Implementation Document

for

SafeSpace

Version 1.0

Prepared by

Group 10: **Group Name:** 404 Team Not Found

Anirudh Singh	230142	sanirudh23@iitk.ac.in
Archita Goyal	230187	architag23@iitk.ac.in
Brinda Fadadu	230307	brindaf23@iitk.ac.in
Nakul Patel	230676	nakulpatel23@iitk.ac.in
Naman Yadav	230680	namanyadav23@iitk.ac.in
Om Chaudhari	230715	omcc23@iitk.ac.in
Rohit Yadav	230873	rohity23@iitk.ac.in
Sayani Patra	230943	sayanip23@iitk.ac.in
Vivek	231169	vivek23@iitk.ac.in
Yatharth Sharma	231198	yatharths23@iitk.ac.in

Course: CS253

Mentor TA: *Souvik Mukherjee*

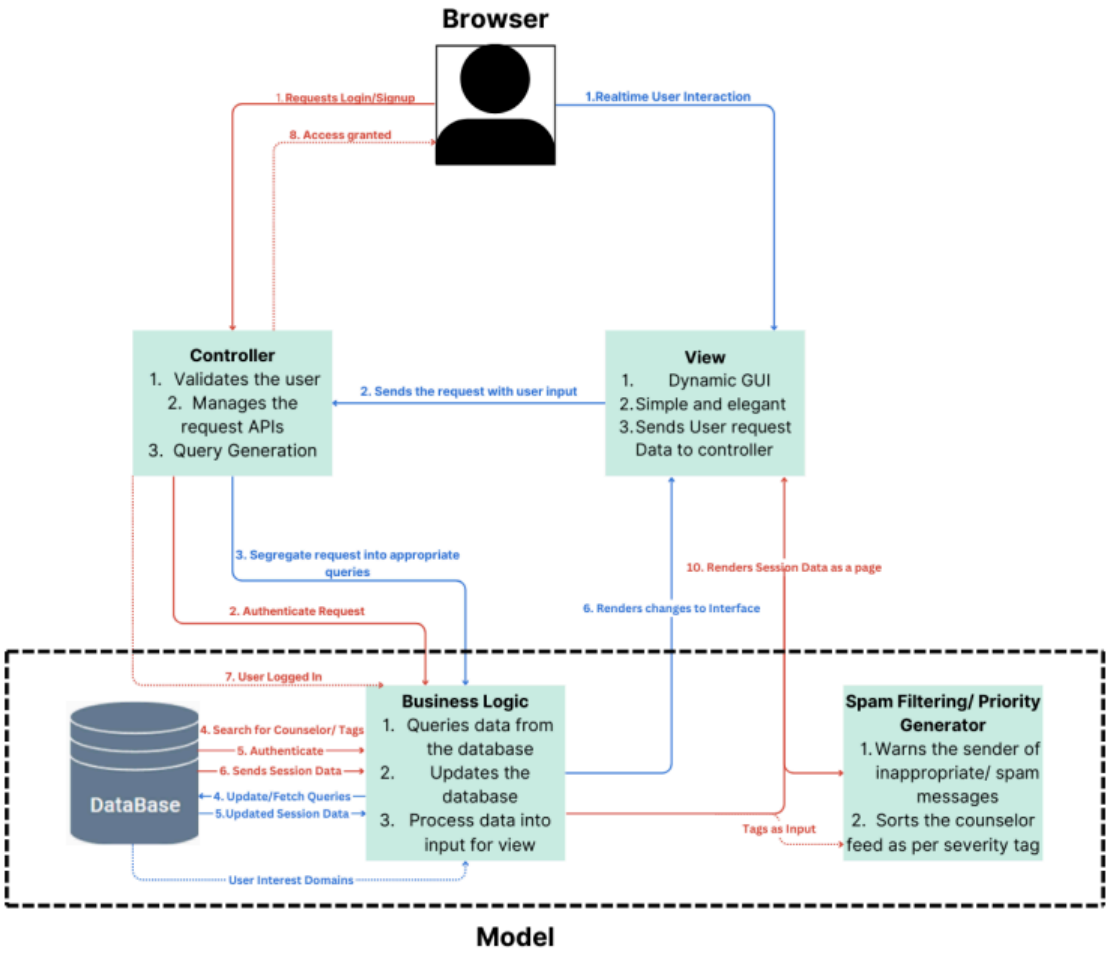
Date: 28/03/2025

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
v1.0	Anirudh Singh Archita Goyal Brinda Fadadu Nakul Patel Naman Yadav Om Chaudhari Rohit Yadav Sayani Patra Vivek Yatharth Sharma	The First Draft of SafeSpace Implementation	28/03/25

1 Implementation Detail

SafeSpace implements a Model View Controller Architecture for integrating the backend and frontend of the webpage. This Architecture is essential in the building of this application as it allows us to make individual components which are streamlined to efficiently perform the specific tasks that they need and provide a clean code space hierarchy for us to navigate and work.



Architectural Hierarchy:

1. View (Frontend): This is the User Interface which the user will interact with and acts as a layer between User (Browser) and Backend. We have implemented a React based web application frontend for this role.

2. Controller (Frontend) – First level of processing of User interaction data

a. User Authentication: This logic authenticates any Login request from the user and generates a user session for a successful login. It also handles the cases of registration differently.
b. User Interaction Handler: This proactively detects any interaction of the User with the frontend and generates the necessary Model requests which will generate the required changes which have to be operated on the View.

3.Model (Backend) – Combination of Business Logic Tier, Database Tier and Priority Generator

a. Query Logic: This is the part of the Business Logic which generates the necessary Queries to the MongoDB Database. This includes any CRUD Command (Create Read/Fetch Update Delete) for any user, post, chat, profile or bookings.
b. Database: This is key to our Software. The Database stores current users' data, their profile data, their posts/comments, booked appointments and even personalized messages. Most API calls that are made to the server involve the database.
c. View Update: This is the front-end handler which updates the exact React components for any interaction the user makes with the View. This is the component-based model of ReactJS.
d. Priority Generator: We make an API call to the Gemini model to analyze user concerns and classify them into severe, moderate, mild, or none based on emotional intensity. The counselor feed is then sorted by severity, ensuring urgent cases get priority.
e. Spam Filtering: We use the Detoxify library, which leverages a machine learning model to detect and classify hate speech, toxicity, and inappropriate messages. If a user sends such a message, they receive an automatic warning, ensuring a safe and respectful environment on the platform.

Code Implementation Sub-Codebases:

1. **Frontend:**
   This directory contains all the front-end components used in developing the user interface of the application. The front end is built using Vite with React, a component-based framework that efficiently updates the UI based on user interactions. ReactJS merges JavaScript and JSX, allowing dynamic content updates without reloading the page. We use CSS for styling and enhancing the visual appearance, ensuring a responsive and interactive user experience.

   Why was ReactJS extensively used for frontend development?
   a. **Performance Optimization:**
      ReactJS features like memoization (optimize performance of functions by caching results of expensive function calls), PureComponent (base class that implements a shallow comparison of props and state to determine whether a component should re-render) and React.memo (higher-order component provided by React that memoizes the result of a functional component rendering) to optimize rendering performance and reduce unnecessary re-renders.

b. **Virtual DOM:**
In a state change of the application (State Diagram) only the necessary components of the Virtual DOM (in-memory representation of the actual DOM) get updated. This efficient reconciliation algorithm minimizes the performance overhead associated with updating the UI.

c. **Component-Based Architecture:**
React breaks down the UI elements into reusable modular components which enhances maintainability and scalability in large complex UIs.

d. **Unidirectional Data Flow:**
React promotes a unidirectional data flow from parent components to child components. This simplifies data management and makes it easier to reason about the state of the application during the development phase of the software.

We have manually created all React UI components to ensure full customization and flexibility. This approach allows us to tailor the user interface to our specific requirements while maintaining a consistent design and functionality. Building components from scratch ensures better control over performance, accessibility, and user experience.

2. Backend (main-backend):
This directory contains all the necessary functions to react to all the API requests that are called off the User interactions with the interface and call necessary Database Query requests and View Updates. This backend is written in Python. FastAPI, a popular web framework in Python, is used to provide a route-based approach to defining HTTP endpoints and handling requests and responses.

Why Python and its FastAPI framework was used over other languages?

a. High Performance & Asynchronous Processing
FastAPI is built on Starlette and Pydantic, making it one of the fastest Python web frameworks. It supports async/await, allowing efficient handling of multiple requests without blocking execution. This makes it ideal for real-time applications, microservices, and high-concurrency workloads.

b. Automatic Validation & API Documentation
FastAPI uses Pydantic for automatic data validation, ensuring type safety and reducing errors. It also generates interactive API documentation (Swagger UI & ReDoc) automatically, allowing easy testing and integration without extra effort.

c. Scalability & Database Integration
FastAPI seamlessly integrates with databases like PostgreSQL, MySQL, and MongoDB using ORMs like SQLAlchemy. It supports dependency injection, making authentication, database sessions, and API management more modular and scalable.

d. Security & Modern Development Features
Built-in OAuth2, JWT authentication, and API key support enhance security. It also supports WebSockets, which was used to develop chats, and cloud-native deployments with Docker and Kubernetes, making it an excellent choice for modern, production-ready applications.

3. Backend (chat-system):
This directory contains the complete implementation of the chat system, built primarily using Node.js with Socket.IO and Express.js. The system follows a Microservices Architecture, offloading chat-related operations from the primary backend to enhance scalability and efficiency. Express.js handles HTTP requests, while Socket.IO enables real-time, bidirectional communication between users. The chat server runs using nodemon, allowing automatic restarts on code changes for a smoother development experience.

Why did we choose Node.js and these libraries for Chat Implementation?

a. **Node.js** is event-driven and optimized for asynchronous, non-blocking I/O operations, making it ideal for handling multiple real-time connections efficiently. With its vast community support and ease of development, it accelerates feature implementation and maintenance.
b. **Socket.IO** provides a seamless way to implement real-time communication, supporting WebSockets and fallback mechanisms for reliability.
c. **Express.js** offers a minimal yet powerful framework to manage HTTP endpoints alongside the WebSocket server, keeping the service lightweight and scalable.
d. Deployment of Node.js microservices is straightforward and integrates smoothly with the existing architecture, ensuring a modular and maintainable system.

This combination of Node.js, Socket.IO, and Express.js ensures a robust, scalable, and efficient chat system with low latency and high concurrency support.

Database Management:

1. **MongoDb:** We are using MongoDb as the primary centralized database for storing all the data regarding users, their profile data, their appointments, their blog entries, their comments as well as their personal messages. MongoDb is a very popular pick for database because:

    a. **Flexible Schema Design:** MongoDB is a document-oriented database that uses a flexible schema design. Unlike traditional relational databases, MongoDB does not require a predefined schema, allowing developers to store and manipulate data in a more flexible and dynamic manner.
    b. **Scalability:** MongoDB is designed to be scaled horizontally enabling it to handle large volumes of data and high traffic loads. If the software is tested successfully, it can soon be brought to handle real-time load.
    c. **Cross-Platform Compatibility:** MongoDB is platform-independent and runs on various operating systems also providing official drivers and client libraries for various popular programming languages such as Python and JavaScript etc. This Enables developers to integrate MongoDB into their applications regardless of the technology stack.

    2. **Mongoose:** A popular Object Data Modeling (ODM) library for MongoDB in NodeJS which provides a higher-level abstraction over the MongoDB NodeJS driver, allowing developers to interact with MongoDB databases using a more expressive and intuitive syntax. Mongoose supports built-in custom data validation rules that can be applied to schema fields. Mongoose also gives the freedom of Schema modelling which gives the back end engineer a lot of freedom and flexibility to work with.

    3. PyMongo: PyMongo is the official Python driver for MongoDB, enabling efficient CRUD operations, indexing, and aggregation. It supports connection pooling, authentication, and automatic failover for reliability. With BSON handling and schema-less management, PyMongo simplifies working with dynamic and scalable databases.

Development and version control environment:

1. We have used **Git** as our version control system. It is the most used version control system that is used for software development and other version control tasks. It tracks the changes you make to files, so you have a record of what has been done and allows us to revert to specific versions whenever we need to. It also makes collaboration easier, allowing changes by multiple people to all to be merged into one source.
2. **GitHub**, a web-based Git repository hosting service was also used to manage our repositories and collaborate amongst ourselves. This online service currently stores

our Software code and has made deployment and testing way easier subconsciously

Multi Server Backend:
In this system, a single view connects to two backend servers—one running FastAPI (Python) for the main backend and another using Node.js with Socket.IO for the chat microservice. Both servers share the same database, and a controller decides which server to handle each API request.

Why is such a system better?

1. Language Specialization:
FastAPI (Python) is excellent for API performance, data validation, and machine learning integration. It provides automatic documentation, built-in async support, and is faster than traditional Python web frameworks.
Node.js with Socket.IO is better suited for real-time WebSocket-based chat systems, thanks to its event-driven architecture and non-blocking I/O model.
Using both languages optimizes performance by assigning the best tool for each job.

2. Micro Performance Optimization:
FastAPI handles API requests efficiently, leveraging Python's rich ecosystem for business logic, analytics, and machine learning.
Node.js with Socket.IO ensures low-latency, real-time messaging by efficiently managing multiple simultaneous chat connections.
Each backend operates independently, ensuring neither is overloaded with tasks outside its specialization.

3. Microservices Architecture:
The chat system is implemented as a microservice using Node.js and Socket.IO, while the main backend remains in FastAPI.
This approach ensures independent scaling of chat and main backend services, easier updates—modifying one service does not affect the other. Also, better fault isolation, reducing the risk of system-wide failures.

