Project ID 33

A Project Report
On

FRIDAY – An AI Assistant

Submitted in the Partial fulfillment of the requirement for

awarding the degree of Bachelor of Computer Application (BCA)

SEM 5, Year 2025-26

By

Harsh S. Bheda
Sahli P. Bheda

Under the Guidance of
Prof. Bhargav Rajyagor

Faculty of Computer Application,
Noble BCA College Junagadh

NOBLE UNIVERSITY – JUNAGADH

Noble University, Junagadh

Certificate

Enrollment No.

[1] 230431032
[2] 230431033

This is to certify that the Project / practical work was satisfactorily carried out and hence submitted this report is the Bonafide work of Mr. Harsh S. Bheda, Sahil P. Bheda Student of Bachelor of Computer Applications Sem – 5 in the Laboratory of Noble University, Junagadh during the academic year 2025-26.

Internal Guide	**HOD**
Prof. Bhargav Rajyagor	**Prof. Bhargav Rajyagor**

Principal

Prof. Paresh Vora

Preface

This report contains all the outputs of a project undertaken to develop a working model of **AI Chat Assistant** and to study the existing working system. As part of this study, we analyzed the requirements of the working model and tried to improve it using the technological functionality that the current computer science provides.

This volume contains the documentation summary of the working system, the background study and analysis, the technical requirements and the working specification of the developed system.

We recommend the user to take the advantage of the study presented here and implement the system with their required improvisation. We hope this study will reduce the burden of the reanalysis of the development phase.

Acknowledgment

"It is not possible to prepare a project report without the assistance & encouragement of other people. This one is certainly no exception."

On the very outset of this report, We would like to extend our sincere & heartfelt obligation towards all the personages who have helped us in this endeavor. Without their active guidance, help, cooperation & encouragement, We would not have made headway in the project. We are ineffably indebted to our **Dean, Mr. Paresh Vora Sir** for his conscientious guidance and encouragement to accomplish this assignment.

We are extremely thankful and pay our gratitude to our project guide **Prof. Bhargav Rajyagor** his/her valuable guidance and support on completion of this project in its presently.

We extend our gratitude to Faculty of Computer Application, Noble University, Junagadh for giving us this opportunity.

Thanking You.

Harsh S. Bheda (230431032)
Sahil P. Bheda (230431033)

INDEX

A Project Report on: <u>FRIDAY – An AI Assistant</u>

1- Project Profile

Project on: FRIDAY - An AI Assistant

Developed By: Harsh S. Bheda, Sahil P. Bheda

Project Id: 33

Enrolment No: 220431032, 220431033

Branch: BCA (SEM-5)

Guided By: Prof. Bhargav Rajyagor

Technology: Python, Gemini API Key, Python Libraries, Firebase, HTML, CSS, JS

Platform: Windows, Linux, MAC

Submitted To: Faculty Of Computer Application, Noble University

Definition

FRIDAY - AI Assistant is a web-based conversational AI system that combines natural language processing with system automation capabilities. The application uses **Google's Gemini AI** model to provide intelligent responses while maintaining conversation context through **Firebase integration**. It features user authentication, real-time communication via **WebSockets**, and voice input capabilities, creating a JARVIS-like personal assistant experience.

Scope

Technical Scope:

- Multi-user authentication system using Firebase Authentication (email/password)
- Real-time bidirectional communication via WebSocket protocol on port 8765
- Natural language processing through Google Gemini 2.5 Flash API
- Cloud-based conversation storage in Firebase Firestore database
- Browser-based voice-to-text conversion using Web Speech API
- System command execution (open applications, play media, time/date)
- Context-aware conversation management maintaining last 10 messages
- Responsive web design supporting desktop and mobile devices

Functional Scope:

- User registration with email validation and password confirmation
- Secure login/logout with session management
- Send and receive messages in real-time with typing indicators
- Voice input for hands-free operation with visual recording feedback
- Execute system commands through natural language
- Browse complete chat history with search functionality
- Start new chat sessions while preserving history
- View real-time connection status (connected/disconnected)
- Access previous conversations from sidebar history panel

User Scope:

- Individual users requiring AI assistance for daily tasks
- Multi-user environments with separate accounts and privacy
- Users needing cross-device access to conversation history
- Organizations requiring collaborative AI assistance

Objective

Primary Objectives:

1. Web-Based Accessibility: Develop a browser-accessible AI assistant eliminating installation requirements and supporting cross-platform usage
2. Intelligent Conversations: Implement context-aware dialogues using Google Gemini 2.5 Flash that remember previous interactions
3. Real-Time Communication: Establish WebSocket-based instant messaging for seamless, low-latency user experience
4. Secure Multi-User Support: Provide Firebase-based authentication ensuring data privacy and personalized experiences
5. Persistent History: Store all conversations in Firestore enabling users to search and continue previous discussions
6. Voice Integration: Enable hands-free operation through browser's native Web Speech API
7. System Automation: Allow natural language commands to control local system functions

8. Modern UX: Create visually appealing interface with gradient designs, smooth animations, and responsive layouts

Secondary Objectives:

- Minimize response latency through asynchronous Python backend
- Ensure scalability for multiple concurrent WebSocket connections
- Provide intuitive history search and conversation management
- Handle network failures gracefully with automatic reconnection
- Support cross-browser compatibility (Chrome, Firefox, Safari)
- Implement stop functionality for long-running AI responses

2. System Analysis and Specification

2.1 Existing System

Previous voice assistants (including our earlier console-based version) relied on:

1. Command-Line Interface: Text-only console requiring technical knowledge
2. Local Execution Only: No remote access or cross-device continuity
3. No User Authentication: Single-user systems without personalization
4. No Conversation History: Chats lost when application closed
5. Stateless Interactions: Cannot remember previous context beyond single session
6. Desktop-Only: Platform-specific implementation (Windows SAPI, etc.)
7. No Cloud Integration: All data stored locally without backup

2.2 Limitations of Existing System

Technical Limitations:

1. Platform Dependency: Console-based systems tied to specific operating systems (Windows win32com, etc.)
2. No Persistence: Conversation history deleted on application closure
3. Single User: Cannot support multiple users with separate accounts
4. No Remote Access: Must run on same machine, no web accessibility
5. Limited Scalability: Cannot handle concurrent users efficiently

Functional Limitations:

1. No Context Memory: Basic voice assistants don't maintain dialogue history
2. Poor Error Recovery: Console applications crash without graceful handling
3. No Search Capability: Cannot find previous conversations
4. Synchronous Processing: Blocks during AI response generation
5. Manual Setup: Requires Python installation and dependency management

User Experience Limitations:

1. No Visual Feedback: Command-line lacks intuitive interface
2. Poor Accessibility: Difficult for non-technical users
3. No Mobile Support: Cannot access from phones or tablets

2.3 Feasibility Study

Technical Feasibility:

- Frontend Technologies: HTML5, CSS3, JavaScript ES6+ are mature and universally supported across modern browsers
- Backend Infrastructure: Python 3.x with websockets library provides robust asynchronous communication
- AI Integration: Google Gemini API offers 60 requests/minute free tier with reliable uptime
- Cloud Services: Firebase provides 50K document reads/day free tier suitable for moderate usage
- WebSocket Protocol: RFC 6455 standard supported by all major browsers
- Development Tools: VS Code, Chrome DevTools widely available and well-documented

Economic Feasibility:

- Zero Licensing Costs: All technologies are open-source or free-tier cloud services
- Minimal Infrastructure: Web hosting on standard HTTPS servers (Firebase Hosting, Netlify, Vercel)
- No App Store Fees: Web application avoids iOS/Android marketplace charges
- Low Maintenance: Cloud services handle scaling, backups, and security patches
- Estimated Monthly Cost: $0-5 for small user base on free tiers

Operational Feasibility:

- Universal Access: Works on any device with modern browser (95%+ market coverage)
- Zero Installation: Users access via URL without downloads
- Automatic Updates: Backend updates without user action
- Minimal Training: Intuitive chat interface familiar to all users
- 24/7 Availability: Cloud hosting ensures continuous uptime

Schedule Feasibility:

- Development Time: 4-6 weeks for core features (achievable within semester)
- Testing Phase: 1-2 weeks for integration and user testing
- Deployment: 1-2 days for production setup
- Total Timeline: Fits within academic semester constraints

2.4Need of New System

Critical Requirements:

1. **Cross-Platform Accessibility**
   - Users work across laptops, phones, tablets; need consistent access
   - Web-based solution eliminates OS compatibility issues

2. **Multi-User Support**
   - Families, teams, organizations need separate accounts
   - Firebase Authentication provides enterprise-grade security

3. **Persistent Conversation History**
   - Users want to reference past discussions weeks later
   - Cloud storage ensures data never lost even if device fails

4. **Context-Aware Intelligence**
   - Modern AI assistants must remember conversation flow
   - Gemini with context history enables natural dialogues

5. **Real-Time Responsiveness**
   - Command-line polling creates delays; WebSocket provides instant updates
   - Asynchronous backend prevents blocking during AI processing

6. **Professional User Interface**
   - Console text intimidates non-technical users
   - Modern gradient UI with animations improves engagement

7. **Remote Accessibility**
   - Users should access assistant from office, home, or mobile
   - Web deployment enables access from any location with internet

8. **Scalability**
   - System should support growth from 10 to 10,000 users
   - Asynchronous architecture handles concurrent connections efficiently

2.5 Front End & Back End Tools

Frontend Tools:

1. **HTML5**
   - Semantic markup for accessibility
   - Web Speech API integration for voice input
   - WebSocket API for real-time communication
   - LocalStorage API (note: not used per project constraints)

2. **CSS3**
   - Linear gradients for modern aesthetic
   - Flexbox/Grid for responsive layouts
   - CSS animations and transitions
   - Media queries for mobile optimization
   - Custom scrollbar styling

3. **JavaScript (ES6+)**
   - Async/await for promise handling
   - ES6 modules for code organization
   - Arrow functions and destructuring
   - Template literals for string interpolation

4. **Firebase SDK v10.7.1**
   - firebase-app: Core Firebase initialization
   - firebase-auth: User authentication
   - firebase-firestore: NoSQL database operations

5. **Web Speech API**
   - SpeechRecognition: Voice-to-text conversion
   - Continuous/non-continuous recognition modes
   - Language specification (en-US)

Backend Technology:

1. **Python 3.8+**
   - Primary server-side language
   - Asynchronous programming with asyncio
   - Type hints for code clarity

2. **websockets Library**
   - Asynchronous WebSocket server implementation
   - Ping/pong for connection health monitoring
   - Concurrent client handling with asyncio

3. **google-generativeai**
   - Official Gemini API Python client
   - Streaming and non-streaming response modes
   - Safety settings configuration

4. **Supporting Libraries**
   - json: Data serialization/deserialization
   - datetime: Timestamp handling
   - re: Regular expressions for command parsing
   - webbrowser: System browser control
   - subprocess: Application launching
   - platform: OS detection

3. System Requirement Specification

3.1 Proposed System and Advantages

System Architecture:

The Friday AI Assistant implements a three-tier client-server architecture:

1. Presentation Layer (Client-Side):

- Modern single-page web application (SPA)
- Responsive design using CSS Flexbox/Grid
- Real-time UI updates via WebSocket events
- Firebase SDK for authentication state management

2. Application Layer (Server-Side):

- Python WebSocket server on localhost:8765
- Asynchronous request handling with asyncio
- Command routing and processing logic
- Context management for conversation history

3. Data Layer (Cloud Services):

- Firebase Authentication for user management
- Cloud Firestore for conversation persistence
- Google Gemini API for AI processing

3.2 Development Strategy Model

1. Requirement Analysis

- **Goal:** To thoroughly understand and define what "Friday" needs to do, for whom, and under what conditions. This phase is continuous in an Agile setup, refining requirements over time.
- **Key Activities:**
  - o **Stakeholder Interviews & Workshops:** Engage end-users, business owners, customer service, and administrators to gather needs and expectations.
  - o **Use Case Definition:** Detail specific user interactions (like "Chat with Friday," "Train NLP Model" - as identified in the Use Case Diagram).
  - o **User Stories & Backlog Creation:** Break down large requirements into smaller, manageable user stories (e.g., "As a user, I want Friday to greet me politely"). These form the initial Product Backlog.
  - o **Data Requirements:** Identify types of data Friday will handle (user input, conversation history, knowledge base articles, training data for NLP).
  - o **Functional Requirements:** What Friday must do (e.g., intent recognition, entity extraction, context handling, external API integration).
  - o **Non-Functional Requirements:** How Friday must perform (e.g., response time, scalability, security, usability).

