Chapter 4: Testing And Analysis →

4.1. Test Plan

We will be examining our sofware with the help of two type of test, which are: unit testing and system testing . The aim of testing is to find bugs and explain their significance . With the help of testing we will decrease the risk by proactively identifying and assisting with the resolution of issues.

4.1.1. Test plan of Unit Testing

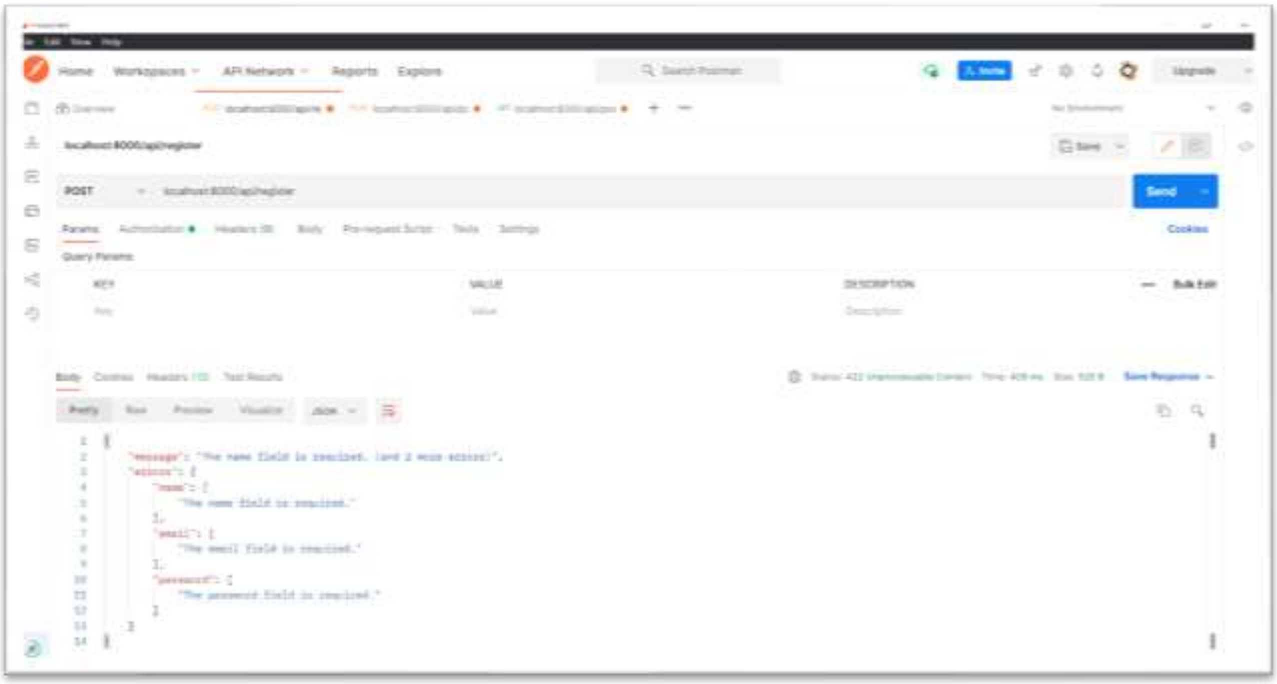
In the Unit testing, we will go through the testing of API's using post man. If we want to check the unit testing of flutter alone, then it is of no use, since the required datas will be backed up by backend. So those features as a whole will be tested only in system testing.

4.1.2. Test plan of System Testing

In System testing, we will use and test all the necessary features. Altogether we will be testing 15 features.

1.2. Unit Testing

4.2.1. Unit test of registering validation API.



localhost:8000/api/register

POST localhost:8000/api/register

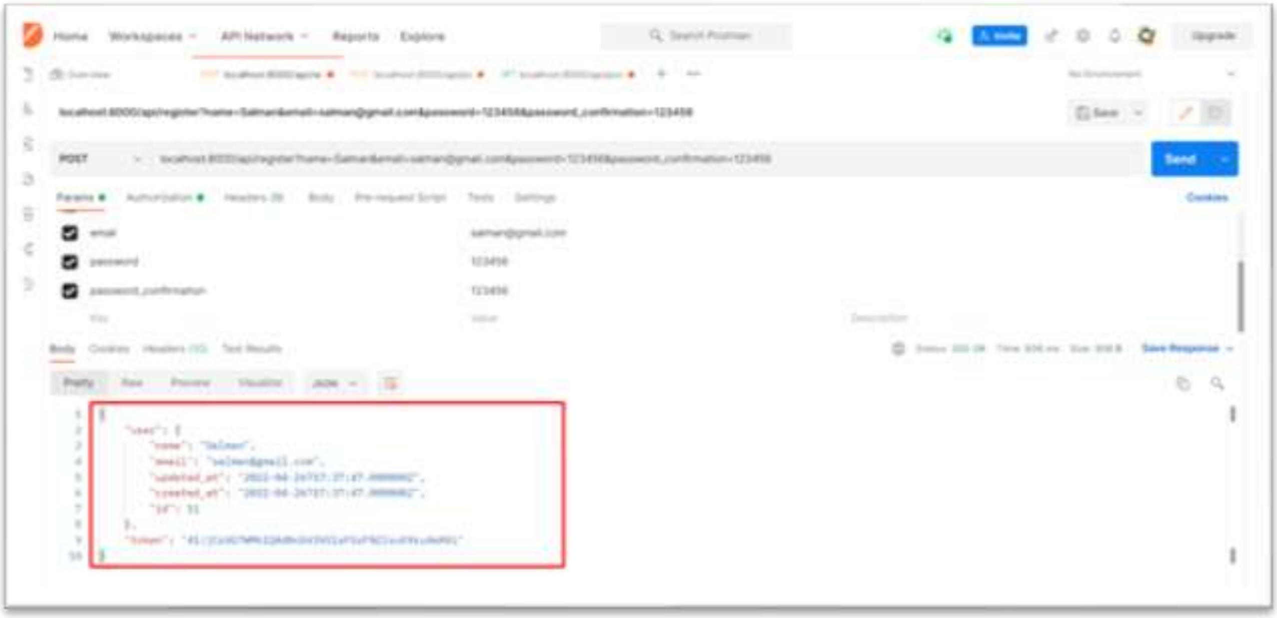
```
{
    "message": "The name field is required. (and 2 more errors)",
    "errors": {
        "name": [
            "The name field is required."
        ],
        "email": [
            "The email field is required."
        ],
        "password": [
            "The password field is required."
        ]
    }
}
```

Figure 1 Registering validation

Objective	To validate user credentials before registering.
Action	In the Laravel file; name, email, and password was made compulsory, which helped when API code was written in postman.
Expected Result	The user must not be registered.
Actual Result	The error message came.
Conclusion	The test was Successful.

Table 1 testing registering a new user to the app

4.2.2. API unit test of registering new user.



localhost:8000/api/register?name=Salman&email=salman@gmail.com&password=123456&password_confirmation=123456

POST localhost:8000/api/register?name=Salman&email=salman@gmail.com&password=123456&password_confirmation=123456

email	salman@gmail.com
password	123456
password_confirmation	123456

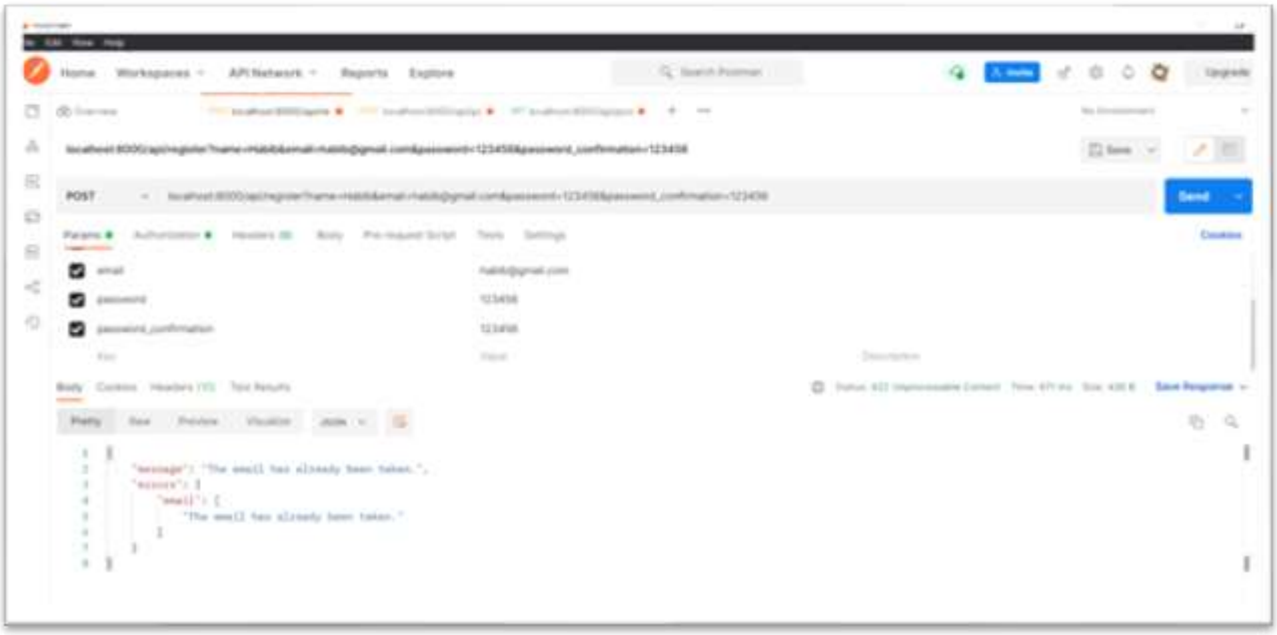
```
{
    "user": {
        "name": "Salman",
        "email": "salman@gmail.com",
        "updated_at": "2022-04-24T17:37:47.000000Z",
        "created_at": "2022-04-24T17:37:47.000000Z",
        "id": 11
    },
    "token": "[illegible]"
}
```

Figure 2 New user, Salman added to the system

Objective	To register new user in the database.
Action	Registering new user API code was used in the postman.
Expected Result	A new user must be made.
Actual Result	A new user named Salman with salman@gmail.com was made.
Conclusion	The test was successful.

Table 2 Unit test of registering new user.

4.2.3. Unit test of registering same user validation.



POST localhost:8000/api/register?name=Habib&email=habib@gmail.com&password=123456&password_confirmation=123456

Key	Value
✓ email	habib@gmail.com
✓ password	123456
✓ password_confirmation	123456

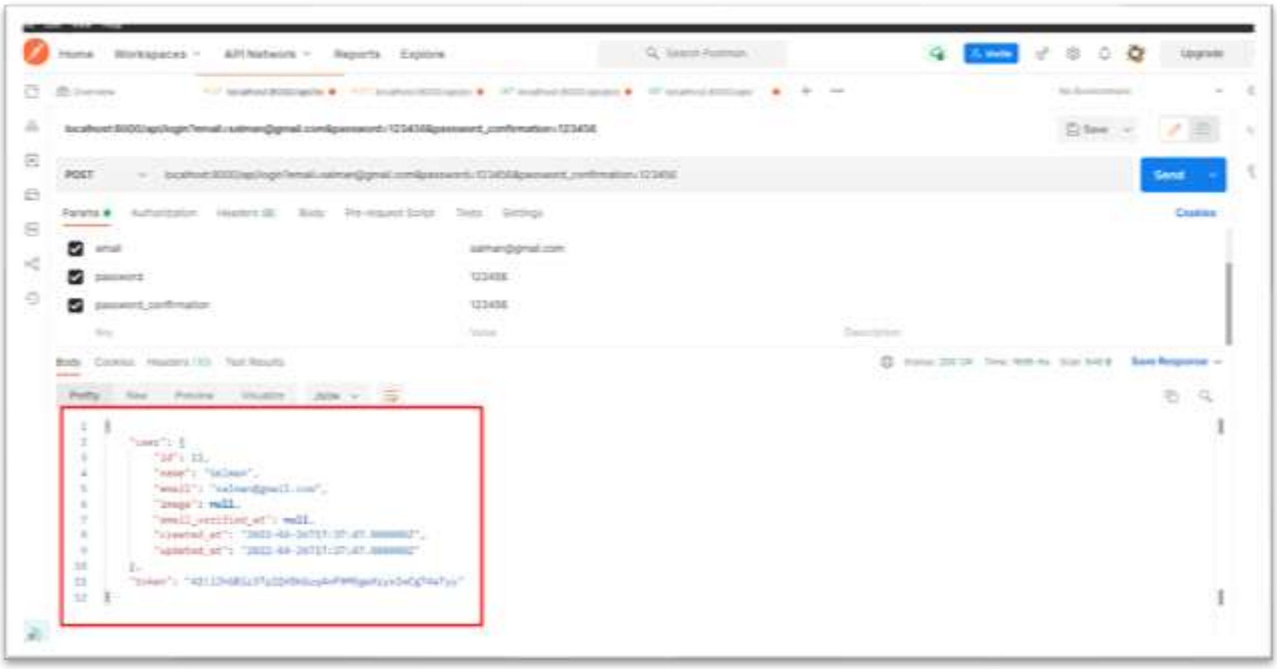
```
{
    "message": "The email has already been taken.",
    "errors": {
        "email": [
            "The email has already been taken."
        ]
    }
}
```

Figure 3 trying to register a registered user.

