**ЗВІТ**

**до лабораторної роботи № 4**  
на тему:  
**"** **Пілотний зразок індивідуального проекту."**

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

**Мета роботи:** Ознайомитися з технологіями JDBC, JPA для роботи з базою даних. Розробити першу (демо) версію проекту. Створити мінімально функціональний інтерфейс у Vue.js. Реалізувати взаємодію front-end і back-end через REST API.

**Хід роботи:**

**Тематика веб-сайту:** Каталог туристичних місць.

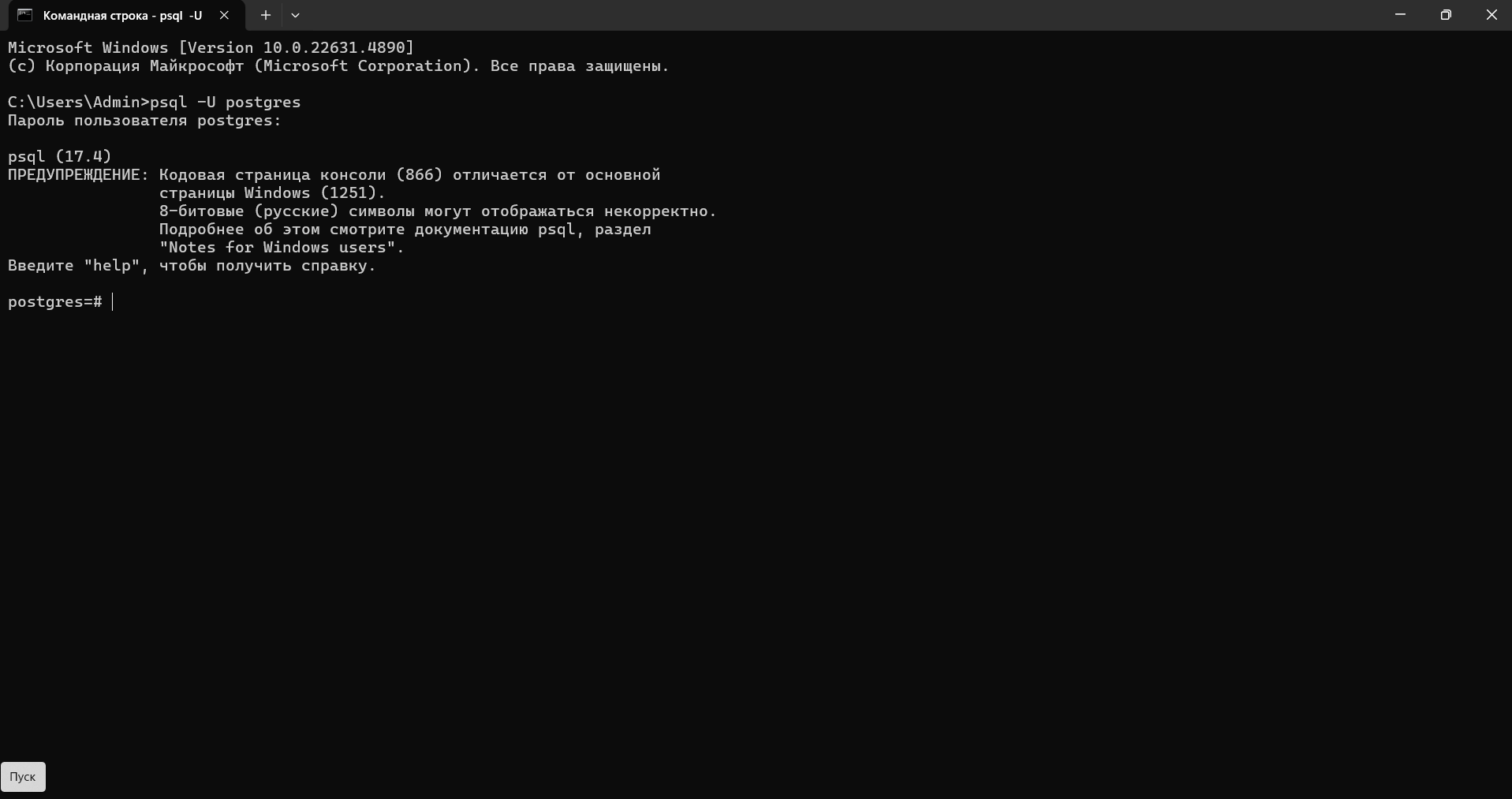
**1. Встановлення та налаштування PostgreSQL:**

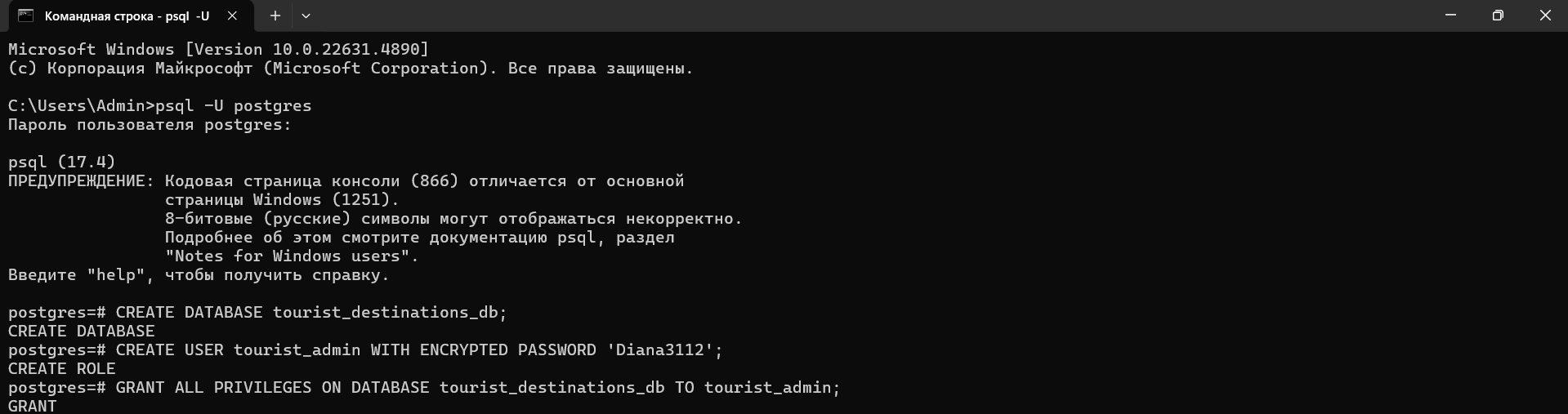
1. Завантажити та встановити PostgreSQL з офіційного сайту<https://www.postgresql.org/>.
2. Під час встановлення створити користувача та базу даних для проекту:

CREATE DATABASE project\_db;

CREATE USER project\_user WITH ENCRYPTED PASSWORD 'password';

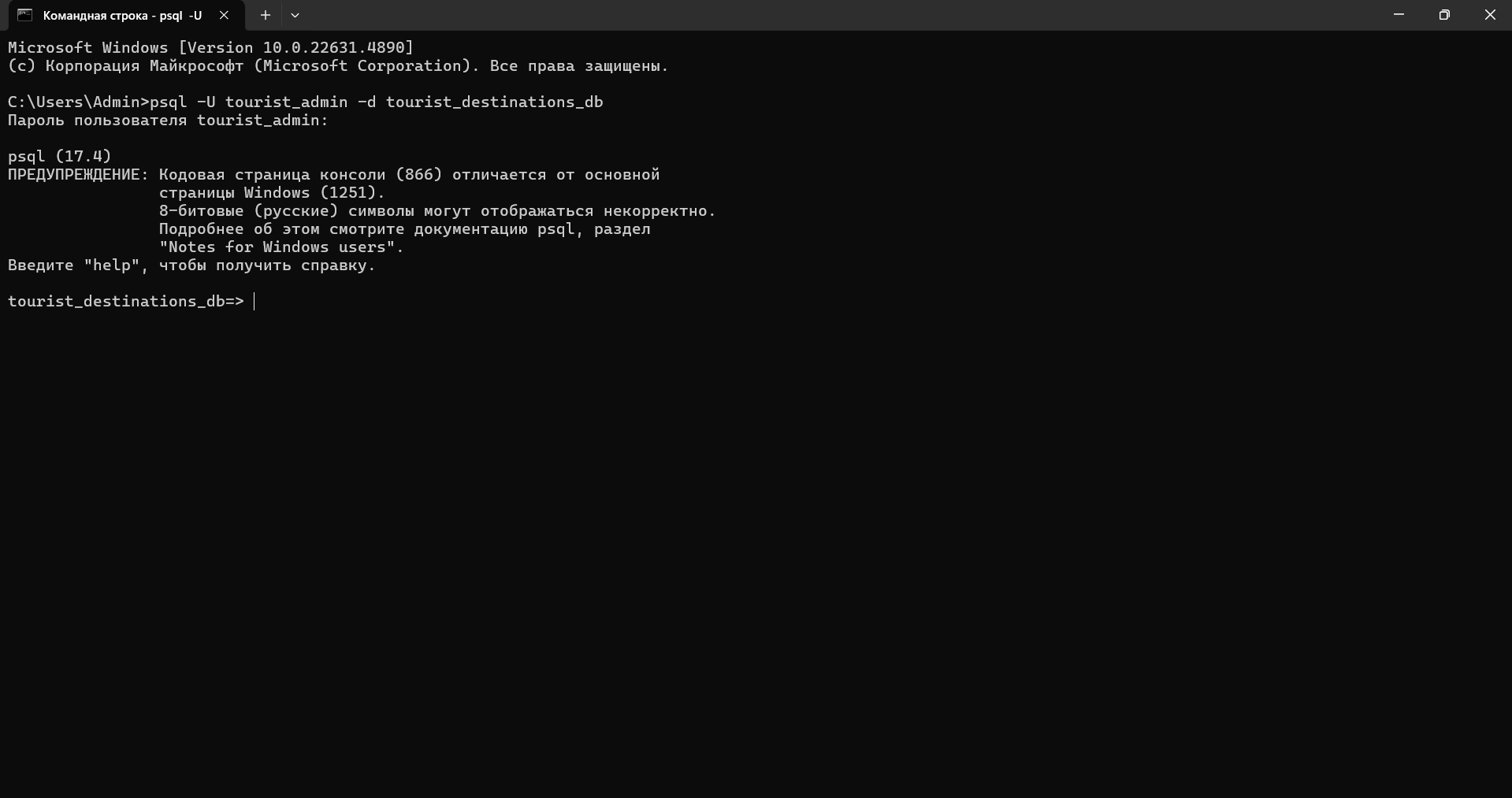
GRANT ALL PRIVILEGES ON DATABASE project\_db TO project\_user;





