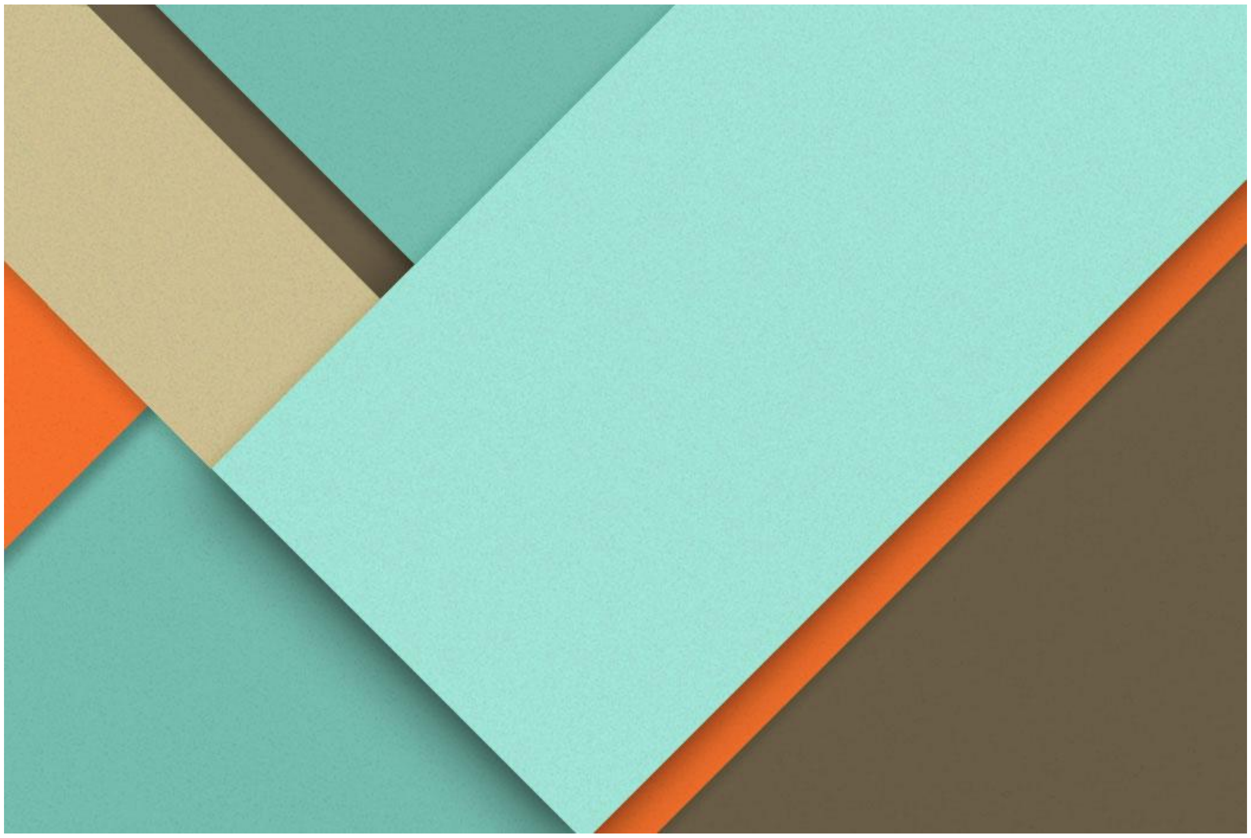


Edu Hub Testing Report

Team 7:

Chintamani Masthanaiah	-	191CS115
Venkata Sravani	-	191CS223
Kruthika K Sudhama	-	191CS224
V Madhan Kumar	-	191CS260
Vinesh S Talpankar	-	191CS265

Table of Contents:

Introduction:

This document explains the various activities performed as part of Testing of EduHub application.EduHub is an open platform for all the events, boot camps, webinars from all the colleges to extend their reach and for students to register for various events. Functionalities of Edu Hub are Login/SignUp , Searching for Events , Event Registration , Event Publishing, Google Calendar Integration.

