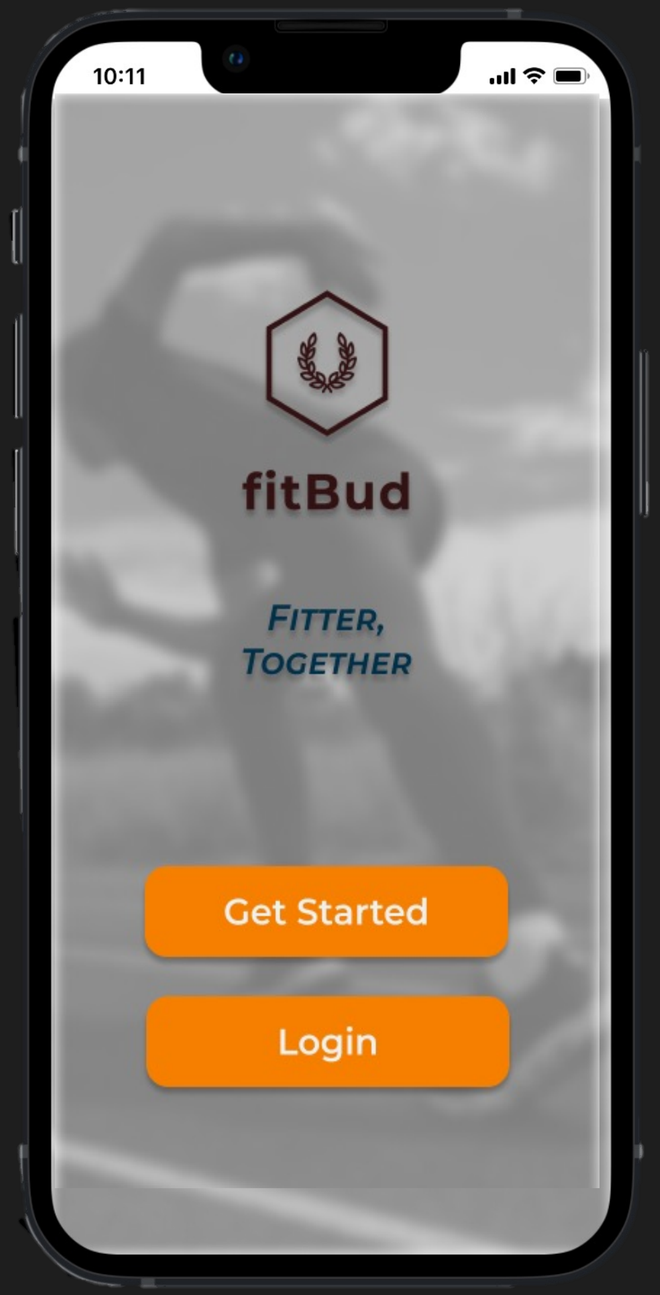



# FitBud

*FITTER TOGETHER*

## SAMMY

Summit | Aaron | Marc | Mingyang | Yu Fei

# The Problem

**7 in 10** NSMen in their 30s fail their IPPT

**Why?**

Lack of Motivation

Don’t have a plan on how to get started

**4 in 10** young adults gaining weight

**Why?**

Lazy

Unhealthy Lifestyle

# What is FitBud?

FitBud is a one-stop platform and a fitness buddy for NSFs and NSMen to train for their IPPT.

# Expected Users

NSFs, NSMen and Fitness Enthusiasts

# Main Functionalities

### Dedicated Fitness Plan

Our app aims to curate a personalized fitness plan for the user.


### Pose Detector

Our app integrates AI to help users detect their posture and help them improve.

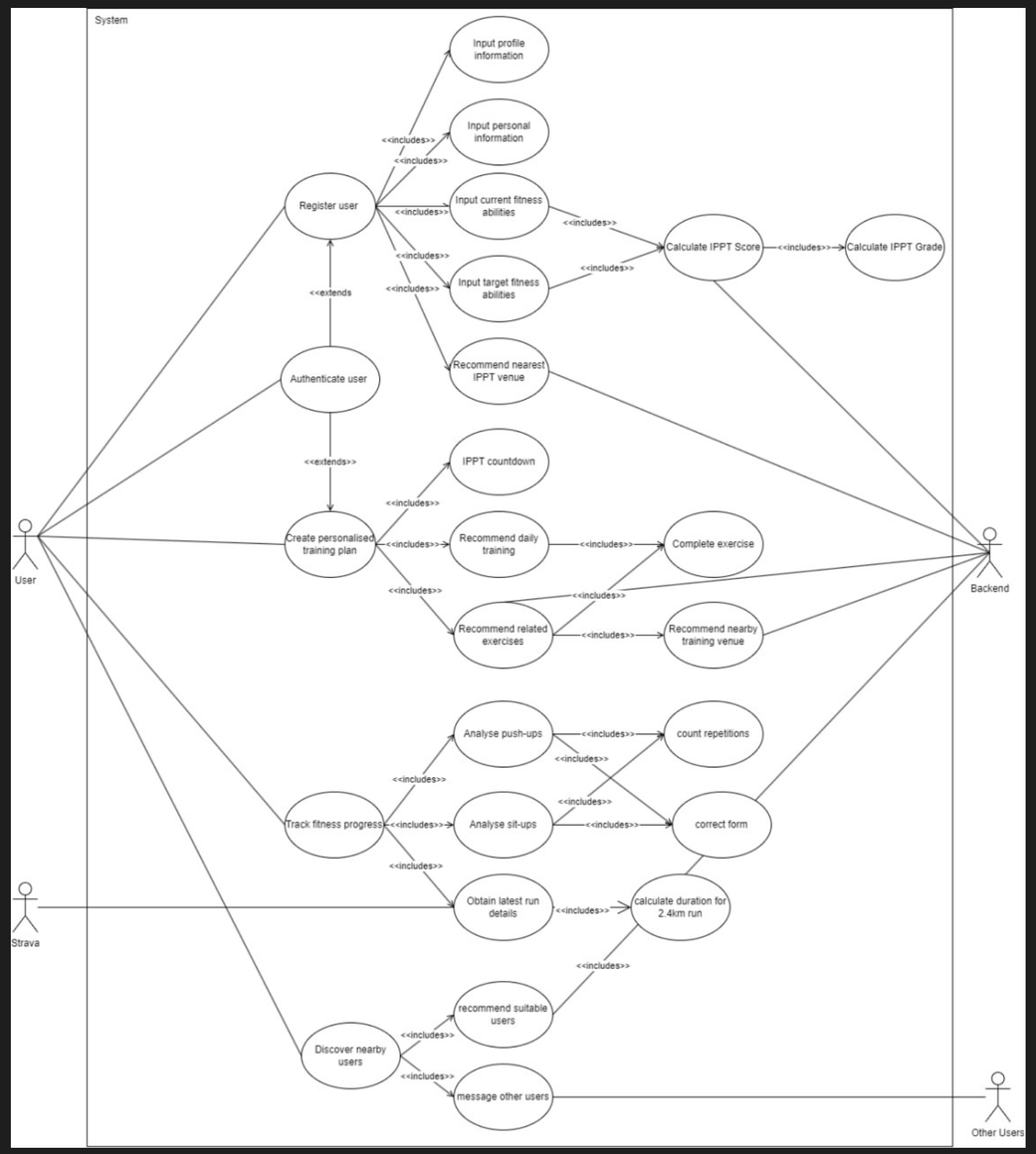
### Connecting With Friends

Our app aims to allow users to connect with other nearby users with similar fitness level so that they have a fitness buddy.

