***Analysis and Design Inspection***

Algorithmic Trading Software

2024-03-27

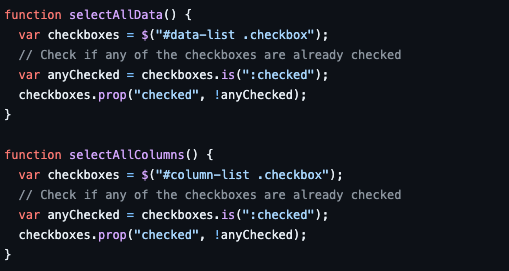
Version 1.0

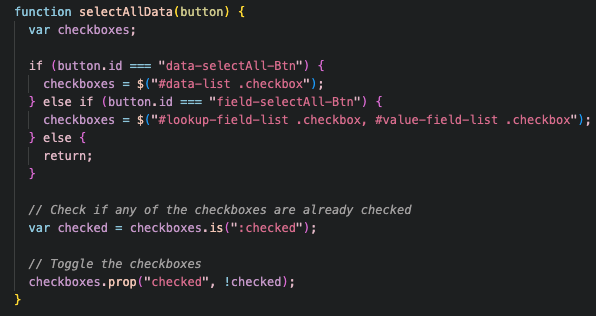
Authors: Jacob Rawlings,Jake Fischer**,** Alan Abdollahzadeh, Benjamin Carrier, Vanessa Dubouzet, Dominic Presch

## Source Code: (Vanessa)

### script.js

* Instead of two separate functions for the selectAll button for the data\_export.html, it is refactored into one function.
* Also renamed the #column-list to #lookup-field-list for clarity.





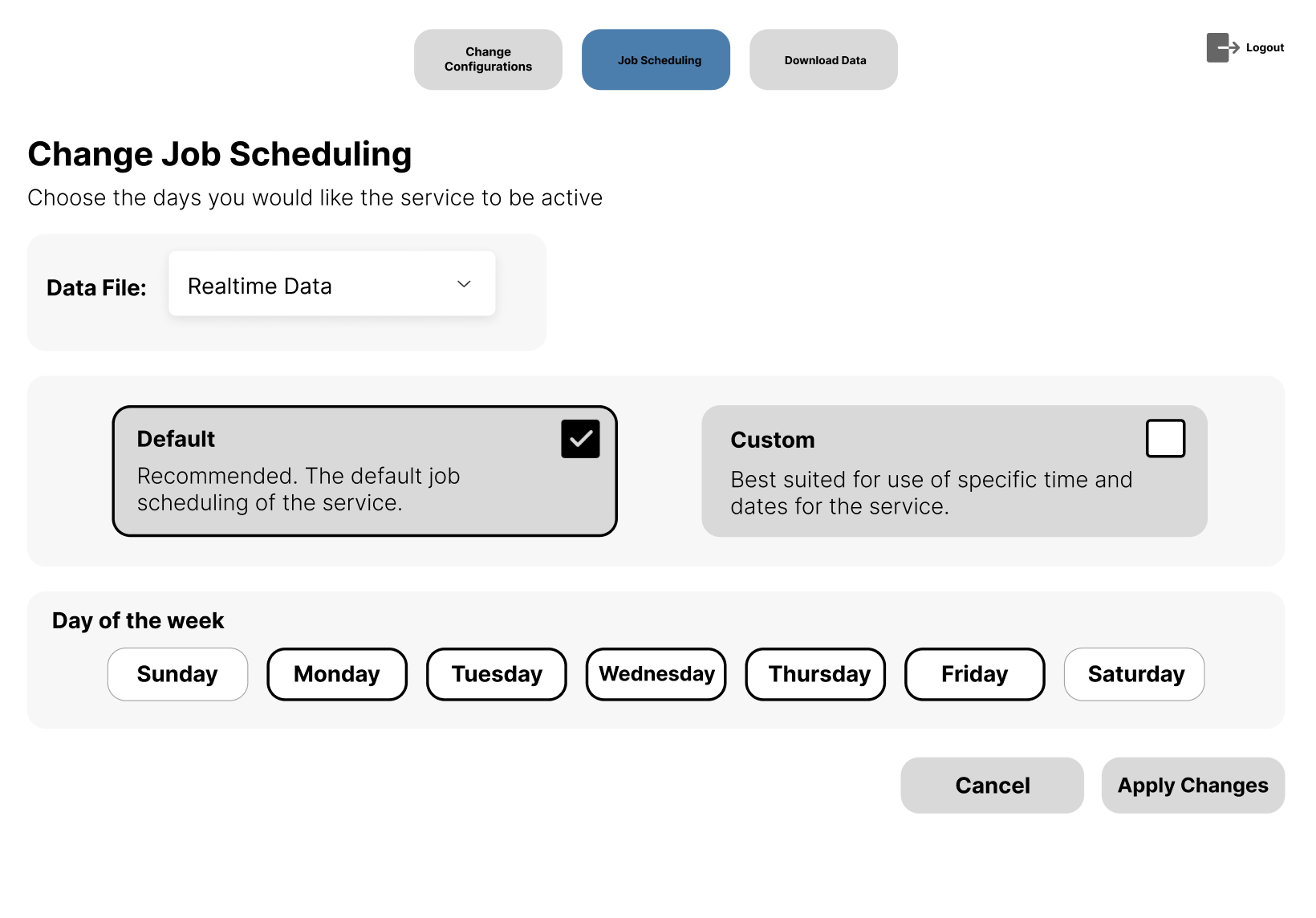
## UI Design: (Vanessa)

