**FIIT**

**FUTURA INNOVATIVE INFORAMRTION TECHNOLOGY**

THE PROJECT WAS RELATED TO THE COURSE

MERN STACK

On The Topic of Gemini Cloning Website

SUBMITTED BY:

**DAVID SHALOM M**

UNDER THE GUIDANCE OF

**AKSHAYA S**

FIIT – FUTURA INNOVATIVE INFORMATION TECHNOLOGY

#1545 1st FLOOR, VELLORE MAIN ROAD,

VENGIKKAL, TIRUVANNAMALAI

**DECLARATION**

I, David, hereby affirm that the documentation, development workflow, and implementation details outlined in this report are accurate and created in good faith as part of my internship. The project is a MERN stack-based web application inspired by Gemini, developed to showcase secure and scalable user authentication mechanisms along with responsive interface design.

The platform demonstrates proficiency in integrating MongoDB, Express.js, React.js, and Node.js, with emphasis on structured route handling, authentication workflows, and a user-centric experience. All technical aspects—including backend logic, middleware integration, frontend responsiveness, and database management—have been thoroughly considered and executed to the best of my ability.

This report reflects the intended architecture and functionality of the MERN stack application and serves as a record of my progress, learning, and commitment to developing reliable full-stack solutions. I remain dedicated to further refining the system to enhance performance, usability, and security.

**Contents**

[**Introduction** 4](#_Toc170947156)

[Objectives: 7](#_Toc170947157)

[**Project Description** 8](#_Toc170947158)

[Tools 1](#_Toc170947159)1

[Requirements: 11](#_Toc170947160)

[Source Code 1](#_Toc170947161)2

[Output 1](#_Toc170947161)2

[**Conclusion:** 12](#_Toc170947165)

1. **INTRODUCTION:**

The evolution of artificial intelligence and digital assistants has inspired developers to build AI-integrated web applications. This internship project involves building a Gemini-inspired AI interface using the MERN stack, focusing on replicating the UI/UX of Gemini and incorporating secure user login using JWT (JSON Web Token) authentication. The platform simulates AI-driven responses in a clean, interactive, and secure web environment.

**2. OBJECTIVES:**

- To design and develop a web application using the MERN stack.

- To replicate Gemini's AI-based interface and simulate interaction.

- To ensure secure user access with JWT-based authentication.

- To implement protected routes and maintain session integrity.

- To understand full-stack deployment and responsive design.

**3. Project Description**

**This Gemini clone allows users to register, authenticate, and interact with a chatbot in a Gemini-inspired interface. Though actual AI models are not integrated, mock or placeholder logic simulates the AI behavior. The app is composed of two major parts:**

- Frontend (React.js + Tailwind CSS): Renders UI components, manages routing, and handles state for user sessions and chat interactions.

- Backend (Node.js + Express): Exposes RESTful APIs for user registration, login, and token verification.

- Database (MongoDB): Stores user information with encrypted passwords and JWT tokens.

**4. Tools Used:**

- Frontend: React.js, Tailwind CSS

- Backend: Node.js, Express.js

- Database: MongoDB Atlas

- Authentication: JWT (jsonwebtoken), bcrypt.js

- Version Control: Git & GitHub

- Deployment: Vercel (Frontend), Render (Backend)

- Testing: Postman

**5. Requirements**

- Node.js (v18 or above)

- MongoDB Atlas account

- React.js environment

- Postman for API testing

**6. Hardware Requirements**

- Minimum 4 GB RAM

- Dual-core processor

- Stable internet connection

**7. Frontend Directory**

/client

├── /src

├── /components

├── /pages

├── /services

├── App.js

**8. Backend Directory**

/server

├── /routes

├── /controllers

├── /models

├── /middlewares

└── server.js

**9. Important Functionality**

- Registration: Securely registers new users with hashed passwords.

- Login: Authenticates users and returns a JWT token.

- Middleware: Verifies JWT to allow access to protected resources.

- Mock AI Handler: Simulates a response to user input.

