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	**Visting Mentor**
Mr. Ranjeet Bidwe	Mr. Moin Hasanfatta
Assistant Professor	

Submitted on: Nov 10, 2024

Department of Computer Science and Information Technology

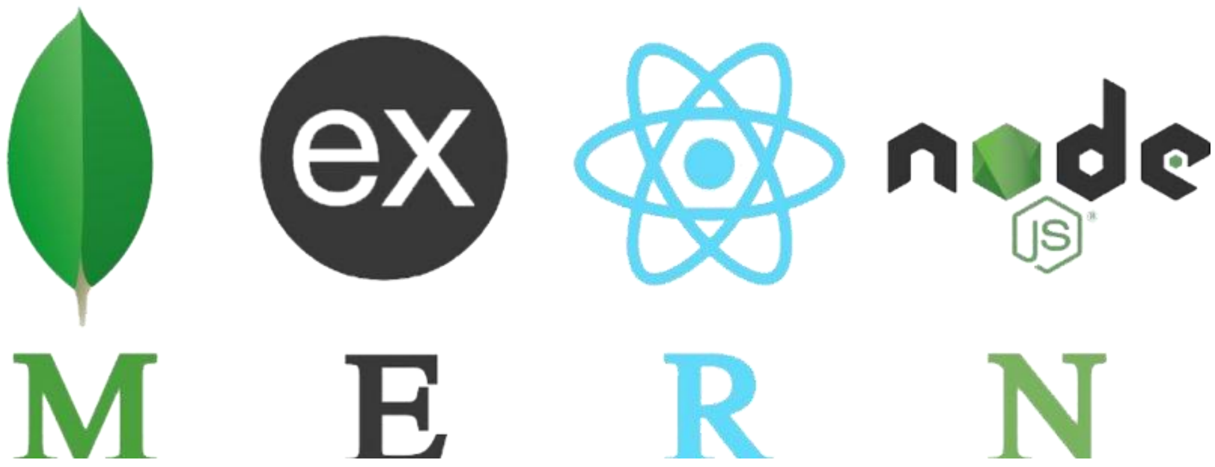
SYMBIOSIS INSTITUTE OF TECHNOLOGY, PUNE

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. CSS

CSS, which stands for 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. 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, 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.

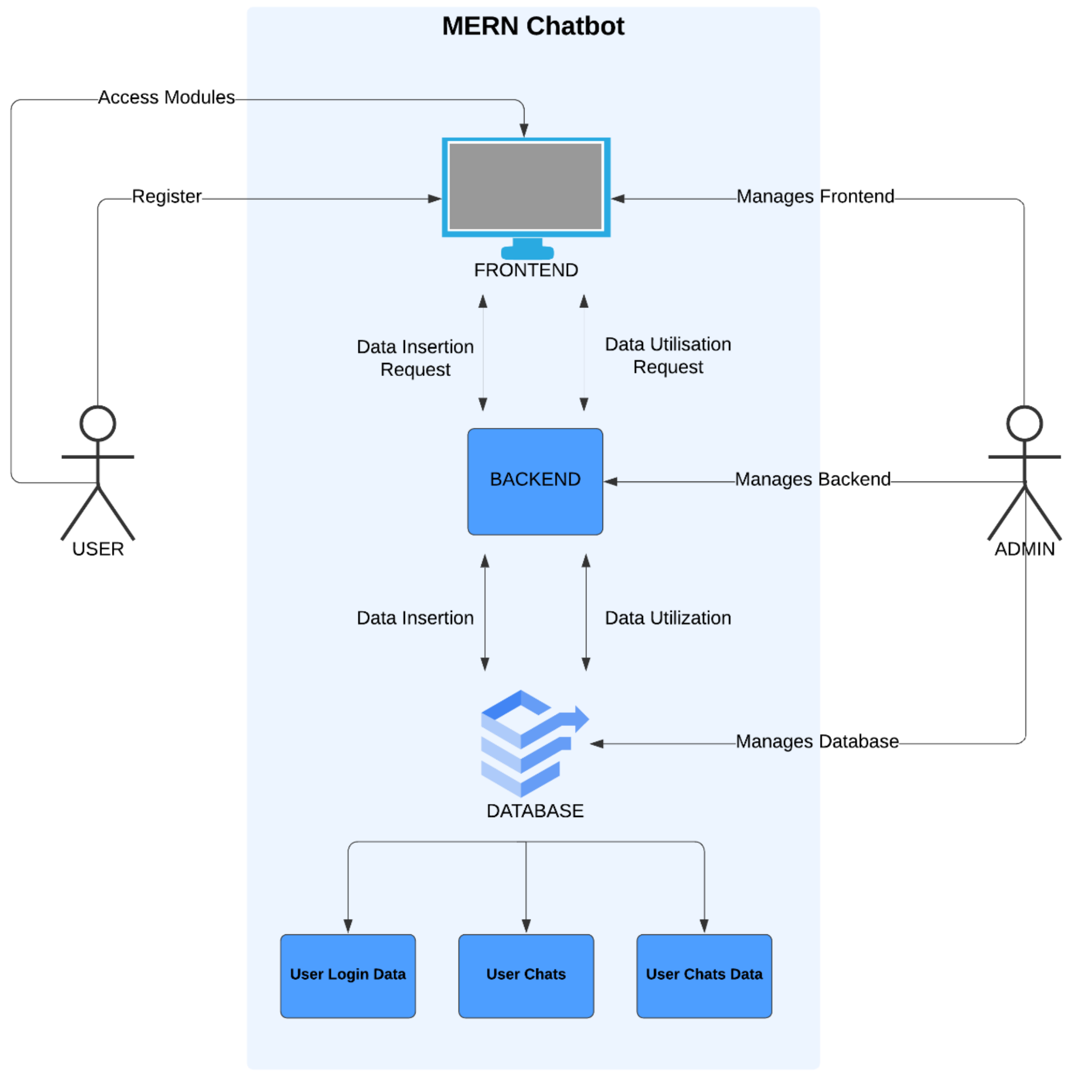
l. Postman

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**:
    - Deletes a chat if it belongs to the authenticated user.

3. userController.js — Handles User Authentication and Management

This file manages user login, OTP-based verification, and profile management.

- **loginUser**:
    - Checks if a user exists by their email.
    - If the user exists, it generates a JWT token for authentication.
    - If not, it generates an OTP and sends it to the user’s email for verification.

- **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**:
    - Retrieves and returns the authenticated user's profile details.

4. isAuth.js — Middleware for User Authentication

- **Purpose**: This middleware ensures that a user is authenticated before accessing protected routes.
- **Key Features**:
    - It checks for a JWT token in the request header.
    - If the token is valid, the user's data is retrieved from the database and attached to the req.user object.
    - If the token is missing or invalid, it responds with a 403 error.

5. sendMail.js — Handles OTP Sending via Email

- **Purpose**: This module is responsible for sending OTP emails to users.
- **Key Features**:
    - Uses nodemailer to send an email with the OTP to the user's provided email address.
    - The email is sent in an HTML format, styled with inline CSS for a better presentation.

6. chat.js — Chat Model

- **Purpose**: This file defines the MongoDB schema and model for the Chat collection.
- **Key Features**:
    - Each chat document contains a reference to the User and stores the latestMessage.
    - timestamps are enabled, so the document tracks when it was created or updated.

7. conversation.js — Conversation Model

- **Purpose**: Defines the schema for conversations between the user and the chatbot within a particular chat.
- **Key Features**:
  - Each conversation includes a reference to the related Chat, along with the question and answer fields.
  - timestamps are enabled for tracking when conversations were created or updated.

8. user.js — User Model

- **Purpose**: This file defines the MongoDB schema for the User collection.
- **Key Features**:
  - Each user document contains a unique email field.
  - timestamps are enabled to track when the user was created or updated.

9. chatRoutes.js — Handles Routes for Chat Operations

- **Purpose**: This file defines the routes for chat operations, using the chatController functions.
- **Key Routes**:
  - POST /new: Creates a new chat (requires authentication).
  - GET /all: Fetches all chats for the authenticated user.
  - POST /:id: Adds a new conversation to a chat.
  - GET /:id: Fetches all conversations for a specific chat.
  - DELETE /:id: Deletes a specific chat.

10. userRoutes.js — Handles Routes for User Operations

- **Purpose**: This file defines the routes related to user authentication and profile management.
- **Key Routes**:
  - POST /login: Handles user login and OTP sending.
  - POST /verify: Verifies the OTP and logs in the user.
  - GET /me: Fetches the authenticated user's profile.

General Flow (Backend)

1. **User Authentication**:
    - When a user attempts to log in, the backend either fetches the user from the database (if the user exists) or sends an OTP for verification (if the user doesn’t exist).
    - Once the OTP is verified, a new user is created, and a JWT is issued for further authentication.
2. **Chat Management**:
    - After logging in, the user can create a new chat, which is associated with their user ID.
    - Users can interact with the chatbot by submitting questions (which are saved as conversations in a chat).
    - Each chat document stores the latest message, and each conversation stores the question and answer pairs.
3. **Database Models**:
    - The User, Chat, and Conversation models represent the structure of your database documents.
    - MongoDB's ObjectId references are used to link users with their chats and conversations.
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 Check: If the user is not authenticated, they are prompted to log in.
3. Login: The user submits their email to log in, which triggers the authentication process.
4. OTP Verification: The user is sent an OTP, which they must enter to complete the login process.
5. Home Page: After successful verification, the user is directed to the home page, where they can manage their chats.
6. Sidebar: The sidebar lets users create, delete, and select chats.
7. Chat Interaction: Once a chat is selected, users can send prompts/messages and view responses from the backend.
8. Logout: The user can log out, which clears authentication data and redirects to the login page.

5. CHALLENGES AND SOLUTIONS

1. **Natural Language Processing (NLP) Complexity**

- **Challenge**: Implementing NLP into a chatbot to accurately interpret user input and generate relevant responses can be complex, especially with diverse phrasing and sentence structures.
- **Solution**: Using advanced NLP libraries or APIs, such as those provided by Gemini AI, helped mitigate this complexity. By leveraging pretrained models, the chatbot can identify keywords, intents, and respond contextually, simplifying natural language understanding without extensive manual training.

