UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Grado en Tecnologías Específicas de Telecomunicación

Mención en Telemática

Aplicación Multiplataforma En Flutter Con Capacidad De Chatbot Para Proyecto De Telerehabilitación

Autor:

Dña. Alejandra Cecilia East Garijo

Tutores:

Dra. Míriam Antón Rodríguez

Dr. Mario Martínez Zarzuela

TÍTULO:	**Aplicación Multiplataforma En Flutter Con Capacidad De Chatbot Para Proyecto De Telerehabilitación**
AUTOR:	**Dña. Alejandra Cecilia East Garijo**
TUTORES:	**Dra. Míriam Antón Rodríguez**
	Dr. Mario Martínez Zarzuela
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**Míriam Antón Rodríguez**
VOCAL:	**David González Ortega**
SECRETARIO:	**Mario Martínez Zarzuela**
SUPLENTE 1:	**Carlos Gómez Peña**
SUPLENTE 2:	**Jesús Poza Crespo**

FECHA:	**Septiembre 2025**
CALIFICACIÓN:	

RESUMEN DEL TRABAJO FIN DE GRADO

En el presente Trabajo de Fin de Grado se ha desarrollado RehaBot, una aplicación multiplataforma basada en Flutter. Se ha diseñado para apoyar a la rehabilitación de personas con parálisis cerebral mediante sesiones de ejercicios dirigidos por terapeutas. La aplicación integra un sistema de Chatbot interactivo que guía a los pacientes en la realización de los ejercicios a través de vídeos e instrucciones simples. También permite la recopilación de realimentación en tiempo real con aspectos como claridad, dificultad, dolor percibido, utilidad, repetición y comentarios dando la posibilidad de añadir videos de los usuarios. También se ha implementado una línea de tiempo que organiza los videos realizados por los pacientes y facilita el seguimiento permitiendo a los usuarios ver su progreso de una forma clara. Además, se han incorporado funcionalidades de recompensa con el fin de fomentar la motivación y adherencia al programa. Este proyecto ofrece una solución complementaria a la rehabilitación presencial, dando acceso a programas personalizados y apoyando la continuidad del tratamiento a largo plazo.

PALABRAS CLAVE

Flutter, rehabilitación, chatbot, parálisis cerebral, aplicación móvil, multiplataforma.

ABSTRACT

This Bachelor's Thesis presents RehaBot a cross-platform mobile application developed with Flutter. It was designed to support the rehabilitation of children with cerebral palsy through therapist-directed exercise programs. The application integrates an interactive an interactive chatbot system which guides patients through their exercises with videos and simple instructions. It also allows for feedback collection in real time on aspects such as clarity, difficulty, perceived pain, usefulness, exercise repetition and comments with the possibility of adding a video of the user performing them. In addition, a timeline feature has been implemented which organises the user videos, with feedback, so as to easily track progress. Rewards elements have also been incorporated to enhance motivation and adherence to the program. This project serves as a supplementary solution to traditional rehabilitation, enabling access to tailored programs and supporting consistent long-term treatment.

KEYWORDS

Flutter, rehabilitation, chatbot, cerebral palsy, mobile application, multi-platform.

AGRADECIMIENTOS

Quiero expresar mi agradecimiento a mis tutores por su orientación y apoyo durante el desarrollo de este proyecto, así como al resto del equipo de RehaBot: Hichem y Daniel.

También quiero agradecer a todas las personas que han contribuido con sus conocimientos y apoyo haciendo posible la realización de este Trabajo de Fin de Grado.

Contents

Index of Figures

Index of Tables

1. Introduction

1.1. Motivation

Cerebral Palsy (CP) is a neurological disorder affecting movement, posture, and balance, occurring in approximately two to three out of every 1,000 live births. It results from brain injury due to multiple aetiologies and manifests in different movement disorders, primarily spasticity, which affects 80% of children with CP. These movement impairments can lead to secondary complications such as hip pain, balance difficulties and hand dysfunction [1].

Considering the challenges mentioned above, effective rehabilitation approaches are crucial to improving motor function and quality of life for these individuals. Over the years, home-based therapy programs have gained recognition as a valuable complement or alternative to centre-based rehabilitation. These programs allow children to engage in continuous training within a familiar environment, enabling parents to integrate therapy into daily routines. Furthermore, this approach enhances motor learning, increases training intensity and fosters greater parental involvement, ultimately strengthening the relationship of the caregivers with the health professionals [2].

Despite the possible advantages of home-based therapy, challenges may arise such as parental stress, compliance and a change in parenting roles: these issues must be carefully addressed to ensure successful implementation. To support this, the present project aims to develop a cross-platform application that facilitates therapist-directed, home-based rehabilitation.

This project is developed in the context of the project "*RehaBot: Smart Assistant to Complement and Assess the Physical Rehabilitation of Children with Cerebral Palsy in their Natural Environment",* financed by the Spanish Ministry of Science and Innovation (PID2021-124515OA-I00) between 2023 and 2025. The project's objective is to complement traditional neurorehabilitation treatment for paediatric patients with cerebral palsy with different tools such as chatbots, wearables, computer vision, artificial intelligence and virtual reality technologies. While the primary focus is on children, this solution has potential to be applied to adults with cerebral palsy or who have suffered a stroke. This approach provides a cost-effective, patient-centred solution while also enabling research into the effectiveness of both conventional and technology-based treatments in promoting motor learning transfer to daily living activities [3].

Initially, this was implemented through Telegram, offering a basic platform for interaction. However, this method presented limitations such as privacy and

security concerns, exposure to inappropriate content and lack of parental controls which we will explore in more detail later. Additionally, some parents voiced concerns over their children using Telegram, preferring a more secure and dedicated environment for therapy-related interactions. To address these concerns, this cross-platform approach ensures accessibility and usability across various devices, allowing families to seamlessly integrate therapy in their daily lives.

The app will offer structured sessions, video demonstration and real-time feedback, helping to improve engagement, adherence and overall effectiveness of therapy for children with CP. By transitioning from Telegram to a specialised application, the system will provide a more efficient, scalable and personalised experience to better meet the needs of both parents and children while ensuring a secure and dedicated space for therapy.

1.2. Objectives

The primary objectives of RehaBot, the cross-platform application for rehabilitation for children with cerebral palsy are:

1) **Support therapist-directed, home-based rehabilitation:** To facilitate structured, rehabilitation programs personalised by a qualified therapist that are tailored to each patient's needs, including video demonstrations and real-time feedback.
2) **Enhance accessibility:** To provide a multi-platform, user-friendly application that ensures accessibility on various devices, enabling seamless integration of therapy into their daily lives.
3) **Improve engagement and adherence:** To increase user engagement by offering interactive features that promote adherence to therapy routines such as progress tracking, reminders, motivational prizes as well as an intuitive interface that encourages participation.
4) **Provide a secure and dedicated environment:** To create a secure, dedicated platform for therapy, addressing privacy concerns and offering a space free from any inappropriate content or distractions found in messaging platforms.
5) **Allow for scalability and future integration:** To ensure the scalability of the platform, including the possibility of integrating modern technologies such as AI for advanced features such as video analysis to assess performance, identify patterns and provide further insights into the process. This allows for continuous innovation and improvement to meet the evolving needs of patients with CP.
6) **Foster parental involvement and empowerment:** To encourage parental participation in the process by providing tools and resources that empower caregivers to support the healthcare professionals. Features such as a progress

timeline and user feedback on sessions will allow for development monitoring and higher motivation.

7) **Improve the quality of therapy:** To deliver high-quality, personalised rehabilitation that is adapted to each child's progress, allowing therapists to monitor outcomes remotely and make necessary adjustments in real time.
8) **Promote long-term success:** To guarantee the sustainability and long-term success of programs and patients by enabling continuous updates, tracking progress over time and ensuring that the therapy remains relevant and effective.

In this document, Chapter 1 introduces the motivation and objectives behind the development of RehaBot, highlighting the need for effective home-based rehabilitation for children with CP and the goals of the dedicated, cross-platform application. Chapter 2 provides the background and the state of the art with an overview of CP rehabilitation, home-based rehabilitation, cloud-based messaging APIs, cross-platform development frameworks and a detailed discussion about Flutter. Chapter 3 presents the materials and methods used in this project, covering the transition from Telegram to a dedicated application, prototypes and chatbot selection, project development, features and user experience and interface design. Chapter 4 describe the testing and evaluation process. Finally, Chapter 5 concludes the document with a summary of finding and potential future work.

2. Background and State of the Art

2.1. Current State of Cerebral Palsy (CP) and Rehabilitation

Cerebral Palsy (CP) defined as *"an umbrella term for a group of disorders of movement and posture, caused by a non-progressive interference in the developing brain."* [4, p. 1495], is the most common cause of physical disability in childhood, affecting approximately 2.1 per 1000 live births globally. It involves not only motor impairments but also associated challenges including sensory, cognitive, communication and behavioural difficulties.

Recent studies have shown a slight but consistent decline in CP in many high-income countries due to improvements in neonatal and perinatal care. However, in low and middle-income countries, where access to such care is limited, the prevalence remains stable. This is supported by population-based studies in low-income regions such as Uganda, Bangladesh and India among others [4]. In contrast, *Figure 1* illustrates the temporal trends for high-income regions where a downward trend is observed in most.

Figure 1: Global prevalence of cerebral palsy by region. Adapted from McIntyre et al.[4]

It is known that early diagnosis, facilitated by neuroimaging and standardised assessment tools, is increasingly prioritised. These methods enable timely intervention during periods of increased neuroplasticity. Physiotherapy plays a crucial role in the management of CP, aiming to enhance movement, increase strength and prevent muscles from weakening or becoming shortened [5]. Evidence also suggests that functional, goal-oriented approaches, are effective in improving motor functions in children with CP [6].

Despite these advances, significant barriers remain, particularly for children in remote or underserved areas. Delays in accessing physiotherapy can lead to irreversible harm and long-term consequences [7]. To address these problems, there is a growing interest in home-based rehabilitation technologies, which offer higher accessibility, continuity of care and increased patient engagement. The development of tools like the RehaBot app align with this need, providing both a flexible and scalable solution for these children and their caregivers.

2.2. Home-Based Rehabilitation for CP

Home-based rehabilitation programs are becoming increasingly vital in the management of Cerebral Palsy (CP), particularly in conjunction with traditional centre-based therapies. This rehabilitation from home provides a practical solution for families by offering continuous training, personalised care and it also encourages greater involvement from parents or caregivers.

Therapy at home offers several advantages: One of the key benefits is the continuity of care, allowing the patients to maintain progress between clinical sessions [8]. These programs, especially when therapist led, can be tailored to a child's daily routine and environment giving a more personalised and flexible approach compared to strictly institutional settings [5].

Additionally, these types of programs strengthen active parental involvement, which has been proven to enhance outcomes. An example of this can be seen during the COVID-19 pandemic where parents had to take on the role of informal therapists. Although challenging, this provided insight into how routine involvement improves family bonding as well as engagement in therapy [9].

Another clear benefit of home-based therapy is increased access to rehabilitation services. Access, defined as the availability, coverage and use of services is often constrained by many factors. Bright et al. [10] identified key barriers which limited this access including geographic distance to centres, affordability of services, transportation issues and the time required to travel and attend therapy sessions. These challenges significantly reduce the likelihood of adherence to physical

therapy; home-based solutions help mitigate these risks and promote equitable service distribution.

In spite of these benefits, families can also face barriers to implementing home-based rehabilitation effectively, one of these barriers is parental stress. Many caregivers feel unprepared to conduct sessions and worry whether they are delivering the exercises correctly [9]. Unfortunately, stress can lead to inconsistent practice and reduced adherence to therapy.

As noted by Lillo-Navarro et al. [8], *"Participants reported that regular adherence monitoring by health professionals allowed them to voice the problems that they had complying with the exercise program."* This emphasises the importance of ongoing support and monitoring to maintain engagement.

As we can see in *Figure 2* there are multiple factors that influence adherence to home exercise programs such as parental confidence, exercise preference and routine integration. The RehaBot app developed in this project incorporates features designed to tackle these issues. These features include calendar reminders, a structured timeline to track the patient's progress, feedback mechanisms and progress milestones. These elements not only guide families through each session but enhance accountability, motivation and the sense of progress thus increasing long-term engagement and better therapeutic results.

Theme	Subtheme	Categories of coded statements
Characteristics of exercises and home programs	Preference for exercises	• Exercises that improve outcomes • Enjoyable exercises • Exercises without adverse effects (pain, discomfort) • Non-complex exercises (without the need for technical skills)
	Amount of exercises	• Time consumption • Disruption of the affective or recreational family relationship • Excessive burden, according to the real needs of family
Physiotherapist's teaching style	Building parents' confidence in implementing the exercise program	• Demonstrating exercises with the child • Providing feedback • Providing written instructions • Providing information and support to parents
	Helping incorporation into daily routines	• Giving reminders to incorporate into daily routines
	Incentivising adherence	• Perception of achievements • Incentives based on goals • Changes in the child's exercise performance • Gaining peace of mind
	Monitoring and giving support to adherence	• Perception of regular monitoring of home program

Figure 2: Summary of factors perceived to influence adherence to home exercise programs (adapted from Lillo-Navarro et al., [8].

2.3. Cloud-based Messenger APIs

Cloud-based messenger APIs allow real-time, asynchronous communication between users and systems. Typically, these systems have a cloud-based server

and an API, with the API serving as the bridge between the server and the user application.

The cloud-hosted server handles the core messaging service such as storage, routing and delivery, security and scalability whereas the API (*Application Programming Interface)* exposes endpoints that allow developers to perform operations like sending and receiving messages, using push or pull models, managing groups or channels, uploading media and Bot actions. Client applications primarily interact with the API through HTTPS requests.

Bots, short for robots, are software built to carry out automated, unsupervised, repetitive tasks. Some cloud-based messenger APIs support their integration, extending their capabilities to include automation, personalised responses and data collection. Chatbots are a subclass of bot that automate tasks by interacting with users through conversation. Some of the most notable platforms which integrate these chatbots are Telegram, WhatsApp, Facebook Messenger and Discord.

Telegram

Telegram is a cloud-based messenger which is a globally accessible and free service. Telegram has cross-platform support and, as of March 2025 according to Telegram, has 1 billion active [11]. This demonstrates its widespread adoption and role as a major communication tool in addition to its scalability and ease of access. Telegram offers features such as large file sharing, bot integration as well as other functionalities which make it suitable for various uses such as educational and therapeutic applications.

Telegram bots are a particular type of account in Telegram created by third-party developers which can take user inputs, process them and display results. In addition, they can deliver updates and notifications. These bots operate using the Telegram Bot API, which allows developers to interact with Telegram servers using HTTPS requests. It supports a range of operations such as sending and receiving text, videos and documents as well as interactive elements. These bots can be used in direct conversations with users or by broadcasting information.

Telegram is a popular choice for developers when creating chatbot applications as it is a well-documented, scalable and a simple API to use. However, it may be less suitable depending on the region and the use of Telegram and the lack of end-to-end encryption for cloud chats. It is only encrypted during transit which could raise concerns when handling health related information.

WhatsApp

WhatsApp is a free cloud-based messaging service owned by Meta, it is renowned for its end-to-end encryption and integration with mobile devices. According to Statista [12], as of 2025, WhatsApp has over 2.9 billion monthly users, making it the most used messaging platform worldwide (*Instant Messaging | Definition, History, & Facts | Britannica*, 2025).

WhatsApp included chatbot integration with the WhatsApp Business API. Meta [13] states that using this cloud API, hosted by Meta, allows businesses to connect thousands of customers with both programmatic and manual communication.

The API can also be integrated with various backend systems like CRM and marketing platforms. It allows requests using HTTP endpoints abstracting complexities from developers. It includes features like automated message flows, session-based messaging, template broadcasts and interactive elements. WhatsApp does have stricter access controls making businesses register and undergo verification while complying with WhatsApp's policies.

With this Applications global reach and strong security features it could serve as an effective platform for a chatbot like RehaBot, in particular for sending reminders, updates and automated support in a familiar environment. On the other hand, it may be limited by the need to verify businesses, potential usage cost and, as of recently, the announcement of plans to introduce advertising; all this could reduce flexibility and expose users to unwanted content.

Facebook Messenger

Facebook Messenger is a highly used cloud-based messaging platform which forms part of larger Facebook ecosystem. In January 2025 this platform had roughly 947 million active monthly users [14] making it widespread and generally accepted.

In 2016, the Messenger Platform API allowed for bot integration via a webhook-based architecture. Its three main capabilities are a send/receive API, generic message template and a welcome screen with Null state CTAs (*Call to Action*). It also includes natural language assistance through wit.ai [13].

This ecosystem of a messenger app with bot capabilities along with the social media aspect of Facebook facilitates customer service automation, reminders and informational bots making them an effective marketing tool.

Given Messenger's large user base and extensive features, it could be a viable chatbot channel for a health-based application. On the contrary, Facebook has declining popularity among younger generations, ongoing privacy concerns due to

Meta's data handling policies as well as a minimum age requirement of 13 years to create an account.

In conclusion, Cloud-based messenger APIs provide powerful, scalable and flexible infrastructures for building chatbot applications, each with their own strengths and limitations. Telegram stands out for its openness, simplicity as well as its end-to-end encryption in cloud chats create restrictions for more sensitive contexts. WhatsApp offers incredible reach and robust security, however, stricter business verification and costs limit flexibility. Facebook Messenger, which combines bot capabilities with social media integration is very effective for engagement and customer services but faces challenges with declining popularity and continuing privacy concerns. While they all demonstrate potential as chatbots their constraints highlight the value of a dedicated solution to ensure greater control over cost, functionality, user experience and data management while maintaining flexibility to incorporate features without being bound by the restrictions of external platforms.

2.4. Overview of Cross-Platform Development and Comparison

Cross-platform mobile development refers to an approach used in software development where a single mobile application is created with the capability to be compiled or exported to multiple operating systems. This system has the advantage of reducing redundancy by allowing the creation of a single codebase which is run on various platforms. This approach is becoming more important in the context of mobile health applications where efficiency, broad accessibility and consistent user experiences are critical. It avoids the need to develop separate apps for Android and iOS reducing development time and cost while enabling rapid updates and better scalability [15].

According to recent data, the combined market share of iOS and Android amounts to approximately 99.2% as depicted in *Figure 3*. Therefore, to effectively access most of the mobile device market it is necessary to be present on both platforms.

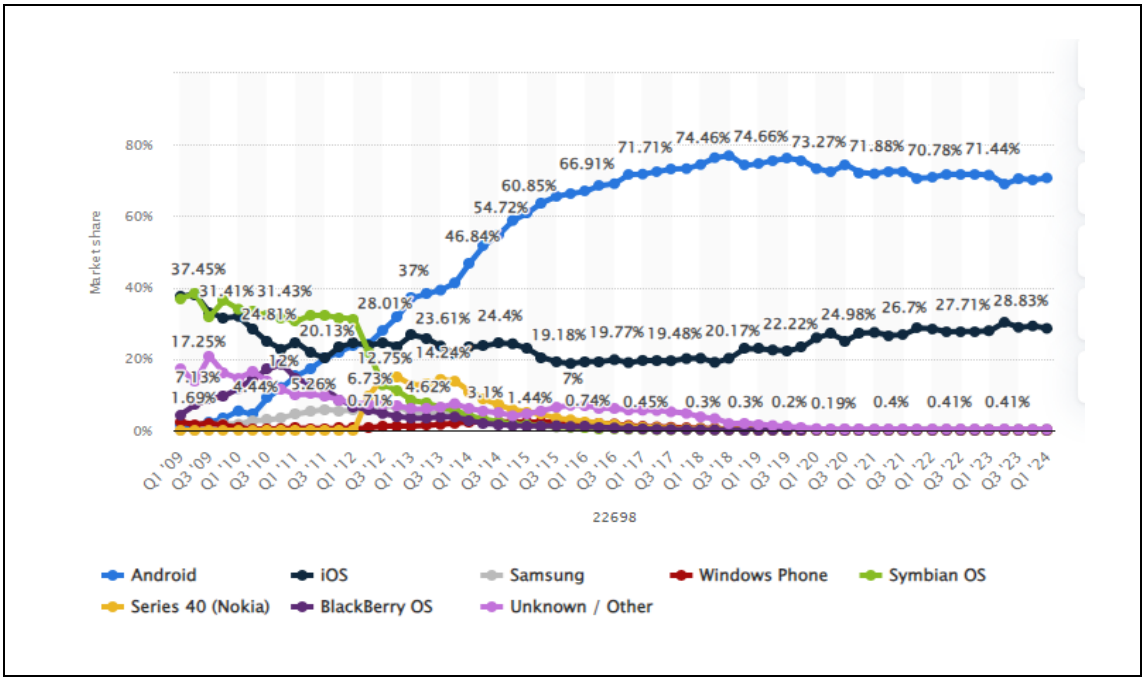


Figure 3: Market share of mobile operating systems worldwide from 2009 to 2024. Adapted from Statista[15].

These types of applications have many notable benefits:

- **Higher target audience exposure:** By building a single application, suitable for different platforms developers can target a much broader range of people.
- **Reduced development cost:** Cross-platform apps follow the principle "write once, run everywhere". This idea of reusable code and agile development can significantly reduce costs.
- **Easier maintenance:** As there is only one application it increases the ease of updating and fixing any issues that may arise.
- **Faster development:** Due to the single source code development times are substantially reduced.
- **Reusable code:** Code can be utilised for the different targeted platforms saving time and resources.

There are multiple frameworks available to develop cross-platform applications the most notable, as of 2023, being Flutter, React Native, Cordova, Unity, Ionic and Xamarin as we can see in *Figure 4*.

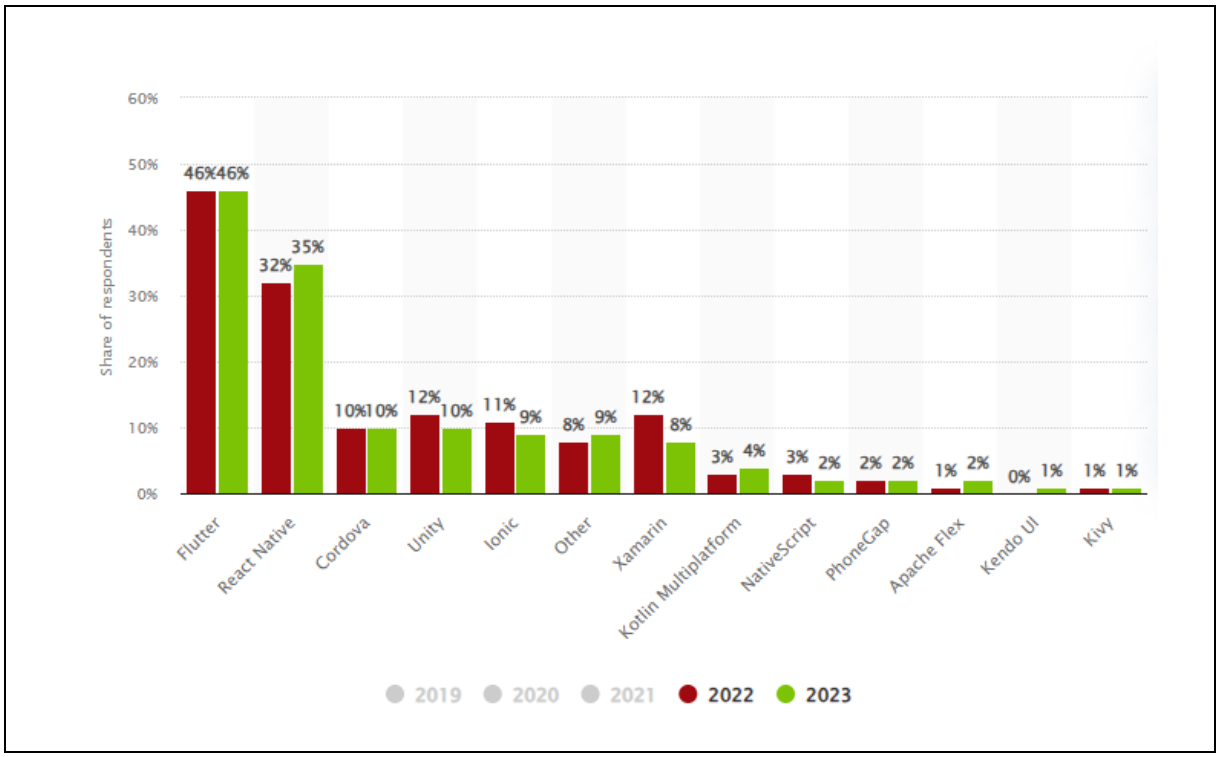


Figure 4: Cross-platform mobile frameworks used by software developers in 2022 and 2023. Adapted from Statista [16] based on data from JetBrains.

