



TECHNOLOGY PARK MALAYSIA

CT124-3-2-MAE

MOBILE APP ENGINEERING

GROUP ASSIGNMENT

GROUP: Group 11

Name	TP Number
Karen Guang Sing Qiao	TP067104
Chong Mun Hei	TP067333
Tang Chye Fong	TP080086

Table of Contents

1.0 Introduction.....	4
2.0 Problem Solving and Design	5
2.1 Problem Statement.....	6
2.2 Proposed Solution	7
2.3 Functionalities.....	9
2.3.1 User Functionalities	9
2.3.2 Coach Functionalities.....	10
2.3.3 Admin Functionalities.....	11
2.4 Use Case Diagram.....	12
2.5 System Architecture Design.....	13
2.6 Dynamic Data Management	15
2.6.1 Firebase.....	15
2.6.2 Database Sections	16
2.7 CRUD (Create, Read, Update, and Delete)	22
2.7.1 Users	22
2.7.2 Coaches.....	29
2.7.3 Admins	33
2.8 Validation	41
2.9 Users Permission.....	43
2.9 Wireframe	45
2.9.1 General.....	45
2.9.2 User.....	47
2.9.3 Coach	51
2.9.4 Admin.....	54
3.0 Use Manual	57
3.1 General.....	57

3.2 User	59
3.3 Coach	73
3.4 Admin.....	80
4.0 Automated System Testing.....	93
5.0 Conclusion	99
6.0 References.....	100
7.0 Appendix	101

1.0 Introduction

In today's fast-paced world, maintaining a healthy lifestyle has become a critical priority for individuals across the globe. However, the journey to physical fitness can often be overwhelming, with many people struggling to track their workouts, nutrition, and overall progress. Fitness apps have emerged as valuable tools in this domain, providing users with a platform to monitor their activities, receive guidance, and maintain motivation. Despite the availability of various fitness applications, there remains a gap in providing an all-in-one solution that integrates personalized workout routines, real-time progress tracking, educational resources, and community engagement.

"FITSPHERE" is a comprehensive mobile fitness tracker designed to address this gap by offering a holistic platform that caters to users, coaches, and administrators. Whether users are looking to follow a specific workout routine, track their weight and fitness progress, or gain insights from professional coaches, FITSPHERE offers a tailored experience for every individual. Through an intuitive interface and a wide range of interactive features, users can select workout categories, view instructional videos, record completed workouts, and engage with a supportive community of like-minded individuals.

The application offers distinct functionalities for three primary user roles: Users, Coaches, and Admins. Users can log in to access personalized workout videos, community posts, weight tracking features, and nutritional content. Coaches are empowered to create and share workout-related content, track user engagement with their posts, and provide valuable feedback. Admins oversee the entire app, managing user content, monitoring activity, and ensuring smooth app operation.

FITSPHERE leverages the power of Firebase as its backend, offering seamless integration between user data, workout videos, posts, feedback, and activities. This mobile application also provides social features, including community engagement, notifications, and feedback mechanisms, which encourage users to remain motivated and connected on their fitness journey.

This app not only assists users in achieving their fitness goals but also fosters a sense of community, where coaches can provide expert guidance, users can share their progress, and admins can maintain the app's integrity and growth. "FITSPHERE" is the ultimate solution for individuals seeking a comprehensive, interactive, and motivating fitness platform.

2.0 Problem Solving and Design

With the rise of fitness-conscious individuals, people are more inclined toward using fitness apps to track their progress, learn new exercises, and maintain a healthy lifestyle. However, many existing fitness apps often fail to deliver a comprehensive, user-friendly, and engaging experience. These apps typically focus on tracking exercises and calorie intake, without fostering community interaction, offering personalized feedback, or providing sufficient educational content. Moreover, there is often a lack of effective ways for coaches to interact with users, provide tailored advice, and track their progress.

Users of these apps often find it difficult to maintain motivation because they lack personalized coaching and interaction, which can be a game-changer in their fitness journey. Furthermore, many apps fail to provide a well-rounded experience by including essential features like nutrition education, progress tracking, social interaction, and the ability for users to connect with certified coaches who can provide guidance. As a result, users may struggle to stay engaged, lack accountability, or may even quit their fitness routines due to the absence of proper support.

From the coaches' perspective, many fitness apps fail to offer an efficient platform for them to connect with and manage users. Coaches often lack the tools to create and share content, track user progress, or even engage with users beyond simple notifications. They need a platform where they can post training resources, provide feedback on users' progress, and even interact with users in real time. Coaches also require tools to track their own posts' engagement, analyze the effectiveness of their content, and keep their clients motivated.

Admins of fitness platforms face similar challenges. They need to manage content effectively, monitor user activities, ensure a high-quality experience across the app, and keep track of user and coach interactions. Moreover, admins must be able to oversee user feedback, posts, and the overall success of the platform. Without proper tools, managing a fitness platform can be complex, leading to operational inefficiencies and lower user satisfaction.

2.1 Problem Statement

- Lack of a centralized platform for fitness tracking, community interaction, and professional coaching.
- Difficulty for users to find personalized coaching, educational resources, and feedback from fitness professionals.
- Insufficient interaction between users, coaches, and administrators.
- Inadequate tools for coaches to track users' progress and engage with them effectively.
- Complexity for admins to manage content, feedback, and user interactions on the platform.

2.2 Proposed Solution

To solve these problems, the "FITSPHERE" app will serve as an all-in-one fitness tracker and community platform that provides personalized workout guidance, tracks physical progress, and offers an engaging user experience. By dividing the app into three user roles—Users, Coaches, and Admins—each role is given the tools and functionalities necessary to address their specific needs, resulting in a comprehensive, user-friendly, and efficient platform.

1. **Comprehensive Fitness Tracking:** The app will allow users to select specific workout categories (e.g., tricep, shoulder, core, etc.) and access high-quality instructional workout videos. Each workout will be accompanied by detailed descriptions and video demonstrations. After completing a workout, users can log their activity, which will be recorded in the system. The app will also track and visualize users' weight progress over time and display it in graphs, giving users a clear picture of their journey.

2. **Community and Social Interaction:** To enhance engagement, users will have access to a community page where they can view posts made by coaches, including workout tips, motivational content, and nutrition advice. Users can interact with these posts by liking, commenting, and reporting inappropriate content. This feature fosters social interaction and provides users with a platform to learn from coaches and connect with other fitness enthusiasts.

3. **Personalized Coaching and Feedback:** The app will provide a search function, allowing users to find specific coaches based on their needs and preferences. Once users select a coach, they can view the coach's profile, including bio and workout videos, and even receive personalized workout plans. Coaches will also be able to provide feedback to users, track their progress, and monitor their workout performance through the app. This ensures that users receive tailored guidance that will help them stay motivated and achieve their fitness goals.

4. **Educational Content:** For users who want to learn more about nutrition, the app will provide an informational page covering topics such as macronutrients, healthy eating tips, and sample meal plans. This educational content will empower users to make informed choices about their diets and improve their overall health and fitness.

5. Admin Dashboard and Content Management: Admins will have access to a comprehensive management dashboard that allows them to:

- Monitor and approve posts made by coaches.
- Create and send announcements to users and coaches.
- Manage user and coach information, including profile edits and content moderation.
- View daily registration statistics and track the success of fitness activities.
- Respond to user and coach feedback in a timely manner, ensuring smooth communication.

6. Coach Empowerment: Coaches will have access to powerful tools that allow them to create posts, track user progress, and manage their own profiles. They can view analytics on how their posts are performing, track likes and comments and interact with users through feedback. Coaches can also use the app to build their own exercise plans, offer personalized advice, and promote their services to a broader audience.

7. Enhanced User Engagement: With features like personalized workout recommendations, tracking tools for weight and activity, educational content, and social interactions with coaches, the app will keep users engaged and motivated. The app's design ensures a seamless experience for users, from logging workouts to tracking their weight and following nutrition tips. Regular notifications and updates will encourage users to stay consistent with their workouts and progress.

8. Real-Time Data Updates: The app will be integrated with real-time data processing, allowing users to instantly log their activities, view real-time workout progress, and receive live feedback from coaches. This ensures that users and coaches have up-to-date information, leading to more accurate tracking and immediate adjustments to fitness plans.

9. Firebase as Backend: To support real-time interactions and ensure a scalable and secure environment, the app will leverage Firebase for its backend. Firebase provides a flexible and secure database structure for storing user data, activity logs, feedback, posts, announcements, and more. Firebase's real-time database capabilities will enable instant updates across all user roles, ensuring a seamless experience.

2.3 Functionalities

2.3.1 User Functionalities

Function	Description
Activity Tracking	User can choose what kind of workout he/she would like to do. After done for the workout, users can mark a workout as done. The system updates the user's activity status, logging the completion.
Community Interactions	Users can view the post created by coaches. Users can like, comment, and report posts created by coaches.
Report Tracking	Users can track their daily weight and workout stats in graphical form.
Nutrition Information	Users can access nutrition-related information.
Profile Management	Users can update their personal information, such as username, height, password, and secret word.
Search Coaches	Users can search for specific coaches and view their profiles.
Notifications of Announcements	Users can view the announcements posted by Admin
Feedback Management	Users can submit feedback to admin and view the response from the admin.

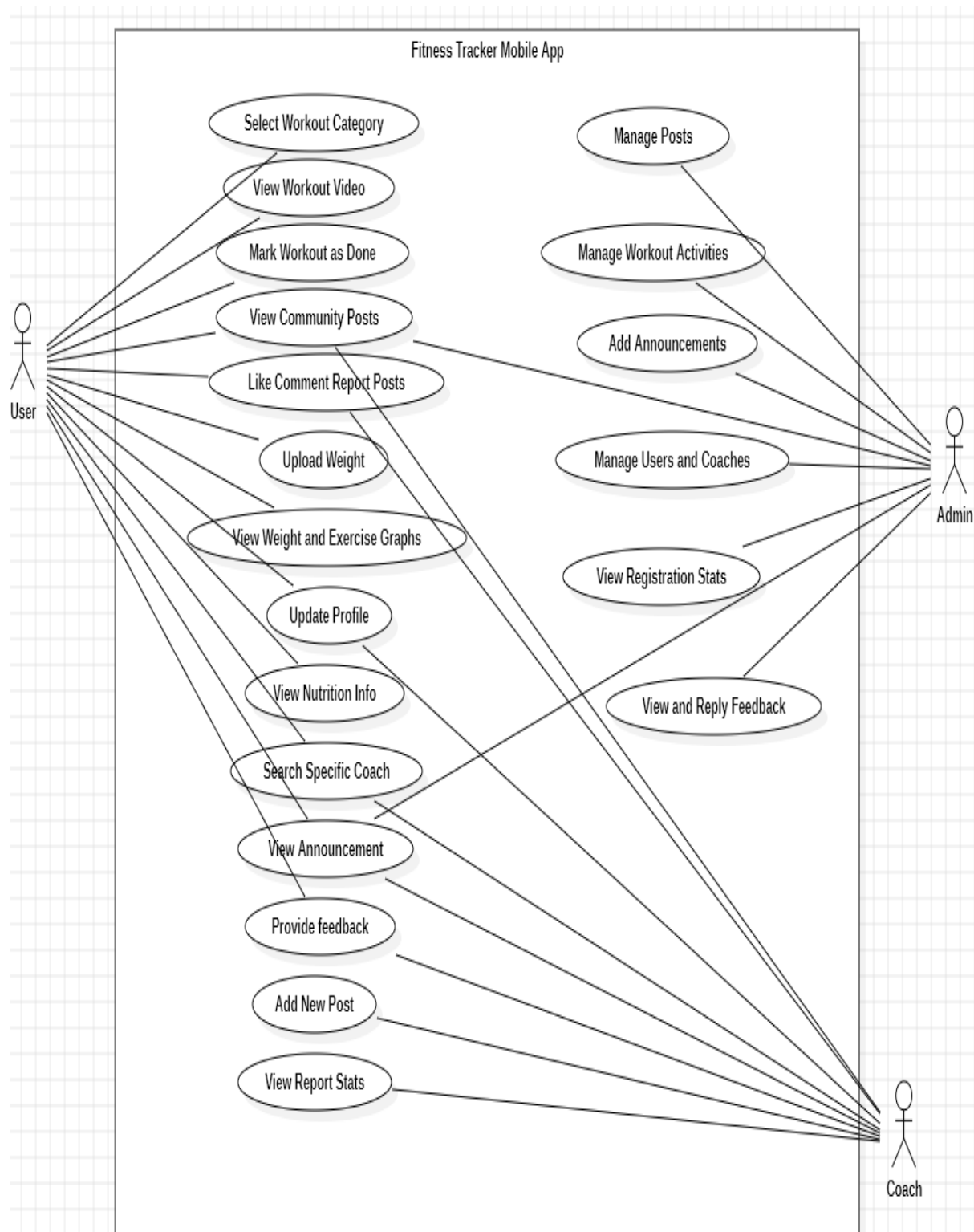
2.3.2 Coach Functionalities

Function	Description
Community Interactions	Coaches can view posts, comment on others' posts, and report posts that violate app guidelines.
Search Coaches	Coaches can search for specific coaches and view their profiles.
Post Creation	Coaches can create posts to share information with users.
Post Engagement	Coaches can track the number of likes and comments on their posts.
Profile Management	Coaches can update their profile, including username, bio, and password.
Notifications of Announcements	Coaches can view the announcements posted by Admin
Feedback Management	Coaches can post feedback to the admin and view their own feedback.

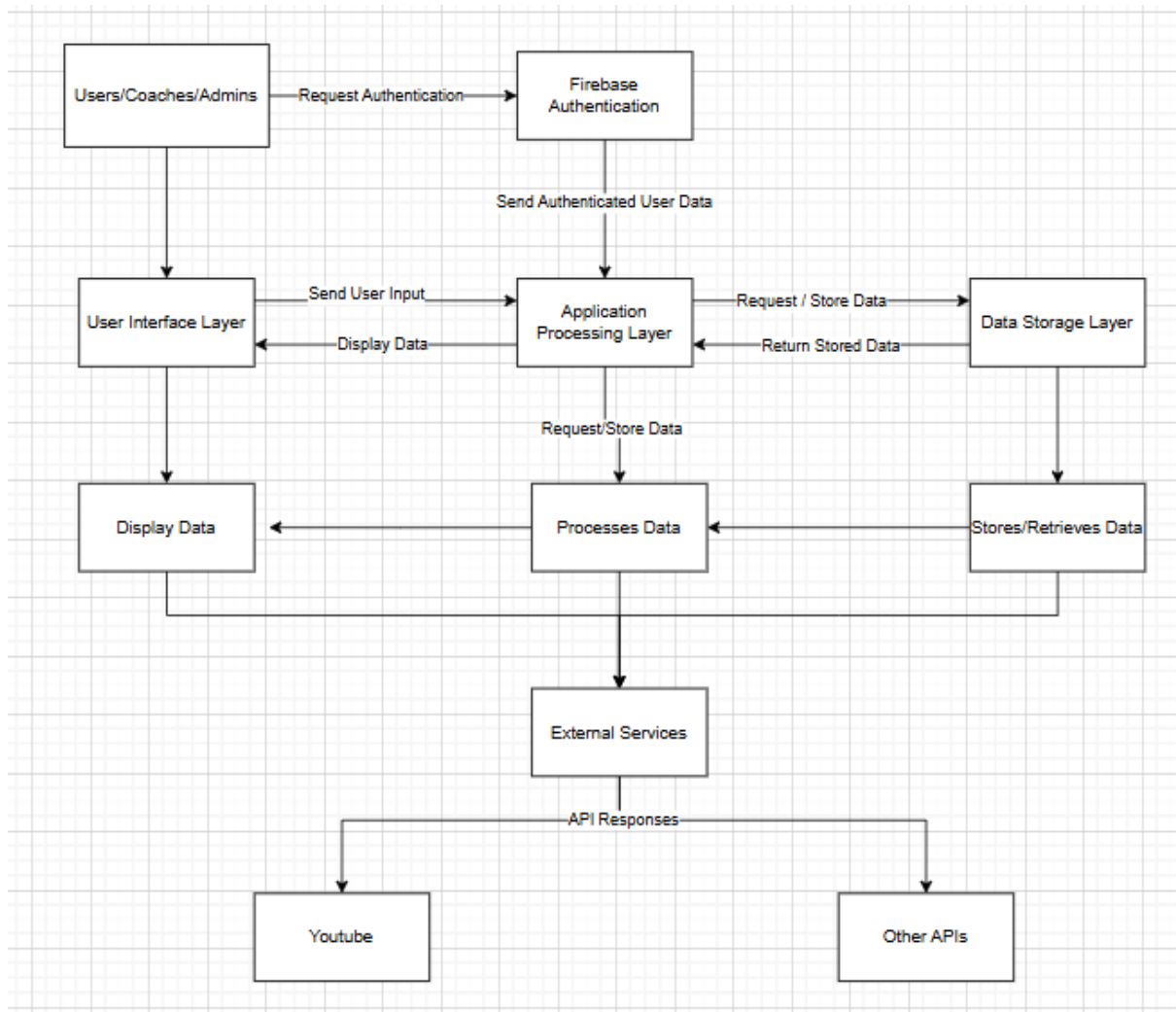
2.3.3 Admin Functionalities

Function	Description
Manage Posts	Admins can view, filter, and approve or disapprove posts created by coaches.
Manage Activities	Admins can add, edit, or delete workout activities (e.g., adding workout videos, editing details).
Create Announcements	Admins can create announcements that will be displayed to users and coaches.
Manage Users & Coaches	Admins can view, edit, delete, and manage information of users and coaches. Admins can also monitor activity.
Manage Feedback	Admins can view and reply to feedback submitted by users and coaches.
Notifications of Announcements	Admins can view the announcements.

2.4 Use Case Diagram



2.5 System Architecture Design



The "FITSPHERE" mobile app is designed as a cross-platform solution using Flutter, enabling it to operate seamlessly across both Android and iOS devices. The app features distinct user interfaces for three different roles: Users, Coaches, and Admins, each with role-specific functionalities. For instance, users can select workouts, participate in community discussions, track their fitness progress, access nutritional information, and manage their profiles. Coaches, on the other hand, can create posts, interact with other coaches, and access analytics on user engagement with their posts. Admins manage the platform by moderating content, managing user and coach accounts, and posting announcements.

At the core of the system architecture is Firebase, serving multiple roles including the backend database via Firebase Realtime Database, which houses all user data, activity logs, posts, and feedback. Firebase Authentication is employed for secure user authentication processes. The backend logic is handled through Firebase Cloud Functions, which manage tasks such as notification dispatch and data processing.

Videos for workouts are hosted on YouTube, integrating external media seamlessly into the user experience. Firebase Analytics is utilized to capture and analyze user interactions and app performance, facilitating continuous improvement of the platform. This integrated approach ensures that FITSPHERE offers a robust, responsive, and user-centric experience, making fitness tracking and community interaction accessible and engaging for all users.

2.6 Dynamic Data Management

2.6.1 Firebase

The backend of the FITSPHERE mobile app is hosted using the Firebase Realtime Database. This structure supports three types of users: Users, Coaches, and Admins. Each user role has different capabilities, and the database structure is designed to manage activities, feedback, announcements, posts, user profiles, weight tracking, workout activities, and more.

```
https://fitness-app-490b6-default-rtdb.asia-southeast1.firebaseio.com/  
├── activity  
├── announcement  
├── feedback  
├── post  
└── user
```

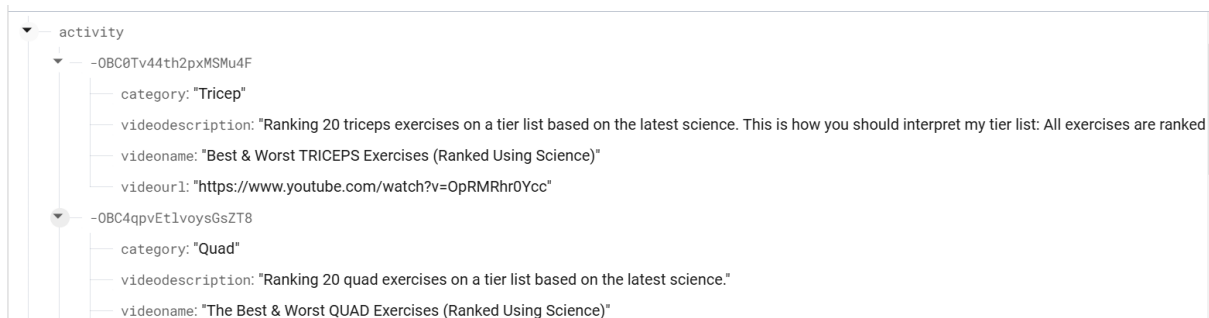
The data is divided into major sections:

- **activity:** This stores workout activities for users, including categories (e.g., Triceps, Quads, Shoulders), descriptions, and the video URL for workout tutorials.
- **announcement:** Stores all announcements made by the admin for coaches and users.
- **feedback:** Stores the feedback submitted by users and coaches, including responses from admins.
- **post:** Stores posts made by coaches, including likes, comments, and reports.
- **user:** Stores the information about users, coaches, and admins, including their profile, activity logs, and weight tracking.

2.6.2 Database Sections

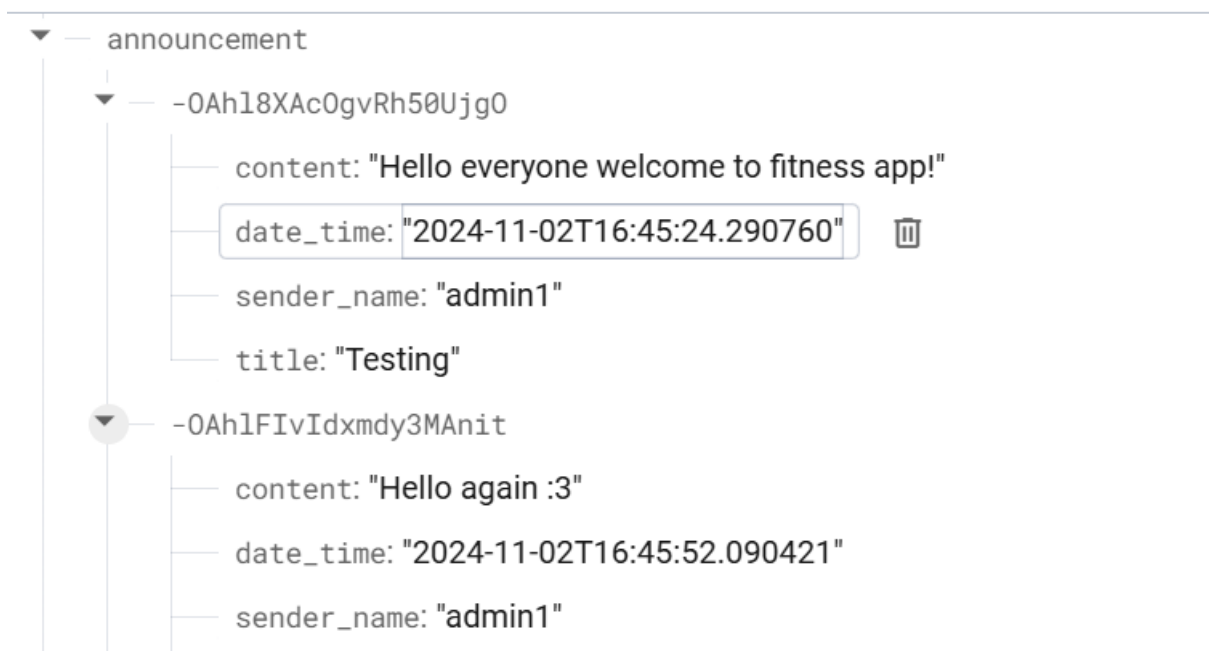
2.6.2.1 Activity

This section holds workout activities for users, categorized by different muscle groups or workout types. Each activity contains information such as a video name, description, category, and the URL to the video.



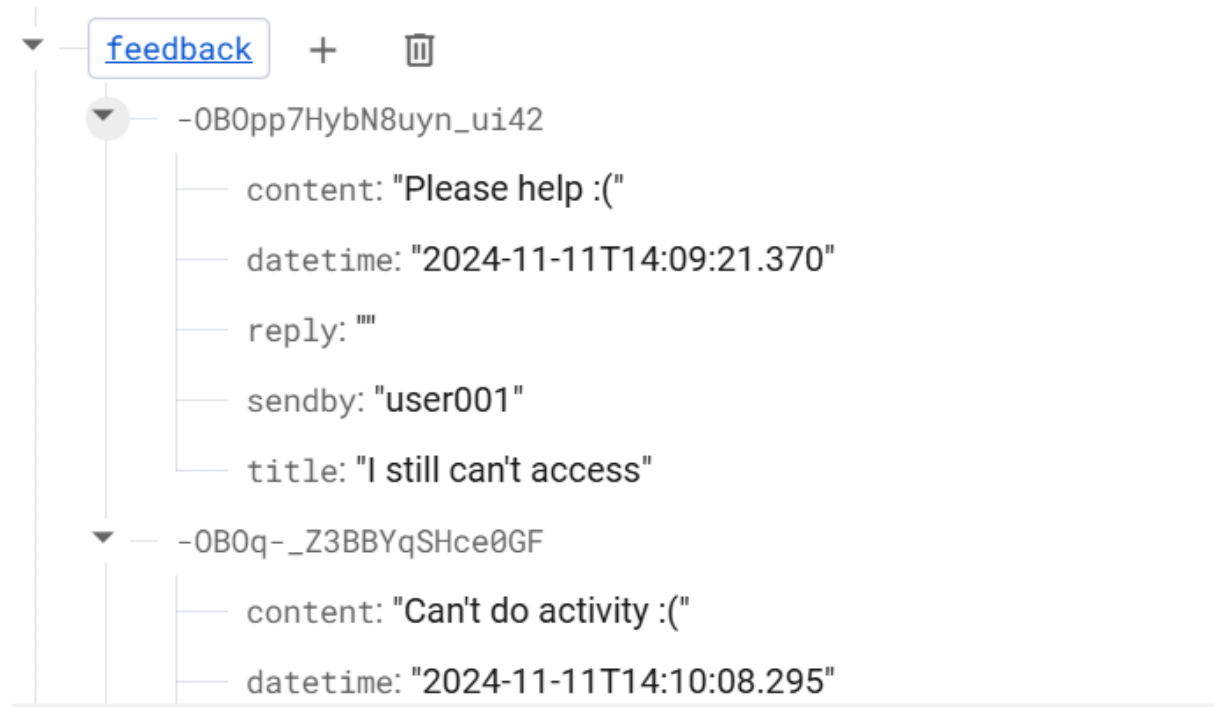
2.6.2.2 Announcement

This section contains announcements made by admins. It stores the announcement's content, title, date/time, and the sender's name (admin).



