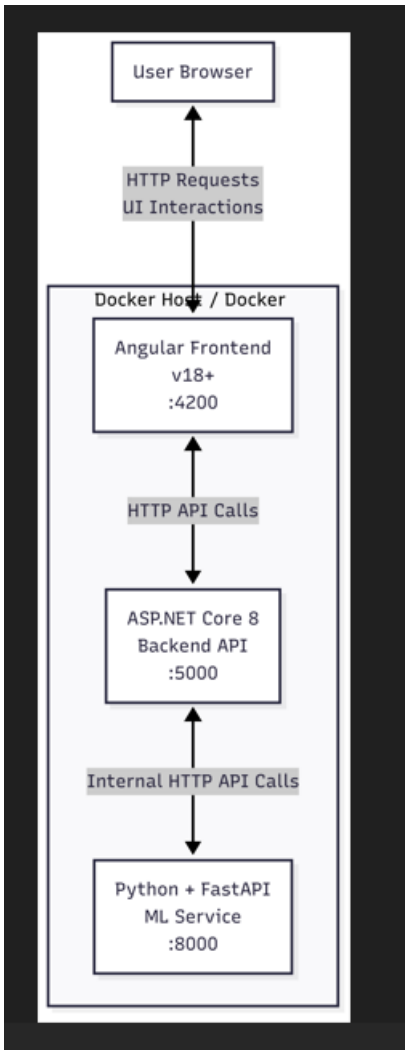
ABB Hackathon Assessment Team 7



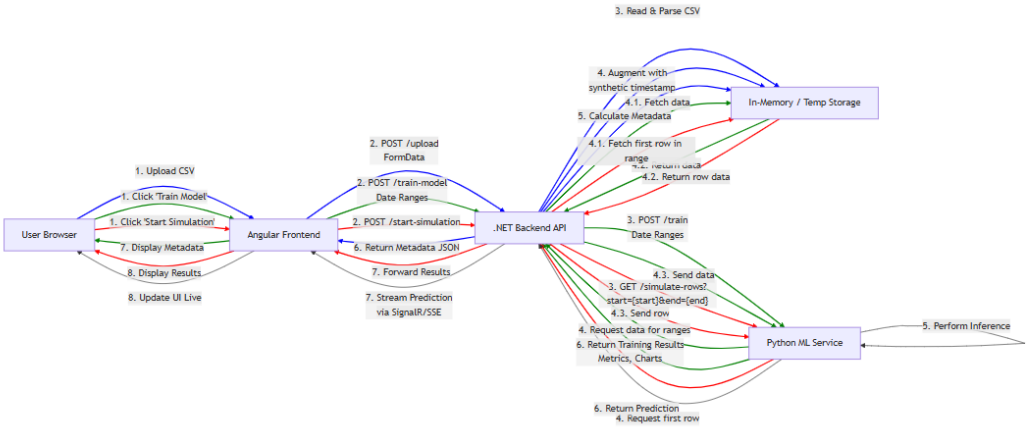
1. System Architecture

Diagram Description:

The provided diagram illustrates the high-level, containerized system architecture for the Intellinspect application. The entire system runs within a single Docker host environment, orchestrated by Docker Compose. It consists of three core microservices:

1. **Angular Frontend Service:** Serves the user interface on port 4200. Its sole responsibility is to present the UI and communicate with the backend API via HTTP requests for all data operations and user commands.
2. **ASP.NET Core Backend API Service:** Acts as the central hub and API gateway for the application, running on port 5000. It handles business logic, user input validation, file processing, and data management. It exposes a RESTful API for the frontend and, in turn, acts as a client to the ML service.
3. **Python ML Service:** A dedicated microservice built with FastAPI, running on port 8000. It is solely responsible for all machine learning operations, including model training and real-time inference. It receives commands from the backend API.

The primary communication flow is linear: the **User's Browser** interacts with the **Angular Frontend**, which makes calls to the **.NET Backend API**, which then orchestrates calls to the **Python ML Service**. This separation of concerns ensures modularity, scalability, and ease of development.



2. Data Flow

Diagram Description:

The data flow diagram visualizes the sequence of events and data exchange between components for the three main user journeys: Dataset Upload, Model Training, and Real-Time Simulation. Each journey is color-coded for clarity (Blue, Green, Red).

2.1 Dataset Upload & Processing (Blue Path)

This journey begins when a user uploads a CSV file through the Angular UI.

1. The file is sent via a `POST` request to the `/upload` endpoint on the .NET backend.
2. The backend receives the file as `FormData`, reads and parses the CSV content.
3. The parsed data is augmented with a synthetic timestamp column (if missing) and temporarily persisted in memory or storage.
4. The backend calculates crucial metadata from the dataset, such as row count, pass rate, and date range.
5. This metadata is returned as a JSON response to the frontend.
6. Finally, the frontend displays this summary information to the user, confirming a successful upload and enabling the next step.

