

# **CHAPTER 1**

## **INTRODUCTION**

### **1.1 ABOUT THE PROJECT**

The Competitions Tracker is a mobile application designed to streamline the organization and tracking of competitions for MCA department. It caters to inter-college and intra-college events, including technical competitions such as coding, debugging, and quizzes, as well as non-technical events like cultural activities. The application provides tailored roles for students and staff, enabling students to access competition details, track their participation history, view achievements, and stay updated with real-time notifications. Staff members can create and manage competitions, monitor participation, and analyze event outcomes effectively.

Using Flutter for the frontend and Firebase for the backend, the application ensures a seamless and responsive user experience. Secure login through Firebase Authentication, real-time updates via Firestore, and cloud functions ensure reliability and scalability. A unique points system motivates students by awarding participation and winning points, which contribute to leaderboards that recognize top performers. Notifications keep users informed about registration deadlines, event reminders, and results, while dashboards provide insights into participation trends and individual achievements. With features like real-time notifications and easy access to event schedules, the application eliminates communication gaps and ensures participants are always informed. Additionally, preloaded datasets with event details streamline competition management for staff, while secure data storage in Firebase ensures user privacy and reliability. By integrating a user-friendly design with powerful backend capabilities, the Competitions Tracker is poised to transform how competitions are managed and experienced.

## **1.2 EXISTING SYSTEM**

Competition details are communicated through notice boards, emails, or verbal announcements, often causing delays and limiting accessibility. Students face challenges in staying updated about upcoming events due to scattered and inconsistent information.

Tracking participation history and achievements is unorganized and difficult for students. Event management for staff relies on manual methods like spreadsheets or paper-based systems, which are time-consuming and prone to errors.

Tasks such as scheduling, participant registration, and result tracking lack automation and real-time updates. There is no unified platform to centralize competition data or provide real-time notifications for users. Analytical tools for participation statistics, department performance, and competition trends are absent, making event planning inefficient.

## **1.3 DRAWBACKS OF EXISTING SYSTEM**

- Event details are not easily accessible due to reliance on notice boards, emails, or verbal announcements, leading to missed opportunities for participation.
- There is no single platform to store and manage competition-related information, resulting in scattered and unorganized data.
- Tasks such as participant registration, scheduling, and result tracking are handled manually, making the process time-consuming and error-prone.
- The absence of real-time updates and notifications reduces student engagement and participation rates.
- Students and staff cannot easily access past participation histories, performance data, or achievements, making progress tracking difficult.
- Organizers lack tools to analyze participation trends, department performance, and event success, affecting future planning and decision-making.

## **1.4 PROPOSED SYSTEM**

The Competitions Tracker application will improve the current system by providing a centralized platform to manage and track competitions for MCA department. It will send real-time notifications to keep students and staff updated on upcoming events, registration deadlines, and results. Students can easily view competition details, track their participation history, and monitor achievements, while staff can create and manage events and analyse competition data.

## **1.5 ADVANTAGES OF PROPOSED SYSTEM**

- Centralized platform for easy access and management of competition information.
- Real-time notifications to keep participants updated on event details, deadlines, and results.
- Efficient competition management for staff with reduced manual effort and errors.
- Enhanced student engagement through a points system and leaderboards.
- Data-driven insights with analytics and reports to improve event planning.
- Scalable design to support competitions across all college departments.
- Streamlined tracking of participation history and achievements for students.
- Automated updates and reports, reducing manual work and ensuring accuracy.

## **CHAPTER 2**

### **SYSTEM ANALYSIS**

#### **2.1 IDENTIFICATION OF NEED**

The current system for managing competitions in colleges is fragmented, manual, and inefficient, leading to communication delays, disorganized data, and limited participation tracking. There is a need for a centralized platform that can streamline the management of both technical and non-technical competitions for MCA department. Students and staff need an easy way to access event details, track participation, and stay updated with real-time notifications. Additionally, there is a demand for a system that can automate tasks like registration, result tracking, and data analysis, reducing the risk of errors and saving time. The Competitions Tracker application addresses these gaps by providing a comprehensive solution that improves communication, engagement, and efficiency, ultimately enhancing the overall competition experience for both students and staff.

#### **2.2 FEASIBILITY STUDY**

A feasibility study is essential to assess whether the proposed Competitions Tracker application can be successfully developed and implemented in terms of cost, operation, and technology. The purpose of this study is to evaluate the practicality of the project from three key perspectives: economic, operational, and technical. By analyzing these factors, we can determine whether it is financially viable, if it aligns with operational goals, and whether the necessary technology and resources are available for successful implementation. This study ensures that the proposed system will be both sustainable and efficient in meeting the needs of students and staff for MCA department.

Three considerations involved in feasibility are

- Economic Feasibility
- Operational Feasibility
- Technical Feasibility

### **2.2.1 Economic Feasibility**

The development of the Competitions Tracker application is a cost-effective solution for managing competitions for MCA department. With the use of Flutter for cross-platform development, the application can be deployed on both Android and iOS, reducing the need for separate development for each platform. Firebase offers a scalable backend with a pay-as-you-go pricing model, which helps keep initial costs low and scales with the number of users. The application eliminates the need for physical infrastructure or paper-based systems, resulting in long-term savings. Additionally, the application can reduce manual labor for staff, improving efficiency and reducing operational costs over time.

### **2.2.2 Operational Feasibility**

The Competitions Tracker application is highly operationally feasible, as it is designed to streamline current competition management processes. Students and staff can access the application with minimal training due to its user-friendly interface. The application's features—real-time notifications, competition tracking, and automated report generation—make the process more efficient, freeing up staff from manual administrative tasks. The system's scalability ensures it can handle the growing number of participants and events across various departments. Its cloud based infrastructure ensures minimal maintenance needs and can be easily updated to accommodate future features.

### **2.2.3 Technical Feasibility**

The technical feasibility of the Competitions Tracker application is solid, as it is built using widely adopted technologies—Flutter for frontend development and Firebase for the backend. Flutter’s cross-platform capabilities ensure the application runs smoothly on both Android and iOS. Firebase provides a reliable backend infrastructure, offering real-time databases, authentication, and notifications, which are essential for the application’s functionality. The application can handle a significant number of users and competitions, and the integration of tools like Power BI for reporting is technically achievable. The application is designed to be scalable, allowing for easy expansion and updates in the future without significant technical hurdles.

## **2.3 SOFTWARE REQUIREMENTS SPECIFICATION**

The Software Requirements Specification (SRS) for the Competitions Tracker application provides a detailed description of the software's intended functionality, features, and system requirements. This document serves as a comprehensive guide for the development, implementation, and maintenance of the application, ensuring all stakeholders—developers, project managers, and users—have a clear understanding of the application's objectives and technical needs. The Competitions Tracker application is designed to streamline the management of both technical and non-technical competitions across various for MCA department helping students and staff to track, participate, and organize events more efficiently. This SRS outlines the hardware and software requirements necessary to support the application's functionalities and ensure its smooth operation across all platforms. By establishing clear requirements, this document provides a foundation for development, testing, and future upgrades.

### 2.3.1 Hardware Requirements

#### 1. Server Requirements (for Backend)

- **Processor:** Multi-core processor with a minimum speed of 2 GHz.
- **RAM:** At least 8 GB of RAM to handle data processing and storage needs.
- **Storage:** Cloud-based storage (provided by Firebase) for storing user data, competition details, and multimedia files.
- **Network:** Reliable high-speed internet connection for real-time data synchronization and handling user requests

#### 2. Client Device Requirements

- **Mobile Devices:** Devices running Android 5.0 (Lollipop) or higher, and iOS 11.0 or higher.
- **Screen:** Mobile devices with a screen size of 5 inches or more for a better user experience.
- **Internet Connection:** Active internet connection required for data synchronization, notifications, and accessing the application's features.

#### 3. Backup and Recovery

- Cloud-based backup solutions for regular data backups, ensuring that all competition and user data is stored safely with minimal risk of loss.

### 2.3.2 Software Requirements

The software for the Competitions Tracker application is selected based on factors like frontend environment compatibility, flexibility in coding languages, and the suitability of backend technology for real-time data processing and scalability.

