Test Document

for

ConnVerse

Version 1.0

Prepared by

Group 03: **Group Name: MahaDevS**

Aaditi Anil Agrawal	**220006**	**aaditiaa22@iitk.ac.in**
Arshit	**220209**	**arshitsk22@iitk.ac.in**
Chayan Kumawat	**220309**	**chayank22@iitk.ac.in**
Dobariya Jenil Bharatbhai	**220385**	**dobariyajb22@iitk.ac.in**
Harsh Agrawal	**220425**	**harshag22@iitk.ac.in**
Harshit	**220436**	**harshit22@iitk.ac.in**
Harshit Srivastava	**220444**	**harshitsr22@iitk.ac.in**
Naman Kumar Jaiswal	**220687**	**namankj22@iitk.ac.in**
Prem Kansagra	**220816**	**premk22@iitk.ac.in**
Priyanshu Singh	**220830**	**spriyanshu22@iitk.ac.in**

Course: **CS253**

Mentor TA: ***Abhilash***

Date: **29th March 2024**

Index

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
Version 1.0	MahaDevS	Initialized the document and added the necessary testing information about our project.	29/03/2024

1 Introduction

- **Our Test Strategy :**
  In our Project we have done both manual and automated testing to check the robustness and payload of our software. We have used Postman software to send HTTP /POST and /GET requests to the backend server for manual testing. Using Postman software we were able to send manual requests to the backend very similar to the request we send from the frontend, requests to different endpoints/backend routes and tallied the received responses against the expected response.

- **Testing Timeline :**
  We conducted our software testing in three distinct phases. Initially, we rigorously tested each component individually prior to their integration into the complete codebase. This phase ensured that each component functioned correctly in isolation.

  In the second phase, which ran concurrently with the implementation process, developers tested their own branches thoroughly before merging them into the main codebase. Manual integration and system testing were carried out to verify the seamless integration of components and to identify any potential issues that arose during this phase.

  The third phase involved the development of a Python script. This script was designed to test the platform comprehensively, employing various testing methodologies. The output generated by the script was meticulously examined through console printing to ensure its accuracy and reliability.

  Throughout the second phase, particular attention was given to addressing challenges encountered during the integration period. Significant features or security enhancements were subjected to thorough testing through the opening of pull requests. Prior to merging these pull requests, extensive testing and conflict resolution were undertaken to maintain the integrity of the codebase.

☐ **Testers and the Testing Approach :**

We have divided the team in two sub groups with the responsibility of testing frontend/Integration testing and the backend testing. There were also two members who were making sure that the whole application was being tested. We adopted a unique and good approach that the testing of the backend of the web app was done by the students who worked on the front end, and vice versa.

☐ **Coverage Criteria Utilized :**

In manual testing, we consider the number of lines of code tested by reviewing intermediate values through printing and assessing whether the required functionalities are fulfilled by our software. Additionally, we incorporate Branch Coverage criteria during testing to ensure a comprehensive evaluation. Recognizing the impracticality of testing all potential errors, we set a target coverage of 95% to strike a balance between thoroughness and feasibility.

☐ **Testing Tool Utilization :**

We employed a range of testing tools throughout our testing process, including Postman for backend testing, Selenium, and pytest, encompassing unit, system, and integration testing phases.

- Postman for backend testing
- Selenium
- pytest, unittest, requests

2 Unit Testing

1. Category: Authentication

a. POST /login

The test aimed to confirm the correctness of the user login function (backend). We have tested for a wide range of login inputs (correct and incorrect).

Unit Details: *Method: POST, Module: authn, Function: Login()*
Description: Backend Function for user login
API Call: "/login"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [23/03/2024]-[23/03/2024]*
Test Results:
Correct Username and Password – "Login Successful"
Correct Username and wrong Password – "Please Try Again! Don't Forget To Verify Your Details"
Incorrect Username – "Please Try Again! Don't Forget To Verify Your Details"
Structural Coverage: *Branch Coverage: 100%, Code Coverage: 100%*
Additional Comments: *Invalid Username formats and null inputs are handled at the frontend itself without sending an API call.*

b. POST /signup

The test aimed to confirm the correctness of the new user signup function (backend). We have tested for a wide range of signing-up inputs (correct and invalid).

Unit Details: *Method: POST, Module: authn, Function: SignUp()*
Description: Backend Function for new user registration
API Call: "/signup"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [23/03/2024]-[24/03/2024]*
Test Results:
Invalid IITK Email - "Please enter valid IITK Email"
Valid IITK Email but incorrect OTP – "Invalid OTP or User Already Exists"
Valid IITK Email Correct OTP but user already exists – "Invalid OTP or User Already Exists"
Valid IITK Email Correct OTP but password and confirm password do not match – "Passwords Do Not Match"
Valid New Entries – "New User is registered"
Structural Coverage: *Branch Coverage: 100%, Code Coverage: 100%*
Additional Comments: *Invalid email formats and null inputs are handled at the frontend itself without sending an API call.*

c. POST /send-otp

The test aimed to confirm whether the validated OTP task is correct for the SignUp function. This is tested for a variety of email addresses as input.

Unit Details: *Method: POST, Module: authn, Function: SendOTP()*
Description: Backend Function for sending OTP
API Call: "/send-otp"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [23/03/2024]-[24/03/2024]*
Test Results:

Valid Email Address: Valid email address provided in the request body.
Result: OTP successfully sent to the specified email address.

