Performa For The Approval Project Proposal

PNR No.: Rollno:

1. Name of the Student:

---

2. Title of the Project:

---

3. Name of the Guide:

---

4. Teaching experience of the Guide:

---

5. Is this your first submission? Yes ☐ No ☐

Signature of the Student

Date: ……

Signature of the Guide

Date: …….

Signature of the coordinator

Date: ……

Artificial Intelligent Mentor for Mixed Martial Art (AIMMMA)

A Project Report

Submitted in partial fulfillment of the Requirements

for the award of the Degree of BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY)

By

Purushotam Bohra

1066498

Under the esteemed guidance of

Mr. Sohil Luhar

Faculty

People's Education Society's

Siddharth College Of Commerce & Economics

DEPARTMENT OF INFORMATION TECHNOLOGY

Siddharth College of Commerce & Economics, (Fort)

(Affiliated to University of Mumbai)

CITY, 400001

MAHARASHTRA

2023-2024

DEPARTMENT OF INFORMATION TECHNOLOGY

ABSTRACT

AIMMMA is an AI-powered fitness tracking application designed to help MMA enthusiasts of all levels improve their training and performance. It uses computer vision and artificial intelligence to accurately monitor and track fundamental MMA exercises, such as punches, kicks, pushups, and squats. AIMMMA provides real-time feedback to users during their workouts, helping them improve their form and technique. It also generates post-workout analysis to help users identify areas for improvement.

ACKNOWLEDGEMENT

I sincerely thank my parents and relatives for their unwavering support during this project. Their comfortable and supportive home environment enabled me to focus and achieve my best. I also thank my brother for financing the project, without which it would have been impossible to start. Without their support, I would have been unable to achieve my goals. Their unwavering belief in my vision and their unflinching willingness to help me through the inevitable challenges I faced were essential to my success.

I am profoundly grateful to my Project Guide, Mr. Sohil Luhar, for his relentless and thoughtful support throughout the project. His guidance and expertise were invaluable. I am truly grateful for his willingness to go the extra mile. He was always available to answer my questions, provide feedback, and encourage me to push myself to achieve my best. I am confident that I could not have completed this project without his support.

I am also deeply indebted to my friends for their steadfast support during the project. They were my unwavering champions and my unwavering encouragement.

Lastly, I also want to thank all the music and movie producers and the people involved. Your art was there when no one else was, to fill my unproductive time.

DECLARATION

I hereby declare that the project entitled, "**Artificial Intelligent Mentor for Mixed Martial Art (AIMMMA) " done** at **Home**, has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfillment of the requirements for the award of degree of **BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY)** to be submitted as a final semester project as part of our curriculum.

Name and Signature of the Student

TABLE OF CONTENTS

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

> “I do not fear the man who has practiced 10000 types of kicks one time, but the man who has trained 1 kick 10000 times.” ~ Bruce lee

Within the realm of fitness and training, many **individuals** encounter challenges related to monitoring and feedback, which often lead to inconsistencies in their exercise routines. "AI-based Mentor for MMA," abbreviated as AIMMMA, is an application developed to offer users a simple yet valuable tool for tracking and enhancing their fitness experiences. While the initial focus of this application revolves around fundamental exercises such as punches, kicks, pushups, squats, and their combinations, its primary aim is to provide users with an accessible platform for monitoring, evaluating and tracking their progress.

In today's fitness landscape, an abundance of high-tech fitness apps and devices are available. However, these tools often fall short in delivering personalized guidance and feedback tailored to individual needs and goals. AIMMMA recognizes these limitations and endeavours to address them by concentrating on tracking body movement using Computer vision.

What sets this application apart is its dedication to providing a straightforward and user-friendly experience. Its core functionality centers around tracking and recording basic exercises using Computer vision that constitute vital components of MMA training. Users can depend on the application to accurately monitor, log and analyze their punches, kicks, pushups, squats, and combinations thereof.

The primary objective of this application is to assist users in maintaining a consistent fitness routine by furnishing them with a convenient tool for tracking their workouts and seamlessly getting real-time audio feedback. It is important to emphasize that the

application does not seek to replace the physical effort required for exercise but rather aims to enhance it by offering a convenient means of progress tracking.

While future iterations of the application may consider expanding its features and capabilities, the current version provides a practical solution for individuals seeking to monitor their performance in fundamental exercises. The application remains committed to delivering a straightforward and user-centric experience, making it an accessible choice for those interested in basic MMA-inspired workouts.

In addition to its core features, the AIMMMA application offers a distinctive advantage that becomes particularly valuable in situations like a pandemic, where people may be confined to their homes for extended periods with limited access to trainers or fitness facilities. The application's ability to utilize Computer Vision for tracking body movements ensures that users can receive accurate feedback on their exercises without the need for external supervision.

During lockdowns or periods of social isolation, individuals may face challenges in maintaining their fitness routines due to the absence of in-person guidance. AIMMMA steps in as a virtual training companion, providing users with real-time audio feedback based on their movements. This not only addresses the need for personalized guidance but also alleviates the reliance on a physical trainer or gym environment.

1.2 Objectives

- The ability to update ML models with additional data, ensuring that the models adapt and learn from new information over time.
- Use of computer vision and artificial intelligence to monitor and record movements during activities such as punching, kicking, pushups, and squats accurately.
- To leverage computer vision and AI to recognize complex combinations of movements or exercises performed by the user accurately.

- Real-time monitoring of exercise sessions, including measuring the pace of repetitions and timing intervals between sets, ensuring effective workout tracking.
- Provide real-time feedback to users during their workouts through pre-recorded audio cues or instructions, helping them improve their form and performance.
- Development of an intuitive and user-friendly interface that allows users to easily track their fitness progress, view statistics, and set goals.
- Implementing robust measures to ensure the privacy and security of user data, particularly when collecting and processing personal fitness information using AI.
- Providing post-workout analysis that helps individuals get aware about their shortcomings which can help improve their future performance.

1.3 Purpose, Scope and Applicability

1.3.1 Purpose

The purpose of the AI-based Mentor for MMA (AIMMMA) application is to simplify fitness tracking for individuals engaged in mixed martial arts (MMA) training. This application offers an intuitive platform to monitor, evaluate and record user data for their basic exercises such as punches, kicks, pushups, squats, and combinations thereof. Its primary goal is to provide users with a user-friendly tool that enhances their fitness experiences by facilitating progress tracking and analysis.

1.3.2 Scope

The scope of this application is centered on MMA training, particularly the tracking of fundamental exercises. It employs computer vision and artificial intelligence to accurately monitor and record these exercises, providing real-time feedback to users. While its initial focus is on basic exercises, the application has the potential to expand its scope to cover a broader range of MMA-related activities and even adapt to other

fitness disciplines.

1.3.3 Applicability

This application is applicable to MMA enthusiasts and individuals interested in tracking their progress during training sessions. It caters to users of varying experience levels, from beginners seeking guidance to advanced practitioners looking for a convenient way to monitor their workouts. Its user-friendly design and accurate exercise tracking make it a valuable tool for anyone engaged in MMA-inspired workouts, ensuring that users can maintain consistency and improve their performance effectively.

1.4 Achievements

The AIMMMA project has reached significant milestones throughout its development journey, demonstrating exceptional accomplishments. It has not only effectively fulfilled all project requirements but has also seamlessly delivered the anticipated functionalities.

The application boasts a user-friendly interface, ensuring effortless navigation and interaction for its users. The project's primary objectives, centered around precise exercise tracking and real-time AI feedback, have been triumphantly achieved. Users can now confidently monitor their progress in MMA training and gain valuable insights, all within the framework of a straightforward and intuitive user experience.

These accomplishments underscore AIMMMA's unwavering commitment to elevating fitness tracking and providing robust support for MMA enthusiasts, cementing its status as an invaluable tool for those looking to enhance their training regimens.

1.5 Organization of Report

1.5.1 Requirement and Analysis

- The proposed system is an AI based application developed using C++ and

TensorFlow. The objective of the system is to provide users with a user-friendly and efficient tool that allows them to track, monitor and analyze their performance in real-time as well as post workout.

- Software requirements used for the system include NodeJS, TensorFlow. Additionally, web technologies such as HTML, CSS, and JavaScript will be used for the user interface. More libraries like React, Typescript would be utilized to develop a slick front-end. React apps could later be migrated to native apps using React Native.
- Hardware requirements for the system include a laptop or computer running on Windows operating system for back-end with a minimum of 8GB RAM, an Intel Core i5 processor, 11th gen or greater or equivalent of it, and a display resolution of 1024x768 or higher. And a client device to test the app such as Mobile.
- This chapter also shows which conceptual models will be prepared while making the project/system such as the Use Case Diagrams, Activity Diagram and many more.
- Softwares used to develop and test the applications include:
    - Browser - Preferably Chrome.
    - IDE - Visual Studio C++

1.5.2 System Design

- The system design is included in all conceptual schema diagrams which explains the interrelation between the modules and their dependencies.
- The features, working and flow of the entire system is described in this chapter.

1.5.3 Implementation and Testing

- The Implementation process is described in this chapter.
- Describes which all testing approaches are going to take place in this project.

1.5.4 Results and Discussion

- In the end, we got a full-fledged application that runs on a server and uses RESTful

APIs to control the client side app.

1.5.5 Conclusion

The proposed system is a website that provides image editing features such as background changing, image enhancement, and cropping. The system will be developed using Python's Django framework and will use web technologies like HTML, CSS, and JavaScript. The hardware requirements for the system include a computer with at least 4GB of RAM and a Windows operating system. The system aims to provide users with a user-friendly interface and efficient image editing tools for various use cases.

Chapter 2

Survey of Technology

Core Component

- **C++**, a high-performance language that is well-suited for machine learning and backend development, is at the heart of this project. C++ is known for its speed and efficiency, making it ideal for complex tasks such as training and deploying machine learning models. Additionally, C++ is a versatile language that can be used to develop a wide range of applications, from backend systems to web applications to machine learning models.
- **TensorFlow**, a popular open-source machine learning framework, is used in conjunction with C++ to develop and train the machine learning model for this project. TensorFlow provides a wide range of features for building and training machine learning models, including pre-trained models, and layers, and optimizers. This makes it easy to develop and deploy complex machine-learning models without having to start from scratch.
- Beast. Asio, a powerful library for handling HTTP and WebSocket protocols in C++, has been integrated into the project. By incorporating Beast. Asio, the system gains robust networking capabilities, enabling efficient and real-time communication between the core component and the frontend. This facilitates seamless data exchange and interaction, enhancing the overall user experience.

Neural Network Training

- **P5.js**, a JavaScript library for creating interactive graphics and animations, is used to develop a user interface for training the machine learning model. P5.js

makes it easy to create visually appealing and interactive interfaces, which can help users to understand and interact with the machine learning training process.

