# Chapter 1: Introduction

## 1.1 Background

As technology advances and social media platforms become a dominant space for communication and interaction, it has become important to understand how to build and manage such platforms. My project, GabGrid, aims to create a social media platform using the MERN stack (MongoDB, Express.js, React.js, and Node.js). I am motivated by the need to develop a scalable, secure, and user-friendly system that integrates real-time interaction and community-building features. By working on this project, I seek to address the challenges of modern social media platforms while gaining hands-on experience with full-stack development.

## 1.2 Objectives

The objectives of my project are as follows:

1. To create a fully functional social media platform using the MERN stack.
2. To implement real-time features such as instant updates, notifications, and live interactions.
3. To ensure the platform is scalable, allowing it to handle multiple users and large amounts of data efficient
4. To provide secure user authentication and data privacy using industry-standard techniques.
5. To design a responsive and intuitive user interface that enhances the overall user experience across various devices.

### 1.2.1 Purpose

The purpose of this project is to create a platform that meets the growing demands for instant communication and secure social media environments. It aims to facilitate interaction among users while ensuring robust data protection and scalability, which are often challenges in existing platforms.

### 1.2.2 Scope

The scope of my project includes:

1. Frontend Development: I will use React.js to create a responsive and dynamic user interface.

2. Backend Development: The backend will be built with Node.js and Express.js to manage data and handle API requests.

3. Database: MongoDB will serve as the database, storing user information, posts, comments, and other data
4. Security: I will implement user authentication using JWT (JSON Web Tokens) and bcrypt for password encryption to ensure data security.
5. Real-Time Features: Real-time updates and notifications will be implemented using WebSockets (Socket.io) to enhance user engagement.

# Chapter 2: Survey of Technologies

## Implementation

**1. Tech Stack**

**GabGrid is built using the MERN stack (MongoDB, Express.js, React.js, Node.js) for a seamless full-stack experience.**

**2. User Authentication**

**Implemented JWT-based authentication for secure login, signup, and session management.**

**3. Tweet Functionality**

**Users can post tweets with text or images, view their timeline, and interact with content.**

**4. Social Features**

**Users can like tweets, follow/unfollow others, and engage within the platform.**

**5. Real-time Updates**

**Implemented WebSockets for instant notifications and live updates on tweets and interactions.**

**6. Database & Storage**

**Tweets, user profiles, and interactions are stored in MongoDB, while images are handled using Cloudinary/AWS S3.**

**7. Responsive UI**

**Designed a mobile-first UI using React.js & Tailwind CSS for an intuitive user experience.**

**8. Deployment**

**Deployed using Vercel for frontend and Render/DigitalOcean for backend, ensuring scalability.**

# Chapter 3: System Analysis

## 2.1 Existing System

Existing social media platforms like Twitter and Facebook have millions of users, and they need to handle high volumes of data and interactions. However, these platforms often face challenges related to scalability, security, and real-time data processing. I aim to address these challenges by building a system that focuses on scalability and real-time interaction.

## 2.2 Proposed System

The system I am proposing, GabGrid, is a social media platform that allows users to share posts, comment, and interact in real-time. Built using the MERN stack, the platform focuses on providing a seamless user experience across devices, with secure authentication and live updates. It will offer features such as:

Real-time post and comment updates. Secure login and

session management.

Scalable architecture to handle large user bases.

## 2.3 Requirement Analysis

For the platform to function effectively, the following requirements need to be addressed:

Functional Requirements:

User registration and login.

Creating and editing posts.

Real-time commenting and interaction.

Non-Functional Requirements:

Performance: The platform should be able to handle a large number of concurrent users without slowing down.

Security: User data must be kept secure, especially during authentication and storage.

Scalability: The system should be able to scale easily as the number of users increases.

## 2.4 Hardware Requirements

For developing and running GabGrid, I need the following hardware: A development

machine with the required software installed.

Hosting services (such as AWS or Heroku) for deploying the platform

## 2.5 Software Requirements

The software I will be using includes:

Frontend: React.js for building the user interface, with tools like CSS and Bootstrap/Material-UI for styling.

Backend: Node.js and Express.js for server-side development and handling API requests.

Database: MongoDB for managing data.

Security: JWT for user authentication and bcrypt for password encryption.

Real-Time Communication: Socket.io to enable live updates and notifications.

Testing Tools: Postman for API testing and Jest for unit testing.

## 2.6 Justification for Selection of Technology

I chose the MERN stack for this project because of its flexibility, scalability, and widespread use in modern web development:

MongoDB: A NoSQL database that allows for efficient management of unstructured data.

Express.js: Simplifies server-side development and routing.

React.js: Provides an excellent framework for building interactive user interfaces.

Node.js: Facilitates building a scalable, event-driven backend that can handle real-time communication.

Socket.io: Ideal for implementing real-time updates and notifications, which are essential for social media platforms.

# Chapter 4: System Design

## 3.1 Module Division

I divided the system into several modules to make development more manageable:

User Module: Handles user registration, authentication, and profile management.

Post Module: Manages the creation, editing, and deletion of user posts.

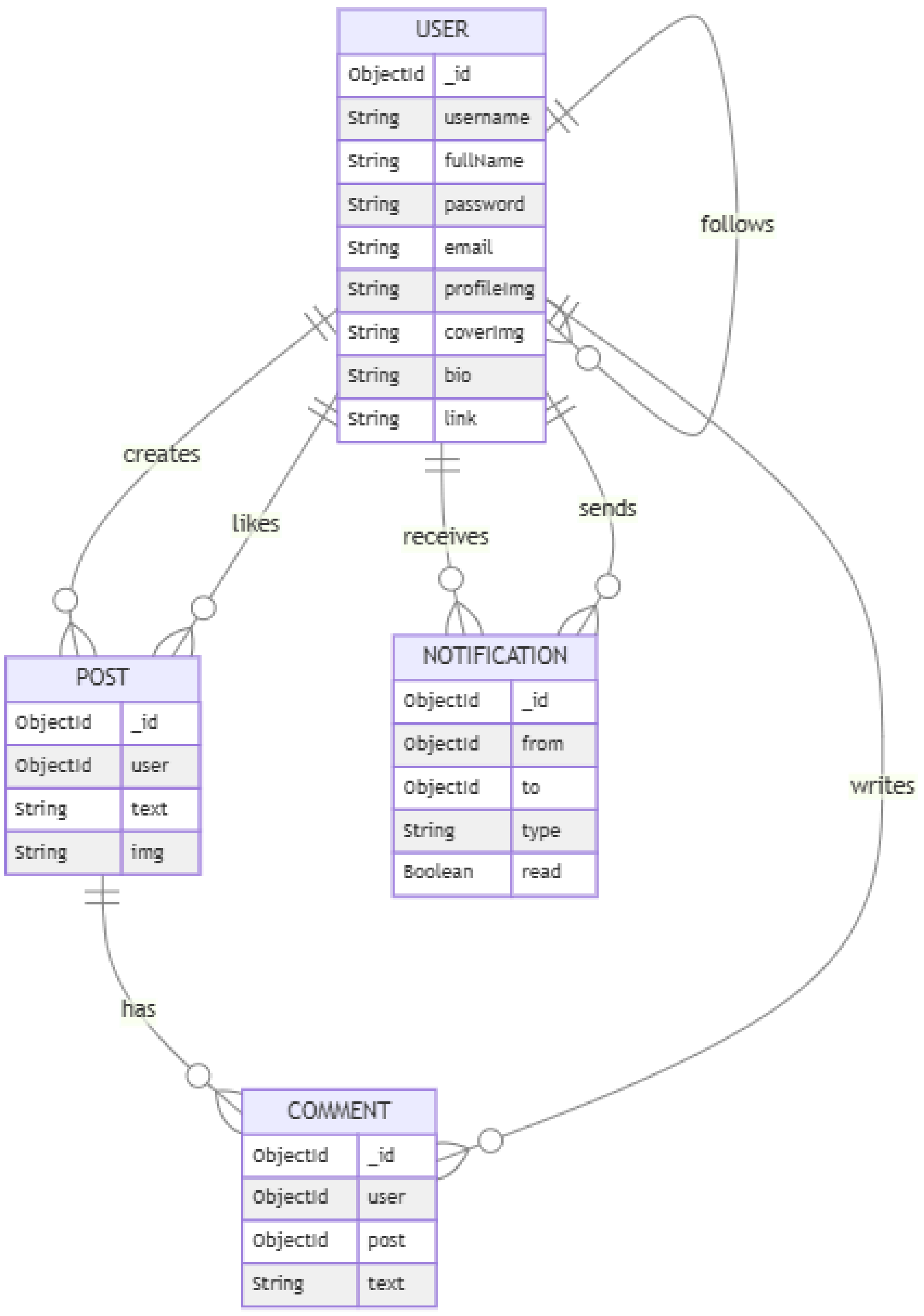
Comment Module: Handles comments on posts, including editing and deleting comments.

Real-Time Communication Module: Uses Socket.io to enable real-time notifications and updates for users.

