National College of Ireland

Software Project

Software Development

2020/2021

Aaron Hynes

X17742385

X17742385@student.ncirl.ie

WORKAROUND

Technical Report

Contents

Executive Summary

This document will report on the specifications on my 4th Year Software project, WorkAround, the dynamic workout app, targeting the casual gym goer. This report will cover several aspects of the software project, including a discussion about the necessity of WorkAround and its objectives, identifying its list of functional requirements, the technology and architecture behind the application, and a run through of the Graphical User Interface as well as some of more notable code running in the background. The report will discuss unit, widget and integration testing and application evaluation and how the application could be improved upon further development and research.

1.0 Introduction

1.1. Background

The market is saturated by applications that make your sessions revolve around their routines, they will be designed around specific workouts or types of exercise, some require subscriptions and others require large premiums, while sacrificing quality on free services.

I wanted to make an application that does the opposite, WorkAround tailors a workout to the time that you have and provides you with a workout that other applications might not be capable of giving. WorkAround targets the office worker who wants to exercise during a 15 minute break, or the work-from-home parent with no real time to dedicate to a 60 minute or more workout routine every other day after wrestling the kids away from their video call meetings.

This application will adjust a workout dynamically, so you can always go at your own pace and feel like you had a balanced workout. For example, if the user took a bit longer than usual then the application may adjust the rest of the workout by dynamically removing sets so that the user can a full workout instead of leaving out exercises. If the user is found to have regained the time they lost, it will also add the sets that were removed, back.

Research on the market suggests that there are apps that provide shorter workouts, but you are stuck doing the routines they provide and even still, it can take longer without the approach WorkAround takes. All of this is why I believe that this product will be innovative.

1.2. Aims

WorkAround aims to create a workout application for the average person and the casual gym-goer. It facilitates those who may not be able to dedicate an hour a day, or even every other day by providing them with the capacity to define the length of their own custom workouts. It contains exercises for different mobility levels and provides exercises depending on the equipment a user may have, in fact, WorkAround provides over 120 unique exercises for the user to choose from and has the capacity to easily add more.

A number of objectives were outlined in WorkAround's project proposal, however time constraints or the evolution of the project over time changed the implementation of some the objectives somewhat.

Through WorkAround, I aim to provide a user with the capacity to:

- Generate a workout based on a muscle or muscle group, equipment available and amount of time the user has.
- Create, edit and delete their own workouts, customising the reps, sets and weight.
- Perform a workout with one of their workouts.
- Persist user data across multiple devices, such as their account information, workouts and workout history.
- Dynamically adjust a workout routine based on the users pace and remaining time.
- Ensure the user will be able to complete their workout within their designated timeframe.
- View past workouts.
- See additional information about exercises, such as instructions and demonstrations through gifs.
- Provide exercises for mobility reduced persons.
- Create a WorkAround account.

One could take a look at this list of objectives and compare it to the list proposed initially in the project proposal, there are a few differences, mostly due to how the implementation of functionality evolved the application over time. For instance, originally one of the objectives was to allow the user to enable cloud backups, however, this evolved into the data persistence objective as the implementation of many of the functions provided inherent persistence of the User objects, meaning they did not have to manually opt into the feature.

1.3. Technology

In this section of the report, I will outline several technologies that were integral to the development of WorkAround.

Flutter

Flutter is a free, open-source toolkit by Google that lets you build natively compiled programs for desktop, mobile and the web, all in a single codebase. It features abilities that provide faster development, such as its Hot Reload and Restart features. Hot Reloading an app restarts your application in a fraction of a second, while keeping state on the device, so you can make changes to UI or functionality and see it update on the device in just a moment.

As of 2019, Flutter was in the top 11 software repos on GitHub stars and has been used to develop thousands of applications across multiple app stores. Flutter works by using objects called Widgets, these widgets are made up of several parts making up structural, styling and layout elements, multiple Widgets are combined to make up an applications GUI. Flutter provides these, however, a developer can make their own Widgets through the combination of other Widgets. Flutter offers reactive style views without the need for a JavaScript bridge by implementing Dart. In the end, Flutter is a development tool that transcends platform, allowing you to build native apps for iOS and Android and provides excellent performance through the Dart programming language. (ConciseSoftware, 2019)

Dart

Dart is a high-level language developed by Google and is used as an alternative to JavaScript, it is easy to learn and provides high performance over apps created in JavaScript. What gives Dart its edge though, is its Ahead of Time(AOT) and Just in Time(JIT) compilations. Dart code can be converted into native machine code through AOT, making apps run natively across platforms. (CodeCarbon, 2020) Dart is a null safe language, usually when a system receives a response that is null or blank, it will crash, however, Dart provides Null Safety, so your app will not crash on a null response. (Oda, 2021)

Stacked

Stacked is an MVVM (Model-View-ViewModel) architecture that is used to provide common functionalities and code principles to make application development easier and more maintainable. The major components of Stacked are the View, ViewModel and Services. I will dive more into the Stacked architecture in the Design & Architecture section of this documentation.

Provider

Provider is a state manager helper that provides the capacity to pass state along a Widget tree in a Flutter application and automatically rebuilds the UI when changes have been detected. (Sande, 2020)

Technology-wise, Provider is a wrapper around one of Flutter's Widgets called InheritedWidget, making them more reusable and easier to implement.

Firebase

Firebase is Google's Backend-as-a-Service for mobile and web applications. It consists of many services that let you build and manage your applications. The modules that we are concerned with in particular are the services that WorkAround uses, Cloud Firestore and Firebase Authentication. Cloud Firestore is a real-time NoSQL database designed for enterprise use and provides a console that can be used to view data in the applications Firestore. Firebase Authentication is an authentication service specifically designed for applications using Firebase and provides the capacity to authenticate an applications users for logging in and resetting passwords etc. (AltexSoft, 2019)

1.4. Structure

This report documents the development of my project, WorkAround. Section 2 will define the functional requirements of the application, as well as discussion on the topics of the design and architecture of the software. I will demonstrate notable code in the Implementation section and display the most notable parts of the Graphical User Interface. Then I will discuss the testing techniques implemented and how I evaluated the system over the course of the project, ending with a conclusion on what I learned about the application during development and outlining several aspects the application can be researched and developed further, should the project be pursued after.

The appendices will contain required information such as my reflective journals and a copy of my project proposal for reference against this documentation.

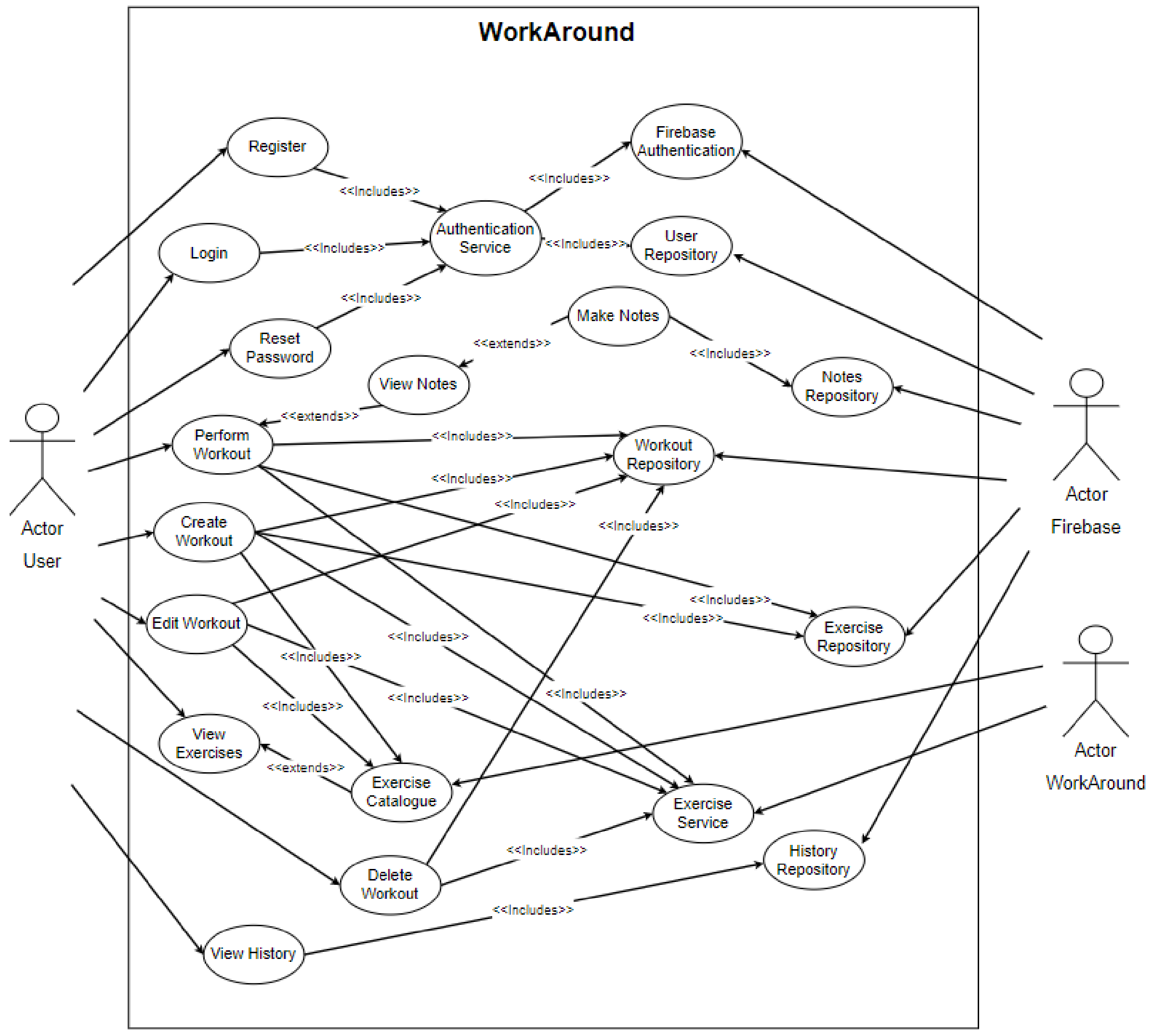
2.0 System

2.1. Requirements

These requirements will be achievable by most users, users should not take more than a few uses of the application to become familiar with the processes as it will aim to be as simple as possible.

2.1.1. Functional Requirements

2.1.1.1. Use Case Diagram



2.1.1.2. Requirements

2.1.1.2.1. Requirement 1: Perform Workout

2.1.1.2.1.1. Description & Priority

This is the main functional requirement of the application and what gives it its innovation and competitive edge. It is the reason the application is being made and thus is the foremost priority.

2.1.1.2.1.2. Use Case

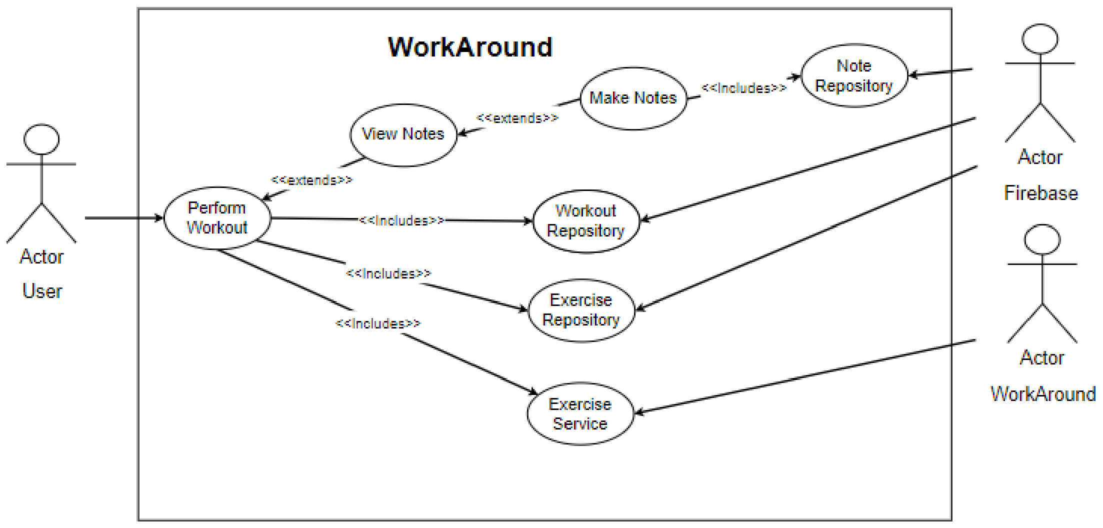
Scope

The scope of this use case is to allow the user to perform a workout that dynamically adjusts depending on how long the user is taking.

Description

The key point of the application is the dynamic adjusting of the workout when the user performs a workout, at certain points in the workout, the workout will adjust itself depending on the remaining time the customer has left, which they defined at the start of the workout.

Use Case Diagram



Flow Description

Precondition

The user must have an account and be logged in and they must have entered the time they have to complete the workout.

Activation

This use case starts when a user presses the “Workout” button on their home page.

Main flow

1. The user presses the workout button from their list of workouts on the home page.
2. The user inputs the time they have, in minutes, to complete the workout.
3. The system generates a workout depending on the time they input.
4. The user will perform their workout, tapping on the buttons that represent their sets when they complete an exercise.
5. The system readjusts the workout depending on how much time is left until they are done.

Alternate flow

Due to the design of the application, there is no alternate way of accessing the workout page as the user should not be able to access this page without a workout. If the user has no workouts, there will simply be no button to press.

Exceptional flow

E1 : <User cancels the workout>

1. At step 2 of the main flow, the user does not input a time to complete the workout and cancels.
2. The system returns to the home page.

E2 : <User finishes workout early>

1. At step 4 of the main flow, the user exits the workout.
2. The system returns to the home page.

Termination

This use case ends when the user completes a workout or hits either of the exception flows.

Post condition

User is returned to their home page.

2.1.1.2.2. Requirement 2: Create Workout

2.1.1.2.2.1. Description & Priority

This use case describes the creation of new workouts. Its priority is second to only the perform workout requirement. An argument can be made that since a workout needs to be created to be performed, that this requirement should have a higher priority, but the perform workout is the purpose of the application.

2.1.1.2.2.2. Use Case

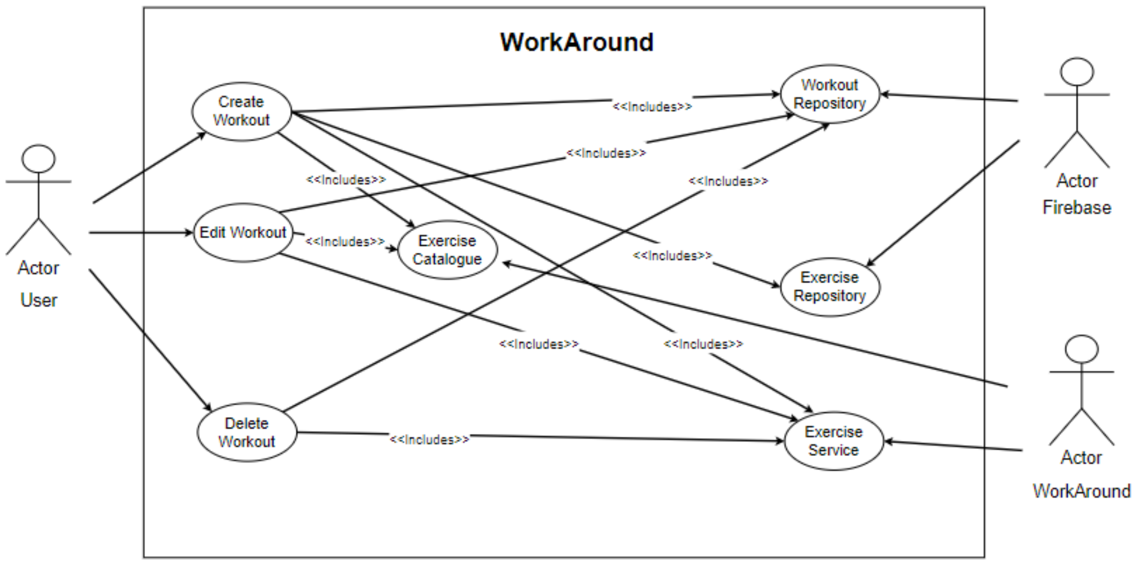
Scope

The scope of this use case is to let the user create a workout.

Description

This use case describes the functionality for the user to be able to create a new workout.

Use Case Diagram



Flow Description

Precondition

User is logged in and on their home page.

Activation

This use case begins when the user taps the “Create workout” button.

Main flow

1. User taps the “create workout” button on their home page.
2. User names their workout.
3. User taps the “Add exercise” button.
4. User scrolls through a list of exercises.
5. User taps an exercise.
6. User inputs the number of sets, reps and weight they wish to do.
7. User repeats step 2 of the main flow until they have created their ideal workout.
8. User taps the “Done” button.
9. Workout is added to the user’s list of workouts.

Alternate flow

The main flow is the only way to create a workout by design.

Exceptional flow

E1 : <User cancels creation of workout>

1. At step 3 of the main flow, the user cancels the process of creating a new workout instead of adding an exercise.
2. System returns to the home page.

Termination

The use case terminates once a workout has been created.

Post condition

A workout is created, and the user is returned to the home page.

2.1.1.2.3. Requirement 3: Edit Workout

2.1.1.2.3.1. Description & Priority

This use case describes the editing of current workouts. Its priority comes after the "Create workout" requirement, as it cannot be performed unless there is a workout already created.

2.1.1.2.3.2. Use Case

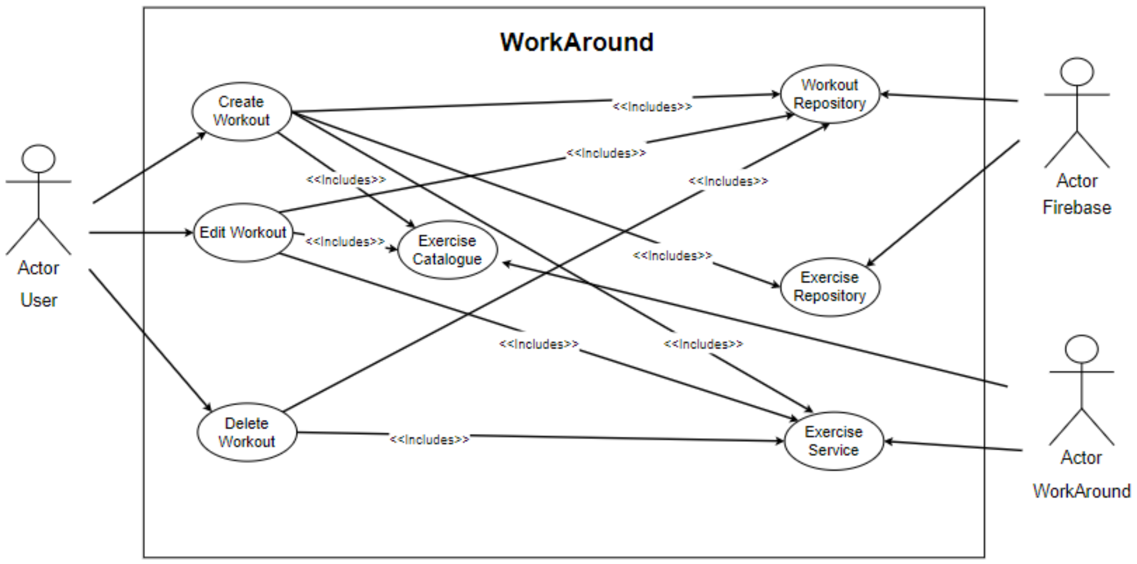
Scope

The scope of this use case is to let the user edit a workout.

Description

This use case describes the functionality for the user to be able to edit a current workout.

Use Case Diagram



Flow Description

Precondition

User is logged in and on their home page and have at least one workout created on their workout list.

Activation

This use case begins when the user taps the “Edit” button on a workout tile.

Main flow

1. User taps the “EDIT” button on their home page on the workout tile.
2. User can add an exercise by tapping the “Add Exercise” button.
3. User scrolls through list of exercises.
4. User taps on an exercise
5. User inputs their desired sets, reps and weight.
6. User confirms edit of workout.
7. User repeats step 2 until they have created their ideal workout.
8. User taps the “Done” button.
9. System returns to home page.

Alternate flow

A1 : <User edits a current exercise>
1. At step 2 of the main flow, the user taps the “Edit” button.
2. User enters their desired sets, reps and weight
3. User taps the “Done” button.
4. User returns to step 2 of the main flow.

A2 : <User deletes a current exercise>
1. At step 2 of the main flow, the user taps the “Delete” button.
2. User returns to step 2 of the main flow.

Exceptional flow

E1 : <User cancels editing of workout>
1. At step 2 of the main flow, the user cancels the process of editing a workout.
2. System returns to the home page.

Termination

The use case terminates once a workout has been edited.

Post condition

A workout is edited, and the user is returned to the home page.

2.1.1.2.4. Requirement 4: Delete Workout

2.1.1.2.4.1. Description & Priority

This use case describes the deleting of workouts. Its priority is equal to the "Edit Workout" requirement but comes after "Create Workout" as a workout must exist.

