
Test Document

for

Roombie

Version 1.0

Prepared by

Group #: 7

Aarsh Jain	230015
Aritra Ambudh Dutta	230191
Aritra Ray	230193
Bhukya Vaishnavi	230295
Bikramjeet Singh	230298
Hitarth Makawana	230479
Shlok Jain	230493
Ronav Puri	230815
Rathod Ayushi	230844
Saksham Verma	230899
Surepally Pranaysriharsha	231057

Group Name: Marauders

aarshjain23@iitk.ac.in
aritraad23@iitk.ac.in
aritrar23@iitk.ac.in
bhukyav23@iitk.ac.in
bsingh23@iitk.ac.in
hitarthkm23@iitk.ac.in
jainshlok23@iitk.ac.in
ronavgp23@iitk.ac.in
rathoday23@iitk.ac.in
sakshamv23@iitk.ac.in
surepally23@iitk.ac.in

Course: CS253

Mentor TA: *Nij Bharatkumar Padariya*

Date: 04/04/2025

Contents

<u>CONTENTS</u>	2
<u>REVISIONS</u>	3
 1 <u>INTRODUCTION</u>	4
 2 <u>UNIT TESTING</u>	6
 3 <u>INTEGRATION TESTING</u>	80
 4 <u>SYSTEM TESTING</u>	127
 5 <u>CONCLUSION</u>	140
 <u>APPENDIX A - GROUP LOG</u>	142

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
V1.0	Aarsh Jain Aritra Ambudh Dutta Aritra Ray Bhukya Vaishnavi Bikramjeet Singh Hitarth Makawana Shlok Jain Ronav Puri Rathod Ayushi Saksham Verma Surepally Pranaysriharsha	First Version of the Testing Document	04/04/2025

1 Introduction

Test Strategy:

We used manual testing to test our web application, **Roombie**, wherein each of the testers individually tested the application for various inputs or cases and matched the obtained result with the desired one.

The testers were keen to have a precise check on all the functional and non-functional requirements to be fulfilled by the application as per the **SRS Document**.

Testing Period:

Testing was an integral part of the development process, and we manually tested the code, but a major portion of the testing was done after the implementation process.

The partial testing during development made it easier for us to find errors during the final testing since any bugs encountered were easy to tackle and fix as a result of the knowledge of the code structure as well as the possible edge cases in mind. This made the testing efficient and less time-consuming.

Testers of the App:

Developers were the main testers of the main application due to lack of time and size of the team. However we ensured that the persons who developed a certain component did not test the same so that it was tested efficiently and with neutrality. Being the developers, we were well aware of the various input/test cases and hence carried out the testing extensively, ensuring the robustness of the platform. The various errors identified were corrected as and when discovered.

Coverage Criteria:

We have used functional and non-functional coverage. For these, we tried to stick to the **SRS Document** to guide our way through system testing.

Tools used for Testing:

We used **Postman** to conduct comprehensive testing of our web platform's APIs. It is a widely recognized and powerful API testing tool that enables developers to design, test, and document APIs efficiently. The key advantages that lured us into using it for testing as well as for manual checking during the development process:

- **Intuitive Interface:** Postman provides a clean and user-friendly interface, which greatly simplifies the process of creating, sending and analyzing API requests. It allows the testers to build test cases without dealing with complex configurations.
- **Seamless Integration:** Postman integrates seamlessly with various development workflows, supporting environments for automated testing, API documentation, and mock server setups. This ensures a smooth transition between the development and testing phases.
- **Robust Feature Set:** Postman offers a range of advanced features, including support for authentication mechanisms, automated testing with scripts, detailed request history tracking, and many more. Although we did not use all the features exhaustively, the few we used greatly helped us in streamlining the testing process.
- **Cross-Platform & Cloud Support:** Postman is available as a desktop application and a cloud-based tool, allowing developers to sync their work across devices and collaborate efficiently.

By leveraging this tool, we ensured thorough API testing of the web platform.

2 Unit Testing

1a. SignUp-Tenant-Backend

In this unit we tested the backend function through which a new tenant can sign up on **Roombie**.

Unit Details: Testing of this unit ensures that a new user is registered correctly with the entered details. It also ensures that no two users are registered with the same email.

Test Owner: Bikramjeet Singh

Test Date: 01/04/2025

Test Results: A new user was correctly registered. An object of the class “Tenant” was created. If the email already existed a prompt was shown alerting the user about the same.

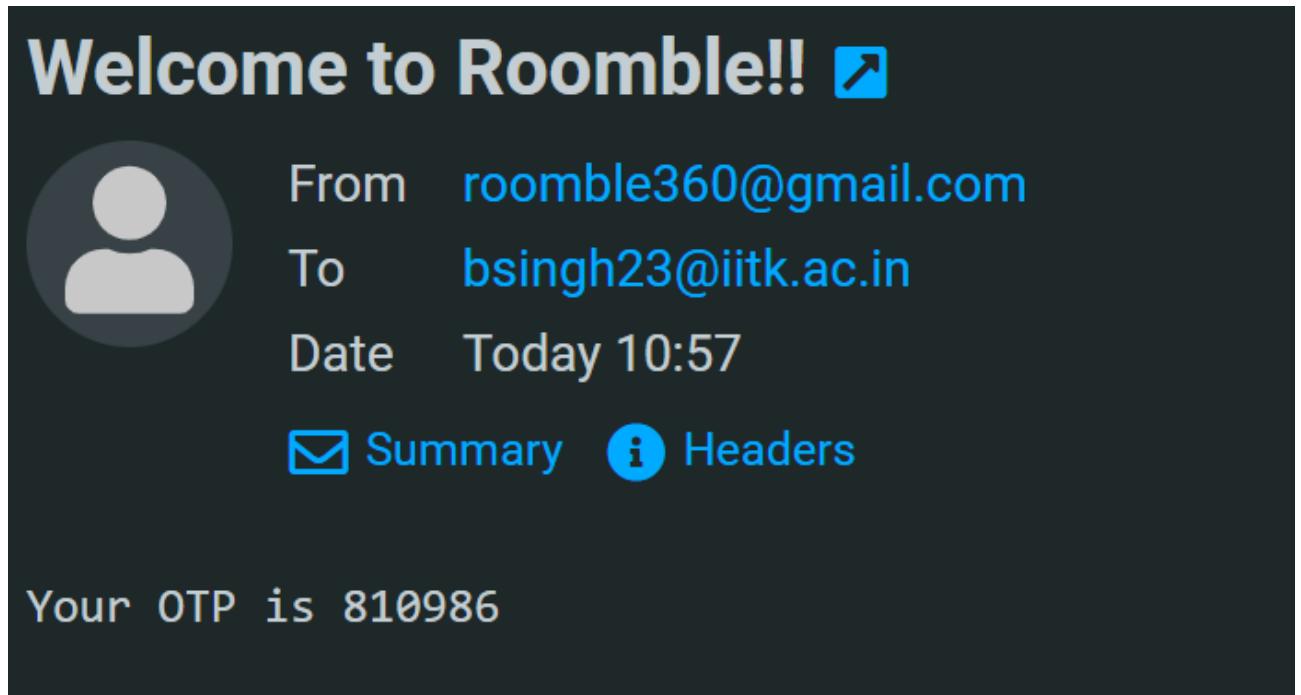
- New Email:

```

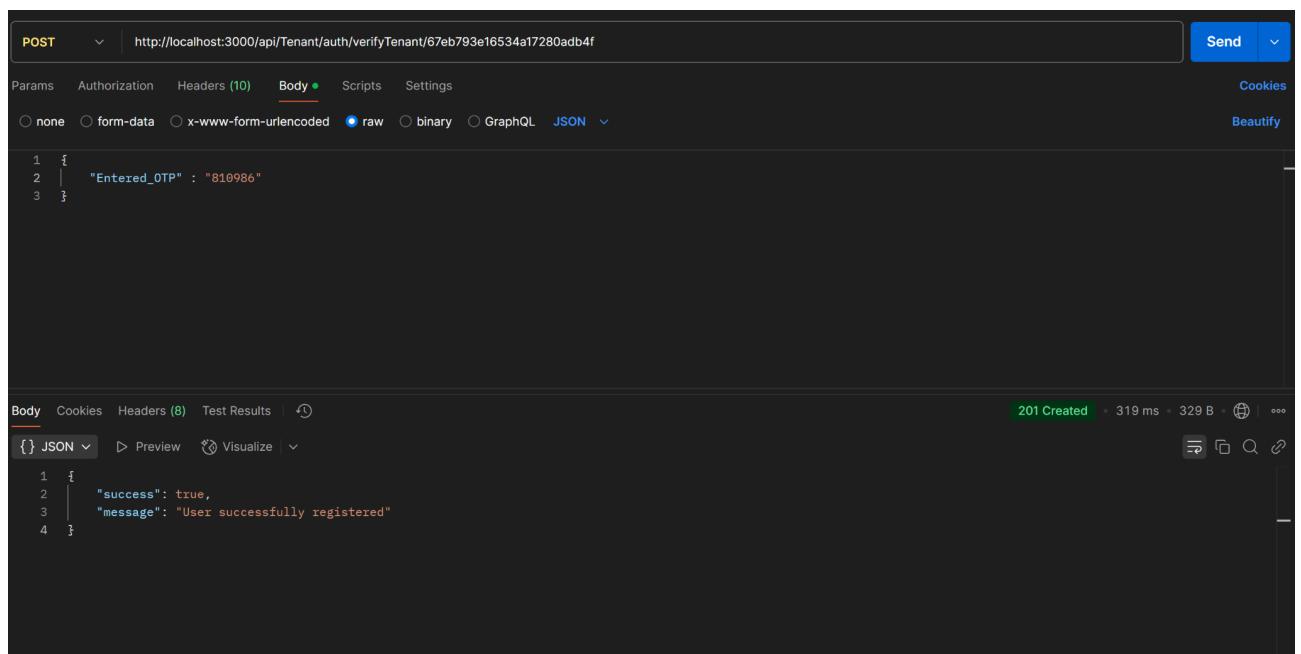
POST http://localhost:3000/api/Tenant/auth/Tenant_register
Params Authorization Headers (10) Body Scripts Settings
Body raw JSON
1 {
2   "name": "Bikramjeet Singh",
3   "email": "bsingh23@iitk.ac.in",
4   "password": "123456",
5   "locality": "Juhu",
6   "city": "Mumbai",
7   "gender": "true",
8   "smoke": "false",
9   "veg": "false",
10  "pets": "false",
11  "flatmate": "false"
12 }

Body Cookies Headers (8) Test Results
{} JSON > Preview Visualize
1 {
2   "success": true,
3   "message": "67eb793e16534a17280adb4f"
4 }
  
```

For a new email, at first an ID is generated and an email is sent with an OTP which is used for verification.



*After sending the OTP along with the ID in the params
The user is successfully registered.*



A screenshot of the Postman application interface. A POST request is being made to the URL `http://localhost:3000/api/Tenant/auth/verifyTenant/67eb793e16534a1280adb4f`. The request body is a JSON object with one key-value pair: `"Entered_OTP" : "810986"`. The response status is 201 Created, with a response time of 319 ms and a response size of 329 B. The response body is a JSON object with keys `"success"` and `"message"`, both containing the value `"User successfully registered"`.

The user along with his/her details, is available now in the Tenant Database.

```

_id: ObjectId('67eb7a3d16534a17280adb52')
name : "Bikramjeet Singh"
type : "tenant"
email : "bsingh23@iitk.ac.in"
password : "$2b$10$mAoIeheiepmJthU0Y0hx7ucfBrVNv1W1Lct.8hfPmTGIxR0q0tJ30"
locality : "Juhu"
city : "Mumbai"
gender : true
smoke : false
veg : false
pets : false
flatmate : false
description : "This user hasn't setup a description yet"
▶ conversations : Array (empty)
▶ bookmarks_tenants : Array (empty)
▶ bookmarks_property : Array (empty)
Images : "http://127.0.0.1:3000/Pictures/Default.png"
▶ reviews : Array (empty)
__v : 0

```

- Existing Email:

The screenshot shows a POST request in Postman to the URL `http://localhost:3000/api/Tenant/auth/Tenant_register`. The request body is a JSON object representing a user profile:

```

1 {
2   "name" : "Bikramjeet Singh",
3   "email" : "bsingh23@iitk.ac.in",
4   "password" : "123456",
5   "locality" : "Juhu",
6   "city" : "Mumbai",
7   "gender" : "true",
8   "smoke" : "false",
9   "veg" : "false",
10  "pets" : "false",
11  "flatmate" : "false"
12 }

```

The response status is 400 Bad Request, with the message: "A user already exists with the given credentials".

The user is prompted regarding the existence of an account with the same credentials.

Structural Coverage: Functional Coverage (related to register route in the backend), Decision coverage (covers the check for existing user with the same email).

Additional Comments: None

1b. SignUp-Tenant-Frontend

In this unit we tested the frontend part of the signup-tenant functionality.

Unit Details: Testing of this unit ensures that a new user can easily sign up through the user-friendly interface of our application.

Test Owner: Bikramjeet Singh

Test Date: 01/04/2025

Test Results: A new tenant can easily sign-up by first entering the email, password and the necessary details.

- New Email:

The screenshot shows the Roomble website's sign-up process for tenants. At the top, there's a navigation bar with a logo, 'Home', 'Sign-Up as Tenant', 'Sign-Up as Landlord' (which is underlined, suggesting it's the active or intended path), and 'Login'. Below this, the main content area has a title 'Signup as a Tenant'. It contains four input fields: 'Full Name' with 'Bikramjeet Singh' typed in, 'Email Address' with 'bsingh23@iitk.ac.in', 'Password' with '*****', and 'Confirm Password' with '*****'. At the bottom right of the form is a large red 'Next' button. A small note at the bottom right of the page says 'With Roomble, you'll stumble on the perfect place to rumble!'

User enters the email, password, and the matching password

If the passwords do not match the user is prompted to correct it.

Afterwards the user is asked to fill a form with some necessary details if he/she misses some of them a related prompt alerts the user regarding the same.

The screenshot shows the Roomble sign-up process. At the top, there's a navigation bar with links for Home, Sign-Up as Tenant, Sign-Up as Landlord, and Login. Below the navigation is the Roomble logo and tagline "YOUR PERFECT SPACE, JUST A CLICK AWAY!". The main form area has a title "A Few Questions About You". It includes dropdowns for location ("Where would you like to look for a property? (For better recommendations)"), gender ("Gender"), and select locality. There are also buttons for "Do you drink/smoke?", "Do you have pets?", "Food Preferences", and "Are you seeking a flatmate?". At the bottom are "Back" and "Sign up" buttons.

Where would you like to look for a property?
(For better recommendations)

Mumbai

Select Locality

Gender ! Please select an item in the list. FEMALE

Do you drink/smoke? YES NO

Do you have pets? YES NO

Food Preferences VEG NON-VEG

Are you seeking a flatmate? YES NO

← Back Sign up

Afterward, an OTP is sent to the concerned mail of the user and the user is directed to the following page, wherein he/she needs to enter the correct OTP, entering the wrong OTP gives an error message.

The screenshot shows the OTP verification page. It features the Roomble logo and tagline. The main title is "Verify OTP". Below it, an error message says "Invalid OTP, please try again". There's an "Enter OTP" field with a numeric keypad showing "6 0 1 7 1 1". A "Verify OTP" button is below the keypad. At the bottom, a footer message reads "With Roomble, you'll stumble on the perfect place to rumble."

Invalid OTP, please try again

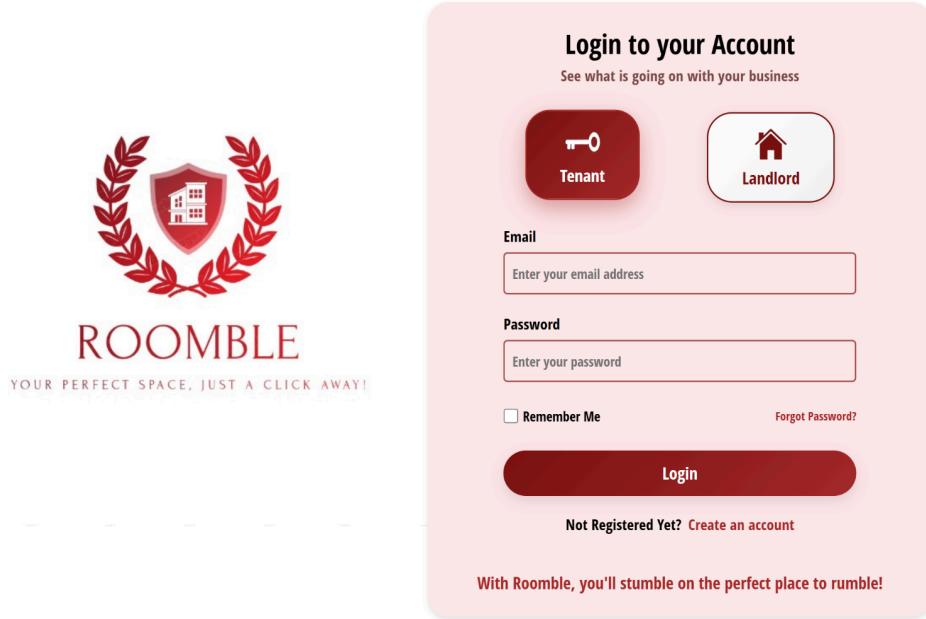
Enter OTP

6 0 1 7 1 1

Verify OTP

With Roomble, you'll stumble on the perfect place to rumble.

Entering the correct OTP, redirects the user to the login page:



1c. SignUp-Landlord-Backend

In this unit, we tested the backend function that ensures that a new landlord can sign up on **Roomble**.

Unit Details: Testing of this unit ensures that a new user is registered correctly with the entered details. It also ensures that no two users are registered with the same email.

Test Owner: Bikramjeet Singh

Test Date: 01/04/2025

Test Results: A new user was correctly registered. An object of the class “Tenant” was created. If the email already existed a prompt was shown alerting the user about the same.

- New Email:

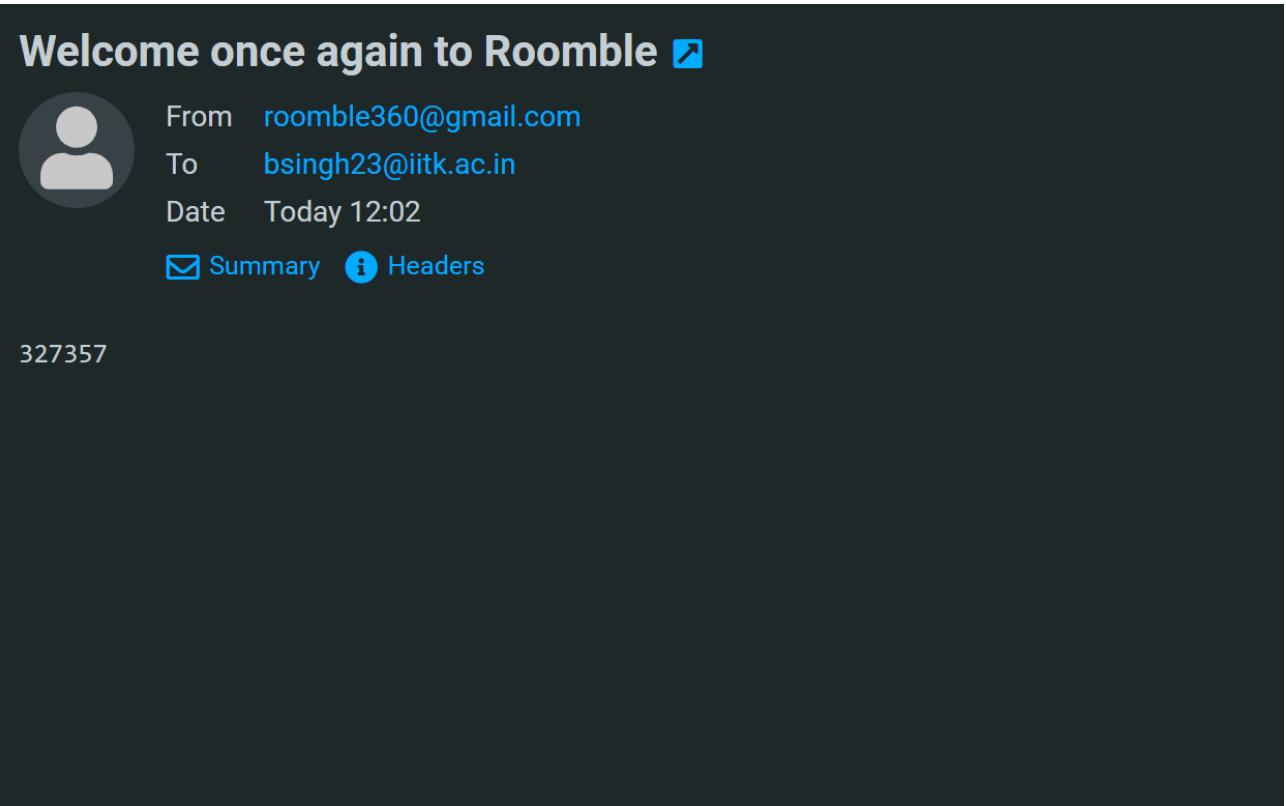
The screenshot shows the Postman application interface. At the top, it says "POST" and "http://localhost:3000/api/Landlord/auth/Landlord_register". Below this, there are tabs for "Params", "Authorization", "Headers (10)", "Body", "Scripts", and "Settings". The "Body" tab is selected and has a "raw" radio button selected. The body content is a JSON object:

```
1 {
2   "name" : "Bikramjeet Singh",
3   "email" : "bsingh23@iitk.ac.in",
4   "password" : "123456"
5 }
```

At the bottom right of the main area, it shows "200 OK" with a response time of "3.42 s" and a size of "320 B". Below the main area, there are tabs for "Body", "Cookies", "Headers (8)", "Test Results", and "Visualize". The "Body" tab is selected and shows the JSON response:

```
1 {
2   "message": "67eb88748d0b1b43cf49ff42",
3   "success": true
4 }
```

For a new email, at first, an ID is generated and an email is sent with an OTP, which is used for verification.



*After sending the OTP along with the ID in the params
The user is successfully registered.*

The screenshot shows a POST request to `http://localhost:3000/api/Landlord/auth/verifyLandlord/67eb88748d0b1b43cf49ff42`. The request body is a JSON object with one field: `"Entered OTP": "327357"`. The response status is `201 Created` with a response time of `298 ms` and a size of `324 B`. The response body is a JSON object with fields `"success": true` and `"message": "Successfully registered"`.

The user along with his/her details, is available now in the Tenant Database.

The screenshot shows a MongoDB query interface with a single result. The document contains the following fields:

```

_id: ObjectId('67eb89478d0b1b43cf49ff45')
name: "Bikramjeet Singh"
type: "landlord"
email: "bsingh23@iitk.ac.in"
password: "$2b$10$Hd/WTAUJS4gxM/MTD0BdVefiPr3EZ9wIW9lnUJu8i2dIdSnm0Y2Y6"
propertyList: Array (empty)
conversations: Array (empty)
Images: "http://127.0.0.1:3000/Pictures/Default.png"
reviews: Array (empty)
__v: 0
  
```

- Existing Email:

The user is prompted regarding the existence of an account with the same credentials.

The screenshot shows a Postman interface. The request method is POST, the URL is http://localhost:3000/api/Landlord/auth/Landlord_register, and the Body is set to JSON. The JSON payload is:

```
1 "name" : "Bikramjeet Singh",
2 "email" : "bsingh23@iitk.ac.in",
3 "password" : "123456"
```

The response status is 400 Bad Request, with a response time of 87 ms and a response size of 367 B. The response body is:

```
1 {
2   "success": false,
3   "message": "A user already exists with the given credentials",
4   "status": 400
5 }
```

Structural Coverage: Functional Coverage (related to register route in the backend), Decision coverage (covers the check for existing user with the same email).

Additional Comments: None

1d. SignUp-Landlord-Frontend

In this unit we tested the frontend part of the signup-landlord functionality.

Unit Details: Testing of this unit ensures that a new user can easily sign-up through the user-friendly interface of our application.

Test Owner: Bikramjeet Singh

Test Date: 01/04/2025

Test Results: A new tenant can easily sign-up by first entering the email, password and the necessary details.

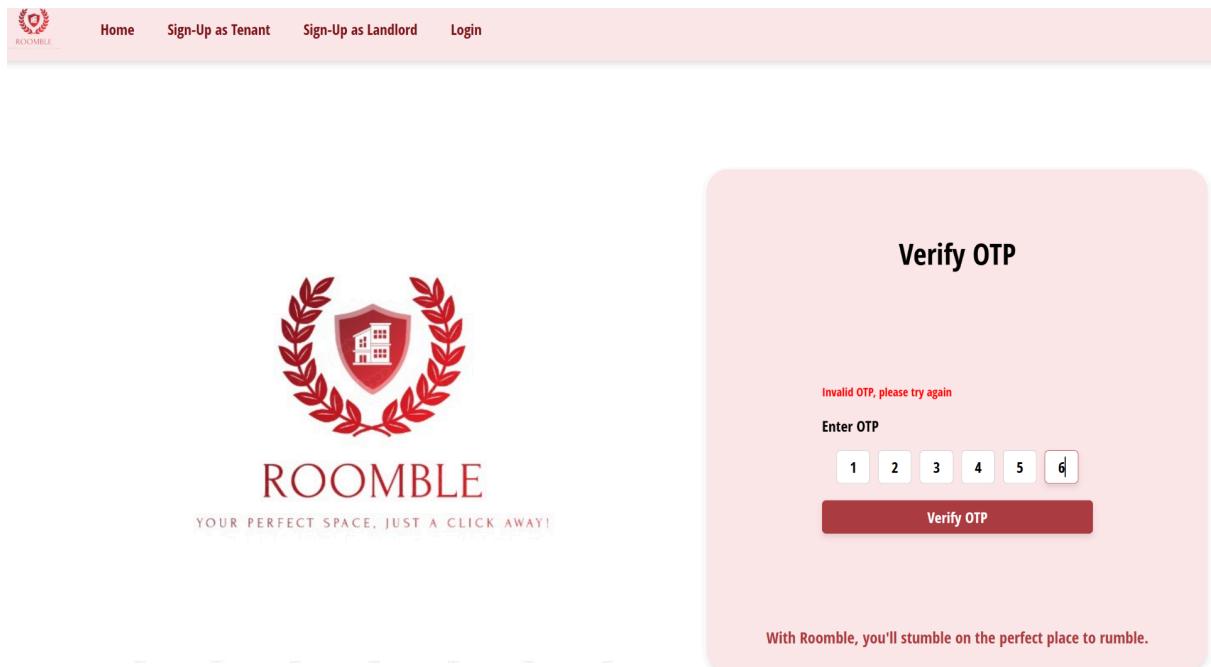
- New Email:

The screenshot shows the Roomble website's sign-up page for landlords. At the top, there is a navigation bar with links for Home, Sign-Up as Tenant, Sign-Up as Landlord, and Login. The main content area features the Roomble logo (a red shield with a building inside, surrounded by a laurel wreath) and the text "ROOMBLE" and "YOUR PERFECT SPACE, JUST A CLICK AWAY!". To the right, a large pink rounded rectangle contains the "Signup as a Landlord" form. The form fields are: "Full Name" (Bikramjeet Singh), "Email Address" (bikramjeetsinghgill2015@gmail.com), "Password" (*****), and "Confirm Password" (*****). Below the form is a "Sign up" button and a note: "With Roomble, you'll stumble on the perfect place to rumble!".

User enters the email, password, and the matching password

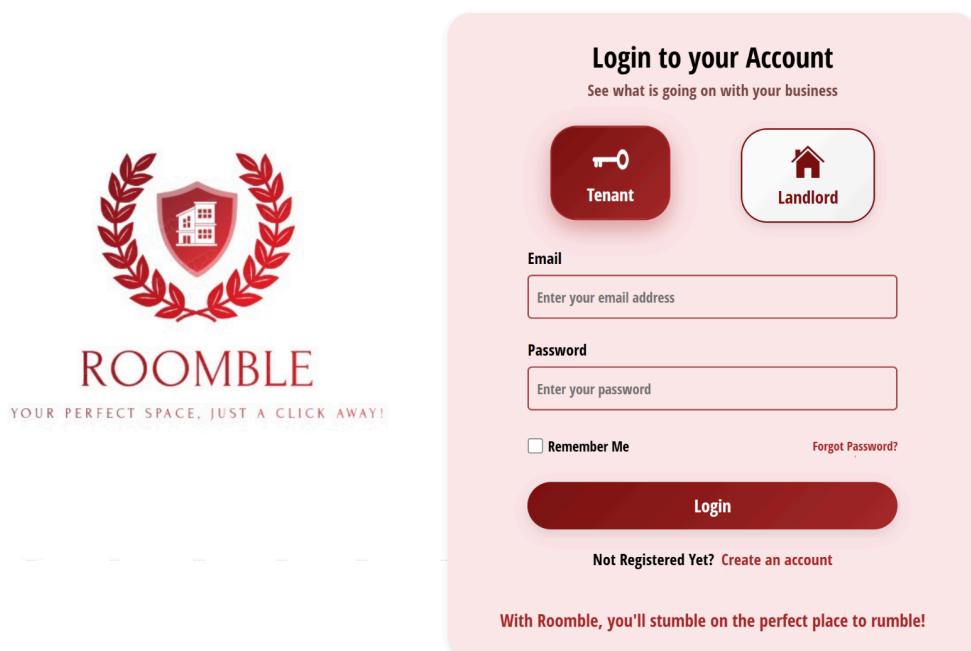
This screenshot is identical to the one above, showing the Roomble landlord sign-up page. The user has entered the same information: Full Name (Bikramjeet Singh), Email Address (bikramjeetsinghgill2015@gmail.com), Password (*****), and Confirm Password (*****). However, a red error message at the bottom of the form states: "Password and Confirm password do not match!". The rest of the page, including the "Sign up" button and the promotional note, remains the same.

If the passwords do not match the user is prompted to correct it.



Afterward, an OTP is sent to the concerned mail of the user and the user is directed to the following page, wherein he/she needs to enter the correct OTP, entering the wrong OTP gives an error message.





Entering the correct OTP, redirects the user to the login page:

2a. Login-Tenant-Backend

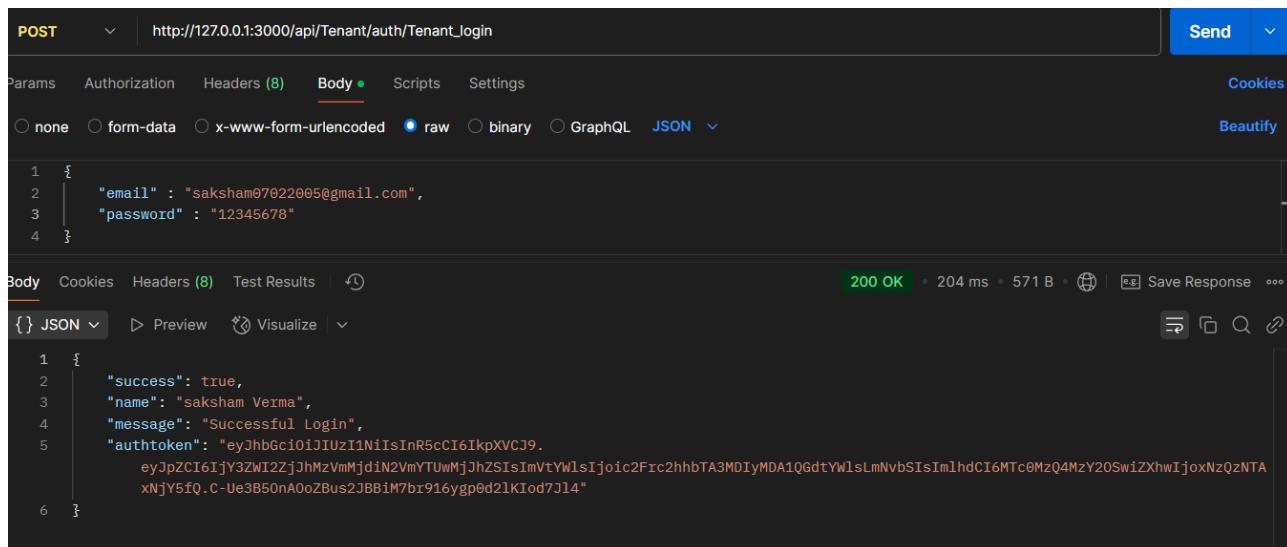
In this unit we have tested the backend function that logs in a Tenant.

Unit Details: Testing of this unit ensures that a valid Tenant is allowed to sign in, if the entered details are correct. It also ensures that an unauthenticated Tenant is not allowed to sign in.

Test Owner: Saksham Verma

Test Date: 01/04/2025

Test Results: An authenticated Tenant(valid details) was allowed to log in successfully. An unauthenticated Tenant was prevented from logging in.



