studio platform documentation

Test-Frontend Folder (File Upload Interface)

The **test-frontend folder** contains a simple, static HTML page that allows users to test the file upload functionality. This front-end is used specifically for testing purposes, providing a user interface for file uploads in XLSX format.

Core Responsibilities:

- **File Upload**: Provides an HTML form to allow users to upload files. The uploaded files are then sent to the backend for processing.
- **Basic Styling**: The front-end uses basic **HTML** and **CSS** to create a simple and functional interface for testing.
- **File Format Handling**: It specifically supports the uploading of **XLSX files**, which are used to test the backend file processing pipeline.

Test-Frontend Folder Structure:

1. **HTML File (upload.html)**: Contains the file upload form where users can select and upload their XLSX files. The form submits the file to the backend for processing.
2. **CSS File (styles.css)**: Provides basic styling for the HTML form, making the file upload process visually clear and user-friendly.
3. **JavaScript** : If needed, this file may handle additional functionalities like form validation, progress bars, or user feedback during the upload process.

Backend Overview

The backend is composed of several components, including configuration files, server management, data handling scripts, and a series of Python files for processing uploaded Excel files. The server handles the entire processing pipeline for the uploaded data and communicates with the frontend for data uploads. Below is an explanation of the structure and responsibilities of each component in the backend.

1. Configuration Files

- **docker-compose.yml**: This file is used to configure and manage Docker containers. It defines the services, networks, and volumes needed for the backend, ensuring that the application runs in a containerized environment. It is standard and does not require modification.
- **package.json & package-lock.json**: These files define the dependencies and configuration for the Node.js environment, which handles the server operations. No changes are made to these files.

2. Server Management (server.js)

The server.js file is responsible for managing the backend server. It handles the requests and responses and coordinates the data processing tasks by invoking the various **data processing files**. It serves as the central controller, making sure that the correct files are executed in the proper sequence.

- **Responsibilities**:
  - Handles incoming requests (likely file uploads or API calls).
  - Coordinates the execution of the data processing scripts (i.e., a_org.js, b_balance.js, etc.).
  - Manages server operations, including starting and stopping the server.

3. Data Processing Files (a_org.js, b_balance.js, etc.)

The backend processes and organizes data into different categories for insertion into the database. Each of the following files is responsible for populating a specific table in the database:

- **a_org.js**: Handles the population of the organization table in the database.
- **b_balance.js**: Responsible for inserting data into the balance_sheet table.
- **c_profit_loss.js**: Populates the profit_loss table.
- **d_cash.js**: Handles the cash_statements table data insertion.
- **e_ratio.js**: Manages the population of the ratios table.

Each of these files extracts, processes, and formats the data from the uploaded files, ensuring it fits the expected database structure before it's inserted into the respective tables.

4. Python Files for Excel Processing

The backend also includes a set of **15 Python files** dedicated to processing and converting the Excel files uploaded by the frontend. These Python scripts handle everything from renaming the uploaded files to restructuring and cleaning the data before it's ready to be stored in the database.

Here’s a breakdown of what each Python script does:

1. **0_rename.py**: Renames the uploaded Excel file from the frontend to a consistent naming convention, helping with file management and processing.
2. **1_input_excel.py**: Detects the sheets present in the uploaded Excel file and separates them accordingly. If multiple sheets are found, this script organizes them for further processing.
3. **2_remap_excel.py**: Reads the index.xlsx file and renames the sheets in the Excel file according to the names provided in the index.xlsx file. This ensures consistency in the sheet names.
4. **3_delete_excel.py**: Removes unnecessary or irrelevant data points and sheets from the Excel file before further processing. This is useful for cleaning up the uploaded files.
5. **4_detect_table_profiles.py**: Detects and identifies the company profile table within the Excel file. This table contains important data about the company being processed.

6. **5_detect_table.py**: Detects the remaining tables in the Excel file that are relevant for processing (e.g., financial tables, balance sheets, etc.).
7. **6_profile_extract.py**: Extracts text data from the detected **company profile table**. This data is crucial for populating the profile information in the database.
8. **7_column_checker_profile.py**: Validates and checks the dimensions (width and height) of the **profile table**, ensuring that the data is correctly formatted for further processing.
9. **8_column_checker.py**: Similar to 7_column_checker_profile.py, but for the other tables (financial tables, balance sheets, etc.).
10. **9_exceltojson.py**: Converts the cleaned Excel data into a **JSON format** using the **Pandas** library, which simplifies the data for processing and database insertion.
11. **10_exceltojson_profile.py**: Converts the extracted profile data into JSON format, ensuring that it aligns with the database structure.
12. **11_restruct_json.py**: Restructures the extracted JSON data, ensuring that it is in the correct format for database insertion.
13. **12_profile_remap.py**: Specifically remaps the **profile JSON data** to match the database schema before it's ready for upload.
14. **13_final.py**: Performs the final cleaning and setup of the JSON data, making sure that everything is formatted and structured correctly before being sent to server.js for database upload.
15. **14_clear_cache.py**: Clears any residual data or cache that might have been created during the processing of the Excel files, ensuring that future operations are not affected by leftover data.

General Flow of Backend Processing:

1. **File Upload**: The frontend uploads an Excel file (e.g., XLSX) through the API.
2. **File Renaming**: 0_rename.py renames the file to a standardized name.
3. **Sheet Detection and Organization**: 1_input_excel.py detects the sheets in the Excel file and separates them accordingly.
4. **Excel Remapping**: 2_remap_excel.py ensures the sheets are renamed according to the index provided.
5. **Cleaning and Data Removal**: 3_delete_excel.py removes unnecessary data and irrelevant sheets.
6. **Table Detection**: 4_detect_table_profiles.py and 5_detect_table.py identify and extract relevant tables (profile and financial data).
7. **Column Validation**: 7_column_checker_profile.py and 8_column_checker.py check the dimensions of the tables for correctness.
8. **Data Extraction**: 6_profile_extract.py extracts necessary data from the identified tables.
9. **Conversion to JSON**: 9_exceltojson.py and 10_exceltojson_profile.py convert the cleaned data into JSON format for easier processing.
10. **Restructuring**: 11_restruct_json.py and 12_profile_remap.py ensure the extracted JSON aligns with the required structure for database upload.
11. **Final Cleaning**: 13_final.py prepares the final version of the data for upload.
12. **Cache Clearance**: 14_clear_cache.py clears any cache or leftover data.
13. **Database Upload**: The processed data is passed to server.js, which handles the insertion into the relevant database tables (organization, balance sheet, etc.).

Conclusion

This backend system ensures a seamless process for uploading, processing, cleaning, and storing Excel data. With a combination of Node.js and Python scripts, the system transforms raw Excel files into well-structured JSON data, which is then inserted into the appropriate database tables for use in the application.