جامعة مصر للمعلوماتية
EGYPT UNIVERSITY
OF INFORMATICS

Egypt University of Informatics

Computer and Information Systems

Software Engineering Course

Technical Report

Deliverable 5

Software Architecture & REST Web Services

Submitted by:

Nada Ashraf 22-101043

Aly Zaki 22-101096

Ahmed Waleed 22-101058

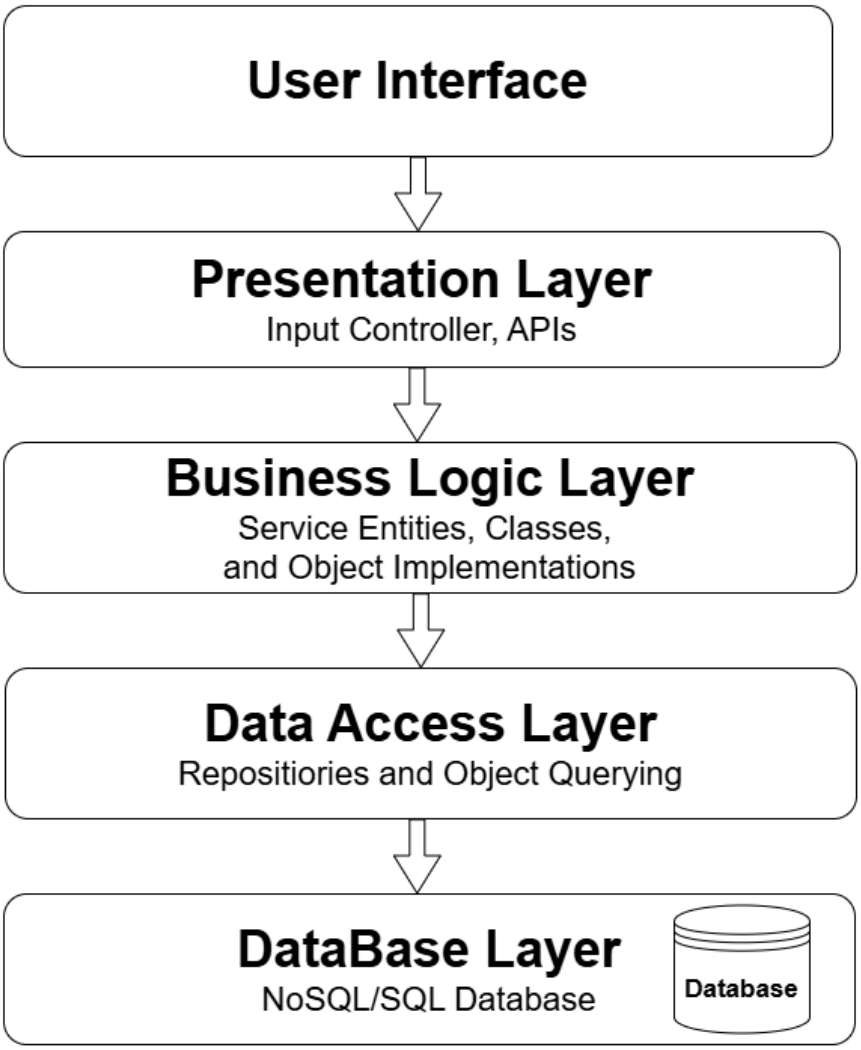
Omar Bayoumi 22-101022

Submission Date: 23rd May 2025

EduCertify's Architectural Patterns

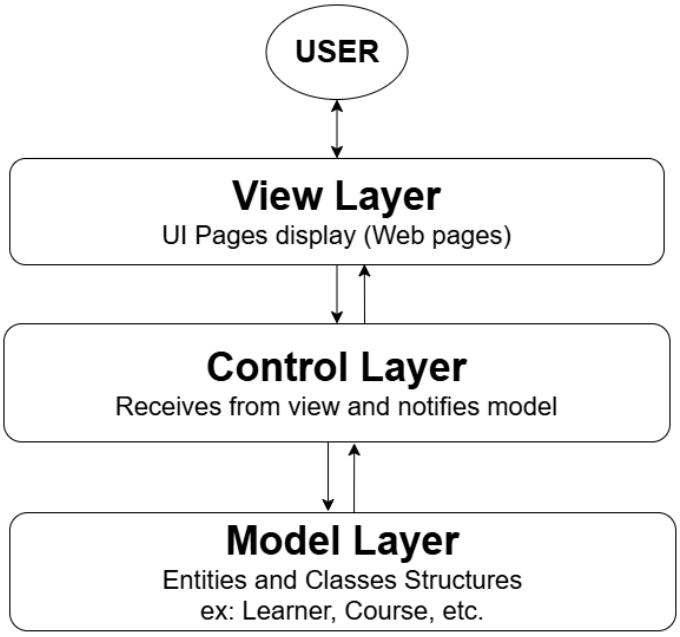
1) Layered (N-Tier) Architecture:

- EduCertify's back-end core.
- Organizes the system into distinct layers, each responsible for a specific set of functionalities.
    - Presentation Layer: User Interaction, API exposure.
    - Business Logic Layer: Implements system's core rules and workflows
    - Data Access Layer: Handles storage and retrieval from database
    - Database Layer: Physical data storage
- *Why Layered Architecture?*
    - Separation of concerns:
        - Each layer is responsible for a single functionality
        - Codebase clean and maintainable
    - Scalability
    - Security
        - Sensitive logic and data access are isolated from client.
    - Testability
        - Each layer can be tested separately
    - Reusability
- *Used Where?*
    - Core back-end system
    - User management
