FLEXI PROJECT REPORT

MERN CHATBOT

Submitted by

Aaryan Dhawan	Abhishek Rajput	Arnav Jain	Gautam Rajhans
22070122002	22070122007	22070122030	22070122068

Under the guidance of

Faculty Mentor

Mr. Ranjeet Bidwe

Assistant Professor

Visting Mentor

Mr. Moin Hasanfatta

Submitted on: Nov 10, 2024

Department of Computer Science and Information Technology

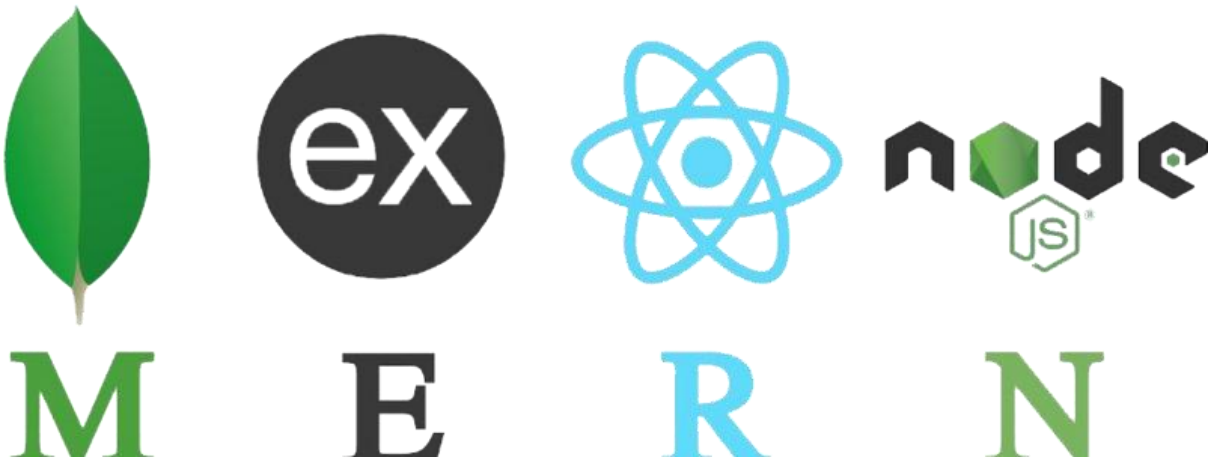
SYMBIOSIS INSTITUTE OF TECHNOLOGY, PUNE

TABLE OF CONTENTS

1. INTRODUCTION

The surge of Artificial Intelligence (AI) and Natural Language Processing (NLP) technologies has nowadays changed how customer service, communication, and user interaction has been improvised. Of all AI augmentations that left some of the the impact, one has been successfully creative – the chatbot that can be used for automatising mundane processes of responding to people's questions as well as making their communication fluid. This project investigates the creation of an interactive chatbot using the MERN (MongoDB, Express, React, Node.js) stack. With the implementation of AI features and the MERN stack, the present chatbot aims to promptly and correctly respond to users and practice natural conversation to increase user's interaction with the system.

The technology behind the MERN stack also makes integrating and designing this chatbot easier. For example, the use of MongoDB helps practice conversing data management, analytics and retrievals moreover, express and Node is useful for server-side assimilation and implementing dynamic query responses to interactions. Users may readily interact with the chatbot on a single page application; which is built using React and allows the users to talk to the chatbot. In this report, the design and implementation stage of the project will be discussed and each component of MERN stack to determine its role will be shown too. For this particular project, the intention is to develop a chatbot which is AI based and able to comprehend the user inputs and then formulate a response as well, demonstrating the usability of the MERS stack and how it can be used to build scalable conversational AI applications.



1. 1 OBJECTIVES

1. Put up a Chat Interface that Users will Easily Interact With: Use React to create a chat interface that the chatbot can connect to easily and users can easily communicate with the chatbot.
2. Provide server side Support for the Chatbot Interactions: Integrate Node.js and Express to setup server requirements and create the framework for the bot interaction to take place, this is to allow the UI to communicate with the server.
3. Employ Persistent Data storage in MongoDB Databases: MongoDB will be used to store the interaction logs, user input as well as the chatbot input to enable the information retrieval as well as storing and examining data.
4. Implement the AI/ NLP Techniques into the Chatbot: Include simple answering mechanisms whereby the basic AI can be used to receive key inputs or search for patterns so that the chatbot can give a proper response to the user.
5. Focus on Scalability and Performance: The design and the creation of the chatbot should allow it to be scaled up to be able to serve many users at the same time without jeopardizing performance.

1.2 PROBLEM STATEMENT

The modern-day business cannot afford to overlook customer service to enhance the firm's image. However, most businesses struggle with traditional support systems because these do not have the capacity to handle a large number of customer inquiries. Delays caused as a result of this leads to unsatisfied users and decreased efficiency. This is where chatbots come in as they take care of fairly simple queries or stick to basic repetitive routines. The major limitation, however, is that the market is saturated with chatbots that are able to work well in only one unchanging situation so creating one seems unrealistic. It is this problem that this project focuses upon; the development of a flexible and dynamic structure of a chatbot using the MERN stack.

Chatbots have been made possible by means of the MERN stack as useable features have been put in place allowing retrieval of conversation data, switching from one user to another simultaneously, as well as providing replies in real time. The chatbot has this integrated ability of basic AI that enables it interact with users and respond intelligently to their different inputs as well. The goal of this project is to develop a chatbot that would not only meet user needs of getting responses instantly and specifically to their questions without much complications but also compliment the traditional support methods which have their limitations. This type of chatbot can carry out multiple tasks. It is easy to set up, comfortable and it is intended to operate in high demand conditions and streamline the process of customer service.

2. TECHNICAL STACK

a. HTML

HTML (HyperText Markup Language) is a universal language that is used in the forming and organizing of web pages. HTML outlines a web page's structure, its components and includes such items as titles, texts, push buttons, forms, and hyperlinks. HTML forms the basis of web development, which is then enhanced with CSS and functional elements through JavaScript. Generally, HTML is employed for the front-end of the MERN chatbot project in structuring the client-side interface of the chatbot and allowing the chatbot users to communicate using a browser.

b. Tailwind CSS

Tailwind CSS, which stands for Tailwind Cascading Style Sheets, is a language employed to style and format the HTML elements that have been defined on the webpage. It specifies the style properties like color, font, space and position which helps UI developers to come up with attractive and useful interfaces. Tailwind CSS can be written on its own or combined with other frameworks such as Bootstrap to achieve a layout that can fit any form of device. As for this chatbot project, Tailwind CSS is applied to the UI of the chatbot, which optimizes the look of the chat box, buttons and the text to make the conversation simpler and increase the satisfaction rate of the end user.

c. JavaScript

JavaScript is an object-oriented, concise and integrated programming language designed specifically for web development. The main achievement of developers when creating JavaScript programming language was to make it easy to integrate into browsers. Thanks to its design as a scripting language, JavaScript also enables the creation of interactive content on websites. As the world and users' needs evolved, JavaScript's capabilities expanded, allowing for the development of SPA (Single Page Applications) with JavaScript frameworks in the front-end such as Vue.js, Angular, and React .

d. NodeJS

Node.js can be defined as a server-side JavaScript environment that allows the use of JavaScript for the development of backend services. Node.js is built on the V8 JavaScript engine developed by Google, and its popular features are: Asynchronous and event-driven, this allows concurrent requests to be processed efficiently. In the MERN stack, Node.js acts as the bedrock of backend architecture where developers are able to create APIs, manage server's functionalities, and link the application to a document-oriented database such as MongoDB. It has become one of the best tools for building large applications and integrates various packages that enable fast building and deploying of sophisticated functionality.

e. MongoDB

It is a NoSQL database that uses BSON (Binary JSON) documents as storage instead of tables. Unlike other databases, MongoDB does not use a fixed pattern, but rather utilizes a form, allowing for storages that are more versatile and document based. The application is easily expandable and is fast, meaning it can deal with large size data sets, Thus this application is ideal for use where large amounts of data are to be quickly accessed and delivered. For MERN full stack applications, MongoDB acts as the common IoT database for user, chat and other IDs as structured and unstructured data. In addition, it provides active querying and horizontal scaling, which allows it to work efficiently even in real-time applications such as chatbots.

f. ExpressJS

Express is an open-source and lightweight web framework for Node.js. It accelerates the creation of APIs. washers, and web apps by offering routing and middles. Within the MERN stack, Express operates as the backend, processes requests, issues routes, and specifies where specific REST APIs lie. A straightforward approach to designing applications with Express allows managing requests from the front-end and responses from the back-end in an orderly fashion. It is suitable for creating profile-oriented web apps due to its ease of use and adaptability.

g. React

React is a powerful JavaScript library that provides a way to create rich, interactive user interfaces for applications, including single page applications. React follows a component-based architecture which makes it easy for developers to develop self-sufficient reusable UI logic and components and create extensive and manageable applications. This also makes it faster and more responsive by employing a virtual DOM that renders changing areas only, improving application speed.

h. Vite

Vite is a modern tool that moderates the React development process and makes it more efficient by providing a quick and reliable build environment. For example, earlier React projects were introduced as Single Page Application that were developed using a bundler called Webpack that would increase in speed over time making the entire development process sluggish. What Evan You developed in Vite is able to solve this problem by avoiding the need to load a web application with ES modules employing more components than necessary.

i. Nodemailer

It is a Node.js module used for sending emails from within an application. It supports various email services and allows developers to configure and send emails programmatically. In the chatbot project, Nodemailer can be used to send notifications, confirmations, or password reset emails to users, enhancing the chatbot's functionality by providing additional communication methods. For instance, if the chatbot includes account-related services, Nodemailer can be used to notify users of successful interactions or to send automated follow-up emails.

j. Axios

Axios is a Javascript library which is used to perform an HTTP request from the client to the server and third-party services. It streamlines the tasks involved in sending, handling, or dealing with asynchronous requests with its responses. Because of this reason, For instance, in this case, it applies to fetching data from any Third-party API using React. The same applies when post-informs the server with some queries. About the MERN chatbot project, Axios may help transmit queries from users to the backend send responses back to users or even use other APIs to make the chatbot richer and enable synchronization of information between the front and back ends of the application.

k. GeminiAI

Gemini AI is Google DeepMind's next-generation AI model that combines advanced generative and reasoning capabilities. Part of Google's evolving AI ecosystem, Gemini represents a step forward in AI technology by integrating powerful language understanding, multi-modal capabilities, and improved contextual reasoning. It can process not only text but also images and other data formats, making it useful for a range of applications including content generation, complex data analysis, and interactive assistance.

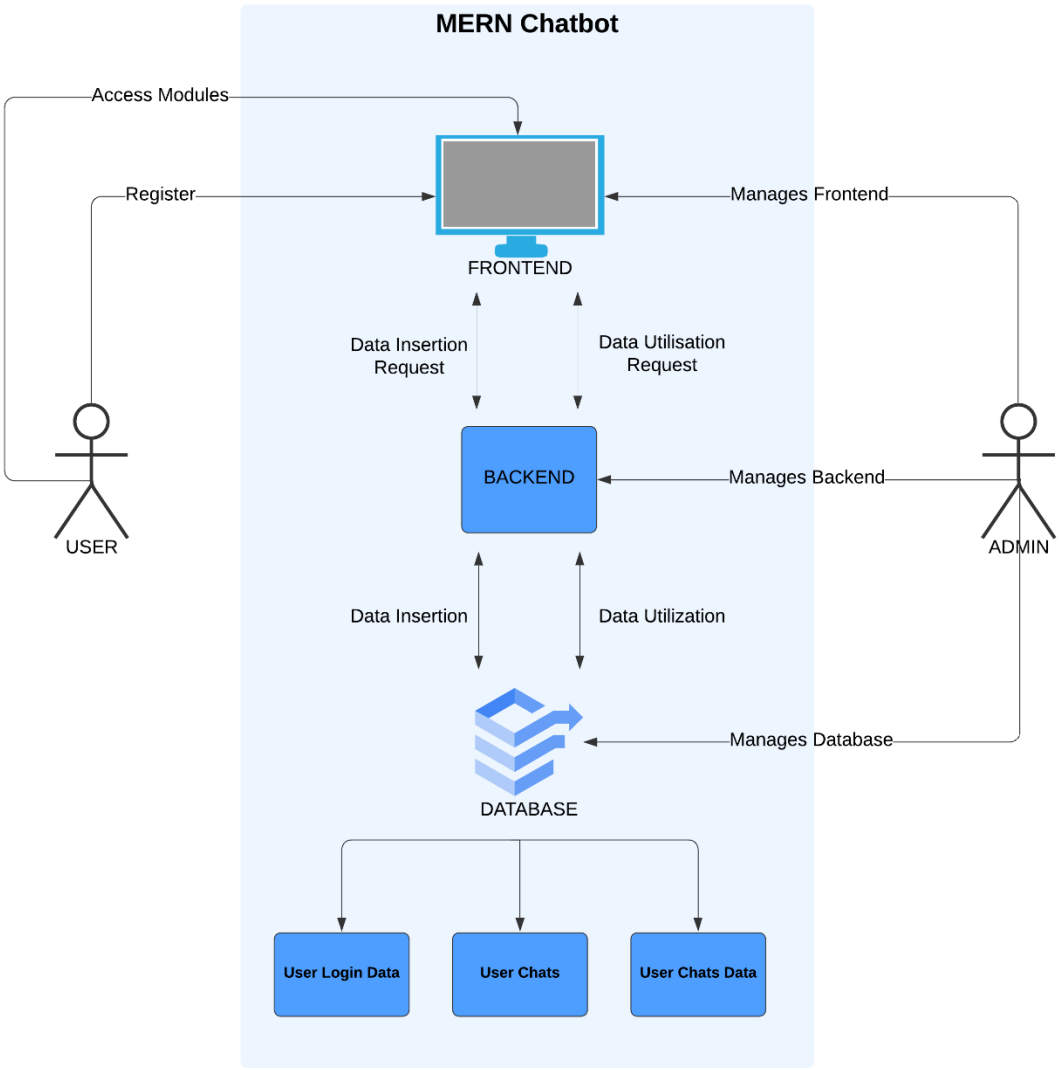
l. Postman

It is a popular tool for API development and testing, widely used by developers to simplify interactions with APIs. It provides a user-friendly interface for creating, testing, and documenting RESTful APIs without needing to write additional code or scripts. With Postman, developers can send HTTP requests (GET, POST, PUT, DELETE, etc.) to API endpoints, examine responses, and validate the performance and functionality of an API.

m. Figma

Figma is a cloud application for designing which focuses on the design of user interfaces, wireframes, and interactive design. Real-time collaboration is featured; therefore, users may work on the same project simultaneously, and they need not be in the same location, as just an internet browser will do. The tool is very intuitive and comprehensive, with features such as version history, along with integration and plugins found here. The teamwork nature of Figma highly improves the speed of project delivery and design for software development.