Objective	To check if already registered user are again registered or not.
Action	To put registered user credentials api in the postman.
Expected Result	Error message must appear.
Actual Result	No registered user was allowed to register again.
Conclusion	The test was successful.

Table 3 Re-Register user api validation api unit test.

4.2.4. Log in api unit tests.



POST localhost:8000/api/login?email=salman@gmail.com&password=123456&password_confirmation=123456

Key	Value
email	salman@gmail.com
password	123456
password_confirmation	123456

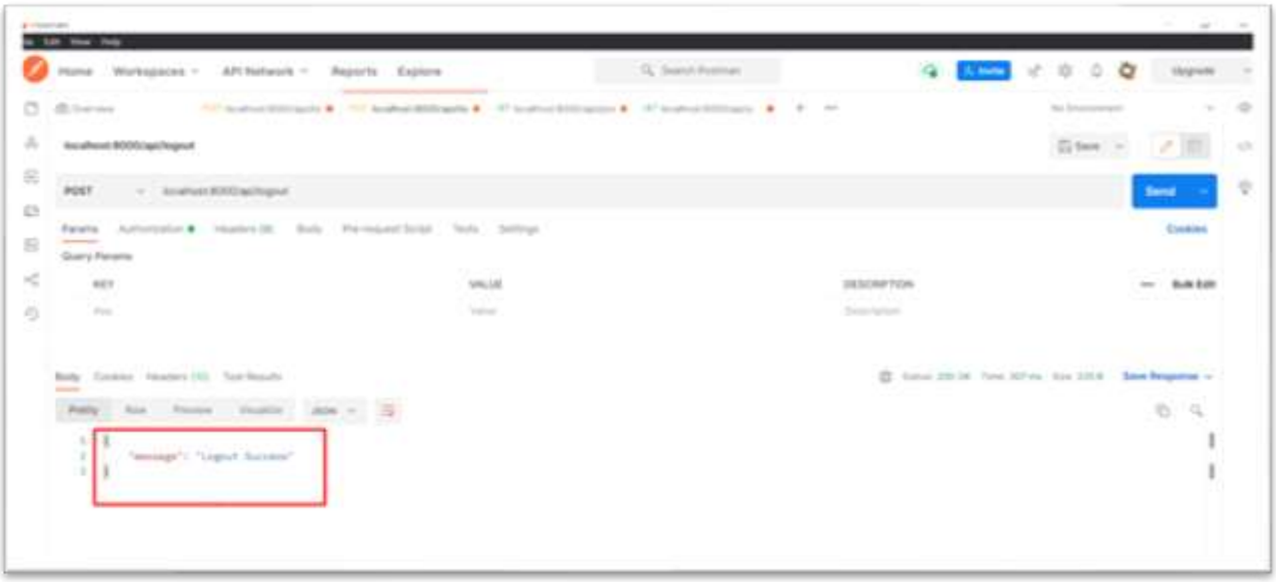
```
{
    "user": {
        "id": 11,
        "name": "Salman",
        "email": "salman@gmail.com",
        "image": null,
        "email_verified_at": null,
        "created_at": "2022-04-26T17:37:47.000000Z",
        "updated_at": "2022-04-26T17:37:47.000000Z"
    },
    "token": "[illegible]"
}
```

Table 4 Login api test

Objective	To log in to the database system.
Action	Log in api was used in the postman.
Expected Result	Registered user must be able to log in to the system.
Actual Result	Log in to the system was successful.
Conclusion	The test was succesful.

Table 5 Log in unit test of api

4.2.5. Logout feature api unit tests.



localhost:8000/api/logout

POST localhost:8000/api/logout

```
{
    "message": "Logout Success"
}
```

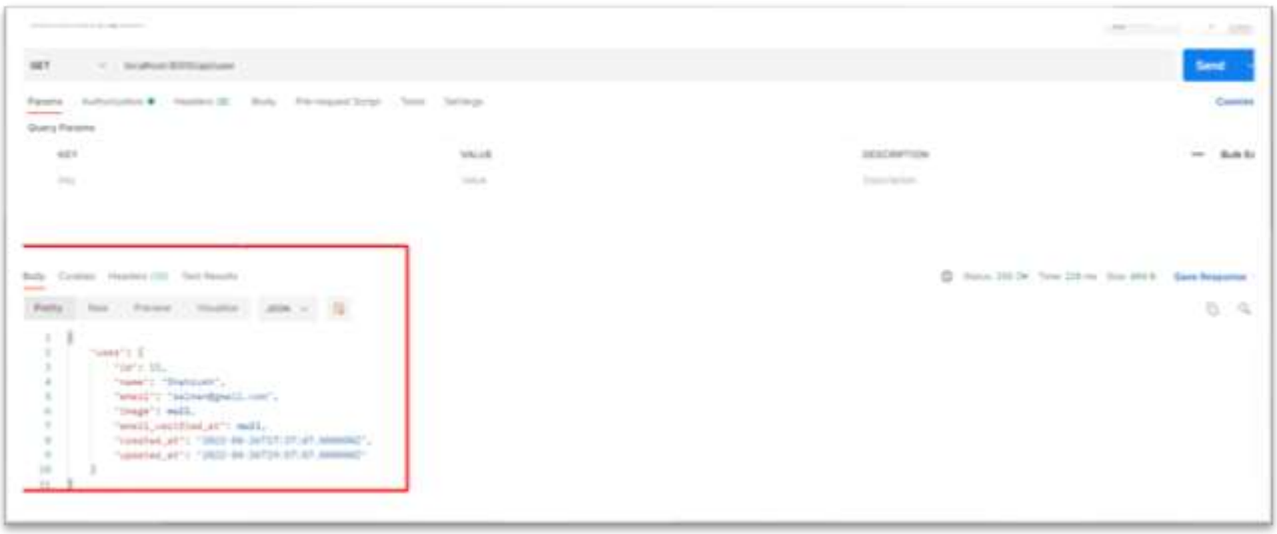
Figure 4 Logout successfully form the system

Objective	To logout user from the system.
Action	User logout api code in the postman.
Expected Result	User must be able to logout from the system.
Actual Result	User was able to be logged out.
Conclusion	The test was successful.

Table 6 User logout api unit test.

4.2.6. Get user data

4.2.7. Update user data



GET localhost:8000/api/user

Params Authorization Headers Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers Test Results

Pretty Raw Preview Visualize JSON

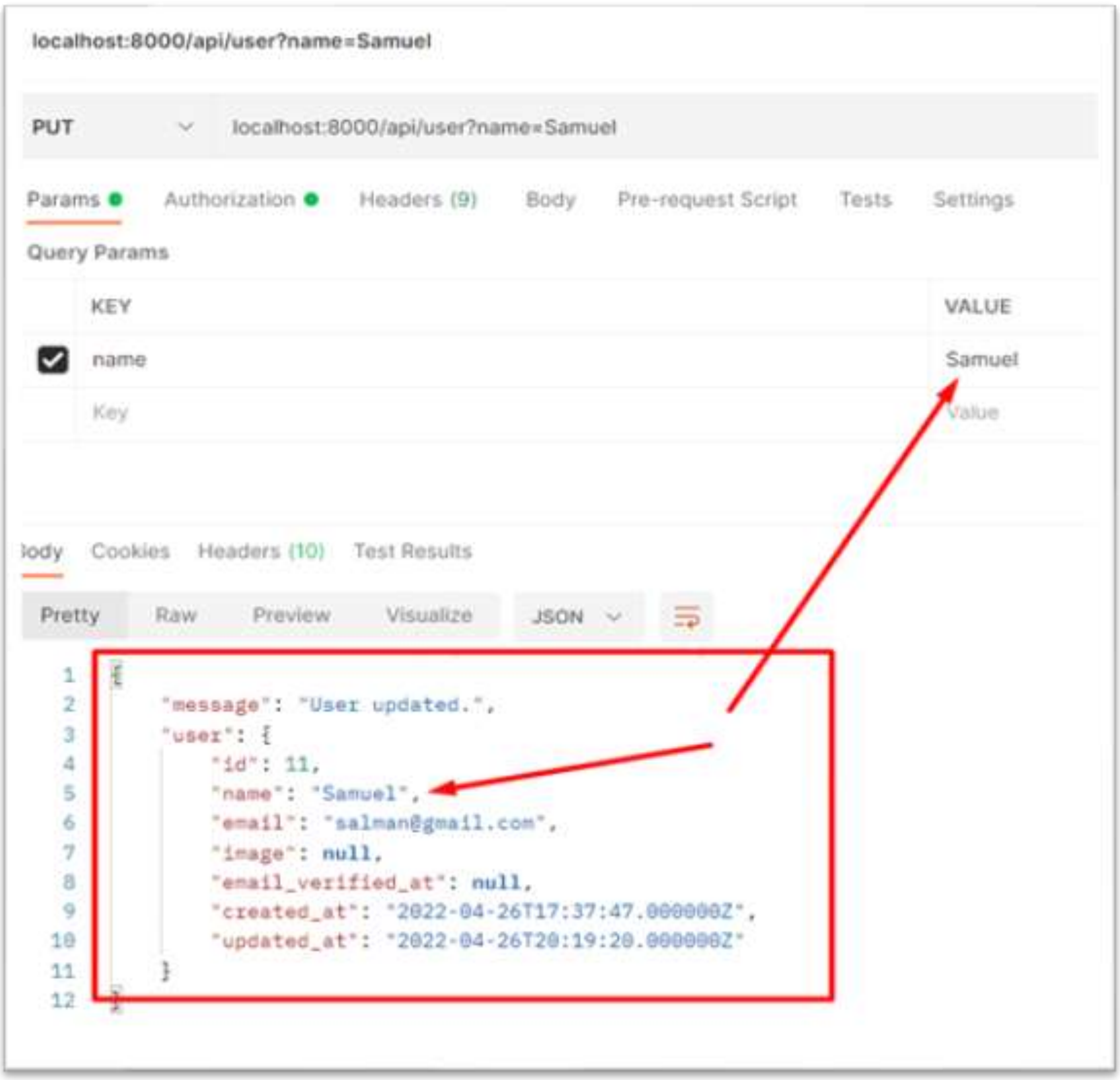
```
{
    "user": {
        "id": 11,
        "name": "[illegible]",
        "email": "[illegible]@gmail.com",
        "image": null,
        "email_verified_at": null,
        "created_at": "2022-04-[illegible]",
        "updated_at": "2022-04-[illegible]"
    }
}
```

Table 7 Get user data using user API

Objective	To know detail about a user.
Action	User API code was used in the postman.
Expected Result	Detail of user must be returned.
Actual Result	Detail of user was printed.
Conclusion	The test was successful.

Table 8 User detail api unit test.

4.2.7. Update user data


localhost:8000/api/user?name=Samuel

PUT localhost:8000/api/user?name=Samuel

KEY	VALUE
name	Samuel
Key	Value

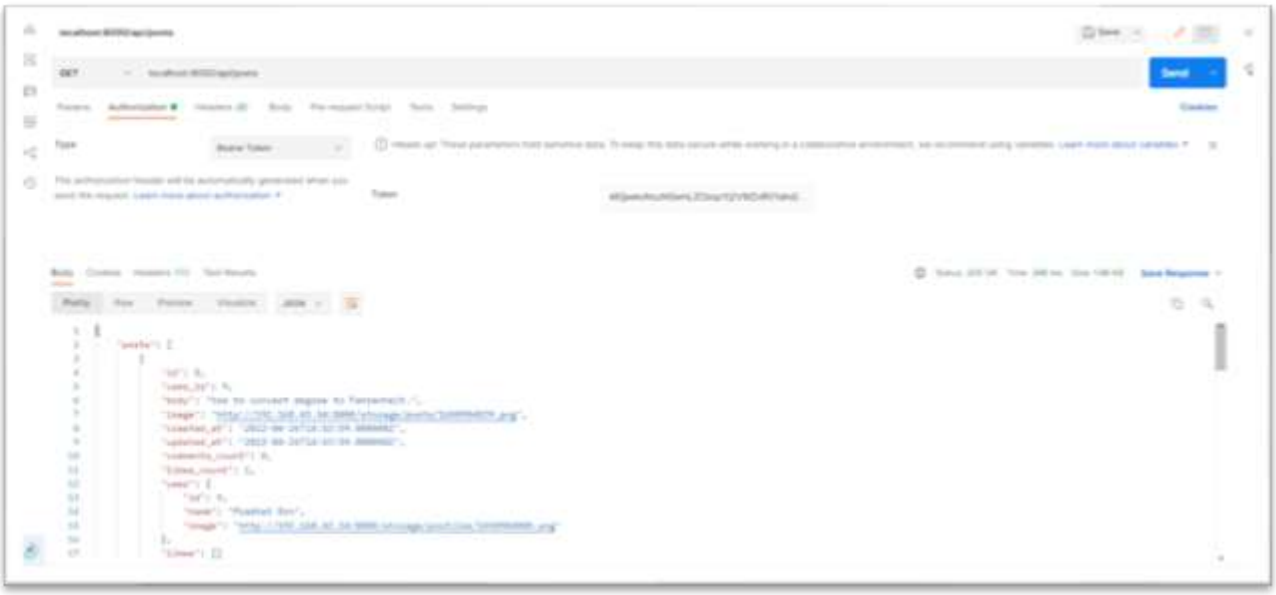
```
{
    "message": "User updated.",
    "user": {
        "id": 11,
        "name": "Samuel",
        "email": "salman@gmail.com",
        "image": null,
        "email_verified_at": null,
        "created_at": "2022-04-26T17:37:47.000000Z",
        "updated_at": "2022-04-26T20:19:20.000000Z"
    }
}
```

Table 9 User name was updated.

