failed+to+find+known+source+wi
22trunk%22&type=Issues

If none exists, create a ticket, with the template displayed above, on:
https://github.com/CocoaPods/CocoaPods/issues/new

Be sure to first read the contributing guide for details on how to properly submit a tic
https://github.com/CocoaPods/CocoaPods/blob/master/CONTRIBUTING.md

Don't forget to anonymize any private data!

Looking for related issues on cocoapods/cocoapods...
 - Pod install returning error
   https://github.com/CocoaPods/CocoaPods/issues/2944 [closed] [3 comments]
   15 Dec 2014
```

Pods error for the IOS

```
flutter clean
rm -Rf ios/Pods
rm -Rf ios/.symlinks
rm -Rf ios/Flutter/Flutter.framework
rm -Rf ios/Flutter/Flutter.podspec
```

Commands to fix the pods error

21/03/2021

Independent Study

Duration: 1 hour

Today, I started to implement the Unit testing for the authentication part, which will take care of the state. Currently, only two state tests implemented to test the initial and log out states.

Failed two tests

First, both of the tests have failed due to the implementation problem in the authentication state class. After making changes, everything was working.

```
class Uninitialized extends AuthenticationState {
  @override
  String toString() => 'Uninitialized';

  @override
  // TODO: implement props
  List<Object> get props => throw UnimplementedError();
}
```

The previous Uninitialized class

```
class Uninitialized extends AuthenticationState {}
```

Changed Uninitialized class

After the change, the initial state test passed, and the logout test failed. To pass both of them, forgotten state keyword added to the unit test

```
00:02 +1 -1: Some tests failed.
ECE-MacBook-Pro:flutter_app ececaliskan$ flutter test
00:02 +1 -1: LoggedOut emits [uninitialized, loading, unauthenticated] when token is deleted [E]
  Expected: should do the following in order:
            • emit an event that <Instance of 'Uninitialized'>
            • emit an event that <Instance of 'Unauthenticated'>
    Actual: <Instance of 'AuthenticationBloc'>
     Which: was not a Stream or a StreamQueue

  package:test_api                                  expectLater
  package:flutter_test/src/widget_tester.dart 468:10  expectLater
  athentication_test.dart 52:11                     main.<fn>.<fn>
```

Only one of the tests failed.

```dart
group('LoggedOut', () {
  test(
      'emits [uninitialized, loading, unauthenticated] when token is deleted',
          () {
        final expectedResponse = [
          Uninitialized(),
          Unauthenticated(),
        ];

        expectLater(
          authenticationBloc,
          emitsInOrder(expectedResponse),
        );

        authenticationBloc.dispatch(LoggedOut());
      });
});
```

The previous unit test without state keyword

```dart
group('LoggedOut', () {
  test(
      'emits [uninitialized, loading, unauthenticated] when token is deleted',
          () {
        final expectedResponse = [
          Uninitialized(),
          Unauthenticated(),
        ];

        expectLater(
          authenticationBloc.state,
          emitsInOrder(expectedResponse),
        );

        authenticationBloc.dispatch(LoggedOut());
      });
});
```

New unit test with state keyword

```
00:02 +2: All tests passed!
```

The result of the two tests

22/03/2021

Independent Study

Duration 3 hours

Today, I started testing the User repository class, which has function such as sign-in, sign-up and sign-out.

```
class MockUser extends Mock implements User{}
final MockUser _mockUser = MockUser();
class MockFirebaseAuth extends Mock implements FirebaseAuth{
  @override
  Stream<User> authStateChanges() {
    return Stream.fromIterable([_mockUser]);
  }
}

Future<void> main() async {
  setupFirebaseAuthMocks();
  await Firebase.initializeApp();

  final MockFirebaseAuth mockFirebaseAuth = MockFirebaseAuth();
  final UserRepository user = UserRepository(firebaseAuth: mockFirebaseAuth);

// ALL MY TESTS WOULD BE HERE
test("emit occurs", () async {
  expectLater(user.userTest, emitsInOrder([_mockUser]));

});
}
```

Initializing mocks

Using the setUpFirebaseAuthMocks, I initialized the firebase mock to the tests.

