**ЗВІТ**

**до лабораторної роботи № 6**  
на тему:  
**"Створення REST-сервісу з використанням анотації @RestController."**

**Виконала:**  
студентка групи **МІТ-21**  
**Йовхимищ Діана**

**Мета роботи:** Ознайомитися з основами створення REST-сервісів у Spring Boot, використовуючи @RestController. Навчитися реалізовувати CRUD-операції для взаємодії з базою даних PostgreSQL, використовуючи Spring Boot JPA та Lombok.

**Завдання:**

1. В git репозиторії створити гілку lab6.

2. Провести необхідне удосконалення front-end частини, створеної у лаб 4.

3. Розробити REST сервіс з використанням анотації @RestController:

1. забезпечити виконання CRUD операцій для збереження інформації в базі даних Postgres;
2. для роботи з базою даних використати Spring Boot JPA;
3. для створення Entity класу використати Lombok.

4. Забезпечити сумісну роботу розроблених у п.п. 2, 3 частин системи.

5. Результати викласти у репозиторій GitHub у гілці lab6.

6. Підготуватися до захисту проекту.

**Хід роботи:**

**Крок 1. Створення нової гілки у GitHub**

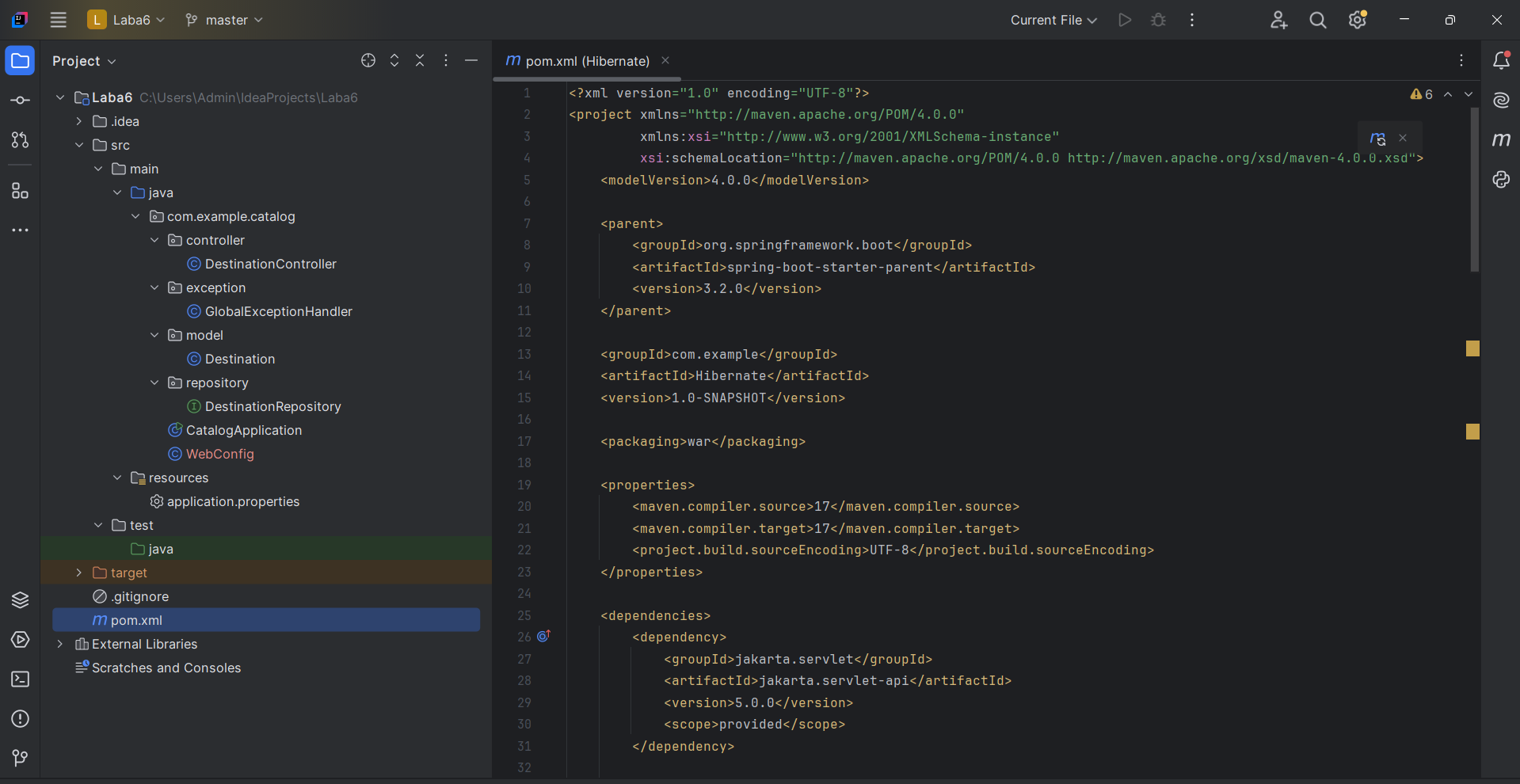
Перед початком роботи необхідно створити нову гілку lab6 у вашому репозиторії:

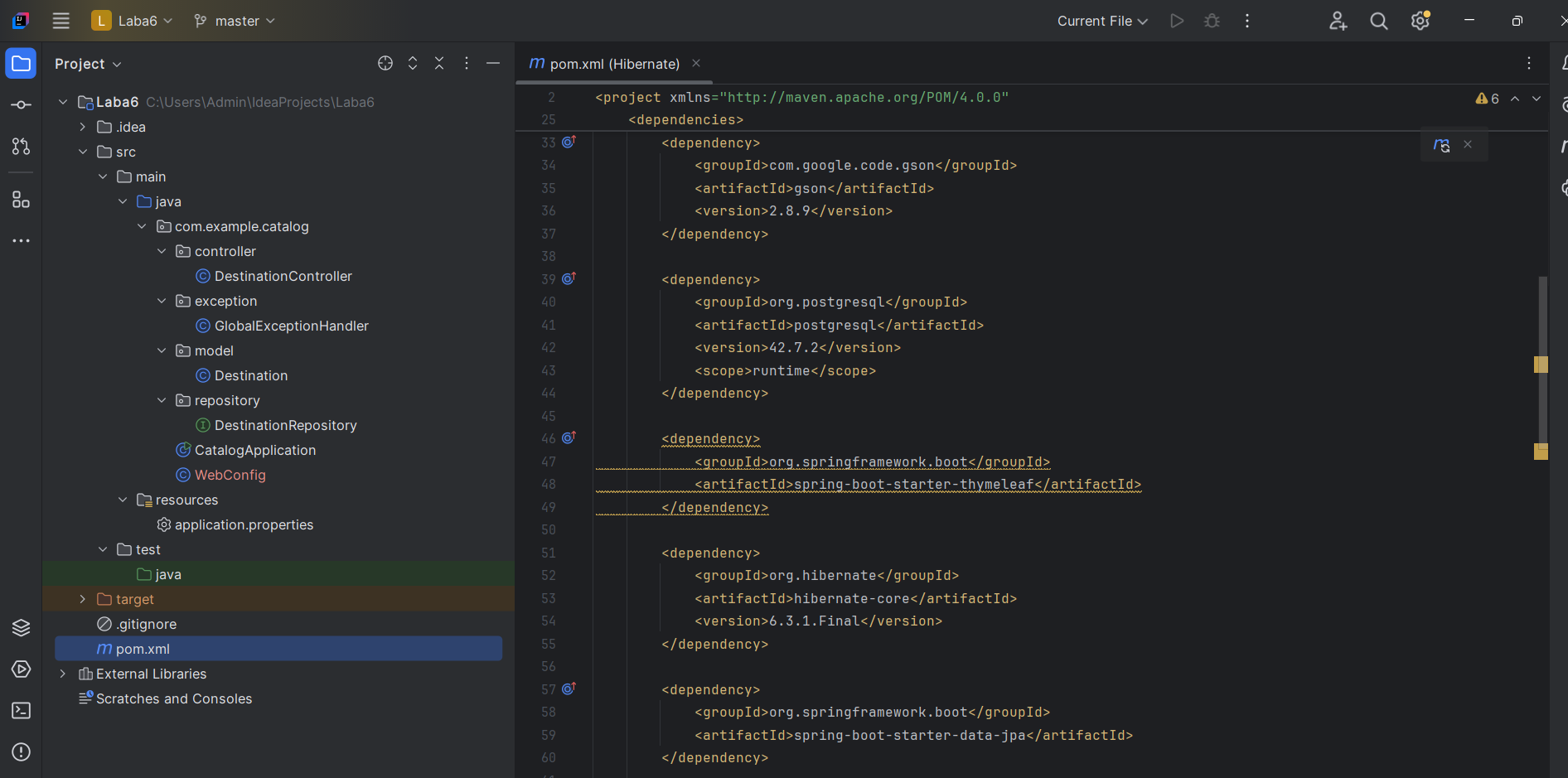
|  |
| --- |
| git checkout -b lab6 git push origin lab6 |
|  |

