

## **Project Description**

QuickGPT is a MERN stack based AI chatbot web application inspired by ChatGPT. It allows users to chat with an AI model, generate images, and explore community-shared posts. The system includes a credit-based model in which users must purchase credits to use AI features. The project also provides stripe-based payment integration, JWT authentication and clean, responsive frontend UI.

## **Project Objective**

- To develop a functional AI chatbot, using MERN stack
- To allow users to generate text and images using AI
- To implement a credit-based usage system
- To integrate secure payments using stripe
- To provide a clean and responsive UI using modern web technologies

## **Technology Stack**

### ➤ **Frontend**

- React.js
- Tailwind CSS

### ➤ **Backend**

- Node.js
- Express.js
- MongoDB + mongoose

### ➤ **Authentication**

- JSON web token

### ➤ **Payments**

- Stripe API

### ➤ **AI Features**

- Google Gemini API

### ➤ **Third-party API**

- Official Joke API

## Links

➤ **GitHub repository link**

- [https://github.com/AP-anjali/QuickGPT\\_CC\\_PROJECT](https://github.com/AP-anjali/QuickGPT_CC_PROJECT)

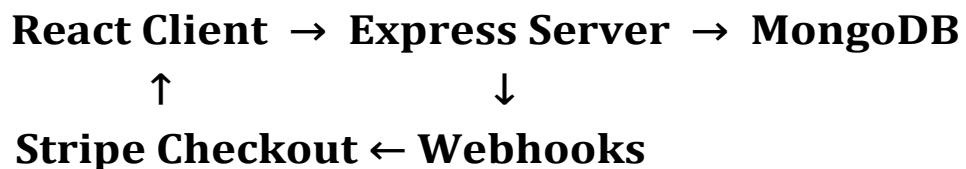
➤ **Deployed website link**

- **Frontend :** <https://202412063-quick-gpt.vercel.app/>
- **Backend :** <https://quick-gpt-server-plum.vercel.app/>

## Contribution

<b><u>Student ID</u></b>	<b><u>Name</u></b>	<b><u>Contribution</u></b>
202412063	Anjali Patel	Entire Backend and testing
202412065	Dhara Patel	Entire Frontend
202412088	Hirwa Saraiya	Payment integration
202412121	Viraj Sharma	Third-party integration

## **System Architecture [as a development view]**



### ➤ **Flow Summary :**

- User interacts with frontend
- API calls sent to backend
- Backend verifies token and then processes request if valid token
- AI answer generation and credits deducted
- Payments handled via stripe checkout
- MongoDB stores users, chats, messages, credits & transactions

## Architecture Diagram [as a deployment view]

### ➤ Step 1 : define the business problem, that needs to be solved

- QuickGPT must allow end users to interact with an AI chat/image-generation service and a community posts area while enforcing a credit-based pay-to-use model, providing a secure, responsive UI and safe, scalable backend handling of AI API calls, payments(stripe), authentication(JWT), and community content.
- Primary success criteria :
  - Low-latency chat and image generation for paying users
  - Secure payments and credit accounting (no free bypass)
  - Clear separation of real-time request handling VS background work
  - Observability and fault-tolerance

### ➤ Step 2 : define the necessary business process, that needs to be automated by the software application

- Sub-function 1 : Interact with the customer (presentation) : user interface
- Sub-function 2 : Provide application logic to automate business process
  - Customer logic
  - Fulfillment logic
- Sub-function 3 : Share and manipulate data
  - Customer information
  - Chat data

➤ **Step 3 : translate the business process into the requirements (functions and features) for the software**

○ **Functional :**

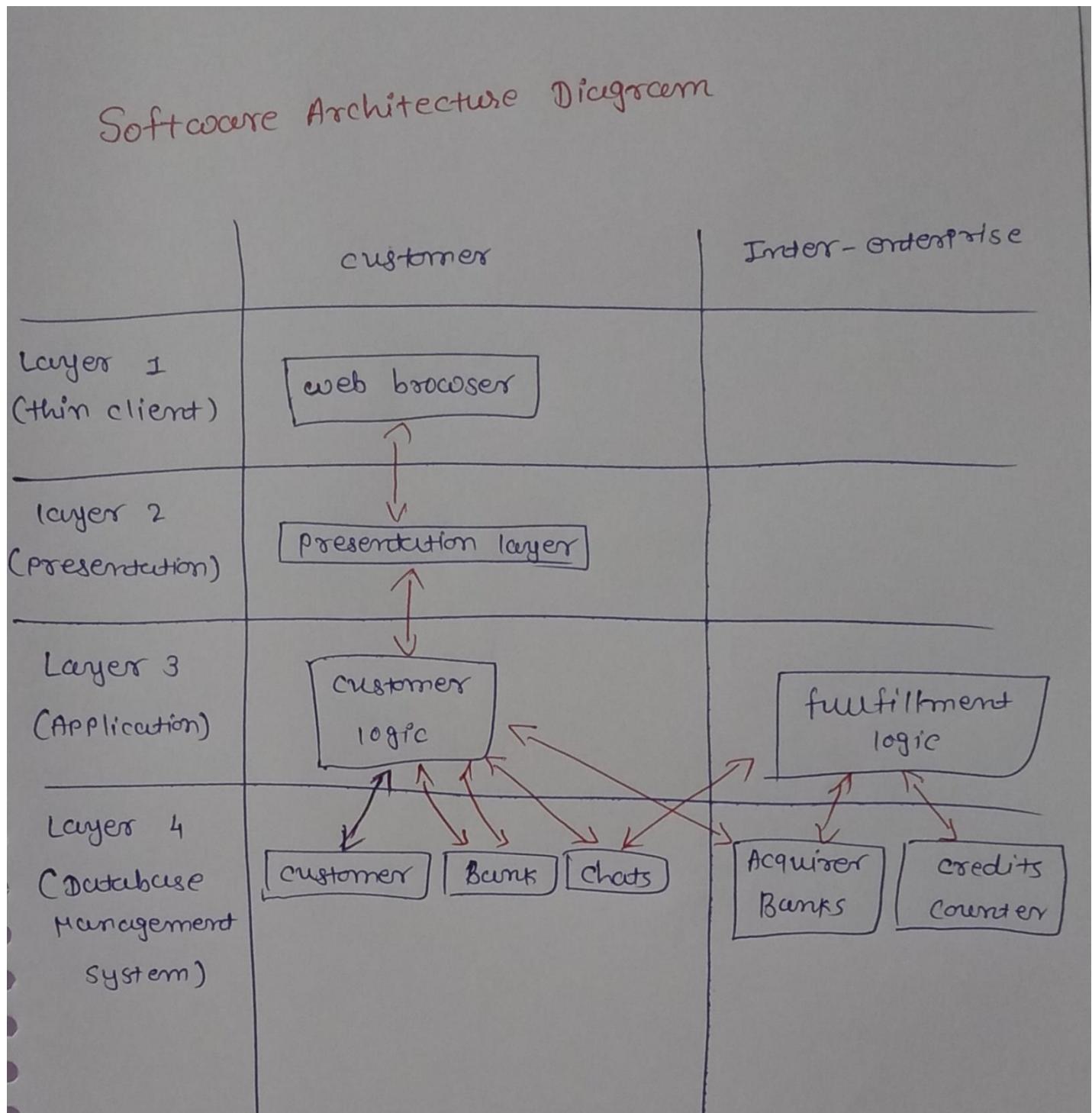
- JWT authentication
- Stripe integration
- Credit accounting microservice
- DB storage

○ **Non-Functional :**

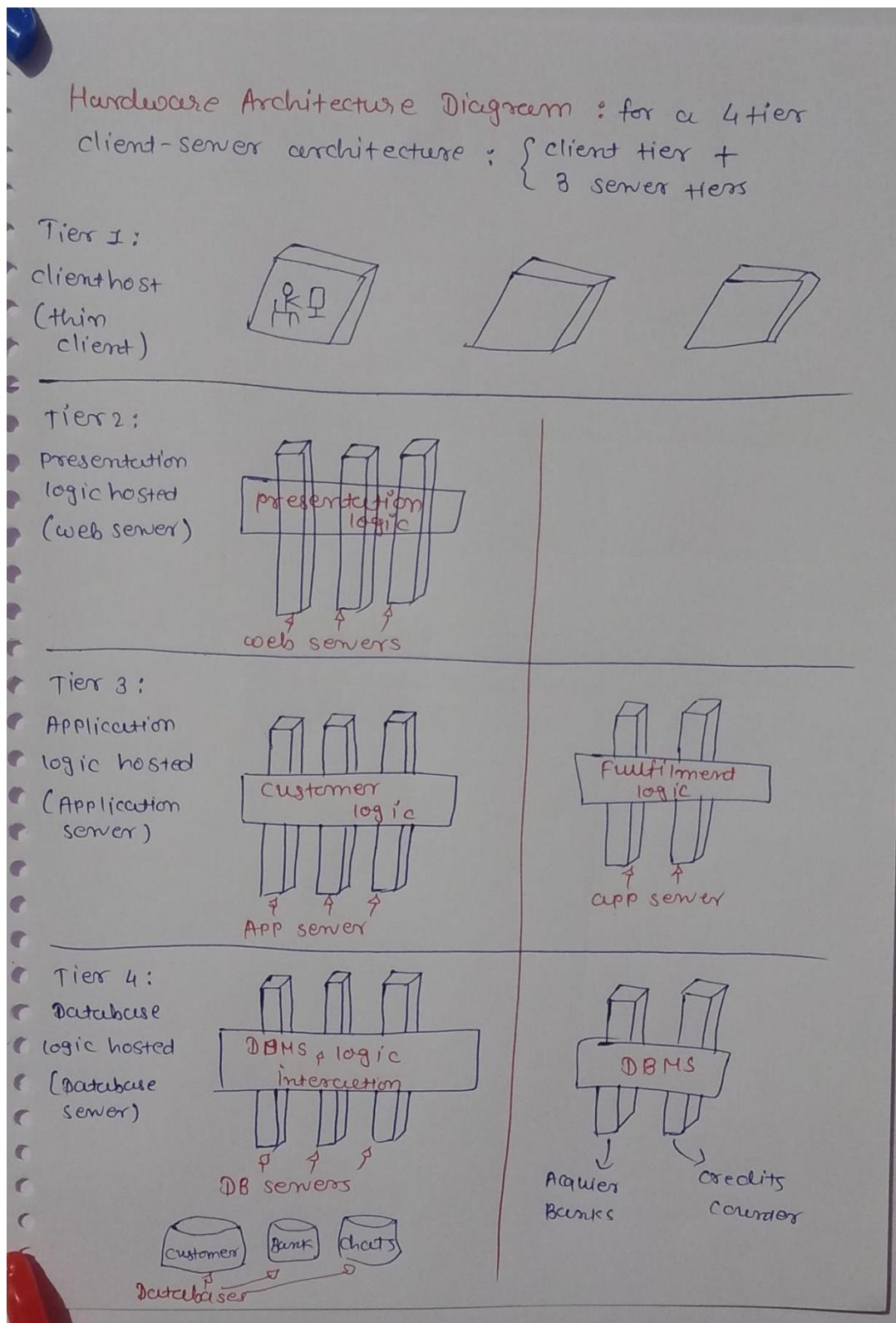
- Horizontal scalability, auto scaling
- High availability for critical services

➤ **Step 4 : define the software architecture diagram : we have the following sub-functions (from step 2)**

- Layer 1 : **thin client** : enable the graphical user interface for the client on client's customer
- Layer 2 : **to realize sub-function 1** : present the necessary information (chat and published images, ...), create the GUI for the client, and enable the visualization, data entry, communication with the application and output of results
- Layer 3 : **to realize sub-function 2** : perform the application logic
  - Customer logic and fulfillment logic
- Layer 4 : **to realize sub-function 3** : manipulate and share data



## ➤ Step 5 : define the hardware architecture diagram



## **Main Functionality**

### ➤ **User authentication :**

- User can register with name, email, password
- Password is hashed using bcrypt
- JWT token generated during login
- Authenticated routes are protected using middleware

### ➤ **Chat system :**

- Users can create chats
- Chats include messages between user and AI
- Chat name can be edited
- Chat can be deleted anytime
- Messages are stored in mongoDB

### ➤ **AI Text generation :**

- Users can send prompts
- System calls AI API
- Returns response
- 1 credit deducted for each request

### ➤ **AI Image generation :**

- Users can generate images using AI
- 2 credits per image generation
- Generated images can be published in community gallery

➤ **Credit purchase system :**

- Users can buy credits via stripe
- Plans include predefined credit amounts
- Stripe checkout session is created and user is redirected
- Successful payments are confirmed via webhook
- Corresponding credits added into user's account

➤ **Community Page :**

- All users can view AI-generated community images
- Each post shows image + user details + prompt

➤ **Third-Party API integration for random jokes :**

- A random joke is fetched from Official joke API
- Showcased in demo pages

## API Documentation [overview]

### ▪ Auth

<u>Method</u>	<u>Endpoint</u>	<u>Description</u>
POST	/api/users/register	Register user
POST	/api/users/login	Login user
GET	/api/users/me	Get logged in user

### ▪ Chat

<u>Method</u>	<u>Endpoint</u>	<u>Description</u>
GET	/api/chats	List all chats
POST	/api/chats	Create chat
GET	/api/chats/:id	Get chat
PUT	/api/chats/:id	Update chat
DELETE	/api/chats/:id	Delete chat

### ▪ AI answer generation

<u>Method</u>	<u>Endpoint</u>	<u>Description</u>
POST	/api/messages/text	Text generation
POST	/api/messages/image	Image generation

