
Test Document

for

SafeSpace

Version 1.0

Prepared by

Group #: 10

Anirudh Singh	230142
Archita Goyal	230187
Brinda Fadadu	230307
Nakul Patel	230676
Naman Yadav	230680
Om Chaudhari	230715
Rohit Yadav	230873
Sayani Patra	230943
Vivek	231169
Yatharth Sharma	231198

Group Name: 404 Team Not Found

sanirudh23@iitk.ac.in
architag23@iitk.ac.in
brindaf23@iitk.ac.in
nakulpatel23@iitk.ac.in
namanyadav23@iitk.ac.in
omcc23@iitk.ac.in
rohity23@iitk.ac.in
sayanip23@iitk.ac.in
vivek23@iitk.ac.in
yatharths23@iitk.ac.in

Course: CS253

Mentor TA: Souvik Mukherjee

Date: 5 April 2025

CONTENTS.....	2
REVISIONS.....	3
1 INTRODUCTION.....	4
2 UNIT TESTING.....	2
3 INTEGRATION TESTING.....	3
4 SYSTEM TESTING.....	4
5 CONCLUSION.....	5
APPENDIX A - GROUP LOG.....	6

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	Group 10 404 Team Not Found	Completed version of Test Document of the web app SafeSpace built by Group 10.	05/04/2025

1 Introduction

Test Strategy

SafeSpace is a confidential and supportive online platform designed exclusively for students at IITK. The platform aims to address students' mental health challenges and provide a safe environment to connect with peers and counselors from the Institute Counseling Service. Its primary objective is to foster mental wellness, open communication, and mutual understanding within the student community.

Our team's testing strategy for such an app with a plethora of options was manual testing wherein each of the testers would individually test the app for various inputs or cases and match the obtained output with the desired output.

The testers have subjected the app to different possible input combinations and verified the proper response of the app as and when required. The tests not only consisted of proper verification of output on the app (frontend), but also consisted of constant verification for the proper entries in the database (backend). This leads to a precise check of all the functional and non-functional requirements to be fulfilled by the app as mentioned in the SRS document.

Testing Period of the App

The testing was an integral part of the development process. During the implementation, the app was subjected to tests corresponding to basic functions such as logins, bookings, chatting, posting etc.

However, the majority of the testing of the app was done after the implementation of the app. Since the basic functionality of the app was set in stone and tested, any bugs encountered in the app were easier to tackle since they involved changes only in specific features of the app (certain functions of the source code). This made testing a lot more efficient and less time-consuming.

Testers of the App

The developers of the app were the testers themselves. The benefit of having the developers being the testers was that we knew the different possible inputs and hence were able to carry out extensive testing to ensure the robustness of the application. Multiple errors were identified and corrected as and when discovered.

Coverage Criterion for Testing

Functional requirements were the criterion used for the testing.

Tools used in testing

We used Postman for backend testing, a tool that enables testers to send HTTP requests to the server and view responses in a unified interface. It was utilized to test all backend endpoints.. The entire frontend was manually tested.

2 Unit Testing

1. POST /login

The backend function for login was tested. All the testable branches functioned as expected.

Unit Details: Backend function to handle Post Request “/login”

Test Owner: Yatharth Sharma

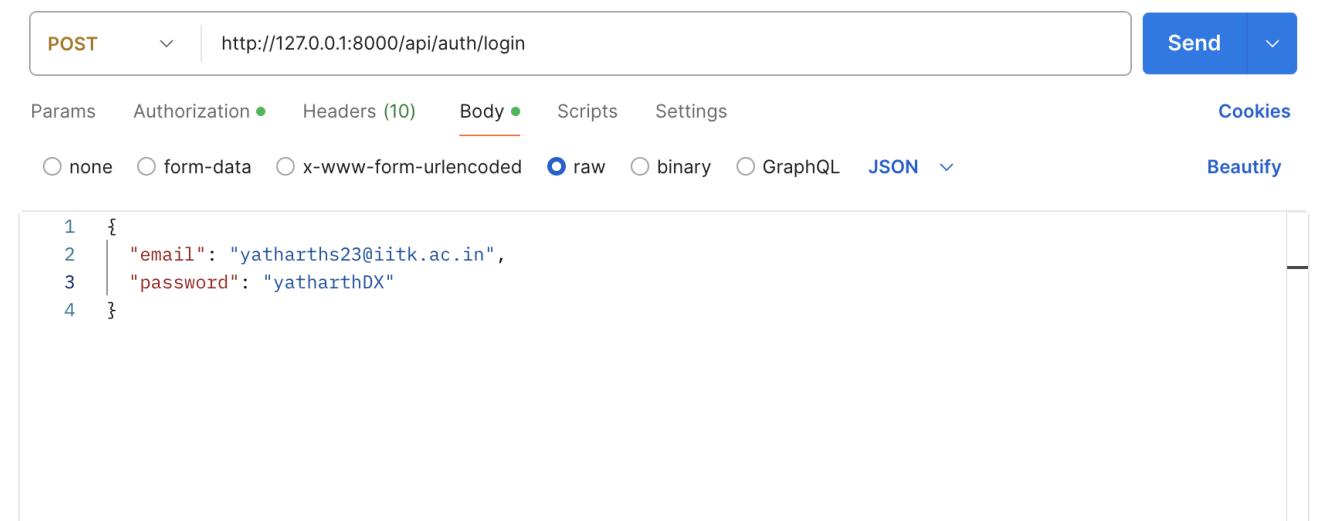
Test Date: [03/26/2025] - [03/26/2025]

Test Results:

Correct Username and Password - “Login Successful”

Incorrect Username or Password - “Invalid credentials”

Structural Coverage: Branch Coverage 100%



POST http://127.0.0.1:8000/api/auth/login

Params Authorization Headers (10) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

```
1  {
2    "email": "yatharth23@iitk.ac.in",
3    "password": "yatharthDX"
4 }
```

Body Cookies (1) Headers (5) Test Results

200 OK • 316 ms • 570 B •

{ } JSON Preview Visualize

```
1  {
2    "success": true,
3    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
4      eyJzdWIiOiJ5YXRoYXJ0aHMyM0BpaXRrLmFjLmluIiwicm9sZSI6InN0dWRlbnQiLCJleHAiOjE3NDM3Njc0ODV9.
5      ljsWdJeirwTq6Hvt4L5eUMh7_Ee36H_cqJ8rxkxW7Fg",
6    "token_type": "bearer"
7 }
```

The screenshot shows the Postman interface with a POST request to `http://127.0.0.1:8000/api/auth/login`. The request body is a JSON object:

```
1  {
2    "email": "yatharth23@iitk.ac.in",
3    "password": "notpass"
4 }
```

The response status is **401 Unauthorized**, with a duration of 718 ms and a size of 167 B. The response body is:

```
1  {
2    "detail": "Invalid credentials"
3 }
```

2. POST /register

The backend function for registering was tested. All the testable branches functioned as expected.

Unit Details: Backend function to handle Post Request “/register”

Test Owner: Yatharth Sharma

Test Date: [03/26/2025] - [03/26/2025]

Test Results:

Existing User - “User already exists”

New Username - “User registered successfully”

Structural Coverage: Branch Coverage 100%

The screenshot shows two separate API requests made using the Postman application.

Request 1 (Successful Registration):

- Method: POST
- URL: <http://127.0.0.1:8000/api/auth/register>
- Body (JSON):

```

1  {
2    "email": "namanyadav23@iitk.ac.in",
3    "password": "namanpass",
4    "name": "naman"
5  }

```

- Response Status: 200 OK
- Response Body (JSON):

```

1  {
2    "success": true,
3    "message": "User registered successfully"
4  }

```

Request 2 (Failed Registration - User Already Exists):

- Method: POST
- URL: <http://127.0.0.1:8000/api/auth/register>
- Body (JSON):

```

1  {
2    "email": "yatharth23@iitk.ac.in",
3    "password": "yatharthpass",
4    "name": "yatharth23"
5  }

```

- Response Status: 400 Bad Request
- Response Body (JSON):

```

1  {
2    "detail": "User already exists"
3  }

```

3. POST /send-otp

The backend function for send-otp was tested. All the testable branches functioned as

expected.

Unit Details: Backend function to handle Post Request “/send-otp”

Test Owner: Sayani Patra

Test Date: [03/26/2025] - [03/26/2025]

Test Results:

Request for registering, with already existing registered email - “Email is already registered”

Request for forgot password (reset password), with email not already registered email - “User not found”

Email credential of admin is wrong, failed to send opt - “Failed to send OTP”

Sent OTP for signup - “OTP sent successfully”

Sent OTP for forgot - “OTP sent successfully”

Structural Coverage: Branch Coverage 100%

If we try to send otp with incorrect admin privileges it leads to failure

The screenshot shows a POST request to `http://127.0.0.1:8000/api/auth/send-otp`. The request body is JSON:

```
1 {  
2   "email": "yatharth23@iitk.ac.in",  
3   "requestType": "forgot"  
4 }
```

The response status is 500 Internal Server Error with the message:

```
1 {  
2   "detail": "Failed to send OTP"  
3 }
```

The screenshot shows two separate API requests made using the Postman application.

Request 1:

- Method: POST
- URL: <http://127.0.0.1:8000/api/auth/send-otp>
- Body (raw JSON):

```

1  {
2    "email": "abcd@iitk.ac.in",
3    "requestType": "forgot"
4  }

```

Response 1:

- Status: 404 Not Found
- Time: 57 ms
- Size: 159 B
- Content: {"detail": "User not found"}

Request 2:

- Method: POST
- URL: <http://127.0.0.1:8000/api/auth/send-otp>
- Body (raw JSON):

```

1  {
2    "email": "yatharth23@iitk.ac.in",
3    "requestType": "signup"
4  }

```

Response 2:

- Status: 400 Bad Request
- Time: 146 ms
- Size: 174 B
- Content: {"detail": "Email is already registered"}

The screenshot shows two separate POST requests made to the endpoint `http://127.0.0.1:8000/api/auth/send-otp` using the Postman application.

Request 1 (Top):

- Method:** POST
- URL:** `http://127.0.0.1:8000/api/auth/send-otp`
- Body (JSON):**

```

1 {
2   "email": "yatharth23@iitk.ac.in",
3   "requestType": "forgot"
4 }
```
- Response:** 200 OK | 3.91 s | 175 B | `{ "success": true, "message": "OTP sent successfully" }`

Request 2 (Bottom):

- Method:** POST
- URL:** `http://127.0.0.1:8000/api/auth/send-otp`
- Body (JSON):**

```

1 {
2   "email": "namanyadav@iitk.ac.in",
3   "requestType": "signup"
4 }
```
- Response:** 200 OK | 5.17 s | 175 B | `{ "success": true, "message": "OTP sent successfully" }`

Both requests were successful, returning a 200 OK status with an OTP message sent successfully.

4. POST /verify-otp

The backend function for verify-otp was tested. All the testable branches functioned as expected.

Unit Details: Backend function to handle Post Request “/verify-otp”

Test Owner: Anirudh Singh

Test Date: [03/26/2025] - [03/26/2025]

Test Results:

Wrong OTP or submitting OTP after 10 minutes - “Invalid or expired OTP”

OTP verified - “OTP verified successfully”

Structural Coverage: Branch Coverage 100%

HTTP <http://127.0.0.1:8000/api/auth/verify-otp> Save Share

POST <http://127.0.0.1:8000/api/auth/verify-otp> Send

Params Authorization Headers (10) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL [JSON](#) [Beautify](#)

```
1 {  
2   "email": "yatharth23@iitk.ac.in",  
3   "otp": "469426",  
4   "requestType": "forgot"  
5 }
```

Body Cookies (1) Headers (4) Test Results [🕒](#) 400 Bad Request • 8 ms • 169 B • [🌐](#) [☰](#)

{ } [JSON](#) [Preview](#) [⚡](#) [Visualize](#) [🔗](#) [📄](#) [🔍](#) [🔗](#)

```
1 {  
2   "detail": "Invalid or expired OTP"  
3 }
```

The screenshot shows a Postman interface with the following details:

- Request URL:** http://127.0.0.1:8000/api/auth/verify-otp
- Method:** POST
- Body (JSON):**

```

1  {
2    "email": "yatharth23@iitk.ac.in",
3    "otp": "211562",
4    "requestType": "forgot"
5  }

```
- Response Status:** 200 OK
- Response Headers:** 7 ms, 179 B
- Response Body (JSON):**

```

1  {
2    "success": true,
3    "message": "OTP verified successfully"
4  }

```

5. POST /reset-password

The backend function for reset-password was tested. All the testable branches functioned as expected.

Unit Details: Backend function to handle Post Request “/reset-password”

Test Owner: Rohit Yadav

Test Date: [03/26/2025] - [03/26/2025]

Test Results:

Invalid User - “User not found”

Reset Password - “Password reset successfully”

Structural Coverage: Branch Coverage 100%

POST http://127.0.0.1:8000/api/auth/reset-password

Body (9) **Headers** (9) **Body** **Send**

```

1  {
2    "email": "rohity23@iitk.ac.in",
3    "password": "rohit@123"
4  }

```

Body **Cookies (1)** **Headers (4)** **Test Results** | **200 OK** • 591 ms • 181 B • **Beautify**

```

1  {
2    "success": true,
3    "message": "Password reset successfully"
4  }

```

HTTP **http://127.0.0.1:8000/api/auth/reset-password** **Save** **Share**

POST **http://127.0.0.1:8000/api/auth/reset-password** **Send**

Params **Authorization** **Headers (10)** **Body** **Scripts** **Settings** **Cookies**

none form-data x-www-form-urlencoded raw binary GraphQL **JSON** **Beautify**

```

1  {
2    "email": "yatharths24@iitk.ac.in",
3    "password": "yatharthnewpass"
4  }

```

Body **Cookies (1)** **Headers (4)** **Test Results** | **404 Not Found** • 67 ms • 159 B • **Beautify**

```

1  {
2    "detail": "User not found"
3  }

```

6. POST /upload-profile-picture

The backend function for uploading the profile picture of a user was tested. All the testable branches functioned as expected.

Unit Details: Backend function to handle Post Request “/upload-profile-picture”

Test Owner: Yatharth Sharma

Test Date: [03/26/2025] - [03/26/2025]

Test Results:

Uploaded correct format image - “Profile picture uploaded successfully”

Structural Coverage: Branch Coverage 100%

The screenshot shows a REST API testing interface with the following details:

- Method:** POST
- Endpoint:** /api/auth/upload-profile-picture/
- Request Body:**
 - Name:** file * required
 - Description:** string (\$binary)
 - Value:** Choose file: Santanusir_Head.jpg
- Responses:**
 - Curl:**

```
curl -X 'POST' \
  'http://127.0.0.1:8000/api/auth/upload-profile-picture/?username=counsellor2' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@Santanusr_Head.jpg;type=image/jpeg'
```
 - Request URL:** http://127.0.0.1:8000/api/auth/upload-profile-picture/?username=counsellor2
 - Server Response:**

Code	Details
200	Response body: <pre>{ "message": "Profile picture uploaded successfully" }</pre>

7. GET /get-profile-picture/{username}

The backend function to get the profile picture of a particular user was tested. All the testable branches functioned as expected.

Unit Details: Backend function to handle Get Request “/get-profile-picture/{username}”

Test Owner: Archita Goyal

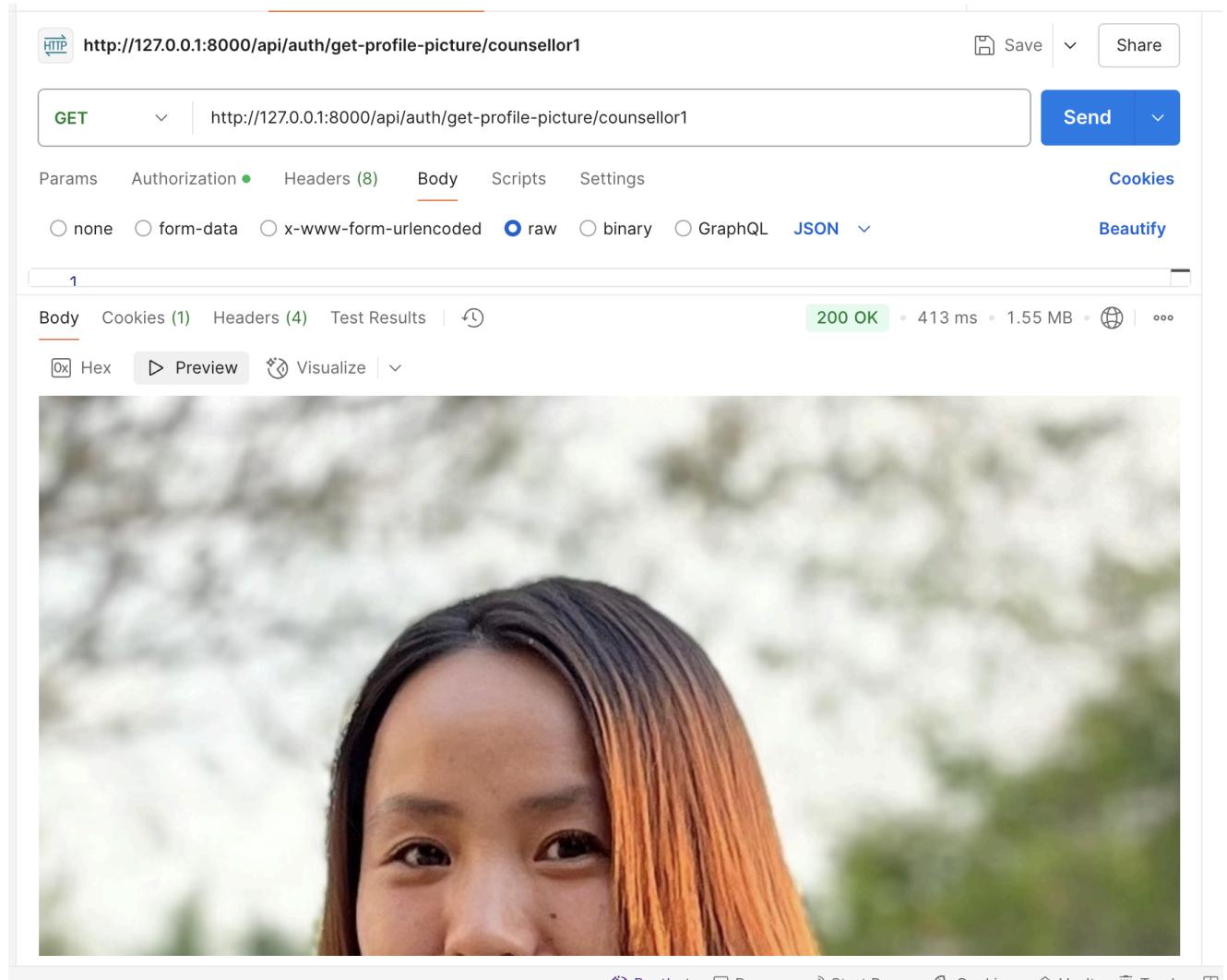
Test Date: [03/26/2025] - [03/26/2025]

Test Results:

No profile picture uploaded for the user - “No profile picture found”

Authorised user - “Fetched profile picture successfully”

Structural Coverage: Branch Coverage 100%



8. GET /getuserdetails

The backend function to get the details of the current user was tested. All the testable branches functioned as expected.

Unit Details: Backend function to handle Get Request “/get-user-details”

Test Owner: Archita Goyal

Test Date: [03/26/2025] - [03/26/2025]

Test Results:

Unauthorised User - "Unauthorised User"

Authorised User - "Fetched user details successfully"

Structural Coverage: Branch Coverage 100%

The screenshot shows the Postman application interface with two API requests.

Request 1 (Unauthenticated User):

- Method: GET
- URL: `http://127.0.0.1:8000/api/auth/getuserdetails`
- Body tab selected, showing raw JSON:

```
1 {  
2   |   "detail": "Not authenticated"  
3 }
```
- Response status: 401 Unauthorized
- Response time: 8 ms
- Response size: 191 B

Request 2 (Authenticated User):

- Method: GET
- URL: `http://127.0.0.1:8000/api/auth/getuserdetails`
- Body tab selected, showing raw JSON:

```
1 {  
2   |   "username": "archita",  
3   |   "profile_picture": null,  
4   |   "_id": "67dc44104145ca175f0b21b0",  
5   |   "role": "student",  
6   |   "avatar": "2"  
7 }
```
- Response status: 200 OK
- Response time: 394 ms
- Response size: 234 B

9. POST /logout

The backend function for logout was tested. All the testable branches functioned as expected.

