AI-Powered Smart Shopping Assistant

# 1. Code (Documented Example – Java-like pseudocode)

// Class: RecommendationEngine  
// Purpose: Generate personalized product recommendations based on user preferences and session history.  
  
public class RecommendationEngine {  
   
 private AIModel model; // Trained AI model for making predictions  
   
 // Constructor  
 public RecommendationEngine(AIModel model) {  
 this.model = model;  
 }  
  
 /\*\*  
 \* Generates product recommendations based on user data  
 \* @param user The user to generate recommendations for  
 \* @return List of recommended products  
 \*/  
 public List<Product> generateRecommendations(User user) {  
 List<Product> preferences = user.getPreferences();  
 List<Product> recommendations = model.predict(preferences);  
 return recommendations;  
 }  
}

# 2. Build Instructions (using Maven or similar)

# Step-by-step build commands  
  
# 1. Clone the repository  
git clone https://github.com/your-repo/smart-shopping-assistant.git  
cd smart-shopping-assistant  
  
# 2. Install dependencies  
mvn install  
  
# 3. Compile the project  
mvn compile  
  
# 4. Package the project  
mvn package  
  
# 5. Run the application  
java -jar target/shopping-assistant.jar

💡 \*Note: Replace Maven with npm, pip, or Gradle if using JavaScript or Python.\*

# 3. Unit Test (Example using JUnit)

// Unit test for RecommendationEngine  
import static org.junit.Assert.\*;  
import org.junit.Test;  
import java.util.\*;  
  
public class RecommendationEngineTest {  
  
 @Test  
 public void testGenerateRecommendations() {  
 AIModel model = new MockAIModel(); // Mocked model for testing  
 RecommendationEngine engine = new RecommendationEngine(model);  
 User testUser = new User();  
 testUser.addPreference(new Product("Eco Bottle", true));  
  
 List<Product> results = engine.generateRecommendations(testUser);  
   
 // Assert that the recommendations are not empty  
 assertFalse(results.isEmpty());  
  
 // Assert that all recommendations are eco-friendly  
 for (Product p : results) {  
 assertTrue(p.isEcoFriendly());  
 }  
 }  
}

# 4. DevOps Overview

## CI/CD Pipeline (GitHub Actions or GitLab CI)

name: Build and Test  
  
on: [push, pull\_request]  
  
jobs:  
 build:  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout Code  
 uses: actions/checkout@v2  
  
 - name: Set up JDK  
 uses: actions/setup-java@v1  
 with:  
 java-version: '17'  
  
 - name: Build with Maven  
 run: mvn clean install  
  
 - name: Run Unit Tests  
 run: mvn test

## DevOps Best Practices

|  |  |
| --- | --- |
| Area | Practice |
| Version Control | Use Git with feature branches and pull requests |
| CI/CD | Automate build, test, and deploy via GitHub Actions |
| Monitoring | Integrate tools like Prometheus and Grafana (optional) |
| Deployment | Containerize using Docker, deploy to Heroku, AWS, or Azure |
| Security | Use GitHub secrets, encrypt user data, run vulnerability scans |