LABORATORY REPORT

Submitted by

PRASANTH S (23BIR039)

in partial fulfilment of the

requirements for the

award of thedegree of

BACHELOR OF SCIENCE

IN

INFORMATION SYSTEMS

DEPARTMANT OF COMPUTER TECHNOLOGY-UG

KONGU ENGINEERING COLLEGE

(Autonomous)

PERUNDURAI, ERODE – 638 060

Generative Ai And Prompt Engineering

Experiment 1: Set up the development environment with Node.js, Next.js, Git, and Google AI Studio

Aim:

Set up a full-stack AI development environment.

Algorithm:

- - Install Node.js and Git.
- - Create a Next.js project using create-next-app.
- - Initialize Git and commit changes.
- - Install Axios for HTTP requests.
- - Set up Google AI Studio and obtain Gemini API key.
- - Run the development server to verify.

Program:

```
npx create-next-app ai-app
cd ai-app
git init
npm install axios
```

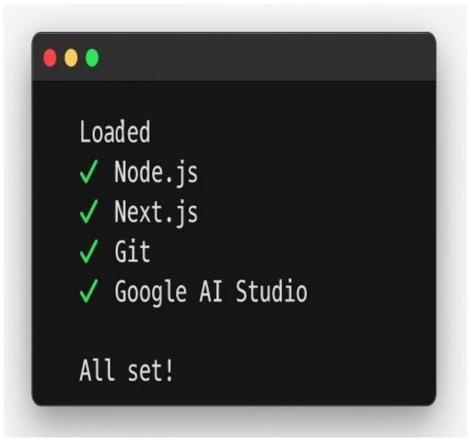
Output:

Development server starts at http://localhost:3000

Result:

Environment successfully configured.

Final Output Display Screen:



```
Loaded
✓ Node.js
✓ Next.js
✓ Git
✓ Google AI Studio

All set!
```

Experiment 2: Create your first AI conversation using Google Gemini

Aim:

Build a basic chatbot using the Gemini text model.

Algorithm:

- - Create form to capture user input.
- - Send input to Gemini API.
- - Receive and display the AI response.

Program:

1. npm init -y

2. npm install @google/generative-ai dotenv

3. Create .env file:

GOOGLE_API_KEY=your_api_key_here

4. Run:

node gemini-chat.js

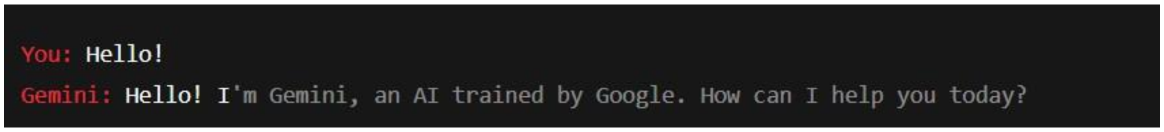
Output:

User enters text, receives AI-generated reply.

Result:

Text-based AI conversation implemented.

Final Output Display Screen:



```
You: Hello!
Gemini: Hello! I'm Gemini, an AI trained by Google. How can I help you today?
```

Experiment 3: Implement image upload functionality in a Next.js app

Aim:

Allow users to upload and preview images.

Algorithm:

- - Create file input field.
- - Convert file to previewable URL.
- - Show the uploaded image in the UI.

Program:

```
<input type="file" onChange={handleChange} />
{image && <img src={image} width="200" />}
```

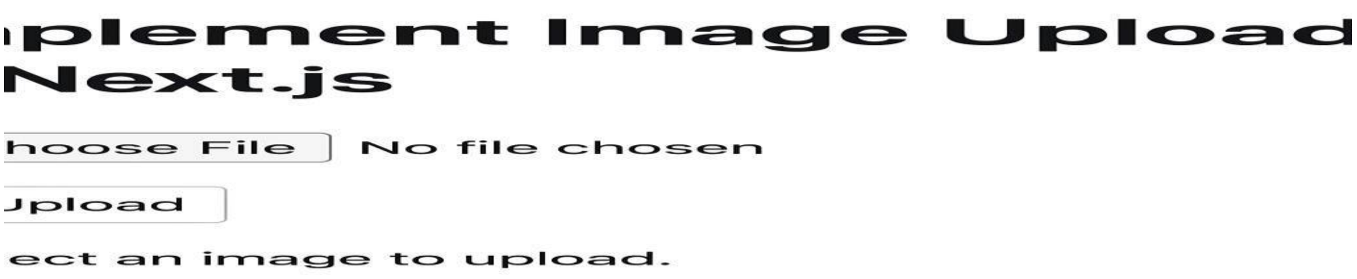
Output:

Uploaded image preview appears in browser.