Unit Details: Backend function to handle Post Request “/logout”

Test Owner: Om Chaudhari

Test Date: [03/26/2025] - [03/26/2025]

Test Results:

Cookies updated (JWT token removed) - “Logged out successfully”

Structural Coverage: Branch Coverage 100%

The screenshot shows a Postman request configuration and its response. The request method is POST, URL is <http://127.0.0.1:8000/api/auth/logout>, and the response status is 200 OK. The response body is a JSON object with two fields: "success" (true) and "message" ("Logged out successfully").

```

1 {
2   "success": true,
3   "message": "Logged out successfully"
4 }
    
```

10. POST /change-avatar

The backend function to change the avatar of the current user was tested. All the testable branches functioned as expected.

Unit Details: Backend function to handle Post Request “/change-avatar”

Test Owner: Yatharth Sharma

Test Date: [03/26/2025] - [03/26/2025]

Test Results:

No change - “User not found or avatar not changed”

Authorised user - “Avatar changes successfully”

Structural Coverage: Branch Coverage 100%

The screenshot shows a POST request to `http://127.0.0.1:8000/api/auth/change-avatar`. The request body is a JSON object with a single key `"avatar": "4"`. The response status is 200 OK, with a response time of 263 ms, a response size of 181 B, and a global icon. The response body is:

```

1  {
2   |   "success": true,
3   |   "message": "Avatar updated successfully"
4 }

```

11. GET /get-avatar

The backend function to get the avatar of a particular user was tested. All the testable branches functioned as expected.

Unit Details: Backend function to handle Get Request “/get-avatar”

Test Owner: Archita Goyal

Test Date: [03/26/2025] - [03/26/2025]

Test Results:

User whose avatar is accessed is not found - “User not found”

The screenshot shows a GET request to `http://127.0.0.1:8000/api/auth/get-avatar?id=67de66d742c41bf90788fa05`. The response status is 500 Internal Server Error, with a response time of 101 ms, a response size of 192 B, and a global icon. The response body is:

```

1 {
2   |   "detail": "Database error: 404: User not found"
3 }

```

Authorised user - “Fetched the avatar of the current user”

The screenshot shows a Postman request configuration. The URL is `http://127.0.0.1:8000/api/auth/get-avatar?id=67db0bfbc0e259833fb08f9`. The method is GET. The Headers tab is selected, showing a value of 7. The Body tab is selected, showing "raw" and "JSON". The response body is displayed as JSON:

```

1  {
2      "avatar": "3"
3  }

```

The response status is 200 OK, with a duration of 303 ms and a size of 139 B.

Structural Coverage: Branch Coverage 100%

12. POST /update-role

The backend function to change the role of a particular user was tested. All the testable branches functioned as expected.

Unit Details: Backend function to handle Post Request “/update-role”

Test Owner: Yatharth Sharma

Test Date: [03/26/2025] - [03/26/2025]

Test Results:

Unauthorised - “Unauthorised”

User not found - “User not found”

Admin request for change of role of a user to counsellor - “Role updated to counsellor”

Admin request for change of role of a user to student - “Role updated to student”

Structural Coverage: Branch Coverage 100%

POST /api/update-role Update Role

Parameters

Name Description

email * required
string
(query)
yatharths23@iitk.ac.in

new_role * required
string
(query)
student

admin_email * required
string
(query)
safespaceiitk@gmail.com

Execute Clear

Responses

Curl

```
curl -X 'POST' \
'http://127.0.0.1:8000/api/update-role?email=yatharths23%40iitk.ac.in&new_role=student&admin_email=safespaceiitk%40gmail.com' \
-H 'accept: application/json' \
-d ''
```

Request URL

```
http://127.0.0.1:8000/api/update-role?email=yatharths23%40iitk.ac.in&new_role=student&admin_email=safespaceiitk%40gmail.com
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "success": true, "message": "Role updated to student" }</pre> <p>Response headers</p> <pre>access-control-allow-credentials: true access-control-expose-headers: * content-length: 52 content-type: application/json date: Fri, 04 Apr 2025 11:30:49 GMT server: uvicorn</pre>

Responses

Code	Description	Links

POST /api/update-role Update Role

Parameters

Name Description

email * required
string
(query)
yatharths23@iitk.ac.in

new_role * required
string
(query)
counsellor

admin_email * required
string
(query)
safespaceiitk@gmail.com

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/api/update-role?email=yatharths23%40iitk.ac.in&new_role=counsellor&admin_email=safespaceiitk%40gmail.com' \
  -H 'accept: application/json' \
  -d ''
```

Request URL

```
http://127.0.0.1:8000/api/update-role?email=yatharths23%40iitk.ac.in&new_role=counsellor&admin_email=safespaceiitk%40gmail.com
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "success": true, "message": "Role updated to counsellor" }</pre> <p>Download</p> <p>Response headers</p> <pre>access-control-allow-credentials: true access-control-expose-headers: * content-length: 55 content-type: application/json date: Fri, 04 Apr 2025 11:28:53 GMT server: unicorn</pre>

POST /api/update-role Update Role

Parameters

Name Description

email * required
string
(query)
yatharths23@iitk.ac.in

new_role * required
string
(query)
counselor

admin_email * required
string
(query)
notsafeplaceiitk@gmail.com

Execute Clear

Responses

Curl

```
curl -X 'POST' \
'http://127.0.0.1:8000/api/update-role?email=yatharths23%40iitk.ac.in&new_role=counselor&admin_email=notsafeplaceiitk%40gmail.com' \
-H 'accept: application/json' \
-d ''
```

Request URL

```
http://127.0.0.1:8000/api/update-role?email=yatharths23%40iitk.ac.in&new_role=counselor&admin_email=notsafeplaceiitk%40gmail.com
```

Server response

Code	Details
403 <i>Undocumented</i>	Error: Forbidden Response body <pre>{ "detail": "Unauthorized" }</pre> Response headers <pre>access-control-allow-credentials: true access-control-expose-headers: * content-length: 25 content-type: application/json date: Fri, 04 Apr 2025 11:29:27 GMT server: uvicorn</pre>

Download

POST /api/update-role Update Role

Parameters

Name	Description
email * required string (query)	notyatharths23@iitk.ac.in
new_role * required string (query)	counselor
admin_email * required string (query)	safespacespaceiitk@gmail.com

Responses

Curl

```
curl -X 'POST' \
'http://127.0.0.1:8000/api/update-role?email=notyatharths23%40iitk.ac.in&new_role=counselor&admin_email=safespacespaceiitk%40gmail.com' \
-H 'accept: application/json' \
-d ''
```

Request URL

```
http://127.0.0.1:8000/api/update-role?email=notyatharths23%40iitk.ac.in&new_role=counselor&admin_email=safespacespaceiitk%40gmail.com
```

Server response

Code	Details
404 <i>Undocumented</i>	Error: Not Found Response body <pre>{ "detail": "User not found" }</pre>
	Response headers <pre>access-control-allow-credentials: true access-control-expose-headers: * content-length: 27 content-type: application/json date: Fri, 04 Apr 2025 11:30:09 GMT server: uvicorn</pre>

13. GET /counselors

The backend function to get the list of all the counsellors was tested. All the testable branches functioned as expected.

Unit Details: Backend function to handle Get Request "/counselors"

Test Owner: Sayani Patra

Test Date: [03/26/2025] - [03/26/2025]

Test Results:

Requested for the list of current counsellors - “Fetched the list of current counsellors”

Structural Coverage: Branch Coverage 100%

The screenshot shows a POSTMAN interface with the following details:

- Method:** GET
- URL:** http://127.0.0.1:8000/appointments/counselors
- Headers:** Headers (6) - Body (selected), Scripts, Settings
- Body Content Type:** raw
- Response Status:** 200 OK
- Response Time:** 440 ms
- Response Size:** 2.56 MB
- Response Data (Pretty JSON):**

```

1  [
2    {
3      "email": "counselor1@iitk.ac.in",
4      "name": "counselor1",
5      "profile_picture": "/9j/4AAQSkZJrgABAQEASABIAAD/6gP6QVJPVAAAAAAA+0AfAAC1PgAAX1AADJ+AABingAAw74AAJPeACu/
6        gAAHR8BAARAAQBWYQEA4IEBA0WiAQD1wwEAweUBAMgGAgAPKAIA7kgCAGJqAgAXjAIAmq0CA0TOAgBb8AIAKBEDAGoyAwCXUwMap3QDAFiVAwCItgMAkdcDAAP4AwCd
7        GAQAzkEAD9bBActfQa5Z8EAJbBBABg4wQAtguFADYnBQBiSQUAcwsFAjgMBQ91giuABTAFABzvBQ4EwYA2zQGAE1WBgDndwYAfpkGA1L268gDv2wYASf0GAPUDbwD
8        7PgcAamAHAPyBwAdpAcAvUMHABPnBwBGCAgAvioIAAxMCADPbAgAI44IAFuvcAdi0AgAAPMIAPMVCQa50AkAb1kJALp4CQCm1gKAiBQJADDRQAp7gka8gkkACU1cg
9        AfgAoAy1skAFp3CgD0kgoA+a0KATzICgBn5AoABQMLA1wjcwBSQwsATWMLAfDCwDLogsAfslAOxrcwDS/
10       wsAdh4MAKg9DAdwXAwAbn4MALmeDAAvgwAeN4MAB8ADQCLtg0AH0QNAFmDQDMiQ0AtqoNAF/KDQCP6g0AzwsoAGksDgDyTQ4AKHAOACWTdgC0tQ4ADtk0AIT
11       +DgAfJQ8AFE0PANB1DwCcnw8AFCp0PALt0dwAqIBAnEoQAK91EABqQRAYM4QAGL8EACCKREAs1URAKUCE0DCixEA8N0RA0EMEdDDoxIAJ2kJSAj1wEgC4wxAo/
12       ASAM8eEwAyThMAB38TA0mtEwCX3BMATgoUAGI2FADAZBQAgpMUADTCFAA28BQAKh8VAFNQFQA5fhUAOK4VAL7eFQCaDBYAKzcWAC1ifgDhjBYASigwADFifgAPDRcAy
13       DVXA61gfWaj1hcAPrUXANPhFwCtDxgAdjgYACpvGAATnxgApM8YAGN/GAD/LxKA/2AZAB+SGQAwBkAY00ZANcaGd+RxoAbxUaAHCigB4zRoAf/
14       caAD4gGwDXShsAC3YbAGOhGwDDzhnsAXfsbAEUnHACMUxwAioAcAPwsHADM2BwAqwQdAGoxHQByXx0A4o0dACG9HQc87R0A3B8eAFFTHgcnhh4AHiseAAfwHgDbIh8A2
15       FUfAAWJHwDpuw8Az+4fAf8hIAD7UyAAboggA0e9IACN8yAACyshAAhkIQAlnCEA69UhADIQIgCeSiIAM4Q1AAvAIgA6/
16       SIA7jkjADp3IwBmtCMAsPMjAA4wJADVa1oAHaYkAC3k3AbiTyUaE2Q1ACKnJ0BK7CUARDImAMh2JgBSuSYAc/
17       smAC47JwBDeicAozonA0v7JwAFP1gAhn8oAmvBKAABSKAAAD/
    
```

14. GET /counselors/available_slots

The backend function to get available slots for appointment of a particular counsellor for a particular date was tested. All the testable branches functioned as expected.

Unit Details: Backend function to handle Get Request “/counselors/available_slots”

Test Owner: Archita Goyal

Test Date: [03/26/2025] - [03/26/2025]

Test Results:

Requested for available slots of appointment of the counsellor for a particular date - time_slots (list) for that day

Structural Coverage: Branch Coverage 100%

The screenshot shows a REST API endpoint for 'Available Slots'. The URL is `/appointments/counselors/available_slots`. The method is `GET`. The description is 'Get Available Slots'.

Parameters:

Name	Description
<code>counselor_email</code> * required string (query)	<code>counsellor1@iitk.ac.in</code>
<code>date</code> * required string(\$date-time) (query)	<code>2025-03-22T00:00:00.000+00:00</code>

Buttons: Execute, Clear

Responses:

Curl:

```
curl -X 'GET' \
  'http://127.0.0.1:8000/appointments/counselors/available_slots?counselor_email=counsellor1%40iitk.ac.in&date=2025-03-22T00:00:00.000+00:00' \
  -H 'accept: application/json'
```

Request URL:

```
http://127.0.0.1:8000/appointments/counselors/available_slots?
counselor_email=counsellor1%40iitk.ac.in&date=2025-03-22T00:00:00.000+00:00
```

Server response:

Code	Details
200	Response body: <pre>["time_slots": ["10:30", "11:00"]]</pre> Download
	Response headers: <pre>content-length: 32 content-type: application/json date: Fri,04 Apr 2025 16:46:02 GMT server: uvicorn</pre>

Responses:

15. POST /appointments

The backend function to create a new appointment was tested. All the testable branches functioned as expected.

Unit Details: Backend function to handle Post Request “/appointments”

Test Owner: Archita Goyal

Test Date: [04/04/2025] - [04/04/2025]

Test Results:

User is not student - "Only Students can book appointments"

User is student and asked for unavailable time slot - "Selected time slot is not available"

User is student and asked for available time slot - “Booked the appointment successfully”

Structural Coverage: Branch Coverage 100%

HTTP <http://127.0.0.1:8000/appointments/appointments> Save Share

POST [http://127.0.0.1:8000/appointments/appointments](#) Send

Params Authorization Headers (9) **Body** Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {  
2     "user_name": "counselor2",  
3     "user_email": "counselor2@iitk.ac.in",  
4     "counselor_email": "counselor1@iitk.ac.in",  
5     "date": "2025-04-04T00:00:00.000+00:00",  
6     "time_slot": "15:00",  
7     "description": "I need to talk",  
8     "contact_no": "9039491300",  
9     "status": "pending"  
10 }
```

Body Cookies Headers (4) Test Results | ⚡

403 Forbidden • 5 ms • 180 B • 🌐 | ...

{ } JSON ▼ ▶ Preview ⟳ Visualize ⌄

1 ⌞ {
2 "detail": "Only students can book appointments"
3 }

The screenshot shows the Postman application interface. At the top, it displays the URL `http://127.0.0.1:8000/appointments/appointments`. Below the URL, there are buttons for 'Save' and 'Share'. The main area shows a 'POST' request being made to the same URL. The 'Body' tab is selected, showing the following JSON payload:

```

1  {
2    "user_name": "yatharth",
3    "user_email": "yatharths23@iitk.ac.in",
4    "counselor_email": "counselor1@iitk.ac.in",
5    "date": "2025-04-04T00:00:00.000+00:00",
6    "time_slot": "15:30",
7    "description": "I need to talk",
8    "contact_no": "9039491300",
9    "status": "pending"
10 }

```

Below the body, the response section shows a green '200 OK' status with a timestamp of '137 ms' and a size of '226 B'. The response body is also JSON:

```

1  {
2    "message": "Appointment request submitted successfully",
3    "appointment_id": "67f0101aabb46163646b5121"
4  }

```

16. PATCH /appointments/{appointment_id}

The backend function to update the status of an appointment was tested. All the testable branches functioned as expected.

Unit Details: Backend function to handle Patch Request “/appointments/{appointment_id}”

Test Owner: Yatharth Sharma

Test Date: [03/26/2025] - [03/26/2025]

Test Results:

Current user is not the counsellor whose appointment is taken - “You are not authorised to update this appointment”

Current user is counsellor and accepting/rejecting his/her appointment request - “Appointment accepted/rejected successfully”

Structural Coverage: Branch Coverage 100%

The screenshot shows a POSTMAN interface with the following details:

- Method:** PATCH
- URL:** http://127.0.0.1:8000/appointments/appointments/67f0101aabb46163646b5121
- Body (JSON):**

```

1  {
2   |   "status": "accepted"
3   }

```
- Response:** 403 Forbidden
 - Time: 54 ms
 - Size: 194 B

The screenshot shows a POSTMAN interface with the following details:

- Method:** PATCH
- URL:** http://127.0.0.1:8000/appointments/appointments/67f0101aabb46163646b5121
- Body (JSON):**

```

1  {
2   |   "status": "accepted"
3   }

```
- Response:** 200 OK
 - Time: 100 ms
 - Size: 172 B

17. GET /counselors/appointment_requests

The backend function to get appointment requests for counsellor was tested. All the testable branches functioned as expected.

Unit Details: Backend function to handle Get Request “/counselors/appointment_requests”

Test Owner: Yatharth Sharma

Test Date: [04/04/2025] - [04/04/2025]

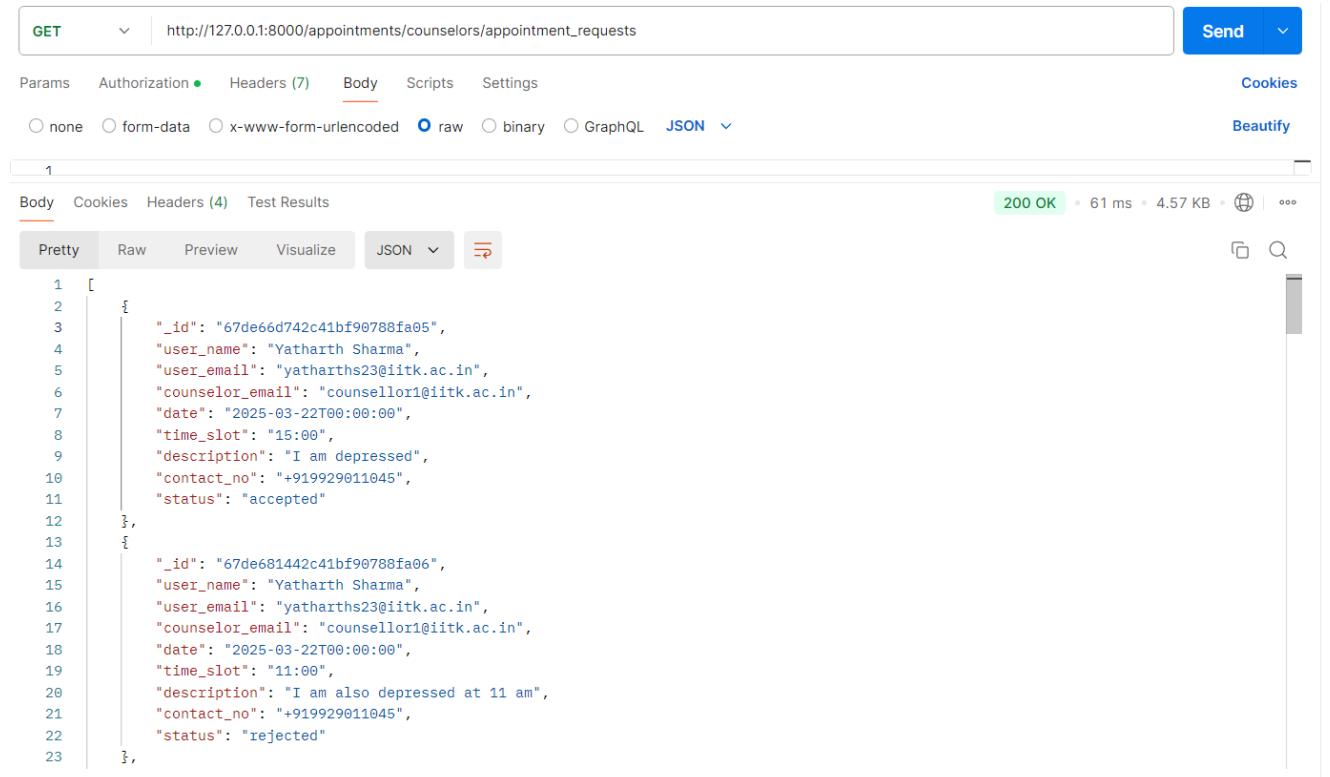
Test Results:

Current user is not a counsellor - “Only counsellor can view appointment requests”

Current user is counsellor and requested for the list of his/her appointment requests -

“Fetched the counsellor’s appointment requests”

Structural Coverage: Branch Coverage 100%



The screenshot shows the Postman interface with the following details:

- Method: GET
- URL: http://127.0.0.1:8000/appointments/counselors/appointment_requests
- Headers: Authorization (set to Bearer token), Content-Type (set to application/json)
- Body: Raw JSON response (shown below)
- Status: 200 OK
- Time: 61 ms
- Size: 4.57 KB

```
1 [
2   {
3     "_id": "67de66d742c41bf90788fa05",
4     "user_name": "Yatharth Sharma",
5     "user_email": "yatharths23@iitk.ac.in",
6     "counselor_email": "counselor1@iitk.ac.in",
7     "date": "2025-03-22T00:00:00",
8     "time_slot": "15:00",
9     "description": "I am depressed",
10    "contact_no": "+919929011045",
11    "status": "accepted"
12  },
13  {
14    "_id": "67de681442c41bf90788fa06",
15    "user_name": "Yatharth Sharma",
16    "user_email": "yatharths23@iitk.ac.in",
17    "counselor_email": "counselor1@iitk.ac.in",
18    "date": "2025-03-22T00:00:00",
19    "time_slot": "11:00",
20    "description": "I am also depressed at 11 am",
21    "contact_no": "+919929011045",
22    "status": "rejected"
23  }
```

The screenshot shows the POSTMAN interface with the following details:

- Request URL:** `http://127.0.0.1:8000/appointments/counselors/appointment_requests`
- Method:** GET
- Headers:** Authorization (set to Bearer [redacted]), Headers (7), Body (selected), Scripts, Settings
- Body Content:** Raw JSON response:


```

1  {
2    "detail": "Only counselors can view appointment requests"
3  }
```
- Status:** 403 Forbidden
- Response Time:** 4 ms
- Response Size:** 190 B

18. POST /counselors/update_slots

The backend function to update the available slots of appointments of counsellor was tested. All the testable branches functioned as expected.

