# ASSIGNMENT 1

**Student Name**  Abhishek Sharma

**Branch:** CSE

**Semester:** 6<sup>th</sup>

**Subject Name**: System Design

**UID**  23BCS12330

**Section/Group**  KRG_3A

**Subject Code:** 23CSH-314

**Q1 : Explain SRP and OCP in detail with proper examples.**

**Solution:**

**1. Single Responsibility Principle (SRP)**

**Definition**

*A class should have only one reason to change.*

This means **one class = one responsibility**.

**Why SRP is important**

- Easier maintenance
- Better readability
- Reduced impact of changes
- Improved testability
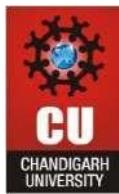
**Example (Violation)**

```
class Report {
    void generateReport() { }
    void saveToFile() { }
    void printReport() { }
}
```

One class handling **business logic + storage + printing**

**SRP-Compliant Version**

```
class ReportGenerator {
    void generateReport() { }
}
```

```
class ReportSaver {
    void saveToFile() { }
}

class ReportPrinter {
    void printReport() { }
}
```
<span style="color:green">Each class has **one responsibility**</span>

## 2. Open–Closed Principle (OCP)
**Definition**
*Software entities should be open for extension but closed for modification.*
**Why OCP is important**
- Avoids breaking existing code
- Supports scalability
- Improves reliability
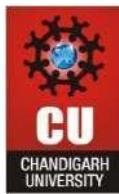
**Example (Violation)**
```
class Discount {
    double calculate(String type) {
        if(type.equals("Student")) return 0.1;
        if(type.equals("Senior")) return 0.2;
        return 0;
    }
}
```

<span style="color:red">Adding a new discount requires modifying the class</span>

**OCP-Compliant Version**
```
interface Discount {
    double calculate();
}

class StudentDiscount implements Discount {
    public double calculate() { return 0.1; }
}
```

```
class SeniorDiscount implements Discount {
    public double calculate() { return 0.2; }
}
```
New discounts can be added without changing existing code

**Q2. Discuss in detail about the violations in SRP and OCP along with their fixes.**
**Solution :**

**SRP Violation**
Violation
- A class performs multiple tasks
- Example: Authentication + Logging + Database operations

Problems
- Difficult to maintain
- High coupling
- Hard to test

Fix
➢ Split responsibilities into separate classes

**OCP Violation**
Violation
- Use of if–else or switch for behavior changes
- Modifying existing code for new functionality

Problems
- Risk of introducing bugs
- Code becomes rigid

Fix
➢ Use interfaces, inheritance, and polymorphism

**Q3. Design an HLD for an Online Examination System applying these principles.**

**Functional Requirements:**
- User should be able to register and login
- System should support different roles (Student, Admin)

- Admin should be able to create and schedule exams
- Admin should be able to add/manage questions
- Student should be able to attempt exams online
- System should evaluate answers automatically
- System should generate results
- System should notify users about exam and results

**Non Functional Requirements:**

A. Scalability
- System should handle multiple students simultaneously during exams

B. Security
- Secure authentication and authorization
- Exam data should be protected

C. Performance
- Low response time during exam submission

D. Reliability
- No data loss during exam submission

E. Maintainability
- Easy to add new exam or evaluation types

**Core Entities:**
- User
- Exam
- Question
- Answer
- Result

**API Endpoints**

Auth & User Service APIs
- POST /login
- POST /register
- GET /users/{id}

Exam Management Service APIs
- POST /exams

- GET /exams

Question Bank Service APIs
- POST /questions
- GET /questions/{examId}

Evaluation Service APIs
- POST /evaluate
  E. Result Service APIs
- GET /results/{userId}
-

**High Level Diagram :**
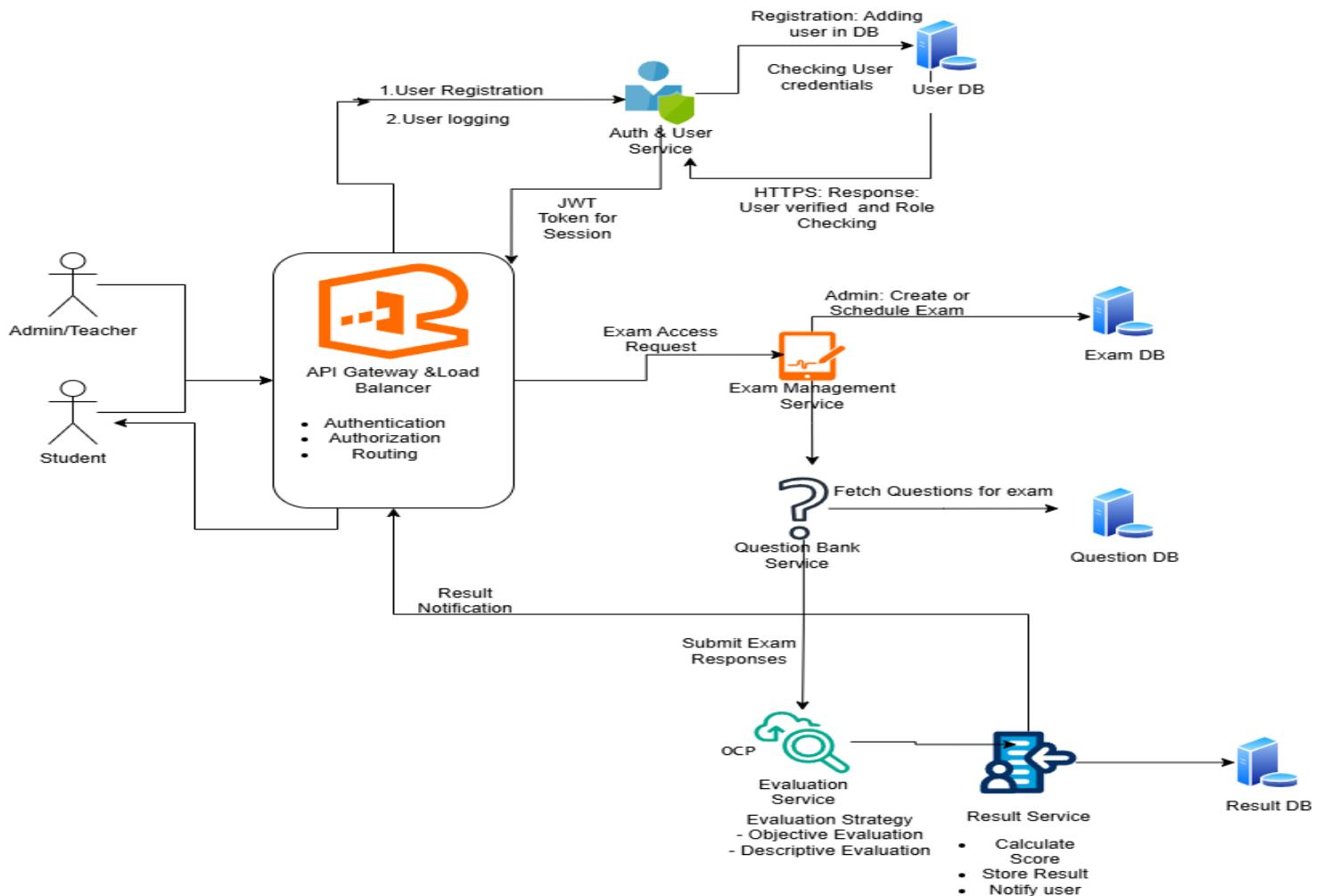


*Figure 1: HLD of Online Examination System*