VNPT-SIMN

HỘI THẢO

JAVA SPRING BOOT ỨNG DỤNG DỰ ÁN

Trình bày: Lưu cang kim long

Phối hợp: Huỳnh Đình Kim Điền
Hồ Huy Luật
Nguyễn Quốc Khánh
Phạm Trương Thị Tuyết Nhi

Nội Dung

LÝ THUYẾT

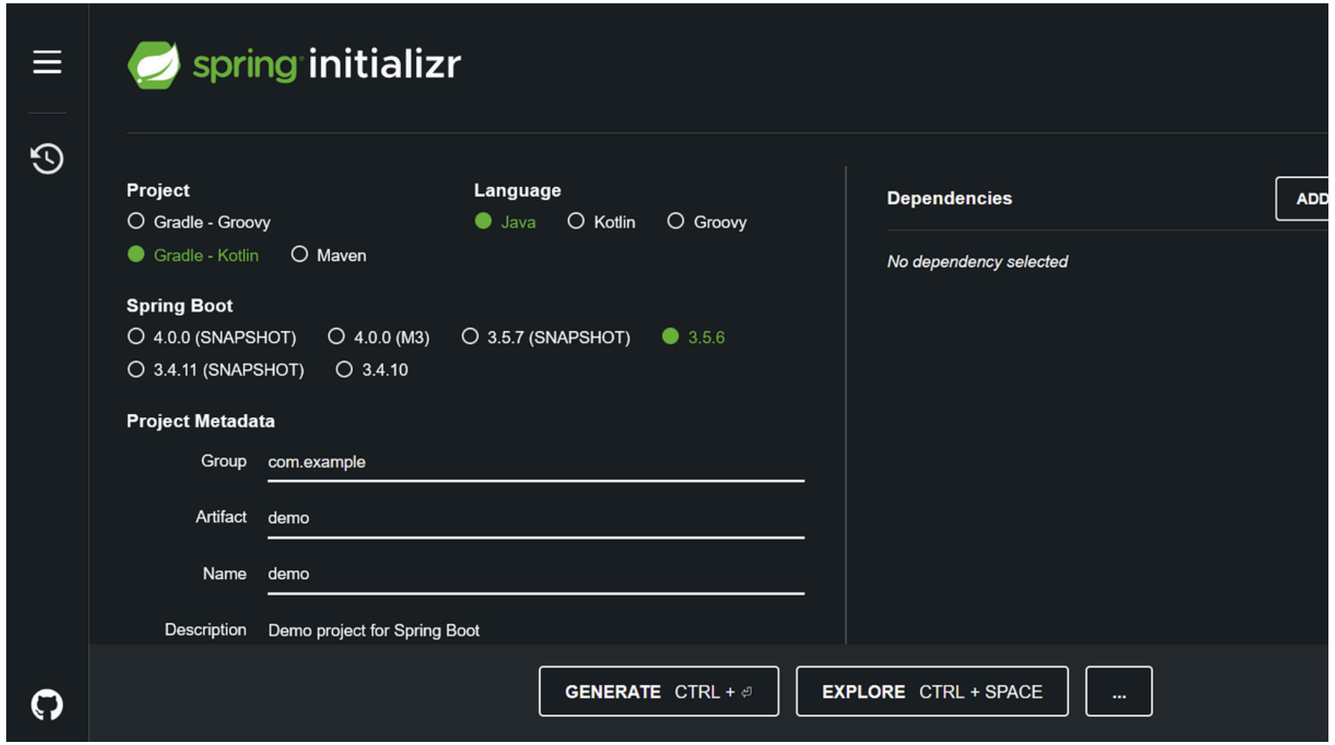
BÀI TẬP MẪU:

I.Tạo dự án Spring Boot đầu tiên

- Đối với Spring Boot một trong hai IDE là Eclipse (miễn phí) và IntelliJ IDEA Ultimate (bản Community không có hỗ trợ Spring).
- Tải xuống:IntelliJ IDEA: *https://www.jetbrains.com/idea/download/*

Khởi tạo Spring

Spring Initializr có thể truy cập trên web tại *http://start.spring.io/*, hoặc với IntelliJ thì có tích hợp luôn vào khi tạo project luôn.