Missing Email Address: Empty or missing email address in the request body.
Result: Error response indicating missing email address.

Invalid Email Address Format: Invalid email address format provided in the request body.
Result: Error response indicating invalid email address format.

Email Sending Failure: Valid email address, but failure occurs during the email sending process.
Result: Error response indicating failure to send OTP email.

Structural Coverage: *Branch Coverage: 100%, Code Coverage: 100%*
Additional Comments: *Invalid email formats and null inputs are handled at the frontend itself without sending an API call.*

d. POST /logout

The test aims to validate the functionality of logging out a user by invalidating their session token.

Unit Details: *Method: POST, Module: authn, Function: LogOut()*
Description: Backend function for logging out a user and invalidating their session token
API Call: "/logout"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [23/03/2024]-[24/03/2024]*
Test Results:

Successful Logout: User requests to logout with a valid session token.
Result: Session token is invalidated, and the user receives a success message.

Structural Coverage: *Branch Coverage: 100%, Code Coverage: 100%*

e. POST /forgot-password

The test aims to validate the functionality of the forgot password feature by sending a password reset request to the server.

Unit Details: *Method: POST, Module: authn, Function: ForgotPassword()*
Description: Backend function for handling password reset requests.
API Call: "/forgot-password"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [23/03/2024]-[24/03/2024]*
Test Results:

Valid Password Reset Request: JSON object containing email and old password for password reset.

Result: Password reset request is successfully processed, and a new token is generated for the user's session.

Incorrect Password: JSON object containing an incorrect old password.
Result: Server returns an unauthorized error message indicating an invalid password.

Nonexistent User: JSON object containing the email of a nonexistent user.
Result: Server returns a not found error message indicating that the user does not exist.

Structural Coverage: *Branch Coverage: 100%, Code Coverage: 100%*

2. Category: Blog

a. POST /blog/compose/new

The test aims to validate the functionality of creating a new blog post by any validated user. The system must ensure proper validation of input data and successful insertion of the post into the database.

Unit Details: *Method: POST, Module: blog, Function: CreateBlogPost()*
Description: Backend Function for composing new blog
API Call: "/blog/compose/new"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [24/03/2024]-[24/03/2024]*
Test Results:

Valid Blog Post: Valid blog post content including title, body, Images and tags.
Result: Blog post successfully created.

Valid Blog Post without Image: Valid blog post content including title and body with optional tags and Images.
Result: Blog post successfully created.

Missing Title: Blog post content without a title.
Result: Error response indicating missing title.

Missing Body: Blog post content without a body.
Result: Error response indicating missing body.

Empty Request: Empty POST request.
Result: Error response indicating empty request.

Structural Coverage: *Branch Coverage: 100%, Code Coverage: 100%*
Additional Comments: *A non-validated user cannot compose blog posts.*

b. POST /blog/compose/delete/:id

The test aims to validate the functionality of deleting an existing blog post by any validated user. The system must ensure proper validation of the blog post ID and successful removal of the post from the database.

Unit Details: *Method: POST, Module: blog, Function: DeleteBlogPost()*
Description: Backend Function for deleting a blog post.
API Call: "/blog/compose/delete/:id"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [24/03/2024]-[25/03/2024]*
Test Results:

Valid Blog Post ID: Deleting valid blog post ID of an existing post.
Result: Blog post successfully deleted with message - "Blog post deleted successfully"

Invalid Blog Post ID: Invalid blog post ID that does not exist in the database.
Result: Error response indicating invalid blog post ID - "Invalid postID"

Structural Coverage: *Branch Coverage: 100%, Code Coverage: 100%*
Additional Comments: *Only validated users are authorized to delete blog posts. Also, only existing blogs have the delete option available to users.*

c. GET /blog/:id

The test aims to validate the functionality of retrieving a blog post by its ID.

Unit Details: *Method: GET, Module: blog, Function: RetrieveBlogPost()*
Description: Backend Function for retrieving a blog.
API Call: "/blog/:id"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [24/03/2024]-[25/03/2024]*
Test Results:

Valid Blog Post ID: Valid blog post ID of an existing post.
Result: Blog post successfully retrieved.

Invalid Blog Post ID: Invalid blog post ID that does not exist in the database.
Result: Error response indicating blog post not found with message - "Blog post not found"

Empty Request: Empty GET request.
Result: Error response indicating empty request.

Structural Coverage: *Branch Coverage: 100%, Code Coverage: 100%*
Additional Comments: *The proper handling of this function is done to ensure better user satisfaction.*

d. POST /blog/retrieve

The test aims to validate the functionality of retrieving blog posts based on the author ID through the /blog/retrieve endpoint.

Unit Details: *Method: POST, Module: blog, Function: RetrieveBlogPosts()*
Description: Backend Function for retrieving Blogs
API Call: "/blog/retrieve"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [24/03/2024]-[25/03/2024]*
Test Results:

Valid Author ID: Valid author ID.
Result: Blog posts authored by the specified author successfully retrieved.

No Blogs with Author ID: Request body with author ID which has no blogs written.
Result: Empty array

Invalid Author ID: Invalid author ID that does not exist.
Result: Error response indicating Invalid Author ID - "Failed to retrieve blog posts".

Empty Request: Empty POST request.
Result: Error response indicating empty request.