**10. Source Code**

**Backend..**

const express = require('express');

const axios = require('axios');

const { body, validationResult } = require('express-validator');

const Chat = require('../models/Chat');

const auth = require('../middleware/auth');

const router = express.Router();

// Gemini AI API configuration

const GEMINI\_API\_URL = 'https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent';

const GEMINI\_API\_KEY = process.env.GEMINI\_API\_KEY;

// Validate API key

if (!GEMINI\_API\_KEY || !GEMINI\_API\_KEY.startsWith('AIza')) {

  console.error('Invalid or missing Gemini API key');

  process.exit(1);

}

// Update the text chat route

router.post('/text', [

  auth,

  body('prompt').trim().isLength({ min: 1, max: 2000 }).withMessage('Prompt must be 1-2000 characters')

], async (req, res) => {

  try {

    const errors = validationResult(req);

    if (!errors.isEmpty()) {

      return res.status(400).json({ errors: errors.array() });

    }

    const { prompt } = req.body;

    const payload = {

      contents: [{

        parts: [{

          text: prompt

        }]

      }],

      generationConfig: {

        temperature: 0.7,

        topK: 40,

        topP: 0.95,

        maxOutputTokens: 1024,

      }

    };

    const geminiResponse = await axios.post(GEMINI\_API\_URL, payload, {

      headers: {

        'Content-Type': 'application/json',

        'x-goog-api-key': GEMINI\_API\_KEY

      },

      validateStatus: function (status) {

        return status < 500;

      }

    });

    if (geminiResponse.status !== 200) {

      console.error('Gemini API Error:', {

        status: geminiResponse.status,

        data: geminiResponse.data

      });

      throw new Error(geminiResponse.data.error?.message || 'Failed to get response from Gemini');

    }

    const response = geminiResponse.data.candidates?.[0]?.content?.parts?.[0]?.text;

    if (!response) {

      throw new Error('Invalid response format from Gemini API');

    }

    const chat = new Chat({

      user: req.user.\_id,

      prompt,

      response,

      type: 'text',

      model: 'gemini-pro'

    });

    await chat.save();

    res.json({

      message: 'Text response generated successfully',

      data: {

        id: chat.\_id,

        prompt: chat.prompt,

        response: chat.response,

        type: chat.type,

        createdAt: chat.createdAt

      }

    });

  } catch (error) {

    console.error('Text chat error:', {

      message: error.message,

      status: error.response?.status,

      data: error.response?.data

    });

    let statusCode = 500;

    let errorMessage = 'Error generating text response';

    if (error.response) {

      switch (error.response.status) {

        case 400:

          statusCode = 400;

          errorMessage = 'Invalid request to Gemini API';

          break;

        case 401:

          statusCode = 401;

          errorMessage = 'Invalid API key';

          break;

        case 429:

          statusCode = 429;

          errorMessage = 'Rate limit exceeded';

          break;

      }

    }

    res.status(statusCode).json({

      message: errorMessage,

      error: error.message

    });

  }

});

// Image description route using Gemini

router.post('/image', [

  auth,

  body('prompt').trim().isLength({ min: 1, max: 2000 }).withMessage('Prompt must be 1-2000 characters')

], async (req, res) => {

  try {

    const errors = validationResult(req);

    if (!errors.isEmpty()) {

      return res.status(400).json({ errors: errors.array() });

    }

    const { prompt } = req.body;

    const imagePrompt = `Describe in vivid detail the image I want: "${prompt}". Include specifics like colors, composition, lighting, style, and atmosphere.`;

    const payload = {

      contents: [{ parts: [{ text: imagePrompt }] }]

    };

    const geminiResponse = await axios.post(`${GEMINI\_API\_URL}?key=${GEMINI\_API\_KEY}`, payload, {

      headers: { 'Content-Type': 'application/json' }

    });

    const response = geminiResponse.data.candidates[0]?.content?.parts[0]?.text || 'No response from Gemini';

    const imageUrl = `https://picsum.photos/512/512?random=${Date.now()}`;

    const chat = new Chat({

      user: req.user.\_id,

      prompt,

      response,

      type: 'image',

      imageUrl,

      model: 'gemini-pro'