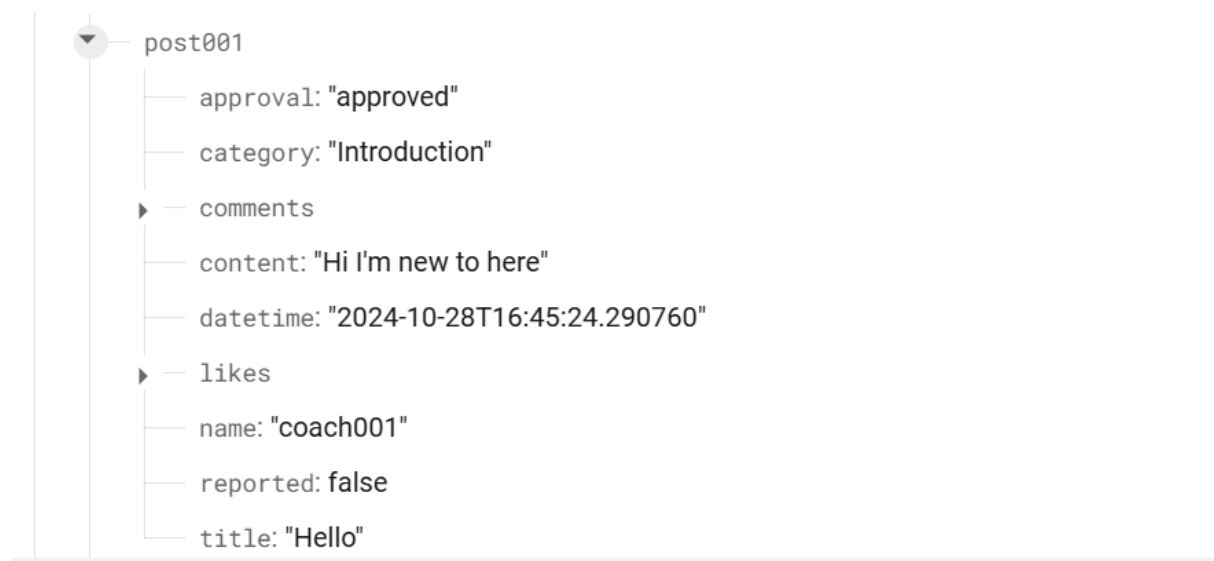
2.6.2.3 Feedback

This section holds user feedback. Each feedback includes content, a timestamp, a response from an admin (if available), and the user/coach ID who submitted the feedback.



2.6.2.4 Post

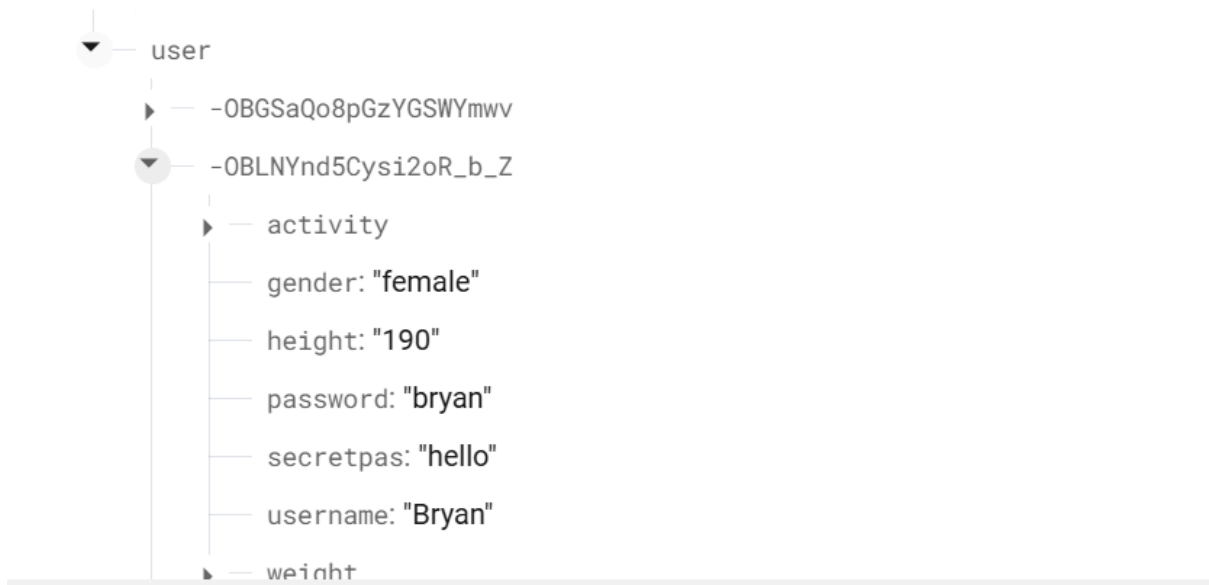
Coaches can create posts that users can view. Each post contains the title, content, comments, likes, and whether the post is approved or reported. Admins have the ability to approve or disapprove posts.



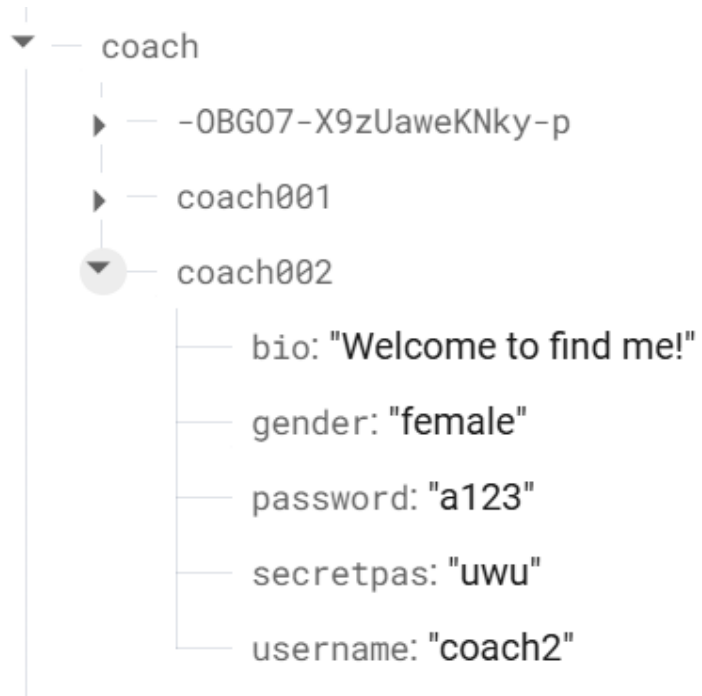
2.6.2.5 User

This section contains all the data for each user, coach, and admin. Each user has their profile information (username, gender, height), activity records (workouts completed), weight tracking, and feedback submissions. Coaches and admins have similar data, but coaches can also create posts, and admins can approve or disapprove posts.

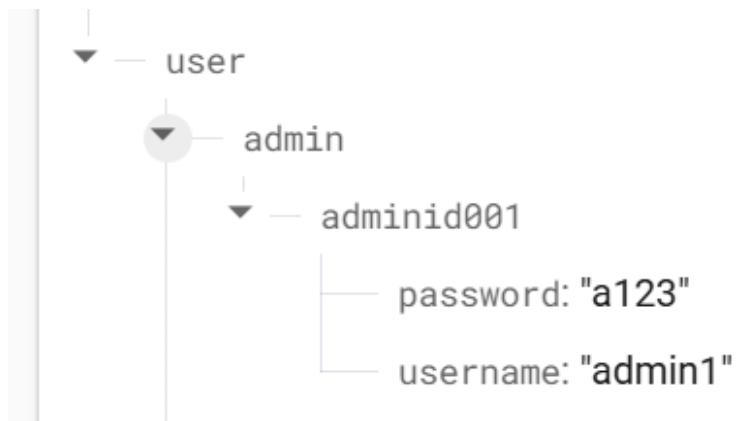
For Users:



For Coaches:



For Admins:



2.7 CRUD (Create, Read, Update, and Delete)

2.7.1 Users

2.7.1.1 Create

2.7.1.1.1 Create Profile

```
        controller: _userheightController,  
        decoration: const InputDecoration(  
          labelText: 'Height',  
          border: OutlineInputBorder(),  
        ), // InputDecoration  
      ), // TextField  
      const SizedBox(height: 20),  
      TextField(  
        controller: _userpasswordController,  
        decoration: const InputDecoration(  
          labelText: 'Password',  
          border: OutlineInputBorder(),  
        ), // InputDecoration  
      ), // TextField  
      const SizedBox(height: 20),  
      TextField(  
        controller: _usersecretpasController,  
        decoration: const InputDecoration(  
          labelText: 'Secret Word',  
          border: OutlineInputBorder(),  
        ), // InputDecoration  
      ), // TextField  
      const SizedBox(height: 20),  
      ElevatedButton(  
        onPressed: () async {  
          bool isSuccess = await saveNewUser(  
            null, _usernameController.text,  
            _isMale ? 'male' : 'female', _userheightController.text,  
            userpasswordController.text, usersecretpasController.text
```

This code provides a form with TextField inputs for the user's height, password, and secret word, each with labeled InputDecoration and a border outline. An ElevatedButton triggers the saveNewUser function, which saves the user information to a database. If the save operation is successful, a success dialog (_showSuccessDialog) is displayed. The user's gender is set based on the _isMale boolean. Users can create a profile by providing basic information such as username, height, password, and secret word.

2.7.1.1.2 Create Workout Activity

```
// Function to update user activity in Firebase
Future<void> updateUserActivity(String userId) async {
  final currentDate =
    DateFormat('yyyy-MM-dd').format(DateTime.now()); // Get current date
  final userActivityPath =
    'user/user/$userId/activity'; // Path to user's activities

  // Fetch existing activities
  final response =
    await http.get(Uri.parse('${databaseURL}$userActivityPath.json'));

  if (response.statusCode == 200) {
    final Map<String, dynamic>? activitiesData = json.decode(response.body);

    // If there's no activity data yet (for new users)
    if (activitiesData == null || activitiesData.isEmpty) {
      // Create a new activity for the user with the current date and set count
      final newActivityKey =
        'activity001_$userId'; // Generate a key for the new activity
      await http.put(
        Uri.parse('${databaseURL}$userActivityPath/$newActivityKey.json'),
        body: json.encode({
          'date': currentDate,
          'number': 1, // Start with 1 for the new activity
        }),
      );
      return; // Exit the function as we have added the initial activity
    }
  }
}
```

This function, `updateUserActivity`, updates a user's activity data by checking if there's an existing entry for the current date in the database. If found, it increments the activity count; if not, it creates a new activity entry with a count of 1, ensuring activity data is updated or initialized based on daily usage. Users can create a workout activity record by completing a workout (this updates their activity data in the backend).

2.7.1.1.3 Create Feedback

```
@override
void initState() {
  super.initState();
  fetchUserFeedback();
}

Future<void> fetchUserFeedback() async {
  List<Map<String, dynamic>> feedback = await getUserFeedback(widget.userId);
  setState(() {
    feedbackList = feedback;
  });
}

Future<void> addFeedback(String title, String content) async {
  await addNewFeedback(widget.userId, title, content);
  fetchUserFeedback(); // Refresh feedback list
}
```

The `fetchCoachFeedback` function retrieves a list of feedback for a specific coach using `getCoachFeedback` with the `userId`, and updates `feedbackList` with the data, refreshing the UI with `setState`. The `addFeedback` function adds new feedback by calling `addNewFeedback` with the coach's `userId`, `title`, and `content`, then calls `fetchCoachFeedback` to update the feedback list immediately after. Users can create feedback to administer.

2.7.1.2 Read

2.7.1.2.1 View Profile

```
@override
void initState() {
  super.initState();
  _fetchUsername();
  _fetchUsergender();
  _fetchUserheight();
  _fetchUserpassword();
  _fetchUsersecretpas();
  _usernameController = TextEditingController(text: "");
  _userheightController = TextEditingController(text: "");
  _userpasswordController = TextEditingController(text: "");
  _usersecretpasController = TextEditingController(text: "");
  _isMale = true;
}
```

These functions (`_fetchUsername`, `_fetchUsergender`, `_fetchUserheight`, `_fetchUserpassword`, and `_fetchUsersecretpas`) each retrieve specific user details (username, gender, height, password, and secret pass) from the database using the `searchUser` function. They update the respective controllers and variables for displaying this information in the UI, and then trigger a `setState` to refresh the display. Users can view their own profile and activity records.

2.7.1.2.2 View Workout Videos

```
@override
void initState() {
  super.initState();
  _fetchVideos(); // Fetch videos based on category
}

Future<void> _fetchVideos() async {
  List<Map<String, dynamic>> videos =
    await fetchWorkoutVideos(widget.category);
  setState(() {
    _videos = videos;
  });
}
```

In this code, the `initState` method is overridden to call `_fetchVideos` as soon as the widget is initialized, which asynchronously retrieves workout videos for a specific category using `fetchWorkoutVideos`. The retrieved videos are then stored in the `_videos` list, and `setState` is called to update the UI with the fetched data. Users can view workout videos on the workout video page.

2.7.1.2.3 View Post

```
@override
void initState() {
  super.initState();
  fetchPosts();
}

Future<void> fetchPosts() async {
  final fetchedPosts = await getApprovedPosts();
  setState(() {
    posts = fetchedPosts;
  });
}

Future<void> _toggleLike(String postId, String userId, bool isLiked) async {
  await toggleLike(postId, userId, isLiked);
  fetchPosts();
}

Future<void> _addComment(String postId, String userId, String content) async {
  await addComment(postId, userId, content);
  fetchPosts();
}

void _showCommentDialog(String postId) {
  final TextEditingController commentController = TextEditingController();
  showDialog(
    context: context,
```

This code defines functions for managing posts and interactions. The `fetchPosts` function retrieves approved posts and updates the UI, while `_toggleLike` toggles a like status for a post and then refreshes the posts. The `_addComment` function adds a comment to a specific post and refreshes the list. The `_showCommentDialog` function displays a dialog box with a text field for users to input a comment, calling `_addComment` and closing the dialog once the comment is submitted. Users can view posts from coaches on the community page.

2.7.1.2.4 View Daily Weight Records and Workout Progress

```
body: Column(  
  children: [  
    // TabBar placed under the AppBar  
    TabBar(  
      controller: _tabController,  
      tabs: const [  
        Tab(text: 'Weight'),  
        Tab(text: 'Exercise'),  
      ],  
    ), // TabBar  
    // TabBarView to show content of selected tab  
    Expanded(  
      child: TabBarView(  
        controller: _tabController,  
        children: [  
          UserWeightPage(userId: widget.userId),  
          UserExercisePage(userId: widget.userId),  
        ],  
      ), // TabBarView  
    ), // Expanded  
  ],  
), // Column
```

This code defines a layout with a `TabBar` and `TabBarView` that provides two tabs, "Weight" and "Exercise." The `TabBar` allows the user to switch between tabs, while the `TabBarView` displays the content for each selected tab: `UserWeightPage` and `UserExercisePage`, each initialized with a `userId` to display user-specific data. The `Expanded` widget ensures the `TabBarView` occupies the available space below the `TabBar`. Users can view daily weight records and workout progress (displayed as graphs) in the report page.

2.7.1.2.5 View Nutrition Information

```
class _UserNutritionState extends State<UserNutrition> {
  Widget build(BuildContext context) {
    // AppBar
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: ListView(
        children: [
          const Text(
            'Nutrition Knowledge',
            style: TextStyle(fontSize: 26, fontWeight: FontWeight.bold),
            textAlign: TextAlign.center,
          ), // Text
          const SizedBox(height: 20),
          const Text(
            'Understanding Nutrition',
            style: TextStyle(fontSize: 22, fontWeight: FontWeight.bold),
          ), // Text
          const Text(
            'Nutrition is essential for maintaining good health and well-being. '
            'It involves the processes by which our bodies use the food we consume to support growth, energy, and overall health.',
            style: TextStyle(fontSize: 16),
          ), // Text
          const SizedBox(height: 20),
          Image.asset(
            'lib/assets/nutrition_image1.jpeg', // Your first nutrition image
            height: 200,
            fit: BoxFit.cover,
          ), // Image.asset
          const SizedBox(height: 20),
          const Text(
```

This code creates a scrollable `ListView` with various educational sections on nutrition, including text content on nutrition basics, macronutrients, healthy eating tips, and a sample meal plan. It includes supporting images and a YouTube video player for interactive learning. Each section has titles and detailed descriptions, with formatted `Text.rich` for lists and embedded assets to visually enrich the user's understanding of nutrition. Users can view nutrition information (e.g., nutrition basics, macronutrients, sample meal plans).

2.7.1.3 Update

2.7.1.3.1 Update Profile Information

```
Future<void> saveNewUser(String? userId, String username, String gender, String height, String password, String secretpas) async{
  final url = Uri.parse('${databaseURL}/user/user/$userId.json');

  final userData = {
    'username': username,
    'gender': gender,
    'height': height,
    'password': password,
    'secretpas': secretpas
  };

  try {
    await http.post(url, body: json.encode(userData));
  } catch (error){
    throw Exception('$error');
  }
}
```

This code defines an asynchronous function, `saveNewUser`, which sends a POST request to save a new user's data (username, gender, height, password, and a secret pass) to a specified database URL, using the provided `userId` as the unique identifier. Users can update their profile information such as username, password, and secret word.

2.7.1.3.2 Update Weight

```
Future<void> _loadWeightData() async {  
  List<Map<String, dynamic>> data = await fetchWeightData(widget.userId);  
  // Filter to keep only the latest entry per date  
  Map<String, Map<String, dynamic>> latestEntries = {};  
  for (var entry in data) {  
    String dateKey = DateFormat('yyyy-MM-dd').format(DateTime.parse(entry['date']));  
    DateTime entryDate = DateTime.parse(entry['date']);  
    if (!latestEntries.containsKey(dateKey) || entryDate.isAfter(DateTime.parse(latestEntries[dateKey]['date']))) {  
      latestEntries[dateKey] = entry;  
    }  
  }  
  setState(() {  
    // Filter out entries with weight values that don't make sense (like 1730)  
    _weightData = latestEntries.values.where((entry) => entry['value'] < 300).toList();  
    _filterDataByWeek();  
    _calculateMinMaxWeight();  
  });  
}
```

This asynchronous function `_loadWeightData` retrieves weight data for a user and processes it to retain only the latest entry for each date. It filters out entries with implausible weight values (over 300) and stores the cleaned data in `_weightData`. After updating the state, it also calls `_filterDataByWeek` and `_calculateMinMaxWeight` to further process the data by week and determine minimum and maximum weights for display. Users can update their weight entries every day.

2.7.1.3.3 Update Track Progress

```
// Function to update user activity in Firebase  
Future<void> updateUserActivity(String userId) async {  
  final currentDate =  
    DateFormat('yyyy-MM-dd').format(DateTime.now()); // Get current date  
  final userActivityPath =  
    'user/user/$userId/activity'; // Path to user's activities  
  
  // Fetch existing activities  
  final response =  
    await http.get(Uri.parse('${databaseURL}$userActivityPath.json'));  
  
  if (response.statusCode == 200) {  
    final Map<String, dynamic>? activitiesData = json.decode(response.body);  
  
    // If there's no activity data yet (for new users)  
    if (activitiesData == null || activitiesData.isEmpty) {  
      // Create a new activity for the user with the current date and set count  
      final newActivityKey =  
        'activity001_$userId'; // Generate a key for the new activity  
      await http.put(  
        Uri.parse('${databaseURL}$userActivityPath/$newActivityKey.json'),  
        body: json.encode({  
          'date': currentDate,  
          'number': 1, // Start with 1 for the new activity  
        }),  
      );  
      return; // Exit the function as we have added the initial activity  
    }  
  }  
}
```

This function, `updateUserActivity`, updates a user's activity data by checking if there's an existing entry for the current date in the database. If found, it increments the activity count; if not, it creates a new activity entry with a count of 1, ensuring activity data is updated or

initialized based on daily usage. Users can create a workout activity record by completing a workout (this updates their activity data in the backend). Users can select and change workout categories and track progress.

2.7.2 Coaches

2.7.2.1 Create

2.7.2.1.1 Create Post

```
void addNewPost() async {
  if (_titleController.text.isNotEmpty && _contentController.text.isNotEmpty) {
    final response = await addPostToDatabase(
      widget.userId,
      _titleController.text,
      _contentController.text,
    );

    if (response) {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Post added successfully!')),
      );
      _titleController.clear();
      _contentController.clear();
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Failed to add post.')),
      );
    }
  } else {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('Please fill in both title and content.')),
    );
  }

  Navigator.pushNamed(context, '/coach_home', arguments: {'userId': widget.userId});
}
```

The addNewPost function checks if the title and content fields are filled, then attempts to add a new post to the database using addPostToDatabase. If successful, it shows a success message and clears the input fields; otherwise, it displays an error message. If fields are empty, a prompt asks the user to complete them. Finally, it navigates back to the coach's home page, passing the userId as an argument. Coaches can create and post content on the community page.

2.7.2.1.2 Add Comment and Like to Post

```
Future<void> _toggleLike(String postId, String userId, bool isLiked) async {
  await toggleLike(postId, userId, isLiked);
  fetchPosts();
}

Future<void> _addComment(String postId, String userId, String content) async {
  await addComment(postId, userId, content);
  fetchPosts();
}
```

The `_toggleLike` function asynchronously updates the like status of a post by calling `toggleLike` with the `postId`, `userId`, and `isLiked` status, then refreshes the list of posts by calling `fetchPosts`. Similarly, `_addComment` adds a comment to the specified post using `addComment`, also followed by `fetchPosts` to update the displayed post data. Coaches can add comments and like for community engagement or advice.

2.7.2.1.3 Create Profile

```
@override
void initState() {
  super.initState();
  _fetchUsername();
  _fetchUsergender();
  _fetchUserbio();
  _fetchUserpassword();
  _fetchUsersecretpas();
  _usernameController = TextEditingController(text: "");
  _userbioController = TextEditingController(text: "");
  _userpasswordController = TextEditingController(text: "");
  _usersecretpasController = TextEditingController(text: "");
  _isMale = true;
}
```

These functions retrieve specific user details (username, gender, bio, password, and secret pass) asynchronously from the database using the `searchUser` function, update corresponding text controllers for display, and call `setState` to refresh the UI with the fetched data. Gender is set based on whether the retrieved value is 'male' or not. Coaches can create a profile with bio, username, password, and secret word.

2.7.2.1.4 Create Feedback

```

@Override
void initState() {
    super.initState();
    fetchCoachFeedback();
}

Future<void> fetchCoachFeedback() async {
    List<Map<String, dynamic>> feedback = await getCoachFeedback(widget.userId);
    setState(() {
        feedbackList = feedback;
    });
}

Future<void> addFeedback(String title, String content) async {
    await addNewFeedback(widget.userId, title, content);
    fetchCoachFeedback(); // Refresh feedback list
}

```

The `fetchCoachFeedback` function retrieves a list of feedback for a specific coach using `getCoachFeedback` with the `userId`, and updates `feedbackList` with the data, refreshing the UI with `setState`. The `addFeedback` function adds new feedback by calling `addNewFeedback` with the coach's `userId`, `title`, and `content`, then calls `fetchCoachFeedback` to update the feedback list immediately after. Coaches can create feedback to administer.

2.7.2.2 Read

2.7.2.2.1 View Posts

```

@Override
void initState() {
    super.initState();
    fetchPosts();
}

Future<void> fetchPosts() async {
    final fetchedPosts = await getApprovedPosts();
    setState(() {
        posts = fetchedPosts;
    });
}

```

The `fetchPosts` function asynchronously retrieves a list of approved posts using `getApprovedPosts` and updates the `posts` variable with the fetched data, calling `setState` to refresh the UI and display the latest posts. Coaches can view posts from other coaches.

2.7.2.2.2 View Report of Posts

```
@override
void initState() {
  super.initState();
  totalLikesCommentsFuture = calculateTotalLikesAndComments(widget.userId);
  coachPostsFuture = fetchCoachPosts(widget.userId);
}

void sortPosts(String option) {
  setState(() {
    selectedSortOption = option;

    if (option == 'Order by likes') {
      coachPosts = sortPostsByLikes(coachPosts);
    } else if (option == 'Order by comments') {
      coachPosts = sortPostsByComments(coachPosts);
    }
  });
}
```

The sortPosts function updates the sorting of coachPosts based on the selected option. When called, it updates selectedSortOption and reorders coachPosts by either likes or comments using sortPostsByLikes or sortPostsByComments accordingly, refreshing the UI with setState to display the sorted posts. Coaches can view their own posts, including likes and comments.

