**Architectural Design Document – E-Commerce Application**

**Version:** 1.0  
**Prepared By:** [Insert Name]  
**Date:** [Insert Date]

**1️⃣ Objective**

Design a scalable, secure, and cloud-hosted **E-Commerce Application** supporting:

* Customer-facing storefront
* Secure checkout and payment
* Admin panel for backend operations
* API-driven architecture for flexibility

**2️⃣ Architecture Overview**

**📊 High-Level Architecture Diagram**

*(Include a diagram with these layers:)*

* **Client Layer:** Web browsers, mobile app (optional)
* **API Gateway Layer**
* **Application Layer (Backend Services)**
* **Database Layer**
* **Third-Party Services (Payments, Notifications)**
* **Admin Portal**

**3️⃣ Key Architectural Components**

| **Component** | **Technology** | **Purpose** |
| --- | --- | --- |
| Front-End | React.js / Angular / Vue.js | Customer storefront UI |
| API Gateway | AWS API Gateway / Nginx / Express Gateway | Central routing for backend APIs |
| Backend APIs | Node.js / Java Spring Boot / .NET Core | Handle business logic and data operations |
| Database | MySQL / PostgreSQL / MongoDB | Store transactional data |
| Payment Integration | Razorpay / PayU / Stripe | Secure payment handling |
| Notification Services | Twilio / AWS SNS / SMTP | SMS and email notifications |
| Admin Panel | React.js / Angular / Vue.js | Backend operations interface |
| Cloud Infrastructure | AWS / Azure / GCP | Hosting, load balancing, scaling |
| CI/CD | Jenkins / GitLab CI / Azure DevOps | Automated deployment pipelines |

**4️⃣ Data Flow Diagram (Level 1)**

1. Customer browses products →
2. Front-end sends API requests via API Gateway →
3. Backend APIs retrieve data from DB and send responses →
4. Customer adds to cart, places order →
5. Payment gateway processes transaction →
6. Backend updates order and sends notifications →
7. Admin manages inventory/orders via admin panel.

**5️⃣ Key Design Principles**

* **API-First Architecture:** RESTful APIs for all operations.
* **Security by Design:** HTTPS, JWT tokens, role-based access control, PCI-DSS compliance.
* **Modular Backend Services:** Each functional area modularized.
* **Scalability:** Cloud auto-scaling with load balancers.
* **Resilience:** Multi-zone deployment for high availability.

**6️⃣ Database Design**

* **Relational DB (MySQL/PostgreSQL):**  
  Tables: users, products, categories, orders, order\_items, cart, payments, notifications.
* **NoSQL Option (MongoDB):**  
  Product catalog stored as collections for flexible data models (optional).
* **ER Diagram:**  
  *(Include ER diagram showing entity relationships.)*

**7️⃣ Security Architecture**

* SSL/TLS enforced on all endpoints.
* JWT-based authentication for APIs.
* Secure password storage (bcrypt hashing).
* Payment data handled via PCI-DSS certified gateways.
* Role-based access control (RBAC) for admin module.

**8️⃣ Cloud Deployment Model**

| **Layer** | **AWS Example** |
| --- | --- |
| Front-End | S3 + CloudFront (Static Hosting) |
| API Gateway | AWS API Gateway |
| Backend APIs | EC2 / Lambda (Microservices) |
| Database | RDS (MySQL/PostgreSQL) |
| Load Balancer | Application Load Balancer (ALB) |
| Monitoring | CloudWatch / ELK Stack |
| CI/CD | AWS CodePipeline / Jenkins |

**9️⃣ Performance and Scalability**

* Horizontal scaling via load balancer.
* Caching frequently accessed data (Redis or in-memory caching).
* CDN for static content (images, CSS, JS).
* Optimized database queries.

**🔟 Disaster Recovery & Backup**

* Daily automated backups of databases and media files.
* Replication across multiple availability zones.
* Infrastructure as Code (IaC) for easy restoration.

**1️⃣1️⃣ Future Considerations**

* Progressive Web App (PWA)
* Microservices split for core modules
* Multi-vendor support
* AI/ML integrations (product recommendations)

**1️⃣2️⃣ Approval**

| **Name** | **Role** | **Signature** | **Date** |
| --- | --- | --- | --- |
| Solution Architect | Architect |  |  |
| Project Manager | PM |  |  |
| Development Lead | Technical Lead |  |  |