Chatbot Project Report: Library Service Assistant

1. Introduction & Domain

Domain of Operation: This chatbot is designed to become a personal library service assistant. Its goal is to automate replies to frequently asked questions and transactions and reduce the amount of work in librarians.

Use Case:

- Inquiry about library policies (e.g., opening hours, borrowing limits).
- Search for specific books in the catalog.
- Perform real-time transactions (borrowing and returning books), which updates the library's inventory database.
- Experience personalized interaction through context memory (e.g., the bot remembers the user's name).

2. Process Reflection (Iterative Development)

The development of this chatbot followed a structured iterative process, strictly aligning with the weekly learning modules of the course:

Iteration 1: Chatbot part 1 (Weeks 1-4)

In the first 4 weeks, we downloaded Python and set up the environment for this course. I learned how to use anaconda. Because I learned Python during my high school, I am having a great time these weeks.

Iteration 2: Chatbot part 2 (Weeks 5-6)

In these two weeks, we learned what dictionaries are which can store and read the intent more efficiently. Also, introduced to the NLTK library for text analysis during week 5. After adding NLTK to my chatbot, the chatbot finally looks smarter.

Iteration 3: Chatbot part 3 (Weeks 7-8)

In this iteration, we are moving towards a file system to apply JSON file into the chatbot. Following the course materials, I create an intents.json file to store the

patterns and responses of my chatbot which makes me easier to manage and adjust the intents.

Iteration 4: Chatbot part 4 (Weeks 9-10)

We explored the techniques for cleaning text (e.g. remove stop words and punctuation), stemming and lemmatization. And when I use lemmatization in the chatbot, it always classifies “borrowing” as a noun. So, I implemented a filter dictionary to explicitly map words like “borrowing” and “loaned” to verb. In addition, I decided on the topic of my chatbot to be the library assistant. Therefore, I focused on those advance features after setting up the basic chatbot.

3. Code Organization

The project follows software engineering best practices by ensuring high modularity and separation of concerns:

1. Data Separation:
    - intents.json: Stores conversational patterns and responses.
    - books.json: Stores the inventory database (book titles, authors, quantities).
2. Function Separation:
    - Core Functions: load_json handles file I/O; chatbot_main manages the lifecycle of the application.
    - NLP Pipeline: preprocess_input handles tokenization, POS tagging, and lemmatization.
    - Logic Router: get_response acts as the main controller. It delegates specific tasks to get_response_of_books when book-related intents are detected.
    - Helper Functions: Specialized functions like check_availability and update_books to reduce the cyclomatic complexity.

4. Advanced Features

Beyond standard pattern matching, this chatbot implements several advanced features:

- Inventory Management System: Unlike a static FAQ bot, this system reads and writes to a database (books.json). It calculates available_copies = total - loaned in real-time. When a user borrows a book, the loaned_quantity updates, and the changes are saved back to the JSON file upon exit.
- Contextual Memory: The bot uses Regex capture groups (e.g., My name is (.*)) to extract user details and store them in a user_memory dictionary. It uses template replacement to generate personalized responses (e.g., "Nice to meet you, {name}." become "Nice to meet you, Alice.").
- Advanced NLP Preprocessing: The preprocess_input function uses Part-of-Speech (POS) tagging to improve lemmatization accuracy. It includes a custom exception handler for words like "borrowing" or "loaned" to ensure they match the talking situation.

5. Test Cases

The following test cases demonstrate the chatbot's robustness and functionality:

Test Case 1: Context & Memory

- Goal: Verify the bot can extract entities and recall them.
- Input: "My name is Alice"
- Behavior: The regex (.*) captures "Alice". The capture_information function stores it in user_memory.
- Output: The bot responds with "Nice to meet you, Alice!" confirming the data was stored and retrieved.

Test Case 2: NLP Preprocessing

- Goal: Verify lemmatization handles word variations.
- Input: "What are the borrowing limits?"
- Behavior: The raw input is tokenized. The POS tagger identifies

"borrowing". The custom correction logic forces it to be treated as a verb, lemmatizing it to "borrow".

- Match: Matches the pattern "borrowing limit" (which might be normalized) or keywords related to borrow_limit intent.

Test Case 3: Book Transaction Logic

- Goal: Verify the inventory system updates correctly.
- Scenario: A user wants to borrow "Test Book" (Initial stock: 1 available).
- Input: "I want to borrow Test Book"
- Behavior: The bot identifies the book, checks if loaned < total, increments loaned_quantity, and returns a success message.
- Verification: The internal state of books.json is updated, reducing the available stock to 0.