HEMVATI NANDAN BAHUGUNA GARHWAL UNIVERSITY

(A Central University)

Srinagar (Garhwal), Uttarakhand-246174

School of Engineering and Technology

Session (2019-2023)

A PROJECT REPORT ON

GIMME COMMENTS

Submitted in partial fulfilment for the award of the degree of

Bachelor of Technology

In

Computer Science and Engineering

Hemvati Nandan Bahuguna Garhwal University (A Central University), Srinagar Garhwal (Uttarakhand)

Guided By: -

Mr. Rohan Verma

Dept. of Computer Science and Engineering

Hemvati Nandan Bahuguna Garhwal University

Submitted By: -

Rohit Singh Chauhan (19134501030)

Vishesh Gupta (19134505025)

Pramesh Kumar (19134501027)

B.Tech CSE 8th Semester

DECLARATION

We, **Rohit Singh Chauhan**, bearing the roll no. 19134501030, **Vishesh Gupta**, bearing roll no. 19134505025 and **Pramesh Kumar**, bearing roll no. 19134501027 students of Bachelor of Technology in Computer Science and Engineering (CSE) at Hemvati Nandan Bahuguna Garhwal University (A Central University), Srinagar (Garhwal), submit this project entitled **"Gimme Comment"** to Dept. of Computer Science and Engineering, Hemvati Nandan Bahuguna Garhwal University, Srinagar (Garhwal) for the award of the **Bachelors of Technology in Computer Science and Engineering** and declaring that the work done is genuine and produced under the guidance of **Mr. Rohan Verma**, Department of Computer Science and Engineering, Hemvati Nandan Bahuguna Garhwal University, Srinagar (Garhwal), Uttarakhand. We further declare that the reported work in this project has not been submitted and will not be submitted, either in part or in full, for the award of any degree in this institute or any institute or university.

Rohit Singh Chauhan	**Vishesh Gupta**	**Pramesh Kumar**
19134501030	19134505025	19134501027

Date: 31 July 2023

Place: Srinagar, (Garhwal), Uttarakhand, 246174

CERTIFICATE

This is to certify that, this project report entitled **"Gimme Comment"** submitted by **Rohit Singh Chauhan** (Roll No. 19134501030), **Vishesh Gupta** (Roll No. 19134505025), **Pramesh Kumar** (Roll No. 19134501027) is a bonafide record of the work carried out by them in partial fulfillment of the requirement of the award of Bachelor of Technology in **Computer Science and Engineering** degree from **Hemvati Nandan Bahuguna Garhwal University (A Central University)** at Srinagar (Garhwal), Uttarakhand. This Project report has not been submitted to any other University or Institution for the award of any degree.

Mr. Rohan Verma

Dept. of Computer Science and Engineering

Hemvati Nandan Bahuguna Garhwal University

ACKNOWLEDGEMENT

We would like to express our deepest gratitude to all people for sprinkling their help and kindness in completing this project. I would like to start this moment by invoking my purest gratitude to **Mr. Rohan Verma**, Department of Computer Science and Engineering, Hemvati Nandan Bahuguna Garhwal University (A Central University), Srinagar (Garhwal), Uttarakhand, our project instructor.

Completing this project could not have been possible without his expertise and invaluable guidance in every phase at Hemvati Nandan Bahuguna Garhwal University (A Central University), Srinagar (Garhwal), Uttarakhand for helping me. We would like to thank our **Dean Prof. Dr. M.P. Thapliyal, HOD Prof. Y.P Raiwani, Prof. M.M.S. Rauthan, Associate Prof. Pritam Singh Negi, , Associate Prof. Dr. Prem Nath, Assistant Prof. Om Prakash, Assistant Prof. Vijay Prakash Bijlwan** and all the lab assistants and other staff of Computer Science and Engineering Department, Hemvati Nandan Bahuguna Garhwal University (A Central University), Srinagar (Garhwal) Uttarakhand, for their kind support. Last but not least, we would like to thank our parents and friends for their unwavering belief despite the ups and downs in my journey.

INDEX

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This project aims to demonstrate the working of a server providing comments as a service. We will provide this service with a website called Gimme Comment that provides comment box functionality to other third-party websites that do not intend to create their comment box.

The Frontend of this project will be a website called Gimme Comment where users would register and will be able to request a unique comment box service for each of their webpages/websites. They will be provided with a unique key and a set of HTML codes that they need to include in their codebase. Tech Stack that we will use includes ReactJS, ExpressJs, MUI, etc.

The Backend is the core of this project serving authentication and user/comments database management with real-time updates. The backend will be made using ExpressJS with the help of MongoDB services.

CHAPTER 1

INTRODUCTION

Comments are a response to your content from people who took the time (we hope) to read it. Comments can be positive or critical, they can ask questions, and they can add to the relevant information on the page. That last item is the most important part for folks actively involved in content marketing.

Commenting service websites, also known as commenting systems or platforms, are online services that enable users to leave comments on web content. This content could include blog posts, news articles, multimedia files, products, and more. They are usually implemented as a feature of the site they are used on, enriching the content by encouraging conversation, debate, and interaction among users.

These services typically come with a host of features designed to facilitate and moderate discussions. These may include user authentication, threading of conversations, like/dislike buttons, social media integration, and advanced moderation tools to help control spam, abuse, and inappropriate content.

Examples of popular commenting service websites include Disqus, Livefyre, and Commento. Some social media platforms like Facebook also offer their own commenting systems that can be integrated into other websites.

The use of commenting services can have many benefits. They can increase user engagement, provide valuable feedback and insights, foster a sense of community, and potentially improve the SEO performance of a webpage.

However, these services also face significant challenges, such as dealing with abusive behavior and spam, ensuring user privacy, maintaining interoperability across different platforms and devices, and striking the right balance between open discussion and moderation. Despite these challenges, commenting services continue to play a vital role in the digital landscape, providing a space for public discourse and community building.

With Gimme Comment your website gains a feature-rich comment system complete with social network integration, advanced administration and moderation options, and other extensive community functions. Most importantly you are instantly plugging into our web-wide community network, connecting millions of global users to your small blog or large media hub.

1.1 According to Gimme Comment, the key points of the application are:

1) User Authentication: A user authentication policy is a process in which you verify that someone who is attempting to access services and applications is who they claim to be. This can be accomplished through a variety of authentication methods, such as entering a password to login in your account.

2) User Data Management: User data management is the process of acquiring, organizing, and using customer data to better understand customers and ultimately increase conversions and retention.

3) Comment Management: Comment management is the process by which a business or agency oversees the receipt, control, and security of comments solicited from the public regarding a specific project or general service.

4) Realtime Updates: Realtime Updates enable the ability to instantly modify features without requiring users to refresh a page or restart an application. Features can be launched, modified, or disabled instantaneously.

5) Privacy: Privacy generally means the ability of a person to determine for themselves when, how, and to what extent personal information about them is shared with or communicated to others. This personal information can be one's name, location, contact information, or online or real-world behavior.

6) Confidentiality: Confidentiality involves a set of rules or a promise usually executed through confidentiality agreements that limits access or place restrictions on certain types of information.

CHAPTER 2
DESIGNING OF APPLICATION

2.1 Methodology

A. **Planning and Design:** Define the project requirements, functionality, and user interface. Build wireframes and mock-ups for applications.
B. **Development:** Front and backend work. Create a database and user authentication system.
C. **Testing:** Perform rigorous testing to identify and fix bugs and ensure data security.
D. **Distribution:** Make the app available globally.

2.2 Purpose and Solution

The purpose of the Gimme Comment Website is to provide a platform for users to express their opinions, share ideas, ask questions, and engage in discussions related to a piece of content. This could be an article, blog post, video, or other forms of content that are hosted on different websites. It provides people with a user-friendly and effective commenting service. This project focuses on comments as a service. With Gimme Comment your website gains a feature-rich comment system complete with social network integration, advanced administration and moderation options, and other extensive community functions. Most importantly you are instantly plugging into our web-wide community network, connecting millions of global users to your small blog or large media hub.

The main objectives of the Gimme Comment application are:

1. Provide Free Commenting Service:

The application aims to provide users feature to add a free commenting service to their website. This will help reduce their time to generate and manage an individual comment section on their website.

2. Comment Management:

Gimme Comment allows the user to manage the comments people post on their websites.

3. Time Assistance:

It will be very easy to use, users have to add just a few lines of code into their codebase and all the comments facilities will be provided automatically. This will help developers to do their work in less time.

4. Space Efficient

To use this service, users just have to add a few lines of code into their codebase and they will be provided our service, by this the space of the user's codebase will be less and will be efficient.

5. Ease of Implementation:

These services offer ready-to-use, scalable solutions that can be easily integrated into different platforms, saving developers the time and effort required to build their commenting systems.

6. Cross-platform Compatibility:

These services ensure that the commenting system works well across a variety of devices, platforms, and browsers.

2.3 Key Features of the Application

According to Gimme Comment, the key points of the application are:

- User authentication
- User Data Management
- Comment Management
- Real-time updates
- Privacy
- Confidentiality

CHAPTER 3

FLOW CHART AND THEORETICAL ANALYSIS OF THE APP

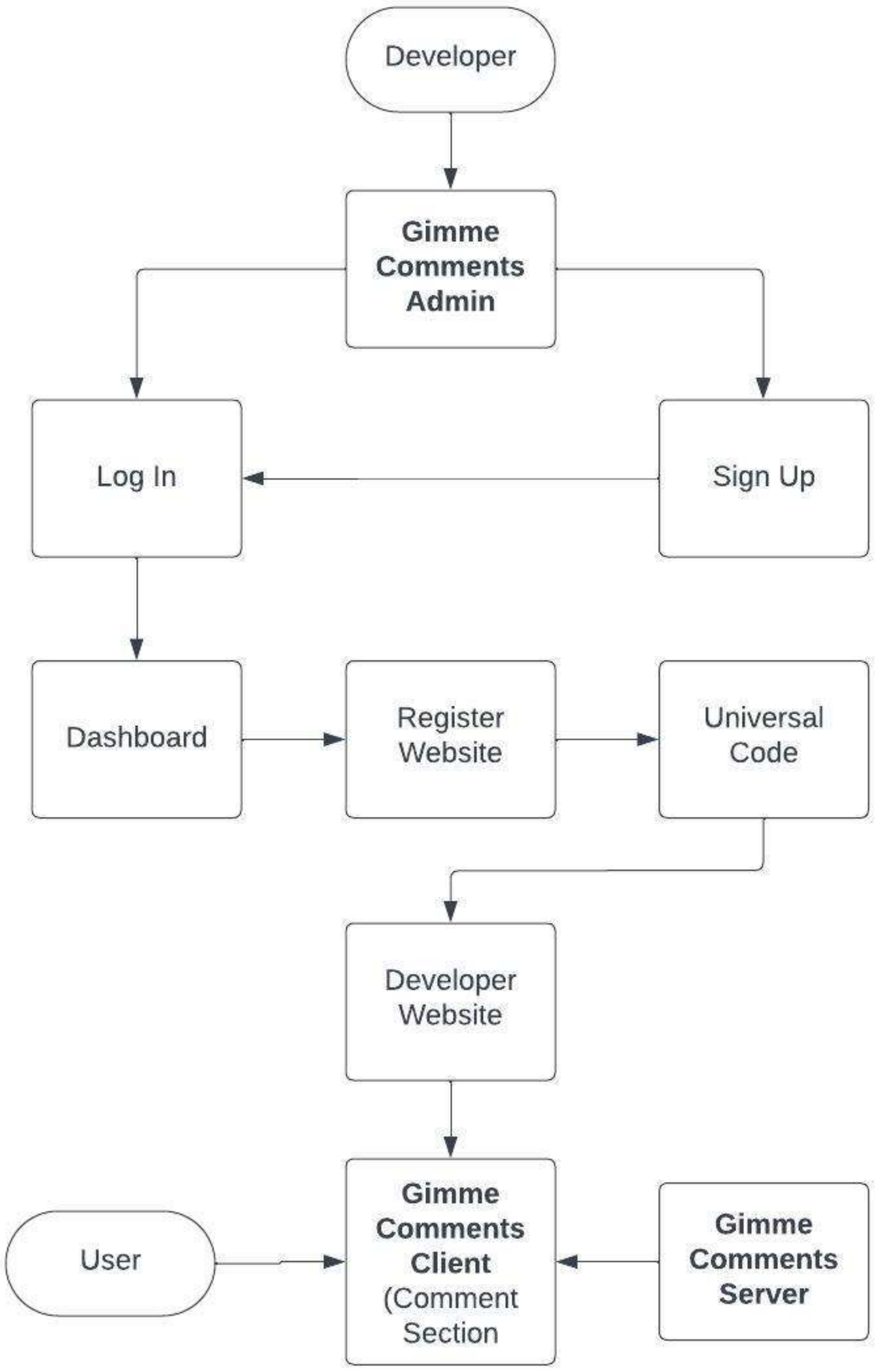


Fig 3.1 Flow Chart of Application

CHAPTER 4
BENEFITS AND CHALLENGES

4.1 Benefits

1. Easy to access

With the Gimme Comment website, commenting service can be provided anytime and anywhere, which is a very convenient method to host and manage comments on a once-self-owned website. If website developers find it difficult to handle the user data and their comments they can easily reach out to Gimme Comment to get good comment services for each website.

2. User Engagement

Comments sections can drive user engagement by encouraging visitors to interact with the content and with each other. This can increase time spent on the site and encourage users to return regularly.

3. Feedback Mechanism

Comments provide a platform for users to share their thoughts, feedback, and perspectives. This can be valuable for content creators and businesses who want to understand their audience better.

4. Independent

Comment Box in general relies on the user codebase and makes their database heavy. Unlike traditional comment sections, Gimme Comment will be independent, as our codebase will not be depending on the user's codebase or any modules which are developed by them. It will be independent of the user's codebase.

5. Manage Worry

Users do not have to worry about the comment part of their creating application. All the functionalities and services will be provided by our side as we will be managing all the backend and frontend UI parts of our Gimme Comment application.

6. Ease of Implementation

Gimme Comment provides a simple, plug-and-play solution that website owners can easily integrate into their websites. This can save time and resources compared to building a proprietary commenting system.

7. SEO Benefits

More engagement, including comments, can potentially improve a site's Search Engine Optimization (SEO). Google's algorithms often interpret user engagement as a signal of quality content.

8. Community Building

Comment sections can help to foster a sense of community among users, especially when there are regular discussions and interactions. This can lead to a loyal user base and increase the value and appeal of the website.

4.2 Challenges

A website that provides commenting service to other websites can be a valuable tool for those who want to deal with and manage their comments on their websites. However, there are some difficulties in creating and using such tools. Some of the important topics are:

1. Moderation: Perhaps the most significant challenge is the management of inappropriate or abusive comments. This includes spam, personal attacks, hate speech, misinformation, and other types of harmful content. Manual moderation can be labor-intensive and expensive, while automated moderation can be inaccurate or too strict, resulting in false positives.

2. Scalability: As the number of comments grows, it becomes harder to maintain the performance and reliability of the commenting service. This is especially true for popular websites or posts, where thousands or even millions of comments may be made in a short period.

3. User authentication: Ensuring that users are who they say they are can be challenging. While anonymous commenting can lead to more engagement, it can also increase the risk of abusive behavior. On the other hand, requiring users to sign in through a social media account can lead to privacy concerns.

4. Privacy: Protecting user data is critical, especially as users become more aware and concerned about their online privacy. This includes not only keeping comments and personal information secure from hackers but also respecting users' privacy in terms of data collection and use.

5. Interoperability: Ensuring that the commenting system works well across a variety of websites, platforms, devices, and browsers can be a technical challenge.

6. User Engagement: Encouraging users to leave comments, and fostering a healthy and engaging discussion, is not always easy. This often involves striking a balance between open discussion and moderation.

7. Legal issues: Depending on the jurisdiction, the operator of a commenting system may be legally liable for the content that users post. This can lead to challenges in managing and enforcing community guidelines.

8. Algorithmic bias: Automated systems used to sort and highlight comments can inadvertently favor or discriminate against certain types of content or users. This can lead to biased or homogeneous discussions.

9. Echo chambers: A commenting service may unintentionally create echo chambers, where users are only exposed to opinions that align with their own. This can polarize discussions and create a distorted view of consensus or public opinion.

10. Monetization: Finally, it can be challenging for commenting service websites to generate revenue. Advertising can alienate users, while subscription or pay-per-use models can reduce the number of comments and users.

CHAPTER 5

SOFTWARE AND HARDWARE REQUIREMENTS

5.1 Software Requirement

5.1.1 ReactJS (JavaScript Library)

React (also known as React.js or ReactJS) is an open-source, front-end JavaScript library for building user interfaces or UI components. It was developed by Facebook and is maintained by Facebook and a community of individual developers and companies.

React allows developers to create large web applications that can change data, without reloading the page. It aims to be simple, fast, and scalable and works only on user interfaces in applications.

