QuickFit:
The Micro-Fitness App

By: Josh, Abdullah, Robert, Ishaan

Table of Contents

Introduction

Users

Problem Statements

Requirements

Use Cases

SDLC Model

Architecture

Interfaces

Design Patterns

Design Principles

Testing Plan

UI/UX

Introduction

QuickFit is a micro-fitness app designed to help maximize activity done throughout the day.

Our objective is to motivate people to live a healthy life by working out in short bursts throughout the day. “No time? No problem! Get QuickFit”. We encourage doing physical, mental, and stretching exercises to keep the body active!

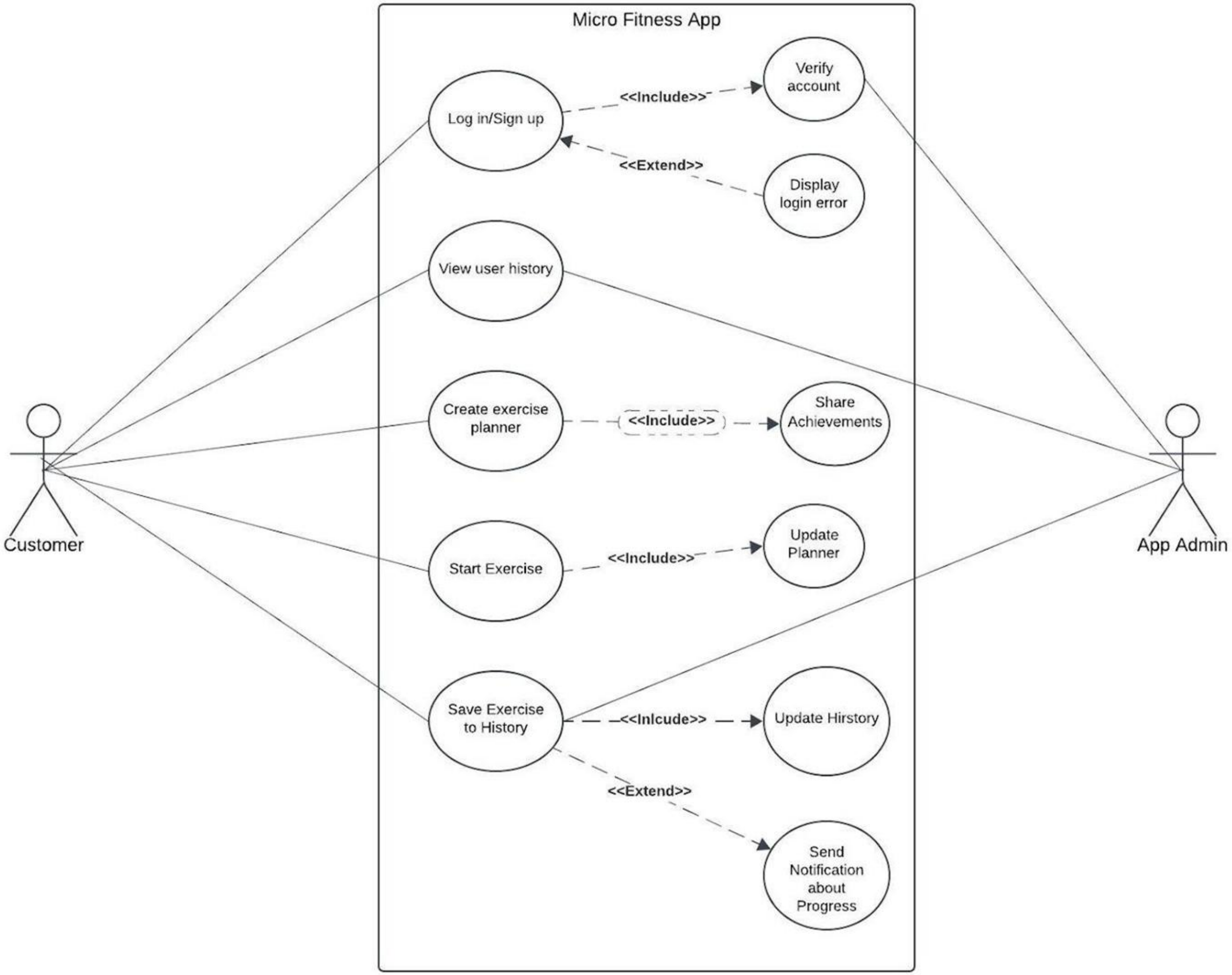
Users

Customers:

People using the app to do short burst exercises

Admin:

People who responsible for developing and maintaining the app



User Diagram

Problem Statement

01 Clark Kent is a software engineer at the Daily Bugle and works remotely. He wants to workout to strengthen his muscles and core, but is unable to do so due to frequent work meetings.