- **ML5**, a JavaScript library for machine learning in the browser, is used to train the machine learning model using P5.js. ML5 provides a variety of machine learning algorithms, including linear regression, logistic regression, and neural networks. This makes it easy to train a variety of machine learning models without having to leave the browser.

Frontend

- **React**, a popular JavaScript library for building user interfaces, is used to develop the frontend of the web application. React is known for its speed, scalability, and ease of use. React is also a component-based library, making it easy to develop complex user interfaces by breaking them down into smaller, reusable components.
- **TensorFlow.js**, a JavaScript library for training and deploying machine learning models in the browser, is used to deploy the machine learning model to the frontend. TensorFlow.js makes it easy to deploy machine learning models to the browser without having to rely on a backend server. This allows users to interact with the machine learning model directly from the browser, without having to send data back and forth to a server.
- **Vis.js**, a JavaScript library for data visualization, is used to visualize the neural network training process. Vis.js makes it easy to create interactive and informative data visualizations, which can help users to understand the machine learning training process.

Database

- **MongoDB**, a NoSQL database that is scalable and flexible, is used to store the data for the web application. MongoDB is a good choice for this task because it

is easy to use and can handle large amounts of data efficiently.

- **Firebase** for real time.

Communication

- **HTTP**, a protocol for communication between the client and server, is used to communicate between the frontend and the core component. HTTP is a well-established protocol that is supported by all major browsers and servers. Additionally, HTTP is a stateless protocol, which means that each request is independent of all other requests. This makes it easy to scale the web application horizontally by adding additional servers.
- **WebSockets**, a protocol for real-time communication between the client and server, is used to provide real-time feedback to the user during the neural network training process. WebSockets allows the frontend to receive updates from the core component without having to send a request. This makes it possible to create a more responsive and interactive user experience.

Chapter 3

Requirement and System Analysis

3.1 Problem Definition

3.1.1 Problem Definition

Current system lacks the ability to track body pose and classify exercises effectively. They do not analyze the speed of movements, monitor reps and sets using AI technology. Losing crucial real time and deep insights on form and speed of exercises.

3.1.2 Existing System

AI-powered fitness apps can design effective diet and workout plans based on user data, biological facts, and expert opinions. Some apps can track progress, but this may require manual input. Other apps integrate with wearables to track progress. Some apps provide video or 3D instructions on how to do exercises properly. Most apps use scientific data to provide general workout and diet plans and track progress. Few apps have body tracking capabilities, but this is not a core feature.

3.1.3 Proposed System

The proposed system can efficiently track body pose, provide feedback on camera angle and lighting before workouts for optimal body pose classification and analysis,

accurately track particular movements (e.g., push-ups, punches, squats, kicks), efficiently track pose changes and classify them (which can be used to track combos, sets, and reps of exercises), and is user-friendly and free. Most importantly, the system is extensible to newer exercises without significant code refactoring or model retraining.

3.2 Requirement Analysis

Functional

1. Record
2. Pose Detection and Analysis
3. Light Detection and Analysis
4. Camera Angle Detection and Analysis
5. Audio feedback.

Non-Functional

1. Usability
2. Reliability
3. Performance
4. Supportability
5. Safety
6. Security
7. Scalability

3.3 Planning and Scheduling

	Task	Start	End	2023		2024	
				Nov	Dec	Jan	Feb
	Project	11/1/23	2/9/24				
1	Planning	11/1/23	11/22/23				
2	Designing	11/22/23	12/21/23				
3	Data Analysis	12/4/23	1/6/24				
4	Implementation	12/4/23	1/18/24				
5	Testing	1/18/24	2/9/24				

Figure 1: GANTT Chart

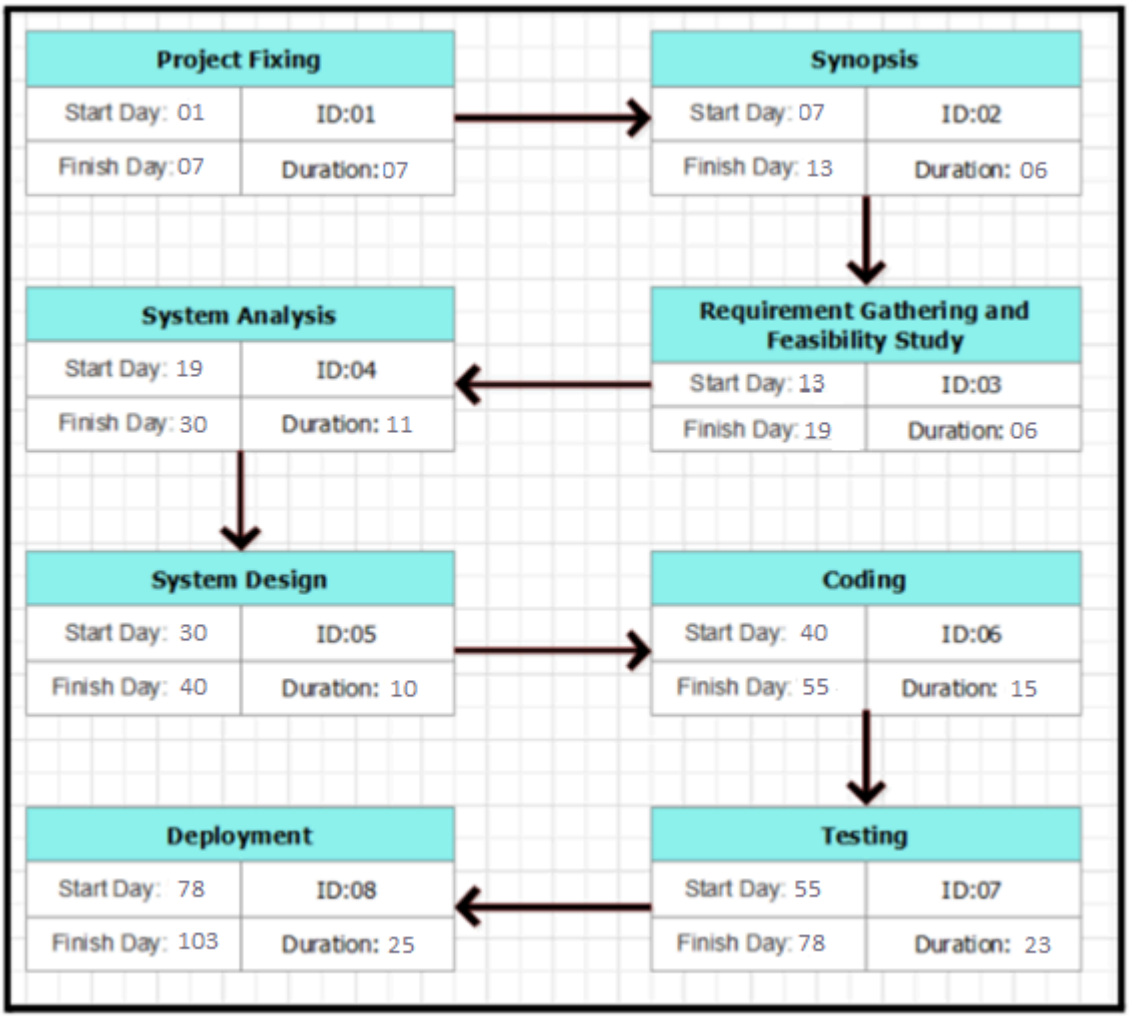


Figure 2: PERT Chart

3.4 Hardware Requirements

Computer

- CPU: Intel i5 12th gen or equivalent AMD CPU
- GPU: Nvidia RTX 2050 or higher (GTX 1650 model is also powerful enough)
- RAM: 8-16 GB
- Storage: Enough storage space to store application data and machine learning models

Client

- Device: Any mobile or computer device with a camera and a browser that supports the Navigation Web API, Camera Web API.
- Speaker: A speaker or Bluetooth handset to receive audio feedback

3.5 Software Requirements

IDE:

- Visual Studio: A complete integrated development environment for building modern applications for Windows, macOS, Linux, iOS, and Android.
- Visual Studio Code: A lightweight but powerful code editor with support for many languages and frameworks.

Compiler:

- Microsoft C++: A compiler for the C++ programming language, developed by Microsoft.

Storage:

- MongoDB: A document database that stores data in JSON-like documents.
- MongoDB Driver for C++: A C++ driver for MongoDB.

Frameworks:

- TensorFlow: An open-source machine learning framework for building and deploying machine learning models.

Libraries:

- React: A JavaScript library for building user interfaces.
- ml5.js: A JavaScript library for machine learning in the browser.
- p5.js: A JavaScript library for creative coding.
- tfjs: A JavaScript library for TensorFlow.js.

3.6 Preliminary Product Description

AIMMMA is an AI-powered fitness tracking application designed to help MMA enthusiasts of all levels improve their training and performance. It uses computer vision and artificial intelligence to accurately monitor and track fundamental MMA exercises, such as punches, kicks, pushups, and squats. AIMMMA provides real-time feedback to users during their workouts, helping them improve their form and technique. It also generates post-workout analysis to help users identify areas for improvement.

Chapter 4

System Design

Decoupling the user interface, research interface and the backend is key to the extensibility of this application. This way the application could be supported on multiple platforms as the user interface will be independent of the backend architecture and the research interface independent of the backend architecture. An abstraction mechanism needs to be established to decouple these three components.

UI to Backend Decoupling: RESTful APIs, WebSocket API.

Backend to RI Decoupling: Model Input and Output Standardization.

4.1 Basic Modules

4.1.1 Pose Classification and Research Interface (Javascript)

Pose classification made with ml5 and p5 libraries. They provide a rough prototypical interface on a white canvas. p5 library enables artists, designers, and engineers to develop quick interfaces for faster prototyping ideas. ml5 and p5 together can be used to create a sophisticated interface to train various models for this application. These modules will help the application to re-iterate the models trained for the long term.

4.1.1.1 Recorder and Pose estimator (Javascript)

This module will help set the camera, the interface for streaming camera recording in real-time then, detect and estimate poses of the person in frame and save it as JSON file to be fed into training the tfjs models.

Process 1: Calculate body distance from the camera. [Requires extensive R&D]

Process 2: Choose a pose to record. Choose the angle of the camera relative to the body.

[Independent of process 1]

Process 3: Analyze body occlusion and detect the full body in the frame. [Depends on process 2] [Prevents poor quality of data to enter model training stage]

Process 4: Start estimating poses and record in JSON format. [Depends on process 3]

Process 5: Analyze pose data for outliers and missing values.

Process 5: Data Augmentation and Normalize data for model feeding. [Requires process 1 input] [Requires extensive R&D]

Rotation Matrix augmentation, Body transformation augmentation. Distance augmentation.

