SAMPLE FILLED WORKBOOK

System Design – E-Commerce Platform

PART 1: Selected System

Group Name:

Team Alpha Architects

Chosen Use Case:

Scalable E-Commerce Platform

Target Users:

Urban online shoppers aged 18–45

Primary Business Goal:

Increase conversion rate and handle high seasonal traffic (festive sales)

PART 2: Requirements Definition

A. Functional Requirements

1. Users can browse products by category

2. Users can search products
3. Users can add products to cart
4. Users can complete checkout with payment integration
5. Admins can manage inventory
6. Users can track orders

B. Non-Functional Requirements

- High availability
- Low latency
- High throughput
- Fault tolerance
- Eventual consistency (for cart and analytics)

Custom:

1. Must support flash sales
2. Must scale during peak events

C. Prioritization

Performance over strict consistency (minor delay in cart sync is acceptable)

Availability over perfect accuracy during peak traffic

PART 3: Scale Estimation

Daily active users: 1 million

Peak concurrent users: 150,000

Peak requests per second: 20,000 RPS

Data stored per year: 5 TB

Expected growth rate: 30% annually

PART 4: High-Level Architecture

User

→ CDN

→ Load Balancer

→ API Gateway

→ Microservices Layer

→ Databases / Cache

→ Response

Components Used

- CDN for product images

- Load balancer

- API gateway
- Microservices (Catalog, Cart, Orders, Payments, Users)
- Redis cache
- SQL database (orders, payments)
- NoSQL database (product catalog)
- Message queue for order processing

Architecture Style

Microservices

Justification:
Independent scaling of services (cart scales more than user profile). Better fault isolation during sales.

PART 5: CAP Theorem Decision

Chosen: Availability + Partition Tolerance

Reason:
During high traffic, the system must remain operational even if some nodes fail. Minor temporary inconsistency is acceptable.

PART 6: Scalability Strategy

Horizontal scaling

Application servers auto-scale based on traffic

Caching Strategy

Cache product details and frequently searched queries

Eviction strategy: TTL for trending products, LRU for general cache

---

PART 7: Database Strategy

Polyglot persistence

SQL for transactions (ACID compliance required)
NoSQL for product catalog (flexible schema, fast reads)

Sharding: Yes

Sharding key: User ID for user service, Product ID for catalog

---

PART 8: Communication Design

Hybrid

Synchronous for user-facing APIs
Asynchronous for order processing and inventory updates

Event streaming used for analytics and order status updates

PART 9: GenAI Extension

AI Feature: AI Product Advisor

Purpose:
Provide personalized recommendations and explain why a product suits the user.

RAG Used: Yes

Documents embedded:

- Product descriptions
- Reviews
- FAQ
- Size guides

Stored in:
Vector database

New Risks:

Latency: LLM call adds 1–2 seconds
Cost: Token usage increases during peak
Hallucination: Incorrect product attributes
Security: Prompt injection attempts

Key Risk Explained:
Hallucination may lead to incorrect product claims, affecting customer trust.

PART 10: Trade-Off Reflection

“Our system prioritizes availability over strict consistency because maintaining uptime during sales is critical for revenue.”

Biggest bottleneck:
Database writes during flash sales

Version 2 Improvement:
Introduce write queues and optimize indexing

1-Minute Presentation Summary

We designed a microservices-based e-commerce system optimized for high availability and scalability.
We used polyglot databases and horizontal scaling.
We accepted eventual consistency for better uptime.
We extended the system with an AI product advisor using RAG.

1.