Lab 8

Part 4

Problem:
A problem with the API gateway is that it is a **single point of failure**.

Solution:
To solve this, we can use **multiple API gateway instances** and configure them with **load balancing and replication**, similar to Eureka replicas. Clients should connect through a load balancer (e.g., Ribbon, Nginx, AWS ELB) which will distribute traffic among several gateway instances. If one gateway fails, requests are routed to another, ensuring **high availability**.

Part 5 – Business Processes

Problem:
A problem with a microservice architecture is that it is **difficult to keep track of the business processes** that run across multiple microservices.

Solution:
We can solve this by implementing **process monitoring and tracing** tools. Examples:

- **Distributed tracing** (e.g., Spring Cloud Sleuth, Zipkin, Jaeger) to follow requests across microservices.
- **Centralized monitoring** (e.g., Prometheus + Grafana, ELK stack) to track workflows.
- **Event logs or orchestration engines** (like **Camunda** or **Temporal**) to visualize and manage the business processes across services.

This gives a clear overview of how processes flow through multiple services.

Part 5 – Interfaces

Problem:
A problem with a microservice architecture is that it is **difficult to keep the interfaces of the different microservices in sync with each other**.

Solution:
We can solve this by using **API contracts and versioning**:

- **Contract-first design** using tools like **OpenAPI/Swagger** to define shared API specifications.
- **Schema registry** (e.g., for messaging: Confluent Schema Registry with Avro/Protobuf).
- **Versioned APIs** to ensure backward compatibility.
- **Continuous integration tests** across services to check interface compatibility.

This ensures that if one microservice updates its interface, dependent services won't break