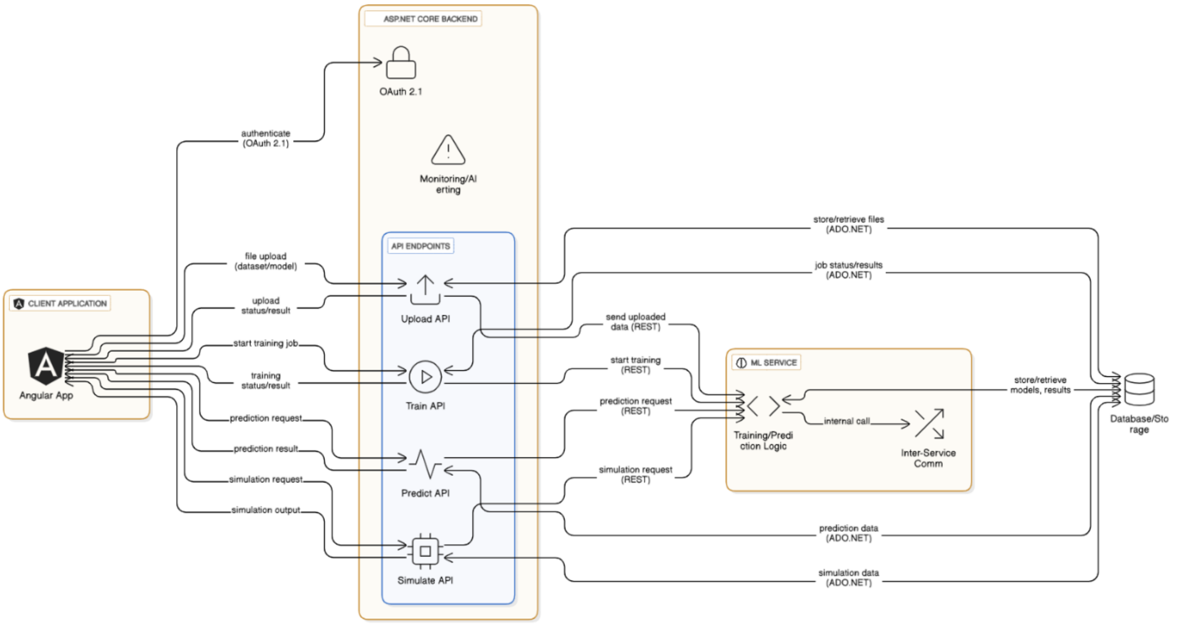
I. System Architecture



The project's system architecture is a multi-service, containerized application designed to handle a full machine learning pipeline. It is composed of three main services: a frontend, a backend API gateway, and a Python-based ML service.

Components and Services

1. **Frontend (Angular)**:

- **Role**: The user-facing web interface. It allows users to upload datasets, define date ranges, start model training, and view simulation results.
- **Technology**: Built with Angular as a standalone application.
- **Deployment**: A multi-stage Docker build process creates a static HTML/CSS/JS bundle, which is then served by an Nginx web server.
- **Communication**: The frontend communicates exclusively with the backend API gateway using HTTP requests.

2. **Backend API (ASP.NET Core)**:

- **Role**: Acts as the central API gateway and orchestrator. It receives requests from the frontend, handles data ingestion and preprocessing, and forwards requests to the ML service.
- **Technology**: Built with ASP.NET Core.
- **Data Handling**: Reads and writes a processed_data.csv file, which is shared via a Docker volume. It adds a synthetic timestamp to each data row.
- **Communication**: Routes POST requests for model training and simulation directly to the ML service. It also has endpoints for uploading and validating data.

