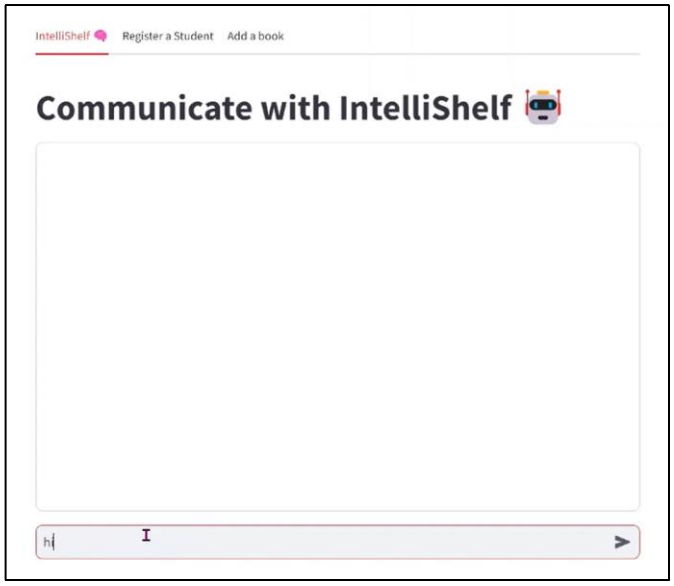


IntelliShelf – A Smart Library System

PRESENTED BY

SAAD JAVED   F2021376120
ALI ADIL WASEEM   F2021376126
ABDULLAH FAISAL   F2021376111

Abstract

IntelliShelf is an automated library book borrowing system implemented using ESP32 and MFRC522 scanner. The system facilitates efficient book management and user interaction through RFID technology and chat functionality. The project integrates AIML for basic chat responses and Llama 3 API for advanced queries. Neo4j is utilized as the database to store user, book, and interaction data.

Features

- **Add Books:** Register new books in the library.
- **Register Students:** Add new students to the system.
- **Borrow Books:** Students log in by scanning their RFID card, then scan the book to borrow it.
- **Return Books:** Students scan the book again to return it.
- **Chat Functionality:** Users can interact with a chat bot to inquire about books, check borrowed books, and more.
- **Data Integrity and Authentication:** Ensures accurate data handling and secure access.
- **Logout:** Allows users to log out to maintain system security and enable others to access the system.

Components

- **ESP32 & MFRC522 Scanner:** Core hardware components for RFID scanning.
- **LCD Display:** Displays information to the user.
- **MIFARE Classic 1K Cards:** Used for student and book identification.
- **FastAPI Server:** Facilitates communication between the ESP32 and the application.
- **Neo4j Database:** Stores data related to users, books, and interactions.
- **AIML & Llama 3 API:** Implemented for chat functionality.

System Architecture

Hardware Components

- **ESP32 Microcontroller:** Manages RFID scanning and communicates with the server.
- **MFRC522 RFID Reader:** Scans RFID chips in books and student cards.
- **LCD Display:** Shows system status and information.

Software Components

- **FastAPI:** Used as the backend server to handle communication between hardware and application.
- **Neo4j Database:** Stores user data, book data, and chat interactions.
- **AIML & Llama 3 API:** Provide chat responses.

Database Schema

The Neo4j database schema includes nodes and relationships representing users, books, and interactions.

User Nodes

- **User:** Represents a student with properties such as name, ID, and department.
- **SensoryMemory:** Stores user interactions (prompts and sentences).
- **EpisodicMemory:** Stores user sessions (episodes).

Book Nodes

- **Book:** Represents a library book with properties such as title, author, and RFID.

Relationships

- **HAS_MEM:** Connects a user to their sensory or episodic memory.
- **HAS_PROMPT:** Connects sensory memory to prompts.
- **HAS_SENT:** Connects prompts to sentences.
- **HAS_WORD:** Connects sentences to words.
- **NEXT:** Connects words in sequence.
- **HAS_EP:** Connects episodic memory to episodes.

Functionalities

Adding Books

To add a new book, the admin uses the frontend interface to enter book details. The book is then registered in the Neo4j database.

Registering Students

Admins can register new students through the frontend. Each student is assigned an RFID card, and their information is stored in the Neo4j database.

Borrowing and Returning Books

Students log in by scanning their RFID card, then scan the book they wish to borrow. The system updates the Neo4j database with the borrowing information. Scanning the book again returns it to the library.