2.7.2.2.3 Search Specific Coaches and View Profiles

```
@override
void dispose() {
  _searchController.dispose();
  super.dispose();
}

Future<void> _searchCoach() async {
  if (_searchController.text.isEmpty) return;

  List<Map<String, dynamic>> results =
    await searchCoaches(_searchController.text);
  setState(() {
    _searchResults = results;
  });
}
```

The _searchCoach function performs a search for coaches based on the input in _searchController. If the input is not empty, it retrieves matching results using searchCoaches, updates _searchResults with the search outcome, and refreshes the UI using setState to display the results. Coaches can search for specific coaches and view their profiles.

2.7.2.3 Update

2.7.2.3.1 Update Profile Details

```
@override
void initState() {
  super.initState();
  _fetchUsername();
  _fetchUsergender();
  _fetchUserbio();
  _fetchUserpassword();
  _fetchUsersecretpas();
  _usernameController = TextEditingController(text: "");
  _userbioController = TextEditingController(text: "");
  _userpasswordController = TextEditingController(text: "");
  _usersecretpasController = TextEditingController(text: "");
  _isMale = true;
}
```

Each of these functions asynchronously retrieves a specific user attribute (username, gender, bio, password, and secret pass) from the database using `searchUser`, then updates corresponding `TextEditingController` fields and local variables. `setState` is called to refresh the UI, with `_isMale` set to true or false based on the gender value. Coaches can update their profile details such as bio, username, password, and secret word.

2.7.3 Admins

2.7.3.1 Create

2.7.3.1.1 Create Announcements

```
void addAnnouncement() {
  final title = _titleController.text;
  final content = _contentController.text;

  if (title.isNotEmpty && content.isNotEmpty) {
    final dateTime = DateTime.now().toIso8601String();
    saveAnnouncement(widget.userId, dateTime, title, content);

    _titleController.clear();
    _contentController.clear();
    fetchAnnouncements();
  }
}
```

The `addAnnouncement` function captures the title and content from the input controllers, checks if both are non-empty, and then generates a timestamp (`dateTime`) for the announcement. It saves the announcement with `saveAnnouncement`, clears the input fields, and calls

fetchAnnouncements to refresh the displayed list of announcements. Admins can create announcements to send out to users and coaches.

2.7.3.1.2 Add New Workout Activities

```
// Function to add a new activity
Future<void> addActivity(Map<String, dynamic> activity) async {
  final response = await http.post(
    Uri.parse('${databaseURL}activity.json'),
    body: json.encode(activity),
  );

  if (response.statusCode != 200) {
    throw Exception('Failed to add activity');
  }
}
```

The addActivity function sends a POST request to add a new activity to the database by encoding the activity data as JSON and posting it to the specified activity.json endpoint. If the response status is not 200 (indicating success), it throws an exception indicating the failure to add the activity. Admins can add new workout activities (e.g., video name, description, and category).

2.7.3.2 Read

2.7.3.2.1 Read Feedback

```
// AppBar
body: Center(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      TabBar(
        controller: _tabController,
        tabs: const [
          Tab(text: 'Pending'),
          Tab(text: 'Replied'),
        ],
      ), // TabBar
      Expanded(
        child: TabBarView(
          controller: _tabController,
          children: const [
            AdminPendingPage(),
            AdminRepliedPage(),
          ],
        ), // TabBarView
      ), // Expanded
    ],
  ), // Column
), // Center
```

This code creates a centered layout with a TabBar and TabBarView to manage two tabs, "Pending" and "Replied." The TabBar allows switching between the tabs, while the TabBarView displays either AdminPendingPage or AdminRepliedPage, depending on the selected tab. The Expanded widget ensures that TabBarView fills the remaining vertical space within the centered column layout. Admins can read feedback from users and coaches.

2.7.3.2.2 Read Post

```
body: Center(  
  child: Column(  
    mainAxisAlignment: MainAxisAlignment.center,  
    children: [  
      TabBar(  
        controller: _tabController,  
        tabs: const [  
          Tab(text: 'Approved'),  
          Tab(text: 'Disapproved'),  
          Tab(text: 'Reported'),  
        ],  
      ), // TabBar  
      Expanded(  
        child: TabBarView(  
          controller: _tabController,  
          children: [  
            AdminHomeApproved(userId: widget.userId),  
            AdminHomeDisapproved(userId: widget.userId),  
            AdminHomeReported(userId: widget.userId),  
          ],  
        ), // TabBarView  
      ), // Expanded  
    ],  
  ), // Column  
, // Center
```

This code defines a centered layout with a `TabBar` and `TabBarView` containing three tabs: "Approved," "Disapproved," and "Reported." The `TabBar` allows navigation between these tabs, while the `TabBarView` displays different content based on the selected tab: `AdminHomeApproved`, `AdminHomeDisapproved`, or `AdminHomeReported`, each receiving `userId` as an argument. The `Expanded` widget ensures the `TabBarView` fills the available space in the column layout. Admins can read posts submitted by coaches and decide whether to approve or disapprove them.

2.7.3.2.3 View Statistics

```
), // AppBar  
body: Column(  
  children: [  
    // TabBar placed under the AppBar  
    TabBar(  
      controller: _tabController,  
      tabs: const [  
        Tab(text: 'Coach'),  
        Tab(text: 'User'),  
      ],  
    ), // TabBar  
    // TabBarView to show content of selected tab  
    Expanded(  
      child: TabBarView(  
        controller: _tabController,  
        children: const [  
          AdminCoachPage(),  
          AdminUserPage(),  
        ],  
      ), // TabBarView  
    ), // Expanded  
  ],  
, // Column
```

This code sets up a TabBar and TabBarView within a column layout, providing two tabs: "Coach" and "User." The TabBar allows toggling between these tabs, while the TabBarView displays either AdminCoachPage or AdminUserPage based on the selected tab. The Expanded widget ensures that TabBarView fills the remaining space below the TabBar in the column layout. Admins can view statistics (e.g., user and coach activity logs, registrations) and generate reports.

2.7.3.2.4 View User and Coach Profiles

```
Expanded(  
  child: TabBarView(  
    controller: _tabController,  
    children: const [  
      AdminCoachPage(),  
      AdminUserPage(),  
    ],  
  ),  
)
```

This code snippet creates an Expanded widget containing a TabBarView, which uses _tabController to manage the display of two pages: AdminCoachPage and AdminUserPage. The Expanded widget ensures that the TabBarView fills the available space, allowing the user to switch between these two pages within the view. Admins can view user and coach profiles.

2.7.3.2.4 View Workout Activities

```
// Function to fetch activities from Firebase  
Future<List<Map<String, dynamic>>> fetchActivities() async {  
  final response = await http.get(Uri.parse('${databaseURL}activity.json'));  
  
  if (response.statusCode == 200) {  
    final Map<String, dynamic> data = json.decode(response.body);  
    List<Map<String, dynamic>> activities = [];  
  
    data.forEach((key, value) {  
      activities.add({  
        'id': key, // Store activity ID  
        'category': value['category'],  
        'videodescription': value['videodescription'],  
        'videoname': value['videoname'],  
        'videourl': value['videourl'],  
      });  
    });  
  
    return activities;  
  } else {  
    throw Exception('Failed to load activities');  
  }  
}
```

The `fetchActivities` function retrieves a list of activities from the database by sending a GET request to the `activity.json` endpoint. If successful (status code 200), it decodes the response into a map, then iterates through each entry, adding activity details (ID, category, video description, video name, and video URL) to a list of maps. This list is returned as the function result. If the request fails, an exception is thrown. Admins can view the workout activities.

2.7.3.3 Update

2.7.3.3.1 Update User and Coach Profiles

```
@override
Widget build(BuildContext context) {
  return AlertDialog(
    title: const Text('Edit Coach'),
    content: Column(
      mainAxisAlignment: MainAxisAlignment.min,
      children: [
        TextField(controller: _usernameController, decoration: const InputDecoration(labelText: 'Username')),
        Row(
          mainAxisAlignment: MainAxisAlignment.spaceAround,
          children: [
            IconButton(
              icon: Icon(Icons.male, color: _isMale ? Colors.blue : Colors.grey),
              onPressed: () => setState(() => _isMale = true),
            ), // IconButton
            IconButton(
              icon: Icon(Icons.female, color: !_isMale ? Colors.pink : Colors.grey),
              onPressed: () => setState(() => _isMale = false),
            ), // IconButton
          ],
        ), // Row
        TextField(controller: _bioController, decoration: const InputDecoration(labelText: 'Bio')),
        TextField(controller: _secretPassController, decoration: const InputDecoration(labelText: 'Secret Pass')),
        TextField(controller: _passwordController, decoration: const InputDecoration(labelText: 'Password')),
      ],
    ),
  );
}
```

This code builds an `AlertDialog` for editing a coach's information. It includes fields for the coach's username, gender (selectable with male and female icons), bio, secret pass, and password. The dialog has two actions: a "Save" button that updates the coach's information via the `updateCoach` function with the edited details and closes the dialog, and a "Cancel" button that simply closes the dialog without saving. The `setState` method is used to update the selected gender dynamically. Admins can update user and coach profiles (e.g., change username, password, or secret word).

2.7.3.3.2 Edit Workout Activities

```
// Function to edit an existing activity
Future<void> editActivity(
    String activityId, Map<String, dynamic> updatedActivity) async {
    final response = await http.put(
        Uri.parse('${databaseURL}activity/$activityId.json'),
        body: json.encode(updatedActivity),
    );

    if (response.statusCode != 200) {
        throw Exception('Failed to edit activity');
    }
}
```

The editActivity function updates an existing activity in the database by sending a PUT request to the specified activity/\$activityId.json endpoint with the updatedActivity data in JSON format. If the response status code is not 200, indicating a failure, it throws an exception to signal that the activity could not be edited. Admins can edit workout activities.

2.7.3.4 Delete

2.7.3.4.1 Delete Posts by Coaches

```
@override
void initState() {
    super.initState();
    fetchPosts();
}

Future<void> fetchPosts() async {
    final fetchedPosts = await getPosts('disapproved', false);
    setState(() {
        posts = fetchedPosts;
    });
}
```

In this code, the initState method initializes by calling fetchPosts to load posts marked as "disapproved." The fetchPosts function retrieves these posts using getPosts and updates the posts variable in setState to refresh the UI. The _showReportDialog function displays a

confirmation dialog when the user wants to report or update a post. If the user confirms, `updatePost` is called to either set the post as reported (disapproved) or update its status based on the action. The posts list is refreshed after updating by calling `fetchPosts`. Admins can delete inappropriate or flagged posts by coaches.

2.7.3.4.2 Delete User and Coach Account

```
Future<void> deleteCoach(String coachId) async {
  final url = Uri.parse('$databaseURL/user/coach/$coachId.json');
  await http.delete(url);
}

Future<void> deleteUser(String userId) async {
  final url = Uri.parse('$databaseURL/user/user/$userId.json');
  await http.delete(url);
}
```

These two functions, `deleteCoach` and `deleteUser`, each delete a specified record from the database. `deleteCoach` sends a DELETE request to remove a coach entry identified by `coachId`, while `deleteUser` sends a DELETE request to remove a user entry identified by `userId`. Both functions construct the URL based on the `databaseURL` and the appropriate path for each entry type. Admins can delete user and coach accounts if necessary.

2.7.3.4.3 Delete Workout Activities

```
// Function to delete an activity
Future<void> deleteActivity(String activityId) async {
  final response =
    await http.delete(Uri.parse('${databaseURL}activity/$activityId.json'));
  if (response.statusCode != 200) {
    throw Exception('Failed to delete activity');
  }
}
```

The `deleteActivity` function removes an activity from the database by sending a DELETE request to the `activity/$activityId.json` endpoint. If the operation is unsuccessful (status code not equal to 200), it throws an exception to indicate that the deletion of the activity failed. Admins can delete the workout activities.

2.8 Validation

```
import 'dart:convert';
import 'package:http/http.dart' as http;

const String databaseURL = 'https://fitness-app-490b6-default-rtdb.asia-southeast1.firebaseio.com/';

Future<Map<String, String>?> checkLogin(String username, String password) async
{
  final url = Uri.parse('${databaseURL}user.json');
  final response = await http.get(url);

  if (response.statusCode == 200)
  {
    final Map<String, dynamic> data = json.decode(response.body);

    for (var category in ['admin', 'coach', 'user'])
    {
      if (data.containsKey(category))
      {
        final userData = data[category];

        for (var userId in userData.keys)
        {
          final user = userData[userId];
          if (user['username'] == username && user['password'] == password)
          {
            return {'category': category, 'userId': userId};
          }
        }
      }
    }

    return null;
  }
  else
  {
    throw Exception('Failed to load user data');
  }
}
```

During the login process, the system looks for the validation of the entered username and password entered to check whether it really matches the data present in the Firebase database. When the user submits their credentials, the system retrieves that stored data from Firebase and matches it with the entered credentials. In cases when both the username and password that were entered match any of the accounts contained in the database, a user can access and log in successfully. If such credentials do not find a match in the data contained in Firebase, the user will see an error message stating that either the username or the password is wrong. This confirms validation for authenticity and verification at the time of use.

```

import 'dart:convert';
import 'package:http/http.dart' as http;

const String databaseURL = 'https://fitness-app-490b6-default-rtdb.asia-southeast1.firebaseio.com/';

Future<bool> checkSecretWord(String username, String secretWord) async
{
  final url = Uri.parse('${databaseURL}user.json');
  final response = await http.get(url);

  if (response.statusCode == 200)
  {
    final Map<String, dynamic> data = json.decode(response.body);

    for (var category in ['admin', 'coach', 'user'])
    {
      if (data.containsKey(category))
      {
        final userData = data[category];

        for (var userId in userData.keys)
        {
          final user = userData[userId];
          if (user['username'] == username && user['secretpas'] == secretWord)
          {
            return true;
          }
        }
      }
    }
    return false;
  }
  else
  {
    throw Exception('Failed to load user data');
  }
}

```

The system has employed a two-tier authentication method, which verifies the actual user in case of a forgotten password. A user who wants to reset the password must enter their username and a secret passcode that they had set up earlier in the system. It then verifies through Firebase whether the entered username and secret passcode exist within the database and matches the stored information of the user. When a match occurs, this means the identity of the user is confirmed, and he can change his password. If no match is found, the system gives an error message that either username or secret passcode is incorrect. An extra layer of verification does add to the security because unauthorized users cannot reset other accounts' passwords.

2.9 Users Permission

```
void _handleLogin() async {  
  final username = _usernameController.text.trim();  
  final password = _passwordController.text.trim();  
  
  final result = await checkLogin(username, password);
```

Here, on attempting to log in, the system begins the retrieval of a username and password entered through the respective text fields by the user. Immediately after that, the system invokes the checkLogin function, which interacts with the database for credential verification. This function is designed to verify the entered information against an existing record in the database, ensuring that what the user has entered corresponds with a valid account.

```
if (result != null) {  
  _userId = result['userId'];  
  final category = result['category'];
```

The system returns the userId and the userCategory in case the entered username and password match with any in the database to identify the role of the user, such as admin, coach, or regular user.

```
if (category == 'admin') {  
  Navigator.pushNamed(context, '/admin_home', arguments: {'userId': _userId});
```

Once the system has checked the user's role, it checks whether the userCategory matches an admin. In case the latter is true, the system forwards to the admin homepage, passing the userId to make sure that the admin's particular data and permissions are available on the page.

```
} else if (category == 'coach') {  
  Navigator.pushNamed(context, '/coach_home', arguments: {'userId': _userId});
```

Depending on the userCategory, the user is immediately sent either to the coach homepage or to the regular user homepage. It also passes the userId parameter to this page so that the coach can manage his exercises or view user progress.

```

} else if (category == 'user') {
  Navigator.pushNamed(context, '/user_home', arguments: {'userId': _userId});
}

```

If it's just a regular user, then the system routes to the user homepage, appending the `userId` to grant access to personalized data and features relevant to a standard user.

```

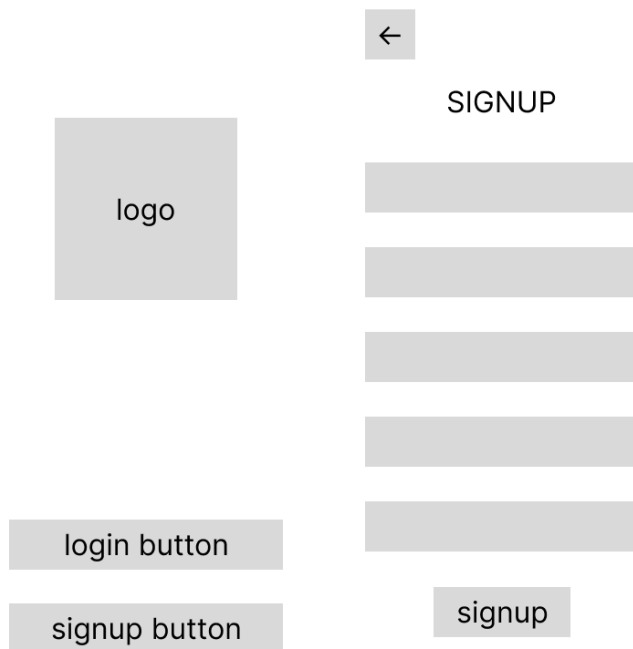
} else {
  setState(() {
    _showError = true;
  });
  Future.delayed(const Duration(seconds: 2), () {
    setState(() {
      _showError = false;
    });
  }); // Future.delayed
}

```

In case the function `checkLogin` fails to find a matching pair of username and password in the database, it means that the credentials are invalid, and an error will be caught by the system to handle this situation elegantly. The appearing error message will inform the user about the wrong attempt to log in. It will be displayed for a very short time-2 seconds-implemented by using `Future.delayed`-after which it just disappears, and thus this will not interfere with further user actions if he wants to log in again.

2.9 Wireframe

2.9.1 General



The first page of the application, shown in Image 1, contains a logo that is centred in the middle of the page. This will be the first greeting seen when the application is opened. Once forwarded to the next page as depicted in Image 1, the user will be taken to a login and sign-up page. Here, there is a logo in the centre at the top and two buttons: the "Login" button, directing to the login page, and the "Signup" button, redirecting to the signup page. Image 2 is for the signup page where several text fields provide space for users to input their information. Below those fields, there is a "Signup" button for completing the registration process. There is also a "Click here" button for previously registered users to go back to the login page and a "Back" button for going back to the previous screen (Image 1).

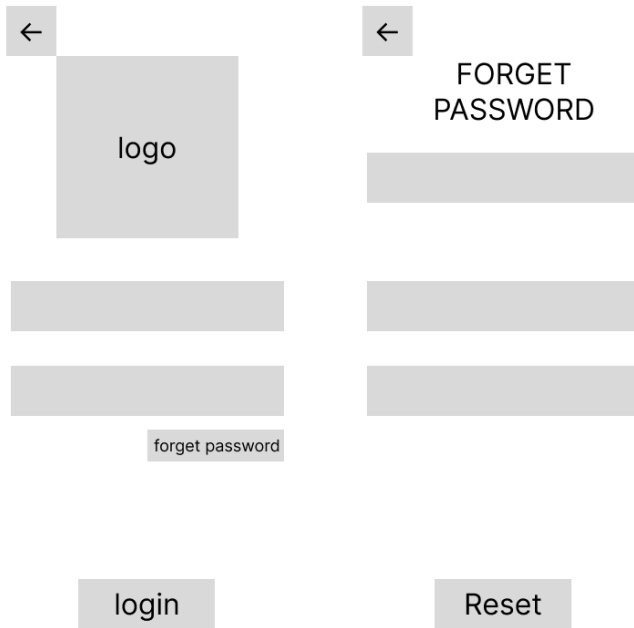


Image 3 is the login page, which contains text fields for entering a username and password. There is a "Back" button to return to the previous screen (Image 2), a "Forgot Password" button that leads to the password recovery page (Image 4), and a "Login" button to proceed with signing in. Lastly, Image 4 is the "Forgot Password" page, where users can enter necessary details for password validation and update their password through the provided text fields. A "Reset" button updates the new password, and a "Back" button allows users to return to the login page (Image 3).

2.9.2 User

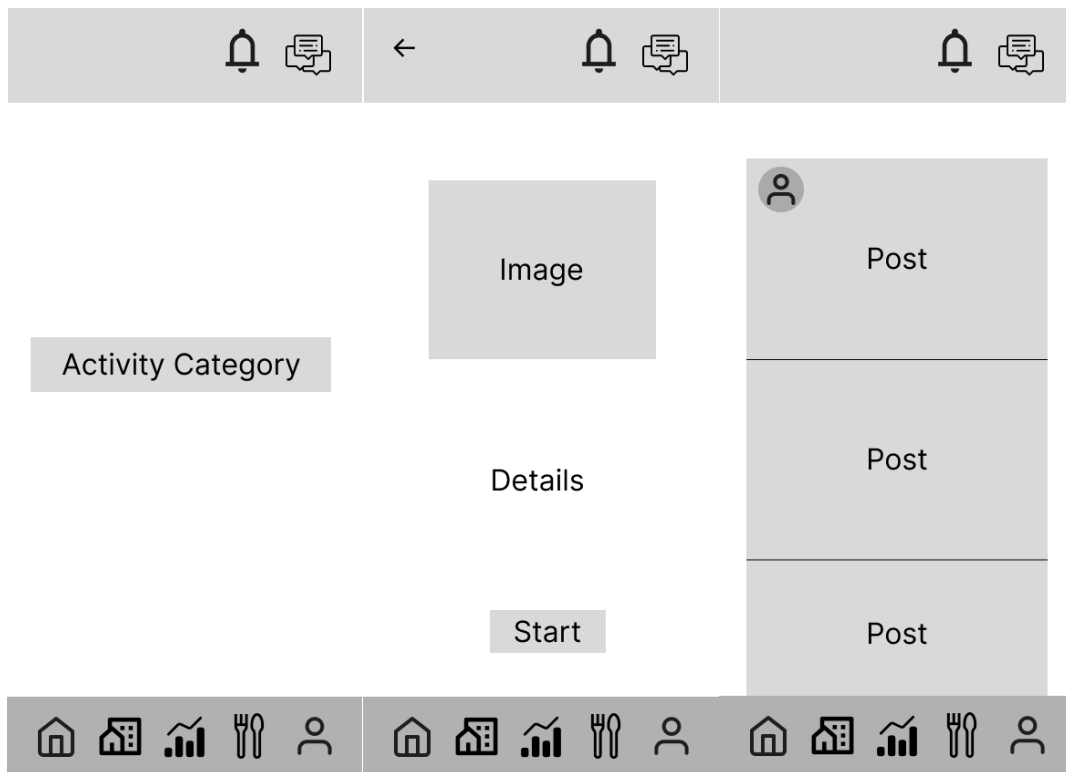


Image 1 is the exercise activity page, where users can view all exercise categories. They can click on an activity leads to its detailed view (Image 2). A bottom navigation bar provides access to various sections: the "Home" button (exercise and activity, Image 1), "Community" button (posts and discussions, Image 3), "Report" button (activity report, Image 6), "Food" button (food suggestions, Image 7), and "Profile" button (user profile, Image 8). At the top right corner, there is a "Notification" button (Image 9) to view notifications and a "Feedback" button (Image 10) for sending and viewing feedback.

In Image 2, users can view specific details of an activity or exercise, including an image and a textual description. A "Start" button allows users to begin the activity, and a "Back" button returns them to the exercise activity page (Image 1). Image 3 is the community page, displaying all posts. Users can search for specific individuals via a search bar or click on a profile image to view a user's profile (Image 4).

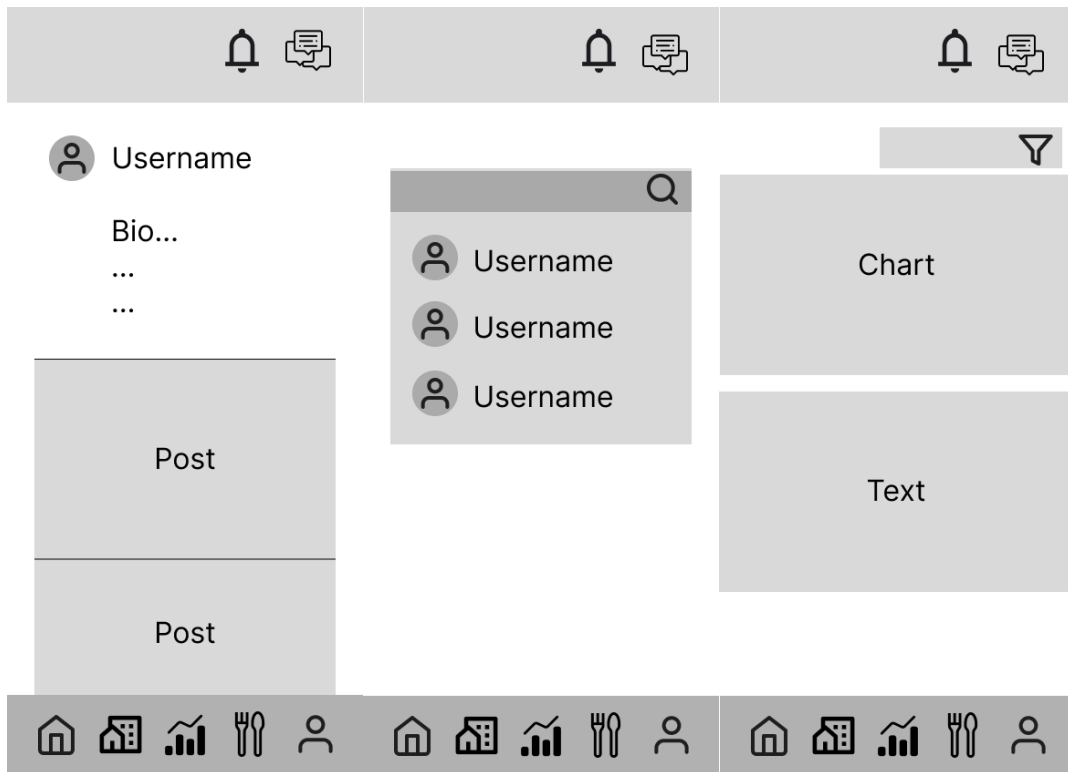


Image 4 is the user profile page, displayed when a user clicks on another user's profile image in the community page (Image 3). This page shows the specific user's username, bio, posts, and other relevant details. Image 5 is the search page, where users can enter a username to search for specific individuals. Search results display relevant users, and clicking on a result will take the user to the corresponding profile page (Image 4). Image 6 is the report page, which presents a summary of the user's activity report, such as the number of activities completed on different days. Below the report, a text area provides more detailed information in written form.