JAVA CODE

I.1. Lập trình hướng đối tượng (OOP) : Đóng gói, kế thừa, đa hình, trừu tượng

1. Đóng gói (Encapsulation)

Là kỹ thuật ẩn thông tin bên trong lớp và chỉ cho phép truy cập thông qua getter/setter.

```
class Person {
    private String name;  // private -> không truy cập trực
    private int age;

    // Getter và Setter
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }
}
```

công cụ hỗ trợ viết lớp Java theo nguyên tắc OOP nhanh và gọn hơn.

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.12</version>
  </dependency>
```

JAVA CODE

I.1. Lập trình hướng đối tượng (OOP)

2. Kế thừa (Inheritance): Cho phép tái sử dụng code, lớp con thừa hưởng thuộc tính và phương thức của lớp cha.

Cú pháp cơ bản

```
class SuperClass {
    void display() {
        System.out.println("This is the SuperClass");
    }
}

class SubClass extends SuperClass {
    void show() {
        System.out.println("This is the SubClass");
    }
```

Sử dụng:

```
public class Main {
    public static void main(String[] args) {
        SubClass obj = new SubClass();
        obj.display(); // kế thừa từ SuperClass
        obj.show();    // phương thức riêng của SubC
    }
}
```

JAVA CODE I.1. Lập trình hướng đối tượng (OOP)

3. Đa hình (Polymorphism):

1. Overloading (Nạp chồng) – Compile-time Polymorphism, Khái niệm: cùng tên phương thức, khác số lượng hoặc kiểu tham số.

```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }
}

public class Main {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        System.out.println(calc.add(2, 3));     //
        System.out.println(calc.add(2.5, 3.5)); //
    }
}
```

2. Overriding (Ghi đè) – Runtime Polymorphism

Khái niệm: lớp con ghi đè phương thức lớp cha.

```
class Animal {
    void sound() {
        System.out.println("Some sound");
    }
}

class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Dog barks");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal a = new Dog();
        a.sound(); // Dog barks (đa hình runtime)
    }
}
```

JAVA CODE

I.1. Dependency injection áp dụng vào Spring Boot

Có 3 loại chính:

Constructor-based injection: Dùng inject các module bắt buộc. Các module được inject nằm trong constructor, và được gán lần lượt vào các field.

```java
@Service
public class UserService {
    private final UserRepository userRepository; // dependency bắt buộc

