Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Automation of Applied Informatics

Jena Woodroffe

DEVELOPMENT OF FLUTTER APPLICATIONS FOR PHYSIOTHERAPISTS

SUPERVISOR

David Sik

BUDAPEST, 2025

Contents

STUDENT DECLARATION

I, **Jena Woodroffe**, the undersigned, hereby declare that the present BSc thesis work has been prepared by myself and without any unauthorized help or assistance. Only the specified sources (references, tools, etc.) were used. All parts taken from other sources word by word, or after rephrasing but with identical meaning, were unambiguously identified with explicit reference to the sources utilized.

I authorize the Faculty of Electrical Engineering and Informatics of the Budapest University of Technology and Economics to publish the principal data of the thesis work (author's name, title, abstracts in English and in a second language, year of preparation, supervisor's name, etc.) in a searchable, public, electronic and online database and to publish the full text of the thesis work on the internal network of the university (this may include access by authenticated outside users). I declare that the submitted hardcopy of the thesis work and its electronic version are identical.

Full text of thesis works classified upon the decision of the Dean will be published after a period of three years.

Budapest, 21 January 2025

...…….
Jena Woodroffe

Abstract

With the rise of mobile technology, many medical fields have made a gradual but major shift towards integrating mobile devices and applications to enhance patient care and clinic management. Some fields, such as physiotherapy, have taken longer to create comprehensive and satisfactory options that meet the diverse needs of smaller clinics. Existing tools are either too focused on exercise management and fall short in terms of other functions or are prohibitively expensive.

PhysioHub and PhysioLink are Flutter-based cross platform applications designed to bridge this gap. PhysioHub provides physiotherapists with tools for managing patient records, tracking appointments, prescribing exercises and maintaining notes. Complementing this application, PhysioLink provides patients with access to their assigned exercises, appointment schedules and a direct line of communication with their therapist.

These applications streamline the management of physiotherapy clinics and ensure better maintenance and accessibility of data, as well as promoting patient adherence to treatment plans.

In conclusion, PhysioHub and PhysioLink demonstrate the potential for tailored, affordable solutions to improve physiotherapy practices.

Absztrakt

A mobiltechnológia térhódításával számos orvosi terület fokozatosan, de nagymértékben áttért a mobileszközök és alkalmazások integrálására a betegellátás és a klinika irányításának javítása érdekében. Néhány területen, például a fizioterápiában, hosszabb időbe telt, amíg olyan átfogó és kielégítő lehetőségeket hoztak létre, amelyek megfelelnek a kisebb klinikák különböző igényeinek. A meglévő eszközök vagy túlságosan a gyakorlatok irányítására összpontosítanak, és más funkciók tekintetében elmaradnak, vagy pedig megfizethetetlenül drágák.

A PhysioHub és a PhysioLink olyan Flutter-alapú, keresztplatformos alkalmazások, amelyek ezt a szakadékot hivatottak áthidalni. A PhysioHub eszközöket biztosít a fizioterapeuták számára a betegnyilvántartások kezeléséhez, a találkozók nyomon követéséhez, a gyakorlatok felírásához és a feljegyzések vezetéséhez. Ezt az alkalmazást kiegészítve a PhysioLink hozzáférést biztosít a páciensek számára a kijelölt gyakorlatokhoz, az időpontok ütemezéséhez és a terapeutával való közvetlen kommunikációhoz.

Ezek az alkalmazások racionalizálják a fizioterápiás rendelők irányítását, és biztosítják az adatok jobb karbantartását és hozzáférhetőségét, valamint elősegítik a betegek kezelési tervekhez való ragaszkodását.

Összefoglalva, a PhysioHub és a PhysioLink megmutatja a testre szabott, megfizethető megoldások lehetőségét a fizioterápiás gyakorlatok javítására.

1 Introduction

This paper discusses the development of two Flutter-based applications, PhysioHub and PhysioLink, designed to improve communication and streamline operations for small physiotherapist clinics and their patients.

There are several inefficiencies in how many smaller physiotherapy clinics manage communication and patient care due to a lack of comprehensive or accessible organisational software. Common alternatives to the functionalities provided by a central organisation system include communication through instant messaging platforms, which damages the boundaries between professional and personal communication, and hardcopy exercise sheets and patient notes which can easily be lost or damaged. These inefficiencies can stunt patient progress and create difficult working environments for physiotherapists.

These challenges inspired the creation of PhysioHub and PhysioLink. PhysioHub provides physiotherapists with a dedicated platform for managing communication, appointments, and patient data—all in one place. Notifications can be turned off outside work hours, offering therapists greater control over their time and work-life balance. PhysioLink, designed for patients, ensures secure access to digital copies of their exercise plans, complete with detailed instructions and video demonstrations. This eliminates the need for physical handouts and provides patients with a convenient, reliable way to follow their treatment plans.

PhysioHub and PhysioLink address common pain points in physiotherapy practice with innovative, user-friendly solutions. By centralizing communication and digitizing patient resources, these applications aim to improve both the efficiency of clinic operations and the quality of care provided to patients.

1.1 Structure of the Thesis

The project documentation is divided into nine chapters to provide a thorough analysis of how these applications were researched and implemented.

The first chapter includes an initial literature review of medical health applications and analyses some of the existing solutions available on the market for physiotherapists, concluding with the motivations behind choosing this project topic.

The second chapter provides a detailed description of what the applications are expected to do, along with a comprehensive use case diagram showing the complete range of functionalities provided by the different applications.

The third chapter is dedicated to the research of the technological tools used to develop this project. The framework and programming language, backend services, APIs and third-party packages are included and described in detail.

Chapter four details the design and architecture aspects of the applications. The client-side and server-side architectures are analysed, and a thorough explanation of data storage decisions is provided.

The fifth chapter encompasses the implementation of PhysioHub and PhysioLink and is divided into subchapters based on different functionalities provided by these applications. Each subchapter includes screenshots from the applications to show the user interface and details the code behind implementing that functionality.

The sixth chapter is dedicated to comparing the previous Android version of PhysioHub and PhysioLink to the current Flutter versions, focusing on how asynchronous programming, state management, and UI implementation are handled in these different languages.

Chapter seven provides a summary of this project and outlines future development plans for these applications.

References for all the research done, both in the field of medical applications and the resources required for learning Dart and Flutter, are provided in the eighth and final chapter.

1.2 Significance of Technologies Used

These two applications were developed originally in Android in 2023 and early 2024 and were subsequently recreated using Dart and Flutter for this project. The growing prevalence of Dart in application development was a significant factor in the decision to learn and code in this language. Dart's clean syntax, hot reload feature and robust asynchronous programming capabilities make it an excellent choice for creating applications that required real-time communication and dynamic updates.

Flutter's ability to build cross-platform applications with a single codebase was particularly advantageous, eliminating the need for platform-specific development. This

allowed for the creation of apps that run seamlessly on both iOS and Android without requiring access to Mac products.

Recreating the applications in Dart and Flutter not only enhanced their functionality but also provided invaluable skills in cross-platform development.

1.3 Role of Healthcare Applications

The rapid expansion of mobile health applications has been instrumental in reshaping healthcare delivery, particularly in enhancing patient engagement and self-management [28]. Patient engagement refers to the 'active involvement of patients in their healthcare journey' [31], which can have a significant impact on the speed and quality of a patient's recovery, especially in the field of physiotherapy. Physiotherapy relies heavily on patients' commitment to doing their rehabilitation exercises consistently and correctly, which proves difficult to monitor between check-ups. Studies indicate that adherence rates to home exercise programs can be as low as 25% to 50%, hindering effective rehabilitation [33].

Medical health applications offer patients convenient access to health information, monitoring tools, and support for behaviour modification, thereby empowering individuals to take an active role in monitoring their health. These applications have proved especially important in assisting patients with chronic diseases [29].

The development of mobile health applications is expanding at an unprecedented rate and holds great promise, supported by evidence of positive outcomes for both patients and healthcare providers. A randomised field experiment found that patients who adopted a mobile health application for diabetes undertook more exercise, consumed healthier food and experienced reductions in blood glucose and haemoglobin levels [30]. These findings suggest that mobile health technologies have the potential to significantly enhance patient engagement and health outcomes.

However, challenges such as ensuring data privacy and security, and addressing disparities in access to technology, remain critical considerations [31].

1.4 Equivalent Solutions and Motivation

There are several examples of other applications that aim to tackle and solve similar problems to those listed above. These applications tend to fit two categories:

firstly, applications focused on exercise prescription and health tracking, and secondly, applications with a broader focus on clinic management tools and exercise prescription. Examples of applications in each category are given below, with brief discussions showing why PhysioLink and PhysioHub provide an overall better user experience and range of functions.

1.4.1 MoveHealth

MoveHealth is a free application that has advanced features for exercise prescription and patient progress tracking. The app offers a robust library of exercises with video demonstration and guides users through each step of their exercise plan, timing each exercise and providing rest periods. MoveHealth provides comprehensive results for their patients, with information on how their specific injuries are healing and how many exercise sessions they have completed in the last week.

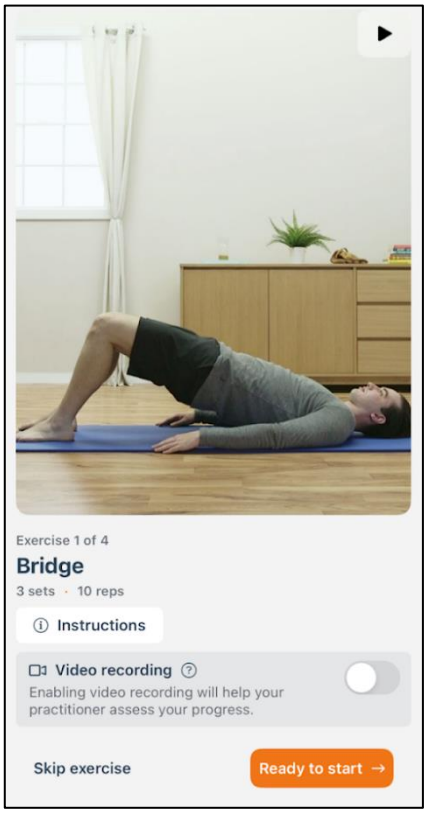

Figure 1.1 MoveHealth Example [22]

This application falls into the first category of solutions found. MoveHealth provides no additional features besides exercise prescription and tracking and cannot assist physiotherapy clinics in managing their practice. There are many applications similar to MoveHealth, each varying in aesthetics and ease of use, with some offering a

more intuitive interface and visually appealing design. However, MoveHealth stands out as one of the more aesthetically pleasing options available.

1.4.2 PhysiApp and PhysiTrack

PhysiApp and its companion application, PhysiTrack, are professional-grade healthcare tools designed to enhance physiotherapy practice. These applications fall into the second category of solutions I found, with a broader focus on clinic management alongside exercise prescription. PhysiApp serves clinicians, offering features like telehealth sessions, seamless integration with electronic health records, and tools to create and manage exercise prescriptions. Patients access their personalised exercise programs through PhysiTrack, which provides videos, instructions and progress tracking,

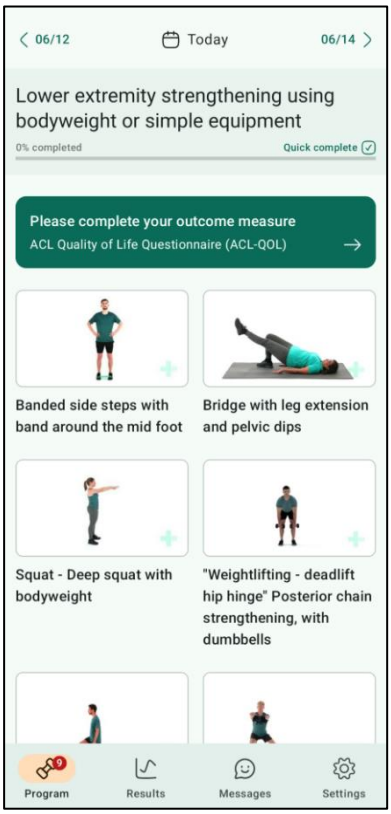

Figure 1.2 PhysiApp Example [23]

While these applications work well together, their extensive feature set can be overwhelming to both patients and clinicians, making them challenging to learn. Moreover, physiotherapists can only prescribe and edit exercise plans through the web-based PhysiApp, limiting the system's utility for mobile, on-the-go use. Additionally, while PhysiTrack allows patients to provide feedback on exercises and pain levels, it lacks direct communication features with practitioners, requiring users to rely on external methods to connect with their physiotherapist.