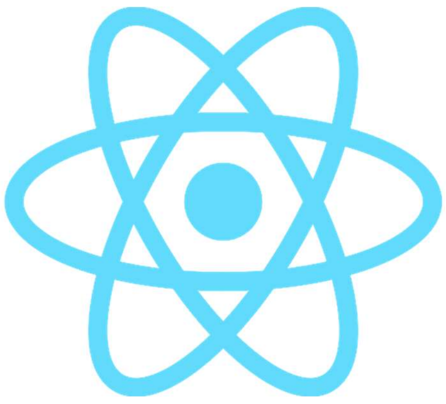


Fig 5.1 ReactJS(https://upload.wikimedia.org/wikipedia/commons/thumb/a/a7**)**

Features

Declarative

• React adheres to the declarative programming paradigm. Developers design views for each state of an application and React updates and renders components when data changes. This is in contrast with imperative programming.

Components

• React code is made of entities called components. These components are reusable and must be formed in the SRC folder following the Pascal Case as its naming convention (capitalize camelCase). Components can be rendered to a particular element in the DOM using the React DOM library. When rendering a component, one can pass the values between components through "props".

Functional components

• Function components are declared with a function that then returns some JSX

•Class-based components

Class-based components are declared using ES6classes.

React hooks

• Hooks are functions that let developers "hook into" React state and lifecycle features from function components. Hooks do not work inside classes — they let you use React without classes. React provides a few built-in hooks like useState, useContext, useReducer, useMemo, and useEffect. Others are documented in the Hooks API Reference. estate and useEffect, which are the most commonly used, are for controlling state and side effects respectively

5.1.2 NodeJS

Node.js is an open-source, cross-platform JavaScript runtime environment that executes JavaScript code on the server side. It was developed in 2009 by Ryan Dahl and has since become one of the most popular and widely-used back-end technologies.

Fig 5.2 NodeJS(https://pluralsight2.imgix.net/paths/images/nodejs-45adbe594d.png)

Node.js is built on Google's V8 JavaScript engine, the same engine that powers the Google Chrome web browser. This makes Node.js fast and efficient, and capable of handling high-traffic, real-time applications with ease. One of the key benefits of Node.js is its ability to handle multiple connections simultaneously. This makes it well-suited for building real-time applications, such as chat apps, online games, and streaming services.

Node.js is also highly scalable, which means that it can be easily scaled up or down to meet the demands of your application. Another key benefit of Node.js is its npm (Node Package Manager), a repository of over one million packages, including libraries and tools, that can be easily installed and used in Node.js projects. npm makes it easy to find and use the tools and libraries you need for your project, without having to reinvent the wheel.

5.1.3 ExpressJS

Express.js, or simply Express, is a back-end web application framework for building RESTful APIs with Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js.

Fig 5.3 ExpressJS (https://geekflare.com/wp-content/uploads/2023/01/expressjs.png**)**

The original author, TJ Holowaychuk, described it as a Sinatra-inspired server, meaning that it is relatively minimal with many features available as plugins. Express is the back-end component of popular development stacks like the MEAN, MERN, or MEVN stack, together with the MongoDB database software and a JavaScript front-end framework or library.

5.1.4 MongoDB

MongoDB is a popular, open-source NoSQL database that is designed for handling large amounts of data in a scalable and flexible manner. It was first released in 2009 and has since become one of the most widely used databases for web and mobile applications.

Fig 5.4 MongoDB (https://upload.wikimedia.org/wikipedia/commons/thumb)

MongoDB is a popular, open-source NoSQL database that is designed for handling large amounts of data in a scalable and flexible manner. It was first released in 2009 and has since become one of the most widely used databases for web and mobile applications.

One of the main benefits of MongoDB is its document-oriented data model, which allows developers to store and retrieve data more naturally and intuitively than traditional relational databases. This makes MongoDB an excellent choice for applications that require complex, hierarchical data structures and real-time data access.

Another key feature of MongoDB is its scalability. With MongoDB, developers can easily scale their databases as their applications grow, without having to worry about the complexities of database sharding and replication. This means that developers can build large and complex applications without having to worry about the performance and reliability of their databases.

5.1.5 Vercel

Vercel is a cloud platform that provides deployment and serverless computing services, focusing primarily on the deployment of front-end assets. It was developed by the creators of the Next.js framework but supports many other development frameworks as well, including Angular, Nuxt.js, Create React App, Gatsby, Vue.js, and more.

Here are some key features of Vercel:

1. **Easy Deployments:** Vercel offers a simple workflow for developers to deploy their applications. You can connect your GitHub, GitLab, or Bitbucket account and it will automatically deploy your projects with every push.
2. **Serverless Functions:** Vercel provides serverless computing capabilities where you can write and deploy code without worrying about the underlying infrastructure.

3. **Performance-focused:** Vercel automatically optimizes your projects for the best performance by applying best practices like minification, compression, caching, CDN distribution, and more.
4. **Edge Network:** Vercel distributes your content across a global Edge Network, ensuring faster load times by serving your content from the point of presence (PoP) closest to your users.
5. **Preview Deployments:** For every commit, Vercel creates a unique preview URL for testing and feedback. This allows you to test every change thoroughly before it goes into production.
6. **Environment Variables:** Vercel allows you to configure environment variables, which lets you customize the behavior of your application based on the current environment.

Fig 5.5 Vercel (https://mms.businesswire.com/media/20211123005573/en/929867/23)

5.1.6 GitHub

GitHub is a web-based platform that provides hosting for software development projects. It is widely used for version control, collaboration, and code sharing among developers. Here are some of the key features of GitHub:

1. **Version Control:** GitHub uses Git, a distributed version control system, to track changes to files and manage project history. Developers can create branches, make changes, and merge their work back into the main project.

2. **Repositories:** A repository, or "repo," is a collection of files and folders associated with a project. GitHub hosts repositories and allows users to create public or private repositories depending on their needs.
3. **Pull Requests:** When working on a team, developers can use pull requests to propose changes to the main project. Other team members can review the changes, provide feedback, and approve or reject the request before merging it.
4. **Issue Tracking:** GitHub includes a built-in issue tracking system, allowing users to report bugs, suggest new features, or discuss ideas. Issues can be assigned, labeled, and managed throughout the development process.
5. **Collaboration:** GitHub facilitates collaboration among developers by providing tools for commenting on code, managing discussions, and collaborating on projects, making it easier for teams to work together effectively.
6. **Wikis and Documentation**: Projects on GitHub can have wikis and documentation associated with them. This allows developers to maintain project documentation and provide essential information for contributors and users.
7. **GitHub Actions:** GitHub Actions is a powerful feature that allows developers to automate workflows, such as building, testing, and deploying projects, directly from their repositories.
8. **Integrations:** GitHub integrates with various third-party tools and services, enabling developers to connect their repositories to services like CI/CD platforms, code quality tools, project management systems, and more.
9. **Social Networking:** GitHub has a social aspect, allowing users to follow other developers, star repositories they find interesting, and contribute to open-source projects.
10. **Security:** GitHub provides security features like vulnerability scanning, code scanning, and dependency insights to help developers identify and address potential security issues in their projects.

Overall, GitHub's features make it a powerful and popular platform for developers and teams to collaborate, manage their code, and contribute to open-source projects effectively. It has played a significant role in promoting the open-source community and enabling smooth software development practices.

5.1.7 Visual Studio Code

Visual Studio Code (often simply referred to as VS Code) is a free, open-source, lightweight but powerful source code editor developed by Microsoft for Windows, Linux, and macOS. It comes with built-in support for JavaScript, TypeScript, and Node.js, and has a rich ecosystem of extensions for other languages, including C++, C#, Python, PHP, Go, and more. It also supports tools and services for DevOps like Microsoft Azure.

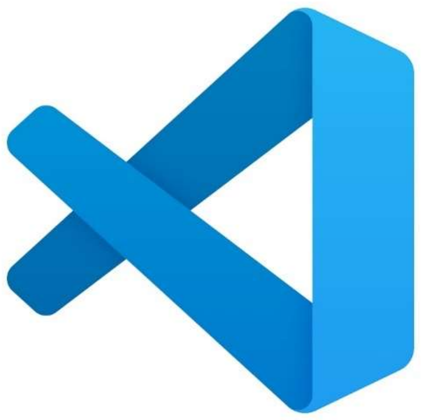


Fig 5.6 Visual Studio Code (https://yt3.googleusercontent.com**)**

Here are some key features of Visual Studio Code:

1. **Syntax Highlighting and Automatic Indentation:** VS Code supports syntax coloring and automatic indentation for more than 30 programming languages, which enhances readability and reduces coding errors.
2. **IntelliSense:** IntelliSense provides smart completions based on variable types, function definitions, and imported modules. It's a coding feature for autocompleting code, offering suggestions, and providing documentation to aid developers in writing code more quickly and accurately.
3. **Debugging:** VS Code comes with built-in debugging support, allowing for launching and debugging code from the editor. The debugger includes features like breakpoints, stack traces, and a debug console.

4. **Git Integration:** VS Code has out-of-the-box Git integration that includes features like diffs, pull requests, and viewing changes.

5. **Extensions and Customization:** VS Code has a marketplace that offers thousands of extensions, adding new features and capabilities to the software. This allows you to customize VS Code to your work style, and increase productivity and efficiency.

5.1.8 Render

Render is a unified platform to build and run applications, websites, and other services that require a backend infrastructure. It was designed to abstract away the complexities of managing and orchestrating cloud infrastructure so that developers can focus on creating and deploying software.

Here are some of the key features of Render:

1. **Fully Managed Services:** Render provides a range of fully managed services such as servers, static sites, cron jobs, PostgreSQL databases, and private services that help in deploying full-stack applications completely on Render without managing a server.

2. **Automatic SSL:** Every service deployed on Render automatically gets a free SSL certificate, and it's automatically renewed as well.

3. **Pull Request Previews:** Render can automatically create a new instance of your application for every pull request, making it easy to test changes in a live environment before they're merged.

4. **Zero-downtime Deployments:** Render ensures that there is zero downtime during deployments, and if a deployment fails, Render automatically falls back to the last stable version.

5. **Infrastructure as Code:** With Render, you can describe your infrastructure using a YAML file, version control it, and track changes over time. It's easy to replicate your infrastructure across different environments.

5.1.9 Postman

Postman is a popular API (Application Programming Interface) development tool that helps developers build, test, and document APIs. It started in 2012 as a side project to simplify the API workflow and has since grown into a tool used by millions of developers around the world.

Here are some key features of Postman:

1. **API Client:** Postman allows you to send requests to APIs and view responses in its friendly user interface. You can send various types of HTTP requests like GET, POST, DELETE, PUT, PATCH, etc.
2. **Testing:** Postman provides a feature to test APIs by writing test scripts. It has a powerful testing sandbox that lets you write JavaScript code to test your API responses. You can write tests to verify the correctness of APIs, and these tests can be used in automated testing.
3. **Automation Testing:** Using the Collection Runner or Newman, you can run sets of tests in multiple iterations, simulating actual user data and behavior.
4. **API Documentation:** You can automatically generate and publish documentation for your APIs, making it easy for developers and stakeholders to understand how to use them.
5. **Collaboration:** You can share your collections with your team, keep them in sync, and collaboratively debug and fix issues.

5.1.10 AWS S3

Amazon Simple Storage Service (Amazon S3) is a scalable object storage service offered by Amazon Web Services (AWS). It is designed to store and retrieve any amount of data from anywhere on the web. It is designed for 99.999999999% (11 9's) of durability and stores data for millions of applications used by market leaders in every industry.

Key features of Amazon S3 include:

1. **Scalability:** Amazon S3 can store any amount of data and can scale in response to demand.

2. **High Durability:** Amazon S3 is designed for 99.999999999% of durability. It keeps your data safe by storing copies of it across multiple systems.
3. **Security:** Amazon S3 supports data transfer over SSL and automatic data encryption once it is stored. You can also configure bucket policies to manage object permissions and control access to your data using AWS Identity and Access Management (IAM).
4. **Bucket and Object Management:** Data in Amazon S3 is stored in containers called "buckets." Within these buckets, data objects are organized and can be managed.
5. **Versioning:** Amazon S3 provides versioning for your objects, allowing you to preserve, retrieve, and restore every version of every object in your bucket.

5.2 Hardware Requirement

- Processor Intel Pentium 4 or later, or equivalent.
- Storage: 200 MB or more of free disk space.
- Graphics card: A graphics card with 128 MB of memory or more is recommended.

Operating System Requirement

- Windows 7, 8, 10,11 / MAC
- Windows 10 or later, macOS 10.12 or later, Linux, or Chrome OS.

CHAPTER 6
PACKAGES USED

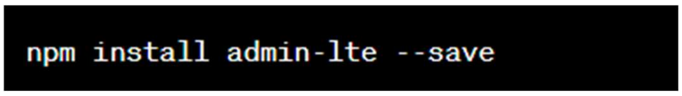
Name	Last Version Tested With
Admin-lte	3.2.0
Axios	0.27.2
Jsonwebtoken	9.0.0
React	18.2.0
React-router-dom	6.3.0
Bootstrap	5.0
Sendgrid/mail	7.7.0
Aws-SDK	2.1407.0
Cors	2.8.5
Express	4.18.2
Mongoose	6.9.0
Multer	1.4.5-lts.1

Table: 6.1 Packages Used

6.1 Admin-lte

AdminLTE is a popular open-source admin dashboard & control panel theme. It's based on Bootstrap 3 and 4, and it provides a range of responsive, reusable, and commonly used components, enabling developers to create attractive and consistent user interfaces.

Installation: Run the following command in your terminal or command prompt:

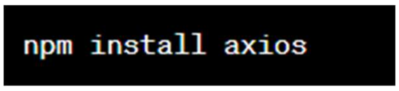


```
npm install admin-lte --save
```

6.2 Axios

Axios is a popular, promise-based HTTP client for the browser and Node.js. It has a simple and easy-to-use API that allows developers to send asynchronous HTTP requests to REST endpoints and perform CRUD operations (Create, Read, Update, and Delete).

Installation: Run the following command in your terminal or command prompt:

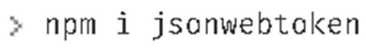


```
npm install axios
```

6.3 Jsonwebtoken

Jsonwebtoken is a JavaScript library that allows you to create and verify JSON Web Tokens (JWTs). JWTs are a popular way to authenticate users and protect sensitive data.

Installation: Run the following command in your terminal or command prompt:

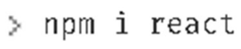


```
> npm i jsonwebtoken
```

6.4 React

The React package contains only the functionality necessary to define React components. It is typically used together with a React renderer like react-dom for the web or react-native for the native environments.

Installation: Run the following command in your terminal or command prompt:



```
> npm i react
```

6.5 React-Router-Dom:

React Router DOM is a JavaScript library that provides routing functionality for React applications. It allows you to create single-page applications (SPAs) that can be navigated between different pages without reloading the entire page.

React Router DOM is a popular library for routing in React applications.

Installation: Run the following command in your terminal or command prompt:

```
> npm i react-router-dom
```

6.6 Bootstrap

Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains HTML, CSS, and JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components

Installation: Run the following command in your terminal or command prompt:

```
> npm i bootstrap
```

6.7 Sendgrid/mail

SendGrid Mail is a cloud-based email delivery service that helps you send, track, and analyze email. It is a popular choice for businesses of all sizes, as it offers a variety of features and functionality, including **Email delivery, Email tracking, Email analytics, etc.**

Installation: Run the following command in your terminal or command prompt:

```
> npm i @sendgrid/mail
```

6.8 Aws-SDK

An AWS SDK (Software Development Kit) is a set of tools that developers use to interact with Amazon Web Services (AWS). SDKs provide a consistent and familiar programming interface for accessing AWS services, regardless of the programming language you are using.

There are SDKs available for a variety of programming languages, including Java, Python, JavaScript, and C#. SDKs also provide a variety of features, such as **Authentication, APIs, and Utilities.**

Installation: Run the following command in your terminal or command prompt:

```
> npm i aws-sdk
```

6.9 CORS

CORS (Cross-Origin Resource Sharing) is a mechanism that allows web browsers to make requests for resources from other origins. By default, web browsers restrict cross-origin requests to protect users from malicious content.

The Cors npm package is a Node.js middleware that can be used to enable CORS in your applications.

Installation: Run the following command in your terminal or command prompt:

```
> npm i cors
```

6.10 Express

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It is designed to be easy to use and extend, and it is used by millions of developers around the world.

Installation: Run the following command in your terminal or command prompt:

```
> npm i express
```

6.11 Mongoose

Mongoose is a MongoDB Object Data Modeling (ODM) library for Node.js. It provides a straightforward, schema-based solution to model your application data. Mongoose manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.

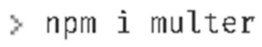
Installation: Run the following command in your terminal or command prompt:

```
> npm i mongoose
```

6.12 Multer

Multer is a node.js middleware for handling multipart/form data, which is primarily used for uploading files. It is written on top of the busy box for maximum efficiency.

Installation: Run the following command in your terminal or command prompt:



```
> npm i multer
```

CHAPTER 7
FULL STACK DEVELOPMENT

7.1 Introduction

Full-stack development refers to the development of both the front-end (client-side) and the back-end (server-side) parts of a web application. This kind of web development requires a wide variety of skills because a full-stack developer should be able to work with a broad range of technologies and languages.

