TANMAY AGRAWAL

PROJECT DOCUMENTATION

AI Chat Assistant

Github Link for Project - https://github.com/Tanmay0029/ChatBot

Model Used - TinyLlama/TinyLlama-1.1B-Chat-v1.0

Link to HuggingFace model(to download locally) - https://huggingface.co/TinyLlama/TinyLlama-1.1B-Chat-v1.0

Project Overview

The AI Chat Assistant is a web-based application that provides an interactive chat interface powered by TinyLlama downloading the model from HuggingFace, a lightweight language model. The application allows users to engage in conversations, ask questions, and receive both text and code-based responses. The system includes features like chat history management, theme switching (dark/light mode), and specialized code display with syntax highlighting.

Tech Stack

Backend Technologies

- **FastAPI**: A modern Python web framework for building APIs
- **PyTorch**: Deep learning framework used with TinyLlama model
- **TinyLlama**: Lightweight language model (1.1B parameters) for text generation
- **Torch DirectML**: For GPU acceleration support
- **Jinja2**: Template engine for rendering HTML
- **Pydantic**: Data validation using Python type annotations

Frontend Technologies

- **HTML5/CSS3**: Structure and styling of the web interface
- **JavaScript (Vanilla)**: Client-side functionality
- **Axios**: HTTP client for making API requests
- **highlight.js**: Code syntax highlighting
- **Material Icons**: UI icons from Google's Material Design
- **CSS Custom Properties**: For theme management

System Architecture

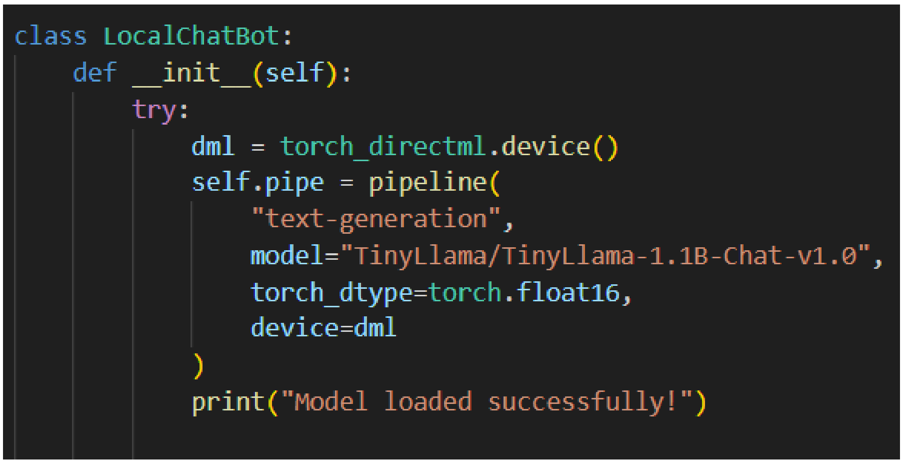
API Endpoints

1. **POST /chat**
    - Handles incoming chat messages
    - Processes user input through the TinyLlama model
    - Returns structured responses with generation time
2. **GET /**
    - Serves the main chat interface
    - Renders the chat.html template
3. **GET /chat-history**
    - Retrieves conversation history
4. **POST /clear-history**
    - Clears the conversation history
5. **POST /toggle-theme**
    - Manages theme switching functionality

File-by-File Analysis

1. app.py - Backend Implementation

Class: LocalChatBot



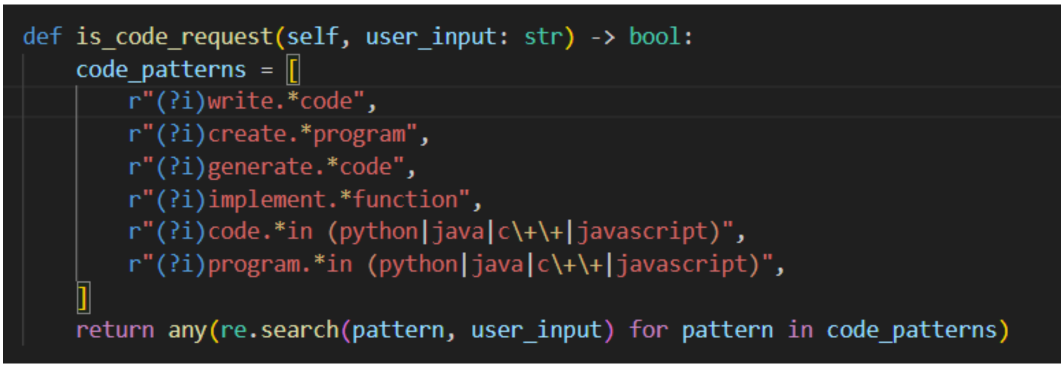
```
class LocalChatBot:
    def __init__(self):
        try:
            dml = torch_directml.device()
            self.pipe = pipeline(
                "text-generation",
                model="TinyLlama/TinyLlama-1.1B-Chat-v1.0",
                torch_dtype=torch.float16,
                device=dml
            )
            print("Model loaded successfully!")
```

Key Components:

1. Model Initialization

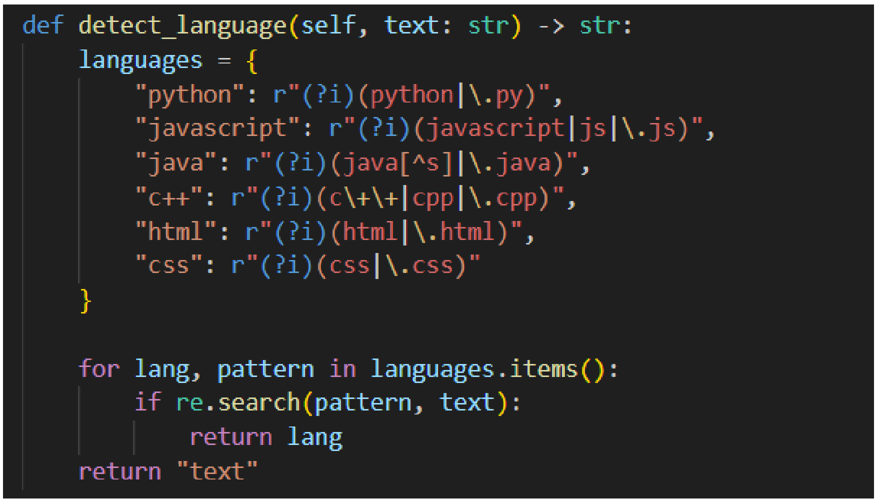
- Uses TinyLlama-1.1B-Chat-v1.0 model
- Implements DirectML for GPU acceleration
- Configures float16 precision for optimal performance
- Initializes system message for chat context

2. Pattern Detection Methods



```
def is_code_request(self, user_input: str) -> bool:
    code_patterns = [
        r"(?i)write.*code",
        r"(?i)create.*program",
        r"(?i)generate.*code",
        r"(?i)implement.*function",
        r"(?i)code.*in (python|java|c\+\+|javascript)",
        r"(?i)program.*in (python|java|c\+\+|javascript)",
    ]
    return any(re.search(pattern, user_input) for pattern in code_patterns)
```

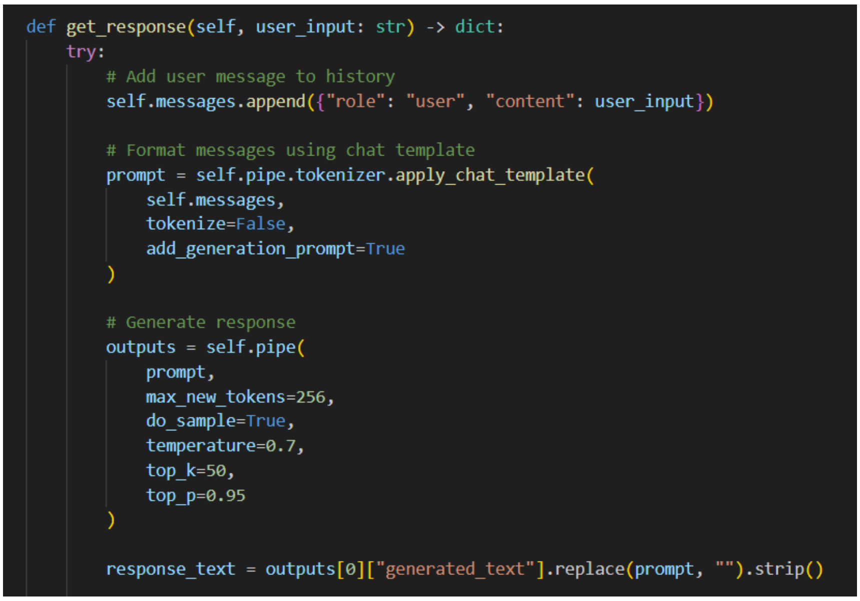
- Utilizes regex patterns to identify code-related requests
- Patterns cover various programming languages
- Checks for keywords like "write", "create", "implement”, “generate”



```
def detect_language(self, text: str) -> str:
    languages = {
        "python": r"(?i)(python|\.py)",
        "javascript": r"(?i)(javascript|js|\.js)",
        "java": r"(?i)(java[^s]|\.java)",
        "c++": r"(?i)(c\+\+|cpp|\.cpp)",
        "html": r"(?i)(html|\.html)",
        "css": r"(?i)(css|\.css)"
    }

    for lang, pattern in languages.items():
        if re.search(pattern, text):
            return lang
    return "text"
```

- Identifies programming language from text
- Supports Python, JavaScript, Java, C++, HTML, CSS
- Uses regex patterns for language detection