Objective	To update user credentials.
Action	User update API was used in the postman.
Expected Result	User name must be changed.
Actual Result	The name was changed.
Conclusion	The test was successful.

Table 10 User profile change.

4.2.8. Getting to view question post detail



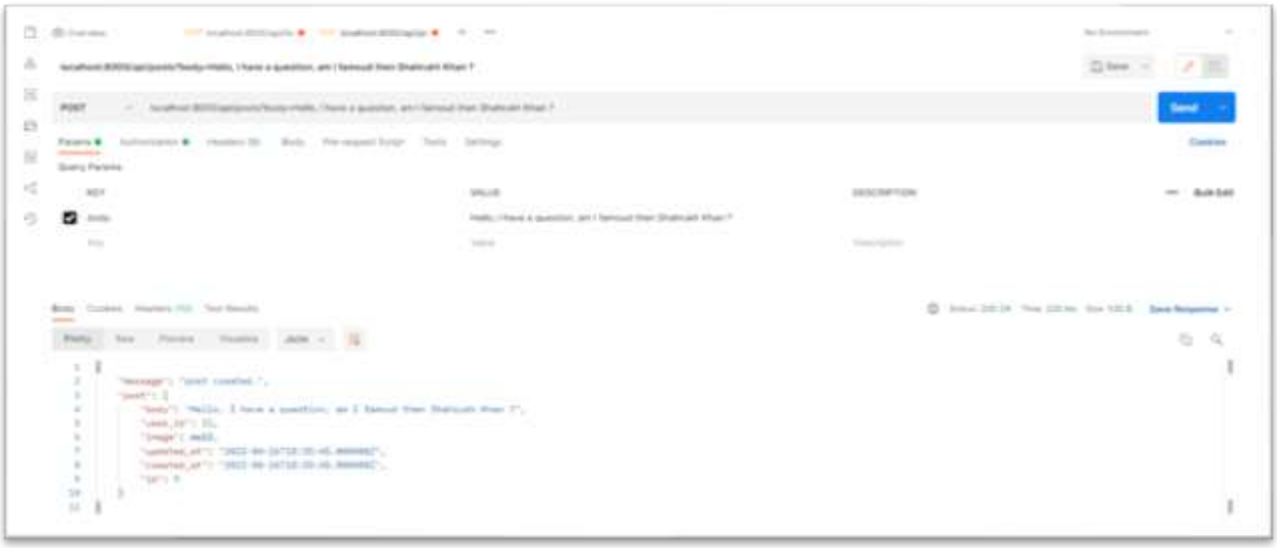
```
{
    "posts": [
        {
            "id": [illegible],
            "user_id": [illegible],
            "body": "[illegible]",
            "image": "[illegible]",
            "created_at": "[illegible]",
            "updated_at": "[illegible]",
            "comments_count": [illegible],
            "likes_count": [illegible],
            "user": {
                "id": [illegible],
                "name": "[illegible]",
                "image": "[illegible]"
            },
            "likes": []
```

Figure 5 Question answer post

Objective	To view detail of a post.
Action	Use post get API in the postman.
Expected Result	A detail of post must be returned.
Actual Result	The detail of a post was printed.
Conclusion	The test was successful.

Table 11 View detail of a post.

4.2.9. Ask question or post a question in a post.



POST localhost:8000/api/posts?body=Hello, I have a question, am I famous then Shahrukh Khan ?

KEY	VALUE	DESCRIPTION
body	Hello, I have a question, am I famous then Shahrukh Khan ?	

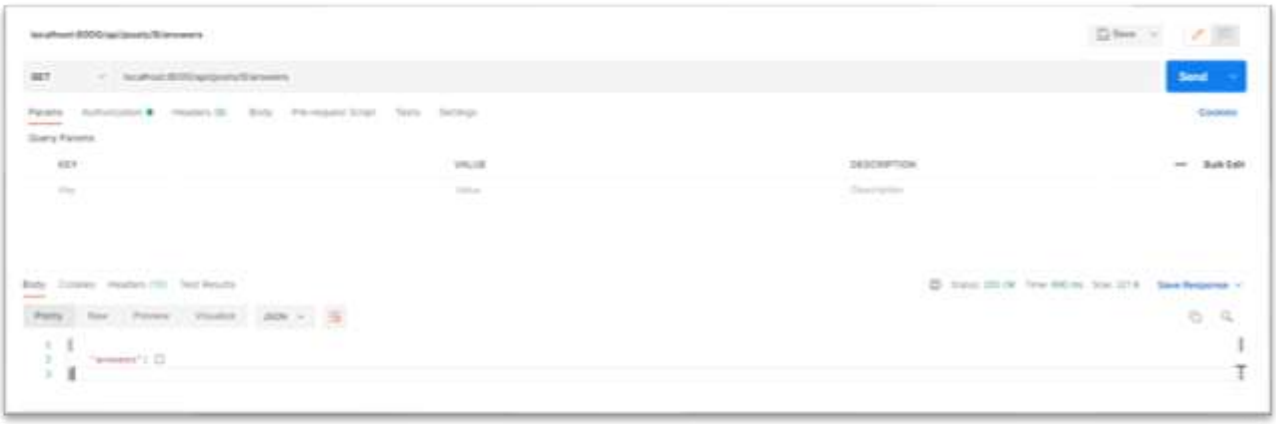
```
{
    "message": "post created.",
    "post": {
        "body": "Hello, I have a question, am I famous then Shahrukh Khan ?",
        [illegible]
    }
}
```

Figure 6 Making a question post.

Objective	To ask a question in a post by the user.
Action	API of a Body of a post was used in the postman.
Expected Result	User must be able to post a question.
Actual Result	User was able to include body part of the post where qeustion was asked.
Conclusion	The test was successful.

Table 12 Unit test of asking question on a post.

4.2.11. Unit test of answering to the posted question.



localhost:8000/api/posts/9/answers

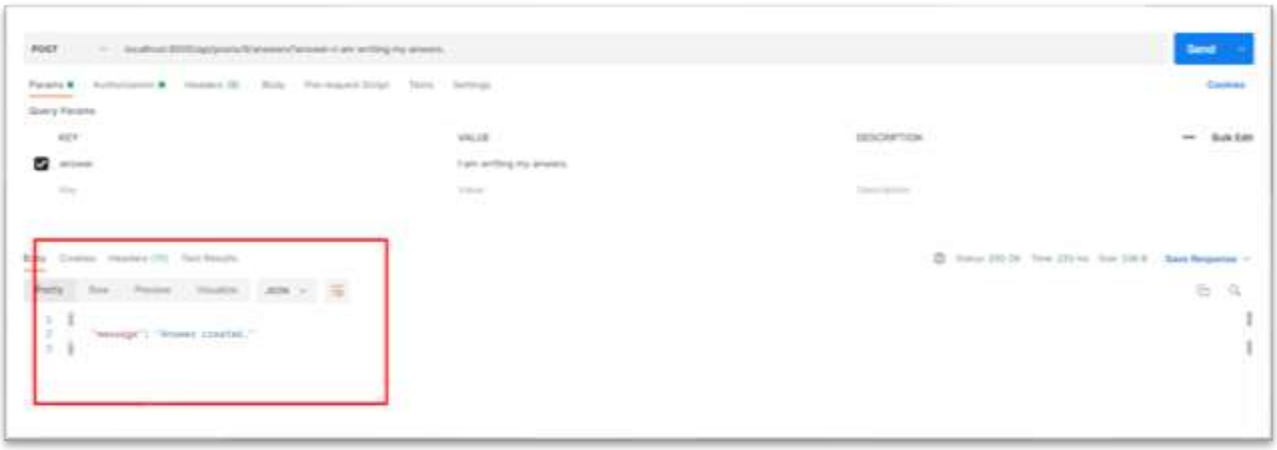
```
{
    "answers": []
}
```

Figure 7 Giving answers on the post id 9

Objective	To view a answer of a post.
Action	View answer API code was used in the postman.
Expected Result	User must be able to view answers written by other users.
Actual Result	User was able to read answer of the post.
Conclusion	The test was successful.

Table 13 Unit test of answering feature.

4.2.12. Getting to view question and answer posts in api unit tests.



localhost:8000/api/posts/9/answers?answer=I am writing my answers.

answer | I am writing my answers.

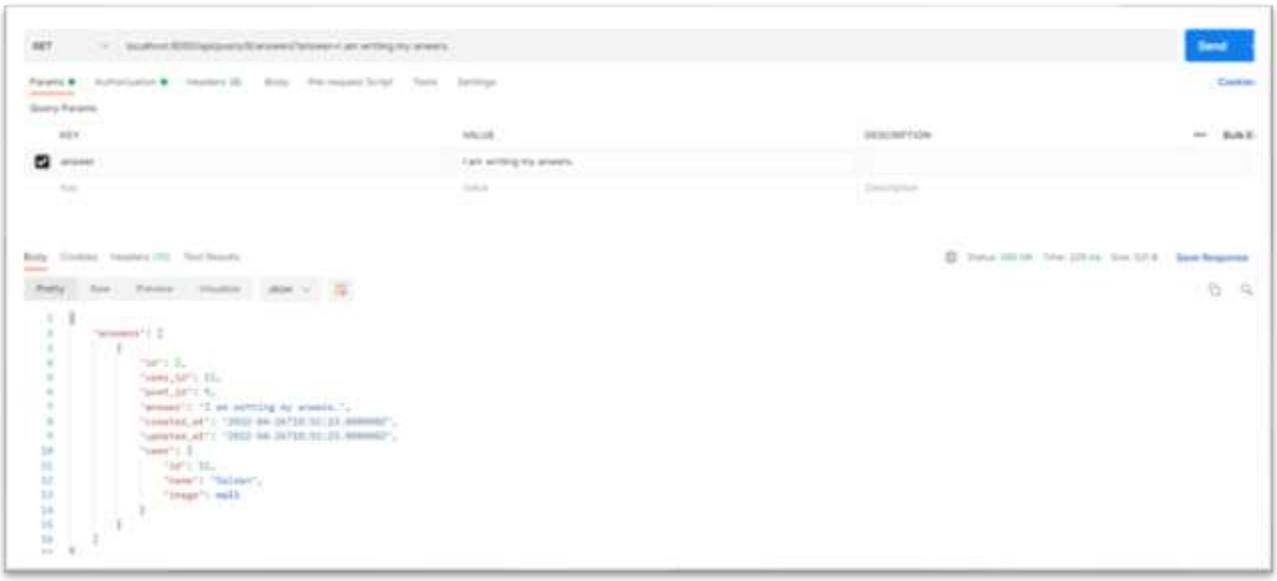
```
{
    "message": "Answer created."
}
```

Figure 8 writting an answer on the post.

Objective	To let user write an anwer to questions asked in a post.
Action	API of writing answer was applied in the postman.
Expected Result	User must be able to anwer to a certain question in the post.

Actual Result	User was able to answer to a question of the post.
Conclusion	The test was successful.

4.2.13. Getting to know the detail about answer of the post.



KEY	VALUE	DESCRIPTION
✓ answer	I am writing my answers.	
Key	Value	Description

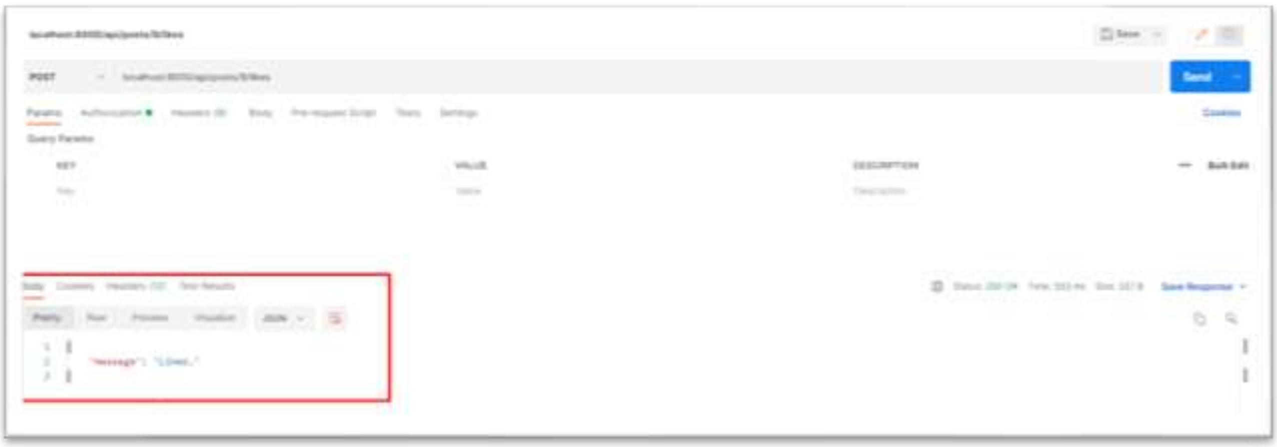
```
{
    "answers": [
        {
            "id": 3,
            "user_id": 11,
            "post_id": 4,
            "answer": "I am writing my answers.",
            "created_at": "2022-04-26T10:51:23.000000Z",
            "updated_at": "2022-04-26T10:51:23.000000Z",
            "user": {
                "id": 11,
                "name": "Salman",
                "image": null
            }
        }
    ]
}
```

Figure 9 API test to view answer written.