1.4.3 Rehab Guru

Rehab Guru provides solutions for many of the issues my applications aim to address. Used by physiotherapists, fitness professionals and rehabilitation specialists, Rehab Guru provides an extensive set of features for prescribing exercises and streamlining clinic workflow, falling clearly into the second category of solutions found. The platform has an extensive exercise library of over 6000 exercises and provides clinicians with the ability to create programmes tailored to their patients' needs. With Rehab Guru, physiotherapists can keep secure, detailed treatment notes on their patients and efficiently track patient progress. Rehab Guru also stands out due to its many clinic management features. These include an impressive diary feature with appointment scheduling and reminders, and payment handling with integrated invoicing and payment tracking. With Rehab Guru Client, patients can access their treatment exercises with demonstrational videos and give feedback on their pain and progress.

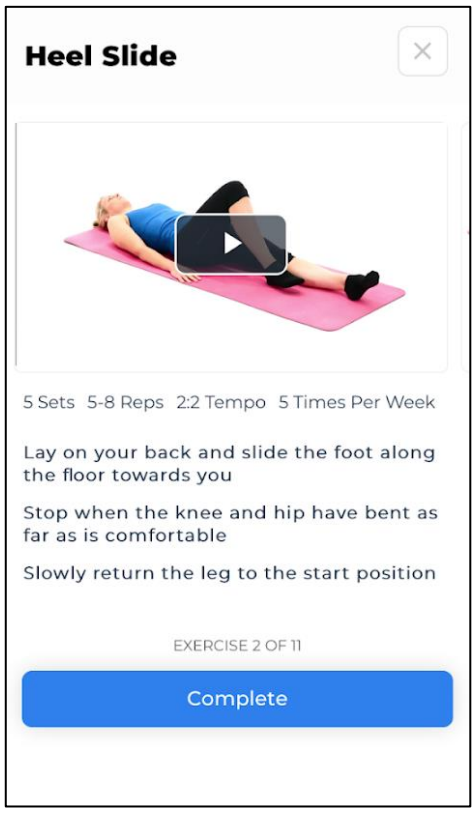

Figure 1.3 Rehab Guru Client Example [24]

However, the design of both the applications, but especially Rehab Guru Client, are rather unappealing, with little focus on a pleasant user interface. The applications are also far more expensive than alternative solutions, making them inaccessible to smaller clinics and practices. Furthermore, Rehab Guru Client also does not provide any means for direct communication with physicians.

1.5 Motivation

An analysis of the existing solutions available for physiotherapists reveals many applications with a sole focus on exercise prescription, aimed not only at physiotherapists, but also fitness professionals. These applications lacked any of the features that would be required to run a clinic. Other solutions were more expansive and provided varying clinic management capabilities, but their companion applications for clients often lacked features and finesse. Furthermore, none of the apps provided direct communication between practitioners and patients.

Recognising these gaps, PhysioHub and PhysioLink are designed to combine robust clinic management, intuitive design, and a built-in chat feature. These tools aim to address the specific needs of smaller physiotherapy clinics, offering an all-in-one solution to enhance both efficiency and patient engagement.

2 Task Specification

This chapter provides a description of the functions of each of the applications and a use case diagram.

2.1 Detailed Description

This project comprises two applications: PhysioHub and PhysioLink. While the applications share many features, PhysioHub provides physiotherapists with greater access and control of data than PhysioLink provides the patients.

When registering with PhysioHub, clinicians are required to provide relevant personal information and set up their email and password. With PhysioLink, patients must provide an invitation code, a one-time code supplied by their PhysioHub-registered physician that will link the patient's account to theirs. After inputting a valid invitation code, patients will be allowed to progress with a similar registration. If a user has previously registered and logged into the system without logging out, upon opening the application they will be automatically logged in and directed to the Home Screen.

The Home Screen is identical on both applications, displaying a time-sensitive greeting with the user's name. Below this, the user's next upcoming appointment will be displayed if available. Navigation between the Home Screen and other times is provided through the bottom navigation bar.

With PhysioHub, physicians can view a list of all their patients on the Patients Screen. Patients are displayed with their name, profile photo and a notification of how many unread messages the physician has from this patient. By clicking on a patient, the physician will be directed to the chat feature with that patient. From this screen, clinicians can view the detailed patient information and edit the patient notes, as well as access this patient's exercise prescription. From the exercise screen, the clinician can edit the exercises by deleting, updating or adding new exercises.

The Chat Screen is accessible to patients on PhysioLink through the bottom navigation, and they will only have one chat available - that being with their linked physiotherapist. This feature allows for real-time communication between patients and their doctors, displaying the time that each message is sent and if it has been read or not.

Patients can access their exercise plan through the Exercise Screen, where they can see the animation of each exercise as well as the number of sets and reps they need to complete. Clicking on an exercise will display its detailed information, including step by step instructions.

PhysioHub also comes with a calendar feature, allowing physiotherapists to quickly add new appointments to their schedule that will be immediately linked and displayed to the corresponding patient.

Finally, both applications have a Settings Screen where the users can edit their name and profile picture and log out.

2.2 Use Case Diagram

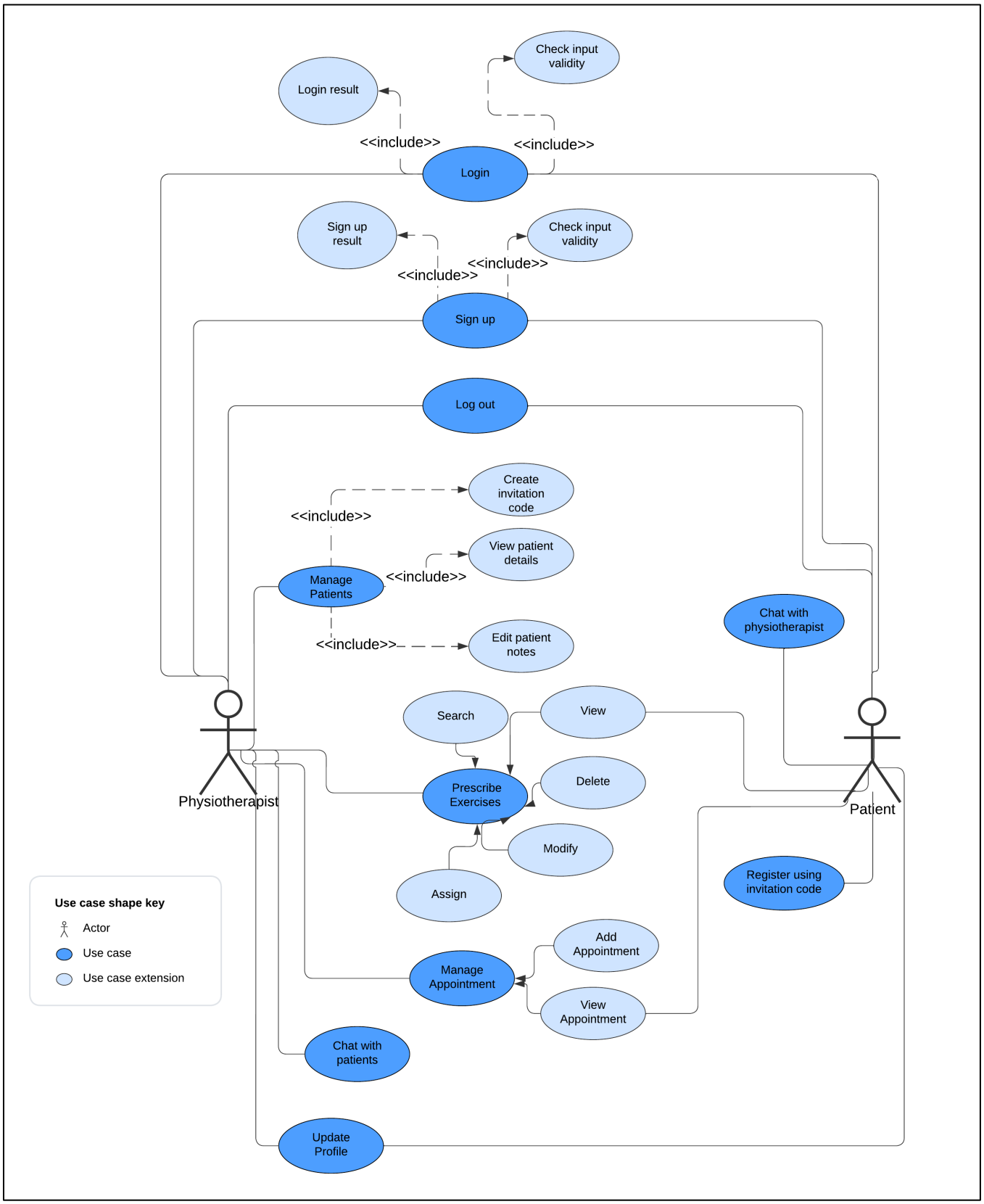

Figure 2.1 Use Case Diagram

The use-case diagram above illustrates the distinct functionalities available to the two primary actors—Physiotherapist and Patient—within the PhysioHub and PhysioLink applications, respectively. PhysioHub, designed for physiotherapists, offers comprehensive tools for managing patient information, prescribing exercises, and scheduling appointments. In contrast, PhysioLink serves as a companion application tailored for patients, providing limited but essential access to their treatment plans and communication needs.

Both applications incorporate essential features such as registration and log-in. Physiotherapists access PhysioHub by creating an account or logging in with existing credentials, while patients register through PhysioLink using an invitation code generated by their assigned physiotherapist.

Physiotherapists can manage patient records, prescribe, modify or delete exercises from patient exercise regimens, and schedule appointments. Patients can only view assigned exercises and upcoming appointments through PhysioLink. Integrated chat features enable direct communication between physiotherapists and patients.

3 Technological Overview

This chapter examines the existing literature on the various technologies used in this thesis. Flutter and Dart are respectively the primary platform and language for the applications, while Firebase was used for authentication and database management.

3.1 Flutter

Flutter, released by Google in 2017, is a cross-platform development framework specifically designed to work with the Dart programming language. It enables the creation of high-performance applications that run seamlessly across its five supported platforms: Android, iOS, Windows, macOS, Linux, and Fuchsia. By compiling to native ARM code, Flutter delivers performance comparable to native apps.

One of Flutter's standout features is its ability to launch the same codebase on both iOS and Android, significantly reducing production time for projects. The framework also offers an extensive set of pre-built, highly customizable widgets, rendered using its own graphics engine. This ensures a cohesive and consistent UI across all platforms, which can often be difficult to achieve through native app development.

Backed by Google and supported by a strong corporate ecosystem, Flutter provides long-term stability and regular updates, making it a reliable choice for modern application development.

3.2 Dart

Dart is a client-optimised programming language created to power Flutter framework. The object-oriented language is tailored for client-side development offering sub-second stateful hot reload for rapid iteration and ensuring high quality production across supported platforms [15]. The language is type safe, using static type checking and runtime checks to ensure the value of a variable always matches its type [16].

Notably, Dart has built in sound null safety, meaning that variables can only hold null values when explicitly allowed through their declaration. This offers strong protection against null-related errors during runtime.

Dart supports Ahead-of-Time (AOT) compilation, producing fast, predictable performance and smooth animations in Flutter apps [18].

3.3 Asynchronous Programming

3.3.1 Future and Stream

Dart manages asynchronous programming mainly through the use of the Future and Stream classes. Future represents a single asynchronous computation or operation that does not produce a result immediately. These objects are used with async/await to create more readable asynchronous code. Future is often used to fetch data from an API or database and for handling file IO operations.

Streams are used for sequences of asynchronous events over time. Streams can handle a continuous flow of data and can provide real-time updates. This can be utilised when creating chat features or providing live data feeds. The stream class is equipped with many helper methods, engineered to execute frequently used operations on a stream.

3.3.2 Provider

Provider is a state management solution built on *InheritedWidget* that manages UI states from asynchronous operations. It serves three key functions in asynchronous programming: *FutureProvider* handles and caches asynchronous operations and their loading states [19], *StreamProvider* integrates Stream data into the app's state management, and Consumer efficiently rebuilds widgets that depend on asynchronous data.

3.4 User Interface Development

3.4.1 Flutter Widgets

Flutter’s user interface is entirely built using widgets, which form the building blocks of the framework. Each UI element, from structural layout components like padding to the physical, interactive components such as buttons, are individual widgets. Flutter categorises its widgets into two main types:

- StatelessWidget: used to create immutable, fixed components that do not change state during the app’s lifecycle.
- StatefulWidget: designed for dynamic components that can change their state or appearance based on user interaction or data updates [20].

These widgets can be nested to create complex UI elements while maintaining flexibility and simplicity. For further customisation, Flutter allows developers to create custom, reusable widgets that combine and encapsulate more complex UI elements. This ensures easier maintainability of the code and a more consistent interface design throughout the application.

