***Algorithmic Trading***

***System***

***Software Design Document***

Okanagan College

Algorithmic Trading System

2024-04-09

Version 1.8.2

Authors: Jacob Rawlings,Jake Fischer**,** Alan Abdollahzadeh, Devon Volberg, William Blake, Benjamin Carrier, Vanessa Dubouzet, Dakota Flath, Parker Green, Yuan Hu, Sachin Kamboj, Jacob Labelle, Isaac Lengacher-Bergeron , Dominic Presch, Gurneer Singh Sandhu, Dylan Soares, Jaeden Soukoroff

# TABLE OF CONTENTS

[**TABLE OF CONTENTS 1**](#_heading=h.gjdgxs)

[**1 Introduction 1**](#_heading=h.30j0zll)

[**1.1 Purpose 1**](#_heading=)

[**1.2 Scope 1**](#_heading=)

[**1.3 Definitions, Acronyms, and Abbreviations 1**](#_heading=h.3znysh7)

[**1.4 References 2**](#_heading=)

[**2 Design Overview 2**](#_heading=)

[2.1 Background Information 2](#_heading=h.3as4poj)

[Stakeholders 2](#_heading=h.49x2ik5)

[Project Risks & Issues 2](#_heading=h.23ckvvd)

[Assumptions 3](#_heading=h.ihv636)

[Dependencies 3](#_heading=h.32hioqz)

[**2.2 Alternatives 3**](#_heading=)

[**3 User Characteristics 6**](#_heading=)

[3.1 Potential System Users 6](#_heading=h.meukdy)

[3.2 System Design 9](#_heading=h.36ei31r)

[**4 Requirements and Constraints 9**](#_heading=)

[4.1 Performance Requirements 9](#_heading=h.1ljsd9k)

[**4.2 Security Requirements 10**](#_heading=)

[**4.3 Design Constraints 10**](#_heading=)

[**5 System Architecture 11**](#_heading=)

[**6 Detailed Design 15**](#_heading=h.261ztfg)

[**6.1 Data Collection 17**](#_heading=)

[Processing for Data Collection 17](#_heading=h.l7a3n9)

[Data Collection Interface Description 18](#_heading=h.356xmb2)

[6.2 OLTP Database 19](#_heading=h.1kc7wiv)

[Processing for Database 22](#_heading=h.44bvf6o)

[Database Interface Description 23](#_heading=h.2jh5peh)

[6.3 Design Alternative - Data Warehouse 23](#_heading=h.ymfzma)

[Processing for Warehouse 23](#_heading=h.3im3ia3)

[Warehouse Interface Description 24](#_heading=h.1xrdshw)

[**7 Data Architecture 24**](#_heading=h.4hr1b5p)

[**8 Interface Requirements 25**](#_heading=)

[8.1 Required Interfaces 25](#_heading=h.2gb3jie)

[8.1.1 IONOS 25](#_heading=h.vgdtq7)

[8.1.2 Financial Modelling Prep API 25](#_heading=h.hppq2xv45k9r)

[8.1.3 Machine Learning Frameworks 25](#_heading=h.1ulbmlt)

[8.2 External System Dependencies 25](#_heading=h.4ekz59m)

[**9 User Interface 25**](#_heading=)

[IONOS 26](#_heading=h.1maplo9)

[phpMyAdmin 26](#_heading=h.o9iyhlbqol7b)

[ATS User Interface 27](#_heading=h.figfd7yrtdn9)

[**10 Non-Functional Requirements 29**](#_heading=)

**Revision Sheet**

| Revision | Date | Brief Summary of Changes |
| --- | --- | --- |
| Version 0.0(draft) | 2023-10-01 | Baseline document draft |
| Version 0.1(draft) | 2023-10-01 | Document structure expanded |
| Version 0.2 | 2023-10-01 | Began writing sections 1-3 |
| Version 0.3 | 2023-10-02 | Completed sections 1-3, began sections 4, 5 |
| Version 1.0 | 2023-10-03 | Version 1.0 completed - all sections written (except 9 - out of scope) |
| Version 1.1 | 2023-10-10 | Slight modification to DC and DB text to reflect design changes |
| Version 1.2 | 2023-10-10 | System architecture: changed MS to pipeline  Authors revision  Removed devs from stakeholders |
| Version 1.3 | 2023-10-12 | Revised top level pipeline process diagram  Revised content to ensure accurate reflection of system processes  Inserted diagram demonstrating current implementation  Added “user interface section” with images  Removed old process flow diagram  Added DC activity diagram |
| Version 1.4 | 2023-10-22 | Revised/added documents including: Class Diagram, Use Case Diagram, Domain Diagram, Activity Diagram |
| Version 1.5 | 2023-11-05 | Added additional design diagrams including: Class diagrams, domain diagrams, and activity diagrams. Reviewed and updated related sections. |
| Version 1.6 | 2023-12-07 | General editing and updating. Maintaining consistency with current project design. Update UCD. |
| Version 1.7 | 2024-01-30 | General revisions. Removal of DW mentions in places. Adding YAML design changes. Adding initial UI general info. |
| Version 1.8 | 2024-02-28 | User Interface: Updated ATS User Interface section  Updated use case diagram  Updated architecture diagram |
| Version 1.8.1 | 2024-03-18 | Class diagram 2 and 4 minor fixes |
| Version 1.8.2 | 2024-04-07 | Updated User Interface screenshots |
| Version 1.8.3 | 2024-04-09 | Final Revisions |

# Introduction

## Purpose

The purpose of this document is to outline the design of the Algorithmic Trading System (ATS) software system for the automatic collection and storage of information on various financial assets. This information will be analyzed by the XGBoost Machine Learning (ML) model to generate short-term forecasts for market trends, such as asset prices. The ATS will provide a robust dataset for ML model analysis, thus serving financial analysts and investors by supplying timely, detailed, and accurate financial asset forecasts.

## Scope

The ATS system will retrieve a variety of different financial data, including information on: stocks, index composites, treasury bond rates, and commodity prices. Periodically the software will also retrieve company information such as: the number of employees, industry sector, market cap, revenue, etc.

Data collected by the system will be cleaned and preprocessed, and inserted into an OLTP database for intermediate storage purposes. Researchers and data analysts will be able to pull from this database via a user interface for use with their ML models.

The intended application of the system is to provide ML models with a greater quantity of data that is broader in scope to promote quality analysis and greater accuracy in forecasting. Agile design methodologies will be employed to create and manage the system, with a focus on modularity for the developed system.

This document is subject to revision throughout the project lifecycle due to future scope or design changes.

## Definitions, Acronyms, and Abbreviations

* Cleaning: The process of validating data to ensure accuracy and consistency
* Dow Jones index: stock market index of 30 prominent companies listed on stock exchanges in the United States
* S&P 500 index: stock market index that tracks the stock performance of 500 large-cap U.S. companies
* NASDAQ index: stock market index that includes almost all stocks listed on the Nasdaq stock exchange (more than 2500 stocks)
* API: Application Programming Interface
* URI: Uniform Resource Identifier
* ML: Machine Learning
* ATS: Algorithmic Trading System
* DC: Data Collection
* DB: OLTP Database
* DW: Data Warehouse
* ETL: Extract, Transform, Load
* Transit: The transfer or exchange of data between different components
* {Text}: Text that is subject to change in this document

## References

* [Database Design Document](https://drive.google.com/file/d/1hXYSwt-7PjY0tIh1io-94s8UpBedwiSX/view?usp=drive_link) [R1]
* [STSPF Research Paper](https://arxiv.org/abs/2309.00618) [R2]
* System Requirements Specification [R3]
* ATS Requirements Traceability Matrix [R4]
* [MySQL Connector package](https://www.mysql.com/products/connector/) [R5]
* [SQL Alchemy Documentation](https://docs.sqlalchemy.org/en/20/) [R6]
* [Financial Modelling Prep](https://site.financialmodelingprep.com/developer/docs/pricing) plans [R7]
* [IONOS web hosting plans](https://www.ionos.com/hosting/web-hosting#plans) [R8]
* ATS Developers Guide [R9]

# Design Overview

## Background Information

Recent developments with AI and machine learning software have resulted in significant disruption across multiple industries. The ATS software is being developed to fulfill the demand for data driven decision making in the investment industry by leveraging machine learning models to perform analysis of large datasets.

### Stakeholders

Albert Wong: [alwong@langara.ca](mailto:alwong@langara.ca)

Gaetan Hains: [gaetan.hains@proton.me](mailto:gaetan.hains@proton.me)

Youry Khmelevsky: [ykhmelevsky@okanagan.bc.ca](mailto:YKhmelevsky@okanagan.bc.ca)

### Project Risks & Issues

* **Data Reliability**: Ensuring the accuracy of financial data is vital. If data is unreliable, it could result in significant financial consequences for users.
* **Volume:** The system must be able to store several years worth of data, collecting data from 503 companies every day
* **Backups:** Backups must be kept in case of unexpected database issues.
* **Sources**: Data collection is dependent on external data sources. If there is a problem with external data providers, then new record insertion could be affected.
* **Machine Learning Model Accuracy**: The accuracy of machine learning models depends on the quality and timeliness of the collected data. Data consistency validation and insertion error handling must be robust to not compromise data integrity.

### Assumptions

* External data sources are trusted and readily accessible.
* The system has sufficient computing resources for data collection and processing.
* The database hosting service will support a storage engine that facilitates a high level of read and write operations.
* The database hosting service will have sufficient storage capacity.
* The database hosting service will have sufficient support for warehousing operations.

### Dependencies

* Available infrastructure for data storage
  + Currently provided by IONOS 3rd-party hosting
* MySQL V8.2
* Python 3.12
* Git
* Financial Modelling Prep API availability

## Alternatives

**Programming Language Alternatives**

The proof of concept design created prior to this project's inception utilized PHP as the primary programming language for processes. PHP was discarded after initial exploratory work revealed the PHP interpreter was not part of the default installation package for the IONOS hosting environment. Python was selected as the implementation language due to the language being part of the default installation packages, usability, library support, and widespread documentation.

**Data Collection Design Alternatives**

XML was the one format that was considered as the data format used during transit between components. XML is a markup language and requires a tag structure which makes it overly complex for this use case. Due to the configurable nature of the input files, it was decided that files needed to be more readable and usable by humans. Collected API data does not consist of any complex binary data types (i.e. images, videos, etc…), therefore there is no need for XML’s data exchange properties.

JSON was considered as the next data format, and is what was implemented in the prototype and first release of the ATS system. JSON was chosen as it is more user friendly for configuring. It was also the data format returned for API output, so it was thought that maintaining that consistency would be beneficial. However, what we found was that using JSON added complexity to script logic due to the dynamic and malleable nature of the ATS configuration. JSON is generally used for data storage, rather than system configuration.

For development of the next major release, it was decided that configuration would be migrated to YAML format. YAML is a commonly used format for storing configuration data. It is also easily readable, allows for comments, and most importantly allows for referencing other data objects - ensuring the ability to serialize complex data structures.

**Data Processing Design Alternatives**

The original design of the Data Processing process called for the use of the MySQL-Connector [R5] package. However, this was discarded and replaced with the SQLAlchemy package [R6].