Unit Details: Backend function to handle Post Request “/counselors/update_slots”

Test Owner: Archita Goyal

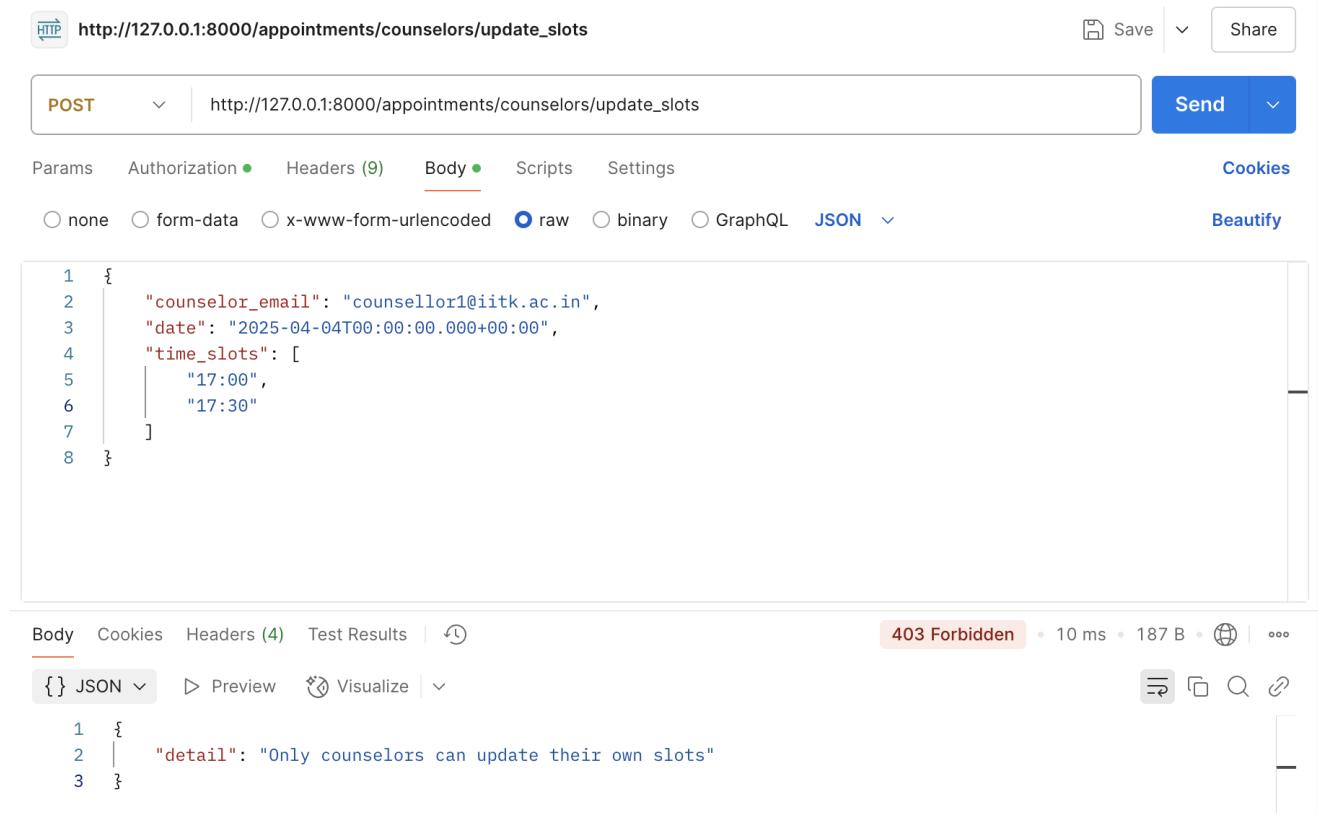
Test Date: [04/04/2025] - [04/04/2025]

Test Results:

User is not counsellor - “Only counsellor can update their own slots”

Date and list of time slots (of that date) to be made available - “Available slots updated”

Structural Coverage: Branch Coverage 100%



The screenshot shows a POST request to `http://127.0.0.1:8000/appointments/counselors/update_slots`. The request body is a JSON object:

```

1 {
2   "counselor_email": "counselor1@iitk.ac.in",
3   "date": "2025-04-04T00:00:00.000+00:00",
4   "time_slots": [
5     "17:00",
6     "17:30"
7   ]
8 }

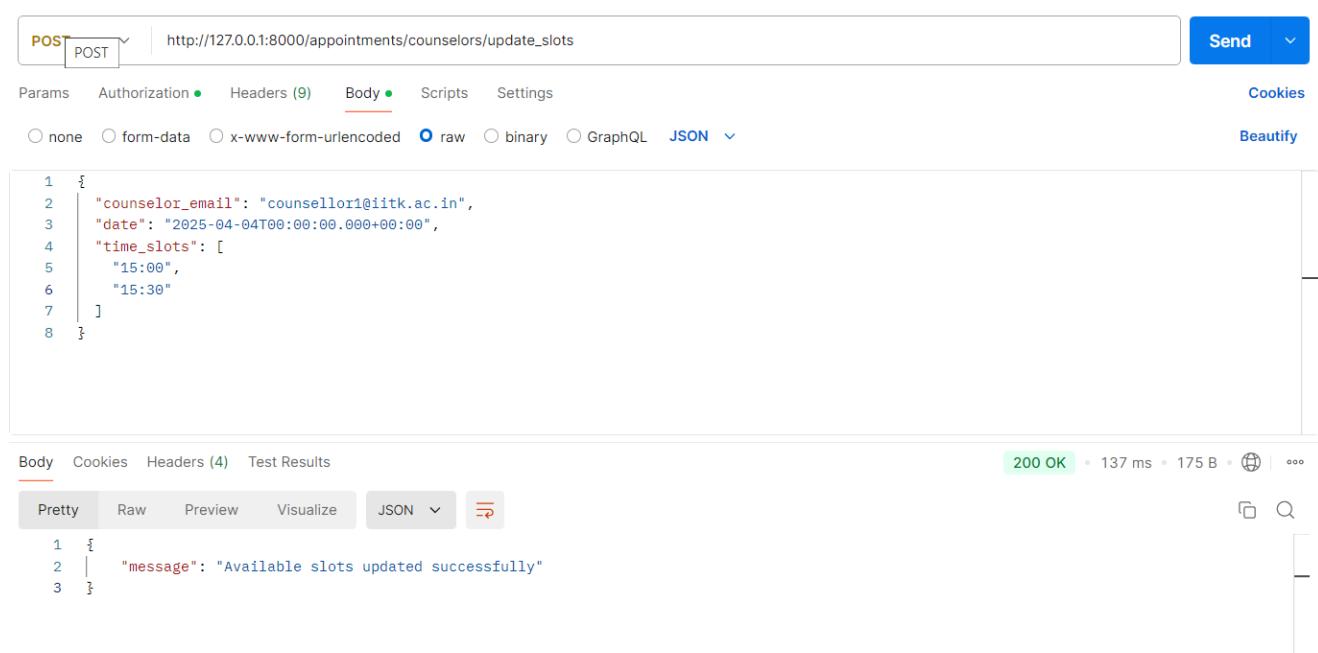
```

The response status is **403 Forbidden**, with a response time of 10 ms and a response size of 187 B. The response body is:

```

1 {
2   "detail": "Only counselors can update their own slots"
3 }

```

The screenshot shows a POST request to `http://127.0.0.1:8000/appointments/counselors/update_slots`. The request body is a JSON object:

```

1 {
2   "counselor_email": "counselor1@iitk.ac.in",
3   "date": "2025-04-04T00:00:00.000+00:00",
4   "time_slots": [
5     "15:00",
6     "15:30"
7   ]
8 }

```

The response status is **200 OK**, with a response time of 137 ms and a response size of 175 B. The response body is:

```

1 {
2   "message": "Available slots updated successfully"
3 }

```

19. GET /getappointments

The backend function to get the appointments of a student was tested. All the testable branches functioned as expected.

Unit Details: Backend function to handle Get Request "/getappointments"

Test Owner: Yatharth Sharma

Test Date: [03/26/2025] - [03/26/2025]

Test Results:

Current user is student and requested for his/her current appointments - Appointment details of that student

Current user is not student - "Only students can view their appointments"

Structural Coverage: Branch Coverage 100%

The screenshot shows a Postman API request configuration and its resulting response.

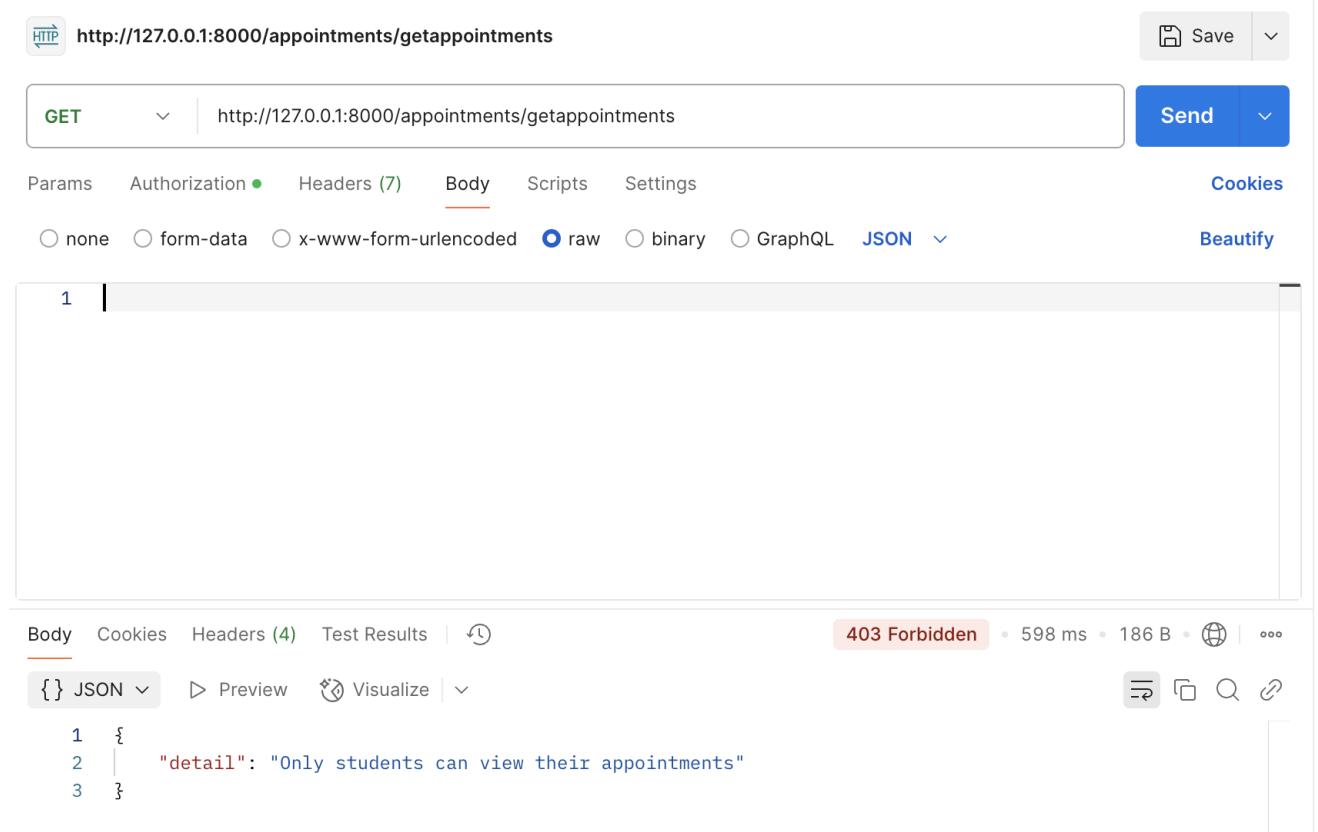
Request Configuration:

- Method: GET
- URL: `http://127.0.0.1:8000/appointments/getappointments`
- Headers: (7)
- Body: Raw JSON
- Params, Authorization, Scripts, Settings, Cookies, and Beautify buttons are also visible.

Response:

- Status: 200 OK
- Time: 86 ms
- Size: 1.73 KB
- Content-Type: application/json
- Raw JSON Response:

```
1  [
2    {
3      "_id": "67de66d742c41bf90788fa05",
4      "user_name": "Yatharth Sharma",
5      "user_email": "yatharthsh23@iitk.ac.in",
6      "counselor_email": "counselor1@iitk.ac.in",
7      "date": "2025-03-22T00:00:00",
8      "time_slot": "15:00",
9      "description": "I am depressed",
10     "contact_no": "+919929011045",
11     "status": "accepted"
12   }
```



20. GET /search_blog_search-get

The backend function to search for blogs with specific keywords/tags was tested. The functionality to present the user with the blogs with such keywords/tags was verified and all the testable branches performed as expected.

GET /search_blog/search-get/ Search Posts Get

Search posts based on text and tags (GET method).

- Returns posts where any input tag matches with the post's relevance tags
- Returns posts where input text (or part of it) matches with post title
- Posts with more tag matches are ranked higher

Parameters

Name	Description
text	Text to search in titles string (string null) (query)
tags	Comma-separated list of tags to search string (string null) (query)

Responses

Curl

```
curl -X 'GET' \
'http://127.0.0.1:8000/search_blog/search-get/?text=Stress' \
-H 'accept: application/json'
```

Request URL

http://127.0.0.1:8000/search_blog/search-get/?text=Stress

Server response

Code Details

200 Response body

```
[
  {
    "id": "67e04f5d63279d4df3120223",
    "title": "Socially depressed, academically stressed",
    "content": "what a perfect combo for a chaotic masterpiece! On one hand, my social life is as active as a broken WiFi router, and on the other, my academics are putting me through a stress test harsher than any NASA equipment. It's a beautiful cycle of avoiding people, drowning in deadlines, and questioning all my life choices at 2 AM. But hey, at least my sleep schedule is consistently nonexistent, and my caffeine intake is at an all-time high. Who needs balance when you can have a breakdown instead?",
    "relevant_tags": [
      "stress",
      "academic",
      "social",
      "work-life balance"
    ],
    "tag_match_count": 175
  },
  {
    "id": "67a056aa63279d4df3120225",
    "title": "Socially depressed, academically stressed , healthy relationship",
    "content": "Socially depressed, academically stressed, but at least my relationship is thriving-because nothing screams 'romance' like stress-induced breakdowns and existential crises! While my social life is as lively as a deserted island and my grades are playing a dangerous game of hide-and-seek, my partner gets the absolute privilege of dating a caffeine-fueled disaster. Honestly, at this point, I'm not sure if they love me or are just conducting a long-term psychological study. Either way, at least one aspect of my life hasn't completely collapsed yet.",
    "relevant_tags": [
      "stress",
      "academic",
      "social",
      "relationships",
      "health"
    ],
    "tag_match_count": 175
  }
]
```

[Download](#)

21. POST /blogs/blogs

The backend function to create a new blog post was tested. The functionality to handle blog creation, including input validation and database storage, was verified, and all testable branches performed as expected.

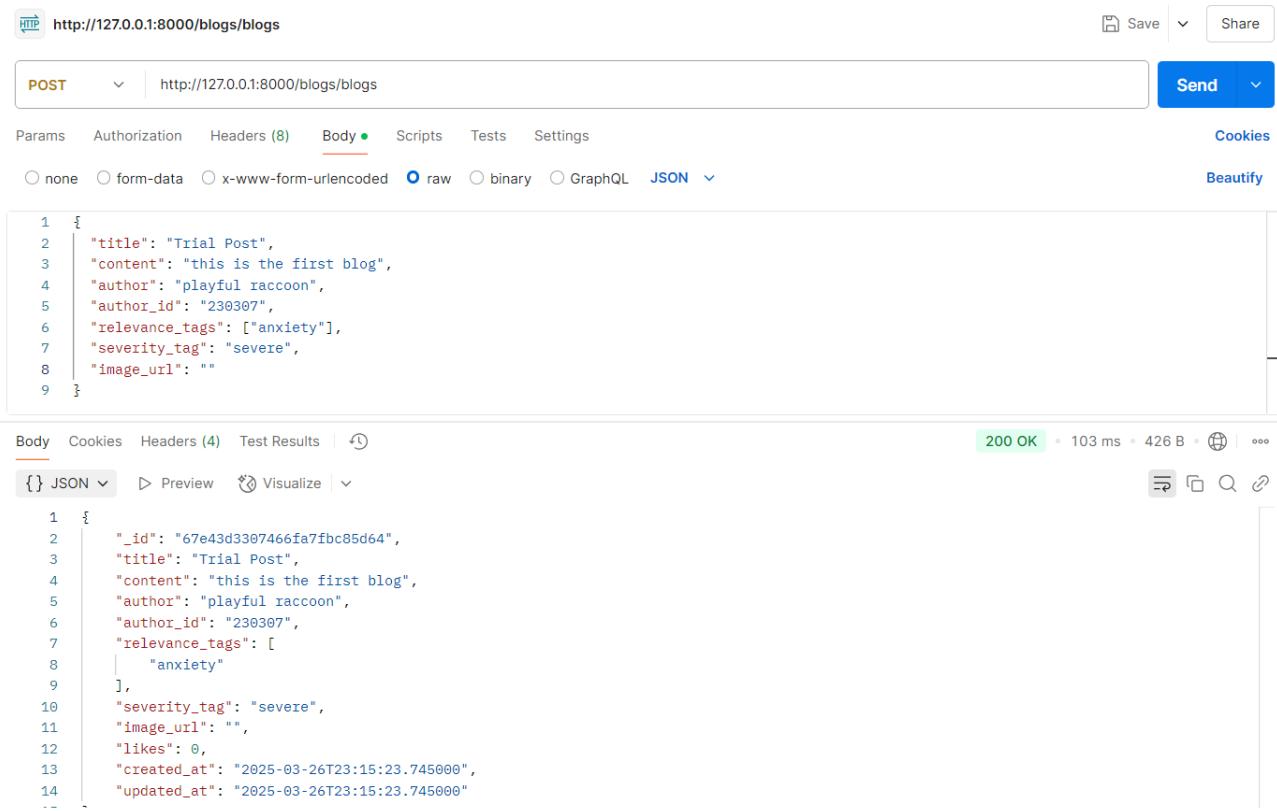
Unit Details: Backend function to handle POST Request "/blogs/blogs"

Test Owner: Vivek

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Blog title, content, and metadata provided - "Blog successfully created and stored in the database"

Structural Coverage: Branch Coverage 100%



```

1 {
2   "title": "Trial Post",
3   "content": "this is the first blog",
4   "author": "playful raccoon",
5   "author_id": "230307",
6   "relevance_tags": ["anxiety"],
7   "severity_tag": "severe",
8   "image_url": ""
9 }

```

Body Cookies Headers (4) Test Results | ⏱

{ } JSON ▾ ▷ Preview ⚡ Visualize | ⏱

```

1 {
2   "_id": "67e43d3307466fa7fbc85d64",
3   "title": "Trial Post",
4   "content": "this is the first blog",
5   "author": "playful raccoon",
6   "author_id": "230307",
7   "relevance_tags": [
8     "anxiety"
9   ],
10  "severity_tag": "severe",
11  "image_url": "",
12  "likes": 0,
13  "created_at": "2025-03-26T23:15:23.745000",
14  "updated_at": "2025-03-26T23:15:23.745000"
...

