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 for a small company, I developed 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 and 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).

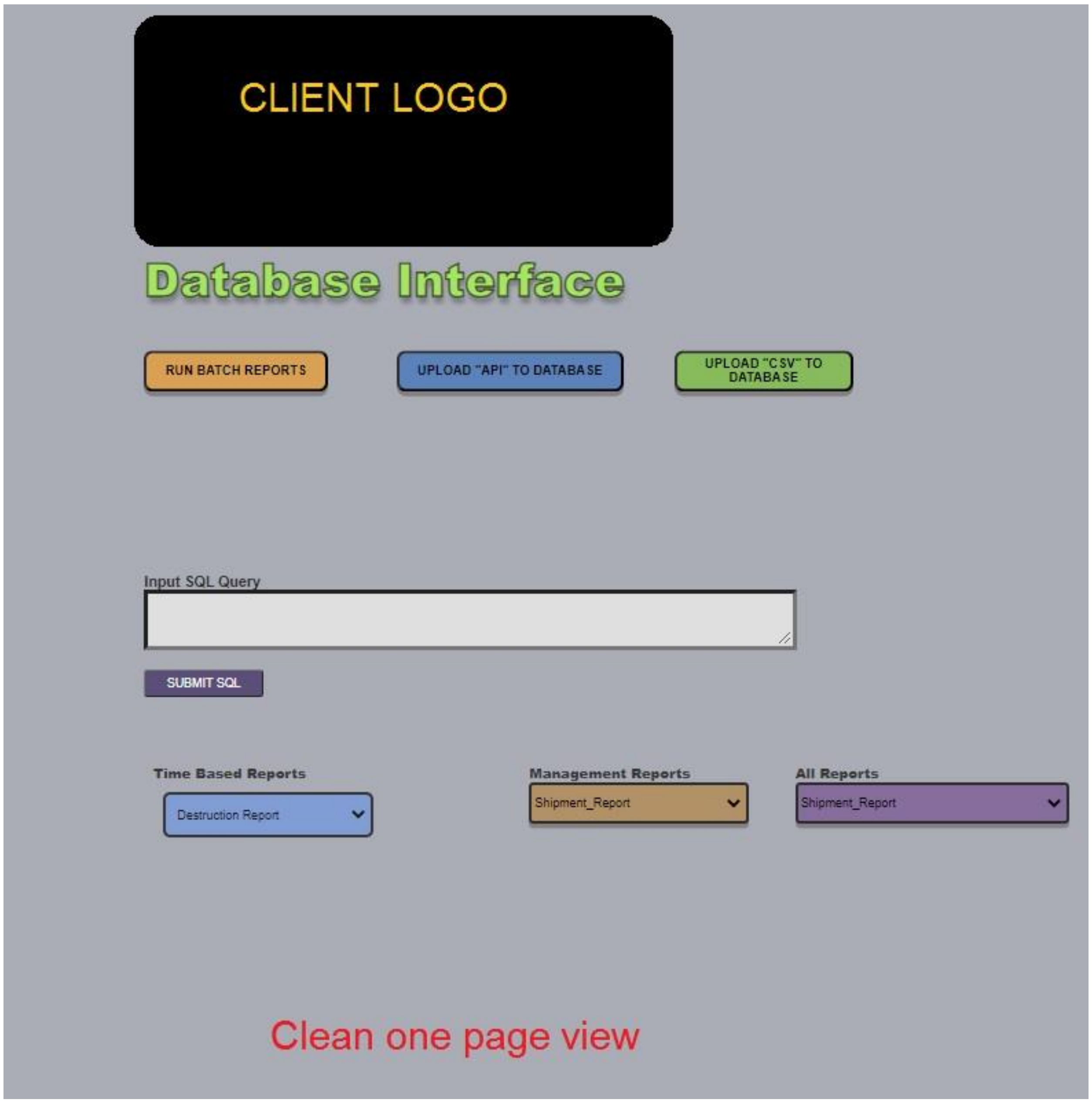
A local web page is the UI where the user controls the data going in and out with easy-to-use buttons and forms. Data from the database can output directly into the page in formatted HTML tables and then be downloaded as PDF or CSV. The user can request data by either custom SQL queries input directly 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.

