**Ourtube: A Scalable and Secure Media Streaming Platform**

A Project Report submitted in partial fulfilment of the requirements

for the award of the degree of

Bachelor of Technology

in

**Computer Science and Engineering**

By:- Prakhar Agarwal (2115000728)

Group No.: G28

Under the Guidance of **Mrs. Mona Kumari** Assistant Professor

Department of Computer Engineering & Applications

**Institute of Engineering & Technology**



GLA University Mathura- 281406, INDIA December, 2024

## CONTENTS

DECLARATION I

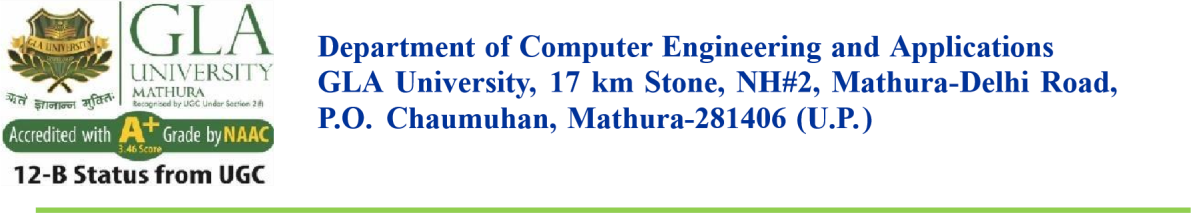
CERTIFICATE II

ACKNOWLEDGEMENT III

ABSTRACT IV

LIST OF FIGURES V

LIST OF TABLES VI



## DECLARATION

I hereby declare that the work which is being presented in the B.Tech. Project “**Our tube: A Scalable and Secure Media Streaming Platform**”, in partial fulfillment of the requirements for the award of the Bachelor of Technology in Computer Science and Engineering and submitted to the Department of Computer Engineering and Applications of GLA University, Mathura, is an authentic record of my own work carried under the supervision of **Ms. Mona Kumari and Assistant Professor .**

The contents of this project report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree.

Sign \_

Name of Student: Prakhar Agrawal

University Roll No.: 2115000726

## CERTIFICATE

This is to certify that the above statements made by the candidate are correct to the best of my/our knowledge and belief.

Shape

Sign - \_\_\_\_\_\_\_\_

**Supervisor**

(Ms. Mona Kumari)

Assistant Professor

Dept. of Computer Engg, & App.

ShapeShape

Sign - \_\_\_\_\_\_\_\_ Sign - \_\_\_\_\_\_\_\_

**Project Co-ordinator ProgramCoordinator (Dr. Mayank Srivastava) (Dr. Nikhil Govil)**

Associate Professor Associate Professor

Dept. of Computer Engg, & App. Dept. of Computer Engg, &App.

Date:

ACKNOWLEDGEMENT

It gives us the immense pleasure to present the report of the B.Tech. Major Project undertaken during B.Tech. 4th Year. This project would never have seen the light of the day without the help and guidance that we have received.

Our heartiest thanks to **Mrs. Mona Kumari, Assistant Professor** for providing us with an encouraging platform to develop this project, which thus helped us in shaping our abilities towards a constructive goal.

We owe special debt of gratitude to **Mrs. Mona Kumari,** for his constant support and guidance throughout the course of our work. He has showered us with all his extensively experienced ideas and has also taught us about the latest industry-oriented technologies. We also do not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind guidance and cooperation during the development of our project.

Sign:- 

Name of Student: Prakhar Agarwal

University Roll No.:21150000726

## ABSTRACT

Ourtube is a media streaming site intended to give users a hassle-free experience for uploading, watching, and managing video files. The overall goal of the project was to create the backend and frontend of the site along with smooth connections between them.

The backend of Ourtube was developed with Node.js, wherein different APIs, routes, models, and middleware were implemented to manage user authentication, video uploading, and data storage. Technologies such as MongoDB for database storage, Cloudinary for hosting media, JWT for secure authentication, and CORS for cross-origin request management were utilized. Postman was used for testing API endpoints to guarantee functionality and reliability.

On the frontend side, React was utilized to create a user-friendly interface, such that there is a seamless experience for users. Axios was included for API requests, such that real-time interactions between the frontend and backend are enabled. The aim was to achieve an intuitive UI in which users can easily stream and upload videos.

One of the biggest challenges in development was maintaining a stable connection between the frontend and backend. CORS issues and API request handling efficiently needed extensive debugging and testing. Deploying the project was another challenge, as making sure the backend and frontend ran smoothly on a live server meant setting up cloud services, database connections, and security settings correctly.

The skills and experience obtained in the process of this project have enhanced my appreciation of full-stack development

CHAPTER 1: Introduction

* 1. Overview and Motivation
  2. Objective
  3. Issues and Challenges
  4. Organization of the Project

CHAPTER 2: Requirement Analysis

* 1. Hardware Requirements
  2. Software Requirements

CHAPTER 3: System Design

* 1. Architecture Overview
  2. Database Schema Design
  3. RESTful API Design
  4. Frontend Workflow

CHAPTER 4: Implementation and User Interface

* 1. Backend Development (Express.js and MongoDB)
  2. Frontend Development (React.js)
  3. Integration of Frontend and Backend
  4. User Interface Design and Features

CHAPTER 5: Testing, Debugging, and Deployment

* 1. Unit Testing and Tools (Jest, Supertest)
  2. Debugging Techniques and Challenges
  3. Deployment Workflow (CI/CD, Hosting on Heroku and Vercel)

CHAPTER 6: Conclusion and Summary

* 1. Conclusion
  2. Summary

References

***Chapter 1***  ***Introduction***

## Overview and Motivation:

The "Watch Pool" project focuses on creating a full-stack web application using the MERN (MongoDB, Express.js, React.js, Node.js) stack, which is widely adopted for developing scalable, responsive, and modern web applications. By adopting an open-source approach, the project promotes collaboration with the developer community and encourages continuous feedback and innovation.

Motivations for this project include:

* + - Leveraging modern tools for educational purposes.
    - Demonstrating the effectiveness of community-driven software development.
    - Building a scalable and feature-rich platform.

The project name, *Watch Pool*, reflects the collaboration and pooling of resources, data, and community efforts to create a unified, user-friendly application.

## Objectives Key objectives of the project are:

1. Develop a Full-Stack Web Application: Utilize the MERN stack to create a seamless user experience with dynamic content handling and real-time updates.
2. Encourage Open-Source Contributions: Make the project publicly available on GitHub to allow external developers to enhance and expand its features.
3. Establish a CI/CD Workflow: Automate deployment, testing, and updates using tools like GitHub Actions, Heroku, and Vercel.

## Issues and Challenges: -

## Throughout the development of the project, several challenges were encountered across technical, community, and deployment dimensions:

**The project faces the following challenges:**

1. **Technical Challenges:**
   * **Scalable Architecture Design:**  
      Architecting the application to handle growth in users, data, and traffic required careful planning of database schema, API performance, and frontend rendering logic. Balancing performance with maintainability was a key concern.
   * **Integration of RESTful APIs with React.js:**  
      Connecting backend APIs to the React frontend involved complexities such as state management, asynchronous data fetching, error handling, and ensuring a smooth user experience during dynamic content updates.
2. **Community Challenges:**
   * **Managing External Contributions:**  
      As an open-source project, maintaining a structured workflow for pull requests, issue tracking, and collaboration posed difficulties. Clear contribution guidelines and active issue triaging were necessary to avoid bottlenecks and confusion.
   * **Maintaining Code Quality and Consistency:**  
      Ensuring that code submitted by contributors adhered to established standards (such as naming conventions, file structures, and testing protocols) was crucial to avoid technical debt and maintain a cohesive codebase.
3. **Deployment Challenges:**
   * **Scalable Architecture Design:**  
      Architecting the application to handle growth in users, data, and traffic required careful planning of database schema, API performance, and frontend rendering logic. Balancing performance with maintainability was a key concern.
   * **Integration of RESTful APIs with React.js:**  
      Connecting backend APIs to the React frontend involved complexities such as state management, asynchronous data fetching, error handling, and ensuring a smooth user experience during dynamic content updates.
   * **Deploying on Render:**  
      Deploying the backend and frontend on Render introduced unique configuration requirements, such as proper build and start commands, environment variable setup, and handling cold starts. Additionally, troubleshooting runtime issues specific to Render’s infrastructure demanded extra attention during the deployment phase.
   1. **Organization of the Project**

This report is structured into six comprehensive chapters, each addressing a critical phase of the project’s development lifecycle:

**Chapter 1: Introduction, Motivation, Objectives, and Challenges**

This chapter provides an overview of the project, including its purpose, background, and real-world relevance. It outlines the primary objectives of the project, details the motivation behind choosing this particular topic, and discusses the major technical, community, and deployment challenges faced during development.

