MuscleMate System Specification

Department of Computer Science, Seattle Pacific University
CSC3150: Systems Design
Professor: Andy Cameron
Student: Allie Miller

Table of Contents

1. Executive Summary

This system specification document analyzes MuscleMate, a specialized fitness-tracking app designed for bodybuilders and powerlifters. The goal is to create a user-friendly application that meets the specific needs of advanced lifters, focusing on efficiency and customization.

Document Coverage:

- **Introduction**: Highlights the need for MuscleMate, emphasizing the lack of advanced features in existing fitness apps tailored for experienced lifters. It also outlines the project's vision to offer a streamlined, customizable workout logging tool.
- **System Capabilities**: Detailed descriptions of the app's functionalities, including user registration, workout logging, template creation, progress tracking, and data synchronization across iOS devices.
- **Structural Model**: UML class diagrams and metadata descriptions that show the system's architecture and the relationships between different components.
- **Architecture Design**: An overview of the system's three-tier architecture, including deployment diagrams and infrastructure details, such as server and network configurations.
- **User Interface Design**: Wireframes and navigation diagrams that illustrate the app's layout and user interactions, ensuring an intuitive and cohesive experience.
- **Security Plan**: Comprehensive security measures to protect user data, including encryption, multi-factor authentication, and protection against unauthorized access and cyber threats.

Current Progress:

The project has completed its initial planning stages, including defining functional and non-functional requirements, feasibility analysis, and risk assessment. Functional prototypes showcasing the core features of MuscleMate have been developed, providing a foundation for the subsequent phases of development.

Next Steps:

The upcoming phase involves the implementation of the app's core functionalities, rigorous testing, and security assessments to ensure the robustness and reliability of MuscleMate. The development team will focus on creating a seamless user experience by refining the app's features and addressing any identified risks.

Conclusion:

MuscleMate is poised to fill a gap in the market for advanced lifters seeking a highly customizable and efficient workout tracking tool. This document is a comprehensive guide for the development team, outlining the technical and design specifications necessary to achieve the project's goals. By adhering to these guidelines, the development team can ensure the successful creation and deployment of a high-quality fitness-tracking app that meets the needs of its target users.

2. Introduction

2.1. Problem Statement / Project Vision

MuscleMate is a workout-tracking app designed for advanced lifters like bodybuilders and powerlifters. It addresses the limitations in current fitness applications that predominantly cater to beginners, with extensive tutorials and pre-made exercise templates. Unlike these apps, MuscleMate focuses solely on quick workout logging, customizable templates, and comprehensive progress tracking.

The vision for MuscleMate is to redefine the approach to fitness tracking by offering an intuitive and efficient tool tailored to meet the needs of experienced lifters. The app is designed to eliminate beginner-oriented content and provide a clean environment that maximizes usability for its target audience. This focused approach ensures that users can enhance their training efficiency without navigating through irrelevant features.

Stakeholders like advanced lifters, fitness coaches, project sponsors, and potential collaborators each have specific expectations that MuscleMate aims to fulfill. Advanced lifters require a highly customizable and efficient app; fitness coaches look for accurate progress-tracking features to better manage client workouts; the project sponsor, Andy Cameron, focuses on ensuring the app meets high-quality standards and aligns with user needs; and collaborators seek opportunities for mutual benefits. MuscleMate addresses these needs through its specialized features and strategic focus, positioning it as a leading solution in fitness tracking for advanced athletes.

2.2. System Capabilities

The Functional Requirements for MuscleMate are listed with their respective Use Case names and ID numbers. For more details, please refer to Section 4 of the System Proposal.

1. **User Registration and Authentication: Register and Login (UC-1)**
   Users must be able to register and log into their accounts using an email address, username, and password.
2. **Workout Logging: Log and Edit Workouts (UC-2)**
   Users must be able to log their workouts, including exercises, sets, reps, weights, and RPE, and edit these entries.
3. **Create Workout Template (UC-3)**
   Users must be able to create, save, and reuse customized workout templates.
4. **Track Progress (UC-4)**
   The app must create progress reports for the user's key lifts.
5. **Access Workout History (UC-5)**
   Users must be able to view their past workouts, organized and displayed with the workout's name and dates.
6. **Synchronize User Data (UC-6)**
   User data should be consistent across iOS devices.
7. **Administer User Accounts and App Maintenance (UC-7)**
   Administrators should manage user accounts and perform maintenance tasks like software updates and system monitoring.
8. **View Client Progress – Coaching Portal (UC-8)**
   Coaches should be able to view client progress and provide feedback.
9. **Export Workouts/Progress Graphs (UC-9)**
   Users can export their workout goals and progress graphs directly from MuscleMate to social media.

2.3. Non-functional Requirements and Design Constraints

This section outlines the non-functional requirements and constraints for MuscleMate. These are essential for ensuring the system operates efficiently, securely, and within operational limits. They provide a basis for evaluating the MuscleMate's success and measuring its performance. For a detailed breakdown, refer to Sections 3 (Feasibility) and 4 (Requirements)in the System Proposal.

2.3.1. Non-functional Requirements

- **Performance**
  - The app should load and display workout logs within 2 seconds.
  - The system shall support both online and offline functionalities to enhance user accessibility.
  - The system must prevent as many user errors as possible by restricting incorrect input through validation checks.
- **Security**
  - MuscleMate will encrypt user data to prevent unauthorized access.
  - Multi-factor authentication will be implemented to verify user identities, enhancing security during login.
- **Availability**
  - The system aims for 99% uptime with well-defined maintenance schedules to ensure reliable access.
- **Scalability**
  - The backend will scale dynamically to support up to 10,000 concurrent users without degradation in performance.
- **Compliance**
  - The app will comply with accessibility standards and cater to a diverse user base.

2.3.2. Constraints

- **Platform Limitation**
  - It is only available on iOS, restricting user reach but simplifying the development and testing phases.
- **Budget and Time**
  - Development is constrained by budget limitations and tight schedules, which may necessitate prioritizing certain features over others.
- **Resource Limitation**
  - With a small development team, the pace of development might be slower, potentially impacting the project's timelines.
- **Market Focus**
  - Targeting a niche market of advanced lifters could limit initial user adoption but provides a focused market entry strategy.

2.3.3.Additional Comments

The success of MuscleMate depends on meeting the non-functional requirements and effectively managing the outlined constraints. While focusing on user convenience and data privacy is essential, adhering to compliance standards like GDPR is critical to maintaining user trust and meeting legal obligations.

2.4. System Evolution

As MuscleMate continues to evolve beyond its MVP release, our development strategy includes introducing more dynamic and engaging features. These enhancements are designed to improve user experience, functionality, and scalability, responding directly to the evolving needs of our user base.

2.4.1. Version 2 Changes

Expanded Social Interaction:

- Integrate social media functionalities, enabling users to post their daily workouts, share workout progress graphs, and upload photos from the gym. The platform will allow users to add friends, who can then like and comment on each other's posts, creating a vibrant and interactive fitness community.

Advanced Coaching Tools:

- Upgrade the coaching portal to allow coaches to create and share customized workout templates. This feature will enable coaches to design tailored workout plans that can be directly applied to their clients' training regimens.

2.4.2. Version 3 and beyond Changes

Nutrition Tracking Integration:

- Add nutrition tracking features that allow users to log their daily food intake, track macronutrients, and calories, and see how their diet interacts with their fitness goals.

Wearable Technology Integration:

- Extend support to include syncing with wearable devices like Apple Watch and other popular fitness trackers to provide users seamless real-time health data integration.

Infrastructure and Security Enhancements:

- **Robust Cloud Infrastructure**: Enhance the server infrastructure to support the expanded user base and increased data loads without compromising performance.
- **Rigorous Security Measures**: Strengthen security protocols with advanced encryption methods and regular security assessments to safeguard user data, especially as the app integrates more deeply with social features and external devices.

Conclusion

MuscleMate is committed to continuous improvement and innovation, ensuring that each new version meets and exceeds user expectations. Our roadmap is shaped by user feedback, which guides the development of features that make MuscleMate an indispensable tool for fitness enthusiasts. We aim to create an engaging platform that supports users' fitness journeys while fostering a strong and supportive community.

2.5. Document Outline

The following sections of this System Specification will cover multiple aspects of the design of the System:

1. **Structural Model**
   - **Model Introduction**: Overview of the system structure and its components.
   - **Class Diagrams**: UML diagrams showing system classes, attributes, methods, and relationships.
   - **Metadata Descriptions**: Detailed descriptions of each class and its role within the system.
2. **Architectural Design**
   - **Architecture Overview**: High-level view of the system architecture.
   - **Infrastructure Model**: Detailed infrastructure setup, including server and network configurations.
   - **Deployment Diagrams**: Visual representations of infrastructure and deployment strategies.
3. **Hardware and Software Requirements**:
   - **Hardware Specifications**: Necessary hardware components for system operation.
   - **Software Stack**: Software components and platforms used in MuscleMate.
4. **Non-functional Requirements and Design Constraints**

- **Performance Requirements**: Speed, responsiveness, and efficiency metrics.
- **Security Specifications**: Security protocols, data protection measures, and compliance standards.
- **Scalability Plans**: Approaches to handle growth in user base and data volume.
- **Compliance and Accessibility**: Compliance with legal standards and accessibility guidelines.

5. **User Interface Design**
    - **UI Requirements and Constraints**: Specific design requirements and limitations.
    - **Navigation Maps and Flows**: How users will move through the app; includes UML state diagrams.
    - **Wireframes**: Visual drafts of UI screens across different app states.
    - **Interaction Design**: Description of the interaction between the user and the system.
6. **Appendices**
    - **Glossary**: Definitions of technical terms used within the document.
    - **References/Bibliography**: Sources and references that have informed the design of MuscleMate.
    - **Supporting Documentation**: Additional documents that provide further details or context.
7. **Conclusion**
    - **Summary of Specifications**: Recap of the key points from the specification document.
    - **Future Work and Considerations**: Additional notes on potential future enhancements not covered in previous versions.

