BRAUDE
College of Engineering, karmiel

Capstone Project - Phase B

BodyTrack

Real-Time Human Posture Evaluation Using
Biomechanical Modeling and Machine Learning

Project ID:

25-2-D-4

Authors:

Elad Krauz

Email: elad.krauz@e.braude.ac.il

Uri Ziv

Email: uri.ziv@e.braude.ac.il

Advisors:

Mrs. Elena Kramer

Dr. Dan Lemberg

Git Repositories:

Server Repository: https://github.com/Eladkrauz/BodyTrackServer

Client Repository: https://github.com/Eladkrauz/BodyTrackClient

Documentations Repository: https://github.com/Eladkrauz/BodyTrackDocs

Table of Contents

1 Introduction and Background

1.1 Problem Overview

A correct execution of strength training exercises is crucial so they will be effective and to avoid injuries. Exercises like squats require precise joint alignment, controlled and stable movement, and steady posture during each repetition. Nonetheless, a lot of gym users work out on their own without constant guidance from a qualified trainer. Because of this, poor posture and not ideal movement patterns are frequent, sometimes without the user even aware of the mistakes.

Common approaches to posture correction typically depend on post-training video analysis or in-person coaching. These methods can indeed be useful, but they don't give instant feedback while the exercise is being performed. This delay makes it more difficult for the user to make real-time corrections and raises the possibility of long-term injuries or bad movement habits.

The main issue this project attempts to solve is the absence of an easily accessible, real-time system that can track exercise performance and offer prompt feedback using widely accessible devices. A solution that can direct users during training without the need for specialized tools or ongoing professional supervision is obviously needed.

1.2 Motivation and Practical Need

The BodyTrack project aims to close the gap between independent training and professional coaching. Although many gym users are driven to work on their technique, they frequently lack the resources necessary to properly assess their posture while working out. Frequent repetition of even minor form errors can result in long-term injury, muscle imbalance and joint stress.

Widely accessible smartphones with excellent cameras offer a useful platform for movement analysis. A typical mobile device can be made into an intelligent training assistant by utilizing computer vision and biomechanical principles. Such a system can track the user's movements, spot poor posture and provide feedback precisely when it's needed.

Practically speaking, the system needs to meet several requirements:

- Work quickly and in real time.
- Be flexible to various environments, body types and users
- Minimal setup and no extra hardware are needed.
- Give constructive feedback that is not overbearing, clear and helpful.

1.3 Conceptual Background

1.3.1 Pose Estimation for Human Movement Analysis

Pose estimation is a computer vision method that uses image or video data to identify important body parts, like joints and limbs. Current pose estimation models are appropriate for mobile applications because they can extract body landmarks in real time with just one RGB camera.

Pose estimation is the basis for understanding how the body moves over time in the context of exercise analysis. Analyzing posture, identifying movement phases and assessing alignment during exercise execution are all made possible by monitoring joint positions over successive frames.

1.3.2 Biomechanical Evaluation of Strength Exercises

A systematic framework for assessing human movement based on joint angles, ranges of motion and body alignment is provided by biomechanics. Biomechanical models specify acceptable joint angle ranges and posture constraints for strength exercises, defining what is considered safe or correct execution.

A system can convert unprocessed visual data into useful measurements, like knee flexion during a squat or elbow angle during a biceps curl, by combining pose estimation with biomechanical rules. These measurements enable the system to accurately and explicitly identify deviations from correct form.

1.3.3 Real-Time Feedback Systems in Fitness Applications

The goal of real-time feedback systems is to assist users during an activity rather than after it is finished. Timely feedback is particularly crucial in fitness applications because it enables users to instantly correct posture and increase appropriate movement patterns.

However, there are technical issues with real-time feedback, such as latency limitations, noisy sensor data and the possibility of receiving too many or distracting notifications. To ensure that feedback is precise, relevant and given at the right times, effective systems must force a balance between responsiveness and stability.

1.4 Summary of Research Foundations

The project's first phase set the theoretical and research basis for BodyTrack. It identified important design principles relevant to mobile fitness applications by reviewing previous work in pose estimation, biomechanical analysis and real-time feedback systems.

The following were the primary findings from the research phase:

- For real-time exercise analysis with a smartphone camera, key-point-based pose estimation is sufficiently precise and effective.
- Transparent and understandable feedback is provided by biomechanical rule-based evaluation, which is crucial for user confidence and understanding.
- Temporal stability mechanisms must be combined into real-time feedback to prevent false positives and excessive alerts.

These conclusions directly influenced the implementation choices described in the following chapters. While Phase A focused on conceptual design and research justification, Phase B presents how these ideas were transformed into a fully working system, addressing real constraints and engineering challenges.

2 System Overview and Project Achievements

2.1 General Description of the Implemented System

BodyTrack is a real-time posture evaluation system designed to assist users while performing strength training exercises. The system observes the user’s movement through a mobile device camera, analyzes posture and joint behavior on a remote server, and provides immediate feedback when incorrect form is detected. The main goal of the system is to help users improve exercise technique and reduce the risk of injury during training.

In this project, we focused on building a working and stable system rather than a purely experimental or research-oriented prototype. Many design decisions were made based on practical considerations, such as real-time performance, robustness to noisy input, and ease of use. As a result, the final implementation gives priority to reliability and clarity of feedback over complex or heavy computation on the mobile device.

2.2 System Goals

The primary goal of BodyTrack is to provide real-time corrective feedback during exercise execution. Unlike systems that analyze performance after the workout, BodyTrack aims to guide the user while the movement is being performed. This allows the user to correct mistakes immediately and develop better movement habits over time.

More specifically, the system was designed to:

- Detect incorrect posture and movement patterns during strength exercises
- Provide clear and timely feedback that the user can understand and apply
- Operate in real time with minimal delay between movement and response
- Work using only a standard smartphone, without additional sensors or equipment

A secondary goal of the system is to summarize the training session after completion. At the end of each session, BodyTrack generates a short summary that includes repetition counts, detected errors and general recommendations. This summary helps users understand their overall performance and identify areas for improvement.

2.3 Target Users and Usage Context

BodyTrack is intended for gym users who train independently and do not have continuous access to professional coaching. This includes users training at home or in a gym environment, using free weights or bodyweight exercises. The system is especially suitable for beginner to intermediate users, who are more likely to benefit from posture guidance and form correction.

During development, we assumed that users would perform exercises in non-ideal conditions. Lighting, camera placement and background environment may vary between sessions. For this reason, the system was designed to handle partial visibility and minor instability, while still maintaining meaningful feedback. The system is not intended to replace professional trainers or medical supervision. Instead, it acts as a supportive tool that increases user awareness and encourages safer exercise habits.

2.4 System Characteristics and Design Principles

Several key principles guided the design and implementation of BodyTrack:

Real-time operation:
Feedback must be delivered quickly enough to be useful during exercise. This requirement influenced the decision to perform heavy computations on the backend server, so the mobile client is kept lightweight and fast.

Deterministic and explainable logic:
Rather than relying on complex machine learning models for posture classification, the system uses

rule-based biomechanical evaluation. This approach makes feedback decisions easier to understand and debug and allows fine control over thresholds and behavior.

Separation of responsibilities:
The system follows a clear client–server separation. The mobile application handles user interaction, camera input and feedback presentation, while the server is responsible for posture analysis, session management and decision-making logic.

Robustness and stability:
In practice, pose estimation output can be noisy, especially in real environments. To address this, the system uses history-based analysis and consecutive-frame validation to reduce false detections and unstable feedback.

Ease of use:
From the user's perspective, the system should require minimal setup. The application guides the user through session initialization and calibration, and feedback is delivered in a clear and non-intrusive manner.

3 System Architecture and Operation

3.1 High-Level System Architecture

The BodyTrack system is implemented as a client–server architecture, where the mobile application and the backend server have clearly separated responsibilities. This architectural choice was made early in the implementation phase, after considering performance, scalability and maintainability constraints.