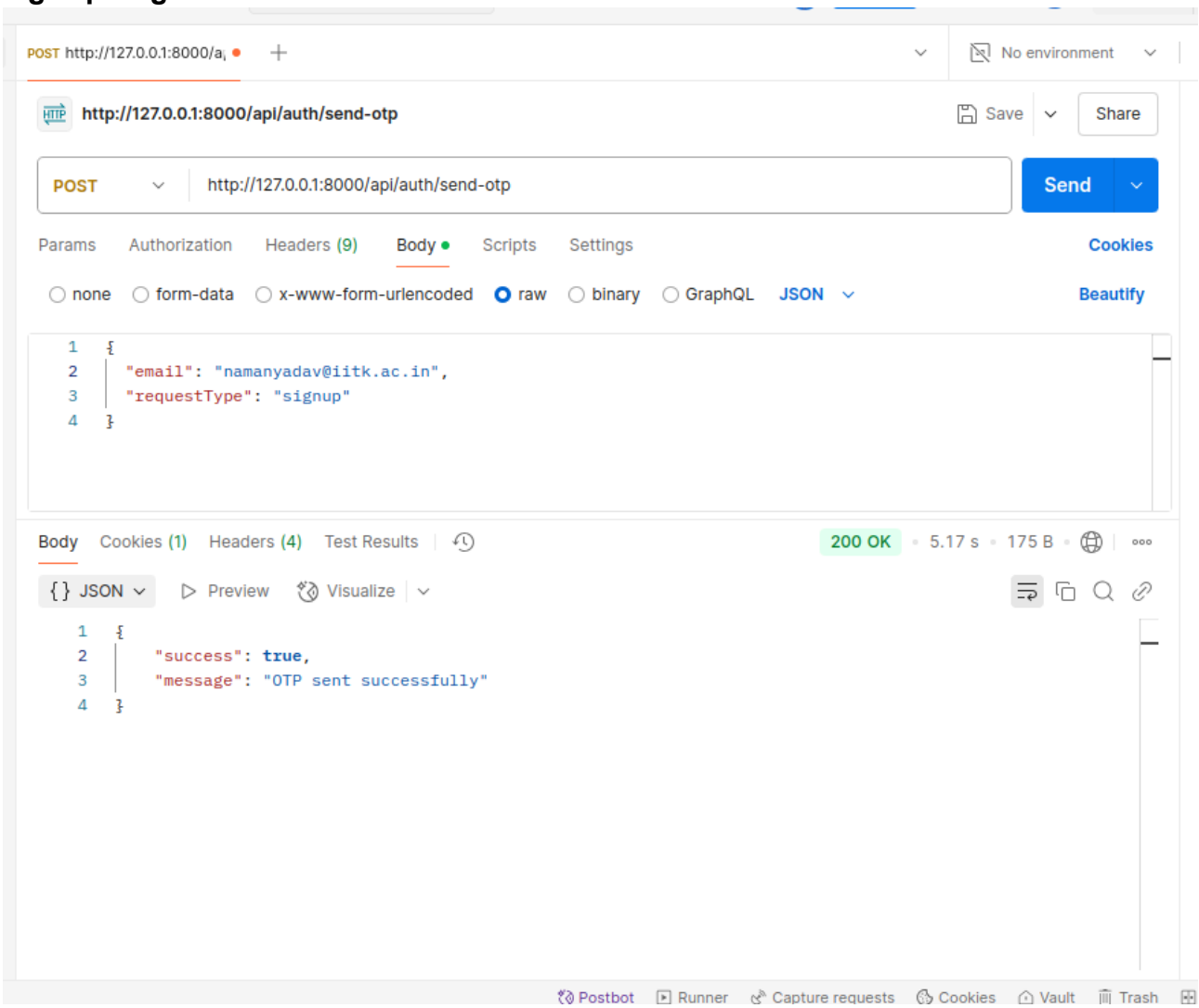
4. Multiple Servers:
The frontend sends requests to either FastAPI or the chat service, distributing the load efficiently. Both backends access the same database, ensuring consistent data across the system.
This ensures lower server load, preventing any single service from becoming a bottleneck and improved fault tolerance, as one server can continue operating even if the other experiences downtime.

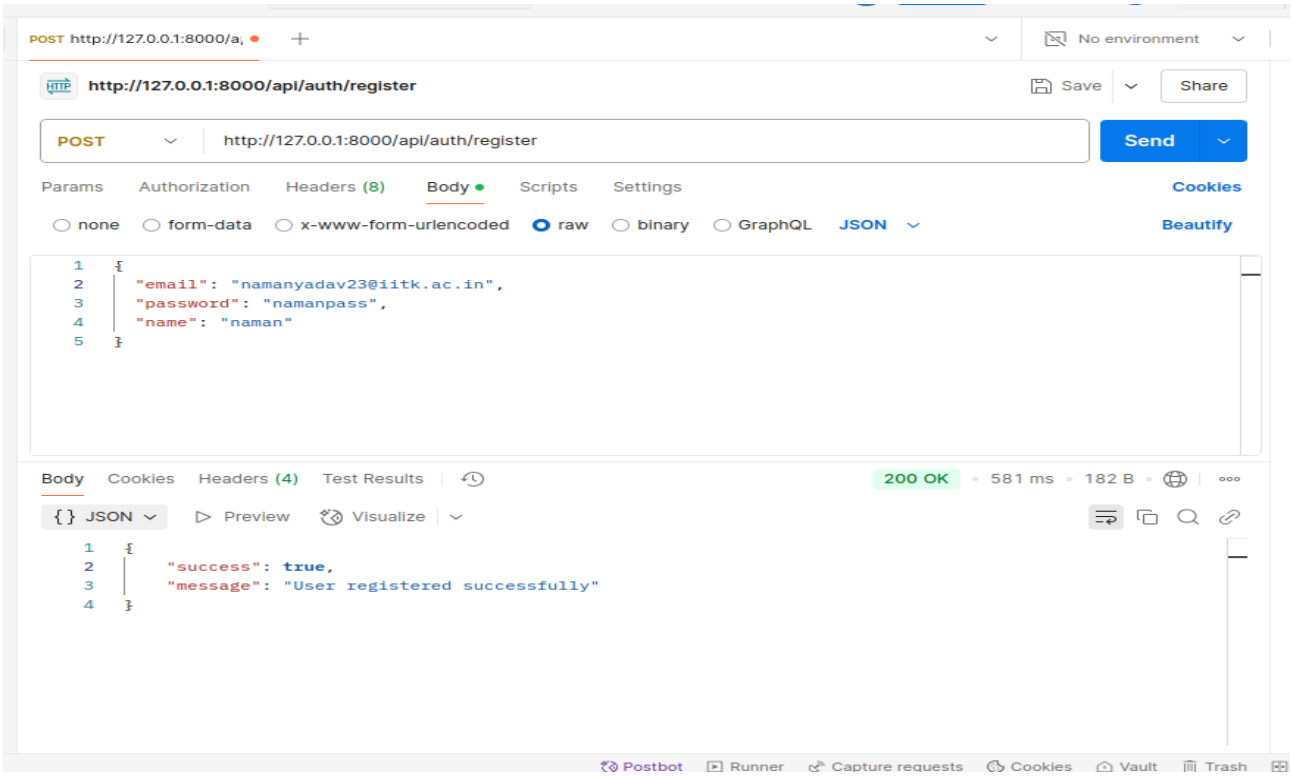
Authentication and Authorization

1. **JWT (JSON Web Token):** JWT is a stateless and scalable authentication method, eliminating the need for server-side session storage. Its compact format enables efficient transmission in HTTP headers, making it ideal for APIs, microservices, and SSO. Expiry and refresh tokens enhance security, but proper implementation is crucial to prevent token leakage and misuse.

