

MOFIT

Undergraduate Project Proposal

|  |  |  |
| --- | --- | --- |
| Anthony Weathersby | [Antdog\_Indahouse@csu.fullerton.edu](mailto:Antdog_Indahouse@csu.fullerton.edu)  <https://github.com/AnthonyWeathersby99/Capstone-Project-MOFIT> | 889127700 |

CPSC 491-08

Professor Marc Velasco

12 December 2024

# Table of Contents

[Table of Contents 2](#_Toc185097069)

[Abstract 11](#_Toc185097070)

[Introduction 11](#_Toc185097071)

[Overview 12](#_Toc185097072)

[Problem Statement 13](#_Toc185097073)

[Problem Objective 13](#_Toc185097074)

[Target Audience 14](#_Toc185097075)

[Project Vision 14](#_Toc185097076)

[Vision Statement 14](#_Toc185097077)

[Elevator Pitches 14](#_Toc185097078)

[Capstone Summary 15](#_Toc185097079)

[Uniqueness 15](#_Toc185097080)

[Breadth of CS Knowledge 16](#_Toc185097081)

[Mobile Development 16](#_Toc185097082)

[Machine Learning 16](#_Toc185097083)

[Cloud Computing 16](#_Toc185097084)

[Features 17](#_Toc185097085)

[User Profile 17](#_Toc185097086)

[Progress Tracker 17](#_Toc185097087)

[Form Check 18](#_Toc185097088)

[Workout Library 18](#_Toc185097089)

[Application Architecture 19](#_Toc185097090)

[Chart Overview 19](#_Toc185097091)

[Architecture and Design 20](#_Toc185097092)

[Architecture Pattern: Model-View-Controller (MVC) 20](#_Toc185097093)

[Model 20](#_Toc185097094)

[View 21](#_Toc185097095)

[Controller 21](#_Toc185097096)

[MVC Diagram 21](#_Toc185097097)

[Scenario 21](#_Toc185097098)

[Design Pattern: Observer Pattern 23](#_Toc185097099)

[Observer Diagram 23](#_Toc185097100)

[Scenario 23](#_Toc185097101)

[UX Design 25](#_Toc185097102)

[Aesthetic 25](#_Toc185097103)

[Design Philosophy 25](#_Toc185097104)

[Color Scheme 25](#_Toc185097105)

[Primary Colors 25](#_Toc185097106)

[Neutral Colors 25](#_Toc185097107)

[Accent Colors 25](#_Toc185097108)

[Typography 26](#_Toc185097109)

[Font Family 26](#_Toc185097110)

[Font Sizes 26](#_Toc185097111)

[Line Spacing 26](#_Toc185097112)

[Interaction Design 27](#_Toc185097113)

[User Research 27](#_Toc185097114)

[Personas 28](#_Toc185097115)

[Persona 1: Steven Hill 28](#_Toc185097116)

[Background 28](#_Toc185097117)

[Goals 29](#_Toc185097118)

[Challenges 29](#_Toc185097119)

[Tech Usage 29](#_Toc185097120)

[How MOFIT Can Help 30](#_Toc185097121)

[Persona 2: Maggie Thompson 30](#_Toc185097122)

[Background 30](#_Toc185097123)

[Goals 30](#_Toc185097124)

[Challenges 31](#_Toc185097125)

[Tech Usage 31](#_Toc185097126)

[How MOFIT Can Help 31](#_Toc185097127)

[Persona 3: Robert Irvine 31](#_Toc185097128)

[Background 32](#_Toc185097129)

[Goals 32](#_Toc185097130)

[Challenges 32](#_Toc185097131)

[Tech Usage 32](#_Toc185097132)

[How MOFIT Can Help 33](#_Toc185097133)

[UX Framework 33](#_Toc185097134)

[BASICUX Framework 33](#_Toc185097135)

[What is BASICUX? 33](#_Toc185097136)

[Beauty 33](#_Toc185097137)

[Accessibility 34](#_Toc185097138)

[Simplicity 34](#_Toc185097139)

[Intuitiveness 34](#_Toc185097140)

[Consistency 35](#_Toc185097141)

[UX Architecture and Design 35](#_Toc185097142)

[Overview 35](#_Toc185097143)

[High-Level Information Architecture 35](#_Toc185097144)

[Navigation Model 36](#_Toc185097145)

[UX Mockup 36](#_Toc185097146)

[Mobile Development 38](#_Toc185097147)

[UI Design (Android) 38](#_Toc185097148)

[Layout 39](#_Toc185097149)

[Workout Interaction 39](#_Toc185097150)

[Platforms 40](#_Toc185097151)

[Android Integration 40](#_Toc185097152)

[Other Platforms 40](#_Toc185097153)

[Machine Learning 40](#_Toc185097154)

[Data Processing 40](#_Toc185097155)

[Motion Analysis 40](#_Toc185097156)

[Feature Extraction 41](#_Toc185097157)

[Model Development 41](#_Toc185097158)

[Android Architecture 41](#_Toc185097159)

[Model Deployment 42](#_Toc185097160)

[On-Device Inference 42](#_Toc185097161)

[Model Updates 42](#_Toc185097162)

[Cloud Computing 42](#_Toc185097163)

[Cloud Services Integration 43](#_Toc185097164)

[Virtual Machines 43](#_Toc185097165)

[Storage and Networking 43](#_Toc185097166)

[Database Services 43](#_Toc185097167)

[Regulatory Compliance 44](#_Toc185097168)

[Data Encryption 44](#_Toc185097169)

[Compliance Standards 44](#_Toc185097170)

[Scope 44](#_Toc185097171)

[Feasibility 45](#_Toc185097172)

[Activities 45](#_Toc185097173)

[Risk & Mitigations 48](#_Toc185097174)

[Release Plan / Future Support 50](#_Toc185097175)

[Prototype 51](#_Toc185097176)

[Installation 61](#_Toc185097177)

[Requirements 61](#_Toc185097178)

[How to Install 61](#_Toc185097179)

[Setup 61](#_Toc185097180)

[Navigation 62](#_Toc185097181)

[Service Level Documentation 63](#_Toc185097182)

[Overview 63](#_Toc185097183)

[Service Level Objectives (SLOs) 63](#_Toc185097184)

[Availability 63](#_Toc185097185)

[Performance 63](#_Toc185097186)

[Service Level Agreements (SLAs) 64](#_Toc185097187)

[Core Application Availability 64](#_Toc185097188)

[Real-Time Processing Performance 64](#_Toc185097189)

[Data Protection 64](#_Toc185097190)

[Service Level Indicators (SLIs) 64](#_Toc185097191)

[Performance Metrics 64](#_Toc185097192)

[User Experience Metrics 65](#_Toc185097193)

[Form Analysis Accuracy 65](#_Toc185097194)

[Application Launch Time 66](#_Toc185097195)

[Test Cases 66](#_Toc185097196)

[Test Planning 66](#_Toc185097197)

[Unit Tests 67](#_Toc185097198)

[Test Cases 67](#_Toc185097199)

[Test: User Log in with invalid authorization code 67](#_Toc185097200)

[Expected Results 67](#_Toc185097201)

[Bugs/Fixes 68](#_Toc185097202)

[Test: User Log in with valid Authorization Code 68](#_Toc185097203)

[Expected Results 69](#_Toc185097204)

[Bugs/Fixes 69](#_Toc185097205)

[Performance Tests 71](#_Toc185097206)

[Test Cases 71](#_Toc185097207)

[Test: Performance Test for Authentication 71](#_Toc185097208)

[Expected Results 73](#_Toc185097209)

[Bugs/Fixes 73](#_Toc185097210)

[Test: User Profile saving and loading 74](#_Toc185097211)

[Expected Result 76](#_Toc185097212)

[Security Tests 77](#_Toc185097213)

[Test Cases 77](#_Toc185097214)

[Test: API Request Headers 77](#_Toc185097215)

[Expected Results 78](#_Toc185097216)

[Test: AWS Credentials Security 78](#_Toc185097217)

[Expected Results 79](#_Toc185097218)

[Test: Password Storage Security 79](#_Toc185097219)

[Expected Results 80](#_Toc185097220)

[Test: Session Cache Encryption 81](#_Toc185097221)

[Expected Results 82](#_Toc185097222)

[Test: Token Refresh Security 82](#_Toc185097223)

[Expected Results 83](#_Toc185097224)

[Test: Token Format Validation 83](#_Toc185097225)

[Expected Results 84](#_Toc185097226)

[Test: URL Security 84](#_Toc185097227)

[Expected Results 85](#_Toc185097228)

[Test: User Profile Data Security 85](#_Toc185097229)

[Expected Results 87](#_Toc185097230)

[Conclusion 87](#_Toc185097231)

[References 89](#_Toc185097232)

# Abstract

My project aims to coach individuals who seek to improve their health by creating a motion-tracking software application that will guide them into performing exercises. Beginning with the introduction, we’ll present the problem statement and the key objective of our application. In the overview section, we state how our application will use machine learning algorithms to analyze the user’s body motions and determine by a certain percentage if the user is doing the exercise correctly. Uniqueness focuses on explaining the details of our app which sets it apart from others; many other fitness apps focus on rehabilitation or have hefty subscriptions which is what we are getting rid of to create an all-inclusive app. Following this, the breadth of our upper division courses will be explained, and how this project covers them. Then in our user stories, we will provide real-life examples of our key features and when they’d be useful. Finally, the conclusion will emphasize the need for our product in the current health & fitness landscape and the application’s dedication to creating a safer workout environment for its users.

Keywords— Motion-tracking, fitness, AI, reps, training, health

# Introduction

Motion Oriented Fitness Improvement Tracker, or MOFIT, represents a breakthrough in fitness tracking, acting as a knowledgeable coach who assesses your exercise form to optimize muscle development and ensure safety. Its core mission is to guide individuals who seek to enhance their health through weightlifting. Many people starting off in their fitness journey often do not have proper guidance and fail to self-correct bad habits concerning exercise. This lack of guidance on proper technique leads to ineffective workouts, which underscores the need for a platform that offers direct and personalized feedback. MOFIT seeks to address this problem by providing real-time, personalized feedback on exercise forms, which will help users correct and build healthier workout habits. MOFIT gets rid of the need for external devices by using only your phone’s camera to capture your movements. With access to a database of correct form and motion patterns, MOFIT empowers users to master proper lifting techniques efficiently and securely, fostering a rewarding and injury-free fitness journey.

# Overview

MOFIT is an application that strives to improve the health of its users by training their muscles using data to safely determine an individual’s proper rep range, number of sets, speed, and range of motion for their exercises. MOFIT will keep track of your progress throughout your entire fitness journey and adapt your workout plan as you increase your intensity so you never feel as if you are not making progress within the app. MOFIT will be developed as an app in Unity primarily for Android devices to utilize their cameras to track the user’s movements. Cloud computing will be used to manage the application to allow for easy scalability, security, availability, and cost efficiency. Machine learning algorithms will analyze the user’s exercises and determine whether improvements need to be made on form, reps, and sets. A database of optimal movements will be the base of the algorithms and will also store the user’s information and other fitness advice.

## Problem Statement

Maintaining proper form during exercises is a significant and overlooked challenge for individuals starting in their fitness journey. Incorrect form affects a beginner's ability to develop healthy habits and may even increase the risk of injury. Current fitness apps often focus on tracking metrics like calories burned or distance covered, with less emphasis on current workout form.

## Problem Objective

The objective of MOFIT is to enhance the workout experience of users by providing immediate, accurate feedback on exercise forms using machine learning techniques. The goal is to help users achieve proper exercise techniques, correct mistakes in real time, and help them achieve their fitness goals safely and effectively. Integrating intelligent video analysis on a user’s mobile device, MOFIT aims to become a personal training companion that supports users in performing exercises correctly.

## Target Audience

The primary target audience is fitness enthusiasts in the beginning stages of their journey. These individuals may lack access to professional coaching and prefer working out at home, thus needing guidance on proper exercise forms. By focusing on these groups, MOFIT addresses the needs of those who are most likely to benefit from mobile, real-time feedback on their workout routines.

# Project Vision

## Vision Statement

MOFIT strives to make the world a healthier place for everyone. Everyone will have the ability to progress their fitness journey with exercises they wish to improve on. We aim to improve our tracking technology to determine better when a user is performing an exercise for a variety of workouts. No longer will you need to pay hundreds of dollars to personal trainers. The goals of MOFIT are to use advanced tracking, machine learning, and encouragement to better the lives of our users. Achieve your fitness goals by being coached on your time.

## Elevator Pitches

1. Have you ever felt like you aren’t progressing with your fitness goals? The internet tells you to do one thing, and personal trainers say another thing. With MOFIT, we use a database of workouts that are done correctly and determine how well you perform your workout using precision tracking and machine learning. Next time someone asks how you achieved your goals, you tell them MOFIT helped me.
2. Everybody needs a starting point when beginning a new chapter in their life. Why not start on the right foot? MOFIT will get you to where you need to be with your fitness goals by using advanced machine learning and tracking that fits any body type. You’ll quickly become an expert in your form and be pushed to your limits with your personal AI trainer. Welcome to your future, where your mind and body are as strong as the AI that trained you.

# Capstone Summary

## Uniqueness

MOFIT is unique by setting itself apart from similar applications by creating ease of access. Other applications such as K-Coach have pricey subscriptions and expensive accessories that are required. No external devices are required to track the exercise movements, only your phone is required to use the app. The focus of this app is to safely and optimally grow muscles for all levels of experience from beginners to advanced lifters, while other applications such as Kemtai focus on therapy and rehabilitation. There will be no subscription paywalls or trial periods, and strive to be inclusive to all body types, so everyone who uses the app feels welcomed.

## Breadth of CS Knowledge

### Mobile Development

The MOFIT project involves the creation of a mobile application so that users can easily get personalized feedback anywhere they have their mobile device with them. The project will involve the use of Unity for platform development and the integration of device-specific APIs such as OpenCV and YOLOv8 to enable real-time video processing and tracking. By choosing this development environment, I will be able to better understand the challenges of optimizing mobile applications for performance and user experience.

### Machine Learning

Machine learning is at the core of the MOFIT application, which focuses on real-time analysis of user movements to provide feedback on exercise forms. The application will use advanced algorithms, including object detection and pose estimation models like YOLOv8 to analyze video data captured by the user’s device. This will expose me to the complexities of training and deploying machine learning models on mobile devices, which requires consideration of model efficiency and resource constraints.

### Cloud Computing

MOFIT will employ the use of cloud-based services to manage data storage, processing, and dynamic scaling of resources. By integrating services such as AWS, the MOFIT project will demonstrate how virtual machines, and managed databases can support the backend of the application. This will include handling complex processing tasks not performed on-device, securely managing user data, and storing workout videos.

# Features

## User Profile

As a user, I’d want to be able to create a user profile that will hold all my personal information and fitness goals. I want to log my workout preferences such as how many days I want to go a week or how much time I’d like to dedicate per day at most. I also want to log any existing injuries or limitations for my workout that the app AI trainer can be aware of to personalize my training better.

A user profile will help the app gather information on users’ exercise styles/preferences and tailor their workouts for them. This ties into the core idea of our app which is to create an AI fitness coach that will help you with your workouts and maximize growth. To give better advice, trainers usually get to know their clients first and the user profiles will help our app do that. This engages the user more intimately which could increase satisfaction overall.

## Progress Tracker

As a user, I want to be able to track my workouts to maximize my results by getting the proper amount of exercise and rest. I would like to use a progress tracker within the app to add workouts I’d like to do to my weekly split schedule and keep track of my sets/reps and weights being lifted. I want to keep track of my workouts to monitor my growth and do progressive overload over time to grow my muscles.

Progress tracking is a part of identifying growth. The progress tracker serves as a built-in digital notebook that keeps track of all users’ workouts and training progress. It is important to the core idea of our project because monitoring your workout is what helps you improve. Without something to look back on, it's hard to see how far you’ve come so a progress tracker will help users look back on their fitness journey and see their growth - encouraging them to keep going.

## Form Check

As a user, I want to be able to compare my exercise form with the optimal form so that I can understand how to improve my form. FormCheck should provide me with a visual comparison between my recorded exercise form and the optimal form for that exercise. It should highlight the difference between my form and the optimal way to do that exercise. This will help me make any necessary adjustments to improve my overall workout experience.

