Objective & Scope:

The complex backend project I want to share was part of a summer internship at a financial technology company. The project's objective was to develop a real-time trade settlement system for processing stock trades in a highly secure and efficient manner. The scope included designing the architecture, implementing the backend components, and integrating them with existing systems while ensuring compliance with financial regulations.

Technology Used:

To achieve the project's objectives, we used a range of technologies. We chose Java as the primary programming language due to its robustness and compatibility with the existing codebase. We employed Spring Boot for building the microservices architecture, Kafka for message queuing, and Docker for containerization. For data storage and retrieval, we utilized PostgreSQL and Apache Cassandra for their scalability and data consistency properties.

Challenges & Impact:

Several significant challenges emerged during the project. Firstly, ensuring data consistency and low-latency processing in a high-throughput environment was a complex task. We had to design an event-driven architecture to handle real-time updates and maintain transactional integrity. Additionally, we faced regulatory compliance issues that necessitated stringent security measures and audit trails.

Furthermore, integrating the new system with the legacy infrastructure was challenging due to differences in data formats and communication protocols. Data mapping and transformation became a major hurdle. We also encountered performance bottlenecks that required extensive optimization.

The impact of these challenges was notable. Project timelines had to be extended, and additional resources were allocated to address the issues. We engaged in frequent code reviews and collaboration with experienced engineers to tackle the hurdles effectively.

Outcomes:

After months of dedicated work, the project was successfully completed. The real-time trade settlement system significantly enhanced the efficiency and reliability of trade processing. It reduced settlement times by over 50%, leading to substantial cost savings for the company. Moreover, the system met stringent regulatory requirements and passed compliance audits.

Reflecting on this experience, I gained insights into the importance of robust architecture and the complexities of handling financial data. It reinforced the value of teamwork and mentorship, as collaboration with more experienced colleagues was instrumental in overcoming challenges. Overall, this project provided me with a profound understanding of designing and implementing complex backend systems in the financial industry, and it highlighted the critical role of adaptability and problem-solving skills in the face of unforeseen challenges.

Task 2:

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.web.bind.annotation.\*;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.mongodb.repository.MongoRepository;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

import org.springframework.data.mongodb.repository.config.EnableMongoRepositories;

import javax.persistence.\*;

import java.sql.Timestamp;

import java.util.List;

@SpringBootApplication

@EnableJpaRepositories

@EnableMongoRepositories

public class EmployeeManagementApplication {

public static void main(String[] args) {

SpringApplication.run(EmployeeManagementApplication.class, args);

}

}

// Employee Entity for PostgreSQL

@Entity

class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String firstName;

private String lastName;

private String email;

private Long departmentId;

private Timestamp createdAt;

private Timestamp updatedAt;

// Getters and setters

}

// Department Entity for MongoDB

@Document(collection = "departments")

class Department {

@Id

private String departmentId;

private String departmentName;

private Timestamp createdAt;

private Timestamp updatedAt;

// Getters and setters

}

// Employee Repository for PostgreSQL

interface EmployeeRepository extends JpaRepository<Employee, Long> {

}

// Department Repository for MongoDB

interface DepartmentRepository extends MongoRepository<Department, String> {

}

@RestController

@RequestMapping("/employees")

class EmployeeController {

private final EmployeeRepository employeeRepository;

EmployeeController(EmployeeRepository employeeRepository) {

this.employeeRepository = employeeRepository;

}

@GetMapping

List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

@PostMapping

Employee createEmployee(@RequestBody Employee employee) {

return employeeRepository.save(employee);

}

@PutMapping("/{id}")

Employee updateEmployee(@PathVariable Long id, @RequestBody Employee employee) {

// Implement update logic

return null;

}

@DeleteMapping("/{id}")

void deleteEmployee(@PathVariable Long id) {

employeeRepository.deleteById(id);

}

}

@RestController

@RequestMapping("/departments")

class DepartmentController {

private final DepartmentRepository departmentRepository;

DepartmentController(DepartmentRepository departmentRepository) {

this.departmentRepository = departmentRepository;

}

@GetMapping

List<Department> getAllDepartments() {

return departmentRepository.findAll();

}

@PostMapping

Department createDepartment(@RequestBody Department department) {

return departmentRepository.save(department);

}

@PutMapping("/{id}")

Department updateDepartment(@PathVariable String id, @RequestBody Department department) {

// Implement update logic

return null;

}

@DeleteMapping("/{id}")

void deleteDepartment(@PathVariable String id) {

departmentRepository.deleteById(id);

}

}