The screenshot shows a POST request to `http://127.0.0.1:3000/api/Tenant/auth/Tenant_login`. The Body tab is selected, showing raw JSON input:

```

1 {
2   "email" : "saksham07022005@gmail.com",
3   "password" : "12345678"
4 }

```

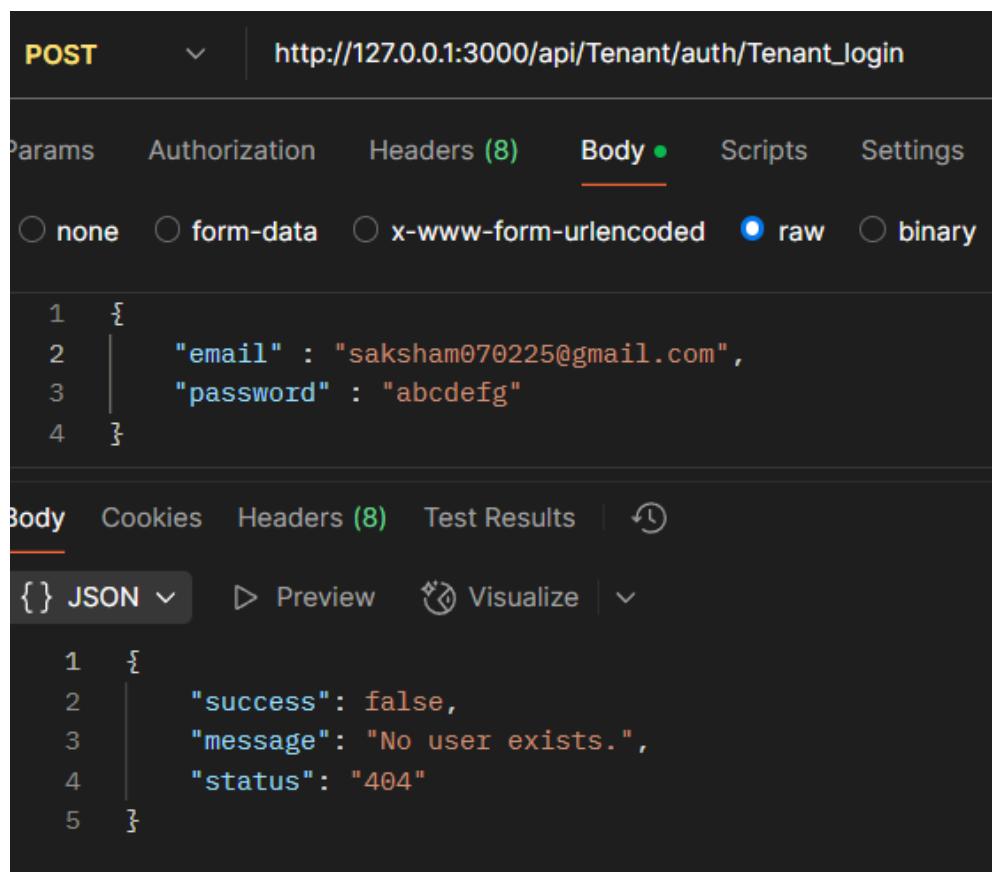
The response status is **200 OK**, with a response time of 204 ms and a body size of 571 B. The response JSON is:

```

1 {
2   "success": true,
3   "name": "saksham Verma",
4   "message": "Successful Login",
5   "authtoken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpY19fQ_C-Ue3B50nA0oZBus2JBBiM7bi916ygpd2KIod7Jl4"
6 }

```

Upon successful login, the tenant receives an Authtoken with success = true



The screenshot shows a POST request to `http://127.0.0.1:3000/api/Tenant/auth/Tenant_login`. The Body tab is selected, showing raw JSON input:

```

1 {
2   "email" : "saksham070225@gmail.com",
3   "password" : "abcdefg"
4 }

```

The response status is **200 OK**, with a response time of 204 ms and a body size of 571 B. The response JSON is:

```

1 {
2   "success": false,
3   "message": "No user exists.",
4   "status": "404"
5 }

```

If the User isn't registered, he receives success = false, with message = "No user exists"

The screenshot shows a Postman test configuration for a POST request to `http://127.0.0.1:3000/api/Tenant/auth/Tenant_login`. The request method is `POST`. The Headers section contains 8 items. The Body section is set to `raw` and contains the following JSON payload:

```

1  {
2    "email" : "saksham07022005@gmail.com",
3    "password" : "abcdefg"
4  }

```

The Response section shows the following JSON output:

```

1  {
2    "status": "401",
3    "message": "Wrong password, entry denied",
4    "success": false
5  }

```

If the User tries to Login with wrong Password, he receives success = `false`, and message = “Wrong password, entry denied”

Structural Coverage: Functional coverage (covers Tenant_Auth), Branch Coverage checks for Tenant Existence and valid credentials

Additional Comments: None

2b. Login-Tenant-Frontend

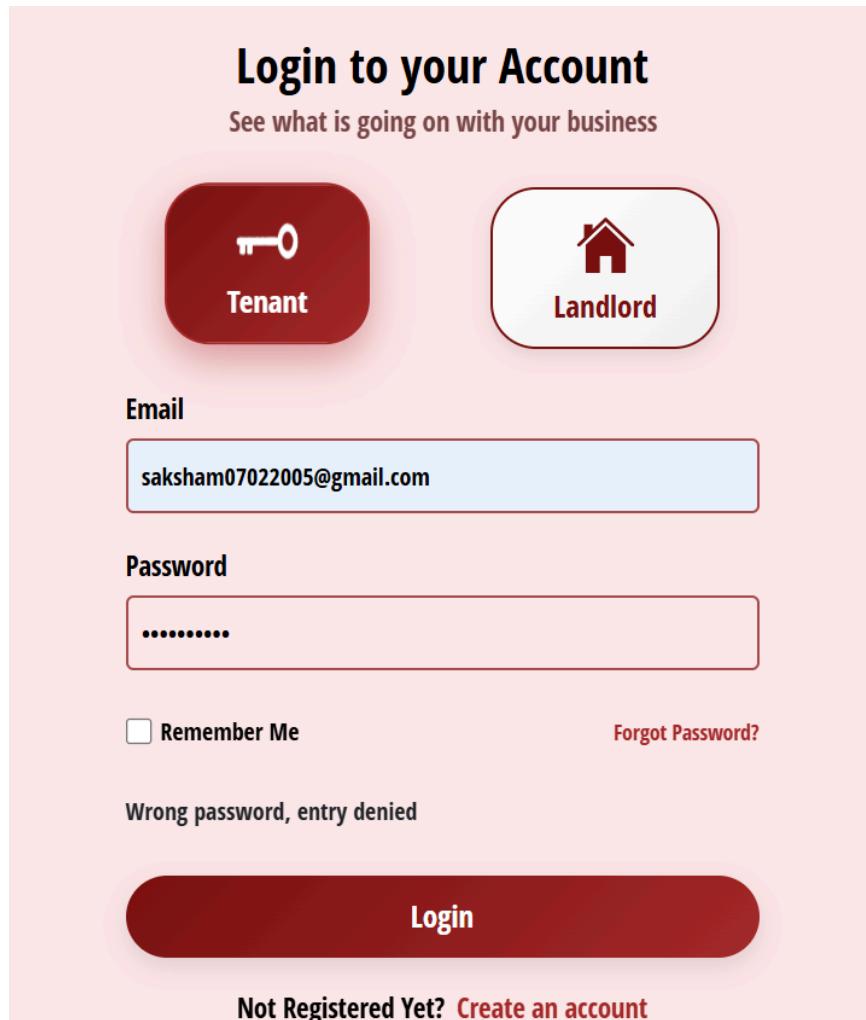
In this unit we have tested the backend function that logs in a Tenant

Unit Details: Testing of this unit ensures that a valid Tenant is allowed to sign in, if the entered details are correct. It also ensures that an unauthenticated Tenant is not allowed to sign in.

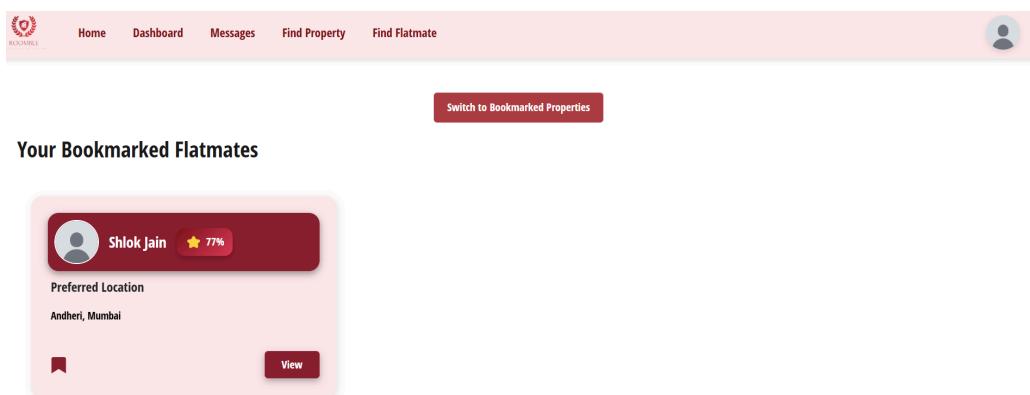
Test Owner: Saksham Verma

Test Date: 01/04/2025

Test Results: An authenticated Tenant(valid details) was allowed to log in successfully. An unauthenticated Tenant was prevented from logging in.



Upon entering wrong password, the Tenant is not allowed to Log in.



Upon entering correct credentials a Tenant is redirected to his Homepage.

The screenshot shows the Roomble login page. The title "Login to your Account" is at the top, followed by the subtext "See what is going on with your business". Below this are two buttons: a red one for "Tenant" with a key icon and a white one for "Landlord" with a house icon. The "Email" field contains the text "saksham07022005@gmail.com". The "Password" field contains several dots, indicating a password has been entered. Below the password field is a checkbox labeled "Remember Me" and a link "Forgot Password?". A message "Wrong password, entry denied" is displayed above the "Login" button. At the bottom, there is a link "Not Registered Yet? Create an account".

Upon entering a wrong Password the Tenant isn't allowed to log in

Structural Coverage: Functional coverage (covers Tenant_Auth), Branch Coverage checks for Tenant Existence and valid credentials

Additional Comments: None

2c. Login-Landlord-Backend

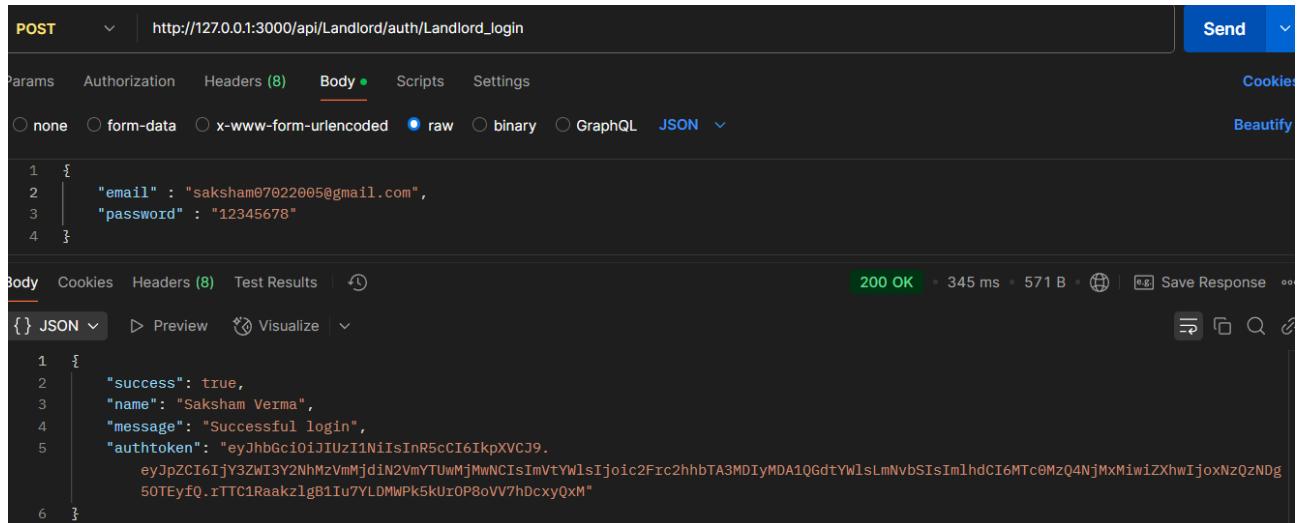
In this unit we have tested the backend function that logs in a Landlord

Unit Details: Testing of this unit ensures that a valid Landlord is allowed to sign in, if the entered details are correct. It also ensures that an unauthenticated Landlord is not allowed to sign in.

Test Owner: Saksham Verma

Test Date: 01/04/2025

Test Results: An authenticated Landlord(valid details) was allowed to log in successfully. An unauthenticated Landlord was prevented from logging in.



POST http://127.0.0.1:3000/api/Landlord/auth/Landlord_login

Params Authorization Headers (8) Body Scripts Settings

Body (raw) JSON

```
1 {
2   "email" : "saksham07022005@gmail.com",
3   "password" : "12345678"
4 }
```

Body Cookies Headers (8) Test Results

200 OK 345 ms 571 B Save Response

{ } JSON Preview Visualize

```
1 {
2   "success": true,
3   "name": "Saksham Verma",
4   "message": "Successful login",
5   "authToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdC16IjY3ZWl3Y2NMZVmMjdIN2vMyTuwMjMwNCIsImVtYWlsIjoic2Frc2hhbTA3MDIyMDA1QGdtYWlsLmNvbSIiMlhCI6MTc0MzQ4NjMxMiwiZXhwIjoxNzQzNDg5OTEyfQ.rTTC1RaakzlgB1Iu7YLDMWPk5kUi0P8oVV7hDcxyQxM"
```

Upon entering correct credentials, a success code of 200 and auth token is returned

The screenshot shows a POST request to `http://127.0.0.1:3000/api/Landlord/auth/Landlord_login`. The Body tab is selected, showing raw JSON input:

```
1 {
2   "email" : "saksham022005@gmail.com",
3   "password" : "145678"
4 }
```

The response status is **404 Not Found**, and the response body is:

```
{ } JSON ▾ ▷ Preview ⚡ Visualize ▾
```

```
1 {
2   "success": false,
3   "message": "No such user exists"
4 }
```

Upon entering an invalid email, a success code of 200 is returned

The screenshot shows a POST request to `http://127.0.0.1:3000/api/Landlord/auth/Landlord_login`. The Body tab is selected, showing raw JSON input:

```
1 {
2   "email" : "saksham07022005@gmail.com",
3   "password" : "145678"
4 }
```

The response status is **401 Unauthorized**, and the response body is:

```
{ } JSON ▾ ▷ Preview ⚡ Visualize ▾
```

```
1 {
2   "status": "401",
3   "message": "Wrong password, entry denied",
4   "success": false
5 }
```

Upon entering wrong password, a success code of 401 is returned

Structural Coverage: Functional coverage (covers Landlord_Auth), Branch Coverage checks for Landlord Existence and valid credentials

Additional Comments: None

2d. Login-Landlord-Frontend

In this unit we have tested the frontend function that logs in a Landlord

Unit Details: Testing of this unit ensures that a valid Landlord is allowed to sign in, if the entered details are correct. It also ensures that an unauthenticated Landlord is not allowed to sign in.

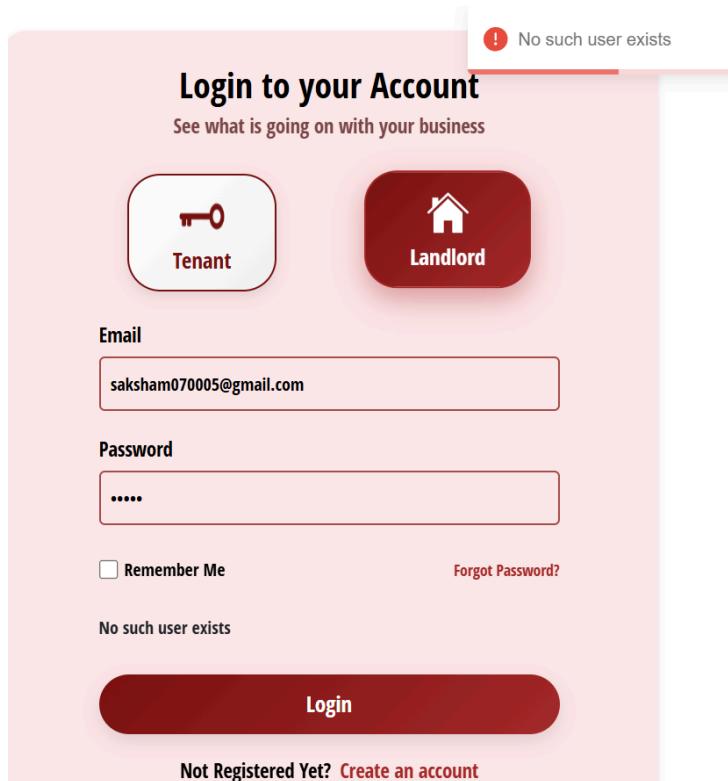
Test Owner: Saksham Verma

Test Date: 01/04/2025

Test Results: An authenticated Landlord(valid details) was allowed to log in successfully. An unauthenticated Landlord was prevented from logging in.

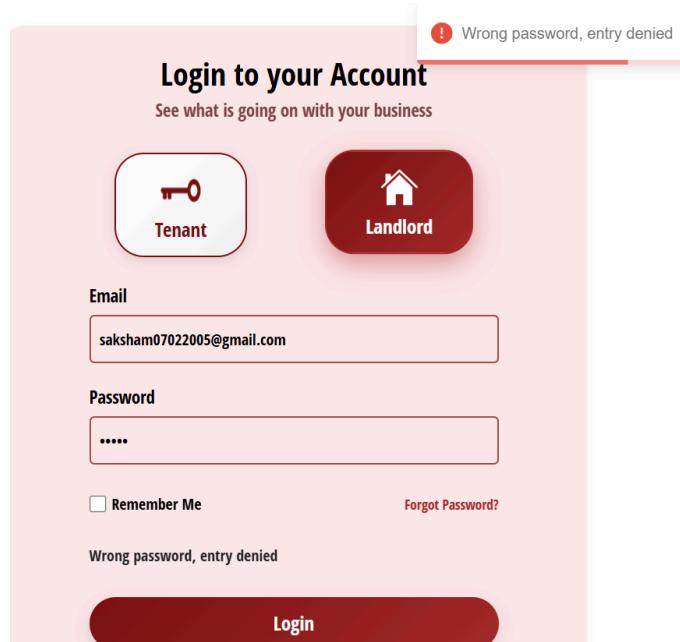


Upon entering correct credentials the Landlord is directed to his Homepage



The screenshot shows the 'Login to your Account' page. At the top right, there is a red error message box with the text 'No such user exists' and an exclamation mark icon. Below the message, the title 'Login to your Account' and the subtitle 'See what is going on with your business' are displayed. There are two buttons: 'Tenant' (white background with a key icon) and 'Landlord' (dark red background with a house icon). The 'Email' field contains 'saksham070005@gmail.com'. The 'Password' field contains '.....'. Below the fields are 'Remember Me' and 'Forgot Password?' links. A message 'No such user exists' is displayed below the login button. The 'Login' button is large and dark red.

Upon entering wrong email , a message showing “No such user exists” is displayed



The screenshot shows the 'Login to your Account' page. At the top right, there is a red error message box with the text 'Wrong password, entry denied' and an exclamation mark icon. Below the message, the title 'Login to your Account' and the subtitle 'See what is going on with your business' are displayed. There are two buttons: 'Tenant' (white background with a key icon) and 'Landlord' (dark red background with a house icon). The 'Email' field contains 'saksham07022005@gmail.com'. The 'Password' field contains '.....'. Below the fields are 'Remember Me' and 'Forgot Password?' links. A message 'Wrong password, entry denied' is displayed below the login button. The 'Login' button is large and dark red.

Upon entering the wrong password, a message showing “Wrong Password” is displayed

Structural Coverage: Functional coverage (covers Landlord_Auth), Branch Coverage checks for Landlord Existence and valid credentials

Additional Comments: None

3a. Forgot Password-Backend

In this unit we have tested the backend route, which allows the user to get a new password in case he/she forgets it.

Unit Details: Testing of this unit ensures that if a user forgets his/her password, he/she can opt for making a new one using his/her registered email ID, and the new password gets updated in the backend.

Test Owner: Bikramjeet Singh

Test Date: 01/04/2025

Test Results: Using this route the password was successfully changes in the backend.

```

POST http://localhost:3000/api/forgotPassword/enteremail
Body (10)
Params Authorization Headers (10) Body Scripts Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON
1 {
2   "email" : "bsingh23@iitk.ac.in",
3   "accounttype" : "tenant"
4 }

```

Body Cookies Headers (8) Test Results

{ } JSON > Preview Visualize

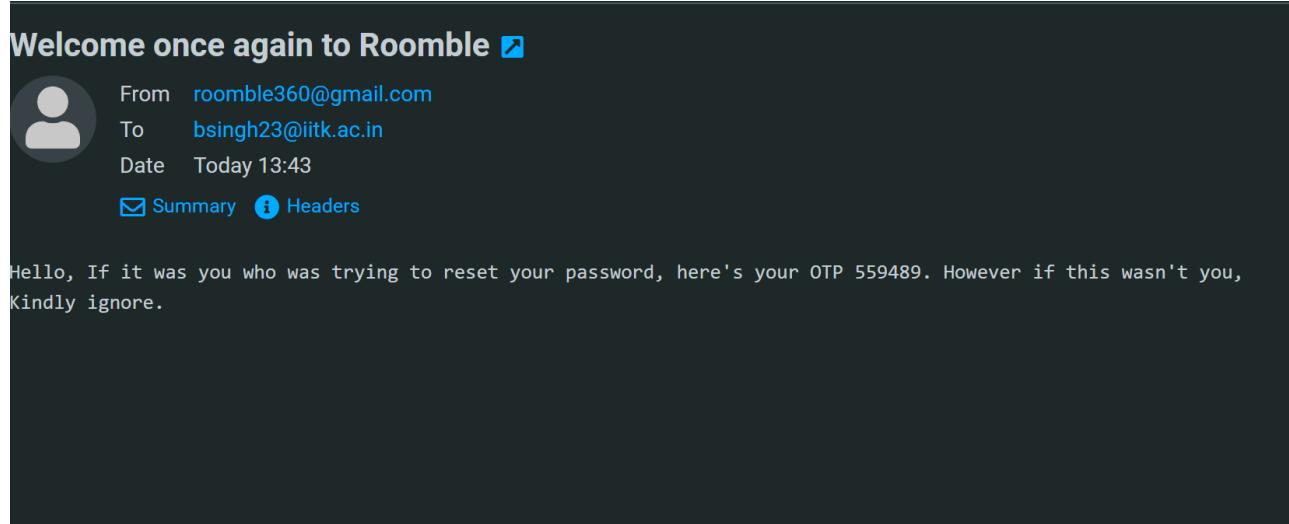
```

1 {
2   "success": true,
3   "authToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
4   eyJpZC16Ijy3ZWI4NTAwOQwYjF1NDNjZjQ5ZmYyYSIsImVtYWlsIjoiYnNpbmdoMjNAaWl0ay5hYy5pbiiSImlhCI6MTc0MzQ5NTIzNSwiZhwIjoxNzQzNDk1NTM1fQ.
v_0DTIwMFzObRia2PxfqWkbWdNX2eEg9zUkzUI2QolI"
}

```

200 OK 3.75 s 510 B

On sending the email and the accounttype in a POST request an OTP is sent to the email and an auth-token is generated for safety concerns.



A screenshot of a POST request in Postman. The URL is `http://localhost:3000/api/forgotPassword/enterOTP`. The Body tab is selected, showing raw JSON input:

```
1 {  
2   "Entered OTP" : "559489",  
3   "accounttype" : "tenant"  
4 }
```

The response status is 200 OK, with a response time of 177 ms, a size of 332 B, and a message: "Correct OTP entered, access granted."

Headers		Value	Description	Bulk Edit	Presets
Key	Value				
<input checked="" type="checkbox"/> authtoken	eyJhbGciOiJIUzI1NiisInR5cCI6IkpXVCJ9.eyJpZCI6IjY3ZWl4NTAwOG...				
Key	Value	Description			

On sending the correct OTP in the body alongwith the previously generated auth-token in the header the access to change the password is granted.

The screenshot shows a Postman interface with a POST request to `http://localhost:3000/api/forgotPassword/ForgotPassword`. The request body is set to raw JSON:

```

1  {
2   "newPassword" : "abcdeñ",
3   "accounttype" : "tenant"
4 }

```

The response status is 200 OK, with a response body indicating success and a message: "Successfully updated".

On sending the new-password in the body, the password gets successfully updated. This is also depicted in the backend as well:

password : `"$2b$10$3X1vjrc/dUOdWy2Ae1WKmOETVR.ANu9Bqxckmw1IyR37Wmsx/v0"`

This is the previously hashed password

password : `"$2b$10$0M.dz1aQMzsGG7.wVVAdef07JvWo0VB70HuytHaptWg65W8v6166"`

This is the new password after changing it through Postman.

Structural Coverage: The above testing majorly addresses the functional coverage part, the branch coverage has been done in the frontend part immediately following this part.

Additional Comments: None

3b. Forgot Password-Frontend

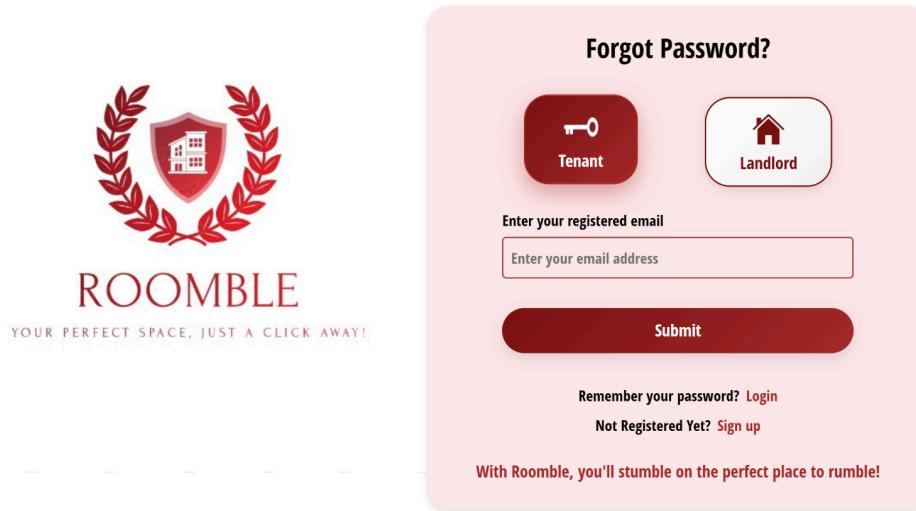
In this unit, we have tested the feature that allows the user to get a new password he/she forgets through the user-friendly interface of our application.

Unit Details: Testing of this unit ensures that if a user forgets his/her password, he /she can easily make a new one using his/her registered email ID.

Test Owner: Bikramjeet Singh

Test Date: 01/04/2025

Test Results: Using this feature a user was able to change his password.



On clicking Forgot password on the Login Page user is redirected here where he/she needs to enter his/her registered email.



ROOMBLE
YOUR PERFECT SPACE, JUST A CLICK AWAY!

Forgot Password?


Tenant


Landlord

Enter your registered email

abc@gmail.com

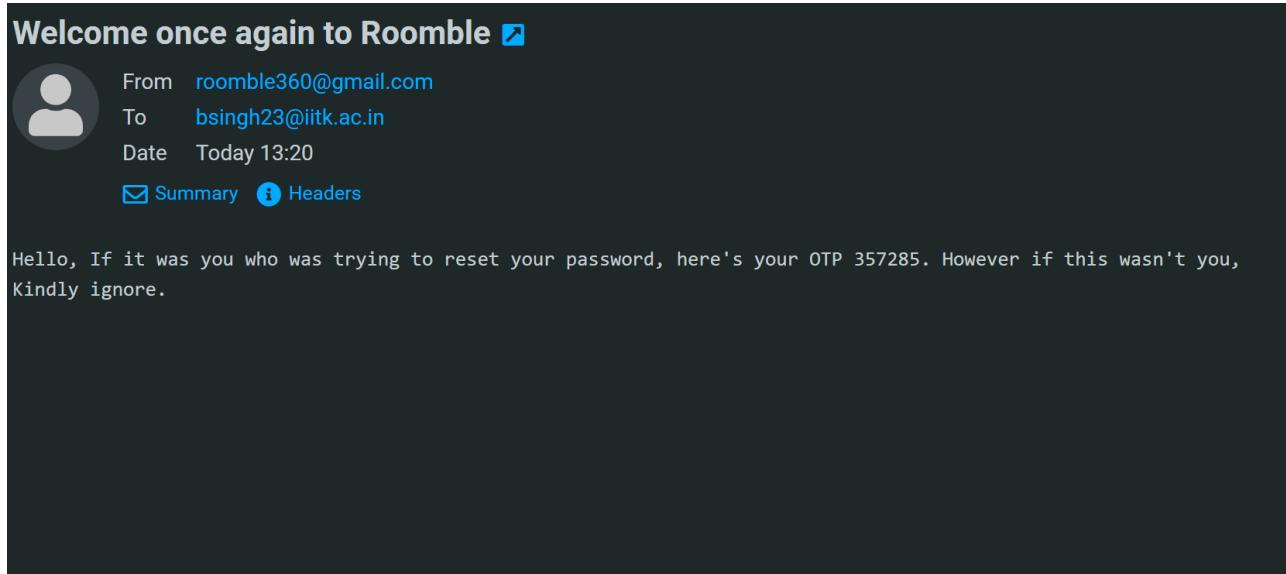
No such user exists

Submit

[Remember your password? Login](#)
[Not Registered Yet? Sign up](#)

With Roomble, you'll stumble on the perfect place to rumble!

If the user is not registered he will get a No such user exists message



Entering the correct email sends an OTP to the concerned email.



ROOMBLE
YOUR PERFECT SPACE. JUST A CLICK AWAY!

Reset Password

Enter New password

Confirm New Password

Submit

Remember your password? [Login](#)

Not Registered Yet? [Sign up](#)

With Roomble, you'll stumble on the perfect place to rumble!

Entering the correct OTP leads to the above page where user can enter and confirm the new password.



ROOMBLE
YOUR PERFECT SPACE. JUST A CLICK AWAY!

localhost:5173 says
Password reset successful!

Set Password

OK

Enter New password
.....

Confirm New Password
.....

Redirecting...

Remember your password? [Login](#)

Not Registered Yet? [Sign up](#)

With Roomble, you'll stumble on the perfect place to rumble!

After doing so, an alert appears on the screen indicating success of the process. After this the user is redirected to the login page where he/she can use his/her new password.

Structural Coverage: Functional Coverage and Branch Coverage for invalid email and OTP.

Additional Comments: None

4a. View Self profile-Backend

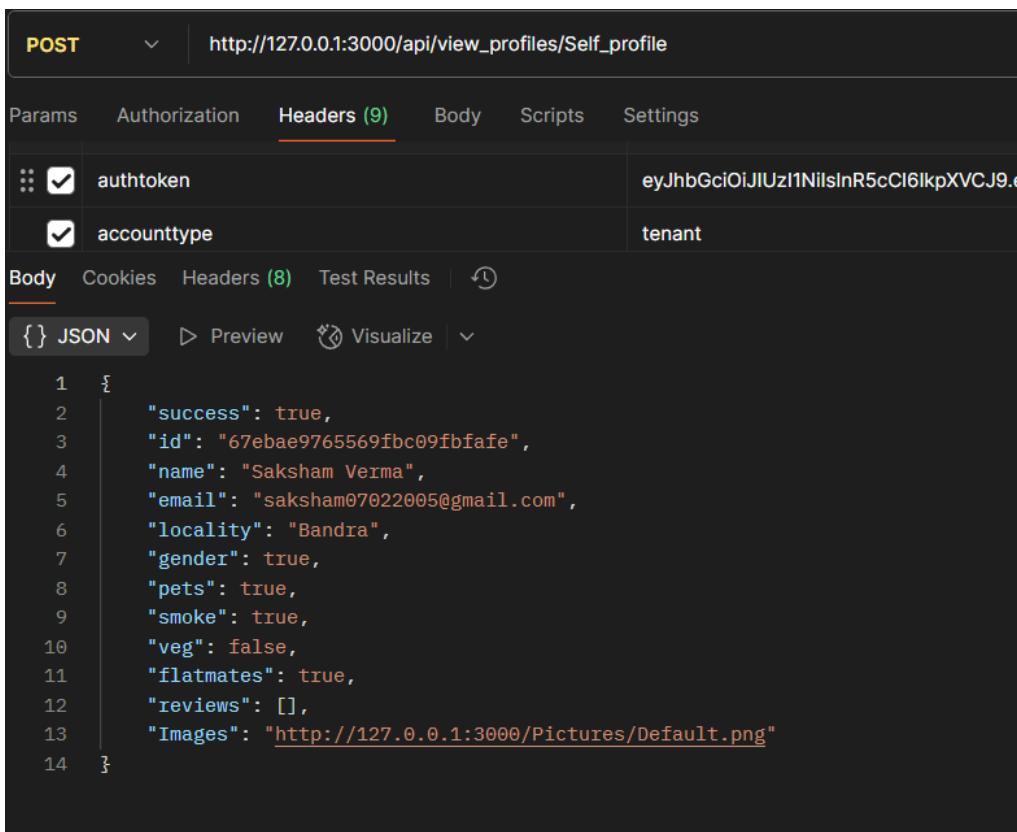
In this unit, we have tested the backend function that allows the user to view his own profile

Unit Details: Testing of this unit ensures that a users profile is fetched correctly to them

Test Owner: Rathod Ayushi

Test Date: 01/04/2025

Test Results: Using this feature a user was able to correctly fetch his own profile.



The screenshot shows a POST request in Postman. The URL is `http://127.0.0.1:3000/api/view_profiles/Self_profile`. The Headers tab is selected, showing two checked fields: `authtoken` with value `eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9.e` and `accounttype` with value `tenant`. The Body tab is selected, showing a JSON response with the following content:

```

1  {
2   "success": true,
3   "id": "67ebae9765569fbc09fbfafe",
4   "name": "Saksham Verma",
5   "email": "saksham07022005@gmail.com",
6   "locality": "Bandra",
7   "gender": true,
8   "pets": true,
9   "smoke": true,
10  "veg": false,
11  "flatmates": true,
12  "reviews": [],
13  "Images": "http://127.0.0.1:3000/Pictures/Default.png"
14 }

```

If a tenant with a valid authtoken sends this request, he gets his own profile

POST | http://127.0.0.1:3000/api/view_profiles/Self_profile

Headers (9)

<input checked="" type="checkbox"/>	authtoken	eyJhbGciOiJIUzI1NilsInR5cCl6IkpxVCJ9eyJpZCI6IjY3Z...
<input checked="" type="checkbox"/>	accounttype	tenant

Body | Cookies | Headers (8) | Test Results | ⏱ 4 ms

400 Bad Request

{ } JSON ▾ | Preview | Visualize | ▾

```

1 {
2   "message": "Invalid Token",
3   "success": false
4 }
```

If a tenant who hasn't logged in or doesn't have a valid authtoken tries to access his profile, he gets a response as above

POST | http://127.0.0.1:3000/api/view_profiles/Self_profile

Headers (9)

<input checked="" type="checkbox"/>	authtoken	eyJhbGciOiJIUzI1NilsInR5cCl6IkpxVCJ9eyJpZCI6IjY3Z...
<input checked="" type="checkbox"/>	accounttype	landlord

Body | Cookies | Headers (8) | Test Results | ⏱ 201 ms | 1.05 KB | 🗑 | 🎯

200 OK

{ } JSON ▾ | Preview | Visualize | ▾

```

1 {
2   "success": true,
3   "name": "Saksham Verma",
4   "email": "saksham07022005@gmail.com",
5   "message": "You own 1 properties.",
6   "Properties": [
7     {
8       "_id": "67ebaf4965569fbc09fbfb2a",
9       "city": "Mumbai",
10      "town": "Andheri",
11      "address": "Andheri East, Near Dutta Pan Palace",
12      "area": 2000,
13      "bhk": 2,
14      "description": "Very good place to live",
15      "amenities": [
16        "Gym",
17        "Pan Palace",
18        "Swimming Pool",
19        "Bathroom"
20      ],
21      "price": 20000,
22      "available": true,
23      "Images": [
24        "http://localhost:3000/Pictures/property/67ebaf4965569fbc09fbfb2a/0_inet"
25      ]
26    }
27  ]
28 }
```

If a landlord with a valid authtoken sends this request, he gets his own profile

The screenshot shows a POST request in Postman. The URL is `http://127.0.0.1:3000/api/view_profiles/Self_profile`. In the Headers tab, there are two entries: `authtoken` with value `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3Z...` and `accounttype` with value `landlord`. The response status is `400 Bad Request`, and the response body is a JSON object:

```
1 {  
2     "message": "Invalid Token",  
3     "success": false  
4 }
```

If a landlord who hasn't logged in or doesn't have a valid authtoken tries to access his profile, he gets a response as above

Structural Coverage: Functional Coverage and Branch Coverage on whether a user has logged in.

Additional Comments: None

4b. View Self profile-Frontend

In this unit, we have tested the frontend function that allows the user to view his own profile

Unit Details: Testing of this unit ensures that a users profile is shown correctly to them

Test Owner: Saksham Verma

Test Date: 01/04/2025

Test Results: Using this feature a user was able to correctly view his own profile.

The screenshot shows the 'Edit Profile' section of the Roomble application. At the top, there is a placeholder for a user's profile picture, accompanied by the message 'You have not set up a description yet'. Below this, a table lists various profile details:

Name	:	Saksham Verma
Email	:	saksham07022005@gmail.com
City	:	Mumbai
Locality	:	Bandra
Gender	:	Male ♂
Are you seeking a flatmate?	:	Yes ✓
Do you drink/smoke?	:	Yes ♀
Food Preferences	:	Non-Veg 🍗
Do you have pets?	:	Yes 🐾

At the bottom right are two buttons: 'Edit' and 'Logout'.

If a tenant is logged in he can correctly view his profile

The screenshot shows the 'View Profile' section of the Roomble application. It displays basic user information:

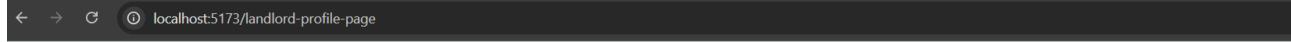
Full Name : Saksham Verma
Email Address : saksham07022005@gmail.com
Properties Count : You own 1 properties.

Below this, there is a thumbnail image of a house and detailed property information:

Price : 20000
Andheri, Mumbai
BHK : 2

A 'View' button is located at the bottom right of the property card.

If a tenant is logged in, he can correctly view his profile



This is the output shown to a user who isn't logged in, but tries to view his profile

Structural Coverage: Functional Coverage and Branch Coverage on whether a user has logged in.

Additional Comments: None

5a. View Others profile-Backend

In this unit, we have tested the frontend function that allows the user to view his own profile

Unit Details: Testing of this unit ensures that a user's profile is shown correctly to other users

Test Owner: Saksham Verma

Test Date: 01/04/2025

Test Results: Using this feature a user was able to correctly view other profiles.