By providing the user with a clear visual comparison between their current exercise and the optimal form, users can easily correct and adjust their form. FormCheck will provide tailored coaching advice for safer and more effective workouts. By integrating this feature into MOFIT, we’ll help people approach fitness with a focus on precision. This ensures users aren’t just going through the motions but actively engaging in improving their techniques for better results and reduced injury risk.

## Workout Library

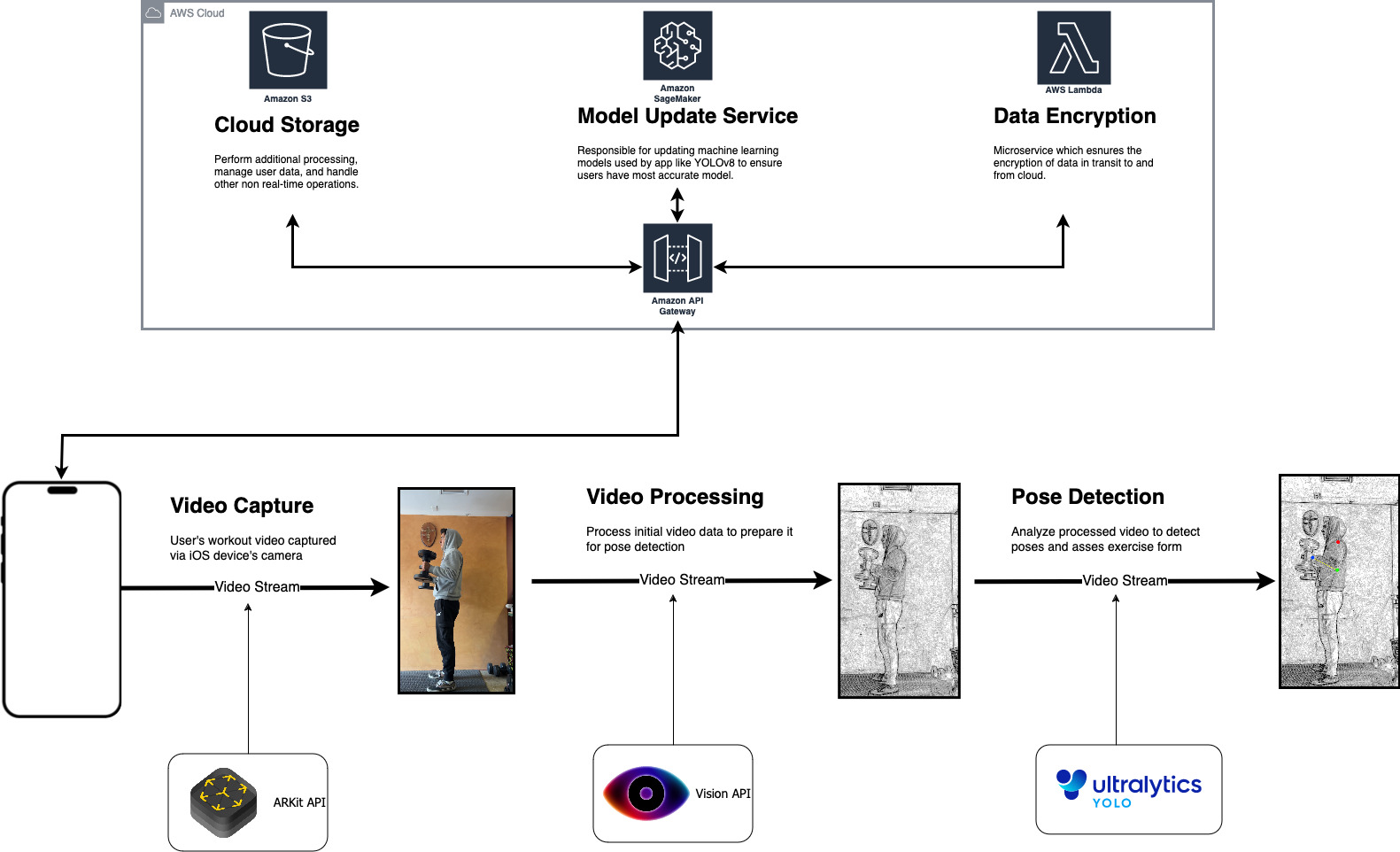
As a user, I want to access instructional video content so that I can learn proper workout techniques for different exercises. With the Workout Library, I can access a variety of routines and exercises organized into different categories such as muscle group or difficulty level. Each exercise comes with detailed descriptions and tips on proper form so that I can first see how the workout is properly done before attempting it myself. Having the ability to browse and select new workouts will help me try new exercises in my workout routine.

The Workout Library offers a well-organized collection of exercises so users can explore and tailor their fitness journey to their needs and goals. It serves as the first step for trying a new workout or exercise on our platform, enabling users to access instructional videos so they know how to lift with proper form and technique. The Workout Library is a core important of the application as it educates users and helps them progress in their pursuit of health and fitness.

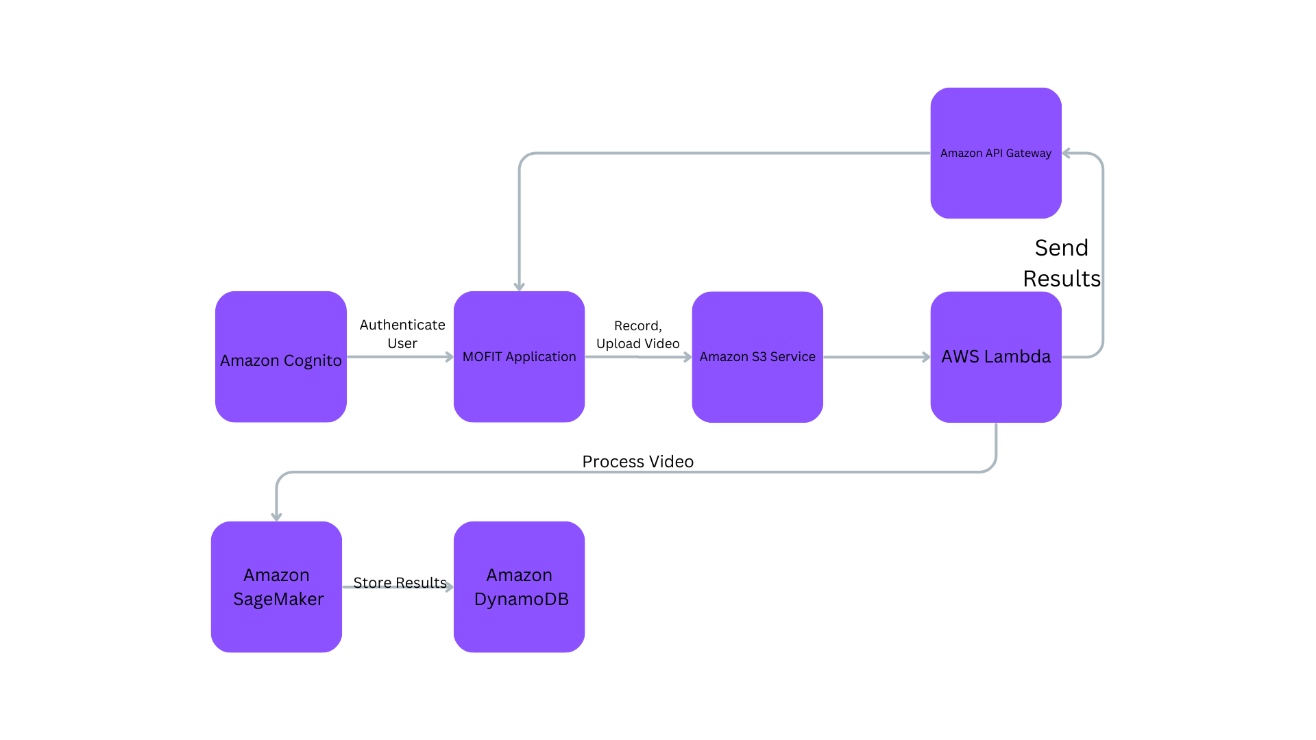
# Application Architecture

## Chart Overview

\*490 Chart



491 Chart



## Architecture and Design

### Architecture Pattern: Model-View-Controller (MVC)

Using MVC is appropriate for MOFIT because it allows for easy, organized and maintainable code because the app’s logic is separated into three components. MVC allows for modularity, so each component can be developed and tested on their own. These components will be flexible and reusable, which is helpful with the different types of workouts that plan to be implemented in the future.

#### Model

The model in MOFIT will represent the data, which includes the user profiles, exercise data, and workouts.

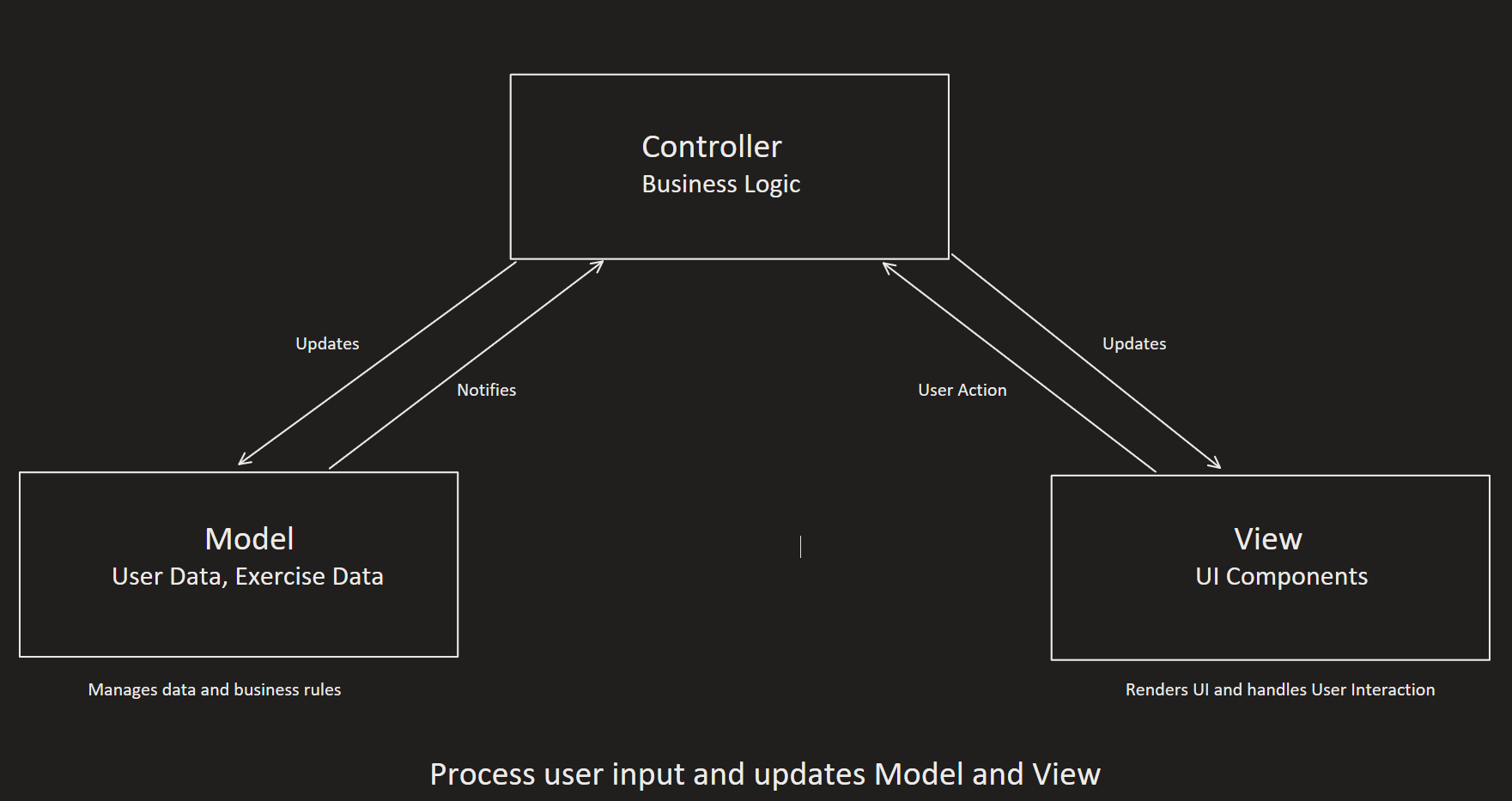
#### View

The view represents the UI components of the app. Examples of this will include the workout screen, the progress trackers, and the exercise libraries.

#### Controller

The controller will act as a liaison between the model and the view, meaning that it will tie the functions of the app together.

#### MVC Diagram



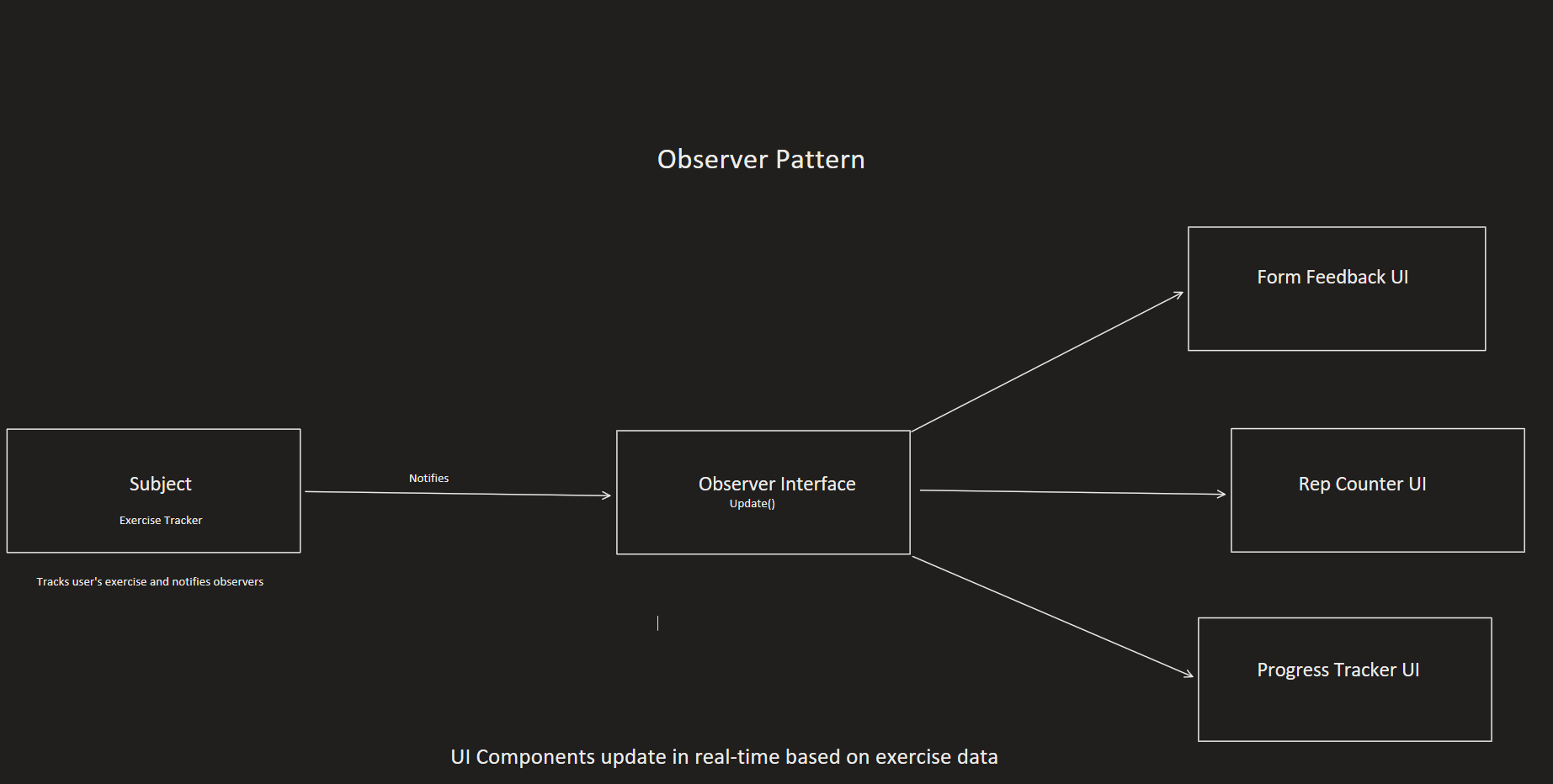
#### Scenario

1. User Accesses Profile
   * The user navigates to the profile page in the app.
   * The controller (Profile Controller) receives this action and requests the profile data from the model (UserProfile).
   * The model returns the data, and the controller instructs the view to display the data.
2. User Edits Information
   * The user decides to update their weight when they wake up in the morning.
   * They input their current weight in the View.
3. User Submits changes
   * User hits “save”.
   * The view notifies the controller of the save action and passes the data along.
4. The controller processes update
   * Controller receives update request from view.
   * Validates the data to ensure proper values are inputted.
   * If values are valid, controller calls on the model to update the user’s weight.
5. Model updates data
   * The model updates it’s internal state to the new weight.
   * May trigger backend operations such as saving the data to a database.
   * Model notifies the controller that the update is successful.
6. Controller updates View
   * When confirmation from the model is received, controller tells view to refresh.
7. View reflects changes
   * The view updates to reflect the new weight the user originally inputted.
   * It may also display to the user that the operation was a success.