### Recommend Exercise Location

Our app integrates government data sets on parks and gyms to recommend locations for users to train.

# Overview of Use Case Diagram





## Main Use Cases

### Register User

To prompt for user information during registration.

### Authenticate User

To prompt for username and password.

### Create Personalized Training Plan

To create a personalized training plan for the user based on their current and target fitness.

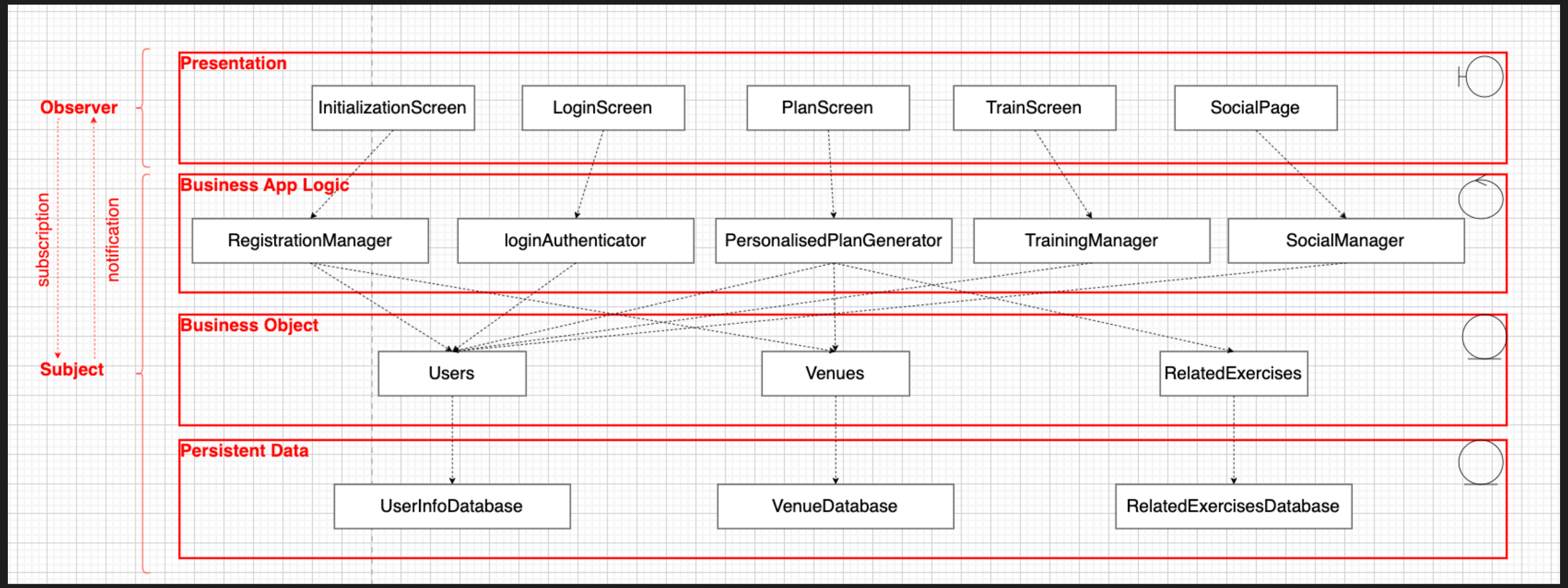
### Track Fitness Progress

To track user's push-ups, sit-ups and run timings.

### Discover Nearby Users

To connect nearby users with similar IPPT scores and arrange meet-ups for users.

# Overview of System Design



# Good Software Practices

## SOLID Framework in UML

- Single Responsible Principle
- Open-Closed Principle
- Liskov's Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

Achieve loose coupling and high cohesion, resulting in **reusability**, **extensibility** and **maintainability**

## SCRUM Framework

- Focuses on iterative development
- We adopted a 1-week sprint cycle with daily SCRUM meetings and weekly reviews.

Allows efficient delivery of project. Reviews allow to gather feedback and re-optimize methods.

## Testing

White box testing:
- Basis path testing
- Control flow testing

Black box testing:
- Equivalence class testing
- Boundary value testing

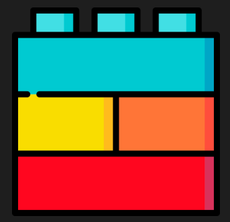
Good mix of black and white box testing allows for comprehensive bugs identification

# Good Software Practices

## Managing Requirements

- Continuously reviewing documentations as we go along.
- Bringing up discrepancy during SCRUM meetings
- Maintaining forward and backward traceability of documents

Essential in quality and process control



## Component Based Architecture

- Adopting a component-based software development
- Reusing react-native and custom-built components

Improves maintainability and extensibility of software

## Control Changes to Software

- Decomposing architecture into subsystems and assigning them
- Creating branches for team members to commit to

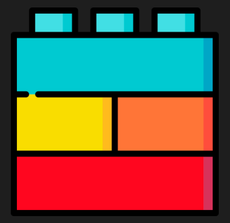
Ensures that there is no overlapping of work and previous work done is not affected.

# How can we support future upgrades?

Having a SOLID framework in our UML reduces dependencies and promotes extensibility. Software can be updated or upgraded without impacting other areas of our code.

Using SCRUM as our methodology allows us to break down our upgrades to smaller pieces. This allows us to build iteratively and efficiently.

Managing documentation is important because it allows us to find gaps within our software. This aids us in our upgrades.



A component-based architecture allows us to stack components on one another. This makes implementation efficient and simpler.