    });

    await chat.save();

    res.json({

      message: 'Image description generated successfully',

      data: {

        id: chat.\_id,

        prompt: chat.prompt,

        response: chat.response,

        type: chat.type,

        imageUrl: chat.imageUrl,

        createdAt: chat.createdAt

      }

    });

  } catch (error) {

    console.error('Image generation error:', error.message);

    res.status(500).json({

      message: 'Error generating image description',

      error: error.message

    });

  }

});

// Get chat history

router.get('/history', auth, async (req, res) => {

  try {

    const page = parseInt(req.query.page) || 1;

    const limit = parseInt(req.query.limit) || 20;

    const skip = (page - 1) \* limit;

    const chats = await Chat.find({ user: req.user.\_id })

      .sort({ createdAt: -1 })

      .skip(skip)

      .limit(limit)

      .select('-user');

    const total = await Chat.countDocuments({ user: req.user.\_id });

    res.json({

      message: 'Chat history retrieved successfully',

      data: {

        chats,

        pagination: {

          current: page,

          total: Math.ceil(total / limit),

          limit,

          count: total

        }

      }

    });

  } catch (error) {

    console.error('Get history error:', error.message);

    res.status(500).json({ message: 'Error retrieving chat history' });

  }

});

// Delete chat

router.delete('/:id', auth, async (req, res) => {

  try {

    const chat = await Chat.findOneAndDelete({

      \_id: req.params.id,

      user: req.user.\_id

    });

    if (!chat) {

      return res.status(404).json({ message: 'Chat not found' });

    }

    res.json({ message: 'Chat deleted successfully' });

  } catch (error) {

    console.error('Delete chat error:', error.message);

    res.status(500).json({ message: 'Error deleting chat' });

  }

});

module.exports = router;

**FRONTEND**

import React, { useState, useEffect, useRef } from 'react';

import { useAuth } from '../context/AuthContext';

import { toast } from 'react-hot-toast';

import axios from 'axios';

import {

  Send,

  Image,

  MessageCircle,

  LogOut,

  User,

  Trash2,

  Bot,

  RefreshCw

} from 'lucide-react';

function Dashboard() {

  const { user, logout } = useAuth();

  const [messages, setMessages] = useState([]);

  const [inputMessage, setInputMessage] = useState('');

  const [isLoading, setIsLoading] = useState(false);

  const [chatType, setChatType] = useState('text');

  const [history, setHistory] = useState([]);

  const messagesEndRef = useRef(null);

  useEffect(() => {

    fetchChatHistory();

  }, []);

  useEffect(() => {

    scrollToBottom();

  }, [messages]);

  const scrollToBottom = () => {

    messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' });

  };

  const fetchChatHistory = async () => {

    try {

      const response = await axios.get('http://localhost:5000/api/chat/history?limit=10');

      setHistory(response.data.data.chats);

    } catch (error) {

      console.error('Error fetching chat history:', error);

    }

  };

 const handleSendMessage = async () => {

    if (!inputMessage.trim()) return;

    const userMessage = {

      id: Date.now(),

      content: inputMessage,

      type: 'user',

      timestamp: new Date()

    };

    setMessages(prev => [...prev, userMessage]);

    setIsLoading(true);

    try {

      const endpoint = chatType === 'text'

        ? 'http://localhost:5000/api/chat/text'

        : 'http://localhost:5000/api/chat/image';

      // Add request headers and log request data

      const config = {

        headers: {

          'Content-Type': 'application/json',

          'Authorization': `Bearer ${localStorage.getItem('token')}`

        }

      };

      // Log request details

      console.log('Sending request to:', endpoint);

      console.log('Request data:', { prompt: inputMessage });

      const response = await axios.post(endpoint, {

        prompt: inputMessage

      }, config);

      // Log successful response

      console.log('Response received:', response.data);

      const botMessage = {

        id: Date.now() + 1,

        content: response.data.data.response,

        type: 'bot',

        chatType: response.data.data.type,

        imageUrl: response.data.data.imageUrl,

        timestamp: new Date()