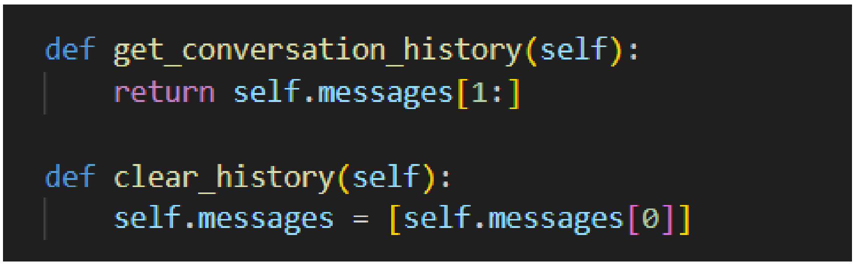
3. Response Generation



```
def get_response(self, user_input: str) -> dict:
    try:
        # Add user message to history
        self.messages.append({"role": "user", "content": user_input})

        # Format messages using chat template
        prompt = self.pipe.tokenizer.apply_chat_template(
            self.messages,
            tokenize=False,
            add_generation_prompt=True
        )

        # Generate response
        outputs = self.pipe(
            prompt,
            max_new_tokens=256,
            do_sample=True,
            temperature=0.7,
            top_k=50,
            top_p=0.95
        )

        response_text = outputs[0]["generated_text"].replace(prompt, "").strip()
```

- Processes user input through TinyLlama model
- Formats chat messages using tokenizer
- Structures response based on content type (code/text)
- Generates responses with specified parameters:
    - max_new_tokens=256
    - temperature=0.7
    - top_k=50
    - top_p=0.95

4. History Management

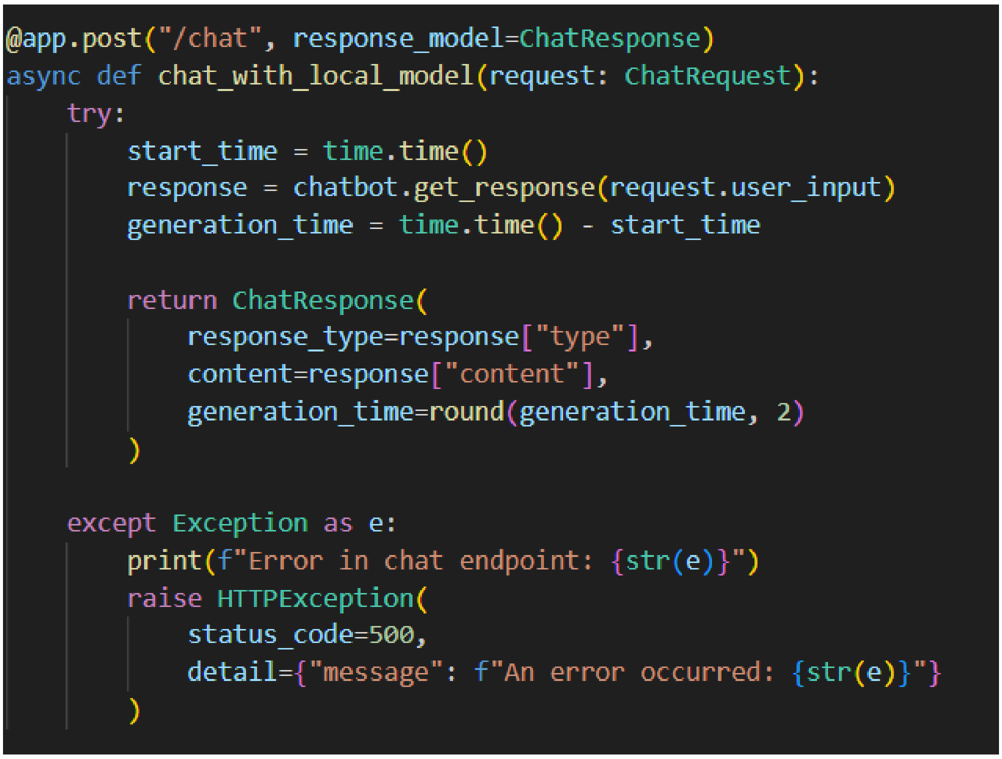


```python
def get_conversation_history(self):
    return self.messages[1:]

def clear_history(self):
    self.messages = [self.messages[0]]
```

- Maintains conversation context
- Provides history retrieval and clearing functionality

FastAPI Routes

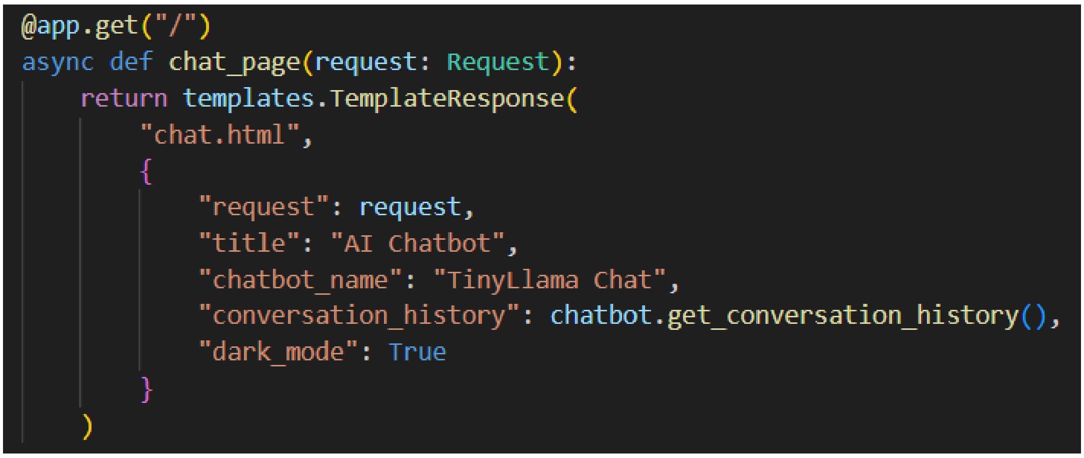
1. Chat Endpoint



```python
@app.post("/chat", response_model=ChatResponse)
async def chat_with_local_model(request: ChatRequest):
    try:
        start_time = time.time()
        response = chatbot.get_response(request.user_input)
        generation_time = time.time() - start_time

        return ChatResponse(
            response_type=response["type"],
            content=response["content"],
            generation_time=round(generation_time, 2)
        )

    except Exception as e:
        print(f"Error in chat endpoint: {str(e)}")
        raise HTTPException(
            status_code=500,
            detail={"message": f"An error occurred: {str(e)}"}
        )
```

- Handles incoming chat messages
- Processes through LocalChatBot instance
- Returns structured response with timing information
- Implements error handling

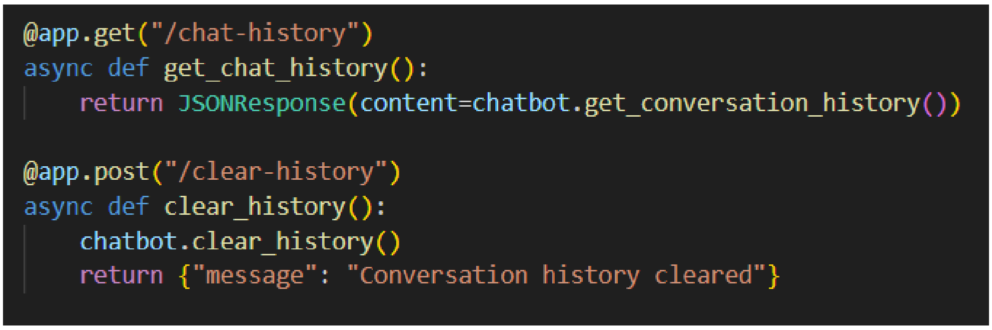
2. Main Page Route



```
@app.get("/")
async def chat_page(request: Request):
    return templates.TemplateResponse(
        "chat.html",
        {
            "request": request,
            "title": "AI Chatbot",
            "chatbot_name": "TinyLlama Chat",
            "conversation_history": chatbot.get_conversation_history(),
            "dark_mode": True
        }
    )
```

- Serves chat interface
- Passes conversation history and theme settings
- Renders using Jinja2 templates

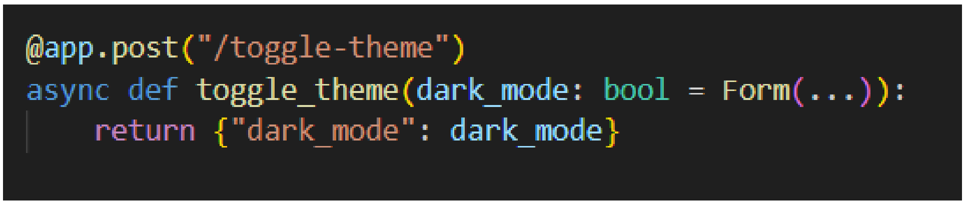
3. History Management Routes



```
@app.get("/chat-history")
async def get_chat_history():
    return JSONResponse(content=chatbot.get_conversation_history())

@app.post("/clear-history")
async def clear_history():
    chatbot.clear_history()
    return {"message": "Conversation history cleared"}
```

- Handle conversation history retrieval and clearing
- Return JSON responses

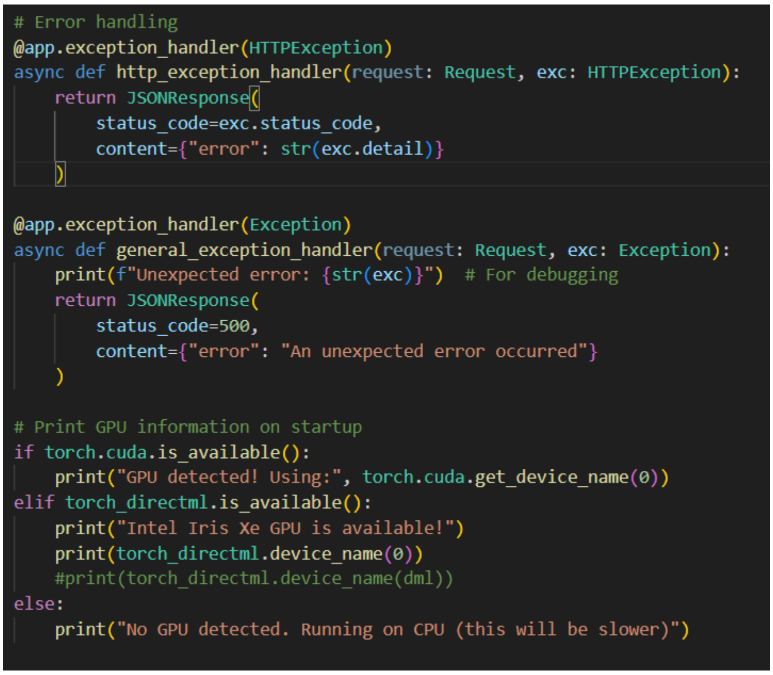
4. Theme Toggle Route



```
@app.post("/toggle-theme")
async def toggle_theme(dark_mode: bool = Form(...)):
    return {"dark_mode": dark_mode}
```