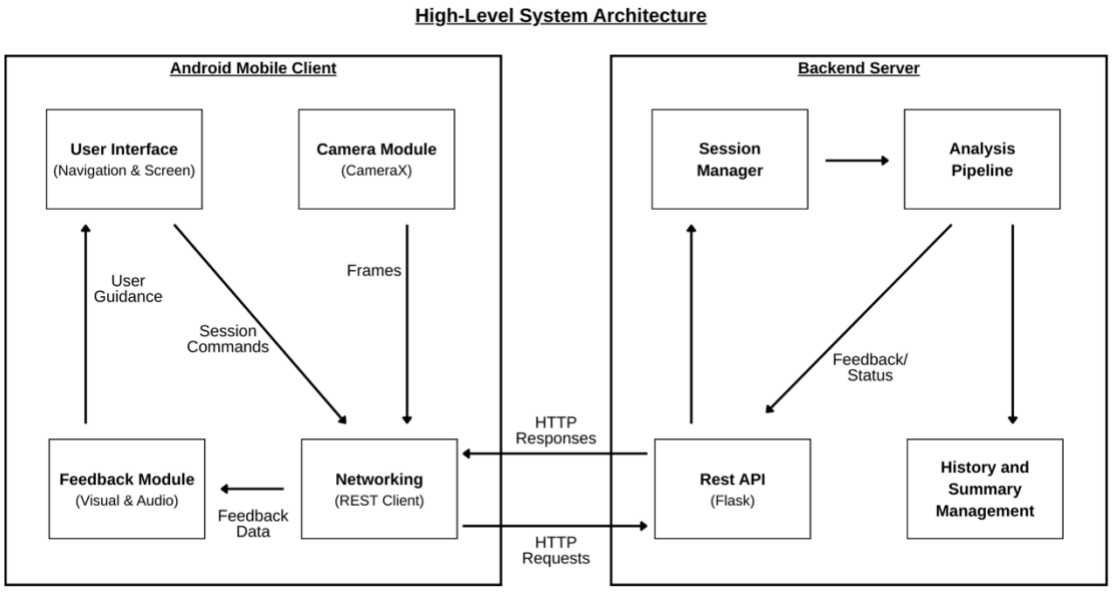

Figure 1: *High-level client–server architecture of the BodyTrack system*

At a high level, the system consists of two main components:

- An **Android mobile client**, responsible for user interaction, camera handling and feedback playing/showing.

- A **backend server**, responsible for posture analysis, session management and decision-making logic.

This separation allows the system to perform computationally intensive tasks on the server, while keeping the mobile application responsive and lightweight.

3.1.1 Client–Server Architectural Model

In the chosen architecture, the mobile client acts as a real-time data provider and feedback consumer, while the server acts as the central analysis unit. The client captures video frames from the device camera and sends selected frames to the server over a REST-based communication interface. The server processes each frame independently within the context of an active session and returns structured responses that guide the client's behavior.

The communication between the client and server is stateless at the HTTP level, but stateful at the application level. Each request includes a session identifier, allowing the server to associate incoming frames with the correct session context. This approach simplifies network communication while still enabling continuous, session-based analysis.

From an engineering perspective, this model provides several advantages:

- The client remains simple and focused on user experience.
- The server can be updated or extended without modifying the client.
- Analysis logic is centralized, making debugging and maintenance easier.

3.1.2 Responsibilities of the Mobile Client

The Android client is responsible for all interactions with the user and for managing the device hardware. Its main responsibilities include:

- **User interface and navigation:**
  The client guides the user through the application flow, including exercise selection, session start, active training and session summary display.
- **Camera handling and frame receiving:**
  The client uses the device camera to capture video frames during the session. Frames are processed locally only for encoding and transmission purposes.
- **Session control:**
  The client initiates and controls the session lifecycle by sending requests to the server to register, start and end sessions, and of course analyze frames.
- **Network communication:**
  The client sends frames and session commands to the server and receives analysis results and feedback responses.
- **Feedback presentation:**
  Based on server responses, the client presents feedback to the user using visual pop-up texts and audio cues.

Importantly, the client does not perform posture analysis or decision-making. This design choice avoids duplication of logic and ensures that all evaluation rules are applied consistently on the server side.

3.1.3 Responsibilities of the Backend Server

The backend server is the core analytical component of the system. It is responsible for interpreting incoming data and deciding when and how feedback should be generated. Its main responsibilities include:

- **Session management:**
  The server maintains the state of each active session, including initialization, calibration, active analysis and termination.
- **Frame analysis pipeline:**
  For each received frame, the server executes a structured analysis pipeline that includes pose estimation, biomechanical evaluation and error detection.
- **History and temporal analysis:**
  The server stores recent frame data and analysis results to evaluate movement over time, rather than relying on single-frame decisions.
- **Feedback decision logic:**
  Based on the current session state and aggregated history, the server determines whether feedback should be played and selects the appropriate feedback type.
- **Summary generation:**
  At the end of a session, the server aggregates collected data to produce a session summary, including repetition counts and detected issues.

By centralizing these responsibilities on the server, the system ensures consistent behavior across sessions and devices. It allows future improvements to be implemented without changes to the mobile application.

3.2 Android Client Architecture

The Android client is responsible for all interaction with the user and for managing the device camera. It was designed to be simple, stable and easy to maintain. During development, we made a clear effort to separate visual logic, session control and network communication, to avoid tightly coupled code and reduce complexity. The client does not perform any posture analysis. Its main role is to capture video frames, send them to the server, receive analysis results, and present feedback to the user in a clear way.

3.2.1 Application Structure and Navigation Flow

The application is built using a state-based navigation flow. Each screen represents a clear step in the user journey, and transitions between screens depend on user actions or session state.

The main application flow includes the following stages:

1. **Home screen** – the entry point of the application.
2. **Exercise selection and setup** – the user selects the exercise and prepares for the session.
3. **Camera session** – the active training stage, where frames are captured, and feedback is delivered.
4. **Session summary** – the final screen that presents the results of the training session.

Navigation between these stages is handled in a controlled manner, ensuring that the user cannot accidentally skip required steps such as session initialization or calibration. This structure also simplifies error handling and session cleanup when the application is paused or closed.

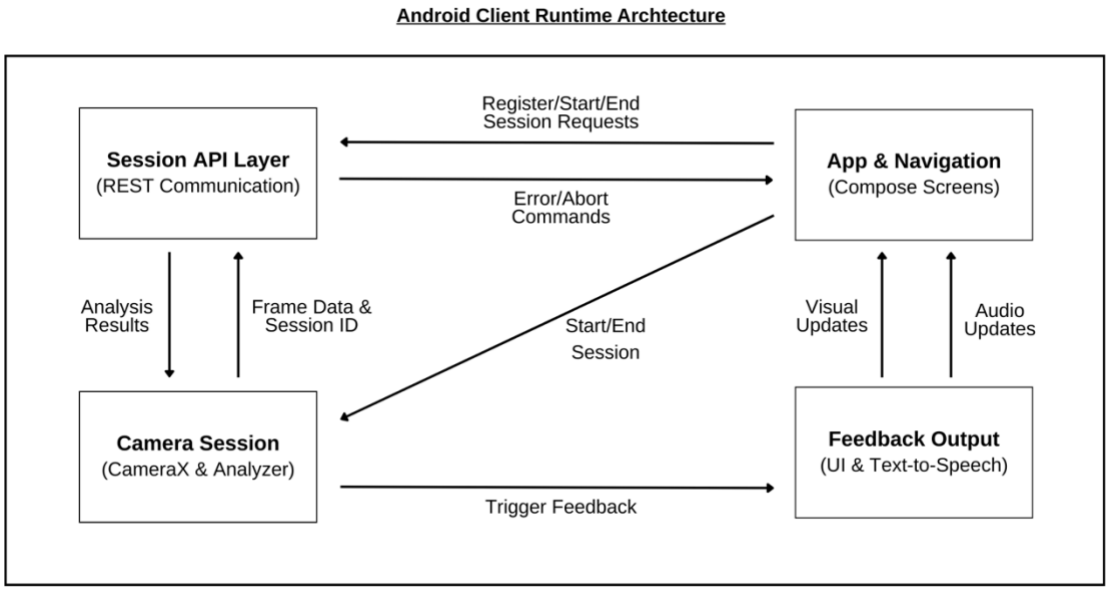

Figure 2: *Main components of the Android client and their interactions*

3.2.2 Camera and Frame Receiving Pipeline

During an active training session, the client uses the device camera to capture video frames in real time. The camera is managed through a dedicated camera session component, which is isolated from the rest of the application logic.

Each captured frame goes through the following steps on the client side:

- The frame is received from the camera in image format.
- The image is converted into a compressed format suitable for network transfer.
- A unique frame identifier is assigned.
- The frame is sent asynchronously to the backend server for analysis.

To avoid overloading the network and the server, frames are not sent at the maximum camera frame rate (fps). Instead, the client limits the sending rate to a controlled frequency. This approach reduces unnecessary traffic while still preserving enough temporal information for accurate posture analysis.

Frame transmission is performed in the background, without blocking the user interface. This ensures that the application remains responsive even when network delays occur.

3.2.3 Networking and Client–Server Communication

Communication between the client and the server is performed using HTTP requests over a REST-based interface. Each request includes a session identifier, allowing the server to associate incoming frames with the correct session.

The client sends different types of requests during a session, including:

- Session registration and start requests

- Frame analysis requests
- End session requests
- Summary retrieval requests after session completion

All network operations are handled asynchronously. When a response is received from the server, the client updates its internal session state and triggers the appropriate user interface or feedback action.

Server responses may represent different situations, such as calibration progress, active feedback, session completion or even session abort in some conditions. The client interprets each response type and reacts accordingly, without needing to understand the internal analysis logic performed on the server.