```

200 OK | 103 ms | 426 B | ⌂ | ⏱

22. GET /blogs/blogs

The backend function to retrieve all blogs was tested. The response containing a list of blogs was validated, and all testable branches functioned correctly.

Unit Details: Backend function to handle GET Request “/blogs/blogs”

Test Owner: Vivek

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Request made - “List of all blogs returned successfully”

Structural Coverage: Branch Coverage 100%

GET /blogs/blogs Get Blogs

Parameters

Name	Description
skip integer (query)	0
limit integer (query)	20

Responses

Curl

```
curl -X 'GET' \
'http://127.0.0.1:8000/blogs/blogs?skip=0&limit=20' \
-H 'accept: application/json'
```

Request URL

<http://127.0.0.1:8000/blogs/blogs?skip=0&limit=20>

Server response

Code	Details
200	Response body <pre>{ "relevance_tags": ["health", "self-care"], "severity_tag": "", "image": null, "likes": 4, "created_at": "2025-03-23T15:01:18.215000", "updated_at": "2025-03-27T01:28:08.953000" }, { "id": "67df56721cbd658c63aa1946", "title": "jbrembre", "content": "rjbj4v", "author": "spiderman", "author_id": "67df45f252d0edf35dbeb2f9", "relevance_tags": ["stress", "motivation"], "severity_tag": "", "image": null, "likes": 14, "created_at": "2025-03-23T06:01:46.478000", "updated_at": "2025-03-27T01:28:08.953000" }</pre>

23. POST /blogs/classify

The backend function to classify text in a blog was tested. The classification logic and output were verified, and all testable branches worked as expected.

Unit Details: Backend function to handle POST Request "/blogs/classify"

Test Owner: Vivek

Test Date: [04/04/2025] - [04/04/2025]

Test Results:

Text input provided - Text classified with appropriate label

Structural Coverage: Branch Coverage 100%

The screenshot shows the Postman application interface with two separate requests to the endpoint `/blogs/classify`.

Request 1 (Top):

- Method: POST
- URL: `http://127.0.0.1:8000/blogs/classify`
- Headers: (9)
- Body (raw JSON):

```
1 {
2   "text" : "He is stupid"
3 }
```

Response 1 (Top):

- Status: 200 OK
- Time: 164 ms
- Size: 130 B

Request 2 (Bottom):

- Method: POST
- URL: `http://127.0.0.1:8000/blogs/classify`
- Headers: (8)
- Body (raw JSON):

```
1 {
2   "text" : "Loneliness in a Crowd"
3 }
```

Response 2 (Bottom):

- Status: 200 OK
- Time: 1.01 s
- Size: 135 B

24. POST /blogs/classify-severity

The backend function to classify the severity of blog content was tested. The severity scoring logic was validated, and all testable branches performed correctly.

Unit Details: Backend function to handle POST Request “/blogs/classify-severity”

Test Owner: Vivek

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Content provided - “Severity score assigned successfully”

Structural Coverage: Branch Coverage 100%

POST http://127.0.0.1:8000/blogs/classify-severity

Params Authorization Headers (8) Body Scripts Settings Cookies Beautify

Body (Pretty) `1 {
2 "text" : "Burnout is Consuming Me"
3 }`

Headers (4) Test Results 200 OK 2.42 s 148 B

Body (Raw) `1 {
2 "severity": [
3 "severe"
4]
5 }`

POST http://127.0.0.1:8000/blogs/classify-severity

Params Authorization Headers (8) Body Scripts Settings Cookies Beautify

Body (Pretty) `1 {
2 "text" : "Loneliness in a Crowd"
3 }`

Headers (4) Test Results 200 OK 2.42 s 150 B

Body (Raw) `1 {
2 "severity": [
3 "moderate"
4]
5 }`

POST http://127.0.0.1:8000/blogs/classify-severity

Params Authorization Headers (8) Body Scripts Settings Cookies Beautify

Body (Pretty) `1 {
2 "text" : "Loneliness in a Crowd"
3 }`

Headers (4) Test Results 200 OK 2.42 s 150 B

Body (Raw) `1 {
2 "severity": [
3 "moderate"
4]
5 }`

25. GET /blogs/blogs/{blog_id}

The backend function to retrieve a specific blog by ID was tested. The response with blog details was verified, and all testable branches worked as expected.

Unit Details: Backend function to handle GET Request “/blogs/blogs/{blog_id}”

Test Owner: Vivek

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Valid blog ID provided - “Blog details returned successfully”

Structural Coverage: Branch Coverage 100%

GET /blogs/blogs/{blog_id} Get Blog

Parameters

Name	Description
blog_id * required	string (path) 67f00efele6ce5b4df2d1d77

Responses

Curl

```
curl -X 'GET' \
'http://127.0.0.1:8000/blogs/blogs/67f00efele6ce5b4df2d1d77' \
-H 'accept: application/json'
```

Request URL

http://127.0.0.1:8000/blogs/blogs/67f00efele6ce5b4df2d1d77

Server response

Code Details

200 Response body

```
{
  "id": "67f00efele6ce5b4df2d1d77",
  "title": "(",
  "content": "i am very sad please help me",
  "author": "omcc23",
  "author_id": "67e3e2e4ade920f9ad955cde",
  "relevance_tags": [
    "health"
  ],
  "severity_tag": "moderate",
  "image": null,
  "likes": 2,
  "created_at": "2025-04-04T16:55:26.263000",
  "updated_at": "2025-04-04T16:58:10.701000"
}
```

Download

26. PUT /blogs/blogs/{blog_id}

The backend function to update a specific blog by ID was tested. The update logic and database changes were validated, and all testable branches functioned correctly.

Unit Details: Backend function to handle PUT Request “/blogs/blogs/{blog_id}”

Test Owner: Vivek

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Updated blog data provided - “Blog updated successfully in the database”

Structural Coverage: Branch Coverage 100%

PUT /blogs/blogs/{blog_id} Update Blog

Parameters

Name	Description
blog_id <small>required</small>	string (path) 67f00efele6ce5b4df2d1d77

Request body required

application/json

```
{
  "title": "(",
  "content": "i am sad please help me, how to be happy",
  "author": "omcc23",
  "author_id": "67e3e2e4ade920f9ad955cde",
  "relevance_tags": [
    "health"
  ],
  "severity_tag": "moderate",
  "image": null
}
```

Responses

Curl

```
curl -X 'PUT' \
'http://127.0.0.1:8000/blogs/blogs/67f00efele6ce5b4df2d1d77' \
-H 'Accept: application/json' \
-H 'Content-type: application/json' \
-d '{
  "title": "(",
  "content": "i am sad please help me, how to be happy",
  "author": "omcc23",
  "author_id": "67e3e2e4ade920f9ad955cde",
  "relevance_tags": [
    "health"
  ],
  "severity_tag": "moderate",
  "image": null
}'
```

Request URL

http://127.0.0.1:8000/blogs/blogs/67f00efele6ce5b4df2d1d77

Server response

Code	Details
200	Response body <pre>{ "_id": "67f00efele6ce5b4df2d1d77", "title": "(", "content": "i am sad please help me, how to be happy", "author": "omcc23", "author_id": "67e3e2e4ade920f9ad955cde", "relevance_tags": ["health"], "severity_tag": "moderate", "image": null, "likes": 0, "created_at": "2025-04-04T16:55:26.263000", "updated_at": "2025-04-05T15:29:33.155000" }</pre>

Download

27. DELETE /blogs/blogs/

The backend function to delete a specific blog by ID was tested. The deletion logic and database cleanup were verified, and all testable branches worked as expected.

Unit Details: Backend function to handle DELETE Request “/blogs/blogs/{blog_id}”

Test Owner: Vivek

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Valid blog ID provided - “Blog deleted successfully from the database” else

“Blog not found”

Structural Coverage: Branch Coverage 100%

The screenshot displays two API endpoints for managing blogs.

DELETE /blogs/blogs/{blog_id}

Parameters

Name	Description
blog_id * required	string (path) 67f00efe1e6ce5b4df2d1d77

Responses

Curl

```
curl -X 'DELETE' \
'http://127.0.0.1:8000/blogs/blogs/67f00efe1e6ce5b4df2d1d77' \
-H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/blogs/blogs/67f00efe1e6ce5b4df2d1d77
```

Server response

Code	Details
200	Response body

```
{
  "message": "Blog deleted successfully"
}
```

GET /blogs/blogs/{blog_id}

Parameters

Name	Description
blog_id * required	string (path) 67f00efe1e6ce5b4df2d1d77

Responses

Curl

```
curl -X 'GET' \
'http://127.0.0.1:8000/blogs/blogs/67f00efe1e6ce5b4df2d1d77' \
-H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/blogs/blogs/67f00efe1e6ce5b4df2d1d77
```

Server response

Code	Details
404	Error: Not Found <i>Undocumented</i>
	Response body

```
{
  "detail": "Blog not found"
}
```

28. POST /blogs/blogs/{blog_id}/upload-image

The backend function to upload an image for a specific blog was tested. Image handling and storage were validated, and all testable branches performed correctly.

Unit Details: Backend function to handle POST Request “/blogs/blogs/{blog_id}/upload-image”

Test Owner: Vivek

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Image file provided - “Image uploaded and linked to blog successfully”

Structural Coverage: Branch Coverage 100%

The screenshot shows a REST API testing interface with the following details:

- Method:** POST
- Endpoint:** /blogs/blogs/{blog_id}/upload-image
- Parameters:**
 - blog_id** (required, string, path): 67e04f5d63279d4df3120223
- Request body:** multipart/form-data
- File:** file (required, string(\$binary)): spidy_wallpaper.jpg
- Buttons:** Execute, Clear
- Responses:**
 - Curl:** curl -X 'POST' \ 'http://127.0.0.1:8000/blogs/blogs/67e04f5d63279d4df3120223/upload-image' \ -H 'accept: application/json' \ -H 'Content-Type: multipart/form-data' \ -F 'file=@spidy_wallpaper.jpg;type=image/jpeg'
 - Request URL:** http://127.0.0.1:8000/blogs/blogs/67e04f5d63279d4df3120223/upload-image
 - Server response:**
 - Code:** 200
 - Details:** Response body
 - JSON Response:** {"message": "Image uploaded successfully", "image_url": "/uploads/90595e17-c696-4f71-80c3-5c46d2ad9e09.jpg"}
 - Buttons:** Copy, Download
 - Response headers:**

29. GET /blogs/blogs/{blog_id}/comments

The backend function to retrieve comments for a specific blog was tested. The response with comment data was verified, and all testable branches worked as expected.

Unit Details: Backend function to handle GET Request “/blogs/blogs/{blog_id}/comments”

Test Owner: Vivek

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Valid blog ID provided - “List of comments returned successfully”

Structural Coverage: Branch Coverage 100%

The screenshot shows a REST API testing interface with the following details:

- Method:** GET
- Endpoint:** /blogs/blogs/{blog_id}/comments
- Description:** Get Comments
- Parameters:**
 - blog_id** (required, string, path): 67e04f5d63279d4df3120223
- Buttons:** Execute (highlighted in blue), Clear
- Responses:**
 - Curl:**

```
curl -X 'GET' \
  'http://127.0.0.1:8000/blogs/blogs/67e04f5d63279d4df3120223/comments' \
  -H 'accept: application/json'
```
 - Request URL:** http://127.0.0.1:8000/blogs/blogs/67e04f5d63279d4df3120223/comments
 - Server response:**

Code	Details
200	Response body <pre>[{ "_id": "67f013f83200f1e4273c86ad", "blog_id": "67e04f5d63279d4df3120223", "content": "testing", "author": "omcc23", "author_id": "67e3e2e4ade920f9ad955cde", "created_at": "2025-04-04T17:16:40.614000" }, { "_id": "67e059a9683c43c47e62db40", "blog_id": "67e04f5d63279d4df3120223", "content": "testing", "author": "archita", "author_id": "67dc44104145ca175f0b21b0", "created_at": "2025-03-24T00:27:45.079000" }, { "_id": "67e05238a04ca3600647986a", "blog_id": "67e04f5d63279d4df3120223", "content": "bsdk", "author": "anirudh", "author_id": "67db1278e480b17b41bd33aa", "created_at": "2025-03-23T23:56:00.008000" }, { "_id": "67e05236a04ca36006479869", "blog_id": "67e04f5d63279d4df3120223", "content": "comment 4", "author": "anirudh", "author_id": "67db1278e480b17b41bd33aa", "created_at": "2025-03-23T23:56:00.008000" }]</pre>
- Download:** A button to download the response body.

30. POST /blogs/blogs/{blog_id}/comments

The backend function to create a comment for a specific blog was tested. Comment creation and storage were validated, and all testable branches functioned correctly.

Unit Details: Backend function to handle POST Request "/blogs/blogs/{blog_id}/comments"

Test Owner: Vivek

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Comment text provided - "Comment created and linked to blog successfully"

Structural Coverage: Branch Coverage 100%

The screenshot shows a REST API testing interface. At the top, it says "POST /blogs/blogs/{blog_id}/comments Create Comment". Below this, there are sections for "Parameters" and "Request body". In the "Parameters" section, there is a field for "blog_id" with the value "67e04f5d63279d4df3120223". In the "Request body" section, the content type is set to "application/json", and the JSON payload is:

```
{
  "content": "new comment for testing",
  "author": "omcc23",
  "author_id": "67e3e2e4ade920f9ad955cde"
}
```

At the bottom, there are "Execute" and "Clear" buttons. The "Responses" section shows a "curl" command and a "Request URL" of "http://127.0.0.1:8000/blogs/blogs/67e04f5d63279d4df3120223/comments". The "Server response" section shows a 200 status code with a JSON response body:

```
{
  "id": "67f106eb06715abf1d29074f",
  "blog_id": "67e04f5d63279d4df3120223",
  "content": "new comment for testing",
  "author": "omcc23",
  "author_id": "67e3e2e4ade920f9ad955cde",
  "created_at": "2025-04-05T16:03:15.272000"
}
```

31.PUT /blogs/blogs/{blog_id}/likes

The backend function to update likes for a specific blog was tested. The like count adjustment

logic was verified, and all testable branches worked as expected.

Unit Details: Backend function to handle PUT Request “/blogs/blogs/{blog_id}/likes”

Test Owner: Vivek

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Like data provided - “Likes updated successfully”

Structural Coverage: Branch Coverage 100%

The screenshot shows a REST API testing interface with the following details:

- Parameters:**
 - blog_id** (required, string, path): Value: 67e04f5d63279d4df3120223
- Request body** (required): Value:

```
{ "likes": 10 }
```
- Content-Type**: application/json
- Execute** button
- Responses** section:
 - Curl** command:

```
curl -X 'PUT' \
'http://127.0.0.1:8000/blogs/blogs/67e04f5d63279d4df3120223/Likes' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{' \
'"likes": 10
}'
```
 - Request URL**: http://127.0.0.1:8000/blogs/blogs/67e04f5d63279d4df3120223/likes
 - Server response** (Code 200):
 - Response body**:

```
{
  "message": "Likes updated successfully",
  "likes": 10
}
```
 - Download** button

32. POST /blogs/blogs/{blog_id}/like

The backend function to like a specific blog was tested. The like action and database update were validated, and all testable branches performed correctly.

Unit Details: Backend function to handle POST Request “/blogs/blogs/{blog_id}/like”

Test Owner: Vivek

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Like request made - “Blog liked successfully by user”

Structural Coverage: Branch Coverage 100%

The screenshot shows the Postman application interface. In the top navigation bar, 'New Request - My Workspace' is selected. The left sidebar shows 'My Workspace' with 'Collections' and 'Environments'. Under 'Collections', there is a 'New Collection' entry. The main workspace shows a 'New Request' with a 'POST' method and the URL 'http://localhost:8000/blogs/blogs/67e04a7a3ff8b5d4508650f2/like'. The 'Body' tab is selected, showing a JSON payload with one key: 'user_id': '67df45f252d0edf35dbebe2f9'. Below the request, the response is displayed: a 200 OK status with a response time of 169 ms and a body size of 172 B. The response body is a JSON object with a message: "Blog liked successfully" and a likes count of 2.

33. POST /blogs/blogs/{blog_id}/unlike

The backend function, unlike a specific blog, was tested. The unlike action and database update were verified, and all testable branches worked as expected.

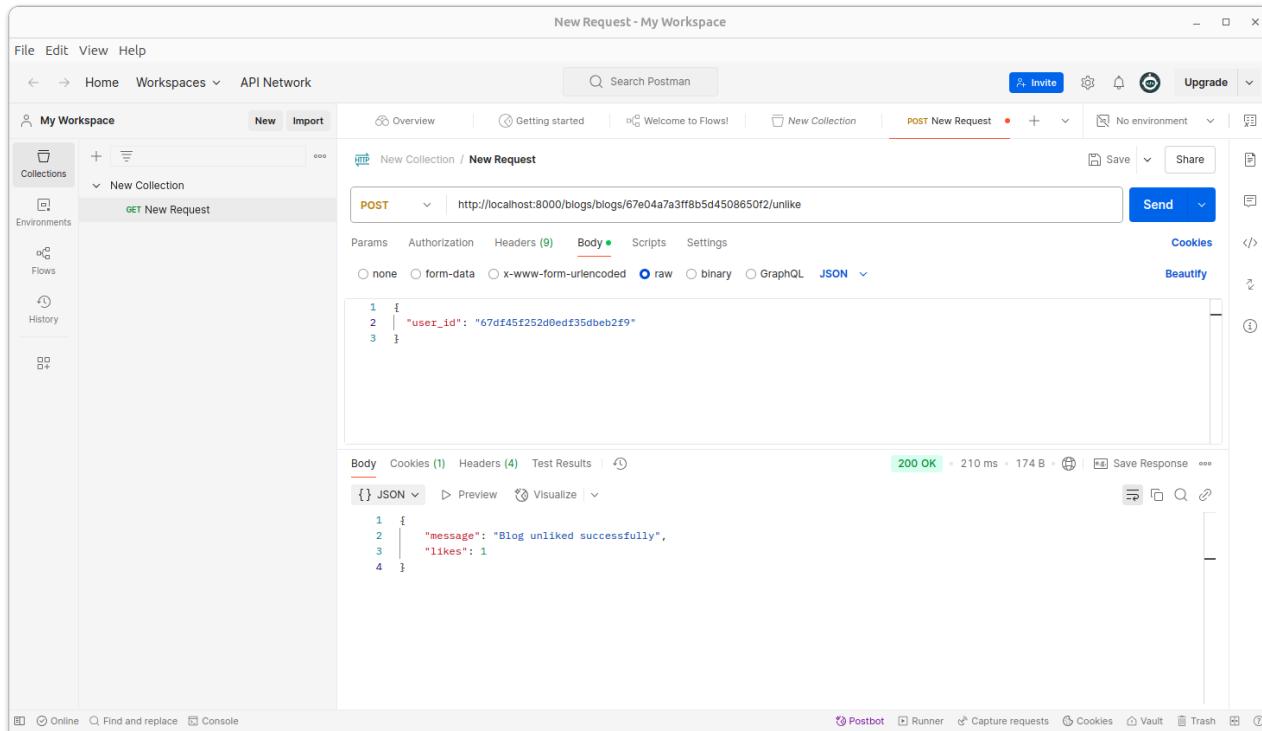
Unit Details: Backend function to handle POST Request “/blogs/blogs/{blog_id}/unlike”

Test Owner: Vivek

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Unlike request made - “Blog unliked successfully by user”

Structural Coverage: Branch Coverage 100%



34. GET /blogs/user/liked-posts

The backend function to retrieve a user's liked posts was tested. The response with liked blog data was validated, and all testable branches functioned correctly.