3. SYSTEM ARCHITECTURE



4. CODE EXPLANATION

4.1 FRONTEND

1. login.jsx

- **Purpose**: This file handles the login functionality, where a user enters their email to log in.
- **Key Components**:
  - useState hook to manage the email state.
  - UserData context to access the loginUser function and btnLoading state.
  - A form that takes an email input and calls the loginUser function when submitted.
  - The submit button shows a loading spinner while the btnLoading state is true.
- **Explanation**:
  - When the form is submitted, loginUser is called with the entered email and navigate is used to redirect the user to the verification page (/verify).
  - The button shows a loading spinner if the btnLoading state is true.

2. home.jsx

- **Purpose**: This component serves as the main dashboard or home page, displaying the main chat interface once the user has logged in and verified their account.
- **Key Components**:
  - Header component to display a greeting or prompt the user to create a new chat.
  - Sidebar component to navigate between chats and allow the user to manage them.
  - ChatData context to manage chats, including fetching and deleting chats.
  - UserData context to handle user data and log out functionality.
  - A list of chats, where each chat can be clicked to load and display its messages.
- **Explanation**:
  - The page fetches the list of chats and displays them.
  - A new chat can be created from the sidebar, and each chat has an option to be selected or deleted.
  - It also allows for logout by triggering the logoutHandler from UserData.

3. verify.jsx

- **Purpose**: This component is for OTP verification after the user logs in.
- **Key Components**:
  - useState hook to manage the OTP (otp) state.
  - UserData context for the verifyUser function and btnLoading state.
  - ChatData context to call fetchChats after successful verification.
  - A form for the user to enter the OTP and submit it for verification.
- **Explanation**:
  - When the form is submitted, the verifyUser function is called with the OTP, and if successful, the user is redirected to the main page (/), and chats are fetched.
  - The button shows a loading spinner while btnLoading is true.

4. chatcontext.jsx

- **Purpose**: This file manages all the chat-related logic, including fetching and creating chats, deleting chats, and fetching messages.
- **Key Components**:
  - createContext and useContext for managing and consuming context.
  - fetchChats, createChat, fetchMessages, deleteChat, and fetchResponse functions to interact with the backend.
  - messages, prompt, newRequestLoading, and chats state to track chat data and the current user's prompt.
- **Explanation**:
  - fetchChats: Fetches all chat data from the server and sets it to chats state.
  - createChat: Creates a new chat and refreshes the list of chats.
  - fetchMessages: Fetches messages for the currently selected chat.
  - deleteChat: Deletes a selected chat and refreshes the chat list.
  - fetchResponse: Sends a prompt to the backend and updates the chat with the response.
  - The ChatContext.Provider wraps the children components and provides all the chat data and functions.

5. usercontext.jsx

- **Purpose**: This file handles user authentication, including login, verification, and fetching user data.
- **Key Components**:
  - createContext and useContext for managing and consuming user context.
  - loginUser and verifyUser functions to handle user authentication.
  - fetchUser to fetch the user data from the server.
  - logoutHandler to log out the user.

- **Explanation**:
  - loginUser: Sends the email to the server and handles the login process, storing the verifyToken in localStorage.
  - verifyUser: Verifies the OTP provided by the user and sets the token and user in localStorage.
  - fetchUser: Fetches user data from the server and updates the isAuth state to track the authentication status.
  - logoutHandler: Clears the localStorage and logs out the user.
  - The UserContext.Provider wraps the children components and provides authentication data and functions.

6. header.jsx

- **Purpose**: This is a simple header component that displays a greeting message.
- **Key Components**:
  - The ChatData context is used to access the chats data.
- **Explanation**:
  - Displays a greeting message or a prompt to create a new chat if there are no existing chats.

7. loading.jsx

- **Purpose**: Contains various loading spinner components used to indicate the loading state.
- **Key Components**:
  - LoadingSpinner: A small spinning circle for loading.
  - LoadingBig: A bigger loading spinner with bouncing circles.
  - LoadingSmall: A smaller version of LoadingBig for less intrusive loading indicators.
- **Explanation**:
  - These components are used throughout the app whenever a loading state is needed to show the user that something is in progress.

8. sidebar.jsx

- **Purpose**: This component handles the sidebar which contains the list of existing chats and options to create or delete chats.
- **Key Components**:
  - ChatData context for interacting with chats (createChat, deleteChat, setSelected, etc.).
  - UserData context to access the logoutHandler.
- **Explanation**:
  - deleteChatHandler: Confirms and deletes the chat.
  - clickEvent: Sets the selected chat and closes the sidebar.
  - The sidebar is toggleable and displays a list of recent chats, with options to create new chats and log out.

9. index.html

- **Purpose**: This is the main HTML template for your app.
- **Explanation**:
  - Contains the id="root" where your React app will be rendered.
  - The script src="/src/main.jsx" is the entry point for the React app that Vite compiles and bundles.

10. app.jsx

- **Purpose**: This file is the main entry point for the application and defines the routing structure using react-router-dom. It determines which component should be displayed based on the authentication state (isAuth) of the user.
- **Key Components**:
  - **BrowserRouter**: This component wraps the entire application and enables the use of routing in React.
  - **Routes and Route**: These components define the routes for the application. Each route corresponds to a URL path and specifies which component should be rendered for that path.
  - **UserData context**: The UserData context provides the user's authentication data (isAuth, user, and loading). It is used to check if the user is logged in and whether the app is still loading.
  - **LoadingBig**: This component is displayed when the loading state is true, indicating that some data is still being fetched or processed (e.g., when checking authentication status).

- **Explanation**:
    - The App component conditionally renders the LoadingBig spinner if the loading state from UserData is true, indicating that authentication or some initial data is still loading.
    - If the app is not in a loading state, the BrowserRouter is used to manage routing. The Routes component contains multiple Route components:
        - The root path (/) and /login route both render the Home component if the user is authenticated (isAuth is true). Otherwise, they render the Login component.
        - The /verify route renders the Home component if the user is authenticated, otherwise it renders the Verify component for OTP verification.

11. main.jsx

- **Purpose**: This file serves as the entry point to initialize and render the entire React application. It includes the necessary context providers and renders the App component into the root DOM element.
- **Key Components**:
    - **React.StrictMode**: This wrapper helps highlight potential problems in the application during development. It doesn't affect the production build but provides additional warnings and checks during development.
    - **UserProvider and ChatProvider**: These context providers wrap the App component and provide global state for user data and chat data. The UserProvider manages authentication and user data, while the ChatProvider handles chat-related data and actions.
    - **server constant**: This stores the base URL of the backend server (http://localhost:5000), which can be used throughout the application for making API requests.
- **Explanation**:
    - The ReactDOM.createRoot function is used to render the React application into the DOM element with the id="root". This is the entry point for rendering all the components.
    - The UserProvider and ChatProvider components are wrapped around the App component to provide the necessary global states (user authentication and chat data) to all child components. These providers make their respective data available throughout the application by utilizing React's Context API.
    - The server constant can be used wherever API calls are made to connect to the backend server, ensuring that the server's URL is centralized and easily modifiable.

General Flow (Frontend)

1. Initial Setup (main.jsx):

- **React and Context Providers**:
  - The app starts by rendering main.jsx which initializes the React application by wrapping the root component (App) with necessary context providers.
  - UserProvider manages the user's authentication data (like login state and user info).
  - ChatProvider manages chat-related data (like messages, chat creation, and deletion).

2. Routing Setup (app.jsx):

- **Authentication Check**:
  - App.jsx handles the routing for the application using react-router-dom. It first checks if the user is authenticated (isAuth state) and whether the app is in a loading state (loading state).
  - **Loading State**:
    - If loading is true (i.e., authentication or data is still being processed), the LoadingBig spinner is displayed to the user.
  - **Authenticated Routes**:
    - If the user is authenticated (isAuth is true), they are redirected to the main Home component.
  - **Unauthenticated Routes**:
    - If the user is not authenticated, they will be redirected to either the Login or Verify pages, depending on the state of their authentication.

3. Authentication Flow:

a. Login (login.jsx):

- **Login Form**:
  - The Login component allows the user to input their email.
  - Upon submitting the email, the loginUser function (from UserData context) is called to authenticate the user.
  - If authentication is successful, the user is redirected to the /verify route for OTP verification.
  - A loading spinner is shown while the btnLoading state is true (indicating that authentication is in progress).

b. OTP Verification (verify.jsx):

- **OTP Form**:
  - The user enters the OTP sent to their email in the Verify component.
  - Upon submitting the OTP, the verifyUser function (from UserData context) is called to verify the OTP.
  - If OTP is verified successfully, the user is redirected to the Home component, and the fetchChats function (from ChatData context) is called to fetch all previous chats.

4. Home Page (Home.jsx):

- **Welcome and Chat Overview**:
  - After successful login/OTP verification, the user is shown the Home component. It contains a greeting message and the list of available chats.
  - If no chats exist, the user is prompted to create a new chat.

5. Sidebar (sidebar.jsx):

- **Chat Management**:
  - The sidebar contains a list of recent chats and allows the user to:
    - **Create a New Chat**: Clicking on the "New Chat +" button triggers the createChat function (from ChatData context) to create a new chat.
    - **Delete Chats**: Clicking the delete button next to a chat triggers the deleteChat function to delete the selected chat.
    - **Select a Chat**: Clicking on a chat in the list sets it as the selected chat (via setSelected function from ChatData context).

6. Chat Interaction:

- **Chat Interface**:
  - Once a chat is selected, the user can send messages or prompts.
  - The message or prompt is sent to the backend via fetchResponse (from ChatData context), and the response from the backend is displayed in the chat.
  - The chat interface updates in real-time as new messages are sent and received.

7. Loading States (loading.jsx):

- **Loading Indicators**:
  - Throughout the app, loading indicators (like LoadingBig, LoadingSmall, and LoadingSpinner) are shown whenever an action is in progress (e.g., when loading user data, chats, or when creating a new chat).
  - These loading spinners are used in various components like Sidebar, Home, and Login to provide feedback to the user.

8. User Logout:

- **Logout Functionality**:
  - The logoutHandler function (from UserData context) is triggered when the user clicks the "Logout" button in the sidebar.
  - This clears the authentication token from local storage and logs the user out, redirecting them to the login page.

4.2 BACKEND

1. index.js — Server Setup

- **Purpose**: This is the entry point for your server, where the Express application is initialized and configured.
- **Key Features**:
  - Loads environment variables using dotenv.
  - Initializes the Express app and sets up middleware (express.json() for parsing JSON requests and cors() for enabling Cross-Origin Resource Sharing).
  - Imports and uses the routes from userRoutes and chatRoutes.
  - Listens on the port defined in the .env file and establishes a connection to MongoDB via connectDb().

2. chatController.js — Handles Chat-Related Operations

This file contains the core logic for handling chat-related requests, such as creating new chats, fetching conversations, and adding new messages to chats.

- **createChat**:
  - Creates a new chat document for the logged-in user.
  - The user._id is fetched from the authenticated user (using the isAuth middleware).
- **getAllChats**:
  - Fetches all chats for the authenticated user, sorted by creation time.
- **addConversation**:
  - Adds a conversation (question and answer) to a specific chat.
  - Updates the latest message in the chat after a new conversation is added.
- **getConversation**:
  - Retrieves all conversations for a particular chat by chatId.
- **deleteChat**:
  - This function is in charge of deleting a chat provided the authenticated uses owns such a chat.

3. userController.js — User Login and Profile Management

This file contains informational relations that expose user login, one-time password verification, and profile management functions.