    // Spring sẽ gọi constructor này và inject UserRepository
    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    public User findById(Long id) {
        return userRepository.findById(id).orElse(null);
    }
}
```

JAVA CODE

I.1. Dependency injection áp dụng vào Spring Boot

- Setter-based injection: Dùng inject các module tùy chọn. Mỗi module sẽ được inject thông qua setter, nằm ở tham số và cũng gán cho field nào đó.

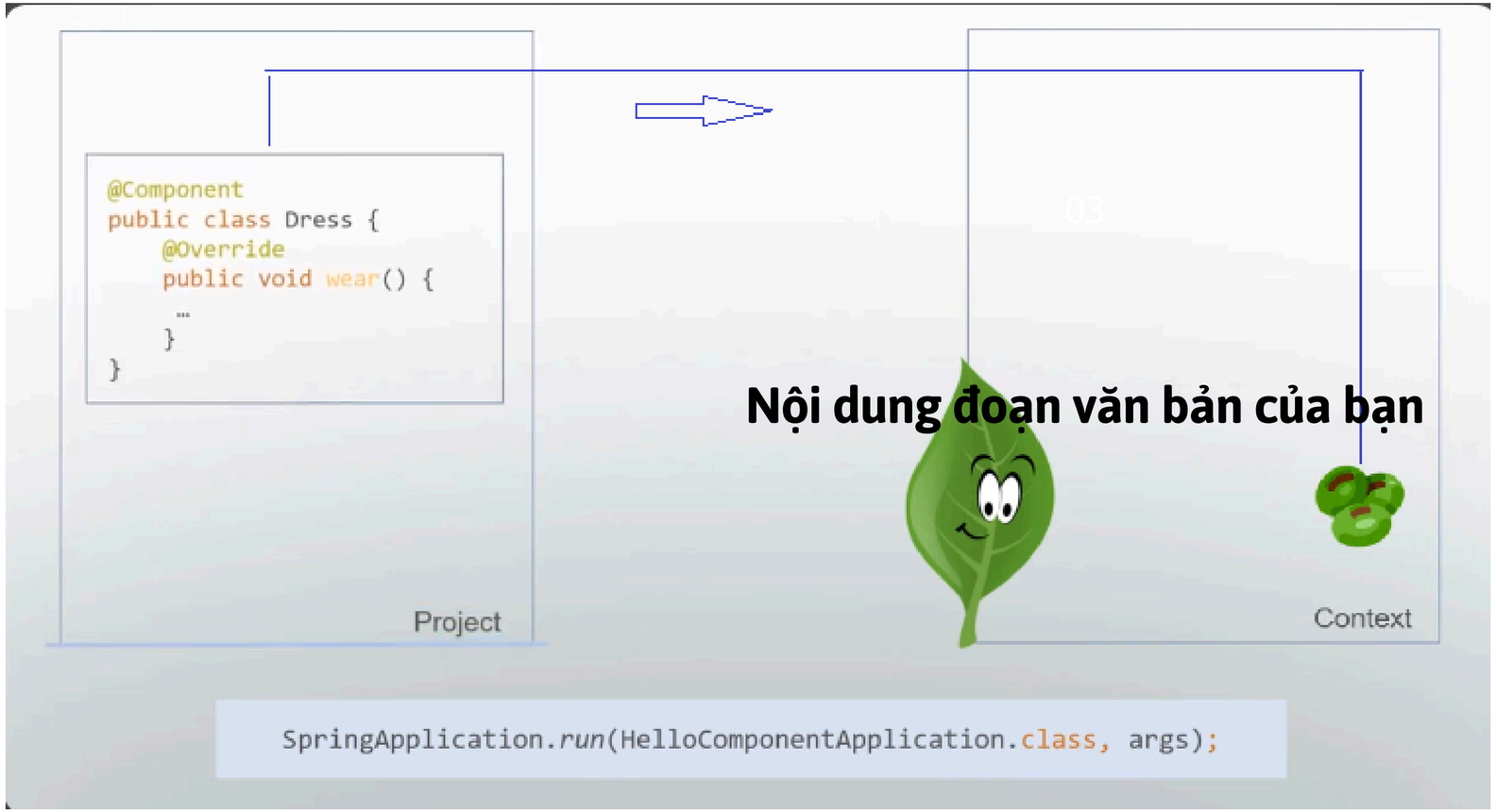
```java
@Service
public class UserService {
    private final UserRepository userRepository; // dependency bắt buộc

    // Spring sẽ gọi constructor này và inject UserRepository
    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    public User findById(Long id) {
        return userRepository.findById(id).orElse(null);
    }
}
```

JAVA CODE I.1. Dependency injection áp dụng vào Spring Boot

- annotation @autowired

```java
@Service
public class UserService {
    @Autowired
    private UserRepository userRepository; // Field injection
}
```

II. Spring Bean



context.getBean(Dress.class)

II. Spring Bean- Cách tạo Bean

1. **Dùng @Component, @Repository, @Service, @Controller**

