Iteration 5: Final Design Document

Jeff Caldwell edited this page now · 28 revisions

Barrios Technology - Forecasting and Prediction Modeling

Team Noname

Jonathan Morgan

Carlos Cardenas

Josue Lozano

Jeremy Miles

Jeff Caldwell

1. 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. Design goals

1. Provide a clean, usable, intuitive interface for Barrios employees to upload user-generated data and view analysis results.
2. Provide accurate and useful analyses, including supply-chain optimization, predictive modeling, and prediction accuracy analysis.

iii. 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
- **ASGI**: Asynchronous Server Gateway Interface - An asynchronous calling convention for Python web servers
- **DOM**: The *Document Object Model* - A data structure that represents an `HTML` document.

iv. 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
- Hypermedia APIs vs. Data APIs
- PostgreSQL: A relational SQL database
- Pandas: A Python library for exploring and manipulating tabular data
- Prophet: A forecasting algorithm
- SciPy: A collection of algorithms that include optimization functions
- Docker: Virtualization software commonly used to deploy web applications

v. 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.

2. Current software architecture

The current architecture is, for the most part, dictated by the architecture of the Django web framework. This framework follows a "Model-View-Controller" pattern, though it uses different names for each component of that pattern internally. Django provides the `View` class as a traditional controller and a `Template` as a view. `Models` remain the same in Django as in traditional MVC.

3. Proposed software architecture

i. Overview

Clone this wiki locally

`https://github.com/4306-te`

As stated above, the architecture will be based on a variation of `MVC`, and the organization of each architecture component will be centered around individual use cases that follow the application's user interface. Each discrete page presented through the application interface is represented within the system's directory structure as what Django refers to as an `App`. In practical terminology, an `App` is simply a subdirectory within the project containing the views, models, and associated complimentary classes or functionality to help everything work together.

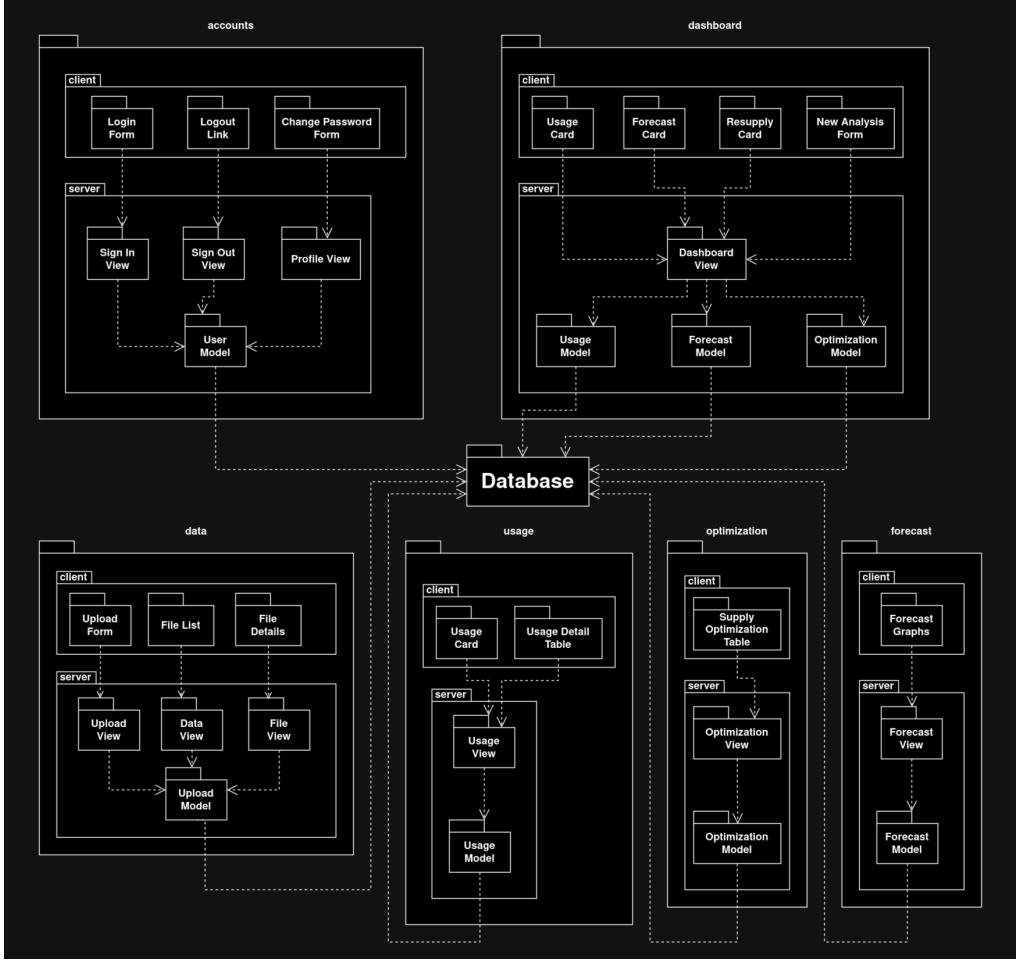
```
# An example use case or "App" directory

├── data
│   ├── admin.py
│   ├── apps.py
│   ├── forms.py
│   ├── migrations
│   ├── models
│   ├── tests
│   ├── urls.py
│   ├── utils
│   └── views.py
```

