***Algorithmic Trading System***

***Developers Guide***

Algorithmic Trading Software

2024-04-09

Version 1.2.5

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

**Revision Sheet**

| Revision | Date | Brief Summary of Changes |
| --- | --- | --- |
| Version 1.0 | 2023-11-23 | Baseline document draft |
| Version 1.1 | 2023-12-07 | General editing and revisions |
| Version 1.2 | 2024-01-16 | General review and cleanup |
| Version 1.2.1 | 2024-02-26 | * Updated any references to JSON format for configuration files to YAML, following the transition. * Added section for testing   + Test still need to be documented here * Added section for IONOS web hosting   + Will need to add details here once this is setup |
| Version 1.2.2 | 2024-03-14 | Updated to reflect new project structure, fixed some typos, removed some links and references that are no longer relevant |
| Version 1.2.3 | 2024-03-27 | * Fixed some headings * Added content for User interface under section 7 |
| Version 1.2.4 | 2024-04-07 | * Changed the Cron setup in the server setup section |
| Version 1.2.5 | 2024-04-09 | * Code/field formatting * Changes to reflect refactoring   + Utility scripts   + Abstracted data structures * Updated section 7.2 * Added section 7.3 - UI Flask App Deployment |

[**1. Introduction 3**](#_heading=h.1fob9te)

[1.1 Overview 3](#_heading=h.3znysh7)

[1.2 Key References 3](#_heading=h.2et92p0)

[1.3 Required Skills 3](#_heading=h.tyjcwt)

[**2. IONOS 3**](#_heading=h.3dy6vkm)

[2.1 Managing ATS Databases 3](#_heading=h.1t3h5sf)

[2.1.1 Creating the Database 4](#_heading=h.4d34og8)

[2.1.2 Accessing the database 4](#_heading=h.2s8eyo1)

[**3. Setup 7**](#_heading=h.t8uqsmked0f9)

[**4. Data Collection 7**](#_heading=h.17dp8vu)

[4.1 Configuration Files 8](#_heading=h.26in1rg)

[4.1.1 Shared Structure 8](#_heading=h.lnxbz9)

[4.1.2 Specific fields 8](#_heading=h.35nkun2)

[4.2 Collection Scripts 8](#_heading=h.1ksv4uv)

[4.3 Output Files 9](#_heading=h.44sinio)

[**5. Database 10**](#_heading=h.2jxsxqh)

[5.1 Schema 10](#_heading=h.z337ya)

[5.1.1 Stock/Company Tables 10](#_heading=h.3j2qqm3)

[5.1.2 Commodity Tables 11](#_heading=h.1y810tw)

[5.1.3 Bond Tables 11](#_heading=h.4i7ojhp)

[5.1.4 Index Tables 11](#_heading=h.2xcytpi)

[5.2 Database Creation 11](#_heading=h.1ci93xb)

[5.2.1 DDL 11](#_heading=h.3whwml4)

[5.2.2 Triggers 12](#_heading=h.2bn6wsx)

[ID Generation Triggers 12](#_heading=h.3as4poj)

[Symbol Change Trigger 12](#_heading=h.49x2ik5)

[**5.3 Database Operations 12**](#_heading=h.2p2csry)

[5.3.1 Insertion Scripts 13](#_heading=h.147n2zr)

[5.3.2 Symbol Change Script 14](#_heading=h.23ckvvd)

[5.3.3 Dependencies 14](#_heading=h.ihv636)

[**6. Automation 14**](#_heading=h.32hioqz)

[6.1 Scripts 14](#_heading=h.t6ii5l4zbzpc)

[6.2 Cron 15](#_heading=h.uwtwg6crcmuj)

[6.2.1 Cron specifics 15](#_heading=h.5daa1re6gh1r)

[**7. ATS User Interface 16**](#_heading=h.itg2rnc4wkfn)

[7.1 Technologies 16](#_heading=h.htp4ha9blu7)

[7.1.1 Front-end 16](#_heading=h.muit1mtcxsvo)

[7.1.2 Back-end 17](#_heading=h.ba9rp4f25bn1)

[7.2 Directory and File Structure 17](#_heading=h.oe5sfig62604)

[7.2.1 Directories 18](#_heading=h.tsxklg3sw8o0)

[7.2.2 Templates 18](#_heading=h.c49znbec0wn3)

[7.2.3 Blueprints 19](#_heading=h.wh14iuuiuo3y)

[7.2.4 Other Files 20](#_heading=h.dlei01osqe6z)

[7.3 UI Flask App Deployment 21](#_heading=h.iir86ycdcawg)

[7.3.1 Setup 21](#_heading=h.x2oete3ltwbl)

[7.4 IONOS Web Hosting 23](#_heading=h.6zuvjrgtlrdw)

[**8. Running the System Locally 23**](#_heading=h.vx1227)

[8.1 Steps 23](#_heading=h.pcayniaodpkg)

[**9. Server Setup 23**](#_heading=h.1v1yuxt)

[1. Pre-Requisites 23](#_heading=h.4f1mdlm)

[2. Using SSH 24](#_heading=h.2u6wntf)

[3. Git Pulling 24](#_heading=h.19c6y18)

[4. Using SFTP 24](#_heading=h.3tbugp1)

[5. Setting up CRON jobs 25](#_heading=h.28h4qwu)

[**10. Use Case Diagram 25**](#_heading=h.nmf14n)

[**11. Architecture 25**](#_heading=h.37m2jsg)

[**12. Testing 26**](#_heading=h.1mrcu09)

# 

# 

# Introduction

## Overview

This document is intended to be used to recreate the Algorithmic Trading System (ATS), encompassing the collection and storage of company information, stocks, indexes, commodities, and bonds. The ATS is a stock market tracking product that populates a database with real-time and historical financial information from the Financial Modelling Prep API. The ATS automatically pulls data from the API and inserts it into the database, at regularly scheduled times, using the various technologies outlined in this document.

