FTL (Forecasting Time Series Library)
Software Design Specification

Julian Albert, Nathaniel Mason, Hannah Sagalyn, Carlos Villarreal-Elizondo
Team-5
Github link: https://github.com/n-mason/CS422_Team5_Project1

1. SDS revision history

The revision history of this SDS document:

Date	Name	Description
5/4/23	Hannah	Created and Formatted document with table of contents, title, subsections etc.
5/4/23	Julian	Designed diagrams
5/5/23	Hannah	Wrote section 2 System Overview
5/5/23	Hannah	Wrote section 3.1 (frontend) and 3.3
5/5/23	Hannah	Wrote section 4.1.1, 4.1.2, 4.1.5, 4.1.6. 4.2.1, 4.2.2, 4.3.1, 4.4.1, part of 4.4.2, 4.7.1, 4.5.1, 4.5.2, 4.5.5, 4.5.6
5/5/23	Julian	Completed diagrams
5/5/23	Julian	Edited 4.6

Date	Name	Description
5/5/23	Nate	Added backend and database information
5/5/23	Carlos	Began 4.6
5/5/23	Carlos	Added links to reference page
5/5/23	Carlos	Completed most of 4.6 except models, added references, changed acknowledgments
5/5/23	Nate	Wrote 3.2 and 3.1 (backend)
5/5/23	Nate	Wrote 4.3.2,4.3.5,4.3.6
5/5/23	Nate	Wrote 4.7.2, 4.7.3, 4.7.6

2. System Overview

Time Series Forecasting is a common problem in Machine Learning and Data science. Forecasting is an incredibly important tool used to predict future outcomes across all domains of activity. The results of these predictions are used by clients to inform and influence decisions about the future that can have an enormous impact on thousands of people. It is then evident that the accuracy of such forecasts is crucial to the success of any Machine Learning engineer. Our system allows Machine Learning engineers to train and test their forecasting methods by providing a comprehensive error analysis between their forecasted solution and real data. In our system, Contributors will be able to specify a forecasting task and upload the relevant Time Series data split into a visible Training Set and a separate private Test Set. Machine Learning Engineers will then be able to access a repository of Training Data with their respective forecasting tasks to train their forecasting methods. Upon completion, the MLE may upload their solution to the system to receive a comprehensive error analysis between their forecasted data set and the actual original Test Set provided by the Contributor. They will also be able to compare the accuracy of their forecast against all previously uploaded solutions for their chosen forecasting task.

Our system consists of seven main components. The Home Page, Contributor Upload page, MLE download page, MLE upload page, Error Analysis Display page, the All Solutions Display page, and the Error Table. The Home Page serves as a starting point for Users to specify their role or view all the Error Tables currently a part of the application. The Contributor Upload page is where Contributors may upload Training Data, Test Data, specify a forecasting task and fill out metadata for each file to be stored in the database. The MLE Download page is where all the training sets in the database are displayed with the relevant forecasting task data and available for download by the MLE. The MLE Upload page is where an MLE submits their name and solution to the database to receive an error analysis. The Error Display page is generated after

the MLE upload. It is where the error algorithms are run on the uploaded solution data, and stored in the database where they are paired with the specified forecasting task. The Error Table is then updated with the new solution's error analysis from the database and displays it for the user along with a graph that compares the test data and the forecasting solution data. Finally the All Solution's Display page fetches all the analyses for every forecasting task from the database and displays them in Error Table's where they can be manually sorted by desired metric.

3. Software Architecture

3.1 Components and their functionality

Frontend:

HomePage

- Allows Users to specify their role as a MLE or Contributor.

Contributor Upload

- Contributors can upload two data files, a Training Set and a Test Set
- Contributor's can enter all meta data for both their Training Set and Test Set data files. Contributors can also specify their forecasting task here

MLE Download Page

- MLE's can view all Training Sets previously uploaded by contributors and their respective forecasting tasks.
- MLE's can download their preferred Training data Sets

MLE Upload page

- Where Machine Learning engineers can upload their forecasted solution and username for error analysis.

Error Analysis Display page

- Displays a graph comparing the test set data and the MLE forecast data
- Error Table
    - Displays an interactive error table for that specific forecasting task with the error analysis for all previously uploaded solutions generated by other MLE users
    - The Table can be sorted by User, Parameter, MAE, MAPE, SMAPE, MSE, RMSE, r-coefficient?