2. Design

- **Goal:** To plan how "Friday" will be built, detailing its architecture, components, data structures, and user interface. This phase is also iterative; designs evolve with each sprint.
- **Key Activities:**
    - **System Architecture Design:** Define the high-level components of Friday (e.g., Chat Interface, Dialogue Manager, NLP Processor, Knowledge Base, Integrator) and their interactions (as shown in DFDs, Class Diagram).
    - **Database Design:** Develop the schema for storing user data, conversation history, intents, responses, and entities (as shown in the ER Diagram).
    - **NLP Model Design:** Choose NLP frameworks, define intent/entity structures, plan for training data formats.
    - **Dialogue Flow Design:** Map out conversational paths, decision points, and error handling for key use cases.
    - **API Design:** Define interfaces for communication between internal components and external services.
    - **UI/UX Design:** Create wireframes, mockups, and prototypes for the chatbot interface, ensuring intuitiveness and user-friendliness.
    - **Security Design:** Incorporate security measures (e.g., authentication, authorization, data encryption, input validation) into the architecture.

3. Development

- **Goal:** To build and implement the "Friday" AI Chatbot system based on the defined requirements and designs. This is the core "coding" phase, conducted in iterative sprints.
- **Key Activities (within Sprints):**
    - **Backend Development:**
        - Implement the core Dialogue Manager logic.
        - Develop API endpoints for user interaction and internal services.
        - Integrate with databases for data storage and retrieval.
        - Build connectors for external services (e.g., weather API, CRM).
    - **NLP Model Development & Training:**
        - Implement intent classification and entity extraction models using chosen NLP frameworks.
        - Gather, clean, and annotate training data.
        - Iteratively train, evaluate, and refine the NLP models.
    - **Frontend Development:**
        - Develop the user-facing chatbot interface (web widget, mobile component).
        - Implement real-time messaging functionalities.
    - **Infrastructure Setup (DevOps/MLOps):**
        - Set up development, staging, and production environments.
        - Implement version control for code, data, and models.
        - Configure CI/CD pipelines for automated builds and deployments.
        - Code Reviews: Conduct regular peer reviews to ensure code quality and adherence to standards.

4. Testing

- **Goal:** To systematically verify that "Friday" meets its requirements, is free of defects, and performs reliably. This phase is integrated throughout development in an Agile model.
- **Key Activities:**
    - **Unit Testing:** Test individual functions, methods, and classes to ensure they work in isolation.
    - **Integration Testing:** Verify communication and data flow between different modules and components (e.g., Dialogue Manager with NLP Processor, chatbot with external API).
    - **Conversational Testing:**
        - **Functional Testing:** Execute test cases covering core interactions, intent recognition, entity extraction, context handling, fallback mechanisms (as defined in our test cases).
        - **Regression Testing:** Ensure new changes don't break existing functionalities.
        - **Edge Case Testing:** Test unusual inputs, complex multi-turn scenarios, and error conditions.
    - **Performance Testing:** Assess response times, throughput, and scalability under various load conditions.
    - **Security Testing:** Conduct vulnerability scans, penetration testing, and review against security best practices.
    - **User Acceptance Testing (UAT):** Involve actual end-users and stakeholders to validate that Friday meets business needs and is intuitive to use in a real-world setting.
    - **Model Evaluation:** Rigorously evaluate NLP model accuracy, precision, and recall using held-out test datasets.

5. Deployment

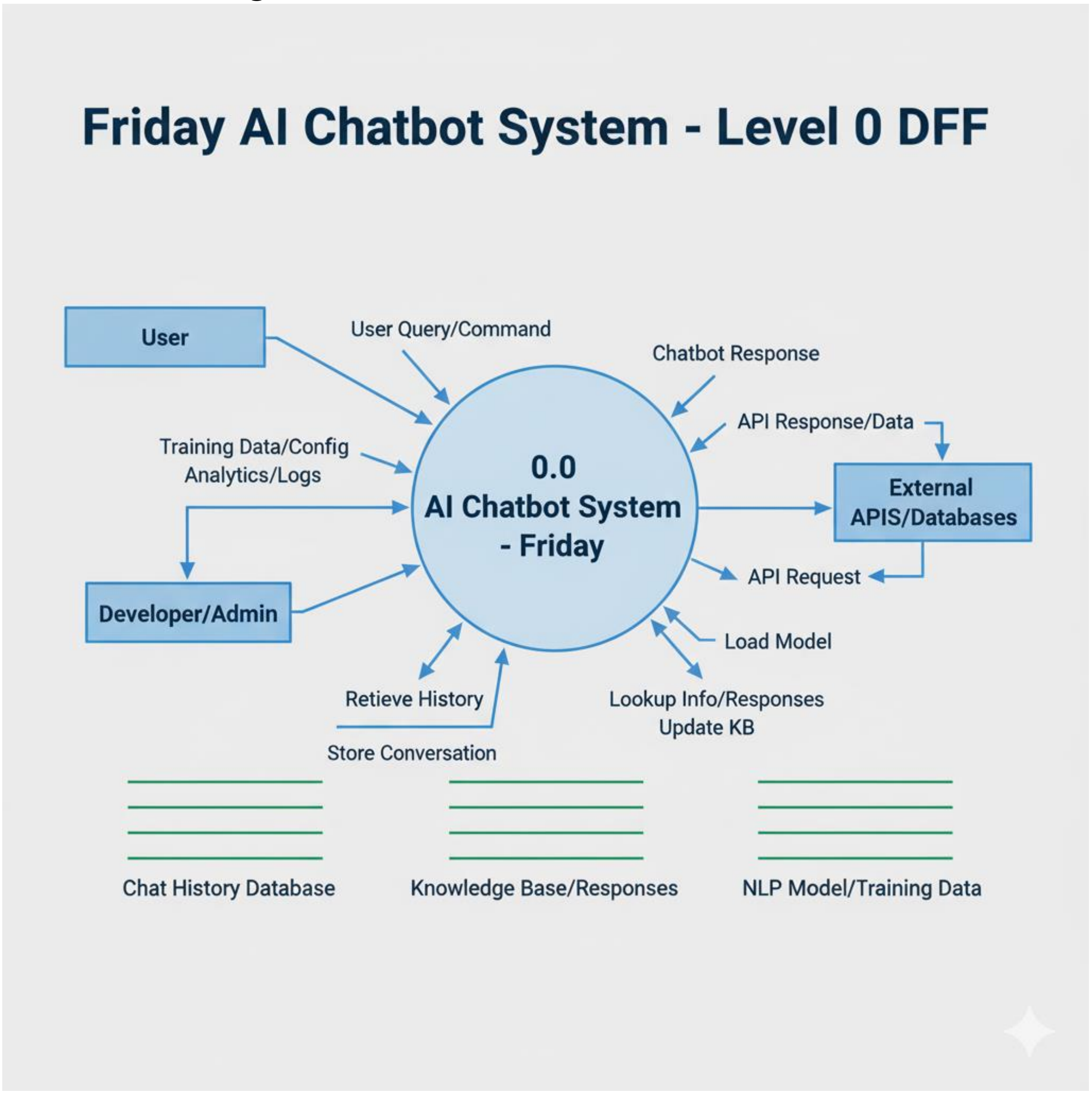
- **Goal:** To release the "Friday" AI Chatbot system into a live production environment, making it available to end-users. This is automated via CI/CD.
- **Key Activities:**
    - **Environment Preparation:** Configure production servers, databases, and cloud services for optimal performance and security.
    - **Automated Deployment:** Utilize CI/CD pipelines to deploy the latest tested code and trained models to the production environment.
    - **Configuration Management:** Manage environment-specific settings (API keys, database connections).
    - **Smoke Testing:** Perform quick, high-level tests immediately after deployment to ensure critical functionalities are working.
    - **Rollback Strategy:** Establish procedures to revert to a previous stable version in case of critical issues.
    - **Documentation:** Update deployment guides and operational procedures.

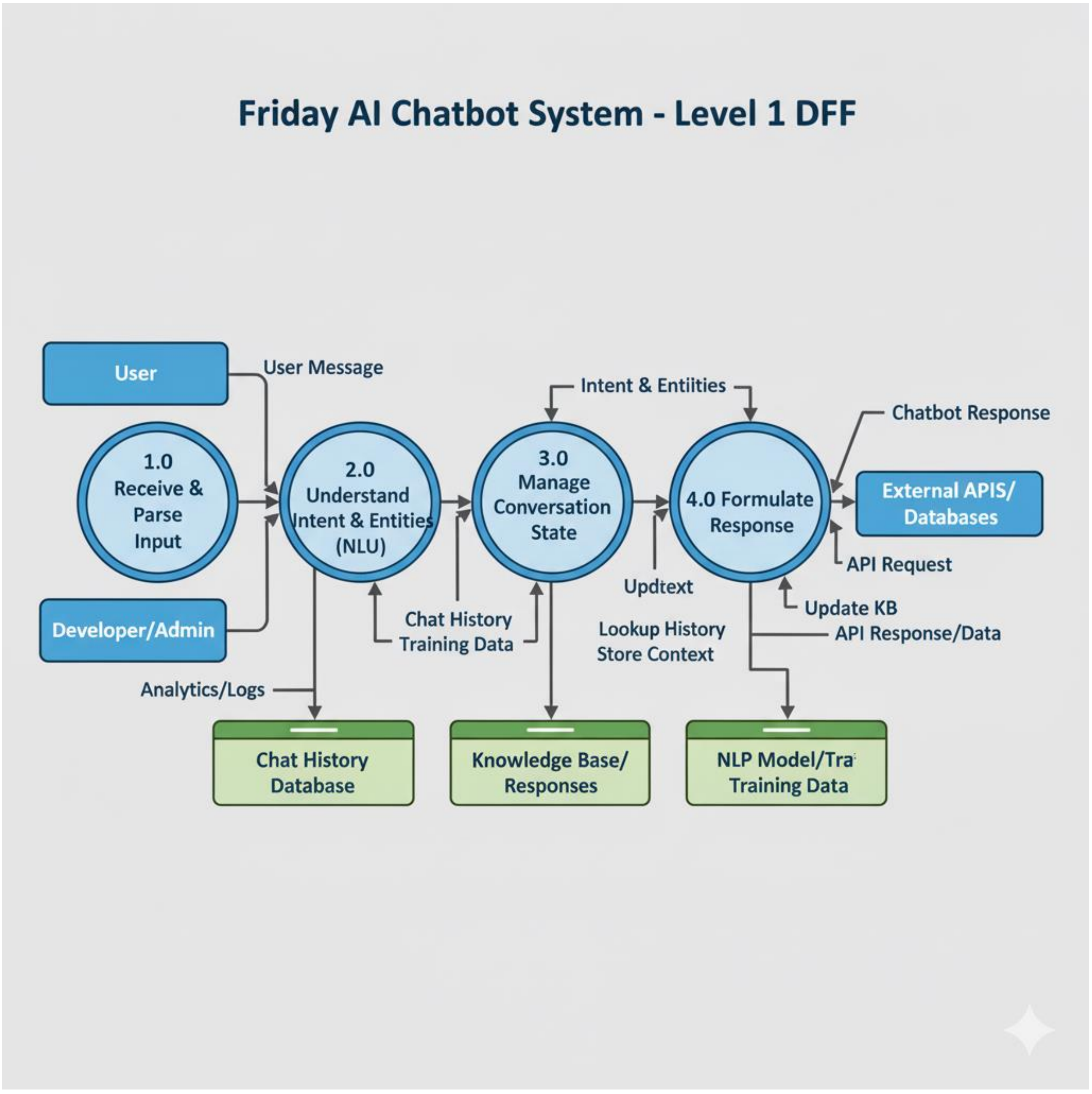
6. Maintenance

- **Goal:** To ensure "Friday" remains operational, accurate, secure, and continues to evolve and improve over its lifespan. This is an ongoing, continuous process.
- **Key Activities:**
    - **Monitoring & Alerting:** Continuously monitor system health, performance metrics, and chatbot accuracy (e.g., intent confidence, fallback rates). Set up alerts for anomalies.
    - **Bug Fixing & Incident Management:** Address and resolve any issues or bugs that arise in the production environment.
    - **Model Retraining & Improvement:**
        - Collect new user interaction data (conversation logs, feedback).
        - Annotate new data to improve intent/entity recognition.
        - Retrain and evaluate NLP models periodically or as new data accumulates.
        - Deploy updated models via the MLOps pipeline.
    - **Feature Enhancements:** Implement new functionalities, integrations, or improve existing conversational flows based on user feedback, analytics, and evolving business needs.
    - **Performance Optimization:** Ongoing tuning of database queries, code, and infrastructure to maintain optimal performance.
    - **Security Updates:** Apply security patches, update libraries, and conduct regular security audits.
    - Documentation Updates: Maintain up-to-date documentation for the system, including API specifications, architecture diagrams, and operational guides.