2.1.1.2.4.2. Use Case

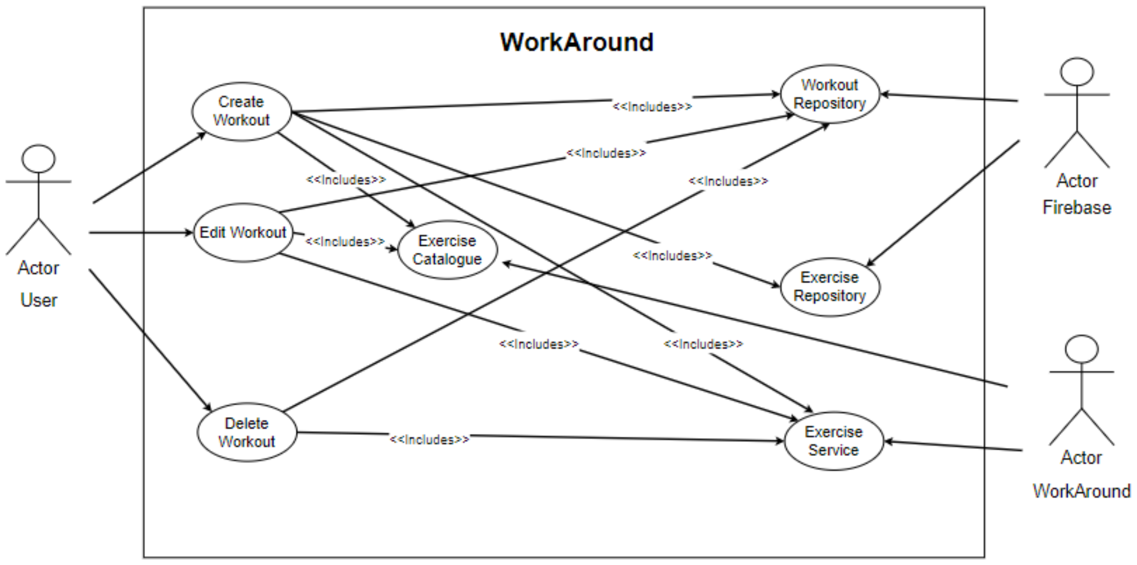
Scope

The scope of this use case is to let the user delete a workout.

Description

This use case describes the functionality for the user to be able to delete a workout.

Use Case Diagram



Flow Description

Precondition

User is logged in and on their home page and have at least one workout created on their workout list.

Activation

This use case begins when the user taps the "Delete" button on a workout tile.

Main flow

1. User taps the "DELETE" button on their home page on the workout tile.
2. User is prompted to confirm deletion of their workout.
3. Workout is removed to the user's list of workouts.

Exceptional flow

E1 : <User cancels deletion of workout>

1. At step 2 of the main flow, the user cancels the process of deleting a workout.
2. System returns to the home page. **Termination**

The use case terminates once a workout has been deleted.

Post condition

Workout is removed from the list of workouts and user is returned to the home page.

2.1.1.2.5. Requirement 5: Exercise Catalogue

2.1.1.2.5.1. Description & Priority

The app will hold a readable collection of exercises, these will be the exercise the user can create workouts with, and the collection will also provide additional information on how to perform the exercise.

2.1.1.2.5.2. Use Case

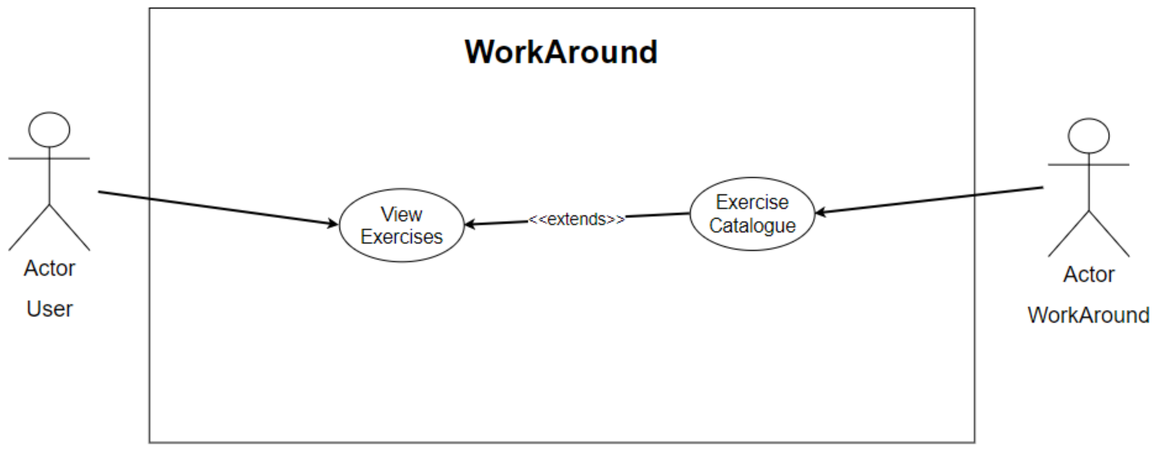
Scope

Allow the user to browse the collection of exercises stored in the app.

Description

This use case describes the process of the user looking through the app's collection of exercises for additional information.

Use Case Diagram



Flow Description

Precondition

User is logged in and on the home page.

Activation

This use case activates when user decides to browse the app's collection of exercises.

Main flow

1. The user taps the "Exercise Catalogue" button on the home page.
2. The user browses the collection of exercises by scrolling through the list of exercises.
3. The user taps an exercise.
4. System displays extended information about the exercise.

Alternate flow

A1 : <Accessing catalogue through drawer>

1. The user taps the drawer button on the top left of the home page.
2. The user taps "Exercises" on the drawer.
3. User proceeds from step 2 of the main flow.

A2 : <Searching exercises>

1. From step 2 of the main flow, user can input text in the search field.
2. User proceeds from step 2 of the main flow.

A2 : <Searching exercises part II>

1. From step 2 of the main flow, user can tap a button that filters the list of exercises by equipment or muscle group.
2. User proceeds from step 2 of the main flow.

Exceptional flow

E1 : <User cancels search>

1. From step 2 or 3 of the main flow, the user returns to the previous page.

Termination

The use case terminates when the user leaves the exercise catalogue.

Post condition

The app displays additional exercise information.

2.1.1.2.6. Requirement 6: View Workout History

2.1.1.2.6.1. Description & Priority

The app will hold a collection of workouts the user has completed with all the information provided through the workout, such as the sets they completed.

2.1.1.2.6.2. Use Case

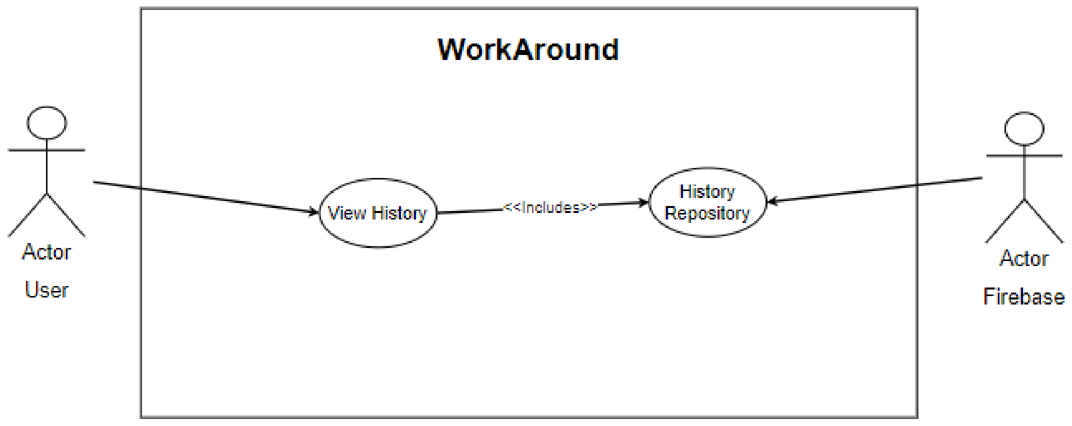
Scope

Allow the user to browse their workout history.

Description

This use case describes the process of the user looking through their history of workouts.

Use Case Diagram



Flow Description

Precondition

User is logged in and on the home page and has completed at least one workout.

Activation

This use case activates when user decides to browse their workout history.

Main flow

1. The user taps the “Workout History” button on the home page.
2. The user browses the list of workouts in their list.
3. The user taps on a workout.
4. System displays extended information about the workout, such as the sets they completed.

Alternate flow

A1 : <Browsing workout specific exercises>

1. The user taps the drawer button on the top left of the home page.
2. The user taps “History” on the drawer.
3. User proceeds from step 2 of the main flow.

Exceptional flow

E1 : <User cancels search>

1. From step 2 or 3 of the main flow, the user returns to the previous page.

Termination

The use case terminates when the returns from the Workout History view.

Post condition

The app returns to the home page.

2.1.1.2.7. Requirement 7: Log in

2.1.1.2.7.1. Description & Priority

This requirement describes the log in functionality of the app; it has a low priority due to the importance of the rest of the requirements.

2.1.1.2.7.2. Use Case

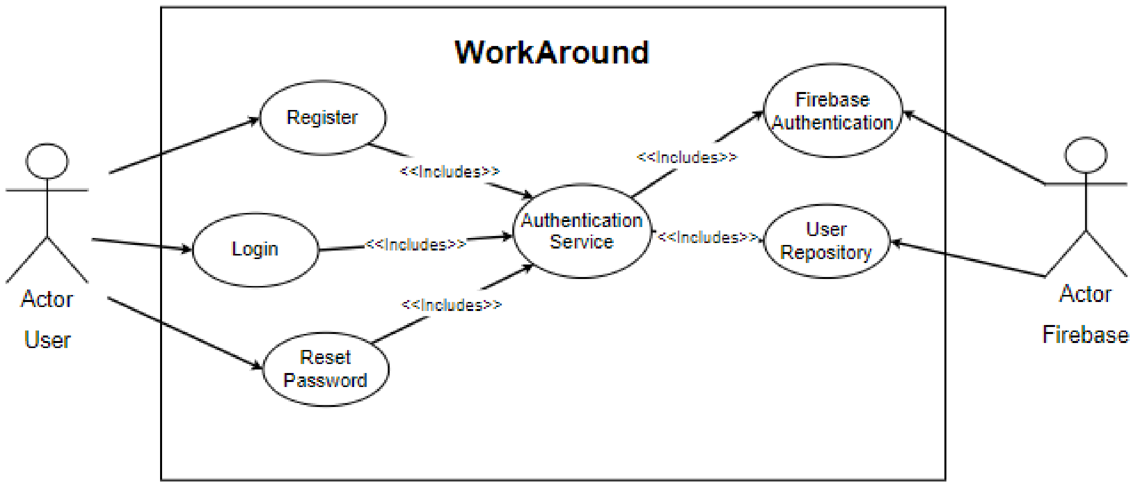
Scope

The scope of this use case is to allow an existing user to log in.

Description

This use case describes the process of logging into the app as an existing user.

Use Case Diagram



Flow Description

Precondition

The app is booted and on the welcome screen and has an existing account.

Activation

This use case starts when a user boots the application and taps the Log in button

Main flow

1. The user taps the login button on the welcome screen.
2. The user inputs their login details.
3. User confirms details and logs in.
4. User is sent to the home page.

Exceptional flow

E1 : <Incorrect Details>

1. At step 2 of the main flow, the user inputs incorrect login details.
2. The system will respond, telling them their details are incorrect.
3. Use case restarts from step 2 of the main flow.

E2: <No account detected>

1. At step 2 of the main flow, the system finds there is no matching email or account name.
2. System will prompt the user to register an account.

Termination

The use case terminates when the user successfully logs in.

Post condition

The user is logged in and sent to the home page.

2.1.1.2.8. Requirement 8: Register

2.1.1.2.8.1. Description & Priority

This requirement describes the Registration functionality of the app; it is equal in priority to the Log In requirement due to the importance of the rest of the requirements.

2.1.1.2.8.2. Use Case

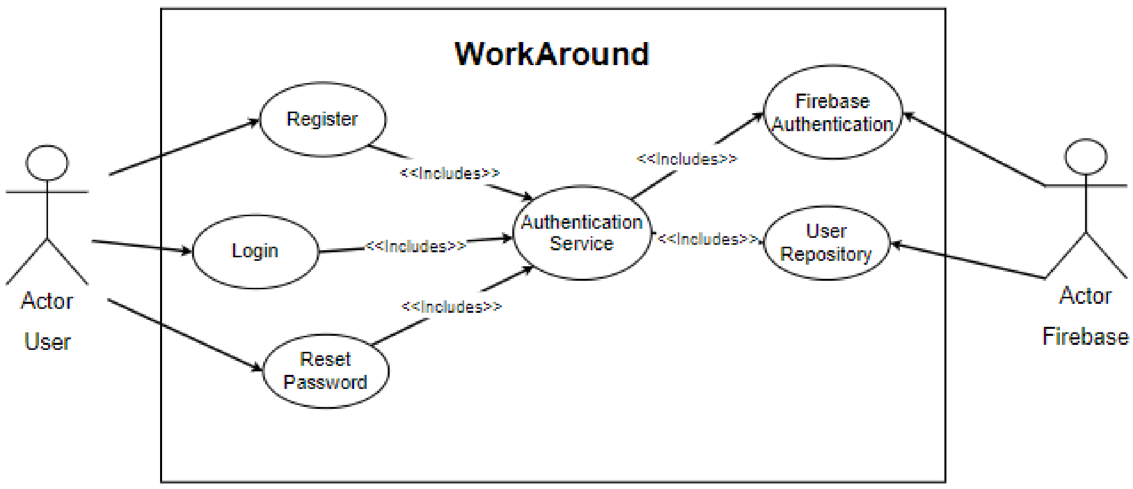
Scope

The scope of this use case is to allow a user to register an account.

Description

This use case describes the process of creating an account as a new user.

Use Case Diagram



Flow Description

Precondition

The app is booted and on the welcome screen and does not have an existing account.

Activation

This use case starts when a user boots the application and taps the Register button.

Main flow

1. The user taps the register button on the welcome screen.
2. The user inputs their details.
3. User confirms details and registers an account.
4. User is sent to home page.

Exceptional flow

E1 : <Account with details already exists>

1. At step 2 of the main flow, the user inputs existing login details.
2. The system will respond, telling them their details match an existing account.
3. Use case restarts from step 2 of the main flow.

Termination

The use case terminates when the user successfully registers and account.

Post condition

The user is registered, logged in and sent to the home page.

2.1.1.2.9. Requirement 9: Reset Password

2.1.1.2.9.1. Description & Priority

This requirement describes the Reset Password functionality of the application. If the user forgot their password, they may request to change it. Its priority is equal to that of the login and register requirements.

2.1.1.2.9.2. Use Case

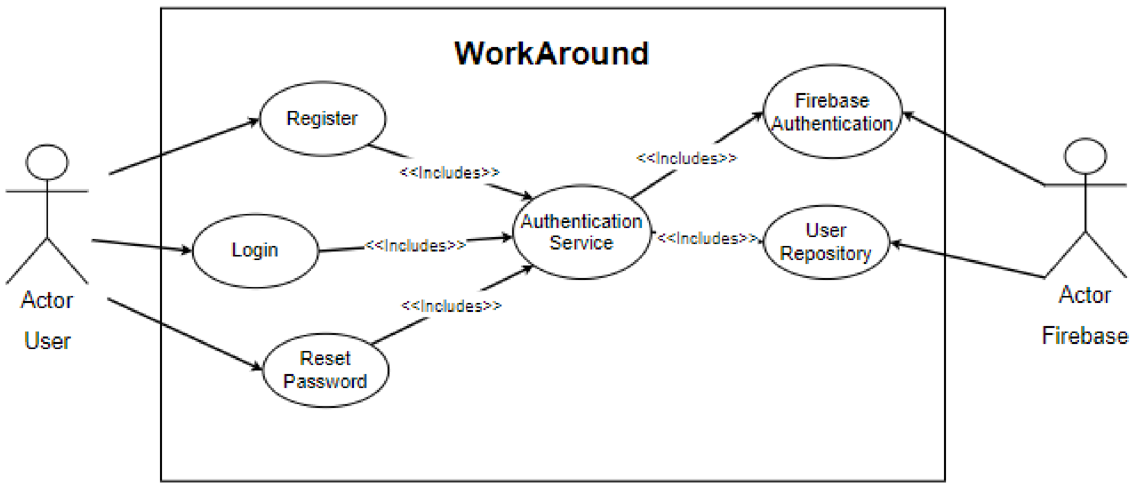
Scope

The scope of this use case is to allow a user to reset their password.

Description

This use case describes the process of a user's password.

Use Case Diagram



Flow Description

Precondition

The app is booted and on the login screen, but the user forgot their password.

Activation

This use case starts when the user taps the "Forgot password?" link on the login page.

Main flow

1. User taps the "Forgot password?" button.
2. User enters their email.
3. User taps the submit button to receive an email to reset their password.

Exceptional flow

E1 : <User remembers password>

1. User taps the “Remembered password?” button, redirecting them to the log in page.

Termination

The use case terminates when the user successfully resets their password.

Post condition

The user’s password is reset.

2.1.2. Data Requirements

The application takes in a name, email and password. The name is used to create an account for the app, and the email and password is used for Firebase Authentication.

WorkAround uses gifs and text from a third party website called MuscleWiki. All information taken is in accordance with the MuscleWiki copyright, which states that “The .gif files, text, videos which can be found on youtube.com and muscle information can be used with the MuscleWiki branding and with links back to musclewiki.com.”. This content can be used free of charge and without prior consent so long as the MuscleWiki branding is intact and there are links back to the website. (MuscleWiki, 2021) This has been provided with every exercise.

2.1.3. User Requirements

Using the objectives outlined in the project proposal as a reference for plotting out a list of user requirements we can identify a number of requirements that the average user should expect.

These user requirements are:

- The user must be able to create a custom workout.
- The user must be able to edit and delete their custom workouts.
- The app must provide the user with the ability to generate a routine with the given workout.
- The app must provide the user with the ability to dynamically update a routine as the user performs the workout and that they complete the workout within the time they have allotted.
- The user must have access to additional information on an exercise, such as instructions or video demonstrations.
- The app must provide the user with a large list of exercises to create a workout with, including exercises to cater to persons with limited mobility.

These requirements are what were kept in mind when building the functional requirements of the application.

2.1.4. Environmental Requirements

This app will have no effect on any external environment, however, due to the nature of Firebase, the user must have an active internet connection to be able to use the application to its fullest potential.

2.1.5. Usability Requirements

The aim of the application was to provide the user with an easy to learn UI, the UI should not take long to become accustomed to, and after an absence, the user could still use the application.

2.2. Design & Architecture

The application is designed via the Stacked architecture. Following this architecture, the application is split up into three main components, these are Views, View Models and Services. The Views provide the user with the user interface while the View Models handle the logic and manage state. Services are classes used to wrap a set of features into a single entity called a "service". For example, in the case of WorkAround, its main service is the *ExerciseService* in the file *exercise_service.dart* and its responsible for a lot of the logic in the application. To get the most out of the Stacked architecture, there were a number of principles that needed to be followed during development of WorkAround. These include but are not limited to:

- Views should not use services directly; this is the responsibility of the View Model.
- Views contain as little logic as possible and preferably none at all, this is again the responsibility of the View Model.
- View Models should not know about other View Models

(FilledStacks, 2021)

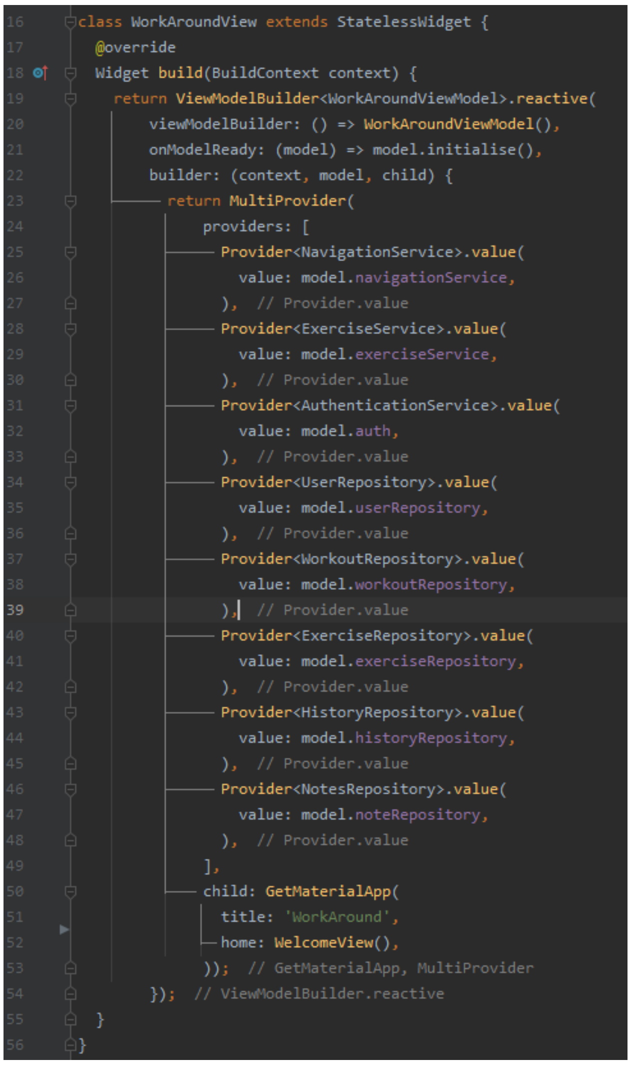
When we build a View, we associate it with a class that manages its state and logic, this is the View Model, the View should remain free of logic and should only be concerned with displaying the UI. The View Model is built with the classes it needs to perform its functionality, passed in by Provider. These View Models inherit from different classes depending on the requirements of each View, but I will go into far more detail on what all of this looks like in the Implementation section of this documentation. WorkAround consists of many Views and View Models, as well as several Services that make up the backbone of its functionality. These Services can be further separated into a standard Service and a Repository. Where the standard Services are classes that wrap a set of functionalities for the application, Repositories act as an interface between the application and the Cloud Firestore database and contain all the functions for the Create, Read, Update and Delete operations of the database.