3. Structural Model

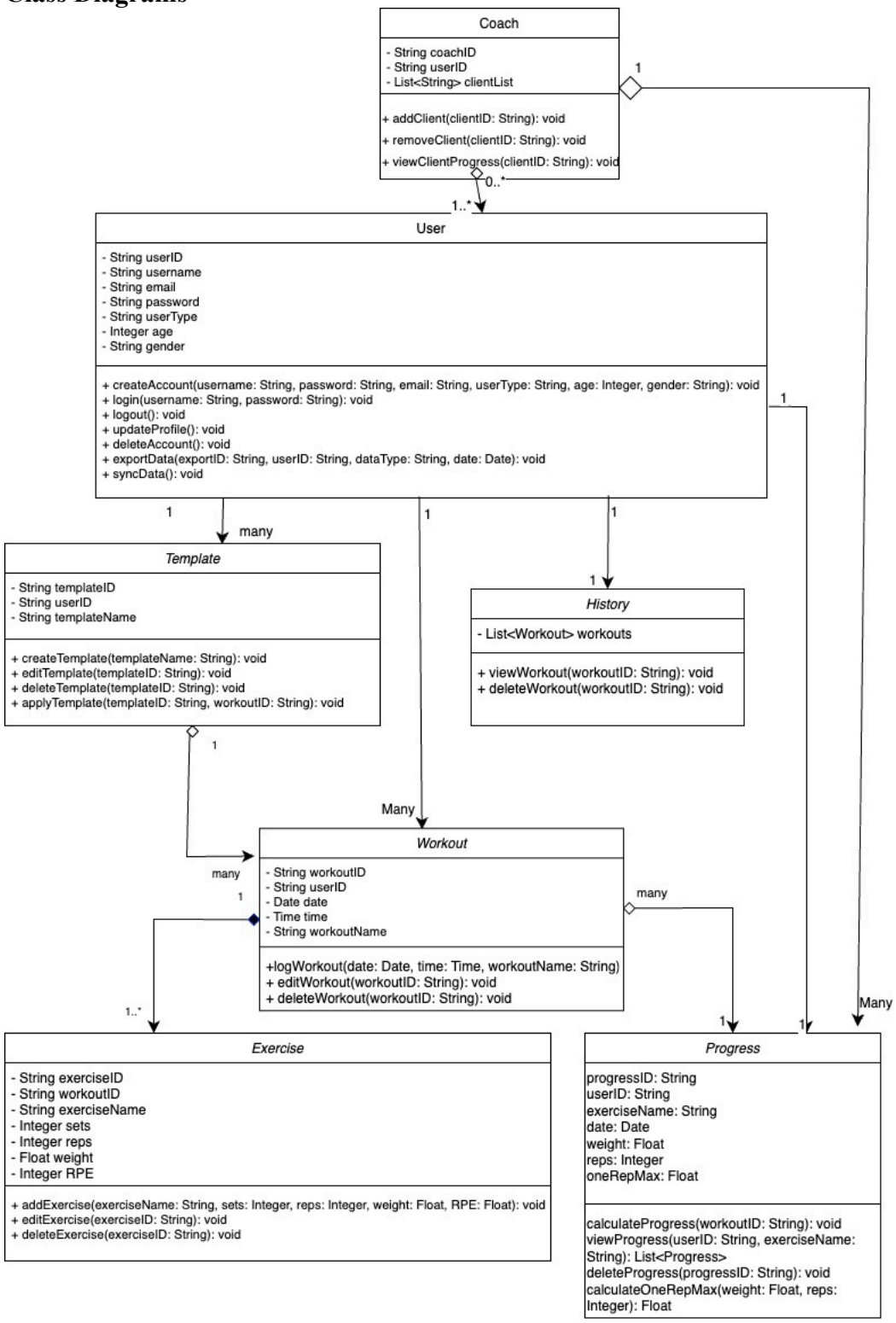
3.1. Model Introduction

This section outlines the structural framework of MuscleMate using Unified Modeling Language (UML). UML is a way to visually represent the structure and behavior of a system using diagrams. This section includes:

- **Class Diagrams (Section 3.2, page 9)**: Visual representations detailing each class's relationships, attributes, and methods within the system. These diagrams illustrate the system's architecture and the interconnections between classes.
- **Metadata Descriptions (Section 3.3, pages 10-18)**: Detailed descriptions of each class's role, attributes, and the operational logic of their methods. This section deepens understanding of each class's functionality and its contribution to the system.

Sections 3.2 and 3.3 are designed to complement each other, providing developers with a clear and detailed understanding of the system's structure and behavior.

3.2. Class Diagrams



https://drive.google.com/file/d/1yO0NHwpt-Ahl40RdW04k3GfR8QQdA2XY/view?usp=sharing

3.3. Metadata

The Metadata section provides an in-depth look into each class within the MuscleMate application, showing their attributes, methods, and relationships. This section is designed to give developers a comprehensive understanding of the system's structure and functionality. The descriptions include the purpose of each class, visibility, and high-level pseudocode for the methods to facilitate implementation. Below is the table of contents listing each class in alphabetical order:

Coach Class Diagram

- **Description**: Represents a coach who manages clients (users).
- **Visibility**: Public
- **Is Abstract**: No

Additional Information:

Attributes:

Name	Description	Read Only	Multiplicity
coachID	Unique identifier for the coach	Yes	1
userID	Identifier of the user who is a coach	Yes	1
clientList	List of user IDs representing clients	No	0..*

Operations:

Name	Description	Is Query	Is Polymorphic
addClient	Adds a client to the coach's client list	No	No
removeClient	Removes a client from the client list	No	No
viewClientProgress	Views the progress of a specified client	No	No

Processing Outlines:

- **addClient(clientID: String)**:
  - Append the clientID to clientList.
  - Return void.
- **removeClient(clientID: String)**:
  - Remove the clientID from clientList.
  - Return void.
- **viewClientProgress(clientID: String)**:
  - Fetch the progress data for the given clientID.
  - Display the progress details.
  - Return void.

Exercise Class Diagram

- **Description**: Represents an exercise within a workout.
- **Visibility**: Public
- **Is Abstract**: No

Attributes:

Name	Description	Read Only	Multiplicity
exerciseID	Unique identifier for the exercise	Yes	1
workoutID	Identifier of the associated workout	Yes	1
exerciseName	Name of the exercise	No	1
sets	Number of sets	No	1
reps	Number of reps	No	1
weight	Weight used	No	1
RPE	Rate of Perceived Exertion	No	1

Operations:

Name	Description	Is Query	Is Polymorphic
addExercise	Adds a new exercise	No	No
editExercise	Edits an existing exercise	No	No
deleteExercise	Deletes an exercise	No	No

Processing Outlines:

- **addExercise(exerciseName: String, sets: Integer, reps: Integer, weight: Float, RPE: Integer)**:
  - Validate input parameters.
  - Create a new exercise record.
  - Return void.
- **editExercise(exerciseID: String)**:
  - Fetch exercise details based on exerciseID.
  - Update the exercise with new details.
  - Return void.
- **deleteExercise(exerciseID: String)**:
  - Remove the exercise record based on exerciseID.
  - Return void.

History Class Diagram

- **Description**: Represents the user's history of workouts.
- **Visibility**: Public
- **Is Abstract**: No

Attributes:

Name	Description	Read Only	Multiplicity
workouts	List of workout records	No	1

Operations:

Name	Description	Is Query	Is Polymorphic
viewWorkout	Views details of a workout	No	No
deleteWorkout	Deletes a workout	No	No

Processing Outlines:

- **viewWorkout(workoutID: String)**:
  - Fetch workout details based on workoutID.
  - Display workout details.
  - Return void.
- **deleteWorkout(workoutID: String)**:
  - Remove the workout record based on workoutID.
  - Return void.

Progress Class Diagram
Description: Represents the progress tracking for a user's exercises.
Visibility: Public
Is Abstract: No

Attributes:

Name	Description	Read Only	Multiplicity
progressID	Unique identifier for the progress	Yes	1
userID	Identifier of the user	Yes	1
exerciseName	Name of the exercise	No	1
date	Date of the progress entry	No	1
weight	Weight used	No	1
reps	Number of reps	No	1

Operations:

Name	Description	Is Query	Is Polymorphic
logProgress	Logs a new progress entry	No	No
viewProgress	Views the progress of a user	No	No
deleteProgress	Deletes a progress entry	No	No

Processing Outlines:

1. logProgress(exerciseName: String, date: Date, weight: Float, reps: Integer):
    - Validate input parameters.
    - Create a new progress record.
    - Return void.
2. viewProgress(userID: String):
    - Fetch progress details for the given userID.
    - Display progress details.
    - Return void.
3. deleteProgress(progressID: String):
    - Remove the progress record based on progressID.
    - Return void.

Template Class Diagram

- **Description**: Represents a template for workouts that can be reused.
- **Visibility**: Public
- **Is Abstract**: No

Attributes:

Name	Description	Read Only	Multiplicity
templateID	Unique identifier for the template	Yes	1
userID	Identifier of the user who created it	Yes	1
templateName	Name of the template	No	1

Operations:

Name	Description	Is Query	Is Polymorphic
createTemplate	Creates a new template	No	No
editTemplate	Edits an existing template	No	No
deleteTemplate	Deletes a template	No	No
applyTemplate	Applies a template to a workout	No	No

Processing Outlines:

- **createTemplate(templateName: String)**:
  - Validate input parameters.
  - Create a new template record.
  - Return void.
- **editTemplate(templateID: String)**:
  - Fetch template details based on templateID.
  - Update template with new information.
  - Return void.
- **deleteTemplate(templateID: String)**:
  - Remove the template record based on templateID.
  - Return void.
- **applyTemplate(templateID: String, workoutID: String)**:
  - Apply the template to the specified workout.
  - Return void.