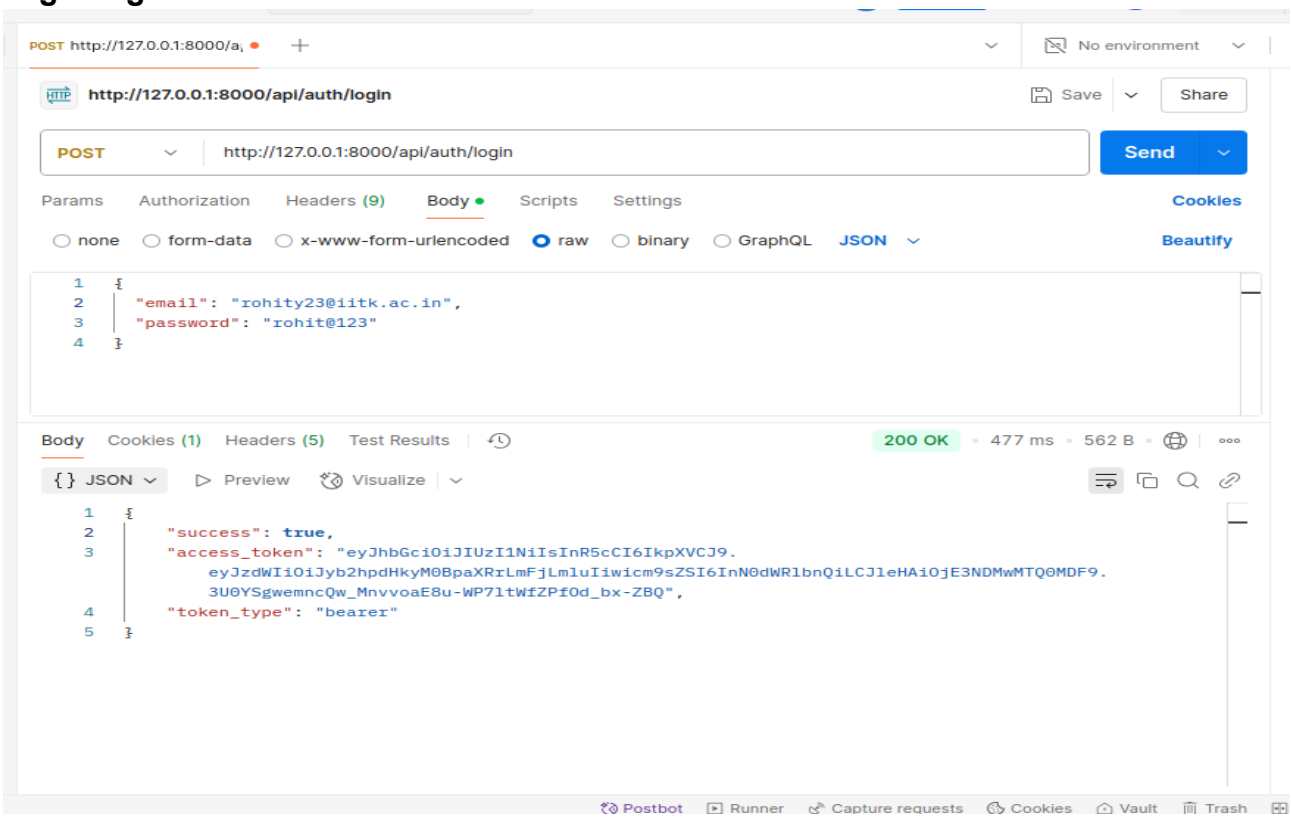
API Endpoints

- **SignUp/Register**

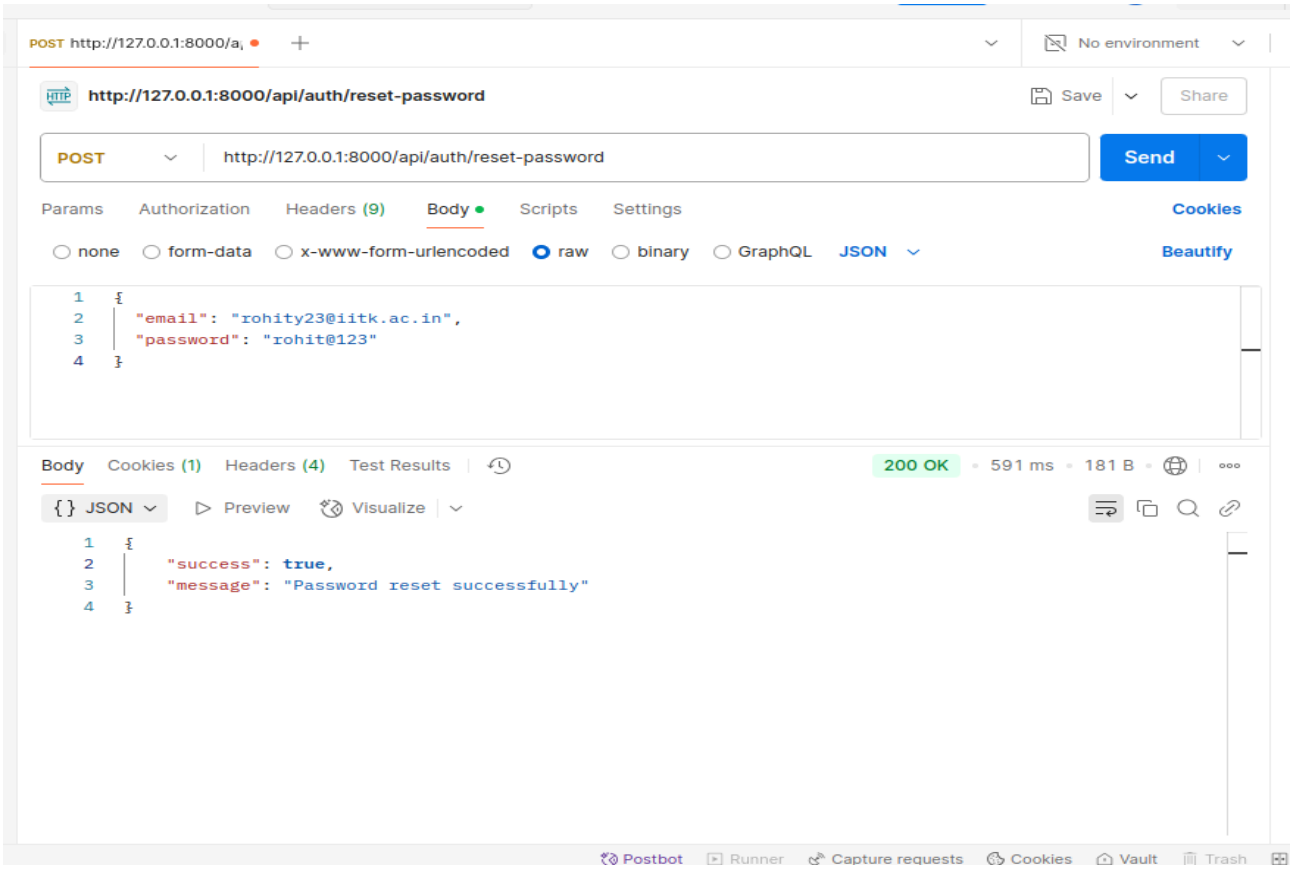




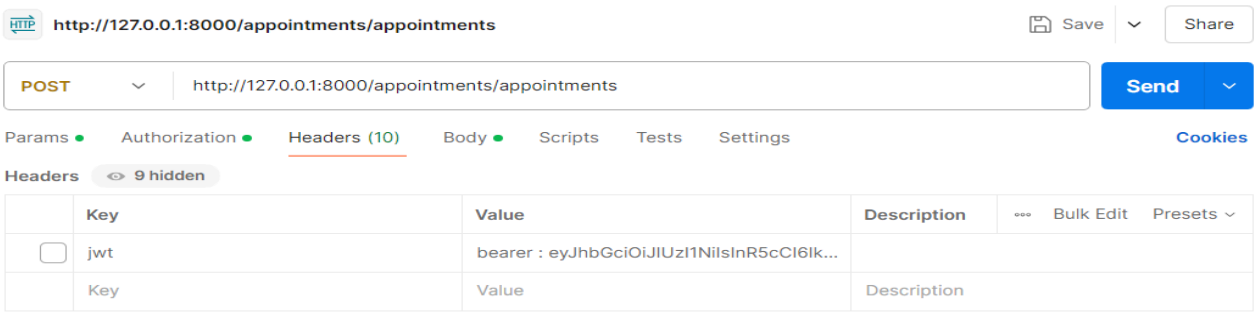
- **Login/SignIn**

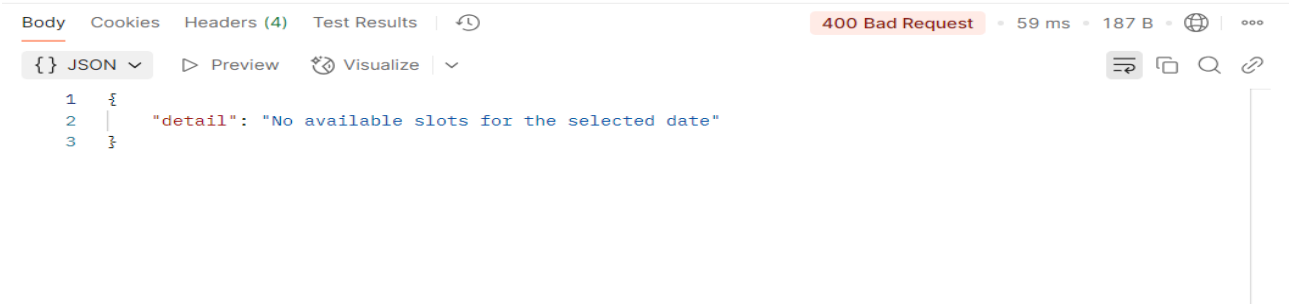


- **Reset Password**

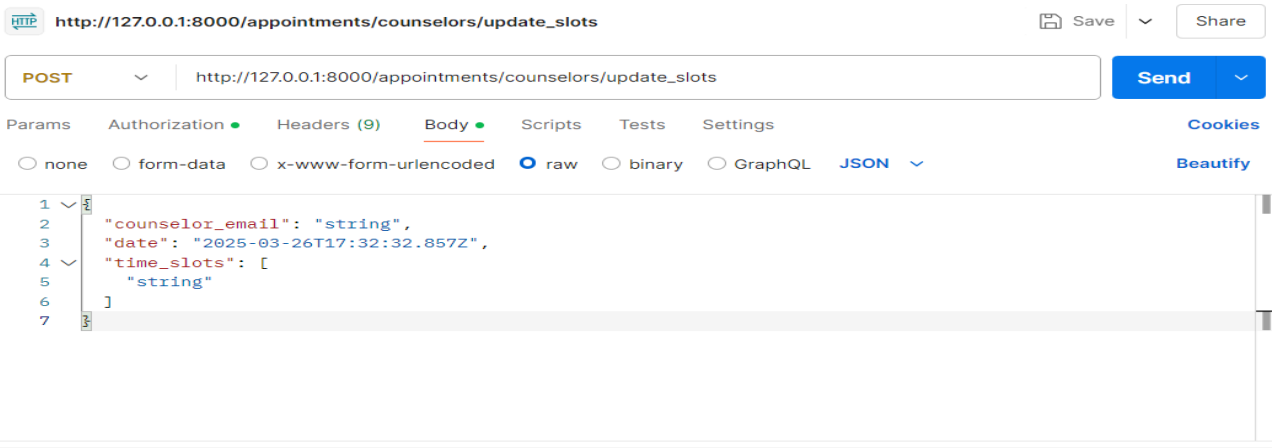


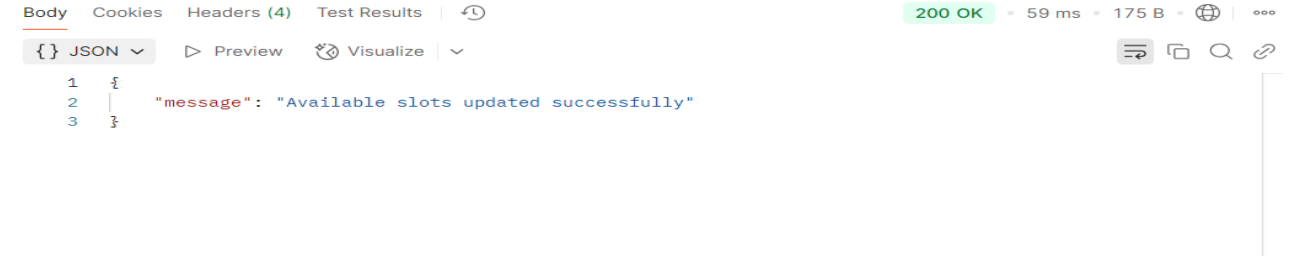
- **Book Appointments**



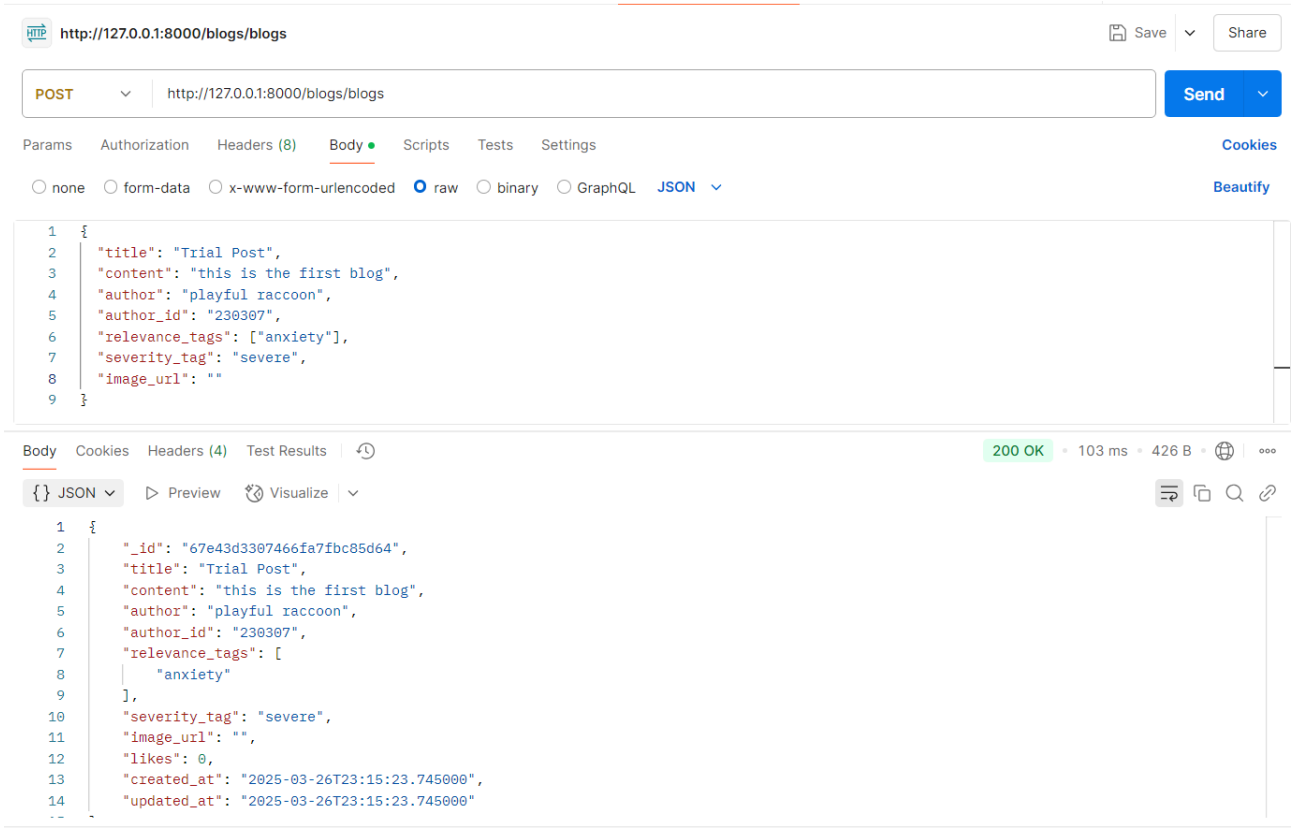


- **Update Counselor Slots**

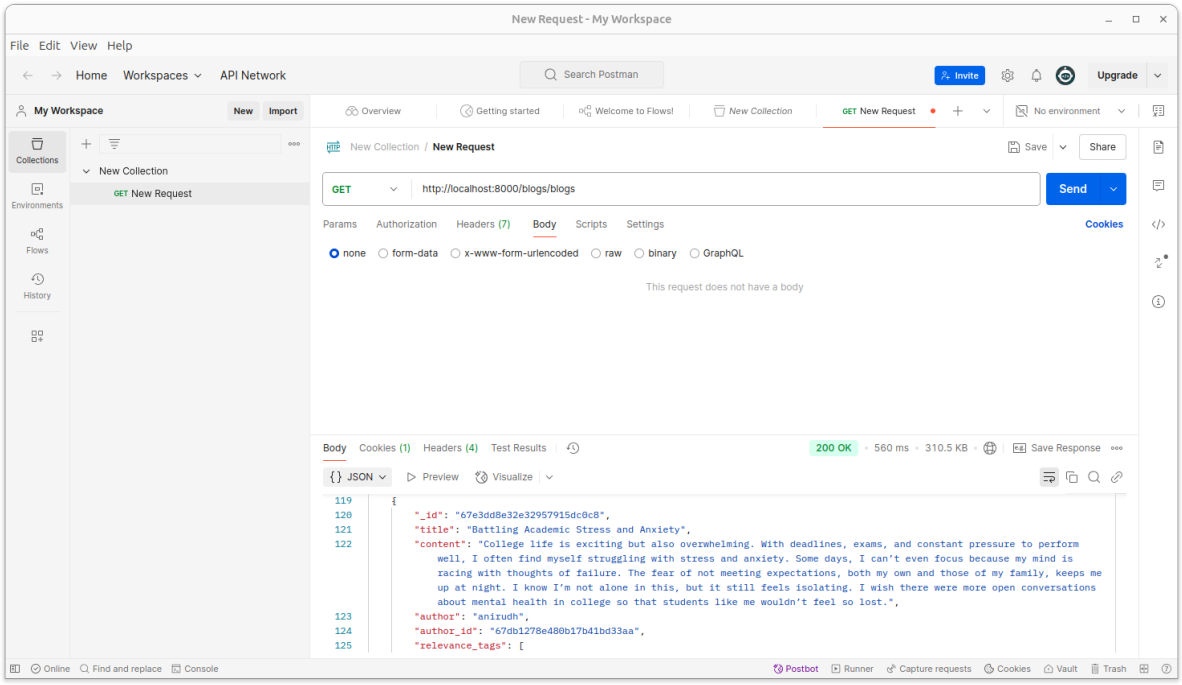




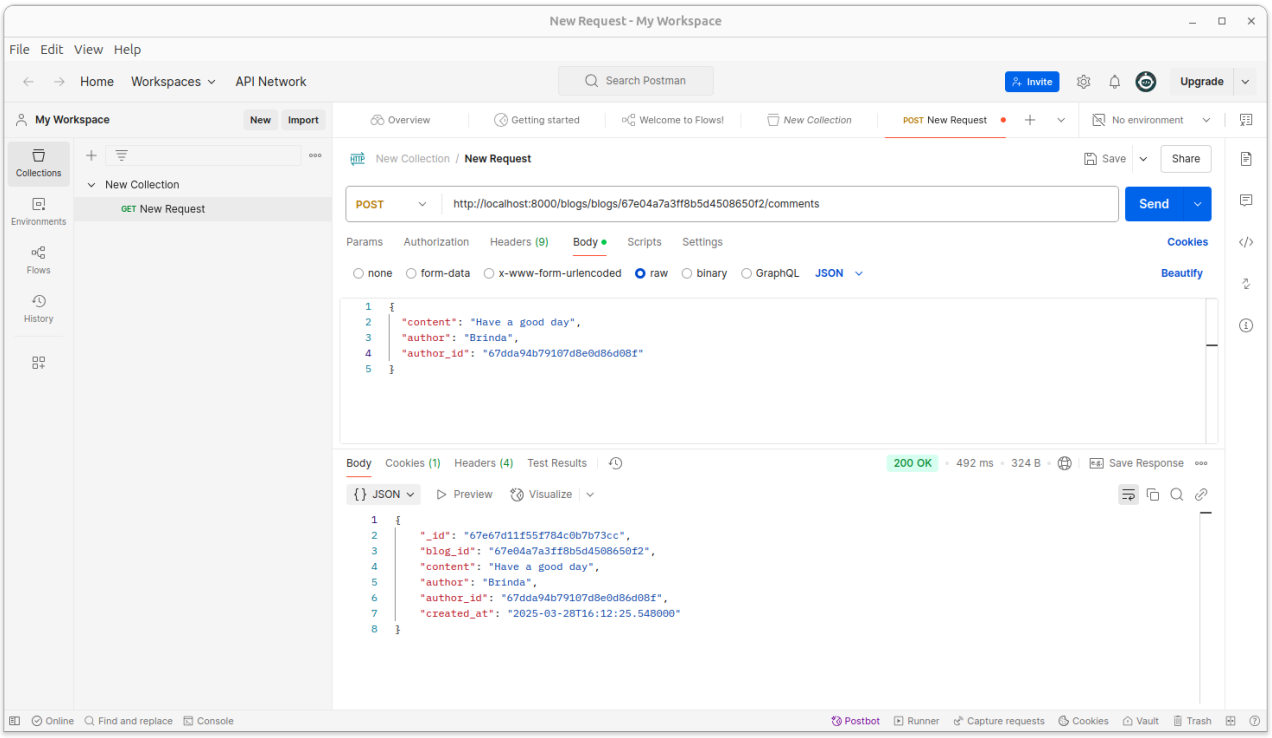
- **Create Post**



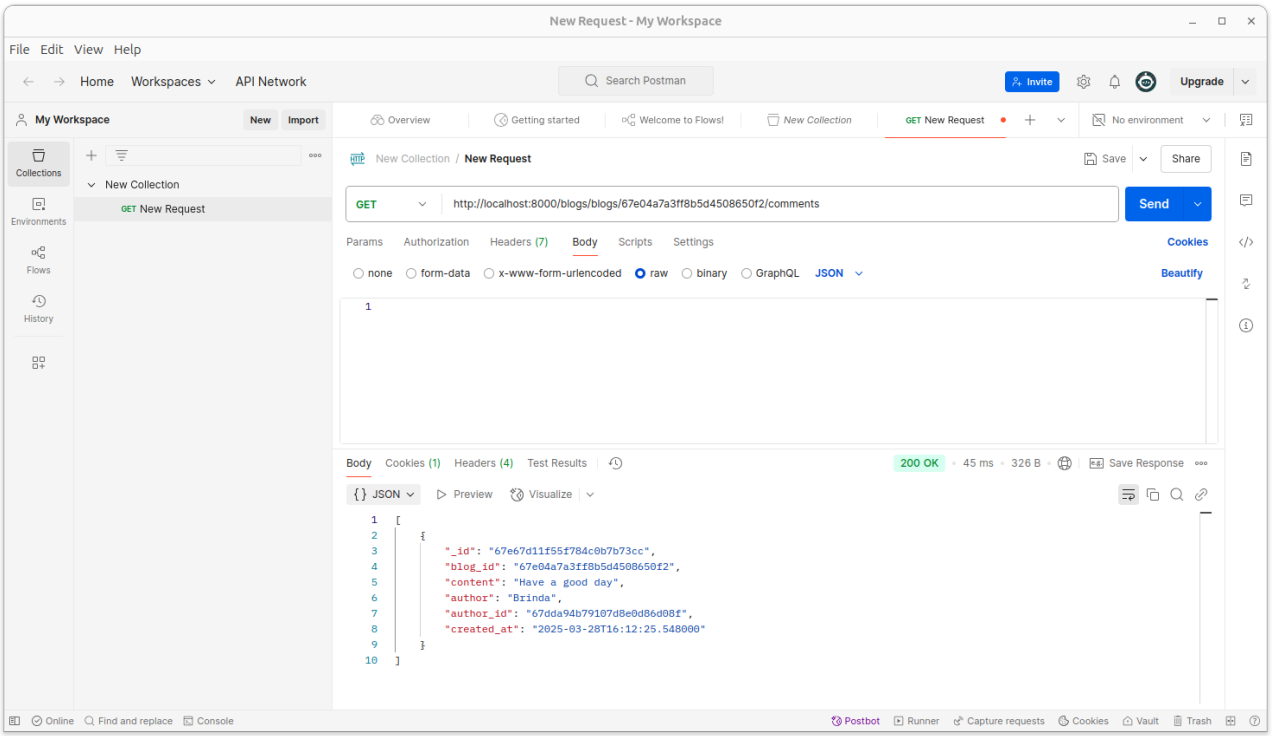
- **Get all the Posts**



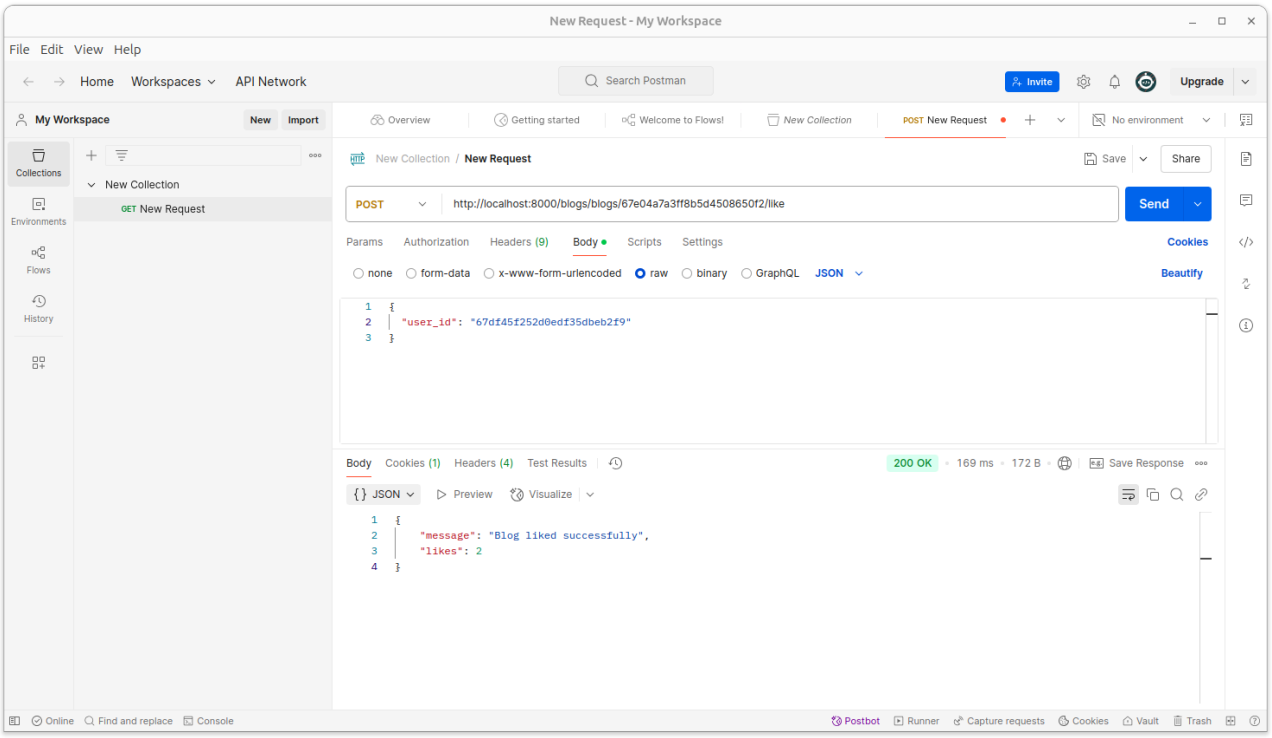
- **Add Comment to a post**



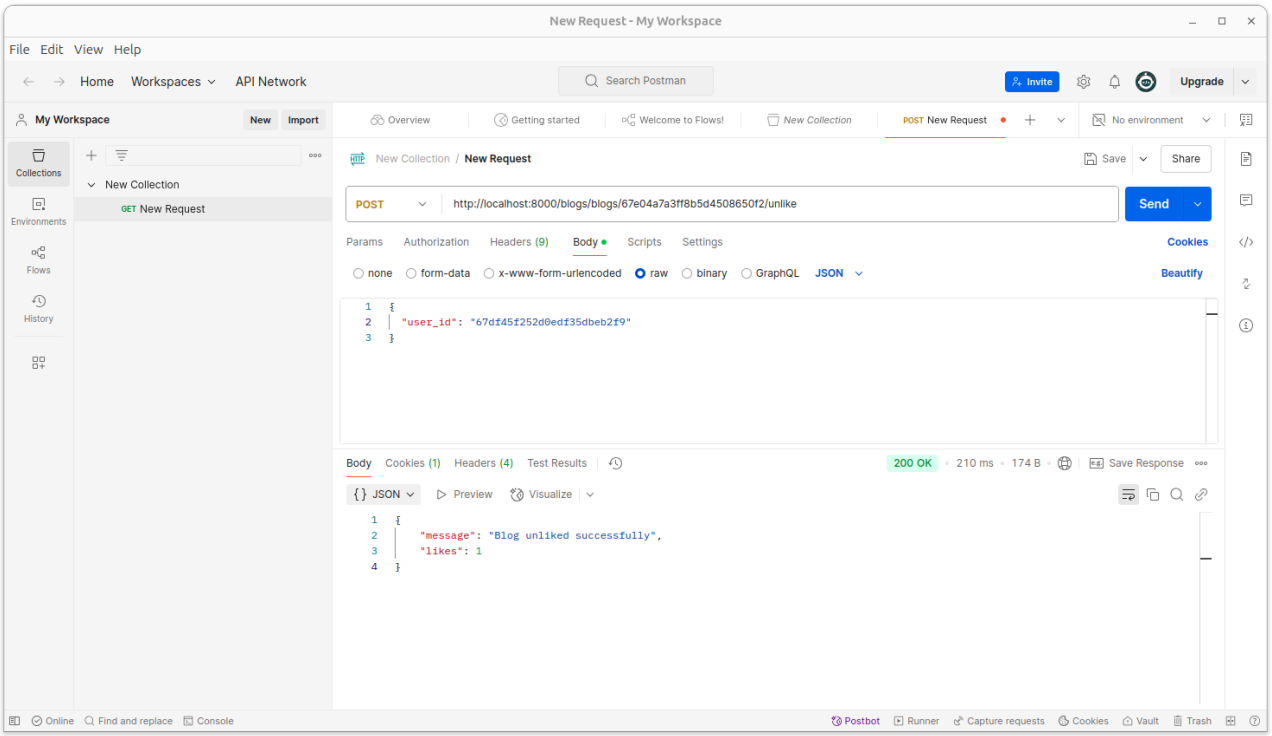
- **Get all the Comments for a post**



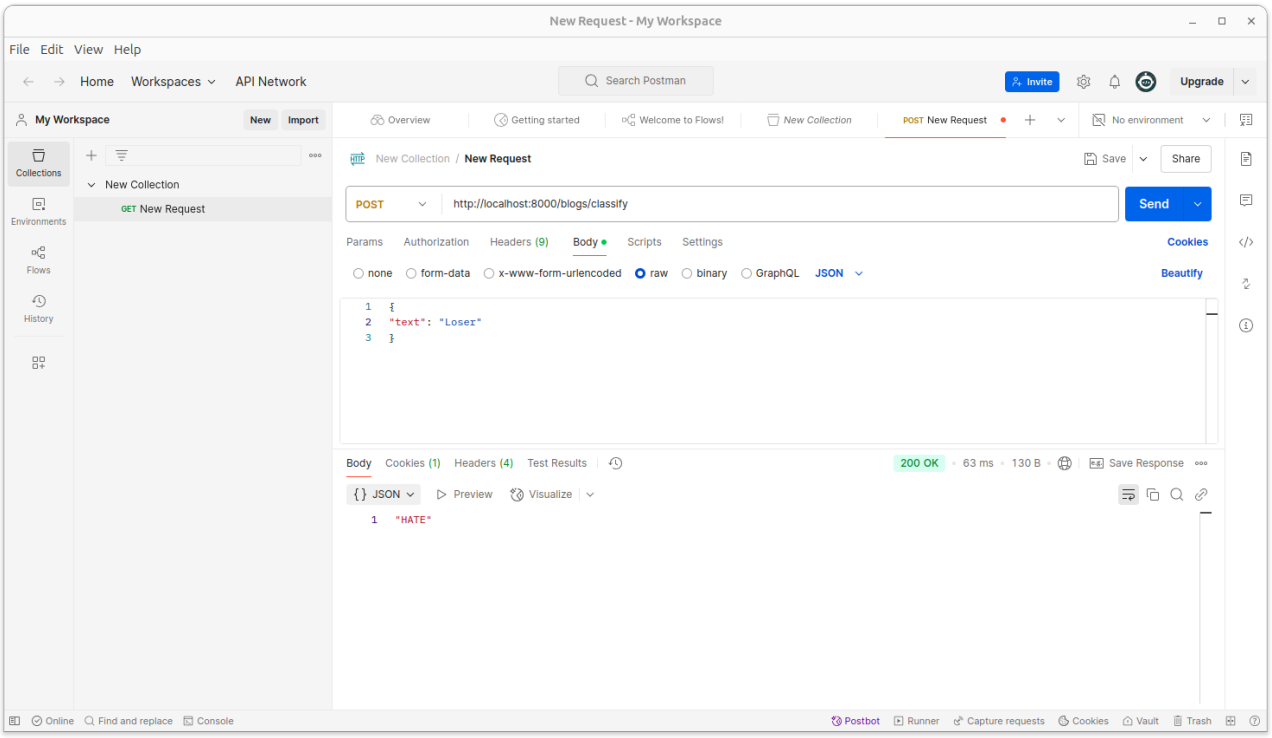
- **Like a Post**



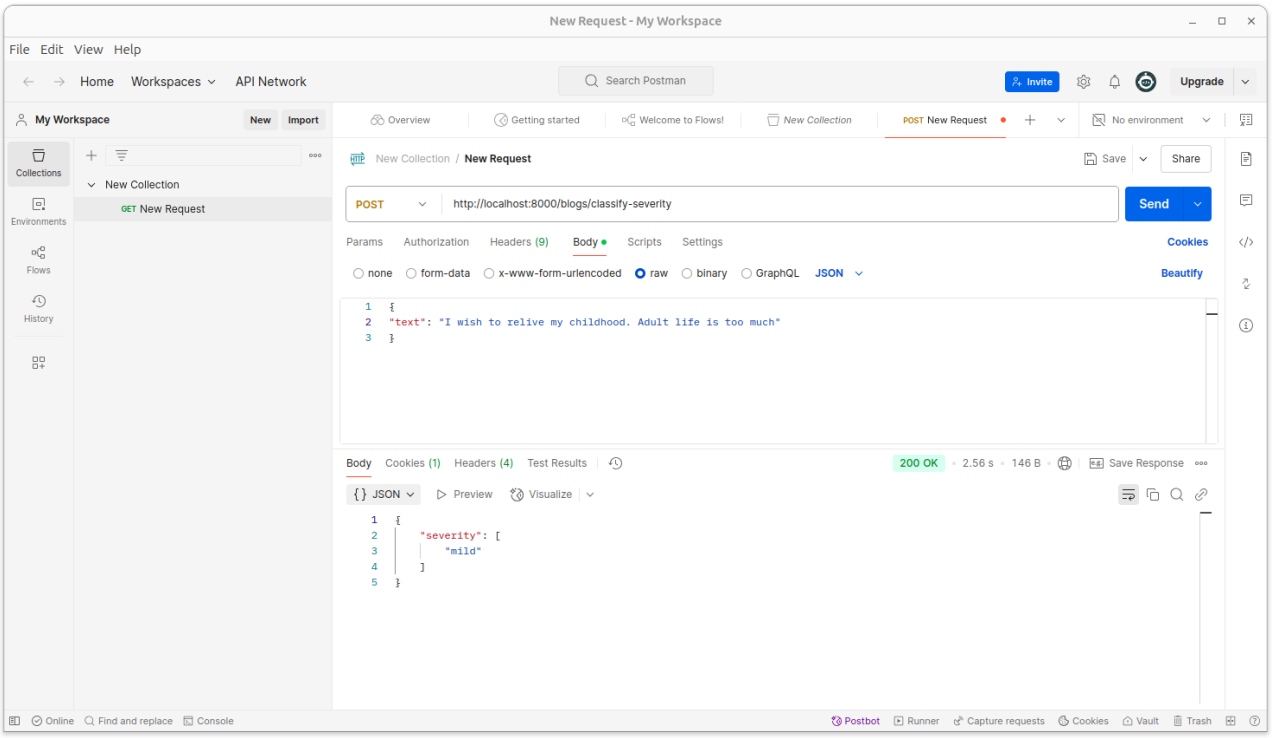
- **Unlike a Post**



- **Spam Filtering**



- **Severity Detection**

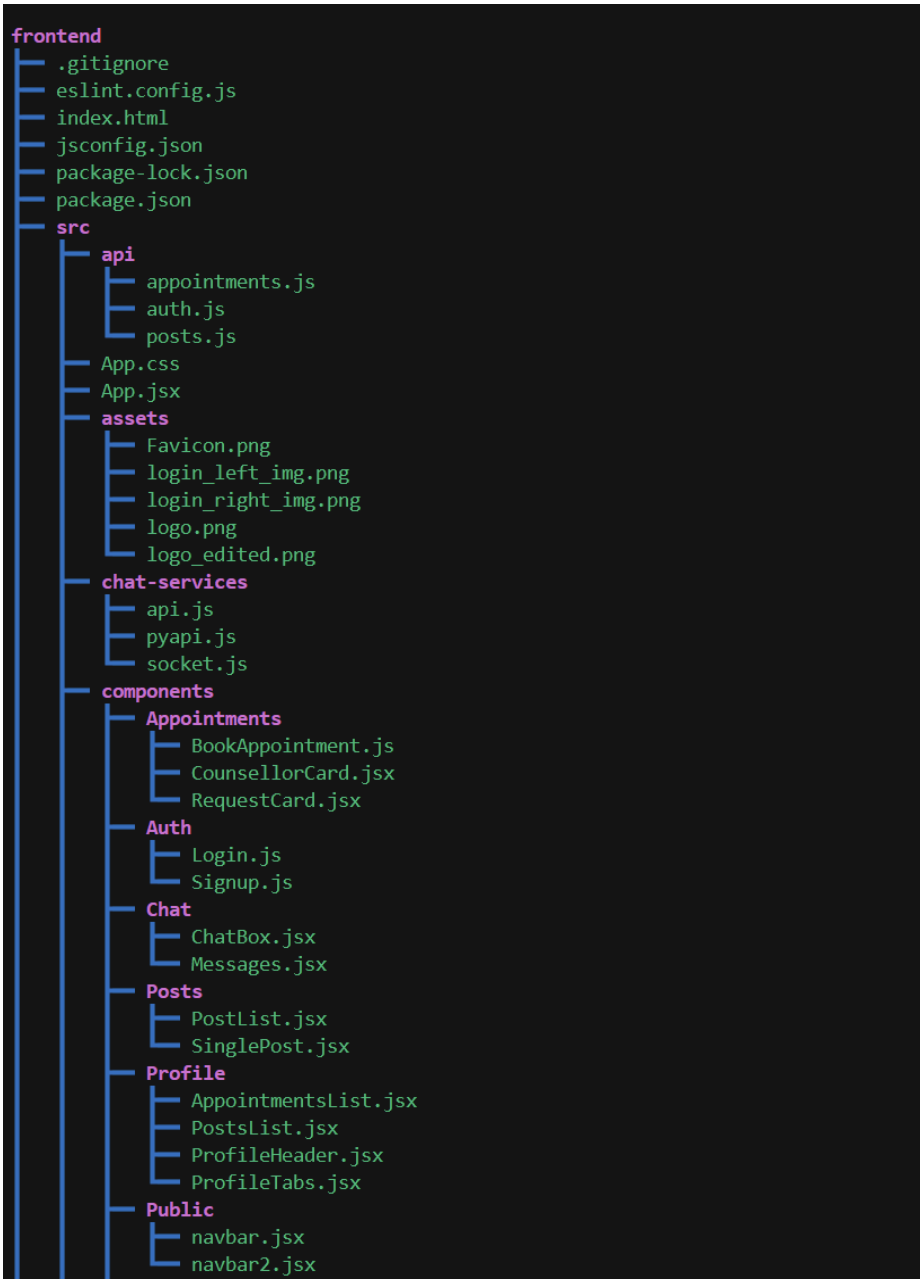


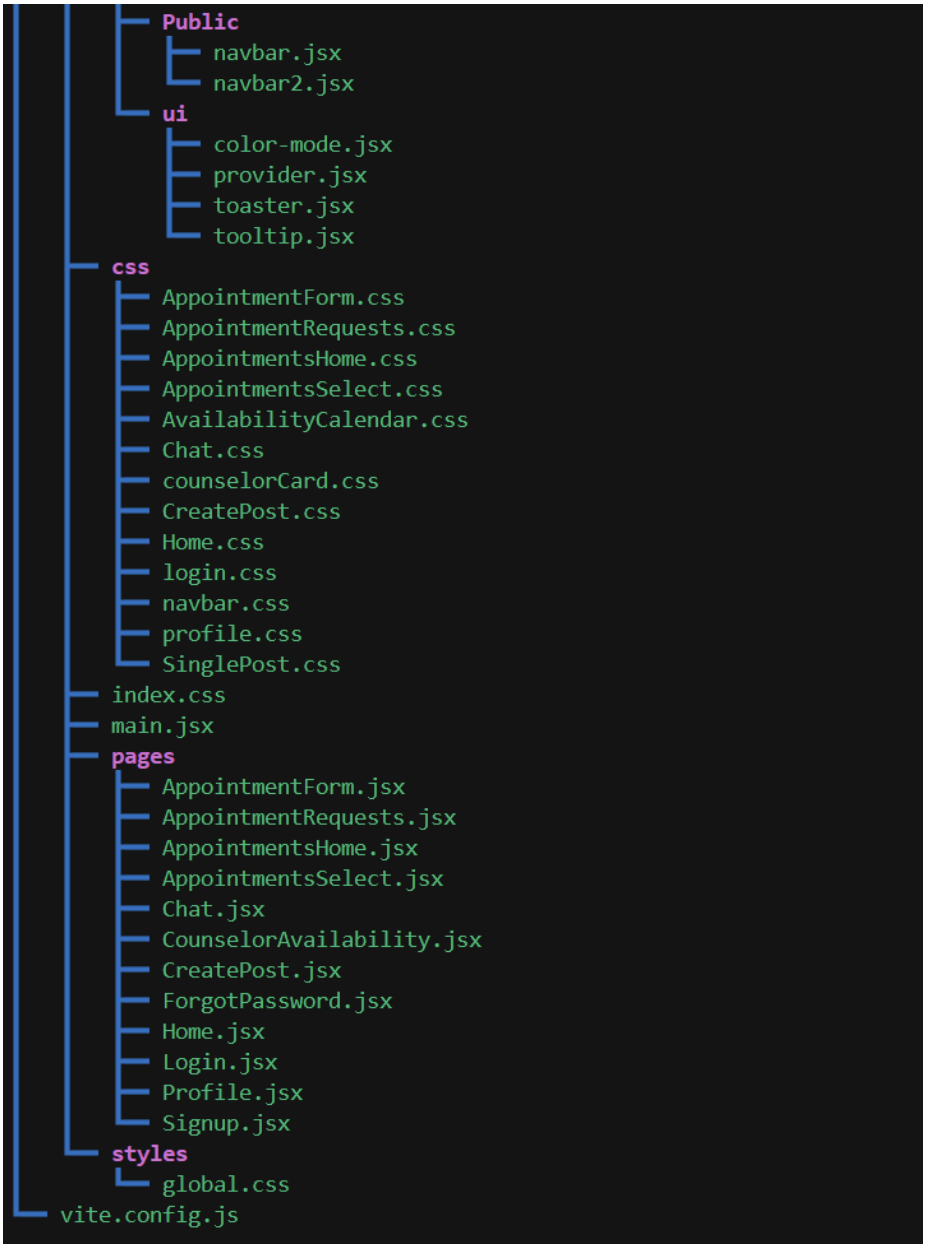
2 Codebase

GitHub link of code implementation: https://github.com/YatharthDX/SafeSpaceV2

We have structured a single repository to include both the backend and the react frontend. The repository is organized into separate directories for **Code Implementation** and **Documentation**, with the Code Implementation further divided into three components: frontend, backend, and chat-backend.

1. **Frontend:**



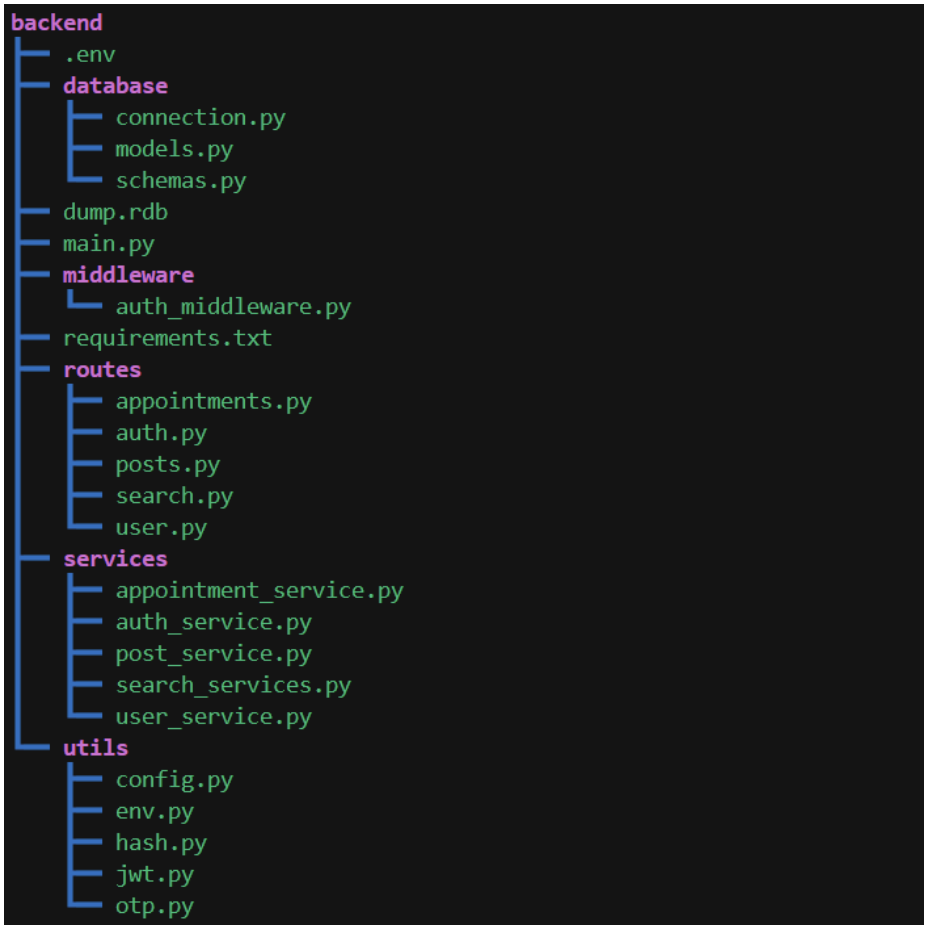


We have 67 files over 15 directories:

a. **index.html** - This is the primary web page loaded in the browser, where React components are injected and rendered.