Despite the challenges of familiarizing the team with SQLAlchemy, it was deemed necessary as the team pursued a goal of designing the system with as many system agnostic components as possible. SQLAlchemy utilizes Object Relational Mapping instead of plain language SQL, making it an ideal candidate if the underlying database management system were to change.

**Database Schema Design Alternatives**

* Index Composite Data
  + The original design called for 3 separate tables to store data specific to the DowJones, S&P500, and NASDAQ indexes
  + This design was revised for the following reasons:
    - Extensibility: the addition of new indexes to the dataset would require the creation of new tables for each additional index
    - Redundancy: iteration of the data collection and processing testing revealed each index had identical data fields

* Company Data - Iteration 1
  + The original design called for 2 tables:
    - company\_stocks
      * Used to hold stock related data such as:
        + open price
        + close price
        + high price
        + low price
        + etc…
    - company\_info
      * Used to store quarterly company data such as:
        + company name
        + stock symbol
        + industry
        + sector
        + etc…
  + The design was revised for the following reasons:
    - Updates: if a symbol or company name changed, it would create update anomalies as each row would associated with the update would need to be updated
    - Tracking: for the same reason above, a revision was necessary to keep track of symbol and name changes for historical purposes

* Company Data - Iteration 2
  + This iteration of the design called for 5 tables:
    - stock\_ids
      * used to hold a unique record for each stock
        + Each row contains a stock symbol and its unique ID.
    - stock\_values
      * Used to hold stock related data such as:
        + open price
        + close price
        + high price
        + low price
        + etc…
    - company\_ids
      * used to hold a unique record for each company
        + Each row contains a company name and its unique ID.
    - company\_statements
      * Used to store quarterly company data such as:
        + company name
        + stock symbol
        + industry
        + sector
        + etc…
    - changelogs
  + This design was revised for the following reasons:
    - Redundancy: company\_ids and stock\_ids contained similar data, so they were consolidated in a revision
    - Querying: CompanyName and Symbol were kept in separate tables, requiring a query to retrieve one or the other which would put unnecessary traffic load on the database server and complicated the creation of IDs for newly tracked stocks and companies
* Bonds Data
  + The original design called for separate tables: 2yr, 5yr, 10yr
  + This design was revised for the following reasons:
    - Extensibility: the addition of new treasuries to the dataset would require the creation of new tables for each additional bond
    - Redundancy: the bonds contained identical data columns, thus the decision to consolidate them was made
    - Accuracy: keeping all rates for the same day in the same table makes more sense from a logical and user experience perspective

**Data Warehouse Design Alternatives**

* Star Schema
  + The original warehouse was designed with a star schema in mind due to limitations of the hosting environment and project needs
  + This design was revised for the following reasons:
    - Separation: It is important to keep each data section separate, as not all sections contain the same data fields
    - Calculation: The fact table stores quantitative data that we need to calculate. A fact table would have been needed for each data type, due to the separate data sections.
    - Null Values: One fact table could have been used but would require the use of many null values, which the team wished to avoid

# User Characteristics

## Potential System Users

* + 1. **Data Analysts**
* Expertise: Moderate

Interactions:

* Use ML models for comprehensive data analysis
* Retrieve data from database for use with ML models
* Requires ability to access historical and current data
* Requires accurate and up-to-date data
* Requires assistance of Data Administrator to make change requests
  + 1. **Database Management Team**
* Expertise: High

Interactions:

* Assist in database management
* Manually loads or deletes data when required
* Write and update database triggers
* Inherits interactions of Data Analyst
  + 1. **Database Administrator**
* Expertise: High

Interactions:

* Performs database maintenance
* Monitors performance of the database systems
* Manages database schema and backups
* Requires full permissions and access to database
* Inherits interactions of Database Management Team
  + 1. **Data Administrator**
* Expertise: Very High

Interactions:

* Manage data sources
* Initiating manual data retrieval and processing operations
* Managing configuration of the data retrieval systems
* Monitoring performance of data retrieval systems
* Requires full access to data retrieval system
* Inherits interactions of Database Management Team
  + 1. **Student Research Assistant**
* Expertise: Very High

Interactions:

* Inherits all interactions of Database Administrator, Data Administrator, and Data Analyst.

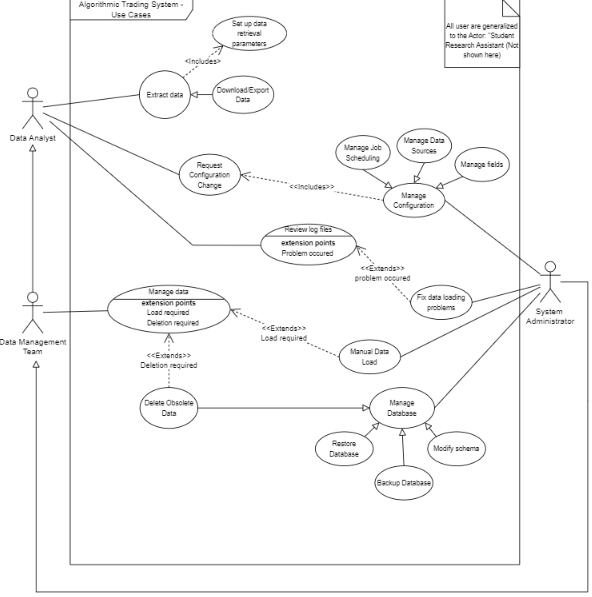


Figure 1: Use Case Diagram

## System Design

To meet client and user requirements, various system design features must be implemented:

1. Ensure data is objective, accurate, and up-to-date.
2. Ensure data is collected automatically at regular, predetermined intervals.
3. Data must be collected from the Financial Modelling Prep API.
4. The system should be constructed in a manner that handles different RESTful API providers.
5. Data processing should be conducted within 24 hours maximum.
6. Ensure consistency between data collection output and database.
7. Ensure compatibility with the XGBoost ML model.

# Requirements and Constraints

## Performance Requirements

To meet client and user requirements, the following performance requirements must be met:

1. Data collection scripts for stocks, index composites, and commodities must be run on a daily basis.
2. Treasury bond scripts must be run on a weekly basis
3. Company information and earnings scripts must be run on a monthly basis.
4. Data collection should have the capacity to run up to 300 API calls per minute. This is the current limit imposed by the subscription plan to the Financial Modelling Prep service [R7].
5. Collected data should cleaned, processed, and available for retrieval within 24 hours from collection
6. The system should be able to scale horizontally up to 500 individual companies, 3 Index composites, and 3 years of bond rates.
7. Data must be available for retrieval during and outside of trading hours.
8. Data should be stored for 3 years. After which point it can be considered ‘obsolete’ and should be automatically deleted from the database. This process must occur monthly.
9. The system should function automatically, requiring minimal manual intervention.

These conditions will be met by designing an effective and efficient process flow; well structured database design; and leveraging cron jobs for automated scheduling.

## Security Requirements

The following security concerns must be addressed:

* All sensitive data must be secure through transit and when at rest
  + - This includes API keys and SSH or environment credentials
* Sensitive data will not be present in public documentation or repositories
* Administrative systems should not be accessible to the public internet

The IONOS hosting environment provides server security and user authentication through their platform, which includes secure shell and secure file transfer protocol access. API key and login credential security concerns will be managed by closely monitoring development environments to ensure they are censored or redacted before being pushed to the source code repository.

## Design Constraints

The ATS project is on a limited schedule. A functional software artifact must be delivered by April 12, 2024.

Software and hosting services are constrained to what has been provided by Okanagan College and the project sponsors.

* Languages used by the system must be supported by Debian11, the OS used by IONOS
* Package dependencies must be supported by Debian11
* SQL syntax must be supported by MySQL or MariaDB, the DBMS used by IONOS
* Database sizes must not exceed 2GB, the maximum size supported by the current IONOS subscription
* System processes must not exceed 600MB of RAM for usage, a limitation of the environment provided by IONOS [R8]

The project team is composed entirely of students, resulting in the following constraints:

* The project team will have restrictions on times they can meet, due to the following factors
  + Students have work scheduled outside of class time
  + Students have varied class schedules
* Time that can be allocated to the project outside of scheduled classes is competing with other priorities such as exams, etc.

These constraints are being addressed by:

* Employing timeboxing as a strategy to minimize unnecessary time spent in meetings or discussion
* Employing agile principles
* Establishing dedicated blocks of time for working
* Leveraging Confluence, Jira, and system versioning tools
* Leveraging flexible collaboration tools to facilitate online meetings

# System Architecture

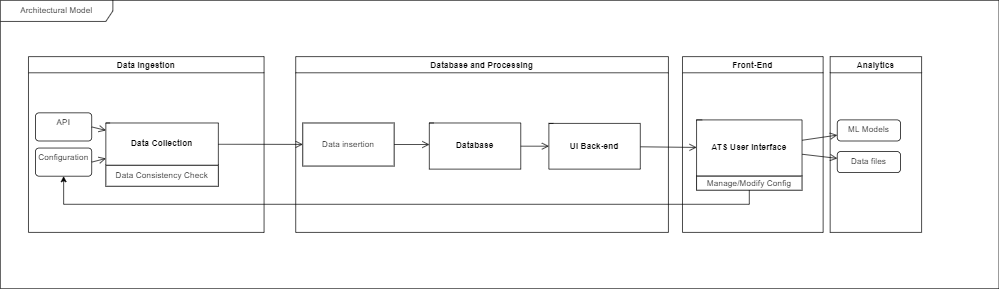
The system will be deployed in an environment provided by IONOS - a third party hosting service - for database and cloud storage. The system follows a pipeline software architecture model to promote horizontal scaling, modularity, and reusability. There are three main components of the pipeline: Data Ingestion, Database and Data processing. 

Figure 2: Top-level diagram of the ATS pipeline architecture

The Data Collection component contains query logic to support the processes that request data from API sources, clean, and preprocess the retrieved data. Each data retrieval process is constructed to be independent of the others, following modular design principles, to reduce the corrective resources required in the event one of the processes fails. The processes in this system component are written in Python 3, and utilize YAML and JSON as the exchange formats through the data pipeline. A configuration file is loaded by each query process to determine the API parameters, subsequently determining which information is returned.

