ChronoWave
Software Design Specification

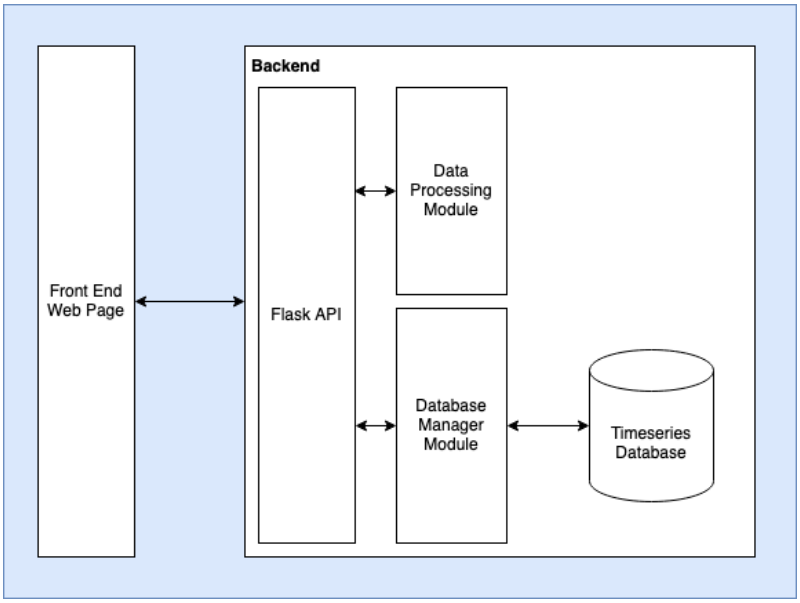
Table of Contents

1. System Overview

The system is a web application that 1.) provides the ability for Contributors to upload time series data sets for sharing with the community, 2.) provides DS/MLEs the ability to search for and download time series datasets, 3.) Allows DS/MLEs to upload forecasts for time series data sets for error metric calculations that get shared with the rest of the users on the application.

The system consists of five core components: 1.) The front end which is written with HTML, CSS, and Javascript along with the Bootstrap front end library, 2.) A flask API that generates dynamic HTML web pages on the fly as users interact with the application, 3.) A Data Analyzer module that calculates error metrics for forecasts using Pandas, 4.) A Database Manager module that acts as the IO system for time series and metadata storage, and 5.) A MongoDB Instance that runs on Mongo Atlas that is used for actual data storage.

2. Software Architecture



1. Components
    a. User Interface/Front End module - This is the pure HTML, CSS, and Javascript template code that acts as the basis of the web page sent to the user. Additional HTML and CSS is generated by the Flask Module in order to load dynamic content into the web pages.
    b. Dynamic Content/Flask Module - This module acts as the controller of the web application, taking commands from the Front End Module and calling the appropriate back end modules(either Database Manager or Data Analyzer) to get dynamic content. The Flask Module then generates dynamic web pages using the content and sends that back to the user. This is written in the form of Flask routes.
    c. Database Manager Module - This module implements all the database application code for Input/Output of time series data and metadata from the system. It's interface is defined abstractly so that if a different database is used in the future the only piece of software that needs to be changed is the Database Manager Module as the interface for using it will remain the same. The current Database Manager Module works using MongoDB and talking to a MongoDB instance. The Database Manager Module is a single python 3 class.
    d. Data Analyzer Module - This module implements all data analysis functionality including generating error metrics between forecasts and testing data, as well as generating plots comparing forecasts and testing data. This is in the form of a python 3 class.
2. Interactions and Rationale
    The software architecture was designed in a hierarchical way that ensures monodirectional dependencies. That is, the Flask Module talks to the Database

Manager Module, but the Database Manager Module has no knowledge of the Flask app. Then the Database Manager Module talks to the MongoDB instance. The Flask Module also talks to the Data Analyzer Module, but the Data Analyzer Module has no knowledge of the Flask Module. All communication between the Database Manager and the Data Analyzer happens via the Flask Module(that is, the Flask Module asks the Data Analyzer to generate metrics, then the Flask Module stores those metrics via the Database Manager). This hierarchical, monodirectional architecture limits dependencies between components and ensures limited coupling of functionality, which increases future maintainability.