- Manages theme switching functionality
- Accepts form data for theme state

Error Handling



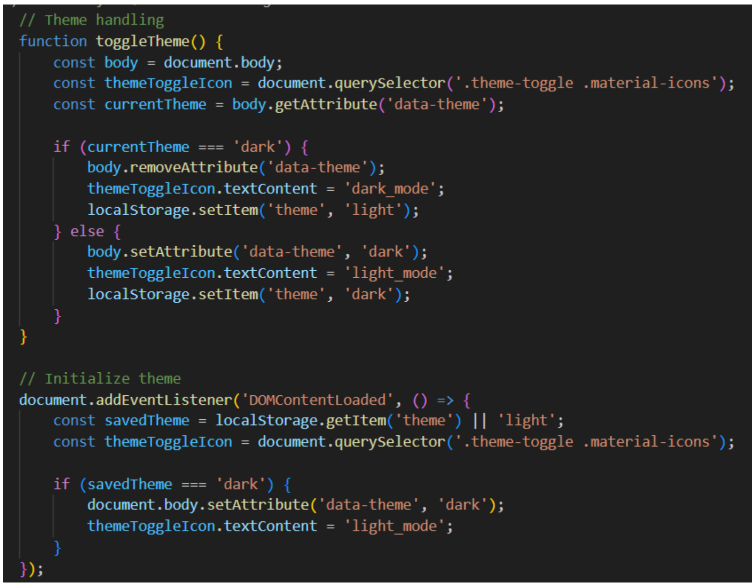
```python
# Error handling
@app.exception_handler(HTTPException)
async def http_exception_handler(request: Request, exc: HTTPException):
    return JSONResponse(
        status_code=exc.status_code,
        content={"error": str(exc.detail)}
    )

@app.exception_handler(Exception)
async def general_exception_handler(request: Request, exc: Exception):
    print(f"Unexpected error: {str(exc)}")  # For debugging
    return JSONResponse(
        status_code=500,
        content={"error": "An unexpected error occurred"}
    )

# Print GPU information on startup
if torch.cuda.is_available():
    print("GPU detected! Using:", torch.cuda.get_device_name(0))
elif torch_directml.is_available():
    print("Intel Iris Xe GPU is available!")
    print(torch_directml.device_name(0))
    #print(torch_directml.device_name(dml))
else:
    print("No GPU detected. Running on CPU (this will be slower)")
```

- Custom exception handlers for HTTP and general exceptions
- Structured error responses
- Debug logging for unexpected errors

2. main.js - Frontend Implementation

Core Functions

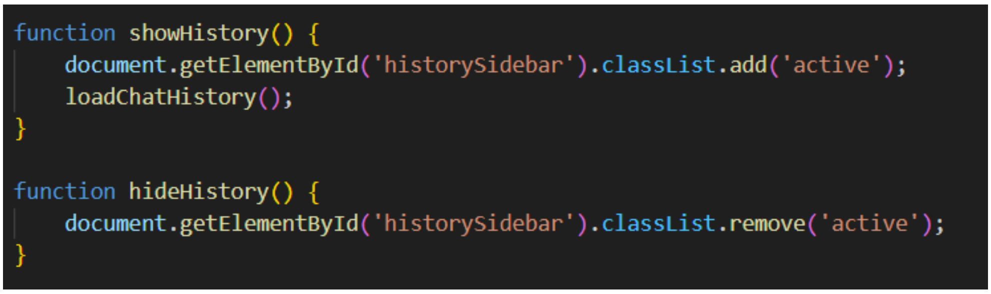
1. Theme Management



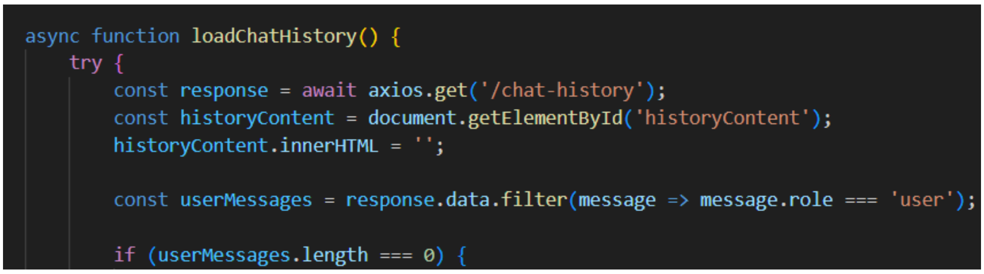
```javascript
// Theme handling
function toggleTheme() {
    const body = document.body;
    const themeToggleIcon = document.querySelector('.theme-toggle .material-icons');
    const currentTheme = body.getAttribute('data-theme');

    if (currentTheme === 'dark') {
        body.removeAttribute('data-theme');
        themeToggleIcon.textContent = 'dark_mode';
        localStorage.setItem('theme', 'light');
    } else {
        body.setAttribute('data-theme', 'dark');
        themeToggleIcon.textContent = 'light_mode';
        localStorage.setItem('theme', 'dark');
    }
}

// Initialize theme
document.addEventListener('DOMContentLoaded', () => {
    const savedTheme = localStorage.getItem('theme') || 'light';
    const themeToggleIcon = document.querySelector('.theme-toggle .material-icons');

    if (savedTheme === 'dark') {
        document.body.setAttribute('data-theme', 'dark');
        themeToggleIcon.textContent = 'light_mode';
    }
});
```

- Switches between light/dark themes
- Updates UI elements
- Persists theme preference in localStorage
- Manages theme-specific icon changes

2. History Management



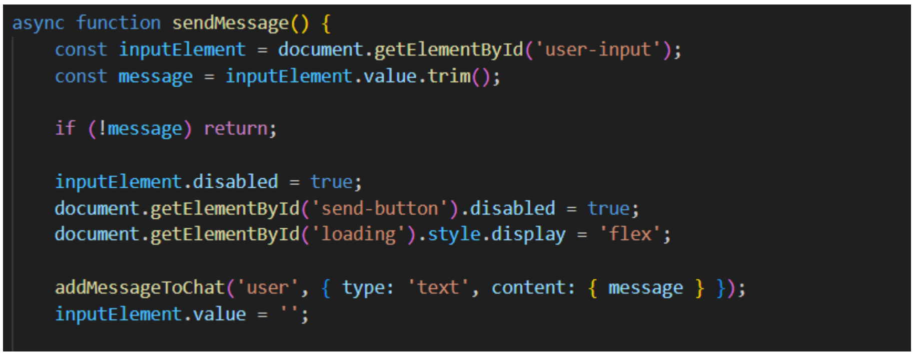
```
function showHistory() {
    document.getElementById('historySidebar').classList.add('active');
    loadChatHistory();
}

function hideHistory() {
    document.getElementById('historySidebar').classList.remove('active');
}
```



```
async function loadChatHistory() {
    try {
        const response = await axios.get('/chat-history');
        const historyContent = document.getElementById('historyContent');
        historyContent.innerHTML = '';

        const userMessages = response.data.filter(message => message.role === 'user');

        if (userMessages.length === 0) {
```

- Controls history sidebar visibility
- Fetches and displays chat history
- Implements click-to-load functionality
- Formats timestamps and messages

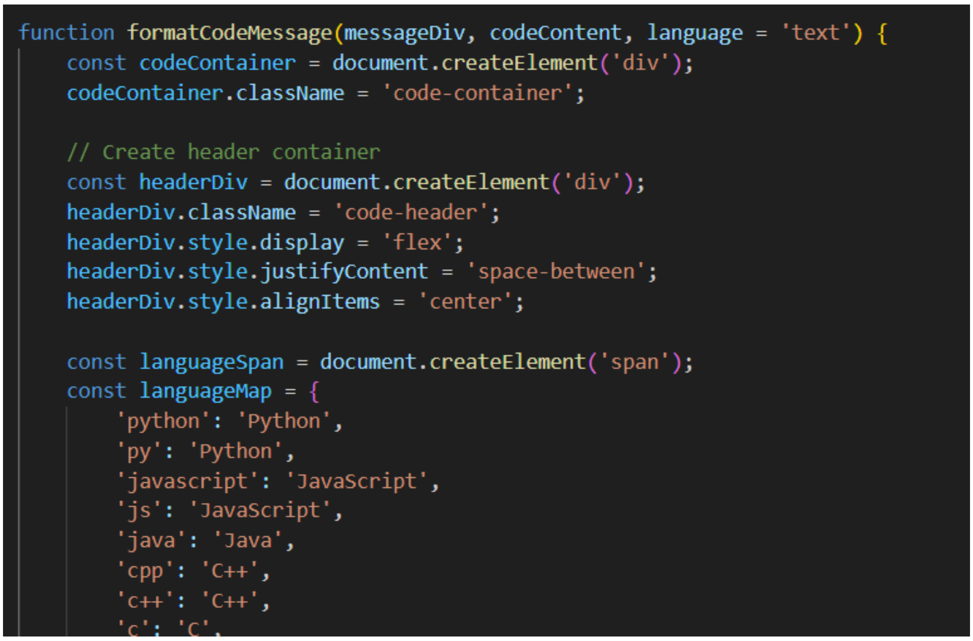
3. Message Handling



```
async function sendMessage() {
    const inputElement = document.getElementById('user-input');
    const message = inputElement.value.trim();

    if (!message) return;

    inputElement.disabled = true;
    document.getElementById('send-button').disabled = true;
    document.getElementById('loading').style.display = 'flex';

    addMessageToChat('user', { type: 'text', content: { message } });
    inputElement.value = '';
```

- Captures user input
- Manages UI states (loading, disabled)
- Makes API requests using Axios
- Handles response processing
- Updates chat interface