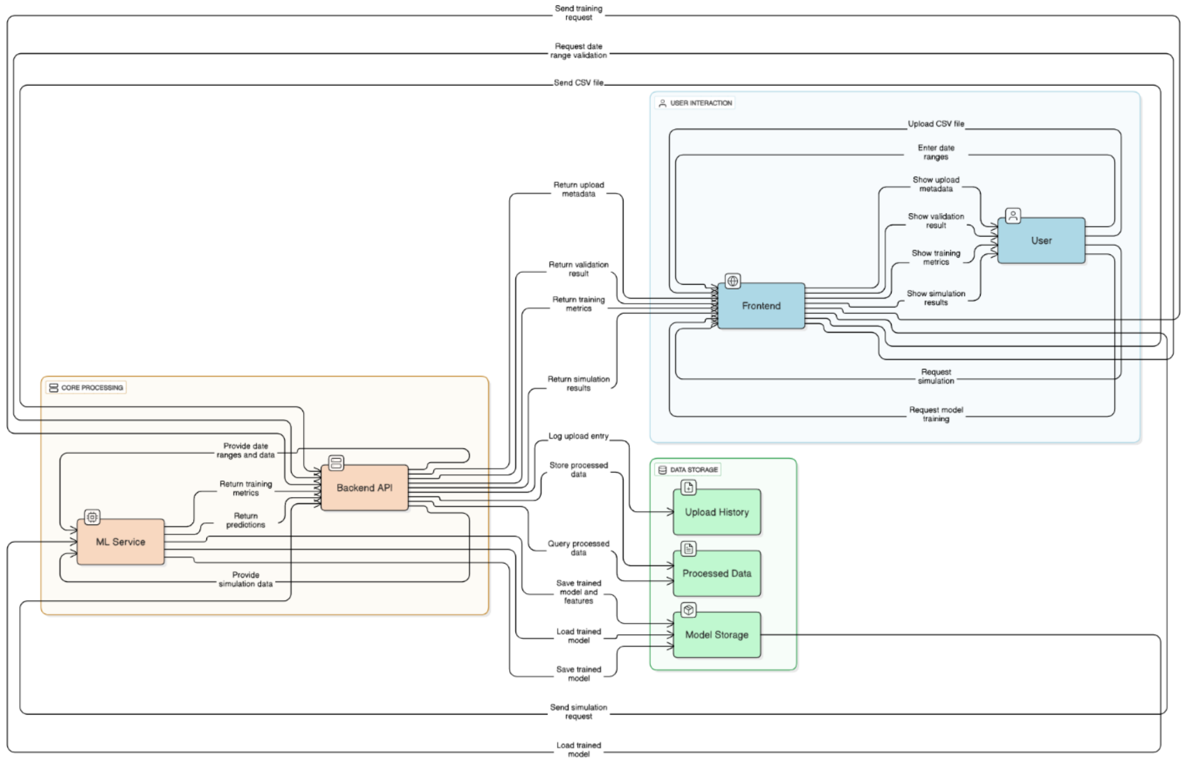
3. **ML Service (Python/FastAPI)**:

- **Role**: The core machine learning engine. It receives training and simulation requests from the backend, trains an XGBoost model, and performs predictions.
- **Technology**: A Python application built with the FastAPI framework.
- **Machine Learning**: Uses pandas for data manipulation, xgboost for model training and prediction, and joblib for saving and loading the trained model.
- **Deployment**: Runs as a separate Docker container with all its dependencies installed.

How the Services Interact

- **File Upload**: The user uploads a CSV file through the **Frontend**.
- **Data Processing**: The **Frontend** sends this file to the **Backend**. The Backend processes the file, adds a synthetic timestamp, and saves it to a shared data/processed_data.csv file.
- **Model Training**: The user specifies date ranges and clicks "Train Model." The **Frontend** sends a request to the **Backend**, which forwards the request and the date ranges to the **ML Service**.
- **Model Simulation**: The **ML Service** loads the shared processed_data.csv file, trains a new model on the specified date range, and saves the model to data/model.joblib.
- **Live Predictions**: The **Backend** then calls the /simulate endpoint on the **ML Service**, which loads the newly trained model and runs a real-time prediction simulation on the data from the Simulation Period.

II. Data Flow



1. Data Ingestion (File Upload)

The data flow begins when a user uploads a CSV file via the Frontend.

1. **User to Frontend**: A user selects and uploads a .csv file.
2. **Frontend to Backend**: The Frontend sends the raw CSV file to the Backend's upload-dataset endpoint as a multipart/form-data request.
3. **Backend Processing**: The Backend receives the file, adds a synthetic timestamp, and performs initial preprocessing.
4. **Backend to Disk**: The preprocessed data is saved to a shared volume as processed_data.csv. This file is accessible to both the Backend and the ML Service.

2. Model Training

After the data is processed, the user defines date ranges to initiate the model training process.

1. **User to Frontend**: The user specifies training, testing, and simulation date ranges in the Frontend UI.
2. **Frontend to Backend**: The Frontend sends the selected date ranges to the Backend's validate-ranges endpoint as a JSON payload.
3. **Backend to ML Service**: The Backend validates the date ranges and then forwards a POST request to the ML Service's train-model endpoint. The payload includes the training and testing date ranges.