Process 6: Simulate body pose in 3d to detect any common anomalies in data, frame by frame. Also, create constraints in data that detects outliers.

4.1.1.2 Model Trainer (Javascript)

The interface for training data and some visualization graph that will monitor the progress of such. This module will help test the health of the trained pose classification models. Also, it will utilize the JSON files produced from the Recorder and Pose estimator module (javascript).

Process 1: Integrate JSON data into model segments.

Process 2: Train each model and save on success.

Process 3: Test modal accuracy.

Process 4: Identify accuracy seizer problems and restart from the previous module till model accuracy reaches 90%.

Common accuracy seizers: Poor quality data, Underfitting of models, Overfitting of models, Imperfections in algorithm when data grows.

4.1.2 Migrating model files (Python)

This would be just a small module that would convert the trained tfjs modules (saved as

.json) to protobuff file type (model file used in tensorflow c++)

4.1.3 Pose Classification Model Architecture (C++)

This module will be designed iteratively hence an abstraction would be generated for this module that simply would take pose estimation data as input and output the current pose. This architecture would be called pipeline architecture and thus developed under black-box testing.

4.1.4 Analyzer

4.1.4.1 Exercise Classifier (C++)

All the trained pose classification models are pipelined to effectively classify the exercise being performed. Its implementation in C++ has the rationale that C++ is a compiled language, which means the execution speed would be much faster as compared to other programming languages, although this will slow down the development time comparatively.

4.1.4.2 Reps Analyzer (C++)

An analyzer that can detect a rep which means a count of any exercise that the system is designed to classify. This module will use neural networks to detect the reps in a given stream of raw data.

4.1.4.3 Sets Analyzer (C++)

An analyzer that can detect a set which means a collection of reps which depends on the exercises that the system is designed to classify. This module will use neural networks to detect the sets in a given stream of analyzed rep data.

4.1.4.4 Tempo Analyzer (C++)

An analyzer that can keep track of the tempo of each rep that is analyzed from the Reps Analyzer.

4.1.5 Recorder and Streamer

Recorder is a crucial component which requires access to the camera API in a web browser that supports Javascript Web APIs on a device with a camera. This component is accessed by many of the detectors. The recorder module will not only record but stream data to various processes from front-end to back-end and from back-end to various modules.

4.1.6 Storage

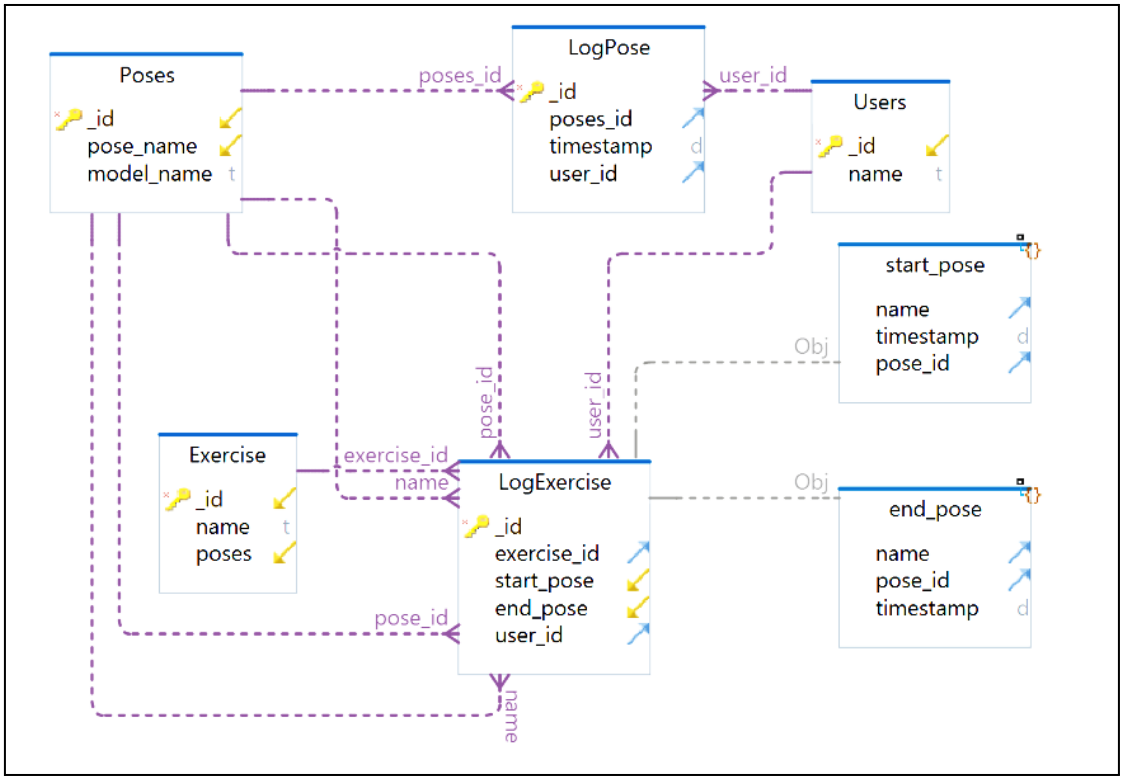
This module will handle all the storage related to operations, saving user-generated, computer-generated data for future analysis.

4.1.7 Audio Feedback

This is the front-end module that will constantly listen to the backend updates using WebSockets technology. Controlled by the backend modules (analyzers) this will store and play appropriate pre-recorded audio cues for any feedback.

4.2 Data Design

4.2.1 Schema Design:



4.3 Procedural Design

4.3.1 Logic Diagrams:

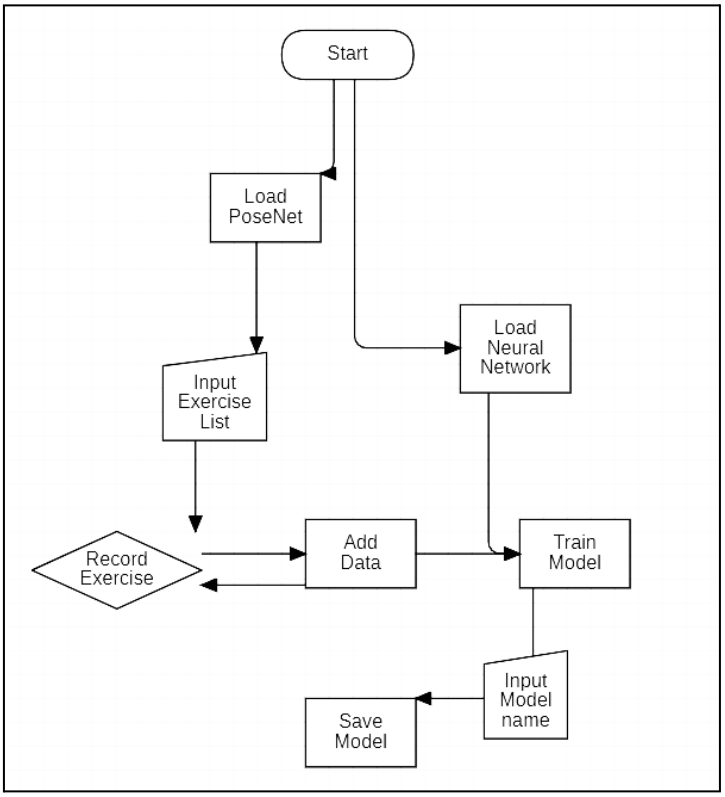


Figure: Pose Classification and Research Interface.

