AI-Powered Product Recommendation Engine – 7-Day Project Documentation

# Project Context

This project was developed as part of the i95dev AI Engineering Internship assignment, aimed at demonstrating full-stack development and AI/LLM integration in an e-commerce setting. The system leverages FastAPI/Flask, React, and OpenAI GPT models to generate personalized product recommendations from user preferences and browsing history.

# Day-Wise Work Log

## Day 1 – Project Setup & Familiarization

- Reviewed assignment brief, technical requirements, and starter kit provided by the company.  
- Set up local development environment:  
 - Installed Python environment & dependencies (Flask, FastAPI, Pydantic, Uvicorn, OpenAI, dotenv).  
 - Installed frontend dependencies (React 18, React-DOM, react-scripts, testing libraries).  
- Verified backend starter structure (app.py, services/llm\_service.py, product\_service.py).  
- Verified frontend starter (Catalog.js, UserPreferences.js, etc.).  
- Ran both servers locally to confirm setup (http://localhost:3000 frontend, http://127.0.0.1:8000 backend).  
  
Deliverable: Local environment successfully running with starter code.

## Day 2 – Backend Foundations

- Implemented ProductService:  
 - Loaded product data from products.json.  
 - Implemented filtering, search, and catalog retrieval functions.  
- Defined REST endpoints in FastAPI/Flask:  
 - /products → fetch full catalog.  
 - /preferences → accept user preferences.  
 - /history → process browsing history.  
 - /recommendations → return recommendations from LLM.  
- Added CORS middleware for frontend-backend communication.  
- Wrote error-handling utilities with structured JSON responses.  
  
Deliverable: Backend API endpoints for catalog, preferences, and history.

## Day 3 – LLM Integration

- Configured .env for secure OpenAI API key handling.  
- Built LLMService:  
 - Constructed prompts using user preferences + browsing history + product metadata.  
 - Implemented response parsing for recommendations + explanations.  
 - Added error handling (timeouts, malformed responses).  
- Tested recommendation flow with mock user input.  
  
Deliverable: Functional backend integration with GPT, returning recommendations.

## Day 4 – Frontend Core Components

- Implemented Catalog Component:  
 - Displayed products with filtering and search.  
 - Added sorting (by price, rating).  
- Implemented UserPreferences Component:  
 - Form for categories, price range, and style preferences.  
 - On submit → preferences sent to backend.  
- Implemented BrowsingHistory Component:  
 - Allowed users to "click" products to add to simulated browsing history.  
 - History displayed dynamically.  
  
Deliverable: Interactive frontend with catalog, preference form, and browsing history.

## Day 5 – Recommendation Display & State Management

- Implemented Recommendations Component:  
 - Displayed AI-generated recommendations with explanation text.  
 - Added confidence scores (parsed from LLM response).  
- Set up API client (api.js) for unified backend calls.  
- Implemented state management:  
 - Centralized state in App.js using React Hooks (useState, useEffect, useCallback, useMemo).  
 - Ensured minimal re-renders for performance.  
  
Deliverable: Fully functional end-to-end recommendation system visible on frontend.

## Day 6 – Optimization, UX, and Security

- Frontend Enhancements:  
 - Added loading states, error boundaries, responsive layout.  
 - Implemented debounced search for performance.  
- Backend Enhancements:  
 - Caching repeated API responses.  
 - Added pagination for catalog.  
- Security:  
 - Sanitized inputs.  
 - Validated JSON requests with Pydantic.  
 - Ensured no API keys were exposed in code.  
  
Deliverable: Optimized, secure system with better user experience.

## Day 7 – Testing, Documentation & Final Review

- Wrote unit tests:  
 - Backend → API endpoints (products, preferences, recommendations).  
 - Frontend → Component rendering and state updates.  
- Documented:  
 - Setup instructions for backend & frontend.  
 - Prompt engineering strategy.  
 - System architecture overview.  
- Conducted final review:  
 - Verified product catalog search, preferences, browsing history, and recommendation accuracy.  
 - Confirmed app runs in both dev and prod environments.  
  
Deliverable: Completed project with testing, documentation, and deployment readiness.

# Key Features Implemented

- Backend  
 - REST API with FastAPI/Flask.  
 - Product filtering, search, and history management.  
 - LLM-powered recommendation engine with explanations.  
  
- Frontend  
 - Product catalog with filtering/sorting.  
 - User preferences form.  
 - Browsing history tracker.  
 - AI-driven recommendation display.  
  
- AI/ML  
 - Context-aware prompt engineering.  
 - Confidence scoring for recommendations.  
 - Fallback handling when LLM response fails.

# Final Outcome

A production-ready full-stack AI-powered recommendation engine was built within 7 days.  
The system:  
- Accepts user preferences and browsing history.  
- Analyzes product catalog.  
- Generates personalized product recommendations via GPT with explanations.  
- Provides a seamless, responsive user experience.  
  
This project demonstrates:  
- AI integration skills (prompt engineering, GPT).  
- Backend proficiency (FastAPI/Flask, service layer, API design).  
- Frontend skills (React components, hooks, state management).  
- Deployment readiness (error handling, optimizations, documentation).