IPC – Intelligent Personalized Care

A Solution to promote Physical Agility and Recovery

Authors: Guilherme Cepeda
Tiago Martinho
Rodrigo Neves

Supervisor: Paulo Pereira, ISEL

Project report written for Project and Seminar 22/23
Computer Science and Engineering Bachelor's degree

June 2023

Instituto Superior de Engenharia de Lisboa

IPC – Intelligent Personalized Care

A Solution to promote Physical Agility and Recovery

47531	Guilherme Pedro Serra Cepeda
	a47531@alunos.isel.pt
48256	Tiago Alexandre Monteiro Martinho
	a48256@alunos.isel.pt
46536	Rodrigo Fernandes das Neves
	a46536@alunos.isel.pt
Supervisor:	Paulo Pereira
	palbp@cc.isel.ipl.pt

Project report written for Project and Seminar 22/23
Computer Science and Engineering Bachelor's degree

May 2023

Abstract

According to the World Health Organization (WHO) [1], approximately one-fifth of the global population lives with musculoskeletal conditions causing chronic pain. However, there is a shortage of healthcare professionals available to provide continuous rehabilitation exercises required to prevent physical problems resulting from weakened and inflexible muscles. Additionally, the lack of physical activity and exercise contributes significantly to the prevalence of chronic diseases, which remain leading causes of death worldwide.

To address these challenges, the Intelligent Personalized Care project has been developed. Its primary objective is to combat the rising sedentary lifestyle and social isolation by leveraging advanced artificial intelligence technologies to deliver personalized and effective remote care for patients across diverse health scenarios. The project utilizes an Android application that promotes physical agility and recovery through tele-exercise and tele-rehabilitation, incorporating immersive approaches facilitated by vision techniques and artificial intelligence. Moreover, the application offers comprehensive patient monitoring and follow-up functionalities, empowering physiotherapists to closely track and assess patient progress.

By combining human capabilities with cutting-edge technology, the Intelligent Personalized Care project aims to revolutionize prevention, treatment, and the overall quality of life. Through its innovative approach, it seeks to mitigate sedentary lifestyles, prevent physical complications, and promote active engagement in exercise and rehabilitation. This solution holds immense potential to bridge the gap between demand and available resources, enabling more individuals to access the care they need, while enhancing their overall well-being.

Contents

List of Figures

Listings

Chapter 1

Introduction

The following chapter introduces the motivation behind the project and also the objectives and main functionalities that are to be developed.

1.1 Motivation

According to the WHO[1], about 20% of the world's population lives with a painful musculoskeletal condition. Continuous rehabilitation exercises are needed and there are not enough professionals to do this. It is necessary to avoid injuries and other physical problems that result from weakened and inflexible muscles. Keeping in mind that the lack of physical activity/exercise is one of the biggest causes of death, through chronic diseases.

There is a need to rethink prevention, treatment and improvement of the quality of life. It is imperative to discover the best way to combine human capabilities with technology to ensure better services and experience for people.

1.2 Goals

Our *Intelligent Personalized Care* project aims to respond to the rapidly increasing sedentary lifestyle and isolation of the population. Using advanced artificial intelligence technologies to provide personalized and effective remote care for patients in different musculoskeletal health scenarios, it will be possible to reduce and prevent sedentary lifestyle, and promote physical exercise and rehabilitation in people.

This project is based on an Android application to promote physical agility and recovery, through (tele) exercise and (tele) rehabilitation using immersive approaches with vision techniques and artificial intelligence. In addition, the application will offer advanced patient monitoring and follow-up features, allowing physiotherapists to monitor the patient progress.

1.3 Main functionalities

So that the project could reach the desired outcome, the definition of the requirements was needed, thus becoming the following:

- Create a relational database to store all data related to the application's resources;
- Create a database based on Firebase to store all media files;
- Create a RESTful Web API to access database resources;
- Use Artificial Intelligence techniques to validate physical exercises;
- Create a mobile application that allows the recording of exercises, using the services provided by the aforementioned API;

1.4 Report Structure

This report is composed by 9 chapters, that will outline how the project architecture is assembled and the technologies used in it.