WorkAround defines a number of Models and caters to the persistence of these entities via the Firestore database, when a user creates an account, the account information is stored in Firestore, when they create a workout, it is stored in Firestore, this design provides the user with persistent data across multiple devices and speeds up loading times of certain functions of the application since the app does not get bogged down with storing most of its data locally.

The main difficulties of this application came from the series of functions that generate the dynamic workouts and designing and implementing the database. As the algorithm that generates dynamic workouts is the sum of a series of functions that interact with the database, it was tough at the beginning to imagine what the database would look like, and how the function can be designed around it.

2.3. Implementation

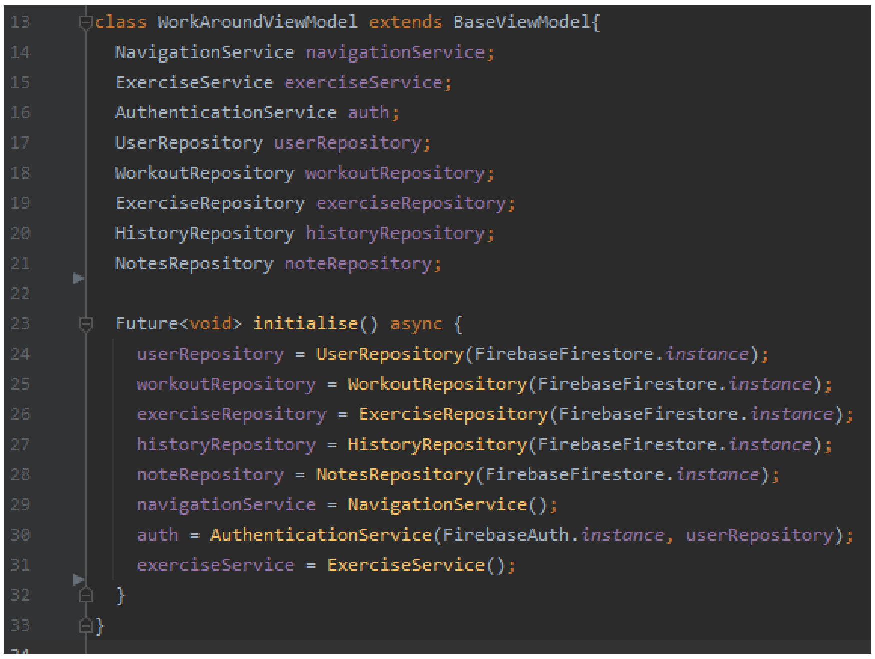
Figure 1. workaround_view.dart



```
16  class WorkAroundView extends StatelessWidget {
17    @override
18    Widget build(BuildContext context) {
19      return ViewModelBuilder<WorkAroundViewModel>.reactive(
20          viewModelBuilder: () => WorkAroundViewModel(),
21          onModelReady: (model) => model.initialise(),
22          builder: (context, model, child) {
23            return MultiProvider(
24                providers: [
25                  Provider<NavigationService>.value(
26                    value: model.navigationService,
27                  ), // Provider.value
28                  Provider<ExerciseService>.value(
29                    value: model.exerciseService,
30                  ), // Provider.value
31                  Provider<AuthenticationService>.value(
32                    value: model.auth,
33                  ), // Provider.value
34                  Provider<UserRepository>.value(
35                    value: model.userRepository,
36                  ), // Provider.value
37                  Provider<WorkoutRepository>.value(
38                    value: model.workoutRepository,
39                  ), // Provider.value
40                  Provider<ExerciseRepository>.value(
41                    value: model.exerciseRepository,
42                  ), // Provider.value
43                  Provider<HistoryRepository>.value(
44                    value: model.historyRepository,
45                  ), // Provider.value
46                  Provider<NotesRepository>.value(
47                    value: model.noteRepository,
48                  ), // Provider.value
49                ],
50                child: GetMaterialApp(
51                  title: 'WorkAround',
52                  home: WelcomeView(),
53                )); // GetMaterialApp, MultiProvider
54          }); // ViewModelBuilder.reactive
55    }
56  }
```

The *WorkAroundVIew* is the widget the main function in *main.dart* runs, it is essentially what kickstarts the application and its here where we can see the Provider in action. The application builds a *MultiProvide*r object on line 23, which in turn returns the first see-able view in the app, the *WelcomeView*, the *Multiprovider* takes in a list of Providers that are essentially the state of a number of services, giving any class that calls Provider access to the service and its fields/functions.

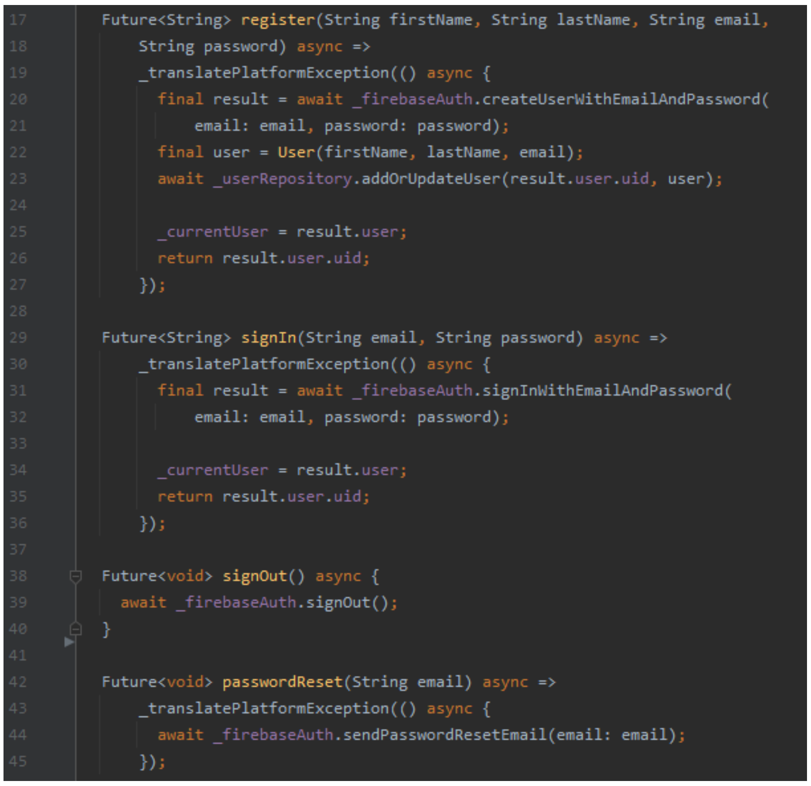
Figure 2. workaround_view_model.dart



```dart
class WorkAroundViewModel extends BaseViewModel{
  NavigationService navigationService;
  ExerciseService exerciseService;
  AuthenticationService auth;
  UserRepository userRepository;
  WorkoutRepository workoutRepository;
  ExerciseRepository exerciseRepository;
  HistoryRepository historyRepository;
  NotesRepository noteRepository;

  Future<void> initialise() async {
    userRepository = UserRepository(FirebaseFirestore.instance);
    workoutRepository = WorkoutRepository(FirebaseFirestore.instance);
    exerciseRepository = ExerciseRepository(FirebaseFirestore.instance);
    historyRepository = HistoryRepository(FirebaseFirestore.instance);
    noteRepository = NotesRepository(FirebaseFirestore.instance);
    navigationService = NavigationService();
    auth = AuthenticationService(FirebaseAuth.instance, userRepository);
    exerciseService = ExerciseService();
  }
}
```

The *WorkAroundViewModel* instantiates all of the services that go into the *Multiprovider*.

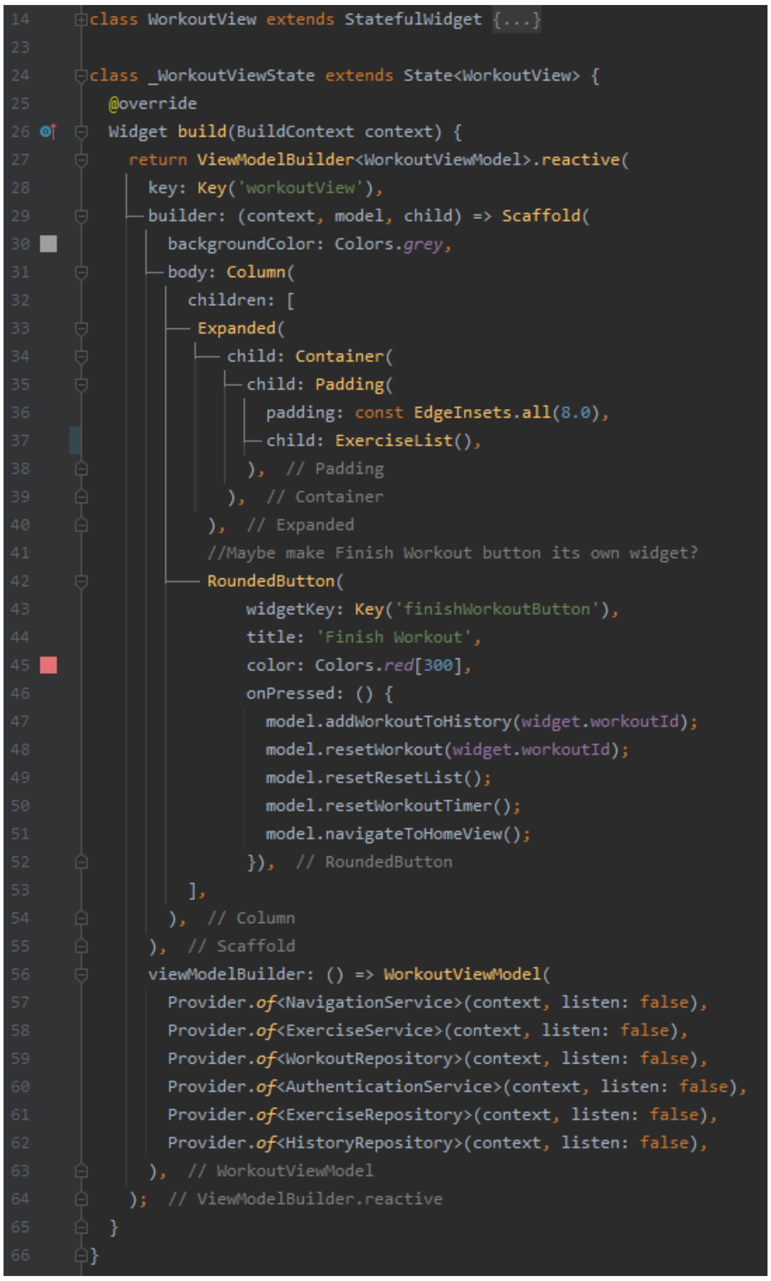
Figure 3. authentication_service.dart



```dart
  Future<String> register(String firstName, String lastName, String email,
      String password) async =>
      _translatePlatformException(() async {
        final result = await _firebaseAuth.createUserWithEmailAndPassword(
            email: email, password: password);
        final user = User(firstName, lastName, email);
        await _userRepository.addOrUpdateUser(result.user.uid, user);

        _currentUser = result.user;
        return result.user.uid;
      });

  Future<String> signIn(String email, String password) async =>
      _translatePlatformException(() async {
        final result = await _firebaseAuth.signInWithEmailAndPassword(
            email: email, password: password);

        _currentUser = result.user;
        return result.user.uid;
      });

  Future<void> signOut() async {
    await _firebaseAuth.signOut();
  }

  Future<void> passwordReset(String email) async =>
      _translatePlatformException(() async {
        await _firebaseAuth.sendPasswordResetEmail(email: email);
      });
```

As an example of one of the services, Figure 3 is a snippet of some of the major code in the *AuthenticationService*. These functions interact with the *FirebaseAuthentication* class which is a class from the Firebase library. Note, the *register()* function called the *UserRepository*, the *addOrUpdateUser* takes in the newly created user and adds them to the Firestore database, this makes it so that a user can access their account from any device.

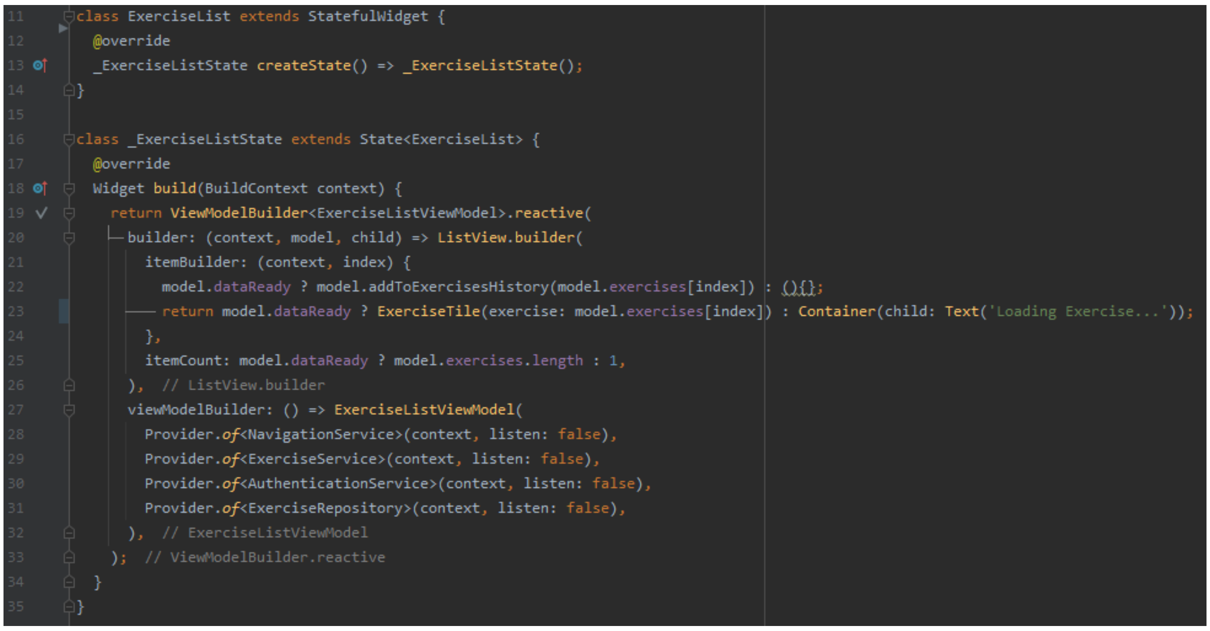
Figure 4. workout_view.dart



```
14  class WorkoutView extends StatefulWidget {...}
23
24  class _WorkoutViewState extends State<WorkoutView> {
25    @override
26    Widget build(BuildContext context) {
27      return ViewModelBuilder<WorkoutViewModel>.reactive(
28        key: Key('workoutView'),
29        builder: (context, model, child) => Scaffold(
30          backgroundColor: Colors.grey,
31          body: Column(
32            children: [
33              Expanded(
34                child: Container(
35                  child: Padding(
36                    padding: const EdgeInsets.all(8.0),
37                    child: ExerciseList(),
38                  ), // Padding
39                ), // Container
40              ), // Expanded
41              //Maybe make Finish Workout button its own widget?
42              RoundedButton(
43                  widgetKey: Key('finishWorkoutButton'),
44                  title: 'Finish Workout',
45                  color: Colors.red[300],
46                  onPressed: () {
47                    model.addWorkoutToHistory(widget.workoutId);
48                    model.resetWorkout(widget.workoutId);
49                    model.resetResetList();
50                    model.resetWorkoutTimer();
51                    model.navigateToHomeView();
52                  }), // RoundedButton
53            ],
54          ), // Column
55        ), // Scaffold
56        viewModelBuilder: () => WorkoutViewModel(
57          Provider.of<NavigationService>(context, listen: false),
58          Provider.of<ExerciseService>(context, listen: false),
59          Provider.of<WorkoutRepository>(context, listen: false),
60          Provider.of<AuthenticationService>(context, listen: false),
61          Provider.of<ExerciseRepository>(context, listen: false),
62          Provider.of<HistoryRepository>(context, listen: false),
63        ), // WorkoutViewModel
64      ); // ViewModelBuilder.reactive
65    }
66  }
```

To showcase the View and View Model relationship between the UI and logic, we will go through the code to see what it takes to generate a workout. In Figure 4 we see the *WorkoutView*, this class is what builds the user interface, it is in charge of what the user will see. Starting from like 31, the UI is essentially a series of Widgets/UI elements built to give us the complete UI of the page. It culminates on line 37 with *ExerciseList()* which is a widget built to contain a list view of all the exercises in a workout. All of these widgets are wrapped in a *ViewModelBuilder* on line 27, which specifies its type as a *WorkoutViewModel* and on line 56, we can see the instances of every class that Provider provides to the view model. The functions on lines 47 through 51 are all defined in the view model, so all the logic is done there and abstracted away from the UI. This is a consistent set up across many files in the app.

Figure 5. exercise_list.dart



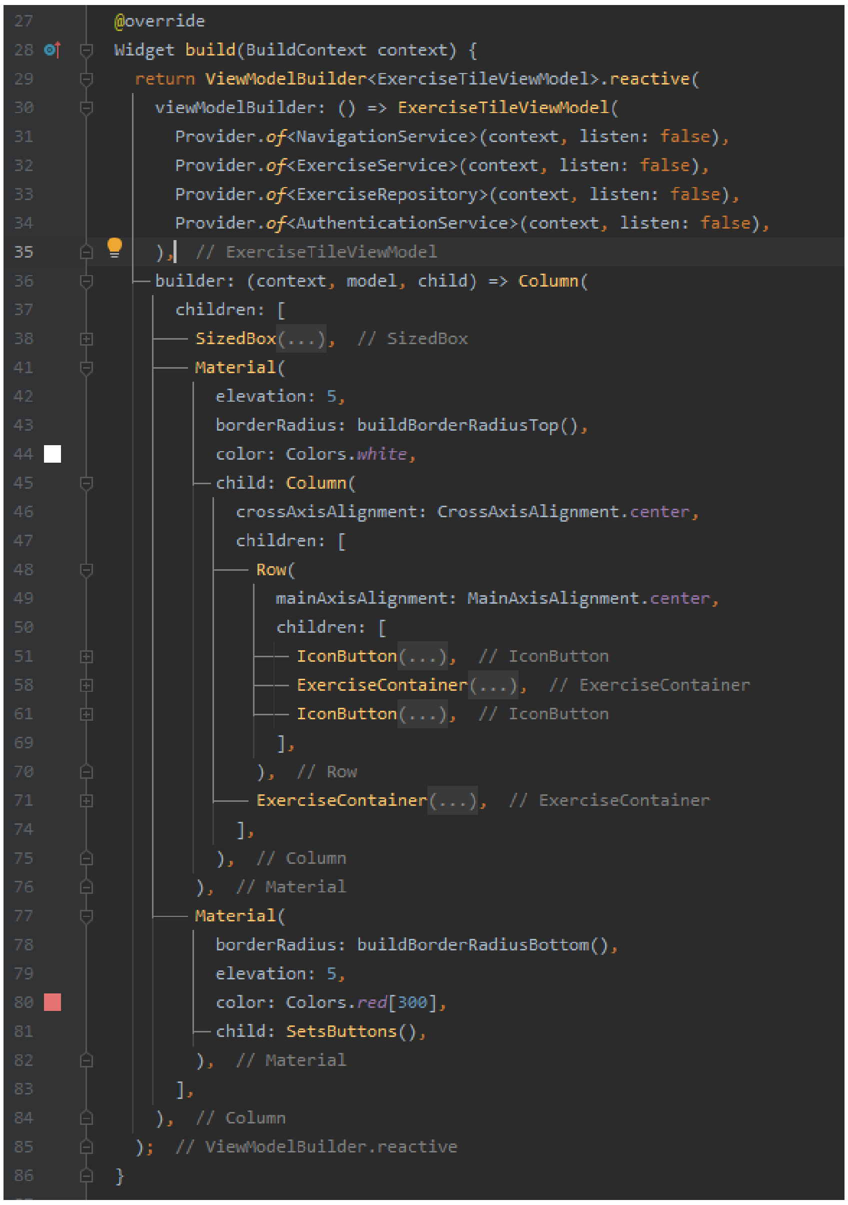
```dart
class ExerciseList extends StatefulWidget {
  @override
  _ExerciseListState createState() => _ExerciseListState();
}

class _ExerciseListState extends State<ExerciseList> {
  @override
  Widget build(BuildContext context) {
    return ViewModelBuilder<ExerciseListViewModel>.reactive(
      builder: (context, model, child) => ListView.builder(
        itemBuilder: (context, index) {
          model.dataReady ? model.addToExercisesHistory(model.exercises[index]) : (){};
          return model.dataReady ? ExerciseTile(exercise: model.exercises[index]) : Container(child: Text('Loading Exercise...'));
        },
        itemCount: model.dataReady ? model.exercises.length : 1,
      ),  // ListView.builder
      viewModelBuilder: () => ExerciseListViewModel(
        Provider.of<NavigationService>(context, listen: false),
        Provider.of<ExerciseService>(context, listen: false),
        Provider.of<AuthenticationService>(context, listen: false),
        Provider.of<ExerciseRepository>(context, listen: false),
      ),  // ExerciseListViewModel
    );  // ViewModelBuilder.reactive
  }
}
```

Next on the series of events that generate a workout is the *ExerciseList.* It is a *ListView* that builds and contains an *ExerciseTile* for every exercise that is streamed from the *ExerciseRepository* connected to the Firestore database. *model.exercises* is a getter method for returning the list of exercises from the database. *model.dataReady* is a built in getter from the Stacked library that returns true when the data streamed from the database is ready to be used, without this, the UI would error because it would try to build widgets with data that is not there yet.

model.dataReady ? ExerciseTile(exercise: model.exercises[index]) : Container(child: Text('Loading Exercises…'));

This line is a ternary operation, it basically means, when the data is ready, display the *ExerciseTile*, if not, display the loading container.