      };

      setMessages(prev => [...prev, botMessage]);

      fetchChatHistory();

      toast.success(`${chatType === 'text' ? 'Text' : 'Image'} response generated!`);

    } catch (error) {

      // Enhanced error handling

      console.error('Error details:', {

        message: error.message,

        status: error.response?.status,

        statusText: error.response?.statusText,

        responseData: error.response?.data

      });

      // Show more specific error message to user

      let errorMessage = 'Failed to send message';

      if (error.response) {

        if (error.response.status === 500) {

          errorMessage = 'Server error. Please try again later.';

        } else if (error.response.status === 401) {

          errorMessage = 'Please login again.';

          logout(); // Assuming you have access to the logout function

        } else {

          errorMessage = error.response.data?.message || errorMessage;

        }

      } else if (error.request) {

        errorMessage = 'No response from server. Check your connection.';

      }

      toast.error(errorMessage);

    } finally {

      setIsLoading(false);

      setInputMessage('');

    }

};

  const handleKeyPress = (e) => {

    if (e.key === 'Enter' && !e.shiftKey) {

      e.preventDefault();

      handleSendMessage();

    }

  };

  const clearChat = () => {

    setMessages([]);

    toast.success('Chat cleared');

  };

  const deleteHistoryItem = async (id) => {

    try {

      await axios.delete(`http://localhost:5000/api/chat/${id}`);

      fetchChatHistory();

      toast.success('Chat deleted');

    } catch (error) {

      toast.error('Failed to delete chat');

    }

  };

  const formatTimestamp = (timestamp) => {

    return new Date(timestamp).toLocaleTimeString([], {

      hour: '2-digit',

minute: '2-digit'

    });

  };

  return (

    <div className="min-h-screen bg-gray-100 flex">

      {/\* Sidebar \*/}

      <div className="w-1/4 bg-white shadow-lg">

        <div className="p-4 border-b border-gray-200">

          <div className="flex items-center justify-between">

            <div className="flex items-center space-x-2">

              <User className="w-8 h-8 text-purple-600" />

              <div>

                <h2 className="font-semibold text-gray-800">{user?.username}</h2>

                <p className="text-sm text-gray-600">{user?.email}</p>

              </div>

            </div>

            <button

              onClick={logout}

              className="p-2 text-gray-600 hover:text-red-600 transition-colors"

            >

              <LogOut className="w-5 h-5" />

            </button>

          </div>

        </div>

        <div className="p-4">

          <div className="flex items-center justify-between mb-4">

            <h3 className="font-semibold text-gray-800">Chat History</h3>

            <button

              onClick={fetchChatHistory}

              className="p-1 text-gray-600 hover:text-purple-600 transition-colors"

            >

              <RefreshCw className="w-4 h-4" />

            </button>

          </div>

          <div className="space-y-2 max-h-96 overflow-y-auto">

            {history.map((chat) => (

              <div

                key={chat.\_id}

                className="p-3 bg-gray-50 rounded-lg hover:bg-gray-100 transition-colors group"

              >

                <div className="flex items-start justify-between">

                  <div className="flex-1">

                    <p className="text-sm text-gray-800 truncate">

                      {chat.prompt.substring(0, 50)}...

                    </p>

                    <div className="flex items-center space-x-2 mt-1">

                      <span className={`px-2 py-1 text-xs rounded-full ${

                        chat.type === 'text'

                          ? 'bg-blue-100 text-blue-800'

                          : 'bg-purple-100 text-purple-800'

                      }`}>

                        {chat.type}

                      </span>

                      <span className="text-xs text-gray-500">

                        {formatTimestamp(chat.createdAt)}

                      </span>

                    </div>

                  </div>

                  <button

                    onClick={() => deleteHistoryItem(chat.\_id)}

                    className="opacity-0 group-hover:opacity-100 p-1 text-red-600 hover:text-red-800 transition-all"

