
Implementation Document

for

ConnVerse

Version 1.0

Prepared by Team MahaDevS

Group #3:

Aaditi Anil Agrawal
Arshit
Chayan Kumawat
Dobariya Jenil Bharatbhai
Harsh Agrawal
Harshit
Harshit Srivastava
Naman Kumar Jaiswal
Prem Kansagra
Priyanshu Singh

220006
220209
220309
220385
220425
220436
220444
220687
220816
220830

Group Name: *MahaDevS*

aaditiaa22@iitk.ac.in
arshitsk22@iitk.ac.in
chayank22@iitk.ac.in
dobariyajb22@iitk.ac.in
harshag22@iitk.ac.in
harshit22@iitk.ac.in
harshitsr22@iitk.ac.in
namankj22@iitk.ac.in
premk22@iitk.ac.in
spriyanshu22@iitk.ac.in

Course: CS253

Mentor TA: *Abhilash*

Date: March 18, 2024

Table of Contents

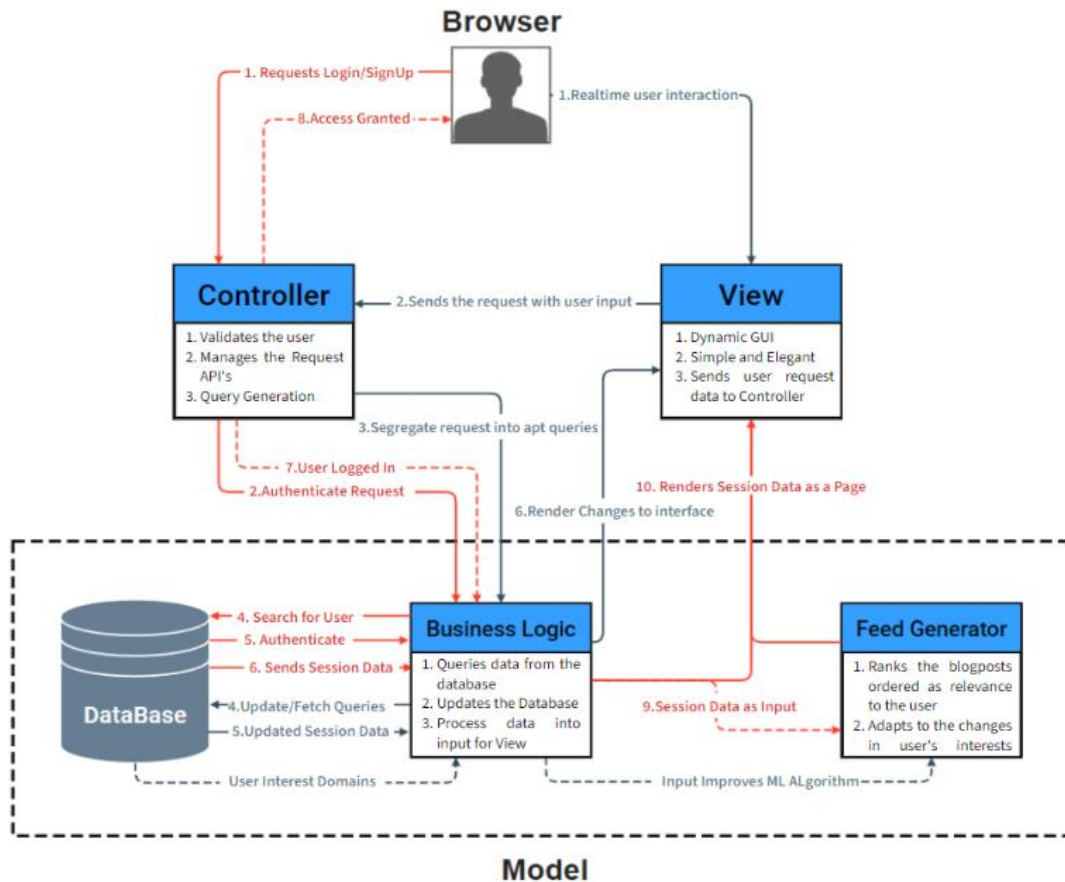
CONTENTS	II
REVISIONS	II
1 IMPLEMENTATION DETAILS.....	1
2 CODEBASE	2
3 COMPLETENESS.....	3
APPENDIX A - GROUP LOG	4

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.1	Aaditi Anil Agrawal Arshit Chayan Kumawat Dobariya Jenil Bharatbhai Harsh Agrawal Harshit Harshit Srivastava Naman Kumar Jaiswal Prem Kansagra Priyanshu Singh	The First Draft of ConnVerse Implementation.	18/03/2024

1 Implementation Details

ConnVerse Implements a Model View Controller Architecture for integrating the backend and frontend of the Application. 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 generate a user session for a successful login. It also handles the cases of register/ alum Register differently. We are also using Crypto JS for the task of Authentication Encryption. We are using an Authentication algorithm in Go for this role.
 - b. **User Interaction Handler:** This proactively detects any interaction of the User with the frontend and generate 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 and Database Tier**

- 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, blog, chat, or profile.
- b. **Database:** This is key to our Software. The Database stores current users' data, their profile data, their blogs/comments, and even personalized messages. Most API call that is made to the server, involves 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. **Feed Generator:** This ranks the feeds in the order of relevance to the user's profile and the interests he has selected as tags.

Code Implementation Sub-Codebases:

1. Frontend:

This directory contains all the front-end components that are used in developing the user interface of the application. We have primarily used **HTML**, **CSS**, and **JavaScript** for UI design. While HTML is the code that determines what the website will contain, CSS controls how it will look. JavaScript and its frameworks make the website interactive and responsive to a user's actions. HTML provides the structure while CSS makes the web page aesthetic and JavaScript makes it lively.

The front end is developed using **ReactJS** (a JavaScript framework used to make the content on a page dynamic). Why ReactJS was 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 while development phase of the software.

Some of the React UI components are picked from **MUI (Material UI)** and **ChakraUI** - popular open-source React component library that provides a set of advanced accessible and customizable UI components.

This helps in saving time in developing some of the react components from scratch and decrease the time to market so that we can start the Customer feedback loop and start working on what the customer wants.

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 **Go. Gin**, a popular web framework in Go, is used to provide a route-based approach to defining HTTP endpoints and handling requests and responses.

Why Go and Gin were chosen for writing a large chunk of Backend over staple meta-picks like Java or Python?

- a. **Concurrency:** Go was designed with concurrency in mind, making it easy to write efficient, concurrent programs. Goroutines (lightweight threads) and channels provide powerful abstractions for concurrent programming, allowing thousands of concurrent connections and tasks.
- b. **Speedy Performance and Memory Management:** Go compiles to native machine code, resulting in fast execution speed and low resource consumption. Go has an automatic memory management through garbage collection. This reduces the risk of memory leaks and segmentation faults common in languages with manual memory management, making backend code more robust and reliable. Go's runtime environment has a relatively small memory footprint compared to some other languages and runtimes.
- c. **Efficient Garbage Collection:** Go's garbage collector (GC) is optimized for low-latency and concurrent operation. It uses a concurrent, tri-color mark and sweep algorithm, which minimizes pauses and allows the program to continue running smoothly even during garbage collection cycles. This means that backend services built in Go experience fewer interruptions due to GC pauses, ensuring consistent performance even under heavy load.
- d. **Cross-platform Support:** Backed by Google, Go supports compilation to various platforms and architectures, making it easy to write backend services that can run on different operating systems and environments without modification.