With this structure, views 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 through models that use either the Pandas data library, SciPy, and the Prophet algorithm. The system will also facilitate uploading and storing client-generated data using PostgreSQL.

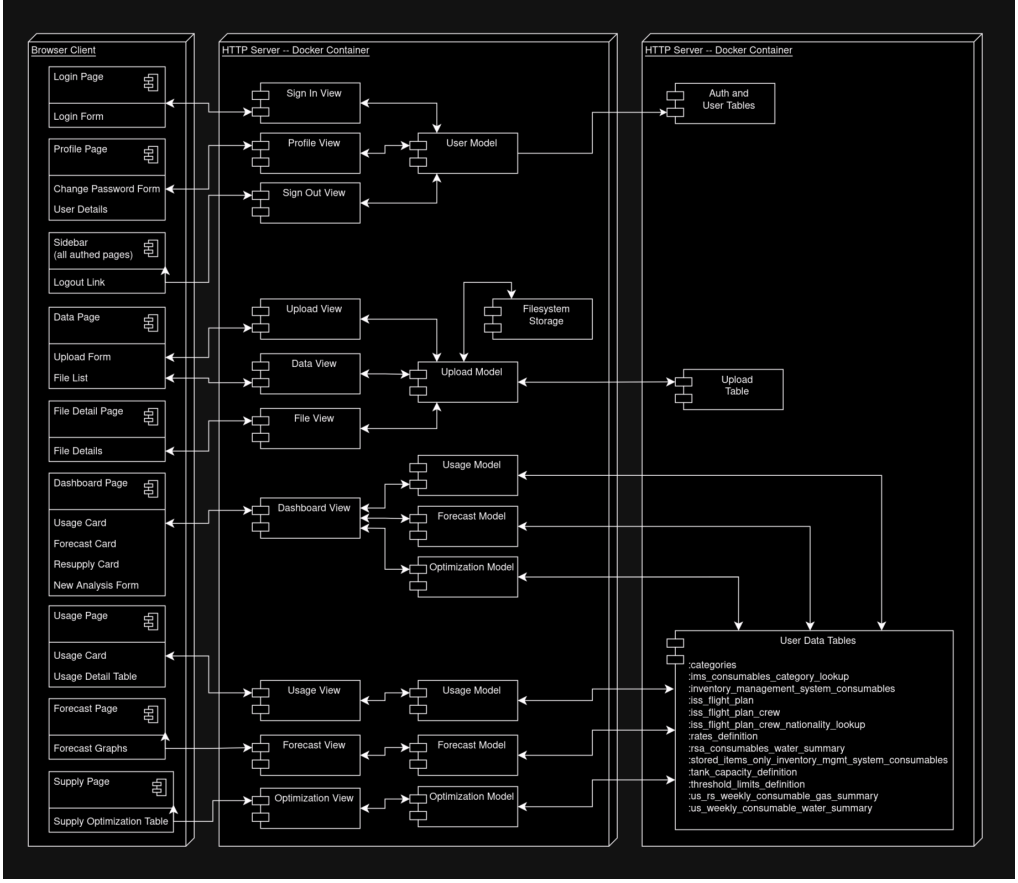
ii. Subsystem decomposition

This high-level overview of the ISS analysis system is based on the architecture of the Django HTTP framework. `Apps` within the larger server application contain `Views` and `Models`. `Views` coordinate between `Models`, `Templates`, and the user interface on the client's browser. `Models` represent different types of data needed for user management and analysis, and they coordinate the persistence and retrieval of data from the database. A detailed definition of these interactions will be shown in this document's **Subsystem Services** section.