## ▪ Credits

<u>Method</u>	<u>Endpoint</u>	<u>Description</u>
GET	/api/credits/plans	Plans list
POST	/api/credits/purchase	Stripe session
GET	/api/credits/me	User credits
GET	/api/credits/transactions	Transactions
PATCH	/api/credits/transactions/:id	Update

## ▪ Stripe

<u>Method</u>	<u>Endpoint</u>	<u>Description</u>
POST	/api/stripe/webhook	Raw webhook endpoint

## ▪ Third-Party API

<u>Method</u>	<u>Endpoint</u>	<u>Description</u>
GET	/api/jokes/random	Generate random joke

## System Snapshots

## [01] running backend

The screenshot shows a Microsoft Visual Studio Code interface. The title bar displays "node - server". The top navigation bar includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is currently selected), and PORTS. The main area contains a terminal window with the following content:

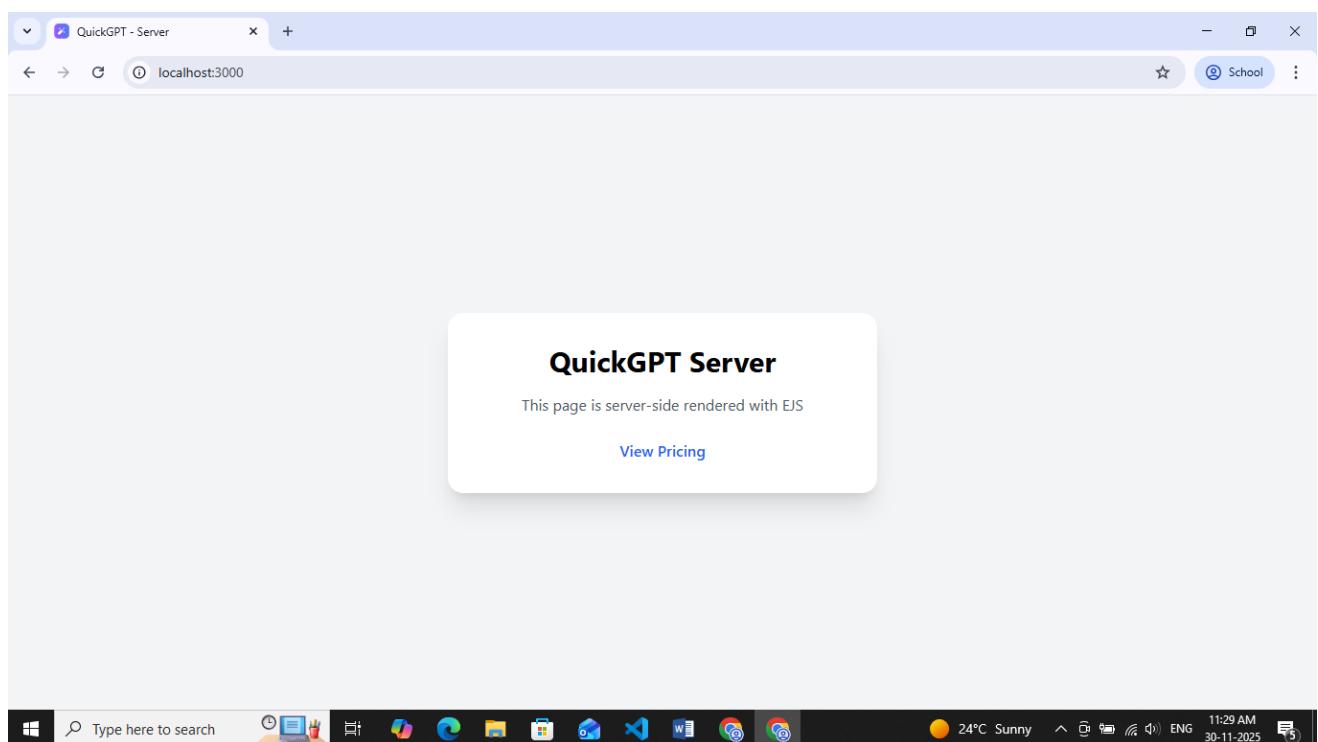
```
● PS F:\QuickGPT_WMD_PROJECT> cd server
○ PS F:\QuickGPT_WMD_PROJECT\server> npm run dev

> server@1.0.0 dev
> nodemon server.js

[nodemon] 3.1.11
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
database connected..
server running on port 3000
```

The status bar at the bottom shows system icons for battery, signal, and network, along with the text "CODEGPT" and the date/time "30-11-2025 11:28 AM".

## [02] server-side rendering using EJS template engine



## [03] server-side rendering pricing page

Pricing Plans

These plans are server-side rendered using EJS

Basic	Pro	Premium
₹0	₹199	₹499
For beginners	For regular users	Unlimited usage
10 prompts/day	100 prompts/day	Unlimited prompts
Basic speed	Fast speed	Ultra fast
Email support	Priority support	24/7 support

Back to Home

## [04] logs

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS F:\QuickGPT_MDN_PROJECT> cd server
PS F:\QuickGPT_MDN_PROJECT\server> npm run dev
> server@1.0.0 dev
> nodemon server.js

[nodemon] 3.1.11
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
database connected...
server running on port 3000
GET / 200 39.574 ms - 874
[2025-11-30T05:59:05.092Z] GET / 200 - 49ms
GET /pricing 200 3.873 ms - 3222
[2025-11-30T05:59:16.272Z] GET /pricing 200 - 6ms
GET /favicon.ico 404 1.777 ms - 1008
[2025-11-30T05:59:16.505Z] GET /favicon.ico 404 - 3ms
GET / 304 2.317 ms -
[2025-11-30T05:59:28.757Z] GET / 304 - 3ms

```

## [05] running frontend

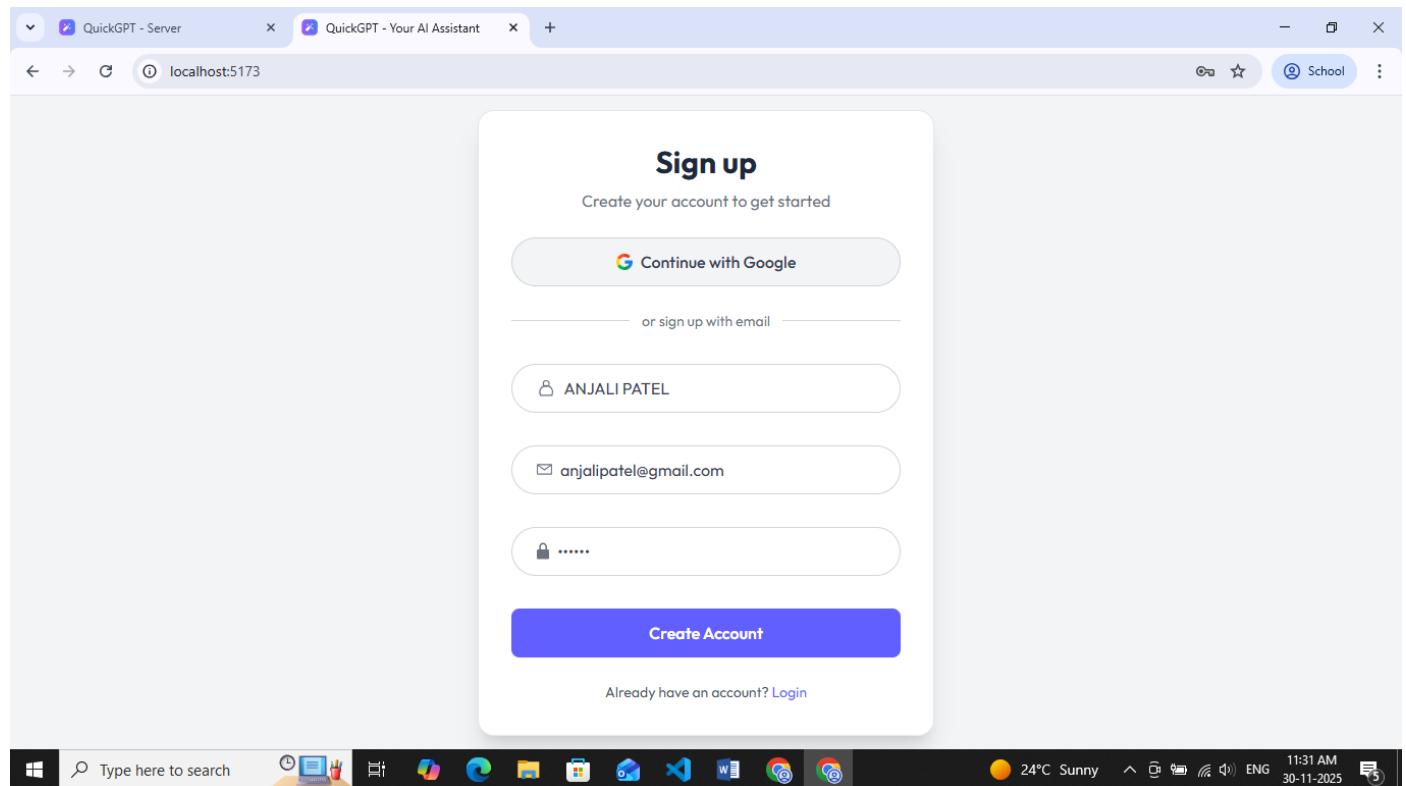
A screenshot of the Visual Studio Code interface. The terminal tab is active, showing the command line output of running the frontend development server. The output shows the project directory, the command run, and the Vite server starting up.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS F:\QuickGPT_WMD_PROJECT> cd client
PS F:\QuickGPT_WMD_PROJECT\client> npm run dev

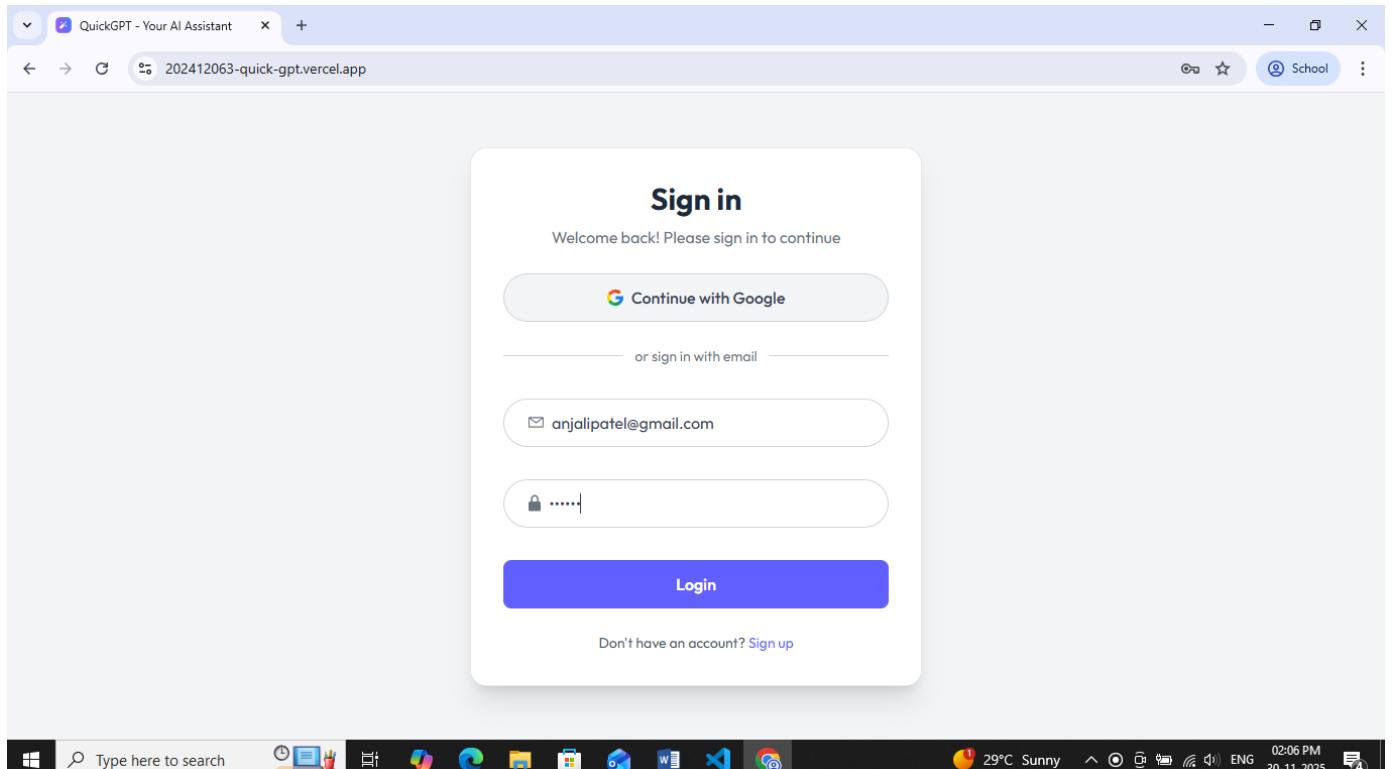
> client@0.0.0 dev
> vite

VITE v7.1.3 ready in 5199 ms
→ Local: http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
```