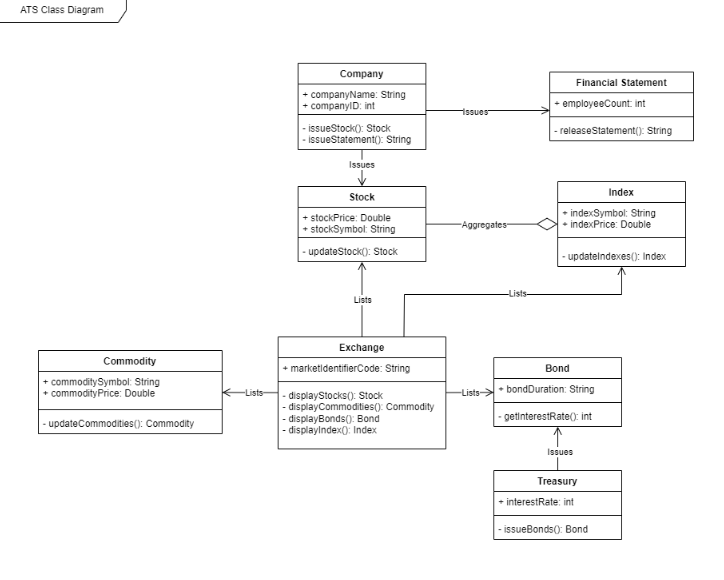


Figure 3: Class Diagram 2 - Assets

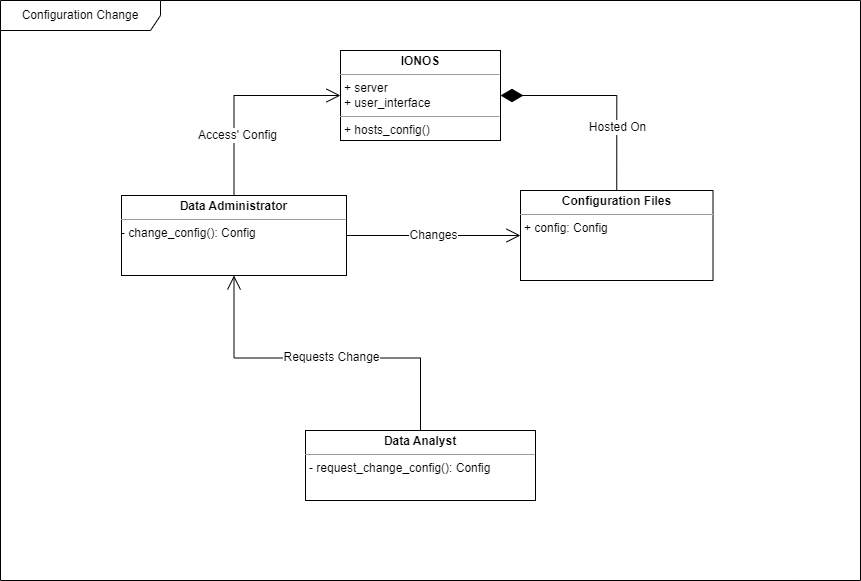


Figure 4: Class Diagram 3 - Configuration Settings

The Database and processing component contains processes that support secondary validation of data, organization of data, and insertion of the data into the database. The database is structured to take advantage of triggers and constraints to supplement the external scripted processes. Processes in this component use Python 3 and SQL, with JSON as the data input object.

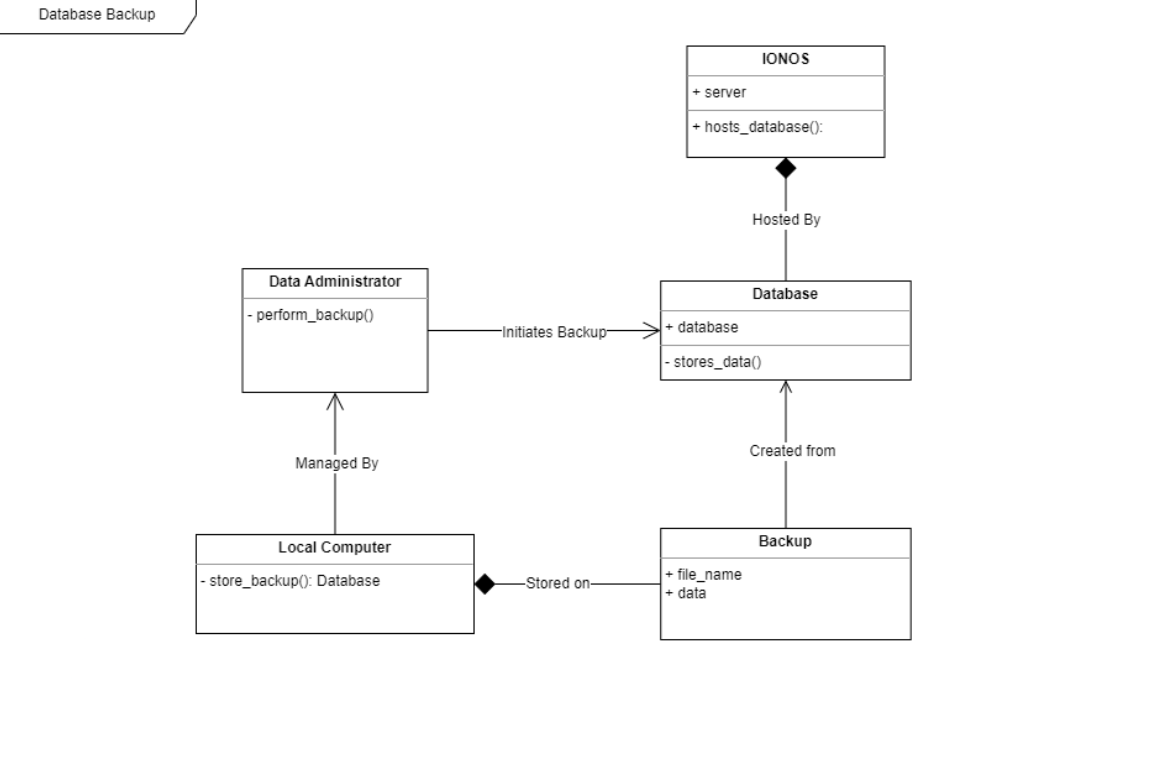


Figure 5: Class Diagram 4 - DB backup

The Data Storage component contains processes for extracting, transforming, and integrating the data from the Database component. The warehouse will be used as a long term storage facility, and will be populated every 3 months. Data stored in the warehouse is provisioned for analysis by ML models, which have an independently developed process for extracting the data outside the scope of the current project. It is expected that processes in this component will use a combination of SQL and other language(s), but the design of this system component is still in early iteration.