3.2.4 Feedback Presentation (Visual and Audio)

Feedback is a central part of the BodyTrack system, since it connects the analysis performed on the server with the user's actual training experience. The client is responsible for presenting feedback in a way that is clear, helpful and not distracting.

Visual feedback is displayed directly on the camera session screen while the exercise is performed by the user. In addition to visual cues, the system uses audio feedback through text-to-speech. Audio feedback allows the user to receive guidance without needing to look at the screen, which is important during strength exercises. Feedback messages are short and focused, describing the detected issue in simple terms.

To prevent feedback from becoming overwhelming, the client applies basic control rules when presenting feedback. Feedback messages are not repeated continuously, and ongoing audio feedback is managed so that messages do not overlap. This ensures that feedback remains useful and does not interrupt the user unnecessarily.

3.2.5 Session Lifecycle Management on the Client

The client manages the session lifecycle in coordination with the backend server. A session represents a complete training activity, from initialization to completion, and includes all frames, feedback events and results.

At the beginning of a session, the client sends a registration request to the server and waits for confirmation before starting frame transmission. Once the session is started, the client enters an active capture mode, where frames are periodically sent for analysis.

When the user ends the training session, the client sends a termination request to the server and stops frame capture. After termination, the client requests a session summary from the server and presents the results to the user in a dedicated summary screen.

By managing the session lifecycle explicitly, the client ensures that resources are released correctly and that partial or invalid sessions are avoided. This approach also simplifies recovery from interruptions and supports clean transitions between sessions.

3.3 Backend Server Architecture

The backend server is the core analytical component of the BodyTrack system. It is responsible for managing sessions, analyzing incoming frames and deciding when feedback should be generated. The

server was designed to handle multiple sessions in a stable and structured way, while keeping the analysis logic modular and easy to extend.

Unlike the mobile client, which focuses on user interaction, the server focuses entirely on correctness, consistency and performance of the analysis process.

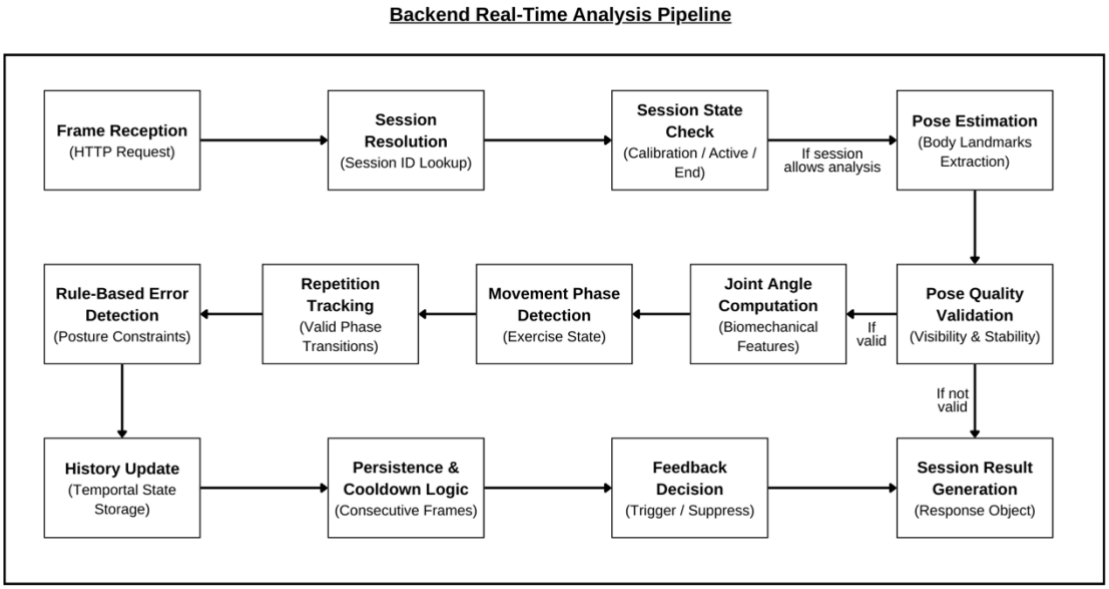

Figure 3: *Real-time analysis pipeline executed on the backend server*

3.3.1 Server Entry Point and Initialization

When the server starts, it performs a controlled initialization process before accepting client requests. During this stage, configuration files are loaded, logging is initialized, and the communication interface is prepared.

The server runs as a long-lived process and exposes a REST-based API that listens for incoming requests from mobile clients. Initialization is performed only once, ensuring that configuration parameters and system resources are ready before any session is created.

This clear startup process simplifies deployment and allows the server to be restarted or reconfigured without affecting the client-side application.

3.3.2 REST API Design and Endpoints

Communication between the client and server is based on HTTP requests using a REST-style interface. Each endpoint serves a clear purpose related to session control or frame analysis.

The main types of endpoints include:

- **Session management endpoints**, used to register, start and end sessions.
- **Frame analysis endpoint**, used to send captured frames for posture evaluation.
- **Summary and status endpoints**, used to retrieve session results and system information.

Each request includes a session identifier, allowing the server to associate incoming data with the correct session. Responses returned by the server are structured and indicate the current system state, such as calibration progress, active feedback, or session completion.

This design keeps the communication protocol simple and predictable, which is important for real-time systems.

3.3.3 Session Management and State Handling

The server manages each session as a separate logical unit. A session stores all information related to a single training activity, including its current state, recent analysis results and accumulated history.

Sessions progress through several stages, such as initialization, calibration, active analysis and termination. The server ensures that requests are handled only when they are valid for the current session state. For example, frame analysis requests are accepted only after a session has been properly started.

To maintain stability, the server also monitors session activity and performs cleanup of inactive or completed sessions. This prevents resource leaks and ensures that the system can operate continuously over time.

Session state handling is centralized on the server, which guarantees consistent behavior even if the client is temporarily interrupted or experiences network delays.

3.3.4 Configuration, Logging, and Monitoring

The server uses a centralized configuration mechanism to control system behavior. Configuration parameters define thresholds, time limits and analysis settings, allowing the system to be adjusted without changing the core implementation.

Logging is used throughout the server to record important events, such as session creation, state transitions and analysis results. These logs support debugging, performance analysis and system monitoring.

In addition to logging, the server records internal analysis traces during frame processing. This information is useful during development and testing, as it allows developers to understand how decisions were made at each stage of the pipeline, for each frame.

3.4 Real-Time Analysis Pipeline (Core of the System)

The real-time analysis pipeline is the heart of the BodyTrack system. It defines how incoming video frames are processed, how movement is evaluated and how feedback decisions are made. The pipeline was designed to operate continuously during a session, while remaining stable and resistant to noisy input.

Each frame sent from the mobile client is analyzed independently, but always within the context of the current session state and recent history. This allows the system to react to the user's movement in real time, while avoiding incorrect decisions based on single unstable frames.

3.4.1 Frame Reception and Validation

When the server receives a frame from the client, the first step is basic validation. The server checks that:

- The session identifier is valid
- The session is in a state that allows analysis

- The frame data can be decoded correctly

Frames that fail validation are ignored, and the session continues without interruption. This prevents corrupted data or unexpected requests from making an effect over future decisions.

After validation, the frame is converted into an internal image representation that can be used by the analysis modules.

3.4.2 Pose Estimation and Landmark Extraction

Once the frame is ready, the server applies pose estimation to extract body landmarks. These landmarks represent key points on the human body, such as shoulders, hips, knees and elbows.

The pose estimation stage provides the raw spatial information needed for further analysis. At this stage, no decisions are made regarding posture correctness. The output consists only of landmark positions and confidence values.

If pose estimation fails or the detected landmarks are unreliable or unstable, the frame is marked as invalid and handled accordingly in later stages.

3.4.3 Pose Quality Evaluation

Before analyzing movement or posture, the system evaluates the quality of the detected pose. This step checks whether the user is sufficiently visible and properly positioned in the camera frame.

Pose quality evaluation includes checks such as:

- Whether enough landmarks are visible for the performed exercise
- Whether the body is within the camera frame
- Whether the pose detection confidence is higher from a predefined threshold

If the pose quality is not acceptable, the system does not proceed with deeper analysis. Instead, it updates internal counters that track visibility stability and may trigger calibration-related feedback.

This step is especially important during the early stages of a session, where the system verifies that the user is visible, and the camera is stable.

3.4.4 Joint Angle Computation

For frames that pass the pose quality checks, the system computes joint angles based on the extracted landmarks. Joint angles are calculated using the analyzed landmarks and connections between them.

These angles represent biomechanical properties of the movement, such as knee flexion or elbow extension. Joint angle values are used as the main input for detecting exercise phases and posture errors.

The use of joint angles provides a clear and explainable representation of movement, which supports clear and meaningful decision-making and easier debugging.

3.4.5 Movement Phase Detection

After joint angles are computed, the system determines the current movement phase of the exercise. Each supported exercise is defined by a sequence of phases, such as starting position, lowering phase and lifting phase.