3. Software Modules

3.1. User Interface/Front End

3.1.1. Module Role

The role of this module is to consolidate the HTML and CSS templates that are used as the baseline web pages in the creation of the web pages that are created by the Flask Module and sent to the user. This is a very simple module as they are just HTML/CSS template files, so there isn't much software architecture involved.

3.1.2. Module Connections

This module connects to the Flask Module as the Flask Module generates the dynamic web pages using the template web pages in this module.

3.1.3. Models(N/A)

Simple HTML templates. Not dynamic.

3.1.4. Design Rationale

The design rationale came from necessity as Flask requires the existence of web page templates to act as the baseline for dynamic web pages.

3.2. Flask Module

3.2.1. Module Role

The role of this module is to act as the middle man component between the two back end components(Database Manager and Data Analyzer) and the Front End Module. This generates dynamic web page content by making appropriate calls to the Database Manager and Data Analyzer, and then loading that dynamic into custom web pages that are then returned to the users.

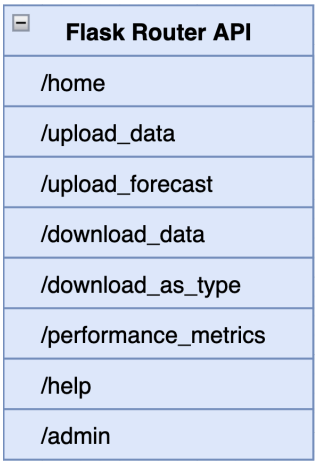
3.2.2. Module Connections

This module is the central component as it talks to every other component in our software architecture. It talks to the Database Manager Module in order to fetch time series data that users request as well as to store time series data that users upload. It talks to the Data Analyzer to analyze forecasts that user have

uploaded, and finally it talks to the front end by generating web pages for the user.

3.2.3. Models

3.2.3.1. Flask Module API Diagram



3.2.4. Design Rationale

We decided on the master controller component in our design as it allowed for easy consolidation of functionality between different software components. By having the Flask Module be aware of the other modules but the other modules not aware of each other, we minimized the number of dependencies that most code modules have, increasing maintainability. In addition to this, the choice of using Flask allowed us to easily generate dynamic web pages.

3.3. Database Manager Module

3.3.1. Module Role

The role of this module is to consolidate all database application code in a single python 3 class that follows a standard abstract interface. The Database Manager class exposes generic functions like *store_timeseries_set()* and *store_forecast()* that appear storage agnostics(that is, the user of the Database Manager needs no knowledge of what database is actually being used). This module is responsible for all storage and retrieval of data in the web application.

3.3.2. Module Connections

This module is connected to a MongoDB database instance as this is what it uses for data storage. This connection is established via the pymongo python library. In addition, the module is used by the Flask Module, so there is an omnidirectional dependency there from the Flask Module to the Database Manager Module.

3.3.3. Models

3.3.3.1. DatabaseManager Class Interface Diagram

3.3.4. Design Rationale

We decided to consolidate all of the database application code into a single class with an abstract interface as this ensured that if we ever needed to change the database used down the line, we would only have to rewrite code in this one location. This ensured high code maintainability as it decoupled all database application code from the rest of the web app.

3.4. Data Analyzer Module

3.4.1. Module Role

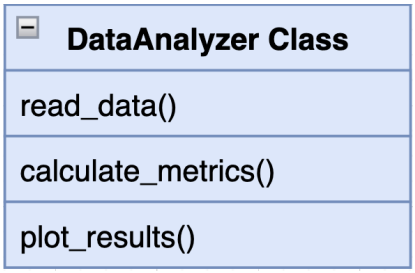
The role of this module is to perform all necessary error metric calculations for a time series forecast in comparison to a time series test data set. This is consolidated into it's own python 3 class with a predefined abstract interface. The module does metric analysis using Pandas data frames and sklearn metric calculation functions.