Testing Scope:

The scope of the testing for our project can be categorized as follows:-

a. In Scope:
   Functional Testing for the following modules are in Scope of Testing
   - Authorization Module
   - Participant Module
   - Admin Module

b. Out of Scope:
   Performance Testing was not done for this application.

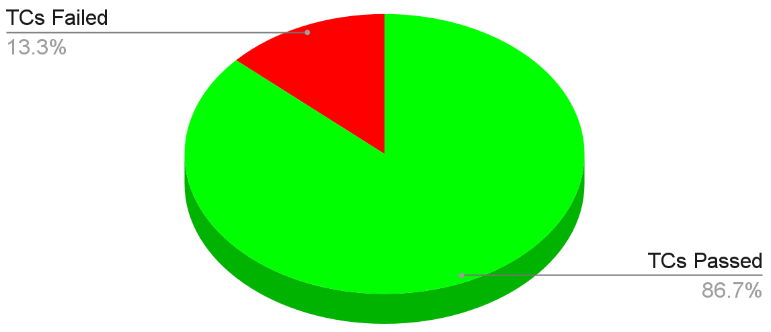
c. Items not tested:
   Many inbuilt widgets provided by Flutter were used in this project. These widgets don't require testing as they are well tested. Firebase is also used in this project to implement Google SignIn/SignUp and doesn't require testing.

Metrics:

A. No. of Test Cases Planned vs Executed
B. No. of Test Cases Passed/Failed

Test Cases Planned	Test Cases Executed	TCs Passed	TCs Failed
20	**15**	**13**	**2**