Phase detection is based on joint angle trends and threshold values. By tracking how angles change over time, the system can detect transitions between phases and identify repetitions.

Phase detection is state based, meaning that transitions are allowed only in valid orders. This prevents incorrect repetition counting due to sudden or unstable angle changes.

3.4.6 Error Detection and Classification

Once the movement phase is known, the system evaluates posture correctness for the current frame. Error detection is performed using predefined biomechanical rules that compare joint angles and body alignment against acceptable ranges.

Each detected issue is classified into a specific error type. Examples include incorrect joint alignment or insufficient range of motion.

Importantly, errors are not immediately reported based on a single frame. Instead, the system tracks error occurrences over multiple frames to confirm that an issue is persistent before generating feedback.

3.4.7 History Management and Temporal Smoothing

To improve stability, the system maintains a history of recent frames and analysis results. This history is used to:

- Count consecutive valid or invalid frames
- Track repeated posture errors
- Smooth sudden changes in detected angles
- Associate errors with specific repetitions

By relying on history rather than single frames, the system reduces false positives and avoids unstable feedback behavior.

3.4.8 Feedback Decision Logic

Based on the current analysis results and accumulated history, the system decides whether feedback should be generated. The feedback decision logic considers:

- The current session state
- The severity and duration of detected errors
- Recent feedback activity, to avoid repetition

If feedback is required, the system selects an appropriate feedback type and prepares a structured response for the client. If no feedback is needed, the system continues processing frames silently.

This decision logic ensures that feedback is meaningful, timely, and not excessive.

3.4.9 Response Generation and Transmission

In the final stage of the pipeline, the server creates a response message and sends it back to the client. The response may represent - Calibration progress, Active corrective feedback, Session completion or summary availability.

Responses are kept compact and clearly structured, allowing the client to react quickly and present feedback without delay.

After the response is sent, the server updates the session history and waits for the next frame. This cycle repeats continuously until the session ends.

3.5 System Operation Flow

This section describes how the BodyTrack system operates from the moment a user starts a session until the session is completed. The flow is presented from a functional point of view, showing how the client and server work together during each stage of the training process.

The operation flow is divided into three main phases: session initialization and calibration, active training, and session termination.

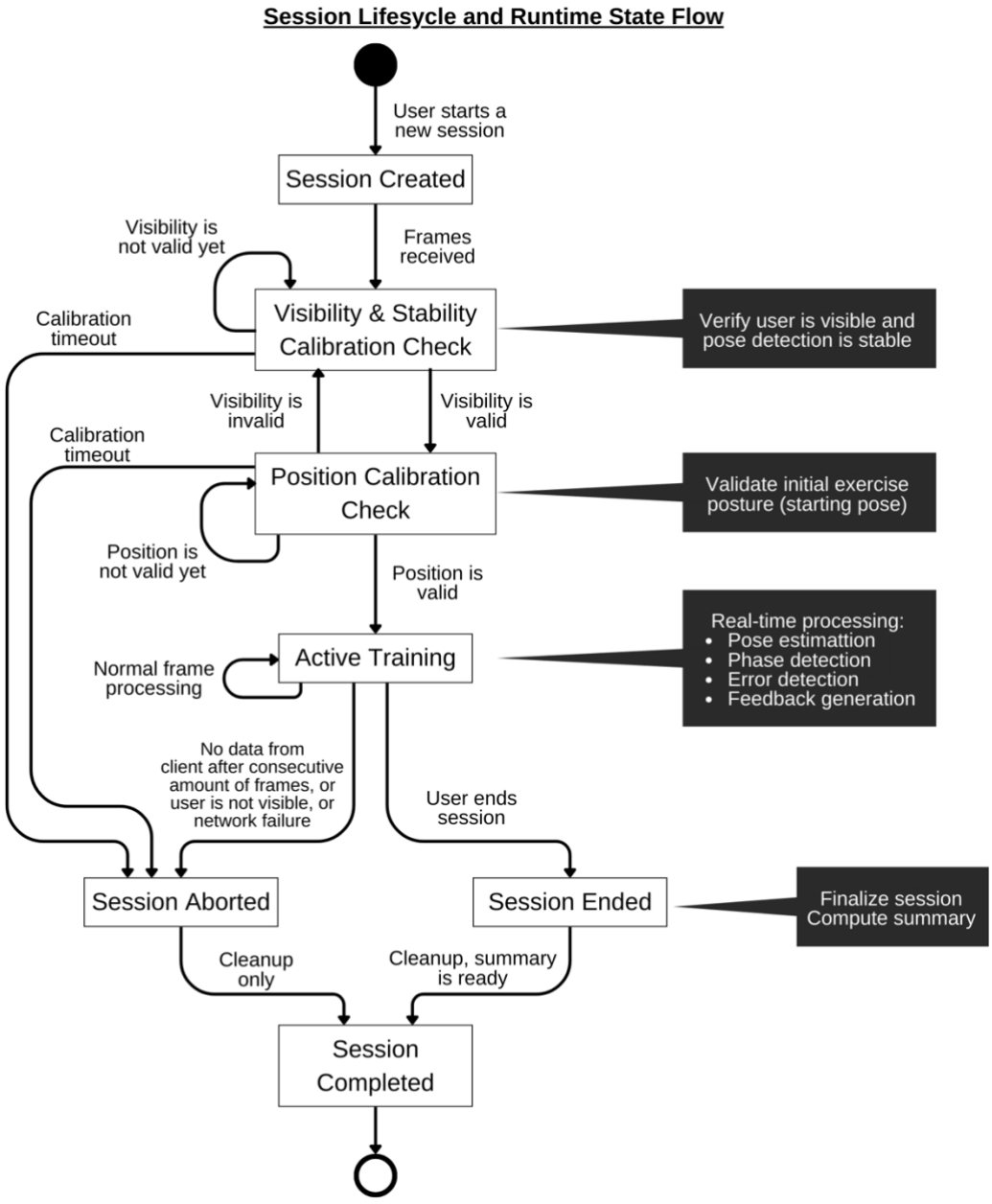

Figure 4: *Session lifecycle and main operation flow.*

3.5.1 Session Initialization and Calibration Phases

When the user starts a new training session, the mobile client first registers the session with the backend server. This step creates a new session context on the server, which will store all information related to the upcoming training activity.

After registration, the client starts sending frames to the server. At this stage, the system enters the initial calibration phase. The purpose of this phase is to verify that the user is visible, correctly positioned and ready for analysis. During calibration, the system does not evaluate exercise correctness. Instead, it focuses on basic conditions such as camera visibility and body stability.

The calibration process is divided into two stages:

- **Visibility validation**, where the system checks that the user is fully visible and that pose detection is stable.
- **Initial position validation**, where the system verifies that the user is standing in the correct starting posture for the selected exercise.

After sending enough (predefined number of) consecutive valid frames, the system automatically moves to the active training phase.

3.5.2 Active Training Phase

During the active training phase, the system continuously analyzes incoming frames and evaluates the user's movement in real time. Each frame is processed through the full analysis pipeline, including pose estimation, joint angle computation, phase detection and error evaluation.

The system tracks exercise repetitions by monitoring movement phase transitions. At the same time, it evaluates posture correctness and checks for persistent errors. When an error is confirmed, the system generates feedback and sends it to the client.

Feedback during this phase is delivered in short and clear messages, allowing the user to adjust posture without interrupting the exercise. If no issues are detected, the system continues silently (produces a 'silent' feedback).

Throughout the active phase, the server maintains a detailed session history. This history is later used to generate a summary of the training session.

3.5.3 Session Completion and Summary Generation

When the user ends the training session, the client sends a termination request to the server and stops sending frames. The server finalizes the session state and performs summary calculations based on the accumulated session data.

The session summary includes information such as the number of completed repetitions, detected posture issues and general recommendations. This summary is stored on the server and made available to the client upon request.

After receiving the summary, the client presents it to the user on a dedicated summary screen. The session is then considered complete, and system resources associated with the session are released.

4 Development Process

The development process followed an iterative and incremental approach, allowing the system to evolve based on testing results, observed limitations and practical constraints.

4.1 Development Strategy and Methodology

From the beginning of the implementation phase, we adopted an incremental development strategy. Core infrastructure components were implemented first, followed by analysis logic, and finally user-facing functionality. This approach reduced risk and made it easier to identify and fix issues early.

Throughout development, we preferred simple and reliable solutions over complex ones. When multiple approaches were possible, decisions were made based on real-time performance, ease of debugging and long-term maintainability.

4.2 Evolution from Conceptual Design to Implementation

During Phase A, several design ideas were explored at a conceptual level. However, once implementation began, it became clear that some assumptions needed to be adjusted.

