**Fix Synchronization Issues:**

* **Ensure Unique Identifiers: Both the Google Sheet and the database should have a unique identifier (e.g., id) for each row. This helps track changes and maintain data consistency.**
* **Two-Way Sync Logic: Make sure your sync logic covers both directions:**
  + **Google Sheets → Database: Detect changes in Google Sheets and apply them to the database.**
  + **Database → Google Sheets: Detect changes in the database and apply them to Google Sheets.**
* **Avoid Duplicate Entries: Implement logic to avoid inserting duplicate rows. This often involves checking if a record already exists before inserting.**
* **Concurrency Handling: If changes can happen concurrently in both Google Sheets and the database, consider implementing mechanisms to handle conflicts (e.g., using timestamps or change logs).**

**Best Way to Go Forward:**

* **Refactor and Modularize:**
  + Refactor your current code to separate CRUD operations into individual modules.
  + Create a central sync.py file that coordinates synchronization using these modules.
* **Debug Synchronization Issues:**
  + Log the changes being detected and applied to see where things might be going wrong.
  + Ensure that data is not duplicated or overwritten unintentionally.
* **Test Step-by-Step:**
  + Start by testing CRUD operations on both Google Sheets and the database individually.
  + Test one-way synchronization first (e.g., Google Sheets to the database), then the other way around.
  + Finally, test full two-way synchronization with conflict resolution.

If you share your current code structure or specific errors you're encountering, I can help you further troubleshoot and refine the synchronization logic.

In manual\_sync\_CRUD.py:

Does manuad crud operations

Data\_sync.py does the bidirectional sync between google sheet and db

On running manual\_sync\_CRUD.py:

 Add an entry in Google Sheets and the database.

 Read the data from both sources.

 Update the first row in both sources.

 Delete the first row in both sources.

On running data\_sync.py

 Read data from Google Sheets and synchronize it with the database.

 Clear the existing data in Google Sheets.

 Write the latest data from the database back to Google Sheets.

**Roadmap**

1. **Setup Flask Project**
2. csv\_importer/
3. ├── app/
4. │ ├── \_\_init\_\_.py
5. │ ├── routes.py
6. │ └── templates/
7. │ ├── base.html
8. │ └── upload.html
9. ├── venv/
10. ├── config.py
11. ├── run.py
12. └── README.md
13. **Google Sheets API**
    * Set up Google Cloud project.
    * Enable Google Sheets API.
    * Generate OAuth 2.0 credentials.
14. **Backend**
    * Implement file upload route.
    * Parse uploaded CSV files.
    * Allow column selection and data filtering.
    * Integrate with Google Sheets API.
15. **Frontend**
    * Create a user-friendly interface.
    * Implement drag-and-drop file uploads.
    * Create UI for column selection and filtering.
16. **Production Checklist**
    * Optimize for large CSV files ✔️
    * Test with different file sizes ✔️
    * Ensure column selection and filtering work ✔️
    * Documentation ✔️
    * Video Recording ✔️

**Approach**

**Technology Stack:** I chose to build this project using Flask, a lightweight and flexible web framework for Python. Additionally, I used HTML, CSS, and JavaScript for the frontend to create a user-friendly interface.

**Design Decisions:**

* **User Interface:** I designed a simple and intuitive user interface using HTML and CSS. Users can drag and drop CSV files onto the web application, making it easy and familiar for them.
* **Backend Processing:** When a user drops a CSV file, the Flask backend handles file upload, validates the CSV, and presents the user with options to select columns for import. I designed this flow to be step-by-step and user-friendly.
* **CSV Import to Google Sheets:** To import data into Google Sheets, I used the Google Sheets API, and I provided users with the option to either append data to an existing sheet or create a new one.
* **Error Handling:** I implemented robust error handling to provide clear feedback to users in case of issues such as invalid CSV files or API errors.

**Challenges:**

* **Performance:** Optimizing the application to handle large CSV files without causing performance issues was a concern. I implemented efficient data processing to mitigate this.

**Features:**

* **Drag-and-Drop:** Users can easily upload CSV files by dragging and dropping them onto the web application.
* **Column Mapping:** Users have the option to map CSV columns to Google Sheets columns, ensuring flexibility in data import.
* **Data Filtering (Brownie Points):** I added a feature that allows users to filter data before importing it into Google Sheets. This can help streamline the import process and reduce unnecessary data.

**Testing:** I thoroughly tested the application to ensure its functionality and reliability. Test cases included:

* Uploading valid and invalid CSV files.
* Testing various column mapping scenarios.
* Importing data into existing and new Google Sheets.
* Performance testing with large CSV files to ensure the application remains responsive.

**Final Thoughts:** Building this CSV Importer for Google Sheets using Flask was a rewarding experience. It addresses a practical problem faced by data analysts and offers a user-friendly solution. I enjoyed the challenge of integrating with the Google Sheets API and ensuring the application's reliability and performance. This project has reinforced my skills in web development, API integration, and user interface design.

**Checklist**

* My code's working just fine! 🥳
* I have recorded a video showing it working and embedded it in the README ▶️
* I have tested all the normal working cases 😎
* I have even solved some edge cases (brownie points) 💪
* I added my very planned-out approach to the problem at the end of this README 📜