POST http://127.0.0.1:3000/api/view_profiles/other_users

Params Authorization Headers (10) Body Scripts Settings

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

```

1  {
2    "accounttype" : "tenant",
3    "requested_id" : "67ebb688a115332bbd459875"

```

Body Cookies Headers (8) Test Results 200 OK

{ } JSON ▾ ▷ Preview ⚡ Visualize ▾

```

1  {
2    "success": true,
3    "id": "67ebb688a115332bbd459875",
4    "name": "Shlok Jain",
5    "city": "Mumbai",
6    "locality": "Andheri",
7    "gender": true,
8    "pets": false,
9    "smoke": false,
10   "veg": true,
11   "flatmates": true,
12   "reviews": [],
13   "Images": "http://127.0.0.1:3000/Pictures/Default.png",
14   "description": "This user hasn't setup a description yet"
15 }

```

If the correct ID of a tenant / landlord is entered, their profile is retrieved

POST http://127.0.0.1:3000/api/view_profiles/other_users

Params Authorization Headers (10) Body Scripts Settings

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

```

2  {
3    "accounttype" : "tenant",
4    "requested_id" : "67ebb688a115332bbd452875"
5  }

```

Body Cookies Headers (8) Test Results 404 Not Found

{ } JSON ▾ ▷ Preview ⚡ Visualize ▾

```

1  {
2    "success": false,
3    "message": "No such user exists."
4  }

```

If a wrong ID of tenant/landlord is entered, a message displaying “No such user exists”

```

POST http://127.0.0.1:3000/api/view_profiles/other_users

Params Authorization Headers (10) Body Scripts Settings
Body raw binary GraphQL JSON
2 "accounttype" : "tenant",
3 "requested_id" : "67ebb685332bbd452875"
4 }

Body Cookies Headers (8) Test Results 400 Bad Request
[{"error": "67ebb685332bbd452875 is invalid ID"}]
  
```

If an invalid ID is entered(MongoDB follows certain ID syntaxes) a status code of 400 is shown.

Structural Coverage: Functional Coverage and Branch Coverage on wrong ID being asked, ID not existing in backend and a valid ID entered.

Additional Comments: None

6. Navbar

In this unit, we have tested the frontend function that allows the user to navigate to different pages through Navbar.

Unit Details: Testing of this unit ensures that a user can move through different pages using Navbar.

Test Owner: Ronav Puri

Test Date: 01/04/2025

The screenshot shows a user interface for managing bookmarked flatmates. At the top, there is a navigation bar with links for Home, Dashboard, Messages, Find Property, and Find Flatmate. On the right side of the header is a user profile icon. Below the header, a button labeled "Switch to Bookmarked Properties" is visible. The main content area is titled "Your Bookmarked Flatmates". It displays two entries for bookmarked flatmates, each in a card-like format:

- Saksham Verma** (41% completion) - Preferred Location: Bandra, Mumbai. Includes a "View" button.
- Shlok Jain** (34% completion) - Preferred Location: Andheri, Mumbai. Includes a "View" button.

The Navbar for Tenant has the following options.

The screenshot shows a user interface for landlords. At the top, there is a navigation bar with links for Home, Dashboard, Add Property, and Messages. On the right side of the header is a user profile icon. Below the header, the title "Your Properties" is underlined, followed by the message "No properties to show".

The Navbar for Landlord has the following options.

Test Results: Clicking on the Navbar options successfully redirects the user to different pages, and the NavBar options change for different types of users.

Structural Coverage: Functional coverage and Branch coverage on fetching relevant details to be displayed on the navbar based on the usertype and page elements.

Additional Comments: None

7. Add-Property-Backend

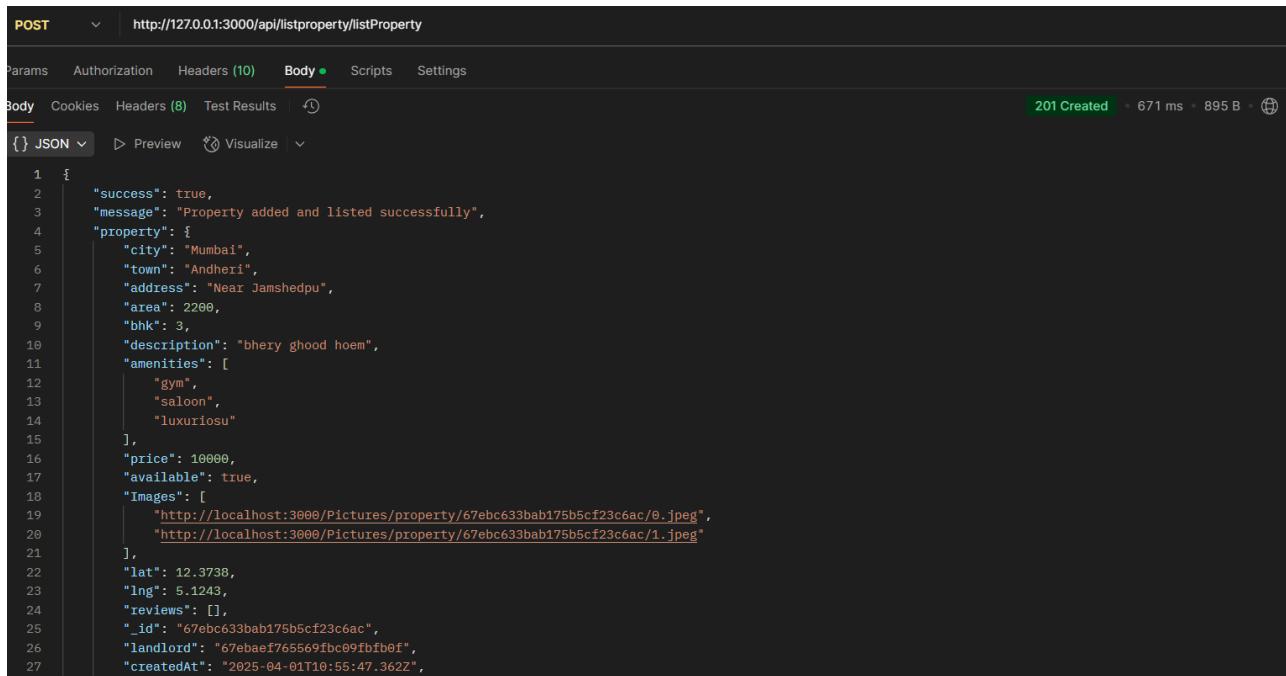
In this unit, we have tested the backend function which enables a landlord to add properties for renting.

Unit Details: Testing of this unit ensures that a landlord can successfully list a property.

Test Owner: Saksham Verma

Test Date: 01/04/2025

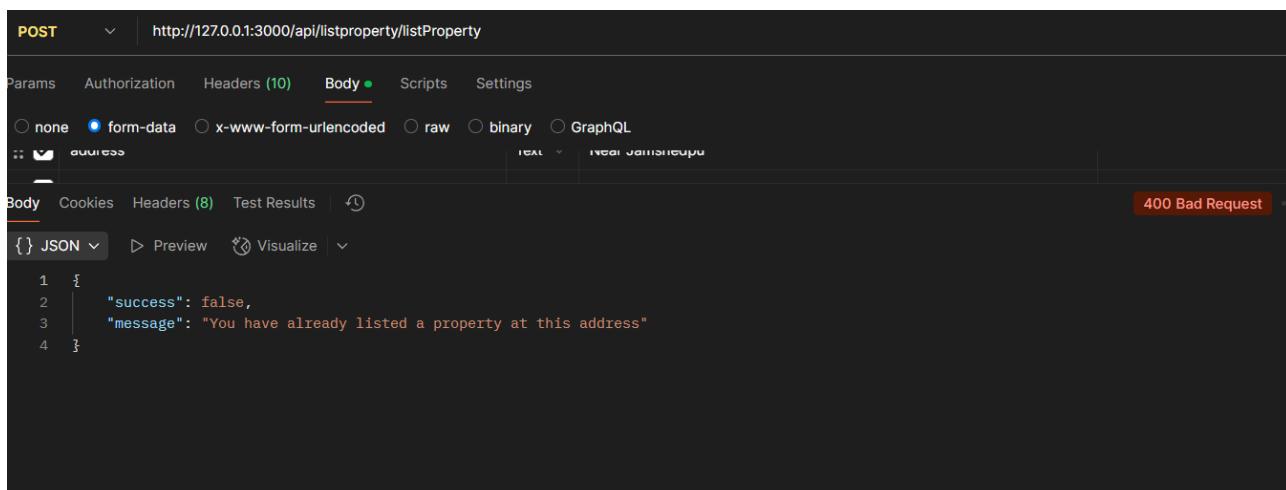
Test Results: Using this test, the software ensures that a landlord is able to add properties and also throw an error whenever something unexpected is given to it.



```

POST http://127.0.0.1:3000/api/listproperty/listProperty
{
    "success": true,
    "message": "Property added and listed successfully",
    "property": {
        "city": "Mumbai",
        "town": "Andheri",
        "address": "Near Jamshedpur",
        "area": 2200,
        "bhk": 3,
        "description": "bhergy ghood hoem",
        "amenities": [
            "gym",
            "saloon",
            "luxurious"
        ],
        "price": 10000,
        "available": true,
        "Images": [
            "http://localhost:3000/Pictures/property/67ebc633bab175b5cf23c6ac/0.jpeg",
            "http://localhost:3000/Pictures/property/67ebc633bab175b5cf23c6ac/1.jpeg"
        ],
        "lat": 12.3738,
        "lng": 5.1243,
        "reviews": [],
        "_id": "67ebc633bab175b5cf23c6ac",
        "landlord": "67ebeaf765569fbc09fbfb0f",
        "createdAt": "2025-04-01T10:55:47.362Z",
        "updatedAt": "2025-04-01T10:55:47.362Z"
    }
}
  
```

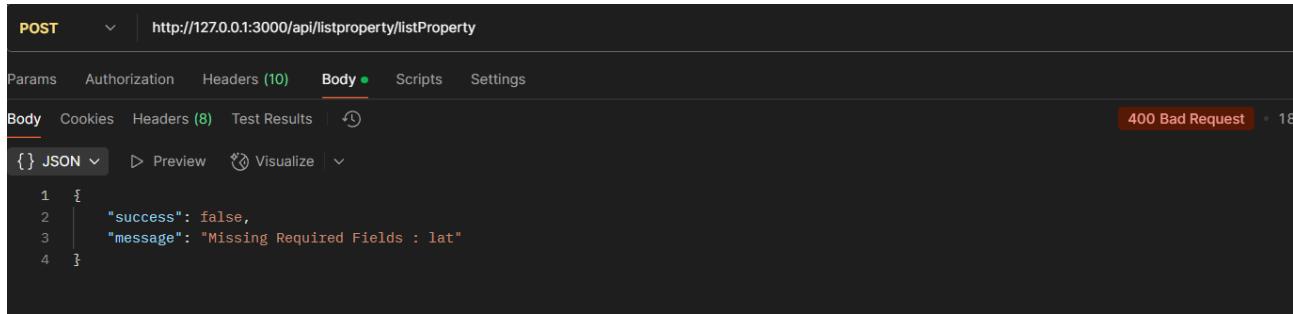
A user can successfully list properties by entering all given fields.



```

POST http://127.0.0.1:3000/api/listproperty/listProperty
{
    "success": false,
    "message": "You have already listed a property at this address"
}
  
```

If a landlord tries to add 2 properties at the same address, then the software throws an error



POST http://127.0.0.1:3000/api/listproperty/listProperty

Params Authorization Headers (10) Body Scripts Settings

Body Cookies Headers (8) Test Results

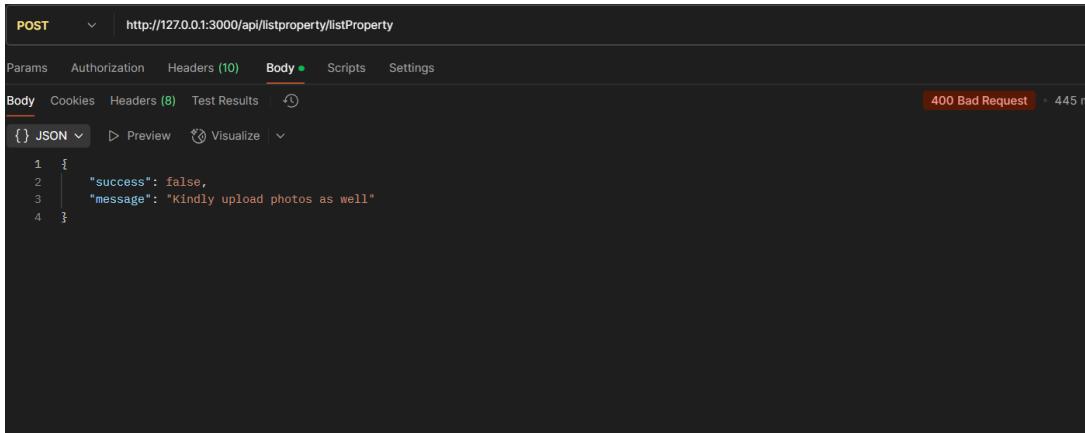
400 Bad Request 18

```
{ } JSON ▾ D Preview ⚡ Visualize ▾
```

```

1  {
2   "success": false,
3   "message": "Missing Required Fields : lat"
4 }
```

If a landlord misses to fill any fields then a message is displayed indicating the missing fields



POST http://127.0.0.1:3000/api/listproperty/listProperty

Params Authorization Headers (10) Body Scripts Settings

Body Cookies Headers (8) Test Results

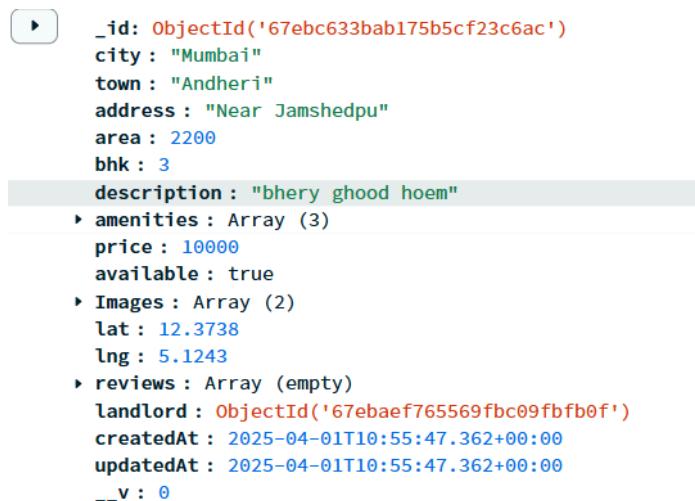
400 Bad Request 445 ms

```
{ } JSON ▾ D Preview ⚡ Visualize ▾
```

```

1  {
2   "success": false,
3   "message": "Kindly upload photos as well"
4 }
```

If a landlord uploads a property without Images, a message is displayed, prompting the landlord to upload Images



```

▶ _id: ObjectId('67ebc633bab175b5cf23c6ac')
  city : "Mumbai"
  town : "Andheri"
  address : "Near Jamshedpu"
  area : 2200
  bhk : 3
  description : "bhery ghodd hoem"
  ▶ amenities : Array (3)
    price : 10000
    available : true
  ▶ Images : Array (2)
    lat : 12.3738
    lng : 5.1243
  ▶ reviews : Array (empty)
  landlord : ObjectId('67ebaef765569fbc09fbfb0f')
  createdAt : 2025-04-01T10:55:47.362+00:00
  updatedAt : 2025-04-01T10:55:47.362+00:00
  __v : 0

```

Upon successful property addition a new document was created in our database

Structural Coverage: Functional Coverage and Branch Coverage on whether all details are filled correctly while adding a property, Images Uploaded, Image size and number of Images uploaded and whether property already exists.

Additional Comments: None

8. Bookmarks-Backend

In this unit, we have tested the backend function which enables a Tenant to bookmark and unmark other flatmates and Properties.

Unit Details: Testing of this unit ensures that a tenant can successfully bookmark and unmark a property and flatmate

Test Owner: Saksham Verma

Test Date: 01/04/2025

Test Results: Using this test, the software ensures that a tenant will be able to bookmark properties and other flatmates and also throw an error whenever a wrong ID is tried sent to backend for processing.

```

POST http://127.0.0.1:3000/api/BookMarking_Routes/edit_bookmarks

Params Authorization Headers (10) Body Scripts Settings
 none  form-data  x-www-form-urlencoded  raw  binary  Graphql

1 {
2   "action" : "bookmark",
3   "id" : "123",
4   "thing" : "property"
5 }

Body { } JSON ▾ Preview Visualize ▾
1 {
2   "error": "123 is invalid ID"
3 }
  
```

400 Bad Request • 22

An error is thrown whenever a wrong ID is entered

POST http://127.0.0.1:3000/api/BookMarking_Routes/edit_bookmarks

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

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

```

1  {
2    "action" : "bookmark",
3    "id" : "67ebc633bab175b5cf23c6ac",
4    "thing" : "property"
5 }
```

Body 200 OK • 296 ms

{ } JSON Preview Visualize

```

1  {
2    "success": true,
3    "message": "Successfully added property to your Bookmarks"
4 }
```

The user can successfully add bookmarks to the property by entering correct property id

```

pets : true
flatmate : true
description : "This user hasn't setup a description yet"
  ▶ conversations : Array (2)
  ▶ bookmarks_tenants : Array (empty)
  ▶ bookmarks_property : Array (1)
    0: "67ebaf4965569fbc09fbfb2a"
    Images : "http://127.0.0.1:3000/Pictures/Default.png"
  ▶ reviews : Array (empty)
```

Upon Successful addition the Bookmark is added to our database and also a successful response is given to the client

The screenshot shows the Compass API testing interface. A POST request is made to `http://127.0.0.1:3000/api/BookMarking_Routes`. The request body is JSON, containing:

```
1 {  
2   "action": "unmark",  
3   "id": "67ebb688a115332bbd459875",  
4   "thing": "flatmate"  
5 }
```

The response is 200 OK, with a duration of 129 ms and a size of 317 B. The response body is:

```
{ } JSON ▾
```

```
1 {  
2   "success": true,  
3   "message": "Successfully unmarked"  
4 }
```

On the right side, there is a sidebar with user information and a query editor.

User Information (Saksham Verma):

```
name : Saksham Verma  
type : "tenant"  
email : "saksham07022005@gmail.com"  
password : "$2b$10$xM6B8f.lsir3KA.xRvdX0uo/eEJnq16M3/aCHxzC"  
locality : "Bandra"  
city : "Mumbai"  
gender : true  
smoke : true  
veg : false  
pets : true  
flatmate : true  
description : "This user hasn't setup a description yet"  
true  
conversations : Array (2)  
bookmarks_tenants : Array (empty)  
bookmarks_property : Array (empty)  
Images : "http://127.0.0.1:3000/Pictures/Default.png"  
reviews : Array (empty)
```

Query Editor:

```
Type a query: { field: 'value' }
```

Upon successful Unmarking of a flatmate a success code of 200 is returned and also removed from our database

POST http://127.0.0.1:3000/api/BookMarking_Routes/edit_bookmarks

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

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

```
1 {  
2   "action" : "unmark",  
3   "id" : "67ebb688a115332bbd459875",  
4   "thing" : "flatmate"  
5 }
```

Body 404 Not Found • 1

{ } JSON Preview Visualize

```
1 {  
2   "success": true,  
3   "message": "No such bookmark exists"  
4 }
```

If someone tries to unmark a flatmate who isn't bookmarked in the first place, status code of 404 is returned along with a message which says the same

POST http://127.0.0.1:3000/api/BookMarking_Routes/edit_bookmarks

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

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

```

1  {
2      "action" : "bookmark",
3      "id" : "67ebc633bab175b5cf23c6ac",
4      "thing" : "property"
5 }
```

Body 200 OK • 296 ms

JSON

```

1  {
2      "success": true,
3      "message": "Successfully added property to your Bookmarks"
4 }
```

```

    pets : true
    flatmate : true
    description : "This user hasn't setup a description yet"
    ▶ conversations : Array (2)
    ▶ bookmarks_tenants : Array (empty)
    ▶ bookmarks_property : Array (1)
        0: "67ebaf4965569fbc09fbfb2a"
        Images : "http://127.0.0.1:3000/Pictures/Default.png"
    ▶ reviews : Array (empty)
```

Successfully bookmarking a property

```

_id: ObjectId('67ebae9765569fbc09fbfate')
name : "Saksham Verma"
type : "tenant"
email : "saksham07022005@gmail.com"
password : "$2b$10$xM6B8f.lsir3KA.xRvdX0uo/eEJnq16M3/aCHxzq1/l1bWUyN8K72"
locality : "Bandra"
city : "Mumbai"
gender : true
smoke : true
veg : false
pets : true
flatmate : true
description : "This user hasn't setup a description yet"
conversations : Array (2)
bookmarks_tenants : Array (empty)
bookmarks_property : Array (empty)
Images : "http://127.0.0.1:3000/Pictures/Default.png"
reviews : Array (empty)

```

POST http://127.0.0.1:3000/api/BookMarking_Routes/edit_bookmarks

Params	Authorization	Headers (10)	Body	Scripts	Settings
<input type="radio"/> none	<input type="radio"/> form-data	<input type="radio"/> x-www-form-urlencoded	<input checked="" type="radio"/> raw	<input type="radio"/> binary	<input type="radio"/> GraphQL
<pre> 1 { 2 "action" : "unmark", 3 "id" : "67ebaf4965569fbc09fbfb2a", 4 "thing" : "property" 5 }</pre>					
Body	↻		200 OK	• 152 ms	
[] JSON ▼ ▶ Preview ⟳ Visualize ▼ <pre> 1 { 2 "success": true, 3 "message": "Successfully unmarked the property" 4 }</pre>					

Upon successful Unmark of property a successful response is sent to client and the property is also removed from the database

POST | http://127.0.0.1:3000/api/BookMarking_Routes/edit_bookmarks

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

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

```

1  {
2    "action" : "unmark",
3    "id" : "67ebc633bab175b5cf23c6ac",
4    "thing" : "property"
5 }
```

Body 404 Not Found • 9

{ } JSON ▾ | ▷ Preview | ⚡ Visualize | ▾

```

1  {
2    "success": false,
3    "message": "No such property is bookmarked"
4 }
```

Trying to unmark a property which isn't bookmarked in the first place

POST | http://127.0.0.1:3000/api/BookMarking_Routes/edit_bookmarks

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

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

```

1  {
2    "action" : "bookmark",
3    "id" : "67ebb688a115332bbd459825",
4    "thing" : "property"
5 }
```

Body | Cookies | Headers (8) | Test Results | 404 Not Found

{ } JSON ▾ | ▷ Preview | ⚡ Visualize | ▾

```

1  {
2    "success": false,
3    "message": "No such Property exists"
4 }
```

```

POST http://127.0.0.1:3000/api/BookMarking_Routes/edit_bookmarks
Params Authorization Headers (10) Body Scripts Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON
1 {
2   "action" : "bookmark",
3   "id" : "67ebb688a115332bbd459825",
4   "thing" : "flatmate"
5 }

Body Cookies Headers (8) Test Results 404 Not Found
[] JSON Preview Visualize
1 {
2   "success": false,
3   "message": "No such Tenant exists"
4 }

```

The system always check whether a Flatmate or Property exists with the entered ID, if not an error message is returned with a success code of 404

Structural Coverage: The branch coverage covers the cases when a property which doesn't exists in bookmarks is trying to be unmarked, Wrong syntax of ID, An ID which doesn't exists.

Additional Comments: None

10. Interested in a Property- Backend

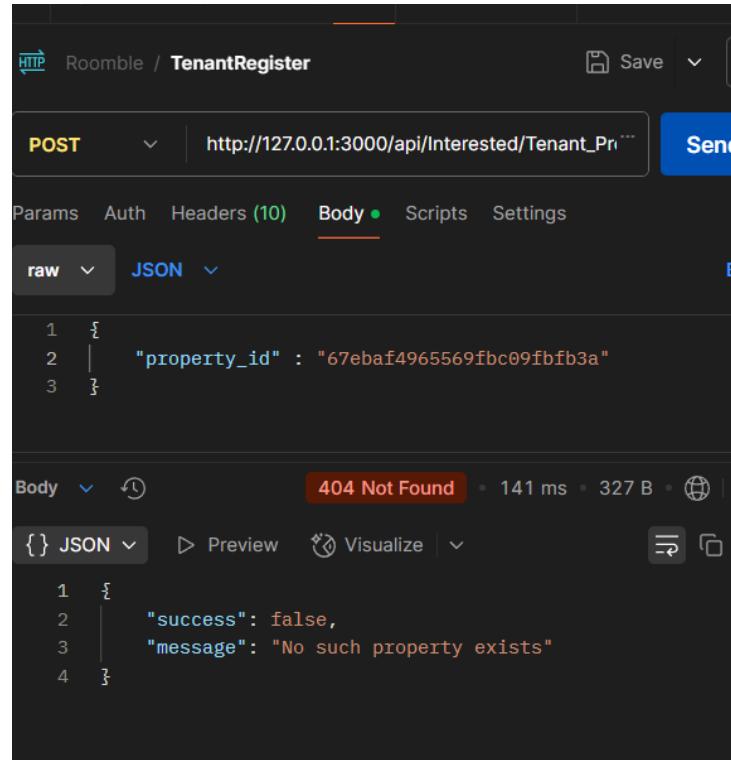
In this section we have tested the backend function that notifies a Landlord whenever a Tenant is interested in his Property.

Unit Details: Testing of this unit ensures that the software can successfully notify a Landlord via mail, whenever a Tenant invokes this function.

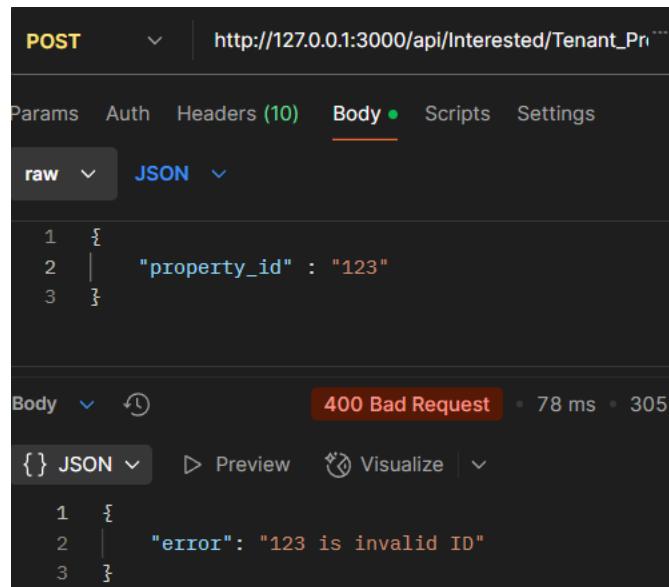
Test Owner: Saksham Verma

Test Date : 03/04/2025

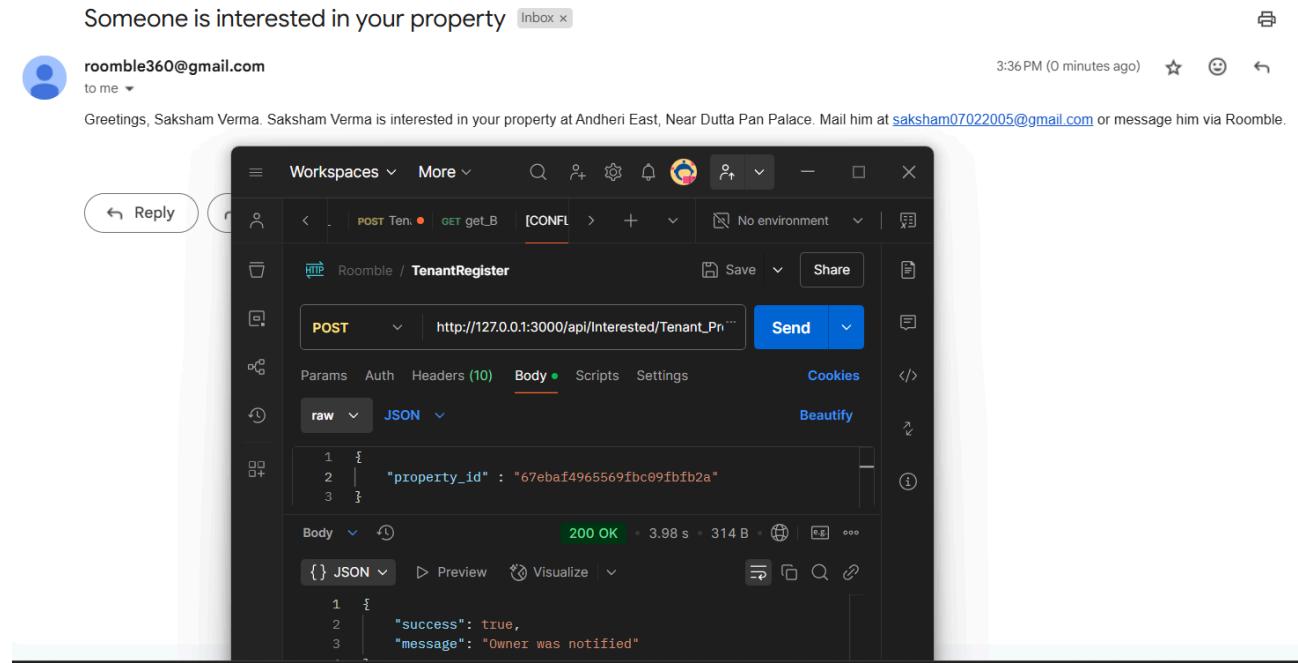
Test Results: Using this functionality the software was successfully able to notify the Landlord via mail whenever a Tenant invoked this function.



The software returns a status code of 404 whenever a wrong Property ID is entered which doesn't exists



The software returns a status code of 400 (Bad request) whenever an Invalid Property ID is entered (Syntax wise)



The owner of the property is notified if the property ID is correct and a status code of 200 is returned

Structural Coverage: This testing took into consideration the branch coverage since both the branches of whether the property exists or not were covered.

Additional Comments: None

11. Enlisting and Delisting a Property- Backend

In this section we have tested the backend function that enables Landlord to successfully enlist and delist a property.

Unit Details: Testing of this unit ensures that a Landlord can successfully enlist and delist his properties.

Test Owner: Saksham Verma

Test Date : 03/04/2025

Test Results: Using this functionality a Landlord was successfully able to Delist and enlist his properties and the software restricted People from enlisting / Delisting other peoples properties

The image contains two vertically stacked screenshots of the Postman API testing application.

Screenshot 1 (Top): A POST request is made to `http://127.0.0.1:3000/api/Listing_Delisting>List...`. The Body tab shows a JSON payload with the following content:

```
1 {  
2   "action" : "Enlist",  
3   "list_id" : "123"
```

The response status is **400 Bad Request**, with a response time of 68 ms and a body size of 345 B. The response JSON is:

```
{ } JSON ▾ 400 Bad Request Preview Visualize  
1 {  
2   "success": false,  
3   "message": "Action can only be 'enlist' or 'delist'"  
4 }
```

Screenshot 2 (Bottom): A POST request is made to `http://127.0.0.1:3000/api/Listing_Delisting>List...`. The Body tab shows a JSON payload with the following content:

```
{  
  "action" : "enlist",  
  "list_id" : "123"
```

The response status is **400 Bad Request**, with a response time of 49 ms and a body size of 323 B. The response JSON is:

```
JSON ▾ 400 Bad Request Preview Visualize  
1 {  
2   "success": false,  
3   "message": "Invalid ID format"  
4 }
```

A status code of 400(Bad request) is sent whenever the user doesn't enter a valid ID or a valid Action

POST | http://127.0.0.1:3000/api/Listing_Delisting>Listings | Send

Params Auth Headers (10) Body Scripts Settings

raw JSON

```

2   "action" : "enlist",
3   "property_id" : "67ebaf4965569fbc09fbfb3a"
4 }
```

Body 401 Unauthorized • 232 ms • 343 B

{ } JSON ▾ Preview Visualize

```

1 {
2   "success": false,
3   "message": "This property Doesn't belong to you."
4 }
```

A status code of 401 is sent if the property doesn't exists in the Landlords_properties Array

http://127.0.0.1:3000/api/Listing_Delisting>Listings | Send

Auth Headers (10) Body Scripts Settings Cookies

JSON

```

"action" : "enlist",
"property_id" : "67ebaf4965569fbc09fbfb2a"
```

200 OK • 263 ms • 326 B

ON ▾ Preview Visualize

```

{
  "success": true,
  "message": "Successfully enlisted property"
}
```

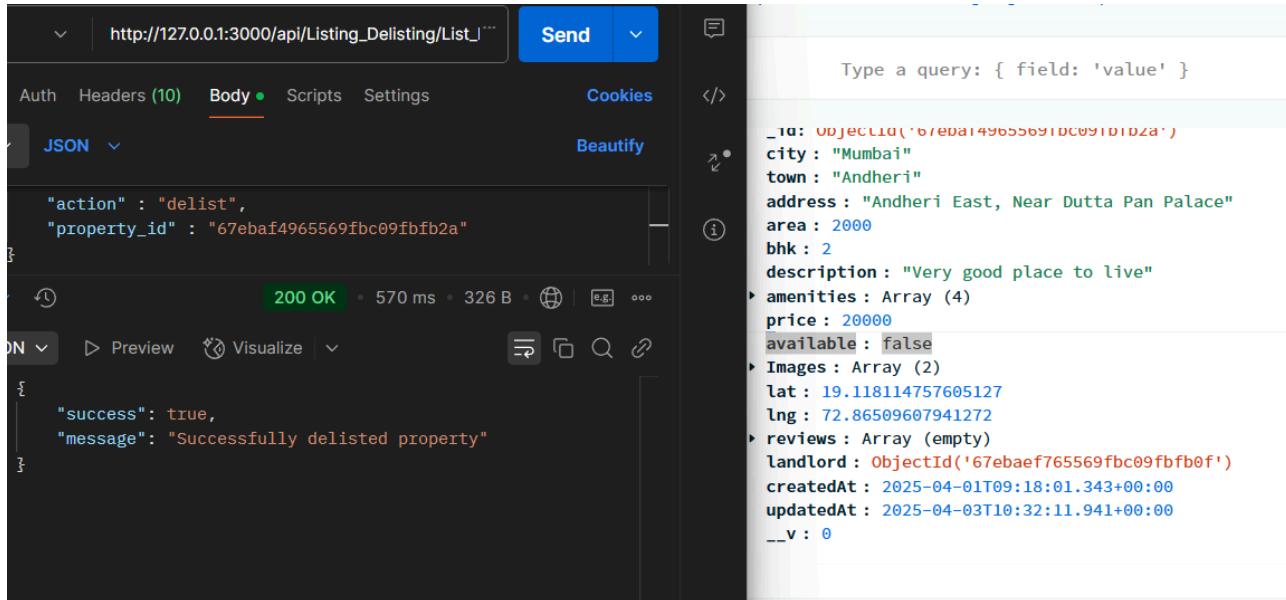
queries from natural language in Compass Type a query: { field: 'value' }

RESULTS: 1-4 OF 4

- `_id: ObjectId('67ebaf4965569fbc09fbfb2a')`
- `city: "Mumbai"`
- `town: "Andheri"`
- `address: "Andheri East, Near Dutta Pan Palace"`
- `area: 2000`
- `bhk: 2`
- `description: "Very good place to live"`
- `amenities: Array (4)`
- `price: 20000`
- `available: true`
- `Images: Array (2)`
- `lat: 19.118114757605127`
- `lng: 72.86509607941272`
- `reviews: Array (empty)`
- `landlord: ObjectId('67ebaef765569fbc09fbfb0f')`

Console Postbot Runner Vault

If the property is owned by him, A status code of 200 is returned and the Available attribute is marked as true



```

{
  "success": true,
  "message": "Successfully delisted property"
}

```

```

_type: ObjectId('67ebaef765569fbc09fbfb2a')
city: "Mumbai"
town: "Andheri"
address: "Andheri East, Near Dutta Pan Palace"
area: 2000
bhk: 2
description: "Very good place to live"
amenities: Array (4)
price: 20000
available: false
Images: Array (2)
lat: 19.118114757605127
lng: 72.86509607941272
reviews: Array (empty)
landlord: ObjectId('67ebaef765569fbc09fbfb0f')
createdAt: 2025-04-01T09:18:01.343+00:00
updatedAt: 2025-04-03T10:32:11.941+00:00
__v: 0

```

Upon successful Delist, the available attribute is marked false and status code of 200 is returned

Structural Coverage: This testing took into consideration the branch coverage since both the branches of whether the property exists or not and Property belongs to him or not were covered.