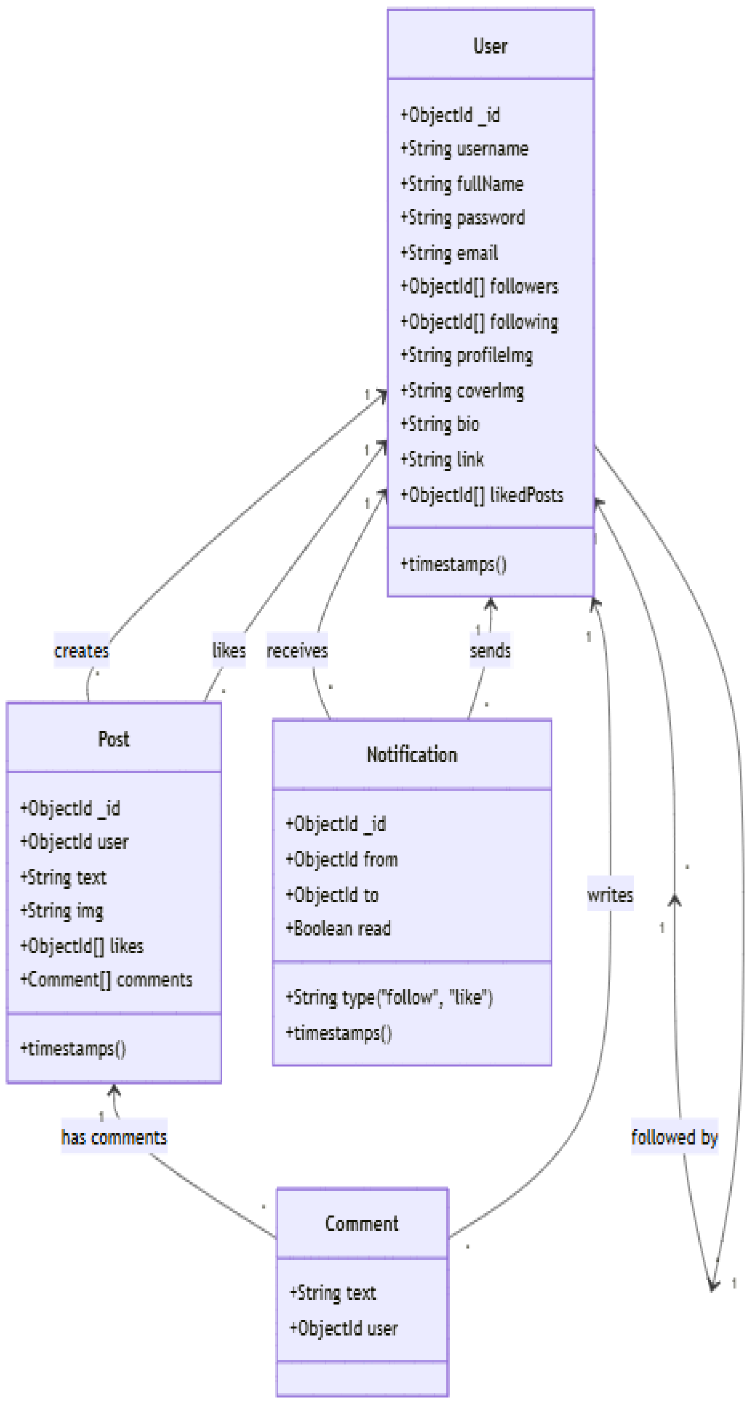
## 3.2 Data Dictionary

Here is the structure of some key data elements used in the system:

## 3.3 ER Diagrams



## 3.4 UML Diagrams



# Chapter 5: Implementation and Testing

## 4.1 Code (Core Segments)

During the implementation phase of my project, I focused on developing key segments of code that contribute to the overall functionality of GabGrid. The main areas of development include:

**Frontend**: I used React.js to develop the user interface. Core components such as the feed, profile page, and post creation were implemented using reusable components to maintain a modular codebase. I utilized CSS and Material-UI for styling.

**Backend**: The backend was developed using Node.js with Express.js. I built a RESTful API that communicates with the frontend, allowing users to register, log in, and manage posts. I also used JWT for user authentication and bcrypt to securely hash passwords.

**Database**: I used MongoDB to store user data, posts, comments, and related information. Key features include real-time updates and efficient data retrieval.

**Real-time Functionality**: For real-time updates, I used Socket.io to ensure that users get instant notifications and updates to their feed when someone comments on or likes a post.

The core implementation required integrating all these technologies to create a seamless user experience, ensuring that each component interacts with the other efficiently.

## 4.2 Testing Approach

Testing was an essential part of the development process to ensure that the application functions as expected and is free from bugs. I adopted a multi-step testing approach which included unit testing and integration testing.

### 4.2.1 Unit Testing (Test Cases and Test Results)

I conducted unit testing to validate individual components of the system. Some of the areas tested include:

User Authentication: Tested the signup and login functionality, ensuring that user credentials are securely handled and verified. For this, I created test cases that validated if the correct user details were being processed and whether the system handled incorrect inputs gracefully.

Post Creation: I tested the post creation feature to ensure that users could create, edit, and delete posts. The tests also validated that posts appeared on the feed instantly and were stored correctly in the database.

Real-Time Features: I tested the real-time functionality provided by Socket.io to ensure that notifications and updates occurred in
real-time. Test cases checked whether users received live notifications when someone interacted with their post.

For testing, I used Jest as the primary testing tool. All critical features passed the test cases, ensuring that each component worked as expected.

### 4.2.2 Integration System (Test Cases and Test Results)

Once individual components were tested, I moved on to integration testing to ensure that all components of the system worked together seamlessly. This step was crucial to identify issues that might arise when components communicate with one another.

Some of the areas tested include:
User Authentication and Database Interaction: Ensured that user login triggered a JWT token generation and that the user's session was correctly managed across the system.

Post Management and Feed Updates: Verified that when a user creates or edits a post, it is correctly saved in the database and reflects on the user feed immediately.

Real-time updates were tested by having multiple users logged in and interacting with the same post simultaneously.

Notifications: Tested if real-time notifications triggered correctly when different users interacted with each other’s posts.

Code Details

1. backend/db/connectMongoDB.js

```
import mongoose from "mongoose";

const connectMongoDB = async () => {
    try {
        const conn = await
mongoose.connect(process.env.MONGO_URI);
        console.log(`MongoDB connected:
${conn.connection.host}`);
    } catch (error) {
        console.error(`Error connection to
mongoDB: ${error.message}`);
        process.exit(1);
    }
};

export default connectMongoDB;
```

2. backend/controllers/auth.controller.js

```
import { generateTokenAndSetCookie } from
"../lib/utils/generateToken.js";
import User from "../models/user.model.js";
import bcrypt from "bcryptjs";

export const signup = async (req, res) => {
    try {
        const { fullName, username, email,
password } = req.body;

        const emailRegex =
/^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        if (!emailRegex.test(email)) {
            return res.status(400).json({ error:
"Invalid email format" });
        }

        const existingUser = await User.findOne({
username });
        if (existingUser) {
            return res.status(400).json({ error:
"Username is already taken" });
        }

        const existingEmail = await User.findOne({
email });
        if (existingEmail) {
            return res.status(400).json({ error:
"Email is already taken" });
        }

        if (password.length < 6) {
            return res.status(400).json({ error:
"Password must be at least 6 characters long" });
        }

        const salt = await bcrypt.genSalt(10);
        const hashedPassword = await
bcrypt.hash(password, salt);

        const newUser = new User({
            fullName,
            username,
            email,
            password: hashedPassword,
        });

        if (newUser) {
            generateTokenAndSetCookie(newUser._id,
res);
            await newUser.save();

            res.status(201).json({
                _id: newUser._id,
                fullName: newUser.fullName,
                username: newUser.username,
                email: newUser.email,
                followers: newUser.followers,
                following: newUser.following,
                profileImg: newUser.profileImg,
                coverImg: newUser.coverImg,
            });
        } else {
            res.status(400).json({ error: "Invalid
user data" });
        }
    } catch (error) {
        console.log("Error in signup controller",
error.message);
        res.status(500).json({ error: "Internal
Server Error" });
    }
};

export const login = async (req, res) => {
    try {
        const { username, password } = req.body;
        const user = await User.findOne({ username
});
```

```
        const isPasswordCorrect = await
bcrypt.compare(password, user?.password || "");

        if (!user || !isPasswordCorrect) {
            return res.status(400).json({ error:
"Invalid username or password" });
        }

        generateTokenAndSetCookie(user._id, res);

        res.status(200).json({
            _id: user._id,
            fullName: user.fullName,
            username: user.username,
            email: user.email,
            followers: user.followers,
            following: user.following,
            profileImg: user.profileImg,
            coverImg: user.coverImg,
        });
    } catch (error) {
        console.log("Error in login controller",
error.message);
        res.status(500).json({ error: "Internal
Server Error" });
    }
};

export const logout = async (req, res) => {
    try {
        res.cookie("jwt", "", { maxAge: 0 });
        res.status(200).json({ message: "Logged
out successfully" });
    } catch (error) {
        console.log("Error in logout controller",
error.message);
        res.status(500).json({ error: "Internal
Server Error" });
    }
};

export const getMe = async (req, res) => {
    try {
        const user = await
User.findById(req.user._id).select("-password");
        res.status(200).json(user);
    } catch (error) {
        console.log("Error in getMe controller",
error.message);
        res.status(500).json({ error: "Internal
Server Error" });
    }
};
```

3. backend/controllers/Notification.controller.js

```
import Notification from
"../models/notification.model.js";

export const getNotifications = async (req, res)
=> {
    try {
        const userId = req.user._id;

        const notifications = await
Notification.find({ to: userId }).populate({
```

```
            path: "from",
            select: "username profileImg",
        });

        await Notification.updateMany({ to: userId
}, { read: true });

        res.status(200).json(notifications);
    } catch (error) {
        console.log("Error in getNotifications
function", error.message);
        res.status(500).json({ error: "Internal
Server Error" });
    }
};

export const deleteNotifications = async (req,
res) => {
    try {
        const userId = req.user._id;

        await Notification.deleteMany({ to: userId
});

        res.status(200).json({ message:
"Notifications deleted successfully" });
    } catch (error) {
        console.log("Error in deleteNotifications
function", error.message);
        res.status(500).json({ error: "Internal
Server Error" });
    }
};
```