Let's break down the two major components of full-stack development:

7.2 Front-end development

Front-end development, also known as client-side development, is the practice of creating the visual elements that users interact with when they access a web application in a web browser. Essentially, front-end developers make sure the website's interface is easy to use and match the design specification.

Front-end development involves three main languages:

1. **HTML (HyperText Markup Language):** This is the backbone of any website. It creates the structure and content of a web page, like text, images, and so on.
2. **CSS (Cascading Style Sheets):** This is what styles a website and lays out its look and feel. It sets the colors, fonts, and overall layout of the page. It also handles responsive design, which ensures websites look good on all screen sizes.
3. **JavaScript:** This makes a web page interactive. It can be used to create dynamic content like dropdown menus, form submissions, interactive maps, animations, and even more complex application logic.

In addition to these core technologies, modern front-end development often involves several other tools and frameworks to help create more complex applications more efficiently. These include:

- **Front-End JavaScript Frameworks and Libraries:** These provide pre-written JavaScript code to use for routine programming features and tasks, like React.js, Angular.js, Vue.js, etc. They provide a structure for the web application and allow developers to create scalable, robust, and maintainable web apps.
- **CSS Frameworks:** Like Bootstrap, Tailwind CSS, and Material-UI, which help speed up the development process and make it easy to create responsive and attractive web pages.
- **Version Control Systems:** Like Git, which tracks changes made to the code over time and allows developers to revert to any previous version of their work.
- **Package Managers:** Like npm (Node Package Manager) or Yarn, which are used to automate the process of installing, upgrading, configuring, and removing software libraries/packages consistently.
- **Build Tools:** Like Webpack, Gulp, or Grunt. These tools are used to automate tasks such as compiling code, minification (which makes your files take up less space), and auto-prefixing (which makes sure your CSS works across many different browsers).
- **Testing Libraries and Frameworks:** Like Jest, Mocha, Jasmine, etc. These tools help developers to test their code and ensure it behaves as expected.

The main goal of front-end development is to create clear, clean, and user-friendly interfaces that function across different browsers, platforms, and devices. It plays a crucial role in web accessibility and user experience.

7.3 Back-end development

Back-end development, often referred to as server-side development, is all about what goes on behind the scenes of a web application. While the front end is the part of the web users interact with, the back end is where the logic, server configuration, data management, and other non-user-facing aspects of web applications and websites are implemented.

Here are the major components of back-end development:

1. **Server-side Programming Languages:** These are languages that are used to interact with the server and databases. They contain the logic of the applications and manipulate data.

Some common server-side languages are:

- JavaScript (Node.js)
- Python (Django, Flask)
- Ruby (Ruby on Rails)
- Java (Spring)
- PHP (Laravel)
- .NET (C#)

2. Databases and Database Languages: Databases store the data involved in a web application, and database languages (like SQL for relational databases or query languages for NoSQL databases) are used to manage the data and interact with it. Common databases include MySQL, PostgreSQL, MongoDB, and SQL Server.

3. Server: A server is a computer that processes requests and delivers data to other computers over a local network or the internet. Knowledge of server architecture, as well as working with server software (like Apache, Nginx, or Microsoft's IIS), is crucial for back-end developers.

4. API (Application Programming Interface): APIs are a set of rules and protocols for building and interacting with software applications. REST and GraphQL are popular types of APIs used in web development.

5. Version Control Systems: Like Git, back-end developers use these to keep track of changes to the codebase and collaborate with others.

6. Testing/Debugging: Back-end developers use various tools to debug and test their code. These can include unit testing, integration testing, and functional testing tools.

7. Security: Back-end developers need to ensure data privacy and compliance by understanding various security concerns like data breaches, SQL injection, cross-site scripting (XSS), etc., and by implementing appropriate security measures.

The back-end part of an application is responsible for things like calculations, business logic, database interactions, and performance. Most of the code that is required to make an application work will be done on the back end. Back-end developers need to consider what the user will need in the future and create flexible, scalable systems. They also work closely with front-end developers to integrate their work and to ensure that the application works seamlessly as a whole.

CHAPTER 8

WORKING

8.1 Introduction

The Gimme Comments project is broadly divided into 3 components including:

- Gimme Comments Server
- Gimme Comments Admin
- Gimme Comments Client

These three separate components work hand-in-hand with each other, the server is the backend connecting with MongoDB Atlas database providing a place to store user, website, and comments data, the admin panel serves as a dashboard for users who are developers and wish to register their website on Gimme Comments and take benefit from its service. The client is the code that is served to the developer's website which is the actual frontend encountered by general users to post comments and interact with each other.

8.2 Gimme Comments Server

8.2.1 Introduction

Gimme Comments Server is hosted on Render.com and is the backbone of our project. It acts as a medium between frontend (including admin panel and client) and database which is MongoDB Atlas and provides various features including:

- Authentication: There are various APIs for secure authentication using email password strategy and verification of email using OTP-based verification. Passwords are also encrypted using JWT.
- Websites: It provides APIs to perform CRUD (Create, Read, Update, and Delete) operations on websites used by developers from the admin panel who want to use the Gimme Comments service on their websites.
- Comments: Server server client or comments section to all registered websites and having client code installed correctly. Comments service includes authentication, posting comments, likes, replies, and editing.

8.2.2 Folder Structure

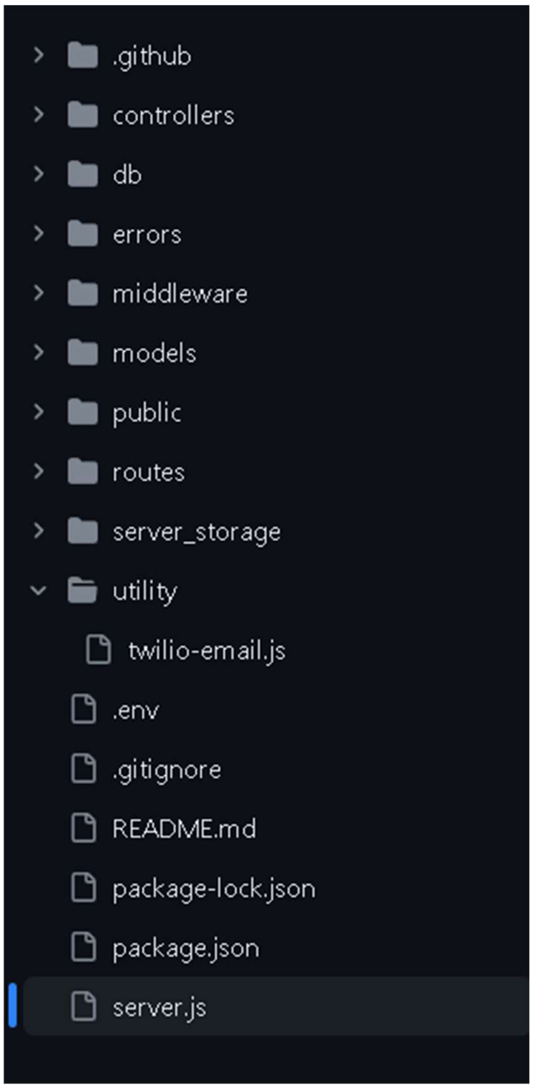
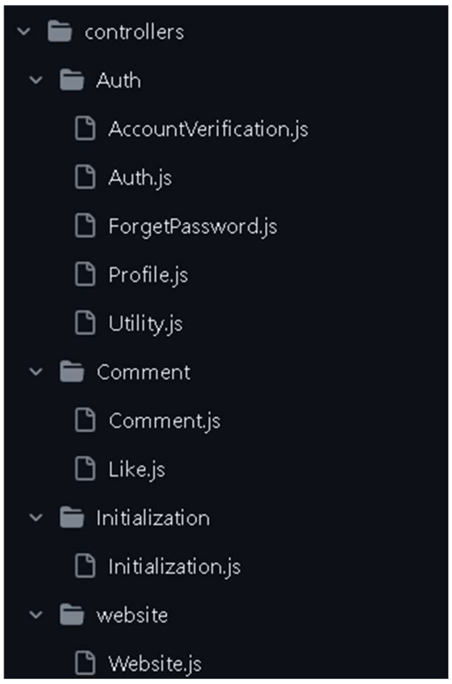

```
> .github
> controllers
> db
> errors
> middleware
> models
> public
> routes
> server_storage
v utility
    twilio-email.js
  .env
  .gitignore
  README.md
  package-lock.json
  package.json
  server.js
```

Fig 8.1 Server Folder Structure

1. **.github:** Contains node.js.yml file which contains data about Github Action Runners and is used to auto-update code on the Render.com server whenever server code is updated in the Github repository.
2. **Controllers:** Contains functions related to various apps which connect with the database and provide create, read, update, and delete functionality or other utility services. Various files included in the controller's folder include:
   - **AccountVeriifcation:** Contains function related to account verification using OTP.
   - **Auth:** Contains functions of Login and Signup of user/developer with all types of validations.

- **ForgetPassword:** Included functions required to find an account, send OTP, and reset the password if the correct OTP is entered.
- **Profile:** This file contains functions related to user profile data update, profile image update, profile delete, and getting profile data.
- **Utility:** Server intermediate functions required by other controllers like user_exists, user_already_verified, generate_otp, verify_otp.
- **Comment:** Contains create, read, update, and delete operations on comments with validation and when getting comments for a website, prefills it with several likes.
- **Like:** Deals with creating and deleting likes on comments.
- **Website:** Includes function related to creating, reading, updating, and delete of website data by developers and checking the existence of the website for providing gimme comments service.
- **Initialiation:** Sends build and initial code to load javascript and CSS files of client build folder to the client requesting gimme comments service on their website.

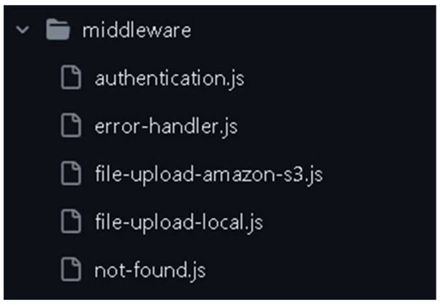


```
controllers
  Auth
    AccountVerification.js
    Auth.js
    ForgetPassword.js
    Profile.js
    Utility.js
  Comment
    Comment.js
    Like.js
  Initialization
    Initialization.js
  website
    Website.js
```

Fig 8.2 Server Controllers

3. **Db:** This folder contains database connection configuration files that use the Mongoose library to connect with MongoDB Atlas.

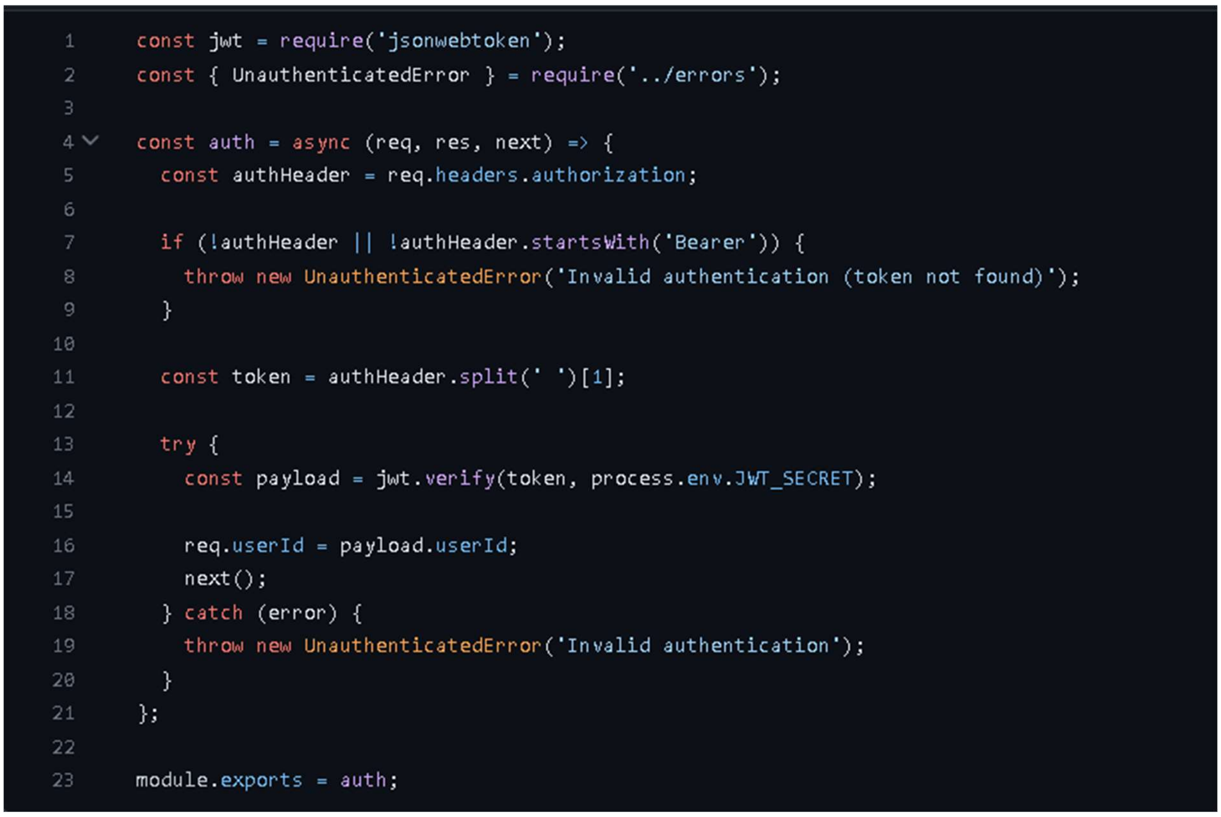
4. **Errors:** It contains files related to various errors and handles them in the server sending proper error codes to the client.
5. **Middlewares:** This contains various middlewares which are required by other APIs and serve crucial functioning to other components.



```
middleware
    authentication.js
    error-handler.js
    file-upload-amazon-s3.js
    file-upload-local.js
    not-found.js
```

Fig 8.3 Server Middleware

- **Authentication:** Contains code that checks the authorization header in every request and extracts the token and uses JWT to decode the token to find the userId of the user who is requesting, attaches the userId to the next requests, and also sends appropriate error messages.

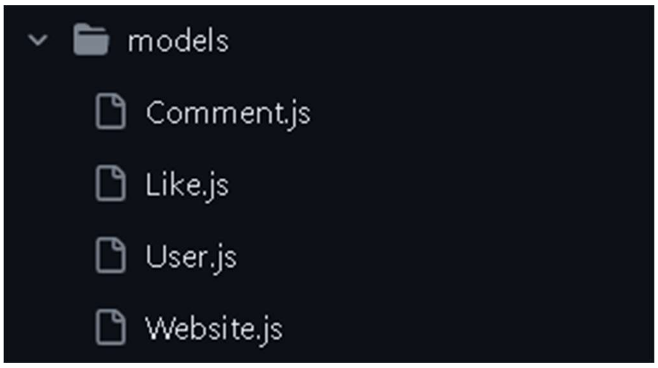


```javascript
const jwt = require('jsonwebtoken');
const { UnauthenticatedError } = require('../errors');

const auth = async (req, res, next) => {
  const authHeader = req.headers.authorization;

  if (!authHeader || !authHeader.startsWith('Bearer')) {
    throw new UnauthenticatedError('Invalid authentication (token not found)');
  }

  const token = authHeader.split(' ')[1];

  try {
    const payload = jwt.verify(token, process.env.JWT_SECRET);

    req.userId = payload.userId;
    next();
  } catch (error) {
    throw new UnauthenticatedError('Invalid authentication');
  }
};

module.exports = auth;
```

Fig 8.4 Authentication Middleware

- **Error-handler:** This is included in the server and handles all errors which are thrown at any step in the flow of request-response and sends appropriate error messages to the client in response.
- **File-upload-amaon-s3:** It contains two functions that are used to upload images to the AWS S3 bucket and delete already uploaded images from the same bucket. When the server sends a request in multipart form data, it decodes this data using the multi library and then uses the upload and delete function to deal with appropriate functioning.
- **File-upload-local:** Uses multi to upload and delete images provided by the server and sends requests in multipart form data, it decodes this data using the multi library and then uses the upload and delete function to deal with appropriate functioning.
- **Not-found:** It handles all requests which are directed toward invalid routes and send an appropriate response with the message "Route does not exist..." and 404 status code.