User Class Diagram

- **Description**: Represents a user of the MuscleMate app.
- **Visibility**: Public
- **Is Abstract**: No

Attributes:

Name	Description	Read Only	Multiplicity
userID	Unique identifier for the user	Yes	1
username	Username of the user	No	1
email	Email address of the user	No	1
password	Password for user authentication	No	1
userType	Type of user (e.g., Coach, Lifter)	No	1
age	Age of the user	No	1
gender	The gender of the user	No	1

Operations:

Name	Description	Is Query	Is Polymorphic
createAccount	Creates a new user account	No	No
login	Authenticates the user	No	No
logout	Logs out the user	No	No
updateProfile	Updates user profile information	No	No
deleteAccount	Deletes the user account	No	No
exportData	Exports user data	No	No
syncData	Synchronizes user data across devices	No	No

Processing Outlines:

- **createAccount(username: String, password: String, email: String, userType: String, age: Integer, gender: String)**:
  - Validate input parameters.
  - Create a new user record.
  - Return void.
- **login(username: String, password: String)**:
  - Validate the provided credentials.
  - Set user session.
  - Return void.
- **logout()**:
  - Clear user session.
  - Return void.
- **updateProfile()**:
  - Fetch the current user profile.
  - Update the profile with new information.
  - Return void.
- **deleteAccount()**:
  - Remove user records from the database.
  - Clear user session.
  - Return void.
- **exportData(exportID: String, userID: String, dataType: String, date: Date)**:
  - Fetch data based on the exportID, userID, dataType, and date.
  - Format the data for export.
  - Return void.
- **syncData()**:
  - Fetch the latest user data.
  - Synchronize data across devices.
  - Return void.

Workout Class Diagram

- **Description**: Represents a workout session containing multiple exercises.
- **Visibility**: Public
- **Is Abstract**: No

Attributes:

Name	Description	Read Only	Multiplicity
workoutID	Unique identifier for workout	Yes	1
userID	Identifier of the user	Yes	1
date	Date of the workout	No	1
time	Time of the workout	No	1
workoutName	Name of the workout	No	1

Operations:

Name	Description	Is Query	Is Polymorphic
logWorkout	Logs a new workout	No	No
editWorkout	Edits an existing workout	No	No
deleteWorkout	Deletes a workout	No	No

Processing Outlines:

- **logWorkout(date: Date, time: Time, workoutName: String)**:
  - Validate the input parameters.
  - Create a new workout record.
  - Return void.
- **editWorkout(workoutID: String)**:
  - Fetch the workout details based on workoutID.
  - Update the workout with new details.
  - Return void.
- **deleteWorkout(workoutID: String)**:
  - Remove the workout record based on workoutID.
  - Return void.

4. Architecture Design

4.1. Architecture Overview

MuscleMate will use a three-tier architecture model, which consists of the following layers:

1. **Presentation Layer:** This includes the user interface on iOS devices developed using Swift.
2. **Application Layer:** This includes the business logic, API services, and the Authentication Server:
    - **Web Server:** Manages core application logic and handles HTTP requests from the MuscleMate app.
    - **Authentication Server:** Managed user authentication, ensuring secure login and access control.
3. **Data Layer:** This includes relational (MySQL) databases for storing user data, workout logs, and progress tracking:
    - **Database Server:** Stores all necessary data for the application, including user profiles and workout information.

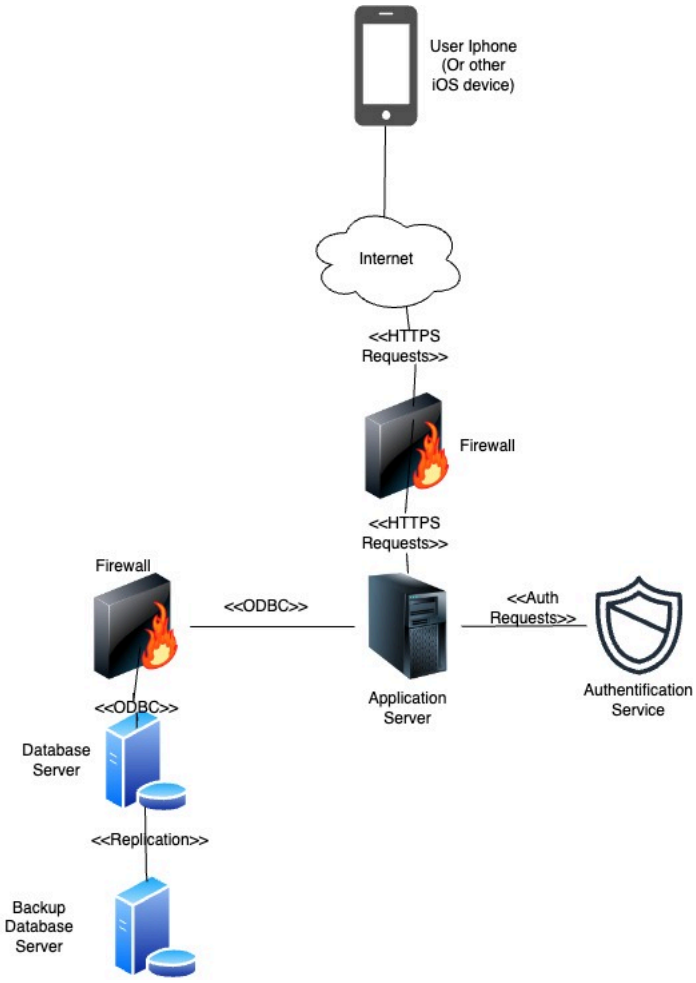
The following subsections provide detailed insights into the Infrastructure Model, Hardware and Software Requirements, and Security Plan:

- **Infrastructure Model**: This section presents two deployment diagrams. The first diagram illustrates the overall system structure, including the various components and their interactions. The second diagram details the physical and logical entities, their connections, and how elements are distributed across the infrastructure.
- **Hardware and Software Requirements**: This section outlines the hardware and software components required for the MuscleMate system. It includes specifications for servers, storage solutions, and the software tools and frameworks used in development and deployment.
- **Security Plan**: The security plan addresses the measures and strategies implemented to protect user data and ensure the integrity and confidentiality of information within the MuscleMate application.

4.2. Infrastructure Model

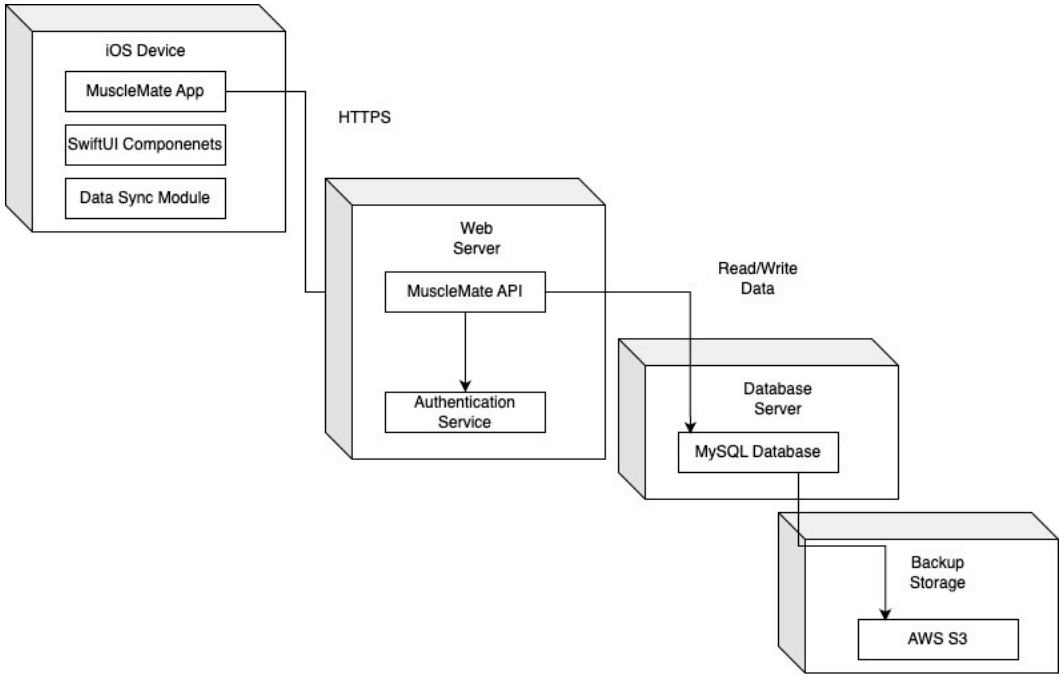
The two deployment diagrams below clearly show the significant components of MuscleMate's physical system architecture.

4.2.1. Deployment Diagram 1 – Architecture Overview



https://drive.google.com/file/d/14A93bwchDlBVYz_P3hu_XClAXD2nw6fW/view?usp=sharing

4.2.2. Deployment Diagram 2 – Nodes and Artifacts



https://drive.google.com/file/d/1Y3Wo4bxIZGqaR6MCG0zXFqpMd5noCTBv/view?usp=sharing

4.3. Hardware and Software Requirements

The following section will define the Hardware Components of this system's architecture and the software requirements that must be met to complete this project.

4.3.1. Hardware Components

The hardware components required for the implementation of MuscleMate are detailed below.

1. **Web Server:**
    - Service: AWS Elastic Compute Cloud (EC2)
    - Description: The MuscleMate web server will be hosted on AWS EC2, providing flexible resources that can scale according to user demand.
    - Alternative: A virtual private server (VPS) from providers like DigitalOcean or Linode for lower costs but potentially lower scalability and performance.
2. **Database Server:**
    - Service: AWS Relational Database Service (RDS)
    - Description: The application will use MySQL on AWS RDS for reliable and scalable database management.
    - Alternative: A self-hosted MySQL database on an on-premises server.
3. **Backup Database Server:**
    - Service: AWS Simple Storage Service (S3)
    - Description: AWS S3 will be used to store database backups for backup and disaster recovery. AWS S3 offers high durability, scalability, and security for data storage.
    - Alternative: Use an on-premises NAS (Network Attached Storage) device for local backups.
4. **Devices:**
    - **iOS Mobile Devices:** Users will need an iPhone or iPad running iOS 13 or higher to install and use the MuscleMate app.
    - **Development OS:** macOS is used to develop and publish iOS applications. This is necessary because Xcode, the IDE used for iOS development, only runs on macOS.
    - **Internet Access:** Users must have a reliable internet connection to interact with the app’s servers.