4. backend/controllers/post.controller.js

```
import Notification from
'../models/notification.model.js'
import Post from '../models/post.model.js'
import User from '../models/user.model.js'
import { v2 as cloudinary } from 'cloudinary'

export const createPost = async (req, res) => {
  try {
    const { text } = req.body
    let { img } = req.body
    const userId = req.user._id.toString()

    const user = await User.findById(userId)
    if (!user) return res.status(404).json({
message: 'User not found' })

    if (!text && !img) {
      return res.status(400).json({ error: 'Post
must have text or image' })
    }

    if (img) {
      const uploadedResponse = await
cloudinary.uploader.upload(img)
      img = uploadedResponse.secure_url
    }

    const newPost = new Post({
      user: userId,
      text,
      img,
    })

    await newPost.save()
    res.status(201).json(newPost)
  } catch (error) {
    res.status(500).json({ error: 'Internal server
error' })
    console.log('Error in createPost controller:
', error)
  }
}

export const deletePost = async (req, res) => {
  try {
    const post = await
Post.findById(req.params.id)
    if (!post) {
      return res.status(404).json({ error: 'Post
not found' })
    }

    if (post.user.toString() !==
req.user._id.toString()) {
      return res
        .status(401)
        .json({ error: 'You are not authorized to
delete this post' })
    }

    if (post.img) {
      const imgId =
post.img.split('/').pop().split('.')[0]
      await cloudinary.uploader.destroy(imgId)
    }

    await Post.findByIdAndDelete(req.params.id)

    res.status(200).json({ message: 'Post deleted
successfully' })
  } catch (error) {
    console.log('Error in deletePost controller:
', error)
    res.status(500).json({ error: 'Internal server
error' })
  }
}

export const commentOnPost = async (req, res) => {
  try {
    const { text } = req.body
    const postId = req.params.id
    const userId = req.user._id

    if (!text) {
      return res.status(400).json({ error: 'Text
field is required' })
    }
    const post = await Post.findById(postId)

    if (!post) {
      return res.status(404).json({ error: 'Post
not found' })
    }

    const comment = { user: userId, text }
    post.comments.push(comment)
    await post.save()

    res.status(200).json(post)
  } catch (error) {
    console.log('Error in commentOnPost
controller: ', error)
    res.status(500).json({ error: 'Internal server
error' })
  }
}

export const likeUnlikePost = async (req, res) =>
{
  try {
    const userId = req.user._id
    const { id: postId } = req.params

    const post = await Post.findById(postId)

    if (!post) {
      return res.status(404).json({ error: 'Post
not found' })
    }

    const userLikedPost =
post.likes.includes(userId)

    if (userLikedPost) {
      // Unlike post
      await Post.updateOne({ _id: postId }, {
$pull: { likes: userId } })
      await User.updateOne({ _id: userId }, {
```

```
$pull: { likedPosts: postId } })

      const updatedLikes = post.likes.filter(
        (id) => id.toString() !==
userId.toString()
      )
      res.status(200).json(updatedLikes)
    } else {
      // Like post
      post.likes.push(userId)
      await User.updateOne({ _id: userId }, {
$push: { likedPosts: postId } })
      await post.save()

      const notification = new Notification({
        from: userId,
        to: post.user,
        type: 'like',
      })
      await notification.save()

      const updatedLikes = post.likes
      res.status(200).json(updatedLikes)
    }
  } catch (error) {
    console.log('Error in likeUnlikePost
controller: ', error)
    res.status(500).json({ error: 'Internal server
error' })
  }
}

export const getAllPosts = async (req, res) => {
  try {
    const posts = await Post.find()
      .sort({ createdAt: -1 })
      .populate({
        path: 'user',
        select: '-password',
      })
      .populate({
        path: 'comments.user',
        select: '-password',
      })

    if (posts.length === 0) {
      return res.status(200).json([])
    }

    res.status(200).json(posts)
  } catch (error) {
    console.log('Error in getAllPosts controller: 
', error)
    res.status(500).json({ error: 'Internal server 
error' })
  }
}

export const getLikedPosts = async (req, res) => {
  const userId = req.params.id

  try {
    const user = await User.findById(userId)
    if (!user) return res.status(404).json({ 
error: 'User not found' })
```

```
    const likedPosts = await Post.find({ _id: {
$in: user.likedPosts } })
      .populate({
        path: 'user',
        select: '-password',
      })
      .populate({
        path: 'comments.user',
        select: '-password',
      })

    res.status(200).json(likedPosts)
  } catch (error) {
    console.log('Error in getLikedPosts
controller: ', error)
    res.status(500).json({ error: 'Internal server
error' })
  }
}

export const getFollowingPosts = async (req, res)
=> {
  try {
    const userId = req.user._id
    const user = await User.findById(userId)
    if (!user) return res.status(404).json({
error: 'User not found' })

    const following = user.following

    const feedPosts = await Post.find({ user: {
$in: following } })
      .sort({ createdAt: -1 })
      .populate({
        path: 'user',
        select: '-password',
      })
      .populate({
        path: 'comments.user',
        select: '-password',
      })

    res.status(200).json(feedPosts)
  } catch (error) {
    console.log('Error in getFollowingPosts
controller: ', error)
    res.status(500).json({ error: 'Internal server
error' })
  }
}

export const getUserPosts = async (req, res) => {
  try {
    const { username } = req.params

    const user = await User.findOne({ username })
    if (!user) return res.status(404).json({
error: 'User not found' })

    const posts = await Post.find({ user: user._id
})
      .sort({ createdAt: -1 })
      .populate({
        path: 'user',
        select: '-password',
      })
      .populate({
        path: 'comments.user',
        select: '-password',
      })

    res.status(200).json(posts)
  } catch (error) {
    console.log('Error in getUserPosts controller:
', error)
    res.status(500).json({ error: 'Internal server
error' })
  }
}
```

5. backend/controllers/user.controller.js

```
import bcrypt from "bcryptjs";
import { v2 as cloudinary } from "cloudinary";

// models
import Notification from
"../models/notification.model.js";
import User from "../models/user.model.js";

export const getUserProfile = async (req, res) =>
{
    const { username } = req.params;

    try {
        const user = await User.findOne({ username
}).select("-password");
        if (!user) return res.status(404).json({
message: "User not found" });
```

```
        res.status(200).json(user);
    } catch (error) {
        console.log("Error in getUserProfile: ",
error.message);
        res.status(500).json({ error:
error.message });
    }
};

export const followUnfollowUser = async (req, res)
=> {
    try {
        const { id } = req.params;
        const userToModify = await
User.findById(id);
        const currentUser = await
User.findById(req.user._id);

        if (id === req.user._id.toString()) {
            return res.status(400).json({ error:
"You can't follow/unfollow yourself" });
        }

        if (!userToModify || !currentUser) return
res.status(400).json({ error: "User not found" });

        const isFollowing =
currentUser.following.includes(id);

        if (isFollowing) {
            // Unfollow the user
            await User.findByIdAndUpdate(id, {
```

```
$pull: { followers: req.user._id } });
            await
User.findByIdAndUpdate(req.user._id, { $pull: {
following: id } });

            res.status(200).json({ message: "User
unfollowed successfully" });
        } else {
            // Follow the user
            await User.findByIdAndUpdate(id, {
$push: { followers: req.user._id } });
            await
User.findByIdAndUpdate(req.user._id, { $push: {
following: id } });
            // Send notification to the user
            const newNotification = new
Notification({
                type: "follow",
                from: req.user._id,
                to: userToModify._id,
            });

            await newNotification.save();

            res.status(200).json({ message: "User
followed successfully" });
        }
    } catch (error) {
        console.log("Error in followUnfollowUser:
", error.message);
        res.status(500).json({ error:
error.message });
    }
```

```
};

export const getSuggestedUsers = async (req, res)
=> {
    try {
        const userId = req.user._id;

        const usersFollowedByMe = await
User.findById(userId).select("following");

        const users = await User.aggregate([
            {
                $match: {
                    _id: { $ne: userId },
                },
            },
            { $sample: { size: 10 } },
        ]);

        // 1,2,3,4,5,6,
        const filteredUsers = users.filter((user)
=>
!usersFollowedByMe.following.includes(user._id));
        const suggestedUsers =
filteredUsers.slice(0, 4);

        suggestedUsers.forEach((user) =>
(user.password = null));

        res.status(200).json(suggestedUsers);
    } catch (error) {
        console.log("Error in getSuggestedUsers:
", error.message);
```

```
        res.status(500).json({ error:
error.message });
    }
};

export const updateUser = async (req, res) => {
    const { fullName, email, username,
currentPassword, newPassword, bio, link } =
req.body;
    let { profileImg, coverImg } = req.body;

    const userId = req.user._id;

    try {
        let user = await User.findById(userId);
        if (!user) return res.status(404).json({
message: "User not found" });

        if ((!newPassword && currentPassword) ||
(!currentPassword && newPassword)) {
            return res.status(400).json({ error:
"Please provide both current password and new
password" });
        }

        if (currentPassword && newPassword) {
            const isMatch = await
bcrypt.compare(currentPassword, user.password);
            if (!isMatch) return
res.status(400).json({ error: "Current password is
incorrect" });
            if (newPassword.length < 6) {
                return res.status(400).json({
```

```
error: "Password must be at least 6 characters
long" });
            }

            const salt = await bcrypt.genSalt(10);
            user.password = await
bcrypt.hash(newPassword, salt);
        }

        if (profileImg) {
            if (user.profileImg) {
                //
https://res.cloudinary.com/dyfqon1v6/image/upload/
v1712997552/zmxorcxexpdbh8r0bkjb.png
                await
cloudinary.uploader.destroy(user.profileImg.split(
"/").pop().split(".")[0]);
            }

            const uploadedResponse = await
cloudinary.uploader.upload(profileImg);
            profileImg =
uploadedResponse.secure_url;
        }

        if (coverImg) {
            if (user.coverImg) {
                await
cloudinary.uploader.destroy(user.coverImg.split("/
").pop().split(".")[0]);
            }

            const uploadedResponse = await
```

```
cloudinary.uploader.upload(coverImg);
            coverImg =
uploadedResponse.secure_url;
        }

        user.fullName = fullName || user.fullName;
        user.email = email || user.email;
        user.username = username || user.username;
        user.bio = bio || user.bio;
        user.link = link || user.link;
        user.profileImg = profileImg ||
user.profileImg;
        user.coverImg = coverImg || user.coverImg;

        user = await user.save();

        // password should be null in response
        user.password = null;

        return res.status(200).json(user);
    } catch (error) {
        console.log("Error in updateUser: ",
error.message);
        res.status(500).json({ error:
error.message });
    }
};
```