Scripting Language	: Dart
Frontend	: Flutter
Backend	: FireBase
Database	: FireStore
Tool	: Android Studio

#### FRONT END

The frontend of the Competitions Tracker application is developed using Flutter, a platform framework that allows the application to be built specifically for Android devices. Dart, the programming language used with Flutter, handles the application's logic, user interface, and interactions, ensuring fast performance and smooth animations. The application's interface is designed to be intuitive and responsive, providing users with easy access to competition details, registration, and tracking of participation. Flutter's rich set of pre-built widgets is used to create visually appealing layouts, including interactive dashboards, competition details pages, and real-time notifications.

#### BACK END

The backend of the Competitions Tracker application is powered by Firebase, a comprehensive cloud platform that provides essential services for managing user data, real-time interactions, and notifications. Firebase's Firestore is used as the NoSQL database, allowing the application to store and retrieve data such as competition details, user information, and participation records in real time. Firebase Authentication ensures secure login and user management for both students and staff. Additionally, Firebase Cloud Messaging is employed to send real-time push notifications to users about competition updates, deadlines, and results. The backend is designed to be scalable, secure, and easy to integrate with the frontend, allowing



for efficient data management and smooth communication between the application and its users.

## **DATA BASE**

Firestore allows the application to store and manage data in a flexible, scalable manner, supporting dynamic updates without requiring complex server-side logic. It stores various types of data, including user information (students and staff), competition details, registration data, and results. Firestore ensures that data is synchronized in real time across all devices, providing up-to-date information to users as they interact with the application. The database's structure supports efficient querying and data retrieval, which is essential for features like tracking participation history, displaying competition schedules, and updating user rankings. Additionally, Firestore's integration with Firebase Authentication provides secure data access, ensuring that only authorized users can modify or access sensitive information.

## **CHAPTER 3**

### **SYSTEM DESIGN**

#### **3.1 MODULE DESCRIPTION**

The Competitions Tracker application is divided into several interconnected modules, each responsible for specific functionalities to ensure smooth operation and usability. These modules are designed to work collaboratively, providing a comprehensive platform for managing inter- and intra-college competitions.

The project contains following modules:

1. Student Home Page
2. Competitions Registration
3. Admin Home Page
4. Student Achievement
5. Trending Participant

##### **Student Home Page**

The Student Home Page Module serves as a centralized dashboard for students to manage their profiles, view active competitions, and track their participation history. Key features include a profile view displaying personal details, a list of ongoing competitions available for registration, and a tab to browse past competitions and record participation outcomes, such as ranks or victories.

## **Competitions Registration**

The Competitions Registration Module streamlines the process of registering for competitions, ensuring ease of use for students and accurate data handling for organizers. This module allows students to view available competitions, fill out registration forms, and submit their entries seamlessly. Key features include a user-friendly interface for form filling, where students can input necessary details like name, department, team members (if applicable), and competition-specific requirements.

## **Admin Home Page**

The Admin Home Page Module offers administrators a management interface to oversee competitions and student activities. Admins can add, view, and organize competitions while tracking student participation and achievements across events.

## **Student Achievement**

The Student Achievement Module focuses on recording and verifying student accomplishments in competitions. It allows winners to register their achievements, upload supporting documents such as certificates, and maintain a transparent record of successes.

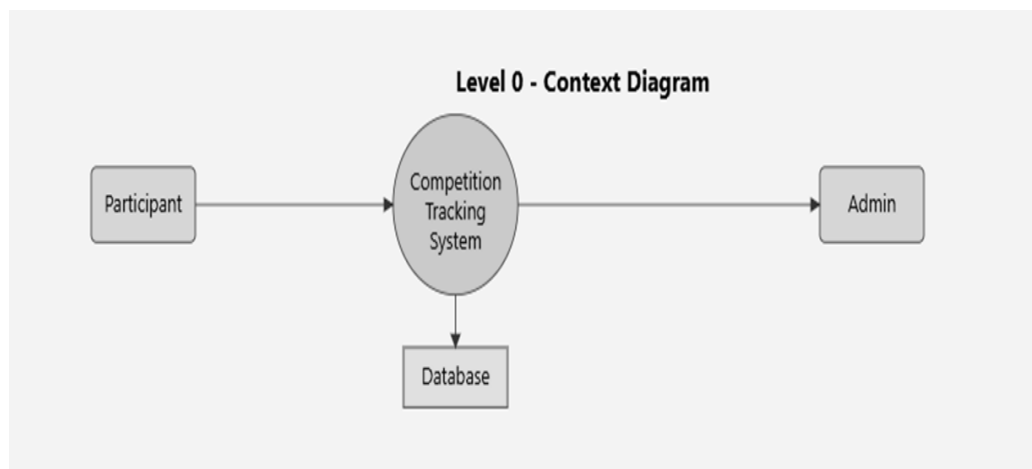
## **Trending Participant**

Finally, the Trending Participant Module highlights top-performing students by calculating points based on participation and competition rankings. It displays a leaderboard featuring the highest-scoring participants, fostering motivation and healthy competition among students.

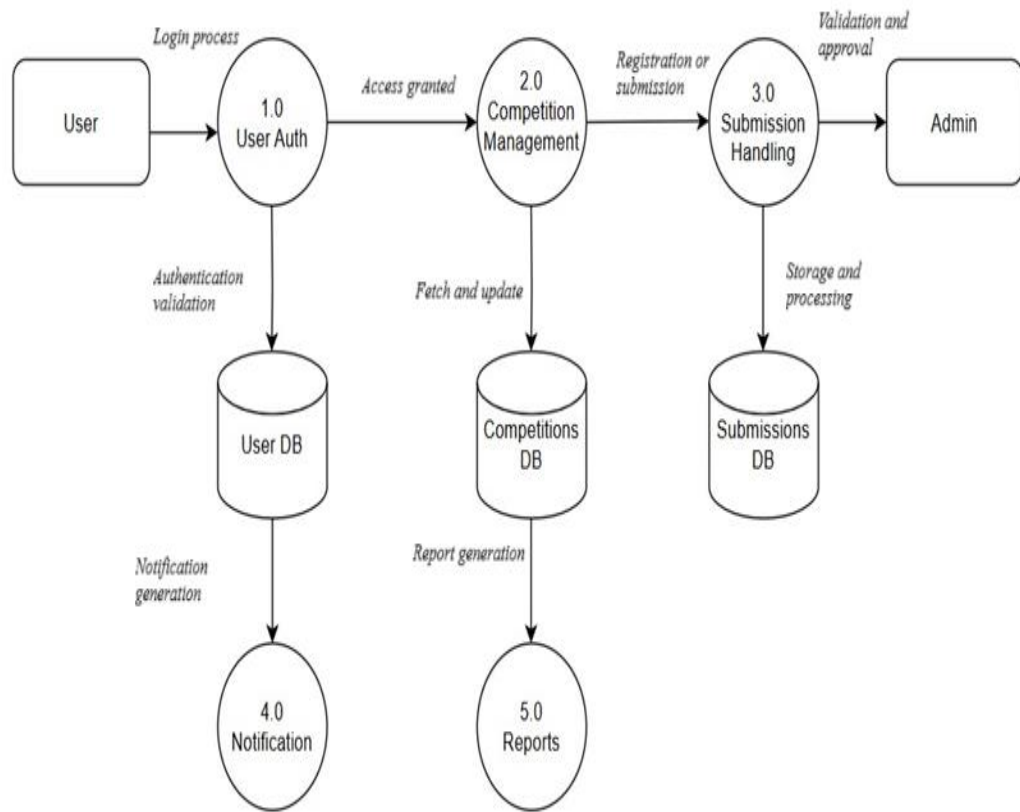
### 3.2 DATA FLOW DIAGRAM

The Data Flow Diagram provides information about the inputs and outputs of each entity and process itself.