Flutter

Flutter is a cross-platform development framework created by Google which allows developers to create applications in various operating systems using a single codebase. Flutter is developed using Dart an object-oriented, class-based, garbage collected language with a C-style syntax; it also supports interfaces, abstract classes and type inference among other things [16]. A more detailed exploration of Flutter's capabilities and architecture will be presented in upcoming sections.

Some well-known apps created with Flutter include Google Ads, Google Pay and New York Times.

React Native

React Native is an open-source UI software framework developed by Meta Platforms first released in 2015. It allows developers to create applications with JavaScript and React. According to Adam Boduch, the goal of React Native is *"React components everywhere, not write once run everywhere"*.

React Native is different from other cross-platform frameworks as it is able to render real native components which gives it an improved performance compared to a more traditional hybrid framework such as Cordova. This framework uses a bridge translating JavaScript into native API calls allowing for up to 90% of code being reused.

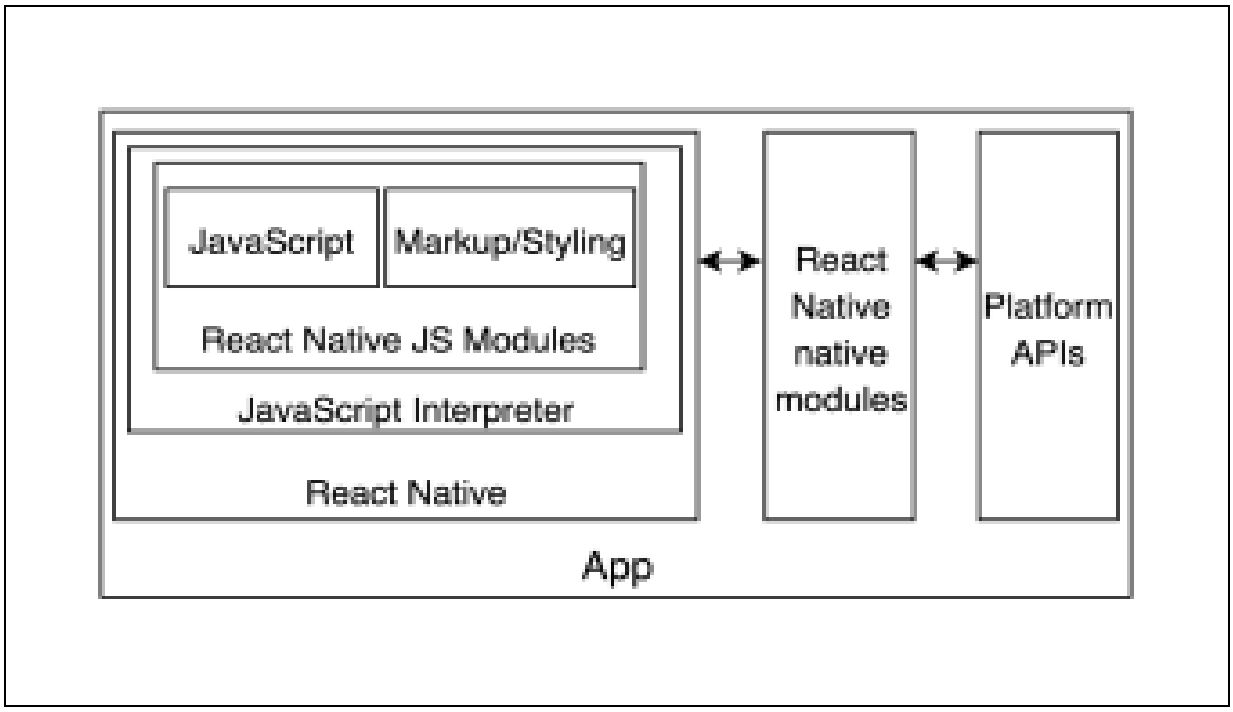


Figure 5: React Native/interpreted approach architecture. Adapted from Bridging the gap: Investigating device-feature exposure in cross-platform development by A. Biørn-Hansen and G. Ghinea, [18]

React Native has gained widespread adoption in e-commerce to healthcare because of its active community, its large ecosystem of plugins and hot reloading which make development much faster. Some of its limitations include performance bottlenecks for graphically intensive applications and the need for native development knowledge when creating a custom module. It is a compelling choice for apps that need both speed and versatility.

The most important applications created with React Native are Instagram, Facebook, Walmart and Airbnb.

Apache Cordova

Apache Cordova, previously known as PhoneGap is a mobile application framework created by Nitobi at an Apple software development camp in 2008 and then acquired by Adobe. Currently, it is an open-source framework as it was donated to the Apache Software Foundation. Cordova allows developers to convert HTML, JavaScript and Cascading Style Sheets into a native application that you can run on iOS, Android, Windows and Electron [17].

It uses a native wrapper around a WebView allowing access to many native device APIs through a series of plugins. The plugin architecture, as seen in *Figure 6,* consists of JavaScript code, a native bridge interface and the corresponding native implementation which allows interaction with features such as the battery status, camera, contacts, geo-localisation, Media and Network Information among others [18].

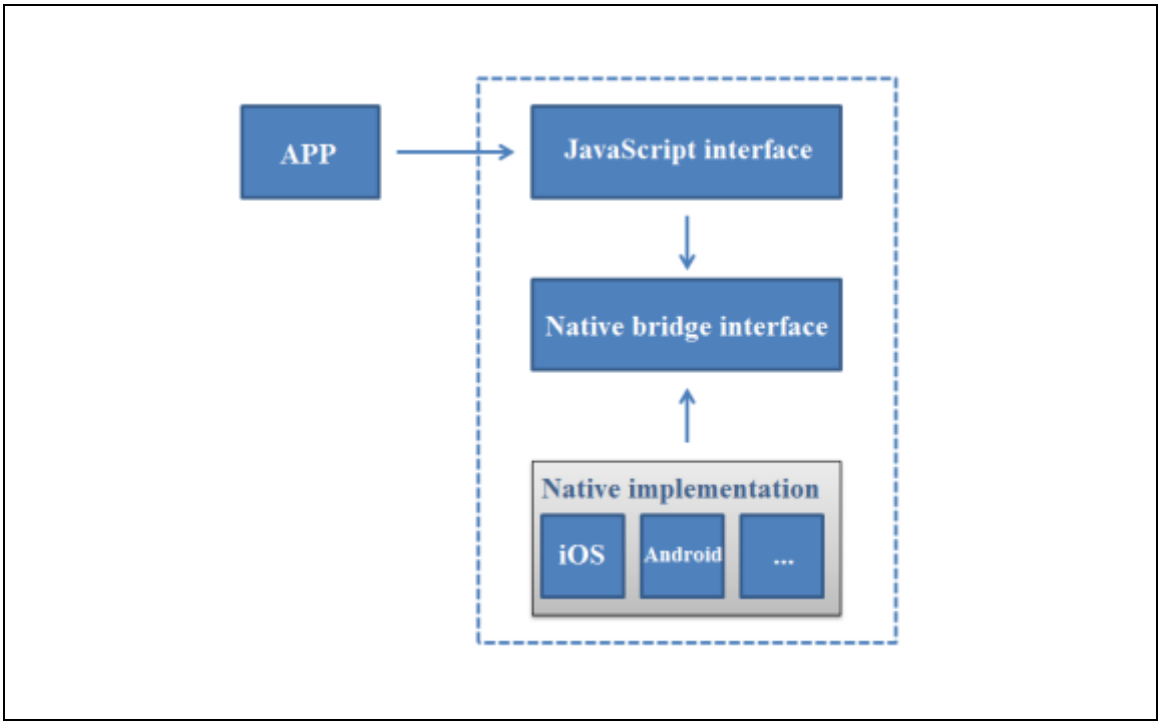


Figure 6: Plugin architecture. JavaScript. Adapted from The development of hybrid mobile applications with Apache Cordova [20].

Cordova is most suitable for building simple to moderately complex applications that do not require high-performance graphics or complex animations. Examples include administrative tools, content-driven apps or applications with functions similar to a website. It is particularly advantageous if you need to develop an application quickly at a low cost. However, as Cordova uses a WebView to render content rather than a native UI component it may have performance issues with more demanding application or those that need a highly responsive interface.

Some well-known apps built using Apache Cordova are Khan Academy, Duolingo, Robinhood and MyFitnessPal.

Unity

Unity is a multi-platform game engine created by Unity Technologies, first released in 2005. It uses C# as its primary programming language allowing developers to build 2D and 3D applications. It was initially created with a focus towards games, however, due to its flexibility and rendering capabilities it is now used in multiple industries such as education, architecture, automotive design and healthcare [19].

Unity lets developers create immersive and interactive applications across multiple platforms including Android, iOS and Windows. Using the Unity Editor developers are able to manage assets, script interactions, animations and simulate physics. One of its main strengths is the component-based architecture allowing for reusable scripts and behaviours that can be attached to game objects, simplifying complex developments [20].

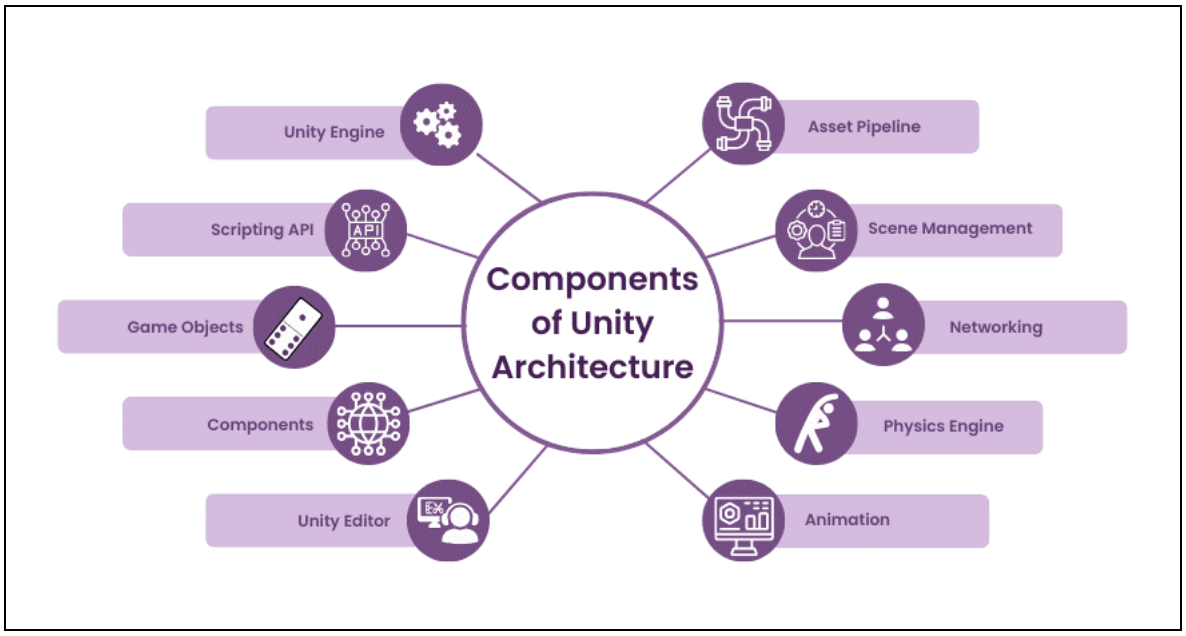


Figure 7: Figure X. Unity architecture components. Adapted from What are the components of Unity Architecture?, by The Knowledge Academy, [21].

Another key feature is Unity's real-time 3D rendering capabilities, making it suitable for augmented reality and virtual reality. This has led to its use in rehabilitation and patient engagement. Unity has extensive documentation, a large developer community and access to assets through the Unity Asset Store which help with efficient development and rapid prototyping.

Despite its many advantages Unity is not suited to standard user interface-based apps due to its large build size and performance overhead.

Some of the most important apps developed in Unity are Pokémon Go, Among Us and NASA's Virtual Mars Rover Experience.

Ionic

Ionic is an open-source UI toolkit used to build multi-platform applications and was founded in 2012 by Max Lynch and Ben Sperry. Ionic is written using web technologies such as HTML, CSS and JavaScript and are primarily built using Angular although it also supports React and Vue [21].

Ionic is a hybrid development model rendering the app's UI within a WebView and enabling access to native device feature through Capacitor. Capacitor acts as a bridge that allows developers to write native code and call it from JavaScript improving performance and flexibility [22].

One of the main advantages of Ionic is its pre-designed UI components that follow modern design standards such as Material Design and Cupertino. These allow developers to create visually appealing and responsive interfaces with less effort.

Ionic's integration with Angular and its support for progressive web apps make it a good choice for those familiar with web development.

As a hybrid framework, Ionic can encounter performance issues in graphically demanding applications or if there is high-frequency sensor input. Ionic's use of WebView means that although application will look and behave similarly across platforms, their performance against apps built with native UI components will be poorer.

Popular applications built with Ionic include Sworkit, Shipt and McDonald's Turkiye.

Xamarin

Xamarin is an open-source platform developed by Xamarin Inc., later acquire by Microsoft in 2016. With Xamarin, developers can create native mobile applications for Android, iOS and Windows using `.NET` and `C#` [23]. Xamarin lets developers share a large portion of code across all platforms, on average about 90%, meaning that they can write all their business logic in a single language but achieve native performance, look and feel, following the principle of "write once, run everywhere".

Xamarin uses native compilation attributing to its high performance and native user experience. As shown in *Figure 8*, Xamarin uses a common .NET codebase which communicates with native SDKs through bindings for `Xamarin.iOS` and through the Managed Callable Wrapper and Android Callable Wrapper for `Xamarin.Android`. This architecture allows access to most device features such as sensors, GPS, cameras and file systems.

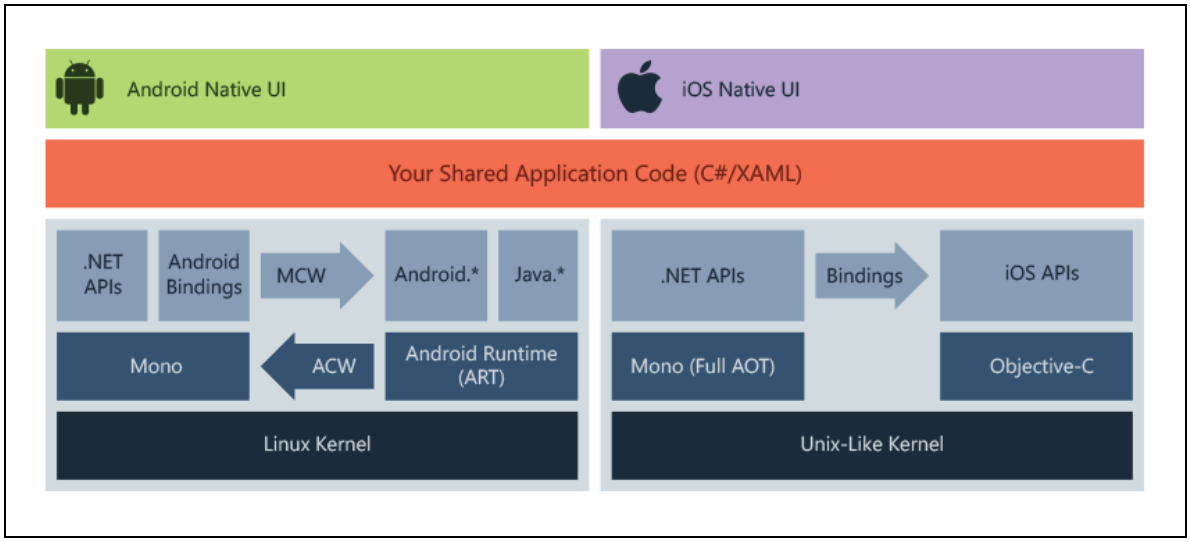


Figure 8: How Xamarin works: Architecture overview for cross-platform development. Adapted from What is Xamarin?, by Microsoft, [25]

Despite its many advantages, Xamarin also has some limitations. The initial learning curve can be difficult for those not familiar with `.NET` or `C#`. Additionally, Xamarin apps can also be larger in size compared to those developed using native SDKs.

Another key issue is Xamarin's declining support with the introduction of `.NET MAUI` (Multi-platform App UI) as it's official successor. This means that Xamarin has not received updates beyond May 2024 making it less future proof for new projects [24].

Some of Xamarin's most notable applications are UPS, BBC Good Food and Alaska Airlines.

Comparison

	Flutter	React Native	Cordova	Unity	Ionic	Xamarin
Language	Dart	JavaScript + React	HTML/CSS/ JavaScript	C#	HTML/CSS/ JavaScript (Angular/ React/ Vue)	C# + .NET
Rendering	Native compiled widgets	Native components via bridge	WebView	Game engine 3D/2D	WebView + capacitor	Native UI rendering
Performance	High	High	High	Medium	High	Medium
Multiplatform	✅	✅	✅	✅	✅	✅
Ease of Use	Medium	High	High	Medium	High	Medium
Speed of Development	Fast	Fast	Fast	Slow – Large builds	Fast	Medium
App Size	Large	Medium	Small	Very Large	Medium	Medium-Large
Open Source	✅	✅	✅	✅ (free version)	✅	✅
Resources	Good + growing	Extensive	Limited – Legacy	Strong	Good	Good
Access to Native Features	✅	✅	⚠ (via plugins)	✅	⚠ (via Capacitor)	✅

Limitations	Larger app size. Newer ecosystem	Occasional need for native code	Poor performance. Outdated plugins.	Heavy build	Not ideal for intensive tasks.	Difficult to learn. Shrinking support.
Best use cases	Complex UI. Animations. Mobile-first apps.	Standard apps	Simple apps. Rapid prototyping. Like web page. Limited budget.	Games. AV/VR.	UI-rich apps. Prototyping.	Business apps. Backend Integration.

Table 1: Comparison of cross-platform development frameworks.

As seen in the table above, each cross-platform framework presents its individual advantages and disadvantages depending on the use case, amount of development experience and performance requirements. When selecting a framework for a mobile rehabilitation application like *RehaBot* it is essential to consider factors such as consistent performance, native-like UI, ease of development, scalability and community support.

Taking all pros and cons into account Flutter emerged as the most suitable choice for RehaBot for several reasons:

1. **Native performance:** Flutter compiles directly to native ARM code using Dart, allowing for a smooth and consistent experience. This is essential for an application like *RehaBot* which relies on video streaming, smooth UI transitions and real-time user feedback.
2. **Rich UI capabilities:** Flutter offers a wide range of customisable widgets and supports complex UI designs making it idea for engaging and accessible user interfaces. The “everything is a widget” idea gives precise control over UI and layout.
3. **Single codebase across platforms:** Using Flutter *RehaBot* will be able to maintain a single codebase for both Android and iOS reducing development and maintenance overhead. This supports one of the goals which is broad accessibility, in particular for those users who may already face barriers when accessing physical therapy.
4. **Growing ecosystem and community support:** Flutter has a rapidly growing community, extensive documentation and an increasing number of packages and plugins. This accelerates development and troubleshooting allowing for evolution and scalability.

5. **Integration with video and multimedia:** *RehaBot* relies heavily on video interactions between therapist and patient. Flutter provides robust support for multimedia which are essential for the development of this application.
6. **Developer productivity:** Features like hot reload, integrated development tools and support for modern development paradigms enhance productivity. This allows faster iterations and rollouts.

In contrast, frameworks like React Native and Xamarin require more platform specific adjustments for a truly native experience. Cordova and Ionic, while easier to use depend heavily on WebView rendering which, as we have seen, can cause performance issue particularly problematic for apps with real-time feedback or animation like *RehaBot*. Unity, though extremely powerful, is more suited to game development and heavy graphical applications.

In conclusion, Flutter was believed to offer the best balance of performance developer efficiency, visual fidelity and community support for the development of *RehaBot*: A multi-platform rehabilitation assistant application intended to be scalable, responsive and user-friendly in both clinical and home environments.

2.5. Flutter

Flutter is an open source, user interface, cross-platform development framework created by Google and initially released in 2017. Its versatility lies in its capacity to create applications for multiple operating systems including Web, Android, iOS, Linux, macOS and windows from a unified codebase. It combines a high-performance rendering engine with a declarative user interface language (Dart) to create smooth and expressive applications.

2.5.1. Layered Architecture

One of the most important features of Flutter is its architectural model as seen in *Figure 9*. It is designed to be flexible, extensible with high performance. The architecture consists of 3 main layers:

- **Framework Layer:** Built in Dart, this is the layer most developers interact directly with. It provides the foundation for building applications with essential libraries, gesture handling, animation as well as rendering. The framework also includes two sets of design widgets: Material Design, for Android Apps and Cupertino for iOS apps. This allows developers to create UIs that align with both ecosystems while sharing a codebase.
- **Engine Layer:** Written primarily in `C++`, the engine layer provides low-level core functionality. It includes the `Skia 2D` graphics engine for rendering, the Dart runtime for JIT (Just-in-Time) and AOT (Ahead-of-Time) compilation, text

layout engines and accessibility support. This layer is crucial for Flutter to render every pixel directly, bypassing platform native UI elements and giving the developer full control over the app's appearance.

- **Embedder Layer:** This platform specific integration layer allows the Flutter engine to connect with the underlying operating system. It is responsible for input event handling, setting up the rendering surface and managing events and lifecycle changes such as launching, backgrounding and closing. There are different embedders available for Android, iOS, desktop systems and even for some embedded devices making Flutter extremely versatile.

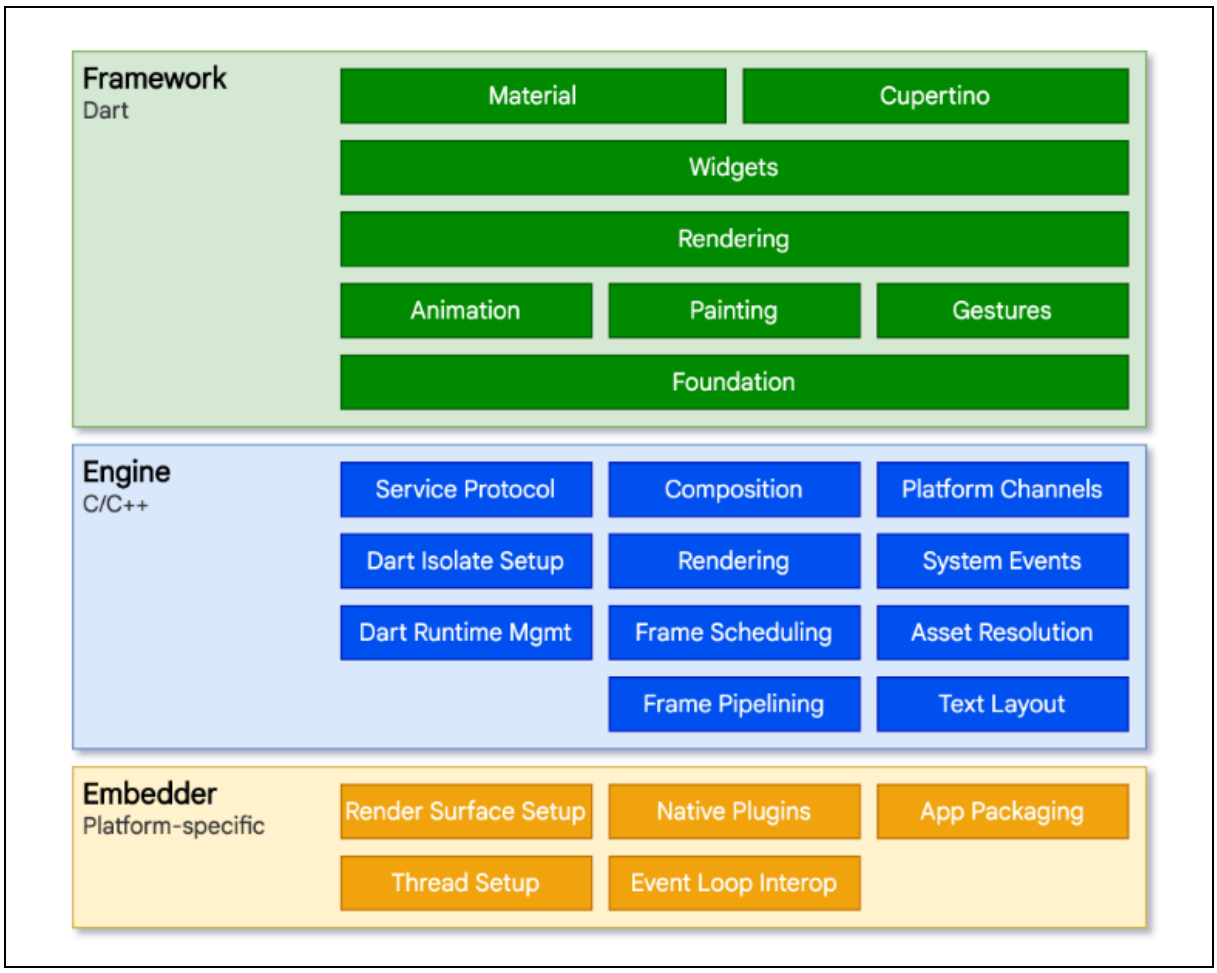


Figure 9: Flutter architectural overview. Adapted from Flutter [27]

Flutter App Composition

As seen in *Figure 10,* a Flutter App generated with `flutter create` involves several constructs:

- **Dart App:** Composes widgets into the UI, implements business logic and is built by the developer.

