**High-Level Design (HLD) Document**

**Project Title:** E-Commerce Application  
**Version:** 1.0  
**Date:** [Insert Date]  
**Prepared By:** [Insert Name]

**1️⃣ Document Purpose**

This document provides a high-level overview of the technical architecture, key system components, and interactions for the development of the E-Commerce Application.

**2️⃣ System Overview**

The E-Commerce Application aims to provide an online platform for customers to browse, purchase products, and make secure payments, with an admin interface for managing products, orders, and reports.

**3️⃣ Architectural Diagram**

(Include architecture diagram here – e.g., using AWS, Azure, or a generic 3-tier model.)

**Typical Architecture:**

* **Front-End:** Web & Mobile Interface
* **API Gateway:** Central entry point for APIs
* **Back-End:** Application Layer (business logic, services)
* **Database:** Product, order, and user data storage
* **Third-Party Integrations:** Payment gateway, SMS/Email notifications

**4️⃣ Core Components**

**4.1 Front-End**

* React.js / Angular / Vue.js
* Responsive web design
* REST API or GraphQL consumption

**4.2 Back-End**

* Node.js / Java Spring Boot / .NET Core
* RESTful APIs for front-end interaction
* JWT-based authentication
* Business logic handling

**4.3 Database**

* MySQL / PostgreSQL / MongoDB
* Tables: Users, Products, Orders, Payments, Inventory, Promotions

**4.4 Admin Panel**

* Manage products, inventory, orders, and reports
* Role-based access control (RBAC)

**4.5 Third-Party Integrations**

* Payment Gateway (Razorpay / PayU / Stripe)
* SMS & Email Notifications (AWS SNS / Twilio / SMTP)
* Analytics tools (Google Analytics)

**5️⃣ Data Flow Diagram**

Include a Level-1 Data Flow Diagram (DFD) showing:

* Customer placing order
* API calls from front-end to back-end
* Payment gateway processing
* Notifications to customer
* Admin dashboard operations

**6️⃣ Key Design Decisions**

* Cloud-native design for scalability (AWS/Azure)
* API-first approach for integration flexibility
* Role-based access control for admin security
* Token-based authentication (JWT)
* Modular codebase for maintainability

**7️⃣ Non-Functional Design Considerations**

| **Category** | **Design Approach** |
| --- | --- |
| Security | HTTPS, data encryption, PCI-DSS compliance |
| Performance | Load-balanced APIs, optimized DB queries |
| Scalability | Auto-scaling (AWS EC2 / Azure VM Scale Sets) |
| Availability | 99.9% uptime using multi-zone deployments |
| Backup & Recovery | Daily backups, disaster recovery plan |

**8️⃣ Risks and Mitigation**

| **Risk** | **Mitigation Strategy** |
| --- | --- |
| Payment gateway failure | Multi-gateway fallback (Phase 2) |
| Traffic spikes during promotions | Implement auto-scaling |
| Data breach | Regular security audits, encryption |

**9️⃣ Future Enhancements**

* Native mobile app integration
* Multi-vendor marketplace (Phase 2)
* Loyalty & rewards management
* AI-driven product recommendations

**🔟 Approval**

| **Name** | **Role** | **Signature** | **Date** |
| --- | --- | --- | --- |
| [Solution Architect] | Architect |  |  |
| [Project Manager] | PM |  |  |
| [Business Lead] | Product Owner |  |  |