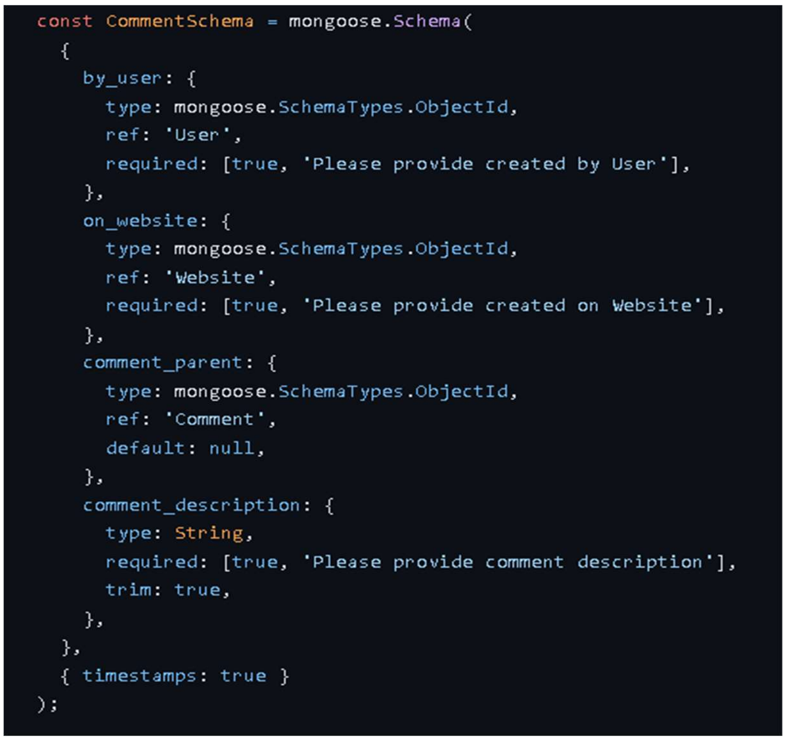
6. **Models:** This folder contains various database schemas. a schema is a formal description of the structure of data. It is a blueprint that defines the types of data that can be stored in a database, as well as the relationships between those data. Our server contains the following schemas:



```
models
  Comment.js
  Like.js
  User.js
  Website.js
```

Fig 8.5 Server Models

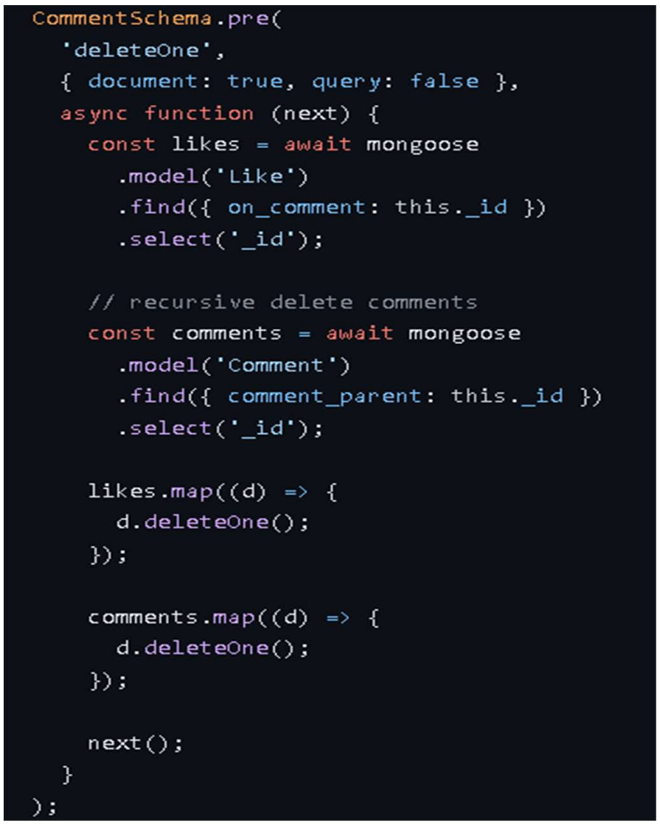
- **Comment:** This has the following fields in the schema with proper validation in every field. The Code "{timestamps: true}" automatically adds createdAt and updatedAt fields in the database and automatically handles them on every update.



```
const CommentSchema = mongoose.Schema(
  {
    by_user: {
      type: mongoose.SchemaTypes.ObjectId,
      ref: 'User',
      required: [true, 'Please provide created by User'],
    },
    on_website: {
      type: mongoose.SchemaTypes.ObjectId,
      ref: 'Website',
      required: [true, 'Please provide created on Website'],
    },
    comment_parent: {
      type: mongoose.SchemaTypes.ObjectId,
      ref: 'Comment',
      default: null,
    },
    comment_description: {
      type: String,
      required: [true, 'Please provide comment description'],
      trim: true,
    },
  },
  { timestamps: true }
);
```

Fig 8.6 Comments Schema

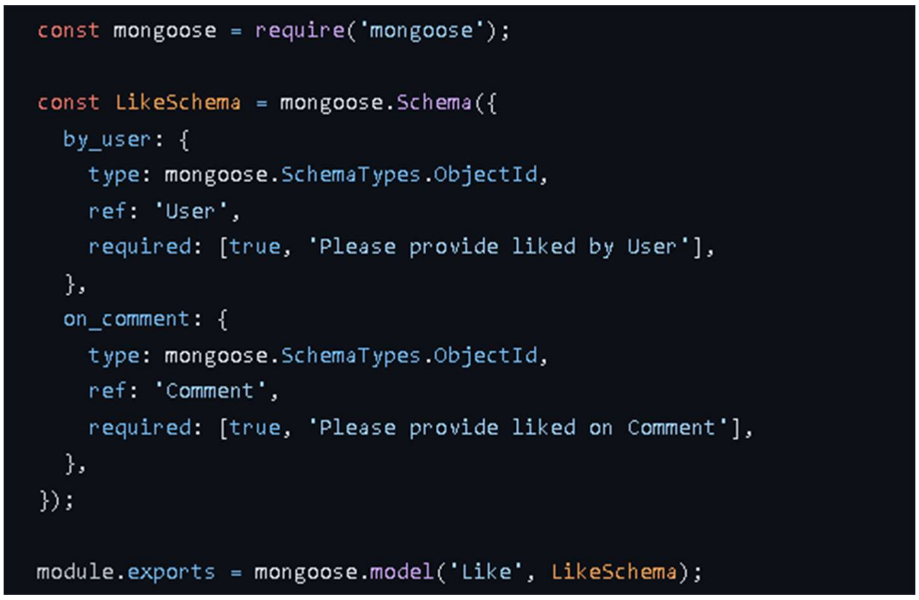
Comment's cleanup function deletes nested comments recursively and likes on them.



```
CommentSchema.pre(
  'deleteOne',
  { document: true, query: false },
  async function (next) {
    const likes = await mongoose
      .model('Like')
      .find({ on_comment: this._id })
      .select('_id');

    // recursive delete comments
    const comments = await mongoose
      .model('Comment')
      .find({ comment_parent: this._id })
      .select('_id');

    likes.map((d) => {
      d.deleteOne();
    });

    comments.map((d) => {
      d.deleteOne();
    });

    next();
  }
);
```

Fig 8.7 Comments Cleanup Function

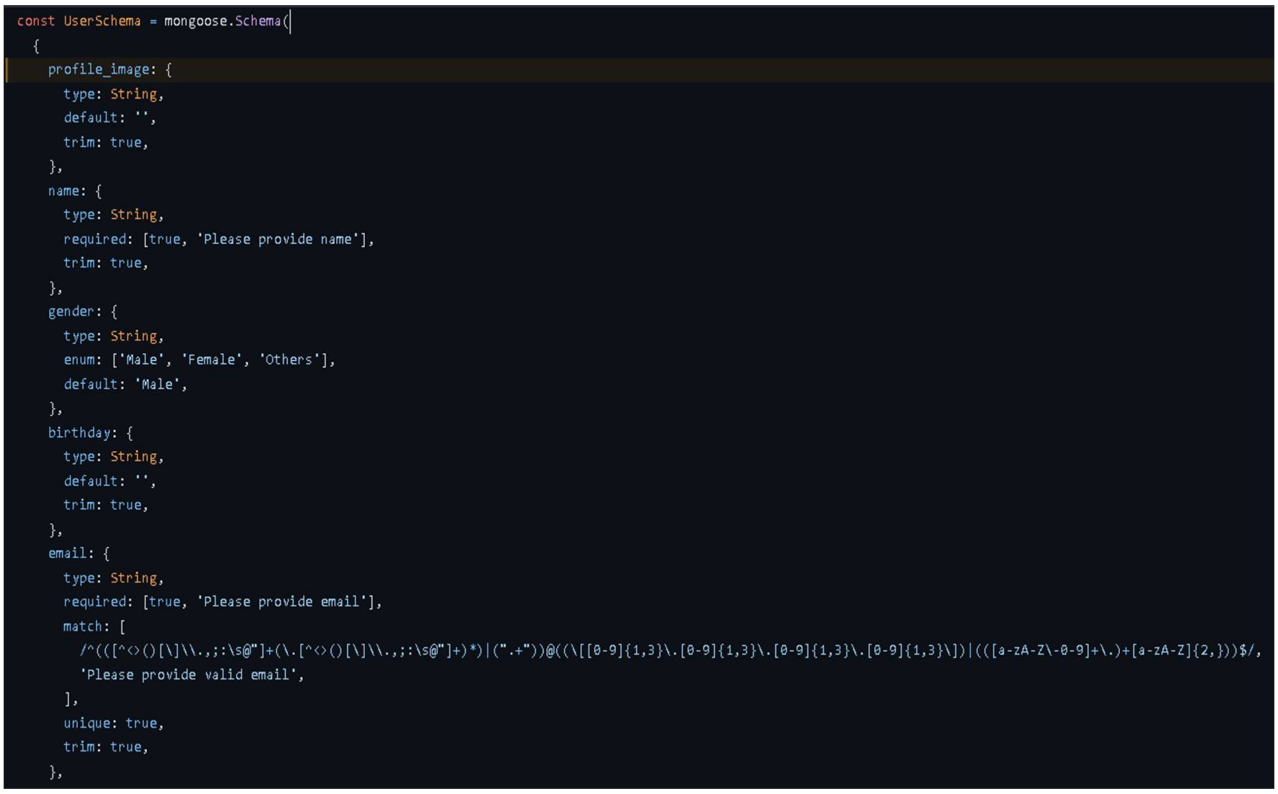
- **Like:** Tracks likes made by which user and on which comment.



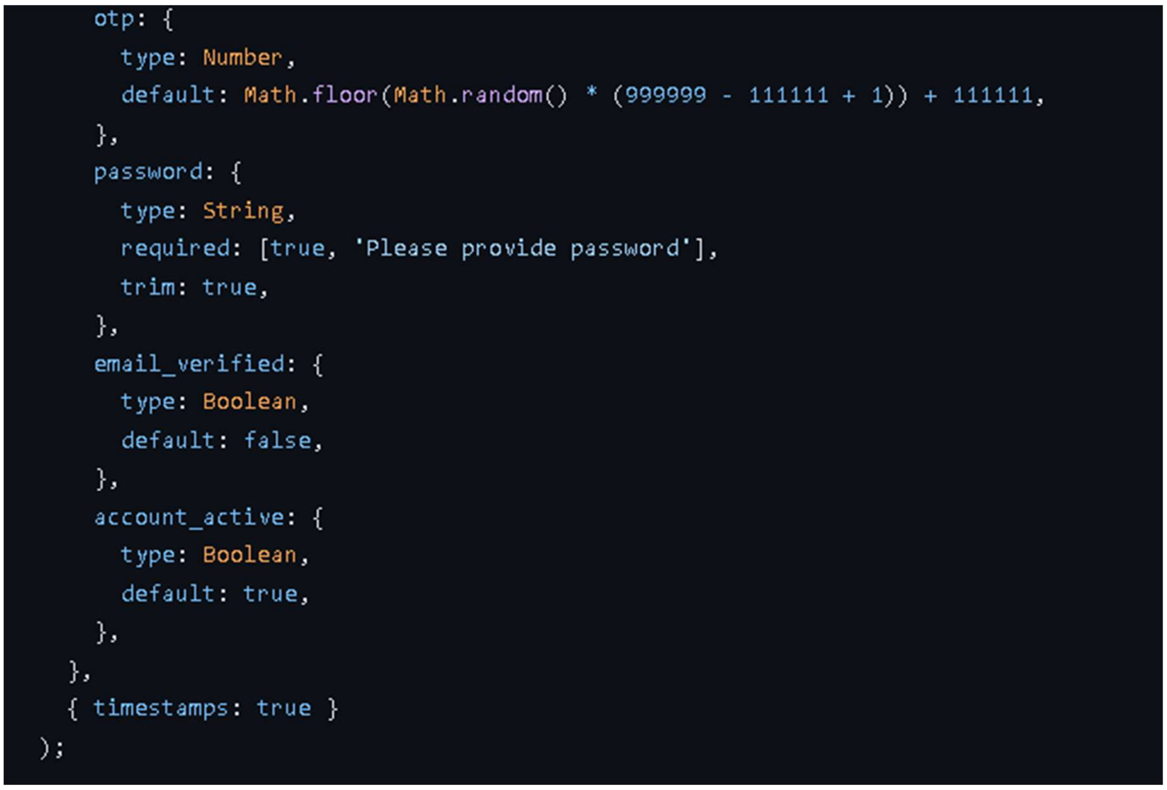
```
const mongoose = require('mongoose');

const LikeSchema = mongoose.Schema({
  by_user: {
    type: mongoose.SchemaTypes.ObjectId,
    ref: 'User',
    required: [true, 'Please provide liked by User'],
  },
  on_comment: {
    type: mongoose.SchemaTypes.ObjectId,
    ref: 'Comment',
    required: [true, 'Please provide liked on Comment'],
  },
});

module.exports = mongoose.model('Like', LikeSchema);
```

Fig 8.8 Like Schema

- **User:** Contains user schams with fields like profile_image, name, gender, birthday, email with regular expression validation, password (hashed using bcrypt), and other utility fields.



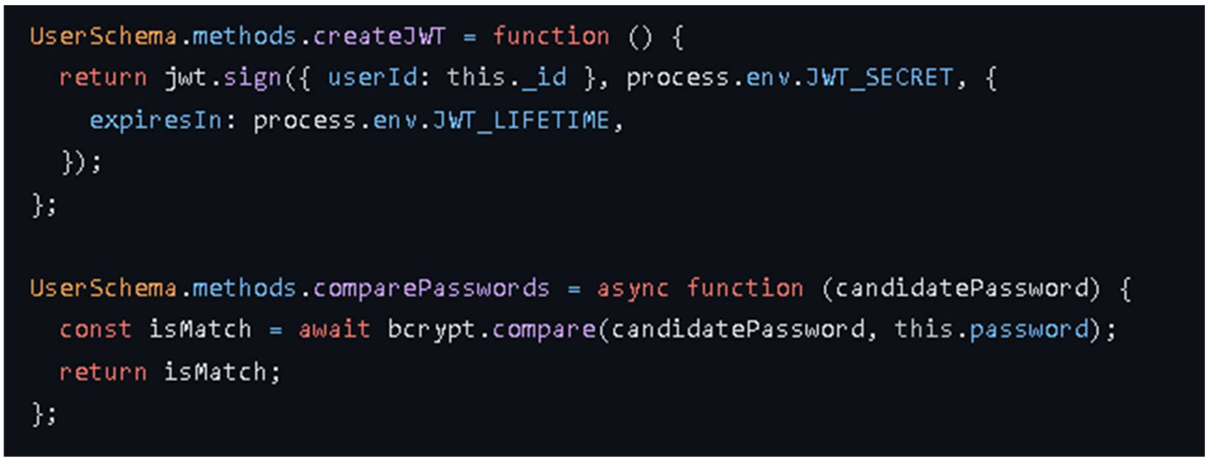
```
const UserSchema = mongoose.Schema(
  {
    profile_image: {
      type: String,
      default: '',
      trim: true,
    },
    name: {
      type: String,
      required: [true, 'Please provide name'],
      trim: true,
    },
    gender: {
      type: String,
      enum: ['Male', 'Female', 'Others'],
      default: 'Male',
    },
    birthday: {
      type: String,
      default: '',
      trim: true,
    },
    email: {
      type: String,
      required: [true, 'Please provide email'],
      match: [
        /^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/,
        'Please provide valid email',
      ],
      unique: true,
      trim: true,
    },
```



```
    otp: {
      type: Number,
      default: Math.floor(Math.random() * (999999 - 111111 + 1)) + 111111,
    },
    password: {
      type: String,
      required: [true, 'Please provide password'],
      trim: true,
    },
    email_verified: {
      type: Boolean,
      default: false,
    },
    account_active: {
      type: Boolean,
      default: true,
    },
  },
  { timestamps: true }
);
```

