06 - Iteration 3: RAD and First Implementation

Edit New page

Jeff Caldwell edited this page now · 2 revisions

Barrios Technology - Forecasting and Prediction Modeling

Team Noname

Jeff Caldwell

Carlos Cardenas

Josue Lozano

Jeremy Miles

Jonathan Morgan

Introduction

i. Purpose of the system

The ISS analysis tool is a web application that provides analysts at Barrios, an aerospace company, with logistics optimization, usage forecasting, and prediction accuracy analysis surrounding consumable delivery to the International Space Station (ISS).

ii. Scope of the system

The system encompasses secure data input and transport via HTTP, data storage in a relational database, prediction accuracy analysis, logistics optimization analysis, and predictive modeling. All system functionality will be presented to users through a simplified and intuitive web-based interface.

iii. Objectives and success criteria of the project

iv. Definitions, acronyms, and abbreviations

- **ISS**: International Space Station
- **IMS**: Inventory Management System - Barrios' ISS inventory management datastore

- **ORM**: Object Relational Mapper - A framework-specific set of classes and methods for interacting with a database

v. References

- Django: A "full-stack" web framework written in Python
- HTMX: A client-side JavaScript library that enhances HTML for easy communication with the server
- PostgreSQL: A relational SQL database
- Pandas: A Python library for exploring and manipulating tabular data
- Prophet: A forecasting algorithm

vi. Overview

The ISS analysis tool will give Barrios analysts the ability to quickly transform user-provided data into three types of analysis: prior prediction accuracy assessment, logistics optimization, and predictive modeling.

Current System

The current system is a simplified approach based on the requirements defined in iteration 2. With this iteration, the code's focus is basic file upload and user data persistence. Additional work is underway to prototype predictive modeling, optimization, and accuracy analysis functionality that will be integrated into the application during a future iteration.

Proposed System

The proposed system will use an HTTP server framework named Django to facilitate a basic Model-View-Controller application structure. With this structure, controllers will respond to HTTP requests sent from a browser-based HTML interface augmented with the HTMX library and coordinate the creation of optimization analysis and predictive modeling using the Pandas data library and the Prophet algorithm. The system will also facilitate uploading and storing client-generated data using PostgreSQL.

i. Overview

ii. Functional Requirements

- Data upload, validation, and persistence functionality
- Accuracy analysis, logistics optimization, and predictive modeling algorithms
- Intuitive and aesthetically pleasing user interface

iii. Non-functional requirements

a. Usability

Interface:

- The web application should have an intuitive and appealing user interface that works on various screen sizes.

User management:

- Administrators should be able to create additional users with different levels of access.

Account management:

- Users should be able to log in, log out, and reset their passwords.

Ease of use:

- Users should be able to upload and verify data in a straightforward and intuitive way. Additionally, users should be able to view analyses in a high-level overview and have access to more detailed views of each analysis.

b. Reliability

Data:

- User data should be persisted to the appropriate database table and stored as a redundant flat file on the filesystem.

c. Performance

Client performance:

- Consideration for browser performance and interface response times should be paramount. Undue amounts of JavaScript should be avoided.
- The display of large amounts of tabular data should be approached iteratively — using pagination or "infinite scrolling" to load smaller subsets of the data at a time.
- Visualizations should be lightweight.

Server performance:

- Persistence of user data should be fast, especially in the face of multi-gigabyte source uploads.
- Analysis functions should take every measure to perform as quickly as possible.

d. Supportability

- The application will be easy to use on any modern web browser.
- Documentation of the underlying application code and application deployment guidance will be provided.

e. Implementation

- Must work well in all modern browsers.

f. Interface

- The interface should be responsive.
- The interface should look good and communicate intention.

g. Packaging

- The system will be deployed to cloud-based infrastructure using a containerized environment for the server.
- Could infrastructure must include a PostgreSQL database.

h. Legal

- The system should use open source libraries with permissive licensing.
- Client data should remain confidential
- Student should remain the intellectual property of the students

iv. System Models

a. Scenarios

File Upload Scenario

Initial Assumption: A Barrios analyst is logged into the analysis application and has the required permissions to upload files. The files to be uploaded are in CSV format.

Normal: The analyst presses the "Upload Data" button located in the left sidebar, then chooses a CSV file or a set of CSV files to upload to the application. Uploaded files are immediately validated against a set of "dictionaries" that match known Barrios datasets and inserted into the appropriate user data table. An error is returned to the user if the file does not match a known dictionary. An entry is then created in an upload table and each entry in the user data table is associated with that upload record.

After the upload and persistence process is completed, a list of links to file previews is presented to the user. Each link leads to a page with an HTML table representation of the data they uploaded.

Request Analysis Scenario

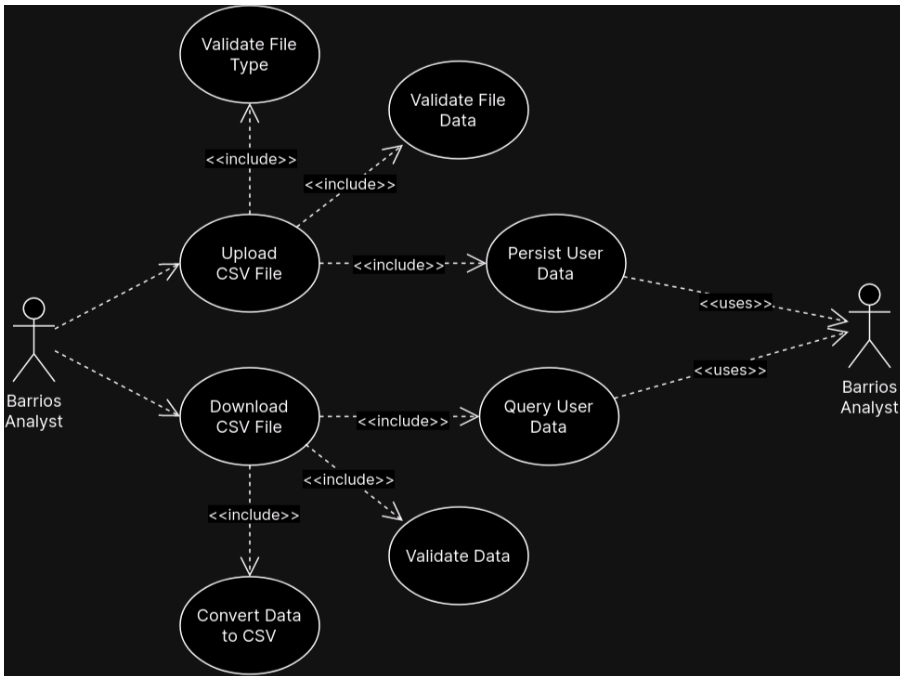
Initial Assumption: A Barrios analyst is logged into the analysis application and has the required permissions to view and generate analyses.