Structural Coverage: *Branch Coverage: 100%, Code Coverage: 100%*
Additional Comments: *This function is not called without a valid AuthorID.*

e. POST /blog/comment/:id

The test aims to validate the functionality of adding a comment to a specific blog post identified by its ID.

Unit Details: *Method: POST, Module: blog, Function: CreateComment()*
Description: Backend function for adding a comment to a blog post

API Call: "/blog/comment/:id"
***Test Owner**: Naman Kumar Jaiswal*
***Test Date**: [25/03/2024]-[25/03/2024]*
Test Results:

Valid Blog Post ID and Comment: Valid blog post ID and comment content.
Result: Comment successfully added to the specified blog post.

Missing Comment Content: Request body missing comment content.
Result: Error response indicating missing comment content.

Invalid Blog Post ID: Invalid blog post ID that does not exist in the database.
Result: Error response indicating blog post not found - "Invalid blog post ID"

Empty Request: Empty POST request.
Result: Error response indicating empty request.

***Structural Coverage:** Branch Coverage: 100% Code Coverage: 100%*
***Additional Comments:** The option of adding comments is only available to the user beside blogs of valid blog ID.*

f. POST /blog/react/:id

The test aims to validate the functionality of reacting to a specific blog post identified by its ID.

***Unit Details:** Method: POST, Module: blog, Function: LikeUnlikeBlogPost()*
Description: Backend function for liking or unliking a blog post
API Call: "/blog/react/:id"
***Test Owner**: [Naman Kumar Jaiswal]*
***Test Date**: [25/03/2024]-[25/03/2024]*
Test Results:

Like Blog Post: Valid blog post ID, user ID, and action set to 1 (like).
Result: Blog post successfully liked by the user. Like Count Increased by 1.

Unlike Blog Post: Valid blog post ID, user ID, and action set to 1 (unlike).
Result: Blog post successfully unliked by the user. Like Count Decreased by 1.

Dislike Blog Post: Valid blog post ID, user ID, and action set to 2 (dislike).
Result: Blog post successfully disliked by the user.

Undislike Blog Post: Valid blog post ID, user ID, and action set to 2 (undislike).
Result: Blog post successfully undisliked by the user.

Missing User ID: Request body missing user ID.

Result: Error response indicating missing user ID -

Missing Action Type: Request body missing action type.
Result: Error response indicating missing action type - "Method not allowed"

Invalid Blog Post ID: Invalid blog post ID that does not exist in the database.
Result: Error response indicating blog post not found.

Empty Request: Empty POST request.
Result: Error response indicating empty request.

Structural Coverage: *Branch Coverage: 100% Code Coverage: 100%*
Additional Comments: *Liking and Disliking option is only visible to validated users and on valid blog posts.*

g. POST /blog/search

The test aims to validate the functionality of searching for blog posts.

Unit Details: *Method: POST, Module: blog, Function: SearchBlogPosts()*
Description: Backend function for searching blog posts
API Call: "/blog/search"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [25/03/2024]-[26/03/2024]*
Test Results:
Valid Search Query: Valid search query containing keywords or tags.
Result: Blog posts matching the search query successfully retrieved.

Empty Search Query: Empty search query.
Result: No Response - regular feed.

No Results Found: Search query that does not match any blog posts.
Result: Empty result array.
Structural Coverage: *Branch Coverage: 100% Code Coverage: 100%*
Additional Comments: *None*

3. Category: Feed

a. GET /feed/load/:chunk/:id

The test aims to validate the functionality of loading feed items starting for a specified user ID.

Unit Details: *Method: GET, Module: Feed, Function: LoadFeed()*
Description: Backend function for loading feed
API Call: "/feed/load/:chunk/:id"

***Test Owner**: Naman Kumar Jaiswal*
***Test Date**: [26/03/2024]-[26/03/2024]*
Test Results:

Valid User ID: Valid user ID provided in the request parameters.
Result: Feed items successfully loaded starting for the specified user ID.

Missing User ID or chunk parameter: Request parameters missing User ID or chunk parameter.
Result: Error response indicating missing User ID or chunk parameter - "Could not resolve parameters".

Invalid Chunk or ID: Invalid chunk number or starting ID format.
Result: Error response indicating invalid parameters - “Error retrieving blog post”.

No Results Found: No Blog Posts suited for feed for the User ID.
Result: Empty array indicating no feed items found.

Empty Request: Empty GET request.
Result: Error response indicating empty request.

***Structural Coverage:** Branch Coverage: 100% Code Coverage: 100%*
***Additional Comments:** Only a valid User ID has the option of watching his feed.*
.

b. GET /feed/reload/:id

The test aims to validate the functionality of reloading the feed for a specific user.

***Unit Details:** Method: GET, Module: Feed, Function: ReloadFeed()*
Description: Backend function for reloading the feed for a specific user
API Call: “/feed/reload/:id”
***Test Owner**: Naman Kumar Jaiswal*
***Test Date**: [26/03/2024]-[27/03/2024]*
Test Results:

Valid User ID: Valid User ID provided in the request parameters.
Result: Feed successfully reloaded for the specified user.

Missing User ID: Request parameter missing user ID.
Result: Error response indicating missing parameter.

Invalid User ID: Invalid user ID format.
Result: Error response indicating invalid parameter.

No Blog Posts Found: No blog posts available in the database.
Result: Success response with Empty Array indicating no blog posts found.