Chapter 2 lists the different functionalities within the project, and the different roles that exist.

Chapter 3 describes the project architecture concerning the server (backend) and the client (frontend), explaining in a grand manner how each of them work.
In this chapter the different functionalities and roles will be explained with more detail as well as the technologies used, namely which were chosen and why they were chosen, it will also explain the database model built and used and how its entities relationships with one another.

Chapter 4 will describe in further detail the server side implementation with an explanation about its structure and layers, the authentication and authorization process, the exception handling and our working technique.

Chapter 5 will describe in further detail the client side implementation, providing information about the Android Application and its components, as well as the requests to the API are made and how we handle with exceptions originated from the requests made. Also explains the authorization and authentication process in the client and It will provide an introduction to the artificial intelligence used in the project.

Chapter 6 describes the artificial intelligence model used, and the techniques to verify the specific exercises

Chapter 7 shows the functional aspects of the project in a graphical way.
Chapter 8 explains how does the connection between a monitor and a client works.
Chapter 9 reflects our thoughts and conclusions regarding the course of the project

Chapter 2

Functionalities

The following section presents all functionalities that the application will be capable of executing. Graphical examples can be found in chapter 7.

The system supports two types of roles: Monitor and Client.

- **Monitor:** Able to accept or decline a client request, able to do a specific plan with specific exercises for a specific client, assigns that plan to the client and gives feedback after the client executed the exercise.
- **Client:** Able to request to connect to a specific monitor, able to see the plan assigned, able to do the exercises of the plan, after choosing a exercise records a video and sends it and after receives the monitor feedback.

The present functionalities in the project are:

- **Sign-in and Sign-up:** Non-Members can apply to be a client or a monitor, using the system verification procedures.

- **Creation of a Client:** Every non-member who does not have an account can create one easily.

- **Creation of a Monitor:** The creation of a monitor, needs to be verified as the person who is trying to register himself as a monitor needs to pass through a checking phase where we verify the authenticity of the document inserted.

- **Client can see the available Monitors:** The client can see the available monitors and choose one based on rating.

- **Connection request from the Client to the Monitor:** The client is able to send a connection request to the monitor with a description of his problem, which the monitor can accept or decline

- **Creation of a plan:** The monitor creates a plan.

- **Assign a plan:** The monitor can assign a specific plan to a specific client

- **Client uses the plan:** When the client uses the plan he can choose one of the exercises in it and record himself doing it, after the completion of the exercise he can send the video to the Storage

- **Feedback form the Monitor:** The monitor can give feedback on a specific exercise done by one of his clients

All these functionalities were mandatory according to the group, however there are a few additional features to be implemented in the future as described in the last chapter 10.

Chapter 3

Architecture and Data Model

The architecture of the developed system is described in this chapter, with its components illustrated in figure 1.