For example, while machine learning–based posture classification was initially considered, early experiments showed that such approaches would increase system complexity without providing clear benefits for the supported exercises. As a result, we decided to use deterministic, rule-based biomechanical evaluation, which proved to be more stable and easier to control in real-time conditions.

Similarly, the importance of temporal stability became clearer during implementation. Single-frame decisions led to unstable behavior and frequent false alerts. This observation motivated the introduction of history-based analysis and consecutive-frame validation, which significantly improved system reliability.

4.3 Incremental Development and Integration

The implementation of BodyTrack progressed through several incremental stages:

Server infrastructure setup: The backend server was implemented first, including communication endpoints, session management and basic request handling.

Analysis pipeline skeleton: A structured pipeline was created to process frames step by step, even before all analysis logic was fully implemented.

Pose and biomechanical analysis: Pose estimation and joint angle computation were added, followed by movement phase detection and error evaluation.

History and feedback logic: History tracking and feedback decision mechanisms were introduced to improve stability and control feedback behavior.

Android client development: The mobile application was implemented with a focus on camera handling, session control and feedback presentation.

End-to-end integration: The client and server were integrated and tested together in live sessions, allowing validation of real-time behavior.

4.4 Validation During Development

Validation was performed continuously throughout development. Testing included both controlled experiments using recorded videos and live testing using a mobile device during actual exercise execution.

In addition to technical testing, informal feedback was collected from gym users who interacted with the system. Their observations helped identify usability issues, such as unclear feedback messages or calibration delays, and guided further improvements.

5 Tools, Frameworks and Technologies

This section describes the main tools, frameworks, and technologies used to implement the BodyTrack system. The choices were guided by practical considerations such as stability, ease of development, and suitability for real-time processing.

5.1 Mobile Client Technologies

The mobile client was developed as an Android application using **Kotlin**. Kotlin was chosen because it is the recommended language for modern Android development and provides strong support for safety, readability and asynchronous programming.

For the user interface, the application uses **Jetpack Compose**. This framework allows building UI components in a clear and structured way, based on application state. Compose simplified navigation between screens and made it easier to manage dynamic UI changes during the training session.

Camera access is handled using the **CameraX** library. CameraX provides a stable and consistent interface for camera usage across different Android devices. It also simplifies lifecycle management, which is important for avoiding camera-related issues when the application is paused or resumed.

Audio feedback is delivered using the built-in Android **Text-to-Speech** engine. This allows the system to provide spoken feedback without requiring the user to constantly look at the screen.

5.2 Backend Server Technologies

The backend server was implemented using **Python**. Python was selected due to its strong ecosystem for computer vision and rapid development. The server exposes a REST-based API using the **Flask** framework. Flask provides a lightweight and flexible structure for handling HTTP requests and managing server routes. Cross-origin communication support was added to allow interaction with mobile clients.

For image processing and numerical operations, the system uses common scientific libraries. These libraries support decoding image data, processing pose information, and performing geometric calculations required for biomechanical analysis.

The server runs as a long-lived process (on a cloud service) and supports multiple sessions through internal session management and synchronization mechanisms.

5.3 External Libraries and APIs

Pose estimation is performed using **MediaPipe**, which is an external pose estimation framework capable of extracting body landmarks in real time from RGB images. This framework serves as the foundation for all further biomechanical analysis.

Additional utility libraries are used for configuration loading, logging, and data handling.

5.4 User Feedback and Informal Evaluation Process

Although the project did not involve a formal external client, feedback was collected informally from gym users during development. These users interacted with the system during real training sessions and provided observations regarding feedback clarity, responsiveness and usability.

This informal evaluation helped guide improvements in feedback phrasing, calibration flow and system responsiveness.

6 Engineering Challenges and Solutions

Developing a real-time posture evaluation system introduced several engineering challenges. These challenges were not only technical, but also related to system stability and usability.

6.1 Real-Time Performance and Latency Constraints

The system is required to operate in real time, meaning that feedback must be delivered shortly after the movement occurs. Sending every camera frame to the server caused unnecessary network load and increased latency, which affected responsiveness.

To address this issue, the client limits the rate at which frames are sent to the server. Frames are transmitted at a controlled frequency that balances responsiveness and performance. All network communication is handled asynchronously, ensuring that the user interface remains responsive even when network delays happen.

6.2 Stability of Pose Detection and Noise Reduction

Pose estimation output can be unstable, especially in non-ideal conditions such as varying lighting, partial visibility or fast movement. Relying on single-frame analysis often led to false detections and inconsistent feedback.

To solve this, the system uses history-based analysis instead of making decisions based on individual frames. Consecutive-frame validation is applied during calibration, and error detection requires persistence over multiple frames before feedback is generated. These reduces the effect of momentary noise and ensures that feedback reflects consistent movement patterns rather than isolated detection errors.

6.3 Session Consistency and State Synchronization

Since the system is distributed between a mobile client and a backend server, maintaining consistent session state was essential. Interruptions such as app backgrounding or temporary network issues could lead to invalid or incomplete sessions.

To solve this, session lifecycle management was centralized on the server. The client explicitly notifies the server of session start and termination events. The server validates each request based on the current session state and ignores invalid transitions. This ensures consistent behavior and prevents corrupted session data. It also simplifies recovery from interruptions and allows the server to clean up inactive sessions safely.

6.4 Feedback Control and Over-Notification Prevention

Providing too much feedback can confuse or distract users, especially during continuous movement. Early versions of the system generated repeated feedback for the same issue, which reduced its effectiveness.

To solve this, feedback generation is controlled using internal counters and cooldown logic. Once feedback is delivered, the system waits before issuing additional messages. Errors must persist over time to trigger feedback again. This approach ensures that feedback remains meaningful and actionable. Users receive guidance when it is needed, without being overwhelmed by repeated notifications.

6.5 Repetition Detection and Movement Phase Handling

Correctly identifying exercise repetitions is more complex than counting movement peaks. Noise or incorrect phase transitions can lead to missed or duplicated repetitions.

Our repetition detection is based on explicit movement phase transitions rather than raw joint angle values. Only valid phase sequences are counted as repetitions, and invalid transitions reset the detection process. This structured approach improves repetition accuracy and allows the system to associate detected errors with specific repetitions. It also supports reliable session summaries at the end of training.

7 Results, Conclusions and Lessons Learned

7.1 Achieved Project Goals

The final system supports complete training sessions, including session initialization, calibration, real-time analysis, feedback delivery and session summary generation. The client–server architecture operates reliably in real time, and feedback is delivered with minimal delay during exercise execution.

The system can:

- Detect posture issues during supported exercises
- Provide clear audio and visual feedback during training
- Track repetitions and associate errors with movement phases
- Generate a structured summary at the end of each session

These capabilities demonstrate that the system functions as intended and provides practical value for users training independently.

7.2 Evaluation of the Implemented System

From an engineering perspective, the system meets the requirements defined at the beginning of the project. The use of deterministic biomechanical rules, combined with history-based analysis, resulted in stable and predictable behavior.

Real-time testing showed that the system responds consistently under normal training conditions. Calibration phases reduced false detections, and feedback control mechanisms prevented excessive

notifications. While no formal accuracy study was conducted, subjective evaluation through live testing and user feedback indicated that the system behaves reasonably and provides useful guidance during exercise execution.

7.3 Design Trade-Offs and Engineering Decisions

Several trade-offs were made during development. The decision to avoid complex machine learning models in favor of rule-based evaluation improved system transparency and stability, at the cost of reduced flexibility for very complex movements. Similarly, limiting the frame transmission rate reduced latency and improved responsiveness, but required careful tuning to maintain sufficient temporal resolution.

Overall, these trade-offs were considered acceptable and appropriate for the goals of the project and the intended usage context.

7.4 Lessons Learned

7.4.1 Technical Lessons

One important lesson was the importance of temporal analysis in real-time systems. Decisions based on single frames were unreliable, while history-based evaluation significantly improved stability and usability. Another lesson was the value of clear separation of responsibilities. Dividing the system into a lightweight client and an analytical server simplified development and made debugging easy.

7.4.2 Architectural and Design Insights

Designing the system as a state-driven pipeline helped manage complexity and maintain consistency. Explicit session states and phase transitions reduced unexpected behavior and made the system easier to reason about. Early attention to configuration and logging also proved valuable. These mechanisms allowed rapid adjustment of thresholds and supported efficient debugging without major code changes.

7.4.3 Reflections and Possible Improvements

Looking back, additional time could be invested in expanding exercise coverage and conducting a formal usability or accuracy evaluation. Future improvements could also include personalization of thresholds based on individual users and improved visual feedback.

Despite these possible extensions, the current implementation fulfills its objectives and provides a solid foundation for future development.

8 User Guide

8.1 Entry Screen

The entry screen serves as the entry point to the BodyTrack application. It displays the application logo along with a short motivational message. Pressing the **Start** button navigates the user to the Home screen, where the exercise selection and training session setup begin.