    - Course, enrollment, certificate processing



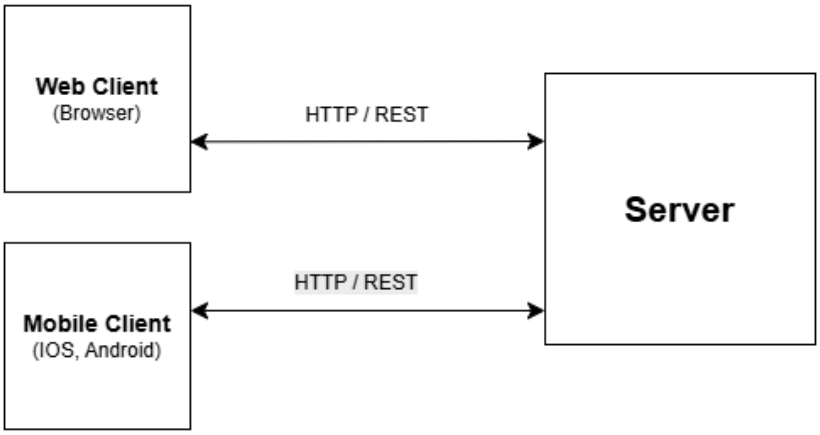
2) Model-View-Controller (MVC):

- Splits UI logic into three components
    - Model: Represent the data
    - View: User Interfaces (Web pages and mobile screens)
    - Controller: Handles input and updates the model and view.
- *Why MVC?*
    - Clear structure for our UI-heavy application
    - Easy to add and extend front-end features
    - Support multiple views (mobile and web) with shared logic
    - Better collaboration between front-end and back-end teams.
- *Used Where?*
    - Web portal
    - Mobile applications



3) Client-Server Architecture

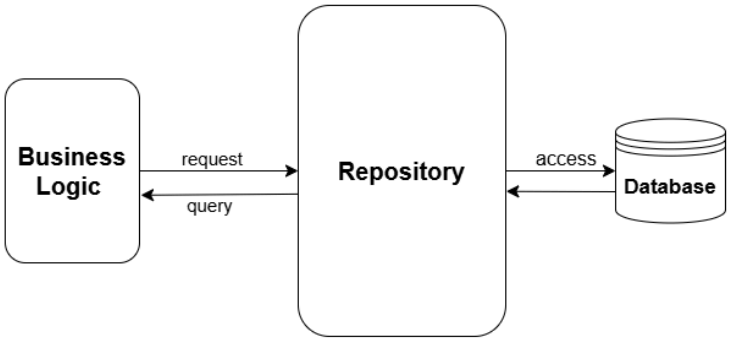
- The system is split into clients (user devices/ browsers/ mobile apps) and a centralized server which is the application backend.
- *Why Client-Server Architecture?*
    - Loose coupling:
        - Clients and servers can evolve independently
    - Scalability
        - Add more clients or scale the server separately
    - Distribution
        - Users can access the platform from anywhere they want
- *Used Where?*
    - EduCertify as a whole
        - Users can interact via web/mobile clients
        - Talk to the backend sever over (HTTP or REST APIs)
    - Third Part Integrations like payment methods