2. **Data Management and Scalability**

- **Challenge**: Storing and managing large volumes of chat logs and interaction data for analysis posed challenges, as scaling was necessary to handle increased traffic without degrading performance.
- **Solution**: MongoDB provided a flexible and scalable data storage solution. Its document-based structure allowed easy storage of structured and unstructured data, making it well-suited for high-traffic chat applications. Horizontal scaling and optimized indexing in MongoDB enabled handling of large data sets efficiently.

3. **Real-Time Communication**

- **Challenge**: Delivering real-time responses to multiple users simultaneously is essential for an interactive chatbot, but achieving this in a single-page React application without latency can be challenging.
- **Solution**: Using WebSocket connections through Node.js and Express allowed the chatbot to establish real-time communication with user. This enabled bidirectional data flow, reducing response time and supporting interaction without repeated API requests.

4. **User Interface Optimization**

- **Challenge**: Creating an intuitive and responsive user interface where users can easily engage with the chatbot while maintaining performance across devices and screen sizes was essential.
- **Solution**: React's component-based structure enabled modular UI design, while CSS and frameworks like Tailwind helped maintain responsive layouts. Leveraging Vite as the build tool also improved efficiency, enhancing performance during UI updates.

5. **Error Handling and User Experience**

- **Challenge**: Managing errors gracefully, especially during network disruptions or incorrect inputs, is critical for user experience.
- **Solution**: Axios allowed handling HTTP requests with error responses to provide fallback messages and ensure that users are informed of issues. Implementing validation checks on user input reduced miscommunication and helped the chatbot respond more accurately.

6. CONCLUSION

The MERN-based chatbot project illustrates the powerful synergy between the MERN stack and AI technologies, resulting in a user-friendly and scalable conversational platform. With MongoDB, Express, React, and Node.js as its foundation, this chatbot effectively handles real-time interactions and data processing needs. MongoDB's document-based structure allows for efficient management of dynamic chat logs and user data, while Express and Node.js handle the backend processes, enabling fast, reliable server responses. React provides a seamless single-page application interface, allowing users to engage fluidly with the chatbot on any device. Together, these components form a robust infrastructure, optimized for responsive and interactive user experiences.

Integrating AI and NLP capabilities into the chatbot elevates its conversational abilities, enabling it to parse language with contextual understanding and respond intelligently to diverse inputs. Through the use of pretrained NLP models such as those in Gemini AI, the chatbot can accurately detect user intent, recognize keywords, and deliver responses that feel natural and contextually relevant. This integration allows the chatbot to move beyond basic, rule-based responses, instead offering a more adaptive, human-like interaction that keeps users engaged and satisfied. By minimizing the need for repetitive training and achieving natural language comprehension, the chatbot demonstrates AI's role in enhancing real-time customer interactions.

This project addresses major limitations in traditional customer service models, such as the inability to handle large volumes of routine queries efficiently. By processing simple inquiries in real-time, the chatbot shortens wait times and frees human agents to focus on complex issues, ultimately improving customer satisfaction. This approach also highlights how AI-driven chatbots can provide scalable solutions that streamline support operations. With a strong technological foundation, this chatbot exemplifies the potential of AI in transforming customer service into a faster, more responsive system that meets evolving user expectations.

7. APPENDIX 1

7.1 ADDITIONAL RESOURCE

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

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

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

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

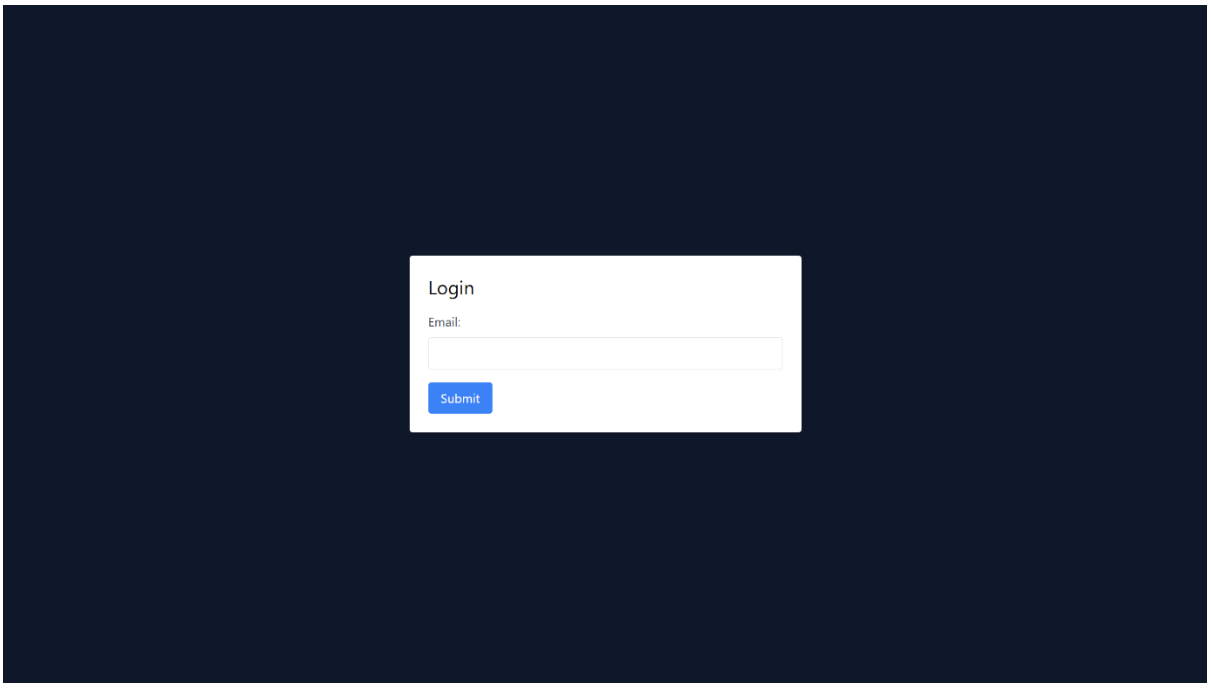
Figma Link: https://www.figma.com/design/2geACCVwnOUAMsbwaLmhzl/Mern-Chatbot?node-id=0-1&t=12LVok9ZDTDWZxH7-1

7.2 REFERENCES

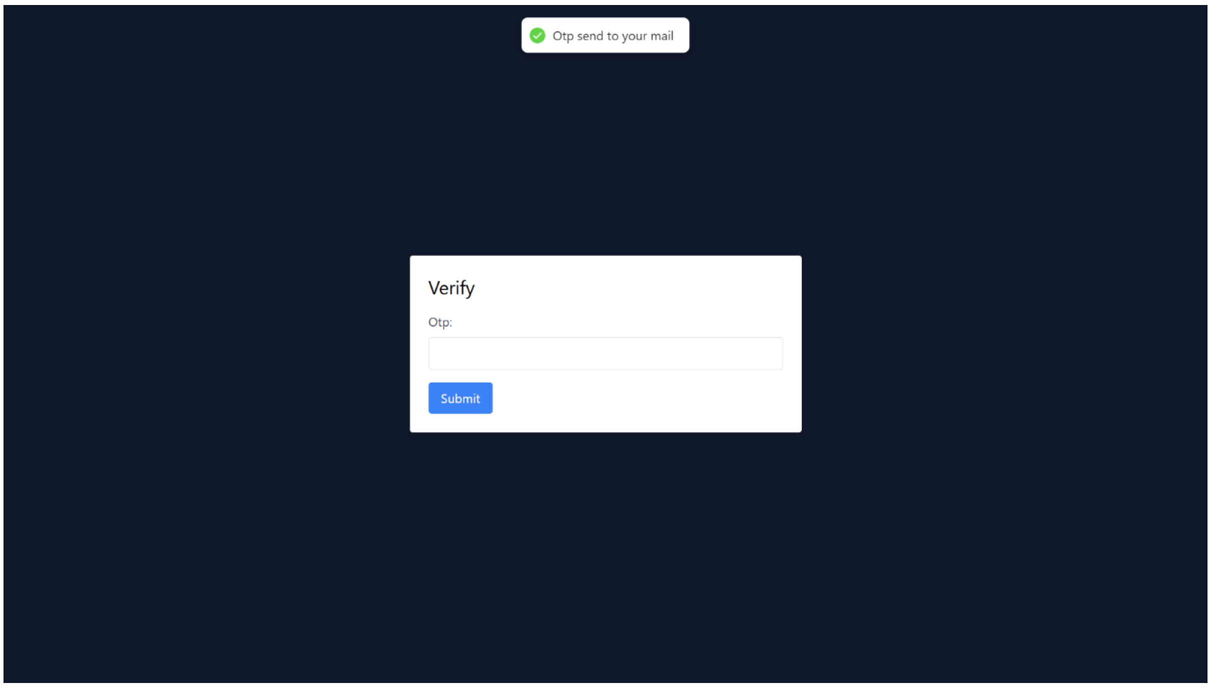
- https://www.geeksforgeeks.org/mern-stack/
- https://www.freecodecamp.org/news/build-an-ai-chatbot-with-the-mern-stack/
- https://medium.com/@sanikakotgire2003/building-your-own-ai-chat-bot-using-mern-mongodb-express-react-and-node-js-stack-306a37a4845d
- https://youtu.be/YZCQafLGuz8?si=r8JQ2pSO-XB2Wfey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