3.4.2. Module Connections

The module is used by the Flask Module, so there is an omnidirectional dependency there from the Flask Module to the Data Analyzer Module. The Flask Module will retrieve test data from the Database Manager and send the test data set along with the forecast data set to the Data Analyzer class for analysis.

3.4.3. Models

3.4.3.1. DataAnalyzer Class Interface Diagram

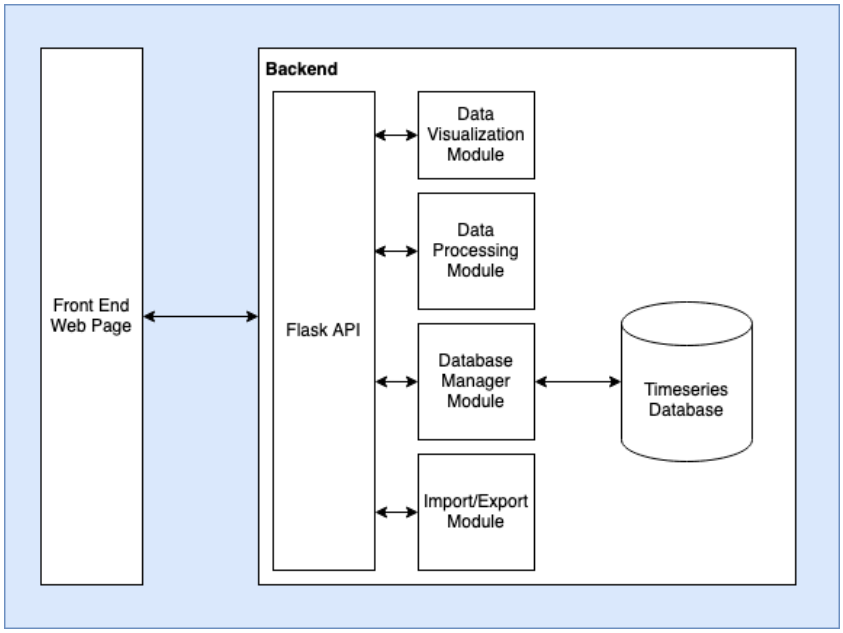


3.4.4. Design Rationale

The Data Analyzer Module is consolidated into it's own python 3 class with a predefined abstract interface just like the Database Manager Module for similar reasons to why we did it for the Database Manager: to increase code maintainability and decouple analysis functionality from the rest of the application.