6. backend/server.js

```
import path from "path";
import express from "express";
import dotenv from "dotenv";
import cookieParser from "cookie-parser";
import { v2 as cloudinary } from "cloudinary";

import authRoutes from "./routes/auth.route.js";
import userRoutes from "./routes/user.route.js";
import postRoutes from "./routes/post.route.js";
import notificationRoutes from
"./routes/notification.route.js";

import connectMongoDB from
"./db/connectMongoDB.js";

dotenv.config();

cloudinary.config({
    cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
    api_key: process.env.CLOUDINARY_API_KEY,
    api_secret: process.env.CLOUDINARY_API_SECRET,
});

const app = express();
const PORT = process.env.PORT || 5000;
const __dirname = path.resolve();

app.use(express.json({ limit: "5mb" })); // to
parse req.body
// limit shouldn't be too high to prevent DOS
app.use(express.urlencoded({ extended: true }));
// to parse form data(urlencoded)

app.use(cookieParser());

app.use("/api/auth", authRoutes);
app.use("/api/users", userRoutes);
app.use("/api/posts", postRoutes);
app.use("/api/notifications", notificationRoutes);

if (process.env.NODE_ENV === "production") {
    app.use(express.static(path.join(__dirname,
"/frontend/dist")));

    app.get("*", (req, res) => {
        res.sendFile(path.resolve(__dirname,
"frontend", "dist", "index.html"));
    });
}

app.listen(PORT, () => {
    console.log(`Server is running on port
${PORT}`);
    connectMongoDB();
});
```

7. frontend/src/pages/auth/login.js

```
import { useState } from "react";
import { Link } from "react-router-dom";

import XSvg from "../../../components/svgs/X";

import { MdOutlineMail } from "react-icons/md";
import { MdPassword } from "react-icons/md";

import { useMutation, useQueryClient } from
"@tanstack/react-query";
```

```
const LoginPage = () => {
    const [formData, setFormData] = useState({
        username: "",
        password: "",
    });
    const queryClient = useQueryClient();

    const {
        mutate: loginMutation,
        isPending,
        isError,
        error,
    } = useMutation({
        mutationFn: async ({ username, password })
=> {
            try {
                const res = await
fetch("/api/auth/login", {
                    method: "POST",
                    headers: {
                        "Content-Type":
"application/json",
                    },
                    body: JSON.stringify({
username, password }),
                });

                const data = await res.json();

                if (!res.ok) {
                    throw new Error(data.error ||
"Something went wrong");
                }
            } catch (error) {
                throw new Error(error);
            }
        },
        onSuccess: () => {
            // refetch the authUser
            queryClient.invalidateQueries({
queryKey: ["authUser"] });
        },
    });

    const handleSubmit = (e) => {
        e.preventDefault();
        loginMutation(formData);
    };

    const handleInputChange = (e) => {
        setFormData({ ...formData,
[e.target.name]: e.target.value });
    };

    return (
        <div className='max-w-screen-xl mx-auto
flex h-screen'>
            <div className='flex-1 hidden lg:flex
items-center  justify-center'>
                <XSvg className='lg:w-2/3
fill-white' />
            </div>
            <div className='flex-1 flex flex-col
justify-center items-center'>
                <form className='flex gap-4
flex-col' onSubmit={handleSubmit}>
```

```
                    <XSvg className='w-24
lg:hidden fill-white' />
                    <h1 className='text-4xl
font-extrabold text-white'>{"Let's"} go.</h1>
                    <label className='input
input-bordered rounded flex items-center gap-2'>
                        <MdOutlineMail />
                        <input
                            type='text'
                            className='grow'

placeholder='username'
                            name='username'

onChange={handleInputChange}

value={formData.username}
                        />
                    </label>

                    <label className='input
input-bordered rounded flex items-center gap-2'>
                        <MdPassword />
                        <input
                            type='password'
                            className='grow'

placeholder='Password'
                            name='password'

onChange={handleInputChange}

value={formData.password}
```

```
                        />
                    </label>
                    <button className='btn
rounded-full btn-primary text-white'>
                        {isPending ? "Loading..."
: "Login"}
                    </button>
                    {isError && <p
className='text-red-500'>{error.message}</p>}
                </form>
                <div className='flex flex-col
gap-2 mt-4'>
                    <p className='text-white
text-lg'>{"Don't"} have an account?</p>
                    <Link to='/signup'>
                        <button className='btn
rounded-full btn-primary text-white btn-outline
w-full'>Sign up</button>
                    </Link>
                </div>
            </div>
        </div>
    );
};
export default LoginPage;
```

8. frontend/src/pages/auth/signup.js

```
import { Link } from "react-router-dom";
import { useState } from "react";

import XSvg from "../../../components/svgs/X";

import { MdOutlineMail } from "react-icons/md";
import { FaUser } from "react-icons/fa";
import { MdPassword } from "react-icons/md";
import { MdDriveFileRenameOutline } from
"react-icons/md";
import { useMutation, useQueryClient } from
"@tanstack/react-query";
import toast from "react-hot-toast";

const SignUpPage = () => {
    const [formData, setFormData] = useState({
        email: "",
        username: "",
        fullName: "",
        password: "",
    });

    const queryClient = useQueryClient();

    const { mutate, isError, isPending, error } =
useMutation({
        mutationFn: async ({ email, username,
fullName, password }) => {
            try {
                const res = await
fetch("/api/auth/signup", {
                    method: "POST",
                    headers: {
                        "Content-Type":
"application/json",
                    },
                    body: JSON.stringify({ email,
username, fullName, password }),
                });

                const data = await res.json();
                if (!res.ok) throw new
Error(data.error || "Failed to create account");
                console.log(data);
                return data;
            } catch (error) {
                console.error(error);
                throw error;
            }
        },
        onSuccess: () => {
            toast.success("Account created
successfully");

            {
                /* Added this line below, after
recording the video. I forgot to add this while
recording, sorry, thx. */
            }
            queryClient.invalidateQueries({
queryKey: ["authUser"] });
        },
    });

    const handleSubmit = (e) => {
        e.preventDefault(); // page won't reload
        mutate(formData);
    };

    const handleInputChange = (e) => {
        setFormData({ ...formData,
[e.target.name]: e.target.value });
    };

    return (
        <div className='max-w-screen-xl mx-auto
flex h-screen px-10'>
            <div className='flex-1 hidden lg:flex
items-center  justify-center'>
                <XSvg className='lg:w-2/3
fill-white' />
            </div>
            <div className='flex-1 flex flex-col
justify-center items-center'>
                <form className='lg:w-2/3
mx-auto md:mx-20 flex gap-4 flex-col'
onSubmit={handleSubmit}>
                    <XSvg className='w-24
lg:hidden fill-white' />
                    <h1 className='text-4xl
font-extrabold text-white'>Join today.</h1>
                    <label className='input
input-bordered rounded flex items-center gap-2'>
                        <MdOutlineMail />
                        <input
                            type='email'
                            className='grow'
                            placeholder='Email'
                            name='email'

onChange={handleInputChange}

value={formData.email}
                        />
                    </label>
                    <div className='flex gap-4
flex-wrap'>
                        <label className='input
input-bordered rounded flex items-center gap-2
flex-1'>
                            <FaUser />
                            <input
                                type='text'
                                className='grow '

placeholder='Username'
                                name='username'

onChange={handleInputChange}

value={formData.username}
                            />
                        </label>
                        <label className='input
input-bordered rounded flex items-center gap-2
flex-1'>

<MdDriveFileRenameOutline />
                            <input
                                type='text'
                                className='grow'
                                placeholder='Full
Name'
                                name='fullName'

onChange={handleInputChange}
```

```
value={formData.fullName}
                            />
                        </label>
                    </div>
                    <label className='input
input-bordered rounded flex items-center gap-2'>
                        <MdPassword />
                        <input
                            type='password'
                            className='grow'

placeholder='Password'
                            name='password'

onChange={handleInputChange}

value={formData.password}
                        />
                    </label>
                    <button className='btn
rounded-full btn-primary text-white'>
                        {isPending ? "Loading..."
: "Sign up"}
                    </button>
                    {isError && <p
className='text-red-500'>{error.message}</p>}
                </form>
                <div className='flex flex-col
lg:w-2/3 gap-2 mt-4'>
                    <p className='text-white
text-lg'>Already have an account?</p>
                    <Link to='/login'>
                        <button className='btn
```

```
rounded-full btn-primary text-white btn-outline
w-full'>Sign in</button>
                    </Link>
                </div>
            </div>
        </div>
    );
};
export default SignUpPage;
```

