**Assignment Review-Guidance**

Please upload the code to a **GitHub/Gitlab repository** and ensure that the repository is **publicly accessible to everyone**.

**Best Practices and Recommendations:**

1. **Use Lombok**: Use Lombok library to reduce boilerplate code by generating getters, setters, constructors.
2. **Implement Spring Validation**: Use Spring's built-in validation mechanisms to ensure data integrity and handle invalid inputs gracefully.
3. **Implement Error Handling and Logical Validations**: Proper error handling and logical validations are essential for robust and reliable applications. Implement appropriate error handling mechanisms and validate user inputs and business rules.
4. **Implement Additional Options for GET APIs**: Consider implementing additional options for GET APIs, such as sorting, ordering, and pagination, to enhance the API's functionality and usability.
5. **Maintain Proper Separation and Definition of Input/Output APIs**: Clearly define and separate the input and output APIs to maintain a clean and organized codebase.
6. **Use Correct Mapping of JPA Objects:** Ensure that your JPA entity mappings are correct and avoid duplicate mappings or unnecessary associations.
7. **Utilize Spring Data Specifications with Annotations**: Spring Data Specifications with annotations can provide a flexible and powerful way to build complex queries.
8. Follow Best Practices for Project Structure: Maintain a clean and organized project structure that adheres to industry best practices and coding conventions.
9. **Implement Proper Database Modeling**: Design and implement a well-structured and normalized database schema to support your application's requirements.
10. **Proper URL Conventions:** Follow industry-standard URL conventions and best practices for designing RESTful APIs.
11. Maintain Consistent Naming Conventions: Follow consistent naming conventions throughout your codebase to improve readability and maintainability.
12. Implement Proper Error Handling: Ensure that you handle errors correctly and provide meaningful error messages to the users or clients.
13. Optimize Database Queries: Ensure that your database queries are optimized and avoid fetching unnecessary data or performing inefficient operations.

**Pitfalls/Don’ts:**

1. Avoid Duplicating Validation Logic: Instead of duplicating validation logic in multiple places, consider centralizing and encapsulating it in dedicated classes or components.
2. Enforce Uniqueness Constraints at the Database Level: Ensure that unique constraints are enforced at the database level, rather than relying solely on application-level validations.
3. Avoid Unnecessary Nesting or Complexity in DTOs: Keep your Data Transfer Objects (DTOs) simple and avoid unnecessary nesting or complexity.
4. Avoid Verbose Coding: While clear and readable code is essential, avoid overly verbose coding practices that can make the codebase harder to maintain.

* Avoid Mapping DTOs Within Services: Follow the principle of separation of concerns and avoid mapping DTOs within service classes.

**Good to have:**

* Implement Containerization (e.g., Docker): Consider containerizing your application using Docker.