All Solutions Display page

- Displays all the error tables of MLE solution analyses for each forecasting task/training set

Backend:

Python Flask File

- Depending on what route the user goes to on their browser, Flask will serve the appropriate HTML page with the corresponding content. File will also store/pull data from Firestore database as needed, depending on what action the user is doing on the web app.

Python Error Calculations File

- Given Pandas dataframes (forecast df and testset df) and target variable(s), will perform error calculations, and return dictionary with the error results and graph corresponding to the MLE solution

Firestore Database

- Contributor Uploads
    - Pair id
    - Test set
        - Test Set Data
            - Array of dictionaries, each dictionary corresponds to one row of data in the contributed file
        - Test Set Metadata
            - Dictionary containing training set form metadata
    - Training Set
        - Training Set Data
            - Array of dictionaries, each dictionary corresponds to one row of data in the contributed file
        - Training Set Metadata
            - Dictionary containing test set form metadata
- MLE Solution Uploads
    - MLE Solution Data
        - Array of dictionaries, each dictionary corresponds to one row of data in the solution file
    - MLE Solution Metadata
        - Dictionary containing MLE solution form metadata
    - Error results
        - Dictionary containing the error metrics (MAE, MAPE, MSE, RMSE, SMAPE, r-Value) for that MLE
    - MLE Solution File Pair Id
        - The pair id that is defined in the solution file header (this is how the corresponding test set is retrieved for comparison)

3.2 Module Interaction

In this system, all the modules are important to the functionality of the web app. The front end pages are necessary so that based on what role a user is, their needs can be met. If the user is at the home page, they can either click the contributor button or MLE button and they will be brought to the next corresponding page. These two pages, the contributor upload page and

MLE download page both interact with the Firestore database. Once a contributor has filled out the contributor upload page and has submitted their data, this gets sent to the FIrestore database using our back end Flask Python file. The MLE download page also is linked to the Firestore database, because training sets and their data are pulled from the database and formatted by our Flask Python file, then sent to the MLE download page for the user to view. The other pages in the web app like the error graph page and page that displays all MLE solutions are display pages. These pages also interact with our Flask Python file, and display results for the user to see. The Flask Python file serves as the link between the front end pages and the Firestore database, and also calls any other functions defined in other Python files, like the error calculations Python file for example.

3.3 Rationale for Architecture

To complete the project we needed our application to be able to accept and clean uploaded data, display data to be downloaded, generate and display an error analysis on uploaded data, and sort the different error analyses.

Our architecture was designed with the plan to create a distinct software module for each required functionality. This structure was intended to increase the independence amongst the application's modules for easy integration and debugging. Thus, we knew we needed to have an upload module for each user type, a download module for MLE users, and a module to generate and display the error analysis with sorting capabilities, and a homepage for user accessibility.

We later added the All Solution's Display page module to display all the tables generated by our application for public access. This way we had some way for Contributors and MLE's to view the uploaded solutions at any time.

We kept the modules separated by user type and functionality for easy frontend integration. This way we could design each module independently and then simply link them with html references in the frontend integration.