Additional Comments: None.

12. DeleteProperty- Backend

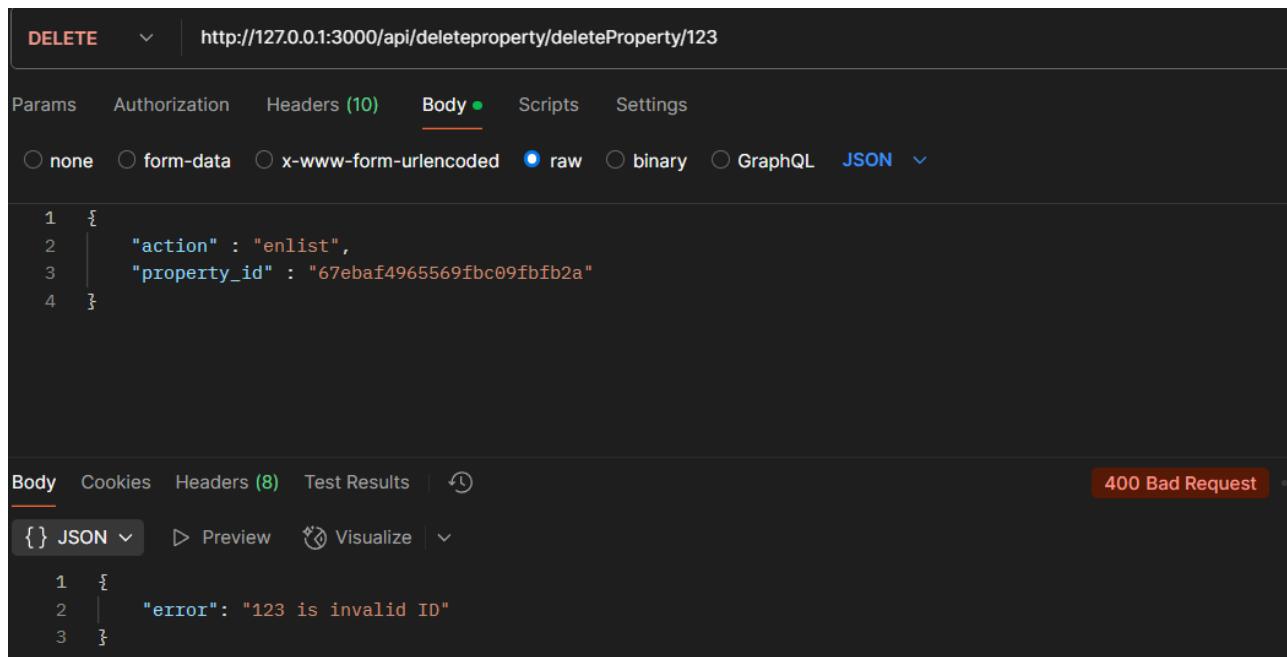
In this unit we have tested the backend function which allows a landlord to delete his own properties from our database

Unit Details: Testing of this unit ensures that a landlord can delete his properties

Test Owner: Saksham Verma

Test Date : 02/04/2025

Test Results: Using this functionality the user was able to successfully able to delete the properties that he had listed



DELETE http://127.0.0.1:3000/api/deleteproperty/deleteProperty/123

Params Authorization Headers (10) Body Scripts Settings

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

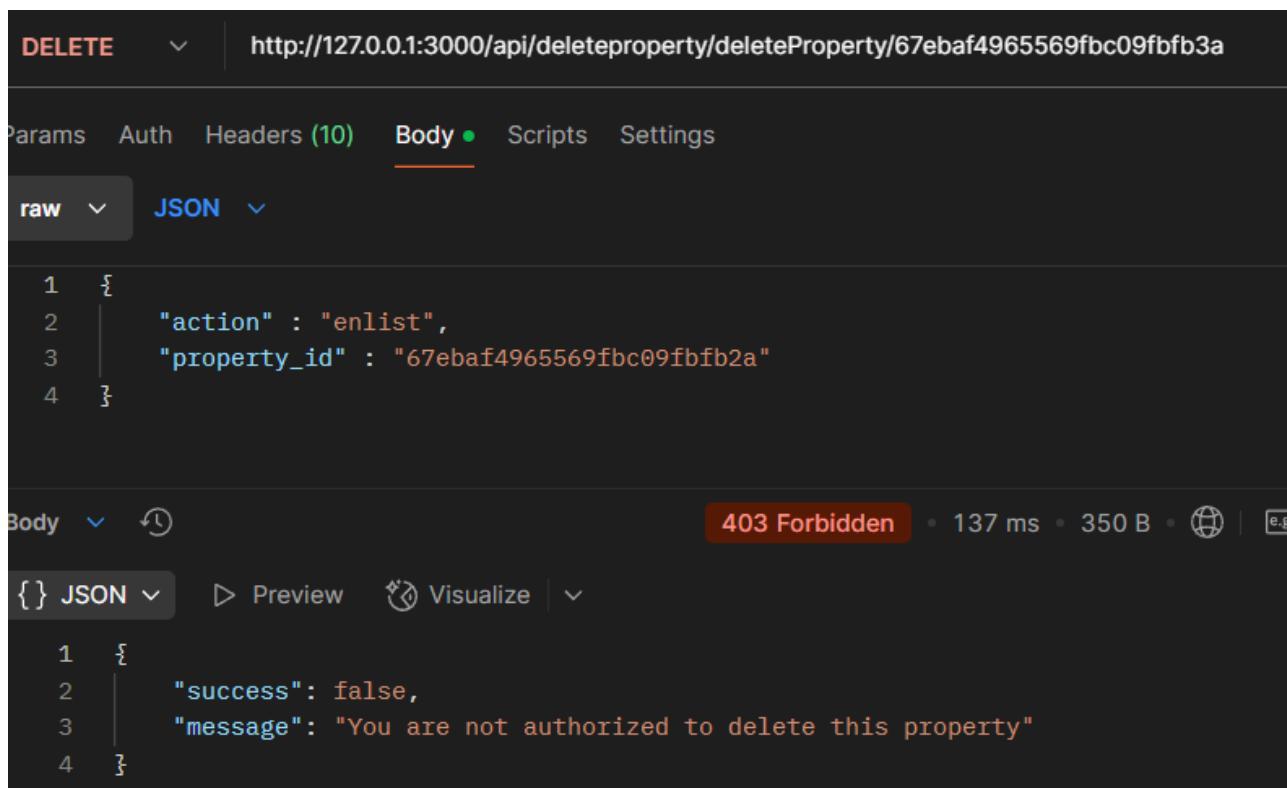
```
1 {  
2   "action" : "enlist",  
3   "property_id" : "67ebaf4965569fbc09fbfb2a"  
4 }
```

Body Cookies Headers (8) Test Results 400 Bad Request

{ } JSON ▾ ▷ Preview ⚡ Visualize ▾

```
1 {  
2   "error": "123 is invalid ID"  
3 }
```

A status code of 400 is returned if the syntax of ID is invalid



DELETE http://127.0.0.1:3000/api/deleteproperty/deleteProperty/67ebaf4965569fbc09fbfb3a

Params Auth Headers (10) Body Scripts Settings

raw JSON

```
1 {  
2   "action" : "enlist",  
3   "property_id" : "67ebaf4965569fbc09fbfb2a"  
4 }
```

Body 403 Forbidden 137 ms 350 B ⌂ e.g.

{ } JSON ▾ ▷ Preview ⚡ Visualize ▾

```
1 {  
2   "success": false,  
3   "message": "You are not authorized to delete this property"  
4 }
```

A status code of 403(Forbidden) is returned if the given ID isn't in Tenant_propertyList

The screenshot shows a Postman API test. The method is **DELETE**, the URL is <http://127.0.0.1:3000/api/deleteproperty/deleteProperty/67ebaf4965569fbc09fbfb2a>. The **Body** tab is selected, showing the following JSON payload:

```

1  {
2    "action" : "enlist",
3    "property_id" : "67ebaf4965569fbc09fbfb2a"
4  }

```

The response is **200 OK** with a duration of **305 ms** and a size of **325 B**. The response body is:

```

{} JSON ▾  ▶ Preview  ⏷ Visualize | ▾
1  {
2    "success": true,
3    "message": "Property deleted successfully"
4  }

```

The property is successfully deleted if the property exists in his Tenant_propertyList

Structural Coverage: This testing took into consideration the branch coverage since both the branches of whether the property exists or not and whether the Property belongs to him or not were covered.

Additional Comments: None.

13. ViewProperty- Backend

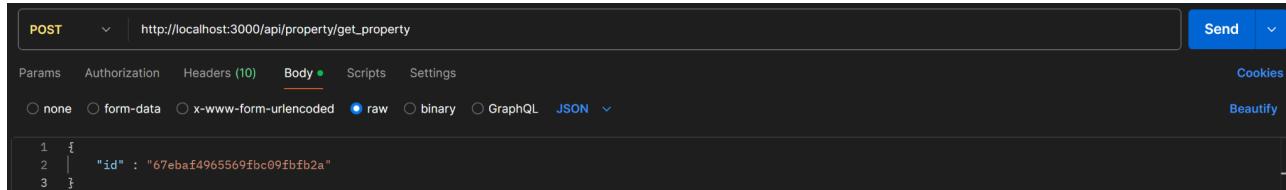
In this unit, we have tested the backend function which allows a user to view the complete details of a property.

Unit Details: Testing of this unit ensures that the user can successfully view any property that has a valid ID in the database.

Test Owner: Rathod Ayushi

Test Date: 02/04/2025

Test Results: Using this functionality the user was able to successfully view the complete details about the property.



On performing the above POST request through Postman

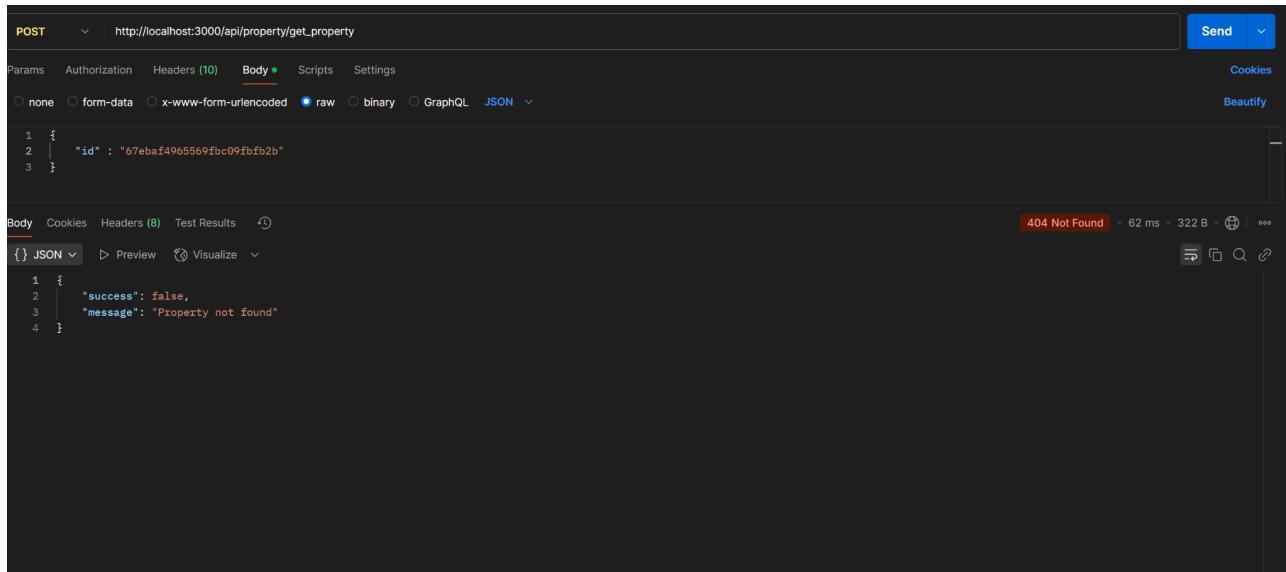
We got the following results as a json-response:

```
{
  "success": true,
  "property": {
    "_id": "67ebaf4965569fbc09fbfb2a",
    "city": "Mumbai",
    "town": "Andheri",
    "address": "Andheri East, Near Dutta Pan Palace",
    "area": 2000,
    "bhk": 2,
    "description": "Very good place to live",
    "amenities": [
      "Gym",
      "Pan Palace",
      "Swimming Pool",
      "Bathroom"
    ],
    "price": 20000,
    "available": true,
    "images": [
      "http://localhost:3000/Pictures/property/67ebaf4965569fbc09fbfb2a/0.jpeg",
      "http://localhost:3000/Pictures/property/67ebaf4965569fbc09fbfb2a/1.jpeg"
    ],
    "lat": 19.118114757605127,
    "lng": 72.86509607941272,
    "reviews": [],
    "landlord": "67ebaef765569fbc09fbfb0f",
    "createdat": "2025-04-01T09:18:01.343Z",
    "updatedat": "2025-04-01T09:18:01.343Z",
    "__v": 0
  },
  "navigatingUrl": "https://www.google.com/maps/place/NaN%NaN%NaN%S+NaN%NaN%NaN%W/@19.114135,72.8738457,41221m/data=!3m1!1e3!4m1!3m7!1s0x3be7c630644dc1:0x5da4ed8fd48c69!2sMumbai,+Maharashtra!3b1!8m2!3d19.0759837!4d72.8776559!16zL20vMDR2bXA!3m3!8m2!3d19.1755556!4d72.8655833?entry=ttu&g_e=EgoMDiIMDmxOS4yIKXMDs0JDEwMjExNjM5SAFQAw%3D0%3D"
}
}
```

This result matches with the details of the property available in the database:

```
_id: ObjectId('67ebaf4965569fbc09fbfb2a')
city: "Mumbai"
town: "Andheri"
address: "Andheri East, Near Dutta Pan Palace"
area: 2000
bhk: 2
description: "Very good place to live"
amenities: Array (4)
  price: 20000
  available: true
  Images: Array (2)
    lat: 19.118114757605127
    lng: 72.86509607941272
  reviews: Array (empty)
  landlord: ObjectId('67ebaef765569fbc09fbfb0f')
  createdat: 2025-04-01T09:18:01.343+00:00
  updatedAt: 2025-04-01T09:18:01.343+00:00
  __v: 0
```

Also, in case of searching for an ID which is not available in the database, we get the following response indicating property not found:



The screenshot shows a Postman API testing interface. The URL is `http://localhost:3000/api/property/get_property`. The method is set to `POST`. The `Body` tab is selected, showing a JSON payload with one item: `{ "id": "67eba4965569fbc09fbfb2b" }`. The response status is `404 Not Found`, with a response time of `62 ms` and a size of `322 B`. The response body is `{ "success": false, "message": "Property not found" }`.

Structural Coverage: This testing took into consideration the branch coverage since both branches of whether the property exists or not were covered.

Additional Comments: Along with the details of the property available in the backend, the user also gets the navigating URL of the property, which can be used to locate the property on the map.

14. UpdateProfile-Backend

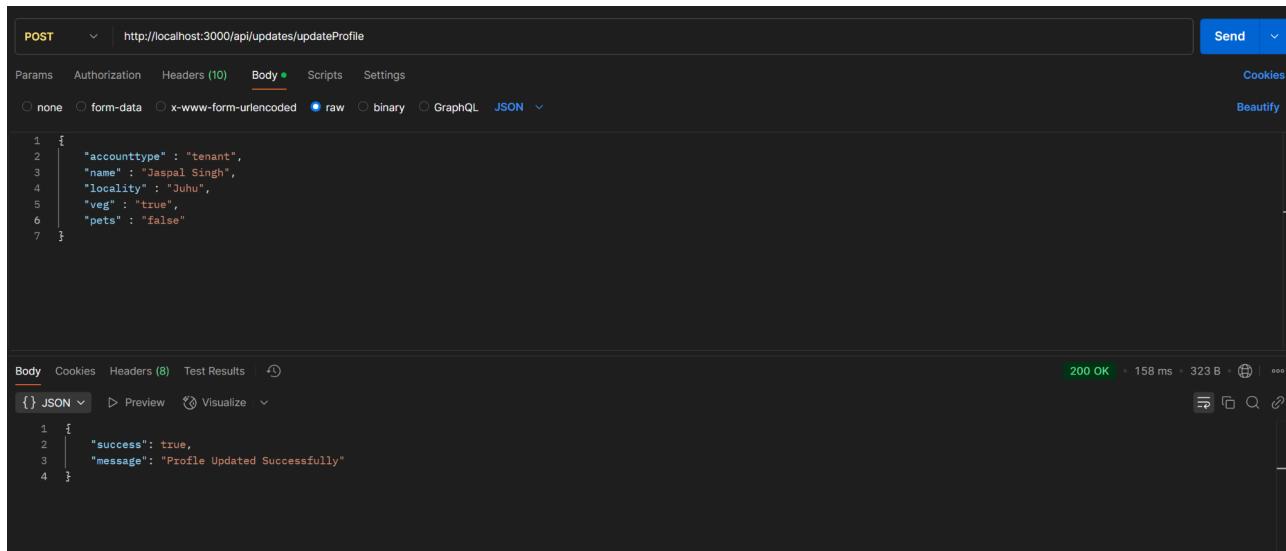
This unit tests the backend function that allows the user to update his/her profile, This functionality is available for both the landlord and the tenant.

Unit Details: This unit ensures that the user is able to update his/her profile by filling out the necessary details and that the changes are reflected in the backend database.

Test Owner: Bikramjeet Singh

Test Date: 02/04/2025

Test Results: The unit successfully updated the user profile by filling out the necessary details.



The screenshot shows a Postman interface with a successful API call. The URL is `http://localhost:3000/api/uploads/updateProfile`. The request method is `POST`. The `Body` tab contains the following JSON payload:

```

1 {
2   "accounttype": "tenant",
3   "name": "Jaspal Singh",
4   "locality": "Juhu",
5   "veg": "true",
6   "pets": "false"
7 }

```

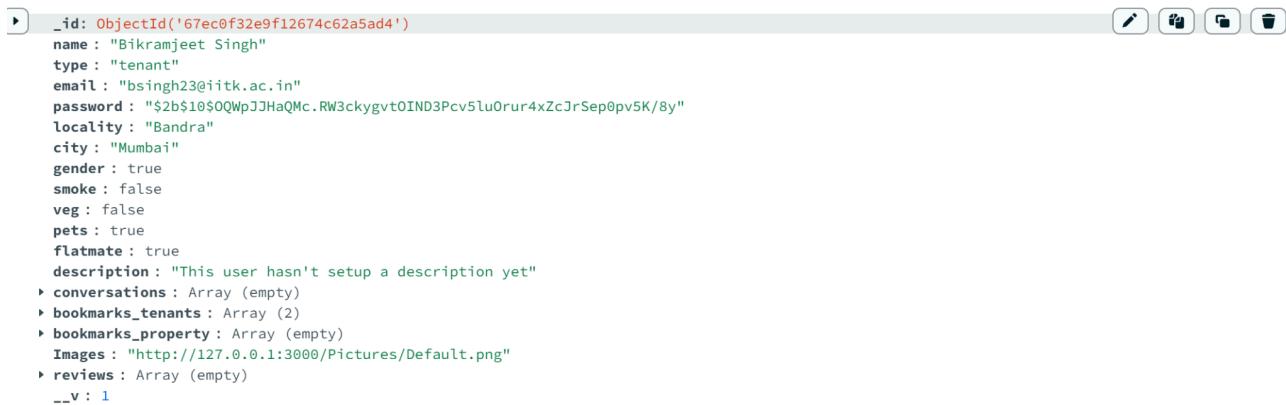
The response status is `200 OK`, with a response time of `158 ms` and a size of `323 B`. The response body is:

```

1 {
2   "success": true,
3   "message": "Profile Updated Successfully"
4 }

```

Upon doing a `POST` request with the account type of the account and the updated fields, the landlord profile was updated successfully, and changes also appeared in the database:



The screenshot shows a MongoDB Compass interface displaying a user profile document. The document has the following fields and values:

- `_id: ObjectId('67ec0f32e9f12674c62a5ad4')`
- `name: "Bikramjeet Singh"`
- `type: "tenant"`
- `email: "bsingh23@iitk.ac.in"`
- `password: "$2b$10$OQwpJJHaQMc.RW3ckygvtoIND3Pcv5luOrur4xZcJrSep0pv5K/8y"`
- `locality: "Bandra"`
- `city: "Mumbai"`
- `gender: true`
- `smoke: false`
- `veg: false`
- `pets: true`
- `flatmate: true`
- `description: "This user hasn't setup a description yet"`
- `conversations: Array (empty)`
- `bookmarks_tenants: Array (2)`
- `bookmarks_property: Array (empty)`
- `Images: "http://127.0.0.1:3000/Pictures/Default.png"`
- `reviews: Array (empty)`
- `--v: 1`

The above is before the update request; afterwards, it looked like this:

```
_id: ObjectId('67ec0f32e9f12674c62a5ad4')
name : "Jaspal Singh"
type : "tenant"
email : "bsingh23@iitk.ac.in"
password : "$2b$10$OQWpJJHaQMc.RW3ckygt0IND3Pcv5luOrur4xZcJrSep0pv5K/8y"
locality : "Juhu"
city : "Mumbai"
gender : true
smoke : false
veg : true
pets : false
flatmate : true
description : "This user hasn't setup a description yet"
▶ conversations : Array (empty)
▶ bookmarks_tenants : Array (2)
▶ bookmarks_property : Array (empty)
Images : "http://127.0.0.1:3000/Pictures/Default.png"
▶ reviews : Array (empty)
__v : 1
```

Clearly, the changes requested in the POST request are reflected successfully in the database/backend.

The screenshot shows a Postman interface with a POST request to `http://localhost:3000/api/uploads/updateProfile`. The request body contains the following JSON:

```
1 "accounttype": "tenant",
2 "name": "Jaspal Singh",
3 "locality": "Juhu",
4 "veg": "true",
5 "pets": "false"
```

The response status is 400 Bad Request, with a message indicating an invalid token. The response body is:

```
1 {
2   "message": "Invalid Token",
3   "success": false
4 }
```

Also, if the auth-token being sent in the headers is invalid/expired, then an appropriate error message is generated, here the auth-token was deliberately modified to test this branch.

Structural Coverage: This unit takes into account the branch coverage of this functionality by checking whether the auth token is valid or not.

Additional Comments: This route does not allow the user to change his/her password or email, we have those routes available in the backend but have not successfully integrated them with the frontend.

15. UpdateProperty-Backend

This unit tests the backend function, which allows the landlord to update his/her property by filling out the necessary details.

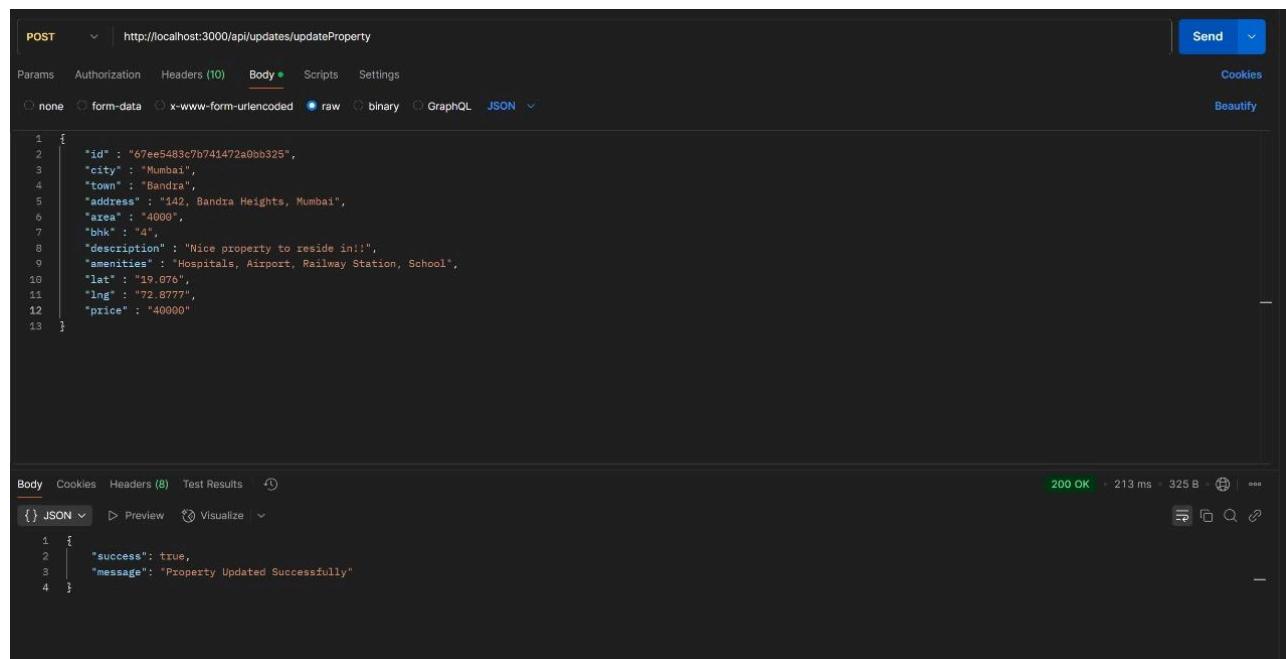
Unit Details: This unit ensures that the landlord can easily update any details regarding his/her property.

Test Owner: Bikramjeet Singh

Test Date: 03/04/2025

Test Results:

This unit was successful in updating the details of the property of a verified landlord, that is, the actual owner of the property.



The screenshot shows a POST request in Postman to the endpoint `http://localhost:3000/api/updates/updateProperty`. The request body is a JSON object containing property details:

```

1  {
2   "id" : "67ee5483c7b741472a0bb325",
3   "city" : "Mumbai",
4   "town" : "Bandra",
5   "address" : "142, Bandra Heights, Mumbai",
6   "area" : "4000",
7   "bhk" : "4",
8   "description" : "Nice property to reside in!!",
9   "amenities" : "Hospitals, Airport, Railway Station, School",
10  "lat" : "19.076",
11  "lng" : "72.8777",
12  "price" : "40000"
13 }

```

The response status is 200 OK, with a response time of 213 ms, a response size of 325 B, and a message indicating success: "Property Updated Successfully".

Upon initiating the above POST request, a success message indicating successful The update of the property was shown, which is also reflected in the database:

*The above are the details of the property before the update; below are the same
After the update:*

```
_id: ObjectId('67ee5483c7b741472a0bb325')
city : "Mumbai"
town : "Bandra"
address : "142, Bandra Heights, Mumbai"
area : 4000
bhk : 4
description : "Nice property to reside in!!"
► amenities: Array (1)
  price : 40000
  available : true
► Images: Array (1)
  lat : 19.076
  lng : 72.8777
► reviews: Array (empty)
landlord : ObjectId('67ee47b5e7e3855709247863')
createdAt : 2025-04-03T09:27:31.258+00:00
updatedAt : 2025-04-03T09:33:00.230+00:00
__v : 1
```

Also in case the user misses some of the necessary fields for the property, then the user is shown the following error message:

►

```
_id: ObjectId('67ee5483c7b741472a0bb325')
city : "Mumbai"
town : "Juhu"
address : "143, Bandra Heights, Mumbai"
area : 4000
bhk : 4
description : "Nice Property to live-in, surrounded by a peaceful atmosphere alongwit..."
► amenities: Array (4)
  price : 30000
  available : true
► Images: Array (1)
  lat : 19.076
  lng : 72.8777
► reviews: Array (empty)
landlord : ObjectId('67ee47b5e7e3855709247863')
createdAt : 2025-04-03T09:27:31.258+00:00
updatedAt : 2025-04-03T09:27:31.258+00:00
__v : 0
```



The screenshot shows a sequence of requests in Postman:

- Request 1:** POST http://localhost:3000/api/updates/updateProperty. Body: JSON (id: 67ee5483c7b741472a0bb325, city: Mumbai, locality: Bandra, address: 142, Bandra Heights, Mumbai, area: 4000, bhk: 4, description: Nice property to reside in!!, amenities: Hospitals, Airport, Railway Station, School, lat: 19.076, lng: 72.8777, price: 40000). Response: 200 OK.
- Request 2:** POST http://localhost:3000/api/updates/updateProperty. Body: JSON (id: 67ebc633bab175b5cf23c6ac, city: Mumbai, town: Bandra, address: 142, Bandra Heights, Mumbai, area: 4000, bhk: 4, description: Nice property to reside in!!, amenities: Hospitals, Airport, Railway Station, School, lat: 19.076, lng: 72.8777, price: 40000). Response: 200 OK.
- Request 3:** POST http://localhost:3000/api/updates/updateProperty. Body: JSON (empty object). Response: 400 Bad Request - "The server could not understand the request. Maybe a bad syntax?"
- Request 4:** POST http://localhost:3000/api/updates/updateProperty. Body: JSON (id: 67ebc633bab175b5cf23c6ac, city: Mumbai, town: Bandra, address: 142, Bandra Heights, Mumbai, area: 4000, bhk: 4, description: Nice property to reside in!!, amenities: Hospitals, Airport, Railway Station, School, lat: 19.076, lng: 72.8777, price: 40000). Response: 400 Bad Request - "You do not own this property".

Also, if any unverified landlord tries to update any property that is his/her auth-token is expired or invalid The following response is shown to the user:

Hence, we can conclude that if a verified landlord tries to update his/her owned property, then the property is successfully updated.

Structural Coverage: This particular test completes branch coverage since we are testing for invalid landlords, invalid properties, and valid properties that are not owned by the concerned landlord.

Additional Comments: None.

16. SearchingFlatmates-Backend

In this unit, we test the backend function that allows a tenant to search for possible flatmates by applying various filters of his/her choice.

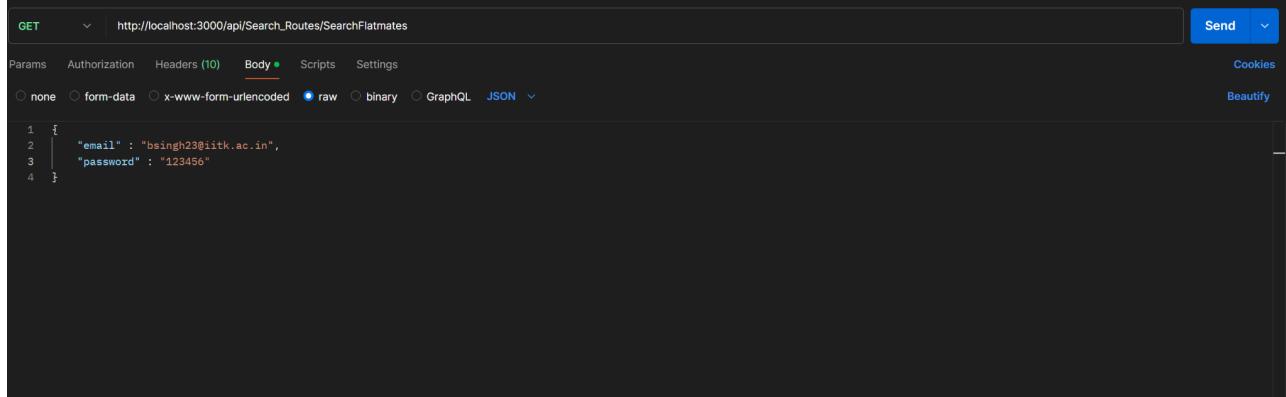
Unit Details: The efficient and correct working of this unit will ensure that the tenant gets the list of flatmates that match his/her priorities and hence would be able to make an informed decision regarding that choice.

Test Owner: Bikramjeet Singh

Test Date: 03/04/2025

Test Results:

The testing of this unit showed that it is indeed working as expected:



GET http://localhost:3000/api/Search_Routes/SearchFlatmates

Params Authorization Headers (10) Body Scripts Settings Cookies Beautify

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

```
1 {
2   "email" : "bsingh23@iitk.ac.in",
3   "password" : "123456"
4 }
```

The above GET request was initiated in POSTMAN

The following list of flatmates was obtained. here no filter is being applied:

```
{
  "success": true,
  "message": "Found 4 matching tenants",
  "data": [
    {
      "_id": "67ebae9765569fbc09fbfafe",
      "name": "Saksham Verma",
      "type": "tenant",
      "email": "saksham07822805@gmail.com",
      "locality": "Bandra",
      "city": "Mumbai",
      "gender": true,
      "smoke": true,
      "veg": false,
      "pets": true,
      "flatmate": true,
      "description": "This user hasn't setup a description yet",
      "conversations": [
        {
          "_id": "67eb788a115329bd4598ea",
          "user_id": "67ebc33dc288d28a44491018"
        }
      ],
      "bookmarks_tenants": [],
      "bookmarks_property": [],
      "Images": "http://127.0.0.1:3000/Pictures/Default.png",
      "reviews": [],
      "_v": 4,
      "recommendationScore": 0.925,
      "bookmarked": true
    },
    {
      "_id": "67ebc4919a575e22057f036f",
      "name": "Hitarth Makwana",
      "type": "tenant",
      "email": "hitarthkm23@iitk.ac.in",
      "locality": "Andheri",
      "city": "Mumbai",
      "gender": true,
      "smoke": false,
      "veg": true,
      "pets": true,
      "flatmate": true,
      "description": "This user hasn't setup a description yet",
      "conversations": [
        {
          "_id": "67ebc5a9a575e22057f03da"
        }
      ],
      "bookmarks_tenants": [],
      "bookmarks_property": [
        {
          "_id": "67ebc94ca9a575e22057f043a"
        }
      ],
      "Images": "http://127.0.0.1:3000/Pictures/Default.png",
      "reviews": []
    }
  ]
}
```

Showing two out of the four due to space and clarity constraints.

Only a verified tenant is allowed to see the flatmates; if a tenant with an expired or invalid auth-token attempts to do so, he/she is denied :

The screenshot shows a JSON response with the following content:

```

Body Cookies Headers (8) Test Results ↻
{} JSON ▾ Preview Visualize ▾
1  {
2   |   "message": "Invalid Token",
3   |   "success": false
4   |

```

This indicates that the request was denied because the token was invalid.

Structural Coverage: This testing took into account the functional coverage and

partial branch coverage since we did not mention here the result with the filter applied to it.

Additional Comments: This function also returns the filtered list of the possible flatmates, although it has not been shown here since it is apt to be shown in the integration part under the frontend part.

