SE3040 – Application Frameworks
BSc (Hons) in Information Technology
Specialized in Software Engineering
3rd Year
Faculty of Computing
SLIIT

2024 – Assignment01

Building a Secure RESTful API for a University Timetable Management System

- **20 Marks**

This assignment requires you to develop a RESTful API for managing a university's timetable system. The system should facilitate the creation, modification, and querying of class schedules for students, faculty, and administrative staff. Emphasizing secure access, data integrity, and user-friendly interfaces, this project aims to simulate real-world software development challenges and solutions within an educational institution context.

Objectives:

- Develop a RESTful API using either Express JS (Node.js) or Spring Boot (Java).
- Use MongoDB to store and manage timetable, courses, users, and other related data.
- Implement secure authentication and authorization mechanisms.
- Apply software development best practices to ensure code quality, maintainability, and application scalability.

Functional Requirements:

1. **User Roles and Authentication**:
    - Define multiple user roles (e.g., Admin, Faculty, Student) with different access levels.
    - Implement secure login functionality and session management using JWT.
2. **Course Management**:
    - Allow CRUD operations on courses, including course name, code, description, and credits.
    - Enable Admins to assign Faculty to courses.
3. **Timetable Management**:
    - Facilitate the creation and modification of weekly timetables for different courses.
    - Include functionality to add, update, and delete class sessions, specifying the course, time, faculty, and location.
4. **Room and Resource Booking**:

- Manage classrooms and resources (e.g., projectors, labs) availability.
- Allow booking of rooms and resources for classes or events, ensuring no overlaps.

5. **Student Enrollment**:
   - Enable students to enroll in courses and view their timetables.
   - Allow Faculty and Admins to view and manage student enrollments in courses.
6. **Notifications and Alerts**:
   - Implement a system to notify users of timetable changes, room changes, or important announcements.

Non-Functional Requirements:

1. **Security**:
   - Secure API endpoints based on user roles.
   - Protect sensitive data through encryption and security practices.
2. **Database Design**:
   - Design an efficient and scalable MongoDB schema for managing timetables, courses, users, and bookings.
   - Ensure data integrity and consistency across the application.
3. **Code Quality and Documentation**:
   - Adhere to REST API design principles and coding standards.
   - Document the API comprehensively using tools like Swagger or Postman.
4. **Error Handling and Logging**:
   - Implement robust error handling for a smooth user experience and debugging.
   - Log critical information for audit and diagnostic purposes.

Evaluation Criteria:

- **Functionality**: Completeness and correctness of the API functions as per requirements.
- **Code Quality**: Clarity, maintainability, and adherence to best practices in coding and API design.
- **Security**: Effectiveness of authentication, authorization, and data protection mechanisms.
- **Database Design**: Logical structure, normalization, and performance of the MongoDB schema.
- **Documentation**: Clarity and completeness of the API documentation.

- **Viva Performance**: Depth of understanding demonstrated in explaining design choices, problem-solving approaches, and technology utilization.

Additional Testing Requirements:

1. **Unit Testing**:
    - Implement unit tests for individual components and functions to validate their behavior in isolation.
    - For Express JS, use testing libraries such as Jest or Mocha with Chai. For Spring Boot, use JUnit and Mockito.
2. **Integration Testing**:
    - Conduct integration tests to ensure different parts of the application work together seamlessly. This includes testing interactions between controllers, services, and the MongoDB database.
    - Test API endpoints to verify correct handling of requests and responses, including error scenarios.
3. **Security Testing**:
    - Perform security tests to identify vulnerabilities within your application, such as SQL injection, cross-site scripting (XSS), and insecure authentication.
    - Tools like OWASP ZAP or Burp Suite can be used for automated security testing.
4. **Performance Testing**:
    - Evaluate the API's performance under various loads to ensure it can handle multiple requests simultaneously without significant latency.
    - Tools like JMeter (for Spring Boot applications) or artillery.io (for Express JS applications) can be used for performance testing.

Submission Guidelines:

- Deadline: 24th March at midnight.
- Submit your source code through GitHub. You must clone the git repository that will be created when you open the following link and push your work to that repository.
    - https://classroom.github.com/a/MhkFIDKy
- Include a README file detailing setup instruction, API endpoint documentation, and how to run the tests. Document any setup required for testing environments, especially for integration and performance tests.

- Prepare for your viva examination by being ready to discuss your project's architecture, challenges faced, and how you addressed them.

This project aims to equip you with the skills needed to tackle complex software development challenges, focusing on backend services with real-world applications. Your ability to design, implement, and secure a RESTful API will be critical in your development as a software engineer. Good luck!

Marking Rubric

Criteria	**Sub-Criteria**	**Description**	**Marks**
Functional Requirements	User Roles and Authentication	Fully implemented and secure	**2**
		Partially implemented with minor issues	1
		Not implemented or major issues	0
	Course Management	Complete CRUD operations with Admin and Faculty functionalities	**1**
		Partial implementation or missing functionalities	0.5
		Not implemented	0
	Timetable Management	Fully functional with all capabilities	**2**
		Partially functional with some features missing	1
		Not implemented or major issues	0
	Room and Resource Booking	Complete booking system without overlaps	**1**
		Partial implementation or minor issues	0.5
		Not implemented	0
	Student Enrollment	Complete enrollment and viewing functionalities	**1**
		Partial implementation or issues	0.5
		Not implemented	0
	Notifications and Alerts	Fully functional notification system	**1**

		Partial implementation or issues	0.5
		Not implemented	0
Non-Functional Requirements	Security	Comprehensive security measures in place	**1**
		Basic security measures in place	0.5
		Lacking security measures	0
	Database Design	Efficient, scalable, and normalized schema	**1**
		Functional but with minor design flaws	0.5
		Poorly designed schema	0
	Code Quality and Documentation	High-quality code with comprehensive documentation	**1**
		Good code quality with adequate documentation	0.5
		Poor code quality or insufficient documentation	0
	Error Handling and Logging	Robust error handling and logging	**1**
		Basic error handling and logging	0.5
		No or minimal error handling and logging	0
Testing	Unit Testing	Comprehensive unit tests covering all components	**2**
		Partial coverage or some tests failing	1
		No unit tests or major issues with tests	0
	Integration Testing	Comprehensive integration tests with all parts working seamlessly	**1**
		Partial integration tests or minor issues	0.5
		No integration tests	0
	Security Testing	Comprehensive security testing with no vulnerabilities	**1**
		Basic security testing performed with minor issues found	0.5

		No security testing or major vulnerabilities	0
Viva Performance	Understanding and Explanation	Excellent understanding and clear explanation	**2**
		Good understanding with some unclear explanations	1
		Poor understanding or explanation	0
	Technical Depth	Demonstrates deep technical knowledge and skills	**2**
		Shows adequate technical understanding with minor gaps	1
		Lacks technical depth or significant gaps in knowledge	0