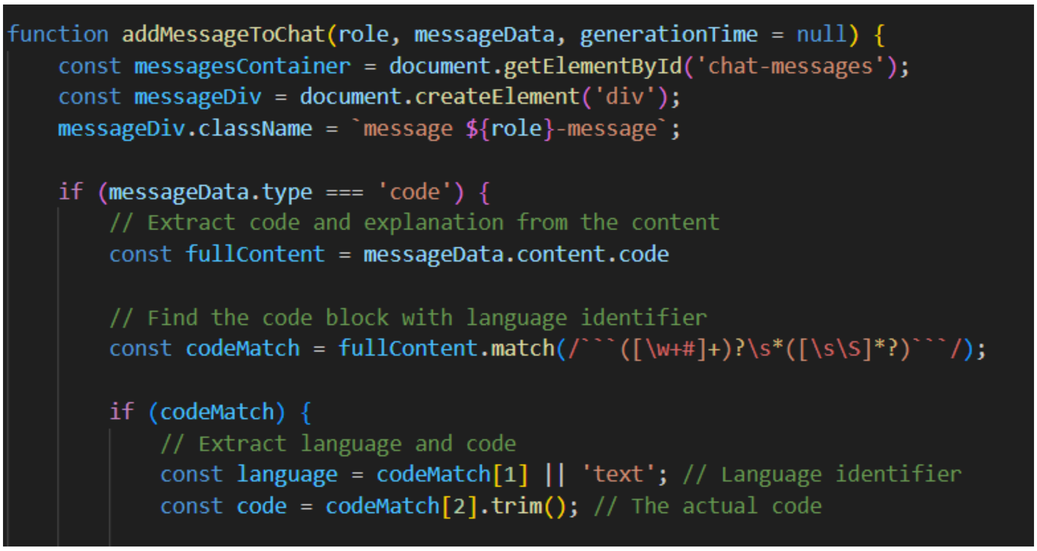
4. Code Formatting



```
function formatCodeMessage(messageDiv, codeContent, language = 'text') {
    const codeContainer = document.createElement('div');
    codeContainer.className = 'code-container';

    // Create header container
    const headerDiv = document.createElement('div');
    headerDiv.className = 'code-header';
    headerDiv.style.display = 'flex';
    headerDiv.style.justifyContent = 'space-between';
    headerDiv.style.alignItems = 'center';

    const languageSpan = document.createElement('span');
    const languageMap = {
        'python': 'Python',
        'py': 'Python',
        'javascript': 'JavaScript',
        'js': 'JavaScript',
        'java': 'Java',
        'cpp': 'C++',
        'c++': 'C++',
        'c': 'C',
```

- Creates structured code containers
- Implements copy/download functionality
- Manages language-specific highlighting
- Handles code header and buttons
- Supports multiple programming languages

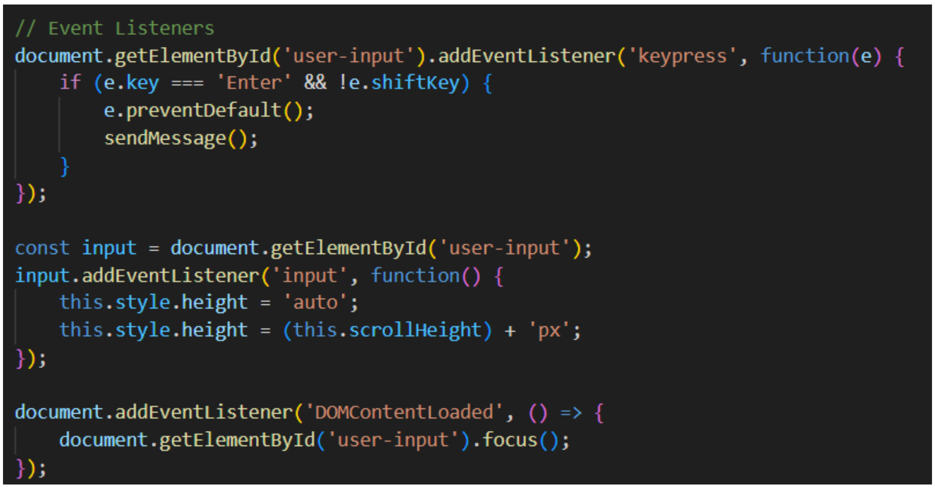
5. Chat Management



```
function addMessageToChat(role, messageData, generationTime = null) {
    const messagesContainer = document.getElementById('chat-messages');
    const messageDiv = document.createElement('div');
    messageDiv.className = `message ${role}-message`;

    if (messageData.type === 'code') {
        // Extract code and explanation from the content
        const fullContent = messageData.content.code

        // Find the code block with language identifier
        const codeMatch = fullContent.match(/```([\w+#]+)?\s*([\s\S]*?)```/);

        if (codeMatch) {
            // Extract language and code
            const language = codeMatch[1] || 'text'; // Language identifier
            const code = codeMatch[2].trim(); // The actual code
```

- Renders messages in chat interface
- Handles different message types (text/code)
- Adds generation time for bot responses
- Manages chat scrolling

6. Event Listeners



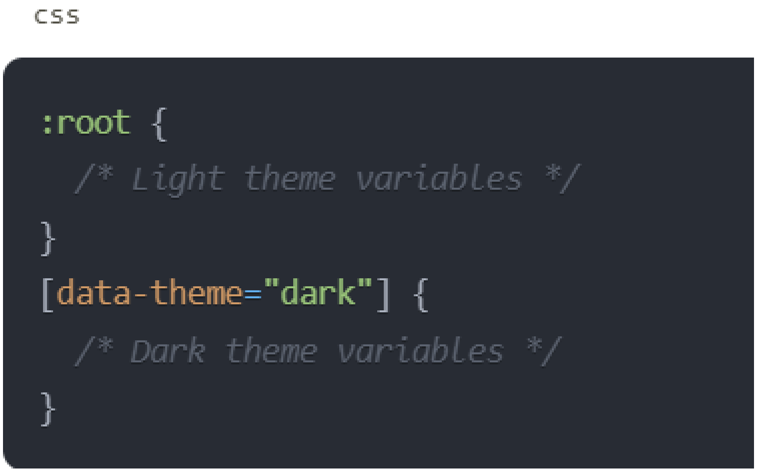
```
// Event Listeners
document.getElementById('user-input').addEventListener('keypress', function(e) {
    if (e.key === 'Enter' && !e.shiftKey) {
        e.preventDefault();
        sendMessage();
    }
});

const input = document.getElementById('user-input');
input.addEventListener('input', function() {
    this.style.height = 'auto';
    this.style.height = (this.scrollHeight) + 'px';
});

document.addEventListener('DOMContentLoaded', () => {
    document.getElementById('user-input').focus();
});
```

- Input handling (keypress, auto-resize)
- Theme initialization
- Focus management
- Button click handlers

3. styles.css - Styling Implementation

Theme System

1. CSS Variables

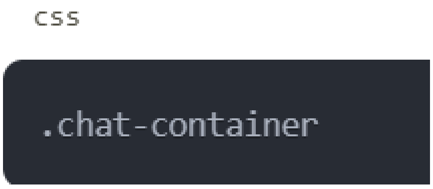


```css
:root {
  /* Light theme variables */
}
[data-theme="dark"] {
  /* Dark theme variables */
}
```

- Complete color scheme for both themes
- Transition properties
- Shadow and border configurations

2. Layout Components

a. Chat Container

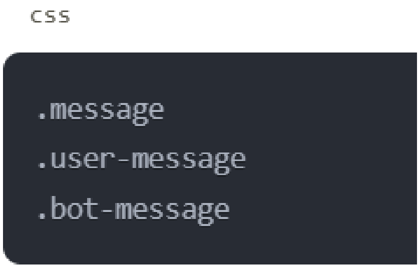


css

```css
.chat-container
```

- Maximum width constraints
- Responsive margins and padding
- Shadow and border effects
- Background colour management

b. Message Styling

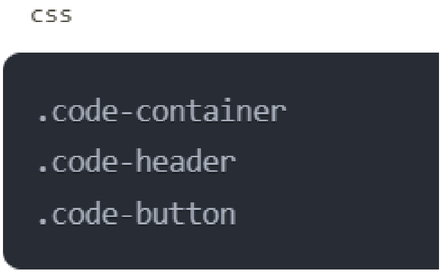


css

```css
.message
.user-message
.bot-message
```

- Message positioning
- Color schemes for different message types
- Responsive text sizing
- Margin and padding management

c. Code Block Styling



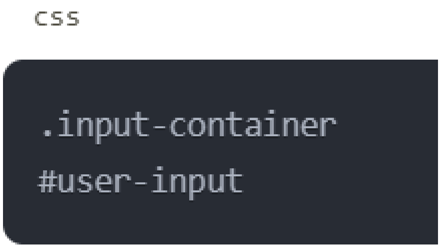
css

```css
.code-container
.code-header
.code-button
```

- Syntax highlighting support
- Copy/download button styling
- Language header formatting
- Code block margins and padding

3. **Interactive Elements**

a. **Input Area**

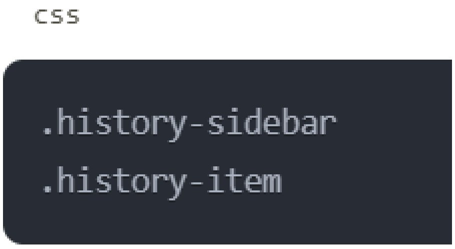


css

```
.input-container
#user-input
```

- Flexible input sizing
- Button positioning
- Focus states
- Placeholder styling

b. **History Sidebar**



css

```
.history-sidebar
.history-item
```

- Sliding animation
- Item hover states
- Scroll behavior
- Responsive sizing

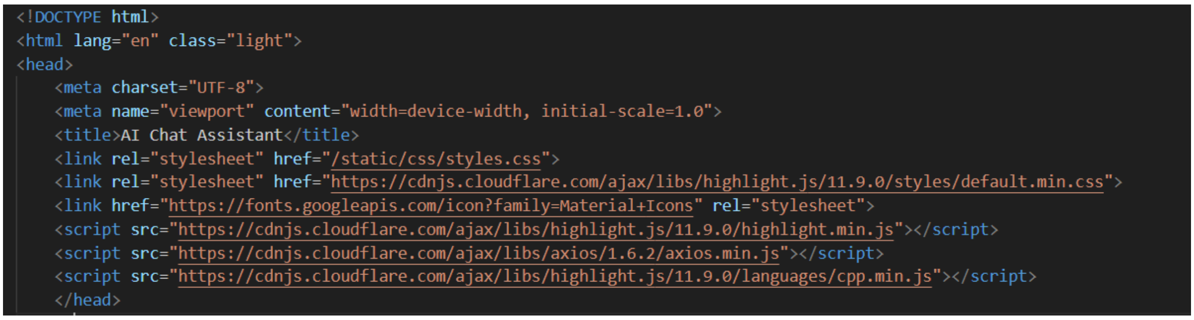
4. **Utility Styles**

- Custom scrollbar
- Loading animations
- Transition effects
- Responsive breakpoints