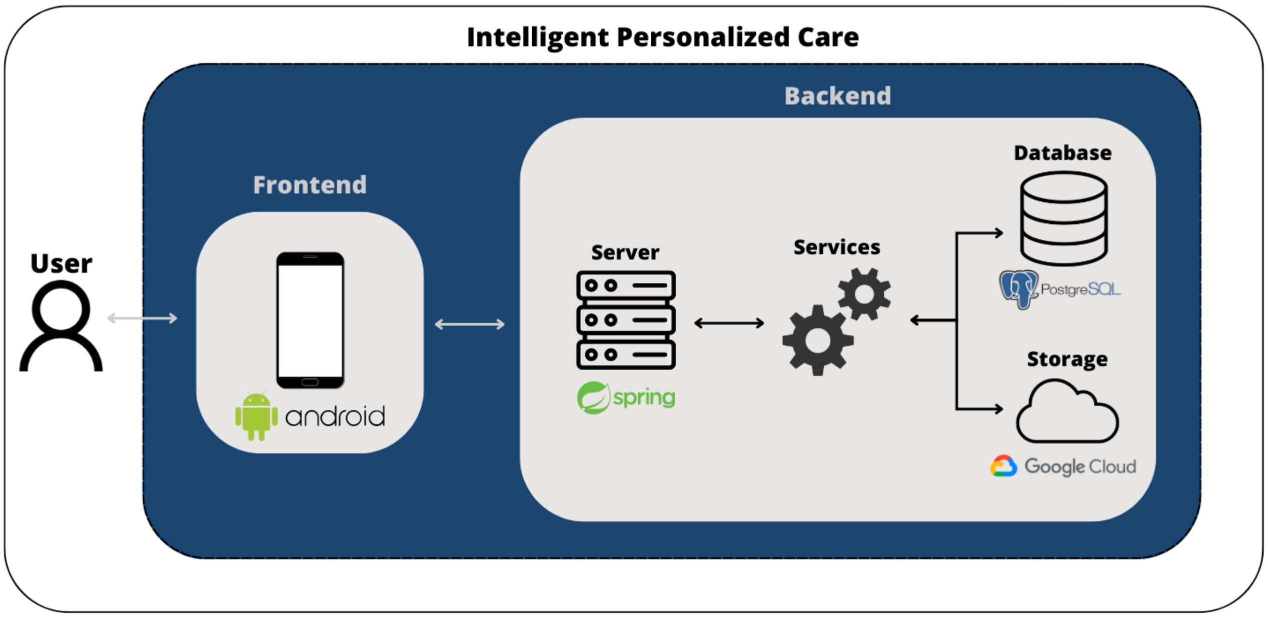

Figure 1- Project Architecture

The system is divides into two parts, the backend and the frontend. In the backend is where most of the system logic is, providing a HTTP API that gives a set of options/capacities to potential users and provides two different ways of storing data: a relational database to store every piece of data related to the system logic and a cloud storage to store images, videos and documents.

In the other part of the system, the frontend, is an android application that serves as a client to the backend application as well as the main factor in our usage of the artificial intelligence model.

Dividing our system in these two parts allowed us to have numerous users, in our mobile app and the users do not need to maintain knowledge about the system or its logic because this is all already available in the backend part.

The android application was chosen for the frontend part of this project because it was the most practical way of putting our project context into practice, a mobile app which promotes remote and personalized musculoskeletal care. It also allows to have just one implementation for it to be compatible with most android devices.

3.1 Technologies

This section describes the technologies used in this application in both backend side and frontend side.

The server application, or backend is an application that exposes a HTTP API and it was developed using the **Kotlin programming language** [2], has all the information regarding the API, relational database, storage and services.

To store the relevant system information in the database we will use **PostgreSQL** [3], while to save the recorded videos, the **Google Cloud Storage Erro! A origem da referência não foi encontrada.** platform will be used. In order to deal with data from the relational database, we will use the **JDBI** [5] library, which will pass the information to the services. These will be exposed through an HTTP API, where we will use the **Spring** [6] framework as a server.

This API serves as the central point for all users which means that all users (mobile) connect with the same server application. To access the functionalities in the API we need an **uri** [6] and this uri can be simplified as a pair consisting of a path and an HTTP method in which it the combination corresponds to a route in a server that will then carry out actions that correspond to its definition.

The android application consists of a frontend application that aims to design a representative and practical User Interface (UI) to access the resources made available in the backend.

As we are going to implement an Android application, we found the choice of the Kotlin programming language essential, as well as the **Jetpack Compose**[8] library for creating the UI. To deal with reading information from the camera, **the API ML Kit Pose Detection** [9] will be used, as it is a lightweight and versatile solution to detect body position in real time.

The ML Kit Pose Detection API used gives us access to a pre trained model, that will be used to identify certain poses in specific exercises, to help the user do the exercises correctly.

To help us improve the code it was used dependencies such as, **ktlint** [9]**, okhttp** [9], **cameraX** [9] which can help maintaining a consistent behavior of the camera.

In this phase **Ngrok** [9] was chosen as the platform to host the app.