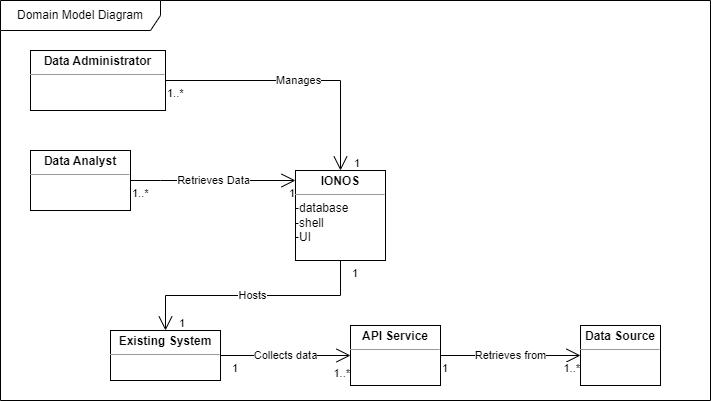


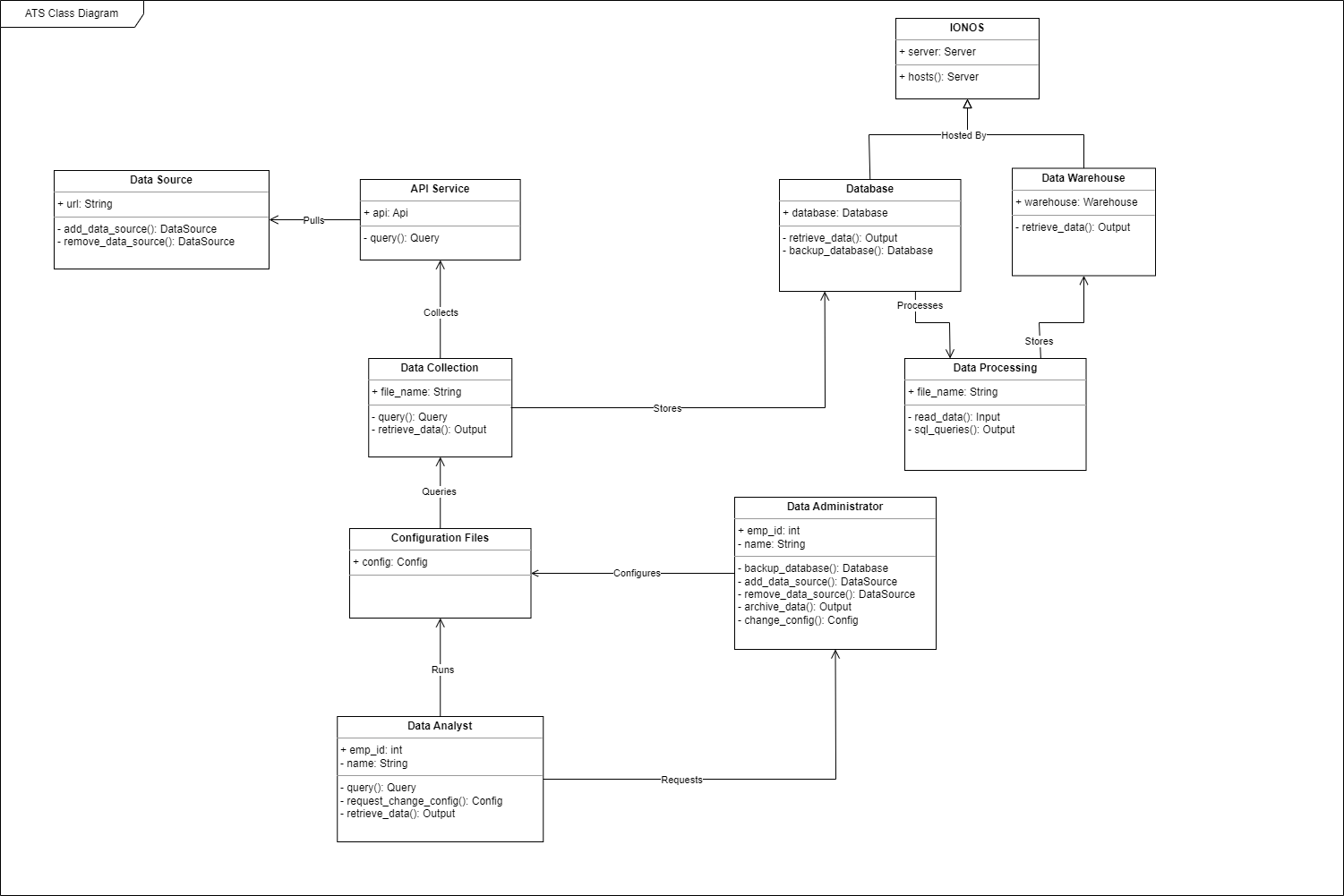
Figure 6: General Domain Model Diagram

Figure 7: Class Diagram 1 - overall system

# Detailed Design

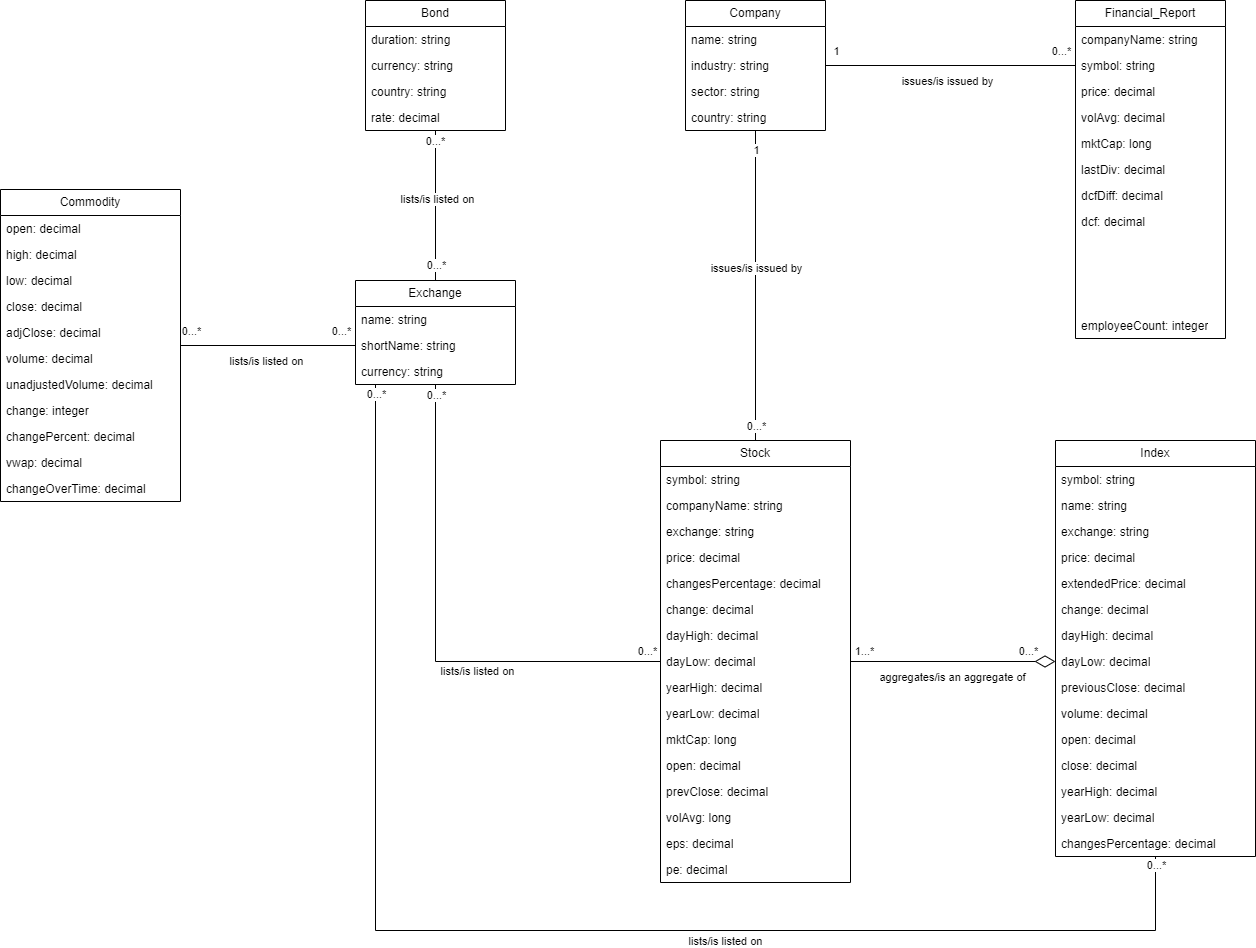


Figure 8: Data Domain Model Diagram

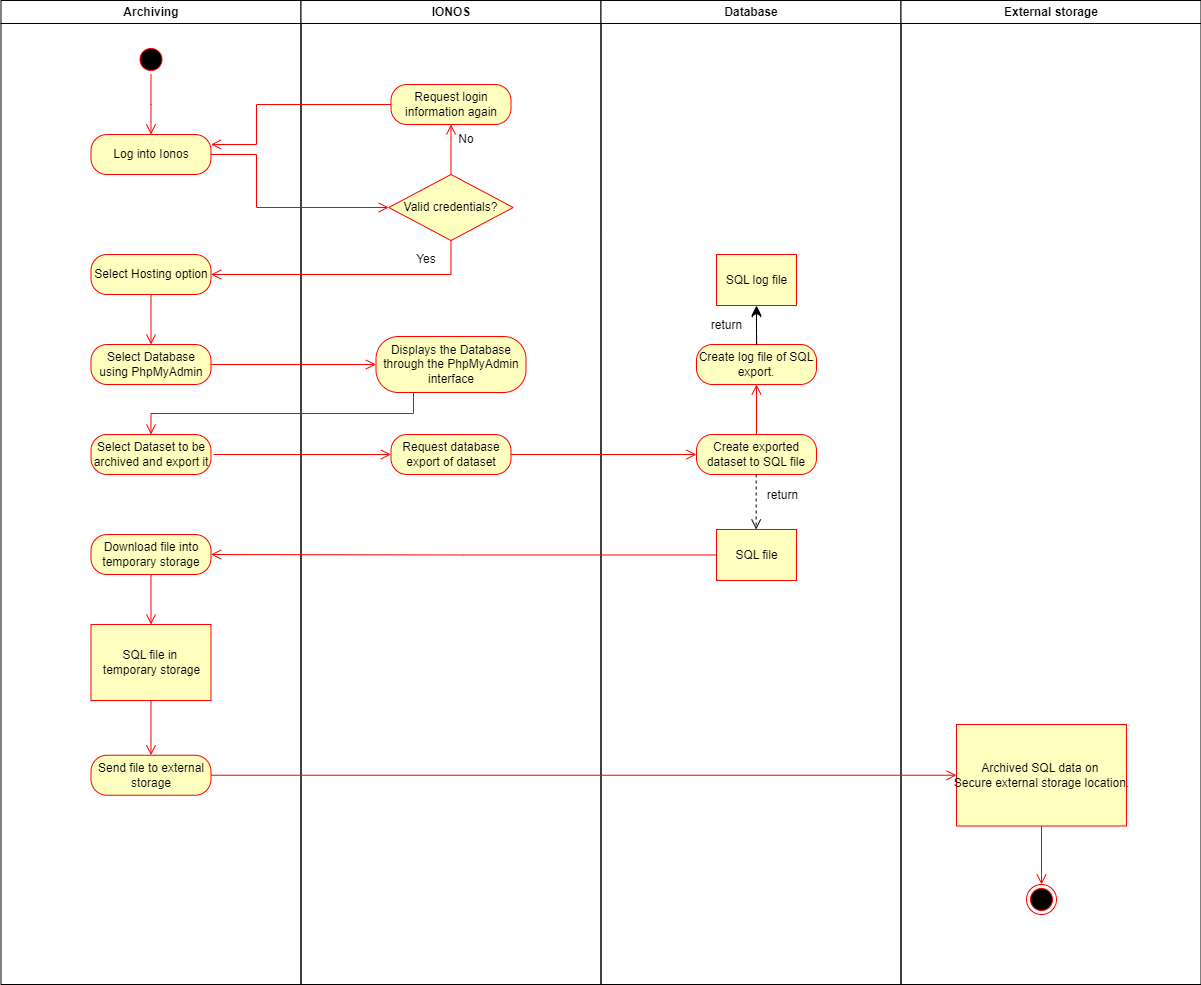


Figure 9: Archiving Data Activity Diagram

## Data Collection

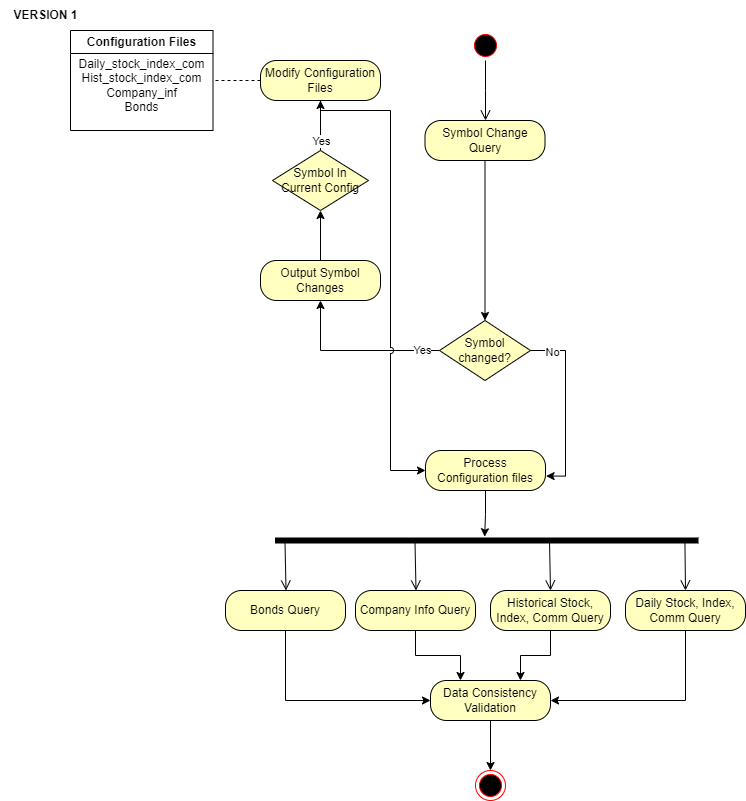


Figure 10: Data collection activity diagram

### Processing for Data Collection

The system retrieves various parameters from the configuration files. All configuration files contain a series of key value pairs associated with different uses. Fields shared among all configuration files are: url, api\_key, rate\_limit\_per\_min, api\_fields, and non\_api\_fields. For historical and real time configurations, a list of asset symbols and names are also present. For a detailed description of the configuration files, see the ATS Developer’s Guide [R9]. Query scripts process these configuration files, prepare API calls, execute API calls, and return data. The data returned is processed for consistency validation, and written to an output file for each respective process.

Processing should execute symbol change queries first, to ensure configuration files and database fields are kept up to date in the event companies change names or symbols. The execution of remaining queries can be conducted with no preference for order. Collection scripts are set to be run at scheduled intervals via cron jobs. Stock, Index, and Commodity collection will occur daily at market close. Treasury bonds will be collected weekly. Company information will be collected monthly.