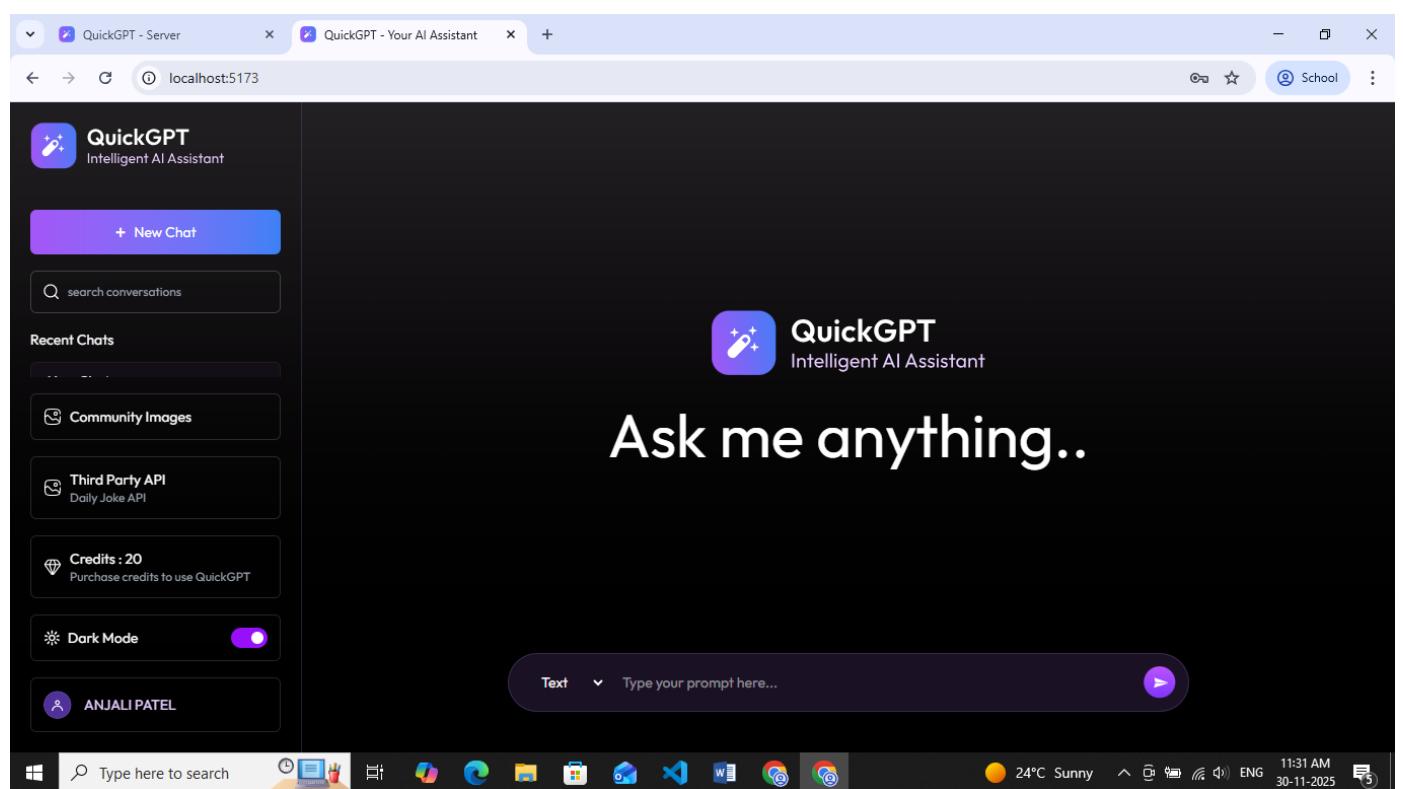
## [06] registration page



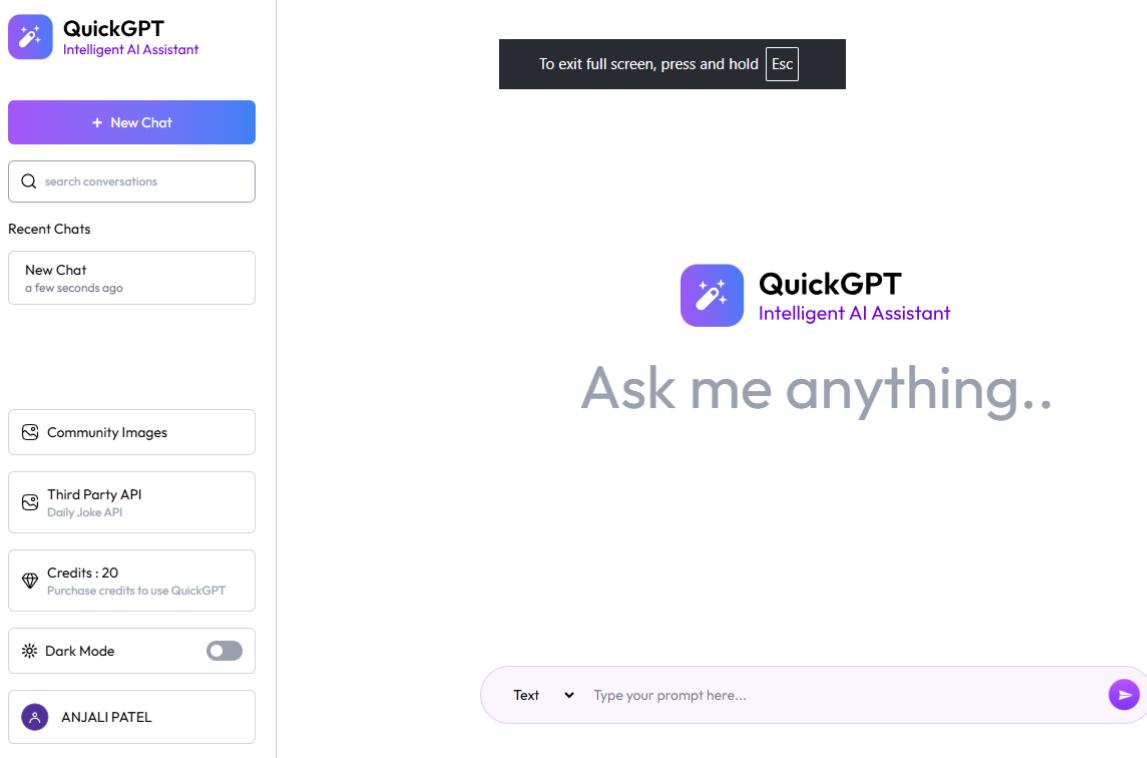
## [07] login page



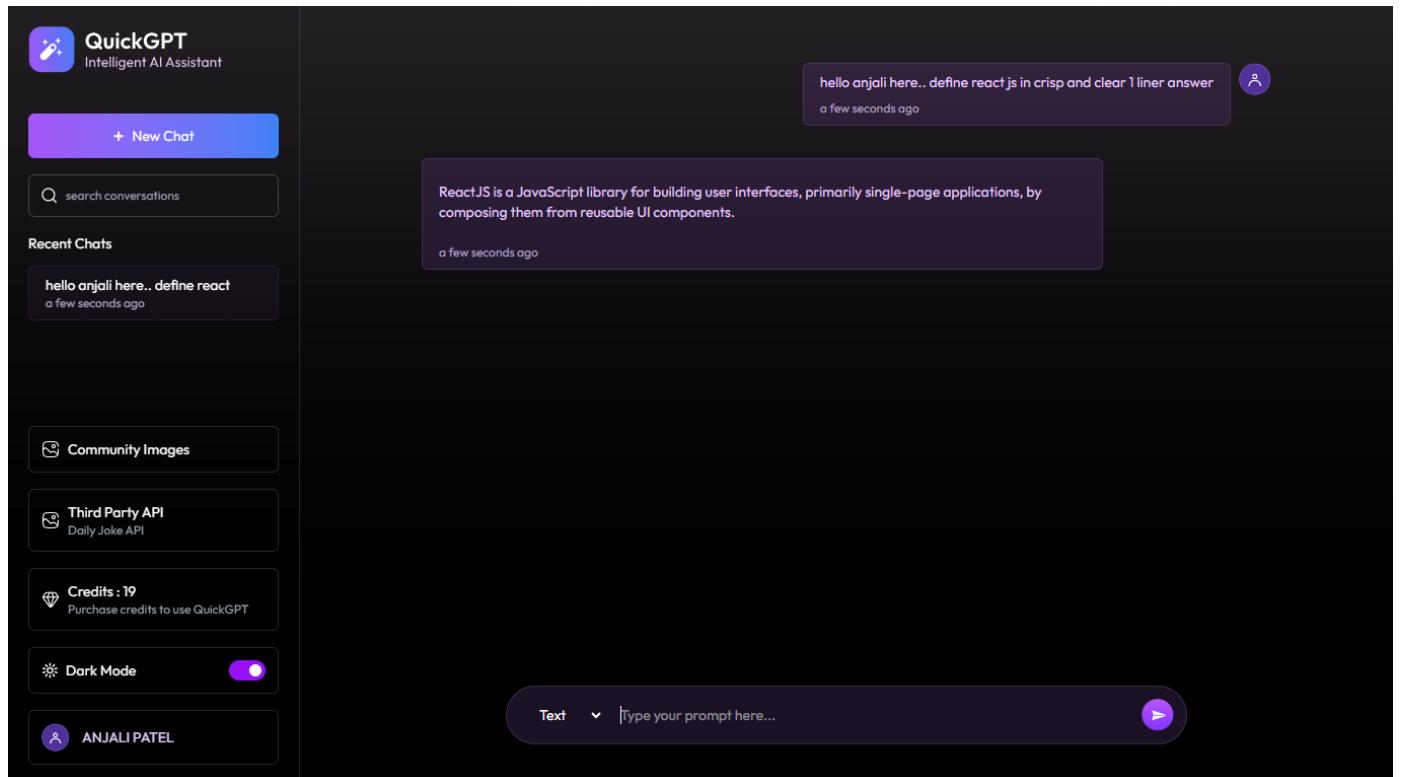
## [08] home page



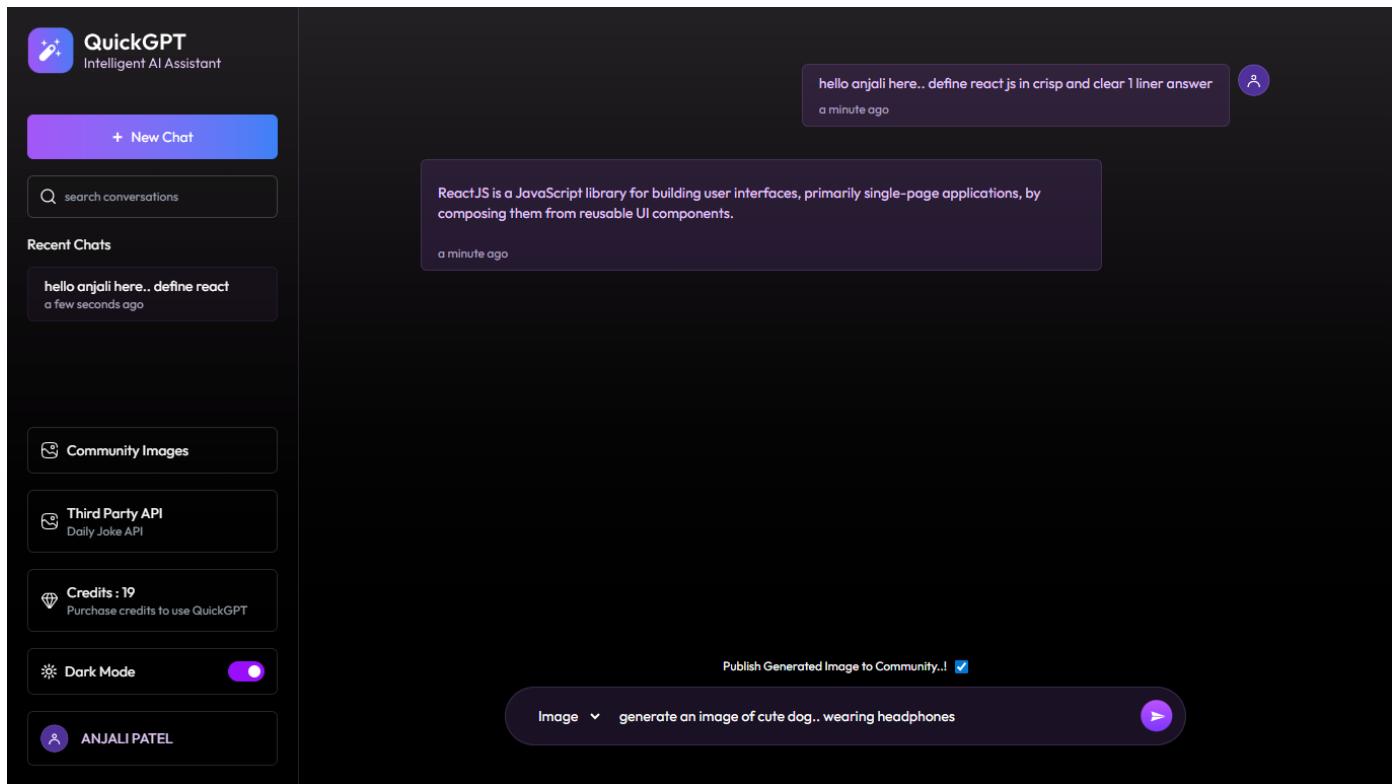
## [09] home page in light mode



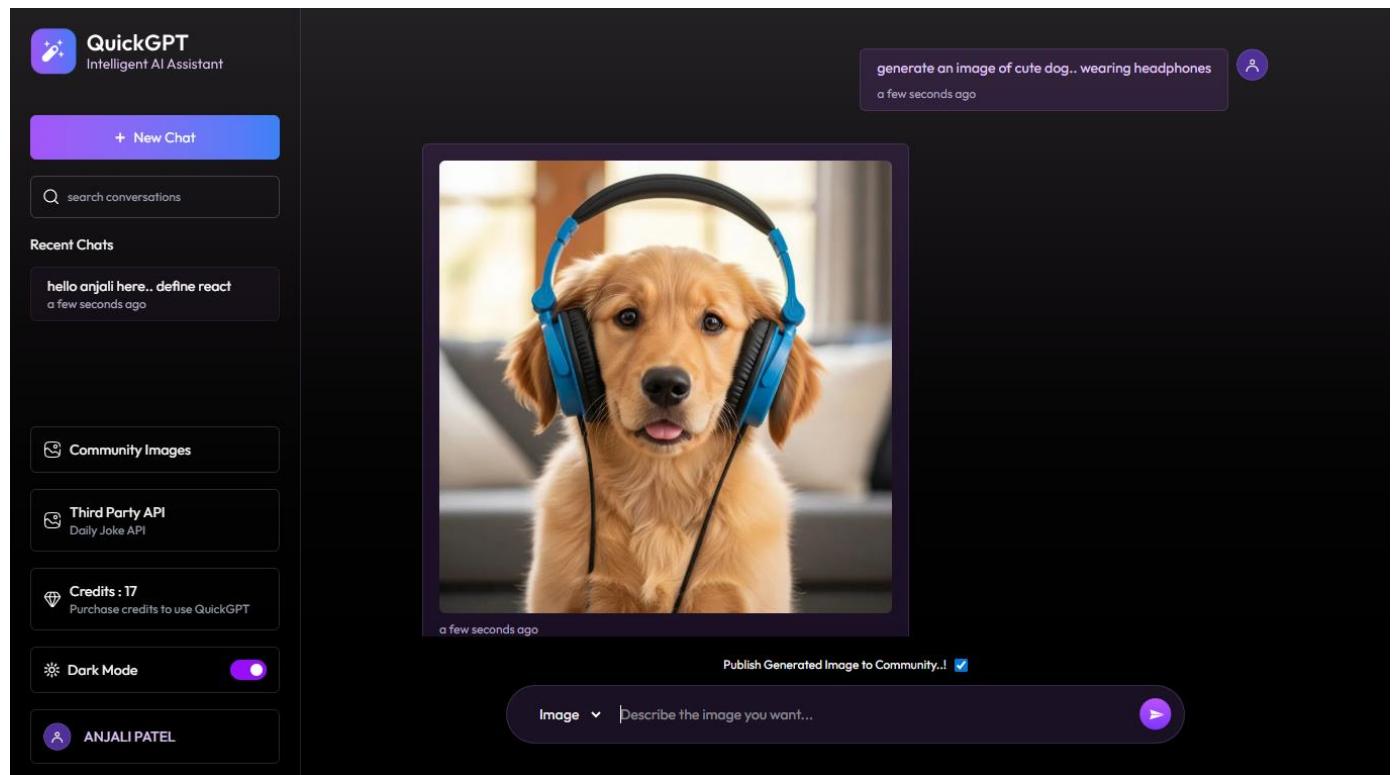
## [10] text message prompt



## [11] image generation prompt and adding image to community



## [12] image generated



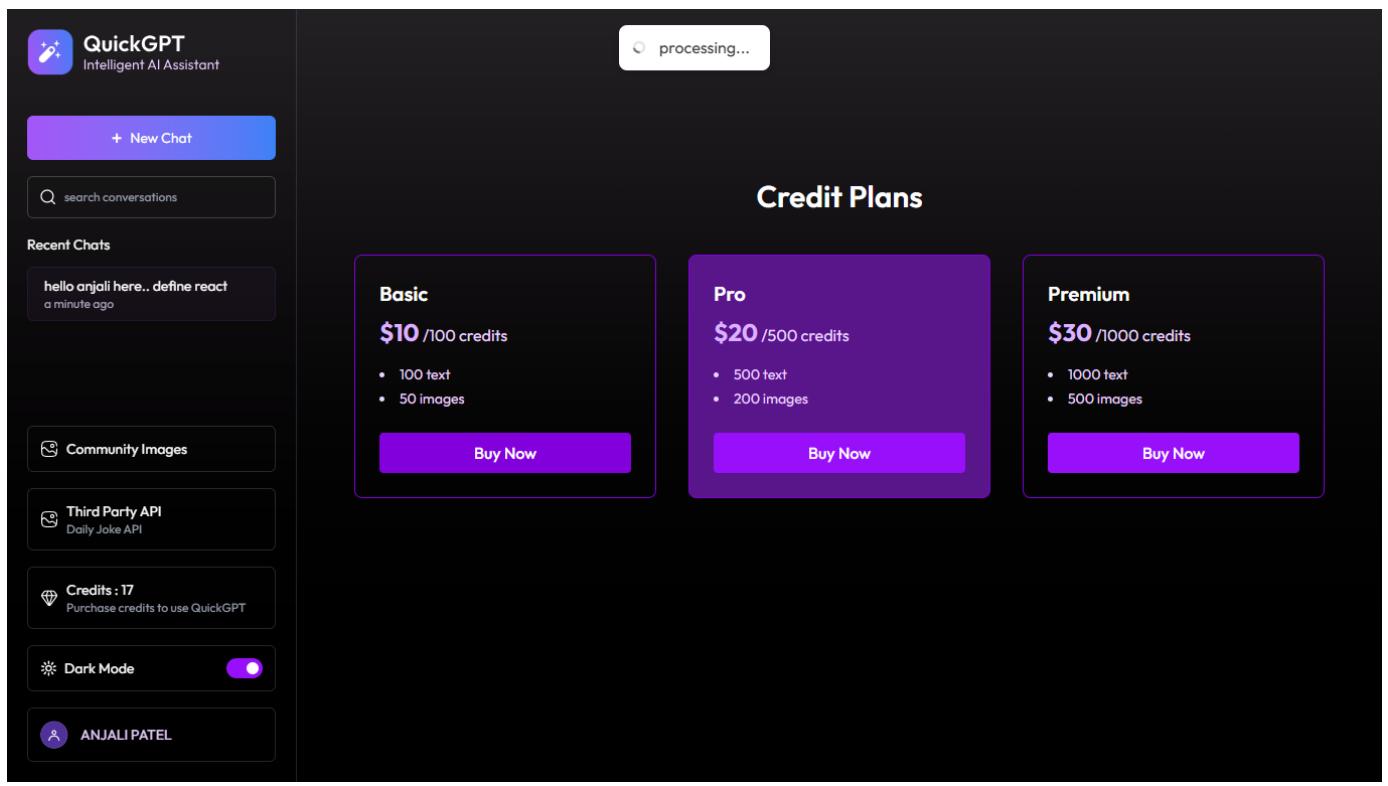
## [13] community images

The screenshot shows the QuickGPT application interface. On the left sidebar, there are several buttons: '+ New Chat', 'search conversations', 'Recent Chats' (with a message 'hello anjali here.. define react'), 'Community Images', 'Third Party API', 'Credits : 17' (Purchase credits to use QuickGPT), 'Dark Mode' (switch is on), and 'ANJALI PATEL'. The main content area is titled 'Community Images' and shows two images: one of a dog wearing headphones and another of a variety of Indian food items.

## [14] pre-defined credit plans

The screenshot shows the QuickGPT application interface. On the left sidebar, there are several buttons: '+ New Chat', 'search conversations', 'Recent Chats' (with a message 'hello anjali here.. define react'), 'Community