#### LEVEL 0



**Figure 3.1 Level 0**

**LEVEL 1****Figure 3.2 Level 1**

### 3.3 DATABASE DESIGN

Database design is a critical process in software development, particularly for application like the Competitions Tracker. In this project, the goal is to organize and structure the data in a way that facilitates efficient storage, retrieval, and management of competition-related information. For Firebase as the backend, the database design will follow a NoSQL approach. Here, instead of using tables and rows like in a relational database, data is stored in collections and documents. This structure offers flexibility, scalability, and the ability to store hierarchical data.

- **Users Collection:** Stores documents representing individual students and staff. Each document includes user-specific information like name, email, role, and participation history.
- **Competitions Collection:** Stores documents for each competition. A document contains attributes like competition name, date, type (technical or non-technical), and a list of participants.
- **Participation Subcollection:** Nested within each competition document or within user documents to track a student's participation in various competitions. It records details such as points earned, rank, and achievements.
- **Achievements Subcollection:** Stores details about awards or prizes won by users in specific competitions.

#### Database Integration

Database integration refers to the process of connecting the application with the database backend. In the case of Firebase, the integration involves setting up Firebase's Cloud Firestore to store, retrieve, and manage the application's data. For Flutter, Firebase offers a set of SDKs that allow smooth communication between the mobile application and the Firebase database. Through Firebase's Firestore SDKs, the application can:

- Read and write user data (e.g., registration details, participation records).
- Manage competition data and updates in real-time.

## Data Independence

Data independence refers to the ability to change the schema or structure of the database without affecting the application logic. In traditional relational databases, changing the schema (e.g., adding a new column to a table) often requires modifications to the application code.

With Firebase and its NoSQL structure, data independence is achieved to some extent because:

- Firebase allows for flexible data models, enabling changes to documents or collections without significant alterations to the application's code.
- New attributes can be added to a document or a subcollection without disrupting other parts of the database.
- Firebase's real-time synchronization ensures that any structural changes made in the backend reflect immediately on the front end.

## Data Security

Data security is a key concern in any application that handles sensitive information, such as user details, competition results, and achievements. Firebase provides robust security features to ensure data integrity, privacy, and confidentiality.

Key aspects of data security for Firebase include:

- **Authentication:** Firebase Authentication ensures that only authorized users (students, staff, admins) can access specific parts of the application.
  - For example, students can only view their participation history, while admins have the authority to create and manage competitions.
- **Firestore Security Rules:** Firebase allows you to set rules for who can read or write to the database. These rules ensure that:
  - Users can only access their own data (e.g., a student cannot view other students' competition results).

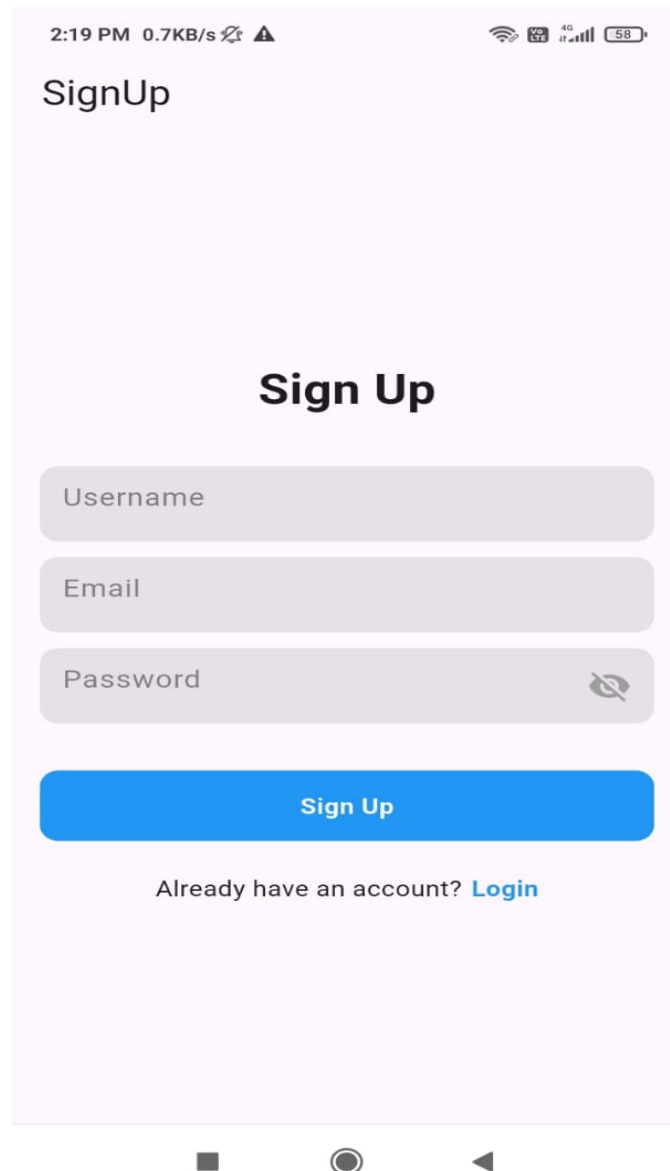
- **Data Encryption:** Firebase encrypts all data both in transit and at rest, ensuring that sensitive data is protected from unauthorized access.
- **Role-Based Access Control:** Firebase allows the application to implement different user roles (e.g., admin, student), restricting access to different levels of the database based on the user's role.

### 3.4 INPUT DESIGN

Input design refers to the process of defining the input requirements and designing the system components that allow users to enter data into the application. It ensures that data is entered in an efficient, user-friendly, and accurate way, providing a smooth interaction between users and the system. In the context of the Competitions Tracker application, input design is critical to ensure that students and staff can easily register for competitions, submit entries, and track their participation.

The input design for the "Add Competition Page" is simple, intuitive, and visually appealing. It features rounded rectangular text fields for entering details like the competition name, optional description, image URL, date, and place, with placeholders inside each field to guide the user. Dropdown menus are provided for selecting predefined options such as competition category, level, and registration fee, ensuring consistency and ease of use. Optional fields like "Details," "Place," and "Registration Fee" allow flexibility without overwhelming the user. The layout is clean, with vertically stacked inputs and adequate spacing for better readability and navigation. The page design follows a soft purple theme with a gradient header, maintaining a modern and cohesive look while focusing on functionality. Additionally, the form likely includes visible "Submit" and "Cancel" buttons to allow users to confirm or discard their entries with ease. The minimalistic layout, aligned with the application soft purple theme, maintains consistency with the overall design while providing clarity and functionality. This balance ensures a user-friendly and efficient interface for adding competition details.