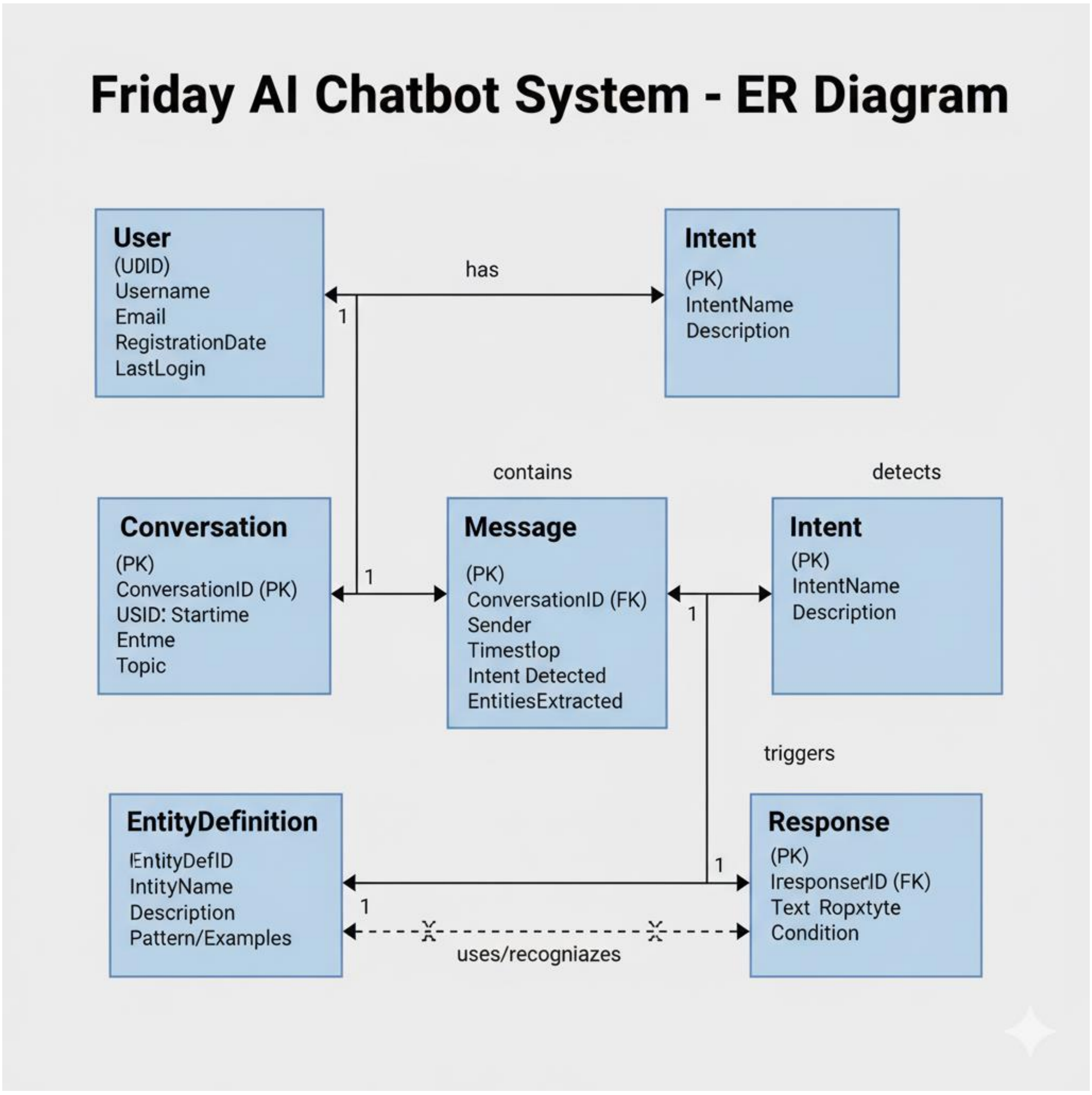
4. System Design

4.1 Data Flow Diagram

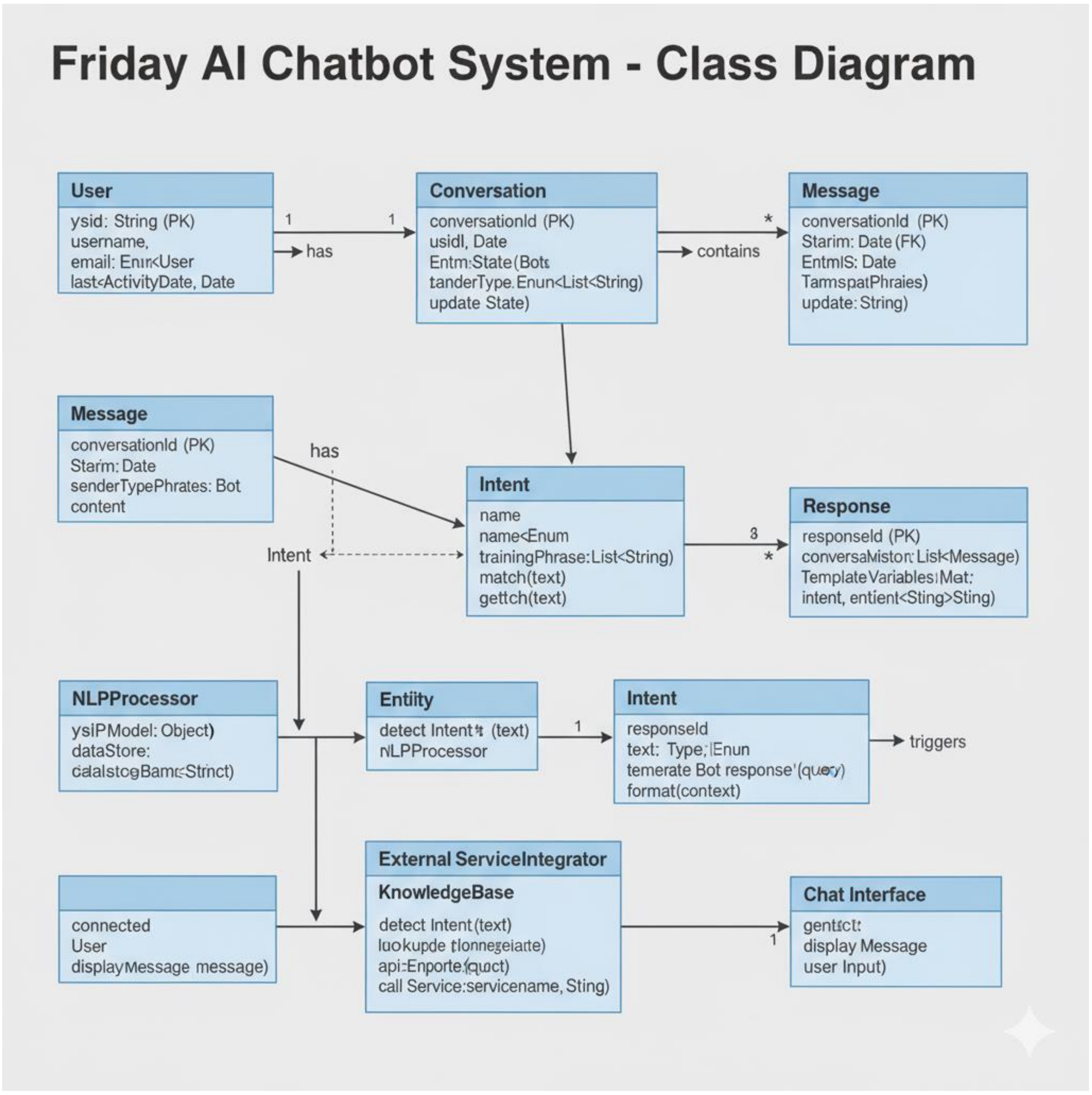



Friday AI Chatbot System - Level 1 DFF
User
User Message
Intent & Entiities
Chatbot Response
1.0 Receive & Parse Input
2.0 Understand Intent & Entities (NLU)
3.0 Manage Conversation State
4.0 Formulate Response
External APIS/ Databases
API Request
Updtext
Update KB
Developer/Admin
Chat History Training Data
Lookup History Store Context
API Response/Data
Analytics/Logs
Chat History Database
Knowledge Base/ Responses
NLP Model/Tra Training Data