Normal: The analyst chooses the type of analysis they want, either a forecast or an optimization, and the date range they wish to use for analysis. Their date-range selection will be limited to a range of historical data in the application's database for that type of analysis.

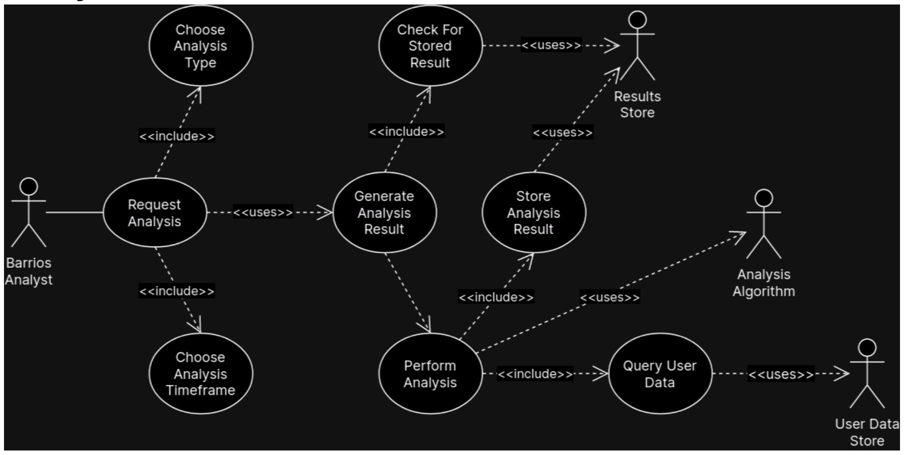
When they are done making their selections, the application will either return a stored analysis result based on their selection or generate the appropriate analysis and persist the results to the database.

