



Lập trình Backend Spring Data JPA

Ths. Vũ Duy Khương

1

Giới thiệu Spring Data JPA

2

Sử dụng @Query, Query Creation

3

Sử dụng @NameQuery trong Entity

4

Sử dụng @Modifying

Spring data JPA, JPA, Hibernate

Spring Data JPA

Hibernate

EclipseLink



Database



Database



Database

Lý do sử dụng Spring Data JPA

- ❖ Thao tác với 1 bảng trong csdl thường có các phương thức phổ biến: insert, update, delete, select.
- ❖ Với mỗi repository đều cần phải triển khai các hàm CRUD.
Việc này khiến mã nguồn bị lặp đi, lặp lại nhiều lần.
- ❖ Để khắc phục vấn đề trên thì Spring Data JPA ra đời. Spring data jpa cung cấp một tập các repository interface. Chỉ cần extends thì có thể sử dụng được các hàm CRUD

Giới thiệu về Spring Data JPA

- ❖ Phần cơ bản Spring Data JPA là tập repository interface.
- ❖ Dạng *Repository interface*<*Entity, ID*>
- ❖ Nó nhận vào một Entity class đại diện cho một bảng trong database, và kiểu dữ liệu của trường ID trong bảng đó.

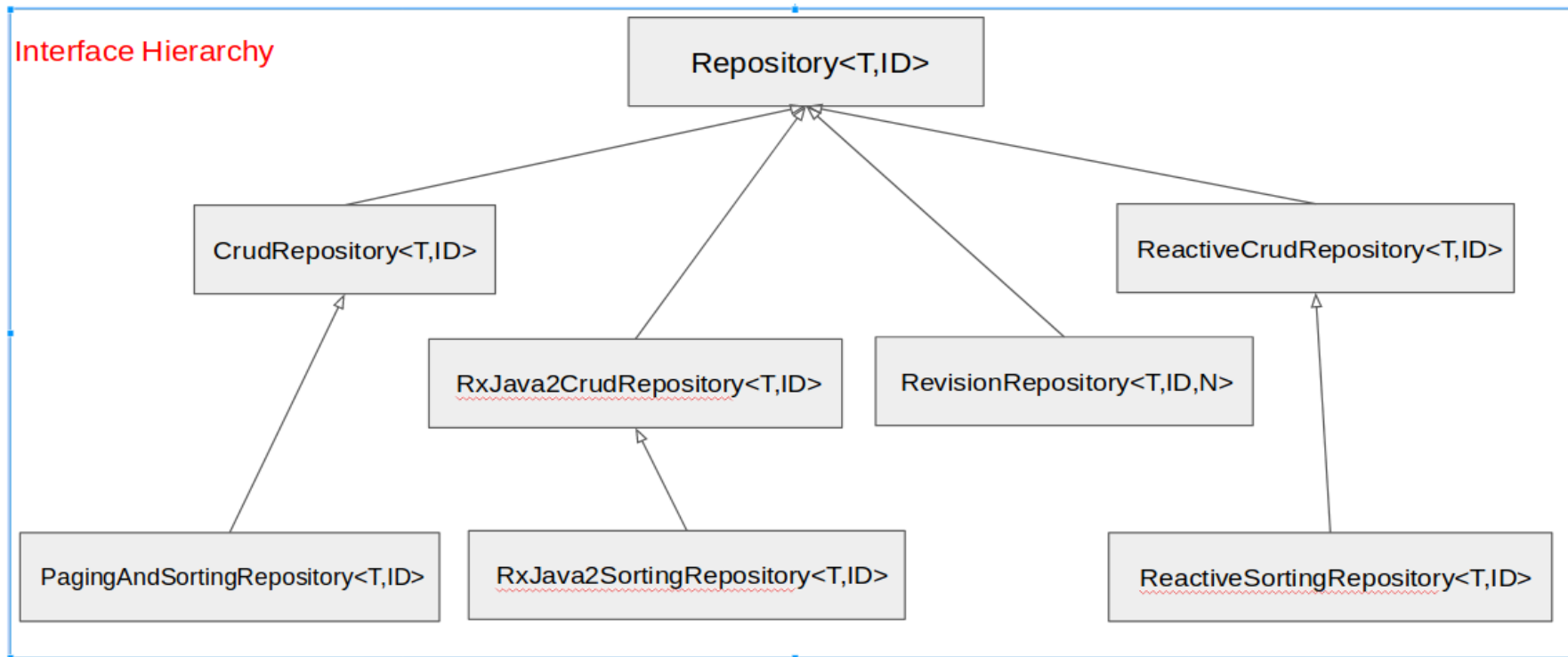
Giới thiệu về Spring Data JPA

Repository Interface cung cấp một số function xoay quanh Entity:

- Thêm, sửa, xóa Entity
- Tìm kiếm
- Lấy danh sách Entity

Giới thiệu về Spring Data JPA

Một số interface trong SpringDataJPA



Giới thiệu về Spring Data JPA

Ví dụ Spring Data JPA:

```
// This is an Interface.  
// No need Annotation here.  
public interface EmployeeRepository extends CrudRepository<Employee, Long> { // Long: Type of Employee ID.  
  
    Employee findByEmpNo(String empNo);  
  
    List<Employee> findByFullNameLike(String fullName);  
  
    List<Employee> findByHireDateGreaterThan(Date hireDate);  
  
    @Query("SELECT coalesce(max(e.id), 0) FROM Employee e")  
    Long getMaxId();  
}
```


Sử dụng @Query, Query Creation

@Query :

- ❖ Một annotation định nghĩa một câu Query tương ứng tham số truyền vào.
- ❖ Có 2 cách sử dụng:

Cách 1: Sử dụng JPQL :

```
@Transactional
public interface DepartmentAnnotationRepository extends JpaRepository<Department,Integer> {

    @Query("select department from Department department")
    Department findAllDepartment();
}
```

Sử dụng @Query, Query Creation

@Query :

Cách 2: Sử dụng Native Query :

```
@Query(  
    value = "SELECT * FROM Department u WHERE u.status = 1",  
    nativeQuery = true)  
Collection<Department> findAllDepartment();
```

Để sử dụng câu lệnh Query thuần giống như ta thực hiện câu Select trong Database thì mình thêm tham số

nativeQuery = true

Sử dụng @Query, Query Creation

@Query :

- Tham số trong @query:

```
@Query("select department from Department department where department.name = ?1" and department.code = ?2)  
Department findByName(String departmentName, int code);
```

@Query("select d from department d where d.name = : d_name and
d.code= :code")

Department findByName(@Param("d_name" String department,
@Param("code") String code)

Sử dụng @Query, Query Creation

Query Creation:

- ❖ Spring Data JPA hỗ trợ cho chúng ta sẵn các phương thức để truy cập vào database.
- ❖ Chúng ta chỉ cần kế thừa JPA Repository là có thể sử dụng được các phương thức mà JPA cung cấp để lấy dữ liệu từ Database.

Sử dụng @Query, Query Creation

Query Creation:

- Cấu trúc câu lệnh :

```
public interface DepartmentQueryCreationRepository extends JpaRepository<Department,Integer> {  
  
    List<Department> findByName (String name);  
    List<Department> findByNameLike (String name);  
    List<Department> findByNameContaining (String name);  
    List<Department> findByNameStartingWith(String name);  
    List<Department> findByNameEndingWith(String name);  
    List<Department> findByNameIgnoreCase(String name);  
  
    /* List<Department> findByNameAndLocal(String name,String local);  
    List<Department> findByNameOrLocal(String name,String local);  
    List<Department> findByNameNot(String name);  
    List<Department> findByDateAfter(Date date);  
    List<Department> findByDateBefore (Date date);  
    List<Department> findByDateBetween(Date from,Date to); */  
}
```

Sử dụng @Query, Query Creation

Query Creation:

- Trong đó **findBy** là từ khoá mà JPA cung cấp, sau từ findBy là tên cột trong database.
- Ví dụ: *findByName*
tìm kiếm các user có tên là tham số name truyền vào. Trong đó, findBy là từ khoá của JPA và Name chính là tên cột trong database. Ngoài findBy thì JPA còn hỗ trợ nhiều phương thức khác.