1. Перевірити роботу серверу за допомогою pgAdmin або командного рядка

psql -U project\_user -d project\_db



1. Додати таблиці відповідно до обраної тематики:

CREATE TABLE items (

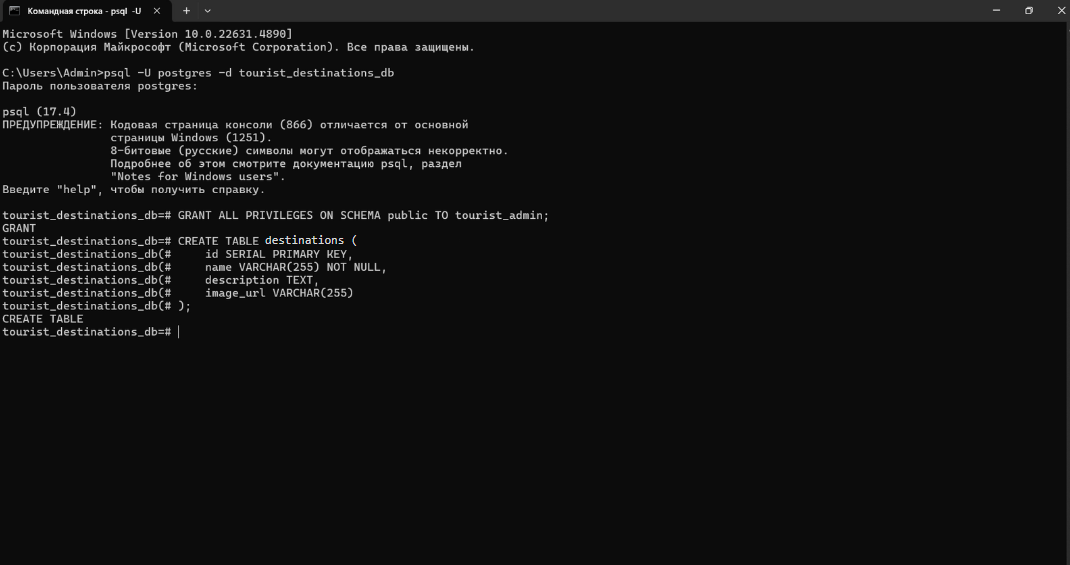
    id SERIAL PRIMARY KEY,

    name VARCHAR(255) NOT NULL,

    description TEXT,

    image\_url VARCHAR(255)

);



2. **Налаштування Hibernate у проєкті**

1. Додати залежності Hibernate та PostgreSQL у pom.xml.

<dependencies>

    <dependency>

        <groupId>org.postgresql</groupId>

        <artifactId>postgresql</artifactId>

        <scope>runtime</scope>

    </dependency>

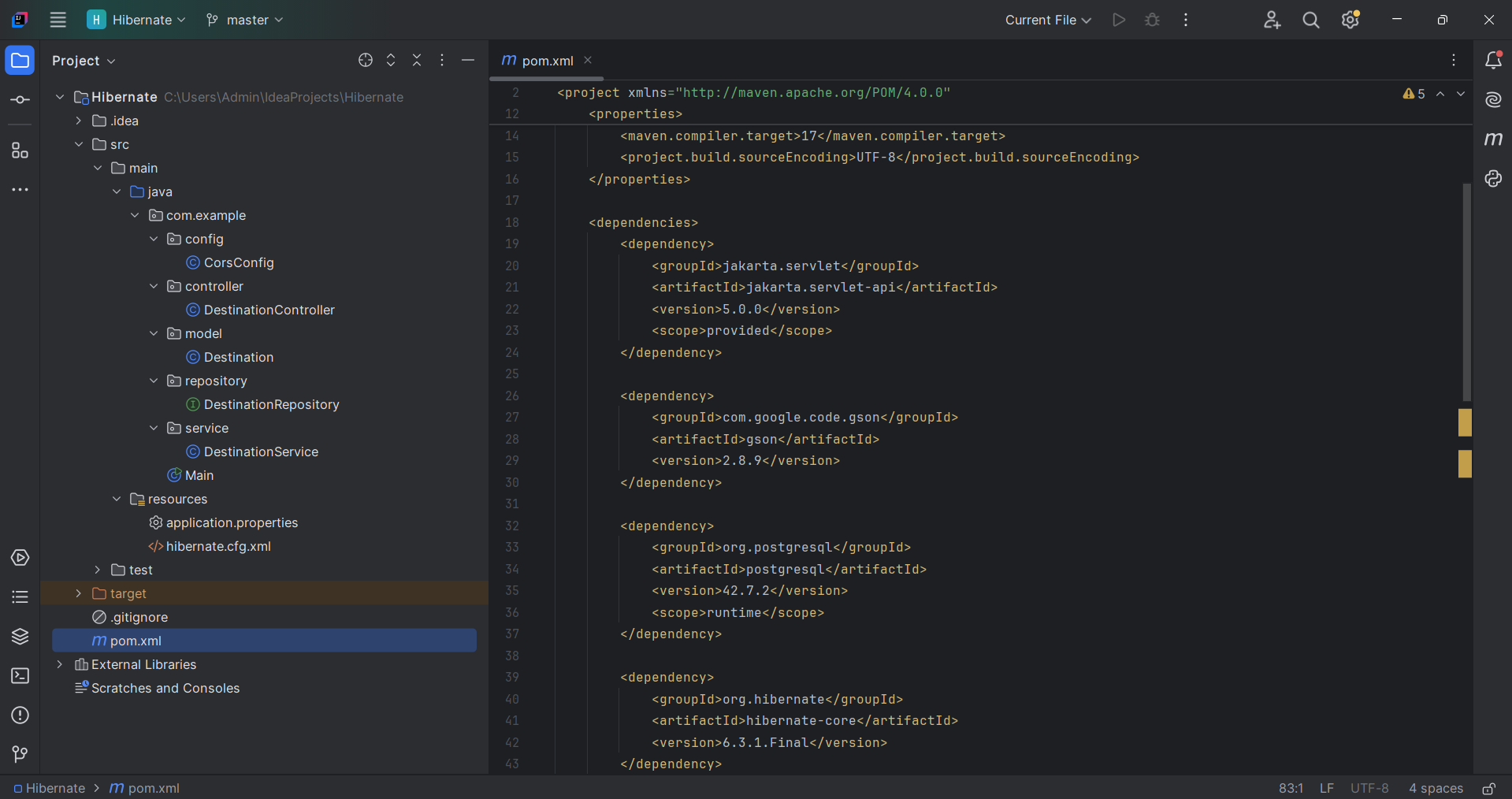
    <dependency>

        <groupId>org.hibernate</groupId>

        <artifactId>hibernate-core</artifactId>

    </dependency>

</dependencies>



1. Налаштувати persistence.xml або application.properties для підключення до бази.

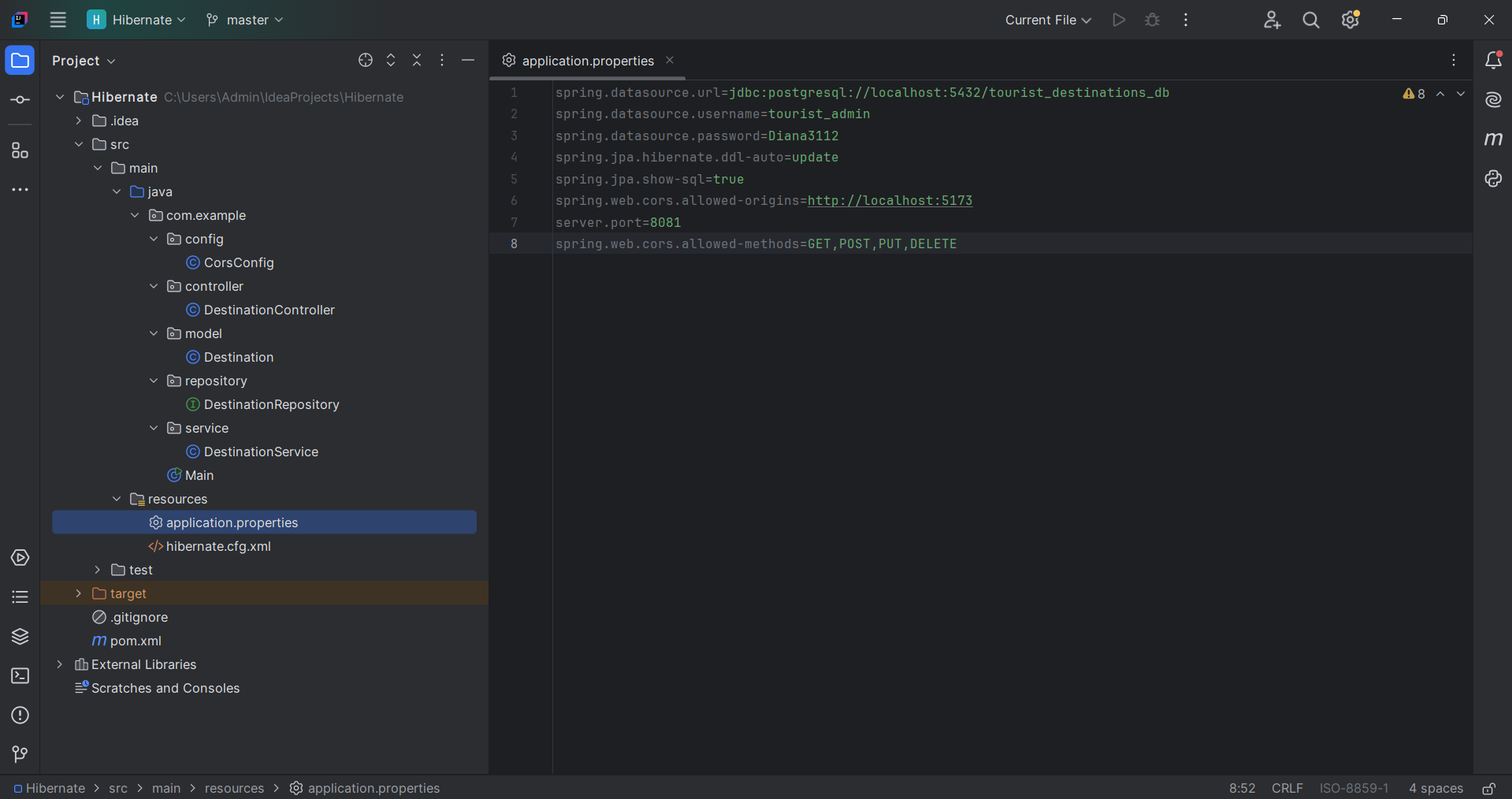
spring.datasource.url=jdbc:postgresql://localhost:5432/project\_db

spring.datasource.username=project\_user

spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true



1. Створити клас сутності (Entity) та описати структуру таблиці.