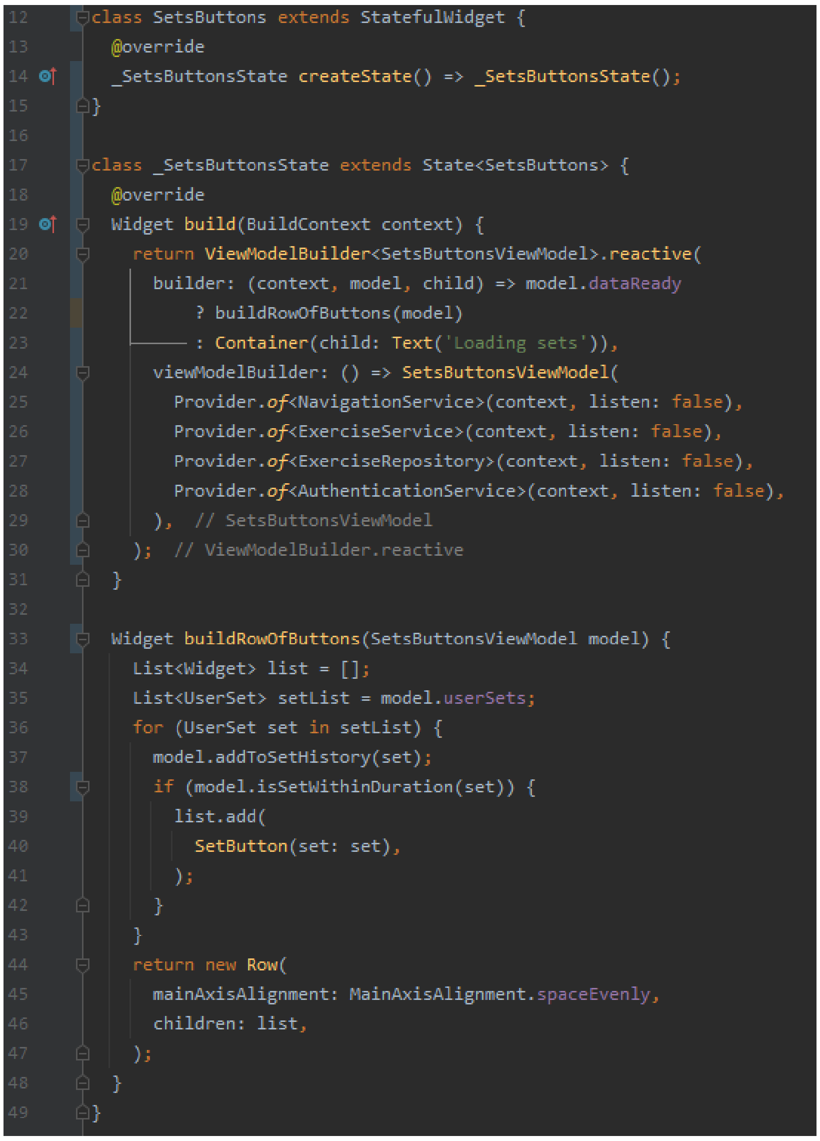
Figure 6. exercise_tile.dart



```
27    @override
28    Widget build(BuildContext context) {
29      return ViewModelBuilder<ExerciseTileViewModel>.reactive(
30        viewModelBuilder: () => ExerciseTileViewModel(
31          Provider.of<NavigationService>(context, listen: false),
32          Provider.of<ExerciseService>(context, listen: false),
33          Provider.of<ExerciseRepository>(context, listen: false),
34          Provider.of<AuthenticationService>(context, listen: false),
35        ), // ExerciseTileViewModel
36        builder: (context, model, child) => Column(
37          children: [
38            SizedBox(...), // SizedBox
41            Material(
42              elevation: 5,
43              borderRadius: buildBorderRadiusTop(),
44              color: Colors.white,
45              child: Column(
46                crossAxisAlignment: CrossAxisAlignment.center,
47                children: [
48                  Row(
49                    mainAxisAlignment: MainAxisAlignment.center,
50                    children: [
51                      IconButton(...), // IconButton
58                      ExerciseContainer(...), // ExerciseContainer
61                      IconButton(...), // IconButton
69                    ],
70                  ), // Row
71                  ExerciseContainer(...), // ExerciseContainer
74                ],
75              ), // Column
76            ), // Material
77            Material(
78              borderRadius: buildBorderRadiusBottom(),
79              elevation: 5,
80              color: Colors.red[300],
81              child: SetsButtons(),
82            ), // Material
83          ],
84        ), // Column
85      ); // ViewModelBuilder.reactive
86    }
```

On the inside of the *ExerciseTile* is another series of widget displaying the UI, the widgets on lines 51 to 61 are collapsed for the sake of the screenshot but line 81 is what we are interested in. In every *ExerciseTile* we build a *SetsButtons* widget.

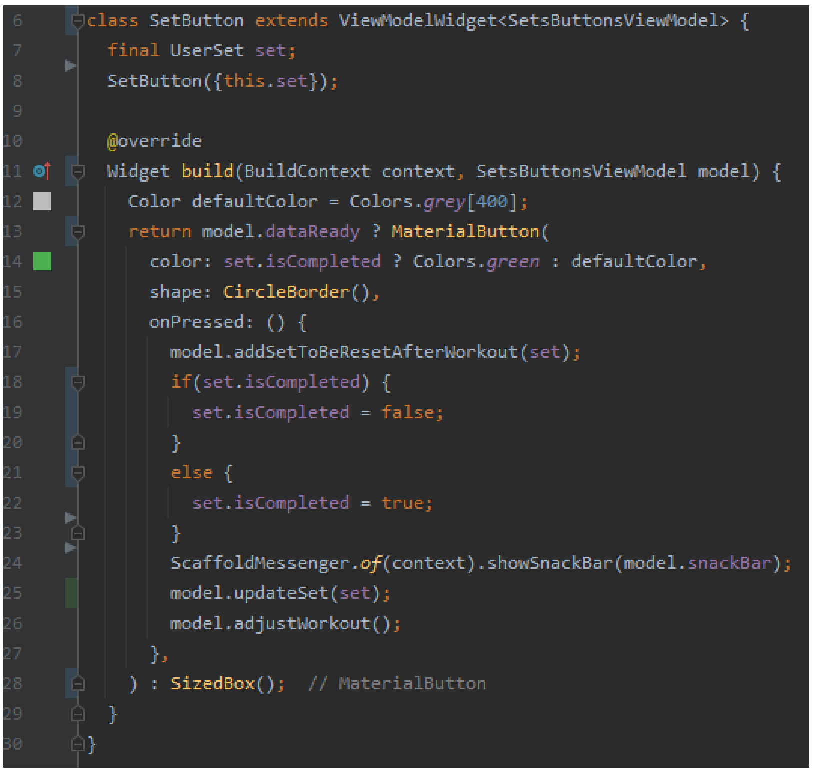
Figure 7. sets_buttons.dart



```
12  class SetsButtons extends StatefulWidget {
13    @override
14    _SetsButtonsState createState() => _SetsButtonsState();
15  }
16
17  class _SetsButtonsState extends State<SetsButtons> {
18    @override
19    Widget build(BuildContext context) {
20      return ViewModelBuilder<SetsButtonsViewModel>.reactive(
21        builder: (context, model, child) => model.dataReady
22            ? buildRowOfButtons(model)
23            : Container(child: Text('Loading sets')),
24        viewModelBuilder: () => SetsButtonsViewModel(
25          Provider.of<NavigationService>(context, listen: false),
26          Provider.of<ExerciseService>(context, listen: false),
27          Provider.of<ExerciseRepository>(context, listen: false),
28          Provider.of<AuthenticationService>(context, listen: false),
29        ), // SetsButtonsViewModel
30      ); // ViewModelBuilder.reactive
31    }
32
33    Widget buildRowOfButtons(SetsButtonsViewModel model) {
34      List<Widget> list = [];
35      List<UserSet> setList = model.userSets;
36      for (UserSet set in setList) {
37        model.addToSetHistory(set);
38        if (model.isSetWithinDuration(set)) {
39          list.add(
40            SetButton(set: set),
41          );
42        }
43      }
44      return new Row(
45        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
46        children: list,
47      );
48    }
49  }
```

Inside the *SetsButtons* widget on line 21 to 22, we build a widget called *buildRowOfButtons* that returns a row of buttons, with each button representing a set of an exercise in a workout, see Figure 24 for reference. Inside this widget is one part of the logic that generates a dynamic workout, from line 35 to 44, we are returning a list of sets that have been streamed from the database and for every set we are checking if that set is within duration of the workout. If that is true, it builds and adds a set button to the row that is being returned to the *ExerciseTile.*

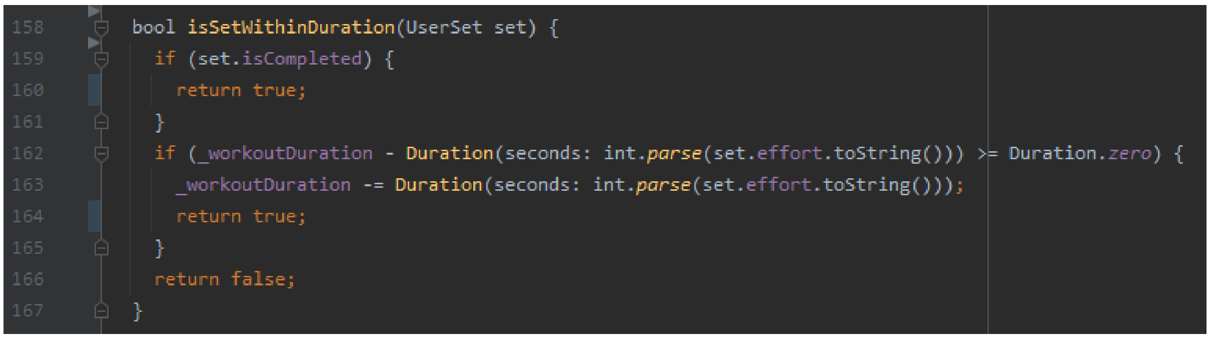
Figure 8. set_button.dart



```dart
class SetButton extends ViewModelWidget<SetsButtonsViewModel> {
  final UserSet set;
  SetButton({this.set});

  @override
  Widget build(BuildContext context, SetsButtonsViewModel model) {
    Color defaultColor = Colors.grey[400];
    return model.dataReady ? MaterialButton(
      color: set.isCompleted ? Colors.green : defaultColor,
      shape: CircleBorder(),
      onPressed: () {
        model.addSetToBeResetAfterWorkout(set);
        if(set.isCompleted) {
          set.isCompleted = false;
        }
        else {
          set.isCompleted = true;
        }
        ScaffoldMessenger.of(context).showSnackBar(model.snackBar);
        model.updateSet(set);
        model.adjustWorkout();
      },
    ) : SizedBox();  // MaterialButton
  }
}
```

Inside the *SetButton* widget, we have a few things going on. First we are checking to see if the set has been completed, if it is, the circle is coloured green, so the user knows that have completed that set. When the *SetButton* is pressed, it checks to see if the set was completed, this means that if the user accidentally tapped a set button prematurely, they could tap it again to undo it. If the set is not completed, it sets the set as completed and updates the Firestore database with the set marked as complete and adjusts the workout. This is important for the *isSetWithinDuration()* function below.

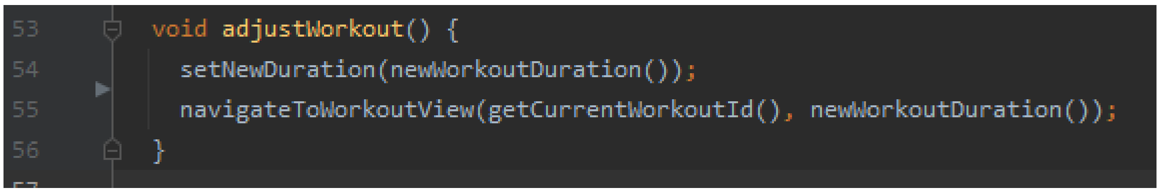
Figure 9. isSetWithinDuration() in exercise_service.dart



```dart
bool isSetWithinDuration(UserSet set) {
  if (set.isCompleted) {
    return true;
  }
  if (_workoutDuration - Duration(seconds: int.parse(set.effort.toString())) >= Duration.zero) {
    _workoutDuration -= Duration(seconds: int.parse(set.effort.toString()));
    return true;
  }
  return false;
}
```

In the *ExerciseService* sits this function that takes in the set object. It checks to see if the set is completed and if so, returns true. This means that when the app is generating a workout, if a set is marked as complete, it will not be considered against the duration of the workout since the user has completed that set, it does not need to be accounted for. If the set is not completed, it checks to see if the remaining duration of the workout minus how long it expects the workout to take is greater than zero, it subtracts the effort of the set from the workout duration and returns true, meaning that set will be added to the exercise. If the set took longer than the remaining time, this would return false, and the set would not be added to the list of *SetButtons* being rendered in the *ExerciseTile*.

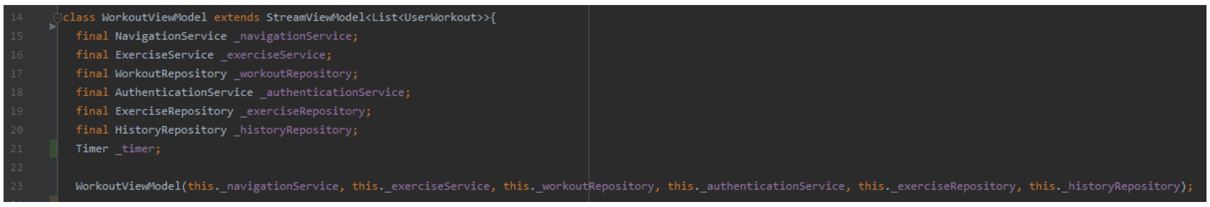
Figure 10. adjustWorkout() in exercise_tile_view_model.dart



```dart
void adjustWorkout() {
  setNewDuration(newWorkoutDuration());
  navigateToWorkoutView(getCurrentWorkoutId(), newWorkoutDuration());
}
```

When the user taps a set button and completes a set, that is when the app will update and adjust the workout depending on how much time is left. It sets the new duration of the workout as the initial duration entered minus the time that has elapsed and rebuilds the *WorkoutView*.

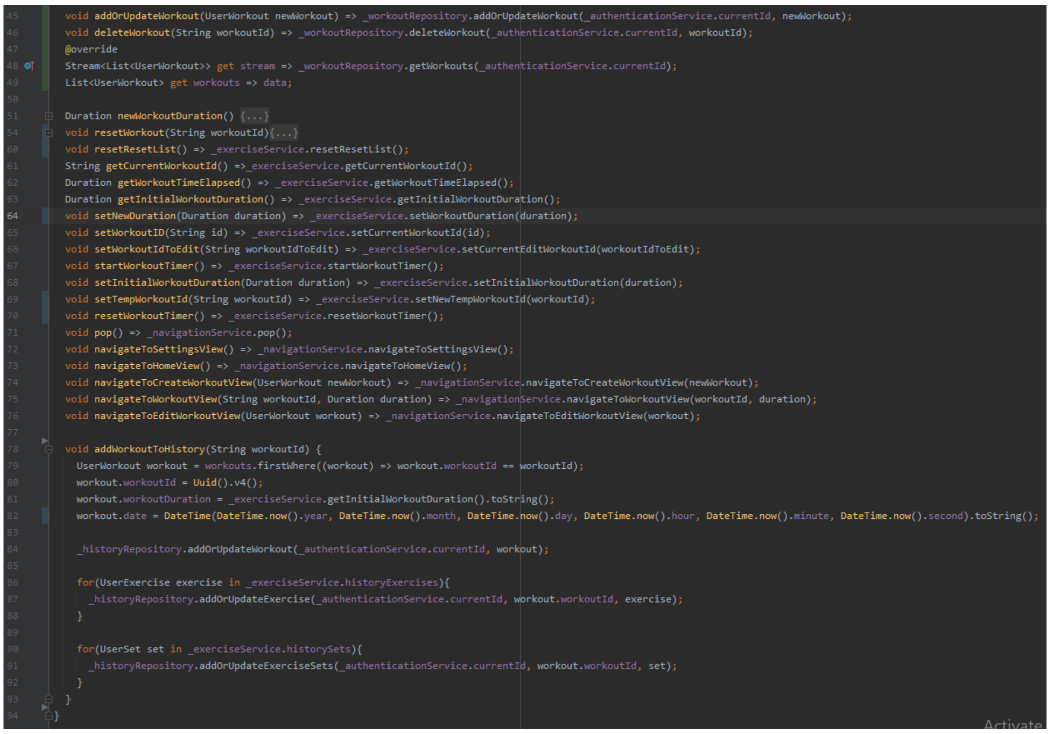
Figure 11. workout_view_model.dart



```dart
class WorkoutViewModel extends StreamViewModel<List<UserWorkout>>{
  final NavigationService _navigationService;
  final ExerciseService _exerciseService;
  final WorkoutRepository _workoutRepository;
  final AuthenticationService _authenticationService;
  final ExerciseRepository _exerciseRepository;
  final HistoryRepository _historyRepository;
  Timer _timer;

  WorkoutViewModel(this._navigationService, this._exerciseService, this._workoutRepository, this._authenticationService, this._exerciseRepository, this._historyRepository);
```

Figures 11 and 12 are the top and bottom half of the *WorkoutViewModel*, it is one of the larger view models in the application. Figure 11 shows that the view model extends the *StreamViewModel* which is a class from the Stacked library, it helps manage state and streaming of the objects that are streamed from the Firestore database. You can see that most of the functions in the view model call on the services that have been passed in by Provider. The function *addWorkoutToHistory()* is what saves the workout in the history collection in the database. The image below can be resized for clarity.