Structural Coverage: *Branch Coverage: 100% Code Coverage: 100%*
Additional Comments: *Only Valid User IDs can reload feed.*

c. POST /feed/add/tags

The test aims to validate the functionality of adding tags to a user's feed for improving feed.

Unit Details: *Method: POST, Module: Feed, Function: AddTags()*
Description: Backend function for adding tags to a user's feed
API Call: "/feed/add/tags"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [26/03/2024]-[27/03/2024]*
Test Results:
Valid Tag Addition: Valid user ID and tags provided in the request body.
Result: Tags successfully added to the user's feed.

Empty Tags: Empty request body.
Result: No change done to the tags in the feed.

Missing Tags: Missing request body.
Result: Error response indicating missing body.

Structural Coverage: *Branch Coverage: 100% Code Coverage: 100%*
Additional Comments: *Only Valid User IDs can add tags.*

4. Category: Profile

a. POST /profile/create

The test aims to validate the functionality of creating a user profile

Unit Details: *Method: POST, Module: Profile, Function: CreateUserProfile()*
Description: Backend function for creating a user profile
API Call: "/profile/create"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [27/03/2024]-[27/03/2024]*
Test Results:

Valid User Profile Creation: Valid user profile details provided in the request body.
Result: User profile successfully created.

Missing Request Body: Empty or missing data in request body.
Result: User profile still created with empty blocks for missing data.

Structural Coverage: *Branch Coverage: 85% Code Coverage: 100%*
Additional Comments: *The necessary details are fetched and not input by user and hence the secondary data can be updated later.*

b. POST /profile/update

The test aims to validate the functionality of updating a user profile.

Unit Details: *Method: POST, Module: Profile, Function: UpdateUserProfile()*
Description: Backend function for updating a user profile
API Call: "/profile/update"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [27/03/2024]-[28/03/2024]*
Test Results:

Valid Profile Update: Valid user profile details provided in the request body.
Result: User profile successfully updated.

Missing Request Body: Empty or missing request body.
Result: No Update done

NickName not Unique: Updated nickname is not unique.
Result: Update Failed with error message - "Nickname is not unique"

Structural Coverage: *Branch Coverage: 100% Code Coverage: 100%*
Additional Comments: *None*

c. POST /profile/search

The test aims to validate the functionality of searching user profiles of other users.

Unit Details: *Method: POST, Module: Profile, Function: SearchUserProfile()*
Description: Backend function for searching user profiles
API Call: "/profile/search"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [27/03/2024]-[28/03/2024]*
Test Results:

Valid Profile Search: Valid search query provided in the request body.
Result: User profiles matching the search query returned successfully.

Missing Request Body: Empty or missing request body.
Result: No user returned - empty user array.

No Matching Profiles: Search query does not match any user profiles.
Result: Empty response indicating no matching profiles found.

Structural Coverage: *Branch Coverage: 100% Code Coverage: 100%*
Additional Comments: *None*

d. GET /profile/:id

The test aims to validate the functionality of retrieving a user profile by user ID.

Unit Details: *Method: GET, Module: Profile, Function: RetrieveUserProfile()*
Description: Backend function for retrieving a user profile by ID
API Call: "/profile/:id"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [28/03/2024]-[28/03/2024]*
Test Results:

Valid Profile Retrieval: Valid user profile ID provided.
Result: User profile corresponding to the provided ID retrieved successfully.

Invalid Profile ID: Invalid or non-existent user profile ID provided in the parameter.
Result: Error response indicating invalid or non-existent profile ID - "User profile not found"

Structural Coverage: *Branch Coverage: 100% Code Coverage: 100%*
Additional Comments: *Missing Profile is dealt the same as invalid profile.*

5. Category: Chat

a. POST /group

The test aims to validate the functionality of creating a group chat when the endpoint "/group" is accessed with a POST request.

Unit Details: *Method: POST, Module: Chat, Function: createGroupChat()*
Description: Endpoint for creating a group chat
API Call: "/group"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [28/03/2024]-[29/03/2024]*
Test Results:

Successful Group Chat Creation: Valid request with users array and chat name provided
Result: Server successfully creates the group chat and returns a 200 status code with the created group chat details.

Missing Users Array: Request without a users array
Result: Server returns a 400 Bad Request error indicating missing users array.

Insufficient Users: Request with less than 2 users in the users array
Result: Server returns a 400 Bad Request error indicating the requirement for at least 2 users to form a group chat.

Missing Chat Name: Request without a chat name
Result: Server returns a 400 Bad Request error indicating missing chat name.

Structural Coverage: *Branch Coverage: 100% Code Coverage: 100%*

b. PUT /rename

The test aims to validate the functionality of renaming a group chat when the endpoint "/rename" is accessed with a PUT request.

Unit Details: *Method: PUT, Module: Chat, Function: renameGroup()*
Description: Endpoint for renaming a group chat
API Call: "/rename"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [28/03/2024]-[29/03/2024]*
Test Results:

Successful Renaming: Valid request with chat ID and new chat name provided
Result: Server successfully renames the group chat and returns a 200 status code with the updated group chat details.

Missing Chat ID: Request without a chat ID
Result: Server returns a 400 Bad Request error indicating missing chat ID.

Missing Chat Name: Request without a new chat name
Result: Server returns a 400 Bad Request error indicating missing chat name.

Chat Not Found: Request with a valid chat ID but does not exist in the database
Result: Server returns a 404 Not Found error indicating the chat is not found.