@Entity

@Table(name = "items")

public class Item {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;

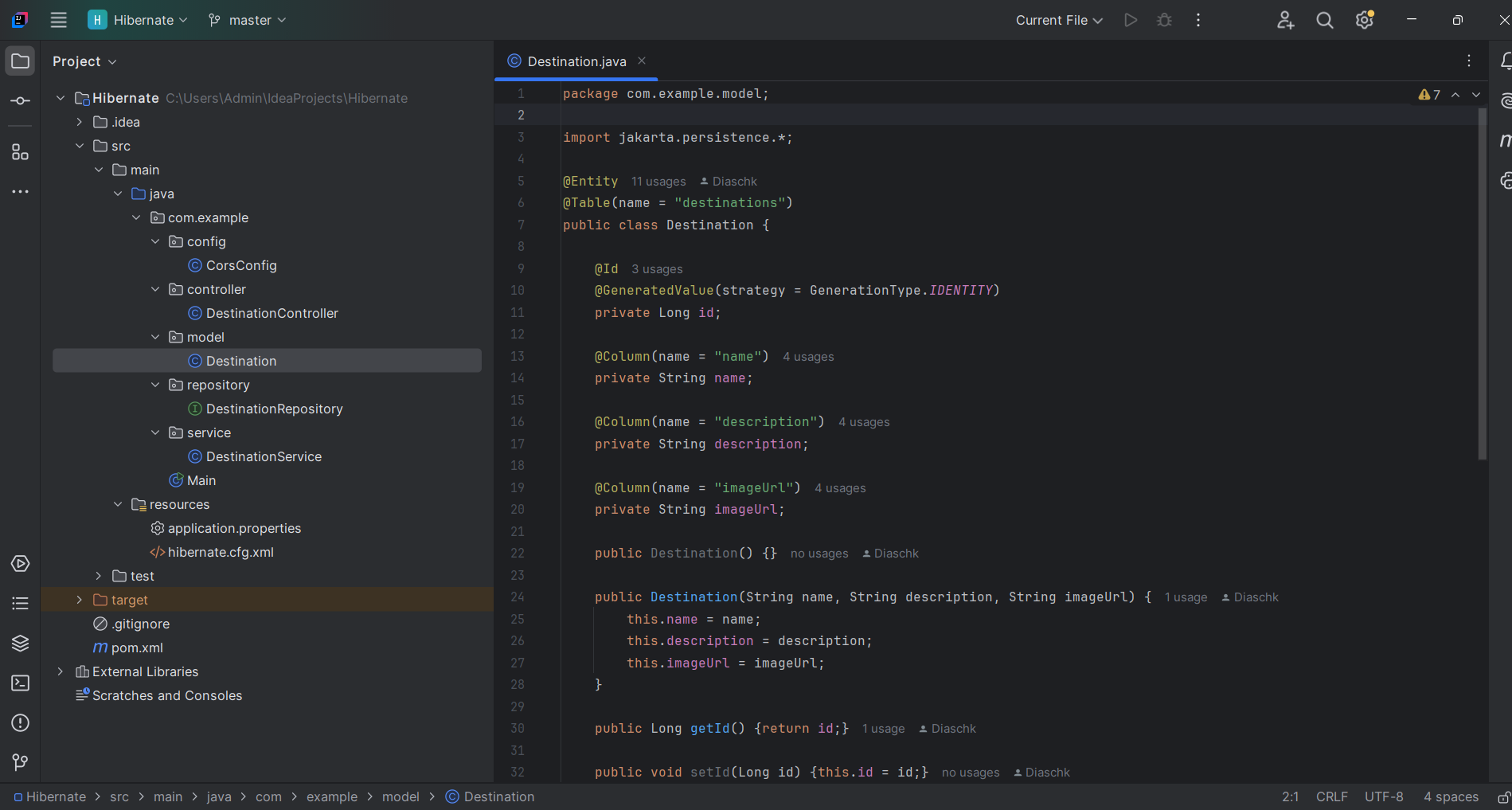
    private String name;

    private String description;

    private String imageUrl;

    // Getters and setters

}

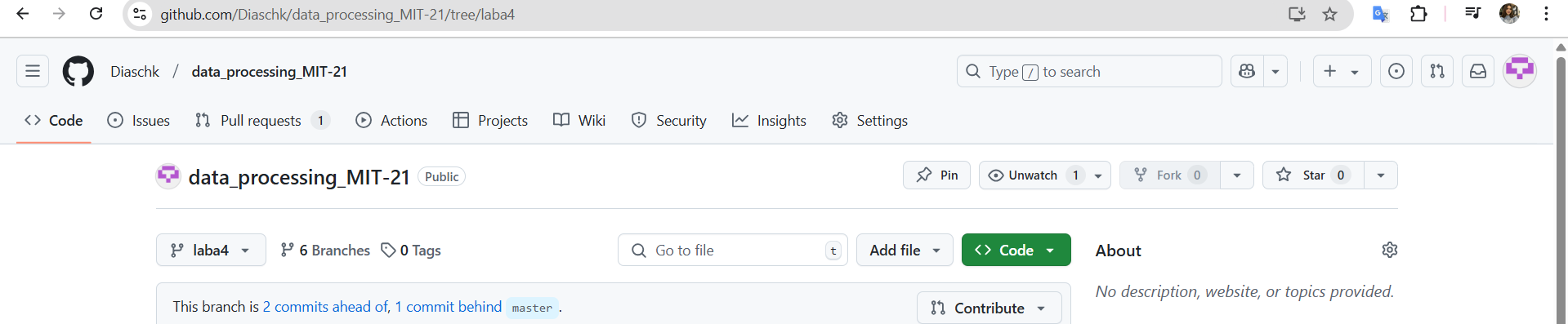




1. Виконати базові CRUD-операції для перевірки роботи.

**3. Підготовка репозиторію GitHub**

1. Створити нову гілку **lab4** в особистому репозиторії.
2. Організувати структуру проекту.



**4. Розробка пілотної версії проекту**

Front-end

1. Створити Vue.js-додаток.
2. Реалізувати головну сторінку з мінімальним інтерфейсом.
3. Відобразити список об'єктів (назва, картинка, опис).

Back-end

1. Використати **JDBC** для підключення до бази даних.

//Додати залежність у pom.xml:

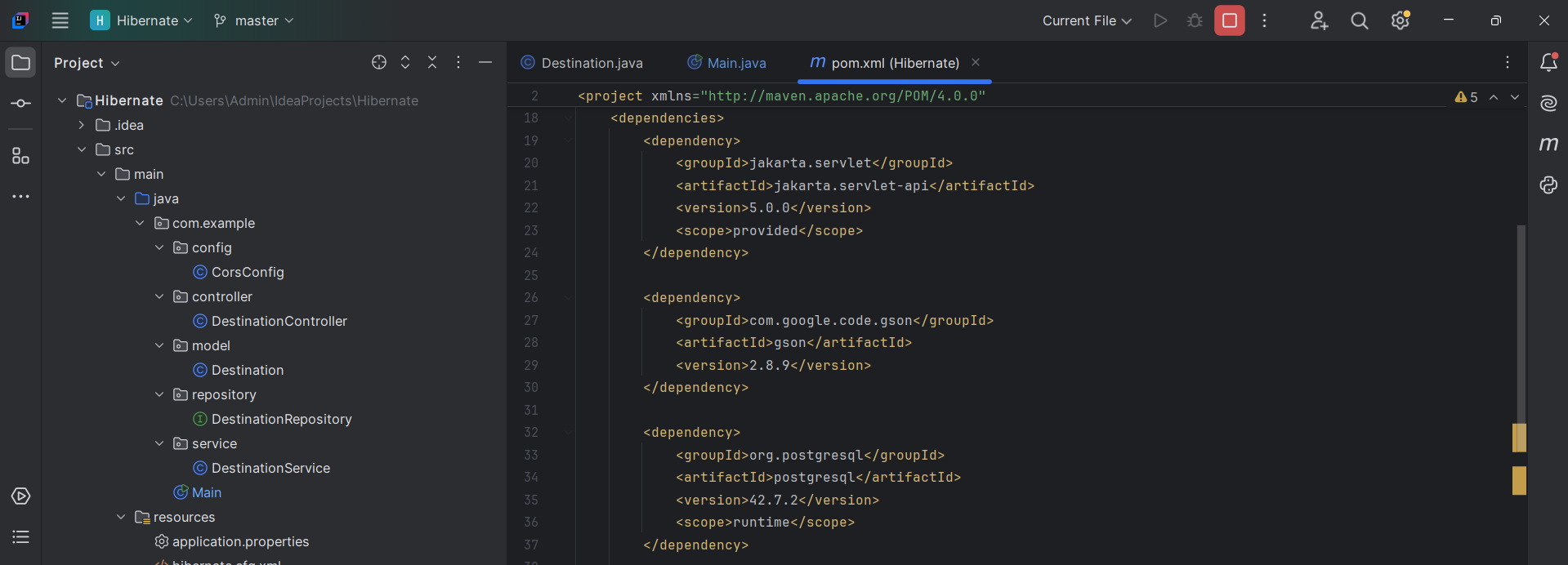
<dependency>

    <groupId>org.postgresql</groupId>

    <artifactId>postgresql</artifactId>

    <scope>runtime</scope>

</dependency>



1. Реалізувати можливість отримання даних з бази через REST API.