Retrospective:

Going into this project I had some experience with Python, Flask, and SQL Alchemy to serve as the REST API and database conduit, with JavaScript as the requester and presenter of the data. However, I had no experience creating a Python executable application, so there was a learning curve with organizing the project and assigning various folder paths (but was surprisingly pain free). For the database, I had to brush up on my skills by taking a SQL essentials course with SQLite, which is the format we went with. The company did not want cloud capability and security was not an issue, and from my research found SQLite could handle a lot of data rapidly (further the company could use DB Browser in addition to the app to access data and my database setup code). Per how the CEO wanted the data to migrate from table to table I researched and tested database triggers, bulk upload techniques and speeds, prior to committing to the db design and taking the job. Further I integrated a prior side project for the UI and evolved that further. I enjoyed developing the automation parts, as well as the summary table in the database for "at a glance" view of all important details for each inventory item. All the technology used was open source. To complete the project, I pulled from my prior tech project experience, and game development methodology to test, iterate, and connect all the features into a workable application.

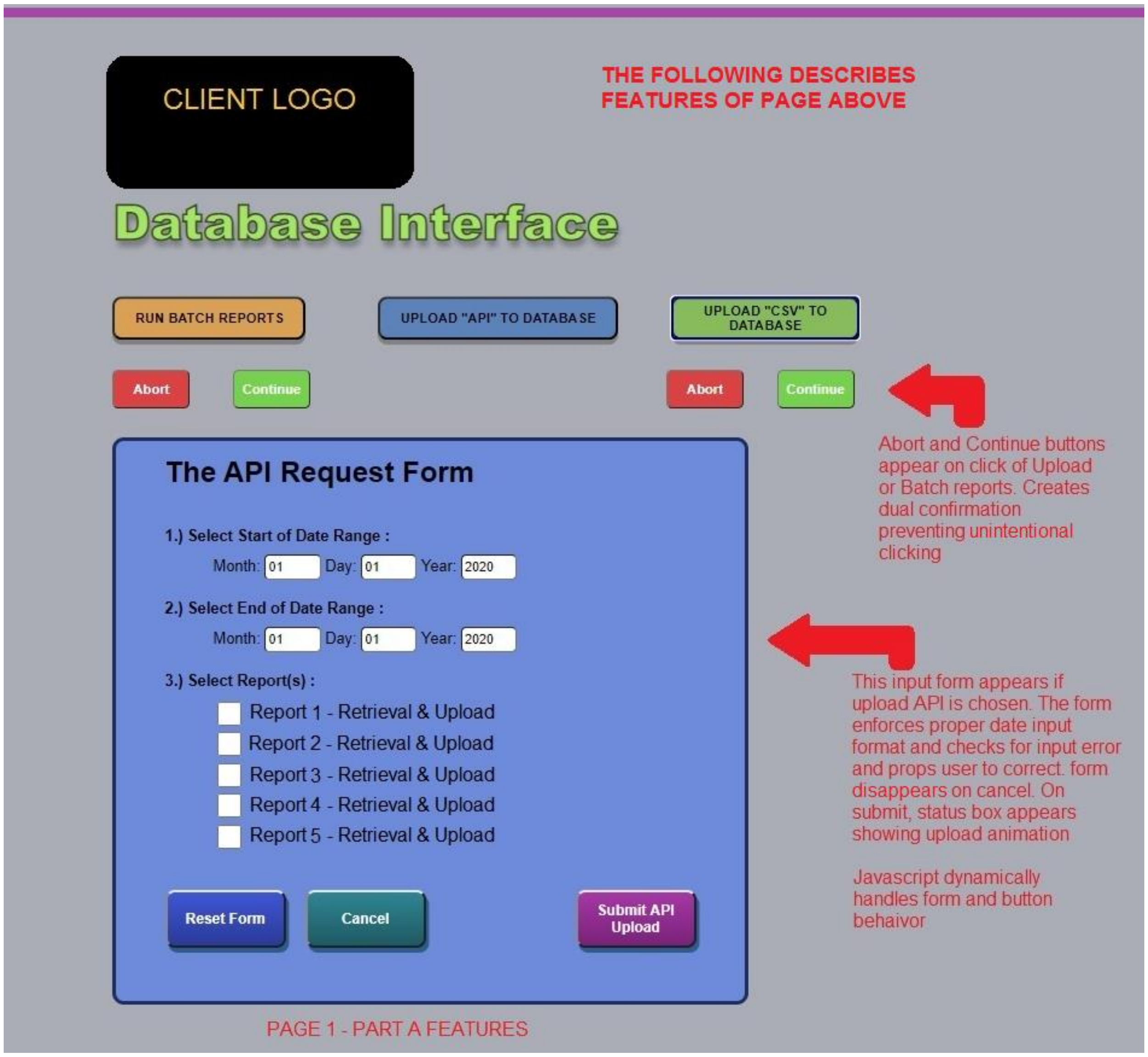
SCREEN SHOT 1 :