4.3.2. Required Software Components

The software components required for the implementation of MuscleMate are detailed below.

- **IDE:** Xcode is used to develop the iOS application. It offers extensions for a fast and productive workflow and is essential for iOS development.
- **Wireframe Tool:** Balsamiq for creating wireframes. Balsamiq helps design clear and user-friendly wireframes that are crucial during the planning phase.
- **Version Control:** Git for managing source code and collaboration among developers. Git is familiar and makes tracking changes and collaboration between teams easy.
- **API Framework:** Node.js with Express is used to create and manage the backend API services.
- **Firewall:** AWS Network Firewall to protect the infrastructure from unauthorized access and cyber threats.
- **UI Framework:** SwiftUI is used to build the user interface. SwiftUI is integrated with Xcode and provides a modern, declarative syntax for defining UI components.
- **Encryption:** GnuPG (GPG) is used to secure data transmission between users and servers.
- **Protection Against DDoS Attacks:** AWS Shield to protect against DDoS attacks.

4.4. Security Plan

4.4.1. Security Overview

The MuscleMate application faces several potential security threats that are crucial to address to ensure the safety and integrity of our user's data. The most significant security concerns include unauthorized access, data breaches, and service disruptions. Implementing robust security measures is essential to mitigate these risks.

Key Security Concerns:

1. **Unauthorized Access:** Ensuring only authorized users can access the system and sensitive data.
2. **Data Breaches:** Protecting user data from unauthorized entities' access or theft.
3. **Service Disruptions:** Preventing disruptions to the service, which cyber-attacks or other malicious activities could cause.

The following security software and measures are recommended to address these concerns and are included in section 4.3.2's "Required Software Components":

- **Encryption: GnuPG (GPG):**
  - **Description:** GnuPG (GPG) provides encryption methods for securing data during transmission and storage. It ensures that sensitive data remains confidential and protected from unauthorized access.
- **Firewall: AWS Network Firewall:**
  - **Description:** AWS Network Firewall is a firewall service that provides advanced protection against unauthorized access and cyber threats. It offers fine-grained

control over network traffic, allowing you to define security rules that filter traffic at the network level.

- **Protection Against DDoS Attacks: AWS Shield:**
  - **Description:** AWS Shield is a managed DDoS protection service that safeguards your web applications running on AWS from DDoS attacks. It automatically detects and mitigates attacks in real-time, ensuring minimal impact on your application's availability and performance. A Distributed Denial of Service (DDoS) attack is a malicious attempt to disrupt the regular traffic of a targeted server, service, or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic.

By implementing these security measures, MuscleMate will be better equipped to handle potential security threats and ensure a secure environment for its users.

4.4.2. Security Plan

Below is the system architecture threats table, which identifies potential threats associated with each component of the system architecture. This section also includes the strategies and measures to control and mitigate these threats.

Components	Physical	Network	Application	File	User Security		
	Loss	Denial of Service (DDoS) Attacks	Data Breaches	Corruption	Stolen User Credentials	Unauthorized Access to Accounts	Leakage User Personal and Privacy Violation
User Phone	11				4, 6	4, 6	4, 6, 12
Internet Connection		1, 2					
Application Server		1, 2	3, 5, 6, 7, 10	6, 9, 12		4, 6, 7	4, 6, 8, 12
Authentication Service		1, 2	3, 5, 6, 7, 10		4, 6	4, 6	4, 6
Database Server		1, 2	3, 5, 6, 7, 10	6, 9, 12		4, 6	4, 6, 8, 12
Backup Database Server	9	1, 2	3, 5, 6, 7, 10	6, 9, 12			4, 6, 12

Controls:

1. **AWS Shield**: Protects against DDoS attacks, ensuring that services remain available during an attack.
2. **AWS Network Firewall**: Adds a layer of network security, preventing unauthorized access and protecting against external threats.
3. **GnuPG (GPG)**: Encrypts sensitive data, preventing unauthorized access and ensuring data integrity during storage and transmission.
4. **In-app multi-factor authentication**: Adds an extra layer of security for accessing the application.
5. **Transport Layer Security (TLS) for data transmission**: Ensures data is encrypted during transmission to prevent interception.
6. **Access Control**: Protects sensitive data by restricting access based on roles.
7. **Regular Security Audits**: Periodically reviews and assesses security measures to identify and rectify vulnerabilities.
8. **User Activity Monitoring**: Tracks user behavior to detect unusual activities that may indicate a security threat.
9. **Secure Backup Solutions**: Regularly backs up data to secure locations to ensure recovery in case of data loss or corruption.
10. **Data Loss Prevention (DLP) Software**: Prevents unauthorized data transfer or leakage, protecting sensitive information.
11. **Physical Security Measures**: Protects physical access to devices and servers, preventing unauthorized access.
12. **Secure Coding Practices**: Ensures code is secure and up-to-date, reducing vulnerabilities in the application.

5. User-Interface

5.1. User-Interface Requirements and Constraints

MuscleMate's user interface aims to provide a streamlined and user-friendly experience tailored to the needs of advanced lifters. The primary focus is on functionality, efficiency, and ease of navigation. Here are the guiding UI principles and constraints:

Guiding Principles:

1. **Efficiency and Simplicity**:
   - **Minimal Clicks**: Users should be able to reach their desired actions within three clicks or less, ensuring a seamless experience.
   - **Clean Layout**: The interface shouldn't contain unnecessary elements that might distract or confuse the user. Each screen should focus on essential functionalities.
2. **User-Centric Design**:
   - **Customizability**: Users can customize their workout templates and progress-tracking features to suit their needs.
   - **Consistency**: The design should maintain consistency in fonts, colors, and layouts across all screens to provide a cohesive experience.
3. **Performance**:
   - **Fast Loading Times**: Each screen, especially those displaying user data such as workout logs and progress graphs, should load within 2.5-3 seconds.
   - **Responsiveness**: The interface should adapt seamlessly to different iOS devices, including various screen sizes and orientations.
4. **Security and Privacy**:
   - **Secure Data Handling**: User data should be encrypted and securely stored, including workout logs and progress graphs. Authentication mechanisms like multi-factor authentication should be implemented to enhance security.

Constraints:

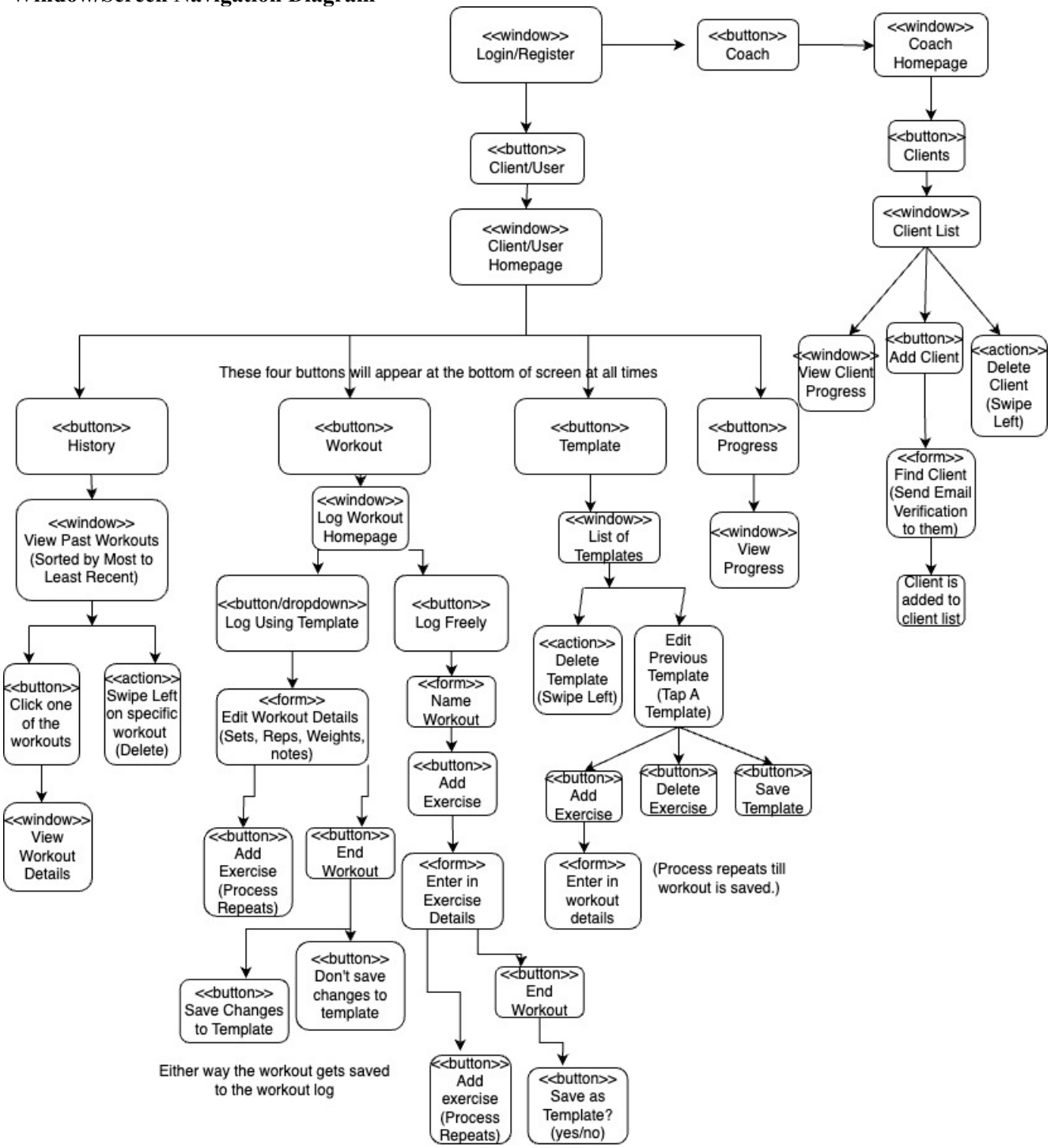
1. **Platform Specificity**:
   - **iOS Only**: MuscleMate is developed exclusively for iOS devices.
2. **Accessibility**:
   - **Compliance with Standards**: The app should comply with accessibility standards to accommodate users with disabilities, including features like voice-over support, adjustable font sizes, and high-contrast themes.
3. **Data Synchronization**:
   - **Cross-Device Sync**: User data must synchronize seamlessly across multiple iOS devices, ensuring users can access their information anytime, anywhere.
4. **User Feedback and Iteration**:
   - **Continuous Improvement**: The design should be flexible to incorporate user feedback and iterate on the interface to better meet user needs.