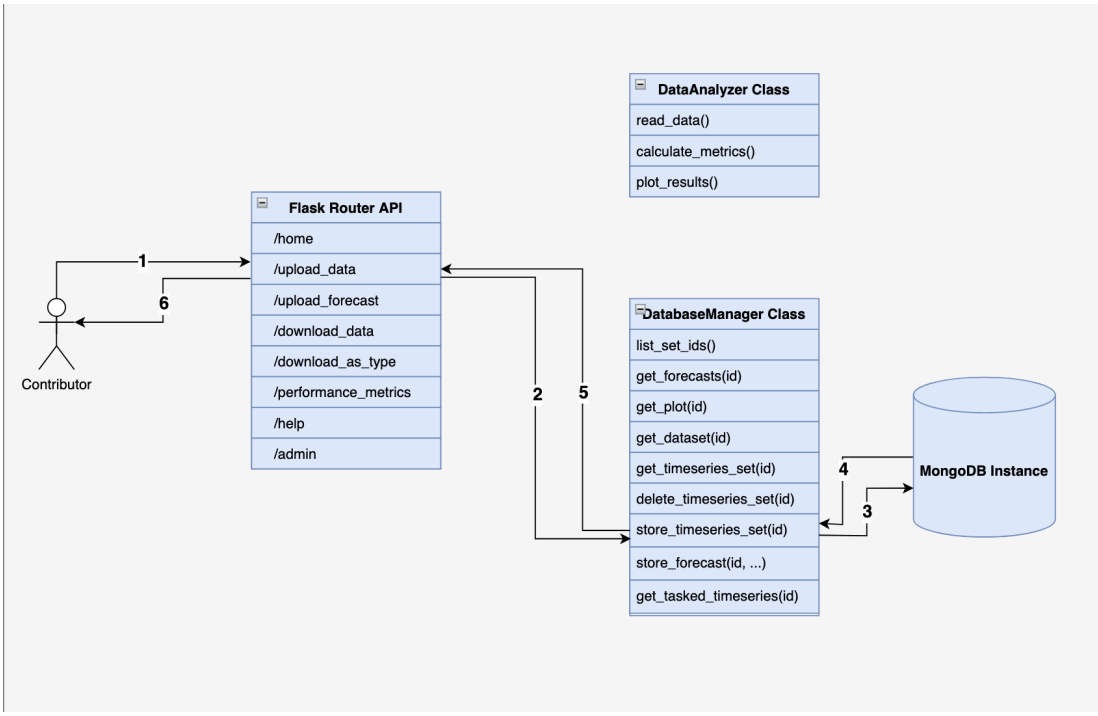
3.5. Alternative Designs

Originally, we had two additional software modules: The Input/Output Module and the Data Visualization Module. We realized early on in development that there was a lot of overlap in data needed between the Input/Output module and the Database Manager module, which indicated the functionality was highly related and should be consolidated into a single class. The same went for the Data Analyzer and the Data Visualizer modules.