@RestController

@RequestMapping("/api/items")

public class ItemController {

    @Autowired

    private ItemRepository repository;

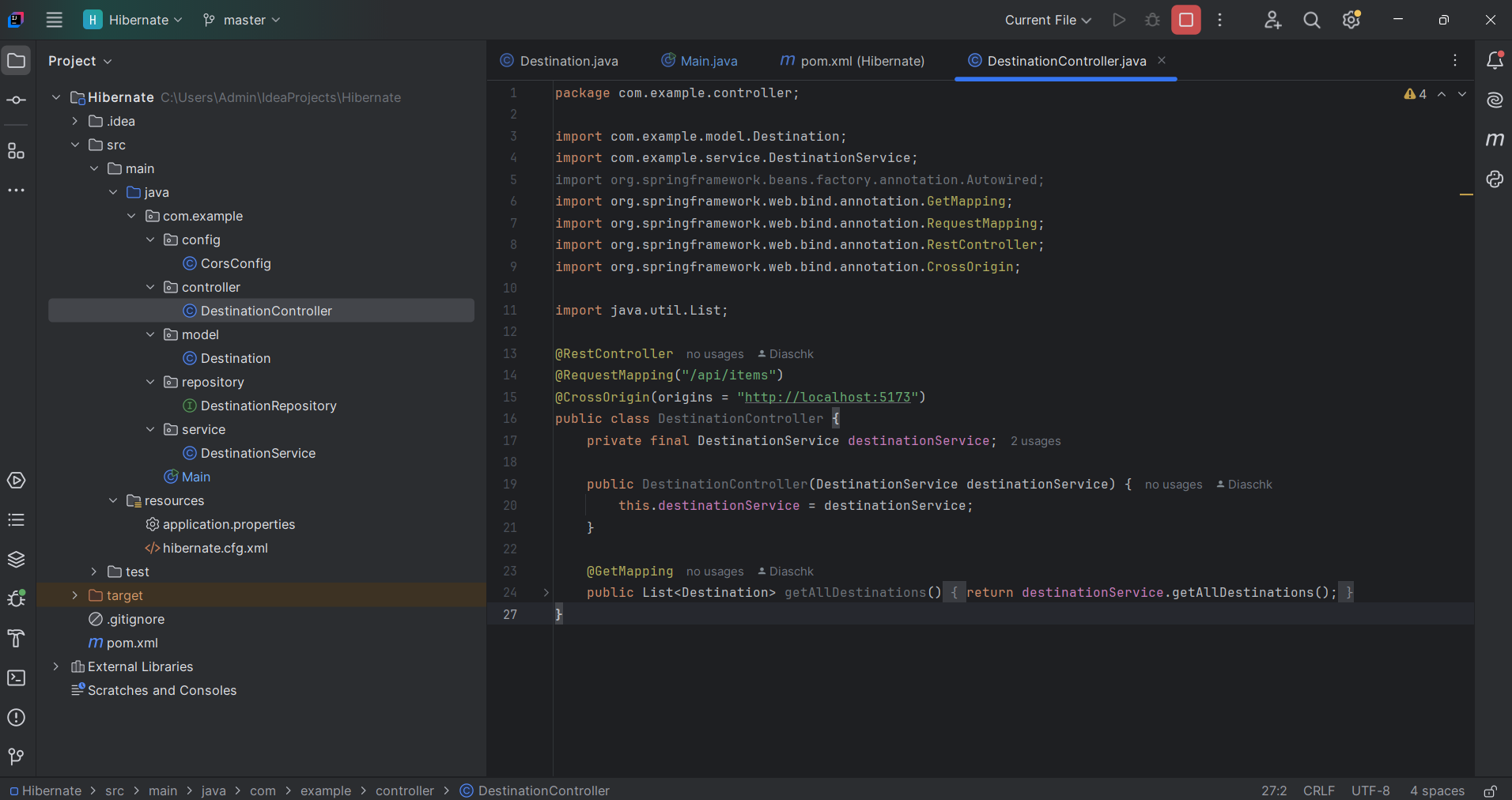
    @GetMapping

    public List<Item> getAllItems() {

        return repository.findAll();

    }

}



1. Використати **Hibernate (JPA)** для реалізації CRUD-операцій.

@Repository

public interface ItemRepository extends JpaRepository<Item, Long> {}

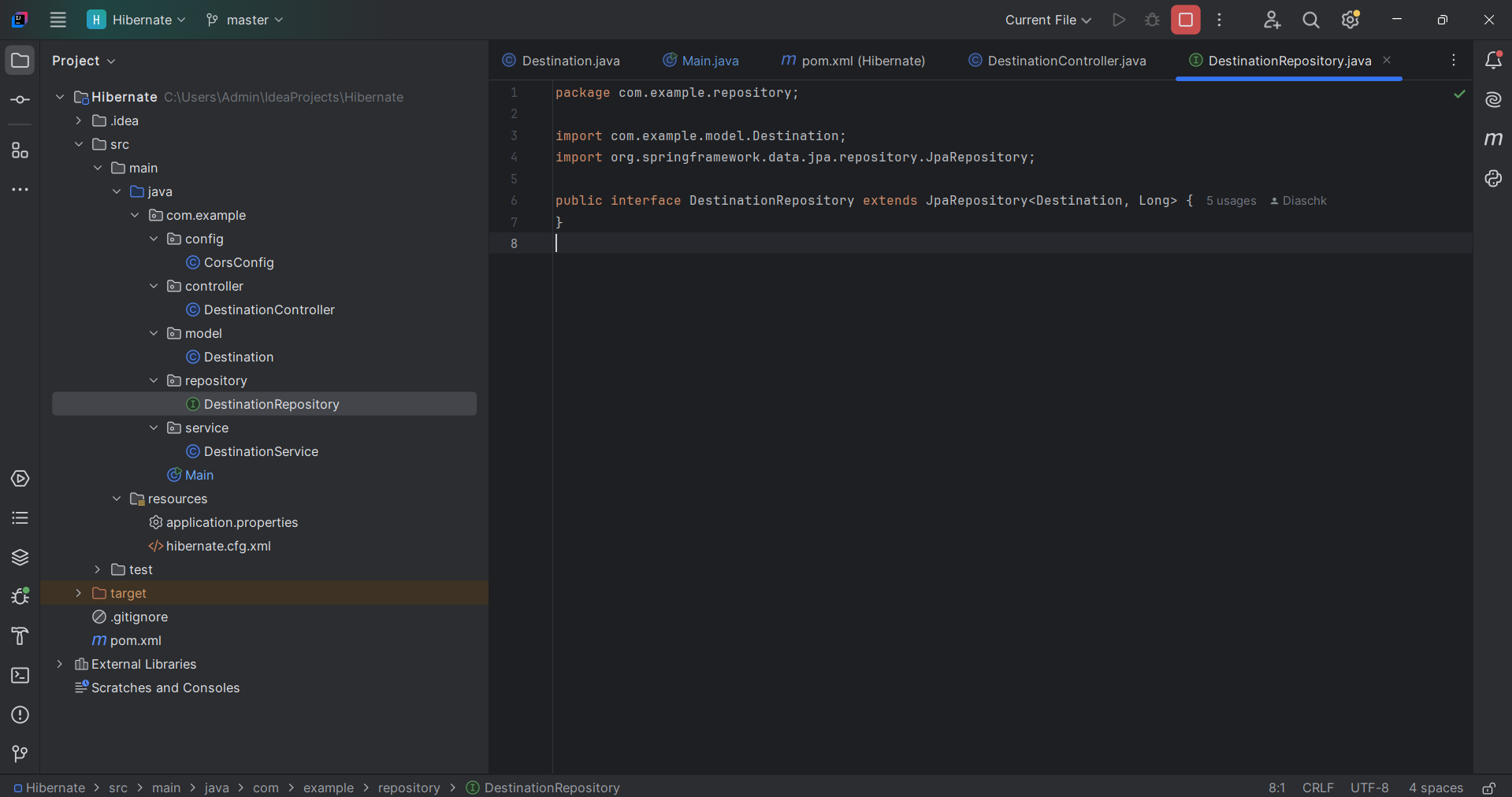
Реалізувати збереження об'єкта:

@PostMapping

public Item createItem(@RequestBody Item item) {

    return repository.save(item);