17. Searching Property-Backend

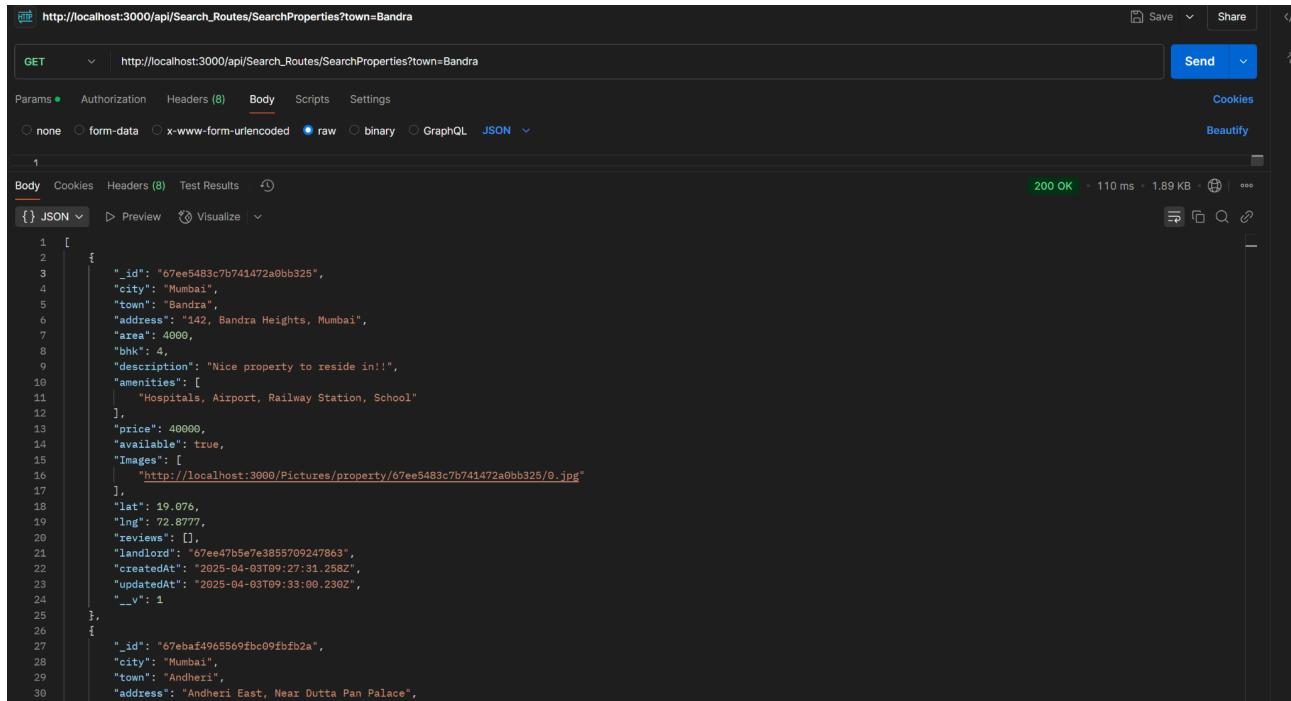
In this unit, we test the backend function that allows a verified tenant to search for properties as per his/her locality of choice and also filter on the basis of BHK, price, and area requirements.

Unit Details: The successful & efficient working of this unit must allow the tenant to search for the property based on the various filters.

Test Owner: Bikramjeet Singh

Test Date: 03/04/2025

Test Results: The unit successfully allowed the tenant to search properties in the locality of his/her choice:

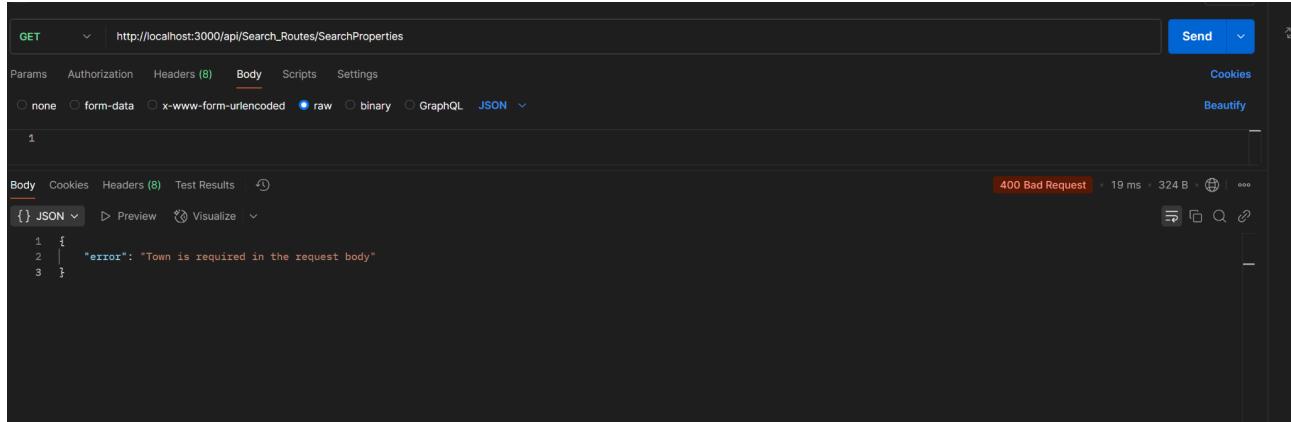


```

1 [
2   {
3     "_id": "67ee5483c7b741472a0bb325",
4     "city": "Mumbai",
5     "town": "Bandra",
6     "address": "142, Bandra Heights, Mumbai",
7     "area": 4000,
8     "bhk": 4,
9     "description": "Nice property to reside in!",
10    "amenities": [
11      "Hospitals, Airport, Railway Station, School"
12    ],
13    "price": 40000,
14    "available": true,
15    "Images": [
16      "http://localhost:3000/Pictures/property/67ee5483c7b741472a0bb325/0.jpg"
17    ],
18    "lat": 19.076,
19    "lng": 72.8777,
20    "reviews": [],
21    "landlord": "67ee47b5e7e3855709247863",
22    "createdAt": "2025-04-03T09:27:31.258Z",
23    "updatedAt": "2025-04-03T09:33:00.230Z",
24    "_v": 1
25  },
26  {
27    "_id": "67eba24965569fbc09fbfb2a",
28    "city": "Mumbai",
29    "town": "Andheri",
30    "address": "Andheri East, Near Dutta Pan Palace",
31  }
32 ]
  
```

Upon initiating the *GET* request using POSTMAN, we get the responses according to the town that has been mentioned in the query string.

Also, in case, we do not send the town in the query string we get the following error indicating that it is indeed mandatory to send the town:



```

1
  
```

```

1 {
2   "error": "Town is required in the request body"
3 }
  
```

Applying filters also gives correct results:

The screenshot shows a POSTMAN interface with a GET request to `http://localhost:3000/api/Search_Routes/SearchProperties?town=Bandra`. The response is a 200 OK status with a response time of 93 ms and a size of 1.89 KB. The JSON response is as follows:

```

1 [
2   {
3     "_id": "67ee5483c7b741472a0bb325",
4     "city": "Mumbai",
5     "town": "Bandra",
6     "address": "142, Bandra Heights, Mumbai",
7     "area": 4000,
8     "bhk": 4,
9     "description": "Nice property to reside in!!",
10    "amenities": [
11      "Hospitals, Airport, Railway Station, School"
12    ],
13    "price": 40000,
14    "available": true,
15    "Images": [
16      "http://localhost:3000/Pictures/property/67ee5483c7b741472a0bb325/0.jpg"
17    ],
18    "lat": 19.076,
19    "lng": 72.8777,
20    "reviews": [],
21    "landlord": "67ee47b5e7e3855709247863",
22    "createdAt": "2025-04-03T09:27:31.258Z",
23    "updatedAt": "2025-04-03T09:33:00.230Z",
24    "__v": 1
25  }
]

```

Before applying any filter, we get the above result; however applying the filter say regarding the maximum price being 35000 provides the following result:

The screenshot shows a POSTMAN interface with a GET request to `http://localhost:3000/api/Search_Routes/SearchProperties?town=Bandra&max_price=35000`. The response is a 200 OK status with a response time of 91 ms and a size of 1.39 KB. The JSON response is as follows:

```

1 [
2   {
3     "_id": "67ebaf4965569fbc09fbfb2a",
4     "city": "Mumbai",
5     "town": "Andheri",
6     "address": "Andheri East, Near Dutta Pan Palace",
7     "area": 2000,
8     "bhk": 2,
9     "description": "Very good place to live",
10    "amenities": [
11      "Gym",
12      "Pan Palace",
13      "Swimming Pool",
14      "Bathroom"
15    ],
16    "price": 20000,
17    "available": true,
18    "Images": [
19      "http://localhost:3000/Pictures/property/67ebaf4965569fbc09fbfb2a/0.jpeg",
20      "http://localhost:3000/Pictures/property/67ebaf4965569fbc09fbfb2a/1.jpeg"
21    ],
22    "lat": 19.118114757695127,
23    "lng": 72.86509607941272,
24    "reviews": [],
25    "landlord": "67ebaf705569fbc09fbfb0f",
26    "createdAt": "2025-04-01T09:18:01.343Z",
27    "updatedAt": "2025-04-03T10:33:15.508Z",
28    "__v": 0
29  }
]

```

which clearly removes the initial property which had a price of more than 35000.

Structural Coverage: The testing covers the functional coverage and partial branch coverage.

Additional Comments: Although not explicitly shown here, there are many more filters that have been implemented to filter out properties on more specific choices of the user.

18. Review Property-Backend

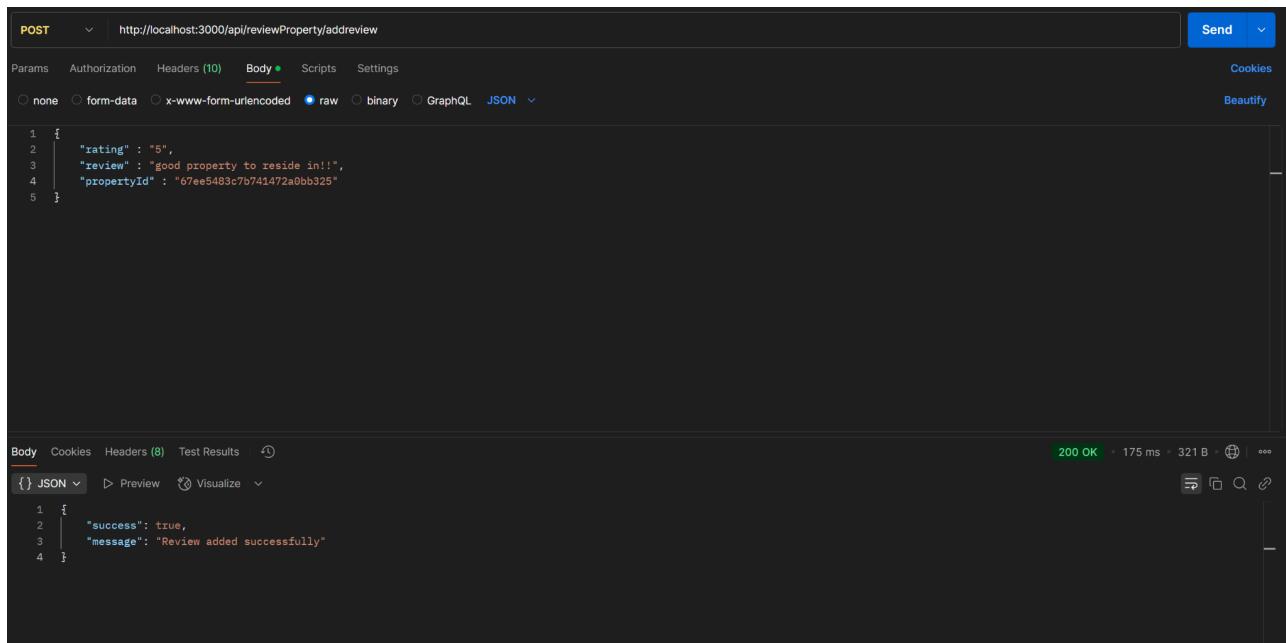
This unit tests the backend function, which allows a user to give a review about any property.

Unit Details: This unit ensures the tenant is able to post a review about some property in the form of a rating out of 5 stars as well as descriptive writing about it.

Test Owner: Bikramjeet Singh

Test Date: 03/04/2025

Test Results: The unit was successful in allowing the user to post a review about a certain property, including rating and the descriptive review.



```

POST http://localhost:3000/api/reviewProperty/addreview
Body
Params Authorization Headers (10) Body Scripts Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON
1 {
2   "rating": "5",
3   "review": "good property to reside in!",
4   "propertyId": "67ee5463c7b741472a0bb325"
5 }

```

Body Cookies Headers (8) Test Results

200 OK 175 ms 321 B

```

{} JSON > Preview Visualize
1 {
2   "success": true,
3   "message": "Review added successfully"
4 }

```

129392

The above POST request successfully updates the review in the database as seen clearly below:

```

▶ _id: ObjectId('67ee5483c7b741472a0bb325')
  city : "Mumbai"
  town : "Bandra"
  address : "142, Bandra Heights, Mumbai"
  area : 4000
  bhk : 4
  description : "Nice property to reside in!!"
  ▶ amenities : Array (1)
    price : 40000
    available : true
  ▶ Images : Array (1)
    lat : 19.076
    lng : 72.8777
  ▶ reviews : Array (empty)
  landlord : ObjectId('67ee47b5e7e3855709247863')
  createdAt : 2025-04-03T09:27:31.258+00:00
  updatedAt : 2025-04-03T11:06:33.258+00:00
  __v : 3

```



The array of reviews is empty before the POST request, but after the POST request, a review appears in the same array:

```

▶ _id: ObjectId('67ee5483c7b741472a0bb325')
  city : "Mumbai"
  town : "Bandra"
  address : "142, Bandra Heights, Mumbai"
  area : 4000
  bhk : 4
  description : "Nice property to reside in!!"
  ▶ amenities : Array (1)
    price : 40000
    available : true
  ▶ Images : Array (1)
    lat : 19.076
    lng : 72.8777
  ▶ reviews : Array (1)
    0: Object
      reviewer : ObjectId('67ec0f32e9f12674c62a5ad4')
      reviewertype : "tenant"
      rating : "5"
      comment : "good property to reside in!!"
    landlord : ObjectId('67ee47b5e7e3855709247863')
    createdAt : 2025-04-03T09:27:31.258+00:00
    updatedAt : 2025-04-03T11:06:33.258+00:00
    __v : 3

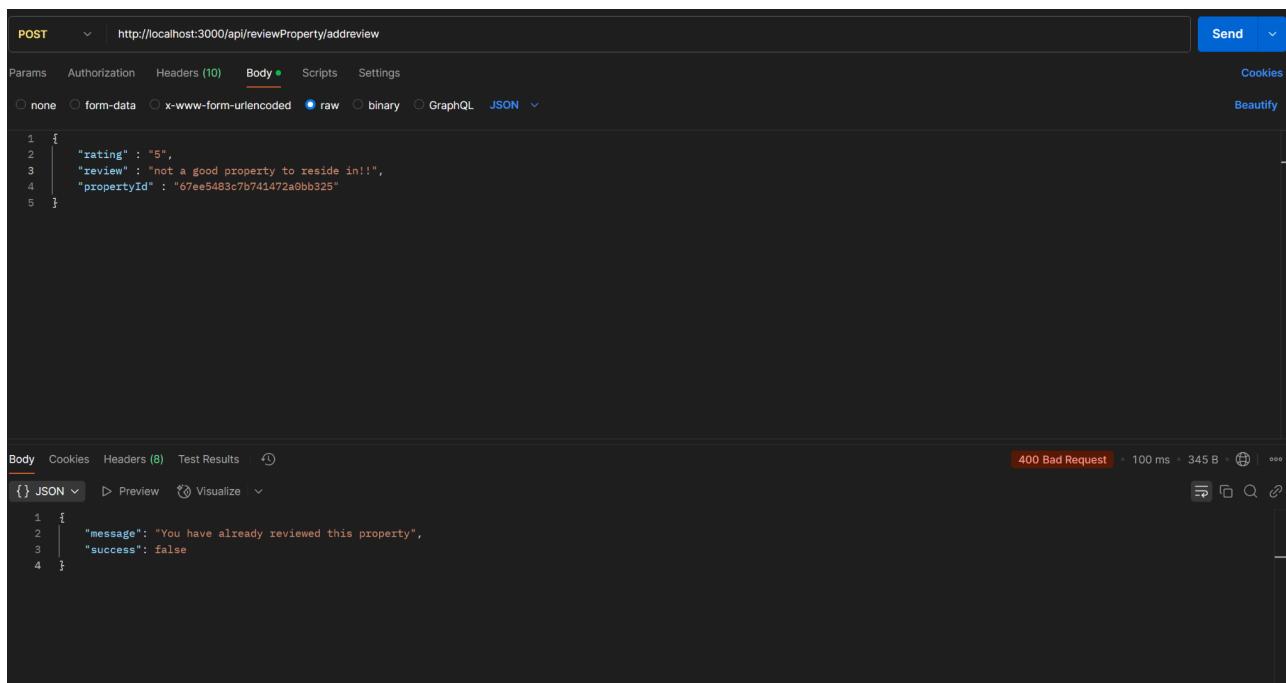
```



Also, if the user tries to review the same property twice/or more he is not allowed to:

Software Test Document for Roombler

72

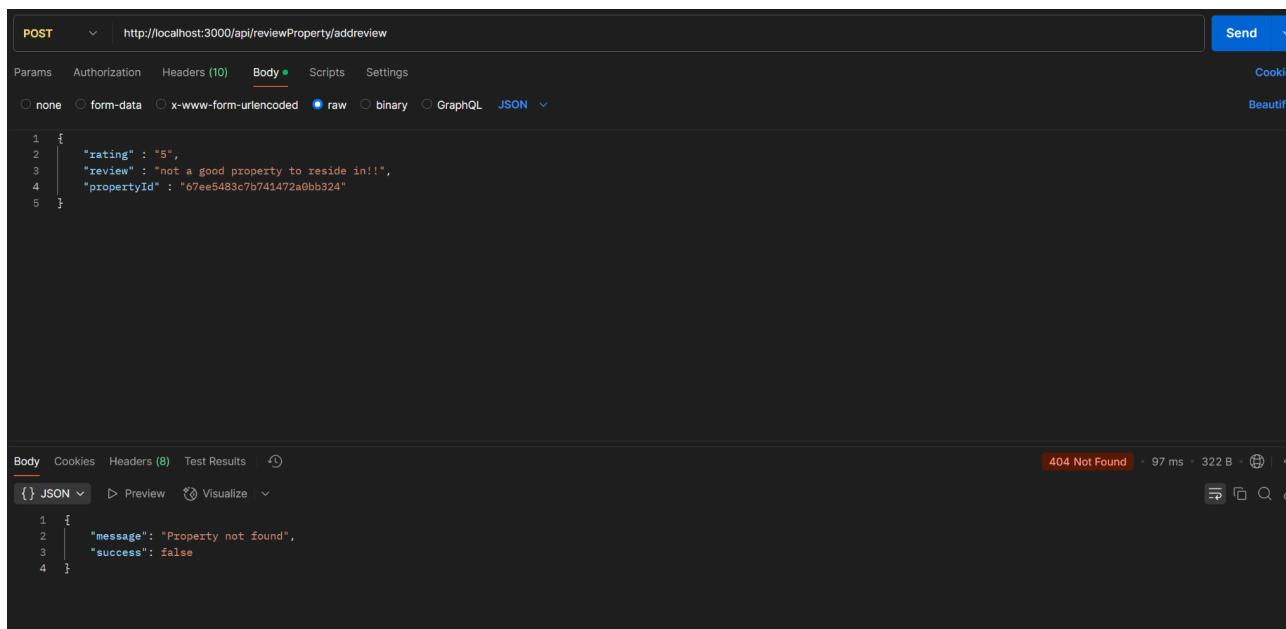


The screenshot shows a POST request to `http://localhost:3000/api/reviewProperty/addreview`. The body contains the following JSON:

```
1 {  
2   "rating": "5",  
3   "review": "not a good property to reside in!!",  
4   "propertyId": "67ee5483c7b741472a0bb325"  
5 }
```

The response status is 400 Bad Request, with a response time of 100 ms, 345 B, and a message: "You have already reviewed this property".

Also, the user is not allowed to give a review to an invalid property:



The screenshot shows a POST request to `http://localhost:3000/api/reviewProperty/addreview`. The body contains the same JSON as the previous test.

The response status is 404 Not Found, with a response time of 97 ms, 322 B, and a message: "Property not found".

Structural Coverage: This testing takes into account complete branch coverage.

Additional Comments: None.

19. Get Property Reviews- Backend

This unit tests the backend function which basically allows a user to fetch and see the reviews about a certain property.

Unit Details : The successful working of this unit ensures that the user is able to see the reviews regarding a particular property along with the person who has posted the review.

Test Owner: Bikramjeet Singh

Test Date : 03/04/2025

Test Results: The unit was successfully tested and gave apt results when reviews of a particular property were demanded.

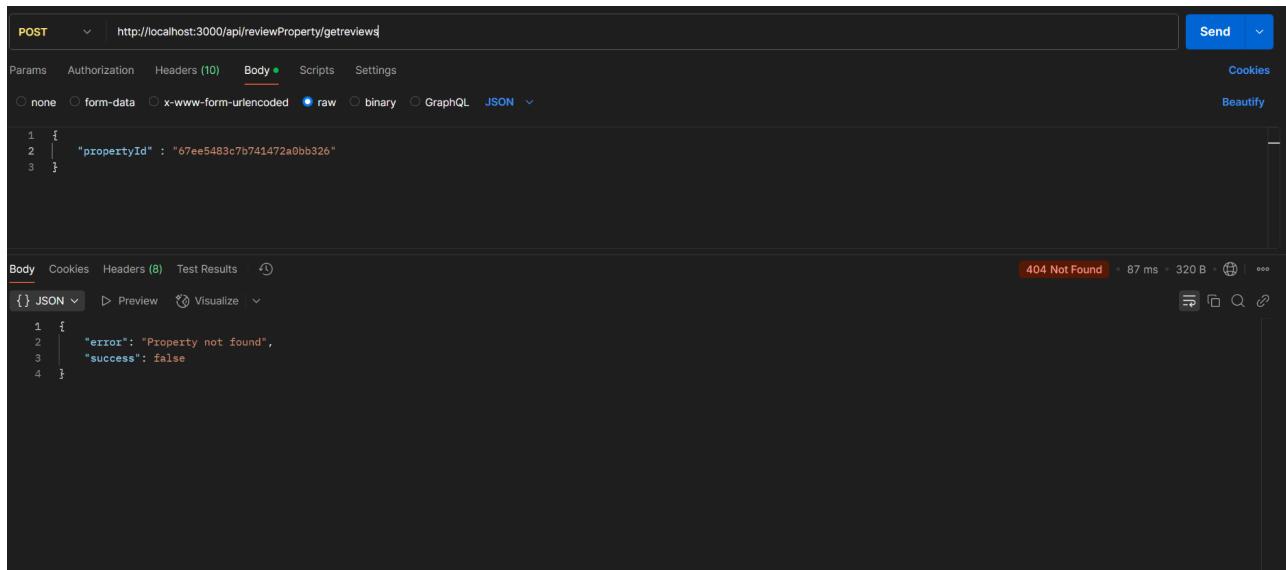
```

POST http://localhost:3000/api/reviewProperty/getreviews
Body
Params Authorization Headers (10) Body Scripts Settings Cookies Beautify
none form-data x-www-form-urlencoded raw binary GraphQL JSON
1 {
2   "propertyId" : "67ee5483c7b741472a0bb328"
3 }

Body Cookies Headers (8) Test Results
{} JSON Preview Visualize
1 {
2   "success": true,
3   "reviews": [
4     {
5       "rating": "5",
6       "comment": "not a good property to reside in!!",
7       "name": "Bikramjeet Singh",
8       "email": "bsingh2@iitk.ac.in",
9       "image": "http://127.0.0.1:3000/Pictures/Default.png"
10    }
11  ]
12 }
200 OK 182 ms 467 B

```

The above POST request generated the response in the above image, which correctly shows all the details about the review be it the rating, the reviewee or the content.



The screenshot shows a Postman interface with a POST request to `http://localhost:3000/api/reviewProperty/getreviews`. The request body contains the following JSON:

```
1 {  
2   "propertyId": "67ee5483c7b741472a0bb326"  
3 }
```

The response status is 404 Not Found, with a response time of 87 ms and a response size of 320 B. The response body is:

```
1 {  
2   "error": "Property not found",  
3   "success": false  
4 }
```

The above POST request which provides an invalid ID is provided an appropriate response regarding the same.

Structural Coverage: The above testing covers functional coverage and branch coverage.

Additional Comments: None.

20. Messaging and concerned functionalities- Backend

This unit tests the backend route, which allows the user to send a message to another user in real time.

Unit Details: The successful working of this unit ensures that the user can send/receive a message to/from another user.

Test Owner: Bikramjeet Singh

Test Date: 03/04/2025

Test Results: The unit was successfully tested and the two users concerned were able to send and receive messages respectively.

The screenshot shows a POST request to `http://localhost:3000/messages/createConversation`. The request body is a JSON object with a single key-value pair: `"user2" : "67ebae9765569fbc09fbfafe"`. The response status is 200 OK, with a response time of 386 ms and a response size of 328 B. The response body is also a JSON object with keys `"conversation_id"` and `"success"`, both of which have the value `true`.

```

POST http://localhost:3000/messages/createConversation
Params Authorization Headers (10) Body Scripts Settings
Body none form-data x-www-form-urlencoded raw binary GraphQL JSON
1 {
2   "user2" : "67ebae9765569fbc09fbfafe"
3 }

Body Cookies Headers (8) Test Results
200 OK 386 ms 328 B
1 {
2   "conversation_id": "67eebe4f6f0c5c11498c4ff8",
3   "success": true
4 }
  
```

At first we need to create a conversation between two users if there does not exist any prior one else they won't be able to chat.

The screenshot shows a POST request to `http://localhost:3000/messages/sendMessage`. The request body is a JSON object with two key-value pairs: `"conversation_id" : "67eebe4f6f0c5c11498c4ff8"` and `"message" : "Hello!"`. The response status is 200 OK, with a response time of 291 ms and a response size of 516 B. The response body is a complex JSON object containing information about the conversation and the new message sent.

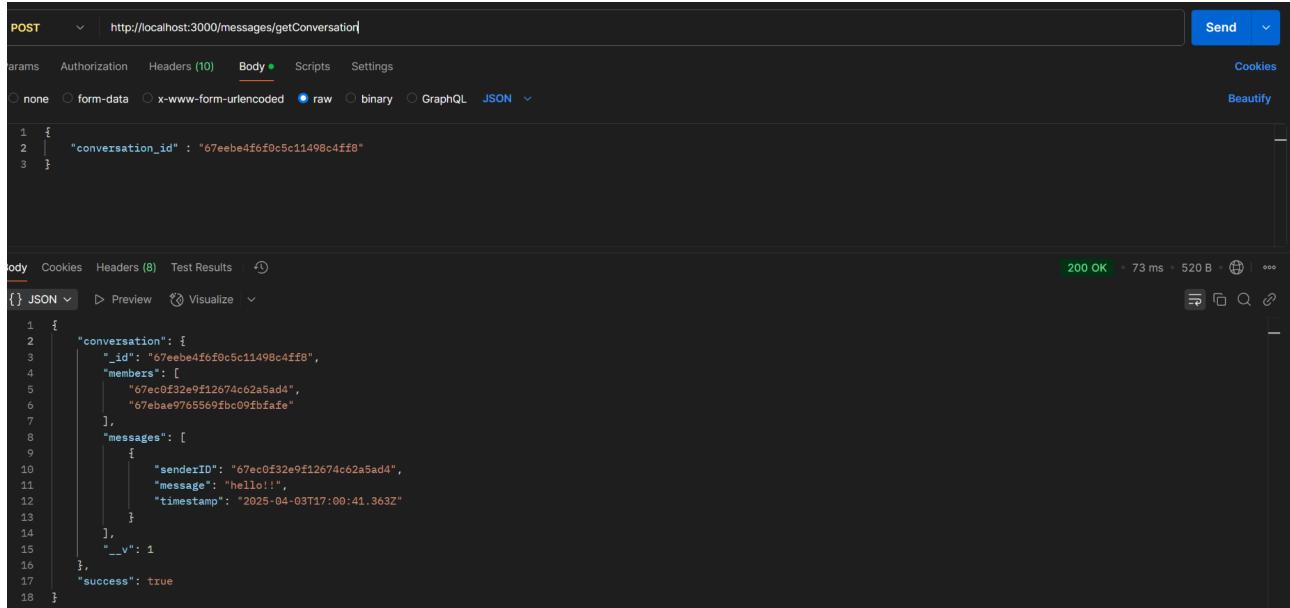
```

POST http://localhost:3000/messages/sendMessage
Params Authorization Headers (8) Body Scripts Settings
Body none form-data x-www-form-urlencoded raw binary GraphQL JSON
1 {
2   "conversation_id" : "67eebe4f6f0c5c11498c4ff8",
3   "message" : "Hello!"
4 }

Body Cookies Headers (8) Test Results
200 OK 291 ms 516 B
1 {
2   "success": true,
3   "messages": [
4     {
5       "_id": "67eebe4f6f0c5c11498c4ff8",
6       "members": [
7         "67ec0f32e9f12674c62a5ad4",
8         "67ebae9765569fbc09fbfafe"
9       ],
10      "messages": [
11        {
12          "senderID": "67ec0f32e9f12674c62a5ad4",
13          "message": "Hello!",
14          "timestamp": "2025-04-03T17:00:41.363Z"
15        }
16      ],
17      "__v": 1
18    }
  
```

Now for sending the message we need to access the `conversation_id` created in the previous request which is stored permanently in the database, and also need to send the message in the body of the request.

On success, we get the response as shown, along with timestamp and the other necessary details.

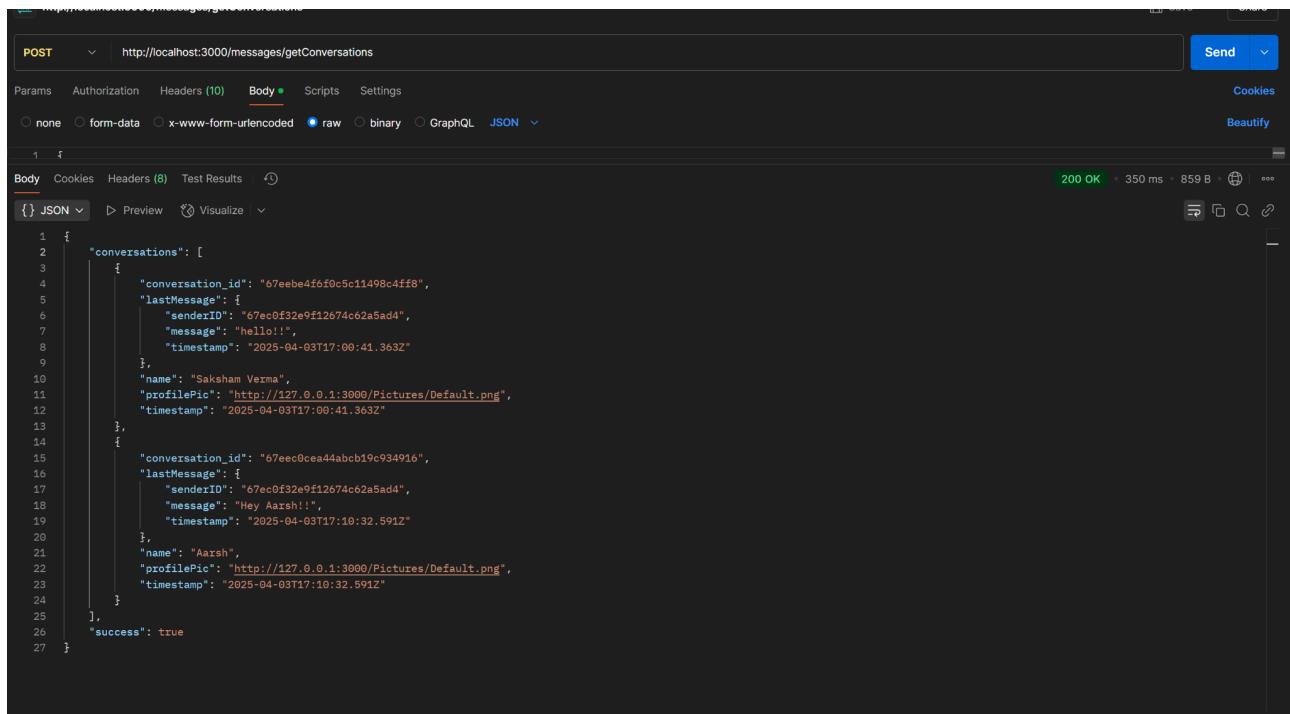


The screenshot shows a Postman request for `http://localhost:3000/messages/getConversation`. The request method is POST. The body is a JSON object with one key-value pair: `"conversation_id" : "67eebe4f6f0c5c11498c4ff8"`. The response status is 200 OK, with a timestamp of 73 ms and a size of 520 B. The response body is a JSON object containing a conversation with a message and a timestamp of 2025-04-03T17:00:41.363Z.

```

POST http://localhost:3000/messages/getConversation
Body
{
  "conversation_id": "67eebe4f6f0c5c11498c4ff8"
}
200 OK
{
  "conversation": {
    "_id": "67eebe4f6f0c5c11498c4ff8",
    "members": [
      "67ec0f32e9f12674c62a5ad4",
      "67ebae9765569bcc09fbfafe"
    ],
    "messages": [
      {
        "senderID": "67ec0f32e9f12674c62a5ad4",
        "message": "hello!",
        "timestamp": "2025-04-03T17:00:41.363Z"
      }
    ],
    "__v": 1
  },
  "success": true
}
  
```

To fetch any conversation, we can do so by using the concerned route which gives the apt output as shown above.

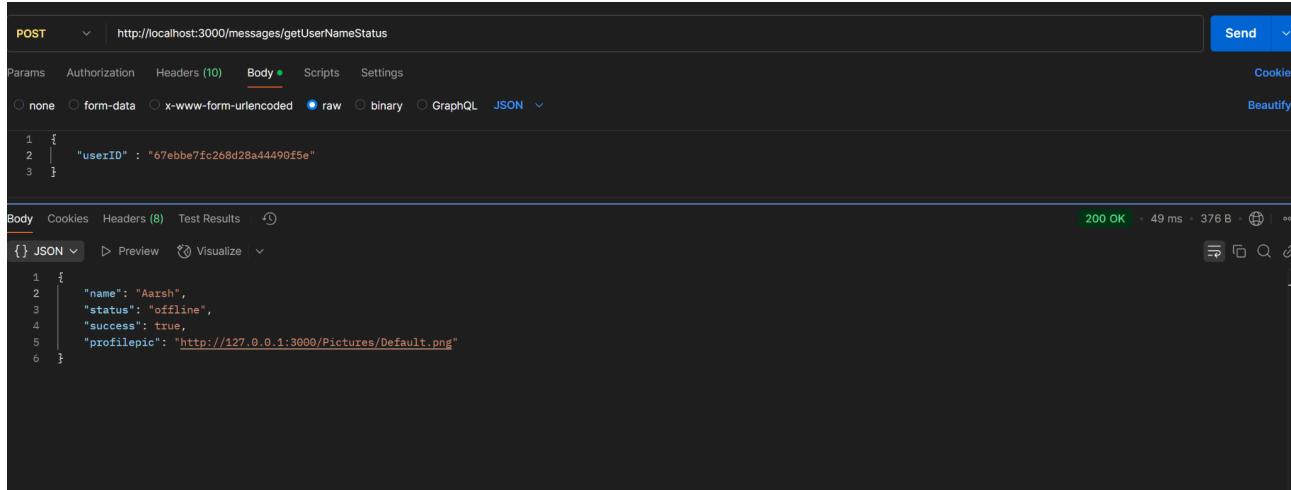


The screenshot shows a Postman request for `http://localhost:3000/messages/getConversations`. The request method is POST. The body is an empty JSON object. The response status is 200 OK, with a timestamp of 350 ms and a size of 859 B. The response body is a JSON object containing two conversations, each with a message and a timestamp of 2025-04-03T17:00:41.363Z.

```

POST http://localhost:3000/messages/getConversations
Body
{
}
200 OK
{
  "conversations": [
    {
      "conversation_id": "67eebe4f6f0c5c11498c4ff8",
      "lastMessage": {
        "senderID": "67ec0f32e9f12674c62a5ad4",
        "message": "hello!",
        "timestamp": "2025-04-03T17:00:41.363Z"
      },
      "name": "Saksham Verma",
      "profilePic": "http://127.0.0.1:3000/Pictures/Default.png",
      "timestamp": "2025-04-03T17:00:41.363Z"
    },
    {
      "conversation_id": "67ec0cea44abcb19c934916",
      "lastMessage": {
        "senderID": "67ec0f32e9f12674c62a5ad4",
        "message": "Hey Aarsh!!",
        "timestamp": "2025-04-03T17:10:32.591Z"
      },
      "name": "Aarsh",
      "profilePic": "http://127.0.0.1:3000/Pictures/Default.png",
      "timestamp": "2025-04-03T17:10:32.591Z"
    }
  ],
  "success": true
}
  
```

To fetch all the conversations of a particular user, we can access the conversations attribute in the user model through the concerned route.