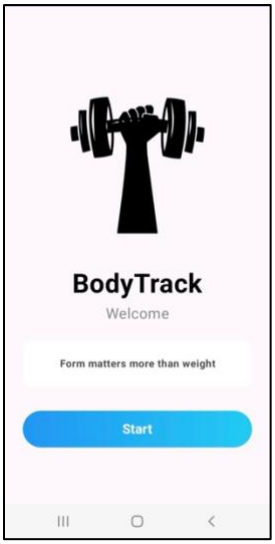

Figure 5: *BodyTrack Entry Screen*

8.2 Home Screen

The Home Screen is the main screen of the BodyTrack application. It shows a short message and the list of exercises that the user can choose from. To start a workout, the user selects an exercise from this screen.

Each exercise is displayed as a card with a video and a title. When the user taps an exercise, such as Squats, the application moves to the session setup and camera-based training flow. The screen also includes an **(i)** button. Pressing this button opens the About Us screen

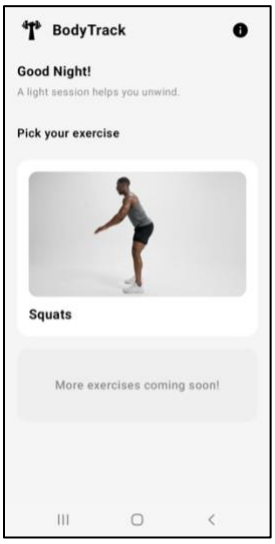

Figure 6: *BodyTrack Home Screen*

8.3 About Screen

The About Us pop up presents basic information about the BodyTrack application, including the creators of the project and its main goal and vision. It also includes links to the developers' LinkedIn profiles and a **Back** button that returns the user to the Home Screen.

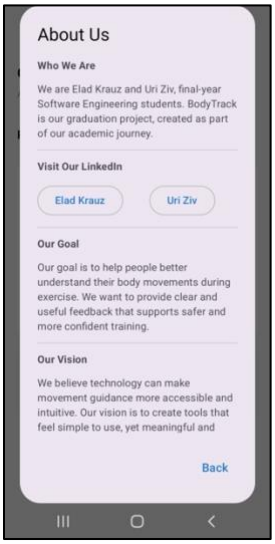

Figure 7: *About Us Pop Up*

8.4 Instruction Screens

After selecting an exercise, the user is guided through a short series of instruction screens. These screens provide key guidelines before the session starts, such as body visibility, camera position, starting posture, and general movement instructions for the exercise.

The user navigates through the instructions using the **Next** and **Back** buttons. Once all instructions are reviewed, the user presses Start Session to begin the live training and feedback session.

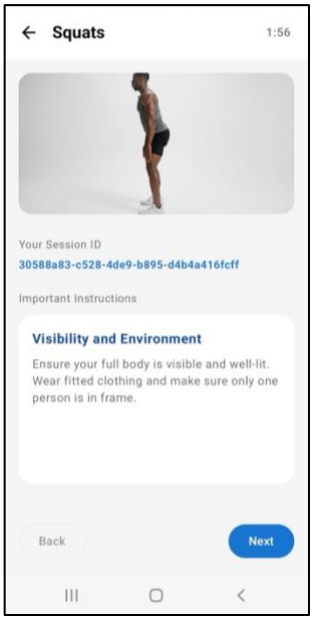

Figure 8: *Visiblity and Environment*

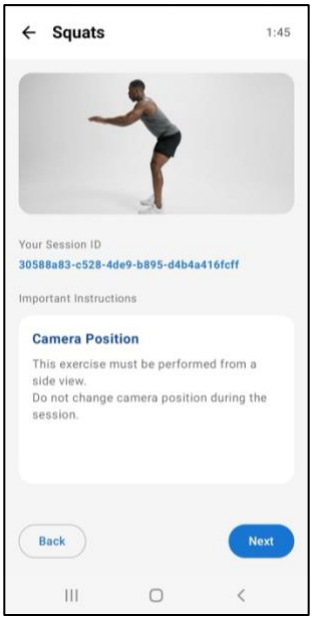

Figure 9: *Camera Position*

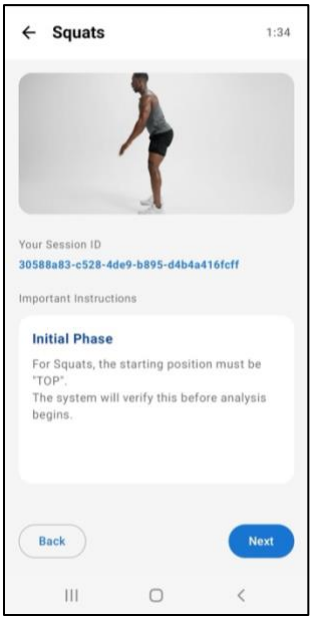

Figure 10: *Initial Phase*

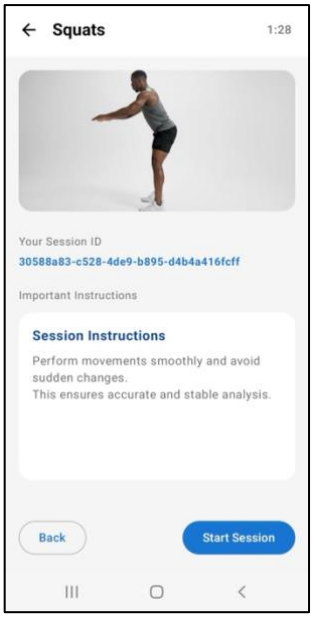

Figure 11: *Session Instructions*

After 120 seconds of no interactions (can be adjusted), the screen closes and returns to Home.

8.5 Session Settings

Before starting a training session, the user needs to adjust a few basic session settings, like turning on/off the voice feedback, turning on/off the text feedback, choosing the camera input (front or back) and setting the session duration.

After setting the preferences, the user can press **Start** to begin the session or **Cancel** to return to the previous screen.

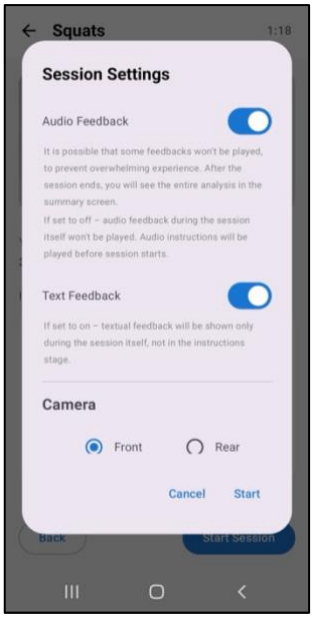

Figure 12: *Session Settings (1)*

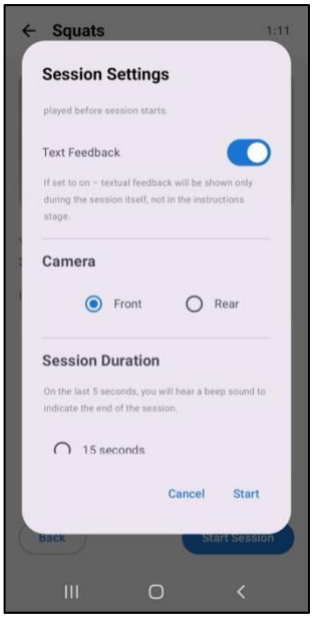

Figure 13: *Session Settings (2)*

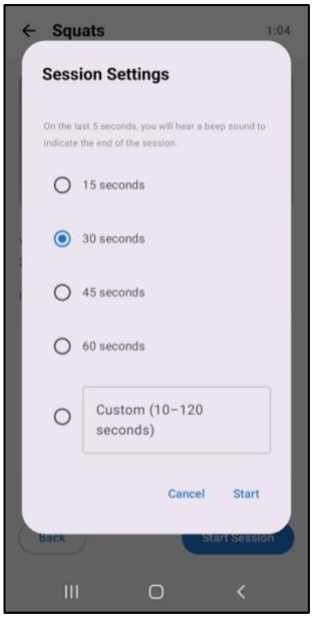

Figure 14: *Session Settings (3)*

8.6 Exit setup

When the user tries to leave the session setup screen, a confirmation message appears. This message explains that leaving will cancel the current setup.

The user can choose **Stay** to continue setting up the session, or **Yes, exit** to cancel and go back to the previous screen.

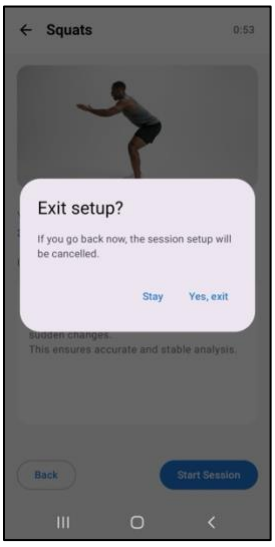

Figure 15: *Exit Setup Pop Up*

8.7 Training Session Screen

During the training session, the user sees a live camera view while performing the exercise. The system analyzes the body position in real time and gives feedback both on the screen and through voice guidance, helping the user correct things like squat depth, hip position and upper body posture. The session can be stopped at any time by pressing the **End** button at the bottom of the screen, which immediately finishes the session.