Figure 12. workout_view_model.dart II



```dart
  void addOrUpdateWorkout(UserWorkout newWorkout) => _workoutRepository.addOrUpdateWorkout(_authenticationService.currentId, newWorkout);
  void deleteWorkout(String workoutId) => _workoutRepository.deleteWorkout(_authenticationService.currentId, workoutId);
  @override
  Stream<List<UserWorkout>> get stream => _workoutRepository.getWorkouts(_authenticationService.currentId);
  List<UserWorkout> get workouts => data;

  Duration newWorkoutDuration() {...}
  void resetWorkout(String workoutId){...}
  void resetResetList() => _exerciseService.resetResetList();
  String getCurrentWorkoutId() =>_exerciseService.getCurrentWorkoutId();
  Duration getWorkoutTimeElapsed() => _exerciseService.getWorkoutTimeElapsed();
  Duration getInitialWorkoutDuration() => _exerciseService.getInitialWorkoutDuration();
  void setNewDuration(Duration duration) => _exerciseService.setWorkoutDuration(duration);
  void setWorkoutID(String id) => _exerciseService.setCurrentWorkoutId(id);
  void setWorkoutIdToEdit(String workoutIdToEdit) => _exerciseService.setCurrentEditWorkoutId(workoutIdToEdit);
  void startWorkoutTimer() => _exerciseService.startWorkoutTimer();
  void setInitialWorkoutDuration(Duration duration) => _exerciseService.setInitialWorkoutDuration(duration);
  void setTempWorkoutId(String workoutId) => _exerciseService.setNewTempWorkoutId(workoutId);
  void resetWorkoutTimer() => _exerciseService.resetWorkoutTimer();
  void pop() => _navigationService.pop();
  void navigateToSettingsView() => _navigationService.navigateToSettingsView();
  void navigateToHomeView() => _navigationService.navigateToHomeView();
  void navigateToCreateWorkoutView(UserWorkout newWorkout) => _navigationService.navigateToCreateWorkoutView(newWorkout);
  void navigateToWorkoutView(String workoutId, Duration duration) => _navigationService.navigateToWorkoutView(workoutId, duration);
  void navigateToEditWorkoutView(UserWorkout workout) => _navigationService.navigateToEditWorkoutView(workout);

  void addWorkoutToHistory(String workoutId) {
    UserWorkout workout = workouts.firstWhere((workout) => workout.workoutId == workoutId);
    workout.workoutId = Uuid().v4();
    workout.workoutDuration = _exerciseService.getInitialWorkoutDuration().toString();
    workout.date = DateTime(DateTime.now().year, DateTime.now().month, DateTime.now().day, DateTime.now().hour, DateTime.now().minute, DateTime.now().second).toString();

    _historyRepository.addOrUpdateWorkout(_authenticationService.currentId, workout);

    for(UserExercise exercise in _exerciseService.historyExercises){
      _historyRepository.addOrUpdateExercise(_authenticationService.currentId, workout.workoutId, exercise);
    }

    for(UserSet set in _exerciseService.historySets){
      _historyRepository.addOrUpdateExerciseSets(_authenticationService.currentId, workout.workoutId, set);
    }
  }
}
```

Figure 13. user_repository.dart

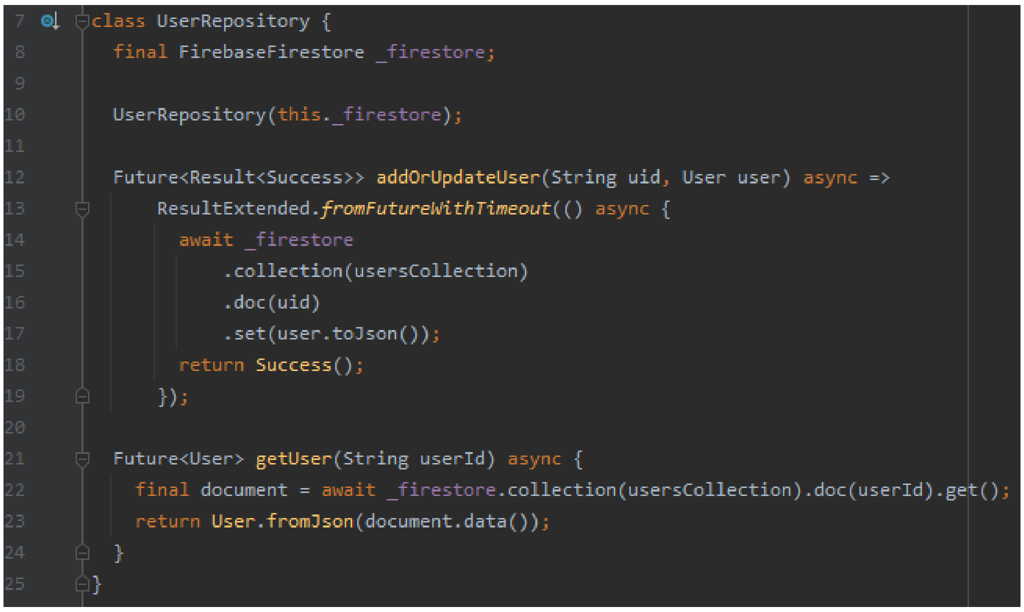
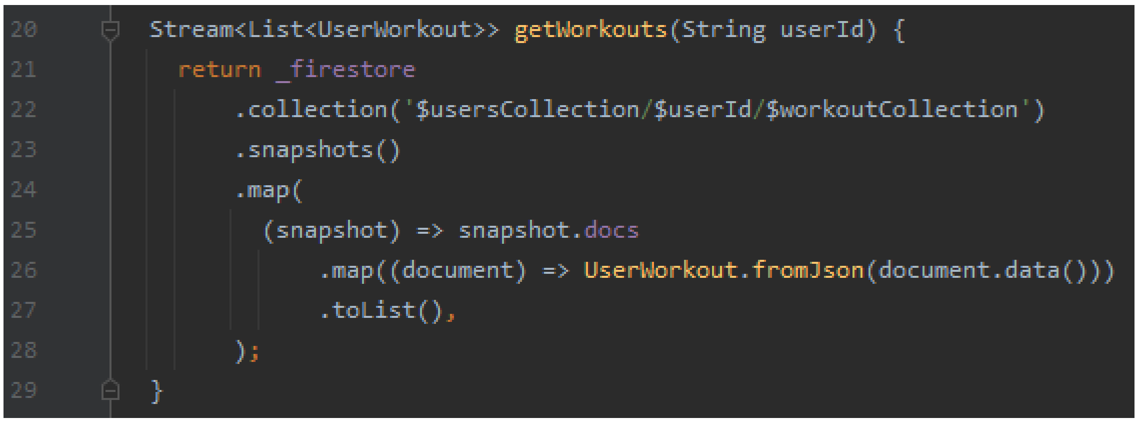

```dart
class UserRepository {
  final FirebaseFirestore _firestore;

  UserRepository(this._firestore);

  Future<Result<Success>> addOrUpdateUser(String uid, User user) async =>
      ResultExtended.fromFutureWithTimeout(() async {
        await _firestore
            .collection(usersCollection)
            .doc(uid)
            .set(user.toJson());
        return Success();
      });

  Future<User> getUser(String userId) async {
    final document = await _firestore.collection(usersCollection).doc(userId).get();
    return User.fromJson(document.data());
  }
}
```

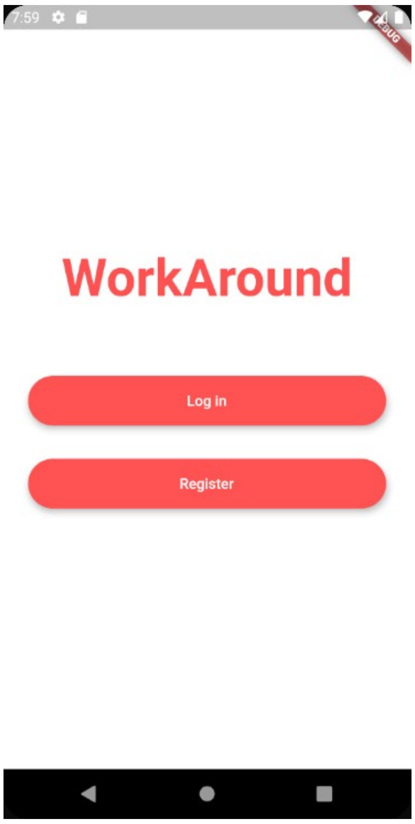
Figure 13 is a screenshot of the *UserRepository* it is the smallest of the repositories that I chose as an example to showcase some calls to a Firebase repository. The *getUser()* function returns a single user document using the *userId* that was retrieved from the *AuthenticationService*. Figure 14 shows what a similar function looks like for returning a list of multiple documents.

Figure 14. getWorkouts from workout_repository.dart


```dart
Stream<List<UserWorkout>> getWorkouts(String userId) {
  return _firestore
      .collection('$usersCollection/$userId/$workoutCollection')
      .snapshots()
      .map(
        (snapshot) => snapshot.docs
            .map((document) => UserWorkout.fromJson(document.data()))
            .toList(),
      );
}
```

2.4. Graphical User Interface (GUI)

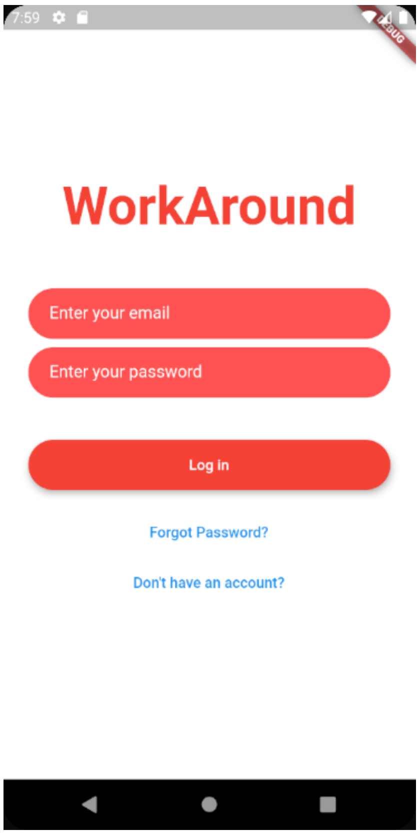
Figure 15. Welcome View



In this section of the report, I will run through all the most notable GUI elements.

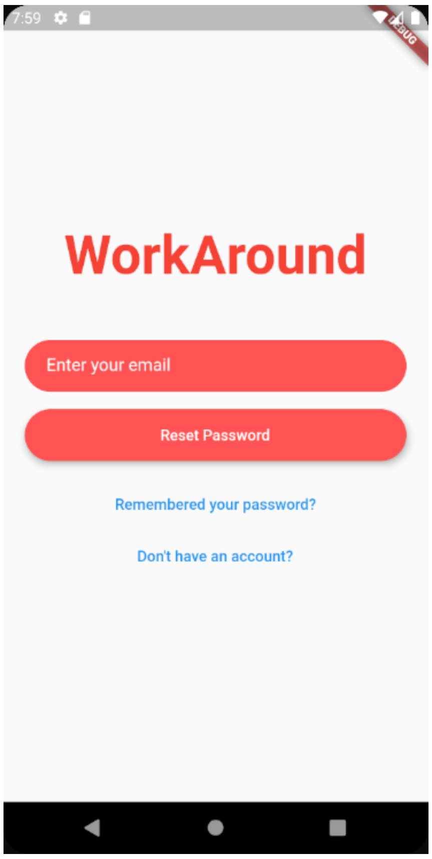
The Welcome View is a simple page with the title and two buttons for registration and login. This is the starting page of the app so it is the first page the users will see. It is kept clean and simple as to not overwhelm the user, this design is similar in all the authorisation views and not just the welcome screen.

Figure 16. Login View



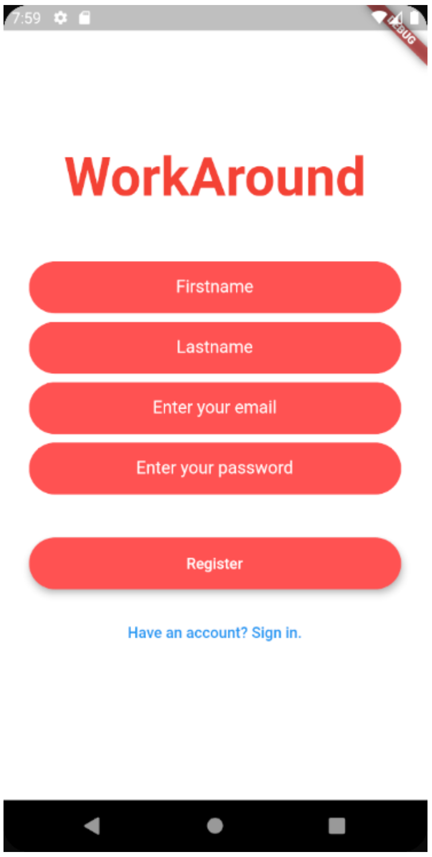
Keeping the clean and simple theme, the Login View has a few text fields and a login button, alongside two more text buttons that redirect the user to the other authorisation views, Register and Reset Password.

Figure 17. Reset Password View



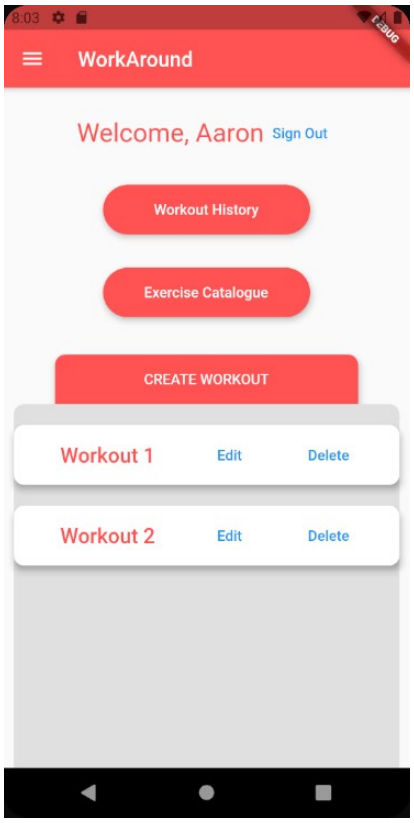
The Reset Password View follows the theme of the previous two pages.

Figure 18. Register View



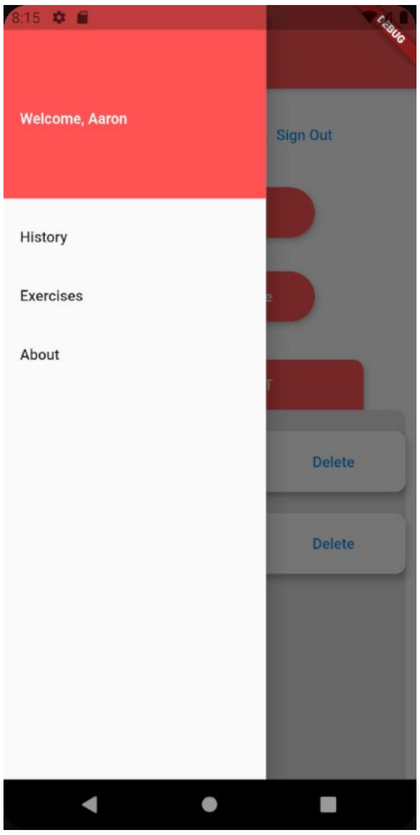
The same goes for the Register View.

Figure 19. Home View



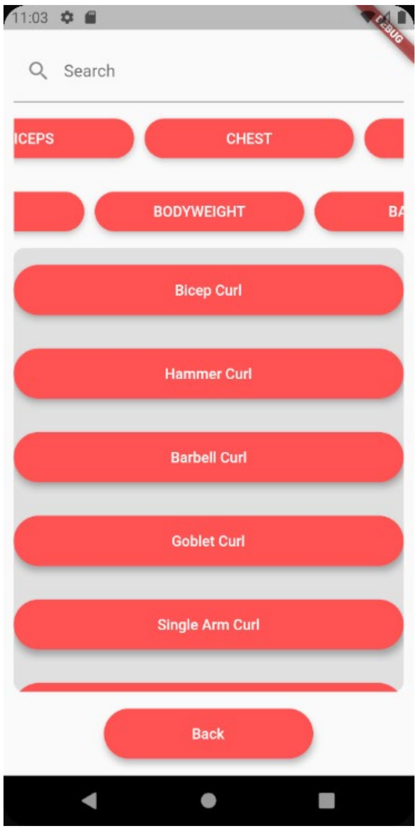
The Home View contains the list of a user's custom workouts and from here, can utilize most of the applications functionality. The user can check their workout history or browse the exercise catalogue, as well as create a new custom workout or edit and delete one of their workouts. Tapping on the workout name in the workout list will prompt the user to input a duration and will lead them to the workout view.

Figure 20. Home View Drawer



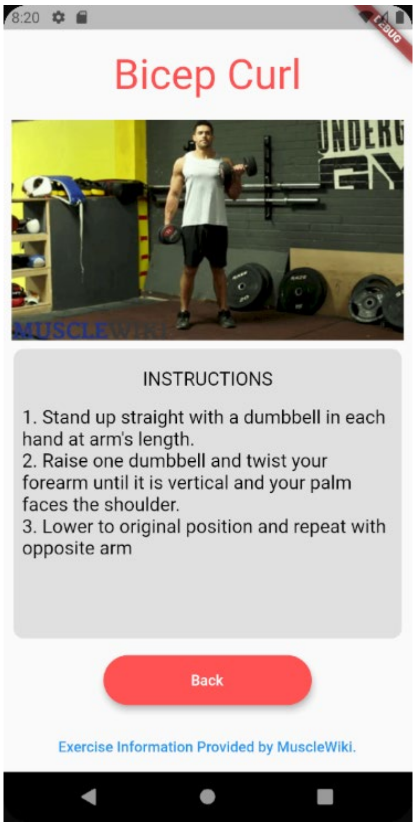
Tapping on the drawer button on the top left of the Home View will open the drawer on the side of the page, here the user has more navigation options, including navigation to the About View.

Figure 21. Exercise Catalogue



The Exercise Catalogue View, in code as *ExerciseView*, is the list of over 120 different exercises the user may add to one of their workouts. The user can filter the list by entering text into the search or by tapping one of the filter buttons above the list. These buttons filter the exercises by either muscle group or by type/equipment.

Figure 22. Exercise Information



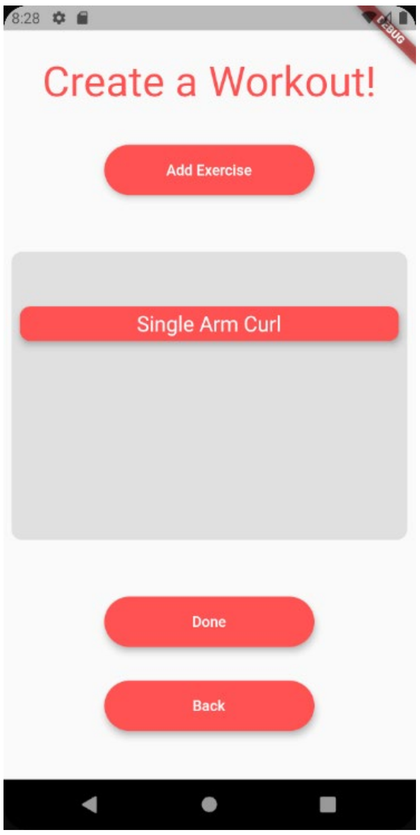
Bicep Curl

INSTRUCTIONS

1. Stand up straight with a dumbbell in each hand at arm's length.
2. Raise one dumbbell and twist your forearm until it is vertical and your palm faces the shoulder.
3. Lower to original position and repeat with opposite arm

Tapping an exercise in the Exercise Catalogue will lead the user to the Exercise Information View, here the user will be shown a visual demonstration of the exercise and written instructions. In adherence to the MuscleWiki copyright, where the information is provided, the MuscleWiki branding is left intact on the gif and a link to the MuscleWiki Website can be seen at the bottom of every exercise page.

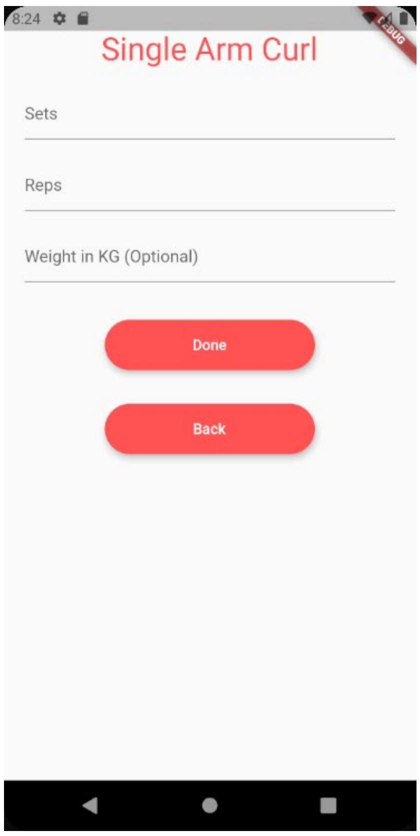
Figure 23. Create Workout View



The Create Workout View lets the user create a workout and add it to their list of workouts. Clicking the "Add Exercise" button will lead them to the Exercise Catalogue. "Done" will add the exercise to the Workout List and "Back" will cancel the process.

Figure 24. Add/Edit Exercise View

The Add/Edit Exercise View will let the user specify the number of sets and repetitions for any given exercise, as well as their desired weight. This view is identical between both Add and Edit Exercise and functions almost entirely the same.



The Workout View is the page the user will see when they start a workout. It will generate a number of sets per exercise depending on the amount of time they put as the duration of the exercise. In this example. I input a duration of 7 minutes and since the app deemed this was not enough time to complete 3 sets of each exercise as was initially set in the workout, it took two sets off the Barbell Curl exercise. This will work inversely too if the app finds that the user has regained time by being faster, it may add sets back. Tapping the circular information button to the left of the exercise name will display the exercises demonstration and instructions. Tapping the icon on the right of the exercise name will open the Notes view where the user may add and delete notes.