### Design Pattern: Observer Pattern

MOFIT will use the observer pattern in order to have real-time updates and synchronization between different components of the app. The user needs to have immediate feedback on their workouts in order for them to save time and improve their workouts in real time. This pattern allows for loose coupling, so the design is more flexible components. It will also help with scalability so new features are easy to add.

#### Observer Diagram



##### Scenario

1. The user begins their workout with the camera pointed at themselves. The exercise tracker (subject) analyzes the user’s movements. In this scenario, the user is doing hammer curls.
2. As the user performs their hammer curl, the exercise tracker continues to update it’s state with new data including:
   1. Arm angle
   2. Movement speed
   3. Rep count
   4. Form accuracy
3. When there is a significant change, such as a completion of a rep, the exercise tracker notifies all registered observers.
4. The observer reacts by updating the proper UIs.
   1. Form Feedback UI
      1. Shows user’s current form
      2. If the user’s elbow is in the wrong position, it will give a warning to the user to correct themselves.
   2. Rep counter UI
      1. Increments the rep count each time a curl is complete.
      2. Updates the set count when the user completes a number of reps.
   3. Progress Tracker UI
      1. Records the weight used and rep count completed for each set.
      2. Updates the user’s progress chart for the given workout.
5. This process continues throughout the entire exercise, with the exercise tracker constantly updating and notifying the observers, and the UI components reacting accordingly.

## UX Design

#### Aesthetic

##### Design Philosophy

MOFIT will be designed to be minimalistic and clear to all users, with clear functionalities and drive user engagement for continual usage throughout their fitness journey. The user will always know where they are, where they can go, and what they are doing at all times, not only through visuals on their screens, but also through audio and physical feedback.

##### Color Scheme

###### Primary Colors

Primary Hex Color #15B8A6 (Mountain Meadow)

* Represents a unique, refreshing feel. Theme color, positive response.

Secondary Hex Color #15B825 (Malachite)

###### Neutral Colors

Background Hex Color #555555 (Emperor)

* Represents a sleek color that does not overpower the senses. Fits primary color well.

Blacks Hex Color #000000

White Hex Color #FFFFFF

###### Accent Colors

Incorrect Color Hex #EA4759 (Mandy)

* Opposite of the primary color, can be used to show incorrect/ wrong inputs

Warning Color Hex ##E69014 (Golden Bell)

* Needs improvement, warning, please wait.

##### Typography

###### Font Family

Primary Font – LiberationSans SDF – Outline

Secondary Font – LiberationSans SDF – Material

###### Font Sizes

\*All Font sizes represented from Unity’s inspector based on a canvas size of 1440x3088

265 – Main Menu Text size (MOFIT Title)

62 – Main Menu Subtitle size

Button Panel fonts

48, 51, 56 – Library button, Work Out!, User Profile button respectively.

###### Line Spacing

Wrapping – Enabled

Overflow – Overflow

Alignment – Middle Center

0 Line spacing

##### Interaction Design

-Button states will be the primary color by default, and when selected will show the secondary color.

-Animations will be used to show transitions between menus, represent the user within the motion tracking

-The application will adapt to different screen sizes and orientations if the user prefers it, such as when watching video playback.

#### User Research

Most users who would likely to be using this app are between the ages 18-34. This is the common age group for people who go to the gym (Spraul, Exercise.com). The main reason for this age group is because this is the age range where individuals have the most energy and motivation to better themselves.

People would want to try out MOFIT because it gives them a way to privately learn how to properly exercise. “People who view themselves as competent exercisers are more likely to stick to a regular routine, while people who think the opposite may struggle to find consistent motivation” (Stefanie Williams, Time.com). If users see themselves as “competent”, they have a higher change to stick to their healthy lifestyle, and MOFIT aims to improve competency. Even older individuals who seek a healthier living see the importance of having a safe, productive method of exercise. While most prefer aerobic, or stretching exercises, strength training remains a solid type of exercise for those who seek it. According to health.harvard.edu, “exercise helps to control appetite, boost mood, and improve sleep. In the long term, it reduces the risk of heart disease, stroke, diabetes, dementia, depression, and many cancers”. With this many reasons to exercise regularly, its no wonder the industry is so large.  
  
With an industry so large and profitable, there are many bad actors giving out incorrect or false information. Younger individuals are likely to find their exercise methodology through these influencers. These influencer’s main goal is to hook in users to visually appealing exercises that appear to be a shortcut to a better body. In most cases these workouts are just for the video and are not actually in that influencer’s routine. MOFIT does not care about getting clicks and impressions, it is just a device to deliver accurate methods on effective training.

#### Personas

##### Persona 1: Steven Hill

Age: 20

Occupation: Computer Science Student

Fitness Level: Beginner

###### Background

Steven is a sophomore majoring in computer science. Because of his major, he spends a lot of time sitting down and focusing on their work rather than their body. Over time he began to feel unhealthy, and wanted to improve their health through working out, but with no knowledge on the subject, he had no clue on where to begin.

###### Goals

Steven had 3 goals in mind:

* Build a consistent workout habit that would span over many years to maintain a healthy living.
* Improve their physical look by gaining muscle and also improve their energy level so they can have energy to do both his schoolwork and fitness work.
* Learn how to properly exercise in key movements to avoid injury.

###### Challenges

Steven had a few challenges he needed to face if he were to endeavor into this new lifestyle.

* With his academic commitments, such as going to class, doing homework, working on projects, and looking for internships, he is under a heavy time constraint.
* Has never been in a gym before, so the environment feels intimidating and foreign to him.
* With no guides in his life for this kind of lifestyle, he as no knowledge on what exercises he should do, how often to do them, and how to properly perform them.

###### Tech Usage

Because of the field he is pursuing, he is very proficient in technology and can use it to his advantage.

* Has the latest models of smartphones and computers, so tech limitations are not a factor
* Already has multiple routine apps to keep him productive during the day.
* Is able to easily pick up new apps and learns on the fly.

###### How MOFIT Can Help

Factoring everything that Steven does, MOFIT can help him reach his goals by providing a base on where to begin. MOFIT will give Steven basic but powerful movements that he can master on his own. Since the app only relies on the user, he can fit his exercises into any part of his busy day without having to work into a trainer/friend’s schedule. MOFIT can also feel like a kind of game, where Steven strives to get the best possible results on each of his movements by reaching an optimal rep, set and time range.

##### Persona 2: Maggie Thompson

Age: 39

Occupation: Office Supervisor

Fitness Level: Intermediate

###### Background

Maggie is an office supervisor and a mother of 2. In her youth, she frequently exercised, but had to abandon it due to other priorities such as raising her children and climbing up the corporate ladder. She is looking to get back into a healthy lifestyle that also fits into her busy days as a important job figure and a mother.

###### Goals

Maggie had 3 goals in mind:

* Aim to a fitness level similar to hers before having children.
* Find other options for working out that can be done in a short amount of time, but also be effective.
* Set an example for her children so they can also have a healthy lifestyle while growing up.

###### Challenges

Maggie has a few challenges she needs to face if she wants to go back to her healthy lifestyle.

* Limited time due to her work and family responsibilities.
* Difficulty getting back into the old routine due to old workouts taking a longer time out of her day.
* Need flexibility and low stress in her workout schedule.

###### Tech Usage

Since Maggie has an office job, she is proficient in most basic technologies.

* Comfortable with smartphones and basic apps.
* Uses calendar apps and organization apps for work.
* Uses social media daily.

###### How MOFIT Can Help

MOFIT gives easy, practical exercises that can be done at any time at home or at the gym. It will give immediate feedback on if Maggie is properly doing her exercises. With its ability to track progress, it will motivate her to keep pursuing her goals.

##### Persona 3: Robert Irvine

Age: 63

Occupation: Retired Financial Advisor

Fitness Level: Beginner-Intermediate

###### Background

Robert recently retired and wants to fill his time staying active. Throughout his life, he has always been healthy, but never had a structured routine to maintain his health. Since he has more free time, he wants to improve his health through exercise and wants to do it in the cheapest way he can.

###### Goals

Robert had 3 goals in mind:

* Improve his strength and flexibility
* Maintain his cognitive health through physical activity
* Stay active throughout his retirement.

###### Challenges

Robert had a few challenges he needed to face due to his aging.

* He as some joint pains and is not able to move as fast as he used to.
* He is cautious about injury; he knows at his age he is more prone to it.
* He is less tech savvy, so modern approaches to fitness may come as a challenge.

###### Tech Usage

Robert’s age means that eventually technology got ahead of him, so his knowledgebase is smaller than people younger than him.

* Uses a smartphone, but only it’s basic functions such as calling, texting, emailing.
* Needs more guidance for newer, more unique apps.

###### How MOFIT Can Help

MOFIT aims to be usable for all age groups, with simple instruction that does not need to be repeated over and over. If Robert takes the time to learn the basics of the app, he will not feel alienated over time. The app will tell Robert if he is doing a workout wrong, so the risk of injury is lower than him doing things by himself. The library within the app will also tell the benefits to each workout that it provides, so that he knows that what he is doing is improving his overall health.

#### UX Framework

##### BASICUX Framework

###### What is BASICUX?

BASIC consists of 5 components: Beauty, Accessibility, Simplicity, Intuitiveness, and Consistency. Following these 5 components will result in a usable product for any user who tries MOFIT.

###### Beauty

MOFIT’s design will be consistent with the aesthetics previously outlined to provide. Whenever a design is being created, questions to ask the developer would include:  
Is it pleasing to the eye?

Does it follow all set guidelines?

Are the visuals of high quality?

Are all aspects in proper alignment?

###### Accessibility

MOFIT is designed for users who seek a healthy lifestyle and will reach out to people of all shapes and sizes. Whether it be to lose weight, gain muscle, or a mixture of both, all users should be able to access the functions of the application, as long as they are able to participate in the exercises provided by the application. As a developer, questions to ask would include:

Does it comply with standards?

Does it operate on all android devices?

###### Simplicity

MOFIT must have easy-to-use systems that don’t require extra reading. No excessive words, options, or complexities with graphics. The user should not have to think about their inputs and should come naturally to them at first or soon after picking up the application. The goal is to make exercise easier and welcoming. Questions to keep in mind include:  
Is there clutter and repetitiveness within the user experience?

Is the function necessary?

Does it reduce the user’s workload?

###### Intuitiveness

MOFIT should be able to be picked up by all age groups ages 13 and up. Once the user goes through the steps of the application, they should not need reminders on how to operate a certain feature. When learning features, it should come to the user easily. The application should run to them like any other application they may frequently use so they feel at home when using MOFIT. Questions to keep in mind:

Is the functionality clear?

Can the user complete their goals with little to no guidance?

Can the user predict outcomes?

###### Consistency

MOFIT’s user should always come back to what they expect. If any new features are created, they should follow previous UX design principles. If the user had their eyes closed and tried to navigate through the application, they should be where they expect if they previously knew the pattern of the application. The application should also be dependable, with little to no outages, and also be speedy and not have varied wait times on the same feature. Questions to keep in mind:

Are patterns/Designs be reused?

Does it perform consistently?

#### UX Architecture and Design

###### Overview

The architecture of MOFIT will provide a user with an easy-to-learn experience which they will be able to navigate through the features within the app with ease. The design will be modular in order for further updates to be easily modified and expanded upon.

###### High-Level Information Architecture

Home dashboard- Welcome screen with all available navigation

Workout Session – Main feature of the app, where the user records their workout

Exercise Library – Shows what workouts are available to do and learn more about

Progress Tracker – shows the data of the user’s workouts, showing their improvements and possible regressions

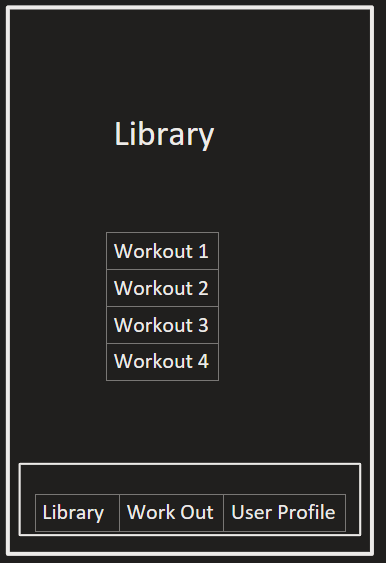
Settings and User Profile – Change functions of the app for better user experience and input their information.

###### Navigation Model

The app will be designed with two major ways of navigation- a persistent navigation bar at the bottom of the screen, and a tabular system that switches to different functions of the app. This should have a feeling of familiarity with how most current phone apps operate, as well as the OS systems of the devices.

#### UX Mockup

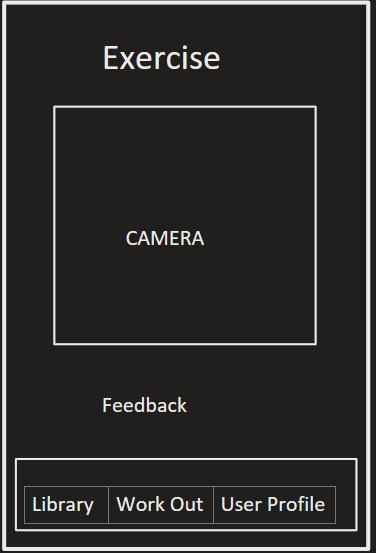
The main menu will have the navigation bar, where the user can go to 3 options, the library, work out, or user profile. Tapping or clicking on any of these options will bring them to a new screen.

The library will have a list of available workouts for the user to choose from. They will be able to interact with any of them, or decide to navigate to another page via the bottom navigation bar.

A screenshot of a black screen

Description automatically generatedThe user profile will also retain the navigation bar at the bottom, and will show the user’s information, which they could edit at any time.

The workout menu will have an interactable button to begin the camera. It could also provide information on how it works. The navigation bar remains at the bottom.

The main component of MOFIT, where data is gathered from the camera and feedback is given to the user during and after the workout at the bottom of the screen. The navigation bar also remains.

## Mobile Development

### UI Design (Android)

Mockup UI on FIGMA: <https://www.figma.com/file/tRiBGJ28KQUbgvGU8DJfZz/MOFIT-UI?type=design&node-id=0%3A1&mode=design&t=SqzxLVBhm5XRQZyo-1>

#### Layout

Upon opening the app, users will be directed to the dashboard home page that includes an ‘Exercise Library’ section, where they can browse and select a variety of exercises categorized by muscle groups or workout type. The home screen will also feature access to other key sections such as ‘Workout History’ and ‘Settings’. The app’s color scheme and typography will be chosen to enhance readability and ensure accessibility for all types of users. The layout ensures that users can navigate the app effortlessly, find information they need quickly, and start workouts promptly.