b. **package.json** - Contains all the necessary dependencies that need to be installed before running the application.
c. **package-lock.json** - Locks the exact versions of installed npm dependencies to ensure consistent installs across all environments.
d. **src/api** - Contains modular functions that handle HTTP requests to the backend (using Axios), ensuring smooth frontend-backend communication.
e. **src/chat-services** - Manages chat-related API calls, authentication, and real-time communication to support messaging and user interactions.
f. **src/components** - Contains all the react components that are used while the app is in use.

g. **src/css** - Contains css files for all the components used.
h. **src/pages** - Contains the page-level components that define the different routes of the application, such as home, login, chat, appointments or post creation pages.

2. Backend:



We have 34 files over 11 directories:

a. **backend/.env** - Stores sensitive configuration details like database URLs, Redis settings, secret keys, and email credentials, keeping them separate from the source code for security and easy environment setup.

b. **backend/main.py** - Initializes the FastAPI app, configures middleware (CORS & custom auth handling), sets up route modules for authentication, users, posts, appointments, and search, and defines a health check endpoint.

c. **backend/requirements.txt** - Lists all the python packages and their versions required to run the app.

d. **backend/database** - Manages connections to MongoDB and Redis and defines Pydantic models to structure, validate, and serialize data across the application.

e. **backend/routes** - Organizes and handles all API endpoints, mapping user actions to corresponding backend services.

f. **backend/services** - Contains the core business logic for handling authentication, post management, appointment scheduling, search operations, and counselor role functionalities, abstracting database operations from route handlers.

g. **backend/utils** - Contains utility modules for configuration management, JWT authentication, OTP generation & email handling, password hashing, and environment setup, providing essential reusable functions across the application.

3. Chat-Backend:

a. **chat-backend/src/controllers** - Handles chat user fetching, message retrieval, sending messages with optional image upload, and real-time message delivery via WebSocket.

b. **chat-backend/src/lib** - Houses core utilities for database connection, cloud image upload and real-time WebSocket setup & online user management.

c. **chat-backend/src/models** - Defines mongoose schemas for user and message entities, enforcing structure, validation, and relationships for database documents.

d. **chat-backend/src/routes** - Defines message-related API endpoints with authentication middleware to handle user fetching, message retrieval, and message sending.

e. **chat-backend/src/index.js** - Bootstraps Express server with middleware, DB connection, message routes, and WebSocket for real-time chat.

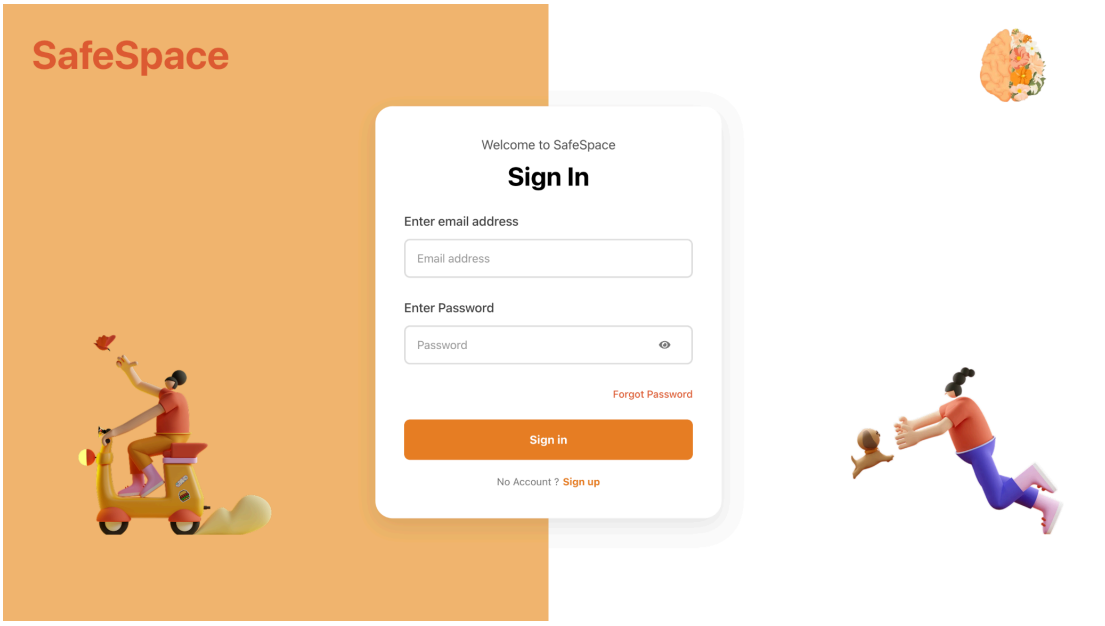
Completeness

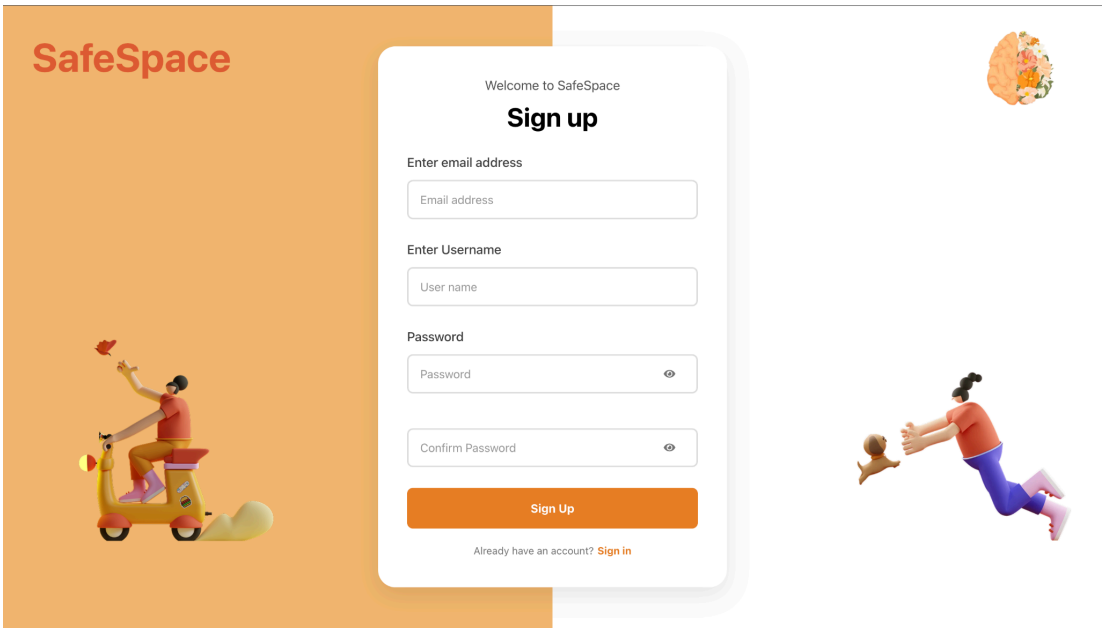
The “Section 3: Specific Requirements” of Software Requirements Specification Document lists all the desired product functionality. Let us go over them one at a time.