b. Use case model

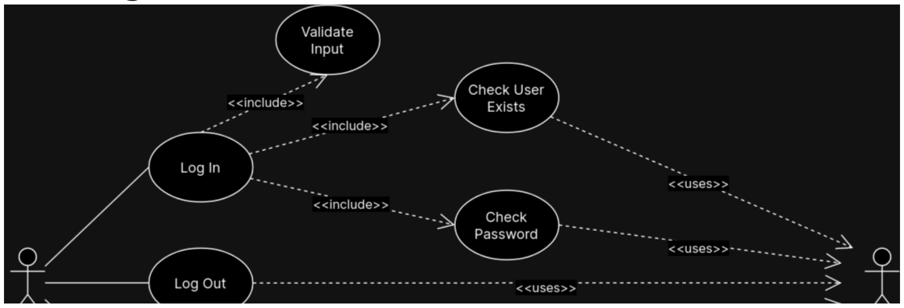
Upload Data Use Case

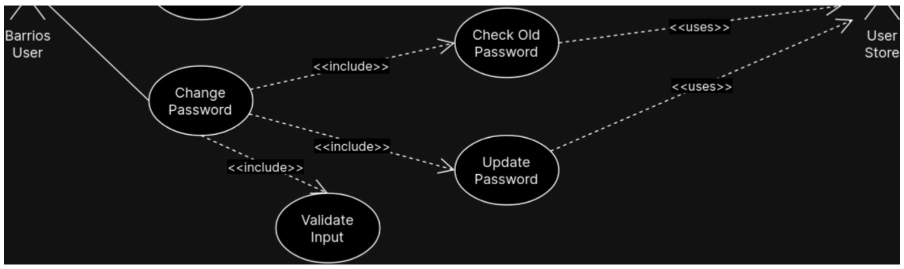


Analysis Use Case

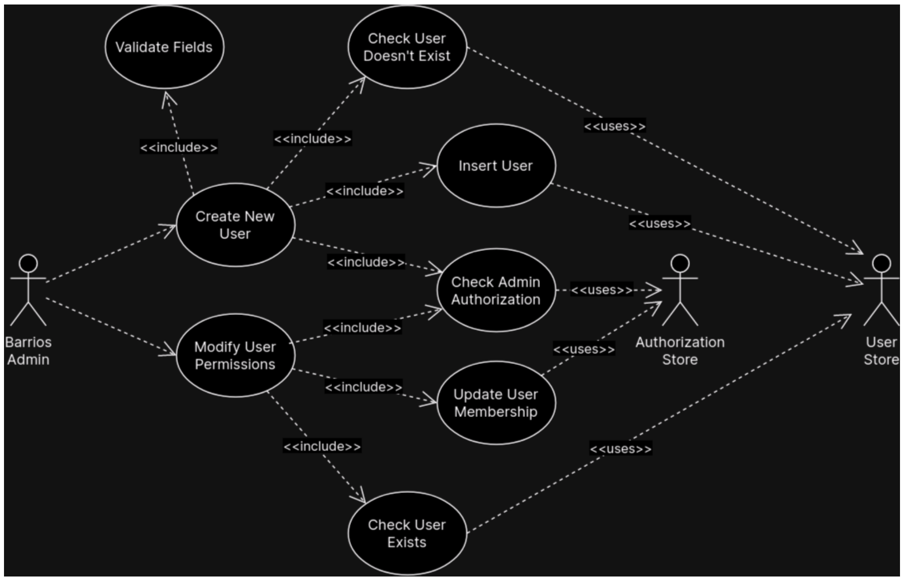


User Login Use Case

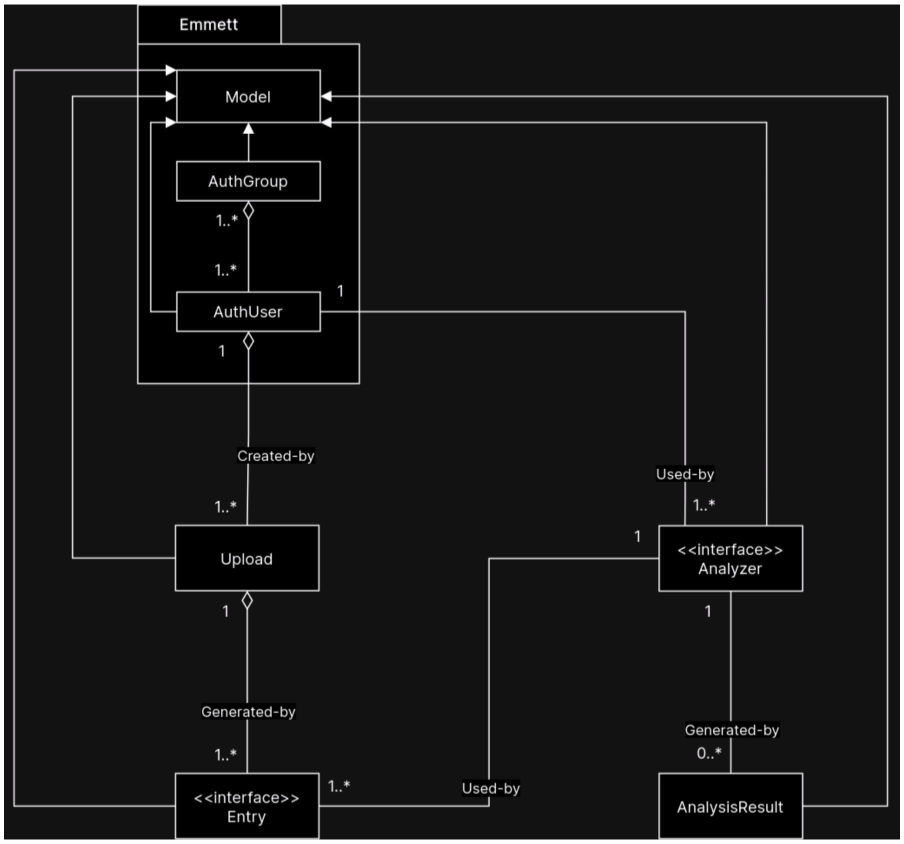


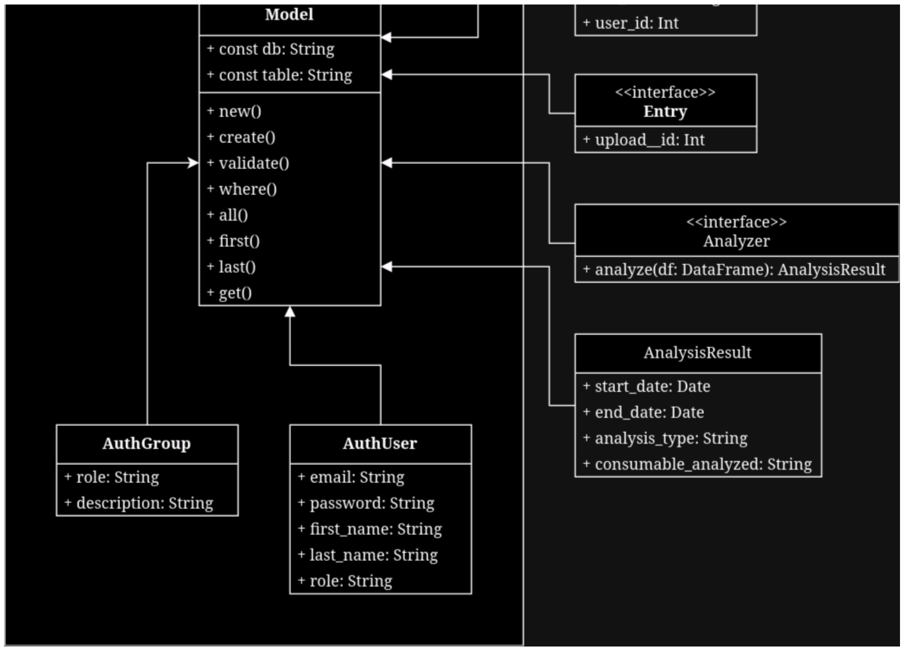


User Administration Use Case



c. Object model