Images', 'Third Party API', 'Credits : 17' (Purchase credits to use QuickGPT), 'Dark Mode' (switch is on), and 'ANJALI PATEL'. The main content area is titled 'Credit Plans' and displays three plans: 'Basic' (\$10 /100 credits, includes 100 text and 50 images), 'Pro' (\$20 /500 credits, includes 500 text and 200 images), and 'Premium' (\$30 /1000 credits, includes 1000 text and 500 images). Each plan has a 'Buy Now' button.

## [15] initiate payment



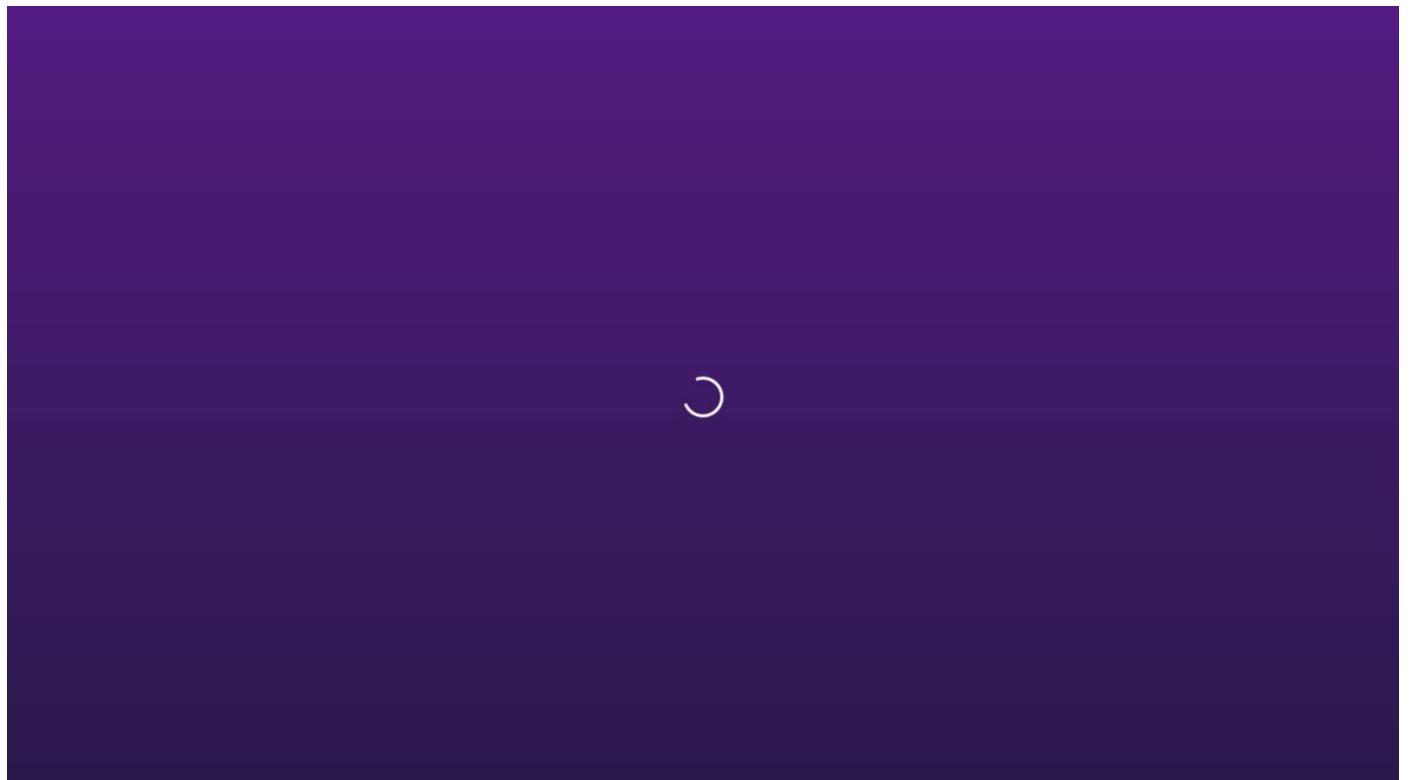
## [16] stripe payment page with dummy details

The screenshot shows a Stripe payment page. At the top, there's a 'TEST MODE' button. Below it, a currency selector shows '₹929.27' (India Rupees) and '\$10.00' (USD). It also notes '1 USD = 92.9270 INR (includes 4% conversion fee)'. The payment amount is listed as '₹929.27'. The right side of the page contains a 'Pay with link' button, an 'OR' option for 'Email' (anjalipatel3074@gmail.com), and a 'Payment method' section. This section includes fields for 'Card information' (card number 4111 1111 1111 1111, expiration 12/28, CVC 123), 'Cardholder name' (ANJALI PATEL), 'Country or region' (India), and a checkbox for 'Save my information for faster checkout' (unchecked). At the bottom is a large blue 'Pay' button.