4) Repository Pattern

- The data access layer logic

- Decouple the business logic from the underlying database
- *Why Used?*
    - Swap databases easily without affecting other components
    - Centralized data access logic
    - Cleaner business logic layer
- *Used Where?*
    - Database Repository to access data of different domains



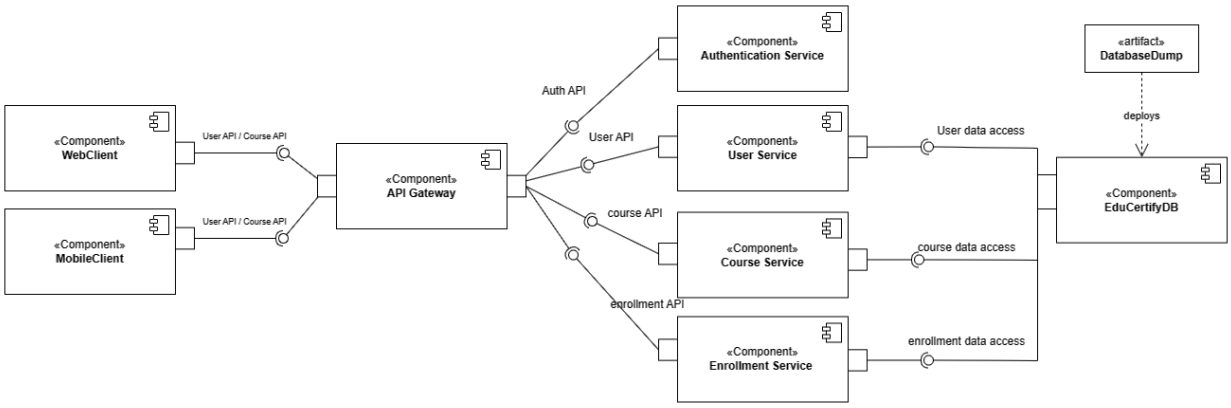
How All Architectural Patterns Work Together?

- *Client Server Architecture:*
    - The big picture in which all communication between front-end clients and centralized server take place.
- *Layered Architecture:*
    - The server is organized in layers
    - Presentation, business logic, data access, and database
- *MVC*
    - Presentation layer or front end uses MVC to structure user interaction
- *Repository:*
    - Business logic uses repositories to interact with physically centralized stored data.

Component Diagram – Development View:

- ***Purpose of Component Diagram:***
    - High level, structural view of EduCertify System.
    - Illustrates major software components
    - Their interactions through interfaces
    - Deployment artifacts required to any component
    - Clarify how system parts are communicating together
- ***Main Components***
    - API Gateway
        - Entry point to all client requests to route them to appropriate backend services.
        - Enable request filtering and centralized authentication
    - Authentication Service:
        - Handles user login, registration, password authentication
    - User Service
        - Manages user profiles, preferences
        - Provides user API interface for CRUD operations on user data
    - Course Service
        - Responsible for course creation, updating, module management
        - Provides the course API interface
    - Enrollment service
        - Handles leaner enrollment and course registration
        - Provides the enrollment API
    - EduCertify DB
        - Centralized database story persistent data separated from user, course, enrollment. (separated from logical)
    - Database Artifact:
        - The deployable SQL database to restore in the EduCertify DB
- ***Scenario***
    - Clients on the web or mobile interact via the API Gateway

- API Gateway delegates the client request to the backend services according to the specific function required (login, user account, course details, enrollment, etc.)
- The backend services perform CRUD or logical functions through the database
- The database is restored using the artifact to ensure quick deployment.