3.4.2 Material Design

Material Design 3, Google's design system, ensures consistent styling and visuals across platforms and serves as Flutter's default design language. Material Widgets are pre-built components that follow Material Design guidelines [21], which emphasise usability, consistency and a visually harmonious experience. These widgets are essential for building Flutter UIs and users trust their familiar aesthetics and functionality. Common examples include *AppBar*, *FloatingActionButton*, *SnackBar*, and *AlertDialog*.

3.5 Firebase

Firebase, a subset of Google Cloud services, is a Backend-as-a-Service app development platform. It offers hosted backend services that easily integrate into mobile and web applications. With features like Authentication, Realtime Databases, Cloud Messaging, Crashlytics, and Performance Monitoring, Firebase can fulfil most backend requirements for app development.

Firebase services prioritize security by encrypting data both in transit and at rest. They also allow for extensive security rules on databases, ensuring that only authenticated users can access or write data.

3.5.1 Authentication

Firebase Authentication simplifies secure log-in creation with diverse methods. Users can register using email, phone number, Google, Facebook, and more. Firebase Authentication is made to seamlessly integrate with the rest of the Firebase services, such as Firestore and Storage to simplify access management and data synchronisation. Firebase Authentication boasts impressive data security, adhering to OAuth 2.0 - a robust protocol for token-based authentication - and encrypting all data between the client and the Firebase servers using Transport Layer Security (TLS). Firebase Authentication also offers multi-factor authentication under their upgraded payment plan to further improve security concerns, which can easily be included in this project as it grows.

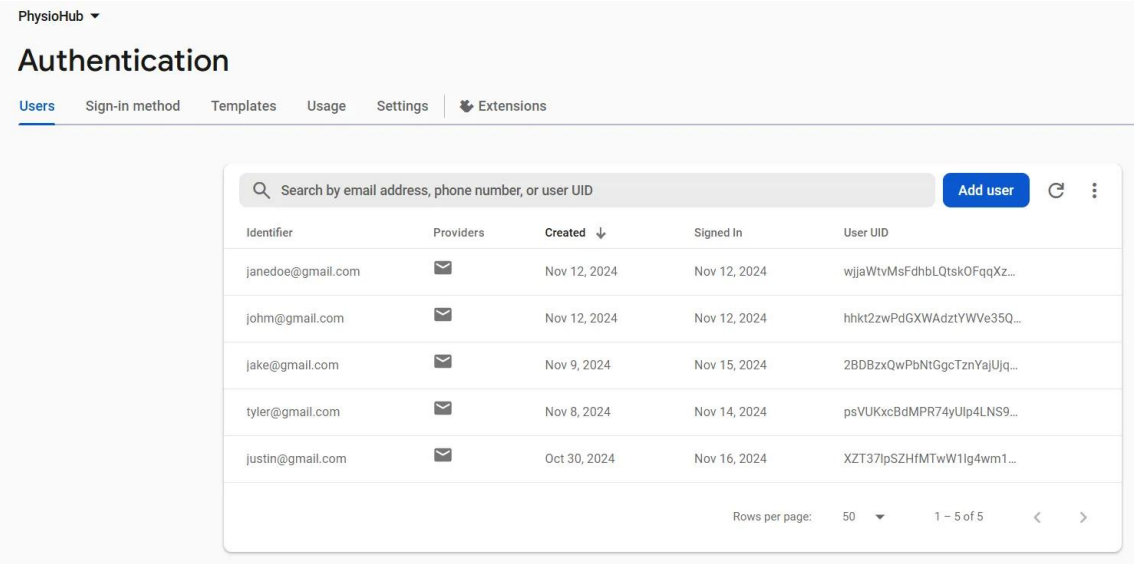

Figure 3.1 Authentication in PhysioHub and PhysioLink

3.5.2 Cloud Firestore

Cloud Firestore, built on Google Cloud infrastructure, offers flexible and scalable NoSQL cloud database solutions. Unlike traditional databases with rows and columns, Firestore organizes data into collections of documents, enabling highly flexible, hierarchical data structures. These documents can contain complex, nested objects and subcollections.

Firestore's expressive querying allows users to retrieve individual documents or all documents within a collection that match specific query parameters. These queries are indexed by default, ensuring query performance is proportional to the result set size rather than the entire dataset. This feature guarantees quick data transactions and minimal application delays.

Real-time data updates are a key feature of Cloud Firestore. Through data synchronization, it updates information on all connected devices without constant database querying. This capability was crucial for implementing real-time messaging between applications.

Firestore databases are secured through highly adaptable security rules built with the foundation of Firebase Authentication. These rules can verify if a request comes from an authenticated user and employ Role-Based Access Control to check a user's role and enforce access policies.

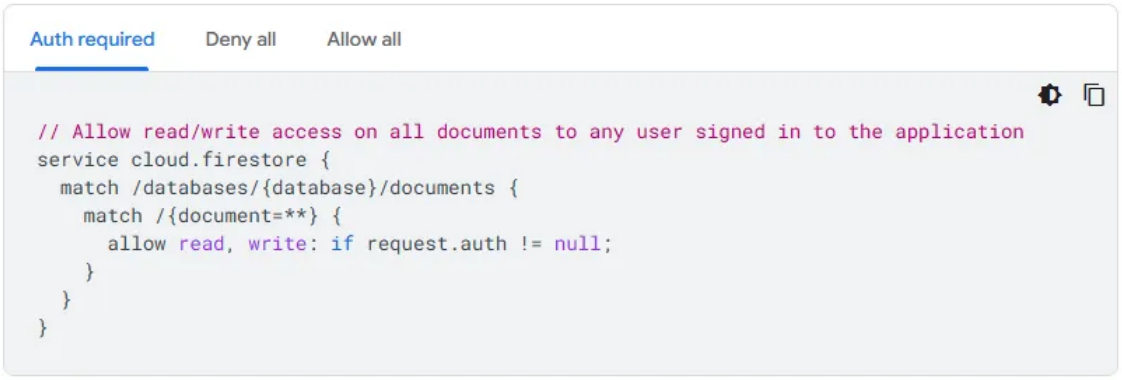

Figure 3.2 Example of Firestore access rules with unauthorised reads and authorised writes

3.6 Cloud Storage

Google's Cloud Storage provides scalable and secure solutions for storing media files in off-site locations. The data model is built around buckets that allow you to organise and access your data. Once a media file has been uploaded to the bucket, the application can retrieve its URL for seamless integration and dynamic loading of media content.

3.7 ExerciseDB API

Database API with a large selection of exercises, specifying the body part, target muscles, instructions and GIF demonstrations. Used to provide the exercises that physiotherapists can select to assign their patients. This API has a free subscription option, with a high service level and only 470ms latency, allowing for quick search responses and loading of the demonstration videos in the app.

3.8 Dart Packages

The Table Calendar (table_calendar) package [7] is used to implement a dynamic and interactive calendar feature in the app, reducing development time and leveraging a pre-built and thoroughly tested solution. The package provides a semi-robust calendar set up, with a collapsing Week-BiWeekly-Monthly calendar view, dots on the calendar days to represent events, and a list of events for the selected day appearing below the calendar view.

3.9 GitHub

GitHub is a cloud-based code sharing platform primarily used for collaborating on coding projects and managing version control. Version control tracks the complete history of changes made to files, eliminating the need to save multiple versions manually and allowing users to see exact changes that occur between commits. These streamlines work tracking and file version management. Git's functionality for isolating workflows ensures that features are developed independently, keeping the main project unaffected. Additionally, the platform serves as an extra online backup for projects.

4 Architecture and Design

This chapter outlines the architecture and design of PhysioHub and PhysioLink, detailing both client-side and server-side implementations. It explores the structural organisation, data flow and security measures that were applied when building these applications.

4.1 Client-Side Architecture

4.1.1 MVC Architecture

To create a clear separation of concerns, the Model-View-Controller design architecture was implemented for these two applications. Each component has distinct roles in order to simplify debugging, testing and updates.

Model classes define the data structure and handle data conversions between Firestore and the application. View classes manage the UI and interact with the Controllers and Models to display data. Controller classes handle business logic, database communication and data processing.

Implementing this design architecture allows for clear distinction between data management, user interface and business logic. The modularity proved effective in managing scalability and maintainability as the applications grew.

Alongside the standard MVC architecture, Provider classes were leveraged in both applications for state management, enhancing the role of the Controller classes. In PhysioHub, a DoctorProvider class is present, while in PhysioLink, a PatientProvider class is used. These classes hold the current state of the logged-in doctor and patient respectively, storing the data locally to avoid constantly re-fetching the information from the database and acting as a single source of truth for the user's information throughout the application. The Provider classes ensure that any changes to the Model of the logged-in user automatically trigger UI updates in the Views, adhering to Flutter's reactive programming style. For example, a change to the logged-in user's profile picture or name is immediately reflected in the UI without manually rebuilding any widgets.

Below is a diagram showing the MVC architecture with the addition of the Provider class:

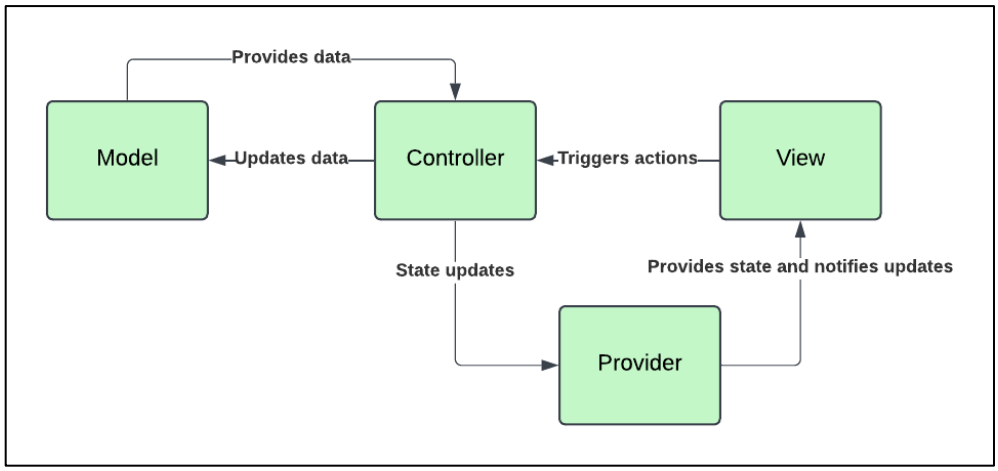

Figure 4.1 PhysioHub and PhysioLink architecture

With respect to the high-level architecture of these applications, both PhysioHub and PhysioLink are created with a custom bottom-navigation bar that allows for quick and easy navigation between the various screens. The diagram below depicts the tasks that each screen in PhysioHub provides. This is a strict superset of the tasks provided in PhysioLink. More information about these tasks and their implementation is provided in the following chapter.

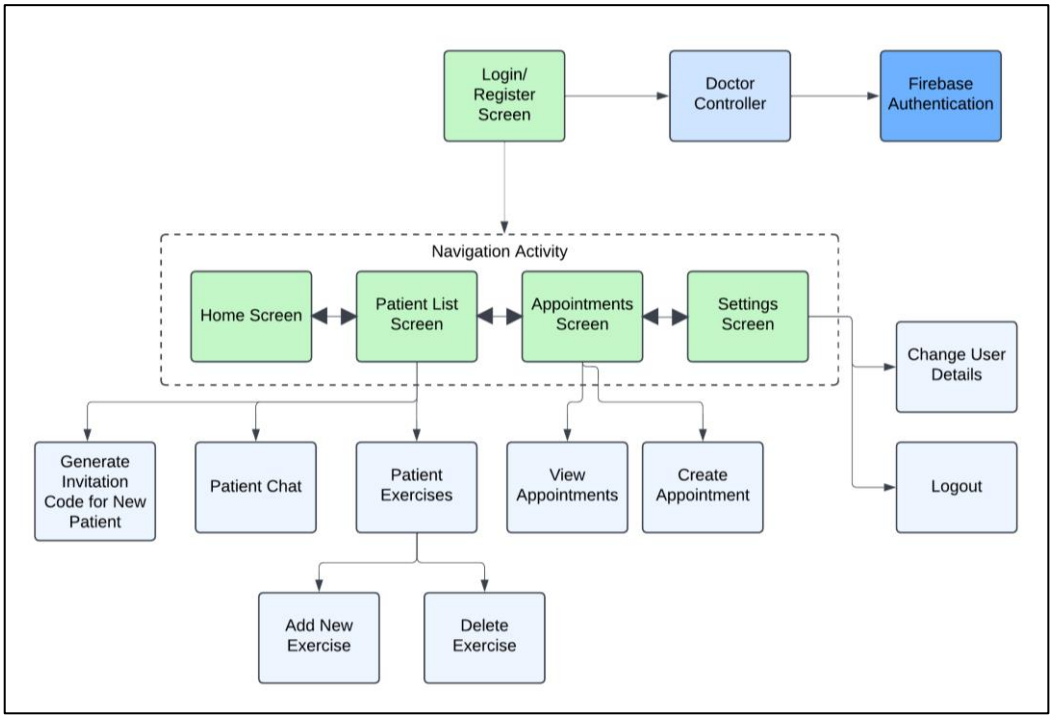

Figure 4.2 PhysioHub high-level architecture design

4.2 Server-Side Architecture

This sub-chapter explores the design and implementation of the backend services for PhysioHub and PhysioLink. As mentioned in Chapter 3, Google's Firebase service is used in these applications for authentication and database hosting. Firebase was selected as the preferred solution due to its comprehensive suite of tools, real-time database capabilities, and seamless synchronisation features. In addition, the prior implementation of Firebase in an earlier Android version of this project demonstrated its ease of use, flexibility and reliability.