#### Workout Interaction

The core feature of our application will be Form Check which is designed to provide users with a seamless and engaging workout experience. Upon selecting an exercise from the library, users are guided to a recording screen where they can see themselves through the phone’s camera. Clear instructions and visual cues are displayed to ensure proper phone positioning and optimal recording. A ‘Start Recording’ button will initiate the capture of the workout session and users will be able to see on screen when exactly their device is capturing their movements. After completing a set, the app will process the video and offer a summary of performance, highlighting areas of improvement and acknowledging satisfactory points.

### Platforms

#### Android Integration

For the Android version of our application, I will utilize Unity’s asset plugin OpenCV to allow for tracking the user’s body. This plugin allows for accurate tracking to enable YoloV8 to analyze the user’s key body parts, and recognize which movement they are performing.

#### Other Platforms

Since development will be done on Unity, the option to integrate other platforms such as IOS, it is possible to have the application on other devices. Only small changes in the code will have to be made in order to recognize which device the user is on.

## Machine Learning

### Data Processing

#### Motion Analysis

Motion analysis is a key aspect of data collection and processing for our application. The app utilizes the phone’s camera to capture the user’s movements during workouts and then analyzes this motion data using machine learning algorithms. Computer vision techniques, such as object detection and tracking, will be employed to analyze the user’s movements via OpenCV and compare them with the optimal form for each exercise. This analysis allows MOFIT to provide real-time feedback and suggestions for improvement, ensuring users can perform exercises safely and effectively. By breaking down each exercise into its fundamental components, MOFIT can assess the user’s form, range of motion, and consistency. This analysis is crucial for identifying deviations from optimal form and providing targeted feedback for correction.

#### Feature Extraction

Feature extraction will be an essential step in the data preparation and analysis process. This involves processing the video footage captured by the user’s phone to identify key characteristics of their movements, such as joint positions, angles, and range of motion. These features are carefully selected to capture the most important aspects of exercise form and technique. By extracting these features, the application can reduce the complexity of the raw video data and focus on the most relevant information for assessing a user’s exercise. These extracted features will serve as the input to the machine learning models, enabling them to make accurate predictions and provide relevant feedback on how to improve the user’s exercise form.

### Model Development

#### Android Architecture

For Android devices, the architecture will incorporate OpenCV tools to handle video capture and feature extraction. Using these tools for image analysis, enabling the app to efficiently process the workout footage captured by the phone camera. The extracted features, like points of motion, will be inputted into the YOLOv8 model. YOLOv8 will then perform the analysis of the user’s exercise form and generate feedback.

### Model Deployment

#### On-Device Inference

On-device inference is a key aspect of model deployment. This approach allows machine learning models, such as YOLOv8, to run directly on the user’s mobile device, enabling real-time analysis and feedback during workouts. On-device inference ensures low latency, as there is no need to send video data to a remote server for processing. It also enhances privacy, as sensitive user data remains on the device. The MOFIT app will be optimized to manage computational resources efficiently, ensuring that the models run smoothly without draining the device’s battery or impacting performance.

#### Model Updates

To maintain the accuracy and relevance of the machine learning models in MOFIT, regular model updates are essential. The app will employ a dynamic updating mechanism that allows new versions of the models to be seamlessly integrated without disrupting user experience. These updates can include improvements in the algorithms, adaptations to new exercise trends, or enhancements based on user feedback. By keeping models up to date, MOFIT ensures that users continue to receive accurate and effective feedback on their exercise form.

## Cloud Computing

According to Microsoft cloud computing is the delivery of services over the internet, referred to as ‘the cloud’. These services include but are not limited to, networking, storage, and databases, allowing for faster innovation, easier scalability, and flexibility of resources. Cloud services are normally paid for and reduce operating costs by allowing businesses to offset software/hardware maintenance and costs.

### Cloud Services Integration

#### Virtual Machines

By using virtual machines (VMs) hosted on cloud platforms such as AWS for GCP, the application can support backend operations and other computational tasks that won’t be performed on-device. While video and data analysis occurs on the user’s device to ensure privacy, VMs can handle storage, and user data management, and help serve as a repository for model updates and application updates.

#### Storage and Networking

Cloud storage is utilized to securely store user profiles, workout videos, and historical workout data. This ensures users can access their data from any device & makes data recovery easy if a user’s phone breaks or is lost. To manage the communication between the mobile app and the cloud, the use of content delivery networks (CDNs) will help speed up connections and data sent to and from the device to the cloud.

#### Database Services

By using managed database services such as Amazon RDS, we don’t have to worry about the complexities of database maintenance, scalability, and security. Services like RDS and Cloud SQL provide automatic backups, recovery, and scaling which helps reduce the development time.

### Regulatory Compliance

#### Data Encryption

Given that MOFIT will be dealing with sensitive health and workout data, data encryption will be a critical aspect to ensure data security and privacy. All data transmitted between the user's device and the cloud will be encrypted using TLS to create a secure channel and data stored in the database will use AES encryption to protect the stored data from unauthorized access.

#### Compliance Standards

We intend our application to be compliant with HIPAA in the United States and GDPR in the EU given the health-related data our application will be dealing with. By complying with HIPAA, MOFIT will ensure all health data is handled properly and with confidentiality. Additionally, we intend MOFIT to be compliant with GDPR for users in the EU. GDPR imposes rules on data processing and what individual rights users have with their data handled on servers, ensuring user trust.

# Scope

MOFIT will be able to track key body parts to interpret the user’s movements, mainly focusing on key joints to use as data. The minimum we plan on implementing is a full analysis of a hammer curl movement, which will have a focus on the arm path of the user. I will not have enough time to add a full library of exercises that cover every muscle group, but if time provides, the goal is to have at minimum four separate exercises that cover main muscle groups, such as lateral pulldowns for the back, a squat for the legs, and a pressing movement for the shoulders.

The architecture of MOFIT will go as planned, with a focus on a mobile app that utilizes machine learning to analyze and extract the information, which will be sent to the cloud for backend operations. Features that may be left out include the leaderboard and the workout library, as they are not vital to the operation of the app, but more additional features that would widen the scope of the project.

# Feasibility

This project and its features will show mastery of key computer science aspects for future careers. Many aspects require hours of research on implementation, and even more hours integrating them into each other for smooth operation. I have learned a lot of fields in computer science and have already begun prototypes of the program, showing that with time, this project will be able to be completed before our 491 class’s semester’s end. We will also have the summer to research, learn, experiment, coordinate, and collaborate to have the smoothest possible development cycle.

# Activities

|  |  |  |
| --- | --- | --- |
| **TODOs (490)** | **Description (490)** | **Date Scheduled to Start (490)** |
| FormCheck | Create FormCheck application needed as part of our app; Main use - checks users’ forms through a captured video | May 17, 2024 |
| Database Creation | Implement a secure database to store all user info and other necessary information | May 27, 2024 |
| Cloud Service Integration | Integrate Cloud Services for increased data storage and security capabilities | Jun 1, 2024 |
| Working Skeleton | Implement the bare minimum running app as a base to build the UI off of | Jun 1, 2024 |
| Frontend Design | Build a cohesive UI that allows users to intuitively navigate for easier use (Focus on color details and UI flow to enhance performance/user reception) | Jun 15, 2024 |
| Frontend Features | Implement all promised features in frontend -integrate backend applications as needed | Jun 30, 2024 |
| Backend Testing | Intensively test all backend functions to ensure they work correctly | Jul 5, 2024 |
| Frontend testing | After integrating the backend, do full app testing to see if the frontend is performing as it should be; backend functions are properly integrated and can be used | Jul 10, 2024 |
| **TODOs (491)** | **Description (491)** | **Date Scheduled to Start**  **(491)** |
| Finalize Project requirements | Update project to 491 requirements for check-ins and drafts. | Sep 9, 2024 |
| Set up Unity Development Environment | Ensure all requirements and dependencies are functional to begin development | Sep 16, 2024 |
| Begin UX, UI Design | Create a friendly user experience that is easy to follow | Sep 23, 2024 |
| Set up Cloud Services | Create an AWS cloud service to handle backend | Sep 30, 2024 |
| User Profile | Users can create their personal profile with their information | Oct 7, 2024 |
| Machine learning model for motion analysis | Create models to analyze what workouts are being done | Oct 14, 2024 |
| Progress Tracking | Users can track their workouts and store them in their user profiles. | Oct 21, 2024 |
| Midterm Presentation | Demonstration and PowerPoint showing my progress for the project | Oct 21, 2024 |
| Workout Library | 3-4 Exercises to be put into the library that users can learn more about and try | Oct 28, 2024 |
| Form Check | User can compare their workouts with what is optimal | Nov 4, 2024 |
| Backend Testing | Intensively test all backend functions to ensure they work correctly | Nov 11, 2024 |
| Frontend Testing | After integrating the backend, do full app testing to see if the frontend is performing as it should be; backend functions are properly integrated and can be used | Nov 18, 2024 |
| Complete additional functionalities | Update and finish any features that need more work for a better experience | Nov 25, 2024 |
| Polish and Documentation | Finalize all features in MVP and update documentation for users | Dec 2, 2024 |
| Final Project Presentation work | Create demonstration, powerpoint | Dec 9, 2024 |

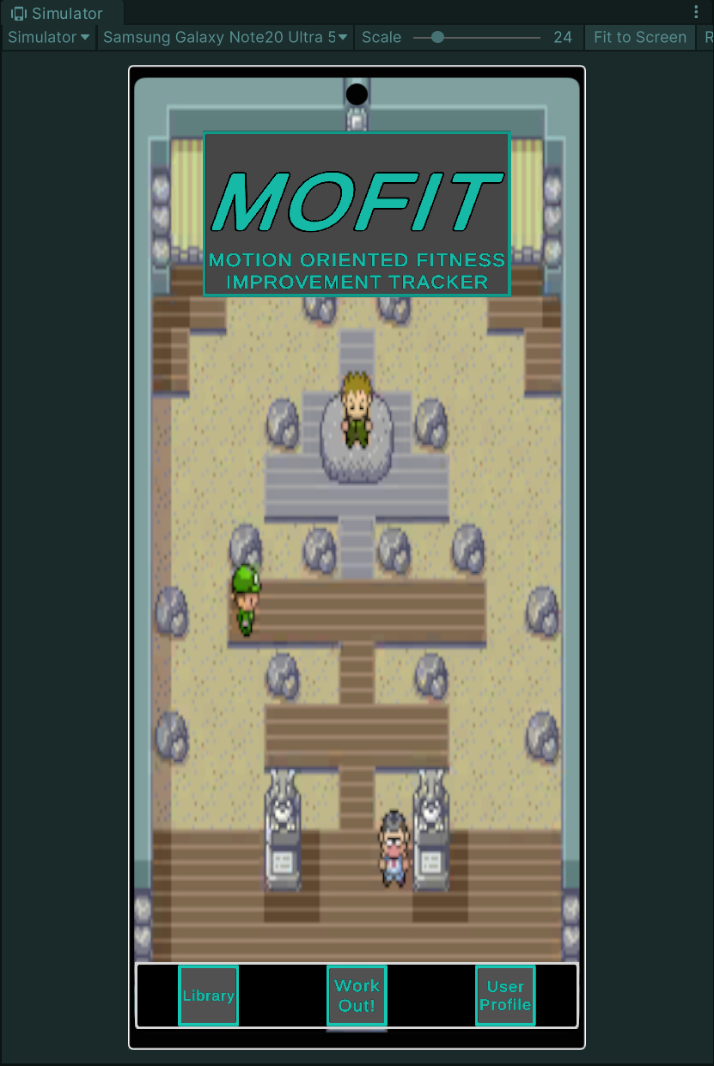
# Risk & Mitigations

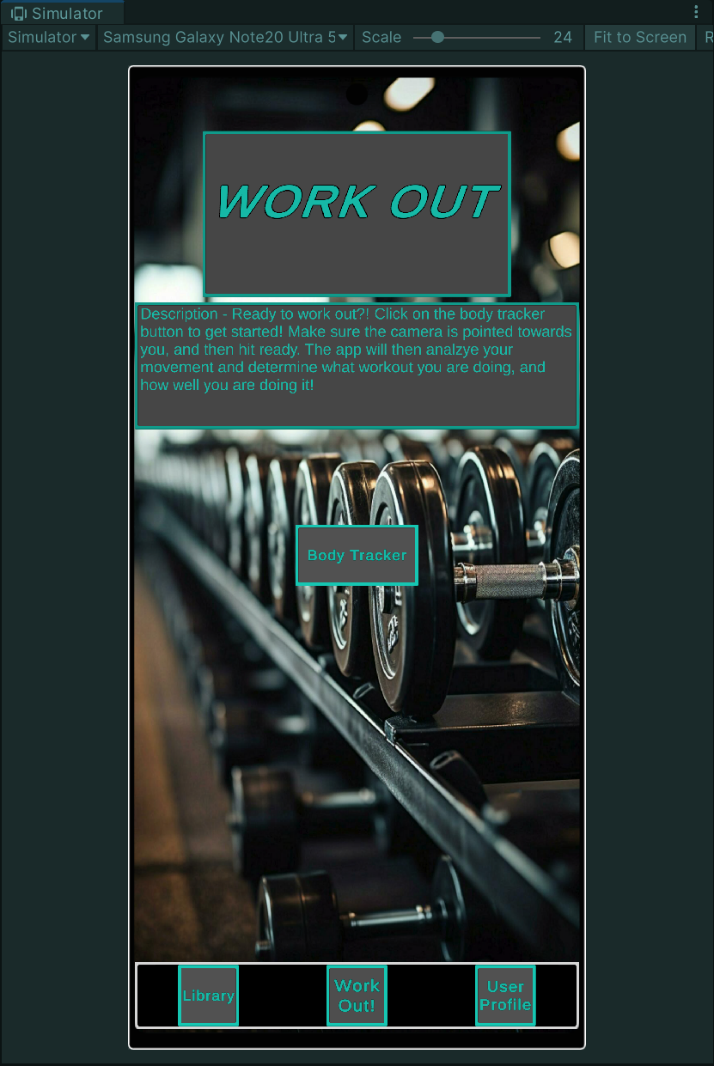
|  |  |  |
| --- | --- | --- |
| **Risk** | **Severity** | **Mitigation** |
| Creating a working app for Android and IOS platforms | Low | 1. Develop only for Android because of no available Apple devices I can use. 2. Release for Android platform in initial update and continue to adapt for IOS platforms in a later update |
| Overpromising on the variety of workouts that can be accurately tracked and corrected | Moderate | 1. Focus on 3-4 core workouts and get them fully functioning 2. Clarify with users, app capabilities 3. Continuously work on and update with new features after ensuring reliability |
| User Engagement | Low | 1. Create an intuitive, user-friendly UI 2. Ask for user feedback regarding the AI trainer and incorporate suggestions into updates |
| Privacy Concerns: regarding user data such as recorded workouts, health/personal info, etc. | High | 1. Explicit user consent before being able to use 2. Strict privacy policies and implementations adhering to current regulations 3. Ensure data is encrypted and stored securely |
| Exoskeleton Tracking Accuracy | Moderate | 1. Conduct extensive testing in various conditions to ensure accuracy 2. Implement control algorithms that can account for inaccuracies |
| Legal Regulatory Compliances | High | 1. Do extensive research on relevant laws and regulations and continuously stay updated to ensure the app is always in compliance with current statutes |
| Risk of Injury | Low | 1. Collaborate with fitness experts to develop accurate algorithms that could give correct advice 2. Include disclaimers and warnings to let users know of risks before use |

# Release Plan / Future Support

|  |  |  |
| --- | --- | --- |
| **Version** | **Plan** | **Feature Notes** |
| Version 1.0 (Initial Release) | Minimum Viable Product (Soft launch) Release | * User Profile Creation * Motion Tracking * Real-time feedback * Workout Library * Progress Tracker * Cloud service integration |
| Version 1.1 | Feature Enhancement Release | * Form Check * Improved machine learning models * Bug fixes and performance improvements |
| Version 1.2 | Social Engagement Release | * Leaderboard integration * Social sharing * Community forums |
| Version 1.3 | Accessibility/Usability Release | * Accessibility Improvements * Usability enhancements |
| Version 2.0 | Major Update Release | * New advanced workout suggestions * Customized workout plans * Compatibility with wearable fitness trackers |