4. chat.html - Interface Structure

Document Structure

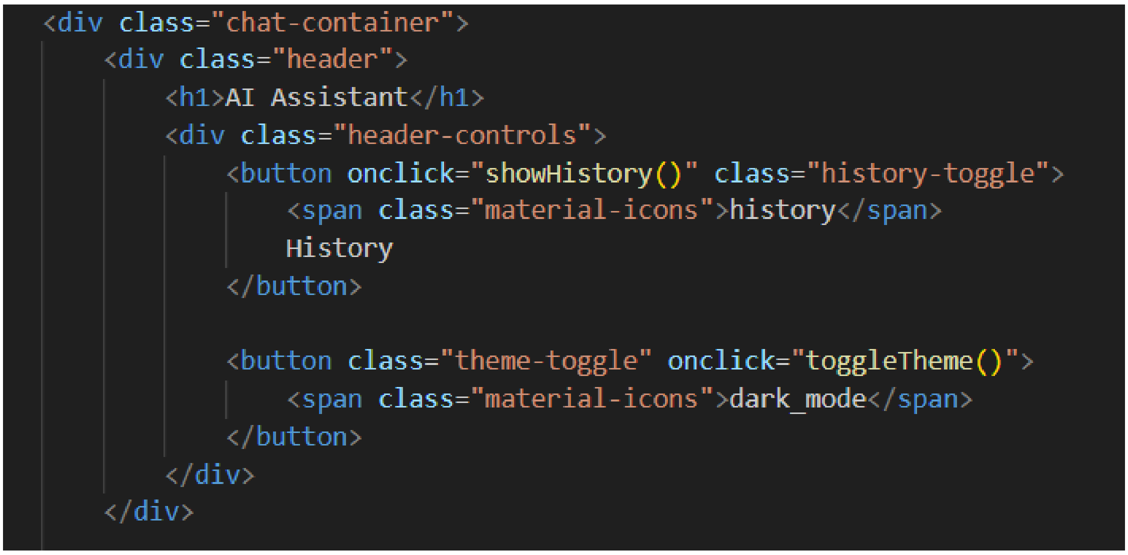
1. Head Section



```html
<!DOCTYPE html>
<html lang="en" class="light">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AI Chat Assistant</title>
    <link rel="stylesheet" href="/static/css/styles.css">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/highlight.js/11.9.0/styles/default.min.css">
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
    <script src="https://cdnjs.cloudflare.com/ajax/libs/highlight.js/11.9.0/highlight.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/axios/1.6.2/axios.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/highlight.js/11.9.0/languages/cpp.min.js"></script>
    </head>
```

- Meta tags for responsiveness
- External resource loading
- CSS and JavaScript imports
- Font loading

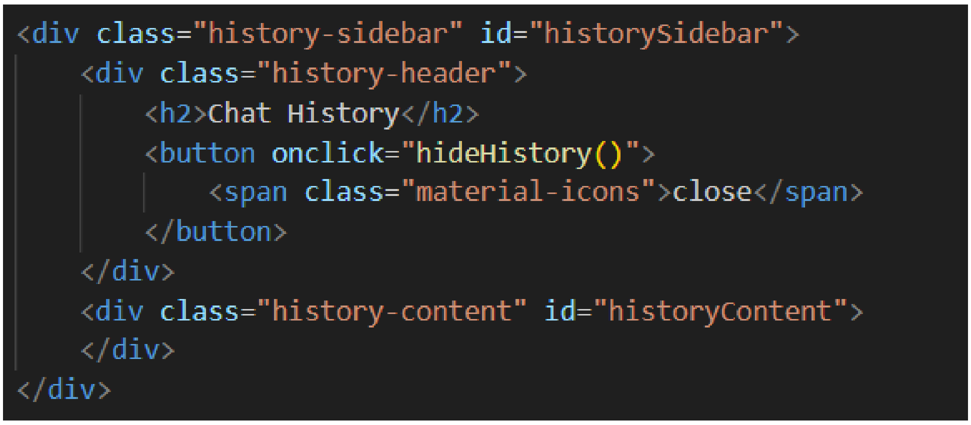
2. Main Container



```html
<div class="chat-container">
    <div class="header">
        <h1>AI Assistant</h1>
        <div class="header-controls">
            <button onclick="showHistory()" class="history-toggle">
                <span class="material-icons">history</span>
                History
            </button>

            <button class="theme-toggle" onclick="toggleTheme()">
                <span class="material-icons">dark_mode</span>
            </button>
        </div>
    </div>
```

- Header with title and controls
- Message display area
- Input section with buttons
- Loading indicator

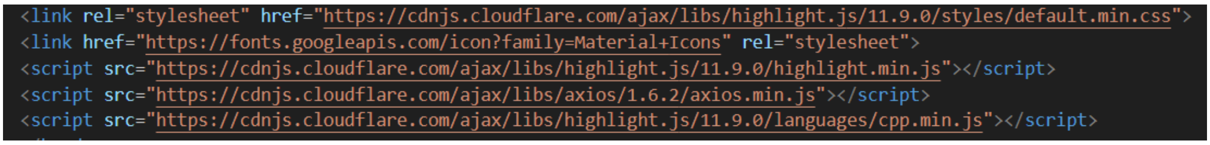
3. History Sidebar



```html
<div class="history-sidebar" id="historySidebar">
    <div class="history-header">
        <h2>Chat History</h2>
        <button onclick="hideHistory()">
            <span class="material-icons">close</span>
        </button>
    </div>
    <div class="history-content" id="historyContent">
    </div>
</div>
```

- Header with close button
- History content container
- Dynamic message loading

4. Script Integration



```html
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/highlight.js/11.9.0/styles/default.min.css">
<link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
<script src="https://cdnjs.cloudflare.com/ajax/libs/highlight.js/11.9.0/highlight.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/axios/1.6.2/axios.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/highlight.js/11.9.0/languages/cpp.min.js"></script>
```

- highlight.js for code syntax
- Axios for API requests
- Custom JavaScript functionality
- Material Icons integration

Key Features

1. Natural Language Processing

- TinyLlama model processes user inputs
- Contextual response generation
- Code detection and language identification

2. Code Handling

- Automatic language detection
- Syntax highlighting
- Copy and download functionality
- Language-specific formatting

3. User Interface

- Real-time response display
- Chat history management
- Theme customization
- Loading states and error handling

4. Performance Optimization

- DirectML GPU acceleration
- Efficient message processing
- Optimized code rendering

5. Responsive Design

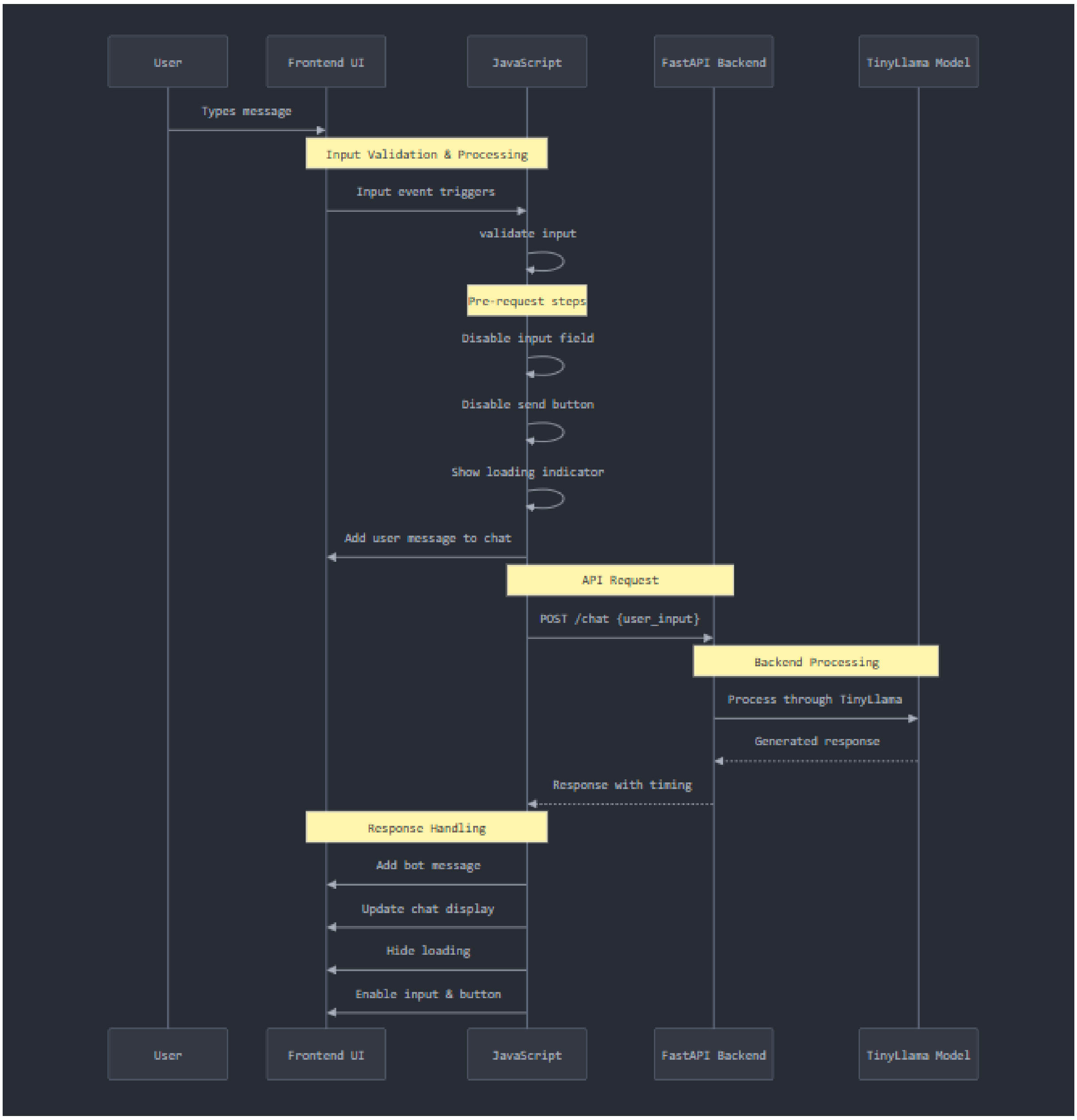
- Mobile-friendly layout
- Flexible containers
- Adaptive sizing