```
@Component  new *
public class EmailService implements MessageService {
    @Override  no usages  new *
    public void SendMessage(String msg) {
        System.out.println("Email Message Sent: " + msg);
    }
}
```

II. Spring Bean- Cách tạo Bean

- **Dùng @Bean trong class có annotation @Configuration**

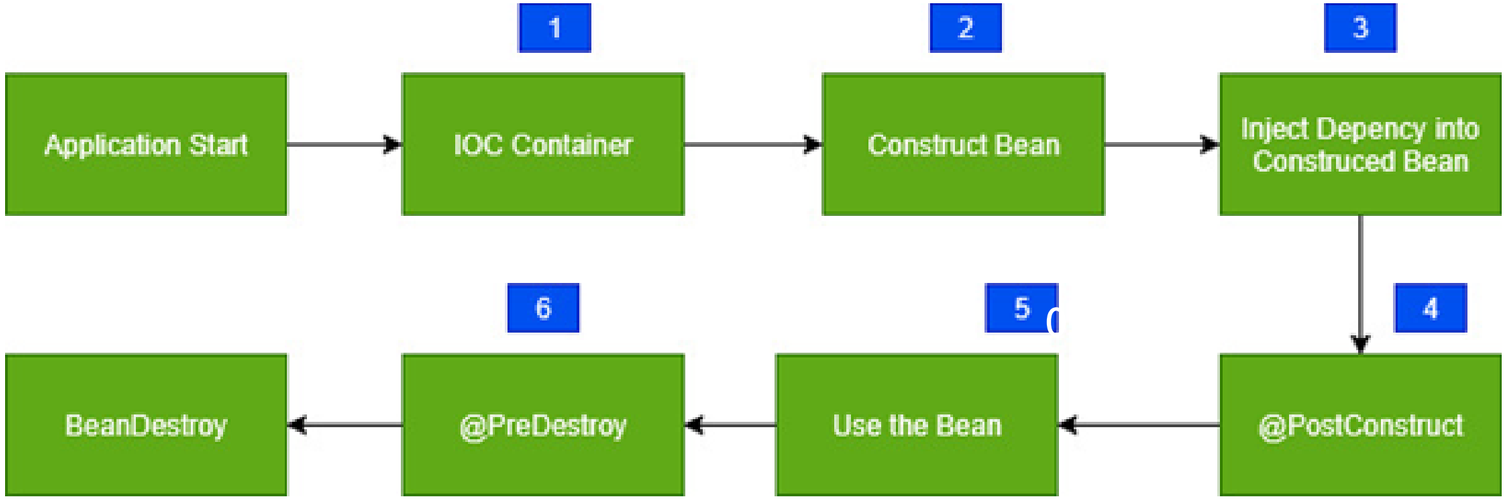
  @Configuration: Là một Annotation đánh dấu trên một class, cho biết rằng lớp đó chứa các thông tin cấu hình cho ứng dụng. Spring Boot sẽ tìm và quét các class được đánh dấu @Configuration để tạo và quản lý các beans.

  @Bean: Là một Annotation đánh dấu trên một method trong class được đánh dấu @Configuration.
  Nó cho Spring Boot biết rằng method đó trả về một bean, và Spring Boot nên quản lý bean đó trong ứng dụng.