3. Backend (chat_system):

This directory contains all the necessary implementation of the chat system. This is written primarily in JavaScript on **NodeJS** runtime environment. This is a Micro Service Architecture we are using to reduce the load on the primary backend. **ExpressJS** is also as a side-by-side NodeJS as a fast, minimalist web framework which provides robust set of features for Server Management while remaining lightweight and flexible.

Why did we choose NodeJS for Chat Implementation and not go or any other language?

- a. NodeJS has excellent ease of development and better community support. NodeJS runtime is event-driven and handles asynchronous non-blocking I/O operations extremely well. The chat system's requirements are well met by NodeJS runtime.
- b. Also deploying NodeJS web services are very easy, and they integrate seamlessly with the existing Model as a microservice architecture.

Database Management:

1. **MongoDb:** We are using MongoDB as the primary centralised database for storing all the data regarding users, their profile data, 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 successful, 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 Go and JavaScript etc. This Enables developers to integrate MongoDB into their applications regardless of the technology stack.
2. **MongoDB Go Driver:** We are using MongoDB Go Driver to establish connection between Go runtime and MongoDB database. This is the official Go Driver, which is not only standardised but also secure, safe, and always kept up to standards by a Google Team. Not using the best as well as the most trustworthy driver for initialising database connection will be foolish.
3. **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.

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:

We have a single View connected to two servers, one being the main backend and the other being the micro service. Both servers are connected to the same database. The controller will decide which server to send the API request to and both the servers are free to fetch queries from the database.

Why is such a system better?

1. **Language Specialization:** Different languages have different strengths and weaknesses. Go has excellent capability for performing concurrent tasks, outstanding garbage collection, speedy execution and memory management. NodeJS has excellent ease of development and better community support. NodeJS runtime is event-driven and handles non-blocking I/O operations extremely well.
2. **Micro Performance Optimization:** Using both the languages for the tasks they are excellent in handling helps the micro handling of the application very well. Go can handle API requests very well while NodeJS event-driven architecture suits the chat implementation very well.
3. **Microservices Architecture:** The chat is implemented as a Web Micro Service the main backend that was written in Go. Microservice architecture breaks down a monolithic application into smaller, more manageable services that can be developed, deployed, and scaled independently.
4. **Multiple Servers:** Use of two servers at the same time ensures division of labour as well as on an average makes sure that the weight on each server is uniformly distributed. This is multi front performance optimization.

Authentication and Authorization:

1. **CryptoJS:** We are also using **CryptoJS** (a JavaScript library that provides cryptographic algorithms and functions) as an encryptor. This helps us in User Authentication (frontend) as well as in Data Authentication (backend). Hashing and Encrypting is essential for securing data from cases of data leaks. The user security is one of the prime concerns for a Connect and Converse App like ConnVerse.
2. **EmailJS:** While registering new users we are also inculcating email verification which is achieved by using EmailJS (a JavaScript library that simplifies the process of sending emails from client-side JavaScript code). EmailJS is the free, simple, easy to setup and has high cross-platform compatibility and the undoubted best option for simple emailing protocols.
3. **JWS (JSON Web Signature):** JWS is a compact, URL-safe means of representing signed content using JSON data structure. It is mandatory to digitally sign JSON data to ensure its integrity and authenticity whenever the data is returned after being processed from diverse sources. JWS is widely adopted and standardised method of digital signature. Given its compactness, ease-of-use, and security, deterring from it makes no sense.

API endpoints:

The screenshot displays two API requests in Postman, both using the 'raw' body type and 'JSON' format.

Request 1: SignUp

- Method: POST
- URL: `http://localhost:8080/signup`
- Body (JSON):

```
{  "email": "premk22@iitk.ac.in",  "old_password": "Prem1234",  "new_password": "Prem1234"}
```

Request 2: Send-OTP

- Method: POST
- URL: `http://localhost:8080/send-otp`
- Body (JSON):

```
{  "email": "premk22@iitk.ac.in"}
```

Response (for Send-OTP):

- Status: 200 OK
- Time: 7.42 s
- Size: 158 B
- Body (JSON):

```
{  "message": "OTP sent successfully"}
```


[illegible]

The screenshot displays a REST client interface with two panels. The top panel shows a POST request to `http://localhost:8080/forgot-password` with a JSON body containing email, old password, and new password. The response is a 401 Unauthorized status with the message "Invalid password". The bottom panel shows a POST request to `http://localhost:8080/blog/compose/new` with a JSON body containing blog details. The response is a 201 Created status with a JSON body containing the created blog details.

Request 1: Forgot Password

Method: POST
URL: `http://localhost:8080/forgot-password`

Body (JSON):

```
{
  "email": "premk22@iitk.ac.in",
  "old_password": "Prem1234",
  "new_password": "Prem1234"
}
```

Response:

Status: 401 Unauthorized
Body (JSON):

```
{
  "error": "Invalid password"
}
```

Request 2: Blog Creation

Method: POST
URL: `http://localhost:8080/blog/compose/new`

Body (JSON):

```
{
  "title": "Blog1",
  "content": "First Blog",
  "image": "",
  "tags": ["AI/ML"],
  "authorid": "220816",
  "authorimage": "",
  "authorname": "Prem Kansagra"
}
```

Response:

Status: 201 Created
Body (JSON):

```
{
  "blog": {
    "ID": "65f8844b949ff9c594969f51",
    "title": "Blog1",
    "content": "First Blog",
    "image": "",
    "authorid": "220816",
    "authorname": "Prem Kansagra",
    "authorimage": "",
    "likes": 0,
    "dislikes": 0,
    "comments": []
  }
}
```

The screenshot displays a REST client interface with two tabs. The top tab, titled 'Blog Search', shows a POST request to `http://localhost:8080/blog/search/`. The request body is in raw format and contains a JSON object: `{ "title": "Celebrating" }`. The bottom tab, titled 'Blog Delete', shows a POST request to `http://localhost:8080/blog/compose/delete/65f8844b949ff9c594969f51`. The request body is in raw format and is empty. Both tabs show the 'Body' tab selected, and the response status is 200 OK for the first request and 401 Unauthorized for the second.