Figure 25. Workout View

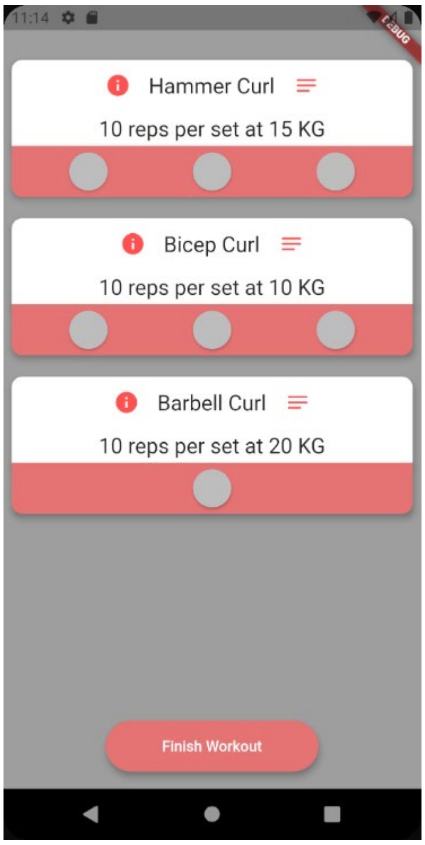
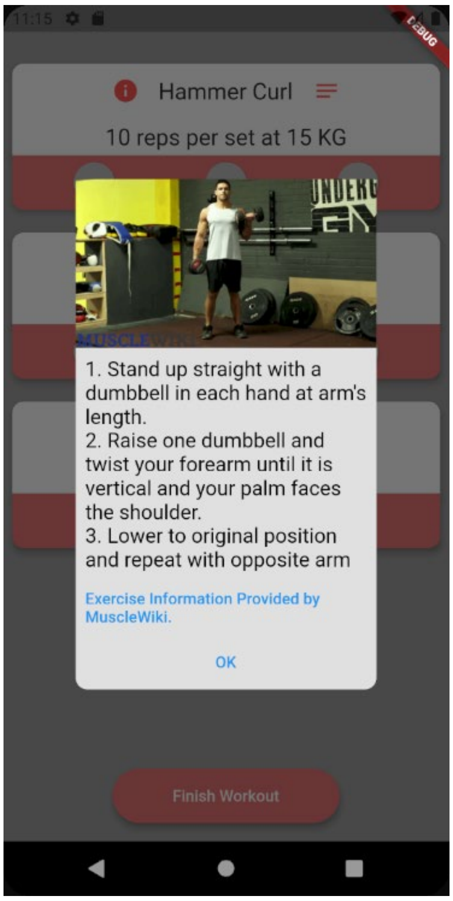


Figure 26. Instructions in Workout View

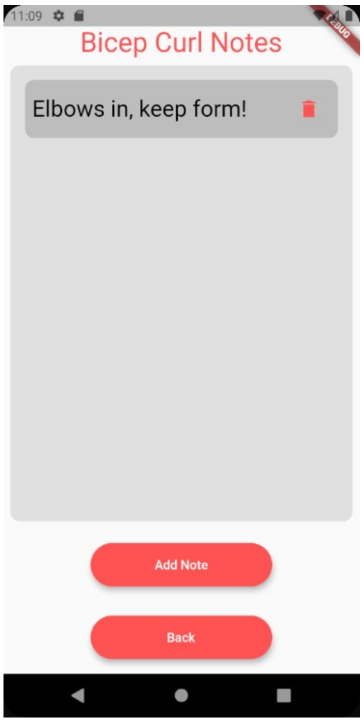


1. Stand up straight with a dumbbell in each hand at arm's length.
2. Raise one dumbbell and twist your forearm until it is vertical and your palm faces the shoulder.
3. Lower to original position and repeat with opposite arm

Exercise Information Provided by MuscleWiki.

When the user taps the exercise information button, they will be shown the exercise instructions and demonstration in a small pop-up window.

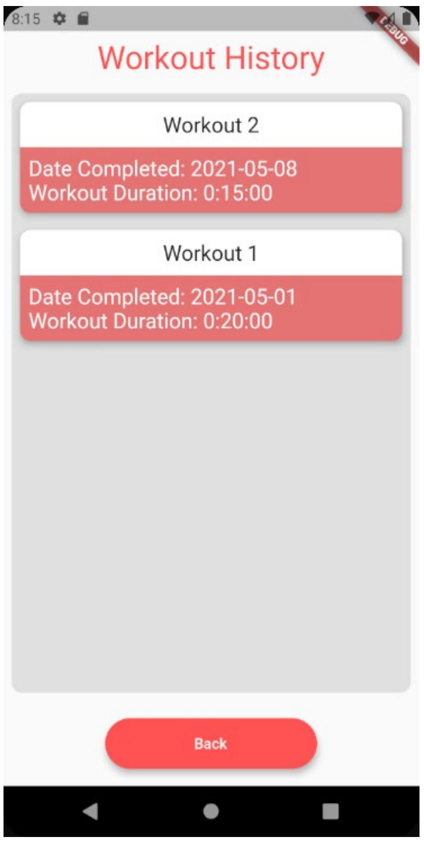
Figure 27. Notes View



Elbows in, keep form!

When the user taps the note icon on the right hand side of the exercise name in the workout view, they will be directed to the Note View where they can add and delete notes. Tapping “Add Note” opens a dialog box where they may input a note.

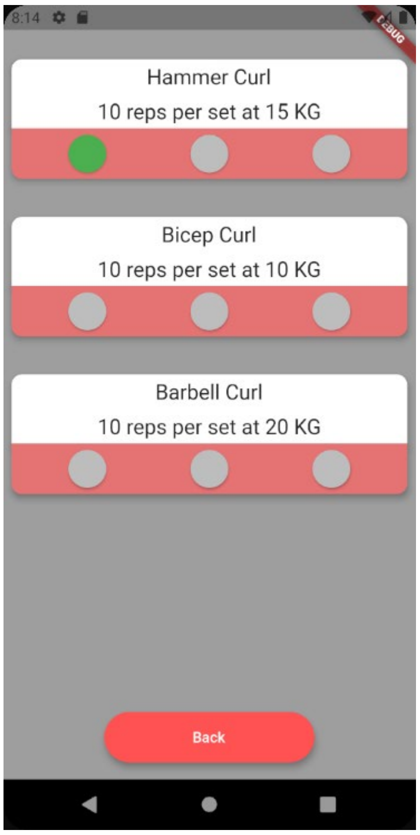
When a user has completed a workout, the workout will be added to their workout history along with the date completed and the initial duration of the exercise. Tapping one of these tiles will display the workout.



Workout 2
Date Completed: 2021-05-08
Workout Duration: 0:15:00

Workout 1
Date Completed: 2021-05-01
Workout Duration: 0:20:00

Figure 29. Workout History View Extended



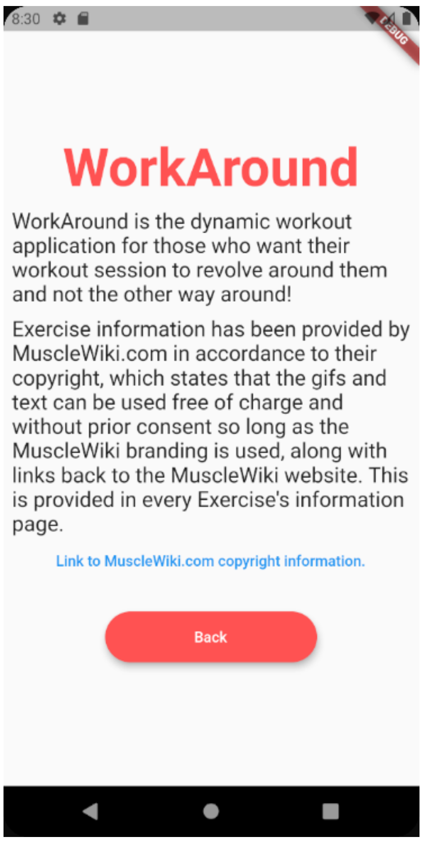
Hammer Curl
10 reps per set at 15 KG

Bicep Curl
10 reps per set at 10 KG

Barbell Curl
10 reps per set at 20 KG

After tapping on a workout in the Workout History view, the user will be shown this page, showing the exercises they did, and the sets that they completed in this workout.

Figure 30. About View



WorkAround

WorkAround is the dynamic workout application for those who want their workout session to revolve around them and not the other way around!

Exercise information has been provided by MuscleWiki.com in accordance to their copyright, which states that the gifs and text can be used free of charge and without prior consent so long as the MuscleWiki branding is used, along with links back to the MuscleWiki website. This is provided in every Exercise's information page.

Link to MuscleWiki.com copyright information.

The About view gives a little description of the WorkAround app and provides the MuscleWiki copyright information.

2.5 Testing

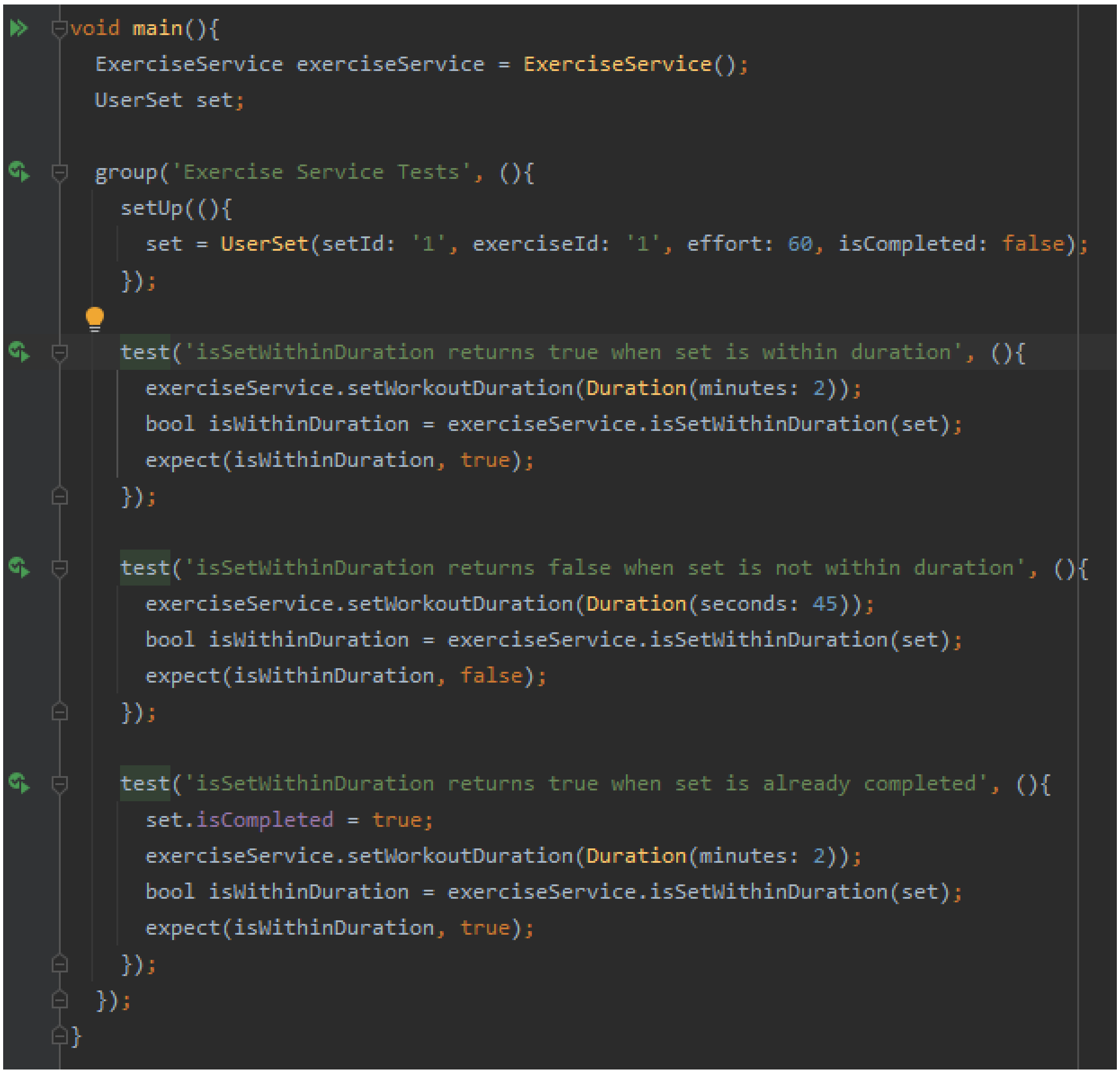
There are three types of testing in Flutter:

- Unit test, A test of a single unit of logic/function or a class
- Widget test, a test of a particular Widget.
- Integration Test, which tests a large portion or a complete app as a whole.

Unit Tests

The goal of a unit test in this project was to determine the correctness of a function under certain conditions. In the ExerciseService class, there is a function called *isSetWithinDuration*. This function is integral to the implementation of the main algorithms that make up the dynamic workout feature of WorkAround. Without this function, the app would not have its innovation.

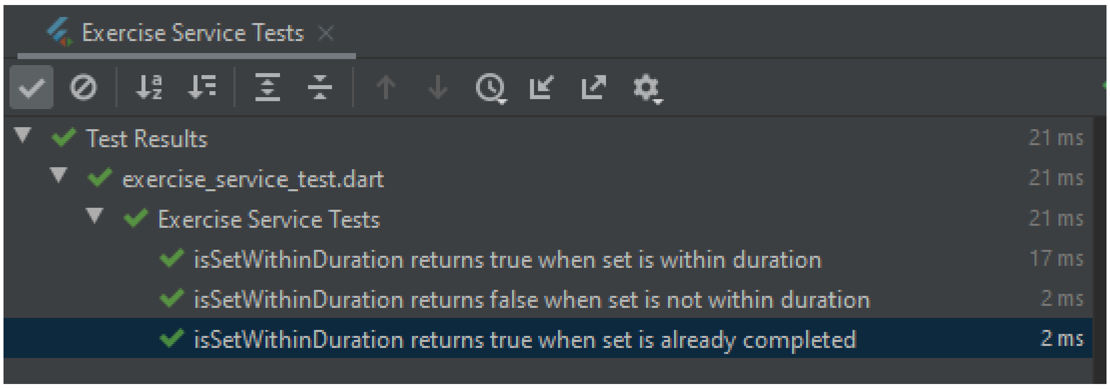
Figure 31. exercise_service_tests.dart



```dart
void main(){
  ExerciseService exerciseService = ExerciseService();
  UserSet set;

  group('Exercise Service Tests', (){
    setUp((){
      set = UserSet(setId: '1', exerciseId: '1', effort: 60, isCompleted: false);
    });

    test('isSetWithinDuration returns true when set is within duration', (){
      exerciseService.setWorkoutDuration(Duration(minutes: 2));
      bool isWithinDuration = exerciseService.isSetWithinDuration(set);
      expect(isWithinDuration, true);
    });

    test('isSetWithinDuration returns false when set is not within duration', (){
      exerciseService.setWorkoutDuration(Duration(seconds: 45));
      bool isWithinDuration = exerciseService.isSetWithinDuration(set);
      expect(isWithinDuration, false);
    });

    test('isSetWithinDuration returns true when set is already completed', (){
      set.isCompleted = true;
      exerciseService.setWorkoutDuration(Duration(minutes: 2));
      bool isWithinDuration = exerciseService.isSetWithinDuration(set);
      expect(isWithinDuration, true);
    });
  });
}
```

In the figure above, the function is tested with several criteria described in the test name, where the output of the function is tested against the expected results.

Figure 32. exercise_service_tests.dart console output



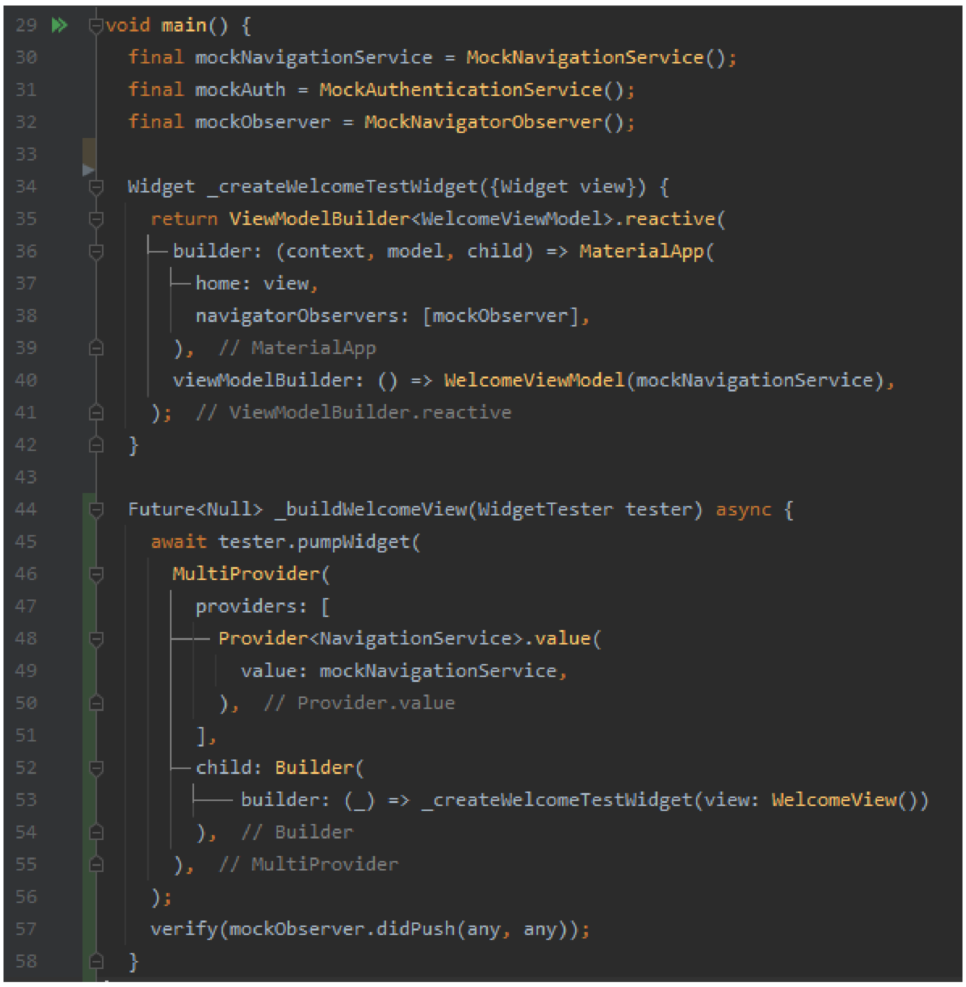
```
Exercise Service Tests
Test Results                                                              21 ms
  exercise_service_test.dart                                              21 ms
    Exercise Service Tests                                                21 ms
      isSetWithinDuration returns true when set is within duration        17 ms
      isSetWithinDuration returns false when set is not within duration    2 ms
      isSetWithinDuration returns true when set is already completed       2 ms
```

In the screenshot of the console after running the tests, we can see these tests pass. If the logic behind the *isSetWithinDuration* function was to ever change, these tests could fail and indicate an issue with the code.

Widget Tests

The aim of a Widget test in this project is the determine that a Widget will interact as expected.

Figure 33. auth_widget_tests.dart

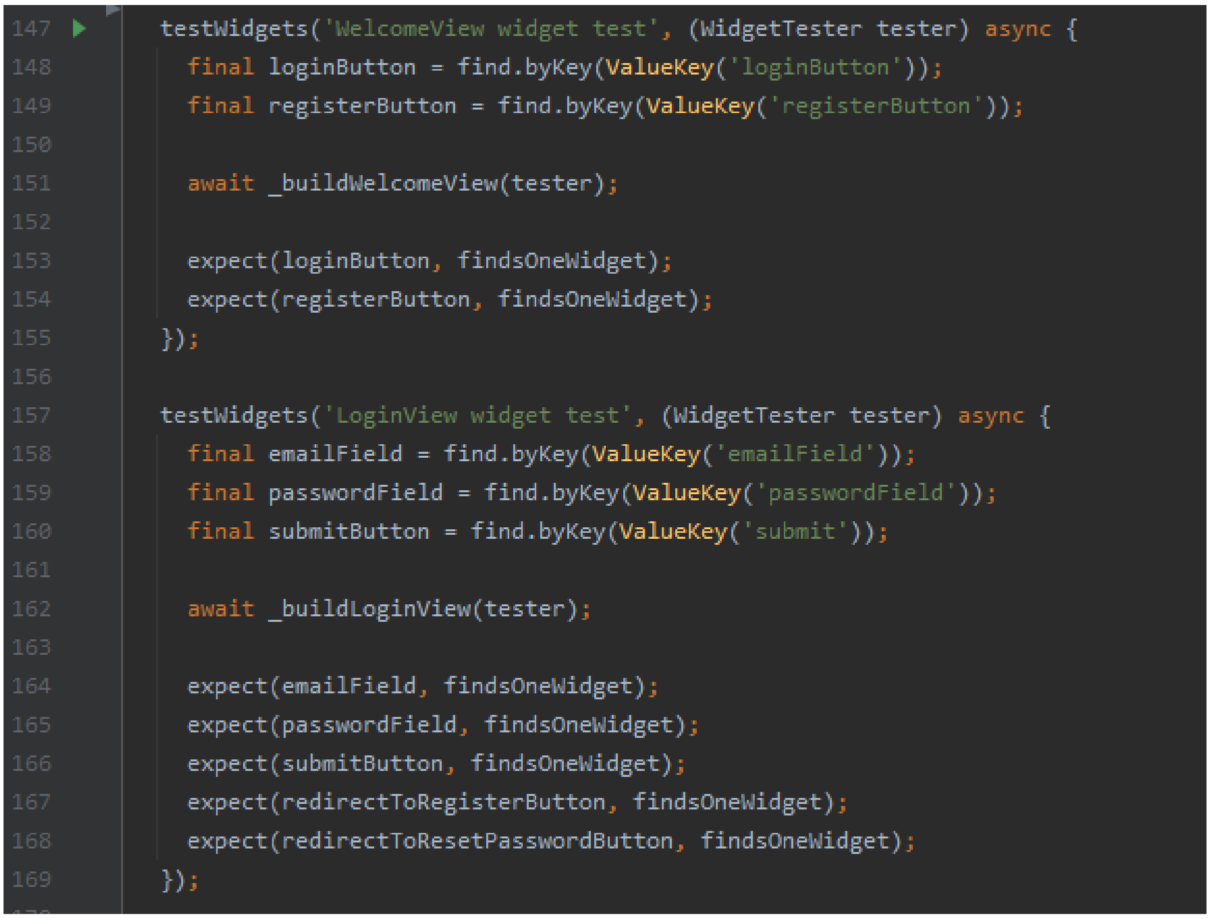


```dart
void main() {
  final mockNavigationService = MockNavigationService();
  final mockAuth = MockAuthenticationService();
  final mockObserver = MockNavigatorObserver();

  Widget _createWelcomeTestWidget({Widget view}) {
    return ViewModelBuilder<WelcomeViewModel>.reactive(
      builder: (context, model, child) => MaterialApp(
        home: view,
        navigatorObservers: [mockObserver],
      ), // MaterialApp
      viewModelBuilder: () => WelcomeViewModel(mockNavigationService),
    ); // ViewModelBuilder.reactive
  }

  Future<Null> _buildWelcomeView(WidgetTester tester) async {
    await tester.pumpWidget(
      MultiProvider(
        providers: [
          Provider<NavigationService>.value(
            value: mockNavigationService,
          ), // Provider.value
        ],
        child: Builder(
          builder: (_) => _createWelcomeTestWidget(view: WelcomeView())
        ), // Builder
      ), // MultiProvider
    );
    verify(mockObserver.didPush(any, any));
  }
```

The figure above is the first image to display in the auth_widget_tests.dart file. It displays the creation of the environment we need to perform the Widget tests. There is a pair of these functions for every

View that we need to build during the tests. Some aspects of these functions to note are the *mockObserver, mockNavigationService*, and *mockAuth* classes and variables. Using a library called Mockito, we can instantiate mocked versions of our real classes to test the implementation of its functions and rather than instantiating the real classes and possibly doing real logic on fake data, we pass these mocked classes into our functions so that no real data is accidentally manipulated but I will cover mocks in more detail later in this section.