```
test("create an account", () async {
  when (mockFirebaseAuth.createUserWithEmailAndPassword(email: "ece@gmail.com", password: "ece123456")
  ).thenAnswer((realInvocation)=> null);

  expect(await user.signUpWithEmail(email: "ece@gmail.com", password: "ece123456"), "Success");
});
}
```

```
ECE-MacBook-Pro:flutter_app ececaliskan$ flutter test
00:02 +3: All tests passed!
ECE-MacBook-Pro:flutter_app ececaliskan$
```

Create account test passed

The first test created to sign the user up to the app, and the tests passed. Next, I applied a unit test to the sign-in function.

```
test("sign in", () async {
  when(mockFirebaseAuth.signInWithEmailAndPassword(
      email: "ece@gmail.com", password: "ece123456"))
      .thenAnswer((realInvocation) => null);

  expect(await user.signInWithEmail(email: "ece@gmail.com", password: "ece123456"), "Success");
  authenticationBloc.dispatch(LoggedIn());
});
```

```
ECE-MacBook-Pro:flutter_app ececaliskan$ flutter test
00:02 +4: All tests passed!
ECE-MacBook-Pro:flutter_app ececaliskan$
```

Sign-in test passed

Finally, I tested the sign-out function, and the result was successful.

```
test("sign out", () async {
  when(mockFirebaseAuth.signOut())
      .thenAnswer((realInvocation) => null);

  expect(await user.signOut(), "Success");
  authenticationBloc.dispatch(LoggedOut());


});
```

```
ECE-MacBook-Pro:flutter_app ececaliskan$ flutter test
00:10 +6: All tests passed!
ECE-MacBook-Pro:flutter_app ececaliskan$
```

Sign-out test passed

23/03/2021

Independent Study

Duration 3 hours

I continued writing the development and implementation part of the dissertation Moreover, I reached how to implement the integrational testing on the Flutter framework.

https://flutter.dev/docs/cookbook/testing/integration/introduction

28/03/2021

Independent Study

Duration 2,5 hours

Today I started the implementation of the integrational testing and implemented the login and sign-up tests. Moreover, finished the development and implementation section of the dissertation.

```
test('integration tester sign into the app with details ttt@ttt.ttt and ttttttttt111', () async {
      //enter details
       await driver.tap(addemailfield);
       await driver.enterText("ttt@ttt.ttt");
       await driver.tap(addpasswordfield);
       await driver.enterText("ttttttttt111");
       //submit
       await driver.tap(addloginbutton);
       //expect homepage
       await driver.waitFor(find.text("Feel"));

   });