Overview of Sections:

1. **Navigation Diagram**: This section demonstrates how each button and navigation element will guide the user through the application, adhering to the principle of minimal clicks.
2. **UI Wireframes**: Rough drafts of screens, layouts, and page flows illustrating how users will interact with the app.

For more details on the user interface design principles and constraints, refer to the relevant sections in the System Proposal, particularly pages 17-20, where the functional and data requirements are thoroughly outlined.

5.2. Window/Screen Navigation Diagram



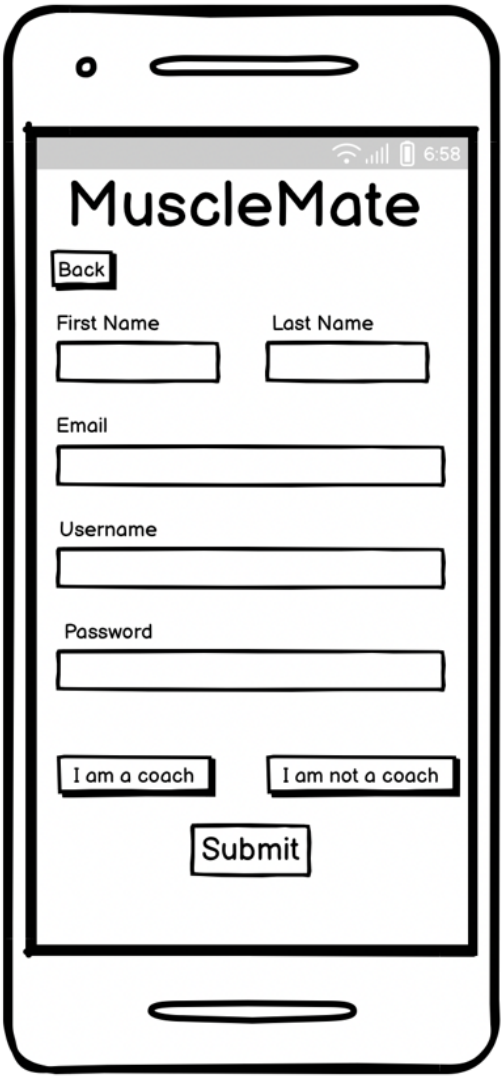
https://drive.google.com/file/d/111QjjE4dI_RzgbP2922Hg4da6HMbRycM/view?usp=sharing

5.3. UI Wireframes

1. Login Screen

After downloading MuscleMate, the user will see this screen once the app loads.

2. Sign Up Page

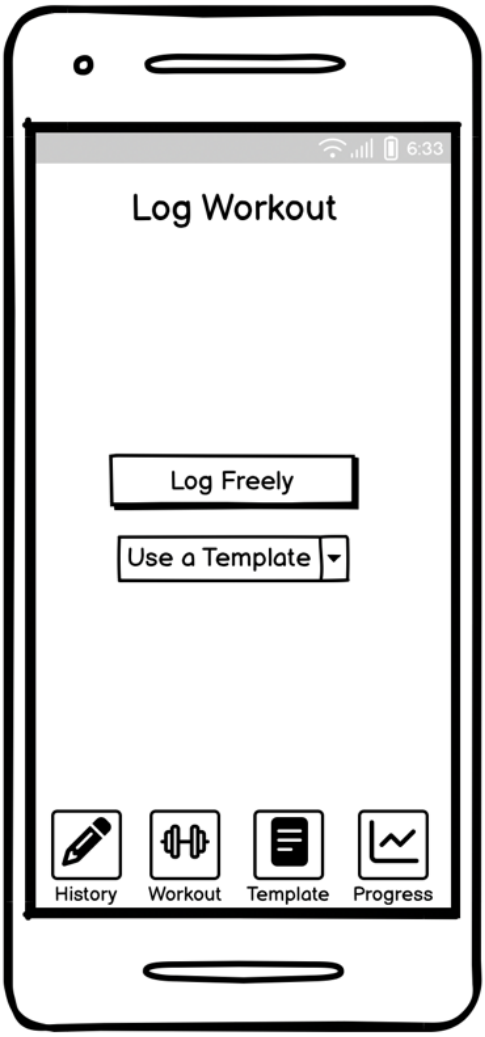


Users who press “Sign up” will be navigated to this page.

3. Homepage

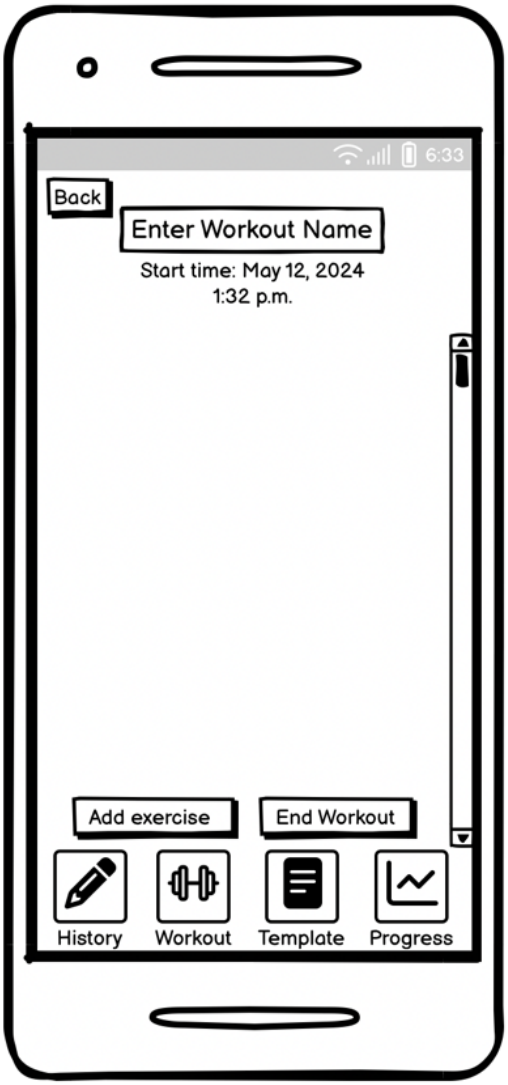
After a user logs in/signs up, they will be navigated to the USER homepage. The homepage is a window where users can navigate their workout history, start a workout, view progress, or view templates.
COACHES WILL NOT HAVE THE SAME HOMEPAGE.

4. Log Workout Homepage



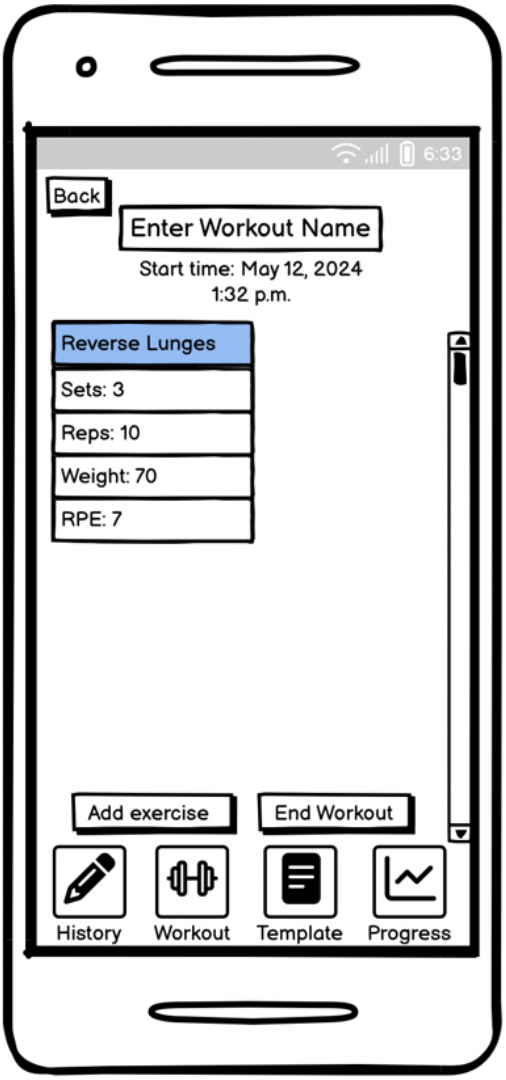
The user gets directed to the "Log Workout Homepage" after clicking the "Workout" icon. The user must choose between logging freely or using a template. If they choose "Use a template," they will pick their specific template from a dropdown.