### Data Collection Interface Description

Input:

* Configuration Files: YAML format
* Financial Modelling Prep API Data

Output:

* Raw Query Data: JSON format

## OLTP Database

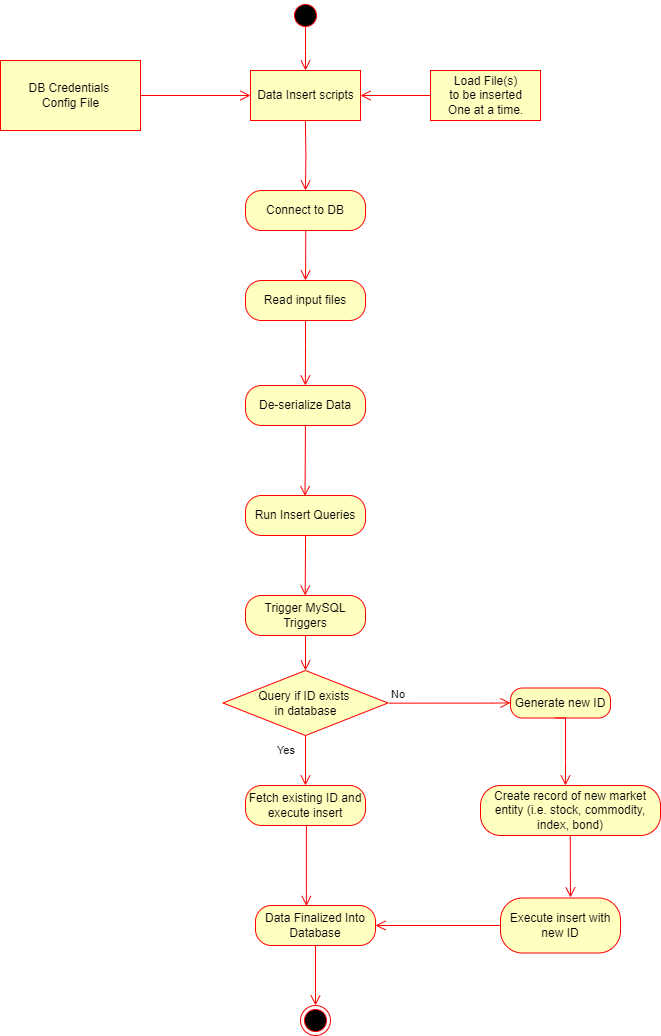


Figure 11: Data Insertion Activity Diagram

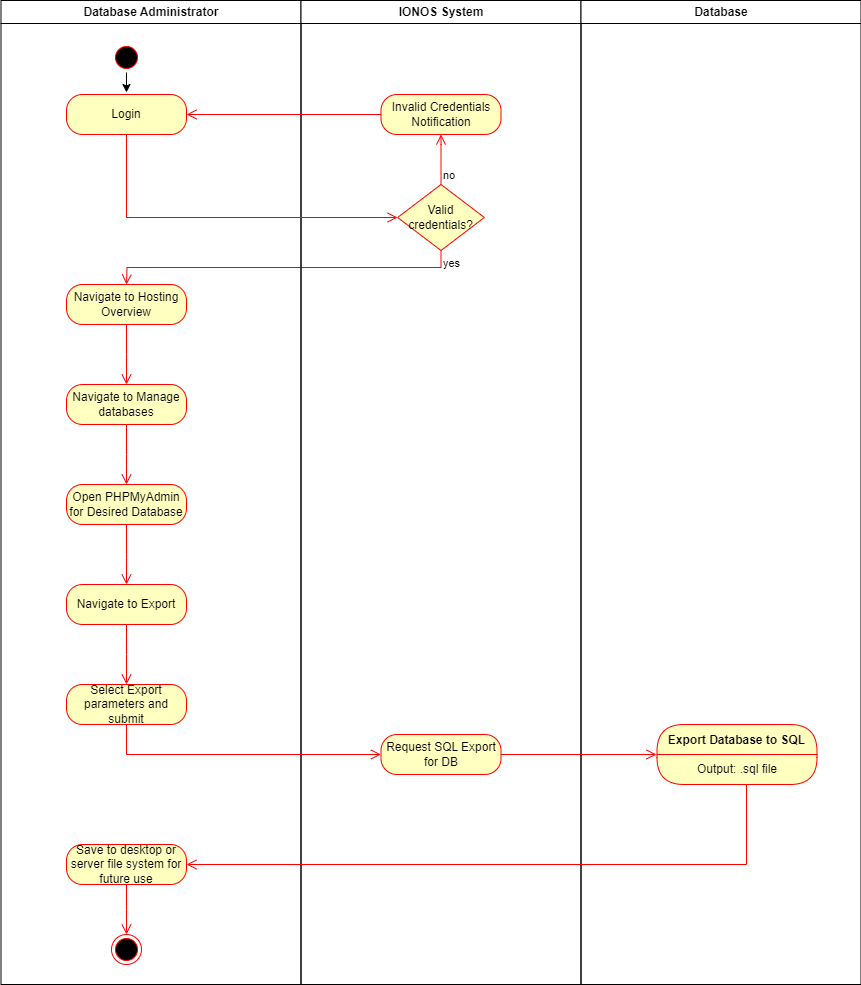


Figure 12: Database Backup Activity Diagram

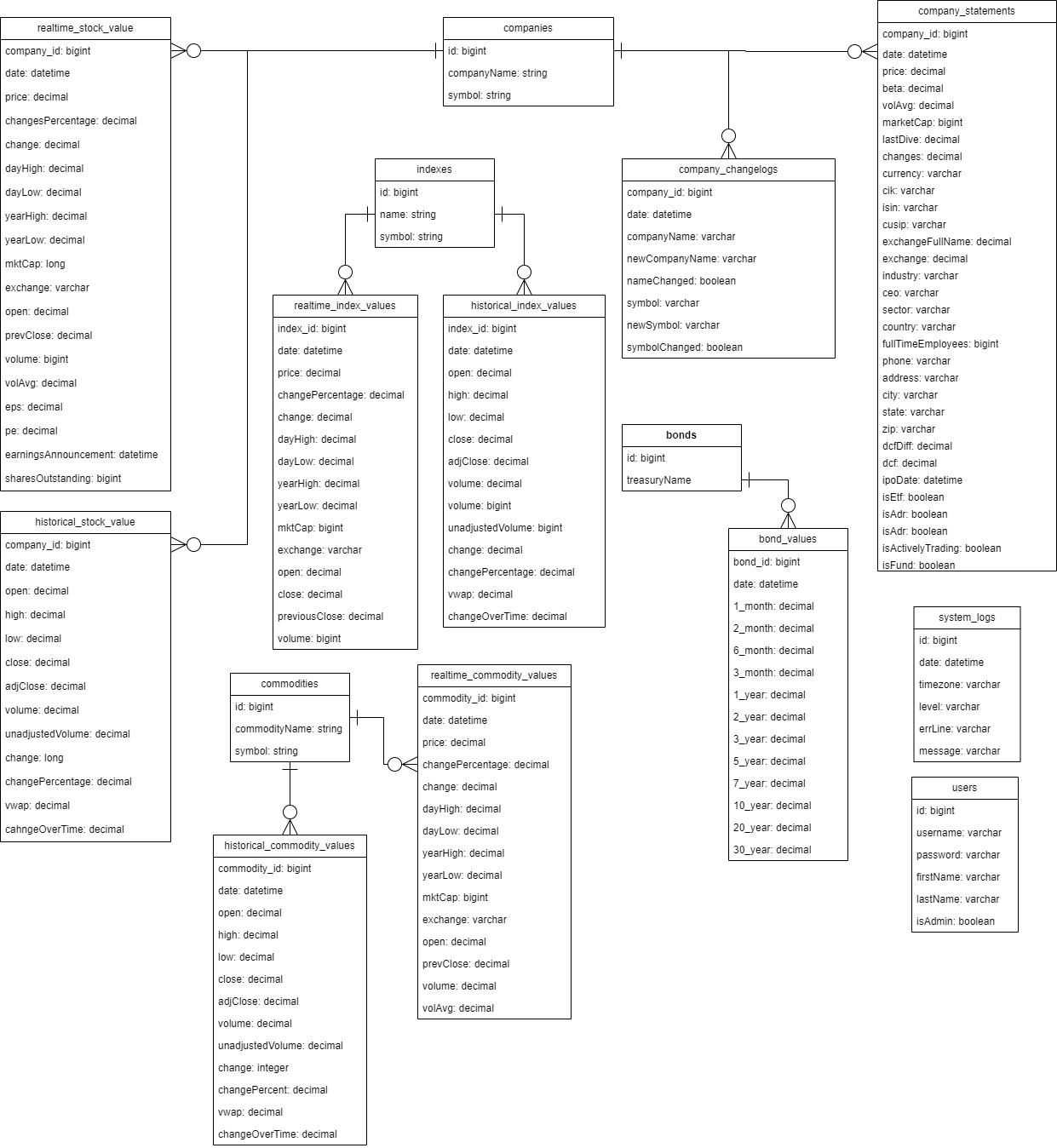


Figure 13: OLTP Database Entity Relationship diagram

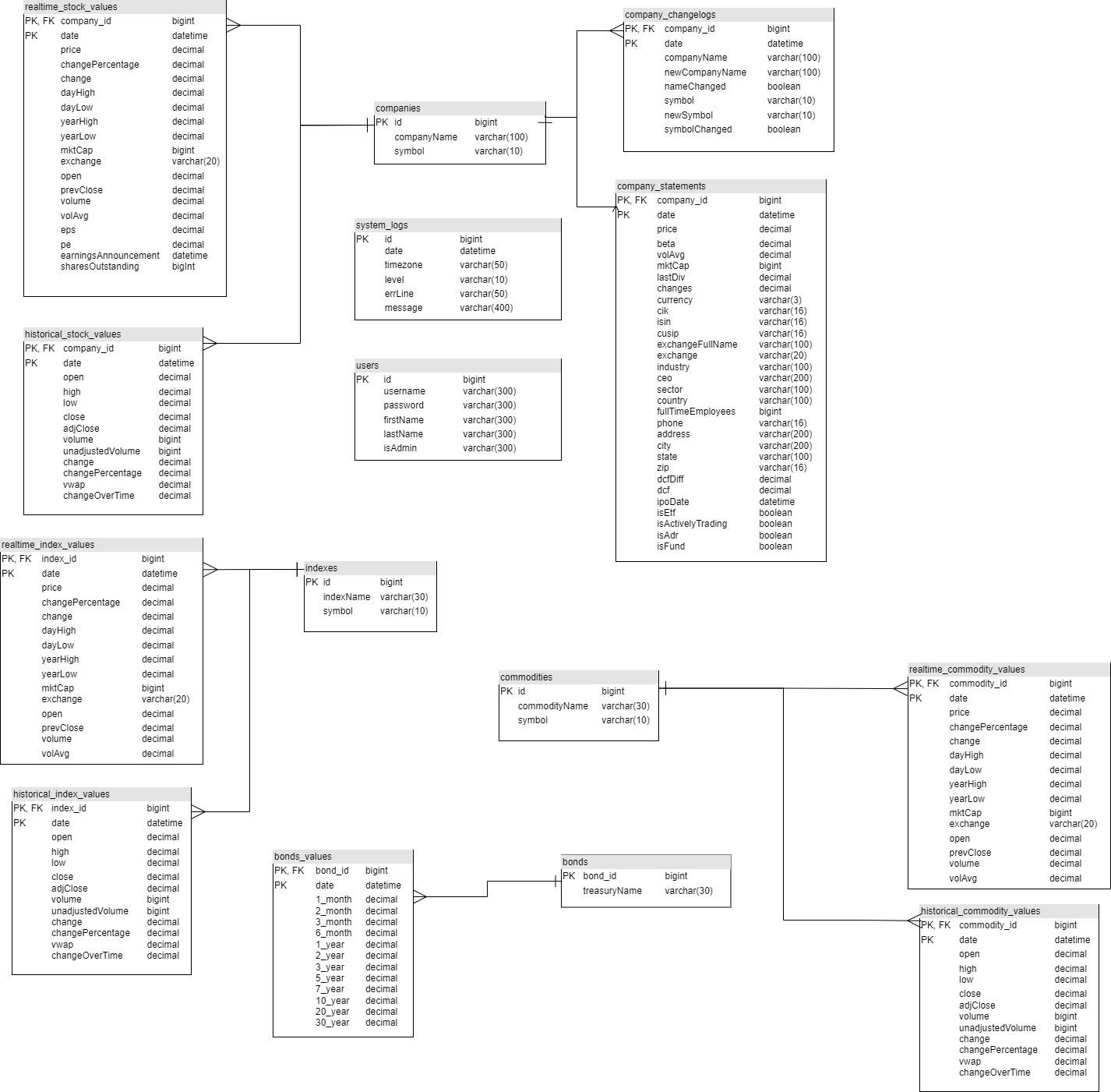


Figure 14: OLTP Database Server model diagram, demonstrating the implementation strategy for a single database.