REST Web Services/APIs:

System Design:

- Handlers
    - Learner Handler, Certificate Handler, Course Handler
- Database
    - SQLite connection and operations
- Data Format
    - JSON for request and response
- Database Schema
    - Schema of student, courses, certificate tables in the database

API Endpoints:

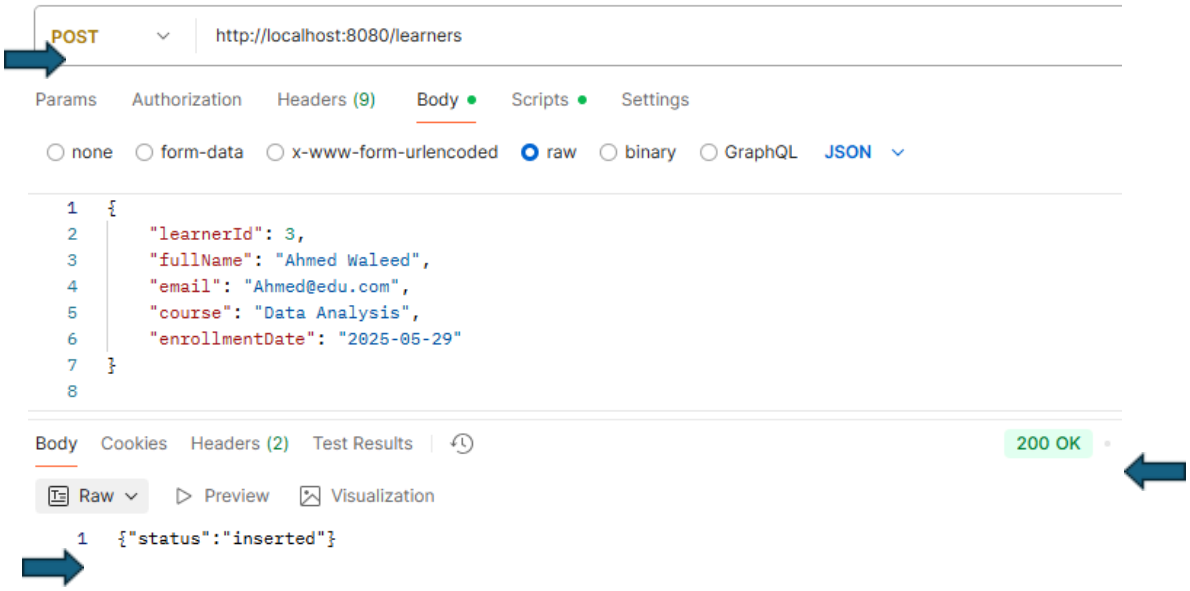
- GET and Post Endpoints
- GET Students: fetch all students
- POST Students: Add a new student
- GET Courses: fetch all courses
- POST Courses: Add a new course
- GET Certificates: fetch all certificates
- POST Certificates: Add a new certificate

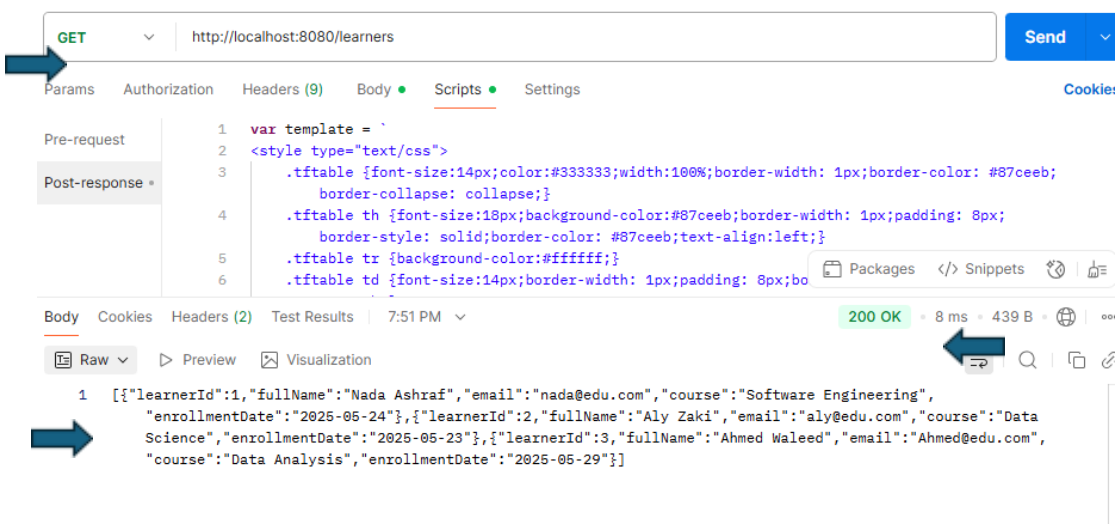
Testing API:

- Tool: Postman used to test the API
    - Successful testing = 200 OK responses

Successful Testing on Postman:

Learner (Get and Post)





Course (Get and Post)

POST http://localhost:8080/courses

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "courseId": 1,
    "courseName": "Java Programming",
    "courseDescription": "Learn the basics of Java programming.",
    "duration": "3 months",
    "price": 100.0
}
```

Body Cookies Headers (2) Test Results 200 OK

Raw Preview Visualize

```
{"status":"inserted"}
```

GET http://localhost:8080/courses Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Pre-request

Post-response

```
var template = `
<style type="text/css">
    .tftable {font-size:14px;color:#333333;width:100%;border-widt
```

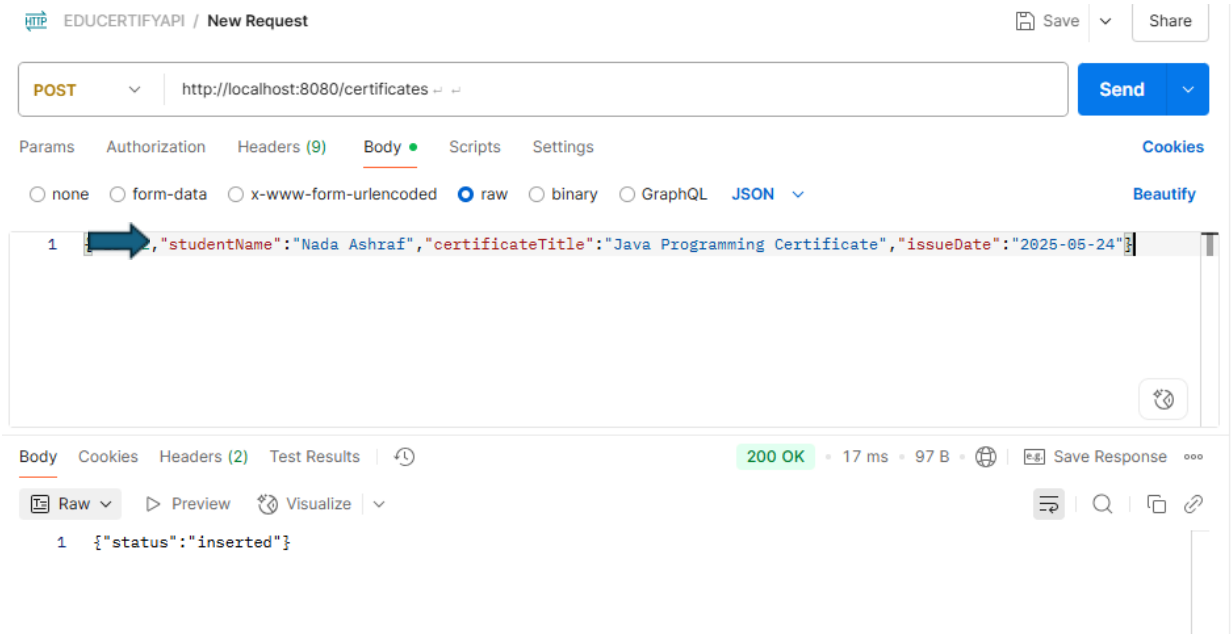
Packages Snippets

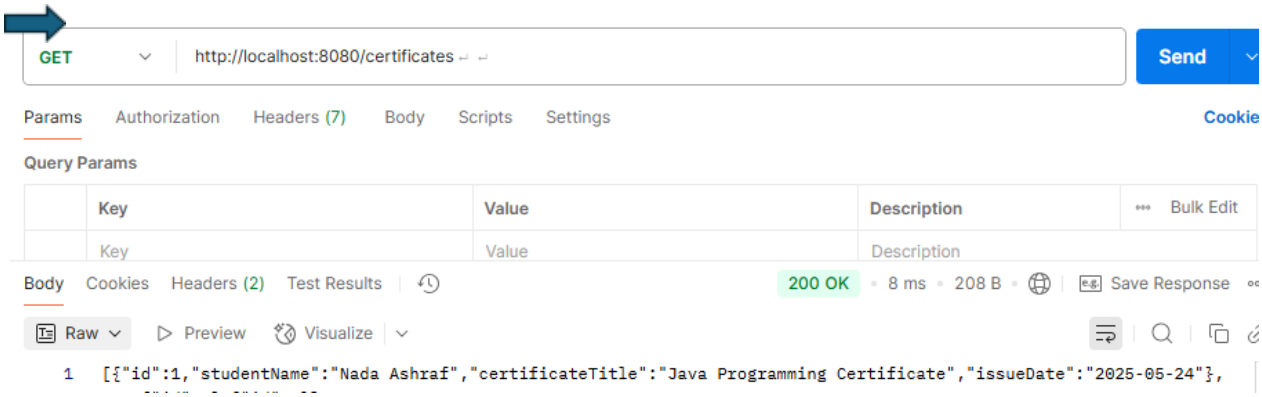
Body Cookies Headers (2) Test Results 200 OK 64 ms 521 B

Raw Preview Visualization

Course ID	Course Name	Course Description	Duration	Price
1	Java Programming	Learn the basics of Java programming.	3 months	100