```
@Configuration  new *
public class AppConfig {


    @Bean  new *
    public Boy boy(){
        return new Boy();
    }
}
```

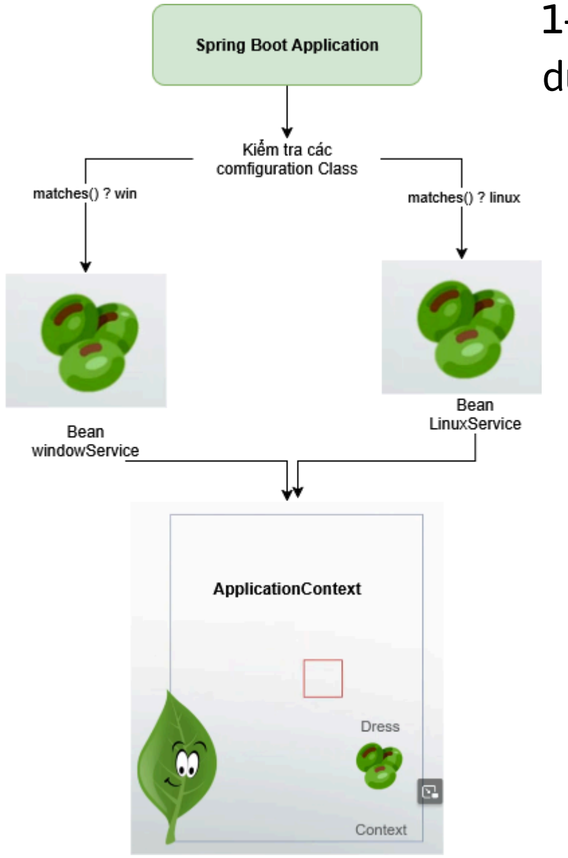
Ghi chú: tại sao đã có cách 1 @Component mà lại xuất hiện cách 2 @Bean, vì 1 số trường hợp không phải lúc nào cũng dùng được @Component ví dụ ta muốn sử dụng 1 só thư viện bên ngoài ta không thể thực hiện @ Component được , lúc đó dùng @Bean

II. Spring Bean- Lifecycle của Bean



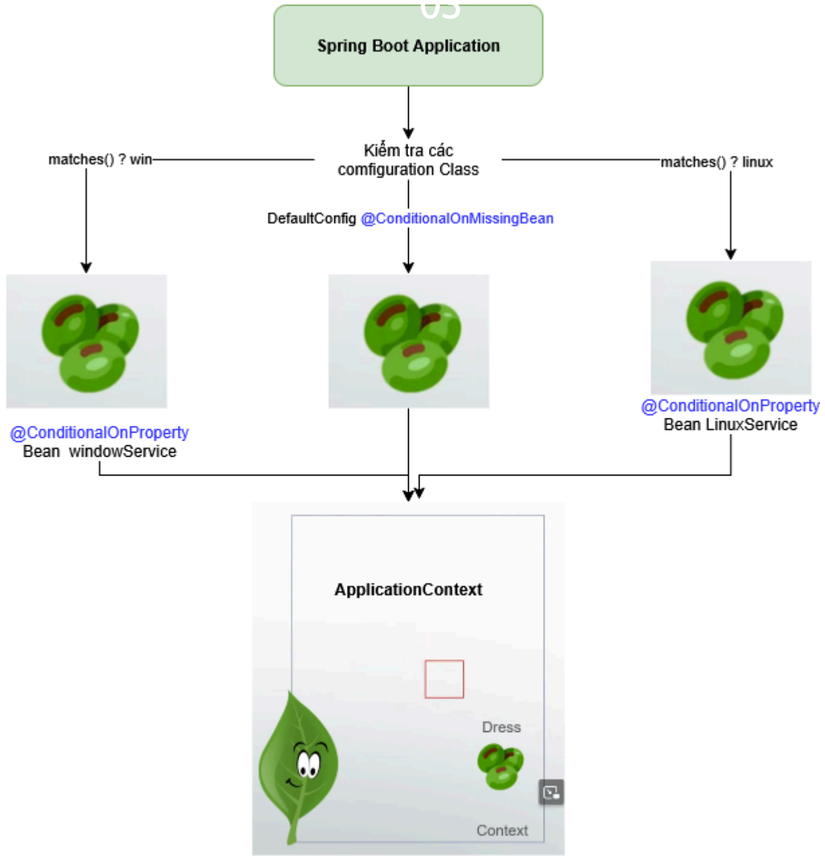
Cơ chế @Lazy: Bean sẽ chỉ được khởi tạo khi nó thực sự được sử dụng, thay vì tạo ra ngay khi container Spring start.

BEAN -@Conditional và @ConditionalOnProperty



