## ****GEN AI PROJECT PHASE 3 SUBMISSION DOCUMENT****

### ****Phase 3: Final Report and Submission****

### ****1. Project Title:****

Chatbot using IBM Watson Assistant and Flask

### ****2. Summary of Work Done****

#### ****Phase 1 – Proposal and Idea Submission (10 Marks):****

In the initial phase, we proposed to build a chatbot system that integrates IBM Watson Assistant with a Flask-based web interface. The main objectives were:

* Design an intelligent chatbot capable of real-time interaction.
* Utilize IBM Watson Assistant’s NLP capabilities for human-like conversations.
* Deploy the chatbot on a user-friendly Flask web app.

We submitted a detailed proposal including the project goals, technology stack, and the envisioned solution architecture.

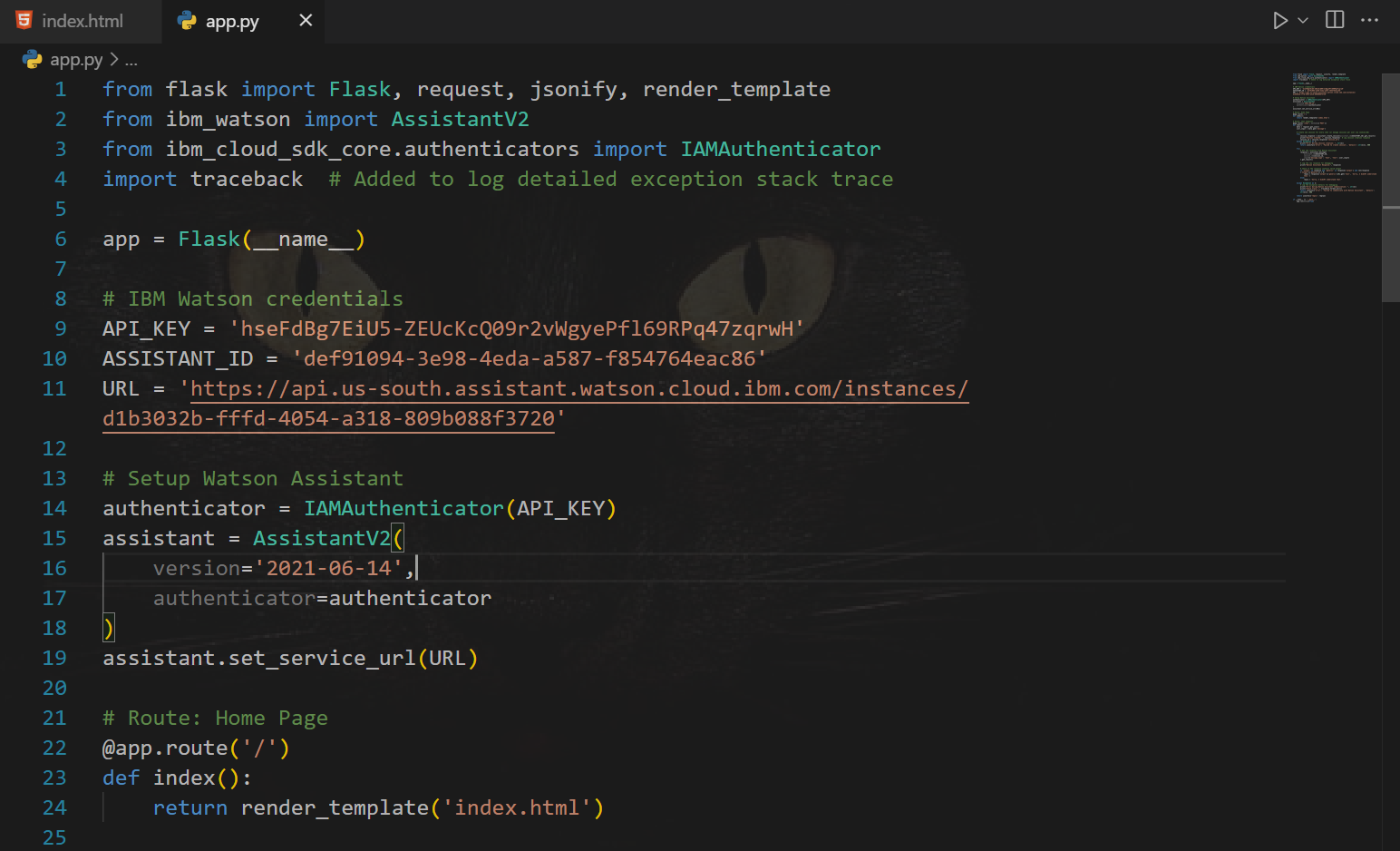
#### ****Phase 2 – Execution and Demonstration (15 Marks):****