Chat Functionality

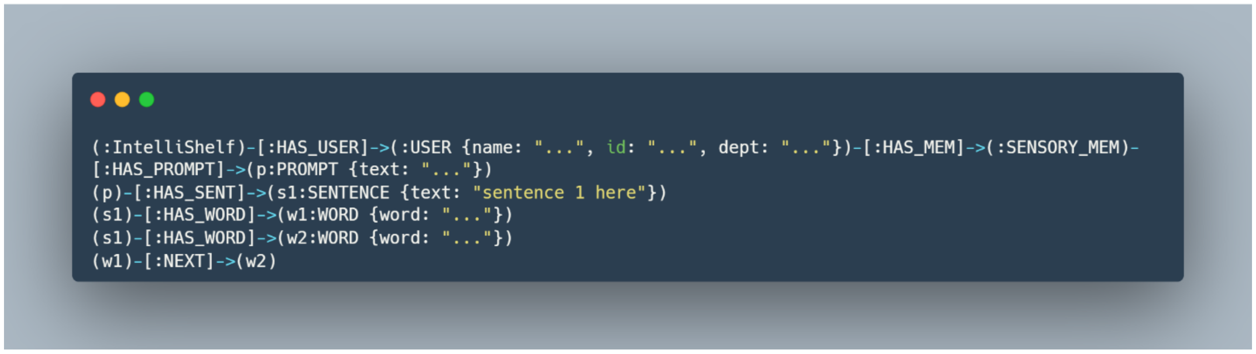
Students can interact with the chat bot after logging in. The chat bot uses AIML for basic responses and Llama 3 API for more complex queries. The chat data is stored in the Neo4j database.

Chat Implementation

Sensory Memory

Sensory memory stores the user's prompts and responses in a structured manner, enabling the system to remember previous interactions.

Graph Structure:



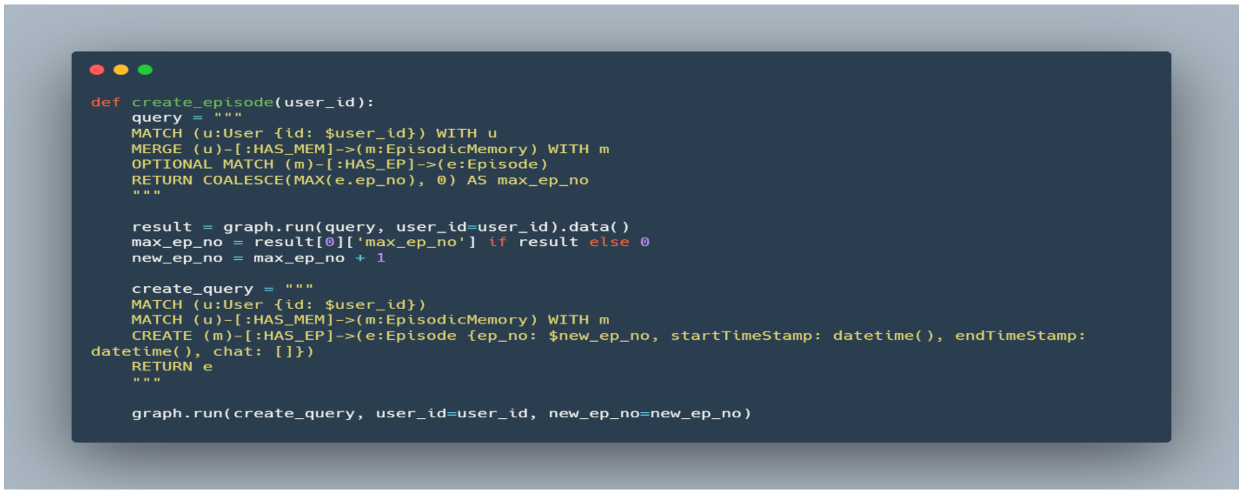
```
(:IntelliShelf)-[:HAS_USER]->(:USER {name: "...", id: "...", dept: "..."})-[:HAS_MEM]->(:SENSORY_MEM)-
[:HAS_PROMPT]->(p:PROMPT {text: "..."})
(p)-[:HAS_SENT]->(s1:SENTENCE {text: "sentence 1 here"})
(s1)-[:HAS_WORD]->(w1:WORD {word: "..."})
(s1)-[:HAS_WORD]->(w2:WORD {word: "..."})
(w1)-[:NEXT]->(w2)
```

Episodic Memory

Episodic memory records user sessions, creating a new episode each time a user logs in.