   test('integration tester sign up to the app with details ccc, cccc@cccc.cccc, ccccccc111', () async {
  //  sign-up details.
       await driver.tap(addusernamefield1);
       await driver.enterText("ccc");
       await driver.tap(addemailfield1);
       await driver.enterText("cccc@cccc.cccc");
       await driver.tap(addpasswordfield1);
       await driver.enterText("ccccccc111");
       //submit the details
       await driver.tap(addregisterbutton1);
     //expect home page
       await driver.waitFor(find.text("Feel"));
});
```

Integrational test for sign-in and sign-up

04/04/2021

Independent Study

Duration 1 hour

I wanted to start the prototype testing the application with the new design. Contacted the volunteers through slack and sent the introduction file. Moreover, I sent them another document that contained the scenario and the question.

08/04/2021

Independent Study
Duration 1 hour

Today I received feedback from the users and created a table from their feedback to keep their sensitive information private.

Name	Things user liked on the application	Things user disliked on the application	Is the scenario achieved
User A	The overall design of the application	The navigation buttons were ambiguous	yes
User B	The colours and information about mindfulness	I didn't understand how to swipe the page on the mindfulness section	yes
User C	Overall, a great application.	I couldn't understand which page I was clicking by looking at the icons	yes

User feedbacks

12/04/2021-15/04/2021

Independent Study
Duration 12 hours

During this period, I implemented the rest of the pages to the integrational testing. Moreover, finished the testing and analysis part of the dissertation.

16/04/2021

Independent Study
Duration 7 hours

Today, I created the apk to start the internal testing on the google play store. I built the application with the workplace named com.example.flutter_app, and the google play store returned an error. To deploy the application to the play store, I changed the workplaces of the project, firebase and android to com.ece.flutter_app. Moreover, I created the key to sign the

application and built the app bundle again. I added the user emails to the tester section and released the test.

18/04/2021

Independent Study
Duration 2 hours

I received the user feedback and created a table from their feedback to keep their personal information private.

Name	Things user liked on the application	Things user disliked on the application	Is the scenario achieved
User A	I liked the circular process and the timer.	More colours can be used to attract more users	yes
User B	The design looks good, and I liked the quotes on the home page	The journaling page can be improved.	yes
User C	Overall, a good application.	I sent my answers. However, the history section didn't show the answers	no

User feedbacks

A bug found by one of the internal testers. However, after I tested it with my device and the emulators, the problem didn't occur. I assumed it is because of their internet connection.

20/04/2021

Independent Study
Duration 2 hours

I finished the critical evolution part of the dissertation and starting to write the conclusion.

25/04/2021

Independent Study
Duration 1 hour

I realised the application for the review to the Google Play Store.

Appendix B

Project Proposal

Flutter Mindfulness Application

Introduction, Aim, Motivation and Background

The project aims to help people protect and improve their wellbeing, especially during challenging times like pandemic through a responsive application developed by using the Flutter development kit, Dart language and SQLite databases. More and more people required to self-isolate due to coronavirus. This situation left most of us feeling lonesome and unmotivated to work remotely or achieve daily tasks productively. By using this app, people will improve their concentration, presence and productivity. Moreover, people will be more peaceful, which will enhance their work and social lives and interactions with others. While there are many applications around mindfulness theme such as Calm, Headspace and Breathe [5], I couldn't find an application which is responsive enough to provide features such as adjustable font size or colour schemes for people with colour blindness. Most of the apps are not free and doesn't have a user guide.

According to the Health Foundation, 69% of adults reported feeling worried about the effect of COVID-19. Moreover, 63% of the adults worried about the future and 56% feeling stressed and anxious due to financial loss and loss of coping mechanisms such as exercise, friends, family members and hobbies. [4] My motivation behind developing a mindfulness app is to be able to reach and help a large number of people who suffers from these problems and create them a safe space to breathe and reflect on their emotions mindfully. Moreover, my passion and enthusiasm on the subject will help me to think about innovations to help people get through these challenging times.

Last summer, I built myself an android to-do list app using Kotlin language and Android studio out of interest, and I enjoyed it. In this final year project, I wanted to challenge myself by learning Flutter and Dart language to build a native cross-platform app with an SQLite database. Even though Flutter is a new software development kit, and not very commonly used as React Native, it has a variety of advantages such as providing faster performance and hot reload function. [3] Another reason for me choosing this project to show my passion for learning new technologies in the industry and improve myself. As Flutter and dart seem very