iii. Hardware/Software Mapping

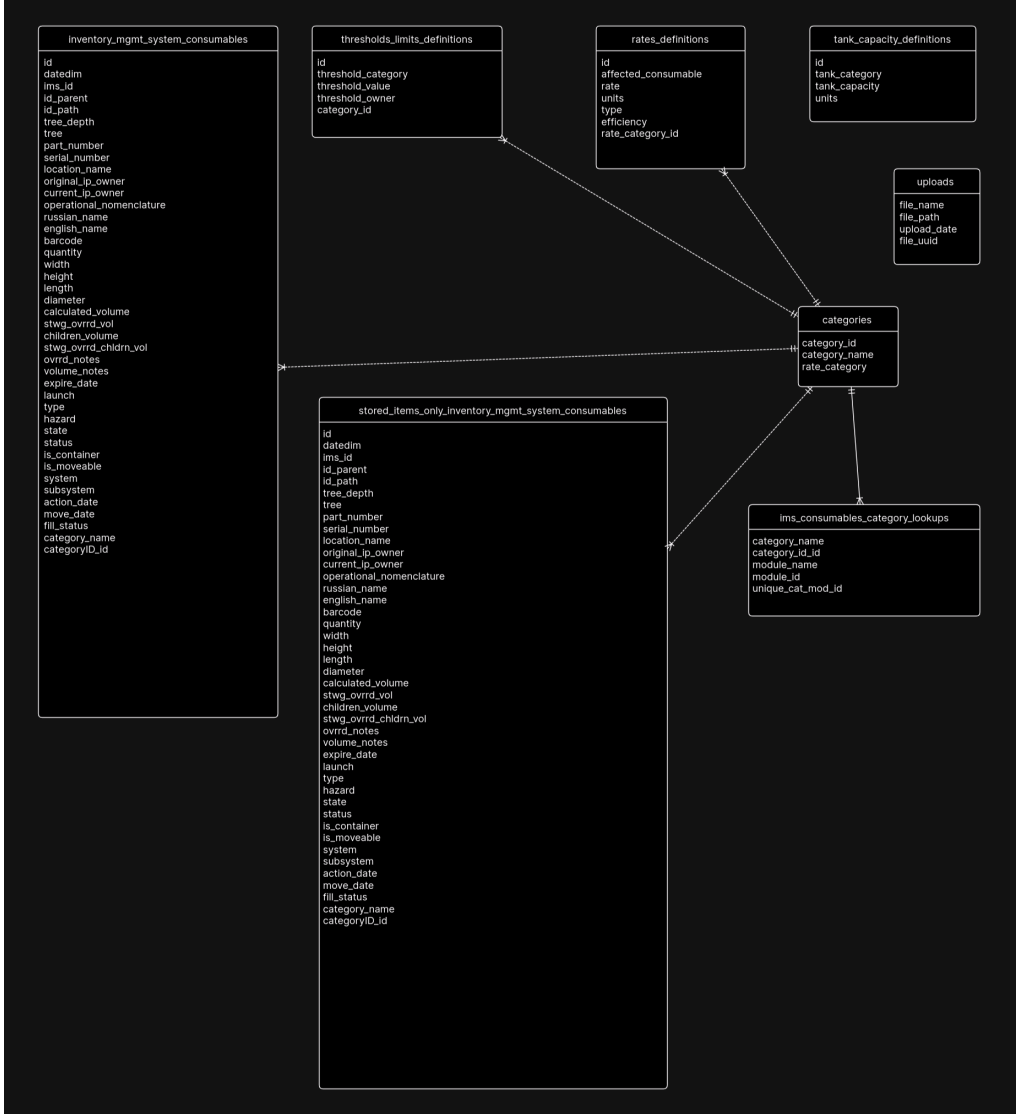
The system will be packaged for deployment on either cloud or on-premises machines using Docker containers. A PostgreSQL server will be required, which can be locally installed on the host machine or accessed through a connection to a remote PostgreSQL server. For the sake of simplicity, the PostgreSQL requirement is shown as a local installation below.



iv. Persistent Data Management

User Data and File Storage Entities

The following set of entities, with the exception of `uploads`, represents the datatypes contained in the client's data that is not specifically time-series data. The majority of the entities in this set are related by a category ID or a category name. Not all references to a category ID or a category name share the same nomenclature, requiring the inclusion of the relation table "categories."