Functions:

Creating an Episode:



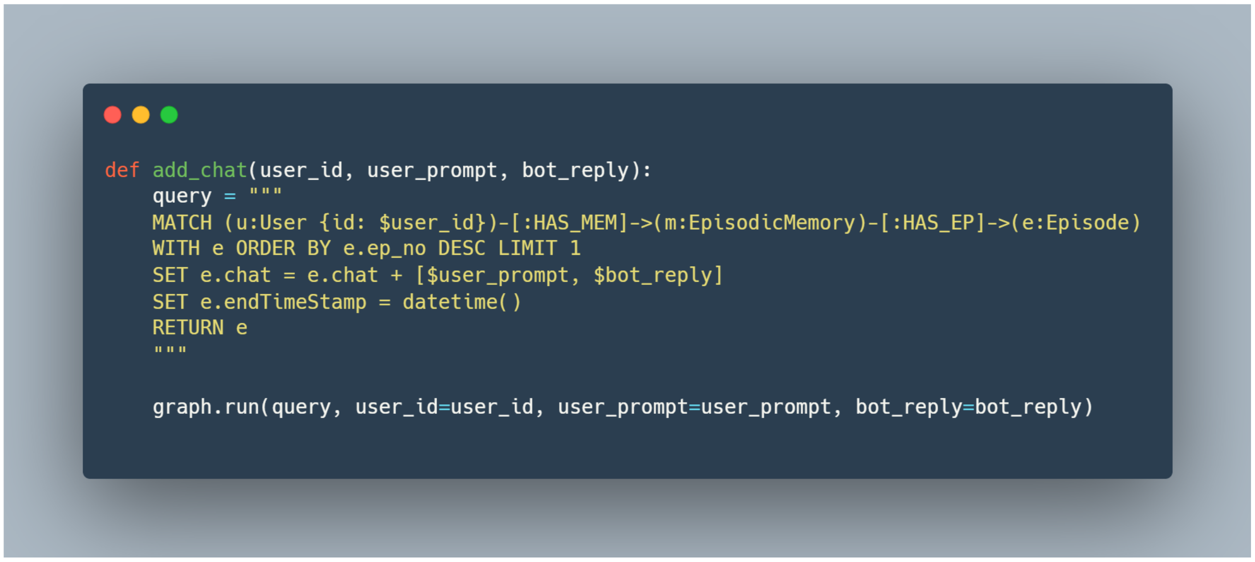
```
def create_episode(user_id):
    query = """
    MATCH (u:User {id: $user_id}) WITH u
    MERGE (u)-[:HAS_MEM]->(m:EpisodicMemory) WITH m
    OPTIONAL MATCH (m)-[:HAS_EP]->(e:Episode)
    RETURN COALESCE(MAX(e.ep_no), 0) AS max_ep_no
    """

    result = graph.run(query, user_id=user_id).data()
    max_ep_no = result[0]['max_ep_no'] if result else 0
    new_ep_no = max_ep_no + 1

    create_query = """
    MATCH (u:User {id: $user_id})
    MATCH (u)-[:HAS_MEM]->(m:EpisodicMemory) WITH m
    CREATE (m)-[:HAS_EP]->(e:Episode {ep_no: $new_ep_no, startTimeStamp: datetime(), endTimeStamp:
datetime(), chat: []})
    RETURN e
    """

    graph.run(create_query, user_id=user_id, new_ep_no=new_ep_no)
```

