ADHD DIAGNOSIS

Attention Deficit/Hyperactivity Disorder based on functional magnetic resonance imaging (FMRI)

Graduation Project

A project submitted in partial fulfilment of the requirements for the degree of Bachelor of Software Engineering

Supervised By:

Dr. Wael Gaballah

Dr. Hala Okasha

Eng. Kholoud Amer

2023/2024

Team Members

#	Student Name	Student ID	Worked on
1	Nour Eldeen Mohamed Mamdouh	802608958	BA / ML / DL
2	Shahd Reda Mohamed Rezk	803085386	Flutter
3	Bansaih Amgad Ali Bakir	803084767	Backend
4	Mohamed Reda Mohamed	800159778	Unity
5	Donia Mohamed Abd Elgwad	803085029	UI/UX
6	Omar Mahmoud Gouda	802603397	Flutter

Abstract

Our project is a revolutionary approach to addressing the needs of children with ADHD. **"ADHD"** Attention-Deficit/Hyperactivity disorder is a complex neurodevelopment condition that affects individuals of all ages. This abstract provides an overview of ADHD, focusing on its core, characteristics, prevalence, and the challenges it poses to those diagnosed with the disorder.

Our innovative approach to ADHD rehabilitation centers on the use of cutting-edge technology to engage children in interactive activities that promote self-confidence, social skills, and creativity. Through our mobile application children can engage in a therapeutic environment that provides a safe and engaging space for them to learn and practice skills.

Our mobile application offers a range of 2D activities that reinforce the same skills, The app's activities are fun and engaging, helping children develop and refine essential social and cognitive skills.

Additionally, our project incorporates artificial intelligence (AI) to provide accurate diagnosis and tailored treatment for individuals with ADHD. Our AI model utilizes data from various sources to provide accurate diagnosis.

Overall, our project represents a significant step forward in the field of ADHD rehabilitation. By combining the latest advancements in AI, and mobile technology, we can provide children with ADHD with a comprehensive and engaging approach to therapy that can help them develop the skills they need to succeed in their daily lives. We believe that our project has the potential to transform the way ADHD is treated, offering new possibilities for improved outcomes and a brighter future for children with ADHD.

Acknowledgement

The success of our project would not have been possible without the contributions of numerous individuals and organizations. On behalf of our team, we would like to express our sincere gratitude to everyone who played a part in making our project a reality.

First and foremost, we extend our heartfelt appreciation to the children with ADHD and their families who participated in our study. Their willingness to engage in our mobile application, and provide invaluable feedback throughout the development process, was critical to the success of our project. We are humbled by their courage and inspired by their resilience.

We would also like to acknowledge our project supervisors:

Dr. Wael Gaballah, Dr. Hala Okasha, Eng. Kholoud Amer for their guidance, support, and encouragement throughout the development process. Their expertise, insights, and unwavering commitment to our project were instrumental in shaping our approach and ensuring its success.

Furthermore, we would like to express our gratitude to the Doctors and specialists who generously shared their knowledge and insights with us. Their contributions helped us develop a more comprehensive understanding of the challenges individuals face with ADHD, and the most effective strategies for addressing them. We are grateful for their guidance and for the wealth of knowledge they shared with us.

In addition, we extend our thanks to our friends and colleagues who provided their support, encouragement, and inspiration throughout the project. Their enthusiasm and positivity were a constant source of motivation and helped us stay focused on our goal of making a meaningful difference in the lives of children with ADHD.

In conclusion, we would like to express our sincere thanks to everyone who contributed to the success of our project. Their contributions, support, and encouragement were critical to our success, and we are grateful for their commitment to improving the lives of children with ADHD.

Contents

Abstract....3
Acknowledgement....4
Chapter 1....13
1.1 Introduction....14
1.2 Problem Statement....14
1.2.1 Inattention....15
1.2.2 Hyperactivity....15
1.2.3 Impulsivity....16
1.2.4 Other Characteristics....16
1.3 Project Objectives....16
1.4 Challenges....17
1.5 Project Contents....18
1.6 Project Timeline....18
Chapter 2....19
2.1 Introduction....20
2.2 Tools....20
2.2.1 UI/UX....20
2.2.2 Game engine....21
2.2.3 Flutter....21
2.2.4 Artificial-Intelligence....22
2.2.5 Back-End....23
2.3 Related Works....25
2.3.1 Focus Genie....26
2.3.2 Impulse....26
2.3.4 Elevate....27
2.3.5 Done-ADHD....28
2.3.6 OFF-Mind....28
Chapter 3....29
3.1 Introduction....30
3.2 User Requirements....30

3.3 System Requirements....30
3.3.1 Functional Requirements....31
3.3.2 Non-Functional Requirements....32
Chapter 4....33
4.1 Introduction....34
4.1.1 Agile Development Life Cycle....34
4.3 Class diagram....36
4.4 sequence diagrams....36
4.5 Requirements tree diagram....39
4.6 Swot Analysis....40
4.7 ERD....40
4.8 Main Components in the System....42
4.9 Business Model....42
Chapter 5....44
5.1 Introduction....45
5.2.1 OFFMIND'S Logo....45
5.2.2 OFFMIND'S Layout....46
Chapter 6....59
6.1 Introduction....60
6.2 Machine learning....61
6.3 Games....86
6.4 Backend....101
6.5 Mobile Application....109
Chapter 7....144
7.1 Conclusion....145
7.2 Future Works....145

List of Tables

Table 2.1 Comparative Analysis of Related work 25
Table 2.2 Focus Genie 26
Table 2.3 Impulse 26
Table 2.4 Vezeeta 27
Table 2.5 Elevate 27
Table 2.6 Done-ADHD 28
Table 2.7 OFF-Mind features 28
Table 3.1 Functional Requirements 31
Table 3.2 Non-Functional Requirements 32

List of Figures

Figure 1.1- Timeline 18
Figure 4.1- Agile phases 34
Figure 4.2- Use Case 35
Figure 4.3 -Class Diagram 36
Figure 4.4 -Sign up Sequence 37
Figure 4.5 -Diagnosis Test Sequence 38
Figure 4.6 -Community Sequence 38
Figure 4.7 -Requirements Tree 39
Figure 4.8 -SWOT Analysis 40
Figure 4.9 -ERD diagram 41
Figure 4.10- System Architecture 42
Figure 4.11 -Business Model 43
Figure 5.1 -OFFMIND Logo 45
Figure 5.2 -Splash screen 46
Figure 5.3 -Intro screens 47
Figure 5.4 -Sign in/up screens 48
Figure 5.5 -Sign in screens 49
Figure 5.6 -Home Screen 49
Figure 5.7 -Note Screens 50
Figure 5.8 -Doctors Screens 51
Figure 5.9 -Centers Screens 52

Figure 5.10 -Specialists Screens 53
Figure 5.11 -Test Screens 54
Figure 5.12 -Test screens cont 55
Figure 5.13 -Games screens 56
Figure 5.14 -Community screens 57
Figure 5.15 -Profile screens 58
Figure 6.1 -Visualization 61
Figure6.2 -Output 62
Figure6.3 -sample file 62
Figure 6.4 -atlases 63
Figure 6.5 -ADHD_AAL_atlas 63
Figure 6.6 -Image to vector 64
Figure 6.7- Vector to dataframe 65
Figure 6.8 -region by time of the image 65
Figure 6.9 -mean intensities 66
Figure 6.10 -features from multiple sub 67
Figure 6.11 -features are most correlate 68
Figure6.12 -store important values 69
Figure6.13 -DX null value 69
Figure 6.14 -drop useless features 70
Figure6.15 -single 'IQ' feature 70
Figure 6.16 -fill null values 71
Figure 6.17 -null-output 71
Figure6.18 -fill gender null 71
Figure 6.19 -fill handedness null 72
Figure 6.20 -fill Verbal IQ null 72
Figure 6.21 -fill Performance IQ null 73
Figure 6.22 -fill IQ null 73
Figure 6.23 -output after fill 73
Figure 6.24 - Normalize helper function 74
Figure 6.25 – Get-Statistics helper function 74
Figure 6.26 – get-accuracies helper functions 75
Figure 6.27 – Evaluate-models helper functions 75
Figure 6.29 - First- Model- Imports 76
Figure 6.30– separate-Data frame 77

Figure 6.31 – separated-Data frame output....77
Figure 6.32 – Evaluate-models....78
Figure 6.33 – Print-confusion-matrices....78
Figure 6.34–confusion-matrices....78
Figure 6.36– Statistics-output....79
Figure 6.37 – Split-Data....79
Figure 6.38– Train-Model....79
Figure 6.39 – Test-Model....80
Figure 6.42 -Second- Model- Imports....80
Figure 6.43 -Condenses- phenotypic....81
Figure 6.44 -subject_region_dataframe....81
Figure 6.45 -subjects_dropped....81
Figure 6.46 -df_region_w_dx....82
Figure 6.47 - Evaluate-models....82
Figure 6.48 - Print-confusion-matrices....82
Figure 6.50 -confusion-matrices....82
Figure 6.51-Model-Evaluation....83
Figure 6.52- Model-Evaluation....83
Figure 6.53 - Split-Data....83
Figure 6.54- Train-Model....83
Figure 6.55 - Test-Model....84
Figure 6.56- Predictions....84
Figure 6.57 - Save-Model....84
Figure 6.58 -Model-API....85
Figure 6.59-Variable Definitions....86
Figure 6.60- Start Method....86
Figure 6.61 - PlayAnimation Method....87
Figure 6.62- DisableButtonOnClick Method....87
Figure 6.63- Namespace Imports....88
Figure 6.64- Class Definition....88
Figure 6.65- Start Method....89
Figure 6.66- StoreInitialPositions Method....89
Figure 6.67-UpdateHearts Method....90
Figure 6.68-ReduceLife Method....90
Figure 6.69 -DragObject Method....90

Figure 6.70-DropObject Method 91
Figure 6.71-Drag and Drop Methods for Specific Objects 91
Figure 6.72-Namespace Imports 92
Figure 6.73-Class Definition 92
Figure 6.74-GoNextScene Method 92
Figure 6.75-Namespace Imports 93
Figure 6.76-Class Definition 93
Figure 6.77-Start Method 94
Figure 6.78-Update Method 94
Figure 6.79-TakeDamage Method 94
Figure 6.80-Namespace Imports 95
Figure 6.81-Class Definition 95
Figure 6.82-Pause Method 96
Figure 6.82-Resume Method 96
Figure 6.83-Restart Method 96
Figure 6.83-Namespace Imports 97
Figure 6.84-Class Definition 97
Figure 6.85-Start Method 98
Figure 6.86-StoreInitialPositions Method 98
Figure 6.87-UpdateHearts Method 99
Figure 6.88-ReduceLife Method 99
Figure 6.89-DragObject Method 100
Figure 6.90-DropObject Method 100
Figure 6.91- Drag and Drop Methods 100
Figure 6.92- LoginReq 101
Figure 6.93- Authentication ..... **Error! Bookmark not defined.**
Figure 6.94- Center's controller ..... **Error! Bookmark not defined.**
Figure 6.94- Chat's controller 104
Figure 6.95- Doctor's controller 105
Figure 6.96- Message's controller 106
Figure 6.97- Messaging function 107
Figure 6.98- Trust Proxies 108
Figure 6.99- Mobile Application Aspects 110
Figure 6.99- needed function 110
Figure 6.100- model folder 111

Figure 6.101-feature folder 111
Figure 6.102-manger folder 112
Figure 6.103-standard cubit state 112
Figure 6.103-Chat Details cubit 113
Figure 6.104-message model 113
Figure 6.104-Chat view 114
Figure 6.105-Chat main 115
Figure 6.106-Chat details 116
Figure 6.107-Chat details cont. 117
Figure 6.107-game feature 118
Figure 6.108-game view 118
Figure 6.109-home feature 119
Figure 6.110-data folder 119
Figure 6.110-call the data 120
Figure 6.111-Medical details 121
Figure 6.112-Appointment view 122
Figure 6.113-home view 123
Figure 6.114-Signup feature 124
Figure 6.115-signup cubit 124
Figure 6.116- login feature 125
Figure 6.117-login cubit 125
Figure 6.118-login view 126
Figure 6.119-splash view feature 127
Figure 6.120-implementation of splash 127
Figure 6.121-intro feature 128
Figure 6.122-view implementation 128
Figure 6.123-profile feature 129
Figure 6.124-data folder 129
Figure 6.125-profile cubit 129
Figure 6.126-profile view 130
Figure 6.127-test feature 131
Figure 6.128-API Service 131
Figure 6.129-Text prediction 132
Figure 6.130-test implementation 133
Figure 6.131-test view 134

Figure 6.132-note feature 134
Figure 6.133-data folder 135
Figure 6.134-name main controller 135
Figure 6.135-Note editor controller 136
Figure 6.136-Note search controller 137
Figure 6.137- note cart 137
Figure 6.138-Note implementation 138
Figure 6.139-Note details 139
Figure 6.140- Note editor view 140
Figure 6.141- note editor view cont. 141
Figure 6.141- Note search view 142
Figure 6.142- main dart file 143

Acronyms

ADHD: Attention Deficit Hyperactivity Disorder.

AI: Artificial Intelligence.

API: Application Programming Interface.

ASD: Autism Spectrum Disorder.

HTML: Hypertext Markup Language.

JSON: JavaScript Object Notation.

LTS: Long-term Support.

ML: Machine Learning.

FMRI: functional magnetic resonance imaging

UI/UX: user interface/user experience

PHP: Hyper Text preprocessor

MVC: Model view controller

MVVM: Model view ViewModel

REST: Representational state transfer

SWOT: Strength, weakness, opportunities, Threats

ERD: Entity relationship diagram

AAL: Automated Anatomical Labeling

IQ: Intelligent quotient

Chapter 1

Introduction

1.1 Introduction

What is Attention-Deficit/Hyperactivity Disorder? Attention-Deficit/Hyperactivity Disorder (ADHD) is a neurodevelopmental disorder that affects both children and adults. Although ADHD can be diagnosed at any age, it is described as a "neurodevelopmental disorder" because symptoms typically emerge in the first childhood years.

Attention-deficit/hyperactivity disorder (ADHD) is a prevalent neurodevelopmental condition affecting an estimated 8-12% of school-aged children. While ADHD can persist into adulthood, symptoms typically emerge during early childhood, a critical period of brain development.

While every child with ADHD experiences it differently, common characteristics include inattention, hyperactivity, and impulsivity. Children with ADHD may also engage in repetitive behaviors, such as fidgeting, talking excessively, or repeating actions (similar to behaviors sometimes seen in children with Autism Spectrum Disorder (ASD)).

It is something that many parents talk about when they describe their children. Your child may be doing these things to help calm themselves during stressful situations or to help occupy or entertain themself.

Children with ADHD may become fixated on specific topics of interest and talk about them extensively, sometimes unaware that others might not share their enthusiasm. The rising prevalence of ADHD is a topic of much discussion, with experts unsure whether it is due to improved diagnosis, a genuine increase in cases, or a combination of both. While there is no cure for ADHD, early diagnosis and appropriate interventions can significantly improve the lives of children with this condition. ADHD can manifest in various ways, making it challenging to develop a one-size-fits-all approach for managing symptoms. This variability highlights the importance of individualized treatment plans.

Our project will be completely different from the rest of the other projects and applications. Through our project, we will rehabilitate children with ADHD through some interactive activities using technology and artificial intelligence to make them more self-confident and more interactive with others and make them highlight their energies and creativity to leave their mark in various fields.

1.2 Problem Statement

The impact of ADHD on a person's life can change significantly as they enter adulthood. some individuals with ADHD develop strategies to manage their symptoms more effectively over time. This can lead to improved social interaction, better integration into their environment, and a reduction in the intensity of behavioral disturbances compared to childhood. In some cases, with proper support, individuals with ADHD may even lead very fulfilling and successful lives. However, for others, the challenges of ADHD persist into adulthood. They may continue to struggle with social interaction and organization. Their symptoms might even become more pronounced as adult responsibilities and complexities increase. This variability underscores the importance of recognizing that ADHD manifests differently in each person. Two individuals with the same diagnosis can present with vastly different strengths, weaknesses, and symptom profiles.

While ADHD manifests differently in each person, some core symptoms appear frequently across individuals:

- Inattention
- Hyperactivity
- Impulsivity

1.2.1 Inattention

ADHD primarily affects a person's ability to focus and maintain attention avoids or does not keep eye contact These behaviors are commonly seen in people with ADHD who struggle with inattention:

- Difficulty sustaining focus they may have trouble concentrating on tasks, especially those that are repetitive or uninteresting.
- Easily distracted, they might be drawn away from a task by sight, sounds, or even their own thoughts.
- Frequent mistakes due to carelessness, they may overlook details or make careless mistakes due to inattention.
- Appearing not to listen they might seem like they're not paying attention when spoken to directly, especially regarding lengthy instructions or explanations.
- Difficulty following through on instructions they may struggle to complete multi-step tasks or follow through on directions.
- Losing or misplaced things they might frequently lose important items like homework, pencils, or even their phone.
- Difficulty with organization they may have trouble organizing tasks and activities, leading to disarray and missed deadlines.
- Daydreaming Frequently their mind might wander often, especially during quiet or monotonous activities.

1.2.2 Hyperactivity

Hyperactivity in ADHD manifests as an excess of energy and an inability to stay still is often associated with excessive energy and an inability to stay still. This hyperactivity can be a hallmark symptom for many people with ADHD, significantly impacting their daily lives. Unlike occasional fidgeting or bursts of energy seen in most children, hyperactivity in ADHD is persistent and disruptive

These behaviors are commonly seen in people with ADHD who struggle with hyperactivity:

- Difficulty staying seated: they have trouble remaining in their seat, especially in situations where it's expected, like classrooms or during meals.
- Always "On the Go" they might seem to have boundless energy and constantly be in motion.
- Excessively talkative they talk excessively and interrupt others in conversations.
- Trouble with play activities requiring stillness quiet games or activities requiring stillness can be very challenging for them.
- Running or climbing in inappropriate situations like running around or climb in situations where it's unsafe or not appropriate.

- Difficulty participating in leisure activities quietly they struggle to engage in hobbies or leisure activities that require calmness or focus.

1.2.3 Impulsivity

Attention deficit hyperactivity disorder (ADHD) can be more than just inattention and hyperactivity. Many people with ADHD also experience difficulties with impulsivity. This means they might act or speak without thinking things through first, leading to challenges in various aspects of life. Unlike a spur-of-the-moment decision, impulsivity in ADHD is frequent and can have negative consequences.

1.2.4 Other Characteristics

- Difficulty keeping things organized
- Struggles with planning ahead
- Learning Difficulties or Delays
- Difficulty managing emotions, leading to outbursts of frustration, anger, or sadness
- Difficulty waiting their turn or following through on conversations.
- Challenges with self-esteem due to struggles with focus
- Anxiety, stress, or excessive worry

1.3 Project Objectives

The aim is to support both parents and children with ADHD by providing a range of resources, tools and technologies to help diagnose, track Manage Symptoms as well as improve communication, socialization, and emotional skills. Specifically, the goal is to help parents diagnose and track their children's progress, provide advice and resources for dealing with behavioral challenges, and provide interactive environments to improve communication and social skills for children with ADHD.

- **Parents objective:**
    1. Helping parents to diagnose their children
    2. Helping parents to know how to deals with their children
    3. Help them to track their children and give them feedback about children's progress
    4. Give them advice about how to deal with them and even in their fits
    5. Provide resources such as learning, training, and play to help children with ADHD

- **Children objective:**

1. Improve their communication skills by providing interactive environments for them
2. How to express their feelings and know other emotion
3. Control behavior
4. Give them some activities to improve their skills
5. Develop conversational skills for interacting with others

- **Software objective:**

1. To do list with reward to improve attention and problem-solving skills.
2. Group that facilitates communication between families and the application.
3. Supports collaboration with professional doctors and ADHD coaches.
4. Help parents develop ways to understand and guide their child's behavior.
5. Options for building a healthy, manageable routine
6. An eating system suitable for people with illness and beneficial foods.
7. Social skills training.
8. Evaluation of the child's condition to see if there is improvement or not.
9. Makes AI-based suggestions.

1.4 Challenges

Our project aims to rehabilitate children with ADHD through interactive activities using technology and artificial intelligence. However, there are several challenges that we may face in the implementation of this project.

Firstly, developing effective and engaging activities for children with ADHD can be a challenge. Where individual needs are paramount, ADHD presents with a wider range of symptoms. However, tailoring activities to a child's specific strengths and weaknesses can significantly boost their engagement.

Secondly, the development of a mobile application for ADHD treatment needs to be user friendly and accessible to children with different levels of abilities. It also needs to be regularly updated and improved to provide the best possible support for the children.

Lastly, the development of an Artificial Intelligence (AI) model to diagnose children with ADHD requires extensive data collection and analysis, which can be time-consuming and challenging. Furthermore, the accuracy of the model needs to be validated and tested to ensure that it is reliable and effective in diagnosing autism.

Despite these challenges, we believe that our project has the potential to make a significant positive impact on the lives of children with ADHD, by providing them with personalized, engaging, and effective treatment options.

1.5 Project Contents

Our ADHD Application will help to diagnose ADHD for children by using artificial intelligence techniques and help them to feel like normal children.

- First step: parents will sign up to create an account in the application and enter his information.
- Second step: parents will make a diagnosis using FMRI or Phenotype data of the child and determine if the child has ADHD or not.
- Last step: if the result is that he has ADHD, the application will recommend doctors and specialists specialized in ADHD. also direct them to our community that will help him to improve his behavior, interact with other people, make him more confident, and learn different skills.

Besides, the activities application will provide tips for parents to learn how to deal with their child and what to do if they have a problem

1.6 Project Timeline

This chapter outlines the comprehensive plan that was devised to guide the completion of the project. The plan details the various stages of development, from conceptualization to implementation, and provides a roadmap for addressing any challenges that may arise along the way. In addition, the plan delineates the roles and responsibilities of the project team members and defines the expected outcomes and deliverables for each stage of the project. By adhering to this plan, we aimed to ensure the project's timely completion while meeting all of its objectives and requirements. As shown in Figure 1.6

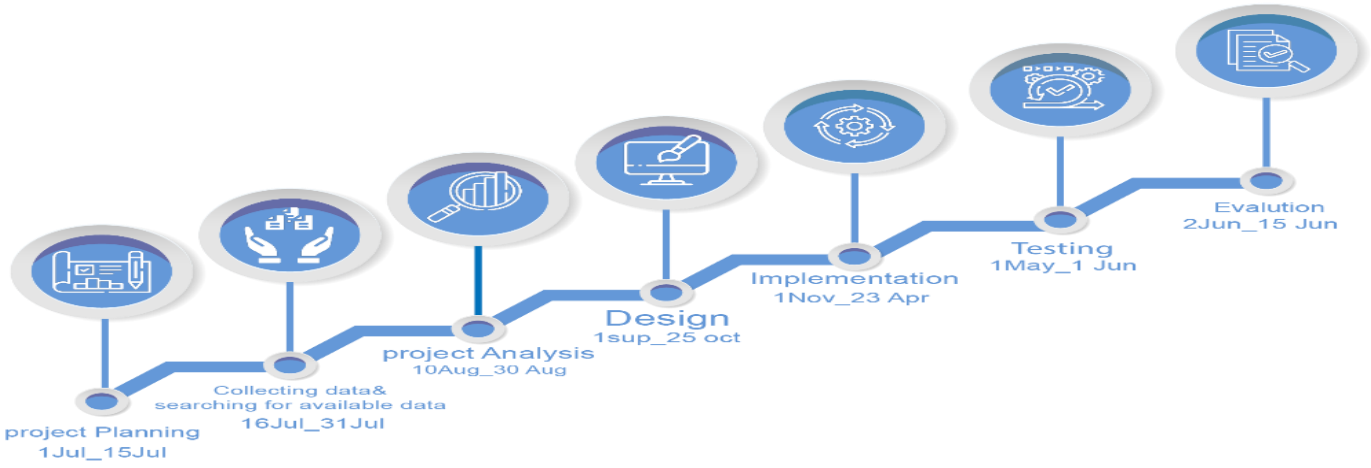


Figure Timeline 1.1

Chapter 2

Background and Related Works

2.1 Introduction

What are we going to talk about in this chapter? In this chapter, we will define all the tools and techniques that we used to build our project, and more background on ADHD and our application, and in the end, we will show what other applications have reached, the defects and advantages of each, and make a comparison between our program and related systems. The Background chapter's emphasis on these technologies and terminologies allows readers to grasp the project's context, challenges, and potential implications. It provides a meaningful framework for decision-making and contributions within the project's domain. By familiarizing themselves with these foundational concepts, readers can navigate the project's objectives and strategies effectively. Ultimately, the Background chapter serves as a crucial steppingstone toward a comprehensive understanding of the project

2.2 Tools

2.2.1 UI/UX

Adobe Illustrator

Figma is a popular design and prototyping tool that is primarily used for user interface (UI) and user experience (UX) design. It operates as a cloud-based service, allowing for real-time collaboration and seamless sharing among team members. Here are some key features and functionalities of Figma

Adobe Photoshop

Adobe Photoshop is a graphics editing and photo manipulation software developed by Adobe Inc. It is one of the most powerful and widely used design and editing tools globally, utilized in various fields such as graphic design, photography, digital art, and advertising

Adobe Illustrator

Adobe Illustrator is a vector graphics editor developed by Adobe Inc. It is widely used by graphic designers, illustrators, and artists to create scalable graphics for various media, including print, web, and mobile.

2.2.2 Game engine

Unity

What is unity? Unity is a cross-platform game engine developed by Unity Technologies. Unity gives users the ability to create games and experiences in both 2D and 3D, and the engine offers a primary scripting Application Programming
Interface (API) in C#. To use the Unity framework, you should be familiar with C# code and logic and the basics of 2D and 3D. What makes Unity so appealing to developers is that it's simple to use so that you don't need to start from scratch. Unity takes care of things like 3D rendering, physics and collision detection. Essentially, Unity has physics built-in so that developers don't need to take care of every last detail. Supported Platform Unity is a cross-platform engine. The Unity editor is supported on Windows, macOS and the Linux platform, while the engine itself currently supports building games for more than 19 different platforms, including mobile, desktop, consoles, and virtual reality. Unity 2020 Long-term Support (LTS) officially supports the following platforms:

- Mobile platforms
- Desktop platforms
- Web platform
- Console platforms

Visual Studio Code

Visual Studio Code is a popular code editor developed by Microsoft. It's known for being lightweight, open-source, and highly customizable. While it can be used for various programming languages, it offers excellent support for Flutter development through extensions.

2.2.3 Flutter

Flutter is an open-source User Interface (UI) software development kit created by Google. It is used to develop cross-platform applications for Android, iPhone Operating System (IOS), Linux, macOS, Windows, Google Fuchsia, and the web from a single codebase.

Android Studio

Android Studio provides a comprehensive development environment with various tools and integrations to streamline the process of building Android applications. It doesn't have a single core engine like Flutter, but rather leverages the Android SDK and various development tools for a smooth development experience.

Dart-language

Dart is a client-optimized language for developing fast apps on any platform. Its goal is to offer the most productive programming language for multi-platform development, paired with a flexible execution runtime platform for app frameworks. Languages are defined by their technical envelope, the choices made during development that shape the capabilities and strengths of a language. Dart is designed for a technical envelope that is particularly suited to client development, prioritizing both development (sub-second stateful hot reload) and high-quality production experiences across a wide variety of compilation targets (web, mobile, and desktop). Dart also forms the foundation of Flutter. Dart provides the language and runtimes that power Flutter apps, but Dart also supports many core developer tasks like formatting, analyzing, and testing code.

2.2.4 Artificial-Intelligence

AI is a wide-ranging branch of computer science concerned with building smart machines capable of performing tasks that typically require human intelligence. AI is an interdisciplinary science with multiple approaches, but advancements in machine learning and deep learning are creating a paradigm shift in virtually every sector of the tech industry.

Artificial intelligence allows machines to model, and even improve upon, the capabilities of the human mind. From the development of self-driving cars to the proliferation of smart assistants like Siri and Alexa, AI is a growing part of everyday life. As a result, many tech companies across various industries are investing in artificially intelligent technologies.

Machine Learning

Machine learning (ML) is a subfield of artificial intelligence (AI) that empowers computers to learn from data without explicit programming. This enables them to perform increasingly complex tasks and make data-driven predictions.

Python

Python is a popular general-purpose programming language that can be used for a wide variety of applications. It includes high-level data structures, dynamic typing, dynamic binding, and many more features that make it as useful for complex application development as it is for scripting or "glue code" that connects components together. It can also be extended to make system calls to almost all operating systems and to run code written in C or C++. Due to its ubiquity and ability to run on nearly every system architecture, Python is a universal language found in various applications.

Jupyter

Jupyter is an open-source project that revolves around interactive computing. It offers tools and standards for creating documents that combine code, visualizations, and explanatory text. This makes it particularly useful for data science, scientific computing, and machine learning.

Flask

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies, and several common framework-related tools.

2.2.5 Back-End

The back end refers to parts of a computer application or a program's code that allow it to operate and that cannot be accessed by a user. Most data and operating syntax are stored and accessed in the back end of a computer system. Typically, the code is comprised of one or more programming languages. The back end is also called the data access layer of software or hardware and includes any functionality that needs to be accessed and navigated to by digital means.

Laravel

Laravel is a web framework for making custom applications. It runs on PHP and is entirely free and open source. Laravel is primarily used for building custom web apps using PHP. It's a web framework that handles many things that are annoying to build yourself, such as routing, templating HTML, and authentication. Laravel is entirely server-side, due to running on PHP, and focuses heavily on data manipulation and sticking to a Model-View Controller design. A framework like React might put most of its attention on user interaction and shiny features, but Laravel simply presents a solid foundation for you to build off and does it right.

How Does Laravel Work?

Laravel uses a design pattern called Model-View-Controller, or MVC. The "Model" represents the shape of the data your application operates on. If you have a table of users, each with a list of posts they've made, that's your model. The "Controller" interacts with this model. If a user requests to see their posts page, the controller talks to the model (often just the database) and retrieves the info. If the user wants to make a new post, the controller updates the model. The controller contains most of the logic for your application. The controller uses that info to construct a "View". The view is a template with which the model can be plugged into and displayed, and it can be manipulated by the controller. The view is all your application's HTML components

PHP

PHP (Hypertext Preprocessor) stands as a widely used, open-source scripting language specifically designed for web development. Its popularity stems from its versatility, ease of use, and ability to handle a wide range of web-related tasks. Its dynamic nature, and the rich ecosystem of frameworks make it a powerful tool for building modern web applications. While dynamic typing offers advantages, it's essential to prioritize secure coding practices.

MySQL

MySQL is a relational database management system based on SQL – Structured Query Language. The application is used for a wide range of purposes, including data warehousing, e-commerce, and logging applications. The most common use for MySQL, however, is for the purpose of a web database. It can be used to store anything from a single record of information to an entire inventory of available products for an online store. In association with a scripting language such as PHP or Perl (both offered on our hosting accounts) it is possible to create websites which will interact in real-time with a MySQL database to rapidly display categorized and searchable information to a website user.

Postman

Postman is a popular collaboration platform for API development. It simplifies each step of building an API and streamlines collaboration so you can create better APIs faster. Here's an overview of its key features and functionalities

REST

Representational State Transfer (REST) is an acronym for Representational State Transfer and an architectural style for distributed hypermedia systems. Roy Fielding first presented it in *2000* in his famous dissertation. Like other architectural styles, REST has its guiding principles and constraints. These principles must be satisfied if a service interface needs to be referred to as RESTful. A Web API (Web Service) conforming to the REST architectural style is a REST API.

2.3 Related Works

There exist several related works in this field, but some of them may have certain drawbacks or limitations. Although they may share similarities with the current study, they may not have explored the topic thoroughly or may have used outdated or less accurate methodologies. It is important to carefully consider the limitations of these works before drawing any conclusions or making any decisions based on their findings. Ultimately, the goal is to build upon the strengths of these previous works while addressing their limitations to contribute new and valuable insights to the field.

Table 2.1 Comparative Analysis of Related work

	FOCUS GENIE	Impulse Brain Training Games	Vezeeta	Elevate	Done-ADHD	OFF-Mind
Diagnosis	✗	✗	✗	✗	✓	✓
Health & Mental care	✗	✗	✓	✗	✗	✓
Brain Training Games	✗	✓	✗	✓	✗	✓
Task Organization	✓	✗	✗	✗	✗	✓

2.3.1 Focus Genie

Focus Genie is an ADHD management platform that offers you a way to better understand and behaviorally manage ADHD, build healthy routines, and enhance daily functioning

Table 2.2 Focus Genie

Focus Genie	
Paid	✓
Reminder	✓
Community	✓
Diagnosis	✗
Health & Mental Care	✗
Brain Training Games	✗
Task Organization	✓

2.3.2 Impulse

Brain Training App offers you a great way to perform mental training by playing entertaining and challenging mind games. Our quick brain workouts along with proper physical exercising and diet may help to keep your brain clear, sharp and ready for day-to-day life challenges

Table 2.3 Impulse

Impulse	
Paid	✓
Reminder	✗
Community	✗
Diagnosis	✗
Health & Mental Care	✗
Brain Training Games	✓
Task Organization	✗

2.3.3 Vezeeta

Vezeeta is the doorway to an improved healthcare experience. Find the best doctors, read verified reviews, book appointments instantly with 20,000+ verified doctors, and more. You can also order your medicines and all your pharmacy needs will be delivered right away.

Table 2.4 Vezeeta

Vezeeta	
Paid	✗
Reminder	✗
Community	✗
Diagnosis	✗
Health & Mental Care	✓
Brain Training Games	✗
Task Organization	✗

2.3.4 Elevate

Elevate is a brain training program designed to improve your mind's focus, memory, speaking abilities, processing speed, math skills, and more. Each person is provided with a personalized training program that adjusts over time to maximize results.

Table 2.5 Elevate

Elevate	
Paid	✓
Reminder	✗
Community	✗
Diagnosis	✗
Health & Mental Care	✗
Brain Training Games	✓
Task Organization	✗

2.3.5 Done-ADHD

Done is a comprehensive ADHD management app that provides a holistic approach to treating the condition. It combines virtual appointments with medical professionals, personalized treatment plans, medication delivery, and symptom tracking tools to help users manage their ADHD effectively.

Table 2.6 Done-ADHD

Done-ADHD	
Paid	✓
Reminder	✓
Community	✓
Diagnosis	✗
Health & Mental Care	✗
Brain Training Games	✗
Task Organization	✓

2.3.6 OFF-Mind

Our project is distinct from any other in its aim to rehabilitate children with ADHD using cutting-edge technology and AI. Our interactive activities will foster self-confidence and promote social interaction, empowering these children to unleash their creativity and reach their full potential. With our project, we seek to cultivate a generation of individuals who can make their mark in diverse fields, and overcome the challenges posed by ADHD. Join us on this journey towards a more inclusive and innovative future.

Table 2.7 OFF-Mind features

OFF-Mind	
Paid	✗
Reminder	✓
Community	✓
Diagnosis	✓
Health & Mental Care	✓
Brain Training Games	✓
Task Organization	✓

Chapter 3

Project Requirements

3.1 Introduction

Success of any project depends on a clear understanding of its goals and requirements. In the case of our ADHD project, the ultimate goal is to provide an innovative and effective approach to rehabilitating children with ADHD. To achieve this goal, we have identified several key requirements that must be met.

These requirements encompass a range of factors, including the needs of our target population, the latest research and best practices in the field of ADHD treatment, and the technical and logistical considerations involved in developing and deploying our treatment approach.

In this section, we will outline the project requirements in detail, providing a comprehensive overview of the factors that will shape the design, development, and implementation of our ADHD treatment program.

3.2 User Requirements

The application is designed for three distinct user groups: parents of ADHD children, the children themselves, and doctors. Parents can use the application to conduct diagnoses by uploading FMRI and Phenotypic data of their child for diagnosis, while also receiving information and advice related to ADHD. Children can use the application to engage in various educational and visual activities, including memory enhancement.

Additionally, doctors can add relevant information about themselves to the application, increasing their visibility to potential patients.

We have 10 user requirements:

1- Sign up for an account

2- Sign in to your account

3- Create or personalize a profile avatar

4- Find a doctor who specializes in ADHD to get a phenotypic diagnosis

5- Access advanced FMRI diagnostics

6- Search for ADHD treatment centers in your area

7- Find specialists who can create a personalized treatment plan

8- Join a supportive community forum to connect with others with ADHD

9- Share experiences and advice with others who understand

10- Learn about treatment options

3.3 System Requirements

In order for a software application to operate smoothly and effectively, it is imperative that the system meets certain requirements. Failure to meet these requirements can result in a host of issues, ranging from installation problems to performance issues
that hinders the functionality of the software.

In general, system requirements can be categorized into two broad types: ***functional requirements*** and ***non-functional requirements***. Understanding and meeting both types of requirements is crucial for ensuring the optimal performance of a software application.

3.3.1 Functional Requirements

Functional requirements are a crucial aspect of any system, as they pertain directly to the services offered by the system itself. These requirements describe the desired behavior of the system and are often listed as a series of functions.

In our system, we have identified the necessary functions and provided a detailed description of each one in a table format. By doing so, we ensure that all necessary functions are accounted for and that our system meets the needs of its intended users as shown in Table 3.1

Table 3.1 Functional Requirements

Functional Requirement	Description
Sign up	*Users sign up by providing information about themselves and their child, including name, gender, number, email, and password. Doctors sign up with their own information.*
Sign in	*User logs in by entering their email and password or changing their password.*
Add photo	*User can choose an avatar or upload a photo of their child.*
ADHD diagnosis	*With the help of AI technology, parents can upload and analyze fMRI images. Provide diagnostic reports based on fMRI data and Phenotypic Information*
Centers	*Parents can choose any center with details about it that are specialized in Treatment of ADHD.*
Set Reminder	*Users can set a reminder to a future time to notify them when this time is coming in.*
Get information	*Parents can access information about ADHD through the application.*
Search for doctor	*Parents can search for and filter doctors to find the best one to communicate with.*
Community	*If the child is diagnosed with ADHD, the application will direct parents to activities that can help improve behavior, includes improving problem-solving and social skills, and building focus and memory.*
Games	*Enhance cognitive skills through engaging and targeted activities.*
Direct Messages	*Users can send messages to doctors & other users.*
Progress Tracking	*Users and clinicians enable to analyze trends and make informed adjustments to treatment plans.*
To-Do Lists	*Create and manage to-do lists for daily tasks, Set priorities and deadlines for tasks. Notes Provide a space for jotting down notes and ideas. Allow categorization and search functionality for notes. Reminders Set reminders for tasks, appointments, and medication.* *Provide notifications and alerts.*

3.3.2 Non-Functional Requirements

Non-functional requirements are a crucial aspect of software development, as they define the characteristics that are necessary for a system to operate effectively and efficiently.

Unlike functional requirements, which describe the features and behavior of the system, non-functional requirements define the performance, security, usability, and other attributes that are essential for a system to meet the needs and expectations of its users. These requirements play a critical role in ensuring that a software application is reliable, maintainable, and scalable, and can be used safely and effectively by its intended users.