02 Bradley Martyn is a workout influencer on Instagram. He uses QuickFit to exercise and wants to motivate his followers to exercise by following his example.

03 Gwen Stacy wants to exercise regularly but loses motivation when working out alone. Her friends, Peter Parker and Harry Osborn, are in similar positions and slowly stop working out to due life stresses.

Requirements

Functional

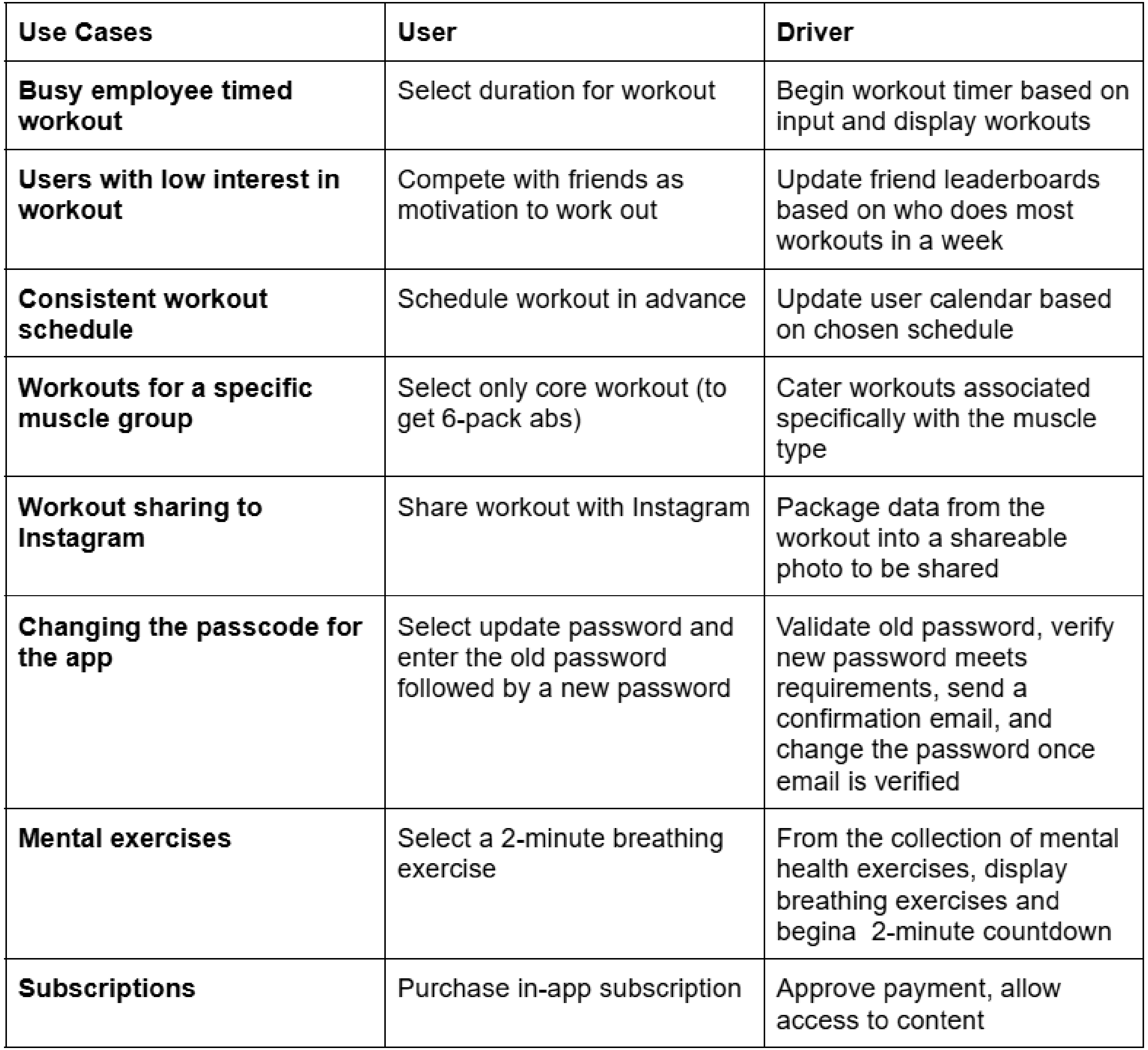
1. Secure user authentication
2. User has access to a wide range of exercises to choose from
3. Ability to be recommend workouts based on input
4. Display timer during workout
5. Allow offline workout download