2	Python Programming	Learn the basics of Python programming.	2 months	80
3	Software Engineering	Learn the concepts of Software Engineering.	5 months	150

Certificate (Get and Post)





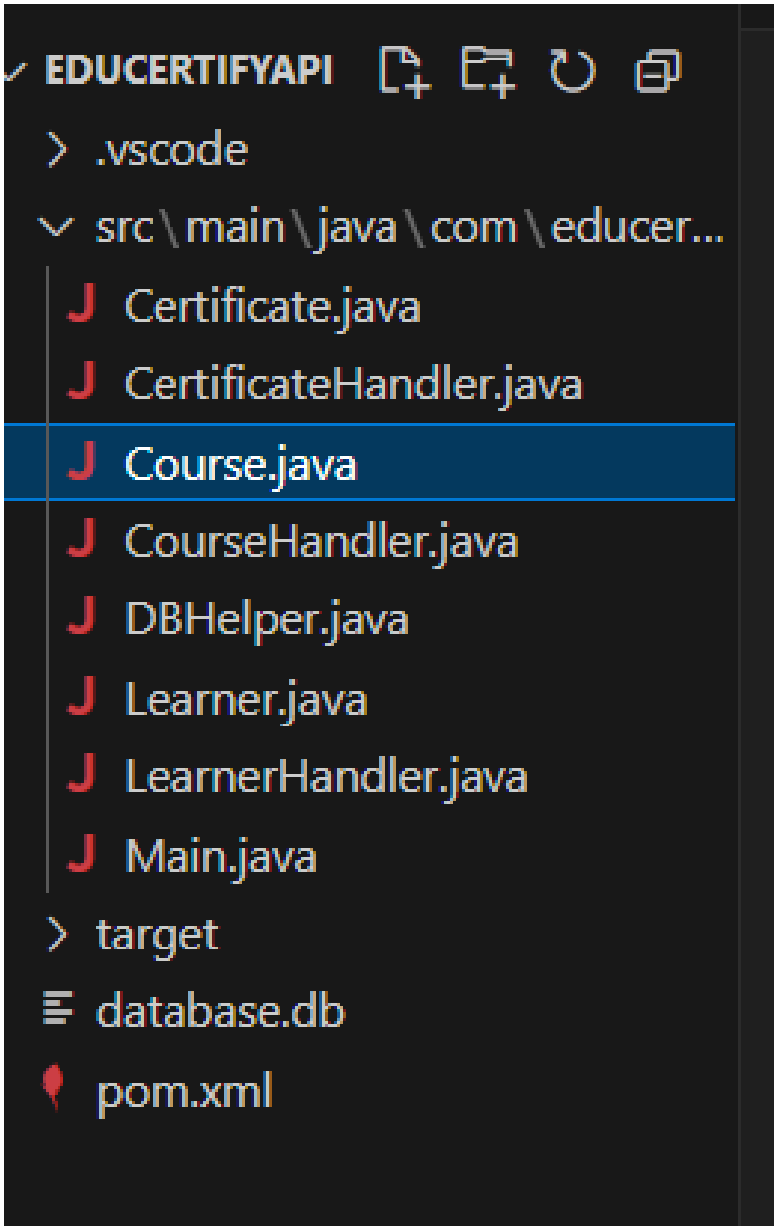
Snippets of the Code

POM:

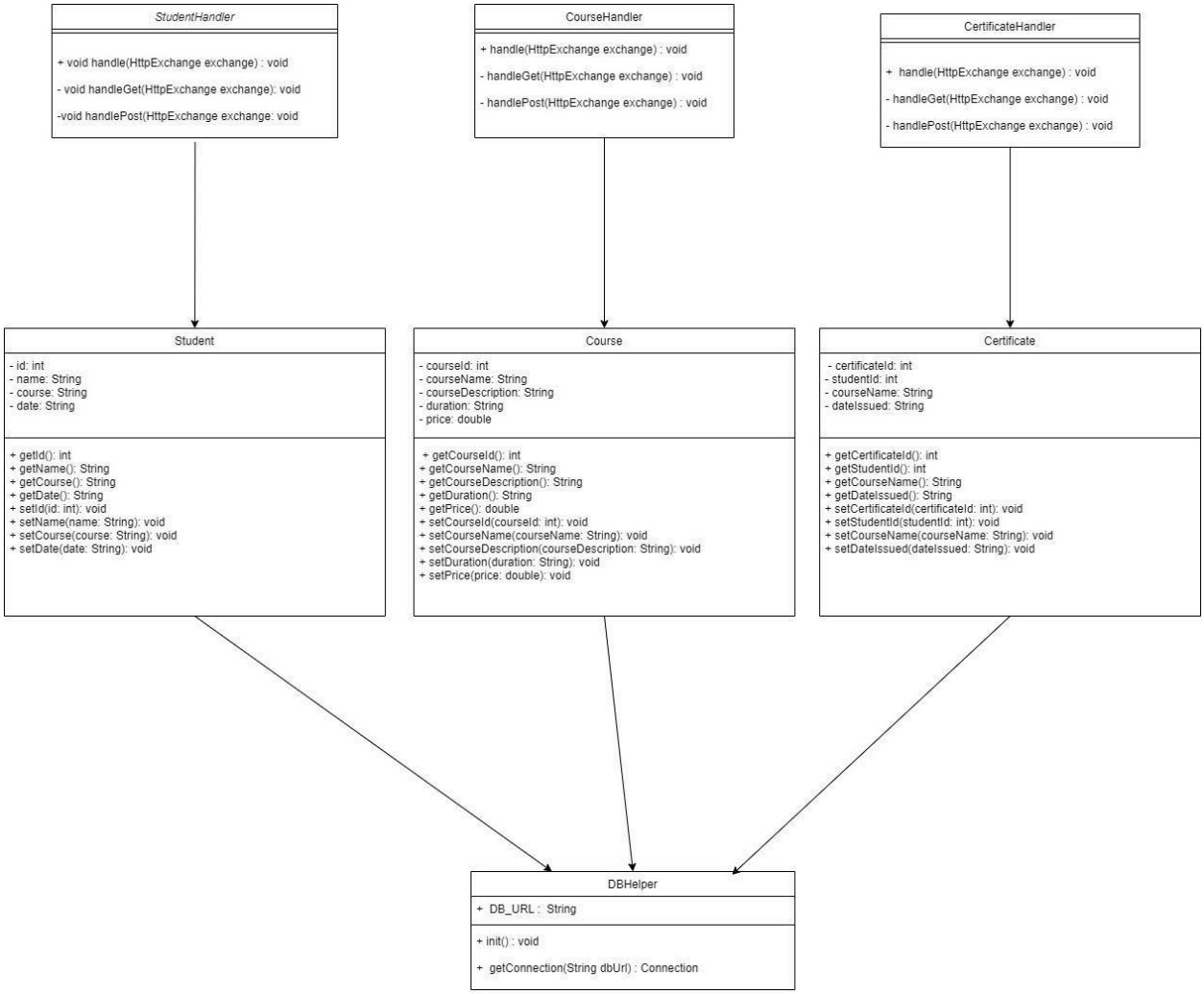
```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/m
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.educertify</groupId>
    <artifactId>educertify-api</artifactId>
    <version>1.0</version>