4. Software Modules

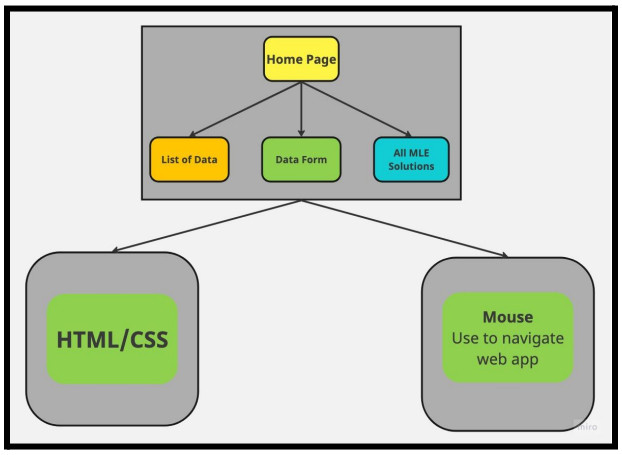
4.1 Home Page

4.1.1 *Role and primary function*

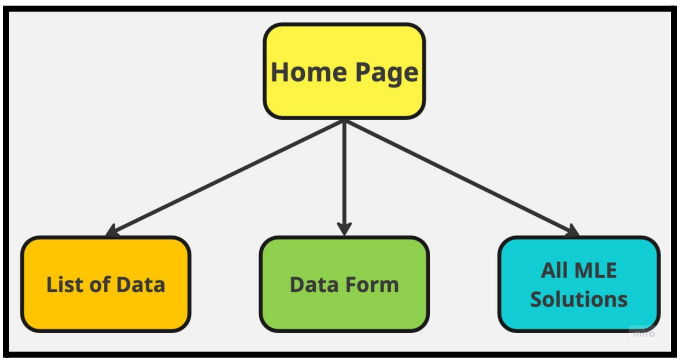
The home page describes the system application and directs the user to choose whether they wish to use the application as a Contributor or a Machine Learning Engineer. It also contains a navigation bar that allows a user to view other parts of the application without having to upload or download their own data.

4.1.2 *Interface specification*

The home page represents the starting point for all applications of the web site. It contains buttons linked to the beginning of each type of user path. Primarily, it contains an MLE button linked to the MLE Download page where an MLE views the available data and a Contributor button linked to the Contributor Upload page where a Contributor may upload their own data. The navigation bar at the top of the page contains alternative links to the MLE Download page and Contributor Upload page in addition to a link to the All Solutions Display page in case the user only wishes to view previously generated analyses. The primary purpose of this module is to direct users to the various applications of the system.



4.1.3 *Dynamic model*



4.1.4 *Static model*

4.1.5 *Design rationale*

The home page was designed to make the user experience as simple as possible. As the very first page a user sees, we wanted to make the application of the site easy to understand by centering the two main user buttons with interactive tooltips to further explain each choice. Rather than make every aspect/page of our application accessible on the home page, we chose to link it to three other pages to increase the simplicity and clarity. Thus the homepage may only lead a user to the beginning of the MLE path, the beginning of the Contributor path, or to the All Solutions display page in the event a user does not wish to submit any data. We decided to make the visual design simple and easy to follow with color coded buttons for the different types of users.

4.1.6 *Alternative design*

The original design for our application contained a user login functionality. With animated modals for new and existing users and prompts for user ID's and passwords. We also initially planned to have a google sign in option. Our initial design for the home page contained only two buttons, one for MLE and one for Contributor that would each trigger a correspondingly color coded login modal. However the login functionality of our program was ultimately discarded as we adjusted our functionality priorities. We also did not originally have a navigation bar so the user could only access one of two paths and no other aspect of the application.

4.2 Contributor Upload page

4.2.1 *Role and primary function*

The Contributor Upload page is where a Contributor is prompted to upload their Time Series data. It contains required fields for metadata intake that will be used to parse, sort, and later catalog the time series data in the database.

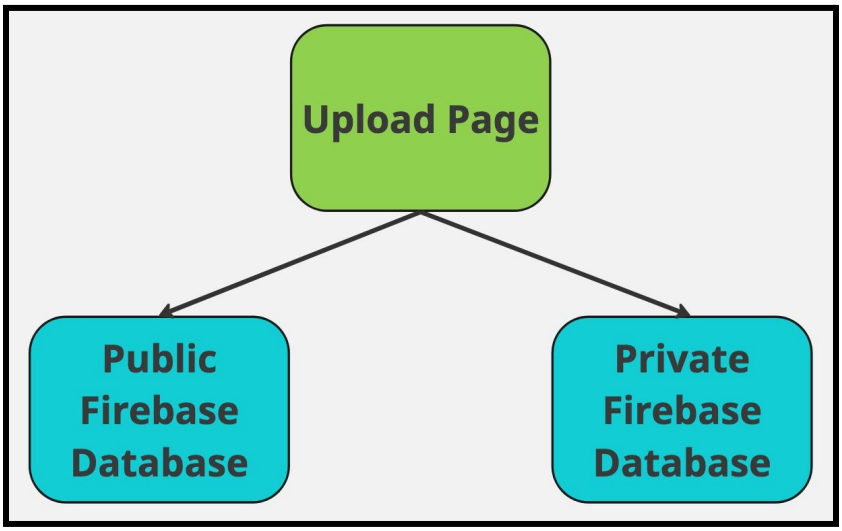
4.2.2 *Interface specification*

Before uploading, the Contributor Upload page prompts the Contributor to fill out two distinct forms to record the metadata of the test and training set the