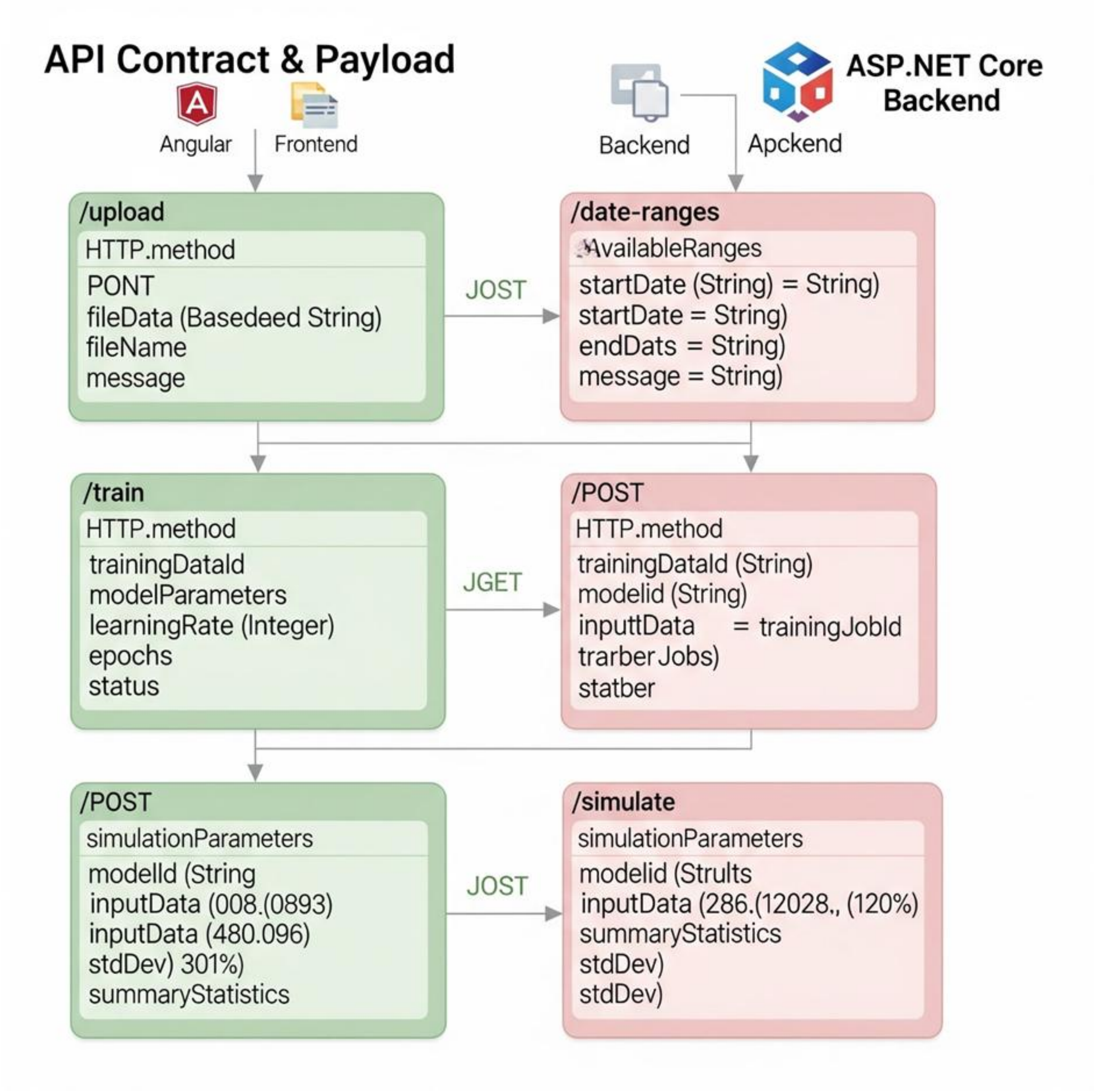
4. **ML Service Processing**: The ML Service reads the processed_data.csv file from the shared volume, trains an XGBoost model using the specified data, and evaluates its performance.
5. **ML Service to Disk**: The trained model is serialized and saved to the shared volume as model.joblib.

3. Model Simulation

Once a model is trained, the system can run a simulation on the specified data to generate predictions.

1. **Backend to ML Service**: The Backend sends a POST request to the ML Service's /simulate endpoint. The payload contains the simulation date range.
2. **ML Service Processing**: The ML Service loads the model.joblib file and uses it to predict on the data within the simulation date range from processed_data.csv.
3. **ML Service to Backend**: The ML Service returns the simulation results and predictions to the Backend as a JSON response.
4. **Backend to Frontend**: The Backend forwards the simulation results to the Frontend for display.
5. **Frontend to User**: The user views the simulation results in the UI.

