Iteration 2: Design

Jump to bottom

Jeff Caldwell edited this page now · 55 revisions

Barrios Technology - Forecasting and Prediction Modeling

Team Noname

Jonathan Morgan

Carlos Cardenas

Josue Lozano

Jeremy Miles

Jeff Caldwell

Vision Statement

Team Noname plans to provide Barrios Technology with a web application that gives users access to predictive modeling and forecasts based on past ISS consumables data. The application's interface will allow users to update past consumables data and view a dashboard that includes analysis of past outcomes and future predictions.

Features

- Interactive dashboard that will include:
  - A high-level overview of all current analyses and predictions
  - The ability to upload updated datasets for adjusted analyses and predictions
  - Detailed views of usage analysis, resupply predictions, and a detailed flight plan
- REST API that will include:
  - Controller-based routing that will:
    - Accept requests from the browser client
    - Use request parameters to call methods on analysis and prediction models
    - Coordinate data persistence through services or repositories
    - Respond with either structured JSON data or a hypermedia representation of the data's state

    - Models that will operate on data provided by the controller via a service or repository
    - Services/repositories whose only responsibility will be to provide an interface to the persistence infrastructure
- Database that will provide the ability to:
    - Store, update and retrieve user-provided datasets
    - Store, update and retrieve prediction and analysis results

Extended Features

- Ability to choose from multiple analysis methods

UI Sketches

BARRIOS

DASHBOARD

USAGE ANALYSIS

RESUPPLY PREDICTIONS

DETAILED FLIGHT PLAN

UPLOAD DATA

USAGE

BELOW PREDICTIONS

10%

USAGE ANALYSIS

NAME	MIN	MAX
O2	400 LBS	700 LBS
N2	100 LBS	200 LBS
H2O	500 L	700 L
FOOD	200 LBS	400 LBS
ACY INSERT	300	600
KTO	10	30
O2	400 LBS	700 LBS
N2	100 LBS	200 LBS
H2O	500 L	700 L
FOOD	200 LBS	400 LBS
ACY INSERT	300	600
KTO	10	30
O2	400 LBS	700 LBS
N2	100 LBS	200 LBS
H2O	500 L	700 L
FOOD	200 LBS	400 LBS
ACY INSERT	300	600
KTO	10	30