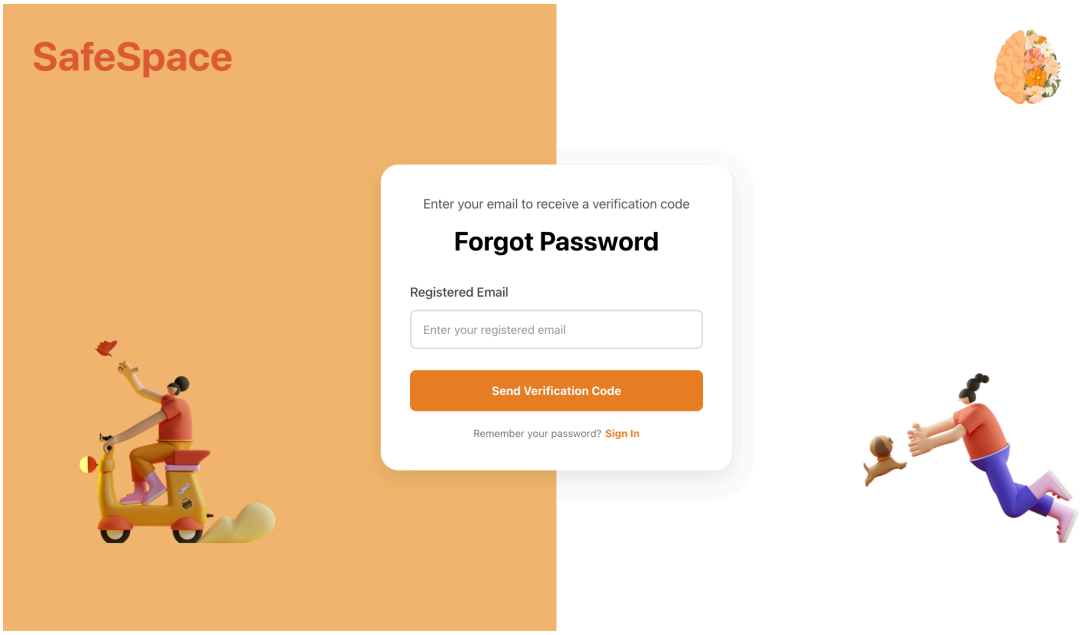
External Interface Requirements:

1. **User Interfaces:**

    - **Login Page, Sign-Up and Forgot Password Page :**

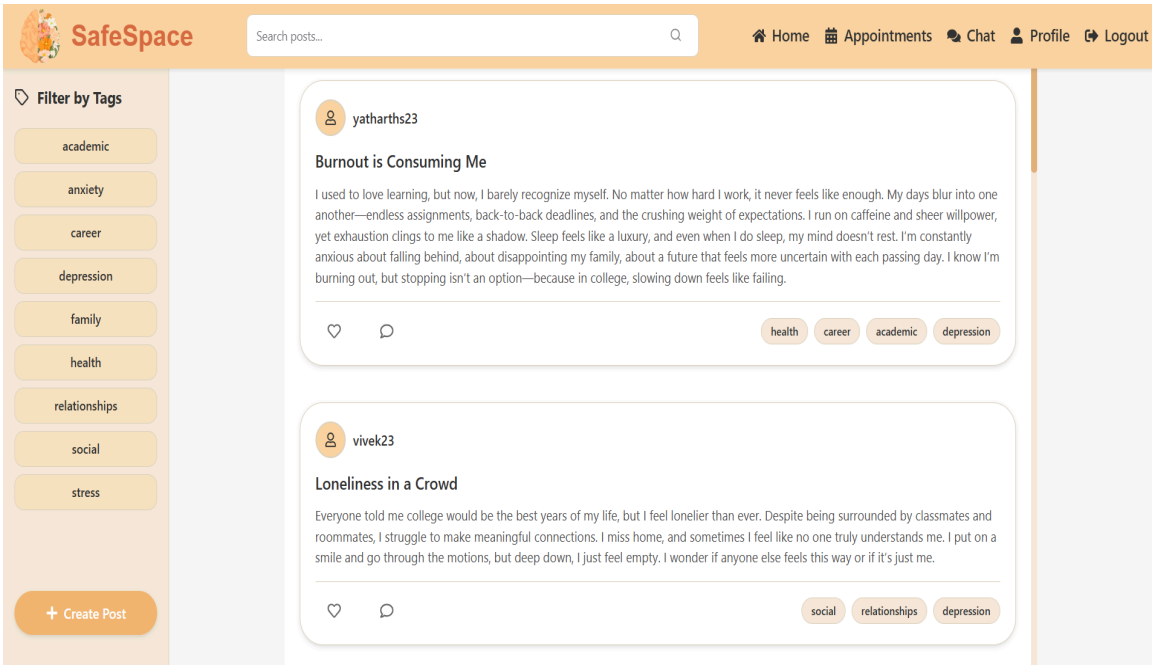






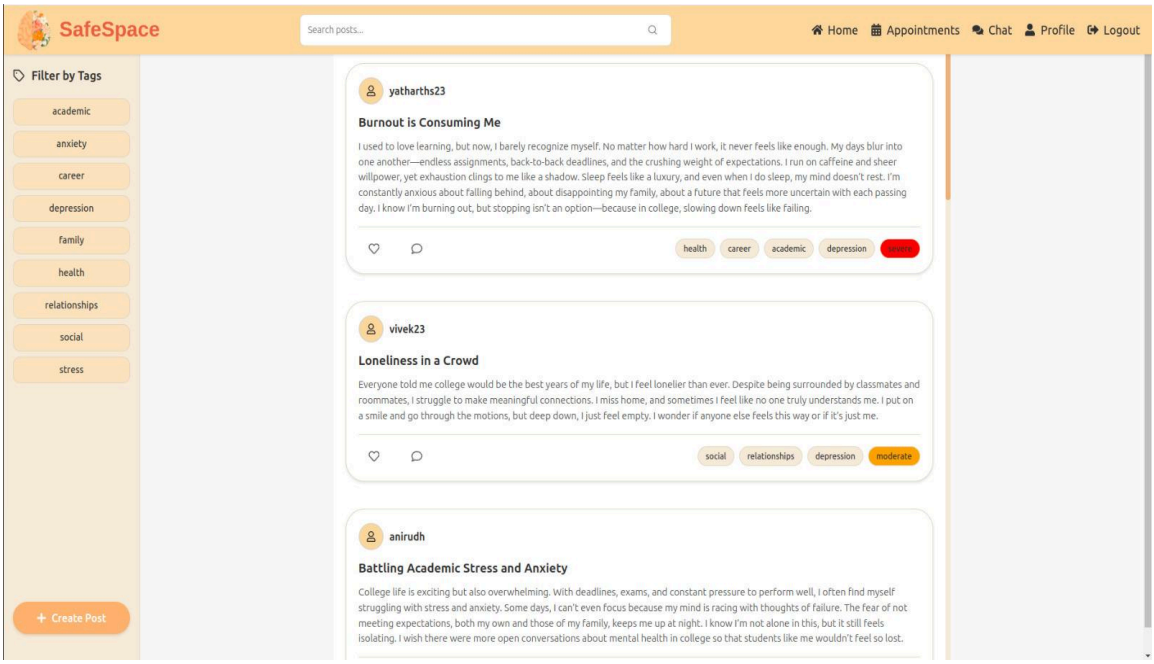
The **Login Page** serves as the landing page where users enter their credentials to access the platform. If not registered, they can navigate to the **Sign-Up Page** to create an account. The **Forgot Password Page** allows users to reset their password by receiving an OTP via email for secure authentication.

- **Homepage:**
  - **For Students**



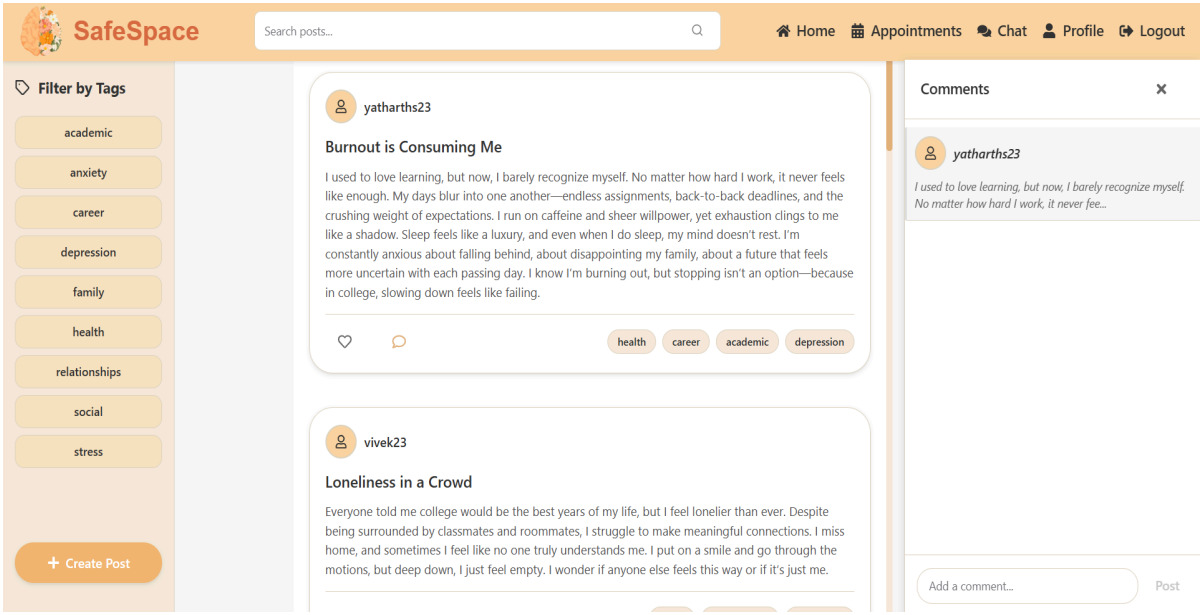
This is the homepage for students where they land after logging in

  - **For Counselor**

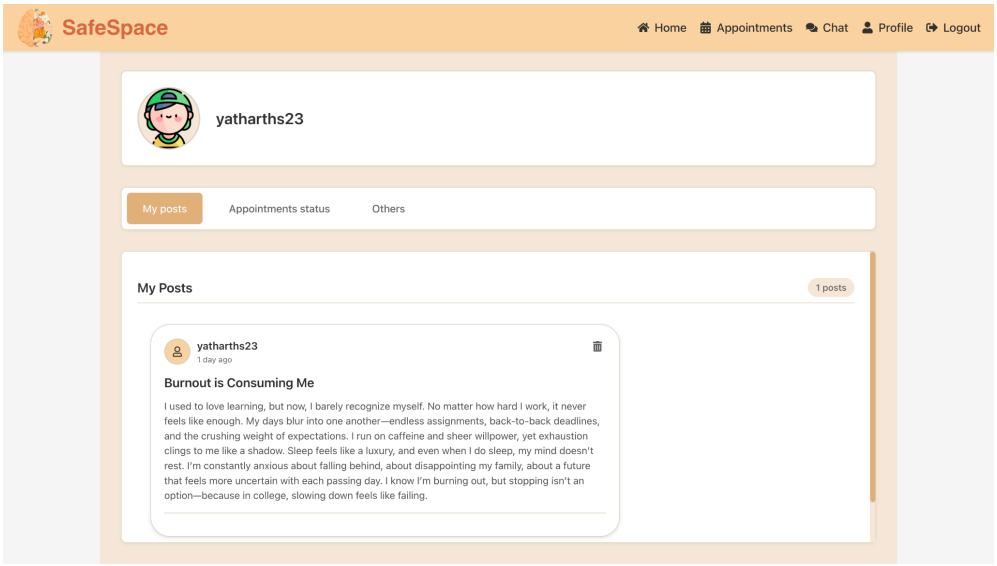


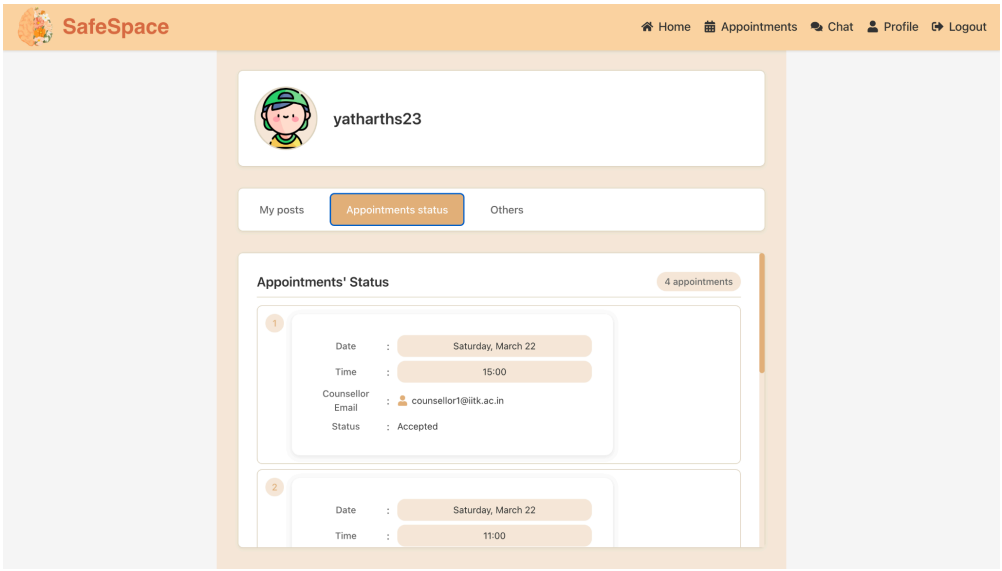
This is the homepage for counselors with an added feature of severity tags on posts.