Figure 16: *Calibration Mode*

Figure 17: *Active Session*

Figure 18: *Active Session*

Figure 19: *Active Session*

8.8 Session Summary Screen

After the training session ends, the system processes the data and then shows the Session Summary screen. This screen gives a clear overview of the workout, including total session duration, number of repetitions and average duration, overall grade that reflects the user's performance, repetition breakdown, aggregated errors and recommendations.

In the repetition breakdown, the user can see each repetition with its duration, whether it was performed correctly, and the posture issues that were detected. The aggregated errors highlight recurring mistakes, while the recommendations offer simple guidance for improvement. The user can return to the Home screen by pressing the **Back to Home** button.

Figure 20: *Loading Summary*

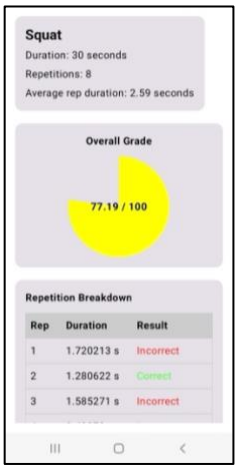

Figure 21: *Session Summary*

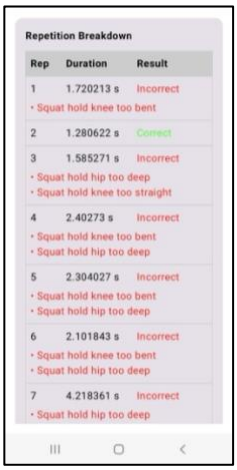

Figure 22: *Session Summary*

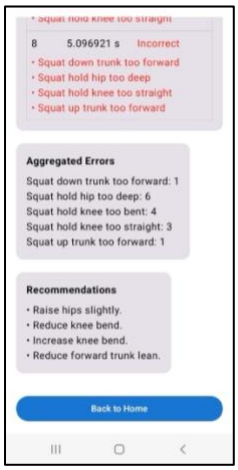

Figure 23: *Session Summary*

9 Maintenance Guide

This maintenance guide explains how to run, maintain and extend the BodyTrack system after the project is completed. It is intended for future developers or maintainers who need to:

- Run the server and the Android application
- Update configurations and thresholds
- Monitor the system and debug issues
- Improve performance or stability
- Add new exercises to the system

9.1 System Overview

BodyTrack is a client-server system composed of two main parts:

Android Client (Kotlin):
The Android application is responsible for:

- Displaying the user interface
- Capturing camera frames
- Sending frames to the server
- Displaying text feedback
- Playing voice feedback
- Showing session summaries

Python Server (Flask):
The server is responsible for:

- Receiving video frames
- Analyzing body posture
- Detecting exercise phases
- Detecting posture errors
- Managing sessions
- Generating real-time feedback
- Producing session summaries

The client and server communicate over HTTP. The server is hosted online, and the Android device connects to it through a Wi-Fi/Cellular network.

9.2 Server Environment and Requirements

9.2.1 Hardware and Network

- Cloud-based virtual machine (Using a droplet like DigitalOcean)
- Public IP address
- Open port (default: 8080)
- Internet connection

The server is deployed on a paid cloud droplet. A droplet is a virtual Linux machine that runs continuously and allows external devices to connect using its public IP address.

9.2.2 Software Requirements

- Python 3.10
- Virtual environment (venv)

Required Python libraries (also shown in the requirements.txt file on the server):

- flask
- flask-cors
- gunicorn
- mediapipe
- opencv-python
- numpy
- psutil

9.3 Server Installation and Setup

9.3.1 Creating the Virtual Environment

From the Server directory, run:

```
python3 -m venv venv
source venv/bin/activate
pip install --upgrade pip
pip install -r requirements.txt
```

This virtual environment isolates all server dependencies and prevents conflicts with other Python projects.

9.3.2 Server Structure

Root Directory	Inner Directories/Files (Directories are in **bold**, files are regular)	
Server.py – *main file*		
Communication	Communication.py - *Shares communication helpers and common request/response definitions* FlaskServer.py - *Defines all HTTP endpoints and request handling* HttpCodes.py - *Represents HTTP known codes*	
Data	**Debug**	FrameEvent.py - *Represents a single pipeline event* FrameTrace.py - *Stores the full trace of frame processing*
	Error	DetectedErrorCode.py - *Enum of all posture error codes* ErrorMappings.py - *Maps detected errors to feedback logic*
	History	HistoryData.py - *Stores per-repetition and per-session history* HistoryDictKey.py - *Defines keys used to organize history data*
	Joints	JointAngle.py - *Defines which joints are analyzed and calculated*
	Phase	PhaseDictKey.py - *Keys related to phase tracking* PhaseType.py - *Enum of exercise phases*
	Pose	PoseLandmarks.py - *Landmark definitions (indices by MediaPipe)* PoseQuality.py - *Pose quality levels* PositionSide.py - *User orientation relative to the camera*
	Response	CalibrationResponse.py - *Calibration and visibility response wrapper* FeedbackResponse.py - *Real-time feedback response wrapper* ManagementResponse.py - *Management and telemetry responses wrapper* SummaryResponse.py - *Session summary response wrapper*
	Session	AnalyzingState.py - *Tracks the current analysis stage* ErrorRecommendations.py - *Maps errors to improvement suggestions* ExerciseType.py - *Enum of supported exercises* FrameData.py - *Represents a single processed frame* SearchType.py – *An enum of search types in the sessions database* SessionData.py - *Stores all data related to an active session* SessionStatus.py - *Defines session states*

<table>
<tr><td rowspan="5">Files</td><td rowspan="3">Config</td><td>ErrorThresholds.JSON - Posture error thresholds</td></tr>
<tr><td>PhaseThresholds.JSON - Movement phase thresholds</td></tr>
<tr><td>ServerConfiguration.JSON - General server settings</td></tr>
<tr><td rowspan="2">Logs</td><td>Archive/ - An Archive for all old logger files</td></tr>
<tr><td>ServerLogger.log – The logging file for the entire server process</td></tr>
<tr><td>Telemetry</td><td>ServerTelemetry.html – A telemetry service for showing current server data</td></tr>
<tr><td>Management</td><td colspan="2">ServerManager.py - Initializes server and all its components, and runs in an infinite loop
SessionManager.py - The central control component of the sessions on the server
SessionSummaryManager.py - Generates and stores summarized results for completed training sessions
TestManager.py - Supports testing, validation, and debugging of server functionality</td></tr>
<tr><td>Pipeline</td><td colspan="2">ErrorDetector.py - Detects posture errors based on joint angles and defined thresholds
FeedbackFormatter.py - Converts internal posture errors into user-facing feedback messages
HistoryManager.py - Stores frame-level data, repetition data, and detected errors across time
JointAnalyzer.py - Calculates joint angles based on detected landmarks
PhaseDetector.py - Detects the current movement phase of the exercise
PipelineProcessor.py - Coordinates the execution of all pipeline stages for a single frame
PoseAnalyzer.py - Uses MediaPipe to extract body landmarks from the camera frame
PoseQualityManager.py - Checks whether the user is visible and positioned correctly in the frame
PositionSideDetector.py – Uses the detected landmarks to determine the standing position</td></tr>
<tr><td rowspan="4">Utilities</td><td rowspan="2">Config</td><td>ConfigLoader.py - Loads configuration files and exposes system settings</td></tr>
<tr><td>ConfigParameters.py - Defines configurable system parameters name keys</td></tr>
<tr><td rowspan="2">Error</td><td>ErrorCode.py - Defines standardized error codes used across the system</td></tr>
<tr><td>ErrorHandler.py - Standardizes how technical runtime errors are handled</td></tr>
<tr><td colspan="2">Logger.py - Provides centralized logging for the entire server</td></tr>
<tr><td colspan="2">SessionIdGenerator.py – Used to generate identifier to a new session</td></tr>
<tr><td colspan="3">venv – Virtual environment</td></tr>
<tr><td colspan="3">requirements.txt – Requirements file with all external libraries need to be installed in the virtual environment</td></tr>
</table>

9.4 Running and Deploying the Server

9.4.1 Development Mode

Use these commands to activate the virtual environment and run the main Python file.

```
source venv/bin/activate
python Server.py
```

The server listens on host 0.0.0.0 and port 8080.

9.4.2 Production Mode (Gunicorn)

Since Server.py exposes Flask application, for production you can use Gunicorn to listen for HTTP requests.

```
gunicorn Server:app --bind 0.0.0.0:8080
```

9.4.3 Service Management and Operational Control

In the deployed environment, the backend server is executed using Gunicorn and managed as a persistent system service. This allows the server to start automatically on system boot and remain available without manual intervention.