BARRIOS

DASHBOARD

USAGE ANALYSIS

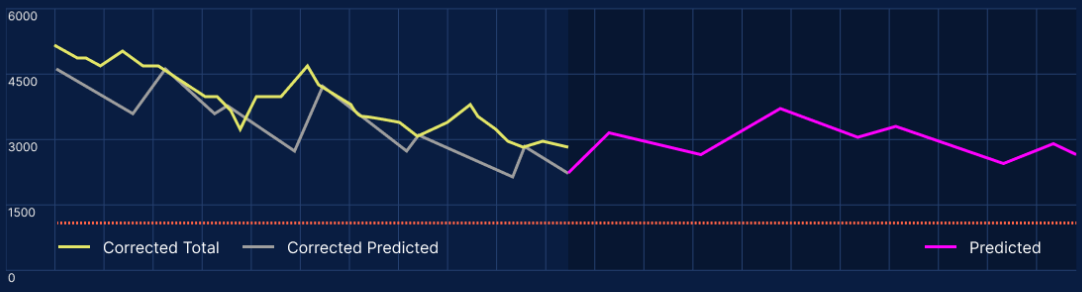
RESUPPLY PREDICTIONS

DETAILED FLIGHT PLAN

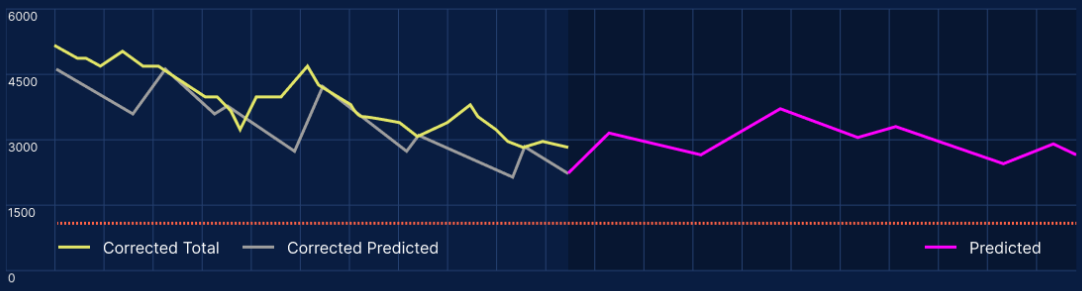
UPLOAD DATA

RESUPPLY PREDICTIONS

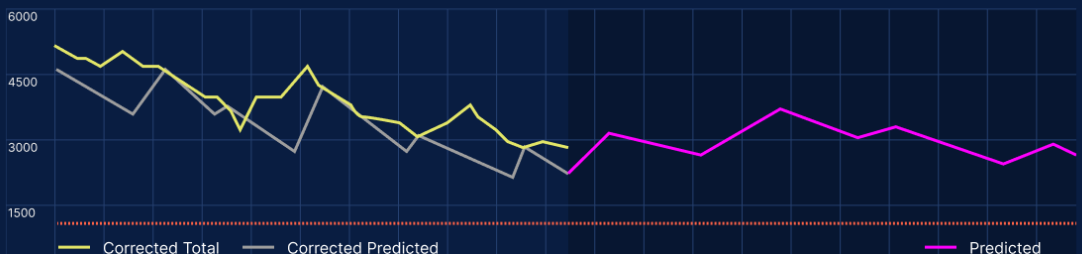
H2O



GASES



GASES



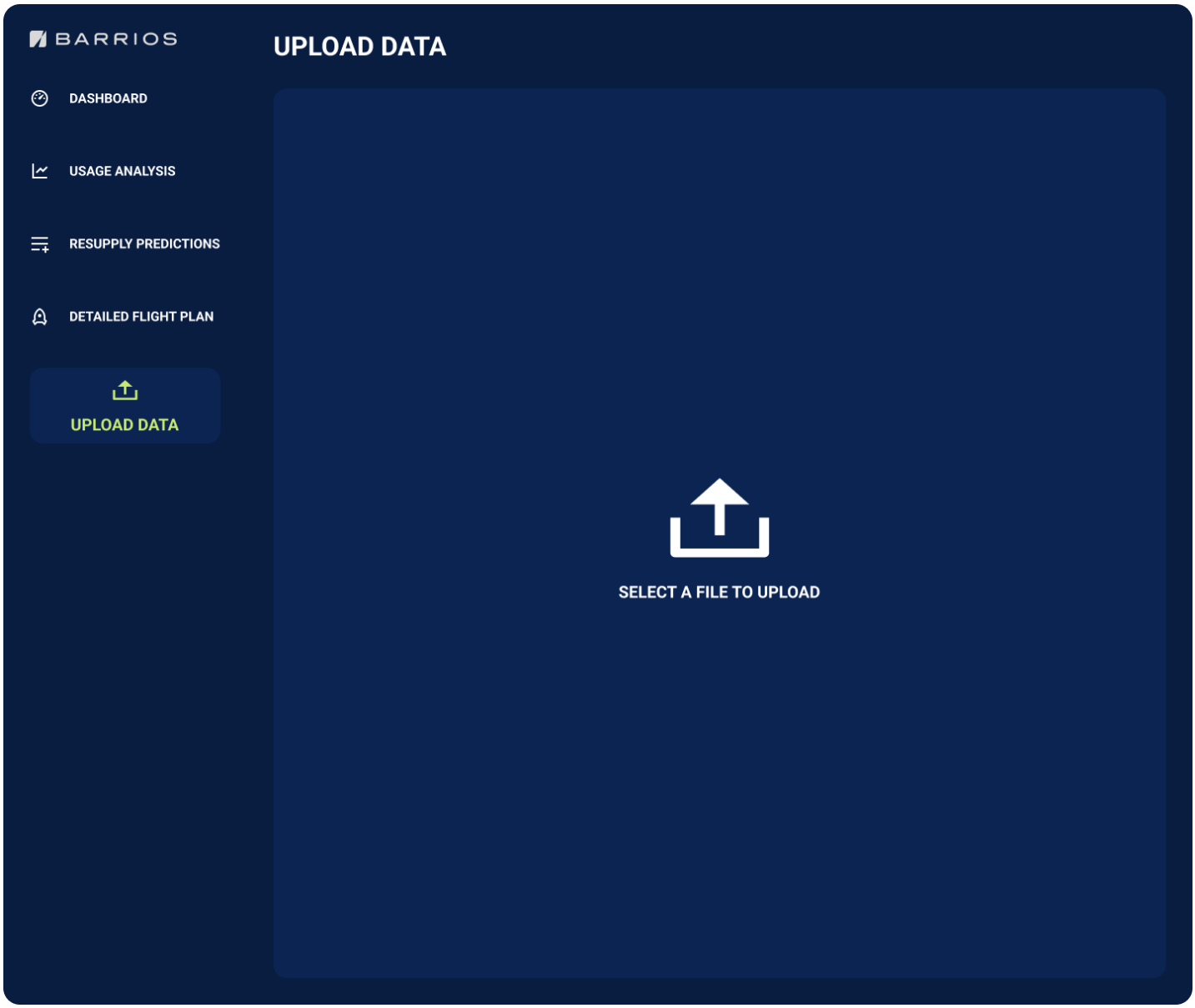
BARRIOS

- DASHBOARD
- USAGE ANALYSIS
- RESUPPLY PREDICTIONS
- DETAILED FLIGHT PLAN

UPLOAD DATA

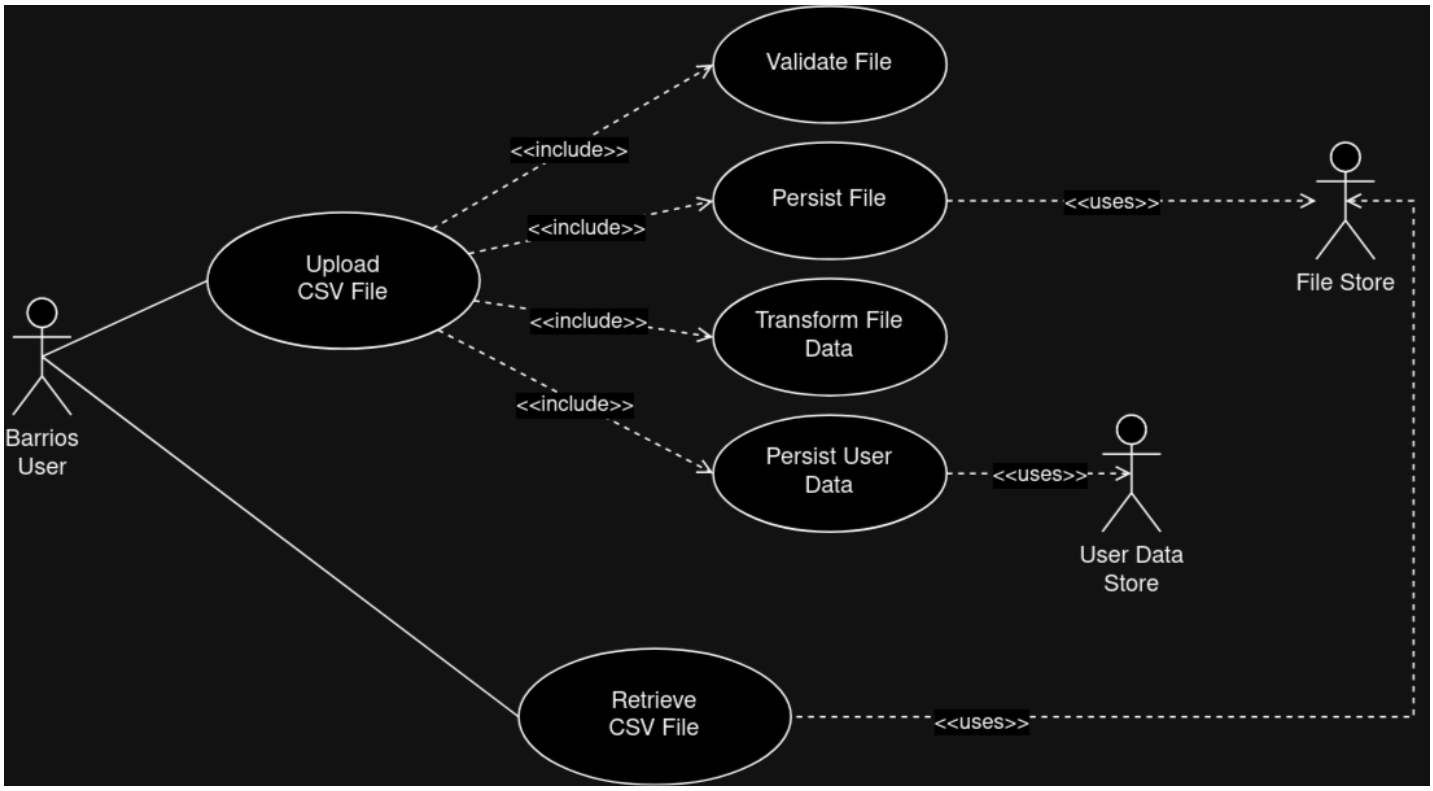
Flight Plan

NAME	TYPE	DATE
69 SOYUZ	UNDOCK	09/27/2023
U (MICROORGANISMS)	EVA	10/12/2023
U (RFG RETRIEVAL)	EVA	10/19/2023
R-61	EVA	10/25/2023
NG-19	UNDOCK	10/30/2023
SPX-29	LAUNCH	11/01/2023
SPX-29	DOCK	11/02/2023
84 PROGRESS	UNDOCK	11/29/2023
86 PROGRESS	DOCK	12/01/2023
69 SOYUZ	UNDOCK	09/27/2023
U (MICROORGANISMS)	EVA	10/12/2023
U (RFG RETRIEVAL)	EVA	10/19/2023
R-61	EVA	10/25/2023
NG-19	UNDOCK	10/30/2023
SPX-29	LAUNCH	11/01/2023
SPX-29	DOCK	11/02/2023
84 PROGRESS	UNDOCK	11/29/2023
86 PROGRESS	DOCK	12/01/2023
69 SOYUZ	UNDOCK	09/27/2023
U (MICROORGANISMS)	EVA	10/12/2023
U (RFG RETRIEVAL)	EVA	10/19/2023
R-61	EVA	10/25/2023
NG-19	UNDOCK	10/30/2023

