MiniML- Real-Time Predictive Quality Control

System Design Document:

Project Name: Real-Time Quality Prediction System
Technologies Used: Angular (Frontend), ASP.NET Core (Backend) + FastAPI (ML Backend), PostgreSQL (Database)

1. System Overview:

The Real-time Quality Prediction System is designed to process simulation data, train machine learning models, and return predictions in near real-time. The system uses a microservice architecture to ensure modularity, scalability, and maintainability. Each service is responsible for a specific functionality, such as handling authentication, file uploads, simulations, and ML inference. The system allows users to log in, upload CSV datasets, configure simulation parameters like date ranges, initiate Training, and run predictions through a seamless UI.

2. System Architecture:

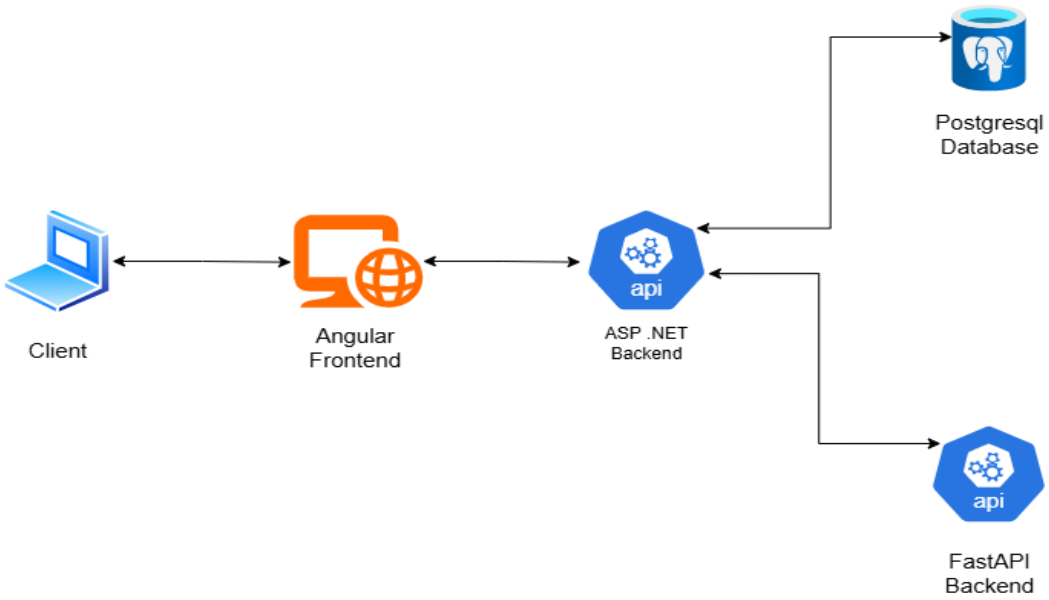

Figure: 1

The system comprises four major parts: the frontend interface, a microservice-based backend, a PostgreSQL-based database layer, and internal communication between services through RESTful APIs. The frontend is built using Angular, providing a reactive, component-driven UI that communicates with backend microservices over HTTP. The backend comprises four core microservices: Login Microservice, Simulation Instance Microservice, CSV Microservice, and ML Microservice. These services are implemented using a combination of ASP.NET Core for authentication and simulation management, and FastAPI in Python for CSV handling and machine learning tasks. Each service interfaces with its own PostgreSQL database, ensuring data separation and service independence. The entire system is containerized using Docker and orchestrated with Docker Compose, making it easy to manage, scale locally, and deploy consistently across environments.