2.2 Model Training & Evaluation (Green Path)

This journey is triggered when the user configures date ranges and clicks "Train Model".

1. The frontend sends the selected date ranges to the backend's `/train-model` endpoint.
2. The backend forwards this request to the Python ML service's `/train` endpoint.
3. The ML service requests the training and testing data for the specified ranges from the backend.
4. The backend fetches this data from temporary storage and sends it to the ML service.
5. The ML service executes the model training pipeline (using e.g., XGBoost), evaluates the model on the test set, and generates performance metrics and charts.
6. These results are sent back to the backend, which forwards them to the frontend.
7. The frontend then displays the evaluation metrics (Accuracy, Precision, Recall, F1-Score) and visualization charts to the user.

2.3 Real-Time Prediction Simulation (Red Path)

This journey mimics a production line by streaming data and performing inference in real-time.

1. The user initiates the process by clicking "Start Simulation" in the UI.
2. The frontend calls a backend endpoint (e.g., `/start-simulation`) to begin the process for the pre-defined simulation date range.
3. The backend calls the ML service's `/simulate-rows` endpoint, specifying the range.
4. For each record within the simulation period, the ML service requests a single row of data from the backend.
5. The backend fetches the next row from storage and sends it to the ML service.
6. The ML service performs inference on the row using the pre-trained model, calculating a prediction and confidence score.
7. The prediction result is returned to the backend.
8. The backend immediately streams this result back to the frontend using a real-time communication technology like **SignalR** or Server-Sent Events (SSE).
9. The frontend receives the streamed data and updates the UI live (charts, counters, table) for the user to observe, with rows appearing at approximately one-second intervals until the simulation is complete.

3. API Contract & Payload Structure

This section defines the key REST API endpoints exposed by the .NET Backend and the Python ML Service.

3.1 .NET Backend API Endpoints

The .NET backend serves as the main API gateway for the Angular frontend.

Endpoint 1: Upload and Process Dataset

- **Method:** `POST`
- **URL:** `/api/upload`
- **Description:** Accepts a CSV file, processes it, adds synthetic timestamps, and returns metadata.
- **Request Headers:** `Content-Type: multipart/form-data`
- **Request Body (FormData):**

json

```
{
  "file": "<CSV_FILE>"
}
```

- **Success Response (200 OK):**
  json

```
{
  "fileName": "production_data.csv",
  "totalRecords": 14704,
  "totalColumns": 5,
  "passRate": 70.25,
  "minTimestamp": "2021-01-01T00:00:00",
  "maxTimestamp": "2021-12-31T23:59:59"
}
```

Endpoint 2: Validate Date Ranges

- **Method:** `POST`
- **URL:** `/api/date-ranges/validate`
- **Description:** Validates the user-selected training, testing, and simulation periods.
- **Request Body:**
  json

```
{
  "trainingStart": "2021-01-01",
  "trainingEnd": "2021-06-30",
  "testingStart": "2021-07-01",
  "testingEnd": "2021-09-30",
  "simulationStart": "2021-10-01",
  "simulationEnd": "2021-12-31"
}
```

- **Success Response (200 OK):**
  json

```
{
  "isValid": true,
  "trainingRecordCount": 8000,
  "testingRecordCount": 3000,
  "simulationRecordCount": 3704,
  "message": "Date ranges validated successfully!"
}
```

Endpoint 3: Train Model

- **Method:** `POST`
- **URL:** `/api/model/train`
- **Description:** Triggers the model training process on the ML service using the validated date ranges.
- **Request Body:** (Uses the same structure as the date range validation endpoint).
- **Success Response (200 OK):**
  json

```
{
  "status": "Success",
  "metrics": {
    "accuracy": 0.924,
    "precision": 0.891,
    "recall": 0.905,
    "f1Score": 0.898
  },
  // Base64 encoded images for charts (optional)
  "accuracyLossChart": "base64_encoded_image_string...",
  "confusionMatrixChart": "base64_encoded_image_string..."
}
```

Endpoint 4: Start Simulation (Server-Sent Events - SSE)

- **Method:** `GET`
- **URL:** `/api/simulation/start?start=2021-10-01&end=2021-12-31`
- **Description:** Opens an SSE stream to receive prediction results in real-time.
- **Response Stream (text/event-stream):**
  text

```
event: prediction
data: {"timestamp":"2021-10-01T00:00:00", "id":12345, "prediction":"Pass",
"confidence":0.87, "temperature":24.5, "pressure":1013.2, "humidity":45.1}

event: prediction
data: {"timestamp":"2021-10-01T00:00:01", "id":12346, "prediction":"Fail",
"confidence":0.92, "temperature":25.1, "pressure":1012.8, "humidity":46.7}

event: complete
data: {"totalProcessed": 3704, "passCount": 2600, "failCount": 1104, "avera
geConfidence": 0.891}
```

3.2 Python ML Service API Endpoints

The .NET backend calls these internal endpoints.