Blog Search

POST `http://localhost:8080/blog/search/` Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

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

```
1 {
2   "title": "Celebrating"
3 }
```

Body Cookies (1) Headers (3) Test Results 200 OK 3.69 s 237.93 KB Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "posts": [
3     {
4       "ID": "65f33c65c2ce3b0df77a68c5",
5       "title": "Celebrating Milestones: The Journey of ConnVerse App Development",
6       "content": "In the realm of technological innovation, few experiences rival the exhilaration of seeing months, even years, of dedication culminate in the birth of a groundbreaking application. Today, as I reflect upon our journey with the MahaDevs app, I find myself overwhelmed with joy and gratitude. Our team's tireless efforts, collective vision, and unwavering commitment have finally borne fruit, igniting a spark of pride that radiates through every member of our team.\n\nThe inception of MahaDevs was not merely a brainstorming session; it was the convergence of passion, expertise, and a burning desire to make a
```

Blog Delete

POST `http://localhost:8080/blog/compose/delete/65f8844b949ff9c594969f51` Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

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

```
1
```

Body Cookies (1) Headers (2) Test Results 401 Unauthorized 1104 ms 85 B Save as example

Pretty Raw Preview Visualize Text

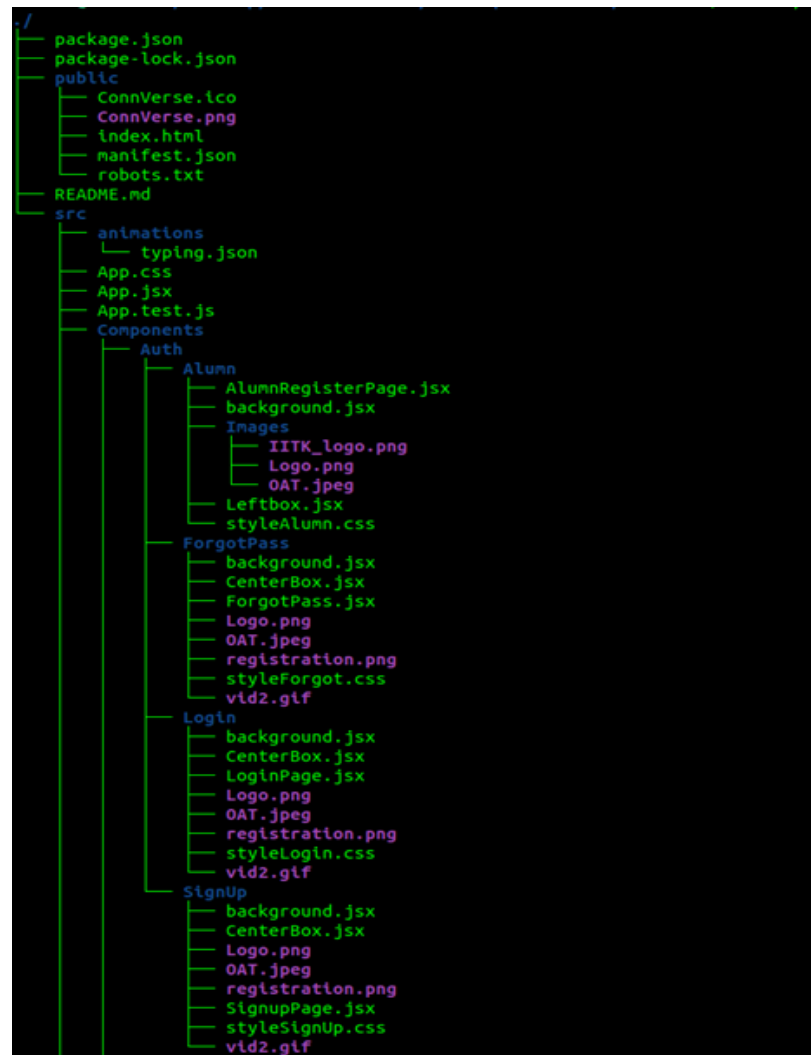
```
1
```

2 Codebase

GitHub link of code implementation: <https://github.com/Naman-K-Jaiswal/ConnVerse.git>

We have built a single repository which contains both the backends as well as the react front-end. We have segregated the Code Implementation from the Documentation into different directories and divided the Code Implementation into three portions – frontend, backend, and chat system.

1. Frontend:



```
├── Authentication
│   ├── Login.js
│   └── Signup.js
├── Blog
│   ├── CreateBlog
│   │   ├── CreateBlogPage.jsx
│   │   └── style.module.css
│   └── IndividualBlog
│       ├── blog_img.jpeg
│       ├── BlogTemplate.jsx
│       └── style.css
├── Chatbox.js
├── ChatLoading.js
├── Contact
│   ├── Contact.jsx
│   ├── Contact.module.css
│   └── Logo.png
├── Footer
│   ├── Footer.jsx
│   └── style.css
├── Home
│   ├── Home.jsx
│   └── style.css
├── LandingPage
│   ├── background.png
│   ├── LandingPage.jsx
│   ├── LandingPage.module.css
│   ├── Logo.png
│   ├── Navbar.jsx
│   ├── Navbar.module.css
│   ├── Particles.jsx
│   ├── Typewriter.css
│   └── Typewriter.jsx
├── Loader
│   ├── Loader1.jsx
│   ├── Loader.css
│   └── Loader.jsx
├── Members
│   ├── filter.png
│   ├── image.png
│   ├── Members.jsx
│   ├── Members.module.css
│   └── pp.jpeg
├── miscellaneous
│   ├── GroupChatModal.js
│   ├── ProfileModal.js
│   ├── SideDrawer.js
│   └── UpdateGroupChatModal.js
├── MyChats.js
├── Navbar
│   ├── Navbar.jsx
│   ├── profile_pic.JPG
│   └── style.css
├── ScrollableChat.js
├── SingleChat.js
├── styles.css
├── User
│   ├── backgroundImage.jpg
│   ├── dummy.jsx
│   ├── UserProfile.css
│   ├── userProfileImage.JPG
│   └── UserProfile.jsx
├── userAvatar
│   ├── UserBadgeItem.js
│   └── UserListItem.js
├── config
│   └── ChatLogics.js
├── Context
│   └── ChatProvider.js
├── data
│   └── messages.js
├── index.css
├── index.jsx
├── logo.svg
├── Metadata.jsx
├── Pages
│   ├── Chatpage.js
│   ├── Homepage.js
│   └── reportWebVitals.js
├── setupTests.js
```

We have 105 files over 28 directories:

- package.json** - Contains all the necessary dependencies that need to be installed before running the application.
- README.md** - Contains step by step instructions on how to boot the server.
- public/manifest.json** - Contains human readable meta-data about the Project.
- public/index.html** - This is the primary web page that gets loaded onto the view and react components are loaded onto it.
- public** - Contains all the static assets that need to be present on the View.
- src/components** - Contains all the react components that are used while the app is in use.

2. Backend:

```
./
├── chat_system
│   ├── config
│   │   ├── db.js
│   │   └── generateToken.js
│   ├── controllers
│   │   ├── chatControllers.js
│   │   ├── messageControllers.js
│   │   └── userControllers.js
│   ├── middleware
│   │   ├── authMiddleware.js
│   │   └── errorMiddleware.js
│   ├── models
│   │   ├── chatModel.js
│   │   ├── messageModel.js
│   │   └── userModel.js
│   ├── package.json
│   ├── package-lock.json
│   ├── routes
│   │   ├── chatRoutes.js
│   │   ├── messageRoutes.js
│   │   └── userRoutes.js
│   └── server.js
├── main-backend
│   ├── authn
│   │   ├── auth_classes.go
│   │   ├── forgot_password.go
│   │   ├── helper_functions.go
│   │   ├── login.go
│   │   ├── logout.go
│   │   └── signup.go
│   ├── blog
│   │   ├── blog_classes.go
│   │   ├── blog_search.go
│   │   ├── comment.go
│   │   ├── create_post.go
│   │   ├── delete_post.go
│   │   ├── edit_post.go
│   │   ├── helper_functions.go
│   │   ├── react.go
│   │   ├── retrieve_blog.go
│   │   └── retrieve_post.go
│   ├── database
│   │   └── db.go
│   ├── feed
│   │   ├── add_tags.go
│   │   ├── feed_classes.go
│   │   ├── helper_functions.go
│   │   ├── reload_feed.go
│   │   └── send_feed.go
│   ├── go.mod
│   ├── go.sum
│   ├── mail
│   │   └── mail_smtp.go
│   ├── main.go
│   ├── middleware
│   │   ├── crud_auth.go
│   │   └── require_auth.go
│   ├── profile
│   │   ├── create_user.go
│   │   ├── helper_functions.go
│   │   ├── profile_classes.go
│   │   ├── profile_search.go
│   │   ├── retrieve_user.go
│   │   └── update_user.go
│   └── routes
│       ├── auth_routes.go
│       ├── blog_routes.go
│       ├── feed_routes.go
│       └── profile_routes.go
```

We have 54 files over 15 directories:

- a. **main-backend/go.sum** - Contains all the expected cryptographic checksums of the content of specific module versions used in the project. It ensures that the downloaded dependencies match the expected checksums to prevent tampering or unexpected changes.
- b. **main-backend/go.mod** - Defines the module path and contains all the module's dependencies along with their versions.
- c. **main-backend/main.go** - The main package's entry point file in the application. The essential file for the server to boot in.
- d. **main-backend/build.sh** - Contains the script for building the backend of the application.
- e. **main-backend/authn** - Contains all functionalities of authentication a login attempt or registering a new user (or alum). Defines all the environment variables.
- f. **main-backend/database** - Initialise the connection with the database.
- g. **main-backend/mail** - Implementing mailing for the purpose of registration.
- h. **main-backend/routes** - Contains the route handlers and routing logic for different endpoints of the application.