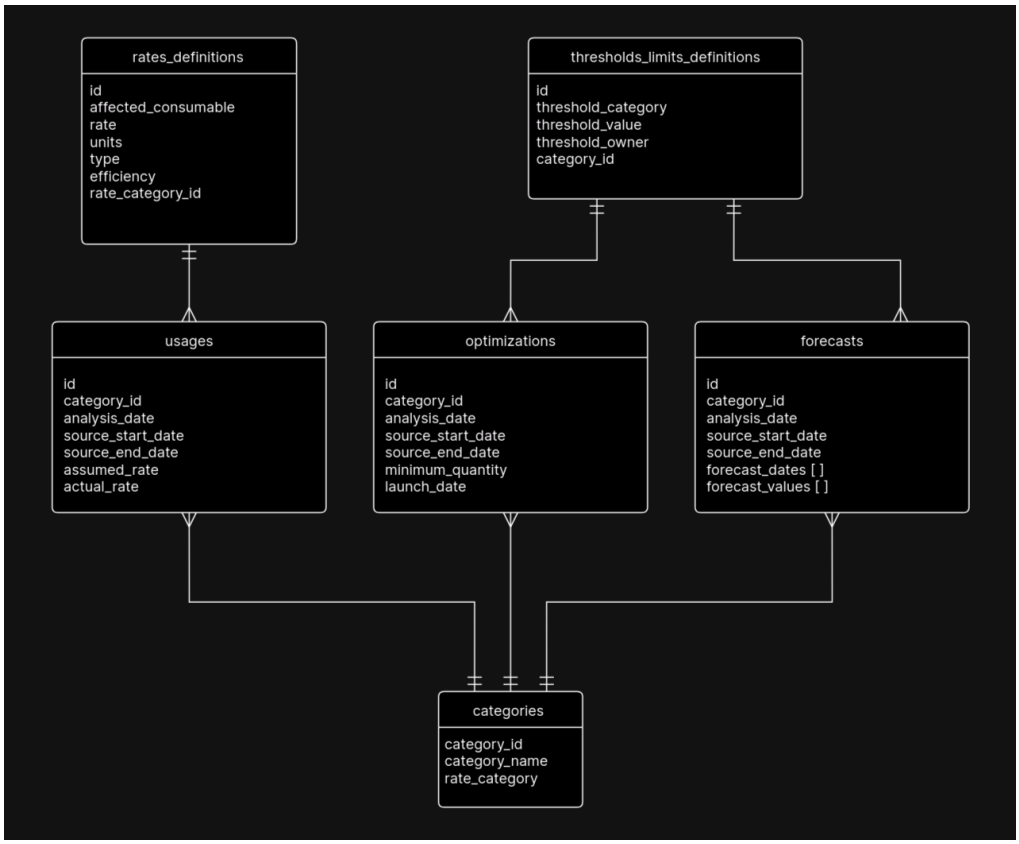


Time Series Entities

The following entities represent user data that follows a time-series structure. Most of the entities are not related, with the exception of the two crew flight plan tables — `iss_flight_plan_crews` and `iss_flight_plan_crew_nationality_lookup`.

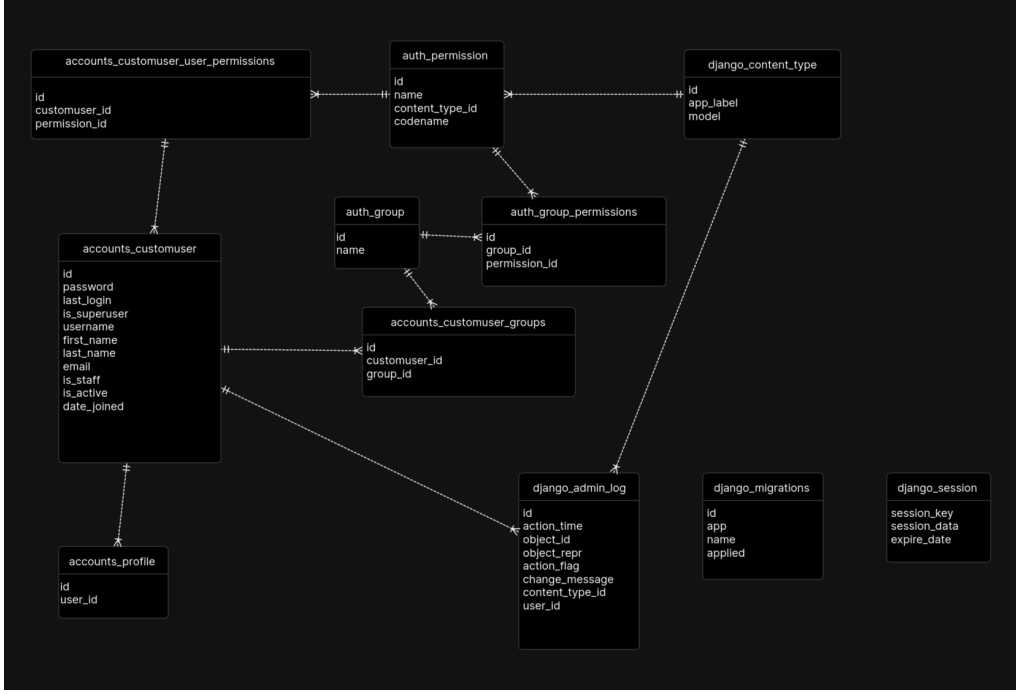
Analysis Entities

Three of the following entities are designed to record the results of different analysis functions. Depending on which type of analysis — usage, optimization, or forecast — some of these entities are related to entities defined earlier. All three entities are related by `category_id`.