Endpoint 1: Train Model

- **Method:** `POST`
- **URL:** `/train`
- **Request Body:** (Same date range structure as above).
- **Response:** (Same metrics structure as above).

Endpoint 2: Get Prediction for Simulation

- **Method:** `POST`
- **URL:** `/predict`
- **Description:** Accepts a single row of data and returns a prediction. Called repeatedly by the backend during simulation.
- **Request Body:**
json

```
{
  "Id": 12345,
  "Feature1": 0.54,
  "Feature2": 250,
  "Temperature": 24.5,
  "Pressure": 1013.2,
  "Humidity": 45.1
}
```

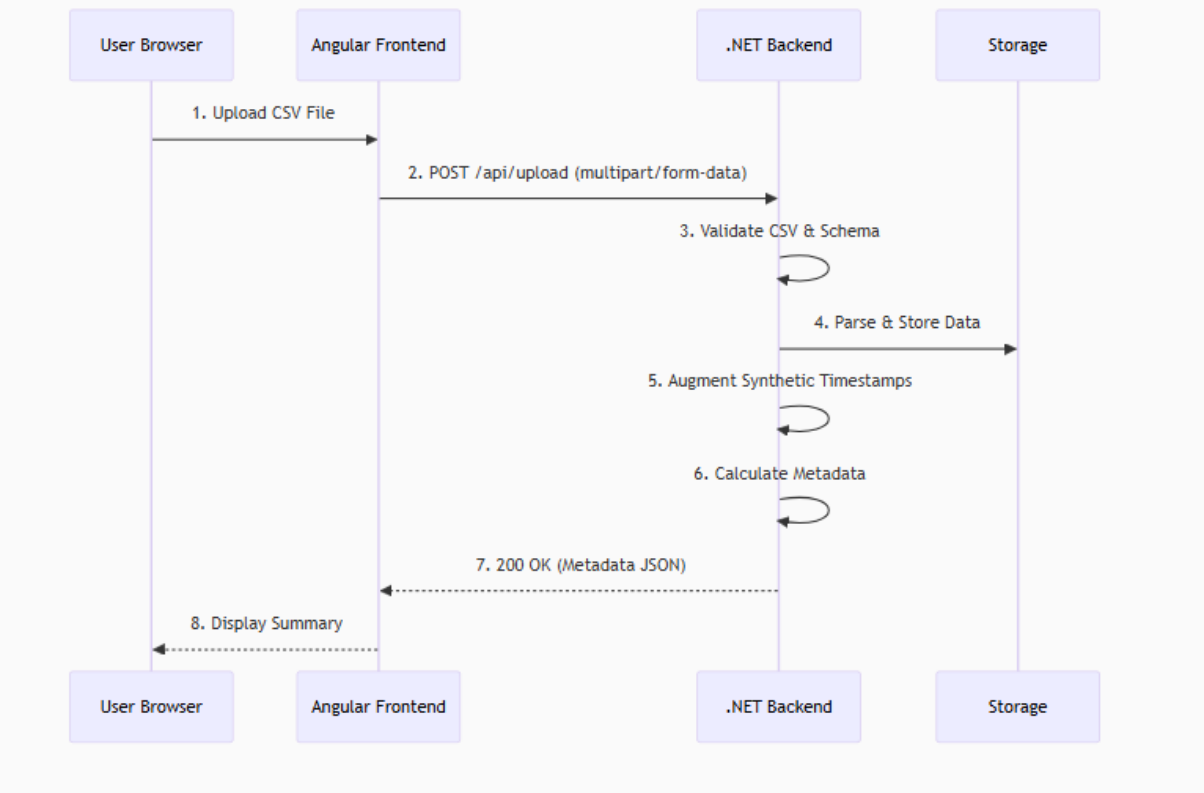
- **Response:**
json

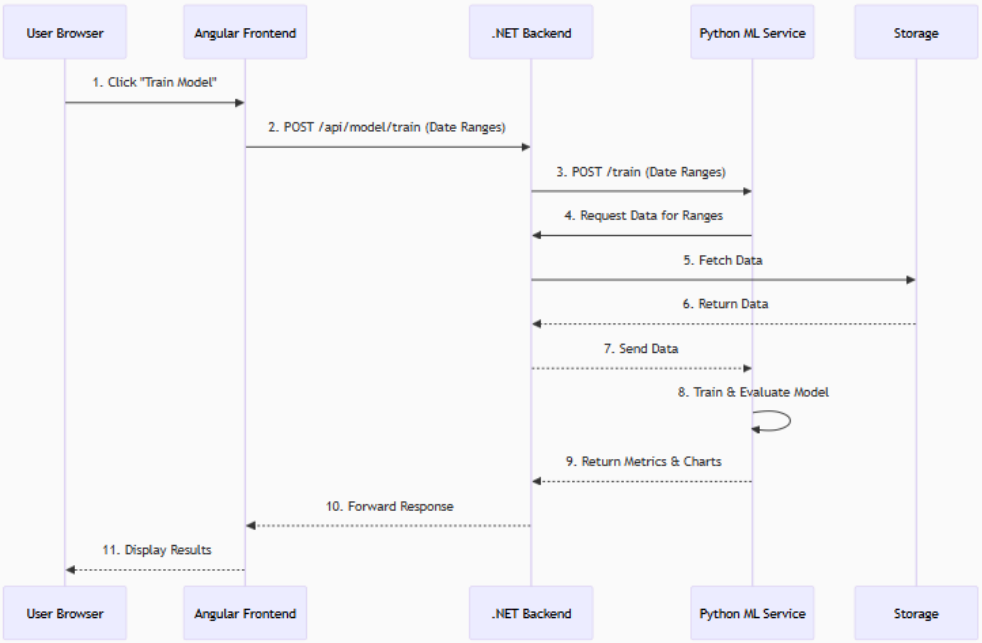
```
{
  "prediction": 1,
  "confidence": 0.87,
  "label": "Pass"
}
```

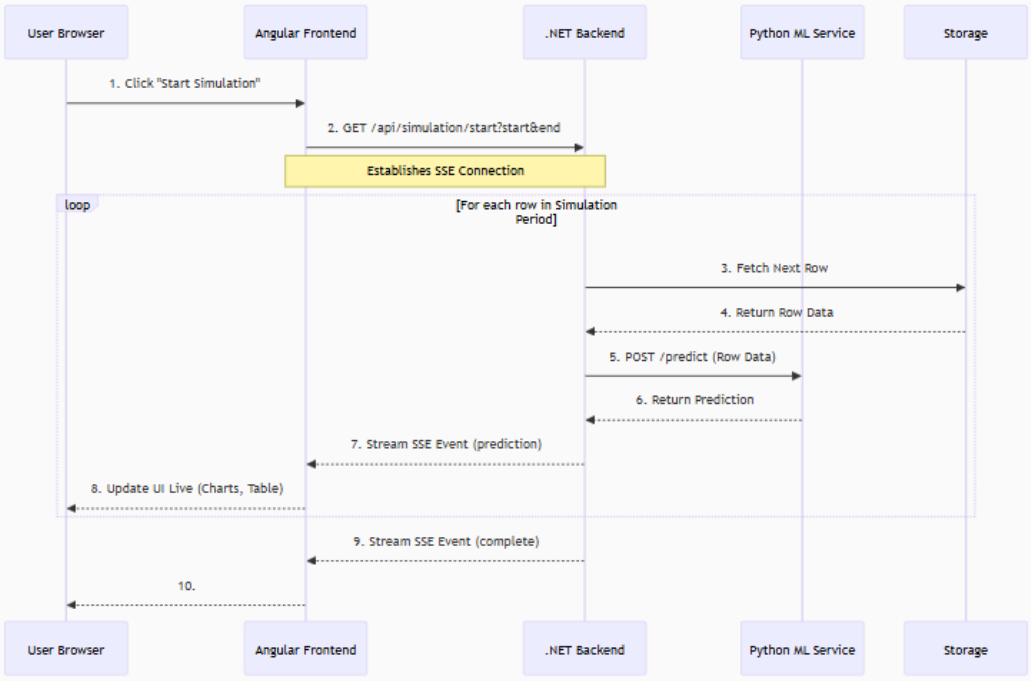
---

4. Mermaid Code for Sequence Diagrams

Here is the Mermaid code to generate sequence diagrams for the three main flows, providing a dynamic view of the API interactions.




User Browser
Angular Frontend
.NET Backend
Python ML Service
Storage
1. Click "Train Model"
2. POST /api/model/train (Date Ranges)
3. POST /train (Date Ranges)
4. Request Data for Ranges
5. Fetch Data
6. Return Data
7. Send Data
8. Train & Evaluate Model
9. Return Metrics & Charts
10. Forward Response
11. Display Results
User Browser
Angular Frontend
.NET Backend
Python ML Service
Storage


User Browser
Angular Frontend
.NET Backend
Python ML Service
Storage
1. Click "Start Simulation"
2. GET /api/simulation/start?start&end
Establishes SSE Connection
loop
[For each row in Simulation Period]
3. Fetch Next Row
4. Return Row Data
5. POST /predict (Row Data)
6. Return Prediction
7. Stream SSE Event (prediction)
8. Update UI Live (Charts, Table)
9. Stream SSE Event (complete)
10.
User Browser
Angular Frontend
.NET Backend
Python ML Service
Storage