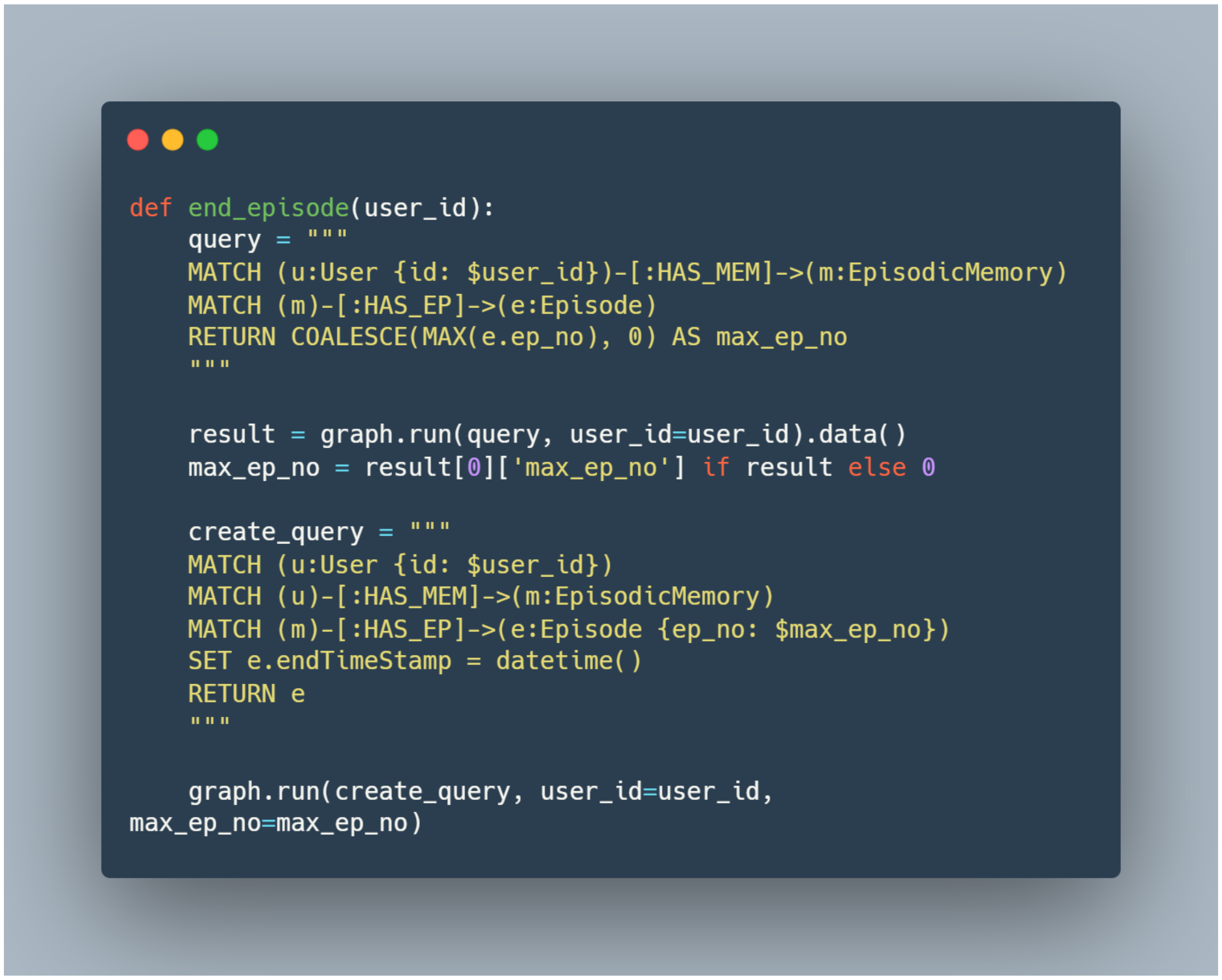
Adding Chat Data:



```
def add_chat(user_id, user_prompt, bot_reply):
    query = """
    MATCH (u:User {id: $user_id})-[:HAS_MEM]->(m:EpisodicMemory)-[:HAS_EP]->(e:Episode)
    WITH e ORDER BY e.ep_no DESC LIMIT 1
    SET e.chat = e.chat + [$user_prompt, $bot_reply]
    SET e.endTimeStamp = datetime()
    RETURN e
    """

    graph.run(query, user_id=user_id, user_prompt=user_prompt, bot_reply=bot_reply)
```

Ending an Episode:



```
def end_episode(user_id):
    query = """
    MATCH (u:User {id: $user_id})-[:HAS_MEM]->(m:EpisodicMemory)
    MATCH (m)-[:HAS_EP]->(e:Episode)
    RETURN COALESCE(MAX(e.ep_no), 0) AS max_ep_no
    """

    result = graph.run(query, user_id=user_id).data()
    max_ep_no = result[0]['max_ep_no'] if result else 0

    create_query = """
    MATCH (u:User {id: $user_id})
    MATCH (u)-[:HAS_MEM]->(m:EpisodicMemory)
    MATCH (m)-[:HAS_EP]->(e:Episode {ep_no: $max_ep_no})
    SET e.endTimeStamp = datetime()
    RETURN e
    """

    graph.run(create_query, user_id=user_id,
max_ep_no=max_ep_no)
```

Communication Protocol

The communication between the ESP32 and the application is handled through a FastAPI server. The data exchange is done in JSON format.

Frontend Interface

The frontend interface allows users to:

- Add new books.
- Register new students.
- Borrow and return books.
- Interact with the chat bot.

Conclusion

IntelliShelf is a comprehensive automated library system that leverages RFID technology and advanced AI to streamline library operations and enhance user experience. The integration of Neo4j for data management ensures efficient storage and retrieval of information, while the AIML and Llama 3 API provide robust chat functionalities.