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.



The API Request Form

1.) Select Start of Date Range :
Month: 01 Day: 01 Year: 2020

2.) Select End of Date Range :
Month: 01 Day: 01 Year: 2020

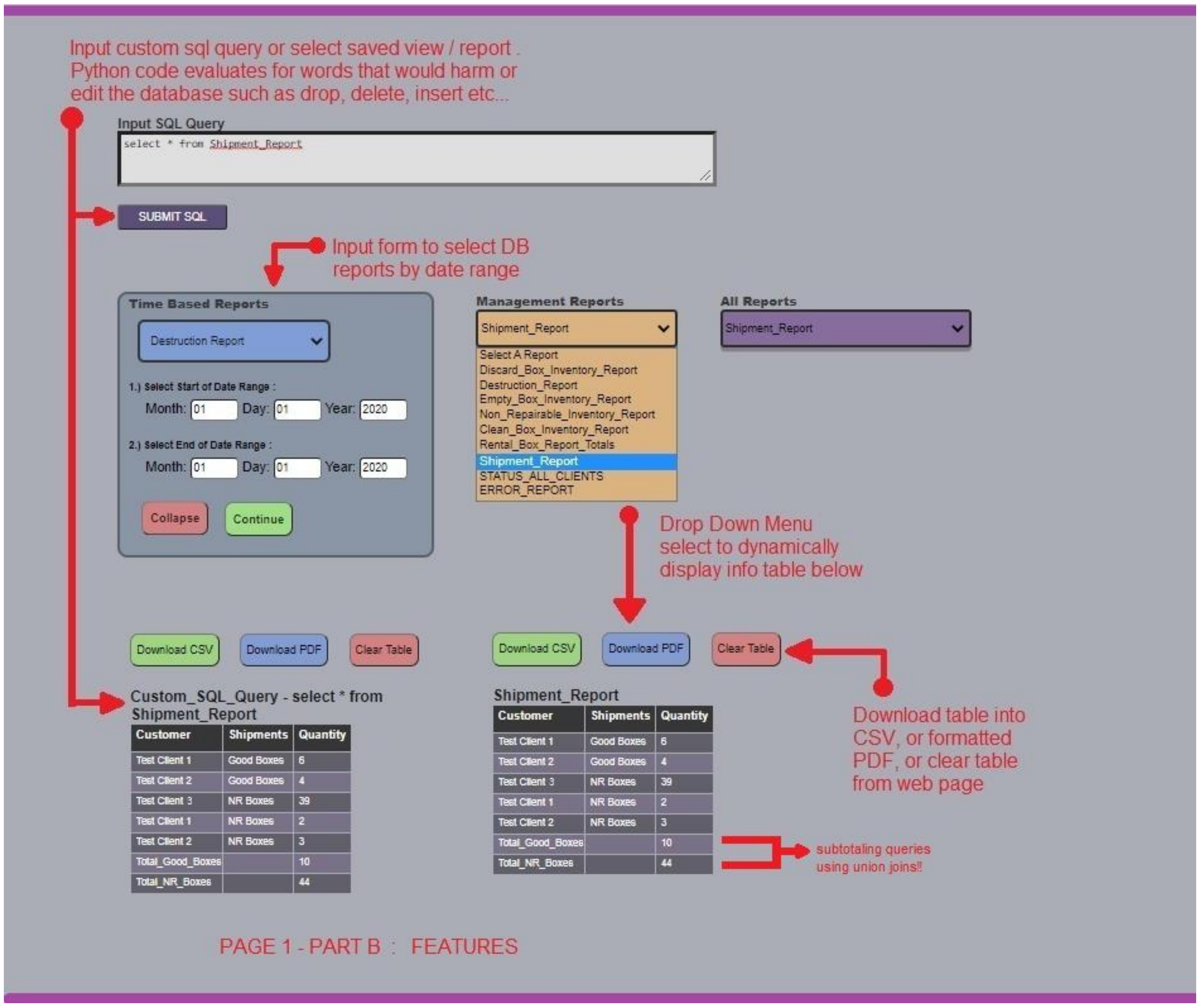
3.) Select Report(s) :
- ☐ Report 1 - Retrieval & Upload
- ☐ Report 2 - Retrieval & Upload
- ☐ Report 3 - Retrieval & Upload
- ☐ Report 4 - Retrieval & Upload
- ☐ Report 5 - Retrieval & Upload