Objective	To view detail of the anwer of a post.
Action	Use API of view anwer of a particular post in a postman.
Expected Result	User must be able to view detail of answer.
Actual Result	User was able to view the detail of answer, like who created and to which question does it belong.
Conclusion	The test was successful.

Table 14 Unit test of intigration to view detail of answer

4.2.14. **Getting to know the detail about anwer of the post.**



```
{
    "message": "Liked."
}
```

Figure 10 Liking on post.

Objective	To let users like a post.
Action	Insert like API in the postman.
Expected Result	User must be able like or dislike a post.
Actual Result	User was able to like and dislike a post.
Conclusion	The test was successful.

Table 15 API test to like a post.

1.3. System Testing

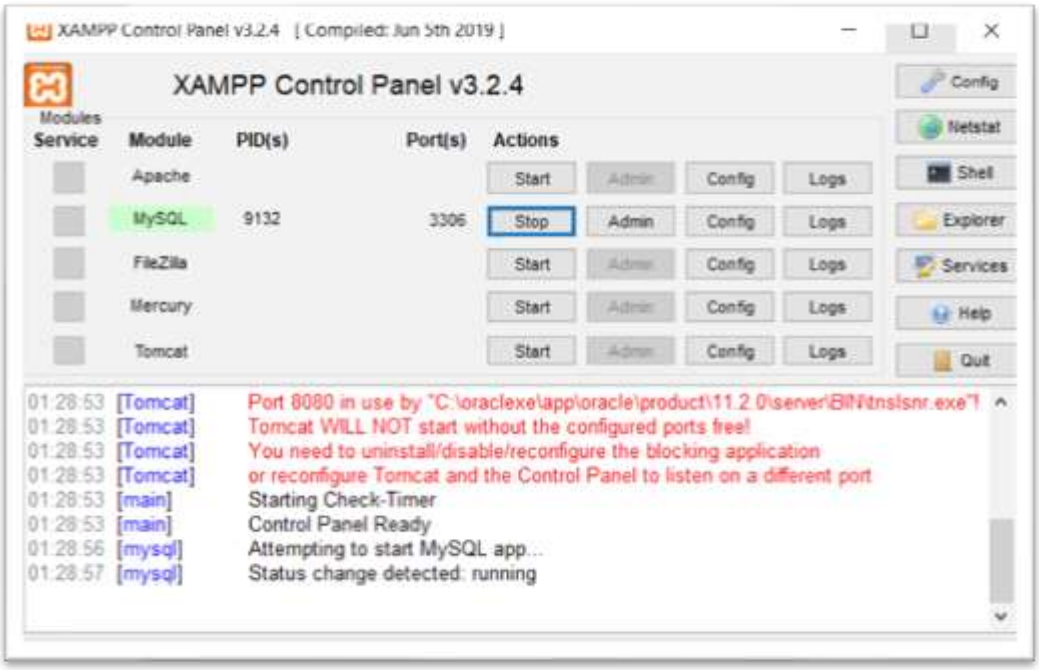
1.3.1. Starting all the application

System testing is all about testing the whole live project after integrating API's with flutter and store data in the database. Especially there were three different phases done to complete system testing:

i) Start mysql database using Xampp.

ii) Visual Studio Code to start and it's terminal to host the Laravel API

iii) Build the whole app in my android phone using VS code.


XAMPP Control Panel v3.2.4 [Compiled: Jun 5th 2019]
XAMPP Control Panel v3.2.4
Config
Netstat
Shell
Explorer
Services
Help
Quit

Modules Service	Module	PID(s)	Port(s)	Actions			
	Apache			Start	Admin	Config	Logs
	MySQL	9132	3306	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

```
01:28:53  [Tomcat]   Port 8080 in use by "C:\oraclexe\app\oracle\product\11.2.0\server\BIN\tnslsnr.exe"!
01:28:53  [Tomcat]   Tomcat WILL NOT start without the configured ports free!
01:28:53  [Tomcat]   You need to uninstall/disable/reconfigure the blocking application
01:28:53  [Tomcat]   or reconfigure Tomcat and the Control Panel to listen on a different port
01:28:53  [main]     Starting Check-Timer
01:28:53  [main]     Control Panel Ready
01:28:56  [mysql]    Attempting to start MySQL app...
01:28:57  [mysql]    Status change detected: running
```

Figure 11 Starting mysql database using xampp.

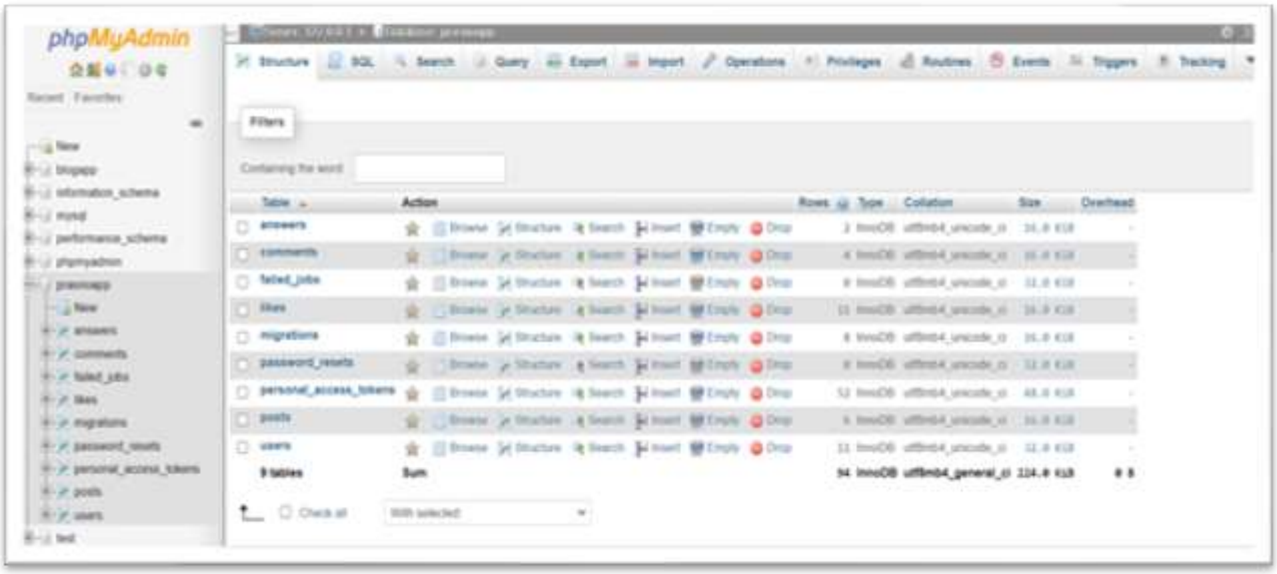
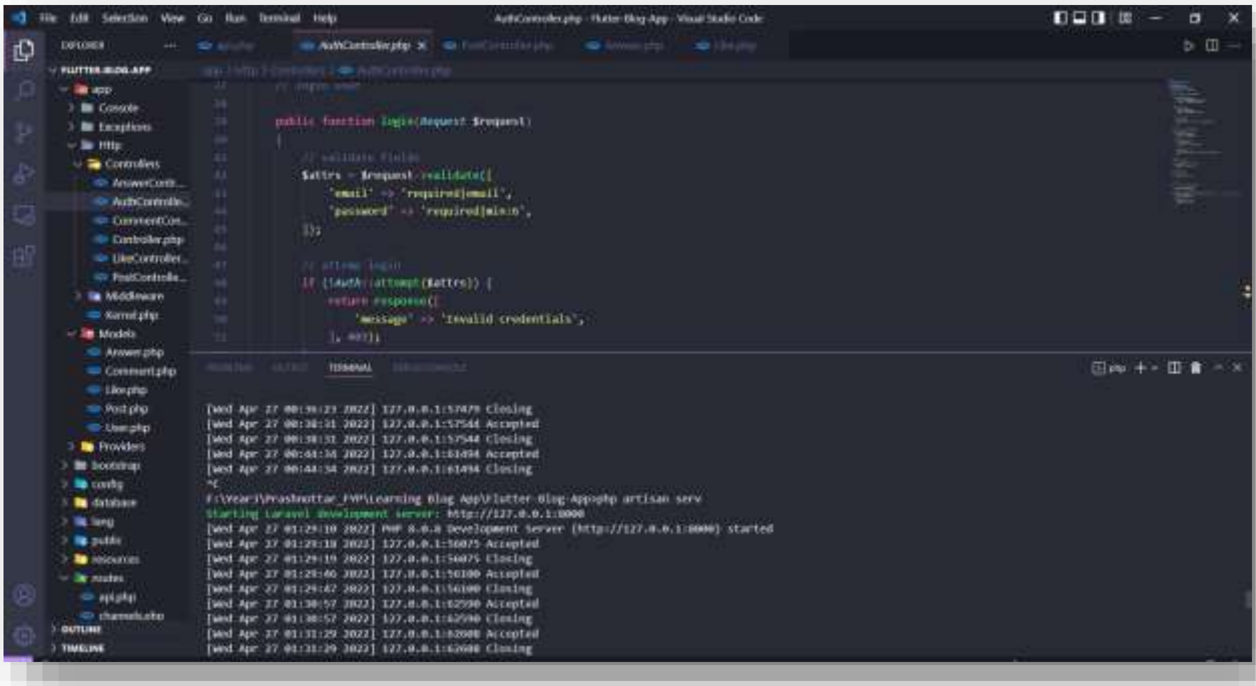


	Table	Action						Rows	Type	Collation	Size	Overhead
☐	answers	Browse	Structure	Search	Insert	Empty	Drop	3	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
☐	comments	Browse	Structure	Search	Insert	Empty	Drop	4	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
☐	failed_jobs	Browse	Structure	Search	Insert	Empty	Drop	0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
☐	likes	Browse	Structure	Search	Insert	Empty	Drop	11	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
☐	migrations	Browse	Structure	Search	Insert	Empty	Drop	8	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
☐	password_resets	Browse	Structure	Search	Insert	Empty	Drop	0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
☐	personal_access_tokens	Browse	Structure	Search	Insert	Empty	Drop	12	InnoDB	utf8mb4_unicode_ci	48.0 KiB	-
☐	posts	Browse	Structure	Search	Insert	Empty	Drop	5	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
☐	users	Browse	Structure	Search	Insert	Empty	Drop	11	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
	9 tables	Sum						54	InnoDB	utf8mb4_general_ci	224.0 KiB	0 B