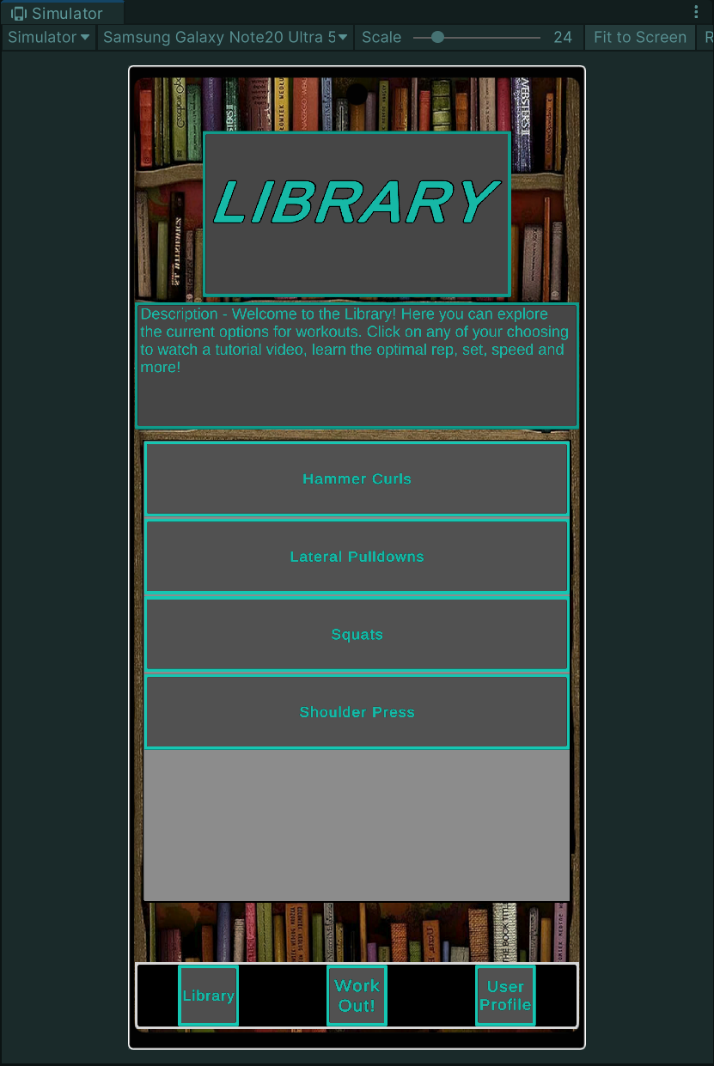
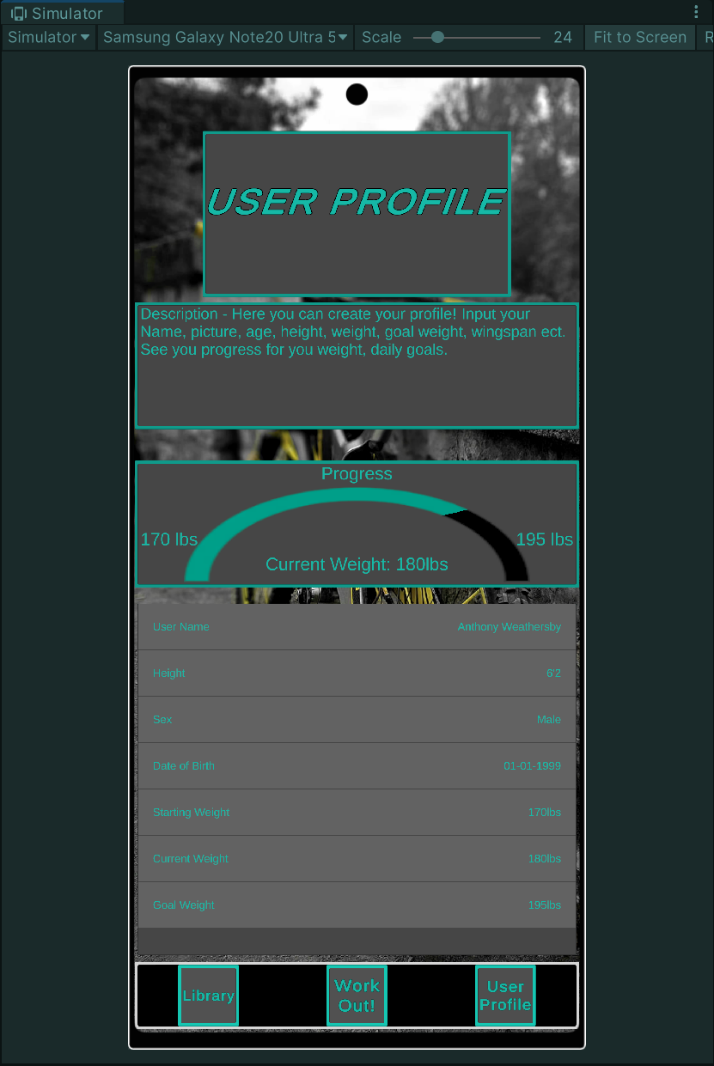
# Prototype

* The main menu, this is what the app opens up to, showing the name of the app at the top and the navigation options at the bottom, being the option to go to the library, work out, or your user profile. Since this is a android application, the simulator within the Unity editor is using one of the latest models of android. The application is working on both Windows 10 and Android 14.  
    
  The navigation bar remains at the bottom, so the user is able to go to wherever they need to at any time.

A screenshot of a person with a beard

Description automatically generated

Upon tapping the work out button, they are taken to the work out page, where it tells you what to do before starting up your camera to begin your workout. Upon Clicking the Body Tracker button in the center (to be enlarged/changed for better viewing), the scene will change to the camera.  
The camera will start and begin tracking the user’s key body parts. In the screenshot above, it is able to recognize me by showing a red border around me. Blue lines are showing that it is tracking my arms, shoulders, and torso. The green is showing that it is tracking my facial features, such as my eyes, nose and ears. Currently this is a proof of concept, with no machine learning developed for specific workouts.



The user profile will hold all of the user’s information so they can quickly glance or change information. It is also showing the user’s progress for their weight but will also show progress on specific workouts in the future.

The library shows what workouts will be available to the user. It is expandible but with time constraints, these may be the only ones functional in the best case. Worse case would be just hammer curls. Upon clicking on a workout, it will show what the description is saying, but currently is not implemented at the time of this prototype.

The code below showcases how the navigation of the menus operates. The MenuManager class is the has a few key features that operate the core of the app.  
1. Dynamic Button Management – flexible and scalable menu configurations

2. Scene Transition Integration – works with another class to provide animated transitions between scenes

3. Input Position-Based Transitions – The direction of the scene transition depends on where the user interacts.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

[System.Serializable]

public class MenuButton

{

    public string buttonName;

    public CrossPlatformInputManager buttonInput;

    public string targetSceneName;

}

public class MenuManager : MonoBehaviour

{

    [SerializeField] private List<MenuButton> menuButtons = new List<MenuButton>();

    private SceneTransitionManager transitionManager;

    private void Start()

    {

        transitionManager = FindObjectOfType<SceneTransitionManager>();

        if (transitionManager == null)

        {

            Debug.LogWarning("SceneTransitionManager not found in the scene. Scene transitions may not work.");

        }

        SetupButtons();

    }

    private void SetupButtons()

    {

        foreach (var button in menuButtons)

        {

            if (button.buttonInput != null)

            {

                button.buttonInput.OnPress.AddListener((pos) => LoadSceneWithTransition(button.targetSceneName, pos));

            }

            else

            {

                Debug.LogWarning($"Button input for {button.buttonName} is not assigned.");

            }

        }

    }

    private void LoadSceneWithTransition(string sceneName, Vector2 inputPosition)

    {

        if (transitionManager != null)

        {

            SceneTransitionManager.TransitionDirection direction = DetermineTransitionDirection(inputPosition);

            transitionManager.TransitionToScene(sceneName, direction);

        }

        else

        {

            Debug.LogWarning("SceneTransitionManager not found. Loading scene without transition.");

            SceneManager.LoadScene(sceneName);

        }

    }

    private SceneTransitionManager.TransitionDirection DetermineTransitionDirection(Vector2 inputPosition)

    {

        if (inputPosition.x < Screen.width / 3)

        {

            return SceneTransitionManager.TransitionDirection.RightToLeft;

        }

        else if (inputPosition.x > Screen.width \* 2 / 3)

        {

            return SceneTransitionManager.TransitionDirection.LeftToRight;

        }

        else

        {

            return SceneTransitionManager.TransitionDirection.BottomToTop;

        }

    }

    // Optional: Method to add a button at runtime

    public void AddButton(string buttonName, CrossPlatformInputManager buttonInput, string targetSceneName)

    {

        MenuButton newButton = new MenuButton

        {

            buttonName = buttonName,

            buttonInput = buttonInput,

            targetSceneName = targetSceneName

        };

        menuButtons.Add(newButton);

        if (buttonInput != null)

        {

            buttonInput.OnPress.AddListener((pos) => LoadSceneWithTransition(targetSceneName, pos));

        }

    }

}

The code below showcases the class SceneTransitionManager, which is responsible for the animated transitions between scenes for an enhanced user experience. Key features within this code include:

1.Customizable transitions – Currently 3 transitions with the ability to add more (right-to-left, left-to-right, bottom-to-top)

2. Flexible animations – With the use of LeanTween, many types of animations and transitions can be displayed.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class SceneTransitionManager : MonoBehaviour

{

    [SerializeField] private RectTransform transitionImage;

    [SerializeField] private float transitionTime = 0.5f;

    public enum TransitionDirection

    {

        RightToLeft,

        LeftToRight,

        BottomToTop

    }

    private void Awake()

    {

        if (transitionImage == null)

        {

            Debug.LogError("Transition image is not assigned in the inspector.");

        }

    }

    public void TransitionToScene(string sceneName, TransitionDirection direction)

    {

        if (transitionImage == null)

        {

            Debug.LogError("Transition image is missing. Loading scene without transition.");

            SceneManager.LoadScene(sceneName);

            return;

        }

        StartCoroutine(PerformTransition(sceneName, direction));

    }

    private IEnumerator PerformTransition(string sceneName, TransitionDirection direction)

    {

        SetInitialPosition(direction);

        AnimateCover(direction);

        yield return new WaitForSeconds(transitionTime);

        SceneManager.LoadScene(sceneName);

        // Wait for the next frame to ensure the new scene is loaded

        yield return null;

        // Find the transition image in the new scene

        Canvas canvas = FindObjectOfType<Canvas>();

        if (canvas != null)

        {

            transitionImage = canvas.transform.Find("TransitionImage") as RectTransform;

            if (transitionImage == null)

            {

                Debug.LogError("TransitionImage not found in the new scene.");

                yield break;

            }

        }

        else

        {

            Debug.LogError("Canvas not found in the new scene.");

            yield break;

        }

        AnimateUncover(direction);

        yield return new WaitForSeconds(transitionTime);

        ResetPosition(direction);

    }

    private void SetInitialPosition(TransitionDirection direction)

    {

        switch (direction)

        {

            case TransitionDirection.RightToLeft:

                transitionImage.anchoredPosition = new Vector2(Screen.width, 0);

                break;

            case TransitionDirection.LeftToRight:

                transitionImage.anchoredPosition = new Vector2(-Screen.width, 0);

                break;

            case TransitionDirection.BottomToTop:

                transitionImage.anchoredPosition = new Vector2(0, -Screen.height);

                break;

        }

    }

    private void AnimateCover(TransitionDirection direction)

    {

        switch (direction)

        {

            case TransitionDirection.RightToLeft:

            case TransitionDirection.LeftToRight:

                LeanTween.moveX(transitionImage, 0, transitionTime).setEase(LeanTweenType.easeInOutQuad);

                break;

            case TransitionDirection.BottomToTop:

                LeanTween.moveY(transitionImage, 0, transitionTime).setEase(LeanTweenType.easeInOutQuad);

                break;

        }

    }

    private void AnimateUncover(TransitionDirection direction)

    {

        switch (direction)

        {

            case TransitionDirection.RightToLeft:

                LeanTween.moveX(transitionImage, -Screen.width, transitionTime).setEase(LeanTweenType.easeInOutQuad);

                break;

            case TransitionDirection.LeftToRight:

                LeanTween.moveX(transitionImage, Screen.width, transitionTime).setEase(LeanTweenType.easeInOutQuad);

                break;

            case TransitionDirection.BottomToTop:

                LeanTween.moveY(transitionImage, Screen.height, transitionTime).setEase(LeanTweenType.easeInOutQuad);

                break;

        }

    }

    private void ResetPosition(TransitionDirection direction)

    {

        switch (direction)

        {

            case TransitionDirection.RightToLeft:

                transitionImage.anchoredPosition = new Vector2(Screen.width, 0);

                break;

            case TransitionDirection.LeftToRight:

                transitionImage.anchoredPosition = new Vector2(-Screen.width, 0);

                break;

            case TransitionDirection.BottomToTop:

                transitionImage.anchoredPosition = new Vector2(0, -Screen.height);

                break;

        }

    }

}

The class CrossPlatFormInputManager allows for multiple types of devices to be able to use the application. Currently because it is being developed on Windows, it needs to recognize mouse clicks, but when testing on the primary device of an Android phone, it needs to also recognize screen taps.

using UnityEngine;

using UnityEngine.Events;

using UnityEngine.EventSystems;

public class CrossPlatformInputManager : MonoBehaviour, IPointerDownHandler, IPointerUpHandler

{

    [System.Serializable]

    public class Vector2Event : UnityEvent<Vector2> { }

    public Vector2Event OnPress;

    public UnityEvent OnRelease;

    private bool isPressed = false;

    public void OnPointerDown(PointerEventData eventData)

    {

        HandlePress(eventData.position);

    }

    public void OnPointerUp(PointerEventData eventData)

    {

        HandleRelease();

    }

    private void HandlePress(Vector2 position)

    {

        if (!isPressed)

        {

            isPressed = true;

            OnPress.Invoke(position);

        }

    }

    private void HandleRelease()

    {

        if (isPressed)

        {

            isPressed = false;

            OnRelease.Invoke();

        }

    }

    private void OnDisable()

    {

        OnPress.RemoveAllListeners();

        OnRelease.RemoveAllListeners();

    }

}

# Installation

## Requirements

* Android Device with minimum Android version 5.1 'Lollipop' (API 22) or above and a working camera.
* Cal State Fullerton Email
* Working email account
* Internet access

## How to Install

* Navigate to <https://drive.google.com/drive/folders/1hkqFAF-mzzoi-lRtPsPuEzRypcrEwFJa?usp=sharing>
* Only Cal State Fullerton Email addresses currently have access to the drive link
* Download the Latest APK file available
* Install on your local device

## Setup

In order to use the MOFIT application, the user is required to create an account through the app

* Launch the application after installation
* In the welcome screen, tap "Log in / Sign up"
* This will take you to your web browser, where you will select "Sign Up" at the bottom of the page
* In order to successfully create a profile, the user must have a working email address and a password with the following:
  + Password must contain a lower-case letter
  + Password must contain an upper-case letter
  + Password must contain a number
  + Password must contain at least 8 characters
  + Password must contain a special character or space
  + Password must not contain a leading or trailing space
* Once all requirements are met, the user can select "Sign up"
* The user will then be sent to a page telling them that in order to confirm the account email, a verification number will be sent to that email address.
* The user finds their verification number in their email, copies it, and enters it in the verification page
* The user will then be successfully authenticated and will be sent to a new page telling them so. This page should redirect them back to the app, but in the case that it does not, they can hit "Return to App" to manually return.

## Navigation

* The user should navigate to the User profile by selecting the button at the bottom of their screen "User Profile"
* The user should fill out their Name, Height (in feet and inches), Sex, Date of Birth (Day, Month, Year), Starting Weight, Current Weight, Goal Weight

The user can then navigate to any desired part of the app, whether it be the library, or work out.

# Service Level Documentation

## Overview

These are the outlines for the Service Level Objectives, Service Level Agreements, and Service Level Indicators for MOFIT. Having these metrics for the application is essential to the success of the application, fulfilling the promise of a high-quality service and user satisfaction.

## Service Level Objectives (SLOs)

#### Availability

The target of MOFIT is to have 95% uptime monthly, because of the importance of a user tracking their workout data when trying to progress in their fitness journey. It also needs be up during peak hours.