**Chapter 2: Requirement Analysis**

This chapter identifies and analyzes both hardware and software requirements essential for developing and running the application. It includes details on development tools, frameworks, libraries, and the minimum system specifications needed for optimal performance.

**Chapter 3: System Design**

This chapter elaborates on the architectural structure of the system. It includes the overall system architecture, database schema design, data flow diagrams, and the structure of RESTful APIs. The goal is to provide a clear blueprint of how various components interact and how data is managed throughout the application.

**Chapter 4: Implementation**

This section focuses on the actual development of the project. It covers the backend development using Node.js and Express, frontend development using React, and the integration of MongoDB for data storage. Additionally, it highlights UI design considerations, key features implemented, and notable code-level insights.

**Chapter 5: Testing and Deployment**

This chapter discusses the strategies and tools used for testing and debugging the application to ensure stability and performance. It also details the CI/CD process, including automated workflows using GitHub Actions and deployments on platforms such as Heroku, Vercel, and Render. Common deployment issues and their resolutions are also covered.

**Chapter 6: Summary and Conclusion**

The final chapter summarizes the key outcomes of the project and reflects on the lessons learned. It evaluates the project’s success in achieving its stated objectives and outlines potential future enhancements or research directions.

## CHAPTER 2: REQUIREMENT ANALYSIS

To ensure the smooth development and deployment of the *Ourtube* web application, both hardware and software requirements were carefully assessed and selected. This chapter outlines the necessary development environment, tools, libraries, and platforms used throughout the lifecycle of the project.

* 1. **Hardware Requirements**
* **Development Machines**

The application was developed on personal computing devices capable of handling modern development workloads, especially for compiling, rendering, and testing full-stack applications.

|  |  |
| --- | --- |
| **Component** | **Specification** |
| **RAM** | Minimum 8 GB |
| **Processor** | Intel i5 (7th Gen or later) / AMD Ryzen 5 |
| **Storage** | Minimum 500 GB (preferably SSD) |
| **O.S** | Windows 10/11, macOS, or Linux (Ubuntu 20.04+) |

* + - **Cloud Hosting Services:**

Ourtube was deployed using modern cloud platforms to ensure high availability, performance, and ease of deployment.

* + - * **Render:** Used to host the backend (Node.js/Express) with automatic scaling and HTTPS support.
      * **Vercel/Netlify:** Used to deploy the frontend (React.js), allowing seamless continuous deployment from GitHub.
      * **MongoDB Atlas:** Used to host the cloud-based NoSQL database.
      * **Postman:** API testing platform used during backend development.
  1. **Software Requirements**

The software stack used in Ourtube development was based on the MERN architecture: **MongoDB, Express.js, React.js, and Node.js**, supported by various tools for development, testing, version control, and deployment.

1. **Backend Tools:**

|  |  |
| --- | --- |
| **Tool** | **Purpose** |
| **Node.js** | JavaScript runtime environment for executing server-side code |
| **Express.js** | Web framework for building RESTful APIs |
| **MongoDB** | NoSQL database for storing user, video, and comment data |
| **Mongoose** | Object Data Modeling (ODM) tool for interacting with MongoDB databases |

1. **Frontend Tools:**

|  |  |
| --- | --- |
| **Tool** | **Purpose** |
| **React.js** | Frontend library for building reusable UI components |
| **Axios** | Promise-based HTTP client for communicating with backend APIs |
| **React Router** | Used for client-side routing to build a single-page application (SPA) |

1. **Version Control:**

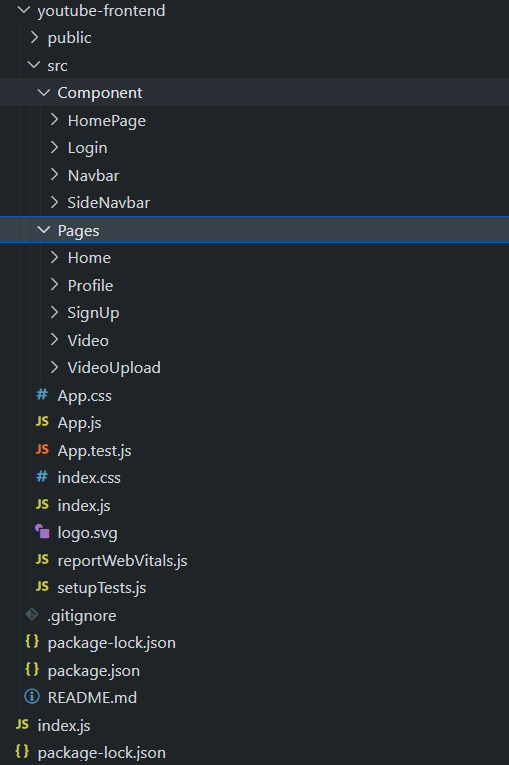
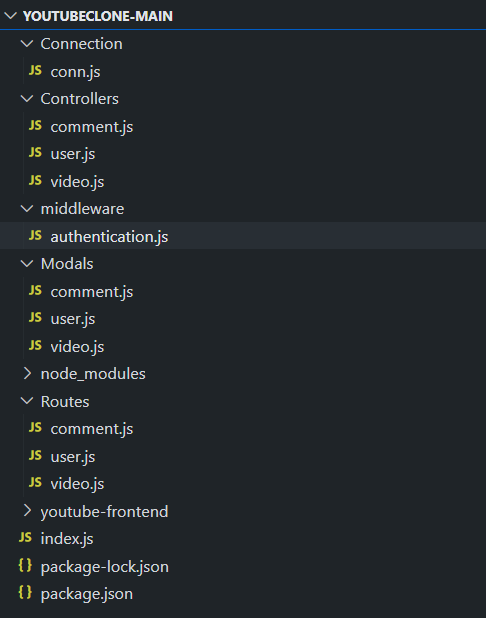
|  |  |
| --- | --- |
| **Tool** | **Purpose** |
| **Git** | Distributed version control for managing source code |
| **GitHub** | Remote repository for collaboration, issue tracking, and CI/CD hooks |

1. **Testing Tools:**

|  |  |
| --- | --- |
| **Tool** | **Usage** |
| **React-Router-Dom** | Enables client-side routing in a React application to create a single-page application (SPA) experience. |
| **Postman** | API testing tool used to send HTTP requests and test the behavior of RESTful APIs. |

1. **CI/CD Tools:**

|  |  |
| --- | --- |
| **Tool** | **Functionality** |
| **GitHub Actions** | Automated testing and build execution on every push or pull request |
| **Heroku / Vercel** | Platform-as-a-Service used for seamless deployment pipelines |



**CHAPTER 3: SYSTEM DESIGN**

* 1. **Architecture Overview**

The project architecture is structured in three layers: frontend, backend, and database. Each layer is independently managed to ensure maintainability, scalability, and separation of concerns.

1. **Frontend Layer**

The frontend is developed using React.js, offering a rich, interactive, and responsive user experience. It handles user interface presentation, client-side routing, and communicates seamlessly with the backend.

**Key Features:**

* Component-Based Structure: The UI is organized into reusable components like Navbar, Home, Video, Profile, VideoUpload, and SignUp.
* Routing with React Router: Enables smooth single-page navigation (SPA) without full page reloads using react-router-dom.
* API Communication with Axios: Data is fetched and posted to the backend using Axios, including user authentication, video upload, and comments.
* Cloudinary Integration: Used for secure and optimized image and video hosting to handle media files efficiently.
* Responsive Design: Built with responsiveness in mind to ensure mobile and desktop compatibility.
  + - **Backend Layer:**

The backend is built using **Node.js** with the **Express.js** framework, which handles all the business logic, routes, and interactions between the frontend and the database.

#### **Core Functionalities:**

* **Modular Express Routes & Controllers**:
  + /api/user – Handles signup, login, user profiles.
  + /api/video – Manages video uploads, viewing, likes, dislikes.
  + /api/comment – Handles comment posting and retrieval.
* **Authentication & Authorization**:
  + Secured using **JWT (JSON Web Tokens)** for user sessions.
  + Role-based access to distinguish between students, teachers, and admins.
* **CORS Configuration**:
  + **Cross-Origin Resource Sharing** is enabled to allow frontend and backend to interact across different domains during development and deployment.

