**PROJECT** : PYTHON APP & RELATIONAL DATABASE

**DEVELOPER** : BRANDON STEINKE

Email: brandon.steinke@yahoo.com | Phone: (415) 271-3377
LinkedIn: https://www.linkedin.com/in/brandon-steinke-2817ba | Tech Portfolio: https://brandino771.github.io

**Overview and screen shots below:**

As an independent contractor I worked for a small company to develop a custom database and user-friendly (standalone) Python desktop application. The app performed the ingestion, and formatting of raw data, database bulk uploads of clean data, and the output of formatted reports from the database. The project started with conceptualizing the database schema with the CEO, which was tricky because the product inventory flow wasn't finalized. The database was developed with automated features, such as triggers that move, copy, delete, and or summarize data and indicate if errors are present in the inventory linear flow. Advanced queries were used to join and subtotal data from saved database views or tables. The app can ingest CSV or online XML data. For CSV sources the user places raw CSV files in a designated folder, which is processed, renamed, and moved to a processed folder when complete. For XML the user completes a form on the web page. Prior to database upload, the raw data is extracted, formatted, filtered for duplicates. Any errors found in the raw data are output to a designated folder as a CSV error log (with data prepended to the top of existing file). After upload, a database log generates as a text file (prepending data to the top of existing log file, which pops up in MS Notepad on the user's screen) that indicates the number of records uploaded, upload time, data rejected. This database log also generates within the web page as another source of user feedback. This log was especially useful during development where I noticed the upload job was hanging for minutes instead of seconds ( of course it all came down to one line of incorrect code in a database trigger).

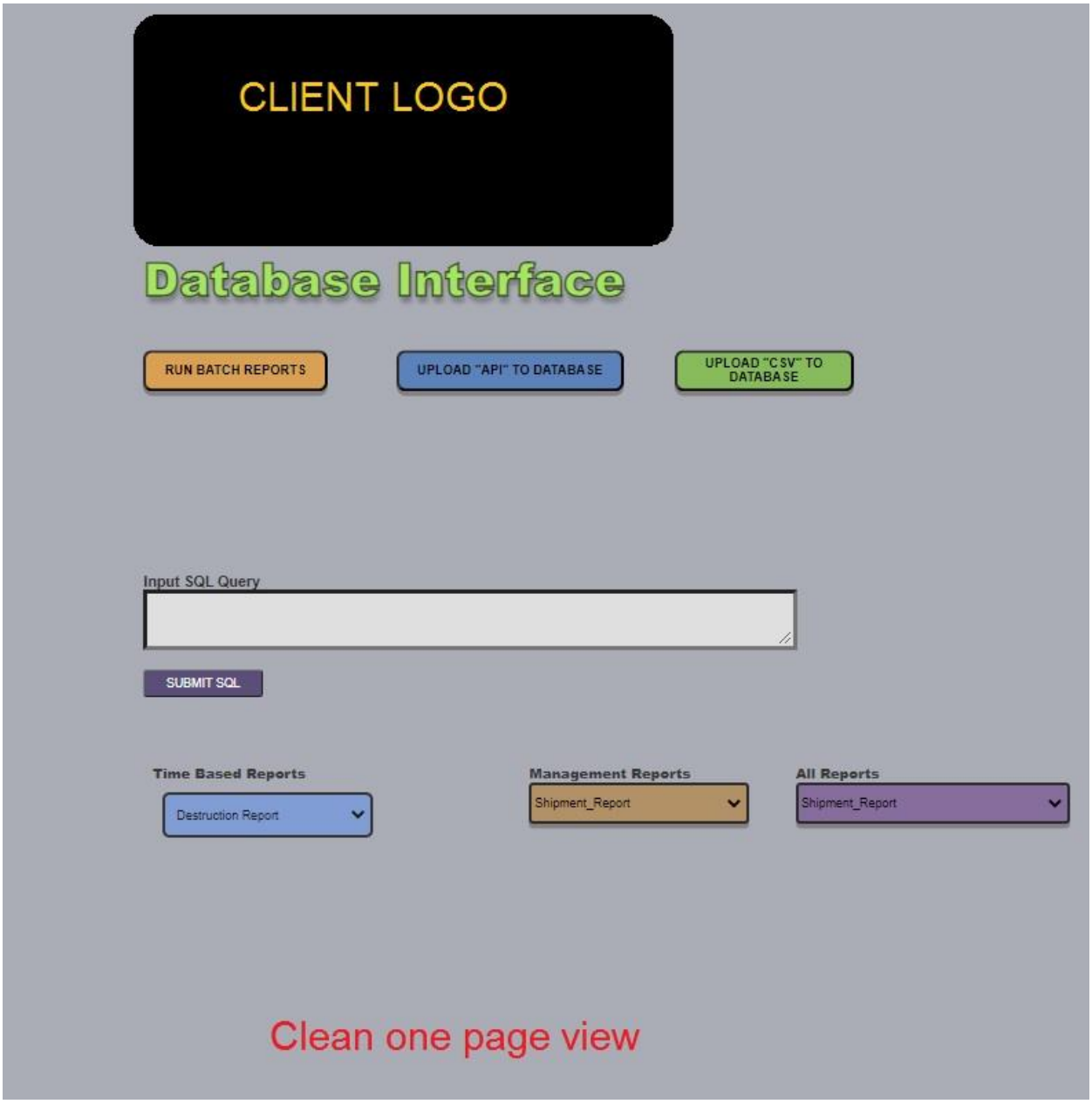
A local web page is UI where the user controls the data going in and out with easy-to-use buttons and forms. Data from the database can output directly into the page in formatted HTML tables and then be downloaded as PDF or CSV. The request data the user can either input custom SQL queries directly into a text field or select reports from drop down menus. Further the user can run batch CSV reports by opening a preformatted CSV file, selecting the desired reports to run, then save, and click the "Run Batch Reports" on the web page to quickly output individual reports to designated folder. Further I integrated a JavaScript pdf library so that HTML table data could be output as PDFs directly from the web page. I had trouble implementing the library features and ended up coding my own custom solution to utilize the library to calculate characters per line, per page, page layout, page size and headers, and page numbers. Further I provided 10 pages of 'how to' documentation for reference.