As such, they are an important consideration throughout the entire software development life cycle, from initial design to final implementation and beyond. Attributes that non-functional requirements define include:

Table 3.2 Non-Functional Requirements

Non-Functional Requirement	Description
Accessibility	*Easy design that is suited for different communities to use.*
Availability	*Available for low versions of android devices as well as high versions.*
Security and Privacy	*Implement strong security measures to protect user data. Comply with relevant health data regulations (e.g., HIPAA).*
Minimize response time (high speed)	*System response within a few seconds.*
Performance	*Ensure the app performs well even with large volumes of data. Optimize load times for games and diagnostic tools.*
Compatibility	*Support multiple devices (smartphones, tablets, web browsers). Ensure compatibility with major operating systems (iOS, Android).*
Usability	*Ensure the app is user-friendly for both children and parents. Include tutorials and help sections to guide new users.*
Maintainability	*Capable of being maintained cost-effectively over its expected lifetime.*
Fault tolerance	*Ability of a system to continue operating without interruption when one or more of its components fail.*

Chapter 4

System Analysis

4.1 Introduction

The present chapter aims to provide an in-depth overview of the essential components of System Analysis and Design, including Software Life Cycle and Software Requirements. Our primary focus is to comprehend the system's anticipated outcomes,
stakeholders, and user interactions with the services provided.

To achieve our objectives, we will adopt **Agile Development methodologies** in this project. The Agile approach emphasizes iterative development and collaboration between cross functional teams to deliver high-quality software that meets customer requirements.

4.1.1 Agile Development Life Cycle

Agile Development Life Cycle is an iterative approach to project management and software development that helps teams deliver value to their customers quickly and efficiently. The Agile methodology emphasizes continuous collaboration between developers, customers, and other stakeholders throughout the project lifecycle.

Unlike traditional approaches, Agile teams deliver work in small, consumable increments, allowing for frequent feedback and adaptation to changing requirements. This results in improved transparency, faster time to market, and higher customer satisfaction.

The Agile Development Life Cycle comprises five key phases, namely planning, design, implementation, testing, and maintenance. During the planning phase, the team defines project goals, identifies customer needs, and creates a roadmap for the project. In the design phase, the team creates detailed specifications and designs the system architecture. The implementation phase involves coding and integrating various system components. In the testing phase, the team verifies that the system works as intended and meets customer requirements. Finally, the maintenance phase involves ongoing support and updates to ensure the system remains efficient and effective over time.

By adopting Agile Development Life Cycle, teams can improve their ability to adapt to changing market conditions and customer needs while maintaining a high level of quality in their software products.

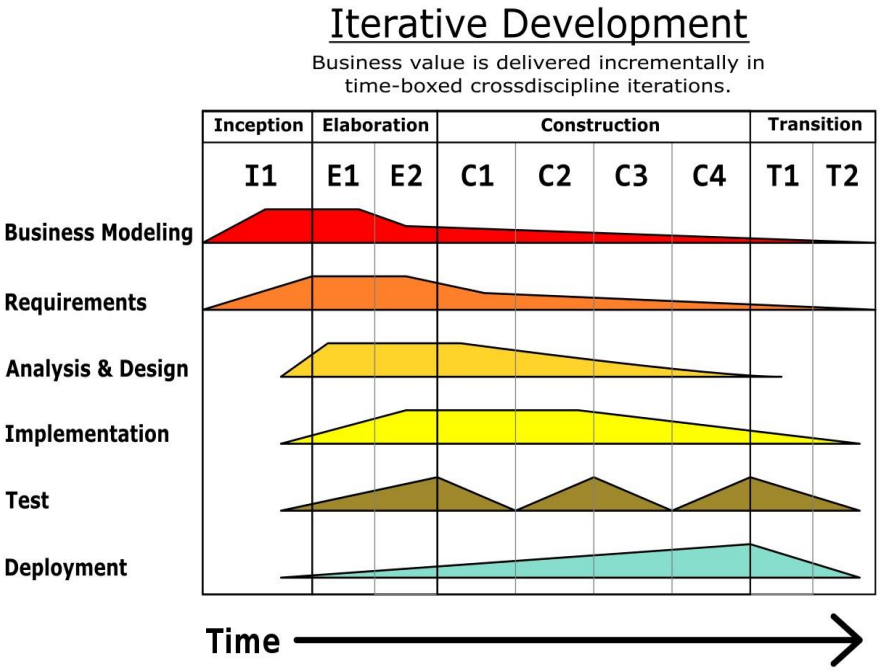


Figure 4.1 Agile phases

4.2 Use case diagram

Use case diagram provides a high-level overview of the functional requirements of a system and the interactions between its actors and the system. It consists of actors, use cases, and relationships between them. Actors are external entities that interact with the system, while use cases represent the specific actions or functions that the system provides to the actors. Relationships between actors and use cases can be either association, indicating a general interaction, or dependencies, indicating that one use case depends on another.

By representing the system's functionality in a visual format, use case diagrams can help stakeholders to better understand the system's requirements and ensure that they meet the needs of its users. Use case diagrams can also serve as a starting point for developing more detailed use cases that describe the specific steps involved in each use case.

The use case explains the function can user and admin make in the application.

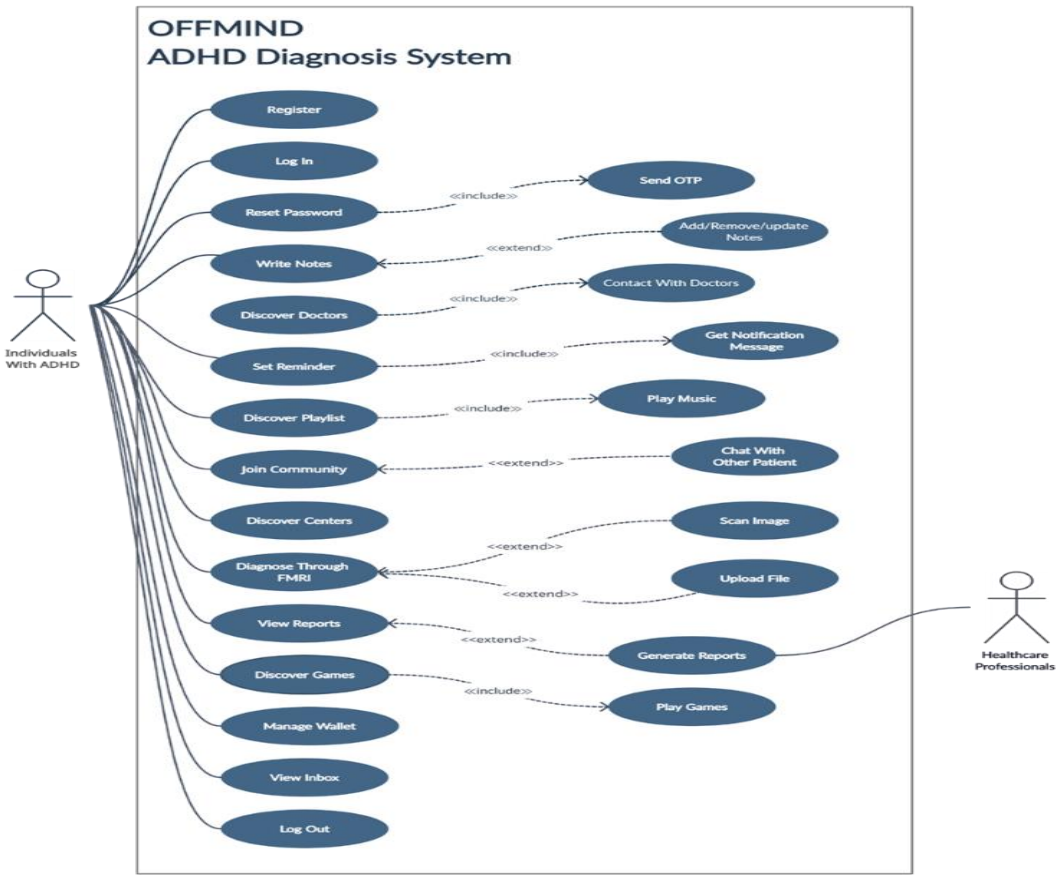
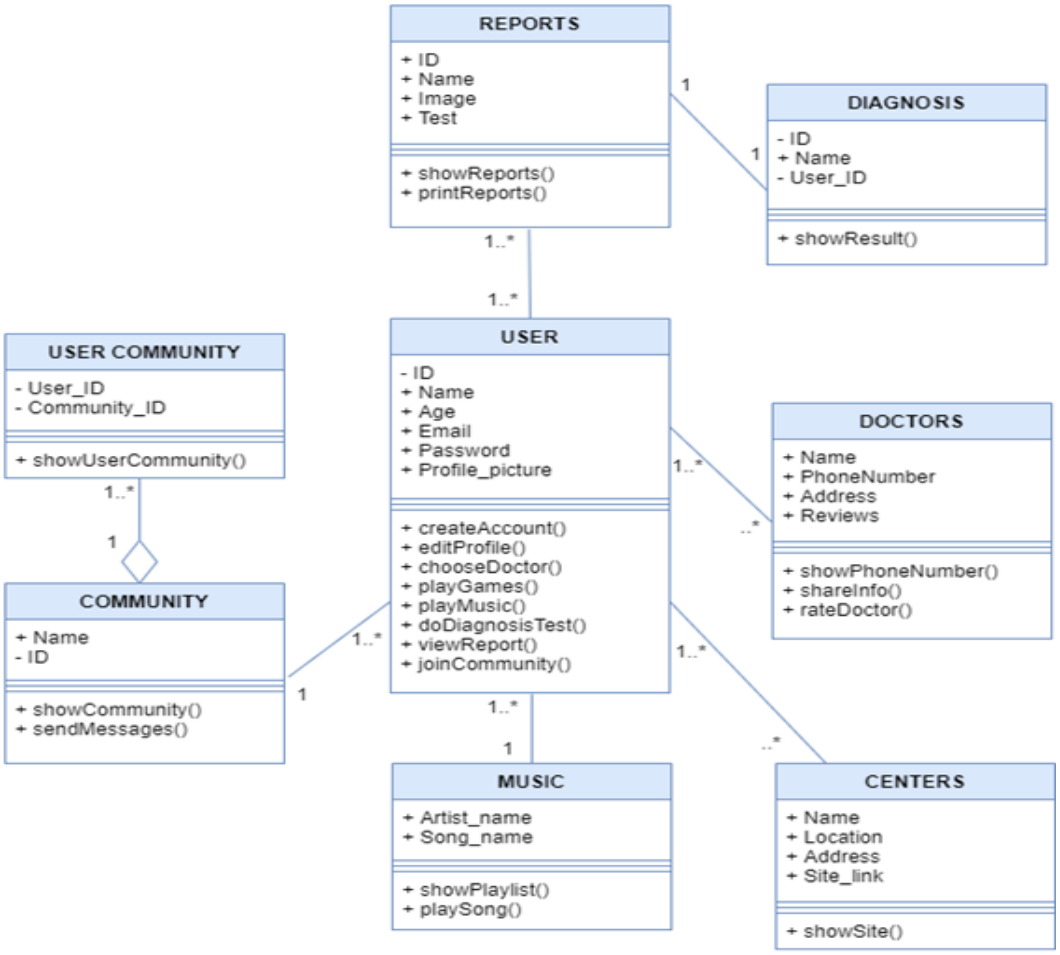


Figure 4.2 Use Case

4.3 Class diagram

Class design phase is a fundamental step in our software development process, focusing on the detailed implementation of individual classes within our system. During this phase, we define the internal structure and behavior of each class, including attributes, methods, visibility, and relationships. By meticulously considering the requirements and specifications, we make informed decisions about data types, algorithms, and design patterns that best align with our system's objectives.

The Actual Class design stage bridges the gap between the high-level conceptual model and the actual coding and implementation. It involves translating abstract representations, such as class diagrams, into concrete and executable code. Through the Actual Class design, we ensure that our classes are optimized, modular, and maintainable while adhering to coding standards and best practices. This phase plays a crucial role in transforming our software architecture into a functional and reliable system, providing developers with clear guidelines for coding and promoting consistency and scalability across the project



4.3 Class Diagram Figure

4.4 sequence diagrams

Sequence diagrams play a pivotal role in our software development process, serving as a robust visual aid that offers a comprehensive depiction of the dynamic behavior and intricate interactions among objects or components within our system. These diagrams act as a window into the chronological sequence of messages exchanged between objects, showcasing their collaborative efforts to achieve specific functionalities. By presenting a clear visualization of the flow of control, sequence diagrams empower us to delve into the intricate details of method invocations, parameter passing, and the precise order of events.

One of the key advantages of sequence diagrams lies in their ability to illuminate the interactions between different elements of our system. Through their graphical representation, sequence diagrams facilitate the identification of potential bottlenecks, ensuring that performance issues and inefficiencies can be promptly addressed. Moreover, they provide invaluable insights into system behavior, allowing us to gain a deeper understanding of how various components interconnect and function together.

Ultimately, sequence diagrams stand as a testament to our commitment to excellence in software development. Their meticulous portrayal of object interactions not only aids in verifying the correctness of our design but also lays the foundation for efficient troubleshooting and future enhancements. By harnessing the power of sequence diagrams, we can elevate our development process, ensuring the delivery of robust and reliable software solutions

The signup explains the steps the user takes when signup

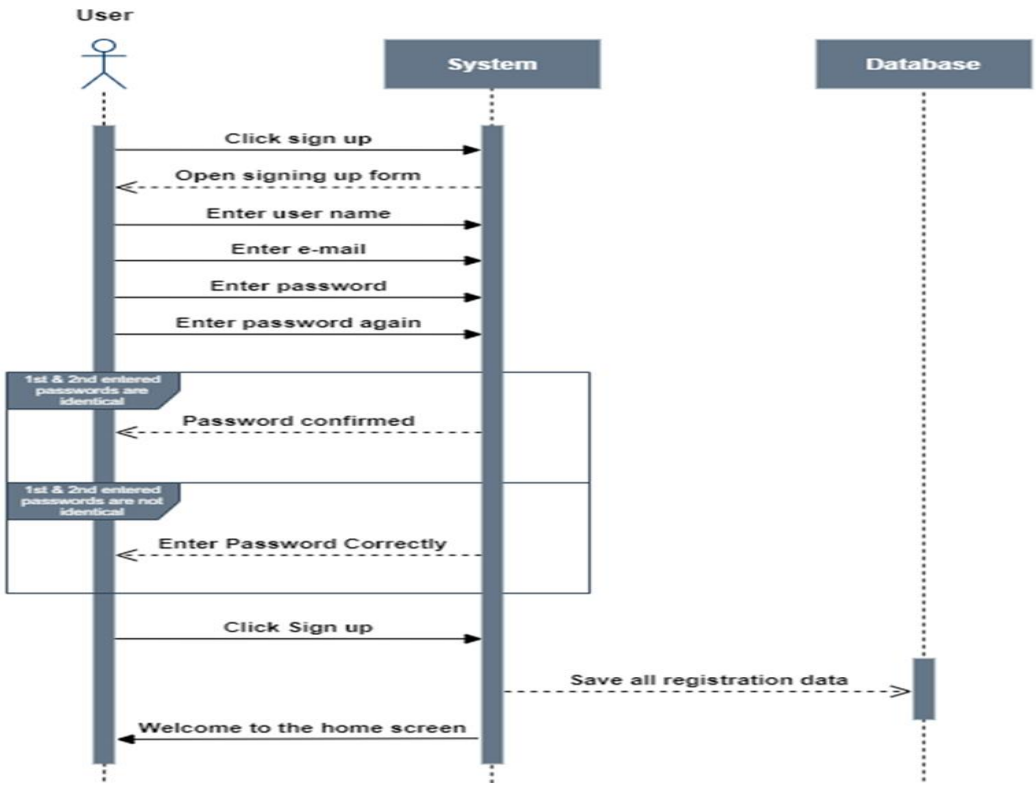


Figure 4.4 Sign up Sequence

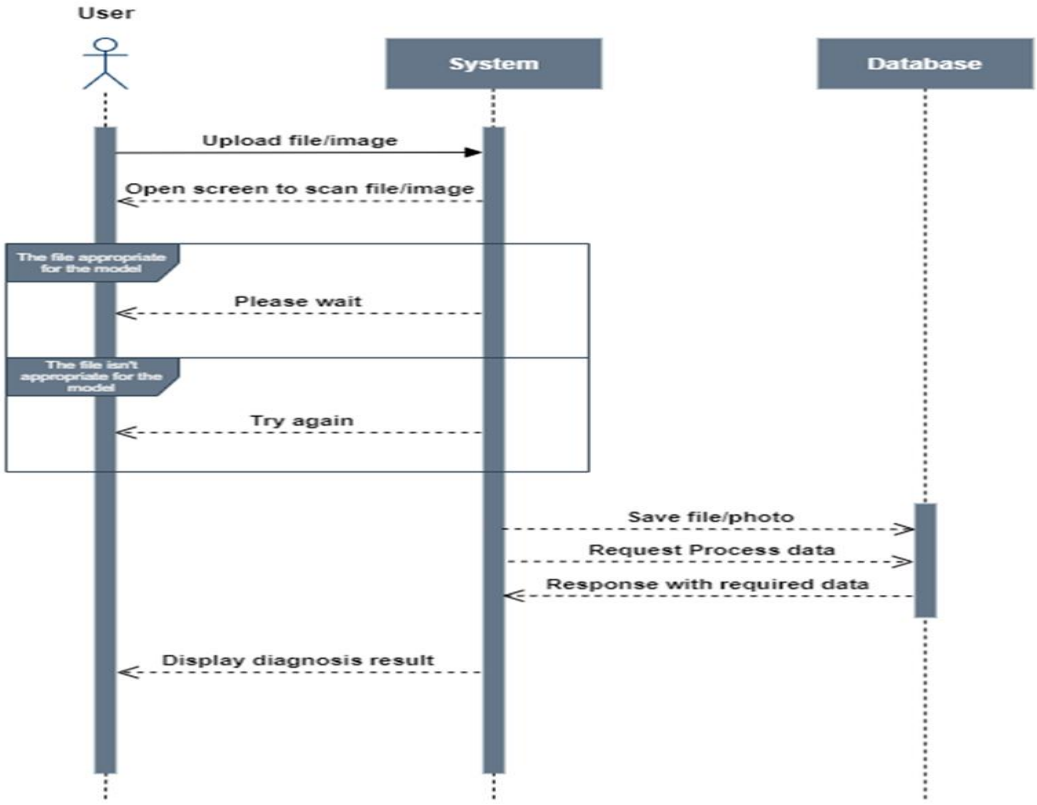


Figure 4.5 Diagnosis Test Sequence

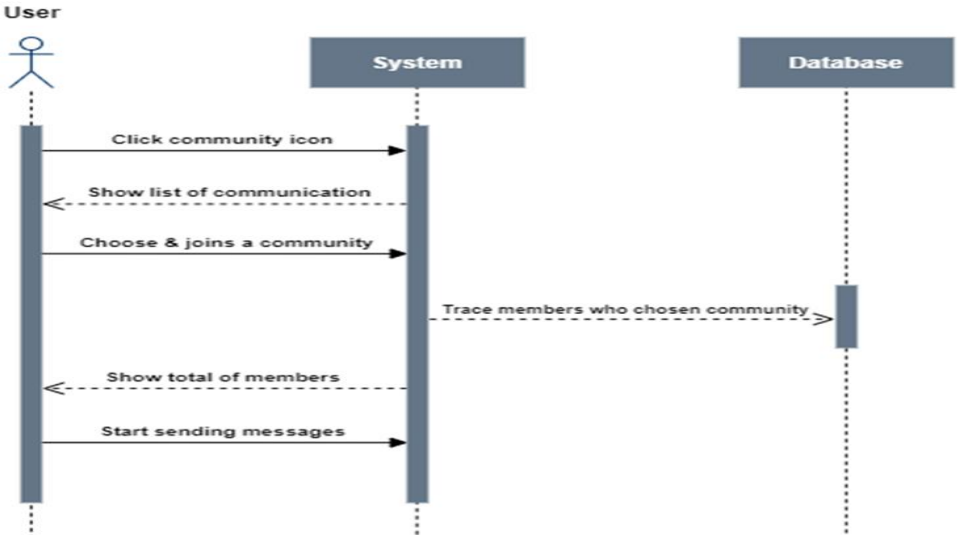


Figure 4.6 Community Sequence

4.5 Requirements tree diagram

A tree diagram is a new management planning tool that depicts the hierarchy of tasks and subtasks needed to complete an objective. The tree diagram starts with one item that branches into two or more, each of which branches into two or more, and so on. The finished diagram bears a resemblance to a tree, with a trunk and multiple branches

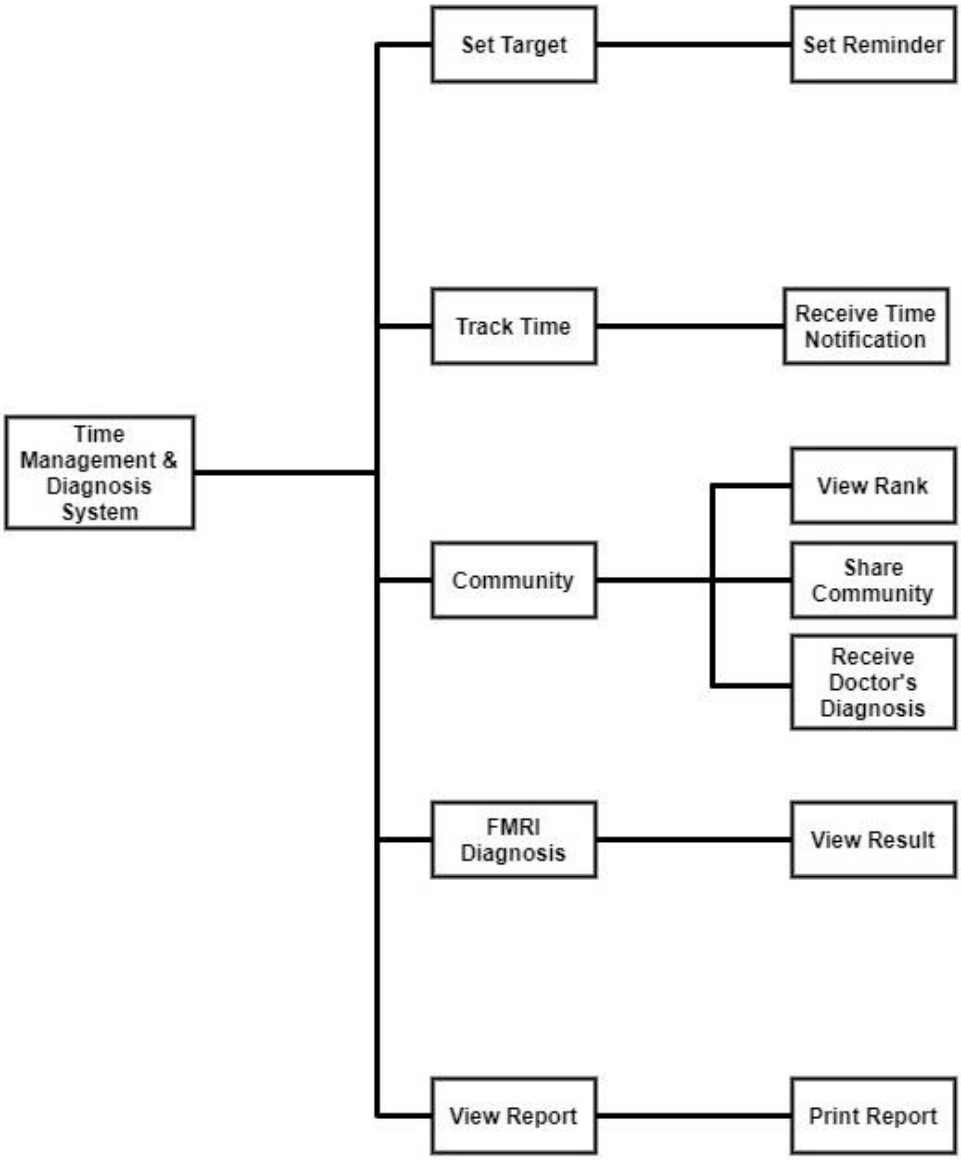


Figure 4.7 Requirements Tree

4.6 Swot Analysis

In today's dynamic business landscape, organizations require robust tools to assess their competitive advantage and inform strategic decision-making. Enter SWOT analysis, a powerful framework that stands for **Strengths, Weaknesses, Opportunities, and Threats**.

This widely used technique offers a structured approach to evaluating both internal and external factors impacting your business. By systematically identifying these elements, you gain a clear understanding of your current position and can develop effective strategies to capitalize on strengths, address weaknesses, exploit opportunities, and mitigate threats.

Figure 4.8 SWOT Analysis

4.7 ERD

ERD (Entity Relationship Diagram) is a data modeling technique that has been graphically illustrated. an information system's entities and the relationship between those entities. This ERD shows 2 main entities: (User, and neck rest) and the relationship between them. Each entity contains attributes with identifying. the foreign key and the primary key.

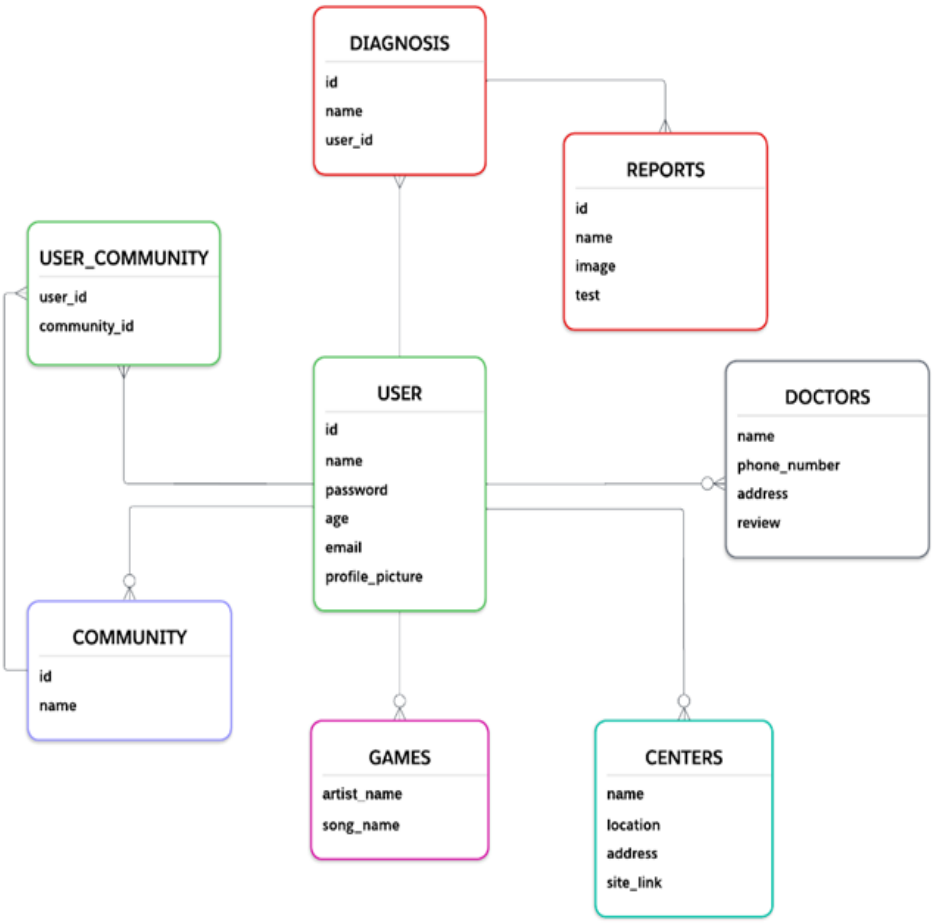


Figure 4.9 ERD diagram

4.8 Main Components in the System

In the realm of system design and architecture, understanding the main components that constitute a system is crucial for building robust and efficient solutions. These components serve as the building blocks, working in harmony to achieve the system's overarching goals and functionality. Each component plays a specific role, contributing to the system's overall performance and reliability.

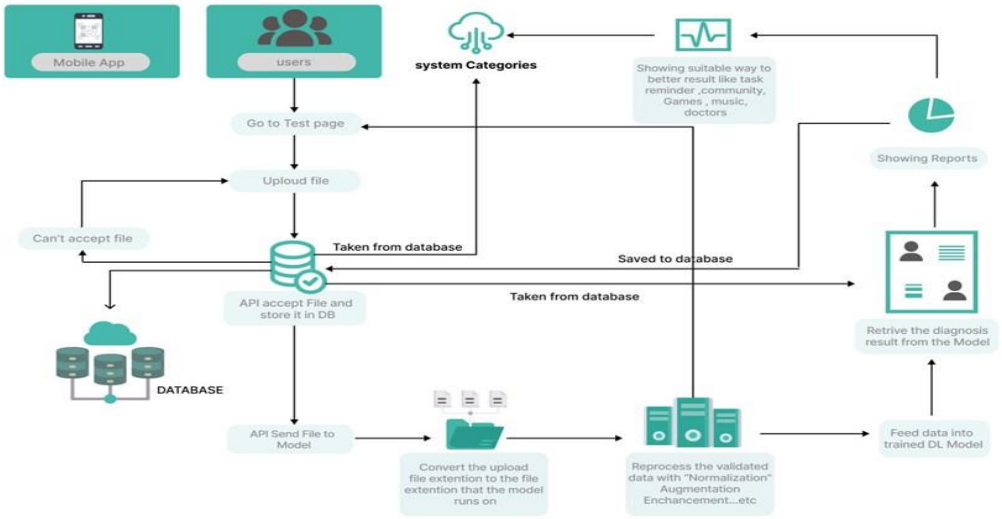


Figure 4.10 System Architecture

4.9 Business Model

A well-designed and robust business model serves as the foundation for sustainable growth and success in today's competitive marketplace. It outlines the strategic framework and operational structure that a company employs to create, deliver, and capture value. A professional business model integrates various elements, including customer segments, value propositions, channels, customer relationships, revenue streams, key activities, resources, and partnerships. By meticulously analyzing customer needs and market trends, a business model enables organizations to identify and target lucrative customer segments effectively. It delineates the unique value propositions that differentiate a company from its competitors, demonstrating how it solves customer problems or fulfills their desires. Furthermore, a business model delineates the most effective channels and customer relationships through which a company delivers its products or services to its target audience. It encompasses revenue streams, depicting the different ways a business generates income, such as product sales, subscriptions, licensing, or advertising. The key activities and resources required to operate the business are identified, along with the strategic partnerships that enhance capabilities or extend reach. A professional business model is dynamic, adaptable, and aligned with the company's overall vision, enabling it to navigate changing market conditions and seize new opportunities. By leveraging a well-crafted business model, organizations can optimize their operations, drive innovation, and foster long-term profitability while consistently delivering value to their stakeholders.

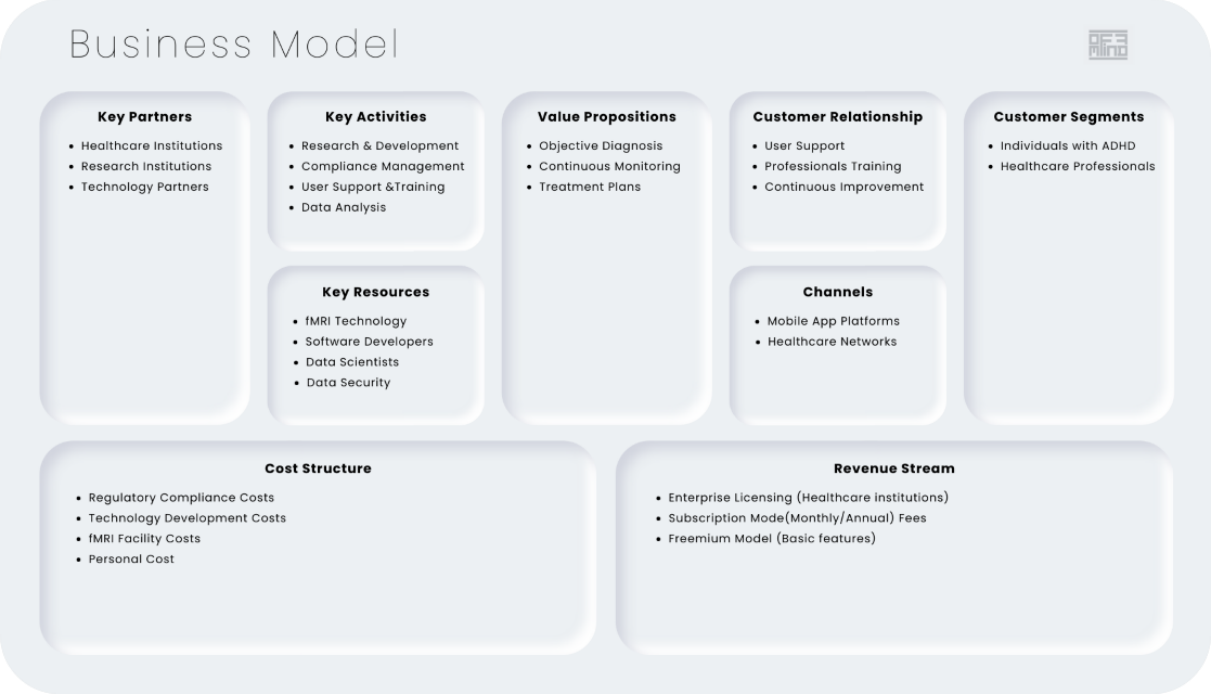


Figure 4.11 Business Model

- **Value Proposition**: Defines the unique value or benefit that sets the business apart from competitors.
- **Customer Segments**: Identifies and understands the specific target audience for tailored products and experiences.
- **Channels**: Encompasses the ways the business reaches and interacts with customers to deliver value.
- **Customer Relationships**: Focuses on building and maintaining customer connections throughout their journey.
- **Revenue Streams**: Outlines income sources, pricing strategy, and revenue models.
- **Key Activities**: Essential tasks required to deliver the value proposition effectively.
- **Key Resources**: Assets and capabilities necessary for business operations.
- **Key Partnerships**: Collaborations with external entities to strengthen operations and access resources.
- **Cost Structure**: Breakdown of all operational costs incurred by the business.

By considering and refining each of these elements, businesses can develop a comprehensive and effective business model aligned with their goals for greater success.

Chapter 5

System Design

5.1 Introduction

This chapter delving into the heart of our mobile application – its system design. We'll embark on a comprehensive exploration of the technological architecture that underpins the app's functionality

Furthermore, we'll meticulously outline the design process, detailing the steps and procedures followed during the system's development. We'll describe the methodology employed, highlighting any modifications or refinements made to the initial design based on practical considerations. Additionally, we'll provide insights into the rationale behind specific design choices, considering factors like scalability, performance, and security.

Through this chapter, we aim to demonstrate the innovative design choices that underpin our project. By combining these elements, we've achieved a robust and efficient system that effectively supports our project's goals.

5.2 User Interface Design

User interface elements

User interface elements include but are not limited to:

- Input controls
- Navigational components
- Informational components
- Containers

5.2.1 OFFMIND'S Logo

Figure 5.1 OFFMIND Logo

5.2.2 OFFMIND'S Layout

Mobile application

- **Splash screen:** OFFMIND'S logo will appear after opening the app for a few seconds then start with the application.

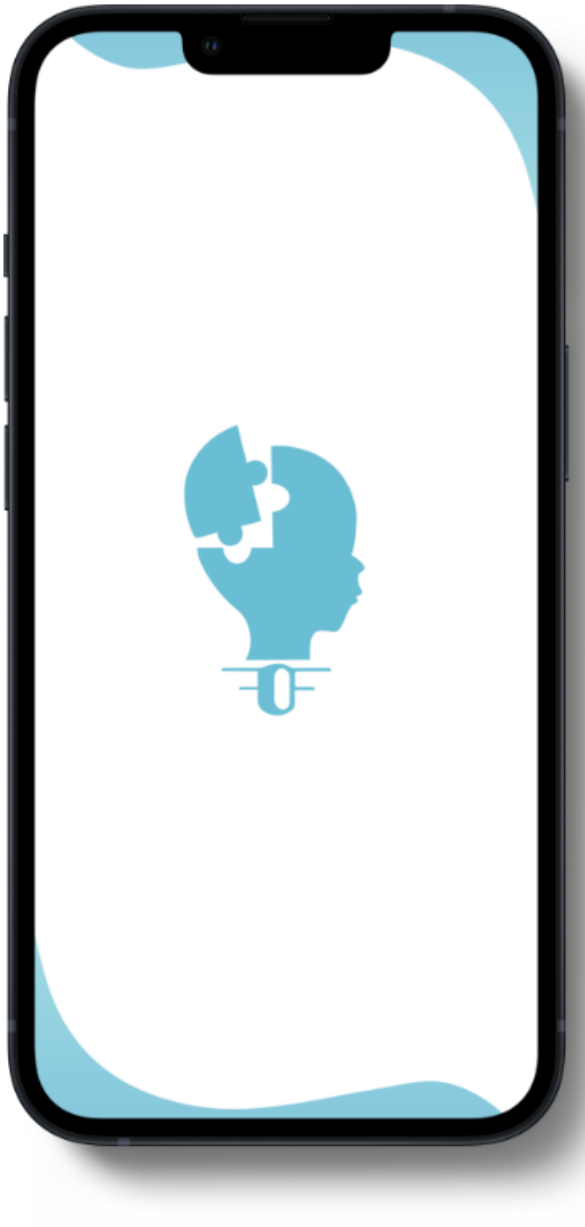


Figure 5.2 Splash screen

It’s information for the user to explain the app’s goals and services that can else do like

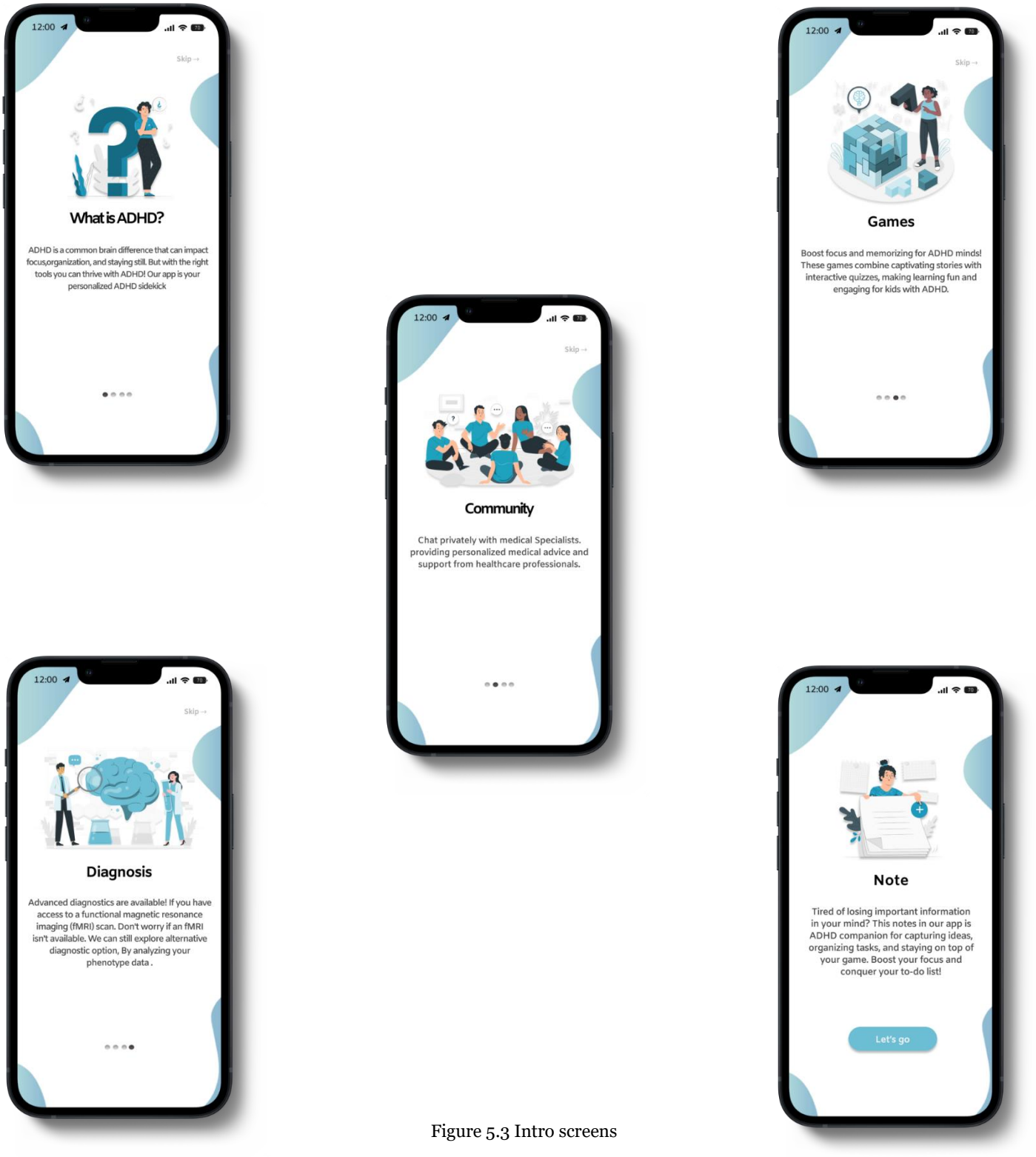


Figure 5.3 Intro screens

- **Sign in or sign-up page:** After the guide screens user will choose to sign up or sign in if he has already an account.