3. Data Flow:

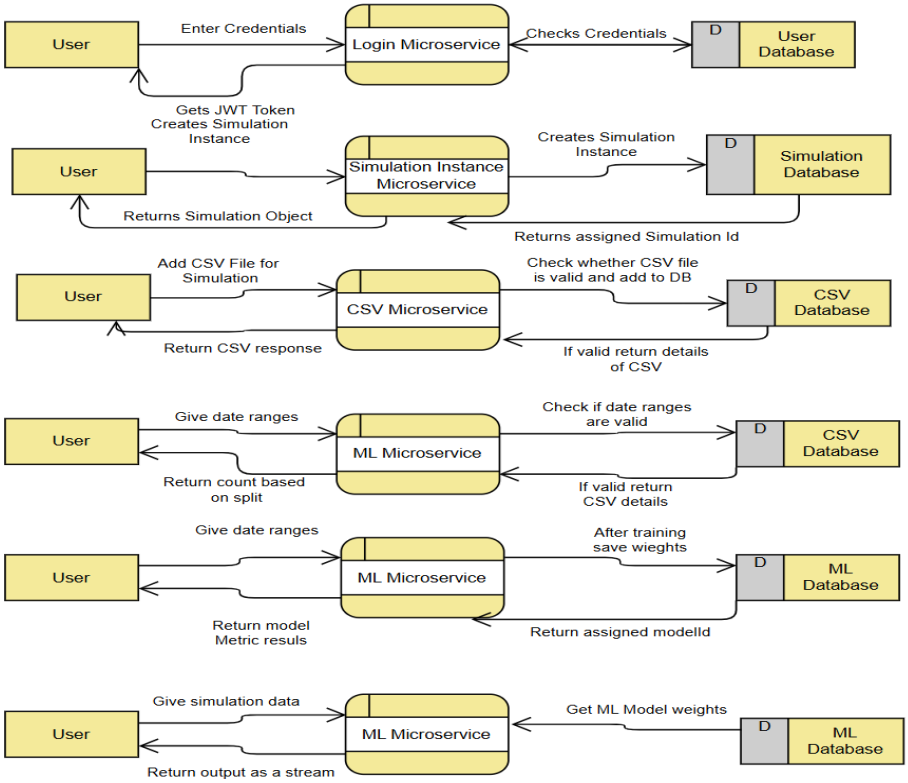

Figure: 2

Each user interaction flows through several microservices, and each service contributes to a specific segment of the data processing pipeline. The Login Microservice handles authentication. When a user logs in, it checks credentials against the User Database and returns a JWT token upon success. This token is used to authorize future requests.

Once authenticated, the user can create a simulation using the Simulation Instance Microservice. This service stores simulation metadata into the Simulation Database and returns a unique simulation ID, which is used to track subsequent activities like uploads and Training.

The user then uploads a CSV file via the CSV Microservice, which validates the structure and format of the file. If valid, the CSV contents are stored in the CSV Database, and a response containing metadata such as row count, date ranges, number of columns, and pass rate is returned to the user.

To begin Training, the user selects a date range for Training, testing, and simulation data. This input is sent to the ML Microservice, which first verifies whether the selected dates match entries in the CSV Database and whether the dates chosen do not overlap. If valid, it returns a split count that helps the user understand the Training and testing data distribution.

Next, the user triggers model Training through the ML Microservice. Using the validated CSV data and specified date ranges, the microservice trains a machine learning model and saves the weights and

model metadata in the ML Database. It then returns a unique model ID that identifies this trained model.

For real-time predictions, the user provides live or simulation data to the ML Microservice and the model ID. The ML Microservice fetches the relevant model weights from the ML Database and runs the inference logic. The predictions are returned to the frontend in a streaming format (data is sent individually). This allows near real-time results and visualization through graphs (line graph and donut graph), a live-prediction table, and a live statistics panel.

4. Conclusion:

In conclusion, the Real-time Quality Prediction System offers a robust end-to-end solution for predictive quality control in manufacturing and simulation environments. The system ensures high modularity, maintainability, and scalability by seamlessly integrating data ingestion, model Training, and real-time inference through a microservice-based architecture. C. Kiranmayi and S. Karthikeyene developed the responsive Angular-based frontend. At the same time, Shivakumar B. was responsible for implementing the ASP .NET backend APIs, integrating them with the Angular frontend and the FastAPI ML service, and deploying them using Docker Compose. The machine learning services were implemented by Ajay Perla, enabling accurate model Training and real-time prediction capabilities. The system delivers a reliable and scalable pipeline for real-time quality assessment and monitoring.