Structural Coverage: *Branch Coverage: 100% Code Coverage: 100%*
Additional Comments: *None*

c. PUT /groupremove

The test aims to validate the functionality of removing a user from a group chat when the endpoint "/groupremove" is accessed with a PUT request.

Unit Details: *Method: PUT, Module: Chat, Function: removeFromGroup()*
Description: Endpoint for removing a user from a group chat
API Call: "/groupremove"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [28/03/2024]-[29/03/2024]*
Test Results:

Valid User Removal: Valid request with chat ID and user ID provided

Result: Server successfully removes the user from the group chat and returns a 200 status code with the updated group chat details.

Chat Not Found: Request with a valid chat ID but does not exist in the database
Result: Server returns a 404 Not Found error indicating the chat is not found.

Structural Coverage: *Branch Coverage: 100% Code Coverage: 100%*
Additional Comments: *Here chat means group*

d. POST /groupadd

The test aims to validate the functionality of adding a user to a group chat when the endpoint "/groupadd" is accessed with a PUT request.

Unit Details: *Method: POST, Module: Chat, Function: addToGroup()*
Description: Endpoint for removing a user from a group chat
API Call: "/groupadd"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [28/03/2024]-[29/03/2024]*
Test Results:

Valid Addition to Group Chat: Valid request with authentication token and existing chat ID and user ID
Result: Server successfully adds the user to the group chat and returns the updated chat details.

Chat Not Found: Request with a valid chat ID but does not exist in the database
Result: Server returns a 404 Not Found error indicating the chat is not found.

Structural Coverage: *Branch Coverage: 100% Code Coverage: 100%*
Additional Comments: *Here chat means group*

e. DELETE /:id

The test aims to validate the functionality of deleting a chat when the endpoint "/:id" is accessed with a DELETE request.

Unit Details: *Method: DELETE, Module: Chat, Function: deleteChat()*
Description: Endpoint for deleting a chat by ID
API Call: " /:id"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [29/03/2024]-[29/03/2024]*
Test Results:

Successful Chat Deletion: Valid request with authentication token and existing chat ID
Result: Server successfully deletes the chat and returns a success message.

Chat Not Found: Valid request with authentication token, but the specified chat ID does not exist in the database.
Result: Server returns a 404 Not Found error indicating that the chat is not found.

Structural Coverage: *Branch Coverage: 100% Code Coverage: 100%*
Additional Comments: *Missing Authentication Token is returned with status code 401 by the middleware protect function.*

f. GET /:chatId

The test aims to validate the functionality of retrieving all messages for a specific chat identified by its ID.

Unit Details: *Method: GET, Module: Chat, Function: allMessages()*
Description: Endpoint for removing a user from a group chat
API Call: "/:chatId"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [29/03/2024]-[29/03/2024]*
Test Results:

Valid Retrieval of Messages: Valid request with authentication token and existing chat ID.
Result: Server successfully retrieves all messages for the specified chat and returns the messages.

Non-existent Chat ID: Request with a non-existent chat ID
Result: Server returns a 404 Not Found error indicating that the chat does not exist.

Structural Coverage: *Branch Coverage: 100% Code Coverage: 100%*
Additional Comments: *We are retrieving an entire chat history.*

g. GET /

The test aims to validate the functionality of retrieving all users or searching for users based on a keyword.

Unit Details: *Method: GET, Module: Chat, Function: allUsers()*
Description: Endpoint for removing a user from a group chat
API Call: "/"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [29/03/2024]-[29/03/2024]*
Test Results:

Successful Retrieval of All Users: Valid request with authentication token
Result: Server successfully retrieves all users (excluding the authenticated user) and returns the user list.

Keyword Search - Matching Name: Valid request with authentication token and search query matching usernames.
Result: Server successfully retrieves users whose names match the search query and returns the filtered user list

Keyword Search - No Match: Valid request with authentication token and search query with no matching users.
Result: Server returns an empty user list since there are no matches for the search query.

Structural Coverage: *Branch Coverage: 100% Code Coverage: 100%*
Additional Comments: *None*

h. POST /

The test aims to validate the functionality of sending a message.

Unit Details: *Method: POST, Module: Chat, Function: sendMessage()*
Description: Endpoint for sending a message
API Call: "/"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [29/03/2024]-[29/03/2024]*
Test Results:

Successful Message Sending: Valid request with authentication token, content, and chat ID
Result: Server successfully creates and sends the message to the specified chat, and returns the message details.

Missing Content: Valid request with authentication token and missing content
Result: Server returns a 400 Bad Request error indicating missing content in the request.

Missing Chat ID: Valid request with authentication token and missing chat ID
Result: Server returns a 400 Bad Request error indicating missing chat ID in the request.

Structural Coverage: *Branch Coverage: 100% Code Coverage: 100%*
Additional Comments: *The option of sending messages is only visible to valid chat ids and requests are not sent if you do not fill any content.*

i. GET /

The test aims to validate the functionality of fetching chats.

Unit Details: *Method: GET, Module: Chat, Function: fetchChats()*
Description: Endpoint for fetching chats.
API Call: "/"
Test Owner*: Naman Kumar Jaiswal*
Test Date*: [29/03/2024]-[29/03/2024]*

Test Results:

Successful Chat Fetching: Valid request with authentication token
Result: Server successfully fetches and returns the user's chats, including group chats, with their latest messages.