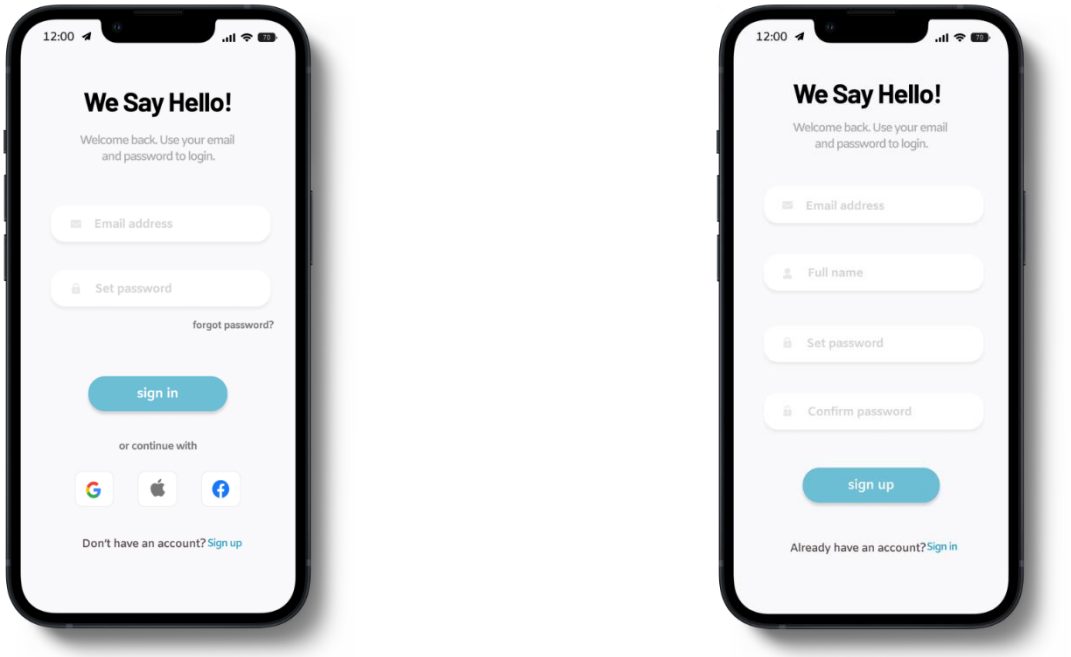


Figure 5.4 Sign in/up screens

- **Sign in page:** If user has account, he can sign in by entering email and password. If he forgot the password, he could easily change password by link in email.

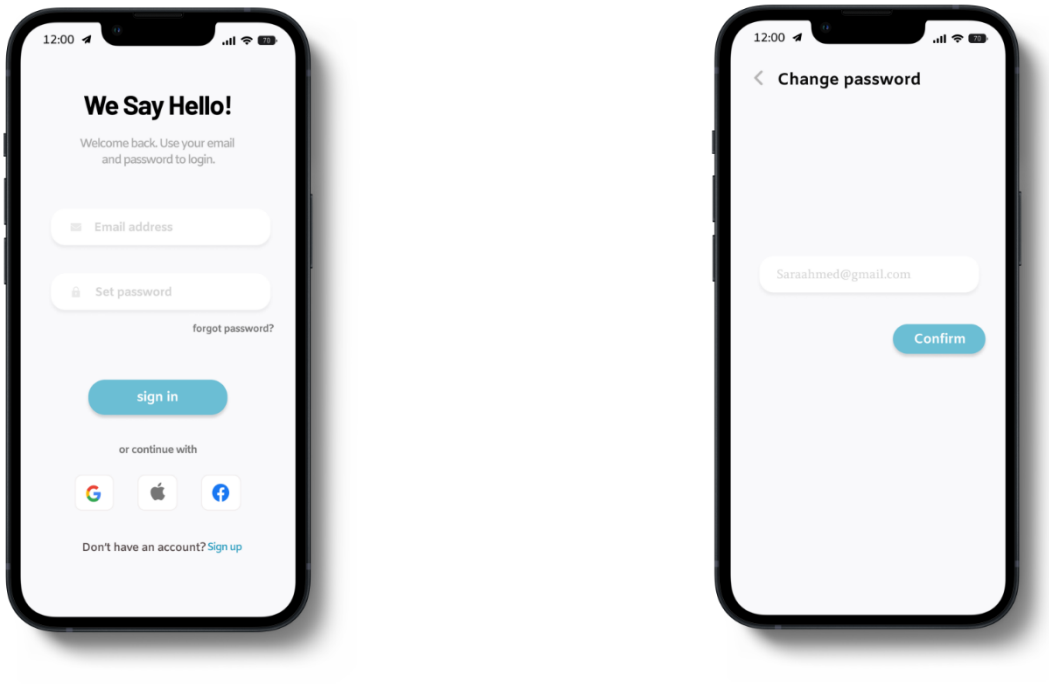


Figure 5.5 Sign in screens

- **Home page:** we will display our feature notes, doctors, centers, and specialists

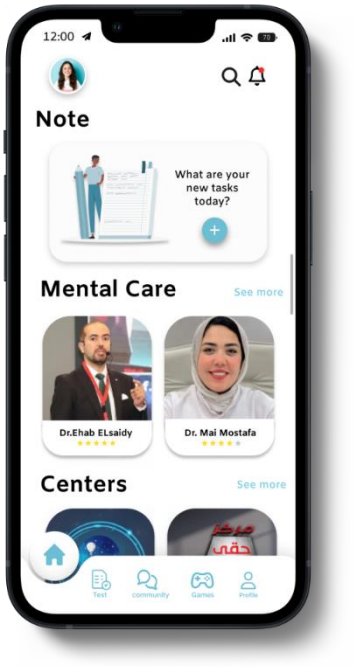


Figure 5.6 Home Screen

- **Note page:** The These Screens contain notes for the tasks that the child will do and that he has finished in an organized form, with the ability to edit, delete, and pin them, and there is a reminder feature if he forgets to perform the tasks in the note.

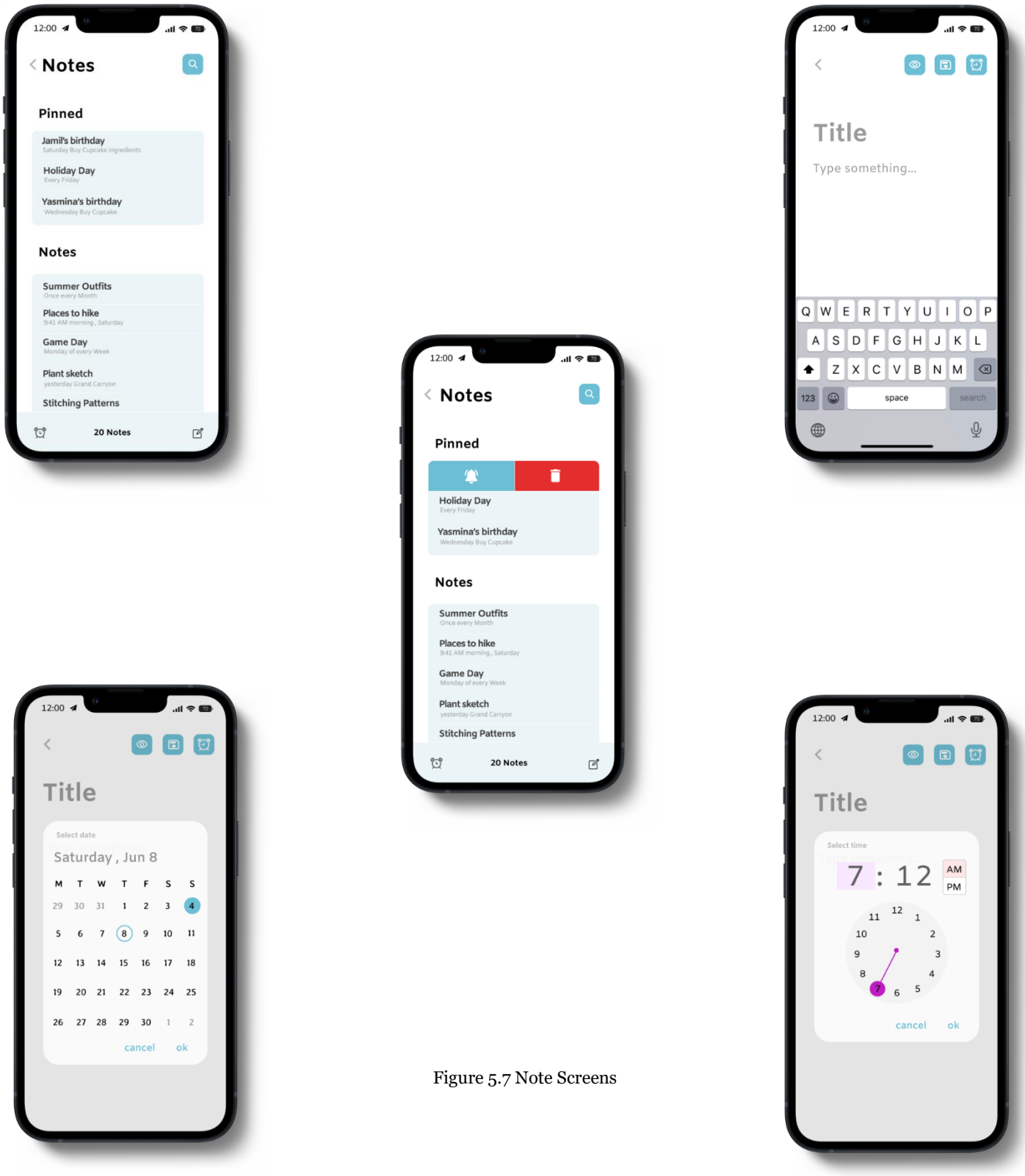


Figure 5.7 Note Screens

• **Doctors page:** The screens provide detailed information about individual doctors. Users can search for a doctor by entering the name or title in the search bar, tap on the doctor's name to learn more about it.

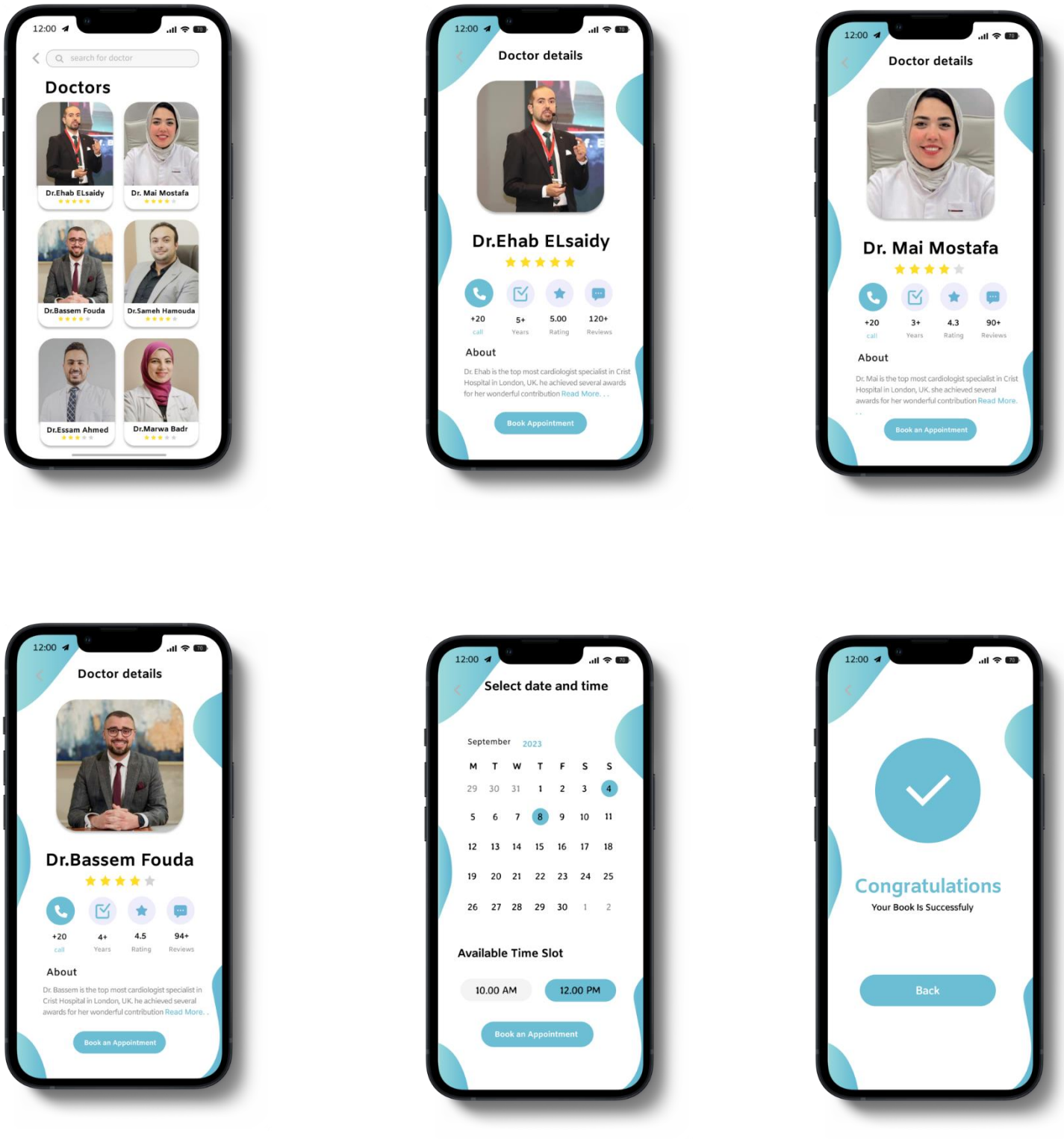


Figure 5.8 Doctors Screens

• **Centers page:** The screens display list of medical centers, user can search for a center by entering the name or location in the search bar, tap on the center's name to learn more about it, each center represented by its logo, name, contact information, including phone number, email address, and website.

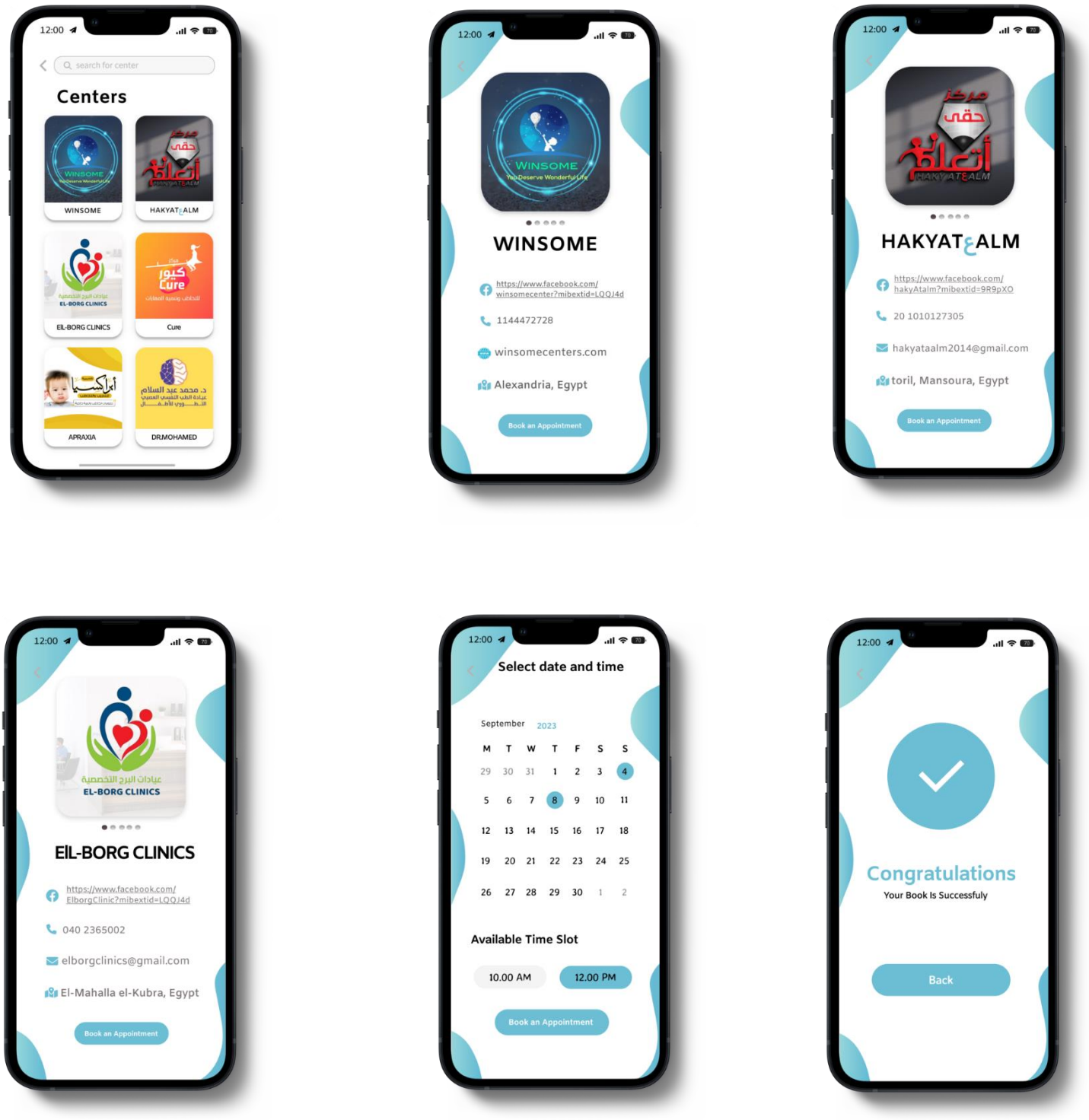


Figure 5.9 Centers Screens

• **Specialists page:** The screens provide detailed information about individual specialists. Users can search for a specialist by entering the name or title in the search bar, tap on the specialist's name to learn more about it.

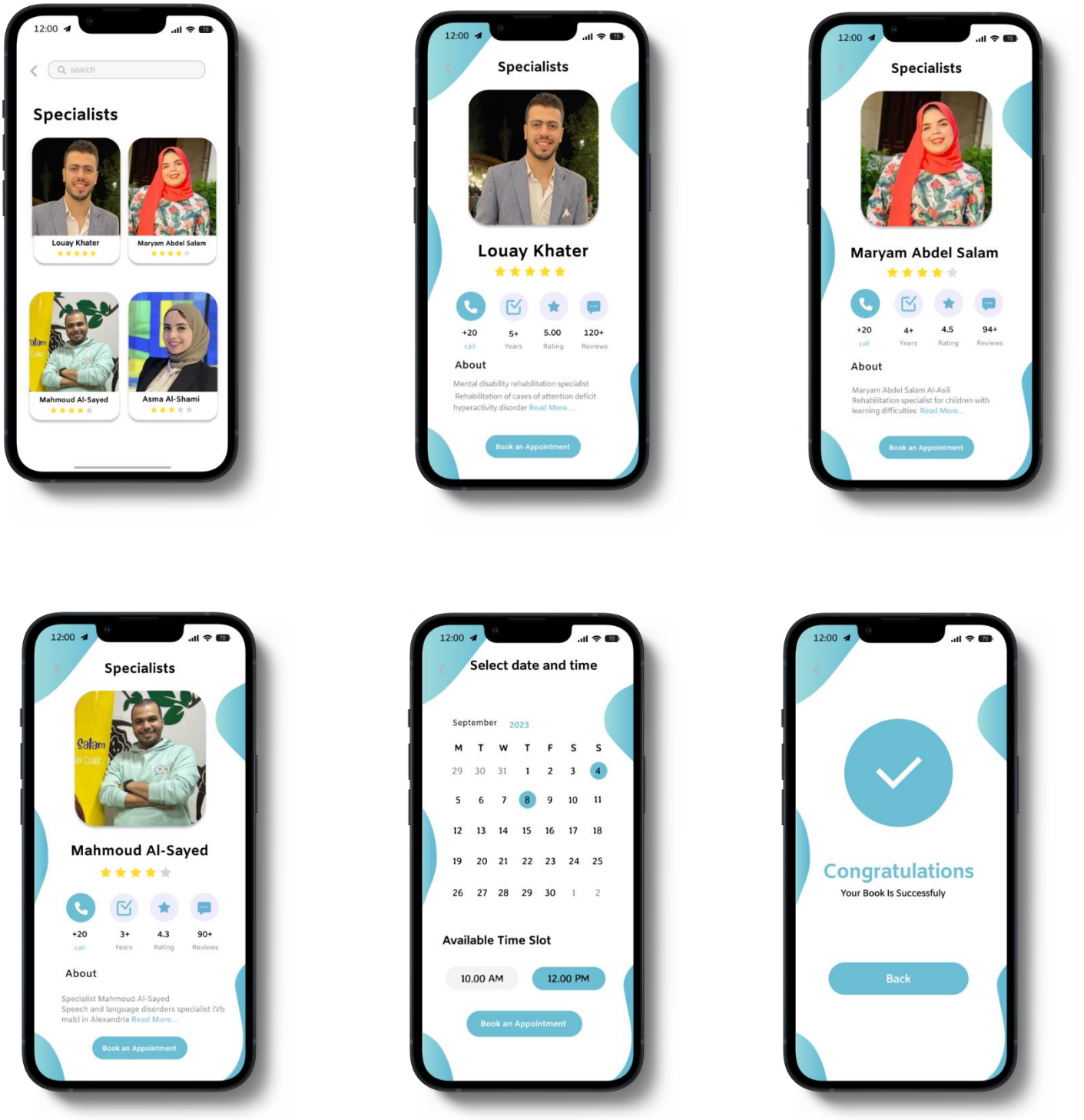


Figure 5.10 Specialists Screens

• **Test Page:** The main page through which the patient can enter his FMRI image through two methods, one is writing data and the second is uploading, and there is a specific page for each of them that explains to him the progress that occur until the diagnosis result is shown to him.

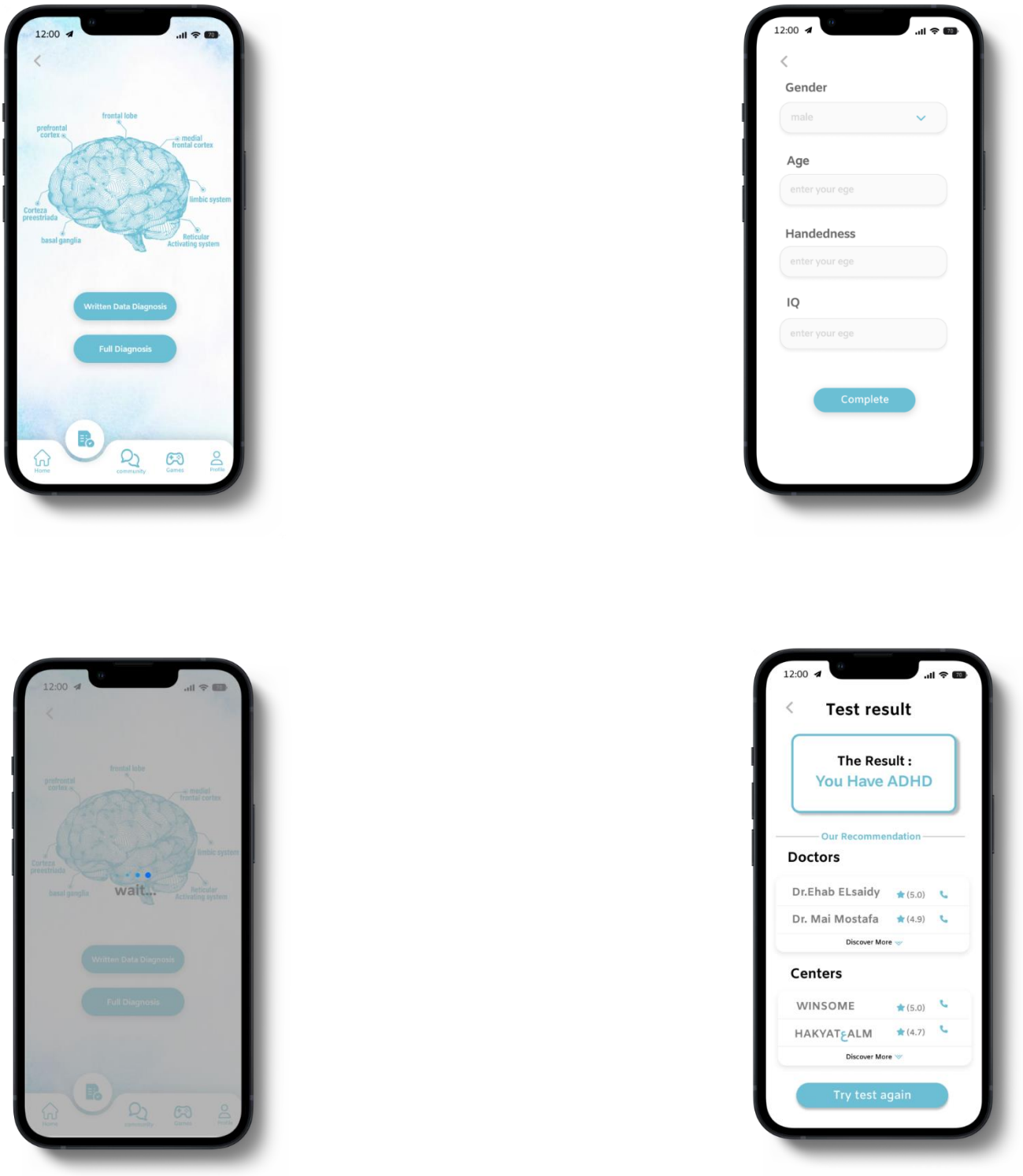


Figure 5.11 Test Screens

- **Test Page:** The main page through which the patient can enter his FMRI image through two methods, one is writing data and the second is uploading, and there is a specific page for each of them that explains to him the progress that occur until the diagnosis result is shown to him.

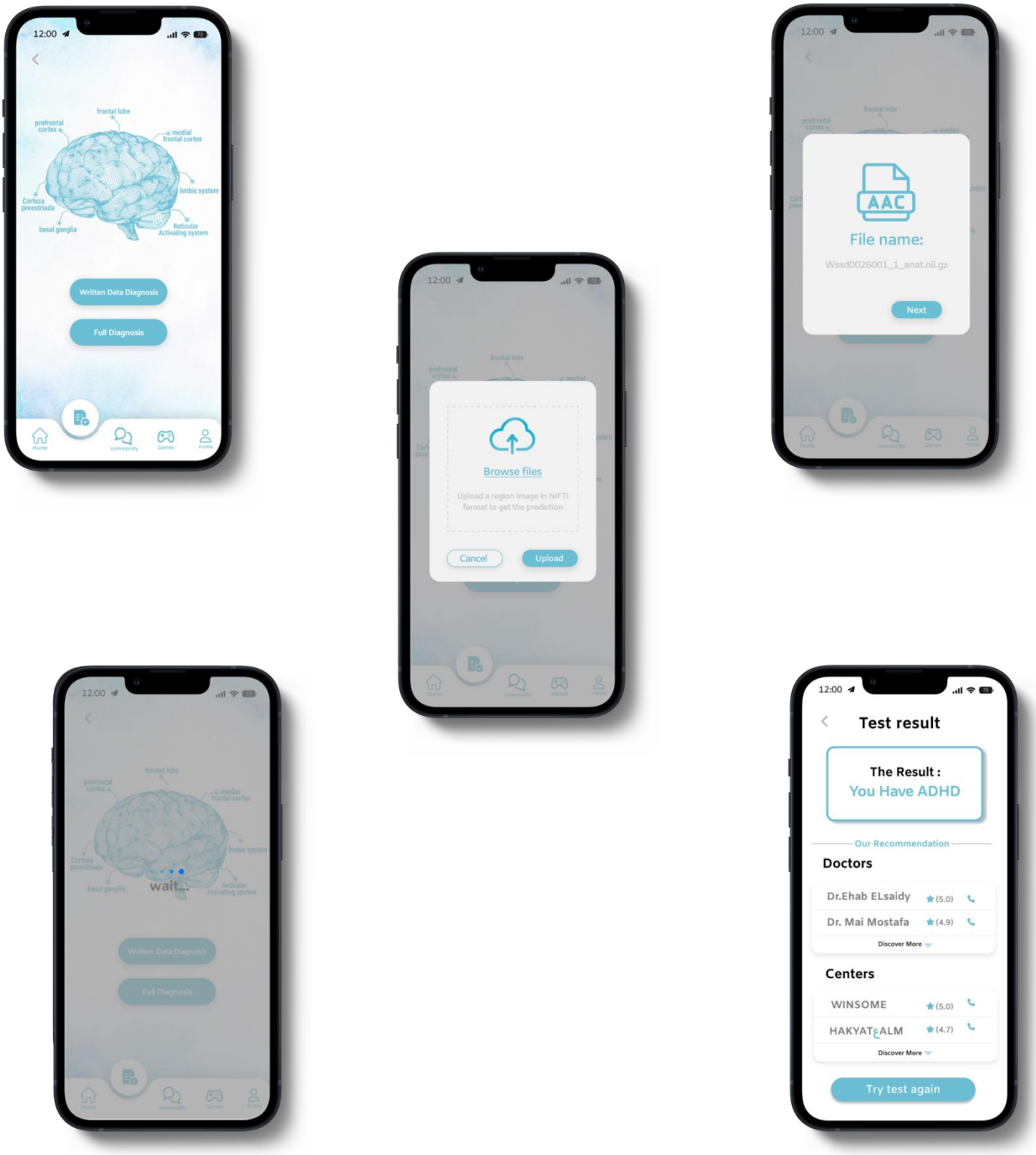


Figure 5.12 Test screens cont.

- **Games Page:** On this page, we have provided a group of games to help ADHD patients avoid distraction and lack of attention

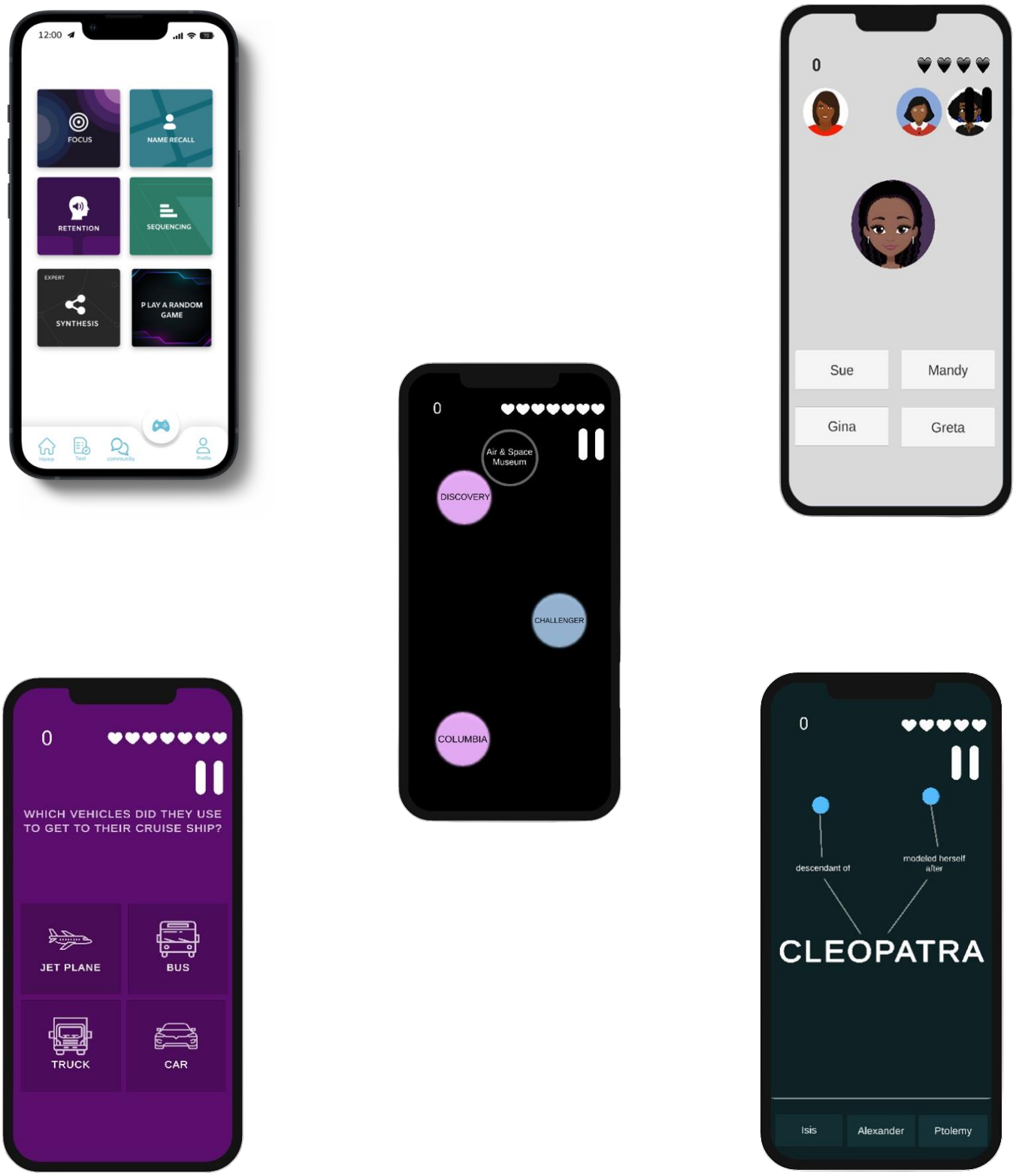


Figure 5.13 Games screens

- **Community Page:** Display a list of "Online doctors." and show an ongoing chat with doctors, the chat includes sent and received messages, and there's an option to type a new message at the bottom and record voice note, users can consult with doctors online, making healthcare more accessible and convenient, it's a great tool for connecting patients with healthcare professionals

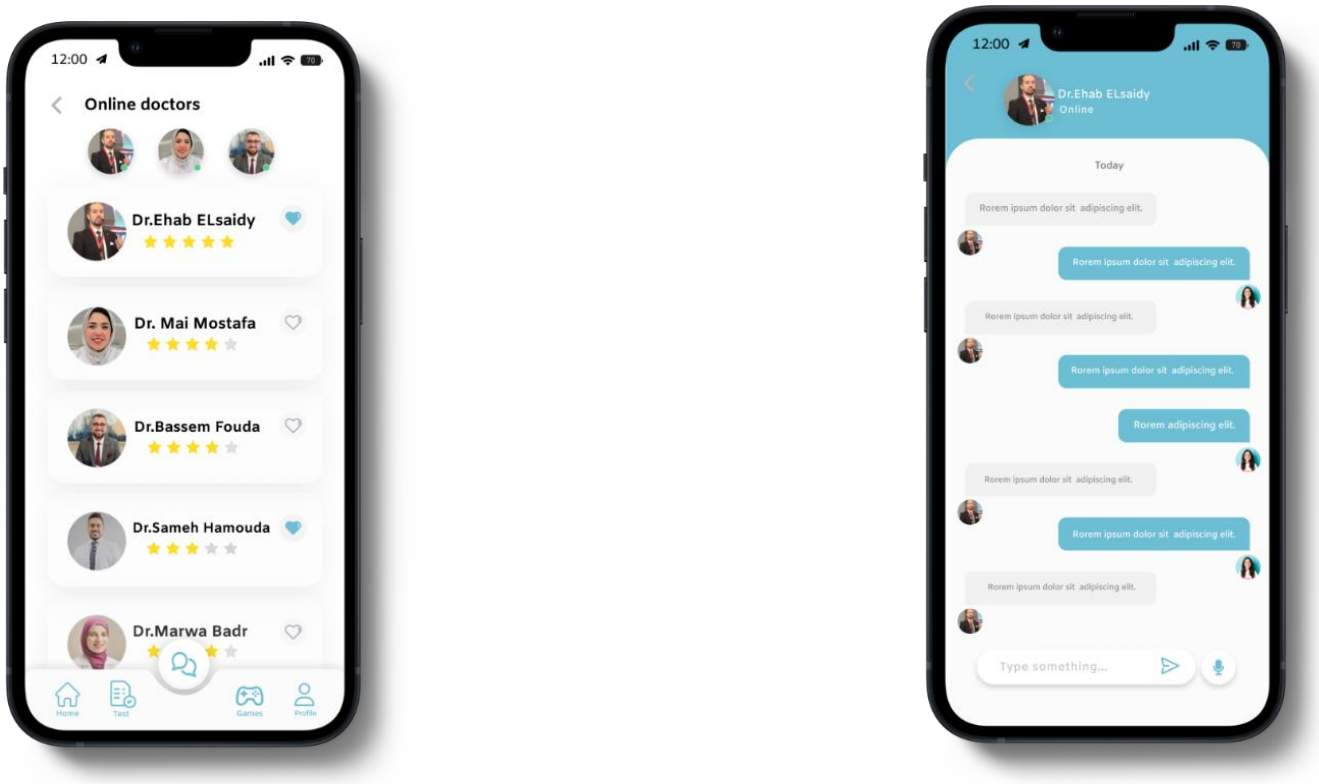


Figure 5.14 Community screens

- **Profile Page:** On the profile page, a set of previous tests, deleting the account, and modifying the password are provided

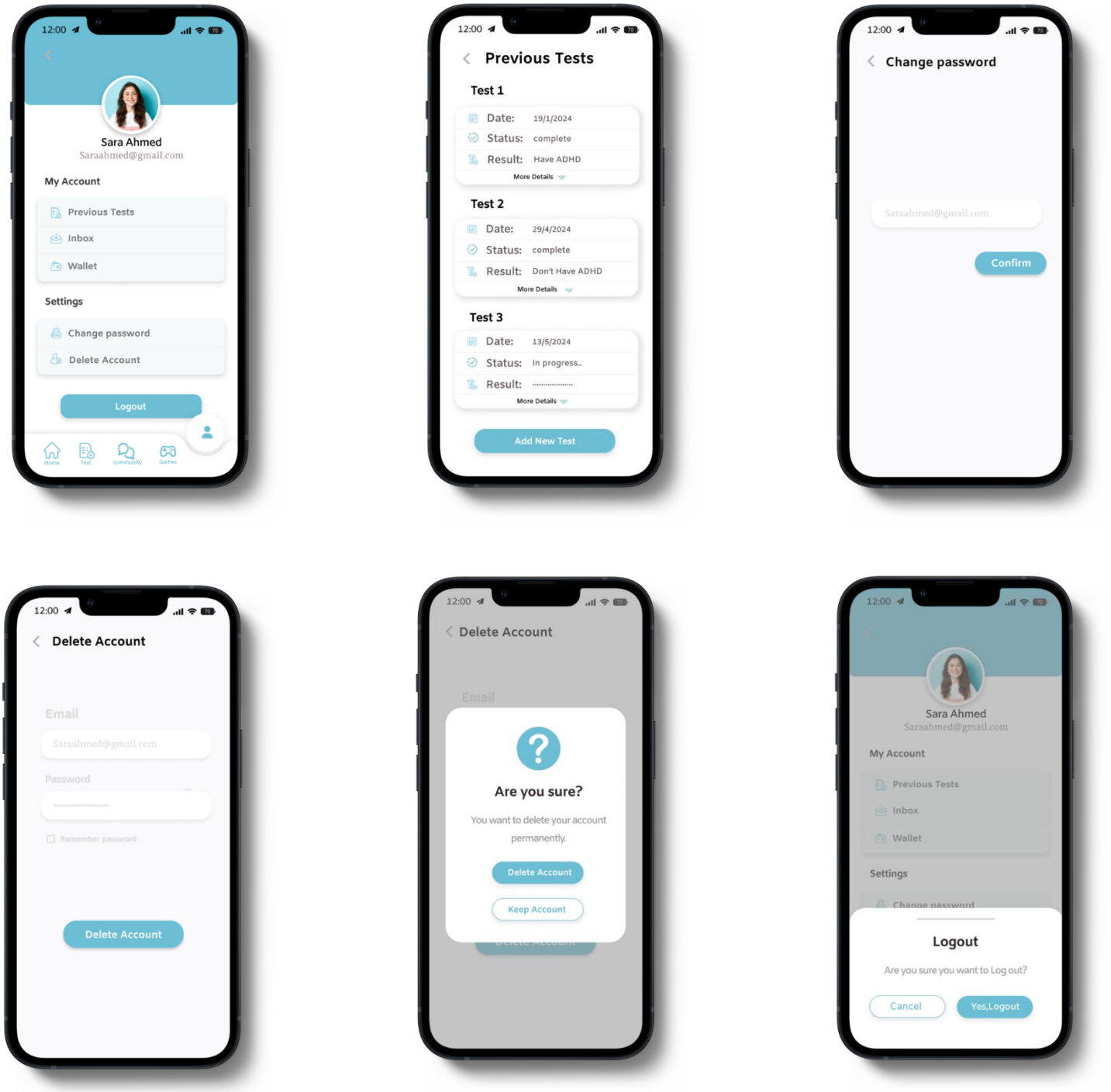


Figure 5.15 Profile screens

Chapter 6

Implementation

6.2 Introduction

The successful implementation of our project, aimed at revolutionizing the approach to addressing the needs of children with ADHD, relies on carefully selecting an optimal implementation platform and meticulous attention to implementation details. Our chosen implementation platform encompasses hardware, software, and infrastructure, providing the foundation for our vision.