7.3 SCREENSHOTS

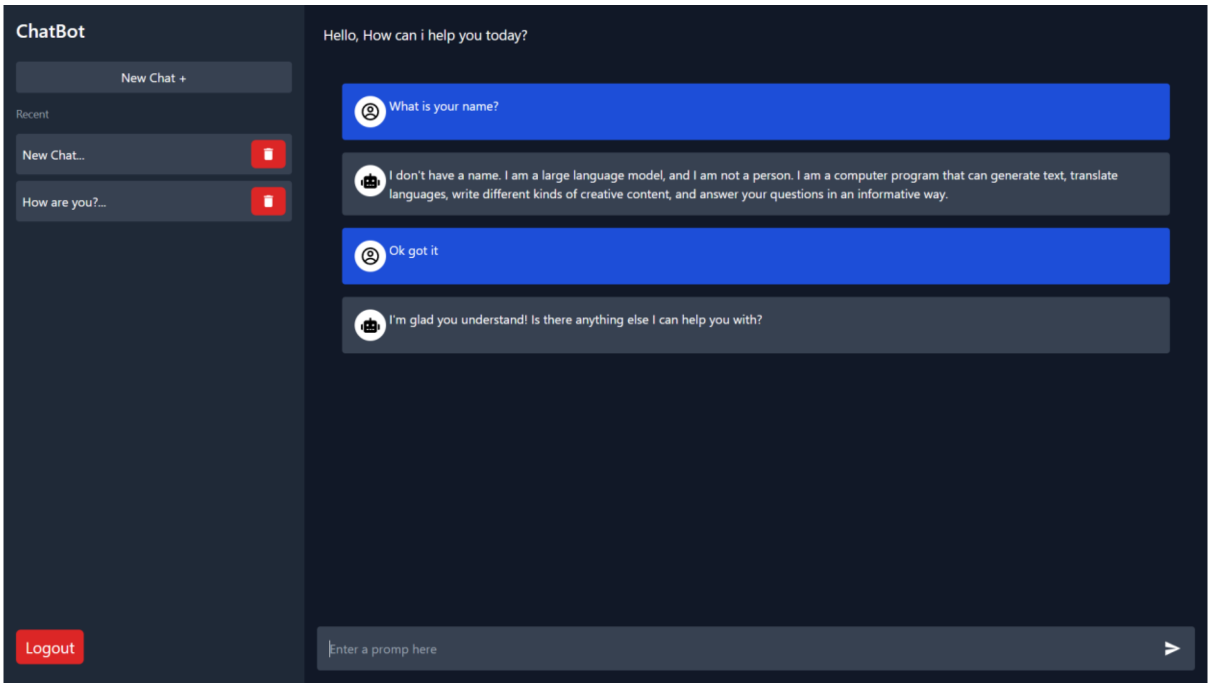
Website



Login



OTP Verification



Hello, How can i help you today?

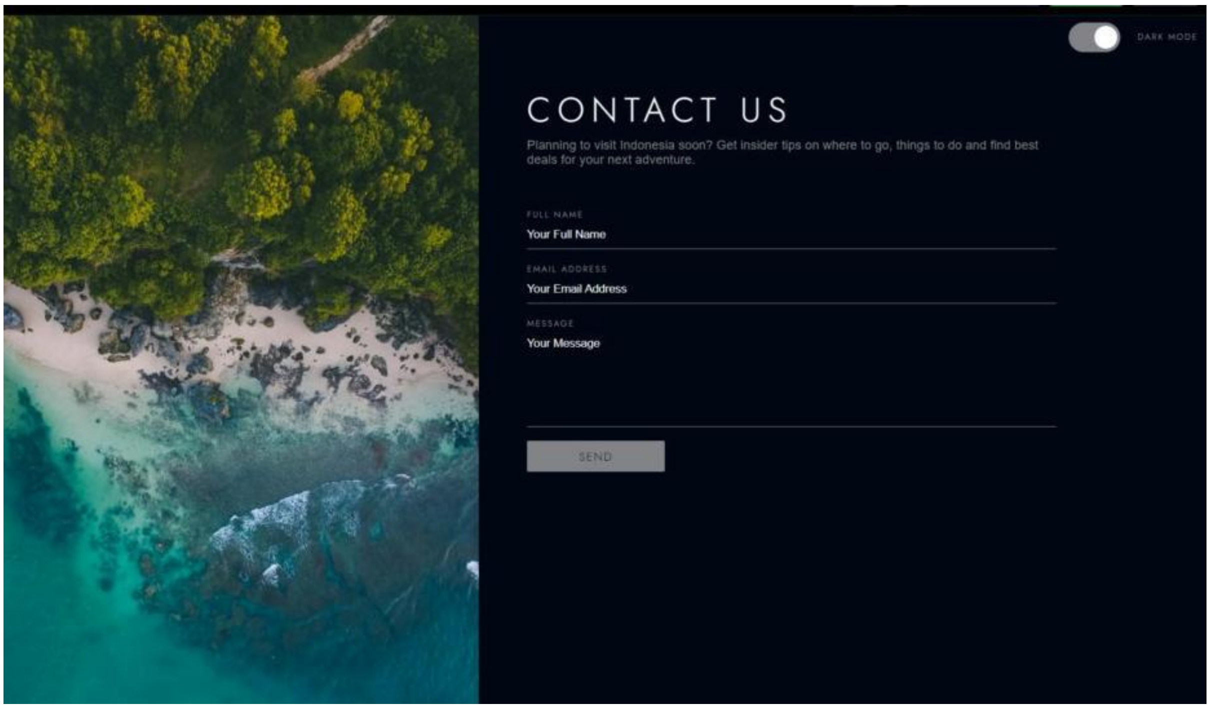
What is your name?

I don't have a name. I am a large language model, and I am not a person. I am a computer program that can generate text, translate languages, write different kinds of creative content, and answer your questions in an informative way.

Ok got it

I'm glad you understand! Is there anything else I can help you with?

Home Page



CONTACT US

Planning to visit Indonesia soon? Get insider tips on where to go, things to do and find best deals for your next adventure.