**Database Layer:**

The **MongoDB** database is used as a flexible NoSQL storage solution to manage application data. It is integrated using **Mongoose**, which provides schema-based modeling and validation.

#### **Database Design:**

* **Users Collection**:
  + Fields: channelName, userName, password, about, profilePic, timestamps.
  + Secure password storage and uniqueness constraints.
* **Videos Collection**:
  + Fields: user, title, description, videoLink, thumbnail, videoType, like, dislike, timestamps.
  + References the user collection to associate each video with an uploader.
* **Comments Collection**:
  + Fields: user, video, message, timestamps.
  + Allows user discussions under video content.
  1. **Database Schema Design**

Database collections and their fields include:

* + - **Users Collection:** Stores information about registered users.

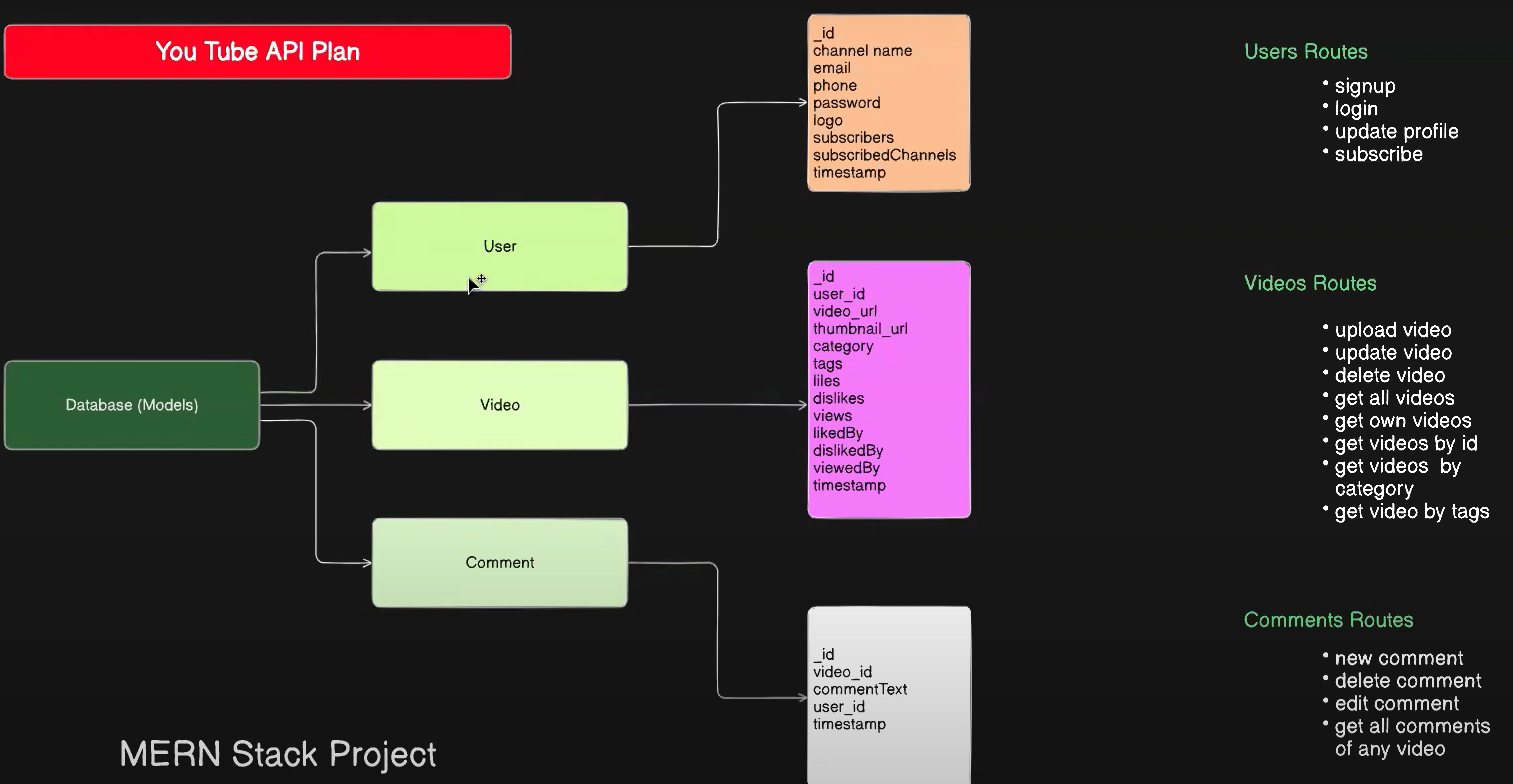
|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| channelName | String | Name of the user's channel |
| userName | String | Unique username |
| password | String | Hashed password |
| about | String | Short user description |
| profilePic | String | URL of profile picture |
| createdAt / updatedAt | Date | Auto-generated timestamps |

* + - **Video Collection:-** Contains all video-related metadata.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| user | ObjectId (ref: 'user') | Reference to uploader |
| title | String | Title of the video |
| description | String | Video description |
| videoLink | String | Cloudinary-hosted video URL |
| thumbnail | String | Cloudinary-hosted thumbnail |
| videoType | String | Category (e.g., Lecture, Club) |
| like | Number | Number of likes |
| dislike | Number | Number of dislikes |
| createdAt / updatedAt | Date | Auto-generated timestamps |

* **Comment Collection-** Stores user-generated comments under each video.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| user | ObjectId (ref: 'user') | Commenter reference |
| video | ObjectId (ref: 'video') | Related video reference |
| message | String | Comment text |
| createdAt / updatedAt | Date | Auto-generated timestamps |



* 1. **RESTful API Design**

The backend exposes RESTful APIs to manage users, videos, and comments. Below are key endpoints grouped by feature:

* + - **User APIs**

|  |  |  |
| --- | --- | --- |
| **Method** | **Endpoint** | **Description** |
| POST | /api/auth/register | Register a new user |
| POST | /api/auth/login | User login |
| GET | /api/user/:id | Fetch user profile |

#### **Video APIs**

|  |  |  |
| --- | --- | --- |
| **Method** | **Endpoint** | **Description** |
| POST | /api/video/upload | Upload a new video |
| GET | /api/video/all | Fetch all videos |
| GET | /api/video/:id | Get details of a specific video |
| PUT | /api/video/:id/like | Like a video |
| PUT | /api/video/:id/dislike | Dislike a video |
| DELETE | /api/video/:id | Delete a video |

#### **Comment APIs**

|  |  |  |
| --- | --- | --- |
| **Method** | **Endpoint** | **Description** |
| POST | /api/comment/add | Add a comment to a video |
| GET | /api/comment/:videoId | Fetch comments for a video |

* 1. **Frontend Workflow**

The frontend of the Ourtube application is built using **React.js**, ensuring a responsive and interactive user experience. The app follows a **component-based architecture** with client-side routing managed by react-router-dom. The application includes dynamic data rendering, conditional layouts, and efficient API integration via Axios.

#### **Main Frontend Architecture**

* **App Entry Point:** The App.js component serves as the main entry point. It includes the navigation bar and defines routes for various pages such as Home, Video Watch, Profile, Upload, and Signup.
* **State Management:** useState and useEffect hooks are used to manage UI state, such as toggling the sidebar and handling API calls.

#### **Routing Structure-**

Routing is implemented using react-router-dom. Below are the defined paths:

|  |  |  |
| --- | --- | --- |
| **Path** | **Component** | **Functionality** |
| **/** | Home | Displays a list of all available videos. |
| **/watch/:id** | Video | Opens a specific video with playback, likes, and comments. |
| **/user/:id** | Profile | Shows a user profile with uploaded videos  . |
| **/:id/upload** | VideoUpload | Upload new videos to the platform. |
| **/signup** | SignUp | Allows new users to register. |

**Component Overview-**

|  |  |
| --- | --- |
| **Component** | **Description** |
| **Navbar** | A persistent top navigation bar that includes branding, links, and toggle sidebar. |
| **Home** | Displays thumbnails and titles of videos fetched from the backend. |
| **Video** | Displays the selected video using a media player, along with comments and actions. |
| **Profile** | Displays channel information, profile picture, and user-uploaded videos. |
| **VideoUpload** | Allows users to upload new videos, enter metadata (title, description), and set thumbnails  . |
| **SignUp** | Registration page for new users with fields for username, password, and profile image. |

# CHAPTER 4: IMPLEMENTATION AND USER INTERFACE

* 1. **Backend Development**

The backend of the *Ourtube* platform is built using Node.js and Express.js, providing a RESTful API for communication with the frontend. Below is a detailed breakdown of the backend structure:

#### **1. Server Setup**

* **Node.js** serves as the runtime environment for running JavaScript on the server.
* **Express.js** is used as the web framework for handling HTTP requests, defining routes, and managing middleware.
* The **server** is structured with modular routes and controllers for easy maintenance and scalability.