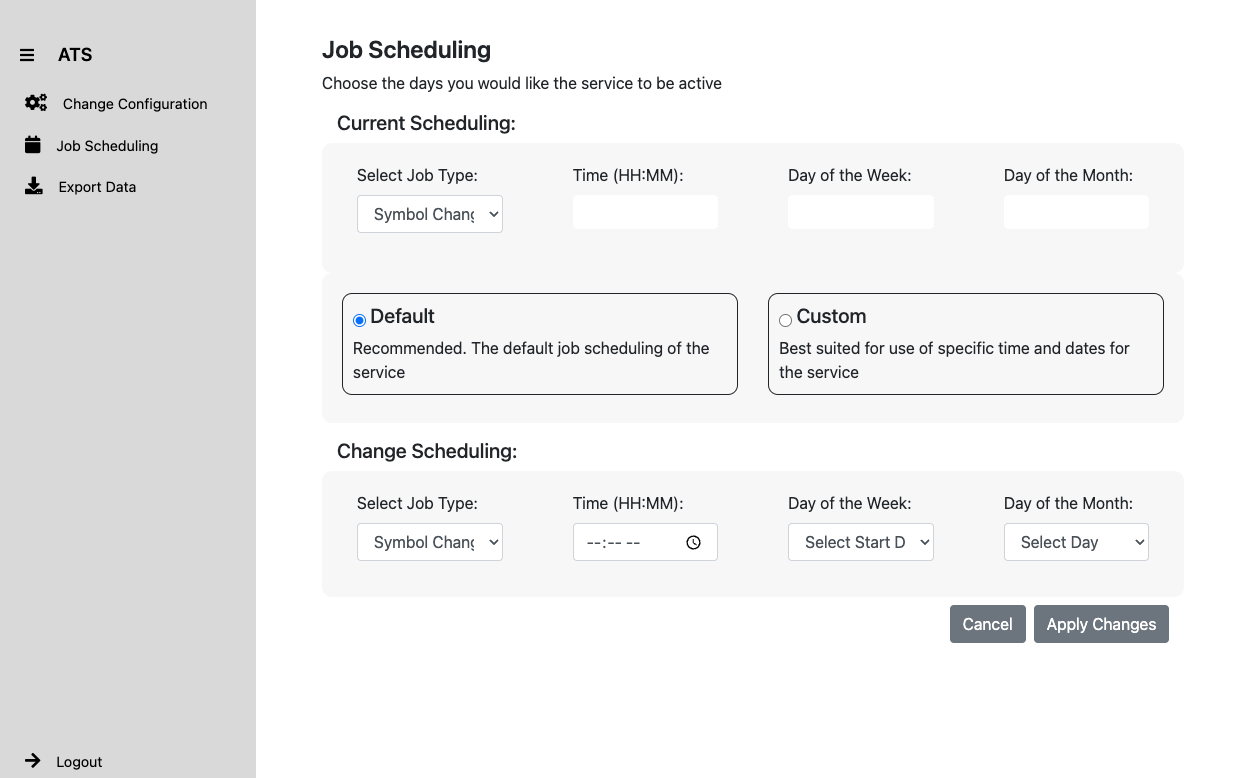
### Navigation Bar:

* Redesigned the Navigation Bar, instead of placing it on top. It is best and easier for the users to navigate using a sidebar.

### Job Scheduling Page:

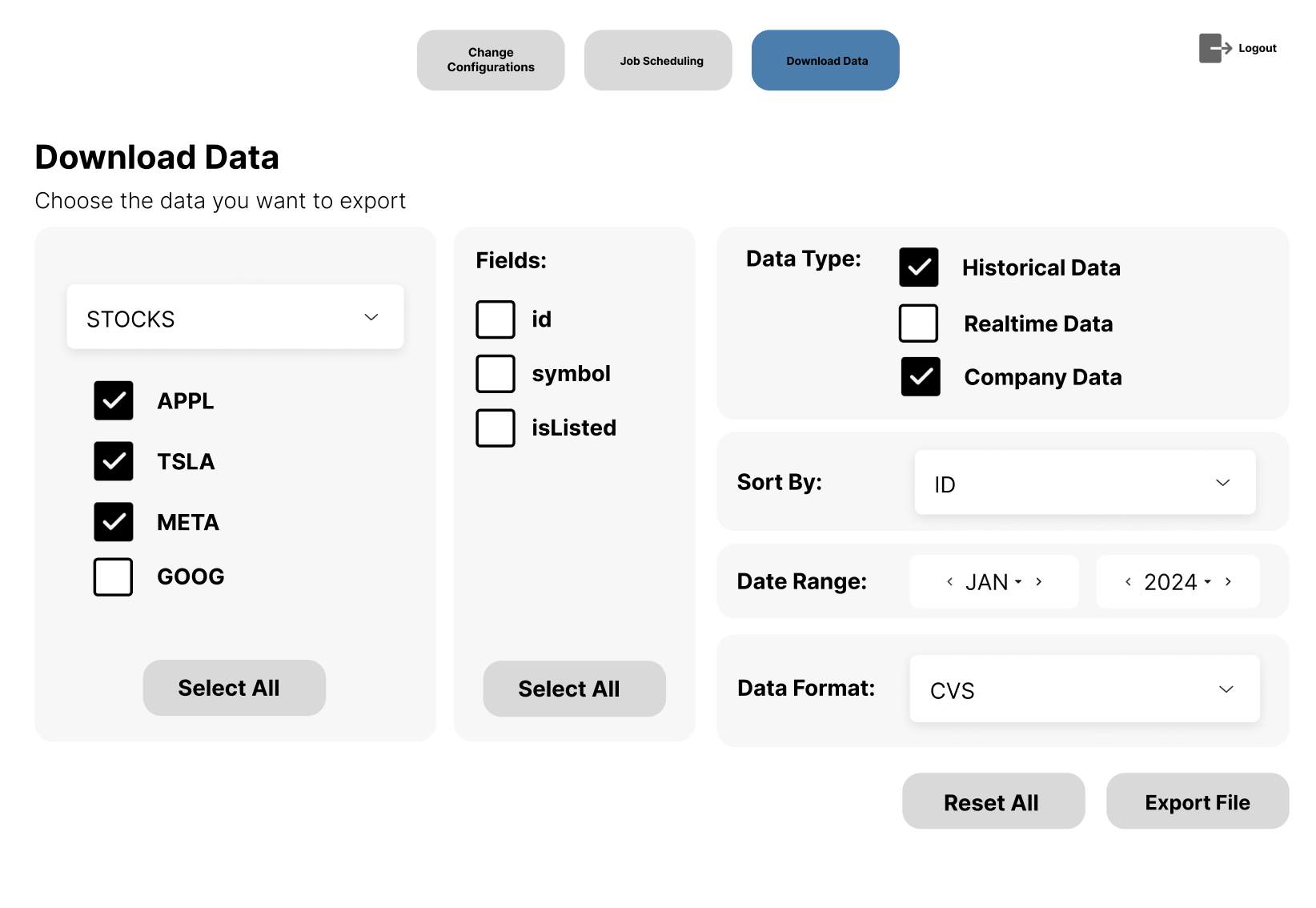
* Removed Data File. Replaced it with the Current Scheduling row. This row shows the current job scheduling for each cronfile. This is best so that users can see the current job scheduling.
* Removed the Day of the Week. Replaced it with Change Scheduling row. The dropdowns for each field is a better user experience than buttons.
* Changed the default and custom input to radioboxes instead of checkboxes. This is best so that users cannot choose both options.