4.2.1 Authentication Design

Firebase Authentication is leveraged in these applications. The service allows for quick integration of both log-in and sign-up functionality and offers a variety of authentication methods, although only the traditional email and password option was used in this project. Firebase Authentication provides automatic session management that ensures that users can stay logged-in on their device until they choose to log out and end their session, allowing users to simply re-open and continue using the applications without logging in each time. Firebase Authentication also seamlessly integrates with Firebase's other services. For example, database access can be determined by the user's authenticated user ID that is provided once a user successfully logs in.

After a successful log-in, the user is directed to their Home Screen and their data is queried based on their authenticated user ID.

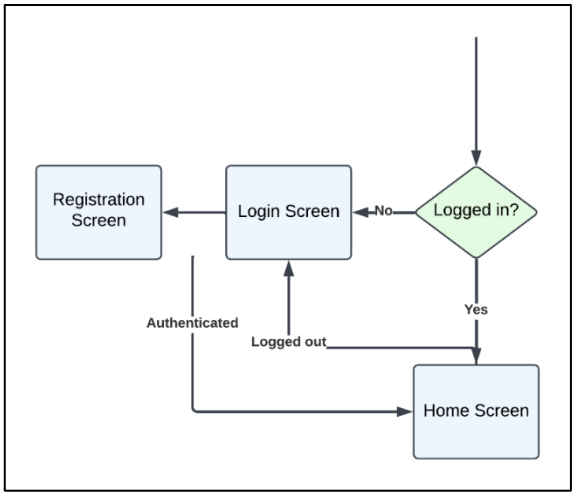

Figure 4.3 Authentication Logic Flowchart

4.2.2 Firestore Collections

These projects utilised Cloud Firestore for all the data storage and hosting, except for the users' profile pictures. Firestore is a NoSQL data storage solution where data is stored in documents that support many data types, including complex, nested objects. These documents are stored in collections. Most data models present in the applications map to a collection in Firestore.

In order to store user data, two collections were created: Patients and Doctors. Upon successful registration in either PhysioHub or PhysioLink, the user's information will be added to the respective collection. With Firebase Authentication, users are assigned a unique string ID that is also assigned to their document in the collection, making it easy to retrieve their data after logging in. The Patients collection has a field for their doctor's user ID to identify which patients belong to which physiotherapists. It also includes a field for notes on the patient that the physiotherapist can access and edit. Furthermore, instead of having a separate collection for storing exercises and assigning patient IDs to these exercises, the Patients collection includes a complex exercises field that stores an array. The exercise elements in this array include some identifying information, including ID, name, target muscles, instructions and the number of sets and repetitions assigned to this patient.

For a physiotherapist to add a new patient to their clinic, invitation codes can be generated. These codes are stored in the Invitations collection, with a unique String value, the ID of the physiotherapist who created the invite, and the expiration date of the invitation. When a patient wishes to register with PhysioLink, they will be required to enter a valid, non-expired invitation code. The patient can then be registered in the Patients collection and linked to their physiotherapist easily.

Appointments are stored in their own collection, with their date, time and the IDs of the relevant physiotherapist and patient.

Chat Messages are the final collection present in the Firestore for this project. In this collection, documents can be found with IDs of physiotherapists. Each of these documents has subcollections with IDs that correspond to that physiotherapist's patients' IDs. These subcollections then store individual documents for each message exchanged between that patient and their doctor. Messages are stored with their text data, the time

that they are sent, the sender and receiver ID as well as a Boolean to indicate if they have been read.

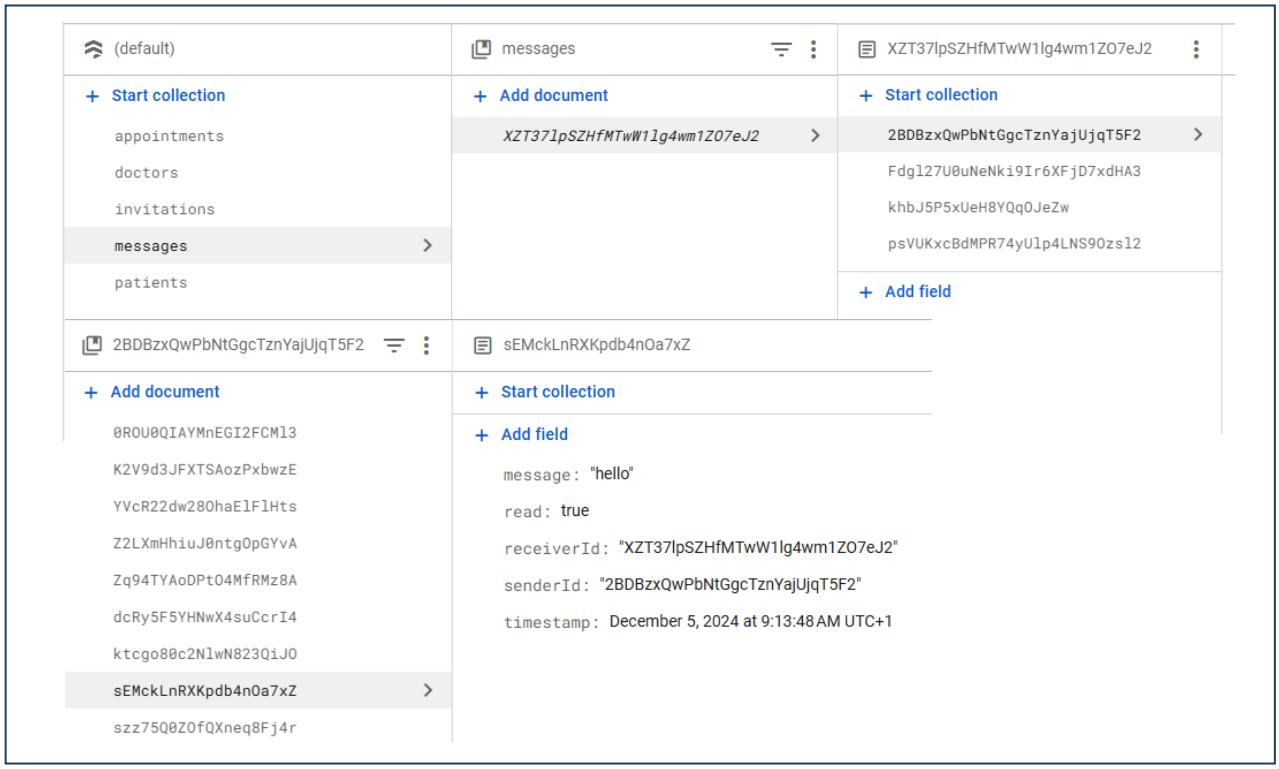

Figure 4.4 Data structure for the Messages Collection

Another reason Firestore was selected is its adaptable security rules, which are particularly important for managing personal and medical data. Firestore rules enable highly granular security management, which proved invaluable during development of these applications.

While most data in these collections are restricted to authenticated users, the Invitations collection required different handling. PhysioLink needed to access and read the Invitations collections prior to the patient even being registered, to check if the invitation code they provided was both present in the collection and still valid. Firestore's flexible rules allowed this functionality without compromising the security of other collections, enabling distinct permissions for reading and writing data in each collection.

Accordingly, the Invitations collection permits public read access, while write access is restricted to authorised users. This approach maintains security while facilitating a seamless registration process.

The Patients collection implements stricter access control to ensure the protection of patients' medical records. Patients can read all their own information and update specific fields, such as contact details. Physiotherapists are permitted to read and edit the information of their assigned patients only, ensuring maximum data privacy.

The Messages collection enforces rules to protect communication privacy. Messages are organised under doctor and patient IDs as previously mentioned, allowing only the relevant doctor and assigned patient to read or write messages in their subcollection.

These adaptable rules are essential to provide a balance between data access and protection in these applications.

```
rules_version = '2';

service cloud.firestore {
  match /databases/{database}/documents {
    // Allow unauthenticated users to read from the 'invitations' collection
    match /invitations/{invitationId} {
      allow read: if true;  // Public read access
      allow write: if request.auth != null; // Only authenticated users can write
    }
  }

  // Restrict access to the 'messages' collection to ensure only relevant users have
access
    match /messages/{doctorId}{
      // Access for specific patient subcollections under each doctor
      match /{patientId}/{messageId} {
        allow read, write: if request.auth != null &&
          (request.auth.uid == doctorId || request.auth.uid == patientId);
      }
    }

  // Patients collection - Allow patients to edit specific fields and doctors to access
assigned patients
    match /patients/{patientId} {
      // Read access:
      allow read: if request.auth != null && (
        request.auth.uid == patientId || // Patients can read their own data
        // Doctors can read assigned patients
        get(/databases/{database}/documents/patients/$(patientId)).data.doctorId ==
request.auth.uid
      );

      // Write access:
      allow write: if request.auth != null && (
        // Patients can edit their own data, but only specific fields
        (request.auth.uid == patientId &&
        request.resource.data.keys().hasOnly(['name', 'phoneNumber', 'email',
'photoUrl'])) ||

        // Doctors can edit all fields for their assigned patients
        get(/databases/{database}/documents/patients/$(patientId)).data.doctorId ==
request.auth.uid
      );
    }

    // Doctors collection - Public read access, restricted write access
    match /doctors/{doctorId} {
```

```
      allow read: if request.auth != null; // Any authenticated user can read doctor
profiles
      allow write: if request.auth.uid == doctorId; // Only doctors can edit their own
profile
    }

  // Rules for other collections that require authenticated access
    match /{document=**} {
      allow read, write: if request.auth != null; // Only allow authenticated access to
other collections
    }
}
```

5 Implementation

This chapter will provide a detailed explanation of the technical features provided in PhysioHub and PhysioLink, complete with code snippets and UI screenshots. This section aims to simplify the coding implementation and explain the connections between the front-end and back-end functionalities. Each sub-chapter represents a different main feature provided in the applications.

5.1 Log-in and Registration

The log-in and registration functionalities rely heavily on asynchronous programming and integration with Firebase Authentication. Both applications implement registration with email and password, and instead of providing 'user roles' and storing all users in one Firestore collection, users are added to either the Doctor or Patient collection depending on whether they are registering with PhysioHub or PhysioLink. To ensure that only registered physiotherapists can log-in to PhysioHub, the code checks for their details within the Doctors collection before they can move forward.

5.1.1 Error messages and user feedback

Error messages during log-in and registration are provided in various ways to ensure an intuitive experience. Missing log-in information (email or password) will result in a *SnackBar* notification reminding the user to enter both their email and password. Incorrect or invalid log-in credentials as well as attempts to log-in to the incorrect application will result in an *AlertDialog*. *AlertDialogs* were chosen over *SnackBar* notifications for these errors as they require greater user attention and *SnackBars* are only displayed briefly.

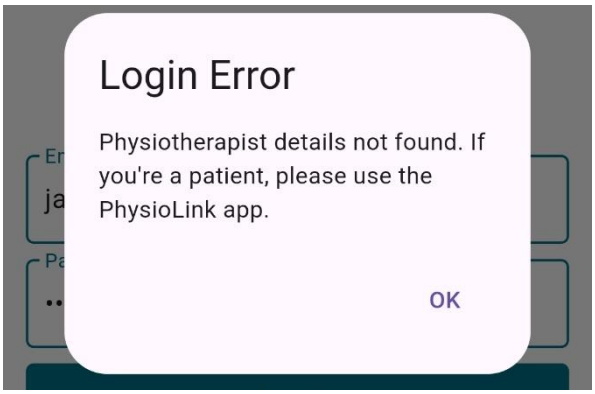

Figure 5.1 AlertDialog when Patients attempt to log-in with PhysioHub

The registration process is designed to provide dynamic error handling. Flutter's *TextFormField*, a UI element, is specifically designed for form-based input with built-in validation capabilities. Each *TextFormField* included in the registration form includes a *Validator* child that receives the input value of that field, validates the input and returns either an error message or null if the input is valid [26].

```
TextFormField(
 controller: _phoneController,
 decoration: InputDecoration(
     border: OutlineInputBorder(),
     labelText: 'Phone Number',
     labelStyle: TextStyle(color: Theme.of(context)
         .primaryColor)),
 keyboardType: TextInputType.phone,
 validator: (value) {
   if (value == null || value.isEmpty) {
     return 'Please enter your phone number';
   }
   if (!RegExp(r'^[0-9]{10}$').hasMatch(value)) {
     return 'Please enter a valid 10-digit phone number';
   }
   return null;
 },
),
```

These *TextFormFields* are built within a *Form* widget that includes a submit button. When the button is pressed, all the fields within the *Form* will be validated, error messages displayed as needed, and progression to the rest of the registration process is only permitted if all fields are in the correct format.