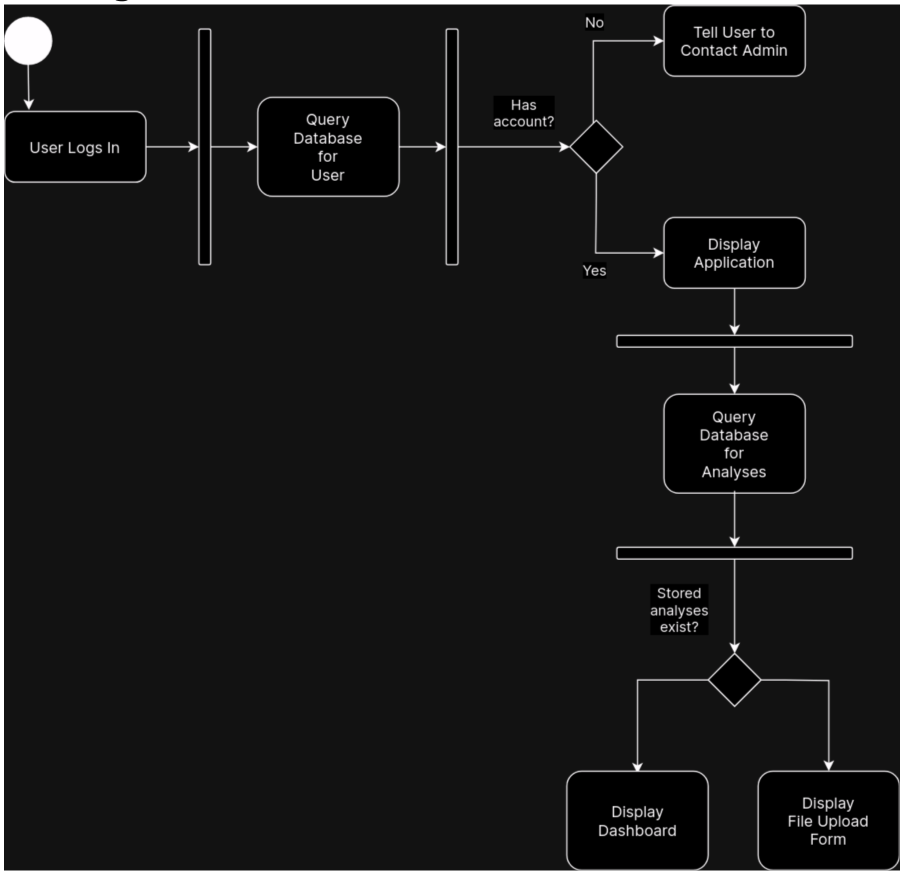




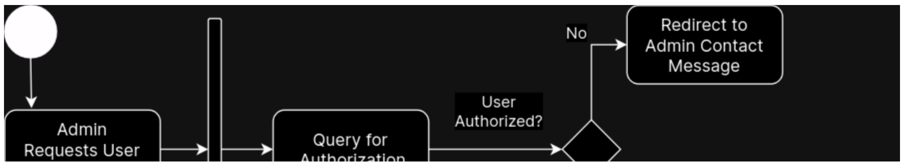
d. Dynamic Models

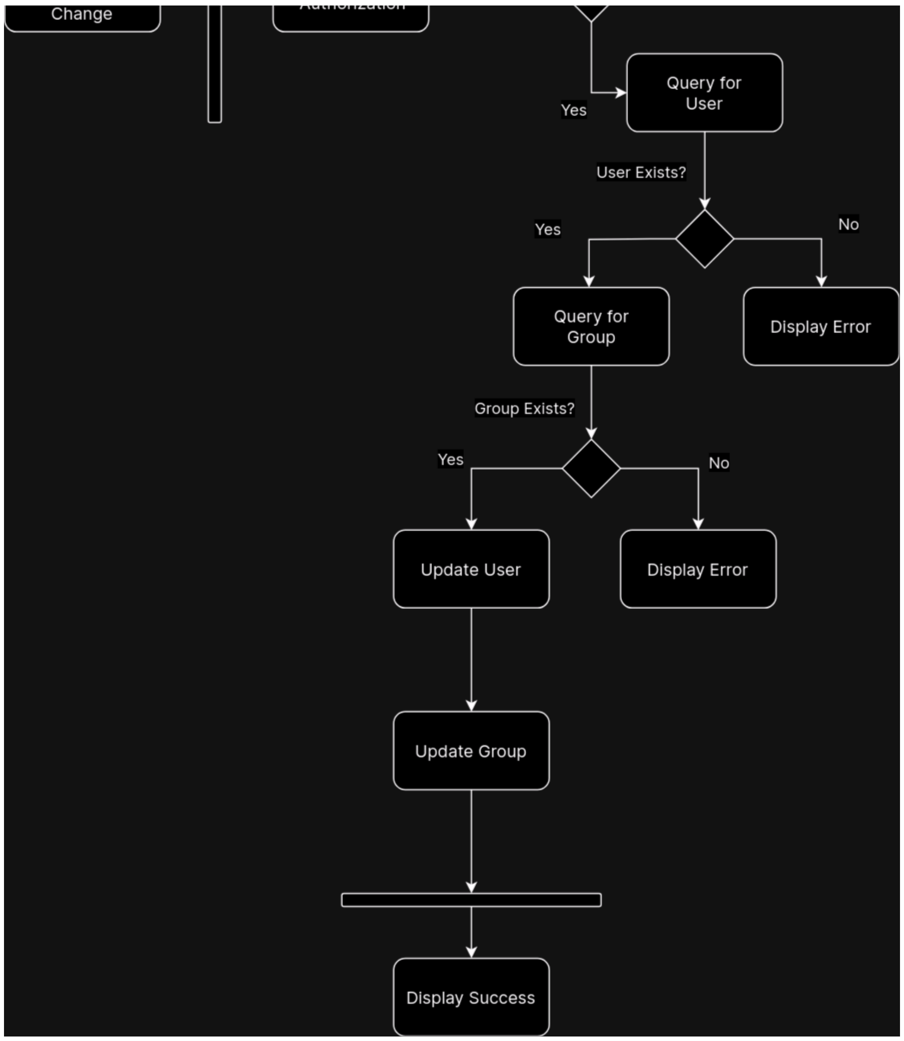
Activity Diagrams

User Sign-in

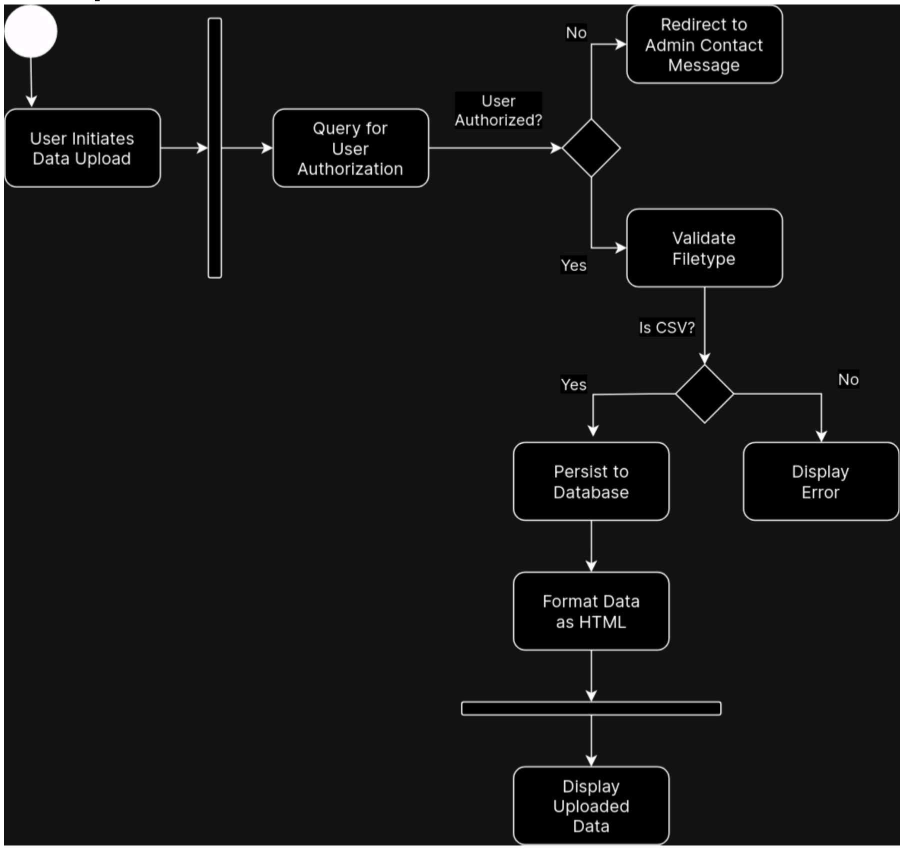


User Administration

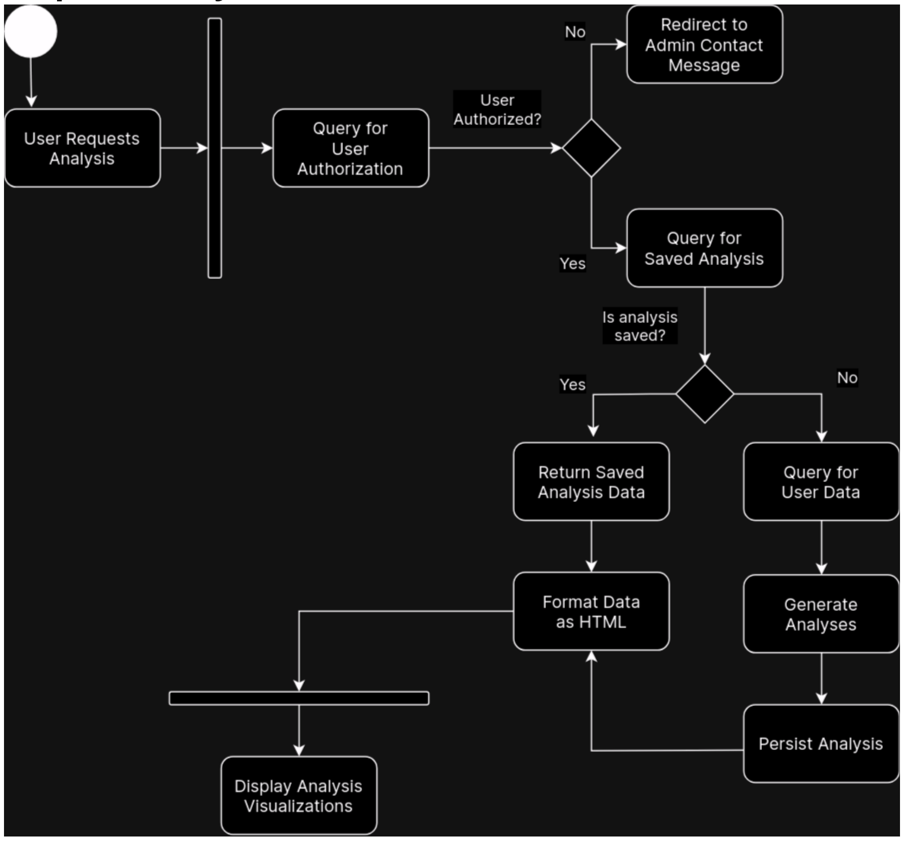



Change
Authorization
Yes
Query for User
User Exists?
Yes
No
Query for Group
Display Error