1- @**Conditiona** :một annotation được sử dụng để kiểm soát việc tạo bean trong ApplicationContext dựa trên các điều kiện cụ thể tại thời điểm runtime.

2- @ConditionalOnProperty : kiểm tra property để tạo bea, nếu không có bean nào thoả mản thì bean @ConditionalOnMissingBean sẽ được chọn



@EnableJpaRepositories (kết nối nhiều loại csdl)

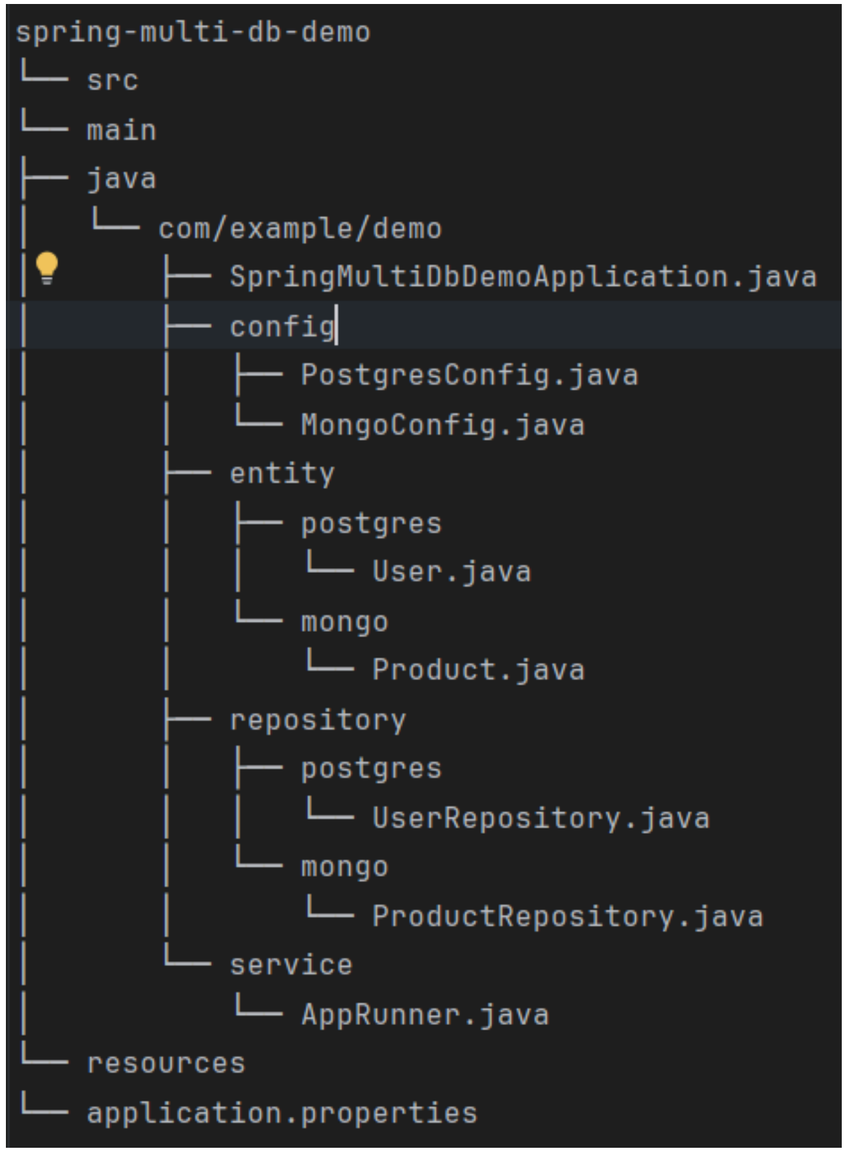
- Spring Data JPA dùng annotation này để bật khả năng quét các interface Repository (extends JpaRepository, CrudRepository...) và tự tạo bean cho chúng.
- Khi dùng nhiều database hoặc nhiều entity manager, cần khai báo rõ basePackages và entity manager reference, để Spring biết repository nào thuộc database nào.

Cấu trúc cơ bản