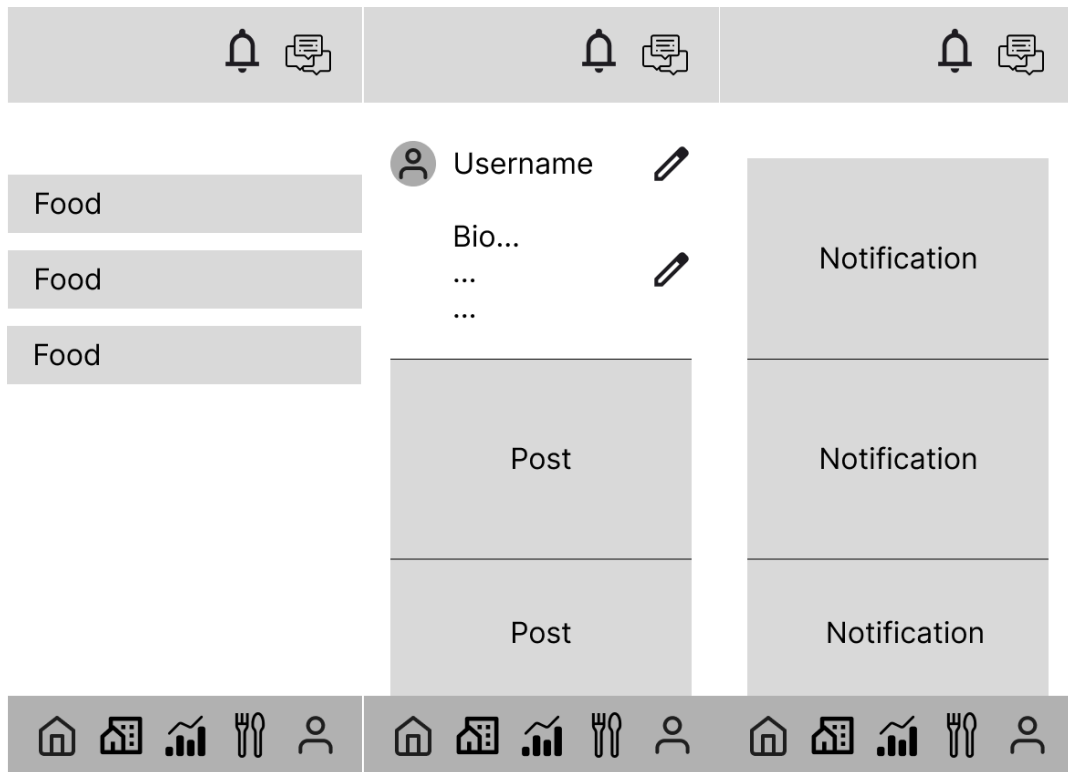


Image 7 is the nutrition page, where users can view meal plans suggested by the system. Users can filter the meal plans by time of day (morning, afternoon, night) to see options for breakfast, lunch, and dinner. Image 8 is the user's profile page, where personal information is displayed. There is an "Edit" button, allowing users to update their information as needed. Image 9 is the notification page, displaying all notifications and announcements relevant to the user.

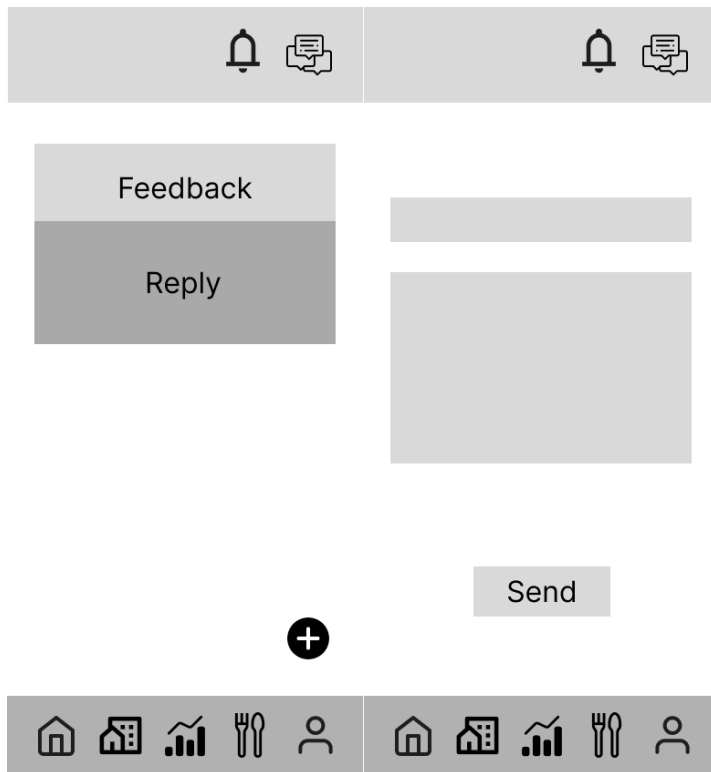


Image 10 is the feedback page, where users can view feedback, they've posted and see any replies from the admin. There is an "Add" button that navigates to the new feedback submission page (Image 11). Image 11 is the add new feedback page, which includes text fields for entering the title, selecting feedback categories, and writing the content. Once completed, users can click the "Send" button to submit their feedback.

2.9.3 Coach

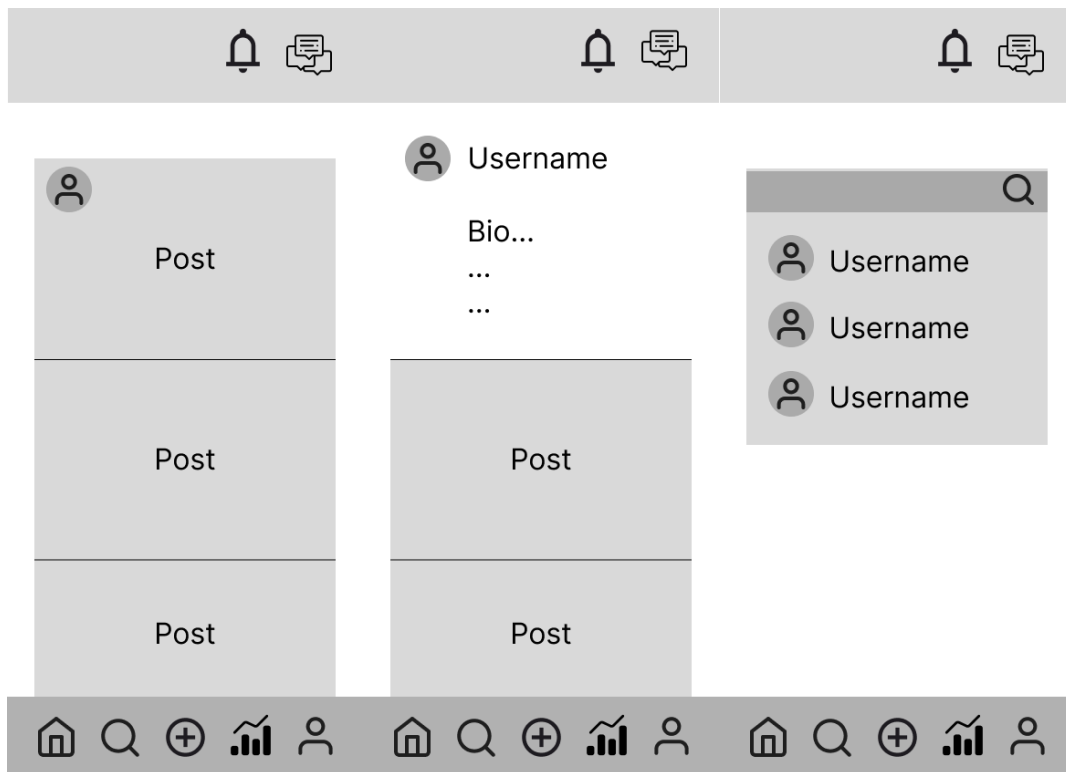


Image 1 is the community page, where all posts are displayed. Users can click on a profile image within a post to view that specific user's profile (Image 2). The bottom navigation bar includes several options: "Home" for viewing all community posts (Image 1), "Search" to find users (Image 3), "Add" to create a new post (Image 4), "Report" to access the user engagement report (Image 5), and "Profile" to display the user's own profile (Image 6). In the top right corner, there are buttons for "Notification" (Image 7), leading to the notification page, and "Feedback" (Image 8), which displays feedback sent by the user along with replies from the admin.

Image 2 shows a specific user's profile, displayed when a user clicks on that user's profile image. This page includes the specific user's username, bio, and posts. Image 3 is the search user page, where users can enter a username to search for specific individuals. The results will display related users, and clicking on a specific user's profile image will take the user to that profile page (Image 2).

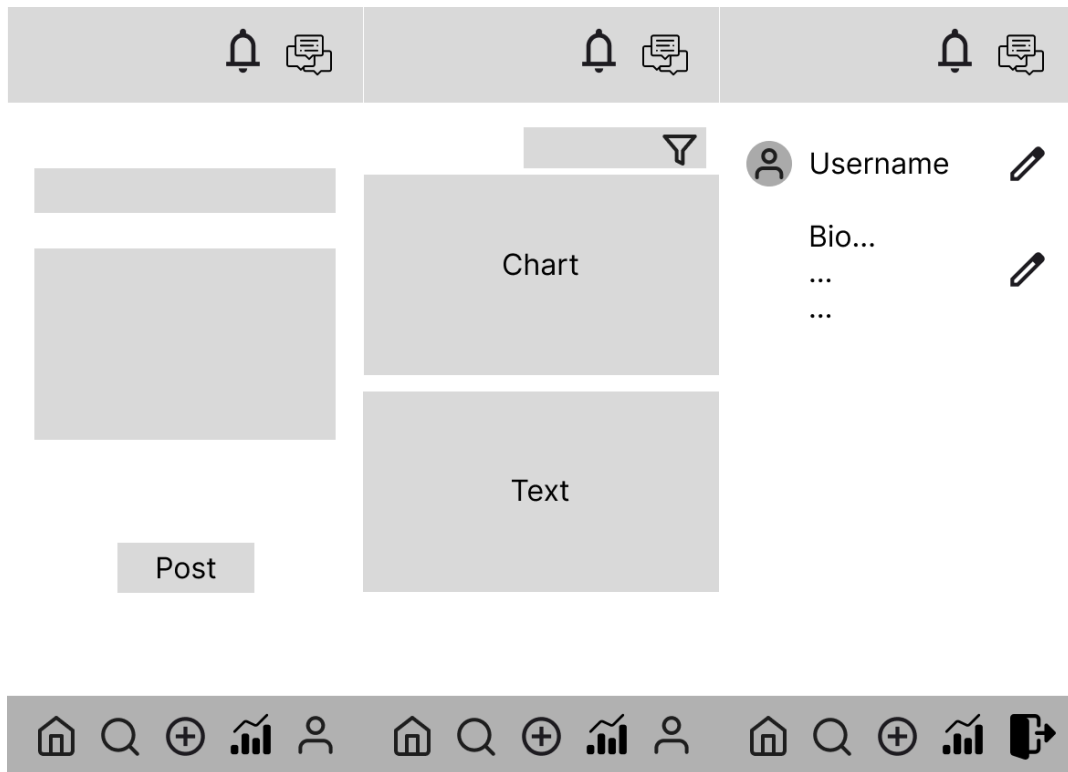


Image 4 is the add new post page, where users can enter a title and content for their post. A "Post" button allows them to submit their title and content to the community (Image 1). Image 5 is the report page, featuring a chart that displays the user engagement report, illustrating metrics such as the number of users who liked specific posts. Image 6 is the profile page, which displays the user's username and bio.

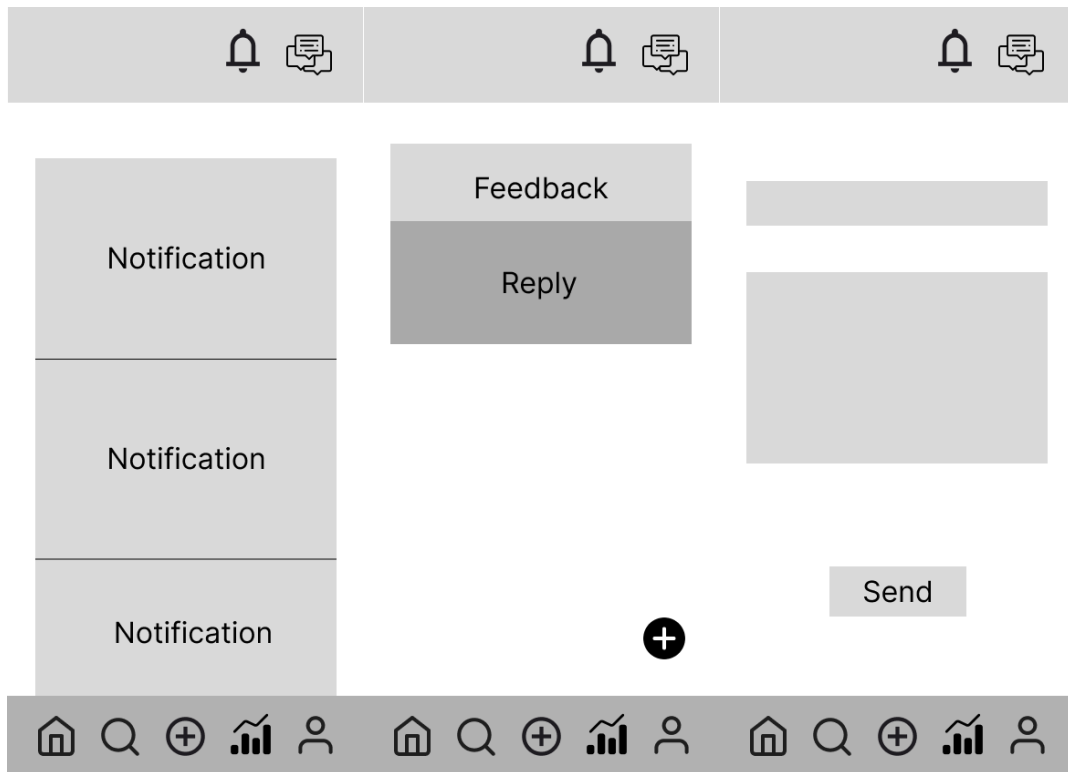


Image 7 is the notification page, which displays all notifications and announcements relevant to the user. Image 8 is the feedback page, where users can view the feedback, they have sent along with any replies from the admin. An "Add" button allows users to navigate to the new feedback submission page (Image 9). Image 9 is the add new feedback page, where users can enter a title and provide the content of their feedback. A "Send" button enables users to submit their feedback.

2.9.4 Admin

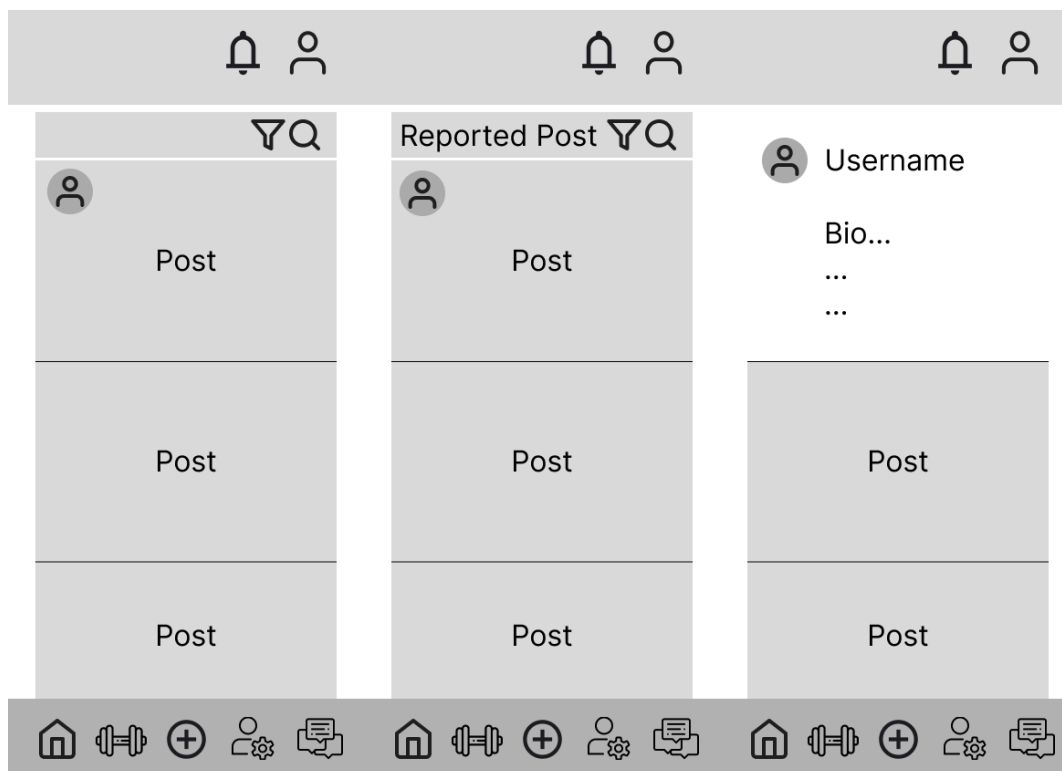


Image 1 displays the posts within the community, where users can view all shared content. Each post includes a profile link that, when clicked, takes the user to the specific user's profile (Image 2). There are also filter and search buttons: the "Filter" button allows users to display only reported posts, while the "Search" button enables users to find specific users.

The bottom navigation bar includes several options: "Home" to view all community posts (Image 1), "Activity" to access exercises and activities (Image 4), "Add" to create a new announcement (Image 5), "Management" to view user engagement reports and manage users (Image 6), and "Feedback" to review all feedback sent by users and coaches (Image 7). In the top right corner, there are buttons for "Notification" (Image 8), leading to the notification page, and "Profile" (Image 9) to display the admin's profile.

Image 2 shows the reported posts page, where all reported content is displayed. Admins can choose to approve or disapprove the posts after reviewing the content. Image 3 is the user profile page, which displays a specific user's information when their profile image is clicked in either Image 1 or Image 2.

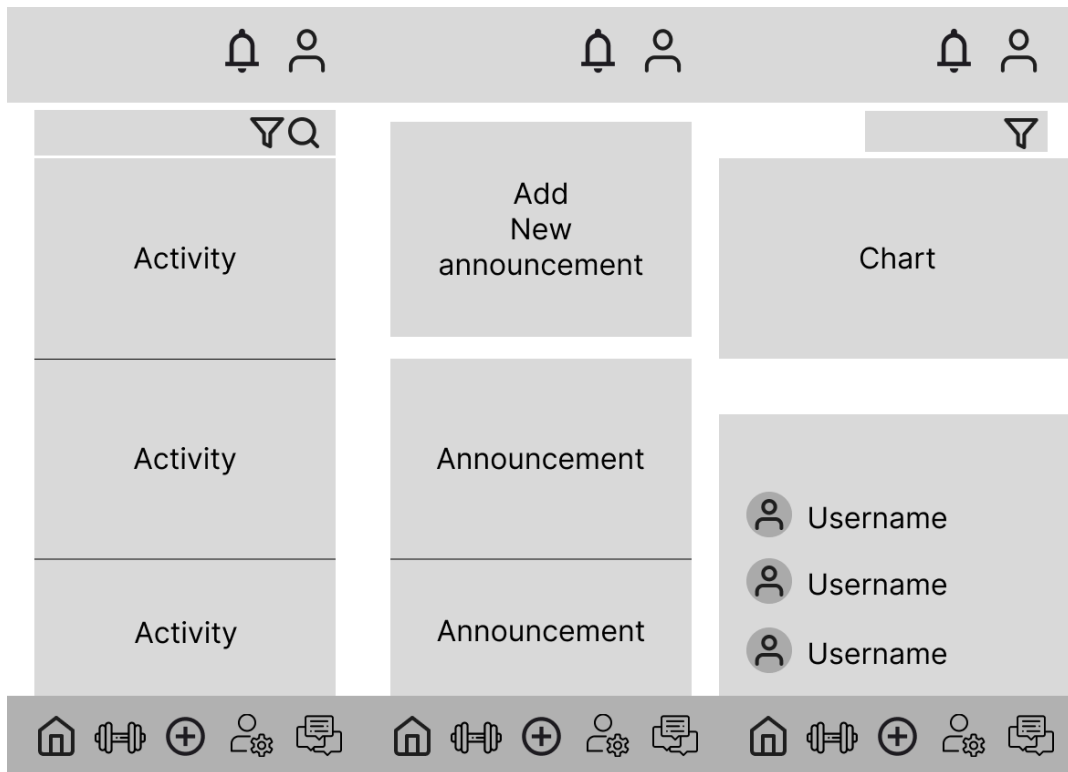


Image 4 is the exercise/activity page, where all exercises and activities are displayed. Admins have the capability to add, edit, or delete activities and exercises, providing comprehensive management of the content. Image 5 is the announcement page, allowing admins to add new announcements or view existing ones, ensuring important information is communicated effectively to users. Image 6 is the report page, which displays the user engagement report. Below the report, admins can view and edit user information, facilitating management and oversight of user activity.

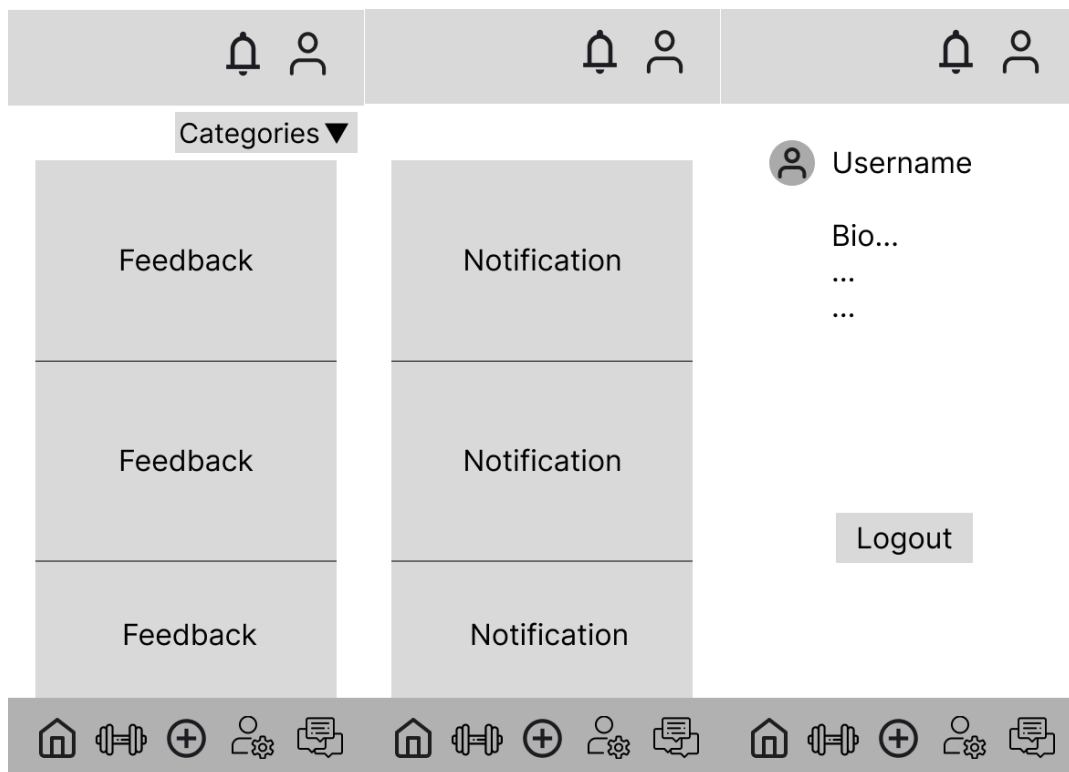


Image 7 is the feedback page, where all feedback sent by users and coaches is displayed. Admins can filter the feedback by categories to easily view and reply to specific feedback, enhancing communication and responsiveness. Image 8 is the notification page, which displays all notifications relevant to the admin, ensuring they stay informed about important updates and messages. Image 9 is the profile page, where the admin's username and information are displayed. A "Logout" button is available for the admin to log out of the application securely.