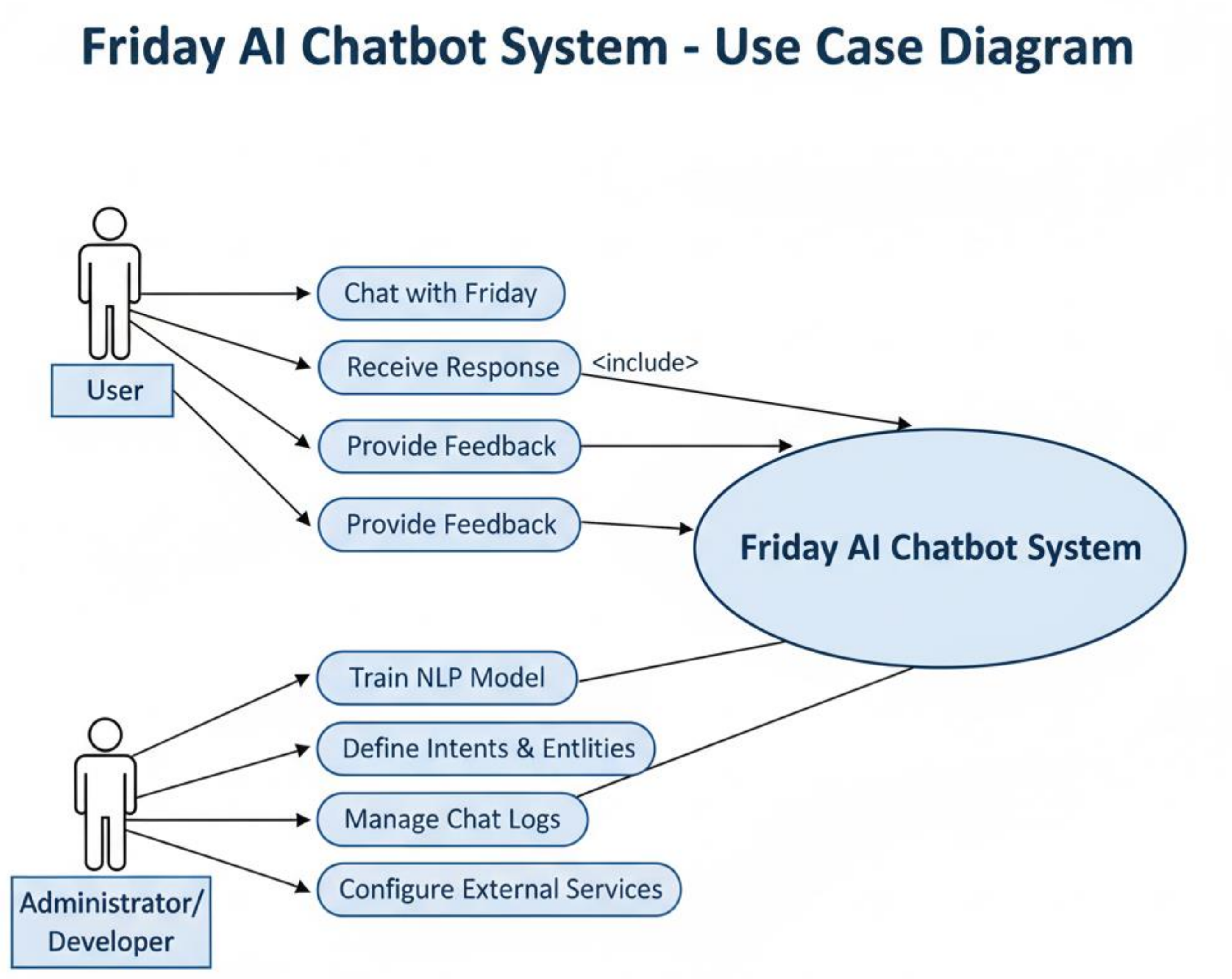
4.2 E-R Diagram



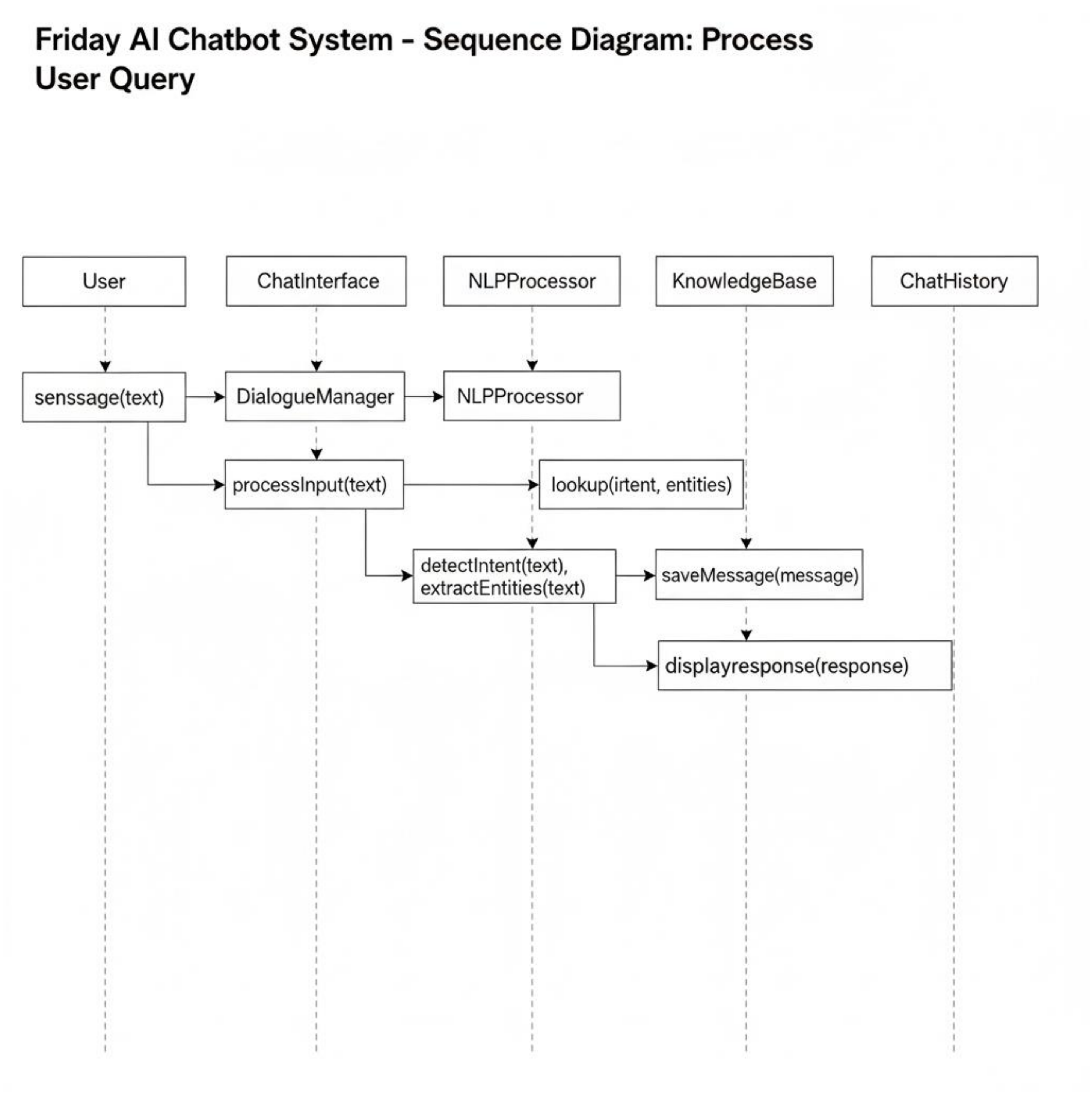
4.3 Class Diagram



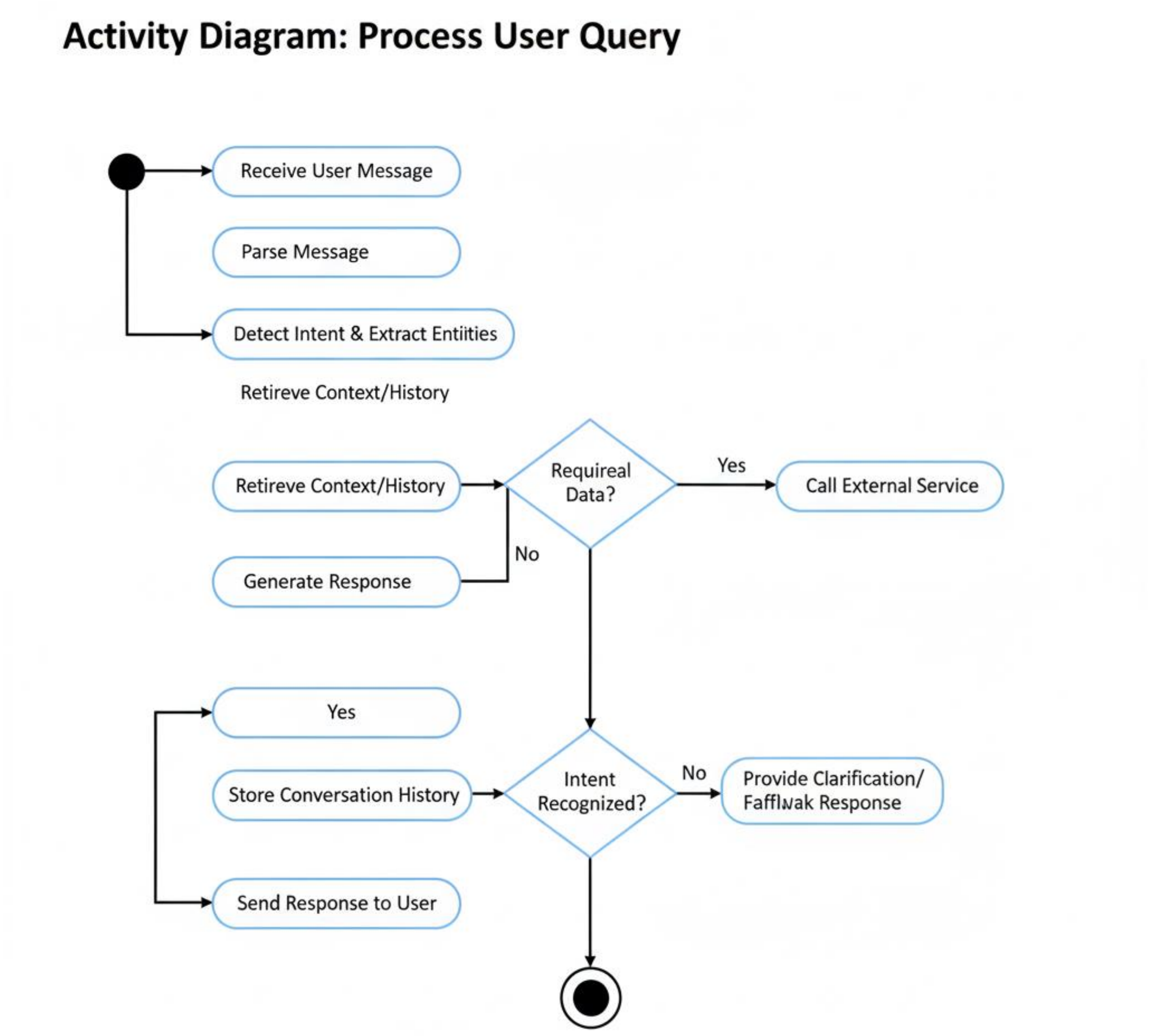
4.4 Use Case Diagram



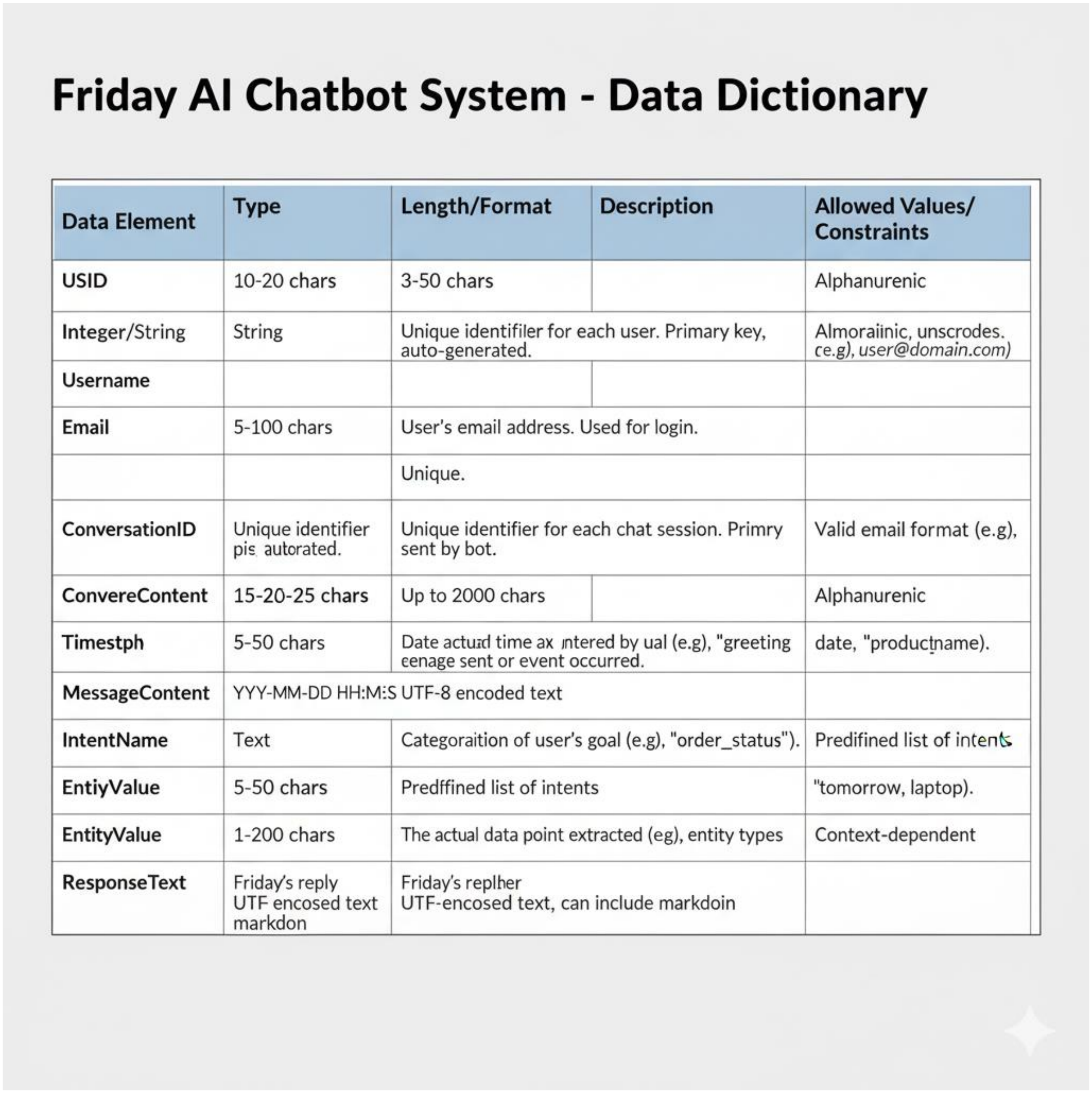
4.5 Sequence Diagram



4.6 Activity Diagram



4.7 Data Dictionary Diagram



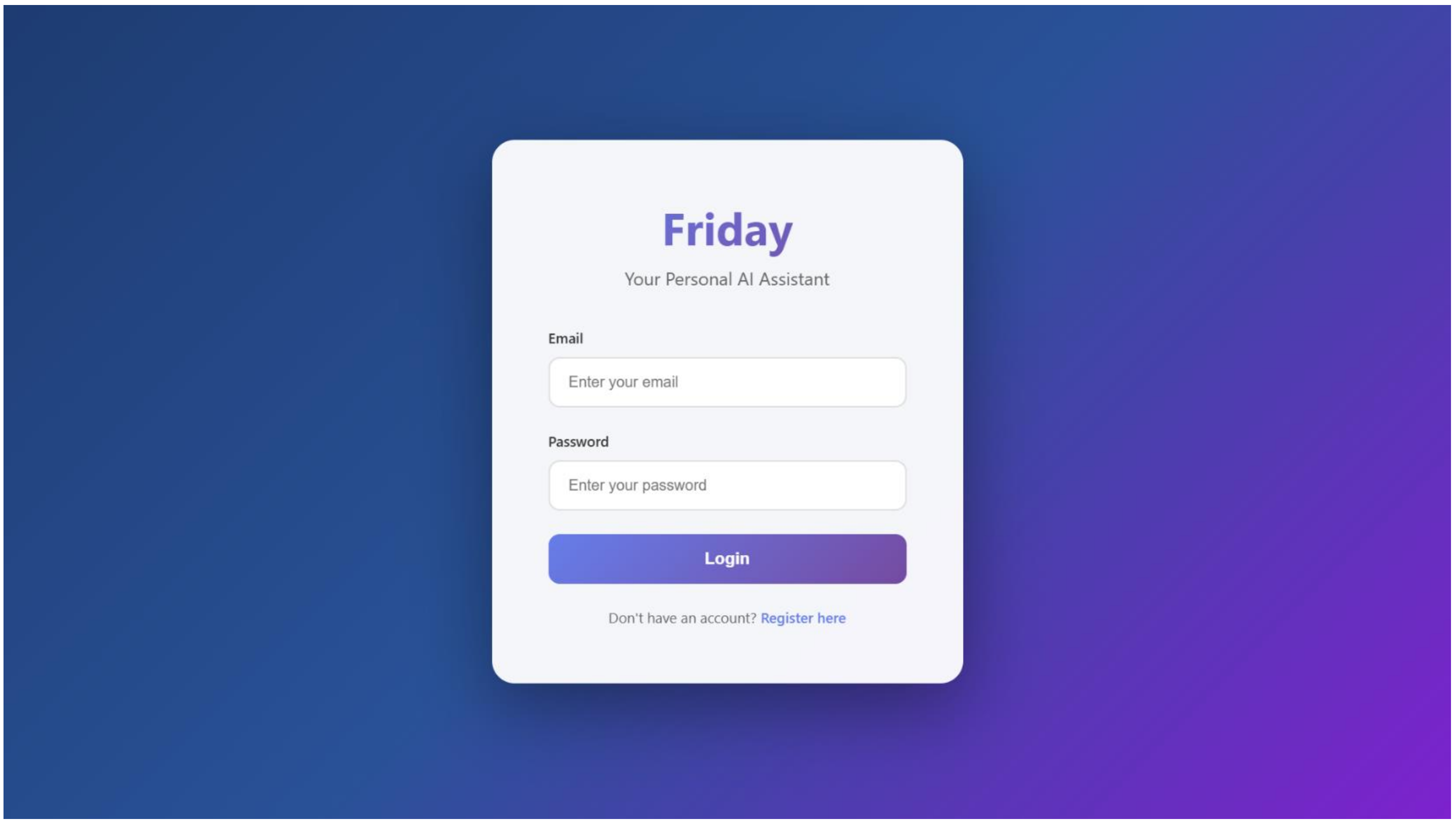
Friday AI Chatbot System - Data Dictionary