```
@EnableJpaRepositories(
    basePackages = "com.example.demo.repository.postgres",
    entityManagerFactoryRef = "postgresEntityManager",
    transactionManagerRef = "postgresTransactionManager"
)
```

- **basePackages:** package chứa các repository cần quét
- **entityManagerFactoryRef:** entity manager mà repository này sẽ sử dụng
- **transactionManagerRef:** transaction manager để quản lý transaction của repository

@EnableJpaRepositories



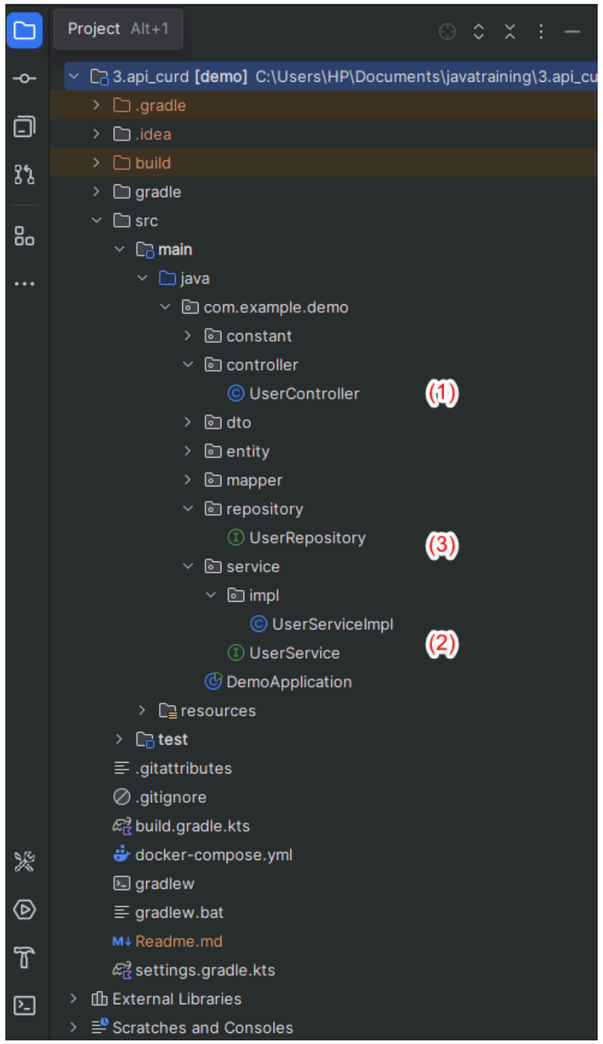
- Database PostgreSQL

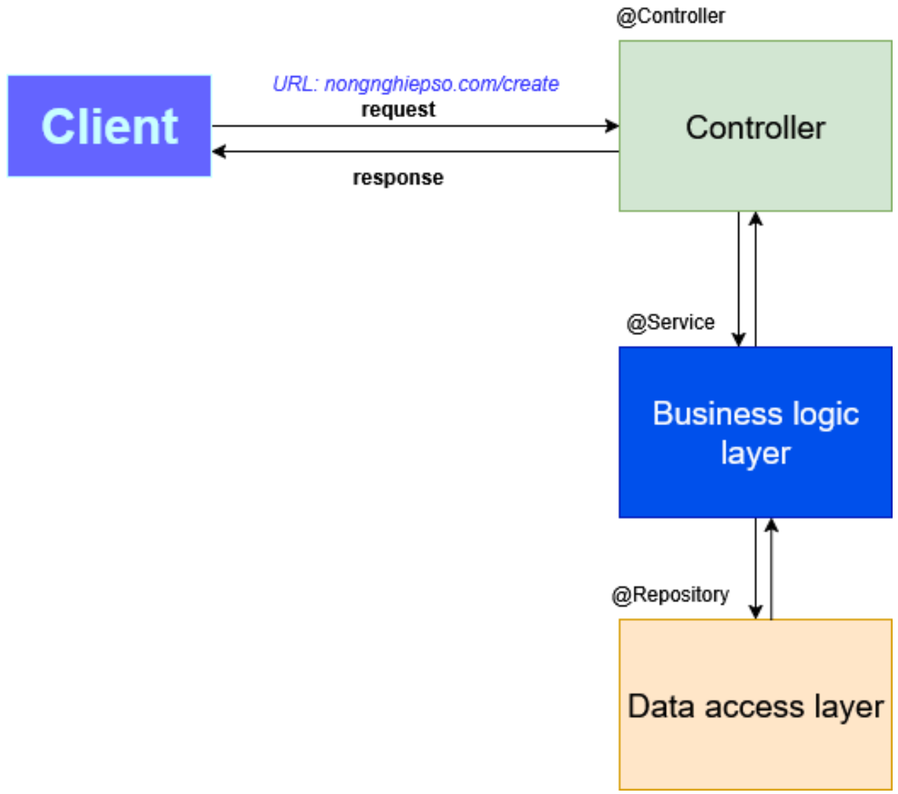
```java
@Configuration  new *
@EnableJpaRepositories(
        basePackages = "com.example.demo.repository.postgres",
        entityManagerFactoryRef = "postgresEntityManager",
        transactionManagerRef = "postgresTransactionManager"
)
@EntityScan(basePackages = "com.example.demo.entity.postgres")
public class PostgresConfig {
```

- Database Mongodb