Sử dụng @NamedQuery trong Entity

@NamedQuery :

```
@Entity
@Table(name = "employee", schema="spring_data_jpa_example")
@NamedQuery(name = "Employee.fetchByLastNameLength",
            query = "SELECT e FROM Employee e WHERE CHAR_LENGTH(e.lastname) =:length ")
)
public class Employee {

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;

    @Column(name = "firstname")
    private String firstName;

    @Column(name = "lastname")
    private String lastname;
}
```

Sử dụng @NamedQuery trong Entity

@NamedQuery :

Trong câu lệnh :

@NamedQuery(query = 'Câu lệnh SQL ')

Chính là câu lệnh thực tế sẽ tương tác với DataBase

Sử dụng @NamedQuery trong Entity

@NamedQuery :

Trong Class Entity, mình sử dụng **@NamedQuery** để tạo câu lệnh Select. Để gọi được câu lệnh `@NamedQuery(name = “Employee.fetchByLastNameLength”)` thì ở JPA Repository, ta phải có phương thức (*fetchByLastNameLength*) giống y như vậy.

Sử dụng @NameQuery trong Entity

@NameQuery : Class Repository

```
@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long>, EmployeeRepositoryCustom {

    List<Employee> fetchByLastNameLength(@Param("length") Long length);
}
```

Bắt buộc có phương thức :
fetchByLastNameLength

Sử dụng @Modifying

@Modifying : update dữ liệu

```
@Modifying
@Query("update User u set u.status = :status where u.name = :name")
int updateUserSetStatusForName(@Param("status") Integer status,
    @Param("name") String name);
```

Kết quả trả về là số lượng dòng đã được cập nhật trong Database.

```
@Modifying
@Query(value = "update Users u set u.status = ? where u.name = ?",
    nativeQuery = true)
int updateUserSetStatusForNameNative(Integer status, String name);
```

Có thể sử dụng Native Query để cập nhật như sau

Sử dụng @Modifying

@Modifying : Insert dữ liệu

Trong Spring Data JPA, chúng ta dùng hàm **Save()** có sẵn để Insert dữ liệu xuống Database. Trong trường hợp dùng Native Query thì chúng ta phải kết hợp @Modifying và câu lệnh Insert chung với nhau vì Spring Data JPA không hỗ trợ chức năng Insert

```
@Modifying
@Query(
    value =
        "insert into Users (name, age, email, status) values (:name, :age, :email, :status)",
    nativeQuery = true)
void insertUser(@Param("name") String name, @Param("age") Integer age,
    @Param("status") Integer status, @Param("email") String email);
```

Thực hành

Bài tập: Sử dụng Spring Boot với Spring Data JPA để thực hiện ví dụ thêm, sửa, xóa dữ liệu với database MySQL. Sau đó hiển thị dữ liệu lên trình duyệt web với **Thymleaf**

Các công nghệ sử dụng:

- Spring Boot 2.0.2
- Maven
- JDK 1.8
- Eclipse + Spring Tool Suite
- Thymeleaf

Thực hành

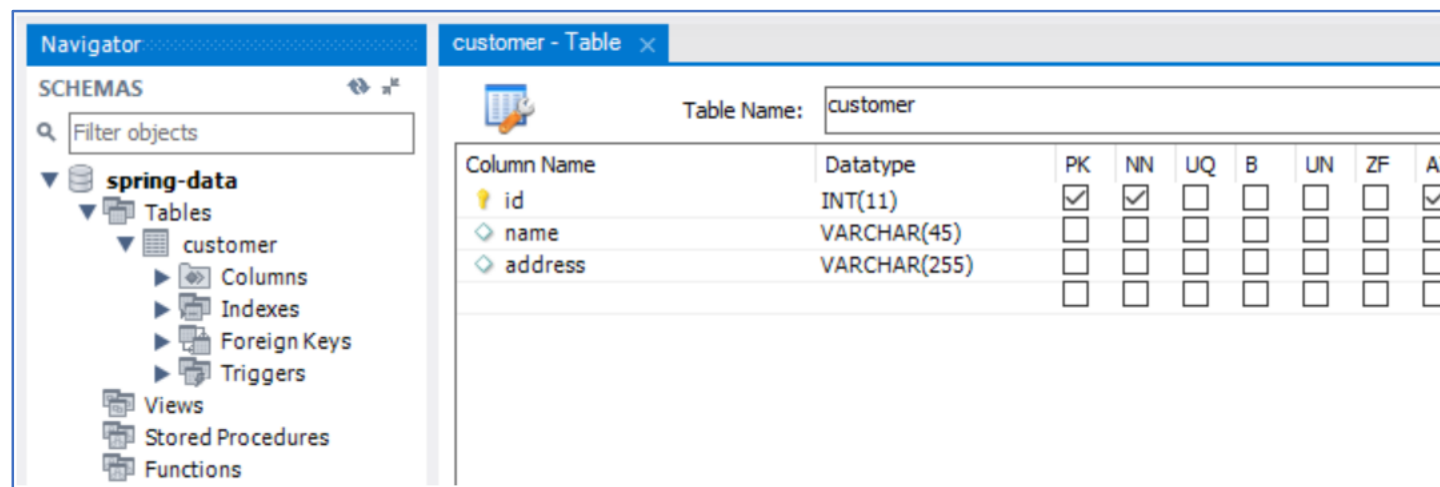
Bước 1: Tạo Database

Tạo database **spring-data** với table **customer**

```

3 CREATE DATABASE IF NOT EXISTS `spring-data`;