- i. **main-backend/profile** - Contains the functionalities that any user will be able to enjoy after logging in.
- j. **main-backend/blog** - Contains the functionalities that enables any user to write new blogs, edit old one, delete old ones.
- k. **main-backend/feed** - Contains the feed generator for the user which ranks the blogs user might be interested in, in terms of relevance.
- l. **main-backend/routes** - Contains the route handlers and routing logic for different endpoints of the application.
- m. **chat_system/package.json** - Contains the human readable metadata about the Chat Micro Service that is written in NodeJS. It also tracks the dependencies that is required to deploy the micro service.
- n. **chat_system/routes** - Contains modules or files that define the routes and route handlers for various parts of the Micro Service.
- o. **chat_system/config** - Contains configuration files (scripts) that establish connection with the database.
- p. **chat_system/server.js** - Serves as the entry point for the server. It is responsible for creating and configuring the Chat application, setting up middleware, defining routes, and starting the server to listen for incoming HTTP requests.

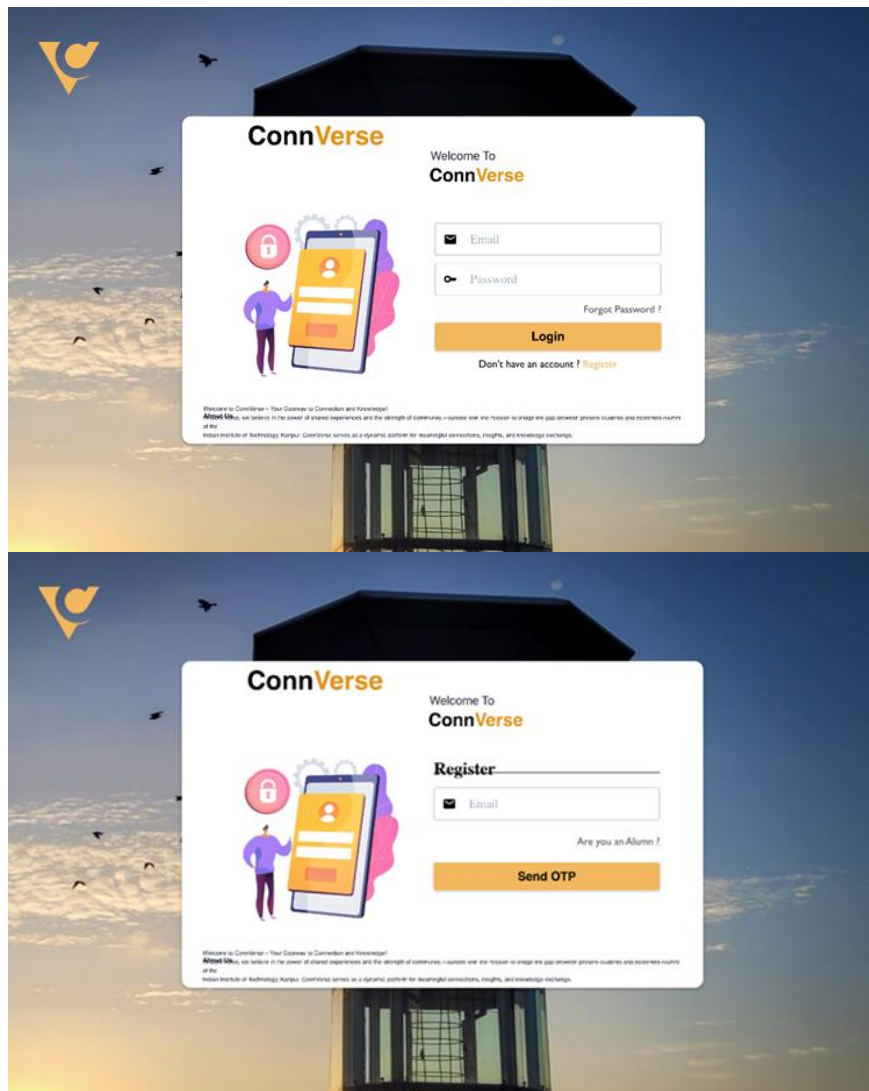
3 Completeness

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:

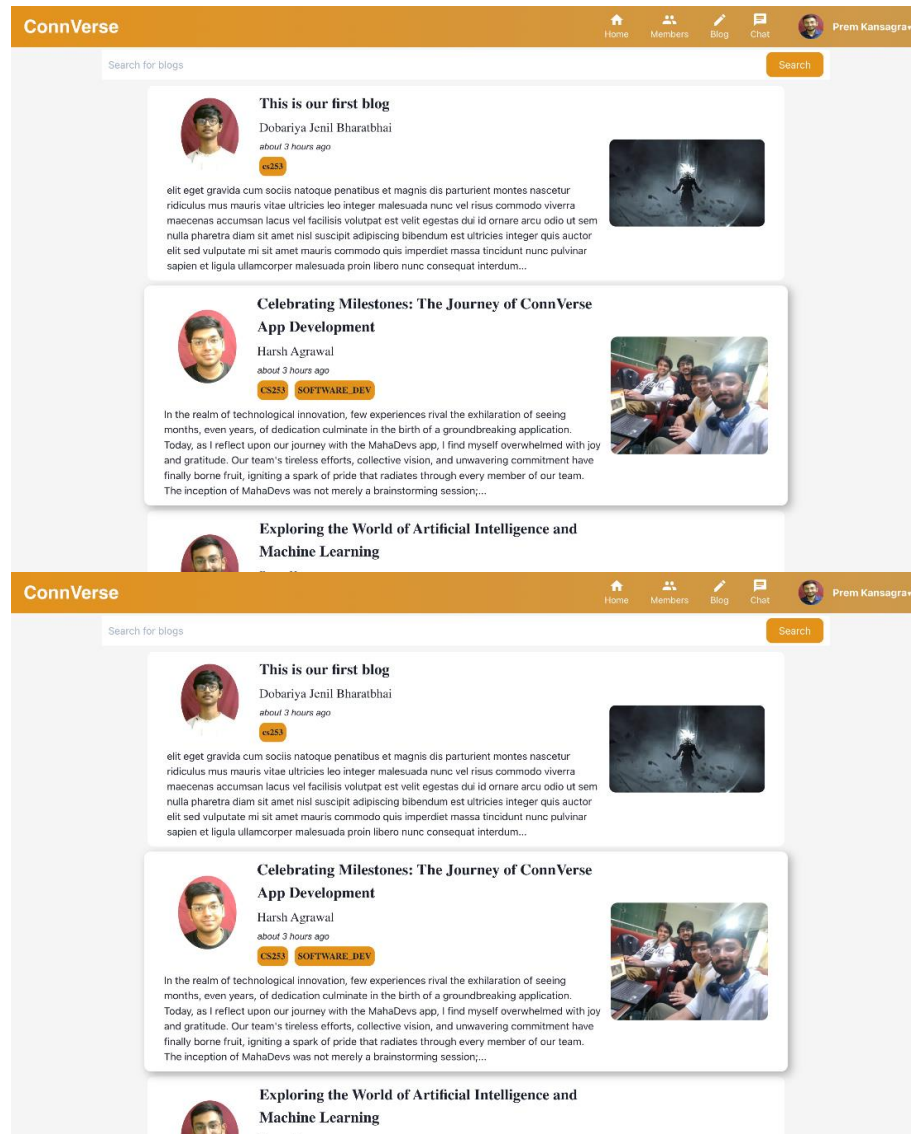
1. User Interfaces:

- Login Page and Sign-Up Page:



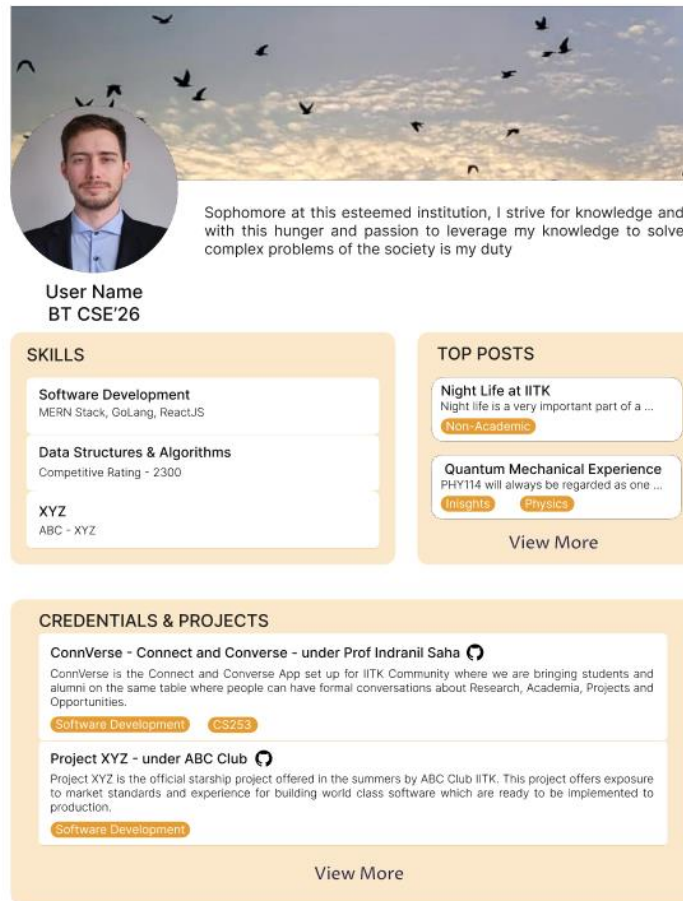
This is the login / Sign Up page where the user will land after the landing page and enter their credentials

- Homepage:



This is the page which will be the default when the user is already signed in with a loaded session data.

- **User Profile:**



The user profile card features a circular profile picture of a man with a beard and a blue shirt, set against a background of a sky with many birds flying. Below the picture is a bio: "Sophomore at this esteemed institution, I strive for knowledge and with this hunger and passion to leverage my knowledge to solve complex problems of the society is my duty". The user's name and year are listed as "User Name" and "BT CSE'26".

SKILLS



- Software Development**
MERN Stack, GoLang, ReactJS
- Data Structures & Algorithms**
Competitive Rating - 2300
- XYZ**
ABC - XYZ

TOP POSTS

- Night Life at IITK**
Night life is a very important part of a ...
Non-Academic
- Quantum Mechanical Experience**
PHY114 will always be regarded as one ...
Insights **Physics**

[View More](#)

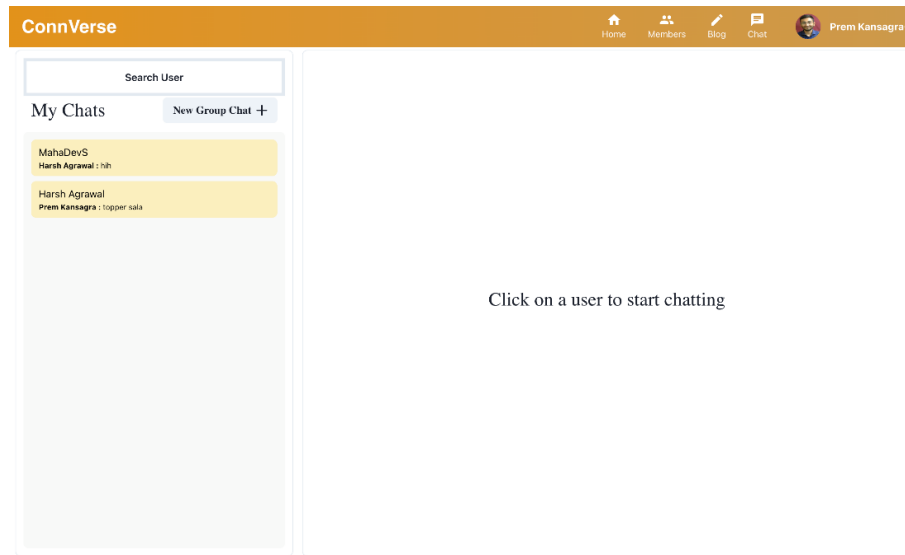
CREDENTIALS & PROJECTS

- ConnVerse - Connect and Converse - under Prof Indranil Saha** 
ConnVerse is the Connect and Converse App set up for IITK Community where we are bringing students and alumni on the same table where people can have formal conversations about Research, Academia, Projects and Opportunities.