Result:

Basic image upload and preview works.

Final Output Display Screen:



plement Image Upload
Next.js

hoose File No file chosen

Jpload

ect an image to upload.

Experiment 4: Analyze uploaded images using Google Gemini Vision API

Aim:

Use Gemini Vision to analyze images.

Algorithm:

- - Convert image to base64.
- - Send to Gemini Vision endpoint.
- - Display description returned by API.

Program:

```
const response = await axios.post(Gemini_URL, {
  contents: [{
    parts: [{
      inlineData: {
        mimeType: "image/jpeg",
        data: base64Data
      }
    }]
  }]
});
```

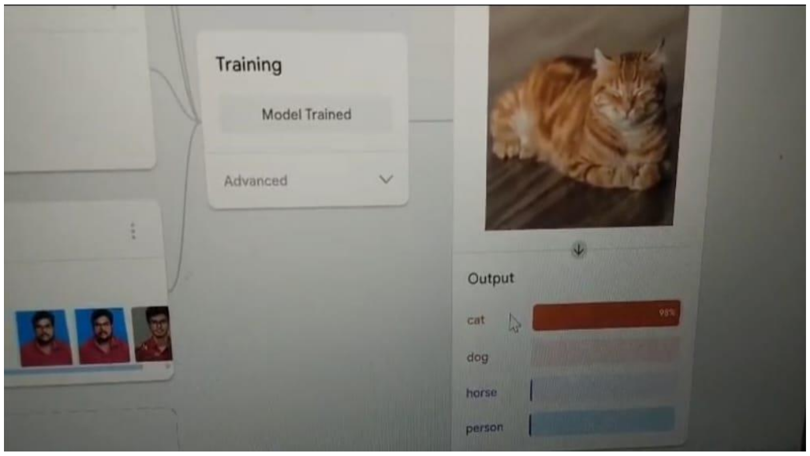
Output:

Image analysis result is shown.

Result:

AI successfully understands uploaded image.

Final Output Display Screen:



Experiment 5: Build an AI-powered product description generator

Aim:

Generate intelligent product descriptions from structured input.

Algorithm:

- - Collect product name and features.
- - Send prompt to Gemini API.
- - Display response in readable format.

Program:

```
if __name__ == "__main__":
    product_name = "Smart Home Hub Pro"
   key_features = ["Voice control", "Energy monitoring", "Device integration", "Enhanced
security"]
    target_audience = "Tech-savvy homeowners"
    tone = "informative and sophisticated"
    length = "medium"

   description = generate_product_description(product_name, key_features,
target_audience, tone, length)
   print(description)
```

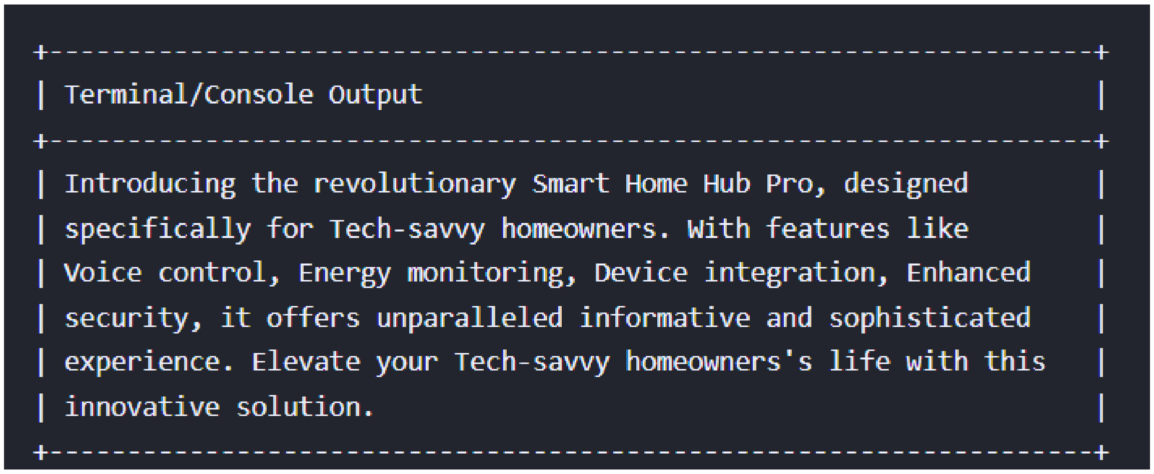
Output:

Detailed marketing description generated.

Result:

Text generated from product info is coherent and clear.