III. API Contract & PayLoad Structure



1. **Upload Dataset**

This endpoint has three primary functions:

- ● File Validation: It checks if the uploaded file is in the correct CSV format.
- ● Data Augmentation: It adds a synthetic timestamp to each record in the dataset.
- ● Metadata Generation: It extracts key information like the total number of records, columns, and the date range, which is then sent back to the frontend.

.

- ● Endpoint: /api/dataset/upload
- ● Method: POST

- Request Payload: multipart/form-data

o Field: file (type: file, format: .csv) - The CSV file to be uploaded.

- Response Payload: application/json

JSON

```
{
  "fileName": "example_data.csv",
  "totalRecords": 1000,
  "totalColumns": 15,
  "passRate": 0.985,
  "dateRange": "2023-01-01 to 2024-01-01"
}
```

2. Date Ranges

- Date Range Validation: It ensures that the date ranges you select are sequential and non-overlapping.
- Dataset Boundary Check: It verifies that your chosen date ranges for training, testing, and simulation all fall within the overall date range of the uploaded dataset.
- Record Count Summary: After a successful validation, it returns a summary of how many records fall within each of your defined ranges, giving you a clear idea of the size of your training, testing, and simulation sets

- Endpoint: /api/dataset/validate-ranges
- Method: POST
- Request Payload: application/json

JSON

```
{
  "training": {
    "start": "2023-01-01",
    "end": "2023-09-30"
  },
  "testing": {
    "start": "2023-10-01",
    "end": "2023-12-31"
  },
  "simulation": {
    "start": "2024-01-01",
    "end": "2024-01-31"
  },
  "datasetMetadata": {
    "fileName": "example_data.csv",
    "totalRecords": 1000,
    "totalColumns": 15,
    "passRate": 0.985,
    "dateRange": "2023-01-01 to 2024-01-01"
  }
}
```

- Response Payload: application/json

JSON

```
{
  "isValid": true,
  "message": "Date ranges are valid.",
  "trainingRanges": {
    "trainStart": "2023-01-01",
    "trainEnd": "2023-09-30",
    "testStart": "2023-10-01",
    "testEnd": "2023-12-31"
  },
  "simulationRange": {
    "simulationStart": "2024-01-01",
    "simulationEnd": "2024-01-31"
  }
}
```

3. Model Training

This endpoint initiates the model training process on the backend. This is a crucial step where the application uses the validated date ranges to:

- Extract Data: It filters the large dataset to get only the records within the defined training and testing periods.
- Train the Model: It uses this filtered data to train a new machine learning model (e.g., an XGBoost classifier).
- Evaluate Performance: It calculates key metrics like accuracy, precision, and recall on the test data to assess the model's performance

- Endpoint: /api/ml/train-model
- Method: POST
- Request Payload: application/json

JSON

```
{
  "trainingRanges": {
    "trainStart": "2023-01-01",
    "trainEnd": "2023-09-30",
    "testStart": "2023-10-01",
    "testEnd": "2023-12-31"
  }
}
```

- Response Payload: application/json

JSON

```
{
  "trainingStatus": "completed",
  "modelId": "model_123456",
  "metrics": {
    "accuracy": 0.95,
    "precision": 0.92,
    "recall": 0.97
  }
}
```

4. Model Simulation

This endpoint performs a live prediction simulation on the test data. This is the final step in the machine learning pipeline, where the trained model is put to use to make predictions on unseen data. The key functions are:

- **Load Model and Data**: It loads the previously trained model (model.joblib) and the processed dataset from the shared volume (processed_data.csv).
- **Filter Data**: It filters the dataset to include only the records within the user-defined simulation date range.
- **Live Prediction**: For each record in the simulation data, it uses the trained model to predict the Response and calculates a confidence score.
- **Stream Results**: The results are formatted as a stream of individual predictions, providing a live feed of the model's output.
- **Endpoint**: /api/ml/simulate
- **Method**: POST
- **Request Payload**: application/json

```
{
 "simulationRange": {
  "simulationStart": "2024-01-01",
  "simulationEnd": "2024-01-31"
 }
}
```

- **Response Payload**: application/json

```
{
 "status": "success",
 "results": [
  {
   "timestamp": "2024-01-01T00:00:00Z",
   "id": 1,
   "prediction": 1,
   "confidence": 0.98,
   "parameters": {
    "temperature": 25,
    "pressure": 1000
   }
  },
    {
      "timestamp": "2024-01-01T00:00:01Z",
      "id": 2,
      "prediction": 0,
      "confidence": 0.92,
      "parameters": {
        "temperature": 26,
        "pressure": 1001
      }
    }
  ]
}
```