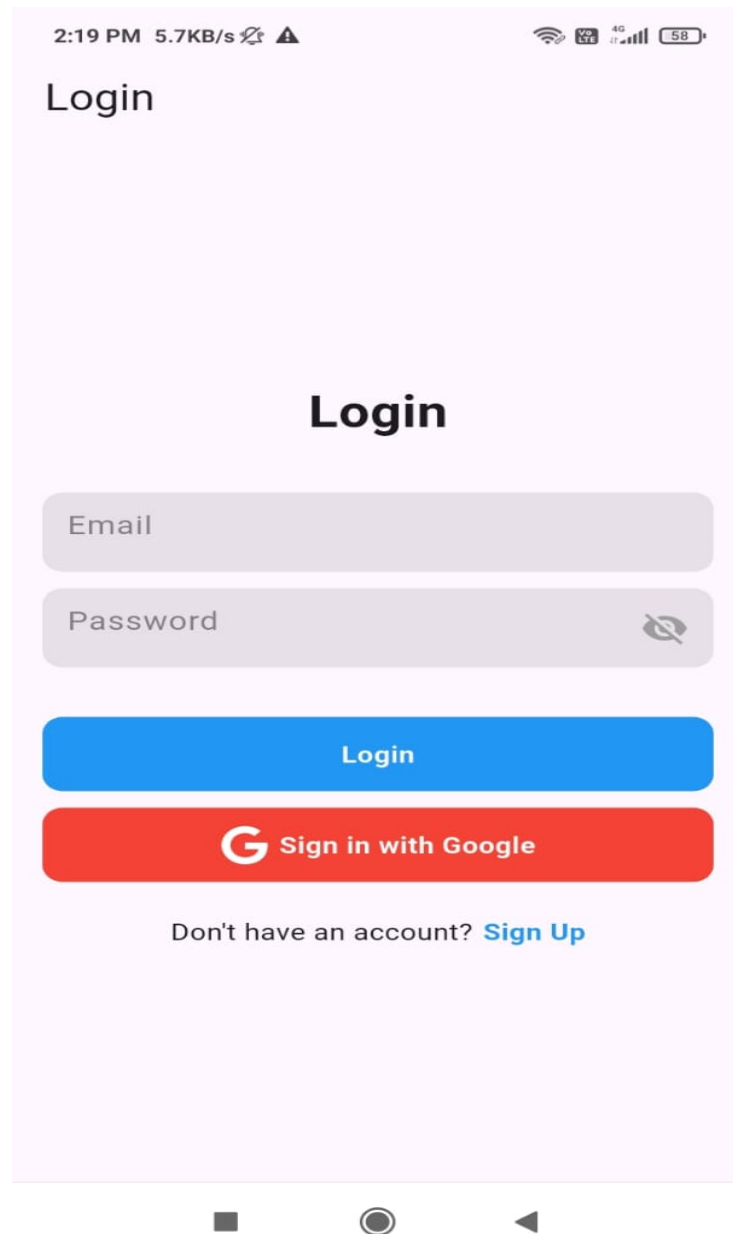




The image shows a mobile application interface for a sign-up page. At the top, the status bar displays the time as 2:19 PM, a data speed of 0.7KB/s, and various connectivity icons. The app's title bar shows 'SignUp'. The main heading is 'Sign Up' in a large, bold, black font. Below the heading are three input fields: 'Username', 'Email', and 'Password'. The 'Password' field includes a toggle icon for visibility. A prominent blue button labeled 'Sign Up' is positioned below the form fields. At the bottom, a link reads 'Already have an account? Login'. The interface is set against a light pink background, and the bottom of the screen shows standard Android navigation icons.

**Figure 3.4.1 Students/admin Sign Up page**

The Sign-Up Page will provide secure way for users to register and access the app. Users will select their role as either a student or staff during the registration process. The form will include fields such as name, email, and a password creation section.

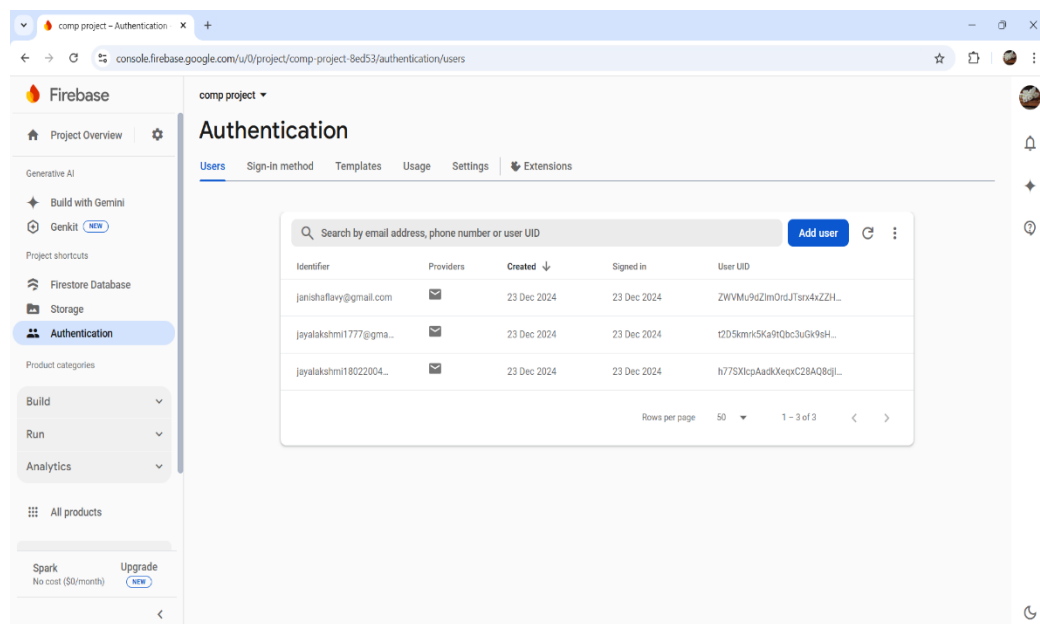
A mobile application login screen with a light pink background. At the top, a status bar shows the time 2:19 PM, data speed 5.7KB/s, and battery level 58%. The word "Login" is displayed at the top left. In the center, the word "Login" is written in a large, bold, black font. Below this, there are two input fields: "Email" and "Password". The "Password" field has a toggle icon on the right. A blue "Login" button is positioned below the input fields. Underneath the button is a red button with the Google logo and the text "Sign in with Google". At the bottom, there is a link that says "Don't have an account? Sign Up". The screen is framed by a white border, and the bottom navigation bar of the mobile device is visible at the very bottom.

**Figure 3.4.2 Students/admin login page**

The student/admin login page allows students to securely log in using their registered email and password, granting access to their personalized dashboard. It includes validation for correct email format and secure password requirements, ensuring safe authentication.

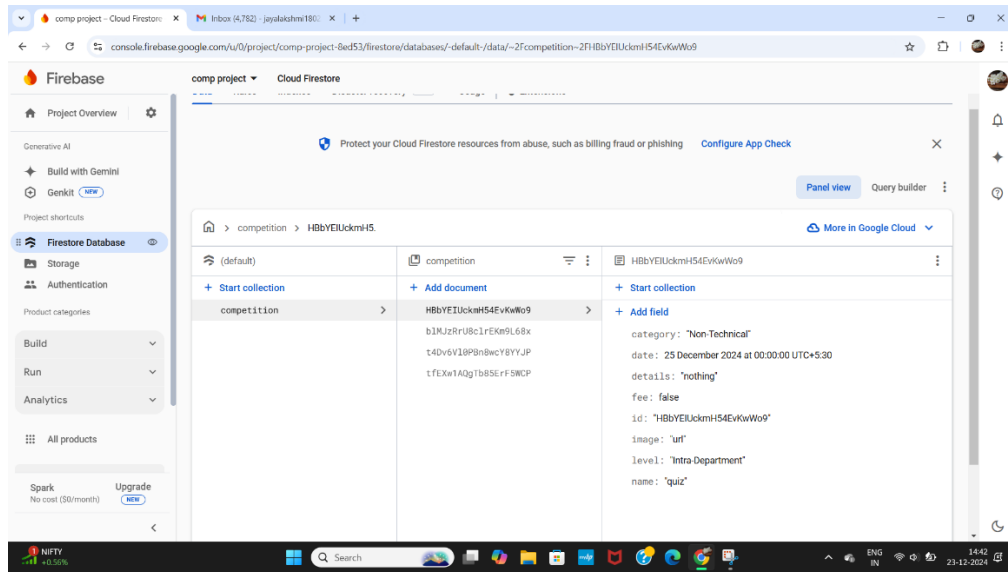
### 3.5 OUTPUT DESIGN

In the Competitions Tracker app, the output data is dynamically generated and displayed using Flutter's powerful UI framework. Information such as competition details, dashboards, leaderboards, and notifications is retrieved from the Firebase backend and rendered in Flutter widgets. The app ensures that this data is stored securely in Firebase's real-time database or Firestore, and updates are reflected immediately on the user interface. Flutter's rich set of widgets enables the creation of responsive and visually appealing outputs, ensuring a seamless experience across various Android devices.



**Figure 3.5.1 Data storing in Firebase**

In the Competitions Tracker app, authentication data is securely managed using Firebase Authentication. User credentials, such as email and password, are securely transmitted and hashed before storage, ensuring data protection.



**Figure 3.5.2 Data storing in Firebase**

Competition details are stored in Firebase Firestore under a collection called competitions. Each document contains key information such as the competition title, date and time, venue, participation type (individual or team-based), organizers' details, and a description.