3.2 Data Model

The following section presents the graphical representation of the database in its Entity Association [14] format, describing the existing entities and the relations between them.

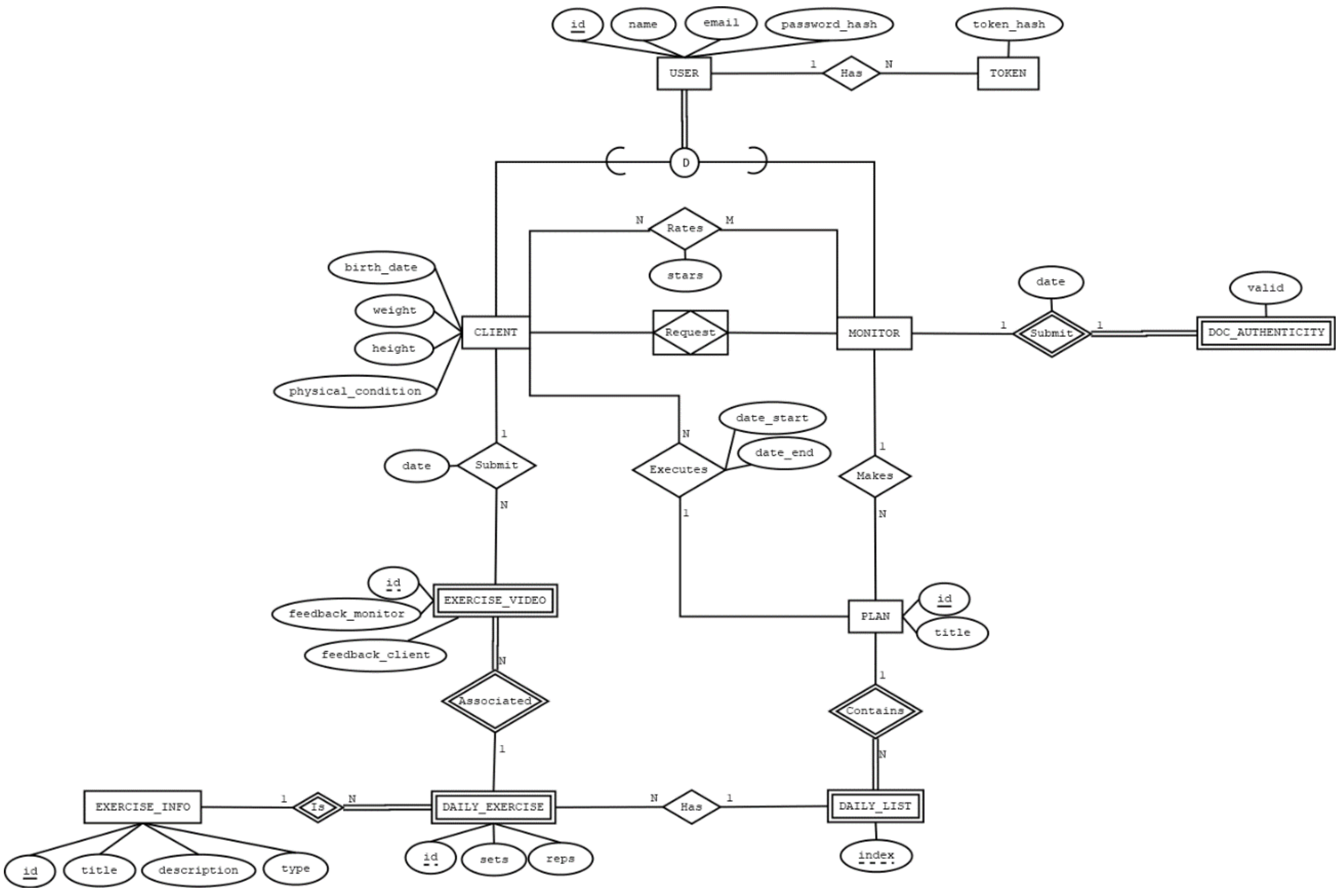

Figure 2- EA Model

As shown in the figure 2, **User** is the core entity, representing a user of the application, and this user is divided into types, a **Client** or a **Monitor**. Each **User** has a **Token** associated to him specifically.