Unit Details: Backend function to handle GET Request “/blogs/user/liked-posts”

Test Owner: Vivek

Test Date: [04/04/2025] - [04/04/2025]

Test Results: User ID provided - “List of liked posts returned successfully”

Structural Coverage: Branch Coverage 100%

The screenshot shows a REST API endpoint for retrieving liked posts for a specific user. The endpoint is `GET /blogs/user/liked-posts` with the description "Get User Liked Posts". The "Parameters" section includes a required parameter `user_id` of type string (query) with the value `67e3e2e4ade920f9ad955cde`. Below the parameters are "Execute" and "Clear" buttons. The "Responses" section shows a curl command to execute the request, the request URL (`http://127.0.0.1:8000/blogs/user/liked-posts?user_id=67e3e2e4ade920f9ad955cde`), and the server response code 200 with the response body containing the user ID `"67e04f5d63279d4df3120223"`. There are "Copy" and "Download" buttons for the response body.

35. GET /blogs/user/{user_id}/blogs

The backend function to retrieve blogs by a specific user was tested. The response with user blog data was verified, and all testable branches worked as expected.

Unit Details: Backend function to handle GET Request “/blogs/user/{user_id}/blogs”

Test Owner: Vivek

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Valid user ID provided - “List of user blogs returned successfully”

Structural Coverage: Branch Coverage 100%

The screenshot shows a REST API documentation interface. At the top, it displays a GET request for the endpoint `/blogs/user/{user_id}/blogs`, labeled "Get User Blogs". Below this, there's a "Parameters" section with one required parameter: `user_id` (string, path), with the value `67db0bfbc0e259833fb08f9`. There are "Execute" and "Clear" buttons. The "Responses" section includes a "curl" command and a "Request URL" (both pointing to `http://127.0.0.1:8000/blogs/user/67db0bfbc0e259833fb08f9/blogs`). Under "Server response", a 200 status code is shown with a "Response body" containing a JSON array of blog posts. One post is displayed in full:

```
[{"id": "67e3def532e32957915dc0ca", "title": "Burnout is Consuming Me", "content": "I used to love learning, but now, I barely recognize myself. No matter how hard I work, it never feels like enough. My days blur into one another--endless assignments, back-to-back deadlines, and the crushing weight of expectations. I run on caffeine and sheer willpower, yet exhaustion clings to me like a shadow. Sleep feels like a luxury, and even when I do sleep, my mind doesn't rest. I'm constantly anxious about falling behind, about disappointing my family, about a future that feels more uncertain with each passing day. I know I'm burning out, but stopping isn't an option--because in college, slowing down feels like failing.", "author": "yatharth23", "author_id": "67db0bfbc0e259833fb08f9", "relevance_tags": ["health", "career", "academic"], "severity_tag": "severe", "image": null, "likes": 1, "created_at": "2025-03-26T16:33:17.162000", "updated_at": "2025-03-27T23:55:56.645000"}]
```

At the bottom right of the "Response body" area are "Copy" and "Download" buttons.

36. POST /blogs/report-blog

The backend function to report a blog was tested. The reporting logic and storage were validated, and all testable branches performed correctly.

Unit Details: Backend function to handle POST Request "/blogs/report-blog"

Test Owner: Vivek

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Report details provided - "Blog reported successfully"

Structural Coverage: Branch Coverage 100%

The screenshot shows a detailed view of a POST API endpoint for reporting a blog. The endpoint is `/blogs/report-blog`. The description is "Report Blog". The parameters section indicates "No parameters". The request body is required and is defined as application/json. The JSON payload is as follows:

```
{
  "blog_id": "67e3def532e32957915dc0ca",
  "reported_author_id": "yatharths23",
  "reported_author_name": "Yatharth Sharma",
  "reporter_email": "yatharths23@iitk.ac.in",
  "reason": "report function testing",
  "description": "testing"
}
```

At the bottom, there are "Execute" and "Clear" buttons. Below the main section, under "Responses", there is a "Curl" section containing the following command:

```
curl -X 'POST' \
  'http://127.0.0.1:8000/blogs/report-blog' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "blog_id": "67e3def532e32957915dc0ca",
    "reported_author_id": "yatharths23",
    "reported_author_name": "Yatharth Sharma",
    "reporter_email": "yatharths23@iitk.ac.in",
    "reason": "report function testing",
    "description": "testing"
}'
```

37. GET /api/messages/{user_id}

The backend function to fetch all the messages sent and received by a particular user was tested

Unit Details: Backend function to handle GET Request “/api/messages/{user_id}”

Test Owner: Anirudh Singh

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Report details provided - “” (details are not provided, rather only the messages are returned)

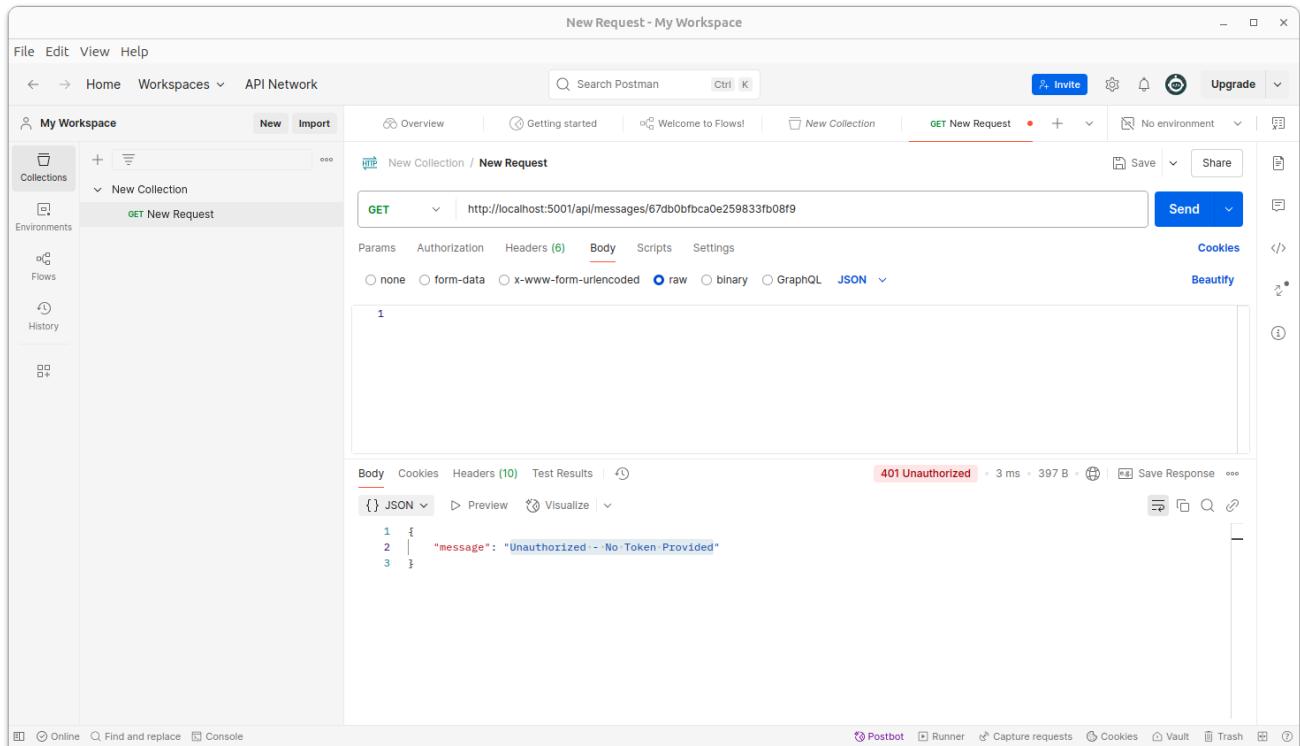
User not authorised - “Unauthorized - No Token Provided”

Structural Coverage: Branch Coverage 100%

The screenshot shows the Postman application interface. The top navigation bar includes File, Edit, View, Help, Home, Workspaces, API Network, and a search bar. The left sidebar has sections for Collections, Environments, Flows, and History. The main workspace shows a 'New Collection / New Request' tab with a GET method selected and the URL `http://localhost:5001/api/messages/67db0bfca0e259833fb08f9`. The 'Body' tab is active, displaying a JSON response with the following content:

```
1 [  
2   {  
3     "_id": "67e1b720690b4f7012ba2271",  
4     "senderId": "67db0bfca0e259833fb08f9",  
5     "receiverId": "67ddc4ca74145ca175f0b21b1",  
6     "text": "Helloo sir ",  
7     "status": "read",  
8     "createdAt": "2025-03-24T19:48:48.774Z",  
9     "updatedAt": "2025-03-24T19:50:15.209Z",  
10    "_v": 0  
11  },  
12 ]
```

The status bar at the bottom indicates a 200 OK response with 381 ms latency and 2.17 KB size.



38. GET /api/messages/users

The backend function to fetch all the users and their details except their passwords was tested. This function is used to fetch and display all the relevant users in the chat interface.

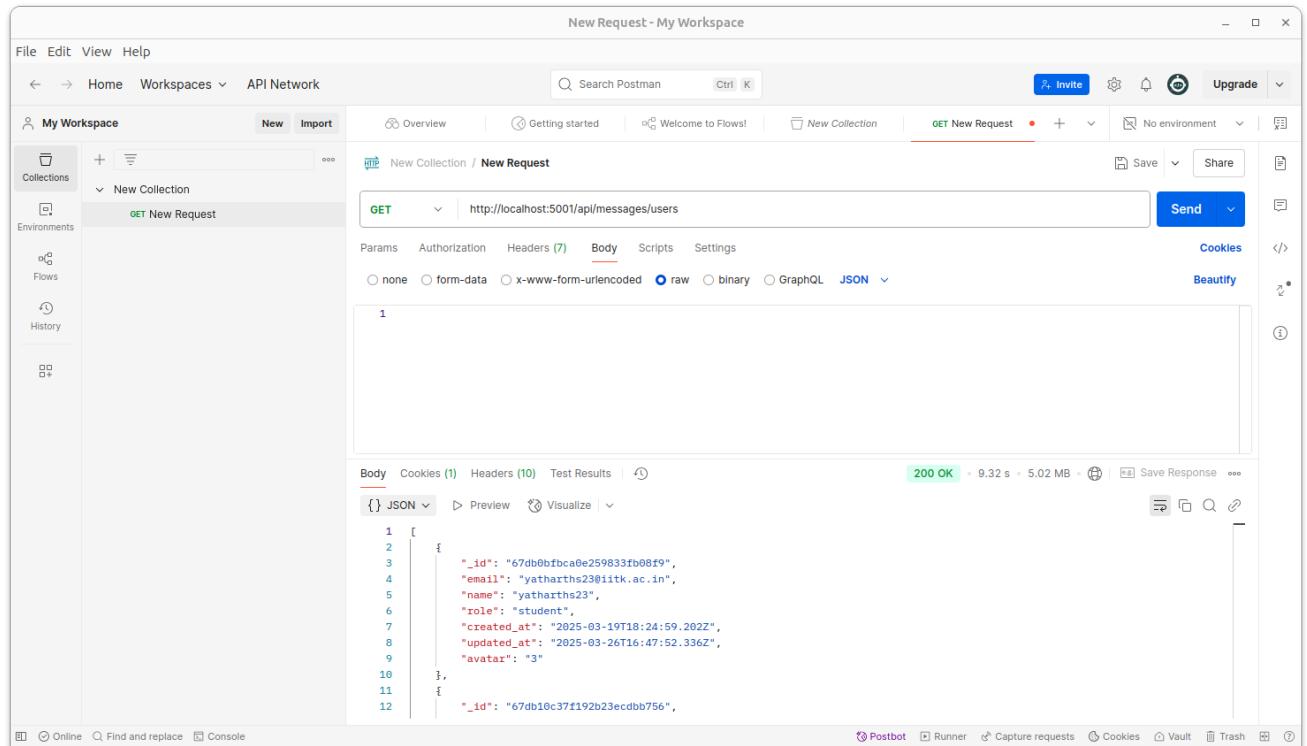
Unit Details: Backend function to handle GET Request “/api/messages/users”

Test Owner: Archita Goyal

Test Date: [04/04/2025] - [04/04/2025]

Test Results: An array of objects of users is returned.

Structural Coverage: Branch Coverage 100%



39. POST /api/messages/send/{receiver_id}

The backend function to send message to a particular user by their id

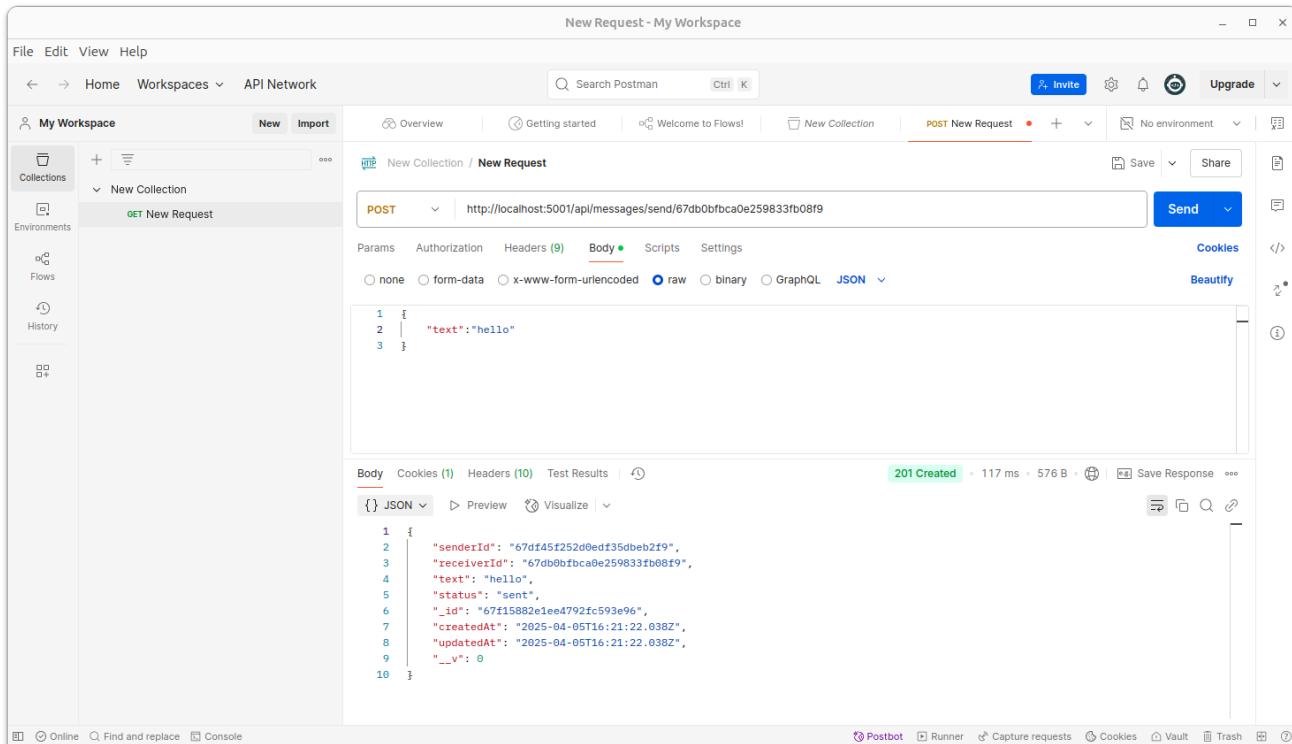
Unit Details: Backend function to handle POST Request “/api/messages/send/{receiver_id}”

Test Owner: Yatharth Sharma

Test Date: [04/04/2025] - [04/04/2025]

Test Results: A copy of the sent message object is returned

Structural Coverage: Branch Coverage 100%



40. PUT /api/messages/read/{senders_id}

The backend function to mark messages sent by a sender as read by the current user.

Unit Details: Backend function to handle PUT Request "/api/messages/read/{senders_id}"

Test Owner: Om Chaudhary

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Positive status response- "Message marked as read"

Structural Coverage: Branch Coverage 100%

The screenshot shows the Postman application interface. The top navigation bar includes File, Edit, View, Help, Home, Workspaces, API Network, and a search bar. The main toolbar features Invite, Settings, Notifications, Upgrade, and a collection of icons. The left sidebar displays 'My Workspace' with sections for Collections, Environments, Flows, and History. A 'New Collection' is currently selected. The central workspace shows a 'New Request' tab with a PUT method and URL `http://localhost:5001/api/messages/read/67db0bfbc0e259833fb08f9`. The 'Body' tab is active, showing a JSON response with one item:

```
1 {  
2   |   "message": "Messages marked as read"  
3 }
```

. The status bar at the bottom indicates a 200 OK response, 1.35 seconds duration, 378 B size, and various tool buttons.

3 Integration Testing

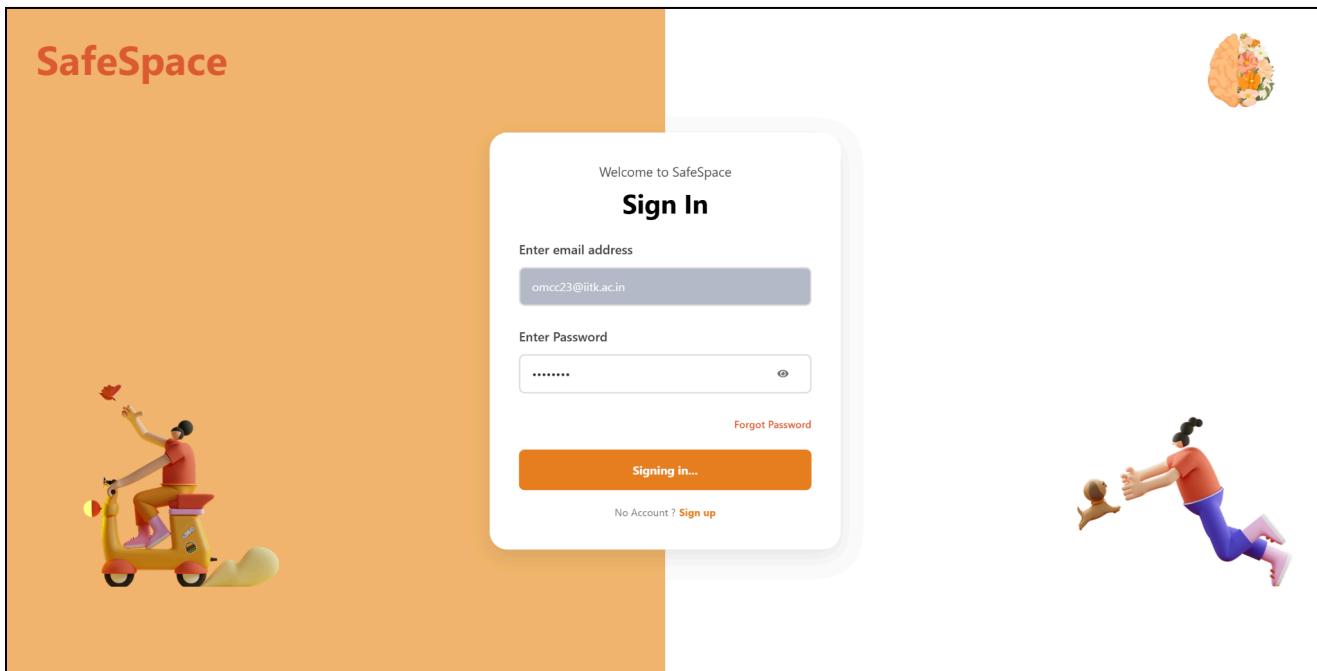
1. Checking the interface between login and Home Page for User

Module Details: login

Test Owner: Sayani Patra

Test Date: [04/02/2025] - [04/02/2025]

Test Results: When entering the correct user login credentials, the software redirects to the corresponding home page.