# Tech Stack

Frontend

React Native

Backend

node JS

mongoDB

Express JS

Cloud service

aws

# Data Sets

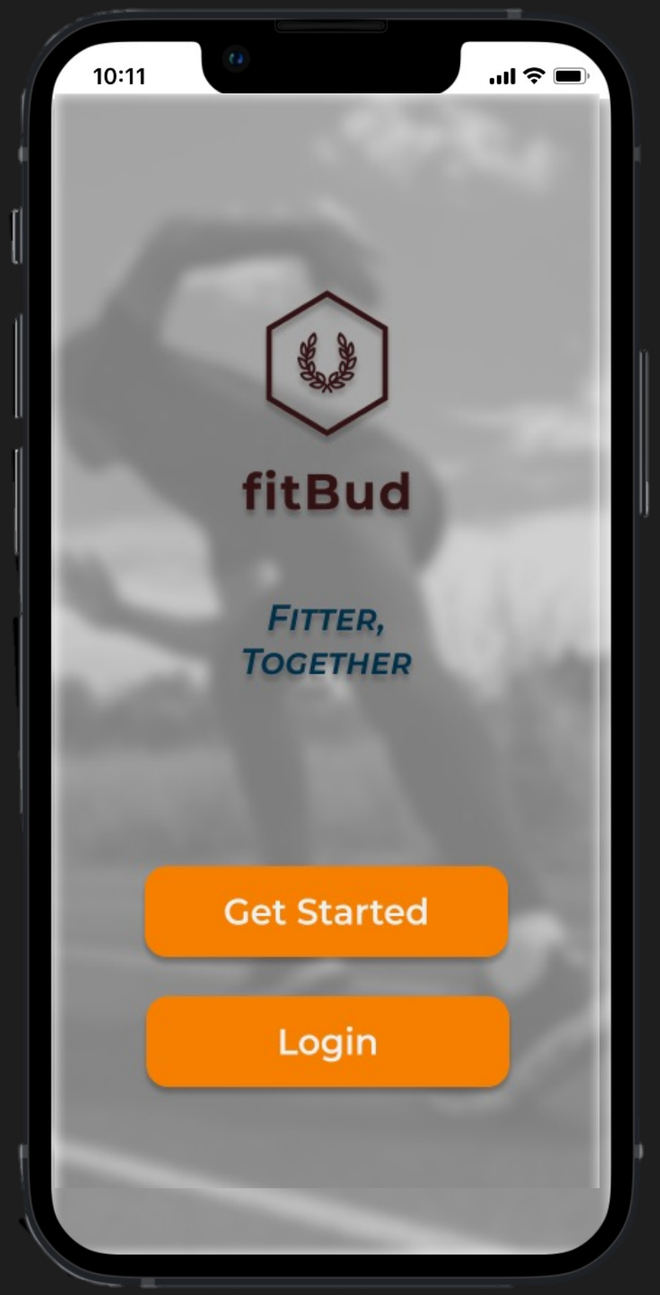
FCC Conditioning Centres

ExRx.net

IPPT in your community

IPPT scoring chart

Gyms@SG




# Live Demonstration

# Traceability of Input Profile Information

## Use Case

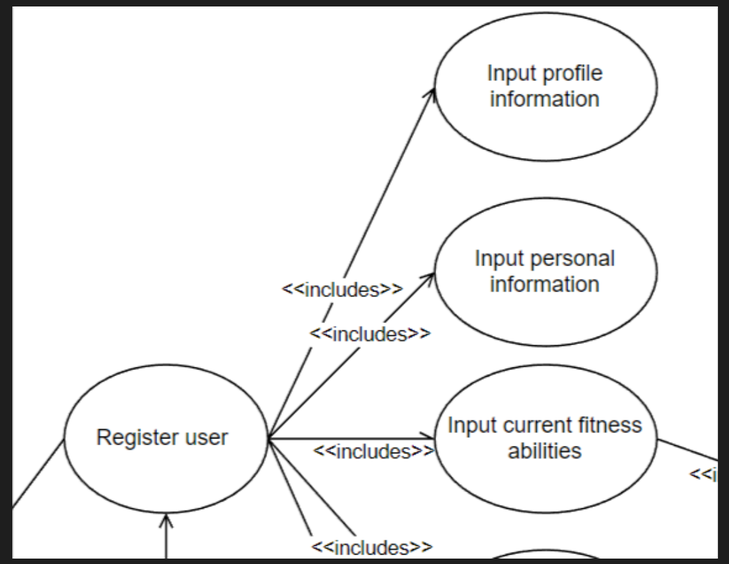
Functional Requirement:

The app must prompt for the user's login information.

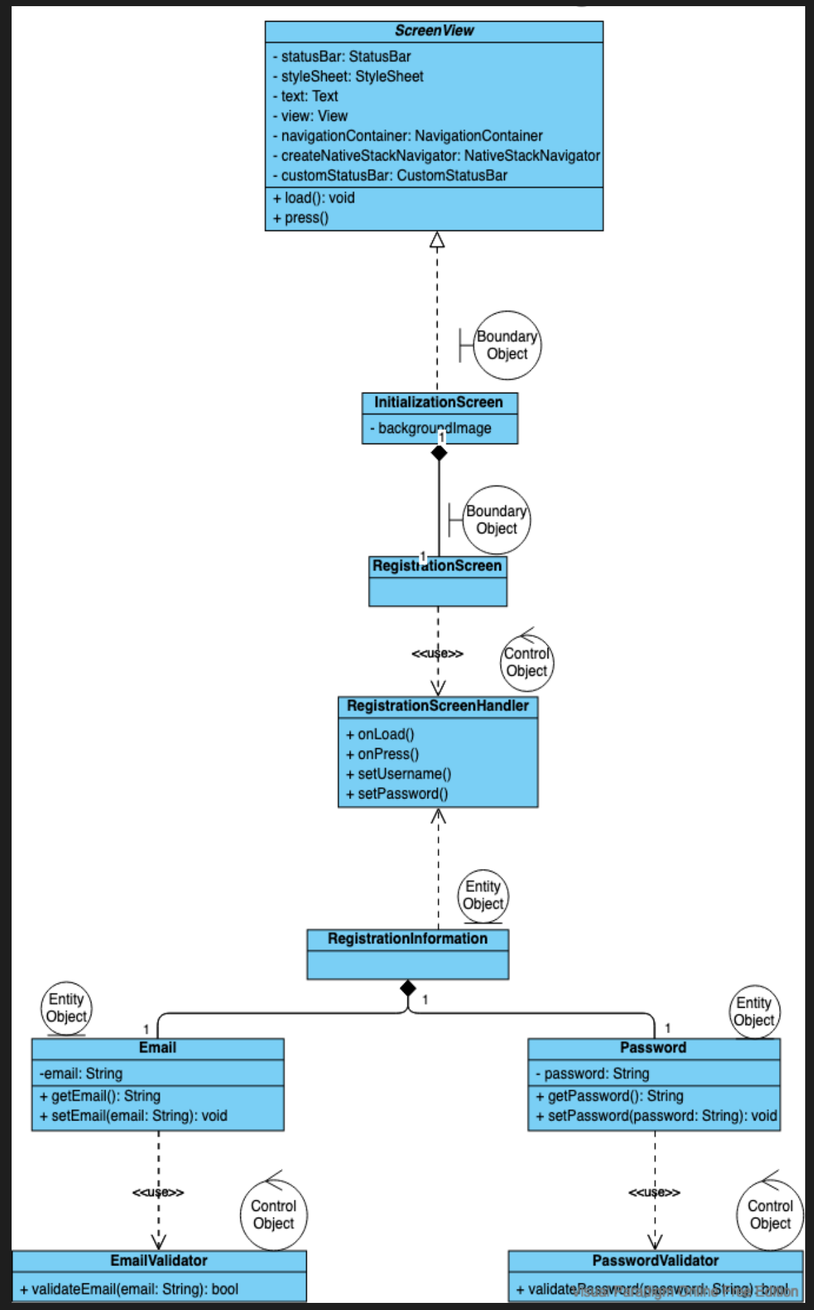
1. The app must prompt for the user's username, which must have at least one character.

2. The app must prompt for the user's account password which length must be at least 8, have at least 1 letter, have at least 1 number and must not contain the user's username.

