Product Requirements & Specification Document

Project Name

BoostHouse - House Price Prediction Platform

Description

BoostHouse is a fullstack platform for predicting house prices using XGBoost. It features a REST API (FastAPI), an interactive frontend (Streamlit), and containerized deployment (Docker). The platform supports feature importance visualization and hyperparameter tuning.

---

1. Objectives

Objective	Description
Accurate Price Prediction	Predict house prices using XGBoost regression
User-Friendly Interface	Interactive frontend for predictions and model insights
Model Interpretability	Visualize feature importance
Hyperparameter Tuning	Allow users to tune model parameters
Scalable Deployment	Containerized solution for easy deployment and scaling

---

2. Core Features

Feature	Description
REST API (FastAPI)	Endpoints for predictions, model info, and tuning
Interactive Frontend (Streamlit)	UI for data input, predictions, visualization, and tuning
XGBoost Model	Trained regression model with scikit-learn pipeline
Feature Importance Visualization	Display feature importances (bar chart)
Hyperparameter Tuning	UI and API for adjusting model parameters and retraining
Dockerized Deployment	Dockerfile and docker-compose for fullstack deployment

---

3. User Stories

As a…	I want to…	So that…
Homebuyer	Predict house prices	I can make informed purchase decisions
Data Scientist	Tune model hyperparameters	I can optimize prediction accuracy
Stakeholder	Visualize feature importance	I understand model decisions

DevOps Engineer	Deploy the platform easily	I can scale and maintain the solution

---

4. Functional Requirements

4.1 REST API (FastAPI)

- **Endpoints:**
  - `POST /predict`: Predict price from input features
  - `GET /feature-importance`: Return feature importance data
  - `POST /tune`: Accept hyperparameters, retrain model, return metrics
  - `GET /health`: Health check
- **Input/Output Example:**

```
// POST /predict
{
  "features": {"sqft": 1200, "bedrooms": 3, "bathrooms": 2}
}
// Response
{
  "predicted_price": 350000
}
```

4.2 Frontend (Streamlit)

- **Pages:**
  - Home: Input features, view predictions
  - Feature Importance: Bar chart visualization
  - Hyperparameter Tuning: Sliders/inputs for parameters, retrain button, metrics display
- **Interactions:**
  - Form-based input for predictions
  - Real-time visualization updates
  - Display of retraining results

4.3 Model & Training

- **Model:** XGBoostRegressor (scikit-learn API)
- **Data:** Standard housing dataset (e.g., Boston, or custom CSV)
- **Pipeline:** Preprocessing, training, evaluation, serialization (joblib/pickle)
- **Hyperparameters:** Learning rate, max depth, n_estimators, etc.

4.4 Deployment

- **Docker:** Dockerfile for API, Dockerfile for frontend, docker-compose.yml for orchestration
- **Ports:** API (8000), Frontend (8501)
- **Environment Variables:** Model path, API URL, etc.

---

5. Non-Functional Requirements

Requirement	Specification
Performance	<1s response time for predictions
Scalability	Support for multiple concurrent users
Security	Input validation, no sensitive data exposure
Usability	Intuitive UI, clear error messages
Portability	Fully containerized, runs on any Docker host

---

6. Technical Stack

Layer	Technology
ML Model	XGBoost, scikit-learn
API	FastAPI
Frontend	Streamlit
Deployment	Docker, docker-compose
Language	Python 3.9+

---

7. Architecture Overview

```
[User] <---> [Streamlit Frontend] <--REST--> [FastAPI Backend] <---> [XGBoost Model
                                                | (Dockerized)
```

---

8. Acceptance Criteria

ID	Criteria
AC1	User can input features and receive a price prediction
AC2	Feature importance is visualized in the frontend
AC3	User can tune hyperparameters and retrain the model
AC4	All components run via Docker
AC5	API returns valid responses and handles errors gracefully

---

9. Milestones

Milestone	Description	Due Date (est.)
API MVP	Basic prediction endpoint	Week 1

Frontend MVP	Input form, prediction display	Week 2
Feature Importance	Visualization in UI/API	Week 3
Hyperparameter Tuning	UI/API for tuning & retraining	Week 4
Dockerization	Fullstack deployment	Week 5

---

10. Risks & Mitigations

Risk	Mitigation
Model overfitting	Cross-validation, regularization
Data quality issues	Input validation, preprocessing
Deployment complexity	Use Docker, clear documentation

---

11. Deliverables

- Source code (API, frontend, model)
- Dockerfiles & docker-compose.yml
- Documentation (README, API docs)
- Example dataset

End of Document