#### Performance

The goal of the motion tracking is to accurately record only the user’s movements with a speedy processing time. The database should always be available in order for user logins and workout history recordings.

## Service Level Agreements (SLAs)

### Core Application Availability

MOFIT will be available 98% of the time, based on the servers of the Amazon Web Services and scheduled downtime for maintenance and updates. Maintenance should only be performed at hours when the userbase is at its lowest.

### Real-Time Processing Performance

MOFIT’s motion tracking technology analyses the user’s movements within milliseconds of the video frame capture. Feedback provided by the app to the user is also given within milliseconds of a complete movement of a rep, and all of the set information is provided immediately after the user’s workouts. User data is synchronized across the user’s device within 5 seconds.

### Data Protection

MOFIT’s user data is all stored within AWS and is secured by the protected given by the service. All user data is encrypted and requires proper authentication to see a user’s data.

## Service Level Indicators (SLIs)

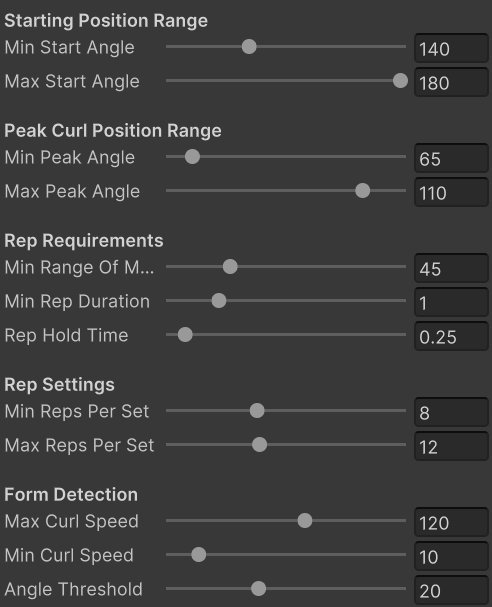
### Performance Metrics

Motion Tracking latency is an issue with the current versions of androids. Even the latest versions of the android version of the application results in low-framerate camera feedback. Despite the low camera framerate, the application can still accurately record the user’s movements. The metrics show that the average framerate is around 3 frames per second. The databases used in MOFIT have low latency and is not noticeable by the user, even if they refresh their pages.

### User Experience Metrics

#### Form Analysis Accuracy

The user’s movements are determined by the angle of their arms. As long as the user’s arms are in frame of the camera, accuracy is very high. Angle ranges that are considered are the starting positions, the peak of the curl position, the minimum range of motion required to be considered a rep.



With these guidelines the feedback given to the user will only be based on authenticated reps.

#### Application Launch Time

MOFIT takes around 5 seconds to get to the login screen. Every page in the navigation takes around 1 second to load in due to the transition effect.

# Test Cases

## Test Planning

Testing for MOFIT is categorized into 3 categories, unit tests, performance tests, and security tests. The benefit of using Unity is that there is a testing system within the editor, which allows for easy and repeatable testing. Due to the majority of the work being done is within the backend and requires authentication, mock tests are performed to simulate the functions within the program. For example, authentication testing uses mock authentication services using the code for the real service to validate both success and failures in the authentication process. The majority of tests are within security due to the user’s private personal information being used within the application. All of their data is assumed to be sensitive and must be handled accordingly. For performance, I want authentication operations to be very quick, within 1000ms with a 95% success rate and 0 data loss for any operations. As of October 24, 2024, the tests have been created after the code for the program, which leads to the issue of the tests not being used to fix bugs and errors, but rather have the tests adapt to the current code, which I know is backwards and improper. All code in the future will be based on test-first test driven development to allow for easier bug tracking and fixing. My knowledge on test driven development was minimal but has improved with the creation of these tests. Having these tests will also prevent any future errors by notifying me if any changed I’ve made to code breaks anything, and will be sure to document these issues.

## Unit Tests

### Test Cases

#### Test: User Log in with invalid authorization code

Objective: Verify that the user log in process functions as expected

Code:

[UnityTest]

public IEnumerator Login\_User\_WithInvalidAuthCode\_ShouldReturnFalse()

{

    string invalidAuthCodeUrl = "https://dzi3ny7huab3j.cloudfront.net/?code=invalid-code";

    var loginTask = \_authManager.ExchangeAuthCodeForAccessToken(invalidAuthCodeUrl);

    yield return new WaitUntil(() => loginTask.IsCompleted);

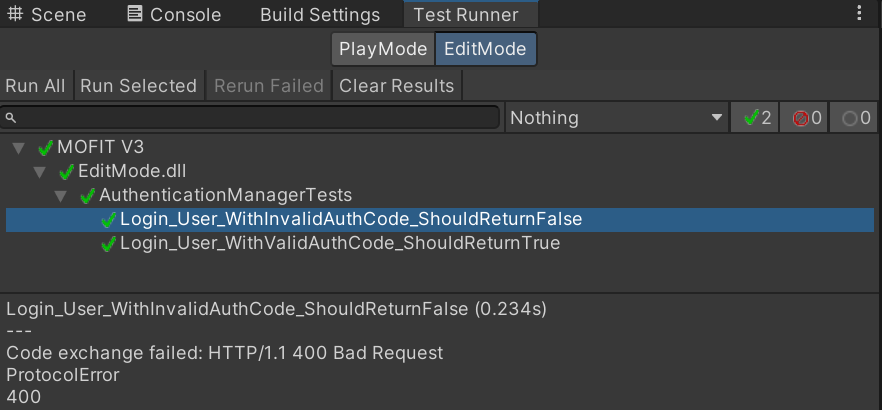
    Assert.IsFalse(loginTask.Result, "Expected login with invalid authorization code to fail.");

}

##### Expected Results

Expected Result: True when given an invalid auth. code

Test Result:



##### Bugs/Fixes

Currently No bugs or fixes needed

#### Test: User Log in with valid Authorization Code

Objective: Verify that the user log in process functions as expected

Code:

[UnityTest]

public IEnumerator Login\_User\_WithValidAuthCode\_ShouldReturnTrue()

{

    // Use a known valid auth code for testing

    string validAuthCode = "test-valid-auth-code";

    string validAuthCodeUrl = $"https://dzi3ny7huab3j.cloudfront.net/?code={validAuthCode}";

    // Set up the mock to expect this auth code

    \_authManager.SetMockValidAuthCode(validAuthCode);

    var loginTask = \_authManager.ExchangeAuthCodeForAccessToken(validAuthCodeUrl);

    yield return new WaitUntil(() => loginTask.IsCompleted);

    if (loginTask.Exception != null)

    {

        Debug.LogError($"Exception occurred: {loginTask.Exception}");

    }

    Assert.IsTrue(loginTask.Result, "Expected login with valid authorization code to succeed.");

    // Verify that session cache was updated

    UserSessionCache sessionCache = \_mockSaveDataManager.GetLastSavedData() as UserSessionCache;

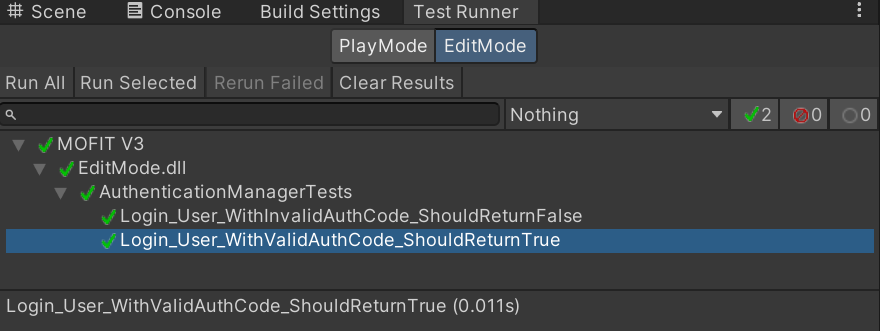
    Assert.IsNotNull(sessionCache, "Session cache should be saved after successful login");

    Assert.IsNotNull(sessionCache.getIdToken(), "ID token should be present in session cache");

}

##### Expected Results

Expected Result: True when given an invalid auth. code

Test Result: 

##### Bugs/Fixes

I first tried to use a real authentication url for the test cases, but found it to be unreliable and could not even get the test to pass with them.

Code for real url case:

[UnityTest]

public IEnumerator Login\_User\_WithValidAuthCode\_ShouldReturnTrue()

{

    // Assuming you have set up valid dependencies in AuthenticationManager.

    string validAuthCodeUrl = "https://dzi3ny7huab3j.cloudfront.net/?code=37cae09d-6700-4191-a8ba-2d9d2c81341c";

    // Run the async method and yield until it completes

    var loginTask = \_authManager.ExchangeAuthCodeForAccessToken(validAuthCodeUrl);

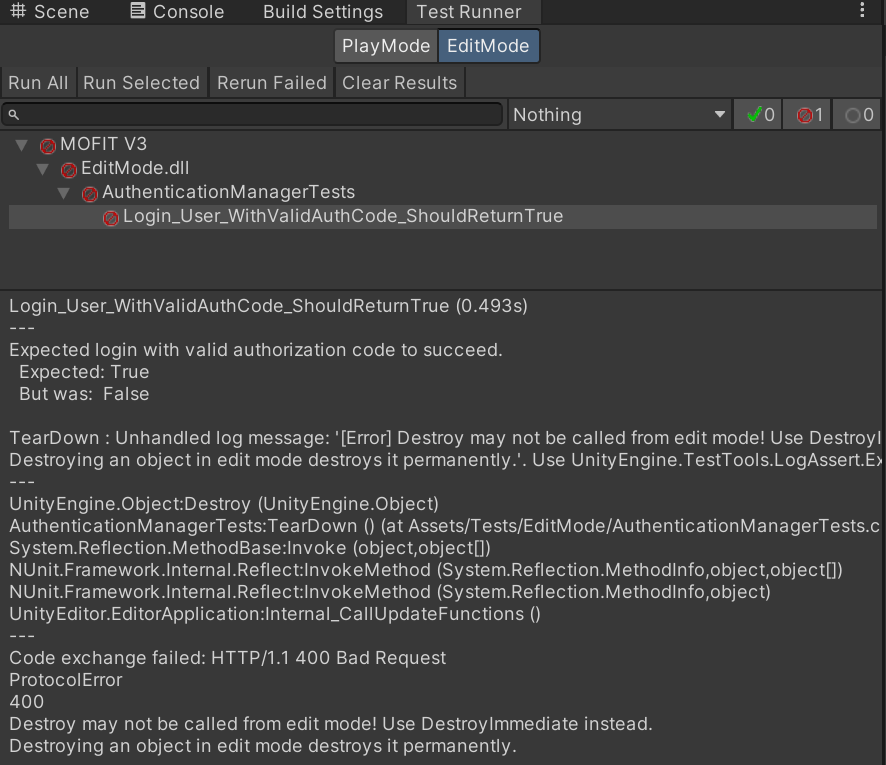
    yield return new WaitUntil(() => loginTask.IsCompleted);

    // Assert that the login was successful

    Assert.IsTrue(loginTask.Result, "Expected login with valid authorization code to succeed.");

}

Result:



## Performance Tests

### Test Cases

#### Test: Performance Test for Authentication

Objective: Test the time it takes for the authentication process to succeed.

Code:

[UnityTest]

    public IEnumerator TestAuthenticationPerformance()

    {

        Stopwatch stopwatch = new Stopwatch();

        long totalLoginTime = 0;

        long totalRefreshTime = 0;

        int loginSuccessCount = 0;

        int refreshSuccessCount = 0;

        // Test Login Performance

        for (int i = 0; i < NUM\_ITERATIONS; i++)

        {

            stopwatch.Reset();

            stopwatch.Start();

            var loginTask = \_authManager.ExchangeAuthCodeForAccessToken(TEST\_AUTH\_URL);

            yield return new WaitUntil(() => loginTask.IsCompleted);

            stopwatch.Stop();

            totalLoginTime += stopwatch.ElapsedMilliseconds;

            if (loginTask.Result)

                loginSuccessCount++;

            yield return null; // Give Unity a frame to process

        }

        // Test Token Refresh Performance

        for (int i = 0; i < NUM\_ITERATIONS; i++)

        {

            stopwatch.Reset();

            stopwatch.Start();

            var refreshTask = \_authManager.CallRefreshTokenEndpoint();

            yield return new WaitUntil(() => refreshTask.IsCompleted);

            stopwatch.Stop();

            totalRefreshTime += stopwatch.ElapsedMilliseconds;

            if (refreshTask.Result)

                refreshSuccessCount++;

            yield return null;

        }

        // Calculate averages

        long averageLoginTime = totalLoginTime / NUM\_ITERATIONS;

        long averageRefreshTime = totalRefreshTime / NUM\_ITERATIONS;

        // Log results

        UnityEngine.Debug.Log("Authentication Performance Results:");

        UnityEngine.Debug.Log($"Average login time: {averageLoginTime}ms");

        UnityEngine.Debug.Log($"Login success rate: {(float)loginSuccessCount / NUM\_ITERATIONS \* 100}%");

        UnityEngine.Debug.Log($"Average refresh time: {averageRefreshTime}ms");

        UnityEngine.Debug.Log($"Refresh success rate: {(float)refreshSuccessCount / NUM\_ITERATIONS \* 100}%");

        // Verify session cache updates

        UserSessionCache sessionCache = \_mockSaveDataManager.GetLastSavedData() as UserSessionCache;

        Assert.IsNotNull(sessionCache, "Session cache should be saved after operations");

        Assert.IsNotNull(sessionCache.getIdToken(), "ID token should be present in session cache");

        // Assert performance metrics

        Assert.That(averageLoginTime, Is.LessThan(MAX\_ACCEPTABLE\_TIME\_MS),

            $"Average login time ({averageLoginTime}ms) exceeds maximum acceptable time ({MAX\_ACCEPTABLE\_TIME\_MS}ms)");

        Assert.That(averageRefreshTime, Is.LessThan(MAX\_ACCEPTABLE\_TIME\_MS),

            $"Average refresh time ({averageRefreshTime}ms) exceeds maximum acceptable time ({MAX\_ACCEPTABLE\_TIME\_MS}ms)");

        // Assert success rates

        Assert.That((float)loginSuccessCount / NUM\_ITERATIONS, Is.GreaterThan(0.95f),

            "Login success rate should be above 95%");

        Assert.That((float)refreshSuccessCount / NUM\_ITERATIONS, Is.GreaterThan(0.95f),

            "Refresh success rate should be above 95%");

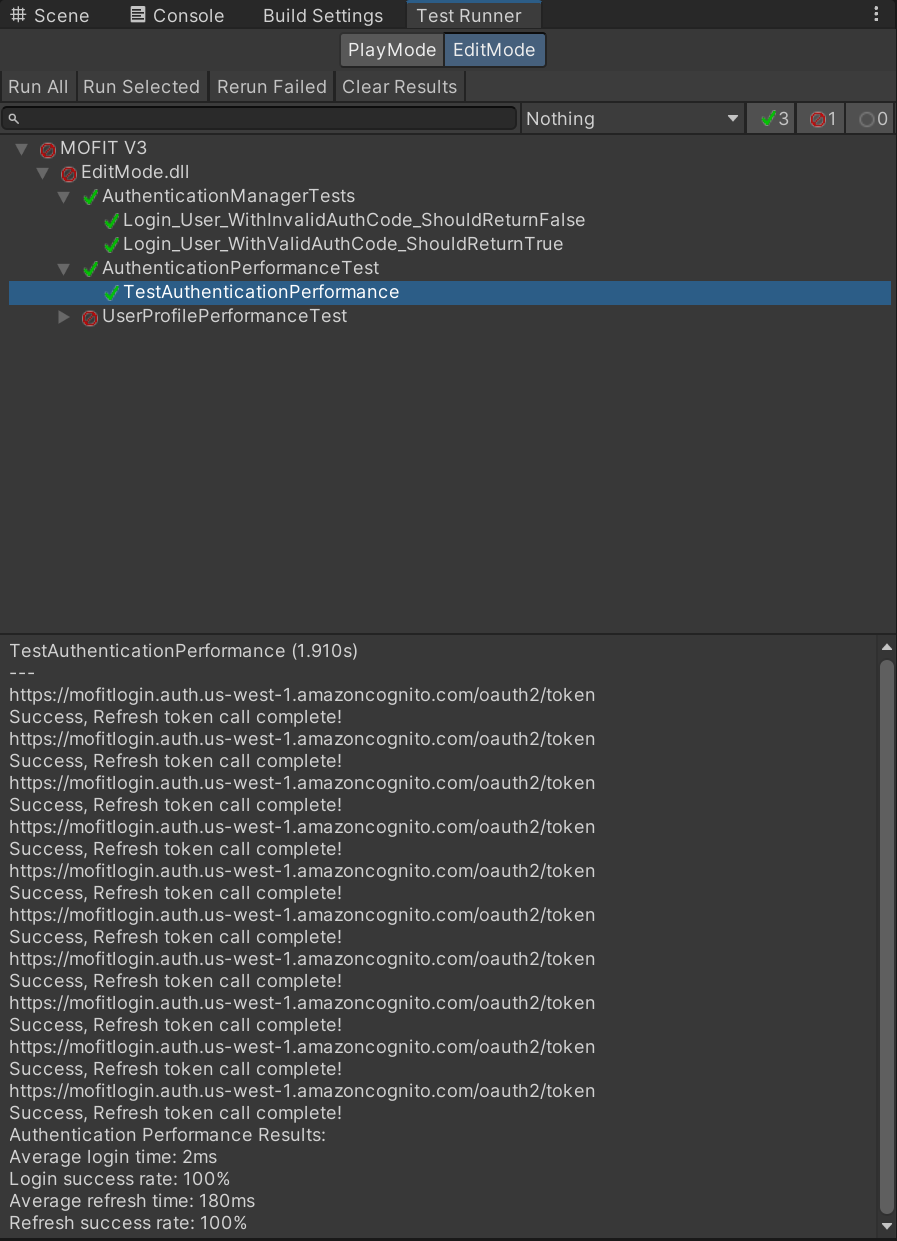
    }

##### Expected Results

Expected Result:

Success Rate and time it takes to authenticate  
The code currently tests for the time it takes for the user to log in with an authentication code and the time to refresh the token. It also tests average login time, login success rate, average refresh time, and refresh success rate.