- **Framework:** It provides a higher-level API to manipulate widgets, handle gestures, accessibility and manage text input. It also composes the widget tree into a scene.
- **Flutter engine:** The engine is responsible for rasterising composited scenes, low-level implementation of Flutter's core APIs, exposes functionality to the framework using `dart:ui` and integrates into the underlying platform using the Embedder API.
- **Embedder:** The embedder interfaces with the underlying OS so as to access services like rendering, accessibility and input, it manages the event loops while exposing platform-specific API to integrate the Embedder into applications.
- **Runner:** This element composes the parts exposed by the platform-specific API of the Embedder into an app package which can then be run on the target platform. This part is also owned by the app developer.

Testing, modularity and code reuse are improved with this type of layered separation.

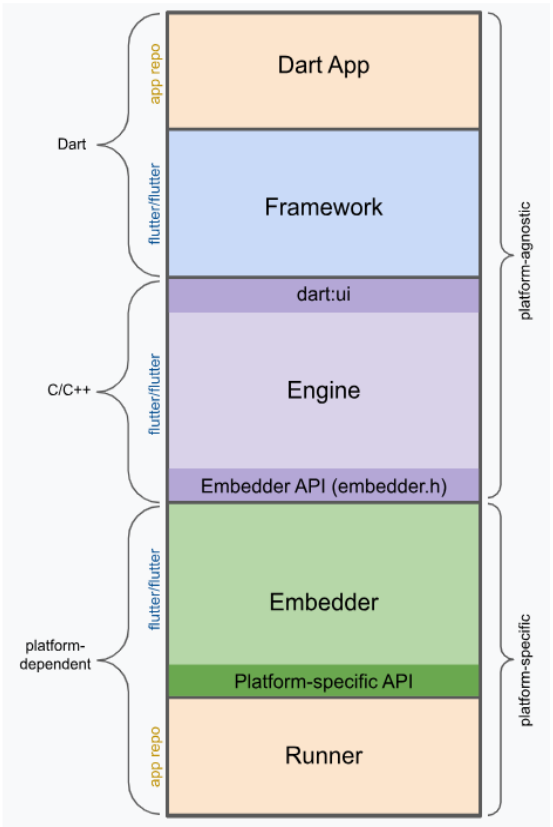


Figure 10: Flutter App Anatomy. Adapted from Flutter [27]

2.5.2. Reactive User Interfaces

Flutter has a reactive, declarative approach to user interface development whereby the interface is expressed directly as a function of the application state. When the state changes, the framework will automatically update the interface, this feature was influenced by Facebook's React framework.

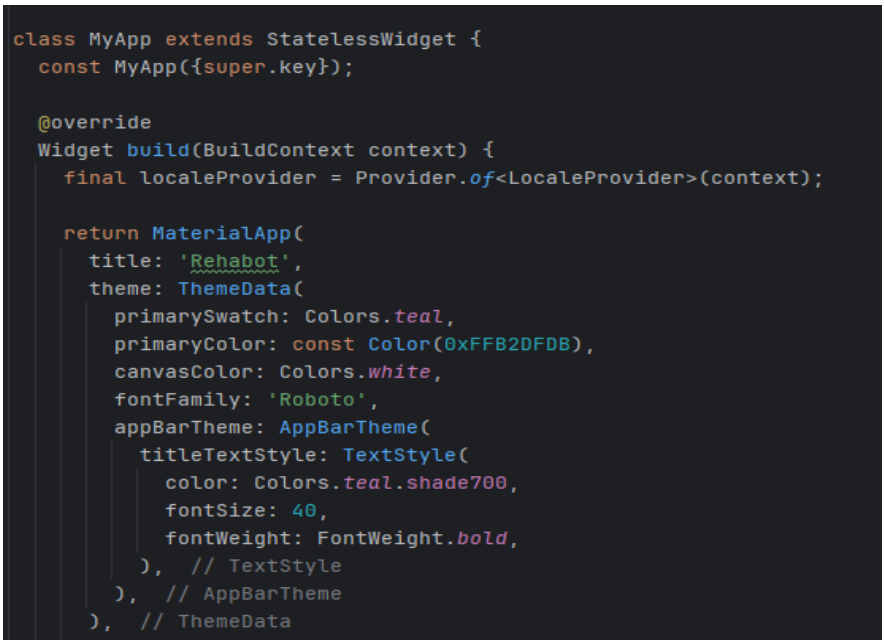
Traditionally developers have had to manually update elements in response to events, which creates complex error-prone applications because of interdependent parts. A key example of this is shown in Flutter Docs [25] when a change in a colour picker's hue slider should also update previews, swatches and selection controls creating a cascade of updates which can introduce bugs or inconsistencies. In contrast, reactive frameworks like Flutter abstract this idea by decoupling interface description from the application logic. A developer only declares what the UI should like for any given state, while the framework ensures consistency during updates.

Flutter uses widgets, similar to components in React, which are represented by immutable classes used to configure a tree of objects. These widgets then manages another tree of objects for layout which, in turn, is used to manage a tree of objects for compositions. In its essence, Flutter can be seen as a series of mechanisms which convert trees of objects into lower-level trees of objects and then propagating the changes from one to another. Each widget effectively overrides the `build()` method to declare its user interface.

2.5.3. Widgets: Flutters Building Blocks

The concept of Widgets is fundamental to understanding Flutter and how user interfaces are constructed. Widgets are the smallest unit of composition within a Flutter app, giving immutable declarations of discrete parts of the interface. Every element of this interface, from structural elements like rows and columns to visual controls, including buttons and sliders, is represented as a widget.

Widgets in Flutter are composed in a hierarchical manner forming a widget tree. Each widget is nested within its parent gaining access to its parent's contextual information. Every widget tree originates from a root widget, which hosts the Flutter application, commonly `MaterialApp`, as shown in *Figure 11*, or `CupertinoApp` and extends downward to the smallest leaf nodes [26].



```dart
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    final localeProvider = Provider.of<LocaleProvider>(context);

    return MaterialApp(
      title: 'Rehabot',
      theme: ThemeData(
        primarySwatch: Colors.teal,
        primaryColor: const Color(0xFFB2DFDB),
        canvasColor: Colors.white,
        fontFamily: 'Roboto',
        appBarTheme: AppBarTheme(
          titleTextStyle: TextStyle(
            color: Colors.teal.shade700,
            fontSize: 40,
            fontWeight: FontWeight.bold,
          ),  // TextStyle
        ),  // AppBarTheme
      ),  // ThemeData
```

Figure 11: Code snippet from RehaBot declaring the root widget.

Whenever an event occurs, for example a user interaction, the application updates the interface by replacing widgets in the tree. Its framework can compare old and new widgets, only changing what is necessary in an optimised and efficient manner. This allows for intuitive development as it is only necessary to describe the interface in function of the current state.

A notable distinction in Flutter is that each user interface control has an implementation in Dart, rather than relying on native controls. This has several benefits:

- Extensibility: Developer can freely create or modify controls without constraints of the underlying UI.
- Performance: Performance is improved as Flutter composites the entire scene in one go without transitions between Flutter and native platform code.
- Decoupling: The separation of application behaviour and OS dependencies ensure the app has the same look and feel independently of the underlying OS version.

Widgets in Flutter are typically built by composing many smaller, single-purpose widgets. The philosophy is that, where possible, the number of design concepts is as reduced as possible while allowing the total vocabulary to be large. We can illustrate this idea with the Container widget, which is composed of multiple widgets itself for example `ConstrainedBox`, `Align` and `Padding` among others. This

approach enables highly customised UI elements by combining existing widgets or by building new ones based on source code of those already established [25].

A widget's visual representation is determined by overriding the `build()` method which returns a new element tree. This function should be free of side effects, regardless of previous calls, it must return a fresh tree each time the framework requests a rebuild supporting swift, interactive applications.

To illustrate how widgets are composed in practice, consider *Figure 12* from the RehaBot app. In this snippet, a `Scaffold` provides the structural layout, while `ListTile` widgets manage user interaction such a picking a time and adding an event to a calendar. The widgets work together declaratively with their visual state reflecting the underlying application state.

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: Text('Add Event')),
    body: ListView(
      children: [
        ListTile(
          title: Text('Pick Event Time'),
          subtitle:
              selectedTime != null
                  ? Text(selectedTime!.format(context))
                  : Text('No time selected'),
          trailing: Icon(Icons.access_time),
          onTap: _pickTime,
        ),  // ListTile
        ListTile(
          title: Text('Add to Calendar'),
          trailing: Icon(Icons.add),
          onTap: _addEventToCalendar,
        ),  // ListTile
      ],
    ),  // ListView
  );  // Scaffold
}
```

Figure 12: Example of Widget within the RehaBot App

There are two main categories of widgets in Flutter:

- **Stateless widgets**: These are widgets with properties that do not change over time such as an icon or a label. They do not hold mutable state and are a subclass of `StatelessWidget`.
- **Stateful widgets**: These are widgets whose properties change based on user interaction or other factors. A simple example would be a counter which increases every time a user taps a button, the value of the counter would be the state. They widgets subclass `StatefulWidget`, separating the immutable widget from a mutable State object. When the state changes, `setState()` is called triggering a rebuild.

This separation of widget and state lets Flutter treat both categories in the same way, parent widgets can recreate children at any time without losing their internal state as the framework manages the reuse of state object where necessary.

As Flutter and its applications have grown, state management has become increasingly important. In Flutter we can consider two state categories:

- **Local state:** State that belongs to a single widget and does not need to be shared like the current text in form field. This is usually managed via `setState()` directly.
- **Shared state:** Here the state may need to be accessible by multiple widgets across the app like authentication or user preferences.

To address this shared state, Flutter introduced the `InheritedWidget`, allowing states to be shared among descendants. This can be done by calling the `.of(context)` method, which returns the nearest ancestor of the specified type. While it is powerful, its implementation can be very verbose in large applications.

This led to higher-level state management with packages like provider which provide a wrapper around `InheritedWidget`.

An example of state management can be seen in *Figure 13* when handling user authentication in RehaBot. Here persistent login information is retrieved from `SharedPreferences`. If the user chose to remain logged in, the stored authentication token is retrieved and injected into an `AuthState` object provided higher up the widget tree. This then makes the authentication state available anywhere in the app without passing manual data [25], [26].

```
Future<void> _checkLoginStatus() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  bool stayLoggedIn = prefs.getBool('stayLoggedIn') ?? false;
  String? token = prefs.getString('token');

  if (!mounted) return;