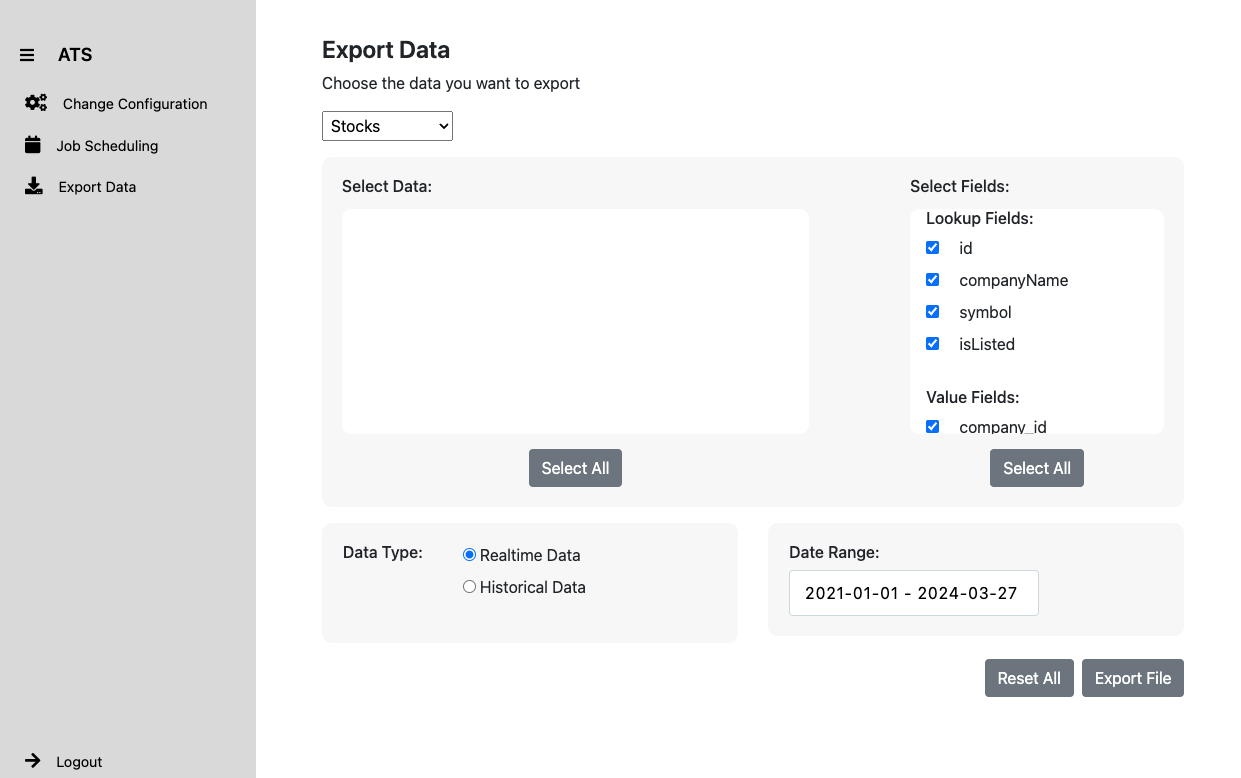




### Data Export Page:

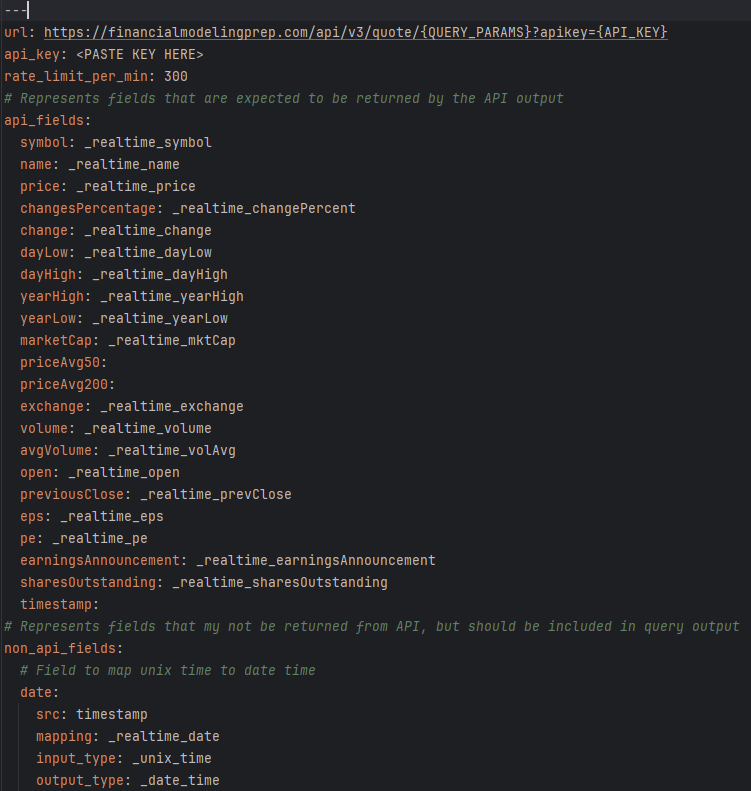
* Removed the Sort by and Data format field. Users won’t be utilizing these fields and it is not included in our requirements.
* Changed the checkbox on the Data Type field to radioboxes. This is best so that users cannot choose all the options.
* Date Range format changed. Instead of showing one date, it is now displaying the date range.
* Instead of placing the data dropdown inside a card, we placed it at the top of the card. This is a better user experience so that it is clear for the users to see what data they are exporting.





## Configuration (Jacob & Ben):

* Redesigned system configuration to use YAML format as opposed to JSON. This was to be more in line with common software practices, allow for comments within config files, and to simplify some of the collection script logic.
* Created a yaml\_handler.py utility file
* Modified collection logic to work with new format