9. frontend/src/pages/home/create.js

```
import { CiImageOn } from "react-icons/ci";
import { BsEmojiSmileFill } from "react-icons/bs";
import { useRef, useState } from "react";
import { IoCloseSharp } from "react-icons/io5";
import { useMutation, useQuery, useQueryClient }
from "@tanstack/react-query";
import { toast } from "react-hot-toast";

const CreatePost = () => {
    const [text, setText] = useState("");
    const [img, setImg] = useState(null);
    const imgRef = useRef(null);

    const { data: authUser } = useQuery({
queryKey: ["authUser"] });
    const queryClient = useQueryClient();

    const {
        mutate: createPost,
        isPending,
        isError,
        error,
    } = useMutation({
        mutationFn: async ({ text, img }) => {
            try {
                const res = await
fetch("/api/posts/create", {
                    method: "POST",
                    headers: {
                        "Content-Type":
"application/json",
                    },
                    body: JSON.stringify({ text,
img }),
                });
                const data = await res.json();
                if (!res.ok) {
                    throw new Error(data.error ||
"Something went wrong");
                }
                return data;
            } catch (error) {
                throw new Error(error);
            }
        },

        onSuccess: () => {
            setText("");
            setImg(null);
            toast.success("Post created
successfully");
            queryClient.invalidateQueries({
queryKey: ["posts"] });
```

```
        },
    });

    const handleSubmit = (e) => {
        e.preventDefault();
        createPost({ text, img });
    };

    const handleImgChange = (e) => {
        const file = e.target.files[0];
        if (file) {
            const reader = new FileReader();
            reader.onload = () => {
                setImg(reader.result);
            };
            reader.readAsDataURL(file);
        }
    };

    return (
        <div className='flex p-4 items-start gap-4
border-b border-gray-700'>
            <div className='avatar'>
                <div className='w-8
rounded-full'>
                    <img src={authUser.profileImg
|| "/avatar-placeholder.png"} />
                </div>
            </div>
            <form className='flex flex-col gap-2
w-full' onSubmit={handleSubmit}>
                <textarea
                    className='textarea w-full
```

```
p-0 text-lg resize-none border-none
focus:outline-none  border-gray-800'
                    placeholder='What is
happening?!'
                    value={text}
                    onChange={(e) =>
setText(e.target.value)}
                />
                {img && (
                    <div className='relative w-72
mx-auto'>
                        <IoCloseSharp
                            className='absolute
top-0 right-0 text-white bg-gray-800 rounded-full
w-5 h-5 cursor-pointer'
                            onClick={() => {
                                setImg(null);

imgRef.current.value = null;
                            }}
                        />
                        <img src={img}
className='w-full mx-auto h-72 object-contain
rounded' />
                    </div>
                )}

                <div className='flex
justify-between border-t py-2 border-t-gray-700'>
                    <div className='flex gap-1
items-center'>
                        <CiImageOn
```

```
className='fill-primary w-6 h-6 cursor-pointer'
                            onClick={() =>
imgRef.current.click()}
                        />
                        <BsEmojiSmileFill
className='fill-primary w-5 h-5 cursor-pointer' />
                    </div>
                    <input type='file'
accept='image/*' hidden ref={imgRef}
onChange={handleImgChange} />
                    <button className='btn
btn-primary rounded-full btn-sm text-white px-4'>
                        {isPending ? "Posting..."
: "Post"}
                    </button>
                </div>
                {isError && <div
className='text-red-500'>{error.message}</div>}
            </form>
        </div>
    );
};
export default CreatePost;
```

10. frontend/src/pages/home/home.js

```
import { useState } from "react";

import Posts from "../../components/common/Posts";
import CreatePost from "./CreatePost";

const HomePage = () => {
    const [feedType, setFeedType] =
useState("forYou");

    return (
        <>
            <div className='flex-[4_4_0] mr-auto
border-r border-gray-700 min-h-screen'>
                {/* Header */}
                <div className='flex w-full
border-b border-gray-700'>
                    <div
                        className={
                            "flex justify-center
flex-1 p-3 hover:bg-secondary transition
duration-300 cursor-pointer relative"
                        }
                        onClick={() =>
setFeedType("forYou")}
                    >
                        For you
                        {feedType === "forYou" &&
(
                            <div
className='absolute bottom-0 w-10  h-1
rounded-full bg-primary'></div>
                        )}
                    </div>
                    <div
                        className='flex
justify-center flex-1 p-3 hover:bg-secondary
transition duration-300 cursor-pointer relative'
                        onClick={() =>
setFeedType("following")}
                    >
```

```
                        Following
                        {feedType === "following"
&& (
                            <div
className='absolute bottom-0 w-10  h-1
rounded-full bg-primary'></div>
                        )}
                    </div>
                </div>

                {/*  CREATE POST INPUT */}
                <CreatePost />

                {/* POSTS */}
                <Posts feedType={feedType} />
            </div>
        </>
    );
};
export default HomePage;
```

11. frontend/src/pages/notification/home.js

```
import { Link } from "react-router-dom";
import { useQuery, useMutation, useQueryClient }
from "@tanstack/react-query";
import { toast } from "react-hot-toast";

import LoadingSpinner from
"../../components/common/LoadingSpinner";

import { IoSettingsOutline } from
"react-icons/io5";
```

```
import { FaUser } from "react-icons/fa";
import { FaHeart } from "react-icons/fa6";

const NotificationPage = () => {
    const queryClient = useQueryClient();
    const { data: notifications, isLoading } =
useQuery({
        queryKey: ["notifications"],
        queryFn: async () => {
            try {
                const res = await
fetch("/api/notifications");
                const data = await res.json();
                if (!res.ok) throw new
Error(data.error || "Something went wrong");
                return data;
            } catch (error) {
                throw new Error(error);
            }
        },
    });

    const { mutate: deleteNotifications } =
useMutation({
        mutationFn: async () => {
            try {
                const res = await
fetch("/api/notifications", {
                    method: "DELETE",
                });
                const data = await res.json();

                if (!res.ok) throw new
Error(data.error || "Something went wrong");
                return data;
            } catch (error) {
                throw new Error(error);
            }
        },
        onSuccess: () => {
            toast.success("Notifications deleted
successfully");
            queryClient.invalidateQueries({
queryKey: ["notifications"] });
        },
        onError: (error) => {
            toast.error(error.message);
        },
    });

    return (
        <>
            <div className='flex-[4_4_0] border-l
border-r border-gray-700 min-h-screen'>
                <div className='flex
justify-between items-center p-4 border-b
border-gray-700'>
                    <p
className='font-bold'>Notifications</p>
                    <div className='dropdown '>
                        <div tabIndex={0}
role='button' className='m-1'>
                            <IoSettingsOutline
className='w-4' />
                        </div>
                        <ul
```

```
                            tabIndex={0}

className='dropdown-content z-[1] menu p-2 shadow
bg-base-100 rounded-box w-52'
                        >
                            <li>
                                <a
onClick={deleteNotifications}>Delete all
notifications</a>
                            </li>
                        </ul>
                    </div>
                </div>
                {isLoading && (
                    <div className='flex
justify-center h-full items-center'>
                        <LoadingSpinner size='lg'
/>
                    </div>
                )}
                {notifications?.length === 0 &&
<div className='text-center p-4 font-bold'>No
notifications 🤔</div>}

{notifications?.map((notification) => (
                    <div className='border-b
border-gray-700' key={notification._id}>
                        <div className='flex
gap-2 p-4'>
                            {notification.type
=== "follow" && <FaUser className='w-7 h-7
text-primary' />}
                            {notification.type
```

};

```
=== "like" && <FaHeart className='w-7 h-7
text-red-500' />}
                            <Link
to={`/profile/${notification.from.username}`}>
                                <div
className='avatar'>
                                    <div
className='w-8 rounded-full'>
                                        <img
src={notification.from.profileImg ||
"/avatar-placeholder.png"} />
                                    </div>
                                </div>
                                <div
className='flex gap-1'>
                                    <span
className='font-bold'>@{notification.from.username
}</span>{" "}

{notification.type === "follow" ? "followed you" :
"liked your post"}
                                </div>
                            </Link>
                        </div>
                    </div>
                ))}
            </div>
        </>
    );
};
export default NotificationPage;
```

12. frontend/src/pages/profile/edit.js

```
import { useEffect, useState } from "react";
import useUpdateUserProfile from
"../../hooks/useUpdateUserProfile";

const EditProfileModal = ({ authUser }) => {
    const [formData, setFormData] = useState({
        fullName: "",
        username: "",
        email: "",
        bio: "",
        link: "",
        newPassword: "",
        currentPassword: "",
    });

    const { updateProfile, isUpdatingProfile } =
useUpdateUserProfile();

    const handleInputChange = (e) => {
        setFormData({ ...formData,
[e.target.name]: e.target.value });
    };

    useEffect(() => {
        if (authUser) {
            setFormData({
                fullName: authUser.fullName,
                username: authUser.username,
                email: authUser.email,
                bio: authUser.bio,
                link: authUser.link,
                newPassword: "",
                currentPassword: "",
            });
        }
    }, [authUser]);

    return (
        <>
            <button
                className='btn btn-outline
rounded-full btn-sm'
                onClick={() =>
document.getElementById("edit_profile_modal").show
Modal()}
            >
                Edit profile
            </button>
            <dialog id='edit_profile_modal'
className='modal'>
                <div className='modal-box border
rounded-md border-gray-700 shadow-md'>
                    <h3 className='font-bold
text-lg my-3'>Update Profile</h3>
                    <form
                        className='flex flex-col
gap-4'
                        onSubmit={(e) => {
                            e.preventDefault();

updateProfile(formData);
                        }}
                    >
                        <div className='flex
flex-wrap gap-2'>
                            <input
```

```
                                type='text'
                                placeholder='Full
Name'
                                className='flex-1
input border border-gray-700 rounded p-2 input-md'

value={formData.fullName}
                                name='fullName'

onChange={handleInputChange}
                            />
                            <input
                                type='text'

placeholder='Username'
                                className='flex-1
input border border-gray-700 rounded p-2 input-md'

value={formData.username}
                                name='username'

onChange={handleInputChange}
                            />
                        </div>
                        <div className='flex
flex-wrap gap-2'>
                            <input
                                type='email'

placeholder='Email'
                                className='flex-1
input border border-gray-700 rounded p-2 input-md'
```

```
value={formData.email}
                                name='email'

onChange={handleInputChange}
                            />
                            <textarea
                                placeholder='Bio'
                                className='flex-1
input border border-gray-700 rounded p-2 input-md'

value={formData.bio}
                                name='bio'

onChange={handleInputChange}
                            />
                        </div>
                        <div className='flex
flex-wrap gap-2'>
                            <input
                                type='password'

placeholder='Current Password'
                                className='flex-1
input border border-gray-700 rounded p-2 input-md'

value={formData.currentPassword}

name='currentPassword'

onChange={handleInputChange}
                            />
                            <input
                                type='password'
                                placeholder='New
Password'
                                className='flex-1
input border border-gray-700 rounded p-2 input-md'

value={formData.newPassword}

name='newPassword'

onChange={handleInputChange}
                            />
                        </div>
                        <input
                            type='text'
                            placeholder='Link'
                            className='flex-1
input border border-gray-700 rounded p-2 input-md'
                            value={formData.link}
                            name='link'

onChange={handleInputChange}
                        />
                        <button className='btn
btn-primary rounded-full btn-sm text-white'>
                            {isUpdatingProfile ?
"Updating..." : "Update"}
                        </button>
                    </form>
                </div>
                <form method='dialog'
className='modal-backdrop'>
                    <button
className='outline-none'>close</button>
```

```
                </form>
            </dialog>
        </>
    );
};
export default EditProfileModal;
```

