OJT Product Requirement Document

Student: KUSHAGRA PANDEY

Roll No: 240410700064

Year & Section: 2nd Year – Semester 4

Project Title: Taxi Trip Duration Prediction for Dispatch / ETA

Project Type: ML-Enabled Web Application

Tech Stack: React, Tailwind CSS, FastAPI, XGBoost, scikit-learn, PostgreSQL, Docker, AWS EC2

1. Problem Statement

Taxi dispatch systems require accurate Estimated Time of Arrival (ETA) predictions before assigning drivers to trips. Inaccurate ETA leads to:

- Driver misallocation
- Increased ride cancellations
- Reduced customer satisfaction
- Revenue inefficiencies

This project builds an end-to-end deployable system that predicts taxi trip duration using historical trip data and exposes predictions through a web interface and REST API.

The goal is to create a production-style ML system, not just a trained model.

---

2. Objectives

Primary Goals

1. Train an optimized XGBoost regression model for trip duration prediction
2. Deploy a REST API for real-time ETA prediction
3. Build a React-based UI for dispatch input
4. Log predictions and model metadata in PostgreSQL
5. Deploy the complete system on AWS EC2 using Docker

---

3. Target Users

- Taxi dispatch operators

- Fleet managers
- Ride-sharing startups
- Transportation analytics teams

---

4. Data Description

Expected Dataset Columns

- pickup_datetime
- pickup_latitude
- pickup_longitude
- dropoff_latitude
- dropoff_longitude
- trip_duration (target variable in seconds)

Derived Features

- Hour of day
- Day of week
- Weekend indicator
- Haversine distance (km)
- Location encoding (cluster-based)

---

5. System Overview

User Flow

Dispatcher → Enter pickup, dropoff, time → Submit → API → Model → ETA returned → Log prediction

Admin Flow

Upload dataset → Validate → Train model → Evaluate → Save model version → Update metrics

---

6. ML Approach

Problem Type

Supervised Regression

Model

XGBoost Regressor (Tree-based Gradient Boosting)

Baseline Model

Mean trip duration predictor

Baseline MAE target (example NYC taxi dataset benchmark):

~300–350 seconds

Final Model Target

Metric	Target Value
MAE	≤ 180 seconds
RMSE	≤ 250 seconds
R^2 Score	≥ 0.80

These targets represent ~40–50% improvement over baseline.

---

7. Data Pipeline

1. CSV Upload
2. Schema validation
3. Missing value handling
4. Outlier removal (extreme trip durations)
5. Feature engineering
6. Train/Test split (80/20)
7. Model training
8. Model evaluation
9. Model serialization (.pkl)
10. Inference via API

---

8. Core Features (Must-Have)

ML Layer

- Automated feature engineering
- XGBoost model training
- Evaluation dashboard (MAE, RMSE, R^2)
- Model version tracking

Backend (FastAPI)

- /predict endpoint
- /retrain endpoint
- /metrics endpoint
- Prediction logging

Frontend (React)

- Trip input form
- ETA display
- Model metrics view
- Retraining trigger

Database (PostgreSQL)

Tables:

- trip_predictions
- model_metadata

---

9. Non-Functional Requirements

Requirement	Target
API Response Time	< 500 ms
System Availability	≥ 95%
Prediction Logging Accuracy	100%
Model Loading Time	< 2 seconds

Deployment Environment	Dockerized

---

10. Deployment Architecture

- Dockerized backend & frontend
- AWS EC2 single instance deployment
- Environment variables managed securely
- Production-ready API documentation (Swagger)

---

11. Timeline (9 Weeks)

Week	Phase	Planned Deliverables
W1	Project Launch & Scope	Finalize problem statement, stakeholders, and success metrics (MAE, latency). Freeze tech stack. Define data schema draft. Setup GitHub repo and project structure.
W2	Product Design & User Flows	Define user flow (Trip → ETA → Log). Define Admin flow (Upload → Retrain → View metrics). Design UI wireframes. Finalize API contract & DB schema.
W3	Technical Design (HLD + LLD) + Feasibility	Create architecture diagrams. Define inference & retraining flows. Risk analysis. Perform EDA, baseline model, initial XGBoost experiment. Record initial MAE/RMSE.
W4	Development Sprint 1 – Foundations	Backend scaffold (FastAPI), frontend scaffold (React), Docker base setup, environment wiring, PostgreSQL connection, health API endpoint.
W5	Development Sprint 2 – Core	Implement feature engineering pipeline. Train optimized XGBoost model. Create /predict API. Integrate DB logging. Connect frontend form to backend. Display ETA.

	Feature Completion	
W6	Development Sprint 3 – First Working Flow	Add model metrics dashboard. Implement retraining endpoint. Add prediction history view. Optimize API latency (<500ms). Functional end-to-end demo (Viva-2 ready).
W7	Deployment & Real Usage	Dockerize frontend & backend. Deploy to AWS EC2. Production testing. Monitor real predictions. Fix deployment issues.
W8	Scalability & Engineering Maturity	Model fine-tuning from usage data. Hyperparameter refinement. Add caching/async logging. Optimize DB queries. Improve retraining pipeline.
W9	Final Evaluation & Industry-Level Defense	Implement stretch feature (SHAP/map/model comparison). Final live demo. Architecture walkthrough. Scaling & roadmap discussion. Viva-3 readiness.

12. Success Metrics

Model Performance

- MAE ≤ 180 seconds
- RMSE ≤ 250 seconds
- $R^2 \geq 0.80$
- ≥ 40% improvement over baseline

System Performance

- API latency < 500 ms (95th percentile)
- No prediction failures under 100 sequential test requests
- Zero data loss in logging

Functional Completion

- 100% working prediction pipeline
- Manual retraining functional
- Deployed public endpoint

13. Key Risks

Risk	Mitigation
Overfitting	Cross-validation
Data leakage	Strict train-test split
High latency	Model caching
Poor data quality	Data validation checks
Deployment errors	Dockerized environment

14. Final Deliverables

- Trained XGBoost model
- Serialized model artifact
- FastAPI prediction service
- React UI
- PostgreSQL schema
- Docker setup
- AWS EC2 deployment
- GitHub repository
- Demo video
- Technical documentation