  if (stayLoggedIn && token != null && token.isNotEmpty) {
    Provider.of<AuthState>(context, listen: false).setToken(token);
    await _fetchUserInfo(token);
    Navigator.of(context).pushReplacementNamed('/main');
  } else {
    Navigator.of(context).pushReplacementNamed('/login');
  }
}
```

Figure 13: State inheritance in RehaBot .Rendering

Flutter's architecture lets it control the rendering pipeline independently of the underlying operating system. Instead of relying on platform UI components, it paints

every pixel using the Flutter Engine. This give unmatched flexibility ensuring visual consistency across all platforms.

As we can see in *Figure 14,* the rendering pipeline starts when the widget tree, the declarative description of the UI, is transformed into an element tree and then into a render tree. This render tree is subsequently passed to the engine which uses the graphics library to rasterise the content onto the screen.

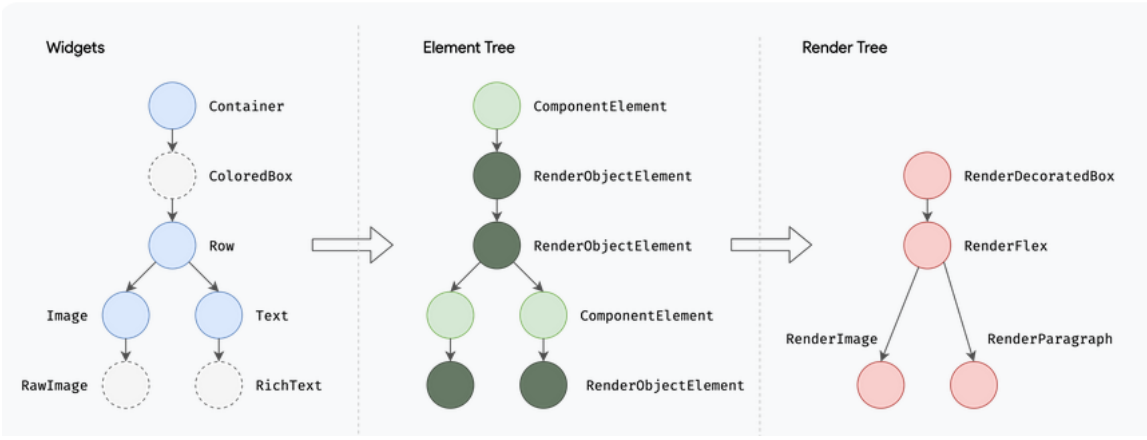


Figure 14: Flutter's rendering pipeline. Adapted from Flutter architectural overview [27].

This process has several stages:

1 Widget tree construction: The developer defines the interface using widgets to express the UI and behaviour in terms of the application state.

2 Element tree: Flutter creates an element tree that represents active instance of widgets. It manages the lifecycle and relationships between widget instances and their children

3 Render tree: The framework produces a render tree made of render object instances. They encapsulate layout, painting and compositing logic, translating into concrete instructions for the engine.

Finally, the rendering process continues with layout, painting, compositing and finally rasterization. Flutter's custom rendering pipeline has substantial performance benefits. As the framework handles its own layout, painting and compositing, it can optimise re-rendering by rebuilding only the parts affected by state changes. The build methods are designed to be side effect free and fast often at a speed of 60 frames per second or higher depending on hardware.

This control minimises overhead and avoids performance bottlenecks, as a result, applications achieve visual precision and interactivity regardless of the underlying system [25], [27].

3. Materials and Methods

3.1. Transition from Telegram to a Dedicated App

Initially, Telegram was selected as the platform for *RehaBot* due to its accessibility, ease of use and familiarity among users. This decision was also influenced by the availability of the chatbot integration. This integration allowed for the creation of a simple and effective communication platform between therapists and families. This initial version of *RehaBot* functioned as a lightweight, automated interface, enabling asynchronous communication, structured data collection and direct access to content, such as instructional videos. Since many families are accustomed to messaging apps for everyday communication this approach was meant to minimise barriers and avoid the need to learn or install new software.

User Feedback and Emerging Challenges

Following the initial deployment, questionnaires were distributed to gather anonymous user and therapist feedback (see Appendix x). As illustrated in *Figure 15*, users and therapists reported a range of perceived benefits, including:

- Creativity.
- Family support.
- Adaptability.
- Usefulness.
- Positive experience.
- Ease of use .

Additionally, 80% of final users and 50 % of therapists who answered reported that they would recommend using *RehaBot* in its current form. When asked about future versions, 50% of users and 78.57% of therapists showed interest in trying the next version. These results show that the idea was well-received overall while enforcing the need for improvement in both user satisfaction and therapist engagement.

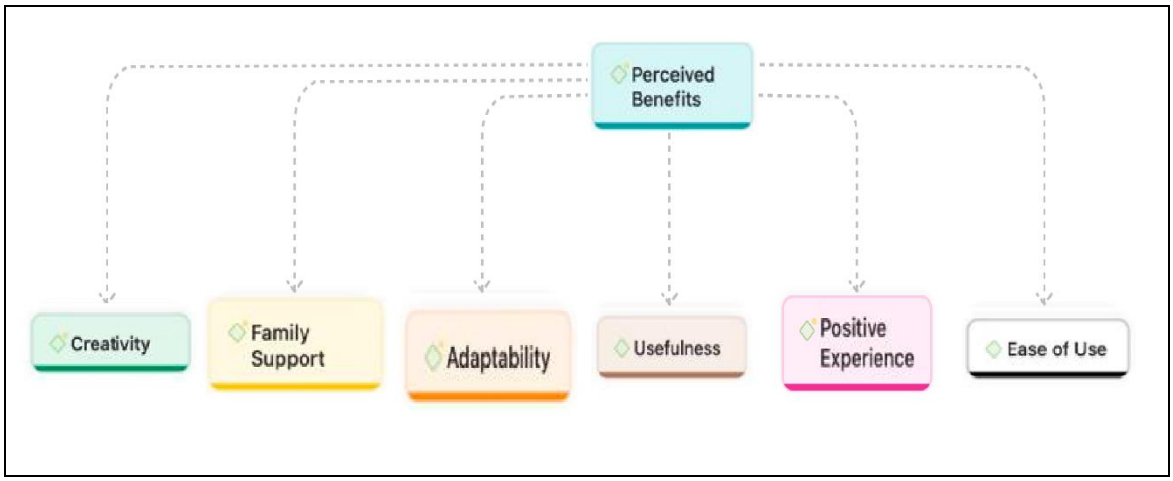


Figure 15: Perceived benefits of using RehaBot from users and therapists.

However, despite the many positive aspects, some difficulties which affected adherence and motivation rose as we can see in *Figure 16*. These included:

- Increased caregiver workload.
- Limited personalisation.
- Technical issues.
- Lack of consistent adherence.
- Insufficient family involvement.

While it may not be possible to fully resolve all these challenges in this next version of the application, the proposed improvements aim to mitigate their effects and enhance overall engagement.

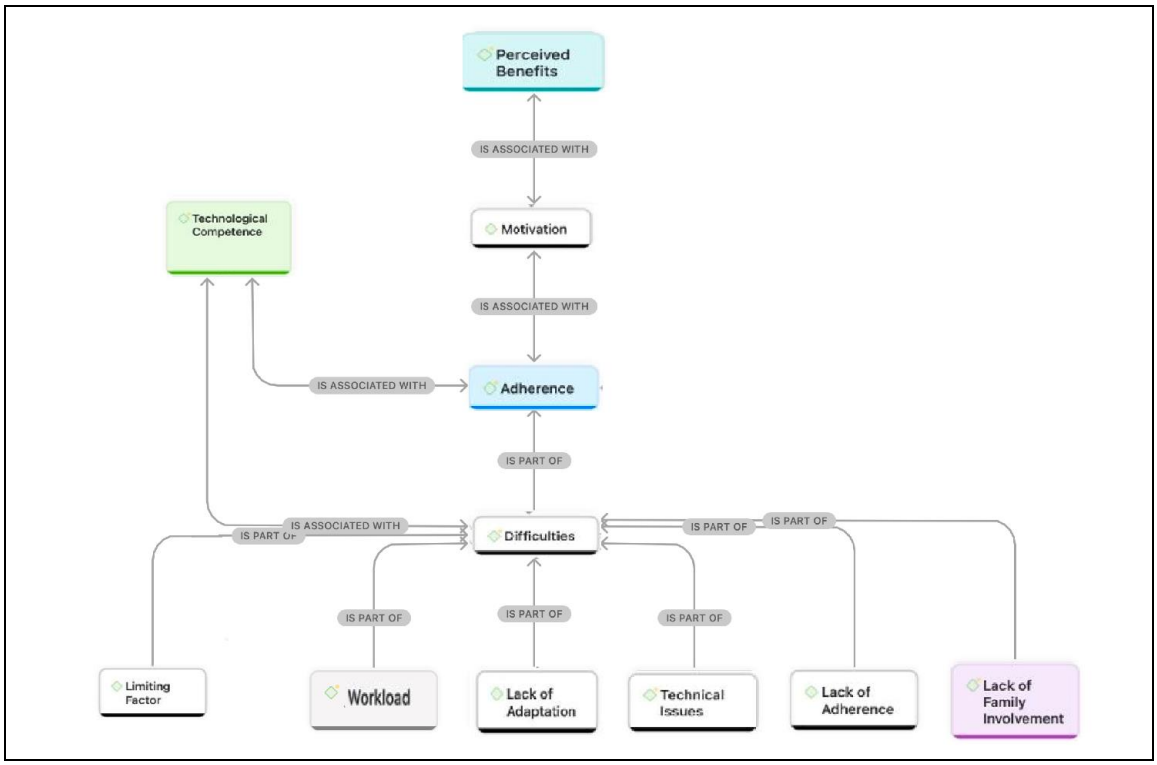


Figure 16: Perceived difficulties and their effects when using RehaBot.

Rethinking the Platform: User-Led Recommendations

Based on this feedback, several practical areas for improvement were proposed. Families asked for more positive reinforcement, a simpler registration process and the ability to upload videos of exercises.

Therapists recommended creating a dedicated app to allow them and families to monitor progress more easily.

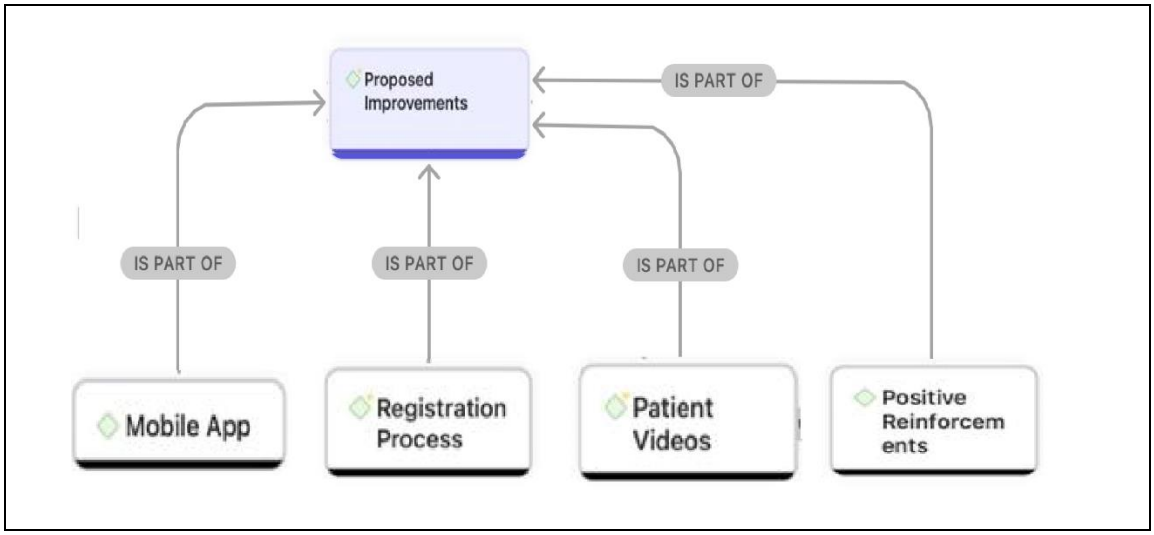


Figure 17: Proposed improvements to implementation of using RehaBot through Telegram.

Some of these key requests to be incorporated were:

1. Sending videos of users performing exercises at home.
2. Unlocking rewards through gamified progress tracking.
3. A timeline displaying user's videos and their feedback allowing for progress-monitoring.

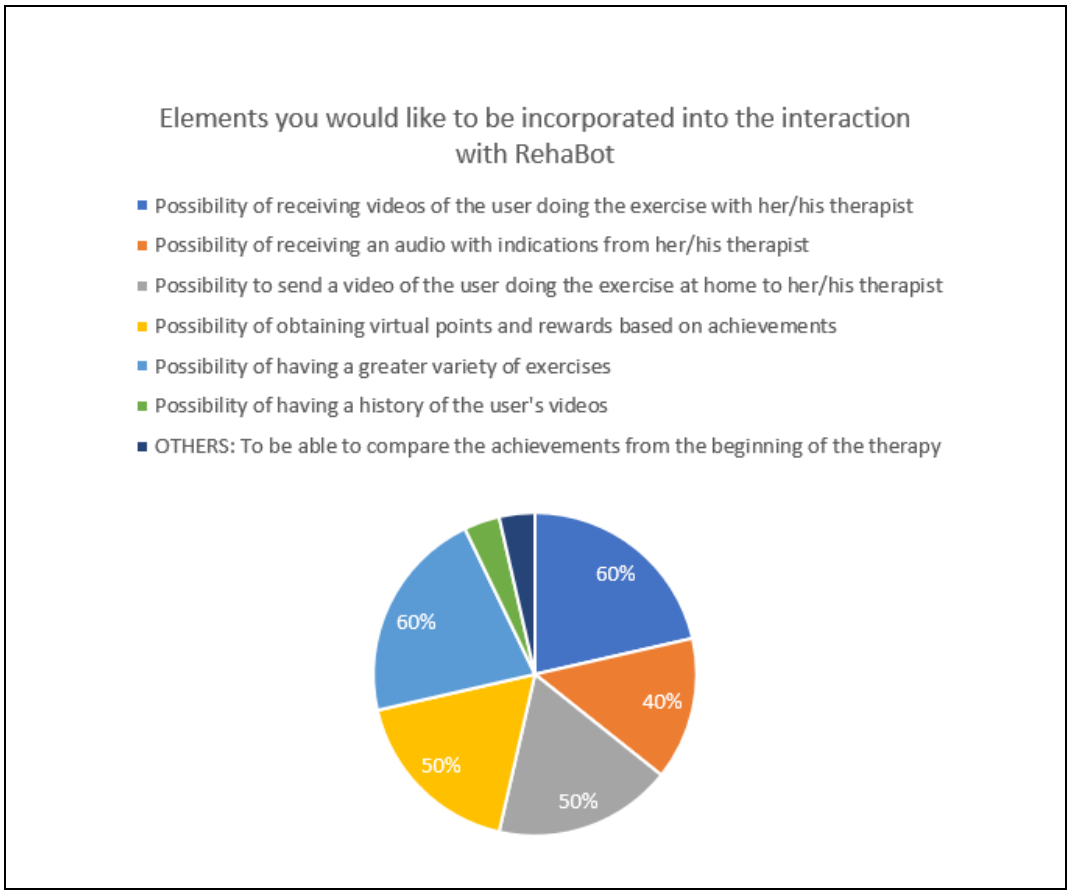


Figure 18: Elements participants would like to incorporate to RehaBot.

Some participants also expressed worries with using a general-purpose messaging app. Families concerns included the potential for children to access unrelated or harmful content through Telegram. Therapists also highlighted in feedback meetings that a dedicated app could make the program easier to use and more tailored to specific therapeutic needs:

Feedback questionnaire 03-04-2024:

"Como mejora, pensamos que creando una aplicación propia de RehaBot en vez de utilizarse por Telegram igual sería más fácil de manejar para las familias. Y con el fin de que a la vez que se hagan los ejercicios existiera la posibilidad de grabar un vídeo mientras se realiza el ejercicio y se enviase a la aplicación y poder ver así si se realiza de manera correcta dicho ejercicio."

Feedback questionnaire 17-04-2024:

"Pensamos que algunos de los aspectos a mejorar son la forma de darse de alta en el programa sin necesidad de que intervenga la familia o los usuarios, que RehaBot sea una aplicación directa, que se pudiera ver la manera de cómo el usuario lleva a cabo el ejercicio, la aplicación tenga refuerzos positivos, motivaciones para fomentar la adherencia al uso del programa."

A Custom Mobile App

Based on this feedback, the project transitioned to a dedicated cross-platform app. This new version preserves the key functionalities of the Telegram-based prototype

such as instructional video delivery, session tracking, and user feedback collection, while introducing enhanced features including:

- Transit encryption.
- User authentication.
- Customisable, age-appropriate interfaces.
- In-app parental monitoring tools.
- Reward systems and gamification elements.
- Timeline-based video comparisons.

This shift represents a natural progression moving from early proof of concept to a production ready solution which better fits the need of families and therapists. It also aligns RehaBot with clinical, ethical, and security standards in digital health [28]. By giving families, a secure, tailored environment, this dedicated app forms greater trust, engagement and long-term participation. It also paves the way for future expansion such as therapist dashboards, AI-generated progress insights, further gamification and personalised therapy pathways.

3.2. Prototypes and Chatbot Choice

In the early design phase, different approaches to the implementation of a chatbot were evaluated in order to balance ease of development, scalability, user experience and security requirements. Several options were considered, ranging from “homemade” solutions to more advanced integrations using Large Language Models (LLMs).

3.2.1. “Homemade” Chatbot

The first approach was to develop a custom chatbot protype coded from scratch in Flutter. This was implemented as a simple mobile app where the bot could greet the user, present a main menu and allow navigation into sessions and sub options as we can see in *Figure 19*.

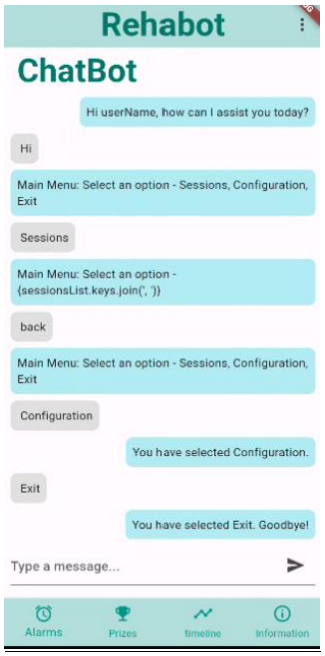


Figure 19: Protype of handmade chatbot.

The code defined a `ChatScreen` with a conversation history, input field and state management for:

- Greeting the user dynamically.
- Displaying a main menu with options like *Sessions, Configuration* and *Exit*.
- Navigation to return to the main menu.
- A basic interface with alternate user and bot message bubbles.

To assess the effectiveness of this protype the solution has been evaluated across a range of categories. Each category considers a specific aspect of the future system. The aim was to easily highlight the strengths and weaknesses in order to provide a balanced view of its potential and pitfalls.

Category	Evaluation	Description
Implementation	Easy	Simple prototype with menus and session flow. As it is built in Flutter it enables rapid development of a functional chatbot.
Scalability	Medium	Protype with hard-coded logic but Flutter's modular widget structure allows for future refactoring and expansion.
Design	Medium	Basic UI with alternating chat bubbles implemented with Flutter layouts. Functional and structured although visually basic.
Functionality	Medium	Supports greeting, main menu navigation, sessions and state management however responses are rigid and rule based.

Integration	High	Easily integrated with other Flutter app features such as APIs, databases or device-native functionality.
Personalisation	Medium	Currently limited but widget customisation would allow visual and functional personalisation.
Security & Privacy	Medium	No built-in security in the prototype but authentication, encryption and secure API communication could be added.
Analytics & Reporting	Low	None – no tracking or logging of interactions implemented. Would require additional development.
Support & Maintenance	High	The simplicity of Flutter and modular design ensures maintainability and smoother transitions in the future.
Cost & Licensing	Low	Development costs are minimal, mainly developer time. Flutter and Dart are both open source so there are no associated licencing costs.
User Experience	Medium	Provides a simple, functional chat flow with clear menus. It does lack natural, flexible conversation. UX may be rigid, but it is intuitive.

Table 2: Evaluation of homemade chatbot.

A we can see a homemade chatbot serves as a minimal but functional prototype, successfully demonstrating the feasibility of chatbot interaction flows with a mobile application. It excels in integration ease, thanks to Flutter's ecosystem, maintainability and its low cost while offering a clear structure that can be expanded upon. Although, it is limited in functionality, user experience, analytics and scalability at this early stage, its simplicity and modular Flutter architecture makes for an excellent foundation for further development.

3.2.2. ikChatBot (Flutter Plugin)

The second option taken into account was based on integrating the ikChatBot package into the Flutter project. This solution leveraged a pre-built and configurable widget that enabled quick addition of a functional chatbot to the app with minimal development effort.

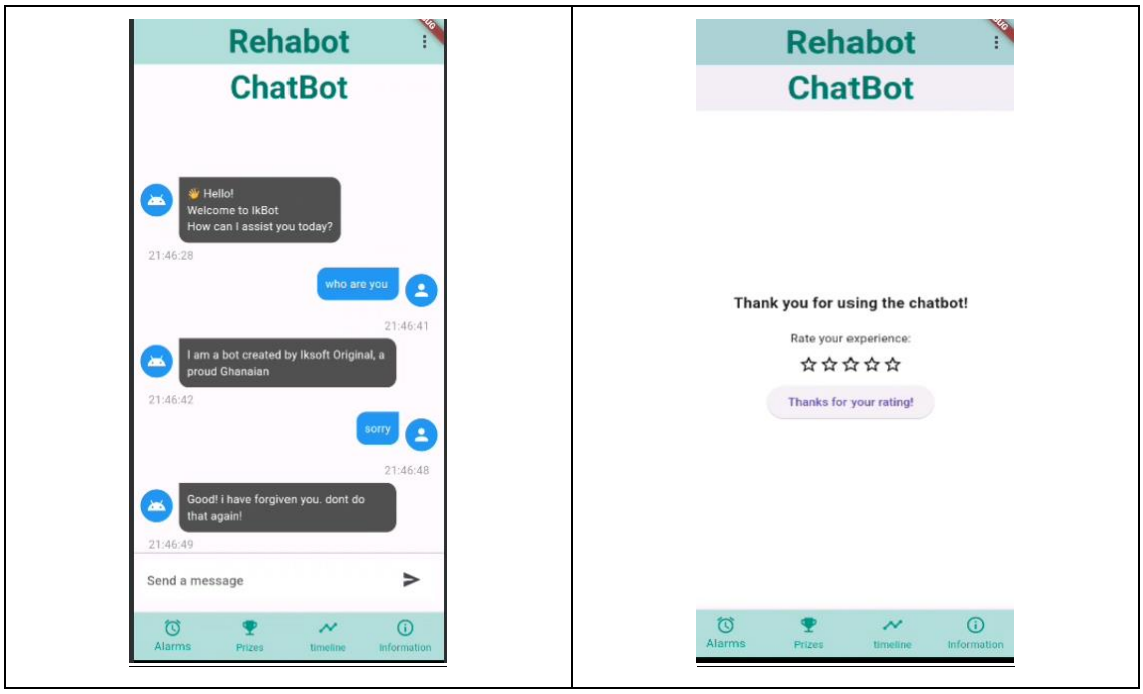


Figure 20: IkChatBot prototype

This plugin offers:

- A ready-to-use IkChatBot widget for Flutter.
- Customisation options for colours, icons, backgrounds and initial greetings.
- Background image support.
- Support for user interaction rating with feedback sent via email using SMTP.
- Basic configuration controls for inactivity, default responses and closing messages.

The source code is open and available on GitHub, allowing further extension if needed (ikChatBot, 2025).

Category	Evaluation	Description
Implementation	Easy	Integration through pubsec.yaml and embedding IkChatBot widget with configuration. Minimal learning curve.
Scalability	Medium	Handles basic chat flows, however, for high user load or more dynamic responses the source code would need to be modified.
Design	Medium	Allows customisation of appearance (colours, icons, background) but the widget structure limits full flexibility.
Functionality	Medium	Provides chat with predefined keyword response sets and a rating system but lacks multi-option navigation.
Integration	Easy	Easily integrates into existing Flutter apps.

Personalisation	Low	Customisable in appearance and some conversation content but limited by the predefined keywords and responses.
Security & Privacy	Low	No built-in encryption or authentication. Use of SMTP for email feedback without robust security measures is risky.
Analytics & Reporting	Low	Only basic interaction are captured and sent via mail. No advanced analytics or dashboards.
Support & Maintenance	Medium	Being open source, support depends on community and maintainers. Last update 23 months ago (as of August 2025).
Cost & Licensing	Low	Free and open source. Licensed under the terms on pub.dev (Iksoft Technologies, 2023).
User Experience	Medium	Intuitive user interactions but restricted to fixed inputs.

Table 3: Evaluation of ikChatBot.

The ikChatBot plugin simplifies the deployment of a chatbot in a Flutter app enabling rapid integration with its readymade widget and customisation options. Its main strengths are ease of use and low development overhead. However, this solution is functionally limited, constrained to keyword based replies, lacks advanced security features with minimal analytics.

3.2.3. Kommunicate

This option used the Kommunicate SDK and cloud platform to add advanced chatbot capabilities to a Flutter application. Kommunicate simplifies the creation, deployment and management of chatbots and offers integration with popular AI models like ChatGPT and NLP (*Natural language processing)* engines.

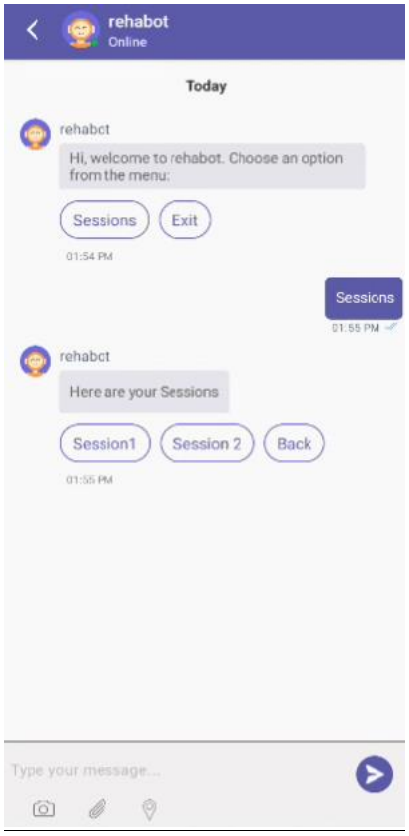


Figure 21: Kommunicate RehaBot prototype.

Kommunicate's AI agent and Bot provides [29]:

- No-code AI Agent Builder.
- Multi-source knowledge integration.
- AI recommendations and brand guardrails.
- Context-aware responses.
- Live agent routing and escalation.
- Customisable appearance.
- File and data capture.
- Language support.
- Analytics and reporting.

Category	Evaluation	Description
Implementation	Easy	Simple integration via SDK and detailed documentation, user-friendly setup through a dashboard and APIs.
Scalability	Medium	Scales well with higher user numbers and chat volume, however, higher usage increases costs significantly.
Design	Easy	Allows easy interface personalisation, with straightforward modifications.

Functionality	High	Advanced chatbot features available, including AI-powered responses, intent handling and integrations.
Integration	Easy	Integrates smoothly due to clear libraries and documentation, supporting third-party systems.
Personalisation	Medium	User experience and appearance are customisable; however, the depth is limited by platform constraints.
Security & Privacy	Medium	Basic security features such as permissions and role management among others. Robust compliance but not suitable for highly sensitive data without additional controls.
Analytics & Reporting	High	Real-time dashboard, recommendations, conversation summaries and expert support included.
Support & Maintenance	High	Dedicated technical support, onboarding assistance and regular optimisation.
Cost & Licensing	High	Requires use of advanced paid plans on a monthly basis.
User Experience	High	Smooth, intuitive and intelligent chat experience.

Table 4: Evaluation of Kommunicate's chatbot.

Kommunicate delivers a feature-rich and scalable chatbot solution that excels in rapid deployment, advanced AI orchestration, flexibility of integration, user experience and business support. However, it comes with high costs more adapted to a large business model.

3.2.4. LLMChat (Self-hosted LLM Chatbot in Flutter)

LLMChat is an open-source project available on GitHub combining a Flutter frontend with a Python FastAPI backend to deliver intelligent, customisable chatbot experiences using advanced LLMs.

Key features:

- Architecture: Flutter front end for UI and Python backend handling chat and LLM interaction via a FastAPI.
- Model Integration: Supports powerful LLMs like ChatGPT and local models, enabling rich conversations.
- Real-time communication: Uses WebSockets for bidirectional chat.
- Conversation history and summaries: Automatically tracks chat history and can generate summaries for analytics and review.
- Security: Built-in authentication, token validation and caching for privacy and scalability.
- Customisation: Open-source codebase allows full control.

Category	Evaluation	Description
Implementation	Medium	Requires backend setup and API integration, in depth documentation.
Scalability	High	Modern infrastructure and scalable models.
Design	High	Enables full customisation of UI/UX and backend workflows in Flutter.
Functionality	High	Supports advanced LLM capabilities, context and multiturn conversations.
Integration	Medium	More technical setup.
Personalisation	High	Open codebase and fully extensible.
Security & Privacy	Medium	Token validation and authentication.
Analytics & Reporting	Medium	Extensible for chat analytics, data storge and reporting
Support & Maintenance	Low	Relies on community and internal resources. Supported and updated by and individual.
Cost & Licensing	Low	Open source only infrastructure costs.
User Experience	High	Natural, robust conversation flow, UX can be optimised as needed.

Table 5: Evaluation of LLMChat.

LLMChat could be an ideal option for a team with technical expertise seeking maximum flexibility, privacy and sophistication however, it demands far more resources and maintenance than a simpler solution.

3.2.5. LangChain.dart (LLM Platform SDK for Dart Flutter)

LangChain is an open-source SDK offering a unified interface for interaction with multiple LLM providers in Dart and Flutter projects allowing developers to create powerful apps with NLP integration [30].

Key Capabilities:

- Multi-provider Integration (OpenAI, Google, local LLMs).
- Real-time data handling with WebSockets
- Data storage and retrieval of chat history for analysis and review.
- Authentication, authorisation and caching for robust and scalable deployments.
- Full customisation via its open-source codebase.

Category	Evaluation	Description
Implementation	Medium	Requires SDK integration, backend modifications and model provider configuration.
Scalability	High	Scales efficiently across providers, backend and data volumes.
Design	High	Enables full customisation of UI/UX and backend workflows in Flutter.

Functionality	High	Supports advanced NLP tasks, multi-provider AI and data management.
Integration	Medium	Flexible API interface but technical integration required.
Personalisation	High	Open codebase and fully extensible.
Security & Privacy	Medium	Token validation and authentication.
Analytics & Reporting	Medium	Extensible for chat analytics, data storage and reporting
Support & Maintenance	Low	Relies on community and internal resources. Supported and updated by an individual.
Cost & Licensing	Low	Open source only infrastructure costs.
User Experience	High	Powerful, customisable and natural user interactions.

Table 6: Evaluation of LangChain.dart.

LangChain.dart offers a complete and highly personalised solution to work with a Flutter project using an LLM model. Its modular architecture and advanced functionality make it an attractive option but too complex for rapid development or small-scale deployment.

3.2.6. Comparison and Final Choice

After evaluating and discussing the different options of chatbot implementation it was decided that the homemade Flutter chatbot was the best solution to integrate with the existing backend. This choice was based on multiple factors:

- **Minimal Cost:** By using open-source technology such as Flutter and Dart for the front end and using the existing backend infrastructure with minimal changes, the project avoids added licensing, hosting and third-party usage fees. This is essential for projects with strict budget constraints.
- **Rapid Prototyping and Development:** This version allowed for full control over the application and code enabling quick iterations, testing and deployment without waiting on external dependencies or schedules. Flutter's hot reload further accelerated UI development.
- **Seamless Integration:** Being native Flutter code allows direct and straight forward integration with existing backend APIs, databases, storage and device-specific functionality. Making reuse of resources simple and efficient.
- **Maintainability and Customisation:** The application's modular and clean architecture facilitates easy updates and targeted enhancements like improving conversational logic, adding analytics or hardening security measures like authentication and encryption.
- **Risk and Vendor Lock-in Control:** The solution does not rely on external services or commercial platforms therefore eliminating any concerns about

pricing changes, discontinuation or restrictive terms crucial for long term sustainability.

- **Scalability for Future Needs:** Although the initial implementation has basic hard-coded logic, Flutter0s flexible widget system allows refactoring and expansion to support more complex flows, AI integration or additional channels as resources allow.

In contrast, ikChatBot offered faster setup but lacked scalability and depth of customisation. Kommunicate deliver business-level capabilities but at a prohibitive cost. LLMChat and LangChain.dart provide unmatched flexibility and conversation sophistication however required significant infrastructure and technical expertise.

Therefore, the project continued based on the homemade prototype giving the best balance between cost, flexibility, maintenance and functionality for the project's current and foreseeable requirements.

3.3. Development

RehaBot was made with the aim of delivering an accessible and effective rehabilitation aid through a mobile application compatible with both Android and iOS platforms with the possibility of later expansion. This allowed for a single codebase reducing development and maintenance overhead in comparison to a native platform-specific development. The use of Flutter aligns with 2025 cross-platform development best practices, emphasising rapid iteration, consistent user experience and efficient resource management as well as its motto "*Write once, run everywhere"*.

Project Objectives

The core objectives of RehaBot were:

- Deliver a dedicated mobile chatbot rehabilitation system for children with cerebral palsy.
- To support therapy at home through structured exercise sessions, instructional videos and some form of gamification.
- To enable families to engage in therapy outside the clinic while giving therapists tools to track their progress.
- Establish a secure, scalable foundation allowing for future expansion.

The main deliverables of this project include:

- A functional mobile app with chatbot interaction.
- Integration of multimedia for exercises.

- Session tracking and reminders through notifications and calendar integration.
- Multi-language support.
- Feedback collection and gamification.

Resource Planning

This project was carried out under limited resources with a low financial budget and a small development team. The main resources included:

- Technological Stack: Flutter and Dart for cross-platform development integrated with a backend via secure HTTPS/JSON communication.
- Development Environment: Android Studio served as the primary IDE, providing a robust tool for building, testing and debugging the application.
- Libraries and Packages: Open-source Flutter dependencies listed in pubsec.yaml (e.g. provider, video_player, flutter_local_notifications etc.) were used to add functionality without additional licensing costs.
- Human Resources: A small team composed of two main roles:
    - Frontend: Flutter app development, UI design and integration of chatbot flow.
    - Backend: API management, session handling and server-side support.
- Budget: Minimal, dependant on open-source technology.
- Timeframe: Development iteration to be short and incremental without long delays or bottlenecks.

Now we will provide a comprehensive overview of the development process of RehaBot focusing on multiple aspects critical to delivering a cross-platform rehabilitation application tailored to individuals with CP. The development was guided by the principles of requirements engineering, modular design and incremental iteration. While the ultimate implementation was tested with Android, the use of Flutter and Dart has ensured compatibility with iOS.

3.3.1. Requirements Engineering

This project began by defining requirements in line with IEEE Standard 29148-2011, which defines a requirement as “A condition or capability need by a user to solve a problem or achieve and objective, or as something demanded of system”. This definition is particularly important within RehaBot because of the dual nature of participants: Rehabilitation specialist and patients with cerebral palsy.

Functional Requirements

These are the requirements which define what the system must do. They directly shaped the coding tasks and were implemented through different Dart files and Flutter widgets. The main FRs were:

- Chatbot Flow (FR-001): The system must provide a conversational interface where users can select sessions, be guided and submit feedback.
- Session Management (FR-002): The system must display rehabilitation sessions with associated multimedia.
- Authentication (FR-003): The system must allow users to log in securely.
- Feedback (FR-004): The system must record and send feedback.
- Gamification (FR-005): The system must provide motivational rewards.
- Calendar Integration (FR-006): The system must allow users to schedule therapy sessions.
- Overview (FR-008): The system must allow users to track their progress.

These FRs and others were implemented in the project tree and supported by different packages as we will see later on.

Non-Functional Requirements

These requirements define the constraints and qualities under which RehaBot must operate. They are influenced by design and architecture, key NFRs include:

- Portability (NFR-001): The app must compile and run on both Android and iOS.
- Performance (NFR-002): The app must remain responsive on low-cost Android devices and have a minimal delay when loading information.
- Security (NFR-003): All communication must occur over HTTPS.
- Usability (NFR-004): The app must provide a clear and accessible interface suitable for users with cognitive impairments.
- Internationalisation (NFR-005): The app must support multiple languages, initially English, French, German and Spanish.
- Maintainability (NFR-006): The app must be structured in a modular way.

These NFRs helped make the architectural and design decisions.

Information Requirements

During the design the API was provided to define exactly what data must be exchanged between the mobile app and the backend:

- User Authentication Data (IRQ-001): The system must authenticate users via email, password and language preference in JSON format.

```dart
final Map<String, dynamic> requestData = {
  'email': _emailController.text,
  'password': _passwordController.text,
  'lang': locale.languageCode,
};
```

Figure 22: User Authentication Data

- User Information (IRQ-002): The system will retrieve and store personal data from the API.

```dart
class UserInformation {
  final String firstName;
  final String lastName;
  final String doB;
  final String email;
  final String phoneNumber;
  final String language;
```

Figure 23: User Information.

- Sessions Data (IRQ-003): The system shall request active sessions and their metadata.