```java
@Configuration  new *
@EnableMongoRepositories(basePackages = "com.example.demo.repository.mongo",
                         mongoTemplateRef = "mongoTemplate")
public class MongoConfig {

    @Bean  new *
    public MongoTemplate mongoTemplate() {
        return new MongoTemplate(MongoClients.create("mongodb://localhost:27017/testdb"),
                databaseName: "testdb");
    }
```

Luồng của ứng dụng





Các annotation dùng để ánh xạ dữ liệu từ HTTP request vào tham số của Controller

🧩 1. @PathVariable – Lấy giá trị từ đường dẫn URL

GET http://localhost:8080/users/5

```
@GetMapping("/users/{id}")
public String getUserById(@PathVariable Long id) {
    return "User ID là: " + id;
}
```

🧩 2. @RequestParam – Lấy dữ liệu từ query string (phần sau dấu ?)

GET http://localhost:8080/search?name=nguyenvanA

```
@GetMapping("/search")
public String searchUser(@RequestParam String name) {
    return "Tìm người dùng: " + name;
}
```

🧩 3. @RequestBody – Lấy dữ liệu từ phần body (JSON/XML) của request

Request gửi:

```
{
  "name": "Long",
  "email": "long@gmail.com"
}
```

```
@PostMapping("/users")
public String createUser(@RequestBody User user) {
    return "Tạo user: " + user.getName();
}
```

🧩 4. @RequestHeader – Lấy dữ liệu từ header của request

Khi request có header:

X-Tenant: agrihub

```
@GetMapping("/tenant")
public String getTenant(@RequestHeader("X-Tenant") String tenant) {
    return "Tenant hiện tại là: " + tenant;
}
```

BÀI CODE MẪU

- CHẠY DEMO PROJECT MẪU THẢO LUẬN

01 1. JAVA SPRING BOOT + MYSQL + CRUD

02 2. JAVA SPRING BOOT + REDIS

03 3. JAVA SPRING BOOT + KAFKA

04 4. JAVA SPRING BOOT + MINIO

05 5. MinIO

06 6. JAVA SPRING BOOT + ELASTIC

Xin cảm ơn!