Group Exists?
Yes
No
Update User
Display Error
Update Group
Display Success

File Upload

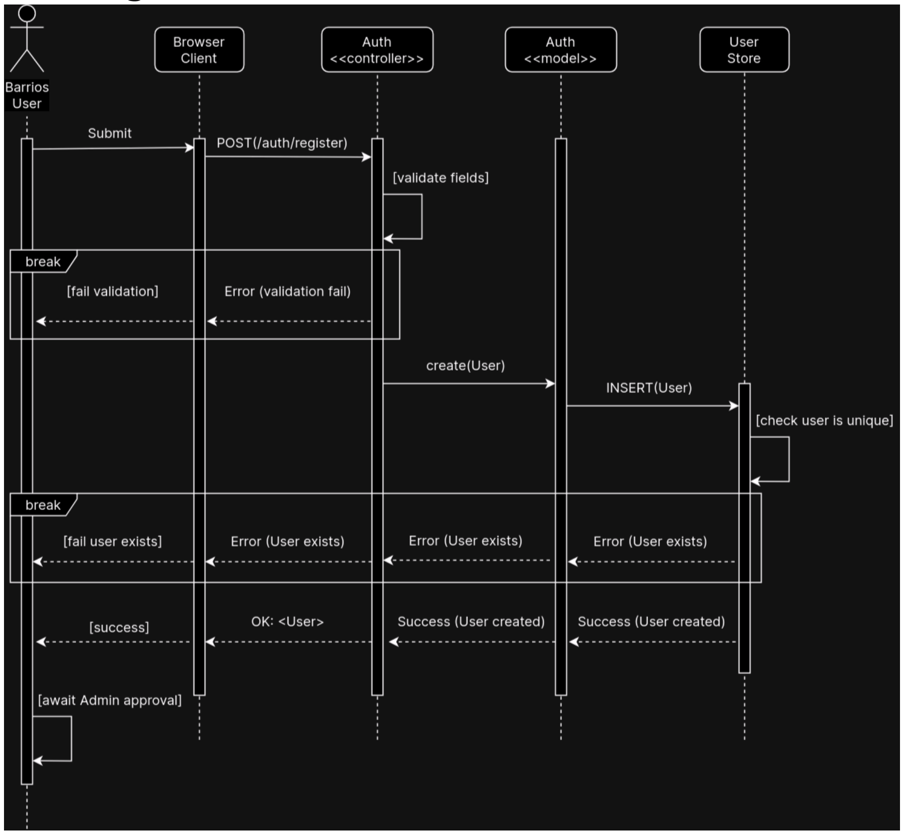

No
Redirect to Admin Contact Message
User Initiates Data Upload
Query for User Authorization
User Authorized?
Yes
Validate Filetype
Is CSV?
Yes
No
Persist to Database
Display Error
Format Data as HTML
Display Uploaded Data