FULL NAME

Your Full Name

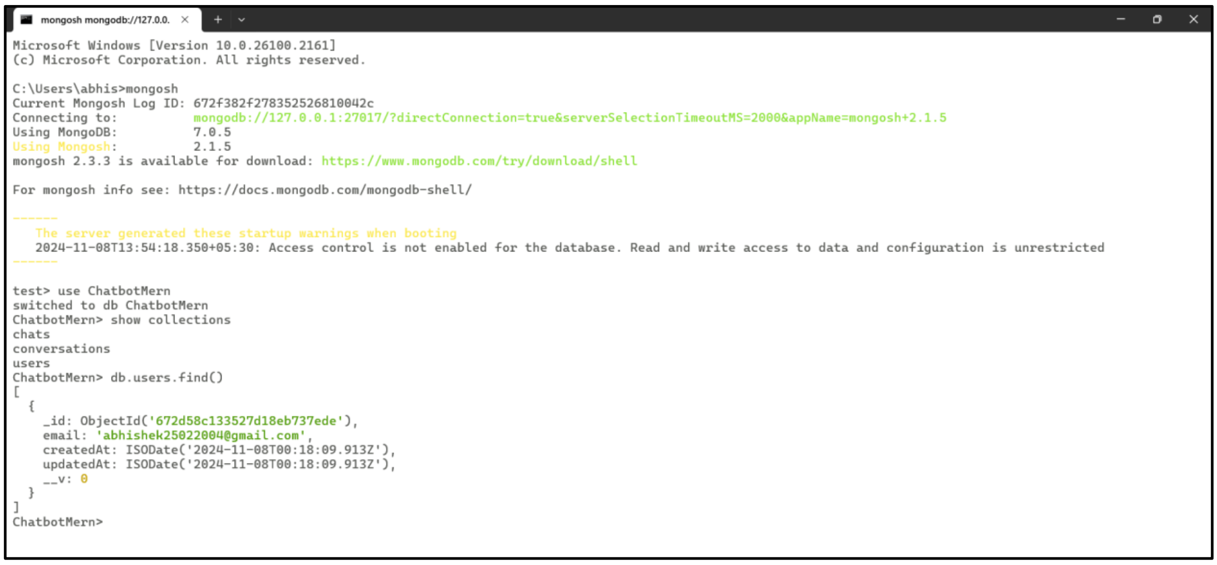
EMAIL ADDRESS

Your Email Address

MESSAGE

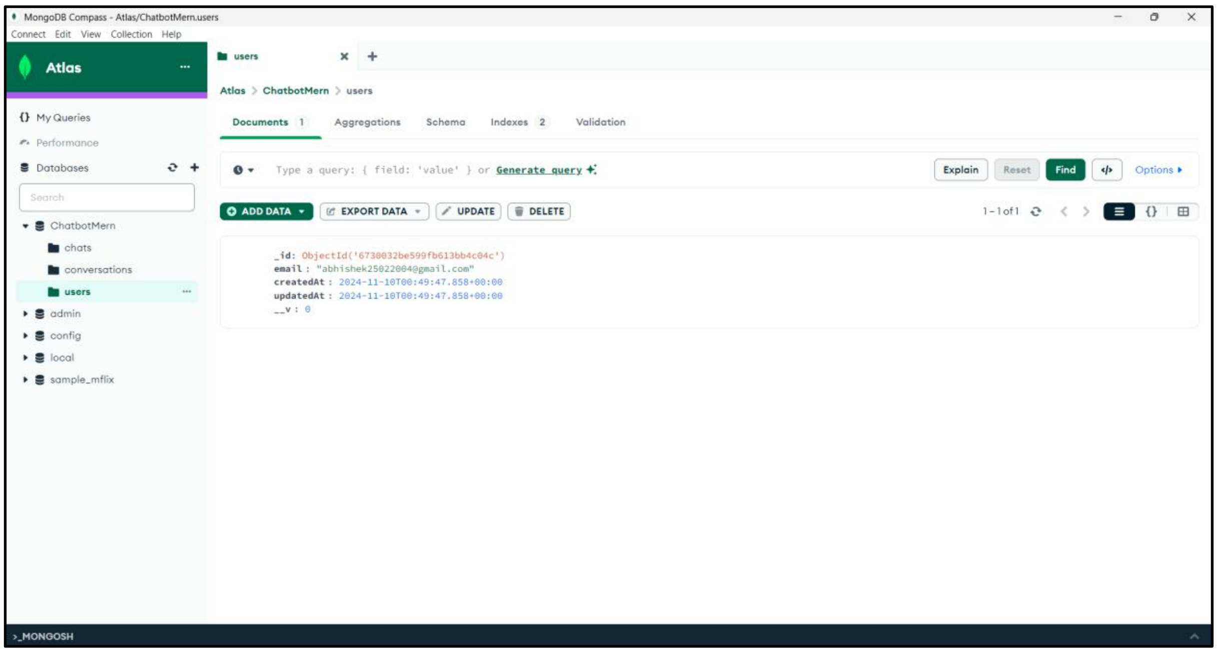
Your Message

Contact Us Page



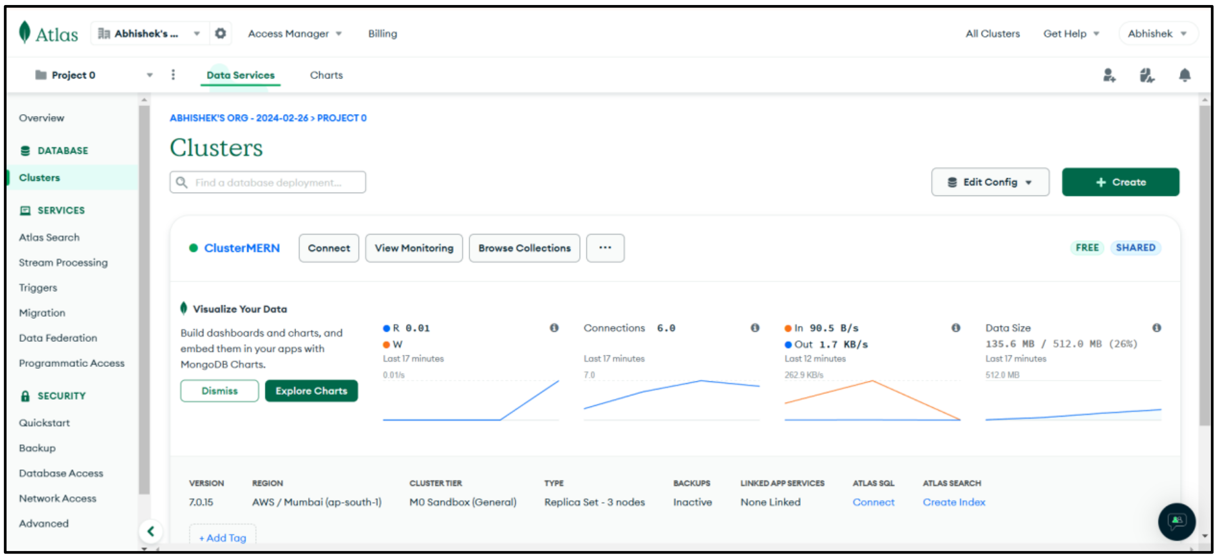
```
Microsoft Windows [Version 10.0.26100.2161]
(c) Microsoft Corporation. All rights reserved.

C:\Users\abhis>mongosh
Current Mongosh Log ID: 672f382f278352526810042c
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.1.5
Using MongoDB:          7.0.5
Using Mongosh:          2.1.5
mongosh 2.3.3 is available for download: https://www.mongodb.com/try/download/shell

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2024-11-08T13:54:18.350+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
------

test> use ChatbotMern
switched to db ChatbotMern
ChatbotMern> show collections
chats
conversations
users
ChatbotMern> db.users.find()
[
  {
    _id: ObjectId('672d58c133527d18eb737ede'),
    email: 'abhishek25022004@gmail.com',
    createdAt: ISODate('2024-11-08T00:18:09.913Z'),
    updatedAt: ISODate('2024-11-08T00:18:09.913Z'),
    __v: 0
  }
]
ChatbotMern>
```

MongoDB



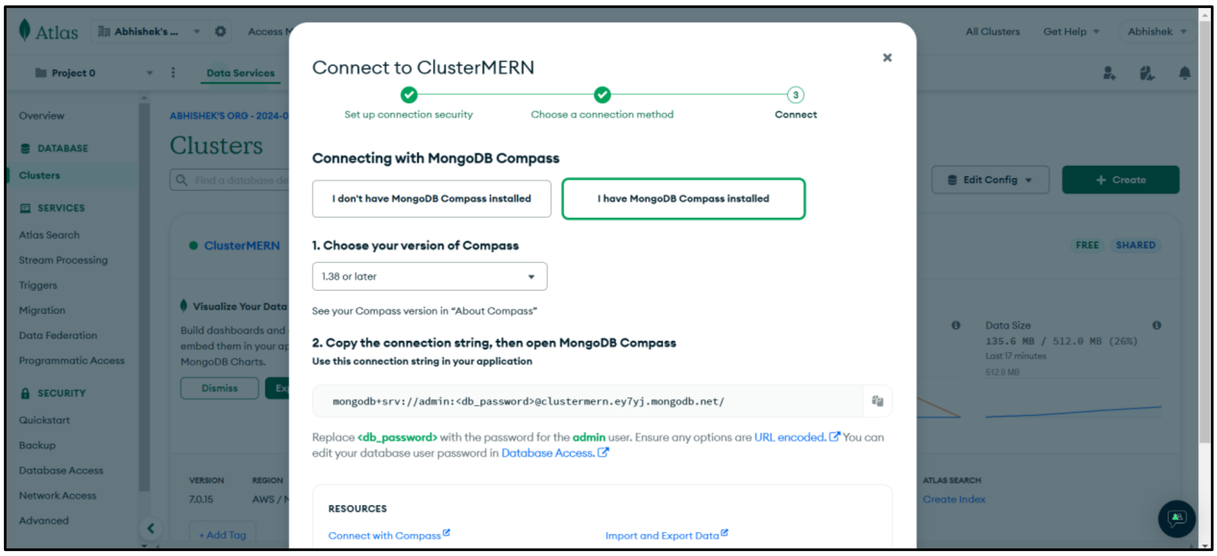
```
_id: ObjectId('6730032be599fb613bb4c04c')
email : "abhishek25022004@gmail.com"
createdAt : 2024-11-10T00:49:47.858+00:00
updatedAt : 2024-11-10T00:49:47.858+00:00
__v : 0
```

MongoDB



VERSION	REGION	CLUSTER TIER	TYPE	BACKUPS	LINKED APP SERVICES	ATLAS SQL	ATLAS SEARCH
7.0.15	AWS / Mumbai (ap-south-1)	M0 Sandbox (General)	Replica Set - 3 nodes	Inactive	None Linked	Connect	Create Index

MongoDB Atlas Cluster



Connect to ClusterMERN

Set up connection security — Choose a connection method — Connect

Connecting with MongoDB Compass

I don't have MongoDB Compass installed | I have MongoDB Compass installed

1. Choose your version of Compass

1.38 or later

See your Compass version in "About Compass"

2. Copy the connection string, then open MongoDB Compass

Use this connection string in your application

```
mongodb+srv://admin:<db_password>@clustermern.ey7yj.mongodb.net/
```

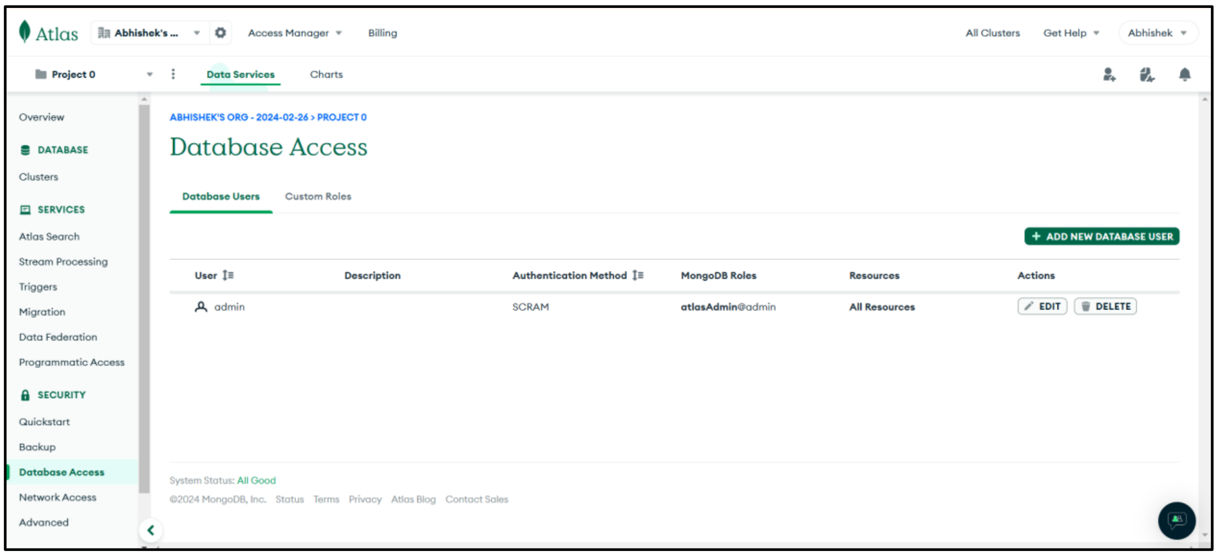
Replace **<db_password>** with the password for the **admin** user. Ensure any options are URL encoded. You can edit your database user password in Database Access.