User and Django-Specific Entities

The Django framework provides these entities for user management, authentication, and database migrations. The only modification made by Team Noname was the creation of a `CustomUser`, which is suggested by the Django documentation as a means to facilitate the addition of custom fields onto the default `User` model.



v. Access control and security

Access matrix

Role	Analyses	View Files	Upload Data	
Viewer	Read			
Analyst	Read/Write/Execute	Read	Read/Write/Execute	
Admin	Read/Write/Execute	Read	Read/Write/Execute	R

vi. Global software control

Global software control for this system will be handled with a server-client architecture implemented using the Django web framework. This will allow the server to respond to HTTP requests from a browser with HTML templates. These templates will be augmented using the HTMX library.

Synchronization between the browser and server will be handled via HTTP. Rather than issuing responses using an intermediate language like `JSON`, server endpoints will respond with HTML. This will be possible by using HTMX, a small JavaScript library that makes asynchronous calls to a web server and replaces appropriate sections of the `DOM` with the server's response. This removes the need for a client-side JavaScript framework and reduces the overall complexity of the application by ensuring all application state remains on the server.

vii. Boundary conditions

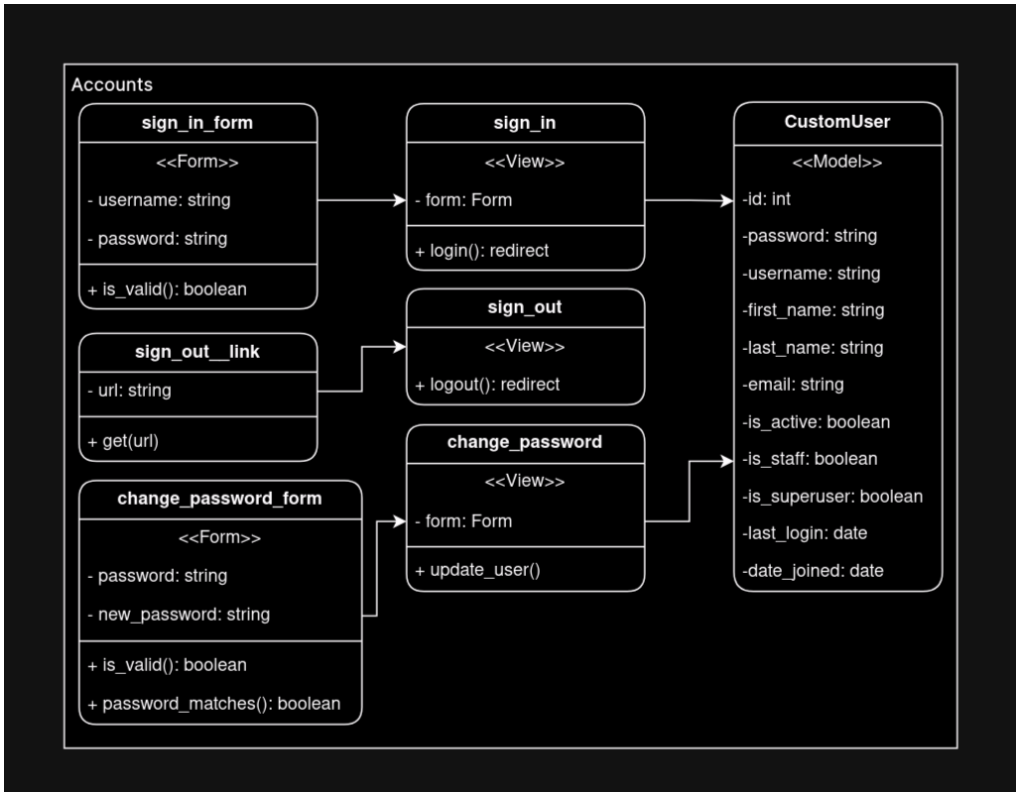
Startup The server software will run on what is essentially an infinite loop, waiting for HTTP requests. The true system, however, will start up when a client makes a request. If that request is targeted at a valid route, the request will be processed by a controller. If not, the system will return a 404 error.