Example of server setup:

var express = require("express");

var app = express();

var port = 4000

const cookieParser = require('cookie-parser');

const cors = require('cors');

app.use(cors({

origin: '<http://localhost:3000>', // Your React app's URL

credentials: true

}))

app.use(express.json());

app.use(cookieParser());

require('./Connection/conn');

const AuthRoutes = require('./Routes/user');

const VideoRoutes = require('./Routes/video');

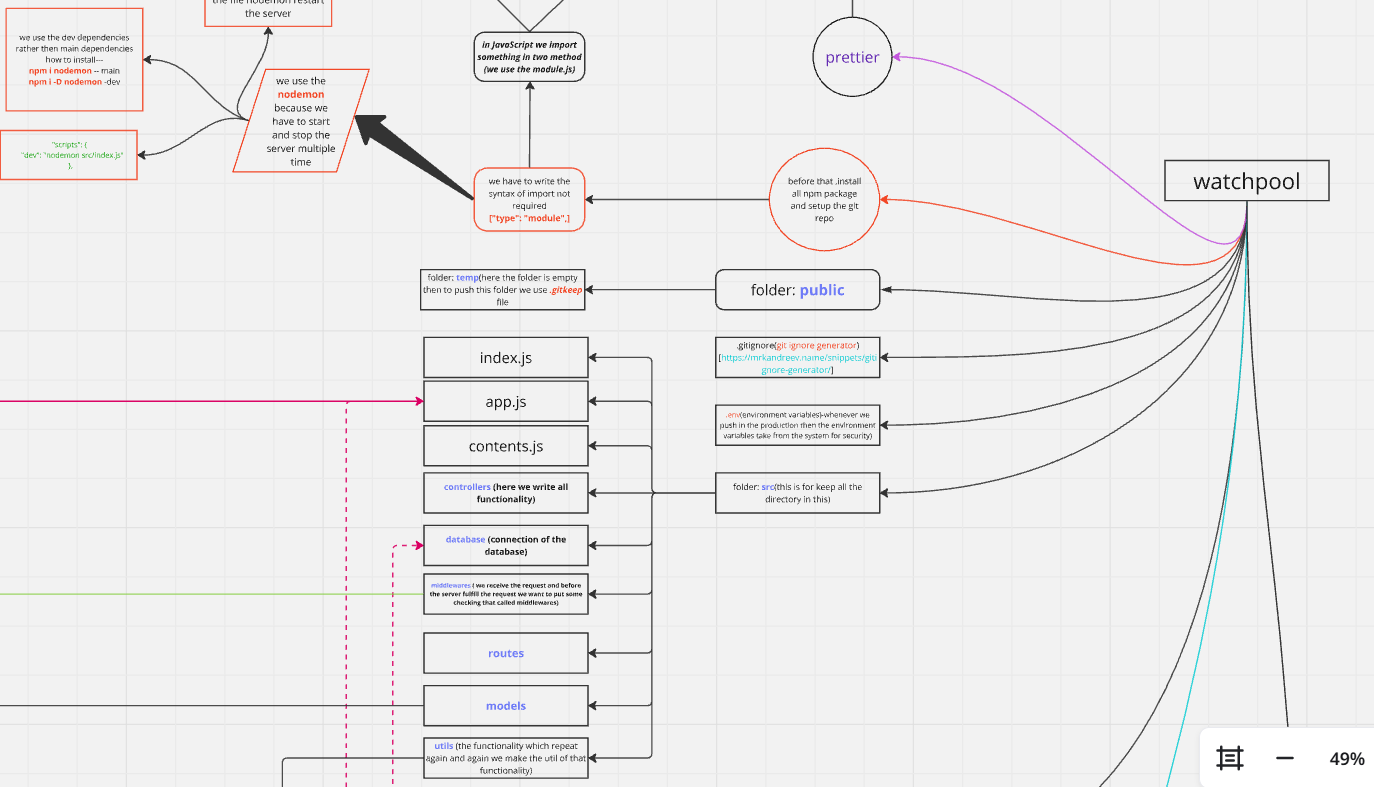
const CommentRoutes = require('./Routes/comment');

app.use('/auth',AuthRoutes);

app.use('/api',VideoRoutes);

app.use('/commentApi',CommentRoutes);

app.listen(port,()=>{console.log("Our backend project is running on Port 4000")});



* + - **Database Connectivity:** The MongoDB database is integrated using Mongoose to handle data models and ensure efficient data manipulation.
* Mongoose schemas define the structure for user, video, and comment data.
* MongoDB stores dynamic content, such as user profiles, video metadata, and comments, allowing for flexible and scalable data storage.

const jwt = require('jsonwebtoken');

const User = require('../Modals/user');

const auth = async (req, res, next) =>{

const token = req.cookies.token;

if(!token){

return res.status(401).json({ error: 'No token, authorization denied' });

}else{

try{

const decode = jwt.verify(token, "Its\_My\_Secret\_Key");

req.user = await User.findById(decode.userId).select('-password');

next();

}catch(err){

res.status(401).json({ error: 'Token is not valid' });

}

}

}

module.exports = auth;

**Middleware:**

JWT (JSON Web Tokens) is implemented as middleware to secure routes that require user authentication. It ensures only authorized users can access specific resources like uploading videos or commenting.

* JWT Authentication is used for both login and signup.
* When a user logs in, the server generates a JWT token and sends it to the frontend for subsequent requests.
* A middleware function is used to validate the token on protected routes.

const jwt = require('jsonwebtoken');

const authenticateJWT = (req, res, next) => {

const token = req.header('Authorization');

if (!token) return res.status(403).send('Access denied');

jwt.verify(token, process.env.JWT\_SECRET, (err, user) => {

if (err) return res.status(403).send('Invalid Token');

req.user = user;

next();

});

};