## **CHAPTER 4**

### **IMPLEMENTATION**

#### **4.1 CODE DESCRIPTION**

The Competitions Tracker application's code is structured to ensure modularity and smooth interaction between the frontend and backend. The application is developed using Flutter for the frontend, with Dart as the programming language. The entry point of the application is in the main dart file, which initializes the application and manages navigation between different screens such as login, dashboard, and competition details pages. The user interface is built using Flutter's pre-defined widgets like List View and Text Form Field, along with custom widgets to display competition details, participant history, and rankings. State management is handled through tools like Provider to store and update data across the application. The Firebase Authentication module is used to manage secure user login and registration, while Firestore serves as the real-time database to store and retrieve data such as competition details, user participation, and results. The application also integrates Firebase Cloud Messaging to send push notifications for important updates. The application's business logic ensures that users can register for competitions, view their achievements, and track rankings in real-time, with all data synchronized through Firestore. This modular approach ensures the application is easy to maintain and scalable, providing a seamless experience for users and robust backend integration.

#### **4.2 STANDARDIZATION OF THE CODING**

The Competitions Tracker application follows strict coding standardization to ensure maintainability, scalability, and collaboration among developers. Consistent naming conventions are applied throughout the code, using camelCase for variables and functions, and Pascal Case for widget names to maintain clarity. The code is well documented with inline comments and docstrings, making it easier for new developers to understand and follow. The application is organized into modular components, with

each feature, such as authentication, competition management, and notifications, separated into different files. Proper error handling is implemented to ensure smooth interaction with Firebase services, addressing potential issues like network failures. Consistent code formatting is maintained with two-space indentation, and Visual Studio Code's formatting tools are used for consistency. Version control is managed using Git, with clear commit messages and a branching strategy to allow for collaborative development. Additionally, the application undergoes regular code reviews and includes unit and widget tests to verify functionality and maintain code quality. These practices ensure that the application's development process is efficient, reliable, and adaptable for future enhancements.

## **4.3 ERROR HANDLING**

### **4.3.1 Firebase interaction**

When interacting with Firebase services (e.g., Firebase Authentication, Firestore), error handling is implemented to manage situations like network failures, authentication errors, or invalid data formats. For example, if the application fails to fetch data from Firestore, an error message is shown to the user, and the application attempts to re-fetch the data. Similarly, during user login, if incorrect credentials are provided, a specific error message is displayed to guide the user to correct the issue.

### **4.3.2 User Input Validation**

Before processing user inputs (such as registration details or competition registrations), the application validates the inputs to ensure they meet the required formats. For instance, if a user enters an invalid email or leaves a required field empty, the application prompts the user to correct the input with an applicationropriate error message. This prevents the application from crashing or behaving unexpectedly due to invalid data.

### **4.3.3 Try-Catch Blocks**

The application uses try-catch blocks in Dart to handle runtime errors. For example, when performing asynchronous operations, such as fetching data from Firestore or sending a notification via Firebase Cloud Messaging, the code is

wrap application in a try block to attempt the operation. If an error occurs (e.g., network issues or service unavailability), the catch block handles the error, logging it or displaying a user-friendly message.

#### **4.3.4 Network Error Handling**

Since the application relies on real-time data from Firebase, network errors can occur. The application handles these scenarios by checking for an active internet connection before attempting to interact with Firebase services. If the network is unavailable, an error message is displayed to inform the user and suggest retrying later.

#### **4.3.5 Error Logging**

For debugging and tracking purposes, the application includes error logging to capture and log errors encountered during execution. This helps developers identify and resolve issues quickly. Firebase's Crashlytics can be integrated to track application crashes in real-time and provide detailed logs for debugging.

## CHAPTER 5

### TESTING AND RESULTS

#### 5.1 TESTING

Testing is a crucial phase in the software development process, as it ensures the reliability, functionality, and stability of the application. In the Competitions Tracker application, various types of testing are performed to identify and resolve bugs, verify the correctness of features, and ensure that the application performs as expected under different conditions.

##### 5.1.1 Unit Testing

Unit testing focuses on testing individual units or components of the application to ensure that each part functions correctly. In the case of the Competitions Tracker application, unit tests are written for functions that perform specific tasks, such as calculating scores, managing user authentication, or processing competition data. The goal is to test these functions in isolation, ensuring they return the correct results based on various inputs.

- **Example:** Testing a function that calculates the points for a competition based on the user's participation.

##### SAMPLE CODE:

```
test('should return correct points for winner', () {  
  
  var result = calculatePoints(true);  
  
  expect(result, equals(10)); // assuming 10 points for winning  
  
});
```



### 5.1.2 Widget Testing

Widget testing is used to test the UI components of the application. It focuses on testing individual widgets, ensuring they render correctly and behave as expected when user interactions occur. For instance, a button widget can be tested to check if it changes its state or triggers the correct function when clicked.

- **Example:** Testing if a button triggers the correct action when pressed.

#### **SAMPLE CODE:**

```
testWidgets('should trigger competition registration on button press',
           (WidgetTester tester)async {

  await tester.pumpWidget(MyApplication());

  await tester.tap(find.byType(RaisedButton));

  await tester.pump();

  // Check if the correct method was called or if the UI updated

  expect(find.text("Registration Successful"), findsOneWidget);

});
```

### 5.1.3 Integration Testing

Integration testing ensures that different parts of the application work together as expected. This type of testing is used to verify that the communication between the frontend (Flutter) and the backend (Firebase) functions properly. For example, testing the user login flow involves verifying that data flows correctly between the application's UI, the authentication API, and Firebase.

#### **5.1.4 Performance Testing**

Performance testing ensures that the application performs well under various conditions, such as high user load, slow network connections, or limited device resources. This helps identify slow areas in the application, or potential crashes that could occur under heavy use. Performance tests are done to ensure the application is optimized for speed and memory usage.

#### **5.1.5 User Acceptance Testing (UAT)**

UAT is performed by the target users (students and staff) to verify that the application meets their requirements and functions as expected in real-world scenarios. During this phase, real users test the application to confirm that all features, such as competition registration, score tracking, and notifications, work correctly and that the application provides a good overall experience.

#### **5.1.6 Validation Testing**

Validation Testing is the process of evaluating the software to ensure it meets the specified requirements and satisfies the needs of the end users. The main goal of validation testing is to ensure that the Competitions Tracker application functions as expected in real-world scenarios and provides the intended outcomes for both students and staff. This type of testing ensures that the application fulfill the business requirements and user expectations.

## **CHAPTER 6**

### **CONCLUSION AND FUTURE ENHANCEMENT**

#### **6.1 CONCLUSION**

The Competitions Tracker application is a user-friendly platform designed to simplify the organization and tracking of inter-college and intra-college competitions for MCA department. By leveraging Flutter for the frontend and Firebase for the backend, the application offers features like real-time notifications, performance tracking, and leaderboards, catering to the needs of both students and staff. It ensures a seamless experience by providing secure access, intuitive navigation, and efficient data management.

This application enhances engagement by encouraging participation and recognizing achievements, fostering a competitive spirit among users. With its robust functionalities and scalable design, the Competitions Tracker sets a foundation for efficiently managing academic and extracurricular activities, empowering students and staff to stay informed, motivated, and connected.

#### **6.2 FUTURE ENHANCEMENT**

The Competitions Tracker application has been designed with scalability and adaptability in mind, paving the way for future enhancements to enrich its functionality. In the next phases, the application can be extended to support additional features such as iOS compatibility, ensuring access across a broader range of devices and users. To enhance user interaction, gamification elements like badges and rewards for milestones can be introduced, making participation more engaging and motivating. Multilingual support and offline functionality can also be added to make the application more inclusive and accessible in diverse environments. These enhancements will ensure the Competitions Tracker application continues to evolve, meeting the dynamic needs of its users effectively.