To simplify server operation and maintenance, a custom command-line utility (bt) is provided. This utility acts as a management interface over the system service and exposes common operational commands.

The supported commands are:

- **bt start** – Starts the Gunicorn service and begins accepting requests.
  Runs: `sudo systemctl start bodytrack`

- **bt stop** – Stops the backend service and terminates all active sessions.
  Runs: `sudo systemctl stop bodytrack`
- **bt restart** – Restarts the Gunicorn process without restarting the host system.
  Runs: `sudo systemctl restart bodytrack`
- **bt status** – Displays the current operational status of the backend service.
  Runs: `sudo systemctl status bodytrack`
- **bt logs** – Streams live service logs for monitoring and debugging purposes.
  Runs: `sudo journalctl -u bodytrack -f`

Internally, these commands control the Gunicorn process through the system service manager. When a reverse proxy such as Nginx is used, it operates independently of Gunicorn, allowing backend restarts without interrupting incoming network connections.

9.4.4 Logging and Troubleshooting

The backend server writes runtime logs through the system service manager. These logs include startup messages, runtime warnings, and error reports generated during request handling and session processing.

For maintenance and debugging purposes, logs can be accessed using the provided management command or directly through the system logging interface. Reviewing logs is the primary method for diagnosing unexpected behavior, failed requests or session interruptions.

9.4.5 Configuration and Safe Modifications

The system is designed to allow certain behavioral adjustments through configuration parameters, such as thresholds and timing values, without modifying the core processing pipeline. These parameters can be updated safely if their structure is preserved.

Changes to the core analysis logic, session management or network interfaces should be performed with caution, as they may affect system stability and client compatibility. Any such modifications should be validated using controlled test sessions before deployment.

9.4.6 Network and Security Notes

Currently, the server uses a public IP address, so communication is done over HTTP. Since HTTP is not secure, future development may include transferring to HTTPS protocol. To achieve that:

- Purchase a domain name
- Point the domain to the server IP (as specified in the cloud service in use)
- Replace IP-based calls (on the client side, under the API file) with domain-based calls
- Enable HTTPS for secure communication

This change does not require modifications to the processing pipeline.

9.5 Configuration Management and Telemetry

9.5.1 Configuration Files

Configuration files are stored in the directory **Files/Config**:

- ServerConfiguration.json
- PhaseThresholds.json
- ErrorThresholds.json

These files control:

- System settings and thresholds, which affect behavior
- Angle limits
- Phase detection thresholds
- Error sensitivity
- Exercise-specific parameters

9.5.2 Runtime Configuration Reload

Configurations can be reloaded without restarting the server using the **/refresh/configurations** endpoint. This endpoint reloads all configuration files from disk and applies them immediately.

The runtime reload will be explained in the next sections.

9.5.3 Telemetry Interface

The server includes a Telemetry Web Interface used for monitoring and maintenance.
The telemetry screen displays:

- Supported exercises
- Maximum allowed clients
- Registered sessions
- Active sessions
- Ended sessions
- Total sessions since startup
- Server memory usage (via psutil)

It also includes a sessions table showing stored sessions, and their info: session ID, client IP, exercise type, session status, current analyzing state and last activity timestamp.

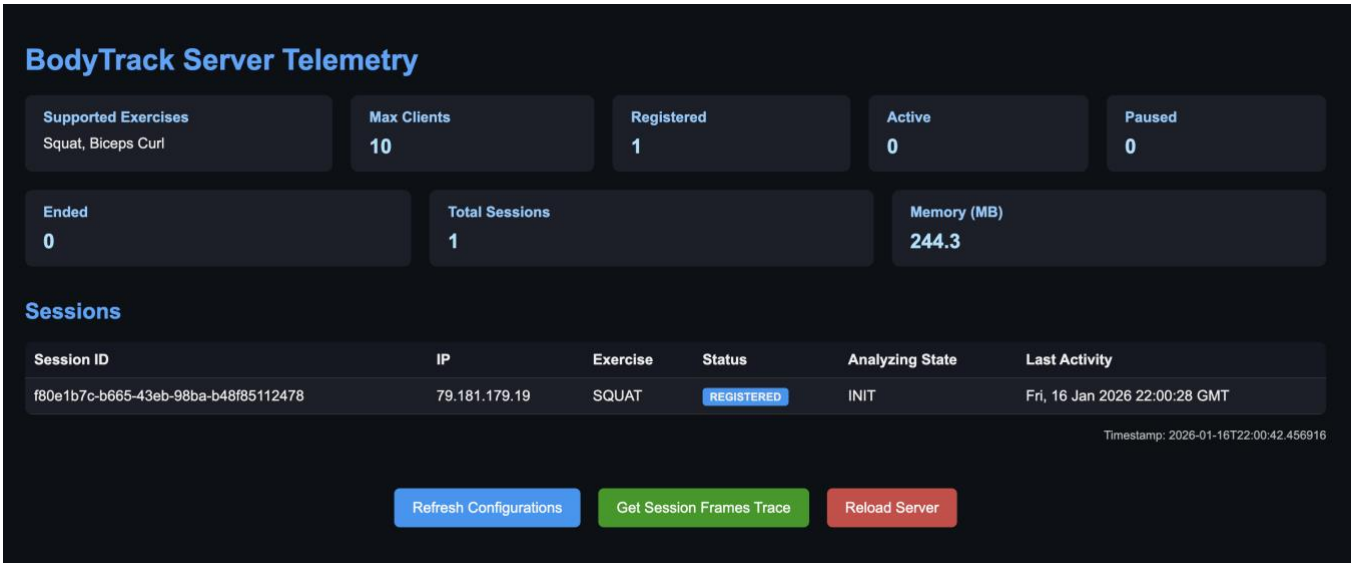

Figure 24: *BodyTrack Telemetry Web Application*

9.5.4 Telemetry Actions

The telemetry interface provides three main buttons:

- **Refresh Configurations:** Triggers the **/refresh/configurations** endpoint and reloads all configuration files at runtime.

- **Get Session Frames Trace:** Retrieves a detailed frame-level trace of analyzed sessions. This is used for debugging pipeline behavior and error detection logic.
- **Reload Server:** Performs a controlled server reload, mainly for development and maintenance.

9.6 Server Processing Pipeline

9.6.1 Frame Reception

Defined in the FlaskServer.py file under the endpoint **POST/analyze**.

1. Client sends a Base64-encoded frame
2. Server decodes it into an OpenCV image
3. Frame is passed to SessionManager.analyze_frame()

9.6.2 Session Handling

The SessionManager class is responsible for:

- Validating session ID
- Tracking session state
- Locking session data
- Routing frames into the correct pipeline stage

Session may be in one of the following states: INIT, READY, ACTIVE, END (for completed sessions)

9.6.3 States Breakdown

	INIT	**READY**	**ACTIVE**
Classes Involved	PoseAnalyzer PoseQualityManager	JointAnalyzer PhaseDetector	PoseAnalyzer PoseQualityManager JointAnalyzer PhaseDetector ErrorDetector HistoryManager FeedbackFormatter
Purpose	To verify that the user is visible, and to validate camera framing and quality	To ensure correct starting posture, and to prevent early session start	To detect repetitions and posture errors, to store repetition history and to generate real-time feedback
Output	Calibration feedback	Calibration feedback	Textual/Audio feedback

9.7 Session Summary Generation

The SessionSummaryManager is responsible for generating a session summary after the session if completed, and upon request using the **/session/summary** endpoint.

Generated summary includes:

- Total session duration
- Number of repetitions
- Average repetition time
- Overall performance grade

- Per-repetition breakdown
- Aggregated errors
- Improvement recommendations

The summary is sent to the client and displayed on the summary screens on the app.

9.8 Adding a New Exercise

9.8.1 Server-Side Changes

File	What to update/add
Data/Session/ExerciseType.py	Add a new exercise type to the enum class
Data/Joints/JointAngle.py	Add a new subclass and its relevant joint angles
Data/Phase/PhaseType.py	Add a new enum for the exercise with its phases
Data/Pose/PositionSide.py	Add the exercise and its supported positioning side/s
Data/Error/DetectedErrorCode.py	Add all the exercise's errors and feedback that might be generated upon user training
Data/Error/ErrorMappings.py	
Data/Error/ErrorRecommendations.py	
Data/Response/FeedbackResponse.py	
Files/Config/ServerConfiguration.json	Add the new exercise to the supported_exercises list
Files/Config/PhaseThresholds.json	Add a new object for the exercise with phase thresholds
Files/Config/ErrorThresholds.json	Add a new object for the exercise with error thresholds

9.8.2 Client-Side Changes

File	What to update/add
home/HomeScreen.kt	Add a new exercise card on the Home screen
exercise/ExerciseType.kt	Add a new element to the exercises enum
res/raw directory	Add a new video animation of the new exercise