### Processing for Database

The outputted data file from each query process in Data Collection is read by the matching insertion process. The insertion process takes this data and performs redundancy checks, as well as adds any fields that are not returned by the API but should be recorded in the database. The script then makes a connection to the database on the server, and executes the required insert or update statement. When these statements occur, various database triggers are run. Triggers will generate unique IDs when necessary, and update symbols or names when necessary. Data will be stored in the database for 3 years before it is considered obsolete and deleted. . Backup procedures will ensure the database can be recovered in the event of failure.

Processing will assess any symbol changes first, writing an entry to the Changelogs table for each symbol changed. Following each entry in Changelogs, the associated company symbol and/or name will be updated in the Companies table. In the preferred implementation, the remaining processes can be conducted concurrently for the sake of efficiency. In the IONOS hosted environment, the remaining operations will be conducted sequentially.

### Database Interface Description

Input:

* Query output data: JSON format

Output:

* Populated Tables in Database
* Data exportation (various formats: csv, sql, etc)

## Design Alternative - Data Warehouse

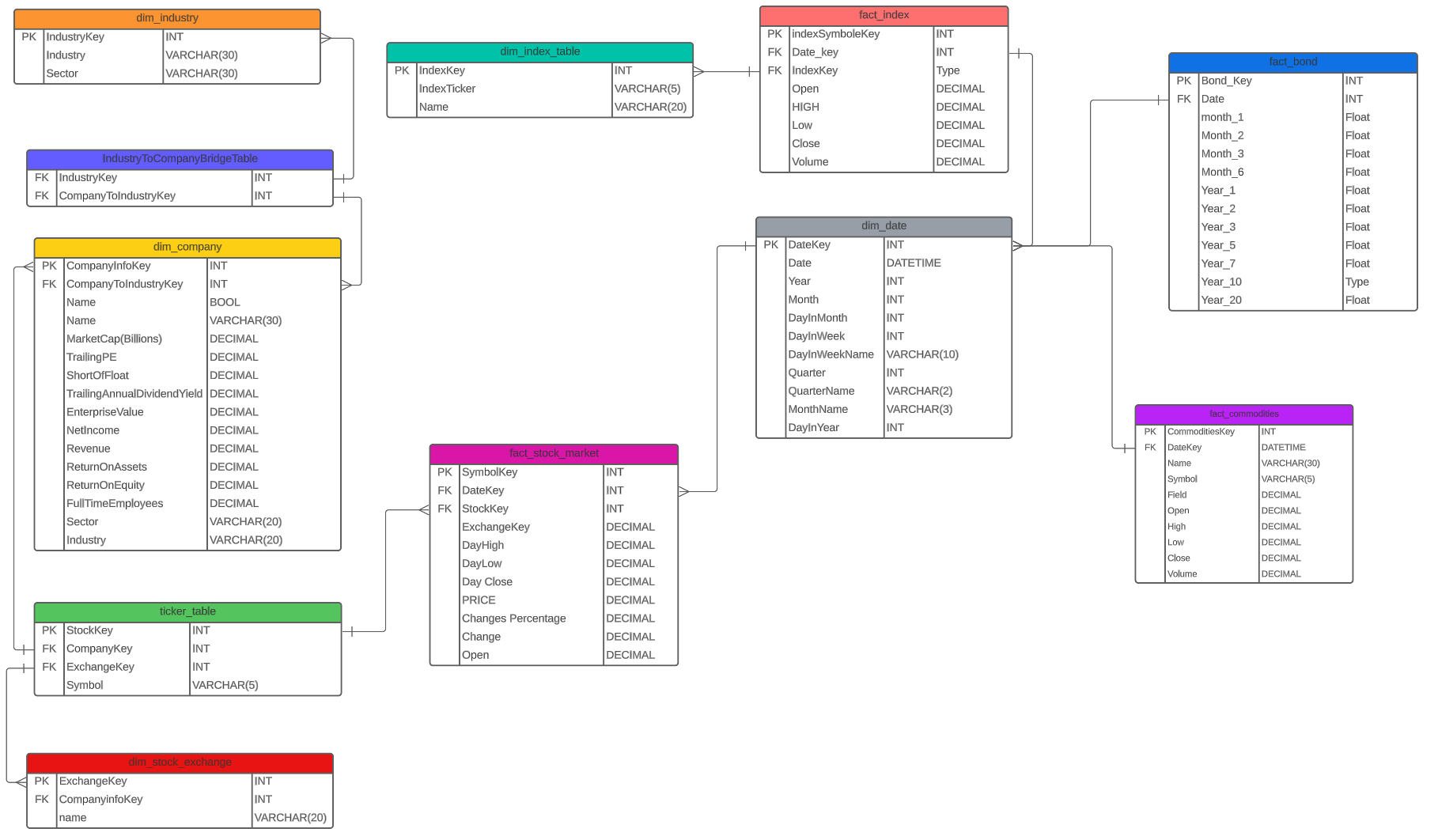


Figure 15: Data Warehouse constellation schema Entity Relationship diagram

### Processing for Warehouse

In initial design for the ATS, a Data Warehouse was considered as a long term storage model separate to the database. This was ultimately shelved during development, primarily due to the end user not requiring long term data storage, meaning the database storage would be sufficient. Though not currently required, a future version of the system may call for data warehousing. If implemented, the process will first establish a connection to the Database, and begin pulling and transforming data. An error logging process will generate log files for any failures or errors, as necessary. Data provided to the Warehouse will be validated before insertion and/or update operations. Cleanup operations will ensure any temporary files generated during this process are removed.

### Warehouse Interface Description

Input:

* Data Files with Database data: *CSV*

Output:

* Populated Warehouse Tables

# Data Architecture

The database schema for this project has been divided into many separate tables. Each of these is responsible for holding different market data. [R1] and Server Model Diagram [Figure 14] for a detailed breakdown of the database structure.

**Database tables**

*COMPANIES*

* *Lookup table for real time stock values, historical stock values, company statements, and company changelogs*
* *Contains a unique id, company name, and symbol*

*REALTIME\_STOCK\_VALUES*

* *Table containing stock data that was collected on a daily basis*
* *contains all relevant fields returned by API*

*HISTORICAL\_STOCK\_VALUES*

* *Table containing historical stock data*
* *contains all relevant fields returned by API*

*INDEX\_VALUES*

* *Table containing index data*

*BOND\_VALUES*

* *Table containing bonds data*

*COMMODITY VALUES*

* *Table containing commodities data*

*COMPANY\_STATEMENTS*

* *Table containing company information such as sector, ceo, number of employees, etc*

*COMPANY\_CHANGELOGS*

* *Table that*

*COMMODITIES*

* *Lookup table*

*INDEXES*

* *Lookup table*

*BONDS*

* *Lookup table*

# Interface Requirements

## Required Interfaces

### 8.1.1 IONOS

* Cloud hosting service
* ATS system interfaces with IONOS to access SSH, databases, and PHPmyAdmin
* Inputs: SFTP/SSH commands, database queries, storage allocation requests
* Outputs: server responses

### 8.1.2 Financial Modelling Prep API

* Financial data API provider
* ATS system interfaces with the API by making calls to it through data collection queries
* Input: API calls
* Output: API responses
* Format: REST API
* Frequency: Real-time at different intervals as required

### 8.1.3 Machine Learning Frameworks

* Software Libraries
* ATS warehouse interfaces with ML models
* Input: Datasets from data warehouse
* Output: ML model output
* Frequency: Monthly

## External System Dependencies

The ATS system has the following external dependencies:

* IONOS Hosting Provider
* Financial data API provider (Financial Modelling Prep)
* Financial Markets
* Machine learning frameworks

# User Interface

### IONOS

IONOS provides a GUI for accessing and managing the creation of databases, and the SSH shell. This is all located under the “Hosting Overview” page.

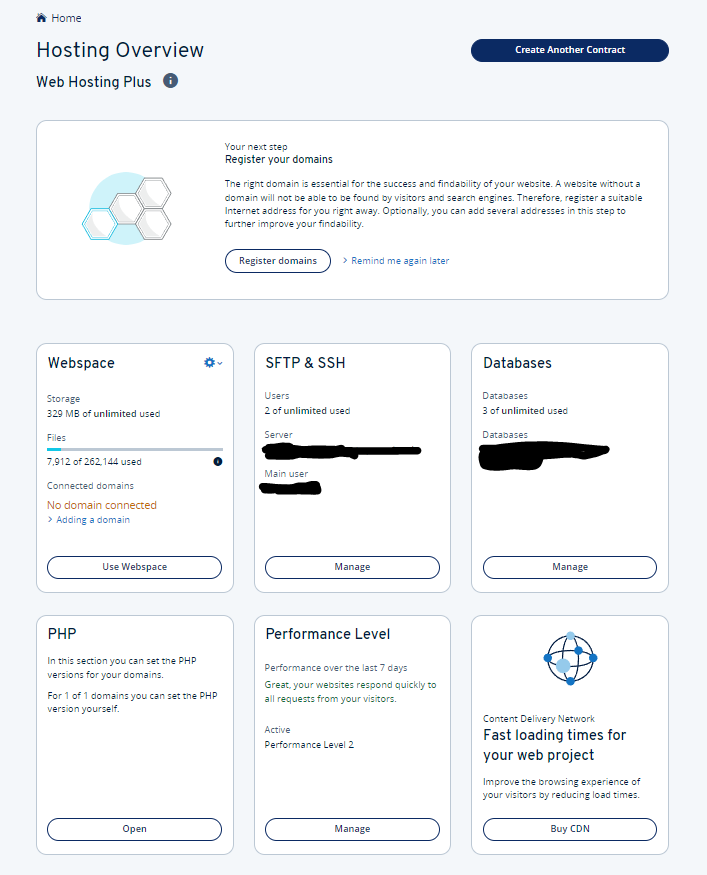


Figure 16: IONOS hosting overview page

### phpMyAdmin

phpMyAdmin is the environment for managing the database. It provides tools to Execute and perform database administrative operations.