13. frontend/src/pages/profile/profile.js

```
import { useEffect, useRef, useState } from
'react'
import { Link, useParams } from 'react-router-dom'

import Posts from '../../components/common/Posts'
import ProfileHeaderSkeleton from
'../../components/skeletons/ProfileHeaderSkeleton'
import EditProfileModal from './EditProfileModal'

import { POSTS } from '../../utils/db/dummy'

import { FaArrowLeft } from 'react-icons/fa6'
import { IoCalendarOutline } from
'react-icons/io5'
import { FaLink } from 'react-icons/fa'
import { MdEdit } from 'react-icons/md'
import { useQuery } from '@tanstack/react-query'
import { formatMemberSinceDate } from
'../../utils/date'

import useFollow from '../../hooks/useFollow'
import useUpdateUserProfile from
'../../hooks/useUpdateUserProfile'
```

```
const ProfilePage = () => {
  const [coverImg, setCoverImg] = useState(null)
  const [profileImg, setProfileImg] =
useState(null)
  const [feedType, setFeedType] =
useState('posts')

  const coverImgRef = useRef(null)
  const profileImgRef = useRef(null)

  const { username } = useParams()

  const { follow, isPending } = useFollow()
  const { data: authUser } = useQuery({ queryKey:
['authUser'] })

  const {
    data: user,
    isLoading,
    refetch,
    isRefetching,
  } = useQuery({
    queryKey: ['userProfile'],
    queryFn: async () => {
      try {
        const res = await
fetch(`/api/users/profile/${username}`)
        const data = await res.json()
        if (!res.ok) {
          throw new Error(data.error || 'Something
went wrong')
        }
        return data
      } catch (error) {
        throw new Error(error)
      }
    },
  })

  const { isUpdatingProfile, updateProfile } =
useUpdateUserProfile()

  const isMyProfile = authUser._id === user?._id
  const memberSinceDate =
formatMemberSinceDate(user?.createdAt)
  const amIFollowing =
authUser?.following.includes(user?._id)

  const handleImgChange = (e, state) => {
    const file = e.target.files[0]
    if (file) {
      const reader = new FileReader()
      reader.onload = () => {
        state === 'coverImg' &&
setCoverImg(reader.result)
        state === 'profileImg' &&
setProfileImg(reader.result)
      }
      reader.readAsDataURL(file)
    }
  }

  useEffect(() => {
    refetch()
  }, [username, refetch])
  return (
    <>
      <div className="flex-[4_4_0]  border-r
border-gray-700 min-h-screen ">
        {/* HEADER */}
        {(isLoading || isRefetching) &&
<ProfileHeaderSkeleton />}
        {!isLoading && !isRefetching && !user && (
          <p className="text-center text-lg
mt-4">User not found</p>
        )}
        <div className="flex flex-col">
          {!isLoading && !isRefetching && user &&
(
            <>
              <div className="flex gap-10 px-4
py-2 items-center">
                <Link to="/">
                  <FaArrowLeft className="w-4 h-4"
/>
                </Link>
                <div className="flex flex-col">
                  <p className="font-bold
text-lg">{user?.fullName}</p>

                  {/* <span className="text-sm
text-slate-500">
                    {POSTS?.length} posts
                  </span> */}
                </div>
              </div>
              {/* COVER IMG */}
              <div className="relative
```

```
group/cover">
                <img
                  src={coverImg || user?.coverImg
|| '/cover.png'}
                  className="h-52 w-full
object-cover"
                  alt="cover image"
                />
                {isMyProfile && (
                  <div
                    className="absolute top-2
right-2 rounded-full p-2 bg-gray-800 bg-opacity-75
cursor-pointer opacity-0
group-hover/cover:opacity-100 transition
duration-200"
                    onClick={() =>
coverImgRef.current.click()}
                  >
                    <MdEdit className="w-5 h-5
text-white" />
                  </div>
                )}

                <input
                  type="file"
                  hidden
                  accept="image/*"
                  ref={coverImgRef}
                  onChange={(e) =>
handleImgChange(e, 'coverImg')}
                />
                <input
                  type="file"
                  hidden
                  accept="image/*"
                  ref={profileImgRef}
                  onChange={(e) =>
handleImgChange(e, 'profileImg')}
                />
                {/* USER AVATAR */}
                <div className="avatar absolute
-bottom-16 left-4">
                  <div className="w-32
rounded-full relative group/avatar">
                    <img
                      src={
                        profileImg ||
                        user?.profileImg ||
                        '/avatar-placeholder.png'
                      }
                    />
                    <div className="absolute top-5
right-3 p-1 bg-primary rounded-full
group-hover/avatar:opacity-100 opacity-0
cursor-pointer">
                      {isMyProfile && (
                        <MdEdit
                          className="w-4 h-4
text-white"
                          onClick={() =>
profileImgRef.current.click()}
                        />
                      )}
                    </div>
                  </div>
                </div>
```

```
              </div>
              <div className="flex justify-end
px-4 mt-5">
                {isMyProfile && <EditProfileModal
authUser={authUser} />}
                {!isMyProfile && (
                  <button
                    className="btn btn-outline
rounded-full btn-sm"
                    onClick={() =>
follow(user?._id)}
                  >
                    {isPending && 'Loading...'}
                    {!isPending && amIFollowing &&
'Unfollow'}
                    {!isPending && !amIFollowing
&& 'Follow'}
                  </button>
                )}
                {(coverImg || profileImg) && (
                  <button
                    className="btn btn-primary
rounded-full btn-sm text-white px-4 ml-2"
                    onClick={async () => {
                      await updateProfile({
coverImg, profileImg })
                      setProfileImg(null)
                      setCoverImg(null)
                    }}
                  >
                    {isUpdatingProfile ?
'Updating...' : 'Update'}
                  </button>
                )}
              </div>

              <div className="flex flex-col gap-4
mt-14 px-4">
                <div className="flex flex-col">
                  <span className="font-bold
text-lg">{user?.fullName}</span>
                  <span className="text-sm
text-slate-500">
                    @{user?.username}
                  </span>
                  <span className="text-sm
my-1">{user?.bio}</span>
                </div>

                <div className="flex gap-2
flex-wrap">
                  {user?.link && (
                    <div className="flex gap-1
items-center ">
                      <>
                        <FaLink className="w-3 h-3
text-slate-500" />
                        <a

href="https://youtube.com/@asaprogrammer_"
                          target="_blank"
                          rel="noreferrer"
                          className="text-sm
text-blue-500 hover:underline"
                        >
                          {/* Updated this after
recording the video. I forgot to update this while
recording, sorry, thx. */}
                          {user?.link}
                        </a>
                      </>
                    </div>
                  )}
                  <div className="flex gap-2
items-center">
                    <IoCalendarOutline
className="w-4 h-4 text-slate-500" />
                    <span className="text-sm
text-slate-500">
                      {memberSinceDate}
                    </span>
                  </div>
                </div>
                <div className="flex gap-2">
                  <div className="flex gap-1
items-center">
                    <span className="font-bold
text-xs">
                      {user?.following.length}
                    </span>
                    <span
className="text-slate-500
text-xs">Following</span>
                  </div>
                  <div className="flex gap-1
items-center">
                    <span className="font-bold
text-xs">
                      {user?.followers.length}
                    </span>
                    <span
className="text-slate-500
text-xs">Followers</span>
                  </div>
                </div>
              </div>
              <div className="flex w-full border-b
border-gray-700 mt-4">
                <div
                  className="flex justify-center
flex-1 p-3 hover:bg-secondary transition
duration-300 relative cursor-pointer"
                  onClick={() =>
setFeedType('posts')}
                >
                  Posts
                  {feedType === 'posts' && (
                    <div className="absolute
bottom-0 w-10 h-1 rounded-full bg-primary" />
                  )}
                </div>
                <div
                  className="flex justify-center
flex-1 p-3 text-slate-500 hover:bg-secondary
transition duration-300 relative cursor-pointer"
                  onClick={() =>
setFeedType('likes')}
                >
                  Likes
                  {feedType === 'likes' && (
                    <div className="absolute
bottom-0 w-10  h-1 rounded-full bg-primary" />
```

```
                    )}
                  </div>
                </div>
              </>
            )}

            <Posts feedType={feedType}
username={username} userId={user?._id} />
          </div>
        </div>
      </>
  )
}
export default ProfilePage
```

## Testing Approach

### 1. Unit Testing

- Implemented Jest and React Testing Library for frontend component testing.
- Used Mocha and Chai for backend API validation.
- Ensured individual functions and modules work as expected.