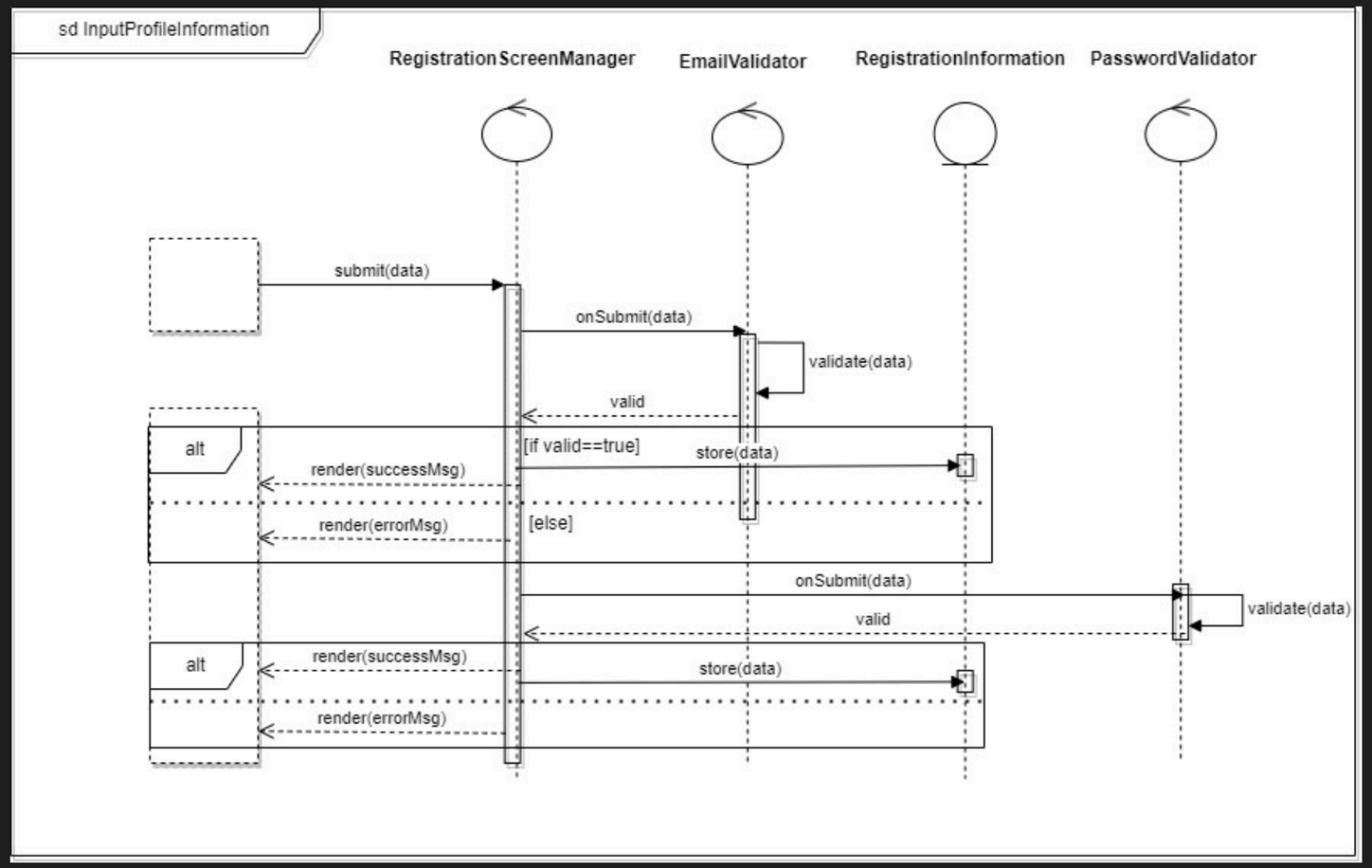
## Use Case Diagram

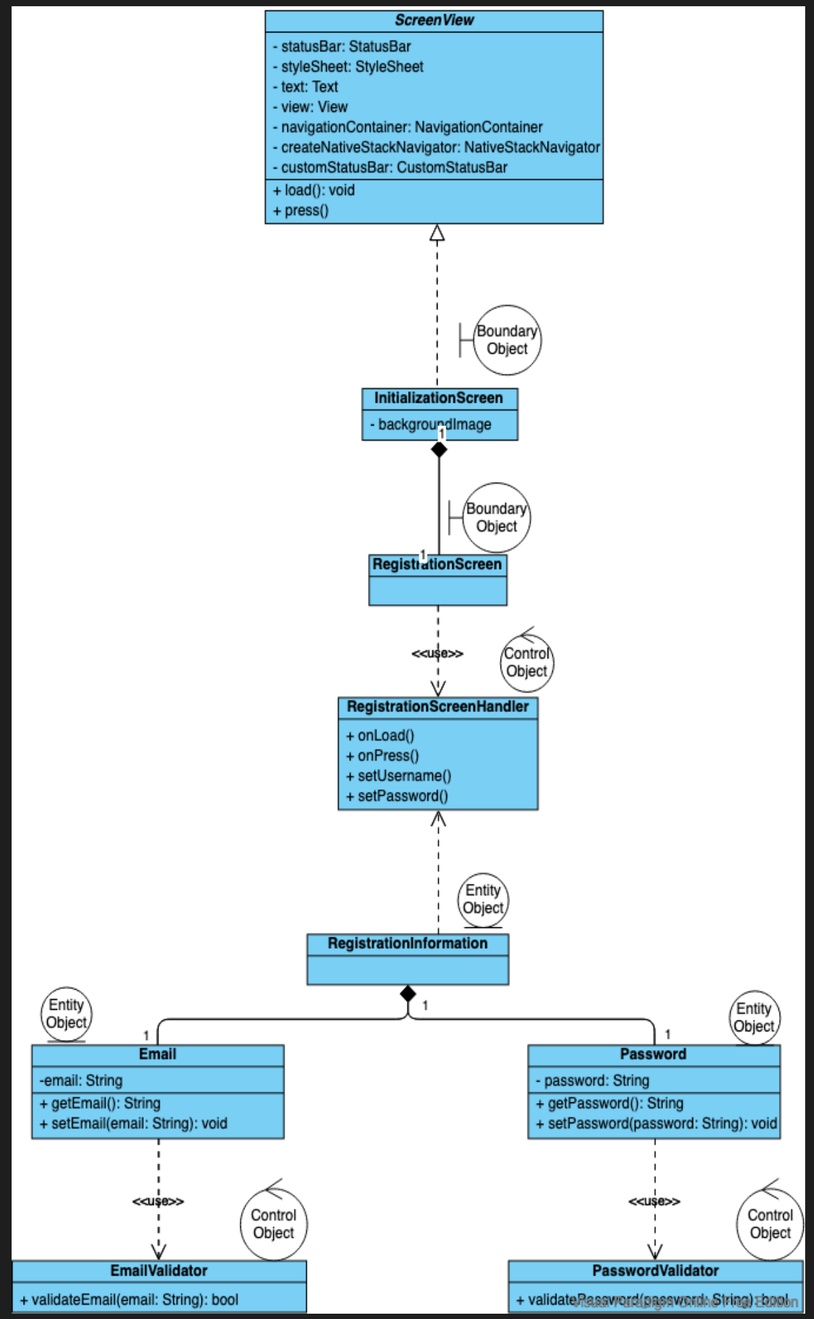


## Class Diagram



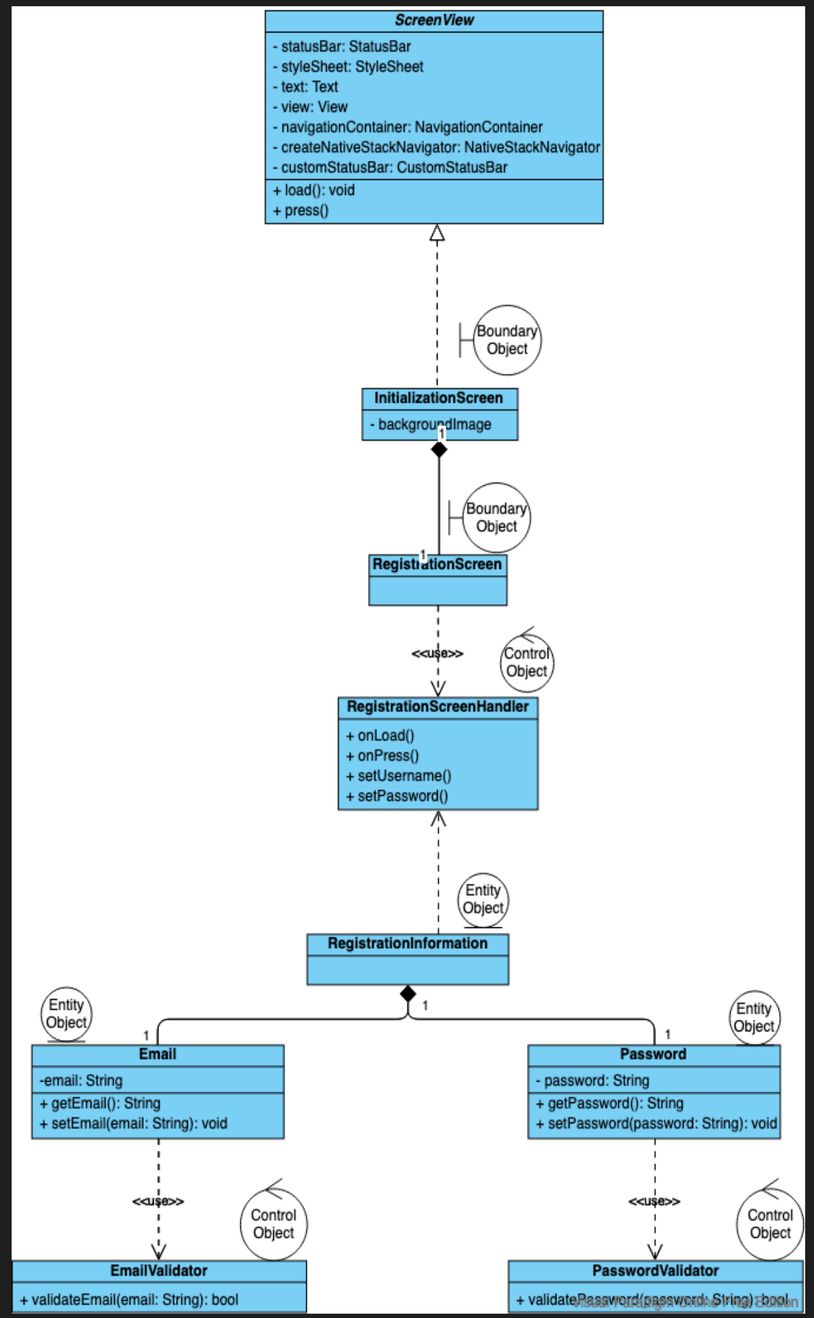

## Sequence Diagram






# Good Design Principles

- Single Responsibility Principle
  - Each of the classes has a specific responsibility. This improves the changeability of the code, with simple and independent units interacting through well defined interfaces
  - The validator is specific to this use case
- Open-Closed Principle
  - RegistrationInformation class is open for extension, but is closed for modification. If there are new registration information required (eg. NRIC number), it is possible to add new registration information without modifying the code in the RegistrationInformation class
- Liskov Substitution Principle
  - RegistrationScreenHandler implements the ScreenHandler class. As the RegistrationScreenHandler child has the same pre- and post- conditions as the ScreenHandler parent, it is able to substitute for the parent



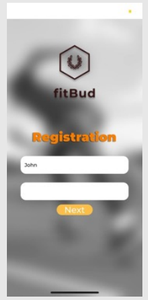

# Good Design Principles

- Interface Segregation Principle
  - Usage of many client specific interfaces instead of one general purpose interface
  - Classes do not depend on interfaces that they do not use
- Dependency Injection Principle