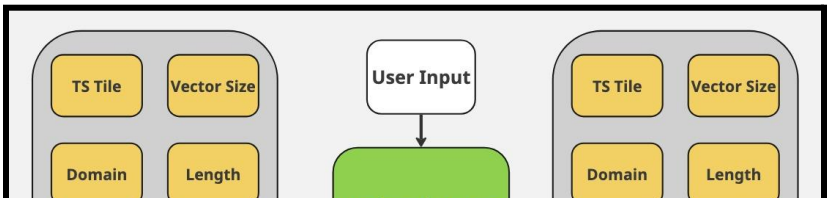
contributor wishes to upload. The contributor must provide the following information:

1) Time Series Title
2) Training Set Metadata
    a) Training Set TS Title
    b) Domain(s) of Time Series (what kind of data it is, finance, weather, etc)
    c) Units of the Time Series
    d) Is the Time Series Univariate or Multivariate
    e) Vector Size of the Time Series
    f) Time Series Length
    g) Time Series Sampling Period
3) Test Set Metadata
    a) Test Set TS Title
    b) Domain(s) of Time Series (what kind of data it is, finance, weather, etc)
    c) Units of the Time Series
    d) Is the Time Series Univariate or Multivariate
    e) Vector Size of the Time Series
    f) Time Series Length
    g) Time Series Sampling Period
4) Forecasting Task description
5) Target variable names for the Forecasting Task

The contributor then must choose a Training Data file, a Test Data file, and click the Upload Files button at the bottom of the page to complete their submission. They will then be notified whether or not their submission has been successful.



4.2.3 *Dynamic model*



4.2.4 *Static model*

4.2.5 *Design rationale*

On the contributor page, metadata fields were included along with the training set upload button and test set upload button so that the MLE can see the information that is important to them. Given time constraints, our current version of the MLE download page displays some of the metadata next to each training set. The metadata chosen was the forecasting task, the target variable that the MLE needs to worry about, and the test set length. These fields were chosen because they are the most relevant when an MLE needs to create a forecasting solution for one of these training sets. The forecasting task gives them their goal, the target variable defines what specific column of the training set their forecasting solution is concerned with (this column will have different “predicted” values and then will be compared with the test set values of that column), and the test set length lets the MLE know how many rows of data their solution should contain (this is not including the initial row which has the headers).

4.2.6 *Alternative design*

Given more time, some of the metadata fields would be condensed since metadata like “Domain” and “Units” are almost always identical between the training set and test set of a time series. Initially, a simpler design with one time series file upload and less metadata in the form was considered. However, after clarifying how a time series file is split up, where there is the training set and test set, and the training set is used separately by the MLE, our design was changed so that the contributor is expected to provide two separate files.

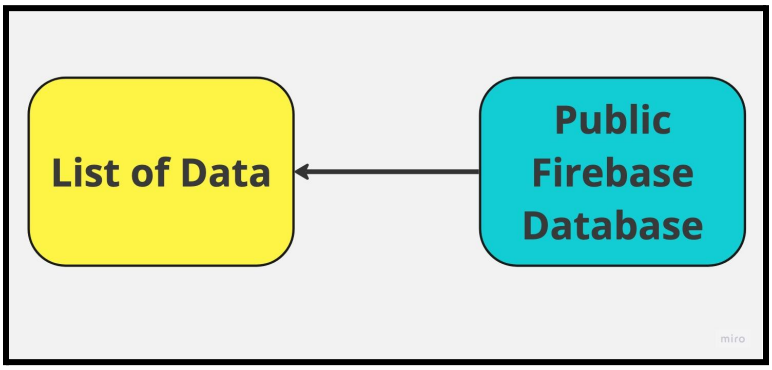
4.3 MLE Download page

4.3.1 Role and primary function

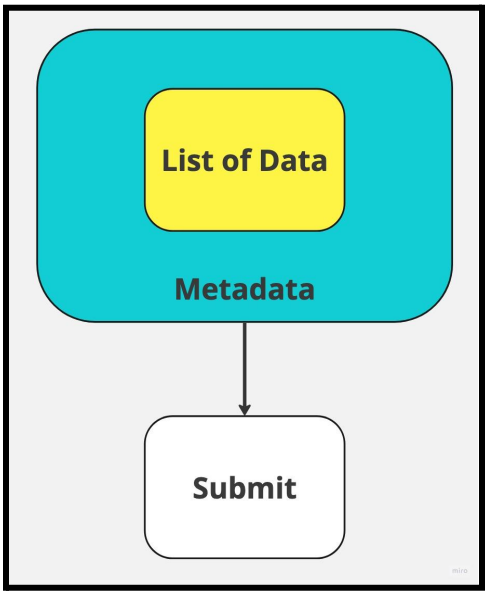
The MLE Download page is the first page on the MLE user path. It is where all the previously uploaded Training Data sets are accessible by the MLE. Here an MLE may view all the forecasting tasks for each Training Data set and choose which Training Data they wish to download to produce their forecasting solution. At the bottom of the page there is an Upload Forecasting Solutions button to prompt the MLE to the next step in the MLE path, which is the MLE Upload page.

