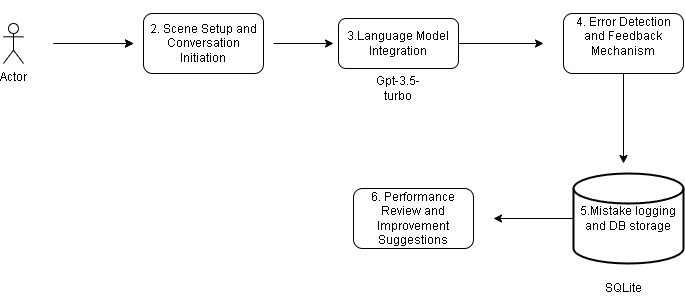
# Language-Learning Chatbot - Documentation

## 1. Introduction

This project is an AI-powered chatbot designed to assist users in learning a new language through interactive conversations. The chatbot collects user preferences, including the target language, known language, and proficiency level, to create a relevant conversation scenario. Throughout the interaction, it corrects user mistakes in real-time, maintains a record of errors, and provides a summary of mistakes along with personalized feedback at the end of the session.  
The chatbot utilizes OpenAI's LLM models for natural language processing and response generation. It leverages LangChain as an AI wrapper library to streamline model interactions. A local SQLite database is used to store user conversation history and track errors, ensuring structured feedback and progress tracking.

## 2. System Architecture

The chatbot follows a structured workflow consisting of multiple components:  
\* Interaction: Collects user input and sets up a relevant conversation scenario.  
\* AI Model Processing: Uses OpenAI models and LangChain for generating responses.  
\* Error Tracking & Database Management\*\*: Stores user interactions and mistakes in SQLite.  
\* Feedback & Summary Generation\*\*: Provides users with a breakdown of errors and improvement suggestions.



Step1:

**1. User Input Handling**

* The chatbot begins by asking the user relevant questions:
  + The language they want to learn.
  + The language they already know.
  + Their current proficiency level.
* This information is stored temporarily to personalize the conversation.

**2. Scene Setup and Conversation Initiation**

* Based on user selections, the chatbot creates a contextual scenario. For example:
  + If the user is learning French and is at the beginner level, the bot might simulate a restaurant ordering scenario in French.
* The chatbot starts conversing with the user in the target language.

**3. Language Model Integration**

* The chatbot is integrated with OpenAI’s API via **LangChain**, a framework that helps handle prompts and memory efficiently.
* User inputs are processed and sent to OpenAI’s API to generate responses.
* The generated responses are returned in the learning language while ensuring they align with the chosen scenario.

**4. Error Detection and Feedback Mechanism**

* The chatbot continuously monitors the user's responses, identifying mistakes in:
  + Grammar
  + Vocabulary
  + Sentence structure
  + Pronunciation (if speech input is supported)
* The system provides real-time corrections and explanations to help users improve.

**5. Mistake Logging and Database Storage**

* Each mistake the user makes is logged in a **local database** (e.g., SQLite).
* The database maintains:
  + The type of mistakes.
  + Corrected versions.
  + Frequency of recurring errors.

**6. Performance Review and Improvement Suggestions**

* At the end of the session, the chatbot retrieves logged mistakes and provides a summary:
  + Common errors and their corrections.
  + Suggestions for improvement

## 3. Installation & Setup

System Requirements:  
- Python 3.9  
- OpenAI API key  
- SQLite database

Installation Steps:

* 1. Clone the repository and navigate to the project folder:

git clone <repository\_url>

cd chatbot\_project

* 2. Install dependencies:

pip install -r requirements.txt

* 3. Run the chatbot:

python chatbot.py

## 4. File & Code Structure

Main Files:

|  |  |
| --- | --- |
| File/Folder | Description |
| main.ipynb or chatbot.py | Main chatbot logic |
| db\_manager.py | Handles SQLite database operations |
| chatbot.db | Stores user conversation history and mistakes |
| requirements.txt | Lists required Python dependencies |

## 5. Database Schema

|  |  |  |
| --- | --- | --- |
| Table Name | Column Name | Description |
| conversations | id | Unique identifier |
| conversations | user\_input | Stores user messages |
| conversations | bot\_response | Stores chatbot replies |
| mistakes | error\_type | Type of mistake (grammar, vocabulary, syntax) |

## 6. API Integration

- Uses OpenAI API for chatbot responses.  
- LangChain is implemented for efficient AI interaction.  
- Example API call:

response = openai.ChatCompletion.create(  
 model="gpt-4",  
 messages=[{"role": "user", "content": user\_input}]  
)

## 7. Features & Functionalities

✔ Dynamic Conversation Scenarios– Tailors responses based on user preferences.  
✔ Grammar & Vocabulary Corrections– Provides real-time feedback.  
✔ Error Tracking– Maintains a log of mistakes for progress tracking.  
✔ Session Summary & Feedback– Gives users a breakdown of their mistakes and improvement areas.  
✔ Database Storage– Stores conversation history and mistakes for reference.

## 8. Future Improvements

- Speech-to-text integration for voice-based conversations.  
- Adaptive learning model that adjusts difficulty based on user performance.  
- Multi-user support with personalized progress tracking and session history.  
- Integration with spaced repetition algorithms for vocabulary reinforcement.

## 9. Conclusion

This chatbot serves as an AI-powered language-learning assistant that engages users in real-time conversations, corrects mistakes, and provides structured feedback. By leveraging OpenAI’s language models, LangChain, and SQLite, the chatbot ensures an interactive and data-driven approach to language acquisition. Future improvements could further enhance user experience through voice integration and adaptive learning mechanisms.

## 10. References

- \*\*OpenAI API Documentation\*\* – https://platform.openai.com/docs/  
- \*\*LangChain Framework\*\* – https://python.langchain.com/  
- \*\*SQLite Documentation\*\* – <https://sqlite.org/docs.html>

## 10. Future improvements

-\*\*Speech-to-Text & Text-to-Speech

-\*\*Gamify the Learning Process