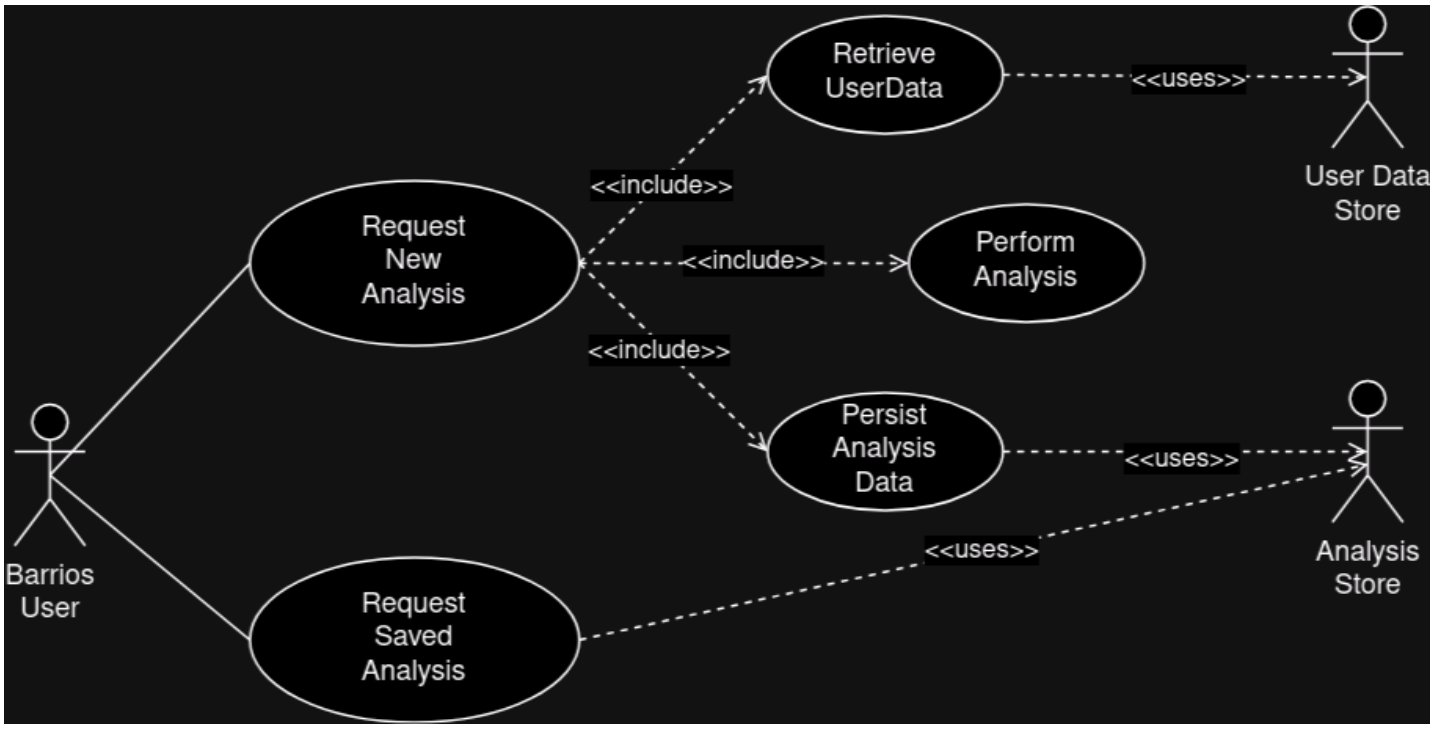


Use Cases

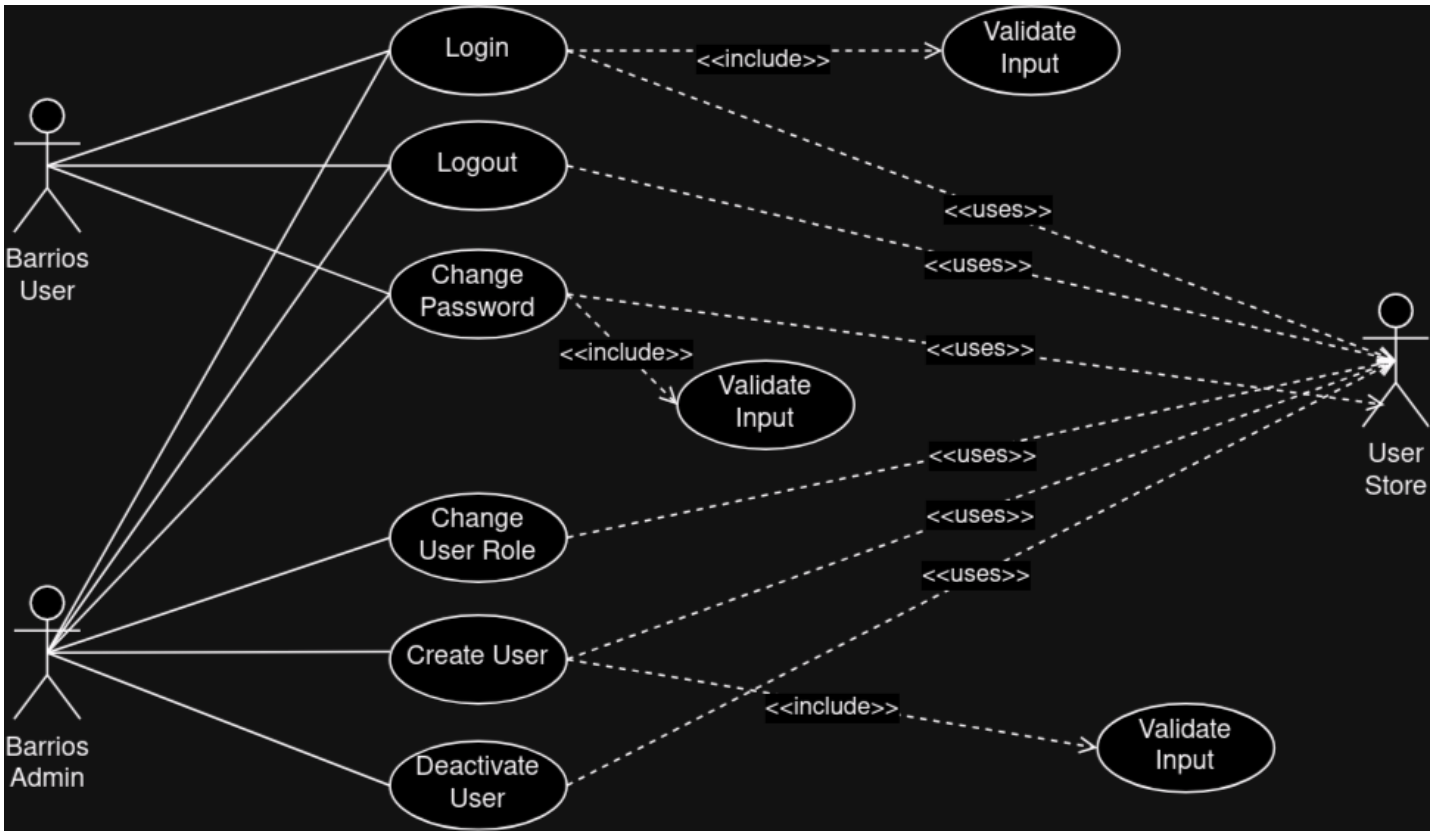
File Upload



Request Analysis



User Management



Key Use Cases

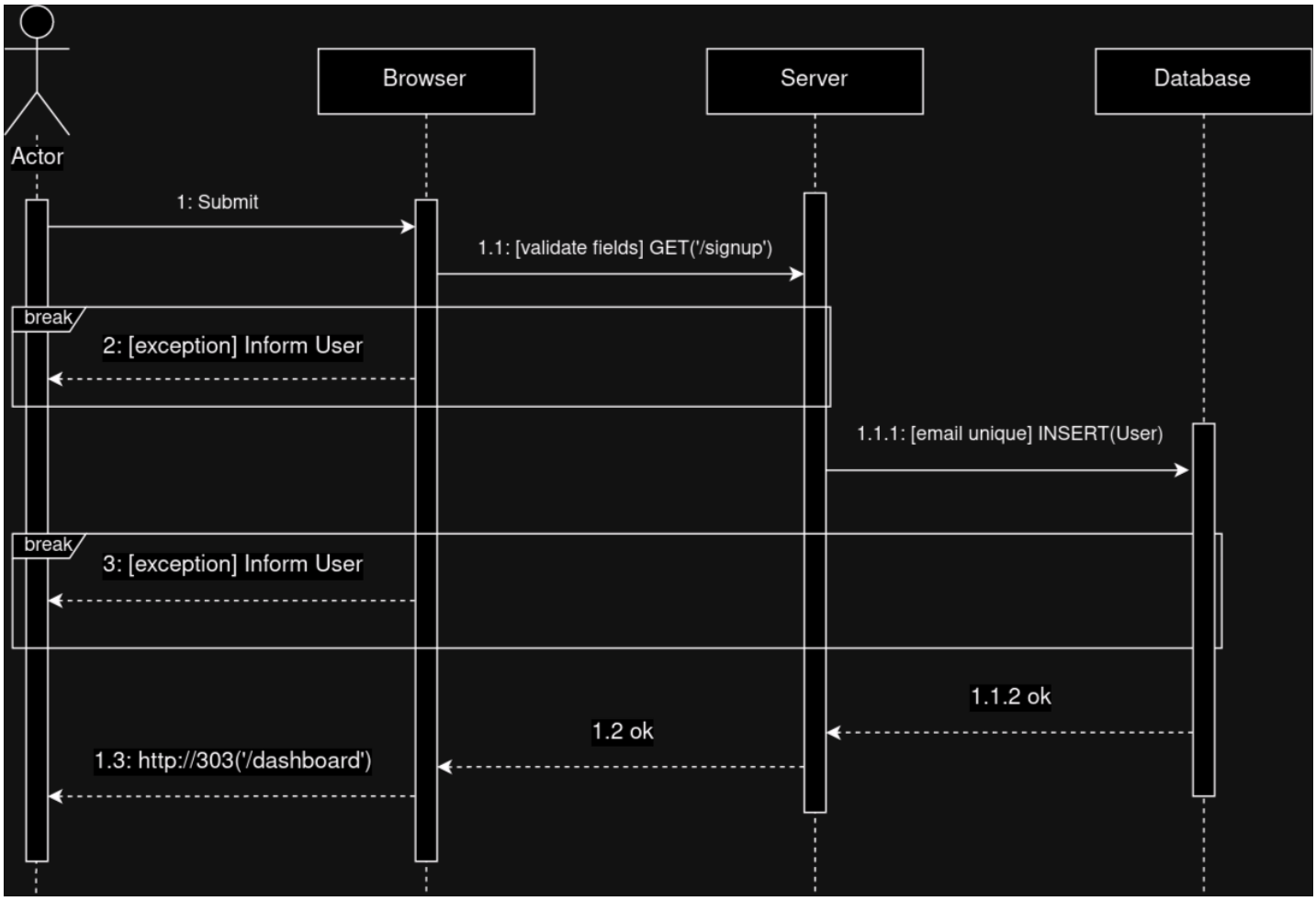
- Upload Barrios user data
- Retrieve stored user data
- View accuracy analysis, forecasting, and optimization with graphical charts
- Retrieve and view past analyses
- Manage users and user roles

Sequence Diagrams

Data Upload

[![Data Upload Sequence Diagram](https://github.com/4306-team-noname/barrios/blob/4f1b36c123e052783b7a8eb5381e305bd7d12efd/assets/Iteration2/Barrios_UploadSequence.drawio.png)](https://github.com/4306-team-noname/barrios/blob/4f1b36c123e052783b7a8eb5381e305bd7d12efd/assets/Iteration2/Barrios_UploadSequence.drawio.png)

User Signup



Architecture

The Barrios analytics application will follow a server/client model with a Hypermedia/REST API (i.e., server endpoints respond with HTML fragments rather than JSON. This approach should simplify the client development by restricting the application state to the server. More advanced, "single-page application" style interactivity will be achieved by using HTMX.

