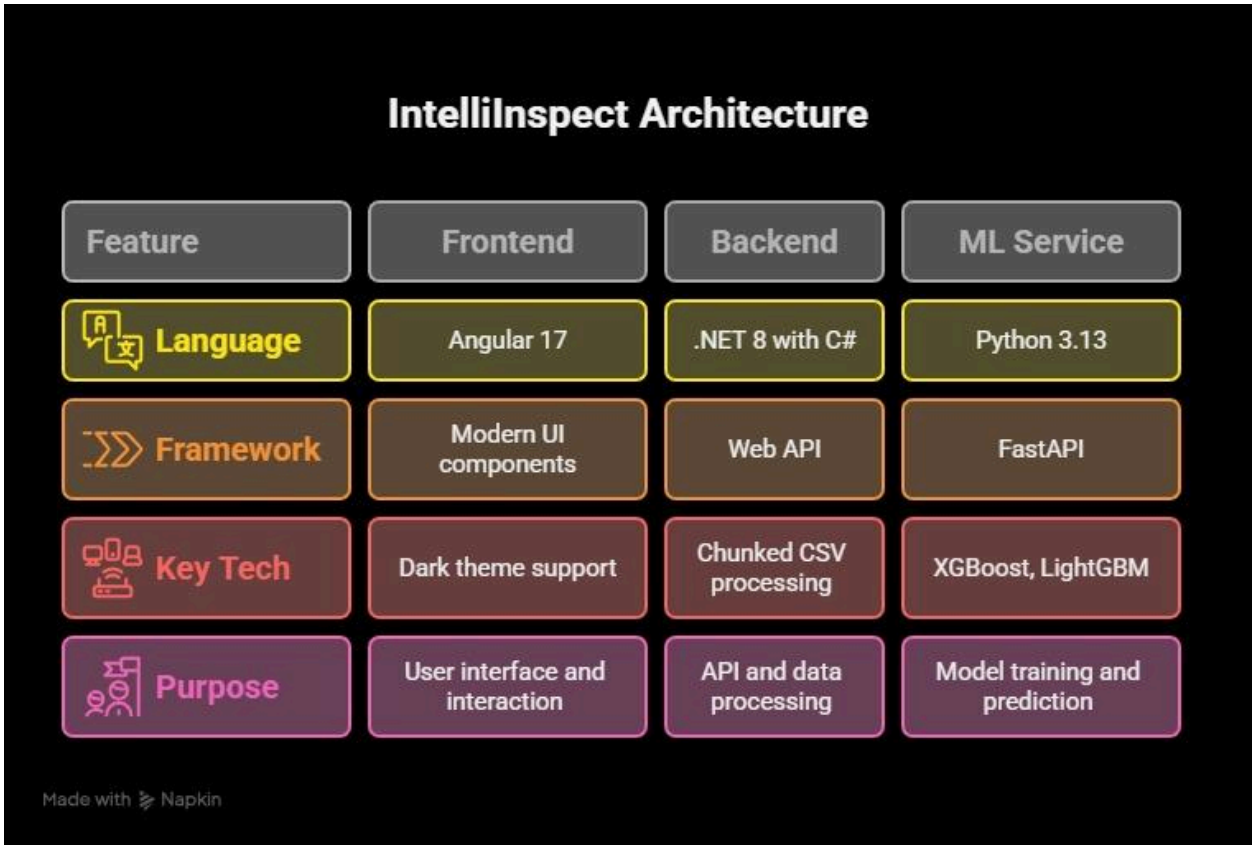
ABB Hackathon Assessment Team 13

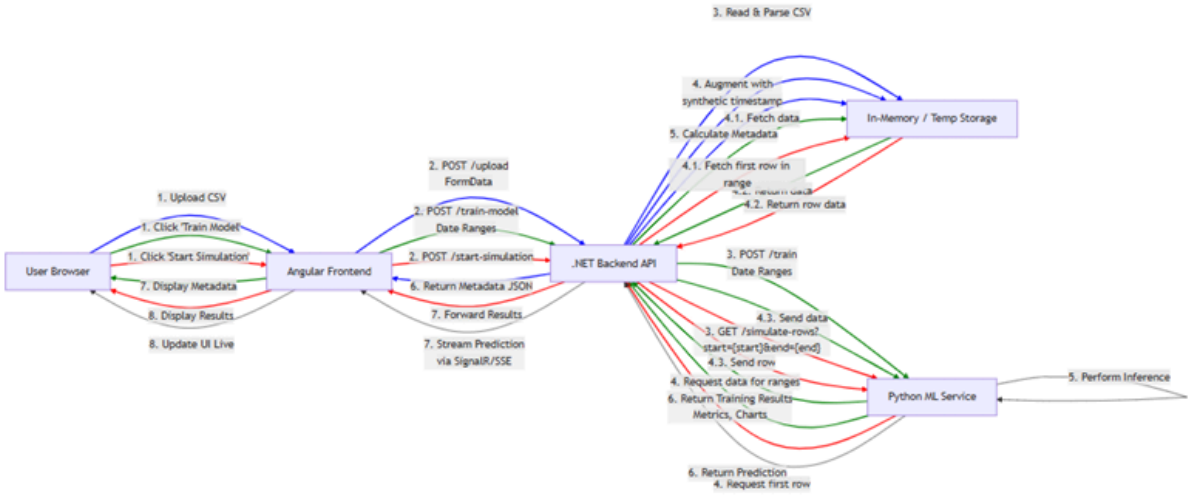


1. System Architecture

Diagram Description

The architecture of **IntelliInspect** is organized into three major services — **Frontend**, **Backend**, and **ML Service** — each optimized for a specific role within the application. The diagram highlights the technologies, frameworks, key features, and primary purpose of each service.

All services are containerized and deployed using Docker Compose, ensuring seamless communication, scalability, and maintainability.



2. Data Flow

2.1 Dataset Upload & Processing

1. The user uploads a CSV file through the Angular interface.
2. The file is sent via POST /api/upload to the backend.
3. The backend reads and parses the CSV content.
4. A synthetic timestamp column is added if it's missing.
5. Metadata such as row count, pass rate, and date range is calculated.
6. The summary is sent back to the frontend for user confirmation.

2.2 Model Training & Evaluation

1. The user selects date ranges and clicks "Train Model".
2. The frontend sends a request to POST /api/model/train.
3. The backend forwards the request to the ML service's /train endpoint.

4. The ML service requests the dataset and trains the model using algorithms like XGBoost.
5. Evaluation metrics such as Accuracy, Precision, Recall, and F1 Score are calculated.
6. The results are sent back to the backend and displayed on the frontend with charts.

2.3 Real-Time Simulation

1. The user starts the simulation by clicking “Start Simulation”.
2. The frontend sends a request to GET /api/simulation/start.
3. The backend calls the ML service’s /simulate-rows endpoint.
4. The ML service requests data one row at a time from the backend.
5. The backend sends rows from storage.
6. The ML service predicts outcomes using the trained model.
7. The prediction results are streamed back via SSE to the frontend.
8. The frontend updates charts and tables in real-time until the simulation is complete.

---