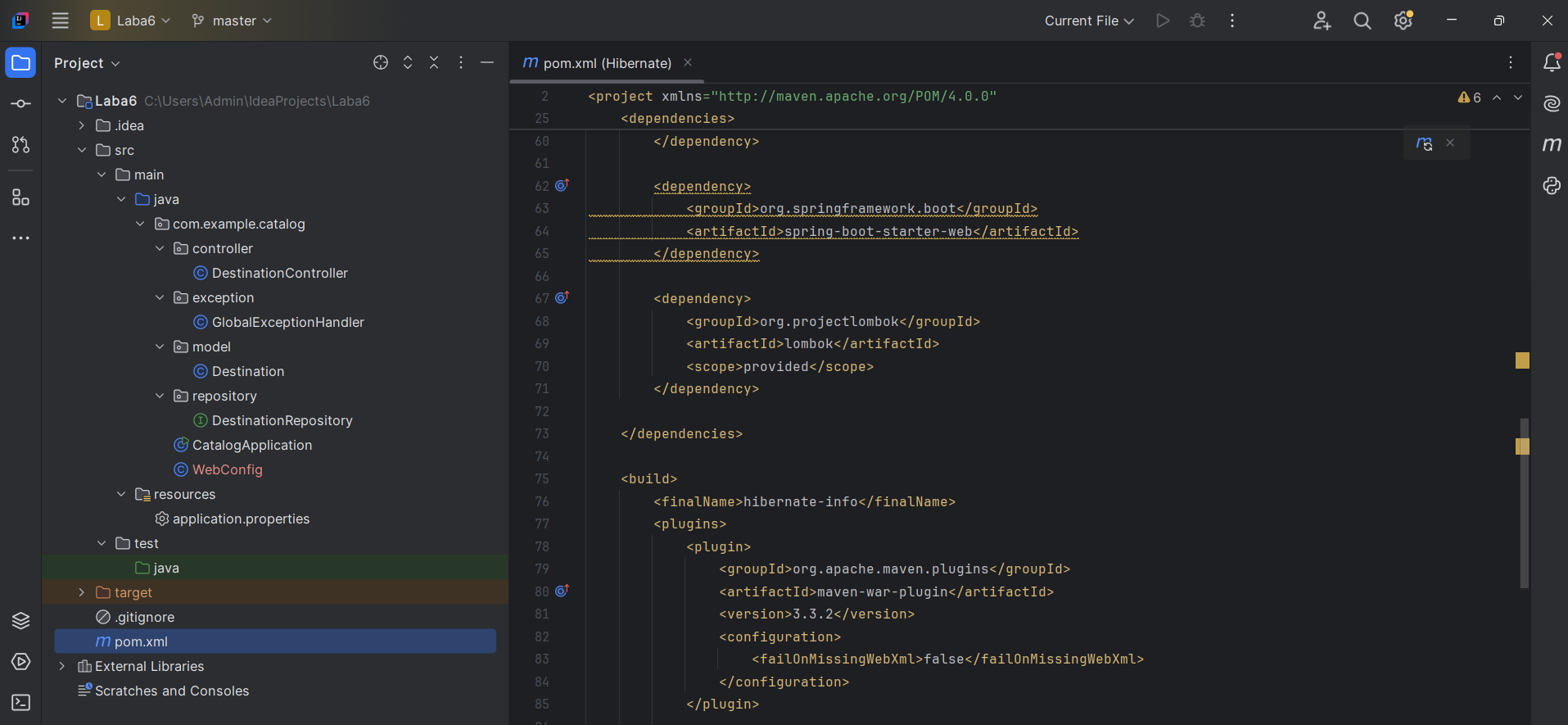
**Крок 2. Налаштування проєкту**

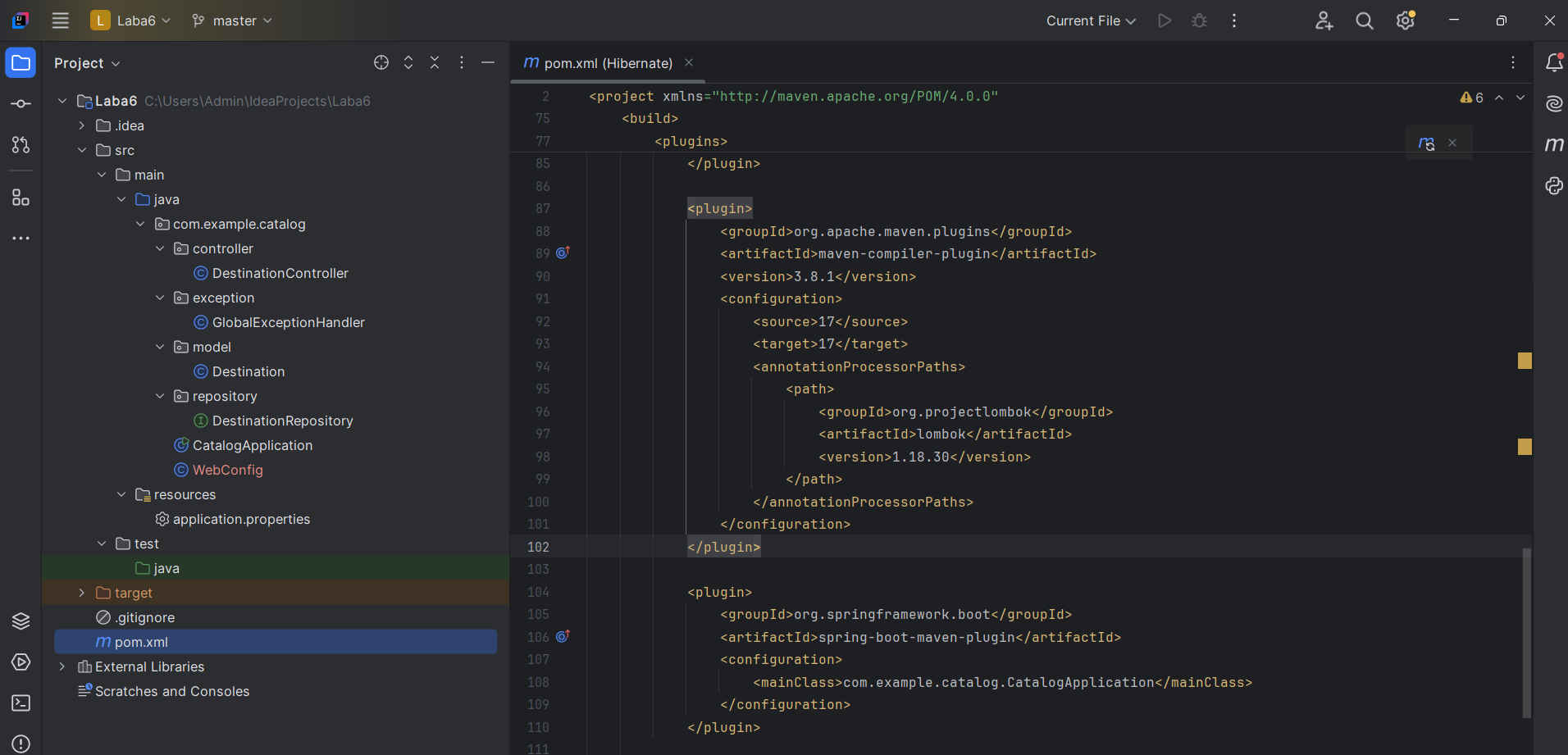
1. **Оновлення pom.xml** – додати залежності:

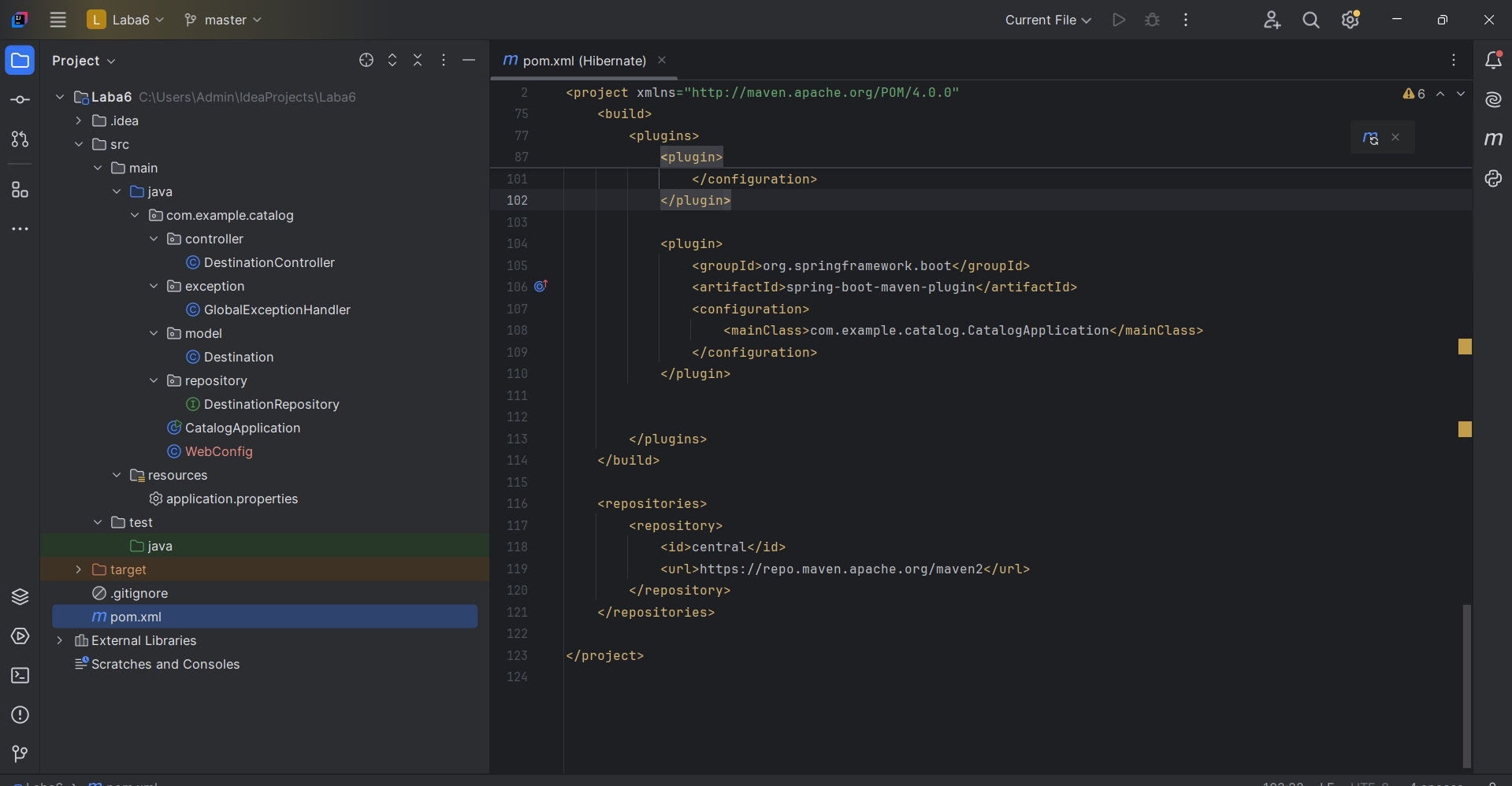
|  |
| --- |
| <dependency>     <groupId>org.springframework.boot</groupId>     <artifactId>spring-boot-starter-data-jpa</artifactId> </dependency> <dependency>     <groupId>org.springframework.boot</groupId>     <artifactId>spring-boot-starter-web</artifactId> </dependency> <dependency>     <groupId>org.postgresql</groupId>     <artifactId>postgresql</artifactId>     <scope>runtime</scope> </dependency> <dependency>     <groupId>org.projectlombok</groupId>     <artifactId>lombok</artifactId>     <scope>provided</scope> </dependency> |