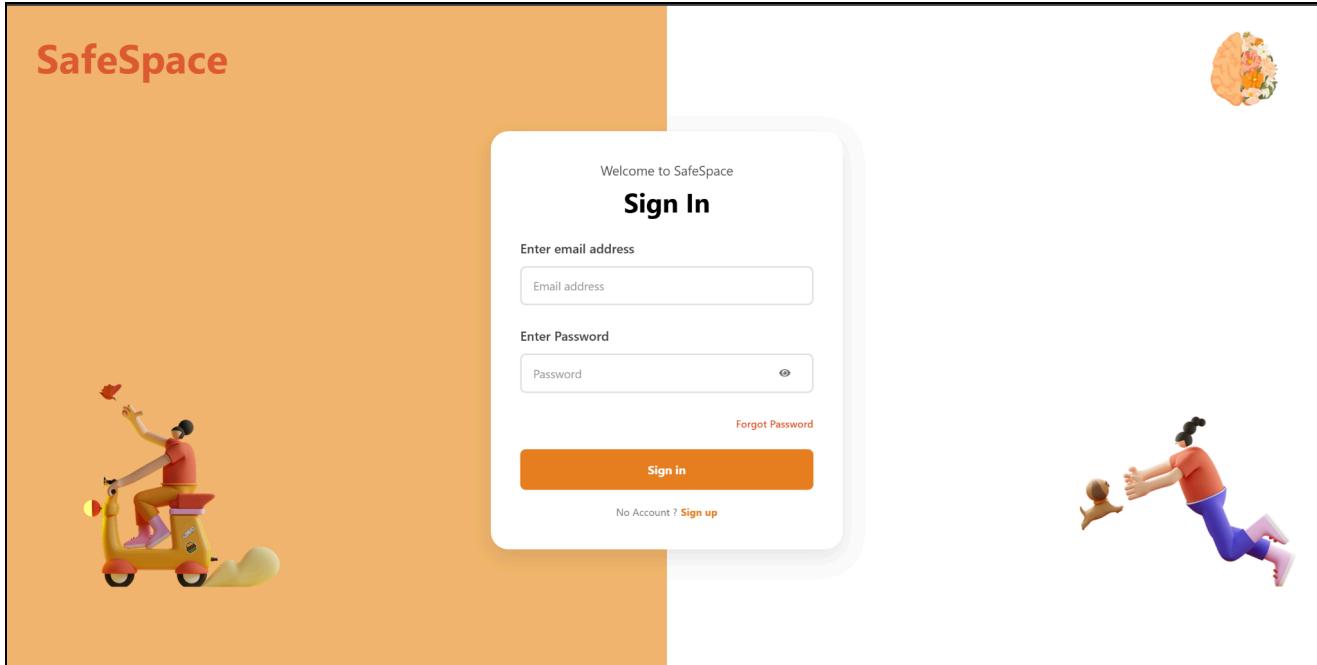
2. Logout redirects any user to the login page

Module Details: logout

Test Owner: Anirudh Singh

Test Date: [04/02/2025] - [04/02/2025]

Test Results: Test is successful for all roles (student and counsellor)



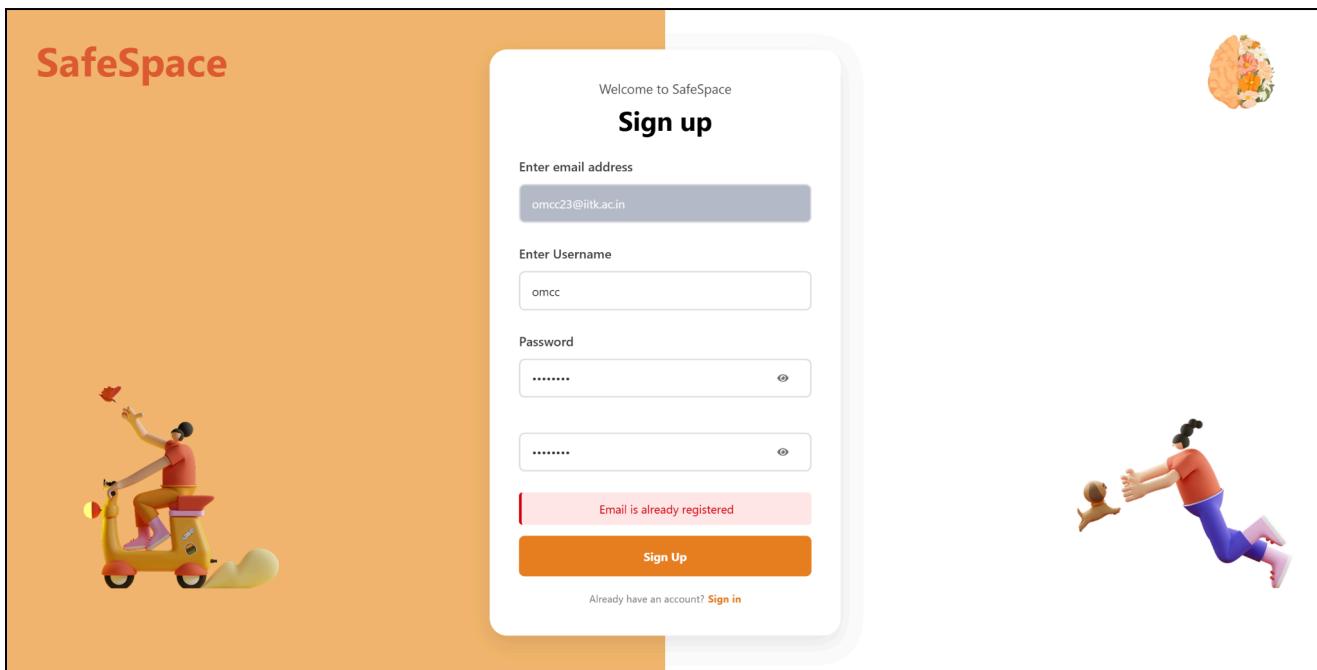
3. User cannot register with an already-used email

Module Details: sign-up

Test Owner: Yatharth Sharma

Test Date: [04/02/2025] - [04/02/2025]

Test Results: Registration is denied, and a prompt appears saying "Email is already registered".



```
_id: ObjectId('67e3e2e4ade920f9ad955cde')
email : "omcc23@iitk.ac.in"
password : "$2b$12$H2jWba.vZooTOh21v88LY0MQUJHyuOrlwVu3h55qP.VC4C8yuEPM."
name : "omcc23"
role : "student"
created_at : 2025-03-26T11:20:04.613+00:00
updated_at : 2025-04-04T16:47:40.175+00:00
```

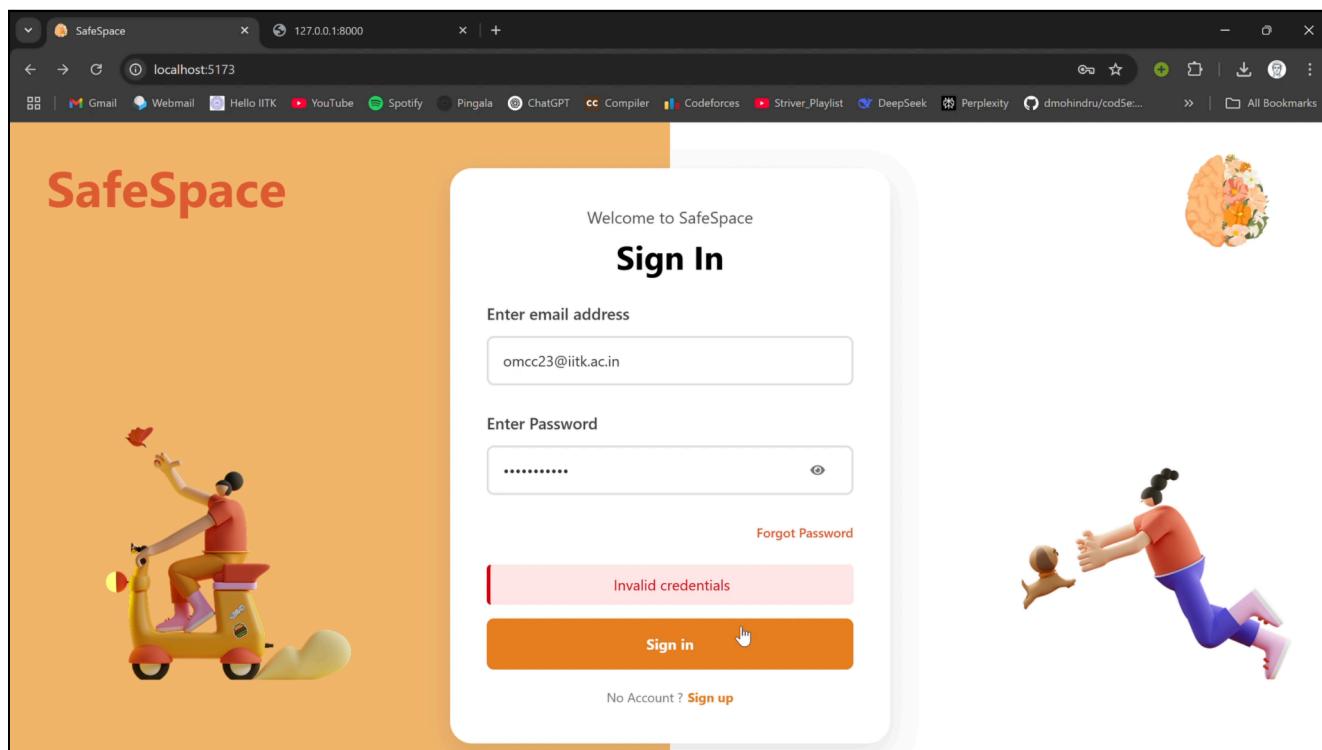
4. Incorrect credentials do not allow login

Module Details: login

Test Owner: Archita Goyal

Test Date: [04/02/2025] - [04/02/2025]

Test Results: The user was not allowed to login with incorrect credentials and was shown a prompt saying “Invalid credentials” when tried to do so.



5. Students can book an appointment and this will be reflected under ‘Appointments Status’ tab in ‘Profile Page’

Module Details: appointments, getappointments

Test Owner: Archita Goyal

Test Date: [04/02/2025] - [04/02/2025]

Test Results: Test passed. When the “Confirm Appointment” button was clicked after selecting a particular slot, the same time slot was reflected in the profile page of the student as well as updated in the database.

```
_id: ObjectId('67f1593025f69891f5183048')
user_name : "archita goyal"
user_email : "architag23@iitk.ac.in"
counselor_email : "counsellor1@iitk.ac.in"
date : 2025-04-05T00:00:00.000+00:00
time_slot : "14:30"
description : "I need to talk."
contact_no : "982*****"
status : "pending"
```

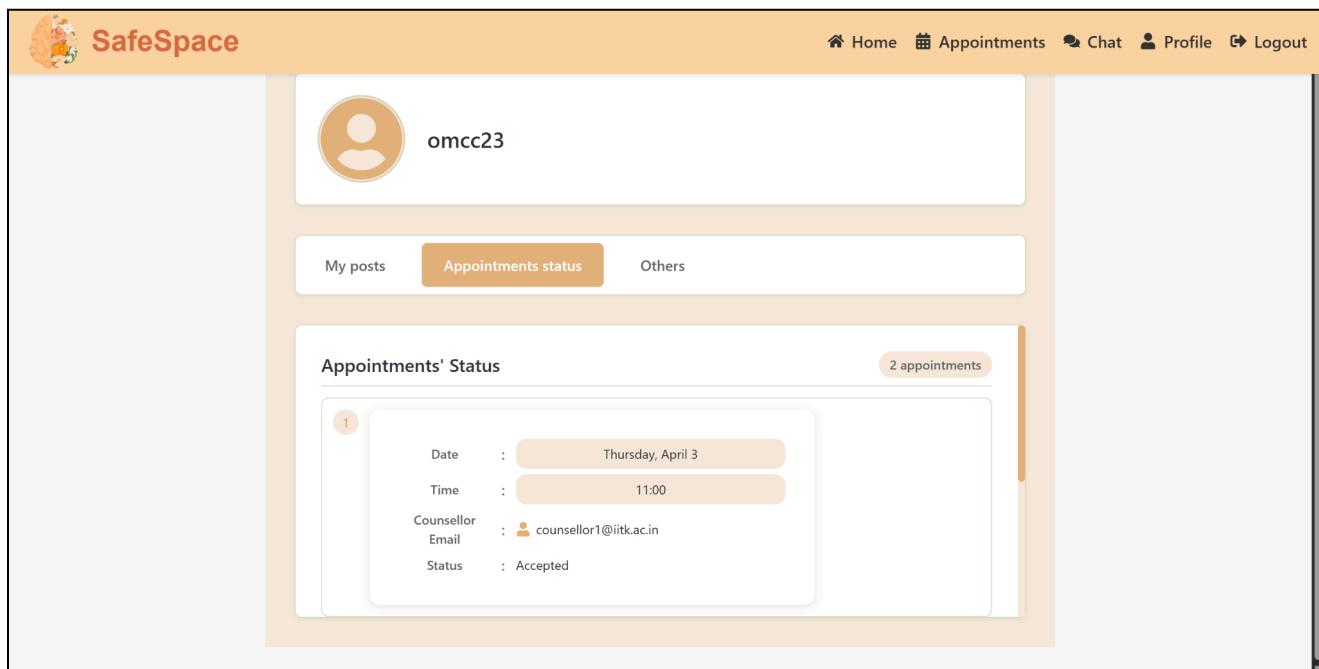
6. Students can view his/her appointments status

Module Details: getappointments

Test Owner: Om Chaudhari

Test Date: [03/04/2025] - [03/04/2025]

Test Results: Appointment details are shown as “pending”, “accepted” or “rejected” in ‘Appointments Status’ tab in ‘Profile Page’.



```

_id: ObjectId('67ee7893b24598334364d080')
user_name : "Om "
user_email : "omcc23@iitk.ac.in"
counselor_email : "counsellor1@iitk.ac.in"
date : 2025-04-03T00:00:00.000+00:00
time_slot : "11:00"
description : "me sed"
contact_no : "9699776686"
status : "accepted"

```

7. Counsellors can see the all the appointments

Module Details: appointment_requests

Test Owner: Nakul Patel

Test Date: [04/02/2025] - [04/02/2025]

Test Results: Appointments details are shown as “pending”, “accepted”, “rejected”, “all requests” under the ‘Requests’ tab in ‘counsellor dashboard’.

The screenshot shows the SafeSpace Counselor Dashboard. At the top, there's a navigation bar with icons for Home, Appointments, Chat, Profile, and Logout. Below the navigation bar, the main title is 'Counselor Dashboard' followed by 'Counseling Requests'. There are four tabs at the top of the list: 'All Requests' (highlighted in orange), 'Pending', 'Approved', and 'Rejected'. The 'Approved' tab shows one item for 'Yatharth Sharma' with the status 'approved'. The details are: Date: 22/03/2025, Time: 3:00 PM, Contact No.: +919929011045, Email: yatharths23@iitk.ac.in, Description: I am depressed. The 'Rejected' tab shows one item for 'Yatharth Sharma' with the status 'rejected'. The details are: Date: 22/03/2025, Time: 11:00 AM, Contact No.: +919929011045, Email: yatharths23@iitk.ac.in, Description: I am also depressed at 11 am.

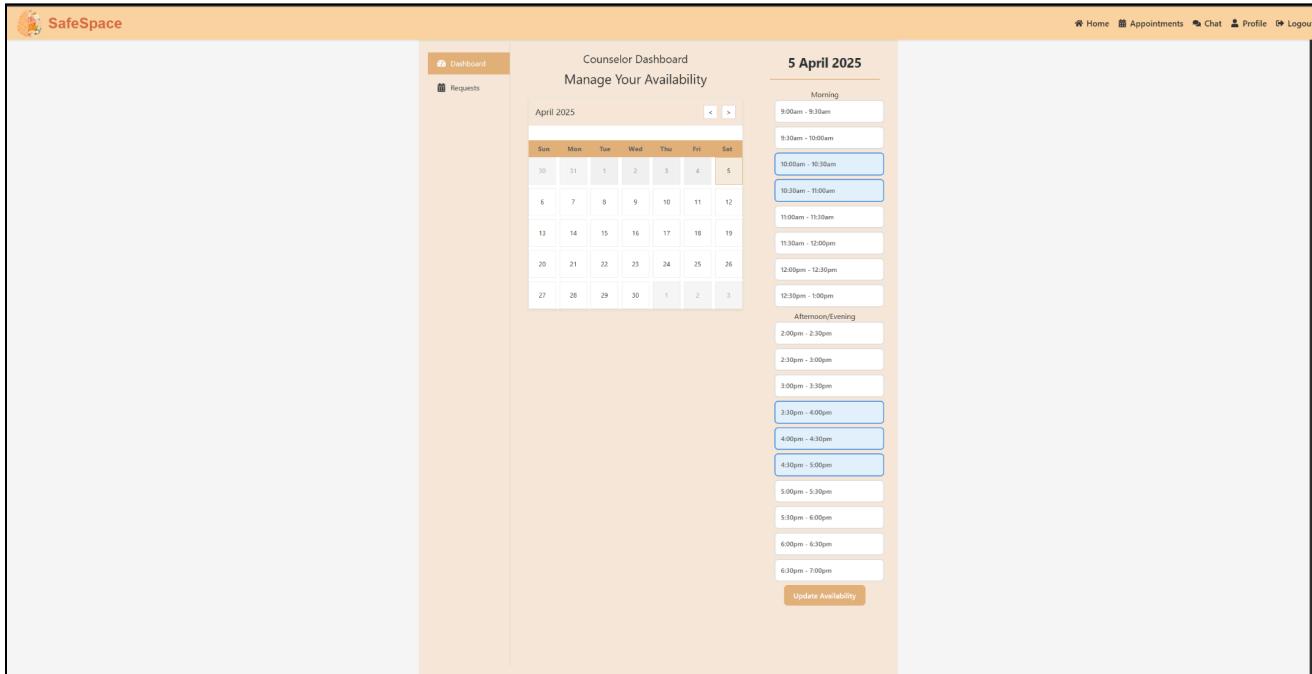
8. Counsellors can update his/her available time slots

Module Details: available_slots, update_slots

Test Owner: Naman Yadav

Test Date: [04/02/2025] - [04/02/2025]

Test Results: In the counsellor dashboard, one can update the availability by choosing date and time from the given calendar.



```

_id: ObjectId('67f0e55b09118eade436adb6')
counselor_email: "counselor2@iitk.ac.in"
date : 2025-04-05T00:00:00.000+00:00
▼ time_slots : Array (5)
  0: "15:30"
  1: "16:00"
  2: "16:30"
  3: "10:00"
  4: "10:30"