## APPENDICES

### A.SAMPLE CODING

#### **pubspec.yaml**

name: TrophyTrail\_admin

description: "A new Flutter project."

publish\_to: 'none'

version: 1.0.0+1

environment:

sdk: ^3.6.0

dependencies:

firebase\_core: ^3.8.1

cloud\_firestore: ^5.5.1

get: ^4.6.6

intl: ^0.20.1

dropdown\_button2: ^2.3.9

json\_annotation: ^4.9.0

flutter: sdk: flutter

cupertino\_icons: ^1.0.8

dev\_dependencies:

json\_serializable: ^6.9.0

build\_runner: ^2.4.14

flutter\_test: sdk: flutter flutter\_lints: ^5.0.0

flutter: uses-material-design: true

**widget\_test.dart**

```
// This is a basic Flutter widget test.

//// To perform an interaction with a widget in your test, use the WidgetTester
// utility in the flutter_test package. For example, you can send tap and scroll
// gestures. You can also use WidgetTester to find child widgets in the widget
// tree, read text, and verify that the values of widget properties are correct.

import 'package:TrophyTrail_admin/main.dart';

import 'package:flutter/material.dart';

import 'package:flutter_test/flutter_test.dart';

import 'package:TrophyTrail_admin/_admin/main.dart';

void main() {

  testWidgets('Counter increments smoke test', (WidgetTester tester) async {

    // Build our app and trigger a frame.

    await tester.pumpWidget(const MyApp());

    // Verify that our counter starts at 0.

    expect(find.text('0'), findsOneWidget);

    expect(find.text('1'), findsNothing);

    // Tap the '+' icon and trigger a frame.

    await tester.tap(find.byIcon(Icons.add));

    await tester.pump();

    // Verify that our counter has incremented.

    expect(find.text('0'), findsNothing);

    expect(find.text('1'), findsOneWidget); });}
```

**main.dart**

```

import 'package:TrophyTrail_admin/pages/welcome_page.dart';

import 'package:flutter/material.dart';

import 'package:get/get.dart';

import 'package:firebase_core/firebase_core.dart';

import 'controller/home_controller.dart'; // Ensure this is the correct path to your
HomeController

import 'firebase_options.dart';          // Ensure this is the correct path to your
Firebase configuration

import 'pages/home_page.dart';          // Update with the correct path to your
HomePage

Future<void> main() async {

  WidgetsFlutterBinding.ensureInitialized();

  // Initialize Firebase

  await Firebase.initializeApp(options: firebaseOptions);

  // Registering the HomeController using GetX

  Get.put(HomeController());

  // Run the app

  runApp(const MyApp());}

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  @override

  Widget build(BuildContext context) {

    return GetMaterialApp(

```

```

    title: 'TrophyTrail Admin',

    debugShowCheckedModeBanner: false,

    theme: ThemeData(

      colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),

      useMaterial3: true,

    ),

    // Initial route or main page

    home: WelcomePage(),

  );
}
}

```

#### **backend: firebase\_option.dart**

```

import 'package:firebase_core/firebase_core.dart';

FirebaseOptions firebaseOptions = FirebaseOptions(

  apiKey: 'AIzaSyDLoSUDOWxSaXfe3X6K1x1BMJf9UOFhIKU',

  appId: '1:547806967343:android:ded232c5110715e14bf8da',

  messagingSenderId: '547806967343',

  projectId: 'comp-project-8ed53');

import 'package:flutter/material.dart';

import 'sign_in_page.dart';

import 'sign_up_page.dart';

class WelcomePage extends StatelessWidget {

```

```

@override

Widget build(BuildContext context) {

  return Scaffold(

    body: Container(

      decoration: BoxDecoration(

        gradient: LinearGradient(

          colors: [Colors.blueAccent, Colors.deepPurpleAccent],

          begin: Alignment.topLeft,

          end: Alignment.bottomRight,

        ),

      ),

      child: Center(

        child: SingleChildScrollView(

          child: Column(

            mainAxisAlignment: MainAxisAlignment.center,

            children: [

              // Use a similar icon like emoji_events for a trophy-like icon

              Container(

                decoration: BoxDecoration(

                  boxShadow: [

                    BoxShadow(

                      color: Colors.black.withOpacity(0.3),

                      blurRadius: 10,

```



```

        offset: Offset(0, 4), // Shadow position
      ),
    ],
  ),
  child: Icon(Icons.emoji_events, size: 100, color: Colors.white),
),
SizedBox(height: 20),

// Title text with elegant font style
Text(
  'TROPHY TRAIL',
  style: TextStyle(
    color: Colors.white,
    fontSize: 36,
    fontWeight: FontWeight.bold,
    fontFamily: 'Roboto',
    letterSpacing: 2.0,
  ),
),
SizedBox(height: 10),

// Welcome text with improved font styling
Text(

```

```

'Welcome to the Adventure of Success!',

style: TextStyle(

  color: Colors.white,

  fontSize: 20,

  fontWeight: FontWeight.w500,

  fontStyle: FontStyle.italic,

  fontFamily: 'Georgia',

),

),

 SizedBox(height: 40),

// Sign In button with animated press effect

AnimatedContainer(

  duration: Duration(milliseconds: 200),

  child: ElevatedButton(

    onPressed: () {

      Navigator.push(

        context, MaterialPageRoute(builder: (_) => SignInPage()));

    },

    style: ElevatedButton.styleFrom(

      backgroundColor: Colors.white,

      foregroundColor: Colors.blueAccent,

      shape: RoundedRectangleBorder(

```



```

        borderRadius: BorderRadius.circular(30),
        padding: EdgeInsets.symmetric(horizontal: 40, vertical: 15),
      ),
      child: Text(
        'SIGN UP',
        style: TextStyle(color: Colors.white, fontSize: 18),
      ),
    ), ],
  ), ), ), ); }}

```

### **sign\_up\_page.dart**

```

import 'package:flutter/material.dart';

class SignUpPage extends StatelessWidget {

  const SignUpPage({super.key});

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      body: Container(

        decoration: BoxDecoration(

          gradient: LinearGradient(

            colors: [Colors.red, Colors.deepPurple],

            begin: Alignment.topLeft,

            end: Alignment.bottomRight,

          ),

        ),

        child: Padding(

```

```
padding: EdgeInsets.all(20),

child: Column(

  mainAxisAlignment: MainAxisAlignment.center,

  crossAxisAlignment: CrossAxisAlignment.stretch,

  children: [

    Text(

      'Create Your Account',

      style: TextStyle(

        color: Colors.white,

        fontSize: 32,

        fontWeight: FontWeight.bold,

      ),

    ),

    SizedBox(height: 20),

    TextField(

      decoration: InputDecoration(

        hintText: 'Full Name',

        filled: true,

        fillColor: Colors.white,

        border: OutlineInputBorder(

          borderRadius: BorderRadius.circular(10),

        ),

      ),

    ),

  ],

),
```

```
    SizedBox(height: 20),  
    TextField(  
      decoration: InputDecoration(  
        hintText: 'Email',  
        filled: true,  
        fillColor: Colors.white,  
        border: OutlineInputBorder(  
          borderRadius: BorderRadius.circular(10),  
        ),  
      ),  
    ),  
    SizedBox(height: 20),  
    TextField(  
      decoration: InputDecoration(  
        hintText: 'Password',  
        filled: true,  
        fillColor: Colors.white,  
        border: OutlineInputBorder(  
          borderRadius: BorderRadius.circular(10),  
        ),  
      ),  
      obscureText: true,  
    ),
```

```
    SizedBox(height: 20),  
    TextField(  
      decoration: InputDecoration(  
        hintText: 'Confirm Password',  
        filled: true,  
        fillColor: Colors.white,  
        border: OutlineInputBorder(  
          borderRadius: BorderRadius.circular(10),  
        ),  
      ),  
      obscureText: true,  
    ),  
    SizedBox(height: 20),  
    ElevatedButton(  
      onPressed: () {},  
      style: ElevatedButton.styleFrom(  
        backgroundColor: Colors.white,  
        foregroundColor: Colors.red,  
        shape: RoundedRectangleBorder(  
          borderRadius: BorderRadius.circular(30),  
        ),  
        padding: EdgeInsets.symmetric(horizontal: 30, vertical: 15),  
      ),
```

```

        child: Text('SIGN UP'),
      ),
    ],
  ),
);
}}