During the implementation process, we followed a well-defined methodology encompassing stages such as requirement analysis, system design, development, testing, and deployment. Each stage involved meticulous planning and attention to detail, ensuring a structured and organized approach.

Data collection played a crucial role in our project, and we employed methodologies to gather relevant data ethically and responsibly. We also established an experimental setup to validate the effectiveness of our project, sharing insights into hardware and software configurations, experimental protocols, and user testing procedures.

To streamline the implementation process and adapt to evolving requirements, we employed software development methodologies such as agile or iterative approaches. These methodologies fostered collaboration, flexibility, and responsiveness to feedback, enhancing the overall effectiveness of our project.

Furthermore, we integrated external systems, APIs, or services to extend our system's capabilities, enabling seamless communication, data exchange, and access to additional functionalities. These integrations required technical expertise and contributed to the success of our project.

Throughout the implementation journey, we encountered challenges and obstacles. However, our ability to adapt, problem-solve, and persevere allowed us to overcome these complexities and achieve the desired outcomes.

By providing a comprehensive overview of our implementation platform and details, we offer transparency and rigor, allowing for critical evaluation, replication, and further exploration by the scientific and engineering community.

6.2 Machine learning

Data Gathering:

- We work on **ADHD 200** dataset downloaded from http://fcon_1000.projects.nitrc.org/indi/adhd200/index.html

- Comprising data from 973 resting-state fMRI was scanned across eight different sites (PekingU, BrownU, KKI, NeuroIMAGE, NYU, OHSU, Upitt, WashU). Accompanying phenotypic information includes diagnostic status, dimensional ADHD symptom measures, age, sex, intelligence quotient (IQ) and lifetime medication status. Preliminary quality control assessments (usable vs. questionable) based upon visual timeseries inspection are included for all resting state fMRI scans.

- The ADHD-200 "Training Dataset" excluded participants with questionable fMRI scan quality, consisting of 668 participants. The "Original Training Dataset" included 776 participants.
- The "Holdout Dataset" consisted of 197 participants for whom diagnostic data were released except for the 26 participants from the BrownU site.

- The code below visualizes a sample of slices of the fMRI can be taken in dataset:

```
import nibabel as nib
import matplotlib.pyplot as plt
from nilearn import plotting

brain_vol = nib.load('Brown/0026001/wmean_mrda0026001_session_1_rest_1.nii.gz')
brain_vol_data = brain_vol.get_fdata()
type(brain_vol_data)
fig_rows = 4
fig_cols = 4
n_subplots = fig_rows * fig_cols
n_slice = brain_vol_data.shape[0]
step_size = n_slice // n_subplots
plot_range = n_subplots * step_size
start_stop = int((n_slice - plot_range) / 2)

fig, axs = plt.subplots(fig_rows, fig_cols, figsize=[10,10])

for idx, img in enumerate(range(start_stop, plot_range, step_size)):
    axs.flat[idx].imshow(ndi.rotate(brain_vol_data[img, :, :], 90), cmap='gray')
    axs.flat[idx].axis('off')

plt.tight_layout()
plt.show()
```

Figure 6.1 Visualization

Output:

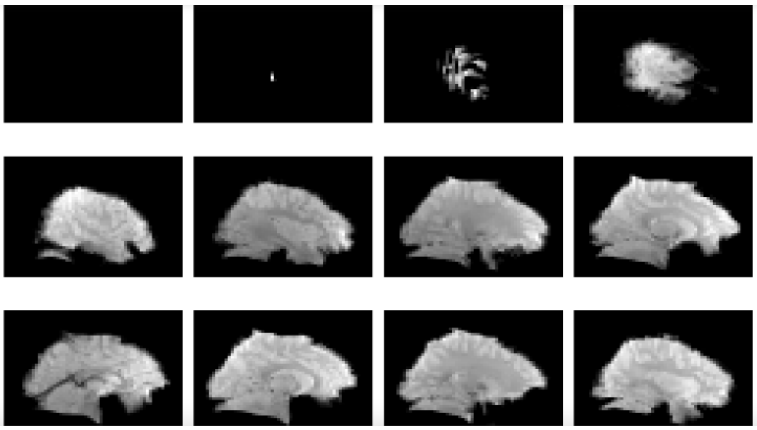


Figure6.2 Output

Data Preprocessing:

Find the Right Atlas

Atlas: in the context of neuroimaging and brain research, an atlas is a standardized reference that divides the brain into distinct regions or areas, each assigned specific coordinates and labels. These atlases are used to map the anatomical structures of the brain and facilitate consistent identification and analysis of brain regions across different studies and datasets.

Benefits:

1. Helps in accurately locating specific brain regions and understanding their functions.
2. Enables researchers to compare findings and results across different studies by referring to the same brain regions defined in the atlas.

This code was created to find the atlas with matching dimensions for the file that will be used for analysis Atlases were accessed from :
website:https://github.com/neurodata/neuroparc/tree/master/atlases/label/Human Files with size 4x4x4 mm

Sample file

This file represents a preprocessed fMRI for patient 0026001

```
]:
import nibabel as nib
import matplotlib.pyplot as plt

img = nib.load('Brown/0026001/sfnwmrda0026001_session_1_rest_1.nii.gz')
img_data = img.get_fdata()

img_data.shape
```

```
]: (49, 58, 47, 247)
```

Figure6.3 sample file

Load Atlases

Many types of the atlases will be loaded and have their dimensions checked using the shape attribute. As discussed previously, the shape to match is (49, 58, 47)

```
AAL_atlas = nib.load('Atlases/AAL_space-MNI152NLin6_res-4x4x4.nii.gz')
AAL_atlas_data = AAL_atlas.get_fdata()

AAL_atlas_data.shape
```

```
(45, 54, 45)
```

```
AICHA_atlas = nib.load('Atlases/AICHAJoliot2015_space-MNI152NLin6_res-4x4x4.nii.gz')
AICHA_atlas_data = AICHA_atlas.get_fdata()

AICHA_atlas_data.shape
```

```
(45, 54, 45)
```

```
Brodmann_atlas = nib.load('Atlases/Brodmann_space-MNI152NLin6_res-4x4x4.nii.gz')
Brodmann_atlas_data = Brodmann_atlas.get_fdata()

Brodmann_atlas_data.shape
```

```
(45, 54, 45)
```

Figure 6.4 atlases

This atlas works! This is what will be applied to the fMRI files.

Within the ADHD 200 download for the Athena pipeline, there is an aal template that could be the correct atlas" AAL Time Courses Corrected Filtering"

```
ADHD_AAL_atlas = nib.load('ADHD200_AAL_TCs_filtfix/templates/aal_mask_pad.nii.gz')
ADHD_AAL_atlas_data = ADHD_AAL_atlas.get_fdata()

ADHD_AAL_atlas_data.shape
```

```
(49, 58, 47)
```

Figure 6.5 ADHD_AAL_atlas

Convert image data into dataframe:

create an efficient way to turn image data into a vector and turn a vector into a region by time dataframe.

Results:

A pandas dataframe with rows representing the voxels of the fMRI file and features representing the region assigned by the atlas and raw data for each time point

Contents

1. Load Data: Loading the fMRI file and atlas
2. Generate Dataframe: Creating the desired dataframe
   a. Time data: Adding the raw data based on the time parameter of the fRMI file
   b. Atlas data: Adding the atlas data to the respective voxels

image_to_vector()

Create a vector from a region by time matrix from a image using the atlas

Input:

- Data for the image to take points of
- Data from the atlas to apply to the image data

Output: A vector of the image's region by time matrix

```
def image_to_vector(image_data, atlas_data):
    '''
    Create a vector from a region by time matrix from a image using the atlas

    Input:
        - Data for the image to take points of
        - Data from the atlas to apply to the image data

    Output: A vector of the image's region by time matrix
    '''
    # Names for columns and index
    column_names = ['time_' + str(i) for i in range(247)]
    region_names = ['region_' + str(region) for region in np.unique(atlas_data.reshape(-1))]

    # Dataframe containing image data with correct column names
    df_times = pd.DataFrame(img_data.reshape(-1, 247), columns = column_names)

    # Dataframe containing image with addition of atlas region
    df_full = pd.concat([pd.Series(atlas_data.reshape(-1)), df_times], axis=1)
    df_full = df_full.rename(columns={0:'atlas_region'})

    # Dataframe of region vs. time
    regions_x_time = df_full.groupby('atlas_region').mean()
    regions_x_time.index = region_names

    # Return vector of region x time dataframe
    regions_x_time_vector = regions_x_time.to_numpy().reshape(-1)
    return regions_x_time_vector
```

Figure 6.6 Image to vector

vector_to_dataframe()

Create a region by time dataframe from a vector

Input: A vector containing the image data (condensed region by time dataframe)

Output: A region by time dataframe

```
def vector_to_dataframe(image_vector):
    '''
    Create a region by time dataframe from a vector

    Input: A vector containing the image data (condensed region by time dataframe)

    Output: A region by time dataframe
    '''
    # Names for columns and index
    column_names = ['time_' + str(i) for i in range(247)]
    region_names = ['region_' + str(region) for region in np.unique(atlas_data.reshape(-1))]

    # Turn vector into dataframe
    image_array = image_vector.reshape(117,247)
    df_vector = pd.DataFrame(data=image_array)

    # Change columns and index to be more clear
    df_vector.columns = column_names
    df_vector.index = region_names

    return df_vector
```

Figure 6.7 Vector to dataframe

Output:

	time_0	time_1	time_2	time_3	time_4	time_5	time_6	time_
region_0.0	-0.010099	0.013498	0.035757	0.045200	0.039266	0.024453	0.009799	-0.00022
region_2001.0	1.692365	1.563411	0.747202	-0.273904	-0.974473	-1.128608	-0.871923	-0.49829
region_2002.0	0.659177	0.639452	0.420765	0.308812	0.456353	0.720466	0.817896	0.62379
region_2101.0	0.151973	0.418374	1.303733	2.337760	2.914812	2.743485	2.029023	1.24938
region_2102.0	0.143743	0.354073	0.600906	0.815291	0.900404	0.797972	0.532047	0.19548

5 rows × 247 columns

Figure 6.8 region by time of the image

Condense Region Dataframe:

A Process to streamline a comprehensive dataset of average region intensities from multiple subjects into a concise, meaningful format. This condensed dataset highlights regions most correlated with the diagnosis, thereby aiding in more efficient and focused analysis.

Purpose

The original dataset contains average region intensity values for numerous subjects collected from various sites. These values represent different brain regions' activity levels. By condensing this dataset, we aim to identify and retain only the regions that significantly correlate with diagnostic outcomes. This reduces the complexity of the dataset and enhances the effectiveness of subsequent analyses, such as predictive modeling and diagnostic research.

extract_features()

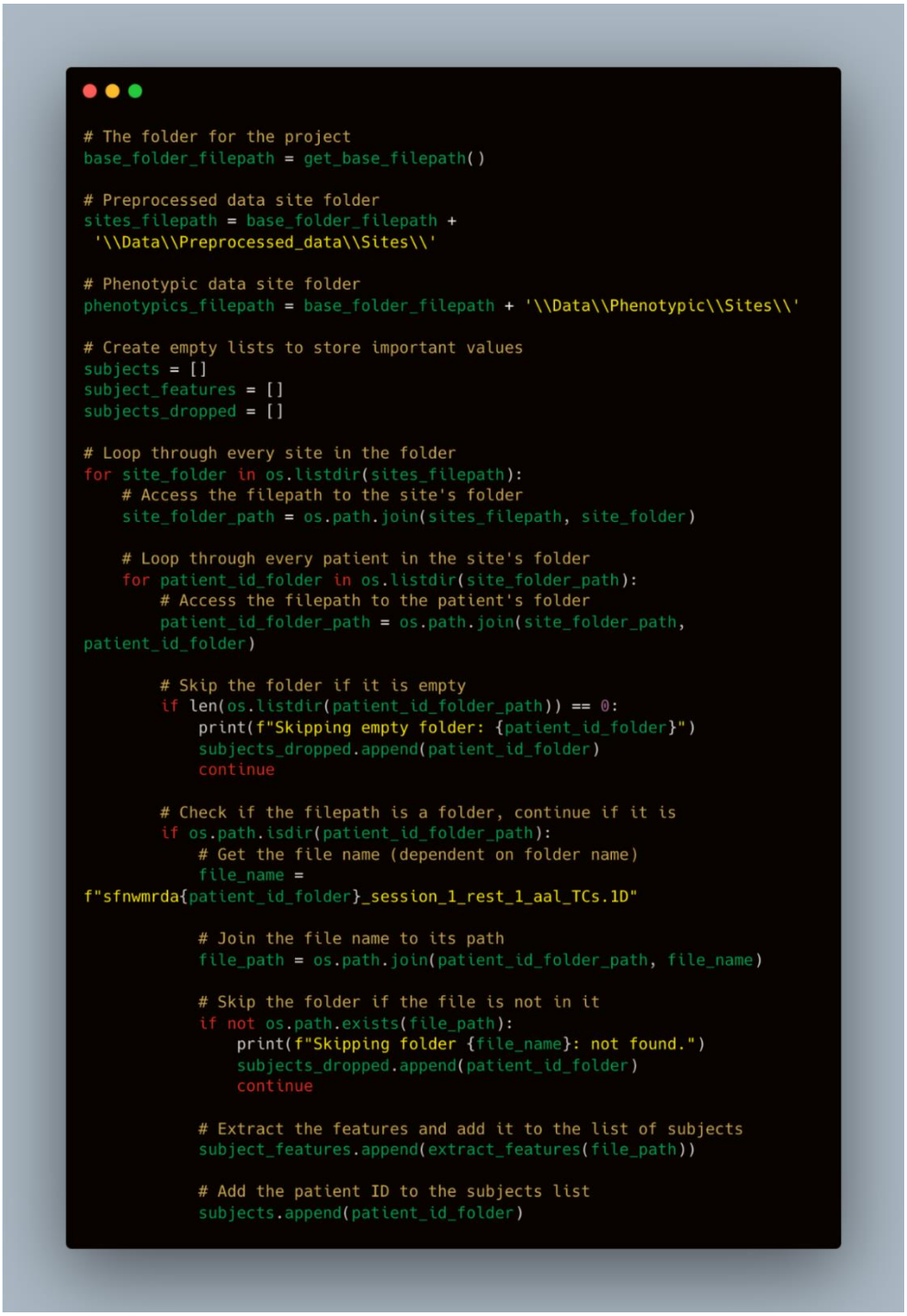
- Analyzing data from numerous subjects helps identify consistent patterns and features that are strongly correlated with the diagnosis, which might not be apparent from a single subject's data.
- **Read the data files for each subject, calculate the mean intensities for each region, and compile these into a single dataframe.**

```
def extract_features(filepath):
    '''
    Create a dataframe using the mean of regions over time.

    Input: filepath to open the dataframe

    Output: dataframe of mean for each region
    '''
    # Read the filepath as a dataframe (use 1 tab as separator and the first
line as the header)
    df = pd.read_csv(filepath, sep=r'\s{1,}', engine='python', header=0)

    # Drop two features that get in the way of evaluation
    df = df.drop('File', axis=1)
    df = df.drop('Sub-brick', axis=1)

    # Return the mean for each of the features (method of vectorizing)
    return df.mean()
```

Figure 6.9 mean intensities

Subjects

- Open the 'sfnwmrda' file for each subject in the study.
- Add the features to a matrix and the subjects to a different matrix.
- There may be instances where the subject does not have the file in their folder. In this case, add the subject to a matrix to be dropped from the phenotypic dataframe later.



```
# The folder for the project
base_folder_filepath = get_base_filepath()

# Preprocessed data site folder
sites_filepath = base_folder_filepath +
 '\\Data\\Preprocessed_data\\Sites\\'

# Phenotypic data site folder
phenotypics_filepath = base_folder_filepath + '\\Data\\Phenotypic\\Sites\\'

# Create empty lists to store important values
subjects = []
subject_features = []
subjects_dropped = []

# Loop through every site in the folder
for site_folder in os.listdir(sites_filepath):
    # Access the filepath to the site's folder
    site_folder_path = os.path.join(sites_filepath, site_folder)

    # Loop through every patient in the site's folder
    for patient_id_folder in os.listdir(site_folder_path):
        # Access the filepath to the patient's folder
        patient_id_folder_path = os.path.join(site_folder_path,
patient_id_folder)

        # Skip the folder if it is empty
        if len(os.listdir(patient_id_folder_path)) == 0:
            print(f"Skipping empty folder: {patient_id_folder}")
            subjects_dropped.append(patient_id_folder)
            continue

        # Check if the filepath is a folder, continue if it is
        if os.path.isdir(patient_id_folder_path):
            # Get the file name (dependent on folder name)
            file_name =
f"sfnwmrda{patient_id_folder}_session_1_rest_1_aal_TCs.1D"

            # Join the file name to its path
            file_path = os.path.join(patient_id_folder_path, file_name)

            # Skip the folder if the file is not in it
            if not os.path.exists(file_path):
                print(f"Skipping folder {file_name}: not found.")
                subjects_dropped.append(patient_id_folder)
                continue

            # Extract the features and add it to the list of subjects
            subject_features.append(extract_features(file_path))

            # Add the patient ID to the subjects list
            subjects.append(patient_id_folder)
```

Figure 6.10 features from multiple sub

Correlation Analysis:

Calculate the correlation between each region's mean intensity and the diagnosis to identify the most significant features. Determine what features are most correlated to the diagnosis.

```
correlations = df_region_w_dx.drop('DX',
axis=1).corrwith(df_region_w_dx['DX'])

correlation_features = correlations.loc[abs(correlations) >= 0.05]
correlation_features.count()

X = df_region_w_dx.drop('DX', axis=1)
y = df_region_w_dx['DX']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

correlations_train = X_train.corrwith(y_train)
correlation_features_train = correlations_train.loc[abs(correlations_train)
>= 0.05]
correlation_features_train.count()

df_correlation_features = df_region_w_dx[correlation_features.index]
df_correlation_features['DX'] = df_region_w_dx['DX']

df_correlation_features_train =
df_region_w_dx[correlation_features_train.index]
df_correlation_features_train['DX'] = df_region_w_dx['DX']
```

Figure 6.11 features are most correlate

Cleaning Phenotypic Data for All Subjects

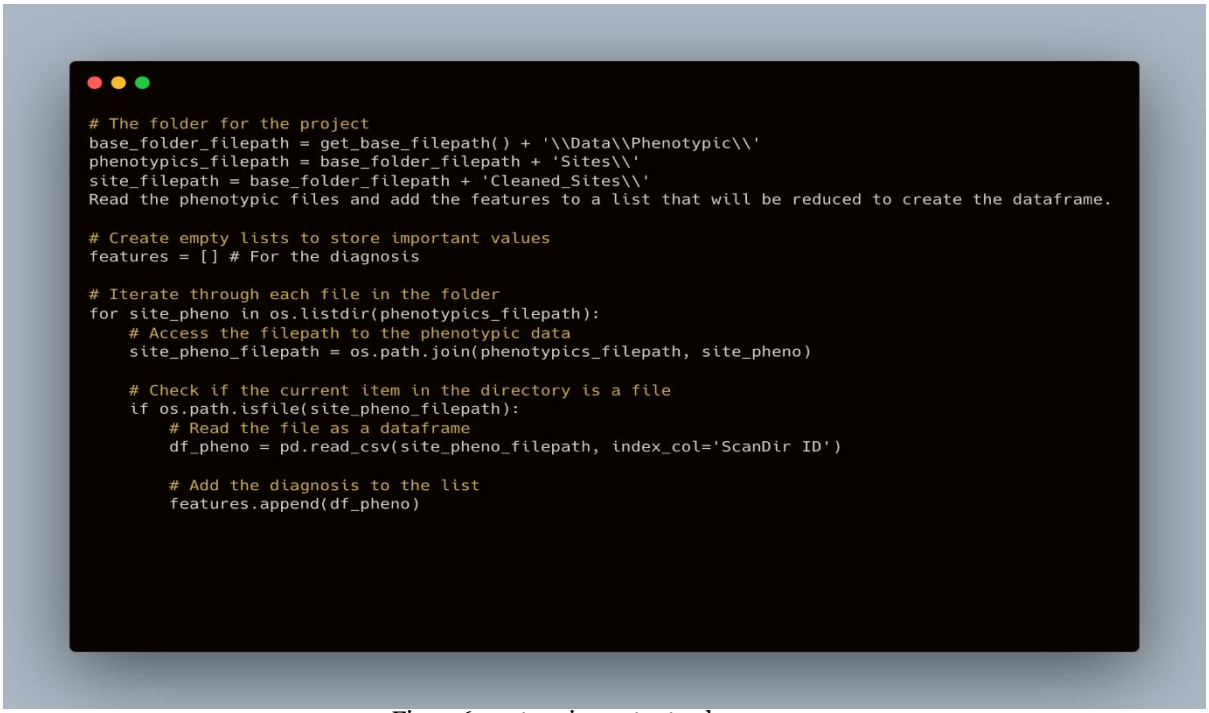
The purpose of cleaning phenotypic data is to Combine all of the individual phenotypic files from each site into a single dataframe and modify that dataframe to be ready to build a machine learning model. Each file has somewhat different recording methods, so it is important that all files follow the same naming conventions. The resulting file is a .csv file of a cleaned dataframe for all subjects and a cleaned dataframe for each of the sites.

Clean and accurate data is essential for drawing valid conclusions and making informed decisions. Data cleaning helps in:

- **Improving Data Quality:** Removing errors and inconsistencies ensures that the analysis results are accurate and reliable.
- **Enhancing Analysis:** Clean data is easier to analyze and interpret, leading to more meaningful insights.
- **Facilitating Data Integration:** Standardized data formats enable the integration of data from different sources, enhancing the scope of analysis.

Detailed Steps of Data Cleaning

1. **Load the Data and add the features to a list that will be reduced to create the dataframe**: Import the raw phenotypic data into a manageable and analyzable format. Use the pandas library to read data from a CSV file into a DataFrame. The DataFrame will serve as the primary data structure for all subsequent cleaning steps.

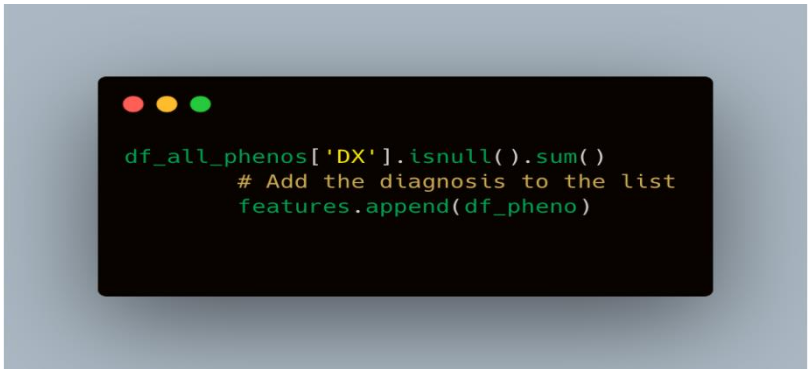


```
# The folder for the project
base_folder_filepath = get_base_filepath() + '\\Data\\Phenotypic\\'
phenotypics_filepath = base_folder_filepath + 'Sites\\'
site_filepath = base_folder_filepath + 'Cleaned_Sites\\'
Read the phenotypic files and add the features to a list that will be reduced to create the dataframe.

# Create empty lists to store important values
features = [] # For the diagnosis

# Iterate through each file in the folder
for site_pheno in os.listdir(phenotypics_filepath):
    # Access the filepath to the phenotypic data
    site_pheno_filepath = os.path.join(phenotypics_filepath, site_pheno)

    # Check if the current item in the directory is a file
    if os.path.isfile(site_pheno_filepath):
        # Read the file as a dataframe
        df_pheno = pd.read_csv(site_pheno_filepath, index_col='ScanDir ID')

        # Add the diagnosis to the list
        features.append(df_pheno)
```

Figure6.12 store important values

2.**Feature Engineering:** Adjust existing features and create new ones to improve the dataframe. Check to see if there are any null values in the diagnosis. There are none, but if there were, this row would need to be dropped

3. since it will not be useful.

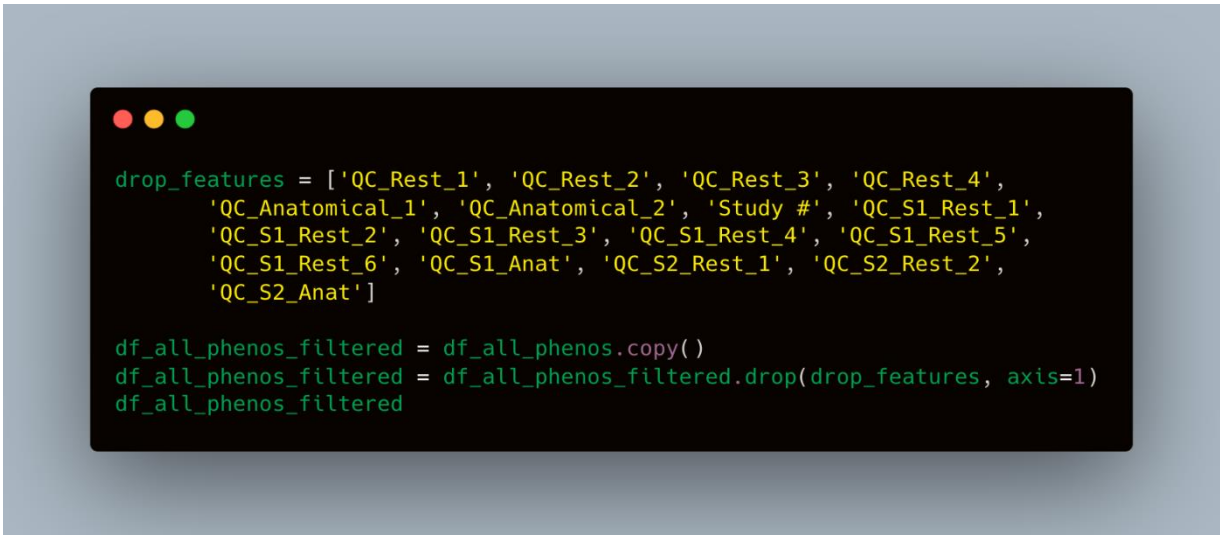


```
df_all_phenos['DX'].isnull().sum()
        # Add the diagnosis to the list
        features.append(df_pheno)
```

Figure6.13 DX null value

Drop Features

Some features in the current dataframe will not be useful for making predictions. The way that the IQ and ADHD values are measured should not be included as an indicator of ADHD. Similarly, the quality of the fMRIs should not determine what diagnosis the patient has.

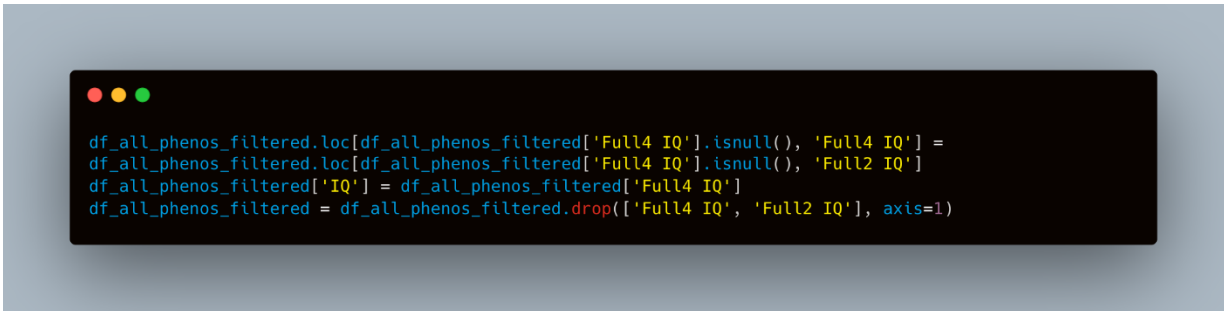


```
drop_features = ['QC_Rest_1', 'QC_Rest_2', 'QC_Rest_3', 'QC_Rest_4',
       'QC_Anatomical_1', 'QC_Anatomical_2', 'Study #', 'QC_S1_Rest_1',
       'QC_S1_Rest_2', 'QC_S1_Rest_3', 'QC_S1_Rest_4', 'QC_S1_Rest_5',
       'QC_S1_Rest_6', 'QC_S1_Anat', 'QC_S2_Rest_1', 'QC_S2_Rest_2',
       'QC_S2_Anat']

df_all_phenos_filtered = df_all_phenos.copy()
df_all_phenos_filtered = df_all_phenos_filtered.drop(drop_features, axis=1)
df_all_phenos_filtered
```

Figure 6.14 drop useless features

IQ

In the phenotypic data cleaning process, it was observed that the subjects who have null values in the 'Full4 IQ' column have corresponding values in the 'Full2 IQ' column. Therefore, to streamline the dataset and avoid redundancy, these two features can be combined to create a single 'IQ' feature. This step helps to consolidate the IQ measurements and simplifies subsequent analyses.



```
df_all_phenos_filtered.loc[df_all_phenos_filtered['Full4 IQ'].isnull(), 'Full4 IQ'] =
df_all_phenos_filtered.loc[df_all_phenos_filtered['Full4 IQ'].isnull(), 'Full2 IQ']
df_all_phenos_filtered['IQ'] = df_all_phenos_filtered['Full4 IQ']
df_all_phenos_filtered = df_all_phenos_filtered.drop(['Full4 IQ', 'Full2 IQ'], axis=1)
```

Figure6.15 single 'IQ' feature

Null Values

- Missing values in the dataset are represented by a placeholder value, -999. These placeholders need to be replaced with None to standardize the representation of missing values.
- Identifying the extent and pattern of missing values helps in deciding the appropriate method for handling them.

```
for col in df_all_phenos_filtered.columns:
    df_all_phenos_filtered.loc[df_all_phenos_filtered[col] == -999, col] = None

null_values = dict()
numeric_cols = ['Gender', 'Age', 'Handedness',
                'Verbal IQ', 'Performance IQ', 'IQ']

df_null_values_train = get_null_values(numeric_cols, df_all_phenos_filtered)

df_null_values_train.head()
```

Figure 6.16 fill null values

Output:

	Gender	Age	Handedness	Verbal IQ	Performance IQ	IQ
null_count	1.0	0.00	1.0	140.0	140.0	0
min_value	0.0	7.09	-999.0	-999.0	-999.0	-999

Figure 6.17 null-output

Gender

There is one null value for gender which can be filled with the most frequent value. The value 1 is the most frequent, so this is what will be used to fill the null value.

```
df_all_phenos_filtered['Gender'] = df_all_phenos_filtered['Gender'].fillna(1)
```

Figure6.18 fill gender null

Handedness

There are seven null values for handedness. The most frequent value is 1 (right-handed), so this will be used to fill the missing values.

The handedness measure at one of the sites measures handedness on a continuous scale unlike the other sites. The values greater than 0 are right-handed and the values less than 0 are left-handed. These categorical values replace the continuous values.

```
df_all_phenos_filtered['Handedness'] = df_all_phenos_filtered['Handedness'].fillna(1)
df_all_phenos_filtered.loc[df_all_phenos_filtered['Handedness'] > 0, 'Handedness'] = 1
df_all_phenos_filtered.loc[df_all_phenos_filtered['Handedness'] < 0, 'Handedness'] = 0
```

Figure 6.19 fill handedness null

Mean-based fill values

The features with IQ have more null values than the other features. For this reason, it is more important to fill the null values with points that are representative of the subject.

With this in mind, the null values for these features will be filled according to the average value for other subjects with the same diagnosis.

Verbal IQ - Performance IQ - IQ

Fill the null Verbal IQ values with the average value for each type of diagnosis.

```
df_all_phenos_filtered.groupby('DX')['Verbal IQ'].mean()
```

```
DX
0.0    114.563694
1.0    109.401639
2.0    105.666667
3.0    107.357895
Name: Verbal IQ, dtype: float64
```

```
df_all_phenos_filtered['Verbal IQ Filled'] = df_all_phenos_filtered['Verbal IQ']

df_all_phenos_filtered.loc[df_all_phenos_filtered['Verbal IQ Filled'].isnull(), 'Verbal IQ Filled'] = df_all_pher

df_all_phenos_filtered.loc[df_all_phenos_filtered['Verbal IQ Filled'] == 0, 'Verbal IQ Filled'] = 114.563694
df_all_phenos_filtered.loc[df_all_phenos_filtered['Verbal IQ Filled'] == 1, 'Verbal IQ Filled'] = 109.401639
df_all_phenos_filtered.loc[df_all_phenos_filtered['Verbal IQ Filled'] == 2, 'Verbal IQ Filled'] = 105.666667
df_all_phenos_filtered.loc[df_all_phenos_filtered['Verbal IQ Filled'] == 3, 'Verbal IQ Filled'] = 107.357895
```

Figure 6.20 fill Verbal IQ null

```
df_all_phenos_filtered.groupby('DX')['Performance IQ'].mean()
```

```
DX
0.0    109.984076
1.0    103.254098
2.0    113.333333
3.0    100.968421
Name: Performance IQ, dtype: float64
```

```
df_all_phenos_filtered['Performance IQ Filled'] = df_all_phenos_filtered['Performance IQ']

df_all_phenos_filtered.loc[df_all_phenos_filtered['Performance IQ Filled'].isnull(), 'Performance IQ Filled'] = d

df_all_phenos_filtered.loc[df_all_phenos_filtered['Performance IQ Filled'] == 0, 'Performance IQ Filled'] = 109.9
df_all_phenos_filtered.loc[df_all_phenos_filtered['Performance IQ Filled'] == 1, 'Performance IQ Filled'] = 103.2
df_all_phenos_filtered.loc[df_all_phenos_filtered['Performance IQ Filled'] == 2, 'Performance IQ Filled'] = 113.3
df_all_phenos_filtered.loc[df_all_phenos_filtered['Performance IQ Filled'] == 3, 'Performance IQ Filled'] = 100.9
```

Figure 6.21 fill Performance IQ null

```
df_all_phenos_filtered.groupby('DX')['IQ'].mean()
```

```
DX
0.0    113.745098
1.0    107.620690
2.0    110.800000
3.0    104.710280
Name: IQ, dtype: float64
```

```
df_all_phenos_filtered['IQ Filled'] = df_all_phenos_filtered['IQ']

df_all_phenos_filtered.loc[df_all_phenos_filtered['IQ Filled'].isnull(), 'IQ Filled'] = df_all_phenos_filtered.lo

df_all_phenos_filtered.loc[df_all_phenos_filtered['IQ Filled'] == 0, 'IQ Filled'] = 113.745098
df_all_phenos_filtered.loc[df_all_phenos_filtered['IQ Filled'] == 1, 'IQ Filled'] = 107.620690
df_all_phenos_filtered.loc[df_all_phenos_filtered['IQ Filled'] == 2, 'IQ Filled'] = 110.800000
df_all_phenos_filtered.loc[df_all_phenos_filtered['IQ Filled'] == 3, 'IQ Filled'] = 104.710280
```

Figure 6.22 fill IQ null

Output:

ScanDir ID	Gender	Age	Handedness	Verbal IQ	Verbal IQ Filled	Performance IQ	Performance IQ Filled	IQ	IQ Filled	DX
1018959	0.0	12.36	1.0	99.0	99.000000	115.0	115.000000	103.0	103.0	0.0
1019436	1.0	12.98	1.0	124.0	124.000000	108.0	108.000000	122.0	122.0	3.0
1043241	1.0	9.12	1.0	128.0	128.000000	106.0	106.000000	120.0	120.0	0.0
1266183	0.0	9.67	1.0	136.0	136.000000	96.0	96.000000	120.0	120.0	0.0
1535233	1.0	9.64	0.0	106.0	106.000000	135.0	135.000000	122.0	122.0	0.0
...	...	...	...	...	...	...	...	...	...	...
15058	0.0	14.08	1.0	NaN	114.563694	NaN	109.984076	115.0	115.0	0.0
15059	0.0	9.05	1.0	NaN	114.563694	NaN	109.984076	103.0	103.0	0.0
15060	1.0	9.76	1.0	NaN	114.563694	NaN	109.984076	137.0	137.0	0.0
15061	1.0	12.04	1.0	NaN	114.563694	NaN	109.984076	98.0	98.0	0.0
15062	1.0	12.88	1.0	NaN	114.563694	NaN	109.984076	108.0	108.0	0.0

728 rows × 10 columns

Figure 6.23 output after fill

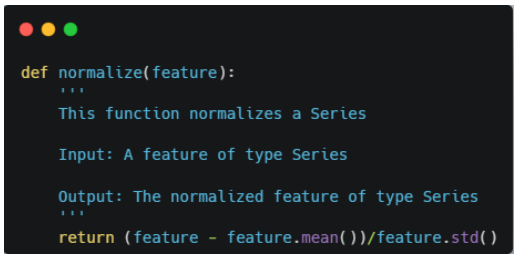
Building ML:

This section focuses on building the machine learning model. We have utilized two distinct machine learning models to explore different aspects of ADHD.