5. Log Workout: Log Freely



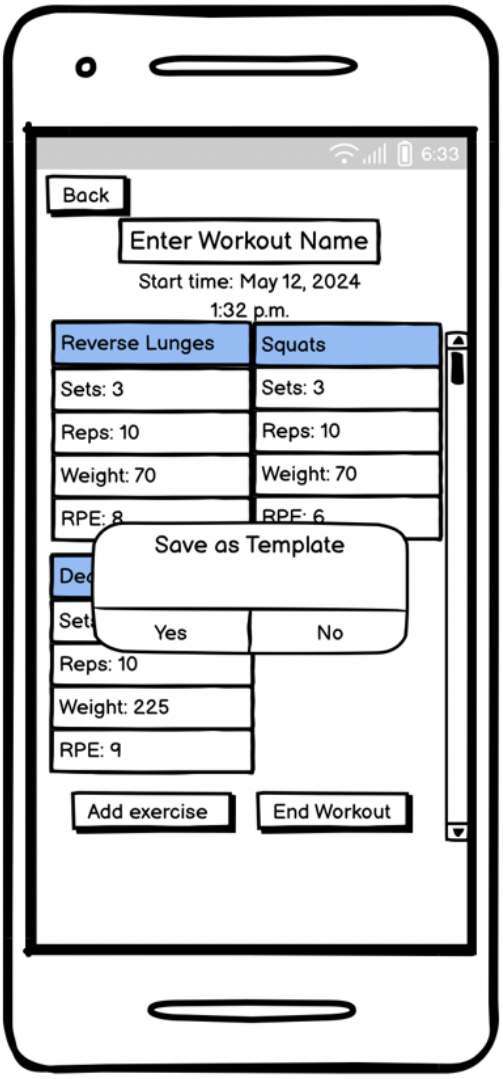
If the user presses “Log Freely,” they must add exercises as they work out. The user can press “Add exercise” to add another exercise. The user can also press “End Workout” to end their workout.

6. Log Freely: Add Exercise



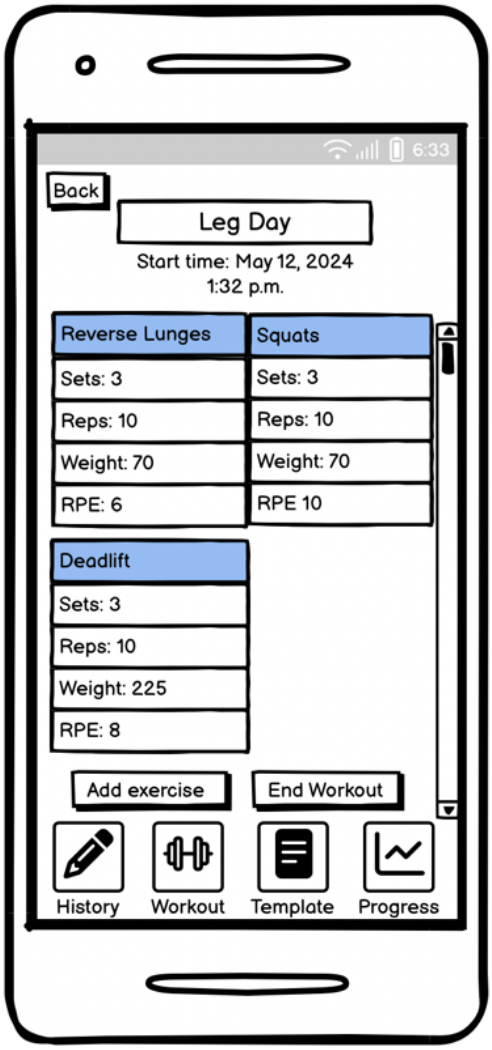
If the user presses “Add Exercise,” they will get a box to fill out their exercise, sets, reps, weight, and RPE (rate of perceived exertion).

7. Log Freely: Ending Workout



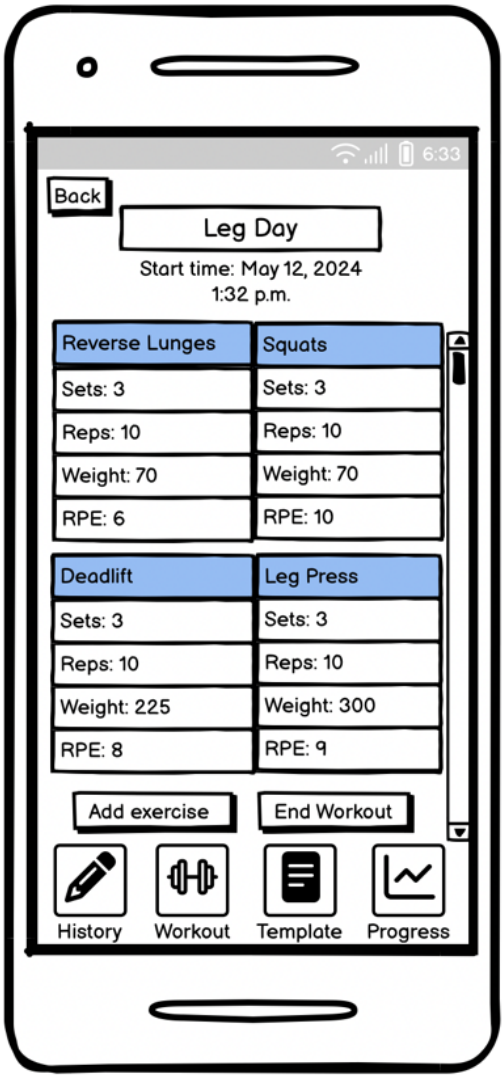
Once the user has finished their workout, they press “End Workout,” and the app will ask the user if they want to save the workout as a template. Either way, the workout gets logged into the user’s past workouts section.

8. Log Workout: Log Using Template



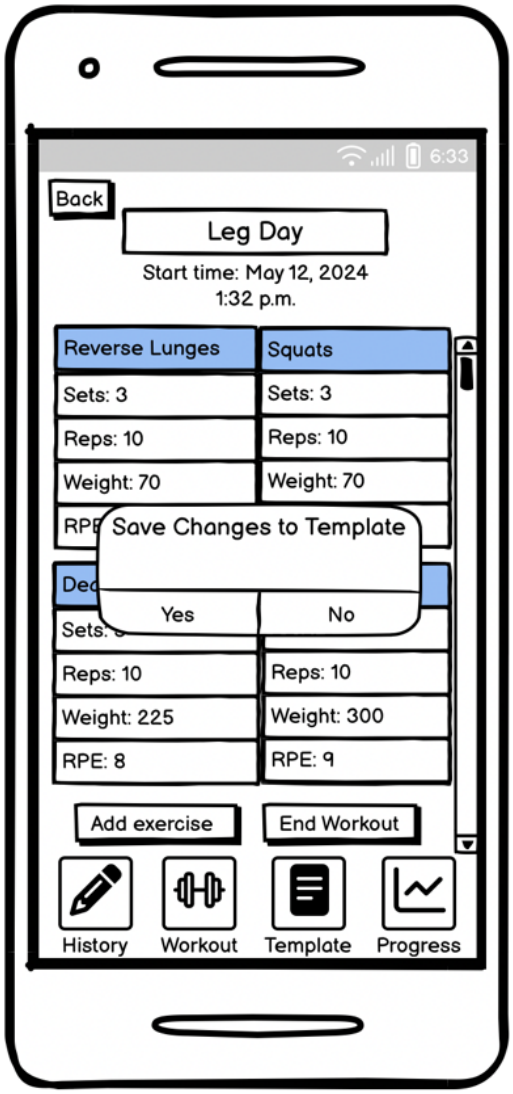
If the user presses “Use Template”, they will choose a template, and the system immediately places it in their logging place. The user can add other exercises if needed, and when they’re done, they press “End Workout”.

9. Log using Template: Add Exercise



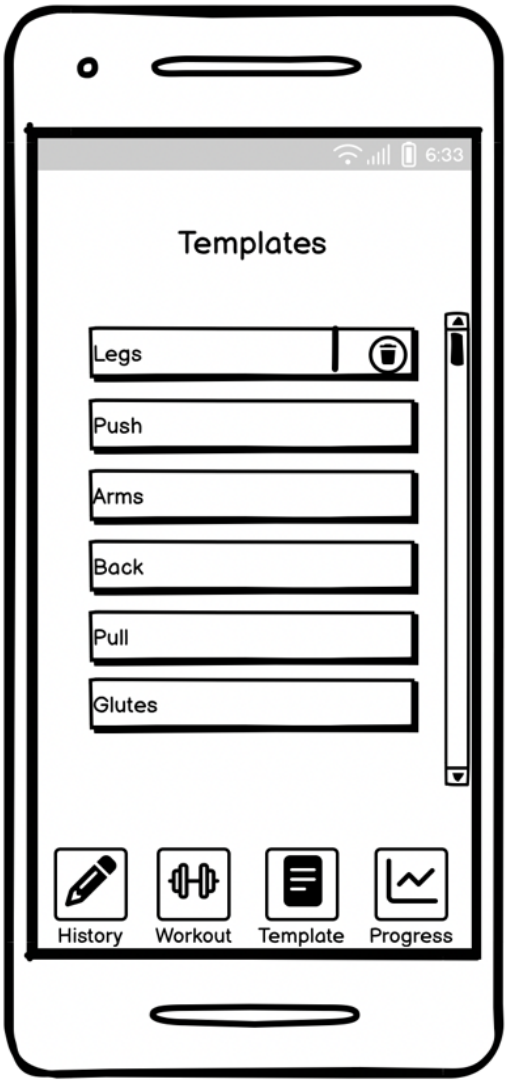
Users can add another exercise to their workout even though they use a template.