**---->**

## 

## 

## 

Old (JSON) New (YAML)

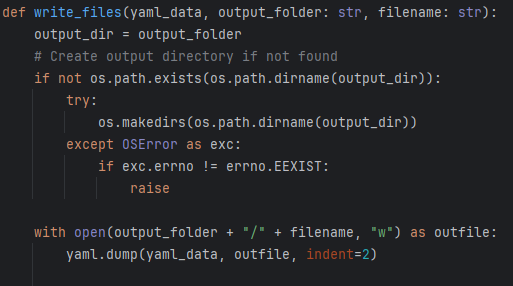
yaml\_handler.py:

## 

## 

## 

## 

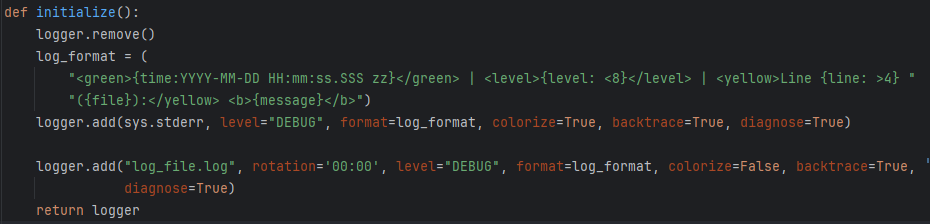


## Source Code (Jacob & Ben):

* Developed system logging using Loguru library
* Standalone initialization file
* Created logging for major system function at various levels (INFO, DEBUG, ERROR, CRITICAL, SUCCESS)
* Log database table
* Log database insertion
* Insert script load\_output function refactored into json\_handler utility

Previously to this development, the only error diagnosis in place was directly printing errors to the python console. For logging, it was only in place for cron jobs. Logging changes were made in order to follow better software engineering practices, and make system maintenance much easier in the future. Having detailed logging for all primary system functions will help developers using the system to understand how it works, and diagnose problems as they arise.