3.0 Use Manual

3.1 General

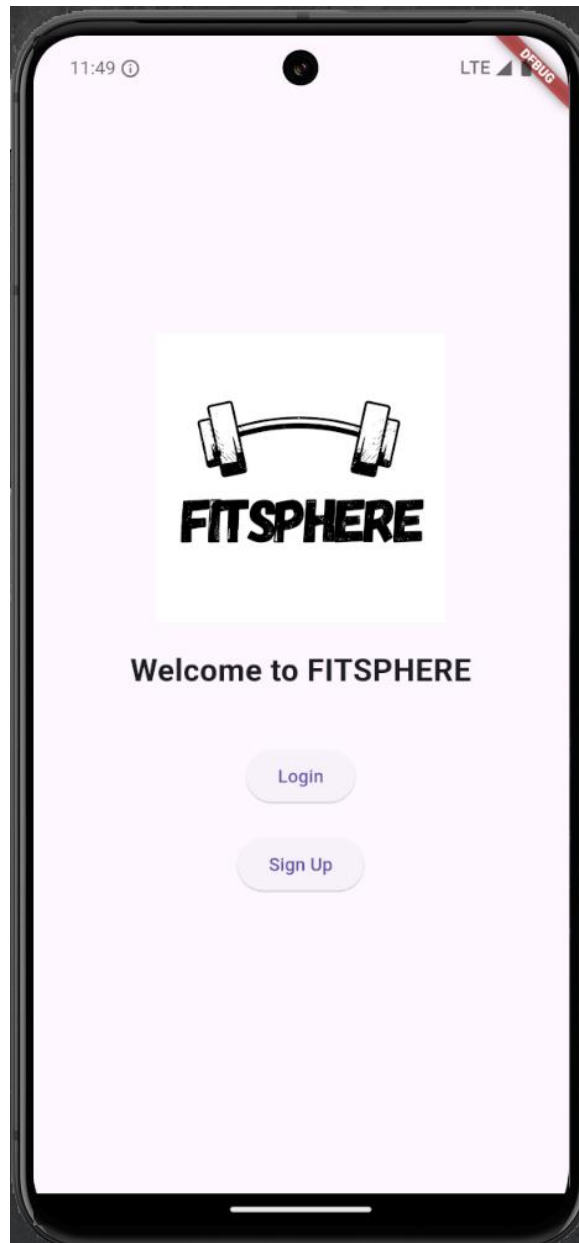


Figure 1 Entry Page

On Entry Page, there is two function which is Login and Sign Up.

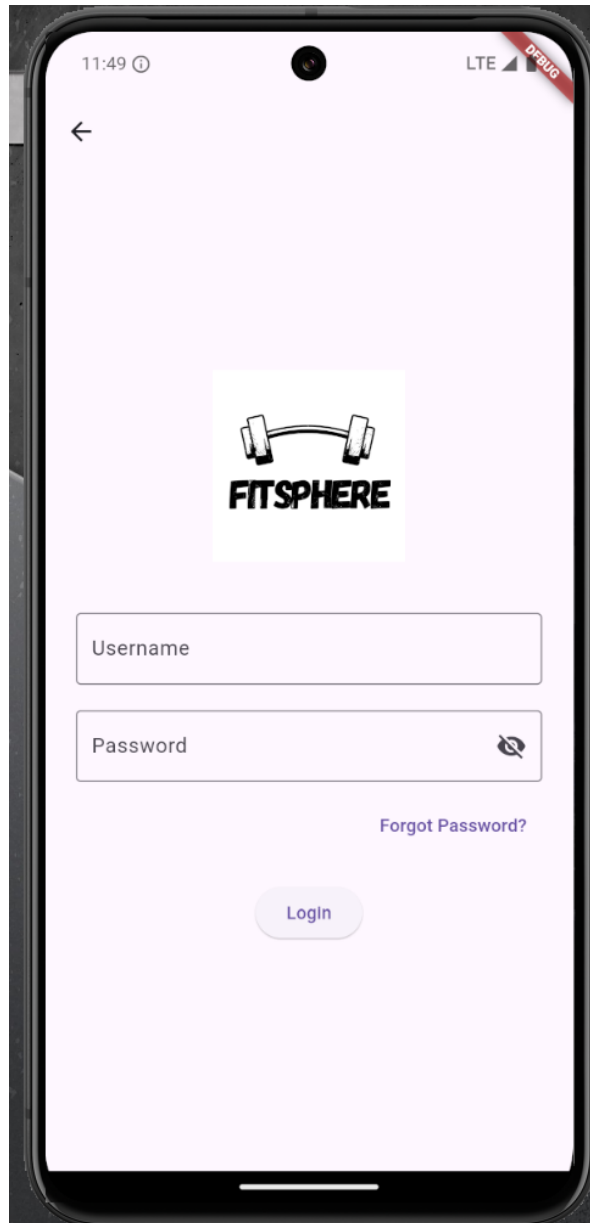
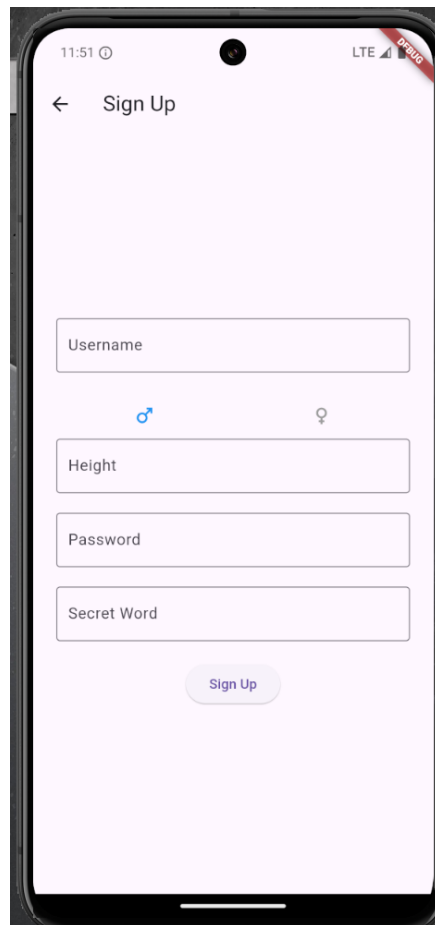


Figure 2 Login Page

User, coach and admin will proceed their login in this figure 2 login page.

3.2 User



11:51 LTE

← Sign Up

Username

♂ ♀

Height

Password

Secret Word

Sign Up

Figure 3 Sign Up Page

New User will have to sign up new account to proceed to homepage of this application.

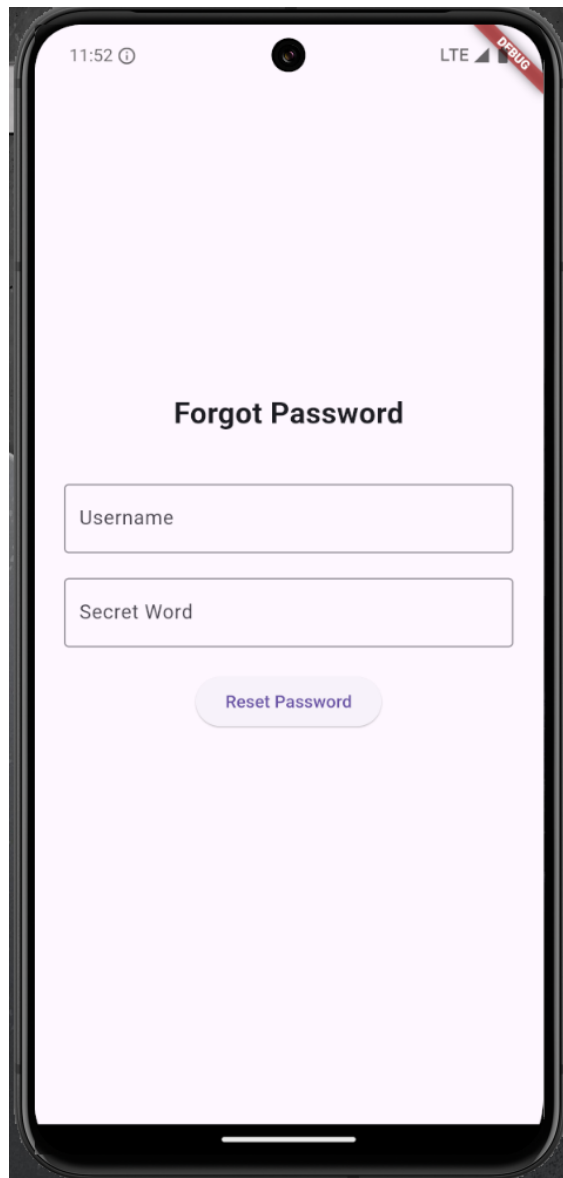


Figure 4 Forget Password Page

When user forgotten their password, use will have to proceed to this page and key username and secret word to change new password.

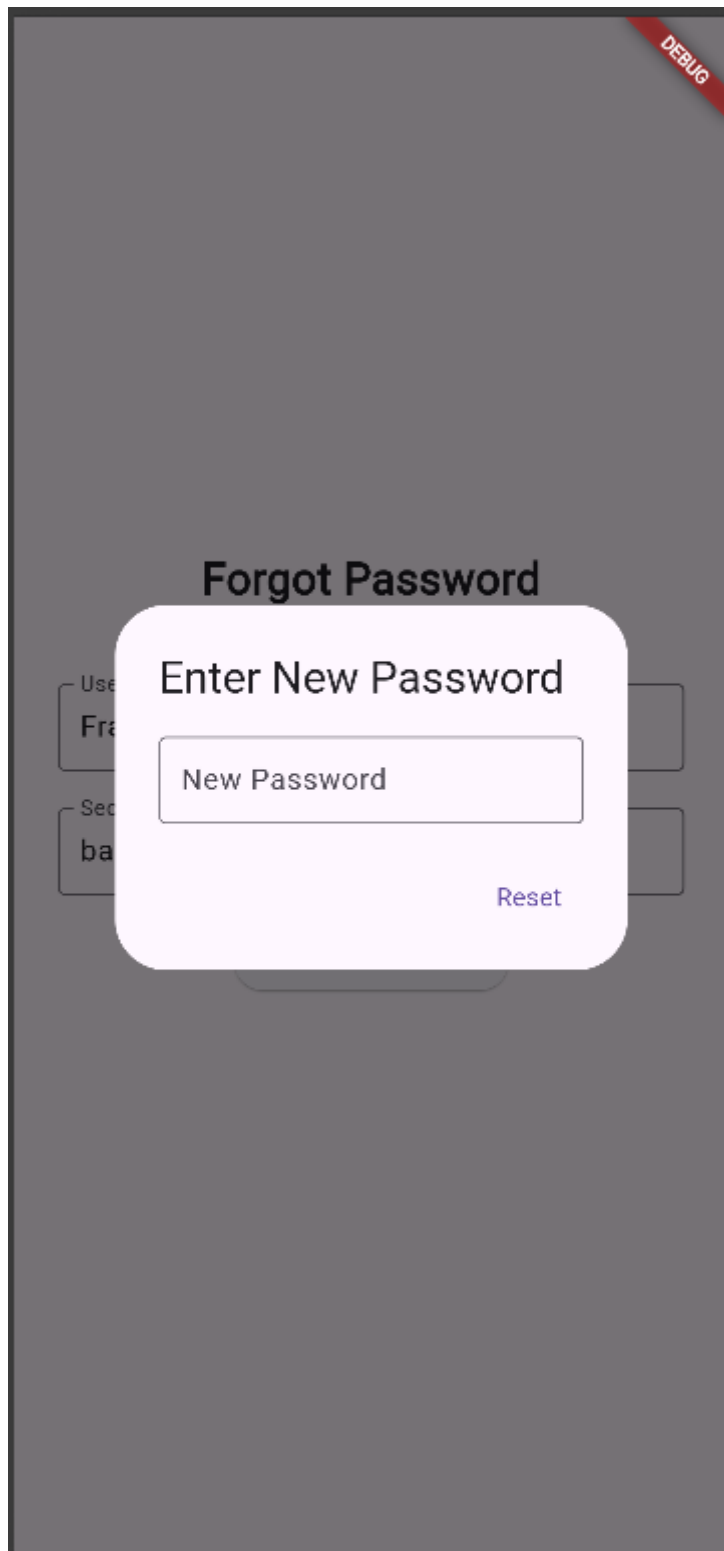


Figure 5 Enter New Password

When a user insert correct username and password, user will type their new password to change the password.

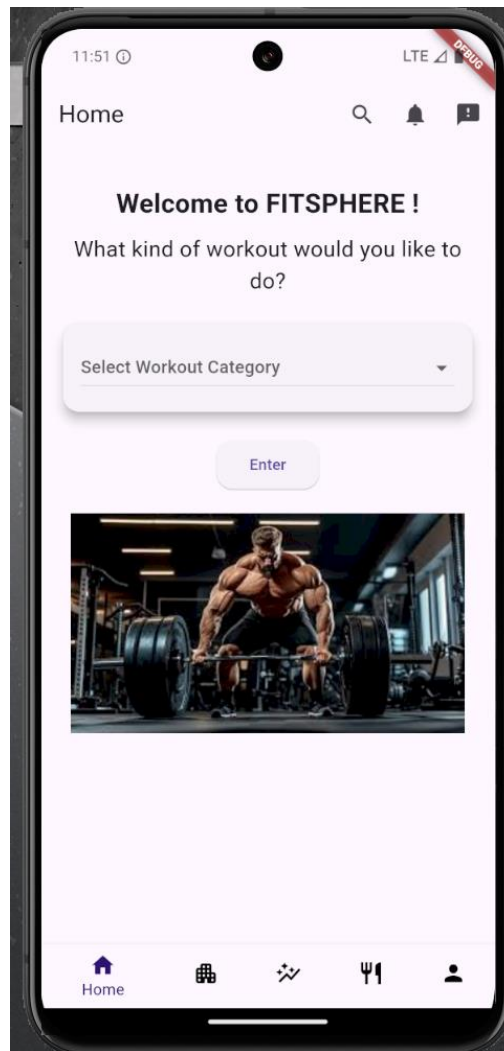


Figure 6 User Homepage

On user homepage, user will choose the workout category and click enter proceed to activity detail page.

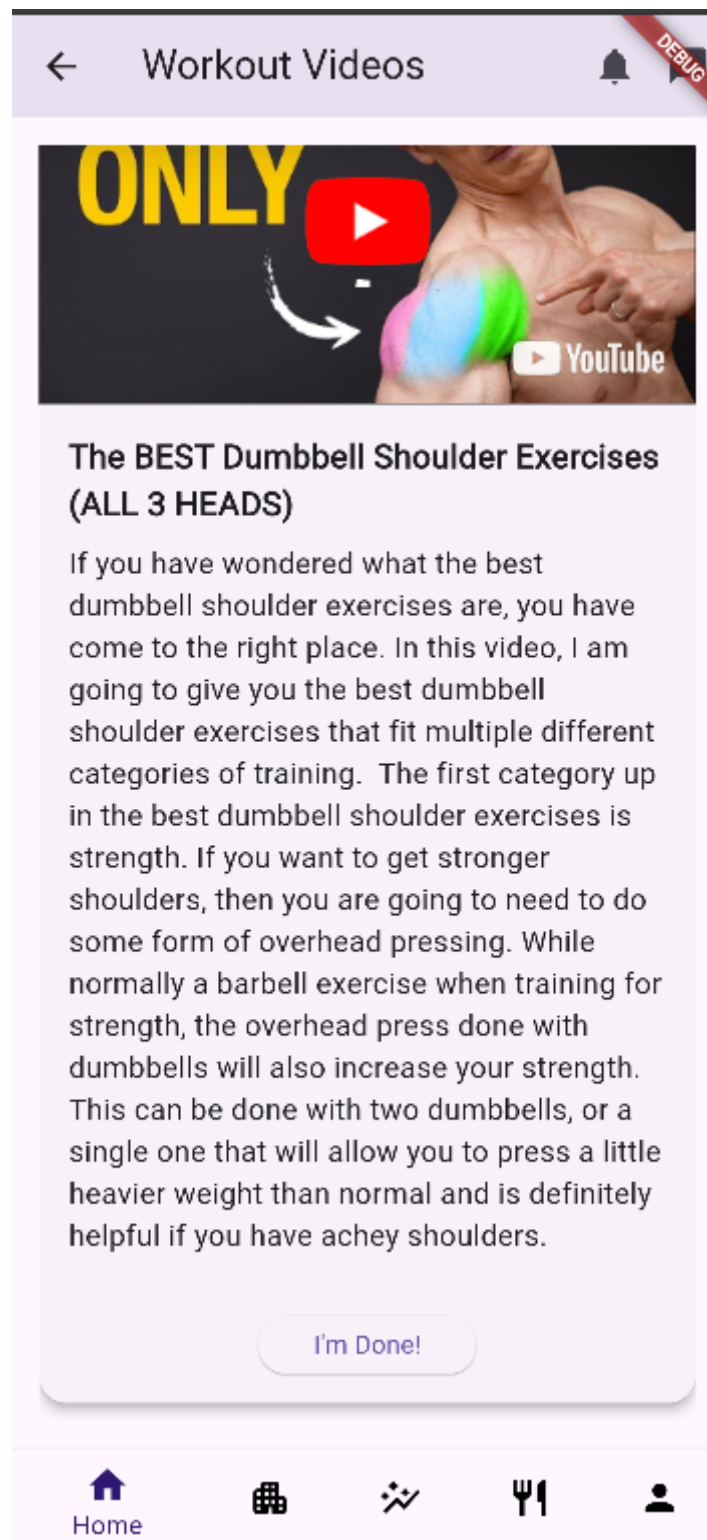


Figure 7 User Activity detail page

On activity detail page, user will view the video and description of the specific exercise. User will click "I'm Done" button when they finish the exercise.

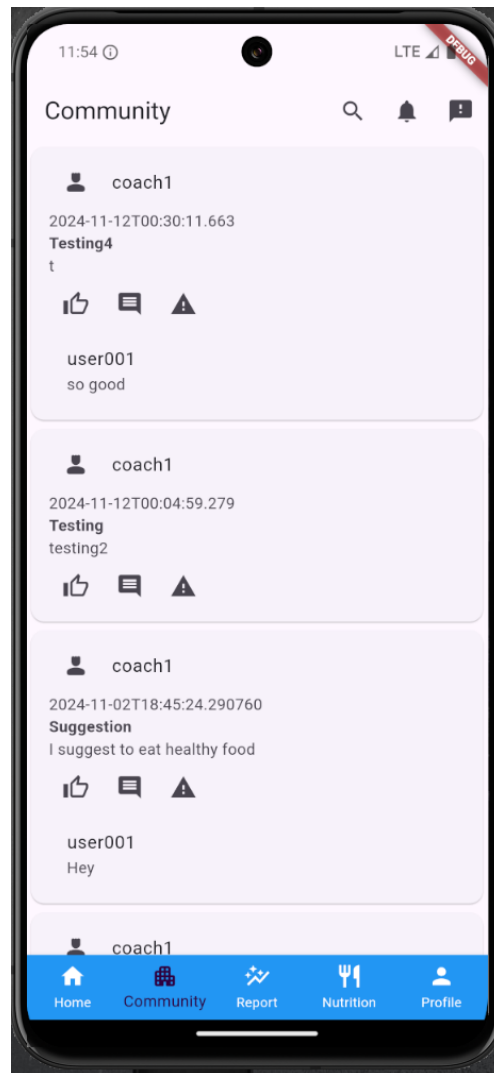


Figure 8 User Community Page

On Community Page, user will see the post that posted by the coach on the information on what coach want to share.

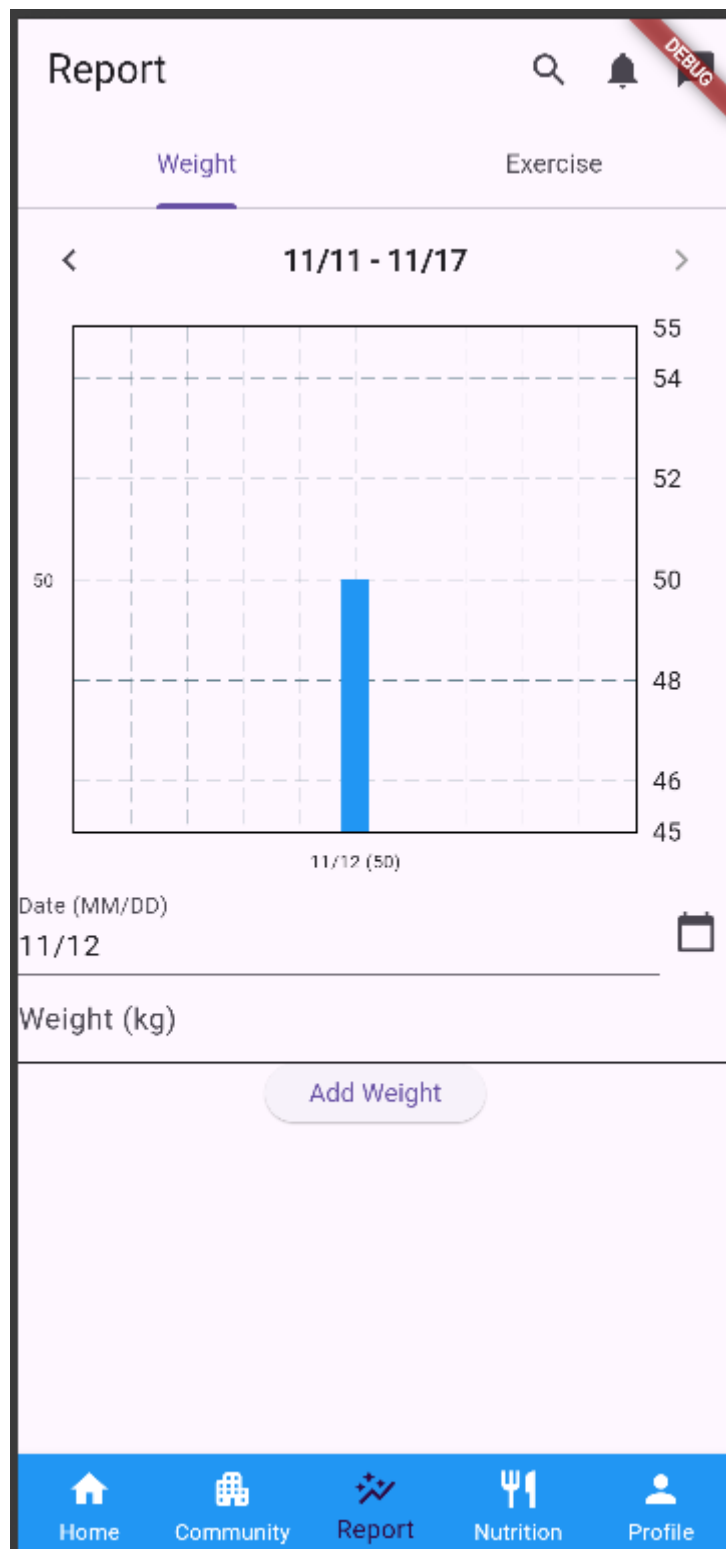


Figure 9 User Report Page (weight)

On report Page which have two choices which is weight and exercise. On this figure shown that the weight of the user thought the time which the wight information will be insert by the user during time to time.

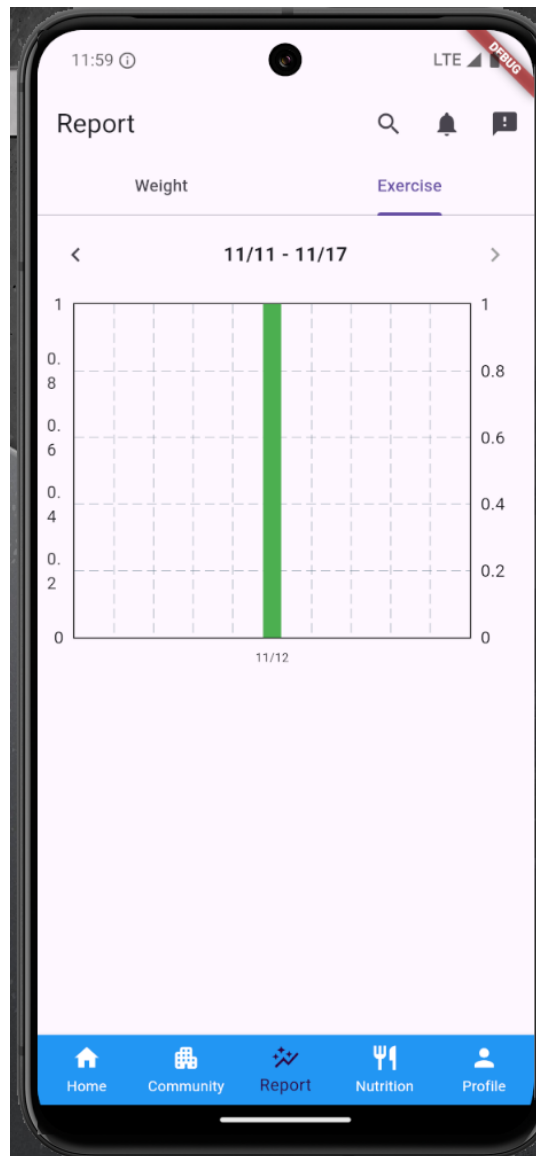


Figure 10 User Report Page (Exercise)

On this figure shown how many times of specific exercise that have did by the user on one day.