## Key References

* [MySQL Documentation](https://dev.mysql.com/doc/refman/8.0/en/miscellaneous-functions.html) [R1]
* [User Manual](https://docs.google.com/document/d/1MSl4BucQyQG3XGQqiSBJuw8ncuh2JzPF/edit#heading=h.5mdv64t4m99i) [R2]
* [Github](https://github.com/COSC-470-2023/SMF_Project_2023/tree/develop/data_collection/configuration)
* [IONOS](https://docs.ionos.com/cloud/)
* [ATS Software Design Document](https://docs.google.com/document/d/1kH4S7RFlHHq6SzOlhkYMcuc-0IA7yYNazzqXoAM9lVY/edit?usp=sharing) [R3]

## Required Skills

Knowledge of the following software, services and data formats are expected of developers working on the Algorithmic Trading System.

* Python and pip
* MySQL
* YAML
* JSON
* Bash and environment variables
* REST API usage
* cron
* IONOS hosting
* phpMyAdmin
* git

# IONOS

## Managing ATS Databases

IONOS is a cloud provider that hosts the ATS software and databases, which can be managed by using its web interface.

In order to manage databases, first log in to the IONOS account as explained in section 4.1 of the [User Manual](https://docs.google.com/document/d/1MSl4BucQyQG3XGQqiSBJuw8ncuh2JzPF/edit#heading=h.5mdv64t4m99i). Afterwards, navigate to the Database section as shown in section 4.2 of the [User Manual](https://docs.google.com/document/d/1MSl4BucQyQG3XGQqiSBJuw8ncuh2JzPF/heading=h.vuyuw39va7t2).

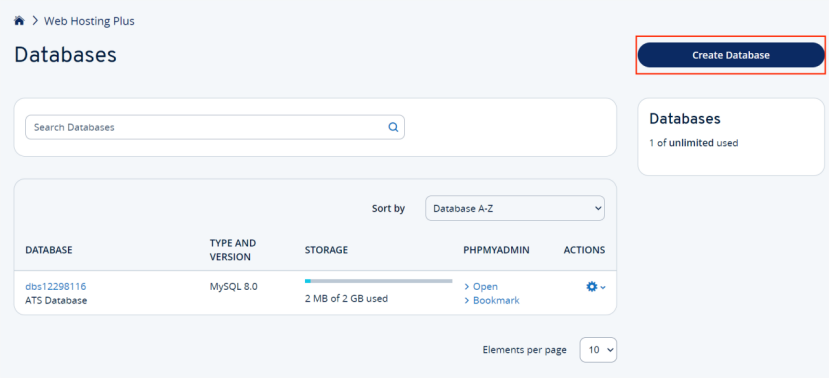
### Creating the Database

In the top right corner of IONOS click the “Create Database” button. *(figure 2.1)* Required information must be entered during the construction phase:

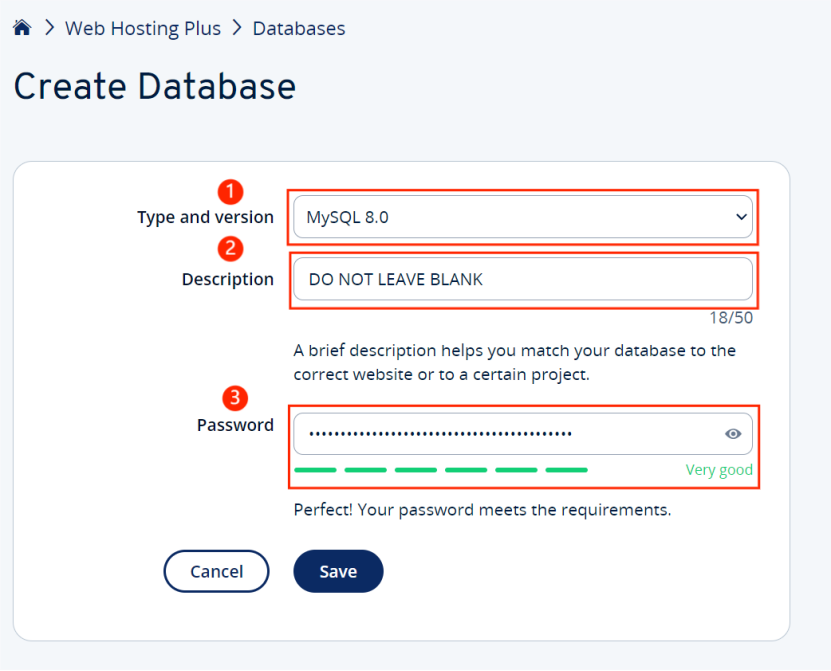
* MySQL version: 8.0
* Provide a password and a description of the database. (*figure 2.2*)
  + The description should never be left blank as IONOS does not provide a way to name the database, the description is the ONLY identifier for databases.
  + Please be advised, that this setup process may take a while.
* Use the [DDL](https://github.com/COSC-470-2023/SMF_Project_2023/blob/develop/database/ddl/create_db.sql) script to create the database tables.
  + Using IONOS phpMyAdmin to create tables. (*figure 2.4*)
    - Choose the database, select the “SQL” tab, and then copy the DDL script into the text box.
    - Press “Go” to execute the SQL statement and create the table