*Loguru Initializer:*

**

*Logging (examples - code):*

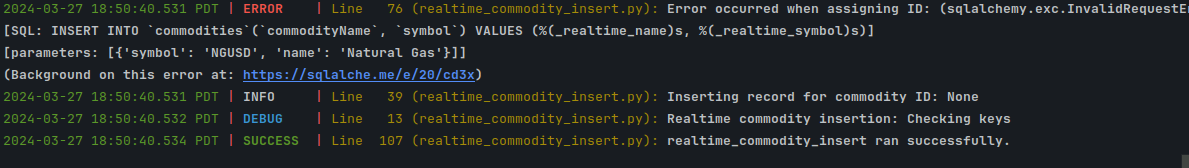
**

**

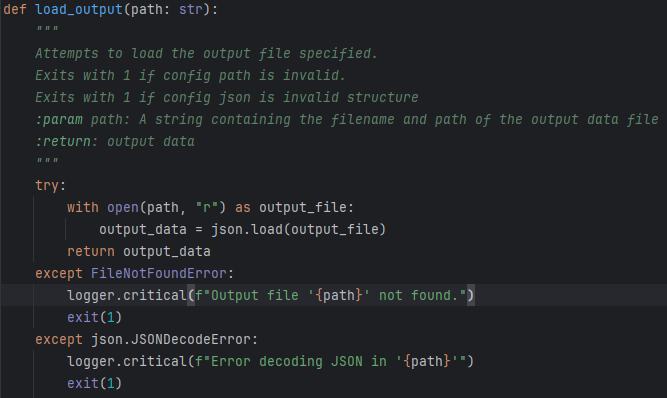
**

**

**

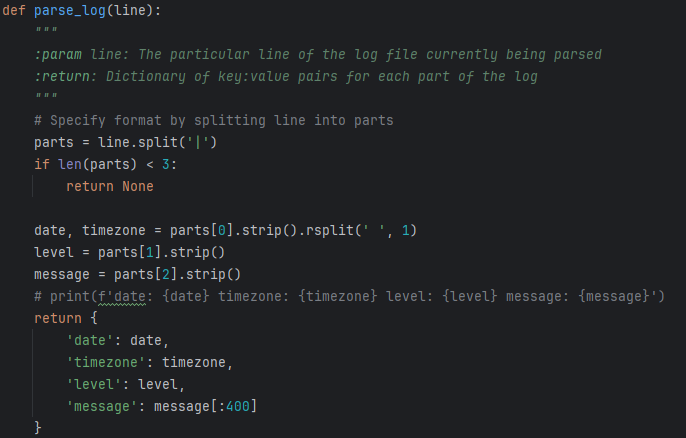
**

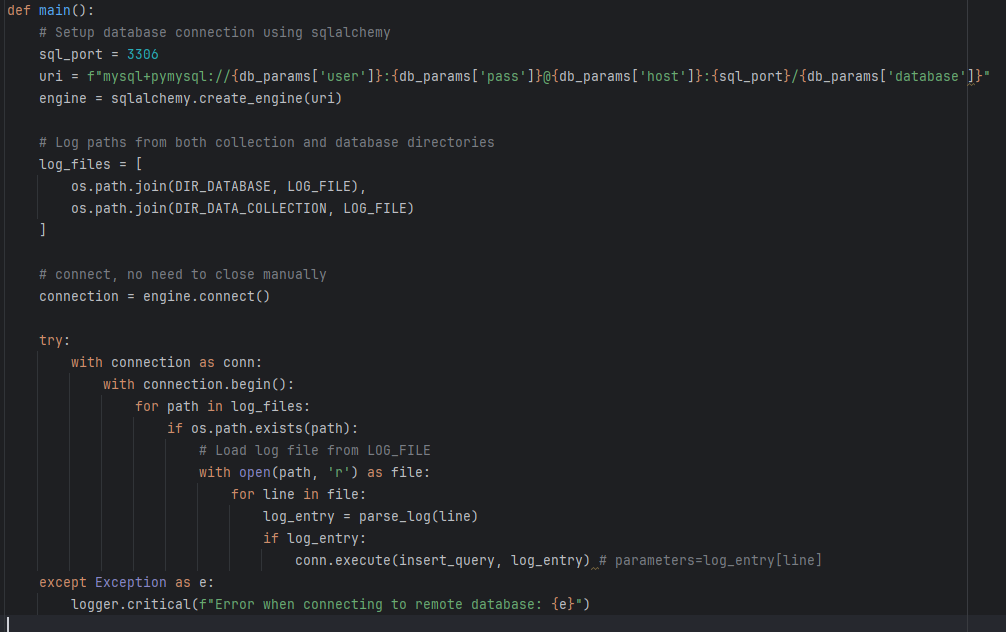
*Load\_output function:*

**

*Log insertion (Primary functionality shown only):*

**

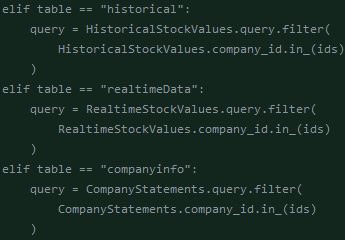
**

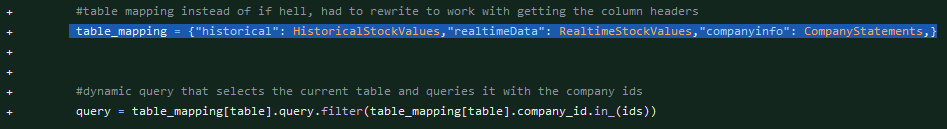
**

## Data Export Backend (Jake and Dominic):

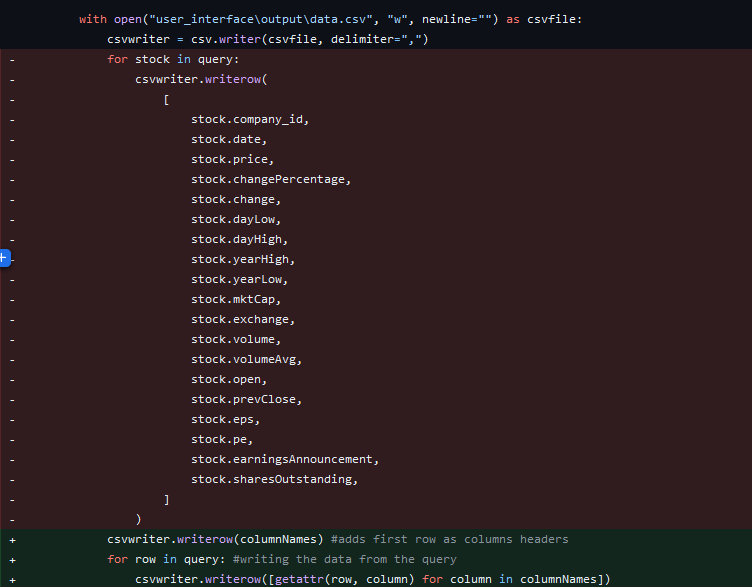
At first the system was only set up to deal with realtime data queries and had static identifiers and later had different tables added to it as requested, but we ran into issues.

We changed away from if else statements into a map to allow for dynamic expansion of new tables later. This change resulted in the use of mapping much more later down the line.





Followed by a changed data export to no longer hold static table header data and instead gets the header information dynamically. In order to accommodate for more tables.