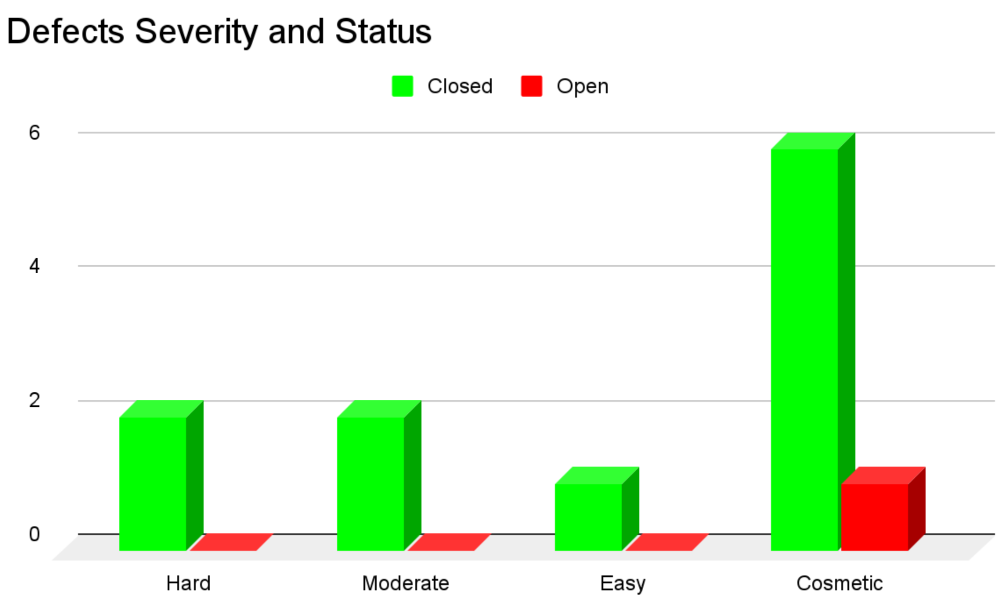
Test Cases Pass vs Fail



C. No of Defects Identified and their Status & Severity

	Hard	Moderate	Easy	Cosmetic	Total
Closed	2	2	1	6	11
Open	0	0	0	1	1

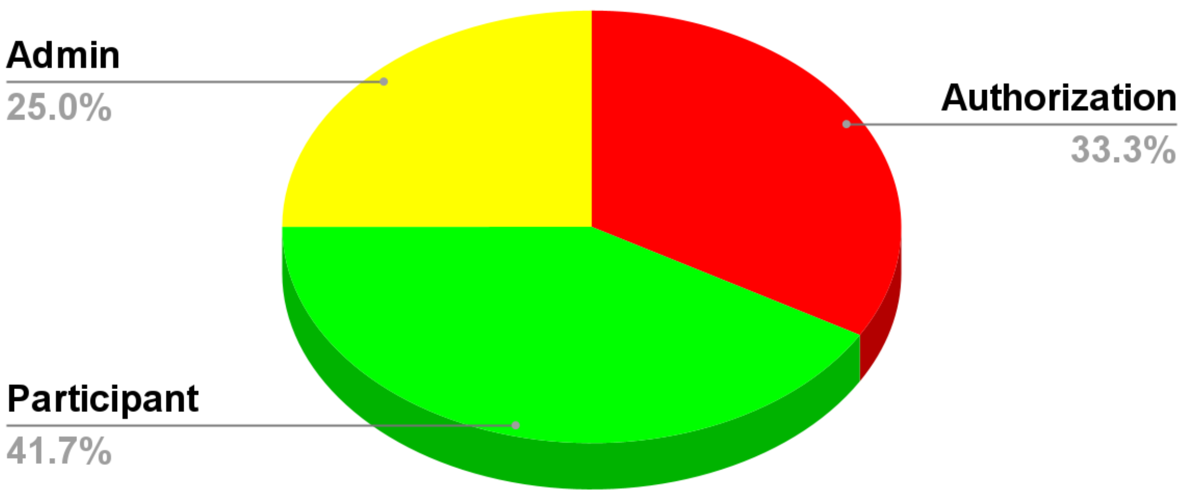
Total = 12



D. Defects Distribution – Module Wise

	Authorization	Participant	Admin	Total
Easy	0	1	0	1
Moderate	2	0	0	2
Hard	1	1	0	2
Cosmetic	1	3	3	7
Total	4	5	3	12

Defects Distribution – Module Wise



Types of Testing Performed :

a) Smoke Testing:
This testing is done whenever a Build is received (deployed into the Test environment) for Testing to make sure the major functionalities are working fine, Build can be accepted and testing can start.

b) System Integration Testing:
This is the Testing performed on the Application under test, to verify the entire application works as per the requirements. Critical Business scenarios were tested to make sure important functionalities in the application works as intended without any errors.

Types of Testing in Flutter:-

Unit Testing
Unit testing is the easiest method to test an application. It is based on ensuring the correctness of a piece of code (a function, in general) o a method of a class. But, it does not reflect the real environment and subsequently, is the least option to find the bugs.

Widget Testing
Widget testing is based on ensuring the correctness of the widget creation, rendering and interaction with other widgets as expected. It goes one step further and provides a near real-time environment to find more bugs.

Integration Testing
Integration testing involves both unit testing and widget testing along with external components of the application like database, web service, etc., It simulates or mocks the real environment to find nearly all bugs, but it is the most complicated process.
Flutter provides support for all types of testing. It provides extensive and exclusive support for Widget testing. In this chapter, we will discuss widget testing in detail.

Test Environment and Tools:

Dart language and Flutter framework provides extensive support for the automated testing of a Flutter application. Hence, the environment and tools used are the same as that of the application itself.

Bugs Patched:

Bug 1 :

Difficulty: Moderate
Where: Under User Profile Update Feature (user_profile_update.dart)
What: New Usernames and Emails were not getting updated in the Postgres Database
Solution:

1. Firstly , we cross verified the queries we were using to update both the username and password

   For updating Username :
   update participant set participant_name=\'$username\' where participant_email=\'$email\' ;

   For updating Email :
   update participant set participant_email =\'$newemail\' where participant_name=\’$username\’ ;

2. As queries were working correctly , we moved on to check the async function (checking whether the data is getting retrieved from the database before or after the widget is build) , here we added await keyword to make sure that all processes waited until the data from our queries were retrieved from our Database.

