1. Event-Driven Architecture for Order Management

- **Objective:** Design an order processing system for an e-commerce platform to handle high concurrency and real-time updates.
- **Solution:**
  - **Spring Boot Microservices:** Modularized into services such as `Order Service`, `Inventory Service`, and `Payment Service`.
  - **Kafka:** Used as an event bus to decouple services and enable asynchronous communication. Each service produces and consumes relevant events (`order-created`, `inventory-checked`, `payment-processed`).
  - **SQL (MySQL):** Used for relational data such as order details and transaction logs to ensure ACID compliance.
  - **NoSQL (MongoDB):** Used for product catalog storage with dynamic schemas for product metadata.
  - **Outcome:** Achieved scalability to handle 10,000+ orders per second with reduced latency by 40%.

---

2. Real-Time Fraud Detection System

- **Objective:** Create a real-time fraud detection system for financial transactions.
- **Solution:**
  - **Spring Boot Microservices:** `Transaction Service` for receiving and processing transactions, `Fraud Detection Service` for anomaly detection, and `Notification Service` for alerts.
  - **Kafka Streams:** Processed transactions in real-time using aggregation and pattern-matching techniques to detect suspicious activity.
  - **SQL (PostgreSQL):** Used to store user profiles, historical transaction data, and fraud cases.
  - **NoSQL (Redis):** Leveraged for caching frequently accessed user and transaction data to enhance performance.
  - **Outcome:** Reduced fraud detection time from minutes to seconds, improving fraud prevention effectiveness by 25%.

---

3. Personalized Recommendation System

- **Objective:** Build a recommendation engine for a video streaming platform.
- **Solution:**
  - **Spring Boot Microservices:** `User Activity Service` for collecting user interactions, `Recommendation Engine` for generating suggestions, and `Content Delivery Service` for streaming.
  - **Kafka:** Streamed user activity data (`play` , `pause` , `like` , `share`) to the recommendation engine in real time.
  - **NoSQL (Cassandra):** Used for storing user activity logs and watch history at scale.
  - **SQL (PostgreSQL):** Used for maintaining user profiles and subscription data.
  - **Outcome:** Achieved 20% higher user engagement and 15% improvement in content consumption metrics.

---

4. Centralized Logging and Monitoring System

- **Objective:** Implement a centralized logging and monitoring system to manage distributed microservices.
- **Solution:**
  - **Spring Boot Microservices:** Integrated centralized logging frameworks with all services using SLF4J and Logback.
  - **Kafka:** Used to aggregate logs from multiple microservices for real-time log streaming.
  - **SQL (Elasticsearch):** Leveraged for indexing and querying logs to enable detailed search capabilities.
  - **Outcome:** Reduced MTTR (Mean Time to Recovery) by 30% and improved operational efficiency.

---

5. Hybrid Data Management System for Customer 360

- **Objective:** Create a 360-degree view of customers for a retail business.

- **Solution:**
  - **Spring Boot Microservices:** `Customer Profile Service`, `Purchase History Service`, and `Loyalty Points Service` as separate modules.
  - **Kafka:** Unified customer events (`profile-updated`, `order-placed`, `points-redeemed`) into a single customer activity stream.
  - **SQL (Oracle):** Used for structured data like customer profile and order history.
  - **NoSQL (Neo4j):** Used for creating a relationship graph of customers, products, and locations for insights.
  - **Outcome:** Delivered a holistic view of 1M+ customers, enhancing targeted marketing campaigns and improving customer retention by 15%.

---

6. Scalable Messaging Platform

- **Objective:** Design a messaging platform for real-time communication and chat history management.
- **Solution:**
  - **Spring Boot Microservices:** `Messaging Service`, `User Management Service`, and `Notification Service`.
  - **Kafka:** Used for real-time message delivery and maintaining chat event streams.
  - **SQL (PostgreSQL):** Stored structured chat metadata (e.g., user-to-user relationships).
  - **NoSQL (DynamoDB):** Stored unstructured chat content and history for high availability.
  - **Outcome:** Scaled to support 50K concurrent users with 99.99% uptime.