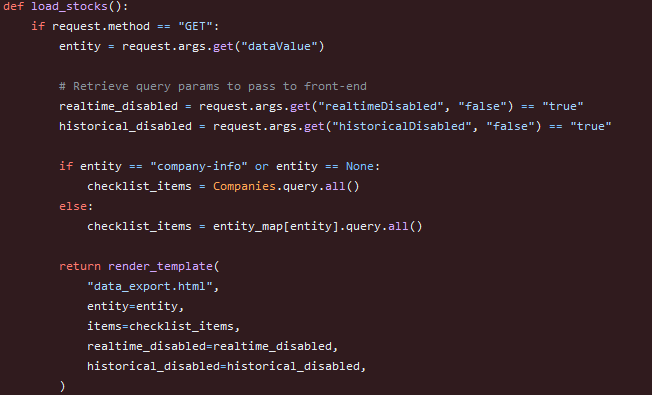


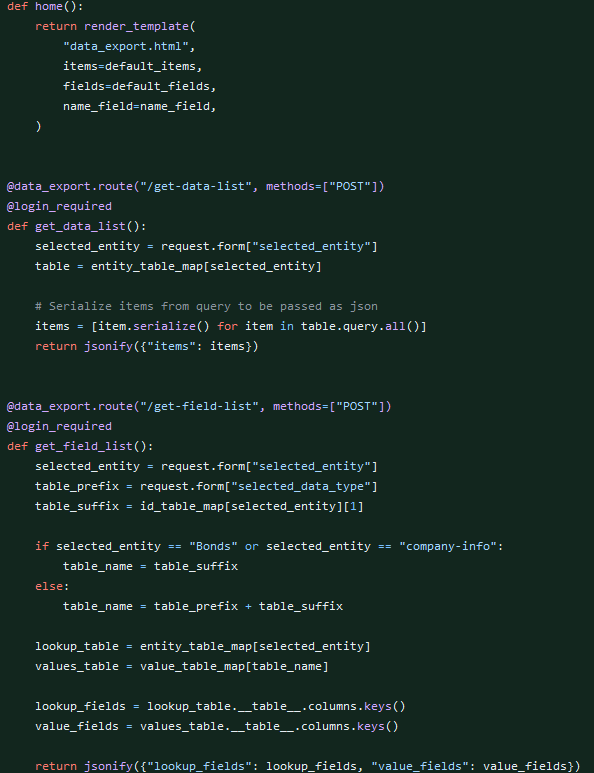
## **Flask Route Segmentation**

A big design change that was made to data\_export.py was to separate all pieces of functionality into their own routes. Initially, there was an attempt to put all functionality into one route and have the data displayed on the page by the server. This became an issue since the page content needs to be dynamic based on the users form selections, reloading the page every time something in the form changed was not scalable with more and more functionality. Below, the load\_stocks() route was used to dynamically load market entity data such as stocks into a list.

All functionality is now split up into 4 routes:

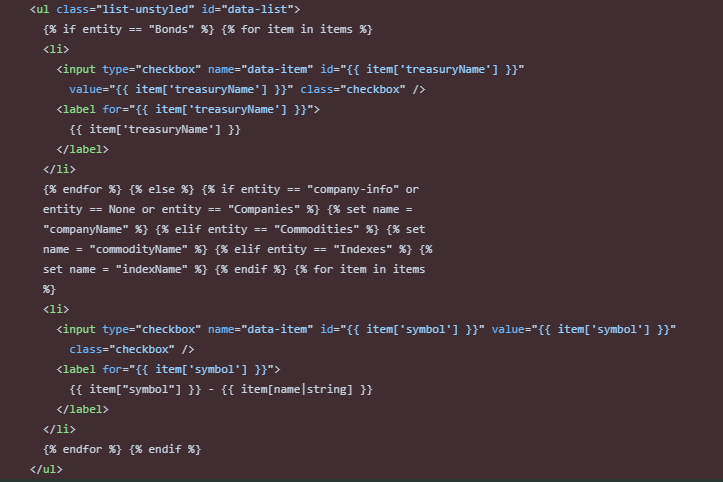
* home()
  + The route used for when the page initially loads.
  + Loads with default data (Stock data)
* get\_data\_list()
  + Queries database for all entities in database (i.e. all individual stocks)
  + Sends list of entities to the client for processing
* get\_field\_list()
  + Gets fields from all related tables and sends the list to the client
* export\_data()
  + Prepares and downloads data in csv format based on the users selections.





## **Server to Client Side Rendering**

Initially, the data list and field lists were being rendered by the server using Jinja templating. This was fine initially, but once we started implementing more things for the page it was only causing us issues.



The refactored solution is to use javascript and ajax to render the list information. The code below is from the onDataChange() change function in script.js. This function is used to handle the rendering of our entity data on the page when the data dropdown selection is changed. This function sends a POST request to the get-data-list route and receives a response back as json (this would be the list of stocks in our default case). It then displays each returned item as a list item in our list.

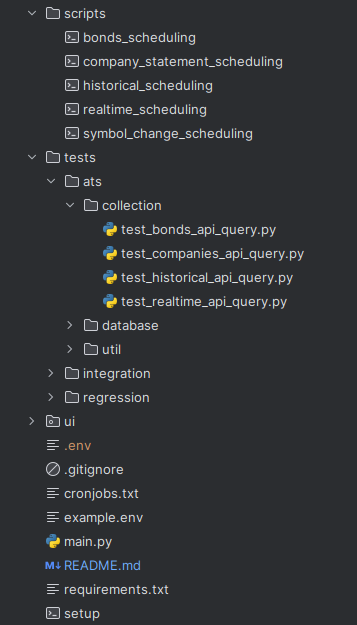
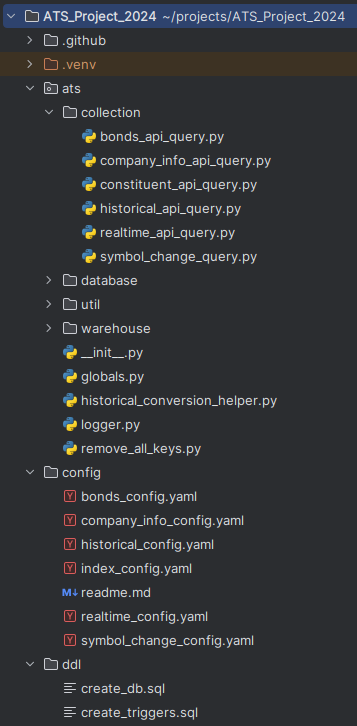
This solution has proven to be much more modular and easier to work with. It also provides a more seamless user experience by not requiring the page to reload on every change of the form.