As it was referred before a **User** can be either a **Client** or a **Monitor**, where the **Monitor** needs to submit a document when registering in the app, in an entity called **Doc_Authenticity.**

A **Client** can make a connection request to a specific **Monitor,** as well as rate him with 1 to 5 stars.

The **Monitor** is also associated with an entity called **Plan,** where he makes a plan to a specific **Client**. Each plan contains numerous **Daily_Lists ,** which represents the list of exercises for that specific day, these exercises are represented by the entity

Daily_Exercise and each of these exercises has an **Exercise_Info** with all the info associated to that exercise. The exercises differ in the **Daily_Exercise** entity where the "same" exercise could have differents sets and reps.

Associated with each of these **Daily_Exercises** we've got an entity called **Exercise_Video,** where after submitting the video, It can can have a feedback from the client and after the monitor has seen the video he can also give some feedback about it.

The **Client** can also execute the plan associated to him by the **Monitor** and after he finishes the specific exercise, the recorded exercise video will be submitted for later monitor evaluation and monitoring progress.

Chapter 4

Server Implementation

In this chapter it will be described the server's implementation, each layers' mission and how they accomplish it.

4.1 API

Our Application Program Interface sets the rules and protocols that allows the communication and interaction between the Frontend and Backend. There are several options that were considerate but in the end we decided that the combination of Kotlin with the Spring framework was the best. For the relational database it was chosen postgreSQL with the Java library JDBI. To store videos and photos we decided to use Google Cloud Storage.

4.2 Authentication and Authorization

The user must **Sign up** in our application to use or services. Both Client and monitor must provide the name, email and password. The client can also provide his weight, height, birth date and is current physical conditional. This is all to help the monitor have a little more context about the problem of the client and make a better decision about the exercises that the client should do. The monitor is obligated to provide his credentials of physiotherapist to provide more safety to the costumers since that way any person can not prescribe any exercises.

After the sign up every request must have in the Authorization Header a Json Web Token to authenticate the request. A JWT has chosen since it is a safer way to transmit information between two parties (this case the Android Application and the web server). The usage of JWT is also useful for the Authorization since JWTs are self-contained, meaning the necessary information for any kind of verification that we chose to do is embedded in the token itself. In our case we store the email, id and role of the user. This is necessary because there are several end points that a monitor can acess that a client can’t, and vice-versa. We decided that the JWT should be signed with the “HmacSHA512” algorithm.

4.3 Handlers

The handlers are the components responsible for handling the incoming requests and generate the responses. In our web API we use the HandlerInterceptor to intercept the requests before it reach the Controllers. We done this to impose the Authentication and Authorization. We also use the injection of arguments in this class to have the user information when it reaches the Controllers.

All the Controllers are annotated with the **@RestController Annotation** since we decided to follow the REST Architecture. In this handlers we call the services were the logic of the API will be made.

We also have the need for an Exeception Handler since we decided to throw our costum exceptions when that is an error. In this Handler the exception is captured and depending of the type of exception a Response is made. Because of that we have a Media Type “application/problem+json” that represents when an operation is not successful and carries the explanation for the problem.

4.4 Services

In the Services is where the business logic is implemented. To do this here, several are requests to the Repositories and the Google Cloud Storage are made. An example of the use of Services is in the creation of an account. If the email does not match the an existing email an exception is thrown here. Another example is when sending a video to the monitor. If the user already send the video for that exercise then it can’t send another video. In the services it’s checked if the video a video for that exercise already exists and if so the services does not let the new video to be stored.

4.5 Google Cloud Storage

To store the videos and profile pictures we decided to use the Google Cloud Storage. This platforms stores blob’s(Binary Large Objects) and is designed for high availability, durability and performance. The auto scalability and flexible prices were also taken into account. In the configuration of the Cloud Storage a back-off algorithm was configured so that in the case of any Internal Server Error in the Google Cloud Storage, 5 attempts of the same request will be made but at maximum it can only last 3 minutes. If after 5 attempts or 3 minutes has passed and the request has not successful because of Google Cloud Storage, the operation is deemed unsuccessful.

