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

2025 – Assignment 01

Building a Secure RESTful API for a Personal Finance Tracker System - 20 Marks

This assignment requires you to develop a RESTful API for managing a Personal Finance Tracker system. The system should facilitate users in managing their financial records, tracking expenses, setting budgets, and analyzing spending trends. Emphasizing secure access, data integrity, and user-friendly interfaces, this project aims to simulate real-world software development challenges and solutions within a financial management context.

Objectives:

- Develop a RESTful API using either Express JS (Node.js) or Spring Boot (Java).
- Use MongoDB to store and manage user, transaction, budget, and report 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 with specific functionalities:
  - **Admin:**
    - Manage all user accounts.
    - Oversee all transactions and reports.
    - Configure system settings (e.g., categories, limits).
  - **Regular User:**
    - Add, edit, and delete personal transactions.
    - Set and manage personal budgets.
    - View and generate reports for personal finances.

- Implement secure login functionality and session management using JWT.

2. Expense and Income Tracking:

- CRUD operations for income and expense entries.
- Categorize expenses (e.g., Food, Transportation, Entertainment).
- **Tag Transactions with Custom Labels:**
  - Allow users to assign custom tags to their transactions for better categorization and filtering.
  - Examples of tags include #vacation, #work, or #utilities. Tags help users group transactions and analyze their spending patterns.
  - Implement functionality to filter and sort transactions by tags for detailed insights.
- **Add Support for Recurring Transactions:**
  - Enable users to define recurring transactions such as monthly subscriptions, rent, or salary.
  - Users should specify recurrence patterns (e.g., daily, weekly, monthly) and end dates if applicable.
  - Provide notifications for upcoming or missed recurring transactions.

3. Budget Management:

- Allow users to set monthly or category-specific budgets.
- Notify users when nearing or exceeding budgets.
- Provide budget adjustment recommendations based on spending trends.

4. Financial Reports:

- Generate reports for spending trends over time.
- Visualize income vs. expenses using charts or summaries.
- Include filters for specific time periods, categories, or tags.

5. Notifications and Alerts:

- Notify users about unusual spending patterns or important deadlines.

- Send reminders for bill payments or upcoming financial goals.

6. Goals and Savings Tracking:

- Allow users to set financial goals (e.g., saving for a car).
- Track progress toward goals with visual indicators.
- Enable automatic allocation of savings from income.

7. Multi-Currency Support:

- Enable users to manage finances in multiple currencies.
- Provide real-time exchange rate updates for accurate reporting.

8. Role-Based Dashboard:

- Provide a dashboard tailored to the user's role:
  - **Admin**: Overview of all users, total system activity, and financial summaries.
  - **Regular User**: Personalized summary of transactions, budgets, and goals.

Non-Functional Requirements:

1. **Security**:
   - Secure API endpoints based on user roles.
   - Protect sensitive data through encryption and secure coding practices.
2. **Database Design:**
   - Design an efficient and scalable MongoDB schema for managing users, transactions, budgets, and reports.
   - 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 mechanisms for a smooth user experience.

- Log critical information for audit and diagnostic purposes.
- Monitor logs for patterns and potential system issues.

Evaluation Criteria:

Criteria	Description
Functionality	Completeness and correctness of the API functions as per requirements.
Code Quality	Clarity, maintainability, and adherence to best practices in coding.
Security	Effectiveness of authentication, authorization, and data protection.
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: 11th of March 11.59 pm.**
- 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.
  - GitHub Classroom Link: https://classroom.github.com/a/xIbq4TFL
- Include a README file detailing setup instructions, 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.	**1**
		Partially implemented with minor issues	0.5
		Not implemented or major issues	0
	Expense and Income Tracking	Fully functional with all capabilities.	**1**
		Partial implementation or missing functionalities	0.5
		Not implemented	0
	Budget Management	Fully functional with all capabilities	**1**
		Partially functional with some features missing	1
		Not implemented or major issues	0
	Financial Reports	Complete and includes filtering options.	**1**
		Partial implementation or minor issues	0.5
		Not implemented	0
	Notifications and Alerts	Fully functional notification system.	**1**
		Partial implementation or issues	0.5
		Not implemented	0
	Goals and Savings Tracking	Complete tracking with visual indicators.	**1**
		Partial implementation or issues	0.5

		Not implemented	0
	Multi-Currency Support	Accurate exchange rate integration.	1
		Partial implementation or issues	0.5
		Not implemented	0
	Role-Based Dashboard	Tailored user dashboards.	
		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