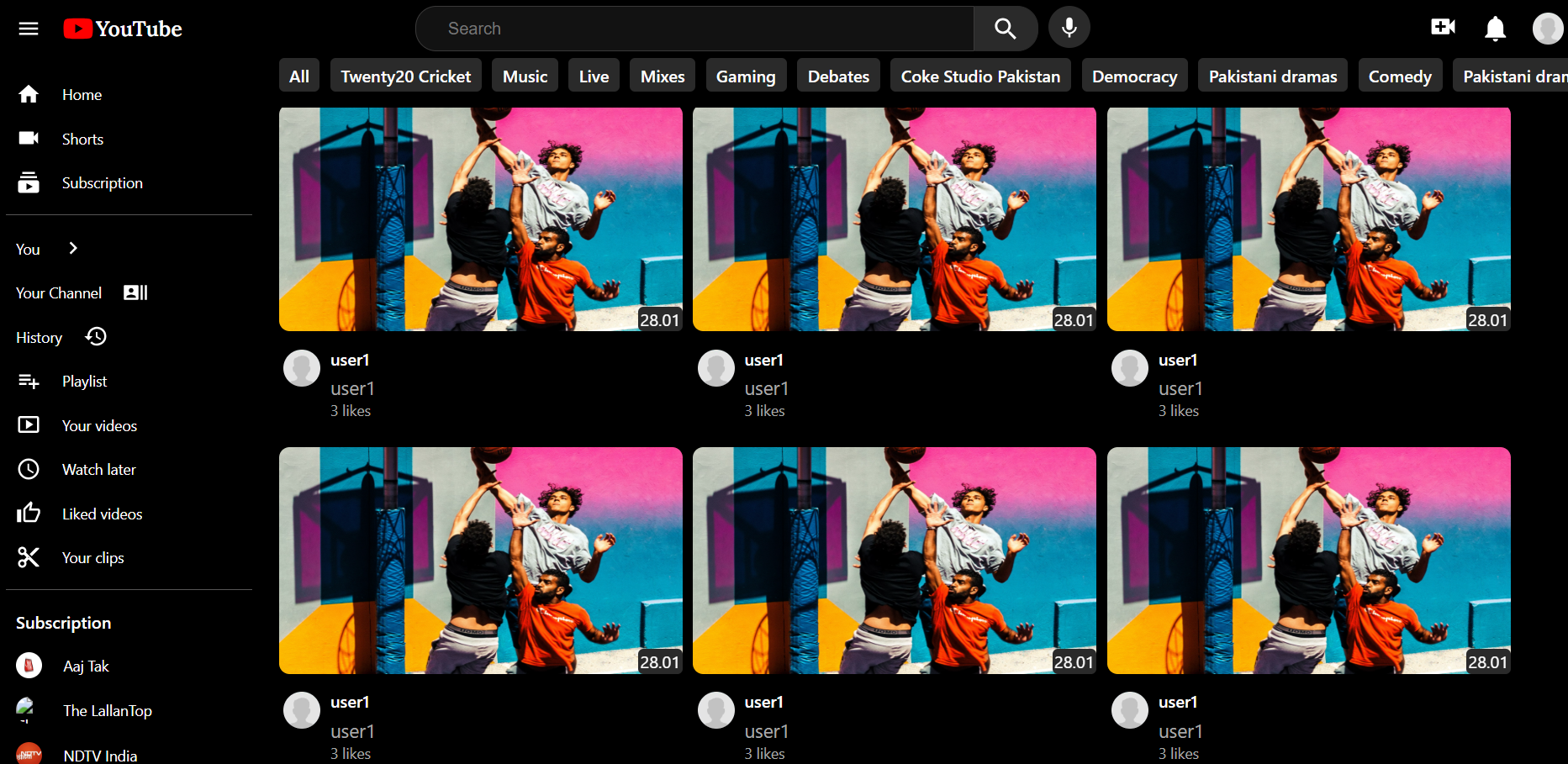
* 1. **Frontend Development-**

The frontend of the *Ourtube* platform is built using React.js, offering a modular, responsive, and dynamic user interface. The structure is component-driven and interacts with the backend using Axios to perform CRUD operations and manage authentication.

**React.js components:**

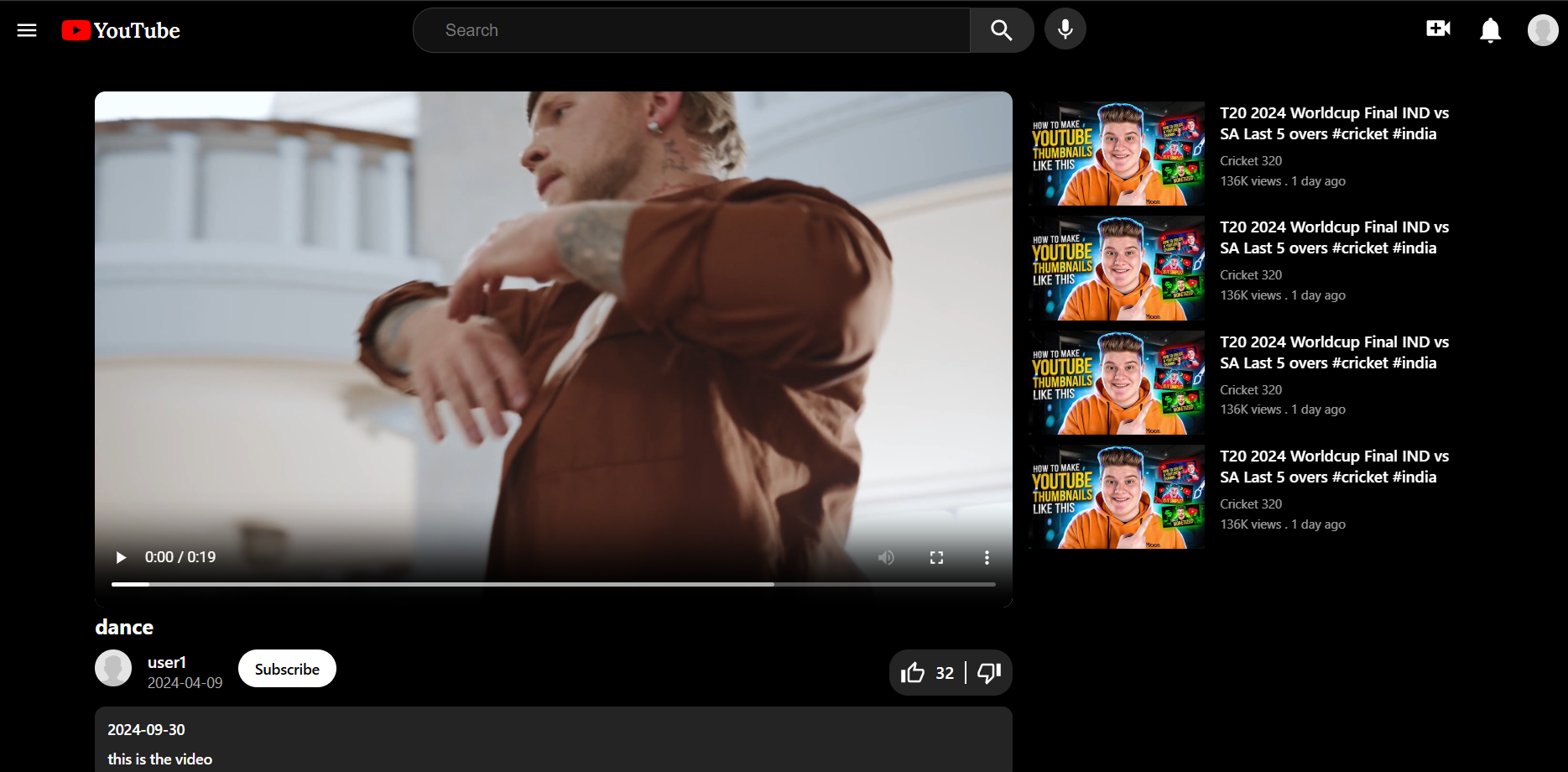
##### **Home Page (Home.js)**

* Displays a list of available videos dynamically by fetching them from the backend via an API call.
* Users see recommended videos, trending content, and featured uploads.
* Videos are rendered in card/grid format.