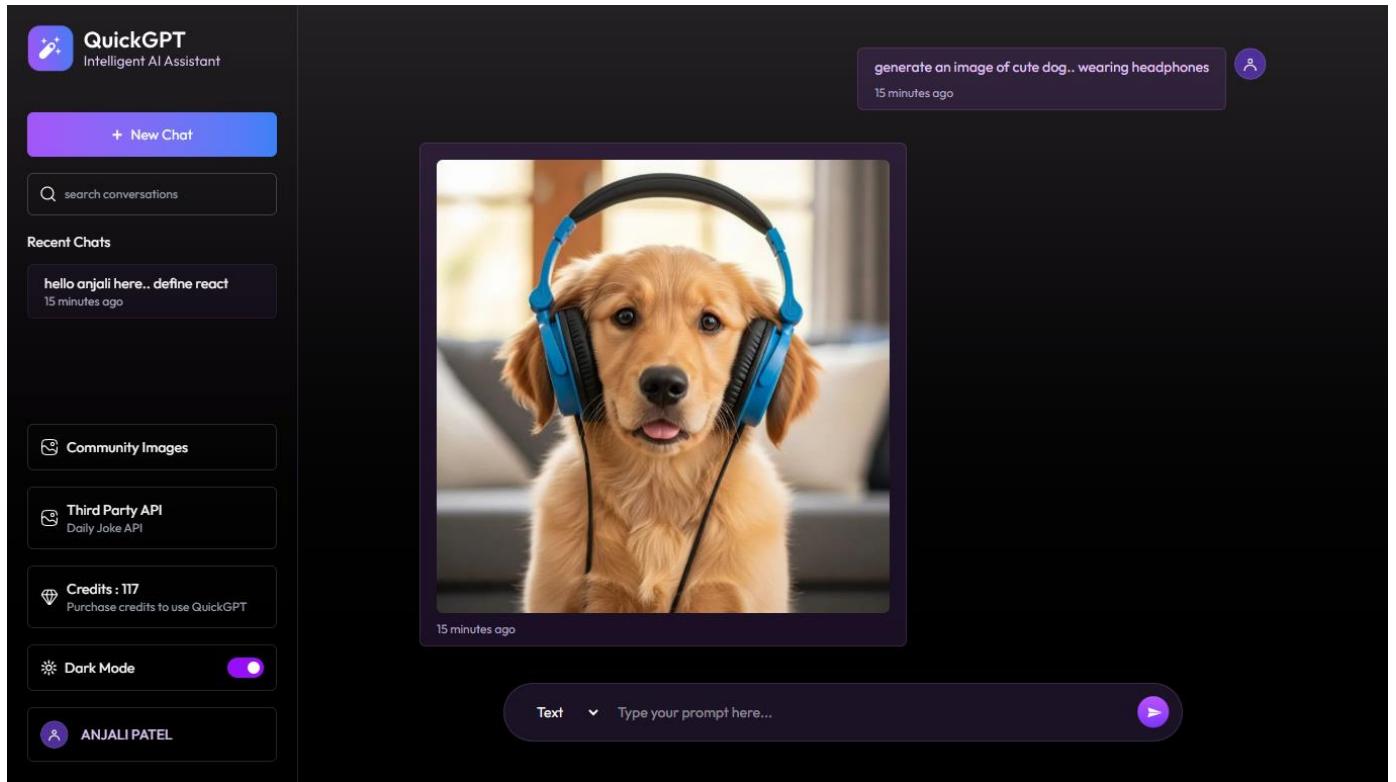
## [17] payment successful

The screenshot shows a payment gateway interface. At the top left is a logo with a gear icon and the text "TEST MODE". Below it, there's a section for "Choose a currency" with two options: ₹929.27 (selected) and \$10.00. A note below says "1 USD = 92.9270 INR (includes 4% conversion fee)". To the right, there's a green button labeled "Pay with link" and an "OR" option. Below these are fields for "Email" (anjalipatel3074@gmail.com) and "Payment method". Under "Card information", there's a placeholder card number (4111 1111 1111 1111), an expiration date (12 / 28), and a CVC code (123). The cardholder name is listed as ANJALI PATEL. Below that, the country or region is set to India. A "Save my information for faster checkout" button is present, along with a note: "Pay securely on this site and everywhere Link is accepted." At the bottom, a blue "Processing" button with a circular loading icon is shown, followed by links to "Powered by stripe", "Terms", and "Privacy".