Structural Coverage: *Branch Coverage: 100% Code Coverage: 100%*
Additional Comments: *The option of sending messages is only visible to valid chat ids and requests are not sent if you do not fill any content.*

3 Integration Testing

1. ***Assessing the interaction between the login process and the user home page.***

Module Details: Login
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: Upon successful login with correct user credentials, the system seamlessly navigates to the user's personalized homepage, displaying unique details such as the user's name and profile picture.

2. ***Upon logout, users are automatically redirected to the landing page.***

Module Details: Logout
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: All roles passed the test successfully.

3. ***Registration is restricted to users with invalid IITK or pre-registered email id's.***

Module Details: SignUp
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: An error prompt displays, indicating "Please Enter Valid IITK Email" whenever a non-existing IITK Email is entered for registration. If a registered user attempts to register again, a prompt appears stating "Invalid OTP or User Already Exists".

4. ***Incorrect credentials prevent users from accessing the system.***

Module Details: Login
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: Users cannot log in with incorrect credentials (email, password). An alert ("Please Try Again! Don't Forget To Verify Your Details" is displayed indicating an error, maintaining confidentiality regarding the incorrect details instead of mentioning what is wrong exactly for safety purposes.

5. ***The system does not send an OTP for signup to a non-existent IITK Email Id.***

Module Details: SendOTP
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: In the event that a user with a valid IITK email, yet not pre-registered, inputs an invalid OTP during sign-up, a prompt will appear indicating "Invalid OTP or User Already Exists". To uphold user safety, our system refrains from explicitly detailing registration errors, thus impeding external intrusion.

6. ***Users can create a new blog, accessible from the Blog section in the navbar, with changes reflected across the website.***

Module Details: ComposeBlog
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: Successful
Additional Comments: Users are encouraged to adhere to the provided blog template to ensure consistency in website content. However, the blog composition feature remains functional even if users do not provide all details such as an image or tags. The sole requirement is that the image must be in JPEG format. Upon successful blog creation, users receive a prompt confirming the creation and can view their new blog at the top of their created blogs list. If the sole requirement isn't met then a prompt "Please Try Again! Failed to create blog post " appears.

7. ***One can view all the blogs.***

Module Details: ViewBlog
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: Successful
Additional Comments: Upon selecting a specific blog, users are directed to its unique page identified by a distinct blog ID. This page displays all the details added by the blog owner during its creation.

8. ***Only the creator has the authority to delete the blog they have authored.***

Module Details: DeleteBlog
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: Successful
Additional Comments: To delete a blog, navigate to the desired blog's page and locate the dustbin icon. Depending on user permissions, clicking this icon will prompt deletion and display a confirmation message.

9. ***Users are free to leave multiple comments on any blog post.***

Module Details: Comment
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: Successful
Additional Comments: Commenting is straightforward: open a blog, type your comment, and upon clicking submit, it becomes visible to all, linked to the respective blog post.

10. ***Users have the option to react to a blog post by either upvoting or downvoting.***

Module Details: React
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: Successful
Additional Comments: The key aspects tested include users being able to upvote, downvote, or remain neutral about a blog post. Each user's reaction is unique to a particular user-blog pair. Upvote counts are updated in real-time; for instance, if a user switches from upvoting to downvoting, the upvote count decreases by 1 accordingly. Additionally, downvote counts are not displayed to foster a more positive viewing atmosphere by minimizing the focus on negative feedback.

11. ***Users can access their own profile by hovering over their profile image in the navigation bar.***

Module Details: ViewProfile
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: Successful. Upon clicking the "Profile" button after accessing their profile through hovering over the navbar image, users are redirected to their unique profile page, accessible only to them via a link containing their unique IITK roll number.
Additional Comments: Thus the user lands on his/her profile page. Upon landing on this page, users have the ability to update their own profiles.

12. *Users can search for any other member of the website.*

Module Details: UserSearch
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: Successful. The search function operates based on names and various available filters located beside the search bar on the members page.
Additional Comments: Upon clicking the search icon, users can view a list of website members who meet the specified filters, displaying the search results. Profiles of other members can be accessed by clicking on the respective search results.

13. *Members are empowered to update their profile details as required.*

Module Details: UpdateProfile
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: Successful. Upon making any alterations to the userProfile page, such as adding or removing items by clicking the +, tick, or dustbin buttons, users will receive a prompt confirming the successful update. In case of any errors during the update process, users will be notified with a prompt indicating the update was unsuccessful, prompting them to reload the page and try again.
Additional Comments: The updating process is expedited by storing changed details on this page simultaneously. Moreover, one doesn't get the option to update other's profile.

14. *Members have access to blog posts authored by all others on their home page.*

Module Details: FeedLoad
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: Successful. When a user requests blog posts from the database for the home page, the system retrieves the ten most relevant blogs to display.
Additional Comments: Additionally, we've implemented query timeout mechanisms, whereby if a database query exceeds a duration of 15 seconds, the request is terminated. This precaution is taken to mitigate the potential performance degradation caused by prolonged queries, which could impact the experiences of other users.

15. *Reloading feed upon reaching the bottom of the home page.*

Module Details: FeedReload
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: Successful. Upon reaching the bottom of the home page, the system seamlessly loads an additional 10 blogs, ensuring smooth performance.