Result:



##### Bugs/Fixes

Original test code:  
[UnityTest]

   public IEnumerator TestAuthenticationPerformance()

   {

       Stopwatch stopwatch = new Stopwatch();

       long totalTime = 0;

       int successCount = 0;

       for (int i = 0; i < NUM\_ITERATIONS; i++)

       {

           stopwatch.Reset();

           stopwatch.Start();

           var refreshTask = \_authManager.CallRefreshTokenEndpoint();

           yield return new WaitUntil(() => refreshTask.IsCompleted);

           stopwatch.Stop();

           totalTime += stopwatch.ElapsedMilliseconds;

           // Count successful refreshes

           if (refreshTask.Result)

               successCount++;

           yield return null; // Give Unity a frame to process

       }

       long averageTime = totalTime / NUM\_ITERATIONS;

       UnityEngine.Debug.Log($"Authentication Performance Results:");

       UnityEngine.Debug.Log($"Average time: {averageTime}ms");

       UnityEngine.Debug.Log($"Success rate: {(float)successCount / NUM\_ITERATIONS \* 100}%");

       // Test metrics

       Assert.That(averageTime, Is.LessThan(MAX\_ACCEPTABLE\_TIME\_MS),

           $"Average authentication time ({averageTime}ms) exceeds maximum acceptable time ({MAX\_ACCEPTABLE\_TIME\_MS}ms)");

   }

* Was trying to use real authentication codes, not viable for testing.

#### Test: User Profile saving and loading

Objective: See the performance of the program saving and loading a user profile by testing 50 mock profiles. The average saving and loading time are logged.

Code:

[UnityTest]

    public IEnumerator TestUserProfileLoadSavePerformance()

    {

        Stopwatch stopwatch = new Stopwatch();

        long totalLoadTime = 0;

        long totalSaveTime = 0;

        int successfulSaves = 0;

        int successfulLoads = 0;

        for (int i = 0; i < NUM\_ITERATIONS; i++)

        {

            string userId = $"test\_user\_{i}";

            UserProfile testProfile = CreateTestProfile(userId);

            // Test Save Performance

            stopwatch.Reset();

            stopwatch.Start();

            var saveTask = \_profileManager.UpdateUserProfile(testProfile);

            yield return new WaitUntil(() => saveTask.IsCompleted);

            stopwatch.Stop();

            if (saveTask.Result)

            {

                totalSaveTime += stopwatch.ElapsedMilliseconds;

                successfulSaves++;

            }

            // Test Load Performance

            stopwatch.Reset();

            stopwatch.Start();

            var loadTask = \_profileManager.GetUserProfile(userId);

            yield return new WaitUntil(() => loadTask.IsCompleted);

            stopwatch.Stop();

            if (loadTask.Result != null)

            {

                totalLoadTime += stopwatch.ElapsedMilliseconds;

                successfulLoads++;

                // Verify loaded profile

                Assert.AreEqual(testProfile.Name, loadTask.Result.Name,

                    $"Loaded profile does not match saved profile for user {userId}");

            }

            yield return null;

        }

        // Calculate averages only if we had successful operations

        float averageLoadTime = successfulLoads > 0 ? totalLoadTime / (float)successfulLoads : 0;

        float averageSaveTime = successfulSaves > 0 ? totalSaveTime / (float)successfulSaves : 0;

        UnityEngine.Debug.Log($"Profile Performance Results:");

        UnityEngine.Debug.Log($"Successful saves: {successfulSaves}/{NUM\_ITERATIONS}");

        UnityEngine.Debug.Log($"Successful loads: {successfulLoads}/{NUM\_ITERATIONS}");

        UnityEngine.Debug.Log($"Average save time: {averageSaveTime}ms");

        UnityEngine.Debug.Log($"Average load time: {averageLoadTime}ms");

        // Assert performance metrics

        Assert.That(successfulSaves, Is.EqualTo(NUM\_ITERATIONS), "Not all save operations were successful");

        Assert.That(successfulLoads, Is.EqualTo(NUM\_ITERATIONS), "Not all load operations were successful");

        Assert.That(averageSaveTime, Is.LessThan(MAX\_ACCEPTABLE\_TIME\_MS),

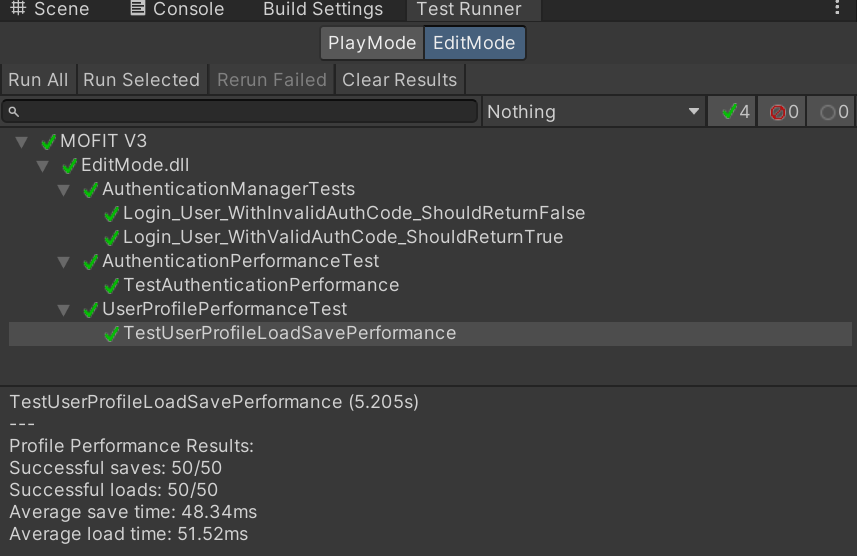
            $"Average save time ({averageSaveTime}ms) exceeds maximum acceptable time ({MAX\_ACCEPTABLE\_TIME\_MS}ms)");

        Assert.That(averageLoadTime, Is.LessThan(MAX\_ACCEPTABLE\_TIME\_MS),

            $"Average load time ({averageLoadTime}ms) exceeds maximum acceptable time ({MAX\_ACCEPTABLE\_TIME\_MS}ms)");

    }

##### Expected Result



## Security Tests

### Test Cases

#### Test: API Request Headers

Objective: Verify that the API requests have a proper security header and authorization tokens.  
Code:

[Test]

    public void Test\_API\_Request\_Headers()

    {

        TestContext.WriteLine("Testing API Request Headers Security");

        TestContext.WriteLine("-----------------------------------");

        TestContext.WriteLine("Purpose: Verify that API requests have proper security headers and authorization tokens");

        // Create and setup test request

        var webRequest = new UnityEngine.Networking.UnityWebRequest("https://api.example.com/test");

        string testToken = \_authManager.GetIdToken();

        TestContext.WriteLine("\nRequest Setup:");

        TestContext.WriteLine($"URL: {webRequest.url}");

        TestContext.WriteLine($"Initial Auth Header: {webRequest.GetRequestHeader("Authorization")}");

        // Add authorization

        webRequest.SetRequestHeader("Authorization", testToken);

        TestContext.WriteLine("\nAfter Setting Authorization:");

        TestContext.WriteLine($"Auth Header Present: {webRequest.GetRequestHeader("Authorization") != null}");

        TestContext.WriteLine($"Contains 'Bearer' prefix: {webRequest.GetRequestHeader("Authorization")?.Contains("Bearer")}");

        // Run assertions

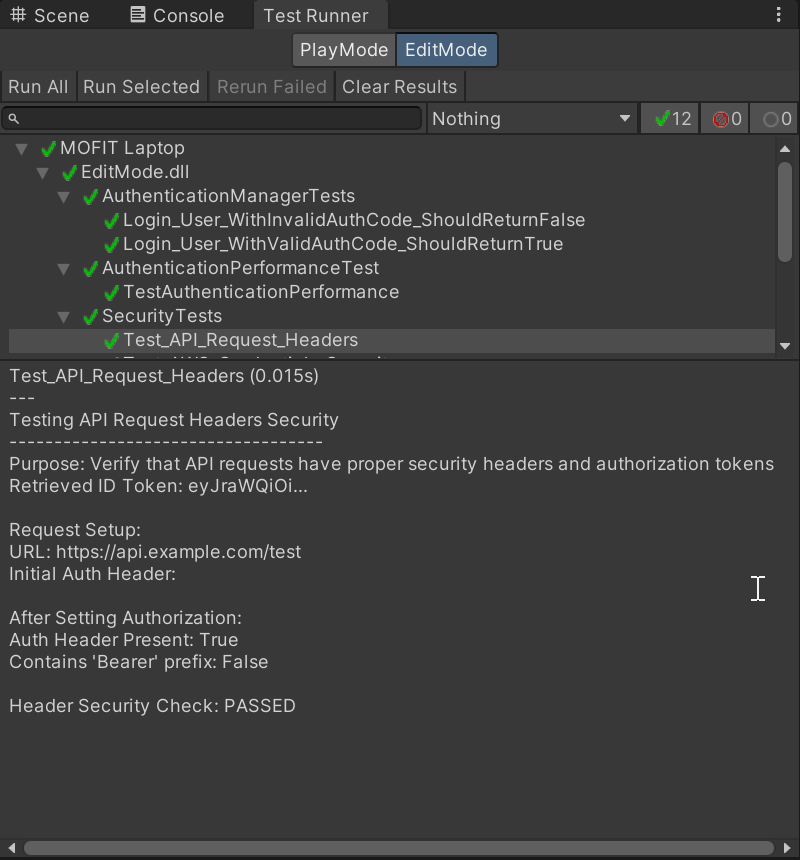
        Assert.IsTrue(webRequest.GetRequestHeader("Authorization") != null, "Authorization header missing");

        Assert.IsFalse(webRequest.GetRequestHeader("Authorization").Contains("Bearer"), "Token should not include 'Bearer' prefix");

        TestContext.WriteLine("\nHeader Security Check: PASSED");

    }

##### Expected Results



#### Test: AWS Credentials Security

Objective: Verify the AWS credentials are properly secured and meet the length requirements.  
Code:

[Test]

public void Test\_AWS\_Credentials\_Security()

{

    TestContext.WriteLine("Testing AWS Credentials Security");

    TestContext.WriteLine("--------------------------------");

    TestContext.WriteLine("Purpose: Verify AWS credentials are properly secured and meet length requirements");

    string accessKey = \_authManager.GetAccessKey();

    TestContext.WriteLine("\nCredentials Check:");

    TestContext.WriteLine($"Access Key Length: {accessKey.Length}");

    TestContext.WriteLine($"Key Format Valid: {accessKey.Length >= 16}");

    TestContext.WriteLine($"Key Present: {!string.IsNullOrEmpty(accessKey)}");

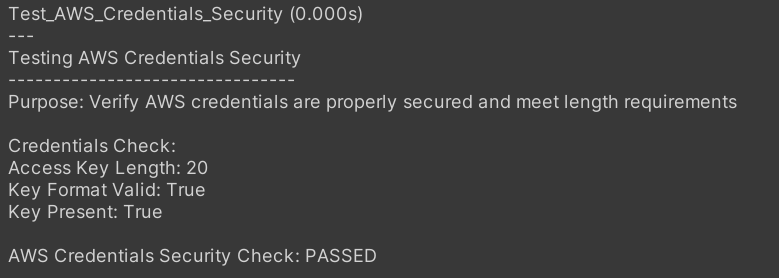
    Assert.IsFalse(string.IsNullOrEmpty(accessKey), "Access key should not be empty");

    Assert.IsFalse(accessKey.Length < 16, "Access key should be proper length");

    TestContext.WriteLine("\nAWS Credentials Security Check: PASSED");

}

##### Expected Results



#### Test: Password Storage Security

Objective: Verify that the passwords cannot be stored directly in PlayerPrefs and must use secure storage.  
Code:

[Test]

    public void Test\_Password\_Storage\_Security()

    {

        TestContext.WriteLine("Testing Password Storage Security");

        TestContext.WriteLine("--------------------------------");

        TestContext.WriteLine("Purpose: Verify that passwords cannot be stored directly in PlayerPrefs and must use secure storage");

        string testPassword = "dummy\_password";

        PlayerPrefs.DeleteAll();

        Dictionary<string, string> secureStorage = new Dictionary<string, string>();

        var mockSecurityManager = new MockSecurityManager(secureStorage);

        try

        {

            // Test direct storage prevention

            PlayerPrefs.DeleteKey("password");

            PlayerPrefs.SetString("password", testPassword);

            PlayerPrefs.Save();

            PlayerPrefs.DeleteKey("password");

            string directlyStoredPassword = PlayerPrefs.GetString("password", "");

            TestContext.WriteLine($"Direct Storage Test:");

            TestContext.WriteLine($"Attempted to store password: {testPassword}");

            TestContext.WriteLine($"Retrieved value from PlayerPrefs: '{directlyStoredPassword}'");

            // Test secure storage

            bool stored = mockSecurityManager.StoreCredential("login\_credential", testPassword);

            TestContext.WriteLine($"\nSecure Storage Test:");

            TestContext.WriteLine($"Storage success: {stored}");

            bool verified = mockSecurityManager.VerifyCredential("login\_credential", testPassword);

            TestContext.WriteLine($"Verification success: {verified}");

            Assert.AreEqual("", directlyStoredPassword, "Direct password storage in PlayerPrefs should not be allowed");

            Assert.IsTrue(stored, "Secure storage should succeed");

            Assert.IsTrue(verified, "Stored credential should be verifiable");

        }

        finally

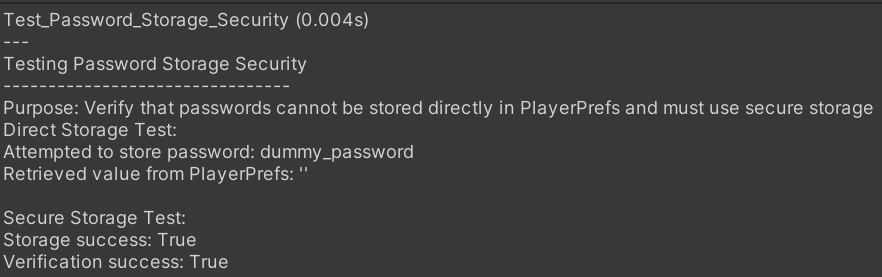
        {

            PlayerPrefs.DeleteAll();