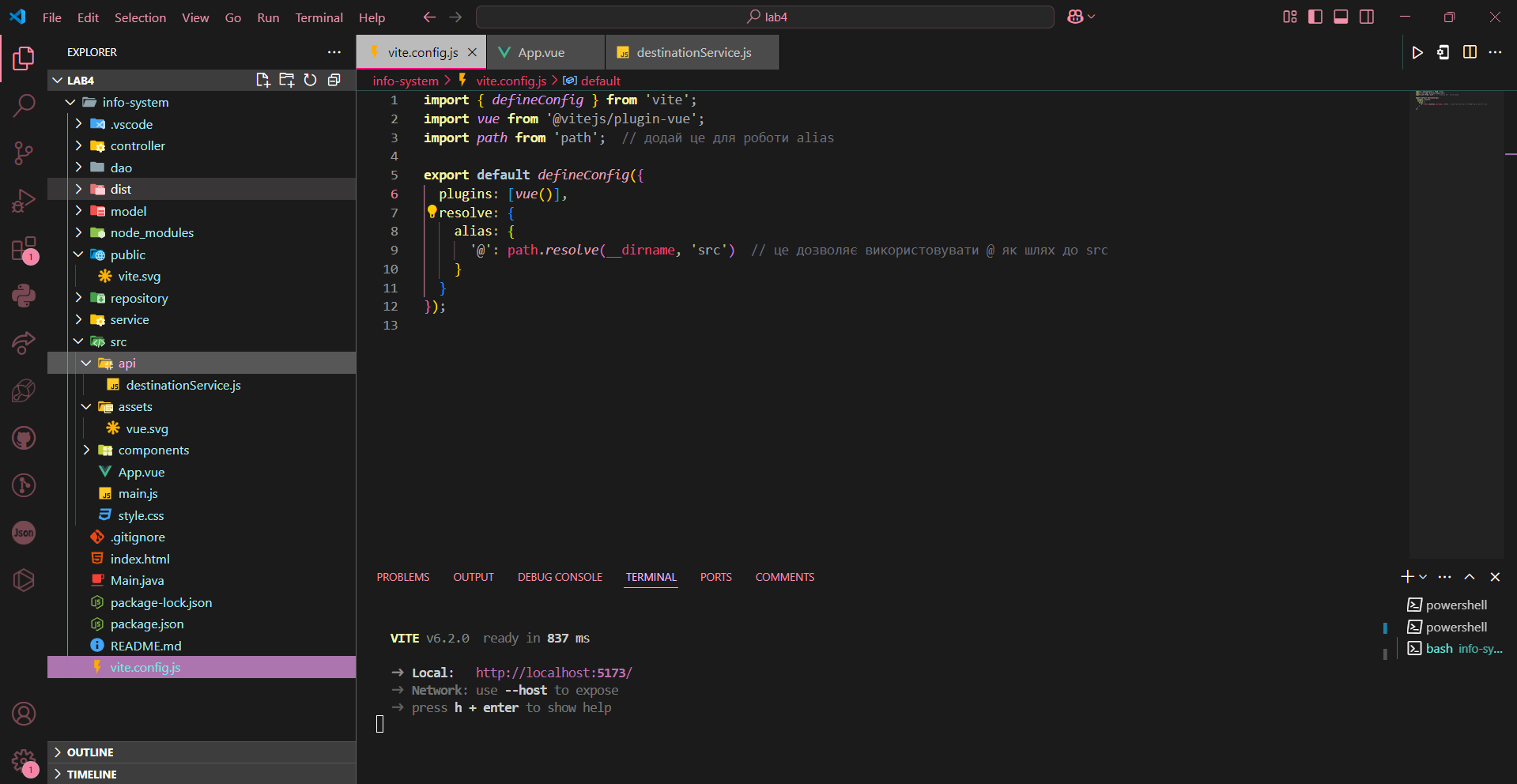
}

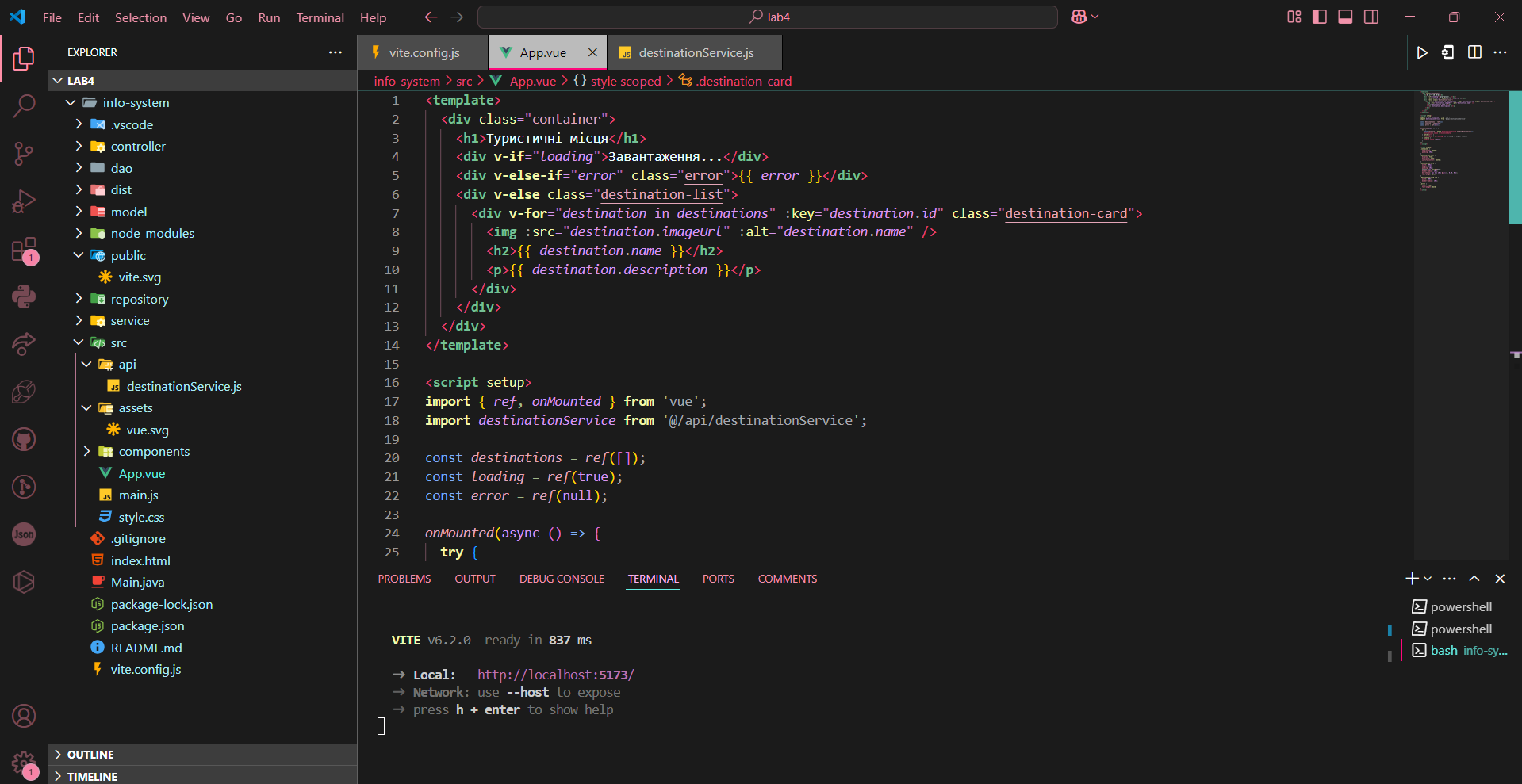


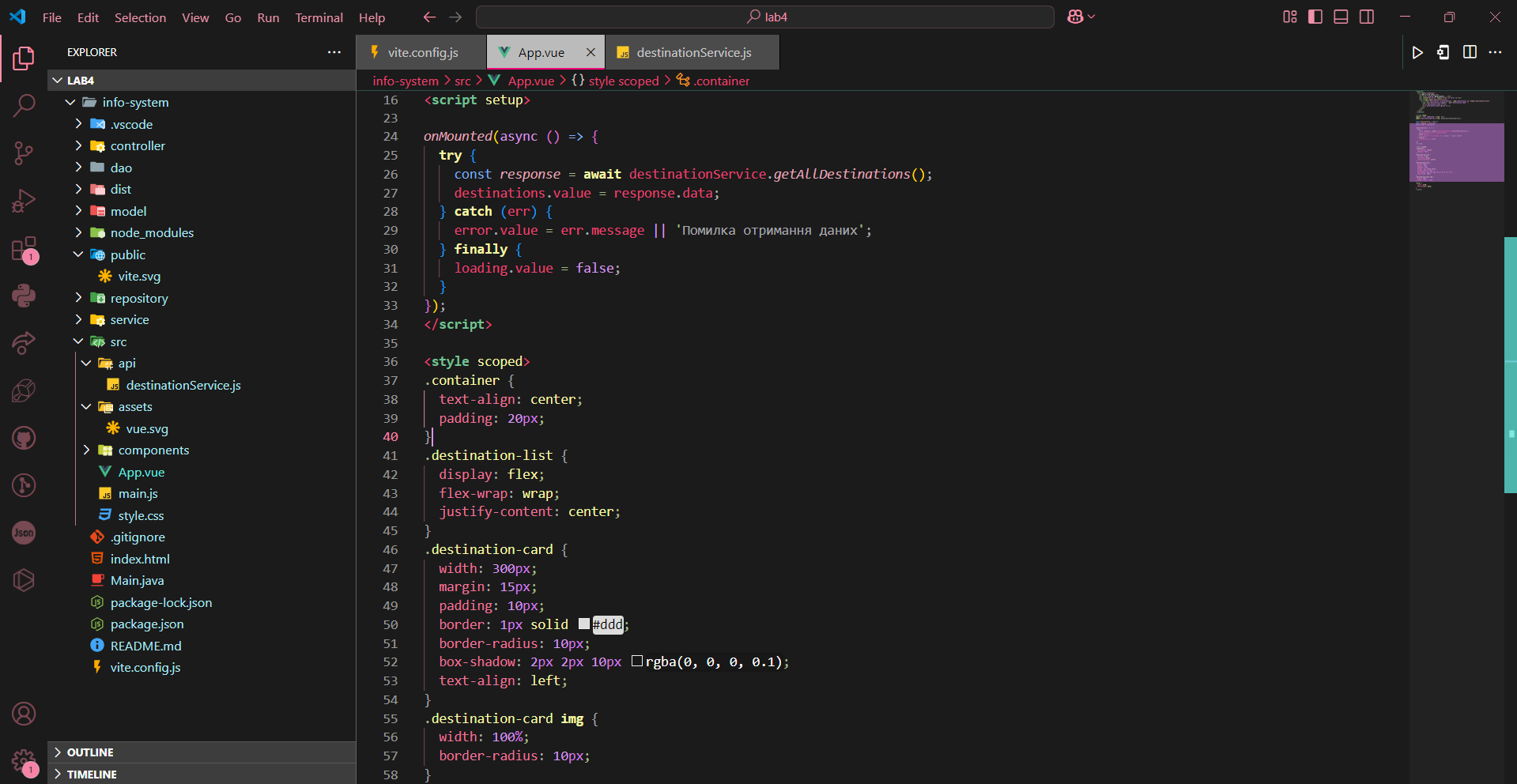
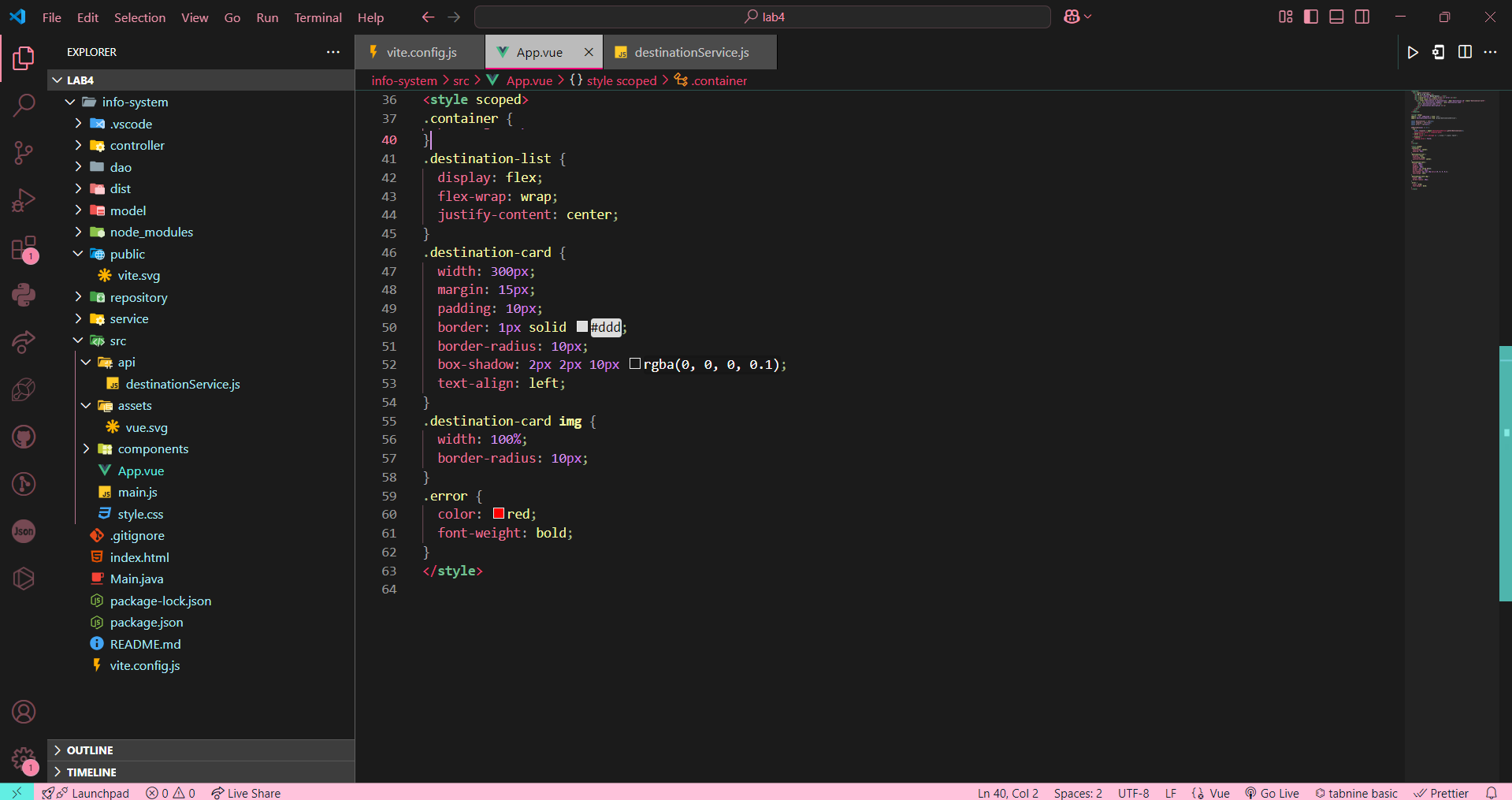
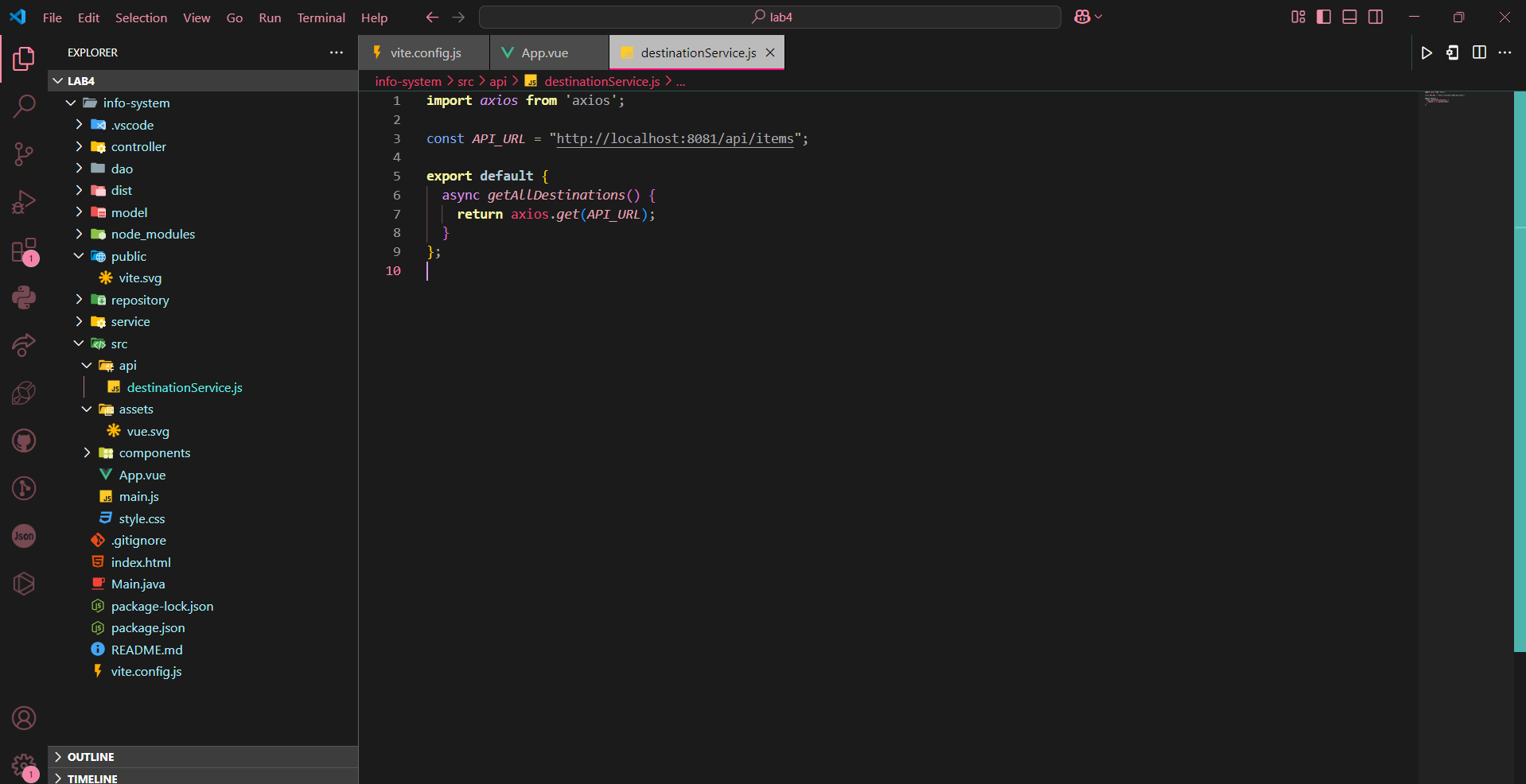