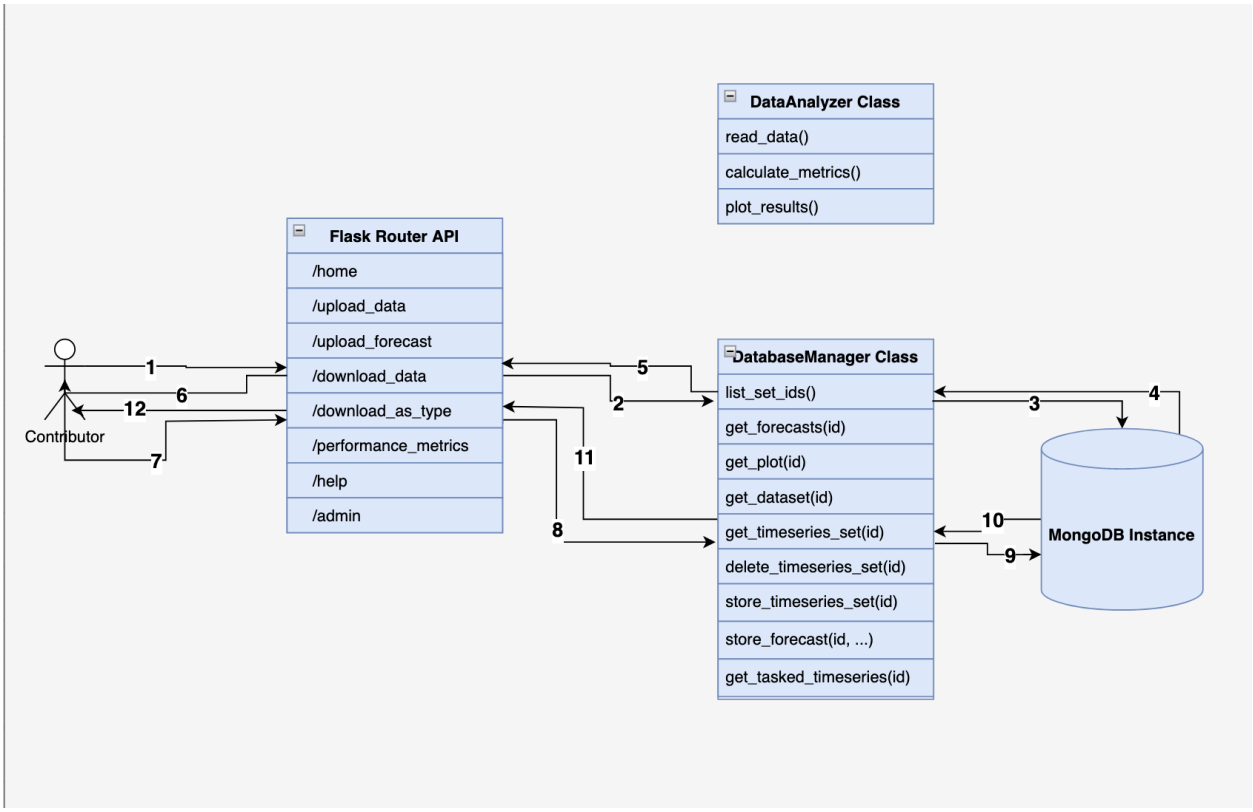


4. Dynamic Models of Operational Scenarios (Use Cases)

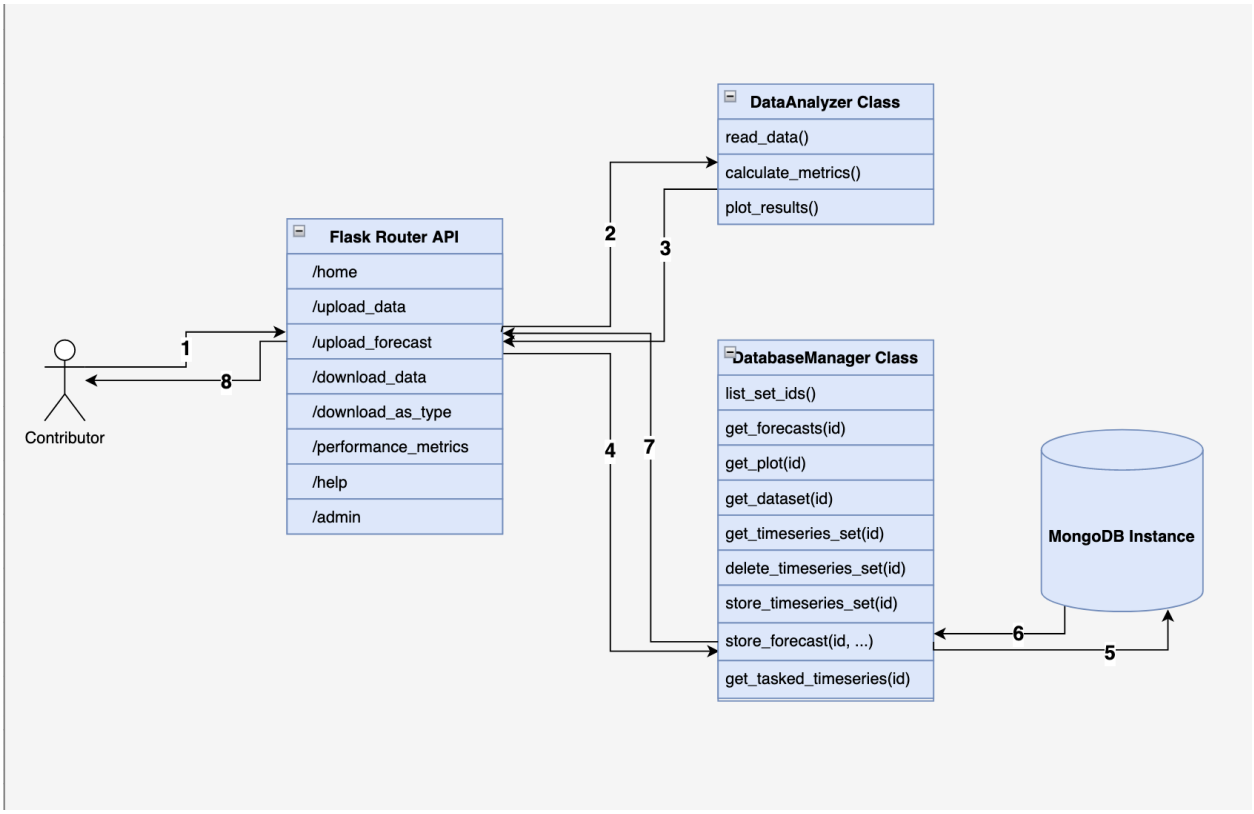
4.1. Contributor Upload Time Series Data Sequence Diagram