Abort and Continue buttons appear on click of Upload or Batch reports. Creates dual confirmation preventing unintentional clicking

This input form appears if upload API is chosen. The form enforces proper date input format and checks for input error and props user to correct. form disappears on cancel. On submit, status box appears showing upload animation

Javascript dynamically handles form and button behaivor

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.


Input custom sql query or select saved view / report . Python code evaluates for words that would harm or edit the database such as drop, delete, insert etc...

Input SQL Query

```
select * from Shipment_Report
```

SUBMIT SQL
Input form to select DB reports by date range
Time Based Reports
Destruction Report
1.) Select Start of Date Range :
Month: 01 Day: 01 Year: 2020
2.) Select End of Date Range :
Month: 01 Day: 01 Year: 2020
Collapse
Continue
Management Reports
Shipment_Report
Select A Report
Discard_Box_Inventory_Report
Destruction_Report
Empty_Box_Inventory_Report
Non_Repairable_Inventory_Report
Clean_Box_Inventory_Report
Rental_Box_Report_Totals
Shipment_Report
STATUS_ALL_CLIENTS
ERROR_REPORT
All Reports
Shipment_Report
Download CSV
Download PDF
Clear Table

Drop Down Menu select to dynamically display info table below

Custom_SQL_Query - select * from Shipment_Report

Customer	Shipments	Quantity
Test Client 1	Good Boxes	6
Test Client 2	Good Boxes	4
Test Client 3	NR Boxes	39
Test Client 1	NR Boxes	2
Test Client 2	NR Boxes	3
Total_Good_Boxes		10
Total_NR_Boxes		44