```dart
class Session {
  final int id;
  final String title;
  final int progress;
  final String initialDate;
  final String finalDate;
  final int daysPerWeek;
  final bool clarityOption;
  final bool difficultyOption;
  final bool painOption;
  final bool usefulOption;
  final bool repeatOption;
  final bool commentOption;
  final Therapist therapist;
  final List<Exercise> exercises;
```

Figure 24: Sessions Data.

- Exercise Data (IRQ-004): Each exercise will be included in its corresponding session.

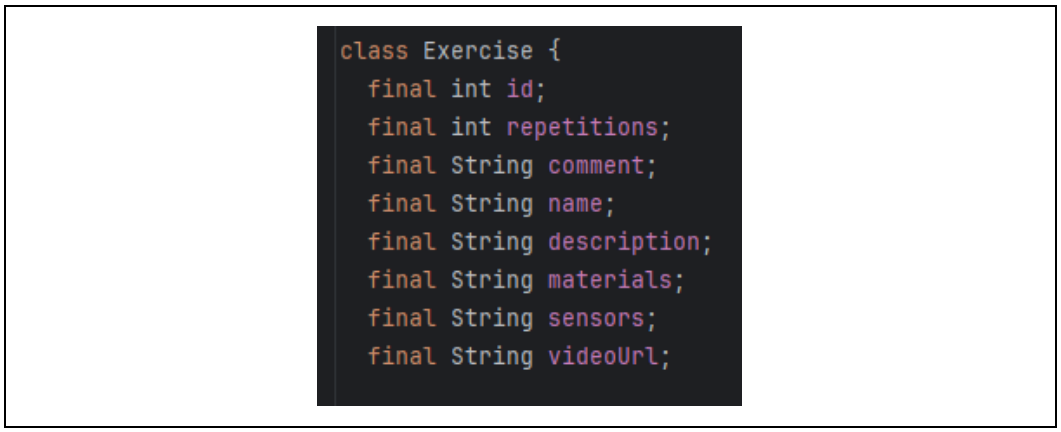


Figure 25: Exercise Data.

- Feedback Data (IRQ-005): After each exercise, when required by the therapist, the system shall store feedback values.

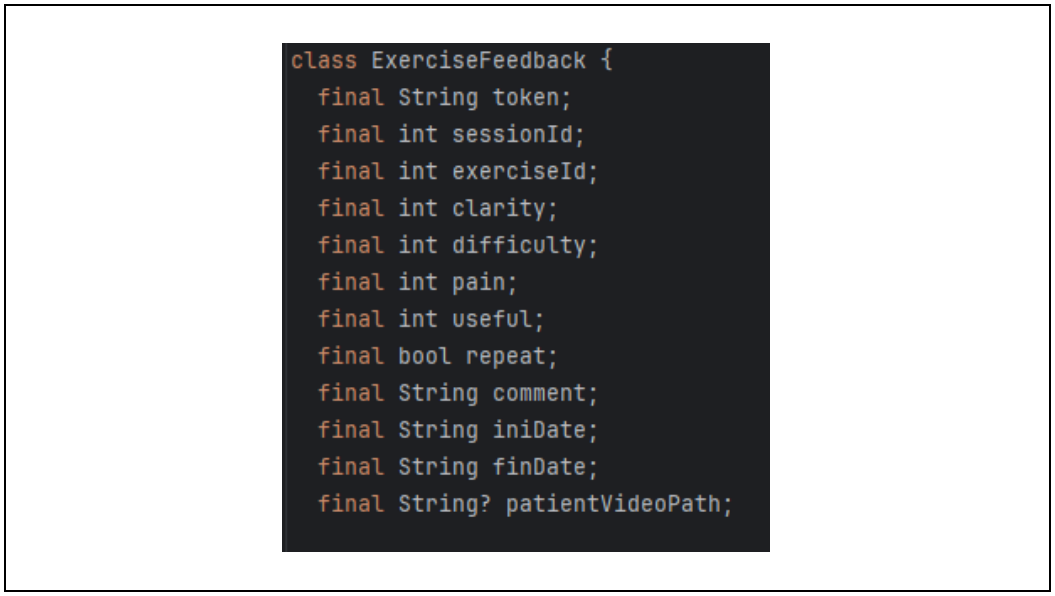


Figure 26: Feedback Data

3.3.2. Modular Project Structure

RehaBot has been organised into a modular project tree to ensure clarity and maintainability, each module is dedicated to a distinct responsibility in line with best practices and project requirements.

Location	Name	Requirement	Functionality
/assets	gifs/	FR-005	Motivational animations
	lang/	NFR-005	Localisation files (en/es/fr/de)
/lib	auth_gate.dart	NFR-002/FR-003	Together these files manage entry points and authentication ensuring all sessions are secure and state changes are isolated. This simplifies future updates to authentication logic or integration with third party providers.
	auth_state.dart		

	calendar.dart	FR-006	Handles the integration with session scheduling.
	chatbot.dart	FR-001	Core chatbot logic and UI. By isolating the chatbot, the conversation features, language models or UI can be extended without affecting unrelated modules.
	fetch_session.dart	FR-002/NFR-003	Centralises backend calls with HTTPS/JSON, offering a single source of information making it easier to test and maintain when changes occur.
	helper.dart		Utility functions.
	login.dart	FR-003	Login screen.
	main.dart	NFR-004	App entry point.
	model.dart		Defines shared data models (User, session, feedback). This prevents redundancy and ensures consistency throughout the project.
	prizes.dart	FR-005	Reward logic and UI.
	scroll_timeline.dart	FR-008	Timeline component.
	setting.dart	NFR-005	User preferences and localisation.

Table 7: Project file with requirement and description of functionality.

3.3.3. Use of Dependencies

In a Flutter project dependencies are defined in the `pubsec.yaml` file, the dependencies were chosen to support specific requirements.

Dependency	Requirement	Justification
flutter	Core	Fundamental SDK for all functionalities.
add_2_calendar	FR-006	Integration with device calendar to schedule therapy sessions.
easy_count_timer	FR-005	Simple countdowns/asynchronous gamification timers.
flutter_localizations, intl	NFR-005	Full internationalisation/localisation, supports multiple languages.
video_player, get_thumbnail_video, video_uploader, camera	FR-004, FR-008	Media capture, playback for engagement and reporting.
http	FR-001, FR-002, NFR-003	API calls for backend user/session data integration.
local_auth	FR-003	Enables biometric authentication for security compliance.
path_provider	FR-001	Handles device storage paths for caching media/data securely.

permission_handler	NFR-002	Manages runtime permissions for camera, notifications, etc..
provider	NFR-002	Robust state management throughout app (recommended for complex UX).
settings_ui	NFR-005, UX	User customization, preferences, and settings interface.
shared_preferences	NFR-002	Local storage for user preferences and persistent settings.
string_similarity	NFR-004	Intelligent chatbot input matching and fuzzy logic in responses.
timelines	FR-008	Visual representation of session progress and history.
url_launcher	FR-001	Playing videos.

Table 8: Project dependencies, requirement mapping and justification.

3.3.4. Iterative Flow

The development process followed and Agile, iterative flow ensuring flexibility and incremental delivery. Each cycle aimed to validate early and progressively refine the system. The main stages included:

1- **Requirement definition**: Functional and non-functional requirements were revised and clarified. This step involved collaboration with the rest of the RehaBot team.
2- **Proof of concept:** Prototypes were developed to explore the feasibility of proposed features or workflows.
3- **Integration:** Once a feature was deemed usable, it was integrated into the application. Doing this continuously helped maintain stability.
4- **Testing and feedback:** Each iteration included testing by manual evaluation.
5- **Refinement:** After testing and integrating different features, they were adjusted to improve usability, performance or maintainability. Here some requirements were redefined or modified feeding into the next iteration.

3.4. Features

The dedicated version of RehaBot has been designed to maintain the essential functions of the initial Telegram Bot while expanding its functionality to address the

needs and recommendations expressed by both users and therapists. The resulting application includes a set of features that combine user engagement, therapeutic guidance and clinical oversight while improving digital health security standards.

Core Functionalities

In its essence RehaBot continues to provide the same core functionalities that made the initial version so effective. These include:

- **Chatbot-based Interaction:** Designed to maintain a familiar conversation style like the Telegram Bot. The chatbot guides the user through exercises, reminders and integrates child-friendly GIFs to make the experience more engaging and motivating for younger users.

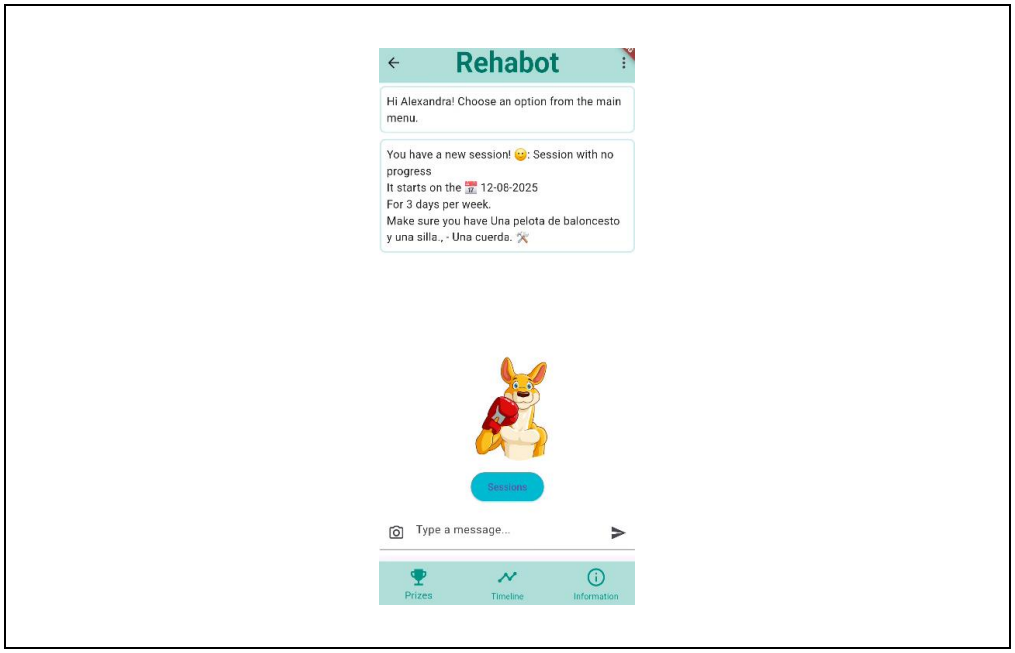


Figure 27: Example of Chatbot interaction.

- **Instructional Videos:** These allow the user to access therapist-directed demonstrations of exercises.

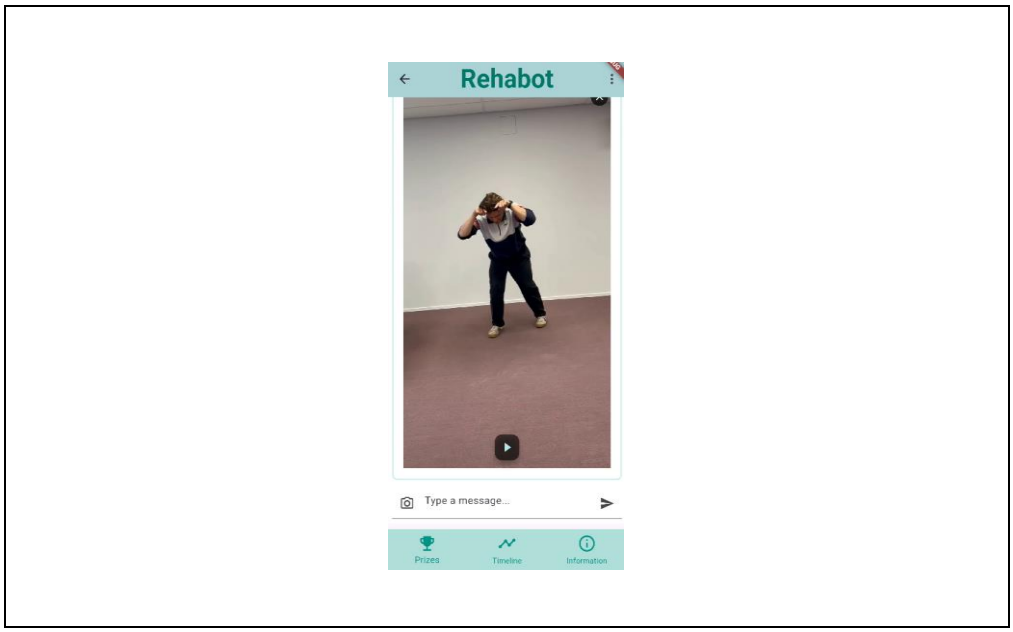


Figure 28: Example of instructional video.

- **Structured Sessions:** These allow families to follow therapy routines in an organised and consistent manner.

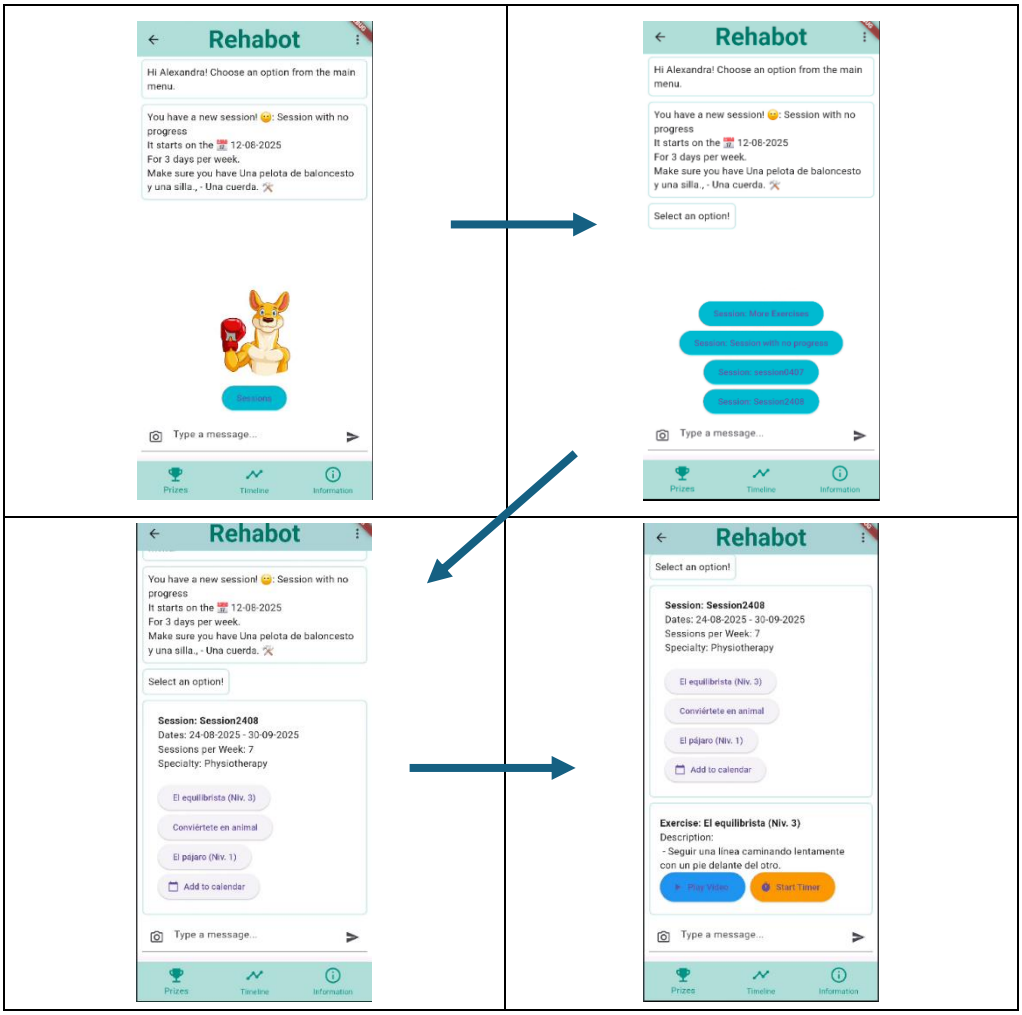


Figure 29: Example of a structured session.

- **Feedback Collection:** Users can report on different aspects of the exercises such as clarity, difficulty, pain experienced, usefulness, whether they want to repeat the activity and optional comments for the therapists.

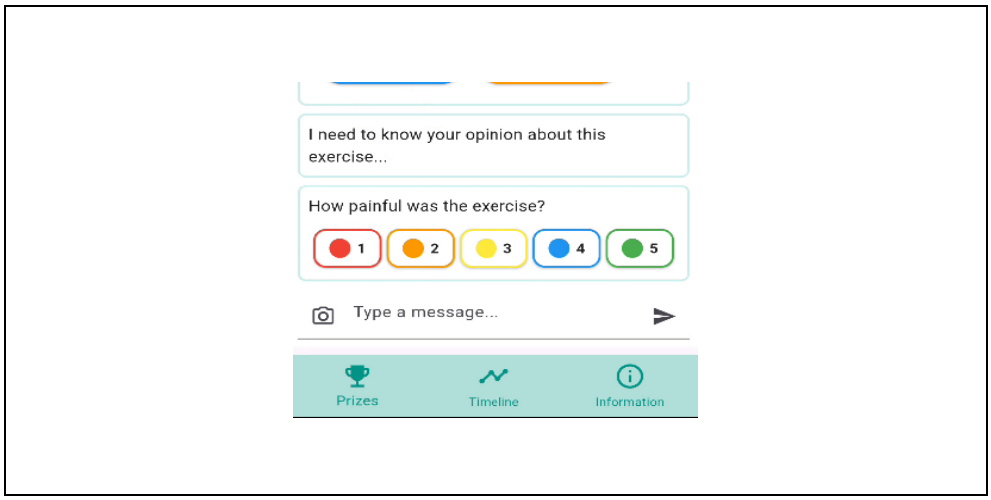


Figure 30: Example of a feedback question.

- **Multilingual Support:** The interface and chatbot are available in English, Spanish, German and French ensure accessibility for a wider population with the possibility to easily expand this range in the future.

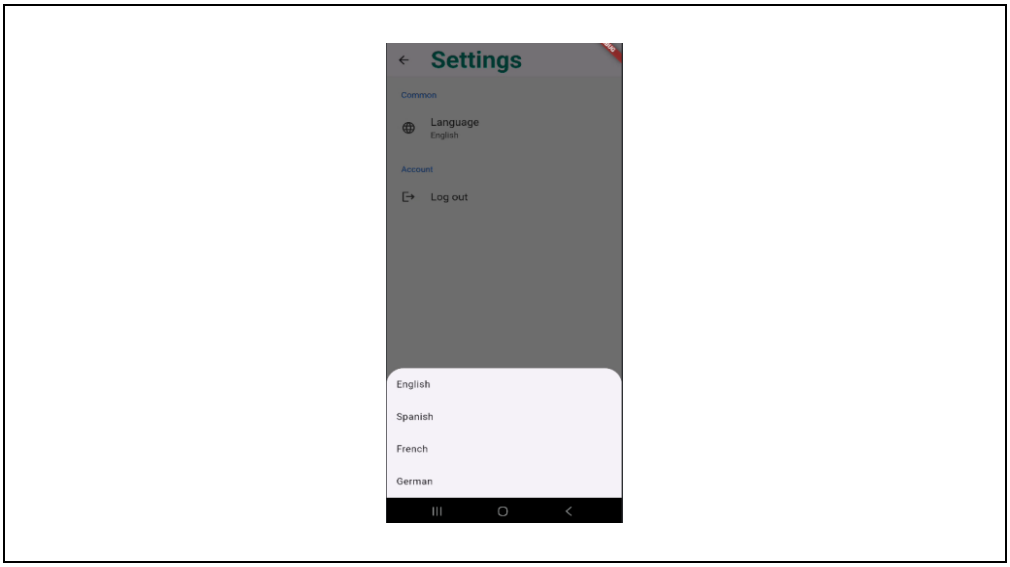


Figure 31: Example of language support.

New Functionalities

Building on the core functionalities, the dedicated application introduces several enhancements aimed at addressing improvements suggested by users and therapists to increase motivation, adherence and therapist-family collaboration:

- **Timeline View:** It displays a chronological record of videos and feedback enabling a more transparent view of progress over time.

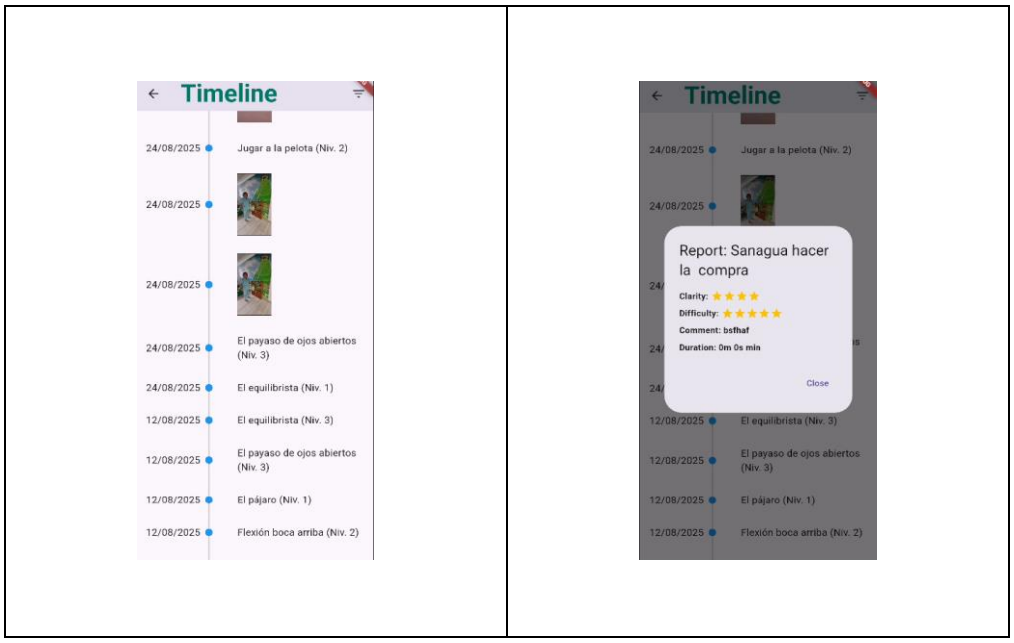


Figure 32: Example of timeline.

- **Video Recording and Uploading:** This allows families to capture the user's performance and share it with therapists for monitoring and feedback.

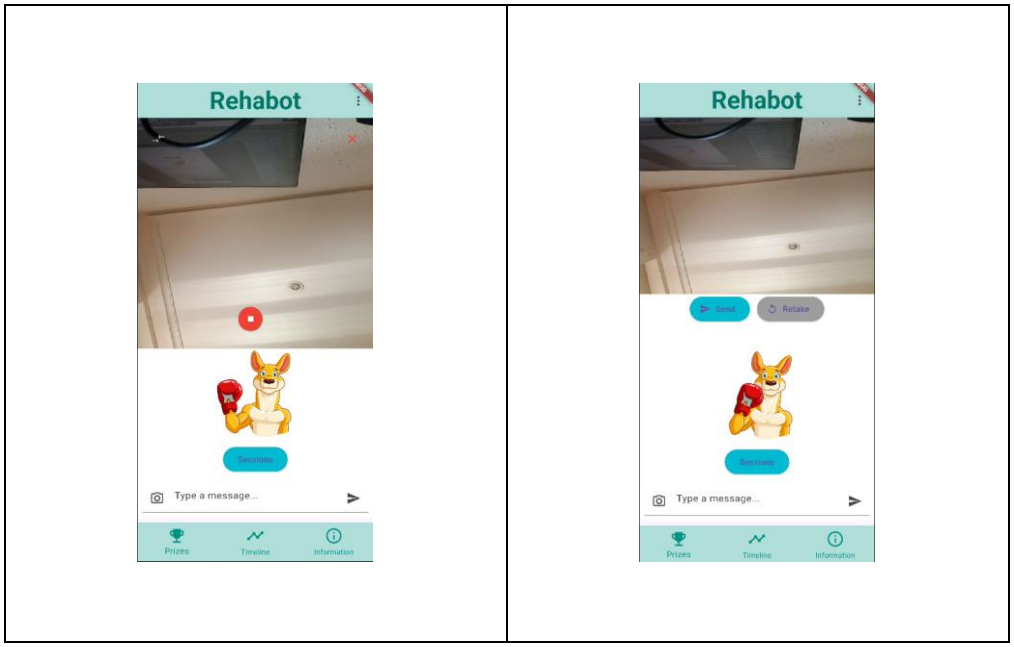


Figure 33: Example of video recording.

- **Prize View:** Rewards users through progress badges depending on how much time they have spent completing their exercises.

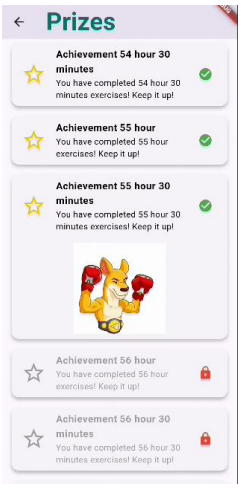
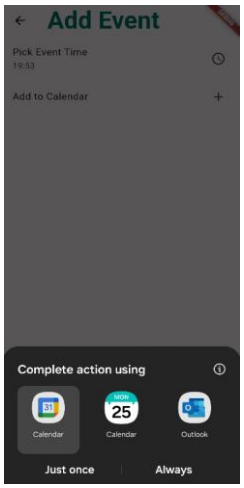
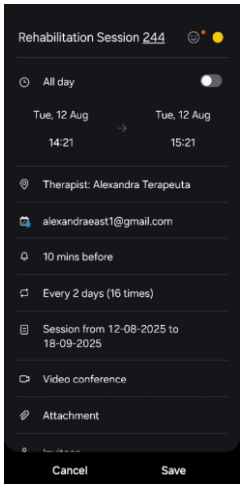


Figure 34: Example of prize view.

- **Calendar Integration:** Gives users the ability to add therapy sessions to their phone calendar, facilitating reminders and improving adherence to planned routines within daily family life.





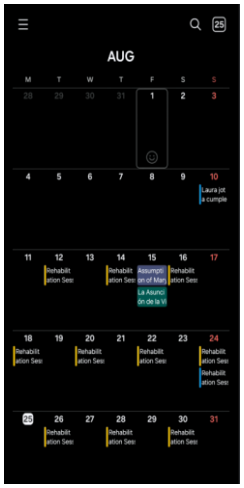


Figure 35: Example of adding a session to the phone calendar.

- **Safe, Dedicated Environment:** Providing families with greater control over how children interact with the app and safeguarding them from exposure to unrelated or unwanted contact as well as distractions.

3.5. User Experience and Interface Design

3.5.1. Introduction

The User Experience (UX) and User Interface (UI) design are central to the success of any digital heath solution, unlike a commercial app that may prioritise aesthetics or entertainment, a rehabilitation app must balance clinical effectiveness, usability and accessibility to have a meaningful impact. A poorly designed interface can create barriers for therapy adherence, directly affecting health outcomes. On the other hand, a well created UX/UI can transform rehabilitation into an engaging experience.

As we have seen, RehaBot's target audience are individuals with cerebral palsy, this means that the app must serve users who may rely on assistive technologies, require large text or high-contrast visuals. These users may also benefit from a straightforward interaction flow. Designing for this diversity means adopting a user-centred and inclusive design making sure that the application is not only functional but also accessible and motivating.

Another important design point is focus on motivation and engagement as adherence is a well-documented challenge in telerehabilitation contexts. A user is more like to maintain their sessions when the interface is intuitive, the content is simple and it gives the users a sense of progress and reward.

In summary, the UX/UI design was guided by three main ideas:

- Inclusivity
- Motivation
- Clinical Utility

3.5.2. Accessibility

Many of RehaBot's users may have varying degrees of motor impairment, visual difficulties or cognitive challenges. Therefore, this app was designed to go beyond minimum accessibility requirements, aligning with standards like Web Content Accessibility Guidelines [31] and platform-specific accessibility frameworks from Apple [32] and Google [33].

Large Fonts and Flexible Layouts

RehaBot, as we can see in *Figure 36*, respects system-level font sizes available in Android and iOS. Flutter's text widgets dynamically adapt to these preferences, ensuring that a user who may rely on large text can still navigate the app without losing functionality or visibility. To support this, RehaBot's layouts are constructed with flexible containers and responsive design principles.

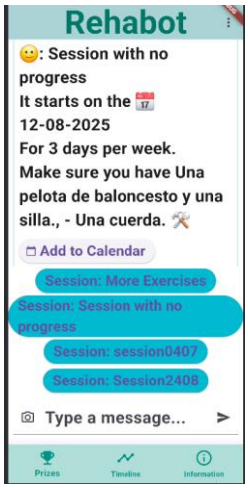


Figure 36: Adaptation of RehaBot's widgets with a large font size.

Screen Readers and Semantic Support

For those users who may have visual impairments, RehaBot integrates with TalkBack (Android) and VoiceOver (iOS), which allow users to receive spoken feedback about the contents of the screen and interact using gestures on mobile.

This is implemented through Semantic Roles, which define the purpose of a UI element. Flutter's standard widgets provide these semantics automatically, however, for a custom component, it is important to use explicit semantic annotations using Flutter's *Semantics* widget [34].

In *Figure 37*, we can see that in the `_inputField()` widget a semantic label , "*Email input field*", is assigned to a text field. This ensures that screen readers announce the purpose of the field to the user, without this, it would be presented as a generic input element.