### Accessing the database

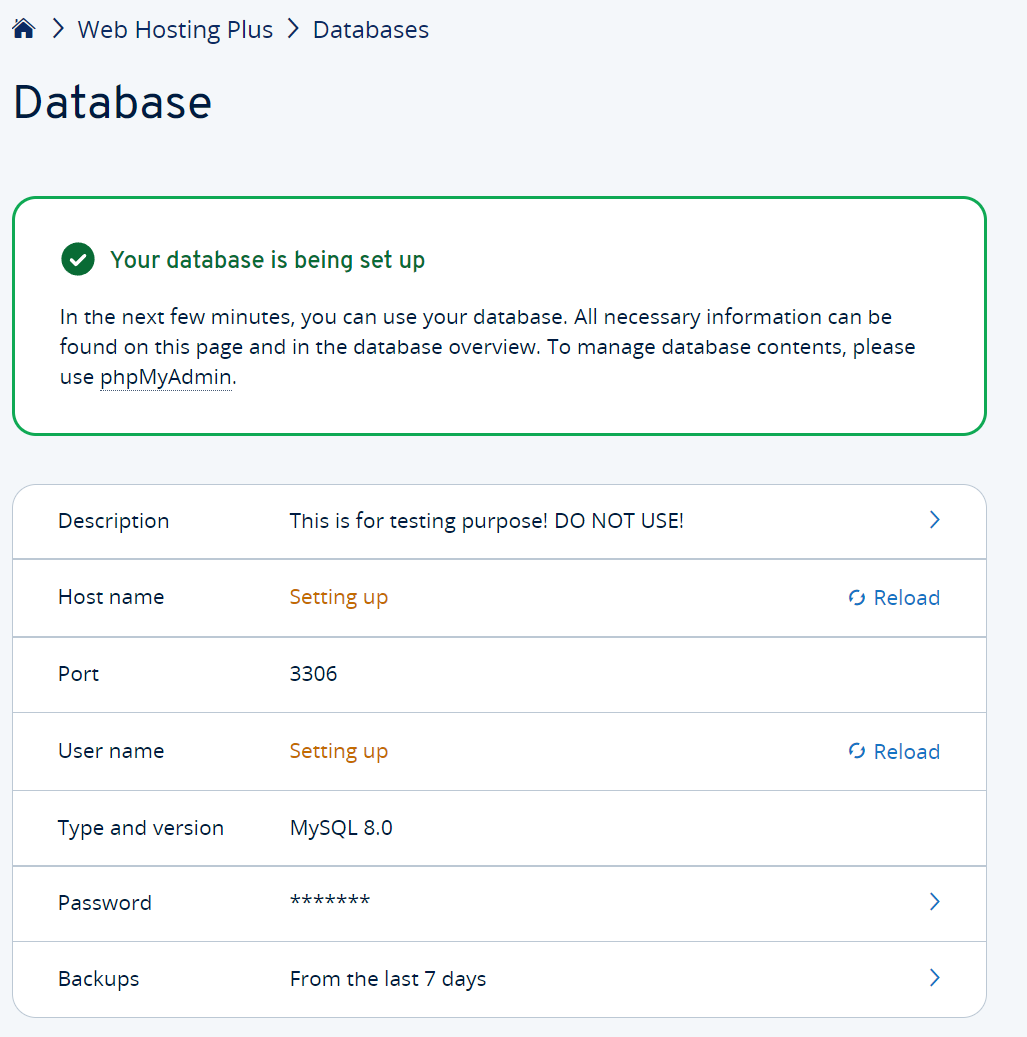
Accessing the Database section is shown in section 4.2 of the [User Manual](https://docs.google.com/document/d/1MSl4BucQyQG3XGQqiSBJuw8ncuh2JzPF/heading=h.vuyuw39va7t2).



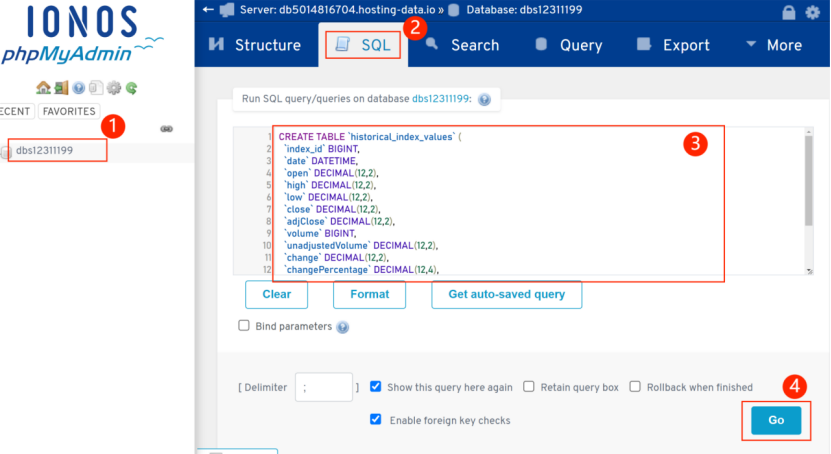
*(figure 2.1: IONOS “manage databases” page)*

**

*(figure 2.2: IONOS “Create Database” page)*

**

*(figure 2.3: database creation success screen)*

**

*(figure 2.4: phpMyAdmin SQL screen)*

# Setup

Follow the instructions in README.md in order to set up the correct environment variables and dependencies. You need to have the correct database credentials and API access in order for the scripts to work.

# Data Collection

The collection process is as follows:

1. Run *constituent\_api\_query* to retrieve the current list of symbols in the S&P500
2. Run *symbol\_change\_query* to update symbols stored in configuration files. This must be run before the collection scripts or the data may have inconsistencies.
3. Run the remaining Python data collection scripts. These can be run independently at separate times or in parallel.
   1. Python scripts read configuration files in order to make the API calls.
   2. Python scripts make requests to the API using the provided environment variables.
   3. Python scripts receive data from the API in JSON format and write data to disk in the output/ folder.

Scripts can be manually run in the root of the project directory with the command

python3 -m path.to.module

## 

## Configuration Files

Configuration files are located in the config/ directory. Each file determines what data is being pulled from the API and must be present in order for data collection to work. Each file is used as input for one or more collection scripts.

### Shared Structure

Most of the configuration files have similar variables that contain important information that are shared between them all:

* **url**: Specifies the API endpoint requests are made to along with any relevant query strings
* **api\_fields**: Contains variables that are expected to be returned by the API call, relevant to the Python scripts doing the data collection.
* **non\_api\_fields**: Used to contain fields that are not returned by the API, but are desired in the final output. For example, Timestamp (rather than Unix Time).

### Specific fields

* company\_info\_config.yaml: Specifies what company information should be collected and for which companies.
  + **companies**: Lists companies to collect data of, by name and stock ticker.
* historical\_config.yaml: Specifies which stocks, indexes, and commodities to collect historical data of, as defined by start and end-date ranges.
  + **index\_composites**: Lists stock indexes to collect data of.
  + **stocks**: Lists specific stocks to collect data of.
  + **commodities**: Lists commodities to collect data of.
* realtime\_config.yaml: Specifies which stocks, indexes, and commodities to collect real-time data of.
  + **stocks**: Lists specific stocks to collect data of, identified by name and stock ticker symbol.
  + **index\_composites**: Likewise, lists stock indexes to collect data of.
  + **commodities**: Likewise, lists commodities to collect data of.

