FA22-BSE-058
SYED MUHAMMAD USMAN

QUESTION 1

---

1. Scenario: Scaling Social Media Platforms (Example: Twitter Outage)

- **Project Goal:** Develop a global social media platform to handle millions of real-time tweets, likes, and interactions.
- **Problem Faced: Scalability Issues.**
  - **Details:** Twitter experienced a major outage during high-traffic events (e.g., global sports finals or breaking news). The monolithic architecture struggled to handle the surge in concurrent users.
  - **Cause:** The lack of horizontal scalability in the underlying architecture led to database bottlenecks.
  - **Solution Attempted:** Migrate to a microservices architecture for better load distribution, but this introduced additional challenges in managing inter-service communication.

---

2. Scenario: Building a Banking App (Example: Wells Fargo Mobile App)

- **Project Goal:** Create a secure and user-friendly mobile banking app for seamless transactions and account management.
- **Problem Faced: Integration with Legacy Systems.**
  - **Details:** The bank relied on COBOL-based mainframe systems that were difficult to connect to the modern app.
  - **Cause:** Legacy systems lacked APIs, and data extraction was cumbersome and error-prone.
  - **Impact:** The app occasionally showed incorrect transaction details or delayed processing.
  - **Solution Attempted:** Introduced middleware for data integration, but it increased latency and complexity.

---

3. Scenario: Launching an E-Commerce Platform (Example: Shopify Black Friday Issues)

- **Project Goal:** Build a reliable platform to support businesses during high-demand shopping events.
- **Problem Faced: Concurrency Management and Database Overload.**

  - **Details:** During Black Friday sales, Shopify faced downtime as thousands of businesses processed millions of transactions simultaneously.
  - **Cause:** The database architecture wasn't optimized for such high write-read concurrency.
  - **Impact:** Merchants lost sales due to downtime.
  - **Solution Attempted:** Implemented caching and sharded databases, but re-architecting under time pressure was challenging.

---

4. Scenario: Developing a Video Streaming Service (Example: Netflix)

- **Project Goal:** Deliver high-quality streaming to millions of users worldwide with minimal buffering.
- **Problem Faced: Content Delivery Optimization in Diverse Networks.**
  - **Details:** Netflix struggled to maintain quality in regions with poor internet connectivity.
  - **Cause:** Centralized server architecture couldn’t adapt to varying network conditions efficiently.
  - **Impact:** Users in remote areas experienced buffering and lower video quality.
  - **Solution Attempted:** Netflix introduced a CDN (Content Delivery Network) using edge servers, but managing these globally added logistical challenges.

---

5. Scenario: Implementing AI for Healthcare Diagnosis (Example: IBM Watson Health)

- **Project Goal:** Develop an AI system to assist doctors in diagnosing diseases by analyzing patient data.
- **Problem Faced: Data Privacy and Security.**
  - **Details:** Handling sensitive medical data required strict compliance with regulations like HIPAA.
  - **Cause:** The architecture didn’t initially incorporate secure data isolation and audit trails.
  - **Impact:** Delays in deployment as the system failed privacy audits.
  - **Solution Attempted:** Redesigning the architecture for better encryption and access control, but this slowed down AI model processing.

QUESTION 2

Scenario Recap: Integration with Legacy Systems in Banking Applications

---

Architecture Used: Monolithic to Middleware

- **Legacy Architecture:** A monolithic legacy system stores data but doesn't support modern integration mechanisms like APIs.
- **Modern Architecture:** Middleware acts as a bridge between the modern application and the legacy system.

Problem Faced

- The legacy system used outdated methods (e.g., plain text) for data storage and retrieval.
- It couldn't directly communicate with the modern banking app, which required data in a JSON format.
- This created a bottleneck, as the banking app couldn't process or display user data effectively.

Solution

Introduce **Middleware** to act as an intermediary:

1. **Fetch Legacy Data:** Middleware connects to the legacy system to fetch data.
2. **Process Data:** Converts the legacy data format into a modern format (e.g., JSON).
3. **Serve Modern App:** The processed data is now accessible and usable by the modern application.

Code to Solve the Problem

Legacy System

Simulates the outdated architecture of the legacy banking system.

```
class LegacySystem {
    public String getAccountData(String accountNumber) {
        // Simulating a legacy system returning data in an old format
        return "Account Number: " + accountNumber + ", Balance: 1500 USD";
    }
}
```

Middleware

Acts as a bridge, fetching and converting legacy data into a modern format.

```
import org.json.JSONObject;

class Middleware {
    private LegacySystem legacySystem;

    public Middleware() {
        this.legacySystem = new LegacySystem();
    }

    public JSONObject getAccountDataInModernFormat(String accountNumber) {
        // Fetching data from the legacy system
        String legacyData = legacySystem.getAccountData(accountNumber);

        // Splitting and processing legacy data
        String[] parts = legacyData.split(", ");
        String accNumber = parts[0].split(": ")[1];
        String balance = parts[1].split(": ")[1];

        // Formatting data into JSON
        JSONObject modernData = new JSONObject();
        modernData.put("accountNumber", accNumber);
        modernData.put("balance", balance);

        return modernData;
    }
}
```

Modern Application

Consumes data from the middleware in a format it understands.

```
public class BankingApp {
    public static void main(String[] args) {
        Middleware middleware = new Middleware();

        // Fetching account data through middleware
        String accountNumber = "987654321";
        JSONObject accountData =
middleware.getAccountDataInModernFormat(accountNumber);

        // Displaying processed data
        System.out.println("Modern Account Data: " +
accountData.toString(2));
    }
}
```

Execution and Output

Execution Steps

1. The modern application requests account data using an account number.
2. The middleware fetches the raw data from the legacy system.
3. Middleware processes and converts the data into JSON format.
4. The application displays the modernized account data.

Output

When you run the application, you'll see:

```
Modern Account Data: {
  "accountNumber": "987654321",
  "balance": "1500 USD"
}
```

Architectural Benefits of This Solution

1. **Separation of Concerns:** Middleware decouples the modern application from the legacy system.
2. **Scalability:** Middleware can evolve independently to support more formats or functionalities.
3. **Maintainability:** The legacy system remains untouched, reducing the risk of breaking old functionalities.

Key Takeaways

This approach demonstrates how middleware can resolve architectural issues without replacing legacy systems. It provides a path for modernization while ensuring continuity of operations.