16. *Users have the capability to search through all blogs using their titles.*

Module Details: BlogSearch
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: Successful. The search functionality operates solely based on blog titles.
Additional Comments: Upon clicking the search icon, users will be presented with a list of blogs that match the search criteria. Clicking on a specific search result allows access to the corresponding blog.

17. *Initiate a direct messaging chat with any registered user by searching him/her.*

Module Details: CreateChat
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: Successful. The search functionality operates based on names.
Additional Comments: Upon pressing the Go button, a list of users matching the search criteria will appear. Selecting a specific search result enables the initiation of a direct messaging chat with that user. The chat is displayed in the MyChats section for both users, with the chat contents/messages visible in the right panel.

18. *Begin a Group Chat with a minimum of 3 members.*

Module Details: CreateGroupChat
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: Successful. Upon clicking the "New Group Chat" button, a popup will appear prompting you to select members for the group and specify the group name, with you designated as the default admin.
Additional Comments: The popup will display an error message if only one or no members are selected to join the group, indicating that a group requires a minimum of three members.

19. *Group Chat Administration Options: Member Management*

Module Details: UpdateGroupChat
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: Successful. Group Chat Management Options: Renaming, Member Addition/Removal (Admin), and Leaving. Displayed upon clicking the visibility icon after accessing the group chat from MyChats.
Additional Comments: Leaving a group chat removes it from MyChats and ejects the user. Adding a member updates MyChats, appends the new member in the group chat. Moreover previous messages are visible to the newly added member. Renaming involves changing the group name.

20. *Delete a Chat.*

Module Details: DeleteChat
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: Successful. Any member can delete a direct message or a group chat by selecting the delete icon on the respective chat in the MyChats list.
Additional Comments: Upon deletion, the corresponding chat is removed from all members' MyChat Lists, ensuring permanent deletion.

21. *Forgot Password.*

Module Details: ForgotPassword
Test Owner: *Prem Kansagra*
Test Date: *28/03/2024 - 28/03/2024*
Test Results: Successful. A user possessing a valid IITK Email address can generate the OTP. However, if the user is not yet registered, the database query will yield no results, resulting in only an additional search query without any modifications to the database. Furthermore, the process closely mirrors the signup procedure.

4 System Testing

1. Requirement: Chat with other users

Users will be able to chat with other users.

Test Owner: [Chayan, Prem, Jenil, Harshit]
Test Date: [23/03/2024]-[25/03/2024]
Test Results: [Success]
During the testing phase conducted from [23/03/2024] to [25/03/2024], we rigorously evaluated the functionality of the chat feature, adhering to the specified requirements through a combination of manual and automated testing.

- Manual Testing: We initiated chat requests, sent messages and verified the recipient is getting the messages in proper latency or not.
- Automated Testing: Automated testing tools such as Selenium were utilized to simulate user interactions and verify the proper functioning of the chat initiation and message received in proper latency.

Result: The chat feature effectively implemented the specified requirements, allowing users to initiate, and engage in conversations with other users while supporting the exchange of text messages. Both manual and automated testing methodologies confirmed the robustness of the feature.

2. Requirement: Searching Blogs & Searching Users

Searching Blogs- Users will be able to search blogs by the name/title of the blog.
Searching Users- Similar to searching blogs, Users will be able to search other user by the name of the user or by applying filter

Test Owner: [Chayan]
Test Date: [23/03/2024]-[25/03/2024]
Test Results: [Success]
Throughout the testing period from [23/03/2024] to [25/03/2024], our evaluation included the utilization of a purpose-built Python script for conducting searches on both blogs and users. This script executed searches based on user input criteria and presented all relevant search data on the console. The script retrieved and displayed blog names, user profiles, and associated details, enhancing the testing process by automating search operations and providing clear and concise visibility into search results.

3. Requirement: Comment, Upvote or Downvote in Blogs

Users will be able to comment on a blog. Users will also be able to upvote/ downvote a blog.

Test Owner: [Chayan]
Test Date: [23/03/2024]-[25/03/2024]
Test Results: [Success]

- Using a Python script, we automated the testing of commenting, upvoting, and downvoting functionalities. The script simulated user interactions by posting comments and voting on blogs, ensuring that these actions were performed as expected.
- The script also verified the integrity of data storage and retrieval, ensuring that comments and votes were stored correctly and displayed accurately on the blog interface.

Result: The manual and Python script testing confirmed the successful implementation of commenting, upvoting, and downvoting functionalities in blogs.

4. Requirement: Composing & Deleting Blogs

Composing Blogs: Users will be able to compose a blog. A blog can contain a Title, Author name, body text, an image and relevant tags according to the content of the blog.
Delete Published Blogs: Only the user who created the blog can delete the blog from the unique individual blog page. Users will not be able to delete any other user's blog.

Test Owner: [Chayan]
Test Date: [23/03/2024]-[26/03/2024]
Test Results: [Success]

This was manually tested. Several deletes and blog publications were made between [23/03/2024] and [26/03/2024]. Although we attempted to create an automated testing script, time constraints limited our ability to do so. Nevertheless, we performed manual tests numerous times for Composing Blogs and Editing Published Blogs, and all tests passed successfully.

5. **Requirement: User Authentication**
User Registration :

All users must register on the system using their email ID provided by the institute and an OTP sent to their email ID. Users must create a password for further log in.

Changing Password for Log In

If a user forgets his/her password for logging in, the user will be able to reset the login password via OTP sent to the registered email id.