4 CREATE TABLE `spring-data`.`customer` (
5   `id` int(11) NOT NULL AUTO_INCREMENT,
6   `name` varchar(45) DEFAULT NULL,
7   `address` varchar(255) DEFAULT NULL,
8   PRIMARY KEY (`id`)
9 );

```

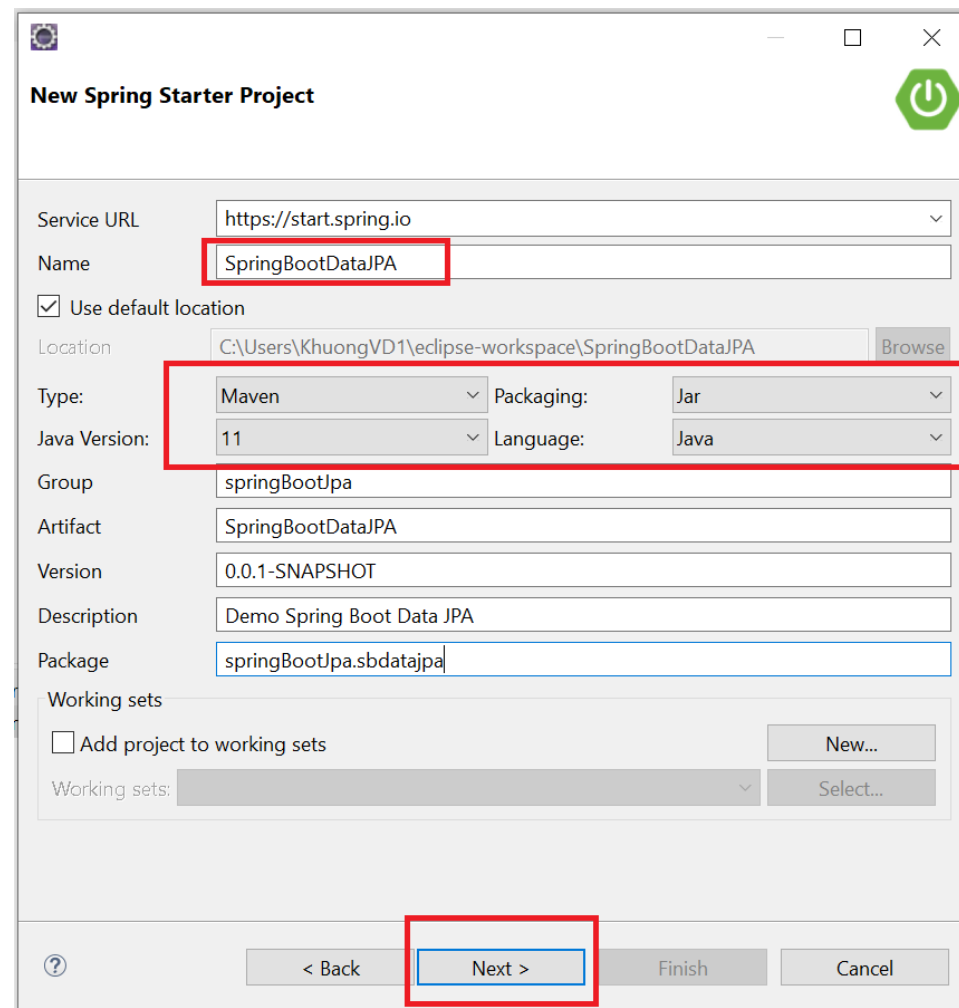
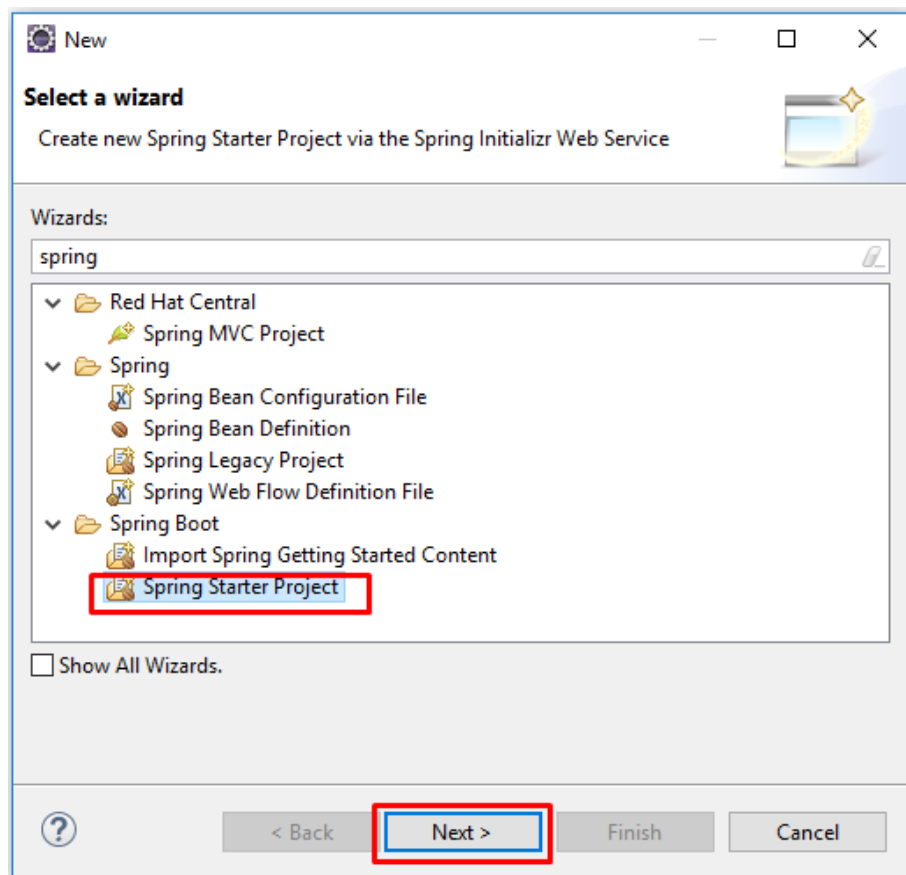