```dart
Widget _inputField() {
  return Column(
    crossAxisAlignment: CrossAxisAlignment.stretch,
    children: [
      Semantics(
        label: "Email input field",
        textField: true,
        child: TextField(
          controller: _emailController,
          decoration: InputDecoration(
            hintText: "username".tr(context),
            border: OutlineInputBorder(
              borderRadius: BorderRadius.circular(18),
              borderSide: BorderSide.none,
            ),  // OutlineInputBorder
            fillColor: Colors.cyan.shade100.withAlpha(25),
            filled: true,
            prefixIcon: const Icon(Icons.person),
          ),  // InputDecoration
        ),  // TextField
      ),  // Semantics
```

Figure 37: Code snippet from RehaBot illustrating the Semantics widget.

Visual Accessibility

RehaBot follows WCAG guidelines for contrast ensuring that critical text and icons remain legible against their backgrounds. To make sure the UI had sufficient contrast ratios the WebAIM Contrast Checker tool [35] was used.

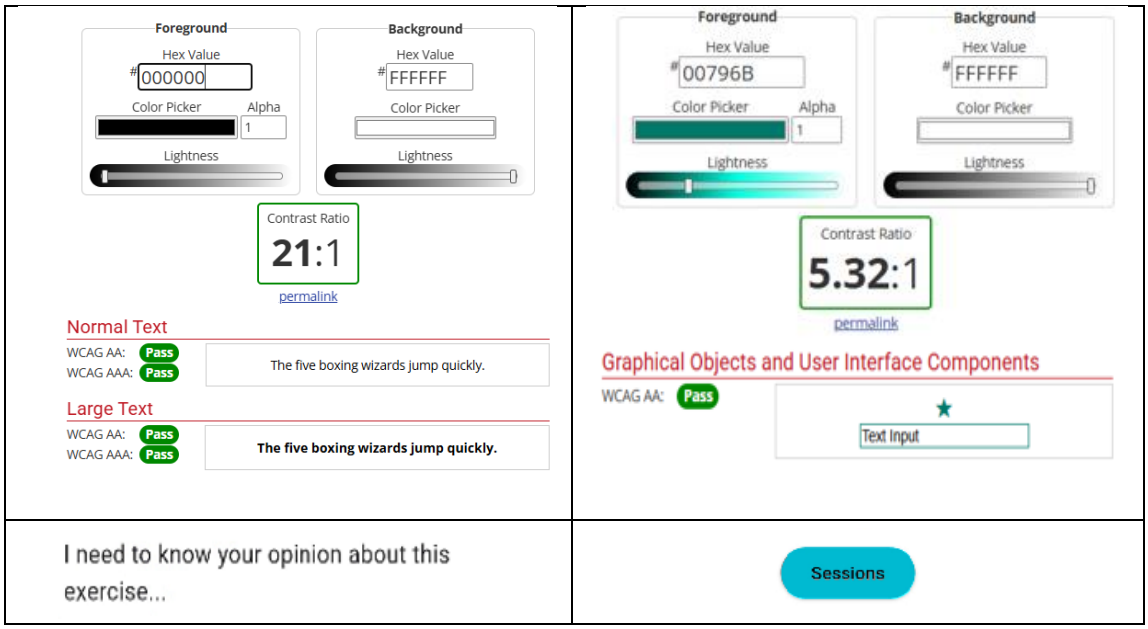


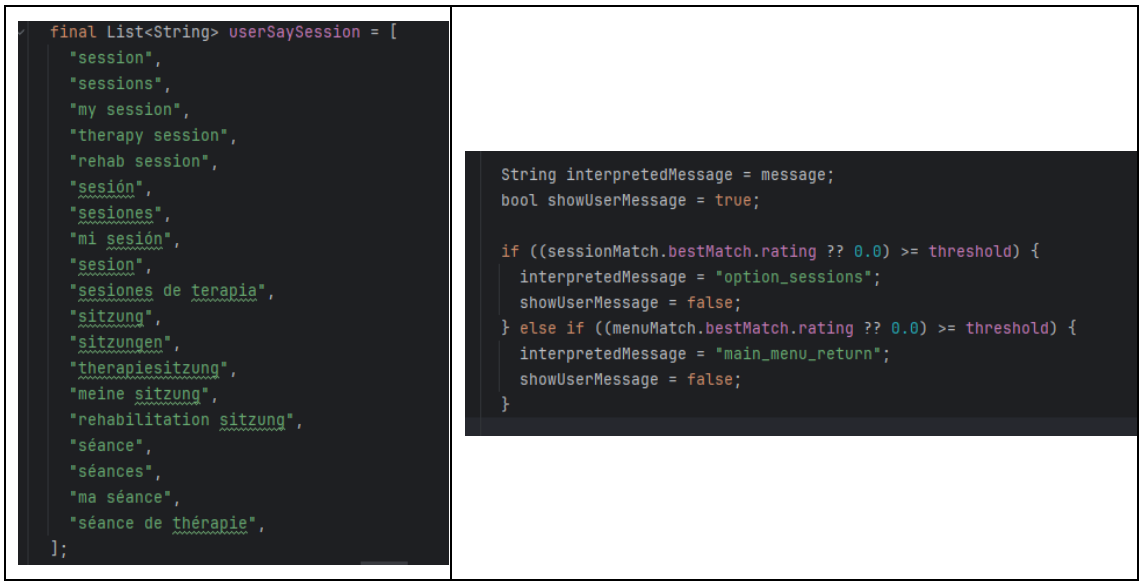
Figure 38: Examples of contrast rating obtained using the WebAIM Contrast Checker[35].

Interaction Accessibility

It is important to note that some users may experience cognitive difficulties or problems with attention and memory, the interaction flows in RehaBot have therefore been kept simple. There is clear labelling and consistent placement of important functions. Instructions are written in concise language and more complex processes, such as feedback submission were broken down into small, manageable steps.

Fuzzy text input matching has also been implemented to account for spelling errors, incomplete input or variation in phrasing. Fuzzy matching is a technique used to compare an input with a set of predefined options, the system calculates a similarity score, if the score is above the defined threshold, the input is interpreted as the closest valid command [36].

This is useful to those users with motor difficulties who may have trouble typing or those who might not remember precise commands . An example of this can be seen in *Figure 39*. where we can see the options for an input to be matched against predefined session commands. If the user types something similar, even with errors, it is interpreted correctly.



```
final List<String> userSaySession = [
  "session",
  "sessions",
  "my session",
  "therapy session",
  "rehab session",
  "sesión",
  "sesiones",
  "mi sesión",
  "sesion",
  "sesiones de terapia",
  "sitzung",
  "sitzungen",
  "therapiesitzung",
  "meine sitzung",
  "rehabilitation sitzung",
  "séance",
  "séances",
  "ma séance",
  "séance de thérapie",
];
```

```
String interpretedMessage = message;
bool showUserMessage = true;

if ((sessionMatch.bestMatch.rating ?? 0.0) >= threshold) {
  interpretedMessage = "option_sessions";
  showUserMessage = false;
} else if ((menuMatch.bestMatch.rating ?? 0.0) >= threshold) {
  interpretedMessage = "main_menu_return";
  showUserMessage = false;
}
```

Figure 39: Code snippet from RehaBot showing fuzzy matching.

Care was also taken to avoid sensory overload. Animations, transitions and message were created to be smooth. Chatbot responses or exercise confirmations were designed to give clarity without overwhelming users.

In addition, techniques to increase focus were introduced, when an activity is completed, the associated messages are automatically removed from the visible history. This avoids the screen from becoming cluttered with irrelevant information

reducing the risk of confusion directing the user's attention to the next actionable step.

To improve navigation, automatic scrolling to the bottom has been implemented as we can see in *Figure 40*. Thus, making sure that users have the most relevant content without having to manually scroll so they do not lose track of the conversation flow

.

```
void _scrollToBottom() {
  WidgetsBinding.instance.addPostFrameCallback((_) {
    _scrollController.animateTo(
      _scrollController.position.maxScrollExtent,
      duration: const Duration(milliseconds: 300),
      curve: Curves.easeOut,
    );
  });
}
```

Figure 40: Code snippet from RehaBot with method to scroll to the bottom of the screen.

3.5.3. UI Components

UI Components

1. **AppBar:** The app bar is used across all main screens to provide titles and navigation consistency. It uses the AppBarTheme, with a bold Roboto font at size 40 with teal shades to emphasise clarity.

```
appBarTheme: AppBarTheme(
  backgroundColor: const Color(0xFFB2DFDB),
  titleTextStyle: TextStyle(
    color: Colors.teal.shade700,
    fontSize: 40,
    fontWeight: FontWeight.bold,
  ),  // TextStyle
),  // AppBarTheme
```

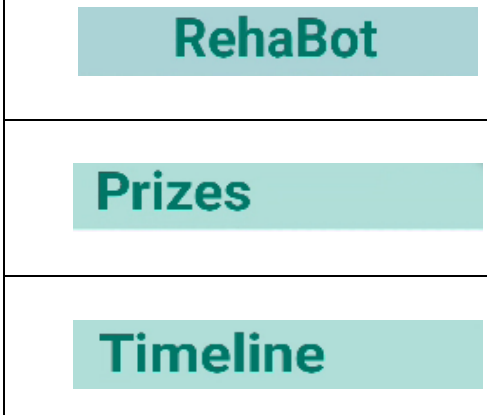


Figure 41: AppBar with code snippet showing its theme.

2. **BottomNavigationBar:** Navigation is implemented with the BottomNavigationBar which has three fixed tabs visible on the main chatbot screen. Labels are always visible to support usability and clarity.

```
bottomNavigationBarTheme: BottomNavigationBarThemeData(
  type: BottomNavigationBarType.fixed,
  backgroundColor: const Color(0xFFB2DFDB),
  selectedItemColor: Colors.teal.shade400,
  unselectedItemColor: Colors.teal.shade400,
  selectedLabelStyle: const TextStyle(
    fontFamily: 'Roboto',
    fontSize: 12,
    fontWeight: FontWeight.w500,
    color: Colors.black,
  ),  // TextStyle
  unselectedLabelStyle: const TextStyle(
    fontFamily: 'Roboto',
    fontSize: 12,
    fontWeight: FontWeight.w500,
    color: Colors.black,
  ),  // TextStyle
  showSelectedLabels: true,
  showUnselectedLabels: true,
  selectedIconTheme: const IconThemeData(size: 30),
  unselectedIconTheme: const IconThemeData(size: 30),
),  // BottomNavigationBarThemeData
```

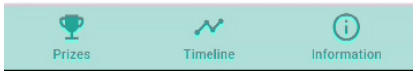


Figure 42: BottomNavigationBar and code snippet with its theme.

3. **Buttons:** Most buttons follow a standard design with ElevatedButtonThemeData as we can see in the code snippet. These buttons are used for core actions like choosing an option from the menu and logging in. For other functionalities different colours were used to allow for context-specific actions.

```
elevatedButtonTheme: ElevatedButtonThemeData(
  style: ElevatedButton.styleFrom(
    foregroundColor: Colors.black,
    backgroundColor: Colors.cyan.shade100,
  ),
),  // ElevatedButtonThemeData
```

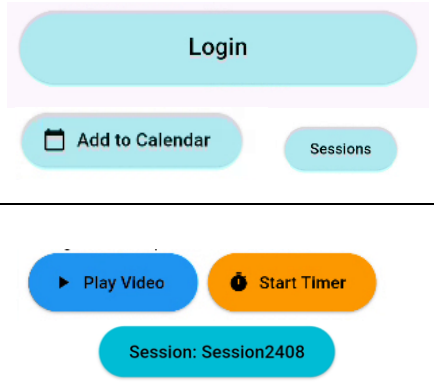


Figure 43: Standard button, special buttons and code snippet show standard button theme.

4. **Dialogs:** These are the chat bubbles for the conservational flow, they use styled containers to display messages exchanged between the user and the bot. Bot messages are aligned to the let with a teal border whereas user messages are to the right with a grey border.