Data Element	Type	Length/Format	Description	Allowed Values/ Constraints
USID	10-20 chars	3-50 chars		Alphanurenic
Integer/String	String	Unique identifiler for each user. Primary key, auto-generated.		Almoraiinic, unscrodes. *ce.g), user@domain.com)*
Username				
Email	5-100 chars	User's email address. Used for login.		
		Unique.		
ConversationID	Unique identifier pis autorated.	Unique identifier for each chat session. Primry sent by bot.		Valid email format (e.g),
ConvereContent	15-20-25 chars	Up to 2000 chars		Alphanurenic
Timestph	5-50 chars	Date actuad time ax ıntered by ual (e.g), "greeting eenage sent or event occurred.		date, "productname).
MessageContent	YYY-MM-DD HH:M:S	UTF-8 encoded text		
IntentName	Text	Categoraition of user's goal (e.g), "order_status").		Predifined list of intents
EntiyValue	5-50 chars	Predffined list of intents		"tomorrow, laptop).
EntityValue	1-200 chars	The actual data point extracted (eg), entity types		Context-dependent
ResponseText	Friday's reply UTF encosed text markdon	Friday's replher UTF-encosed text, can include markdoin		

5. Screen Design

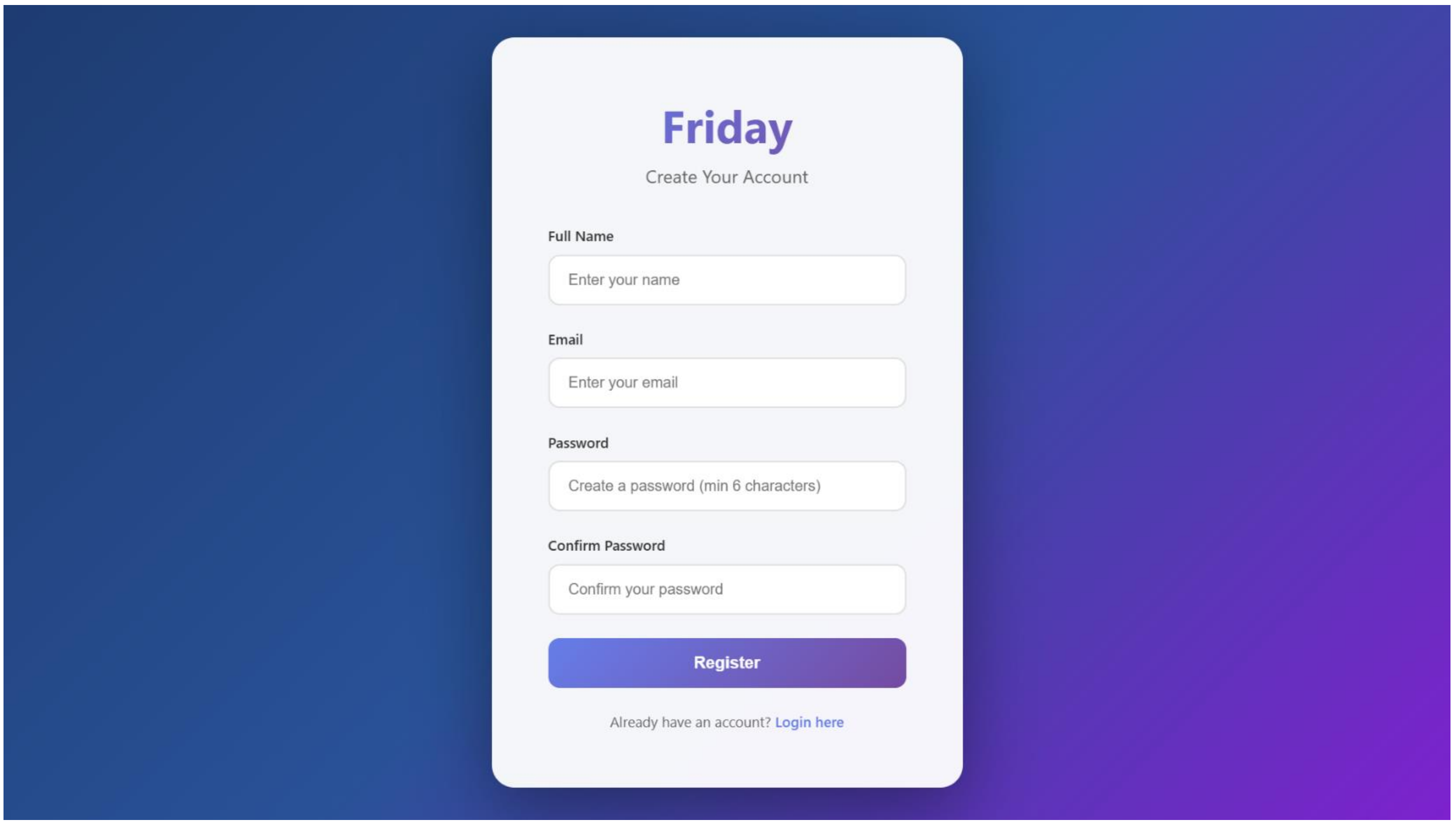
Login Page:

This screen allows existing users to securely access their Friday account. The user must enter a valid email and password to log in. The page includes validation and error messages for incorrect credentials. It is connected to Firebase Authentication for secure sign-in.


Friday
Your Personal AI Assistant
Email
Enter your email
Password
Enter your password
Login
Don't have an account? Register here

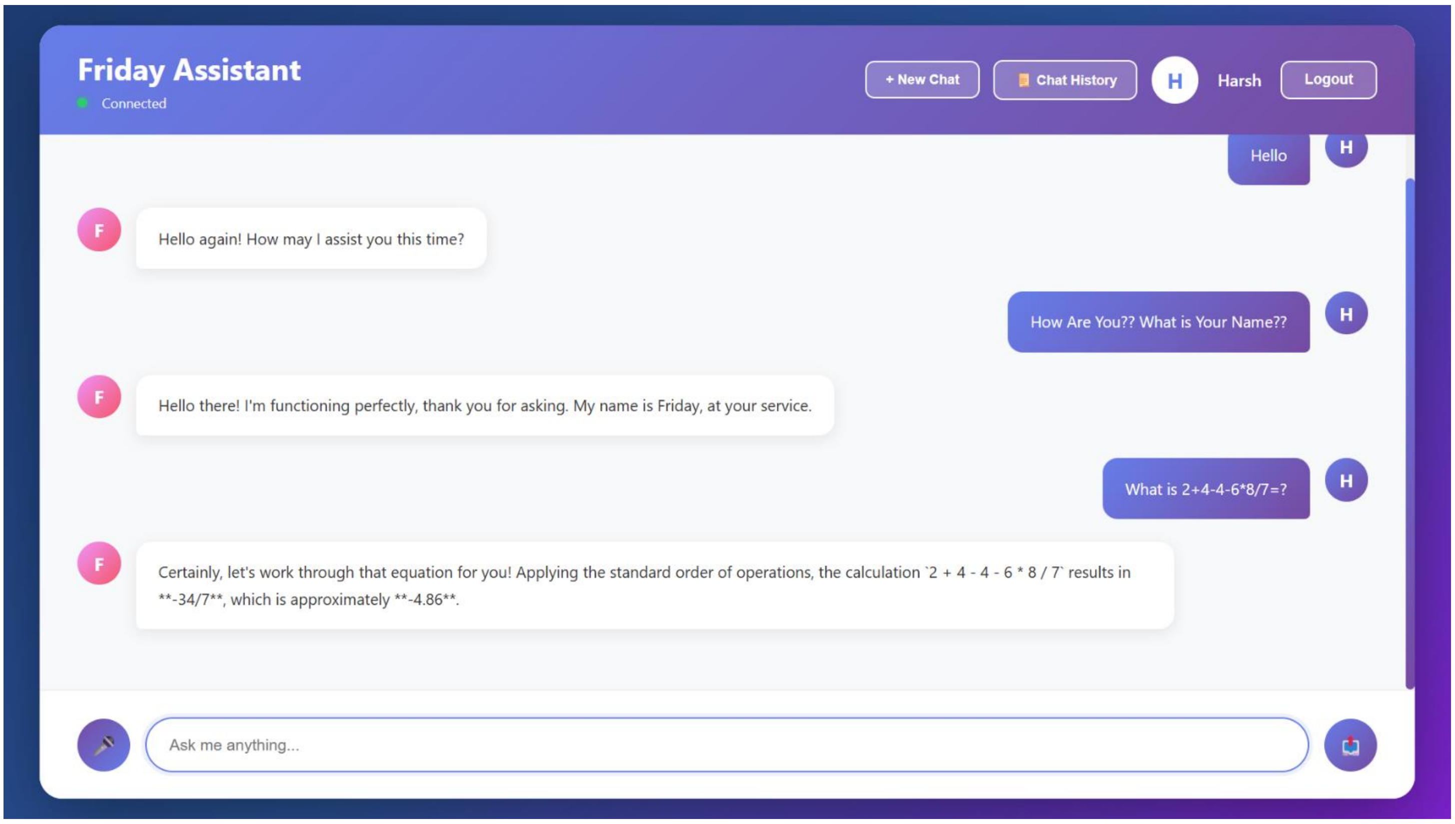
Register Page:

The registration interface enables new users to create an account by providing their name, email, and password. It includes password confirmation and validation for secure sign-up. Successful registration automatically stores the user's information in Firebase Authentication and updates the display name.



Chat Interface

This is the main interaction area between the user and the Friday chatbot. After successful login, the chat screen is displayed. The user can type or use voice input (via microphone button) to communicate. The assistant's responses appear in the chat window dynamically using WebSocket communication. Each conversation is saved to Firebase Firestore.



Hello

Hello again! How may I assist you this time?

How Are You?? What is Your Name??