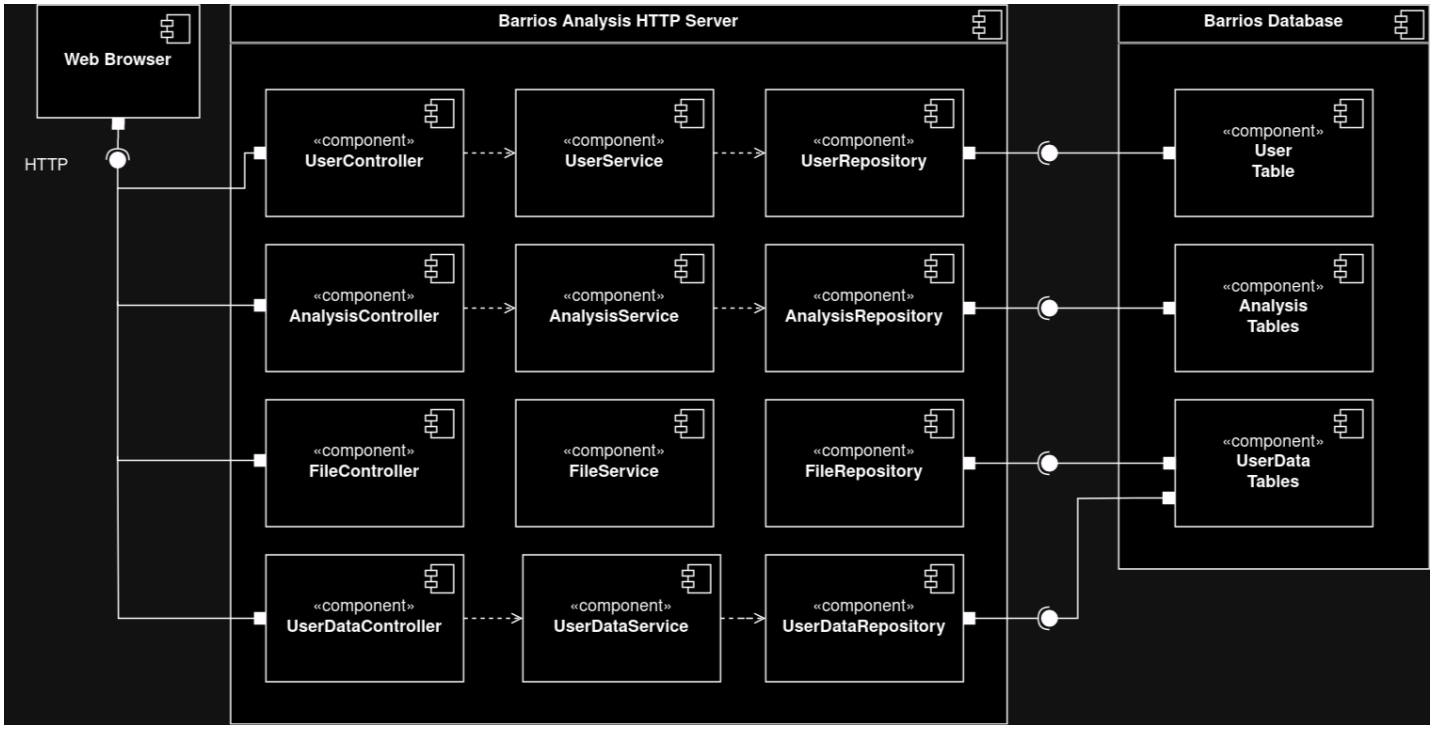
The server will be implemented using the emmett framework. Emmett is an HTTP server framework written in Python that will allow the team to organize the code in whatever form necessary without the need to adhere to a framework's choice of architecture. While this may require some additional code, we hope that the effort saved by reducing front-end complexity can be devoted to crafting the server architecture.

The server's architecture will be modeled following a mixed approach that follows aspects of model-view-controller (MVC) and some aspects of a layered approach. The controllers will be the outer or topmost layer, which will respond to HTTP messages sent to various routes as required by UI functionality. Those controllers will call methods on services to coordinate data persistence/retrieval from repositories and instantiation of and communication with models.

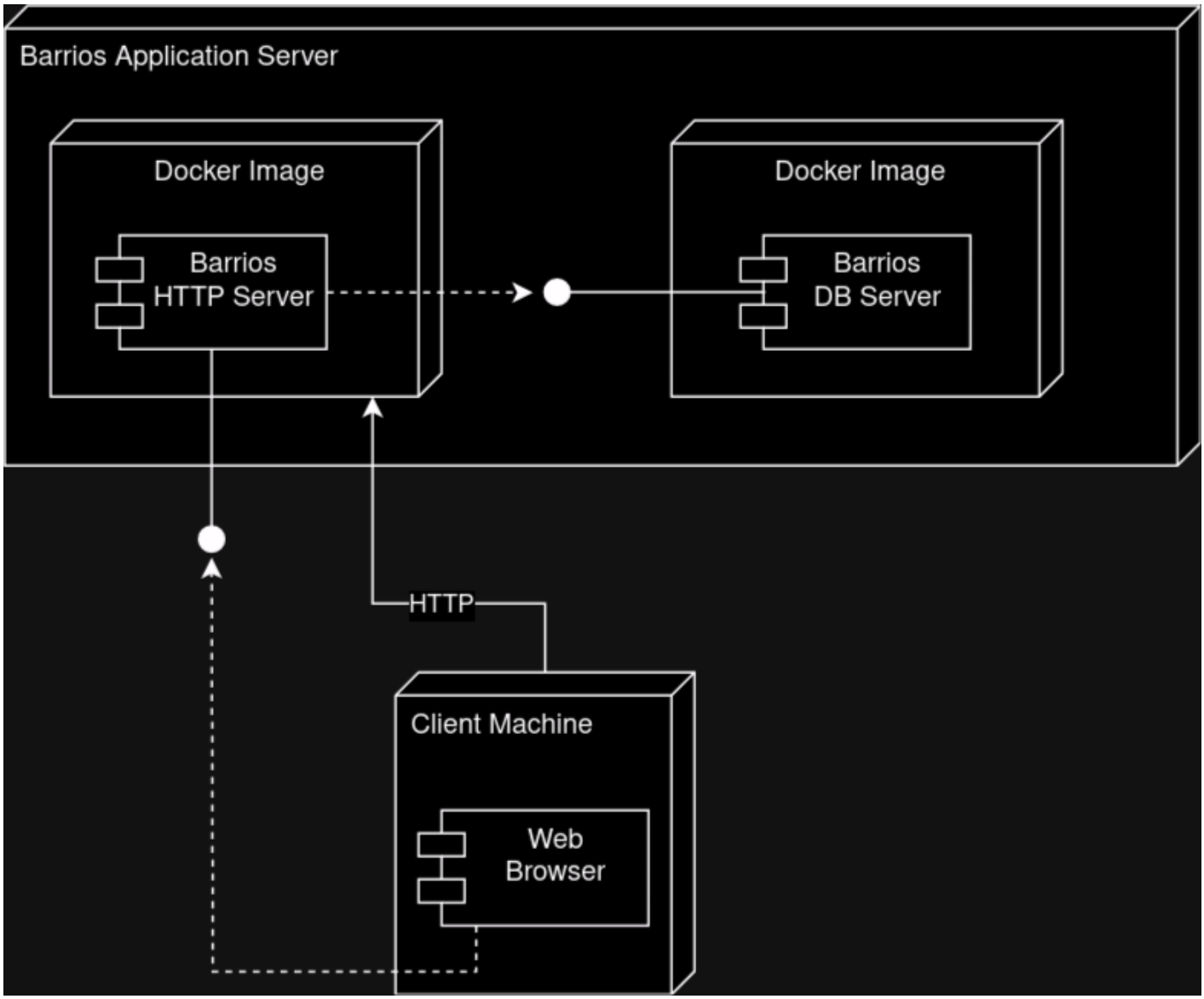
Models will either represent optimized, forecast, or analysis data or act as the implementation of various analytics methods. Finally, at the infrastructure level, repositories will act as interfaces to databases — implementations of which will serve to communicate with their specific database drivers — allowing the developers to accommodate different database engines, depending on the client's needs or the current needs of the development process.

Pandas, Dask, and Numpy will be used to explore, analyze and clean user data and to extract the necessary values for analysis. Services may also use these libraries to transform persisted data for modeling functions.