RESOURCES

Connect with Compass — Import and Export Data

Data Size 135.6 MB / 512.0 MB (26%)

MongoDB Atlas URL



ABHISHEK'S ORG - 2024-02-26 > PROJECT 0

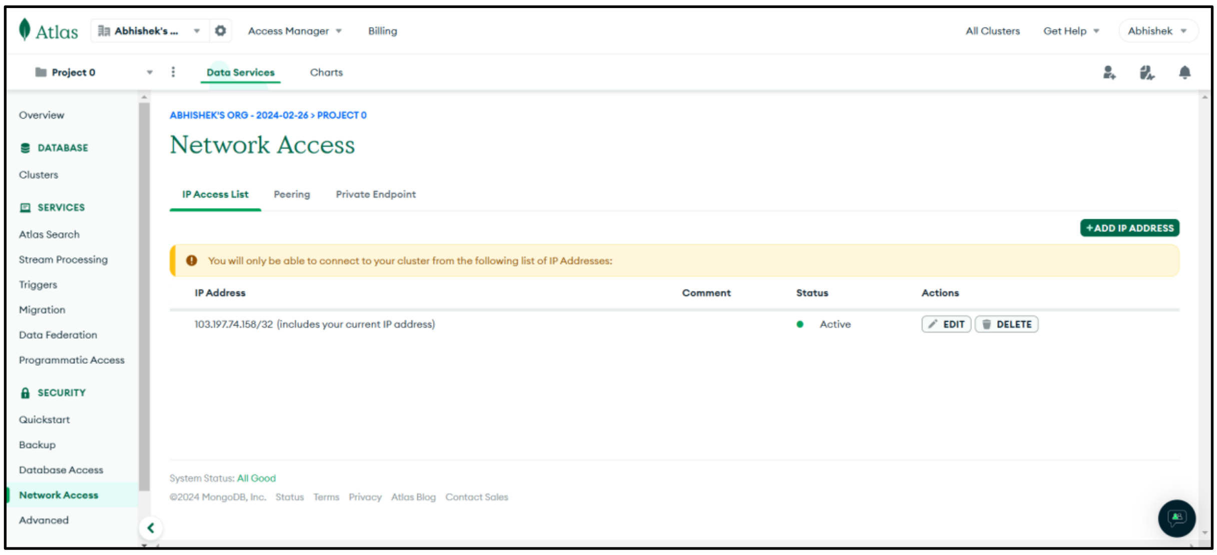
Database Access

User	Description	Authentication Method	MongoDB Roles	Resources	Actions
admin		SCRAM	atlasAdmin@admin	All Resources	EDIT DELETE

System Status: All Good

©2024 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales

MongoDB Atlas Database Access

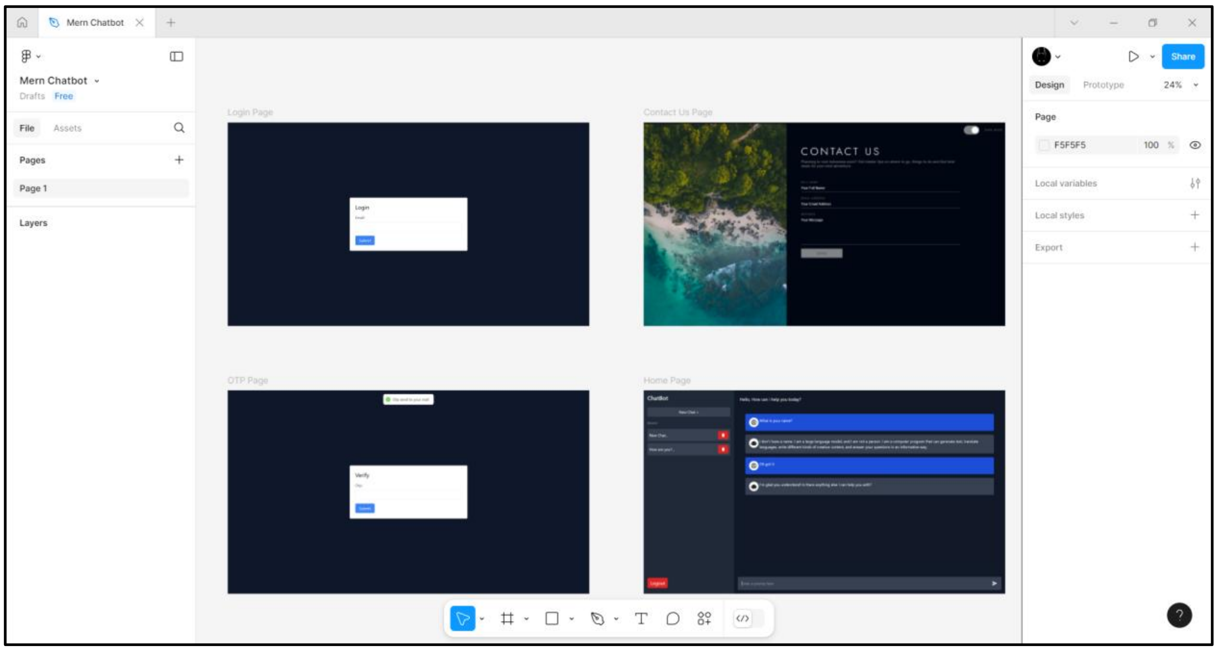


Network Access

You will only be able to connect to your cluster from the following list of IP Addresses:

IP Address	Comment	Status	Actions
103.197.74.158/32 (includes your current IP address)		● Active	EDIT DELETE

MongoDB Atlas Network Access



Figma

8. APPENDIX 2

8.1 PERSONAL PORTFOLIO AND TECHNICAL DETAILS

PRN: 22070122002

Name: Aaryan Dhawan

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

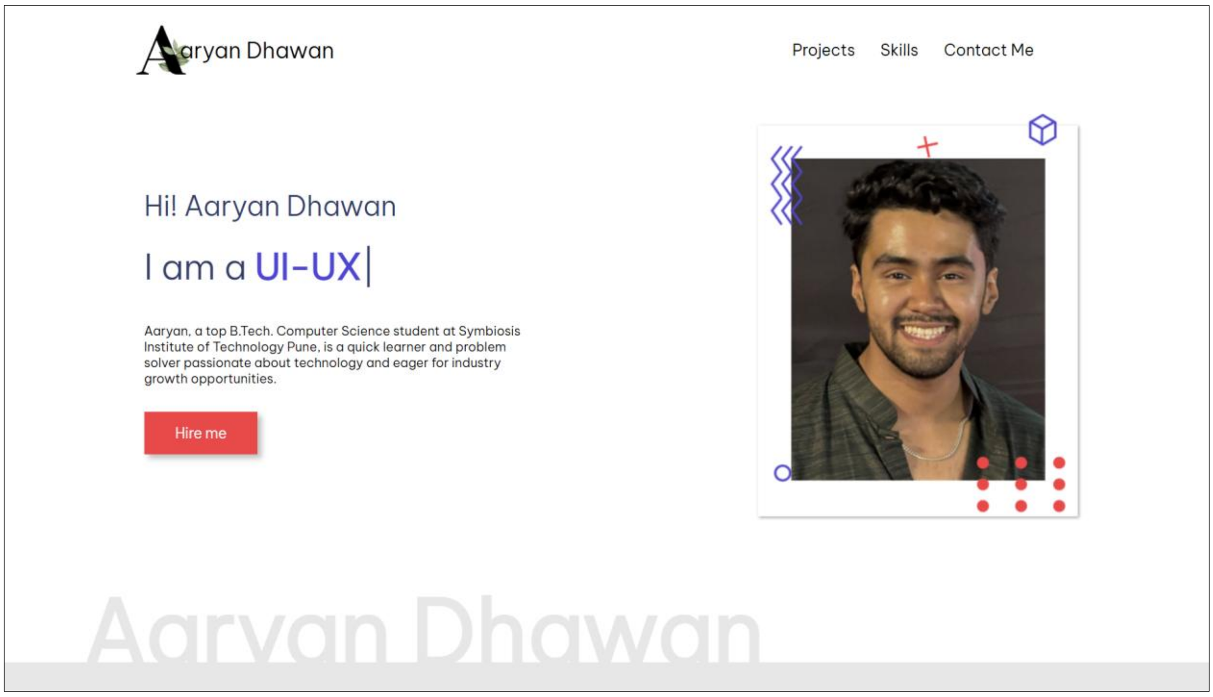
Repository Link: https://github.com/dhawanaaaryan/portfolio

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:

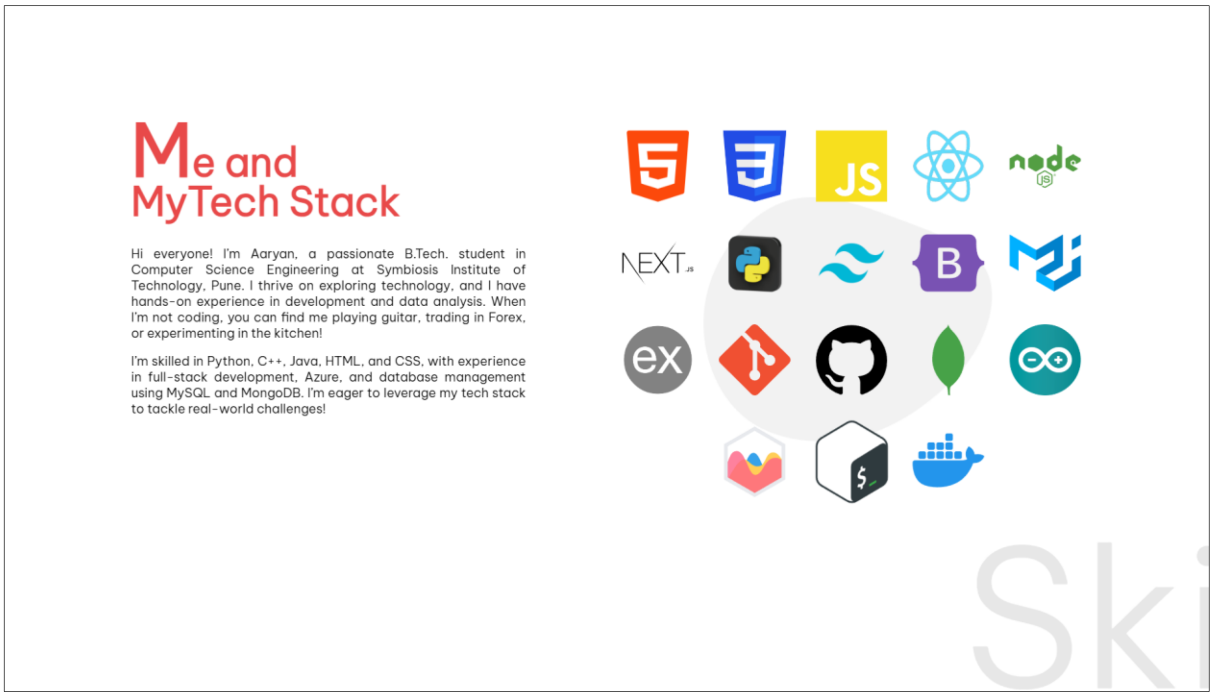
On Desktop



Hi! Aaryan Dhawan

I am a UI-UX|

Aaryan, a top B.Tech. Computer Science student at Symbiosis Institute of Technology Pune, is a quick learner and problem solver passionate about technology and eager for industry growth opportunities.