unfamiliar to me, I want to use all of my abilities to push my limits and create an application which will be beneficial not only for me but for everyone at the end.

Customers

The mobile app is for teenagers, young-adult and adults who want to improve their ability to be mindful and focus on the moment with the calming music, sounds and journaling section which they can acknowledge their emotions, mood and gratitude.

Objectives

Objective 1 Design the pages of the app using Fluid UI.
Objective 2 Implement the user login and sign-in page and store the user to the database.
Objective 3 Implement the welcome page with background music and navigation menu.
Objective 4 Implement the journaling and mindful breathing meditation pages. Objective 5 Do unit, widget and integrational level testing using Flutter features and test classes.

Objective 6 Do the user testing by using Google Play Store and App Store beta testing option.
Objective 7 Deploy the application to the App Store and Goggle Play Store using XCode and Android studio.

Optional Objectives

Objective 1 Migrate the mobile app to the web app using Flutter extension.

Development Requirements

I will develop the app using the MacBook Pro laptop (Retina, 15-inch, Mid 2015. It has 2,2 GHz Quad-Core Intel Core i7 CPU processor and runs the macOS Catalina version 10.15.7 which is the latest release of macOS operating system.

The primary development environment is going to be Android Studio version 4.0 with Flutter and Dart plugins installed from the marketplace. It is an Integrated Development Environment which is used for developing Android apps [2].

However, for testing and deployment purposes, XCode 12.0.01 version will be used, which is a software development environment to develop software for macOS, IOS products. [1] IOS emulator that comes with Flutter and Open android Emulator Android Accelerated Oreo and Pixel XL API 30 will be used for testing purposes in the development stage. Emulators create virtual devices on the screen. It is easier to test the program while developing to see the functionality and the design.

SQLite database engine latest version 3.33.0 will be used to store users. It is one of the most commonly used databases and implements a serverless, transactional and self-contained database engine. [6]

Methodology

The agile methodology will be used as the methodology of the project. Agile methodology is incremental and iterative, more open for changes during the development period according to customer requirements, and its iterative model will be beneficial for meeting the needs of the customer. Since I don't have lots of experience in Flutter environment as well as the dart language, it will allow me to make changes easier when something goes wrong compared to other development methodologies such as Waterfall.

PXP (Personal Extreme Programming) methodology would not be beneficial for the project since I won't be able to do the pair programming process and the designing of the application is as important as the software functionality, the design shouldn't be simple. Kanban methodology is open for changes more than SCRUM. However, it does not contain iterations which will make it hard for me to do the demo to supervisors and get their ideas. By using SCRUM with Agile methodology, I will be able to divide the work into sprints and at the end of each one, receive feedback to produce a high-quality end product. Moreover, sprint planning, prioritising the tasks and sprint retrospectives will be useful for me on analysing the objectives and thinking about what worked well or didn't go as planned.

I will plan the project using the Gantt Chart, Product and Sprint backlogs, decide on the requirements using user stories, use cases and UI prototyping, which will give me the idea of the design of the app as well as the further functionality additions. The project is a mobile-

native cross-platform app. However, as an optional objective, it might become a multi-channelled app. Dart programming language will be used as a primary language of Flutter using Android Studio and XCode development environments with the help of the SQLite database. I will use different Android and IOS emulators to test the functionality and UI design in the development stage. Moreover, I will do the unit, widget and integrational testing in each sprint. At the end of the third sprint, the app will be realised to Google Play Store and Apple Store for beta testing for users, and then I will finally publish it.

Gantt Chart

Objective 1
Design the pages of the app using Fluid UI.
02/11/2020-13/11/2020

Objective 2
Implement the user login and sign-in page and store the user to the database.
16/11/2020-11/12/2020

Objective 3
Implement the welcome page with background music and navigation menu.
21/12/2020-22/01/2021

Objective 4
Implement the journaling and mindful breathing meditation pages.
01/02/2021-05/03/2021

Objective 5
Do unit, widget and integrational level testing using Flutter features and testing classes.
14/12/2020-18/12/2020
11/01/2021-18/01/2021
22/02/2021-05/03/2021

Objective 6
Do the user testing by using Google Play Store and App Store beta testing option.
08/03/2021-19/03/2021

Objective 7
Deploy the application to the App Store and Goggle Play Store using XCode and Android studio.
22/03/2021-26/03/2021

Assignments
ISM – **07/11/2020**
VR3D – **11/01/2021**
Intro, Literature, Methodology – **18/11/2020**
Project Dissertation – **21/11/2020 – 05/05/2021**