Architecture Overview



Deployment Diagram

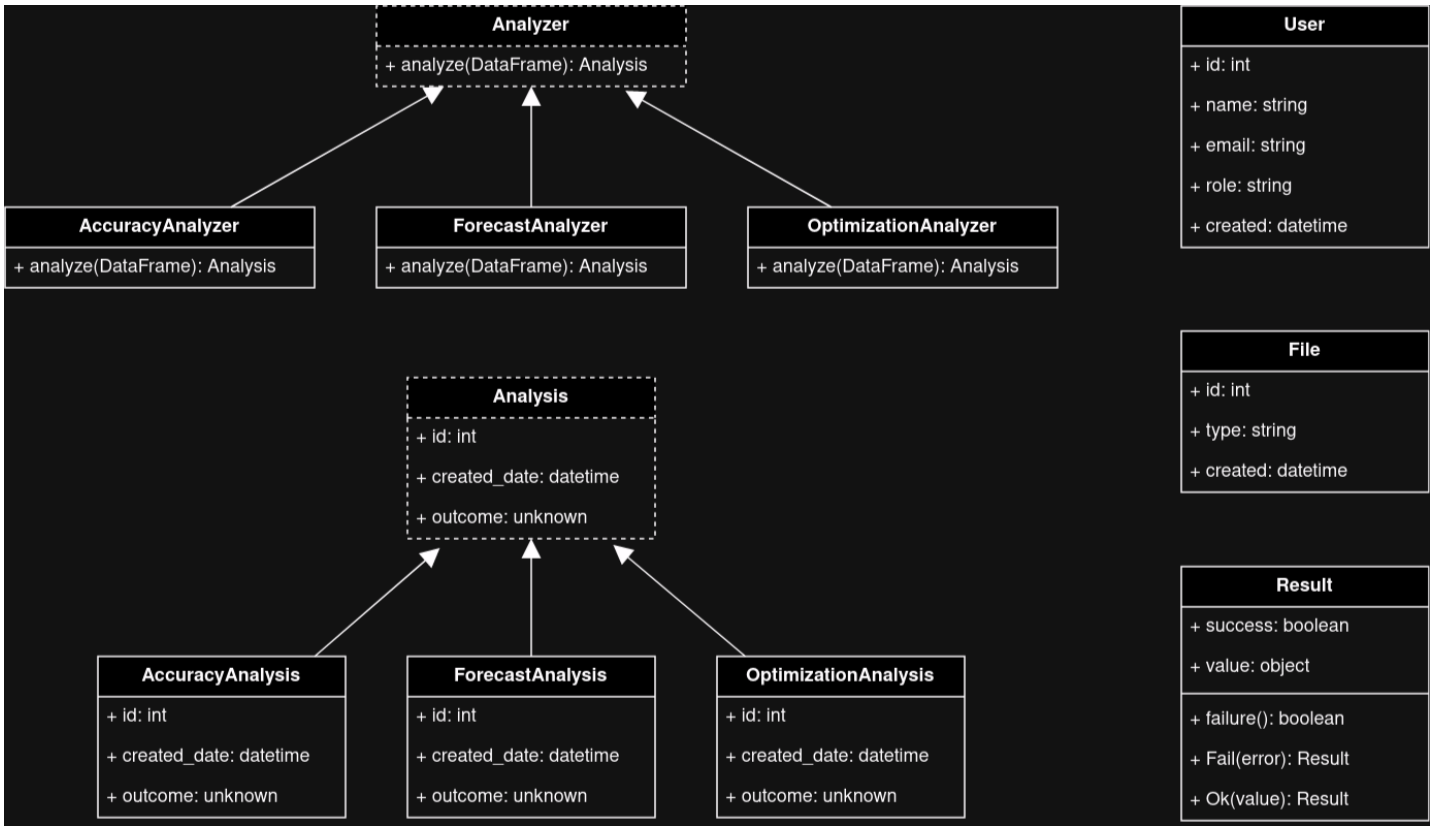


Class Diagrams

Domain Classes

Domain classes are the base components of the system. Most of them, for example, `User` and `File`, represent application data and have no methods. Some, such as the `Analyzer` classes, only exist to run analyses of a specific type and generate various `Analysis` classes. The `Result` class represents an operation's outcome and will assist in consistent error handling across the application.

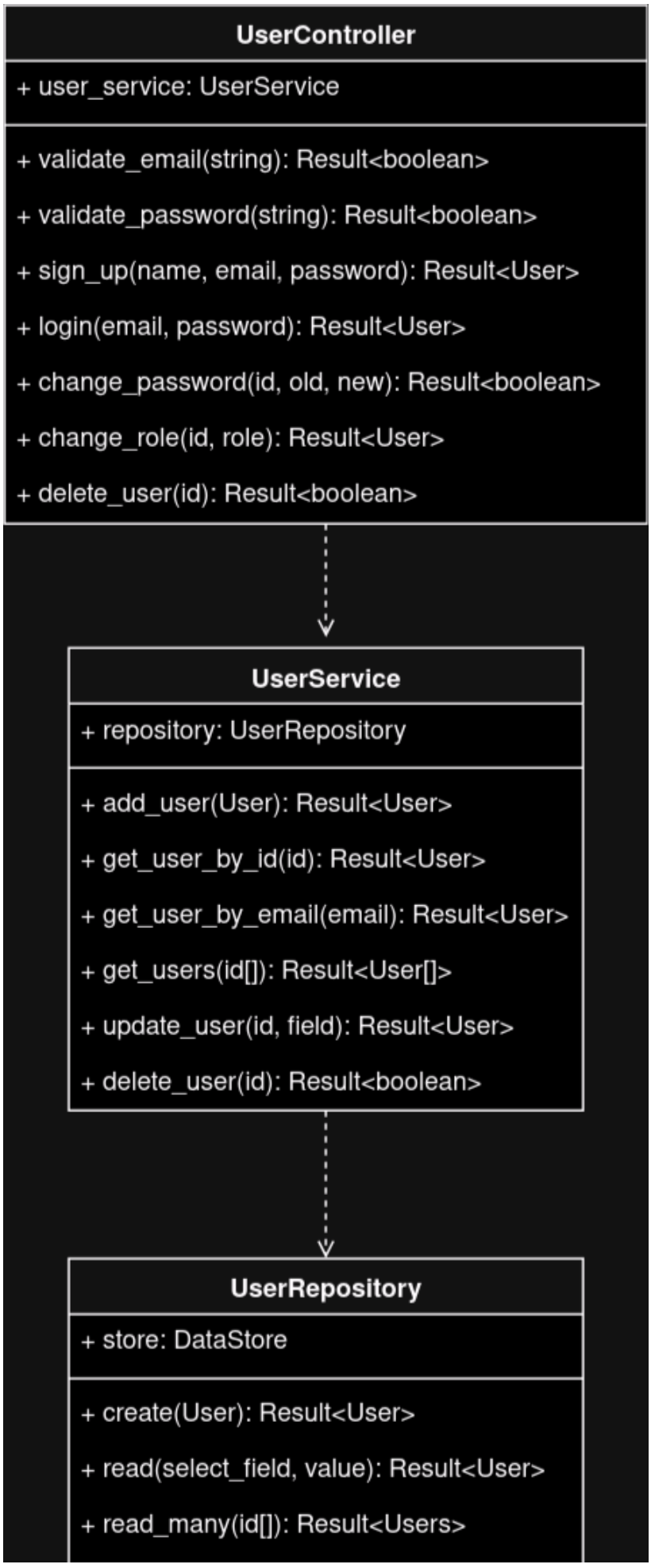
Both `Analysis` and `Analyzer` classes are interfaces to be realized by the implementations shown in the diagram.