### 2. Integration Testing

- Verified seamless interaction between frontend and backend.
- Tested real-time collaboration in the code editor using WebSockets.
- Ensured smooth data flow in resume generation and problem-solving modules.

### 3. Beta Testing

- Conducted testing with a group of students to gather feedback.
- Identified usability issues and performance bottlenecks.
- Validated AI-generated code suggestions for accuracy.

## 4. Modifications and Improvements Test Cases

- Regularly updated test cases based on user feedback and bug reports.
- Ensured backward compatibility after feature enhancements.
- Monitored system performance post-deployment with Cloudflare analytics.

# Chapter 6 - Implementation & Testing Approach

## 1. User Authentication

### TC-001: User Registration

- **Description:** Ensure users can register successfully.
- **Steps:**
    1. Navigate to the registration page.
    2. Enter valid user details (username, email, password).
    3. Click on the "Sign Up" button.
- **Expected Result:** User account is created successfully.
- **Actual Outcome:** As expected.
- **Status:** ✅ Passed

### TC-002: User Login

- **Description:** Validate login with correct credentials.
- **Steps:**
    1. Navigate to the login page.
    2. Enter valid credentials.
    3. Click on the "Login" button.
- **Expected Result:** User logs in successfully.
- **Actual Outcome:** As expected.
- **Status:** ✅ Passed

### TC-003: Invalid Login Attempt

- **Description:** Test login with incorrect credentials.
- **Steps:**
    1. Navigate to the login page.
    2. Enter incorrect username or password.
    3. Click on the "Login" button.
- **Expected Result:** Error message displayed.
- **Actual Outcome:** As expected.
- **Status:** ✅ Passed

## 2. Tweet Functionality

### TC-004: Tweet Creation (Text)

- **Description:** Check if users can post a text tweet.
- **Steps:**
    1. Navigate to the home page.
    2. Enter a text tweet in the tweet box.
    3. Click on the "Post" button.
- **Expected Result:** Tweet is posted successfully.
- **Actual Outcome:** As expected.

- **Status:** ✅ Passed

**TC-005: Tweet Creation (Image)**

- **Description:** Check if users can post a tweet with an image.
- **Steps:**
    1. Navigate to the home page.
    2. Click on the "Add Image" button and upload an image.
    3. Enter optional text and click "Post."
- **Expected Result:** Tweet with an image is uploaded successfully.
- **Actual Outcome:** As expected.
- **Status:** ✅ Passed

**TC-012: Delete a Tweet**

- **Description:** Ensure users can delete their own tweets.
- **Steps:**
    1. Navigate to a previously posted tweet.
    2. Click on the "Delete" button.
    3. Confirm the deletion.
- **Expected Result:** Tweet is removed from the feed.
- **Actual Outcome:** As expected.
- **Status:** ✅ Passed

## 3. User Interactions

**TC-006: Like a Tweet**

- **Description:** Ensure users can like a tweet.
- **Steps:**
    1. Navigate to any tweet.
    2. Click on the "Like" button.
- **Expected Result:** Tweet like count increases.
- **Actual Outcome:** As expected.
- **Status:** ✅ Passed

**TC-007: Unlike a Tweet**

- **Description:** Ensure users can remove their like from a tweet.
- **Steps:**
    1. Navigate to a liked tweet.
    2. Click on the "Unlike" button.
- **Expected Result:** Tweet like count decreases.
- **Actual Outcome:** As expected.
- **Status:** ✅ Passed

**TC-008: Follow a User**

- **Description:** Check if users can follow other users.

- **Steps:**
  1. Navigate to another user’s profile.
  2. Click on the "Follow" button.
- **Expected Result:** User successfully follows another user.
- **Actual Outcome:** As expected.
- **Status:** ✅ Passed

### TC-009: Unfollow a User

- **Description:** Check if users can unfollow another user.
- **Steps:**
  1. Navigate to a followed user’s profile.
  2. Click on the "Unfollow" button.
- **Expected Result:** User successfully unfollows the other user.
- **Actual Outcome:** As expected.
- **Status:** ✅ Passed

## 4. User Profile Management

### TC-010: View Profile

- **Description:** Ensure users can view other users' profiles.
- **Steps:**
  1. Navigate to a user’s profile.
  2. Check if profile details and tweets are displayed.
- **Expected Result:** Profile page displays user details & tweets.
- **Actual Outcome:** As expected.
- **Status:** ✅ Passed

### TC-011: Edit Profile

- **Description:** Check if users can edit their profile details.
- **Steps:**
  1. Navigate to the "Edit Profile" section.
  2. Modify details like username, bio, or profile picture.
  3. Click "Save Changes."
- **Expected Result:** Profile updates successfully.
- **Actual Outcome:** As expected.
- **Status:** ✅ Passed

## 5. Feed & Logout

### TC-013: Feed Display

- **Description:** Ensure users see tweets from followed users.
- **Steps:**
  1. Follow a user.
  2. Navigate to the home feed.
  3. Check if tweets from the followed user appear.

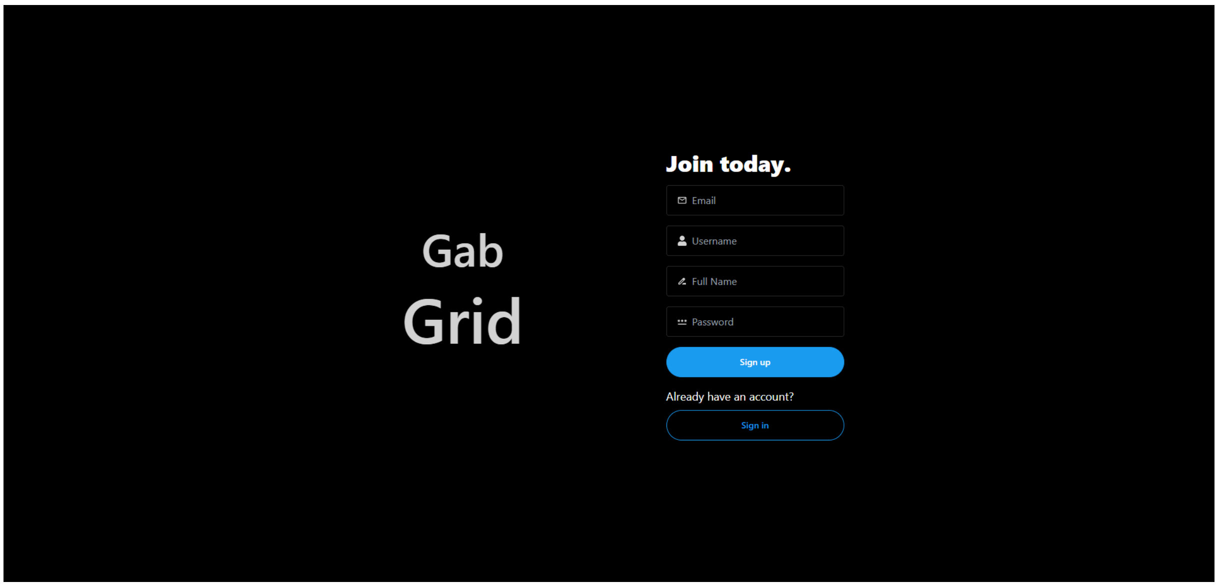
- **Expected Result:** Feed updates dynamically with new tweets.
- **Actual Outcome:** As expected.
- **Status:** ✅ Passed

**TC-014: Logout**

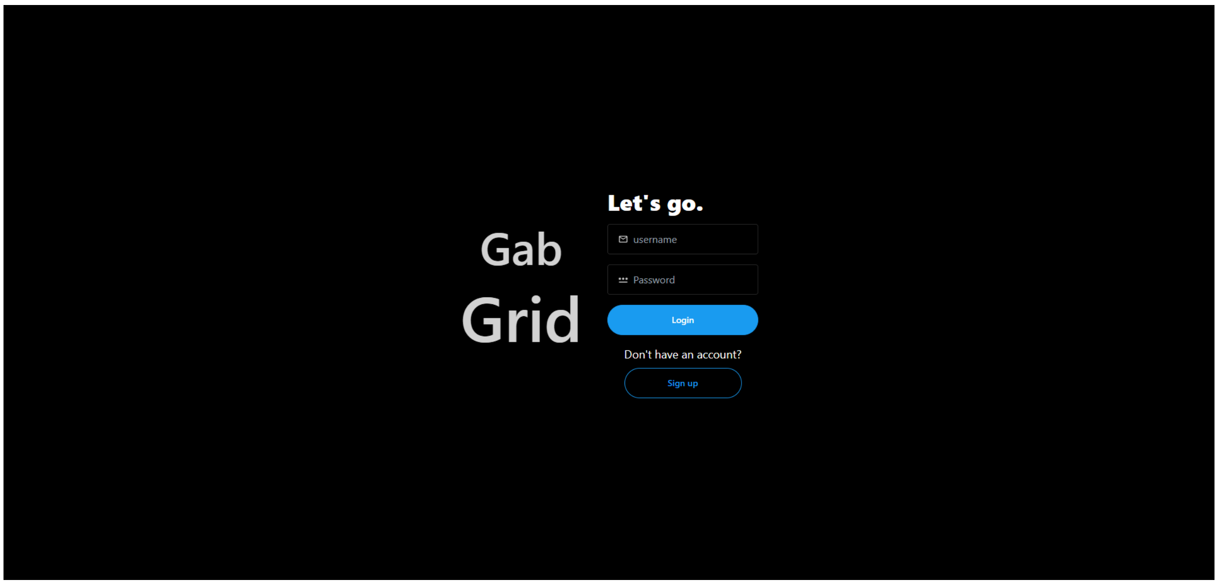
- **Description:** Validate logout functionality.
- **Steps:**
  1. Click on the "Logout" button.
  2. Confirm the action.
  3. Try accessing protected pages.
- **Expected Result:** User is logged out successfully.
- **Actual Outcome:** As expected.
- **Status:** ✅ Passed