    <dependencies>
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.10.1</version>
        </dependency>

        <dependency>
            <groupId>org.xerial</groupId>
            <artifactId>sqlite-jdbc</artifactId>
            <version>3.42.0.0</version>
        </dependency>
    </dependencies>
</project>
```

Main:

```java
import java.net.InetSocketAddress;

public class Main {

    Run | Debug
    public static void main(String[] args) throws Exception {
        DBHelper.init();  // Initialize the database
        HttpServer server = HttpServer.create(new InetSocketAddress(port:8080), backlog:0);

        // Learner Handler
        server.createContext(path:"/learners", new LearnerHandler());

        // Certificate Handler
        server.createContext(path:"/certificates", new CertificateHandler());

        // Course Handler
        server.createContext(path:"/courses", new CourseHandler());

        server.setExecutor(executor:null); // Creates a default executor
        server.start();
        System.out.println(x:"Server started on http://localhost:8080");
    }
}
```


EDUCERTIFYAPI
.vscode
src\main\java\com\educer...
Certificate.java
CertificateHandler.java
Course.java
CourseHandler.java
DBHelper.java
Learner.java
LearnerHandler.java
Main.java
target
database.db
pom.xml

UML


StudentHandler
+ void handle(HttpExchange exchange) : void
- void handleGet(HttpExchange exchange): void
-void handlePost(HttpExchange exchange: void
CourseHandler
+ handle(HttpExchange exchange) : void
- handleGet(HttpExchange exchange) : void
- handlePost(HttpExchange exchange) : void
CertificateHandler
+ handle(HttpExchange exchange) : void
- handleGet(HttpExchange exchange) : void
- handlePost(HttpExchange exchange) : void
Student
- id: int
- name: String
- course: String
- date: String
+ getId(): int
+ getName(): String
+ getCourse(): String
+ getDate(): String
+ setId(id: int): void
+ setName(name: String): void
+ setCourse(course: String): void
+ setDate(date: String): void
Course
- courseId: int
- courseName: String
- courseDescription: String
- duration: String
- price: double
+ getCourseId(): int
+ getCourseName(): String
+ getCourseDescription(): String
+ getDuration(): String
+ getPrice(): double
+ setCourseId(courseId: int): void
+ setCourseName(courseName: String): void
+ setCourseDescription(courseDescription: String): void
+ setDuration(duration: String): void
+ setPrice(price: double): void
Certificate
- certificateId: int
- studentId: int
- courseName: String
- dateIssued: String
+ getCertificateId(): int
+ getStudentId(): int
+ getCourseName(): String
+ getDateIssued(): String
+ setCertificateId(certificateId: int): void
+ setStudentId(studentId: int): void
+ setCourseName(courseName: String): void
+ setDateIssued(dateIssued: String): void
DBHelper
+ DB_URL : String
+ init() : void
+ getConnection(String dbUrl) : Connection