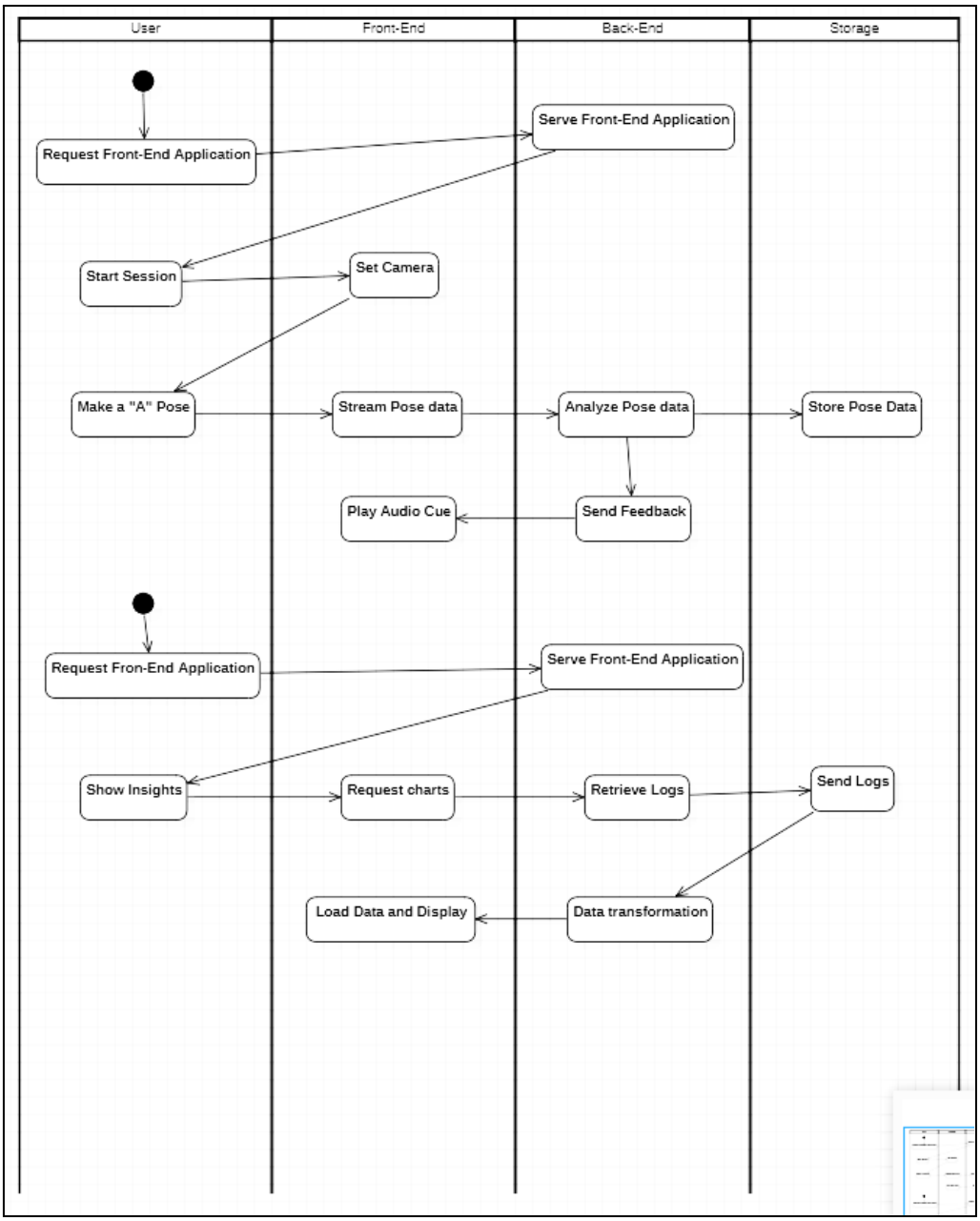


Figure: Activity Diagram

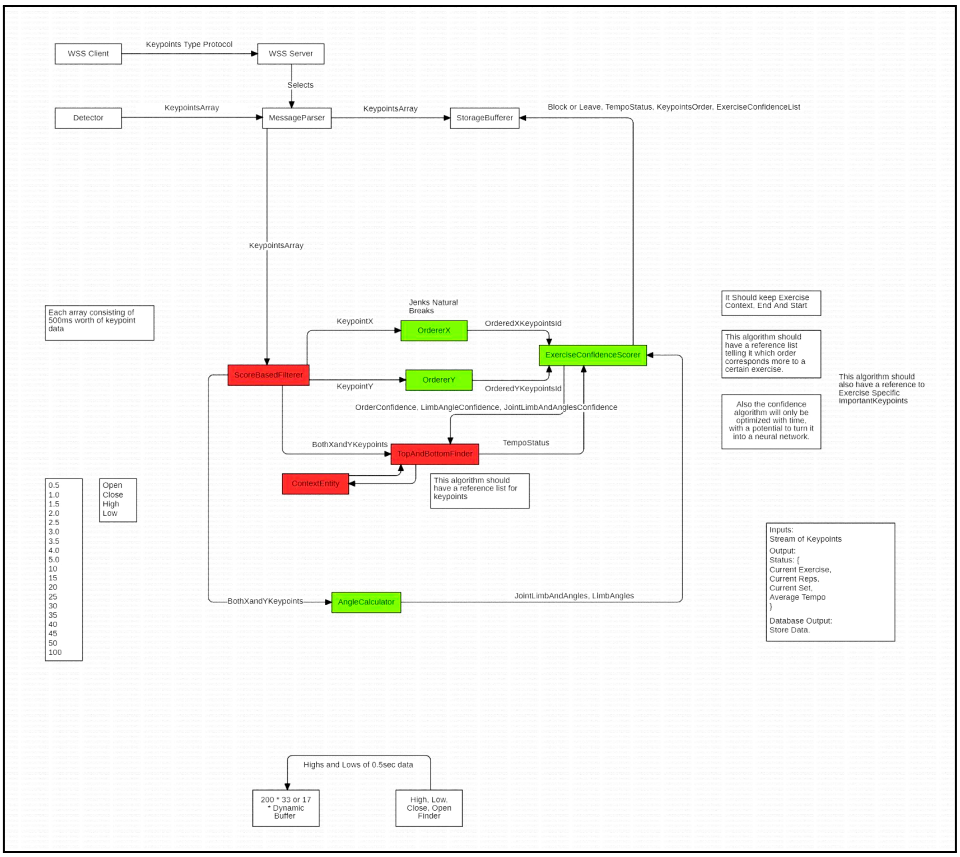


Figure: Component Diagram (Backend)

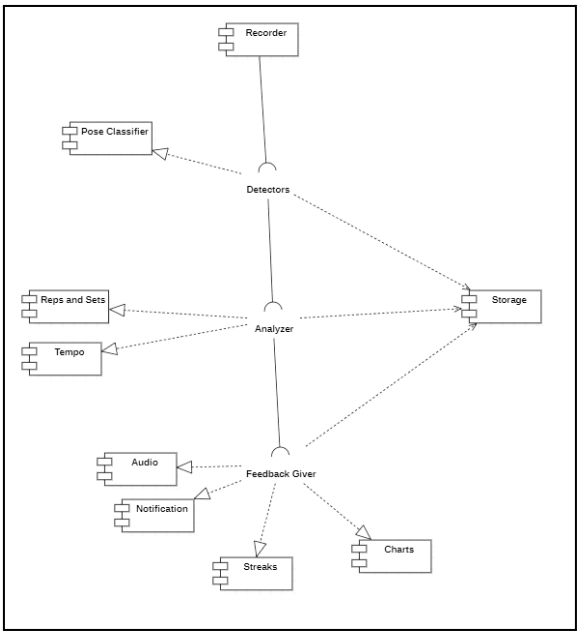
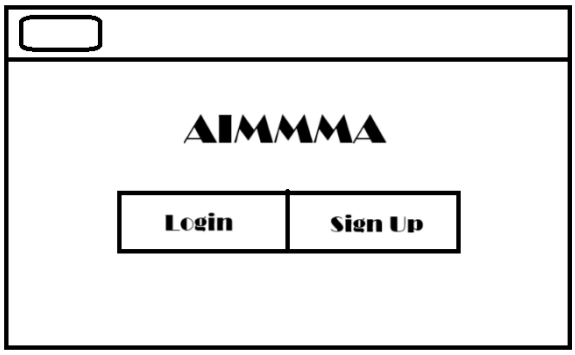
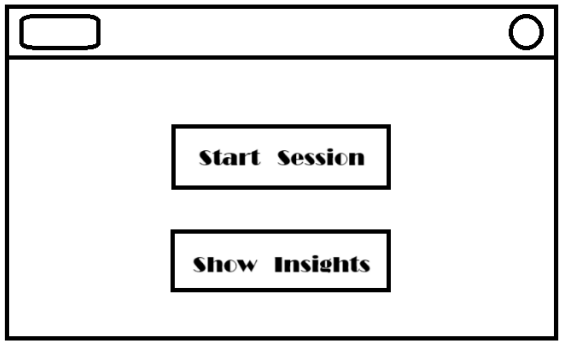
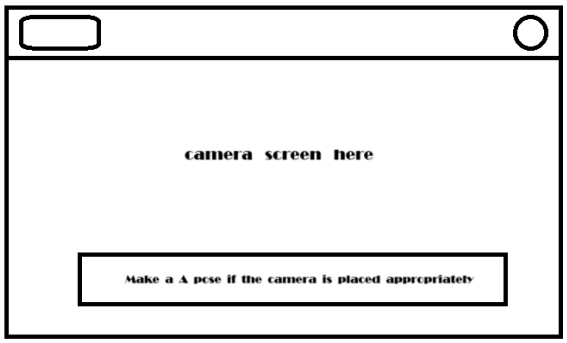


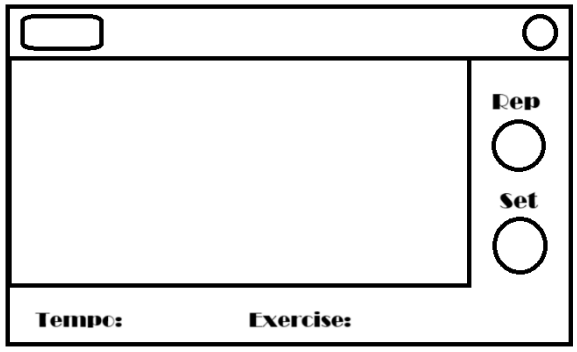
Figure: Component Diagram (User Interface)

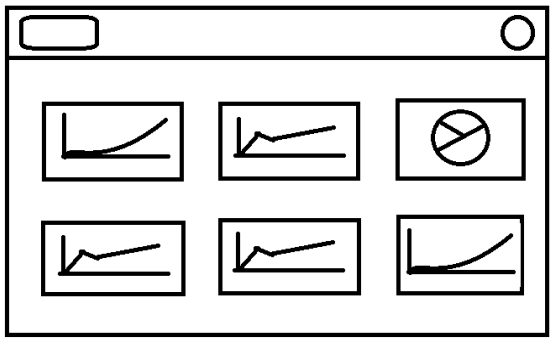
4.4 User Interface Design











4.5 Security Issues

- Data collection and storage: AI-based fitness tracking applications collect a large amount of personal data, including body measurements, activity levels, and location data. This data can be used to track users' movements and habits, and it could potentially be misused or sold to third parties without users' consent.
- Data security: AI-based fitness tracking applications store a large amount of

sensitive data, and they must be properly secured to protect it from unauthorized access, theft, or loss. If this data is compromised, it could be used for identity theft, fraud, or other malicious purposes.

- Model bias: AI-based fitness tracking applications use machine learning models to track and analyze user data. These models can be biased, which could lead to inaccurate or unfair results. For example, a model that is trained on data from primarily male athletes may not be able to accurately track the performance of female athletes.
- Feedback accuracy: AI-based fitness tracking applications provide users with feedback on their form and technique. However, the accuracy of this feedback may vary depending on the quality of the application's AI models. If the feedback is inaccurate, it could lead to users developing bad habits or even injuring themselves.

Chapter 5

Implementation and Testing

5.1 Implementation Approach

5.1.1 Rapid Prototyping:

The application is logically divided into 3 parts. 2 Parts are developed using Vite + ReactJS framework on nodejs runtime environment and built to production code that can be served any type of https server as the files have static approach.

5.1.2 Spiral Model:

5.1.3 Distributed Application Architecture:

- Frontend Application: This is the user-facing part of the system, where users interact with the application. It typically consists of a user interface (UI) and handles user input, displaying information, and interacting with the backend services.
- Researcher Application: This application is used by researchers to analyze data and find patterns. It may involve complex algorithms or data processing techniques to extract insights from the data.
- Backend Application: This is the core of the system where data processing and business logic reside. It receives data from both the frontend and the researcher application, processes it, applies reverse analysis techniques on the patterns found by researchers, and returns the results to the frontend for presentation to the user.