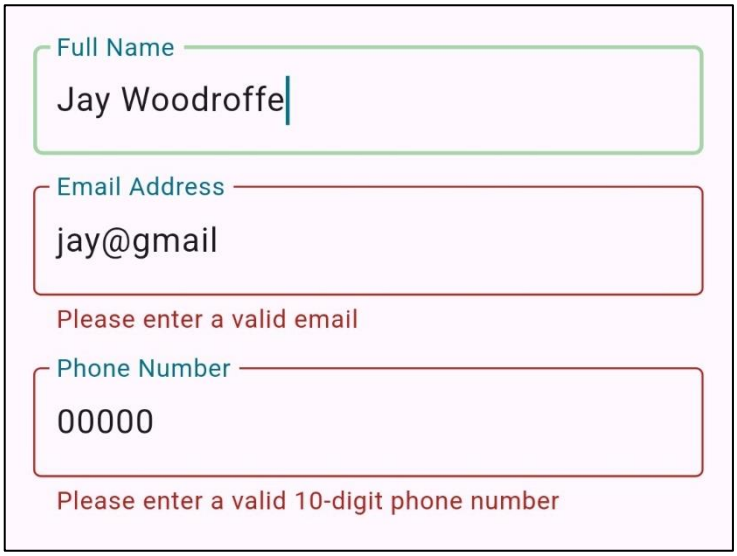

Figure 5.2 TextFormFields for registration with error messages

5.1.2 Asynchronous Programming

Asynchronous programming throughout PhysioHub and PhysioLink was employed using *Future* functions to enhance application performance and user

experience. A *Future* in Flutter is used in combination with '*async'* and '*await'* keywords to enable non-blocking execution of asynchronous operations [27]. This approach allows the application to remain responsive while performing tasks that may take time to complete. All methods that involve interactions with the backend, such as authorising users, registering their information in the database, and retrieving user data from the collection after login, are invoked using '*await'*. This ensures that the application can efficiently handle multiple operations concurrently, thereby minimising the perceived latency.

5.2 Patient Management and Chat Feature

Physiotherapists can access their registered patients through the Messages tab located in the bottom navigation bar. Patient data is retrieved from the Patients collection in Firestore, while a *Stream* is utilised to provide real-time updates on the unread message count for each patient based on the Messages collection.

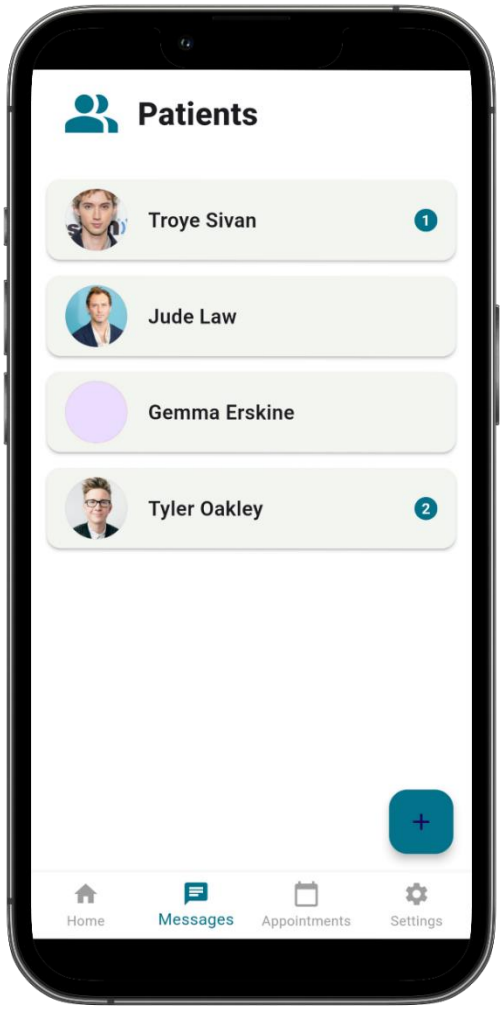

Figure 5.3 Patient list with unread message count in PhysioHub

```
messagesRef = _firestore.collection('messages')
    .doc(doctorId) // Doctor's document
    .collection(patientId); // Patient's subcollection

Stream<int> getUnreadMessageCountStream() {
  return messagesRef.where('read', isEqualTo: false)
      .where('receiverId', isEqualTo: doctorId)
      .snapshots()
      .map((snapshot) => snapshot.docs.length);
}
```

5.2.1 State Management and DoctorProvider

```
import 'package:provider/provider.dart';
```

The '*provider'* package plays a crucial role in enhancing the responsiveness of PhysioHub and PhysioLink. By utilising this package, the dedicated classes *DoctorProvider* and *PatientProvider* were created to manage the state of the applications effectively. When these classes register with the *'provider'* package, any widget in the application can access the provider without needing to pass the provider instance down the widget hierarchy manually. This registration is typically performed in the main.dart file using the *MultiProvider* widget, as shown below:

```
MultiProvider(
          providers: [
          ChangeNotifierProvider(create: (_) => DoctorProvider()),
          ChangeNotifierProvider<ExerciseProvider>(create: (_) => ExerciseProvider()),
                    ],),
```

Widgets that need to respond to changes in *DoctorProvider*, such as updates to the number of unread messages from patients, can use the Consumer widget of the *Provider.of<DoctorProvider>(context)* method. This allows them to listen for changes and rebuild only the necessary components when *notifyListeners()* is called. For instance, the *PatientList* Screen listens to *DoctorProvider* via the Consumer widget. Meanwhile, *DoctorProvider* fetches and maintains the state of patients and their unread message counts by setting up listeners with the *ChatMessageController*, ensuring that updates are propagated back to the view for real-time UI changes, as seen in the code below:

```
void fetchUnreadMessageCounts() {
  if (doctor != null && doctor!.patients.isNotEmpty) {
    for (Patient patient in doctor!.patients) {
      final controller = ChatMessageController(
        doctorId: doctor!.id!,
        patientId: patient.id,
      );
      // Listen to unread message count stream for each patient
      controller.getUnreadMessageCountStream().listen((unreadCount) {
        patient.unreadMessages = unreadCount;
        notifyListeners(); // Update UI when count changes
      });
    }}
}
```

The *notifyListeners()* method is also employed in the *DoctorProvider* class to trigger UI updates when the physiotherapist's data is loaded, patient data is retrieved, and appointments are loaded. This ensures that the UI remains in sync with the underlying data model, providing a seamless user experience.

5.2.2 Chat Feature

The chat feature in these applications was designed to facilitate real-time communication between physiotherapists and their patients. Clicking on any patient in the *PatientList* Screen will open their conversation with the logged-in physiotherapist. This implementation leverages Firebase Firestore for data storage and retrieval, ensuring that messages are synchronized across devices in real-time.

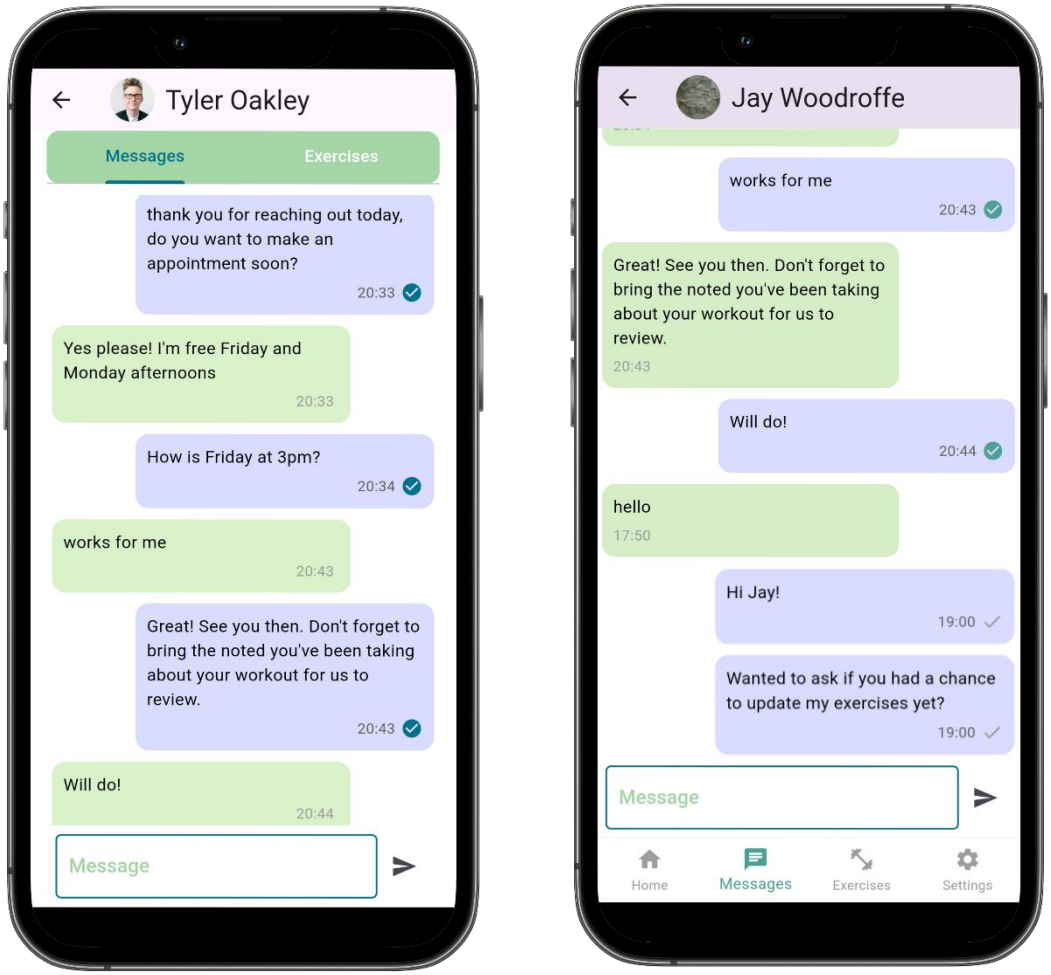

Figure 5.4 Chat Feature in PhysioHub (left) and PhysioLink (right)

At the core of the chat functionality is the *ChatMessageController* class, which manages all message interactions with the Firestore database. The *ChatMessageController* is initialised for each conversation using the physiotherapist and patient IDs and references this specific sub-collection within the database. The *_sendMessage()* method of ChatMessageController generates a new document in the Firestore sub-collection, storing the message content, sender and receiver ID, message time stamp and read status.

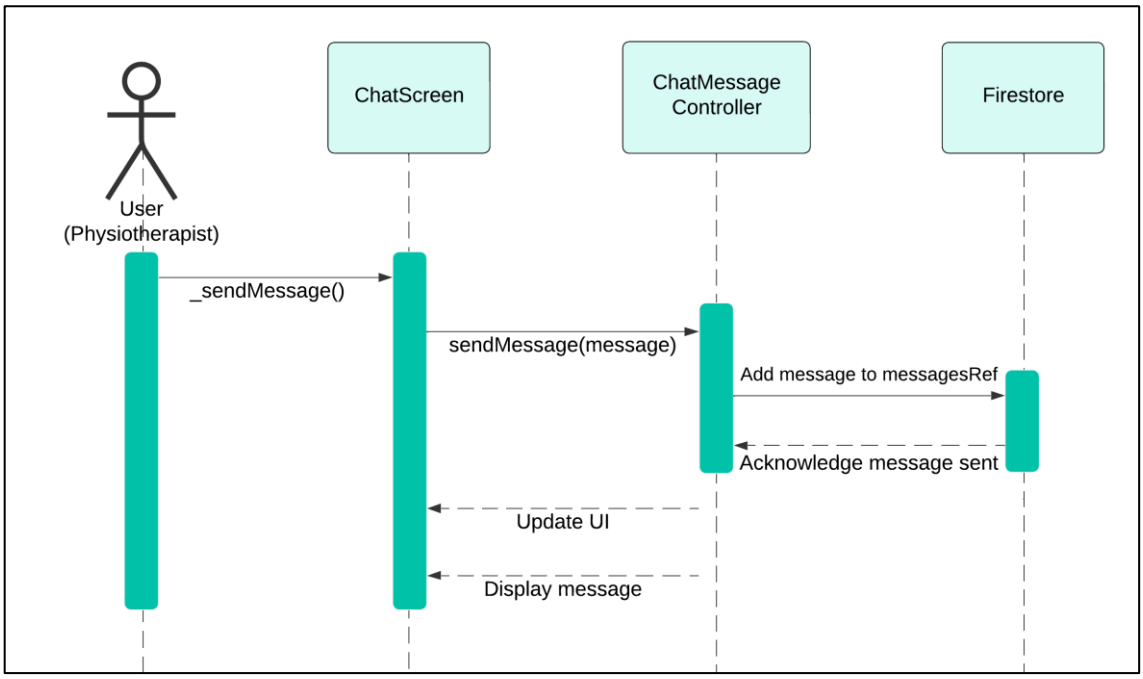

Figure 5.5 Sequence Diagram for sending messages

The *getMessages()* method provides a stream of messages ordered by timestamp, ensuring the chat interface remains updated in real-time as messages are sent and received. The *ChatScreen* widget leverages this stream via the *StreamBuilder* widget, dynamically rebuilding the UI whenever new messages are added or existing ones are updated. This design ensures the chat interface consistently displays the most current conversation data.

