

Software Requirements Specification (SRS)

Use and Learn Chatbot

Version: 1.0

Date: February 23, 2025

Prepared by: Mohammed Sardar Saajit

Project: Use and Learn Chatbot

Table of Contents

1. [Introduction](#)
 - 1.1 Purpose
 - 1.2 Scope
 - 1.3 Definitions, Acronyms, and Abbreviations
 - 1.4 References
 - 1.5 Overview

2. [Overall Description](#)
 - 2.1 Product Perspective
 - 2.2 Product Functions
 - 2.3 User Characteristics
 - 2.4 Constraints
 - 2.5 Assumptions and Dependencies

3. [Specific Requirements](#)
 - 3.1 External Interface Requirements
 - 3.1.1 User Interfaces
 - 3.1.2 Hardware Interfaces
 - 3.1.3 Software Interfaces
 - 3.1.4 Communication Interfaces
 - 3.2 Functional Requirements
 - 3.3 Non-Functional Requirements
 - 3.3.1 Performance Requirements
 - 3.3.2 Security Requirements
 - 3.3.3 Usability Requirements
 - 3.3.4 Reliability Requirements
 - 3.4 Design Constraints

4. [System Features](#)
 - 4.1 Chat Interface
 - 4.2 Learning Mechanism
 - 4.3 Persistent Storage

 5. [Other Requirements](#)
 - 5.1 Documentation
 - 5.2 Legal and Ethical Considerations

 6. [Appendix](#)
-

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) defines the requirements for the "Use and Learn Chatbot," a web-based conversational AI designed to interact with users and learn responses dynamically based on user input. The purpose is to provide a clear blueprint for development, testing, and deployment, ensuring the chatbot meets user needs for simple interaction and adaptive learning.

1.2 Scope

The Use and Learn Chatbot is a Flask-based web application that:

- Allows users to engage in text-based conversations through a modern, clean UI.
- Learns new responses when encountering unknown inputs, storing them persistently.
- Provides a visually appealing interface with real-time feedback and animations.
- Runs locally on a user's machine, accessible via a web browser.

The system is intended for personal or educational use, focusing on simplicity and interactivity rather than complex AI capabilities.

1.3 Definitions, Acronyms, and Abbreviations

- **SRS:** Software Requirements Specification
- **Flask:** A Python web framework used for the backend
- **UI:** User Interface
- **JSON:** JavaScript Object Notation, used for storing the knowledge base
- **HTTP:** HyperText Transfer Protocol, for client-server communication
- **CSS:** Cascading Style Sheets, for styling the UI
- **HTML:** HyperText Markup Language, for structuring the UI

1.4 References

- Flask Documentation: <https://flask.palletsprojects.com/>
- HTML/CSS Standards: W3C (<https://www.w3.org/>)
- Project Codebase: Provided by the user on February 23, 2025

1.5 Overview

This document covers the product's functionality, user interface, and technical requirements. It describes how the chatbot interacts with users, learns, and stores data, along with performance and usability expectations.

2. Overall Description

2.1 Product Perspective

The Use and Learn Chatbot is a standalone web application built with Flask (backend), HTML/CSS (frontend), and JavaScript (client-side logic). It operates as a client-server system where the Flask server processes user inputs and serves a dynamic UI. The chatbot fits into a broader context of educational tools or personal assistants, emphasizing user-driven learning over pre-programmed responses.

2.2 Product Functions

- **Conversation:** Users can send messages, and the chatbot responds in real-time.
- **Learning:** The chatbot learns responses for unknown inputs by prompting users to teach it.
- **Persistence:** Learned responses are saved in a JSON file (chatbot_knowledge.json) for reuse across sessions.
- **Exit Command:** Users can type "exit" to end the conversation gracefully.
- **Visual Feedback:** Includes animations (e.g., typing indicators, fading messages) and distinct styling for user/bot/teaching messages.

2.3 User Characteristics

- **Target Users:** General users, including students, hobbyists, or anyone interested in a simple chatbot.

- **Skill Level:** Basic computer literacy; no programming knowledge required.
- **Usage Context:** Casual or educational use, typically on a personal computer with a web browser.

2.4 Constraints

- Runs on a local server (<http://127.0.0.1:5000>), not designed for multi-user or remote access without modifications.
- Limited to text-based interactions; no multimedia support (e.g., images, voice).
- Learning is single-turn (one message to teach a response); no complex dialogue management.