The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Navigator' pane displays the 'spring-data' database schema, with the 'customer' table selected under the 'Tables' folder. On the right, the 'customer - Table' tab shows the table's structure. The 'Table Name' is 'customer'. Below it, a table lists the columns: 'id' (INT(11), PK, NN, AI), 'name' (VARCHAR(45)), and 'address' (VARCHAR(255)).

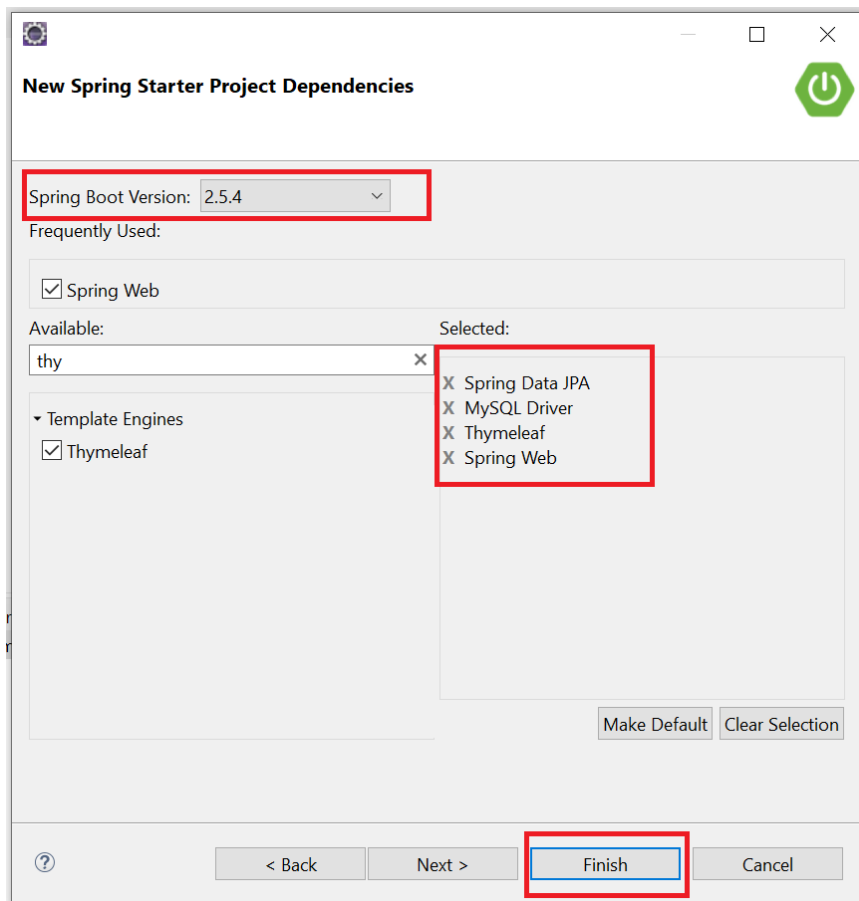
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI
id	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
name	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
address	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Thực hành

Bước 2: Tạo Spring Boot Project

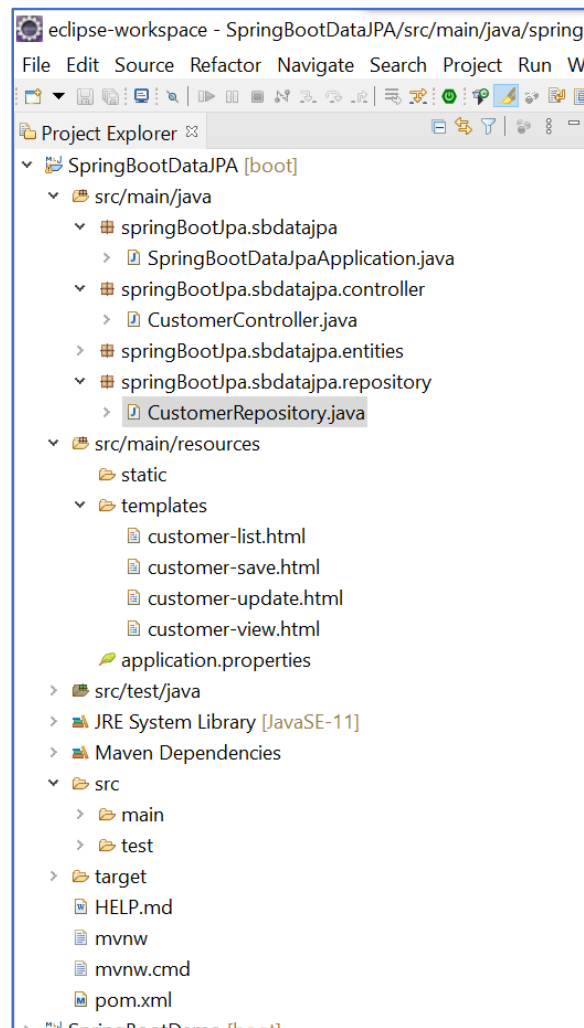


Bước 2: Tạo Spring Boot Project



Thực hành

Bước 2: Cấu trúc Project



Thực hành

Bước 3: Cấu hình Hibernate

Cấu hình các thông số connect tới database và config JPA trong file application.properties.

A screenshot of an IDE window showing the configuration of application.properties. The window has several tabs at the top: 'CustomerCon...', 'customer-lis...', 'customer-sav...', 'customer-upd...', 'customer-vi...', and 'application...'. The 'application...' tab is active. The code in the editor is as follows:

```
1 ## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
2 spring.datasource.driver-class-name=com.mysql.jdbc.Driver
3 spring.datasource.url = jdbc:mysql://localhost:3306/spring_data?useSSL=false
4 spring.datasource.username=root
5 spring.datasource.password=admin
6 ## =====JPA / HIBERNATE=====
7 spring.jpa.show-sql=true
8 spring.jpa.hibernate.ddl-auto=none
9 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
10
```