Data Flow

1. **User Input**
    - User enters message
    - Frontend validates and sends to API
2. **Backend Processing**
    - FastAPI receives request
    - TinyLlama model generates response
    - Response is formatted based on content type
3. **Response Handling**
    - Frontend receives structured response
    - UI updates with new message
    - Code blocks are formatted if present
4. **State Management**
    - Chat history is maintained
    - Theme preferences are preserved

    - Conversation context is managed

Data Flow Diagram – User Input to API Processing



Security Considerations

- Input validation on both frontend and backend
- Error handling and logging
- Secure API endpoints
- Safe code execution handling

Performance Features

- GPU acceleration support
- Efficient message processing
- Optimized code rendering
- Responsive UI design

Future Enhancements

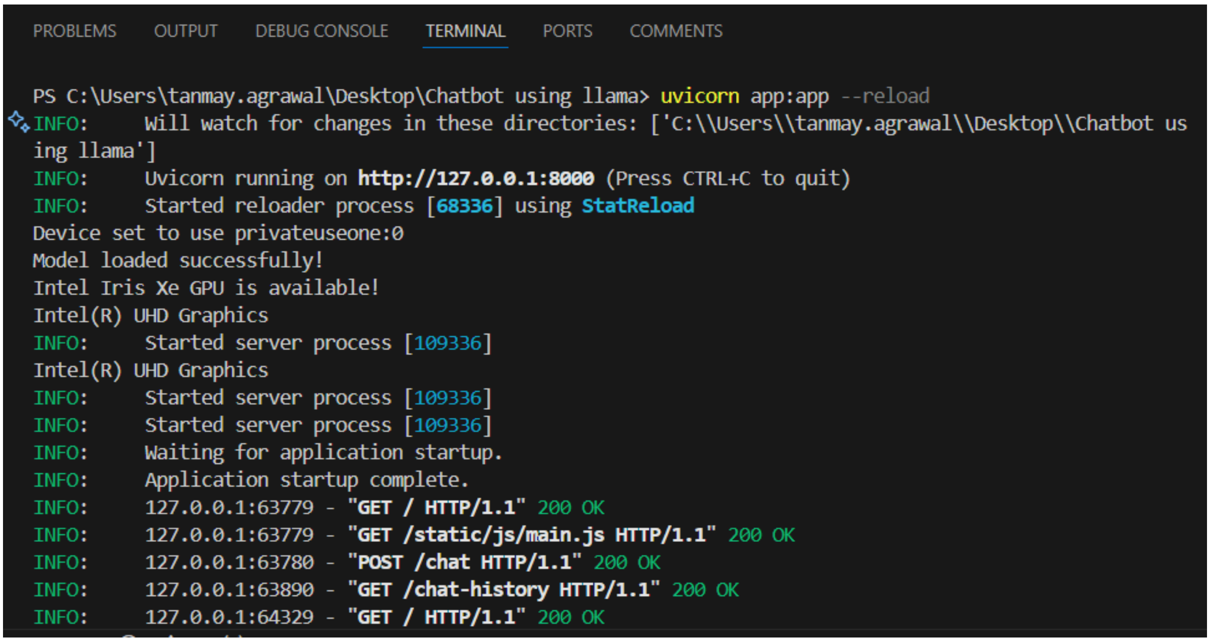
1. **Potential Improvements**
    - User authentication system
    - Enhanced code execution capabilities
    - Multi-language support
    - File upload/download features
    - Voice and video support
    - Advanced conversation context management
2. **Scalability Considerations**
    - Database integration for persistent storage
    - Load balancing for multiple users
    - Reducing Execution/Response Generation time
    - Caching mechanisms
    - API rate limiting

Conclusion

The AI Chat Assistant demonstrates a modern approach to building an interactive chat interface with AI capabilities. It showcases the integration of various technologies while maintaining a focus on user experience and performance. The modular architecture allows for easy maintenance and future enhancements.

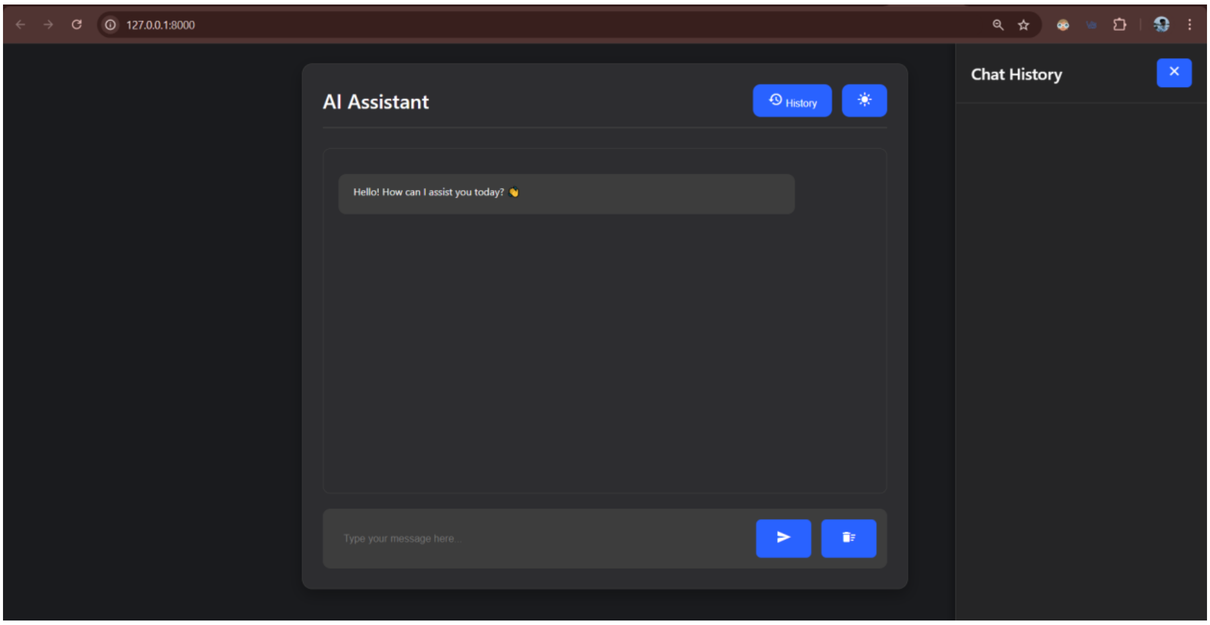
Working Example Screenshots-

1. Running Python code from VS code



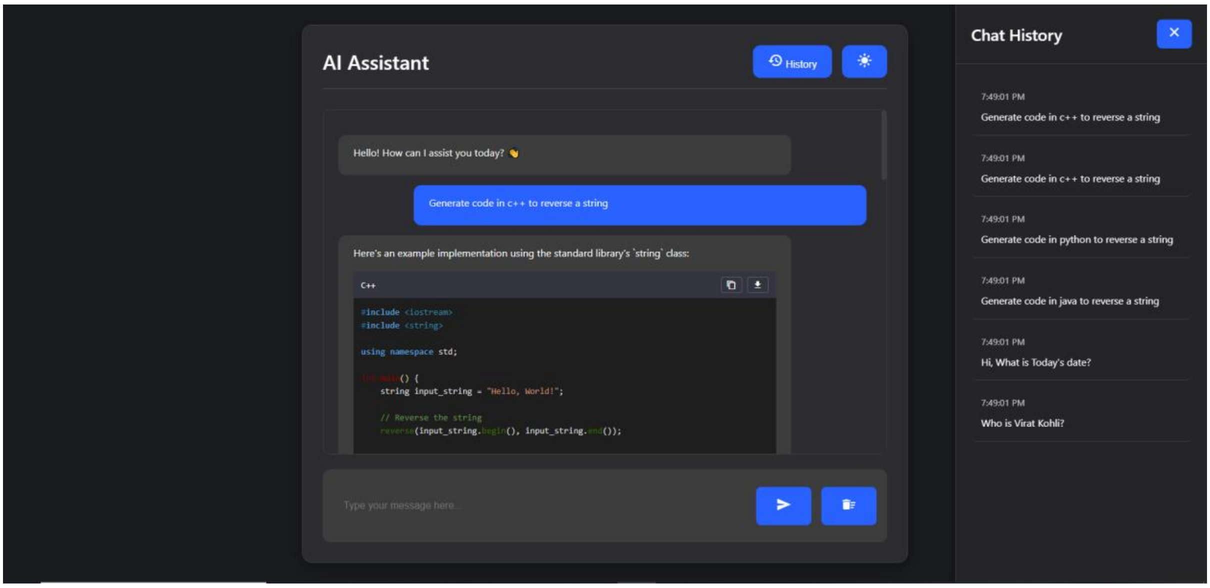
```
PS C:\Users\tanmay.agrawal\Desktop\Chatbot using llama> uvicorn app:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Users\\tanmay.agrawal\\Desktop\\Chatbot us
ing llama']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [68336] using StatReload
Device set to use privateuseone:0
Model loaded successfully!
Intel Iris Xe GPU is available!
Intel(R) UHD Graphics
INFO:     Started server process [109336]
Intel(R) UHD Graphics
INFO:     Started server process [109336]
INFO:     Started server process [109336]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:63779 - "GET / HTTP/1.1" 200 OK
INFO:     127.0.0.1:63779 - "GET /static/js/main.js HTTP/1.1" 200 OK
INFO:     127.0.0.1:63780 - "POST /chat HTTP/1.1" 200 OK
INFO:     127.0.0.1:63890 - "GET /chat-history HTTP/1.1" 200 OK
INFO:     127.0.0.1:64329 - "GET / HTTP/1.1" 200 OK
```