Request Analysis



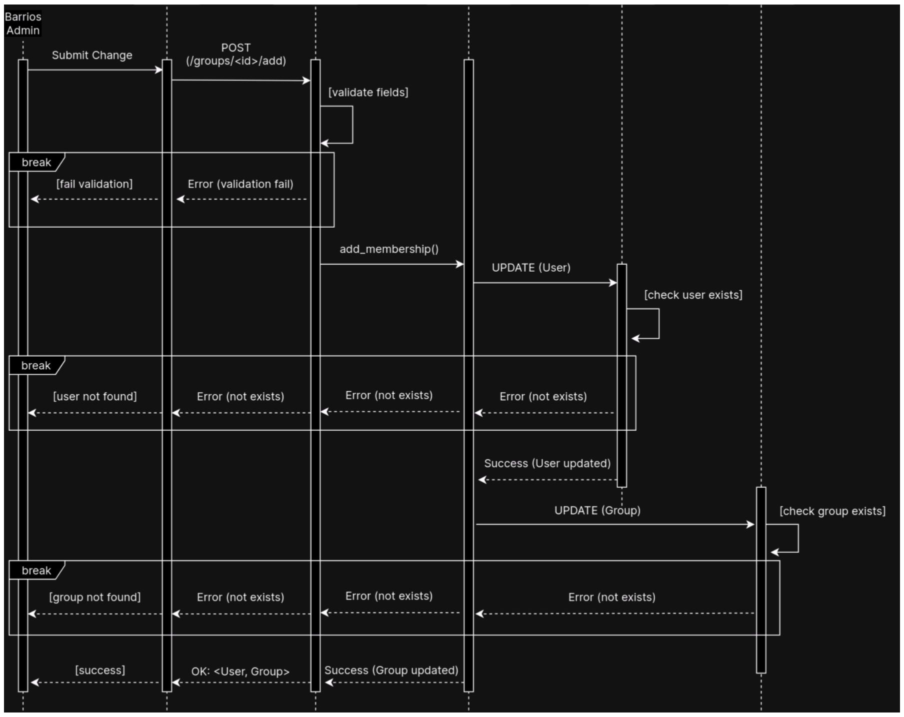
Sequence Diagrams

User Registration

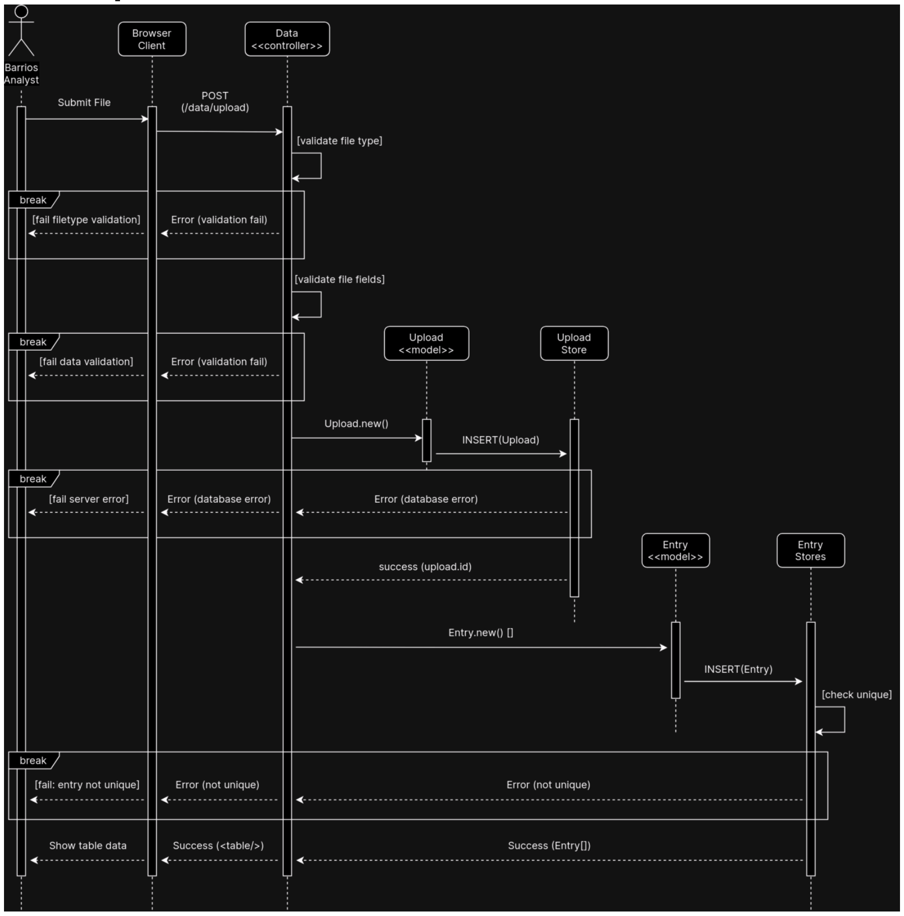


User Administration

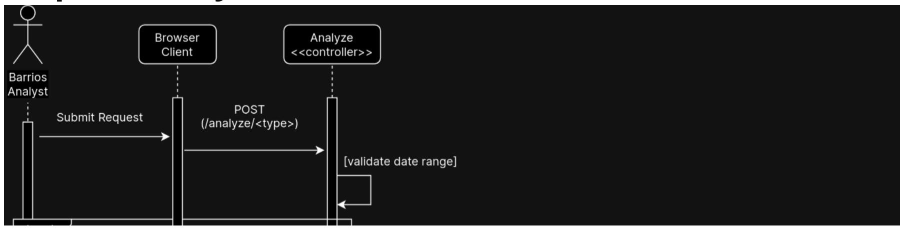
Browser Client
Auth <<controller>>
Auth <<model>>
User Store
Group Store