**Features:**

- Relational database for inventory mgmt w/ one-page UI web interface, one click data reporting & upload.
- Wrote versatile Python ETL script to ingest and transform online XML or local CSV data, with DB upload & error log reporting.
- Automated ETL - with duplicate filtering, data formatting, writing to offline logs, and bulk data uploads
- JavaScript implemented to listen, and process user HTML query requests to and from REST PYTHON API.
- DB results dynamically output to formatted HTML tables, with CSV & PDF download options.
- Database (SQLITE3) - w/ automated triggers (conditional / procedural) saved views/queries for client ease
- Tech Stack: Python, SQL, SQLite3, SQLAlchemy, Flask, JavaScript, D3, HTML, CSS, REST API

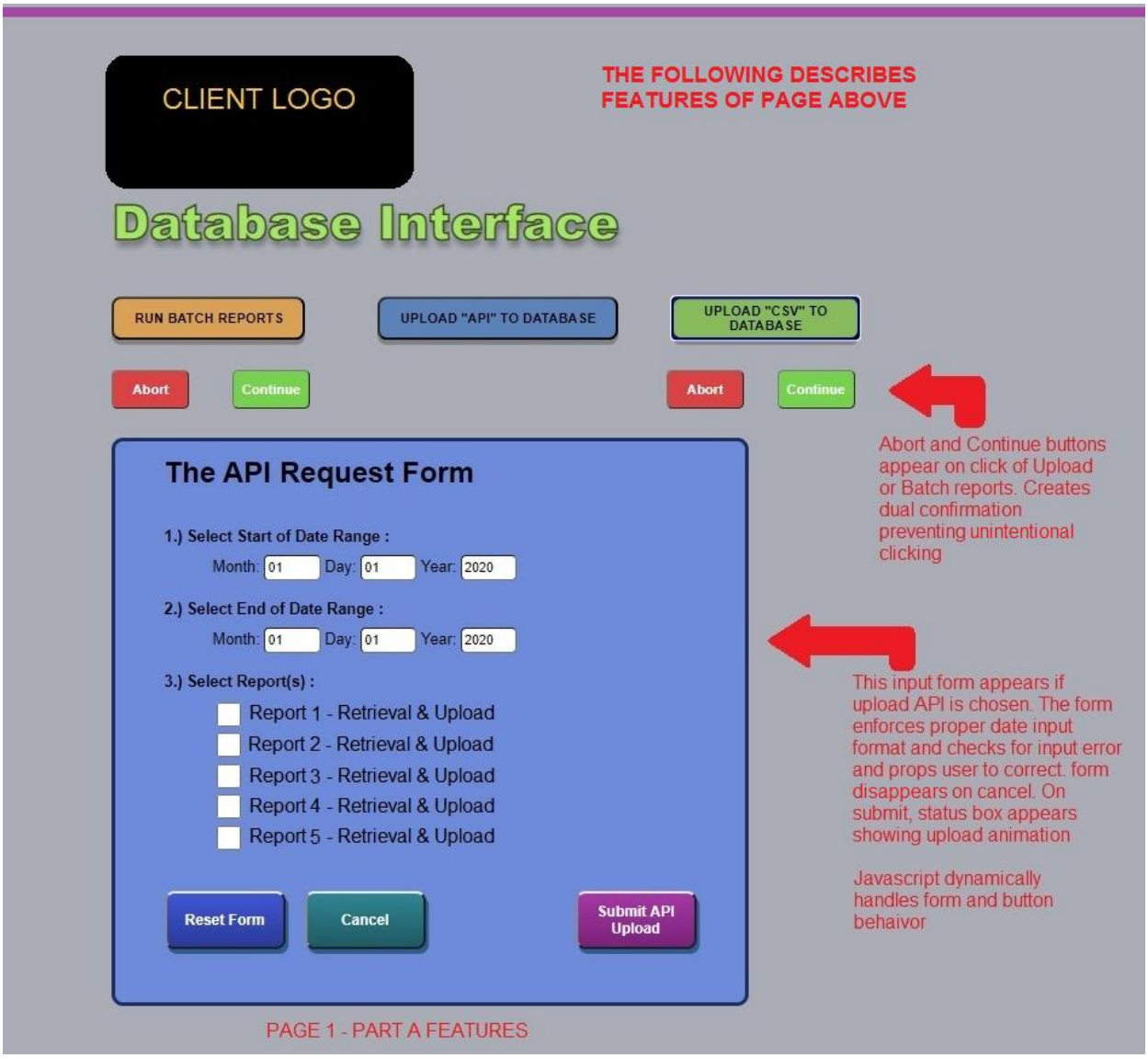
**SCREEN SHOT 1 :**

This is the view of the web page on initial load.  It is clean and slim. Most features and forms are hidden until the user clicks on a button. The page will expand as forms appear or data is populated in tables and contracts as tables are deleted or features are deactivated.