## Collection Scripts

Data collection scripts are located in the ats/collection/ directory.

These scripts query the API with information taken from the configuration files, and write the collected information out to JSON files in the output/ directory.

Python Script Files (including some utility scripts located in ats/util/ and others):

* *file\_handler.py*: logic for reading and writing YAML files JSON files
* *api\_handler*.py: logic used to retrieve data from the API
* *data\_handler*.py: logic used to manipulate and process data from the API
* globals.py: stores important constants used across most scripts
* bonds\_api\_query.py: Queries the API for bond data. Creates date windows and calculates days remaining in 90 day spans for bond information.
* company\_info\_api\_query.py: Queries the API for company information.
* historical\_api\_query.py: Queries the API for historical stock, index, and commodity information.
* realtime\_api\_query.py: Queries the API for current stock information.
* symbol\_change\_query.py: Queries the API for stock or index symbol changes.

Most collection scripts make use of one or more functions that build queries and mappings. Both functions make use of associated decorators in the api\_handler and data\_handler modules. The query builder (query\_builder) is a callback responsible for handling the population of queries in the query manager object for when the API is queried. The mapping builder (make\_mapping) is used to *make* a callback function that is used to create non-API fields when data is being processed. The parameters that are passed into these functions are dependent on the script itself, and the data being collected. For example, a query builder might need to have the stocks listed in a config file. Such a list must be passed into the query manager itself upon fetching, which will in turn be passed into the query builder as a callback. The mapping builder is similar, but has access to certain variables used in data processing, such as the entry currently being processed.

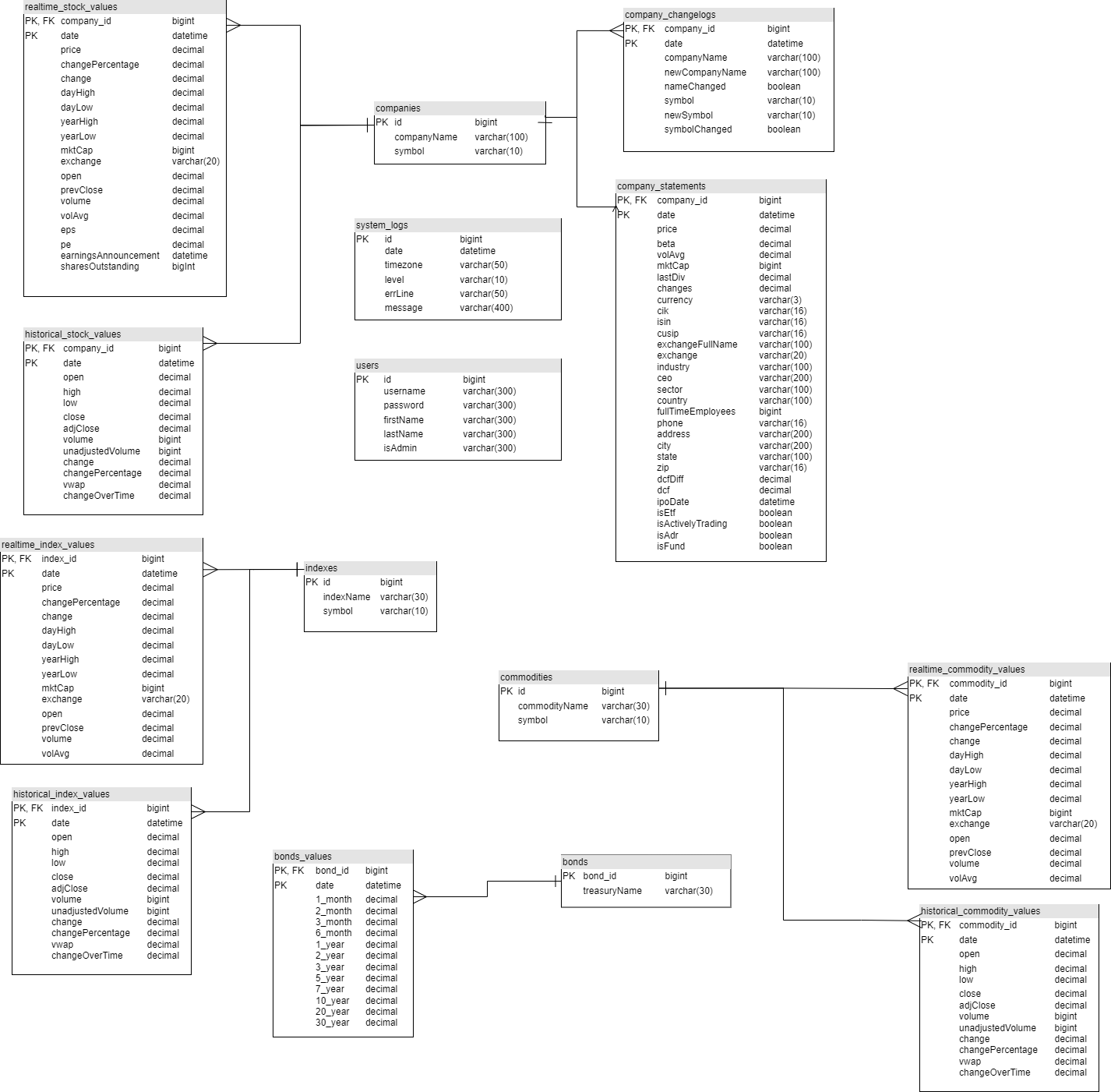
## Output Files

Most scripts create one or more JSON files in the output/ directory. These files are later used by the database scripts to insert data into the database.