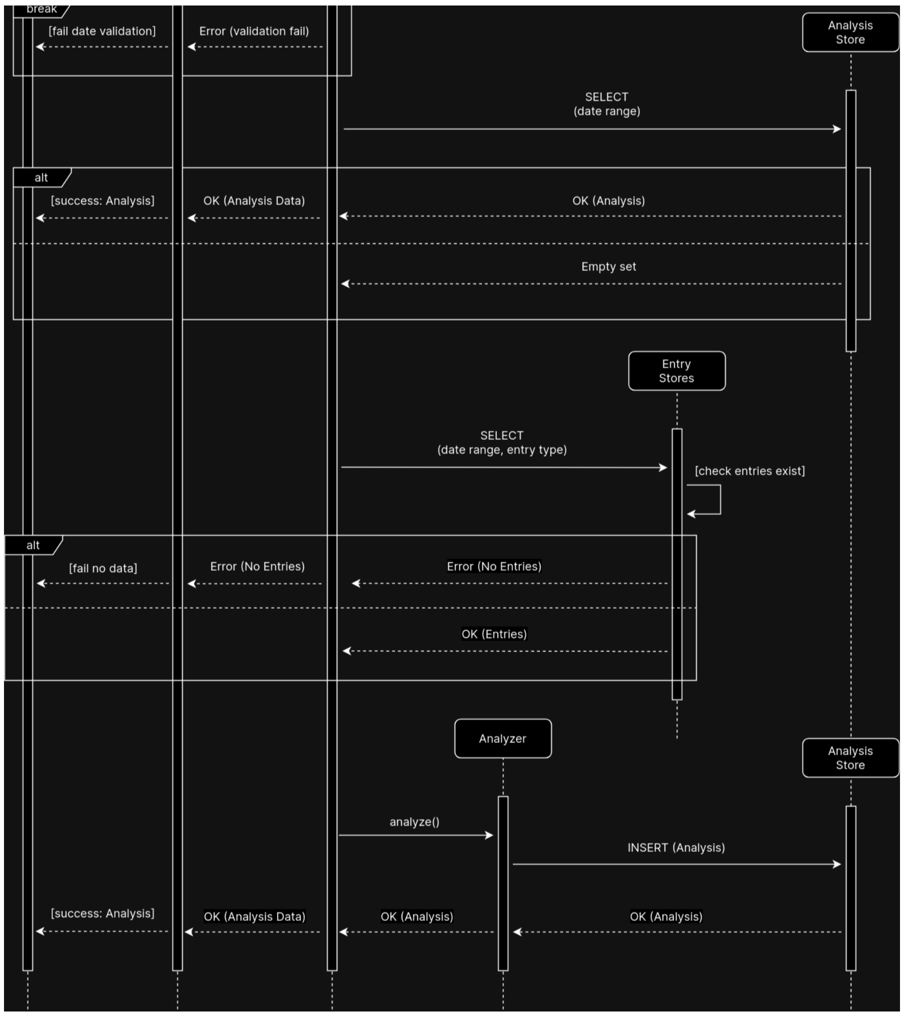


Data Upload

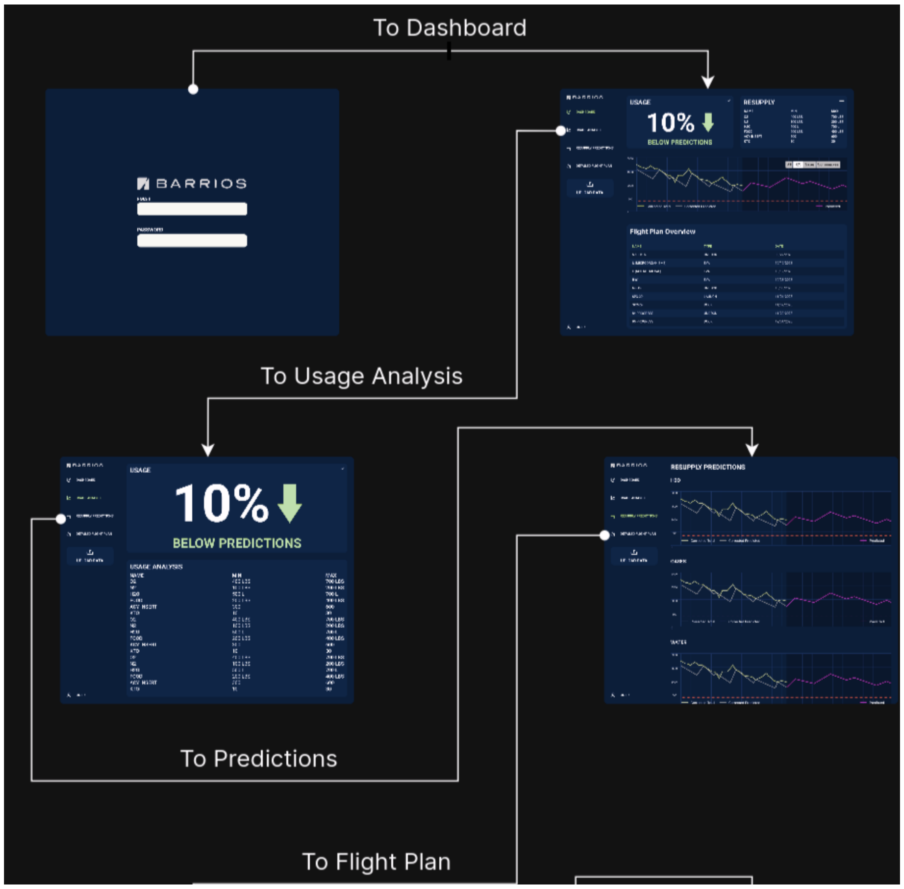


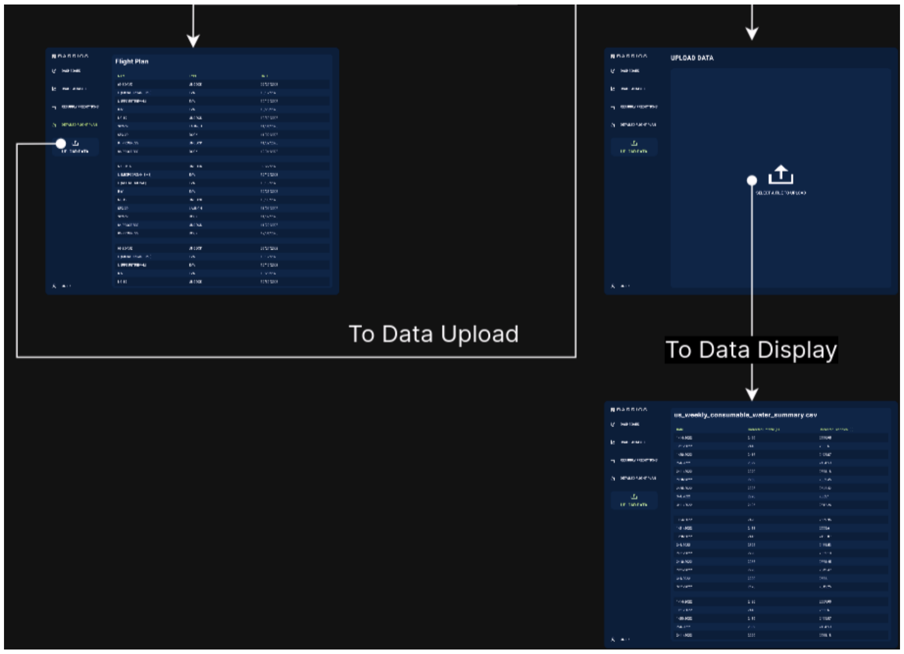
Request Analysis





e. User interface - navigational paths





Glossary

Prophet: A timeseries forecasting algorithm developed by Facebook/Meta **Emmett**: A Python web framework that provides routing (controllers), templating (views), and models. The framework also includes an "object relational mapper" for working with relational databases. **Object Relational Mapper**: Library with methods or functions for interacting with a database without the need for writing queries directly. **Inventory Management System (IMS)**: The database/datastore used by Barrios for tracking logistics and consumables for the International Space Station.