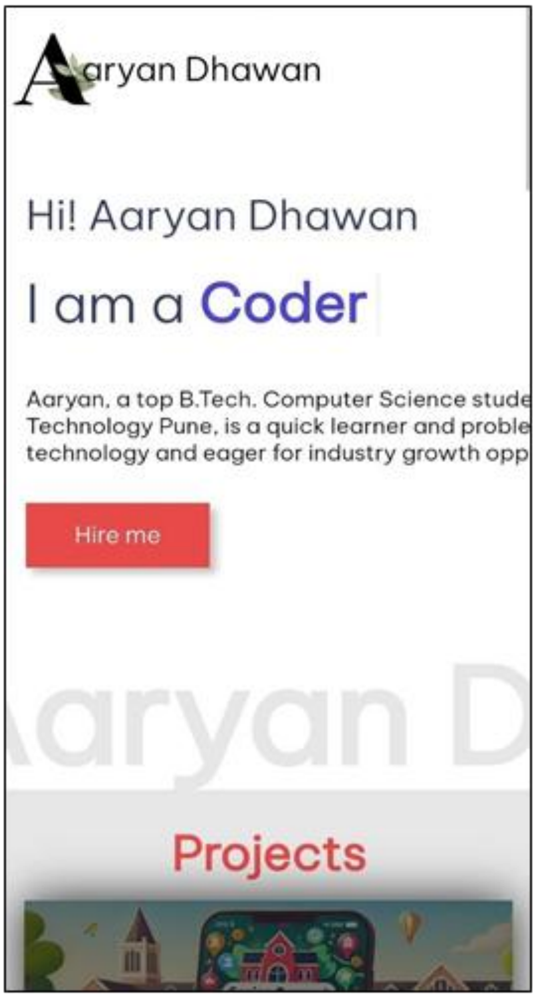


Me and MyTech Stack

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

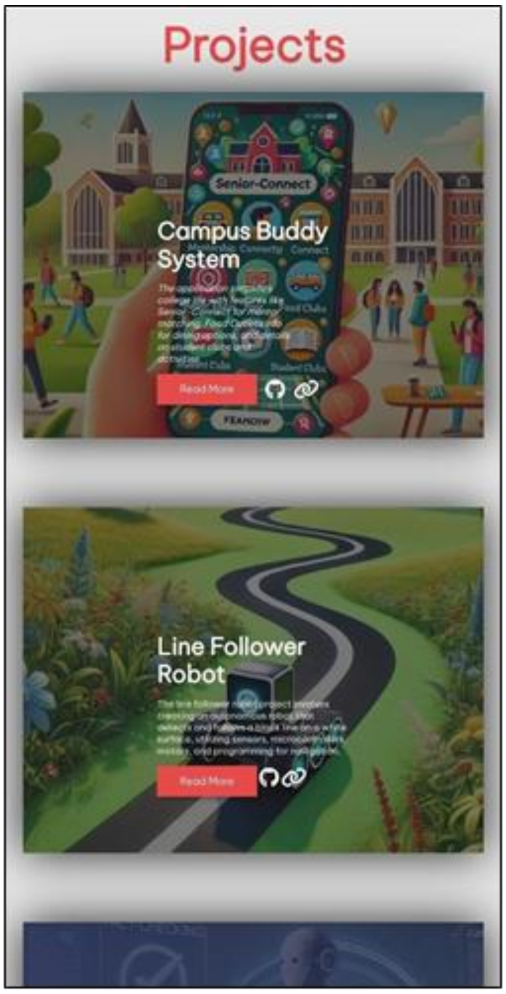


Aaryan Dhawan

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



PRN: 22070122007

Name: Abhishek Rajput

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

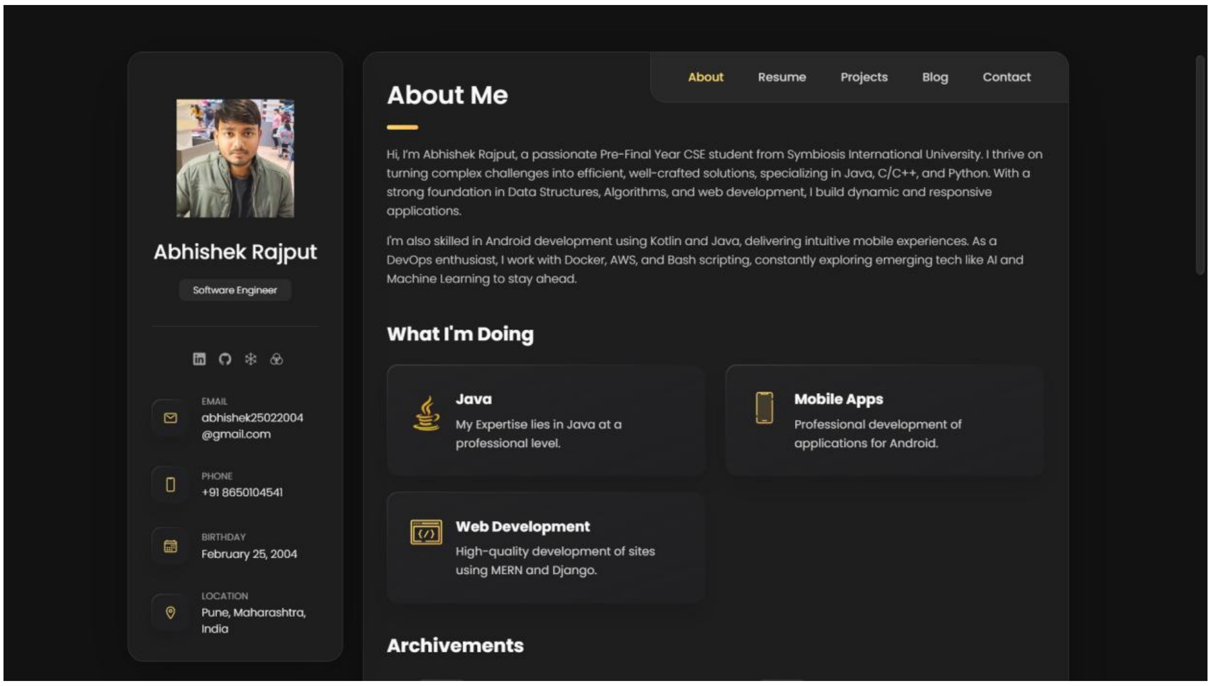
Repository Link: https://github.com/Abhishek-2502/Portfolio

Technical Details:

The portfolio website is a multi-page, fully responsive platform crafted with HTML, CSS, and JavaScript. It follows a modern, clean aesthetic that emphasizes both functionality and ease of navigation. With dedicated pages for the resume, blog, projects, and contact information, the site provides a comprehensive view of professional details. Responsive design principles ensure optimal viewing across various devices, while a modular structure allows for scalable updates. The layout highlights projects and technical skills, utilizing subtle animations and transitions that contribute to a polished, user-friendly experience.

Images:

On Desktop



Abhishek Rajput

Software Engineer

EMAIL
abhishek25022004@gmail.com

PHONE
+91 8650104541

BIRTHDAY
February 25, 2004

LOCATION
Pune, Maharashtra, India

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.

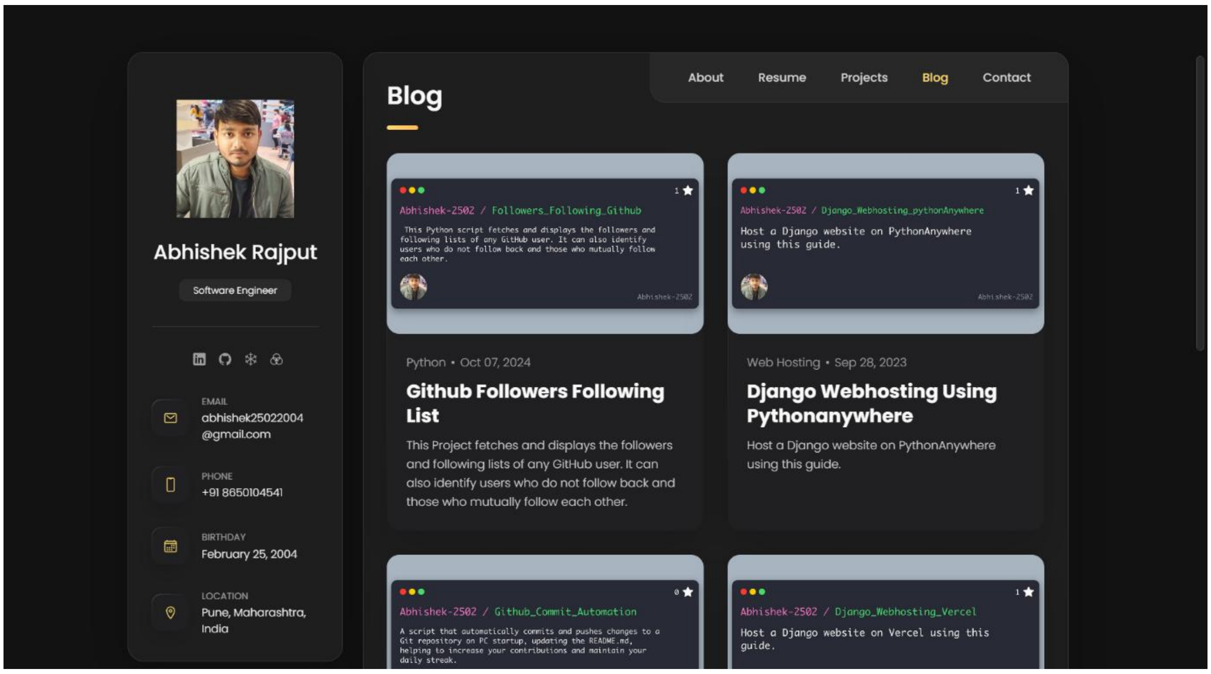
What I'm Doing

Java
My Expertise lies in Java at a professional level.

Mobile Apps
Professional development of applications for Android.