Figure 34. auth_widget_tests.dart II



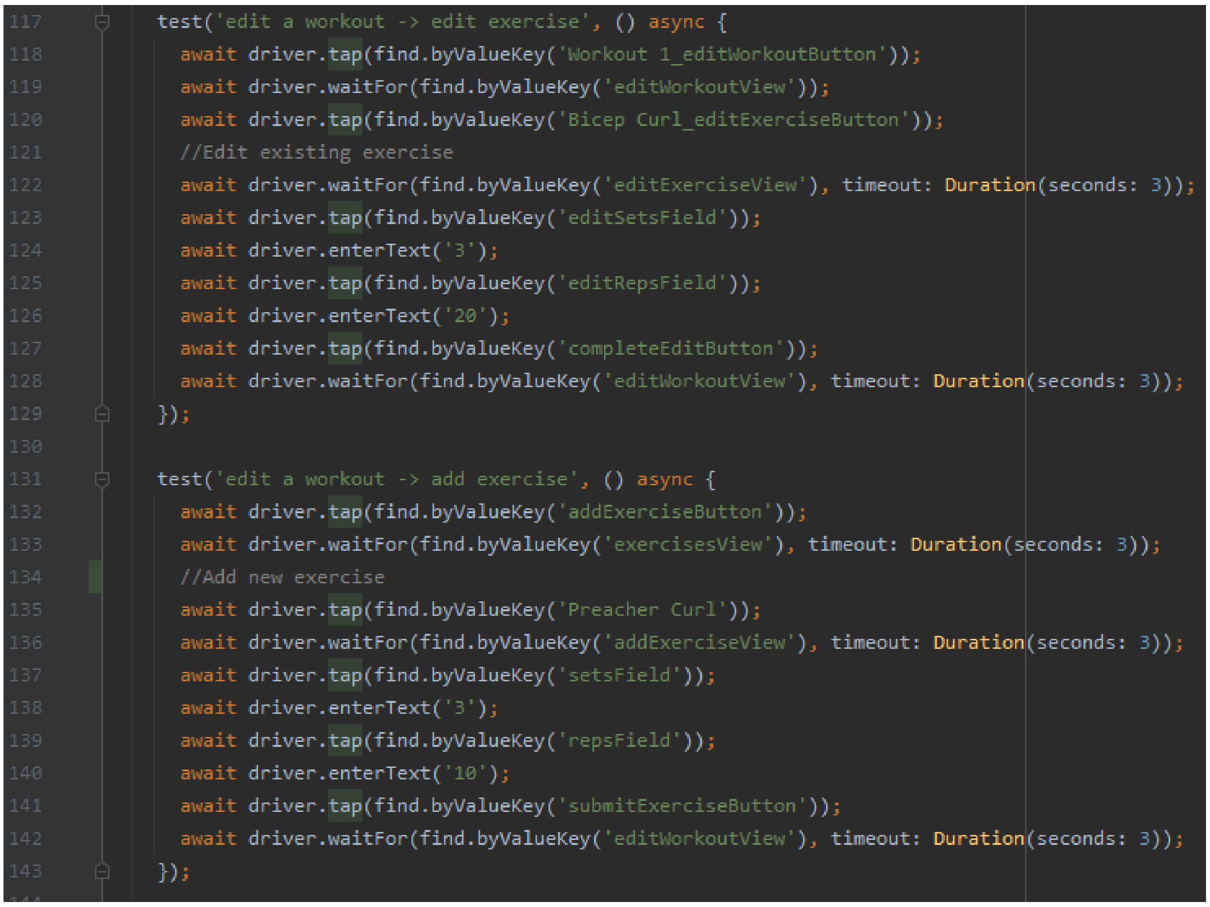
```
testWidgets('WelcomeView widget test', (WidgetTester tester) async {
  final loginButton = find.byKey(ValueKey('loginButton'));
  final registerButton = find.byKey(ValueKey('registerButton'));

  await _buildWelcomeView(tester);

  expect(loginButton, findsOneWidget);
  expect(registerButton, findsOneWidget);
});

testWidgets('LoginView widget test', (WidgetTester tester) async {
  final emailField = find.byKey(ValueKey('emailField'));
  final passwordField = find.byKey(ValueKey('passwordField'));
  final submitButton = find.byKey(ValueKey('submit'));

  await _buildLoginView(tester);

  expect(emailField, findsOneWidget);
  expect(passwordField, findsOneWidget);
  expect(submitButton, findsOneWidget);
  expect(redirectToRegisterButton, findsOneWidget);
  expect(redirectToResetPasswordButton, findsOneWidget);
});
```

In the figure above, we can see the Widget tests in action. Using the first test, *"WelcomeView widget test",* as an example, we can see two variables being defined, these variables are finder methods that will find a Widget by its specified Key, you will notice over the course of examining the project that some Widgets will have a key value defined, this is used to make it easier to find a Widget during testing.
We await the building of the welcome view that we saw in Figure 33 and assert that when the view is built, the *loginButton* and *registerButton* widgets are present.

Integration Test

A Flutter integration test is the test of the functionality of a system as a whole.
The Flutter Driver is an engine that simulates the interactions a user would have with the applications, like tapping a button. Using this driver, we can simulate the user experience in an automated way. Instead of having to test functionality and UI manually, which is monotonous and time consuming, the integration tests and the flutter driver automate this process.

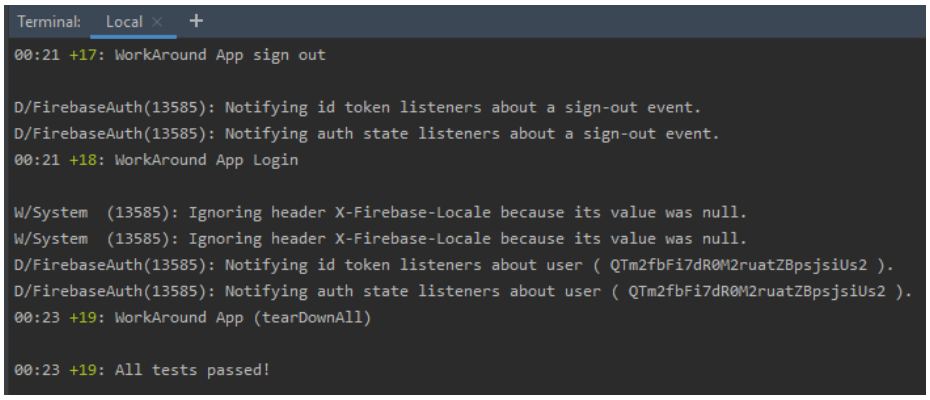
Figure 35. app_test.dart



```
test('edit a workout -> edit exercise', () async {
  await driver.tap(find.byValueKey('Workout 1_editWorkoutButton'));
  await driver.waitFor(find.byValueKey('editWorkoutView'));
  await driver.tap(find.byValueKey('Bicep Curl_editExerciseButton'));
  //Edit existing exercise
  await driver.waitFor(find.byValueKey('editExerciseView'), timeout: Duration(seconds: 3));
  await driver.tap(find.byValueKey('editSetsField'));
  await driver.enterText('3');
  await driver.tap(find.byValueKey('editRepsField'));
  await driver.enterText('20');
  await driver.tap(find.byValueKey('completeEditButton'));
  await driver.waitFor(find.byValueKey('editWorkoutView'), timeout: Duration(seconds: 3));
});

test('edit a workout -> add exercise', () async {
  await driver.tap(find.byValueKey('addExerciseButton'));
  await driver.waitFor(find.byValueKey('exercisesView'), timeout: Duration(seconds: 3));
  //Add new exercise
  await driver.tap(find.byValueKey('Preacher Curl'));
  await driver.waitFor(find.byValueKey('addExerciseView'), timeout: Duration(seconds: 3));
  await driver.tap(find.byValueKey('setsField'));
  await driver.enterText('3');
  await driver.tap(find.byValueKey('repsField'));
  await driver.enterText('10');
  await driver.tap(find.byValueKey('submitExerciseButton'));
  await driver.waitFor(find.byValueKey('editWorkoutView'), timeout: Duration(seconds: 3));
});
```

In the *test_driver* folder is a file called *app_test.dart* where the Integration tests are performed. The figure above shows a couple of tests that make up a part of a large file with a series of tests. If you follow the logic of the test you can see that the driver is interacting with the application as a user would, tapping on the UI and entering text. The console output, however, is not as exciting.

Figure 36.app_test.dart console output

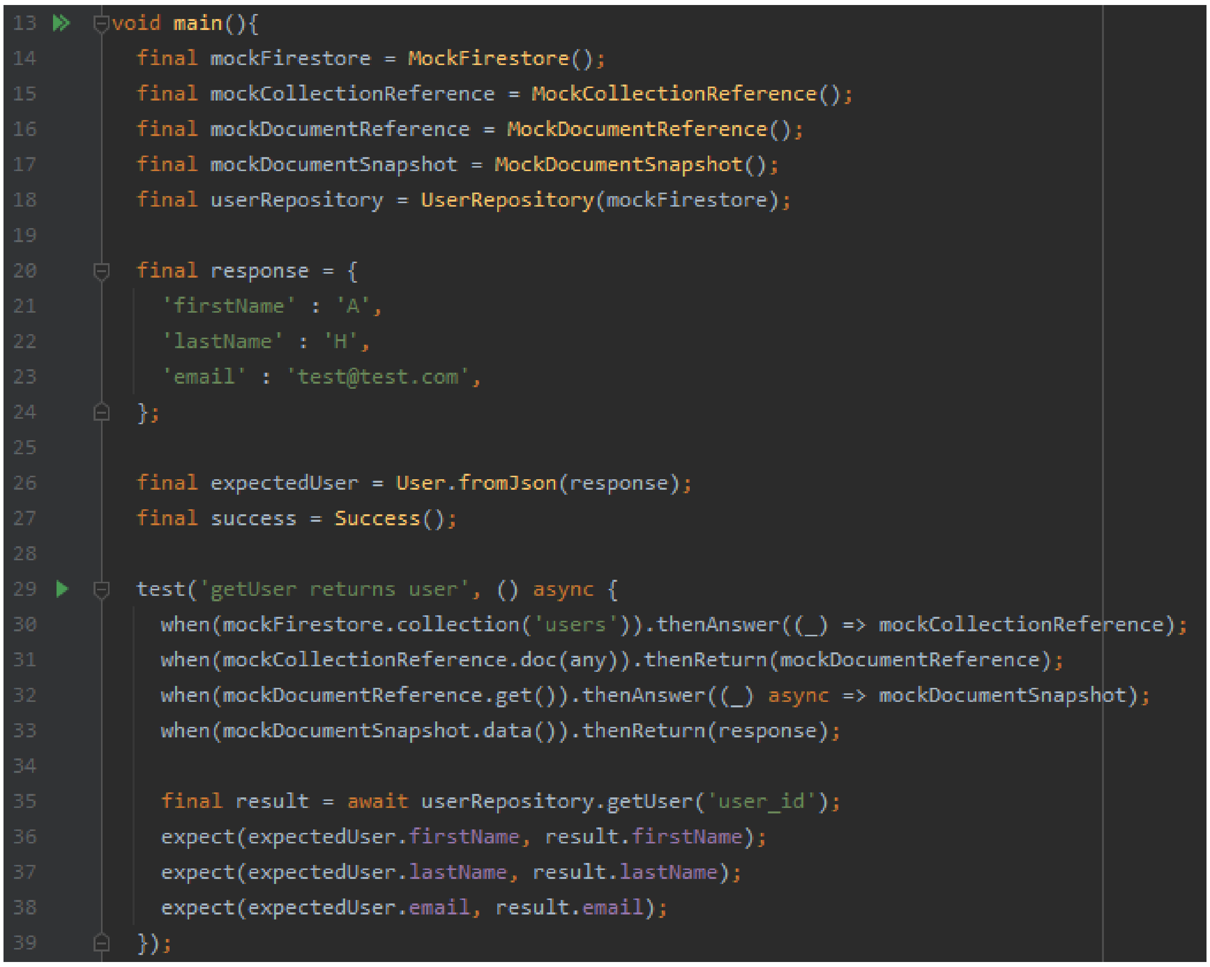


```
00:21 +17: WorkAround App sign out

D/FirebaseAuth(13585): Notifying id token listeners about a sign-out event.
D/FirebaseAuth(13585): Notifying auth state listeners about a sign-out event.
00:21 +18: WorkAround App Login

W/System  (13585): Ignoring header X-Firebase-Locale because its value was null.
W/System  (13585): Ignoring header X-Firebase-Locale because its value was null.
D/FirebaseAuth(13585): Notifying id token listeners about user ( QTm2fbFi7dR0M2ruatZBpsjsiUs2 ).
D/FirebaseAuth(13585): Notifying auth state listeners about user ( QTm2fbFi7dR0M2ruatZBpsjsiUs2 ).
00:23 +19: WorkAround App (tearDownAll)

00:23 +19: All tests passed!
```

Mocking

Mocking was used in a number of repository tests, where I wanted to test the functionality of the calls to the applications Firestore database. When testing the repositories, we do not actually want to call our real database in case we accidentally mess up real data or add test data to the live database. So, in the setting up of the testing environment, we initialise our repository object but pass in the mock database. Using Mockito, we specify the behaviour to return the data we are expecting when the mock database is called.

Figure 37. user_repository_tests.dart



```dart
13  void main(){
14    final mockFirestore = MockFirestore();
15    final mockCollectionReference = MockCollectionReference();
16    final mockDocumentReference = MockDocumentReference();
17    final mockDocumentSnapshot = MockDocumentSnapshot();
18    final userRepository = UserRepository(mockFirestore);
19
20    final response = {
21      'firstName' : 'A',
22      'lastName' : 'H',
23      'email' : 'test@test.com',
24    };
25
26    final expectedUser = User.fromJson(response);
27    final success = Success();
28
29    test('getUser returns user', () async {
30      when(mockFirestore.collection('users')).thenAnswer((_) => mockCollectionReference);
31      when(mockCollectionReference.doc(any)).thenReturn(mockDocumentReference);
32      when(mockDocumentReference.get()).thenAnswer((_) async => mockDocumentSnapshot);
33      when(mockDocumentSnapshot.data()).thenReturn(response);
34
35      final result = await userRepository.getUser('user_id');
36      expect(expectedUser.firstName, result.firstName);
37      expect(expectedUser.lastName, result.lastName);
38      expect(expectedUser.email, result.email);
39    });
```

In the above figure, on lines 14 through 18, we define our mock objects and create the *UserRepository* with the mocked Firestore database. In the *"getUser returns user"* test on lines 30 through 33, we specify the behaviour that we are expecting the series of functions to take and make sure that we return what we are expecting. When *UserRepository.getUser()* is called on line 35, it makes a series of function calls, similar to what we have specified but if something were to happen to break the chain, like the logic behind the function being changed, this test would fail.

2.6. Evaluation

WorkAround was evaluated at every major interval of the project, when a major function was implemented, or the app's design was changed. Volunteers were asked to perform a series of tests on the application.

A Think Aloud test is a test where the participant is asked to use a system while thinking out loud and provides a range of advantages. They are flexible, providing value in every stage of development and will usually lead to good results. A Think Aloud Test requires nothing but a user and a pen and paper to make notes as they talk. (Nielson, 2012)

In my Think Aloud tests, users were asked to perform a number of tasks on the application while verbalising their thoughts, providing invaluable insight and the perspective of a User. No personal identifiable information was ever recorded during any of these sessions and each user explicitly consented to participating.

For the final iteration of the application before submission, two users volunteered to perform one last evaluation, each performing a Think Aloud test with the same tasks, providing value even at this late stage of the project. The list of tasks and the results are available in the table provided in the appendix of this document.

Link to Think Aloud results.

3.0 Conclusions

WorkAround is the dynamic workout application that provides user created custom workouts, they can add as many exercises as they want to a workout, specifying the desired sets, reps and weight, from a collection of over 120 exercises so far. All of these exercises come with a video demonstration and instructions provided by MuscleWiki.com in accordance with their copyright. Users may see their performance on past workouts, including the sets that they completed and the reps they did at the weight they desired. Users can edit their workouts with more exercises or change the number of sets, reps and weight on a current exercise. The user may perform a workout with displayable demonstrations and instructions and make notes on a specific exercise to view for the next workout. All of the user's workout information is stored on a Firebase Firestore, making their workouts, the exercises within them, history and notes persist across multiple devices, they need only to log in!

By following a set of functional requirements outline at the beginning of the project and described in section 2 of this report, I was able to provide a quality app with numerable features. Following the design and guidelines of the Stacked Architecture gave me valuable experience in creating a consistent and maintainable codebase while adhering to good programming principles in the form of its View – View Model – Service implementation. I have discussed some notable aspects of the codebase in the implementation section of the report, highlighting components that were key in the design and success of the project as a whole. The User Interface is kept clean, simple and consistent throughout the application, so the user is not overwhelmed at any given point. This is partially thanks to the Flutter framework, which at the beginning of the project, I took part in over 24 hours of video lessons, letting me utilise better, its faster development features like Hot Reload and its widget based UI structure.

The core concept of WorkAround is that it provides the user with a dynamically adjusting workout that updates as they perform. It provides this through a strong, consistent and maintainable codebase that follows good programming principles, its strength comes from its use of the Firebase Firestore database to provide its users with persistent data across multiple devices, while not sacrificing performance speed. However, since the app is largely online, the apps performance may be impacted somewhat by bad internet signals, a solution of which would be looked into upon further development.

4.0 Further Development or Research

Not every exercise in the world has been added to the workout, it consists of over 120 unique exercises, but more can be added with ease. Future development of WorkAround could see hundreds more exercises added.

Google Admob was identified as a source of revenue for WorkAround as a product at the start of the project, should further development see the release onto the iOS or Android App stores, Admob could be implemented.

Recording and providing our own exercise demonstrations and instructions to move away from the MuscleWiki content.

The application performs well, however, when a mobile or device may have a bad Wi-Fi signal, it might not perform as well as it could. In the future I would implement an offline feature, so the user can use the app in areas with spotty internet connections.

5.0 Bibliography

ConciseSoftware, 2019. What is Flutter? Here is everything you should know. [online] Medium. Available at: <https://medium.com/@concisesoftware/what-is-flutter-here-is-everything-you-should-know-faed3836253f> [Accessed 1 May 2021].

CodeCarbon, 2020. Major advantages and disadvantages of Dart language | Code Carbon. [online] Code Carbon. Available at: <https://codecarbon.com/pros-cons-dart-language/> [Accessed 2 May 2021].

Oda, C., 2021. *5 Advantages of Dart Over JavaScript, Python, and Bash - DZone Web Dev*. [online] dzone.com. Available at: <https://dzone.com/articles/5-advantages-of-dart-over-javascript-python-and-ba> [Accessed 2 May 2021].

Sande, J., 2020. *State Management With Provider*. [online] raywenderlich.com. Available at: <https://www.raywenderlich.com/6373413-state-management-with-provider> [Accessed 5 May 2021].

AltexSoft, 2019. The Good and the Bad of Firebase Backend Services. [online] AltexSoft. Available at: <https://www.altexsoft.com/blog/firebase-review-pros-cons-alternatives/> [Accessed 5 May 2021].

FilledStacks, 2021. stacked | Flutter Package. [online] Dart packages. Available at: <https://pub.dev/packages/stacked> [Accessed 9 May 2021].