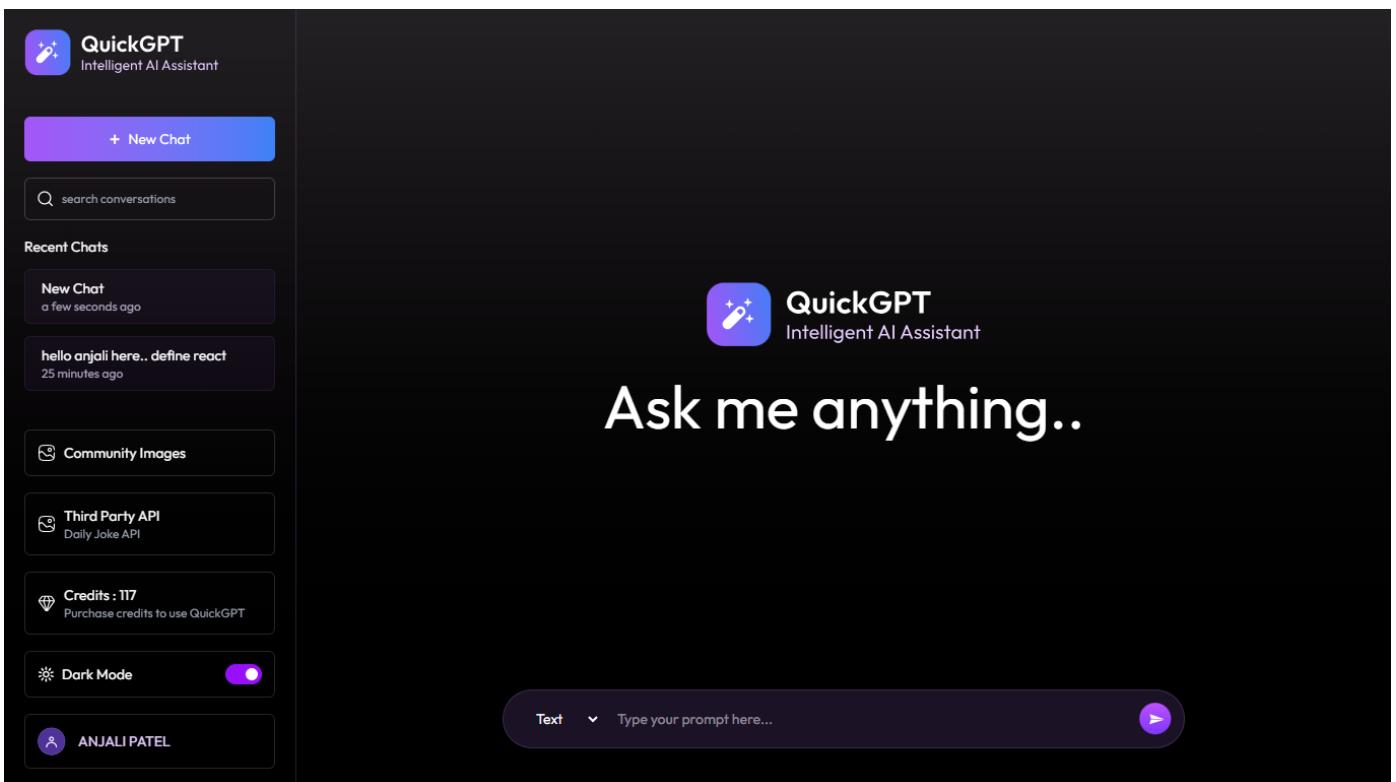
## [18] redirection



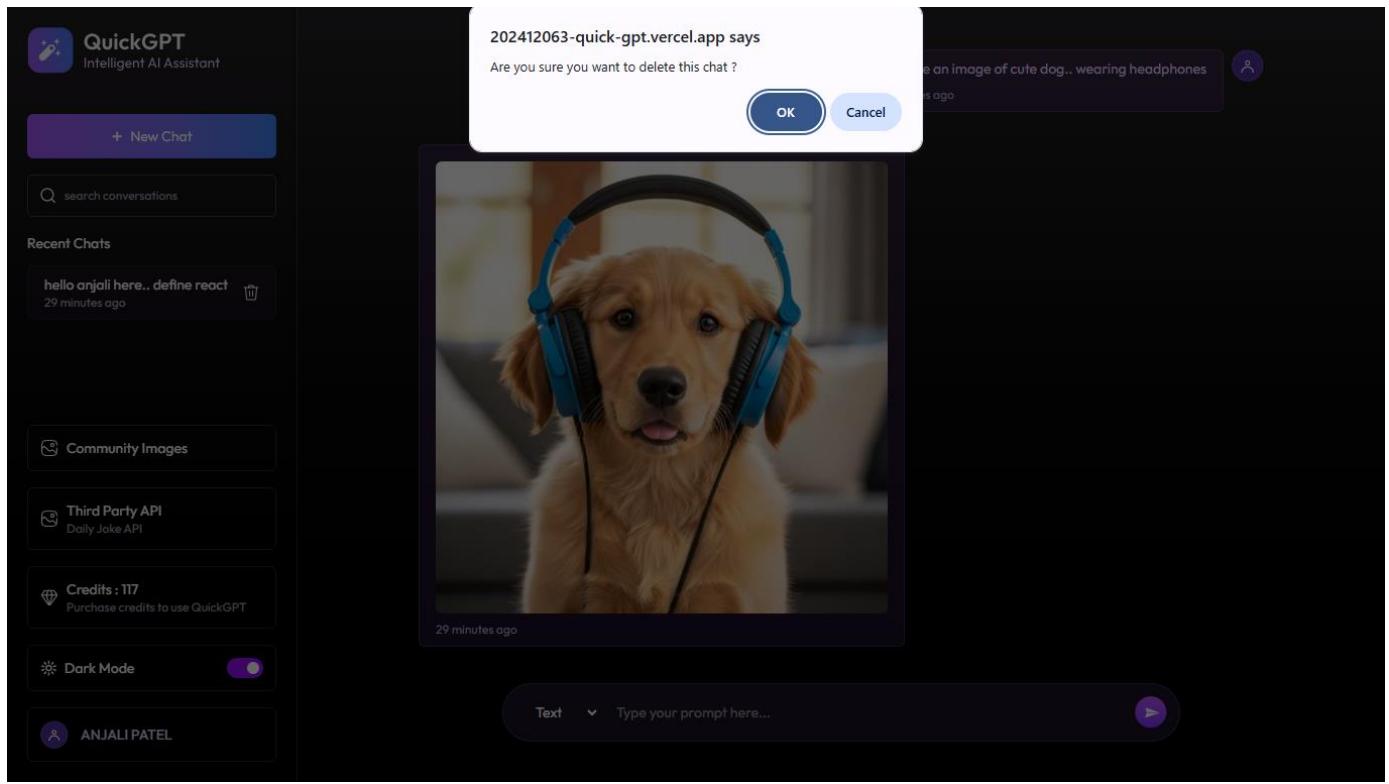
## [19] updated credits



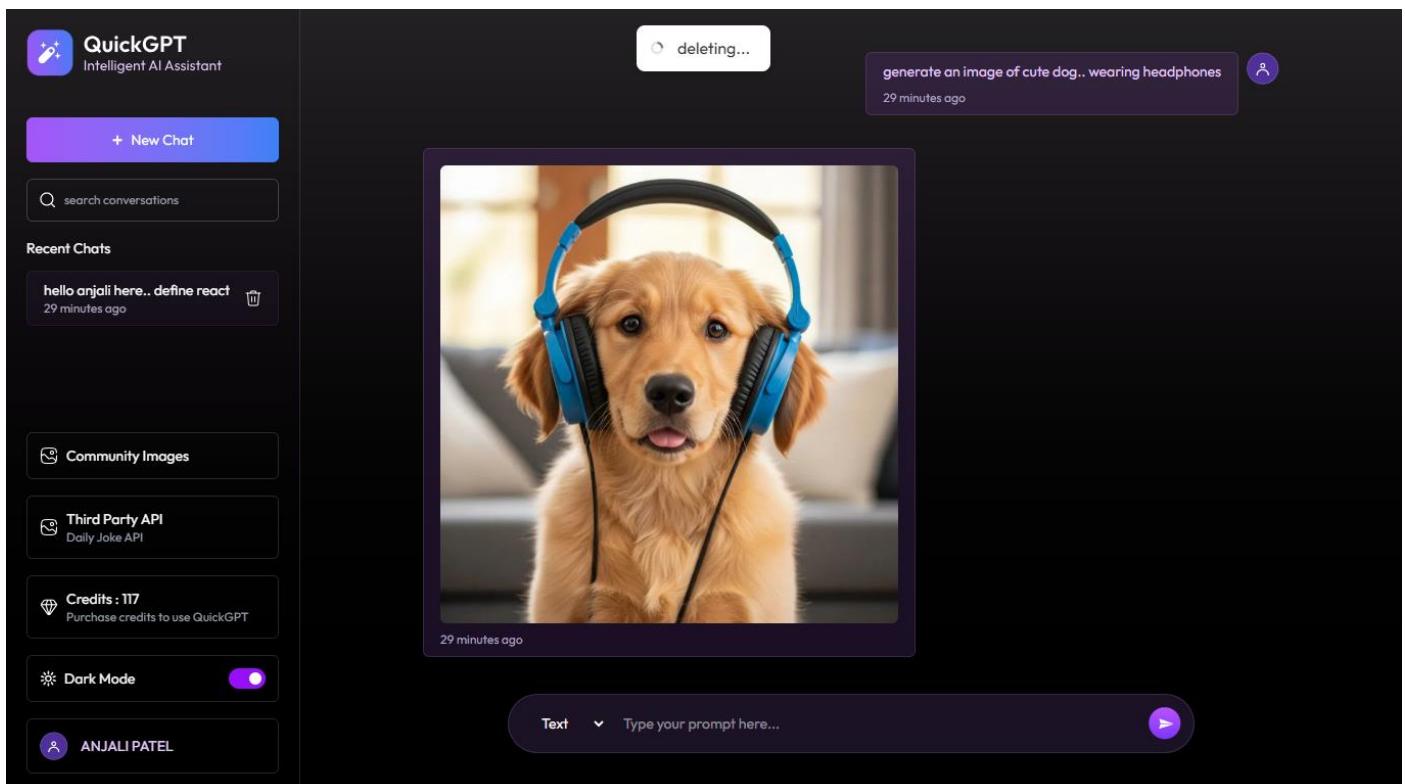
## [20] new chat creation



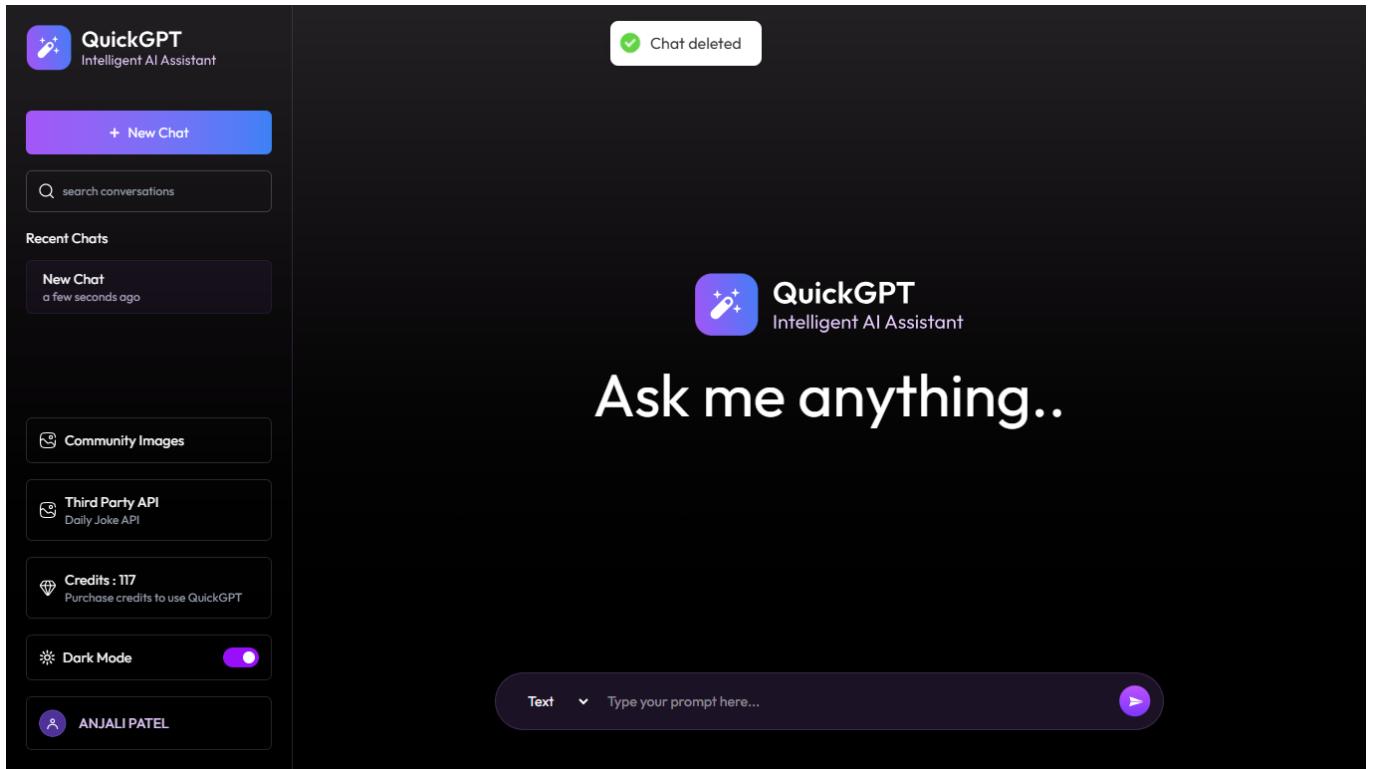
## [21] delete chat



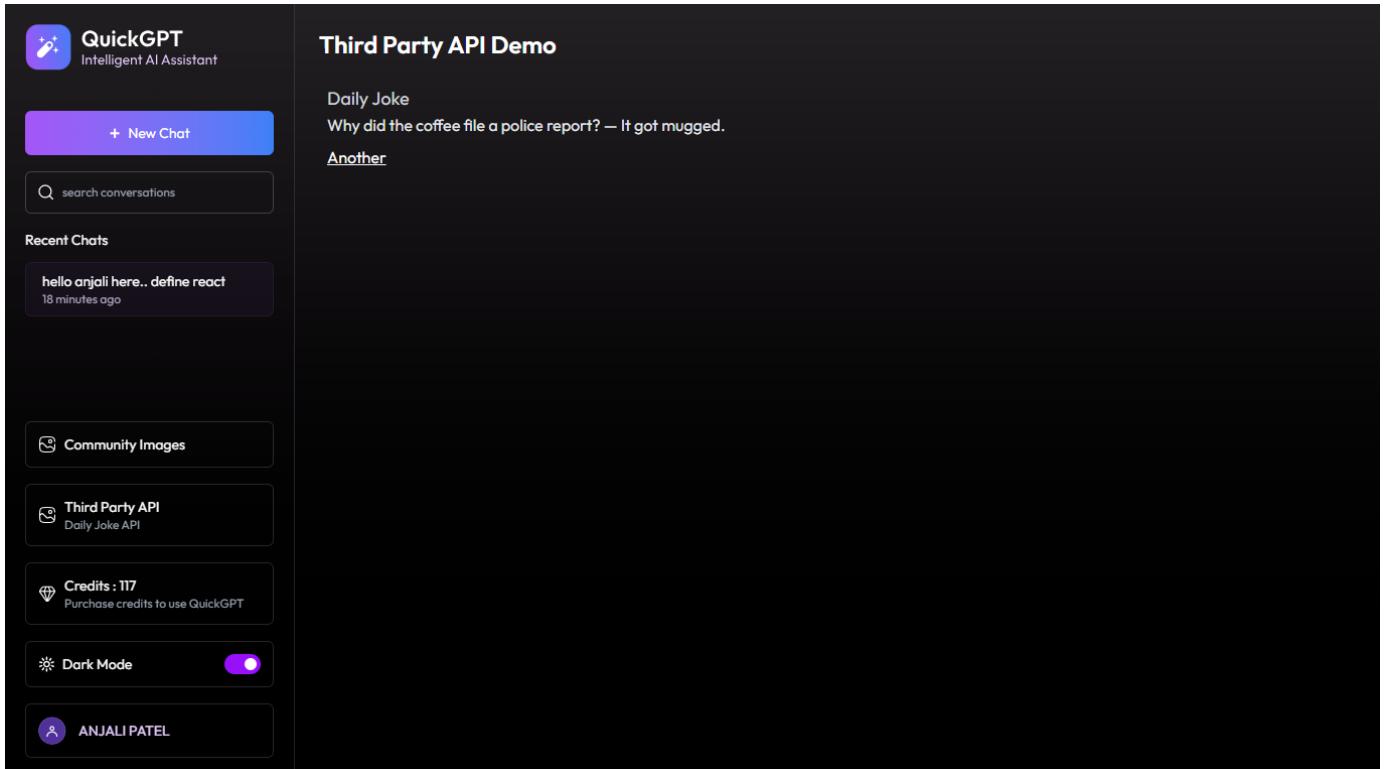
## [22] deleting



## [23] chat deleted



## [24] third-party API



## [25] random joke generation

The screenshot shows the QuickGPT AI chatbot interface. On the left sidebar, there's a logo for "QuickGPT Intelligent AI Assistant" and a purple button labeled "+ New Chat". Below that is a search bar with the placeholder "search conversations". Under "Recent Chats", there's a message from "ANJALI PATEL" that says "hello anjali here.. define react" and was sent "18 minutes ago". Further down the sidebar are buttons for "Community Images", "Third Party API Daily Joke API", "Credits : 117 Purchase credits to use QuickGPT", "Dark Mode" (with a toggle switch), and the user's profile "ANJALI PATEL". The main content area has a title "Third Party API Demo" and a section titled "Daily Joke" with the text "How many seconds are in a year? – 12. January 2nd, February 2nd, March 2nd, April 2nd.... etc". There's also a link "Another" below it.

## [26] logout

The screenshot shows the QuickGPT login page. At the top, a green success message box says "Logged out successfully!". The main form is titled "Sign in" and includes the sub-instruction "Welcome back! Please sign in to continue". It features a "Continue with Google" button and a "or sign in with email" link. Below this are fields for "Email id" and "Password", both with their respective icons. A large blue "Login" button is at the bottom of the form. At the very bottom, there's a link "Don't have an account? Sign up".

## Testing Snapshots

### [01] register user API

The screenshot shows the Postman interface with the 'POST Register User' request selected. The 'Body' tab contains the following JSON payload:

```

1 {
2   "name": "Anjali",
3   "email": "anjali@example.com",
4   "password": "123456"
5 }

```

The 'Test Results' section shows a successful response with status code 201 Created, duration 319 ms, and size 572 B. The response body is:

```

1 {
2   "success": true,
3   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpZC16IjY5MmE4ND0jYmVmzRiyMzNzLzMeWnyIsImlhCI6MTc2NDM5NDayOCwiZXhwIjoxNzY2OTg2MDI4fQ.
aSSNlW0j4puuf77i0LvKj0uASXQrOdo1Y_wpgdtpNGA",
4   "user": {
5     "_id": "692a842cbbeb34bbf376efa07",
6     "name": "Anjali",
7     "email": "anjali@example.com",
8     "credits": 20
9   }
10 }

```

### [02] login user API

The screenshot shows the Postman interface with the 'POST Login User' request selected. The 'Body' tab contains the following JSON payload:

```

1 {
2   "email": "anjali@example.com",
3   "password": "123456"
4 }

```

The 'Test Results' section shows a successful response with status code 200 OK, duration 230 ms, and size 466 B. The response body is:

```

1 {
2   "success": true,
3   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpZC16IjY5MmE4ND0jYmVmzRiyMzNzLzMeWnyIsImlhCI6MTc2NDM5NDayOCwiZXhwIjoxNzY2OTg2MDQ0fQ.
f2FQ2Rwx0Xsw6JxIRlI6qEkMRMu3FFhuozrVBY3Iugs"
4 }

```

## [03] adding token to environment variable

The screenshot shows the Postman interface with the 'Environments' tab selected. A modal window titled 'QuickGPT Local Environment' displays a table of environment variables. One variable, 'token', is highlighted and expanded to show its value as a long, encoded string. The 'What's changed for variables' section includes notes about local values, shared values, and sensitive fields.

Variable	Value
port	3000
baseURL	http://localhost:3000
token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9eyJpZCI6IjY5MmE4NDJjYmVlM2RiYmYzNzZlZmEwNyIsImh0IEdMTc2NDM5NDA0NCwiZXhwIjoxNzY2OTg2MDQ0fQ.f2FQ2RwxbXsw6JxIRl6qEkMRMu3FFhrudrVBY3lugs
chatId	
userId	
transactionId	
Add variable	

## [04] get logged-in user API

The screenshot shows the Postman interface with the 'GET Get Logged-in User' request selected. The 'Headers' tab is active, showing an 'Authorization' header with the value 'Bearer {{token}}'. The 'Body' tab shows a JSON response with the following content:

```

1  {
2     "success": true,
3     "user": {
4         "_id": "692a842cbeb34bbf376efa07",
5         "name": "Anjali",
6         "email": "anjali@example.com",
7         "credits": 20,
8         "__v": 0
9     }
10 }

```

## [05] create chat API

POST Create Chat

HTTP QuickGPT APIs / CHATS / Create Chat

POST {{baseUrl}} /api/chats

Key	Value	Description	Bulk Edit	Presets
Authorization	Bearer {{token}}			
Content-Type	application/json			

```

1 {
2   "success": true,
3   "chat": {
4     "userId": "692a842cbeb34bbf376efa07",
5     "userName": "Anjali",
6     "name": "New Chat",
7     "messages": [],
8     "_id": "692a84a3beb34bbf376efa0c",
9     "createdAt": "2025-11-29T05:29:07.883Z",
10    "updatedAt": "2025-11-29T05:29:07.883Z",
11    "__v": 0
12  }
13}

```

## [06] adding chatID-to environment variable

POST Create Chat

QuickGPT Local Environment

Variable	Value
port	3000
baseUrl	http://localhost:3000
token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZC16IjY5MmE4NDJjYmVmZiRyMjZ...
chatId	692a84a3beb34bbf376efa0c
userId	
transactionId	

## [07] get all chats API

The screenshot shows the Postman interface for the 'Get All Chats' API endpoint. The left sidebar shows the 'QuickGPT\_WMD\_PROJECT' workspace with the 'CHATS' collection selected. The 'GET Get All Chats' request is highlighted. The 'Headers' tab is active, showing an 'Authorization' header with the value 'Bearer {{token}}'. The 'Body' tab shows a JSON response with a single chat object:

```

1 {
2   "success": true,
3   "chats": [
4     {
5       "_id": "692a84a3beb34bbf376efa0c",
6       "userId": "692a842cbeb34bbf376efa07",
7       "userName": "Anjali",
8       "name": "New Chat",
9       "messages": [],
10      "createdAt": "2025-11-29T05:29:07.883Z",
11      "updatedAt": "2025-11-29T05:29:07.883Z",
12      "_v": 0
13    }
14  ]

```

## [08] get chat by ID

The screenshot shows the Postman interface for the 'Get Chat By ID' API endpoint. The left sidebar shows the 'QuickGPT\_WMD\_PROJECT' workspace with the 'CHATS' collection selected. The 'GET Get Chat By ID' request is highlighted. The 'Headers' tab is active, showing an 'Authorization' header with the value 'Bearer {{token}}'. The 'Body' tab shows a JSON response with a single chat object:

```

1 {
2   "success": true,
3   "chat": {
4     "_id": "692a84a3beb34bbf376efa0c",
5     "userId": "692a842cbeb34bbf376efa07",
6     "userName": "Anjali",
7     "name": "New Chat",
8     "messages": [],
9     "createdAt": "2025-11-29T05:29:07.883Z",
10    "updatedAt": "2025-11-29T05:29:07.883Z",
11    "_v": 0
12  }
13

```

## [09] update chat name

The screenshot shows the Postman interface for the QuickGPT\_WMD\_PROJECT workspace. The left sidebar lists collections: AUTH, CHATS, MESSAGES, CREDITS, USER EXTRA, and WEBHOOKS. Under CHATS, there are POST Create Chat, GET All Chats, GET Get Chat By ID, and PUT Update Chat Name. The PUT request is selected. The URL is `PUT {{baseUrl}}/api/chats/ {{chatId}}`. The Body tab shows raw JSON: 

```
1 {  
2   "name": "My New Chat"  
3 }
```

. The response shows a 200 OK status with the following JSON body: 

```
1 {  
2   "success": true,  
3   "chat": {  
4     "_id": "692a84a3beb34bbf376efa0c",  
5     "userId": "692a842cbeb34bbf376efa07",  
6     "userName": "Anjali",  
7     "name": "My New Chat",  
8     "messages": [],  
9     "createdAt": "2025-11-29T05:29:07.883Z",  
10    "updatedAt": "2025-11-29T05:31:26.694Z",  
11    "__v": 0  
12  }  
13 }
```

.