Figure 12 Database prasnoapp of Prashnottar

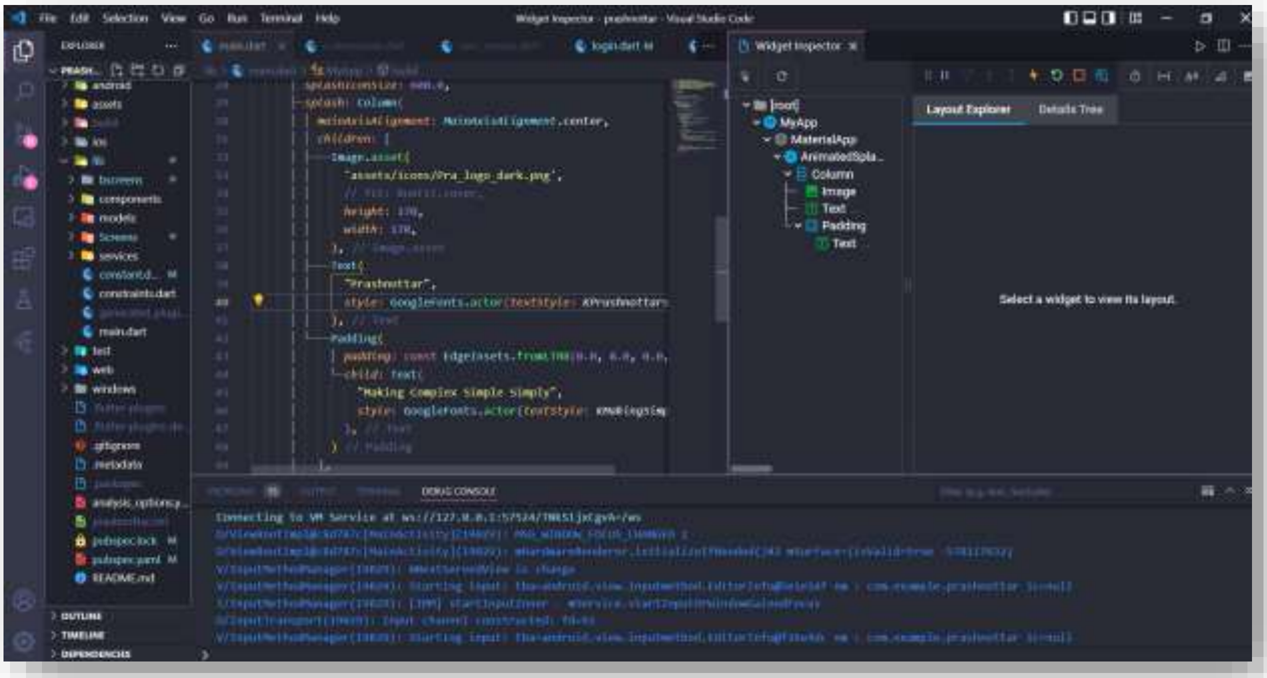


```
public function login(Request $request)
{
    // validate fields
    $attrs = $request->validate([
        'email' => 'required|email',
        'password' => 'required|min:6',
    ]);

    // attempt login
    if (!Auth::attempt($attrs)) {
        return response([
            'message' => 'Invalid credentials',
        ], 403);
```

```
[Wed Apr 27 00:36:23 2022] 127.0.0.1:57479 Closing
[Wed Apr 27 00:38:31 2022] 127.0.0.1:57544 Accepted
[Wed Apr 27 00:38:31 2022] 127.0.0.1:57544 Closing
[Wed Apr 27 00:44:34 2022] 127.0.0.1:61494 Accepted
[Wed Apr 27 00:44:34 2022] 127.0.0.1:61494 Closing
^C
F:\Year3\Prashnottar_FYP\Learning Blog App\Flutter-Blog-App>php artisan serv
Starting Laravel development server: http://127.0.0.1:8000
[Wed Apr 27 01:29:10 2022] PHP 8.0.0 Development Server (http://127.0.0.1:8000) started
[Wed Apr 27 01:29:18 2022] 127.0.0.1:56075 Accepted
[Wed Apr 27 01:29:19 2022] 127.0.0.1:56075 Closing
[Wed Apr 27 01:29:46 2022] 127.0.0.1:56100 Accepted
[Wed Apr 27 01:29:47 2022] 127.0.0.1:56100 Closing
[Wed Apr 27 01:30:57 2022] 127.0.0.1:62590 Accepted
[Wed Apr 27 01:30:57 2022] 127.0.0.1:62590 Closing
[Wed Apr 27 01:31:29 2022] 127.0.0.1:62608 Accepted
[Wed Apr 27 01:31:29 2022] 127.0.0.1:62608 Closing
```