- **loginUser**:
  - This method is used where the email of a user is issued and is for the purpose of determining whether such a user has an account.
  - Where such a user exists it issues a JWT token for verification purposes.

- Otherwise, the system generates an OTP and dispatches it to the user's registered email account for authentication purposes.
- **verifyUser**:
  - Verifies the OTP sent to the user's email and authenticates the user.
  - Upon successful verification, the user is saved to the database, and a JWT token is generated for future authentication.
- **myProfile**:
  - Handles the profile details of the logged-in user and returns them back.

4. isAuth.js — Middleware for User Authentication

- **Purpose**: This middleware restricts access of certain routes to authenticated users only.
- **Key Features**:
  - Looks for a JWT Token in the request header.
  - Fetches user's data from the database and attaches it to req. user object if the token is valid.
  - Responds with a 403 error status if the token is not available or is invalid.

5. sendMail.js — Handles OTP Sending via Email

- **Purpose**: This module is responsible for sending OTP to the mails of the registered users.
- **Key Features**:
  - Allows sending of an email to the user's email with the OTP attached to it.
  - The email is sent in HTML format and is also styled with inline CSS in order to improve its presentation.

6. chat.js — Chat Model

- **Purpose**: This file creates the MongoDB schema and model for chat collection.
- **Key Features**:
  - Each document of chat contains user referencing document and latest messages with its user.
  - Timestamps are enabled so the document created or modified times are recorded.

7. conversation.js — Conversation Model

- **Purpose**: Defines the schema for conversations held inside the user-chatbot interface within a single chat.
- **Key Features**:
  - Every conversation is attached to the corresponding Chat and includes fields for a question and answer.
  - timestamps are present for noting the time when the conversations are created or modified.

8. user.js — User Model

- **Purpose**: This file provides the MongoDB schema for users in the User collection.
- **Key Features**:
  - Every single document belonging to a user has an email field that is unique to that user.
  - User creation and updation date would always be maintained with the help of timestamps.

9. chatRoutes.js — Handles Routes for Chat Operations

- **Purpose**: This file specifies the routes for the given chat operations with the help of the functions from the chatController already defined in the application.
- **Key Routes**:
  - POST /new: A new chat is created, only after somebody logs in.
  - GET /all: New user can get all existing chats for the logged in user.
  - POST /:id: A new conversation can be added in an existing chat.
  - GET /:id: One can get all the conversations that belong to a particular chat.
  - DELETE /:id: Any one chat can be deleted out of the many chats available.

10. userRoutes.js — Handles Routes for User Operations

- **Purpose**: This file Specify all the routes of user login and that of user profile.
- **Key Routes**:
  - POST /login: Perform user login and OTP will be delivered as an sms.
  - POST /verify: User is verified on the basis of the OTP sent and is thereby signed in.
  - GET /me: Allows any user who is logged in to seek details for their profile.

General Flow (Backend)

1. **User Authentication**:
    - In the event of a user login attempt, the backend tries to pull the user from the database or sends an OTP for verification to the user in case a user does not exist.
    - After validating the OTP, new user creation takes place and an issued JWT further identifies user.
2. **Chat Management**:
    - Upon successfully logging in, the user is able to initiate the creation of a new chat, linked to their user ID.
    - Users can ask the chatbot questions which will be recorded as conversations in a chat format which contains the users and their questions.
    - Most recent message is maintained by every chat document while every conversation maintains the question together with the answers that correspond with it.
3. **Database Models**:
    - Represents the documents structure of your database in the form of a User, Chat and Conversations model.
    - The relationships between users, their chats and conversations are created in MongoDB using ObjectId references.
4. **Authentication Middleware**:
    - The isAuth middleware is used to protect routes, ensuring that only authenticated users can access certain operations (like fetching chats or adding conversations).
5. **Email Verification**:
    - An OTP is sent via email to verify the user's identity during the login process, using the sendMail service.

4.3 OVERALL FLOW:

1. User visits the site.
2. Authentication: The user is asked to log in if he/she is not logged in.
3. Login Registration: The user logs in by pressing the login button and entering their email.
4. Security: The user is sent one time password, which the user must enter to secure the login process.
5. Home page: Verified users can now see the home page where they will be able to handle the chat management.
6. Sidebar Menu: Users can manage their chats by creating, deleting, or selecting one from a sidebar.
7. Chatting: After a chat is opened, users send prompts/messages and get the responses from the backend.
8. Logout option: The system user has the ability to log out the user which means the full authentication information is deleted and routed to the login page.

5. CHALLENGES AND SOLUTIONS

1. Natural Language Processing (NLP) Complexity

- **Challenge**: The challenge for any chatbot designer is to adapt the NLP technology so that a chatbot would be able to understand and interpret various users' input data and retrieve the most relevant data amongst many other possible responses that it could generate.
- **Solution**: To some extent, this complexity was alleviated by using the capabilities of Gemini AI. By utilizing pretrained models, the chatbot does not require heavy training; it needs to grasp context and recognize keywords and intents to produce relevant responses.

2. Data Management and Scalability

- **Challenge**: As the number of users increased, scaling was needed to ensure that the system could accommodate user needs while maintaining acceptable performance levels. Dealing with chat logs and the data generated from interactions for analysis presented difficulties.
- **Solution**: MongoDB offered a data storage structure that was both scalable and flexible, making it an ideal solution for data management and storage. Due to its document-oriented model, it was able to store both structured and unstructured data, which is ideal for chat applications with numerous messages. In addition, MongoDB's horizontal scaling and optimized indexing made it possible to manage big data relatively quickly.

3. Real-Time Communication

- **Challenge**: The ability to respond simultaneously to several users is critical for the effective use of an interactive chatbot, which means that a single page react app must do everything appropriately to minimize latency.
- **Solution**: Through Nodejs as well as Express the chatbot was able to communicate with user in real time. Bidirectional data transfer was possible and enabled real time interaction without repetitive API calls.

4. User Interface Optimization

- **Challenge**: It was very important to develop a simple and fast chatbot interface so that users do not get lost when communicating with the bot and that the interface is efficient regardless of the device or screen size.
- **Solution**: Using react in this scenario helped with creating a decentralized UI design since it has a component structure. CSS as well as Tailwind framework was useful in making the layout feel responsive. There was also the added advantage of using Vite as the build tool which is also fairly efficient resulting in better performance when changing the UI.

5. Error Handling and User Experience

- **Challenge**: In any way possible avoiding the wrongful contact with the users service is one of the more important things in any application. Errors are mainly associated with network problems or simply typing in any of the fields incorrectly.
- **Solution**: With the assistance of Axios, it was possible to perform the HTTP request whereby if an error is returned, then appropriate messages will be presented to the user stating the problems encountered. Validation checks on any users input were enforced lowering the chances of miscommunication and making any feedback from the chatbot more concise.

6. CONCLUSION

The MERN-based chatbot project demonstrates the effective integration of MERN stack with AI technologies to create a conversational interface that is attractive and can be expanded. Thanks to the four components: MongoDB, Express, React and Node.js, it is possible to meet the requirements of the chatbot in terms of real time communication and volume of data. Due to the document model of MongoDB, it is easier to store and retrieve chat loger and other profile information of the users. Express and Node.js take care of backend tasks and serve as fast and reliable responders. React creates a single page application that ensures the user can deploy the chatbot from any computer device. All these elements create a healthy environment that presents the user with engaging experiences that are quick and interactive.

The chatbot's functions include advanced English-speech understanding, and context relevance of AI driven NLP's pre-trained models for effective and adaptive bot-user interaction. The Gemini AI chatbot can recognize the user's intention by retrieving essential phrases and provides appropriate answers with ease thanks to the smart and contextually relevant language that transforms alt text images. With this functionality, the virtual assistant improves significantly and can make decisions or provide information that makes the conversation appear more natural rather than operating with the defined set of rules that may not be effective enough.A consummate reduction in the necessity of continuous retraining while achieving natural language understanding, endorses AI potential in address customer queries in the real time which are commonplace.

This project fills the gaps left by the classical models of customer service – that are only effective with a set number of standard questions. Customers do not have to wait for extended periods of time for resolution of simple questions because the chatbot can address such problems at once. It also makes it clear how AI oriented chatbots can replace the previous model operating customer support. Built on robust technological base, this chatbot seems to be the perfect AI example which has the power to entirely change the face of customer service for the better and more efficient.

7. APPENDIX 1

7.1 ADDITIONAL RESOURCE

Deployement Platform: Render (Backend), Vercel (Frontend)

Deployment Link: https://mern-chatbot-kappa.vercel.app/

Github Link: https://github.com/Abhishek-2502/MERN_Chatbot

API Website Link: https://aistudio.google.com/

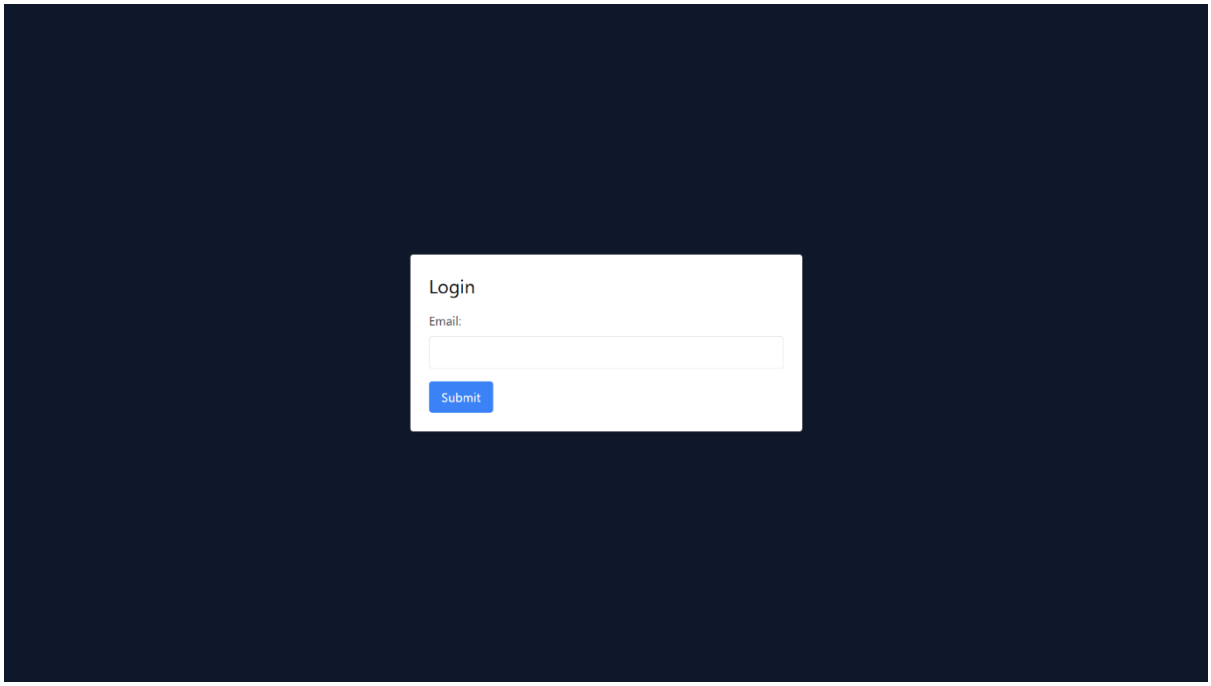
Figma Link: https://www.figma.com/design/2geACCVwnOUAMsbwaLmhzl/Mern-Chatbot?node-id=0-1&t=12LVok9ZDTDWZxH7-1

7.2 REFERENCES

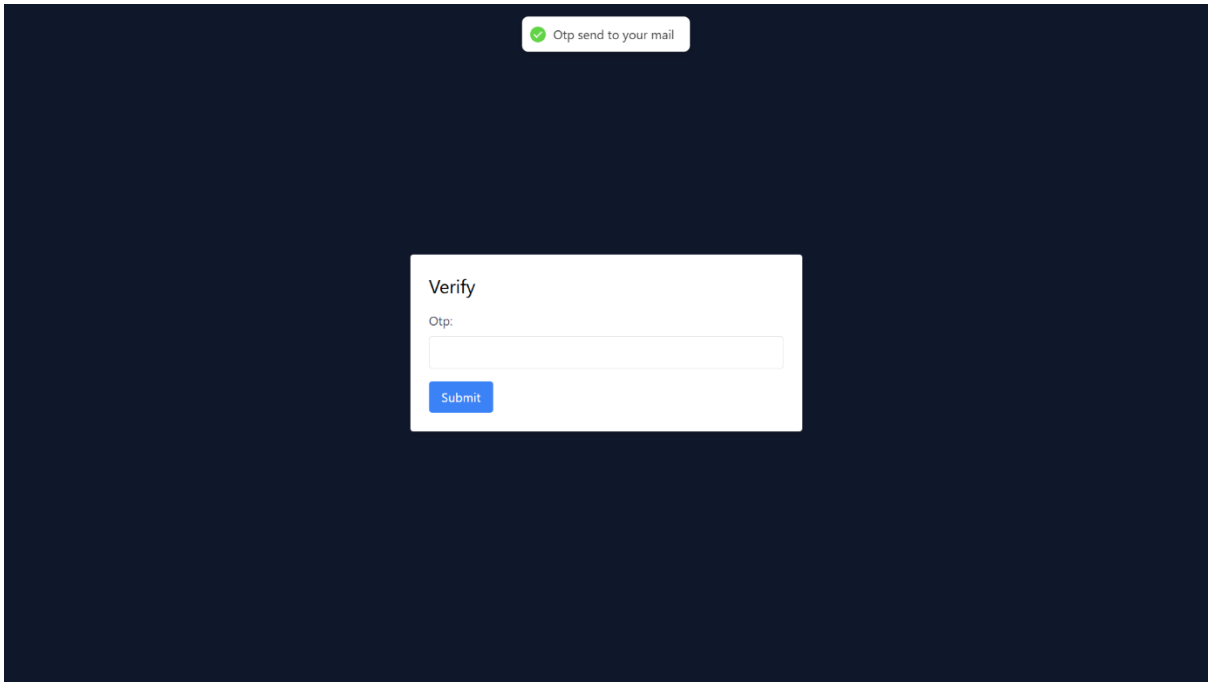
- https://www.geeksforgeeks.org/mern-stack/
- https://www.freecodecamp.org/news/build-an-ai-chatbot-with-the-mern-stack/
- https://medium.com/@sanikakotgire2003/building-your-own-ai-chat-bot-using-mern-mongodb-express-react-and-node-js-stack-306a37a4845d
- https://youtu.be/YZCQafLGuz8?si=r8JQ2pSO-XB2Wfey
- https://youtu.be/iqHmP5ydRZI?si=mXNkZ91fkgX7yzZu
- https://www.oracle.com/in/database/mern-stack/