The screenshot shows the Postman application interface. A POST request is being made to the URL `http://localhost:3000/messages/getUserNameStatus`. The request body is set to raw JSON, containing the following data:

```
1 {  
2   "userID" : "67ebbe7fc268d28a44490f5e"  
3 }
```

The response status is 200 OK, with a response time of 49 ms and a size of 376 B. The response body is also JSON, showing the following data:

```
1 {  
2   "name": "Aarsh",  
3   "status": "offline",  
4   "success": true,  
5   "profilepic": "http://127.0.0.1:3000/Pictures/Default.png"  
6 }
```

A user can also check if another user is offline or online at a certain moment.

All these tests indicate that the various routes concerned with messaging work appropriately producing correct results.

Structural Coverage: This testing takes into account the functional testing of the messaging route.

Additional Comments: None

21. Logout- Frontend

This unit tests the correct working of the front-end functionality that allows the user to log out of the currently running session.

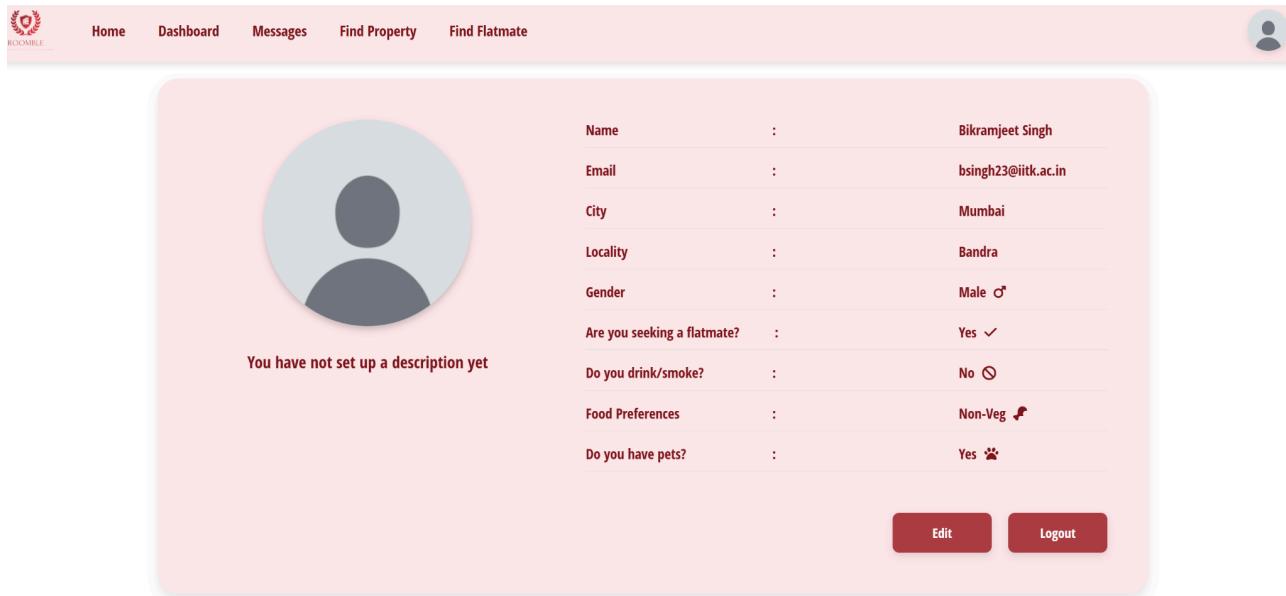
Unit Details: The correct working of the unit is essential so that the user can successfully logout which is an essential part of any web-application.

Test Owner : Bikramjeet Singh

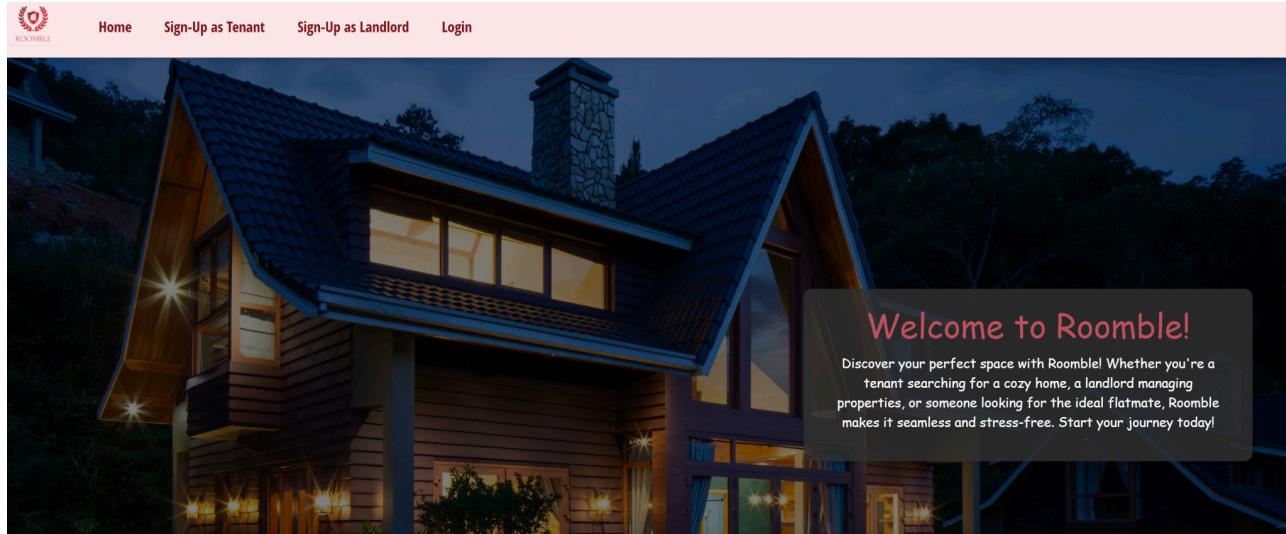
Test Date: 03/04/2025

Test Results:

The unit works correctly since clicking it logs out a user successfully.



The logout button is available in the profile section of the user at the bottom right.



After clicking on logout the user is redirected to this page.

Structural Coverage: This testing covers appropriately the functionality of the logout button.

Additional Comments: Although “Remember Me” is a different functionality as such, it is also a highly related function to logout-login.

The "Remember Me" button allows the user to store his/her credentials locally.

3 Integration Testing

1. Sign Up Tenant

Module Details: This module tests the registration process for a new tenant. It verifies that upon successful sign up, all tenant details and choices are correctly stored in the backend database.

Test Owner: Aarsh Jain

Test Date: 01/04/2025

Signup as a Tenant

Full Name

Email Address

Password

Confirm Password

Next

With Roombie, you'll stumble on the perfect place to rumble!

A Few Questions About You

Where would you like to look for a property?
(For better recommendations)

▼

▼

Gender

MALE
FEMALE

Do you drink/smoke?

YES
NO

Do you have pets?

YES
NO

Food Preferences

VEG
NON-VEG

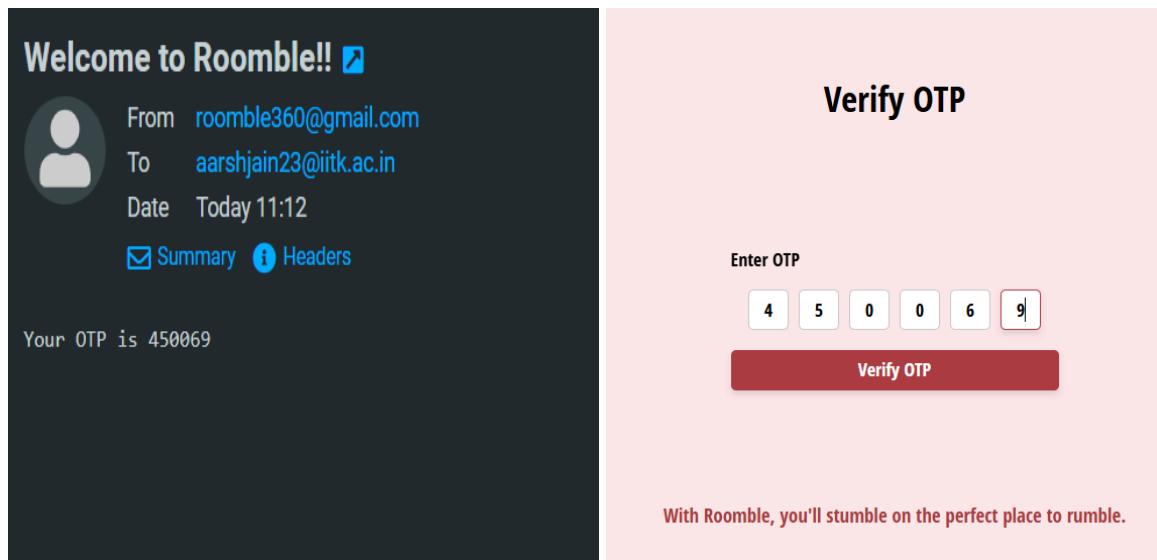
Are you seeking a flatmate?

YES
NO

← Back
Sign up

With Roombie, you'll stumble on the perfect place to rumble!

The user entered the following details, on clicking Sign-Up, the user will be redirected to the OTP page, where he has to enter the OTP sent to the registered email id.



On entering the successful OTP received from the backend, the user should be redirected to the Login page, and details of the user should be stored in the backend.

The screenshot shows the MongoDB Compass interface connected to a cluster. The left sidebar shows connections like cluster0.a9jom.mongodb.net, admin, config, local, oplog.rs, replset.election, replset.minvalid, test, conversations, landlord_otp, landlords, properties, tenant_otp, and tenants. The main area is titled "tenants" and shows a document in the "test" database. The document details are:

```

{
  "_id": ObjectId('67eb7d6fb6ecba52242a5e3'),
  "name": "Aarsh Jain",
  "type": "tenant",
  "email": "aarshjain23@iitk.ac.in",
  "password": "52b1051GJc.TxRnBda4/J2SCcydefiuZi1Ex06QIwSbLD22.KMeu0ofXjzm",
  "locality": "Malad",
  "city": "Mumbai",
  "gender": true,
  "smoke": false,
  "veg": false,
  "pets": true,
  "flatmate": true,
  "description": "This user hasn't setup a description yet",
  "conversations": [],
  "bookmarks_tenants": [],
  "bookmarks_property": [],
  "Images": "http://127.0.0.1:3000/Pictures/Default.png",
  "reviews": []
}

```

Test Results: The new tenant "Aarsh" was successfully registered, and his details were correctly saved in the backend user record.

2. Sign Up Landlord

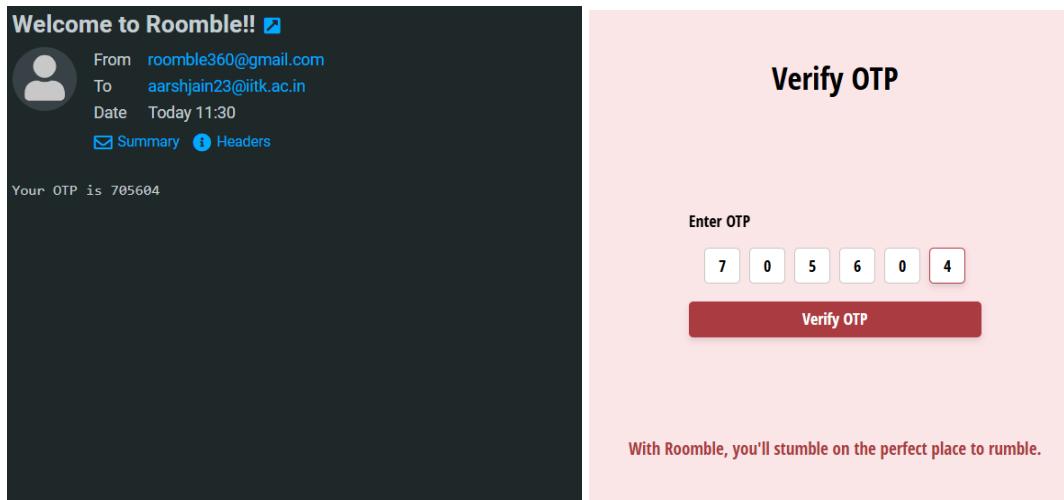
Module Details: This module tests the registration process for a new landlord. It verifies that upon successful sign up, all landlord details are correctly stored in the backend database.

Test Owner: Aarsh Jain

Test Date: 01/04/2025

The screenshot shows a mobile-style form titled "Signup as a Landlord". It has four input fields: "Full Name" (Virat Kohli), "Email Address" (aarshjain23@iitk.ac.in), "Password" (represented by five dots), and "Confirm Password" (also represented by five dots). Below the form is a large red button labeled "Sign up". At the bottom of the screen, there is a footer message: "With Roomble, you'll stumble on the perfect place to rumble!"

The user on clicking Sign Up gets redirected to the OTP page.



```

_id: ObjectId('67eb80fcfb6ecba52242a5e8')
name: "Virat Kohli"
email: "aarsjhain23@iitk.ac.in"
type: "Landlord"
password: "$2b$10$T6GogX4ebQh5Ye5idXmFb.fyQtevzdkhh9IRLHQurcegKa8T9toGw"
OTP: "705604"
propertyList: Array (empty)
conversations: Array (empty)
Allow_changes: false
createdat: 2025-04-01T06:00:28.165+00:00
__v: 0
  
```

The OTP is successfully generated in the backend and mailed to the registered email ID, after entering the correct OTP user is redirected to the Login Page, and the User object with the correct User details is created in the backend

```

{
  "_id": ObjectId("67eb7ccaa35f27b7efa502304"),
  "name": "Saksham Verma",
  "type": "landlord",
  "email": "saksham07022005@gmail.com",
  "password": "$2b$10$svyZDyLyvpb/uw1bQ3X4C20fLEDXQ/QvZ2ypVjPeNtB.mPmBAxinni",
  "propertyList": [],
  "conversations": [],
  "Images": "http://127.0.0.1:3000/Pictures/Default.png",
  "reviews": [],
  "__v": 1
}

{
  "_id": ObjectId("67eb8151fb6ecba52242a5eb"),
  "name": "Virat Kohli",
  "type": "landlord",
  "email": "pareshjain23@itk.ac.in",
  "password": "$2b$10$T66ogX4ebqh5Ye5idXmFb.fyQtevdkh9IRLHQurcegKa8T9toGw",
  "propertyList": [],
  "conversations": [],
  "Images": "http://127.0.0.1:3000/Pictures/Default.png",
  "reviews": [],
  "__v": 0
}

```

Test Results: The Landlord “Virat Kohli” is successfully registered.

3. Login

Module Details: This module includes the tests for validating the details entered for login with the details stored in the database during signup and redirecting to respective user dashboards on successful validation.

Test Owner: Pranay

Test Date: 1/4/2025

Test #1: Tenant Login



Login to your Account

See what is going on with your business



Tenant



Landlord

Email

Password

Remember Me

[Forgot Password?](#)

[Login](#)

Not Registered Yet? [Create an account](#)

With Roomble, you'll stumble on the perfect place to rumble!

Tenant enters details that were entered during signup.

```
_id: ObjectId('67eb77a567dfab515f609f03')
name : "Hitarth Makawana"
type : "tenant"
email : "hitarthkm23@iitk.ac.in"
password : "$2b$10$TBbAwXLrw1qTD5i1Y3FKru27WrkEU8jWyc07/DyisNS3nA0CEtaNK"
locality : "Andheri"
city : "Mumbai"
gender : true
smoke : false
veg : true
pets : true
flatmate : true
description : "This user hasn't setup a description yet"
conversations : Array (empty)
bookmarks_tenants : Array (empty)
bookmarks_property : Array (empty)
Images : "http://127.0.0.1:3000/Pictures/Default.png"
reviews : Array (empty)
__v : 0
```

Data stored in MongoDB database to be validated with login form input, specifically email and password (hashed in MongoDB database for secure authentication).



Your Bookmarked Flatmates

The validation was successful, hence the tenant was redirected to tenant dashboard.

Test #2: Landlord Login

A screenshot of the Roomble landlord login page. The page features a logo with a red laurel wreath and a shield containing a building, followed by the word "ROOMBLE". Below the logo is a tagline: "YOUR PERFECT SPACE, JUST A CLICK AWAY!". The main title "Login to your Account" is displayed at the top, with a sub-instruction "See what is going on with your business". There are two buttons: "Tenant" (white background with a key icon) and "Landlord" (dark red background with a house icon). The "Landlord" button is highlighted. Below these buttons are input fields for "Email" (containing "hitarthkm23@iitk.ac.in") and "Password" (containing "*****"). There are also "Remember Me" and "Forgot Password?" links. A large red "Login" button is centered below the inputs. At the bottom of the form, there's a link "Not Registered Yet? Create an account" and a tagline "With Roomble, you'll stumble on the perfect place to rumble!".

Landlord enters details that were entered during Signup

```
_id: ObjectId('67eb7aea67dfab515f609f1a')
name : "Hitarth Makawana"
type : "Landlord"
email : "hitarthkm23@iitk.ac.in"
password : "$2b$10$TaWuoErY4yaajih/SQRDzulZ25ni0gfHqQ9CK3FfLFMSagi3Rbupi"
propertyList : Array (empty)
conversations : Array (empty)
Images : "http://127.0.0.1:3000/Pictures/Default.png"
reviews : Array (empty)
__v : 0
```

Data stored in MongoDB database to be validated with login form input, specifically email and password (hashed in MongoDB database for secure authentication).

The screenshot shows a web application interface for 'ROOMBLE'. At the top, there's a navigation bar with links for 'Home', 'Dashboard' (which is underlined, indicating it's the active page), 'Add Property', and 'Messages'. On the far right of the header is a user profile icon. Below the header, the main content area has a title 'Your Properties' and a message 'No properties to show'. At the bottom left of the page, there's a small URL bar containing 'localhost:5173/landlord-dashboard'. The overall layout is clean and modern.

The validation was successful, hence the landlord was redirected to landlord dashboard.

Test Results: Both the login pages i.e. for tenant and landlord were successfully integrated with the backend hence the data entered in the login form was successfully validated with the backend database and both the tenant and landlord users were redirected to their respective dashboards after submitting the login form. The procedure flow is as follows:

Data entered in login form -> Data validated with MongoDB database -> User redirected to respective dashboard pages upon successful validation.

Additional Comments: The password shown in the MongoDB database is hashed in order to ensure secure authentication, however, the user needs to input the correct password to login.

4. Authtoken Login

Test #1: Tenant

Whenever a user logs into his account, he receives an auth token, which can be used to log him in again if he does so within 5 hours.

```
> localStorage
<- Storage {authToken: 'eyJhbGciOiJIUzI1NiIsInR5c
  CI6IkpxVCJ9.eyJpZCI6IjY3Z...cyOX0.4w5T6ZmGF06_WK
  raMppp_40yiq5qd-0vd596K9tyFk', length: 1}
```

The auth token was stored successfully.

The screenshot shows a user profile page. At the top, there's a navigation bar with the Roomble logo, Home, Dashboard, Messages, Find Property, Find Flatmate, and a user icon. Below the navigation is a large circular placeholder for a profile picture. To the right of the placeholder, a message says "You have not set up a description yet". On the far right, there's a list of user details with dropdown menus for selection:

Name	:	Shlok Jain
Email	:	shlokjain0177@gmail.com
City	:	Mumbai
Locality	:	Andheri
Gender	:	Male ♂
Are you seeking a flatmate?	:	Yes ✓
Do you drink/smoke?	:	No ❌
Food Preferences	:	Veg 🥑
Do you have pets?	:	No ❌

Successful login to the account on reload.

Test #2: Landlord

Whenever a landlord logs into his account, he receives an auth token, which can be used to log him in again if he does so within 5 hours.

```
> localStorage
<  Storage {authToken: 'eyJhbGciOiJIUzI1NiIsInR5c
  CI6IkpxVCJ9.eyJpZCI6IjY3Z...I1MX0.UQchy_U-YT5LbS
  HZPNrCGrHF_ciSqdrYeaDLoJdWLFM', Length: 1}
```

The auth token was stored successfully.



Successful login to account on reload.

Test Owner: Shlok Jain

Test Date: 1/4/2025

Test Results: Feature working as expected

5. View Self Profile

Module Details: The feature to view self-profile can be accessed at /tenant-profile-page for the Tenant account. The details are consistent across backend and frontend.

Test #1: Tenant

```

_id: ObjectId('67eb6f65f35d79d0a99ccba3')
name : "Shlok Jain"
type : "tenant"
email : "shlokjain0177@gmail.com"
password : "$2b$10$MRHLIW3rqCALXeFlDwtW.QtwIUbMS778qhidf0t1N3EbYmOYPIInC"
locality : "Andheri"
city : "Mumbai"
gender : true
smoke : false
veg : true
pets : false
flatmate : true
description : "This user hasn't setup a description yet"
▶ conversations : Array (empty)
▶ bookmarks_tenants : Array (empty)
▶ bookmarks_property : Array (empty)
Images : "http://127.0.0.1:3000/Pictures/Default.png"
▶ reviews : Array (empty)
__v : 0

```

Name	:	Shlok Jain
Email	:	shlokjain0177@gmail.com
City	:	Mumbai
Locality	:	Andheri
Gender	:	Male ♂
Are you seeking a flatmate?	:	Yes ✓
Do you drink/smoke?	:	No ❌
Food Preferences	:	Veg 🥦
Do you have pets?	:	No ❌

[Edit](#)
[Logout](#)

Test #2: Landlord

For landlords, self-profile can be viewed at /landlord-profile-page. The details are consistent across backend and frontend.

```
_id: ObjectId('67eb7acef35d79d0a99ccbd6')
name : "Shlok Jain"
type : "landlord"
email : "shlokjain0177@gmail.com"
password : "$2b$10$YNLFsAzhBIkQ393PD27P7u2V0v1z1Z.eZ.KnonYhbxP6P7ts/Yz00"
▶ propertyList : Array (empty)
▶ conversations : Array (empty)
Images : "http://127.0.0.1:3000/Pictures/Default.png"
▶ reviews : Array (empty)
__v : 0
```



Test Owner: Shlok Jain

Test Date: 1/4/2025

Test Results: Working as expected

6. Edit Profile

Module Details: The tenant profile edit page can be used to edit the tenant's details and can be accessed at /tenant-edit-page.

Test #1: Tenant

Edit Profile



Choose File No file chosen

Done

Full Name	City
Shlok Jain	Mumbai
Email Address	Locality
shlokjain0177@gmail.com	Andheri
Gender	About Me
Male	This user hasn't setup a description yet
Do you drink/smoke?	YES NO
Food Preferences	VEG NON-VEG
Are you seeking flatmates?	YES NO
Do you have pets?	YES NO

The initial details are loading correctly.

Edit Profile



Choose File No file chosen **Done**

Full Name	City
Shlok	Mumbai
Email Address	Locality
shlokjain0177@gmail.com	Bandra
Gender	About Me
Male	Hello everyone. I am Shlok from IIT Kanpur.
Do you drink/smoke?	YES NO
Food Preferences	VEG NON-VEG
Are you seeking flatmates?	YES NO
Do you have pets?	YES NO

Now, if I change some details as shown and click the Done button, the details are sent to the backend API, which updates it accordingly. And the user is redirected to the profile page.

```

_id: ObjectId('67eb6f65f35d79d0a99ccba3')
name : "Shlok"
type : "tenant"
email : "shlokjain0177@gmail.com"
password : "$2b$10$MRHLIW3rqCALxeFdwtW.QtwIUbMS778qhifd0t1N3EbYmOYPInC"
locality : "Bandra"
city : "Mumbai"
gender : true
smoke : true
veg : false
pets : true
flatmate : true
description : "Hello everyone. I am Shlok from IIT Kanpur."
conversations : Array (empty)
bookmarks_tenants : Array (empty)
bookmarks_property : Array (empty)
Images : "http://127.0.0.1:3000/Pictures/Default.png"
reviews : Array (empty)
__v : 0

```

It is correctly updated in the backend, as shown.

Test #2: Landlord

The landlord profile edit page can be accessed at /landlord-edit-page.



Choose File No file chosen

Shlok Jain
shlokjain0177@gmail.com

Full Name	Email Address
Enter new name	Enter new email address
<input type="text" value="Shlok Jain"/>	<input type="text" value="shlokjain0177@gmail.com"/>

Edit

The initial details are loading correctly.



Choose File cropped_image.png

Shlok
shlokjain0177@gmail.com

Full Name	Email Address
Enter new name	Enter new email address
<input type="text" value="Shlok"/>	<input type="text" value="shlokjain0177@gmail.com"/>

Edit

Now, I will change the profile photo and name as shown. If I click the edit button, the details will be sent to the backend and updated accordingly.

```
_id: ObjectId('67eb7acef35d79d0a99ccbd6')
name: "Shlok"
type: "landlord"
email: "shlokjain0177@gmail.com"
password: "$2b$10$YNLFsAzhBIkQ393PD27P7u2V0v1z1Z.eZ.KnonYhbxP6P7ts/Yz00"
▶ propertyList: Array (empty)
▶ conversations: Array (empty)
Images: "http://localhost:3000/Pictures/landlord/67eb7acef35d79d0a99ccbd6.png"
▶ reviews: Array (empty)
__v: 0
```

Also, the updated image is stored in the backend server folder of /Pictures/landlord/{id}.png, as shown.



Test Owner: Shlok Jain

Test Date: 1/4/2025

Test Results: The landlord and tenant's edit feature is working correctly.

7. Messaging

Module Details: One of the important features of our app is the real-time chat feature between any two users. We store the conversation history between pairs of users in the conversation model. If you want to chat with another user, click the message button on their profile. This will create a new conversation and start appearing on the Messages page.

The screenshot shows a user profile page on the Roomble app. At the top, there's a navigation bar with the Roomble logo, 'Home', 'Dashboard', 'Messages', 'Find Property', 'Find Flatmate', and a user icon. Below the navigation bar, on the left, is a 'Profile Information' section containing the following details:

- Name: Saksham Verma
- City: Mumbai
- Locality: Bandra
- Gender: Male ♂
- Do you drink or smoke? Yes ☺
- Food Preferences: Non-Vegetarian 🍜
- Do you have a pet? Yes 🐶

On the right, there's a large profile card for 'Saksham Verma' with a placeholder image. The card displays the name 'Saksham Verma' and gender 'Male'. Below the card, a message states 'This user hasn't setup a description yet'. At the bottom of the profile card are three buttons: a bookmark icon, a 'Review' button with a star icon, and a 'Message' button with a speech bubble icon.

This creates a new element in the Conversations database and redirects the user to the chat page, as shown.

```
_id: ObjectId('67ebb708a115332bbd4598aa')
▼ members : Array (2)
  0: ObjectId('67ebb688a115332bbd459875')
  1: ObjectId('67ebae9765569fbc09fbfafe')
► messages : Array (empty)
  __v : 0
```

The screenshot shows the Roomble application's messaging interface. At the top, there is a navigation bar with links for Home, Dashboard, Messages, Find Property, and Find Flatmate. On the far right of the header is a user profile icon. Below the header, a sidebar titled "People" contains a search bar and a list of users. One user, "Saksham Verma", is highlighted with a blue background and shows an "offline" status. The main area displays a message thread with Saksham Verma. The first message from Saksham is "Click to start the chat". A message input field at the bottom allows users to type a message, with a smiley face icon and a send button.

The real-time messaging feature is also working correctly, the online/offline status is handled and updated correctly using web sockets (socket.io)

This screenshot shows a different conversation within the Roomble messaging interface. The top navigation bar and sidebar are identical to the previous screenshot. The main message thread shows a message from "Hitarth Makawana" to "Shlok" at 17:48, stating, "Hello, How are you doing? I saw your profile. I think we are a good match for flatmates. What do you think?". Shlok responded at 17:48 with "Hello Shlok". The message input field at the bottom is visible again.

Test Owner: Shlok Jain

Test Date: 1/4/2025

Test Results: The full messaging pipeline works correctly with all its features. The message button on the profile can be used to start a conversation with another user. Then, WebSockets are being used for real-time communication and updating users' online/offline status, which are also tested and working as expected.

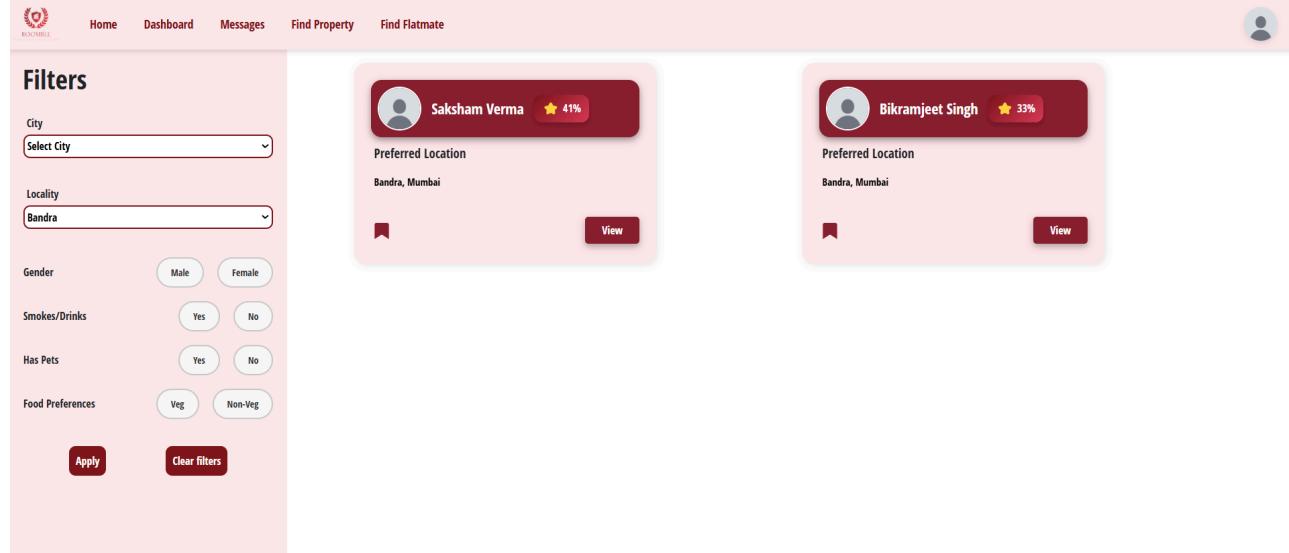
8. Find Flatmates

Module Details: The correctly filtered and sorted search results from the backend are successfully shown in the frontend

Test Owner: Aarsh Jain

Test Date: 01/04/2025

Test #1



The search results are filtered to show only the flatmates with location preference "Bandra".

The backend sends the filtered data correctly and in sorted order of the compatibility score, which the frontend displays accurately.

Test #2

On clicking clear filters, the filters go back to default and we can see the non filtered search results correctly.

Name: Hitarth Makawana	SearchFlatmatesFilter.jsx:61
Compatibility Score: 41	
Name: Saksham Verma	SearchFlatmatesFilter.jsx:61
Compatibility Score: 41	
Name: Shlok Jain	SearchFlatmatesFilter.jsx:61
Compatibility Score: 34	
Name: Bikramjeet Singh	SearchFlatmatesFilter.jsx:61
Compatibility Score: 33	
Name: Aurrie Martinez	SearchFlatmatesFilter.jsx:61
Compatibility Score: 32	
Name: Aritra Ray	SearchFlatmatesFilter.jsx:61
Compatibility Score: 26	

Checking with the backend we get the following sorted data as expected.

Test Results: The Find Flatmates Page is working correctly without any bugs.

9. Adding/Removing Bookmarks

Module Details: This module includes the tests for adding and removing bookmarks for the properties and flatmates the tenant is interested in. The bookmarks are added to the MongoDB database and backend in an array entitled bookmarks_tenants and bookmarks_property.

Test Owner: Hitarth Makawana

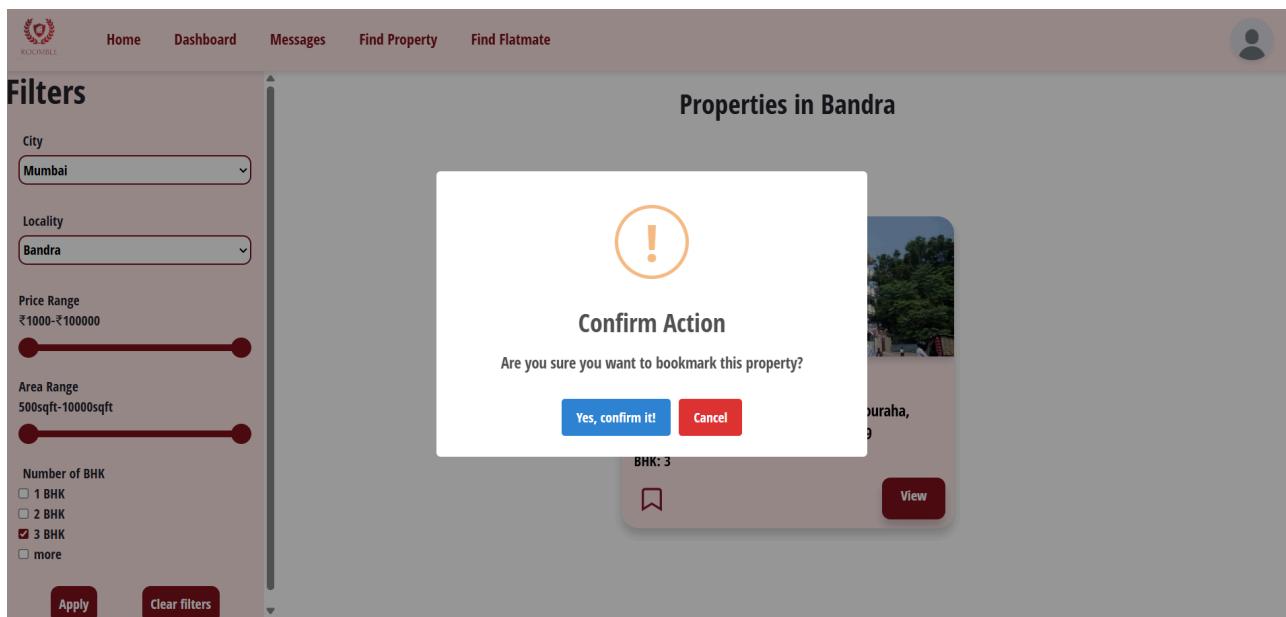
Test Date: 01/04/2025

Test #1: Testing add/delete bookmarks for properties

The screenshot shows the Roombie application's user interface. On the left, there is a sidebar titled "Filters" containing dropdown menus for "City" (set to Mumbai) and "Locality" (set to Bandra), and sliders for "Price Range" (₹1000-₹100000) and "Area Range" (500sqft-10000sqft). Below these are checkboxes for "Number of BHK" with options for 1 BHK, 2 BHK, 3 BHK (which is checked), and more. At the bottom of the sidebar are two buttons: "Apply" and "Clear filters".

The main content area is titled "Properties in Bandra" and displays a single property listing. The listing includes a thumbnail image of a multi-story building, the price "15400", the address "116/628A, Rawatpur Nanak Factory Chouraha, Bandra, Mumbai, Maharashtra - 208019", the BHK count "BHK: 3", and a "View" button. A yellow callout box highlights the "View" button and the "15400" price. To the right of the "View" button is a bookmark icon (a red ribbon-like shape).

Property searched using Find Property and clicked on the bookmark icon for the interested property.