Non-Functional

1. Quick access to the timed workout session
2. Easy to navigate / simple design
3. Scalable for 10,000+ users
4. Reliable
5. Compatible with Apple and Android with built in watch app

Use Cases

8 cases described also cover solutions to the problem statements



Use Cases	User	Driver
Busy employee timed workout	Select duration for workout	Begin workout timer based on input and display workouts
Users with low interest in workout	Compete with friends as motivation to work out	Update friend leaderboards based on who does most workouts in a week
Consistent workout schedule	Schedule workout in advance	Update user calendar based on chosen schedule
Workouts for a specific muscle group	Select only core workout (to get 6-pack abs)	Cater workouts associated specifically with the muscle type
Workout sharing to Instagram	Share workout with Instagram	Package data from the workout into a shareable photo to be shared
Changing the passcode for the app	Select update password and enter the old password followed by a new password	Validate old password, verify new password meets requirements, send a confirmation email, and change the password once email is verified
Mental exercises	Select a 2-minute breathing exercise	From the collection of mental health exercises, display breathing exercises and begina 2-minute countdown
Subscriptions	Purchase in-app subscription	Approve payment, allow access to content

Incremental Agile SDLC Model

Why we chose this model

While we had knew our main specifications, we wanted to make it easier to implement new features through the process

Adaptability to User Feedback

Agile allows you to incorporate user feedback after each increment, validating the app to align with our requirements

Faster delivery of Usable Features

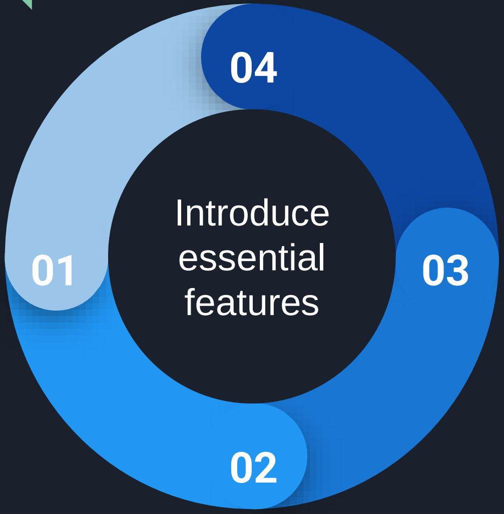
Users get early access to the app with basic functionality, allowing you to generate interest, gather feedback, and verify functionality

Prioritize essential features

This approach allows us to prioritize essential features initially and introduce advanced features over time

Incremental Cycles

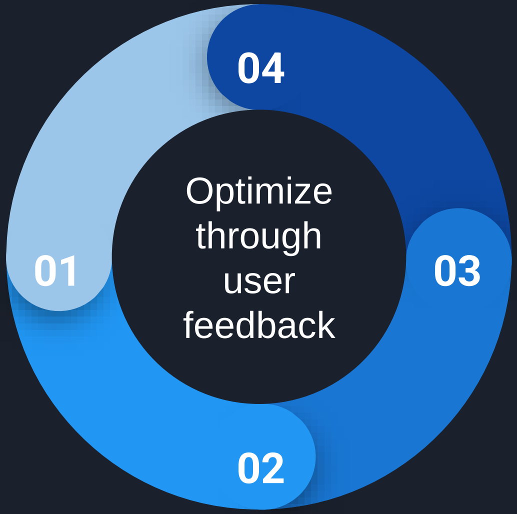
1



Introduce essential features

Timer & basic workouts selection for physical workouts

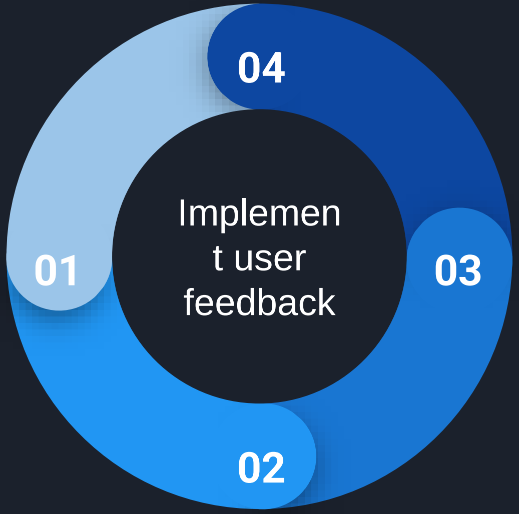
2



Optimize through user feedback

Remaining workout categories, user profiles, and API integration (Instagram sharing)