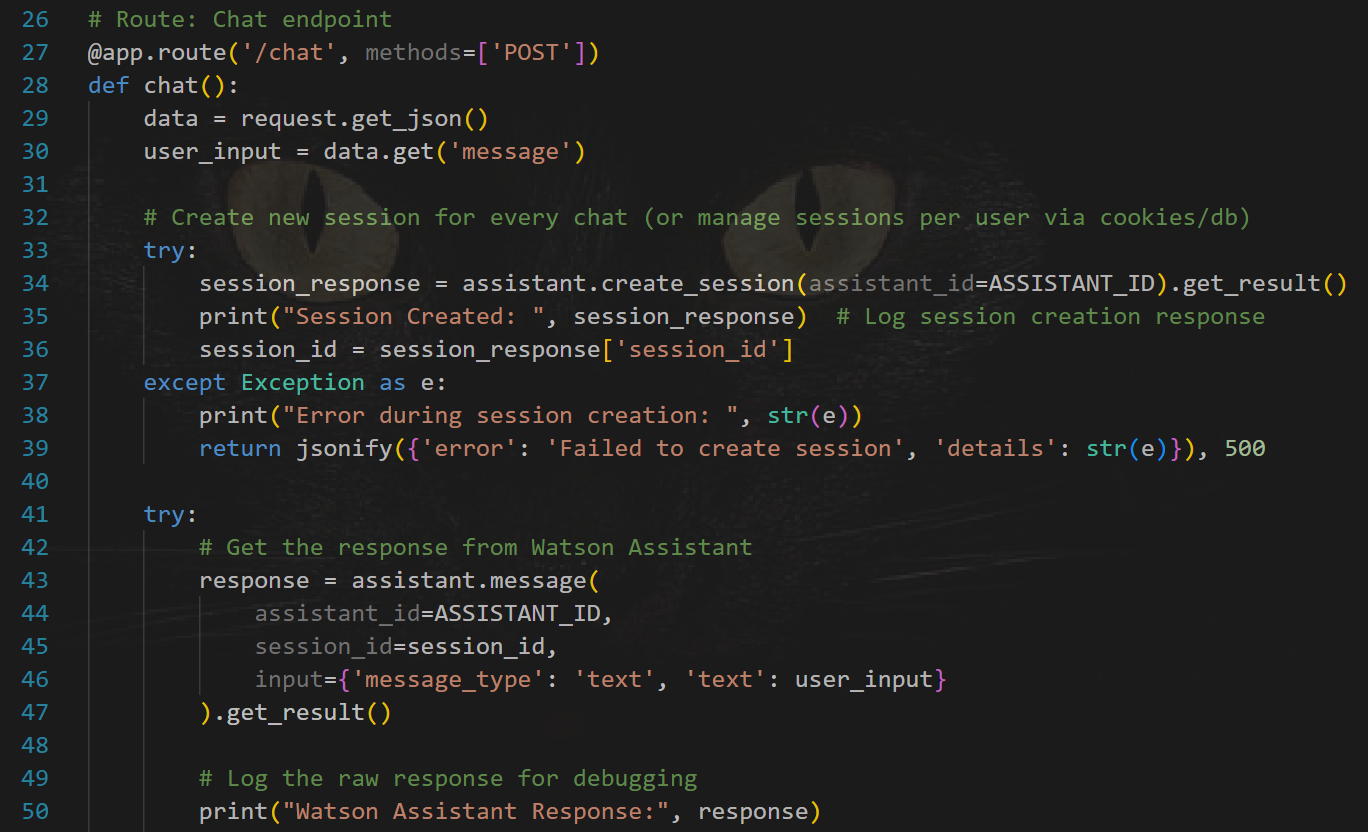
During the second phase, we translated our proposal into a working application. The following steps were executed:

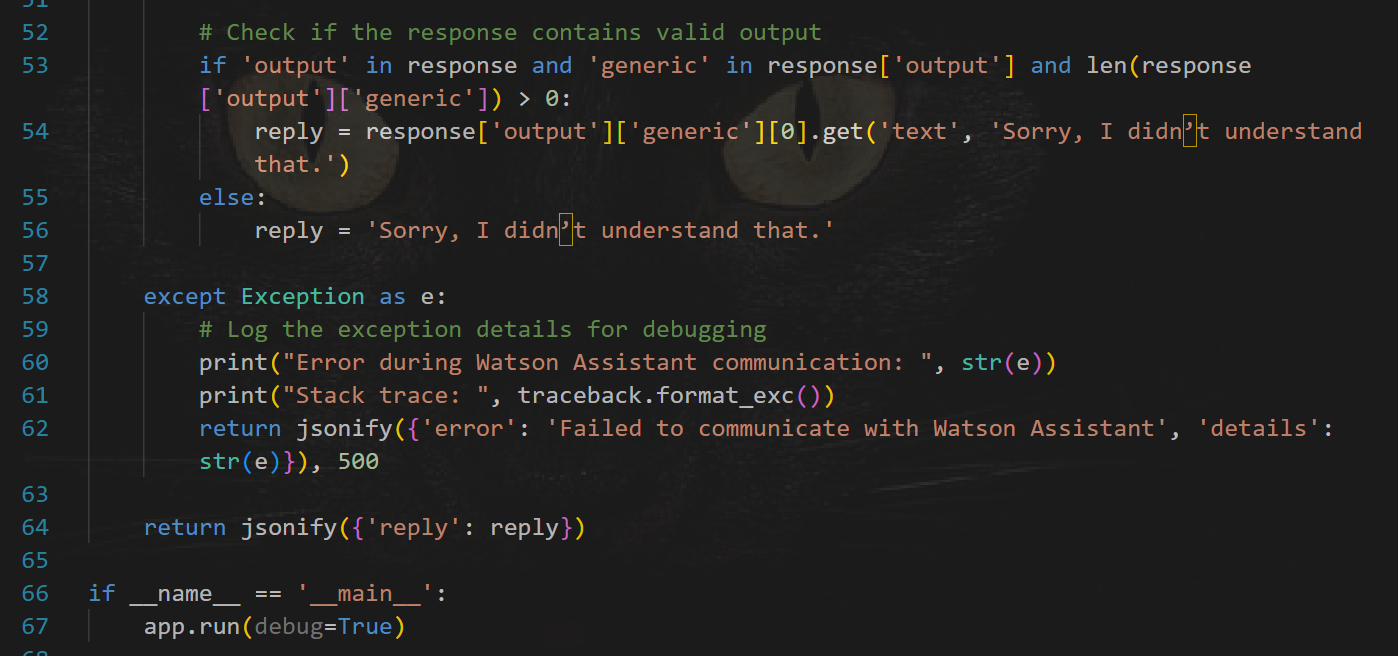
* Configured Watson Assistant and created dialogue intents in IBM Cloud.
* Integrated the Watson API with a Python backend using ibm-watson SDK.
* Developed a Flask web interface to allow users to interact with the chatbot.
* Implemented POST request handling for real-time message processing.
* Resolved issues such as session management and error handling.

Screenshots, codebase, and a working demonstration were provided to showcase the output.