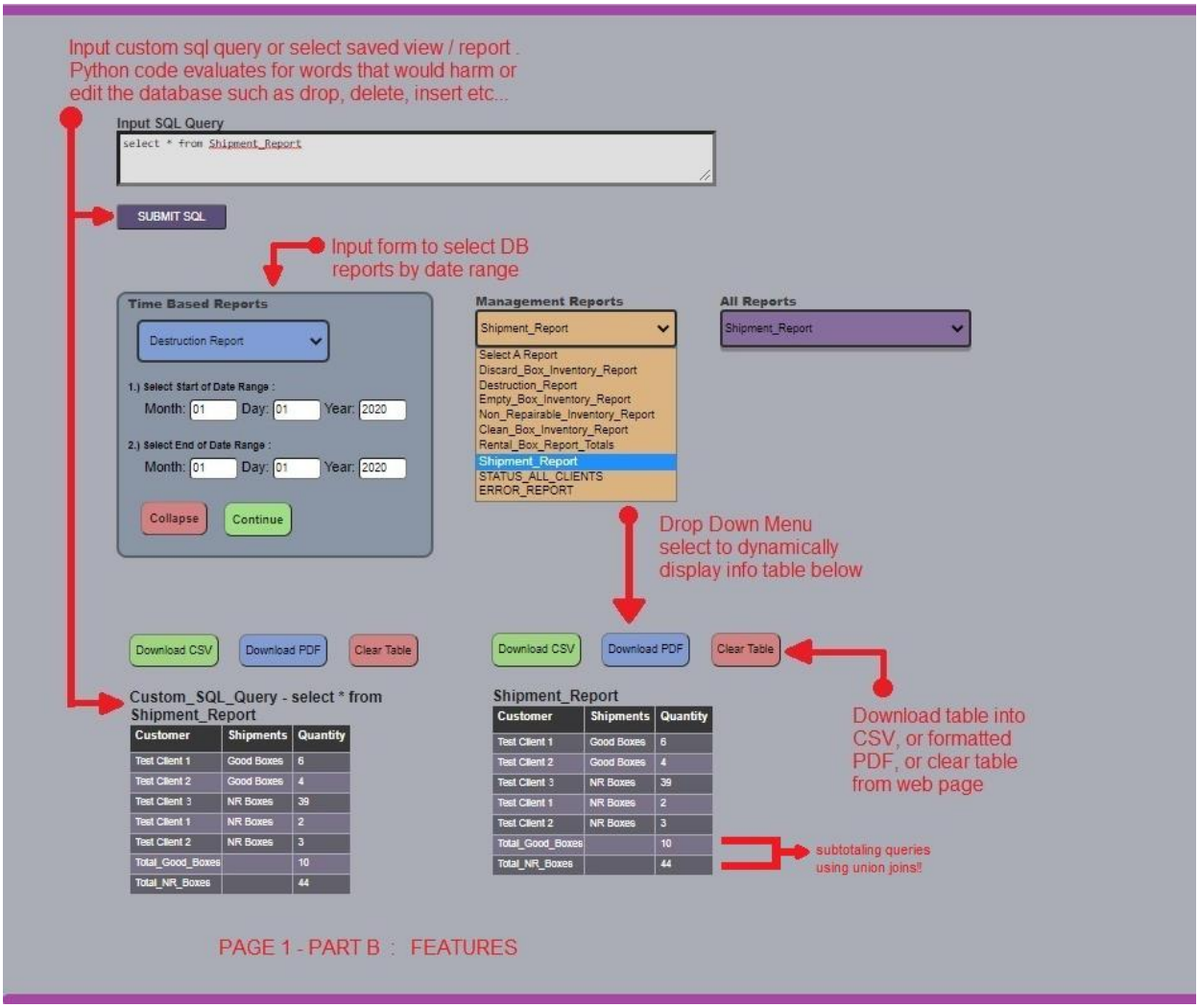


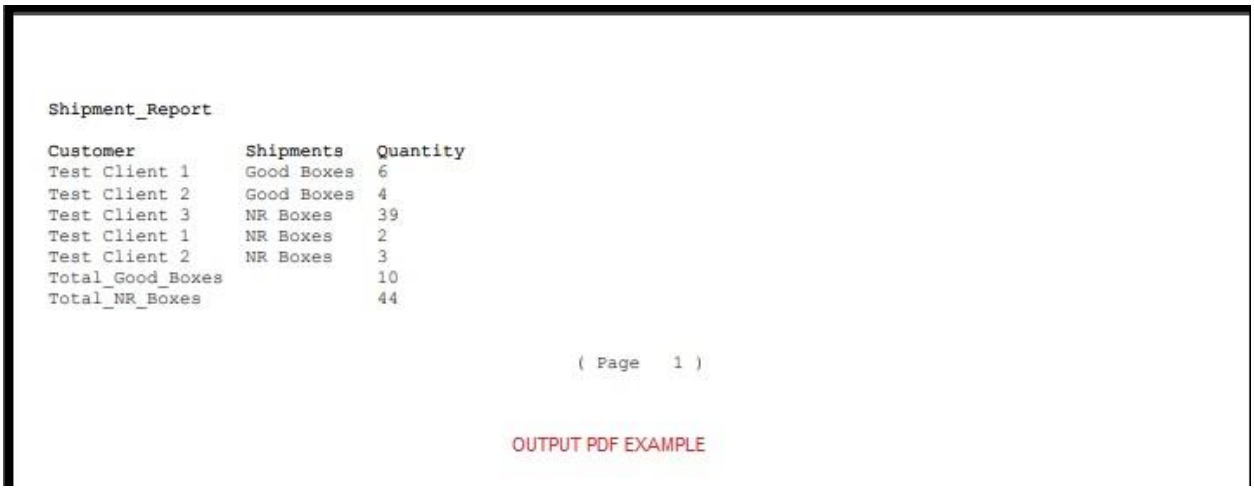
**SCREEN SHOT 2 :**

Here the top three features are fully expanded. **Run Batch Reports, Upload API to Database, Upload CSV to Database.** A lot of effort went into the code for the API request form so the dates would not be accepted if input incorrectly. The user was given text-based error messages if the form was incorrect.