Thực hành

Bước 3: Cấu hình Hibernate

- Cấu hình các thông số connect tới database và config JPA trong file `application.properties`.
- Mặc định Spring Boot sẽ tự động cấu hình JPA và các bean liên quan gồm `DataSourceAutoConfiguration`, `DataSourceTransactionManagerAutoConfiguration`, `HibernateJpaAutoConfiguration` theo cấu hình trong file **`application.properties`**

Thực hành

Bước 4: File Repository

```
CustomerRepo... x customer-lis... customer-sav... customer-upd... customer-vi... application.... »2
1 package springBootJpa.sbdajpa.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5
6 import springBootJpa.sbdajpa.entities.Customer;
7 |
8 @Repository
9 public interface CustomerRepository extends JpaRepository<Customer, Integer> {
10
11 }
```

Khi extends JpaRepository ta có thể sử dụng luôn các method như findAll, findById, save, delete, deleteById...mà không cần phải tạo các class DAO cho nó.

Thực hành

Bước 5: File Entity

```
Customer.java CustomerRepo... customer-sav... customer-upd... customer-vi... application
1 package springBootJpa.sbdajpa.entities;
2
3 import javax.persistence.Column;
4 import javax.persistence.Entity;
5 import javax.persistence.GeneratedValue;
6 import javax.persistence.GenerationType;
7 import javax.persistence.Id;
8 import javax.persistence.Table;
9
10 @Entity
11 @Table(name = "customer")
12 public class Customer {
13     @Id
14     @Column(name = "id")
15     @GeneratedValue(strategy = GenerationType.IDENTITY)
16     private int id;
17     @Column(name = "name")
18     private String name;
19     @Column(name = "address")
20     private String address;
21     // getter - setter
22 }
```

Thực hành

Bước 5: File Controller

```

1 package springBootJpa.sbdajpa.controller;
2
3 import java.util.Optional;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.stereotype.Controller;
6 import org.springframework.ui.Model;
7 import org.springframework.web.bind.annotation.ModelAttribute;
8 import org.springframework.web.bind.annotation.PathVariable;
9 import org.springframework.web.bind.annotation.RequestMapping;
10
11 import springBootJpa.sbdajpa.entities.Customer;
12 import springBootJpa.sbdajpa.repository.CustomerRepository;
13
14 @Controller
15 public class CustomerController {
16     @Autowired
17     private CustomerRepository customerRepository;
18     @RequestMapping(value={"/", "/customer-list"})
19     public String listCustomer(Model model) {
20         model.addAttribute("listCustomer", customerRepository.findAll());
21         return "customer-list";
22     }
23     @RequestMapping("/customer-save")
24     public String insertCustomer(Model model) {
25         model.addAttribute("customer", new Customer());
26         return "customer-save";
27     }
28     @RequestMapping("/customer-view/{id}")
29     public String viewCustomer(@PathVariable int id, Model model) {
30         Optional<Customer> customer = customerRepository.findById(id);
31         if (customer.isPresent()) {
32             model.addAttribute("customer", customer.get());

```

```

32         model.addAttribute("customer", customer.get());
33     }
34     return "customer-view";
35 }
36
37 @RequestMapping("/customer-update/{id}")
38 public String updateCustomer(@PathVariable int id, Model model) {
39     Optional<Customer> customer = customerRepository.findById(id);
40     if (customer.isPresent()) {
41         model.addAttribute("customer", customer.get());
42     }
43     return "customer-update";
44 }
45 @RequestMapping("/saveCustomer")
46 public String doSaveCustomer(@ModelAttribute("Customer") Customer customer, Model model) {
47     customerRepository.save(customer);
48     model.addAttribute("listCustomer", customerRepository.findAll());
49     return "customer-list";
50 }
51 @RequestMapping("/updateCustomer")
52 public String doUpdateCustomer(@ModelAttribute("Customer") Customer customer, Model model) {
53     customerRepository.save(customer);
54     model.addAttribute("listCustomer", customerRepository.findAll());
55     return "customer-list";
56 }
57
58 @RequestMapping("/customerDelete/{id}")
59 public String doDeleteCustomer(@PathVariable int id, Model model) {
60     customerRepository.deleteById(id);
61     model.addAttribute("listCustomer", customerRepository.findAll());
62     return "customer-list";

```

```

62     return "customer-list";
63 }
64 }

```

Q&A





THANK YOU