7.3 SCREENSHOTS

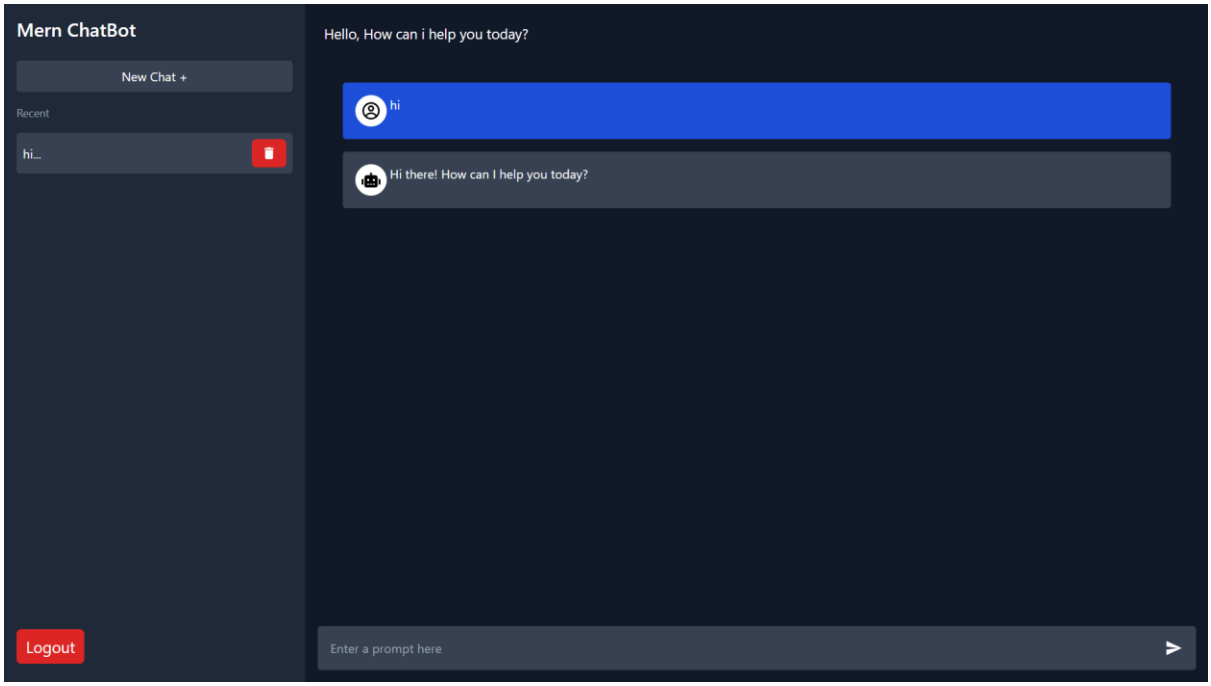
Website on Desktop



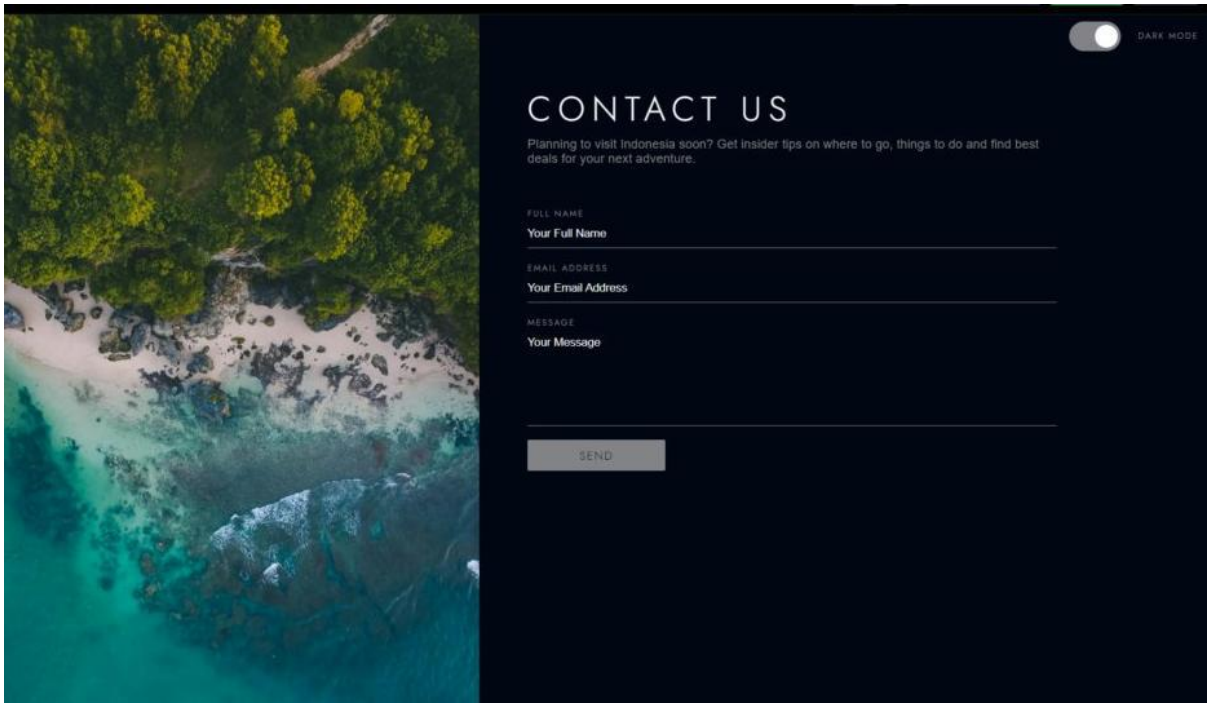
Login



OTP Verification

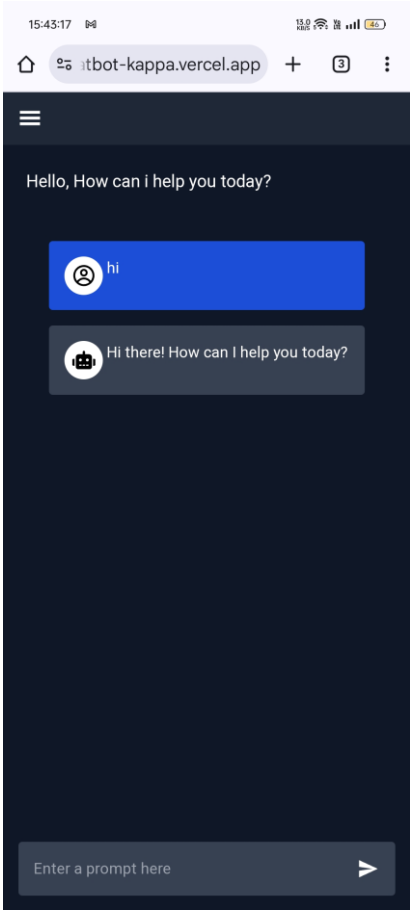


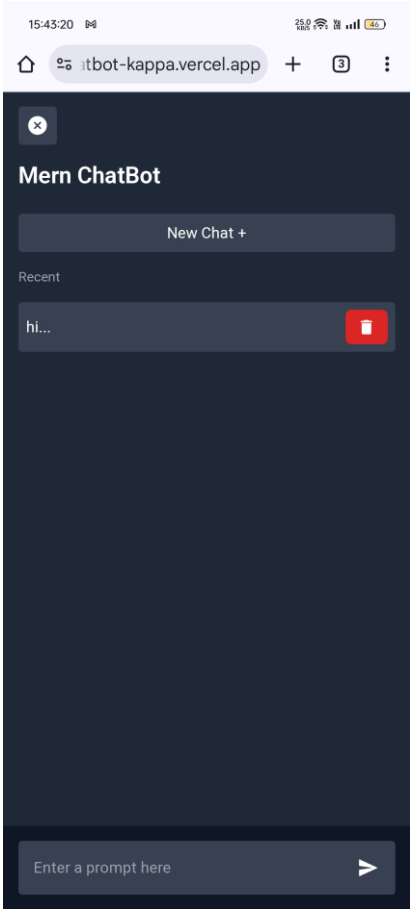
Home Page



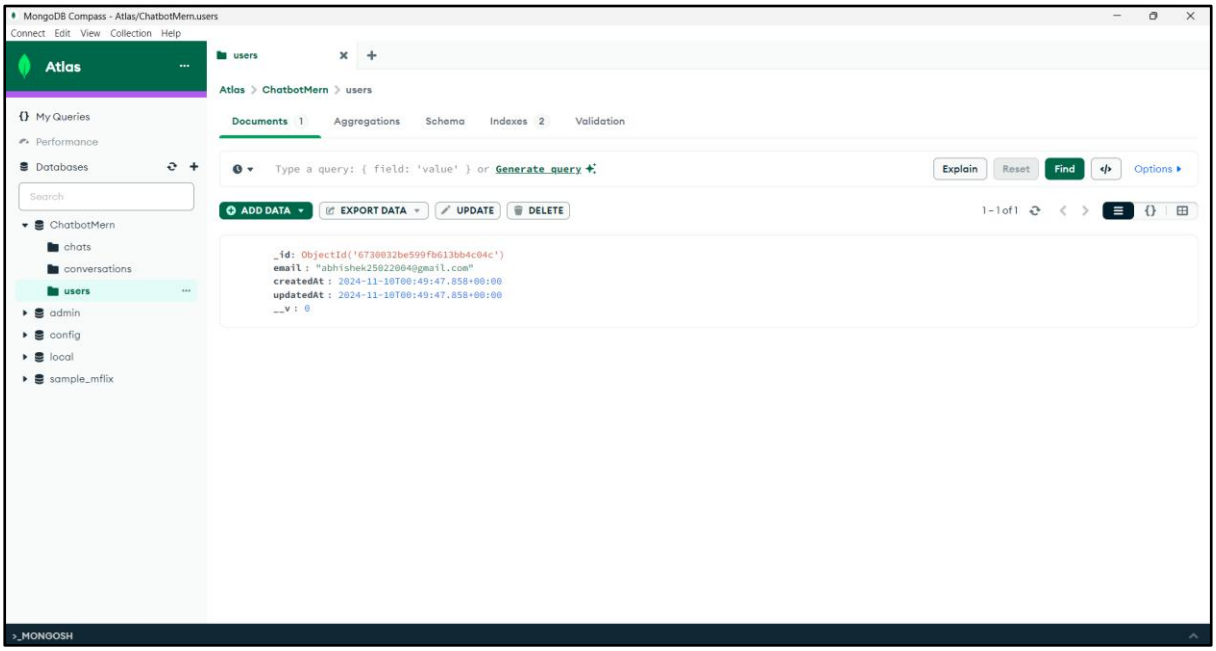
Contact Us Page

Website on Mobile

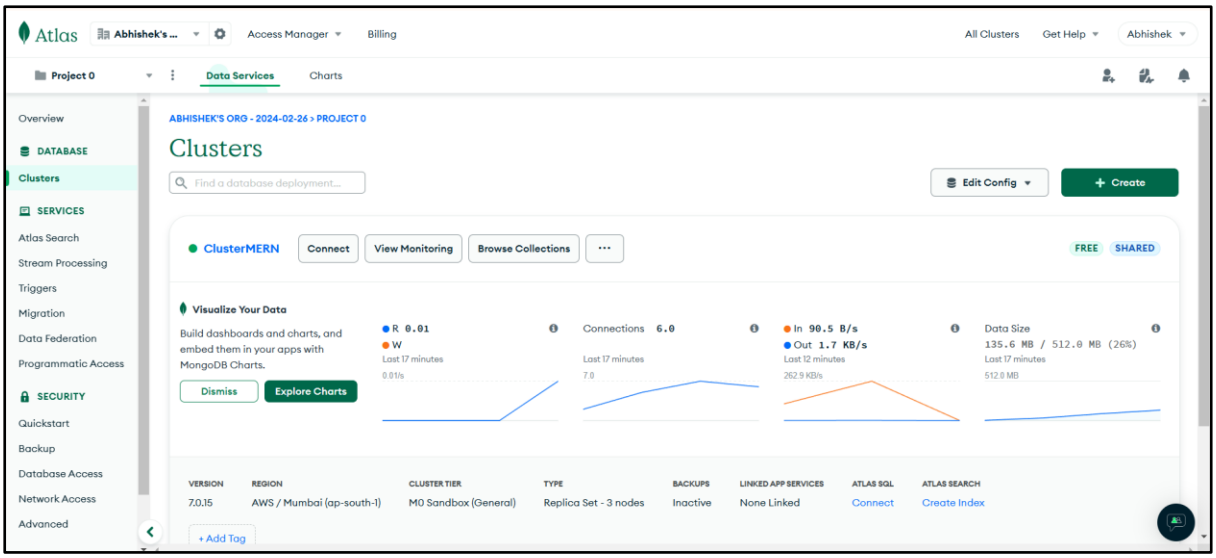




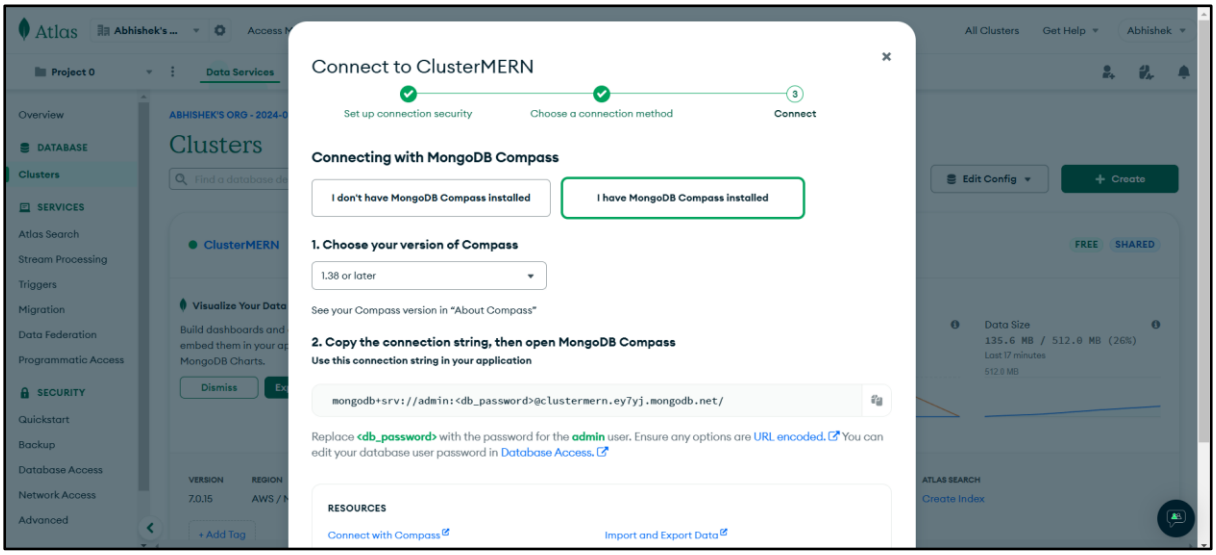
MongoDB



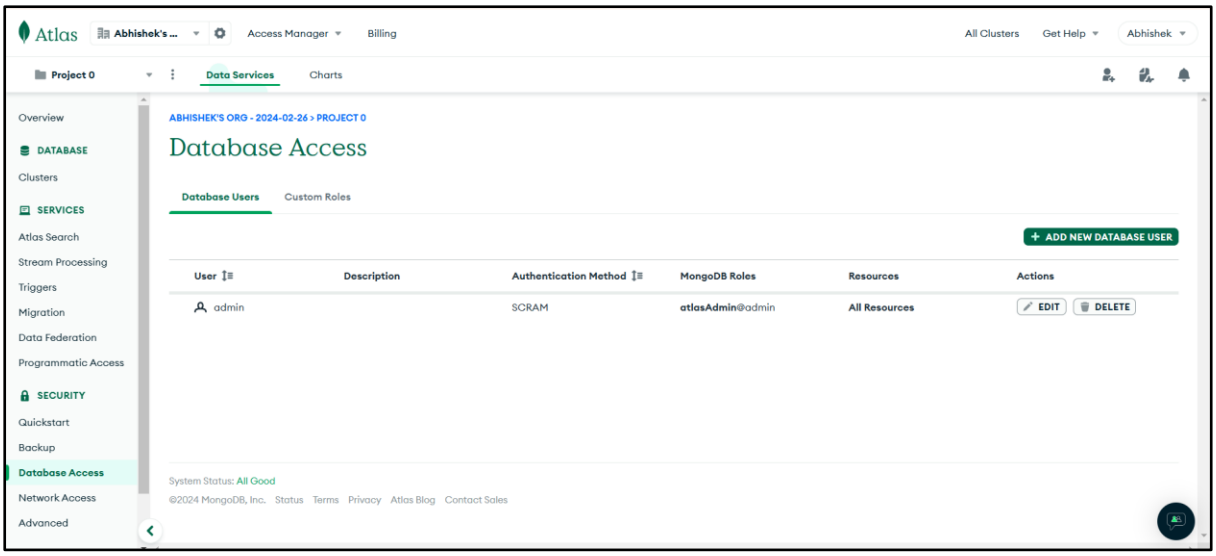
MongoDB



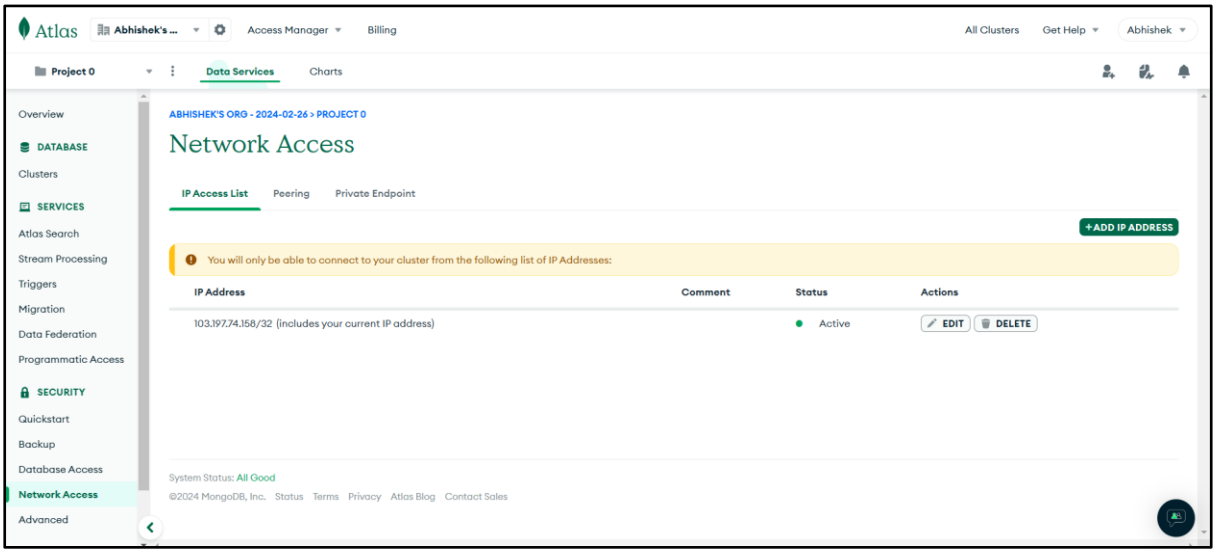
MongoDB Atlas Cluster



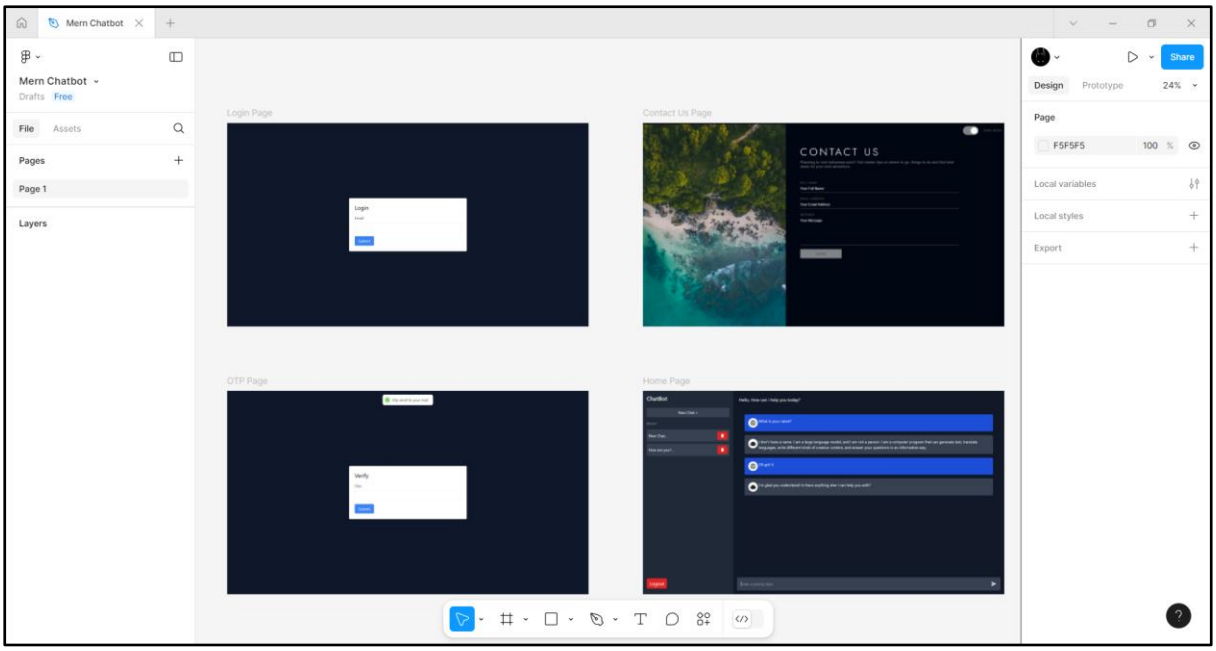
MongoDB Atlas URL



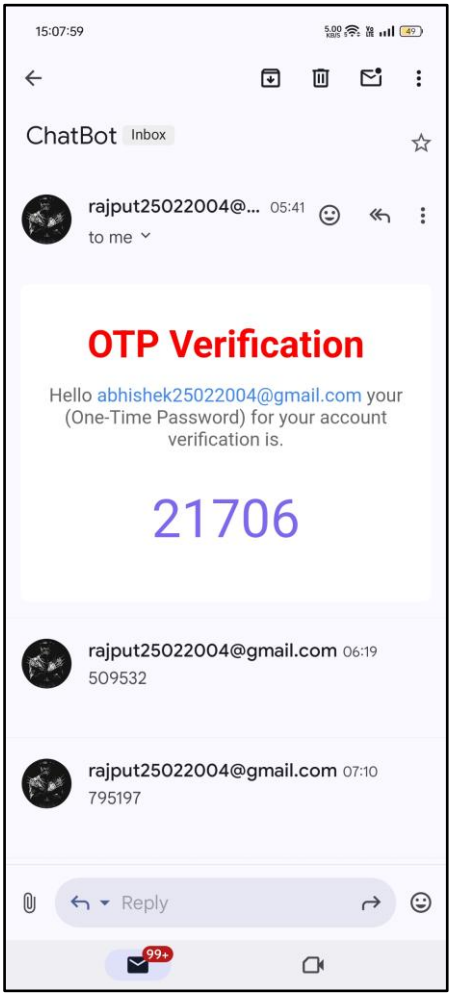
MongoDB Atlas Database Access



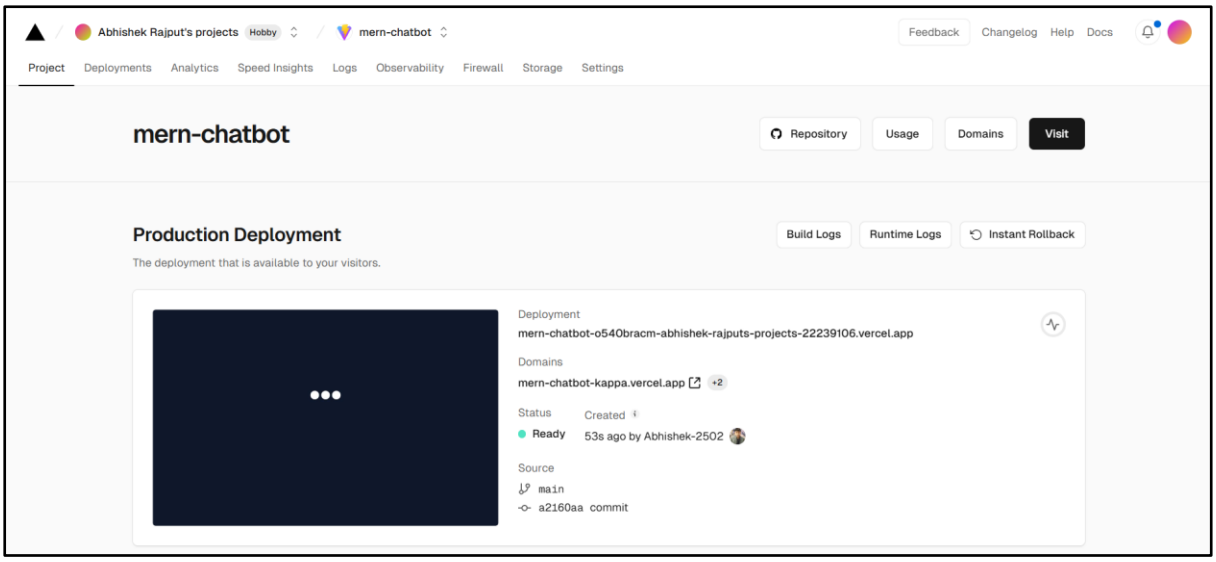
MongoDB Atlas Network Access



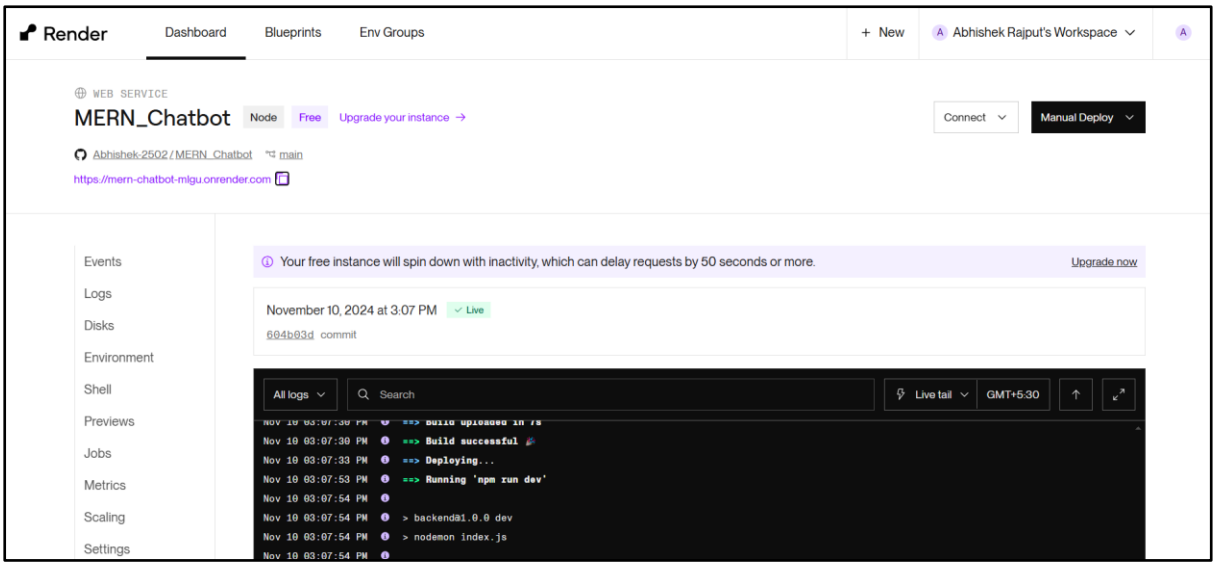
Figma



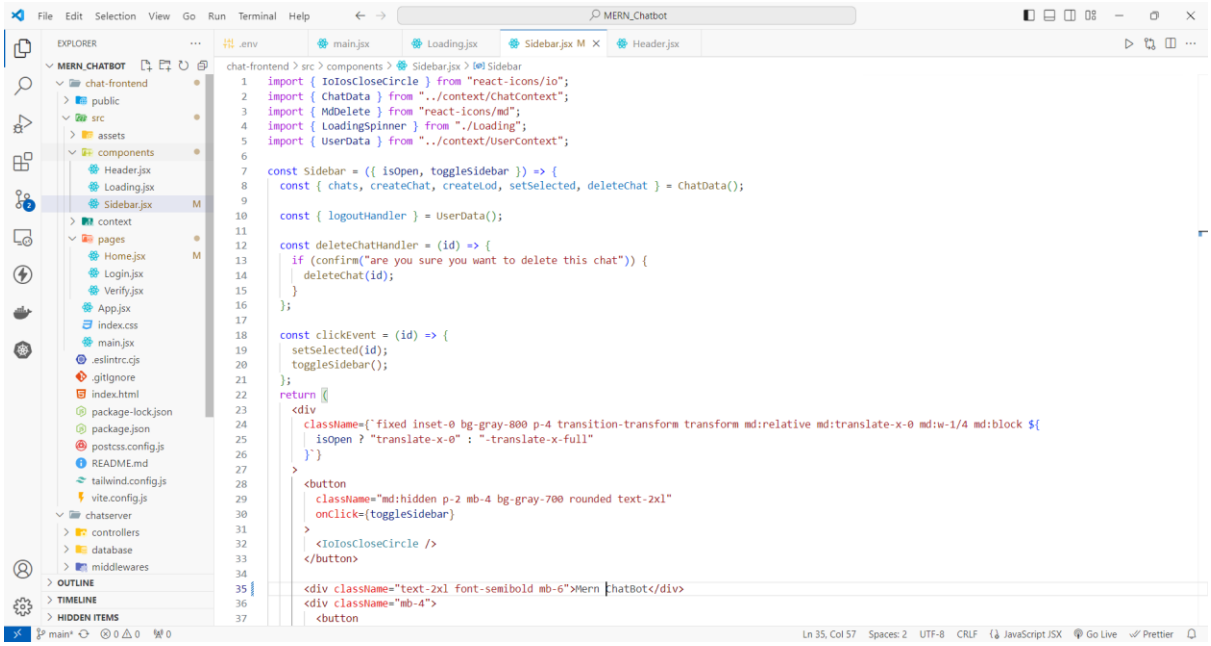
Nodemailer OTP



Vercel Frontend Deployment

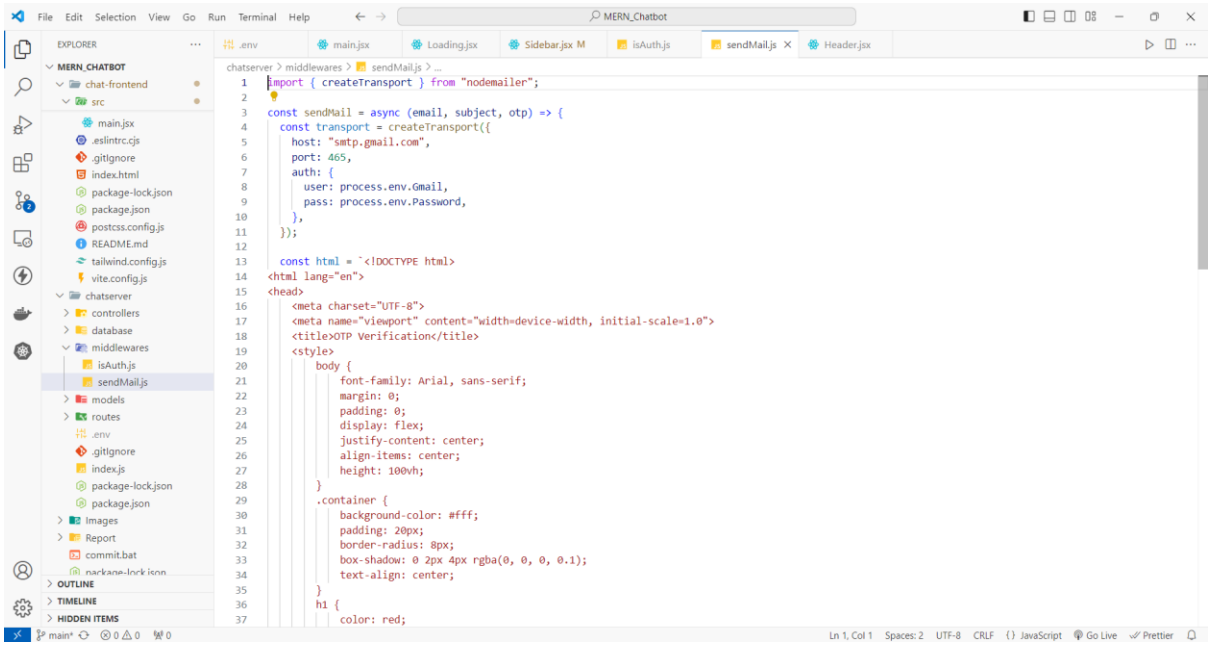


Render Backend Deployment



```jsx
import { IoIosCloseCircle } from "react-icons/io";
import { ChatData } from "../context/ChatContext";
import { MdDelete } from "react-icons/md";
import { LoadingSpinner } from "./Loading";