First, we have defined helper functions that will be used to construct the machine learning model. These functions will encapsulate specific tasks involved in model creation, promoting code reusability and readability.

Helper Functions:

1. **normalize ():** This function normalizes a single Pandas Series (feature). It subtracts the mean from each value and then divides by the standard deviation. Ensure the input feature is a numeric Series.



```
def normalize(feature):
    '''
    This function normalizes a Series

    Input: A feature of type Series

    Output: The normalized feature of type Series
    '''
    return (feature - feature.mean())/feature.std()
```

Figure 6.24 - Normalize helper function

2. **get_statistics ():** This function calculates and returns descriptive statistics (mean, standard deviation, minimum, maximum) for the accuracy scores of a list of machine learning models.

```
def get_statistics(accuracy_list, precision_list, recall_list):

    # Initialize lists for mean, std, max, min
    means_acc = []
    stds_acc = []
    maxes_acc = []
    mins_acc = []
    max_precisions = []
    max_recalls = []

    # Calculate descriptive statistics for accuracies
    for acc in accuracy_list:
        acc_np = np.array(acc)
        means_acc.append(acc_np.mean())
        stds_acc.append(acc_np.std())
        maxes_acc.append(acc_np.max())
        mins_acc.append(acc_np.min())

    # Calculate maximum precision and recall for each model
    for precisions, recalls in zip(precision_list, recall_list):
        max_precisions.append(max(precisions))
        max_recalls.append(max(recalls))

    # Create a DataFrame to store the statistics
    stats = pd.DataFrame({
        'Accuracy Mean': means_acc,
        'Accuracy Std': stds_acc,
        'Accuracy Max': maxes_acc,
        'Accuracy Min': mins_acc,
        'Precision Max': max_precisions,
        'Recall Max': max_recalls
    }, index=['Logistic Regression', 'K-Nearest Neighbors', 'Support Vector Machine',
              'Linear Discriminant Analysis', 'Ensemble Voting 1', 'Ensemble Voting 2'])

    return stats
```

Figure 6.25 – Get-Statistics helper function

3. get_accuracies ():

This function evaluates the performance of multiple machine learning models on a given dataset.

```
def get_accuracies(X, y):
    '''
    Get 100 accuracies for the models.

    Input:
        - Set of features
        - Set of targets

    Output:
        - List of 100 accuracies for models
        - LIst of 100 confusion matrices for models
    '''

    accuracies = [[],[],[],[],[],[]]
    matrices = [[],[],[],[],[],[]]

    # Run 100 iterations of evaluating the model
    for i in range(100):
        # Get the accuracy for this iteratio
        accs, cf_matrices = evaluate_models(X, y)

        for i in range(len(models)):
            accuracies[i].append(accs[i])
            matrices[i].append(cf_matrices[i])

    # Return a list of all accuracies and confusion matrices
    return accuracies, matrices
```

Figure 6.26 – get-accuracies helper functions

4. evaluate_models():

This function takes set of features and targets as input. The function iterates over each model in the list, makes predictions on the testing set, and evaluates the accuracy of the model. The function then stores the results in a dictionary and returns a list of dictionaries, one for each model.

```
def evaluate_models(X, y):
    '''
    Evaluate the performance of models on a set of features and targets.

    Input:
        - Set of features
        - Set of targets

    Output:
        - Confusion matrices for models
        - Accuracy of models
    '''
    # Create empty lists to store values
    predictions = dict()
    accuracies = []
    confusion_matrices = []

    # Separate the data into training and testing sets
    X_trn, X_tst, y_trn, y_tst = train_test_split(X, y)

    # Evaluate the accuracies using each model
    for name, model in models:
        pred, acc = make_predictions(model, X_trn, X_tst, y_trn, y_tst)
        predictions[name] = pred
        accuracies.append(acc)

    # Get the confusion matrix for each set of predictions
    for name, model in models:
        confusion_matrices.append(confusion_matrix(predictions[name], y_tst))

    # Return the accuracy and confusion matrices in a list format
    return accuracies, confusion_matrices
```

Figure 6.27 – Evaluate-models helper functions

5. **print_model_cfms():**

takes a list of confusion matrices as input and prints the average confusion matrix for each model

```
def print_model_cfms(cfms):
    '''
    Print all model confusion matrices

    Input: A list of confusion matrices for each model

    Output: Average confusion matrix for each model printed
    '''
    for i, (name, model) in enumerate(models):
        plt.figure(figsize=(3,2))
        print_confusion_matrix(get_model_cfms(cfms)[i], name)
```

Figure 6.28 – print-model-cfms helper function

The First Model

This model will employ a dataset containing phenotypic data from the ADHD-200 Competition's seven training sites. This dataset includes personal characteristics of each subject, and the model will use this information to predict ADHD.

Here, we import the necessary libraries for building and evaluating the machine learning model:

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pickle

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy_score, confusion_matrix
```

Figure 6.29 - First- Model- Imports

Building the model:

Separate the data in a pandas data frame “df_pheno” into features and target variable by removing the column named “DX” (the target variable) from the data frame “df_pheno” and create a new data frame named X.

```
df_pheno = pd.read_csv(filepath, index_col='ID')

X = df_pheno.drop('DX', axis=1)
y = df_pheno['DX']
```

Figure 6.30– separate-Data frame

ID	Gender	Age	Handedness	Verbal IQ	Performance IQ	IQ
1038415	1	14.92	1.0	109.0	103.0	107.0
1201251	1	12.33	1.0	115.0	103.0	110.0
1245758	0	8.58	1.0	121.0	88.0	106.0
1253411	1	8.08	1.0	119.0	106.0	114.0
1419103	0	9.92	1.0	124.0	76.0	102.0
...	...	...	...	...	...	...
25008	1	15.90	1.0	127.0	105.0	117.0
25009	1	13.87	1.0	135.0	119.0	130.0
25012	0	15.04	1.0	95.0	88.0	90.0
25013	1	13.64	1.0	114.0	109.0	113.0
25014	1	13.72	1.0	103.0	103.0	104.0

171 rows × 6 columns

Figure 6.31 – separated-Data frame output

Evaluating the model:

After building the data frame we will use, determine the accuracy of using this data frame for multiple machine learning models using "get_accuracies" helper function

```
accs, cfms = get_accuracies(X, y)
```

Figure 6.32 – Evaluate-models

Then, view confusion matrices of the models using "print_model_cfms" helper function

```
print_model_cfms(cfms)
```

Figure 6.33 – Print-confusion-matrices

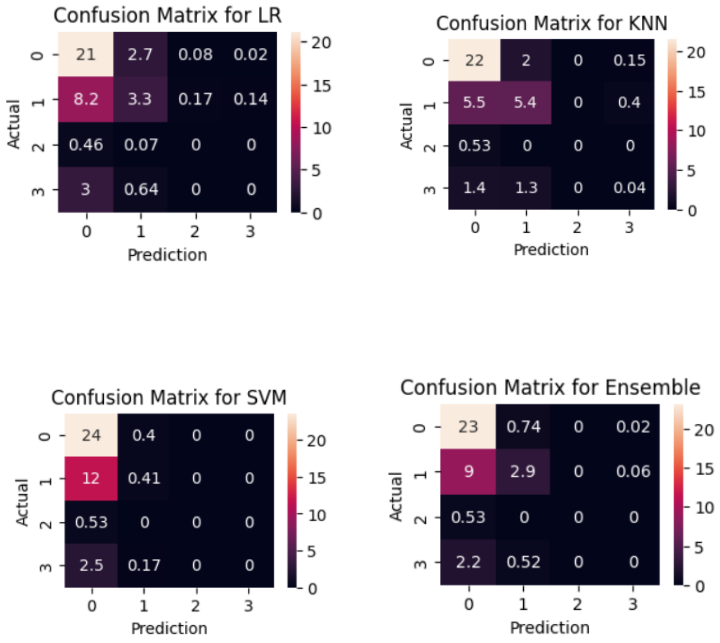


Figure 6.34–confusion-matrices

Finally, calculates various statistical measures, precision, recall, and accuracy for the different classifiers

```
stats = get_statistics(accuracies, precisions, recalls)

results = pd.DataFrame(stats,
                       index=['LR_multiclass', 'KNN_multiclass', 'SVM_multiclass',
                              'LDA_multiclass', 'Ensemble_multiclass', 'Ensemble2_multiclass'])

# Display results
print("Descriptive Statistics for Accuracy and Maximum Precision/Recall:")
results
```

Figure 6.35 – Statistics

	LR_multiclass	KNN_multiclass	SVM_multiclass	LDA_multiclass	Ensemble_multiclass	Ensemble2_multiclass
Mean	0.574651	0.626512	0.710233	0.611395	0.600930	0.590930
STD	0.066700	0.061249	0.067500	0.058213	0.069619	0.068716
Max	0.727442	0.767442	0.730000	0.744186	0.707442	0.714186
Min	0.418605	0.418605	0.608605	0.488372	0.418605	0.418605
Precision	0.596037	0.539064	0.725073	0.620957	0.619064	0.618605
Recall	0.527494	0.553732	0.735796	0.615368	0.592847	0.588487

Figure 6.36– Statistics-output

In Conclusion we found that “SVM” was the most accurate method.

Training the model:

First, use the “train_test_split” function to split data into training and testing sets.

```
X_trn, X_tst, y_trn, y_tst =
train_test_split(X, y)
```

Figure 6.37 – Split-Data

Then, the model-training stage begin

```
svm.fit(X_trn,y_trn)
```

Figure 6.38– Train-Model

Testing the model:

uses the trained model to predict labels for the testing features

```
svm.predict(X_tst)
```

Figure 6.39 – Test-Model

```
array([1, 0, 0, 0, 3, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 1, 0, 1, 3,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0], dtype=int64)
```

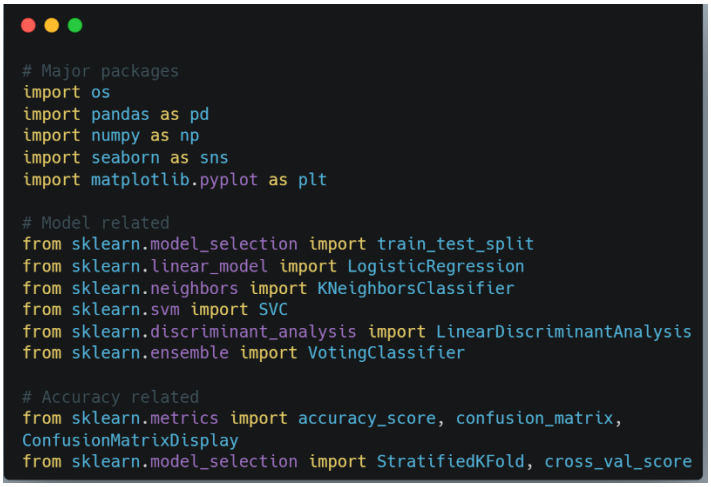
Figure 6.2.41– Predictions

6.5-The Second Model

This model will focus on the neurological underpinnings of the disorder,

allowing us to predict ADHD diagnosis.

Here, we import the necessary libraries for building and evaluating the machine learning model:



```
# Major packages
import os
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Model related
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import VotingClassifier

# Accuracy related
from sklearn.metrics import accuracy_score, confusion_matrix,
ConfusionMatrixDisplay
from sklearn.model_selection import StratifiedKFold, cross_val_score
```

Figure 6.42 -Second- Model- Imports

Building the model:

First, Condenses phenotypic data to a vector:
It begins by creating an empty list called “patient_ids”. This list will be used to store the extracted patient IDs.
Then, employs a for loop to iterate through each sub list within the “pheno_index” list.
Finally, store extracted IDs by appending each extracted patient ID to the “patient_ids” list. This essentially condenses the phenotypic data into a vector containing just the patient IDs

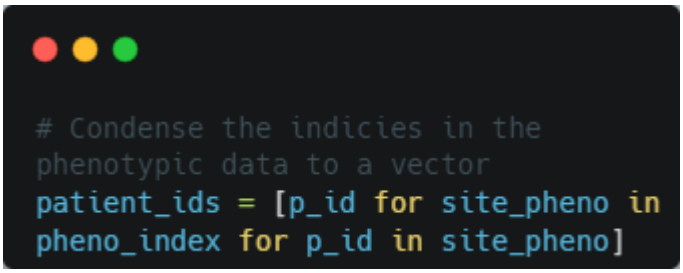


Figure 6.43 Condenses- phenotypic

Second, creates a DataFrame called “df_subject_x_region”, the first argument “subject_features”, is a list containing features and the second argument “index=subjects”, sets the index of the DataFrame to the “subjects”, which is a list containing subject IDs.

Figure 6.44 subject_region_dataframe

Add the diagnosis Series to the regions dataframe by creating copy of the dataframe and filtering the “diagnosis” series by removing rows where the index matches the values in the “subjects_dropped”

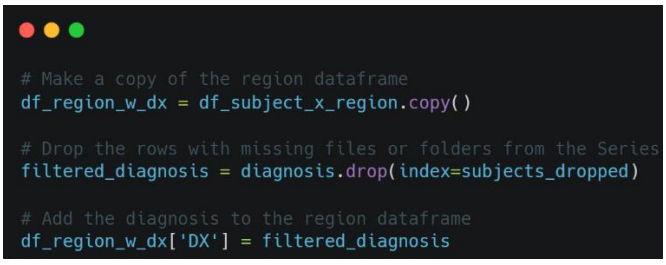


Figure 6.45 subjects_dropped

Separate the data in a pandas dataframe “df_region_w_dx” into features and target variable by removing the column named 'DX'

```
X = df_region_w_dx.drop('DX', axis=1)
y = df_region_w_dx['DX']
```

Figure 6.46 df_region_w_dx

Evaluating the models:

After building the dataframe we will use, determine the accuracy of using this dataframe for multiple machine learning models using “get_accuracies” helper function

```
accs, cfms = get_accuracies(X, y)
```

Figure 6.47 – Evaluate-models

Then, view confusion matrices of the models using “print_model_cfms” helper function

```
print_model_cfms(cfms)
```

Figure 6.48 – Print-confusion-matrices

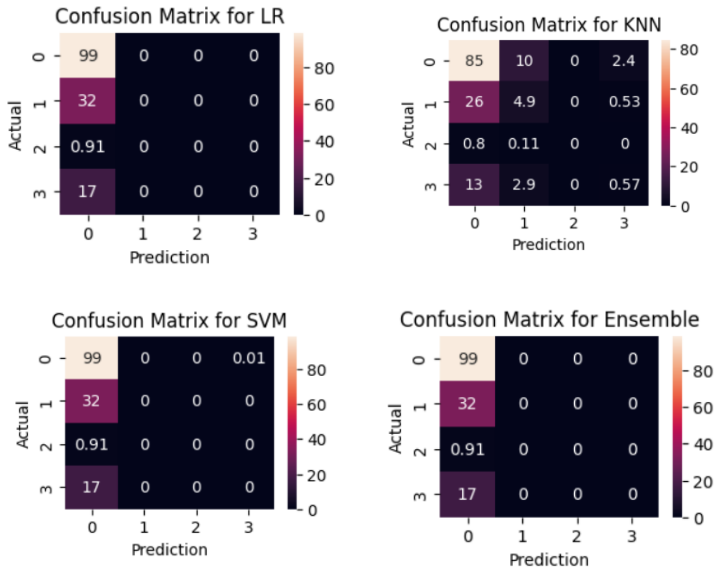


Figure 6.50 -confusion-matrices

calculates various statistical measures, precision, recall, and accuracy for the different classifiers

```
stats = get_statistics(accuracies, precisions, recalls)

results = pd.DataFrame(stats,
                       index=['LR_multiclass', 'KNN_multiclass', 'SVM_multiclass',
                              'LDA_multiclass', 'Ensemble_multiclass', 'Ensemble2_multiclass'])

# Display results
print("Descriptive Statistics for Accuracy and Maximum Precision/Recall:")
results
```

Figure 6.51– Model-Evaluation

	LR_multiclass	KNN_multiclass	SVM_multiclass	LDA_multiclass	Ensemble_multiclass	Ensemble2_multiclass
Mean	0.668344	0.677643	0.736369	0.659936	0.612675	0.590191
STD	0.034984	0.032718	0.035317	0.035692	0.041524	0.052428
Max	0.734204	0.726943	0.793376	0.764204	0.738790	0.747898
Min	0.615860	0.657707	0.645032	0.632229	0.528662	0.548662
Precision	0.721604	0.739064	0.787251	0.760957	0.749064	0.748605
Recall	0.737494	0.743732	0.795796	0.775368	0.732847	0.748487

Figure 6.52– Model-Evaluation

Training the model:

First, use the “train_test_split” function to split data into training and testing sets.

```
X_trn, X_tst, y_trn, y_tst =
train_test_split(X, y)
```

Figure 6.53 – Split-Data

Then, the model-training stage begin

```
svm.fit(X_trn,y_trn)
```

Figure 6.54– Train-Model

Testing the model:

uses the trained model to predict labels for the testing features

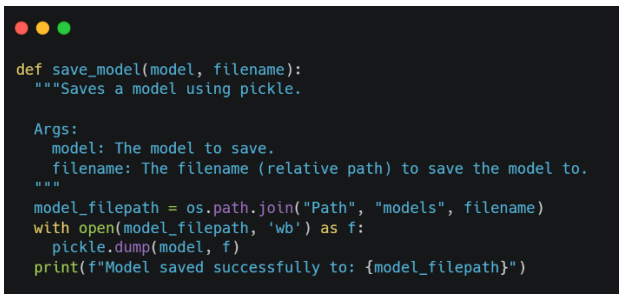
Figure 6.55 – Test-Model

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 2, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 0, 0, 2, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       2, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 3, 0, 0,
       0, 0, 0])
```

Figure 6.56– Predictions

Models deployment

The code below describes the model-deployment stage. We have Built a function to save the models using “Pickle” module.



```python
def save_model(model, filename):
  """Saves a model using pickle.

  Args:
    model: The model to save.
    filename: The filename (relative path) to save the model to.
  """
  model_filepath = os.path.join("Path", "models", filename)
  with open(model_filepath, 'wb') as f:
    pickle.dump(model, f)
  print(f"Model saved successfully to: {model_filepath}")
```

Figure 6.57 – Save-Model

Then we have built “Fast API” application for deploying the saved model

Load a Pre-trained Model:

The code loads a pre-trained model.

Define Input format:

It creates a class named Features that specifies the expected format for the user's input.

Web Service Endpoint:

defines a web service endpoint (/predict) using Fast API. This allows users to send data to the service through a web request.

Process and Make Prediction:

When a user sends data through the /predict endpoint, the code extracts the features and converts them into a format suitable for the loaded model. Then uses the pre-trained model to generate a prediction.

Return the Prediction:

Finally, the predicted value is sent back to the user as a response

```python
app = FastAPI()

class Features(BaseModel):
    Gender: str
    Age: float
    Handedness: float
    IQ: float = None  # Optional for the model without IQs

    @validator('Gender')
    def validate_gender(cls, value):
        if value.upper() not in ("M", "F"):
            raise ValueError("Gender must be 'M' or 'F'")
        return value.upper()  # Convert to uppercase for consistent handling

@app.post("/predict")
async def predict(features: Features = Body(...)):
    # Convert features to a format suitable for the model
    gender_code = 0 if features.Gender == "M" else 1
    data = np.array([[features.Age, features.Handedness, gender_code, features.IQ if
features.IQ is not None else 0]]).reshape(1, -1)

    # Make prediction using the loaded model
    prediction = model.predict(data)[0]  # Assuming prediction is a NmPy int
    response = {"prediction": int(prediction)}  # Convert to int before returning
    return response
```

Figure 6.58 –Model-API

6.3 Games

In this subsection, we will be discussing the implementation tools and providing an overview of the most important code samples used to build our games in the app.

1. Name recall

This code is written in C# and is used in the Unity framework to apply animation effects to buttons when they are clicked. Let's explain each part of the code:

Variable Definitions

```
public Animator animator; // Reference to the Animator component
public Button[] buttons; // Reference to the Button component
public Animator[] additionalAnimators; // Reference to the additional Animator components
```

Figure 6.59- Variable Definitions

- animator: Reference to the Animator component that controls the main animation.
- buttons: An array of buttons that will trigger the animation when clicked.
- additionalAnimators: An array of Animator components that control additional animations.

Start Method

```
void Start()
{
    // Add a listener for the button click event
    // Add a listener for the click event on each button
    foreach (Button button in buttons)
    {
        button.onClick.AddListener(PlayAnimation);
        button.onClick.AddListener(DisableButtonOnClick);
    }
}
```

Figure 6.60- Start Method

- This method is executed once at startup.
- A foreach loop is used to add listeners to the click event for each button in the buttons array.
- Each button has listeners added for the click event that call the PlayAnimation and DisableButtonOnClick methods.

PlayAnimation Method

```
void PlayAnimation()
{
    // Play the animation
    animator.SetTrigger("movebackup");
    foreach (Animator additionalAnimator in additionalAnimators)
    {
        additionalAnimator.SetTrigger("hiddenagain");
        additionalAnimator.SetTrigger("movedown");
    }
}
```

Figure 6.61 - PlayAnimation Method

When any button is clicked, the main animation is triggered using SetTrigger("movebackup").

- Additional animations are triggered for each element in additionalAnimators using SetTrigger("hiddenagain") and SetTrigger("movedown").

DisableButtonOnClick Method

```
void DisableButtonOnClick()
{
    // Disable the button
    foreach (Button button in buttons)
    {
        button.interactable = false;
        button.gameObject.SetActive(false);
    }
}
```

Figure 6.62- DisableButtonOnClick Method

- This method disables all the buttons in the buttons array.
- The interactable property is set to false to disable user interaction with the button.
- The button is hidden by setting its SetActive property to false.

2. Focus listening

Code Explanation

This code is a Unity script written in C# that handles drag-and-drop mechanics in a game. The script manages objects, checks their positions when dropped, updates the score, and manages the player's lives. Let's break down each part of the code:

Namespace Imports

Figure 6.63- Namespace Imports

- Using `System.Collections` and using `System.Collections.Generic`: These namespaces provide classes and interfaces for collections of objects, such as lists and dictionaries.
- `using UnityEngine`: This namespace provides core Unity engine functionalities.
- `using UnityEngine.UI`: This namespace provides access to the UI components in Unity, such as Text.

Class Definition

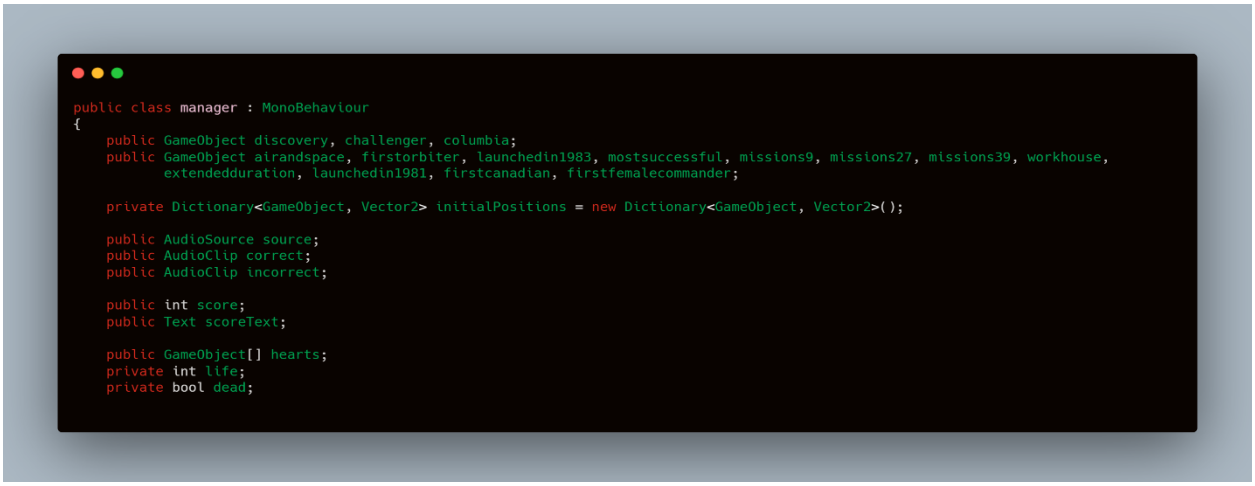


Figure 6.64- Class Definition

- `discovery, challenger, columbia`: These are the target objects that other objects will be dropped onto.
- `airandspace, firstorbiter, etc.`: These are the draggable objects.
- `initialPositions`: A dictionary to store the initial positions of the draggable objects.
- `source`: The audio source component for playing sound effects.
- `correct` and `incorrect`: Audio clips for correct and incorrect drops.
- `score` and `scoreText`: Variables for keeping track of and displaying the score.
- `hearts`: An array of GameObjects representing the player's lives.

- `life`: An integer representing the current number of lives.
- `dead`: A boolean flag to check if the player is out of lives.

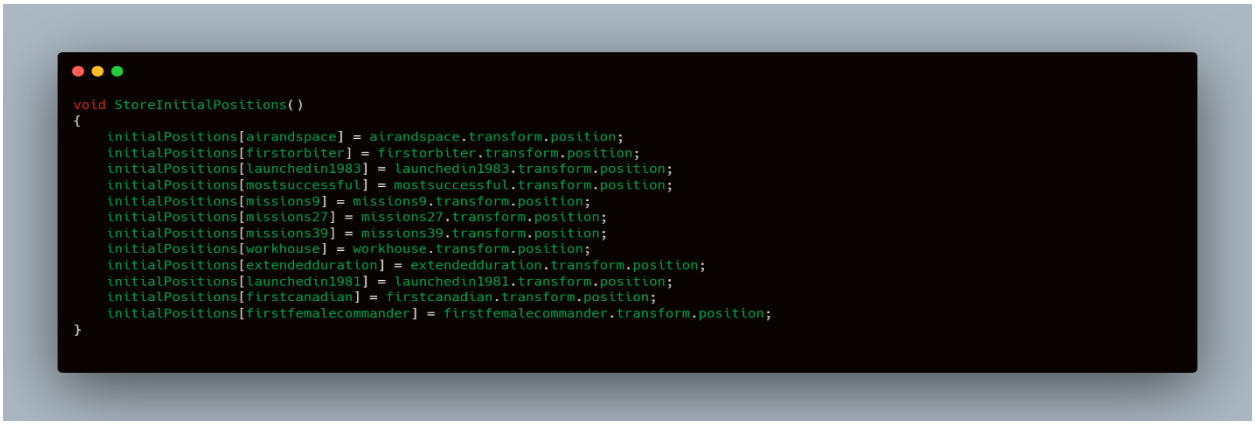
Start Method

```
void Start()
{
    StoreInitialPositions();

    life = hearts.Length;
    scoreText.text = score.ToString();
    UpdateHearts();
}
```

Figure 6.65- Start Method

- `StoreInitialPositions ()`: Calls a method to store the initial positions of the draggable objects.
- Initializes `life` with the number of hearts.
- Updates the score text.
- Updates the display of hearts.

StoreInitialPositions Method



```
void StoreInitialPositions()
{
    initialPositions[airandspace] = airandspace.transform.position;
    initialPositions[firstorbiter] = firstorbiter.transform.position;
    initialPositions[launchedin1983] = launchedin1983.transform.position;
    initialPositions[mostsuccessful] = mostsuccessful.transform.position;
    initialPositions[missions9] = missions9.transform.position;
    initialPositions[missions27] = missions27.transform.position;
    initialPositions[missions39] = missions39.transform.position;
    initialPositions[workhouse] = workhouse.transform.position;
    initialPositions[extendedduration] = extendedduration.transform.position;
    initialPositions[launchedin1981] = launchedin1981.transform.position;
    initialPositions[firstcanadian] = firstcanadian.transform.position;
    initialPositions[firstfemalecommander] = firstfemalecommander.transform.position;
}
```

Figure 6.66- StoreInitialPositions Method

- This method stores the initial positions of all draggable objects in the `initialPositions` dictionary.

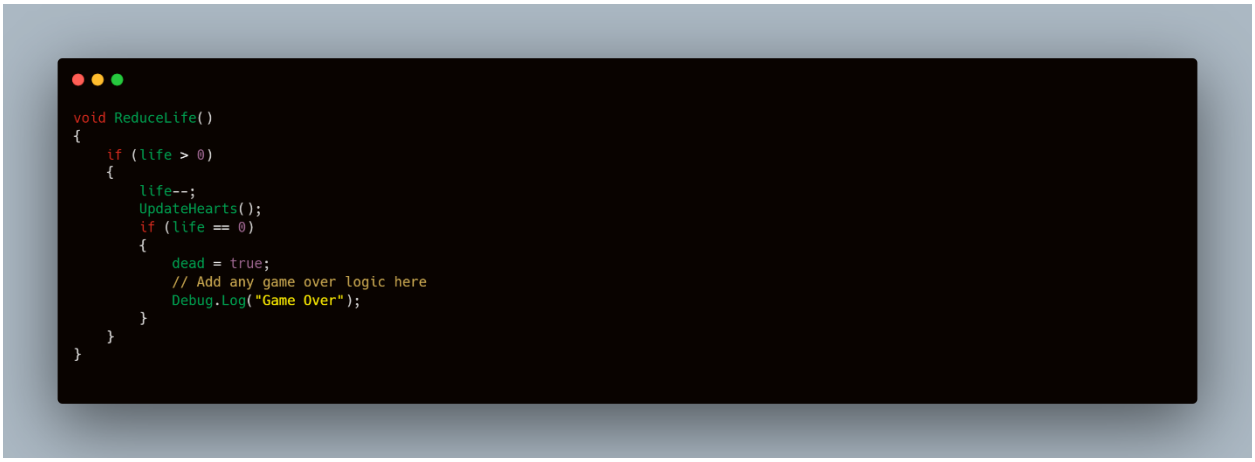
UpdateHearts Method

```
void UpdateHearts()
{
    for (int i = 0; i < hearts.Length; i++)
    {
        hearts[i].SetActive(i < life);
    }
}
```

Figure 6.67-UpdateHearts Method

- This method updates the active state of heart GameObjects based on the current `life` value.

ReduceLife Method

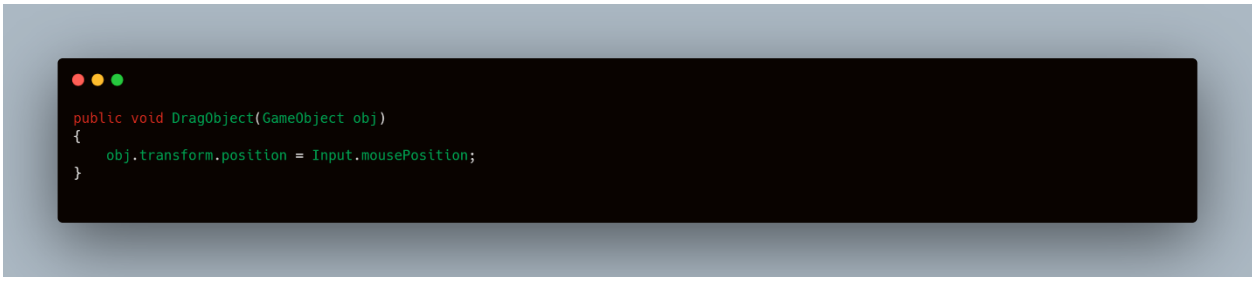


```
void ReduceLife()
{
    if (life > 0)
    {
        life--;
        UpdateHearts();
        if (life == 0)
        {
            dead = true;
            // Add any game over logic here
            Debug.Log("Game Over");
        }
    }
}
```

Figure 6.68-ReduceLife Method

- This method reduces the player's life by one, updates the hearts, and checks if the player is out of lives.

DragObject Method



```
public void DragObject(GameObject obj)
{
    obj.transform.position = Input.mousePosition;
}
```

Figure 6.69 -DragObject Method

- This method sets the position of the dragged object to the current mouse position.

DropObject Method

```
public void DropObject(GameObject obj, GameObject target)
{
    float Distance = Vector3.Distance(obj.transform.position, target.transform.position);
    if (Distance < 100)
    {
        obj.transform.position = target.transform.position;
        source.clip = correct;
        source.Play();

        score += 1000;
        scoreText.text = score.ToString();
        Debug.Log("Correct! Score: " + score);
    }
    else
    {
        obj.transform.position = initialPositions[obj];
        source.clip = incorrect;
        source.Play();

        ReduceLife();
        Debug.Log("Incorrect! Life: " + life);
    }
}
```

Figure 6.70-DropObject Method

- This method checks if the dropped object is within a certain distance (100 units) of the target object.
- If within range, the object snaps to the target's position, plays the correct sound, and updates the score.
- If not, the object returns to its initial position, plays the incorrect sound, and reduces the player's life.

Drag and Drop Methods for Specific Objects

```
public void Dragairandspace() { DragObject(airandspace); }
public void Dragfirstorbiter() { DragObject(firstorbiter); }
public void Draglaunchedin1983() { DragObject(launchedin1983); }
public void Dragmostsuccessful() { DragObject(mostsuccessful); }
public void Dragmissions9() { DragObject(missions9); }
public void Dragmissions27() { DragObject(missions27); }
public void Dragmissions39() { DragObject(missions39); }
public void Dragworkhouse() { DragObject(workhouse); }
public void Dragextendedduration() { DragObject(extendedduration); }
public void Draglaunchedin1981() { DragObject(launchedin1981); }
public void Dragfirstcanadian() { DragObject(firstcanadian); }
public void Dragfirstfemalecommander() { DragObject(firstfemalecommander); }

public void Dropairandspace() { DropObject(airandspace, discovery); }
public void Dropfirstorbiter() { DropObject(firstorbiter, columbia); }
public void Droplaunchedin1983() { DropObject(launchedin1983, challenger); }
public void Dropmostsuccessful() { DropObject(mostsuccessful, discovery); }
public void Dropmissions9() { DropObject(missions9, challenger); }
public void Dropmissions27() { DropObject(missions27, columbia); }
public void Dropmissions39() { DropObject(missions39, discovery); }
public void Dropworkhouse() { DropObject(workhouse, challenger); }
public void Dropextendedduration() { DropObject(extendedduration, columbia); }
public void Droplaunchedin1981() { DropObject(launchedin1981, columbia); }
public void Dropfirstcanadian() { DropObject(firstcanadian, challenger); }
public void Dropfirstfemalecommander() { DropObject(firstfemalecommander, discovery); }
```

Figure 6.71-Drag and Drop Methods for Specific Objects

These methods provide specific drag and drop functionality for each draggable object and its corresponding target.

3. Retention

Namespace Imports

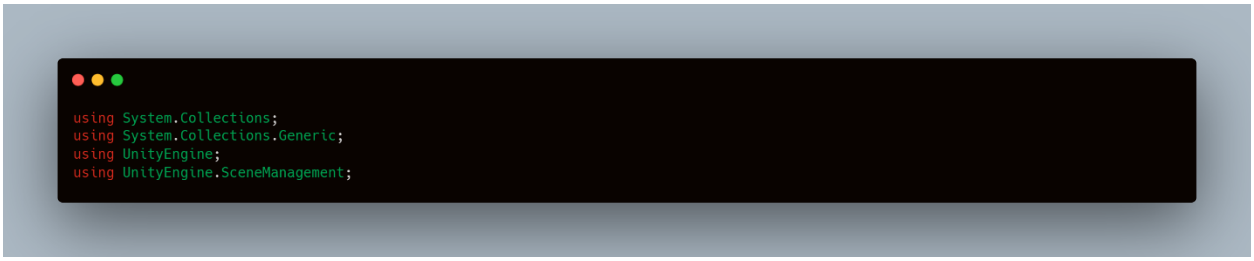


Figure 6.72-Namespace Imports

- using System.Collections and using System.Collections. Generic: These namespaces provide classes and interfaces for collections of objects like lists and dictionaries.
- using UnityEngine: This namespace provides core Unity functionalities.
- using UnityEngine.SceneManagement: This namespace provides access to scene management functionalities in Unity.

Class Definition

Figure 6.73-Class Definition

- The class `begins` inherits from `MonoBehaviour`, making it a Unity component that can be attached to GameObjects.
- public int scene Num: A public integer variable that could potentially be used to store the scene number to load. However, in this particular script, it is not used.

GoNextScene Method

Figure 6.74-GoNextScene Method

- `public void GoNextScene ()`: A public method that can be called to trigger a scene change.

- `SceneManager.LoadScene(3)`: This line of code uses Unity's SceneManager to load a new scene. The scene with the build index 3 will be loaded. The build index corresponds to the order of scenes as listed in the "Build Settings" in Unity.

The script below is for 'health:

Namespace Imports

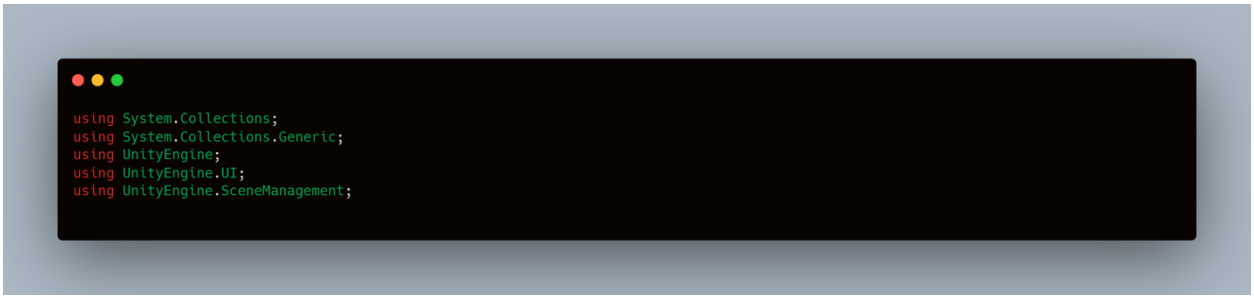


Figure 6.75-Namespace Imports

• using `System.Collections` and `using System.Collections.Generic`: These namespaces provide classes and interfaces for collections of objects like lists and dictionaries.
• using `UnityEngine`: This namespace provides core Unity functionalities.
• using `UnityEngine.UI`: This namespace provides access to UI components in Unity, such as `Text`.
• using `UnityEngine.SceneManagement`: This namespace provides access to scene management functionalities in Unity (although it isn't utilized in this script).