5. Інтеграція частин

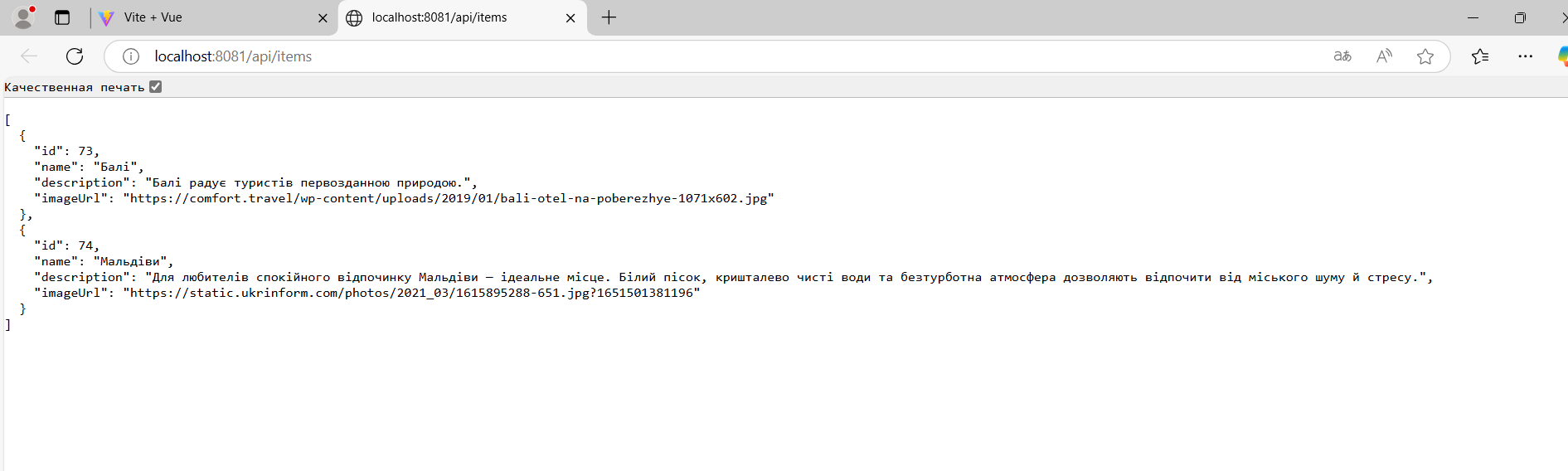
1. Забезпечити обмін даними між front-end і back-end через API.
2. Перевірити коректність відображення інформації на сайті.