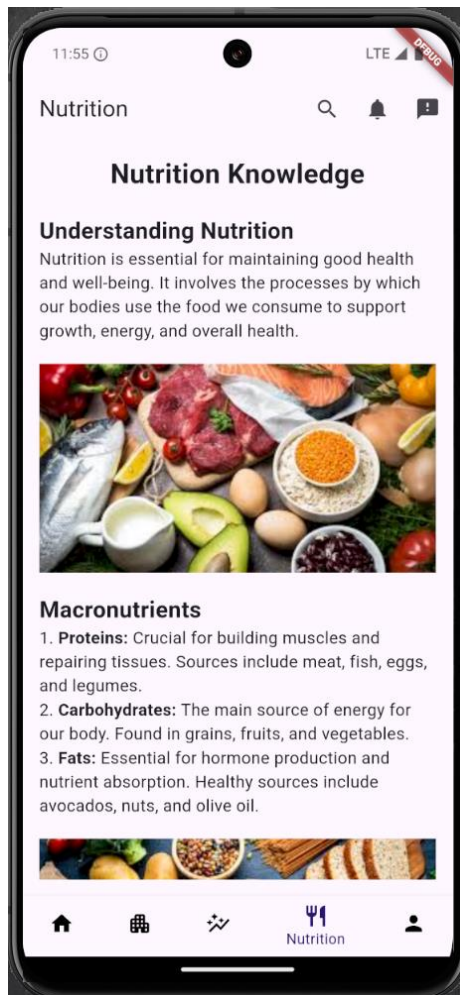


Figure 11 User Nutrition Page

Nutrition Page provide the understanding and the and the information about nutrition for user to make a reference.

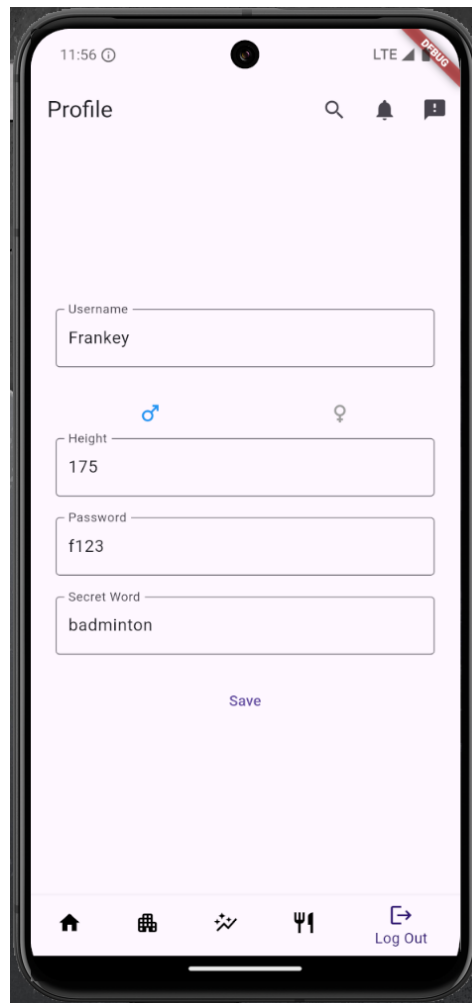


Figure 12 User Profile Page

User will be able to view their own personal information in profile page and able to make editing on the profile, with confirmation and click save by saving the latest personal detail in profile page. User will also proceed their log out in profile page on the button right button.

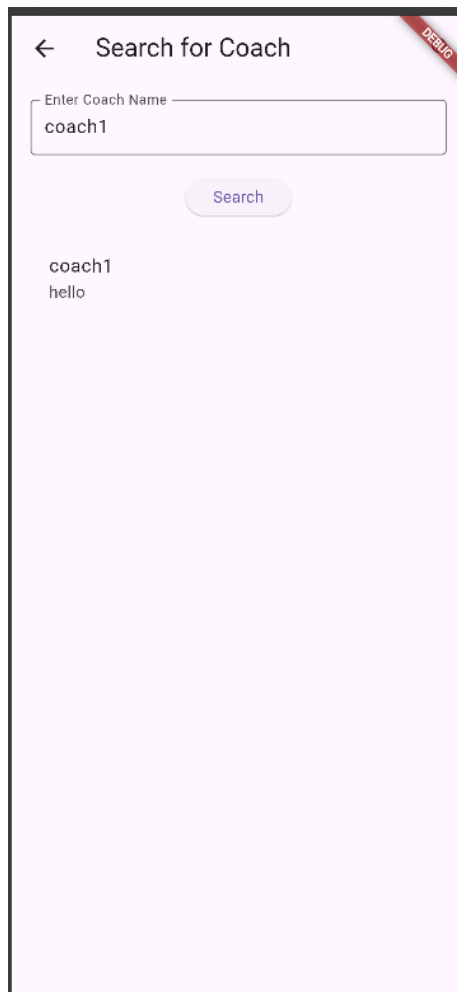


Figure 13 User Search Page

User will be able to find a specific coach name by type the name of the coach in the search page.

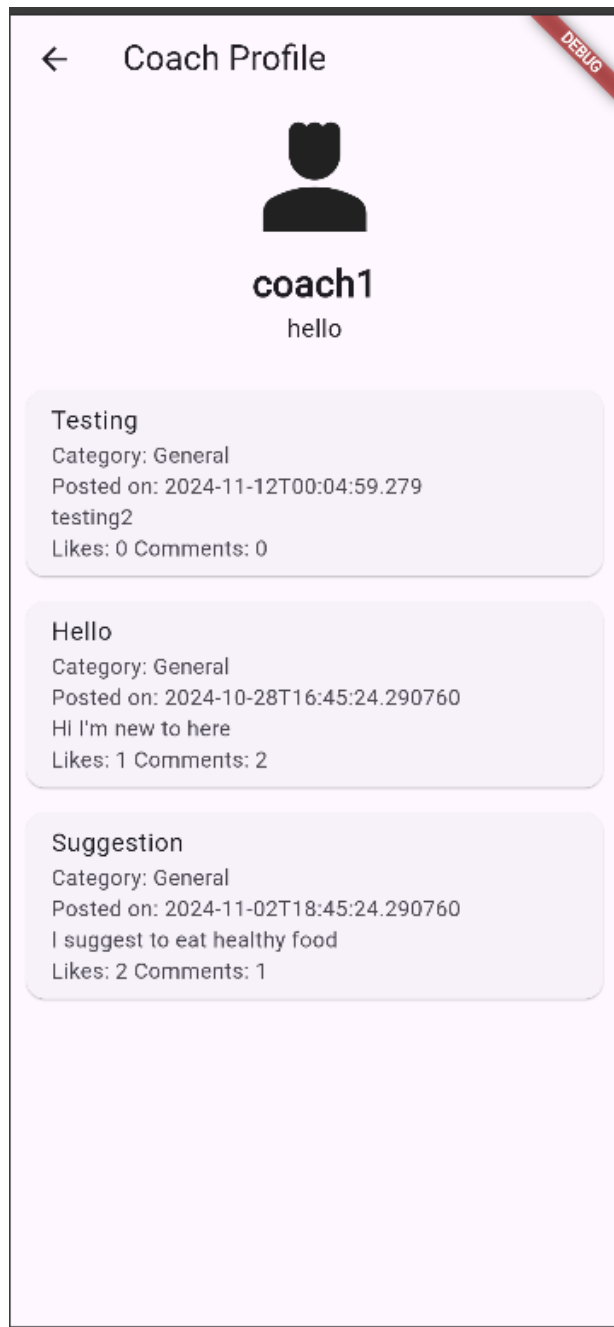


Figure 14 Coach Profile (Search)

After user have search a specific coach name and clicked in. user will be able to see the coach information and the post that posted by the specific coach.

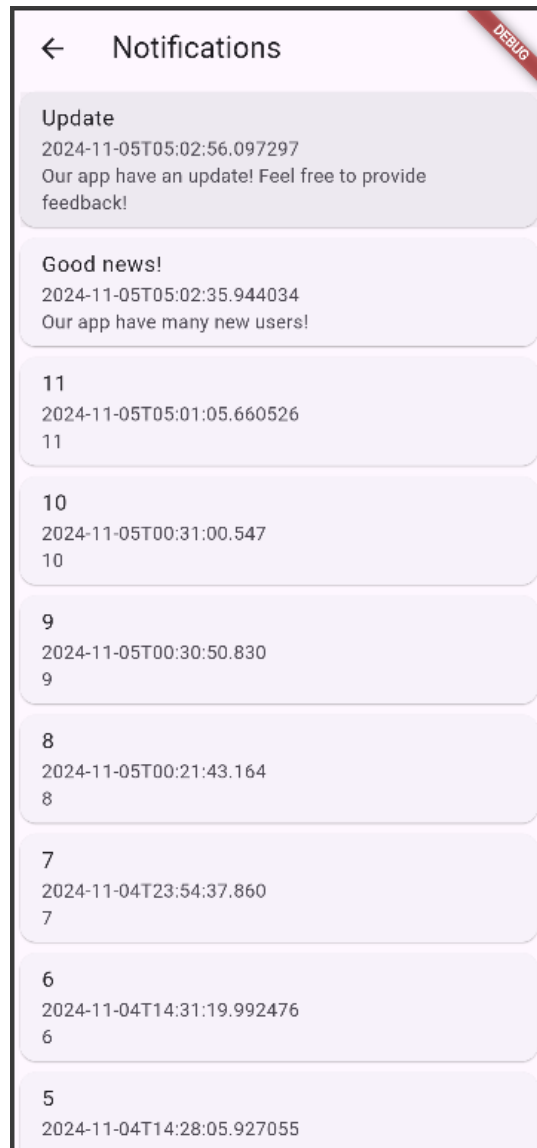


Figure 15 User Notification Page

User will receive the notification on notification by admin when latest information that want to be inform.

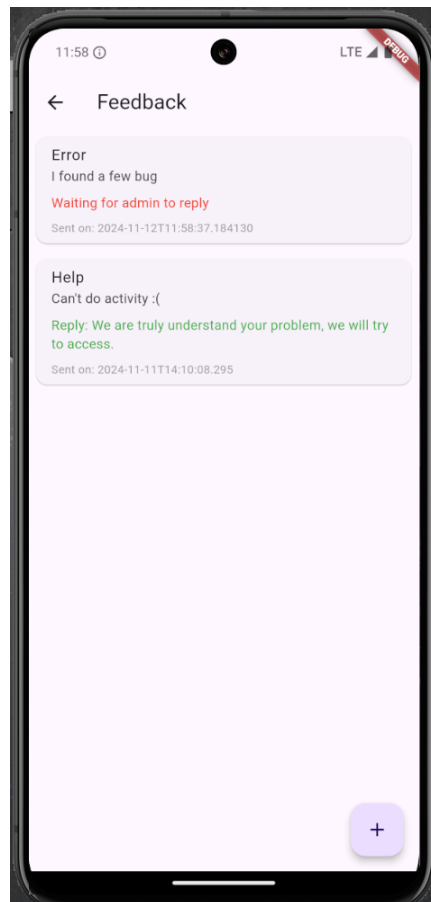


Figure 16 User Feedback Page

On feedback page, user will proceed if there any issue they face in the feedback page and user also able to check if their feedback have been replied or not.

3.3 Coach

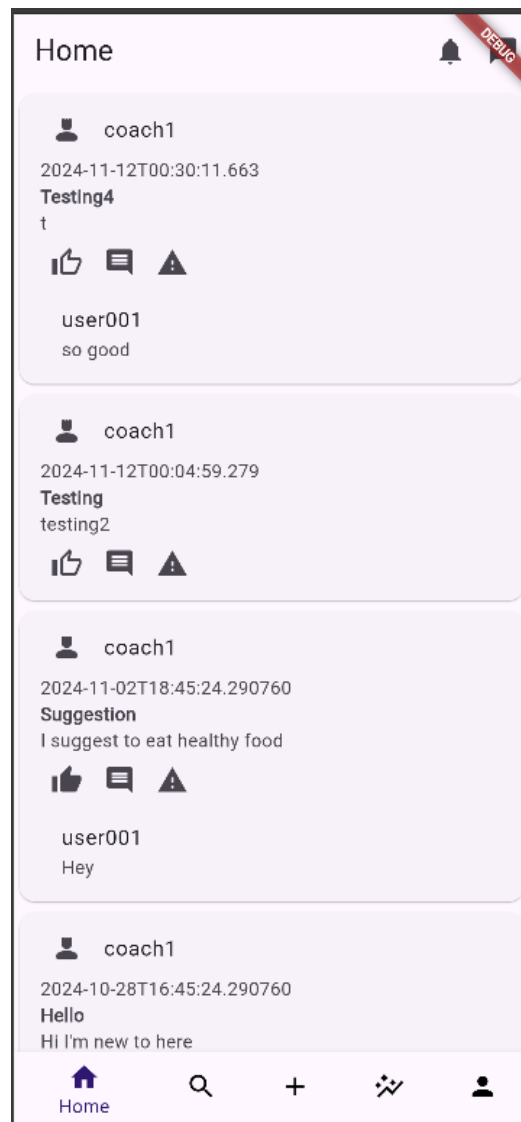


Figure 17 Coach Homepage

Coach homepages provide the post of a coaches sharing their daily or information on coach homepage.



Figure 18 Coach Search Coach

Coach able to search for another specific coach by type their name in the text field and look for the information of the specific coach.

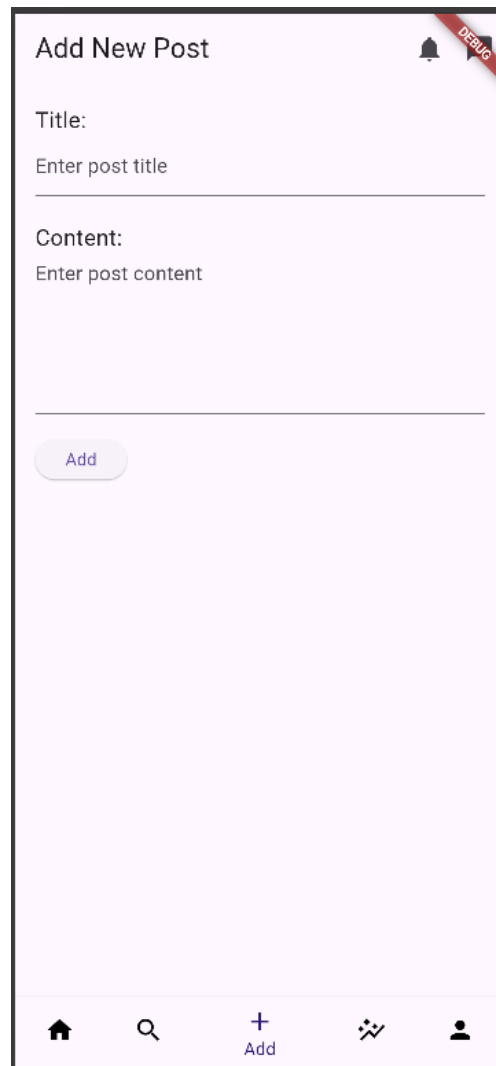
The image shows a mobile application interface for adding a new post. The page has a light pink background. At the top, the title "Add New Post" is displayed in a dark font. To the right of the title is a bell icon and a red ribbon with the word "Disable" in white. Below the title, there are two input fields. The first is labeled "Title:" and contains the placeholder text "Enter post title". The second is labeled "Content:" and contains the placeholder text "Enter post content". Below these fields is a rounded rectangular button with the word "Add" in purple. At the bottom of the screen is a navigation bar with five icons: a home icon, a magnifying glass icon, a plus icon with the word "Add" below it, a share icon, and a profile icon.

Figure 19 Coach Add New Post Page

Coach will post their new post on add new post page. By inserting the information they want to post, click “Add” button will add the new post for the coach and let the public see.

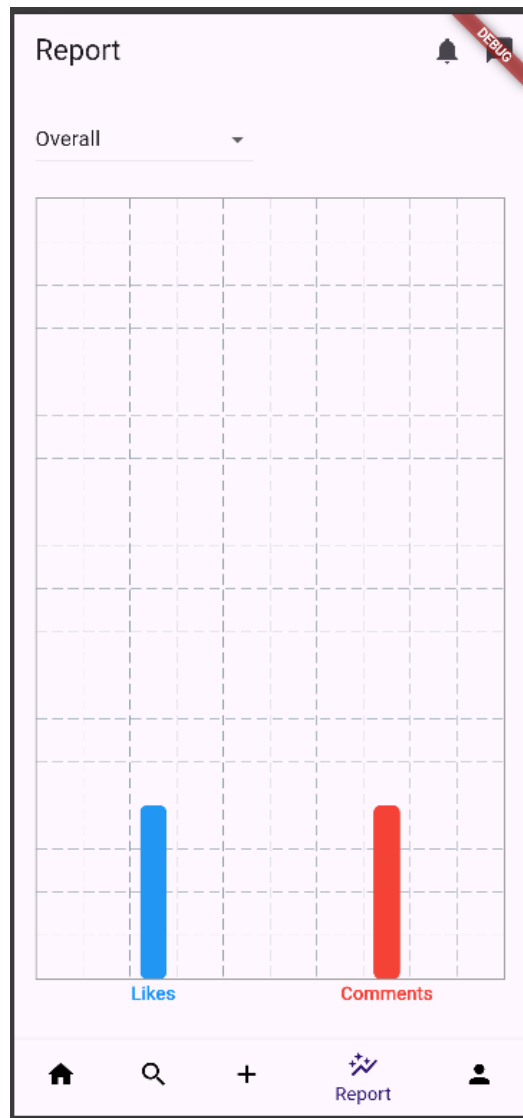
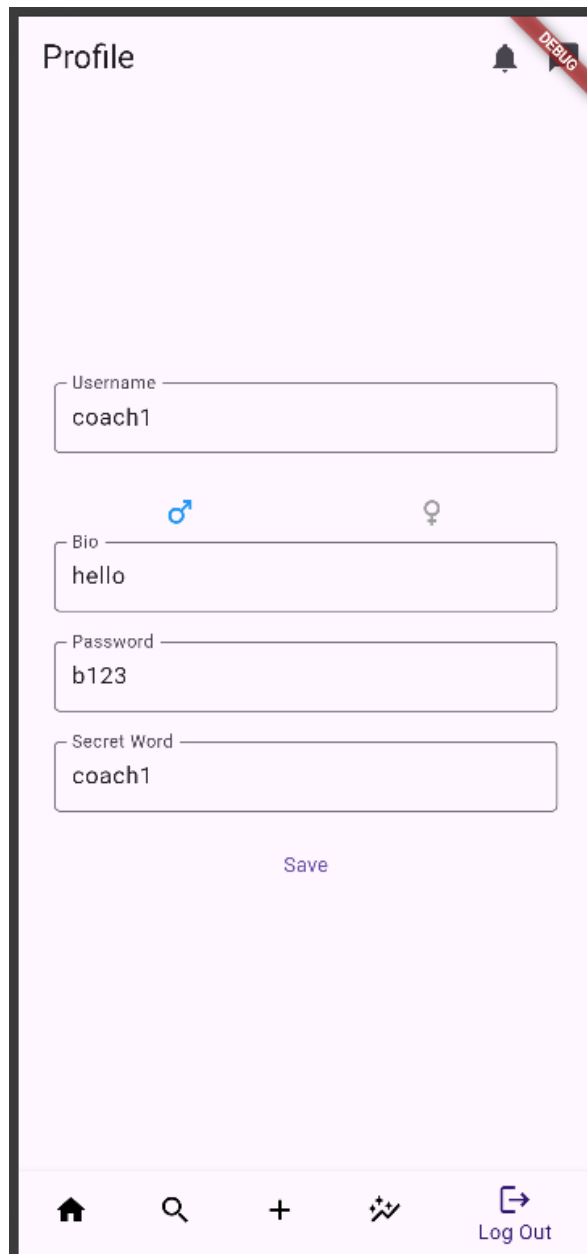


Figure 20 Coach Report Page

Coach will be able to see their report on how many like and comments in the report. Coach also able to choose specific category like order by like or order by comments in which to see the most likes or most comment in report page.



The image shows a mobile application interface for a 'Coach Profile' page. The title 'Profile' is at the top left, and a bell icon and a red 'DEBUG' banner are at the top right. The form contains four input fields: 'Username' with the value 'coach1', 'Bio' with the value 'hello' and gender selection icons (male and female), 'Password' with the value 'b123', and 'Secret Word' with the value 'coach1'. A 'Save' button is centered below the fields. The bottom navigation bar includes icons for home, search, add, and settings, along with a 'Log Out' button.

Profile

Username

coach1

Bio

hello

Password

b123

Secret Word

coach1

Save

Log Out

Figure 21 Coach Profile Page

Coach will be able to view their own personal information in profile page and able to make editing on the profile, with confirmation and click save by saving the latest personal detail in profile page. Coach will also proceed their log out in profile page on the button right button.

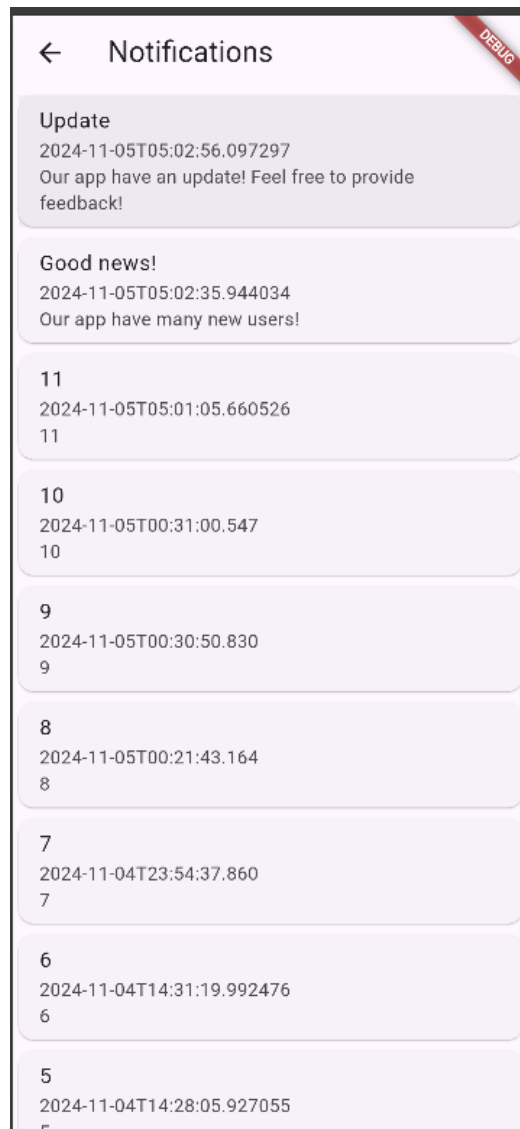


Figure 22 Coach Notification Page

Coach will receive the notification on notification by admin when latest information that want to be inform.

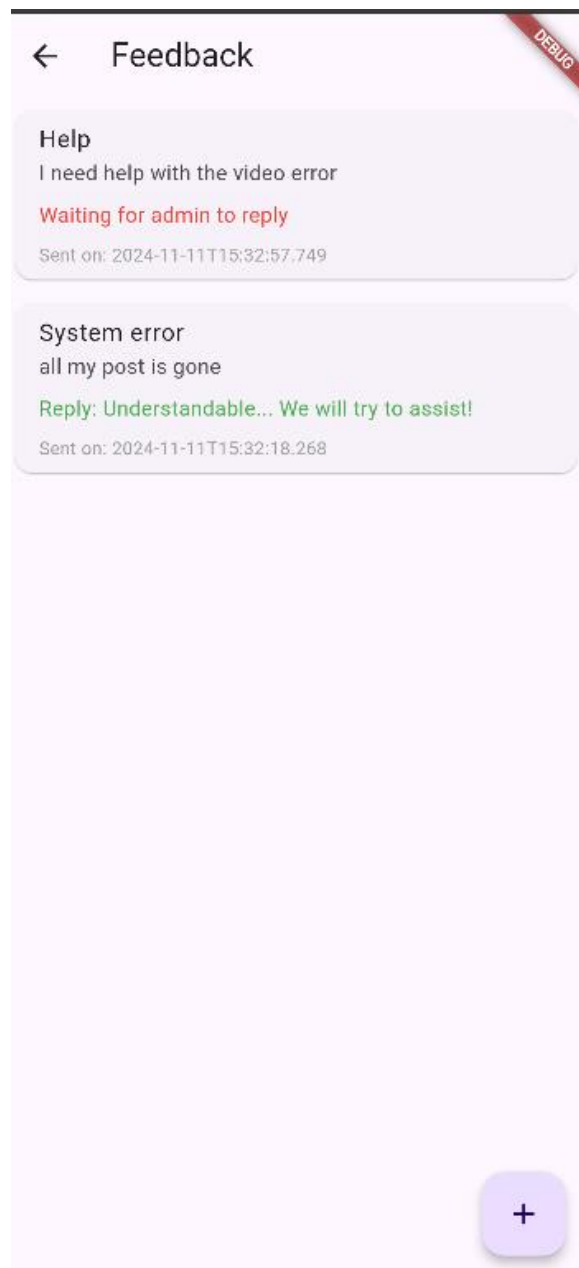


Figure 23 Coach Feedback Page

On feedback page, coach will proceed if there any issue they face in the feedback page and user also able to check if their feedback has been replied or not.