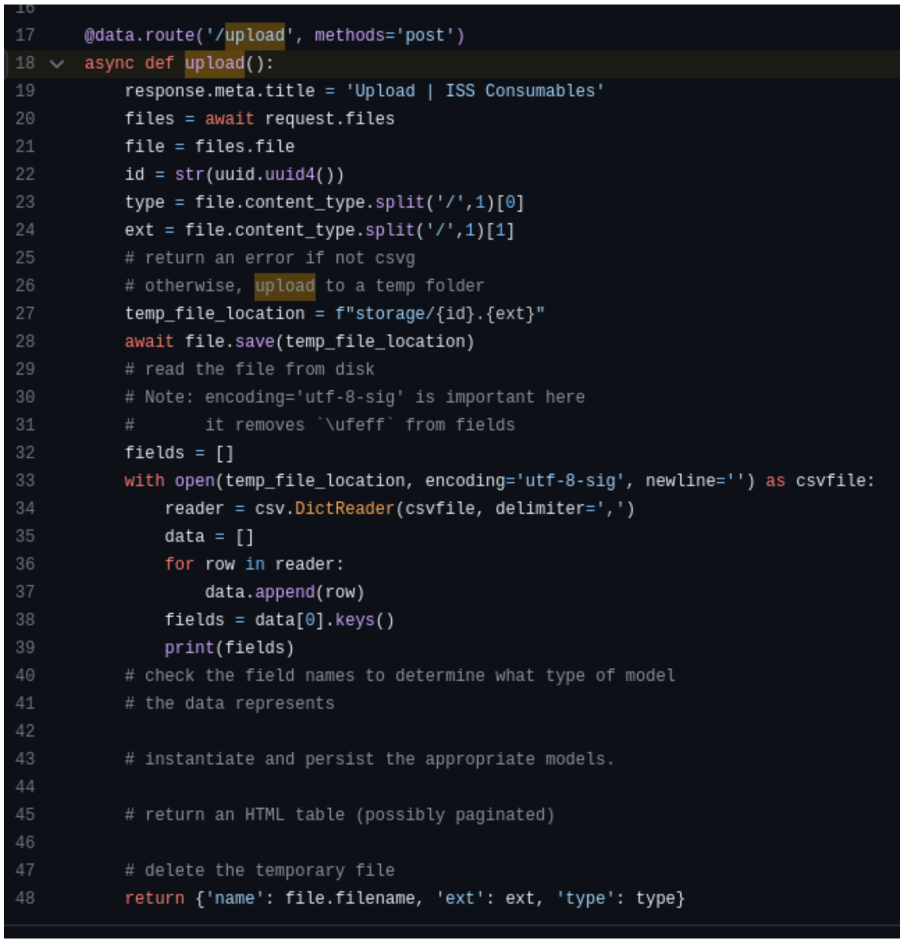
First Iteration

Much of this iteration has been centered around exploring the client data, prototyping analysis functions, and implementing basic functionality.

1. Implemented the beginnings of a file upload mechanism, which currently takes multipart form data, saves a file to the filesystem, and reads the field names of the provided tabular data. In the future, these field names will be used to determine which `UserData` table to save the user data to.

Data Controller - Upload Route

Data Controller - Upload Route



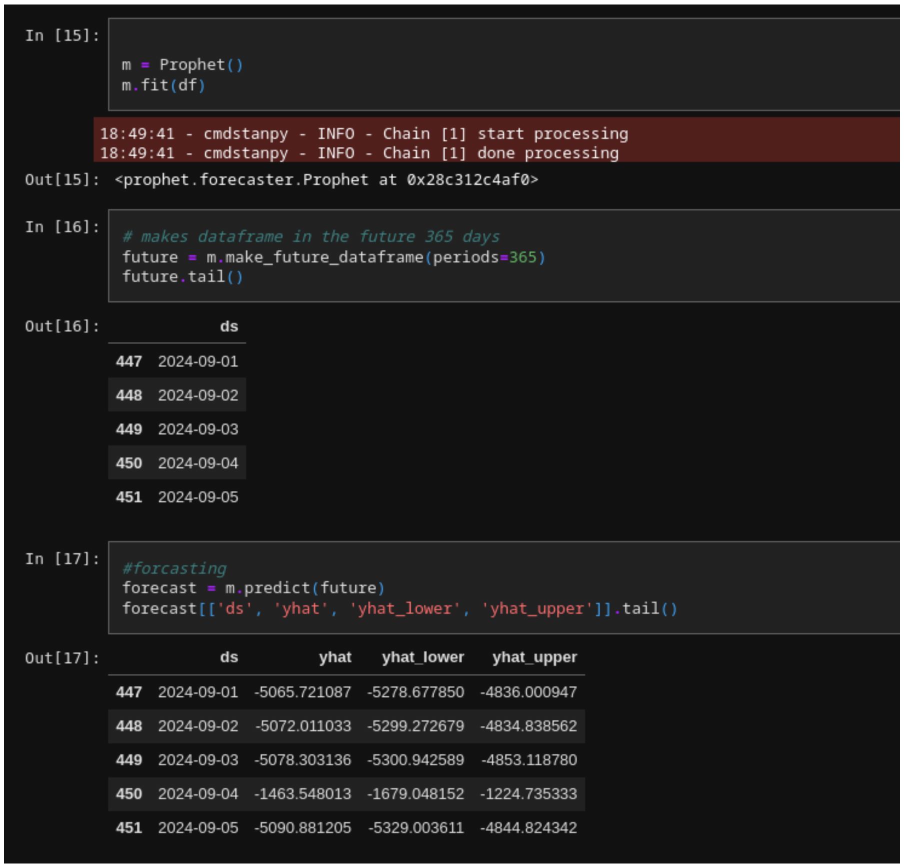
```python
@data.route('/upload', methods='post')
async def upload():
    response.meta.title = 'Upload | ISS Consumables'
    files = await request.files
    file = files.file
    id = str(uuid.uuid4())
    type = file.content_type.split('/',1)[0]
    ext = file.content_type.split('/',1)[1]
    # return an error if not csvg
    # otherwise, upload to a temp folder
    temp_file_location = f"storage/{id}.{ext}"
    await file.save(temp_file_location)
    # read the file from disk
    # Note: encoding='utf-8-sig' is important here
    #       it removes `\ufeff` from fields
    fields = []
    with open(temp_file_location, encoding='utf-8-sig', newline='') as csvfile:
        reader = csv.DictReader(csvfile, delimiter=',')
        data = []
        for row in reader:
            data.append(row)
        fields = data[0].keys()
        print(fields)
    # check the field names to determine what type of model
    # the data represents

    # instantiate and persist the appropriate models.

    # return an HTML table (possibly paginated)

    # delete the temporary file
    return {'name': file.filename, 'ext': ext, 'type': type}
```

Prophet Algorithm - [Prototype Notebook]



```
In [15]:
m = Prophet()
m.fit(df)
```

```
18:49:41 - cmdstanpy - INFO - Chain [1] start processing
18:49:41 - cmdstanpy - INFO - Chain [1] done processing
Out[15]: <prophet.forecaster.Prophet at 0x28c312c4af0>
```

```
In [16]:
# makes dataframe in the future 365 days
future = m.make_future_dataframe(periods=365)
future.tail()
```

Out[16]:

	ds
447	2024-09-01
448	2024-09-02
449	2024-09-03
450	2024-09-04
451	2024-09-05

```
In [17]:
#forcasting
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

Out[17]:

	ds	yhat	yhat_lower	yhat_upper
447	2024-09-01	-5065.721087	-5278.677850	-4836.000947
448	2024-09-02	-5072.011033	-5299.272679	-4834.838562
449	2024-09-03	-5078.303136	-5300.942589	-4853.118780
450	2024-09-04	-1463.548013	-1679.048152	-1224.735333
451	2024-09-05	-5090.881205	-5329.003611	-4844.824342

+ Add a custom footer