


SWER313 Project

Step 1: Grading Rubric – Monolithic Backend Development

1. Functional Completeness (30 points)

✓ Does the backend fully implement the required functionalities?


Criteria	Points
User authentication (JWT/OAuth2) with role-based access control (Admin, Instructor, Student)	5
Course management (CRUD operations)	5
Enrollment system (students can enroll/unenroll and track progress)	5
Content management (upload/view course materials like PDFs/videos)	5
Assessment module (create quizzes, store and retrieve scores)	5
Notification system (send emails/SMS for course updates)	5

 **Deductions:** Partial implementations will receive proportionate deductions.

2. Software Architecture & Design Patterns (10 points)

✓ Did the student follow industry best practices for software architecture?

Criteria	Points
Correct use of Service Layer Pattern (business logic is not directly in controllers)	5
Implementation of DTOs (Data Transfer Objects) to decouple API models from database models	5

 **Deductions:**

✗ Business logic placed inside controllers instead of a service layer.

✗ No use of DTOs, directly exposing database models in APIs.

3. Code Quality & Maintainability (15 points)

✓ Is the code well-structured, modular, and maintainable?


Criteria	Points
Well-structured and reusable code	5
Meaningful variable and function names	5
Proper use of exception handling and logging (e.g., @ControllerAdvice for global exception handling)	5

 **Deductions:** Spaghetti code, poor naming conventions, lack of modularity.

4. Security & Authentication (10 points)

✓ Does the application implement proper authentication and security practices?


Criteria	Points
Secure authentication using JWT/OAuth2	5
Role-based access control (RBAC) properly enforced	5

 **Deductions:** Security misconfigurations, hardcoded secrets, or weak authentication mechanisms.

5. Database Design & API Documentation (15 points)

✓ Is the database well-designed, and are the APIs documented?

Criteria	Points
Well-structured relational database schema (tables, relationships)	5
Proper use of primary and foreign keys	5
API documentation provided (Swagger/Postman collection)	5

 **Deductions:** Lack of proper relationships, poor normalization, missing API documentation.

6. GitHub Usage & Version Control (20 points)

✓ Did the student follow proper GitHub best practices?

Criteria	Points
Frequent commits (commit early and often, rather than large commits)	5
Meaningful commit messages that clearly describe changes	5
Proper use of branches (e.g., feature/course-management, fix/authentication) instead of committing everything to main	5
Well-structured repository with README.md, .gitignore, and proper directory structure	5

 **Deductions:**

- ✗ Committing everything at the last minute (lack of commit history).
 - ✗ Vague commit messages ("Fixed stuff", "Final version", etc.).
 - ✗ Working only on the main branch without using feature branches.
-

Final Notes

- ✓ Late submissions will receive a deduction of 2% per day.
- ✓ Plagiarism or unauthorized code-sharing will result in a zero for the assignment.
- ✓ Students should submit a README.md explaining how to set up and test the backend.

Total Score Calculation:

Category	Max Points
Functional Completeness	30
Software Architecture & Design Patterns	10
Code Quality & Maintainability	15
Security & Authentication	10
Database Design & API Documentation	15
GitHub Usage & Version Control	20
Total	100