The tenant is prompted with a popup asking for confirmation for the bookmark to be added, the tenant clicks on “Yes, confirm it!” and the bookmark is showed on the tenant dashboard as well as the backend database is updated.

```

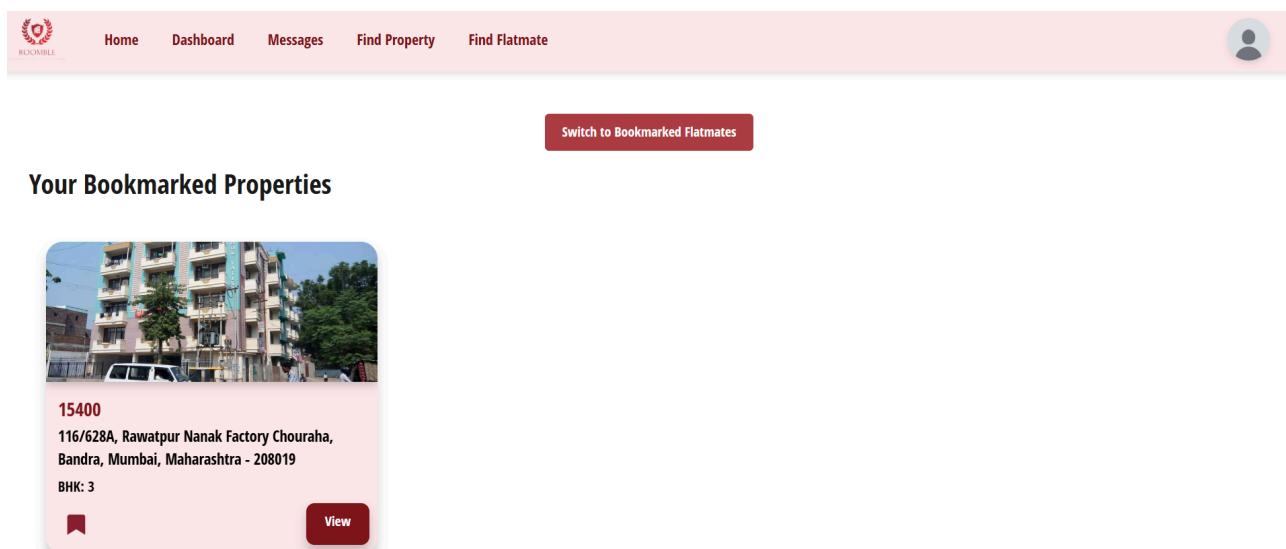
▶ _id: ObjectId('67eb77a567dfab515f609f03')
  name : "Hitarth Makawana"
  type : "tenant"
  email: "hitarthkm23@iitk.ac.in"
  password : "$2b$10$TbbAwXLrwlqTD5i1Y3FKru27WrkEU8jWyC07/DyisNS3nA0CEtaNK"
  locality : "Andheri"
  city : "Mumbai"
  gender : true
  smoke : false
  veg : true
  pets : true
  flatmate : true
  description : "This user hasn't setup a description yet"
  ▶ conversations : Array (empty)
  ▶ bookmarks_tenants : Array (1)
    0: "67eb834d39dab323e423ab0f"
  ▶ bookmarks_property : Array (1)
    0: "67eb81fa67dfab515f609f5c"
    Images : "http://127.0.0.1:3000/Pictures/Default.png"
  ▶ reviews : Array (empty)
  __v : 5

```

The MongoDB database is updated, the property id is added to the bookmarks_property array.

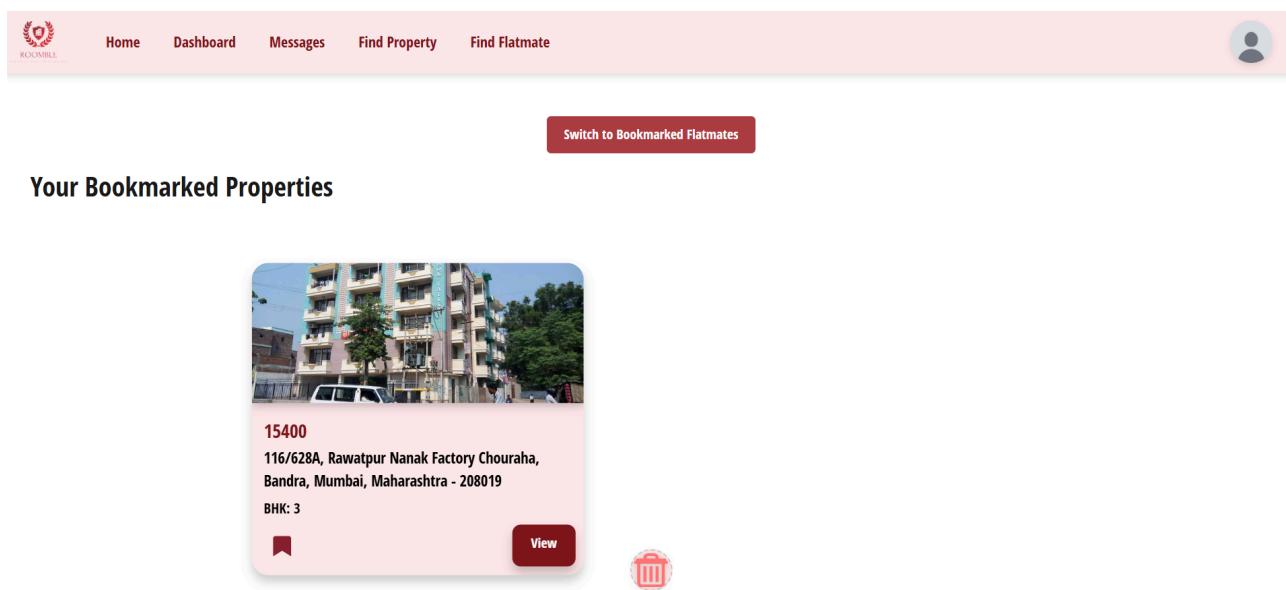
Software Test Document for Roombie

102



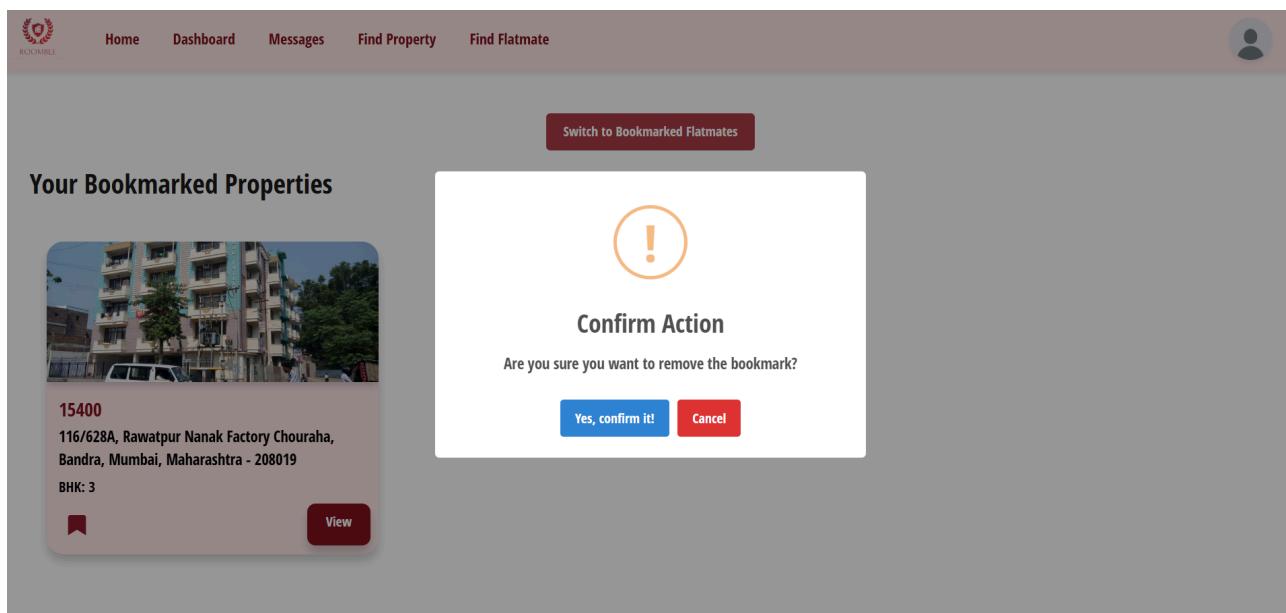
The screenshot shows the 'Your Bookmarked Properties' section of the Roombie tenant dashboard. It displays a property listing for a three-bedroom flat (BHK: 3) at 116/628A, Rawatpur Nanak Factory Chouraha, Bandra, Mumbai, Maharashtra - 208019, with a price of 15400. The listing includes a thumbnail image of the building, the address, the price, and the room type. Below the listing are two buttons: a red bookmark icon and a dark red 'View' button. At the top of the dashboard, there is a navigation bar with links for Home, Dashboard, Messages, Find Property, and Find Flatmate, along with a user profile icon.

The Property is bookmarked and is visible on the tenant dashboard under the Bookmarked Properties heading.

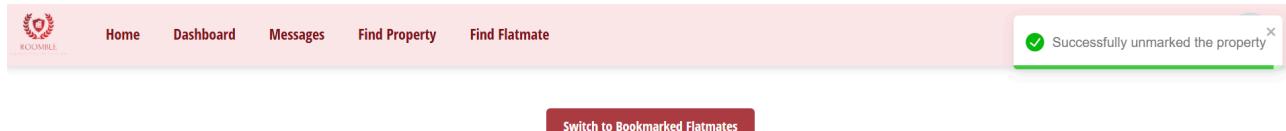


This screenshot is similar to the one above, showing the 'Your Bookmarked Properties' section. It features the same property listing for a three-bedroom flat at 116/628A, Rawatpur Nanak Factory Chouraha, Bandra, Mumbai, Maharashtra - 208019, with a price of 15400. The listing includes the building thumbnail, address, price, and room type. Below the listing are a red bookmark icon and a dark red 'View' button. A new element is a red trash bin icon located to the right of the 'View' button, indicating a delete function. The dashboard's navigation bar and user profile icon are also present at the top.

The tenant can drag and drop the bookmarked property to the bin icon to remove the bookmark.



The tenant is prompted with the confirmation message for the bookmark to be removed. The tenant clicks on “Yes, confirm it!” and the bookmark is removed, the dashboard is updated and the property is no longer visible, the backend is updated too.



The user is prompted with the notification that the bookmark is unmarked successfully.

Test #2: Testing add/delete bookmarks for flatmates

Flatmates searched using Find Flatmate, and clicked on the bookmark icon for the interested flatmate.

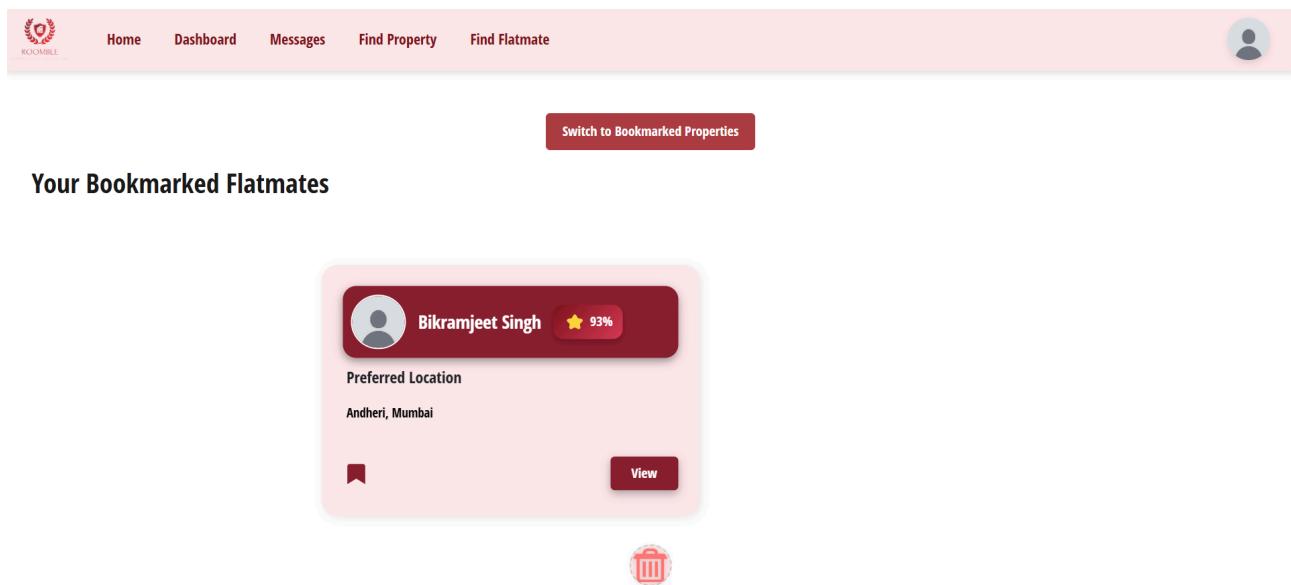
The tenant is prompted with a popup asking for confirmation for the bookmark to be added, the tenant clicks on “Yes, confirm it!” and the bookmark is showed on the tenant dashboard as well as the backend database is updated.

```
▶ _id: ObjectId('67eb77a567dfab515f609f03')
  name : "Hitarth Makawana"
  type : "tenant"
  email : "hitarthkm23@iitk.ac.in"
  password : "$2b$10$TBbAwXlw1qTD5i1Y3FKru27WrkEU8jWyC07/DyisNS3nA0CEtaNK"
  locality : "Andheri"
  city : "Mumbai"
  gender : true
  smoke : false
  veg : true
  pets : true
  flatmate : true
  description : "This user hasn't setup a description yet"
  ▶ conversations : Array (empty)
  ▶ bookmarks_tenants : Array (1)
    0: "67eb834d39dab323e423ab0f"
  ▶ bookmarks_property : Array (1)
    0: "67eb81fa67dfab515f609f5c"
  Images : "http://127.0.0.1:3000/Pictures/Default.png"
  ▶ reviews : Array (empty)
  __v : 5
```

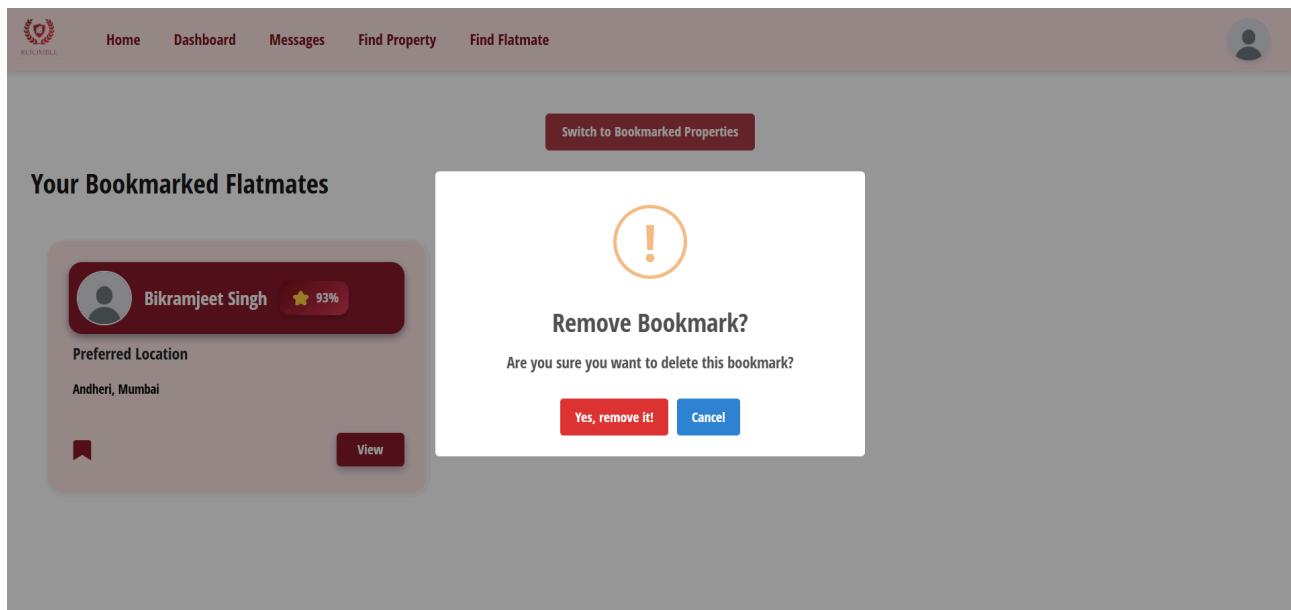
The MongoDB database is updated, the flatmate id is added to the bookmarks_tenants array.

The screenshot shows the Roomble tenant dashboard. At the top, there is a navigation bar with links for Home, Dashboard, Messages, Find Property, and Find Flatmate. On the right side of the header is a user profile icon. Below the header, a button labeled "Switch to Bookmarked Properties" is visible. The main content area is titled "Your Bookmarked Flatmates". It displays a card for a flatmate named Bikramjeet Singh, who has a 93% rating. The card includes a profile picture, the flatmate's name, their rating, and their preferred location (Andheri, Mumbai). There are "View" and "Bookmark" buttons at the bottom of the card.

The Flatmate is bookmarked and is visible on the tenant dashboard under the Bookmarked Flatmates heading.



The tenant can drag and drop the bookmarked flatmate to the bin icon to remove the bookmark.



The tenant is prompted with the confirmation message for the bookmark to be removed. The tenant clicks on "Yes, confirm it!" and the bookmark is removed, the dashboard is updated and the flatmate is no longer visible, the backend is updated too.



Your Bookmarked Flatmates

The user is prompted with the notification that the bookmark is unmarked successfully.

Test Results: The user clicks on bookmark icon for interested properties and flatmates, after which, the user is prompted with the confirmation message to add/delete bookmark, to which, when the user clicks on “Yes, confirm it!”, the bookmark is respectively added or removed, and the user is notified with a success message that the bookmark has been added or removed. Moreover, the bookmarked properties and flatmates are visible at the tenant dashboard. Similarly, the properties/flatmates are removed from the dashboard after removing the bookmarks respectively.

10. Find Property

The tenant can apply filters for finding a property of interest

Module Details: This module contains the tests for validating the filters added during searching for a property of interest. The backend filters the properties based on filters provided during the search. The backend responds with an array of sorted properties according to the matched attributes with filters provided, which are displayed to the tenant. If no property satisfies the given filters, the search results display the message of property not found.

Test Owner: Hitarth Makawana

Test Date: 01/04/2025

The screenshot shows the Roombie application's search interface. On the left, there is a sidebar titled "Filters" with dropdown menus for "City" (set to "Mumbai") and "Locality" (set to "Bandra"), and sliders for "Price Range" (₹19000 - ₹100000) and "Area Range" (1350sqft - 10000sqft). Below these are checkboxes for "Number of BHK": "1 BHK" (unchecked), "2 BHK" (unchecked), "3 BHK" (checked), and "more" (unchecked). At the bottom of the sidebar are two buttons: "Apply" and "Clear filters".

The main area is titled "Properties in Bandra" and displays a single property listing. The listing includes a thumbnail image of a multi-story residential building, the price "20000", the address "116/628A, Rawatpur near Nanak Factory Chouraha, Bandra, Mumbai, Maharashtra - 208019", the BHK count "BHK: 3", and a "View" button.

The tenant applies filters by choice and the search results display relevant properties

```
_id: ObjectId('67ebc94c9a575e22057f043a')
city : "Mumbai"
town : "Bandra"
address : "116/628A, Rawatpur near Nanak Factory Chouraha, Bandra, Mumbai, Maharashtra - 208019"
area : 1400
bhk : 3
description : "This is a very good flat"
amenities : Array (3)
  0: "PS5 console"
  1: " no grass to touch"
  2: " pool table"
price : 20000
available : true
Images : Array (4)
  0: "http://localhost:3000/Pictures/property/67ebc94c9a575e22057f043a/0.png"
  1: "http://localhost:3000/Pictures/property/67ebc94c9a575e22057f043a/1.png"
  2: "http://localhost:3000/Pictures/property/67ebc94c9a575e22057f043a/2.png"
  3: "http://localhost:3000/Pictures/property/67ebc94c9a575e22057f043a/3.png"
lat : 19.05952552667832
lng : 72.83009995687377
reviews : Array (empty)
landlord : ObjectId('67ebc87b9a575e22057f041d')
createdAt : 2025-04-01T11:09:00.316+00:00
updatedAt : 2025-04-01T11:09:00.316+00:00
__v : 0
```

The backend database gives the properties array (here only single element) which is sorted according to the filters provided.

The screenshot shows the Roombie app's search interface. On the left, a sidebar titled "Filters" contains dropdowns for "City" (Mumbai), "Locality" (Juhu), and "Number of BHK" (with checkboxes for 1 BHK, 2 BHK, 3 BHK, and more, where 3 BHK is checked). It also includes sliders for "Price Range" (₹19000 - ₹100000) and "Area Range" (1350sqft-10000sqft). At the bottom of the sidebar are "Apply" and "Clear filters" buttons. The main area displays a message: "Sorry! No property matched with your filters". Below this, a section titled "Other Properties You May Be Interested In" shows a thumbnail of a multi-story building with the text "20000" and "116/628A, Rawatpur near Nanak Factory Chouraha, Bandra, Mumbai, Maharashtra - 208019" along with a "View" button.

If the tenant applies some filters that don't match exactly but matches some other properties, then the user is suggested for the matched properties that he/she may be interested in.

This screenshot is similar to the one above, showing the same search interface with filters applied. The "Number of BHK" section now includes a checkbox for "more" which is checked. The main area again displays the message "Sorry! No property matched with your filters".

If no property exists which satisfies the filters provided, then the search results show there is no property that matched the filters.

Test Results: When the tenant applies filters by choice, the database responds with an array of properties that are sorted according to the filters provided and the properties are displayed in the same order to the tenant. Moreover, if some filters don't match the exact property, but matches with some other properties, then the

user is suggested with the matched properties that he/she may be interested in. If the tenant gives some filters that do not match with any property, then the search results display a message that no property match the filters.

11. Adding and Viewing Reviews

The tenant can review both flatmates as well as properties and it is open to all review system.

Module Details: This module contains tests for adding and viewing reviews for the flatmates and properties. When the tenant clicks on review, he can rate the flatmate/property out of five stars and write a review message.

Test Owner: Hitarth Makawana

Test Date: 01/04/2025

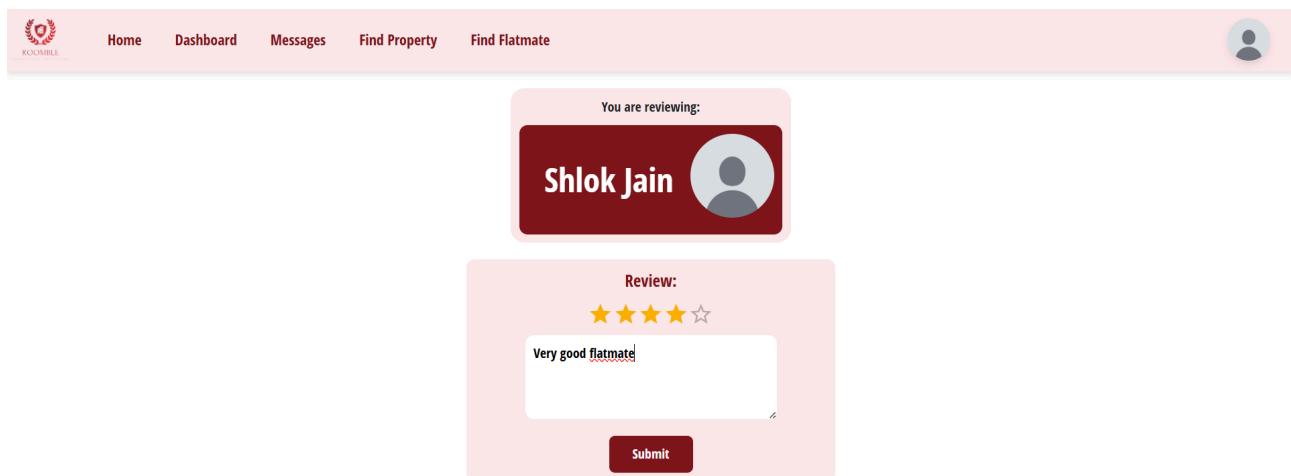
Test #1: Adding/Viewing review for a flatmate

The screenshot shows the Roombie application's user profile page. At the top, there is a navigation bar with links for Home, Dashboard, Messages, Find Property, and Find Flatmate. On the right side of the header is a user icon. Below the header, on the left, is a 'Profile Information' card containing the following details:

- Name: Shlok Jain
- City: Mumbai
- Locality: Andheri
- Gender: Male ♂
- Do you drink or smoke? No ☺
- Food Preferences: Vegetarian 🥦
- Do you have a pet? No ☺

On the right, there is a larger card for 'Shlok Jain' which includes his name, gender, a placeholder profile picture, and a message stating 'This user hasn't setup a description yet'. At the bottom of this card are three buttons: a bookmark icon, a 'Review' button, and a 'Message' button.

The user clicks on review button and is prompted with a form to submit reviews.



The tenant can review the flatmate out of 5 stars, write a message, and click the submit button to submit the review.

The screenshot shows the updated flatmate profile for Shlok Jain. A green success message 'Review submitted successfully' is displayed above the profile. Below the profile, a note says 'This user hasn't setup a description yet'. At the bottom of the profile card are 'Review' and 'Message' buttons. In the reviews section, a review by 'Hitarth Makawana' is listed, showing a 5-star rating and the text 'Very good flatmate'.

The review section of the flatmate is updated, and the current reviewer is prompted with a success message saying the review was submitted successfully.

```

_id: ObjectId('67ebb688a115332bbd459875')
name : "Shlok Jain"
type : "tenant"
email : "shlokjain0177@gmail.com"
password : "$2b$10$f4iurr8NvkvyUE09melQ50a0s6d/ZxgHjYJxdxdfUdmPGI/L0vrES"
locality : "Andheri"
city : "Mumbai"
gender : true
smoke : false
veg : true
pets : false
flatmate : true
description : "This user hasn't setup a description yet"
conversations : Array (2)
bookmarks_tenants : Array (empty)
bookmarks_property : Array (empty)
Images : "http://127.0.0.1:3000/Pictures/Default.png"
reviews : Array (1)
  ▾ 0: Object
    reviewer : ObjectId('67ebc4919a575e22057f036f')
    rating : 4
    comment : "Very good flatmate"
    reviewertype : "tenant"
  --v : 4
  
```

The MongoDB database is updated and the details of the reviewer and the review added by him/her is stored in the database.

Test #2: Adding/Viewing reviews for properties



₹20000/Month

116/628A, Rawatpur near Nanak Factory Chouraha, Bandra, Mumbai, Maharashtra - 208019

Description
This is a very good flat

Amenities

- PS5 console
- no grass to touch
- pool table

Area
1400 sqft

BHK
3

Availability
Available for renting

[Review](#) [Contact Owner](#) [View on Maps](#) [Interested](#)

Reviews

No reviews yet.

The user clicks on review button and is prompted with a form to submit reviews.

The screenshot shows a property listing for a flat. At the top left is an exterior view of a multi-story building with a sign that reads "ASHOK VILLA". To the right of the image are sections for "Description" (This is a very good flat), "Amenities" (PS5 console, no grass to touch, pool table), and a "Leave a review" button. A modal window is open, prompting the user to leave a review with a 5-star rating and the message: "Very good property. Best for CS majors because it doesn't have shower". Below the modal are buttons for "Submit", "Contact Owner", "View on Maps", and "Interested".

₹20000/Month
116/628A, Rawatpur near Nanak Factory Chouraha, Bandra, Mumbai, Maharashtra - 400051

Reviews
No reviews yet.

The tenant can review the property out of 5 stars and can write a review message and click the submit button to submit the review.

The screenshot shows the same property listing as above, but now with a success message: "Review added successfully" in a green box. The "Leave a review" modal is still present, and the "Submit" button has been clicked. The bottom section remains the same with buttons for "Review", "Contact Owner", "View on Maps", and "Interested".

₹20000/Month
116/628A, Rawatpur near Nanak Factory Chouraha, Bandra, Mumbai, Maharashtra - 400051

Reviews

Hitarth Makawana
★★★★☆
Very good property, Best for CS majors because it doesn't have shower

The review section of the property is updated and the current reviewer is prompted with a success message saying review submitted successfully.

```
_id: ObjectId('67ebc94c9a575e22057f043a')
city : "Mumbai"
town : "Bandra"
address : "116/628A, Rawatpur near Nanak Factory Chouraha, Bandra, Mumbai, Maharashtra"
area : 1400
bhk : 3
description : "This is a very good flat"
amenities : Array (3)
  price : 20000
  available : true
  > Images : Array (4)
    lat : 19.05952552667832
    lng : 72.83009995687377
  > reviews : Array (1)
    > 0: Object
      reviewer : ObjectId('67ebc4919a575e22057f036f')
      reviewertype : "tenant"
      rating : 4
      comment : "Very good property, Best for CS majors because it doesn't have shower"
      landlord : ObjectId('67ebc87b9a575e22057f041d')
      createdAt : 2025-04-01T11:09:00.316+00:00
      updatedAt : 2025-04-01T11:44:50.925+00:00
    --v : 2
```

The MongoDB database is updated and the details of the reviewer and the review added by him/her is stored in the database.

Test Results: When the tenant clicks on review button under any of the flatmate/property profiles, then the tenant has to fill a review form where he/she has to rate the flatmate/property out of 5 stars and write a review on them. After submitting the form, the tenant is notified with the success message that the review submitted successfully.

12. Add Property

Module Details: Tested the Add Property Functionality for Landlord, to see if the details the Landlord adds in the frontend are correctly getting stored in the backend.

Test Owner: Aarsh Jain

Test Date: 01/04/2025

Add Property

Upload Photos *

Drag & Drop image uploading

Drag & Drop image here or [Browse](#)



Description

hola amigos!

BHK *

4

Area(sqft) *

2000

Rent(Per Month) *

10000

City *

Mumbai

Location *

Dahisar

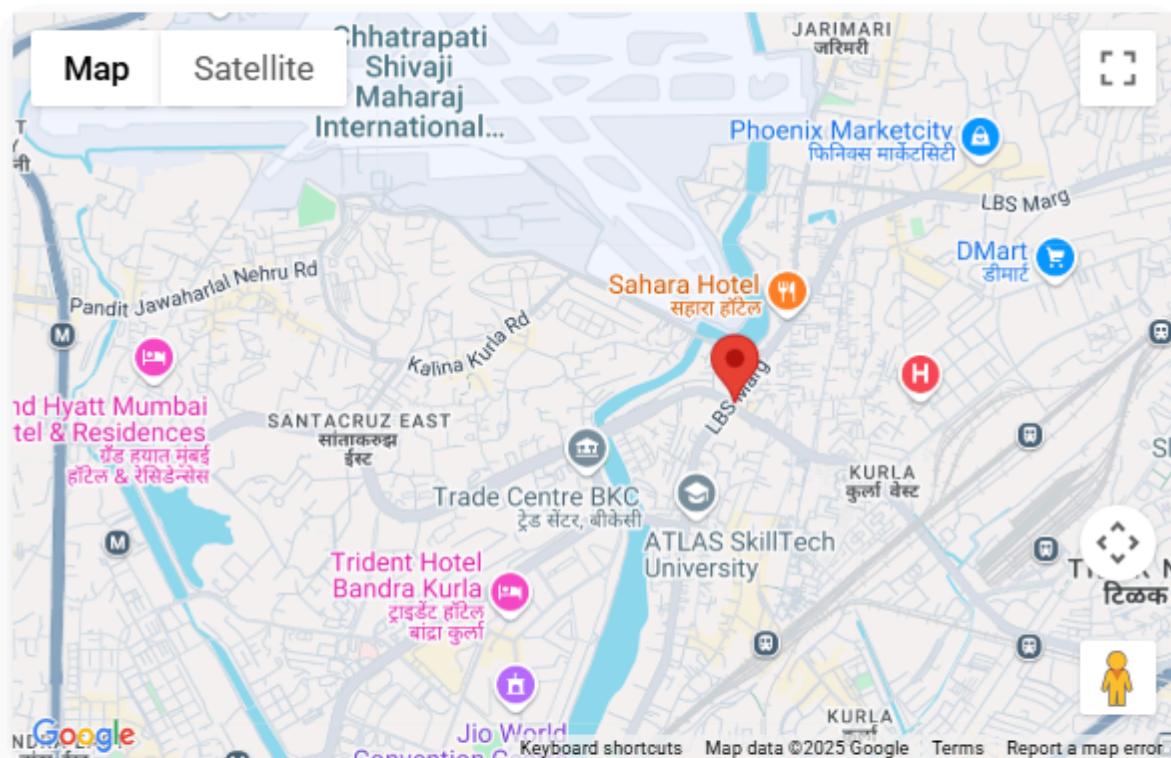
Address *

Hall 12, D112, Dahisar

Amenities

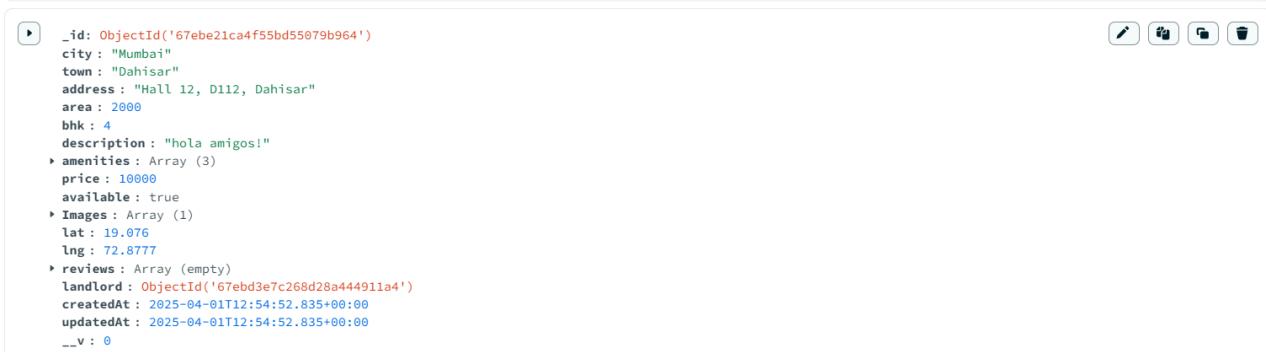
gym, pool, games

Pick the location on Maps

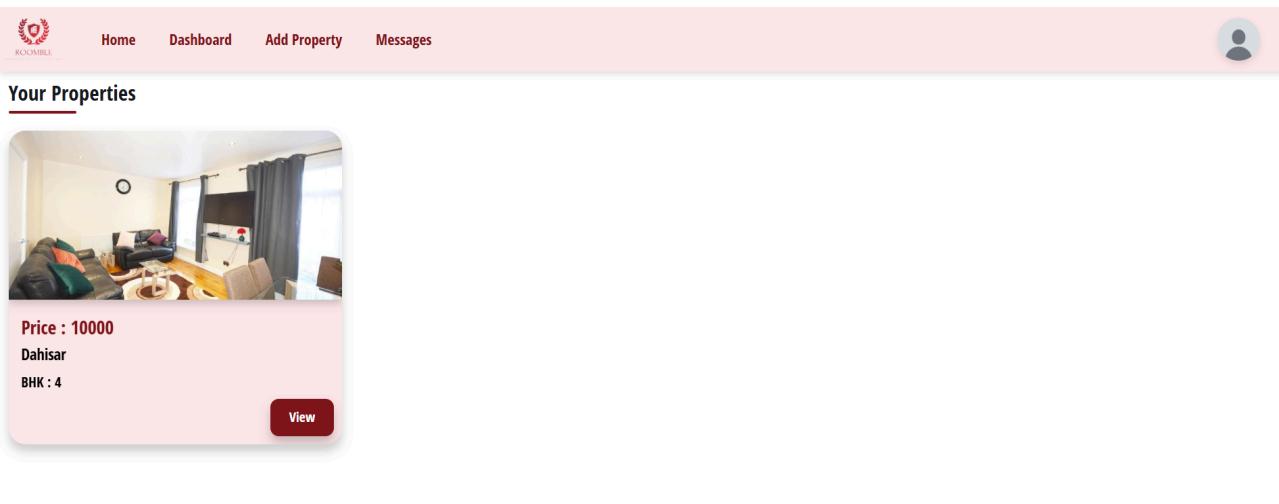


Submit

The landlord when clicks on Submit, the property should be successfully added to the backend.



```
_id: ObjectId('67ebe21ca4f55bd55079b964')
city: "Mumbai"
town: "Dahisar"
address: "Hall 12, D112, Dahisar"
area: 2000
bhk: 4
description: "holo amigos!"
amenities: Array (3)
  price: 10000
  available: true
  Images: Array (1)
    lat: 19.076
    lng: 72.877
  reviews: Array (empty)
  landlord: ObjectId('67ebd3e7c268d28a444911a4')
  createdat: 2025-04-01T12:54:52.835+00:00
  updatedAt: 2025-04-01T12:54:52.835+00:00
  __v: 0
```



Test Results: The property gets stored successfully with the correct details, and gets updated in the user dashboard.