To complement this, the *markMessagesAsRead()* method allows for bulk updtes to the read status of messages. This ensure that the interface reflects read messages with a ‘read’ icon displayed at the bottom-right corner of the messages.

Within the PhysioHub application, clicking on a patient’s name within the chat interface will direct the user to the patient’s detailed contact card, where the patient’s profile picture, cell phone number, age and notes are displayed. The cell phone icon can be pressed to initiate a call directly to the patient, leveraging the device’s capabilities. The

implementation of this feature is handled by the *_callPatient()* method, which constructs a Uri with the *tel* scheme to facilitate the call:

```
void _callPatient() async{
  final Uri phoneUri = Uri(scheme: 'tel', path: widget.patient.phoneNumber);
  if (await canLaunchUrl(phoneUri)) {
    await launchUrl(phoneUri);
  } else {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text("Could not launch phone dialer")),
    );
  }
}
```

This functionality works seamlessly across both Android and iOS platforms, using their native phone applications to carry out the call.

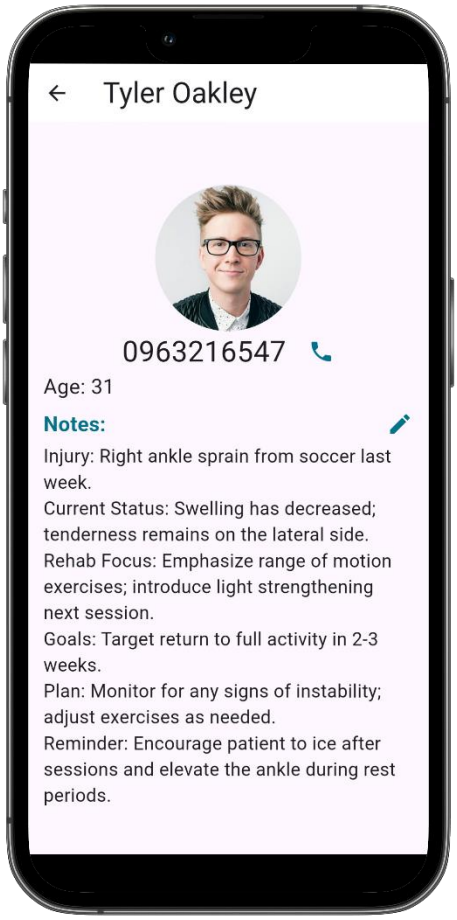

Figure 5.6 Patient Details in PhysioHub

The patient contact card also enables physiotherapists to view and edit notes related to each patient. These notes are exclusively accessible to the physiotherapist, serving as helpful reminders and facilitating personalized care. When a physiotherapist updates or saves notes for a patient, the changes are automatically synchronized and stored in the patient's document within the Patients collection in Firestore.

```
Future<String> updatePatientNotes(String patientId, String notes) async{
    try{
      await _firestore.collection('patients').doc(patientId).update({
        'notes': notes,
      });
      return('Patient notes updated.');
    } on FirebaseException catch (e){
      return('Failed to update patient notes.');
    } catch (e) {
      return ('An unexpected error occured');
    }
  }
```

This code snippet is taken from the *PatientController* class, which is responsible for accessing and updating patient information in both applications. The asynchronous operation is managed through the *Future* class, and a relevant message is returned to the calling class indicating the status of the update.

5.3 Exercise Prescription and Management

Patients can easily find their current exercise regime through the ‘exercises’ tab in PhysioLink’s bottom navigation bar. In PhysioHub, practitioners can navigate to a patient’s exercises through the toggle button at the top of the chat interface for that patient. This toggle button is created by implementing a *TabController* to provide seamless navigation between the two primary views on this screen, Chat and Exercise List. This design leverages Flutter’s *TabBar* and *TabBarView* widgets to create a responsive and intuitive design.

The Exercise Screen allows physiotherapists to view, add and delete exercises associated with a patient, while patients are only able to view.

When the screen is initialised, the exercises for the specified patient are fetched asynchronously using the *ExerciseController*. The screen utilises the *FutureBuilder* widget to handle the asynchronous data fetching, listening for the completion of the *Future* object and updating the UI accordingly:

```
body: FutureBuilder<List<Exercise>>(
  future: _exercisesFuture,
  builder: (context, snapshot) {
    if (snapshot.connectionState == ConnectionState.waiting) {
        // While data is being fetched, show a loading indicator
            return Center(child: CircularProgressIndicator());
    } else if (snapshot.hasError) {
        // If there's an error, display an error message
             return Center(child: Text('Error loading exercises'));
    } else if (!snapshot.hasData || snapshot.data!.isEmpty) {
        // If no exercises are found, display a message
            return Center(child: Text('No exercises found.'));
    } else {
```

```
        // If data is available, show the exercises
            final exercises = snapshot.data!;
      return GridView.builder(
        gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
          crossAxisCount: 2,
          crossAxisSpacing: 2.0,
          mainAxisSpacing: 2.0,
          childAspectRatio: 0.70,
        ),
        itemCount: exercises.length,
        itemBuilder: (context, index) {
          return ExerciseCard(
              exercise: exercises[index],
              patientId: widget.patientId,
              selectionMode: selectionMode,
              isSelected:
                  selectedExerciseIds.contains(exercises[index].id),
              onLongPress: () {
                toggleSelectionMode();
                toggleExerciseSelection(exercises[index].id);
              },
              onSelect: (isSelected) => toggleExerciseSelection((exercises[index].id)));
        },);}},),
```

The builder checks the connection state to display a loading indicator, error message, or the list of exercises based on the data retrieval. The returned exercises are displayed through a *GridView* widget comprised of custom *ExerciseCard* widgets, designed to display the GIF of the exercise, the exercise's name and the number of sets and repetitions the patient was prescribed for that exercise.

5.3.1 Exercise Management

The *ExercisesList* class, part of the PhysioHub application, manages the display and modification of a patient's exercise regimen. A central feature of this screen is the dynamic state management, particularly regarding the behaviour and appearance of the *Floating Action Button* (*FAB*), a Material Design component that serves two distinct purposes depending on the current state of the screen.

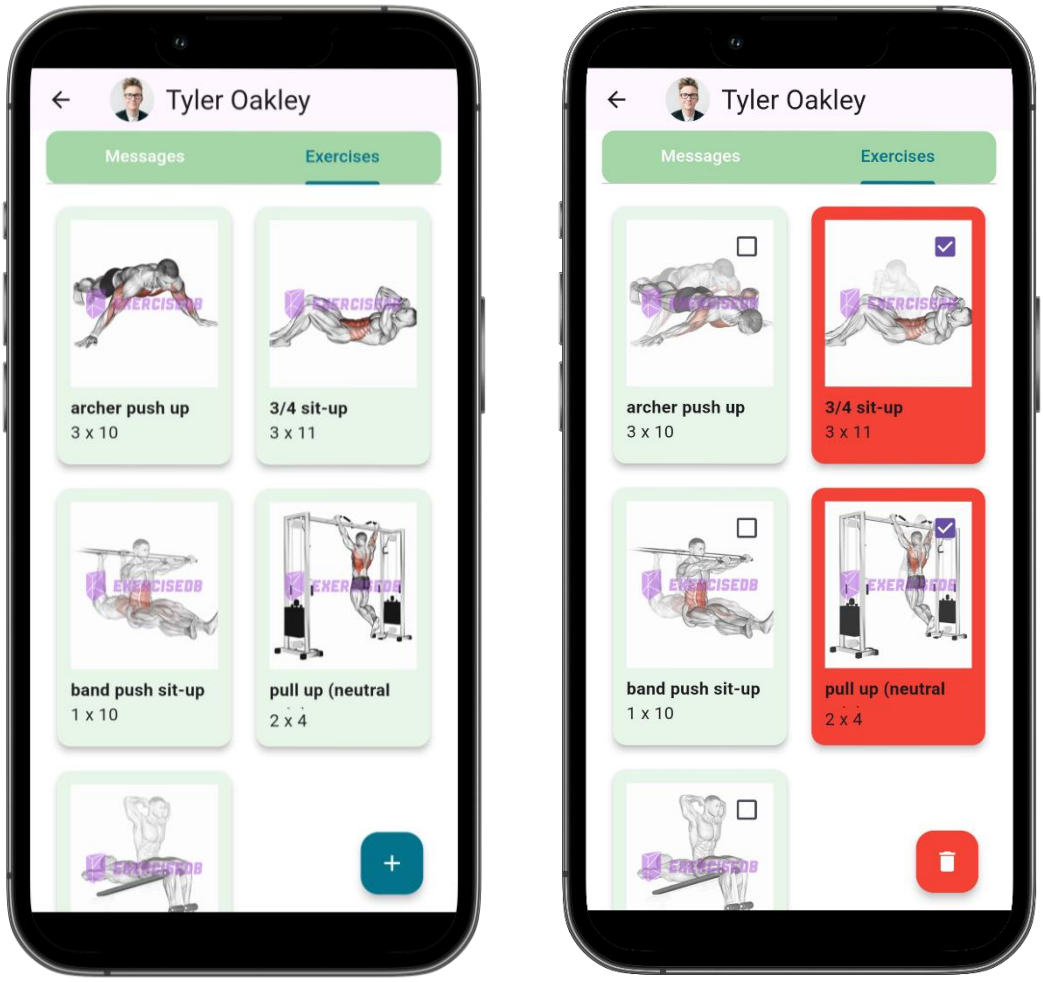

Figure 5.7 Exercise List editing in PhysioHub, with selection mode activated on the right

Implemented as a *StatefulWidget*, the *ExerciseList* screen responds to changes in state by using *setState()*, allowing the UI to update based on user interactions. The class maintains a *selectionMode* boolean variable to track whether the user is in selection or non-selection mode. In non-selection mode, the *FAB* displays a plus icon and allows the user to navigate to the *AddExercise* screen to search for exercises.

When the user long-presses an exercise card, the screen enters selection mode, where selected cards are highlighted, checkboxes appear on each exercise, and the *FAB* transforms into a red dustbin icon for deleting selected exercises. The selected exercise IDs are passed to the *deleteExercises()* method in the *ExerciseController*, which performs a Firestore transaction to safely remove the selected exercises from the patient's workout plan. This method uses the *where()* function to filter out the exercises with IDs in the *selectedIds* list, and then updates the patient's exercise list in Firestore with the remaining exercises.

```
Future<void> deleteExercises(String patientId, List<String> selectedIds) async {
  try {
    //reference the patient's document in firestore
    DocumentReference patientDocRef = _firestore.collection('patients').doc(patientId);

    await _firestore.runTransaction((transaction) async {
      DocumentSnapshot patientSnapshot = await transaction.get(patientDocRef);

      if (patientSnapshot.exists) {
        //retrieve the current exercise list
        List<dynamic> exercises = patientSnapshot.get('exercises') ?? [];
        exercises = exercises.where((exercise) =>
!selectedIds.contains(exercise['id'])).toList();
        //update the exercises field in firestore with the filtered list
        transaction.update(patientDocRef, {'exercises': exercises});
      }
    });
    print('selected exercises deleted successfully');
  } catch (e) {
    print('error deleting selected exercises $e');
    throw e;
  }
}
```

5.3.2 ExerciseDB API

All the exercises referenced in these applications are sourced from the ExerciseDB API [8]. This API has a suite of exercises with instructions and GIF demonstrations. The *ExerciseApiService* class is created to handle all interactions with this API. Within PhysioHub, physiotherapists can search for exercises based on their names and choose from the results to add to their patient's prescription. The physiotherapist can edit the number of sets and repetitions the patient should complete.