3. We check which type of user is logged into the app (whether Participant or Admin), we ran a sample query:

   select admin_email from admino;

   We then properly assigned the type of User to the above queries.

4. Both username and emails were successfully getting updated in the Postgres Database.

5. Bug Fixed

Bug 2 :

Difficulty: **Hard**
Where: Under View Events Feature (FestDetails.dart)
What: The switch was not filtering the registered events from all events
Solution:

1. Firstly we select a particular participant and find the events available to him and the ones he has registered by running queries on the Postgres Database.

2. We have already created the Switch widget in the Scaffold.

3. We must make sure that the OnChanged value is assigned to the isSwitched variable inside a setState() function. So that the build is rerun whenever the isSwitched value changes.

4. Now inside the runQuerry() function (which handles data selection from evento table) we need to make sure that when

   - IF isSwitched == false
     Then all events must be added to result, query is:

     *'select * from evento'*

- ELSE

  Then only registered events must be added to the result, for that we must first find participant_id. Then use that to get all registered event details for the participant.
  Queries are:

  'select participant_id from participant where participant_email= \'$mail\' '

  'Select A.event_id, A.event_name, A.start_date_time, A.end_date_time, A.register_start_date_time, A.register_end_date_time, A.place,A.short_description, A.description, B.group_name, A.price from evento as A , group_participant as B where A.event_id = B.event_id and B.participant_id = \'$regID\' '

5. We can run this and see that the Filter is working properly by verifying the results from step 1.

6. Bug Fixed

Bug 3 :

Difficulty: Easy
Where: Under User Profile Display Feature (userprofile.dart)
What: Username and Email were not getting displayed after updation
Solution:

1. We checked whether the Username and Email were updated by printing them in the flutter console . Both the Username and Email were updated and displayed on the console.

2. We looked into the widget , where we displayed both the credentials on the screen. We had to build the widget everytime we visited the User Profile Display Page.

3. So , we used the setState function , which would build the widget everytime we visited the User Profile Display Page , and display both the credentials in real time (even right after the update).

4. Bug Fixed

Bug 4 :

Difficulty: Hard
Where: Under User Profile Update Feature (user_profile_update.dart)
What: Emails were not getting updated in the Firebase Authentication List
Solution:

1. We first read up on the details regarding updating emails in the Firebase auth list , we got to know that firebase had set a few restrictions on updating any of the important credentials like email or password.

2. We had to Re-Authenticate with the previous Email and Password.

   await FirebaseAuth.instance .signInWithEmailAndPassword(email: email, password: password)

3. Now , we require Email and Password from SignIn Page , we have email already stored in a global variable (widget._user.email) , now we similarly transfer password from SignIn page to User Profile Page using a global variable (under class **g** , method **pass**).

4. After Re-authenticating with credentials , we now update the New Email.

   await user.updateEmail(newemail);

5. Now , we checked whether New Email got updated in Firebase using try and catch block to display errors , if try block fails.

6. New Email was updated in Firebase and we could even login with the updated email in the SignIn Page.

7. Bug Fixed

Bug 5 :

Difficulty: Moderate
Where: Under SignIn Feature (SignIn.dart)
What: Google SignIn & SignUp were not working
Solution:

1. We had to generate SHA 1 and SHA 256 certificate fingerprints for each of our systems and update them in the Firebase SDK Configuration.

2. We generated both the keys using the command:
   Gradle signingReport

3. We added both keys in the Firebase SDK Configuration.

4. Google SignIn and SignUp worked perfectly then on.

5. Bug Fixed

Exit Criteria:

a) All test cases should be executed – Yes

b) All defects in Hard, Moderate ,Easy severity should be verified and closed – Yes.

Conclusion:

Hence, we have successfully tested various modules of the Application and patched all the bugs that were encountered. Therefore, the application is suggested to ‘Go Live’ by our team.