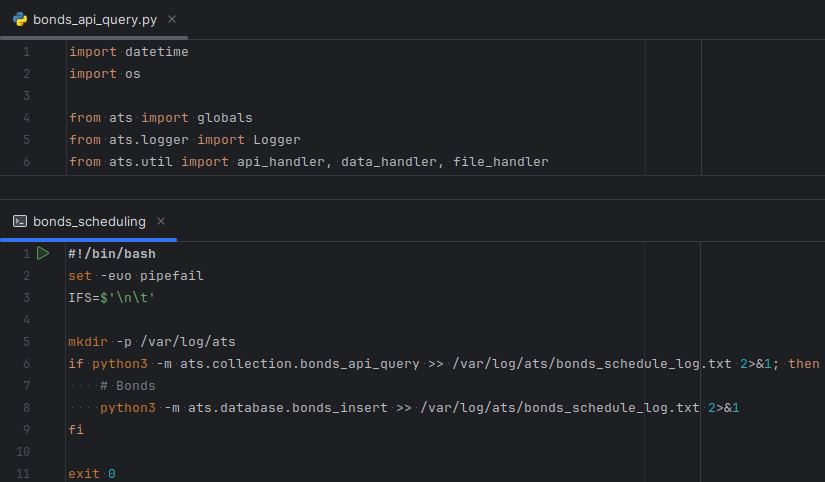
## Refactoring (Alan):

### Project Structure

The entire project directory structure was rebuilt in a more logical manner, clearly separating Python scripts and modules by functionality.

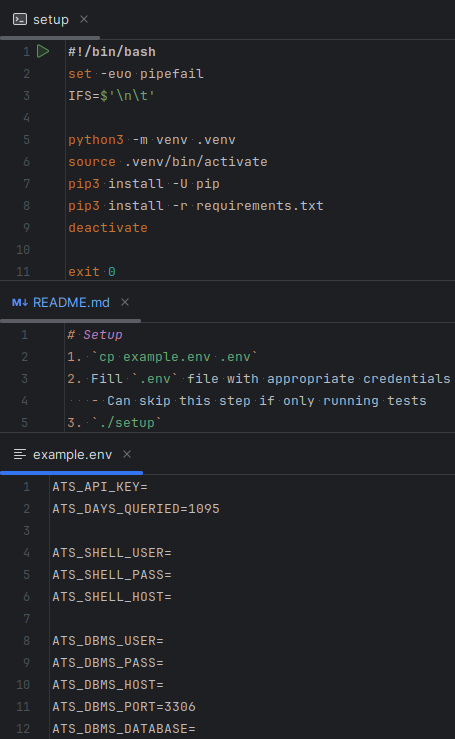


All Python scripts were modified to use absolute imports in order to conform with the new project structure. Likewise, shell scripts and other files were altered to have the correct pathing.



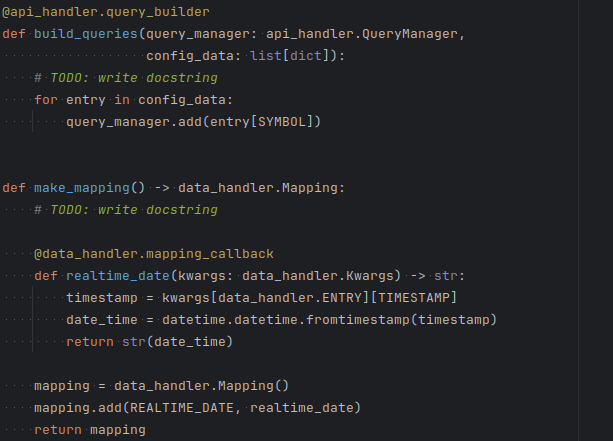
### Local Development

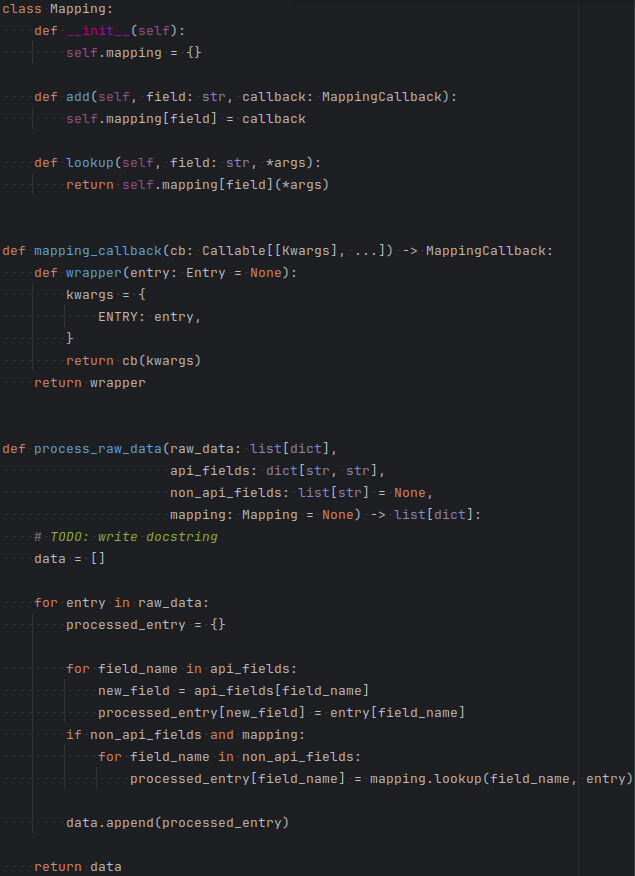
Setup was simplified by creating a shell script to automate the process, alongside a basic README to help with setup. In addition, the python-dotenv package was added in order to cordon off important variables and sensitive information from the rest of the project code, instead being stored in a file that would be loaded as environment variables.



