# Project Specification: Planetary System API with Security and GenAI Evaluation

## 🚀 Project Overview

In teams of two, you will build a Spring Boot application exposing a RESTful API ,with some GraphQL, that manages information about **planets and their moons**. The system will support full **CRUD operations**, additional **custom queries**, and enforce **role-based access control** using Spring Security Basic Authentication. The system will follow best practice and include centralised handling of exceptions, logging and security.

**Follow the development conventions demonstrated in class**, such as using `@ResponseStatus` not `ResponseEntity<>` in methods in the controller layer.

Once you have completed the manual implementation, you will **generate the same project using two AI tools.** You will **analyse and compare the code quality and experience**, then submit a report reflecting on the experience and results.

The manual implementation must be feature-complete.

GraphQL is a small part of this project. Your chosen GenAIs might generate full GraphQL services for all entities when prompted but **full GraphQL service is not required for manual implementation**. See API Endpoints for more information.

## Project Goals

- Master key Spring Boot development skills.
- Understand how generative AI can assist and impact software development.
- Develop critical thinking skills by comparing and contrasting manual and AI-generated code.

# 🔍 AI Comparison Component

***After* completing your manual implementation**:

1. Use **two different AI tools** (e.g., ChatGPT, GitHub Copilot, Perplexity, Claude or others) to generate the same application.
2. Document the exact prompts used. Begin with the same prompt for each GenAI tool then see what happens. Do not use the specification provided above – **generate your own initial prompt**.
3. Test the AI-generated code for correctness and completeness.
4. Write a **critical analysis report (~2000 words)** comparing:
    - Functionality coverage (REST and GraphQL)
    - Code quality
    - Best practices adherence
    - Security implementation
    - AOP logging implementation
    - Differences and similarities between your code and AI outputs.
    - Marks are awarded to students who clearly identify structural/code-quality differences and discuss *why* the AI made certain design choices or where it deviated from best practices.
5. Reflect on the strengths and limitations of AI-generated code versus human coding.
6. State which GenAI tool would be your preferred tool for Spring Boot coding.

Your **AI-generated codebases are expected to *compile and run*, but they may not be fully functional or correct**. The focus of this component is on your **critical evaluation** of the AI-generated output — its structure, adherence to best practices, and code quality — rather than on achieving a perfect implementation.

# 🛠️ Functional Requirements

## 1. Entities

*Planet*

- planet_id (Primary Key)
- name
- type (e.g., terrestrial, gas giant)
- radius_km
- mass_kg
- orbital_period_days

*Moon*

- moon_id (Primary Key)
- name
- diameter_km
- orbital_period_days
- planet_id (Foreign Key to Planet)

*User (for Security)*

- user_id (Primary Key)
- username (unique)
- password (hashed)
- role (enum: ADMIN, STAFF, STUDENT)
- Optional: enabled, timestamps for creation/update

**Preload sample data** to facilitate testing the application via Swagger and POSTMAN[1].

## 2. API Endpoints

### REST

#### Planets

- Add a new planet to the database.
- Retrieve all planets.
- Retrieve a planet by its unique ID.
- Update the details of an existing planet (e.g., change its mass).
- Remove a planet from the database by its unique ID.
- Retrieve planets based on their type (e.g., gas giant, terrestrial).

[1] You may use a GenAI tool to generate this sample data for the manual version as well as for the GenAI version.

- Retrieve specific fields of a planet (e.g., only name and mass_kg for use in other parts of the application).

### Moons

- Add a new moon to the database, ensuring the planet exists
- Retrieve a list of all moons.
- Retrieve a moon by its unique ID.
- Remove a moon from the database by its unique ID.
- List moons by planet name (i.e., retrieve all moons for a specific planet).
- Count the number of moons for a specific planet.

## GraphQL

### User

- Find user by ID (Query)
- Create user (Mutation)

# 3. Security Considerations

- Implement **Basic Authentication** with Spring Security.
- Enforce **role-based access control**:
    - ADMIN: Full access to all endpoints, including user management (GraphQL)
    - STAFF: Can create, update, delete planets and moons.
    - STUDENT: Read-only access to planets and moons.
- Passwords must be securely hashed (use **BCrypt** or similar).
- Secure all endpoints with appropriate role checks.
- Use a combination of **centralized URL-based security config** and **`@PreAuthorize` annotations** for fine-grained control.

# 4. Separation of Concerns (AOP)

Implement separation of concerns using

- Centralised exception handling
- Security handled centrally
- Using AspectJ for logging
    - Demonstrate 3 logging pointcuts

## Non-Functional Considerations

- Data coming through **the request body as Json should be validated** and exceptions handled centrally. Validate using Jakarta annotations  on the DTO and `@Valid`.
- Use **Spring Boot best practices**:
    - Separate layers: Controllers, Services, Repositories
    - Use **DTOs** for API input/output, not entities directly
    - Input **validation** with annotations (@NotNull, @Size, etc.)
    - **Centralized exception handling** with @ControllerAdvice
    - Use **custom JPA queries** (@Query) where needed for filtering or aggregation
- API documentation via **Swagger/OpenAPI**
- Proper logging and meaningful error responses

## 🚨 Miscellaneous

Unit and Integration tests would be provided in a real system. **Tests are not required for this project** as they would take too long to generate and verify.

This module focuses on API service only. It is not full stack with **no front-end required** to consume the API.

**I reserve the right to interview you about your project**, in particular your manually generated code. **Failure to attend the interview when called will result in 0%** for that part of the project.

## 🛠️ Recommended Technologies & Tools

- Maven Build Tool
- Java 17+
- Spring Boot 3.5.X[2]
- Spring Data JPA (Hibernate)

[2] Spring Boot 4 is set for release in November but DO NOT use it.

- H2 (for DB)
- Spring Boot graphql
- Swagger
- Spring Boot Starter Web (for REST etc)
- Spring Boot Starter Validation (for Jakarta)
- Spring Security (Basic Authentication)
- Actuator (access to meta data including mappings of endpoints)
- Swagger (springdoc-openapi)
- Lombok (IDE plugin, for boilerplate reduction)

## 📁 Deliverables

1. **Manual Implementation Codebase**

   Your hand-coded Spring Boot project implementing all functionality. Include details in README.md of any missing functionality.

2. **GenAI Implementation Codebase**

   Projects generated using one GenAI tool, created from prompts **you** design and document. These must compile and run but perhaps are not completely functional. Include details in README.md of any missing functionality.

3. **Technical Report** (~2000 words)

   A comparative analysis of AI generated code and manual code including your experience of using AI. Prompts are included as an Appendix and/or may be included in the main document.

## 📊 Assessment Criteria

| Criterion | Weight |
|---|---|
| Correctness, completeness and quality of your own code | 40% |
| Security implementation and role enforcement | 10% |
| AI code comparison and critical analysis | 40% |
| Quality of technical report | 10% |