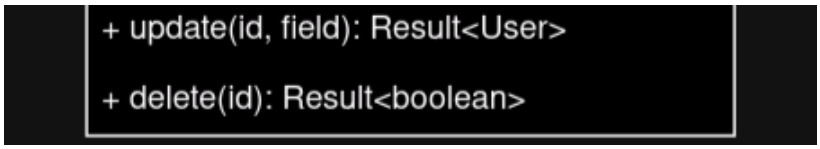


Service Classes

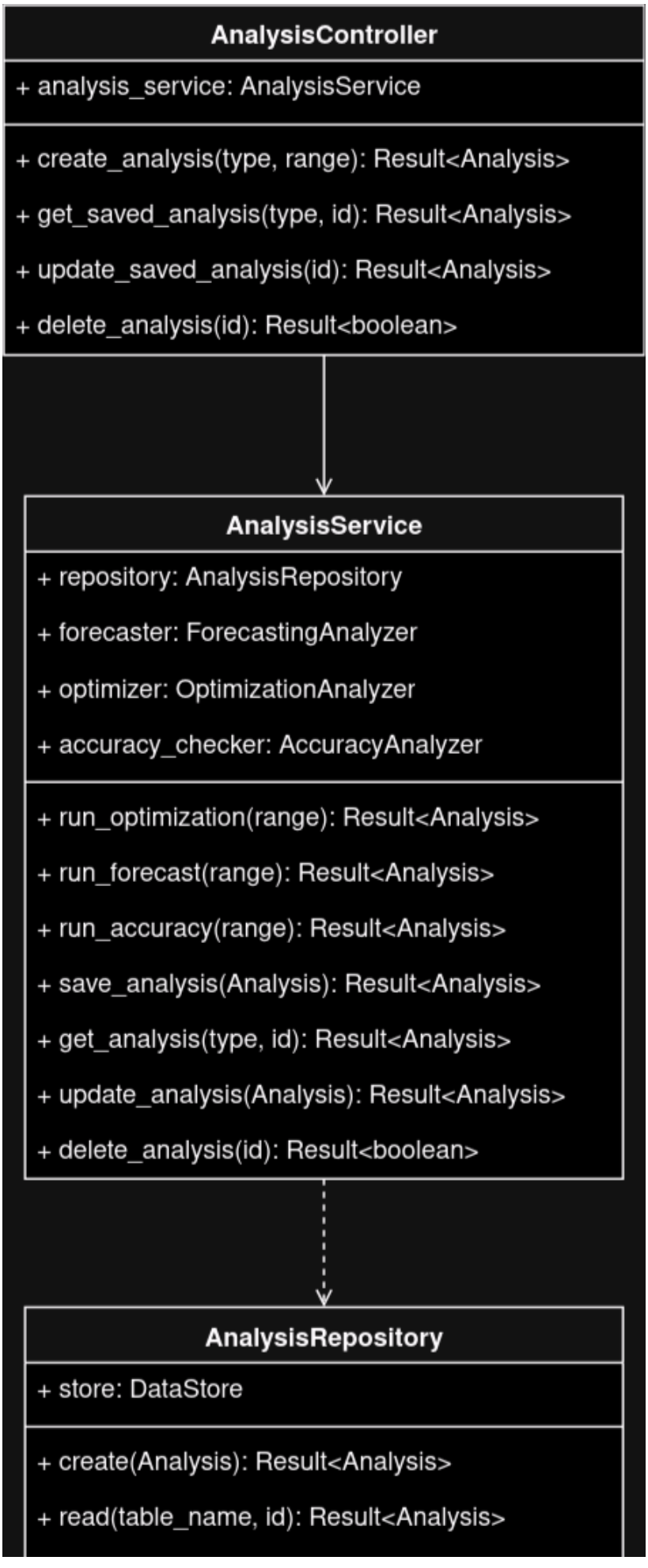
Service classes, such as controllers, services, and repositories, will be implemented using basic dependency injection. As such, their relationships are modeled as dependencies.

User Service


UserController
+ user_service: UserService
+ validate_email(string): Result<boolean>
+ validate_password(string): Result<boolean>
+ sign_up(name, email, password): Result<User>
+ login(email, password): Result<User>
+ change_password(id, old, new): Result<boolean>
+ change_role(id, role): Result<User>
+ delete_user(id): Result<boolean>
UserService
+ repository: UserRepository
+ add_user(User): Result<User>
+ get_user_by_id(id): Result<User>
+ get_user_by_email(email): Result<User>
+ get_users(id[]): Result<User[]>
+ update_user(id, field): Result<User>
+ delete_user(id): Result<boolean>
UserRepository
+ store: DataStore
+ create(User): Result<User>
+ read(select_field, value): Result<User>
+ read_many(id[]): Result<Users>