4.6 Relational Database

To store every information about the users, plans and exercises we decided to use postgresql in combination with JDBI. The biggest selling point to chose a relational database instead of a No-SQL was the the fact that relational databases the ACID compliance that guarantees that the transactions are processed reliably with all changes being committed or rolled back. However to guarantee that the data integrity between the Storage and the postgresql we created our own Transaction Manager that uses the JDBI transactions and in the case of any error, the blob inserted in the storage in deleted.

4.7 Exceptions handling

4.8 Tests

Chapter 5

Client Implementation

The following chapter explains the implementation and thought process behind the client side, that is responsible for the visual element of the project.

Description of the architecture

- **Activities**: Activities are responsible for managing the lifecycle of the application and providing the environment where screens are displayed. In the context of Jetpack Compose, activities host the Compose composer.

- **Screens:** Screens represent individual user interface representations in the application. Each screen is typically composed of one or more composable functions, which define the structure, layout, and visual interaction of the screen. A screen can display data, respond to events, and interact with the ViewModel.

- **ViewModels:** ViewModels are responsible for providing and managing the data required for screen display and interaction. They contain the business logic and are responsible for fetching, transforming, and providing data to the screens. ViewModels are observable and notify screens when data is updated.

The interaction between these components occurs as follows:

1. The Activity hosts the Jetpack Compose composer and manages the overall flow of the application.
2. Each Screen is implemented as a composition of elements using Jetpack Compose functions.
3. The screen interacts with the corresponding ViewModel to obtain the required data for screen display and updates.
4. 4. The ViewModel fetches the necessary data through services, performs transformations and makes them available on the screen through observable properties.
5. The screen observes the ViewModel's properties and is automatically notified when the data is updated, causing the user interface to update as needed.

6. The screen can also send events to the ViewModel, such as button clicks or user interactions, which can trigger additional actions or updates to the data.

This architecture allows for a clear separation of responsibilities, facilitates component reusability, and promotes well-organized and testable code.

5.1 Jetpack Compose

We follow a well-known architecture called **MVVM** (Model-View-ViewModel). This architecture separates responsibilities into distinct layers, making code organization and maintenance easier.

5.2 Views

5.3 API access

We are using a popular open-source HTTP client library for Android apps – OkHttp. Provides efficient communication with RESTful APIs, retrieves data, sends requests with multiple parameters, and handles responses asynchronously. OkHttp simplifies the process of API integration by providing a reliable and efficient networking solution for Android applications.

5.4 Shared Preferences

We are using Android's SharedPreferences to store some user information when creating an account. SharedPreferences is an Android framework class that provides a way to store and retrieve key-value pairs of primitive data types. It is commonly used for storing small amounts of application data that need to persist across app launches or device restarts.

5.5 Exceptions handling

5.6 Internationalization

5.7 Code structure

Chapter 6

ML Kit Pose Detection

The android application artificial intelligence Model and the techniques used will be presented in this chapter.

6.1 Model

ML Kit Pose Detection is an on-device, cross platform (Android and iOS), lightweight solution that tracks a subject's physical actions in real time.
The API produces **a full body 33 point** skeletal match, as seen in figure 3, that includes facial landmarks (ears, eyes, mouth, and nose), along with hands and feet tracking.