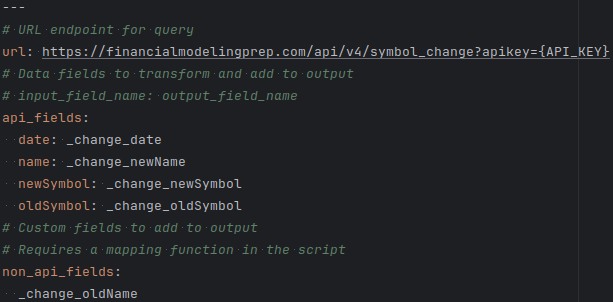
### Source Code and Configuration

All collection scripts were re-written with the goal of replacing repetitive code with abstractions (DRY) and fixing poorly written code. API querying and data processing code was largely moved from each individual collection script into two new utility scripts that dynamically handle different requests and data processing requirements. Several hidden bugs and unwanted behaviours were fixed, and a lot of unnecessary code was removed.

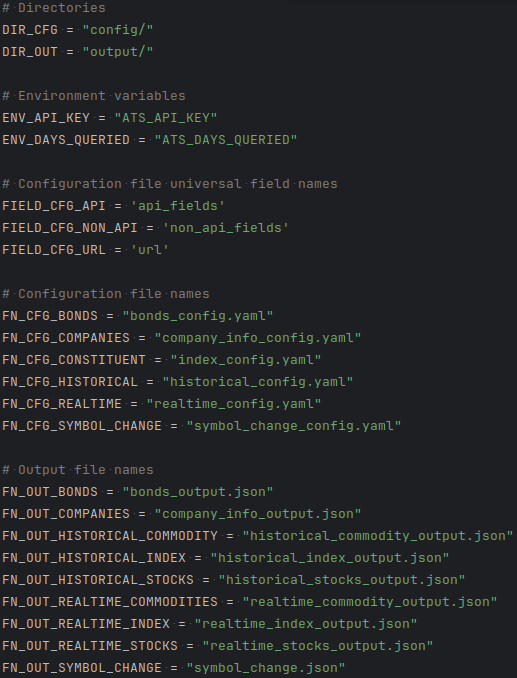




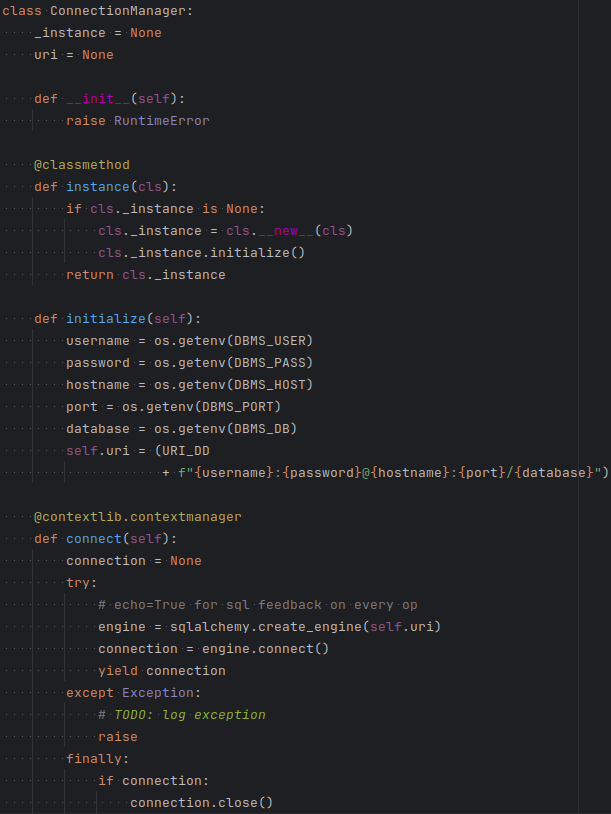
All configuration files were updated to reflect such changes.



Nearly all literals were changed into constants, with commonly used constants stored in a single file to be imported where needed.

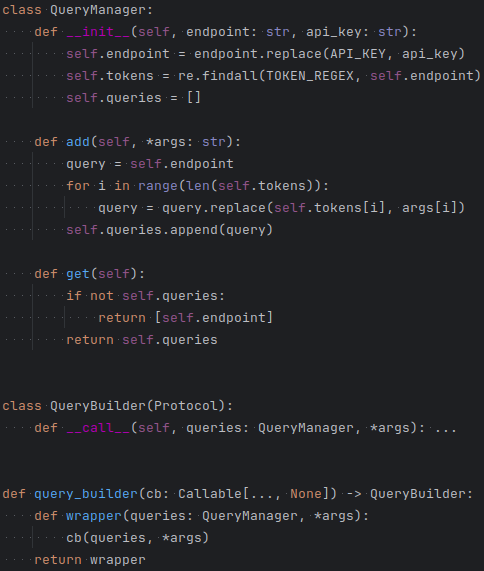


Logger and connection scripts were slightly modified to be singletons to avoid repeated initialization at runtime.



### Testing and Formalization

Scripts were also modified to make extensive use of type hints, using decorators and protocols where necessary. Informal data structures were transformed into classes, with a focus on extensibility and OO principles.



Collection and utility test scripts were rewritten and simplified, thanks to PyTest and abstracted code. The bulk of testing now focuses on API and data utility scripts.