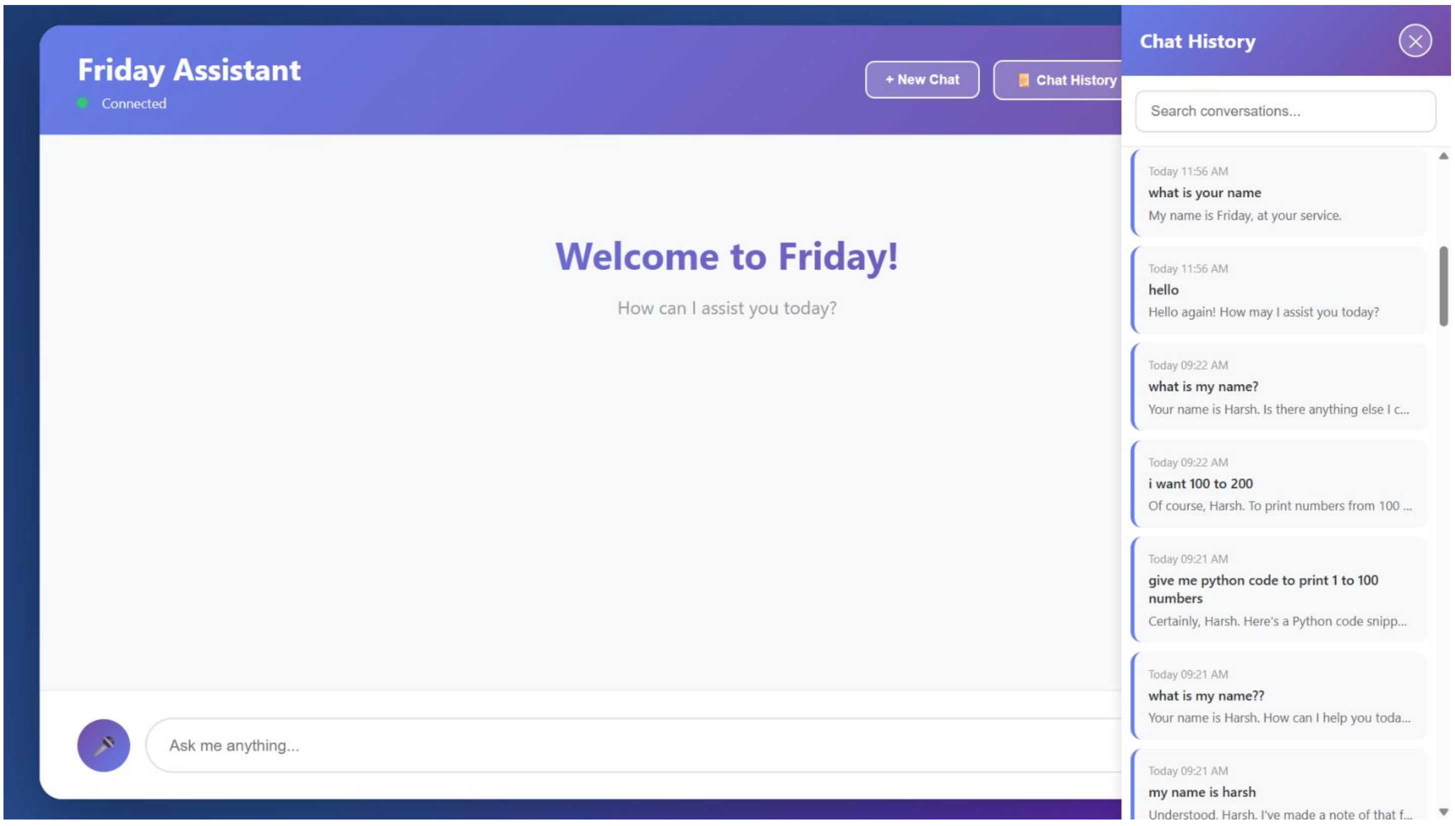
Hello there! I'm functioning perfectly, thank you for asking. My name is Friday, at your service.

What is 2+4-4-6*8/7=?

Certainly, let's work through that equation for you! Applying the standard order of operations, the calculation `2 + 4 - 4 - 6 * 8 / 7` results in **-34/7**, which is approximately **-4.86**.

Chat History Section

This section displays the user's previous conversations stored in the database. When a user logs in, the chat history is automatically fetched and shown in reverse chronological order. It allows the user to view recent interactions and continue the chat seamlessly.



Today 11:56 AM
what is your name
My name is Friday, at your service.

Today 11:56 AM
hello
Hello again! How may I assist you today?

Today 09:22 AM
what is my name?
Your name is Harsh. Is there anything else I c...

Today 09:22 AM
i want 100 to 200
Of course, Harsh. To print numbers from 100 ...

Today 09:21 AM
give me python code to print 1 to 100 numbers
Certainly, Harsh. Here's a Python code snipp...

Today 09:21 AM
what is my name??
Your name is Harsh. How can I help you toda...

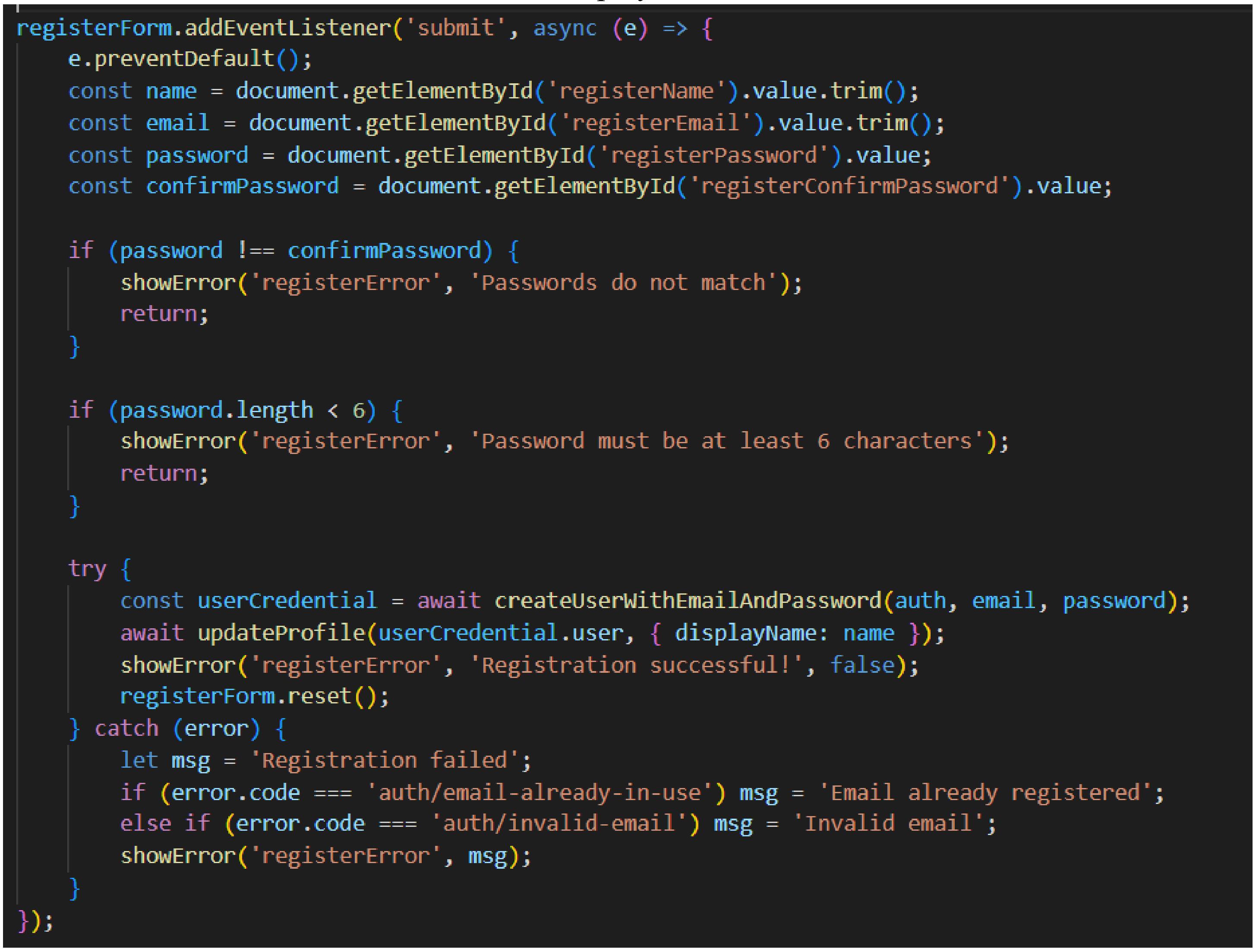
Today 09:21 AM
my name is harsh
Understood, Harsh. I've made a note of that f...

6. Coding (Few Code Snippets)

Core Functionality:

1. **User Registration Handler:**

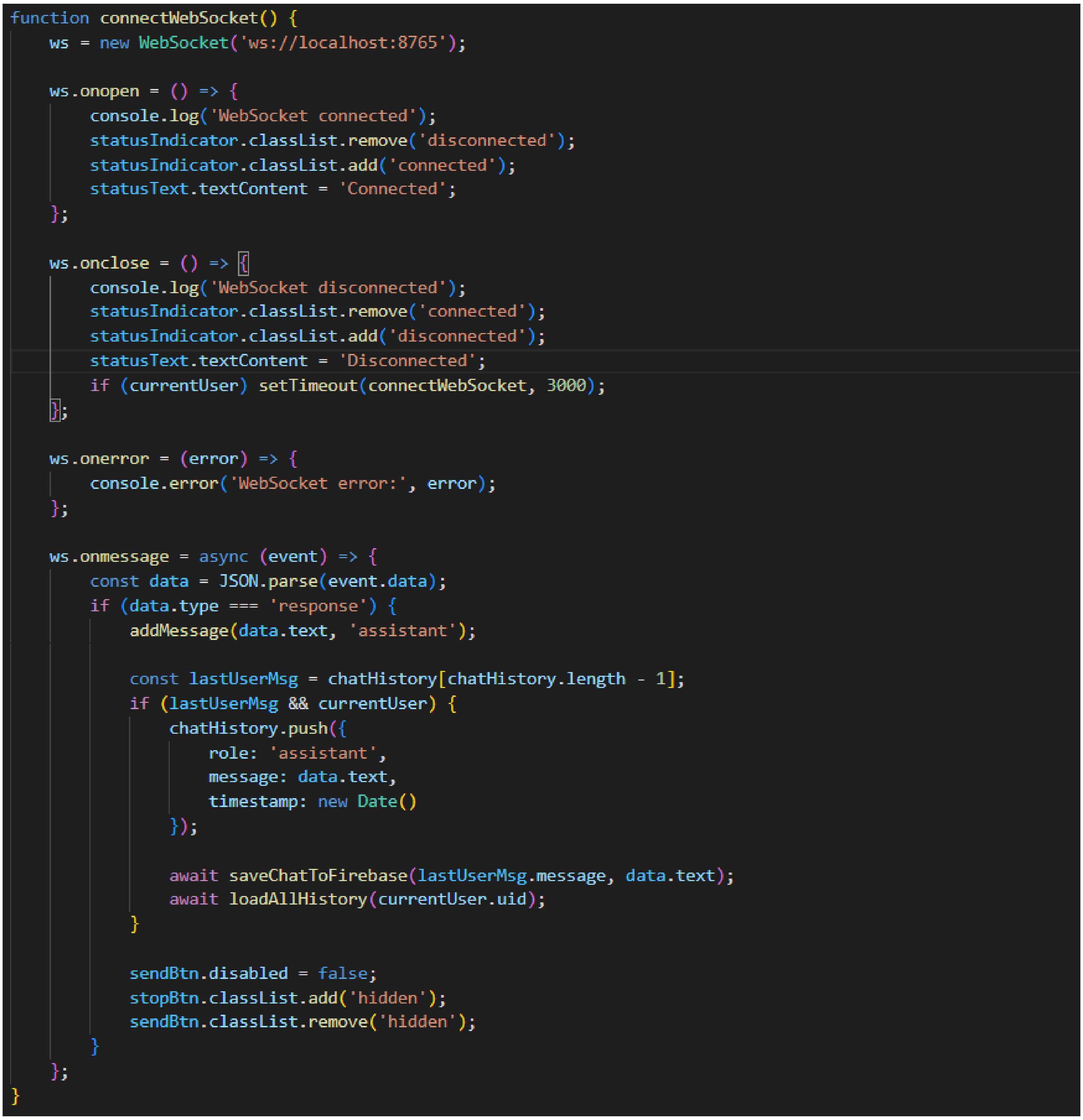
   This JavaScript code handles the user registration form submission. It validates the input, checks for password matching and length, and uses Firebase Authentication to create a new user account with a display name.



```
registerForm.addEventListener('submit', async (e) => {
    e.preventDefault();
    const name = document.getElementById('registerName').value.trim();
    const email = document.getElementById('registerEmail').value.trim();
    const password = document.getElementById('registerPassword').value;
    const confirmPassword = document.getElementById('registerConfirmPassword').value;

    if (password !== confirmPassword) {
        showError('registerError', 'Passwords do not match');
        return;
    }

    if (password.length < 6) {
        showError('registerError', 'Password must be at least 6 characters');
        return;
    }

    try {
        const userCredential = await createUserWithEmailAndPassword(auth, email, password);
        await updateProfile(userCredential.user, { displayName: name });
        showError('registerError', 'Registration successful!', false);
        registerForm.reset();
    } catch (error) {
        let msg = 'Registration failed';
        if (error.code === 'auth/email-already-in-use') msg = 'Email already registered';
        else if (error.code === 'auth/invalid-email') msg = 'Invalid email';
        showError('registerError', msg);
    }
});
```