3. API Contract & Payload Structure

3.1 .NET Backend API Endpoints

---

1. **Upload and Process Dataset**

- **Method**: POST
- **URL**: /api/upload
- **Description**: Uploads a CSV file and returns dataset metadata.
- **Headers**: Content-Type: multipart/form-data
- **Request Body**:

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

- **Response (200 OK)**:

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

---

2. Validate Date Ranges

- **Method**: POST
- **URL**: /api/date-ranges/validate

- **Description**: Validates date range inputs for training, testing, and simulation.

- **Request Body**:

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

- **Response (200 OK)**:

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

---

3. Train Model

- **Method**: POST

- **URL**: /api/model/train

- **Description**: Triggers model training on the ML service.

- **Request Body**: Same structure as date range validation.
- **Response (200 OK)**:

```
{
  "status": "Success",
  "metrics": {
    "accuracy": 0.924,
    "precision": 0.891,
    "recall": 0.905,
    "f1Score": 0.898
  },
  "accuracyLossChart": "base64_encoded_image_string...",
  "confusionMatrixChart": "base64_encoded_image_string..."
}
```

---

4. Start Simulation (SSE)

- **Method**: GET
- **URL**: /api/simulation/start?start=2021-10-01&end=2021-12-31
- **Description**: Streams real-time predictions using Server-Sent Events.
- **Response (text/event-stream)**:

```
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, "averageConfidence":0.891}
```

3.2 Python ML Service API Endpoints

---

1. Train Model

- **Method**: POST
- **URL**: /train
- **Request Body**: Same as date range validation.
- **Response**: Same structure as backend's train model response.

---

2. Get Prediction for Simulation

- **Method**: POST
- **URL**: /predict

- **Description**: Predicts a single row of data during simulation.

- **Request Body**:

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

- **Response**:

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