  - The registration screen is plugged into a framework via the abstract class

# Implementation

```
const pwValidator = () => {
  if (password.length > 8) {
    setValidLen(true)
  } else setValidLen(false);

  const regExpChar = /[A-Za-z]/
  if (password.match(regExpChar)) {
    setValidChar(true);
  } else setValidChar(false);

  const regExpDig = /[0-9]/
  if (password.match(regExpDig)){
    setValidNum(true);
  } else setValidNum(false);

  if (email != '' && password.toLowerCase().includes(email.toLowerCase())){
    setValidNoUser(false);
  } else setValidNoUser(true);

  if (validLen && validChar && validNum && validNoUser){
    setValid(true);
  } else setValid(false);
}
```

# Testing

| Test Input | Oracle | Log |
|---|---|---|
| Password | Expected Output | Actual Output |
| ilovesc2006 | No output message, app allows user to proceed |   |
| johnlovesc2006 | “Password contains username”, user is not allowed to proceed |   |

| Test Input | Oracle | Log |
|---|---|---|
| Password | Expected Output | Actual Output |
| ilovescse | “Password does not contain any numbers”, user is not allowed to proceed. |   |
| 12345678 | “Password does not contain any characters”, user is not allowed to proceed. |   |

# Traceability of Analyze Push-Up/Sit-Up

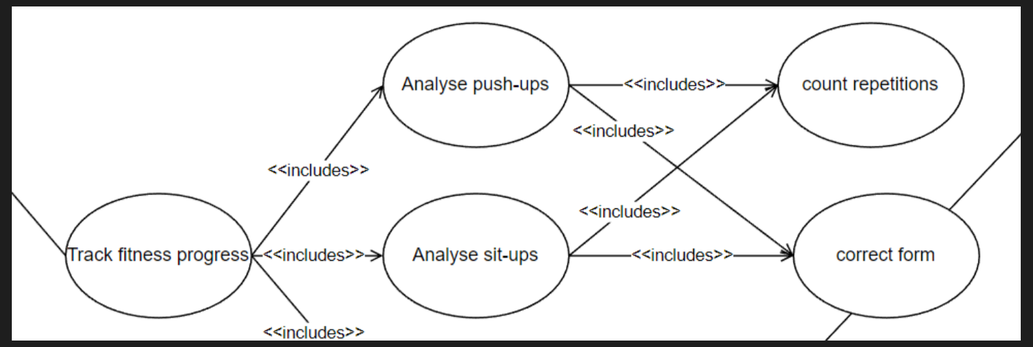
## Use Case

Functional Requirement:

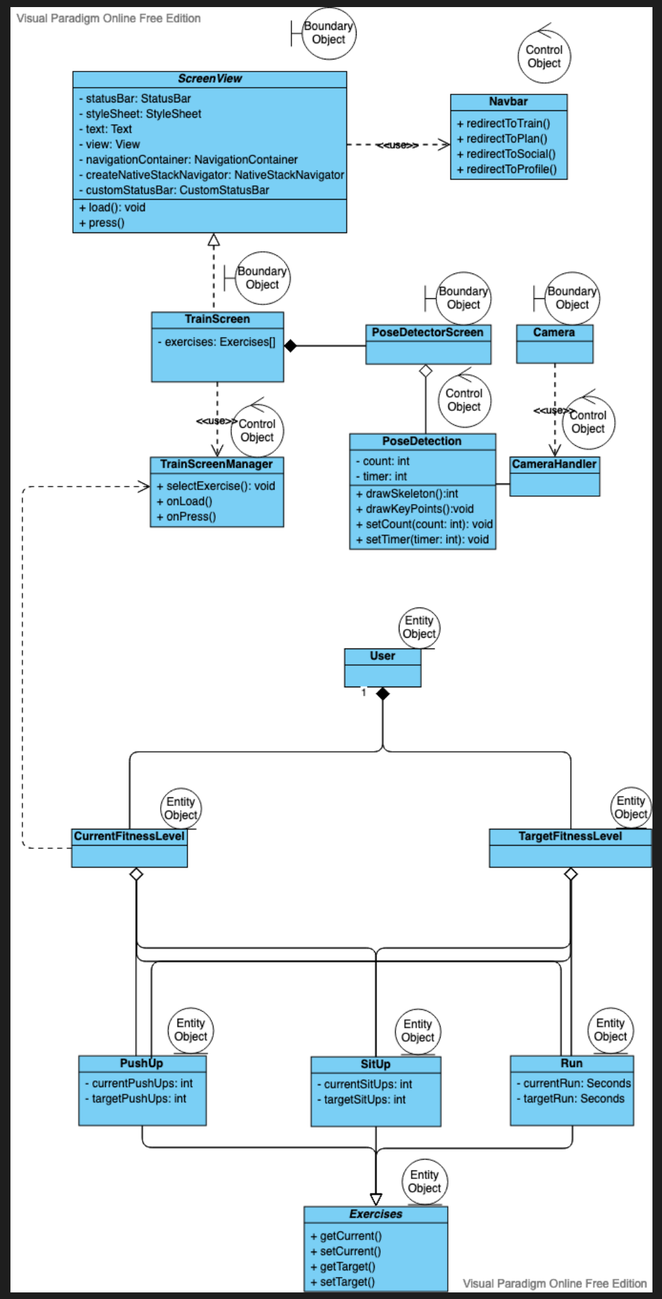
The app must be able to accurately track and correct the user's push-ups during his attempts.

1. The app must allow the user to begin an attempt which will start a 60 second timer.
2. The app must classify the correct push-up form from an incorrect one with an accuracy of 90%.
3. The app must count the number of correct push-ups performed by the user within the attempt.
4. The app must display live feedback on the screen to correct the user's form during the attempt.
5. The app must have the option for users to submit the results of their attempt to be stored.
6. The app must have the option for users to redo their attempt as many times as desired.