                  >

                    <Trash2 className="w-4 h-4" />

                  </button>

                </div>

              </div>

            ))}

            {history.length === 0 && (

              <p className="text-gray-500 text-center py-4">No chat history yet</p>

            )}

          </div>

        </div>

      </div>

      {/\* Main Chat Area \*/}

      <div className="flex-1 flex flex-col">

        {/\* Header \*/}

        <div className="bg-white shadow-sm p-4 border-b border-gray-200">

          <div className="flex items-center justify-between">

            <h1 className="text-2xl font-bold text-gray-800">Gemini ChatBot</h1>

            <div className="flex items-center space-x-4">

              <div className="flex items-center space-x-2">

                <span className="text-sm text-gray-600">Mode:</span>

                <div className="flex bg-gray-100 rounded-lg p-1">

                  <button

                    onClick={() => setChatType('text')}

                    className={`px-3 py-1 rounded-md text-sm font-medium transition-colors ${

                      chatType === 'text'

                        ? 'bg-purple-600 text-white'

                        : 'text-gray-600 hover:text-gray-800'

                    }`}

                  >

                    <MessageCircle className="w-4 h-4 inline mr-1" />

                    Text

                  </button>

                  <button

                    onClick={() => setChatType('image')}

                    className={`px-3 py-1 rounded-md text-sm font-medium transition-colors ${

                      chatType === 'image'

                        ? 'bg-purple-600 text-white'

                        : 'text-gray-600 hover:text-gray-800'

                    }`}

                  >

                    <Image className="w-4 h-4 inline mr-1" />

                    Image

                  </button>

                </div>

              </div>

              <button

                onClick={clearChat}

                className="px-4 py-2 bg-gray-200 text-gray-800 rounded-lg hover:bg-gray-300 transition-colors"

              >

                Clear Chat

              </button>

            </div>

          </div>

        </div>

        {/\* Messages \*/}

        <div className="flex-1 overflow-y-auto p-4 space-y-4">

          {messages.length === 0 ? (

            <div className="text-center text-gray-500 mt-20">

              <Bot className="w-16 h-16 mx-auto mb-4 text-gray-400" />

              <h3 className="text-xl font-semibold mb-2">Welcome to Gemini ChatBot</h3>

              <p>Start a conversation by typing a message below</p>

            </div>

          ) : (

            messages.map((message) => (

              <div

                key={message.id}

                className={`flex ${message.type === 'user' ? 'justify-end' : 'justify-start'}`}

              >

                <div

                  className={`max-w-3xl px-4 py-2 rounded-lg ${

                    message.type === 'user'

                      ? 'bg-purple-600 text-white'

                      : 'bg-white text-gray-800 shadow-sm border border-gray-200'

                  }`}

                >

                  {message.type === 'bot' && (

                    <div className="flex items-center space-x-2 mb-2">

                      <Bot className="w-5 h-5 text-purple-600" />

                      <span className="text-sm font-medium text-purple-600">Gemini</span>

                    </div>

                  <div className="whitespace-pre-wrap">{message.content}</div>

                  {message.imageUrl && (

                    <div className="mt-2">

                      <img

                        src={message.imageUrl}

                        alt="Generated"

                        className="max-w-full rounded-lg shadow-md"

                      />

                    </div>

                  )}

                  <div className={`text-xs mt-1 ${

                    message.type === 'user' ? 'text-purple-200' : 'text-gray-500'

                  }`}>

                    {formatTimestamp(message.timestamp)}

                  </div>

                </div>

              </div>

            ))

          )}

     {isLoading && (

            <div className="flex justify-start">

              <div className="max-w-3xl px-4 py-2 bg-white rounded-lg shadow-sm border border-gray-200">

                <div className="flex items-center space-x-2 mb-2">

                  <Bot className="w-5 h-5 text-purple-600" />

                  <span className="text-sm font-medium text-purple-600">Gemini</span>

                </div>

                <div className="flex items-center space-x-2">

                  <div className="animate-spin rounded-full h-4 w-4 border-b-2 border-purple-600"></div>

                  <span className="text-gray-600">

                    {chatType === 'text' ? 'Thinking...' : 'Generating image description...'}

                  </span>

                </div>

              </div>

            </div>

          )}

          <div ref={messagesEndRef} />

        </div>

        <div className="bg-white border-t border-gray-200 p-4">

          <div className="flex items-end space-x-2">

            <div className="flex-1">

              <textarea

                value={inputMessage}

                onChange={(e) => setInputMessage(e.target.value)}

                onKeyPress={handleKeyPress}

                placeholder={

                  chatType === 'text'

                    ? 'Ask Gemini anything...'