```dart
Widget _buildTextMessage(Message msg) {
  return Padding(
    padding: const EdgeInsets.symmetric(vertical: 4.0),
    child: Align(
      alignment: msg.isBot ? Alignment.centerLeft : Alignment.centerRight,
      child: Container(
        decoration: BoxDecoration(
          color: Colors.white,
          border: Border.all(
            color:
                msg.isBot
                    ? const Color(0xFFCEEEEE) // Bot border colour
                    : const Color(0xFFDEDBDB), // User border colour
            width: 2,
          ),  // Border.all
          borderRadius: BorderRadius.circular(8.0),
        ),  // BoxDecoration
        padding: const EdgeInsets.all(8.0),
        margin: const EdgeInsets.symmetric(horizontal: 8.0),
        child: msg.content,
      ),  // Container
    ),  // Align
  );  // Padding
}
```

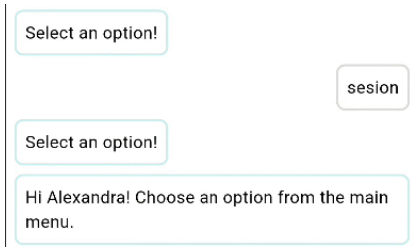


Figure 44: Chat messages and code snippet with their theme.

5. **Gifs:** Animated GIFs are used throughout the application to guide users in real time, help with transitions in the flow and improve engagement.

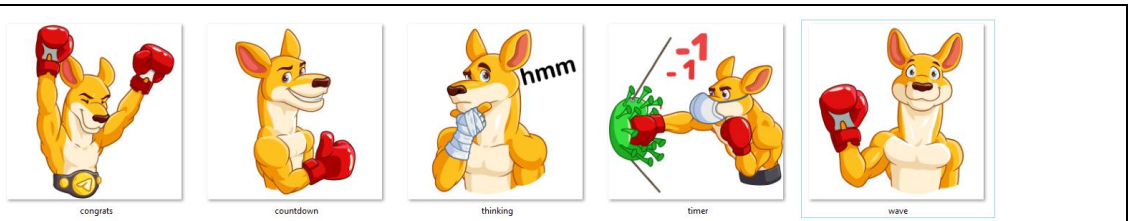


Figure 45: RehaBot's GIFs.

6. **Icons:** Icons are provided with Flutter's `IconThemeData`. The navigation icons are sized at 30px while the rest use 24px. The style used is minimalist in black and teal, consistent with the appearance of the application.

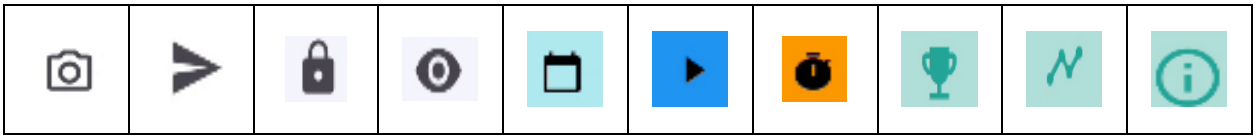


Figure 46: Example of some RehaBot icons.

3.5.4. Visual Style

Colour Palette

The user interface uses the colour palette in *Figure 47* with teal as the primary accent colour. These colours were chosen to create a neutral and calm environment reflecting the therapeutic nature of the application.

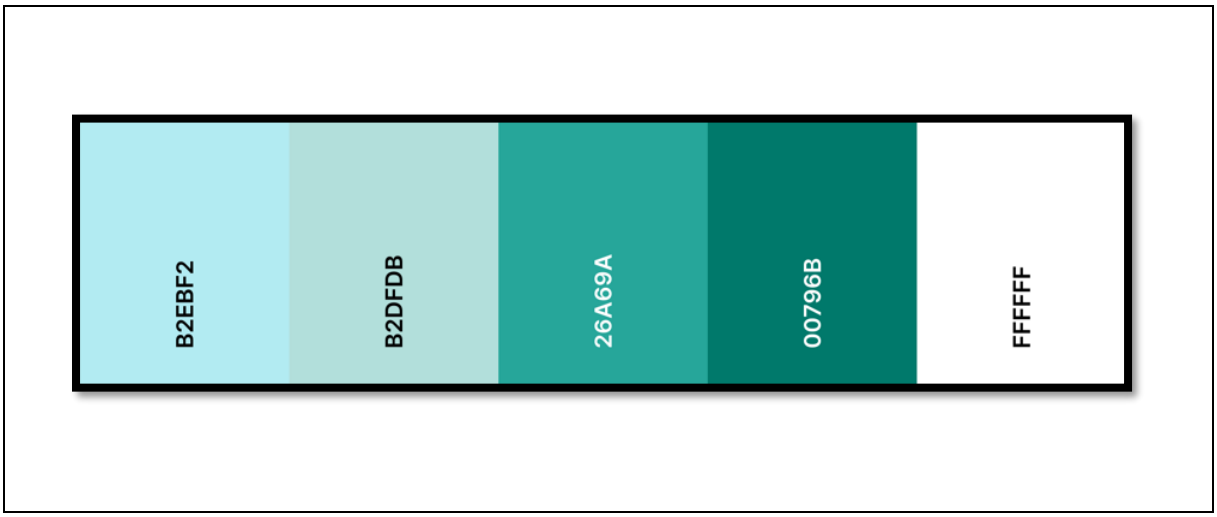


Figure 47: Colour palette applied to application.

Typography

RehaBot uses Roboto as its primary text style across all interface elements, a sans-serif typeface, for visual consistency and readability. This aligns with the WebAIM recommendation to use a simple and familiar font to help with faster and more intuitive reading by minimising character ambiguity [31].

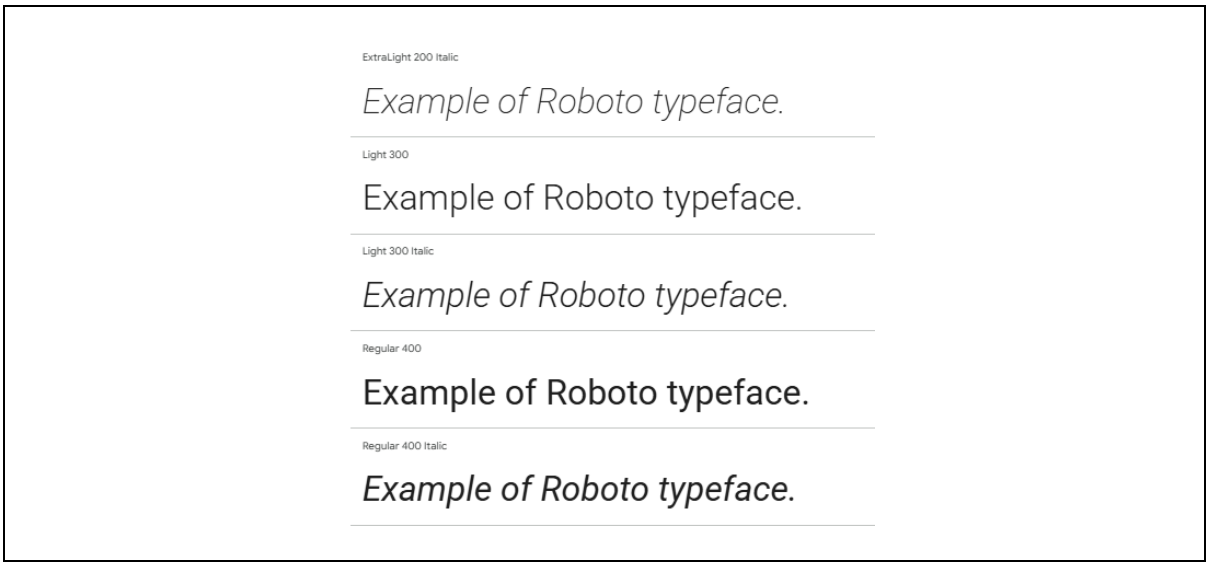


Figure 48: Examples of Roboto, typography chosen for RehaBot.

3.5.5. User Flows

A user flow is a visual or textual representation of the steps a user takes to complete a task within the application. By mapping user interactions, it helps to clarify the way the application should work and that we ensure navigation is intuitive. We will now describe the basic user flows withing the application: Authentication, navigation bar, sessions, feedback, prizes, timeline and settings.

Authentication Flow

This flow allows a user to access their personalised therapy content securely. As we can see in *Figure 49* the user opens the app, the login screen opens, they enter their credentials and they submit them. If they are successful, they will go to the home screen, otherwise they will receive an error message and remain on the login screen.

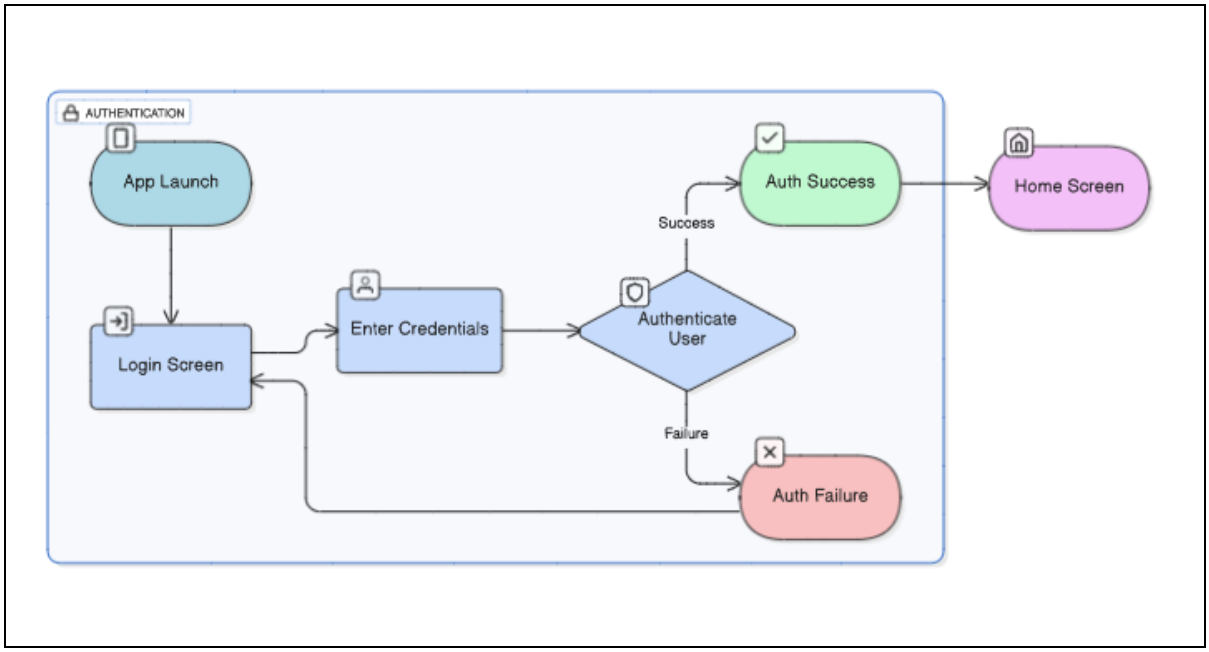


Figure 49: Authentication flow.

Navigation Bar Flow

The navigation bar provides quick access to the three main sections of the application, the chatbot interface is within the home screen. The user taps on a navigation icon and the app switches to the corresponding screen with prizes and the timeline; with legal information a popup is displayed informing the user of their rights.

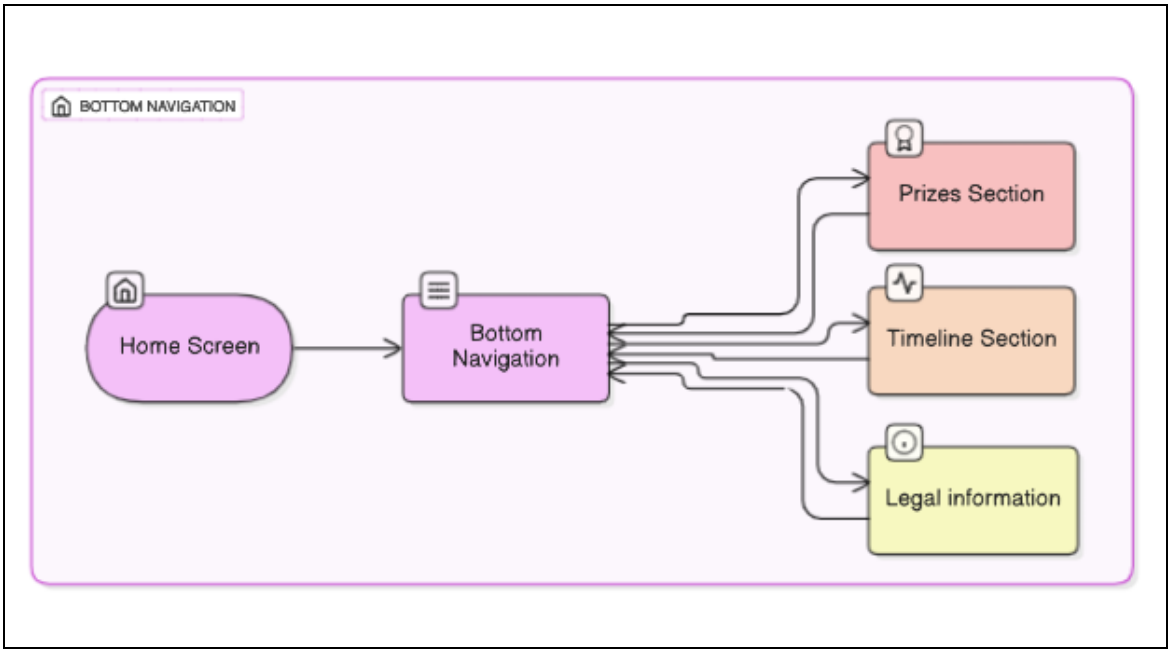


Figure 50: Navigation flow.

Chatbot Flow

The basic chatbot flow guides users through their sessions. The sessions are displayed and the user may tap on one of them. When the user taps on the session the corresponding exercises are displayed, here the user has the option to display the video of the exercise or to take a video of themselves. The user will then start the timer, from here they can cancel and return to the exercise list or tap finish and continue to the feedback flow.

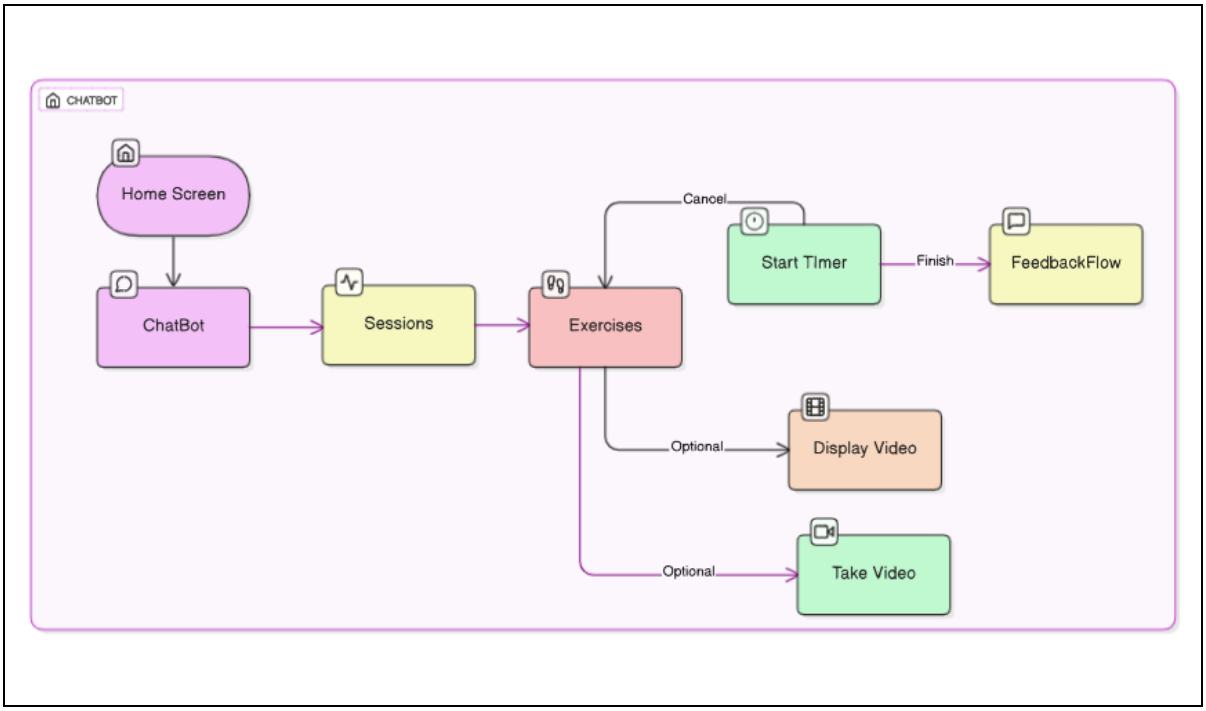


Figure 51: Chatbot flow.

Feedback Flow

The feedback flow captures users feedback if the therapist has requested it, after completing exercises. The user will answer questions about clarity, difficulty, pain, usefulness, repeating the exercise and any comments. If the user has taken a video of themselves, it is attached. The feedback is then submitted and the user returns to the beginning of the chatbot flow.

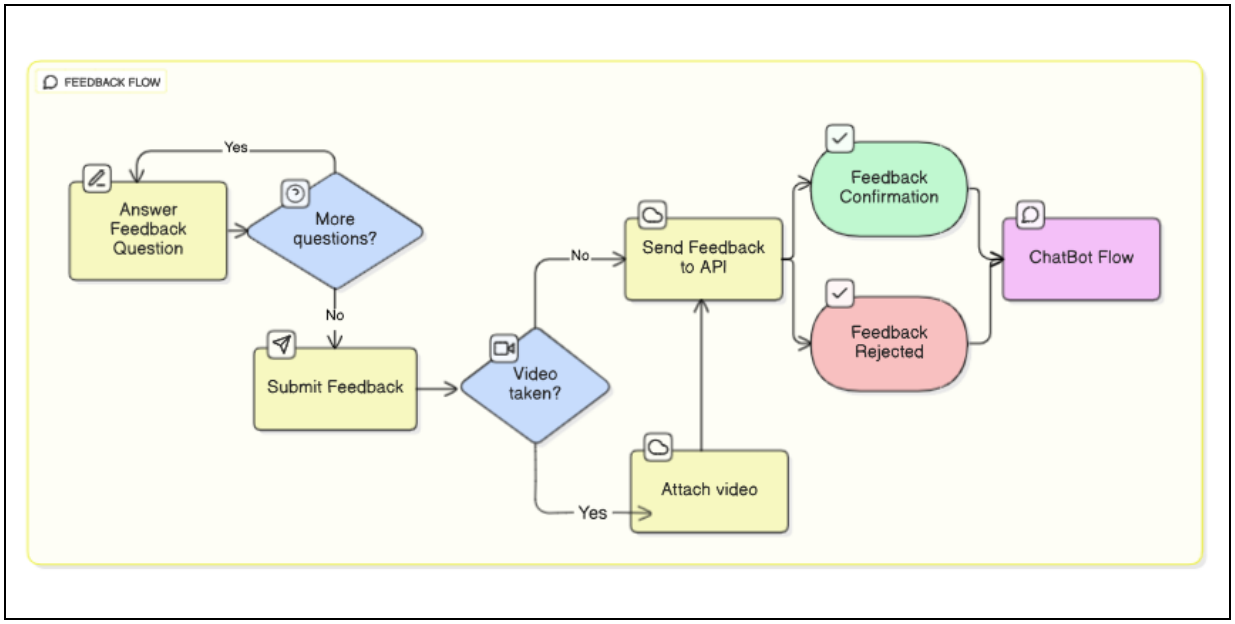


Figure 52: Feedback flow.

Timeline Flow

The timeline flow shows a chronological view of the exercises a user has completed. The user accesses the timeline screen through the navigation bar, timeline items are displayed and the user may scroll through the timeline.

Timeline items have a date on the left, a video or name on the right, and a grey or blue dot indicator. If the user applies filters such as if there is feedback, if there is a video or a date range, the timeline updates dynamically. If the user taps on the grey indicator, there is no feedback, so no action occurs. If the user taps on the blue dot indicator the feedback dialog corresponding to the exercise is displayed.

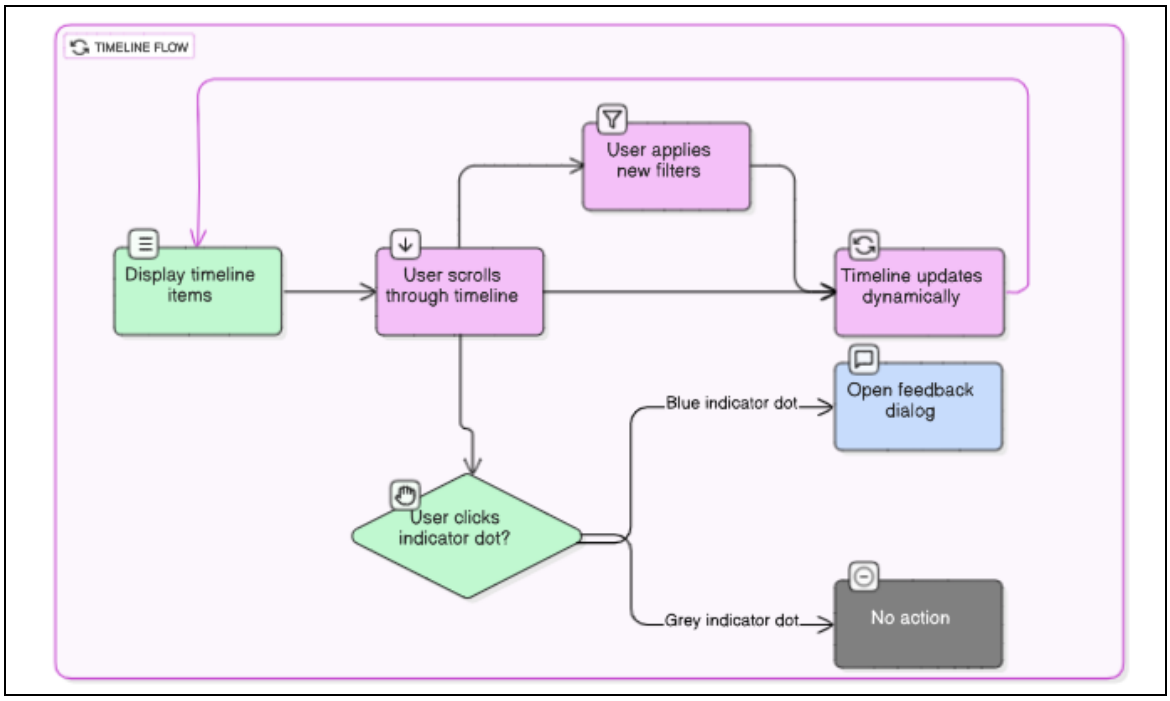


Figure 53: Timeline flow.

Rewards Flow

The rewards flow displays achievements earned by users depending on the total time they have exercised.

The user navigates to the prizes screen through the navigation bar and achievements are loaded. Depending on the time spent, achievements will appear as blocked or unblocked.

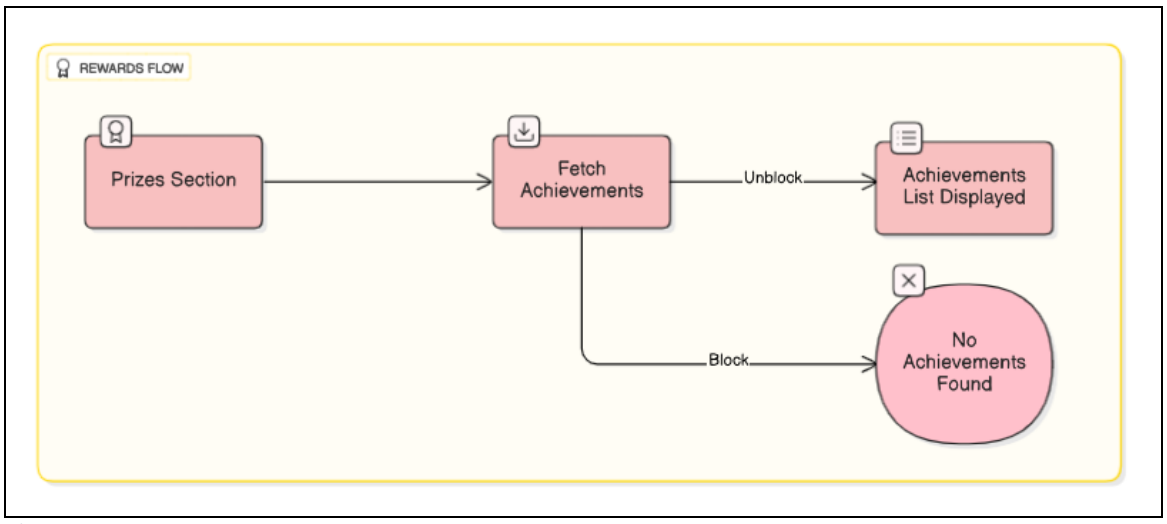


Figure 54: Rewards flow.

Settings Flow

The settings flow allows users to modify their language preference or to logout of the application. The user navigates to settings through the home screen, if the user

changes the language, the chatbot screen is updated; if the user taps log out, they are navigated to the login screen.

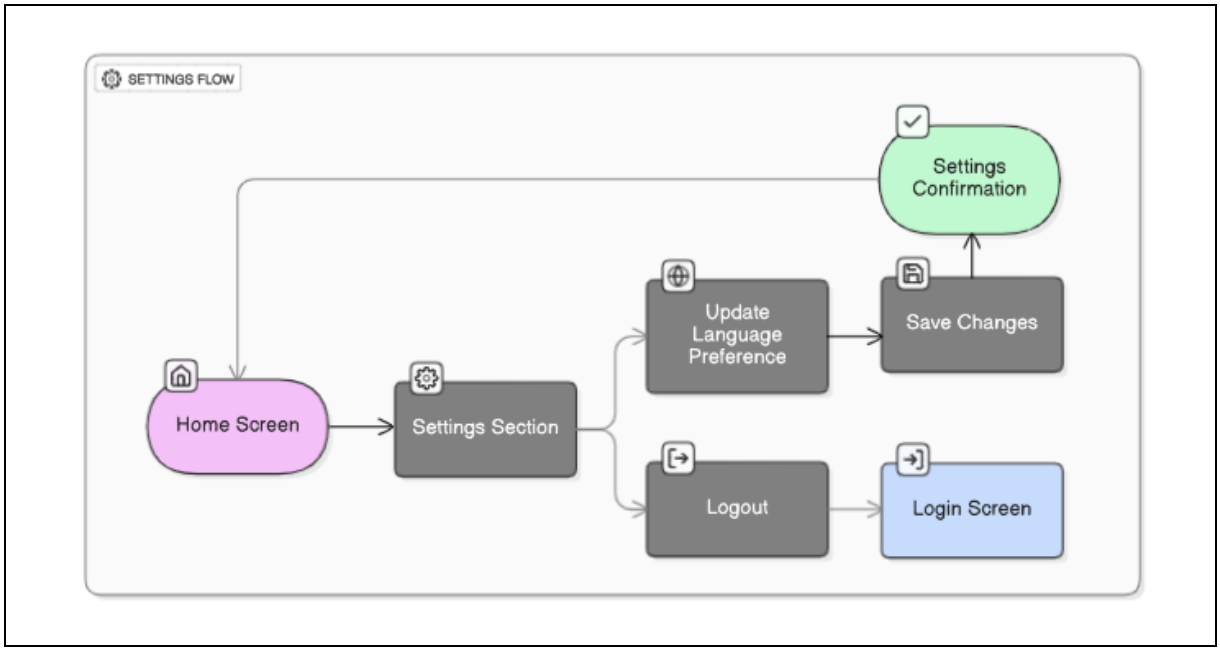


Figure 55: Settings flow.

4. Testing and Evaluation

The testing phase of RehaBot was mainly focused on ensuring that the application not only functioned as intended from a technical point of view, but also aligned with the other objectives of usability, accessibility and therapeutic suitability. While development and prototyping were focused on achieving a stable and scalable design, the testing allowed for closer inspection of how the application would behave under real-world conditions.

Testing Environment

Due to limited resources at the time of testing, it was carried out exclusively on a physical device. As we can see in *Table x*. the device uses was a Samsung Galaxy A14 5G running Android 15. This model is a mid-range Android phone comparable to devices likely to be used many of the families within the target audience.

Using a physical device testing was able to capture practical aspects of performance like touch responsiveness and system resource management. However, this also introduces limitation, the evaluation was unable to include other devices, screen sizes or operating systems narrowing the amount of insight gained at this stage.

At the same time, development relied on *hot reload* and integrated debugging tools within Android Studio, this allowed for rapid iterations and easy troubleshooting. These tools supported efficient refinement of individual components as they were added.

Environment	Version
Development IDE	Android Studio Build #AI-243.25659.59.2432.13423653
Runtime version	21.0.6+-13368085-b895.109 amd64
Physical Device	Samsung Galaxy A14 5G (Model: SM-A146P/DSN)
Device Software	One UI 7.0, Android version 15

Table 9: Testing environment used.

Methodology

Testing was incremental and modular, rather that waiting to integrate components at later stages, individual features were tested separately: chatbot interactions, feedback collection, video upload and playback. Flutter's reactive architecture supported this by keeping components and state management self-contained, this made it easier to detect and resolve issues at the widget or logical level before combining together in the overall application. This separation helps to prevent cascading errors and makes debugging far more effective.

Development and testing were helped by Android Studio's debugging tolls and hot reload. Testing through Android Studio enabled fast iteration, updates were deployed almost immediately, reviewed in real time and adjusted as needed.

The evaluation focused on four key point: Functionality, appearance, backend integration and accessibility:

- **Functionality:** Core features like the chatbot flow, multimedia loading and feedback submission were examined for stability and responsiveness.
- **Appearance:** UI elements were checked to confirm that they aligned with visual and branding requirements.
- **Backend Integration:** Secure and reliable communication with the API endpoints was tested to make sure that data was correctly received, transmitted and stored.
- **Accessibility:** Tests were carried out to ensure WCAG standards were followed with focus on readability, font scaling, colour contrast and ease of navigation.

Results

Testing confirmed that RehaBot was stable and reliable under controlled conditions. The app launched consistently, responded rapidly to user input and had smooth interactions with the backend.

In terms of usability, the interface was judged to be intuitive with some adjustments needed. During iterations, changes were introduced to some elements and flow like the feedback widget as well as introducing different spacing, padding and colour contrast. This was done to improve readability and accessibility, these adjustments although small, were important in making the application more functional and inclusive.

Limitations and Future Testing

The results from testing were encouraging, however, the scope of the testing extremely limited. This means that older Android versions, iOS devices or tablets with different display ratios have not been assessed. As a result, there may be compatibility issues which remain untested.

Another important point is that formal studies with families and therapists of this version of RehaBot remain essential for assessing the application's effectiveness in a real-world context with its intended users.

Therefore, future testing should include:

- **Multiple device validation:** Making sure that the application is compatible with a range of devices from budget Android models to a large screen tablet.
- **Cross-platform validation:** Ensuring that the application is tested in iOS devices as it is one of the objectives of this project.
- **User evaluation:** Engaging families and therapist in testing.

Conclusion

In general, the testing confirmed that RehaBot has a solid, technical foundation for its intended use. The absence of any major technical errors, together with the successful integration of backend services means that the project is in a good position for expansion.

At the same time, it is important to highlight the need for additional testing as mentioned above because the success on one model does not guarantee universal compatibility, nor the ways families may interact with their different systems at home.

5. Conclusion and Future Work

This end of degree project has presented RehaBot, a cross-platform mobile application created to support home-based rehabilitation for children with cerebral palsy under the guidance of a therapist. From its beginning, the projected was guided by the objective of creating a tool which would extend the reach of physical therapy beyond the traditional clinic, making it accessible, secure and engaging.

The creation of this application has shown that RehaBot is able to facilitate therapist-directed, personalised rehabilitation programs through video-based sessions and a structured chatbot interface which guides patients step by step. By using Flutter and Dart, the application ensures accessibility and usability across devices, giving users an intuitive, user-friendly environment. Features like gamification, scheduling and progress tracking have helped to promote engagement and adherence, helping families to integrate physical therapy into their routines.