3+



Implemen t user feedback

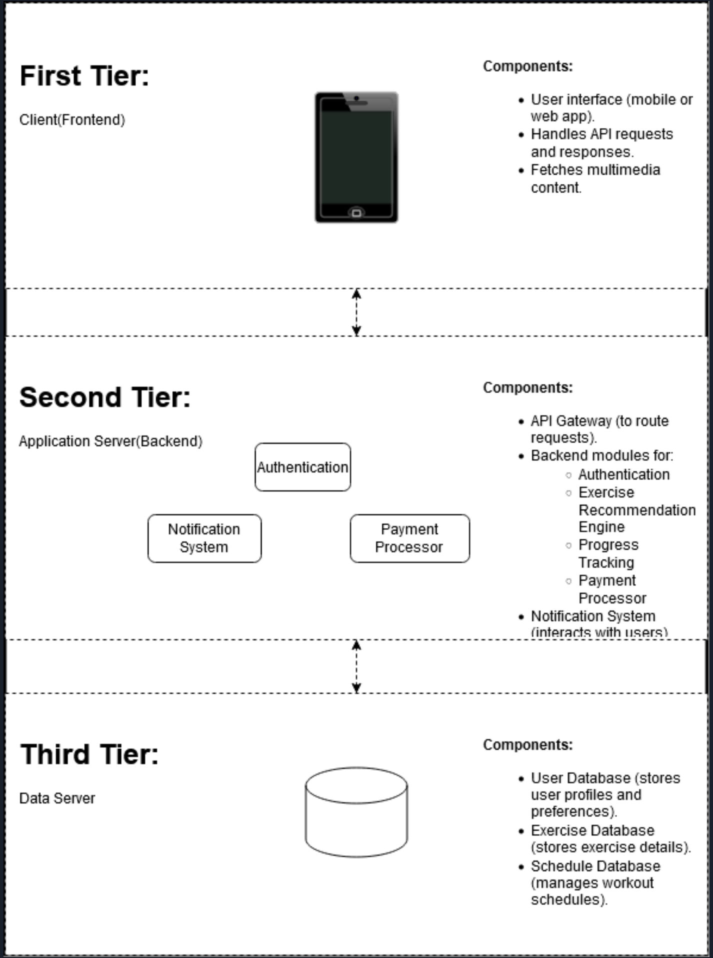
AI recommendations, analytics, offline mode, friends leaderboards, and more

Client Server Architecture

QuickFit uses a three-tier client-server architecture, separating the system into Client (Frontend), Application Server (Backend), and Data Server (Databases) to ensure modularity, scalability, extensibility, security and maintainability.

A client-server architecture with **horizontal scaling** also allows system reusability across different servers. Updates made to one module can be applied across all servers.

Architecture Diagram



First Tier:

Client(Frontend)

Components:

- User interface (mobile or web app).
- Handles API requests and responses.
- Fetches multimedia content.

Second Tier:

Application Server(Backend)

Components:

- API Gateway (to route requests).
- Backend modules for:
  - Authentication
  - Exercise Recommendation Engine
  - Progress Tracking
  - Payment Processor
- Notification System (interacts with users)

Third Tier:

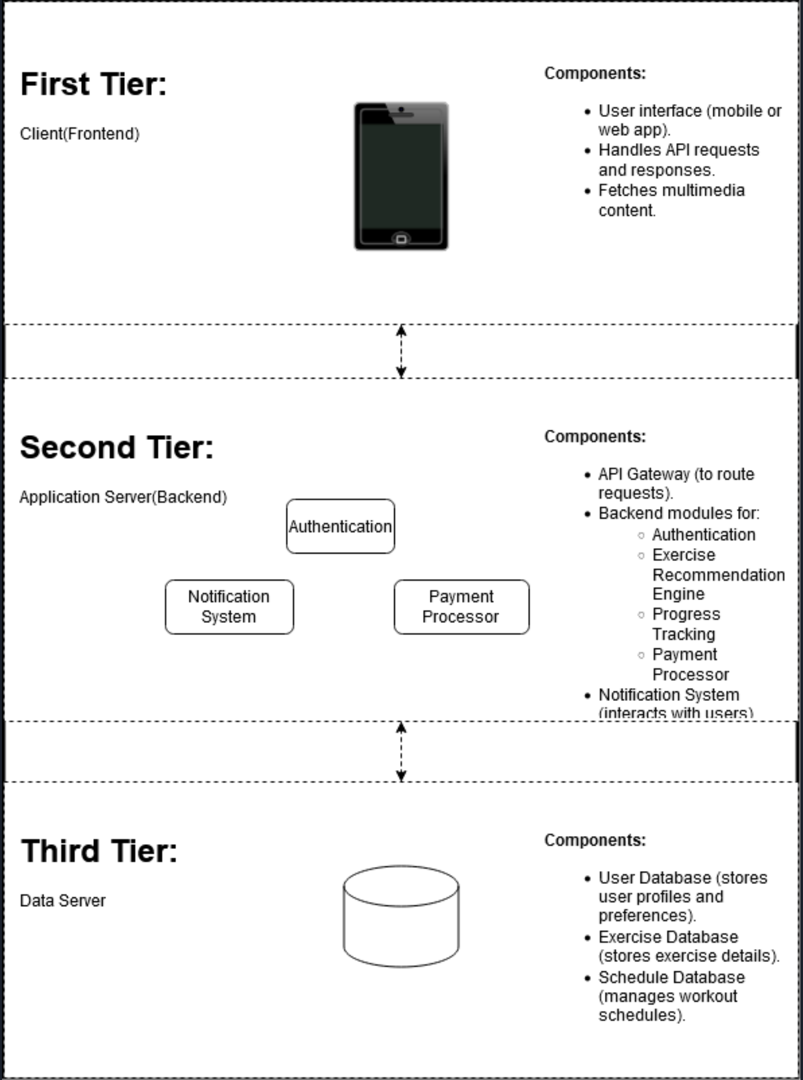
Data Server

Components:

- User Database (stores user profiles and preferences).
- Exercise Database (stores exercise details).
- Schedule Database (manages workout schedules).

Architecture cont…

Backend processes business logic, including authentication, exercise recommendations, progress tracking, and notifications. It also manages payment processing and interacts with databases to retrieve or store data.



Components:

- User interface (mobile or web app).
- Handles API requests and responses.
- Fetches multimedia content.

Components:

- API Gateway (to route requests).
- Backend modules for:
  - Authentication
  - Exercise Recommendation Engine
  - Progress Tracking
  - Payment Processor
- Notification System (interacts with users)

Components:

- User Database (stores user profiles and preferences).
- Exercise Database (stores exercise details).
- Schedule Database (manages workout schedules).

Frontend provides the user interface, captures input, and communicates with the backend via the API Gateway for tasks like fetching workout routines, logging progress, and receiving multimedia content from the CDN.

Databases store user profiles, exercise metadata, and workout schedules, ensuring efficient and secure data management.

Interfaces

Frontend

- BeginWorkout(double time, workout workout_type : bool)
- GetWorkoutTime(vector<double> time_options:double)
- ShareWorkout(double duration, workout session, system instagram_id : image)
- GetScheduleDays(vector<string> days : vector<string>)
- GetScheduleDuration(vector<double> times : double)
- CreateSchedule(vector<string> days, double duration : calendar)
- GetPhysicalWorkout(string muscle_type : vector<string>)
- GetMentalHealthWorkout(string mental_type : vector<string>)
- GetStretchingWorkout(string stretching_type : vector<string>)
- PurchaseSubscription(money amount, string subscription_type : bool)
- Notify(notification message_alert : bool)

Backend

- UpdateApp(: bool)
- ChangePassword(password old, password new : bool)
- SaveProgress(calendarDate date, double duration, workout session : bool)
- LoadProfile(user profile_name : bool)
- GetExerciseTypes(string workout_type : workout)
- FindExercises(workout session : workout)

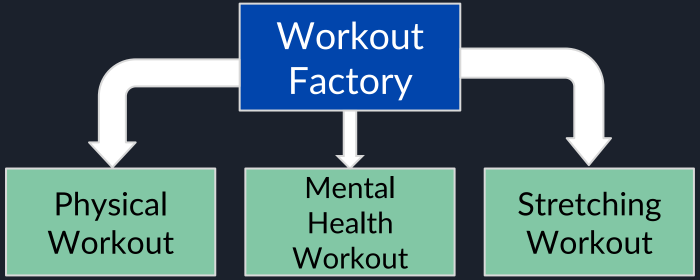
Payment Processor

- InitiatePayment(money amount, user profile_name, string subscription_type : bool)

Authenticator

- Login(string username, password code : bool)
- Logout(user profile_name:bool)
- 2FA(user profile_name, string email_id : bool)

Design Patterns



Factory Pattern: Encapsulate the creation process of Physical, Mental Health, and Stretching Workout objects. The base workout object will act as a template for the other types of workouts.