2. How the Frontend/ UI looks when page appear



Hello! How can I assist you today? 👋

3. Screenshots after giving various text or code prompts



Hello! How can I assist you today? 👋

Generate code in c++ to reverse a string

Here's an example implementation using the standard library's `string` class:

```
#include <iostream>
#include <string>

using namespace std;

int main() {
    string input_string = "Hello, World!";

    // Reverse the string
    reverse(input_string.begin(), input_string.end());
```

7:49:01 PM
Generate code in c++ to reverse a string

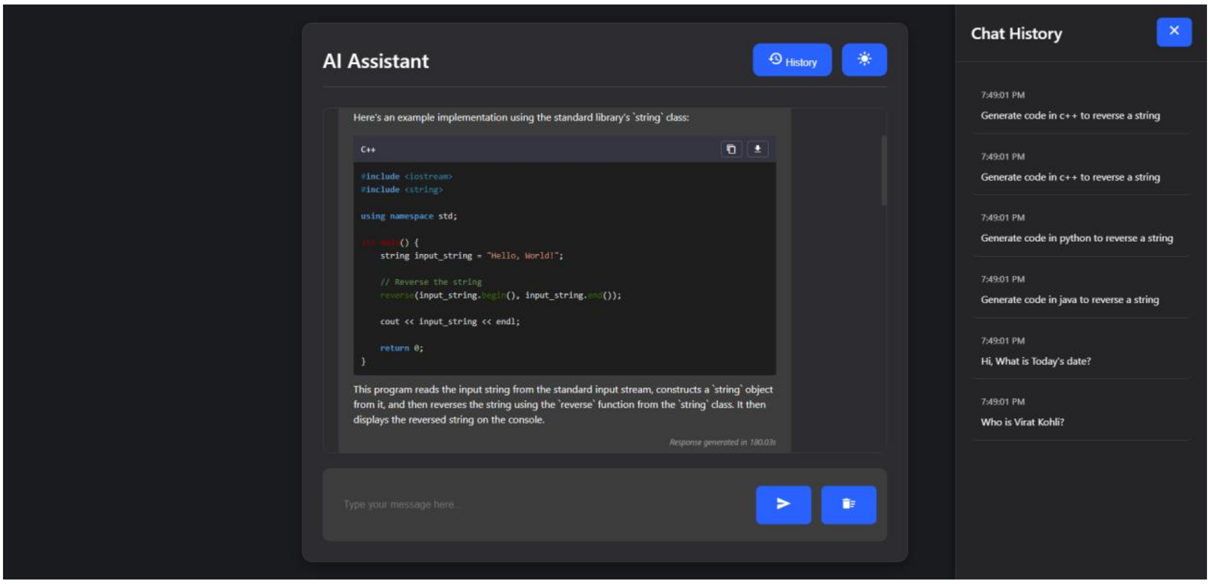
7:49:01 PM
Generate code in c++ to reverse a string

7:49:01 PM
Generate code in python to reverse a string

7:49:01 PM
Generate code in java to reverse a string

7:49:01 PM
Hi, What is Today's date?

7:49:01 PM
Who is Virat Kohli?



Here's an example implementation using the standard library's `string` class:

```
#include <iostream>
#include <string>

using namespace std;

int main() {
    string input_string = "Hello, World!";

    // Reverse the string
    reverse(input_string.begin(), input_string.end());

    cout << input_string << endl;

    return 0;
}
```

This program reads the input string from the standard input stream, constructs a `string` object from it, and then reverses the string using the `reverse` function from the `string` class. It then displays the reversed string on the console.

Response generated in 180.03s

7:49:01 PM
Generate code in c++ to reverse a string

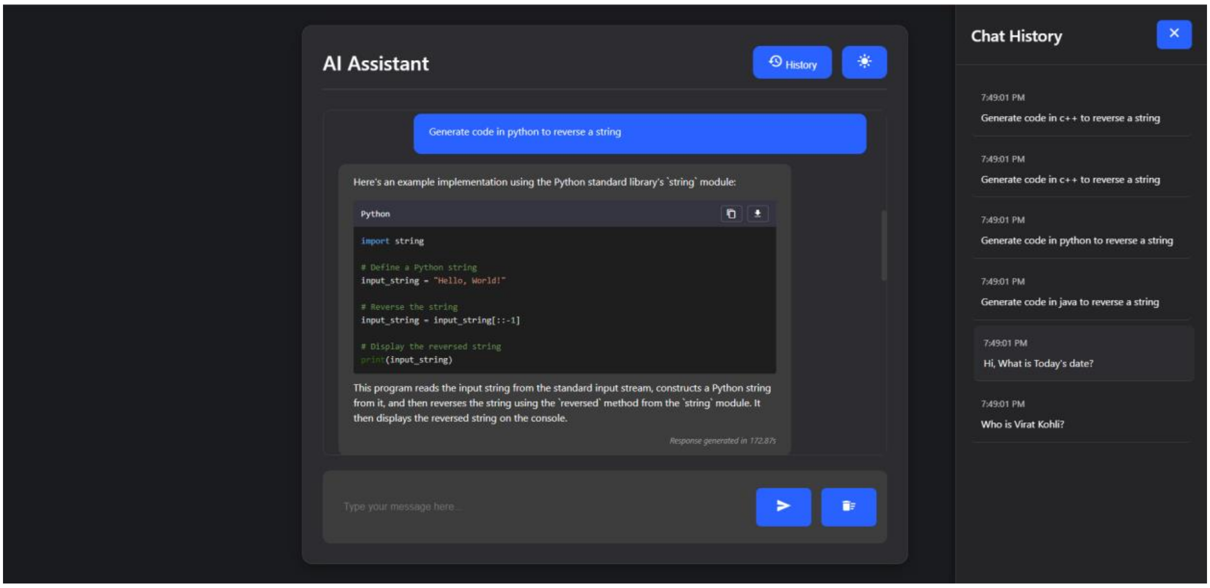
7:49:01 PM
Generate code in c++ to reverse a string

7:49:01 PM
Generate code in python to reverse a string

7:49:01 PM
Generate code in java to reverse a string

7:49:01 PM
Hi, What is Today's date?

7:49:01 PM
Who is Virat Kohli?



Generate code in python to reverse a string

Here's an example implementation using the Python standard library's `string` module:

```
import string

# Define a Python string
input_string = "Hello, World!"

# Reverse the string
input_string = input_string[::-1]

# Display the reversed string
print(input_string)
```

This program reads the input string from the standard input stream, constructs a Python string from it, and then reverses the string using the `reversed` method from the `string` module. It then displays the reversed string on the console.

Response generated in 172.87s

7:49:01 PM
Generate code in c++ to reverse a string

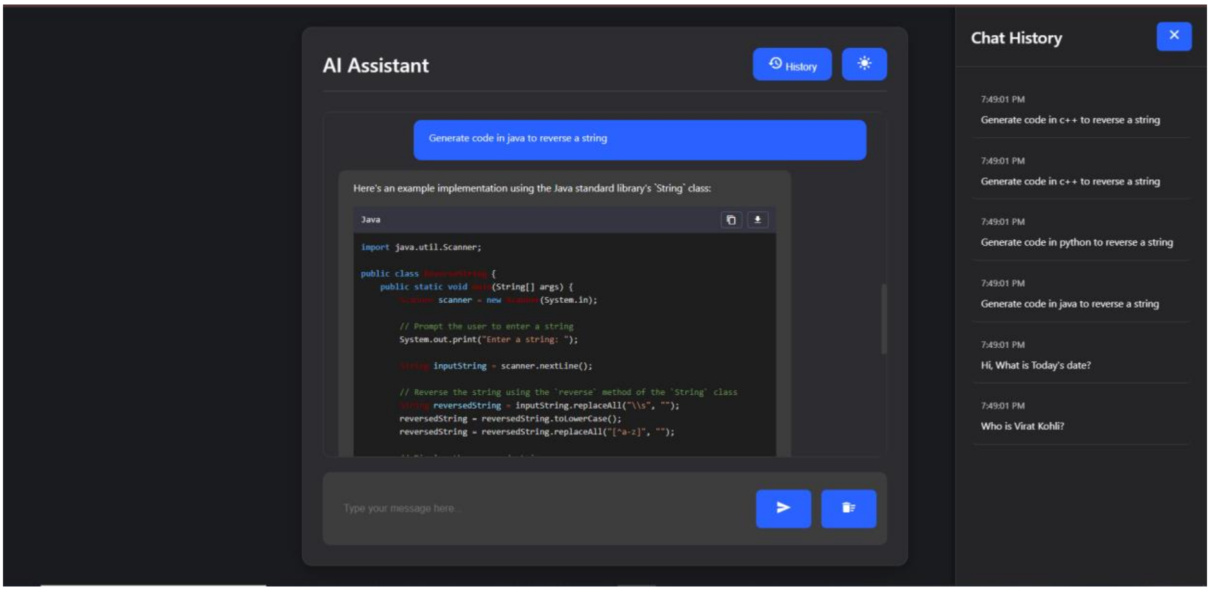
7:49:01 PM
Generate code in c++ to reverse a string

7:49:01 PM
Generate code in python to reverse a string

7:49:01 PM
Generate code in java to reverse a string

7:49:01 PM
Hi, What is Today's date?

7:49:01 PM
Who is Virat Kohli?