```

### **sign\_in\_page.dart**

```

import 'package:flutter/material.dart';

class SignInPage extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      body: Container(

        decoration: BoxDecoration(

          gradient: LinearGradient(

            colors: [Colors.red, Colors.deepPurple],

            begin: Alignment.topLeft,

            end: Alignment.bottomRight,

          ),

        ),

        child: Padding(

          padding: EdgeInsets.all(20),

          child: Column(

            mainAxisAlignment: MainAxisAlignment.center,

            crossAxisAlignment: CrossAxisAlignment.stretch,

            children: [

              Text(

```



```
'Hello\nSign in!',  
  
style: TextStyle(  
  
  color: Colors.white,  
  
  fontSize: 32,  
  
  fontWeight: FontWeight.bold,  
  
),  
  
),  
  
SizedBox(height: 20),  
  
TextField(  
  
  decoration: InputDecoration(  
  
    hintText: 'Email',  
  
    filled: true,  
  
    fillColor: Colors.white,  
  
    border: OutlineInputBorder(  
  
      borderRadius: BorderRadius.circular(10),  
  
    ),  
  
  ),  
  
),  
  
),  
  
SizedBox(height: 20),  
  
TextField(  
  
  decoration: InputDecoration(  
  
    hintText: 'Password',  
  
    filled: true,
```

```

        fillColor: Colors.white,

        border: OutlineInputBorder(

            borderRadius: BorderRadius.circular(10),

        ),

    ),

    obscureText: true,

),

 SizedBox(height: 20),

 ElevatedButton(

    onPressed: () {},

    style: ElevatedButton.styleFrom(

        backgroundColor: Colors.white,

        foregroundColor: Colors.red,

        shape: RoundedRectangleBorder(

            borderRadius: BorderRadius.circular(30),

        ),

        padding: EdgeInsets.symmetric(horizontal: 30, vertical: 15),

    ),

    child: Text('SIGN IN'),

),

 TextButton(

    onPressed: () {},

    child: Text(

```

```

        'Forgot Password?',
        style: TextStyle(color: Colors.white),
    ),
  ],
),
);
}}

```

### **home\_page.dart**

```

import 'package:TrophyTrail_admin/_admin/controller/home_controller.dart';
import 'package:TrophyTrail_admin/_admin/pages/add_comp_page.dart';
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:intl/intl.dart';
import '../controller/home_controller.dart';
import 'add_comp_page.dart'; // For date formatting

class HomePage extends StatelessWidget {
  const HomePage({super.key});

  @override
  Widget build(BuildContext context) {
    return GetBuilder<HomeController>(builder: (ctrl) {
      return Scaffold(
        appBar: AppBar(
          title: const Text(
            'Competitions Admin',
            style: TextStyle(fontWeight: FontWeight.bold, color: Colors.white),
          ),
          centerTitle: true,

```

```

flexibleSpace: Container(
  decoration: const BoxDecoration(
    gradient: LinearGradient(
      colors: [Colors.deepPurple, Colors.purpleAccent],
      begin: Alignment.topLeft,
      end: Alignment.bottomRight,
    ),
  ),
),
elevation: 10.0,
),
body: ctrl.competition.isEmpty
  ? Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: const [
        Icon(Icons.event_busy, size: 80, color: Colors.grey),
        SizedBox(height: 16),
        Text(
          'No competitions available!',
          style: TextStyle(
            fontSize: 18,
            fontWeight: FontWeight.bold,

```

```

        color: Colors.grey,

      ),

    ),
    SizedBox(height: 8),
    Text(
      'Add a competition to get started.',
      style: TextStyle(fontSize: 14, color: Colors.grey),
    ),
  ],
),
)

: ListView.builder(
  itemCount: ctrl.competition.length,
  padding: const EdgeInsets.symmetric(vertical: 10, horizontal: 15),
  itemBuilder: (context, index) {
    // Reverse the order of competitions
    var competition = ctrl.competition[ctrl.competition.length - 1 - index];

    String title = competition.name ?? 'No Title Available';

    String date = competition.date != null
      ? DateFormat('yyyy-MM-dd').format(competition.date!)
      : 'No Date Available';

    return Card(

```

```
elevation: 4,  
  
margin: const EdgeInsets.symmetric(vertical: 8),  
  
shape: RoundedRectangleBorder(  
  borderRadius: BorderRadius.circular(12),  
),  
  
child: ListTile(  
  contentPadding: const EdgeInsets.all(16),  
  
  leading: CircleAvatar(  
    backgroundColor: Colors.deepPurple,  
    child: const Icon(Icons.event, color: Colors.white),  
  ),  
  
  title: Text(  
    title,  
  
    style: const TextStyle(  
      fontSize: 18,  
      fontWeight: FontWeight.bold,  
      color: Colors.black87,  
    ),  
  ),  
  
  subtitle: Text(  
    date,  
  
    style: const TextStyle(fontSize: 14, color: Colors.black54),  
  ),
```

```

trailing: IconButton(

  icon: const Icon(Icons.delete, color: Colors.red),

  onPressed: () {

    showDialog(

      context: context,

      builder: (BuildContext context) {

        return AlertDialog(

          title: const Text(

            'Confirm Deletion',

            style: TextStyle(color: Colors.red),

          ),

          content: const Text(

            'Are you sure you want to delete this competition?',

          ),

          actions: [

            TextButton(

              onPressed: () {

                Navigator.of(context).pop();

              },

              child: const Text('Cancel'),

            ),

            TextButton(

              onPressed: () {

```

```

        if (competition.id != null) {

            ctrl.deleteCompetition(competition.id!);

            Navigator.of(context).pop();

        }

    },

    child: const Text('Delete'),

), ], ); }, );

},

),

),

);

},

),

floatingActionButton: FloatingActionButton(

    onPressed: () {

        Get.to(() => const AddCompPage());

    },

    backgroundColor: Colors.deepPurple,

    child: const Icon(Icons.add, color: Colors.white),

), ); }); }

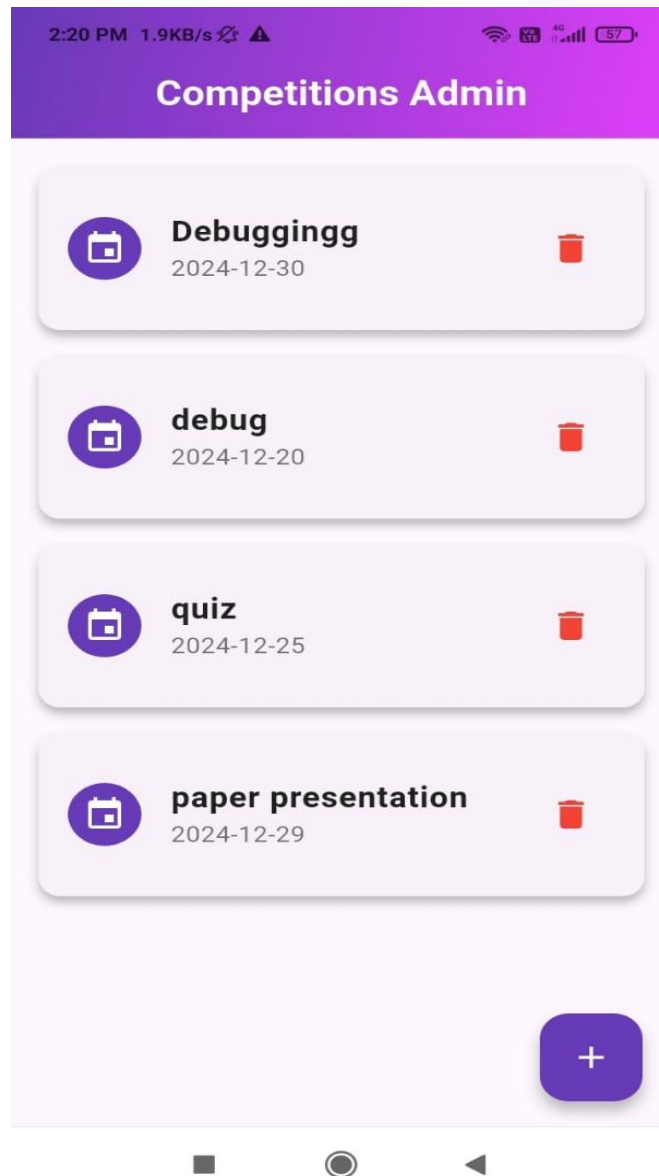
}