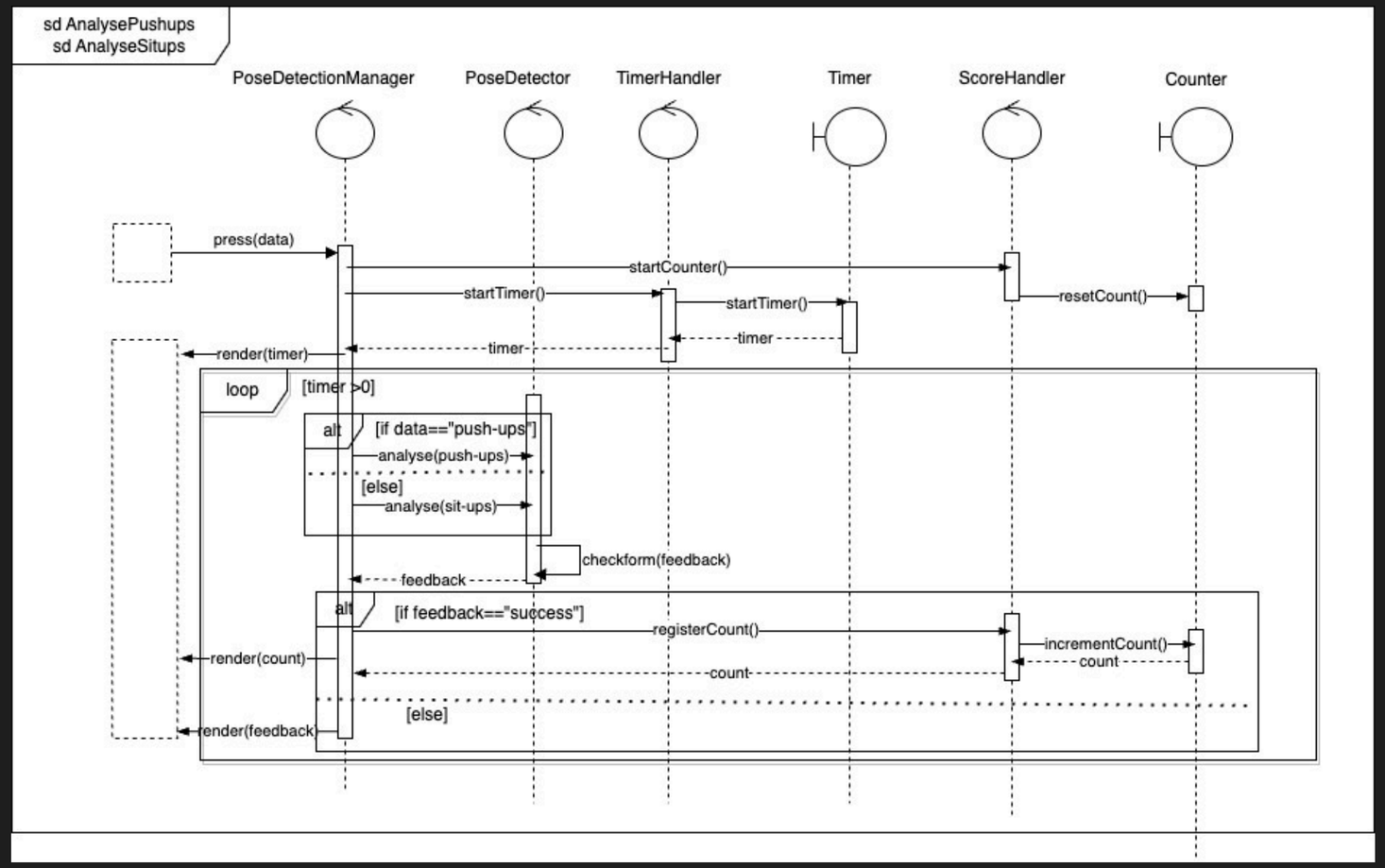
## Use Case Diagram

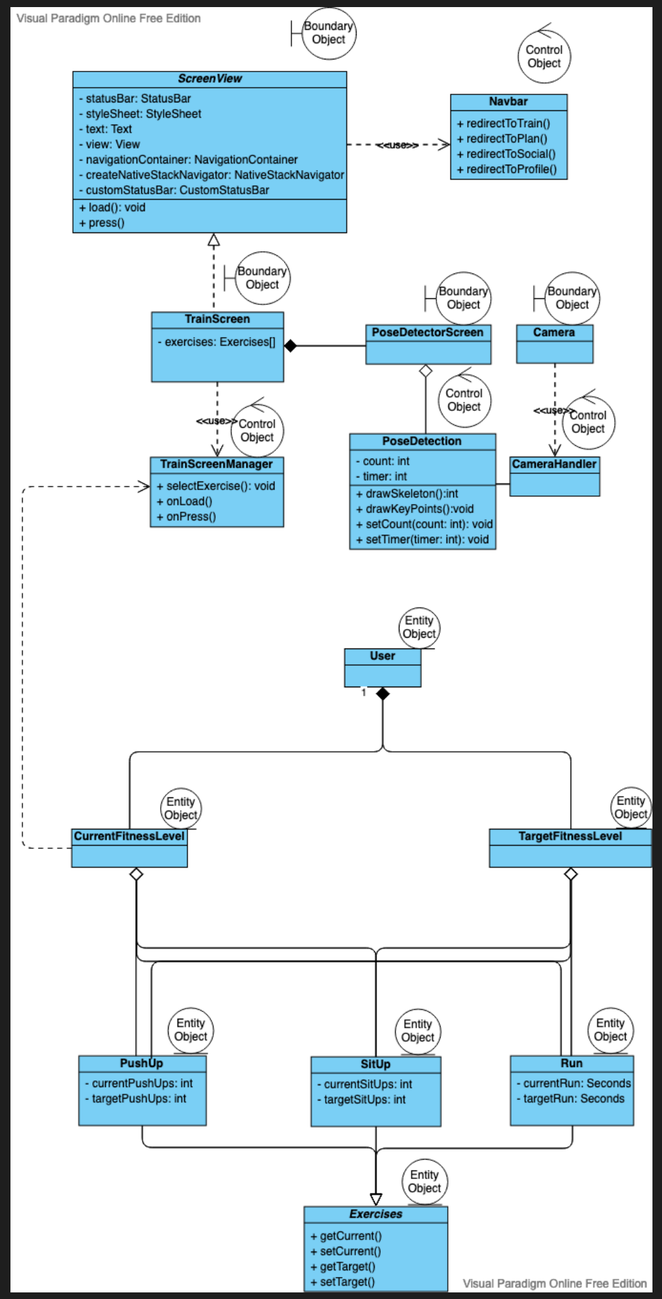


# Class Diagram



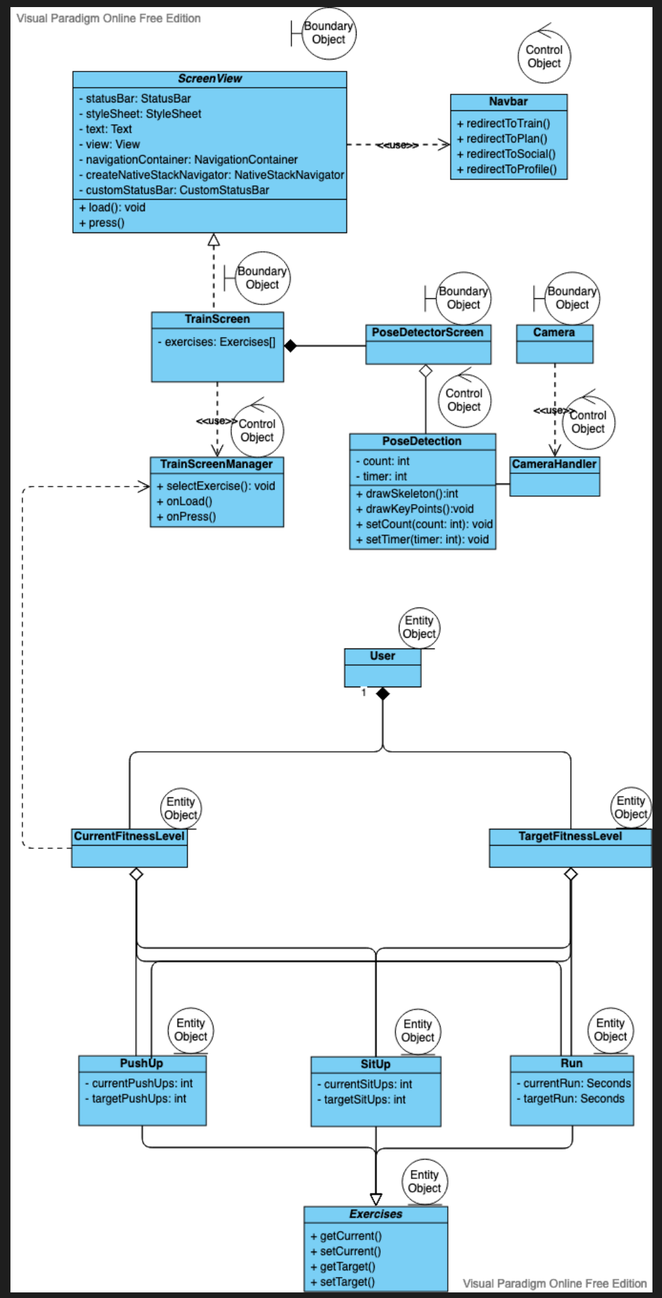

# Sequence Diagram






# Good Design Principles

- Single Responsibility Principle
  - The different components of count repetition, correct form are separated and kept isolated to reduce their complexity and their tendency to be altered .
- Open-Closed Principle
  - The different use cases of count repetition and correct form are kept as separate components in our implementation to encourage extensibility without modifying them.
  - Pose detection can be extended to be used for tracking of other exercises without the need to change its source code
- Liskov Substitution Principle
  - The 2 use-cases of analyse push-ups and sit-ups implement strategy pattern as they are interchangeable with one another as they implement design by contract




# Good Design Principles

- Interface Segregation Principle
  - Usage of many client specific interfaces instead of one general purpose interface
  - Classes do not depend on interfaces that they do not use
- Dependency Injection Principle
  - The pose detection module is complex. Thus, it is separated from the count push-up and sit-up through the TrainScreen module.

# Implementation

Overall Layout

```
<SafeAreaView style={{ flex: 1, justifyContent: "center" }}>
  <TimerDisplay timer={timer} />
  <StartButton setTimer={setTimer} timer={timer} setStart={setStart} />
  {result[0] ? <SubmitButton start={start} score={Math.floor(result[0])} activity={type} /> : null}
  {result[0] ? <ScoreDisplay score={Math.floor(result[0])} /> : null}
  {result[1] ? <DirectionDisplay direction={result[1]} /> : null}
  <Canvas ref={canvasRef} style={{ position: 'absolute', left: 0, top: 0, width: '100%', height: '100%',
  zIndex: 1000, backgroundColor: 'none' }} />
  <ModelCamera model={model} setPredictions={setPredictions} style={{ position: 'absolute', zIndex: 1 }} />
  {result ? <FeedbackDisplay feedback={result[2]} /> : null}
</SafeAreaView>
```

Form correction component:

```
function FeedbackDisplay(props) {
  let itemList = [];
  if (Object.entries(props)[0][1]) {
    for (const [key, value] of Object.entries(Object.entries(props)[0][1])) {
      if (value == false) {
        itemList.push(key + " ")
      }
    }
    return (
      <View style={styles.feedbackContainer}>
        <Text style={styles.feedbackText}>{itemList.length > 0 ? "Incorrect Posture" : "Correct Posture"}:</Text>
        <Text style={styles.feedbackText}>{itemList}</Text>
      </View>
    );
  }
}
```