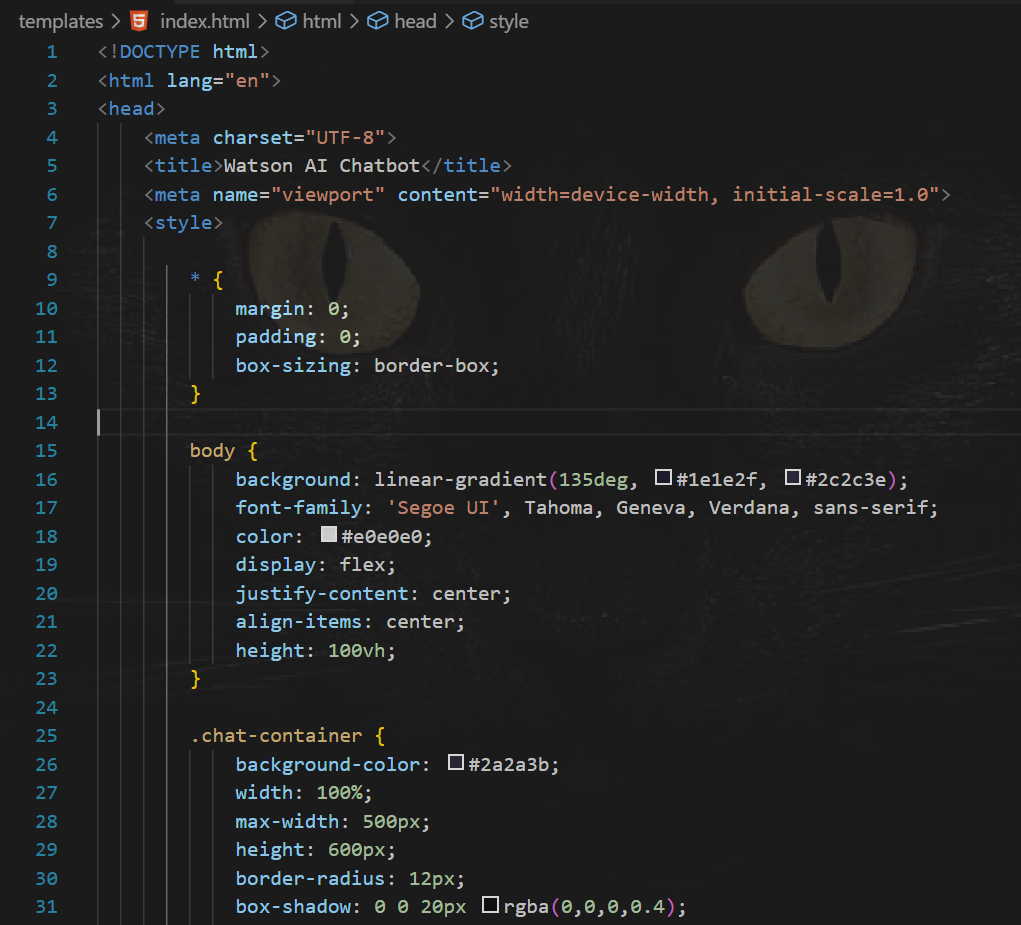
App.py

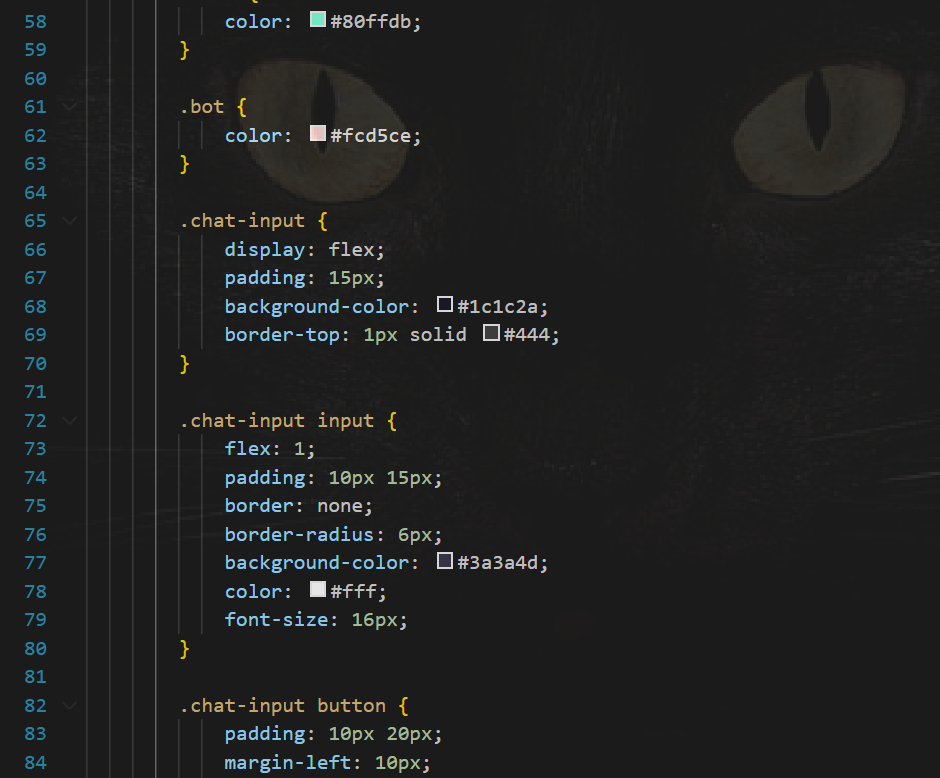


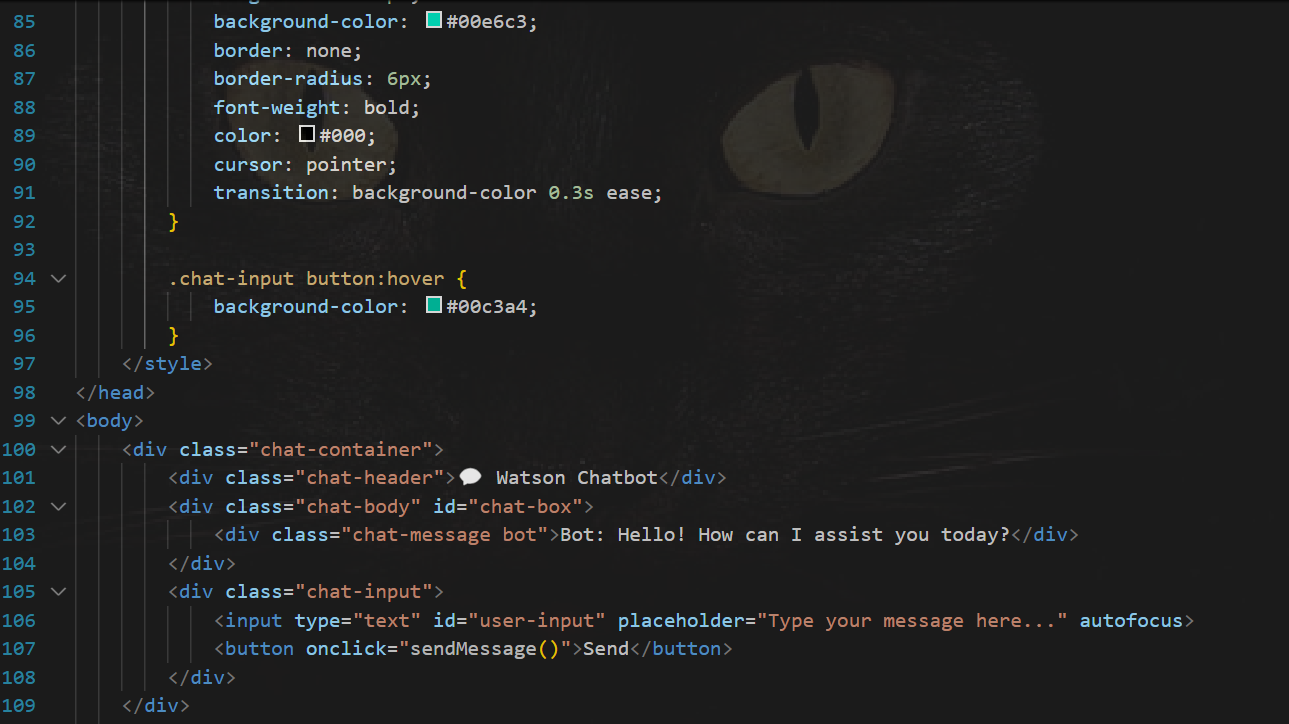




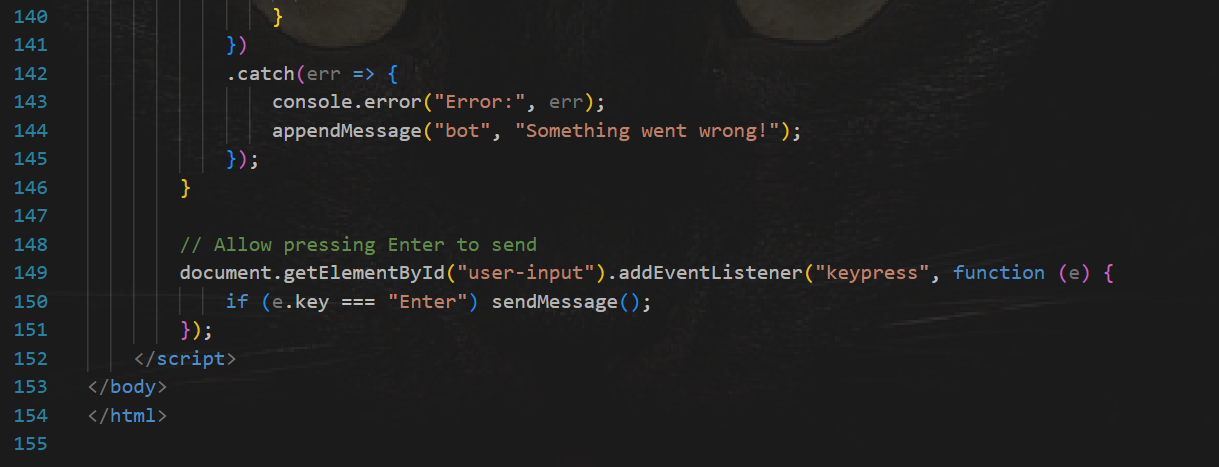
Index.html











### ****3. GitHub Repository Link****

You can access the complete codebase, README instructions, and any related resources at the following GitHub link:

🔗  [**https://github.com/AnshBansalbpl/GenAI-project.git**](%20https://github.com/AnshBansalbpl/GenAI-project.git%20)

### ****4. Testing Phase****

#### ****4.1 Testing Strategy****

To ensure our chatbot performs accurately and handles real-world scenarios, we tested the system extensively with the following focus areas:

* **Input Robustness:** Tested inputs with different sentence structures and languages.
* **Watson Response Validation:** Verified that Watson responds correctly based on defined intents.
* **Session Management:** Ensured sessions are created and closed properly.