2. **WebSocket Connection Management:**

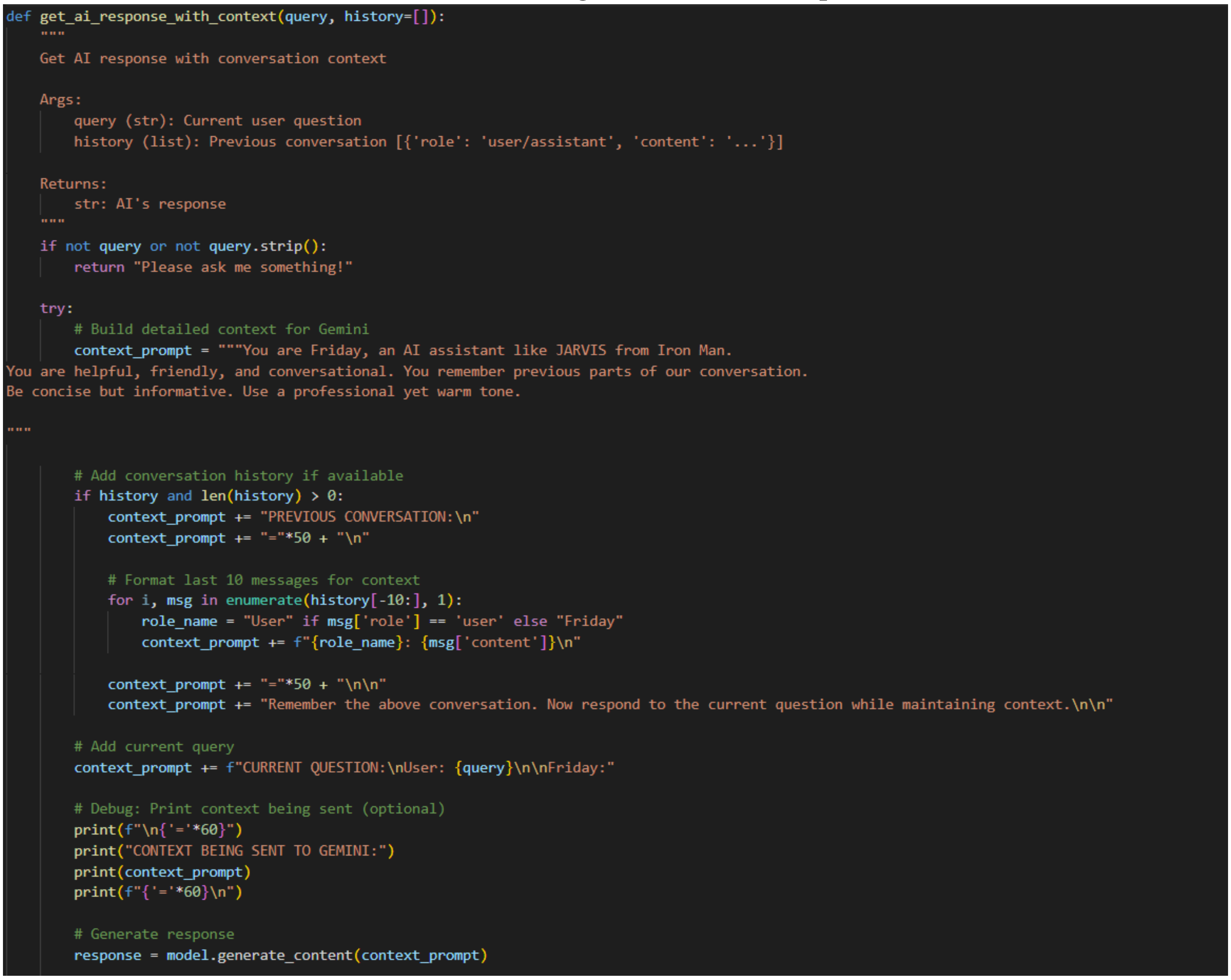
This function establishes a WebSocket connection to a backend server. It manages connection status, automatically reconnects when disconnected, and processes incoming AI responses by displaying them, updating chat history, and saving conversations to Firebase.



```javascript
function connectWebSocket() {
    ws = new WebSocket('ws://localhost:8765');

    ws.onopen = () => {
        console.log('WebSocket connected');
        statusIndicator.classList.remove('disconnected');
        statusIndicator.classList.add('connected');
        statusText.textContent = 'Connected';
    };

    ws.onclose = () => {
        console.log('WebSocket disconnected');
        statusIndicator.classList.remove('connected');
        statusIndicator.classList.add('disconnected');
        statusText.textContent = 'Disconnected';
        if (currentUser) setTimeout(connectWebSocket, 3000);
    };

    ws.onerror = (error) => {
        console.error('WebSocket error:', error);
    };

    ws.onmessage = async (event) => {
        const data = JSON.parse(event.data);
        if (data.type === 'response') {
            addMessage(data.text, 'assistant');

            const lastUserMsg = chatHistory[chatHistory.length - 1];
            if (lastUserMsg && currentUser) {
                chatHistory.push({
                    role: 'assistant',
                    message: data.text,
                    timestamp: new Date()
                });

                await saveChatToFirebase(lastUserMsg.message, data.text);
                await loadAllHistory(currentUser.uid);
            }

            sendBtn.disabled = false;
            stopBtn.classList.add('hidden');
            sendBtn.classList.remove('hidden');
        }
    };
}
```

3. **AI Response Generation with Context:**

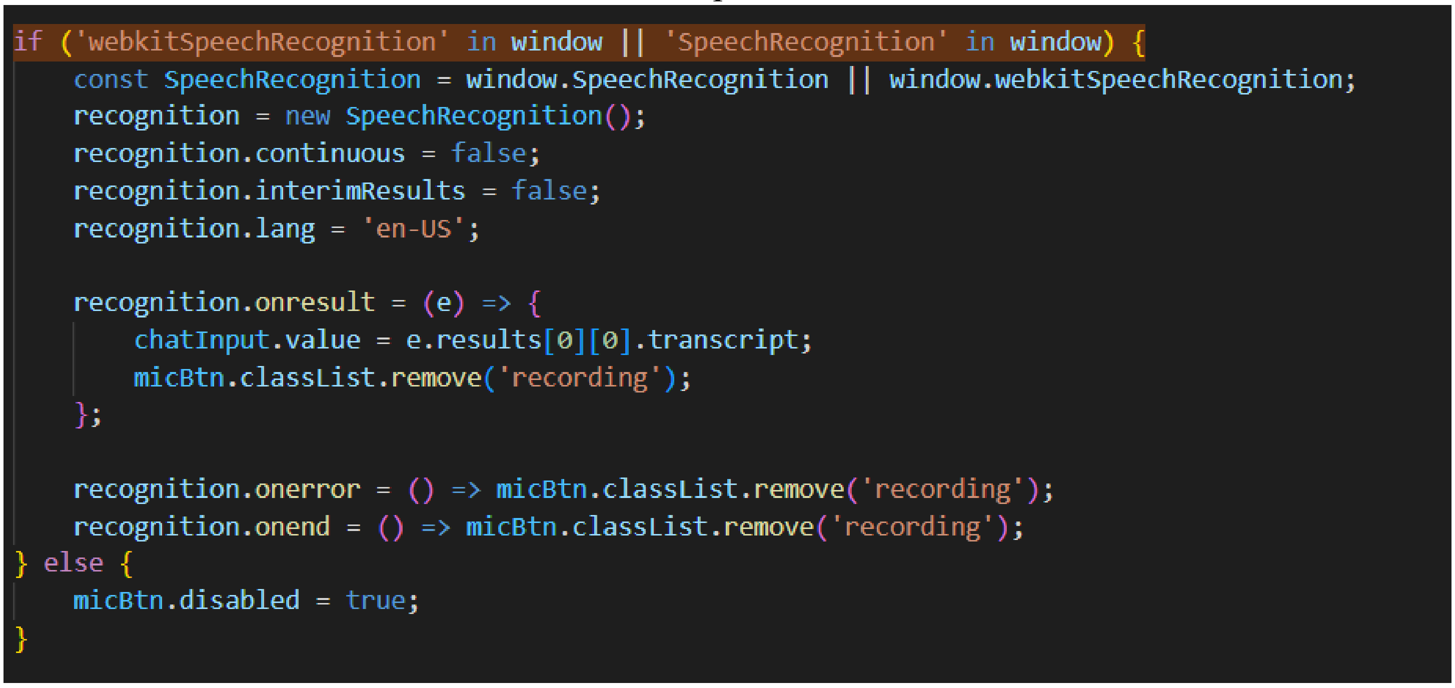
   This Python function generates AI responses using conversation history. It constructs a detailed prompt that includes the chat context and current user query, then sends it to the Gemini AI model to get a contextual response.



```python
def get_ai_response_with_context(query, history=[]):
    """
    Get AI response with conversation context

    Args:
        query (str): Current user question
        history (list): Previous conversation [{'role': 'user/assistant', 'content': '...'}]

    Returns:
        str: AI's response
    """
    if not query or not query.strip():
        return "Please ask me something!"

    try:
        # Build detailed context for Gemini
        context_prompt = """You are Friday, an AI assistant like JARVIS from Iron Man.
You are helpful, friendly, and conversational. You remember previous parts of our conversation.
Be concise but informative. Use a professional yet warm tone.

"""

        # Add conversation history if available
        if history and len(history) > 0:
            context_prompt += "PREVIOUS CONVERSATION:\n"
            context_prompt += "="*50 + "\n"

            # Format last 10 messages for context
            for i, msg in enumerate(history[-10:], 1):
                role_name = "User" if msg['role'] == 'user' else "Friday"
                context_prompt += f"{role_name}: {msg['content']}\n"

            context_prompt += "="*50 + "\n\n"
            context_prompt += "Remember the above conversation. Now respond to the current question while maintaining context.\n\n"

        # Add current query
        context_prompt += f"CURRENT QUESTION:\nUser: {query}\n\nFriday:"

        # Debug: Print context being sent (optional)
        print(f"\n{'='*60}")
        print("CONTEXT BEING SENT TO GEMINI:")
        print(context_prompt)
        print(f"{'='*60}\n")

        # Generate response
        response = model.generate_content(context_prompt)
```

4. **Voice recognition setup:**

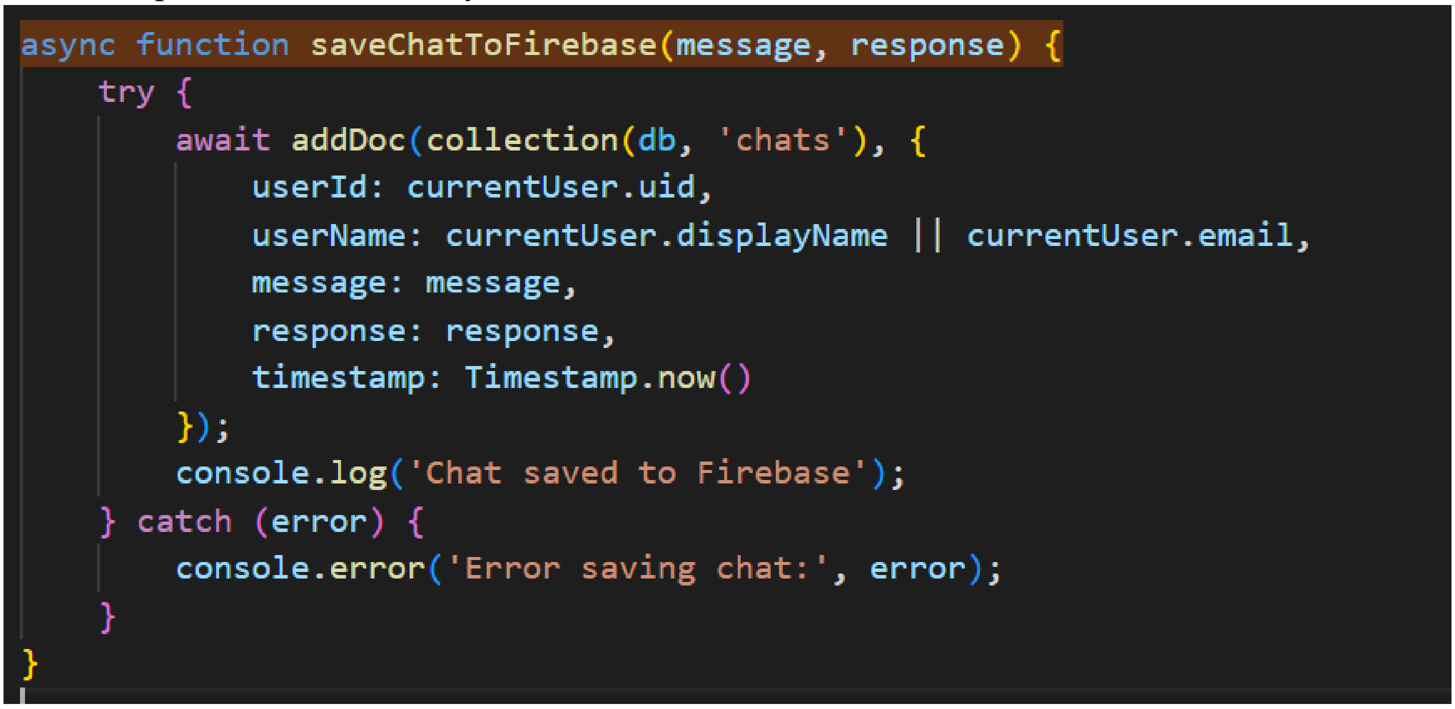
   This code implements browser-based speech recognition. It checks for browser support and initializes the speech recognition service to convert spoken words into text, which is then inserted into the chat input field.



```
if ('webkitSpeechRecognition' in window || 'SpeechRecognition' in window) {
    const SpeechRecognition = window.SpeechRecognition || window.webkitSpeechRecognition;
    recognition = new SpeechRecognition();
    recognition.continuous = false;
    recognition.interimResults = false;
    recognition.lang = 'en-US';

    recognition.onresult = (e) => {
        chatInput.value = e.results[0][0].transcript;
        micBtn.classList.remove('recording');
    };

    recognition.onerror = () => micBtn.classList.remove('recording');
    recognition.onend = () => micBtn.classList.remove('recording');
} else {
    micBtn.disabled = true;
}
```