Manual Registration for Alumni

If an alumnus wants to register, he/she will be able to send an email regarding it to the App's gmail ID (connverse22@gmail.com). After manual verification, alumni will be able to register him/herself and use it.

Test Owner: [Aditi]
Test Date: [23/03/2024]-[25/03/2024]
Test Results:
The test was successfully conducted from [23/03/2024] to [25/03/2024], encompassing the updated requirements for user authentication.
For user registration, the process now involves users registering using their institute-provided email ID and receiving an OTP for verification. They are then prompted to create a password for subsequent logins. Additionally, the functionality to reset the login password via OTP sent to the registered email ID was tested successfully.In the scenario of alumni registration, the process has been refined to require alumni to send an email to the application maintainer expressing their intent to register. Upon manual verification, alumni are permitted to register themselves and utilize the application.
During testing, various login scenarios were covered:

Successful login with correct username and password.
Unsuccessful login with correct username but wrong password.
Unsuccessful login with incorrect username.

Similarly, for registration, the following cases were addressed:

Attempting to register with an already registered email.
Entering an invalid IITK email address.
Successful registration with a new username after receiving an OTP to the entered IITK email.

The flow of data for alumni registration was manually tested, ensuring that the required information was appropriately stored in the backend for verification by the website manager. All test cases yielded positive results, affirming the effectiveness of the user authentication system.

6. Requirement: Testing Non Functional Requirements - Software Quality Attributes

Usability: The application's interface is designed to be user-friendly, ensuring an intuitive experience for first-time users. Users will find it easy to update their profiles, publish blogs, and connect with others through the chat feature.

Portability: The application should be portable across different devices with consistent functionality and user experience. The software must be built on cross-platform frameworks and the GUI must be responsive to the port width.

Interoperability: This application is designed to function smoothly alongside other applications on the device, avoiding any interruptions. When users are in the midst of composing a blog, they can save it as a draft, use other applications concurrently, and return to continue blogging seamlessly from the Recent screen or browser tabs. This design guarantees the application's seamless interoperability with different device functions.

Test Owner: [Chayan, Jenil, Prem, Priyanshu]
Test Date: [21/03/2024]-[24/03/2024]
Test Results:

During the testing period from [21/03/2024] to [24/03/2024], we evaluated the non-functional requirements pertaining to usability, portability, and interoperability of the application.

- Usability: The application's interface was found to be highly intuitive, allowing users to easily navigate through features such as profile updates, blog publishing, and chat interactions. Feedback from users confirmed a positive experience with the interface's user-friendliness.
- Portability: The application demonstrated consistent functionality across various devices, indicating successful implementation of cross-platform frameworks. The GUI was responsive to different port widths, ensuring a seamless user experience irrespective of the device used.
- Interoperability: The application seamlessly integrated with other device functions, allowing users to multitask while composing blogs. The ability to save drafts and resume blogging from the Recent screen or browser tabs further enhanced interoperability.

Result: The application successfully met the non-functional requirements for usability, portability, and interoperability, providing users with an intuitive interface, consistent functionality across devices, and seamless integration with device functions.

5 Conclusion

How Effective and exhaustive was the testing?

The testing was very effective, as we ensured that before anyone from the team pushed or merged the code into the main branch of the repository, it was locally tested by that person. Our application has two types of users: students who are currently on campus and students who have graduated from our college (alumni). We manually and automatically tested all the data flow for these two roles, and all tests passed. Therefore, our testing was exhaustive. We used Python scripts to run the automated tests and simulate real peer-to-peer interactions and activities. Utilizing Postman for backend testing helped automate a portion of the testing, thus relieving us slightly of the exhaustive process.

Which components have not been tested adequately?

Almost all the components have been tested adequately.

What difficulties have you faced during testing?

The manual testing was time-consuming, but we tried to utilize automated testing process as much as possible, which helped us complete it in less time. Therefore, we did not encounter any major difficulties.

How could the testing process be improved?

Half of the testing process was conducted online. We held numerous Google Meet sessions, which were time-consuming compared to what could have been achieved through offline interactions with our teammates. This prolonged the testing process and made it more tiring for us. Additionally, we only ensured that extremely sophisticated queries made to the backend by Postman would function properly for all potential frontend inputs, rather than for any input. By completing the skipped step, we could have further assured the security to the backend.

Appendix A - Group Log

Sr. No.	Date	Activity	Mode	Members
1	21-03-2024	We organised meetings to initiate the testing procedures and in case of any issue.	Offline	All
2	23-03-2024	Discussed and distributed work related to the issues in the backend testing and given this work to the frontend people(who made the frontend)	Online	All
3	24-03-2024	We discussed and allocated tasks related to frontend issues and integration testing. Subsequently, we assigned the tasks to the backend team, who were responsible for developing the backend.	Online	All
4	26-03-2024	This was a 5-hour-long Google Meet session during which we tested all the backend components using Python scripts and the Postman tool. We then assigned additional work to the frontend team, specifically focusing on writing more scripts for search and chat testing.	Online	All
5	27-03-2024	This was a 6-hour Google Meet session during which we tested the search and chat functionalities using automated tools. Additionally, we evaluated the non-functional requirements of the project.	Online	All

6	28-03-2024	We distributed the work to all team members for writing their respective parts in the testing document. Additionally, some team members were assigned to work on the user manual document.	Online	All
7	29-03-2024	A final 2-hour Google Meet session was held to review our testing and user manual documents.	Online	All