#### ****4.2 Types of Testing Conducted****

1. **Unit Testing**
   * Tested Watson integration, message handlers, and response logic independently.
2. **Integration Testing**
   * Validated the interaction between Flask frontend and Watson backend through API requests.
3. **User Testing**
   * Real users interacted with the chatbot and shared usability feedback which helped refine the interface and conversational flow.
4. **Performance Testing**
   * Tested response time under different network conditions and multiple simultaneous queries.

#### ****4.3 Results****

* **Accuracy**: Watson accurately mapped inputs to the right intent, returning meaningful responses.
* **Response Time**: The chatbot handled multiple messages without session timeout errors.
* **UX Feedback**: Users found the interface simple and intuitive.

### ****Future Work****

While the chatbot is fully functional, there are many ways to extend its utility:

1. **Contextual Memory:**  
   Allow the bot to retain past conversation context for a more natural dialogue.
2. **Domain-Specific Training:**  
   Train Watson on custom datasets to serve specific industries like healthcare or banking.
3. **Voice Input Support:**  
   Integrate speech-to-text to allow voice-based queries and replies.
4. **Advanced Analytics Dashboard:**  
   Provide admin with usage data, query frequency, and user satisfaction metrics.
5. **Multi-language Support:**  
   Enable the chatbot to handle multiple languages using Watson’s language services.

### ****6. Conclusion****

The project effectively demonstrates the creation of an AI-powered chatbot using IBM Watson Assistant and Flask. Through all three phases, we progressed from planning and development to deployment and testing. The system showcases how cloud-based NLP tools can be used to build scalable, intelligent conversational agents. This chatbot has potential for use in websites, customer support platforms, and educational services.