Final Output Display Screen:



```
+---------------------------------------------------------------------+
| Terminal/Console Output                                             |
+---------------------------------------------------------------------+
| Introducing the revolutionary Smart Home Hub Pro, designed          |
| specifically for Tech-savvy homeowners. With features like          |
| Voice control, Energy monitoring, Device integration, Enhanced      |
| security, it offers unparalleled informative and sophisticated      |
| experience. Elevate your Tech-savvy homeowners's life with this     |
| innovative solution.                                                |
+---------------------------------------------------------------------+
```

Experiment 6: Experiment with prompt engineering to influence AI responses

Aim:

Refine AI output using structured and varied prompts.

Algorithm:

- - Use multiple prompt styles.
- - Observe changes in responses.
- - Choose optimal structure for your use case.

Program:

const prompt = "Explain artificial intelligence to a 10-year-old in 2 sentences.";

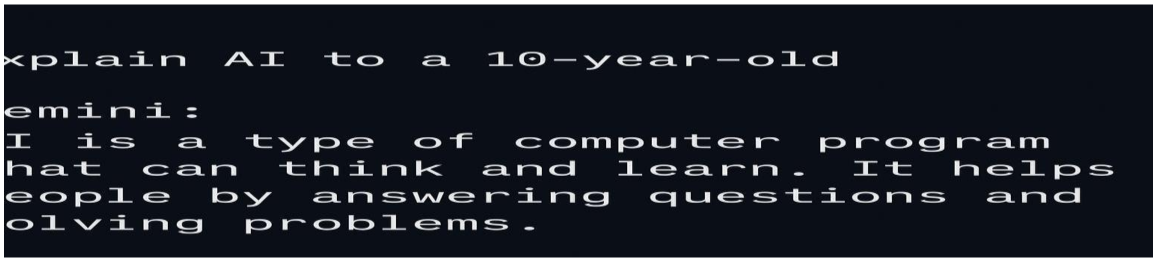
Output:

Short, age-appropriate response.

Result:

Demonstrates control over tone and complexity via prompts.

Final Output Display Screen:



```
xplain AI to a 10-year-old

emini:
I is a type of computer program
hat can think and learn. It helps
eople by answering questions and
olving problems.
```

Experiment 7: Design a chatbot user interface using React and Tailwind CSS

Aim:

Create a styled, user-friendly chatbot UI.

Algorithm:

- - Create layout using React components.
- - Style with Tailwind classes.
- - Add scrollable message area and input bar.

Program:

```
<div className="chat-box">
  {messages.map(msg => <div>{msg.role}: {msg.text}</div>)}
  <input type="text" className="input-box" />
</div>
```

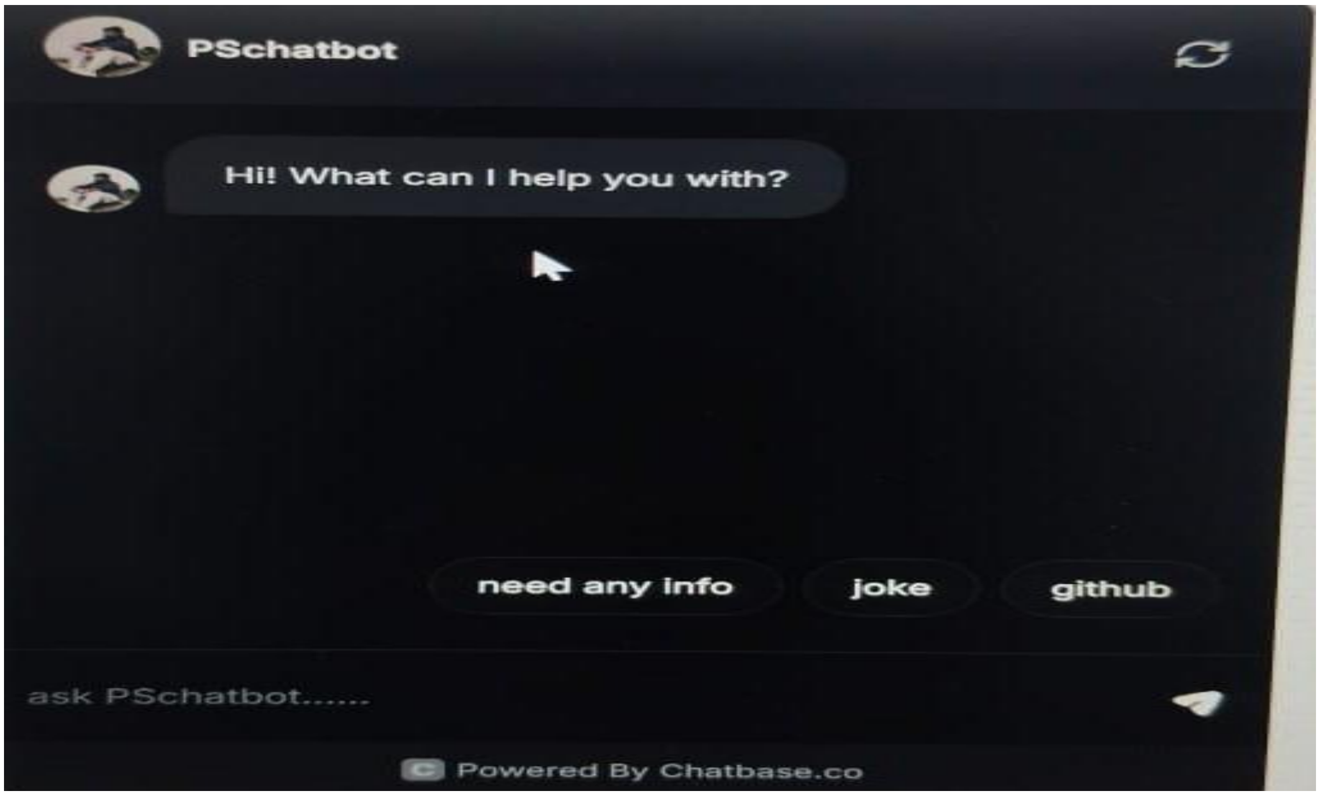
Output:

Styled chatbot interface is rendered.

Result:

Chat UI is visually appealing and interactive.

Final Output Display Screen:



Experiment 8: Maintain chat memory and flow using state management

Aim:

Use React state to preserve and display conversation history.

Algorithm:

- - Store messages in state array.
- - Append new user and AI messages.
- - Render all messages in order.

Program:

```
const [messages, setMessages] = useState([]);
setMessages([...messages, { role: 'User', text: input }, { role: 'AI', text:
response }]);
```

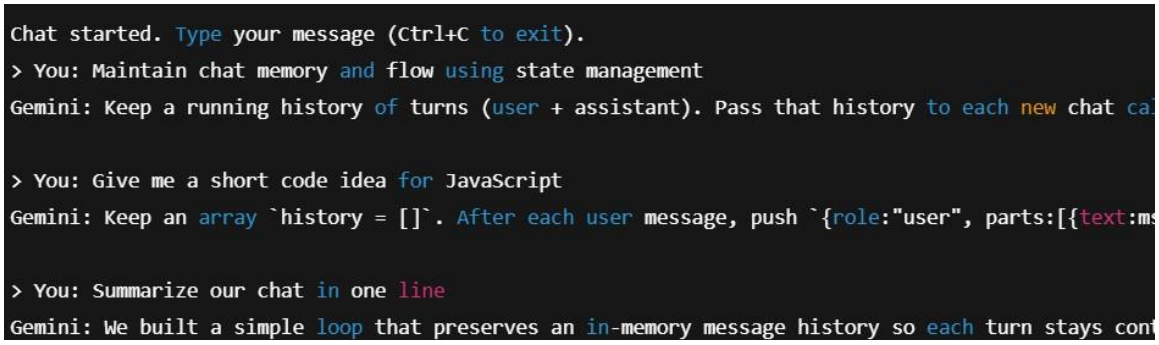
Output:

Message history scrolls and updates in real time.

Result:

State-based memory allows continuous conversations.

Final Output Display Screen:



```
Chat started. Type your message (Ctrl+C to exit).
> You: Maintain chat memory and flow using state management
Gemini: Keep a running history of turns (user + assistant). Pass that history to each new chat ca

> You: Give me a short code idea for JavaScript
Gemini: Keep an array `history = []`. After each user message, push `{role:"user", parts:[{text:ms

> You: Summarize our chat in one line
Gemini: We built a simple loop that preserves an in-memory message history so each turn stays cont
```

Experiment 9: Apply responsible AI practices and structure your codebase

Aim:

Ensure safe, secure, and maintainable AI app code.

Algorithm:

- - Validate user input.
- - Handle API errors.
- - Use environment variables.
- - Organize files into modules.