10. Log using Template: End Workout



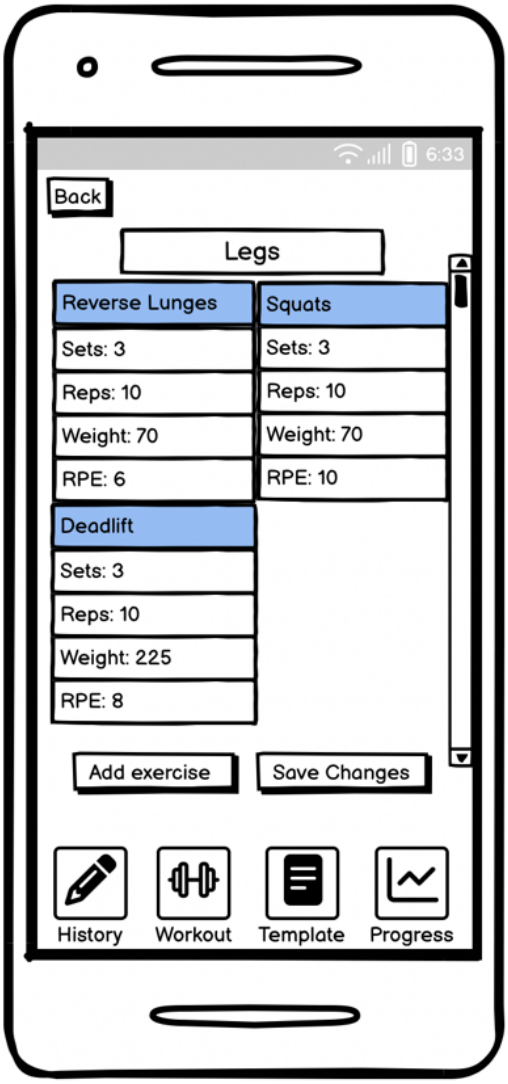
If the user presses “End Workout,” they can save the changes they made to the template. Either way, the workout is saved to the users’ “Workout History”.

11. Template Homepage



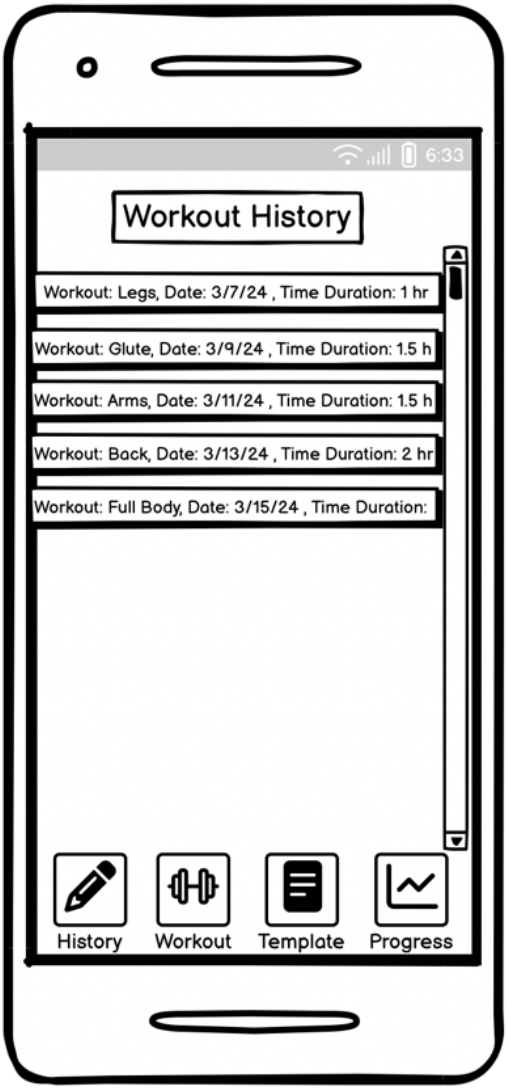
If the user presses the “Template” icon, they will be navigated here. If the user wants to delete a particular template, they can slide left on the template’s name to delete. Users can view/edit any template by pressing on it.

12. View/Edit Template



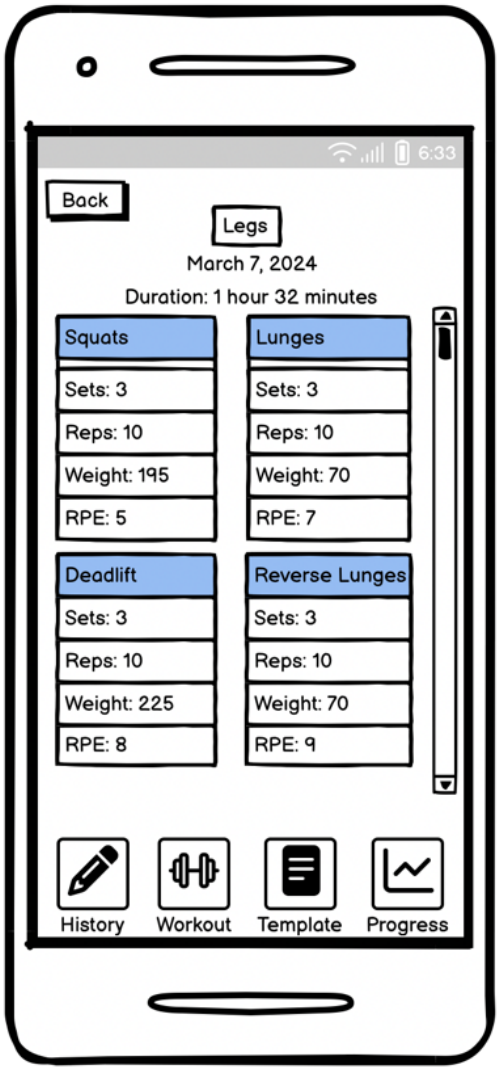
This slide shows if a user presses on one of the templates. Here, the user can view and edit their template and save changes when done.

13. Workout History



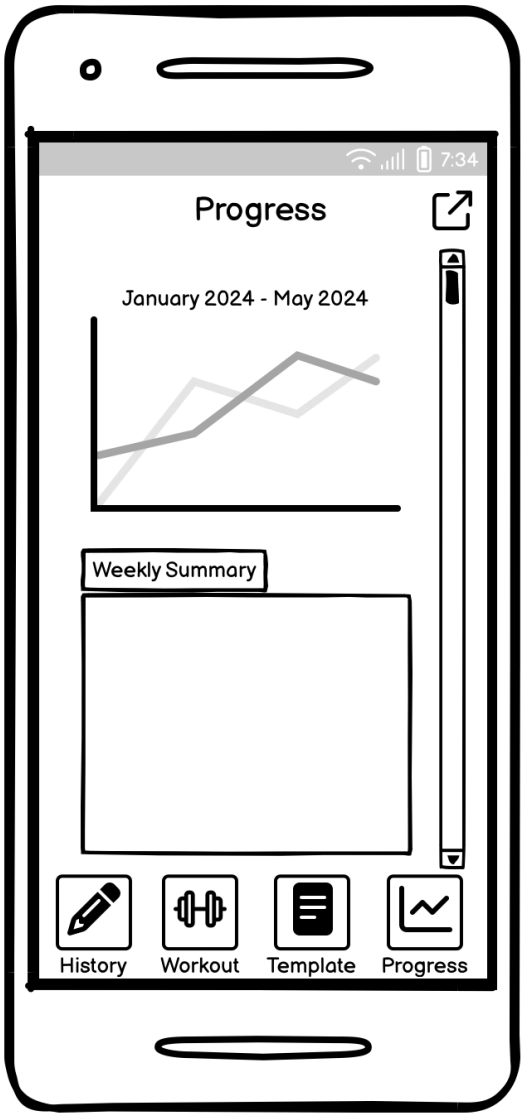
If the user presses the “History” icon, they will navigate here. The user can swipe left to delete a specific logged workout. The user can also tap any workout to view the details of it.

14. View Past Workout



If the user presses on a logged workout, they can see the workout details from that day but can’t edit them.

15. Progress Homepage

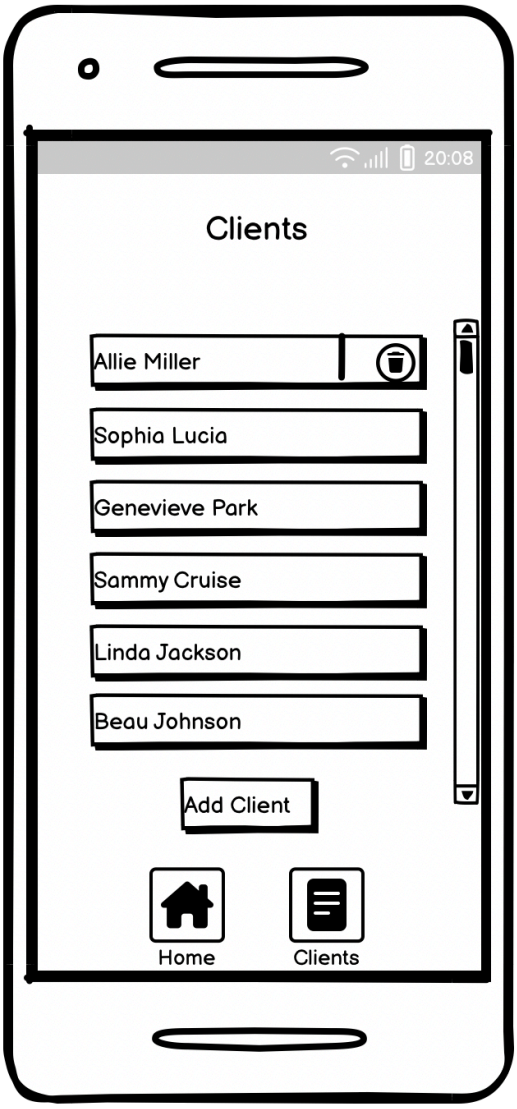


If the user clicks the “Progress” tab, they will be brought to this page. Here, the user can view their progress on squat, bench, and deadlift over some time of their choosing. They can also see their weekly summary.