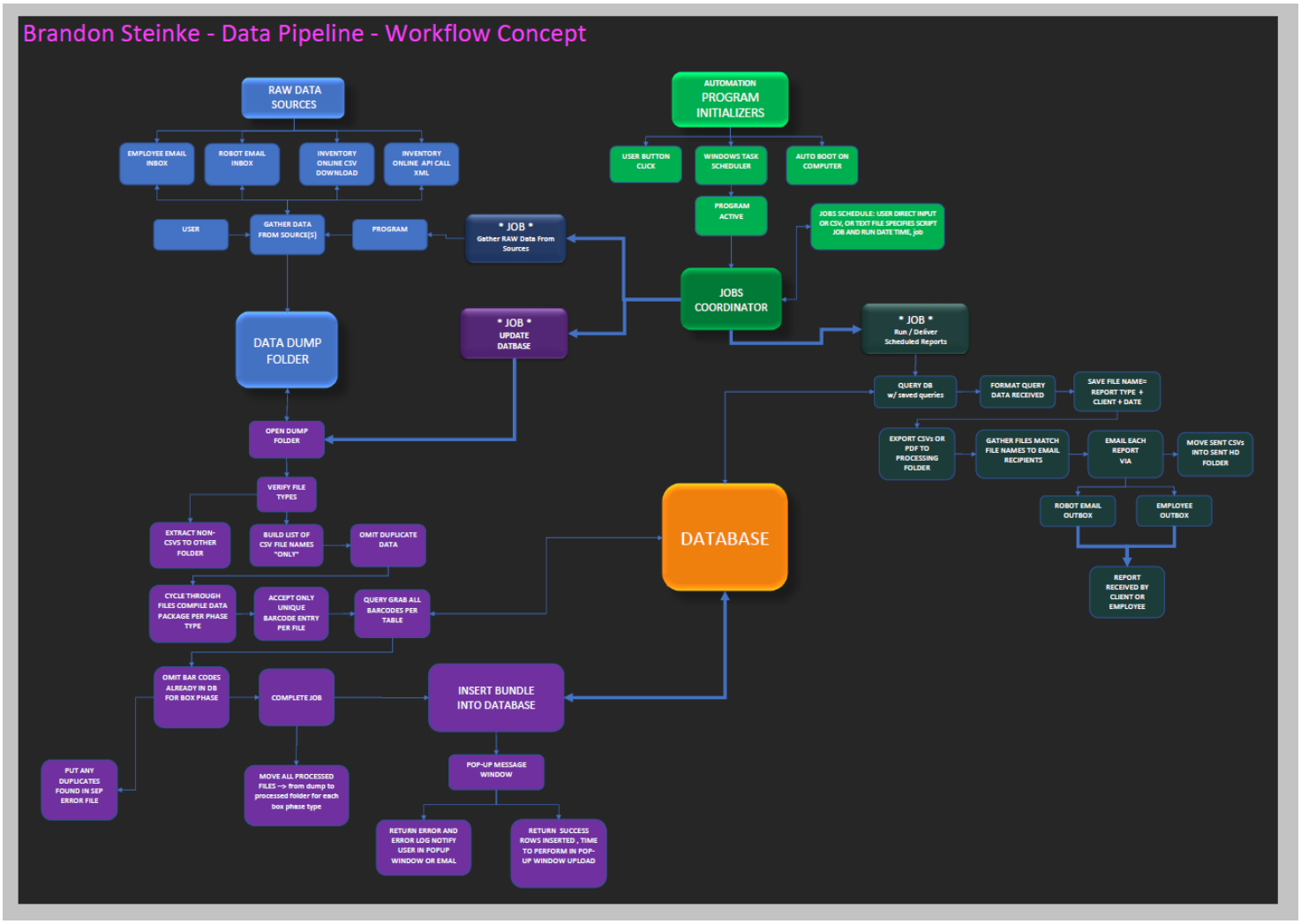
**SCREEN SHOT 2 :** Remaining features are explained below, **Custom Query Input, Time Based Reports, Management Reports, All Reports.** **SCREEN SHOT 3 :** Bottom of page is an example of a PDF report.



**SCREEN SHOT 4 :**

Below is the workflow concept that included heavy automation. Development never reached the fully automated state, but all of the blue “Raw Data Sources” and purple “Update Database” workflow features below were implemented. Please zoom in to read.




Thanks for viewing this project !