import { UserData } from "../context/UserContext";

const Sidebar = ({ isOpen, toggleSidebar }) => {
  const { chats, createChat, createLod, setSelected, deleteChat } = ChatData();

  const { logoutHandler } = UserData();

  const deleteChatHandler = (id) => {
    if (confirm("are you sure you want to delete this chat")) {
      deleteChat(id);
    }
  };

  const clickEvent = (id) => {
    setSelected(id);
    toggleSidebar();
  };
  return (
    <div
      className={`fixed inset-0 bg-gray-800 p-4 transition-transform transform md:relative md:translate-x-0 md:w-1/4 md:block ${
        isOpen ? "translate-x-0" : "-translate-x-full"
      }`}
    >
      <button
        className="md:hidden p-2 mb-4 bg-gray-700 rounded text-2xl"
        onClick={toggleSidebar}
      >
        <IoIosCloseCircle />
      </button>

      <div className="text-2xl font-semibold mb-6">Mern ChatBot</div>
      <div className="mb-4">
        <button
```

Code Snippet Frontend



```node
import { createTransport } from "nodemailer";

const sendMail = async (email, subject, otp) => {
  const transport = createTransport({
    host: "smtp.gmail.com",
    port: 465,
    auth: {
      user: process.env.Gmail,
      pass: process.env.Password,
    },
  });

  const html = `<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>OTP Verification</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
        }
        .container {
            background-color: #fff;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
            text-align: center;
        }
        h1 {
            color: red;
```

Code Snippet Backend

8. APPENDIX 2

8.1 PERSONAL PORTFOLIO AND TECHNICAL DETAILS

PRN: 22070122002

Name: Aaryan Dhawan

Portfolio Link: https://dhawanaaaryan.github.io/portfolio/

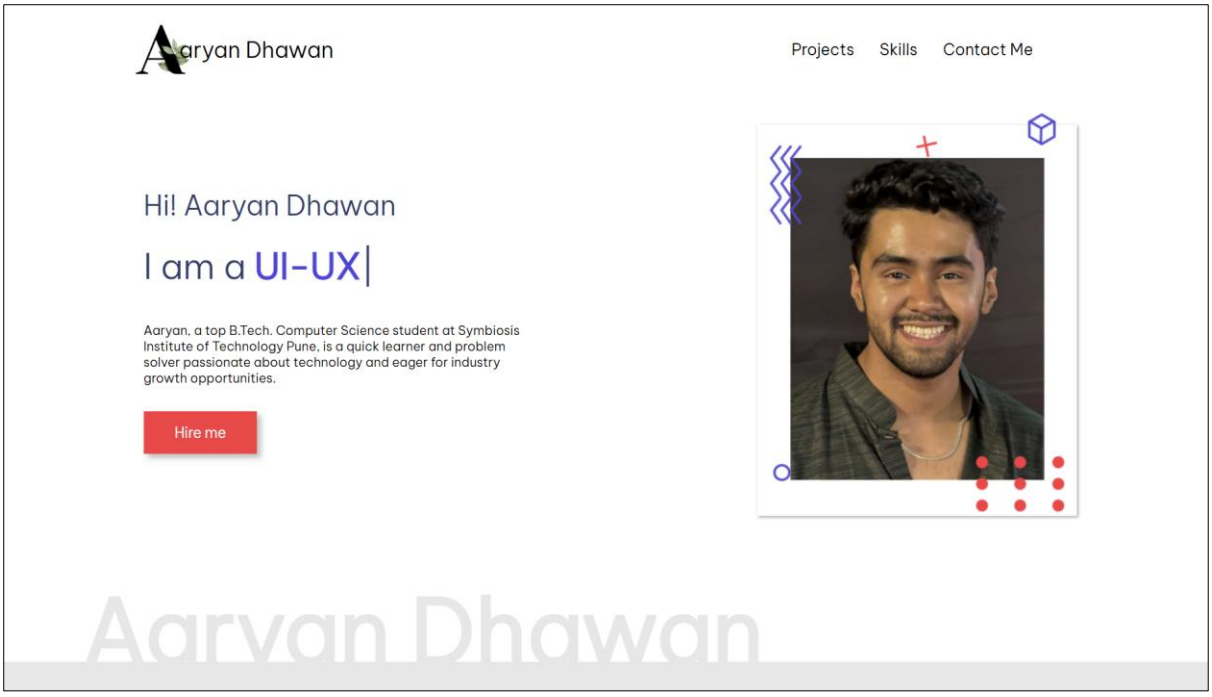
Repository Link: https://github.com/dhawanaaaryan/portfolio

Technical Details:

The portfolio is designed with a clean and modern aesthetic, emphasizes simplicity and ease of navigation. It is built using HTML, CSS, and JavaScript, incorporating responsive design principles to ensure optimal viewing across devices. The minimalistic layout highlights my projects and skills effectively, with visually appealing sections and smooth transitions enhancing the user experience.