```

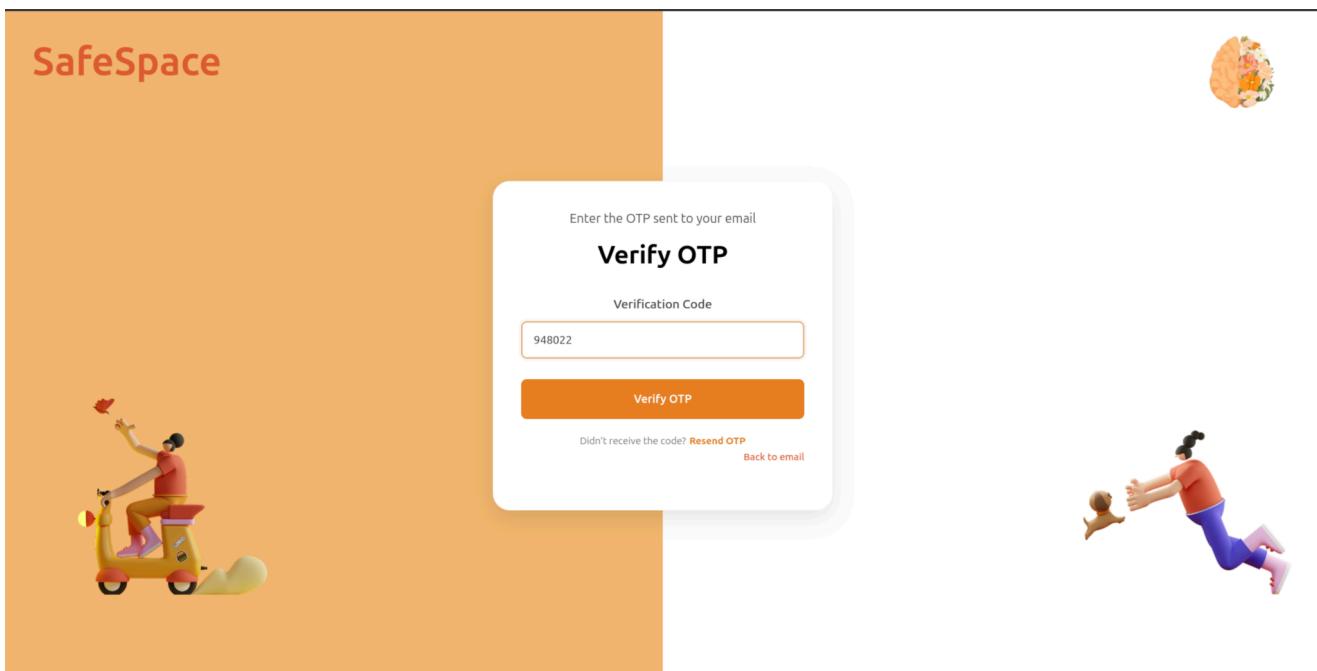
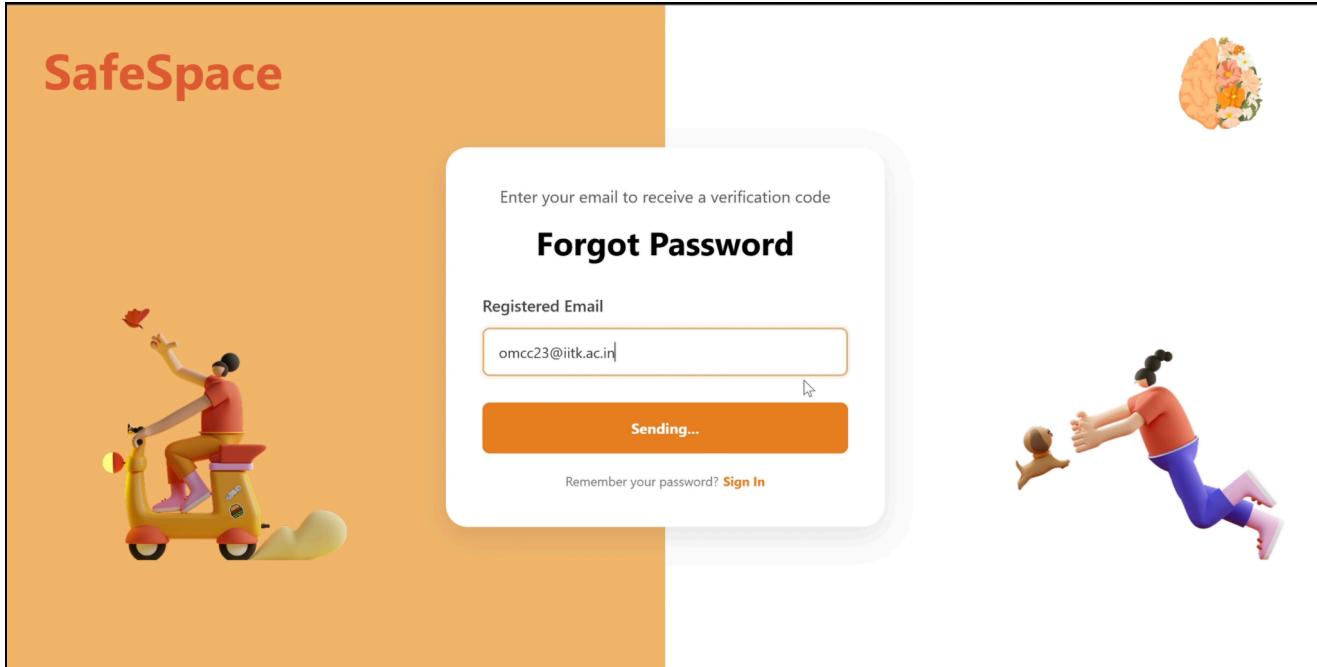
9. Registered users can reset their password

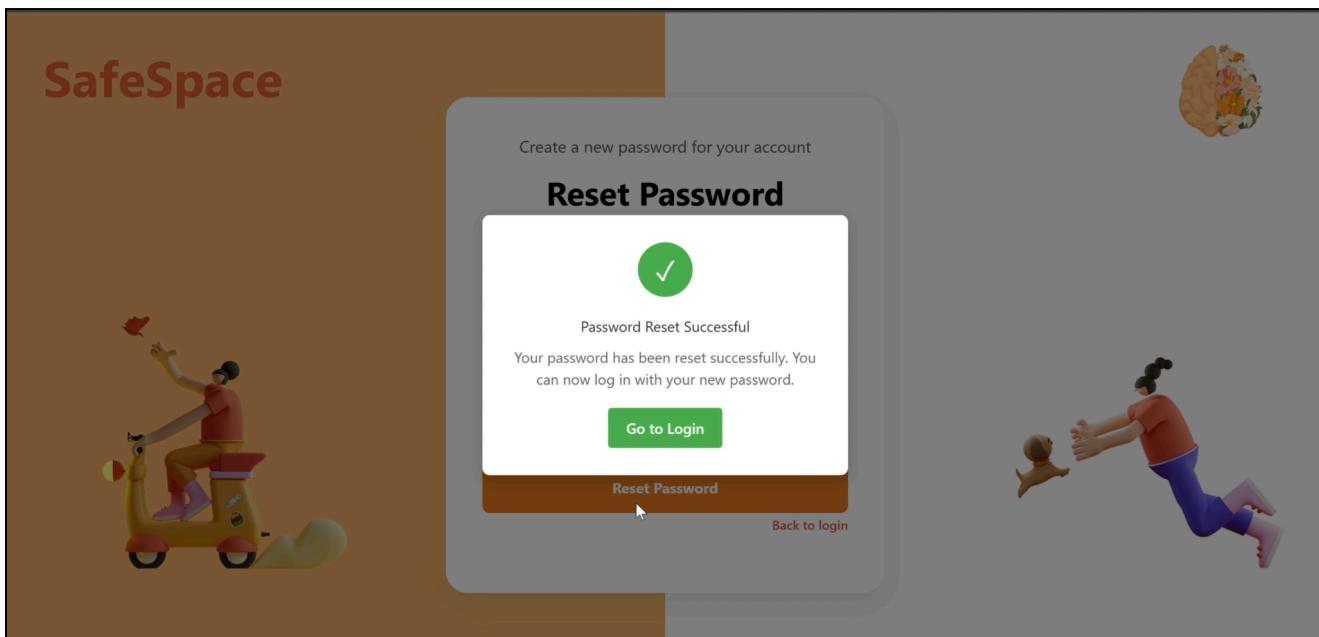
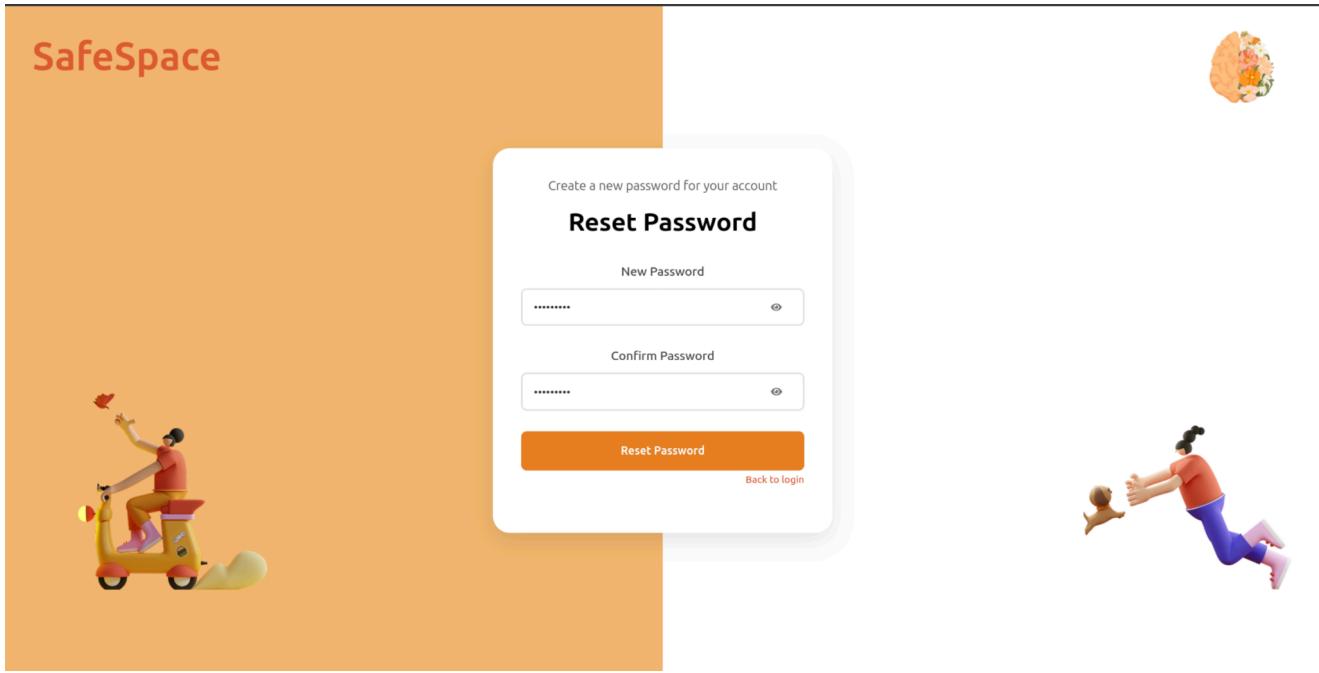
Module Details: reset-password

Test Owner: Om Chaudhari

Test Date: [04/02/2025] - [04/02/2025]

Test Results: On clicking 'Forgot Password', an OTP will be sent to the registered email id using which one can change their password.





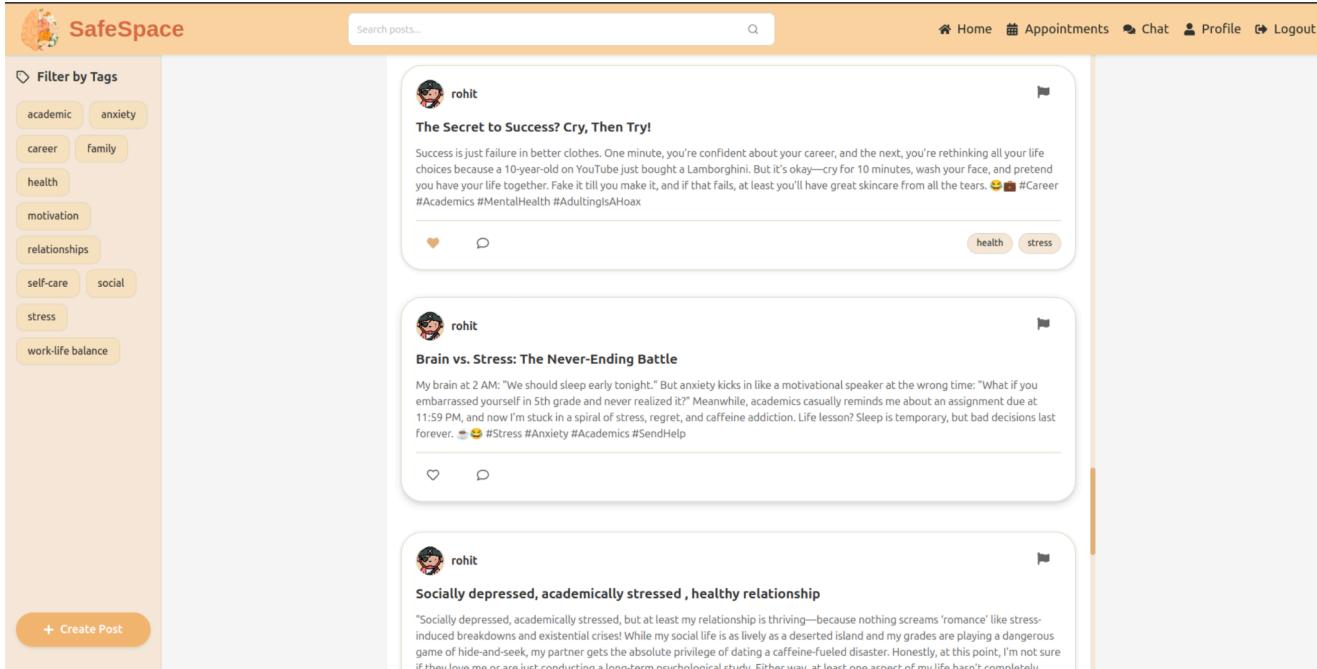
10. Users can see the posts

Module Details: see posts

Test Owner: Sayani Patra

Test Date: [04/02/2025] - [04/02/2025]

Test Results: The user can see the posts of counselors and other students on the home page.



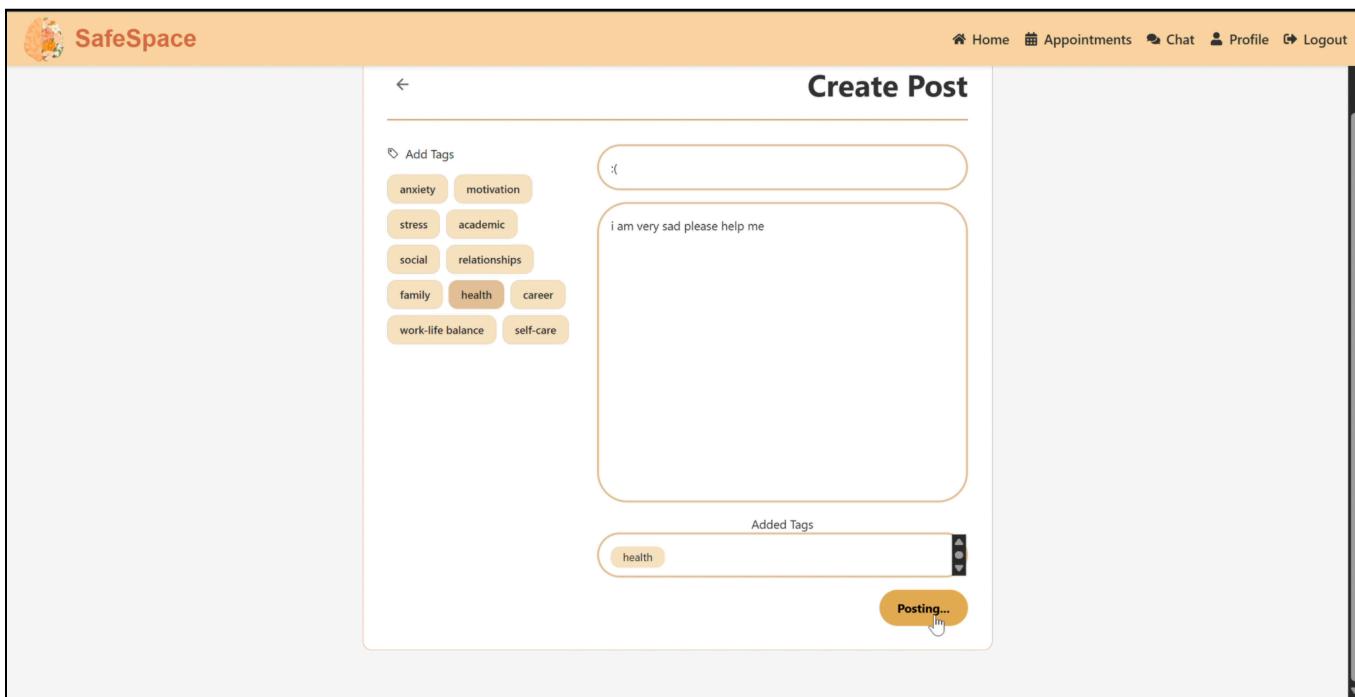
11. Users can upload posts

Module Details: login

Test Owner: Om Chaudhari

Test Date: [04/02/2025] - [04/02/2025]

Test Results: The user: 1) student: can upload posts by clicking on create button and then entering text in it. 2) counsellor: can also post images, videos along with the text.



```

_id: ObjectId('67f00efele6ce5b4df2d1d77')
title: ":(" 
content: "i am very sad please help me"
author: "omcc23"
author_id: "67e3e2e4ade920f9ad955cde"
relevance_tags: Array (1)
severity_tag: "moderate"
image: null
created_at: 2025-04-04T16:55:26.263+00:00
updated_at: 2025-04-04T16:58:10.701+00:00
likes: 2
image_url: null
  
```

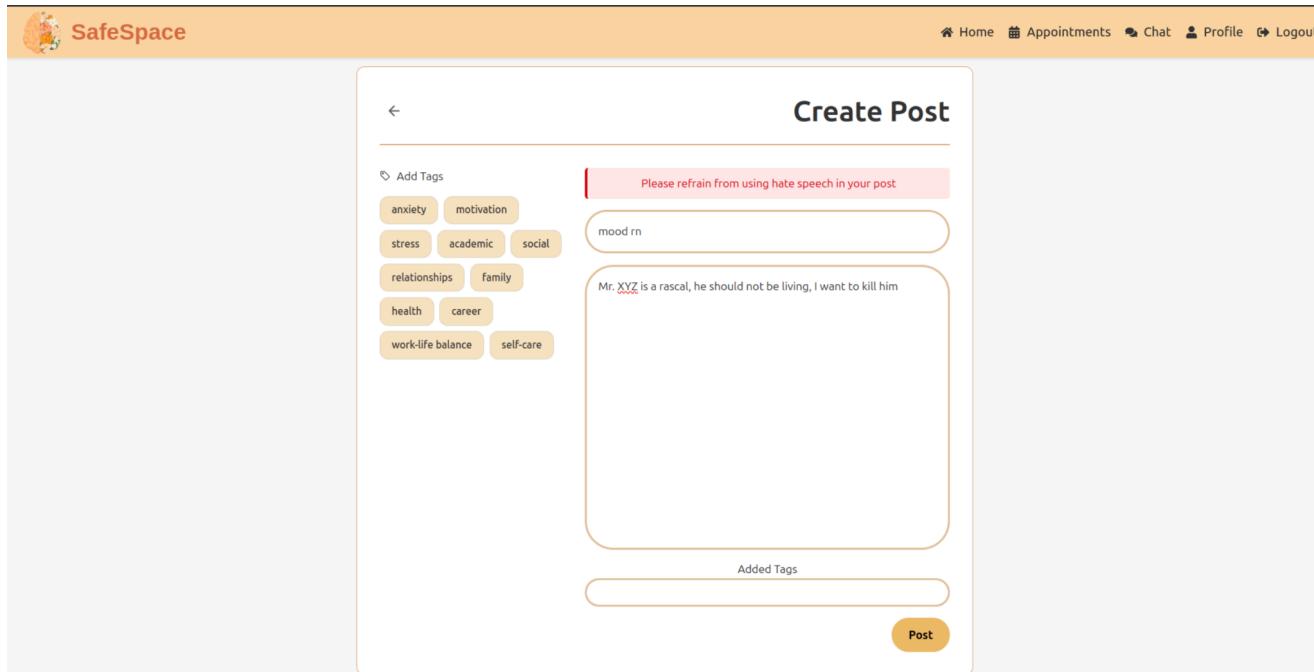
12. Users will be restricted from using foul language while creating posts.

Module Details: posts

Test Owner: Vivek

Test Date: [04/02/2025] - [04/02/2025]

Test Results: If a user or counselor attempts to create a post containing hateful content or offensive language, the system automatically prevents the submission.