## [10] send text message

The screenshot shows the Postman interface for the QuickGPT\_WMD\_PROJECT workspace. The left sidebar lists collections: AUTH, CHATS, MESSAGES, CREDITS, USER EXTRA, and WEBHOOKS. Under MESSAGES, there are POST Send Text Message and POST Generate Image Message. The POST Send Text Message request is selected. The URL is `POST {{baseUrl}}/api/messages/text`. The Body tab shows raw JSON: 

```
1 {  
2   "chatId": "fifchatId{j}",  
3   "prompt": "react js definition"  
4 }
```

. The response shows a 200 OK status with the following JSON body: 

```
1 {  
2   "success": true,  
3   "reply": {  
4     "content": "# React JS Definition:\n\nReact (also known as React.js or ReactJS) is a **free and open-source JavaScript library** for building user interfaces (UIs) or UI components. It is maintained by Facebook (Meta) and a community of individual developers and companies.\n\n**Key Characteristics and Features:**\n\n**Declarative:** You describe the desired UI state, and React efficiently updates the DOM to match that state. This simplifies reasoning about your code and makes it more predictable. Instead of directly manipulating the DOM (imperative), you describe *what* you want to see, and React figures out *how* to get there.\n\n**Component-Based:** React allows you to break down complex UIs into smaller, reusable, and independent pieces called components. Each component manages its own state and logic, making applications easier to develop, maintain, and test. Think of components as building blocks for your UI.\n\n**Virtual DOM:** React uses a virtual DOM (Document Object Model) which is a lightweight copy of the actual DOM. When data changes, React updates the
```

.

## [11] send text message API 2

POST Send Text Message

HTTP QuickGPT APIs / MESSAGES / Send Text Message

POST {{baseUrl}}/api/messages/text

```

1 {
2   "chatId": "{chatId}",
3   "prompt": "react js definition"
4 }

```

Body Cookies Headers (8) Test Results 200 OK 6.37 s 4.29 KB Save Response

```

elements\n\nExample (using JSX):**\n``javascript\nimport React from 'react';\n\nfunction Welcome({props}) {\n  return <h1>Hello, ${props.name}</h1>;\n}\n\nfunction App() {\n  return (\n    <div>\n      <Welcome name="Alice"/>\n      <Welcome name="Bob"/>\n    </div>\n  );\n}\n\nexport default App;
```\nIn this example:\n`Welcome` is a React component that accepts a `name` prop and renders a greeting.\n`App` is another React component that renders two `Welcome` components with different names.\nJSX is used to write the HTML-like syntax within the JavaScript code.\nIn summary, React is a powerful and flexible JavaScript library that empowers developers to create dynamic and efficient user interfaces through its component-based architecture, virtual DOM, and declarative programming style.\n", "role": "assistant", "timestamp": 1764394372306, "isImage": false

```

## [12] generate image message

POST Send Text Message POST Generate Image Message

HTTP QuickGPT APIs / MESSAGES / Generate Image Message

POST {{baseUrl}}/api/messages/image

```

1 {
2   "chatId": "fjchatId",
3   "prompt": "cute dog wearing sunglasses",
4   "isPublished": true
5 }

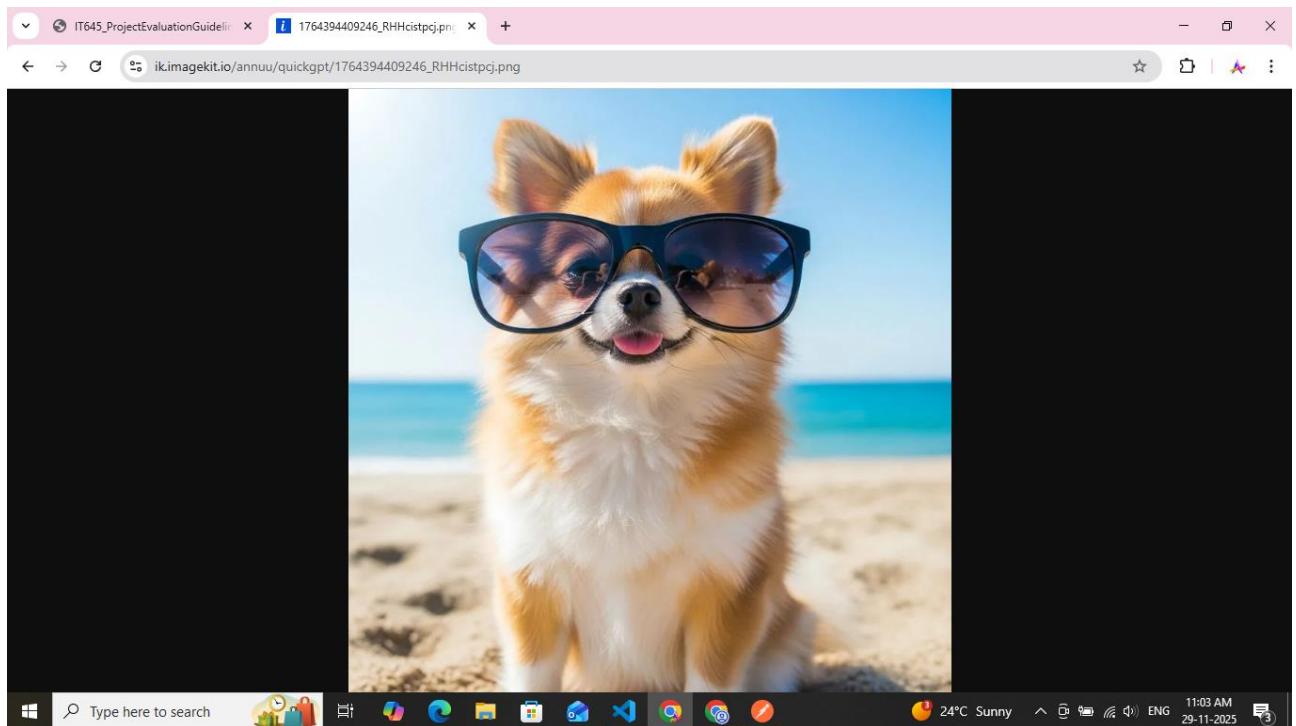
```

Body Cookies Headers (8) Test Results 200 OK 13.59 s 452 B Save Response

```

1 {
2   "success": true,
3   "reply": {
4     "role": "assistant",
5     "content": "https://ik.imagekit.io/annuu/quickgpt/1764394409246_RHHcistpcj.png",
6     "timestamp": 1764394411370,
7     "isImage": true,
8     "isPublished": true
9   }
10 }

```

**[13] getting generated image from link****[14] delete chat API**

Postman

QuickGPT\_WMD\_PROJECT

By Anjali Patel

Collections

Environments

History

Flows

API Network

DEL Delete Chat

HTTP QuickGPT APIs / CHATS / Delete Chat

DELETE {{baseUrl}} /api/chats/ {{chatid}}

Overview Params Authorization Headers (7) Body Scripts Settings Cookies

Headers 6 hidden

Key	Value	Description	Bulk Edit	Presets
Authorization	Bearer {{token}}			
Key	Value	Description		

Body Cookies Headers (8) Test Results

{ } JSON Preview Visualize

```

1  {
2    "success": true,
3    "message": "Chat deleted"
4  }

```

200 OK 141 ms 308 B Save Response

Online Find and replace Console Import Complete

Runner Start Proxy Cookies Vault Trash

24°C Sunny 11:04 AM 29-11-2025

## [15] get plans API

The screenshot shows the Postman interface with the 'QuickGPT\_WMD\_PROJECT' workspace selected. In the left sidebar, under the 'CREDITS' collection, the 'GET Get Plans' endpoint is highlighted. The main panel displays the 'Get Plans' request with the URL `HTTP QuickGPT APIs / CREDITS / Get Plans`. The 'Headers' tab is active, showing a single header entry: 'Key' (Value: 'Content-Type') and 'Description' (Value: 'application/json'). The 'Body' tab shows the JSON response:

```

1  {
2   "success": true,
3   "plans": [
4     {
5       "_id": "basic",
6       "name": "Basic",
7       "price": 10,
8       "credits": 100,
9       "features": [
10         "100 text",
11         "50 images"
12     ],
13   },
14 ]

```

The status bar at the bottom indicates '200 OK' with a response time of 8 ms and a size of 575 B.

## [16] get plans API 2

This screenshot shows the same Postman setup as the previous one, but the 'Body' tab now displays a response with two plan items:

```

22  ],
23  ],
24  ],
25  {
26   "_id": "premium",
27   "name": "Premium",
28   "price": 30,
29   "credits": 1000,
30   "features": [
31     "1000 text",
32     "500 images"
33   ],
34 }
35 ]

```

The status bar at the bottom indicates '200 OK' with a response time of 8 ms and a size of 575 B.

## [17] purchase plan

The screenshot shows the Postman interface with a collection named "QuickGPT\_WMD\_PROJECT". The "POST Purchase Plan" request is selected. The request URL is `POST {{baseUrl}} /api/credits/purchase`. The "Body" tab is active, showing the JSON payload:

```

1 {
2   "planId": "basic"
3 }

```

The response status is 200 OK, with a response time of 1.26 s and a size of 727 B. The response body is a long URL starting with `https://checkout.stripe.com/c/pay/cs_test_a174mLxRpaxlPhSjxhjmuSQhPjxxeK3yEMDklwbN9UwtpO1ZJa1rAh0ubR#fidnandhYHdWcXxpYCc%2FJ2FgY2RwaXEnKSdkdWx...`.

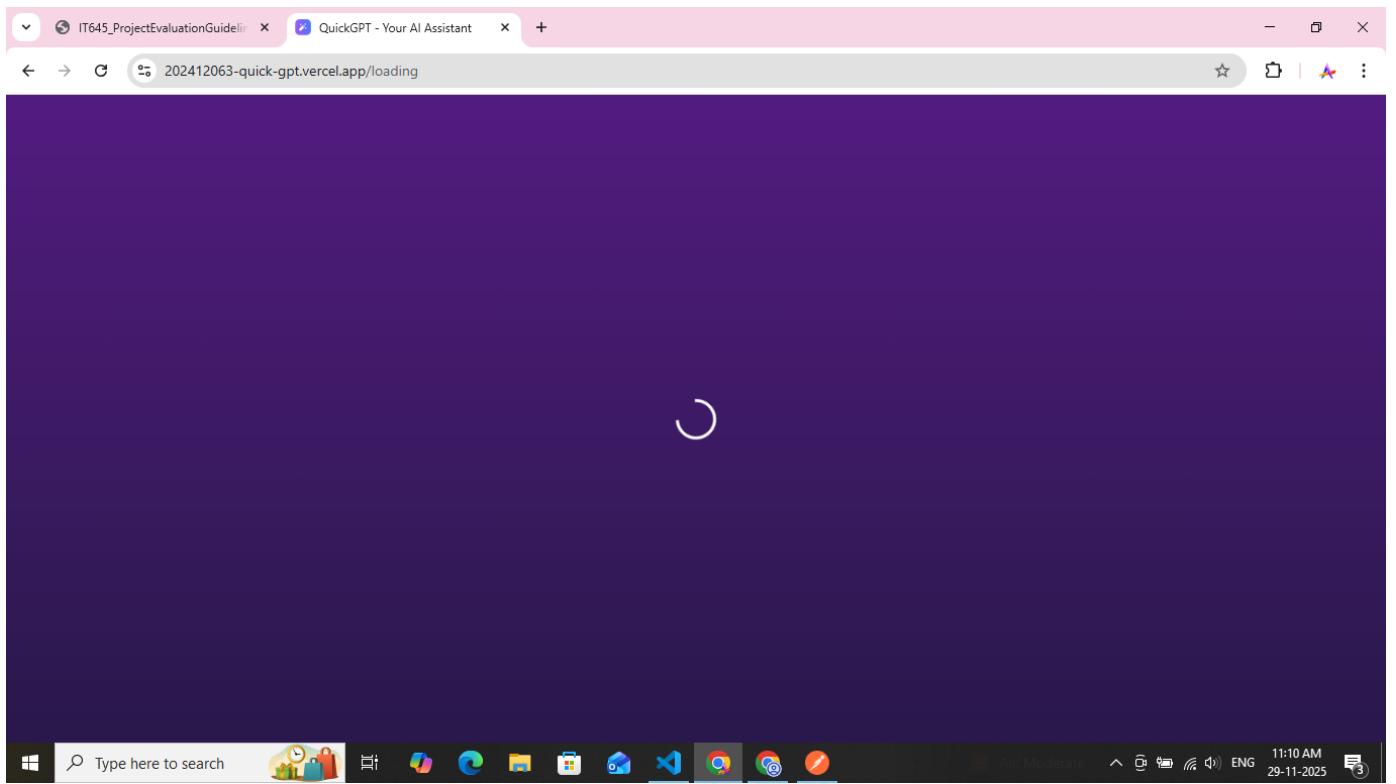
## [18] payment processing

The screenshot shows a browser window for "IT645\_ProjectEvaluationGuideline" with a tab titled "Checkout". The URL is `checkout.stripe.com/c/pay/cs_test_a174mLxRpaxlPhSjxhjmuSQhPjxxeK3yEMDklwbN9UwtpO1ZJa1rAh0ubR#fidnandhYHdWcXxpYCc%2FJ2FgY2RwaXEnKSdkdWx...`. The page is in TEST MODE.

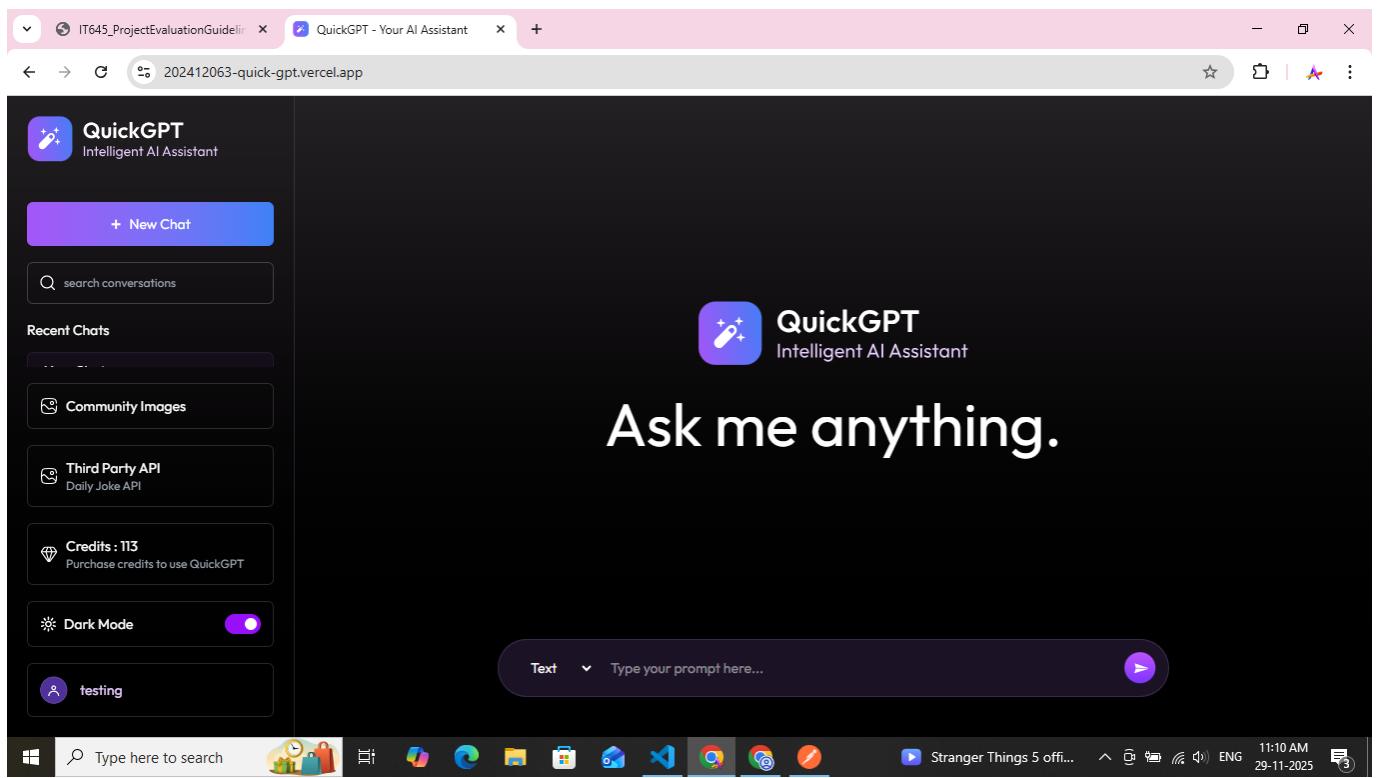
The payment amount is \$10.00 (USD). The payment method section includes:

- Pay with link** button
- OR** separator
- Email**: anjalipatel3074@gmail.com
- Payment method** section:
  - Card information**: Card number 4111 1111 1111 1111, Exp 12/28, CVV 123
  - Cardholder name**: ANJALI PATEL
  - Country or region**: India
  - Save my information for faster checkout** checkbox (unchecked)
- Pay** button

## [19] redirected after sucessful payment



## [20] updated credits



## [21] get my transactions API

The screenshot shows the Postman interface with the following details:

- Collection:** QuickGPT\_WMD\_PROJECT
- Request Type:** GET
- URL:** {{baseUrl}}/api/credits/transactions
- Headers:**
  - Authorization: Bearer {{token}}
- Body:** (JSON) [JSON content]
- Response:** 200 OK (165 ms, 732 B)
 

```

1  {
2    "success": true,
3    "transactions": [
4      {
5        "_id": "692a870476f7f3a4fb28ce65",
6        "userId": "692a842cbeb34bbf376efa07",
7        "planId": "basic",
8        "amount": 10,
9        "credits": 100,
10       "isPaid": false,
11       "createdAt": "2025-11-29T08:39:16.844Z",
12       "updatedAt": "2025-11-29T05:39:16.844Z",
13       "__v": 0
14     },
      ]
    }
  
```

## [22] get credits of particular user API

The screenshot shows the Postman interface with the following details:

- Collection:** QuickGPT\_WMD\_PROJECT
- Request Type:** GET
- URL:** {{baseUrl}}/api/credits/me
- Headers:**
  - Authorization: Bearer {{token}}
  - Content-Type: application/json
- Body:** (JSON) [JSON content]
- Response:** 200 OK (145 ms, 384 B)
 

```

1  {
2    "success": true,
3    "credits": 17,
4    "user": {
5      "_id": "692a842cbeb34bbf376efa07",
6      "name": "Anjali",
7      "email": "anjali@example.com"
8    }
9  }
  
```

## [23] get user by ID API

Postman screenshot showing the 'Get User By ID' API call. The request URL is `GET {{baseUrl}} /api/users/ {userId}`. The response is a 200 OK with a JSON body:

```

1 {
2   "success": true,
3   "user": {
4     "_id": "692a842cbeb34bbf376efa07",
5     "name": "Anjali",
6     "email": "anjali@example.com",
7     "credits": 17,
8     "__v": 0
9   }
10 }

```

## [24] update user credits API

Postman screenshot showing the 'PATCH Update User Credits' API call. The request URL is `PATCH {{baseUrl}} /api/users/ {userId} /credits`. The response is a 200 OK with a JSON body:

```

1 {
2   "success": true,
3   "user": {
4     "_id": "692a842cbeb34bbf376efa07",
5     "name": "Anjali",
6     "email": "anjali@example.com",
7     "password": "$2b$10$xxL8L8cXDq.14R9oYHr40./iuud7Fosj9iw67sx9lmEf2gMGIQxTe",
8     "credits": 5000,
9     "__v": 0
10 }
11 }

```

## [25] get all published images API

The screenshot shows the Postman interface with the 'QuickGPT\_WMD\_PROJECT' workspace selected. In the left sidebar, under 'Collections', the 'USER EXTRA' collection is expanded, and the 'GET Get All Published Images' request is highlighted. The request URL is `GET {{baseUrl}} /api/users/published-images`. The 'Body' tab is selected, showing the response body as JSON:

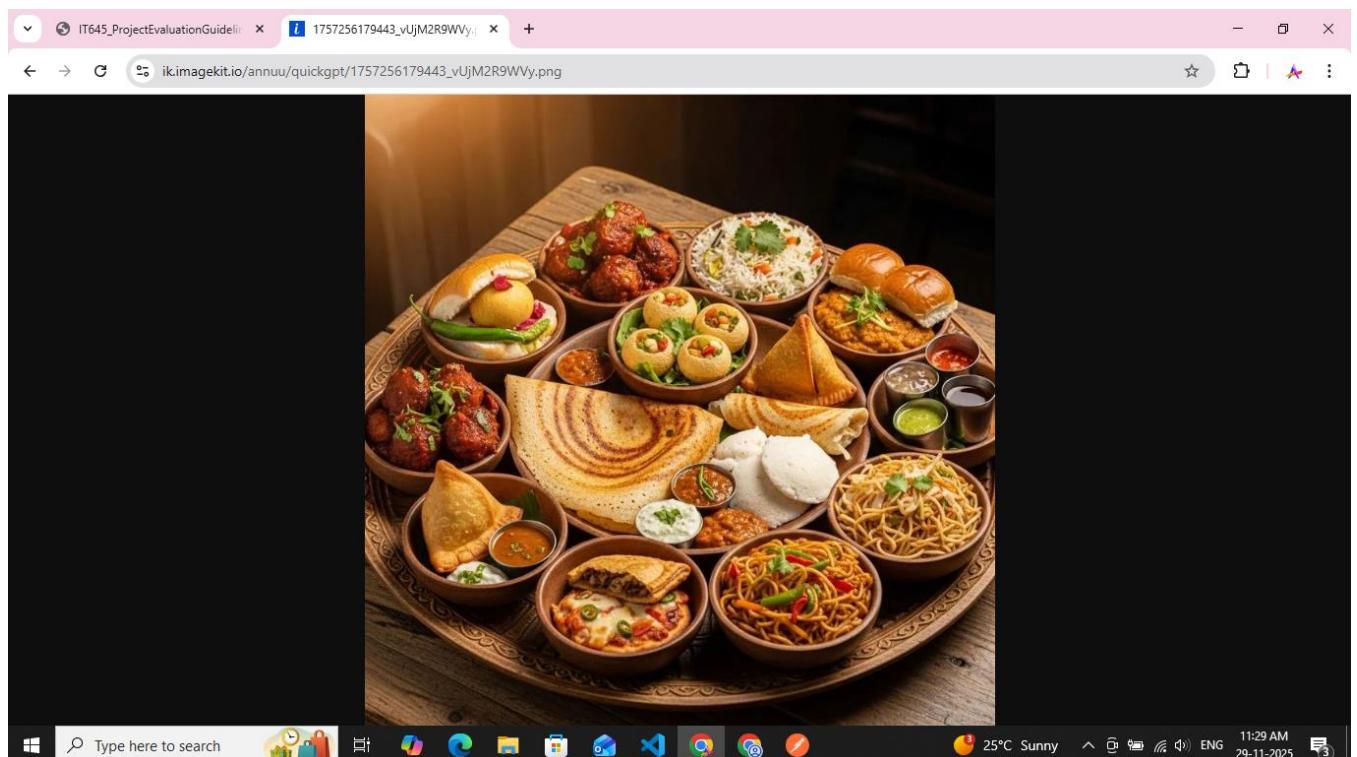
```

1  {
2    "success": true,
3    "images": [
4      {
5        "imageUrl": "https://ik.imagekit.io/annuu/quickgpt/1757256179443_vUjM2R9WVY.png",
6        "userName": "ANJALI PATEL",
7        "createdAt": 1757256183360
8      }
9    ]
10 }

```

The response status is 200 OK with 77 ms latency and 429 B size. The bottom status bar shows the Windows taskbar with the date and time as 29-11-2025 11:29 AM.

## [26] getting published image from link



## [27] get random joke third party API

The screenshot shows the Postman application interface. On the left, the sidebar displays collections, environments, history, and flows. The main workspace shows a collection named "QuickGPT\_WMD\_PROJECT" with a sub-collection "THIRD PARTY API". A specific request titled "GET Get Random Joke" is selected. The request details show a GET method and the URL <https://official-joke-api.appspot.com/jokes/random>. The Headers tab is active, showing an Accept header set to application/json. The Body tab shows the JSON response received:

```

1 {
2   "type": "general",
3   "setup": "if you're American when you go into the bathroom, and American when you come out, what are you when
        you're in there?",
4   "punchline": "European",
5   "id": 38
6 }

```

The response status is 200 OK, with a duration of 543 ms and a size of 566 B. The response body is displayed in JSON format.

## **Justification : motivation, benefits, and need**

QuickGPT is built to give users a simple and reliable way to use AI for text and image generation without dealing with complex setups or high costs. Modern AI tools are powerful but expensive, so there is a real need for a controlled, secure and scalable platform that offers these features in a practical way.

### ➤ **Motivation :**

- Users want quick access to AI chat and image tools in one place
- AI APIs cost money, so a credit-based model is necessary to prevent misuse
- Developers need a real-world project that covers authentication, payments and API integration
- Secure, fast and responsive web apps are now standard expectations

### ➤ **Benefits :**

- Unified platform for chat, image generation and community posts
- Fair usage through credit-based billing
- Safe login and secure payments using JWT and stripe
- Scalable architecture using react, node.js, and mongoDB
- Engaging experience through a community gallery

### ➤ **Need :**

- MERN stack provides speed, flexibility and easy scaling
- JWT enables clean, stateless authentication for API routes
- Stripe handles payments safely without exposing sensitive data
- Backend gatekeeping is essential to validate credits before calling expensive AI APIs
- Proper architecture ensures performance, maintainability and controlled AI usage