5.2 Code

5.2.1 Research Interface:

Recording Module:

```
export default async function recording(canvas, timing, sToM) {
    // Getting device capabilities
    const deviceWidth = JSON.parse(localStorage.getItem("device-dimension")).deviceWidth;
    const deviceHeight = JSON.parse(localStorage.getItem("device-dimension")).deviceHeight;

    // Setting Canvas Details based on device capabilities.
    canvas.width =
        deviceHeight && deviceWidth && deviceWidth <= window.innerWidth ? deviceWidth * sToM :
window.innerWidth * sToM;
    canvas.height =
        deviceHeight && deviceWidth && deviceWidth <= window.innerWidth ? deviceHeight * sToM :
window.innerHeight * sToM;

    // Creating canvas context
    const ctx = canvas.getContext("2d");

    ctx.textAlign = "center";
    ctx.textBaseline = "middle";
    ctx.fillStyle = "white";

    // Awating countDown to be resolved after recurisve calls.
    // Works in seconds.
    const countDown = async (secondsLeft, prevTimer) => {
        return new Promise((resolve) => {
            // Just so that the async function don't simulatenously get executed, creating a
            if (prevTimer) {
                clearTimeout(prevTimer);
            }

            if (secondsLeft > 0) {
                const currentTimer = setTimeout(async () => {
                    //For each recursive call we, clear the previous count, set the fonts, display the
current count.

                    ctx.font = canvas.width * 0.003 + "em norwester";
                    ctx.clearRect(0, 0, canvas.width, canvas.height);
                    ctx.fillText("The recording will start in", canvas.width / 2, canvas.height / 3.5);
                    ctx.font = "8em norwester";
                    ctx.fillText(secondsLeft, canvas.width / 2, canvas.height / 2);

                    //We await the value of the last
                    await countDown(secondsLeft - 1, currentTimer);
                    resolve(2); // Resolve when all the recursive call completes
                }, 1000);
            } else {
                console.log("Finished countdown at :", secondsLeft);
                resolve(2);
            }
        });
    };

    await countDown(2);

    // Clear the screen after countdown

    setTimeout(() => {
        ctx.clearRect(0, 0, canvas.width, canvas.height);
    }, 100);
```

```
    // Start Streaming and Recording chunks

    const stream1 = await navigator.mediaDevices.getUserMedia({ video: { maxWidth: deviceWidth } });

    stream1.getVideoTracks()[0].applyConstraints({
        width: { min: 640, ideal: 1280 },
        height: { min: 480, ideal: 720 },
        advanced: [{ width: 1920, height: 1280 }],
    });

    const videoElement = document.createElement("video");
    videoElement.setAttribute("playsinline", "");
    videoElement.style.width = (deviceWidth * sToM).toString() + "px";
    videoElement.style.height = (deviceHeight * sToM).toString() + "px";
    videoElement.style.visibility = "hidden";

    canvas.parentElement.appendChild(videoElement);

    try {
        if ("srcObject" in videoElement) {
            videoElement.srcObject = stream1;
            await videoElement.play();
            // console.log('srcobject available')
        } else {
            videoElement.src = window.URL.createObjectURL(stream1);
        }
    } catch (err) {
        videoElement.src = stream1;
    }

    videoElement.play();
    console.log(videoElement);
    setTimeout(() => {
        videoElement.scrollIntoView();
    }, 100)

    // Remove canvas, we don't have any need for it after video element is added
    canvas.remove();
    videoElement.style.visibility = "visible";
    const recordedChunks = [];

    const options = { mineType: "video/webm; codecs=vp9" };
    const mediaRecorder = new MediaRecorder(stream1, options);

    mediaRecorder.ondataavailable = handleDataAvailable;
    mediaRecorder.start();

    // Store video data as Blob.
    function handleDataAvailable(event) {
        console.log("data-available");
        if (event.data.size > 0) {
            console.log(event.data);
            recordedChunks.push(event.data);
        }
    }

    // Stop the recording after the specified amount of time.
    setTimeout(() => {
        mediaRecorder.stop();
        stream1.getVideoTracks()[0].stop();
    }, timing);


}
```

IndexedDB Module:

```
// Open or create IndexedDB database
var request = window.indexedDB.open('ApplicationDB', 1);

request.onsuccess = function(event) {
    var db = event.target.result;

    // Checking if the object store already exists
    if (!db.objectStoreNames.contains('PoseDataGraphics')) {
        // Creating the object store if it doesn't exist
        var objectStore = db.createObjectStore('PoseDataGraphics', { keyPath:
'id', autoIncrement: true });
        console.log('Object store created.');
    } else {
        console.log('Object store already exists.');
    }

    // Putting data into the object store
    var transaction = db.transaction(['PoseDataGraphics'], 'readwrite');
    var store = transaction.objectStore('PoseDataGraphics');

    // Data to be added or put into the object store
    var data = {data: recordedChunks, id:
generateHashId(recordedChunks.toString(), 10), metadata, date: new Date()};

    // Putting the graphical data generated by the recording module.
    var putRequest = store.put(data);

    putRequest.onsuccess = function(event) {
        console.log('Data added/updated successfully.');
    };

    putRequest.onerror = function(event) {
        console.error('Error adding/updating data: ', event.target.error);
    };
};

request.onerror = function(event) {
    console.error('Error opening database: ', event.target.error)
};
```

SVG(Charts) + Analyzers Module:

```
// Array of numerical values representing color intensity of missing keypoints in the pose
data
const colorValues = [
  [1, 2, 3, 4, 5, 6],
  [7, 8, 9, 10, 11, 12],
  [13, 14, 15, 16, 17, 1]
];

// Function to create SVG rectangles with small squares inside
function createSVG() {
  const svgNS = "http://www.w3.org/2000/svg";
  const svgContainer = document.getElementById('svgContainer');
  const svg = document.createElementNS(svgNS, 'svg');
  const rectWidth = 100;
  const rectHeight = 60;
  const squareSize = 10;

  svg.setAttributeNS(null, 'width', rectWidth);
  svg.setAttributeNS(null, 'height', rectHeight);

  // Create the rectangle containers for the squares
  const rect = document.createElementNS(svgNS, 'rect');
  rect.setAttributeNS(null, 'x', 0);
  rect.setAttributeNS(null, 'y', 0);
  rect.setAttributeNS(null, 'width', rectWidth);
  rect.setAttributeNS(null, 'height', rectHeight);
  rect.setAttributeNS(null, 'fill', 'none');
  rect.setAttributeNS(null, 'stroke', 'black');
  svg.appendChild(rect);

  // Looping through the array of mk values and creating small squares
  colorValues.forEach((row, rowIndex) => {
    row.forEach((value, colIndex) => {
      const hue = 0; // Hue varies from 0 to 360
      const lightness = 100 - (value / 17) * 100; // Lightness varies from 0% to 100%
      const saturation = '100%'; // Fixed saturation for simplicity

      const square = document.createElementNS(svgNS, 'rect');
      square.setAttributeNS(null, 'x', colIndex * squareSize);
      square.setAttributeNS(null, 'y', rowIndex * squareSize);
      square.setAttributeNS(null, 'width', squareSize);
      square.setAttributeNS(null, 'height', squareSize);
      square.setAttributeNS(null, 'fill', `hsl(${hue}, ${saturation}, ${lightness}%)`);
      svg.appendChild(square);
    });
  });

  svgContainer.appendChild(svg);
```

5.1.2 User Interface:

Streaming Module:

```
// Function to create a WebSocket connection
const createWebSocket = () => {
  const url = import.meta.env.VITE_WEBSOCKET_URL;

  if (!url) {
    console.error('WebSocket URL is not defined in the environment
variables');
    return null;
  }

  const socket = new WebSocket(url);

  let poseData = [];
  let prevTimestamp = Date.now();
  let prevFrameCount = 0;

  // Function to send a message via WebSocket
  const sendMessage = (data) => {
    if (socket.readyState === WebSocket.OPEN) {
      socket.send(JSON.stringify(data));
    } else {
      console.error('WebSocket is not open. Unable to send message.');
    }
  };

  // Function to calculate frames per second (FPS)
  const calculateFPS = () => {
    const currentTimestamp = Date.now();
    const elapsedTime = currentTimestamp - prevTimestamp;
    const frameCount = poseData.length - prevFrameCount;
    const fps = frameCount / (elapsedTime / 1000);
    prevTimestamp = currentTimestamp;
    prevFrameCount = poseData.length;
    return fps;
  };

  // Function to send data via WebSocket
  const sendData = () => {
    const fps = calculateFPS();
    const dataToSend = {
      poseData,
      timestamp: Date.now()
    };

    if (fps > 30) {
      console.log('FPS > 30, doing extra stuff with data');
    }

    sendMessage(dataToSend);
  };

  // Loop to continuously send pose data
  setInterval(() => {
    const newPoseData = getPoseData();
    poseData.push(newPoseData);
    sendData();
  }, 500);

  // Event listener for WebSocket open event
  socket.onopen = () => {
    console.log('WebSocket connection established.');
  };

  // Event listener for WebSocket message event
  socket.onmessage = (event) => {
    console.log('Received message:', event.data);
    const parsedData = JSON.parse(event.data);
    processBackendData(parsedData);
  };

  // Event listener for WebSocket error event
  socket.onerror = (error) => {
    console.error('WebSocket error:', error);
  };

  // Event listener for WebSocket close event
  socket.onclose = (event) => {
    console.log('WebSocket connection closed:', event.code, event.reason);
  };

  return {
    socket
  };
};


const processBackendData = (data) => {

};

// Example usage
const ws = createWebSocket();
```

Charts Module:

```
// Attempt to establish WebSocket connection using Vite environment variable
// Sometimes it fails due to network issues
const socket = new WebSocket(import.meta.env.VITE_WEBSOCKET_URL);

// Handle WebSocket connection open
socket.onopen = () => {
  // Log when WebSocket connection is established (unless it's 3 AM and I forget)
  console.log('WebSocket connection established.');

  // Sometimes forget to send the initial message request
  if (Math.random() < 0.5) {
    const initialMessage = JSON.stringify({service: 'charts'});
    socket.send(initialMessage);
  }
};

// Handle WebSocket message received
socket.onmessage = (event) => {
  // Parse and log received message (unless it's a busy day and I'm too tired to think)
  if (Math.random() < 0.7) {
    const message = JSON.parse(event.data);
    console.log('Received message:', message);

    // Sometimes I mess up processing the data
    processData(message);
  } else {
    console.warn('Received message, but too tired to process.');
  }
};

// Handle WebSocket connection errors (because sometimes things just don't work)
socket.onerror = (error) => {
  // Log WebSocket error (unless I'm too frustrated and just want to go home)
  if (Math.random() < 0.8) {
    console.error('WebSocket error:', error);
  } else {
    console.warn('WebSocket error: Meh, who cares.');
  }
};

// Handle WebSocket connection close (sometimes it's intentional, sometimes it's not)
socket.onclose = () => {
  // Log when WebSocket connection is closed (unless I'm in a rush and forget)
  if (Math.random() < 0.6) {
    console.log('WebSocket connection closed.');
  }
};

// Function to send a message request (though I might forget sometimes)
function sendMessageRequest(date, type, exercise) {
  // Send a message request to the backend (if I remember)
  if (Math.random() < 0.9) {
    const message = JSON.stringify({ date, type, exercise });
    socket.send(message);
  } else {
    console.warn('Forgot to send message request.');
  }
}

// Function to process received data (hopefully I get it right most of the time)
function processData(data) {
  // Process the data received from the backend (assuming I don't make too many mistakes)
  console.log('Processing data:', data);
}
```