# Implementation

Pose detector for push-ups or sit-ups:

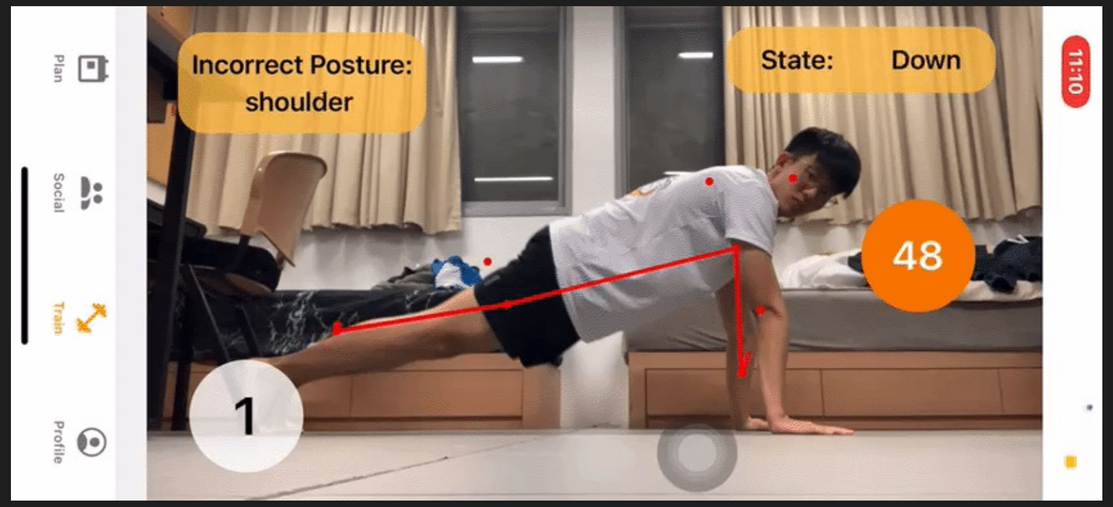
```
  start ? setResult(type == "PushUps" ? drawSkeletonPushUps(predictions["keypoints"], 0.5, ctx, 1, 1) :
  drawSkeletonSitUps(predictions["keypoints"], 0.1, ctx, 1, 1)) : null;
}
```

Repetition counter component:

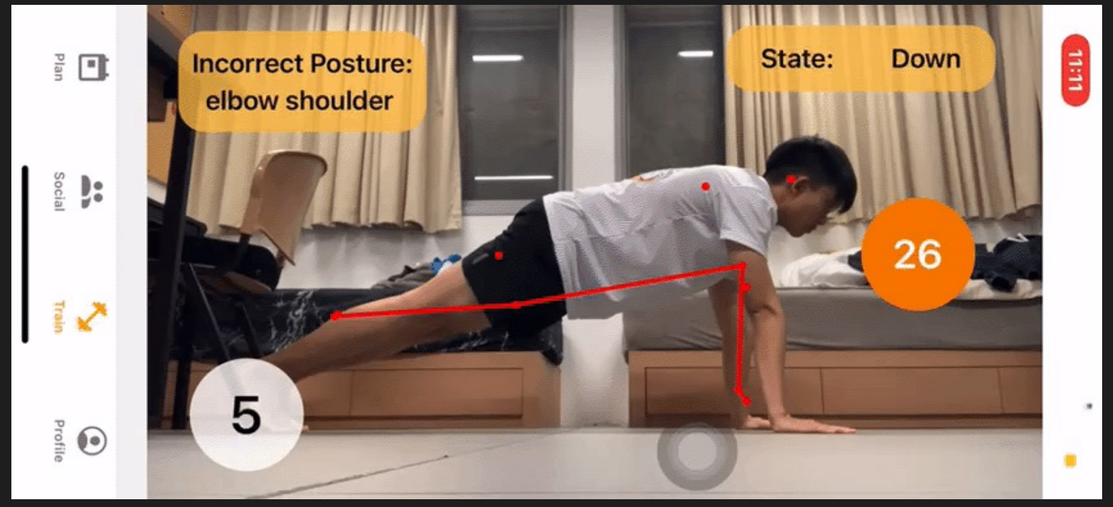
```
function ScoreDisplay({ score }) {
  return (
    <View style={styles.scoreContainer}>
      <Text style={styles.scoreText} >
        {score}
      </Text>
    </View>
  );
}
```

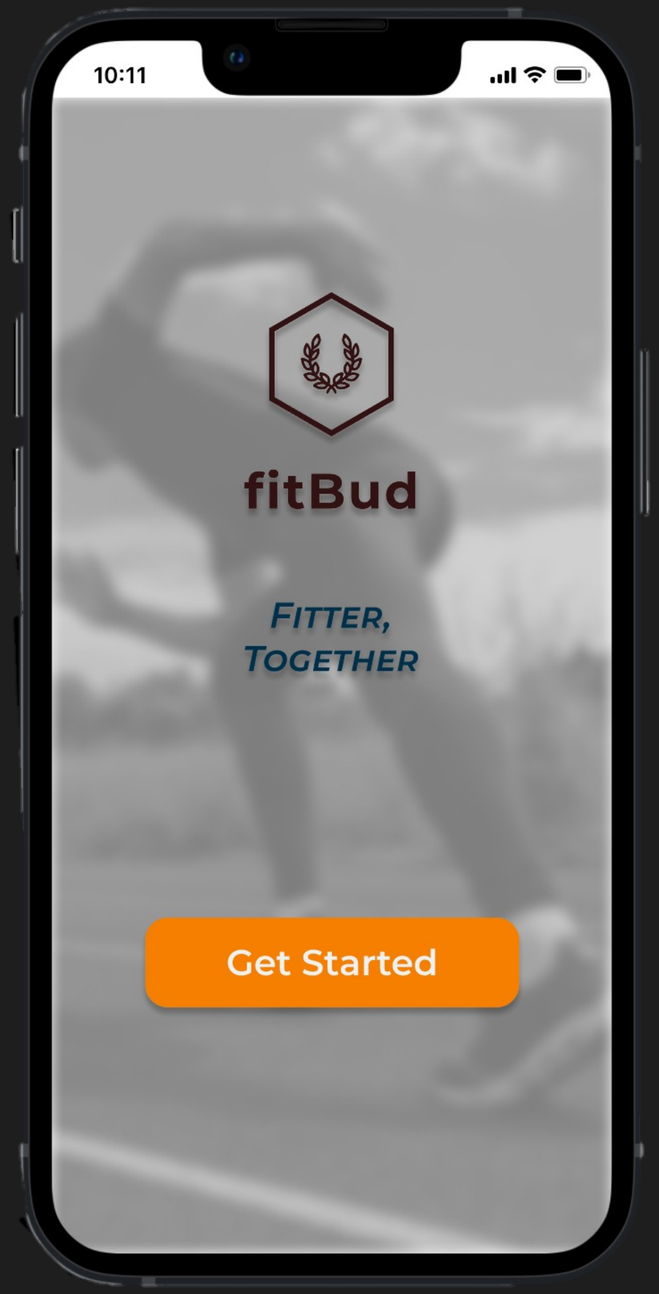
# Testing

## Successful Push-up Attempt:




## Unsuccessful Push-up Attempt:






# Thank You

*JOIN US TODAY TO BE FITTER TOGETHER*