Class Definition

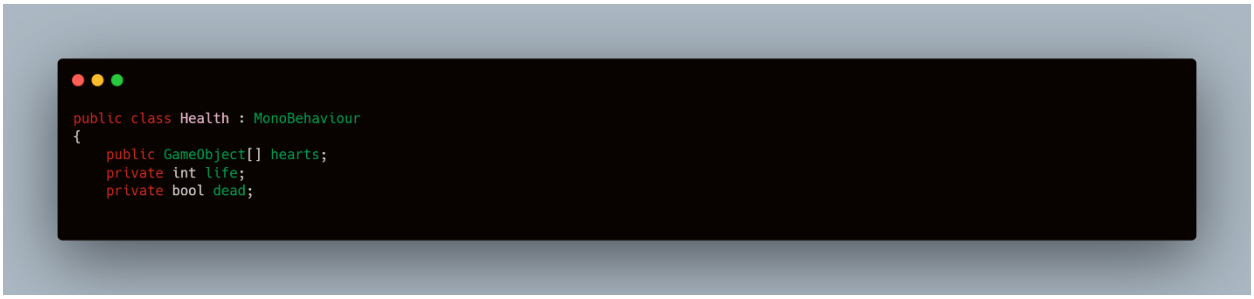


Figure 6.76-Class Definition

- The class Health inherits from `MonoBehaviour`, making it a Unity component that can be attached to GameObjects.
- `public GameObject [] hearts`: An array of GameObjects representing the player's hearts (lives).
- `private int life`: An integer representing the current number of lives.
- `private bool dead`: A Boolean flag to check if the player is dead.

Start Method

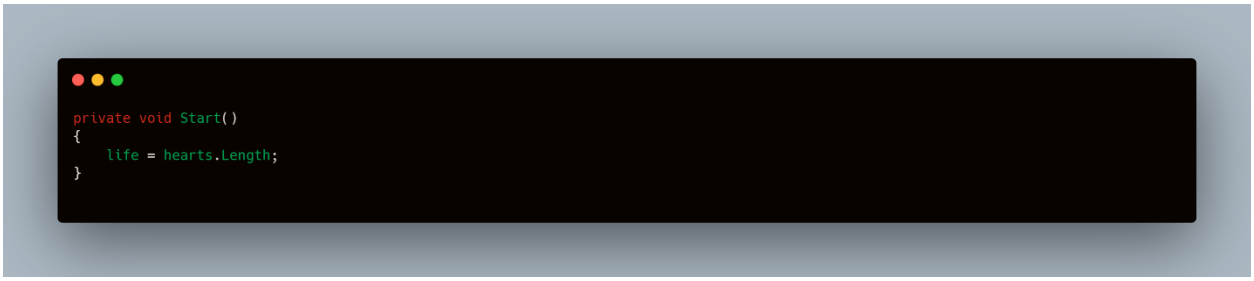


Figure 6.77-Start Method

- private void Start (): This method is called once when the script instance is being loaded.
- life = hearts. Length: Initializes the life variable with the number of hearts.

Update Method

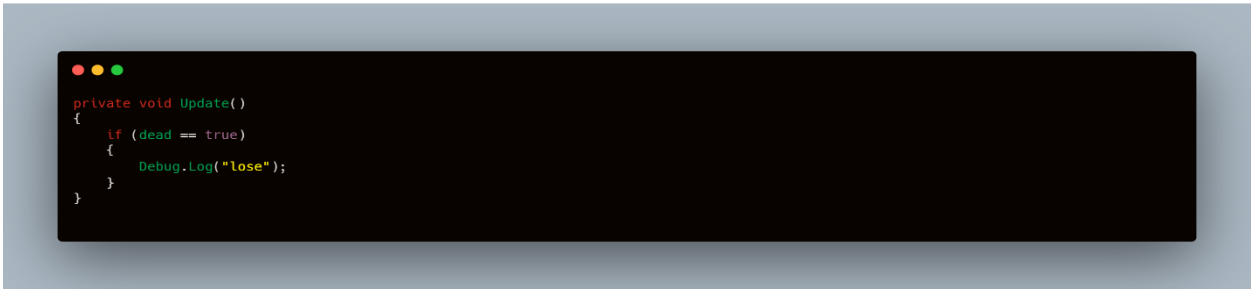


Figure 6.78-Update Method

• private `void Update ()`: This method is called once per frame.
• The `if` statement checks if the player is dead. If `dead` is `true`, it logs the message "keqwfi" to the console.

TakeDamage Method

Figure 6.79-TakeDamage Method

- public void TakeDamage (int d): A public method that reduces the player's life by a specified amount (d).
- life -= d: Decreases the life variable by d.
- Destroy(hearts[life]. game Object (): Destroys the heart Game Object that corresponds to the current life value.
- The if statement checks if life is less than or equal to 1. If so, it sets dead to true; otherwise, it sets dead to false.

The script below is for 'pause menu:

Namespace Imports

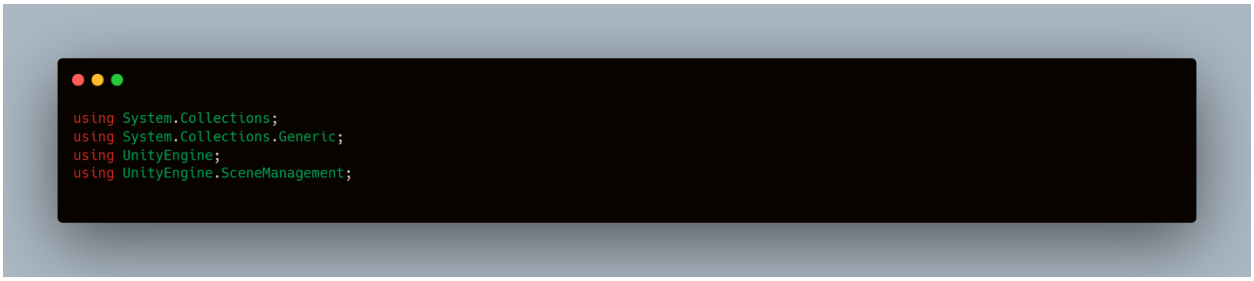


Figure 6.80-Namespace Imports

- using System.Collections and using System.Collections. Generic: These namespaces provide classes and interfaces for collections of objects like lists and dictionaries.
- using UnityEngine: This namespace provides core Unity functionalities.
- using UnityEngine.SceneManagement: This namespace provides access to scene management functionalities in Unity.

Class Definition

Figure 6.81-Class Definition

- The class pause menu is inherited from `MonoBehaviour`, making it a Unity component that can be attached to GameObjects.
- `[SerializeField] GameObject pauseMenu`: A serialized field for the pause menu GameObject, allowing it to be assigned in the Unity editor.
- `[SerializeField] List<AudioSource> gameAudioSources`: A serialized field for a list of `AudioSource` components, allowing multiple audio sources to be managed.

Pause Method

```
public void Pause()
{
    pauseMenu.SetActive(true);
    Time.timeScale = 0f;
    foreach (AudioSource audioSource in gameAudioSources)
    {
        audioSource.Pause(); // إيقاف كل الأصوات عند الضغط على زر الإيقاف المؤقت
    }
}
```

Figure 6.82-Pause Method

- `public void Pause ()`: A public method to pause the game.
- `pauseMenu.SetActive(true)`: Activates the pause menu GameObject.
- `Time.timeScale = 0f`: Freezes the game by setting the time scale to 0.
- The `foreach` loop iterates over all audio sources in `gameAudioSources` and pauses them.

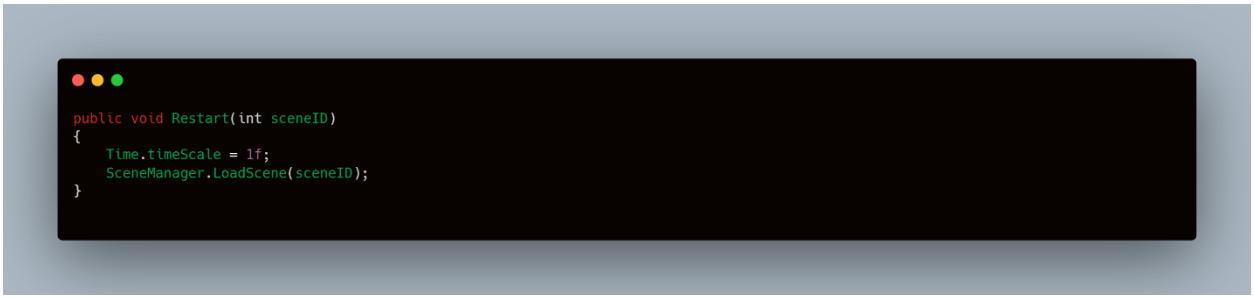
Resume Method

```
public void Resume()
{
    pauseMenu.SetActive(false);
    Time.timeScale = 1f;
    foreach (AudioSource audioSource in gameAudioSources)
    {
        audioSource.UnPause(); // استئناف كل الأصوات عند الضغط على زر الاستمرار
    }
}
```

Figure 6.82-Resume Method

- `public void Resume ()`: A public method to resume the game.
- `pauseMenu.SetActive(false)`: Deactivates the pause menu GameObject.
- `Time.timeScale = 1f`: Resumes the game by setting the time scale back to 1.
- The `foreach` loop iterates over all audio sources in `gameAudioSources` and unpauses them.

Restart Method



```
public void Restart(int sceneID)
{
    Time.timeScale = 1f;
    SceneManager.LoadScene(sceneID);
}
```

Figure 6.83-Restart Method

• public `void Restart (int sceneID)`: A public method to restart the game.
• Time.timeScale `= 1f`: Ensures the game time scale is set to 1 before restarting.
• SceneManager.LoadScene`(sceneID)`: Loads the scene specified by the `sceneID` parameter, effectively restarting the game.

4. Synthesis

Code Explanation

This Unity C# script is designed to manage a drag-and-drop game. The game involves dragging historical figures to specific target circles. Let's break down the code:

Namespace Imports

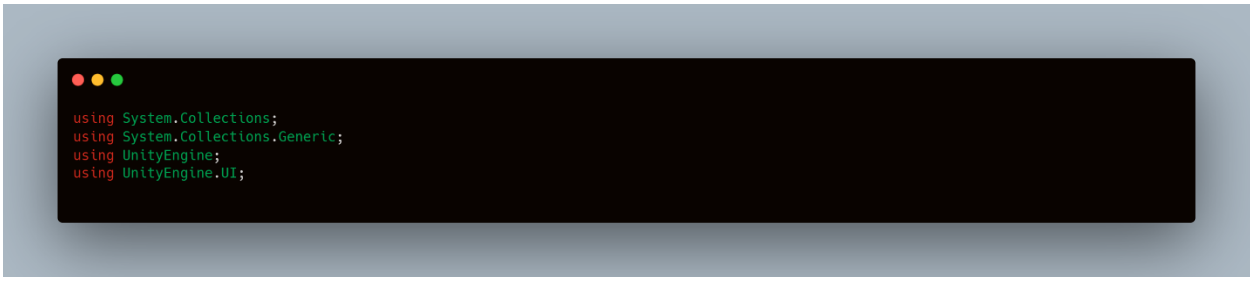


Figure 6.83-Namespace Imports

- using System.Collections and using System.Collections. Generic: These namespaces provide classes and interfaces for collections of objects like lists and dictionaries.
- using UnityEngine: This namespace provides core Unity functionalities.
- using UnityEngine.UI: This namespace provides access to UI components in Unity, such as Text.

Class Definition

Figure 6.84-Class Definition

- `circle1, circle2, circle3, circle4, circle5`: Target circles for dropping the historical figures.
- `ptolemy, alexander, isis, juliuscaesar, pompey, ptolemyxiii`: Draggable objects representing historical figures.
- `initialPositions`: A dictionary to store the initial positions of the draggable objects.

- `source`: The audio source component for playing sound effects.
- `correct` and `incorrect`: Audio clips for correct and incorrect drops.
- `score` and `scoreText`: Variables for keeping track of and displaying the score.
- `hearts`: An array of GameObjects representing the player's lives.
- `life`: An integer representing the current number of lives.
- `dead`: A boolean flag to check if the player is out of lives.

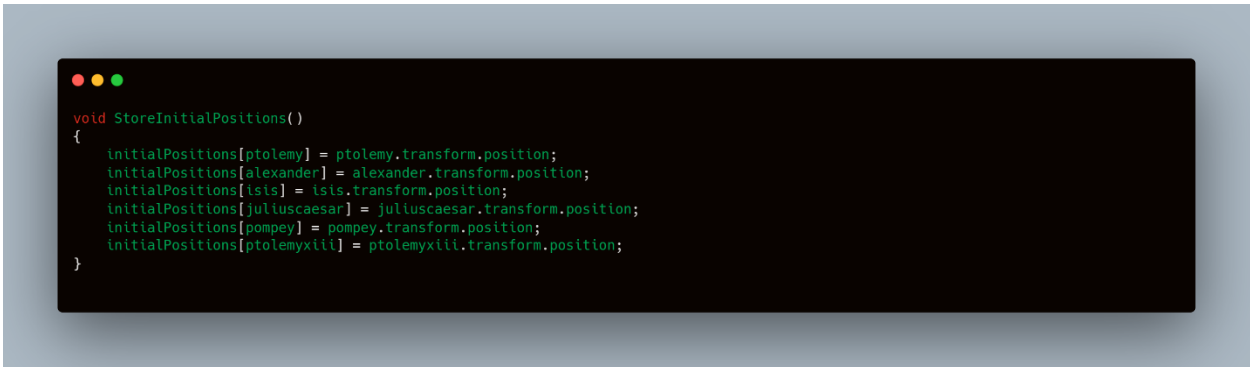
Start Method

```
void Start()
{
    StoreInitialPositions();

    life = hearts.Length;
    scoreText.text = score.ToString();
    UpdateHearts();
}
```

Figure 6.85-Start Method

- Calls StoreInitialPositions () to save the initial positions of the draggable objects.
- Initializes life with the number of hearts.
- Updates the score text.
- Updates the display of hearts.

StoreInitialPositions Method



```
void StoreInitialPositions()
{
    initialPositions[ptolemy] = ptolemy.transform.position;
    initialPositions[alexander] = alexander.transform.position;
    initialPositions[isis] = isis.transform.position;
    initialPositions[juliuscaesar] = juliuscaesar.transform.position;
    initialPositions[pompey] = pompey.transform.position;
    initialPositions[ptolemyxiii] = ptolemyxiii.transform.position;
}
```

Figure 6.86-StoreInitialPositions Method

- Stores the initial positions of all draggable objects in the initialPositions dictionary.

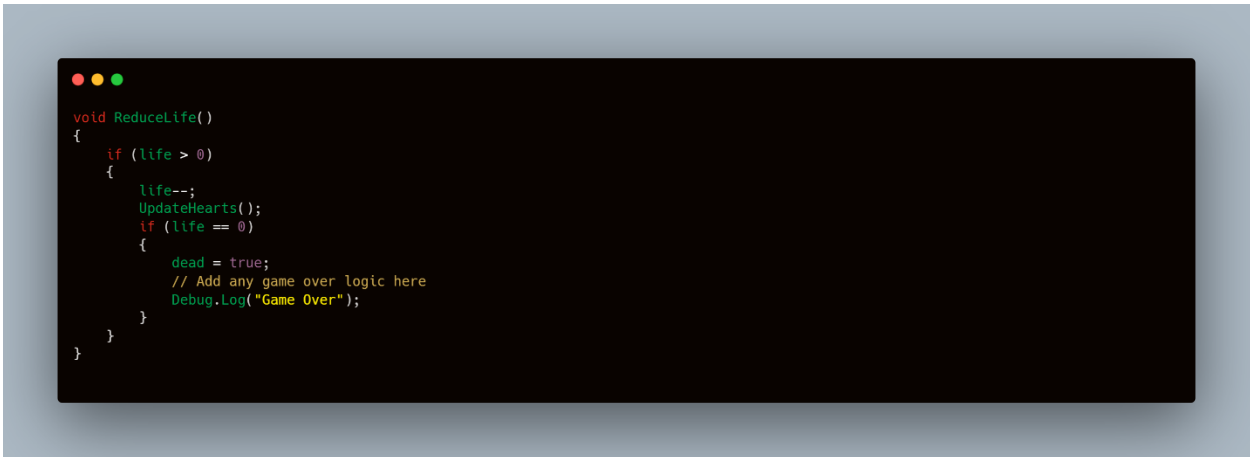
UpdateHearts Method

```
void UpdateHearts()
{
    for (int i = 0; i < hearts.Length; i++)
    {
        hearts[i].SetActive(i < life);
    }
}
```

Figure 6.87-UpdateHearts Method

- Updates the active state of heart GameObjects based on the current life value.

ReduceLife Method

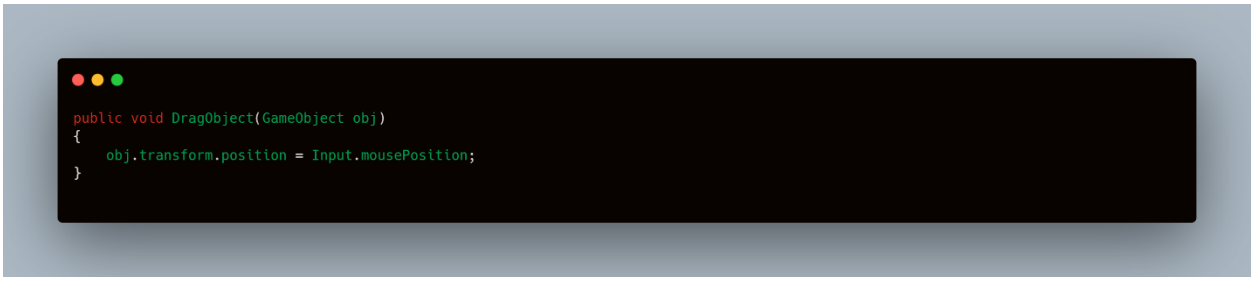


```
void ReduceLife()
{
    if (life > 0)
    {
        life--;
        UpdateHearts();
        if (life == 0)
        {
            dead = true;
            // Add any game over logic here
            Debug.Log("Game Over");
        }
    }
}
```

Figure 6.88-ReduceLife Method

- Reduces the player's life by one, updates the hearts, and checks if the player is out of life.

DragObject Method



```
public void DragObject(GameObject obj)
{
    obj.transform.position = Input.mousePosition;
}
```

Figure 6.89-DragObject Method

- Sets the position of the dragged object to the current mouse position.

DropObject Method

```
public void DropObject(GameObject obj, GameObject target)
{
    float Distance = Vector3.Distance(obj.transform.position, target.transform.position);
    if (Distance < 100)
    {
        obj.transform.position = target.transform.position;
        source.clip = correct;
        source.Play();

        score += 1000;
        scoreText.text = score.ToString();
        Debug.Log("Correct! Score: " + score);
    }
    else
    {
        obj.transform.position = initialPositions[obj];
        source.clip = incorrect;
        source.Play();

        ReduceLife();
        Debug.Log("Incorrect! Life: " + life);
    }
}
```

Figure 6.90-DropObject Method

- Checks if the dropped object is within a certain distance (100 units) of the target object.
- If within range, the object snaps to the target's position, plays the correct sound, and updates the score.
- If not, the object returns to its initial position, plays the incorrect sound, and reduces the player's life.

Drag and Drop Methods for Specific Objects

```
public void Dragptolemy() { DragObject(ptolemy); }
public void Dragalexander() { DragObject(alexander); }
public void Dragisis() { DragObject(isis); }
public void Dragjuliuscaesar() { DragObject(juliuscaesar); }
public void Dragpompey() { DragObject(pompey); }
public void Dragptolemyxiii() { DragObject(ptolemyxiii); }

public void Dropptolemy() { DropObject(ptolemy, circle2); }
public void Dropalexander() { DropObject(alexander, circle5); }
public void Dropisis() { DropObject(isis, circle1); }
public void Dropjuliuscaesar() { DropObject(juliuscaesar, circle3); }
public void Droppompey() { DropObject(pompey, circle5); }
public void Dropptolemyxiii() { DropObject(ptolemyxiii, circle4); }
```

Figure 6.91- Drag and Drop Methods

- These methods provide specific drag and drop functionality for each draggable object and its corresponding target.

6.4 Backend

Understanding the backend implementation of a software project is crucial for appreciating its overall architecture and functionality. In this context, we will delve into the backend aspects of the "Off Mind" project. This project employs the REST API architecture pattern and is constructed using the Laravel framework, renowned for its elegance and robustness in web development. In the following sections, we will guide you through the login process, highlighting key elements and providing a comprehensive overview of how the backend is structured and operates.

This segment of code handles login requests by validating user-entered data, implementing safeguards against malicious login attempts, and delivering error messages to facilitate troubleshooting.

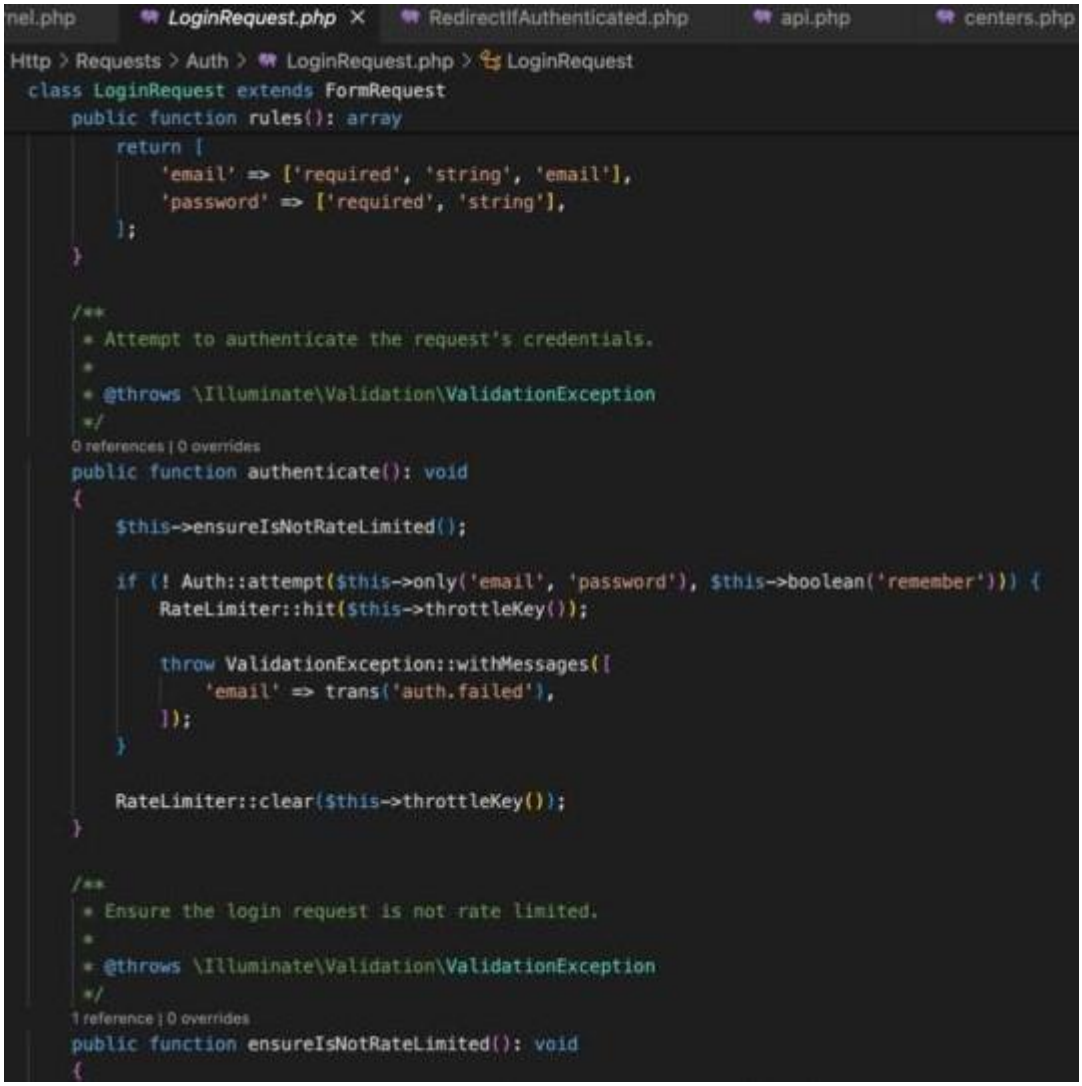
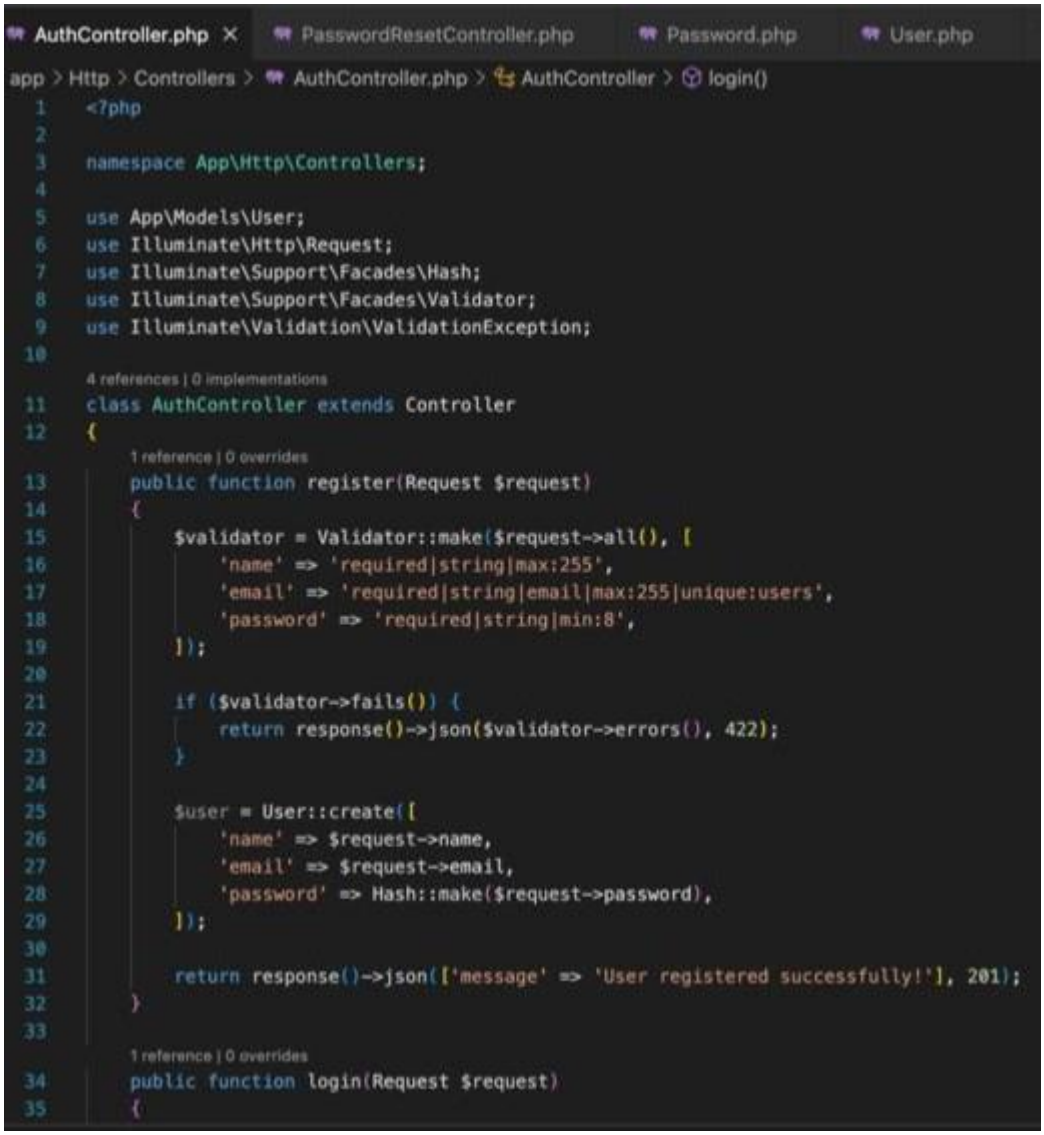


Figure 6.92- LoginReq

Authentication

Authentication plays a critical role in this application by verifying user identities and securing access to its features and data. It ensures that only authorized individuals can interact with the application, thereby safeguarding sensitive information and upholding overall security and integrity

The code manages registration requests by rigorously validating entered data, effectively handling any



encountered errors, securely creating new user records in the database, and ultimately issuing a clear success response upon successful registration completion.

Figure 6.94- Center's controller

Chat controller

In this application, the Chat controller assumes a pivotal role as the primary interface for managing real-time communication functionalities. It serves to facilitate seamless interaction between users, ensuring efficient message handling, notification management, and enhancing overall user engagement and experience.

The code snippet validates the received username against the current username, checks for the existence of the receiving user in the database, and either returns an error if the user is not found or initiates a new conversation if no previous interaction history is present between the users.

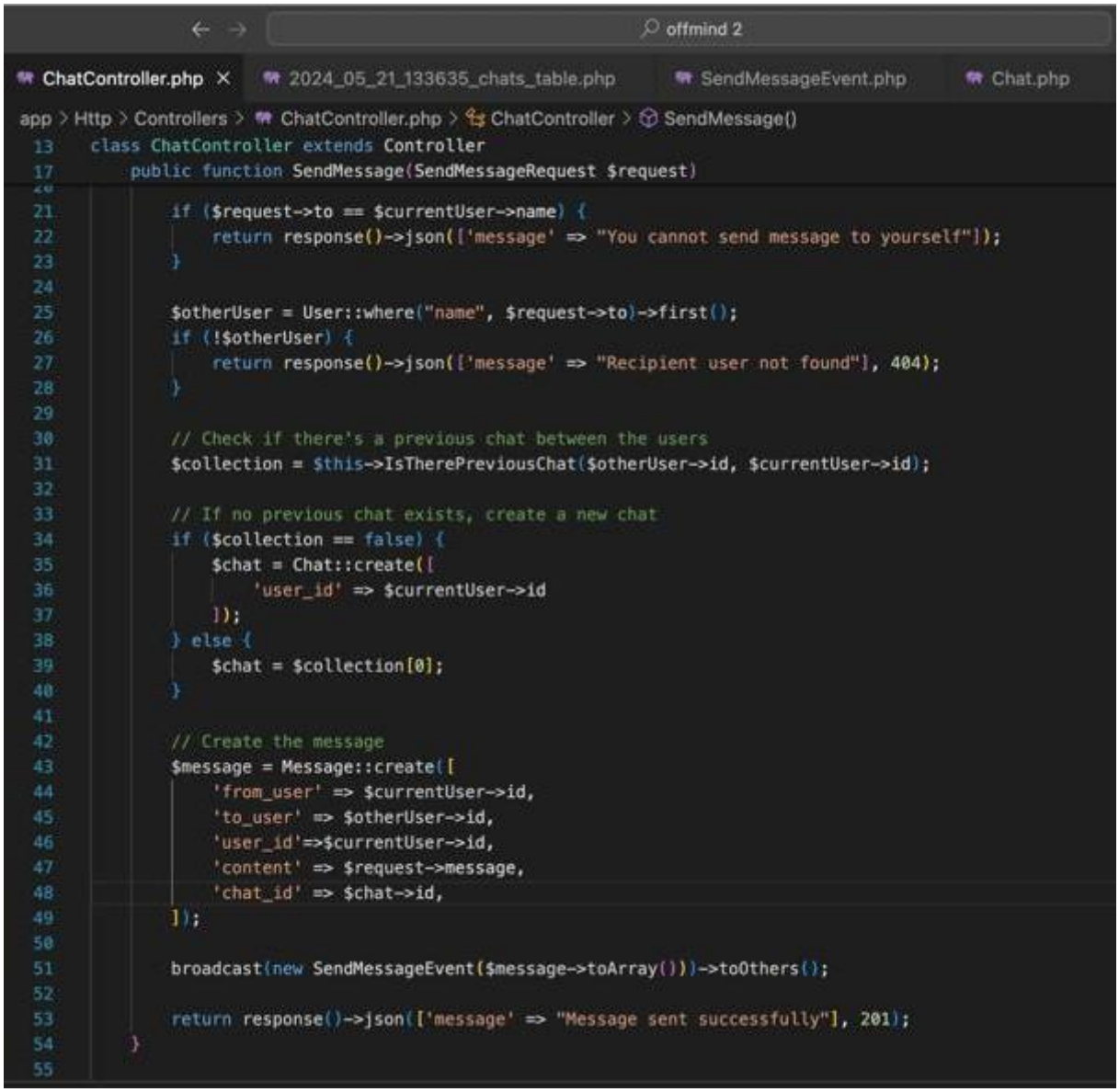


Figure 6.94- Chat's controller

Doctors' controller

The Doctors' controller is integral to this application, acting as the central component for orchestrating critical functionalities related to medical professionals. It enables efficient management of interactions between users and doctors, facilitating seamless scheduling, patient care, and adherence to medical protocols essential for maintaining high standards of healthcare delivery.

The code snippet starts by validating incoming request data against Laravel's rules to ensure all necessary fields are present and correct. Once validated, it creates a new Doctor model instance, representing a doctor's information in the database. The function then assigns values from the validated data to the model's attributes, such as the doctor's name, email, and phone number. Finally, the function saves the Doctor model to the database, creating a new record in the doctor's table.

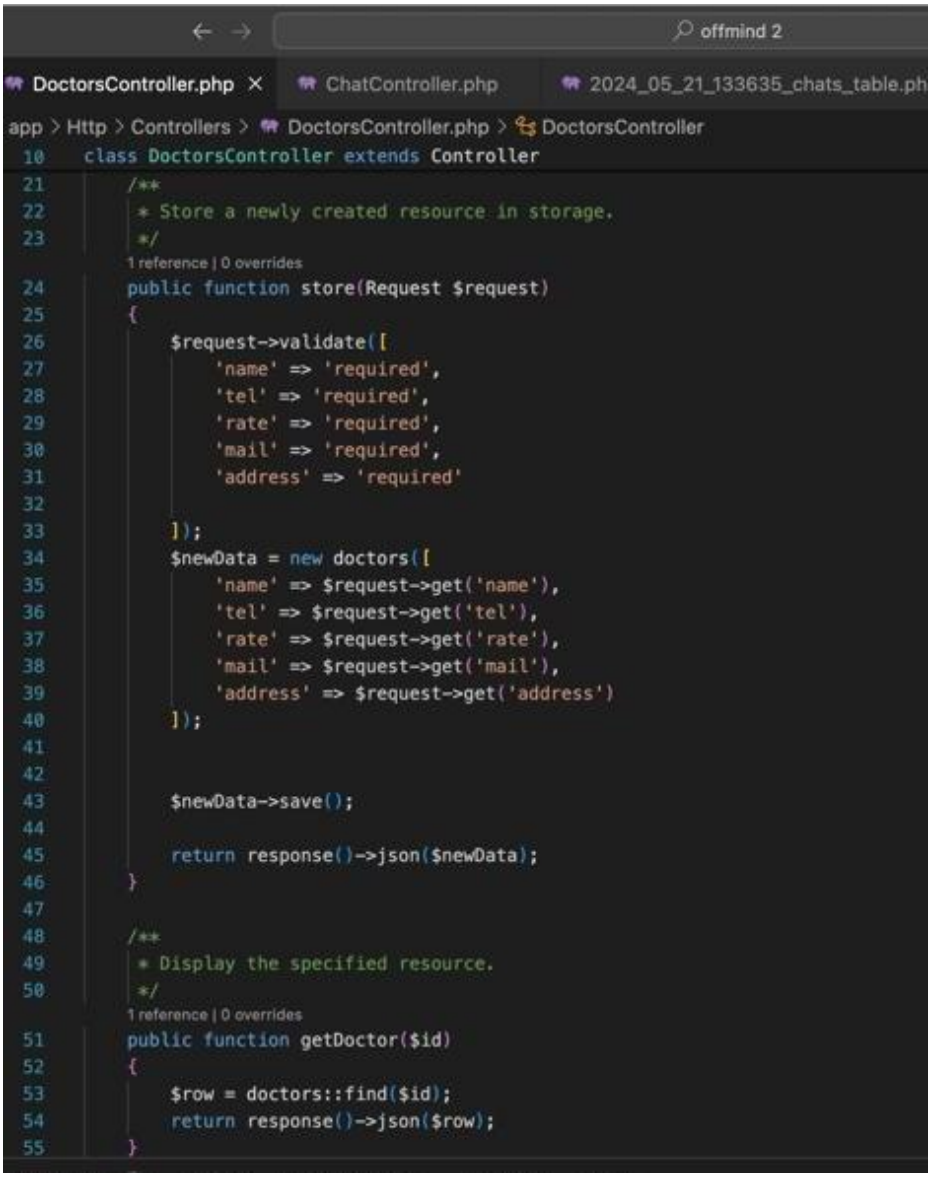


Figure 6.95- Doctor's controller

Diagnosis controller:

The DiagnosisController is essential in this app, overseeing the creation, retrieval, updating, and deletion of diagnostic information. It ensures the precise handling and storage of diagnosis data, which is vital for maintaining comprehensive patient records and facilitating informed medical decision-making.

The function begins by validating the request data against Laravel's rules to ensure all required fields are present and correctly formatted. Upon successful validation, it proceeds to instantiate a new diagnosis model, representing a diagnosis record within the database. The function meticulously assigns values from the validated request data to specific attributes of the diagnosis model, such as the patient's ID, diagnosis date, and description. Finally, the function saves the populated diagnosis model to the database, effectively adding a new entry to the diagnoses table. This process ensures accurate storage and management of diagnostic information within the application.