MuscleWiki, 2021. Musclewiki. [online] MuscleWiki. Available at: <https://musclewiki.com/Copyright> [Accessed 13 May 2021].

Nielson, J., 2012. Thinking Aloud: The #1 Usability Tool. [online] Nielsen Norman Group. Available at: <https://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/> [Accessed 7 May 2021].

6.0 Appendices

6.1. Reflective Journals

6.1.1. October

There is not a whole lot to reflect on this month but there was one big difficulty I was having with my software project at this point, actually considering what project I was going to do from my internship, I had an idea of the technologies and techniques I wanted to use, however, I struggled to nail an actual topic. I wanted to build a distributed system using industry technologies such as Apache Kafka and Docker, as these were the technologies I experienced over my work placement, I was just finding it hard to think of something to do. In the end I was able to come up with the idea of an online book store/e-book reader designed with a microservice architecture. The idea has a lot of merits and the ability to scale in regard to complexity, so I built my pitch around this and scheduled a meeting with my supervisor. I am eager to hear what my supervisor has to say and the feedback she will have, as I am very open to ideas at this early stage of the project.

6.1.2. November

In my last journal, I mentioned that I wanted to create a project based on the technologies I was exposed to during my internship, however, in the end I wasn't able to come up with a project with enough innovativeness to satisfy the project criteria with the technologies I had in mind. For what it's worth, I feel like this worked out in the end, as the project I ended up working on, a gym app that dynamically adjusts your workout as you go, provides a lot of opportunity to provide a innovative product while learning a lot of new technology. For instance, I will be using the Dart language and Flutter framework to create the app, both technologies being new to me, I decided to take an online course/bootcamp on appbrewery.com that teaches me the Flutter/Dart fundamentals. Once I have completed it, it should be easy to create a prototype for the mid-point presentation near the end of December.

6.1.3. December

I was a little stressed for time over the holiday crunch, having the TABA's instead of exams meant that we have quite a few more deliverables across all modules this year but I feel like I got the prototype for my software project into a somewhat decent state, however, it goes without saying that there will be refactoring in the coming months. Initially I praised working from home, though I guess I didn't adjust as quickly as I initially thought, the bootcamp I participated in to learn flutter took longer than expected to go through the 24 hours of video content and deadlines got quite tight by the end.

That being said, I know what to expect now, it became easier to manage my time as I got used to working from home, next semester I don't imagine being as intense near the end as this one was.

6.1.4. January

Unfortunately, it took me a while to get the motivation to work after the holidays but by setting aside time, little by little, I was getting back into the swing of things. I had several interviews to prep for, which may have affected my motivation as well. However, I was able to start working on my app and refactor the project into the architecture I wanted, compared to the state of the app I made for the purpose of having something to show for the mid-point presentation, I consider this a one step back to take two steps forward type of approach, as I traded some progress to gain some later in my project. Since refactoring, I have clearer goals in mind for the next few weeks of the project.

6.1.5. February

The workload began to pack on after classes started in full swing, balancing the work between the project and all the work from the other modules was quite important and naturally development slowed down a little bit on the app as I got used to juggling all of the work. After refactoring the project from the state it was in for the mid-point submission to the Stacked architecture I was aiming for, I also refactored the much of the Firebase authentication and began working on implementing the algorithm that is responsible for the app's gimmick, dynamically generating a workout.

6.1.6. March

A lot of development time was devoted to the implementation of the app's main algorithms. Having implemented a working prototype version, I moved to providing data persistence with Google Firestore, designing a database with that will hold User, Workout, Exercise and ExerciseSet entities. However, implementing Firestore took a lot longer for me to get my head around than initially expected. Designing a database with sound relationships between the entities, while also trying to learn the code behind implementation the functions for creating, reading, updating and deleting documents got difficult, even to a point where I questioned the viability and began to worry about finding other ways to perform these functionalities.

Fortunately, concepts began to click and I was happy to be back on track!

6.1.7. April

April was a month of a lot of submissions, and while I worked as much on the project as I could, a lot of time was redirected to more immediate deliverables. The extended submission deadlines from the IT outage provided some respite but there was still a lot of pressure as I was hard at work with the projects from the Cloud Computing, Usability Design as well as preparing for the Distributed Systems TABA which took place on May 1st.

With them out of the way, I had nothing more to distract myself and was able to begin devoting all of my attention to my final year project, which at this stage, is all starting to come together.

Along with working full time on my final year project, I had been interviewing for graduate positions and dedicated some time to preparing for these. This worked out well as I received an offer letter for one of these roles!

6.2 Project Proposal

National College of Ireland

Project Proposal

<WorkAround – A Dynamic Workout App>

<08/11/2020>

<BSHCSD4>

<Software Development>

<Academic Year 2020/2021>

<Aaron Hynes>

<x17742385>

<x17742385@student.ncirl.ie>

6.2.1. Objectives

WorkAround is a workout app for the casual gym-goer or someone who wants to start exercising despite their busy schedule, who may not always have a lot of time for a gym or workout session but wants to put what little time they do have into an activity or routine of their choice. It is an app that can generate a workout based on the equipment the user has and what time they need to finish, the app will then dynamically adjust the workout as you go.

Main objectives of my application are:

- To generate a workout based on body part, equipment or amount of time the user has.
- Create/ edit and delete their own routines instead of generating one if they should choose to.
- Perform a workout, i.e. enter how many reps/sets they did. Add notes and past performance.
- Dynamically adjust the workout routine based on the users pace and remaining time.
- Ensure the user will be able to complete their workout within their designated timeframe
- See additional information about exercises or how to do them
- Provide extra exercises for mobility reduced persons. E.g. an elderly person who wishes to do a little activity while sitting.
- Create and edit account information.
- To feature cloud backups so the user my sync data across devices.

6.2.2. Background

Aaron is working from home and had a quick bite to eat when he realises that he still has 45 minutes of his lunch break left, he decides he wants to do a home workout to keep active and pulls out his usual gym/workout app and sets about doing his regular routine. Most of the way through his routine, he's realised that he didn't give it enough time, maybe he rested too long between each workout or the routine was supposed to take 60 minutes instead and is faced with the decision to stop the workout prematurely, being late going back to work, or worse, injuring himself making up for lost time.

This is where WorkAround will stand out, the app will dynamically adjust the workout for the user depending on their pace and remaining time. For example, instead of three sets of ten, it may adjust it to two sets to make sure you meet the deadline, this way the workout doesn't have to end prematurely, every exercise gets done and Aaron isn't late to work after lunch.

After some initial market research, I have not found anything that provides this feature and thus I believe that this qualifies as an innovation I am very glad to work on.

Currently the market is saturated by apps that give you set routines with no regards to time or try to include faff like meal plans that can really overwhelm a casual or beginner gymgoer. I've found other apps that do take into consideration shorter workouts but they only offer static routines that may or may not take you 20 minutes or half an hour.

This is where WorkAround sets itself apart, by providing a 20-minute workout routine that will take 20 minutes. The name of the project stemming from the phrase "workaround" meaning to overcome a limitation, that limitation being the amount of time the user has to Work(out) around.

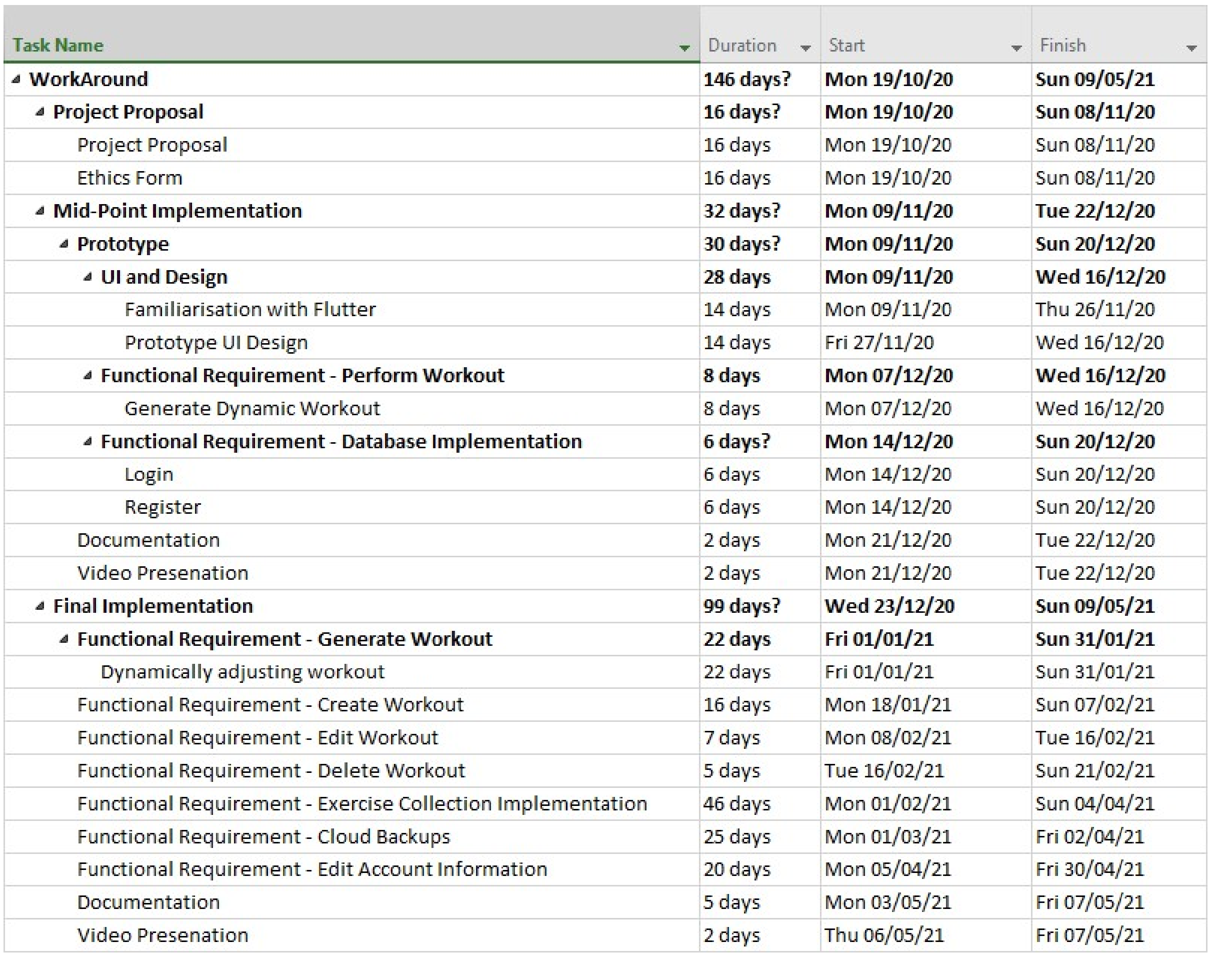
6.2.3. Technical Approach

I will begin by scoping out the minimum viable product features of the application, deciding what aspects of the app I want to work towards first. This means I will focus on releasing a version of the product that has just enough features to be evaluated by the client I am working with, who will then provide feedback for future development. Should the scope of time permit, I will work towards stretch goals, prioritising deliverables as I work towards the project deadline. Over the course of the project, I will be meeting with my client regularly so they can evaluate the progress of the application I am making for them.

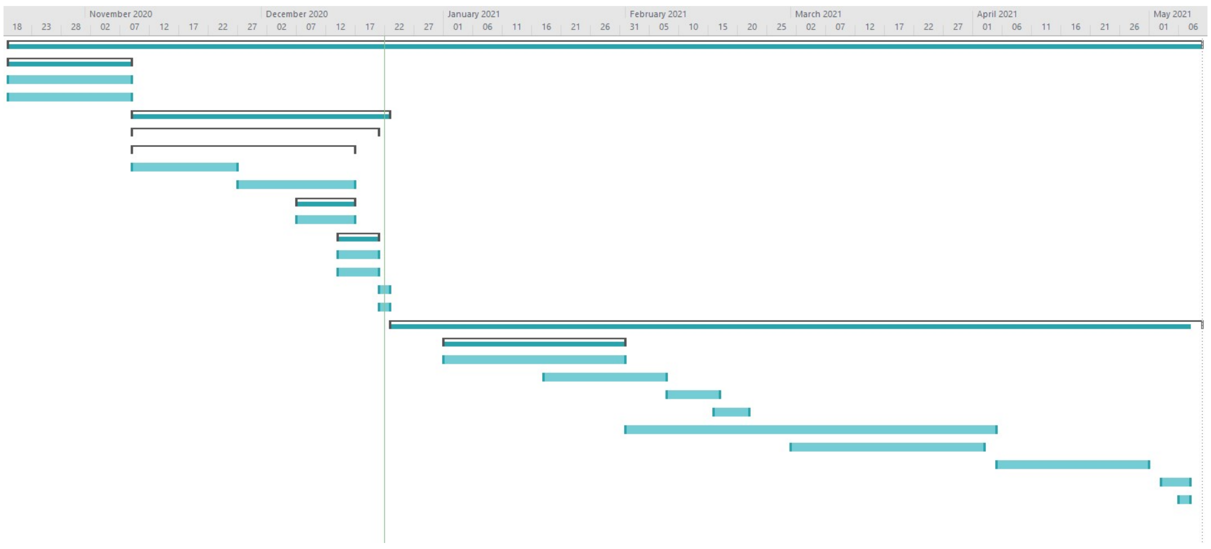
6.2.4. Special Resources Required

This application requires no special resources.

6.2.5. Project Plan



Task Name	Duration	Start	Finish
WorkAround	**146 days?**	**Mon 19/10/20**	**Sun 09/05/21**
Project Proposal	**16 days?**	**Mon 19/10/20**	**Sun 08/11/20**
Project Proposal	16 days	Mon 19/10/20	Sun 08/11/20
Ethics Form	16 days	Mon 19/10/20	Sun 08/11/20
Mid-Point Implementation	**32 days?**	**Mon 09/11/20**	**Tue 22/12/20**
Prototype	**30 days?**	**Mon 09/11/20**	**Sun 20/12/20**
UI and Design	**28 days**	**Mon 09/11/20**	**Wed 16/12/20**
Familiarisation with Flutter	14 days	Mon 09/11/20	Thu 26/11/20
Prototype UI Design	14 days	Fri 27/11/20	Wed 16/12/20
Functional Requirement - Perform Workout	**8 days**	**Mon 07/12/20**	**Wed 16/12/20**
Generate Dynamic Workout	8 days	Mon 07/12/20	Wed 16/12/20
Functional Requirement - Database Implementation	**6 days?**	**Mon 14/12/20**	**Sun 20/12/20**
Login	6 days	Mon 14/12/20	Sun 20/12/20
Register	6 days	Mon 14/12/20	Sun 20/12/20
Documentation	2 days	Mon 21/12/20	Tue 22/12/20
Video Presenation	2 days	Mon 21/12/20	Tue 22/12/20
Final Implementation	**99 days?**	**Wed 23/12/20**	**Sun 09/05/21**
Functional Requirement - Generate Workout	**22 days**	**Fri 01/01/21**	**Sun 31/01/21**
Dynamically adjusting workout	22 days	Fri 01/01/21	Sun 31/01/21
Functional Requirement - Create Workout	16 days	Mon 18/01/21	Sun 07/02/21
Functional Requirement - Edit Workout	7 days	Mon 08/02/21	Tue 16/02/21
Functional Requirement - Delete Workout	5 days	Tue 16/02/21	Sun 21/02/21
Functional Requirement - Exercise Collection Implementation	46 days	Mon 01/02/21	Sun 04/04/21
Functional Requirement - Cloud Backups	25 days	Mon 01/03/21	Fri 02/04/21
Functional Requirement - Edit Account Information	20 days	Mon 05/04/21	Fri 30/04/21
Documentation	5 days	Mon 03/05/21	Fri 07/05/21
Video Presenation	2 days	Thu 06/05/21	Fri 07/05/21



6.2.6. Technical Details

The application will be created using an open-source UI SDK called Flutter, that was created by Google and it is used to develop applications across multiple platforms. It is a framework for developing native applications for many different platforms using the Dart virtual machine, it allows features like allowing modifications to files that can be injected into a running application.

On top of Flutter and Dart, the project will see the implementation of Firebase, a Backend-as-a-Service and offers many functionalities like databases, analytics and crash reporting. In particular, libraries to be used include Cloud Firestore, Firebase Authentication and Admob.

Cloud Firestore offers flexible and scalable databases for mobile and web development, Firebase Authentication provides easy to use SDK's that provide back-end services and ready-made UI libraries for user authentication and Admob, which is a way to monetize an android app using the Google Mobile Ads SDK.

6.2.7. Evaluation

System testing will be performed through the implementation of both unit and integration testing using fake/simulated personal data. On top of this, I will have consistent feedback from the client I will be creating the application for, who will inform me of their requirements throughout the course of the project as well as receiving feedback and evaluation from a number of end users.

As such, I will be submitting an Ethics application form.

6.2.8. Invention Disclosure Form

Please fill in the following sections, if you think your idea is innovative:

1. Title of Invention

WorkAround – A dynamic workout app.

2. Inventors

Name	School/Research Institute	Affiliation with Institute (i.e. department, student, staff, visitor)	Address, contact phone no., e-mail	% Contribution to the Invention
Aaron Hynes	National of College Ireland	Student	97 Mountain View Drive, Rathfarnham, D14HW25	90%
Niamh Gohil			44 Moore Road, London	10%

3. Contribution to the Invention

Each contributor/potential inventor should write a paragraph relating to his/her contribution and include a signature and date at the end of the paragraph.

Aaron Hynes – Developer of the application

Aaron Hynes

Niamh Gohil – Client who commissioned the development of the application

[illegible]

4. Description of Invention

(Please highlight the novelty/patentable aspect. Attach extra sheets if necessary including diagrams where appropriate). What is novel, the 'inventive step'? For more information on patents, please look at http://www.patentsoffice.ie/en/patents.aspx

WorkAround is a workout app for the casual gym-goer or someone who wants to start exercising despite their busy schedule, who may not always have a lot of time for a gym or workout session but wants to put what little time they do have into an activity or routine of their choice. It is an app that can generate a workout based on the equipment the user has and what time they need to finish, the app will then dynamically adjust the workout as you go, which is what I believe innovative.

5. Why is this invention more advantageous than present technology?

What is its novel or unusual features? What problems does it solve? What are the problems associated with these technologies, products or processes? Explain how this invention overcomes these problems (*i.e.* what are its advantages).

Currently the market is saturated by apps that give you set routines with no regards to time or try to include faff like meal plans that can really overwhelm a casual or beginner gymgoer. I've found other apps that do take into consideration shorter workouts but they only offer static routines that may or may not take you 20 minutes or half an hour.

This is where WorkAround sets itself apart, by providing a 20-minute workout routine that will take 20 minutes. The name of the project stemming from the phrase "workaround" meaning to overcome a limitation, that limitation being the amount of time the user has to Work(out) around.

6. What is the current stage of development / testing of the invention?

4th year project

7. List the names of companies which you think would be interested in using, developing or marketing this invention

Techil ltd.

8. Funding Partner(s)

Government Agency & Department	
% Support	
Contract/Grant No.	
Contact Name	
Phone No.	

Address	

Industry or other Sponsor	
% Support	
Contract/Grant No.	
Contact Name	
Phone No.	
Address	

9. Where was the research carried out?

Home/independent research

10. What is the potential commercial application of this invention?

11. Was there transfer of any materials/information to or from other institutions regarding this invention?

If so please give details and provide signed agreements where relevant.

No

12. Have any third parties any rights to this invention?

If yes, give names and addresses and a brief explanation of involvement.

Niamh Gohil is the client in which I am creating the app for.

44 Moore Road, London

13. Are there any existing or planned disclosures regarding this invention?

Please give details.

Ideally, the property will be released to Aaron Hynes and Niamh Gohil

14. Has any patent application been made? Yes/[✓ No]

If yes, give date: ________________ Application No.: ________________

Name of patent agent: ____

Please supply copy of specification.

15. Is a model or prototype available? Has the invention been demonstrated practically?

No

I/we acknowledge that I/we have read, understood and agree with this form and the Institute's *Intellectual Property and Procedures* and that all the information provided in this disclosure is complete and correct.

I/we shall take all reasonable precautions to protect the integrity and confidentiality of the IP in question.

Inventor: Aaron Hynes 08/11/20 Aaron Hynes

Inventor: Niamh Gohil 08/11/20 [illegible]

6.3 Final Think Aloud Test

Task	Participant 1	Participant 2
Register an account.	Registered an account with no issues. Liked that there was a button to redirect to the login page.	Registered an account with no issues, but commented on an annoying interaction between the mobile device's on screen keyboard and the app.
Create a workout.	Created a workout with no issues, praising the exercise catalogue.	Create a workout in a reasonable amount of time. User mistakenly tapped on the "Back" button, cancelling process and had to start over. Approved of the large list of exercises to choose from.
Edit a workout.	User edited a workout with no issues.	User edited a workout with no issues.
View Exercise Information.	Browsed and viewed the exercise catalogue with no issues.	Browsed and viewed the exercise catalogue with no issues.
Start a workout	Started a workout with no issues.	Started a workout almost immediately.
Make a note in a workout.	Made a note in the workout with no issues, however, they commented on the note button not being very clear.	Made a note in the workout with no issue, took a second to find and tap the note button.

Link back to report.