3.4 Admin

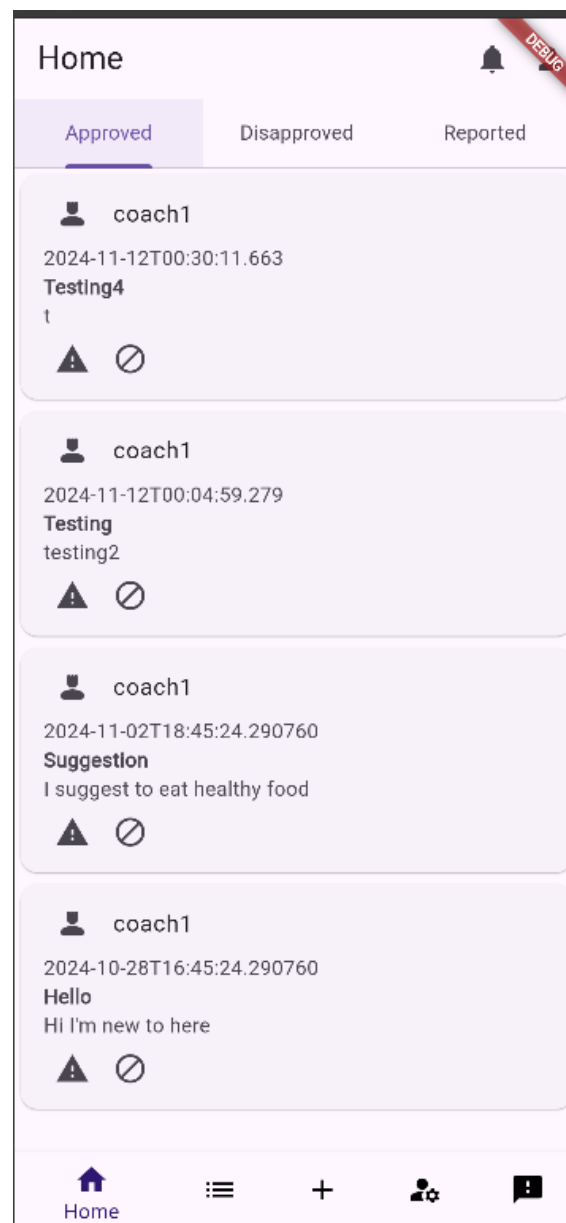


Figure 24 Admin Homepage (Approved)

On admin homepage, admin have the power to verify the good and bad post that posted by the coach and choose whether to disapprove the post or report the post and not let the specific harmful post show in the public.

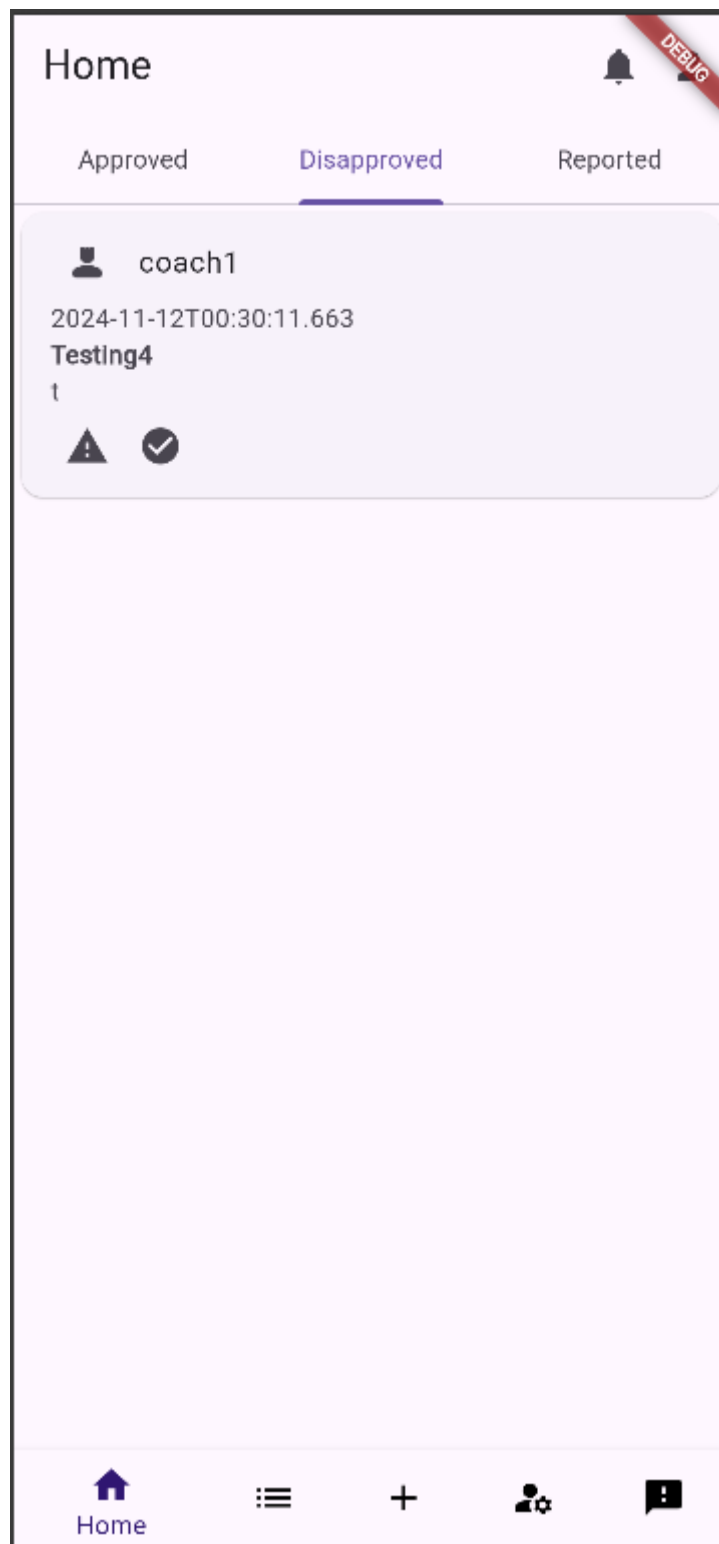


Figure 25 Admin Homepage (Disapproved)

Once admin disapprove the post of the coach, it will proceed to the disapproved page.

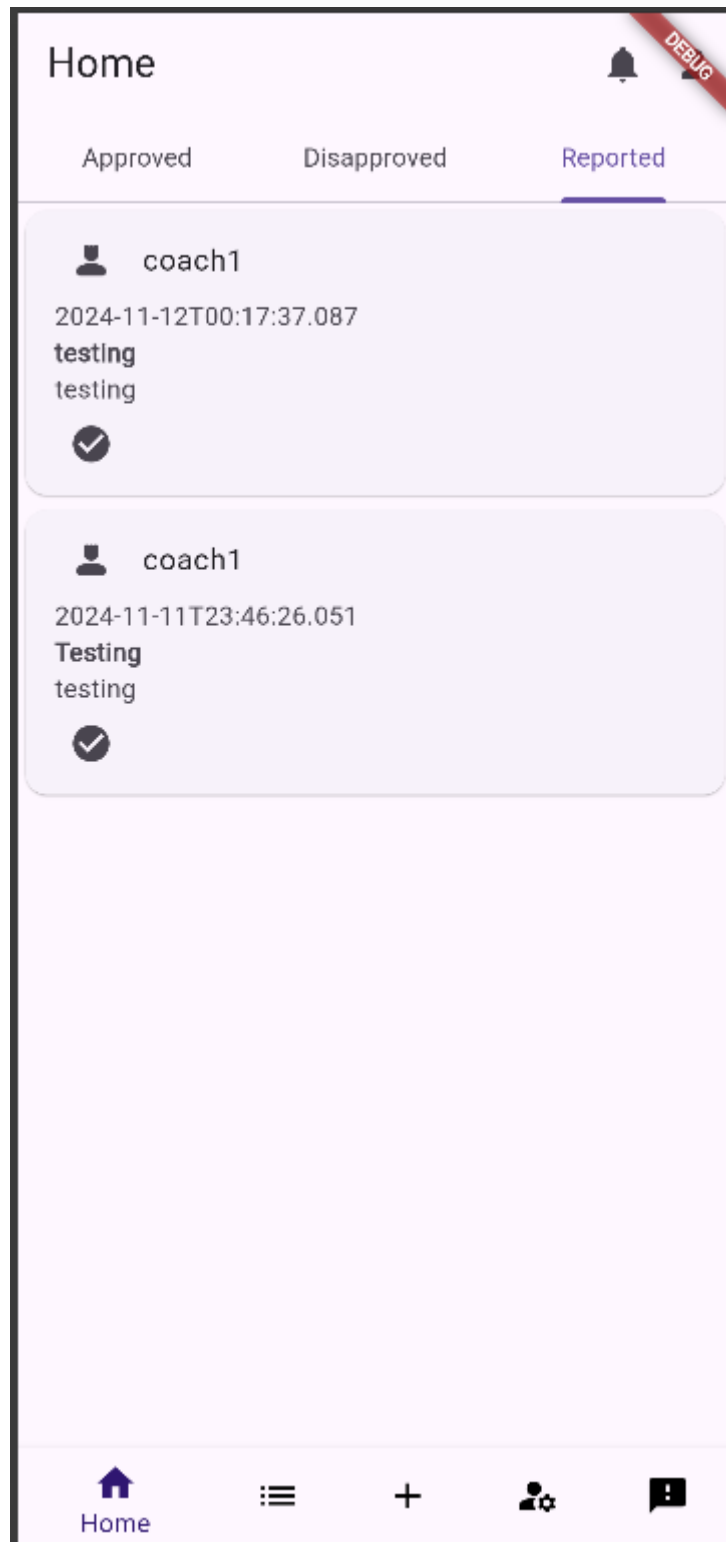


Figure 26 Admin Homepage (Reported)

Once admin report the post of the coach, it will proceed to the reported page.

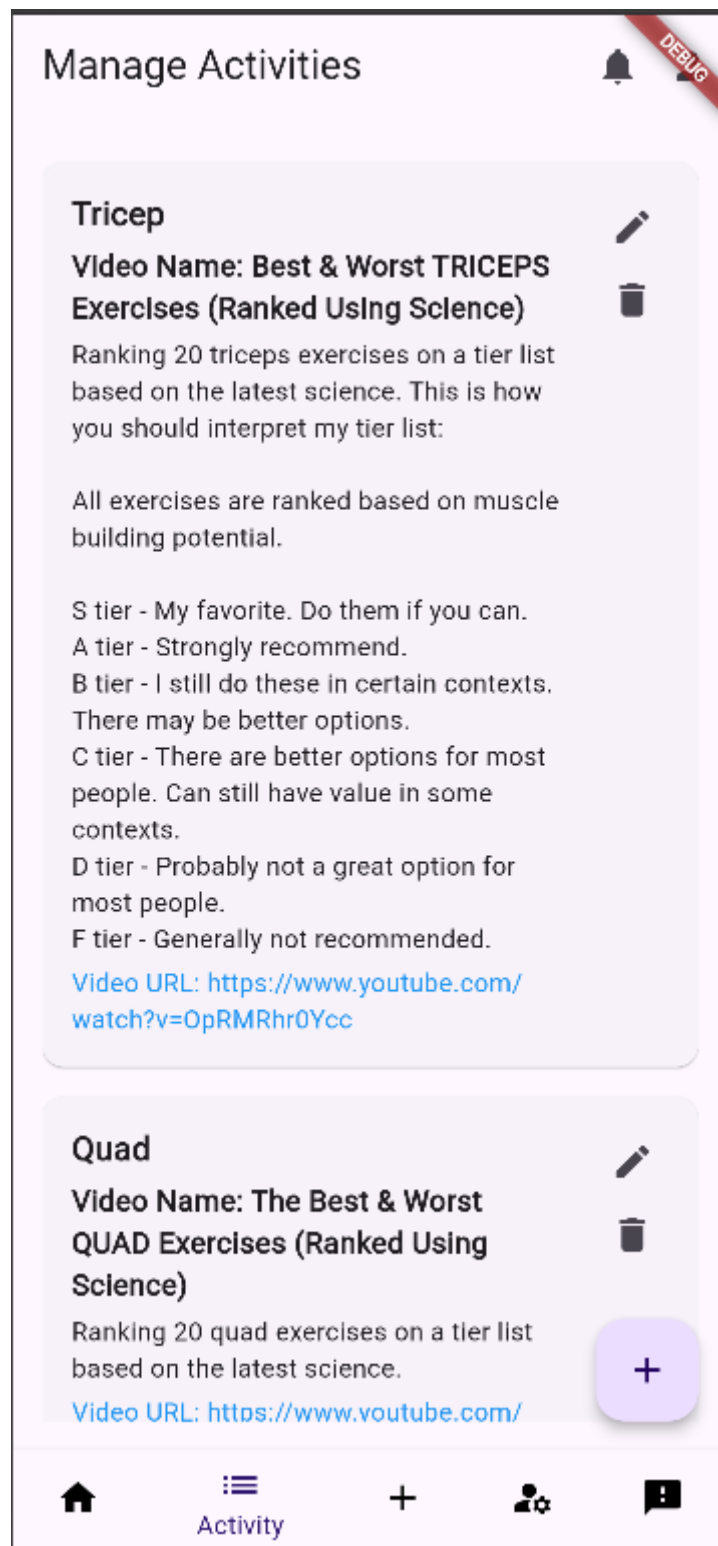


Figure 27 Admin Manage Activities Page

On Manage Activity Page, admin have the function to add, edit, delete and view the activities based on anything needed. For example, admin can edit or delete the specific activity that

have any issue. If there any new activity has to be add, admin can also create a new activity for user to explore more.

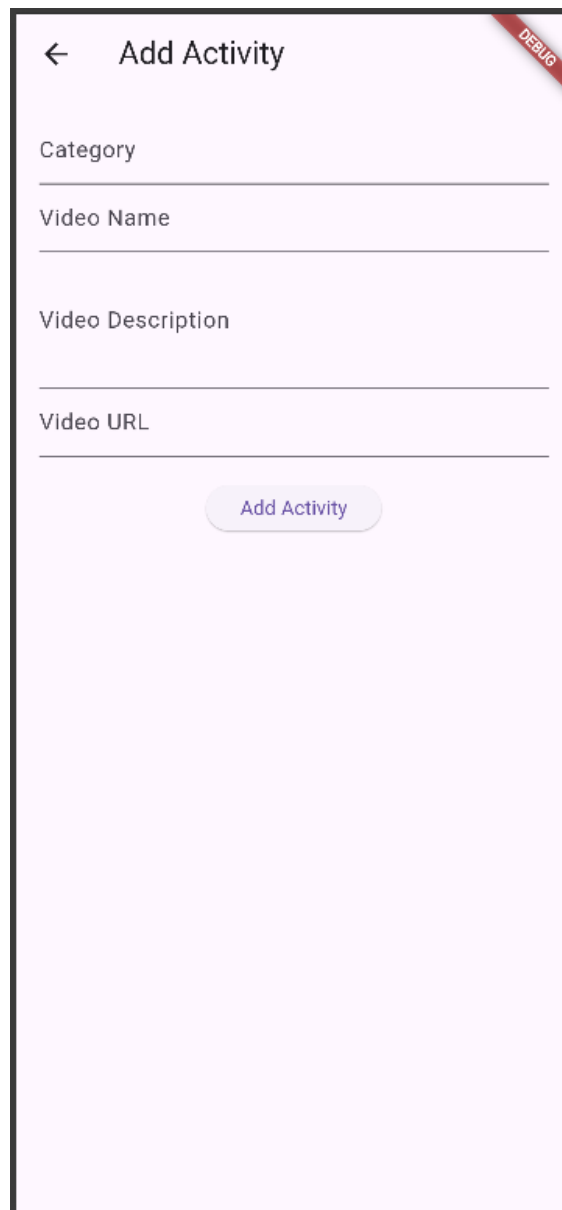
The image shows a mobile application interface for adding a new activity. At the top, there is a header bar with a back arrow on the left and the title 'Add Activity' in the center. A red 'DEBUG' banner is visible in the top right corner. Below the header, there are four text input fields stacked vertically, labeled 'Category', 'Video Name', 'Video Description', and 'Video URL'. At the bottom of the form, there is a rounded rectangular button with the text 'Add Activity' in a purple color.

Figure 28 Admin Manage Activity page (Add Activity)

Admin will fill the information above to add any new activity just by typing in the text field and lastly click on “Add Activity” button to add a new publish activity for user.

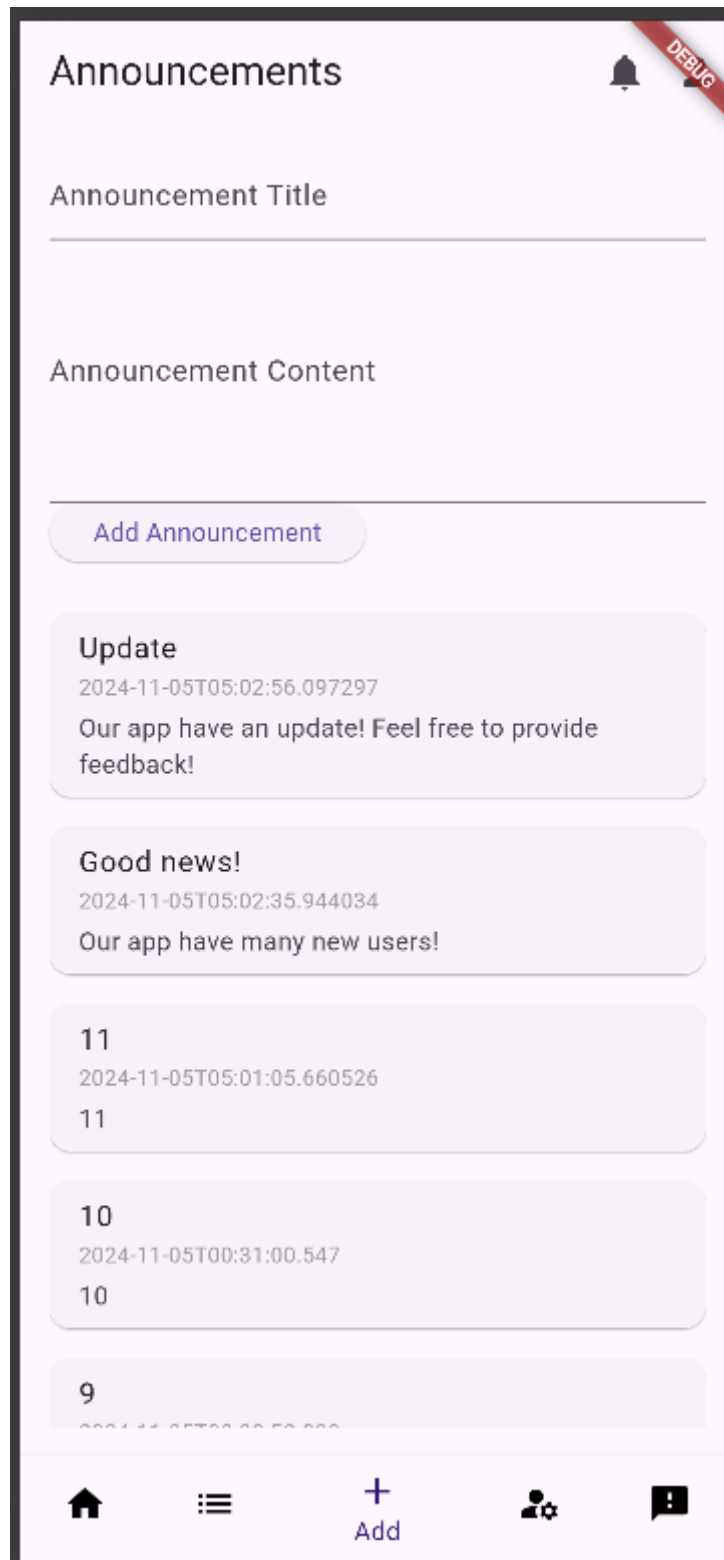


Figure 29 Admin Announcement Page

Admin will post new notification in this page for user, coach and admin to view on notification page about the latest information about this application.

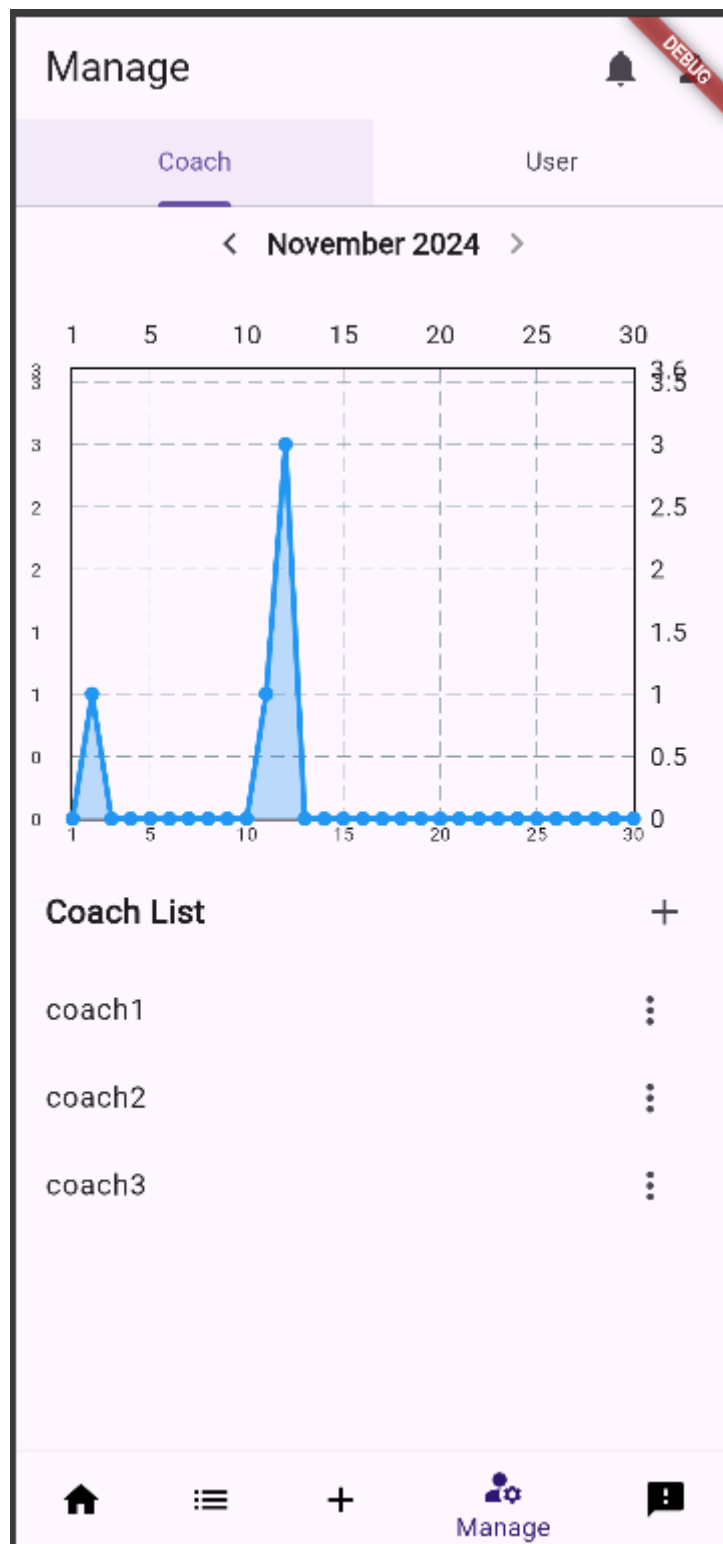


Figure 30 Admin Manage page (coach)

Admin manage page on coach is to view how active is the coach during the specific month. For example, how many total of all the coach have post in the public.

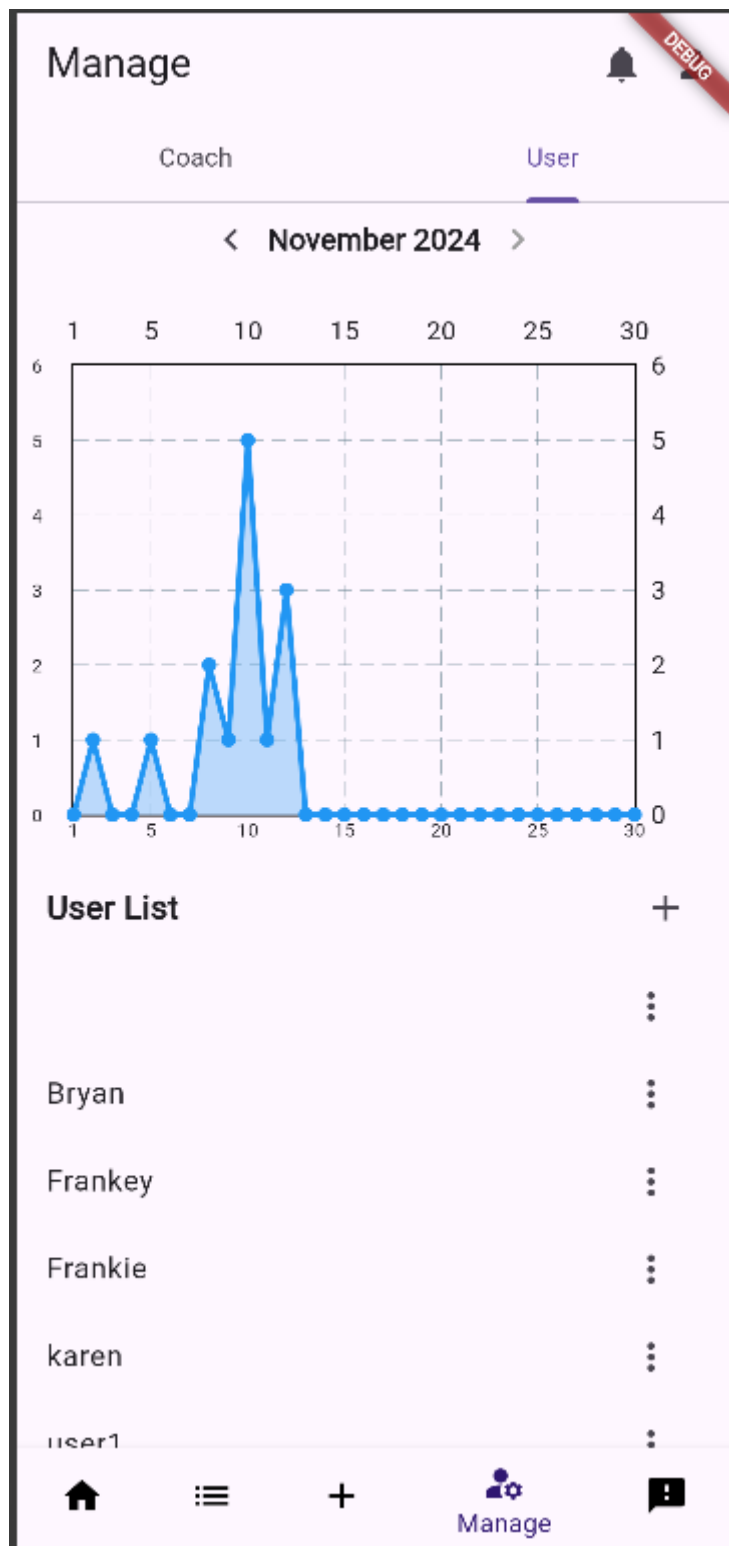


Figure 31 Admin Manage page (User)

Admin able to view the activity of user based on their workout.