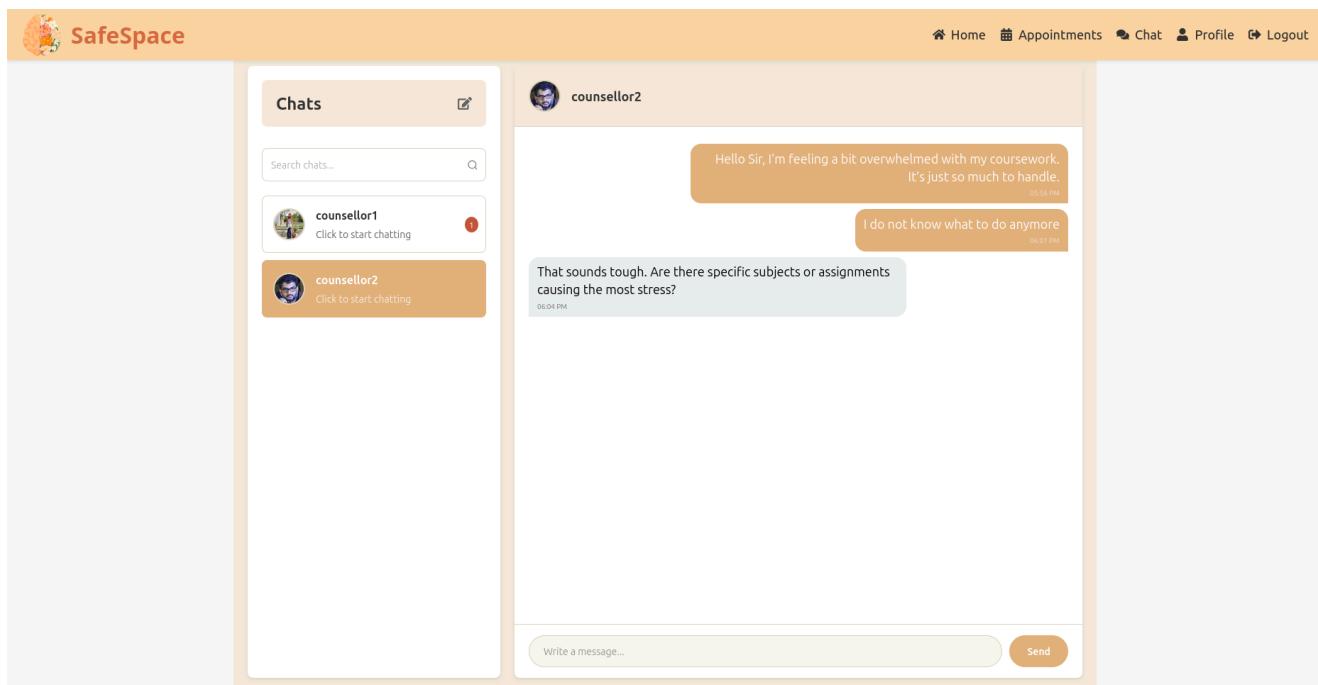
13. Students can Chat with the Counsellors and vice-versa.

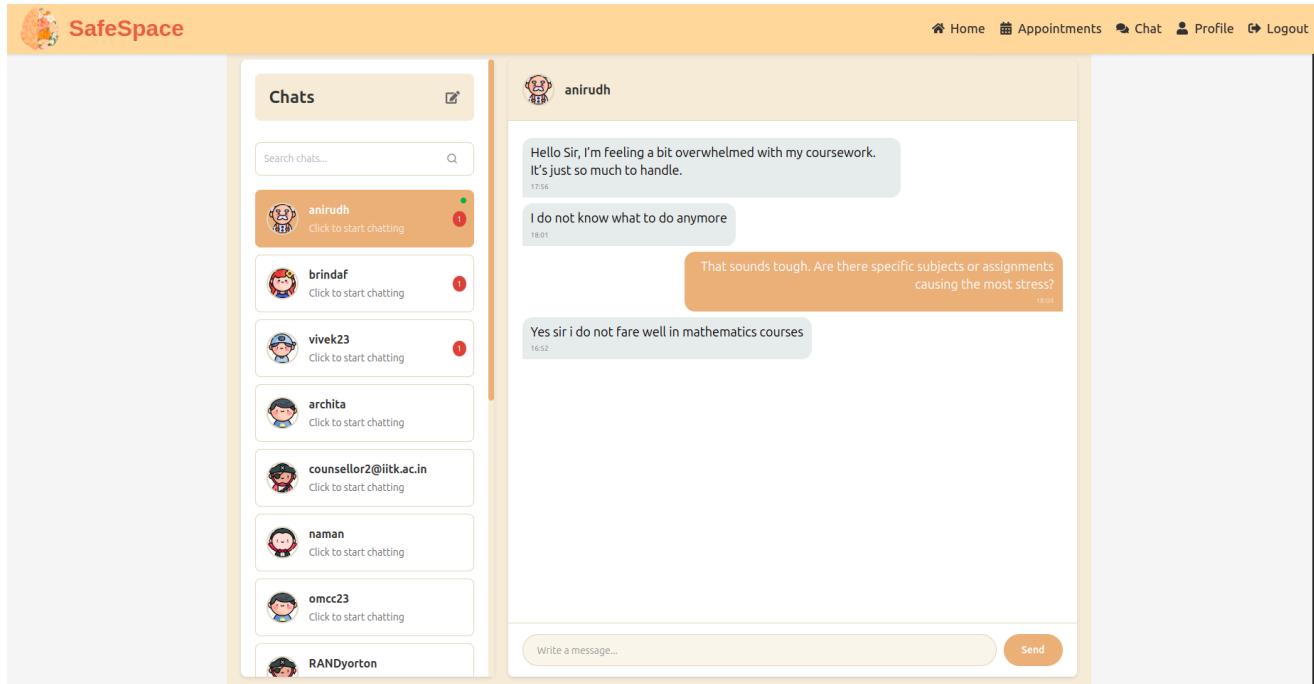
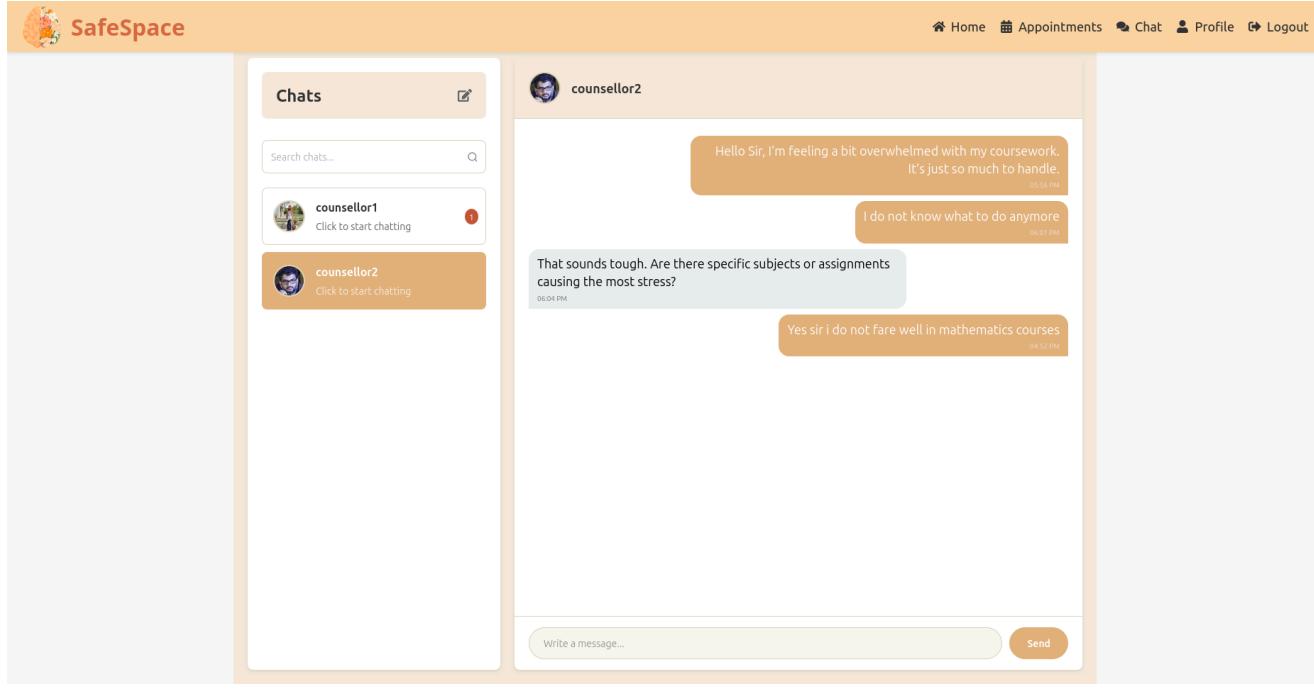
Module Details: Chats

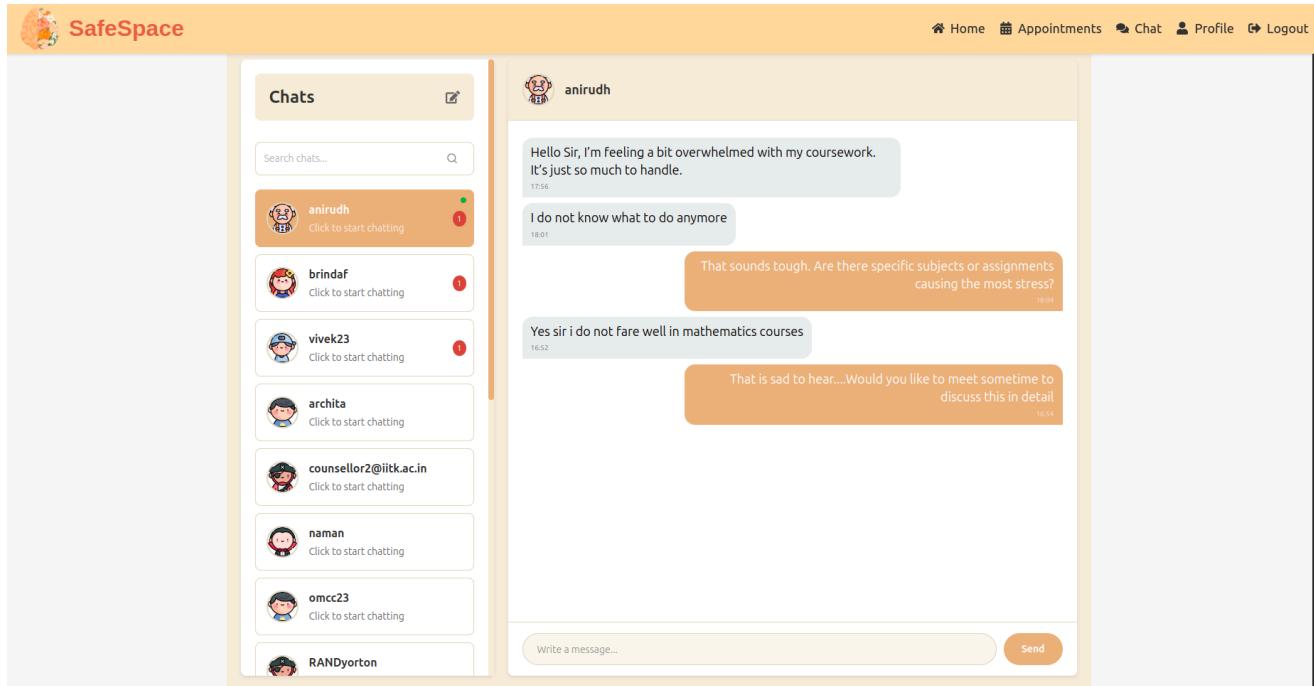
Test Owner: Anirudh Singh

Test Date: [04/02/2025] - [04/02/2025]

Test Results: A student can send messages to a counsellor to seek guidance While a counsellor can receive the said messages and reply to them , thus initiating a chat.







```

_id: ObjectId('67f263fd6677ed9ff230b65')
senderId : ObjectId('67db1278e480b17b41bd3aa')
receiverId : ObjectId('67dc4cb24145ca175f0b21b2')
text : "Yes sir i do not fare well in mathematics courses"
status : "read"
createdAt : 2025-04-06T11:22:37.032+00:00
updatedAt : 2025-04-06T11:23:06.422+00:00
__v : 0
  
```



4 System Testing

1. Requirement: User can create account using the sign-up page

Test Owner: Archita Goyal

Test Date: [02/03/2025] - [02/03/2025]

Test Results: Test was successfully conducted. For testing, users had to click on the 'sign up' button on the login page. When redirected to the registration page, filled up the registration form.

Registration was only successful using institute mail as no otp will be received otherwise. After entering otp, registration was completed. The student or counsellor was then able to login to the system using login credentials entered in the registration form.

Additional Comments: We are considering allowing both students and counselors to register in the same way. Counsellors then have to further mail to SafeSpace to change the role.

2. Requirement: User can login using the login page

Test Owner: Archita Goyal

Test Date: [02/03/2025] - [02/03/2025]

Test Results: Test was successfully conducted. For testing, the user had to enter login credentials on the login page and click on the 'login' button. If the credentials were the same as those entered in the registration form, the user was then able to login and redirected to the home page. An error message was displayed otherwise.

3. Requirement: Forgot Password button present on the login page and Reset Password button in the profile page.

Test Owner: Archita Goyal

Test Date: [02/03/2025] - [02/03/2025]

Test Results: Test was successfully conducted. For testing, users had to click on the forgot password button on the login page that had redirected him to the forgot password page, where he had to enter the institute mail ID and was able to change password after verifying the OTP sent on that ID. The user was also able to change the password by clicking on "Reset Password" button in the "Others" section of the "Profile" tab.

Additional Comments: Both students and counselors can reset their passwords.

4. Requirement: The homepage will feature a navigation bar containing links to key sections such as Chat, User Profile, Appointments, Homepage and also a search bar.

Test Owner: Archita Goyal

Test Date: [02/03/2025] - [03/03/2025]

Test Results: Test passed. A working navbar was present at the top of the webpage and contained the options to navigate to the Appointments tab, chat section, profile page, homepage and a button to logout of the application.

Additional Comments: All the buttons worked as expected.

5. Requirement: The user can edit his/her current avatar through the profile page.

Test Owner: Archita Goyal

Test Date: [01/04/2025] - [01/04/2025]

Test Results: Test was successfully conducted. For testing, users had to navigate to the profile page and click on the avatar and a designated box of available avatars . User was able to click on any of them and the avatar was updated.

Additional Comments: Counsellors have profile pictures additionally. Also, banner images have not been added in this version.

6. Requirement: A student can directly chat with any of the counsellors but not with any other student in the chat section.

Test Owner: Rohit Yadav

Test Date: [03/04/2025] - [03/04/2025]

Test Results: Test was successfully conducted. For testing, students can navigate to the chat page from the navigation bar where only counselors are available to chat on that page that ensures that no students can chat with other students maintaining the anonymity. This was verified by login as counselor and the chat page shows the message sent by the student. Additionally, counselors can chat with any registered students .

7. Requirement: Users can create posts containing: Title, Body text, and Relevant tags. Users can publish blogs immediately, save blogs as drafts. Counselors can also include images in their posts.

Test Owner: Archita Goyal

Test Date: [03/04/2025] - [03/04/2025]

Test Results: The test was successfully conducted. For testing, users had to click the “Create Post” button on the bottom left of the home page. There was an interface to write the content of a blog including its heading, content and attach the relevant tags. There is an additional feature of adding images which appears only when a counselor tries to create a post.

Additional Comments: The user can delete past posts but not edit them or save them as drafts in this version.

8. Requirement: Users can book an appointment with available counsellors and choose a slot based on the counsellor's specified availability.

Test Owner: Rohit Yadav

Test Date: [01/04/2025] - [01/04/2025]

Test Results: Test was successfully conducted. For testing, users had to select a counsellor from the available list on the appointments page. Available slots for the selected counsellor appeared on selecting a date. The suitable slot was selected and clicked “Confirm Appointment” redirecting to a form to fill personal details and description of the reason for booking the appointment. Once the details were filled and the “Confirm” button was clicked, the user was able to see the appointment status in their profile page.

9. Requirement: A student can see the current status (accepted,rejected,pending) of all the appointments requested by him to any counsellor.

Test Owner: Rohit Yadav

Test Date: [01/04/2025] - [01/04/2025]

Test Results: Test was successfully conducted. For testing, users had to click on the profile page button on navbar on the home page. When redirected to the profile page, click on appointments status button to see a designated box for list of past booked appointments with their status.

10. Requirement: Counsellors can accept/reject an appointment request received from a student.

Test Owner: Rohit Yadav

Test Date: [01/04/2025] - [01/04/2025]

Test Results: Test was successfully conducted. For testing, the counsellor had to select "Requests" tab on the appointments page and when redirected to the requests page, selected the "Pending Requests" and clicked on "Accept" or "Reject" for a particular request. The status of that request was accordingly updated in the profile page of that student as well as in the appointment requests of the counsellor.

Additional Comments: The counsellor can initiate chat with the student by searching for name in the chat section after rejecting the request if they wish to.

11. Requirement: Counsellor can choose his/her availability in a particular slot on a particular day.

Test Owner: Rohit Yadav

Test Date: [01/04/2025] - [01/04/2025]

Test Results: Test was successfully conducted. The counsellor navigated to the appointments page, selected a specific day using the calendar in the dashboard and updated available time slots for that day. This update was then verified by logging in as a student and confirming the availability of the selected slots for that counsellor.

12. Requirement: Users can like posts and add their comments to different users' posts.

Test Owner: Archita Goyal

Test Date: [01/04/2025] - [01/04/2025]

Test Results: Test was successfully conducted. The user navigated to the specific post they liked and clicked the heart-shaped button beneath the post. To comment, the user clicks on the comment icon and inputs a text. This was verified by checking the updated number of likes on the post and checking if the comment was added in the comment section of the post.

13. Requirement: Users can search specific posts.

Test Owner: Archita Goyal

Test Date: [03/04/2025] - [03/04/2025]

Test Results: Test was successfully conducted. The user navigated to the search bar in the top navigation bar. Then, the user inputs the title or tag names of desired posts. All the posts which matched the regular expressions of the title and the user-inputted text and the posts which contain the tags.

14. Requirement: Users can search specific counsellors by their names.

Test Owner: Nakul Patel

Test Date: [03/04/2025] - [03/04/2025]

Test Results: Test was successfully conducted. The user navigated to the search bar in the top in the chats window.

15. Requirement: User cannot sign up with an already registered email.

Test Owner: Archita Goyal

Test Date: [03/04/2025] - [03/04/2025]

Test Results: Test was successfully conducted. When the user tried to sign-up with an already registered email, he was not allowed to do that and an error message was displayed showing "Email is already registered".

16. Requirement: User cannot log in with incorrect credentials.

Test Owner: Archita Goyal

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Test was successfully conducted. When the users tried to login with incorrect username or password, they were not allowed to do that and an error message was displayed showing "Invalid Credentials".

17. Requirement: OTP expires after a certain time.

Test Owner: Archita Goyal

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Test was successfully conducted. When the user entered the OTP (generated during signup or changing password) after 10 minutes, it was no longer valid and needed to regenerate a new OTP.

18. Requirement: Message cannot be sent with an empty body.

Test Owner: Archita Goyal

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Test was successfully conducted. The user was not allowed to send a message with only spaces or an empty string.

19. Requirement: Message delivery status updates correctly

Test Owner: Archita Goyal

Test Date: [04/04/2025] - [04/04/2025]

Test Results: Test was successfully conducted. The number of unread messages was correctly displayed along with the name of every user and vanished on opening the chat(i.e, the messages were marked as seen).

5 Conclusion

As far as we can see, the testing was sufficiently exhaustive and we do not expect a significant number of errors to arise if the system were deployed.

The number of bugs we found and corrected was more than ___, indicating that the testing was effective.

The testing process can be improved by having a larger number of students in the team and dividing the team into groups, so as to reduce the amount of communication between the developers and testers of each feature.

There is also a conflict of interest in the testing process if the testers and developers are part of the same group, therefore we tried to keep the two teams separate, and no developer tested a feature that they had developed. Separating the two groups and the assessments of their work would incentivise testers to think as end-users rather than custodians of the software. In fact, this is standard practice in the certification of safety-critical software, i.e. independent regulators certify systems as safe.

Giving more time to the testing process would also have improved the testing process. However, as we had limited time, implementation was given priority, with as exhaustive testing as possible.

Appendix A - Group Log

Sr. No.	Date	Agenda
1	31/03/25	First meet for the discussion for Testing and Manual Document. Work distribution done.
2	01/04/25	Manual System Testing was done and doubts discussed on User Manual
3	02/04/25	Initiated component testing. User Manual reviewed
4	03/04/25	Initiated integration testing. User Manual was finalized
5	04/04/25	User Manual Doc submitted and Testing Doc was Finalized
6	05/04/25	Testing Doc finalized and Submitted