* bonds\_output.json: contains bond data.
* company\_info\_output.json: contains real time company information.
* historical\_stocks\_output.json: contains historical stock data with dates.
* historical\_index\_output.json: contains historical index data with dates.
* historical\_commodity\_output.json: contains historical commodity data with dates.
* realtime\_stocks\_output.json: contains recent stock data.
* realtime\_index\_output.json: contains recent index data.
* realtime\_commodity\_output.json: contains recent commodity data.
* symbol\_change\_list.json: contains data on any company symbol changes if applicable.

# Database

## Schema



*(figure 3.1) Entity Relationship Diagram*

### Stock/Company Tables

* realtime\_stock\_values
  + Used to store real time data related to stock values.
* historical\_stock\_values
  + Used to store historical data related to stock values.
* companies
  + Keeps a unique record of each company encountered by the system.
* company\_changelogs
  + Records changes to any companies from the *companies* table (i.e. if the company has a new name/symbol).
* company\_statements
  + Used to store information related to the companies in the *companies* table (i.e.phone,address,employees).

### Commodity Tables

* commodities
  + Keeps a unique record of each commodity encountered by the system.
* realtime\_commodity\_values
  + Used to store real time data related to commodity values.
* historical\_commodity\_values
  + Used to store historical data related to commodity values.

### Bond Tables

* bonds
  + Keeps a unique record of each bond encountered by the system
* bonds\_values
  + Used to store information related to bonds

### Index Tables

* indexes
  + Keeps a unique record of each index encountered by the system.
* realtime\_index\_values
  + Used to store real time data related to index values.
* historical\_index\_values
  + Used to store historical data related to index values

## Database Creation

### DDL

There is a provided SQL file that will create the database based off of the diagram (See Figure 4.1) above located in the ddl/ directory. The current hosting configuration doesn’t provide the permissions required to create a database while connected through SSH. The database will have to be created through the IONOS hosting dashboard and phpMyAdmin. Please refer to [*Section 2.1.1*](#_heading=h.4d34og8)for more details.

### Triggers

Currently, there are five triggers. Four of these triggers are designed to handle inserts for our companies, indexes, commodities, and bonds tables, while the last trigger handles changes to stock symbols. All of the trigger creation statements are located in the ddl/ directory, and will need to be run after the database is created to ensure proper execution of data insertions into the database.

### 

### ID Generation Triggers

The companies*,* indexes*,* commodities, and bonds tables (See Figure 3.1) are used to hold a unique record of each market entity tracked by the system. A trigger is attached to each of these tables to generate an ID for newly encountered entities. Any insertion on these tables will cause a trigger to fire. IDs are generated using the MySQL UUID\_SHORT() function.



### 

### Symbol Change Trigger

To handle the tracking of changes to company names and symbols, the company\_changelogstableis needed. To populate this table, a trigger is attached to the companiestable, which fires on the update of a company name or symbol. Each update will create a new row in company\_changelogs*.*



## Database Operations

Scripts performing database operations are located under ats/database. For the tables that require both real time and historical values, an insertion script is provided for each.

All of the scripts are dependent on both the db\_handler moduleand the environment variables file for connecting to the database with proper credentials.

### Insertion Scripts

Each of the insertion scripts has 3 specific functions, load\_output\_file, execute\_insert*,* and a script specific get\_id*.* The example below is taken from realtime\_index\_insert.py*:*



* Accepts 1 parameter:
  + path
    - The path to the .yaml output from the API
* Accepts a file path as an input, and parses a json file containing data retrieved from the API.



* Accepts 3 parameters:
  + connection
    - The connection to the database
  + entry
    - A YAML object pulled from the API output.
    - Gets inserted into the corresponding values table (See figure 3.1)
  + index\_id
    - Generated from the script-specific get\_id, and passed through to this function.
* Iterates over API output and runs an insert query for each record.



* Accepts 2 parameters:
  + entry
    - A YAML object pulled from the API output.
    - Gets inserted into the corresponding values table (See figure 3.1)
  + connection
    - The connection to the database.
* Checks for a matching record in the database, and retrieves the ID.
* If no record is found, execute a blank insert to fire the corresponding trigger.
  + New record is created.
  + New ID is pulled from the database
* The resulting ID is returned by the function

### 

### Symbol Change Script

Like the insertion scripts above, symbol\_change\_update.py includes the load\_output\_filefunction, but only includes one other function, update\_symbol*.*



* Accepts 2 parameters:
  + connection
    - The connection to the database.
  + symbol
    - A YAML object pulled from the API output.
    - Replaces the old symbol, and gets inserted into the *companies* table.
* Updates symbol in the companies table to the new symbol given in the function, and updates old\_symbolto be the old symbol.

### Dependencies

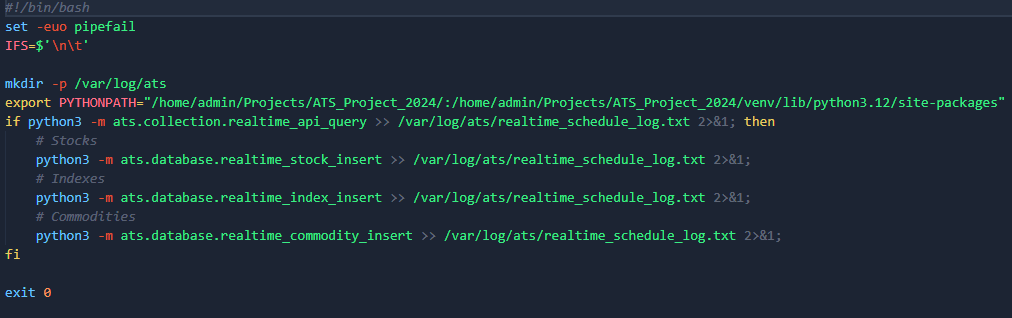
* db\_handler.py
  + Used to create a connection to the database
* .env
  + Holds database and SSH credentials to be used by db\_handler.py
* *SQLAlchemy Library*
  + A Python library used for connecting to the database through the scripts/inserting values.

# Automation

In order for the data to be automatically collected and inserted into the database, there are shell scripts located in the scripts/ directory. These scripts are set up to be run at specific times of the day/week/month using cron to schedule these tasks. Each shell script executes one or more data collection/insertion Python scripts.