                    : 'Describe the image you want to generate...'

                }

                className="w-full px-4 py-2 border border-gray-300 rounded-lg focus:ring-2 focus:ring-purple-500 focus:border-transparent resize-none"

                rows="3"

                disabled={isLoading}

              />

            </div>

            <button

              onClick={handleSendMessage}

              disabled={isLoading || !inputMessage.trim()}

              className="px-6 py-2 bg-purple-600 text-white rounded-lg hover:bg-purple-700 focus:outline-none focus:ring-2 focus:ring-purple-500 focus:ring-offset-2 disabled:opacity-50 disabled:cursor-not-allowed transition-colors flex items-center space-x-3 >

              <Send className="w-5 h-5" />

              <span>Send</span>

            </button>

          </div>

          <div className="mt-2 text-xs text-gray-500 text-center">

            {chatType === 'text' ? 'Text chat mode' : 'Image description mode'} •

            Press Enter to send, Shift+Enter for new line

          </div>

        </div>

      </div>

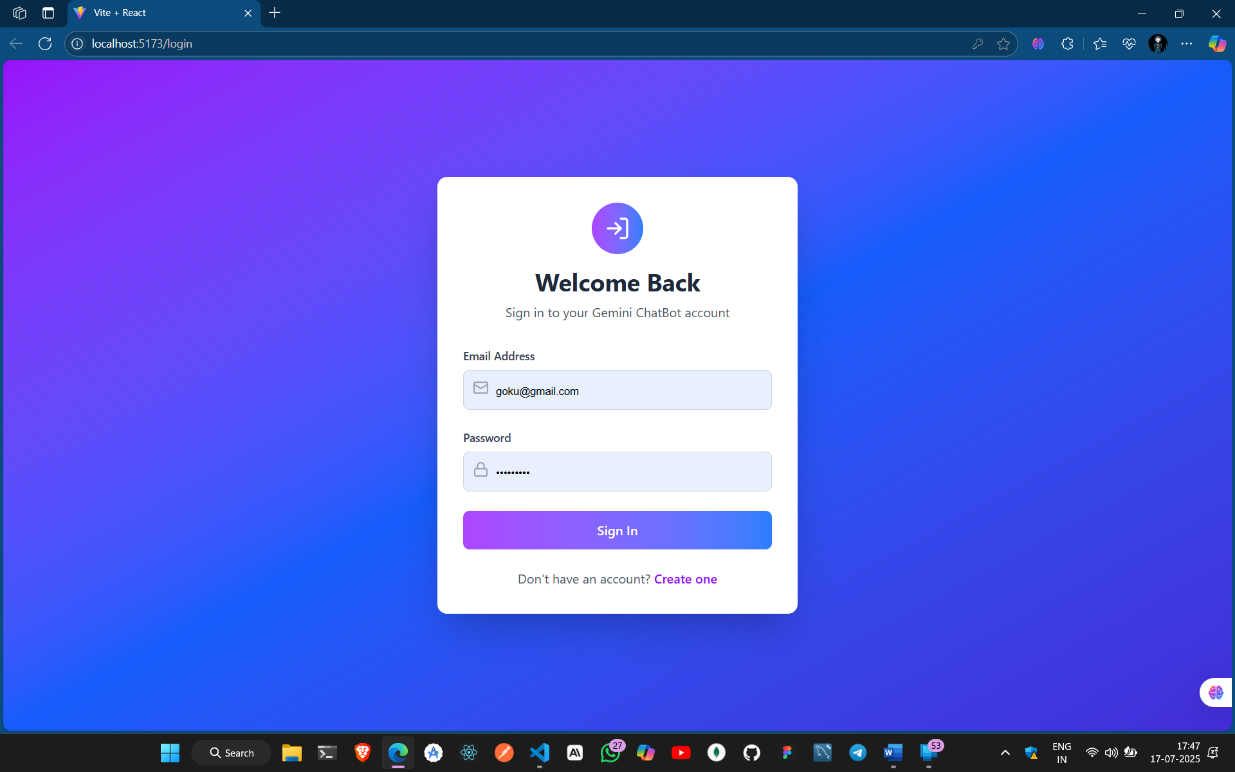
    </div>

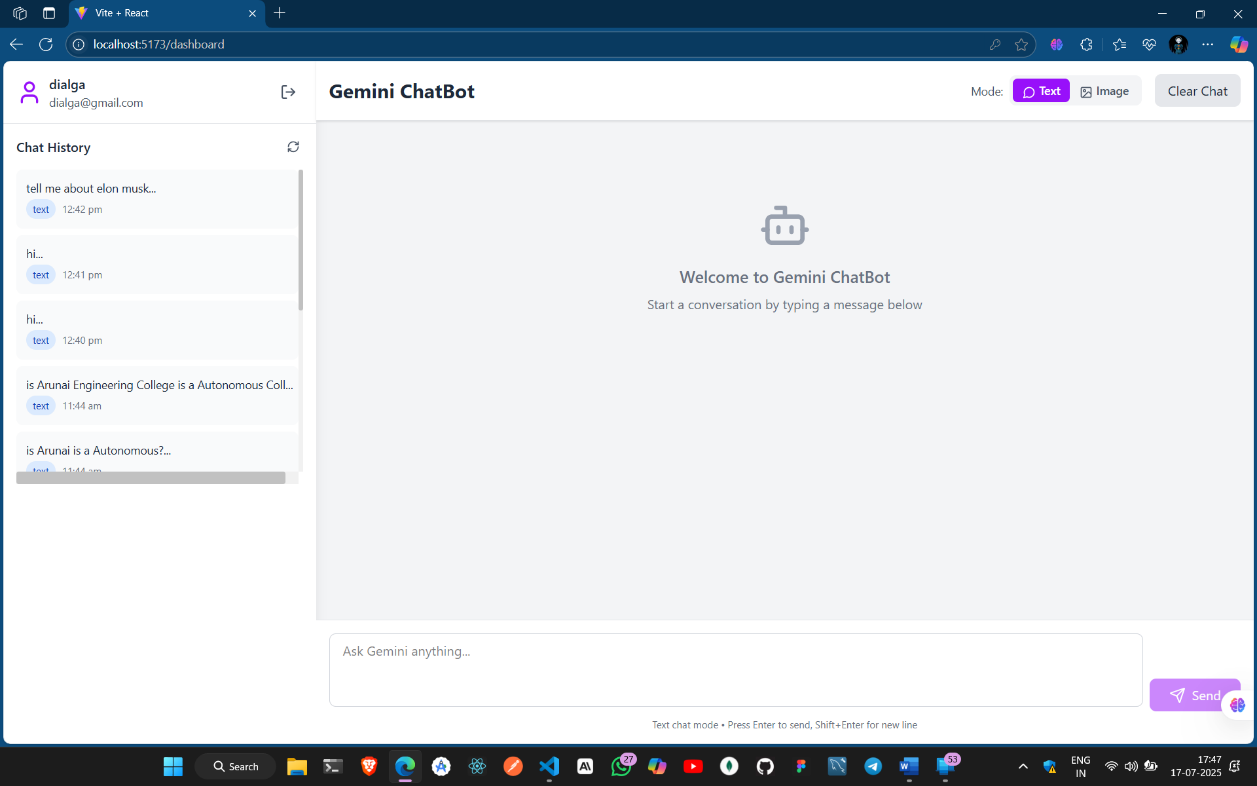
  );