4.3.2 *Interface specification*

Here the interface must access the database to retrieve and display all previously uploaded training data from the various contributors. To provide this service, the documents stored in the Firestore database will be used. The documents will be retrieved from the collection that stores all uploads from Contributors, and then transformed so that they can be displayed as a list of downloadable training set files on the page. The list of these files found on the download page will have the training set name, the forecasting task description, the target variable (column name to be used by the MLE), and the test set length. These pieces of metadata are included because they are necessary pieces of information that the MLE needs to know in order to perform their forecasting successfully.



4.3.3 Dynamic model



4.3.4 *Static model*

4.3.5 *Design rational*

The training sets are what will be listed for MLEs, because these are the files that they need in order to perform their forecasting. Since an MLE should not see a test set, only the training set with the key metadata (forecasting task, target variables) will be listed. Test set length is also listed because this defines how many data rows the MLE solution file should have. The system expects this data row count to match the test set that is stored in the database, so it's important to display this number to the MLE when they go to download a training set file.

4.3.6 *Alternative design*

Initially, a separate structure using InfluxDB was considered. However, after spending a bit of time trying to query data from the database, Firebase Firestore was used instead because retrieving data from documents is very intuitive with the NoSQL structure of collections, documents and then sub documents within those documents. These documents can easily be converted to dictionaries, which allows easy access in Python to any data needed.

4.4 MLE Upload page

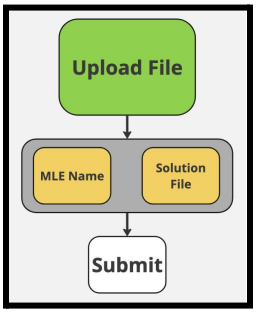
4.4.1 *Role and primary function*

The MLE Upload page is where an MLE user may upload their solution to a forecasting task. This page is designed to submit the solution data set to the database where it will be parsed for error analysis. After uploading the solution dataset it will direct you to the Error Analysis Display page to see the error analysis for the uploaded solution and Error Table for the relevant forecasting task.

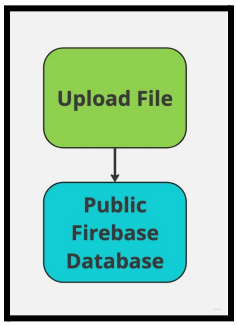
4.4.2 *Interface specification*

Here a user is prompted to input a MLE username that will be later attached to their specific error analysis for easy lookup in the All Solution's Display page. A

MLE user is also prompted to choose the solution file they wish to upload and have tested for accuracy.



4.4.3 *Dynamic model*



4.4.4 *Static model*

4.4.5 *Design rational*

This page was designed to be simple and easy to understand. In the table, the main piece of metadata that is needed is a name to differentiate each MLE, so that is the metadata field on this page that the MLE must fill out. Having an MLE name also makes it easy for an MLE to find their solution in the tables that are listed in the all MLE solutions page. The format of the upload file button was constructed to be similar to the contributor page. This allows one user to easily switch back and forth between roles and not be confused. The main change is the blue color scheme, to let the user know that they are on the MLE path instead of the green path which represents a Contributor page.

4.4.6 *Alternative design*

An alternative design that was considered contained more metadata fields. However, given time constraints and the fact that the main piece of info needed

to differentiate each MLE in the training set tables was an MLE name, the final page ended up only having one metadata field.