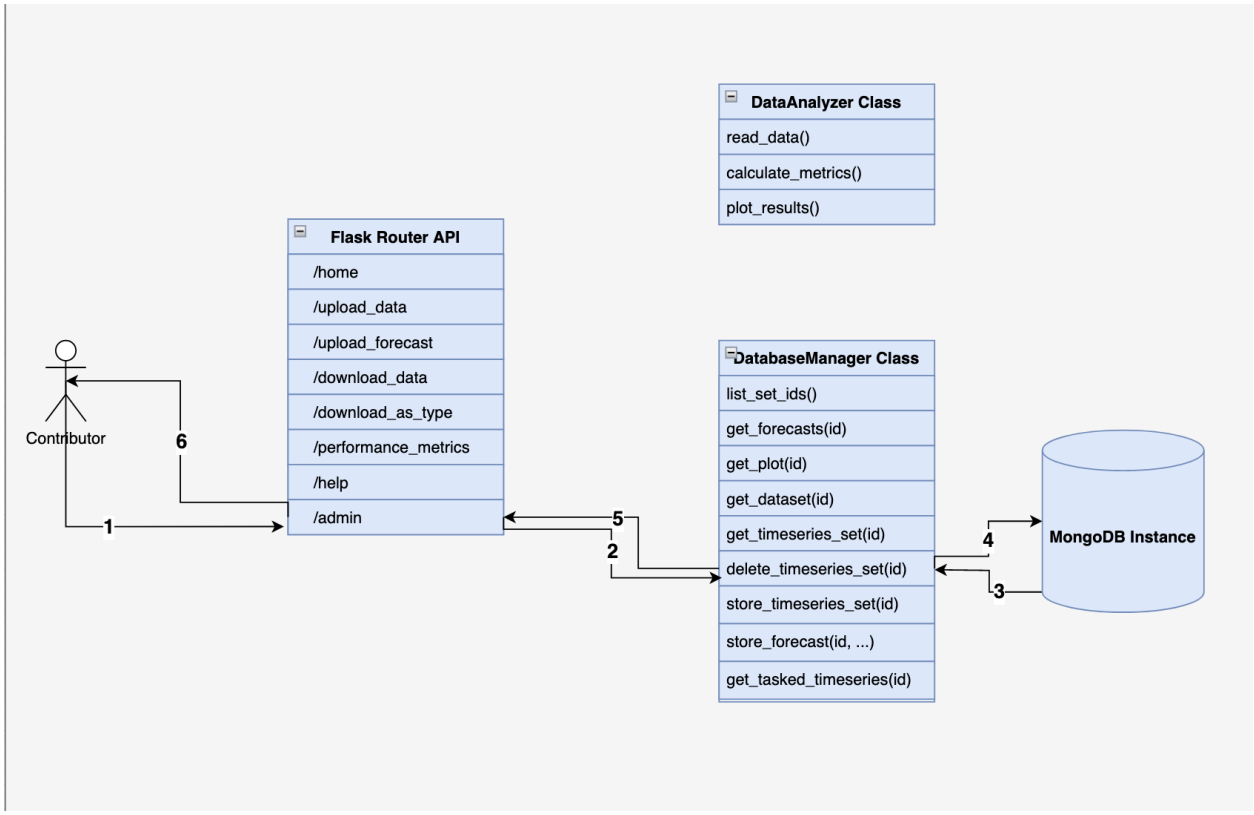
4.2. DS/MLE Download Time Series Data Sequence Diagram



4.3. DS/MLE Upload Forecast Data Sequence Diagram



4.4. Admin Delete Time Series Set Sequence Diagram



5. References

Class Diagram. In *Wikipedia*, n.d. https://en.wikipedia.org/wiki/Class_diagram.

Sequence Diagram. In *Wikipedia*, n.d. https://en.wikipedia.org/wiki/Sequence_diagram.

UML. In *Wikipedia,* n.d. https://en.wikipedia.org/wiki/Unified_Modeling_Language

6. Acknowledgements

Contributors: Aleksandr Stevens, Isaac Perkins, Geoffry Brendel, Cynthia Meneses