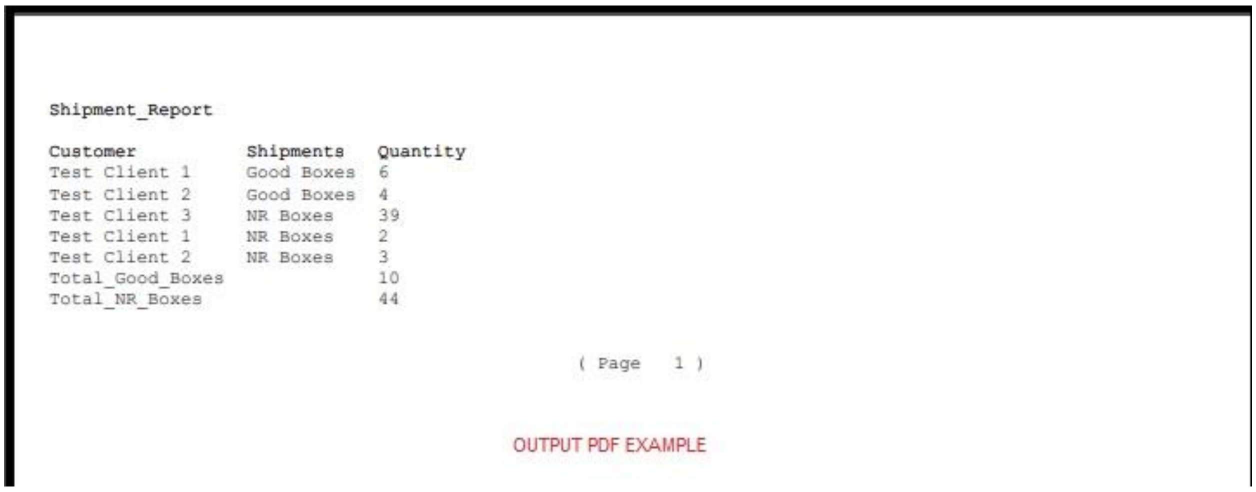
Shipment_Report

Customer	Shipments	Quantity
Test Client 1	Good Boxes	6
Test Client 2	Good Boxes	4
Test Client 3	NR Boxes	39
Test Client 1	NR Boxes	2
Test Client 2	NR Boxes	3
Total_Good_Boxes		10
Total_NR_Boxes		44

Download table into CSV, or formatted PDF, or clear table from web page

subtotaling queries using union joins!!

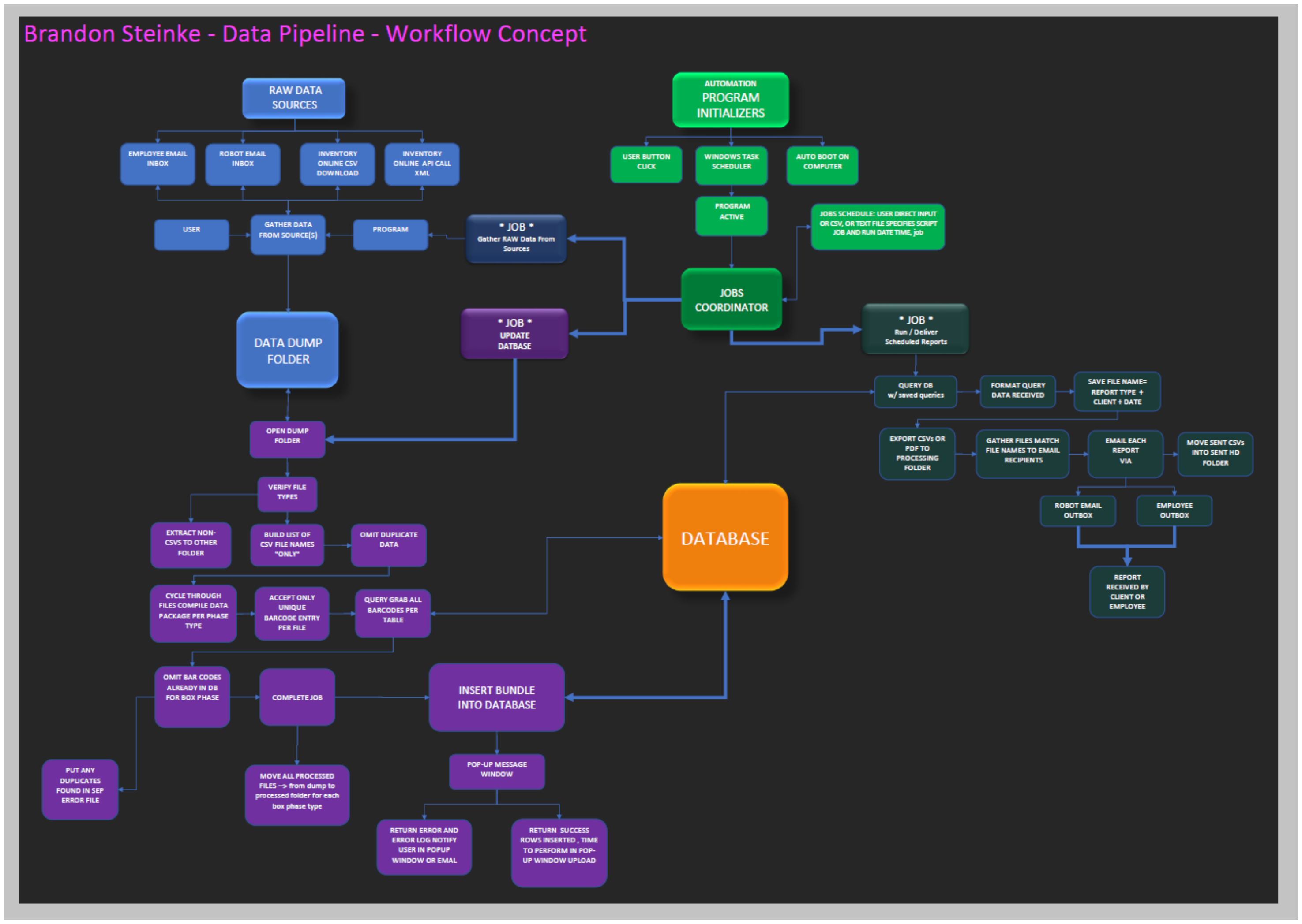
PAGE 1 - PART B : FEATURES


```
Shipment_Report

Customer          Shipments    Quantity
Test Client 1     Good Boxes   6
Test Client 2     Good Boxes   4
Test Client 3     NR Boxes     39
Test Client 1     NR Boxes     2
Test Client 2     NR Boxes     3
Total_Good_Boxes               10
Total_NR_Boxes                 44

                                         ( Page   1 )
```

