How Data Flows When a User Posts a Journal Entry

1 Overview

This document explains the data flow when a user submits a journal entry through an application. The process involves a modern tech stack, including FastAPI, Pydantic, and PostgreSQL, to ensure efficient and reliable handling of user data.

2 User Sends a Request

When a user logs their feelings in the app (e.g., "Feeling tired, slight headache today."), the app sends a POST request to the server's `/journal` endpoint. The request includes the entry in JSON format:

```
{
    "entry": "Feeling tired, slight headache today."
}
```

3 FastAPI Receives the Request

The server, built with **FastAPI**, receives the POST request. FastAPI checks the incoming data to ensure it adheres to the expected JSON structure.

4 Pydantic Validates the Data

FastAPI uses **Pydantic** to validate the JSON data, ensuring the `entry` field exists and contains text. Pydantic also adds a `created_at` timestamp. The validated data looks like:

```
{
    "entry": "Feeling tired, slight headache today.",
    "created_at": "2023-10-05T14:30:00Z"
}
```

5 Calling the 'write_journal' Function

With validated data, FastAPI calls the `write_journal` function in `bot/main.py` to process the journal entry further.

6 Detecting Emotions and Symptoms

The `write_journal` function calls `detect_emotions_and_symptoms` to analyze the entry text. For example, it identifies:

- "tired" as the symptom "fatigue"
- "slight headache" as the symptom "headache"

The analysis returns:

```
{
    "symptoms": ["fatigue", "headache"],
    "emotions": ["headache", "fatigue"]
}
```

Note: The example shows overlapping emotions and symptoms, indicating a potential flaw in categorization.

7 Preparing the Data for Storage

The backend prepares the analyzed data for database storage, formatting it as:

```
text = "Feeling tired, slight headache today."
emotion = "fatigue, headache"
timestamp = "2023-10-05T14:30:00Z"
```

8 Building and Executing the SQL Query

An SQL `INSERT` query is constructed to store the data in the `journal_entries` table:

```
INSERT INTO journal_entries (text, emotion, created_at)
VALUES ('Feeling tired, slight headache today.', 'fatigue,
   headache', '2023-10-05T14:30:00Z')
RETURNING id;
```

The `RETURNING id` clause retrieves the unique ID of the new row. The query is executed using the **asyncpg** driver for asynchronous database communication.

9 Database Saves the Data

The PostgreSQL database processes the query, saves the entry, and assigns a unique ID (e.g., 42), which is returned to the server.

10 FastAPI Sends a Response

FastAPI responds to the user with a `201 Created` status and a JSON payload containing the entry details and analysis:

```json
{
    "id": 42,
    "entry": "Feeling tired, slight headache today.",
    "analysis": {
        "symptoms": ["fatigue", "headache"],
        "emotions": ["headache", "fatigue"]
    }
}
```

11 Summary of the Flow

The data flow can be summarized as follows:

1. User submits a journal entry via POST request.
2. FastAPI validates the data using Pydantic.
3. A timestamp is added, and the entry is processed.
4. Text is analyzed for symptoms and emotions.
5. Data is saved to the database via an SQL query.
6. The database returns the new entry's ID.
7. FastAPI sends a success response with the ID and analysis.

This process is optimized for speed, leveraging FastAPI and asyncpg for efficient handling.

12 Conclusion

The journal entry system ensures seamless data flow from user input to database storage, with robust validation and analysis. For further details on any step, feel free to ask!