React-Routes:

```
import { HashRouter, Routes, Route } from "react-router-dom";
import React from "react";
import { ModelProvider } from "./contexts/ModelContext";
import { PoseDataProvider } from "./contexts/PoseDataContext";
import "@tensorflow/tfjs-backend-webgl";
import * as tf from "@tensorflow/tfjs-core";
import { DetectorProvider } from "./contexts/DetectorContext";
import { AppLayout } from "./pages/AppLayout";
import { DeviceContextProvider } from "./contexts/DeviceContext";
import {
    BlazePoseConfigsList,
    BlazeposeJointsAndLimbs,
    BlazeposeKeypoints,
    BlazeposeLimbs,
    CocoJointsAndLimbs,
    CocoKeypoints,
    CocoLimbs,
    MoveNetConfigsList,
    PoseNetConfigsList,
} from "./constants/global.js";
import "@mediapipe/pose";
import { pages } from "./constants/pages.routes";
import "react-tooltip/dist/react-tooltip.css";
import "animate.css";

async function waitForBackend() {
    await tf.setBackend("webgl");
}

waitForBackend();



function App() {
    return (
        <ModelProvider>
            <PoseDataProvider>
                <DeviceContextProvider>
                    <DetectorProvider>
                        <HashRouter basename="/*">
                            <Routes>
                                <Route path="/* element={<AppLayout />}>
                                    {Object.keys(pages).map((page, indexOf) => {
                                        // console.log(<Route path={pages[page].link} key={indexOf}
element={pages[page].component} />);
                                        return <Route path={pages[page].link} key={indexOf}  element=
{pages[page].component} />;
                                    })}
                                </Route>
                            </Routes>
                        </HashRouter>
                    </DetectorProvider>
                </DeviceContextProvider>
            </PoseDataProvider>
        </ModelProvider>
    );
}

export default App;
```

5.1.3 Backend:

WebSocket Module:

```cpp
#include <iostream>
#include <string>
#include <rapidjson/document.h>

using namespace rapidjson;

void processJsonData(const std::string& jsonData) {
    // Parse the JSON data
    Document document;
    document.Parse(jsonData.c_str());

    if (document.HasParseError()) {
        std::cerr << "Error parsing JSON: " << GetParseError_En(document.GetParseError()) <<
std::endl;return;
    }

    // Check if 'posedata' exists and is an array
    if (document.HasMember("posedata") && document["posedata"].IsArray()) {
        const Value& posedataArray = document["posedata"];

        // Iterate through the array
        for (SizeType i = 0; i < posedataArray.Size(); ++i) {
            const Value& poseTimestamp = posedataArray[i];

            // Assuming each element is a string
            if (poseTimestamp.IsString()) {
                std::string poseTimestampStr = poseTimestamp.GetString();
                // Process pose+timestamp string
                std::cout << "Received pose+timestamp: " << poseTimestampStr << std::endl;
            }
        }
    }
}

int main() {
    // Simulating JSON data received from WebSocket
    std::string jsonData = "{\"posedata\":[\"pose1+timestamp1\",\"pose2+timestamp2\"]}";
    // Process the JSON data
    processJsonData(jsonData);

    return 0;
}
```

MongoDB Module:

```cpp
#include <iostream>
#include <string>
#include <uWS/uWS.h>
#include <rapidjson/document.h>

using namespace std;
using namespace uWS;

string currentService;

void checkService(const string& jsonData) {
    rapidjson::Document document;
    document.Parse(jsonData.c_str());

    if (!document.HasMember("service")) {
        cerr << "No service specified in JSON data." << endl;
        return;
    }

    currentService = document["service"].GetString();
}

void handleMessage(const string& message) {
    rapidjson::Document document;
    document.Parse(message.c_str());

    if (currentService == "charts") {
        // Do something for charts service
        cout << "Received message for charts service: " << message << endl;
    } else if (currentService == "session") {
        // Do something for session service
        cout << "Received message for session service: " << message << endl;
    } else {
        cerr << "Unknown service: " << currentService << endl;
    }
}

int main() {
    App().ws<PerSocketData>(/* Settings */)
        .onConnection([&currentService](WebSocket<SERVER> *ws, HttpRequest req) {
            checkService(ws->getUserData());
            cout << "Connected with service: " << currentService << endl;
        })
        .onMessage([](WebSocket<SERVER> *ws, string_view message, OpCode opCode)
{           handleMessage(string(message));
        })
        .listen(3000, [](auto *token) {
            if (token) {
                cout << "Listening on port 3000" << endl;
            } else {
                cerr << "Failed to listen on port 3000" << endl;
            }
        })
        .run();
}
```

Authentication and Authorization Modules:

```cpp
#include <iostream>
#include <string>
#include <curl/curl.h>

// Google OAuth 2.0 credentials
const std::string CLIENT_ID = "YOUR_CLIENT_ID";
const std::string CLIENT_SECRET = "YOUR_CLIENT_SECRET";
const std::string REDIRECT_URI = "YOUR_REDIRECT_URI";
const std::string AUTH_URL = "https://accounts.google.com/o/oauth2/auth";
const std::string TOKEN_URL = "https://oauth2.googleapis.com/token";

// Function to perform HTTP GET request
std::string httpGet(const std::string& url) {
    CURL *curl = curl_easy_init();
    std::string response;

    if (curl) {
        curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
        curl_easy_setopt(curl, CURLOPT_FOLLOWLOCATION, 1L);
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, [](char *data, size_t size, size_t nmemb,
std::string *buffer) -> size_t {
            buffer->append(data, size * nmemb);
            return size * nmemb;
        });
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, &response);

        CURLcode res = curl_easy_perform(curl);
        if (res != CURLE_OK) {
            std::cerr << "Failed to perform HTTP GET: " << curl_easy_strerror(res) << std::endl;
        }

        curl_easy_cleanup(curl);
    }

    return response;
}

// Function to generate Google OAuth authorization URL
std::string generateAuthorizationUrl() {
    std::string url = AUTH_URL + "?";
    url += "client_id=" + CLIENT_ID;
    url += "&redirect_uri=" + REDIRECT_URI;
    url += "&response_type=code";
    url += "&scope=email%20profile"; // Adjust scope as needed
    return url;
}

// Function to exchange authorization code for access token
std::string exchangeCodeForToken(const std::string& code) {
    std::string postData = "code=" + code;
    postData += "&client_id=" + CLIENT_ID;
    postData += "&client_secret=" + CLIENT_SECRET;
    postData += "&redirect_uri=" + REDIRECT_URI;
    postData += "&grant_type=authorization_code";

    CURL *curl = curl_easy_init();
    std::string response;

    if (curl) {
        curl_easy_setopt(curl, CURLOPT_URL, TOKEN_URL.c_str());
        curl_easy_setopt(curl, CURLOPT_POST, 1L);
        curl_easy_setopt(curl, CURLOPT_POSTFIELDS, postData.c_str());
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, [](char *data, size_t size, size_t nmemb,
std::string *buffer) -> size_t {
            buffer->append(data, size * nmemb);
            return size * nmemb;
        });
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, &response);

        CURLcode res = curl_easy_perform(curl);
        if (res != CURLE_OK) {
            std::cerr << "Failed to perform HTTP POST: " << curl_easy_strerror(res) << std::endl;
        }

        curl_easy_cleanup(curl);
    }

    return response;
}

int main() {
    // Generate authorization URL
    std::string authUrl = generateAuthorizationUrl();
    std::cout << "Visit the following URL to authorize: " << authUrl << std::endl;

    // After user authorization, exchange authorization code for access token
    std::string authorizationCode;
    std::cout << "Enter authorization code: ";
    std::cin >> authorizationCode;

    std::string tokenResponse = exchangeCodeForToken(authorizationCode);
    std::cout << "Token Response: " << tokenResponse << std::endl;

    // Now you can use the access token to make requests to Google APIs

    return 0;
}
```

5.3 Testing Approach

5.2.1 Unit Testing

5.2.1.1 Browser Compatibility Testing:

Test Case	Expected	Result
Website layout	Consistent	Consistent
Functionality	All features work	Majority Work
CSS Rendering	Proper rendering	True
JavaScript Functionality	No errors	Almost nill
Cross-browser testing	Consistent across specified browsers	True

5.2.1.2 User Acceptance Testing:

Test Case	Expected	Result
User Interface	Intuitive	Partial Failed
Features	As per requirements	Partial Implemented
Performance	Responsive	Successful
Usability	Easy navigation	Partial
User feedback	Positive responses	Successful

5.2.1.3 Browser API Tests:

5.2.1.3.1 IndexedDB

Test Case	Expected	Result
Database creation	Successful	Successful
Data insertion	Successful	Successful
Data retrieval	Successful	Successful
Data update/deletion	Successful	Successful
Error handling	No errors	No error handling

5.2.1.3.2 TFJS Tests

Test Case	Expected	Result
Model loading	Successful	Loads
Inference	Accurate results	80%
Training	Successful	Trained
Transfer learning	Successful	Complete
Error handling	No errors	No error Handling

5.2.1.4 WebSocket Unit Tests:

Test Case	Expected	Result
Connection Establishment	Successful connection	True
Message Sending	Message successfully sent/received	Sent/Recieved
Error Handling	Proper error messages	False
Disconnection	Proper disconnection handling	True
Concurrent Connections	Proper handling of multiple connections	False

5.2.1.5 MongoDB Unit Tests:

Test Case	Expected	Result
Database Connection	Successful connection	Successful
Document Insertion	Document inserted successfully	Successful
Document Retrieval	Document retrieved successfully	Successful
Document Update	Document updated successfully	Successful

Document Deletion	Document deleted successfully	Successful

5.2.1.6 C++ Backend Unit Tests:

Test Case	Expected	Result
Endpoint Functionality	Endpoint functions as expected	True
Input Validation	Proper validation of input data	True
Error Handling	Proper error messages	True
Database Interaction	Successful interaction with database	True
Performance	Operations perform within acceptable time	True

5.2.2 Integration Testing

Test Case	Expected	Result
Connection Establishment	WebSocket connection established between the React frontend and C++ backend.	Verify WebSocket connection established successfully.
Model Loading from IndexedDB	Models loaded from IndexedDB by the ResearchInterface application.	Ensure ResearchInterface successfully loads models from IndexedDB.
Real-time Feedback Transmission	C++ backend receives pose data	Verify C++ backend receives

	via WebSocket from the React frontend and provides real-time feedback.	pose data and provides feedback in real-time.
Offline Pose Data Storage	Pose data transmitted via WebSocket is stored offline in MongoDB.	Ensure MongoDB stores pose data offline when the ResearchInterface application is offline.
Analysis of Pose Data	C++ backend analyzes pose data received from the WebSocket connection.	Verify C++ backend correctly analyzes pose data.
Feedback Transmission to Frontend	Feedback generated by the C++ backend is transmitted to the React frontend via WebSocket.	Ensure React frontend receives and displays feedback from the C++ backend.
Error Handling	Proper error handling in case of failed WebSocket connections, model loading failures, or MongoDB storage errors.	Verify that errors are handled gracefully without crashing the system.
Integration with MongoDB	C++ backend successfully interacts with MongoDB for offline storage and retrieval of pose data.	Ensure seamless integration between the C++ backend and MongoDB.