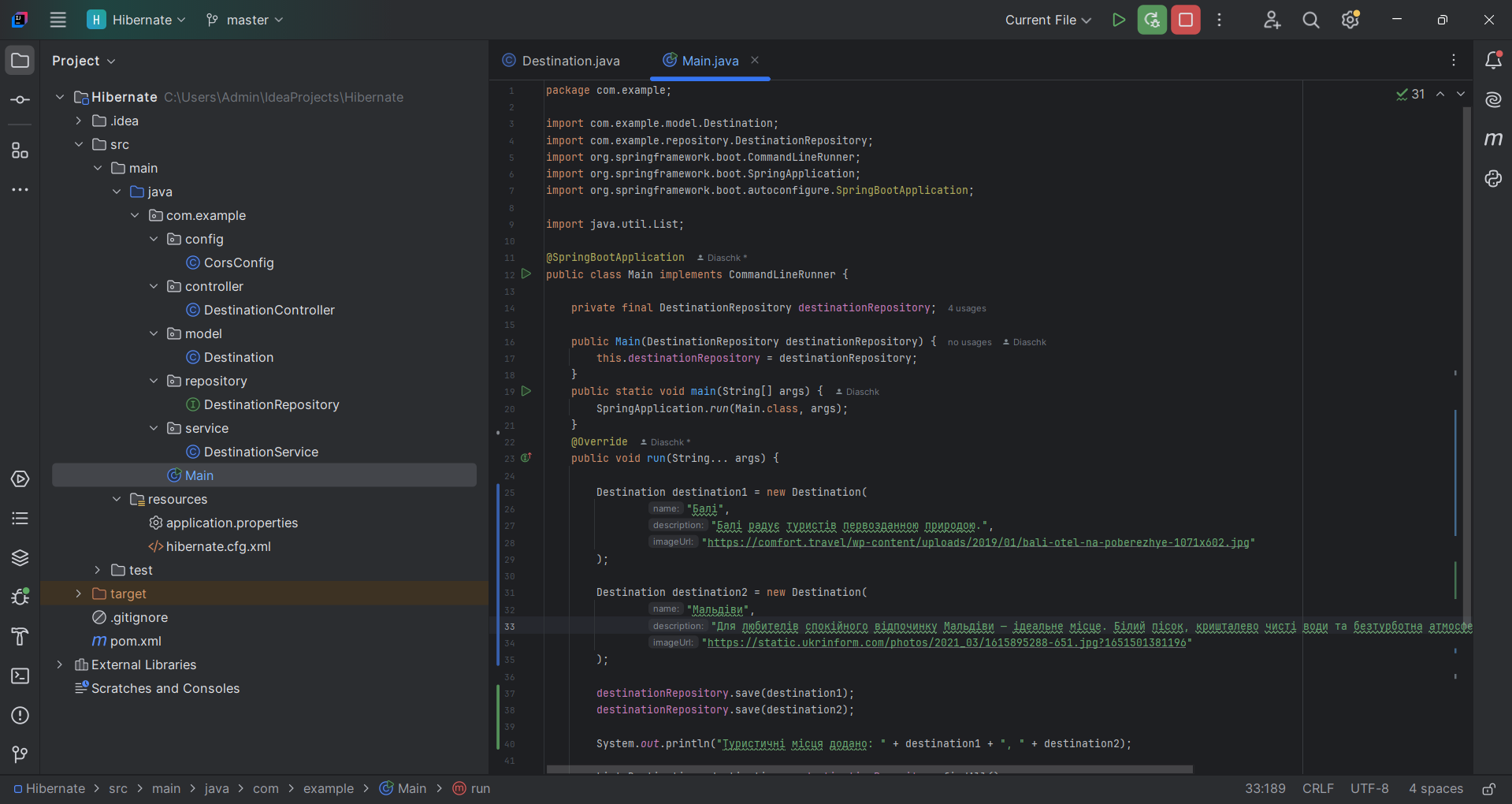
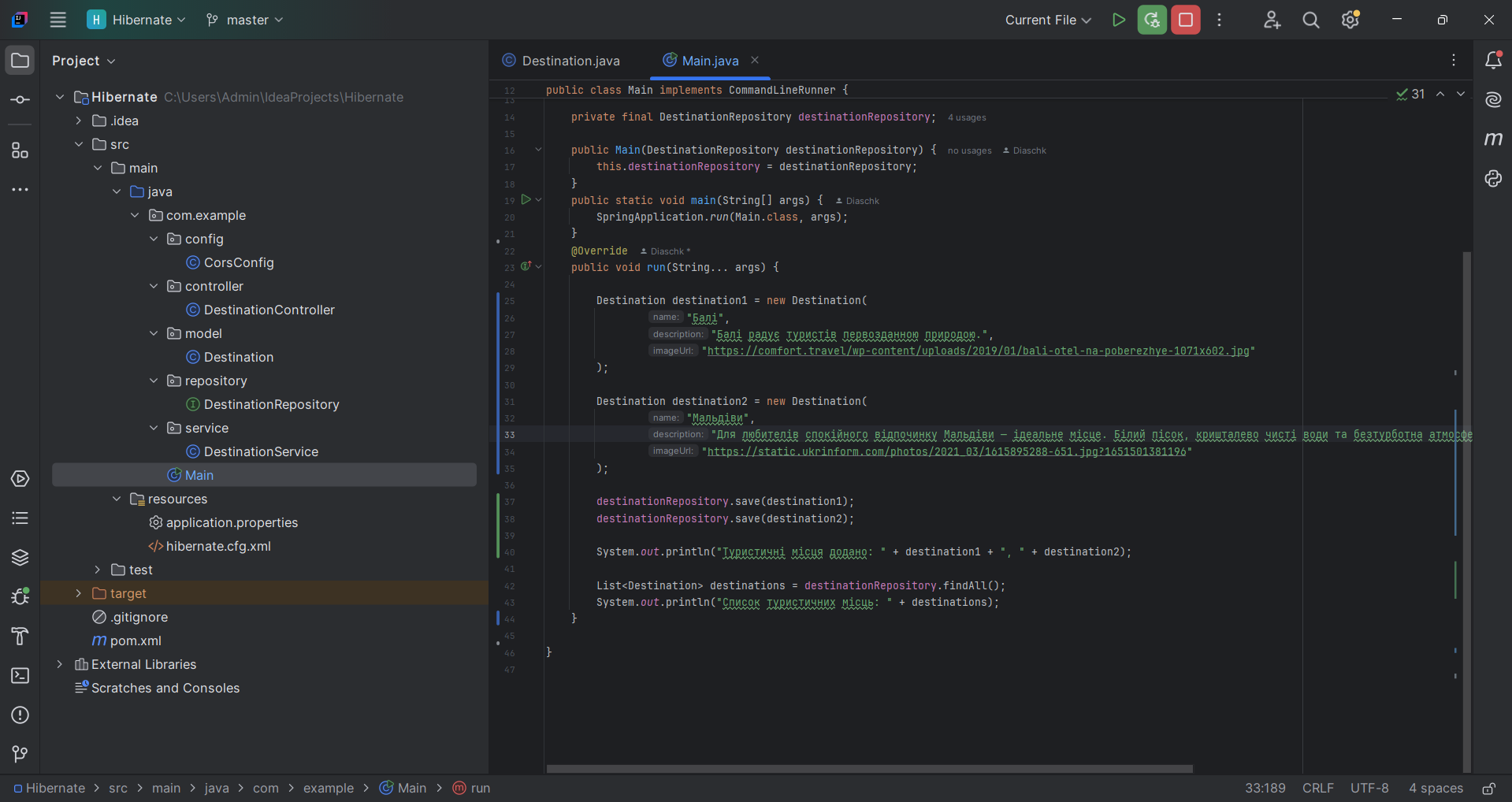
**Результат лабораторної роботи:**