1. **Конфігурація підключення до бази даних у application.properties**:

|  |
| --- |
| spring.datasource.url=jdbc:postgresql://localhost:5432/samples\_db spring.datasource.username=samples\_user spring.datasource.password=password spring.jpa.show-sql=true spring.jpa.hibernate.ddl-auto=update spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect |

**Крок 3. Створення моделі даних**

Файл Sample.java:

|  |
| --- |
| import jakarta.persistence.\*; import lombok.\*;  @Entity @Table(name = "courses") @Data @NoArgsConstructor @AllArgsConstructor public class Sample {     @Id     @GeneratedValue(strategy = GenerationType.IDENTITY)     private Long id;  …     } |

**Крок 4. Створення репозиторію**

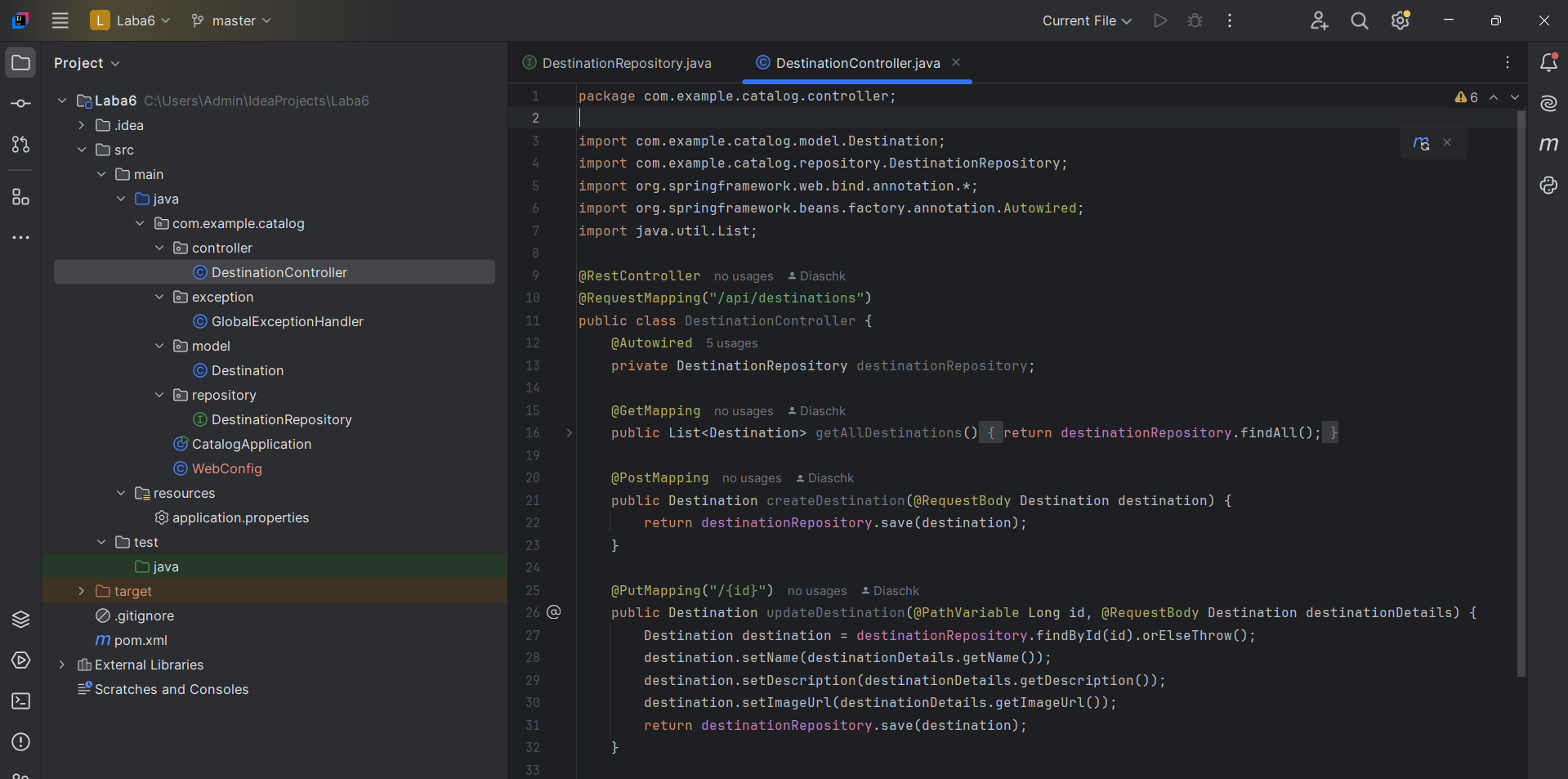
Файл SampleRepository.java:

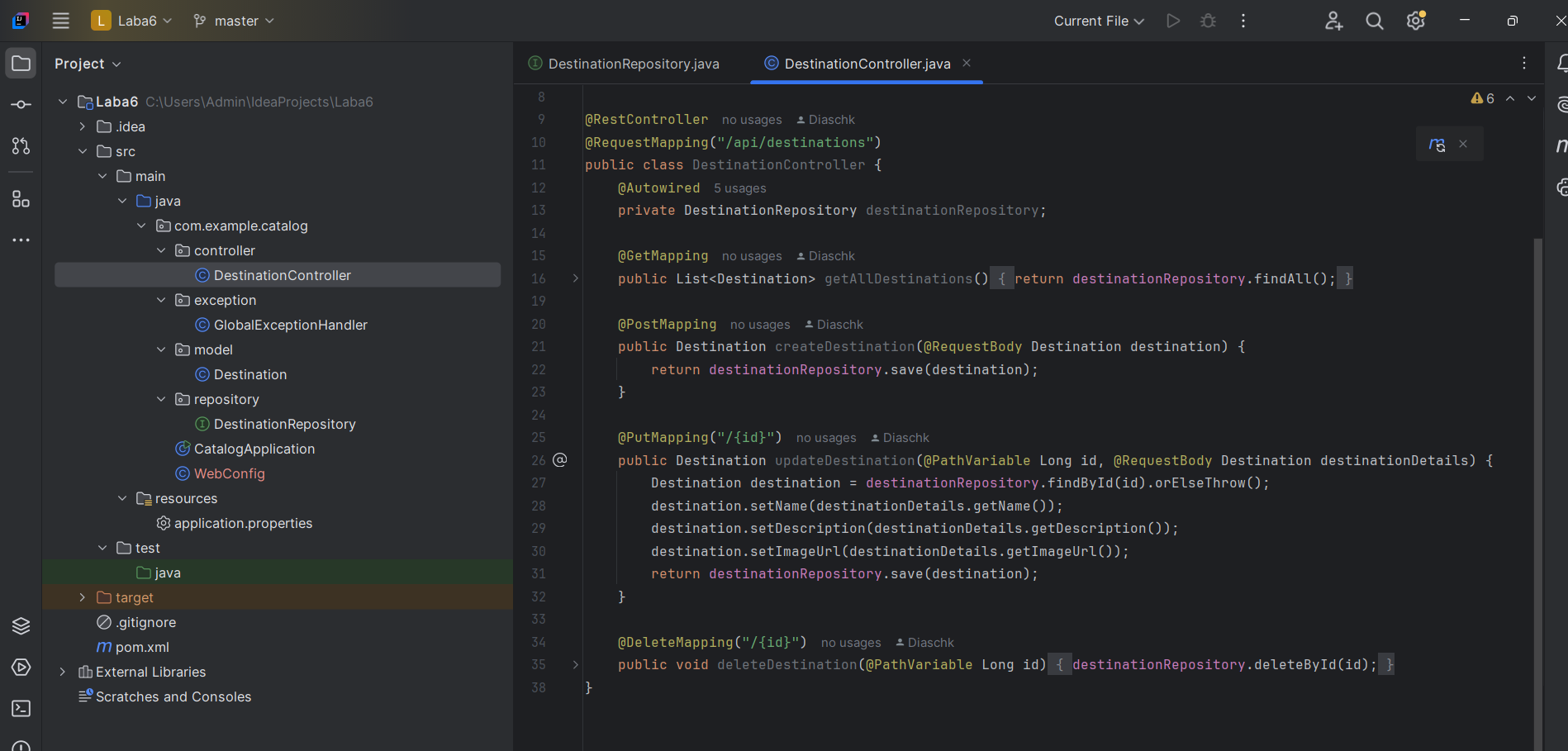
|  |
| --- |
| @Repository public interface SampleRepository extends JpaRepository<Sample, Long> {} |

**Крок 5. Реалізація REST-контролера**

Файл SampleController.java:

|  |
| --- |
| @RestController @RequestMapping("/api/samples") public class SampleController {          @Autowired     private SampleRepository sampleRepository;      @GetMapping     public List<Sample> getAllSamples() {         return sampleRepository.findAll();     }      @PostMapping     public Sample createSample(@RequestBody Sample sample) {         return sampleRepository.save(sample);     }      @PutMapping("/{id}")     public Sample updateSample(@PathVariable Long id, @RequestBody Sample sampleDetails) {         Sample sample = sampleRepository.findById(id).orElseThrow();         sample.setTitle(sampleDetails.getTitle());        …      }      @DeleteMapping("/{id}")     public void deleteSample(@PathVariable Long id) {         sampleRepository.deleteById(id);     } } |





**Крок 6. Тестування API**

Запустити Spring Boot-додаток:

|  |
| --- |
| mvn spring-boot:run |
|  |

Перевірити роботу API за допомогою Postman або cURL:

|  |
| --- |
| curl -X GET http://localhost:8080/api/samples |
|  |

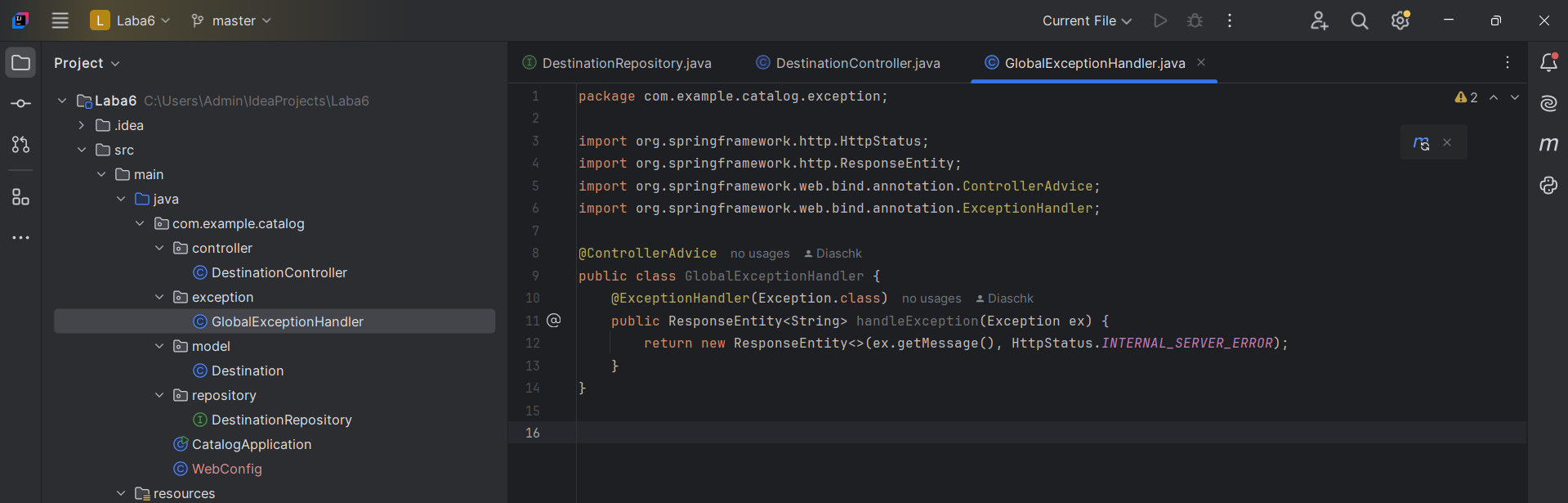
**Крок 7. Завантаження у GitHub**

|  |
| --- |
| git add . git commit -m "Lab 6: Implement REST API with @RestController" git push origin lab6 |

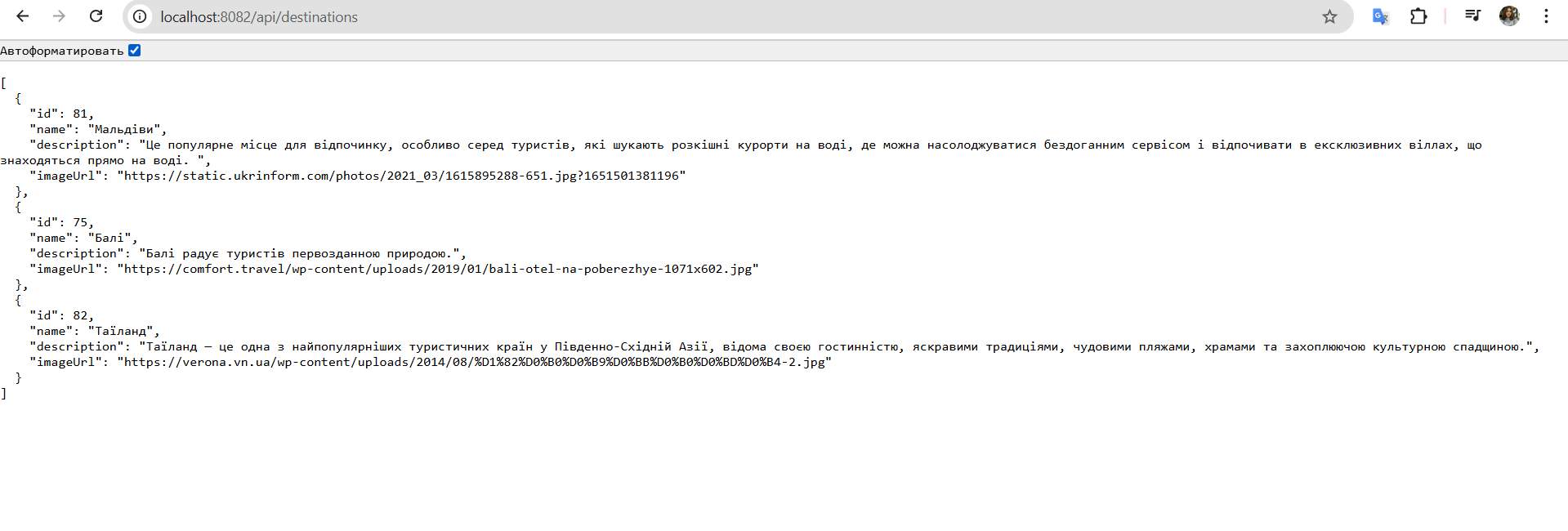
**Обробка помилок у Spring Boot**

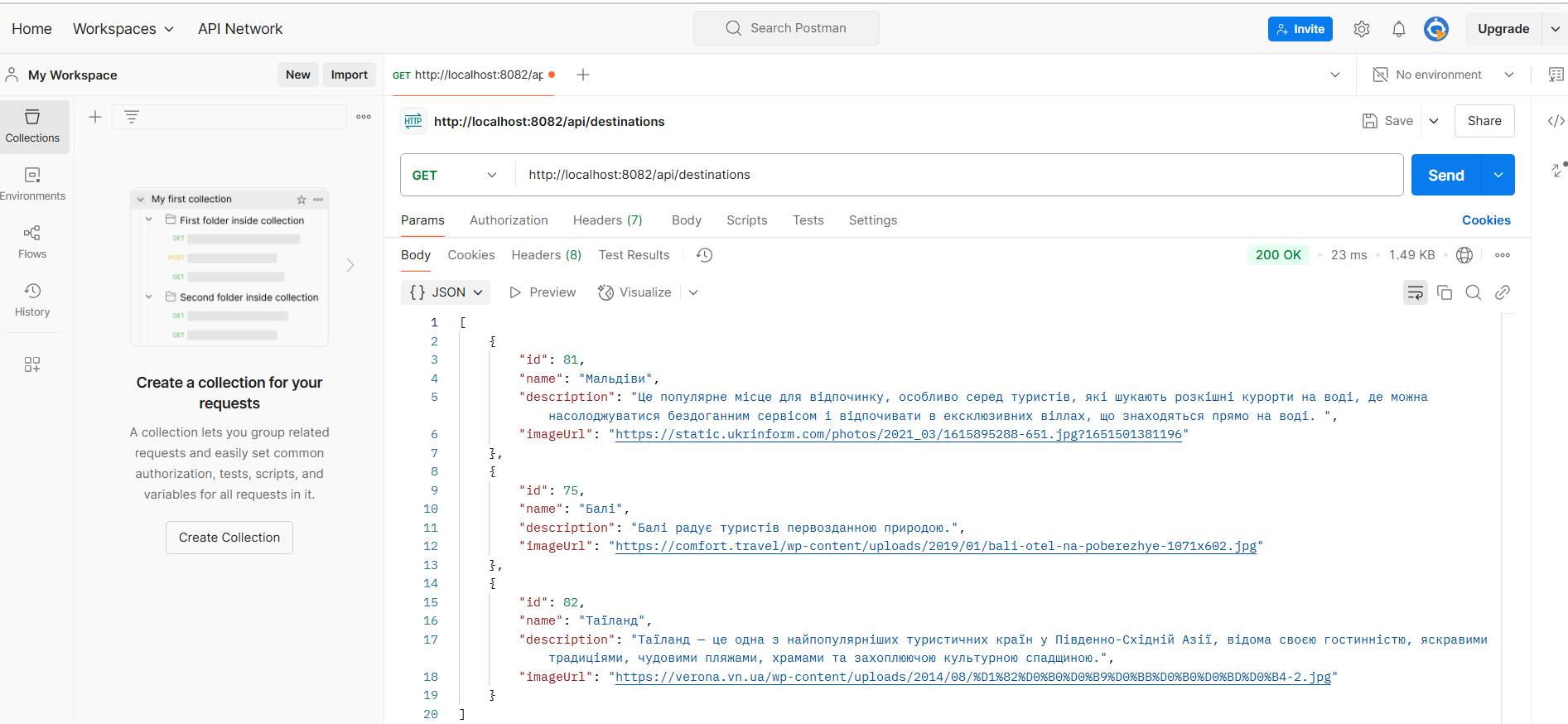
Щоб коректно обробляти помилки, слід створити GlobalExceptionHandler:

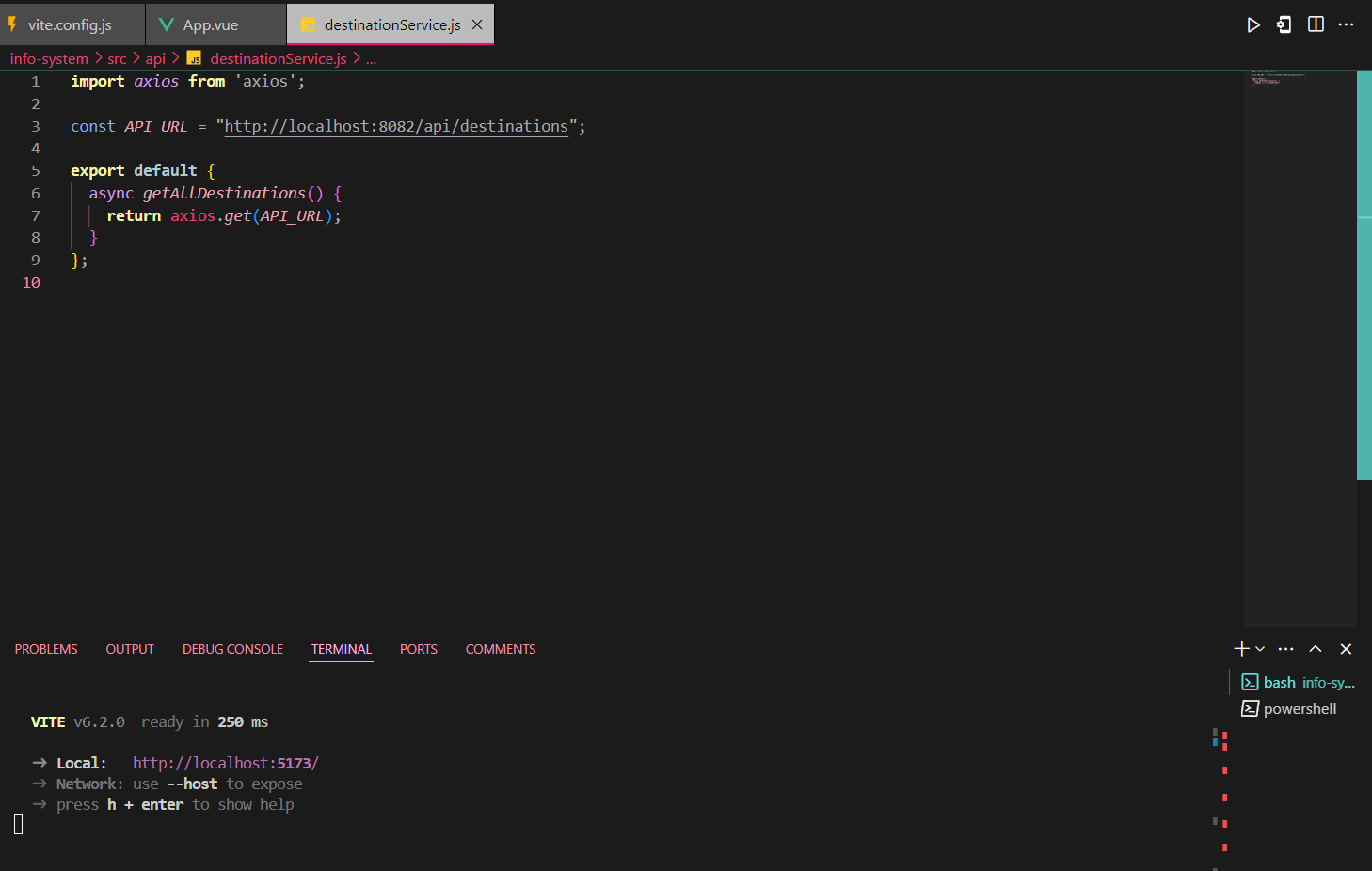
|  |
| --- |
| @ControllerAdvice public class GlobalExceptionHandler {     @ExceptionHandler(Exception.class)     public ResponseEntity<String> handleException(Exception ex) {         return new ResponseEntity<>(ex.getMessage(), HttpStatus.INTERNAL\_SERVER\_ERROR);     } } |