A challenge posed while using this API was that the GIF URLs provided are only valid for 24 hours. As a result, these URLs could not be stored directly in the Patients collection along with the rest of the information about their exercises. Instead, when loading the exercises, the application fetches the GIF URLs for each exercise from the API using the exercise IDs. If the connection to the API is compromised, an error message

‘Failed to fetch GIF URL’ will be shown, but the rest of the exercise information will still be available.

```
Future<String?> fetchGifUrl(String exerciseId) async {
  print(exerciseId);

  //uses the id of exercise from the API
  final url = Uri.parse('$_baseUrl/exercises/exercise/$exerciseId');
  final response = await http.get(url, headers: {
    'X-RapidAPI-Key': _apiKey,
    'X-RapidAPI-Host': 'exercisedb.p.rapidapi.com',
  },);

  if (response.statusCode == 200) {
    final data = json.decode(response.body);
    final url = data['gifUrl'] as String?;

    return url;

  } else {
    throw Exception('Failed to fetch gif URL');
  }
}
```

5.4 Calendar

The Calendar feature is implemented in PhysioHub with the use of the *‘table_calendar’* package serving as a vital component for managing appointments between physiotherapists and patients. The *AppointmentScreen* is designed as stateful widget that effectively handles the calendar user interface and user interactions.

Upon initialisation of *AppointmentScreen*, the current day is automatically selected in the calendar. The application employs *Future.microtask* to asynchronously retrieve the physiotherapist’s existing appointments from the Firestore Appointments collection via the *AppointmentController* class. The asynchronous data retrieval ensures that the UI remains responsive while loading the necessary information.

The application retrieves events for a specific day by filtering the physiotherapist’s appointments based on the selected date in the Calendar. This user interaction triggers a state change that updates the displayed events, allowing users to seamlessly navigate through their appointments.

The process of adding new appointments is facilitated through a user-friendly *dialog* interface. In this interface, users are prompted to select a patient from their registered list and choose a date and time for the appointment. The Flutter framework provides specific functions, *showDatePicker()* and *showTimePicker()*, that streamline user interactions for selecting date and time respectively. By leveraging these built-in functions, the application provides a functional and intuitive user experience, ultimately enhancing the overall usability of the appointment management system.

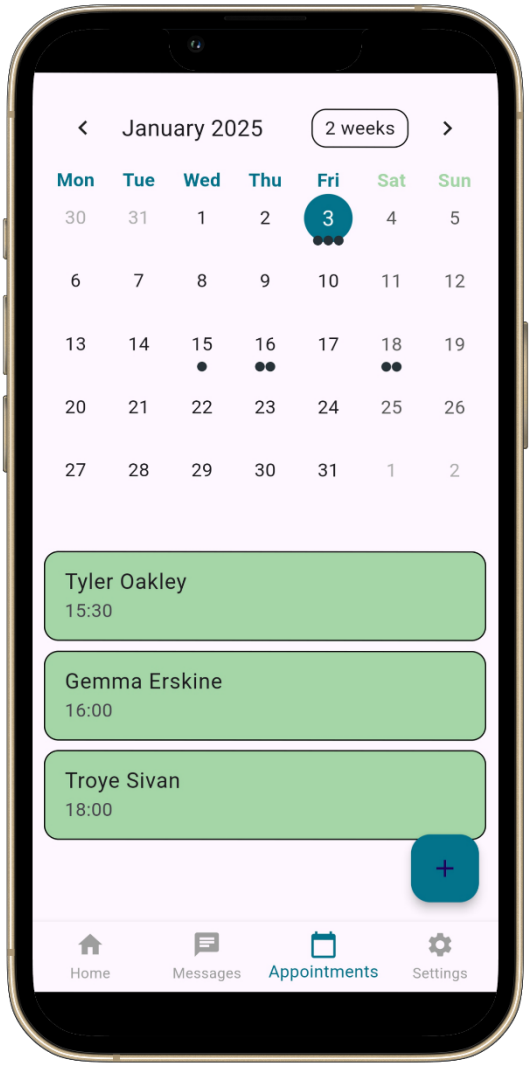

Figure 5.8 Appointment Screen in PhysioHub

5.5 Settings

The *SettingsScreen* is accessible to users of PhysioHub and PhysioLink through the settings icon in the bottom navigation bar. This screen provides users the opportunity to update their profile pictures and names.

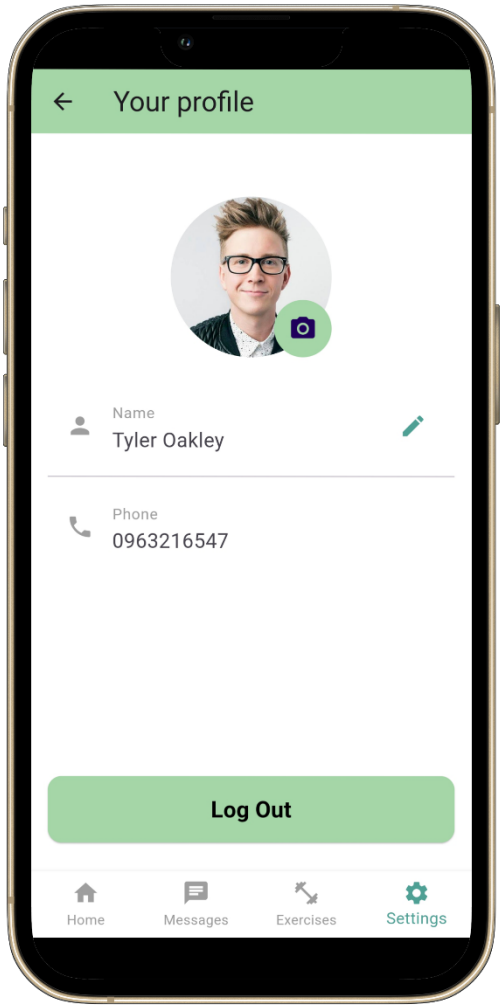

Figure 5.9 Settings Screen in PhysioLink

5.5.1 Profile pictures and GoogleCloud Storage

The process of allowing users to upload their profile picture from their devices' gallery is implemented using the '*image_picker*' Dart package.

```
import 'package:image_picker/image_picker.dart';
```

This package provides a user-friendly API to access the device's photo library or camera, enabling the users to select or capture images directly within the application [32].

```
//Select an image using ImagePicker
final XFile? pickedFile = await _picker.pickImage(source: ImageSource.gallery);
if (pickedFile == null) return "No image selected.";

//Convert to File and get file name
File file = File(pickedFile.path);
String fileName = path.basename(pickedFile.path);
```

In PhysioHub and PhysioLink, Google Cloud Storage is used to securelu store users' profile pictures. When a user selects an image from their device's gallery, Google Cloud Storage is initialised using the credentials.json file. The selected image is then

uploaded to a designated storage bucket via a *StorageApi* object. After the upload is complete, the resulting URL of the stored image is added to the user's profile data within their respective Firestore collection – either Patients or Doctors. This enables seamless retrieval and display of profile pictures across the applications.

5.5.2 Logout

The logout functionality leverages Firebase Authentication to handle the sign-out process. The following code snippet demonstrates the implementation of the logout feature for physiotherapists in PhysioHub:

```
final FirebaseAuth _auth = FirebaseAuth.instance;
Future<String?> logoutDoctor() async {
  try {
    await _auth.signOut();
    return null;     } catch (e) {
    return 'An error occurred while logging out.'
  }
}
```

Firebase's *signOut()* method is called, invalidating the current authentication token and effectively ending the user's session and directing them back to the Login Screen.

6 Comparison between Android and Flutter solutions

As was previously mentioned, PhysioHub and PhysioLink were initially developed in Kotlin before being redeveloped in Dart. This chapter will examine the key differences between the earlier Android versions and the Dart-based versions that have been analysed in this paper. The subsequent sections will explore how asynchronous programming, state management, and user interface designed were approached in each version of PhysioHub and PhysioLink.

6.1 Asynchronous Programming

Upon re-examining the Kotlin project code, it is evident that the handling of asynchronous methods, especially when interacting with Firebase Firestore, is quite clumsy and prone to issues. The Kotlin code primarily relies on *callback* methods when executing Firestore operations. *Callback* functions, when used to handle asynchronous programming, are designed to 'pass one function as a parameter to another function and have this one invoked once the process is complete' [34]. While this approach can be functional, it has several drawbacks that can lead to serious issues. Below is a code snippet from the Kotlin applications for adding an appointment to Firestore:

```
fun addAppointment(selectedClient: String, timestamp: Timestamp, callback: (Boolean) ->
Unit)
    {
        val docCertId = DoctorDataHolder.getLoggedInDoctor()?.certId
        if (docCertId != null) {
            val db = FirebaseFirestore.getInstance()
            // Create a new appointment object
            val appointment = hashMapOf(
                "clientName" to selectedClient,
                "date" to timestamp,
                "doctorCertId" to docCertId
            )
            // Add the appointment to the "appointments" collection
            db.collection("appointments")
                .add(appointment)
                .addOnSuccessListener { documentReference ->
                    // Appointment added successfully
                    callback(true)
                }
                .addOnFailureListener { e ->
                    // Error adding appointment
                    callback(false)
                }
        } else {
            callback(false)
        }
    }
```

Firstly, the use of callbacks increases error handling complexity, as the responsibility of error handling gets spread across multiple locations. The success and failure of a Firestore operation are handled within the *addOnSuccessListener* and *addOnFailureListener* callbacks, respectively. This makes it challenging to handle errors in a consistent manner, as each asynchronous operation requires its own callback for error handling.

Secondly, the use of callbacks can lead to a situation known as 'callback hell', where multiple nested callbacks create a convoluted structure that is increasingly difficult to read and troubleshoot.

The code below shows how the *addAppointment* method is called in the Kotlin project:

```
AppointmentsDataAccess.addAppointment(selectedClient, timestamp ){success ->
          if (success){
             startActivity(Intent(this, Appointments::class.java))
          }
          else
          {
             Toast.makeText(this, "Failed to add appointment", Toast.LENGTH_SHORT).show()
          }
}
```

In contrast, the Dart-based PhysioHub and PhysioLink leverages *'async/await'*, which allows for a more linear and readable approach to asynchronous programming. This approach eliminates the need for callbacks and can handle multiple asynchronous operations sequentially without leading to a clutter of nested callbacks. The following code shows how the same method for adding an appointment to the Firestore collection is handled with Dart:

```
Future<String?> createAppointment(Appointment newAppointment) async {
    try {
        await _appointmentsCollection.add(newAppointment.toFirestore());
        print('Appointment added successfully');
    } catch (e) {
        print('Failed to add appointment: $e');
        throw e;
    }
}
```

This method has a far clearer flow of execution, making it easy to follow the logic. The use of *'try/catch'* blocks for error handling also allows for a more organised approach to managing exceptions, as all error handling is contained within the function scope.

6.2 State Management

There is minimal state management implemented in the Kotlin-version of PhysioHub and PhysioLink. Changes to data are only shared through result callbacks when methods in any of the *DataAccess* classes are called, and screens are reloaded when new data is fetched from Firestore.

In comparison, Dart offers various state management solutions. The solution used in these applications was the *Provider-Consumer* option that was discussed in the previous chapter. *Provider* allows the user to create a *ChangeNotifier* that can notify any widgets that are listening to a certain class that the state has changed. By employing the *notifyListeners()* method, the UI can be decoupled from the business logic, as the UI components do not need to know how the data is managed or updated but can simply listen for changes to the data and rebuild parts when notified. This not only made the Dart code cleaner and more maintainable than its Kotlin counterpart, but also ensured that data across the application was more current and correct, reflecting any new changes that may have occurred since the screen was built.

6.3 User Interface

While there can be a learning curve to using Dart for designing a user interface over XML, it quickly became evident that Dart offers a more pleasant and streamlined approach. The declarative nature of Flutter allows developers to describe the UI in terms of its current state, making it easier to visualise how the interface will look and behave. For instance, the use of *StreamBuilder*, as demonstrated in the Dart code below, enables real-time updates to the chat interface without the need for manual refreshes.

```
StreamBuilder<List<ChatMessage>>(
  stream: getMessagesStream(),
  builder: (context, snapshot) {
    // Update UI based on snapshot data
  },
)
```

In contrast, XML requires a more imperative approach, where the UI is defined statically, and updates must be handled explicitly in the activity or fragment code. Setting up event listeners for all user gestures that should be reacted to and managing data binding with adapters attached to *RecyclerViews* can be cumbersome. The adapters must be manually notified of any data changes, which introduces extra complexity and a higher potential for errors.