```



## B SCREENSHOTS

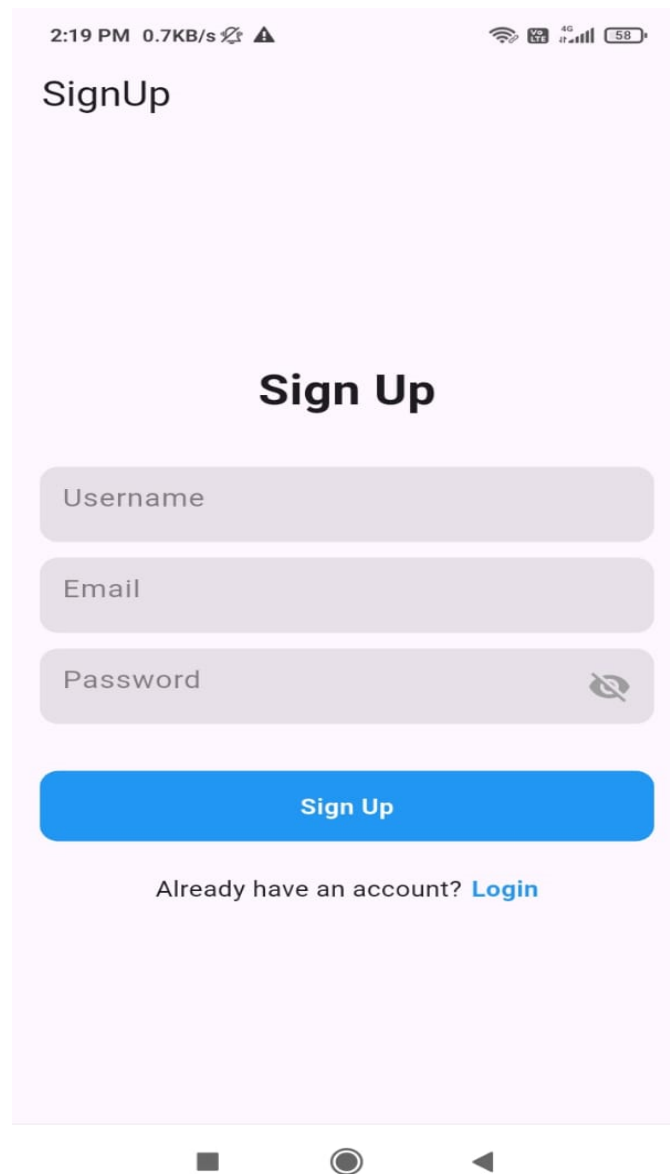
### HOME PAGE



**Figure B.1 Home Page**

It represents the Competitions Admin Page of the app, where staff members can view, manage, and organize competitions. Each competition is displayed as a card, listing the competition name and scheduled date.

## SIGN UP PAGE

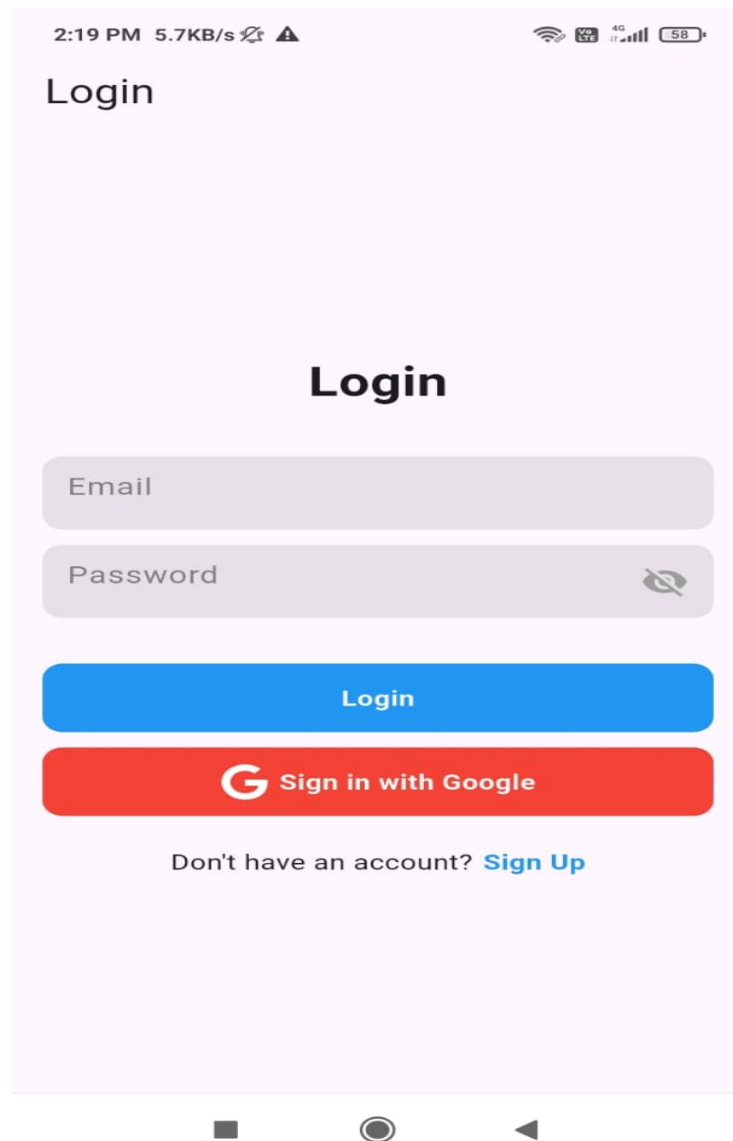


A mobile application interface for a sign-up page. At the top, the status bar shows the time as 2:19 PM, a data speed of 0.7KB/s, and various connectivity icons. The page title 'SignUp' is displayed in a large, bold, black font. Below the title, the heading 'Sign Up' is centered in a bold, black font. The form consists of three input fields: 'Username', 'Email', and 'Password'. The 'Password' field includes a toggle icon for visibility. A prominent blue button labeled 'Sign Up' is positioned below the form fields. At the bottom of the form, there is a link that reads 'Already have an account? Login'. The entire form is set against a light pink background.

**Figure B.2 Students/admin Sign Up page**

The Sign-Up Page will provide secure way for users to register and access the app. Users will select their role as either a student or staff during the registration process. The form will include fields such as name, email, and a password creation section.

## LOGIN PAGE

A mobile application login screen with a light purple background. At the top, a status bar shows the time 2:19 PM, data speed 5.7KB/s, and battery level 58%. The word "Login" is displayed at the top left. In the center, the word "Login" is written in a large, bold, black font. Below this, there are two input fields: "Email" and "Password". The "Password" field has a toggle icon on the right. Below the input fields are two buttons: a blue "Login" button and a red "Sign in with Google" button featuring the Google logo. At the bottom, there is a link that says "Don't have an account? Sign Up". The screen is framed by a white border, and the bottom navigation bar of the phone is visible at the very bottom.

**Figure B.3 Students/admin login page**

The **student/admin login page** allows students to securely log in using their registered email and password, granting access to their personalized dashboard. It includes validation for correct email format and secure password requirements, ensuring safe authentication.

## ADD COMPETITION

2:20 PM 0.1KB/s 57%

← Add Competition

**Add New Competition**

Competition Name

Details (Optional)

Image URL

Date (yyyy-MM-dd)

Place (Optional)

Category ▼ Level ▼

Registration Fee? (Optional)

true ▼

**Figure B.4 Add Competition page**

The add competition includes fields for entering the competition name (mandatory), optional details, an image URL, date, and place. There are dropdowns for selecting the competition category and level, along with an option to specify if a registration fee is required(Optional). This simple and user-friendly design helps users easily add competition details to the app.

## REFERENCES

[1]"Implementing Cloud-Based Mobile Applications for Event Tracking" – Discusses integrating Firebase with mobile apps.

[2]"Web-Based Event Management Systems for Educational Institutions" – Explains tracking and managing events in academic setups.

[3]<https://flutter.dev/docs>

[4]<https://firebase.google.com/docs>