Images:

On Desktop




Me and
MyTech Stack
Hi everyone! I'm Aaryan, a passionate B.Tech. student in Computer Science Engineering at Symbiosis Institute of Technology, Pune. I thrive on exploring technology, and I have hands-on experience in development and data analysis. When I'm not coding, you can find me playing guitar, trading in Forex, or experimenting in the kitchen!
I'm skilled in Python, C++, Java, HTML, and CSS, with experience in full-stack development, Azure, and database management using MySQL and MongoDB. I'm eager to leverage my tech stack to tackle real-world challenges!

On Mobile


aryan Dhawan
Hi! Aaryan Dhawan
I am a Coder
Aaryan, a top B.Tech. Computer Science stude
Technology Pune, is a quick learner and proble
technology and eager for industry growth opp
Hire me
Projects


Projects
Campus Buddy System
Line Follower Robot

PRN: 22070122007

Name: Abhishek Rajput

Portfolio Link: https://abhishek-2502.github.io/Portfolio/

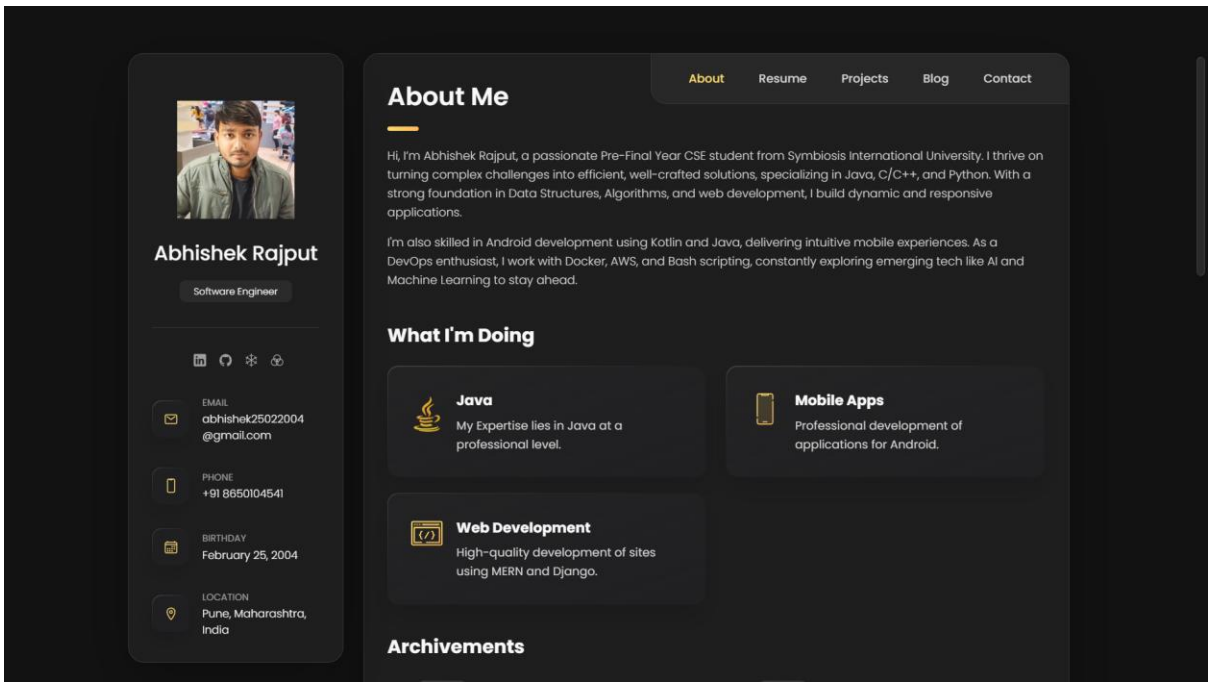
Repository Link: https://github.com/Abhishek-2502/Portfolio

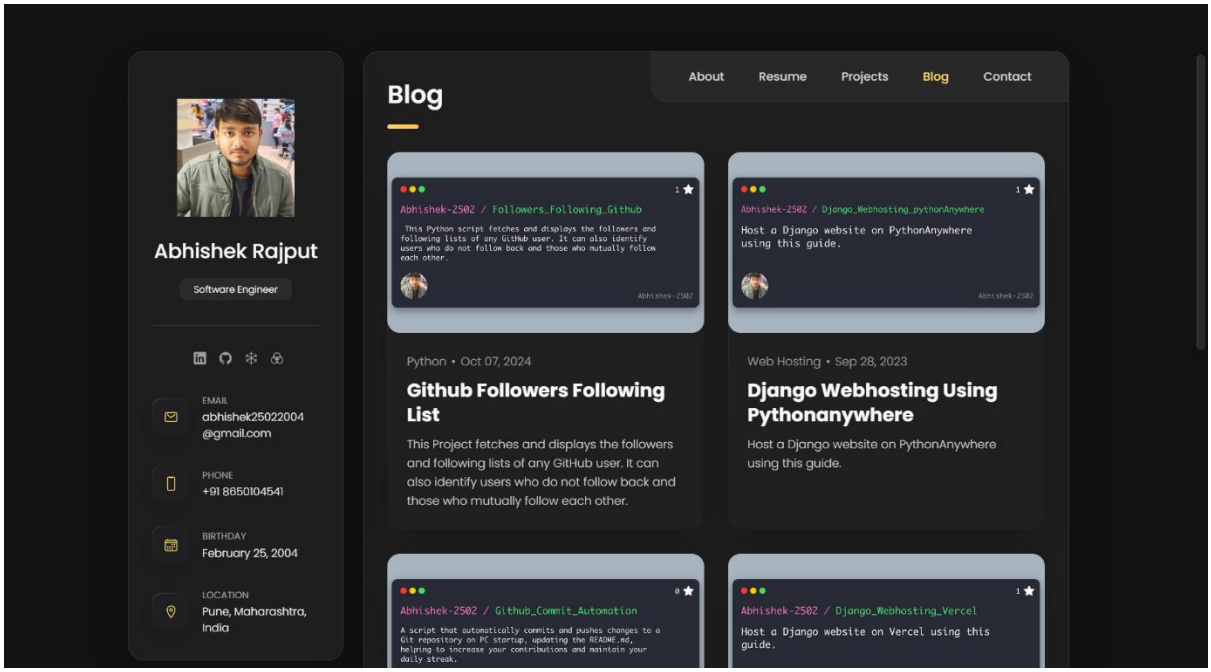
Technical Details:

The portfolio website consists of several pages and is able to adapt to screen resolutions due to the fact that it was built using HTML, CSS, and JavaScript code. All the elements of the website correspond to the modern and simple style, due to which it is both easy to use and attractive. The website encompasses dedicated sections for the resume, blog, projects and contact information, which helps to gain a holistic insight of the professional details. There are a wide range of devices and designs for which the responsive design principles ensure best viewing experiences, and there is a modular structure which makes updates easy. The layout showcases a portfolio of the noticeably illustrated projects along with the technical skills by means of light animations and transitions resulting in a simple and wonderful work flow.

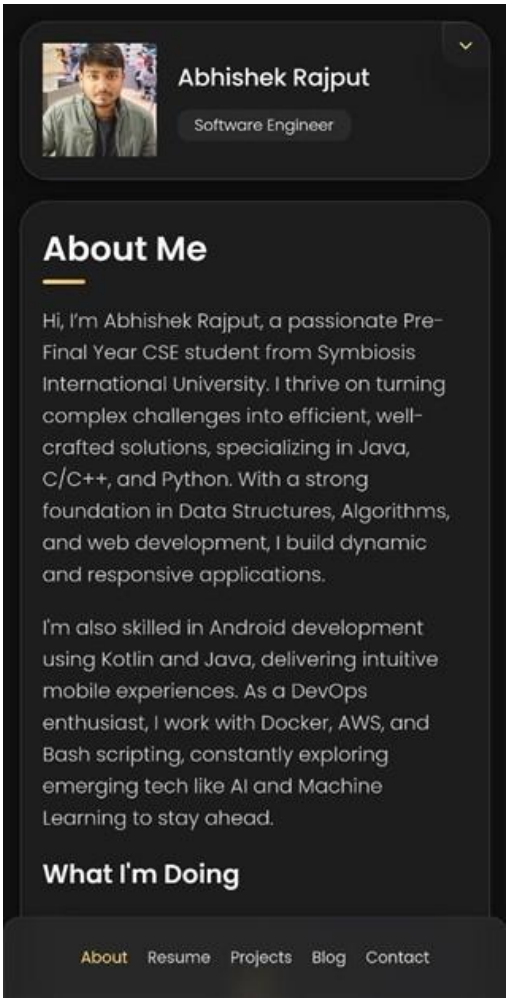
Images:

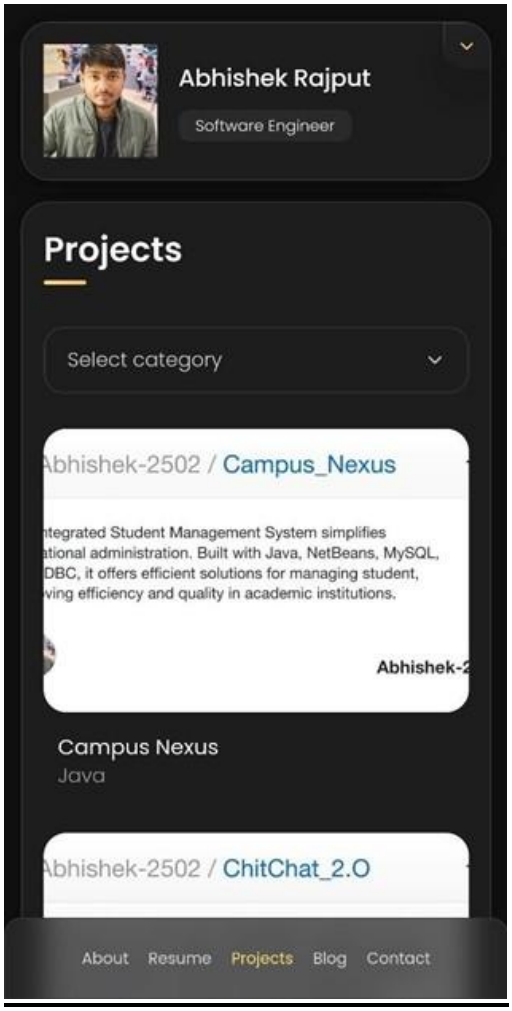
On Desktop




Abhishek Rajput
Software Engineer
EMAIL
abhishek25022004@gmail.com
PHONE
+91 8650104541
BIRTHDAY
February 25, 2004
LOCATION
Pune, Maharashtra, India
Blog
About
Resume
Projects
Blog
Contact
Abhishek-2502 / Followers_Following_Github
This Python script fetches and displays the followers and following lists of any GitHub user. It can also identify users who do not follow back and those who mutually follow each other.
Python • Oct 07, 2024
Github Followers Following List
This Project fetches and displays the followers and following lists of any GitHub user. It can also identify users who do not follow back and those who mutually follow each other.
Abhishek-2502 / Django_Webhosting_pythonAnywhere
Host a Django website on PythonAnywhere using this guide.
Web Hosting • Sep 28, 2023
Django Webhosting Using Pythonanywhere
Host a Django website on PythonAnywhere using this guide.
Abhishek-2502 / Github_Commit_Automation
Abhishek-2502 / Django_Webhosting_Vercel
Host a Django website on Vercel using this guide.