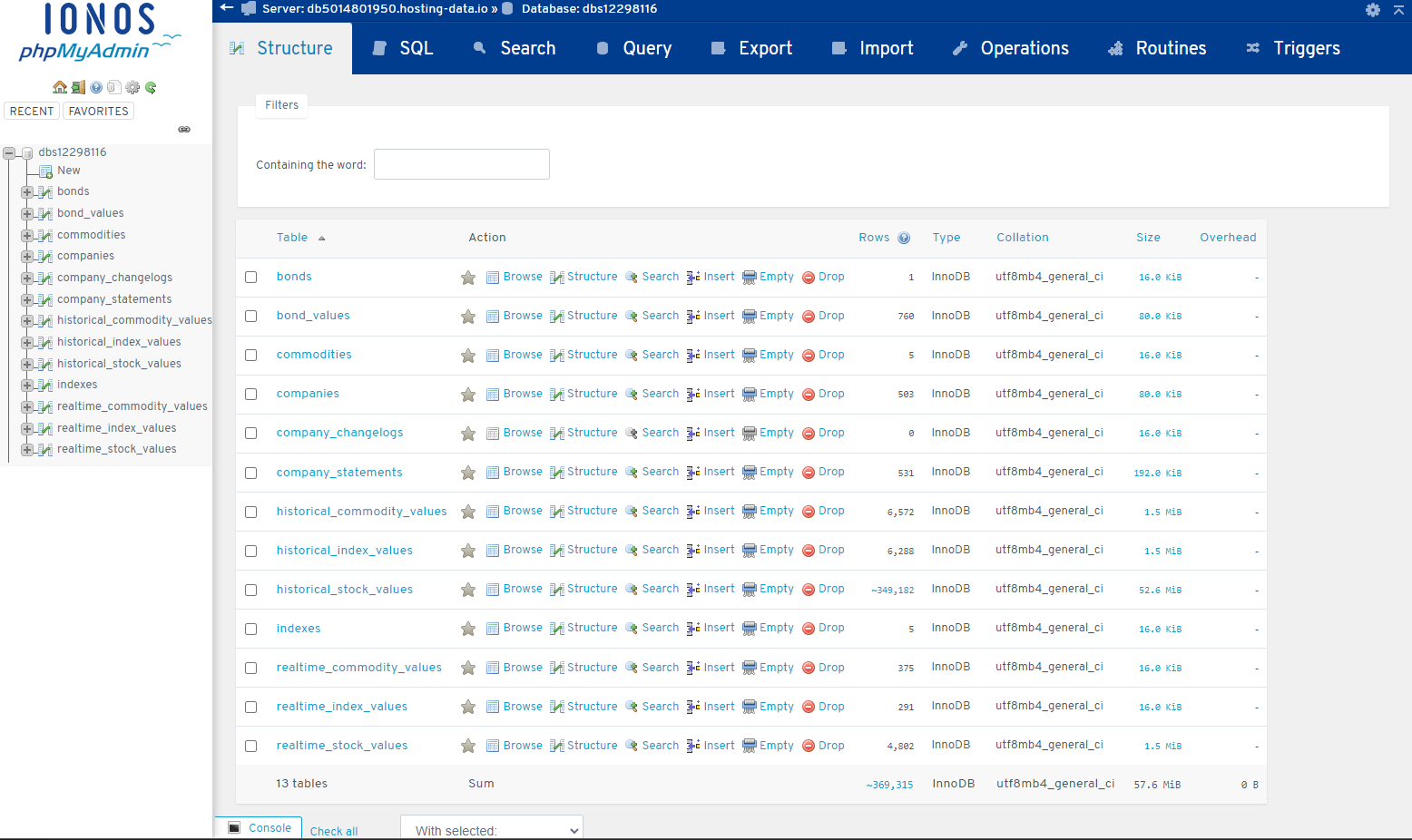


Figure 17: phpMyAdmin interface

### ATS User Interface

ATS User Interface is a web application that allows users to change configuration files, job scheduling and data exportation. It is accessible to admins and regular users such as students. Regular users can only access the Export Data page *(Figure 21)*