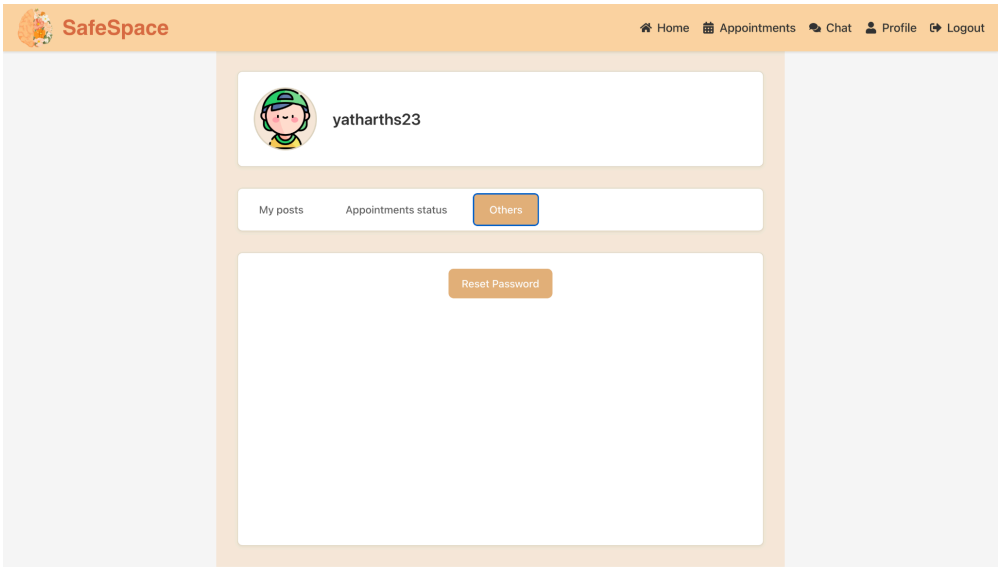
- **Comments**



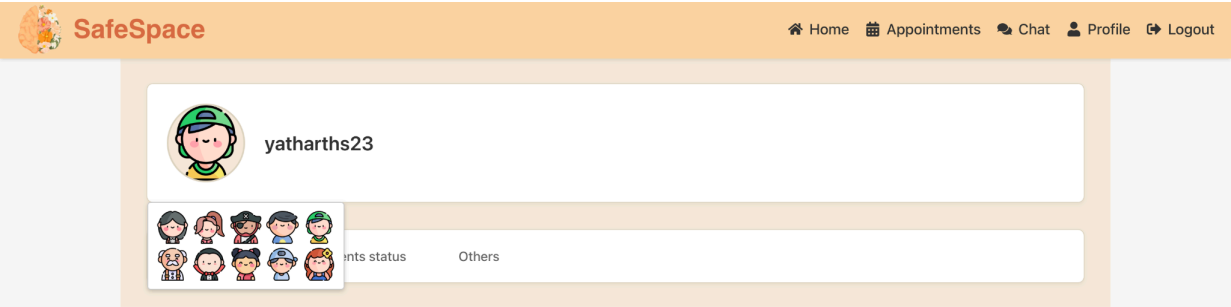
- **User Profile:**


SafeSpace
Home
Appointments
Chat
Profile
Logout
yatharths23
My posts
Appointments status
Others
My Posts
1 posts
yatharths23
1 day ago
Burnout is Consuming Me
I used to love learning, but now, I barely recognize myself. No matter how hard I work, it never feels like enough. My days blur into one another—endless assignments, back-to-back deadlines, and the crushing weight of expectations. I run on caffeine and sheer willpower, yet exhaustion clings to me like a shadow. Sleep feels like a luxury, and even when I do sleep, my mind doesn't rest. I'm constantly anxious about falling behind, about disappointing my family, about a future that feels more uncertain with each passing day. I know I'm burning out, but stopping isn't an option—because in college, slowing down feels like failing.


SafeSpace
Home
Appointments
Chat
Profile
Logout
yatharths23
My posts
Appointments status
Others
Appointments' Status
4 appointments
1
Date : Saturday, March 22
Time : 15:00
Counsellor Email : counsellor1@iitk.ac.in
Status : Accepted
2
Date : Saturday, March 22
Time : 11:00

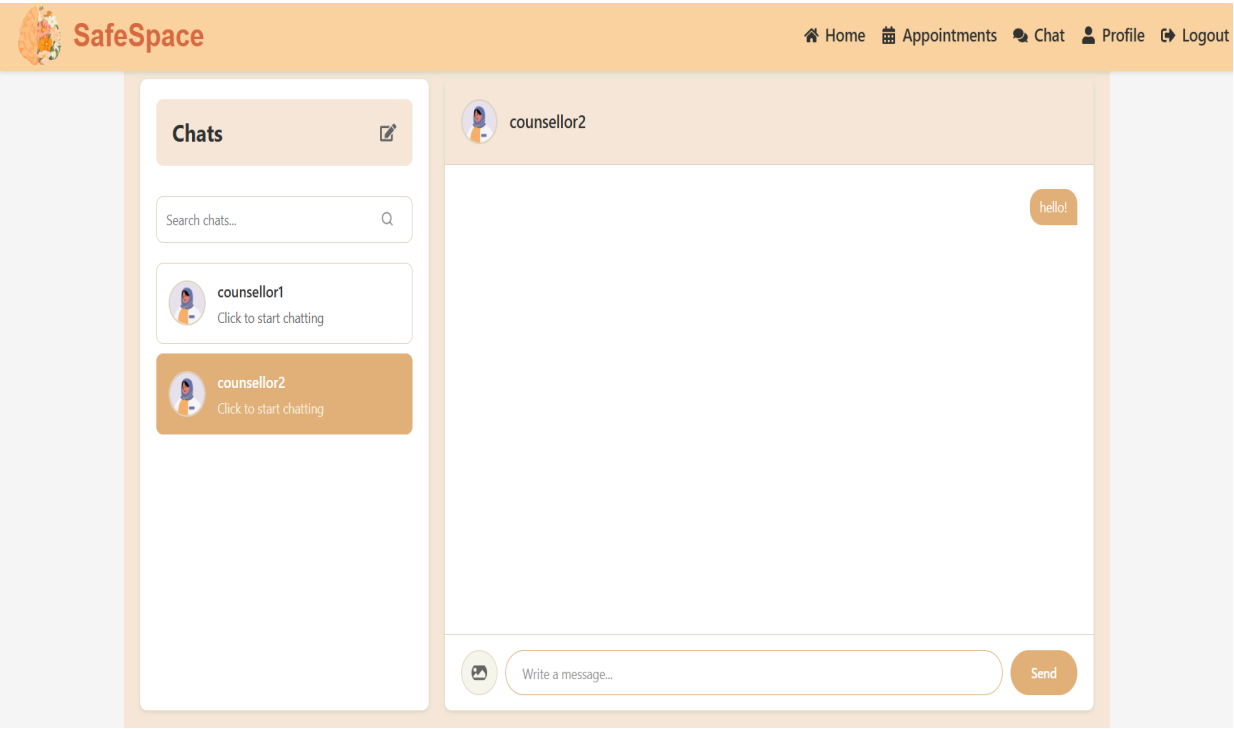

SafeSpace
Home
Appointments
Chat
Profile
Logout
yatharths23
My posts
Appointments status
Others
Reset Password

- **Edit Avatar from Profile:**



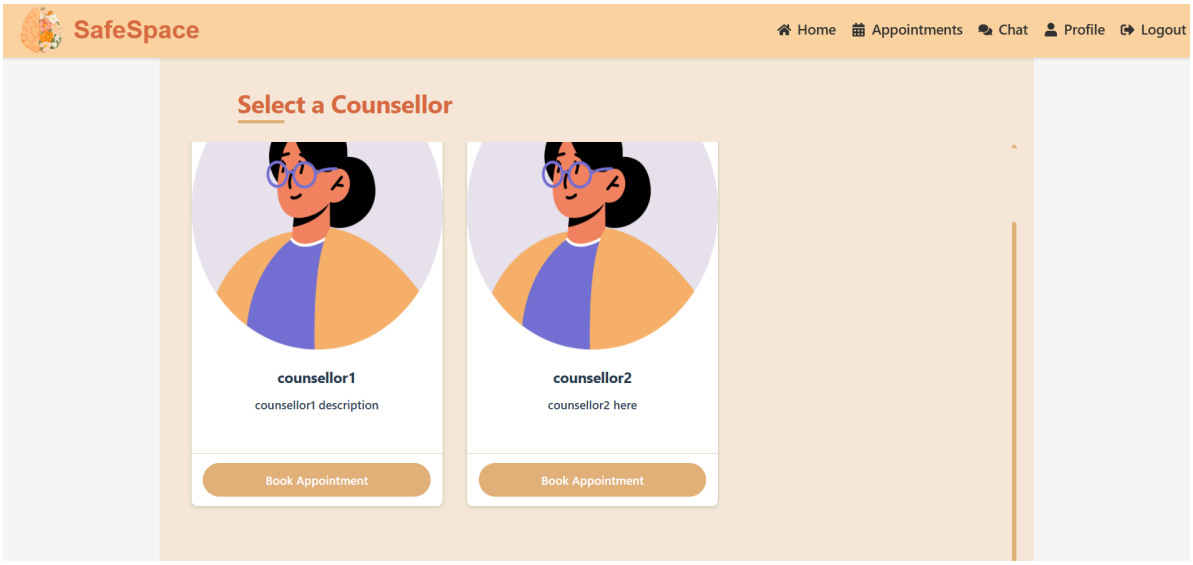
Allows user to change their avatar by clicking on the profile icon on profile page

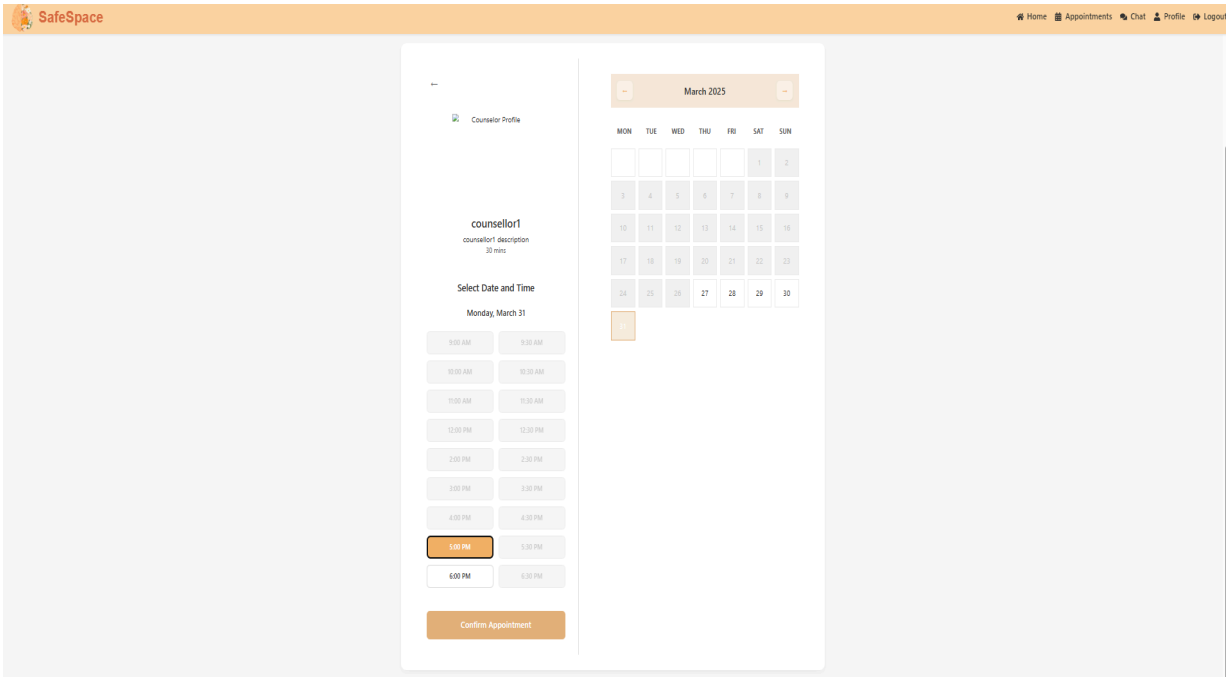
- **Chat:**

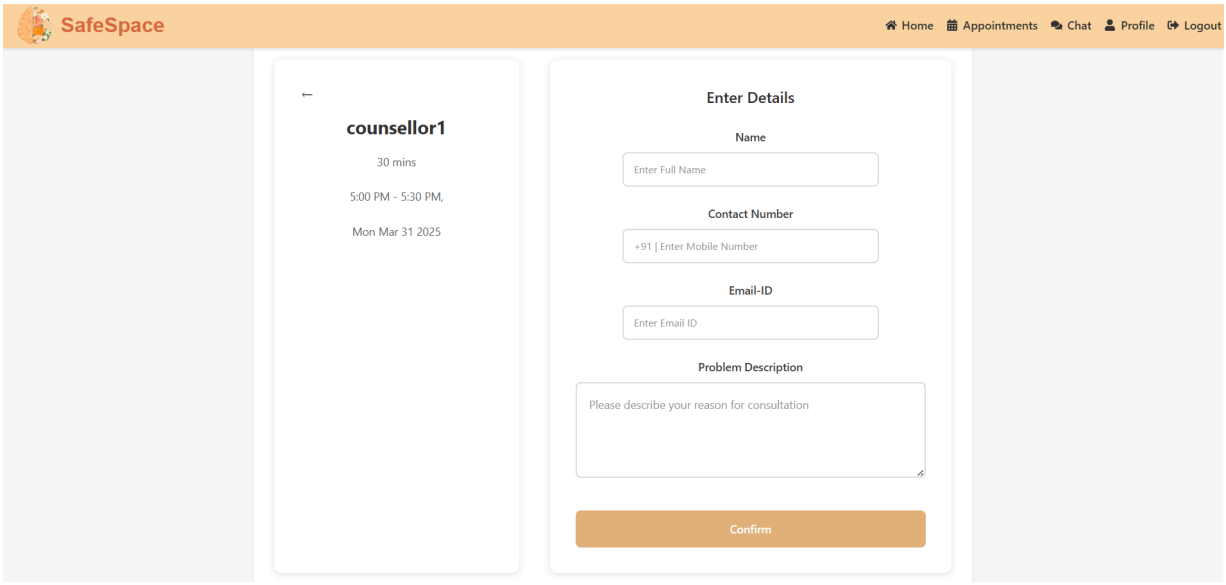


Chat feature allows the user to chat with any of the counselors, with the list of available counselors displayed on the left.

- **Appointments:**

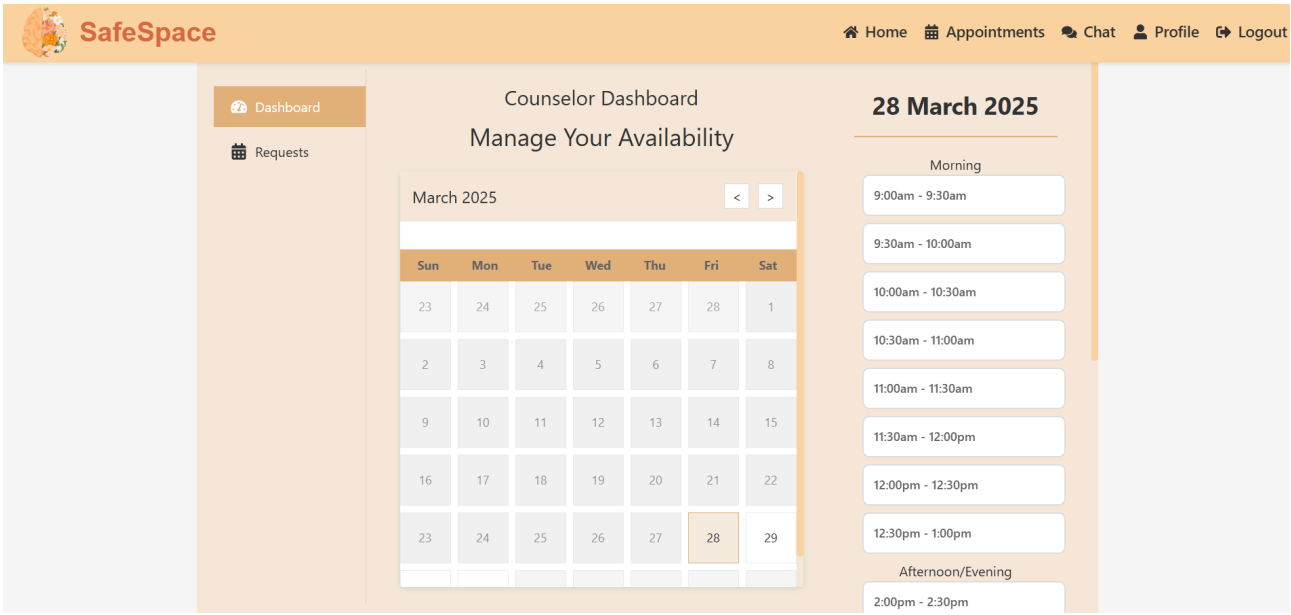

SafeSpace
Home
Appointments
Chat
Profile
Logout
Select a Counsellor
counsellor1
counsellor1 description
Book Appointment
counsellor2
counsellor2 here
Book Appointment