Fig 8.9 User Schema

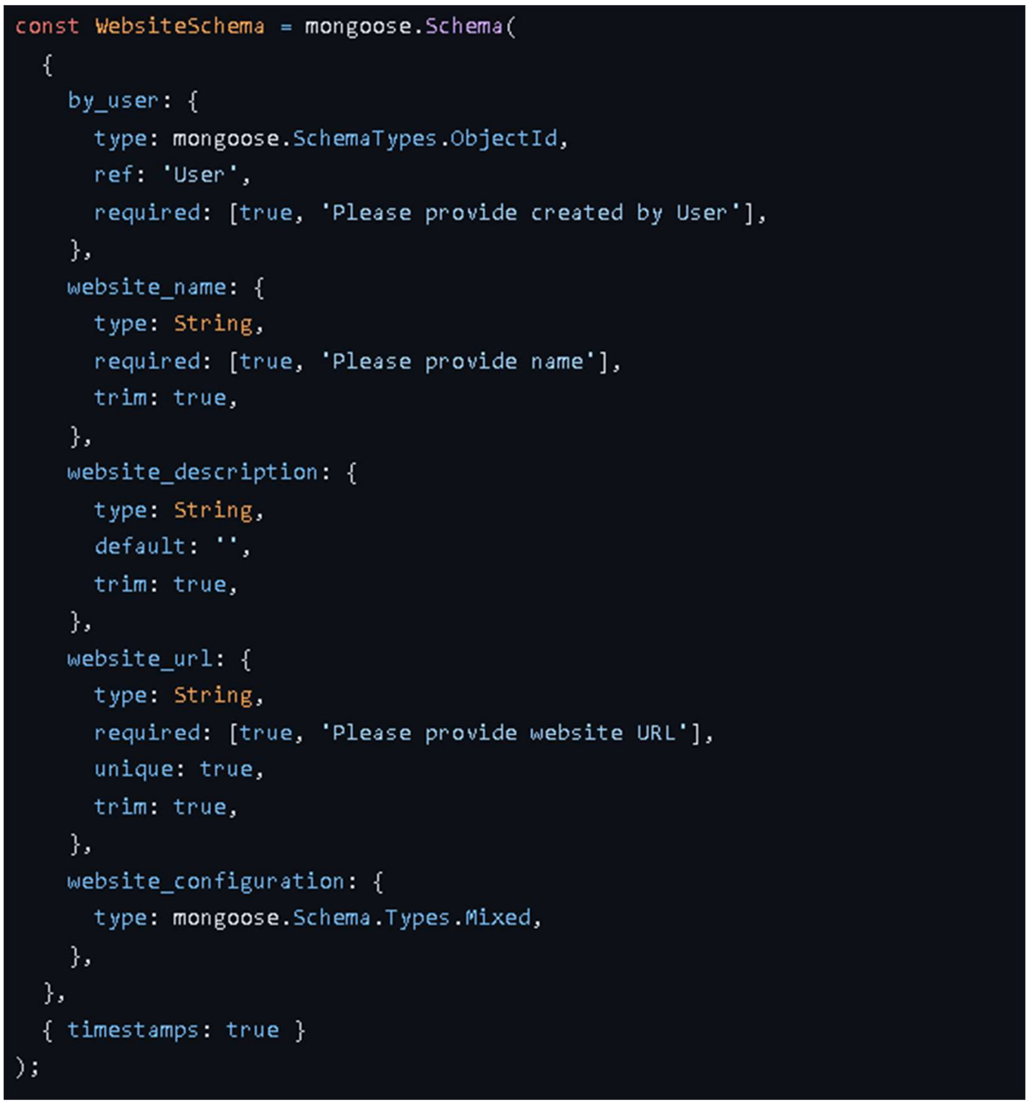
User Schema also contains the utility function **createJWT** which is used to create JSON web token to send to the user to uniquely identify itself with userId as payload and another function **comparePasswords** which compares user-provided and stored passwords.



```
UserSchema.methods.createJWT = function () {
  return jwt.sign({ userId: this._id }, process.env.JWT_SECRET, {
    expiresIn: process.env.JWT_LIFETIME,
  });
};

UserSchema.methods.comparePasswords = async function (candidatePassword) {
  const isMatch = await bcrypt.compare(candidatePassword, this.password);
  return isMatch;
};
```

Fig 8.10 User Schema Functions

- **Website:** This is the website schema that stores data of the developer's website.



```
const WebsiteSchema = mongoose.Schema(
  {
    by_user: {
      type: mongoose.SchemaTypes.ObjectId,
      ref: 'User',
      required: [true, 'Please provide created by User'],
    },
    website_name: {
      type: String,
      required: [true, 'Please provide name'],
      trim: true,
    },
    website_description: {
      type: String,
      default: '',
      trim: true,
    },
    website_url: {
      type: String,
      required: [true, 'Please provide website URL'],
      unique: true,
      trim: true,
    },
    website_configuration: {
      type: mongoose.Schema.Types.Mixed,
    },
  },
  { timestamps: true }
);
```

Fig 8.11 Website Schema

7. **Public:** This folder contains an initialization JavaScript file which is loaded on an initial request from gimme comment client code and also contains a client build folder containing all CSS and JS files of gimme comments clients.
8. **Server.js:** This file is the main JavaScript file for the server. It is typically used to initialize JavaScript libraries and run JavaScript code.

8.2.3 Data flow diagram of Authentication

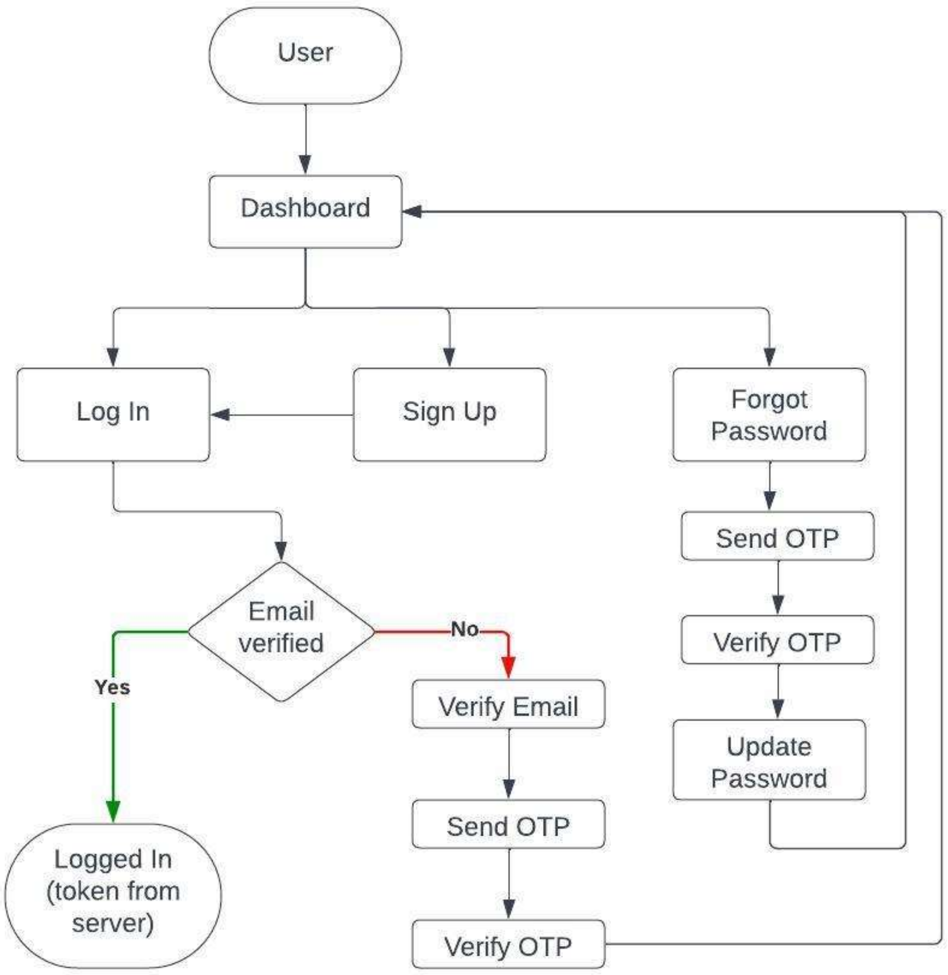


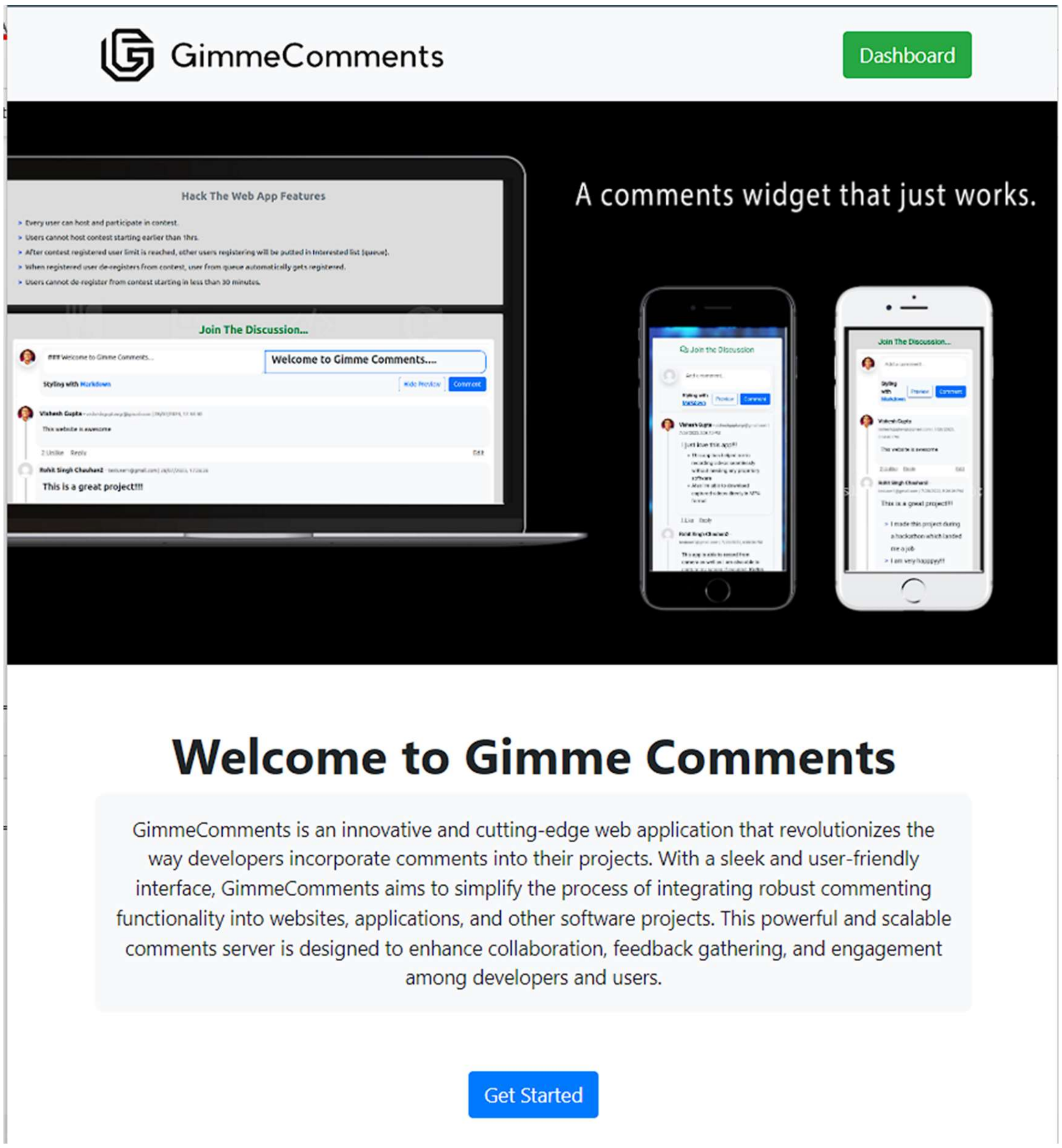
Fig. 8.12 Data flow diagram of User Authentication

8.3 Gimme Comments Admin

Gimme Comments Admin is hosted on Vercel.com and is the website that provides commenting service to users of our project. The link to the admin website is gimme-comments-admin. vertical. app. It acts as the front end of the project and is connected to the MongoDB database.

It contains the following important segments:

1. **Home Page:** This is the page that comes first when some user visits the website. It has a dashboard button that takes the user to Dashboard if he is signed in or takes him to the sign-in page if he isn't logged in.



A comments widget that just works.

Hack The Web App Features

- Every user can host and participate in contest.
- Users cannot host contest starting earlier than 1hrs.
- After contest registered user limit is reached, other users registering will be putted in Interested list (queue).
- When registered user de-registers from contest, user from queue automatically gets registered.
- Users cannot de-register from contest starting in less than 30 minutes.

Welcome to Gimme Comments....

Vishesh Gupta

This website is awesome

Rohit Singh Chauhan2

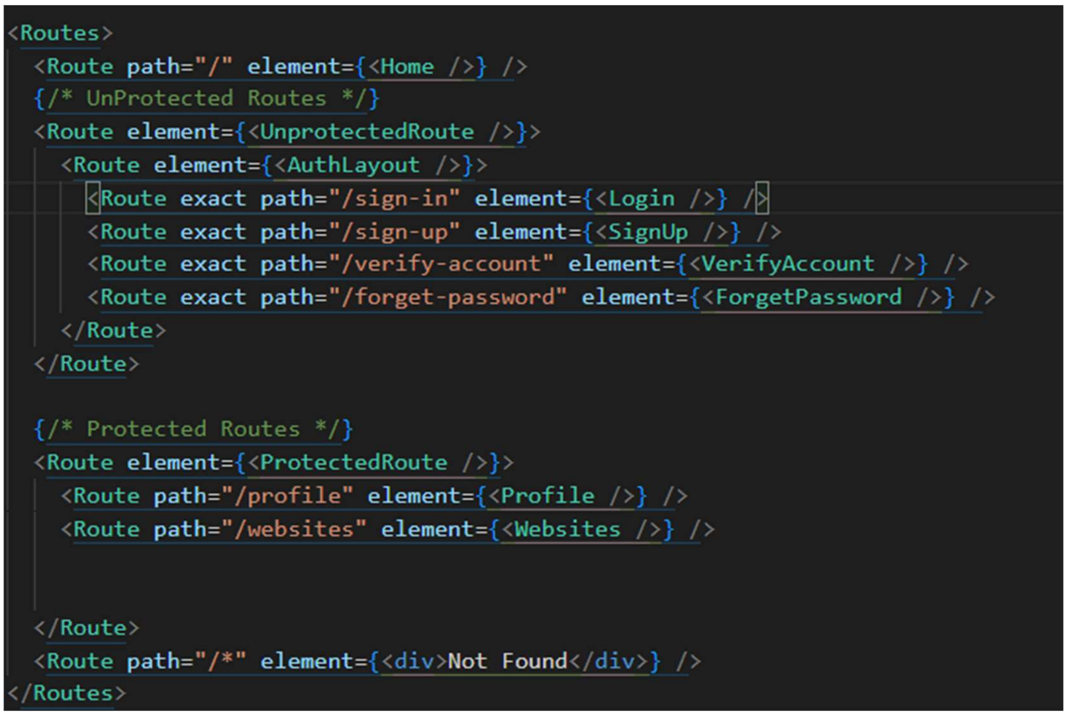
This is a great project!!!

Welcome to Gimme Comments

GimmeComments is an innovative and cutting-edge web application that revolutionizes the way developers incorporate comments into their projects. With a sleek and user-friendly interface, GimmeComments aims to simplify the process of integrating robust commenting functionality into websites, applications, and other software projects. This powerful and scalable comments server is designed to enhance collaboration, feedback gathering, and engagement among developers and users.

Fig 8.14 Admin Home Page

2. **Routing:** Routing in React applications is usually handled using a library called React Router. It allows you to add different "routes" to your application, which correspond to different components being rendered based on the current URL.



```
<Routes>
  <Route path="/" element={<Home />} />
  {/* UnProtected Routes */}
  <Route element={<UnprotectedRoute />}>
    <Route element={<AuthLayout />}>
      <Route exact path="/sign-in" element={<Login />} />
      <Route exact path="/sign-up" element={<SignUp />} />
      <Route exact path="/verify-account" element={<VerifyAccount />} />
      <Route exact path="/forget-password" element={<ForgetPassword />} />
    </Route>
  </Route>

  {/* Protected Routes */}
  <Route element={<ProtectedRoute />}>
    <Route path="/profile" element={<Profile />} />
    <Route path="/websites" element={<Websites />} />


  </Route>
  <Route path="/*" element={<div>Not Found</div>} />
</Routes>
```

Fig 8.15 Admin Routing

The Gimme Comments admin contains two types of routes:

1. **Unprotected Route:** This is the type of route that is accessed when the user isn't signed in. In our project, the unprotected route gives access to the following components :
   - **Sign-in:** This is the part that shows the sign-in feature of the website. The feature on the website looks as follows:

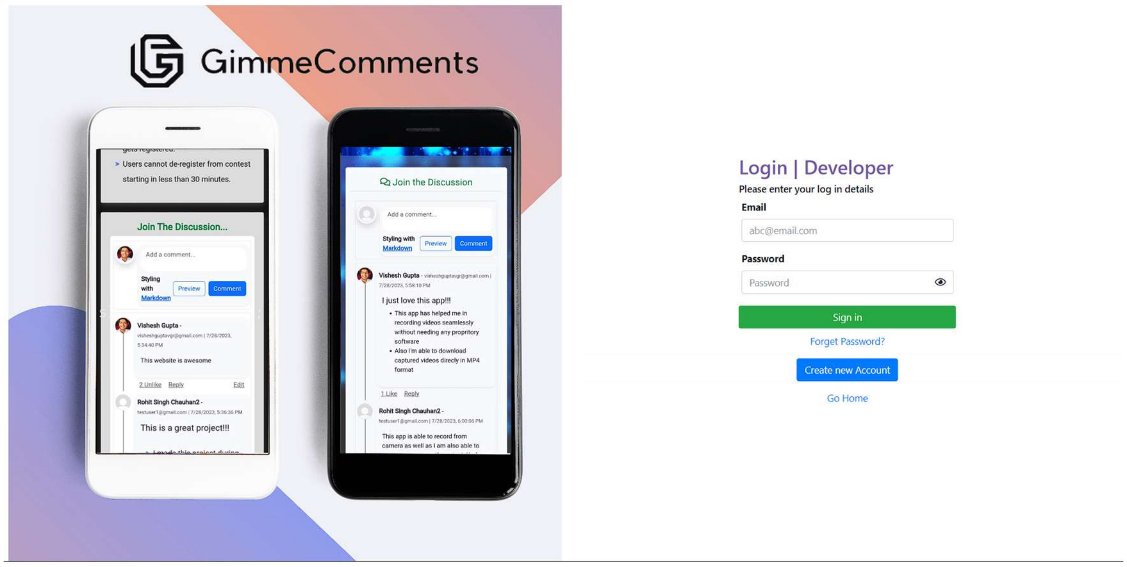


Fig 8.16 Admin Login Page

- **Signup:** Signup means to register a new client to the website. The website has the following way to implement the signup mechanism.