Figure 13 Starting laravel project in VScode terminal.



```
splashIconSize: 600.0,
splash: Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    Image.asset(
      'assets/icons/Pra_logo_dark.png',
      // fit: BoxFit.cover,
      height: 170,
      width: 170,
    ), // Image.asset
    Text(
      "Prashnottar",
      style: GoogleFonts.actor(textStyle: kPrashnottar
    ), // Text
    Padding(
      padding: const EdgeInsets.fromLTRB(0.0, 0.0, 0.0,
      child: Text(
        "Making Complex Simple Simply",
        style: GoogleFonts.actor(textStyle: kMakingSim
      ), // Text
    ) // Padding
```

```
Connecting to VM Service at ws://127.0.0.1:57524/TRKS1jxCgvA=/ws
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

Figure 14 Starting android app in android phone using VS Code.

1.3.2. App icon in mobile phone

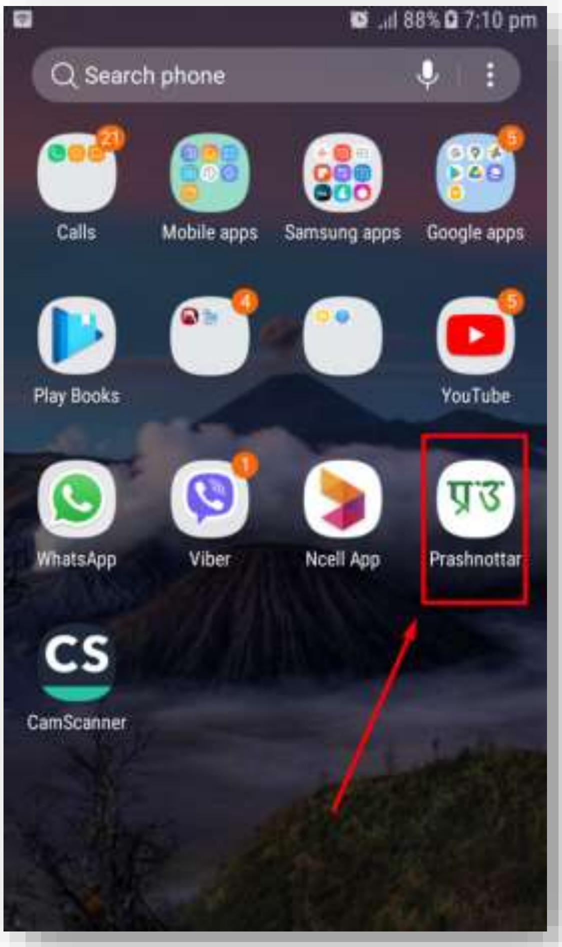


Figure 15 To test if app icon is shown in the android phone.

Objective	To see if app logo is shown in android phone.
Action	Run the app in phone using USB cable and run it using VS Code.
Expected Result	The logo must be shown.
Actual Result	The logo was shown.
Conclusion	The test was successful.

Figure 16 System testing of app logo.

1.3.3. Registering in the PT

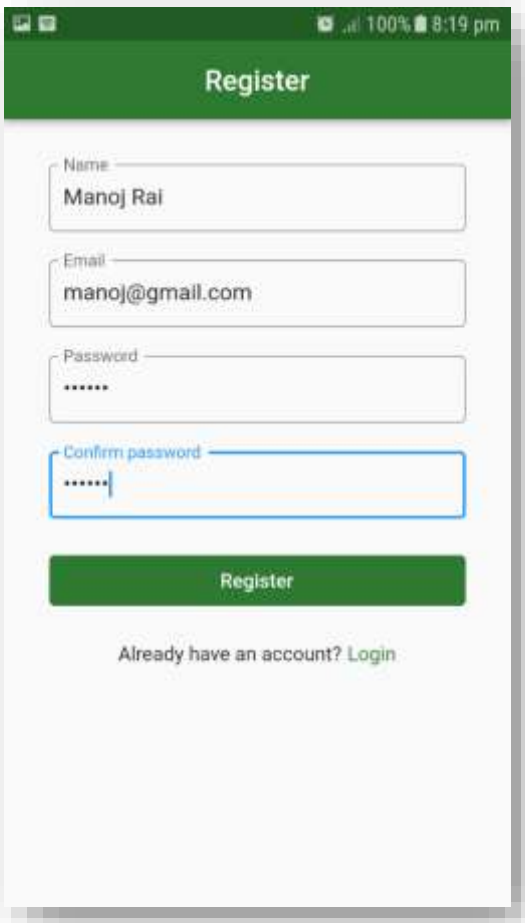


Figure 17 New registration in PT.

Objective	To see if a new user is registered in the app or not.
Action	Run the program and try to fill in the form with non registered email.
Expected Result	The registration must be done.
Actual Result	The registration was successful.
Conclusion	The test was successful.

Figure 18 System testing 2 of user registration.

4.3.4. Log in to PT

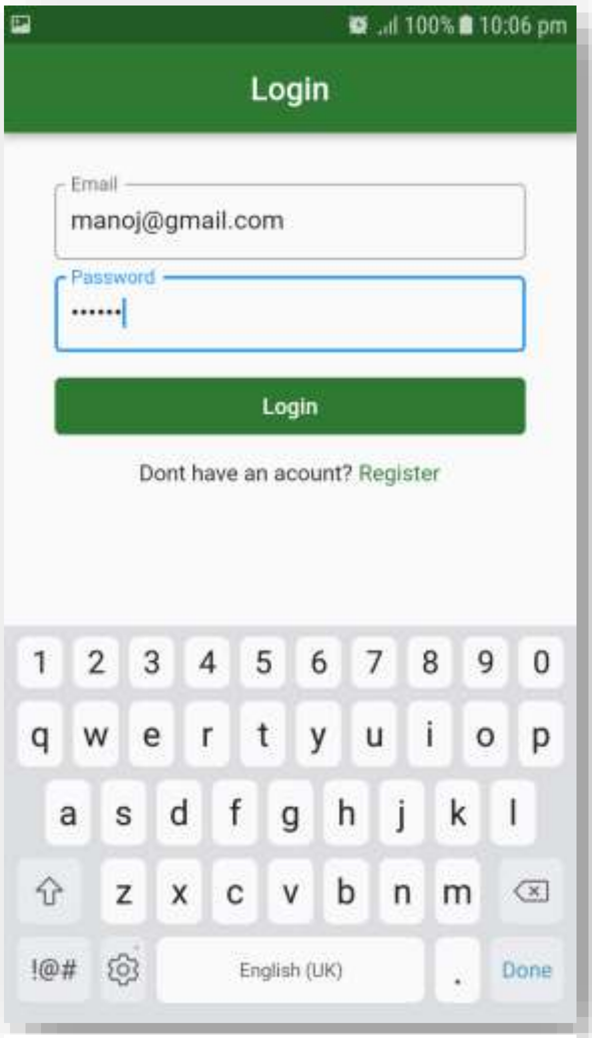


Figure 20 Log in to the PT with previous registered account.

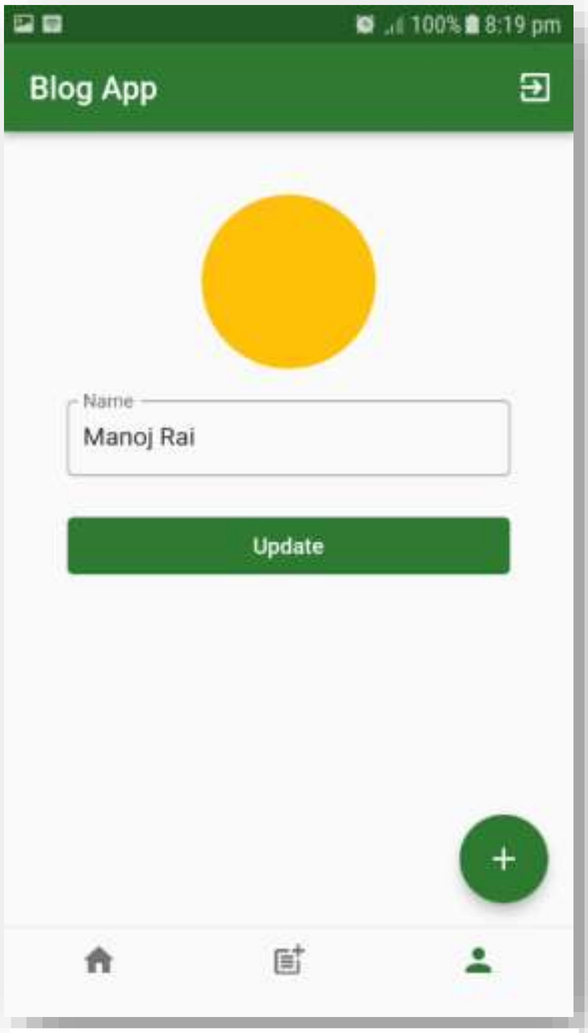


Figure 19 Successfully logged in

Objective	To log in with registered account.
Action	Fill the form with valid email and password.
Expected Result	Afer log in, user must be taken to the homepage.
Actual Result	User was successfully logged in.
Conclusion	The test was successful.

Figure 21 User log in system testing.

4.3.5. Choose Class

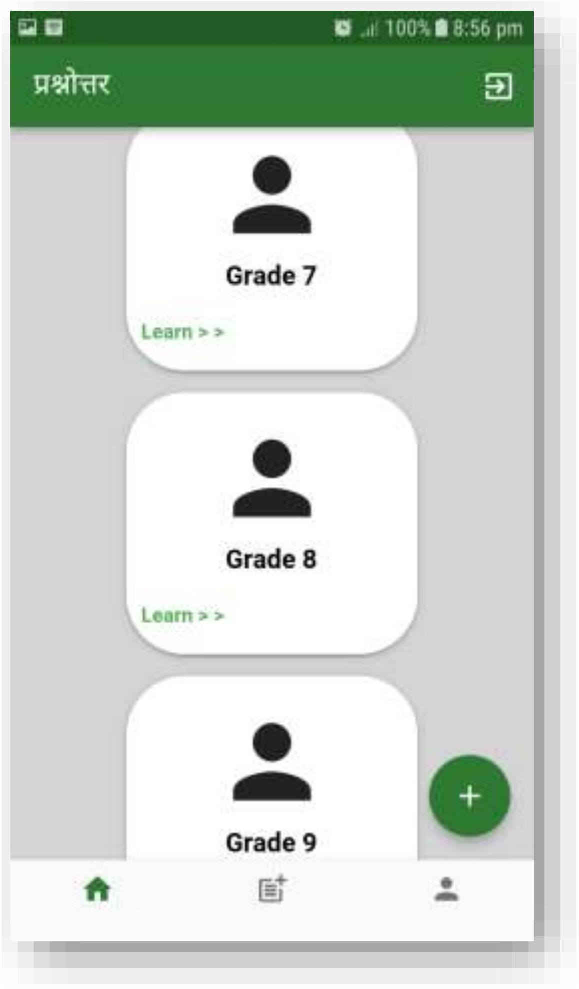


Figure 22 Choose class of the PT.

Objective	To allow user to choose between class 8-10.
Action	To click on a respective class tile.
Expected Result	User must be taken to subject screen.
Actual Result	User was taken to the subject screen.
Conclusion	The test was successful.

Figure 23 System testing to class selection.

4.3.6. Choose Subject

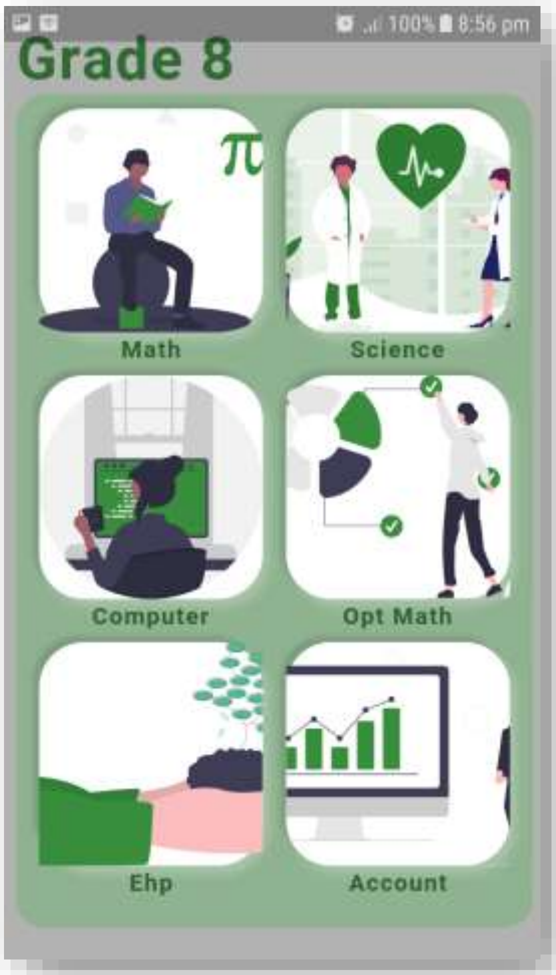
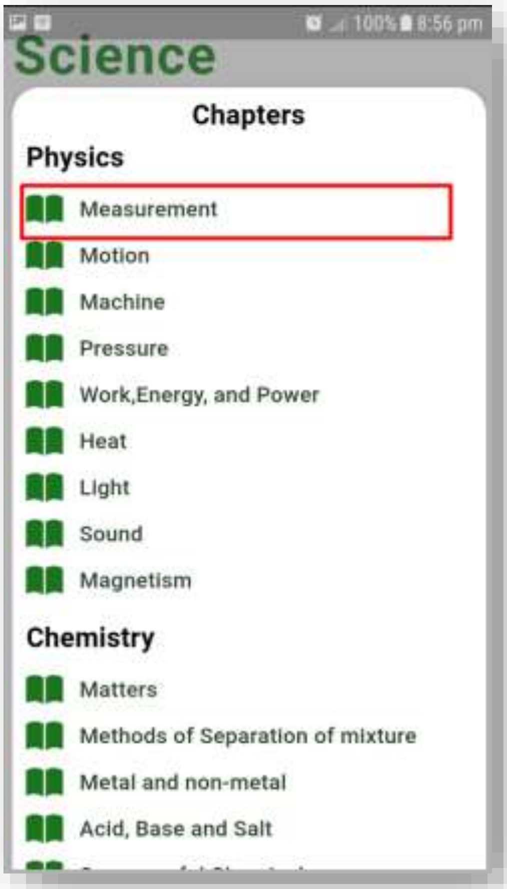


Figure 24 Choose subject of grade 8

Objective	To let user select a particular subject.
Action	To touch a tile of a particular subject.
Expected Result	User must be taken to subject's chapter screen.
Actual Result	User was successfully taken to chapter screen.
Conclusion	The test was successful.

Figure 25 Select a partifular subject.

4.3.7. Choose Chapter of the Subject


Science

Chapters

Physics

- Measurement
- Motion
- Machine
- Pressure
- Work,Energy, and Power
- Heat
- Light
- Sound
- Magnetism

Chemistry

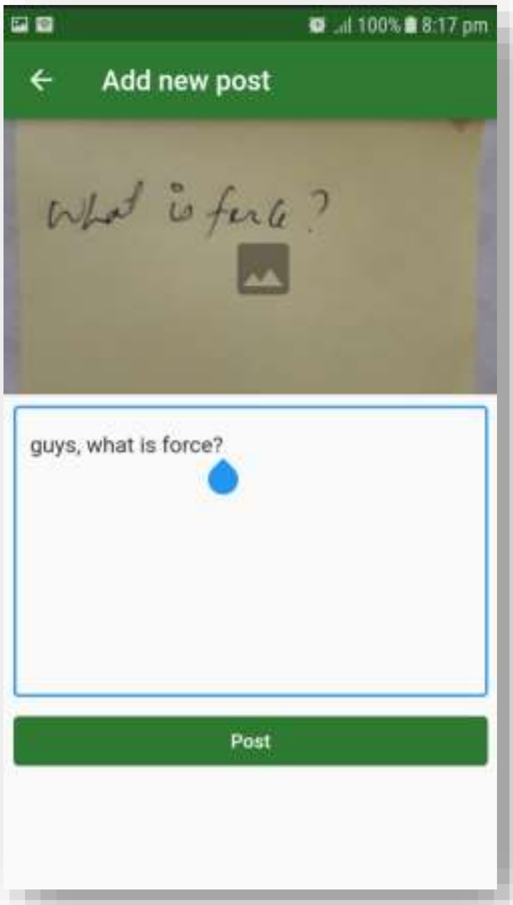
- Matters
- Methods of Separation of mixture
- Metal and non-metal
- Acid, Base and Salt