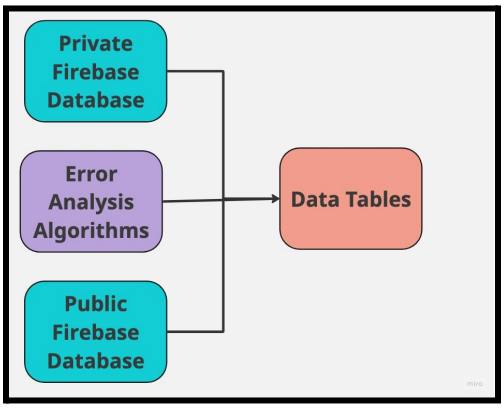
4.5 Error Table

4.5.1 *Role and primary function*

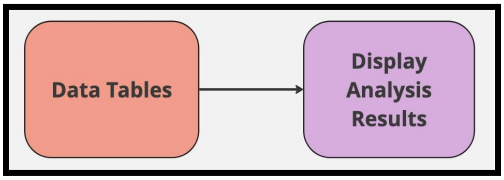
The Error Table is where the error analysis for each solution uploaded by a MLE user is displayed and able to be sorted by whichever metric the user desires. It serves as a dynamic storage element for the various error analyses for each forecasting task. The Error Analysis Display page contains one Error Table that corresponds to the forecasting task chosen by the MLE user. The All Solutions Display page contains Error Tables for all the forecasting tasks in the library.

4.5.2 *Interface specification*

For each MLE solution there are 6 different types of error calculations generated. Those six values are passed as a dictionary to the flask database. The Database stores the error analyses for each forecasting task together. That data is then sent as a list of sublists that each contain 8 values. These values include User, Parameter (variable type), Mean Absolute Error, Mean Absolute Percentage Error, Symmetric Mean Absolute Percentage Error, Mean Squared Error, Root Mean Squared Error, and the correlation coefficient. This data is then iterated over in the html display pages to populate a table element every time the display page is accessed. This way the error tables are constantly updated accordingly since they are constantly accessing all the submissions for each forecasting task.



4.5.3 Dynamic model



4.5.4 Static model

4.5.5 *Design rational*

By generating the table dynamically we are able to keep the library constantly up to date as users upload solutions. Separating and displaying the tables by forecasting tasks on the All Solutions Display page allows contributors to return and view the different solutions for each of their tasks. The current design allows users to sort each table manually for easy access and comparison of all the different metrics by providing interactive buttons as the column labels.

4.5.6 *Alternative design*

Originally, we thought the table would display the error analyses for only one user. This meant our original design was static and was generated only once upon MLE solution submission. This meant it didn't contain the submission's from previous MLE user's and was useless for comparing accuracy between different user submissions. It also meant that our display page would have to display tables for each user which resulted in multiple tables for each forecasting task

that made the page very difficult to navigate. The original design also did not allow for a sorting functionality since it didn't contain submissions from other users.

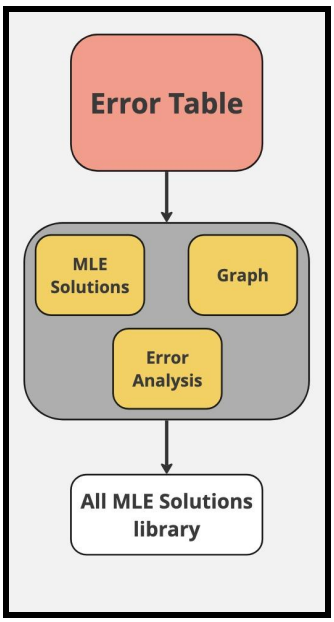
4.6 Error Analysis Display page

4.6.1 Role and primary function

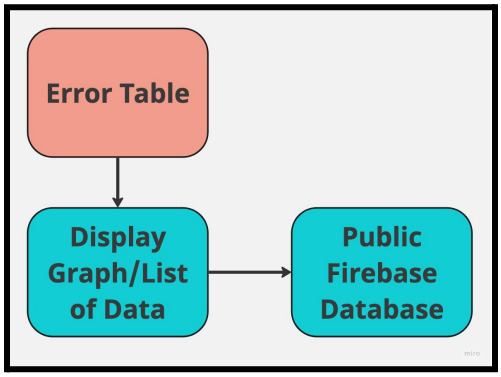
The error analysis display page serves to give the user an insight into the differences between their forecasting model that they uploaded, and the test case that exists for that model. Its primary function is to show the user a graph that will show the differences between their uploaded forecast model and the test case at different points in time for a specific parameter.

4.6.2 Interface specification

The parameter that the user passes in is of utmost importance in this case. The parameter will be passed into a function that will cull the tables to only keep the columns relevant to the parameters. A separate function runs to find the column containing the date. The date will then be put into the x-axis to serve as the measurement of data over time. The y-axis is filled up by two plots, one for the forecast and one for the test set. At each point in time, the plot graphs the values of the parameter (or average between parameters) on the plot, giving a clear view of the differences between the two tables over time, which the MLE can use to ascertain the accuracy of their data.