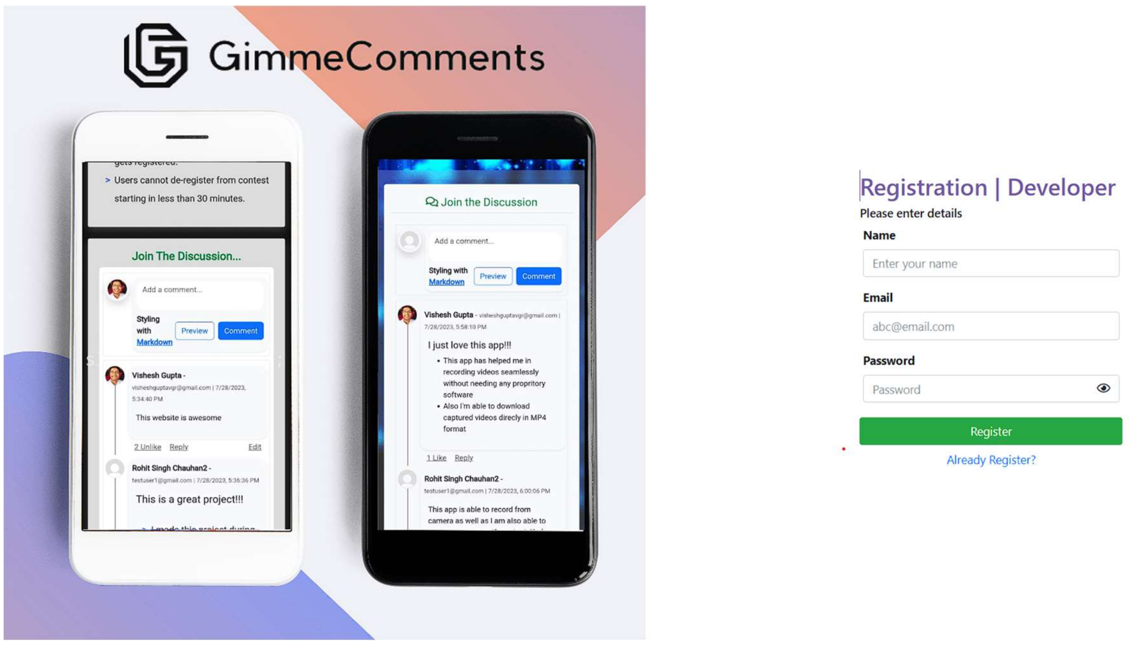


Fig 8.17 Admin Registration Page

- **Forgot Password:** If a user doesn't remember his/her password, he can click on forgot password button on the sign-in page to get to the reset password option using Otp.

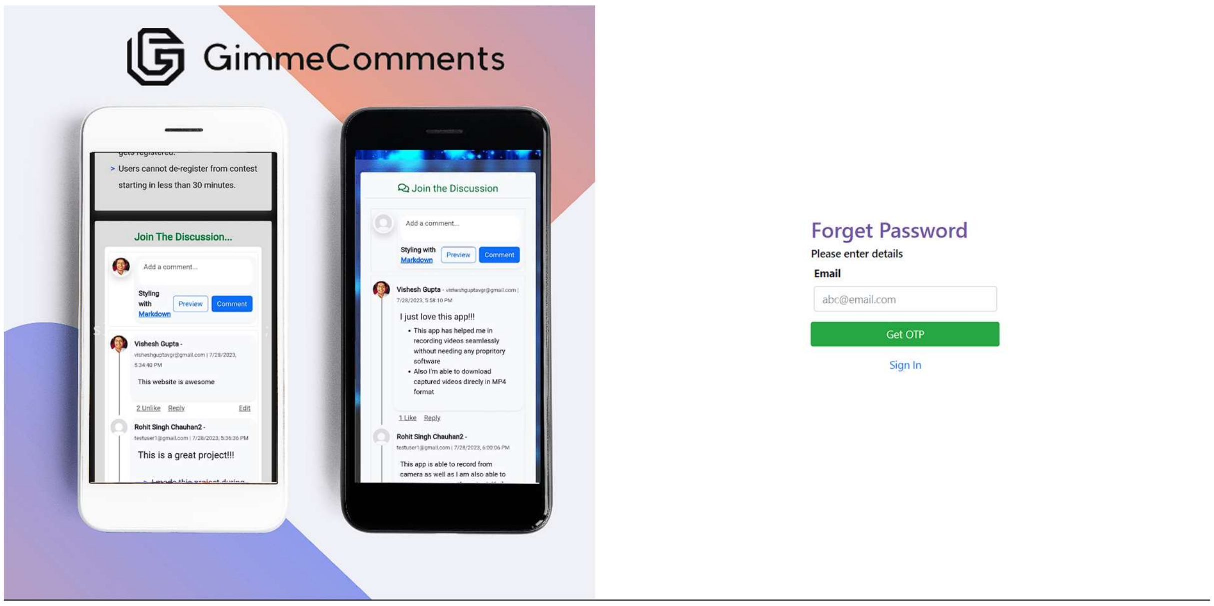
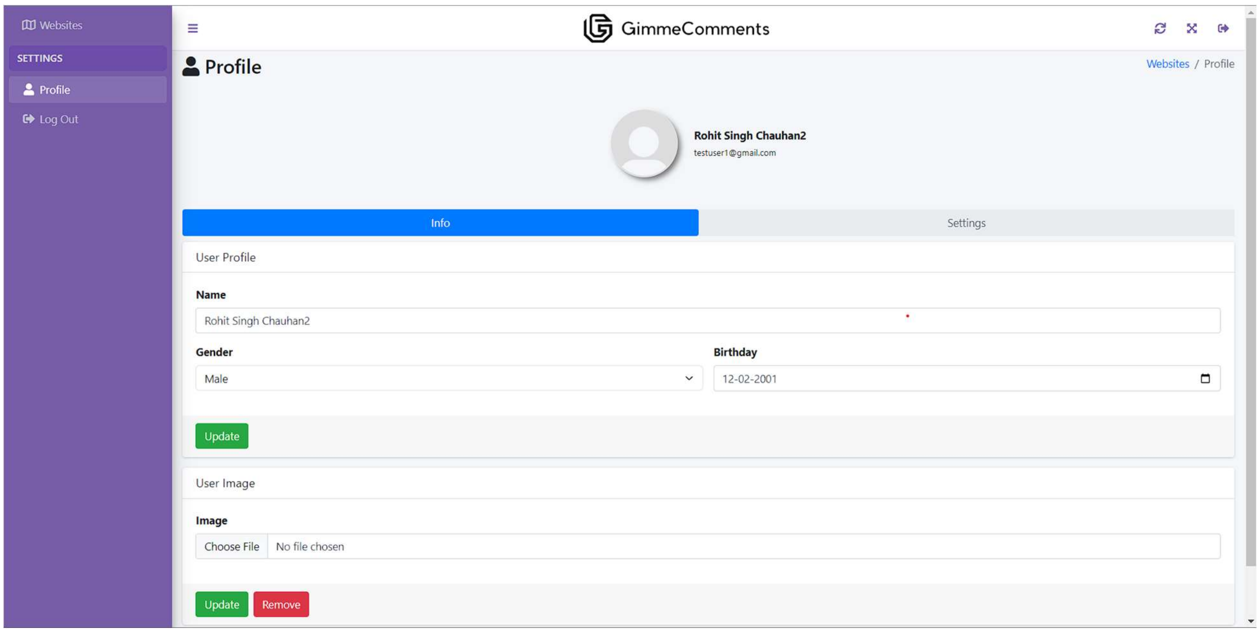


Fig 8.18 Admin Forget Password Page

2. **Protected Route:** This is the type of route that is accessed when the user is signed in. In our project, the protected route gives access to the following components :
   Profile: The profile contains the profile of the user registered to the website. Here we can edit the information of the user and also access the settings of the user account.

   - **Info section in Profile:** We can change name, gender, and birthday. Upload the user profile image in this section.



Rohit Singh Chauhan2
testuser1@gmail.com

User Profile

Name: Rohit Singh Chauhan2

Gender: Male

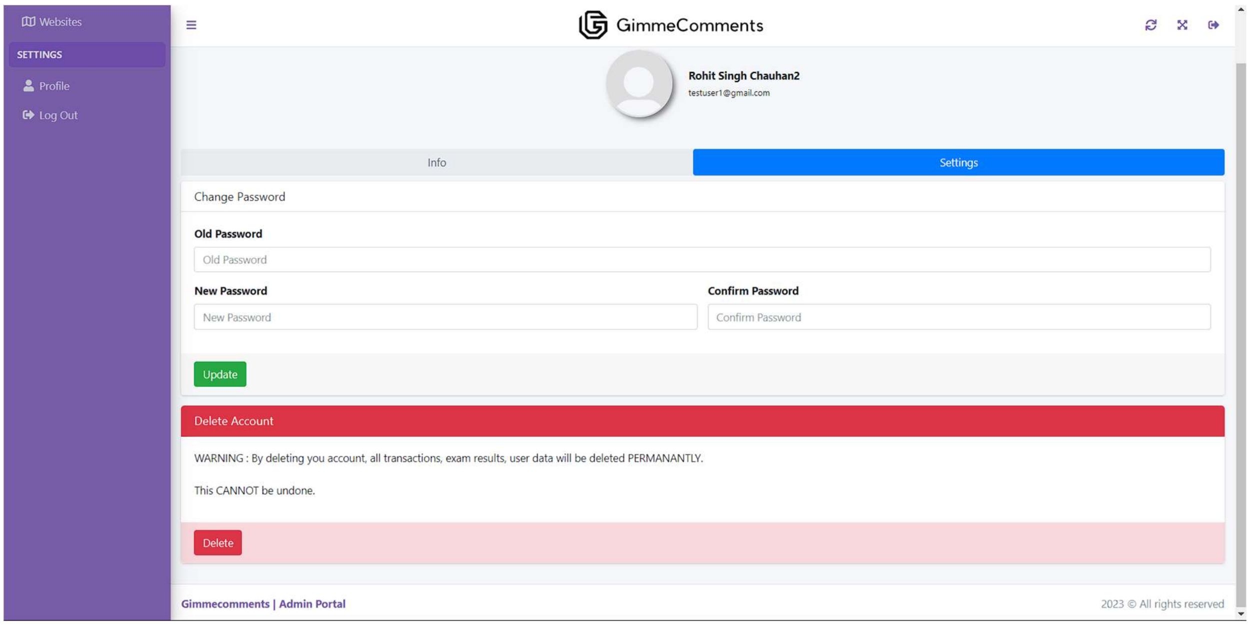
Birthday: 12-02-2001

User Image

Image: Choose File No file chosen

Fig 8.19 Admin Profile Info Section

   - **Settings section in Profile:** We can change the password or delete the user account permanently here. To update the password we need to provide an old password for the user and also add a new password as well as confirm it. To delete the current users we can click on the delete button and on proceeding with the popup, the current user will get deleted.



Rohit Singh Chauhan2
testuser1@gmail.com

Change Password

Old Password: Old Password

New Password: New Password

Confirm Password: Confirm Password

Delete Account

WARNING : By deleting you account, all transactions, exam results, user data will be deleted PERMANANTLY.

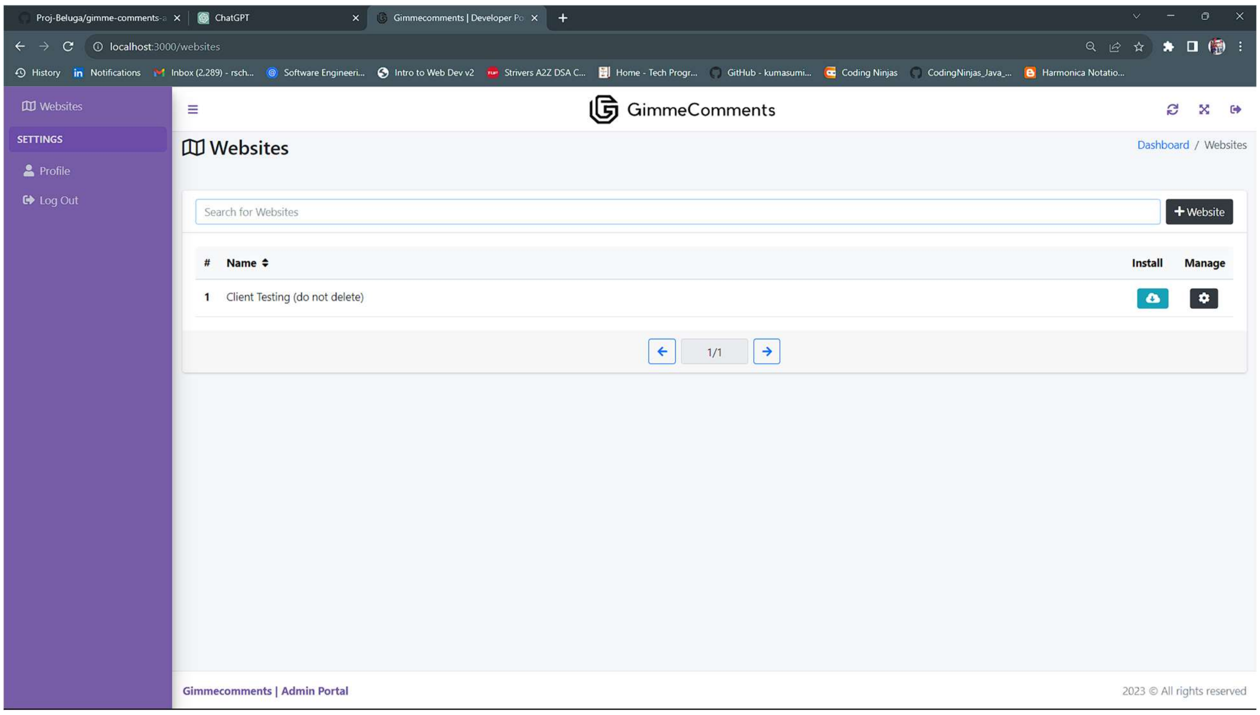
This CANNOT be undone.

Gimmecomments | Admin Portal

2023 © All rights reserved

Fig 8.20 Admin Profile Settings Section

- **Websites/Dashboard:** The website page acts as the dashboard of the Gimme Comments admin. It contains the list of all the websites that the user has registered on the server. It uses the APIs to access website data from the database.



Websites

#	Name	Install	Manage
1	Client Testing (do not delete)		

Gimmecomments | Admin Portal

2023 © All rights reserved

Fig 8.21 Admin Websites/Dashboard Page

The website component contains the following parts:

1. **Add Website button:** It is used to add new user websites to the server.