Errors All exceptions will be collected in an error log for additional analysis by developers. If the exception is severe enough that it fails to process a client request properly, it will be returned to the client as an error message with an HTTP status code.

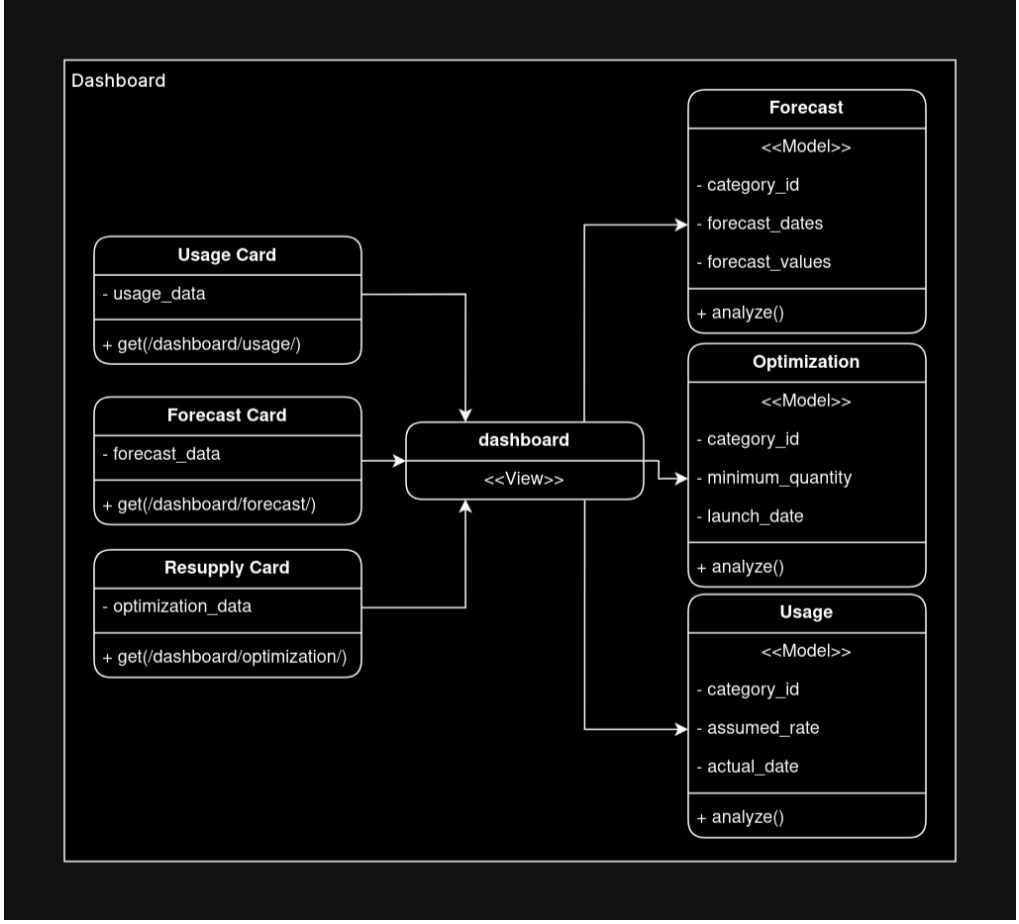
Shut down When a client ends a session, the system will ensure all active database connections used in client requests are closed and that any ongoing transactions are committed to the database.