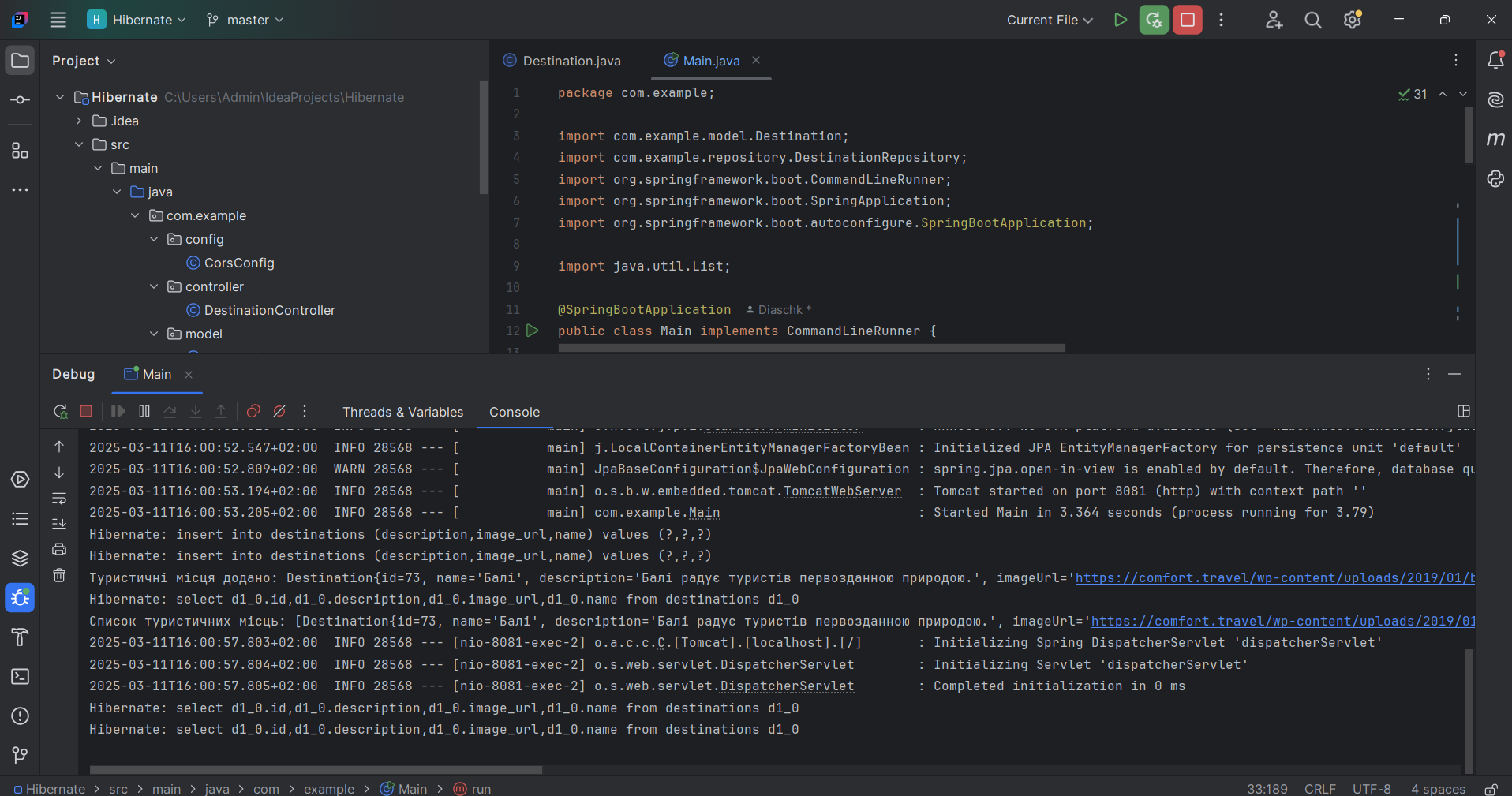


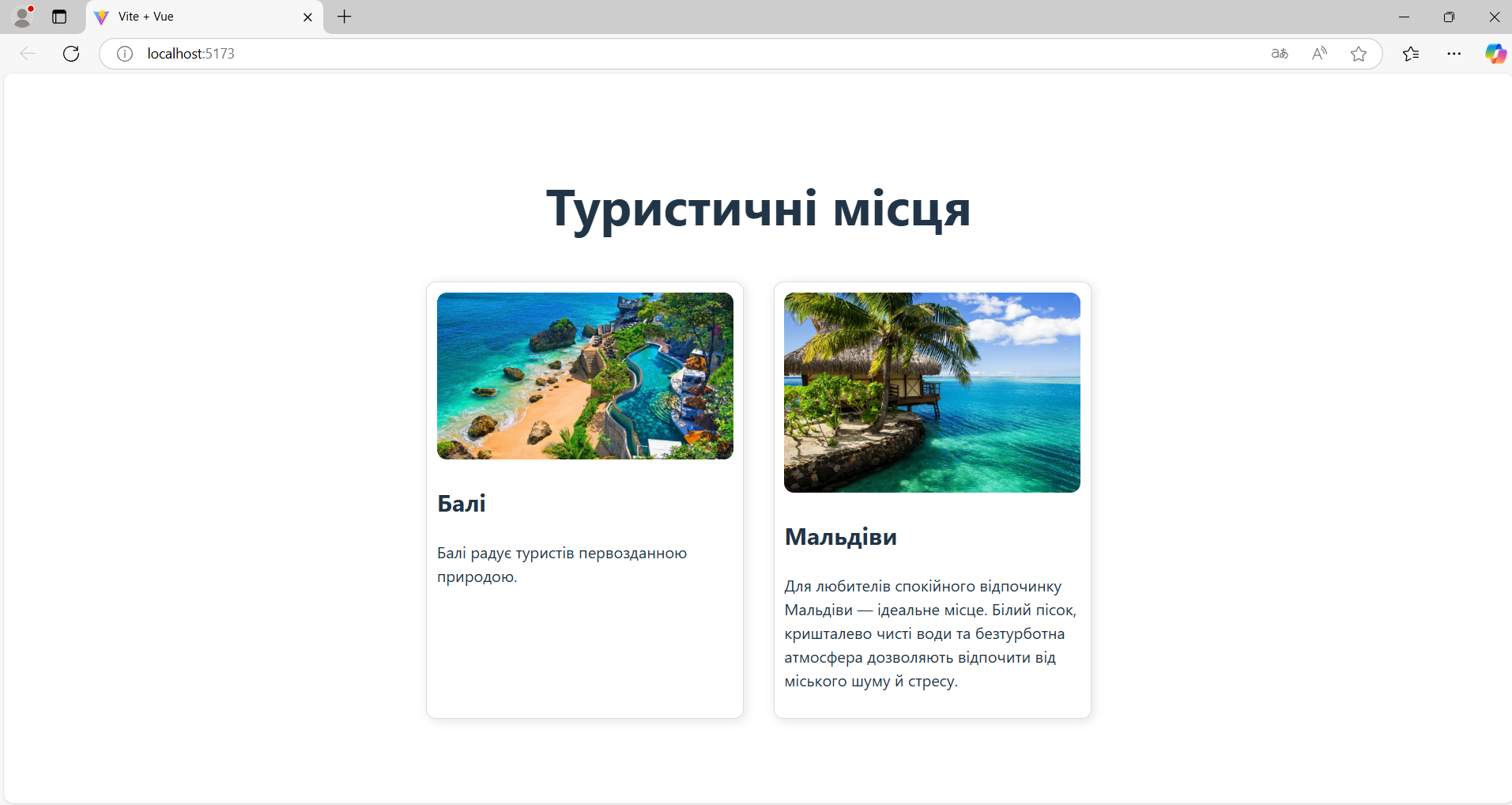






****

****

**Запитання для самоперевірки:**

1. Що таке JDBC і як він використовується у проєкті?

JDBC (Java Database Connectivity) — API для взаємодії з базою даних через SQL-запити. У проєкті використовувався опосередковано через Hibernate, який абстрагує роботу з JDBC.

1. Які основні особливості JPA та чим він відрізняється від JDBC?

JPA — ORM-технологія для роботи з об'єктами (у проекті клас Destination). JDBC вимагає ручного написання SQL. JPA спрощує CRUD, автоматизує мапування об'єктів на таблиці.

1. Які основні етапи розробки веб-застосунку у межах лабораторних робіт?
2. Налаштування PostgreSQL.
3. Створення Spring Boot додатку з JPA.
4. Реалізація REST API.
5. Розробка Vue.js інтерфейсу.
6. Інтеграція через HTTP-запити.
7. Як працює REST API у цьому проєкті?

REST API реалізовано за допомогою контролерів Spring Boot (у проекті DestinationController). Клієнт (Vue.js) відправляє HTTP-запити (GET, POST) на ендпоінти (у проекті /api/items). Бекенд обробляє запити, взаємодіє з базою даних через JPA і повертає дані у форматі JSON.

1. Які основні можливості Vue.js використано у фронтенді?
2. **Компоненти** для структурування інтерфейсу.
3. Автоматичне оновлення даних при змінах.
4. V-for для відображення списку об’єктів.
5. **Axios** для відправки HTTP-запитів до бекенду.
6. Як налаштувати підключення PostgreSQL у Java-проєкті?
   1. Додати залежність PostgreSQL у pom.xml.
   2. Вказати параметри підключення в application.properties
   3. Налаштувати JPA для автоматичного оновлення схеми БД (spring.jpa.hibernate.ddl-auto=update).
7. Які основні кроки необхідні для створення CRUD-операцій у Hibernate?
8. Створити клас-сутність з анотаціями (@Entity, @Id).
9. Реалізувати репозиторій, який розширює JpaRepository<Item, Long>.
10. Додати методи в контролері для обробки запитів (@GetMapping, @PostMapping).
11. Використовувати методи репозиторію (save(), findAll(), deleteById()).
12. Як організовано взаємодію між front-end та back-end?

Front-end (Vue.js) відправляє HTTP-запити до REST API бекенду (Spring Boot).

1. GET-запит на /api/items для отримання списку об’єктів.
2. POST-запит на /api/items для збереження нового об’єкта.  
   Дані передаються у форматі JSON. Бекенд обробляє запити і повертає відповіді, які фронтенд відображає.
3. Які основні проблеми можуть виникнути при інтеграції Vue.js та Spring Boot?
4. **CORS-помилки**: якщо бекенд не налаштований для прийняття запитів з іншого порта.
5. **Невідповідність форматів даних**: фронтенд очікує JSON, а бекенд відправляє XML.
6. **Неправильні URL**: помилки в адресах ендпоінтів.
7. **Відсутність обробки помилок**: коли бекенд повертає HTTP 500, а фронтенд не відображає повідомлення.
8. Як забезпечити коректне збереження та відображення інформації у базі даних?
9. **Валідація даних**: перевірка на бекенді перед збереженням.
10. **Транзакції**: використання @Transactional для атомарності операцій.
11. **Тестування**: перевірка CRUD-операцій за допомогою Postman або інтеграційних тестів.
12. **Логування**: включити spring.jpa.show-sql=true для відстеження SQL-запитів.
13. **Синхронізація фронтенду**: оновлення інтерфейсу після успішних запитів.

**Висновок**

У ході виконання лабораторної роботи було розроблено пілотну версію веб-застосунку, що інтегрує фронтенд на Vue.js та бекенд на Spring Boot. У проєкті було створено базу даних PostgreSQL з таблицею destinations та налаштовано доступ для користувача. На стороні бекенду використано Spring Boot у поєднанні з Hibernate (JPA), що значно спростило реалізацію CRUD-операцій. Було розроблено REST API, яке забезпечує можливість отримання та додавання даних за допомогою HTTP-запитів.

Фронтендова частина побудована на Vue.js і включає інтерфейс для відображення списку об'єктів із зазначенням назви, опису та зображення. Взаємодія між фронтендом і бекендом реалізована через HTTP-запити за допомогою Axios або Fetch API. Інтеграція двох частин застосунку відбулася через REST API, що дозволило забезпечити коректний обмін даними.

Отже, у ході роботи було продемонстровано ефективність використання JPA для роботи з базою даних та зручність Vue.js для створення інтерфейсу.