Тестове Завдання: Middle Java Developer

Тема: Система управління каталогом книжкового магазину (REST API).

Завдання: Розробити REST API для управління каталогом книжок.

1. Стек Технологій:

- **Мова:** Java 17+
- **Фреймворк:** Spring Boot (остання стабільна версія)
- **Збірка:** Maven або Gradle
- **База Даних:** H2 Database (in-memory, для простоти запуску)
- **Тестування:** JUnit 5, Mockito

2. Вимоги до Функціональності:

Створити REST API для об'єкта **Книга (Book)** з такими полями:

Поле	Тип	Опис
id	Long	Унікальний ідентифікатор (генерований)
isbn	String	Унікальний міжнародний стандартний номер книги (унікальний, не може бути null)
title	String	Назва книги (не може бути null)
author	String	Автор
publicationYear	Integer	Рік видання
price	Object	Вартість книги (не може бути null)

Структура **Вартість (Price):**

uah	Decimal	Вартість книги в UAH (не може бути null)
eur	Decimal	Вартість книги в EUR (розраховується автоматично за поточним курсом)

3. REST endpoints:

Реалізувати наступні RESTful endpoints:

Метод	**URI**	**Опис**	**HTTP-статус**
POST	/api/books	Створити нову книгу	201 Created
GET	/api/books	Отримати список всіх книг **(Додатково: Пагінація/Сортування)**	200 OK
GET	/api/books/{id}	Отримати книгу за ID	200 OK
PUT	/api/books/{id}	Оновити існуючу книгу за ID	200 OK
DELETE	/api/books/{id}	Видалити книгу за ID	204 No Content

4. Оновлення курсу EUR до UAH:

Створити таблицю для зберігання актуального курсу EUR до UAH з наступними полями:

date	TIMESTAMP	Дата оновлення курсу
rate	Decimal	Поточний курс

Для отримання курсу EUR до UAH необхідно використовувати публічний API НБУ.
Регламент отовлення - 1 раз на добу о 09:00.

5. Вимоги до Коду та Дизайну:

1. **Архітектура:** Дотримуватися класичної трирівневої архітектури (Controller -> Service -> Repository).
2. **Транзакції:** Правильно використовувати анотацію **@Transactional** у сервісному шарі.
3. **Валідація:** Реалізувати валідацію вхідних даних для створення/оновлення книги за допомогою **Jakarta Bean Validation** (наприклад, @NotNull, @Min, @Pattern).
4. **Обробка Винятків:** Реалізувати глобальну обробку винятків за допомогою **@ControllerAdvice** / **@RestControllerAdvice**. Зокрема, коректно обробити випадок, коли книга не знайдена (повернути **404 Not Found**).
5. **DTO/Record:** Використовувати **Data Transfer Objects (DTO)** або Java Records для вхідних (Request) та вихідних (Response) даних, щоб відокремити API-контракт від JPA-сутності.

6. Тестування (Обов'язкова Частина):

Реалізувати наступні типи тестів:

1. **Unit Тести (для Сервісного Шару):**
    - Створити тести для класу **BookService** з використанням **Mockito** для імітації (mocking) **BookRepository**.
    - Перевірити коректне створення книги.
    - Перевірити обробку ситуації, коли книга не знайдена при спробі оновлення чи отримання.
2. **Integration Тести (для Контролера):**
    - Створити один-два тести для **BookController** з використанням **@WebMvcTest** або **@SpringBootTest** + **MockMvc** для перевірки коректності HTTP-статусів та JSON-контракту.