Equally as important, is the application's ability to provide a secure and dedicated environment, where privacy and safety are prioritised while reducing distractions from general messaging platforms. RehaBot's modular architecture has created the foundation for scaling the application allowing for the integration of more advanced technologies. This project empowers caregivers by giving them tools to monitor progress and provide meaningful support while allowing therapists to deliver high quality and adapted care remotely.

In summary, Rehabot has demonstrated that a dedicated, thoughtfully designed system can lessen the gap between clinical rehabilitation and the home environment. It enhances continuity of care and improves collaboration between all involved contributing to more personalised, engaging and effective treatment experiences.

Future Work

While most of the initial objectives have been achieved, the process has shown many ways for further improvement and innovation of RehaBot. One of the most interesting options to follow is the advancement of gamification within the app. Expanding beyond the current reward system, which is quite basic, to include interactive challenges and games could greatly increase engagement and motivation in particular with younger audiences.

Another important point is the integration of artificial intelligence, particularly in the analysis of exercise videos. This gives the potential to provide real-time evaluation of performance, improvement over time and even deliver feedback directly to the patient. This would make it a stronger tool for remotely monitoring patients while giving therapists more insight into the user's progress.

Expanded testing is of utmost importance, although the system has shown functional reliability, it is necessary to confirm how effective it is in a real-world situation. In the future it would be of use to enhance the accessibility features such as speech recognition ensuring the platform is even more inclusive for children with diverse needs.

Finally, RehaBot's impact in the long term will depend on a commitment to being sustainable and innovating continuously. This means including new exercises, adaptive therapy plans and the integration of new rehabilitation technologies that may appear.

In conclusion, while RehaBot has achieved its objective of providing a digital, rehabilitation tool for children with cerebral palsy, its biggest potential is its capacity to grow and evolve.

References

[1] K. Vitrikas, H. Dalton, and D. Breish, 'Cerebral Palsy: An Overview', *Am. Fam. Physician*, vol. 101, no. 4, pp. 213–220, Feb. 2020.

[2] L. W. M. E. Beckers *et al.*, 'Feasibility and effectiveness of home-based therapy programmes for children with cerebral palsy: a systematic review', *BMJ Open*, vol. 10, no. 10, p. e035454, Oct. 2020, doi: 10.1136/bmjopen-2019-035454.

[3] 'RehaBOT'. Accessed: Aug. 31, 2025. [Online]. Available: https://rehabot.eu/

[4] S. McIntyre *et al.*, 'Global prevalence of cerebral palsy: A systematic analysis', *Dev. Med. Child Neurol.*, vol. 64, no. 12, p. 1495, Dec. 2022, doi: 10.1111/dmcn.15346.

[5] NHS, 'Cerebral palsy – Treatment - NHS'. Accessed: Aug. 30, 2025. [Online]. Available: https://www.nhs.uk/conditions/cerebral-palsy/treatment/

[6] S. P. Das and G. S. Ganesh, 'Evidence-based Approach to Physical Therapy in Cerebral Palsy', *Indian J. Orthop.*, vol. 53, no. 1, pp. 20–34, Feb. 2019, doi: 10.4103/ortho.IJOrtho_241_17.

[7] A. Gregory and A. G. H. editor, 'Children in UK suffering "irreversible harm" due to physiotherapy delays', *The Guardian*, Jan. 31, 2025. Accessed: Aug. 30, 2025. [Online]. Available: https://www.theguardian.com/society/2025/jan/31/children-in-uk-suffering-irreversible-harm-due-to-physiotherapy-delays

[8] C. Lillo-Navarro, F. Medina-Mirapeix, P. Escolar-Reina, J. Montilla-Herrador, F. Gomez-Arnaldos, and S. L. Oliveira-Sousa, 'Parents of children with physical disabilities perceive that characteristics of home exercise programs and physiotherapists' teaching styles influence adherence: a qualitative study', *J. Physiother.*, vol. 61, no. 2, pp. 81–86, Apr. 2015, doi: 10.1016/j.jphys.2015.02.014.

[9] K. S. Bıyık, C. Özal, M. Tunçdemir, S. Üneş, K. Delioğlu, and M. K. Günel, 'The functional health status of children with cerebral palsy during the COVID-19 pandemic stay-at-home period: a parental perspective', *Turk. J. Pediatr.*, vol. 63, no. 2, pp. 223–236, Apr. 2021, doi: 10.24953/turkjped.2021.02.006.

[10] T. Bright, S. Wallace, and H. Kuper, 'A Systematic Review of Access to Rehabilitation for People with Disabilities in Low- and Middle-Income Countries', *Int. J. Environ. Res. Public. Health*, vol. 15, no. 10, p. 2165, Oct. 2018, doi: 10.3390/ijerph15102165.

[11] L. Thomas and S. Bhat, 'A Comprehensive Overview of Telegram Services - A Case Study', May 2022, doi: 10.5281/ZENODO.6513296.

[12] 'WhatsApp: number of monthly active users 2025', Statista. Accessed: Aug. 30, 2025. [Online]. Available: https://www.statista.com/statistics/260819/number-of-monthly-active-whatsapp-users/

[13] 'How To Build Bots for Messenger', Meta for Developers. Accessed: Aug. 30, 2025. [Online]. Available: https://developers.facebook.com/blog/post/2016/04/12/bots-for-messenger/

[14] Statista, 'Facebook Messenger global audience 2025', Statista. Accessed: Aug. 30, 2025. [Online]. Available: https://www.statista.com/statistics/1498791/messenger-potential-audience/

[15] Segun-Falade *et al.*, 'Developing crossplatform software applications to enhance compatibility across devices and systems', pp. 2040–2061, Aug. 2024, doi: 10.51594/csitrj.v5i8.1491.

[16] 'Introduction to Dart'. Accessed: Aug. 30, 2025. [Online]. Available: https://dart.dev/language/

[17] R. Camden, *Apache Cordova in action*. Shelter Island, NY: Manning Publications, 2016.

[18] S. Bosnic, I. Papp, and S. Novak, 'The development of hybrid mobile applications with Apache Cordova', in *2016 24th Telecommunications Forum (TELFOR)*, Belgrade, Serbia: IEEE, Nov. 2016, pp. 1–4. doi: 10.1109/TELFOR.2016.7818919.

[19] TheKnowledgeAcademy, 'Unity Architecture: The Building Block of a Game Engine'. Accessed: Aug. 30, 2025. [Online]. Available: https://www.theknowledgeacademy.com/blog/unity-architecture/

[20] 'What is Unity?', PubNub. Accessed: Aug. 30, 2025. [Online]. Available: https://www.pubnub.com/guides/unity/

[21] 'Build Cross-Platform Mobile Apps with JavaScript | About Ionic', Ionic. Accessed: Aug. 30, 2025. [Online]. Available: https://ionic.io/about

[22] 'Capacitor by Ionic - Cross-platform apps with web technology', Capacitor. Accessed: Aug. 30, 2025. [Online]. Available: https://capacitorjs.com/

[23] Microsoft, ‘What is Xamarin? - Xamarin’. Accessed: Aug. 30, 2025. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/xamarin/get-started/what-is-xamarin
[24] Microsoft, ‘The official Xamarin support policy | .NET’, Microsoft. Accessed: Aug. 30, 2025. [Online]. Available: https://dotnet.microsoft.com/en-us/platform/support/policy/xamarin
[25] ‘Flutter architectural overview’. Accessed: Aug. 30, 2025. [Online]. Available: https://docs.flutter.dev/resources/architectural-overview
[26] B. Vijayan, ‘Flutter SDK Architecture’, DEV Community. Accessed: Aug. 30, 2025. [Online]. Available: https://dev.to/binoy123/flutter-sdk-architecture-2gog
[27] ‘Improving rendering performance’. Accessed: Aug. 30, 2025. [Online]. Available: https://docs.flutter.dev/perf/rendering-performance
[28] E. Palm, ‘Rehabilitation at Home of Patients with Neglect Using a Telemedical Intervention: a Security Perspective’, Master’s Thesis, Umeå University, Department of Computing Science, 2016.
[29] Kommunicate, ‘AI Agent for 24/7 Customer Service | Kommunicate’. Accessed: Aug. 30, 2025. [Online]. Available: https://www.kommunicate.io/
[30] ‘LangChain’. Accessed: Aug. 30, 2025. [Online]. Available: https://www.langchain.com
[31] W. W. A. Initiative (WAI), ‘W3C Accessibility Standards Overview’, Web Accessibility Initiative (WAI). Accessed: Aug. 30, 2025. [Online]. Available: https://www.w3.org/WAI/standards-guidelines/
[32] ‘Human Interface Guidelines’, Apple Developer Documentation. Accessed: Aug. 30, 2025. [Online]. Available: https://developer.apple.com/design/human-interface-guidelines
[33] ‘Accessibility’. Accessed: Aug. 30, 2025. [Online]. Available: https://docs.flutter.dev/ui/accessibility-and-internationalization/accessibility
[34] ‘Semantics class - widgets library - Dart API’. Accessed: Aug. 30, 2025. [Online]. Available: https://api.flutter.dev/flutter/widgets/Semantics-class.html
[35] ‘WebAIM: Contrast Checker’. Accessed: Aug. 30, 2025. [Online]. Available: https://webaim.org/resources/contrastchecker/
[36] ‘BestMatch class - string_similarity library - Dart API’. Accessed: Aug. 30, 2025. [Online]. Available: https://pub.dev/documentation/string_similarity/latest/string_similarity/BestMatch-class.html

Appendix 1: User Manual

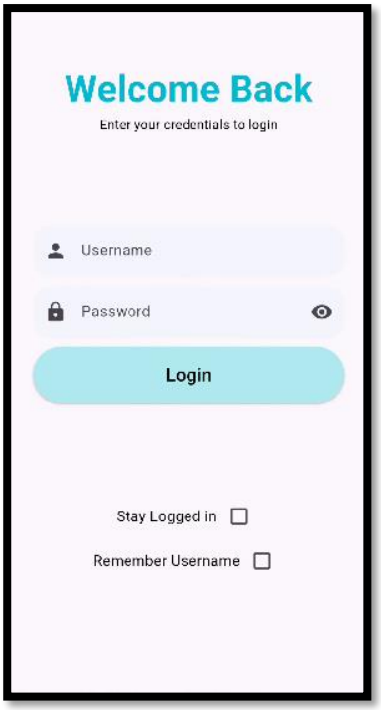
User Manual

Aplicación Multiplataforma En Flutter Con Capacidad De Chatbot Para Proyecto De Telerehabilitación

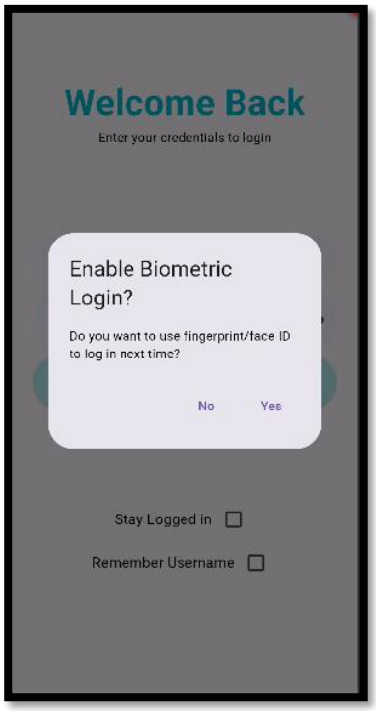
RehaBot is a mobile application designed to support the rehabilitation of individuals with cerebral palsy through therapist directed programs. It provides video-guided exercises, real-time feedback collection, progress tracking and rewards elements to increase motivation and adherence.

1. Getting Started

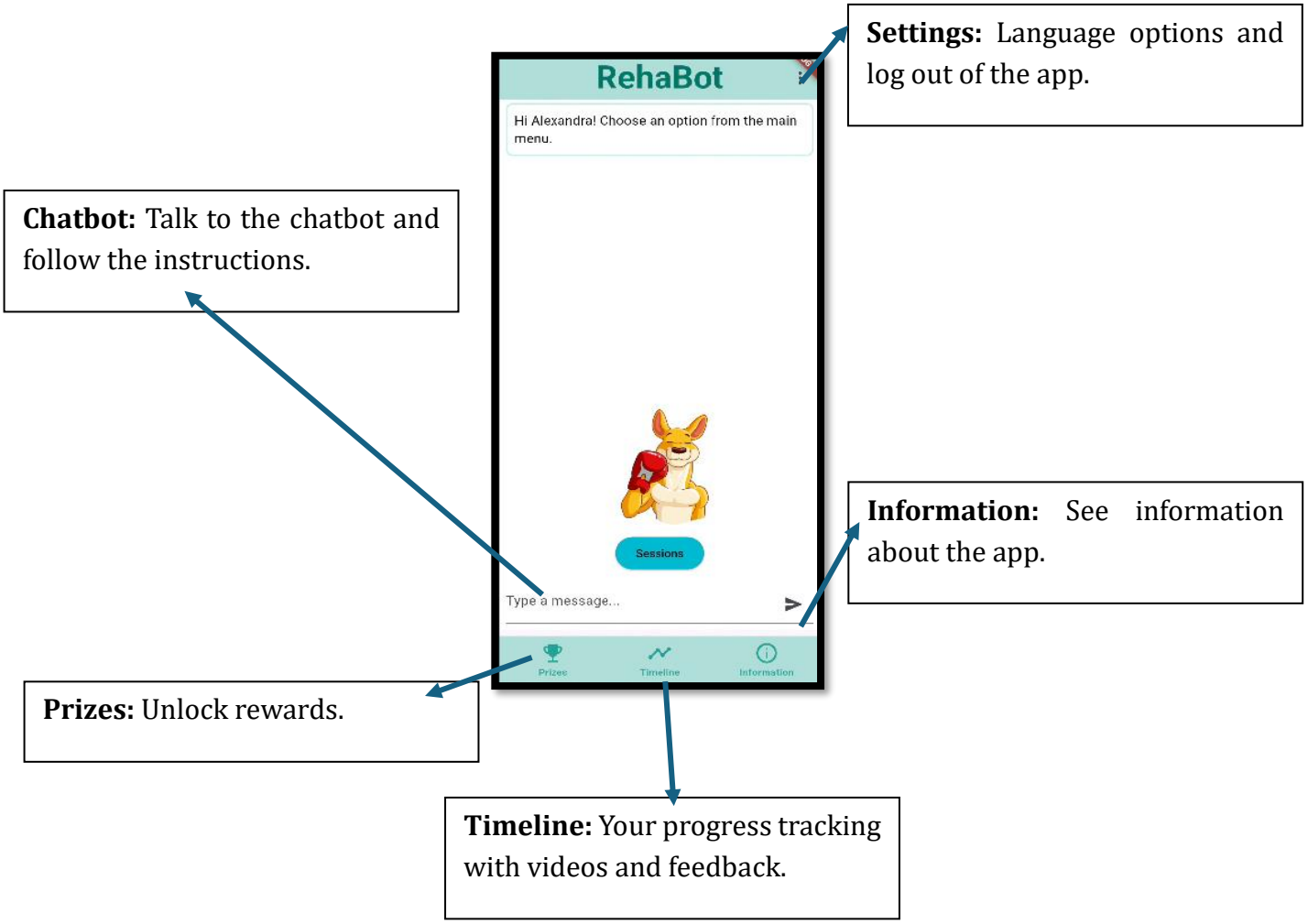
Open the app and log in with your email and password.



Choose if you want to use biometric authentication or not.



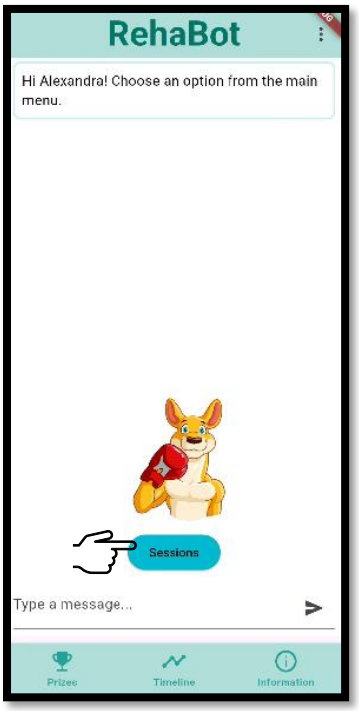
You have now accessed your personalised rehabilitation program. The main screen includes:



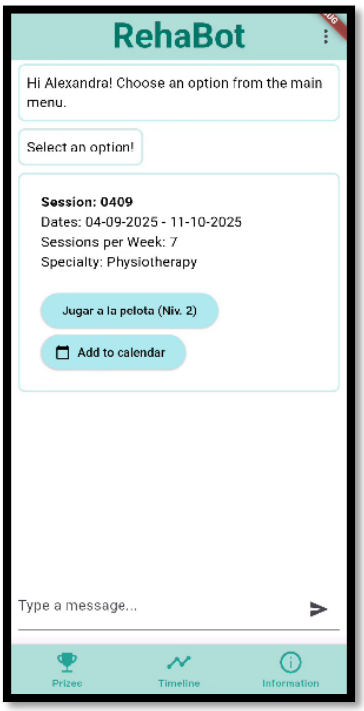
2. Using RehaBot

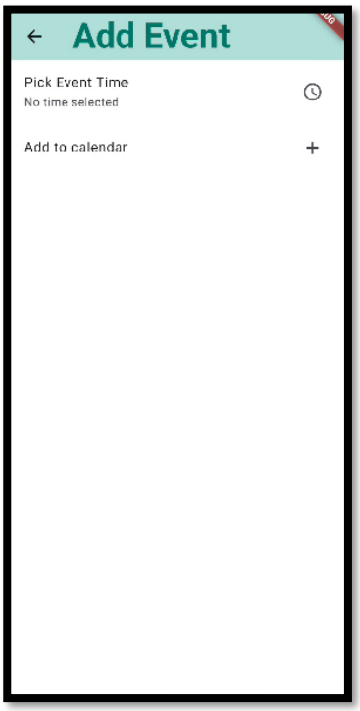
2.1. Sessions and Feedback

To see your sessions tap the sessions button or type it in the message box.

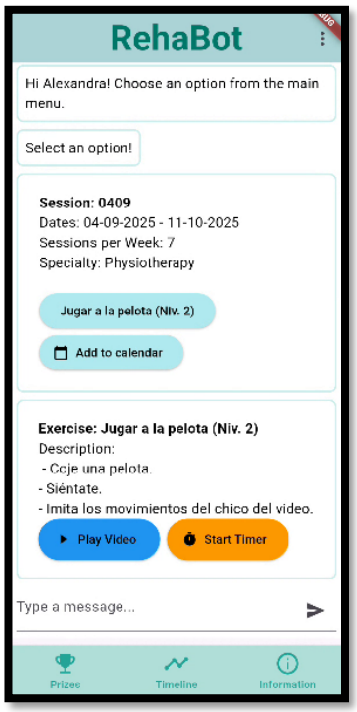


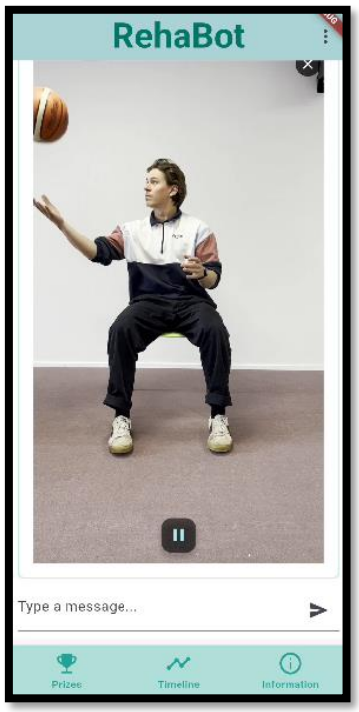
Choose your session and the exercise. You can also add it to your calendar.



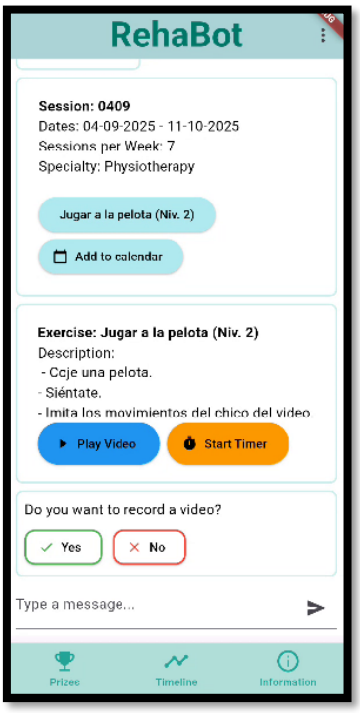


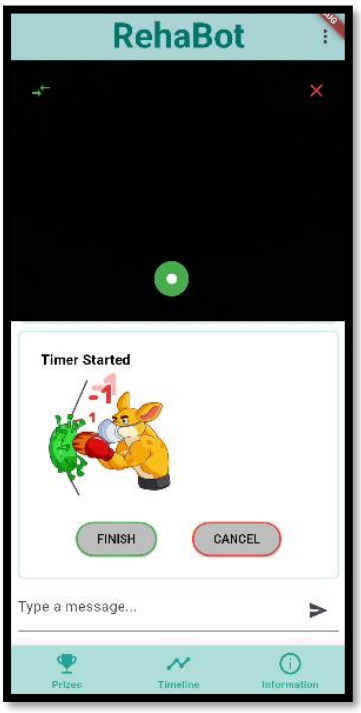
Once you choose the exercise, read the instructions and watch the video.



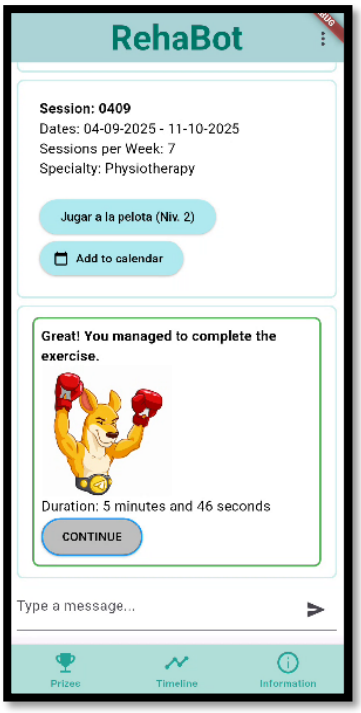


When you’re ready, press start timer and decide if you want to take a video of yourself or not.



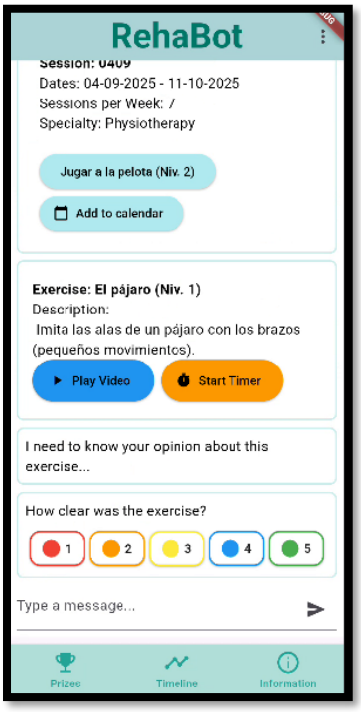


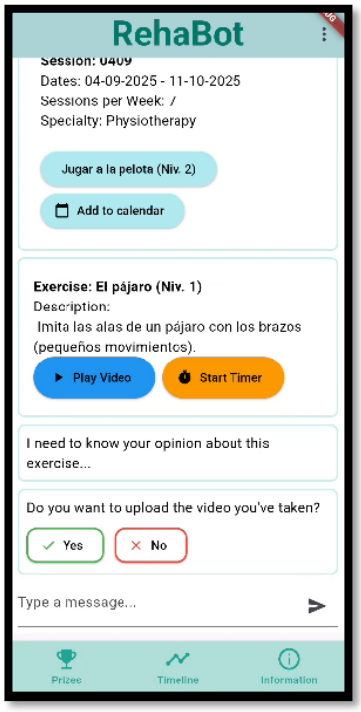
When you are done, press finish and then continue.



If your therapist wants you to give feedback on the exercise, answer the questions.

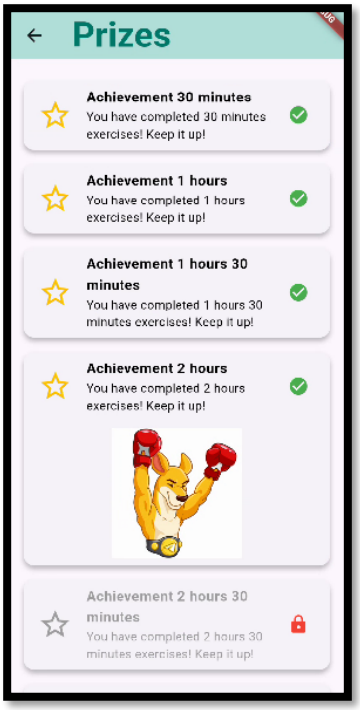
If you took a video, decide if you want to send it!





2.2. Prizes

Completing exercises can earn you badges. The more time you spend, the more you will unblock!

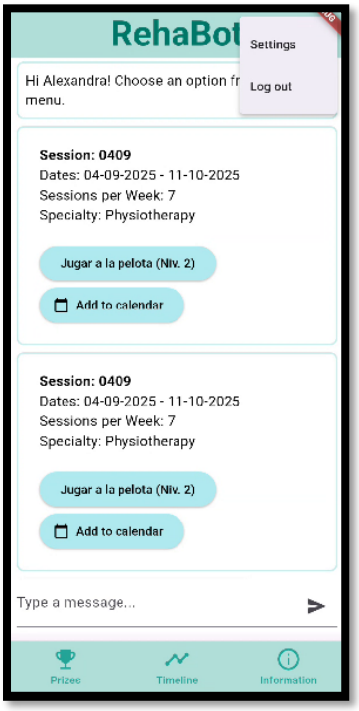


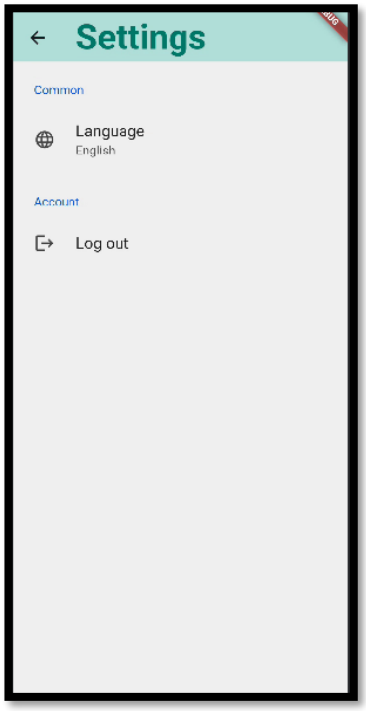
2.3. Timeline

It displays the sequence of exercises you have completed. You can filter by date, feedback or if you took a video or not.

2.4. Settings

To change the language of the app or to log out, go to settings and select your desired language.





3. Troubleshooting & FAQs

- Videos not loading → Check your internet connection.
- Feedback not successful → Make sure you are connected to the internet and restart the app.
- Camera not working → Make sure you have given RehaBot permission to access your camera.
- Cannot log in → Contact RehaBot team.

4. Legal Information

This app manages your personal data in a secure way, according to the GDPR and FAIR principles (Findable, accessible, interoperable and reusable). The information collected is only used in relation to the project PID2021-124515OA-I00, financed by MICIU/AEI/10.13039/501100011033 and by FEDER, UE. It is not used for other purposes and is not shared without consent.

For more information: www.rehabot.eu