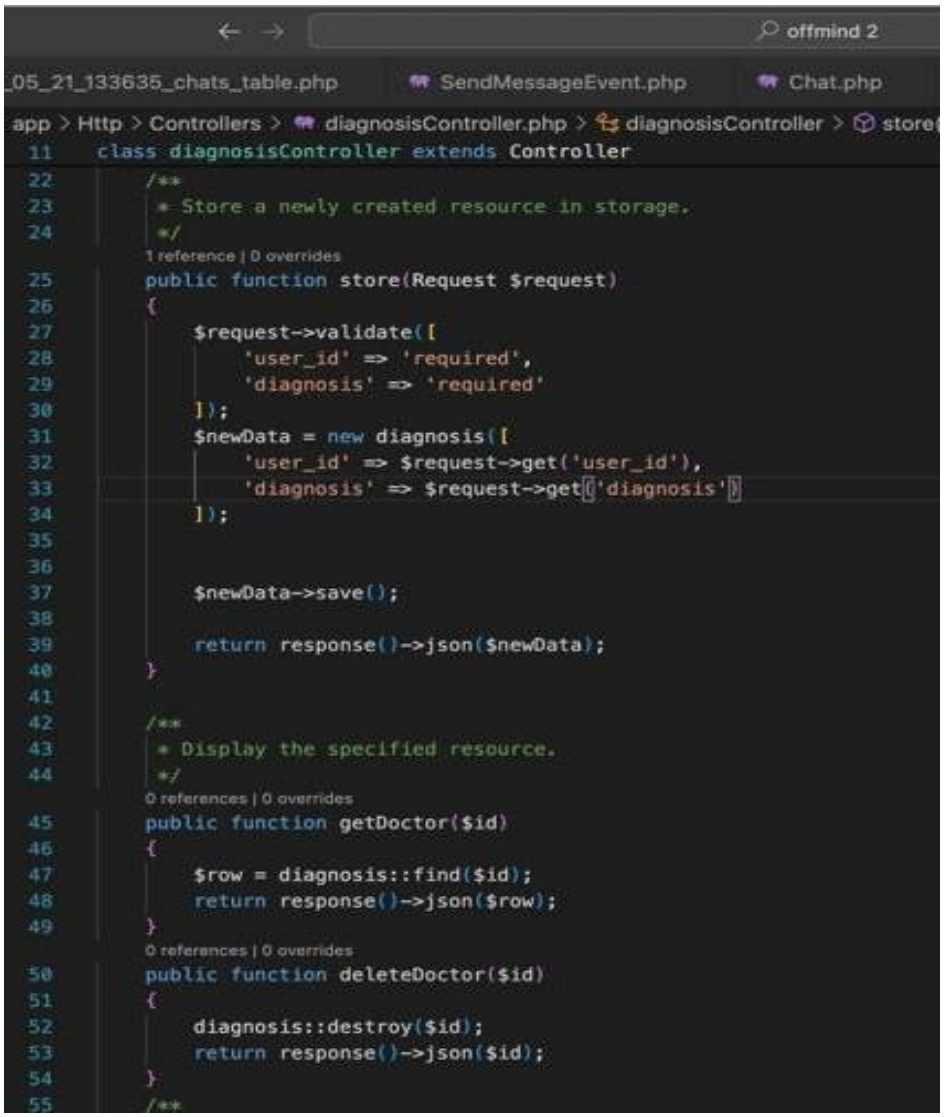


Figure 6.96- Message's controller

Messaging function for chat:

The Messaging function in this app plays a pivotal role in facilitating real-time communication between users. It enables seamless exchange of messages, fostering efficient collaboration and prompt information sharing. This feature enhances user engagement and supports dynamic interaction, crucial for maintaining connectivity and enhancing user experience within the application.

The function acts as a data access layer dedicated to retrieving and filtering previous messages exchanged between two users, adhering to specified criteria. It leverages Laravel's Eloquent ORM and query builder capabilities to interface with the database, ensuring optimal efficiency in data retrieval operations.

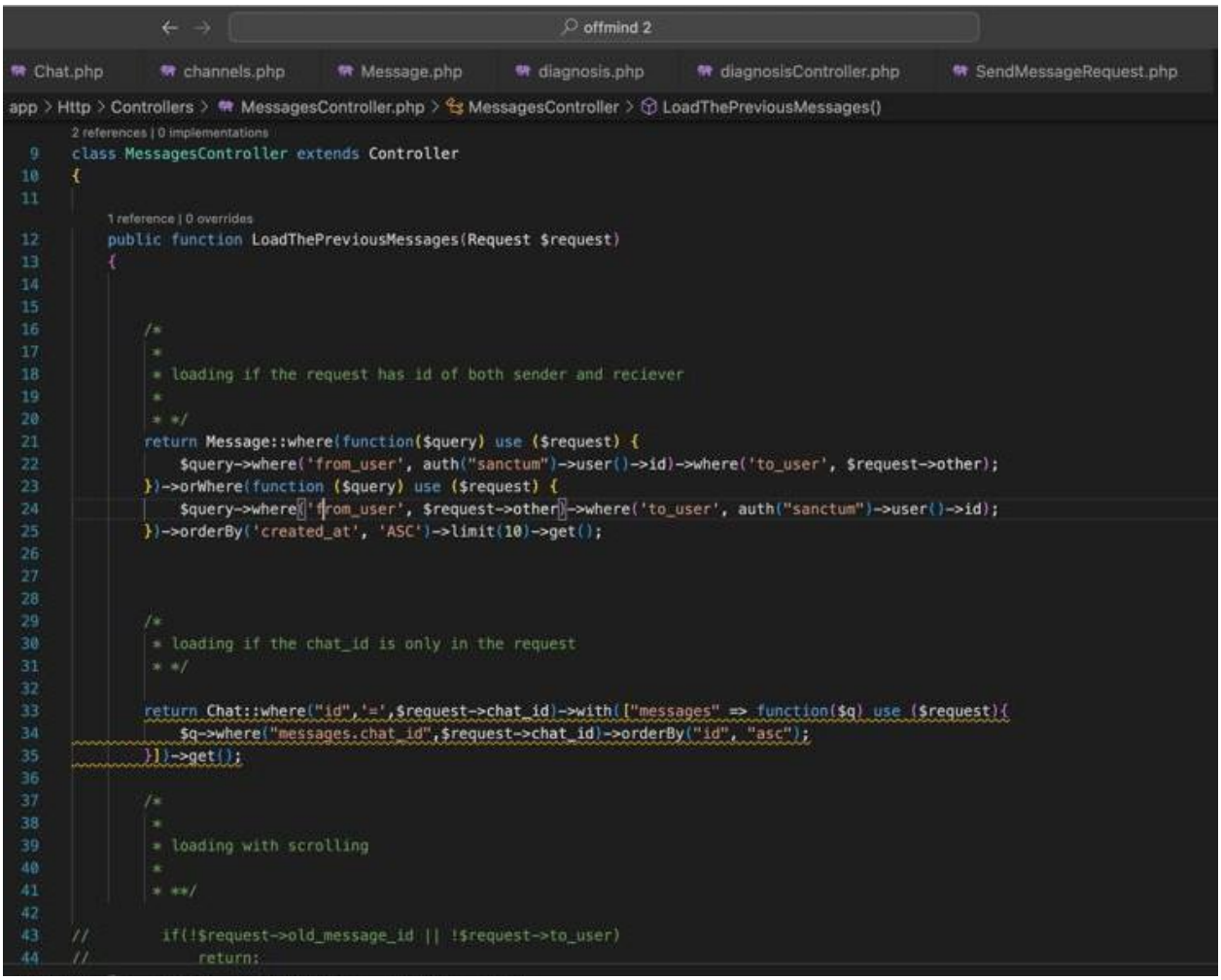


Figure 6.97- Messaging function

Trust Proxies:

In this application, Trust Proxies are essential for securely managing requests from proxy servers, ensuring accurate identification of client IP addresses. This enhances communication security and maintains the integrity of user interactions within the chat environment.

The Send Message function manages the process of sending messages, beginning with recipient validation, followed by the creation and storage of message objects in the database. It effectively utilizes Laravel's Eloquent ORM for seamless database interaction and integrates with the authentication system to ensure secure message handling.

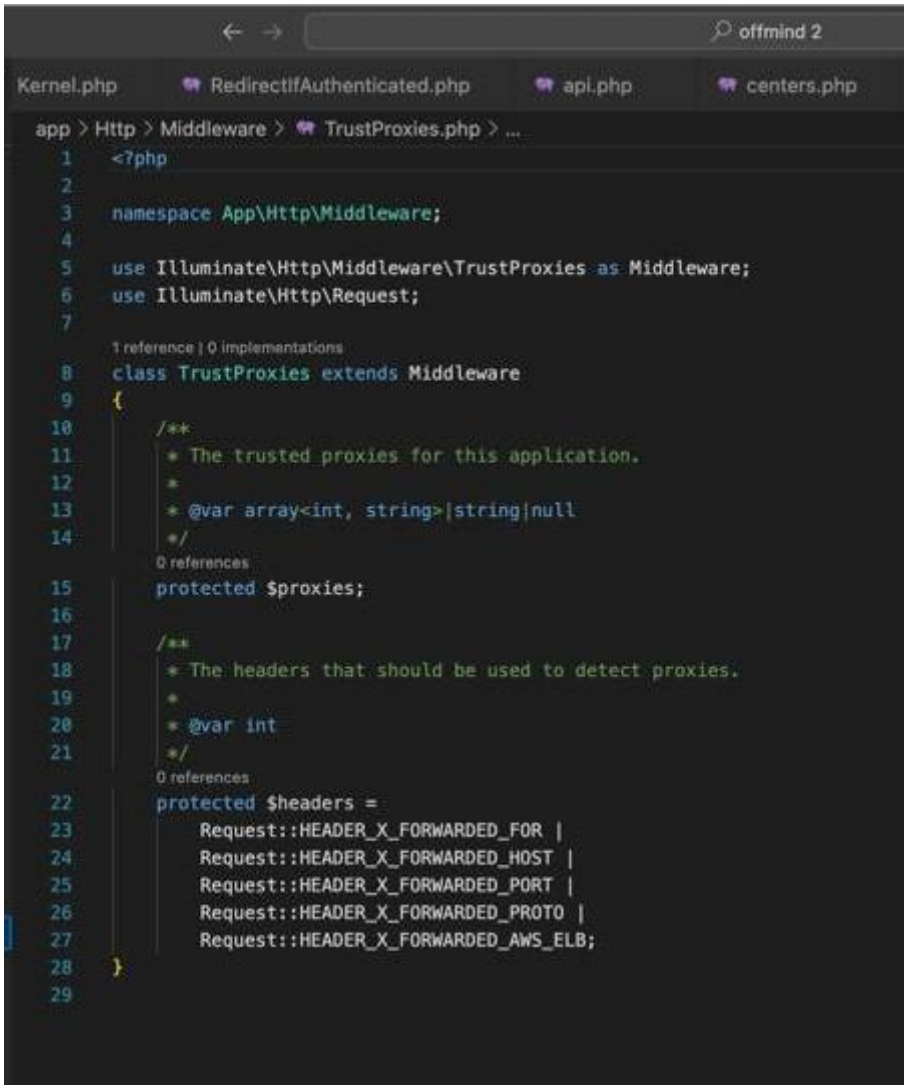


Figure 6.98- Trust Proxies

6.5 Mobile Application

Introduction

The OFFMIND App is an all-encompassing mobile application designed to empower parents in providing exceptional care for their children. Developed using Flutter, this app incorporates a wide array of features to enhance the OFFMIND experience. This document serves as an introduction to the app, highlighting the implementation of Clean Architecture and the utilization of design patterns in its development. The app encompasses an engaging home page featuring expert advice and recommended doctors, an activity tracker to promote child development, a diagnostic module for health assessment, and a customizable settings page. Furthermore, the app seamlessly connects to a database and leverages a machine learning model through a REST API.

MVVM

Similar to Clean Architecture, MVVM (Model-View-ViewModel) serves as a design pattern that structures the OFFMIND App, emphasizing a clear separation of concerns between the user interface and the underlying business logic. This separation promotes modularity, improving the application's testability and maintainability. Let's delve into the three layers of MVVM:

- Model Layer: This layer represents the core data and business logic of the application. It houses the data objects and the rules that govern them, independent of any UI considerations. The Model layer can be implemented without relying on UI components.
- View Layer: This layer is solely responsible for presenting the user interface. It consists of the visual elements that users interact with, but it doesn't contain any business logic or code for data access. The View layer passively displays data provided by the ViewModel and simply forwards user interactions to the ViewModel.
- ViewModel Layer: The ViewModel acts as the bridge between the Model and the View. It prepares data retrieved from the Model layer into a format suitable for presentation in the View. It also handles user interactions received from the View and updates the Model layer accordingly. The ViewModel itself steers clear of any UI code.

We will now explore the Mobile Application Aspects of off Mind. Flutter framework was used to build the Mobile Application with Dart Programing Language

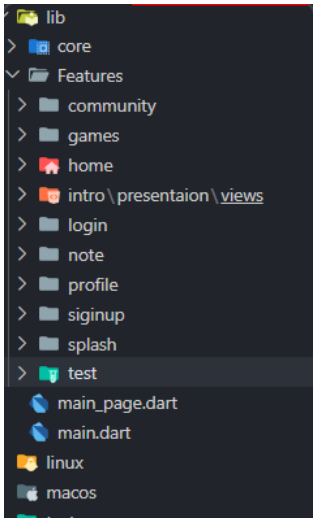


Figure 6.99- Mobile Application Aspects

We use core folder to put the needed function to help with the implementation.

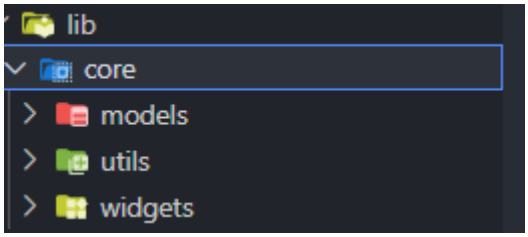


Figure 6.99- needed function

The model folder contains user model dart that have name, email, id:

```dart
class User {
  final String uid;
  final String email;
  final String name;

  User({
    required this.uid,
    required this.email,
    required this.name,
  });

  factory User.fromJson(Map<String, dynamic> json) {
    return User(
      email: json['email'],
      uid: json['uid'],
      name: json['name']
    );
  }

  Map<String, dynamic> toJson() {
    return {
      "email": email,
      "uid": uid,
      "name": name,
    };
  }
}
```

Figure 6.100- model folder

Let's go to the feature folder and see the first feature is community it's like a chat app

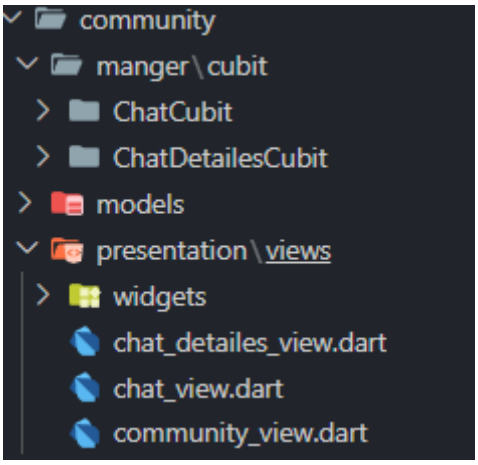


Figure 6.101-feature folder

Let's go to the manger folder: to the chat cubit. To get users form the firebase database and user details from firebase fire store

```dart
part 'states.dart';

class ChatsCubit extends Cubit<ChatsStates> {
  ChatsCubit() : super(ChatsInit());

  List<User.User> users = [];

  Future<void> getUsers() async {
    emit(ChatsLoading());
    try {
      final result = await FirebaseFirestore.instance.collection('users').get();
      for (var i in result.docs) {
        if (i.data()['uid'] == FirebaseAuth.instance.currentUser?.uid) {
          continue;
        }
        users.add(User.User.fromJson(i.data()));
      }
    } catch (e) {
      // ignore: avoid_print
      print(e);
    }
    emit(ChatsInit());
  }
```

Figure 6.102-manger folder

Chat State and the standard cubit state for all the cubit functionality

```dart
abstract class ChatsStates {}

class ChatsInit extends ChatsStates {}

class ChatsLoading extends ChatsStates {}
```

Figure 6.103-standard cubit state

Chat Details cubit to get messages from the firebase database

```dart
part 'states.dart';

class ChatDetailsCubit extends Cubit<ChatDetailsStates> {
  ChatDetailsCubit({required this.user}) : super(ChatDetailsInit());

  final User.User user;

  List<Message> messages = [];
  TextEditingController textEditingController = TextEditingController();
  StreamSubscription? messagesStream;

  Future<void> getMessages() async {
    emit(ChatDetailsLoading());
    final uid = FirebaseAuth.instance.currentUser!.uid;
    messagesStream = FirebaseFirestore.instance
        .collection('chats')
        .doc(uid)
        .collection(user.uid)
        .orderBy('time', descending: true)
        .snapshots()
        .listen((event) {
          messages.clear();
      for (var i in event.docs) {
        messages.add(Message.fromJson(i.data()));
      }
      emit(ChatDetailsInit());
    });
  }

  Future<void> sendMessage() async {
    final text = textEditingController.text;
    // ignore: prefer_is_empty
    if (text.trim().isEmpty || text.trim().length == 0) {
      return;
    }
    textEditingController.clear();
    final uid = FirebaseAuth.instance.currentUser!.uid;
    final message = Message(
      uid: uid,
      message: text,
      time: Timestamp.now(),
    ); // Message
    await FirebaseFirestore.instance
        .collection('chats')
        .doc(uid)
        .collection(user.uid)
        .doc()
        .set(message.toJson());
    await FirebaseFirestore.instance
        .collection('chats')
        .doc(user.uid)
        .collection(uid)
        .doc()
        .set(message.toJson());
    emit(ChatDetailsInit());
  }

  @override
  Future<void> close() {
    textEditingController.dispose();
    messagesStream?.cancel();
    return super.close();
  }
```

Figure 6.103-Chat Details cubit

Let's go to the models. To the message model that have id message and time

```dart
class Message {
  final String uid;
  final String message;
  final Timestamp time;

  Message({
    required this.uid,
    required this.message,
    required this.time,
  });

  factory Message.fromJson(Map<String, dynamic> json) {
    return Message(
      uid: json['uid'],
      message: json['message'],
      time: json['time'],
    );
  }

  Map<String, dynamic> toJson() {
    return {
      "uid": uid,
      "message": message,
      "time": time,
    };
```

Figure 6.104-message model

The implementation of the Chat view

```dart
@override
Widget build(BuildContext context) {
  return BlocProvider(
    create: (context) => ChatsCubit()..getUsers(),
    child: Scaffold(
      backgroundColor: AppColors.white,
      appBar: AppBar(
        backgroundColor: AppColors.white,
        title: const AppText(
          title: 'Online Doctors',
          fontSize: 20,
          fontWeight: FontWeight.bold,
        ), // AppText
        centerTitle: true,
      ), // AppBar
      body: BlocBuilder<ChatsCubit, ChatsStates>(
        builder: (context, state) {
          if (state is ChatsLoading) {
            return const Center(
              child: AppLoadingIndicator(),
            ); // Center
          }
          final users = BlocProvider.of<ChatsCubit>(context).users;
          return Column(
            children: [
              Padding(
                padding: const EdgeInsets.fromLTRB(0, 10, 0, 20),
                child: Row(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    const OnlineAvater(),
                    SizedBox(
                      width: 15.width,
                    ), // SizedBox
                    const OnlineAvater(),
                    SizedBox(
                      width: 15.width,
                    ), // SizedBox
                    const OnlineAvater(),
                  ],
```

Figure 6.104-Chat view

The complete of the chat main implementation

```
), // Padding
Expanded(
  child: ListView.separated(
    padding: const EdgeInsets.all(16),
    itemCount: users.length,
    itemBuilder: (context, index) {
      return GestureDetector(
        onTap: () => RouteUtils.push(
          ChatDetailsView(
            user: users[index],
          ), // ChatDetailsView
        ),
        child: Container(
          height: 100.height,
          width: double.infinity,
          decoration: BoxDecoration(
            color: AppColors.white,
            borderRadius: BorderRadius.circular(25),
            boxShadow: [
              BoxShadow(
                color: Colors.grey.withOpacity(0.5),
                spreadRadius: 1,
                blurRadius: 1,
              ), // BoxShadow
            ],
          ), // BoxDecoration
          child: Padding(
            padding: const EdgeInsets.all(10),
            child: Row(
              mainAxisAlignment: MainAxisAlignment.spaceAround,
              children: [
                const CircleAvatar(
                  radius: 38.0,
                  backgroundImage: AssetImage(
                    Constants.defultProfileImage), // AssetImage
                  child: Align(
                    alignment: Alignment.bottomRight,
                  ), // Align
                ), // CircleAvatar
                Column(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    AppText(
                      title: users[index].name,
                      fontSize: 19,
                      fontWeight: FontWeight.bold,
                    ), // AppText
                    SizedBox(
                      height: 5.height,
                    ), // SizedBox
                    RatingBarIndicator(
                      rating: 5,
                      itemBuilder: (context, index) => const Icon(
                        Icons.star,
                        color: AppColors.yellow,
                      ), // Icon
                      itemCount: 5,
                      itemSize: 25.0,
                      direction: Axis.horizontal,
                    ), // RatingBarIndicator
                  ],
                ), // Column
                Column(
                  mainAxisAlignment: MainAxisAlignment.start,
                  children: [
                    Container(
                      height: 40.height,
                      width: 40.width,
                      decoration: BoxDecoration(
                          color: AppColors.gray.withOpacity(0.4),
                          borderRadius: BorderRadius.circular(50)), // BoxDecorati
                      child: const Icon(
                        Icons.favorite,
                        color: AppColors.blue,
                      ), // Icon
                    ), // Container
                  ],
                ), // Column
              ],
            ), // Row
          ), // Padding
        ), // Container
      ); // GestureDetector
    },
    separatorBuilder: (context, index) => SizedBox(height: 20.height),
  ), // ListView.separated
), // Expanded
```

Figure 6.105-Chat main

The implementation of the Chat detail's view

```dart
class ChatDetailsView extends StatelessWidget {
  const ChatDetailsView({
    Key? key,
    required this.user,
  }) : super(key: key);

  final User.User user;

  @override
  Widget build(BuildContext context) {
    return BlocProvider(
      create: (context) => ChatDetailsCubit(user: user)..getMessages(),
      child: Scaffold(
        backgroundColor: AppColors.blue,
        body: BlocBuilder<ChatDetailsCubit, ChatDetailsStates>(
          builder: (context, state) {
            if (state is ChatDetailsLoading) {
              return const Center(
                child: AppLoadingIndicator(),
              ); // Center
            }
            final cubit = BlocProvider.of<ChatDetailsCubit>(context);
            final messages = cubit.messages;
            return Column(
              children: [
                SizedBox(
                  height: 50.height,
                ), // SizedBox
                Row(
                  children: [
                    IconButton(
                        onPressed: () => RouteUtils.pop(),
                        icon: const Icon(
                          Icons.arrow_back_ios_new,
                          size: 30,
                        )), // Icon // IconButton
                    const CircleAvatar(
                      radius: 38.0,
                      backgroundImage:  AssetImage(
                                        Constants.defultProfileImage), // AssetImage
                      child: Align(
                        alignment: Alignment.bottomRight,
                      ), // Align
                    ), // CircleAvatar
                    SizedBox(
                      width: 10.width,
                    ), // SizedBox
                    Column(
                      crossAxisAlignment: CrossAxisAlignment.start,
                      children: [
                        AppText(
                          title: user.name,
                          color: AppColors.white,
                          fontSize: 14,
                          fontWeight: FontWeight.w600,
                        ), // AppText
                        const AppText(
                          title: 'Online',
                          fontSize: 12,
                          color: AppColors.white,
                          fontWeight: FontWeight.w600,
                        ) // AppText
```

Figure 6.106-Chat details

The complete of the chat details implementation

```
Expanded(
  child: Container(
    width: double.infinity,
    decoration: const BoxDecoration(
        color: AppColors.white,
        borderRadius: BorderRadius.only(
            topLeft: Radius.circular(40),
            topRight: Radius.circular(40))), // BorderRadius.only // BoxDecoration
    child: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        children: [
          const AppText(title: 'Today'),
          Expanded(
            child: ListView.builder(
              reverse: true,
              itemCount: messages.length,
              itemBuilder: (context, index) {
                final message = messages[index];
                final isMe = message.uid ==
                    FirebaseAuth.instance.currentUser?.uid;
                return Row(
                  mainAxisAlignment: isMe
                      ? MainAxisAlignment.end
                      : MainAxisAlignment.start,
                  children: [
                    UnconstrainedBox(
                      child: ConstrainedBox(
                        constraints: const BoxConstraints(
                          minWidth: 10,
                          maxWidth: 300,
                        ), // BoxConstraints
                        child: Container(
                          margin:
                              const EdgeInsets.only(top: 12),
                          padding: const EdgeInsets.all(12),
                          decoration: BoxDecoration(
                            borderRadius:isMe
                                ?const BorderRadius.only(
                              bottomLeft: Radius.circular(18),
                              topLeft: Radius.circular(18),
                              topRight: Radius.circular(18),
                                ) // BorderRadius.only
                               :const BorderRadius.only(
                              bottomRight: Radius.circular(18),
                              topLeft: Radius.circular(18),
                              topRight: Radius.circular(18),
```

Figure 6.107-Chat details cont.

Let’s go to the game feature

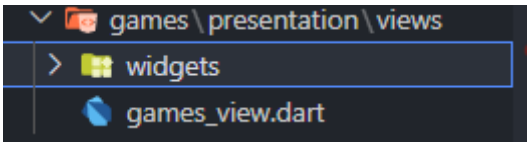


Figure 6.107-game feature

The implementation of the game view

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Padding(
        padding: EdgeInsets.symmetric(horizontal: 20, vertical: 100.height),
        child: GridView.builder(
          gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
            crossAxisCount: 2, // Number of columns in the grid
            crossAxisSpacing: 10, // Spacing between columns
            mainAxisSpacing: 10, // Spacing between rows
          ), // SliverGridDelegateWithFixedCrossAxisCount
          physics: const NeverScrollableScrollPhysics(),
          itemCount: games.length, // Total number of items
          itemBuilder: (BuildContext context, int index) {
            return games[index];
          },
```

Figure 6.108-game view

Let’s go to the home feature

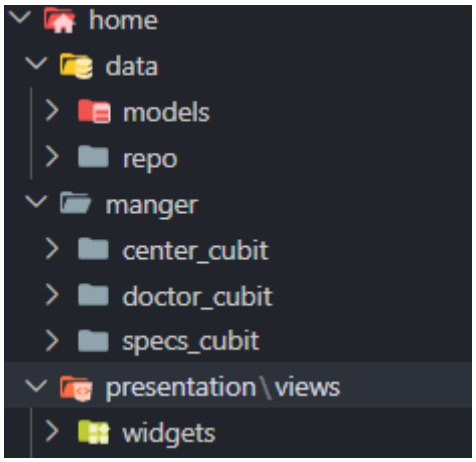


Figure 6.109-home feature

Let’s start with the data folder and go to the models to medical model, to see the structure of the data that have the medical person data like id, name, rate, telephone number, mail, address and title

```dart
class Specs {
  final int id;
  final String name;
  final String tel;
  final int rate;
  final String mail;
  final String address;
  final String title;

  Specs({
    required this.id,
    required this.name,
    required this.tel,
    required this.rate,
    required this.mail,
    required this.address,
    required this.title,
  });

  factory Specs.fromJson(Map<String, dynamic> json) {
    return Specs(
      id: json['id'],
      name: json['name'],
      tel: json['tel'],
      rate: json['rate'] is double ? (json['rate'] as double).toInt() : json['rate'],
      mail: json['mail'],
      address: json['address'],
      title: json['title'],
```

Figure 6.110-data folder

To the repository to call the data for the database and view it in the presentation layer

```
class Repository {

// fetchDoctors
  List<Doctor> doctors = [];
  Future<List<Doctor>> fetchDoctors() async {
    try {
      Response response = await NetworkUtils.get('doc/list');

      if (response.statusCode == 200) {
        for (var i in response.data) {
          final c = Doctor.fromJson(i);
          doctors.add(c);
        }
        Iterable jsonResponse = response.data;
        return jsonResponse.map((doctor) => Doctor.fromJson(doctor)).toList();
      } else {
        throw Exception('Failed to load doctors');
      }
    } catch (e) {
      throw Exception('Failed to load doctors: $e');
    }
  }

  // fetchCenters
  List<MedicalCenter> medicalCenters = [];
  Future<List<MedicalCenter>> fetchCenters() async {
    try {
      Response response =
          await NetworkUtils.get('center/list');

      if (response.statusCode == 200) {
        for (var i in response.data) {
          final c = MedicalCenter.fromJson(i);
          medicalCenters.add(c);
        }
        Iterable jsonResponse = response.data;
        return jsonResponse
            .map((center) => MedicalCenter.fromJson(center))
            .toList();
      } else {
        throw Exception('Failed to load centers');
      }
    } catch (e) {
      throw Exception('Failed to load centers: $e');
    }
  }

  // fetchSpecs
  List<Specs> specss = [];
  Future<List<Specs>> fetchSpecs() async {
    try {
      Response response = await NetworkUtils.get('specs/list');


      if (response.statusCode == 200) {
        for (var i in response.data) {
          final c = Specs.fromJson(i);
          specss.add(c);
        }
        Iterable jsonResponse = response.data;
        return jsonResponse.map((specs) => Specs.fromJson(specs)).toList();
      } else {
        throw Exception('Failed to load SPeces');
      }
    } catch (e) {
      throw Exception('Failed to load SPecs: $e');
    }
```

Figure 6.110-call the data

Medical details view implantation

```dart
class DoctorsDetailesView extends StatefulWidget {
  final int id;
  final String doctorImage;
  const DoctorsDetailesView({
    super.key,
    required this.id,
    required this.doctorImage,
  });

  @override
  State<DoctorsDetailesView> createState() => _DoctorsDetailesViewState();
}

class _DoctorsDetailesViewState extends State<DoctorsDetailesView> {
  final Repository repository = Repository();

  bool isLoading = true;
  Doctor? doctor;
@override
  void initState() {
    super.initState();
    fetchSpecsDetails();
  }

  void fetchSpecsDetails() async {
    try {
      final response = await NetworkUtils.get('doc/get/${widget.id}');
      setState(() {
        doctor = Doctor.fromJson(response.data);
        isLoading = false;
      });
    } catch (e) {
      setState(() {
        isLoading = false;
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      extendBodyBehindAppBar: true,
      appBar: AppBar(
        leading: IconButton(
            onPressed: () {
              RouteUtils.pop();
            },
            icon: const Icon(Icons.arrow_back_ios_new,color: AppColors.darkGray,)), // IconButton
        title: const AppText(
          title: 'Doctor details',
          fontSize: 23,
          fontWeight: FontWeight.bold,
        ), // AppText
        centerTitle: true,
        elevation: 0,
        backgroundColor: Colors.transparent,
      ), // AppBar
      body: isLoading
          ? const Center(child: AppLoadingIndicator())
          : doctor == null
              ? const Center(child: Text('Error loading data'))
              : Stack(
                  fit: StackFit.expand,
                  children: <Widget>[
                    Image.asset(
                'Assets/Images/background1.png', // Replace 'background.jpg' with your image asset pat
                fit: BoxFit.cover,
              ), // Image.asset
```

Figure 6.111-Medical details

Appointment view implementation to make the reservation with the medical professional

```dart
class SetApointmentViewBody extends StatefulWidget {
  const SetApointmentViewBody({super.key});

  @override
  State<SetApointmentViewBody> createState() => _SetApointmentViewBodyState();
}

class _SetApointmentViewBodyState extends State<SetApointmentViewBody> {
  DateTime today = DateTime.now();
  void _onSelectedDate(DateTime day, DateTime foucasDay) {
    setState(() {
      today = day;
    });
  }

  bool _isSelected = false;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      extendBodyBehindAppBar: true,
      appBar: AppBar(
        leading: IconButton(
            onPressed: () {
             RouteUtils.pop();
            },
            icon: const Icon(Icons.arrow_back_ios_new,color: AppColors.darkGray,)), // IconButton
        title: const AppText(
          title: 'Select date and time',
          fontSize: 23,
          fontWeight: FontWeight.bold,
        ), // AppText
        centerTitle: true,
        elevation: 0,
        backgroundColor: Colors.transparent,
      ), // AppBar
      body: Stack(
        fit: StackFit.expand,
        children: <Widget>[
          Image.asset(
            'Assets/Images/background1.png',
            fit: BoxFit.cover,
          ), // Image.asset
          Padding(
            padding: EdgeInsets.only(top: 100.height),
```

Figure 6.112-Appointment view

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: AppColors.white,
      appBar: const HomeAppBar(),
      body: SingleChildScrollView(
        physics: const BouncingScrollPhysics(),
        child: Padding(
          padding: EdgeInsets.symmetric(
            horizontal: 20.width,
          ), // EdgeInsets.symmetric
          child: Column(
            children: [
              SizedBox(
                height: 10.height,
              ), // SizedBox
              const Row(
                children: [
                  AppText(
                    title: 'Note',
                    fontSize: 32,
                    fontWeight: FontWeight.bold,
                  ), // AppText
                ],
              ), // Row
              const NoteHome(),
              SeeMore(
                tittle: 'Doctors',
                onTap: () => RouteUtils.push(MoreDoctorsView()),
              ), // SeeMore
              // doctors//
              BlocBuilder<DoctorCubit, DoctorState>(
                builder: (context, state) {
                  if (state is DoctorLoading) {
                    return const Center(child: AppLoadingIndicator());
                  } else if (state is DoctorError) {
                    return Center(child: Text(state.message));
                  } else if (state is DoctorLoaded) {
                    final doctors = state.doctors;
                    return GridView.builder(
                      itemCount: 2,
                      gridDelegate:
                          const SliverGridDelegateWithFixedCrossAxisCount(
                        crossAxisCount: 2,
                        childAspectRatio: 7 / 10,
                        crossAxisSpacing: 1,
                        mainAxisSpacing: 1,
                      ), // SliverGridDelegateWithFixedCrossAxisCount
                      shrinkWrap: true,
                      physics: const BouncingScrollPhysics(),
                      padding: EdgeInsets.zero,
                      itemBuilder: (BuildContext context, int index) {
                        return DoctorCart(
                          doctor: doctors[index],
                          doctorImg: doctorImages[index],
                        ); // DoctorCart
                      },
                    ); // GridView.builder
                  }
                  return Container();
                },
              ), // BlocBuilder
```

And finally, the home view implementation

Figure 6.113-home view

Let’s go to the Signup feature

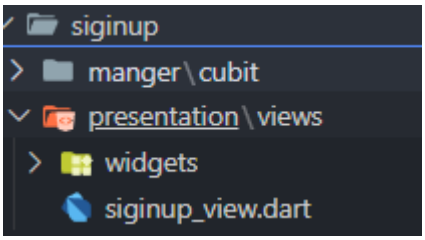


Figure 6.114-Signup feature

In the manger folder there is the signup cubit is responsible to send new users data to the database

```dart
part 'singup_states.dart';

class RegisterCubit extends Cubit<RegisterStates> {
  RegisterCubit() : super(RegisterInit());
    final formKey = GlobalKey<FormState>();
  String? email, password,name;

  Future<void> register() async {
       formKey.currentState!.save();
    if (!formKey.currentState!.validate()) {
      return;
    }
    emit(RegisterLoading());
    try {
      final credential = await FirebaseAuth.instance.createUserWithEmailAndPassword(
        email: email!,
        password: password!,
      );
      if (credential.user?.uid != null) {
        await setUserData(credential.user!.uid);
        await CachingUtils.cacheEmail(email!);
      await CachingUtils.cacheName(name!);
        RouteUtils.pushAndPopAll(const MainPaage());
        return;
      }
      throw FirebaseAuthException(code: "0", message: null);
    } on FirebaseAuthException catch (e) {
      showSnackBar(e.message ?? "Something went wrong", isError: true);
    }
    emit(RegisterInit());
  }

  Future<void> setUserData(String uid) async {
    await FirebaseFirestore.instance.collection('users').doc(uid).set({
      'email': email,
      'uid': uid,
      'name': name,
    });
```

Figure 6.115-signup cubit

Let’s go to the login feature

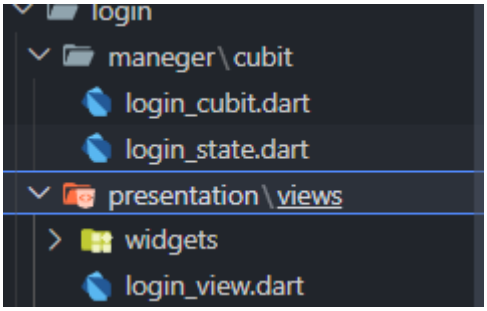


Figure 6.116- login feature

To the manger folder to login cubit to check the user's data and check for authentications

```dart
part 'login_state.dart';

class LoginCubit extends Cubit<LoginStates> {
  LoginCubit() : super(LoginInitial());
  final formKey = GlobalKey<FormState>();
  String? email, password;

  Future<void> login() async {
    formKey.currentState!.save();
    if (!formKey.currentState!.validate()) {
      return;
    }
    emit(LoginLoading());
      try {
      final credential = await FirebaseAuth.instance.signInWithEmailAndPassword(
          email: email!,
          password: password!,
      );
      if (credential.user?.uid != null) {
        RouteUtils.pushAndPopAll(const MainPaage());
        return;
      }
    } on FirebaseAuthException catch (e) {
      showSnackBar(e.message ?? "Something went wrong", isError: true);
    }
    emit(LoginInitial());
  }
```

Figure 6.117-*login cubit*

The implementation of the login view

```
Row(
  mainAxisAlignment: MainAxisAlignment.end,
  children: [
    InkWell(
     onTap: () => RouteUtils.push(ResetPasswordView()),
     child:  const AppText(
       title: 'forgot password?',
       color: AppColors.darkGray,
     ), // AppText
    ), // InkWell
    SizedBox(
      width: 30.width,
    ), // SizedBox
  ],
), // Row
SizedBox(
  height: 70.height,
), // SizedBox

BlocBuilder<LoginCubit, LoginStates>(
  builder: (context, state) {
    if (state is LoginLoading) {
      return const Center(
        child: AppLoadingIndicator(

        ), // AppLoadingIndicator
      ); // Center
    }
    return CustomButton(
      btnText: 'Sign in',
      onTap: cubit.login,
    ); // CustomButton
  },
), // BlocBuilder

SizedBox(
  height: 30.height,
), // SizedBox
const CustomDivider(
  text: 'or continue with',
), // CustomDivider
SizedBox(
  height: 30.height,
), // SizedBox
const Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    CustomIconContainer(icon: 'Assets/Images/google.png'),
    CustomIconContainer(
        icon: 'Assets/Images/apple-logo 1.png'), // CustomIconContainer
    CustomIconContainer(
        icon: 'Assets/Images/facebook.png'), // CustomIconContainer
  ],
), // Row
SizedBox(
  height: 30.height,
), // SizedBox
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    const AppText(
      title: 'Don\'t have an account?',
      fontWeight: FontWeight.w700,
    ), // AppText
    Column(
      children: [
        InkWell(
          onTap: () {
            RouteUtils.push(const SignUpView());
          },
          child: const AppText(
            title: 'Sign up',
            color: AppColors.blue,
            fontSize: 14,
            fontWeight: FontWeight.bold,
          ), // AppText
        ), // InkWell
```

Figure 6.118-login view

Let’s go to the splash view feature

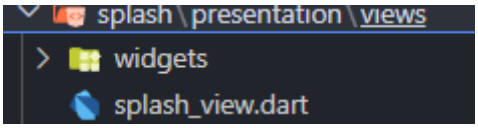


Figure 6.119-splash view feature

The implementation of the splash view

```dart
class SplashViewBody extends StatefulWidget {
  const SplashViewBody({super.key});

  @override
  State<SplashViewBody> createState() => _SplashViewBodyState();
}

class _SplashViewBodyState extends State<SplashViewBody>
    with SingleTickerProviderStateMixin {
  @override
  // void initState() {

  //     super.initState();
  //     SystemChrome.setEnabledSystemUIMode(SystemUiMode.immersive);
  //     Future.delayed(const Duration(seconds: 2), () {

  //       RouteUtils.pushReplacement(const IntroView());
  //     });aaa
  // }
  @override
  void initState() {
    Timer(const Duration(seconds: 2), () {
      final currentUser = FirebaseAuth.instance.currentUser;
      RouteUtils.pushAndPopAll(
          currentUser == null ? const IntroView() : const MainPaage()
      );
    }); // Timer
    super.initState();
  }

  @override
  void dispose() {
    SystemChrome.setEnabledSystemUIMode(SystemUiMode.manual,
        overlays: SystemUiOverlay.values);

    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        decoration: const BoxDecoration(
          image: DecorationImage(
              image: AssetImage('Assets/Images/logo background.png'),
              fit: BoxFit.fill)), // DecorationImage
        ), // BoxDecoration
        child: Row(
          mainAxisAlignment: MainAxisAlignment.end,
          children: [
            Column(
              children: [
                SizedBox(
                  height: 200.height,
                ), // SizedBox
                Image(
                  image: const AssetImage('Assets/Images/logo-01 1.png'),
                  width: 317.width,
                  height: 297,
                ), // Image
                SizedBox(
                  height: 20.height,
                ), // SizedBox
                Row(
                  children: [
                    const CircularProgressIndicator(
                      color: AppColors.blue,
                    ), // CircularProgressIndicator
                    SizedBox(
                      width: 65.width,
                    ), // SizedBox
```