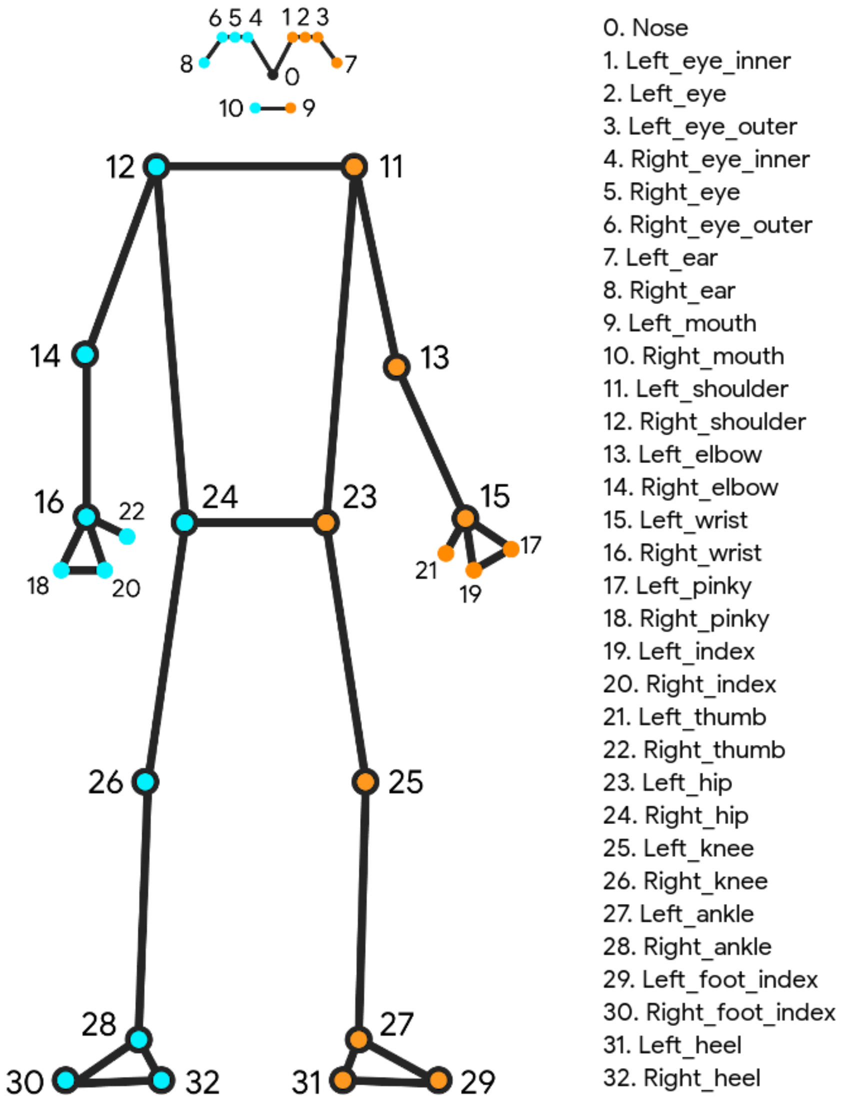

Figure 3- 33 point skeletal match on human body

With the help of the camera and video input, the following image represents the algorithm to find and detect a person pose (landmark detection and pose estimation).