13. Edit Property

Module Details: Tested the Edit Property Functionality for Landlord, to see if the details the Landlord changes in the frontend are correctly getting updated in the backend.

Test Owner: Aarsh Jain

Test Date: 01/04/2025

Add Property

Upload Photos *

Drag & Drop image uploading

Drag & Drop image here or [Browse](#)



Description

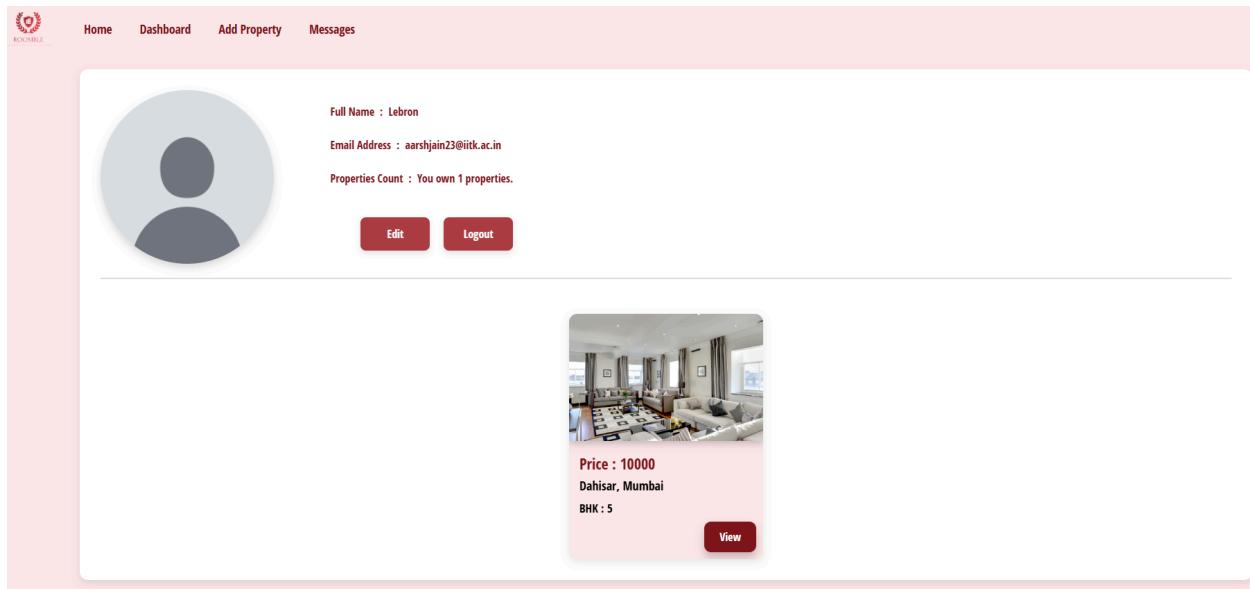
hello guys, I am Aarsh Jain !

BHK *

5

Area(sqft) *

2000



The Landlord clicked on edit, got redirected to this page, and changed the following details. After clicking on submit, I got redirected to the profile page, where the pic and changed details are updated.

```

▶ _id: ObjectId('67ebe21ca4f55bd55079b964')
city: "Mumbai"
town: "Dahisar"
address: "Hall 12, D112, Dahisar"
area: 2000
bhk: 5
description: "hello guys, I am Aarsh Jain !"
amenities: Array (1)
price: 10000
available: true
Images: Array (1)
lat: 19.076
lng: 72.8777
reviews: Array (empty)
landlord: ObjectId('67ebd3e7c268d28a444911a4')
createdAt: 2025-04-01T12:54:52.835+00:00
updatedAt: 2025-04-01T13:13:32.462+00:00
__v: 3

```

Test Results: The property details that were changed in the front end were edited and updated successfully in the backend.

14. Mailing the landlord if the tenant clicks on “Interested” property

The tenant can show interest in a property by clicking on the interested button, which notifies the landlord via mail.

Module Details: This module contains tests for sending an email to the landlord whenever the tenant clicks on the interested button for the property he/she is interested in.

Test Owner: Hitarth Makawana

Test Date: 01/04/2025



₹20000/Month

116/628A, Rawatpur near Nanak Factory Chouraha, Bandra, Maharashtra - 208019

Description

This is a very good flat

Amenities

- PS5 console
- no grass to touch
- pool table

Area

1400 sqft

BHK

3

Availability

Available for renting

[Review](#)

[Contact Owner](#)

[View on Maps](#)

[Interested](#)

Reviews

No reviews yet.

The tenant clicks on the Interested button to notify the landlord that he/she is interested.



₹20000/Month

116/628A, Rawatpur near Nanak Factory Chouraha, Bandra, Maharashtra - 208019

Sending email

Owner was notified

Description

This is a very good flat

Amenities

- PS5 console
- no grass to touch
- pool table

Area

1400 sqft

BHK

3

Availability

Available for renting

[Review](#)

[Contact Owner](#)

[View on Maps](#)

[Interested](#)

Reviews

On clicking the interested button, the tenant is notified with the success message “sending email.” When the email is sent to the landlord, it shows the success message “Owner was notified.”

The screenshot shows an email interface with a dark background. The subject line is "Someone is interested in your property" with a blue envelope icon. On the left, there is a circular profile picture placeholder. Below the subject, the email details are listed: From "roomble360@gmail.com", To "hitarthkm23@iitk.ac.in", and Date "Today 17:47". There are two buttons at the bottom: "Summary" and "Headers". The main body of the email contains a greeting message: "Greetings, Hitarth Makawana. Hitarth Makawana is interested in your property at 116/628A, Rawatpur near Nanak Factory Chouraha, Bandra, Mumbai, Maharashtra - 208019. Mail him at hitarthkm23@iitk.ac.in or message him via Roomble."

The landlord is notified via mail that someone is interested in his/her property. The mail includes the details regarding the tenant interested and the property address that the tenant is interested in.

Test Results: When the tenant clicks on the Interested button, the backend sends a success message of sending an email to the landlord who owns the property and notifies the tenant that the mail is sent. Additionally, the landlord is notified over his/her registered email about the details of the interested tenant and the interested property's address.

15. List/Delist Property

The landlord can list/delist his/her properties depending on whether or not a tenant is staying there. All other users can see the property as grey if it is delisted.

Module Details: This module contains tests for checking the list/delist feature used by the landlord to list/delist his properties, depending upon whether the landlord is satisfied with the property being occupied.

Test Owner: Hitarth Makawana

Test Date: 01/04/2025

Software Test Document for Roomble

122

The screenshot shows a property listing on the Roomble platform. The listing includes a large image of a living room with a sofa, a dining area, and a kitchen. The price is listed as ₹10000/Month and the location as Hall 12, D112, Dahisar. The listing details are as follows:

- Description:** hello guys, I am Aarsh Jain !
- Amenities:** • gym, pool, games
- Area:** 2000 sqft
- BHK:** 5
- Availability:** Available for renting

At the bottom, there are four buttons: Edit, Delist, View on Maps, and Delete.

Reviews:
No reviews yet.

The landlord clicks on delist button to delist his/her property

The screenshot shows the same property listing as before, but now it is marked as delisted. A green success message at the top right says "Successfully delisted property". The listing details remain the same as in the previous screenshot.

Reviews:
No reviews yet.

The landlord is notified that the property is delisted



The property is displayed as grey on the landlord dashboard, after delisting.

```

▶ _id: ObjectId('67ebe21ca4f55bd55079b964')
city: "Mumbai"
town: "Dahisar"
address: "Hall 12, D112, Dahisar"
area: 2000
bhk: 5
description: "hello guys, I am Aarsh Jain !"
▼ amenities: Array (1)
  0: "gym, pool, games"
price: 10000
available: false
▼ Images: Array (1)
  0: "http://localhost:3000/Pictures/property/67ebe21ca4f55bd55079b964/0.png"
lat: 19.076
lng: 72.8777
reviews: Array (empty)
landlord: ObjectId('67ebd3e7c268d28a444911a4')
createdAt: 2025-04-01T12:54:52.835+00:00
updatedAt: 2025-04-01T13:18:37.411+00:00
__v: 4

```

The MongoDB database also updates the available attribute as false after delisting.

The screenshot shows a search interface with a sidebar for 'Filters' and a main area for 'Properties in Dahisar'.

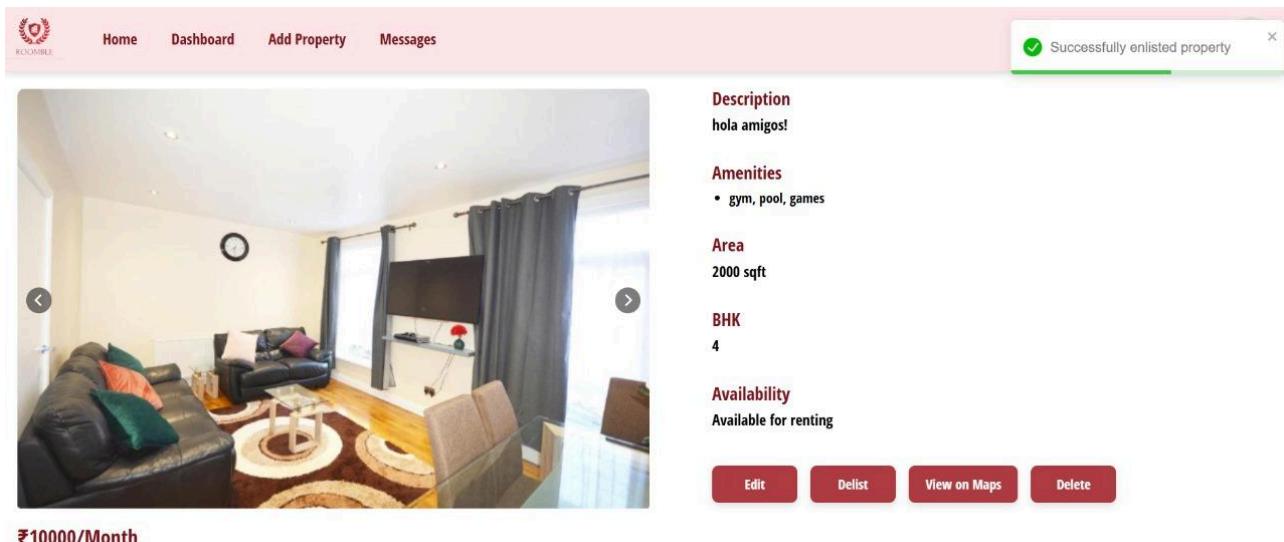
Filters:

- City:** Select City (dropdown menu)
- Locality:** Dahisar (dropdown menu)
- Price Range:** ₹1000 - ₹100000 (sliding scale from ₹1000 to ₹100000)
- Area Range:** 500sqft - 10000sqft (sliding scale from 500sqft to 10000sqft)
- Number of BHK:**
 - 1 BHK
 - 2 BHK
 - 3 BHK
 - more

Properties in Dahisar:

A single property card is shown for a listing in Dahisar. The card includes the price '10000', address 'Hall 12, D112, Dahisar', and BHK '5'. It features a 'View' button and a small thumbnail image of the property interior.

The property is displayed as grey to the tenant also when he/she searches for the property



₹10000/Month

Hall 12, D112, Dahisar

The landlord can similarly click on the list button to list his property back, after which he will be notified that the property is enlisted successfully.

```

    _id: ObjectId('67ebe21ca4f55bd55079b964')
    city: "Mumbai"
    town: "Dahisar"
    address: "Hall 12, D112, Dahisar"
    area: 2000
    bhk: 5
    description: "hello guys, I am Aarsh Jain !"
    amenities: Array (1)
    price: 10000
    available: true
    Images: Array (1)
    lat: 19.076
    lng: 72.8777
    reviews: Array (empty)
    landlord: ObjectId('67ebd3e7c268d28a444911a4')
    createdat: 2025-04-01T12:54:52.835+00:00
    updatedAt: 2025-04-01T13:28:43.347+00:00
    __v: 4
  
```

The MongoDB database also gets updated where the available field is true after enlisting the property.

The screenshot shows a user profile icon at the top right. The main header "Your Properties" is underlined. Below it is a thumbnail image of a room with a sofa, a dining table, and a window with grey curtains. To the left of the image, the text "Price : 10000" is displayed in red, followed by "Dahisar" and "BHK : 4". At the bottom right of the card is a "View" button.

The property is visible again as it earlier was, before delisting to both the landlord and all other users.

Test Results: When the landlord clicks on the delist button, the property is delisted, the availability of the property on the MongoDB database is set to false, and then the property is visible as grey to all users, even the landlord that owns it. Similarly, after clicking on the list button, the MongoDB database updates the availability field to true, and hence, all the users can see the property again as it was before delisting. The landlord is notified every time with a success message for enlisting/delisting his/her property.

16. Delete Property

Module Details: We have implemented a delete property feature; the landlord, who will be the property owner, will get the option to delete it from the Database.

The screenshot shows a user profile icon at the top right. The main header "Your Properties" is underlined. Below it is a thumbnail image of a room with a bed and a doorway. To the left of the image, the text "Price : 12000" is displayed in red, followed by "Andheri" and "BHK : 3". At the bottom right of the card is a "View" button.

So, consider an initial version of the landlord dashboard, as we can see there is a property already added.

The screenshot shows a property listing on the Roombie dashboard. At the top, there's a navigation bar with the Roombie logo, 'Home', 'Dashboard', 'Add Property', 'Messages', and a user profile icon. Below the navigation is a large image of a room with a blue wall and a doorway leading to another room. To the right of the image, there's a 'Description' section with the text 'This is a very good place'. Underneath it is an 'Amenities' section with a bulleted list: 'Gym' and 'Elevator'. The 'Area' is listed as '1234 sqft', and the 'BHK' is '3'. In the 'Availability' section, it says 'Available for renting'. At the bottom of the listing are four buttons: 'Edit', 'Delist', 'View on Maps', and 'Delete'. Below the listing, the text '₹12000/Month' and the address '8, Yogendra society, behind Apollo International School, near saurashtra bank society' are displayed.

After going to this property page, I will get the button to delete it.

The screenshot shows the 'Your Properties' section of the dashboard. It displays a message 'No properties to show'. On the right side, there are two green notification boxes: one saying 'Property deleted successfully' and another saying 'Deleting...'. This indicates that a property has been successfully deleted from the database.

Clicking the delete button will remove the property from the database using the API. As we can see, the property is removed from the dashboard after deleting.

Test Owner: Surepally Pranaysriharsha

Test Date: 1/4/2025

Test Results: The delete functionality is working correctly.

4 System Testing

Functional Requirements

1. Requirement: A new user should be able to sign up by creating an account and providing the required details, such as name and email ID. If the user is a Tenant, they must also answer a few additional questions, including City, Locality, Gender, Smoking/Drinking preferences, Pets, Food preferences, and whether they are looking for flatmates. Finally, the user must set a valid password to complete the registration.

The sign-up functionality allows new users to create accounts by providing the required details as outlined above. To validate its effectiveness, we designed a comprehensive set of test cases covering a variety of scenarios. These tests were conducted by manually navigating to the sign-up page, entering both valid and invalid data into the respective fields, and observing the system's responses. We confirmed that users could successfully register by providing all necessary details and setting a valid password, resulting in the successful creation of their account. In addition, we tested edge cases such as incomplete or incorrect information, duplicate email addresses, and passwords that did not meet the required length constraints (minimum 6 and maximum 10 characters, inclusive) to ensure proper validation and error handling.

One notable case involved the use of the same email address to sign up as both a Tenant and a Landlord. While this is logically acceptable—since a Landlord could potentially also be a Tenant elsewhere—it introduces the risk of a user reviewing their own Landlord profile from their Tenant account. This scenario should be handled appropriately by restricting users from reviewing their own alternate user-type accounts.

Another issue was observed during the Tenant registration process. When a duplicate email ID is detected on the second page (where personal details are entered), the system displays the error message on the first page. Logically, the system should either prevent the user from progressing past the first page if the email already exists or display the message at the end of the second page after submission.

Aside from these issues, no major bugs or inconsistencies were found during testing, confirming that the sign-up functionality performs as intended and meets the defined requirements.

Test Owner: Bhukya Vaishnavi

Test Date: 04/04/2025

Test Results: The Sign-Up page functioned as expected. The first issue—where a user could review their own Landlord account—was resolved by ensuring that users cannot review a Landlord profile linked to their own email ID. The second issue, related to the incorrect placement of the duplicate email error message during Tenant registration, was addressed by updating the code to display the error on the appropriate sign-up page.

Additional Comments: NA

2. Requirement: The user should be able to sign in through his email ID and password if he is already a member.

The sign-in functionality was thoroughly tested to ensure its reliability and effectiveness. We developed a range of test cases covering various scenarios, including both valid and invalid inputs. These tests were performed by manually navigating to the sign-in page, entering different combinations of email IDs and passwords, and observing the system's responses.

For instance, we verified the successful sign-in process by using valid credentials and confirming that the user was correctly redirected to the Dashboard page. We also tested invalid scenarios such as incorrect email formats, unregistered user credentials, and wrong passwords. In each case, the system displayed appropriate error messages—such as “Wrong Password. Entry Denied” for incorrect passwords and “No User Exists” for non-registered users.

No bugs or issues were encountered during testing, confirming that the sign-in functionality works as intended and meets the specified requirements.

Test Owner: Bhukya Vaishnavi

Test Date: 04/04/2025

Test Results: The Sign-in page worked as expected. We tested it thoroughly. No bug was reported.

Additional Comments: NA

3. Requirement: There should be an option of 'Forgot Password?' in case the user forgets the password. Authentication shall be carried out through OTP verification on the email the user had provided earlier.

The "**Forgot Password**" functionality enables users to reset their password through OTP verification sent to their registered email address. To test this feature, we performed a series of scenarios including entering a valid email, receiving and verifying the OTP, and successfully setting a new password. Logging in with the new credentials confirmed that the process worked as intended.

We also validated error handling for edge cases such as entering an invalid or unregistered email, and incorrect OTPs—each of which triggered appropriate error messages and responses.

The "Forgot Password" functionality performed reliably and met all specified requirements.

Test Owner: Aritra Ambudh Dutta

Test Date: 04/04/2025

Test Results: The "**Forgot Password**" feature is now working correctly, and no bugs have been reported.

Additional Comments: NA

4. Requirement: Upon logging in, the **Tenant Dashboard** should display three main sections: **Messages**, **Find Property**, and **Find Flatmates**.

The dashboard should also allow tenants to view their **bookmarked properties and flatmates**, with the ability to switch between the two using a toggle/switch.

In the **Messages tab**, users can view messages from other users and engage in real-time chat. In the **Find Property tab**, users can browse properties using various filters to refine their search. In the **Find Flatmates tab**, users can search for potential flatmates based on their preferences.

Test Owner: Aritra Ambudh Dutta

Test Date: 04/04/2025

Test Results: The Dashboard page worked as expected. We tested it thoroughly. No bug was reported.

Additional Comments: NA

5. Requirement: Upon logging in, the **Landlord Dashboard** should display two main sections: **Messages, Add Property**.

The dashboard should also allow landlords to view their **added properties**. In the **Messages tab**, users can view messages from other users and engage in real-time chat. In the **Add Property tab**, users can add properties for rent.

Test Owner: Aritra Ambudh Dutta

Test Date: 04/04/2025

Test Results: The Dashboard page worked as expected. We tested it thoroughly. No bug was reported.

Additional Comments: NA

6. Requirement: The dashboards shall also have the **Profile** option (where the user can change or update his other details except email).

The **Edit** (Profile Editing) and **Logout** options on the dashboard underwent different test to validate their functionality. Manual testing confirmed that the Profile Edit option enables users to efficiently update their details, excluding email, with changes accurately reflected in the system. Additionally, the Logout option securely logs users out of their accounts, terminating the current session and redirects them to the login page for re-authentication. No bugs were found.

Test Owner: Aritra Ambudh Dutta

Test Date: 04/04/2025

Test Results: The profile option, edit and logout worked as expected. No bug was reported.

Additional Comments: NA

7. Requirement: Prospective landlords will be able to add their properties with appropriate details.

Using the **Add Property** tab Landlords are able to upload their property details, for example, Photos, Description, BHK, Area(sqft), Rent(per Month), City, Location, Address, Amenities. They can also Pick the location of the property on Maps.

Test Owner: Aritra Ambudh Dutta

Test Date: 04/04/2025

Test Results: The Add property option worked as expected. No bug was reported.

Additional Comments: NA

8. Requirement: Landlords will have the option of editing their listed properties. Under the Dashboard, all the listed properties of the Landlord are shown. By clicking on the **View** button of one of them, Using the **Edit** tab Landlods are able to edit their property details, for example, Photos, Description, BHK, Area(sqft), Rent(per Month), City, Location, Address, Amenities.

Test Owner: Aritra Ambudh Dutta

Test Date: 04/04/2025

Test Results: The Edit property option worked as expected. No bug was reported.

Additional Comments: NA

9. Requirement: Landlords will also have the option of delisting/listing, delete their listed properties.

Under the Dashboard, all the listed properties of the Landlord are shown. By clicking on one of them, using **Delist** button, temporarily they can make the property Unavailable for renting and using **List** they can again make them available. Using **Delete** they can delete the property from database. Using **View on Maps** they can view the exact location of the property.

Test Owner: Aritra Ambudh Dutta

Test Date: 04/04/2025

Test Results: All the options worked as expected. No bug was reported.

Additional Comments: NA

10. Requirement: Tenants will have the option of finding properties.

By clicking on the **Find Property** tab on the Dashboard, Tenants can see the filters available to search for a property of their interest. By choosing the filters City, Locality, Price Range and BHK, we can find the best suitable property of our interest.

After clicking on **View** to see a property, **View on Maps** option gives the option to visit the property on maps. If he/she is interested in renting it, clicking on **Interested** button will send an email to the own informing the name and email of the tenant who's interested into his property. By clicking on **Contact Owner**, a conversation through messages can be initiated between them. Also, a button to **Review** the property is also there.

No bugs or issues were encountered during testing, confirming that the functionality works as intended and meets the specified requirements.

Test Owner: Aritra Ambudh Dutta

Test Date: 04/04/2025

Test Results: All the options worked as expected. No bug was reported.

Additional Comments: NA

11. Requirement: Tenants will have the option of finding flatmates.

By clicking on the **Find Flatmates** tab on the **Dashboard**, tenants can access filters to search for potential flatmates based on their preferences. Users can refine their search by selecting filters such as **City**, **Locality**, and **Preferences** (Gender, Smoking/Drinking habits, Pets, and Food preferences). The system then generates a list of tenants ranked from **highest to lowest similarity score** based on the selected criteria.

By clicking on a profile, one can **Review** the User(Tenant) or **Message** them directly. No bugs or issues were encountered during testing, confirming that the functionality works as expected and meets the specified requirements.

Test Owner: Aritra Ambudh Dutta

Test Date: 04/04/2025

Test Results: All the options worked as expected. No bug was reported. Also we assured by testing that a Tenant who has 'No' in "**Are you Seeking a Flatmate?**" option in the profile, won't be visible to the other Tenants looking for flatmates.

Additional Comments: NA

12. Requirement: Tenants will have the option of reviewing Landlords, Tenants and Properties.

As mentioned earlier, a Tenant have the privilege to **Review a Property** or a **Tenant (Potential Flatmate)**. Also, clicking on **Contact Owner** option within a property, we will be redirected to the profile of the **Landlord**. There will be an option to **Review** the Landlord. To review a **Property** or a **User**, you must submit a rating (on a scale of **0 to 5**) along with a text review.

Test Owner: Aritra Ambudh Dutta

Test Date: 04/04/2025

Test Results: All the review options worked as expected. No bug was reported.

Additional Comments: NA

13. Requirement: Tenants will have the option to engage in **two-way messaging** with both **Landlords** and **other Tenants**.

As mentioned earlier, a **Tenant** can access the profiles of both “**potential flatmates**” (other Tenants) and **Landlords** (owners of viewed properties). Clicking on the **Message** button in either case redirects the user to the **Messages** page, where a chat conversation is initiated between the two parties. They can then communicate to discuss the property or potential flatmate arrangements.

Test Owner: Aritra Ambudh Dutta

Test Date: 04/04/2025

Test Results: The chat feature worked as expected. No bug was reported.

Additional Comments: NA

14. Requirement: Tenants will have the option to “**bookmark**” properties of interest and **Tenants** (potential flatmates) of their choice.

As mentioned earlier, a Tenant can browse through available **properties** in the **Find Property** tab and explore potential **flatmates** in the **Find Flatmates** tab. They can bookmark any property or flatmate they are interested in. These bookmarked selections will be reflected in the **Bookmarked Properties** and **Bookmarked Flatmates** sections on the Tenant’s **Dashboard**.

Test Owner: Aritra Ambudh Dutta

Test Date: 04/04/2025

Test Results: The bookmark feature worked as expected. No bug was reported.

Additional Comments: NA

Non-Functional Requirements

1. Requirement: The software is expected to maintain 99% of the period of service, which does not include shut down due to maintenance breaks and software update windows.

Test owner: Aritra Ray

Test Date: 04/04/2025

Test Results: Test passed

Additional Comments: We ran the full software and explored all its features several times with no disruption in the period of service whatsoever.

2. Requirement: The software should be able to respond to the search requests within 5 seconds with the best-matched search results with every search request by the user.

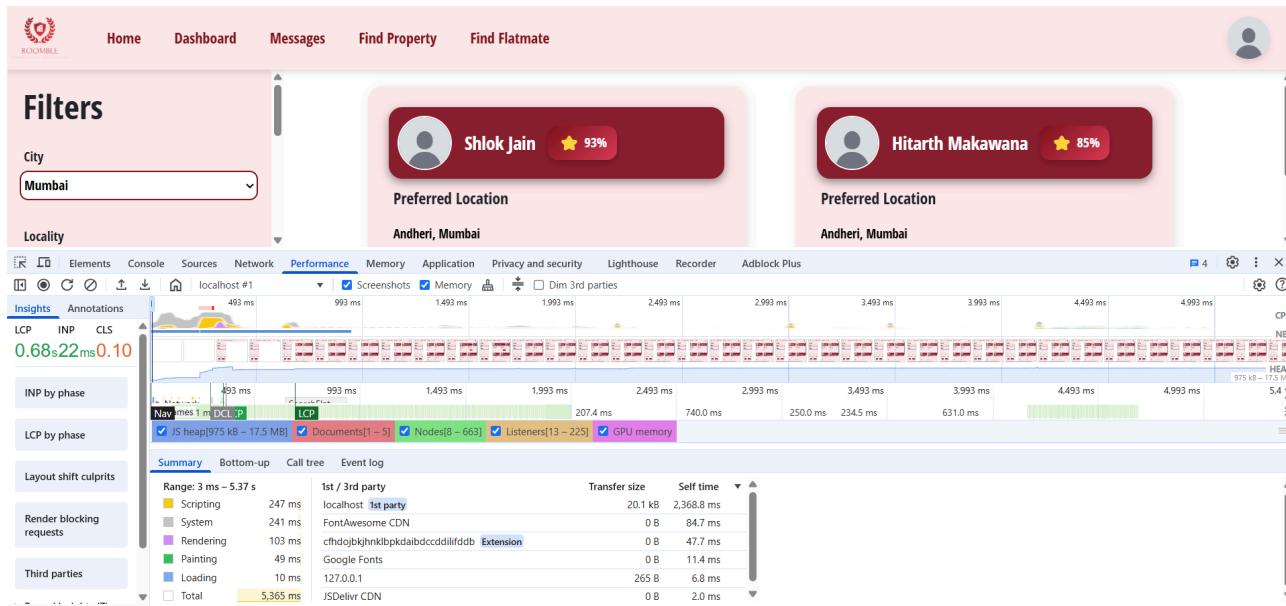
Test owner: Ronav Puri

Test Date: 04/04/2025

Test Results: Test passed

Additional Comments: We used ChromeDevTools to figure out the time latency of search. For searching properties, the mean time (approximate) required was 2.83s, while for searching flatmates, the mean time (approximate) required was 2.68s; both are much faster than 5 seconds. Moreover, always the best-matched results are returned without fail.

The screenshot shows the Roombie software interface. On the left, there is a sidebar with a logo and links to Home, Dashboard, Messages, Find Property, and Find Flatmate. Below these are various filter options: City (Mumbai), Locality (Bandra), Price Range (₹1000 - ₹10000), Area Range (500sqft - 1000sqft), and Number of BHK (1 BHK, 2 BHK, 3 BHK, more). There are also 'Apply' and 'Clear filters' buttons. The main content area displays a property listing for "40000" at "142, Bandra Heights, Mumbai" with "BHK: 4". Below this, there is a section titled "Other Properties You May Be Interested In" showing another property for "10000" near "Near Jamshedpur" with "BHK: 3". On the right side of the screen, the Chrome DevTools Performance tab is open, showing the Network timeline with metrics like LCP (0.64s), INP (22ms), and CLS (0.00). The timeline shows various network requests and their latencies. The bottom of the DevTools window shows a summary of resource types and their load times.



3. Requirement: Validates user credentials such as mobile numbers, emails, passwords, and roles (tenant/landlord).

Test owner: Aritra Ray

Test Date: 04/04/2025

Test Results: Test passed

Additional Comments: User credentials are validated in each step in the backend using JWT authtokens.

4. Requirement: Cross-checks inputs with a securely stored database to ensure only authorized users gain access.

Test owner: Aritra Ray

Test Date: 04/04/2025

Test Results: Test passed

Additional Comments: During login, username and password are securely matched with the data in the database, which is securely stored with a secret MongoURI key. Additionally, passwords are hashed to ensure maximum security.

5. Requirement: Cross-checks inputs with a securely stored database to ensure only authorized users gain access.

Test owner: Ronav Puri

Test Date: 04/04/2025

Test Results: Test passed

Additional Comments: During login, username and password are securely matched with the data in the database, which is securely stored with a secret MongoURI key.

6. Requirement: Encrypts personal information using industry-standard protocols to ensure data is secure during storage and transmission.

Test owner: Aritra Ray

Test Date: 04/04/2025

Test Results: Test passed

Additional Comments: Passwords are hashed using bcrypt and stored securely in database.

7. Requirement: Restricts access to sensitive user data, making it invisible to admins and inaccessible over the server without proper verification.

Test owner: Aritra Ray

Test Date: 04/04/2025

Test Results: Test passed

Additional Comments: Passwords are hashed using bcrypt and stored securely in database, so even admins cannot access it.

8. Requirement: Facilitates landlord-tenant communication via a DM system, keeping contact details private and secure.

Test owner: Aritra Ray

Test Date: 04/04/2025

Test Results: Test passed

Additional Comments: Contact details like phone and email are not shared between the parties, hence privacy of both is preserved. The DM system acts like a secure channel of communication without any privacy concerns.

9. Requirement: Preserves data consistency during system downtime, enabling seamless recovery after maintenance.

Test owner: Aritra Ray

Test Date: 04/04/2025

Test Results: Test passed

Additional Comments: Data is stored in an external MongoDB database, hence system down time does not result in any data loss. Data can be recovered easily after connection is restored with the database.

10. Requirement: Ensures that updating or adding any data does not affect the already stored data.

Test owner: Aritra Ray

Test Date: 04/04/2025

Test Results: Test passed

Additional Comments: MongoDB is used for all database-related requirements, which supports the addition of deletion of data without any changes to the existing data.

11. Requirement: The software should have a user-friendly interface, which allows the users to navigate through desired functionalities without confusion so that the rental process proceeds with ease.

Test owner: Aritra Ray

Test Date: 04/04/2025

Test Results: Test passed

Additional Comments: The interface is fairly intuitive. Also all the features are briefly enumerated in the Home page, so users have no trouble in navigating through the software.

12. Requirement: The architecture, design, implementation, and documentation of the software must be such that they make the system reduce the maintenance overhead as much as possible and adding a particular minor feature should not take as more than 1 person week while fixing a security bug should be done within 2 person days along with updating the documentation.

Test owner: Aritra Ray

Test Date: 04/04/2025

Test Results: Test passed

Additional Comments: The software has been designed in an organized manner so that new features can be added and modifications can be done very easily within two working days. Various components of the software are well organised into folders and **comments** are added wherever required streamline future improvements.

13. Requirement: Since the user requests might vary seasonally hence the database availability might be enough to cater all the requests, i.e. ensure that the software can handle the load as and when the user and property database grows.

Test owner: Aritra Ray

Test Date: 04/04/2025

Test Results: Test passed

Additional Comments: The database has a maximum storage capacity of 512 MB while less than 1 MB is currently occupied. Hence the software can handle 500 times more load, which is reasonable.

14. Requirement: The MTTF (mean time to failure) shall be more than one week.

Test owner: Aritra Ray

Test Date: 04/04/2025

Test Results: Test passed

Additional Comments: The software was tested multiple times throughout the testing week period without a single failure.

15. Requirement: The software must be consistent in providing correct or best matched results according to the prompt filters and search requests.

Test owner: Aritra Ray

Test Date: 04/04/2025

Test Results: Test passed

Additional Comments: The same filters and search criteria for finding flatmates and properties were tried out multiple (10-12) times and consistent results were obtained each time.

16. Requirement: The software is portable and can run on most of the modern web browsers and wider screens.

Test owner: Aritra Ray

Test Date: 04/04/2025

Test Results: Test passed

Additional Comments: Node.js is used for designing the backend part of our software and React.js for designing the frontend part of our software, thus the website can run on most browsers. The software was tested on Chrome, Safari, Firefox, Edge and Opera and it ran successfully.

17. Requirement: The software is portable and can run on most of the modern web browsers and wider screens.

Test owner: Aritra Ray

Test Date: 04/04/2025

Test Results: Test passed

Additional Comments: Node.js is used for designing the backend part of our software and React.js for designing the frontend part of our software, thus the website can run on most browsers. The software was tested on Chrome, Safari, Firefox, Edge and Opera and it ran successfully.

5 Conclusion

- **How Effective and exhaustive was the testing?**

Testing was done almost exhaustively. Each API underwent testing to ensure that they were modifying the database and returning the desired information when needed. We have taken care of all cases, thus ensuring complete branch coverage. The frontend was also tested and bugs were resolved. Validity of each input has been checked in all forms. Appropriate alerts have been shown in such cases. Edge cases were also tested and were confirmed to work as expected. For example, when the entered start date is ahead of the entered end date while viewing reports, an alert is displayed preventing the user from dealing with such invalid inputs.

Also, the testing was quite effective. A developer tested a component that was not developed by him/her, thus eliminating the possibility of bias. The number of bugs we found and corrected was more than 50, indicating that the testing was effective.

- **Which components have not been tested adequately?**

Our test coverage for non functional requirements outlined in the SRS document could be further expanded. Adequate performance tests for non-functional requirements like reliability, maintainability and ease of learning could not be performed.

- **What difficulties have you faced during testing?**

The main difficulty faced during testing was ensuring non-functional requirements are satisfied, in system testing.

- **How could the testing process be improved?**

The testing could have been improved by doing the unit testing or integration testing as automated rather than manual. The developer might subconsciously assume a few crucial things which may be not so obvious to

an end-user or someone who was not involved in the development of the software.

Appendix A - Group Log

Date	Timings	Duration	Discussion & Work done
29 Mar	13:00 - 14:00	1 hour	Meet to discuss the work distribution of the user manual and the testing document among sub-groups. Also initiated the code improvement process to align it with the testing process.
31 Mar	14:00 - 16:00	2 hours	Discussed the progress of the work along with the problems that were faced by the sub-groups.
3 Apr	20:00 - 23:00	3 hours	Finalised the User Manual and completed a large portion of the testing intended to be done
4 Apr	18:00 - 20:00	2 hours	Finalized the Testing Document after reviewing it and the User Manual thoroughly.