4.6.3 Dynamic model



4.6.4 Static model

4.6.5 Design rational

The plot was created this way for a very specific reason. The MLE needed a way to see their data vs the confirmed data in a way that was nice, clean, efficient, and simple. The easiest solution was to plot the two data graphs along the same line, to make for an easy comparison. We also needed a way to graph them over a certain period, and since time couldn't be incorporated into the error functions, since you can't run error functions on dates, so it made sense to set it as the x-axis. Finally, the parameter is used to isolate certain points of data. You can't take an average between two completely different units of data and expect a usable result, so parameters were chosen to give usable data.

4.6.6 Alternative design

There was another variation of the graph, one far less usable. Originally, each pair of data sets was going to be run against every error metric and individually put on a graph, with the x-axis of the graph being all 6 different error metrics. This was obviously not usable, since comparing error metrics against each other won't serve any purpose. It can't give you an average error, so it made more sense to give the error metrics individually together, and give a graph to the user so they themselves could work out their data with the error numbers, instead of automating it with an unreliable average.

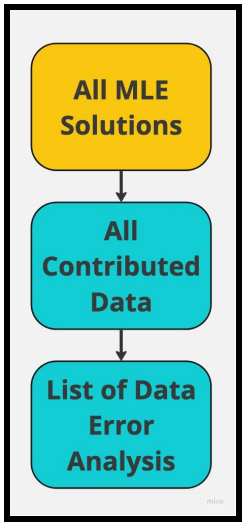
4.7 All Solutions Display page

4.7.1 Role and primary function

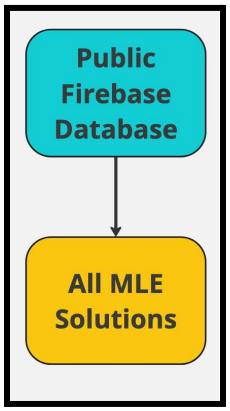
The All Solutions Display page serves as a repository for all the error analyses generated by the application for public view. It presents an Error Table for each forecasting task/Training set in the system. Each Error Table may be sorted by the desired attribute whether it's User Id, MAE, SMAPE, etc. It serves as the display page for all the information generated by our application and is constantly updated.

4.7.2 Interface specification

This display page is constantly updated because it creates the tables based off of what is in the Firestore database. Specifically, for each document in the "time_series_data" collection, a pair id (id for training set and test set from a contributor) is associated with it. So using this pair id, we can retrieve all MLEs from the MLE_solutions collection that have this pair_id saved as their "solution_pair_id". This allows us to split up the tables by training set, and then also retrieve all MLE's associated with each training set and display them in the appropriate table. Also, for each table the rows can be sorted by a column. The user must click the button corresponding to the column they would like sorted (like MAPE for example), and then the table rows will be sorted by that column in ascending order.



4.7.3 Dynamic model



4.7.4 Static model

4.7.5 Design rational

Tables were chosen to be divided by training set, because that is the file associated with a group of MLEs that would like to know how their results compare with other MLEs. It would not make sense to divide the tables by something more vague like Domain, because then the table becomes jammed with too much information and it becomes difficult for an MLE that just uploaded a solution for a specific training set to find their row in the table and see how they compare with other MLEs.

4.7.6 Alternative design

As was just mentioned above, an initial alternative design that was considered was having multiple tables displayed, where each table represented a domain, like Finance or Weather. However, this design did not seem to be a good solution as the table would contain a lot of data for many different training sets, and would be harder for the MLE to read than if the tables are split up by training set. Also, the client requested that the tables be split up by training set so that was a big reason to switch to our current design.

5. References

The Wikimedia Foundation 501(c)(3) (2003). All information for error analysis and error graphs was taken and cited from https://en.wikipedia.org/wiki/Mean_absolute_error

Grinberg, M. (2018). *Flask web development: developing web applications with python*. " O'Reilly Media, Inc."

J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp.

Wes McKinney, Pandas (2008), structure used to get data frames and turn them into data. https://doi.org/10.5281/zenodo.3509134

6. Acknowledgements

We would like to thank our clients Juan Flores and Mohammad Hasani, for not only giving us the opportunity to complete this project, but offering us guidance throughout the design process.