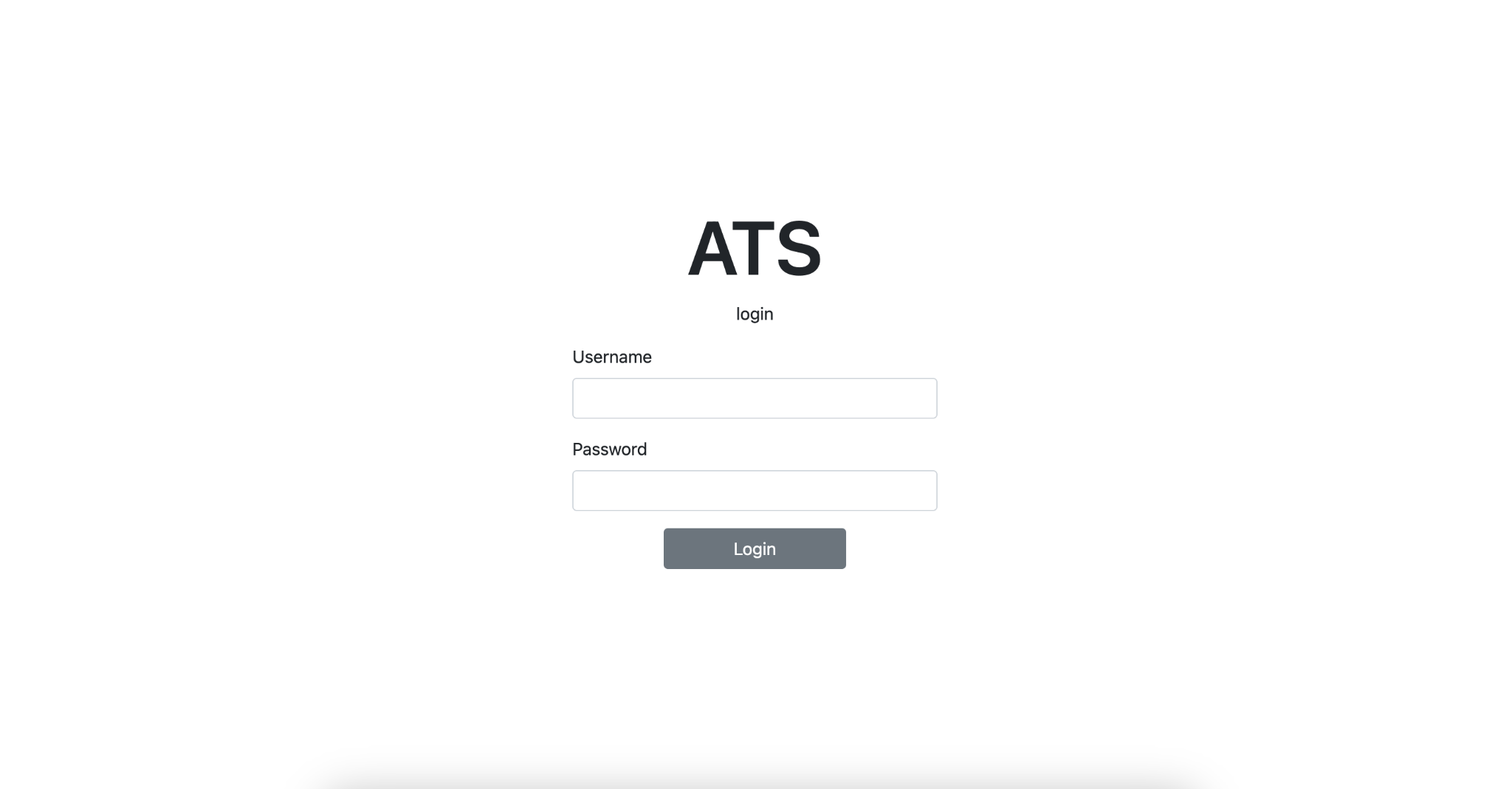
**

Figure 18: ATS Login Page

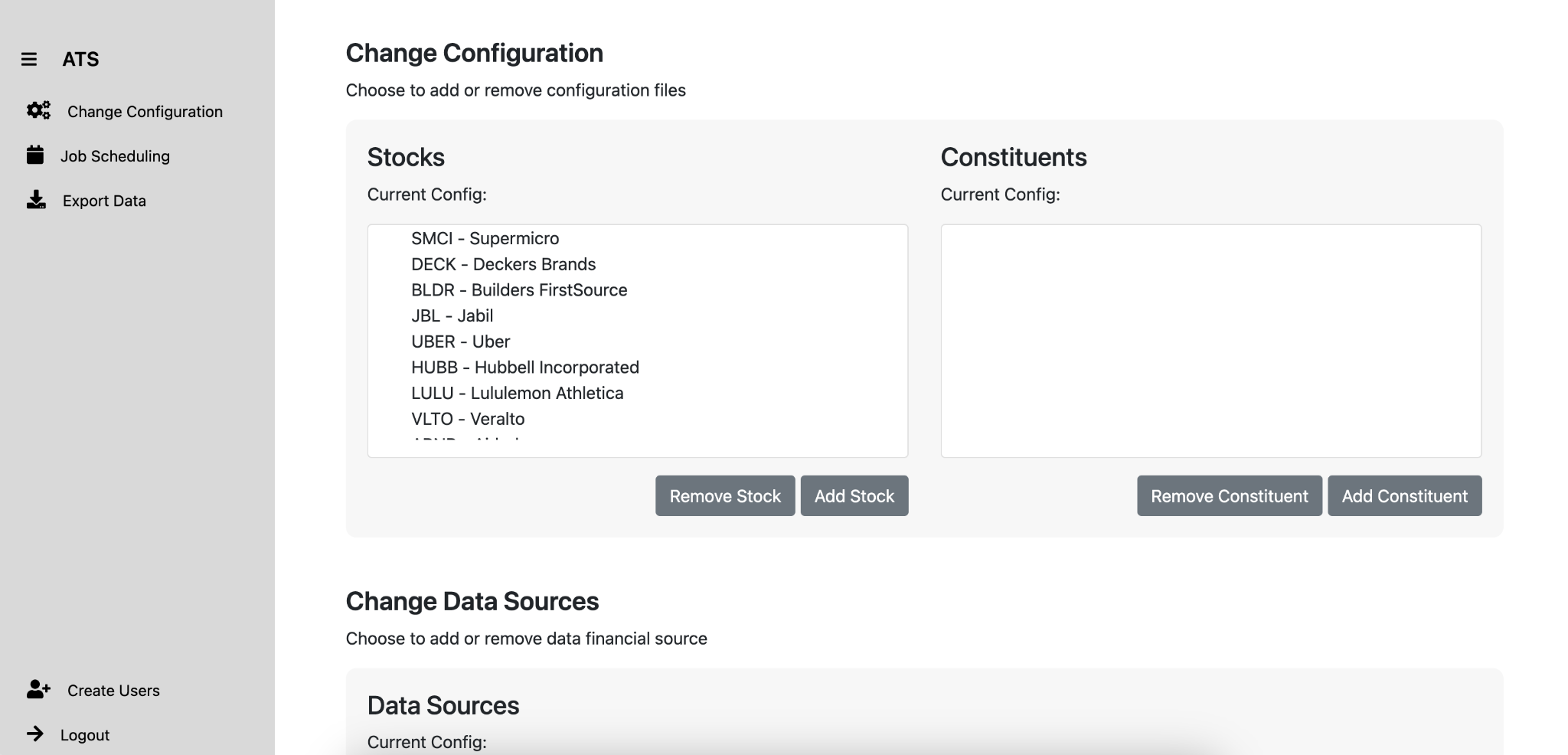


Figure 19: ATS Change Configuration Page

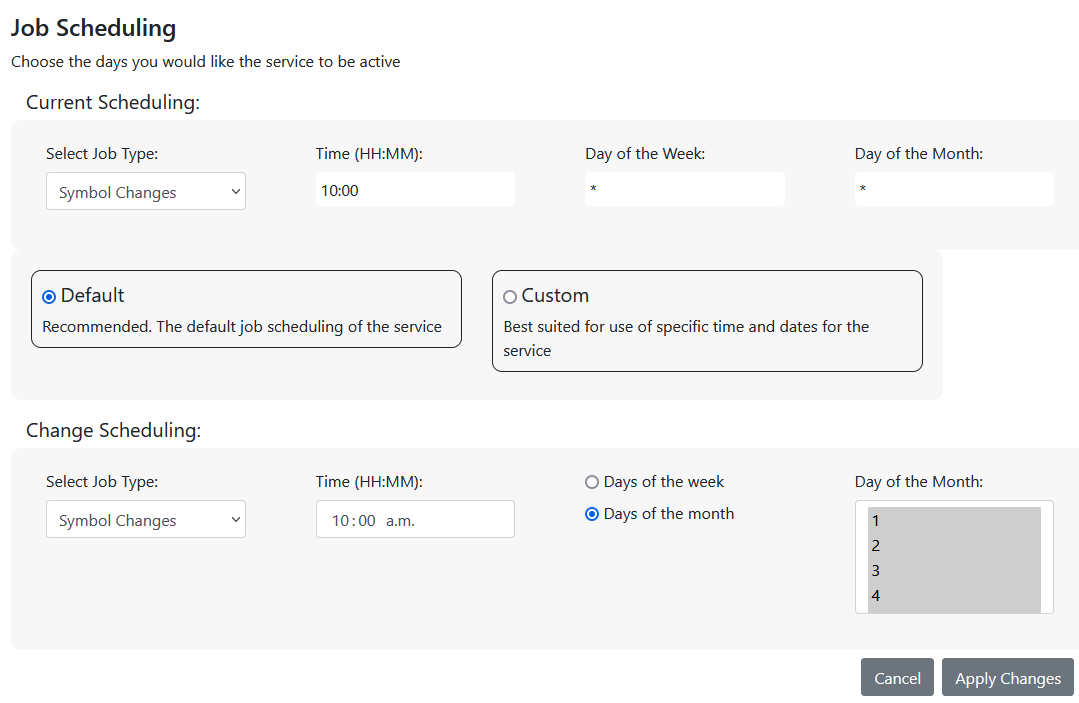


Figure 20: ATS Job Scheduling Page

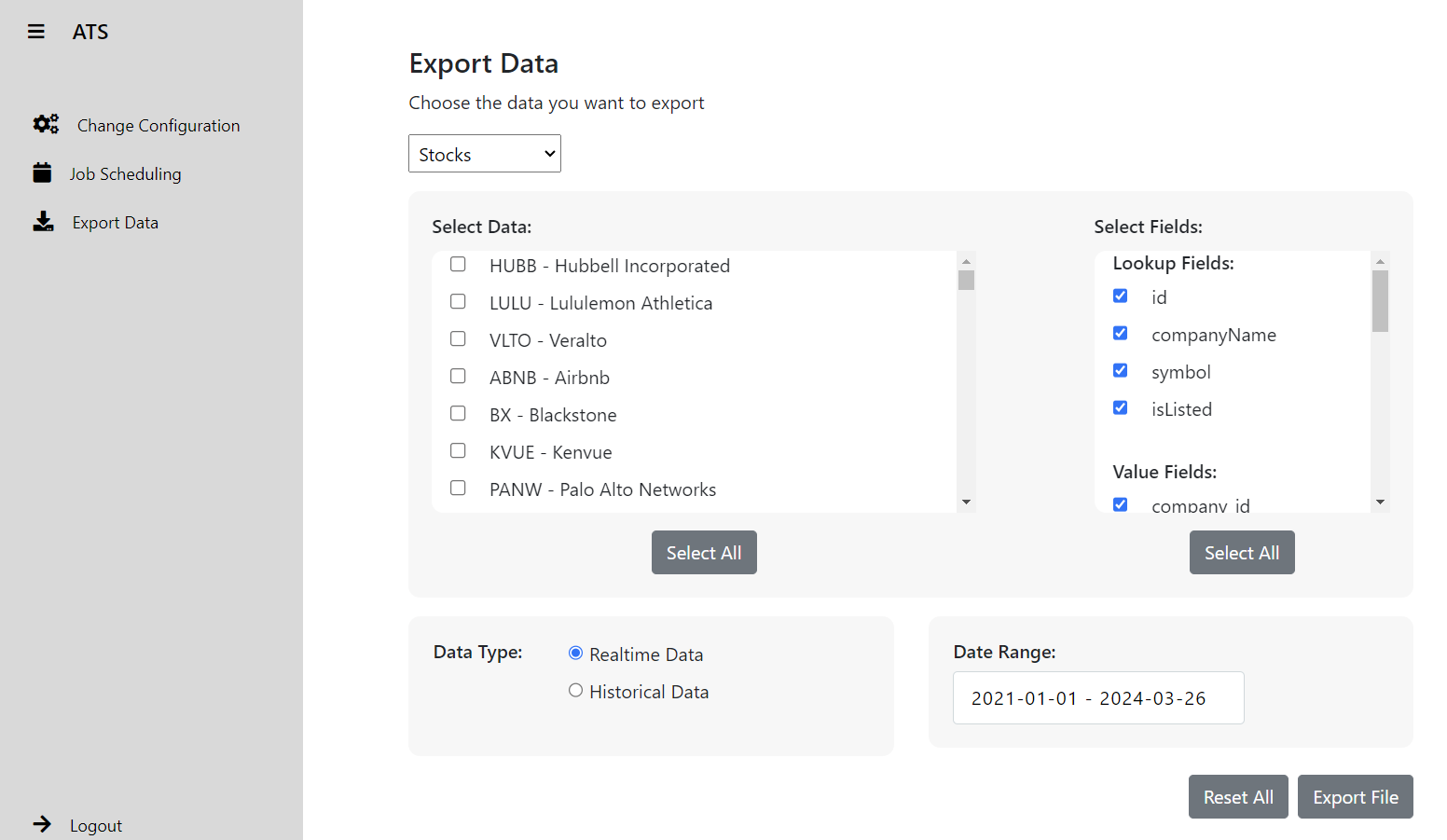


Figure 21: Export Data Page

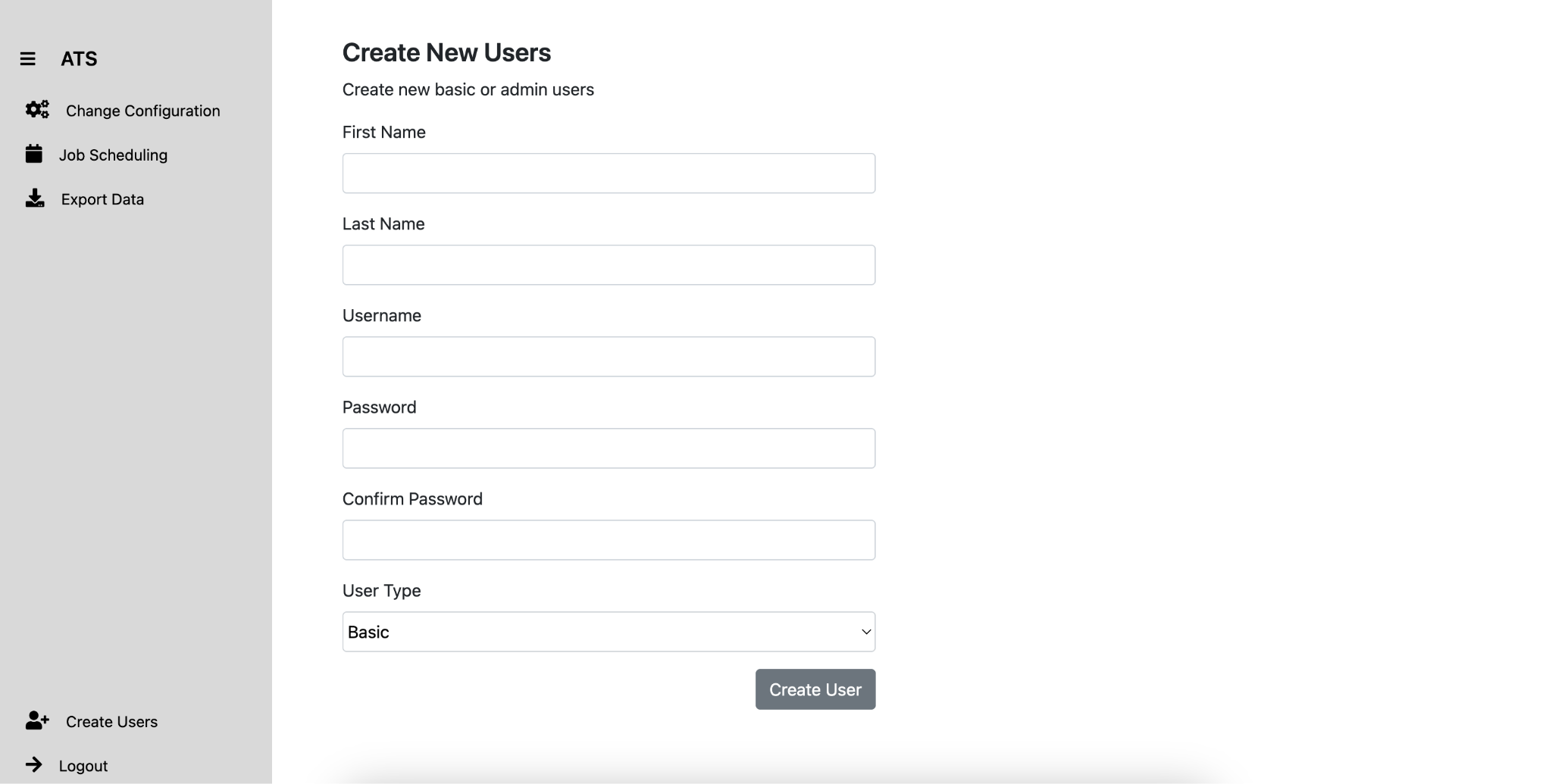
**

Figure 22: Create User Page

# Non-Functional Requirements

* Data is available for retrieval during and after trading hours
* The system will use IONOS to host the system and database
* Data will be processed from collection to insertion within 6 hours maximum
* Current Stock, index, and commodity data will be collected daily at 4:00PM EST
* US Treasury bond data will be collected weekly
* Company data will be collected every month
* 3 years worth of historical data will be stored in the database for stocks, bonds, indexes, commodities, and companies
* OLTP database will be updated daily
* Sensitive data will not be stored anywhere publicly accessible
* The web application should be designed to be accessible to both administrators and regular users.
* The web application should provide a user interface that allows admins to modify configuration files.
* The web application should enable users to export data in CSV format.

**APPENDIX A// REQUIREMENTS TRACEABILITY MATRIX**

*Please refer to the separate ATS Requirements Traceability Matrix (RTM) document [R4]*