Singleton Pattern: Combine workout logs in one location to optimize analysis and workout recommendations

Strategy Pattern: Use in the backend to suggest workouts based on selected duration and category

Design Principles

Separation and Modularity for Loose Coupling: By designing system components with unique, self-contained methods (e.g., getWorkoutType(), getExerciseType()), we ensure **loose coupling**, allowing individual components to scale or be replaced without impacting others. This reflects **Separation of Concerns** and **Single Responsibility**, as each method focuses solely on its task while promoting a clear, maintainable structure.

Extensibility: The design of this system has clear separation of concerns and error-handling mechanisms while supporting extensibility. Separated components can be modified and extended, and more components can be introduced to the system.

Scalability, Extensibility, and Reusability through Generality and Open/Closed Design: Methods like `findExercises()` can handle generalized parameters and return consistent types, supporting **reusability** and **modularity**. New functionalities, such as adding exercise categories or modifying a workout type, can be integrated seamlessly without altering existing components, adhering to the **Open/Closed Principle**.

E.g. Interfaces like **GetWorkoutTime(vector<double> time_options:double)** and workout objects use vectors over arrays to allow additional time and workout category options.

Testing Plan

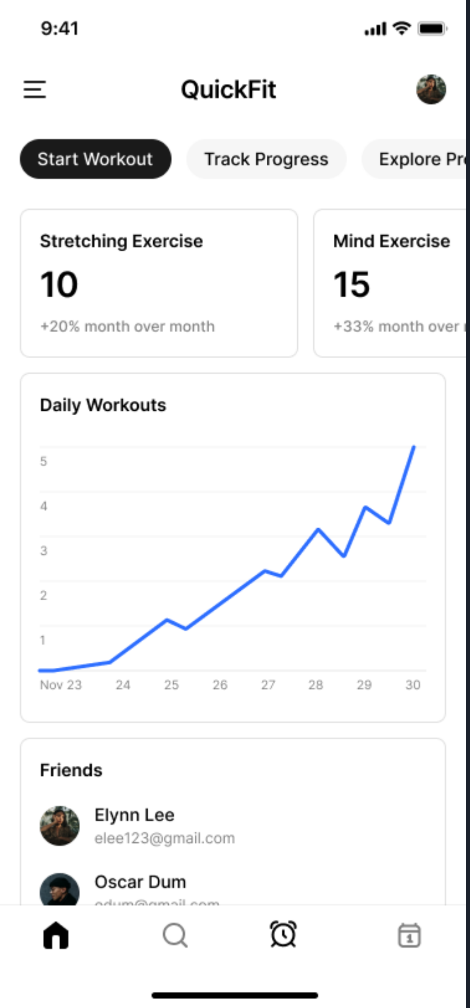
Our interfaces must be able to match the non-functional requirements of the user:

For example:

- **BeginWorkout()** is tested to be successful if the return correct workout
  - This method will considered failed if it returns the improper workout type or null
- **Inputs**: Time (double), workout type (object), duration (vector<double>), and Instagram ID (system input)
- **Outputs**: Boolean success indicators, images, or errors handled appropriately

Reporting Mechanisms:

- Test results will be recorded in a shared GitHub repository
- Bugs will be logged in an issue tracker (e.g., Jira) and assigned to relevant team members



9:41

QuickFit

Start Workout | Track Progress | Explore Pr

Stretching Exercise	Mind Exercise
10	15
+20% month over month	+33% month over

Daily Workouts

Friends

Elynn Lee
elee123@gmail.com

Oscar Dum
odum@gmail.com

QuickFit Home Screen

QuickFit app home screen highlights **quick access**, a key non-functional requirement, with intuitive navigation buttons like **"Start Workout"** and **"Track Progress."** The design ensures usability and responsiveness with features like detailed stats, a **Daily Workouts tracker**, and a "**Friends**" section for engagement. The clean layout enhances accessibility and efficiency, meeting user needs effectively.

Thank you!