Web Development
High-quality development of sites using MERN and Django.

Archivements


Abhishek Rajput

Software Engineer

EMAIL
abhishek25022004
@gmail.com

PHONE
+91 8650104541

BIRTHDAY
February 25, 2004

LOCATION
Pune, Maharashtra, India

About Resume Projects Blog Contact

Blog

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.

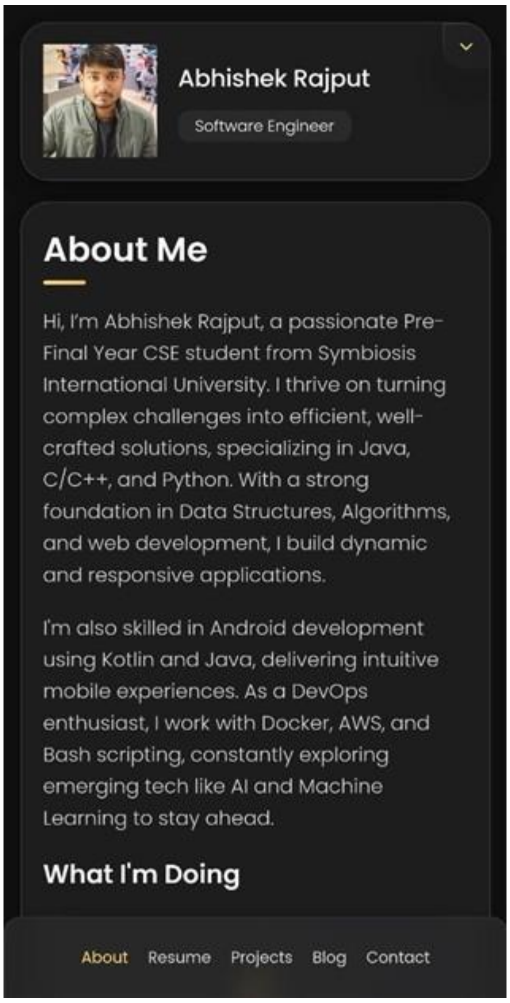
Abhishek-2502 / Github_Commit_Automation

A script that automatically commits and pushes changes to a Git repository on PC startup, updating the README.md, helping to increase your contributions and maintain your daily streak.

Abhishek-2502 / Django_Webhosting_Vercel

Host a Django website on Vercel using this guide.

On Mobile


Abhishek Rajput

Software Engineer

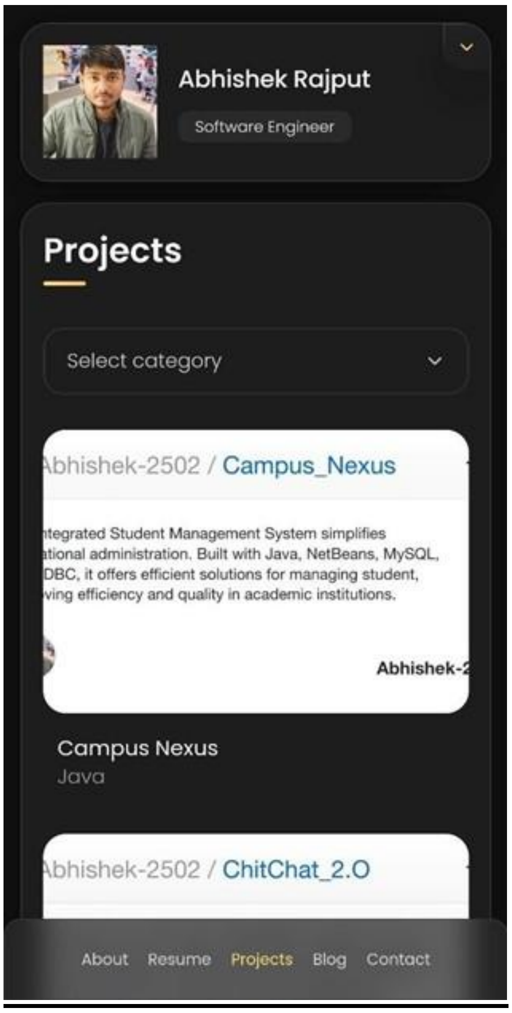
About Me

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

bhishek-2502 / Campus_Nexus

tegrated Student Management System simplifies
tional administration. Built with Java, NetBeans, MySQL,
DBC, it offers efficient solutions for managing student,
ving efficiency and quality in academic institutions.

Abhishek-2

Campus Nexus
Java

bhishek-2502 / ChitChat_2.O

About Resume Projects Blog Contact

PRN: 22070122030

Name: Arnav Jain

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

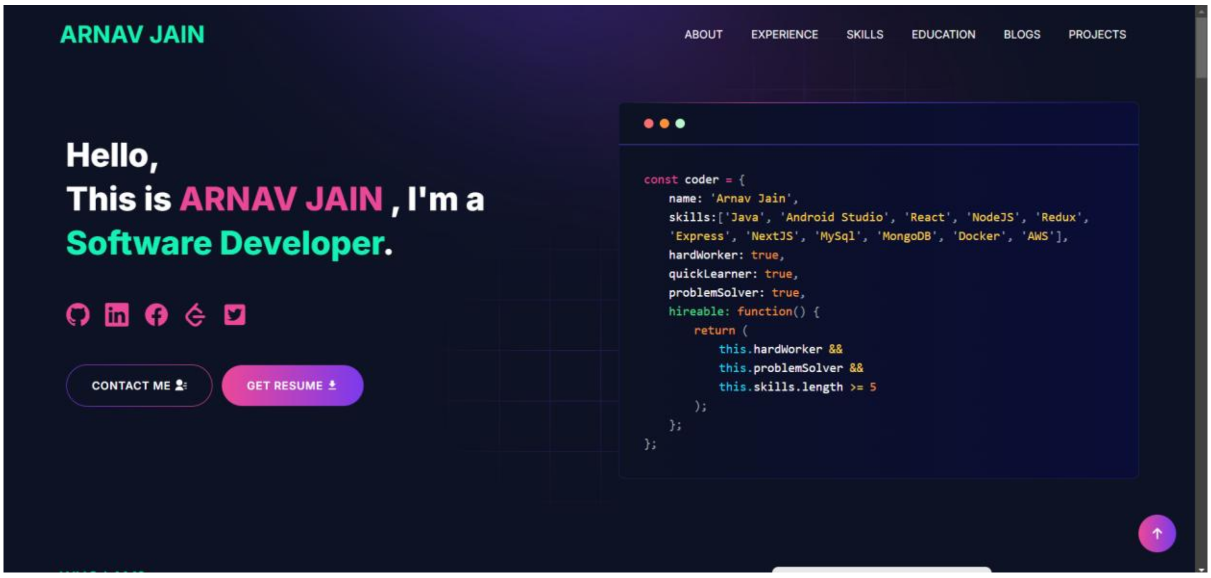
Repository Link: https://github.com/Arnavjain2503/My-Portfolio

Technical Details:

My portfolio is a dynamic platform that showcases the expertise and projects in a clean, engaging layout. Built with a developer-first mindset, it features dedicated sections for the skills, experience, projects, education, and blog, giving a well-rounded view of my background. Designed with user experience, the portfolio is build using HTML, CSS, JS and integrates modern tools like Next.js, Tailwind CSS, and React, ensuring a smooth, responsive interface.

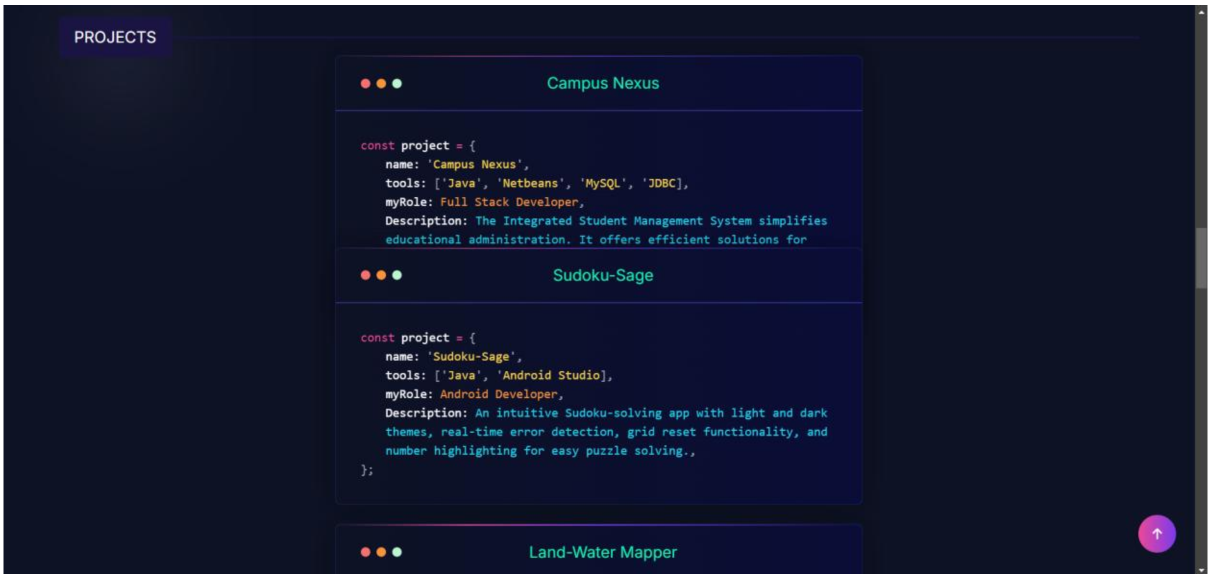
Images:

On Desktop



Hello,
This is ARNAV JAIN , I'm a
Software Developer.

```
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
        );
    };
};
```