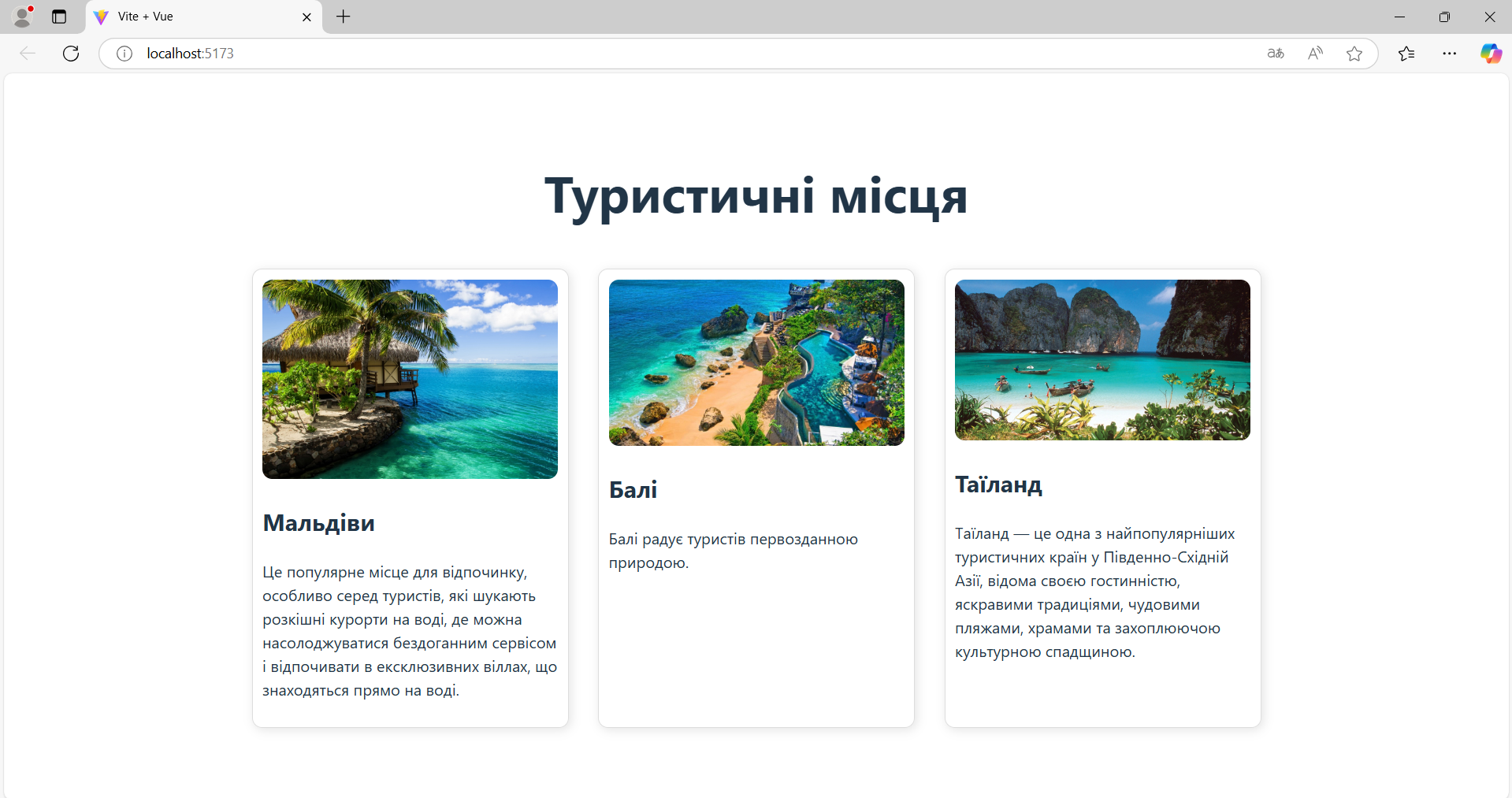


**Результат лабораторної роботи:**

****

****

****

****

**Запитання для самоперевірки:**

1. Що таке REST API, і як він працює?

**REST API (Representational State Transfer)** – це архітектурний стиль веб-служб, що використовує HTTP-протокол для взаємодії між клієнтом і сервером.

REST API працює через стандартизовані HTTP-методи (GET, POST, PUT, DELETE) і передає дані у форматах JSON або XML.

1. Які HTTP-методи використовуються для CRUD-операцій?
2. **GET** – отримання даних;
3. **POST** – створення нового ресурсу;
4. **PUT** – оновлення існуючого ресурсу;
5. **DELETE** – видалення ресурсу.
6. Як працює анотація @RestController у Spring Boot?

Анотація @RestController об'єднує в собі @Controller та @ResponseBody, що дозволяє повертати дані у форматі JSON або XML без потреби додатково вказувати @ResponseBody для кожного методу.

1. Чим відрізняється @RestController від @Controller?

@RestController використовується для створення REST API і автоматично серіалізує об'єкти у JSON або XML. @Controller зазвичай застосовується у веб-додатках для повернення HTML-сторінок через шаблонізатори (наприклад, Thymeleaf).

1. Як створити REST API за допомогою Spring Boot?

Необхідно:

1. Створити Spring Boot проєкт із залежностями spring-boot-starter-web та spring-boot-starter-data-jpa.
2. Створити Entity клас із JPA-анотаціями.
3. Реалізувати CRUD-операції у репозиторії (JpaRepository).
4. Створити REST-контролер із методами для обробки HTTP-запитів.
5. Як працює Spring Boot JPA у взаємодії з базою даних?

Spring Boot JPA (Hibernate) автоматично створює та керує підключенням до бази даних. Він забезпечує ORM, що дозволяє працювати з базою через Java-об'єкти замість SQL-запитів.

1. Що таке Lombok, і які анотації він надає?

**Lombok** – це бібліотека, що спрощує написання Java-коду, усуваючи необхідність вручну створювати гетери, сетери, конструктори тощо.

Основні анотації:

1. @Data – створює гетери, сетери, toString(), equals() та hashCode().
2. @NoArgsConstructor – створює конструктор без аргументів.
3. @AllArgsConstructor – створює конструктор із усіма полями.
4. Як протестувати REST API за допомогою Postman або cURL?
   1. У **Postman** можна створювати HTTP-запити (GET, POST, PUT, DELETE) і переглядати відповіді сервера.
   2. У **cURL** використовується командний рядок, наприклад:

curl -X GET http://localhost:8080/api/destinations

1. Як реалізувати валідацію введених даних у Spring Boot?

Використовуються анотації валідації (@NotNull, @Size, @Min, @Max тощо) у поєднанні з @Valid у контролерах.

1. Як налаштувати логування запитів у Spring Boot?

У файлі application.properties можна активувати логування SQL-запитів:

logging.level.org.springframework.web=DEBUG

logging.level.org.hibernate.SQL=DEBUG

**Висновок**

У цій лабораторній роботі я створила REST-сервіс на Spring Boot, використовуючи анотацію @RestController. Реалізувала CRUD-операції для взаємодії з базою даних PostgreSQL за допомогою Spring Boot JPA. Для спрощення коду використовувала Lombok, що дозволило уникнути зайвого написання гетерів, сетерів та конструкторів.

Під час роботи я також налаштувала підключення до бази даних у application.properties, створила модель даних, репозиторій і контролер.

Для тестування API використовувала Postman та cURL, що допомогло перевірити коректність роботи сервісу.

У результаті отримала повноцінний REST-сервіс, який можна інтегрувати з фронтендом.