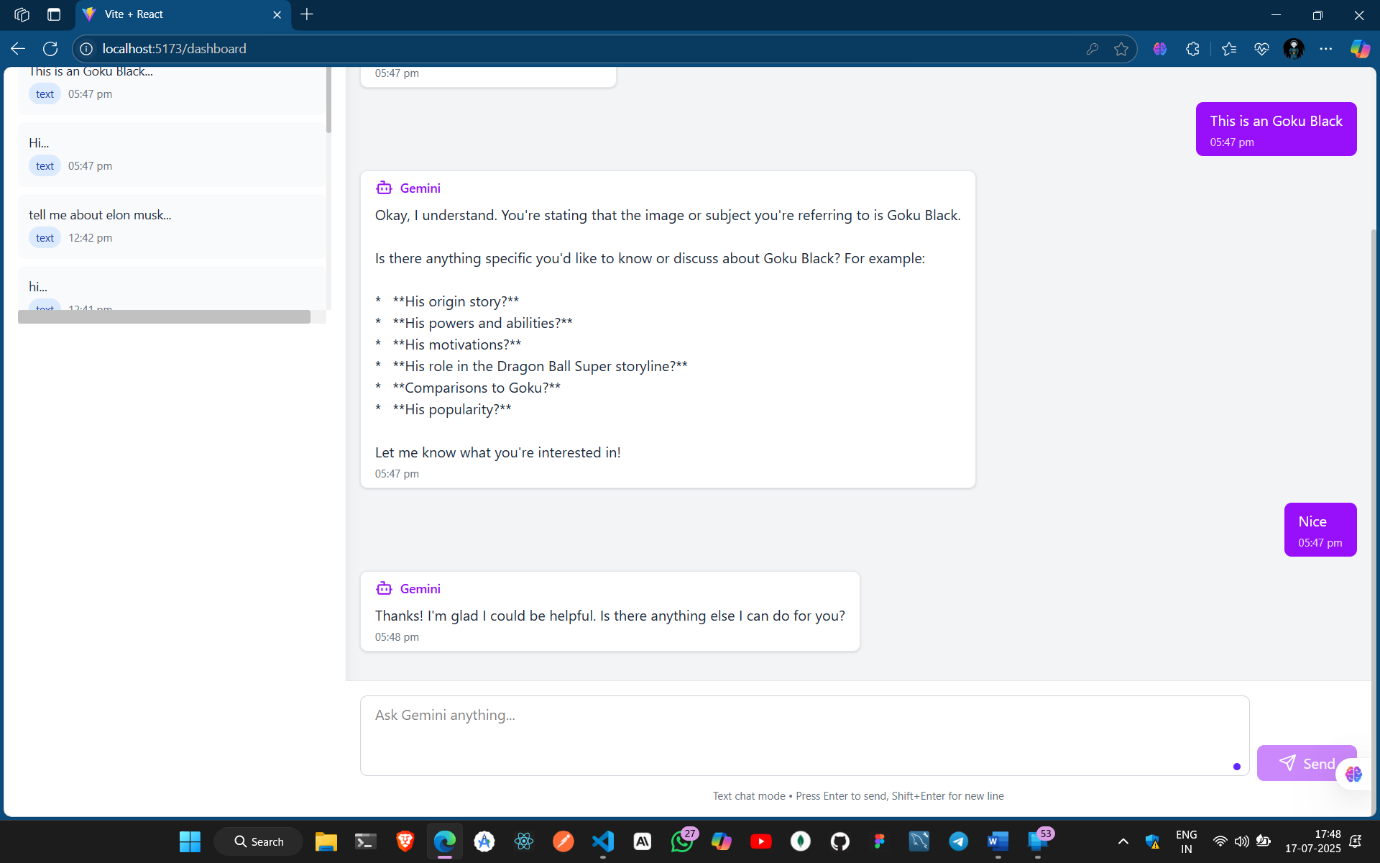
}

export default Dashboard;

**10. OUTPUT**

****

****



**11. Conclusion**

This internship project provided an opportunity to work on full-stack development using modern web technologies. By building a Gemini-like application from scratch, the project reinforced core concepts of component-based design, REST API handling, user authentication, and deployment. This clone not only demonstrates technical implementation but also opens doors to future integration with actual AI services like Google Gemini or OpenAI APIs.