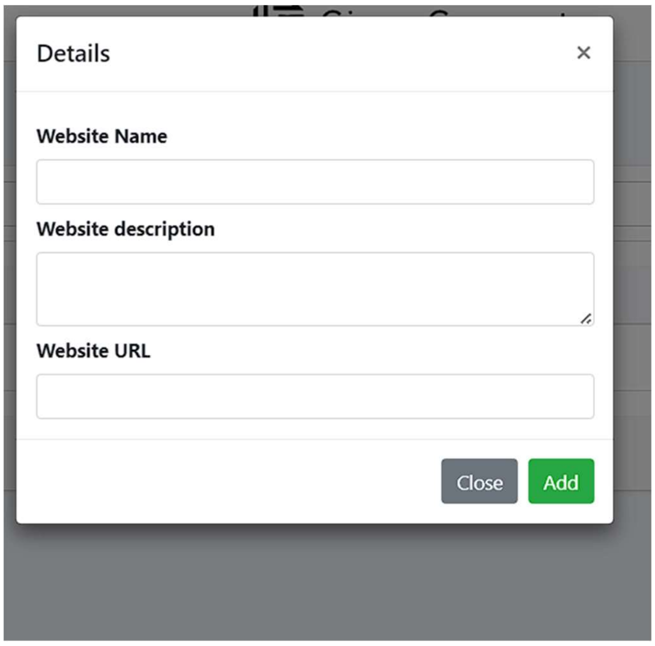
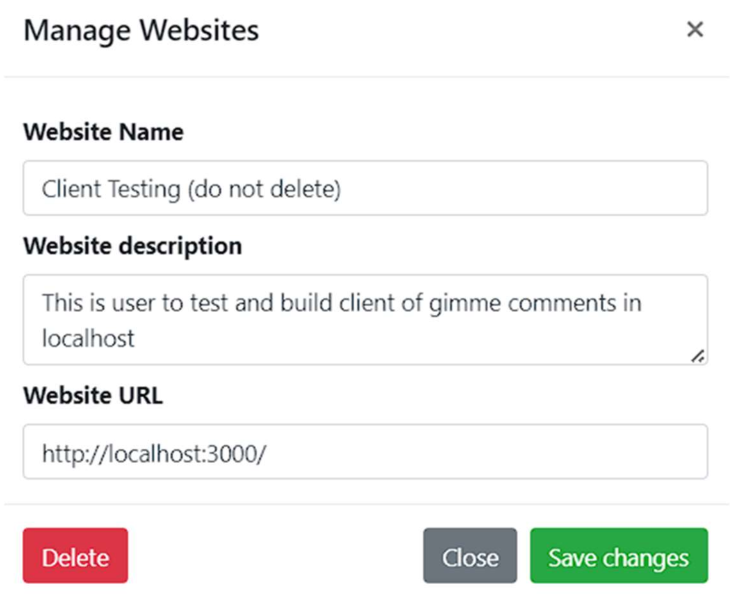


Fig 8.22 Add Website Modal

2. **Manage Website:** It is used to edit the existing user websites in the server.



Website Name

Client Testing (do not delete)

Website description

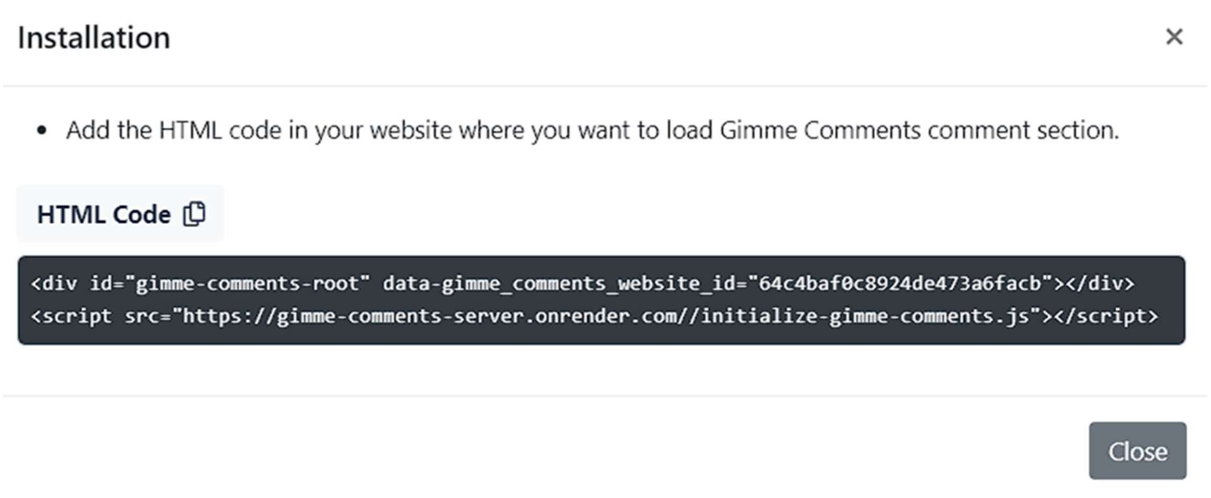
This is user to test and build client of gimme comments in localhost

Website URL

http://localhost:3000/

Fig 8.23 Manage Website Modal

Install Button: It provides the code that the user needs to insert in their actual code so that they can get commenting service on their website.


Installation

- Add the HTML code in your website where you want to load Gimme Comments comment section.

HTML Code

```
<div id="gimme-comments-root" data-gimme_comments_website_id="64c4baf0c8924de473a6facb"></div>
<script src="https://gimme-comments-server.onrender.com//initialize-gimme-comments.js"></script>
```

Fig 8.24 Installation Instructions

8.4 Gimme Comments Client

Gimme Comments Client is hosted on Gimme Comment Server as built and is the comment box that provides commenting service to users of our project. It requires only a few lines of code to initialize the Gimme Comments client app.

It contains the following important segments:

1. **Authentication:** The app contains normal Authentication flow provided by the server including Register, Login, Account verification, and Forgot Password functionalities.

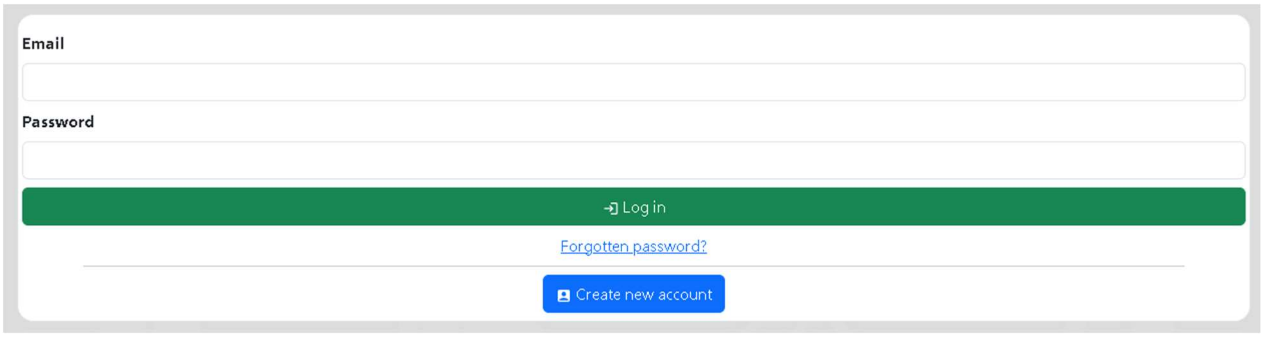


Fig 8.25 Client Authentication

2. **Create Comment:** This section serves as the main component where logged-in users can create comments and also this is integrated with markdown support using React Markdown library.

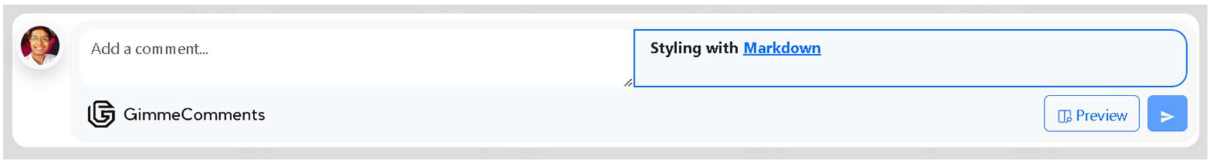
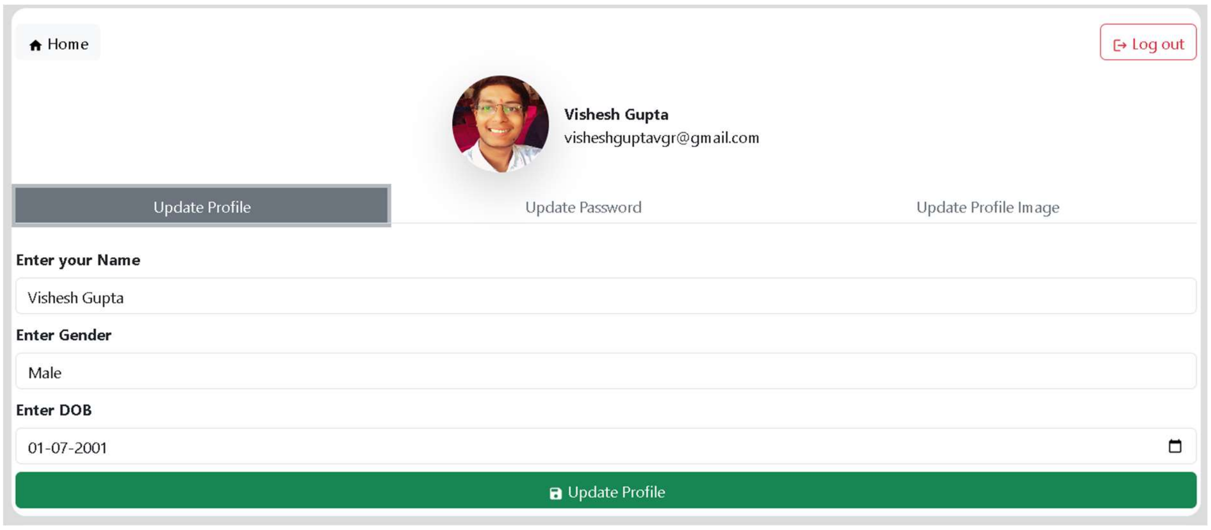


Fig 8.26 Client Create Comment

3. **Profile:** The profile page features three sections for various purposes including:
   - Update Profile: Here the user can update user profile details including Name, Gender, and DOB.



Vishesh Gupta
visheshguptavgr@gmail.com

Enter your Name
Vishesh Gupta
Enter Gender
Male
Enter DOB
01-07-2001

Fig 8.27 Client Update Profile Section

   - **Update Password:** In this section, user can update their password by entering the old password and new password to update with.

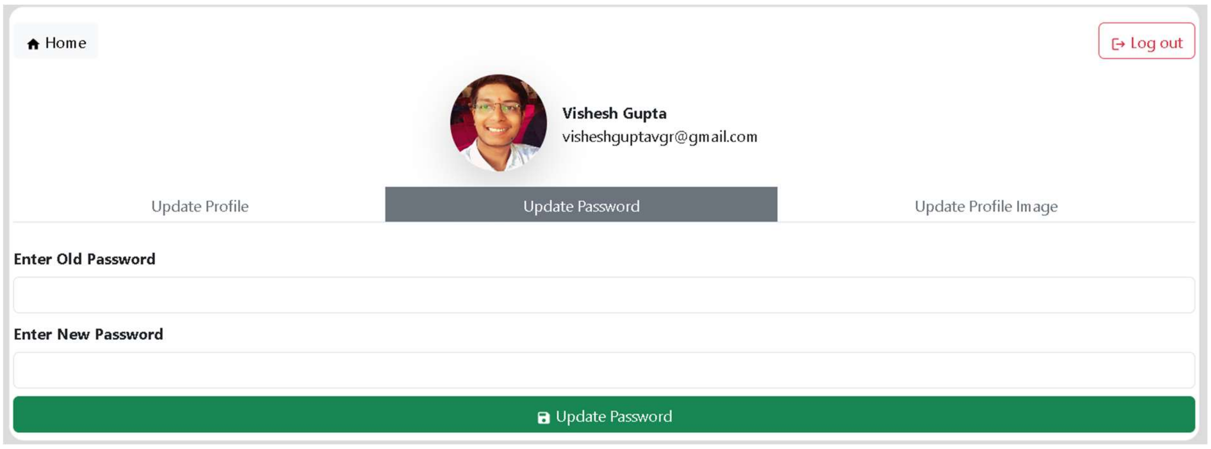


Fig 8.28 Client Update Password Section

- **Update Profile Image:** Using this section user can change his/her profile image and also remove it from local storage.

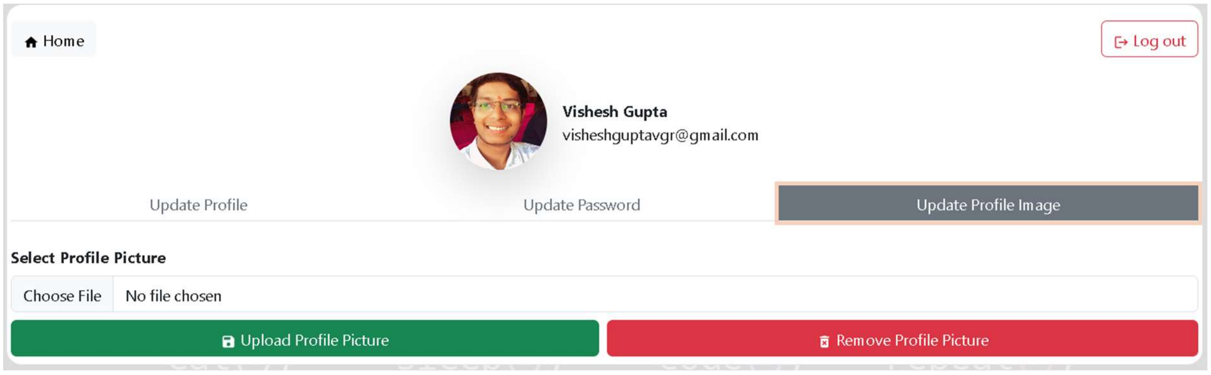
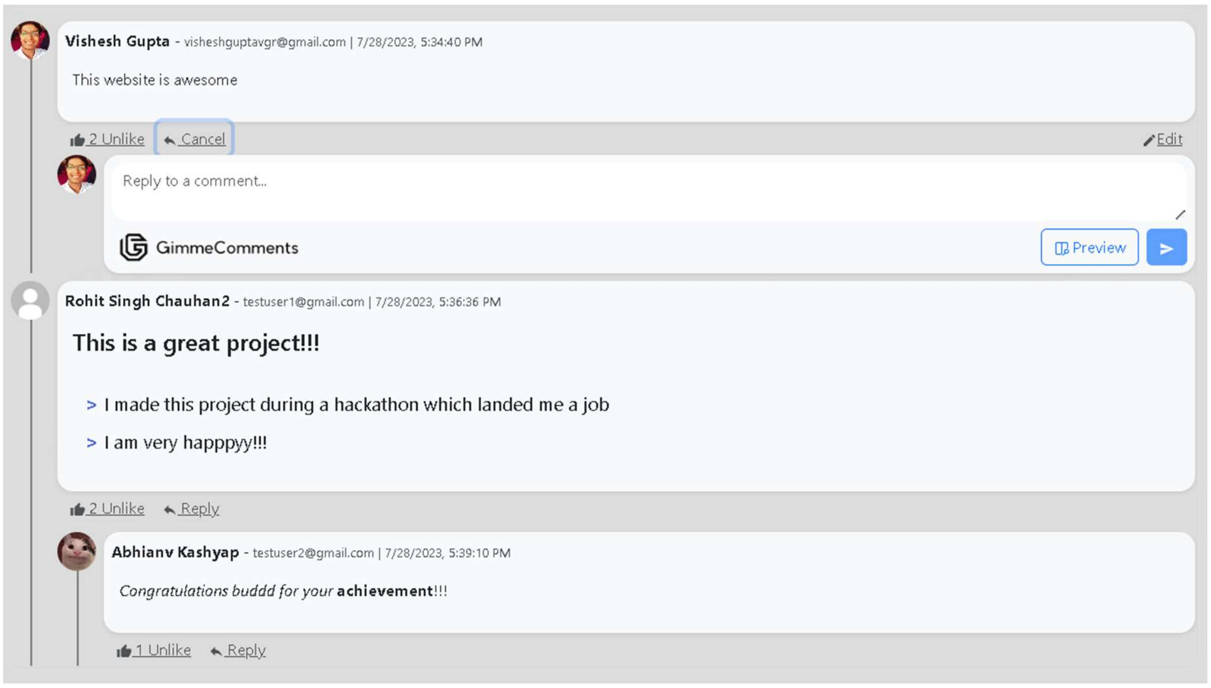


Fig 8.29 Client Update Profile Image Section

4. **All Comments:** Just below Create Comment section, all comments which are posted on this website are listed, where logged-in users can add/remove like, reply to comments, and edit their comments by updating or deleting them.


Vishesh Gupta - visheshguptavgr@gmail.com | 7/28/2023, 5:34:40 PM

This website is awesome

2 Unlike Cancel Edit

Reply to a comment...

GimmeComments Preview

Rohit Singh Chauhan2 - testuser1@gmail.com | 7/28/2023, 5:36:36 PM

This is a great project!!!

> I made this project during a hackathon which landed me a job

> I am very happpyy!!!

2 Unlike Reply

Abhianv Kashyap - testuser2@gmail.com | 7/28/2023, 5:39:10 PM

Congratulations buddd for your **achievement**!!!