SafeSpace
Home
Appointments
Chat
Profile
Logout
Counselor Profile
counsellor1
counsellor1 description
30 mins
Select Date and Time
Monday, March 31
9:00 AM
9:30 AM
10:00 AM
10:30 AM
11:00 AM
11:30 AM
12:00 PM
12:30 PM
2:00 PM
2:30 PM
3:00 PM
3:30 PM
4:00 PM
4:30 PM
5:00 PM
5:30 PM
6:00 PM
6:30 PM
Confirm Appointment
March 2025
MON
TUE
WED
THU
FRI
SAT
SUN

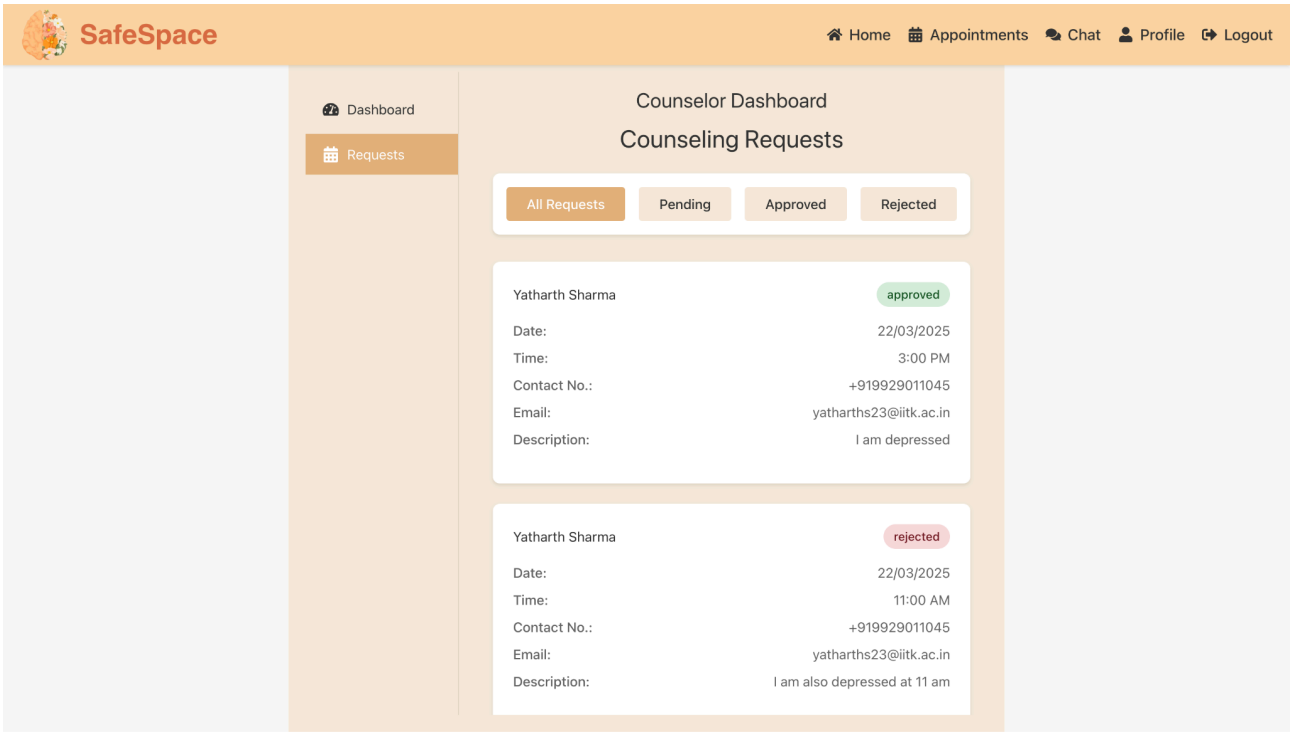


Appointment feature allows the user to book appointments with any counselor they want in any of their available slots.

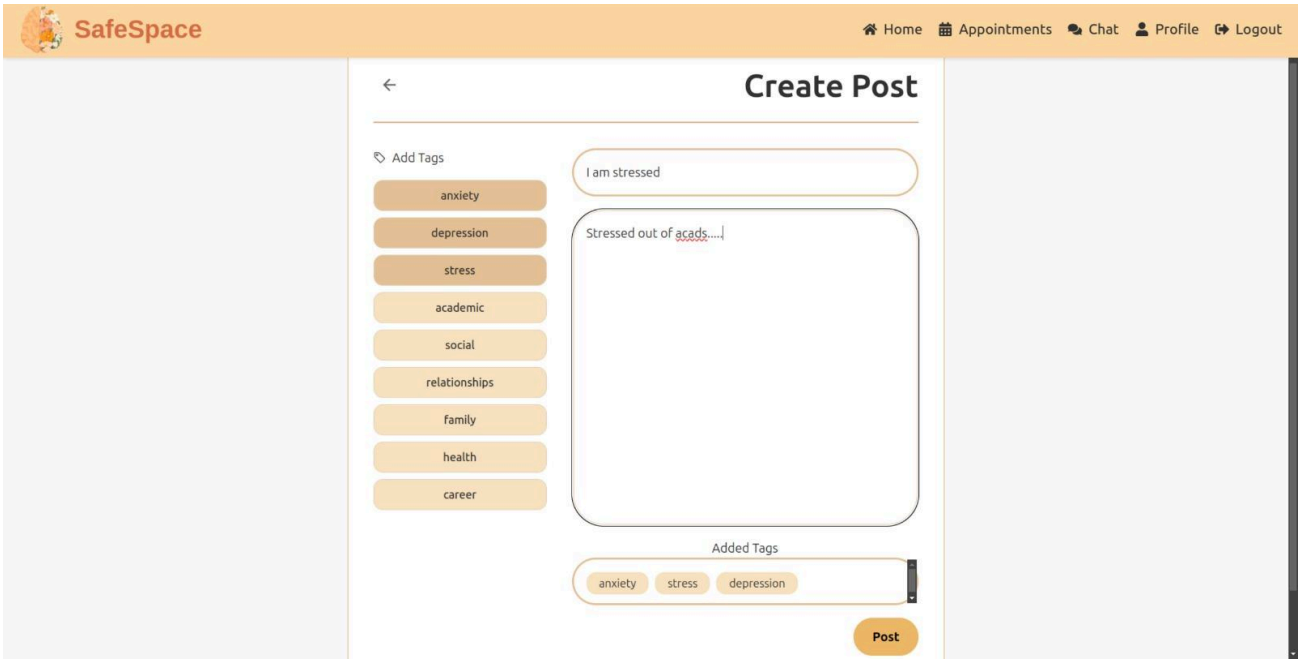
- **Manage Counselor Availability**



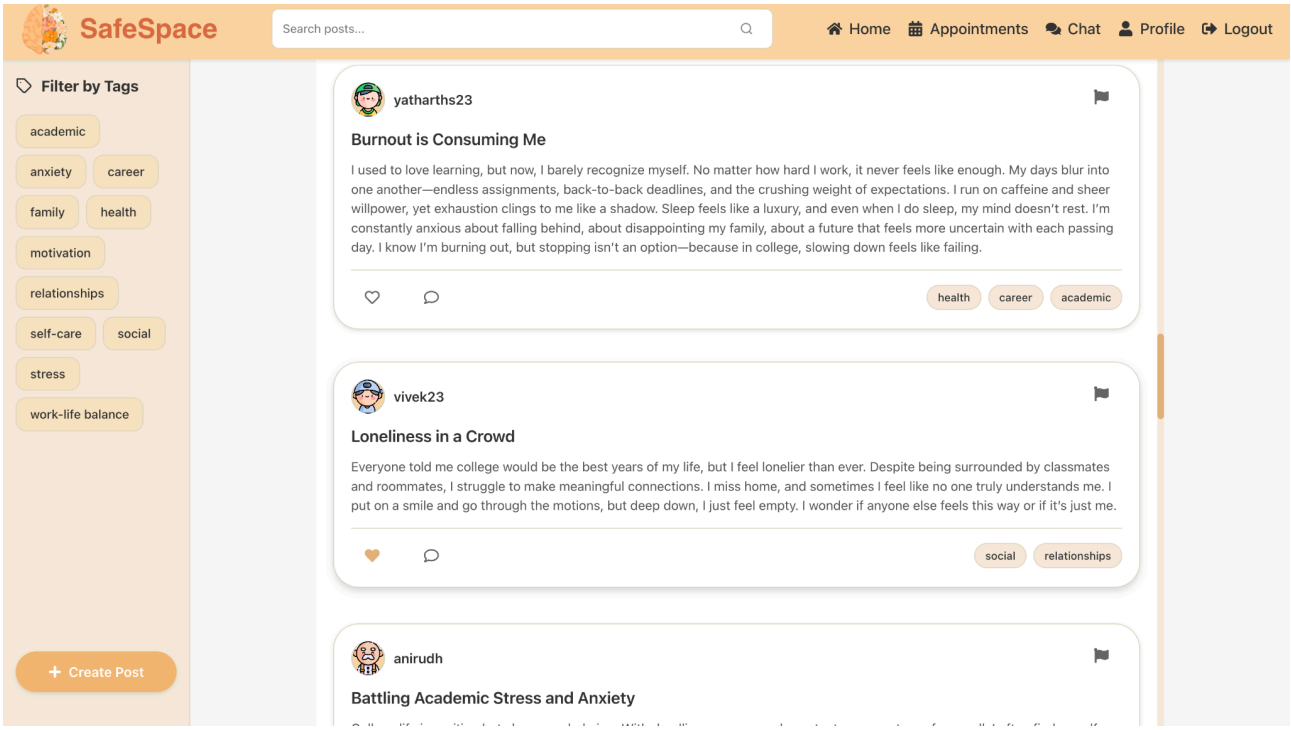
- **Approving/Rejecting Appointment Requests**



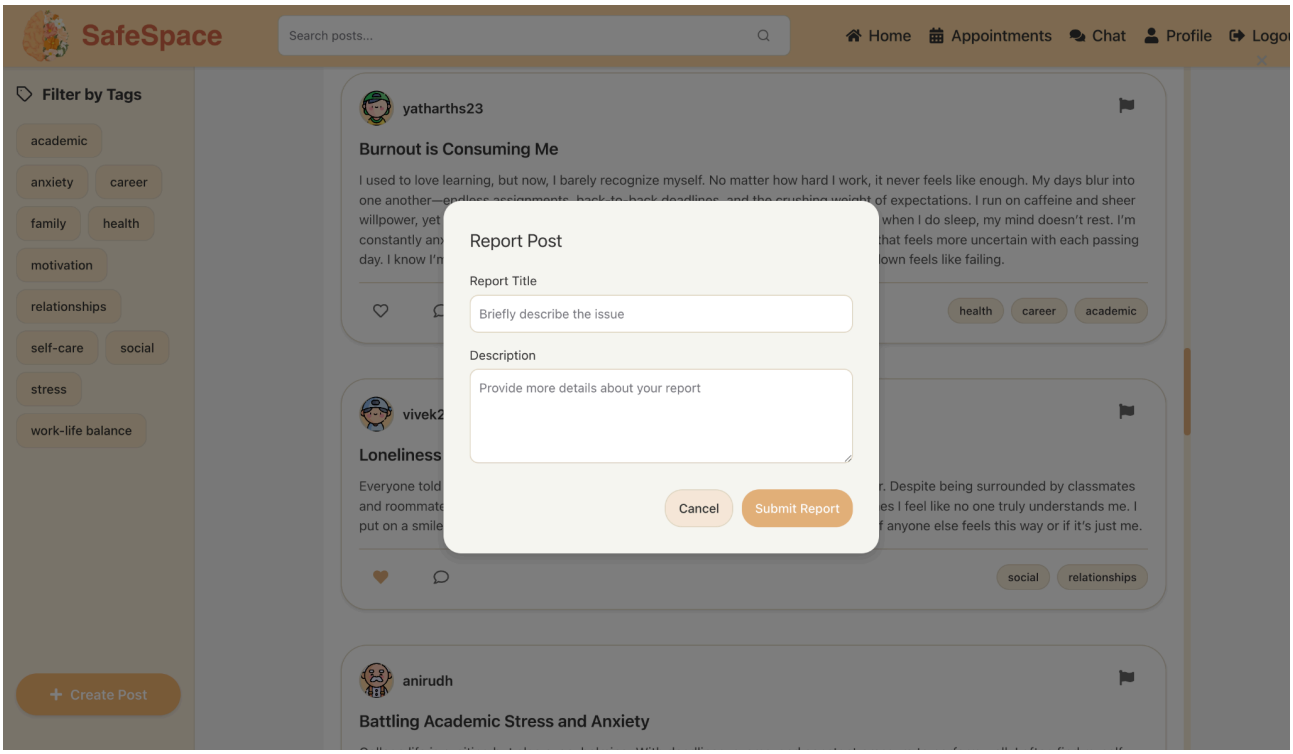
- **Create Post:**



- **Comment or Like in Posts:**



- **Report Posts:**



2. Functional Requirements:

All of these have been implemented in :

- User Registration and Authentication
- Changing Password for Log In
- Chat with Counselors(for students) and vice-versa
- Search by name in chat
- Composing Posts (counselors can add images as well)
- Comment or Like in Posts
- Spam Filtering and Severity-Detection in posts and comments
- Searching Blogs by Tags and Title
- Booking Appointments by Students
- Scheduling Time-Slots by Counselors
- Accepting Appointment Requests
- Edit avatar
- Report posts

Future Development:

1. **Edit Posts:** The functionality of editing posts is still not implemented. We can improve upon our code and open this option to the users in the next versions.
2. **Generate Personalised Feed:** Implement an ML model that provides the user a personalised blog feed based on the posts and tags they previously interacted with.
3. **Online Appointment:** Availing the feature of online appointments with the counselors.
4. **Notification System:** The user gets notifications when they receive a message, or their appointments are approved or when someone comments on their post.

Appendix A - Group Log

Team A : Anirudh Singh, Archita Goyal, Brinda Fadadu, Nakul Patel, Naman Yadav.
Team B : Om Chaudhari, Rohit Yadav, Sayani Patra, Vivek, Yatharth Sharma.

Sr. No.	Date	Description
1	01/03/25	Allocated frontend and backend work to each member of the group.
2	03/03/25	Discussion regarding the workflow of design development
3	04/03/25	Discussion regarding the workflow of chat system and homepage
4	06/03/25	Updates regarding the work progress from both teams
5	08/03/25	Updates regarding the work progress from both teams
6	11/03/25	Discussion regarding components of the frontend
7	15/03/25	Completed the frontend with some final touches
8	18/03/25	Updates regarding work progress on appointments page
9	20/03/25	Completion of backend for homepage and appointments
10	22/03/25	Completion of the backend for chat system

11	23/03/25	Integration of frontend and backend
12	25/03/25	Fixed some bugs and tested the functioning
13	26/03/25	Did some final changes and implemented and updated them in the document