16. Coach View - Homepage

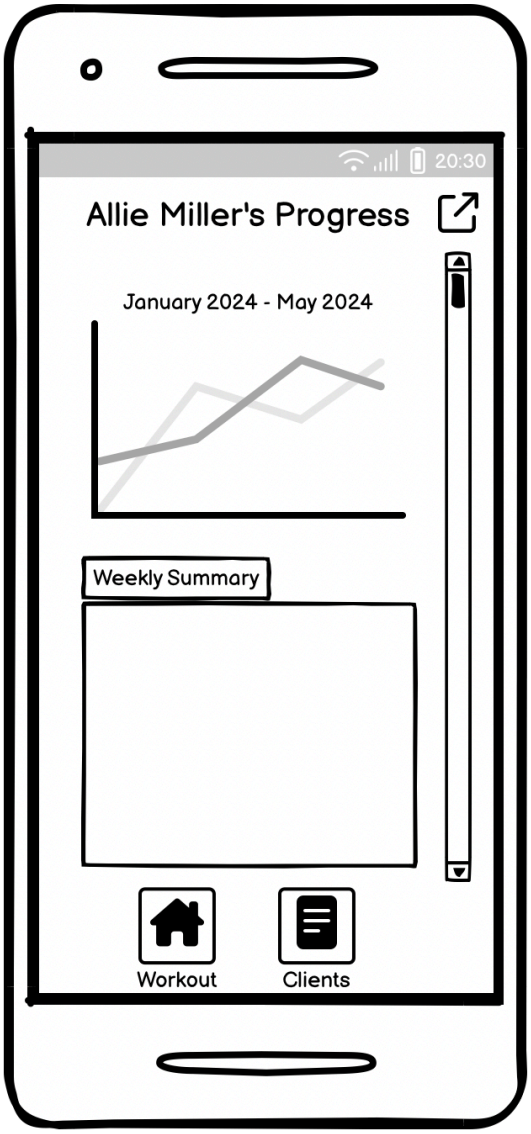
This screen shows the homepage for coaches’ view! It doesn’t have many features at the moment. In version two, another tab might be added called plan workouts, where a coach can plan the workouts for a client and make it a template for them.

17. Coach View – Client List



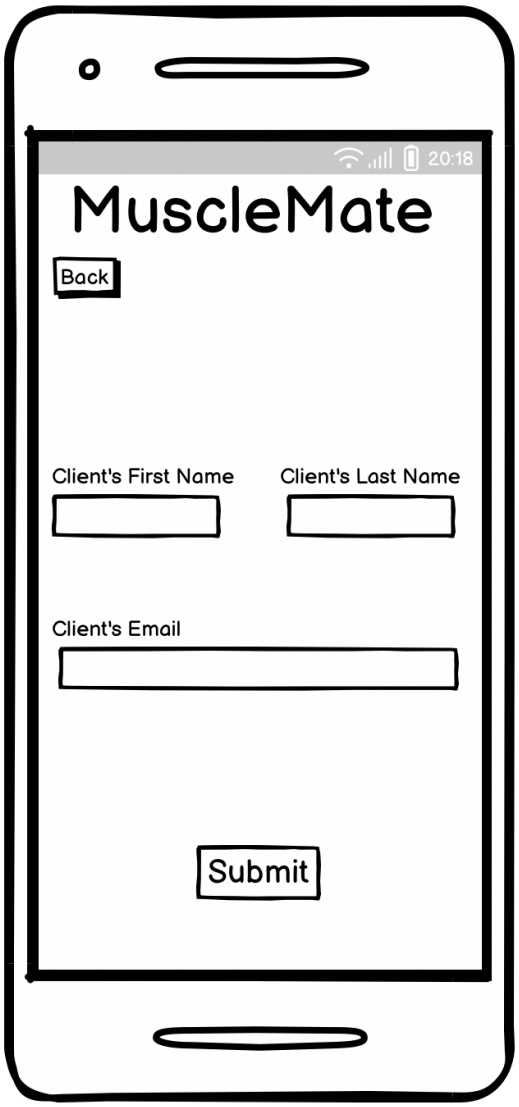
If a coach presses on the “Clients” tab, they will be brought here. This page shows where a coach can see and manage their clients. A coach can swipe left on a client’s name to delete them.

18. View Client Progress



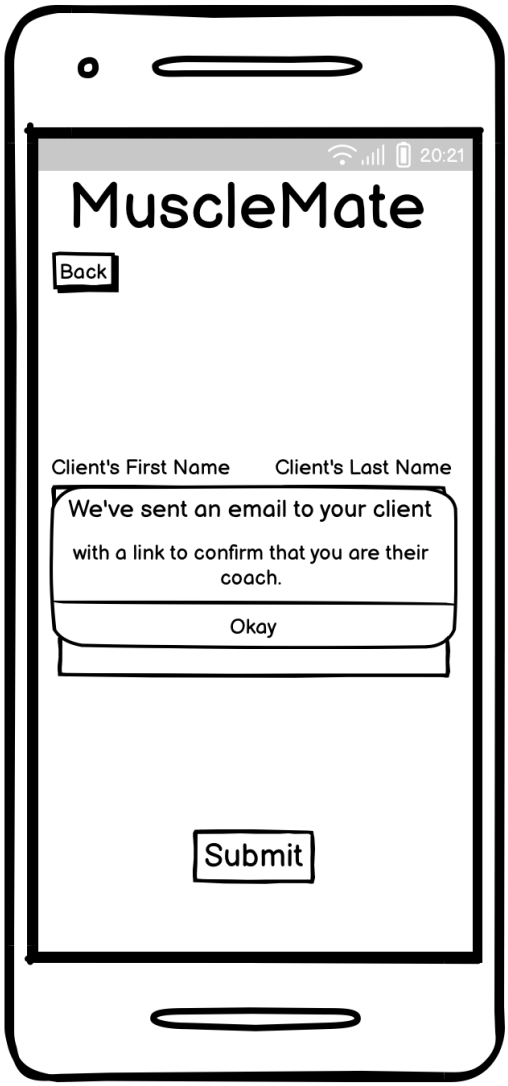
A coach can view their clients’ progress by pressing one of the listed client names. Later versions might allow trainers to see all past workouts as well.

19. Find Client



Users who press “Sign up” will be navigated to this page.

20. Find Clients Notification



After the coach submits the form, they are notified that the user will be sent a link to confirm the partnership!

5.4. Reports: “Formal Output” Design

MuscleMate allows users to export or print reports in the same layout as they are displayed on the IOS device, essentially producing a screenshot of the screen. This ensures a consistent experience, whether viewing information on the device or in a printed/exported format.

Screens to be Exported/Printed:

1. **Workouts in History Log:**
    - Users can print or export the details of a past workout precisely as they appear on the phone.
2. **Progress Tab:**
    - Users can print or export progress details, including graphs and metrics, in the same format displayed on the phone.
3. **Templates:**
    - Users can print or export workout templates with the same structure as viewed on the phone.

These features ensure users have a consistent experience, whether viewing the information on their device or in a printed/exported format.

6. Appendices

6.1. Glossary

Access Control: Restricts access to sensitive data based on roles.
API (Application Programming Interface): Protocols and tools for building and interacting with software applications.
Attribute: Property or characteristic of a class in object-oriented programming.
AWS Network Firewall: Provides network security, preventing unauthorized access and external threats.
AWS Shield: Protects against DDoS attacks, ensuring service availability.
Client-Server System: Network architecture where clients request services from a server.
Cloud Infrastructure: Remote servers are used over the internet to store, manage, and process data.
Data Loss Prevention (DLP) Software: Prevents unauthorized data transfer or leakage.
Data Synchronization: Ensures data consistency across multiple devices or systems.
Deployment Diagram: UML diagram showing the physical arrangement of hardware and software components.
Encryption: Converts data into a code to prevent unauthorized access.
Entity: Object that exists and is distinguishable from other objects, representing a real-world item.
GnuPG (GPG): Encrypts sensitive data, ensuring data integrity.
iOS: The operating system used by Apple devices like iPhone and iPad.
Log Workout: Recording details of a workout session.
Method: Function or procedure associated with a class in object-oriented programming.
Multi-factor authentication: This is a security method that requires more than one method of authentication.
Node: Physical device or location in a network where computing processes occur.
One-Rep Max: Maximum weight a person can lift for one repetition of an exercise.
Physical Security Measures: Protects physical access to devices and servers.
Progress Graph: Visual representation of data showing improvement or changes over time.
RPE (Rate of Perceived Exertion): Scale measuring exercise intensity based on personal perception.
Scalability: Ability of a system to handle increased work or be easily expanded.
Secure Backup Solutions: Regularly backs up data to secure locations for recovery.
Secure Coding Practices: Ensures code is secure and up-to-date, reducing vulnerabilities.
TLS (Transport Layer Security): Encrypts data during transmission to prevent interception.
Template: Predefined workout plan that can be reused.
UML (Unified Modeling Language): Standardized modeling language to visualize system design.
User Activity Monitoring: Tracks user behavior to detect unusual activities.
User Authentication: Verifies user identity before granting application access.
User Interface (UI): Space where interactions between humans and machines occur.
Wireframe: Basic visual guide suggesting the structure of a website or app.
Workout History: Record of all workout sessions logged by a user.

6.2. References / Bibliography

Amazon Web Services. Amazon RDS. Retrieved from https://aws.amazon.com/rds/

Amazon Web Services. Amazon S3. Retrieved from https://aws.amazon.com/s3/

Amazon Web Services. Amazon EC2. Retrieved from https://aws.amazon.com/ec2/

Balsamiq Studios. Balsamiq. Retrieved from https://balsamiq.com/

Coursera. What is an API? Retrieved from https://www.coursera.org/articles/what-is-an-api

draw.io. Retrieved from https://draw.io/

Git SCM. Git. Retrieved from https://git-scm.com/

Amazon Web Services. AWS Network Firewall. Retrieved from https://aws.amazon.com/network-firewall/

Amazon Web Services. AWS Shield. Retrieved from https://aws.amazon.com/shield/

Apple Developer. Xcode. Retrieved from https://developer.apple.com/xcode/

Apple Developer. SwiftUI documentation. Retrieved from https://developer.apple.com/documentation/swiftui

MDN Web Docs. Transport Layer Security (TLS). Retrieved from https://developer.mozilla.org/en-US/docs/Web/Security/Transport_Layer_Security

Larman, C. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative*. Third Edition.