Figure 26 Choose chapter from Science Subject.

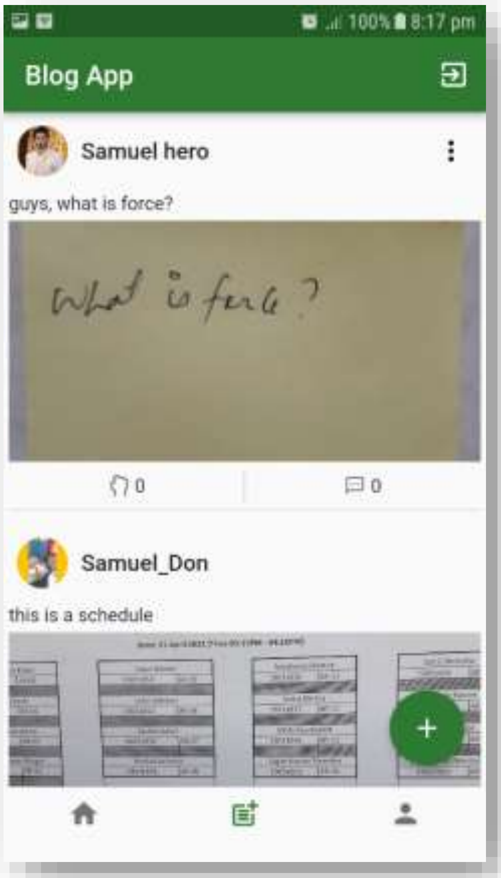
Objective	To let user select a particular subject among other subjects.
Action	Touch a name of the chapter.
Expected Result	Take it to the blog section where different questions related to the chapter would be asked.
Actual Result	It took to the blog section.
Conclusion	The test was successful.

Figure 27 System testing of selecting a chapter.

4.3.8. Make question post with Sharing picture



guys, what is force?



guys, what is force?

this is a schedule

Figure 29 Ask a question in a post sharing a picture.

Figure 28 Question post successfull.

Objective	To be able to ask a question asking a question.
Action	Click that floating plus button and just ask a question with addition of a picture.
Expected Result	User must be able to ask a question in a blog section.
Actual Result	User was able to make a question post in blog section.
Conclusion	The test was successful.

Figure 30 System test of asking a question.

4.3.9. Edit post

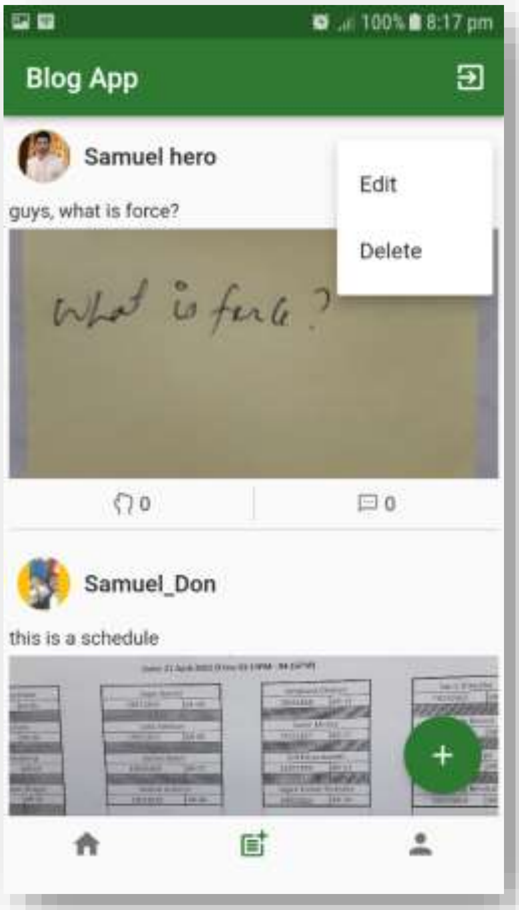


Figure 32 Editing a post

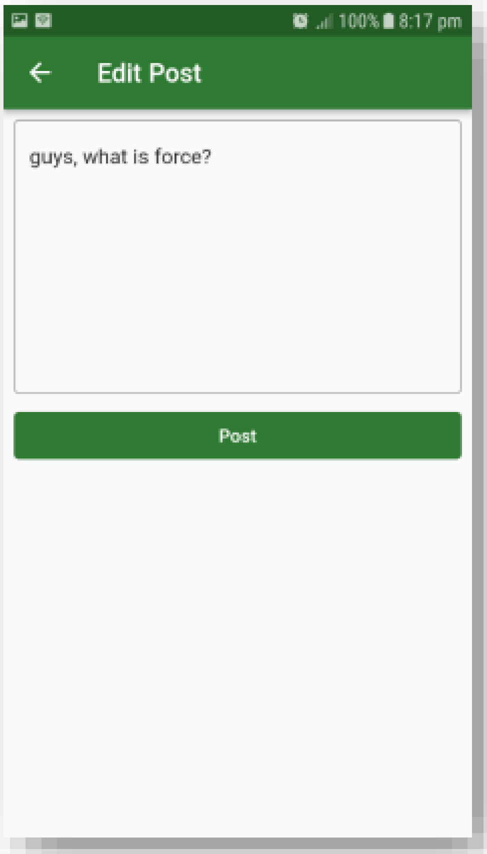
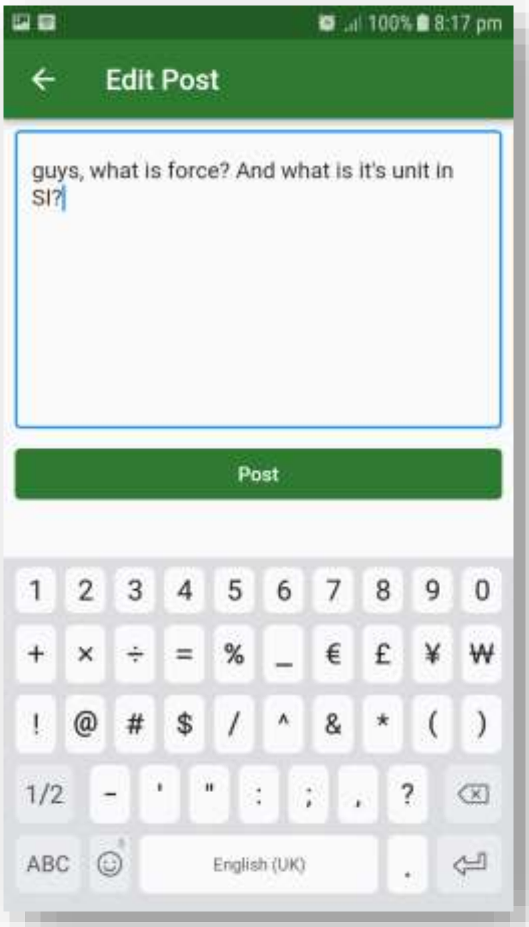
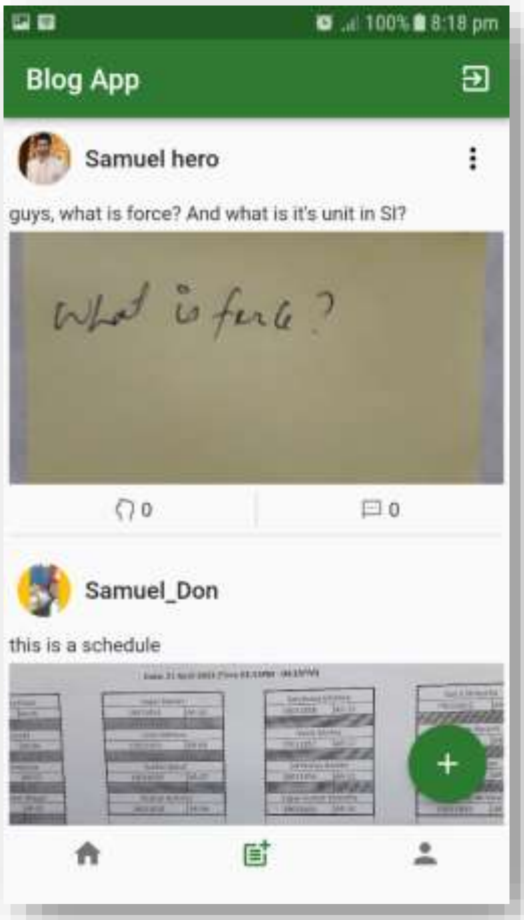


Figure 31 Editing the question post.



guys, what is force? And what is it's unit in SI?



guys, what is force? And what is it's unit in SI?

this is a schedule

Figure 33 Successfully post edited.

Figure 34 Editing the post

Objective	Just let user to edit their question post.
Action	Click on those three dots, and click edit and edit the post.
Expected Result	To be able to edit and save post.
Actual Result	Blog was able to be edited.
Conclusion	The test was successful.

Figure 35 System test of edit post.

4.3.11. Delete Post

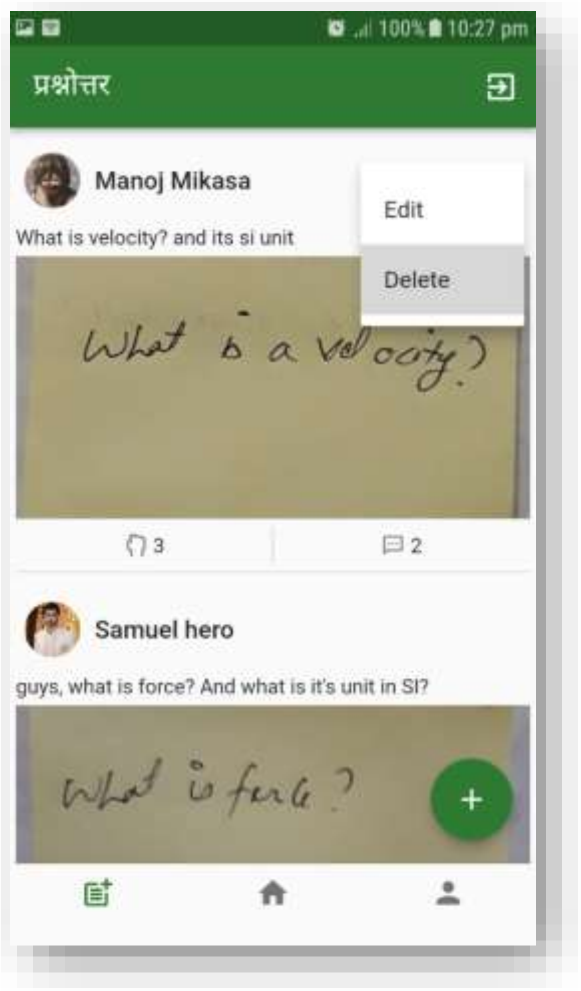


Figure 37 Deleting a post

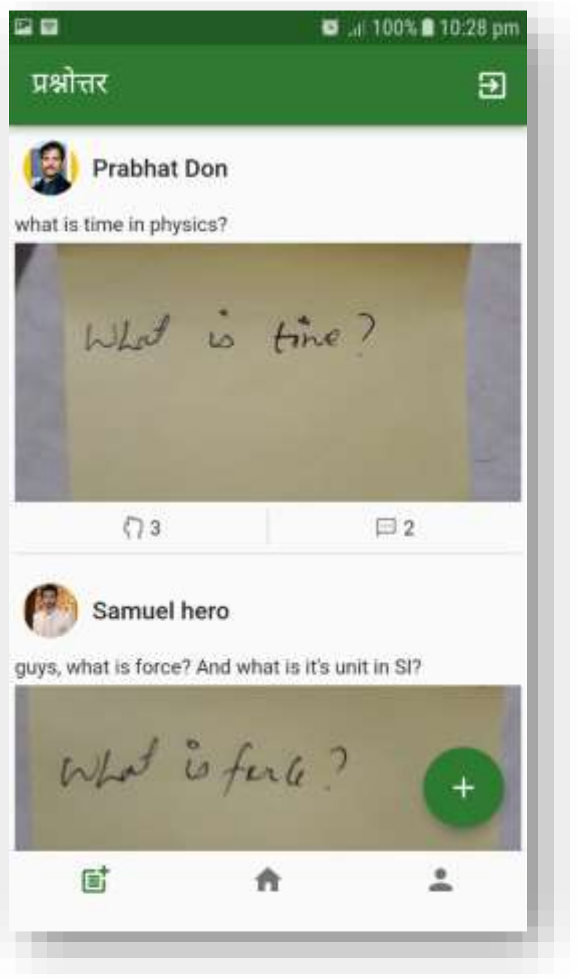


Figure 36 Post deleted succesfully.

Objective	Let user delete their post.
Action	Click on three dots and click delete.
Expected Result	The post must be deleted along with its likes and answers.
Actual Result	The question post was deleted successfully.
Conclusion	The test was successful.

Figure 38 System test of delete question post.

4.3.12. Answer the post

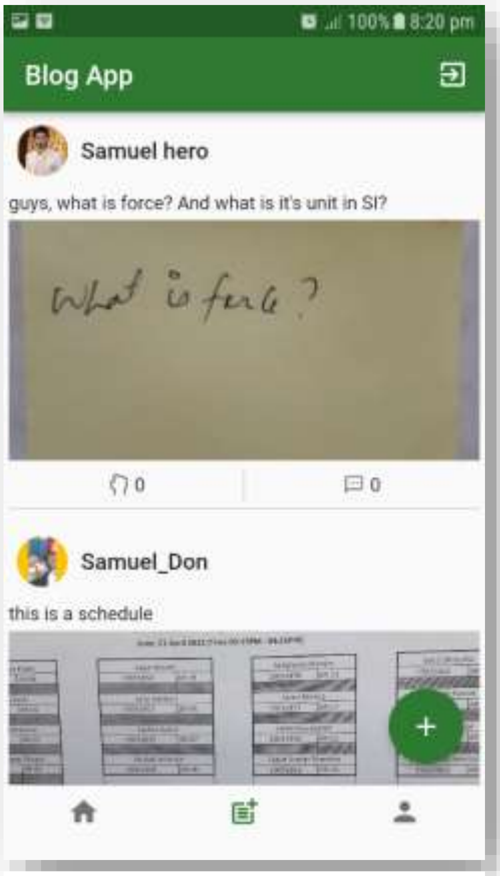
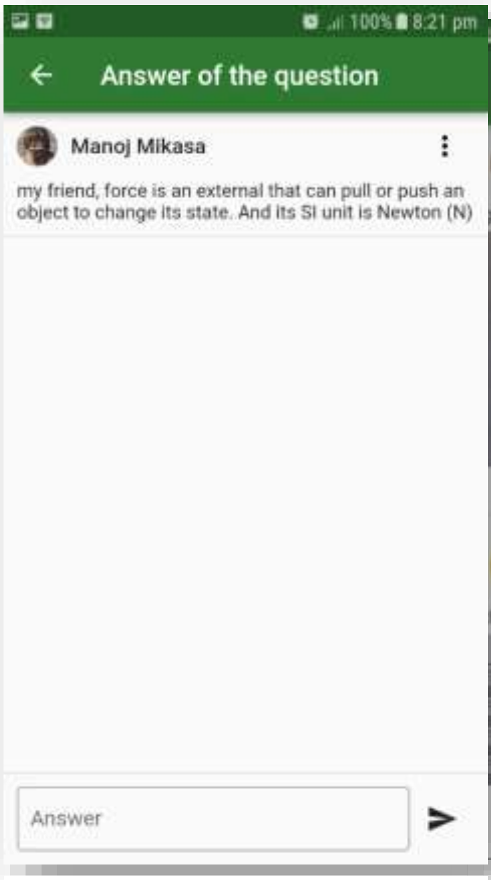


Figure 41 The post doesn't has any of the answer



my friend, force is an external that can pull or push an object to change its state. And its SI unit is Newton (N)

Figure 40 Writting an answer to the post.

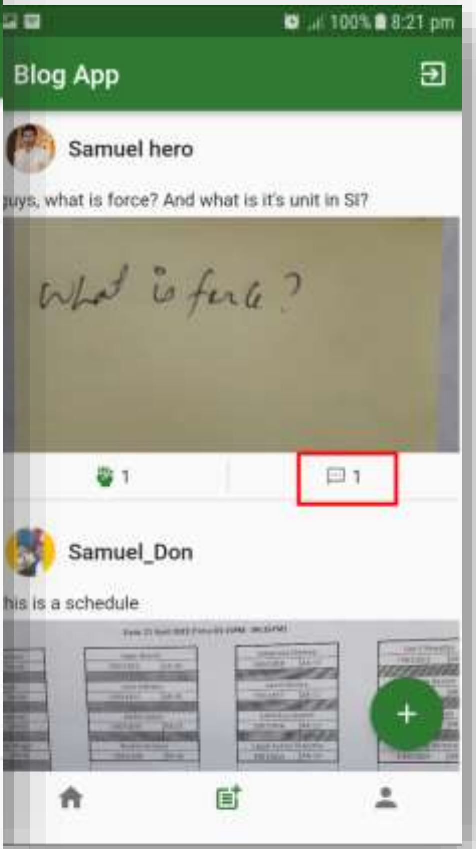
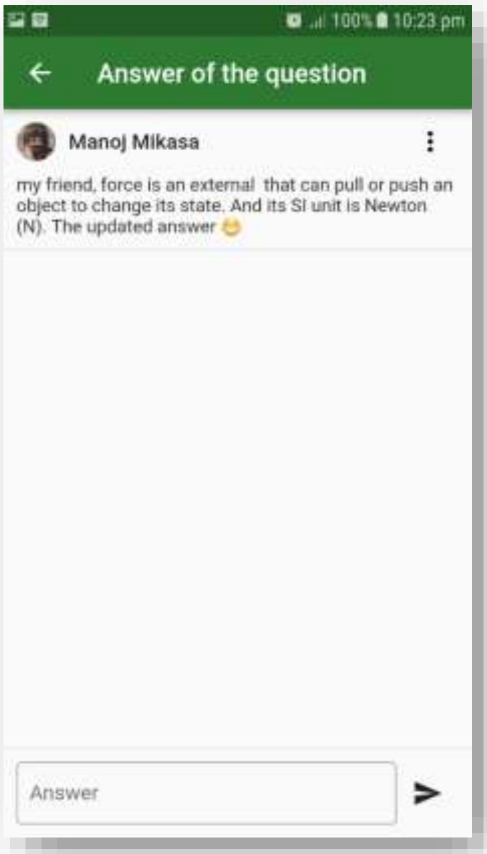


Figure 39 Answer given successfully to the post.

Objective	Let user answer to the question of a post.
Action	Just click that message icon and add answer to the post.
Expected Result	The answer must be saved with a post.
Actual Result	The answer were successfully saved and shown to others.
Conclusion	The test was successfully.

Figure 42 System testing of anwering to a question post.

4.3.13. Edit answer



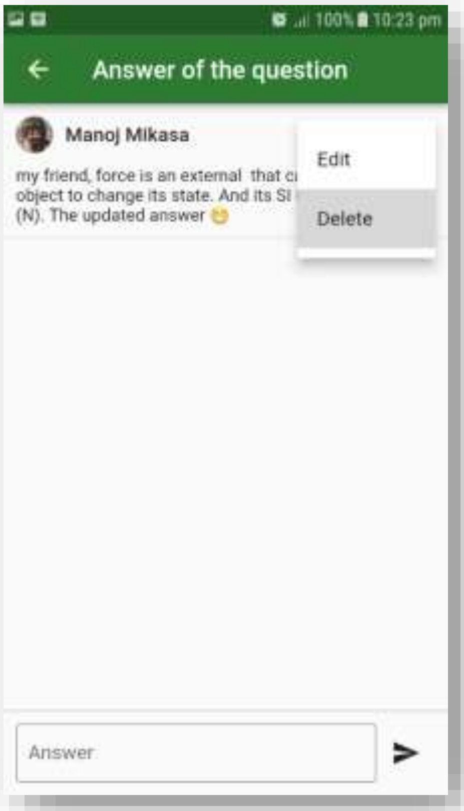
my friend, force is an external that can pull or push an object to change its state. And its SI unit is Newton (N). The updated answer 😊

Figure 43 Answer updated successfully.