4. Subsystem services

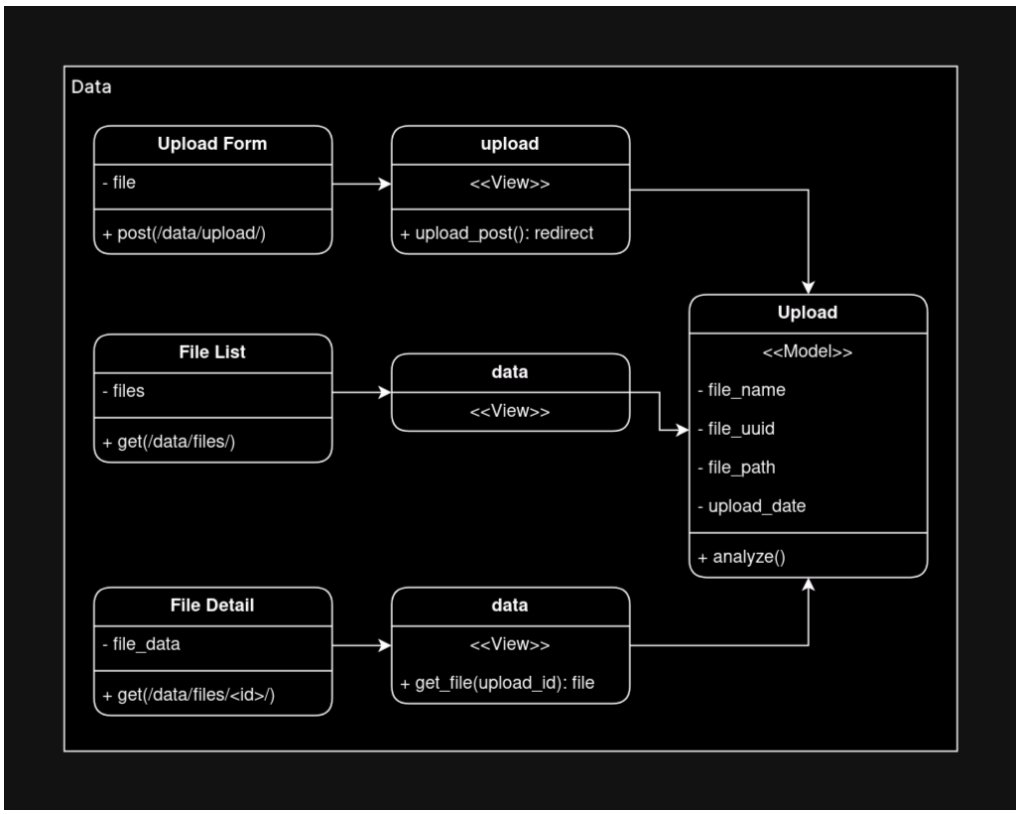
Accounts Subsystem



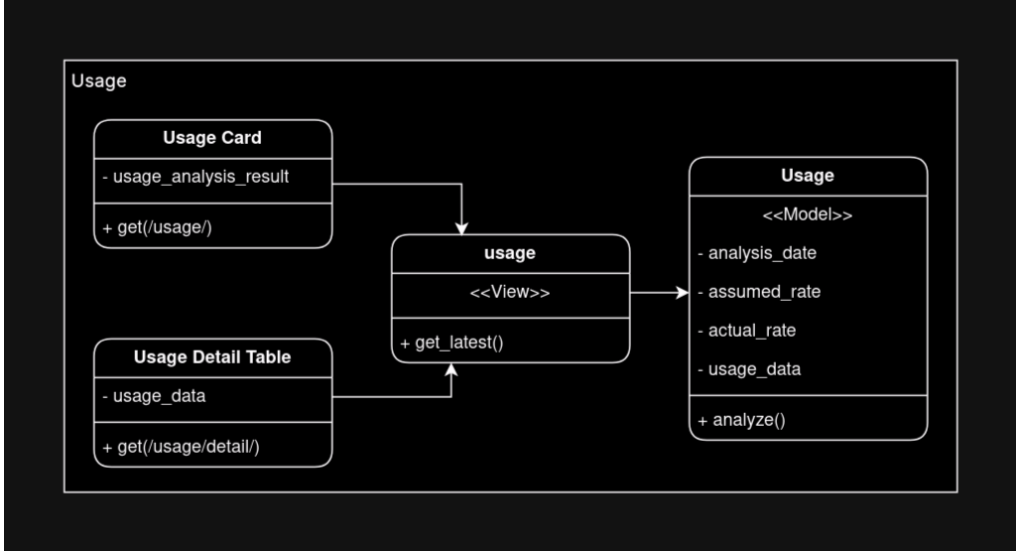
Dashboard Subsystem



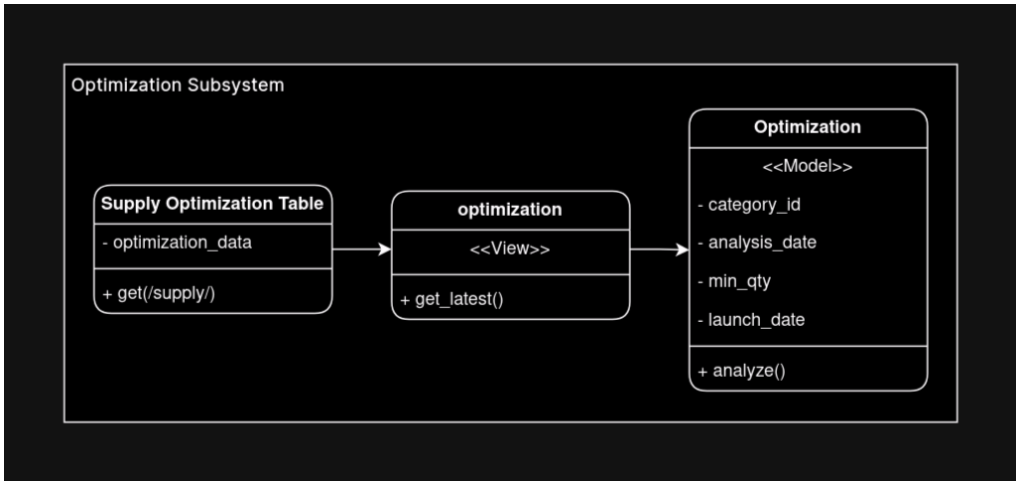
Data Subsystem



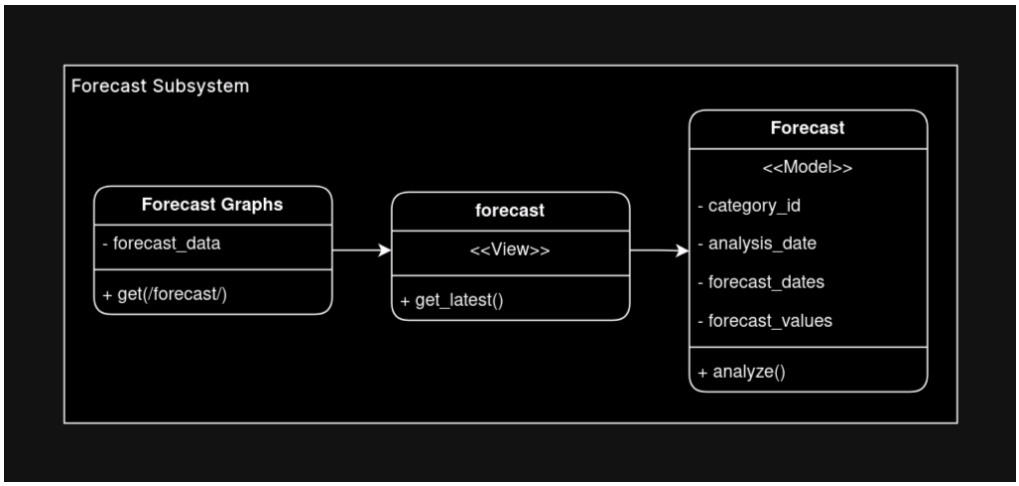
Usage Subsystem



Optimization Subsystem



Forecast Subsystem



5. Implementation

A high-level overview of each subsystem ("App") and the developers responsible for each.

```
.
├── accounts - Jeremy/Jeff
├── barrios - Project.Framework specific
├── dashboard - Jeremy/Jeff
├── data - Jeremy/Jeff
├── forecast - Jonathan/Jeff/Jeremy
├── home - Jeremy/Jeff
├── static - Static files
├── supply - Carlos/Jeff/Jeremy
├── templates - HTML Pages - Jeremy/Jeff
└── usage - Josue/Jeff/Jeremy
```

A detailed view of the project and the files within

```
.
├── accounts
│   ├── admin.py
│   ├── apps.py
│   ├── forms.py
│   ├── migrations
│   ├── models.py
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── barrios
│   ├── asgi.py
│   ├── middleware.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── dashboard
│   ├── admin.py
│   ├── apps.py
│   ├── tests.py
│   ├── urls.py
│   ├── views
│   └── views.py
├── data
│   ├── admin.py
│   ├── apps.py
│   ├── forms.py
│   ├── migrations
│   ├── models
│   ├── tests
│   ├── urls.py
│   ├── utils
│   └── views.py
├── forecast
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   ├── models.py
```

```
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── home
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   ├── models.py
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── manage.py
├── static
│   ├── favicon.svg
│   ├── scripts
│   └── style
├── supply
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   ├── models.py
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── templates
│   ├── assets
│   ├── base.html
│   ├── components
│   ├── dashboard_layout.html
│   ├── index.html
│   └── pages
└── usage
    ├── admin.py
    ├── apps.py
    ├── migrations
    ├── models.py
    ├── tests.py
    ├── urls.py
    └── views.py
```

Glossary

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

ASGI: Asynchronous Server Gateway Interface - An asynchronous calling convention for Python web servers

HTTP: Hypertext Transfer Protocol - The messaging and networking protocol used to request and deliver hypermedia on the web

HTML: Hypertext Markup Language - A language used to define hypertext documents for use by a web browser

DOM: The Document Object Model - A data structure representing an HTML document and all of the elements contained therein.

PostgreSQL: A popular relational database

Appendix

Project Roadmap