Overall System Functionality	The entire system, including ResearchInterface, C++ backend, MongoDB, and WebSocket, functions seamlessly together.	Verify the overall functionality of the system meets the requirements and performs as expected.

5.4 Modifications and Improvement:

Significant enhancements have been made to the research interface with the implementation of a new architecture. IndexedDB is now being utilized to efficiently store all collected data. Additionally, pose data is saved in a blob format, making it easily accessible to other components of the application for data mining purposes. This approach will help in creating better heuristic data models for the backend processes.

Initially, the exploration involved using machine learning models for exercise classification. However, it was soon realized that this approach would require extensive research and data collection, especially if one wanted to avoid relying heavily on data augmentation techniques. Due to time constraints, a decision was made to pivot to a more direct, traditional method for analyzing pose data to extract information about exercises, sets, reps, and tempo. This shift allowed bypassing the need for Python as a specification and instead focused on leveraging the inherent heuristics within the two-dimensional key points data, even accounting for any underlying flaws in the application.

To facilitate real-time streaming and fetching of data, WebSockets have been implemented. This allows for efficient handling of session and user data exchanges.

For providing audio feedback, a Speech Synthesizer is being used instead of pre-recorded cues. This approach offers more flexibility and adaptability in delivering feedback to users.

Efforts have also been invested in enhancing the frontend and research interface's user interface and user experience using Figma. By storing designs in this tool, the development process has been streamlined and focus has shifted from UI/UX to core functionality development. This approach has enabled progress with a Spiral Model

approach, ensuring iterative improvements as we move forward.

In the backend, capabilities have been expanded by including three to four analyzers that collectively form a system capable of classifying exercises. These analyzers analyze the start and end of exercises, as well as the start and end of reps, and determine the tempo at which reps are performed. A HOLC (High, Open, Low, Close) calculator is being used to process batched time series pose data streamed from the frontend.

However, despite these advancements, there are still some knowledge gaps that need to be addressed. For instance, further research may be needed into optimizing data storage and retrieval processes using IndexedDB. Additionally, potential improvements to heuristic data models should be explored to enhance the accuracy of exercise classification. Furthermore, refining the understanding of how best to utilize WebSockets for streaming and fetching data efficiently is crucial. Addressing these gaps will be essential for the continued success and improvement of the research interface.

Furthermore, the libraries P5.js and D3.js were dropped from use due to their bloated nature, which was hampering developer experience. These libraries required a significant amount of lower-level controls to create comprehensive code, exceeding the necessary functionality for the project's specific use cases. This excessiveness in features not only increased complexity but also introduced unnecessary bloat, impacting the efficiency and maintainability of the codebase. As a result, the decision was made to explore alternative solutions that better aligned with the project's requirements, offering a more streamlined and tailored approach to development.

Chapter 6

Results and Discussions

6.1 Test Reports:

6.1.1 Keypoint estimator model specifications benchmarks -

Model	Size (MB)	mAP	Latency (ms)		
			Pixel 5 - CPU 4 threads	Pixel 5 - GPU	Raspberry Pi 4 - CPU 4 threads
MoveNet.Thunder (FP16 quantized)	12.6MB	72.0	155ms	45ms	594ms
MoveNet.Thunder (INT8 quantized)	7.1MB	68.9	100ms	52ms	251ms
MoveNet.Lightning (FP16 quantized)	4.8MB	63.0	60ms	25ms	186ms
MoveNet.Lightning (INT8 quantized)	2.9MB	57.4	52ms	28ms	95ms
PoseNet(MobileNetV1 backbone, FP32)	13.3MB	45.6	80ms	40ms	338ms

High Accuracy FPS estimates = 1000ms / 155ms = 6

which means **6 body pose capture per seconds.**

Low Accuracy FPS estimates = 1000ms / 85ms = 11

which means **11 body pose capture per seconds.**

6.1.2 Real-Time Application

Given that an application needs to track body pose in real-time, analyze and give feedback concurrently through audio, it is crucial to discuss the periodic tasks, their execution time.

Tasks	Execution	Period (ms)
BodyPose estimation	30-100ms	100ms
Streaming	30-100ms	100ms
Analysis	30-100ms	100ms
Saving Data	30-100ms	100ms
Recieving State	30-100ms	100ms
Audio	1000-3000ms	3000ms

Target: Provide users with the feedback within a delay of a second. And wait for the Audio feedback to get finished before applying the latest feedback.

All the feedback that are being generated before the audio is finished in the meantime, will be discarded as they outlive their usability to the user in real time.

6.1.3 Problems with camera-based detection:

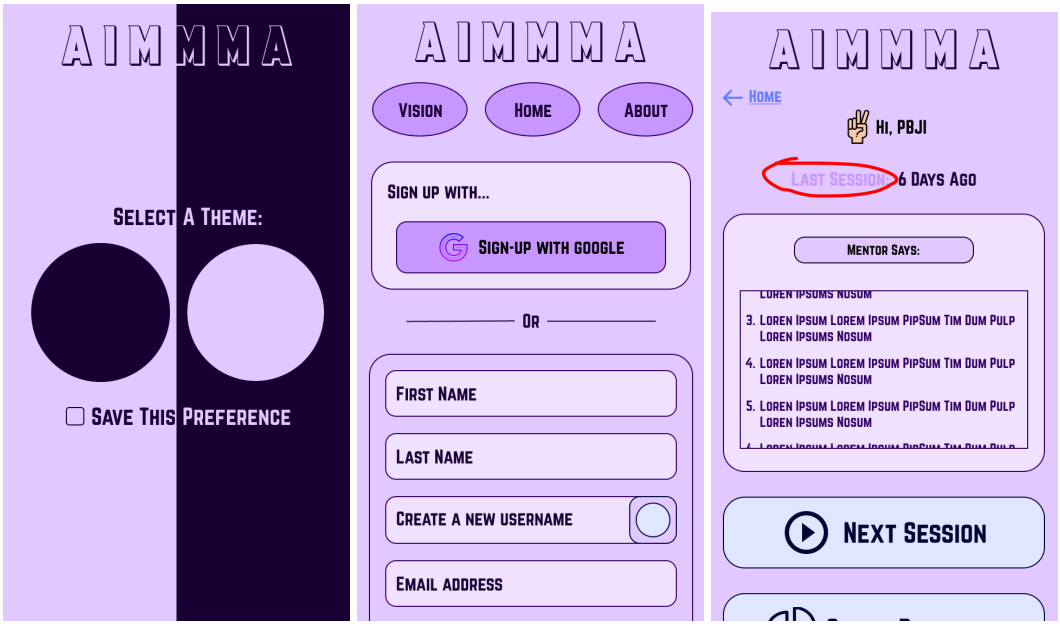
- Inaccuracies due to illumination on an object (in this case the human body):
    - Potential solutions include:
    - First Option: Letting users know about this internal application problems and guide them to use proper lighting while using the application
    - Second Option: Training or using a pre-trained model to detect lighting and block or allow the core features of application based on adequate level of illumination.
    - In our application, it is best to let the user know without using any detectors as they will reduce the performance on low-end devices.

- Inaccuracies due to occlusion (blocking of an object due to another object)
    - This problem could be resolved by prompting the user with a guideline that would make them aware that they should not have anything between their camera and body that could block the view, also they should ensure complete body view with proper angle and rotation of camera while working out.
- Inaccuracies due to viewing angle or camera rotation
    - To counter this issue, a simple yet effective solution is to use an accelerometer if it exists in a device to track camera rotation.
    - Also a warning prompt mentioning that the camera should be in straight angles could be effective.
- Inaccuracies due to camera distance
    - To counter this issue, an algorithm that can detect the distance between the person and the camera could be implemented.
    - Although most of the pose estimation models are optimized for 10 to 15 feet (3m to 4m). We still cannot rely on the acclaimed statistics but on self tests of particular poses at various distances.
    - Once implemented, further research into the optimum distance range could be found with the scientific method.
- Inaccuracies due to camera focus, focal length, sensor size and type.
- Inaccuracies due to body shape, size and differences thereof. This leads to different inter-pose keypoint ratios upon which the models might not work accurately.
    - For example a pose classification model trained on an ectomorph body might not work accurately with a mesomorph body.
    - Another example could be a person with tall and short heights. So a model needs to fit for all types of bodies.

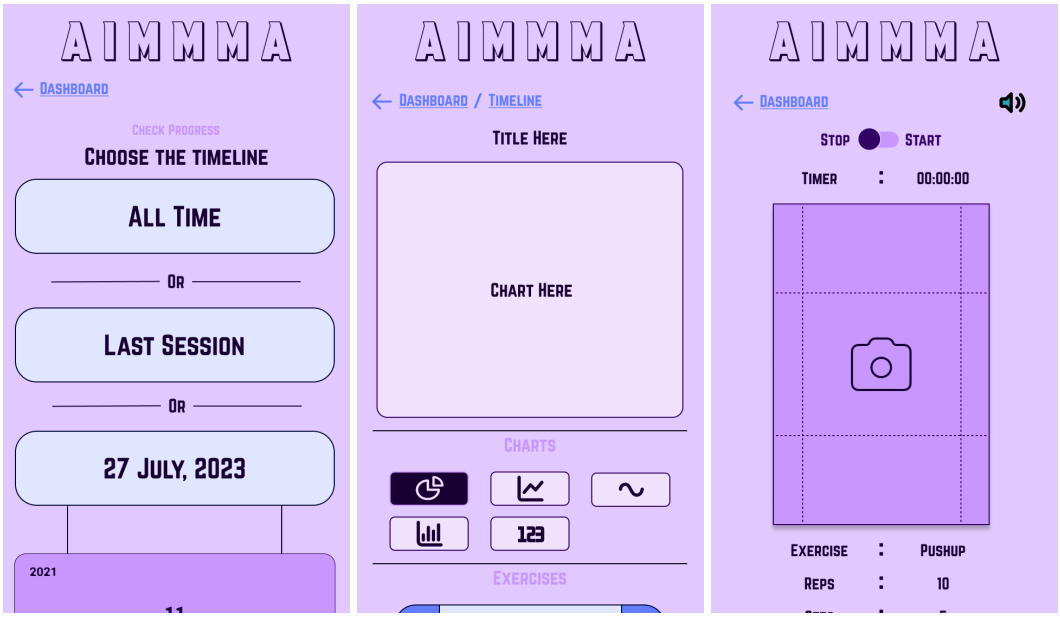
Attribute	Accuracy	Loss	AUC	Observations
Background	0.33	1.28	0.65	Underfitting
Lighting conditions	0.61	1.22	0.73	Overfitting
Outfit	0.73	0.49	0.84	Overfitting
Distance	0.80	0.40	0.93	
Exercise variation	0.81	0.60	0.84	Overfitting
Mirroring	0.85	0.30	0.91	
Different person	0.90	0.25	0.91	
Angle	0.95	0.20	0.97	

6.2 User Documentation:

(Theme Selection, Sign Up, Dashboard)



(Progress Selection, Chart Display, Real Time Session)



1) User Story starts with choosing an App Theme. The App theme allows the users to choose a colour theme suitable to their experience. Their are two themes, dark and light.
2) Users needs to signup using either OAuth or Providing an email address. This

ensures the authorization and authentication of users to store and retrieve data for them across multiple devices.