## Scripts

The data collection Python scripts are always run before the data insertion Python scripts as shown (see Figure 6.1.1). If a collection job is not scheduled, then the insertion jobs do not need to be executed.



*(Figure 6.1.1) Real time data shell script*

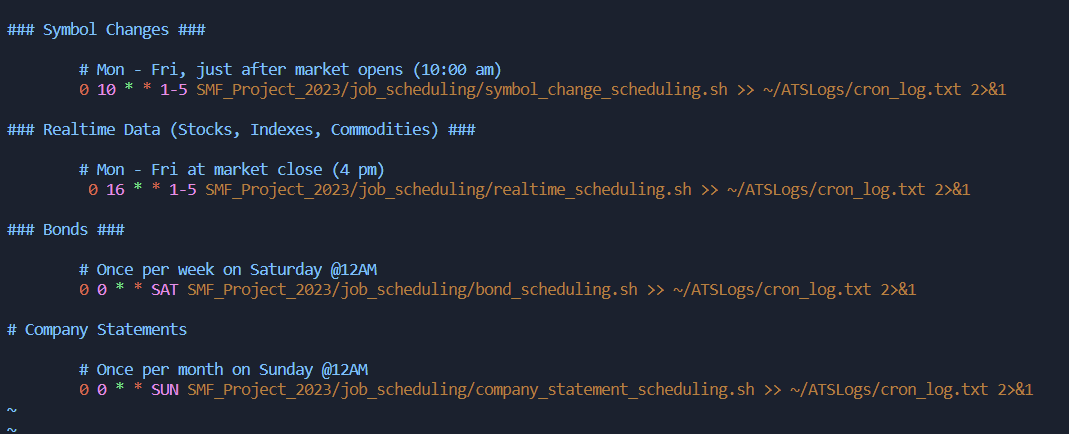
## Cron

cron manages the time and date of when these shell scripts regularly execute. To set the cron jobs up, refer to figure 6.2.1. Each job is scheduled at different times and executes shell scripts to collect and insert data into the system. Jobs can be changed via the Job Scheduling page on the ATS UI, however, default cron configurations are recommended to ensure proper system function.

### Cron specifics

The following are explanations of how the current setup runs the shells scripts and any considerations:

* Symbol changes
  + Default job runs every day Monday - Friday, just after the market opens (10:00 am EST).
  + Must run before any other job at the start of the day to propagate new symbol changes through the system.
  + Pulls daily stock market symbol changes before information on these stocks are then inserted/updated into the database.
  + Can be changed via the Job Scheduling page in the ATS UI, however, the default job is recommended.
* Constituent Update
  + Default job runs every day Monday - Friday, soon after the symbol change job (10:30 am EST).
  + Should be run before and data collection jobs are run.
  + Ensures that the system is up to date with the stocks in the configured constitute (e.g. S&P 500)
* Historical Data (Stocks, Index, Commodities)
  + Default job runs every day Monday - Friday,
* Realtime Data (Stocks, Indexes, Commodities)
  + Runs every day Monday - Friday, at market close (4pm EST).
  + Pulls in the stock, index, and commodities data and then inserts them into the database.
* Bonds Data
  + Runs once per week on Saturday at midnight (12am EST).
  + Pulls in bond data and then inserts them into the database.
* Company Data
  + Runs once per month at midnight on Sunday (12am EST).
  + Pulls in company data and then insert/modifies the database.



*(Figure 6.2.1) Cron scheduling*

# ATS User Interface

This section is a brief overview of the setup, directory structure, and technologies used for the ATS UI application, as well as a possible hosting solution using IONOS web hosting.

## Technologies

### Front-end

* Bootstrap
  + Framework we used to handle most of the webpage structure and styling.
  + Visit the [Bootstrap Documentation](https://getbootstrap.com/docs/4.1/getting-started/introduction/) for setup and usage information.
* Javascript
  + Used to handle most of the client side functionality of the app.
* Jinja2 Templating
  + Used for HTML structure and modularity.
  + Vist the [Jinja Documentation](https://jinja.palletsprojects.com/en/3.1.x/) for setup and usage information.

### Back-end

* Flask
  + Used to handle all server side functionality of the application.
  + Visit the [Flask Documentation](https://flask.palletsprojects.com/en/3.0.x/) for setup and usage information.



## Directory and File Structure

Below is the overview of the UI directory and file structure, along with some descriptions of the directories themselves and what they should be used for.

.

├── ui

│ ├── output

│ ├── static

│ │ ├── script.js

│ │ └── style.css

│ ├── templates

│ │ ├── base.html

│ │ ├── configuration.html

│ │ ├── create\_user.hmtl

│ │ ├── data\_export.html

│ │ ├── job\_scheduling.html

│ │ └── login.html

│ ├── \_\_init\_\_.py

│ ├── auth.py

│ ├── configuration.py

│ ├── data\_export.py

│ ├── decorators.py

│ ├── job\_scheduling.py

│ └── models.py

└── wsgi.py

### Directories

* ui
  + Under the project's root directory
  + Holds all ui related directories, along with flask init and blueprint files.
* ui\_ouput
  + Used to store any UI related output. (i.e. data exported from the export data page.
* static
  + Holds project static files such as images, and css and javascript files.
* templates
  + Holds all project HTML files.

### Templates

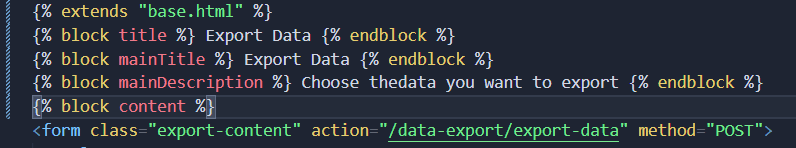
The templates directory is used to hold all HTML code for the user interface. Each UI page will have a file in this directory. There are some things to note about these files:

* base.html
  + Used as a skeleton for all other templates.
  + Includes the UI’s navigation pane so it can be reused on all pages.
  + Page content is put into Jinja blocks for modularity:



*(Figure 7.2.1.1) base.html blocks*

* login.html
  + Layout for the login page.
  + Does not use *base.html,* no navigation is needed.
* Page Templates (configuration.html*,* data\_export.html*, etc…)*
  + Pages for UI content and functionality.
  + Extends *base.html* using Jinja templating.
  + Jinja blocks are used to provide page titles and descriptions, along with a block for the pages main content. See *Figure 7.2.1.2* below.