Figure 6.120-implementation of splash

Let’s go to the intro feature

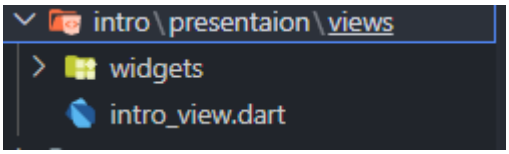


Figure 6.121-intro feature

The into view implementation

```dart
class IntroViewBody extends StatefulWidget {
  const IntroViewBody({super.key});

  @override
  // ignore: library_private_types_in_public_api
  _IntroViewBodyState createState() => _IntroViewBodyState();
}

class _IntroViewBodyState extends State<IntroViewBody> {
  int _currentPage = 0;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: AppColors.white,
      body: Container(
        decoration: const BoxDecoration(
            image: DecorationImage(
                image: AssetImage('Assets/Images/background1.png'),
                fit: BoxFit.cover)), // DecorationImage // BoxDecoration
        child: Column(
          children: [
            CarouselSlider(
              options: CarouselOptions(
                height: 730.height,
                enableInfiniteScroll: false,
                autoPlay: false,
                viewportFraction: 1.0,
                onPageChanged: (index, reason) {
                  setState(() {
                    _currentPage = index;
                  });
                },
              ), // CarouselOptions
              items: pages.map((Widget page) {
                return Builder(
                  builder: (BuildContext context) {
                    return SizedBox(
                      width: MediaQuery.of(context).size.width,
                      child: page,
                    ); // SizedBox
                  },
                ); // Builder
              }).toList(),
            ), // CarouselSlider
            DotsIndicator(
              dotsCount: pages.length,
              position: _currentPage.toDouble(),
              decorator: const DotsDecorator(
                color: Colors.grey,
                activeColor: Colors.black,
              ), // DotsDecorator
            ), // DotsIndicator
```

Figure 6.122-view implementation

Let’s go to the profile feature

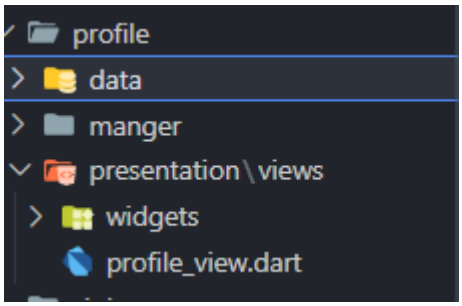


Figure 6.123-profile feature

In the data folder this is a method to get current user data

```dart
class ProfileAuth {
  Future<void> sendPasswordLink(String email) async {
    await FirebaseAuth.instance.sendPasswordResetEmail(email: email);
  }
  Future<DocumentSnapshot> getUserData() async {
  User? user = FirebaseAuth.instance.currentUser;
  if (user != null) {
    DocumentSnapshot userData = await FirebaseFirestore.instance.collection('users').doc(user.uid).get();
    return userData;
  } else {
    throw Exception('No user logged in');
  }
```

Figure 6.124-data folder

And the cubit of the profile doesn’t have a state because it’s a hypered cubit

```dart
class ProfileCubit extends Cubit<File?> {
  ProfileCubit() : super(null) {
    _loadImage();
  }

  Future<void> _loadImage() async {
    String? imagePath = await CachingUtils.getProfileImagePath();
    if (imagePath != null) {
      emit(File(imagePath));
    }
  }

  Future<void> setImage(File image) async {
    await CachingUtils.setProfileImagePath(image.path);
    emit(image);
  }
```

Figure 6.125-profile cubit

The profile view implementation

```dart
class ProfileViewBody extends StatelessWidget {
  ProfileViewBody({
    super.key,
  });

  final ImagePicker _picker = ImagePicker();

  Future<void> _pickImage(BuildContext context) async {
    final pickedFile = await _picker.pickImage(source: ImageSource.gallery);
    if (pickedFile != null) {
      // ignore: use_build_context_synchronously
      context.read<ProfileCubit>().setImage(File(pickedFile.path));
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: AppColors.white,
      extendBodyBehindAppBar: true,
      extendBody: true,
      primary: false,
      body: SingleChildScrollView(
        // physics: BouncingScrollPhysics(),
        child: Column(
          children: [
            Stack(
              // fit: StackFit.loose,
              clipBehavior: Clip.none,
              alignment: Alignment.center,
              children: [
                Container(
                  margin: EdgeInsets.only(bottom: 65.height),
                  width: double.infinity,
                  height: 120.height,
                  decoration: const BoxDecoration(
                      color: AppColors.blue,
                      borderRadius: BorderRadius.only(
                          bottomLeft: Radius.circular(10),
                          bottomRight: Radius.circular(10))), // BorderRadius.only // BoxDecoration
                ), // Container
                BlocBuilder<ProfileCubit, File?>(
                  builder: (context, image) {
                    return image == null
                        ? ProfileImageContainer(
                            image: InkWell(
                              onTap: () => _pickImage(context),
                              child: CircleAvatar(
                                  radius: 60.height,
                                  backgroundColor: AppColors.white,
                                  backgroundImage: const AssetImage(
                                      Constants.defultProfileImage)), // AssetImage // CircleAvatar
                            ), // InkWell
                          ) // ProfileImageContainer
                        : ProfileImageContainer(
                            image: InkWell(
                              onTap: () => _pickImage(context),
                              child: CircleAvatar(
                                  radius: 60.height,
                                  backgroundColor: AppColors.white,
                                  backgroundImage:
                                      // ignore: unnecessary_null_comparison
                                      image != null ? FileImage(image) : null), // CircleAvatar
                            ), // InkWell
                          ); // ProfileImageContainer
                  },
                ), // BlocBuilder
              ],
            ), // Stack
            FutureBuilder<DocumentSnapshot>(
              future: ProfileAuth().getUserData(),
              builder: (context, snapshot) {
                if (snapshot.connectionState == ConnectionState.waiting) {
                  return const Center(child: AppLoadingIndicator());
                }
                if (snapshot.hasError) {
                  return Center(child: Text('Error: ${snapshot.error}'));
                }
                if (!snapshot.hasData || !snapshot.data!.exists) {
                  return const Center(child: Text('User data not found'));
                }
                Map<String, dynamic> data =
                    snapshot.data!.data() as Map<String, dynamic>;
                return Padding(
                  padding: EdgeInsets.symmetric(horizontal: 30.width),
                  child: Column(
                    children: [
                      AppText(
                        title: data['name'] ?? '',
                        fontSize: 20,
                        fontWeight: FontWeight.w600,
                      ), // AppText
                      AppText(
                        title: data['email'] ?? '',
                        fontSize: 17,
                        color: AppColors.darkGray,
                      ), // AppText
```

Figure 6.126-profile view

Let’s go to the test feature

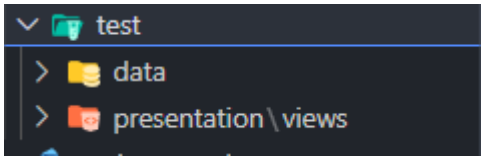


Figure 6.127-test feature

In the data folder this the API Service to handle Ai data request and response

```dart
class TestApiService {
  final Dio _dio = Dio();

  Future<int> textPredict(String gender, String age, String handedness, String iq) async {
    const String url = 'https://mennahmed-fastapi-docker.hf.space/predict';
    final Map<String, String> data = {
      "Gender": gender,
      "Age": age,
      "Handedness": handedness,
      "IQ": iq,
    };

    try {
      final response = await _dio.post(url, data: data);

      if (response.statusCode == 200) {
        return response.data['prediction'];
      } else {
        throw Exception('Failed to load prediction');
      }
    } catch (e) {
      throw Exception('Failed to load prediction');
    }
  }



  // file predict
    Future<int> filePredict(String filePath) async {
    const String url = 'https://off-mind.onrender.com/predict';
    try {
      FormData formData = FormData.fromMap({
        'file': await MultipartFile.fromFile(filePath),
      });

      final response = await _dio.post(url, data: formData);

      if (response.statusCode == 200) {
        return int.parse(response.data.toString());
      } else {
        throw Exception('Failed to upload file');
      }
    } catch (e) {
      throw Exception('Failed to upload file');
```

Figure 6.128-API Service

Text prediction view

```dart
class TextPredictionView extends StatefulWidget {
  const TextPredictionView({super.key});

  @override
  // ignore: library_private_types_in_public_api
  _TextPredictionViewState createState() => _TextPredictionViewState();
}

class _TextPredictionViewState extends State<TextPredictionView> {
  final TextEditingController _genderController = TextEditingController();
  final TextEditingController _ageController = TextEditingController();
  final TextEditingController _handednessController = TextEditingController();
  final TextEditingController _iqController = TextEditingController();
  final TestApiService _apiService = TestApiService();

  String _prediction = 'Prediction will appear here';

  void _getPrediction() async {
    final String gender = _genderController.text;
    final String age = _ageController.text;
    final String handedness = _handednessController.text;
    final String iq = _iqController.text;

    try {
      final int prediction = await _apiService.textPredict(gender, age, handedness, iq);
      setState(() {
         if(prediction==0){
           _prediction = 'Result: no ADHA';
        }if(prediction==1){
           _prediction = 'Result: ADHD';
        }
      });
    } catch (e) {
      setState(() {
        _prediction = 'Error: $e';
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Prediction Screen'),
      ), // AppBar
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          children: [
            TextField(
              controller: _genderController,
              decoration: const InputDecoration(labelText: 'Gender'),
            ), // TextField
            TextField(
              controller: _ageController,
              decoration: const InputDecoration(labelText: 'Age'),
            ), // TextField
            TextField(
              controller: _handednessController,
              decoration: const InputDecoration(labelText: 'Handedness'),
            ), // TextField
            TextField(
              controller: _iqController,
              decoration: const InputDecoration(labelText: 'IQ'),
            ), // TextField
            const SizedBox(height: 20),
            ElevatedButton(
              onPressed: _getPrediction,
              child: const Text('Get Prediction'),
            ), // ElevatedButton
            const SizedBox(height: 20),
            Text(_prediction),
          ],
```

Figure 6.129-Text prediction

The test implementation

```dart
class TestViewBody extends StatefulWidget {
  const TestViewBody({super.key});

  @override
  State<TestViewBody> createState() => _TestViewBodyState();
}

class _TestViewBodyState extends State<TestViewBody> {
  var qrstr = "let's Scan it";

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(
        fit: StackFit.expand,
        children: [
          Image.asset(
            'Assets/Images/testBg.png',
            fit: BoxFit.fill,
          ), // Image.asset
          ListView(
            children: [
              SizedBox(
                height: 80.height,
              ), // SizedBox
              const Image(
                  image: AssetImage('Assets/Images/testBrain.png'),
                  height: 470), // Image
              CustomButton(
                btnText: 'Written Data Diagnosis',
                width: 200.width,
                onTap: () => RouteUtils.push(const TextPredictionView()),
              ), // CustomButton
              SizedBox(
                height: 20.height,
              ), // SizedBox
              CustomButton(
                btnText: 'Full Diagnosis',
                width: 200.width,
                onTap: () => _showFilePickerDialog(context),
              ) // CustomButton
            ],
          ) // ListView
        ],
      ), // Stack
    ); // Scaffold
  }

  void _showFilePickerDialog(BuildContext context) {
    showDialog(
        context: context,
        builder: (BuildContext dialogContext) {
          return Dialog(
            child: Container(
              height: 350,
              width: 500,
              decoration: BoxDecoration(
                  borderRadius: BorderRadius.circular(30),
                  color: AppColors.white), // BoxDecoration
              child: Column(
                children: [
                  Padding(
                    padding: EdgeInsets.all(30.height),
                    child: DottedBorder(
                      borderType: BorderType.RRect,
                      radius: const Radius.circular(5),
                      dashPattern: const [10, 10],
                      color: Colors.grey,
                      strokeWidth: 2,
                      child: Column(
                        children: [
                          const Image(
                              image: AssetImage('Assets/Images/upload.png')), // Image
                          SizedBox(
                            height: 10.height,
                          ), // SizedBox
                          const AppText(
                            title: 'Browse files',
                            textDecoration: TextDecoration.underline,
                            fontSize: 18,
                            color: AppColors.blue,
                          ), // AppText
                          SizedBox(
                            height: 10.height,
                          ), // SizedBox
                          const AppText(
                            title:
                                'Upload a region image in NIFTI format to get the prediction ',
                            textAlign: TextAlign.center,
                            color: AppColors.darkGray,
                          ), // AppText
                          SizedBox(
                            height: 30.height,
                          ), // SizedBox
                        ],
                      ), // Column
                    ), // DottedBorder
                  ), // Padding
                  Padding(
                    padding: EdgeInsets.symmetric(horizontal: 30.width),
                    child: Row(
                      mainAxisAlignment: MainAxisAlignment.spaceAround,
                      children: [
                        CustomButton(
                          btnText: 'Cancel',
                          btnColor: AppColors.white,
                          textColor: AppColors.blue,
                          height: 35.height,
                          width: 100.width,
                          onTap: () => RouteUtils.pop(),
                        ), // CustomButton
                        CustomButton(
                          btnText: 'Upload',
                          height: 35.height,
                          width: 100.width,
                          onTap: () async {
                            Navigator.of(dialogContext).pop();
                            FilePickerResult? result =
                                await FilePicker.platform.pickFiles();
                            if (result != null) {
                              String filePath = result.files.single.path!;
                              _showFileConfirmationDialog(context, filePath);
                            }
```

Figure 6.130-test implementation

The complete of the test view

```
void _showFileConfirmationDialog(BuildContext context, String filePath) {
showDialog(
  context: context,
  builder: (BuildContext dialogContext) {
    return Dialog(
      child: Container(
          height: 350,
          width: 500,
          decoration: BoxDecoration(
              borderRadius: BorderRadius.circular(30),
              color: AppColors.white), // BoxDecoration
              child: Padding(
                padding:  EdgeInsets.symmetric(horizontal: 30.width,vertical: 10.height),
                child: Column(
                      children: [
                        SizedBox(
                          height: 10.height,
                        ), // SizedBox
                         Image(
                            image:const AssetImage('Assets/Images/aac.png'),height: 100.height,width: 100.width,), // Image
                        SizedBox(
                          height: 10.height,
                        ), // SizedBox
                        const AppText(
                          title: 'File name:',
                          fontSize: 25,
                          color: AppColors.blue,
                        ), // AppText
                        SizedBox(
                          height: 10.height,
                        ), // SizedBox
                         AppText(
                          title:
                              filePath,
                          textAlign: TextAlign.center,
                        ), // AppText
                        SizedBox(
                          height: 30.height,
                        ), // SizedBox
                        Row(
                          mainAxisAlignment: MainAxisAlignment.end,
                          children: [
                            CustomButton(
                        btnText: 'Next',
                        height: 35.height,
                        width: 100.width,
                        onTap: () async {
                                Navigator.of(dialogContext).pop();
                                _showLoadingDialog(context);
                                int result = await _uploadFile(filePath);
                                Navigator.of(context).pop();
                                _navigateToIntermediatePage(context, result);
                              },
                      ), // CustomButton
```

Figure 6.131-test view

Let’s go to the note feature

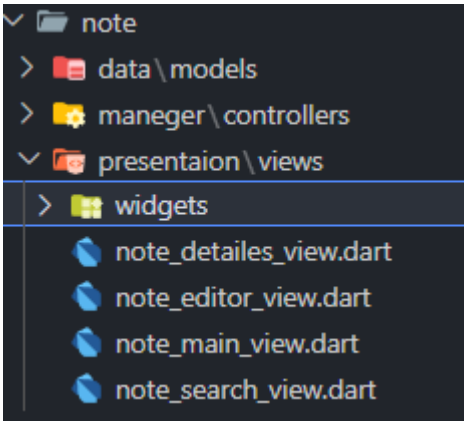


Figure 6.132-note feature

In the data folder there is the note model

```
class Note {
  String title;
  String subtitle;
  String id;

  Note({
    required this.id,
    required this.title,
    required this.subtitle,
  });
}
```

Figure 6.133-data folder

To the manger folder there is name main controller it caches the note and view it in the note main

```
class HomeController {
  List<Note> notes = [];

  Future<void> getCachedNotes() async {
    notes.clear();
    final prefs = await SharedPreferences.getInstance();
    final cachedNotes = prefs.getStringList('notes') ?? [];
    for (var i in cachedNotes) {
      notes.add(
        Note(
          subtitle: jsonDecode(i)['subtitle'],
          title: jsonDecode(i)['title'],
          id: jsonDecode(i)['id'],
        ),
      );
```

Figure 6.134-name main controller

Note editor controller to emit the changes in the note

```dart
class NoteEditorController {
  NoteEditorController({this.note});

  Note? note;

  final formKey = GlobalKey<FormState>();

  TextEditingController titleTXController = TextEditingController();
  TextEditingController subtitleTXController = TextEditingController();

  Future<Note?> addNote() async {
    if (!formKey.currentState!.validate()) {
      return null;
    }
    SharedPreferences prefs = await SharedPreferences.getInstance();
    String id = DateTime.now().millisecondsSinceEpoch.toString();
    List<String> cachedNotes = prefs.getStringList('notes') ?? [];
    cachedNotes.insert(
      0,
      jsonEncode({
        'title': titleTXController.text,
        'subtitle': subtitleTXController.text,
        "id": id,
      }),
    );
    await prefs.setStringList(
      'notes',
      cachedNotes,
    );
    return Note(
      title: titleTXController.text,
      id: id,
      subtitle: subtitleTXController.text,
    );
  }

  Future<Note?> editNote() async {
    if (!formKey.currentState!.validate()) {
      return null;
    }
    SharedPreferences prefs = await SharedPreferences.getInstance();
    note!.title = titleTXController.text;
    note!.subtitle = subtitleTXController.text;
    List<String> cachedNotes = prefs.getStringList('notes') ?? [];
    int index = cachedNotes.indexWhere((element) {
      return jsonDecode(element)['id'] == note!.id;
    });
    cachedNotes.removeAt(index);
    cachedNotes.insert(index, jsonEncode({
      'id': note!.id,
      'title': titleTXController.text,
      'subtitle': subtitleTXController.text,
    }));
    await prefs.setStringList(
      'notes',
      cachedNotes,
    );
    return note;
```

Figure 6.135-Note editor controller

Note search controller for the search

```dart
class SearchController {

  SearchController({required this.notes});
  final List<Note> notes;

  List<Note> filteredNotes = [];

  Future<void> search(String? value) async {
    filteredNotes.clear();
    if (value == null || value.trim().isEmpty) {
      return;
    }
    for (var note in notes) {
      if (note.title.toLowerCase().contains(value.toLowerCase())) {
        filteredNotes.add(note);
      }
    }
```

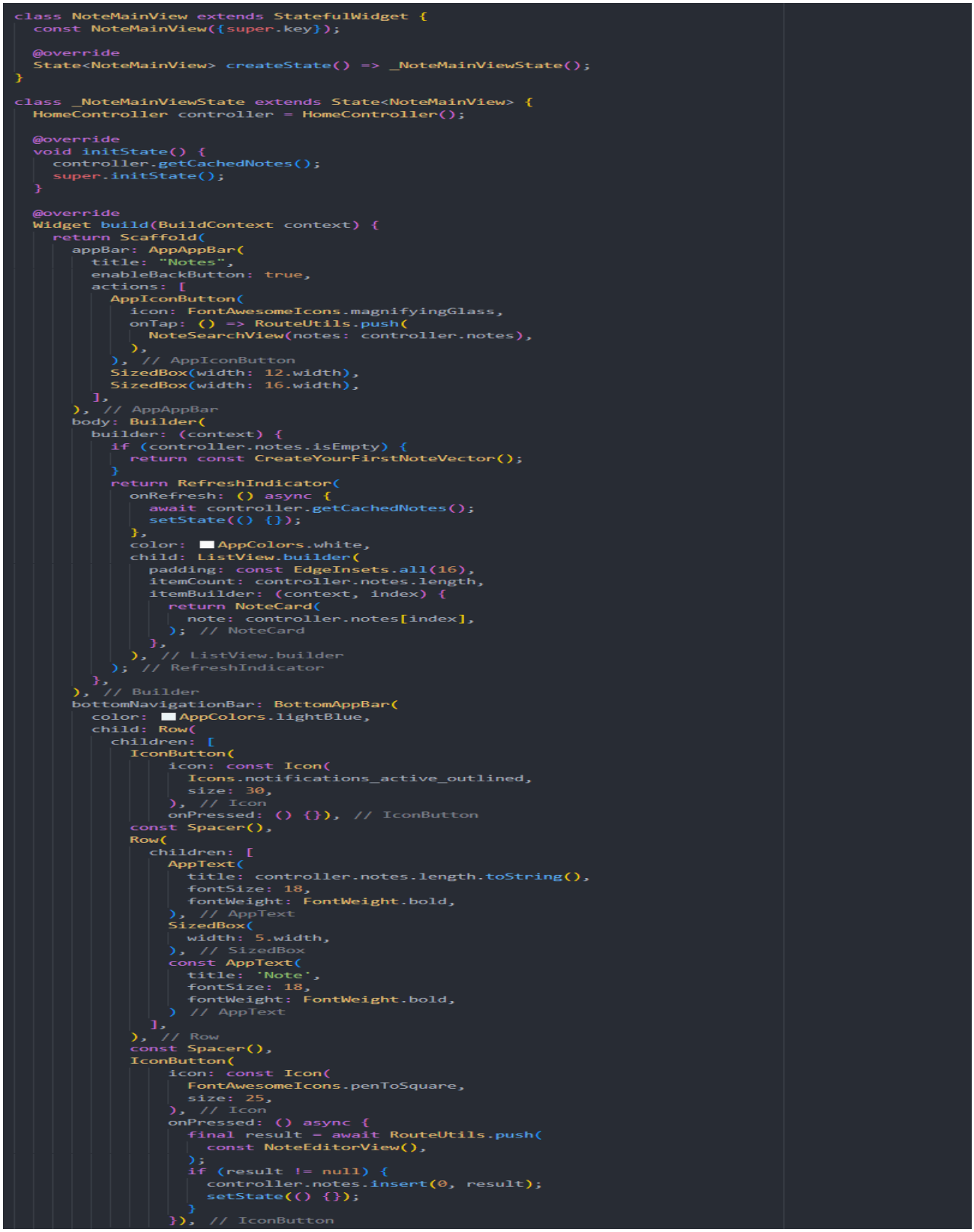
Figure 6.136-Note search controller

In the presentation widget folder, there is the note cart

```dart
class NoteCard extends StatelessWidget {
  const NoteCard({super.key, required this.note});

  final Note note;

  EdgeInsets get _cardMargin => EdgeInsets.only(bottom: 5.height);

  BorderRadius get _radius => BorderRadius.circular(10);

  @override
  Widget build(BuildContext context) {
    return Dismissible(
      key: UniqueKey(),
      onDismissed: (direction) async {
        final prefs = await SharedPreferences.getInstance();
        final cachedNotes = prefs.getStringList('notes') ?? [];
        final index = cachedNotes.indexWhere((element) {
          return note.id == jsonDecode(element)['id'];
        });
        cachedNotes.removeAt(index);
        prefs.setStringList('notes', cachedNotes);
      },
      background: Container(
        margin: _cardMargin,
        width: double.infinity,
        alignment: Alignment.center,
        decoration: BoxDecoration(
          color: AppColors.red,
          borderRadius: _radius,
        ), // BoxDecoration
        child: const Icon(FontAwesomeIcons.trash,color: AppColors.white,),
      ), // Container
      child: Padding(
        padding: _cardMargin,
        child: InkWell(
          borderRadius: _radius,
          onTap: () => RouteUtils.push(
            NoteDetailsView(note: note),
          ),
          child: Container(
            padding: EdgeInsets.symmetric(
              horizontal: 25.width,
              vertical: 16.height,
            ), // EdgeInsets.symmetric
            width: double.infinity,

            decoration: BoxDecoration(
              borderRadius: _radius,
              color: AppColors.lightBlue,
            ), // BoxDecoration
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                AppText(
                  title: note.title,
                  color: AppColors.black,
                  fontSize: 18,
                  fontWeight: FontWeight.bold,
                ), // AppText
                AppText(
                  title: note.subtitle,
                  color: AppColors.darkGray,
                  fontSize: 12,
                ), // AppText
```

Figure 6.137- note cart

Note main view implementation



```
class NoteMainView extends StatefulWidget {
  const NoteMainView({super.key});

  @override
  State<NoteMainView> createState() => _NoteMainViewState();
}

class _NoteMainViewState extends State<NoteMainView> {
  HomeController controller = HomeController();

  @override
  void initState() {
    controller.getCachedNotes();
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppAppBar(
        title: "Notes",
        enableBackButton: true,
        actions: [
          AppIconButton(
            icon: FontAwesomeIcons.magnifyingGlass,
            onTap: () => RouteUtils.push(
              NoteSearchView(notes: controller.notes),
            ),
          ), // AppIconButton
          SizedBox(width: 12.width),
          SizedBox(width: 16.width),
        ],
      ), // AppAppBar
      body: Builder(
        builder: (context) {
          if (controller.notes.isEmpty) {
            return const CreateYourFirstNoteVector();
          }
          return RefreshIndicator(
            onRefresh: () async {
              await controller.getCachedNotes();
              setState(() {});
            },
            color: AppColors.white,
            child: ListView.builder(
              padding: const EdgeInsets.all(16),
              itemCount: controller.notes.length,
              itemBuilder: (context, index) {
                return NoteCard(
                  note: controller.notes[index],
                ); // NoteCard
              },
            ), // ListView.builder
          ); // RefreshIndicator
        },
      ), // Builder
      bottomNavigationBar: BottomAppBar(
        color: AppColors.lightBlue,
        child: Row(
          children: [
            IconButton(
                icon: const Icon(
                  Icons.notifications_active_outlined,
                  size: 30,
                ), // Icon
                onPressed: () {}), // IconButton
            const Spacer(),
            Row(
              children: [
                AppText(
                  title: controller.notes.length.toString(),
                  fontSize: 18,
                  fontWeight: FontWeight.bold,
                ), // AppText
                SizedBox(
                  width: 5.width,
                ), // SizedBox
                const AppText(
                  title: 'Note',
                  fontSize: 18,
                  fontWeight: FontWeight.bold,
                ) // AppText
              ],
            ), // Row
            const Spacer(),
            IconButton(
                icon: const Icon(
                  FontAwesomeIcons.penToSquare,
                  size: 25,
                ), // Icon
                onPressed: () async {
                  final result = await RouteUtils.push(
                    const NoteEditorView(),
                  );
                  if (result != null) {
                    controller.notes.insert(0, result);
                    setState(() {});
                  }
                }), // IconButton
```

Figure 6.138-Note implementation

Note details view implementation

```dart
class NoteDetailsView extends StatelessWidget {
  const NoteDetailsView({super.key, required this.note});

  final Note note;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppAppBar(
        actions: [
          AppIconButton(
            icon: FontAwesomeIcons.penToSquare,
            onTap: () async {
              final result = await RouteUtils.push(
                NoteEditorView(note: note),
              );
              if (result != null) {
                // ignore: use_build_context_synchronously
                Navigator.pop(context, result);
              }
            },
          ), // AppIconButton
          const SizedBox(width: 16),
        ],
        enableBackButton: true,
      ), // AppAppBar
      body: ListView(
        padding: const EdgeInsets.all(16),
        children: [
          AppText(
            title: note.title,
            fontSize: 36,
            fontWeight: FontWeight.w600,
          ), // AppText
          SizedBox(height: 16.height),
          AppText(
            title: note.subtitle,
            fontSize: 24,
          ), // AppText
```

Figure 6.139-Note details

Note editor view implementation

```dart
class NoteEditorView extends StatefulWidget {
  const NoteEditorView({
    super.key,
    this.note,
  });

  final Note? note;

  @override
  State<NoteEditorView> createState() => _NoteEditorViewState();
}

class _NoteEditorViewState extends State<NoteEditorView> {
  late NoteEditorController controller;
  DateTime? _selectedDate;
  TimeOfDay? _selectedTime;

  Future<void> _selectDate(BuildContext context) async {
    final DateTime? picked = await showDatePicker(
      context: context,
      initialDate: _selectedDate ?? DateTime.now(),
      firstDate: DateTime(2000),
      lastDate: DateTime(2101),
    );
    if (picked != null && picked != _selectedDate) {
      setState(() {
        _selectedDate = picked;
      });
    }
  }

  Future<void> _selectTime(BuildContext context) async {
    final TimeOfDay? picked = await showTimePicker(
        context: context, initialTime: _selectedTime ?? TimeOfDay.now()
        );

    if (picked != null && picked != _selectedTime) {
      setState(() {
        _selectedTime = picked;
      });
    }
  }

  void _showDateTimePickerMenu(BuildContext context) {
    showMenu(
      context: context,
      position: const RelativeRect.fromLTRB(50, 50, 50, 50),
      items: [
        PopupMenuItem(
          child: ListTile(
            leading: const Icon(
              Icons.calendar_today,
              color: AppColors.blue,
            ), // Icon
            title: const AppText(
              title: 'Pick Date',
              color: AppColors.darkGray,
            ), // AppText
            onTap: () {
              Navigator.pop(context);
              _selectDate(context);
            },
          ), // ListTile
        ), // PopupMenuItem
        PopupMenuItem(
          child: ListTile(
            leading: const Icon(
              Icons.access_time,
              color: AppColors.blue,
            ), // Icon
            title: const AppText(
              title: 'Pick Time',
              color: AppColors.darkGray,
            ), // AppText
            onTap: () {
              Navigator.pop(context);
              _selectTime(context);
            },
          ), // ListTile
        ), // PopupMenuItem
      ],
    );
  }

  @override
  void initState() {
    controller = NoteEditorController(note: widget.note);
    controller.titleTXController.text = widget.note?.title ?? '';
    controller.subtitleTXController.text = widget.note?.subtitle ?? '';
    super.initState();
  }
```

Figure 6.140- Note editor view

The complete of the note editor view:

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: AppColors.white,
    appBar: AppAppBar(
      enableBackButton: true,
      actions: [
        AppIconButton(
          icon: FontAwesomeIcons.eye,
          onTap: () {},
          padding: EdgeInsets.only(right: 16.width),
        ), // AppIconButton
        AppIconButton(
          icon: FontAwesomeIcons.floppyDisk,
          onTap: () => AppDialog.show(
            context,
            message: "Save changes ?",
            confirmTitle: "Save",
            onConfirm: () async {
              Note? note;
              if (widget.note == null) {
                note = await controller.addNote();
              } else {
                note = await controller.editNote();
              }
              if (note == null) {
                Navigator.pop(context);
              } else {
                Navigator.pop(context);
                Navigator.pop(
                  context,
                  note,
                );
              }
            },
            onCancel: () {
              Navigator.pop(context);
            },
          ),
        ), // AppIconButton
        const SizedBox(width: 16),
        AppIconButton(
          icon:Icons.alarm,
          onTap: () => _showDateTimePickerMenu(context),
        ), // AppIconButton
        const SizedBox(width: 16),
      ],
    ), // AppAppBar
    body: Form(
      key: controller.formKey,
      child: ListView(
        keyboardDismissBehavior: ScrollViewKeyboardDismissBehavior.onDrag,
        padding: const EdgeInsets.all(16),
        children: [
          AppTextField(
            hint: 'Title',
            cursorHeight: 52,
            hintFontSize: 48,
            maxLength: 50,
            controller: controller.titleTXController,
            validator: (v) {
              if (v == null || v.trim().isEmpty) {
                return 'Empty field!';
              }
              return null;
            },
          ), // AppTextField
          SizedBox(height: 16.height),
          AppTextField(
            hint: "Type Something...",
            controller: controller.subtitleTXController,
            validator: (v) {
              if (v == null || v.trim().isEmpty) {
                return 'Empty field!';
              }
              return null;
            },
```

Figure 6.141- note editor view cont.

.

Note search view implementation:

```dart
class NoteSearchView extends StatefulWidget {
  const NoteSearchView({super.key, required this.notes});

  final List<Note> notes;

  @override
  State<NoteSearchView> createState() => _NoteSearchViewState();
}

class _NoteSearchViewState extends State<NoteSearchView> {
  late SearchController controller;

  @override
  void initState() {
    controller = SearchController(notes: widget.notes);
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: PaddingAppBar(
          appBar: AppBar(
            leading: Padding(
              padding: EdgeInsets.only(left: 10.width),
              child: IconButton(
                icon: const Icon(
                  Icons.arrow_back_ios_new,
                  size: 33,
                ), // Icon
                onPressed: () {
                  RouteUtils.pop();
                },
              ), // IconButton
            ), // Padding
            title: SearchTextField(
              onChanged: (v) {
                controller.search(v);
                setState(() {});
              },
            ), // SearchTextField
          ), // AppBar
          height: 80.height), // PaddingAppBar
      body: Padding(
        padding: const EdgeInsets.symmetric(horizontal: 16),
        child: Column(
          children: [
            SizedBox(height: 32.height),
            Expanded(
              child: Builder(
                builder: (context) {
                  if (controller.filteredNotes.isEmpty) {
                    return const NoSearchResultVector();
                  }
                  return ListView.builder(
                    itemCount: controller.filteredNotes.length,
                    itemBuilder: (context, index) {
                      return NoteCard(note: controller.filteredNotes[index]);
                    },
```

Figure 6.141- Note search view

And finally, the main dart file to implement all of the application:

```dart
Run | Debug | Profile
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  await ScreenUtil.ensureScreenSize();
  await CachingUtils.init();
  await NetworkUtils.init();
  runApp(const AdhdApp());
}

class AdhdApp extends StatelessWidget {
  const AdhdApp({
    super.key,
  });

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      navigatorKey: RouteUtils.navigatorKey,
      debugShowCheckedModeBanner: false,
      // routerConfig: AppRouter.router,
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ), // ThemeData
      builder: (context, child) {
        ScreenUtil.init(
          context,
          designSize: const Size(375, 812),
        );
        return child!;
      },
      title: "Ahdh",
      home:
      const SplashView(),
    ); // MaterialApp
```

Figure 6.142- main dart file

Chapter 7

Future Work and Conclusion

Future Work

7.1 Conclusion

In conclusion, our graduation project presents a novel approach to supporting children with ADHD. By leveraging advanced technologies like fMRI, gamified mobile applications, and reminder systems, we've developed a comprehensive and engaging platform to manage ADHD symptoms.

Our project aimed to provide children with ADHD with accessible tools to improve focus, self-regulation, and organizational skills. The fMRI diagnosis offered a deeper understanding of each child's unique brain activity patterns, allowing for personalized treatment plans. The mobile application, with its engaging games and reminder features, aimed to make managing ADHD symptoms fun and interactive for children.

Furthermore, collaboration with medical professionals ensured our approach aligned with best practices in ADHD management. This collaboration also allowed for seamless integration of the app within existing treatment regimens.

Despite the challenges of developing effective games, ensuring data privacy in fMRI analysis, and integrating seamlessly with existing healthcare systems, we remain optimistic about the impact of our project. We believe this approach has the potential to empower children with ADHD, improve their daily lives, and contribute to a better understanding of the condition.

7.2 Future Works

While our graduation project has laid the foundation for a comprehensive and engaging approach to ADHD rehabilitation, there are several avenues for future work and improvement. These include:

1. **Developing a handedness Hardware-Integrated recorder:** Touchscreen interfaces on smartphones, tablets, or dedicated devices can be used to assess handedness through interactive tasks such as drawing shapes, tapping targets, swiping in specific directions, and tracing patterns can be designed to evaluate hand dominance and motor coordination. Leveraging capacitive touch technology, these devices can detect precise finger movements and pressure applied during tasks. The collected data contributes to a comprehensive handedness profile, highlighting dominant hand usage and coordination abilities.

2. **Integrating neurofeedback capabilities into app:** Develop interactive modules that guide users through neurofeedback training sessions. These modules can be designed to be engaging and accessible, using gamification elements or personalized challenges to maintain user motivation.

3. **Incorporate Smart Microphones and Voice Analysis Software:** Analyze speech patterns for signs of impulsivity or inattention during conversations. Provide immediate feedback to users or caregivers about speech patterns associated with ADHD symptoms. Enable users to track their own speech metrics over time, fostering self-awareness and self-monitoring skills.

4. **Merging of Eye-tracking technology:** Can be a powerful tool in an ADHD diagnosis app, offering insights into focus, attention, and gaze patterns which can analyze where a user is looking and for how long, providing data on attentional focus and distractibility and Measure how long the eyes remain on specific points of interest, indicating sustained attention or rapid shifts in focus.

5. **Blending Smartwatches and Fitness Bands:** Track physical activity, heart rate, and sleep patterns, which can provide valuable data on hyperactivity and restlessness.

6. **Combining Home Environment Sensors for app:** Measure ambient light intensity throughout the day to understand its impact on alertness and circadian rhythms. Monitor background noise levels, identifying environments that may be distracting or overstimulating. Track environmental comfort factors that can affect concentration and mood stability.

7. **Upgrades in ADHD diagnosis app:** There are some further software innovations that could enhance our app like ”**Diet and Nutrition Tracking”:** Include features for tracking diet and nutrition, as certain foods and nutritional deficiencies can impact ADHD symptoms, **"Educational Accommodation Recommendations":** that Provide personalized recommendations for educational accommodations based on assessment results and individual needs**,**and”**Executive Function Training**": which develop interactive modules for cognitive training focused on improving executive functions such as planning, organization, and time management.

8. **Virtual Reality (VR) Therapy**: Explore VR applications for exposure therapy or simulated environments to help individuals practice coping strategies in controlled settings.

References

https://fcon_1000.projects.nitrc.org/indi/adhd200/index.html ,

https://asq.org/quality-resources/new-management-planning-tools,

https://github.com/neurodata/neuroparc/tree/master/atlases/label/Human,

https://adhdtest.ai/post/attention-deficit-hyperactivity-disorder-adhd-in-children-understanding-and-supporting-your-child,

https://www.open.edu/openlearn/health-sports-psychology/understanding-adhd/content-section---introduction,

https://www.ncbi.nlm.nih.gov/books/NBK441838/,

https://www.webmd.com/add-adhd/default.htm,

https://www.nimh.nih.gov/health/topics/attention-deficit-hyperactivity-disorder-adhd,

https://chadd.org/,

https://psychcentral.com/adhd,

https://www.additudemag.com/resources/,

https://www.adhdfoundation.org.uk/resources/,