Software Development **CS253**
- Project XYZ - under ABC Club** 
Project XYZ is the official starship project offered in the summers by ABC Club IITK. This project offers exposure to market standards and experience for building world class software which are ready to be implemented to production.
Software Development

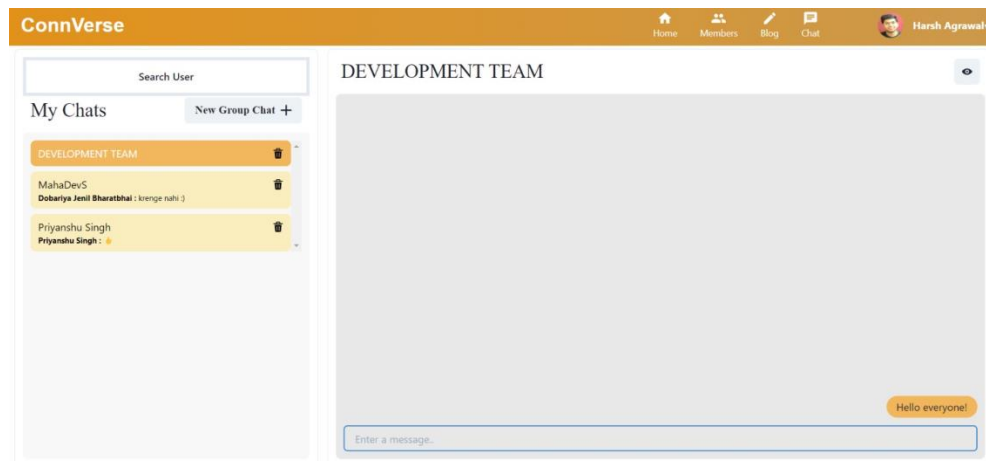
[View More](#)

This is the page which will showcase your User Profile which will be visible to any other user. On a profile page you can browse the number of blogs published by user, contribution metric, a detailed list of user's skills, achievements, credentials, achievements, completed coursework and projects undertaken. There is a search bar at the top which enables you to search other users.

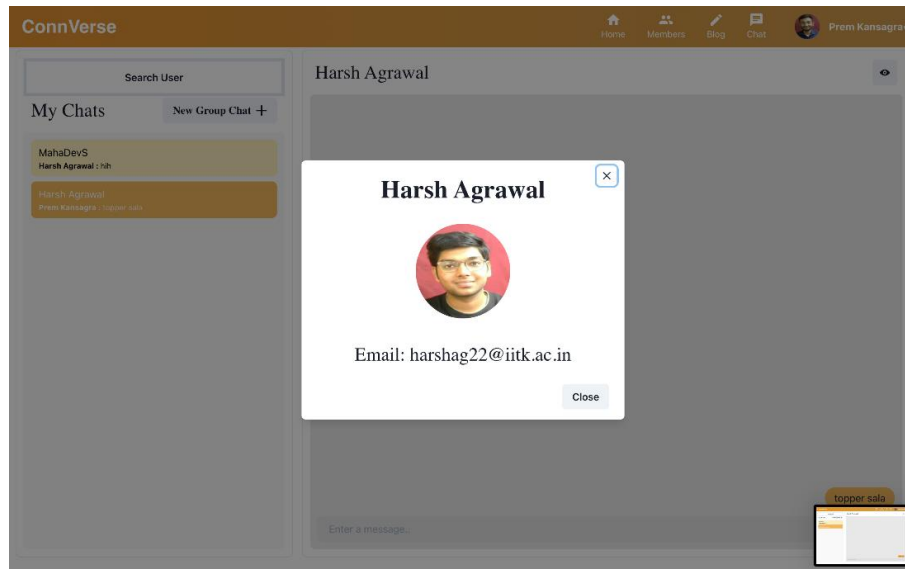
- **Chat:**



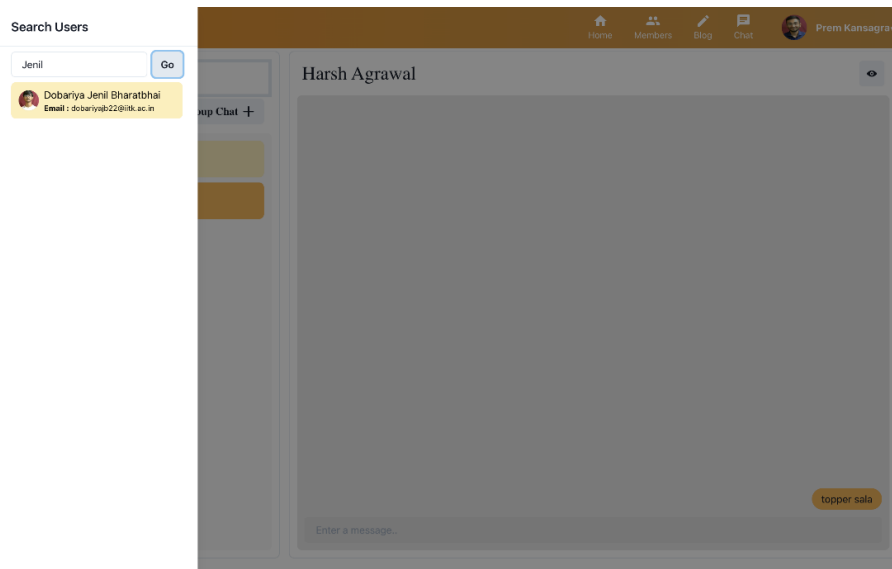
This is the chat page when you open messaging. You can select any user and start conversation with them.



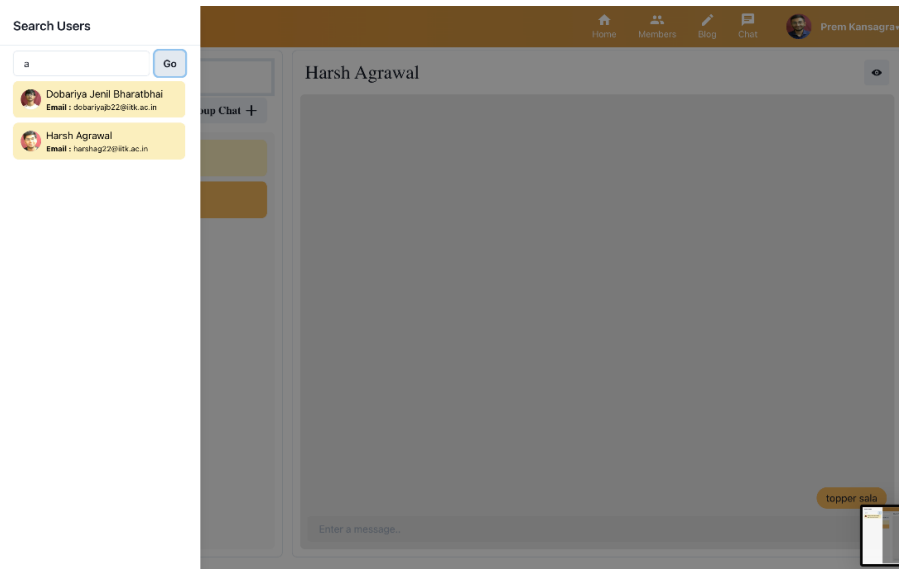
This is a specific chat opened.



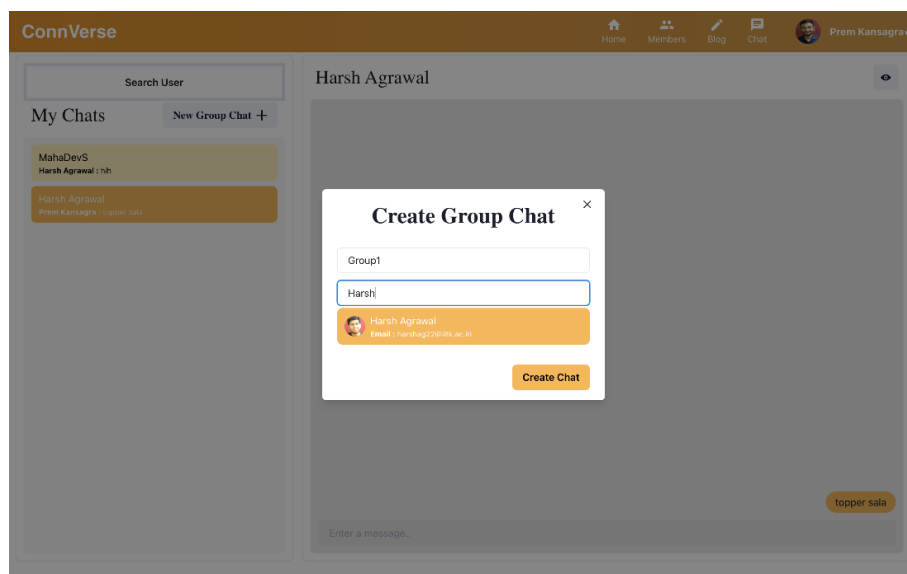
User profile inside a chat.



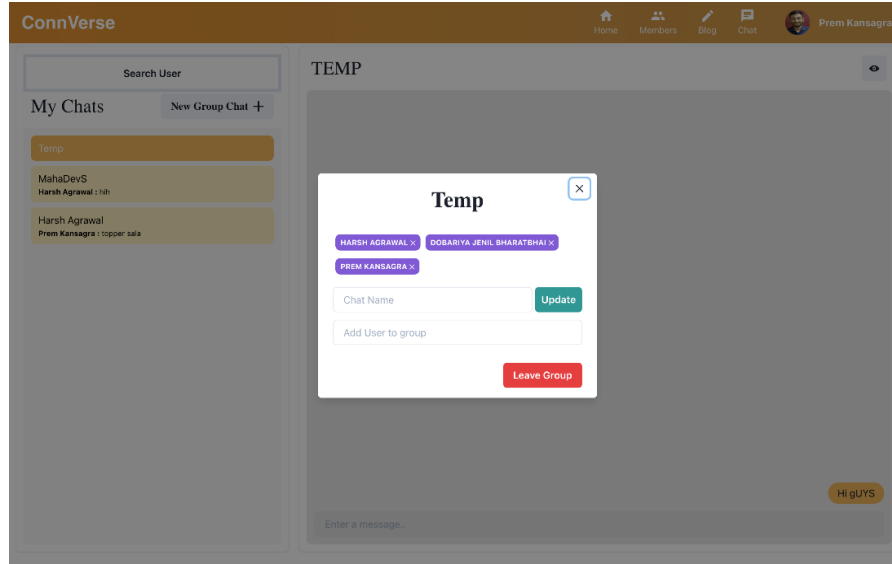
User Search by Name



This is a user search with the help of which you can start new chats.



The users can also enjoy the feature of group chat.



- **Blog:**



This is a blog written by a user posted on his profile.

Functional Requirements:

All of these have been implemented in CodeImplementation/main-backend and CodeImplementation/chat_system:

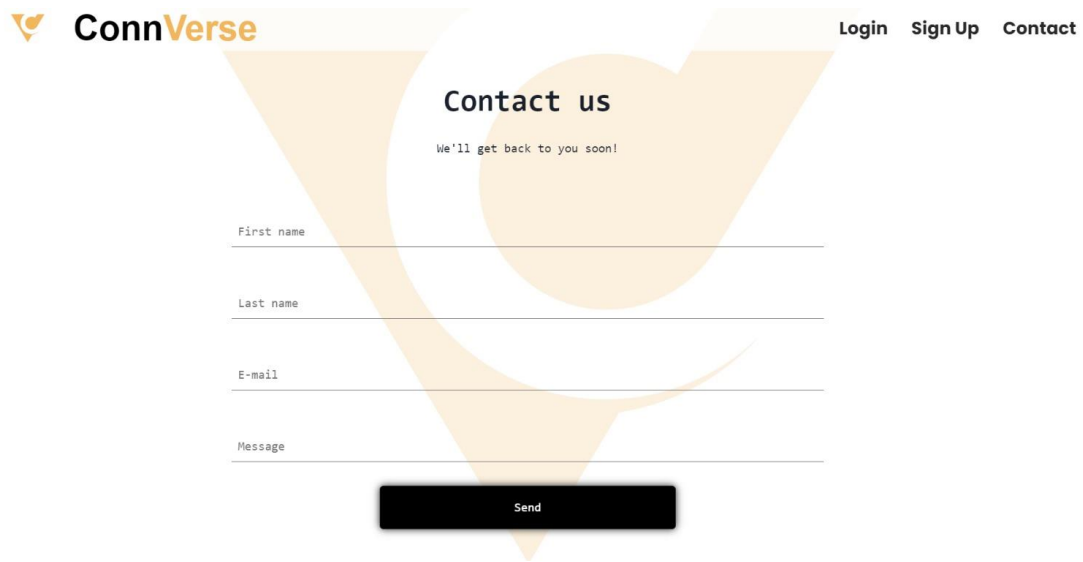
- User Registration

The image displays two screenshots of the ConnVerse user registration process. The top screenshot shows an email titled "ConnVerse One Time Password" with a dark background. It includes a user icon, the email address "connverse22@gmail.com", the date "Today 10:50 pm", and links for "Summary" and "Headers". The message body says "Hello," followed by "This is your OTP for ConnVerse : ExTD5m (do not share with anyone)".

The bottom screenshot shows the ConnVerse registration form. The form has a header with the ConnVerse logo and a "Welcome To ConnVerse" message. Below this is a "Register" section with an email input field containing "namankj22@iitk.ac.in" and a "Send OTP" button. There is also a checkbox labeled "Are you an Alumni?". The form also includes an "About Us" section with a brief description of the platform.

The bottom screenshot shows the ConnVerse registration form. The form has a header with the ConnVerse logo and a "Welcome To ConnVerse" message. Below this is a "Register" section with an email input field containing "ExTD5m", a password input field, and a "SIGN UP" button. There is also a checkbox labeled "Are you an Alumni?". The form also includes an "About Us" section with a brief description of the platform.