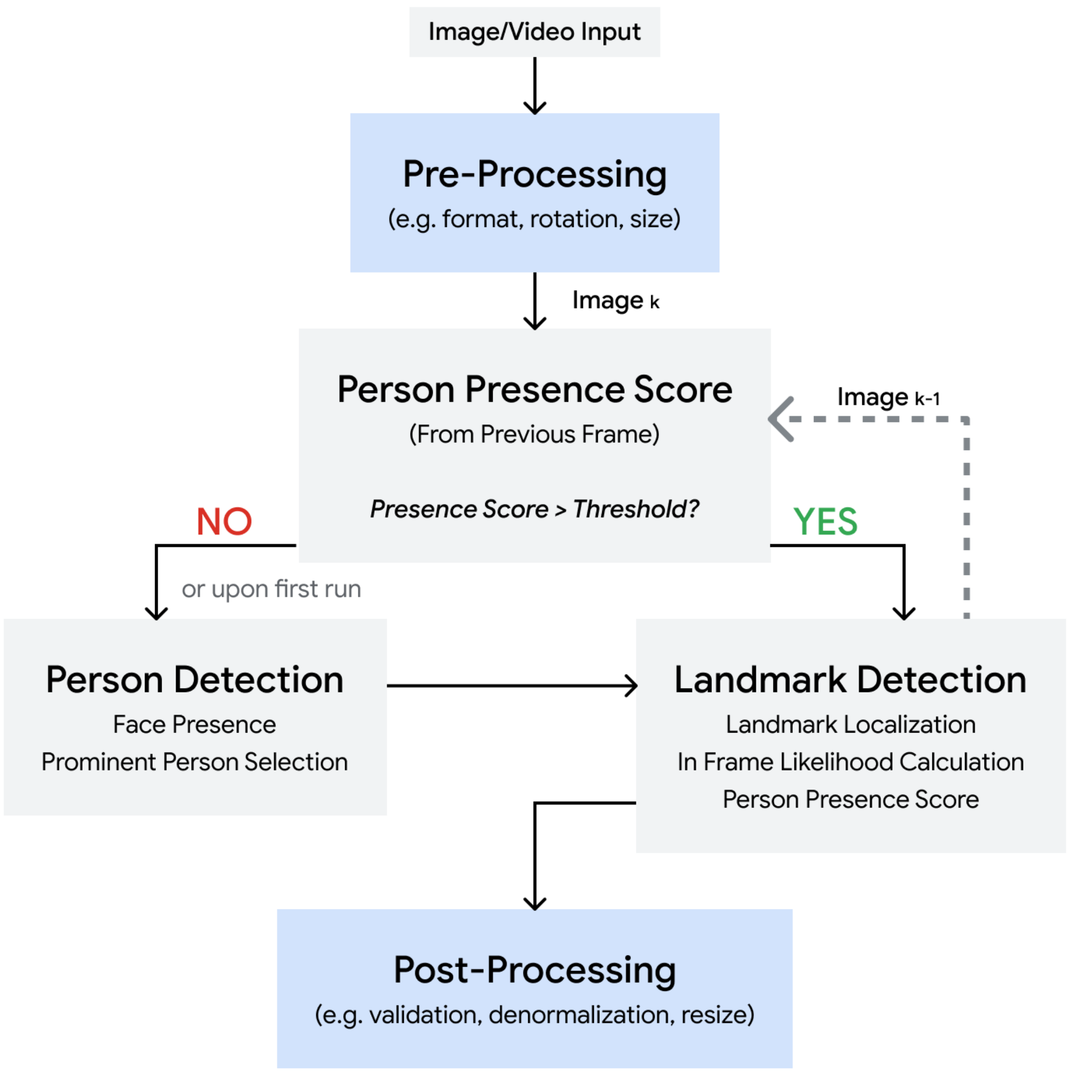

Figure 4 - ML Kit Pose Detection Pipeline

6.2 Exercises

6.3 CameraX and record video

Chapter 7

User Interface and Functionalities

The android application graphical user interface (UI) will be presented in this part. The views below were created with the goal of delivering information as practical and user friendly as possible.

Chapter 8

Connection between a Monitor and a Client

Chapter 9

Conclusion

9.1 Future Work

References

[1] World Health Organization – Musculoskeletal health. https://www.who.int/news-room/fact-sheets/detail/musculoskeletal-conditions

[2] Kotlin programming language - https://kotlinlang.org/

[3] PostgreSQL. https://www.postgresql.org/

[4] Google Cloud Storage, https://cloud.google.com/storage

[5] JDBI. https://jdbi.org/

[6] Spring. https://spring.io/

[7] URI. https://en.wikipedia.org/wiki/Uniform_Resource_Identifier

[8] Jetpack Compose. https://developer.android.com/jetpack/compose?hl=pt-br

[9] ML Kit – API Pose Detection. https://developers.google.com/ml-kit/vision/pose-detection?hl=pt-br, 2023. [Online; accessed 20-03-2023].

[10] Ktlint, https://pinterest.github.io/ktlint/0.49.1/

[11] Okhttp, https://square.github.io/okhttp/

[12] CameraX, https://developer.android.com/training/camerax

[13] Ngrok, https://ngrok.com/

[14] Entity Association, https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model