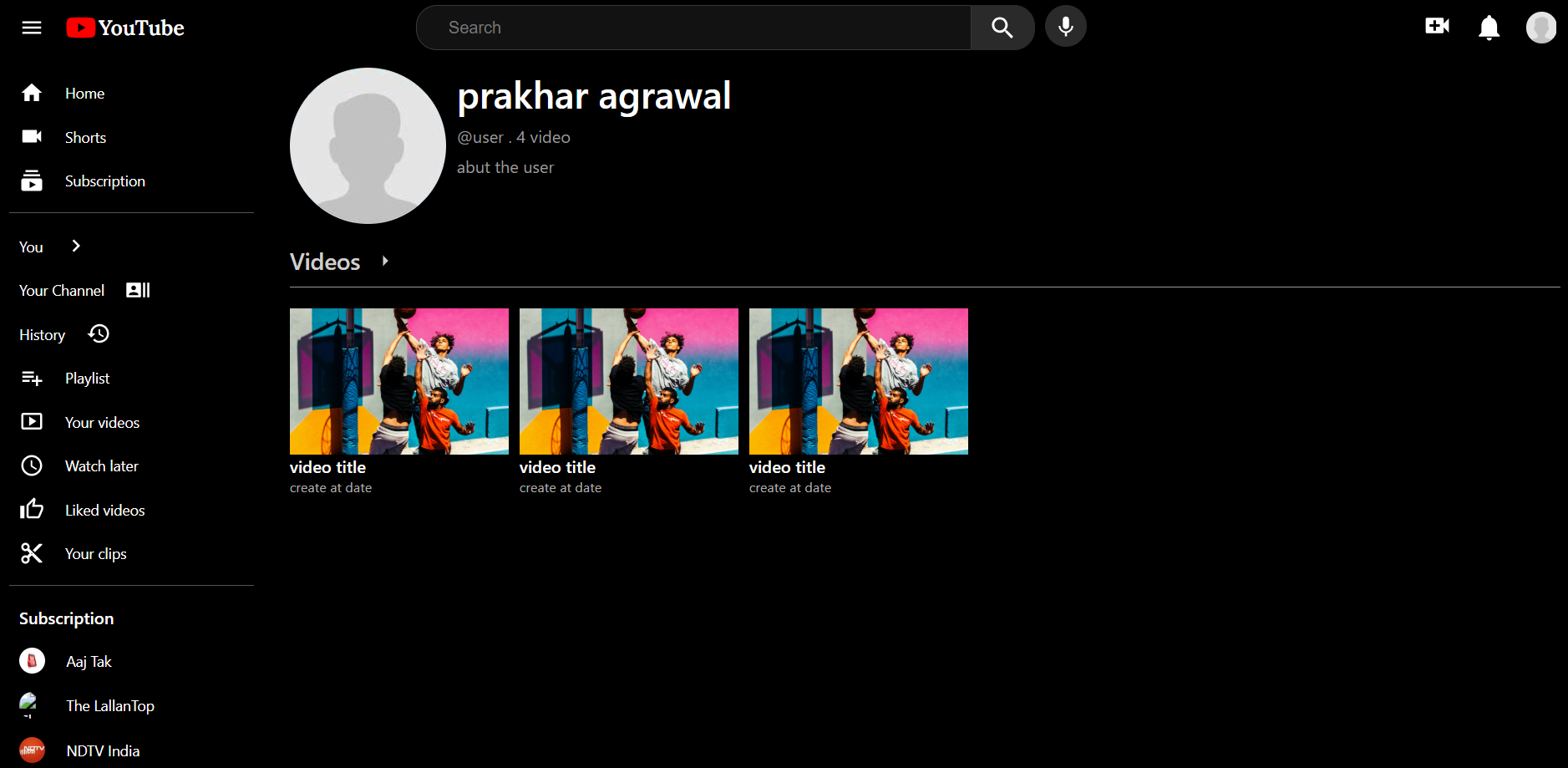
##### **Video Watch Page (Video.js)**

* Displays a selected video using the :id from the URL.
* Supports video playback, likes/dislikes, and comments.
* All content is fetched from the backend using Axios.
* Interactions such as commenting are tied to user authentication.



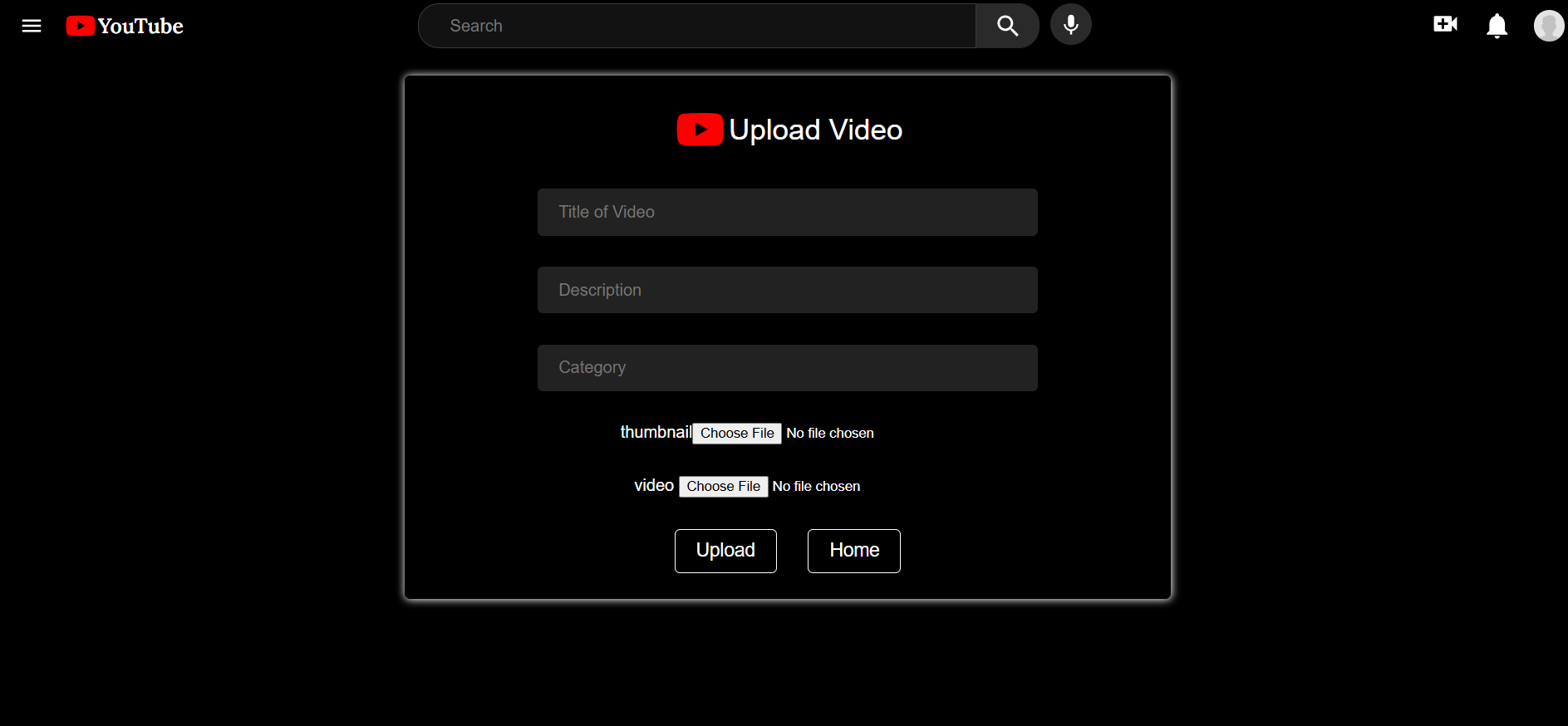
##### **Profile Page (Profile.js)**

* Shows user details, uploaded videos, and about information.
* Uses userId from local storage to fetch user-specific data.
* Side navigation bar visibility is controlled using the sideNavbar prop.



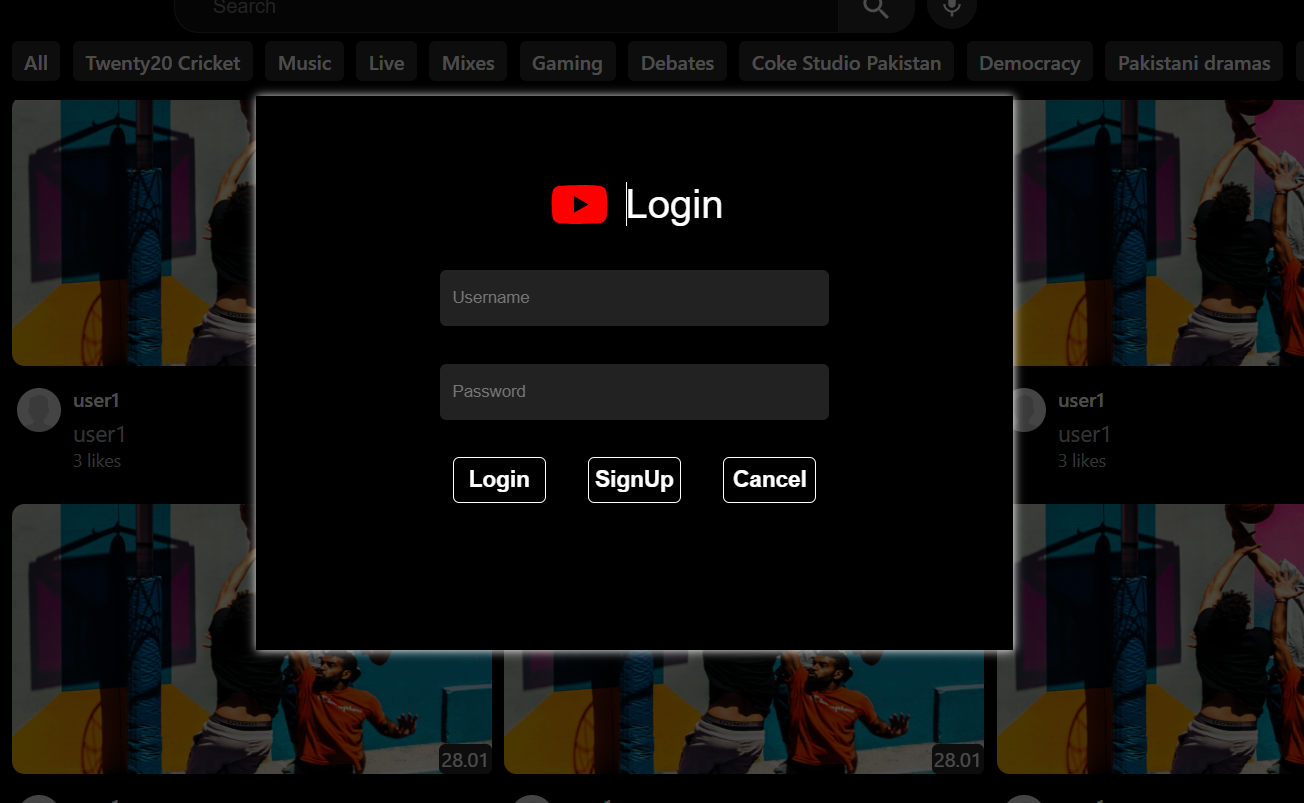
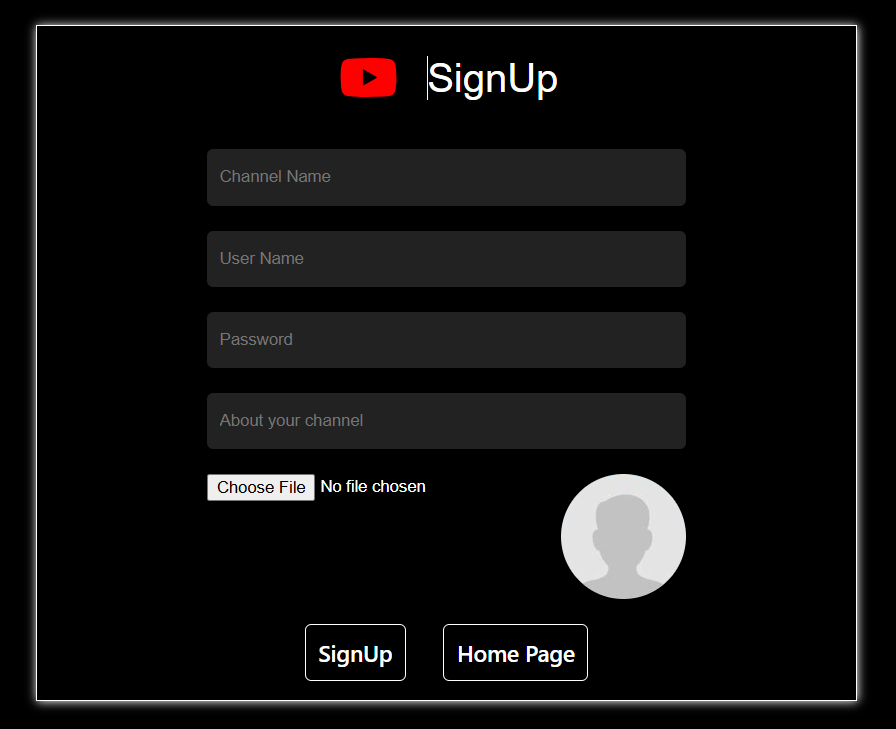
##### **Video Upload Page (VideoUpload.js)**

