Personal Finance Manager

System Design and Implementation Assignment

Problem Statement

This assignment requires the design and implementation of a comprehensive personal finance management system. The system shall enable users to effectively track their income, expenses, and savings goals through a robust web-based application. Users must be able to perform full CRUD operations on financial transactions, categorize them systematically, and generate detailed reports to analyze their spending patterns and financial behavior.

Functional Requirements

1. User Management and Authentication

Registration

Users shall be able to register with the following mandatory fields:

- Username: Valid email address
- Password: Secure password meeting system requirements
- Full Name: Complete user name
- Phone Number: Valid contact number

Login

Users shall authenticate using their username/email and password credentials.

Session Management

- Implementation shall utilize session-based authentication with secure cookies
- All API endpoints except registration and login require valid authentication

Logout

User sessions shall be properly expired and invalidated upon logout.

Data Isolation

Users shall only have access to their own data, ensuring complete data segregation between user accounts.

2. Transaction Management

Create Transaction

Users can add financial transactions with the following attributes:

- Amount: Positive decimal value
- Date: YYYY-MM-DD format, cannot be a future date
- Category: Must reference a valid category accessible to the user
- Description: Optional text description

Read Transactions

- View all transactions sorted by newest first
- Filter capabilities by date range, category, and transaction type

Update Transaction

Users can modify any transaction field except the date field.

Delete Transaction

Deleted transactions shall not be reflected in Savings Goals calculations and Reports.

3. Category Management

Default Categories

The system provides predefined categories that cannot be deleted or modified:

- INCOME: Salary
- EXPENSE: Food, Rent, Transportation, Entertainment, Healthcare, Utilities

Custom Categories

- Users can create custom categories with name and type (INCOME/EXPENSE)
- Custom category names must be unique per user
- Users can view and delete their custom categories
- Categories currently referenced by transactions cannot be deleted

Category Validation

All transactions must reference valid, non-deleted categories.

4. Savings Goals

Create Goal

Users can establish savings goals with the following parameters:

- Goal Name: Descriptive identifier

- Target Amount: Positive decimal value
- Target Date: Must be a future date
- Start Date: Defaults to creation date

Progress Tracking

- Progress calculation: (Total Income - Total Expenses) since goal start date
- Each goal tracks independently
- Display percentage completion and remaining amount

Goal Management

Users can view, update target amount/date, and delete goals as required.

5. Reports and Analytics

Monthly Reports

For specified month/year, display:

- Total income by category
- Total expenses by category
- Net savings (income - expenses)

Yearly Reports

Aggregate monthly data for specified year with comprehensive financial overview.

Technical Requirements

Technology Stack

Component	Technology
Programming Language	Java 17+ or Kotlin
Framework	Spring Boot 3.x
Security	spring-boot-starter-security

Testing Framework	JUnit 5, Mockito
Database	H2 (optional)
Build Tool	Maven or Gradle

Input Validation

- All API inputs must undergo comprehensive validation
- Return appropriate HTTP status codes (2xx for success, 4xx for client errors)
- Provide clear, descriptive error messages in response body

Error Handling

Status Code	Description
`400`	Bad Request (validation errors, malformed input)
`401`	Unauthorized (invalid credentials, expired session)
`403`	Forbidden (accessing other user's data)
`404`	Resource Not Found
`409`	Conflict (duplicate category names, etc.)

Note: No 5xx errors should occur for known scenarios.

Code Quality Expectations

Architecture

- Layered Architecture: Controller → Service → Repository
- DTOs: Separate request/response objects from entities
- Exception Handling: Global exception handler with @ControllerAdvice
- Configuration: Externalize configuration (application.properties/yml)

Testing

- Unit Tests: Minimum 80% code coverage
- Mocking: Mock external dependencies appropriately

Documentation

- README: Setup instructions, API documentation, design decisions
- JavaDoc: Document public methods and classes

Submission Requirements

Deliverables

1. GitHub Repository

Public GitHub repository (e.g., spring-boot-hello-world) containing:

- Source Code: Complete source code of the application
- Documentation: Comprehensive README with setup and usage guide
- Tests: Unit tests with 80% coverage

2. Deployment

Live public APIs accessible via provided URL (e.g., https://demo-ycl1.onrender.com/api)

- Deploy on Render or similar free hosting service.
  You can follow this tutorial here: How to host a Spring Boot application for free with Render
- Test deployment using the provided test script:
  `financial_manager_tests.sh`
  Note: it might take a couple of minutes to run the script for the first time.

```
bash financial_manager_tests.sh https://demo-ycl1.onrender.com/api
```

```
TEST EXECUTION SUMMARY
```

```
================================================

Base URL: https://demo-yc11.onrender.com/api
Total Tests Executed: 86
Tests Passed: 86
Tests Failed: 0
Success Rate: 100%

🎉 ALL TESTS PASSED! 🎉
The Personal Finance Manager API is working correctly.
```

Timeline

- Duration: 1 week from assignment date
- Submission: Email GitHub repository link and deployed URL with screenshot of test report to Fauzia Khan

API Specification

1. User Management and Authentication

Register User

POST `/api/auth/register`

Request:

```
{ "username": "user@example.com", "password": "password123", "fullName": "John Doe", "phoneNumber": "+1234567890" }
```

Response:

```
{ "message": "User registered successfully", "userId": 1 }
```

HTTP Status: `201 Created`, `400 Bad Request`, `409 Conflict`

Login

POST `/api/auth/login`

Request:

```
{ "username": "user@example.com", "password": "password123" }
```

Response:

```
{ "message": "Login successful" }
```

HTTP Status: `200 OK`, `401 Unauthorized`

Cookie: Provides session cookie for subsequent API calls

Logout

POST `/api/auth/logout`

Request: No body (uses session cookie)