- Manual Registration for Alumni



The image shows a web form titled "Contact us" on the ConnVerse website. The form is set against a background with a large, faint "V" shape. It includes input fields for "First name", "Last name", "E-mail", and "Message". A "Send" button is located at the bottom. Navigation links for "Login", "Sign Up", and "Contact" are in the top right corner. A message "We'll get back to you soon!" is displayed below the title.

ConnVerse Login Sign Up Contact

Contact us

We'll get back to you soon!

First name

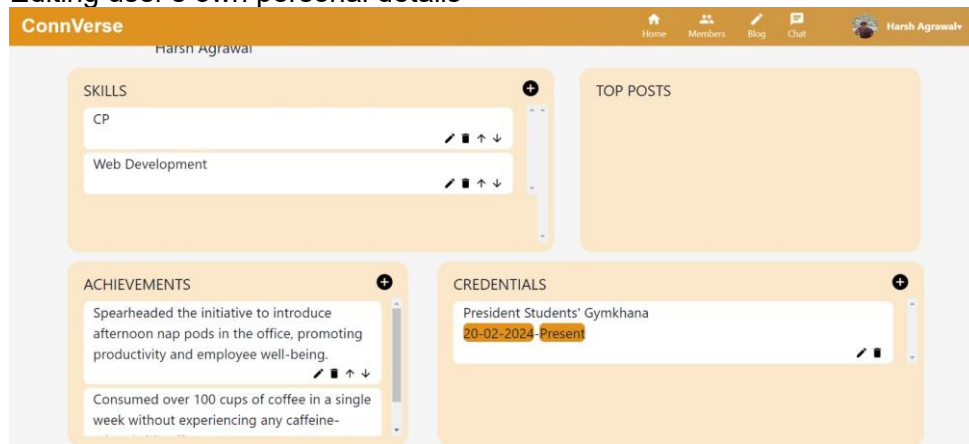
Last name

E-mail

Message

Send

- Editing user's own personal details



The image shows a user profile page for "Harsh Agrawal" on the ConnVerse website. The page has a navigation bar with "Home", "Members", "Blog", "Chat", and a user profile icon. The profile section includes "SKILLS" (CP, Web Development), "TOP POSTS", "ACHIEVEMENTS" (Spearheaded the initiative to introduce afternoon nap pods in the office, promoting productivity and employee well-being. Consumed over 100 cups of coffee in a single week without experiencing any caffeine-), and "CREDENTIALS" (President Students' Gymkhana, 20-02-2024-Present).

ConnVerse Harsh Agrawal

Home Members Blog Chat Harsh Agrawal

SKILLS

CP

Web Development

TOP POSTS

ACHIEVEMENTS

Spearheaded the initiative to introduce afternoon nap pods in the office, promoting productivity and employee well-being.

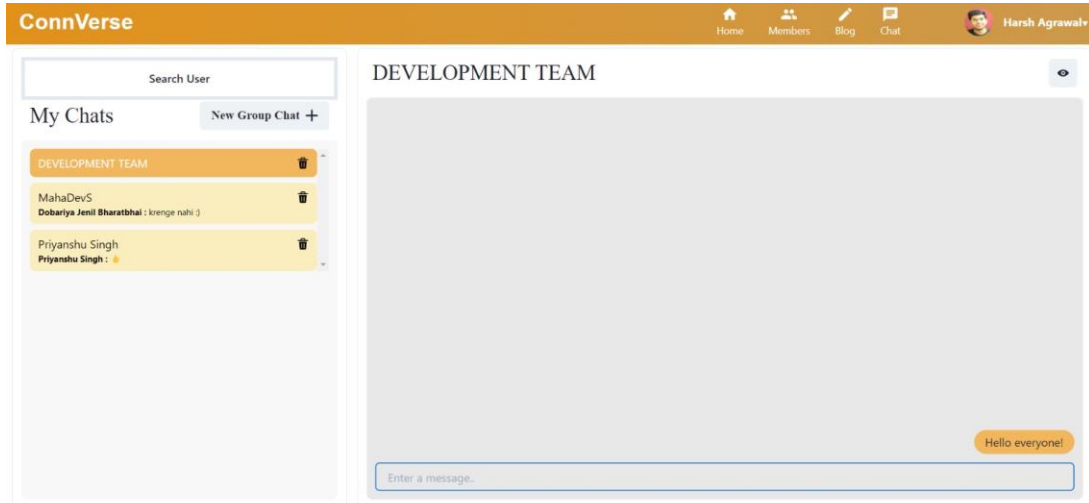
Consumed over 100 cups of coffee in a single week without experiencing any caffeine-

CREDENTIALS

President Students' Gymkhana

20-02-2024-Present

- Chat with other users



The image shows a chat interface on the ConnVerse website. The left sidebar shows "My Chats" with a search bar and a list of chat groups: "DEVELOPMENT TEAM", "MahaDevS", and "Priyanshu Singh". The main chat area is titled "DEVELOPMENT TEAM" and shows a message "Hello everyone!" and a text input field "Enter a message...".

ConnVerse Harsh Agrawal

Home Members Blog Chat Harsh Agrawal

Search User

My Chats

New Group Chat +

DEVELOPMENT TEAM

MahaDevS

Dobariya Jenil Bharatbhai : krenge nahi :)

Priyanshu Singh

Priyanshu Singh :

DEVELOPMENT TEAM

Hello everyone!

Enter a message...

- Composing Blogs

The screenshot shows the 'ConnVerse' web application interface for composing a new blog post. The header includes navigation links: Home, Members, Blog, Chat, and a user profile for Harsh Agrawal. The form fields are as follows:

- TITLE:** The Impact of Social Media on Interpersonal Relationships
- TAGS:** THOUGHTS, LIFE, IITK
- IMAGES:** Choose File, Take What Comes - Facebook Covers - 1694.jpeg
- Content:** A text area containing placeholder Lorem Ipsum text.