## User Documentation:

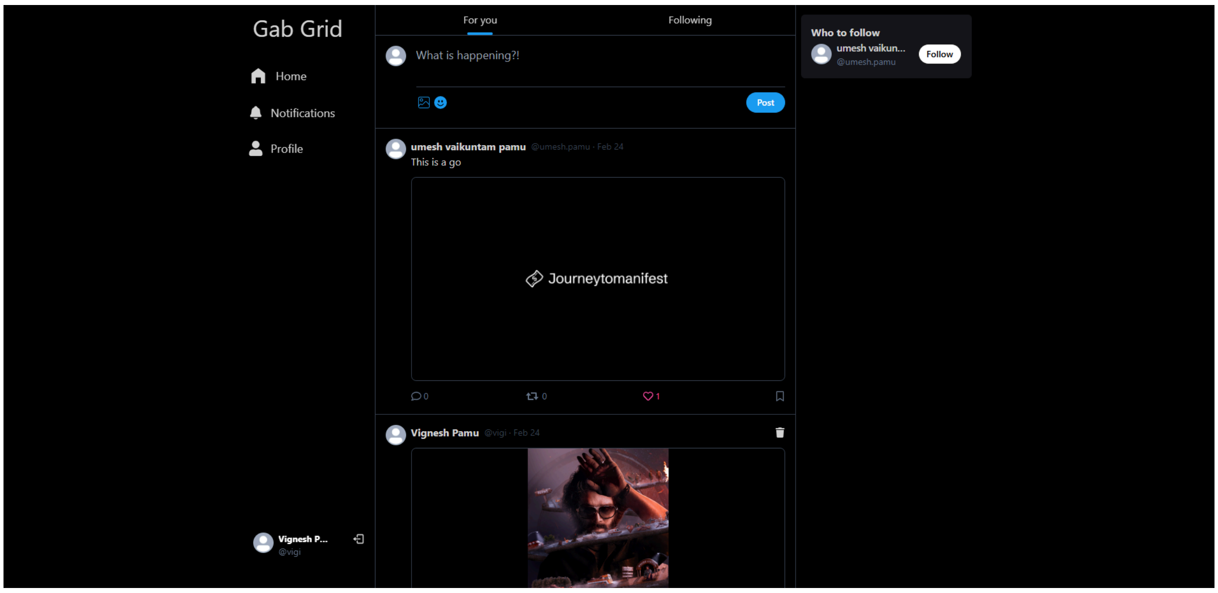
1. SignUp



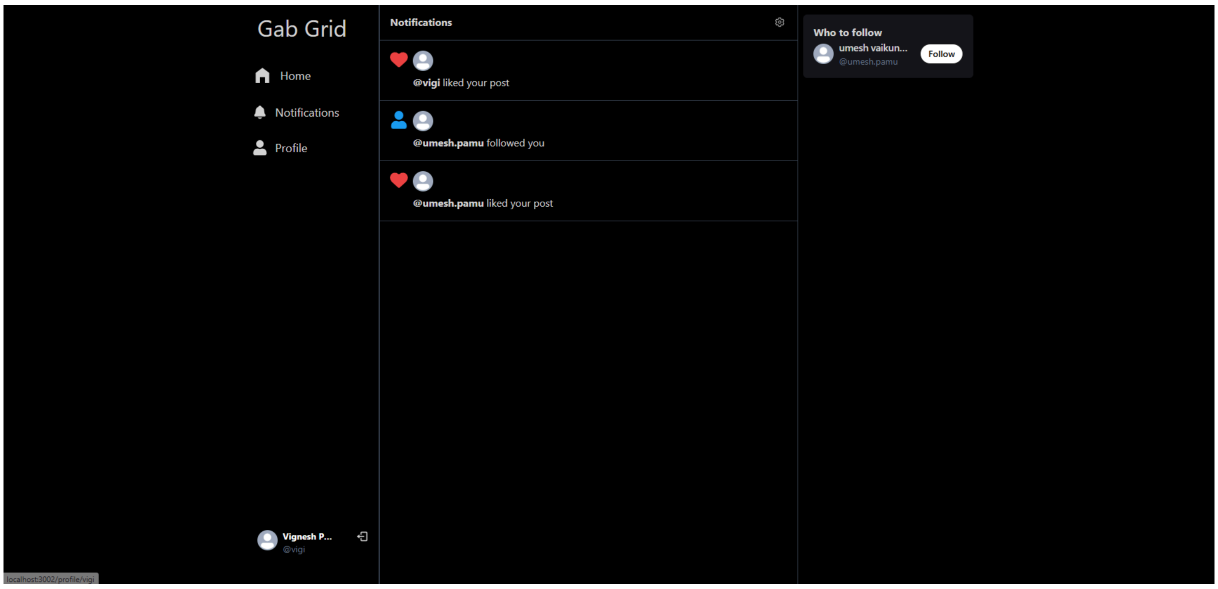
2. SignIn



3. Home



## 4. Notification


Gab Grid

Home

Notifications

Profile

Vignesh P...
@vigi

localhost:3002/profile/vigi

Notifications

@vigi liked your post

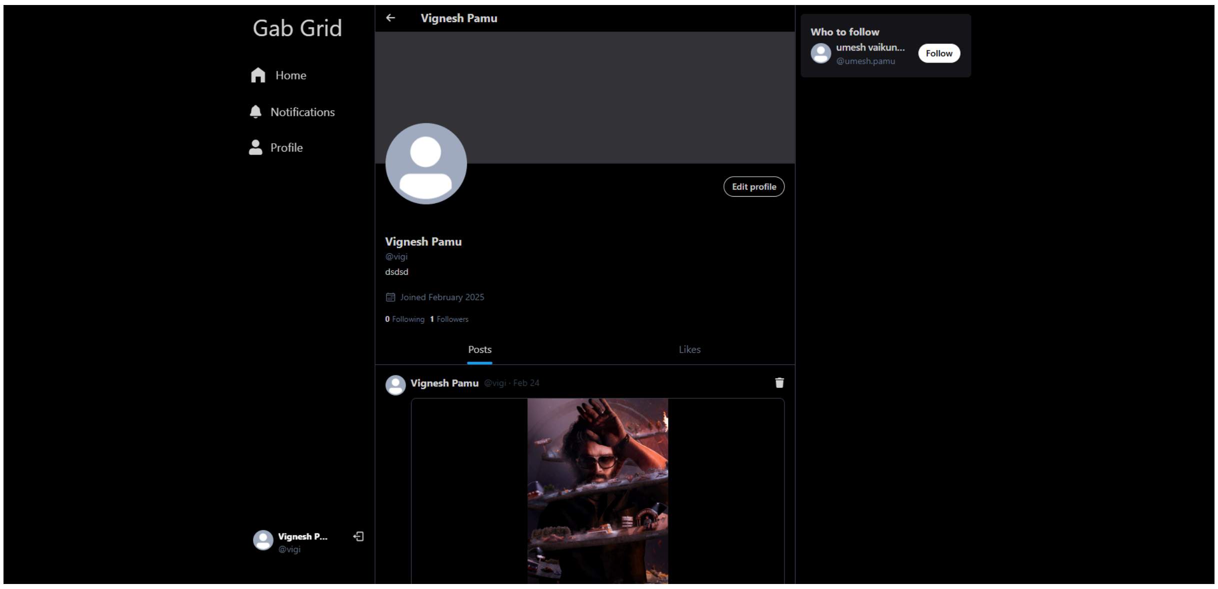
@umesh.pamu followed you

@umesh.pamu liked your post

Who to follow

umesh vaikun...
@umesh.pamu
Follow

## 5. Profile


Gab Grid

Home

Notifications

Profile

Vignesh P...
@vigi

Vignesh Pamu

Edit profile

Vignesh Pamu
@vigi
dsdsd

Joined February 2025

0 Following 1 Followers

Posts Likes

Vignesh Pamu @vigi · Feb 24

Who to follow

umesh vaikun...
@umesh.pamu
Follow

# Chapter 7 - CONCLUSIONS

## Conclusion

GabGrid successfully implements a **social media platform** where users can share thoughts, interact through likes and follows, and engage in a dynamic feed. The **MERN stack** ensures a **scalable, responsive, and real-time** user experience. This project demonstrates proficiency in full-stack development, authentication, database management, and UI/UX design.

## Significance of the System

- Provides a **platform for open expression** through text and image-based tweets.
- Encourages **social engagement** by enabling likes, follows, and interactions.
- Implements **secure authentication** and user data management.
- Offers a **real-time experience** using WebSockets for instant updates.
- Showcases **modern web technologies** and best practices in full-stack development.

## Limitations of the System

- Currently, **no direct messaging** or private communication between users.
- Lacks **content moderation** or AI-driven filtering for inappropriate content.
- Does not support **multi-language** or translation features.
- Limited to **basic engagement features** (no polls, stories, or threaded replies).
- **Scalability constraints** due to the limited infrastructure in the initial deployment.

## Future Scope of the Project

- **Enhancing User Engagement:** Adding features like **commenting, retweets, and threaded conversations**.
- **Direct Messaging:** Implementing a **secure chat system** for private interactions.
- **AI-Based Content Moderation:** Using **NLP and ML algorithms** to detect and filter inappropriate content.
- **Mobile App Integration:** Developing a **React Native app** for seamless mobile access.
- **Monetization & Advertisements:** Exploring **ad-based revenue models** and **premium user subscriptions**.
- **Performance Optimization:** Implementing **server-side rendering (SSR)** for better performance and SEO.