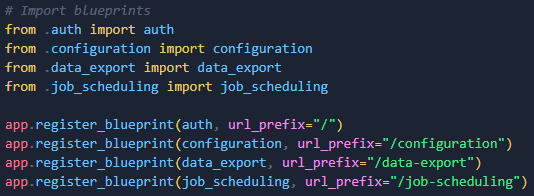
*(Figure 7.2.1.2) Page template block setup*

### Blueprints

Flask Blueprints are used to break the UI functionality for each page into their own files. Blueprints add modularity and clarity to the application structure, allowing functionality to be added or removed easily. In the current version of the app, the following Blueprints are defined:

* auth.py
  + Used to handle user login, authentication, and user registration.
* data\_export.py
  + Functionality for the Export data page
* job\_scheduling.py
  + Provides functionality for changing when jobs are to be run
* configuration.py
  + Provides functionality for changing configuration files via the user interface.

Note: Once a Blueprint is created, it must be registered in \_\_init\_\_.py (*Figure 7.2.1.3)*. For more information, visit the [Flask Blueprint Documentation](https://flask.palletsprojects.com/en/2.3.x/blueprints/)



*(Figure 7.2.1.3) Blueprint registration in \_\_init\_\_.py*

### Other Files

There are a couple other files that are needed for the ATS user interface to function properly:

* models.py
  + Used to define models (tables) to be used by Flask.
  + Clones and tracks changes to tables in our MySQL database.
* \_\_init\_\_.py
  + Used for application setup and creation
* decorators.py
  + File for custom function decorators
    - E.g. admin\_required decorator used for user access control
* wsgi.py 
  + The main code runner file, used to start the Flask application
  + Provides a wsgi entry point for the application.
  + Located under the project's root directory.

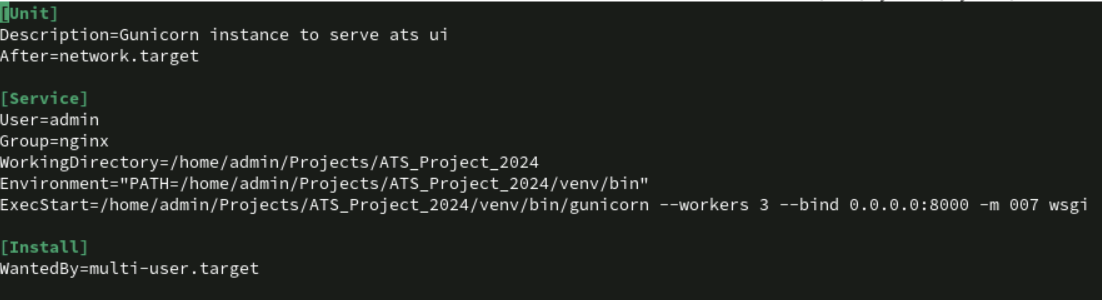
## UI Flask App Deployment

In order to deploy the ATS UI to a production environment, a WSGI server such as [Gunicorn](https://docs.gunicorn.org/en/stable/) must be installed and configured to provide an entry point into the application. It is common practice for Flask app deployment to configure a web server or reverse proxy such as [Nginx](https://nginx.org/en/docs/), to route external traffic to the application. Below are some general steps to follow for deployment. This setup was done on a AlmaLinux9 server using a Gunicorn WSGI server . Note: This is not an all encompassing setup guide, steps may differ depending on the Linux distribution of your server, and the setup of your environment. For more information on deployment, refer to the Flask documentation:

<https://flask.palletsprojects.com/en/2.0.x/deploying>/

### Setup

1. Before getting started you will need to set up the ATS Project environment:
   1. Ensure the project is downloaded on your server in your admin user directory.
   2. Follow the setup instructions in the README.md in the project's root directory to set up your virtual environment.
2. Ensure that the WSGI server is installed in your virtual environment:
   1. pip install gunicorn
3. Next, deactivate your virtual environment or open a new terminal window.
4. Now you will need create a Systemd unit file which allows the servers initialization system to start Gunicorn and serve the application when the server boots
   1. sudo nano /etc/systemd/system/ats.service
   2. Put the following contents into the file:



* 1. Ensure that all file paths under the [Service] section are accurate with your setup.
     1. WorkingDirectory: The path to the projects root
     2. Environment: The path to the python virtual environment
     3. ExecStart: Command to execute the service. Path must point to the location of your Gunicorn installation. “wsgi” tells Gunicorn to run the UI executable wsgi.py located in the project root.
  2. Note that service is bound to 0.0.0.0:8000. You will need to configure your Nginx proxy to listen on whatever port your service is bound to, in this case the service is bound to port 8000.

1. Now, run the following commands to start and enable the service:
   1. sudo systemctl daemon-reload
   2. sudo systemctl start ats
   3. sudo systemctl enable ats
   4. Run sudo systemctl status ats to ensure the service is running.
2. Now you will need to configure an Nginx server to proxy requests:
   1. Vist the [Nginx](https://nginx.org/en/docs/) documentation for information on creating a custom Nginx configuration.
   2. The proxy server requirements can change depending on your setup
      1. For example, our development environment is using a dedicated Nginx proxy server running [Nginx Proxy Manager](https://nginxproxymanager.com/), where configuration is done via a Web UI.
3. Lastly, ensure that no firewall rules are restricting traffic for the proxy into your app.

## IONOS Web Hosting

Due to IONOS being a shared web hosting service, it does not provide us with root permissions on the server. With this restriction, it is near impossible to deploy a Flask application on IONOS. Due to these implications, it is recommended to deploy the ATS UI on a dedicated linux server on an internal network (Okanagan Colleges COSC network) or on virtual private server (VPS) hosted by a cloud provider.