Program:

```
if (!prompt || prompt.length > 500) return;
try {
  const res = await axios.post("/api/gemini", { prompt });
} catch (err) {
  console.error("Error calling AI API");
}
```

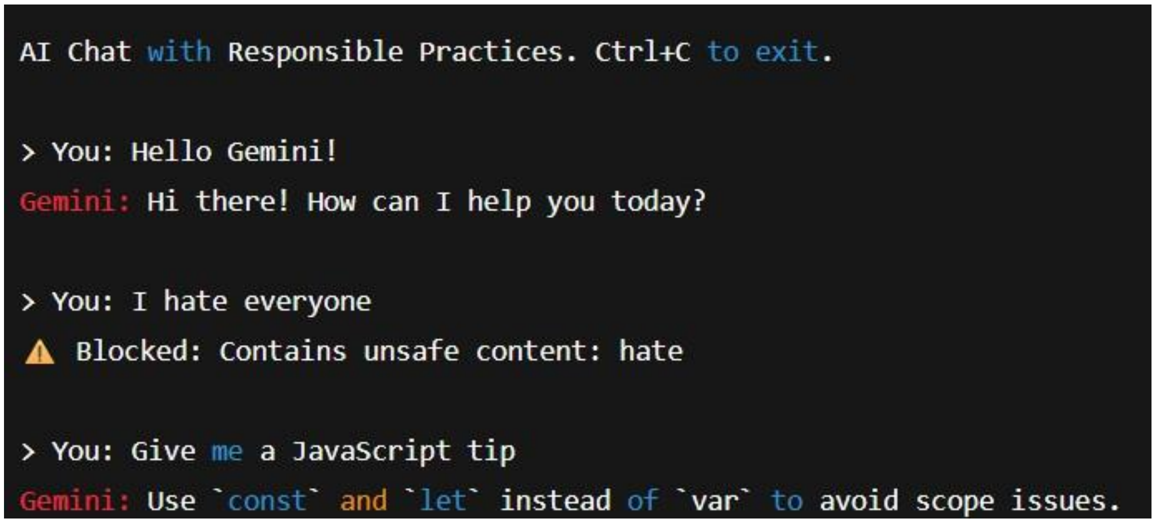
Output:

App handles unexpected inputs gracefully.

Result:

App is production-safe and responsibly designed.

Final Output Display Screen:



```
AI Chat with Responsible Practices. Ctrl+C to exit.

> You: Hello Gemini!
Gemini: Hi there! How can I help you today?

> You: I hate everyone
⚠ Blocked: Contains unsafe content: hate

> You: Give me a JavaScript tip
Gemini: Use `const` and `let` instead of `var` to avoid scope issues.
```

Experiment 10: Deploy your AI web app using Vercel or Firebase

Aim:

Publish the AI web app to the internet.

Algorithm:

- - Push project to GitHub.
- - Connect GitHub repo to Vercel.
- - Set environment variables (Gemini API key).
- - Deploy.

Program:

```
npm install -g vercel

vercel

 npm install -g firebase-tools

firebase login

firebase init hosting

npm run build

firebase deploy
```

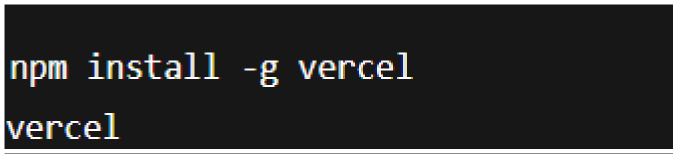
Output:

Site is live and publicly accessible.

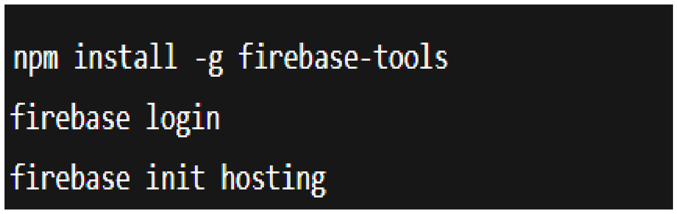
Result:

AI app is deployed and functioning on the web.

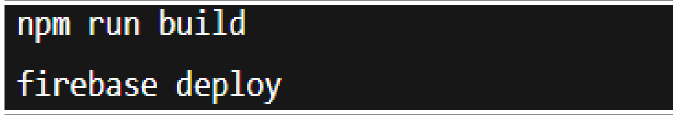
Final Output Display Screen:



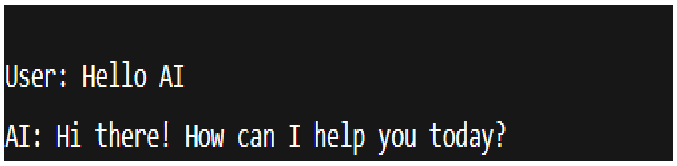
```
npm install -g vercel
vercel
```



```
npm install -g firebase-tools
firebase login
firebase init hosting
```



```
npm run build
firebase deploy
```



```
User: Hello AI
AI: Hi there! How can I help you today?
```