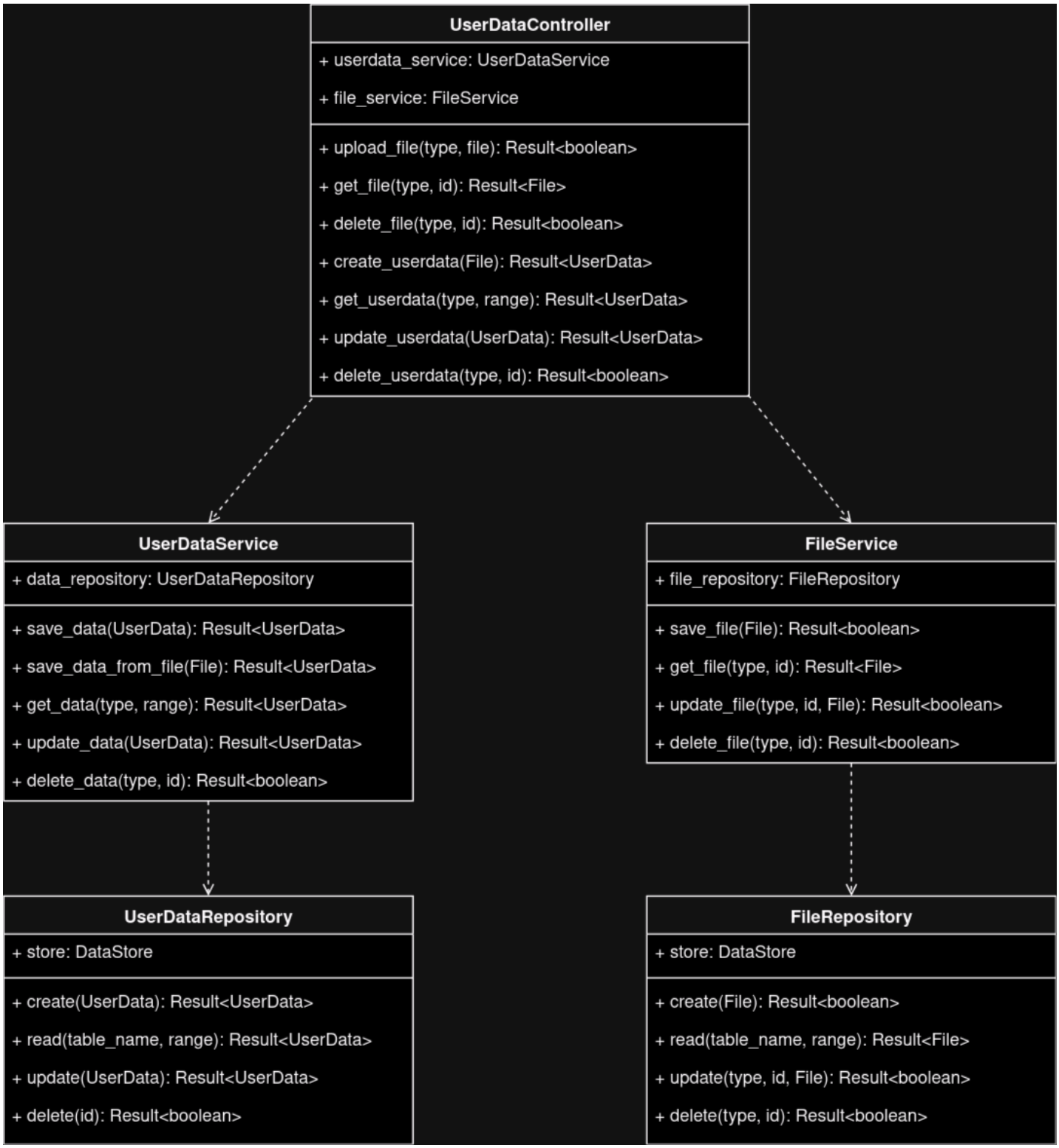
Analysis Service


AnalysisController
+ analysis_service: AnalysisService
+ create_analysis(type, range): Result<Analysis>
+ get_saved_analysis(type, id): Result<Analysis>
+ update_saved_analysis(id): Result<Analysis>
+ delete_analysis(id): Result<boolean>
AnalysisService
+ repository: AnalysisRepository
+ forecaster: ForecastingAnalyzer
+ optimizer: OptimizationAnalyzer
+ accuracy_checker: AccuracyAnalyzer
+ run_optimization(range): Result<Analysis>
+ run_forecast(range): Result<Analysis>
+ run_accuracy(range): Result<Analysis>
+ save_analysis(Analysis): Result<Analysis>
+ get_analysis(type, id): Result<Analysis>
+ update_analysis(Analysis): Result<Analysis>
+ delete_analysis(id): Result<boolean>
AnalysisRepository
+ store: DataStore
+ create(Analysis): Result<Analysis>
+ read(table_name, id): Result<Analysis>

+ update(Analysis): Result<Analysis>

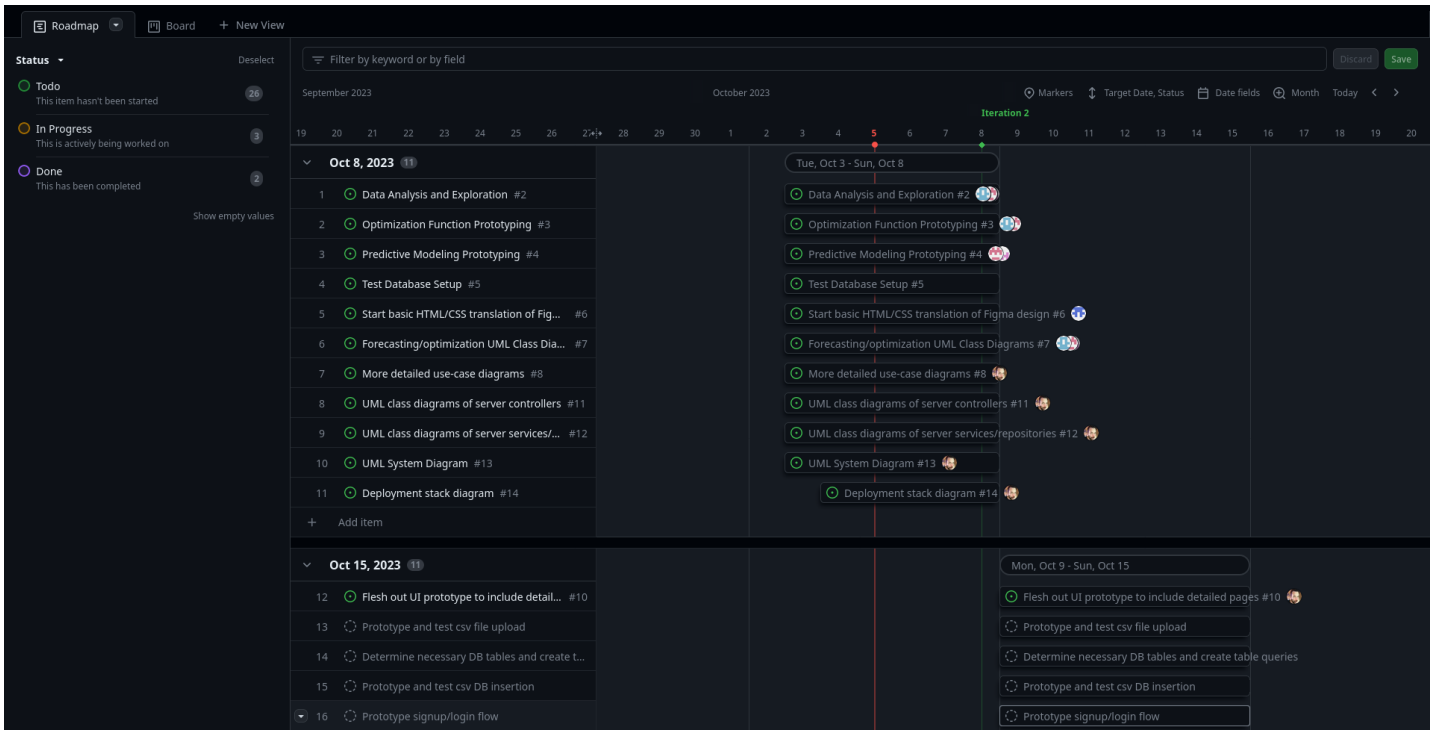
+ delete(id): Result<boolean>

Data Service



Project Timeline

[GitHub Project Roadmap]



Pages 3

Find a page...

- **Home**
- **Iteration 1: Requirements**
- **Iteration 2: Design**
  - Barrios Technology - Forecasting and Prediction Modeling
  - Team Noname
  - Vision Statement
  - Features
    - Extended Features
  - UI Sketches
  - Use Cases
    - File Upload
    - Request Analysis
    - User Management
  - Key Use Cases
  - Sequence Diagrams
    - Data Upload
    - User Signup

Clone this wiki locally

```
https://github.com/4306-team-noname/barrios.wiki.git
```