Furthermore, Dart's widget-based architecture provides greater flexibility and composability in building user interfaces. Each widget can be easily customised and nested, enabling the creation of complex layouts with minimal effort. This was especially useful when creating the chat and exercise list screens.

7 Summary

PhysioHub and PhysioLink are cross-platform mobile applications developed to meet the clinic and patient management needs of smaller physiotherapy clinics. Designed for both physiotherapists and patients, these applications simplify and streamline processes such as patient management, appointment scheduling, exercise prescription, and communication.

PhysioHub provides a secure environment for physiotherapists to take notes on their patients, create and management appointments and manage patient exercise prescriptions. PhysioLink allows patients to view their upcoming appointments and prescribed exercises. Exercises are shown with their demonstration video and step-by-step instructions to assist patients. Both applications include a direct message communication feature that provides physiotherapists the ability to talk directly with their patients through the application.

These applications leverage Flutter's modern UI capabilities to create an aesthetic and pleasant-to-use design, while Firebase's backend services are implemented for scalable, secure and flexible data storage. PhysioHub and PhysioLink were developed with the intention of expanding them into larger, more comprehensive services. This scalability was a key consideration when selecting a development language. Dart was chosen for its clean syntax, strong support for asynchronous programming, and ability to deliver high-performance applications on both Android and iOS.

Developing these applications provided invaluable experience in cross-platform development. Learning Dart and Flutter offered insight into modern programming practices and frameworks, particularly for creating responsive and visually appealing mobile applications.

7.1 Future Developments

PhysioHub and PhysioLink currently fulfil the project's initial requirements, offering robust tools for managing physiotherapy clinics and patient care. However, there are numerous opportunities to expand their capabilities to better serve all users and align with evolving industry standards.

First, the authentication system could be enhanced by including multiple login providers, such as Google or Facebook, and implementing password recovery options to streamline user registration and login. Multi-factor authentication would also further improve the security of these applications.

Introducing app notifications to PhysioHub and PhysioLink would significantly improve usability. Notifications alerting users to new messages and reminding patients when to do their rehabilitation exercises based on their chosen schedule would promote better adherence to treatment plans.

Furthermore, these applications are designed to be used in a medical field and thus store user's personal and medical information. If these applications were to be released, they would need to be improved to comply with location-specific data protection laws, such as the General Data Protection Regulation (GDPR) for the EU, that mandate strict data protection and privacy requirements. While these applications currently meet the standards of the GDPR regarding data encryption both at rest and in transit, this law requires systems to have certain security features, including multi-factor authentication, audit logs and breach-detection systems, that these applications are currently lacking.

There are many different medical record formats commonly used in mobile applications that are designed to facilitate interoperability and efficient data management. Implementing a data exchange standard, such as Fast Healthcare Interoperability Resource [35], would allow PhysioHub and PhysioLink to seamlessly and safely share data across different healthcare systems. This would facilitate collaboration among healthcare providers, ensuring physiotherapists can efficiently exchange patient records with other professionals involved in patient care.

Additionally, PhysioHub's functionality can be expanded to manage financial operations, such as automating invoicing based on appointment details and services rendered. Features for tracking payments and generating detailed financial reports could be developed to further assist with clinic management.

These future developments would enhance the utility and scalability of PhysioHub and PhysioLink, ensuring they remain competitive and capable of meeting the diverse needs of physiotherapy clinics and their patients.

8 References

[1] M. Koko, “🔒 📱 Modern Login UI • Flutter Auth Tutorial ♡,” *YouTube*. Dec. 12, 2022. Available: https://www.youtube.com/watch?v=Dh-cTQJgM-Q. [Accessed: Sep. 11, 2024]

[2] T. Nagar, “Role of mobile apps in the healthcare industry [2023],” *Dev Techny’s UAE*, Dec. 30, 2022. Available: https://devtechnosys.ae/blog/role-of-mobile-apps-in-the-healthcare-industry/. [Accessed: Sep. 16, 2024]

[3] “Navigate to a new screen and back,” *Flutter.dev*, 2024. Available: https://docs.flutter.dev/cookbook/navigation/navigation-basics. [Accessed: Sep. 24, 2024]

[4] “TeleHab | Exercise prescription app for coaches and athletes,” *valdperformance.com*. Available: https://valdperformance.com/products/telehab

[5] M. Merolli, J. J. Francis, P. Vallance, K. L. Bennell, P. Malliaras, and R. S. Hinman, “Evaluation of Patient-Facing Mobile Apps to Support Physiotherapy Care: Systematic Review,” *JMIR mhealth and uhealth*, vol. 12, pp. e55003–e55003, Jan. 2024, doi: https://doi.org/10.2196/55003. Available: https://mhealth.jmir.org/2024/1/e55003. [Accessed: Oct. 01, 2024]

[6] M. Mosa, I. Yoo, and L. Sheets, “A Systematic Review of Healthcare Applications for Smartphones,” *BMC Medical Informatics and Decision Making*, vol. 12, no. 1, Jul. 2012, doi: https://doi.org/10.1186/1472-6947-12-67. Available: https://link.springer.com/article/10.1186/1472-6947-12-67. [Accessed: Oct. 01, 2024]

[7] “table_calendar,” *Dart packages*, Feb. 22, 2019. Available: https://pub.dev/packages/table_calendar. [Accessed: Oct. 21, 2024]

[8] “ExerciseDB,” *Rapidapi.com*, 2022. Available: https://rapidapi.com/justin-WFnsXH_t6/api/exercisedb/playground/apiendpoint_a4034d70-d2c1-4c90-afbe-b3c1abf70c28. [Accessed: Nov. 01, 2024]

[9] “Understand Firebase projects,” *Firebase*, 2024. Available: https://firebase.google.com/docs/projects/learn-more. [Accessed: Nov. 17, 2024]

[10] “Use Firebase,” *Expo Documentation*, 2024. Available: https://docs.expo.dev/guides/using-firebase/. [Accessed: Nov. 17, 2024]

[11] “Firestore | Firebase,” *Firebase*, 2024. Available:

https://firebase.google.com/docs/firestore. [Accessed: Nov. 17, 2024]

[12] "What is Flutter? - Flutter App Explained - AWS," *Amazon Web Services, Inc.*, 2018. Available: https://aws.amazon.com/what-is/flutter/. [Accessed: Nov. 18, 2024]

[13] R. Holewa and Miquido, "Flutter vs Dart: Key Differences and Ideal Use Cases Explained," *Miquido*, Aug. 26, 2024. Available: https://www.miquido.com/blog/flutter-vs-dart/#:~:text=polished%20UI%2FUX.-,What%20is%20the%20Difference%20Between%20Dart%20and%20Flutter%3F,in%20the%20app%20development%20process.. [Accessed: Nov. 18, 2024]

[14] "What is Flutter? - Flutter App Explained - AWS," *Amazon Web Services, Inc.*, 2018. Available: https://aws.amazon.com/what-is/flutter/. [Accessed: Nov. 18, 2024]

[15] "Dart overview," *Dart.dev*, 2024. Available: https://dart.dev/overview. [Accessed: Nov. 18, 2024]

[16] "Dart Features - Javatpoint," *www.javatpoint.com*, 2024. Available: https://www.javatpoint.com/dart-features. [Accessed: Nov. 18, 2024]

[17] A. Javed, "Dart Programming Language: Top Features and Applications," *Xavor Corporation*, Jul. 04, 2023. Available: https://www.xavor.com/blog/dart-programming-language-features-and-applications/#Darts_Sound_Null_Safety. [Accessed: Nov. 18, 2024]

[18] "Flutter Vs. Dart: Revolutionizing App Development," *Dhiwise.com*, 2024. Available: https://www.dhiwise.com/post/flutter-vs-dart-insights-into-the-future-of-app-development. [Accessed: Nov. 18, 2024]

[19] "FutureProvider | Riverpod," *Riverpod.dev*, 2024. Available: https://riverpod.dev/docs/providers/future_provider. [Accessed: Nov. 18, 2024]

[20] K. mistry, "Understanding Flutter Widgets: A Beginner's Guide to Building User Interfaces," *Medium*, Dec. 08, 2023. Available: https://medium.com/@kavyamistry0612/understanding-flutter-widgets-a-beginners-guide-to-building-user-interfaces-eed1acf17070. [Accessed: Nov. 22, 2024]

[21] "Material component widgets," *Flutter.dev*, 2024. Available: https://docs.flutter.dev/ui/widgets/material. [Accessed: Nov. 22, 2024]

[22] VALD, "MoveHealth," *Google.com*, 2021. Available:

https://play.google.com/store/apps/details?id=com.vald.movehealth. [Accessed: Nov. 29, 2024]

[23] Physitrack PLC, "PhysiApp®," *Google.com*, 2021. Available: https://play.google.com/store/apps/details?id=com.physitrack.physiapp&hl=en. [Accessed: Nov. 30, 2024]

[24] R. G. Team, "Rehab Guru Client," *Google.com*, 2021. Available: https://play.google.com/store/apps/details?id=com.rehabguru.client. [Accessed: Nov. 30, 2024]

[25] R. G. Team, "Rehab Guru Pro," *Google.com*, 2021. Available: https://play.google.com/store/apps/details?id=com.rehabguru.rehabguru. [Accessed: Nov. 30, 2024]

[26] Y. Shean, "Validating and customizing TextFormFields, the Flutter way," *Medium*, Mar. 30, 2023. Available: https://medium.com/@yshean/validating-custom-textformfields-the-flutter-way-182a7bb915a2. [Accessed: Jan. 02, 2025]

[27] GeeksforGeeks, "Flutter What is Future and How to Use It?," *GeeksforGeeks*, Apr. 06, 2023. Available: https://www.geeksforgeeks.org/flutter-what-is-future-and-how-to-use-it/. [Accessed: Jan. 02, 2025]

[28] Sven Kernebeck, T. S. Busse, M. D. Böttcher, J. Weitz, J. Ehlers, and U. Bork, "Impact of mobile health and medical applications on clinical practice in gastroenterology," *World Journal of Gastroenterology*, vol. 26, no. 29, pp. 4182–4197, Aug. 2020, doi: https://doi.org/10.3748/wjg.v26.i29.4182. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC7422538/?utm_source=chatgpt.com. [Accessed: Jan. 03, 2025]

[29] W. Peng, Shaheen Kanthawala, S. Yuan, and S. A. Hussain, "A qualitative study of user perceptions of mobile health apps," *BMC Public Health*, vol. 16, no. 1, Nov. 2016, doi: https://doi.org/10.1186/s12889-016-3808-0. Available: https://bmcpublichealth.biomedcentral.com/articles/10.1186/s12889-016-3808-0. [Accessed: Jan. 03, 2025]

[30] A. Ghose, X. Guo, B. Li, and Y. Dang, "Empowering Patients Using Smart Mobile Health Platforms: Evidence From A Randomized Field Experiment," *arXiv.org*, 2021. Available: https://arxiv.org/abs/2102.05506?utm_source=chatgpt.com. [Accessed: Jan. 03, 2025]

[31] C. A. Okolo, O. Babawarun, J. O. Arowoogun, A. O. Adeniyi, and R. Chidi, "The role of mobile health applications in improving patient engagement and health outcomes: A critical review," *International Journal of Science and Research Archive*, vol. 11, no. 1, pp. 2566–2574, 2024, doi: https://doi.org/10.30574/ijsra.2024.11.1.0334. Available: https://ijsra.net/content/role-mobile-health-applications-improving-patient-engagement-and-health-outcomes-critical

[32] P. Rana, "Mastering Image Picking in Flutter with ImagePicker Plugin," *Medium*, Mar. 19, 2024. Available: https://purvikrana.medium.com/mastering-image-picking-in-flutter-with-imagepicker-plugin-ee4d2429ad06. [Accessed: Jan. 03, 2025]

[33] R. Argent, A. Daly, and B. Caulfield, "Patient Involvement With Home-Based Exercise Programs: Can Connected Health Interventions Influence Adherence?," *JMIR mhealth and uhealth*, vol. 6, no. 3, pp. e47–e47, Mar. 2018, doi: https://doi.org/10.2196/mhealth.8518. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC5856927/?utm_source=chatgpt.com. [Accessed: Jan. 05, 2025]

[34] "Kotlin Help," *Kotlin Help*, 2023. Available: https://kotlinlang.org/docs/async-programming.html#callbacks. [Accessed: Jan. 05, 2025]

[35] http://hl7.org/fhir, "Overview-arch - FHIR v6.0.0-ballot2," *Fhir.org*, 2025. Available: https://build.fhir.org/overview-arch.html?utm_source=chatgpt.com. [Accessed: Jan. 06, 2025]