On Mobile


Abhishek Rajput
Software Engineer
About Me
Hi, I'm Abhishek Rajput, a passionate Pre-Final Year CSE student from Symbiosis International University. I thrive on turning complex challenges into efficient, well-crafted solutions, specializing in Java, C/C++, and Python. With a strong foundation in Data Structures, Algorithms, and web development, I build dynamic and responsive applications.
I'm also skilled in Android development using Kotlin and Java, delivering intuitive mobile experiences. As a DevOps enthusiast, I work with Docker, AWS, and Bash scripting, constantly exploring emerging tech like AI and Machine Learning to stay ahead.
What I'm Doing
About
Resume
Projects
Blog
Contact


Abhishek Rajput
Software Engineer
Projects
Select category
Abhishek-2502 / Campus_Nexus
Campus Nexus
Java
Abhishek-2502 / ChitChat_2.0
About
Resume
Projects
Blog
Contact

PRN: 22070122030

Name: Arnav Jain

Portfolio Link: http://arnav-jain.vercel.app

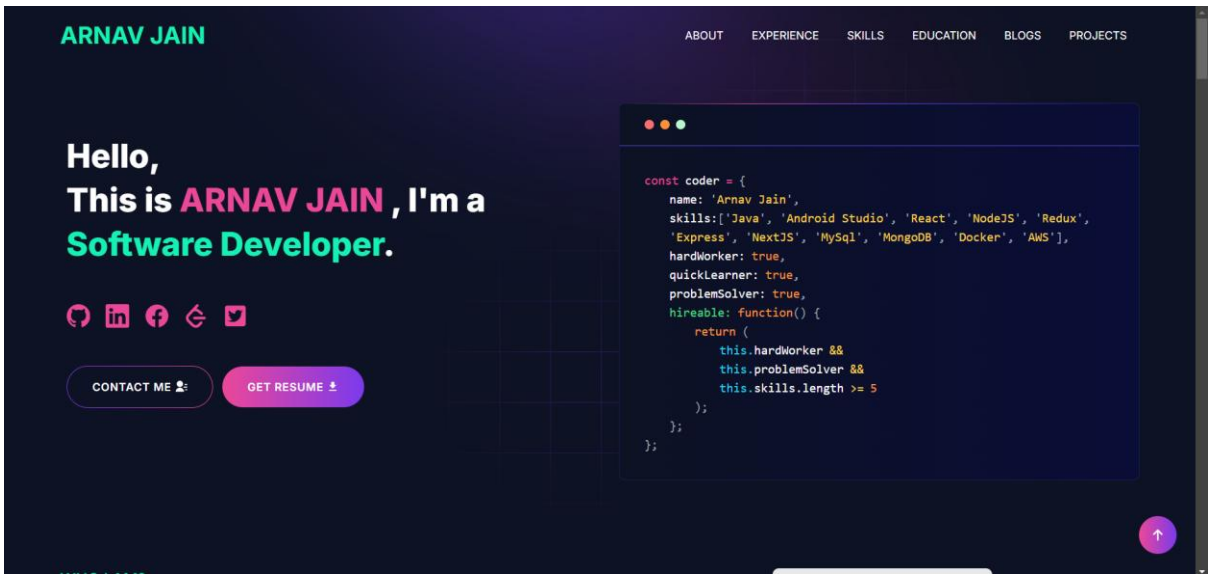
Repository Link: https://github.com/Arnavjain2503/My-Portfolio

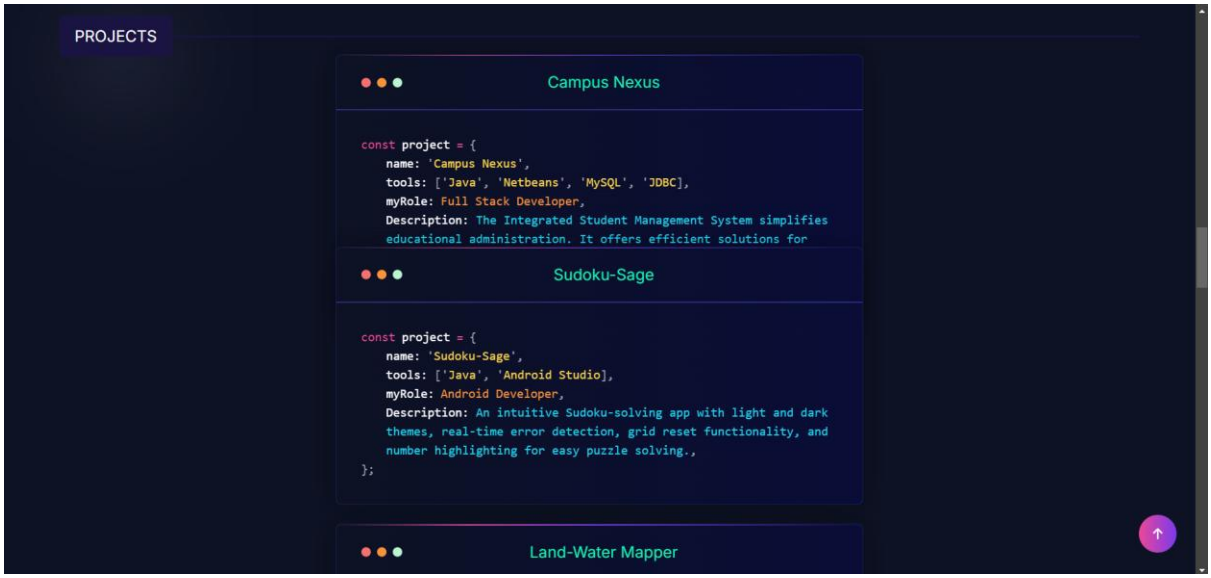
Technical Details:

The main intention of my portfolio is to visualize some works and demonstrate professionalism in an easy-to-use structure. Created with a developer-oriented approach, different parts are dedicated to skill, experience, project, education, and more which gives an overall view about my profile. Including many other useful features, the portfolio is created using HTML, CSS and JS and designed with Next.js, Tailwind CSS and React.

Images:

On Desktop




PROJECTS
Campus Nexus
const project = {
name: 'Campus Nexus',
tools: ['Java', 'Netbeans', 'MySQL', 'JDBC],
myRole: Full Stack Developer,
Description: The Integrated Student Management System simplifies
educational administration. It offers efficient solutions for
Sudoku-Sage
const project = {
name: 'Sudoku-Sage',
tools: ['Java', 'Android Studio],
myRole: Android Developer,
Description: An intuitive Sudoku-solving app with light and dark
themes, real-time error detection, grid reset functionality, and
number highlighting for easy puzzle solving.,
};
Land-Water Mapper

On Mobile


ARNAV JAIN
const coder = {
name: 'Arnav Jain',
skills:['Java', 'Android Studio', 'React',
'NodeJS', 'Redux', 'Express', 'NextJS',
'MySql', 'MongoDB', 'Docker', 'AWS'],
hardWorker: true,
quickLearner: true,
problemSolver: true,
hireable: function() {
return (
this.hardWorker &&
this.problemSolver &&
this.skills.length >= 5
);
};
};
Hello,
This is ARNAV JAIN , I'm
a Software Developer.
CONTACT ME
GET RESUME


Experiences
(Sep 2024 - Present)
WEB DEVELOPMENT INTERN
Project Human City (PHC), Toronto,
Canada (Remote)
(July 2024 - Present)
FULL STACK DEVELOPER INTERN
Sevadeep Foundation, Pune,
Maharashtra

PRN: 22070122068

Name: Gautam Rajhans

Portfolio Link: https://capricode-ui.github.io/

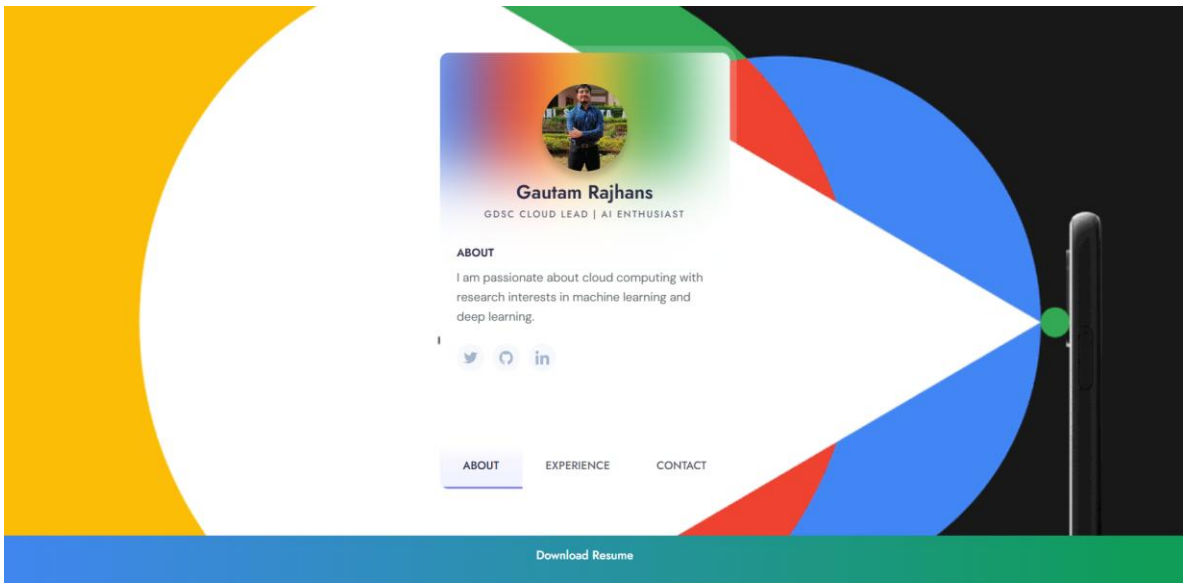
Repository Link: capricode-ui/capricode-ui.github.io

Technical Details:

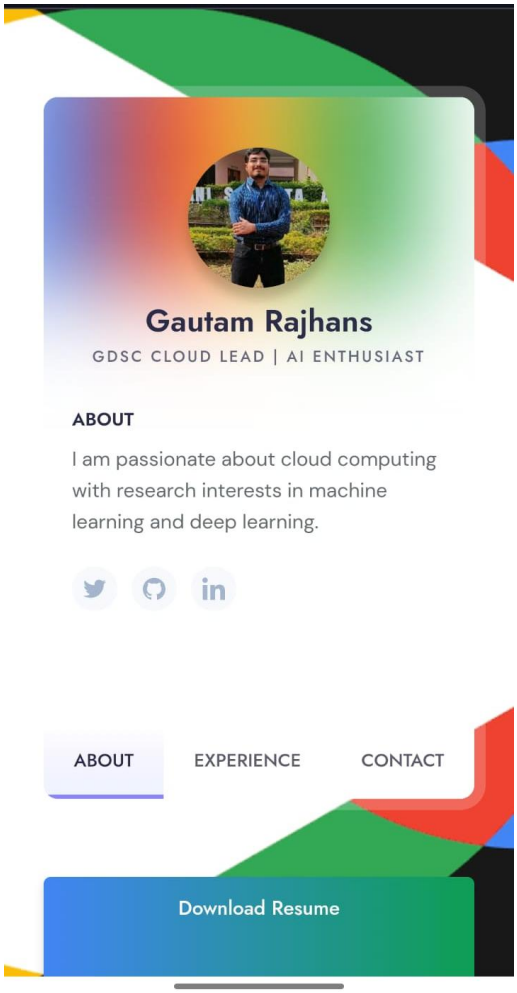
My portfolio website was created with HTML, CSS, and simple Javascript and is hosted on GitHub pages. It includes links to my social media accounts on LinkedIn, GitHub or Twitter as well as a button that allows users to download my CV. It also serves my purpose as it contains data related to my work experience and my academic pursuits, so it is not static. It is completely responsive and provides a satisfactory experience for both desktop and mobile users. The UI is very pleasant and has modern 3d animations in the background enhancing the design. The website is designed in such a way that the navigation is smooth within the site.

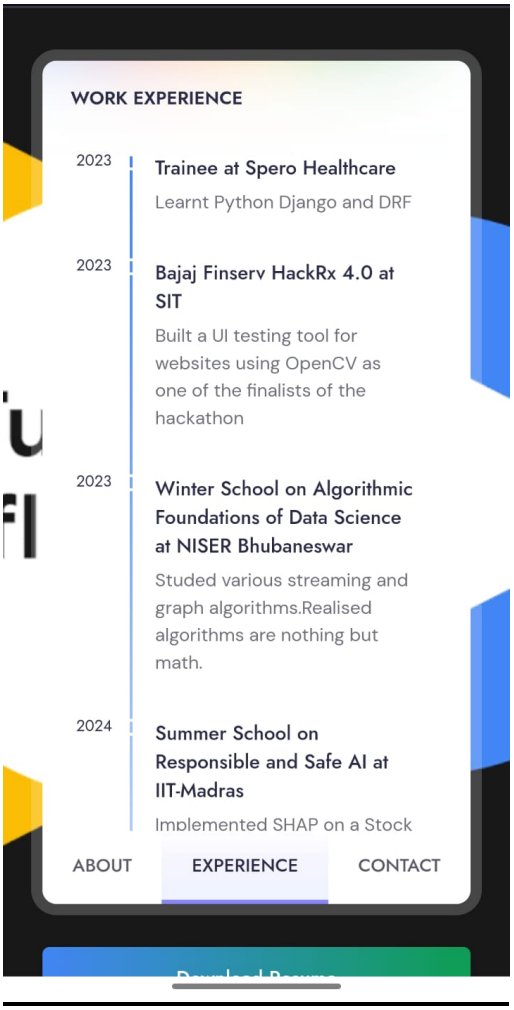
Images:

On Desktop



On Mobile


Gautam Rajhans
GDSC CLOUD LEAD | AI ENTHUSIAST
ABOUT
I am passionate about cloud computing with research interests in machine learning and deep learning.
ABOUT
EXPERIENCE
CONTACT
Download Resume


WORK EXPERIENCE
2023
Trainee at Spero Healthcare
Learnt Python Django and DRF
2023
Bajaj Finserv HackRx 4.0 at SIT
Built a UI testing tool for websites using OpenCV as one of the finalists of the hackathon
2023
Winter School on Algorithmic Foundations of Data Science at NISER Bhubaneswar
Studed various streaming and graph algorithms.Realised algorithms are nothing but math.
2024
Summer School on Responsible and Safe AI at IIT-Madras
Implemented SHAP on a Stock
ABOUT
EXPERIENCE
CONTACT