5. **Firebase Chat Storage:**

   This asynchronous function saves chat conversations to Firebase Firestore. It stores the user's message, AI's response, user identification, and timestamp in the database for persistent chat history.



```
async function saveChatToFirebase(message, response) {
    try {
        await addDoc(collection(db, 'chats'), {
            userId: currentUser.uid,
            userName: currentUser.displayName || currentUser.email,
            message: message,
            response: response,
            timestamp: Timestamp.now()
        });
        console.log('Chat saved to Firebase');
    } catch (error) {
        console.error('Error saving chat:', error);
    }
}
```

7. Testing (Manual, Test Cases and Test Data)

7.1 Manual Testing

Test Environment:

- Operating Systems: Windows 11, macOS Sonoma
- Browsers: Chrome 120+, Firefox 121+, Safari 17+
- Python Version: 3.11.5
- Network: Local (localhost:8765)

Testing Procedure:

1. **Authentication Testing**

Test 1.1: User Registration

- **Steps:**
    1. Navigate to registration page
    2. Enter name: "Test User"
    3. Enter email: "test@example.com"
    4. Enter password: "test123"
    5. Confirm password: "test123"
    6. Click Register
- **Expected:** Account created, redirected to chat
- **Actual:** Success - Account created in Firebase Auth
- **Status:** PASS

Test 1.2: Invalid Registration (Password Mismatch)

- **Steps:**
    1. Enter password: "test123"
    2. Confirm password: "test456"
    3. Click Register
- **Expected:** Error: "Passwords do not match"
- **Actual:** Red banner displayed, form not submitted
- **Status:** PASS

Test 1.3: Duplicate Email

- **Steps:**

  Try registering with existing email
- **Expected:** Error: "Email already registered"
- **Actual:** Firebase error caught and displayed
- **Status:** PASS

Test 1.4: User Login

- **Steps:**
  1. Enter valid credentials
  2. Click Login
- **Expected:** Logged in, chat interface shown
- **Actual:** Session created, WebSocket connected
- **Status:** PASS

2. WebSocket Connection Testing

Test 2.1: Initial Connection

- **Steps:**
  1. Start Python backend (python main.py)
  2. Login to web application

Expected: Green dot, "Connected" status

Actual: Connection established immediately

Status: PASS

Test 2.2: Connection Loss

- **Steps:**

  Stop Python server while logged in
- **Expected:** Red dot, "Disconnected" status
- **Actual:** Status changed within 10 seconds
- **Status:** PASS

Test 2.3: Auto-Reconnection

- **Steps:** Restart Python server after disconnection
- **Expected:** Automatic reconnection after 3 seconds
- **Actual:** Reconnected successfully, status changed to green
- **Status:** PASS

3. Chat Functionality Testing

Test 3.1: Send Text Message

- **Steps:**
    1. Type "Hello Friday"
    2. Click Send
- **Expected:** Message sent, AI response received
- **Actual:** Response: "Hi! I'm Friday. How can I assist you today?"
- **Response Time:** 2.3 seconds
- **Status:** PASS

Test 3.2: Empty Message

- **Steps:** Try sending empty input
- **Expected:** No action, input validation
- **Actual:** Send button remained disabled
- **Status:** PASS

Test 3.3: Long Message

- **Steps:**
  Send 500-word paragraph
- **Expected:** Handled gracefully, AI responds
- **Actual:** Scrollable message bubble, AI summarized content
- **Status:** PASS

7.2 Test Cases

1. **Functional Test Cases (User Interaction)**
   These test cases verify that the core conversational features of Friday work as expected from a user's perspective.

Test - 1: Verify Basic Greeting Response

- **Description:** Ensure Friday responds appropriately to simple greetings.
- **Preconditions:** The user is on the chatbot interface.
- **Steps:**
  1. The user types "Hello" into the chat input.
  2. The user presses Enter/sends.
- **Expected Result:** Friday displays a polite greeting message (e.g., "Hello! How can I help you today?" or "Hi there!").
- **Priority:** High

Test - 2: Verify External API Integration

- **Description:** Ensure Friday successfully retrieves data from configured external services (e.g., weather API).
- **Preconditions:** An External Weather API is correctly configured and accessible. The user is on the chatbot interface.
- **Steps:**
  1. The user types "What's the temperature in New York?"
  2. The user presses Enter/sends.
- **Expected Result:** Friday displays the current temperature for New York, retrieved from the external weather API.
- **Priority:** High

Test - 3: Verify Consistent Responses

- **Description:** Ensure Friday provides consistent responses to identical, non-contextual queries.
- **Preconditions:** The user is on the chatbot interface.
- **Steps:**
  1. The user types "Who are you?"
  2. The user presses Enter/sends.
  3. The user types "Who are you?" again.
  4. The user presses Enter/sends.
- **Expected Result:** Friday provides the same or a very similar answer to both identical queries.
- **Priority:** Low

8. Enhancements

8.1 Advantages of "FRIDAY" AI Assistant

Technical Advantages:

1. **Cross-Platform Web Architecture**
    - No installation required; works on Windows, macOS, Linux, Android, iOS
    - Responsive design adapts to any screen size
    - Progressive Web App ready for offline capabilities

2. **Real-Time Communication**
    - WebSocket protocol provides instant message delivery
    - Sub-second latency for typical responses
    - Automatic reconnection on network failures

3. **Persistent Conversation Memory**
    - Cloud-based storage ensures data never lost
    - Access history from any device
    - Searchable archive of all past interactions

4. **Context-Aware Intelligence**
    - Maintains last 10 messages for coherent conversations
    - Enables natural follow-up questions
    - JARVIS-like personality through prompt engineering

5. **Scalable Architecture**
    - Asynchronous backend handles multiple concurrent users
    - Firebase auto-scales with demand
    - Can support 100+ users on free tier

User Experience Advantages:

1. **Modern, Intuitive Interface**
   - Gradient aesthetics with smooth animations
   - Familiar chat paradigm (like WhatsApp/Messenger)
   - Visual feedback for all actions

2. **Multi-Modal Input**
   - Both text and voice input supported
   - Hands-free operation for accessibility
   - Real-time transcription feedback

3. **Secure Multi-User Support**
   - Firebase Authentication with industry standards
   - Individual user accounts with data isolation
   - Personalized experience per user

Functional Advantages:

1. **System Integration**
   - Control local applications through natural language
   - Open files, play media, get time/date
   - Cross-platform command support

2. **Conversation Management**
   - Browse complete chat history
   - Search through past conversations
   - Load and continue previous discussions
   - Start fresh sessions while preserving history

8.2 Limitations of Your Project

Technical Limitations:

1. **API Dependencies**
    - Relies on Google Gemini API (rate limits, quota)
    - Firebase outages affect availability
    - Internet required for AI features

2. **Security Concerns**
    - API keys currently hardcoded in source files
    - WebSocket not encrypted (WS not WSS)
    - No end-to-end encryption for messages

3. **Scalability Constraints**
    - Single WebSocket server (no load balancing)
    - Limited to one backend instance
    - Free tier quotas restrict large-scale use

4. **Context Window Limitation**
    - Only last 10 messages maintained
    - Older context lost in long conversations
    - No conversation summarization

User Experience Limitations:

1. **Single Language Support**
    - English-only interface
    - No translation capabilities
    - Limited accessibility for non-English speakers

2. **No Offline Mode**
    - Requires internet for all features
    - Cannot function during network outages
    - No local AI fallback

3. **Limited Mobile Optimization**
    - Voice input less reliable on mobile
    - System commands don't work on mobile
    - Smaller screens less optimal for history sidebar

Functional Limitations:

1. **System Commands Platform-Dependent**
   - Application opening only works on machine running backend
   - Different commands for Windows/Linux/macOS
   - No remote system control

2. **Voice Recognition Limitations**
   - Browser-dependent (not all support Web Speech API)
   - Background noise affects accuracy
   - English-only support currently

3. **No File Processing**
   - Cannot upload and analyze documents
   - No image processing capabilities
   - Text-only input/output

4. **No Multi-Modal Output**
   - Text responses only
   - Cannot generate images or videos
   - No voice synthesis for responses

Data & Privacy Limitations:

1. **No Message Encryption**
   - Conversations stored in plaintext
   - Transmitted without encryption (WS not WSS)
   - Firebase security rules basic

2. **No Export Functionality**
   - Cannot download chat history
   - No PDF or CSV export
   - Data locked in Firebase

3. **No User Settings**
   - Cannot customize AI temperature/behavior
   - No theme selection
   - Fixed interface layout

8.3 Future Scope

Short-Term Enhancements:

1. **UI/UX Enhancements**
   - Dark Mode: Toggle between light/dark themes
   - Custom Themes: User-selectable color schemes
   - Typing Indicators: Show "Friday is typing..." animation
   - Message Timestamps: Display time for each message
   - Read Receipts: Show when messages are delivered

2. **Export Functionality**
   - PDF Export: Download conversations as PDF
   - CSV Export: Export for data analysis
   - Share Conversations: Generate shareable links
   - Backup System: Automated cloud backups

Long-Term Enhancements:

3. **Advanced AI Features**
   - Conversation Summarization: Summarize long chats
   - Smart Context Management: Adaptive context window
   - Intent Recognition: Better understand user goals
   - Sentiment Analysis: Detect user mood and adjust tone
   - Follow-Up Suggestions: Propose next questions

4. **Multi-Language Support**
   - Language Detection: Auto-detect user language
   - Translation: Real-time conversation translation
   - Localized UI: Interface in multiple languages
   - Regional Settings: Date/time format customization

5. **File Processing**
   - Document Upload: Analyze PDFs, DOCX, TXT files
   - Image Analysis: Process images with vision AI
   - Code Review: Upload and analyze code files
   - Data Extraction: Extract tables from documents

Research & Innovation:

6. **Experimental Features**
   - 3D Avatar: Animated Friday character
   - AR Integration: Augmented reality assistant
   - Blockchain Storage: Decentralized chat history
   - Quantum-Resistant Encryption: Future-proof security
   - Brain-Computer Interface: Neuralink integration (future)

9. References

Official Documentation:

1. **Google Gemini API**
   - Google AI for Developers. (2024). Gemini API Documentation.
   - URL: https://ai.google.dev/docs
   - Accessed: October 2024

2. **Firebase Platform**
   - Google Firebase. (2024). Firebase Documentation.
   - URL: https://firebase.google.com/docs
   - Sections: Authentication, Cloud Firestore, Hosting

3. **WebSocket Protocol**
   - Internet Engineering Task Force. (2011). RFC 6455: The WebSocket Protocol.
   - URL: https://datatracker.ietf.org/doc/html/rfc6455

4. **Python websockets Library**
   - Puzrin, A. (2024). websockets Documentation.
   - URL: https://websockets.readthedocs.io/
   - Version: 12.0

5. **Web Speech API**
   - Mozilla Developer Network. (2024). Web Speech API.
   - URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API

Programming Languages:

1. **Python Documentation**
   - Python Software Foundation. (2024). Python 3.11 Documentation.
   - URL: https://docs.python.org/3/

2. **JavaScript ES6+**
   - ECMA International. (2024). ECMAScript 2024 Language Specification.
   - URL: https://tc39.es/ecma262/

3. **HTML5 Specification**
   - WHATWG. (2024). HTML Living Standard.
   - URL: https://html.spec.whatwg.org/

4. **CSS3 Specification**
   - W3C. (2024). CSS Snapshot 2023.
   - URL: https://www.w3.org/Style/CSS/

Libraries & Frameworks:

1. **Firebase JavaScript SDK**
   - Google. (2024). Firebase JavaScript SDK v10.7.1.
   - URL: https://firebase.google.com/docs/web/setup

2. **google-generativeai (Python)**
   - Google. (2024). Google Generative AI Python Client.
   - PyPI: https://pypi.org/project/google-generativeai/

3. **asyncio (Python Standard Library)**
   - Python Software Foundation. (2024). asyncio — Asynchronous I/O.
   - URL: https://docs.python.org/3/library/asyncio.html