Generate code in java to reverse a string

Here's an example implementation using the Java standard library's `String` class:

Java

```
import java.util.Scanner;

public class ReverseString {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter a string
        System.out.print("Enter a string: ");

        String inputString = scanner.nextLine();

        // Reverse the string using the `reverse` method of the `String` class
        String reversedString = inputString.replaceAll("\\s", "");
        reversedString = reversedString.toLowerCase();
        reversedString = reversedString.replaceAll("[^a-z]", "");
```

Chat History

7:49:01 PM
Generate code in c++ to reverse a string

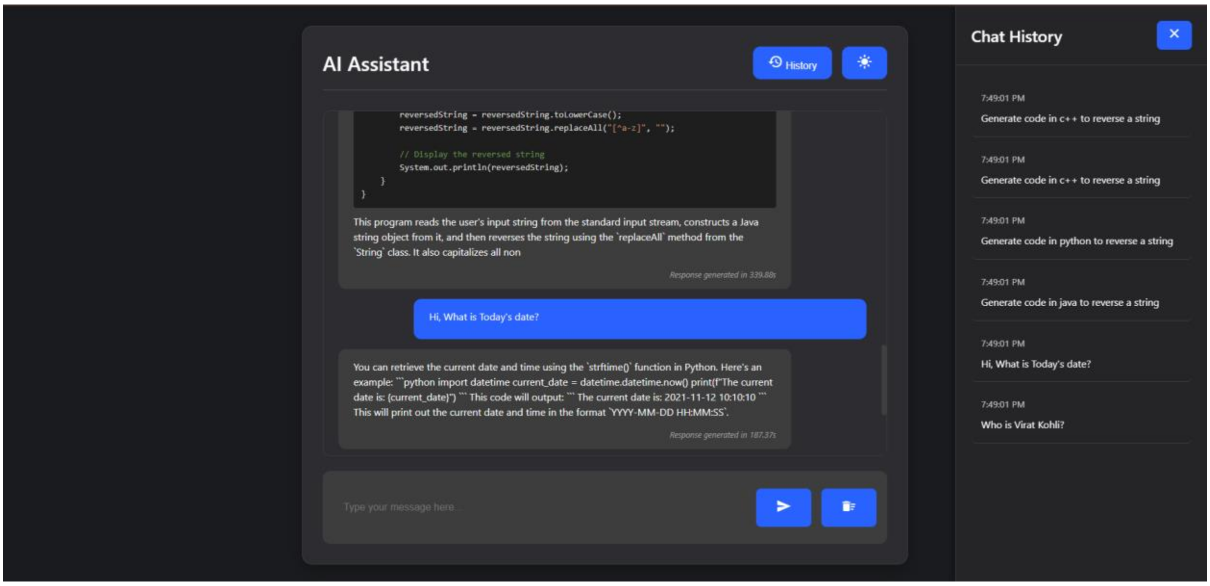
7:49:01 PM
Generate code in c++ to reverse a string

7:49:01 PM
Generate code in python to reverse a string

7:49:01 PM
Generate code in java to reverse a string

7:49:01 PM
Hi, What is Today's date?

7:49:01 PM
Who is Virat Kohli?



```
        reversedString = reversedString.toLowerCase();
        reversedString = reversedString.replaceAll("[^a-z]", "");

        // Display the reversed string
        System.out.println(reversedString);
    }
}
```

This program reads the user's input string from the standard input stream, constructs a Java string object from it, and then reverses the string using the `replaceAll` method from the `String` class. It also capitalizes all non

Response generated in 339.88s

Hi, What is Today's date?

You can retrieve the current date and time using the `strftime()` function in Python. Here's an example: ```python import datetime current_date = datetime.datetime.now() print(f"The current date is: {current_date}") ``` This code will output: ``` The current date is: 2021-11-12 10:10:10 ``` This will print out the current date and time in the format `YYYY-MM-DD HH:MM:SS`.

Response generated in 187.37s

Chat History

7:49:01 PM
Generate code in c++ to reverse a string

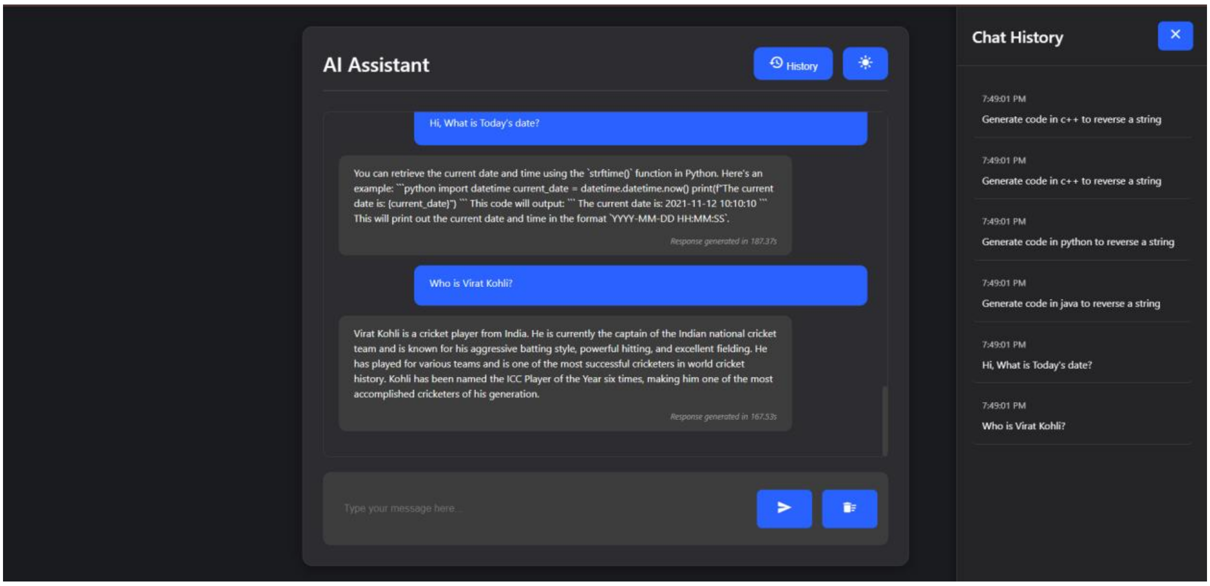
7:49:01 PM
Generate code in c++ to reverse a string

7:49:01 PM
Generate code in python to reverse a string

7:49:01 PM
Generate code in java to reverse a string

7:49:01 PM
Hi, What is Today's date?

7:49:01 PM
Who is Virat Kohli?



Hi, What is Today's date?

You can retrieve the current date and time using the `strftime()` function in Python. Here's an example: ```python import datetime current_date = datetime.datetime.now() print(f"The current date is: {current_date}") ``` This code will output: ``` The current date is: 2021-11-12 10:10:10 ``` This will print out the current date and time in the format `YYYY-MM-DD HH:MM:SS`.

Response generated in 187.37s

Who is Virat Kohli?

Virat Kohli is a cricket player from India. He is currently the captain of the Indian national cricket team and is known for his aggressive batting style, powerful hitting, and excellent fielding. He has played for various teams and is one of the most successful cricketers in world cricket history. Kohli has been named the ICC Player of the Year six times, making him one of the most accomplished cricketers of his generation.

Response generated in 167.53s

Chat History

7:49:01 PM
Generate code in c++ to reverse a string

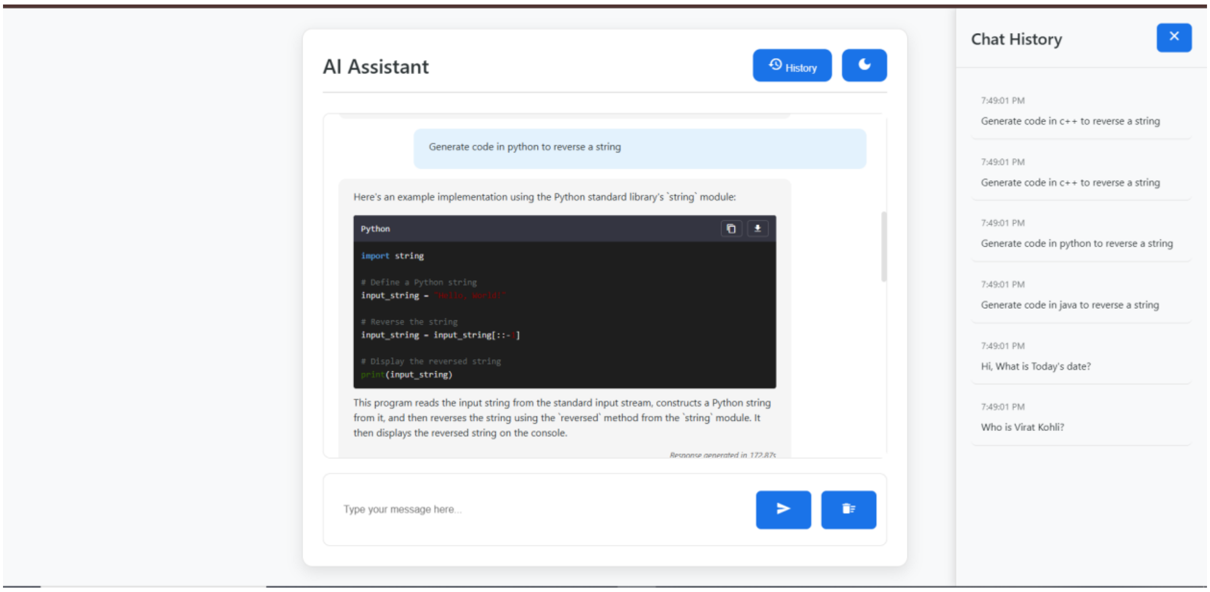
7:49:01 PM
Generate code in c++ to reverse a string

7:49:01 PM
Generate code in python to reverse a string

7:49:01 PM
Generate code in java to reverse a string

7:49:01 PM
Hi, What is Today's date?

7:49:01 PM
Who is Virat Kohli?



Generate code in python to reverse a string

Here's an example implementation using the Python standard library's `string` module:

```python
import string

# Define a Python string
input_string = "Hello, World!"

# Reverse the string
input_string = input_string[::-1]

# Display the reversed string
print(input_string)
```

This program reads the input string from the standard input stream, constructs a Python string from it, and then reverses the string using the `reversed` method from the `string` module. It then displays the reversed string on the console.

Response generated in 172.87s

7:49:01 PM
Generate code in c++ to reverse a string

7:49:01 PM
Generate code in c++ to reverse a string

7:49:01 PM
Generate code in python to reverse a string

7:49:01 PM
Generate code in java to reverse a string

7:49:01 PM
Hi, What is Today's date?

7:49:01 PM
Who is Virat Kohli?

- On clicking the download button, the code gets downloaded to our local device in the format it is generated (.py file for python , .cpp file for c++ , .java file for Java).
- On clicking copy button, only the code part gets copied to clipboard.
- Toggling theme, send button, show-chat-history and clear- history works perfectly.
- On clicking clear code, whole chat gets cleared and chat history is also cleared.