Objective	To let users edit their answer.
Action	Click to the three dots of answer and edit the answer.
Expected Result	Answer must be edited successfully.
Actual Result	Answer was edited.
Conclusion	The test was successfully.

Figure 44 System test of editing an answer.

4.3.14. Delete Answer



my friend, force is an external that c object to change its state. And its SI (N). The updated answer 😊

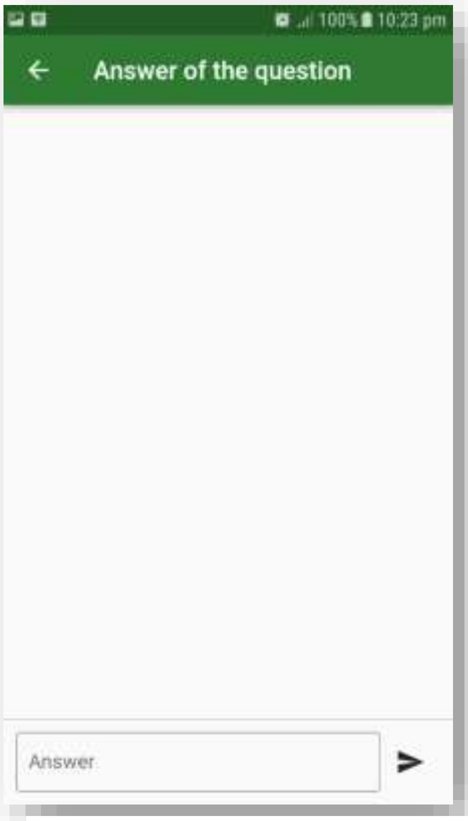
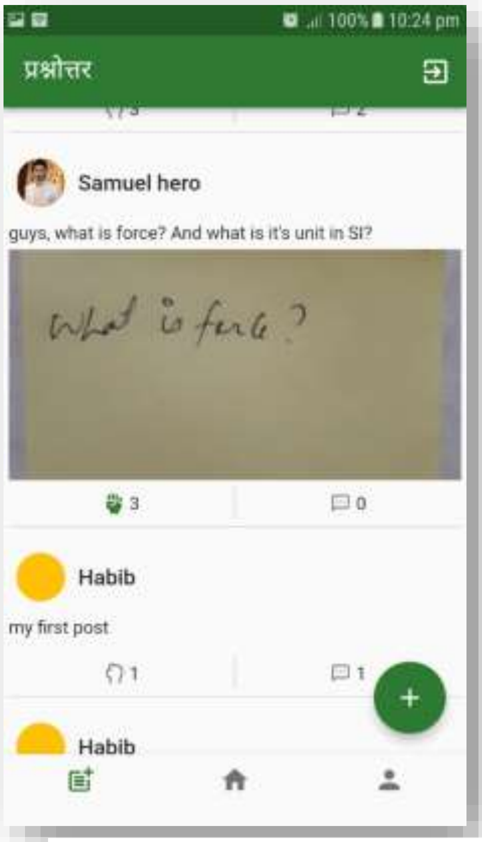


Figure 46 Answer deleted successfully.



guys, what is force? And what is it's unit in SI?

my first post

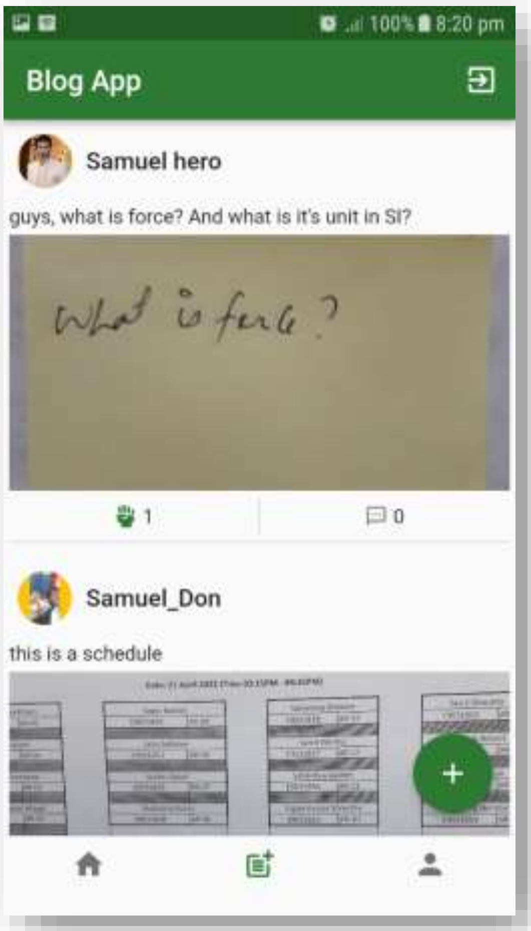
Figure 45 No answer in the post..

Figure 47 Deleting an answer.

Objective	To let user delete their answer.
Action	Just click the three dots, and delte the post.
Expected Result	Answer must be deleted.
Actual Result	The answer was deleted successfully.
Conclusion	The test was successful.

Figure 48 System testing of deleting an answer.

4.3.15. Like the post


Blog App

Samuel hero

guys, what is force? And what is it's unit in SI?

1

0

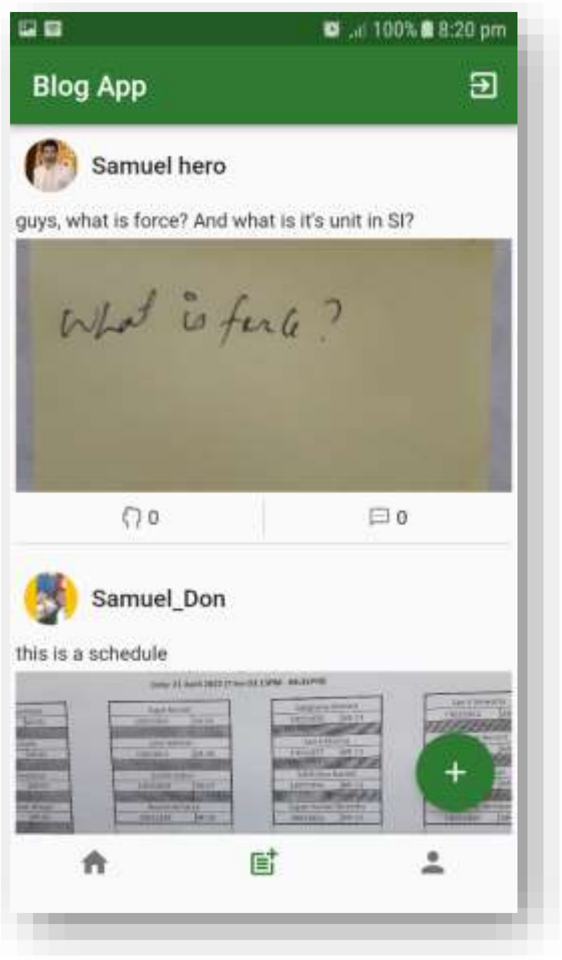
Samuel_Don

this is a schedule

Figure 49 Post is liked.

Objective	To let user like a post for it's question and answer popularity.
Action	Click the fist icon.
Expected Result	The icon must turn green and increase the count by one.
Actual Result	As expected.
Conclusion	The test was successful.

4.3.16. Dislike the post



guys, what is force? And what is it's unit in SI?

What is force?

this is a schedule

Figure 50 Post is disliked.

Objective	To dislike a post.
Action	Click the liked fist icon.
Expected Result	It must turn black and the count must decrease.
Actual Result	As expected, the icon was turned black and it's count decreased.
Conclusion	The test was successful.

4.3.17. Edit Update profile

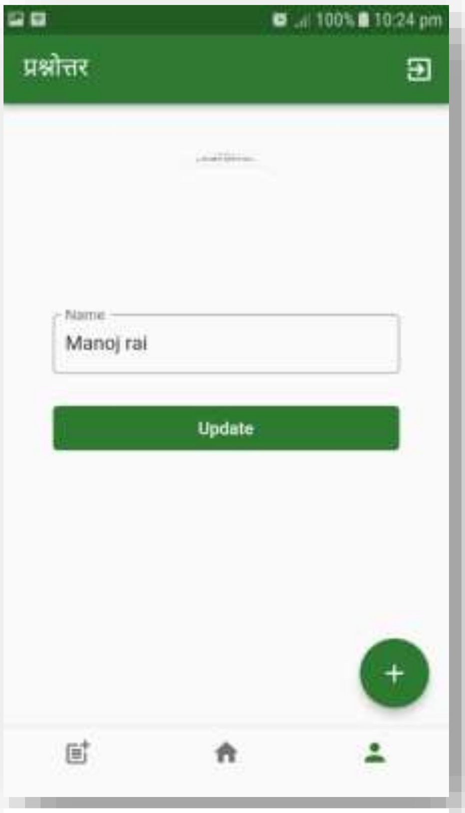


Figure 53 User Manoj Rai to be updated.

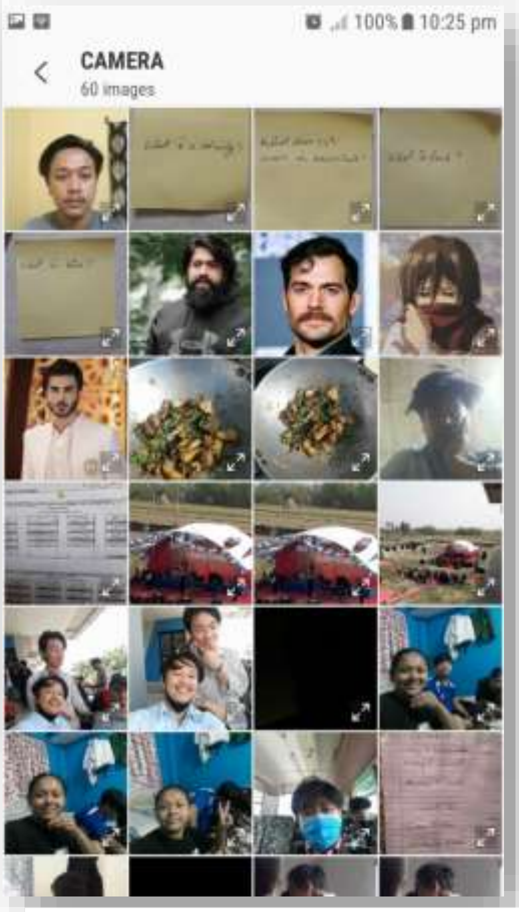


Figure 52 Selecting image from the gallery.

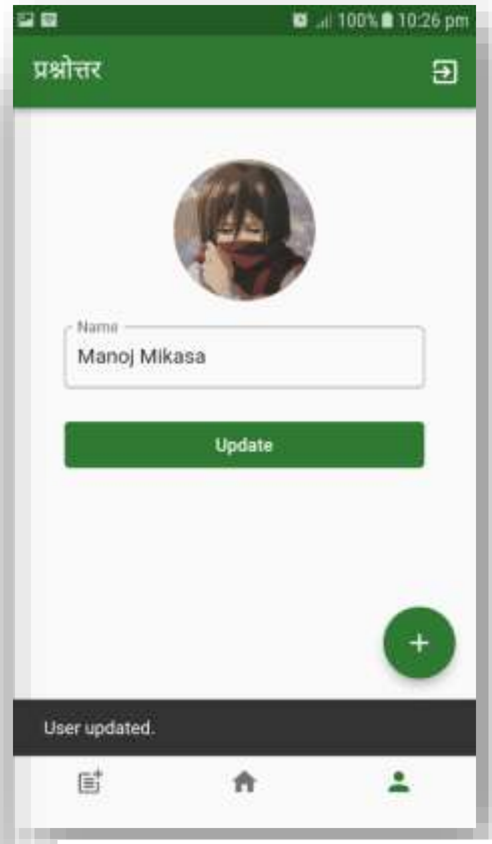


Figure 51 User profile and name updated successfully.

Objective	User must be able to edit their name and profile picture.
Action	User must click to the user icon in the nav bar and click on the picture and name, edit those and click update button.
Expected Result	Profile picture and name must be changed.
Actual Result	The name and picture were changed.
Conclusion	The test was successful.

Figure 54 System test of editing user profile.

4.4. Critical Analysis

The two tests: Unit testing and System testing showed all functionality of features present in the project that they worked as expected with no bugs. The results were as follows:

i) All the API's worked fine.
ii) All the API's were properly integrated with flutter app.
iii) The performance of the system was nice.
iv) The UI was very simple and user friendly.
v) Bookmark feature would be very helpful for students to refer it in exam time, so it would be better to include them.
vi) The whole project made from scratch was supported by an android phone, laptop, laravel, flutter and mysql databases.

The app seem to be very simple so it has to include features like follow person, and bookmark posts. In coming days hopefully those features would also be inlcuded.