* Authenticated users can upload videos along with metadata (title, description, tags, thumbnails).
* Integrates with Cloudinary for uploading media files.



##### **Signup Page (SignUp.js)**

* Allows users to register an account with profile information including username, password, and profile picture.
* Data is validated and sent to the backend via an Axios POST request.



* 1. **Integration of Frontend and Backend**

Axios is used to fetch data from APIs, ensuring seamless integration between the frontend and backend.

* GET Requests: Used to retrieve videos, user profiles, and video details.
* POST Requests: Used for sign-up, login, video uploads, and posting comments.
* Logout Functionality: Implemented using a POST request to /auth/logout, and localStorage is cleared to remove session data.
  1. **User Interface Design and Features**

1. **User Interface Design**

The application’s UI is designed with attention to responsiveness, accessibility, and modern design principles.

* **Responsive Layout**: Utilizes CSS flexbox and media queries to adjust layout across mobile, tablet, and desktop devices.
* **Toggleable Sidebar**: Navbar.js includes a hamburger menu icon (MenuIcon) that toggles the visibility of a sidebar using state (sideNavbar).
* **Dynamic Content Rendering**: All dynamic data such as videos, comments, and profile details are rendered after being fetched, without reloading the page.
* **Interactive Icons**: Material UI icons (e.g., VideoCallIcon, NotificationsIcon) enhance usability and provide intuitive interaction points.

**2. User Interface Features:**

**Responsive Design**

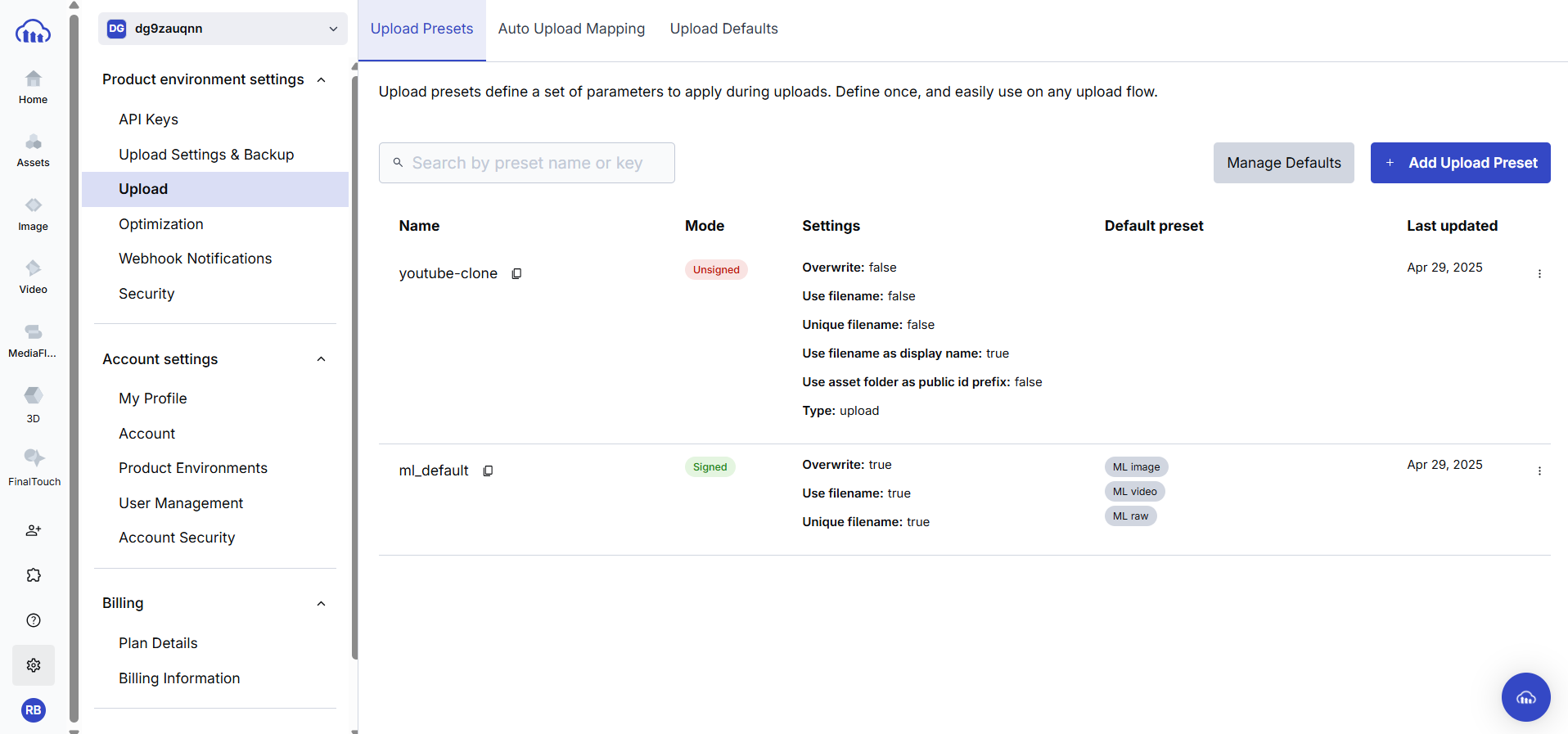
* The layout automatically adjusts based on screen size.
* The sidebar can be toggled using the hamburger menu icon on smaller screens, improving usability on mobile devices.

#### **Authentication Awareness**

* The Navbar checks for login status using localStorage and conditionally displays either the profile avatar or login prompt.
* Logout clears local data and triggers a backend call to invalidate the session.

#### **Media Upload**

* Users can upload video files through the upload form.
* Integration with **Cloudinary** allows videos and thumbnails to be uploaded and retrieved via secure URLs.



#### **Seamless Navigation**

* Navigation is managed using **React Router**, allowing users to move between pages like Home, Watch, Upload, and Profile without reloading the browser.
* URL parameters (e.g., /watch/:id, /user/:id) are used to fetch specific content dynamically.