Campus Nexus

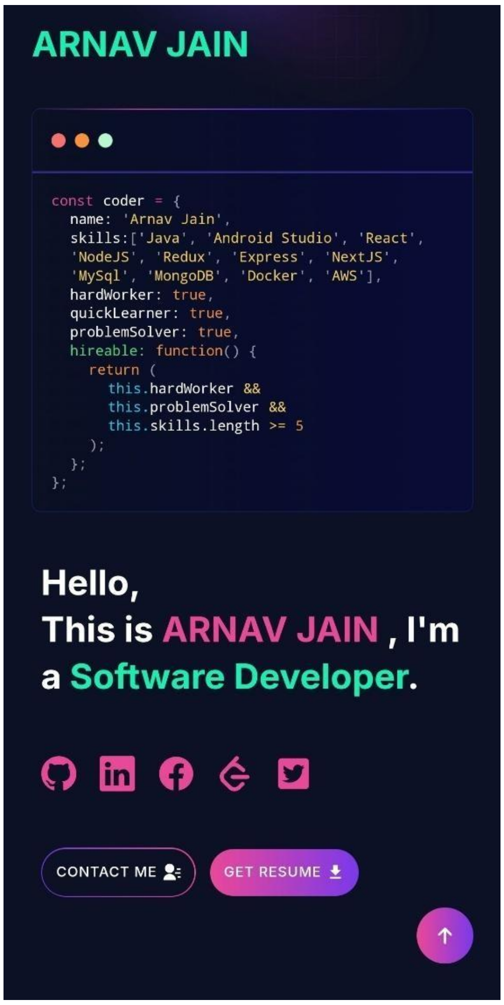
```
const project = {
    name: 'Campus Nexus',
    tools: ['Java', 'Netbeans', 'MySQL', 'JDBC],
    myRole: Full Stack Developer,
    Description: The Integrated Student Management System simplifies
    educational administration. It offers efficient solutions for
```

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

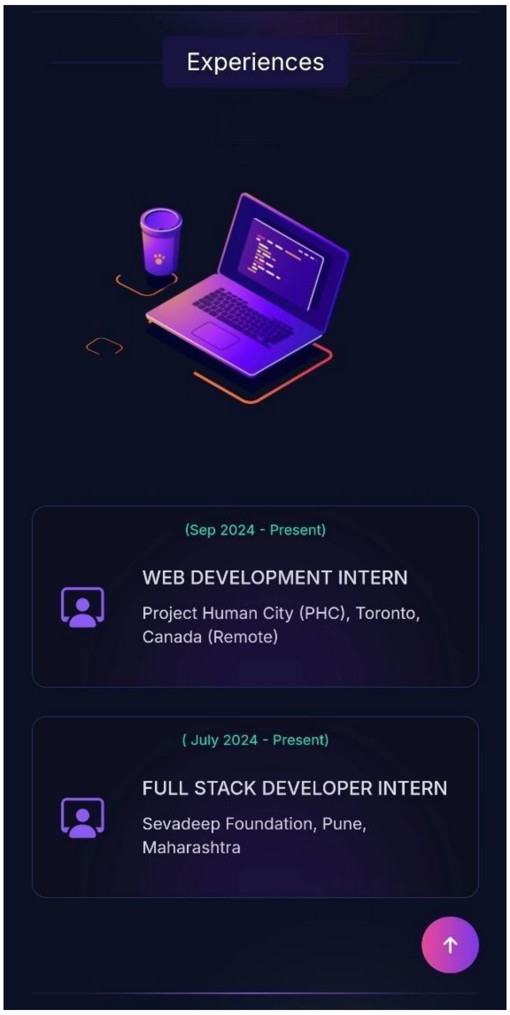
On Mobile



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.



(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/

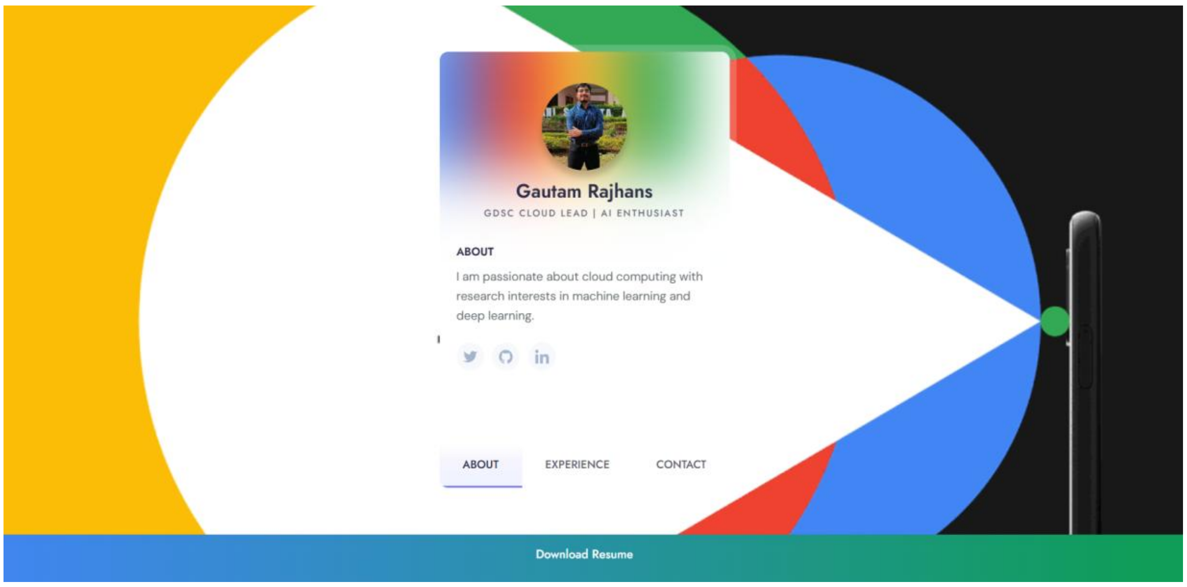
Repository Link: capricode-ui/capricode-ui.github.io

Technical Details:

My portfolio is a made using a combination of HTML, CSS and JavaScript and is hosted on GitHub Pages. It has various features like linking to my social media accounts on LinkedIn, GitHub and Twitter and provides a button to download my resume. My portfolio also showcases my experience and my academic interests and is dynamic in nature. Its fully responsive and works well on desktops as well as mobile devices. The UI is highly aesthetic with modern 3D animations in the background which adds appeal. The website is made so that the user will have a smooth experience while using it.

Images:

On Desktop



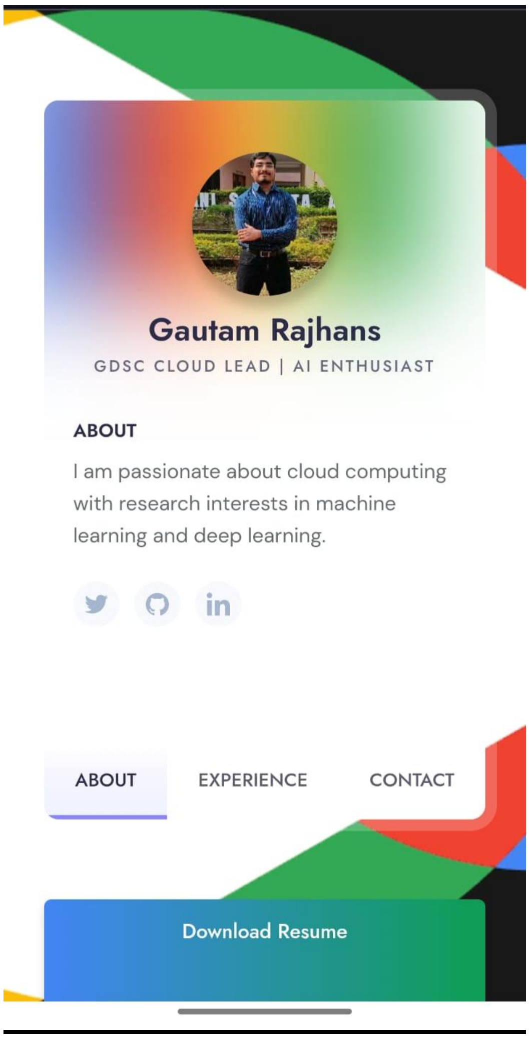
Gautam Rajhans

GDSC CLOUD LEAD | AI ENTHUSIAST

ABOUT

I am passionate about cloud computing with research interests in machine learning and deep learning.

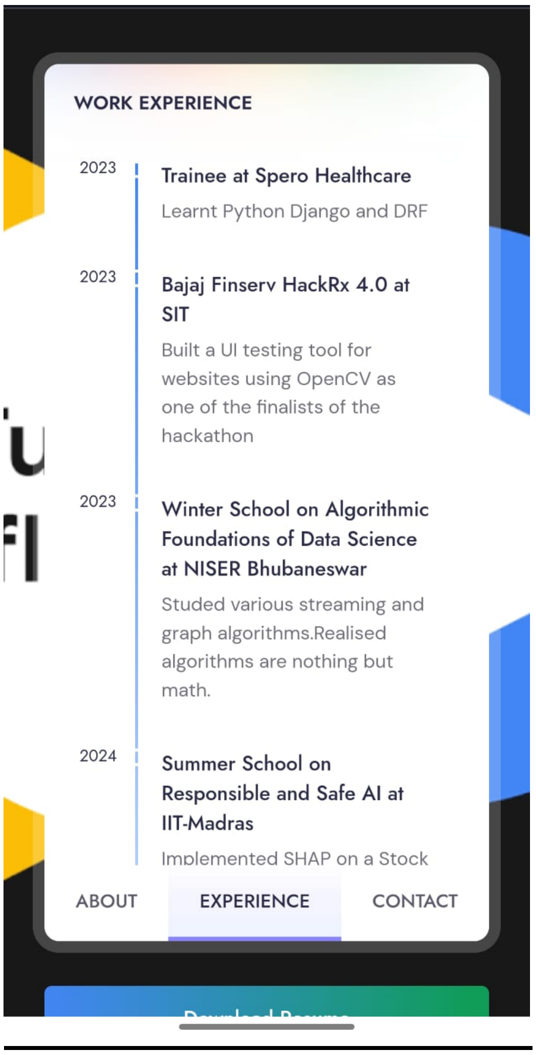
On Mobile


Gautam Rajhans

GDSC CLOUD LEAD | AI ENTHUSIAST

ABOUT

I am passionate about cloud computing with research interests in machine learning and deep learning.


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