Response:

```
{ "message": "Logout successful" }
```

HTTP Status: `200 OK`, `401 Unauthorized`

2. Transaction Management

Create Transaction

POST `/api/transactions`

Request:

```
{ "amount": 50000.00, "date": "2024-01-15", "category": "Salary",
"description": "January Salary" }
```

Response:

```
{ "id": 1, "amount": 50000.00, "date": "2024-01-15", "category": "Salary",
"description": "January Salary", "type": "INCOME" }
```

HTTP Status: `201 Created`, `400 Bad Request`, `401 Unauthorized`

Get Transactions

GET `/api/transactions`

Query Parameters:

```
?startDate=2024-01-01&endDate=2024-01-31&categoryId=1
```

Response:

```
{ "transactions": [ { "id": 1, "amount": 50000.00, "date": "2024-01-15",
"category": "Salary", "description": "January Salary", "type": "INCOME" } ] }
```

HTTP Status: `200 OK`, `401 Unauthorized`

Update Transaction

PUT `/api/transactions/{id}`

Request:

```
{ "amount": 60000.00, "description": "Updated January Salary" }
```

Response:

```
{ "id": 1, "amount": 60000.00, "date": "2024-01-15", "category": "Salary",
"description": "Updated January Salary", "type": "INCOME" }
```

HTTP Status: `200 OK`, `400 Bad Request`, `401 Unauthorized`, `404 Not Found`

Delete Transaction

DELETE `/api/transactions/{id}`

Response:

```
{ "message": "Transaction deleted successfully" }
```

HTTP Status: `200 OK`, `401 Unauthorized`, `404 Not Found`

3. Category Management

Get All Categories

GET `/api/categories`

Response:

```
{ "categories": [ { "name": "Salary", "type": "INCOME", "isCustom": false }, {
"name": "Food", "type": "EXPENSE", "isCustom": false }, { "name":
"CustomCategory", "type": "EXPENSE", "isCustom": true } ] }
```

HTTP Status: `200 OK`, `401 Unauthorized`

Create Custom Category

POST `/api/categories`

Request:

```
{ "name": "SideBusinessIncome", "type": "INCOME" }
```

Response:

```
{ "name": "SideBusinessIncome", "type": "INCOME", "isCustom": true }
```

HTTP Status: `201 Created`, `400 Bad Request`, `401 Unauthorized`, `409 Conflict`

Delete Custom Category

DELETE `/api/categories/{name}`

Response:

```
{ "message": "Category deleted successfully" }
```

HTTP Status: `200 OK`, `400 Bad Request`, `401 Unauthorized`, `403 Forbidden`, `404 Not Found`

4. Savings Goals

Create Goal

POST `/api/goals`

Request:

```
{ "goalName": "Emergency Fund", "targetAmount": 5000.00, "targetDate":
"2026-01-01", "startDate": "2025-01-01" }
```

Response:

```
{ "id": 1, "goalName": "Emergency Fund", "targetAmount": 5000.00, "targetDate":
"2026-01-01", "startDate": "2025-01-01", "currentProgress": 1000.00,
"progressPercentage": 20.0, "remainingAmount": 4000.00 }
```

HTTP Status: `201 Created`, `400 Bad Request`, `401 Unauthorized`

Get All Goals

GET `/api/goals`

Response:

```
{ "goals": [ { "id": 1, "goalName": "Emergency Fund", "targetAmount": 5000.00,
"targetDate": "2026-01-01", "startDate": "2025-01-01", "currentProgress":
1000.00, "progressPercentage": 20.0, "remainingAmount": 4000.00 } ] }
```

HTTP Status: `200 OK`, `401 Unauthorized`

Get Goal

GET `/api/goals/{id}`

Response:

```
{ "id": 1, "goalName": "Emergency Fund", "targetAmount": 5000.00, "targetDate": "2026-01-01", "startDate": "2025-01-01", "currentProgress": 1000.00, "progressPercentage": 20.00, "remainingAmount": 4000.00 }
```

HTTP Status: `200 OK`, `400 Bad Request`, `401 Unauthorized`, `403 Forbidden`, `404 Not Found`

Update Goal

PUT `/api/goals/{id}`

Request:

```
{ "targetAmount": 6000.00, "targetDate": "2026-02-01" }
```

Response:

```
{ "id": 1, "goalName": "Emergency Fund", "targetAmount": 6000.00, "targetDate": "2026-02-01", "startDate": "2025-01-01", "currentProgress": 1000.00, "progressPercentage": 16.67, "remainingAmount": 5000.00 }
```

HTTP Status: `200 OK`, `400 Bad Request`, `401 Unauthorized`, `403 Forbidden`, `404 Not Found`

Delete Goal

DELETE `/api/goals/{id}`

Response:

```
{ "message": "Goal deleted successfully" }
```

HTTP Status: `200 OK`, `401 Unauthorized`, `403 Forbidden`, `404 Not Found`

5. Reports and Analytics

Monthly Report

GET `/api/reports/monthly/{year}/{month}`

Response:


```
{ "month": 1, "year": 2024, "totalIncome": { "Salary": 3000.00, "Freelance": 500.00 }, "totalExpenses": { "Food": 400.00, "Rent": 1200.00, "Transportation": 200.00 }, "netSavings": 1700.00 }
```


HTTP Status: `200 OK`, `401 Unauthorized`

Yearly Report

GET `/api/reports/yearly/{year}`

Response:


```
{ "year": 2024, "totalIncome": { "Salary": 36000.00, "Freelance": 6000.00 }, "totalExpenses": { "Food": 4800.00, "Rent": 14400.00, "Transportation": 2400.00 }, "netSavings": 20400.00 }
```


HTTP Status: `200 OK`, `401 Unauthorized`