1 Unlike Reply

Fig 8.30 Client All Comments Section

CHAPTER 9
FILES IN THE APPLICATION

9.1. GIMME COMMENTS (Modules)

1. App.js

```
import { BrowserRouter } from 'react-router-dom'
import Routing from './Routes/Routing'
import { StoreProvider } from './Contexts/StoreContext'
import './App.css'

function App() {
  return (
    <StoreProvider>
      <BrowserRouter>
        <Routing />
      </BrowserRouter>
    </StoreProvider>
  )
}

export default App
```

2. Routing.js

```
<Routes>
        <Route path="/" element={<Home />} />
        {/* UnProtected Routes */}
        <Route element={<UnprotectedRoute />}>
          <Route element={<AuthLayout />}>
            <Route exact path="/sign-in" element={<Login />} />
            <Route exact path="/sign-up" element={<SignUp />} />
            <Route exact path="/verify-account" element={<VerifyAccount />}
/>
            <Route exact path="/forget-password" element={<ForgetPassword
/>} />
          </Route>
        </Route>

        {/* Protected Routes */}
        <Route element={<ProtectedRoute />}>
          <Route path="/profile" element={<Profile />} />
          <Route path="/websites" element={<Websites />} />
        </Route>
        <Route path="/*" element={<div>Not Found</div>} />
      </Routes>
```

3. Axios.js

```
import axios from 'axios'

let instance

export default function getAxiosInstance() {
  if (!instance) {
    let token = localStorage.getItem('gimme_comment_access_token')

    // if (!token) return axios

    instance = axios.create({
      baseURL: process.env.REACT_APP_BASE_URL,
      headers: {
        Authorization: `Bearer ${token}`,
      },
    })

    //validate response
    instance.interceptors.response.use(
      (response) => {
        return response
      },
      (error) => {
        if (error.response.status === 401) {
          localStorage.removeItem('gimme_comment_access_token')

          window.location.reload()
        }
        return Promise.reject(error)
      },
    )
  }
  return instance
}
```

4.**Profile.jsx**

```
import React from "react";
import { Link } from "react-router-dom";
import ProfileComponent from
"../../Components/ProfileComponent/ProfileComponent";

const Profile = () => {
  return (
    <>
      <div className="content-wrapper">
        <div className="content-header">
          <div className="container-fluid">
            <div className="row mb-2">
              <div className="col-sm-6">
                <h1 className="m-0">
                  <i className="nav-icon fas fa-user me-2" />
                  Profile
                </h1>
              </div>
              <div className="col-sm-6">
                <ol className="breadcrumb float-sm-right">
                  <li className="breadcrumb-item">
                    <Link to="/websites">Websites</Link>
                  </li>
                  <li className="breadcrumb-item active">Profile</li>
                </ol>
              </div>
            </div>
            <ProfileComponent />
          </div>
        </div>
      </div>
    </>
  );
};

export default Profile;

return (
    <>
      <div className="row justify-content-center mt-5">
        <div className="col-md-12">
          <div className="mb-5 d-flex align-items-center justify-content-
center gap-2 gap-sm-4 flex-wrap">
            <img
              src={userData.profile_image || default_profile_image}
              alt="profile"
              style={{
                width: "100px",
                height: "100px",
                borderRadius: "50%",
                objectFit: "cover",
                boxShadow: " 5px 5px 8px -1px #777",
              }}
            />
            <div>
              <strong>{userData.name}</strong>
              <br />
              <small>{userData.email}</small>
            </div>
          </div>
          <nav className="nav nav-pills flex-column flex-sm-row mb-2 bg-body-
secondary">
            <button
              className={`flex-sm-fill text-sm-center nav-link ${
                navTab === 1 && "active"
              }`}
              onClick={() => setNavTab(1)}
            >
              Info
            </button>
            <button
              className={`flex-sm-fill text-sm-center nav-link ${
                navTab === 2 && "active"
              }`}
              onClick={() => setNavTab(2)}
            >
              Settings
            </button>
          </nav>

          {navTab === 1 && (
            <InfoNavTab
              userData={userData}
              setUserData={setUserData}
              setIsLoading={setIsLoading}
              fetchProfileData={fetchProfileData}
            />
          )}
          {navTab === 2 && (
            <SettingsNavTab
              userData={userData}
              setUserData={setUserData}
              setIsLoading={setIsLoading}
              fetchProfileData={fetchProfileData}
            />
          )}
        </div>
      </div>
    </>
  );
```

5. Home.jsx

```
import React from 'react'
import { useNavigate } from 'react-router-dom'

import gimmecomments_logo from '../../assets/gimmecomments_logo.png'
import HomeImage from '../../assets/My project-1.png'
const access_token = localStorage.getItem('gimme_comment_access_token')

const Home = () => {
  const navigate = useNavigate()

  function handleClick(e) {
    e.preventDefault()

    if (access_token) {
      navigate('/websites')
    } else {
      navigate('/sign-in')
    }
  }

  function handleStart() {
    navigate('/sign-up')
  }
  return (
    <>
      {/* Navbar  */}
      <nav className="navbar navbar-light bg-light">
        <div className="container ">
          <a href="/" className="navbar-brand font-weight-bold">
            <img
              src={gimmecomments_logo}
              alt="logo"
              style={{ height: '50px', cursor: 'pointer' }}
            />
          </a>
          <form className="d-flex">
            <button
              className="btn btn-success"
              type="submit"
              onClick={handleClick}
            >
              Dashboard
            </button>
          </form>
        </div>
      </nav>

      {/* Body  */}
      <img
        src={HomeImage}
        alt="home"
        style={{ maxHeight: '80vh', width: '100%', objectFit: 'cover' }}
      />

      <div className="container text-center mt-5 ">
        <h1 className="font-weight-bold">Welcome to Gimme Comments</h1>

        <div className="container p-3 rounded-3 bg-light text-dark">
          Gimme Comments is an innovative and cutting-edge web application
that revolutionizes the way developers incorporate comments into their
projects. With a sleek and user-friendly interface, Gimme  Comments aims
          to simplify the process of integrating robust commenting
functionality into websites, applications, and other software projects.
This powerful and scalable comments server is designed to enhance
collaboration, feedback gathering, and engagement among developers and
          users.
        </div>
      </div>

      {/* Footer */}

      <div className="container mt-5 text-center">
        <button type="button" class="btn btn-primary"
onClick={handleStart}>
          Get Started
        </button>
      </div>
    </>
  )
}

export default Home
```

6 .Websites.jsx

1. FetchWebsiteData:

```
const fetchWebsiteData = async () => {
    try {
      setIsLoading(true);
      const response = await axios().get(`/api/v1/websites`);

      setWebsiteData(response.data.websites);
      console.log(response.data.websites);
      setIsLoading(false);
    } catch (error) {
      console.log(error);
      Toast.fire({
        icon: "error",
        title: error.response.data ? error.response.data.msg :
error.message,
      });
      setIsLoading(false);
    }
  };

  useEffect(() => {
    fetchWebsiteData();
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, [setIsLoading]);

  // FILTERING DATA IN ONE GO
  useEffect(() => {
    // filtering according to search term filter
    const tempCourseCategoriesData = websiteData;
    const tempSearchTermFilterData = tempCourseCategoriesData.filter(
      (category) => {
        if (searchTermFilter === "") {
          return true;
        } else {
          if (
            category["website_name"]
              .toLowerCase()
              .includes(searchTermFilter.toLowerCase())
          ) {
            return true;
          } else {
            return false;
          }
        }
      }
    );

    setFilteredData(tempSearchTermFilterData);
  }, [websiteData, searchTermFilter]);

  // sorting searchTermFilteredData according to sortingOn and
sortingMethod
  useEffect(() => {
    const tempFilteredData = filteredData;

    const asc = (a, b) => {
      if (
        String(a[sortingOn]).toLowerCase() >
String(b[sortingOn]).toLowerCase()
      )
        return 1;
      else if (
        String(a[sortingOn]).toLowerCase() <
String(b[sortingOn]).toLowerCase()
      )
        return -1;
      else return 0;
    };
    const des = (a, b) => {
      if (
        String(a[sortingOn]).toLowerCase() <
String(b[sortingOn]).toLowerCase()
      )
        return 1;
      else if (
        String(a[sortingOn]).toLowerCase() >
String(b[sortingOn]).toLowerCase()
      )
        return -1;
      else return 0;
    };

    tempFilteredData.sort(sortingMethod ? asc : des);
    setSortedData(tempFilteredData);
  }, [filteredData, sortingMethod, sortingOn]);

  // paginating sortedData accordint to currentPage and rowsPerPage
  useEffect(() => {
    const indexOfLastUser = currentPage * rowsPerPage;
    const indexOfFirstUser = indexOfLastUser - rowsPerPage;
    setPaginatedData(sortedData.slice(indexOfFirstUser, indexOfLastUser));
  }, [currentPage, sortedData, rowsPerPage, sortingMethod]);

  const prevPage = () => {
    if (currentPage > 1) setCurrentPage(currentPage - 1);
  };

  const nextPage = () => {
    const totalPage = Math.ceil(sortedData.length / rowsPerPage);
    if (currentPage < totalPage) setCurrentPage(currentPage + 1);
  };
```

2. Manage Websites and Adding new websites:

```
const ManageWebsiteModal = ({ data, fetchWebsiteData }) => {
  const CloseButton = useRef();
  const { setIsLoading } = useStore();
  const initialLocalData = {
    website_name: "",
    website_description: "",
    website_url: "",
  };

  const [localData, setLocalData] = useState(initialLocalData);

  useEffect(() => {
    if (!data) return;

    setLocalData(data);
  }, [data]);

  const handleAddWebsite = async () => {
    try {
      setIsLoading(true);
      const res = await axios().post(`/api/v1/websites`, localData);
      Toast.fire({
        icon: "success",
        title: "Website added",
      });
      setLocalData(initialLocalData);
      CloseButton.current.click();
      setIsLoading(false);
      fetchWebsiteData();
    } catch (error) {
      console.log(error);
      Toast.fire({
        icon: "error",
        title: error.response.data ? error.response.data.msg : error.message,
      });
      setIsLoading(false);
    }
  };
```

3. Handle Website Updates:

```
const handleUpdateWebsite = async () => {
    try {
      setIsLoading(true);
      await axios().patch(`/api/v1/websites/${data._id}`, localData);
      Toast.fire({
        icon: "success",
        title: "Website updated",
      });

      fetchWebsiteData();
      CloseButton.current.click();
      setIsLoading(false);
    } catch (error) {
      console.log(error);
      Toast.fire({
        icon: "error",
        title: error.response.data ? error.response.data.msg : error.message,
      });
      setIsLoading(false);
    }
  };
```

4. Handle Delete Website:

```
const handleDeleteWebsite = async () => {
    Swal.fire({
      icon: "warning",
      title: "Are you sure?",
      html: "<h6>This website will get permanently deleted</h6>",
      showCancelButton: true,
      confirmButtonText: `Delete`,
      confirmButtonColor: "#D14343",
    }).then(async (result) => {
      if (result.isConfirmed) {
        try {
          setIsLoading(true);
          await axios().delete(`/api/v1/websites/${data._id}`);
          Toast.fire({
            icon: "success",
            title: "Website deleted",
          });
          setTimeout(function () {
            fetchWebsiteData();
          }, 500);
          CloseButton.current.click();
          setIsLoading(false);
        } catch (error) {
          console.log(error);
          Toast.fire({
            icon: "error",
            title: error.response.data
              ? error.response.data.msg
              : error.message,
          });
          setIsLoading(false);
        }
      }
    });
  };
```

8. Login

```
const Login = () => {
  const navigate = useNavigate()

  const [email, setEmail] = useState('')
  const [password, setPassword] = useState('')
  const [showPassword, setShowPassword] = useState(false)

  const handleLogin = async (e) => {
    e.preventDefault()
    if (email === '')
      return Toast.fire({ icon: 'error', title: 'Email required' })
    if (password === '')
      return Toast.fire({ icon: 'error', title: 'Password required' })

    try {
      const res = await axios().post('/api/v1/auth/login', {
        email,
        password,
      })

      console.log(res.data)

      Toast.fire({
        icon: 'success',
        title: 'Logged In',
      })
      setTimeout(() => {
        localStorage.setItem('gimme_comment_access_token', res.data.token)
        window.location.reload()
      }, 1000)
    } catch (err) {
      console.log(err)
      if (err.response.data.msg === 'Email is not verified') {
        Toast.fire({
          icon: 'error',
          title: err.response.data ? err.response.data.msg : err.message,
        })
        navigate('/verify-account')
      } else {
        Toast.fire({
          icon: 'error',
          title: err.response.data ? err.response.data.msg : err.message,
        })
      }
    }
  }
```

9. SignUp.jsx

```
const SignUp = () => {
  const navigate = useNavigate()
  const [userDetail, setUserDetail] = useState(initialState)
  const [showPassword, setShowPassword] = useState(false)

  const handleChage = (e) => {
    const { name, value } = e.target

   setUserDetail({
      ...userDetail,
      [name]: value,
    })
  }
  const handleSubmit = async (e) => {
    e.preventDefault()

    try {
      const res = await axios().post('/api/v1/auth/register', userDetail)

      console.log(res)

     Toast.fire({
        icon: 'success',
        title: res.data.msg,
      })

    if (res.data) {
        navigate('/sign-in')
      }
    } catch (err) {
     console.log(err)
      Toast.fire({
        icon: 'error',
       title: err.response.data ? err.response.data.msg : err.message,
      })
    }
 }
```

10. Menu.js

```
return (
    <>
      <aside
        className="main-sidebar sidebar-dark-primary elevation-4"
        style={{ background: '#775DA8', position: 'fixed' }}
      >
        <div className="sidebar" style={{ height: '99vh' }}>
          <nav className="mt-2">
            <ul
              className="nav nav-pills nav-sidebar flex-column flex-nowrap"
              data-widget="treeview"
              role="menu"
              data-accordion="false"
            >
              <>
                <li className="nav-item">
                  <Link to="/websites" className={`nav-link `}>
                    <i className="nav-icon fa fa-map-o" />
                    <p>Websites</p>
                  </Link>
                </li>
              </>

              <li
                className="nav-header rounded-2 text-bold my-2 "
                style={{ background: '#6c4da9' }}
              >
                SETTINGS
              </li>
              <li className="nav-item">
                <Link to="/profile" className="nav-link">
                  <i className="nav-icon fas fa-user" />
                  <p>Profile</p>
                </Link>
              </li>
              <li className="nav-item">
                <a href="/#" className="nav-link" onClick={logOut}>
                  <i className="nav-icon fas fa-sign-out-alt" />
                  <p>Log Out</p>
                </a>
              </li>
            </ul>
          </nav>
        </div>
      </aside>
    </>
  )
}
```

CONCLUSION

In our contemporary digital era, effective and engaging communication is more critical than ever before. The Gimme Comments project undertaken was, at its heart, an attempt to leverage technology to foster more comprehensive, immediate, and engaging discussions in diverse online settings. This project began with a clear vision: to provide an intuitive, user-friendly platform where users could share their thoughts and interact seamlessly.

Over the duration of this project, numerous strides have been made. With its implementation, users are now capable of expressing their views, responding to others, and fostering engaging and meaningful dialogue. The service has empowered users to have their voices heard, effectively democratizing the online conversation landscape.

Moreover, the service has proved to be a vital tool for businesses, marketers, and content creators, providing them with real-time feedback and insights into their audience's thoughts and feelings. This has allowed them to tailor their strategies and content to better cater to their target demographic, bolstering engagement and ensuring a more personalized user experience.

Yet, this project was not without its challenges. Balancing user experience, privacy, and security concerns necessitated careful thought and consideration. This journey highlighted the importance of adaptability in the face of unpredictable challenges and the necessity of proactive problem-solving strategies.

In conclusion, the Commenting Service project signifies a key step towards better online communication and engagement. It underscores the integral role of such services in our increasingly interconnected digital world. We are immensely proud of the work accomplished and are excited to continue evolving the platform to meet the dynamic needs and expectations of the users. Above all, we remain steadfast in our commitment to facilitating meaningful and constructive discussions in the digital space.

FUTURE SCOPE

Looking ahead, there is ample room for further enhancements and refinements to the Gimme Comments Service. Possibilities include developing advanced moderation tools to ensure healthier conversations, incorporating AI to better understand sentiment and context, and integrating with more platforms to reach a wider user base. Some areas for potential future exploration include:

1. **Artificial Intelligence Integration:** Utilizing AI and Machine Learning algorithms could enable advanced features like sentiment analysis, spam detection, and automated content moderation. This would make the platform safer and more enjoyable for users.
2. **Multilingual Support:** As the user base grows and expands geographically, introducing multilingual support can help in catering to a wider, more diverse audience. This would involve implementing translation features or localizing the interface to different languages.
3. **Advanced Analytics:** More detailed analytical tools could be introduced, providing users, especially businesses and content creators, with a deeper understanding of audience engagement, sentiment, and behaviour patterns.
4. **Integration with More Platforms:** The Commenting Service could be made compatible with more digital platforms, extending its reach and applicability. For instance, integration with popular social media platforms, blogging sites, or e-commerce platforms could be explored.
5. **Voice Commenting Feature:** As voice technology becomes more prevalent, a voice commenting feature could be introduced, allowing users to leave voice comments in addition to text.
6. **Improved Accessibility:** Future work could involve making the service more accessible for users with disabilities. This could involve incorporating features like text-to-speech and speech-to-text, or improving visual accessibility for those with impaired vision.

REFERENCES

1. https://legacy.reactjs.org/docs/getting-started.html (React Documentation)
2. https://www.mongodb.com/docs/ (MongoDB Documentation)
3. https://expressjs.com/en/5x/api.html (ExpressJS Documentation)
4. https://nodejs.org/fa/docs (NodeJS Documentation)
5. https://vercel.com/docs (Vercel Documentation)
6. https://getbootstrap.com/docs/5.1/getting-started/introduction/ (Bootstrap V5)