# Running the System Locally

This section will provide a brief overview of the steps required to run the system locally without IONOS hosting.

## Steps

* Github repository
  + Navigate to project github repo at <https://github.com/COSC-470-2023/ATS_Project_2024>
  + Either download the latest release, or clone the project and checkout the desired branch
* Database
  + Install Mysql 8.0 on your machine. <https://dev.mysql.com/doc/refman/8.0/en/installing.html>
  + Create the database using *create\_db.sql* located in the ddl/ directory
  + Create the database triggers using *create\_triggers.sql* (same directory)
* Setup
  + Copy example.env and rename the copy to .env
  + Fill in the .env file with the necessary credentials
* Modify configuration yaml files
  + Examine and modify files located in the config directory if necessary
* Cron jobs
  + Setup up *cron* jobs to run the Python scripts both for the collection of data and the insertion of the data into the database.
  + The shell scripts used to run the collection and insertion scripts are located in the scripts/ directory.

# Server Setup

## Pre-Requisites

* 1. IONOS Server
     1. Linux-based OS
     2. Python 3.x.x installed
     3. MySQL installed
     4. Git installed
     5. Network Connection
  2. Client/Developer
     1. SFTP client installed
     2. SSH client installed\*
     3. Required Credentials
        1. IONOS Credentials
        2. SSH/SFTP Credentials
        3. MySQL Credentials
        4. API Keys

\*If on Windows or Ubuntu, you can use the SSH command in a terminal.

## Using SSH

* 1. Log into the IONOS dashboard and find the SSH module under hosting overview
  2. Find the server address, username, and password
  3. If using the SSH command:
     1. copy the following command and replace the variables:

ssh <server address> -l <username>

* + 1. Once prompted for the password, enter the password you found in step 2.b.
  1. If using an SSH Client:
     1. Supply the hostname, and credentials
     2. Click connect.
  2. Using the new SSH terminal, you should be able to interact with the server.

## Git Pulling

* 1. Using the SSH connection from step above (7.2), use git to pull down the github repository of the current iteration of the system.
  2. You may wish to switch the branch of the project, as the default branch is “main”. Use the following command to switch the branch to the dev build:

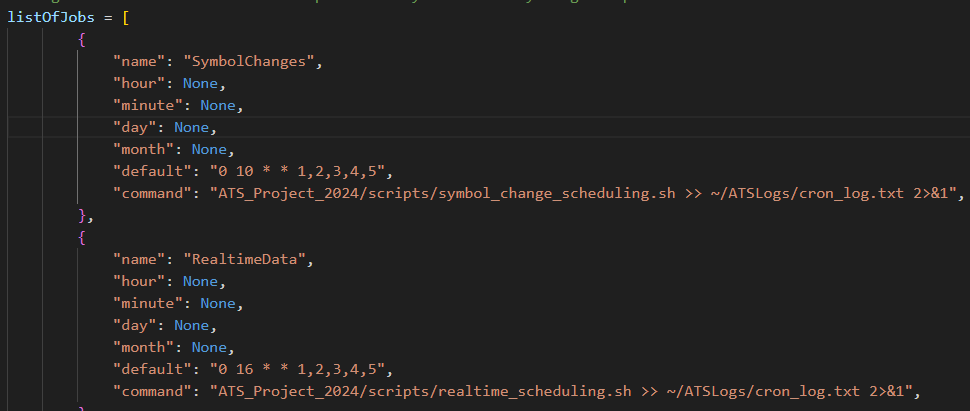
git checkout develop

## Using SFTP

* 1. Log into the IONOS dashboard and find the SSH/SFTP module under hosting overview
  2. Find the server address, username, and password (SFTP should be the same as SSH in most cases)
  3. Use your client and provide the info collected in step 7.4.b

## Setting up CRON jobs

* 1. Once the server has the project download, you will need to set up cron jobs.
  2. Start up the flask server and run wsgi.py.
  3. Open a web browser and log into the system with valid credentials.
  4. navigate to the job scheduling page.
     1. Select the default option and select apply changes for each job.
        1. See the user manual for a visual representation of how to do this.
  5. To edit the jobs times select custom on the job scheduling page
     1. Edit the data fields with valid times
     2. Click apply changes
     3. Repeat as necessary
  6. If modification of the default settings is required.
     1. Navigate to the UI directory of the project and modify the end of the file and modify the jobs information in the relevant fields.
     2. Save changes and return to step b.



# Use Case Diagram

See the ATS Software Design Document [[R3](https://docs.google.com/document/d/1kH4S7RFlHHq6SzOlhkYMcuc-0IA7yYNazzqXoAM9lVY/edit?usp=sharing)], section 3 ‘User Characteristics”, for a Use Case Diagram and breakdown of the primary user interactions with the system.

# Architecture

See the ATS Software Design Document [[R3](https://docs.google.com/document/d/1kH4S7RFlHHq6SzOlhkYMcuc-0IA7yYNazzqXoAM9lVY/edit?usp=sharing)], section 5 ‘System Architecture’ for various system diagrams and details about architecture implementation.

# Testing

Tests for the system are located in the specified testing sub-directories for each part of the system they test. Eg. tests/ats/collection/test\_symbol\_change\_query.py

Descriptions:

* **API output unit test**: Proves the functionality of *write\_files()*. The output of stock, index, and commodity is proven to have the correct yaml format and corresponding value.
* **Index insertion unit test**: Simulates the process of index insertion by calling *self.cursor.execute (“INSERT INTO xxx”)* and compares the results between the inserted entries and the static data.
* **Stock insertion unit test**: Prove the function *realtime\_stock\_insert.excute\_insert* and *get\_company\_id()* is operational.
* **Symbol change insertion unit test**: Create the empty files, convert them to the lists, and pass the lists to the *update\_symbol(). update\_symbol* is functioning.
* **Commodities insertion unit test**: This unit test tests the historical and real time data by taking data from test files and executing insertions into the database and testing the data from the test files against the data from the database. If it’s null then the insertions failed otherwise we check the value against a preset test value and determine if the outcome was successful.