- Comment, Upvote or Downvote in Blogs

The screenshot displays a blog post titled 'Celebrating Milestones: The Journey of ConnVerse App Development' by Harsh Agrawal. The post includes a profile picture, a heart icon with '201', and 'Comments: 15'. The main text describes the journey of developing the app. On the right, a 'Comments' section shows a list of comments, all from 'Harsh Agrawal', with a 'Submit' button at the bottom.

- Searching Users

The screenshot shows the 'ConnVerse' user search results page. The header includes navigation links: Home, Members, Blog, Chat, and a user profile for Naman Kumar Jain. A search bar at the top contains the text 'harsh'. Below the search bar, two user profiles are displayed:

- User 1:** Harsh Agrawal, BTech Computer Science and Engineering, Sophomore @IITK, Web Development.
- User 2:** Harshit Srivastava, BTech Computer Science and Engineering, New Skill, C++.

Future Development:

- **Feed Generator:** Currently there is a fixed algorithm which ranks the blogs in terms of relevance to the user. We can improve upon it and build a ML model which does the ranking which also trains itself to give better results the longer is it run.
- **Edit Blogs:** The functionality of editing blogs is still not implemented. We can improve upon our code and open this option to the users in the next versions.

Appendix A - Group Log

Team A	Jenil, Harshit Srivastava, Harsh, Priyanshu, Aaditi	Feed Development and Profile
Team B	Chayan, Harshit Chikara, Arshit, Naman, Prem	Chat System

Date	Team	Summary of the Meeting	Duration	Location
10th Feb' 24	All	Divided the team into two halves for feed and chat system	45 min	KD Library
11th Feb' 24	All	Allocated frontend and backend work to each member of the group.	60 min	RM building
13 th Feb' 24	A	Discussion regarding the workflow of feed development	90 min	RM building
15 th Feb'24	B	Discussion regarding the workflow of chat system	75 min	CCD
25th Feb' 24	All	Updates regarding the work progress from both teams	30 min	Online
29th Feb' 24	A	Clarification regarding the external features of feed development, such as the blog page and homepage.	75 min	KD Library
3rd Mar' 24	B	Discussion regarding backend development of chat system	60 min	RM building
7 th Mar' 24	All	Updates regarding the work progress from both teams	45 min	RM building
10 th Mar' 24	A	Discussion regarding the changes in some functionalities of the feed development	90 min	KD Library

12 th Mar' 24	B	Clarification regarding some changes in the functionalities	120 min	RM building
15 th Mar' 24	All	Integration of frontend and backend	240 min	RM building
16 th Mar' 24	All	Clearing bugs and fixing the problems in the system	420 min	RM building
18 th Mar' 24	All	Implemented the last adjustments to the documentation.	180 min	RM building