SDA LAB ASSIGNMENT

SUBJECT

SOFTWARE DESIGN AND ARCHITECTURE

TEACHER

SIR MUKHTIAR ZAMIN

SUBMITTED BY

AIMAN SHABBIR

FA22-BSE-104

DEADLINE

1. 4. 2025

Department of Software Engineering

COMSATS University Islamabad

Abbottabad campus

COMSATS UNIVERSITY ABBOTTABAD

Question:

As you know that software version is updated whenever we face any architectural changes. These kinds of architectural changes have major impacts on the system. In the first part, you need to find 5 major architectural problems which were faced in software system and then either the module or the whole system was revamped to solved the issue or a set of issues.

Answer:

SOFTWARE APPLICATION:

Part (A)

Five Major Architectural Problems Faced by Netflix:

Netflix, as a global streaming platform, has undergone significant architectural changes over the years to address challenges and enhance scalability, reliability, and user experience. Below are five major architectural problems faced by Netflix and how they addressed them;

1. Monolithic Architecture Scalability Issues

- **Problem**: Initially, Netflix used a monolithic architecture, which became difficult to scale as the platform's user base grew rapidly. High coupling between components caused bottlenecks and downtime.
- **Solution**: Netflix transitioned to a **microservices architecture**, breaking down the monolith into smaller, independent services. Each service handles specific functionality, improving scalability and fault isolation.

2. Downtime Due to Single Points of Failure

- **Problem**: Netflix's reliance on a single data center posed a risk of total service outage during hardware failures or disasters.
- **Solution**: Netflix adopted a **cloud-based architecture** using Amazon Web Services (AWS). By distributing its services across multiple regions, Netflix ensured high availability and disaster recovery.

3. Data Latency and Performance Issues

- **Problem**: Increasing data latency impacted user experience, especially during peak streaming hours. Centralized databases were unable to handle the growing volume of data efficiently.
- **Solution**: Netflix implemented **distributed caching (e.g., using EVCache)** and **NoSQL databases** like Cassandra. This improved data access speeds and allowed for better horizontal scaling.

4. Complexity in Testing and Deployment

- **Problem**: The growing number of microservices made it challenging to test, deploy, and maintain consistent service quality.
- **Solution**: Netflix introduced the **Simian Army tools** (e.g., Chaos Monkey) to test the resilience of their system. They also implemented **CI/CD pipelines** for automated testing and deployment, ensuring faster and safer rollouts.

5. Personalization and Recommendation Limitations

- **Problem**: Early recommendation systems were less effective due to a lack of efficient algorithms and infrastructure for processing large-scale user data in real-time.
- **Solution**: Netflix developed advanced **machine learning models** using big data infrastructure like Apache Kafka and Apache Spark. These systems power personalized recommendations and predictive analytics.

Part (B)

In the second part you need to replicate one problem and then solve it with the help of coding example.

Answer:

Replicating and Solving One Problem:

(Scalability of Monolithic Architecture)

Problem: Monolithic systems often struggle to handle increased traffic and require significant re-engineering to scale horizontally. We'll replicate a basic scenario of a monolithic service and refactor it into microservices.

Step 1: Monolithic Example

```
public class MonolithicService {
    public void handleUserRequest(String userId) {
        System.out.println("Fetching user details for: " + userId);


    }
    public void handleStreamingRequest(String contentId) {
        System.out.println("Streaming content: " + contentId);
    }
    public static void main(String[] args) {
        MonolithicService service = new MonolithicService();
        service.handleUserRequest("user123");
        service.handleStreamingRequest("content456");
    }
}
```

Step 2: Refactor to Microservices

We will break the system into two services: a **UserService** and a **StreamingService**. These services will communicate via REST APIs.

1. **UserService.java:**

```
import static spark.Spark.*;


public class UserService {
    public static void main(String[] args) {
        port(8081); // Set the port for the service
        get("/user/:userId", (req, res) -> {
            String userId = req.params(":userId");
```

```
            return "User details for: " + userId;
        });
    }
}
```

2. **StreamingService.java:**

```
import static spark.Spark.*;

public class StreamingService {
    public static void main(String[] args) {
        port(8082); // Set the port for the service
        get("/stream/:contentId", (req, res) -> {
            String contentId = req.params(":contentId");
            return "Streaming content: " + contentId;
        });
    }
}
```

3. **Client.java (to test both services):**

```
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Scanner;

public class Client {
    public static void main(String[] args) {
        try {
            // Call UserService
            System.out.println("Connecting to UserService...");
            URL userServiceUrl = new URL("http://localhost:8081/user/user123");
            HttpURLConnection userServiceConnection = (HttpURLConnection)
userServiceUrl.openConnection();
```

```
            userServiceConnection.setRequestMethod("GET");
            Scanner userScanner = new Scanner(userServiceConnection.getInputStream());
            while (userScanner.hasNext()) {
                System.out.println(userScanner.nextLine());
            }
            userScanner.close();

            // Call StreamingService
            System.out.println("Connecting to StreamingService...");
            URL streamingServiceUrl = new URL("http://localhost:8082/stream/content456");
            HttpURLConnection streamingServiceConnection = (HttpURLConnection)
streamingServiceUrl.openConnection();
            streamingServiceConnection.setRequestMethod("GET");
            Scanner streamingScanner = new
Scanner(streamingServiceConnection.getInputStream());
            while (streamingScanner.hasNext()) {
                System.out.println(streamingScanner.nextLine());
            }
            streamingScanner.close();
        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

NOTE: The microservices code implement in Maven project not Ant project.