2.5 Assumptions and Dependencies

- **Assumptions:** Users have Python and Flask installed, and a modern web browser (e.g., Chrome, Firefox) is available.
 - **Dependencies:** Python 3.x, Flask library, a working internet connection for initial setup (to install Flask).
-

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

- **Chat Window:** A scrollable area displaying messages with avatars ( for bot,  for user).

- **Input Area:** A text field and "Send" button, with Enter key support.
- **Visual Design:** Gradient backgrounds, rounded corners, animations (e.g., fade-in messages, spinning send icon).
- **Feedback:** Thinking dots during response generation, distinct bubble colors for user (purple), bot (gray), and teaching prompts (yellow).

3.1.2 Hardware Interfaces

- Runs on standard desktop/laptop hardware with a web browser.
- No specific hardware requirements beyond Python execution capability.

3.1.3 Software Interfaces

- **Flask:** Backend framework for routing and templating.
- **Python:** Core runtime environment.
- **JSON:** File-based storage for the knowledge base.
- **Web Browser:** Renders HTML/CSS/JavaScript frontend.

3.1.4 Communication Interfaces

- **HTTP:** Uses POST requests to /chat endpoint for message processing.
- **Localhost:** Operates on http://127.0.0.1:5000 by default.

3.2 Functional Requirements

1. FR1: Message Display

- The system shall display user and bot messages in the chat area with appropriate avatars and styling.

2. FR2: Message Sending

- The system shall send user input to the backend when the "Send" button is clicked or Enter is pressed.

3. FR3: Response Generation

- The system shall respond to known messages with stored responses or prompt for teaching if unknown.

4. FR4: Learning

- The system shall store a user-provided response for an unknown message in the knowledge base after prompting.

5. FR5: Persistence

- The system shall save the knowledge base to chatbot_knowledge.json after each update.

6. FR6: Exit Functionality

- The system shall display "Goodbye!" and reset teaching mode when "exit" is entered.

3.3 Non-Functional Requirements

3.3.1 Performance Requirements

- Response time: <1 second for local processing.
- Chat area capacity: Supports up to 100 messages without significant lag (scrollable beyond that).

3.3.2 Security Requirements

- No authentication required (local use only).
- Knowledge base file (chatbot_knowledge.json) should be writable by the application.

3.3.3 Usability Requirements

- Intuitive UI with clear visual cues (e.g., teaching mode highlighted in yellow).
- Accessible via keyboard (Enter key support) and mouse.
- ARIA labels for input and button to support screen readers.

3.3.4 Reliability Requirements

- The system shall recover gracefully from network errors, displaying "Oops, something went wrong!" if the backend fails.

3.4 Design Constraints

- Single-user, local deployment.
- No support for concurrent users without server modifications.

- Limited to English text input/output.

4. System Features

4.1 Chat Interface

- **Description:** A web-based interface for real-time text conversation.
- **Input:** User types messages in a text field.
- **Output:** Messages displayed in a scrollable chat area with animations.
- **Priority:** High

4.2 Learning Mechanism

- **Description:** The chatbot learns new responses from user input.
- **Input:** User message triggering "I don't know that yet" → subsequent teaching response.
- **Output:** Confirmation ("Thanks for teaching me!") and updated knowledge base.
- **Priority:** High

4.3 Persistent Storage

- **Description:** Stores learned responses across sessions.
- **Input:** Updated knowledge base after teaching.

- **Output:** JSON file (chatbot_knowledge.json) with key-value pairs (message: response).
 - **Priority:** Medium
-

5. Other Requirements

5.1 Documentation

- Basic setup instructions (e.g., "Install Flask, run python app.py, visit <http://127.0.0.1:5000>").
- No extensive user manual required due to simplicity.

5.2 Legal and Ethical Considerations

- No personal data collected; all data stored locally in chatbot_knowledge.json.
 - Open-source libraries (Flask, Python) used under their respective licenses.
-

6. Appendix

- **Sample Interaction:**

text

WrapCopy

User: hi

Bot: I don't know that yet. What should I say?

User: Hello there!

Bot: Thanks for teaching me!

User: hi

Bot: Hello there!

User: exit

Bot: Goodbye!

- **Knowledge Base Example** (chatbot_knowledge.json):

json

WrapCopy

```
{  
  "hi": "Hello there!",  
  "bye": ""  
}
```