        }

    }

##### Expected Results



#### Test: Session Cache Encryption

Objective: Verify that sensitive data is not stored in plaintext.   
Code:

[Test]

    public void Test\_Session\_Cache\_Encryption()

    {

        TestContext.WriteLine("Testing Session Cache Encryption");

        TestContext.WriteLine("-------------------------------");

        TestContext.WriteLine("Purpose: Verify sensitive session data is not stored in plaintext");

        // Create test session data

        UserSessionCache testCache = new UserSessionCache();

        string testToken = "test\_token";

        TestContext.WriteLine("\nTest Data:");

        TestContext.WriteLine($"Test Token: {testToken}");

        // Test encryption

        string jsonData = testCache.ToJson();

        bool containsPlaintext = jsonData.Contains(testToken);

        TestContext.WriteLine("\nEncryption Check:");

        TestContext.WriteLine($"JSON Data: {jsonData}");

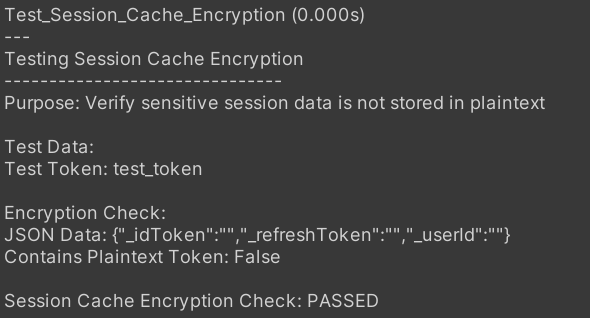
        TestContext.WriteLine($"Contains Plaintext Token: {containsPlaintext}");

        Assert.IsFalse(containsPlaintext, "Sensitive data should not be stored in plaintext");

        TestContext.WriteLine("\nSession Cache Encryption Check: PASSED");

    }

##### Expected Results



#### Test: Token Refresh Security

Objective: Verify that tokens are properly refreshed and changed during refresh process.  
Code:

[UnityTest]

   public IEnumerator Test\_Token\_Refresh\_Security()

   {

       TestContext.WriteLine("Testing Token Refresh Security");

       TestContext.WriteLine("-----------------------------");

       TestContext.WriteLine("Purpose: Verify tokens are properly refreshed and changed during refresh process");

       TestContext.WriteLine("\nInitiating Token Refresh...");

       var refreshTask = \_authManager.CallRefreshTokenEndpoint();

       yield return new WaitUntil(() => refreshTask.IsCompleted);

       TestContext.WriteLine("Token Refresh Completed");

       UserSessionCache sessionCache = new UserSessionCache();

       SaveDataManager.LoadJsonData(sessionCache);

       string newToken = sessionCache.getIdToken();

       TestContext.WriteLine("\nToken Comparison:");

       TestContext.WriteLine($"Old Token: old\_token");

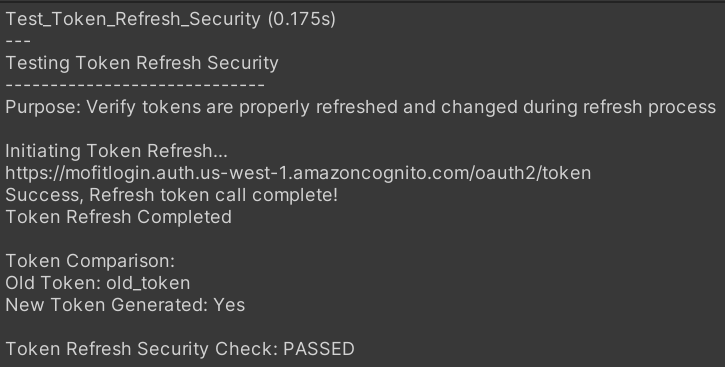
       TestContext.WriteLine($"New Token Generated: {(newToken != "old\_token" ? "Yes" : "No")}");

       Assert.AreNotEqual(sessionCache.getIdToken(), "old\_token", "Refresh should generate new tokens");

       TestContext.WriteLine("\nToken Refresh Security Check: PASSED");

   }

##### Expected Results



#### Test: Token Format Validation

Objective: Verify that tokens follow proper JWT format  
Code:

[Test]

public void Test\_Token\_Format\_Validation()

{

    TestContext.WriteLine("Testing JWT Token Format");

    TestContext.WriteLine("------------------------");

    TestContext.WriteLine("Purpose: Verify that tokens follow proper JWT format (three base64-encoded sections separated by dots)");

    string mockIdToken = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c";

    string jwtPattern = @"^[A-Za-z0-9-\_=]+\.[A-Za-z0-9-\_=]+\.?[A-Za-z0-9-\_.+/=]\*$";

    bool isValidFormat = Regex.IsMatch(mockIdToken, jwtPattern);

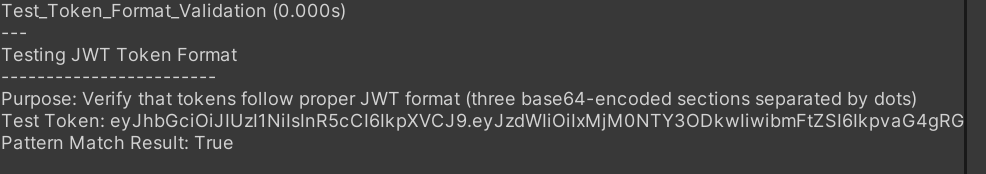
    TestContext.WriteLine($"Test Token: {mockIdToken}");

    TestContext.WriteLine($"Pattern Match Result: {isValidFormat}");

    Assert.IsTrue(isValidFormat, "ID Token does not match JWT format");

}

##### Expected Results



#### Test: URL Security

Objective: Verify login URLs use HTTPS and don’t contain sensitive information.  
Code:

[Test]

    public void Test\_URL\_Security()

    {

        TestContext.WriteLine("Testing URL Security");

        TestContext.WriteLine("-------------------");

        TestContext.WriteLine("Purpose: Verify login URLs use HTTPS and don't contain sensitive information");

        string loginUrl = \_authManager.GetLoginUrl();

        TestContext.WriteLine("\nURL Analysis:");

        TestContext.WriteLine($"Login URL: {loginUrl}");

        TestContext.WriteLine($"Uses HTTPS: {loginUrl.StartsWith("https://")}");

        TestContext.WriteLine($"Contains 'secret': {loginUrl.Contains("secret")}");

        TestContext.WriteLine($"Contains 'password': {loginUrl.Contains("password")}");

        // Run security checks

        bool usesHttps = loginUrl.StartsWith("https://");

        bool containsSecret = loginUrl.Contains("secret");

        bool containsPassword = loginUrl.Contains("password");

        TestContext.WriteLine("\nSecurity Checks:");

        TestContext.WriteLine($"HTTPS Check: {(usesHttps ? "PASSED" : "FAILED")}");

        TestContext.WriteLine($"Secret Check: {(!containsSecret ? "PASSED" : "FAILED")}");

        TestContext.WriteLine($"Password Check: {(!containsPassword ? "PASSED" : "FAILED")}");

        Assert.IsTrue(usesHttps, "Login URL must use HTTPS");

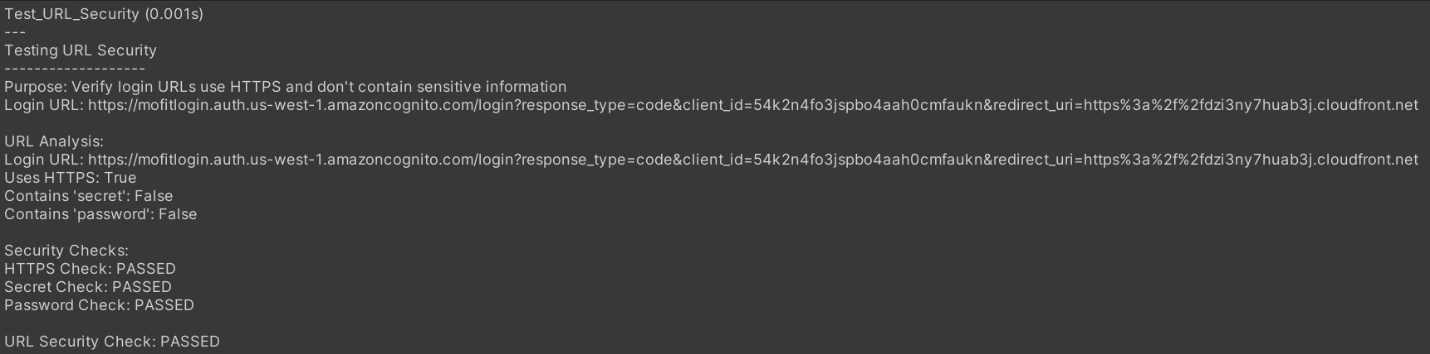
        Assert.IsFalse(containsSecret, "URL should not contain sensitive data");

        Assert.IsFalse(containsPassword, "URL should not contain sensitive data");

        TestContext.WriteLine("\nURL Security Check: PASSED");

    }

##### Expected Results



#### Test: User Profile Data Security

Objective: Verify that user profile data can be securely saved and loaded with data integrity.  
Code:

[UnityTest]

    public IEnumerator Test\_User\_Profile\_Data\_Security()

    {

        TestContext.WriteLine("Testing User Profile Data Security");

        TestContext.WriteLine("---------------------------------");

        TestContext.WriteLine("Purpose: Verify that user profile data can be securely saved and loaded with data integrity");

        Dictionary<string, UserProfile> mockProfileStorage = new Dictionary<string, UserProfile>();

        var mockProfileManager = new MockUserProfileManager(mockProfileStorage);

        string userId = "test\_user\_id";

        UserProfile testProfile = new UserProfile

        {

            UserId = userId,

            Name = "Test User",

            HeightFeet = "5",

            HeightInches = "10",

            CurrentWeight = "70"

        };

        TestContext.WriteLine("\nTest Profile Data:");

        TestContext.WriteLine($"User ID: {testProfile.UserId}");

        TestContext.WriteLine($"Name: {testProfile.Name}");

        TestContext.WriteLine($"Height: {testProfile.HeightFeet}'{testProfile.HeightInches}\"");

        TestContext.WriteLine($"Weight: {testProfile.CurrentWeight}");

        mockProfileManager.SaveProfile(testProfile);

        TestContext.WriteLine("\nProfile saved to mock storage");

        yield return null;

        UserProfile loadedProfile = mockProfileManager.GetProfile(userId);

        TestContext.WriteLine("\nProfile retrieved from mock storage:");

        TestContext.WriteLine($"Retrieved User ID: {loadedProfile?.UserId}");

        TestContext.WriteLine($"Retrieved Name: {loadedProfile?.Name}");

        Assert.IsNotNull(loadedProfile, "Loaded profile should not be null");

        bool dataMatches = testProfile.UserId == loadedProfile.UserId &&

                          testProfile.Name == loadedProfile.Name &&

                          testProfile.HeightFeet == loadedProfile.HeightFeet &&

                          testProfile.HeightInches == loadedProfile.HeightInches &&

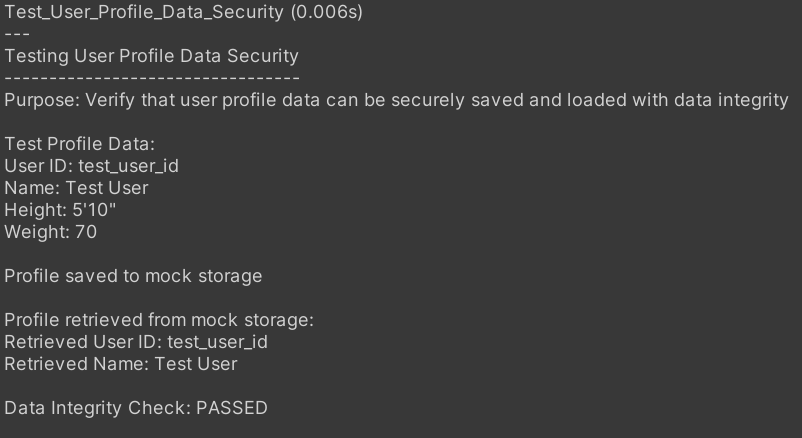
                          testProfile.CurrentWeight == loadedProfile.CurrentWeight;

        TestContext.WriteLine($"\nData Integrity Check: {(dataMatches ? "PASSED" : "FAILED")}");

        Assert.IsTrue(dataMatches, "Profile data integrity check failed");

    }

##### Expected Results



# Conclusion

In conclusion, MOFIT stands as a revolutionary fitness companion, utilizing technology to transform the way individuals approach their workout routines. With the integration of motion tracking, machine learning algorithms, and personalized coaching, MOFIT not only helps users through their workouts but also ensures safety and muscle development optimizations. MOFIT emphasizes inclusiveness, accessibility, and user-centric features such as a progress tracker and a comprehensive workout library for users to choose from. Furthermore, the developers of MOFIT demonstrate our dedication to delivering advanced technology through cloud computing, machine learning models, scalable databases, and many more.

As we proceed into the future of fitness technology, MOFIT sets a new standard by prioritizing user engagement, safety, and progression. By fostering a community of individuals committed to their health and well-being, MOFIT strives to inspire and support users in achieving their fitness goals, one rep at a time.

# References

Amazon Web Services, Inc. (n.d.). Amazon Relational Database Service. AWS. Retrieved May 2, 2024, from <https://aws.amazon.com/rds/>

Amazon Web Services, Inc. (n.d.). What is the AWS Mobile SDK for Unity? AWS Documentation. Retrieved September 28, 2024, from <https://docs.aws.amazon.com/mobile/sdkforunity/developerguide/what-is-unity-plugin.html>

Chi, C. (2022, August 18). Cloud integration: 8 best cloud integration platforms & tools. HubSpot Blog. <https://blog.hubspot.com/marketing/cloud-integration>

Codecademy. (n.d.). MVC: Model, View, Controller. Retrieved October 4, 2024, from <https://www.codecademy.com/article/mvc>

EnoxSoftware. (n.d.). YOLOv8WithOpenCVForUnityExample. GitHub. Retrieved October 4, 2024, from <https://github.com/EnoxSoftware/YOLOv8WithOpenCVForUnityExample>

Exercise.com. (n.d.). Average age of gym-goers. Retrieved October 4, 2024, from <https://www.exercise.com/grow/average-age-of-gym-goers/>

Franco, J. (2018). Formcheck. Devpost. <https://devpost.com/software/form-check-l3vfgq>

Google LLC. (n.d.). Cloud SQL for MySQL, PostgreSQL, and SQL Server. Google Cloud. Retrieved May 2, 2024, from <https://cloud.google.com/sql>

Harvard Health Publishing. (n.d.). Exercise and Fitness. Harvard Health. Retrieved October 4, 2024, from <https://www.health.harvard.edu/topics/exercise-and-fitness>

Kemtai. (2023, September 5). Motion tracking exercise platform for physio and fitness. <https://kemtai.com/>

Munawar, R., Jocher, G., Chaurasia, A., & Jiang, J. (2023, November 12). Ultralytics YOLOv8 Docs. Ultralytics. <https://docs.ultralytics.com/>

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (n.d.). YOLOv8: Real-Time Object Detection. GitHub. Retrieved April 30, 2024, from <https://github.com/ultralytics/ultralytics>

Refactoring.Guru. (n.d.). Observer. Retrieved October 4, 2024, from <https://refactoring.guru/design-patterns/observer>

Smith, D. (n.d.). BasicUX. Retrieved September 28, 2024, from <https://basicux.com/>

Takada, M. (n.d.). Unity-PullToRefresh. GitHub. Retrieved October 4, 2024, from <https://github.com/mtakada216/Unity-PullToRefresh/releases>

Williams, S. (2023, September 27). Why It's So Hard to Motivate Yourself to Exercise. Time. <https://time.com/6590020/why-its-so-hard-to-motivate-yourself-to-exercise/>