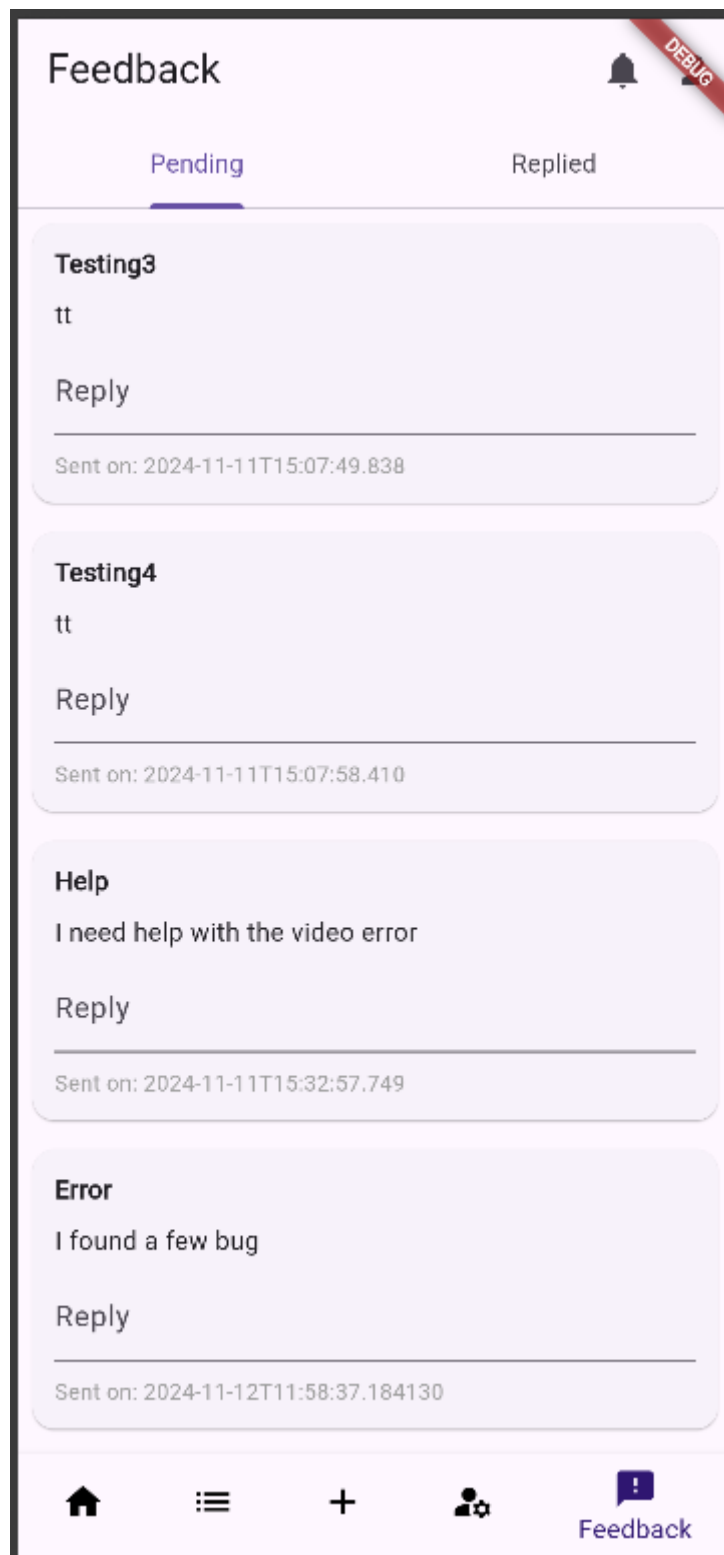


Figure 32 Admin Feedback page (Pending)

Admin able to view on the feedback provided by user or coach which will be appeared on pending category. Admin will have to reply on the pending feedback.

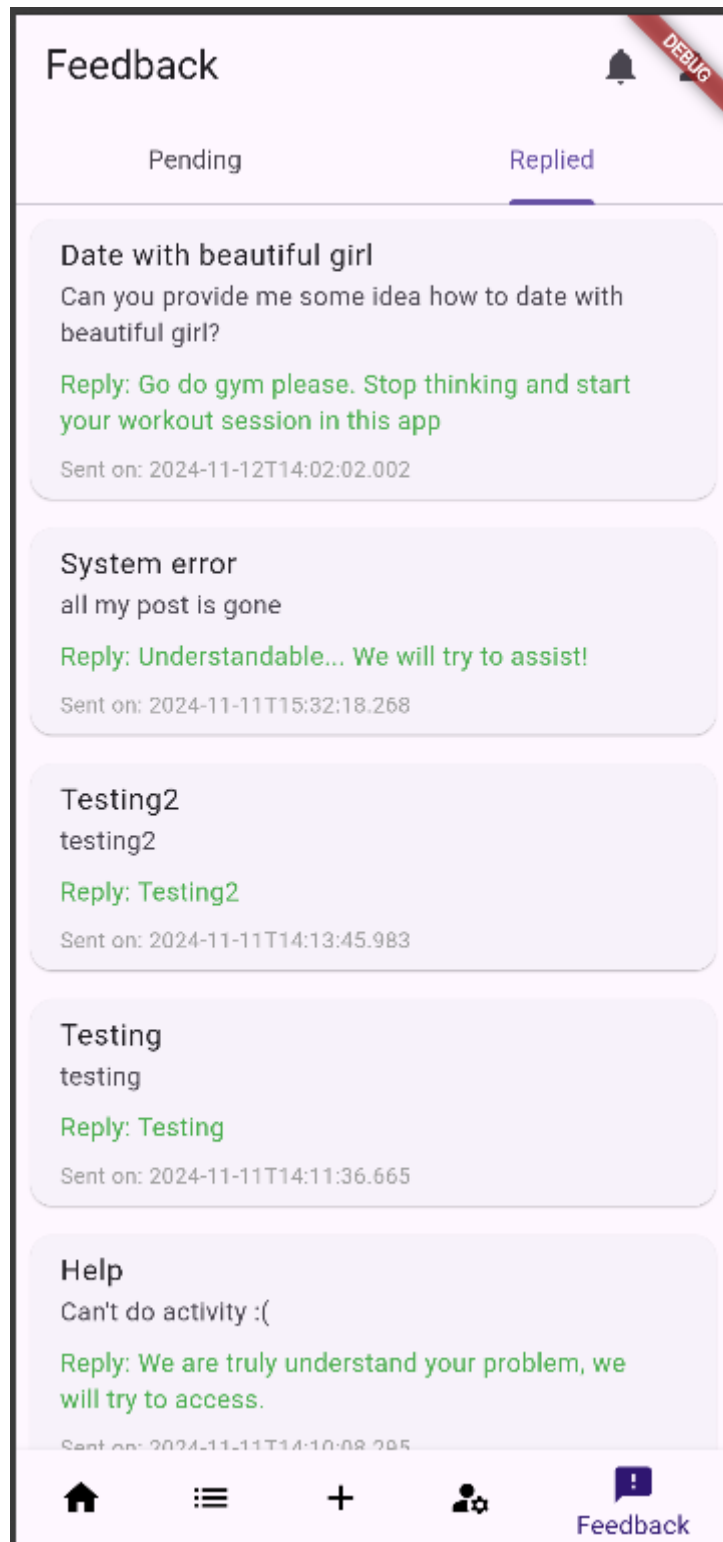


Figure 33 Admin Feedback page (Replied)

After admin have replied to the feedback that provide by the user or coach, it will be appear at replied category.

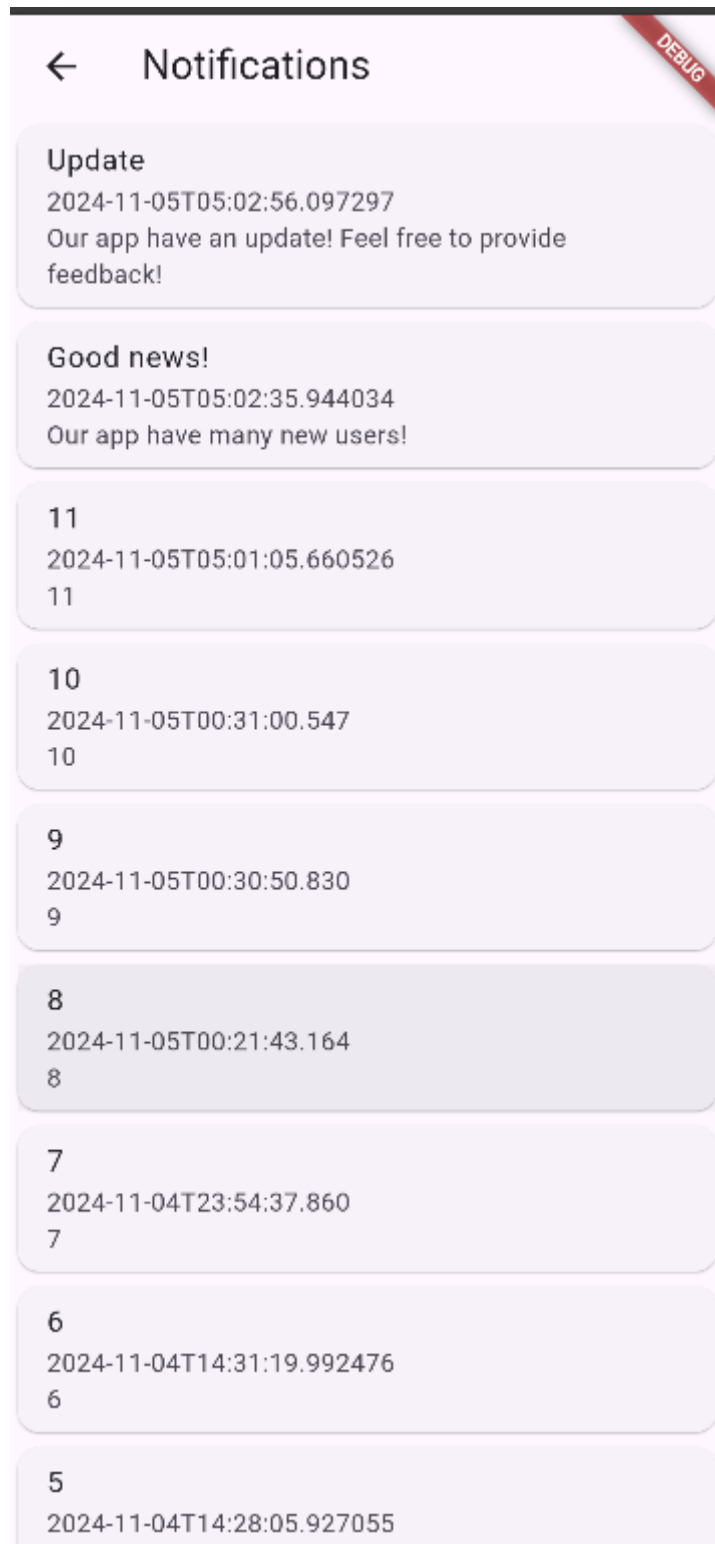


Figure 34 Admin Notification page

Admin will be able to receive notification by admin when latest information that want to be inform.

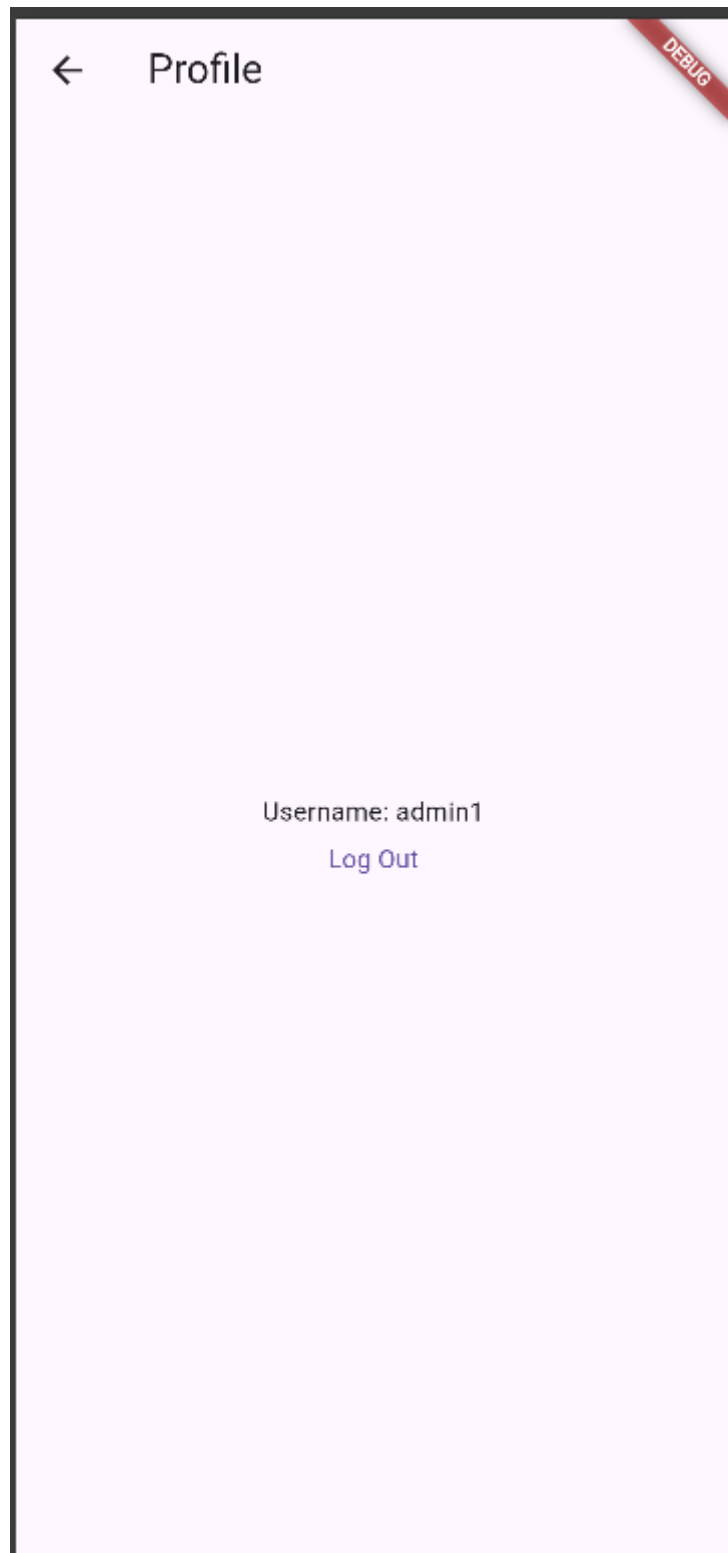


Figure 35 Admin Profile Page

On admin profile page can see username and also log out button for admin to log out.

4.0 Automated System Testing

Robo Test, available in the Firebase Console, is a powerful tool for automated testing of Android applications (Firebase, 2024). It systematically simulates user interactions across the app, performing actions such as clicking buttons, filling text fields, and navigating between screens (Sutanto, 2019). By using Robo Test, we can ensure comprehensive testing coverage, even for complex flows, without the need for manual test scripts. This approach is ideal for identifying potential issues early, as Robo Test thoroughly examines different paths and screens in the app, which helps maintain a high standard of quality.

Passed

11/12/24, 10:23 PM

2m 22s

Portrait

English (United States)

Test Issues

Robo

Logs

Screenshots

Videos

Performance

Accessibility

Crawl duration

2m 16s

Crawl stats ⓘ

Actions

54

Activities

1

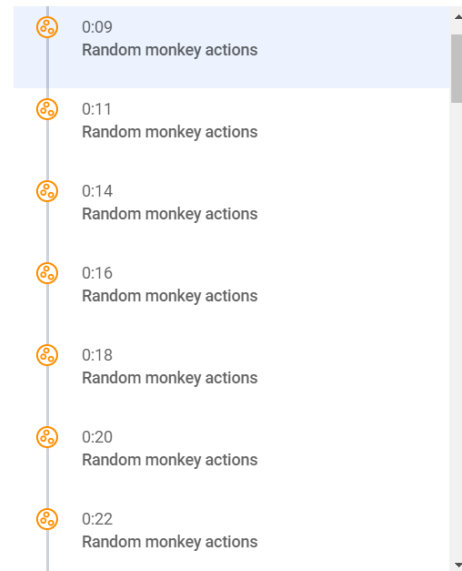
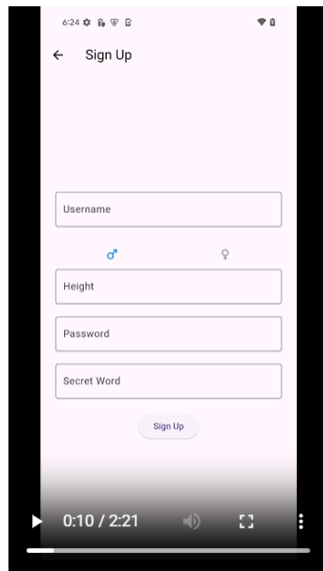
Screens

54

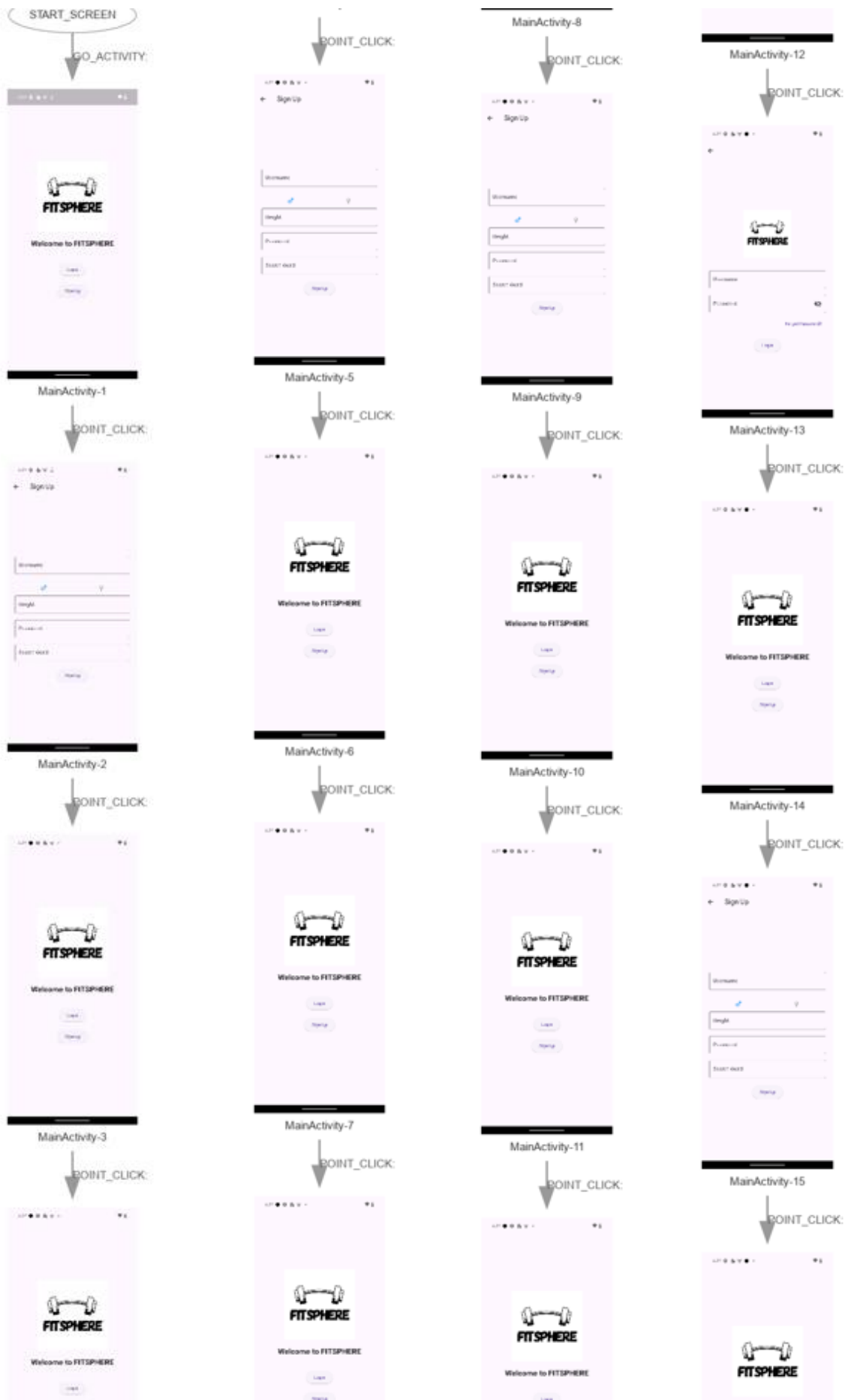
In our test, Robo Test successfully completed the test scenario, with all test cases passing. The entire testing process had a crawl duration of 2 minutes and 16 seconds, efficiently covering various parts of the app, including the login and signup functionalities.

Time to initial display ⓘ Time to full display ⓘ
— 351ms

Performance Over Time



During the login and signup page testing, Robo Test performed multiple random "monkey" actions, simulating a variety of user behaviors. These random actions, such as filling in different text fields, tapping on different areas, and navigating back and forth, helped validate the app's stability and resilience against unexpected user interactions.





POINT_CLICK:



POINT_CLICK:



POINT_CLICK:



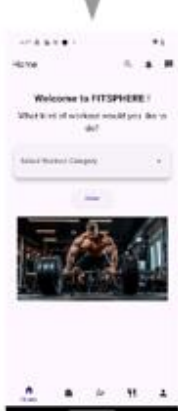
POINT_CLICK:



POINT_CLICK:



POINT_CLICK:



POINT_CLICK:



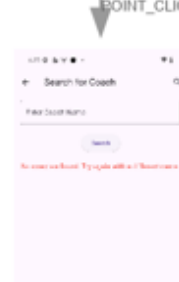
POINT_CLICK:



POINT_CLICK:



POINT_CLICK:



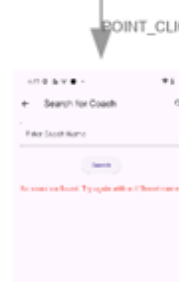
POINT_CLICK:



POINT_CLICK:



POINT_CLICK:







The images provided above represent the crawl graph generated during the testing process. This graph visually outlines the sequence of actions and paths taken by Robo Test, showing the comprehensive steps it conducted to validate the app's functionality and performance. The crawl graph provides a clear, structured view of the testing flow, helping to review and understand the actions that Robo Test performed on the app.

5.0 Conclusion

The development of the FITSPHERE app addresses significant gaps in the current fitness app landscape by providing a comprehensive, user-friendly platform tailored for fitness-conscious individuals, coaches, and admins. By integrating fitness tracking, community interaction, and professional coaching into a single app, FITSPHERE ensures that users can track their progress, access educational resources, and receive personalized guidance, fostering a supportive environment that enhances motivation and engagement. The app empowers coaches with tools to create content, monitor user progress, and offer real-time feedback, while admins benefit from a robust dashboard for efficient content management and user engagement oversight. Through Firebase's real-time capabilities and robust backend, the app facilitates seamless data synchronization, ensuring that all interactions and updates are processed instantly for a smooth user experience. Automated testing with Firebase's Robo Test further assures a high standard of quality, identifying potential issues early and ensuring the app's stability across complex user flows. With these innovative features, FITSPHERE is set to become a valuable asset in the fitness industry, bridging the gap between users and coaches and supporting long-term fitness journeys through accountability, knowledge, and community connection.

6.0 References

Firebase. (11 November, 2024). *Run a Robo test (Android)*. Retrieved from Firebase:
<https://firebase.google.com/docs/test-lab/android/robo-ux-test>

Sutanto, M. (22 July, 2019). *Android Application Testing with Firebase Robo Test*. Retrieved from Medium: <https://medium.com/wantedly-engineering/android-application-testing-with-firebase-robo-test-c674e1754298>

7.0 Appendix

Workload Matrix

	Responsible User	Functionalities
Chong Mun Hei	Coach	<ul style="list-style-type: none"> • Community Interactions • Search Coaches • Post Creation • Post Engagement • Profile Management • Notifications of Announcements • Feedback Management
Karen Guang Sing Qiao	Admin	<ul style="list-style-type: none"> • Manage Posts • Manage Activities • Create Announcements • Manage Users & Coaches • Manage Feedback • Notifications of Announcements
Tang Chye Fong	User	<ul style="list-style-type: none"> • Activity Tracking • Community Interactions • Report Tracking • Nutrition Information • Profile Management • Search Coaches • Notifications of Announcements • Feedback Management