OUTPUT PDF EXAMPLE

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.


Brandon Steinke - Data Pipeline - Workflow Concept
RAW DATA SOURCES
EMPLOYEE EMAIL INBOX
ROBOT EMAIL INBOX
INVENTORY ONLINE CSV DOWNLOAD
INVENTORY ONLINE API CALL XML
USER
GATHER DATA FROM SOURCE(S)
PROGRAM
* JOB * Gather RAW Data From Sources
DATA DUMP FOLDER
* JOB * UPDATE DATBASE
OPEN DUMP FOLDER
VERIFY FILE TYPES
EXTRACT NON-CSVS TO OTHER FOLDER
BUILD LIST OF CSV FILE NAMES "ONLY"
OMIT DUPLICATE DATA
CYCLE THROUGH FILES COMPILE DATA PACKAGE PER PHASE TYPE
ACCEPT ONLY UNIQUE BARCODE ENTRY PER FILE
QUERY GRAB ALL BARCODES PER TABLE
OMIT BAR CODES ALREADY IN DB FOR BOX PHASE
COMPLETE JOB
INSERT BUNDLE INTO DATABASE
PUT ANY DUPLICATES FOUND IN SEP ERROR FILE
MOVE ALL PROCESSED FILES --> from dump to processed folder for each box phase type
POP-UP MESSAGE WINDOW
RETURN ERROR AND ERROR LOG NOTIFY USER IN POPUP WINDOW OR EMAL
RETURN SUCCESS ROWS INSERTED , TIME TO PERFORM IN POP-UP WINDOW UPLOAD
AUTOMATION PROGRAM INITIALIZERS
USER BUTTON CLICK
WINDOWS TASK SCHEDULER
AUTO BOOT ON COMPUTER
PROGRAM ACTIVE
JOBS SCHEDULE: USER DIRECT INPUT OR CSV, OR TEXT FILE SPECIFIES SCRIPT JOB AND RUN DATE TIME, job
JOBS COORDINATOR
* JOB * Run / Deliver Scheduled Reports
QUERY DB w/ saved queries
FORMAT QUERY DATA RECEIVED
SAVE FILE NAME= REPORT TYPE + CLIENT + DATE
EXPORT CSVs OR PDF TO PROCESSING FOLDER
GATHER FILES MATCH FILE NAMES TO EMAIL RECIPIENTS
EMAIL EACH REPORT VIA
MOVE SENT CSVs INTO SENT HD FOLDER
ROBOT EMAIL OUTBOX
EMPLOYEE OUTBOX
REPORT RECEIVED BY CLIENT OR EMPLOYEE
DATABASE

Thanks for viewing this project !