CHAPTER 5: TESTING, DEBUGGING, AND DEPLOYMENT

* 1. **Unit Testing and Tools: -** To ensure the reliability and correctness of both frontend and backend components, a combination of modern JavaScript testing tools and practices were used.
     + **Backend Testing:**
* **Tool Used**: **Postman**
* **Purpose**: Manually test and validate RESTful API endpoints.
* **Coverage**:
  + Verified correct responses and HTTP status codes for routes like:
    - POST /auth/signup and POST /auth/login – User registration and login
    - GET /api/allVideo – Retrieve all videos
    - POST /api/video/upload – Upload new videos
    - POST /api/comment – Add comments
  + Ensured proper handling of authentication with cookies and JWT tokens in protected routes.
  + Simulated various edge cases (e.g., missing fields, invalid video IDs).
    - **Frontend Testing:**
* **Tool Used**: Manual and browser-based testing using React's built-in dev tools.
* **Techniques**:
  + Verified conditional rendering based on login state (e.g., Navbar profile dropdown).
  + Ensured correct routing with React Router for paths like /watch/:id, /signup, and /user/:id.
  + Manually tested video upload and comment submission flows using the actual UI.
  1. **Debugging Techniques and Challenges**

Robust debugging was critical due to the dynamic nature of the application and integration between the frontend, backend, and database.

* + - **Frontend Debugging:**
* **Tools Used**: Chrome DevTools, React Developer Tools
* **Methods**:
  + Monitored component state transitions in Navbar, Home, and Video pages.
  + Inspected API request payloads and responses in the Network tab for Axios calls.
  + Used local Storage inspection and console logs to manage and troubleshoot user sessions.
  1. **Backend Debugging:**
* **Tools Used**: Console logging in Node.js and Express
* **Techniques**:
  + Inserted console.log () statements in route handlers and controllers to trace request flow.
  + Verified database queries using Mongoose logs.
  + Logged JWT validation results and user session details for authentication routes.
  1. **Deployment Workflow**

**Backend:**

* **Platform**: **Render, Cloudinary**
* **Details**:
  + The backend server, built with Node.js and Express.js, is deployed on Render.
  + Environment variables such as MongoDB URI, JWT secret, and Cloudinary API keys are securely configured via Render’s environment settings.
  + CORS (Cross-Origin Resource Sharing) is enabled to allow communication from the frontend hosted on Vercel.
  + Render provides automatic redeployment on code updates and handles server restarts in case of downtime.

#### **Frontend:**

* **Code Repository**: **GitHub**
* **Deployment Platform**: **Vercel**
* **Details**:
  + The frontend, built with React.js, is hosted on GitHub and deployed through Vercel.
  + Vercel integrates with GitHub for automatic deployments upon push to the main branch.
  + Environment variables such as the backend API URL are securely managed within Vercel’s project settings.
  + Vercel provides fast, global CDN delivery, automatic HTTPS, and zero-config builds for optimal performance.

CHAPTER 7 : CONCLUSION AND SUMMARY

Conclusion

The **"Ourtube"** project showcases a robust and scalable architecture using the MERN stack (MongoDB, Express.js, React.js, and Node.js) to deliver a dynamic and secure media streaming experience. By combining modular design, modern cloud deployment tools (Render and Vercel), and third-party services like Cloudinary and JWT for secure media handling and user authentication, Ourtube effectively demonstrates the viability of building a real-world, production-ready application with open-source tools.

The backend, developed with Express and MongoDB, supports user accounts, video uploads, comments, and secure authentication via JWT. The frontend, built in React, ensures a smooth, single-page application experience with dynamic content rendering. Integration with Postman for testing and GitHub Actions for CI/CD confirms the platform's readiness for modern development workflows.

**Summary**

The Ourtube project underscores several key achievements:

* **Full-Stack Implementation:** Leveraging the full MERN stack, the platform handles everything from frontend interactions to backend logic and data storage.
* **Feature-Rich Functionality:** Includes user sign-up, login, profile management, video uploads, commenting, like/dislike systems, and video categorization.
* **Modular Codebase:** Well-structured code with separation of concerns between routes, controllers, and components for scalability and maintainability.
* **Modern UI/UX:** Built using React components, the user interface supports responsive design, dynamic rendering, and smooth navigation via React Router.
* **Secure Authentication:** Implemented using JWT and cookies to protect user sessions and route access.
* **Testing & Debugging:** Postman was used for testing backend API endpoints, while Chrome DevTools helped debug and optimize frontend performance.
* **CI/CD and Deployment:** GitHub Actions automate testing and deployment. The backend is hosted on Render, while the frontend is deployed via GitHub on Vercel with environment variables securely managed.
* **Open-Source Collaboration:** Built in public, this project is designed for community contribution, inviting other developers to improve and extend its functionality.
* **Scalability & Extensibility:** The architecture supports future integration of advanced features, including AI-based recommendation systems.

This project not only delivers a functioning media streaming platform but also demonstrates how developers can use modern web technologies to build scalable, secure, and collaborative applications. Future improvements can include real-time features (e.g., live chat), enhanced security with OAuth, and machine learning-based content recommendations. keep them engaged with the platform. The system’s continuous learning and adaptation can further improve the quality of

The integration of a machine learning-based recommendation system demonstrates how AI can be used to personalize user interactions. As the system evolves, user feedback will improve the quality of recommendations, making the platform more engaging and valuable.

REFERENCES

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No** | **Technology / Tool** | **Reference URL** | **Description** | **Icon** |
| 1 | MongoDB | [mongodb.com/docs](https://www.mongodb.com/docs) | NoSQL database used for storing user, video, and comment data | 🟢 |
| 2 | Mongoose | [mongoosejs.com/docs](https://mongoosejs.com/docs) | ODM for MongoDB, used for schema definition and validation | 🌿 |
| 3 | Express.js | [expressjs.com](https://expressjs.com/) | Backend web framework for building REST APIs | 🚂 |
| 4 | Node.js | [nodejs.org/docs](https://nodejs.org/en/docs) | JavaScript runtime used for building the backend server | 🟩 |
| 5 | React.js | [react.dev](https://react.dev/) | Frontend library used for building UI components | ⚛️ |
| 6 | React Router | [reactrouter.com](https://reactrouter.com/en/main) | Enables single-page application routing in React | 🛣️ |
| 7 | Axios | [axios-http.com](https://axios-http.com/docs/intro) | HTTP client for making API requests from the frontend | 📡 |
| 8 | JSON Web Tokens (JWT) | [jwt.io](https://jwt.io/introduction) | Used for secure user authentication and authorization | 🔐 |
| 9 | Cloudinary | [cloudinary.com](https://cloudinary.com/documentation) | Media storage service for uploading and managing videos/images | ☁️📷 |
| 10 | Postman | [postman.com](https://www.postman.com/) | API testing tool used to manually test backend endpoints | 📬 |
| 11 | Render | [docs.render.com](https://docs.render.com/) | Hosting platform used for deploying the backend | ⚙️🌐 |
| 12 | Vercel | [vercel.com/docs](https://vercel.com/docs) | Hosting platform used for deploying the React frontend | ▲ |
| 13 | GitHub Actions | [github.com/features/actions](https://github.com/features/actions) | CI/CD tool used for automating testing and deployment | 🤖 |
| 14 | Material UI Icons (MUI) | [mui.com/icons](https://mui.com/material-ui/material-icons/) | Icon library used in the React frontend for UI design | 🧩 |
| 15 | Chrome DevTools | [developer.chrome.com/devtools](https://developer.chrome.com/docs/devtools/) | Used to debug frontend issues and inspect network activity | 🛠️ |
| 16 | Heroku (Legacy Deployment) | [devcenter.heroku.com](https://devcenter.heroku.com/categories/reference) | Previously used for backend deployment before switching to Render | 🌐 |
| 17 | CORS | [developer.mozilla.org](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS) | Protocol to allow secure cross-origin requests between frontend and backend | 🔄 |
| 18 | Bcrypt | [npmjs.com/bcrypt](https://www.npmjs.com/package/bcrypt) | Library used to hash user passwords for secure storage | 🔑 |
| 19 | dotenv | [npmjs.com/dotenv](https://www.npmjs.com/package/dotenv) | Loads environment variables from .env file into process.env | 📁 |
| 20 | React Hooks | [reactjs.org/hooks](https://reactjs.org/docs/hooks-intro.html) | Enables functional components to use state and lifecycle methods | 🪝 |

Sign - \_\_\_\_\_\_\_\_

**Supervisor**

(Ms. Mona Kumari)

Assistant Professor

Dept. of Computer Engg, & App.