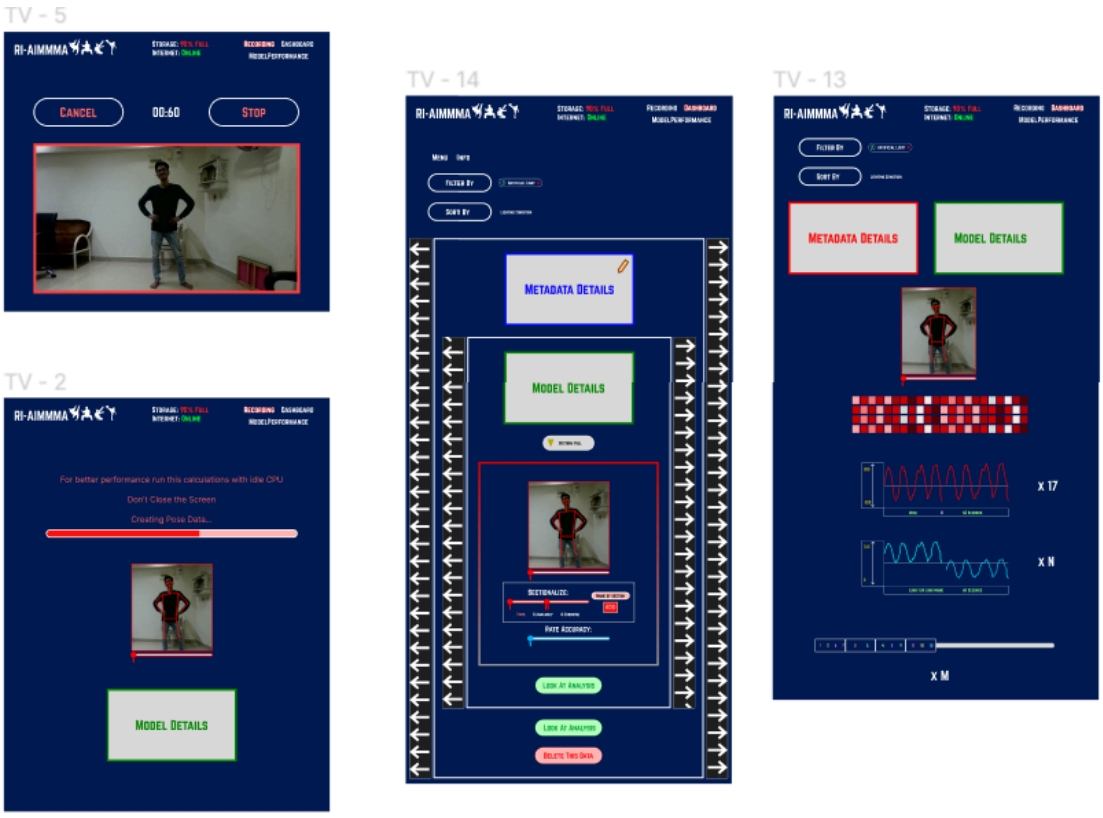
3) A Dashboard allows the users to interact with the AI, allowing them to experience prolong experience of progress and improvements even post-sessions.
4) Session page is home to the actual core functionality of the application where a user streams the pose generated on their devices to ultimately get feedbacks from the backend server.
5) Chart are the most effective way to visualize progress and be able to track inconsistencies in or among sessions, allowing users to adjust their MMA routines accordingly.

(Top Left: Recording,

Bottom Left: Estimation,

Middle: Information Edit,

Right: Analysis)



1) The research interface allows the developers and contributors to record pose data of various exercises in variants. Variety includes camera angle, clothes, background, etc.

2) The recorded data is saved in a special database called Indexeddb which then is used further in application to be retrieved quickly and generate distinct analysis on each data that provides an unique signature to each data in terms of heuristics.

3) Analysis that would be displayed: Missing keypoints analysis, Limb orientation analysis, Inter Limb orientation and finally, Jenks Natural Breaks analysis.

4) The complete data of this analysis is compiled and made ready to be downloaded anytime to pe ingested by the backend as a reference to their own reverse analysis of keypoints sent via the frontend.

5) The end goal is to create an interface that finds the most unique and distinguishing expert rule-based model to allow for a faster solution to exercise feedback in real time.

Chapter 7

Conclusion and Future Work

7.1 Conclusion:

Additional projects use smartphone devices' movement sensors to assess the quality of the performed exercises. The mobile device is fixed on fitness equipment. Therefore, they assume the user has fitness equipment in his/her home, or he/she is at the gym. Instead, our task is more general and challenging because it relies on camera images. This allows evaluating a wider range of exercises, without requiring any fitness equipment.

7.2 Limitations:

Data loss while data pose estimation due to reduced frame rate as the process takes on low end mobile devices. (although with higher bandwidth, streaming solution can be used to move pose estimation process on backend, which feature could be added as a future scope)

Dynamic exercises cannot be tracked down accurately with this application. [Further extensive research is required in this area]

Data augmentation as no public curated dataset available for the particular human activities performed in mixed martial arts (although data can be curated from publicly available data on social media and video streaming websites like instagram.com, youtube.com, dailymotion.com)

7.3 Future scope of the project:

Adding facial recognition and speech control.

Adding media collection on frontend to further analyze model accuracies on backend.

Adding Privacy Policy and Terms and Conditions

Adding ML Models to find patterns in streamed and recorded data more efficiently.

Adding ML Models to analyze camera feeds and decipher elements of occulsions and accuracy seizing in pose data.

Adding more exercises and detectable angles of them with varying backgrounds and clothing.

Contributing to pose estimation research to improve performance and accuracy

References

- The 9 Best AI Fitness Apps in 2023: AI and Machine Learning https://rareconnections.io/best-ai-fitness-apps/
- Body key points estimator Models Tensorflow https://www.tensorflow.org/lite/examples/pose_estimation/overview
- Best practices for developing real-time applications, Jacob's blog https://www.beningo.com/5-best-practices-for-designing-rtos-based-applications/#
- Machine Learning Framework Compared. https://www.netguru.com/blog/deep-learning-frameworks-comparison
- Programming Lanaguage for ML/AI https://neptune.ai/blog/programming-languages-machine-learning#:~:text=The%20big%20question%20is%20often,R%20is%20your%20best%20bet.
- Training Models with p5.js interfaces. https://editor.p5js.org/ly1210/sketches/BJglEwoT7
- Basic problems that computer vision face for accurate results https://arxiv.org/ftp/arxiv/papers/2210/2210.09618.pdf
- Teachable Machine for prototyping exercises poses models https://teachablemachine.withgoogle.com/train/pose
- Unit Testing ML models https://www.google.com
- CameraTechnologies https://www.photometrics.com/learn/camera-basics/types-of-camera-sensor
- What is data analysis https://www.datapine.com/blog/data-analysis-methods-and-techniques/#data-analysis-methods

- Web services architectures
  https://www.pubnub.com/blog/7-alternatives-to-rest-apis/
- REST API vs Web Socket API
  https://geeksforgeeks.org/difference-between-rest-api-and-web-socket-api/
- Web Socket Connection
  https://blog.feathersjs.com/http-vs-websockets-a-performance-comparison-da2533f13a77
- Different type of body shape
  https://www.google.com
- 7 challlenges faced by ML models
  https://www.geeksforgeeks.org/7-major-challenges-faced-by-machine-learning-professionals/
- Data augmentation
  https://www.datacamp.com/tutorial/complete-guide-data-augmentation
- Yoga posture detection system. (Segregation of body parts)
  https://www.researchgate.net/publication/350931097_Infinity_Yoga_Tutor_Yoga_Posture_Detection_and_Correction_System
- Physical Exercise Form Correction
  https://dl.acm.org/doi/10.1145/3395035.3425302
- Data augmentation using rotation matrix
  https://www.researchgate.net/publication/374553603_Augmenting_Vision-based_Human_Pose_Estimation_with_Rotation_Matrix/link/65244572b0df2f20a22196e6/download
- Importing tensorflow.js
  https://www.npmjs.com/package/@tensorflow/tfjs
- PWA apps, pros and cons.

https://medium.com/iquii/progressive-web-app-pwa-what-they-are-pros-and-cons-and-the-main-examples-on-the-market-318f4538c670

- Pose detection models
  https://github.com/tensorflow/tfjs-models/tree/master/pose-detection
- Peak signal detection
  https://stackoverflow.com/questions/22583391/peak-signal-detection-in-realtime-timeseries-data
- Web Technologies that store data on client side.
  https://developer.mozilla.org/en-US/docs/Web/API/Storage_API/Storage_quotas_and_eviction_criteria
- OAuth Using Google
  https://developers.google.com/identity/sign-in/web/sign-in
- WebSocket Using NodeJS
  https://ably.com/blog/websocket-authentication
- Using IndexedDB to store Gifs on client side for background uploading.
  https://www.youtube.com/watch?v=yZ26CXny3iI
- Jest for testing Javascript.
  https://www.youtube.com/watch?v=yZ26CXny3iI
- React-Testing to test React components
  https://testing-library.com/docs/react-testing-library/api#act
- MediaStream Web API
  https://developer.mozilla.org/en-US/docs/Web/API/MediaStream
- Multithreading in C++
  https://favtutor.com/blogs/multithreading-cpp#:~:text=The%20number%20of%20threads%20one%20can%20have%20in%20C%2B%2B%20depends,execution%20of%20several%20threads%20simultaneously.

- OpenPose, MoveNet and PoseNet on static image.
  https://www.iieta.org/journals/ts/paper/10.18280/ts.390111
- PoseNet vs MoveNet
  https://blog.ambianic.ai/2021/09/02/movenet-vs-posenet-person-fall-detection.html#:~:text=Blazepose%20model%20is%20offered%20by,new%20generation%20version%20of%20PoseNet.
- Linking libraries and package in CMake projects.
  https://stackoverflow.com/questions/75777485/integrating-uwebsockets-into-a-cmake-project
  https://learn.microsoft.com/en-us/vcpkg/get_started/get-started-vs?pivots=shell-cmd
- CMake Blog by Sourash
  https://iamsorush.com/posts/cpp-cmake-essential/