



SPRING BOOT SPRING DATA JPA

CYBERSOFT.EDU.VN



Nội dung

- ☐ Tổng quan về Spring Boot.
- ☐ Tạo dự án bằng Spring Boot.
- ☐ Spring JPA là gì?
- ☐ Spring Data Query Creation.
- ☐ Spring Data @Query.

CYBERSOFT

ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

Ôn lại kiến thức

❑ Để tạo một **dự án Spring MVC** phải thực hiện **ít nhất 5 bước** sau:

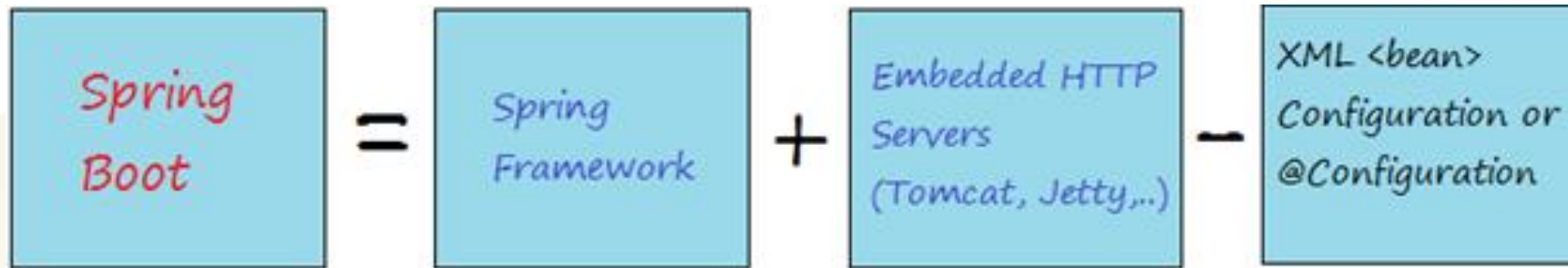
1. Tạo webserver để triển khai ứng dụng lên chạy.
2. Tạo project sử dụng Maven với các dependency cần thiết của Spring MVC và Servlet Api.
3. Cấu hình DispatcherServlet của Spring MVC.
4. Cấu hình Spring MVC.
5. Tạo controller trả về trang JSP khi có request từ client.

➡ Các bước **1, 2, 3, 4** lặp đi lặp lại



Làm thế nào để
tránh lặp lại các
bước này?

Spring Boot là gì?



- ❑ **Spring Boot** là một module của Spring Framework cung cấp tính năng RAD (Rapid Application Development - Phát triển ứng dụng nhanh).
- ❑ **Spring Boot** ra đời nhằm rút ngắn thời gian cài đặt và cấu hình spring mvc project. Tối giản hóa việc tạo beans và các dependences trong project. Giúp các lập trình viên tập trung vào business nghiệp vụ hơn là cấu hình.
- ❑ **Spring Boot** dễ dàng trong việc tích hợp với các hệ sinh thái của Spring như: Spring JDBC, Spring ORM, Spring Security, ...

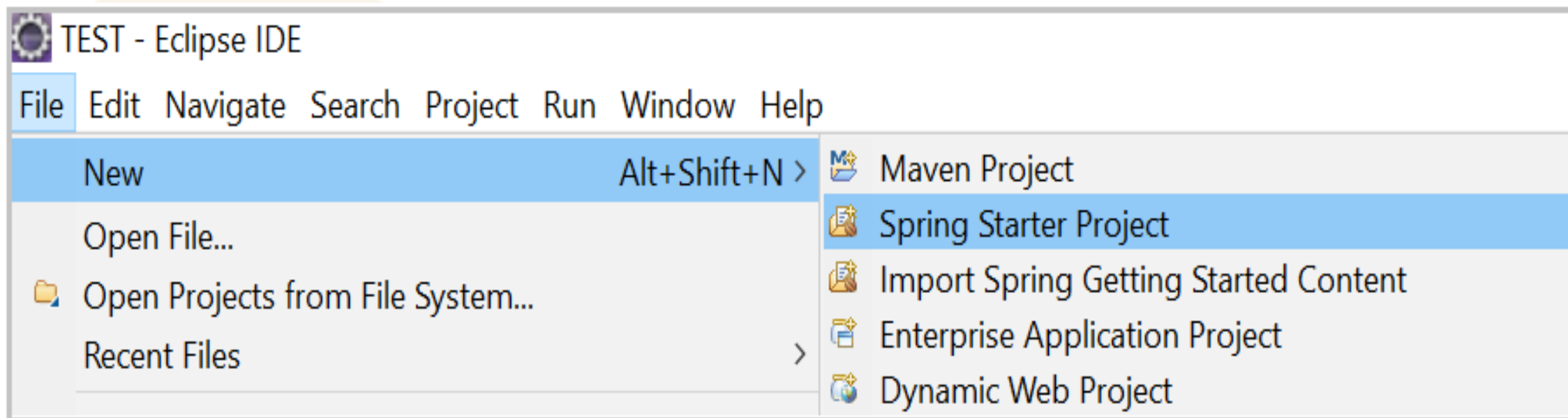
Ưu điểm của Spring Boot



- ☐ Tạo các ứng dụng Spring độc lập.
- ☐ Nhúng trực tiếp **Tomcat**, **Jetty** hoặc **Undertow** (Không cần phải deploy ra file war).
- ☐ Sử dụng các **starter dependency** giúp việc cấu hình Maven đơn giản hơn.
- ☐ **Tự động** cấu hình Spring khi cần thiết.
- ☐ Không sinh code cấu hình và không phải cấu hình bằng XML.
- ☐ Chuẩn cho **Microservices** (Cloud support, giảm việc setup, config các thư viện hỗ trợ).

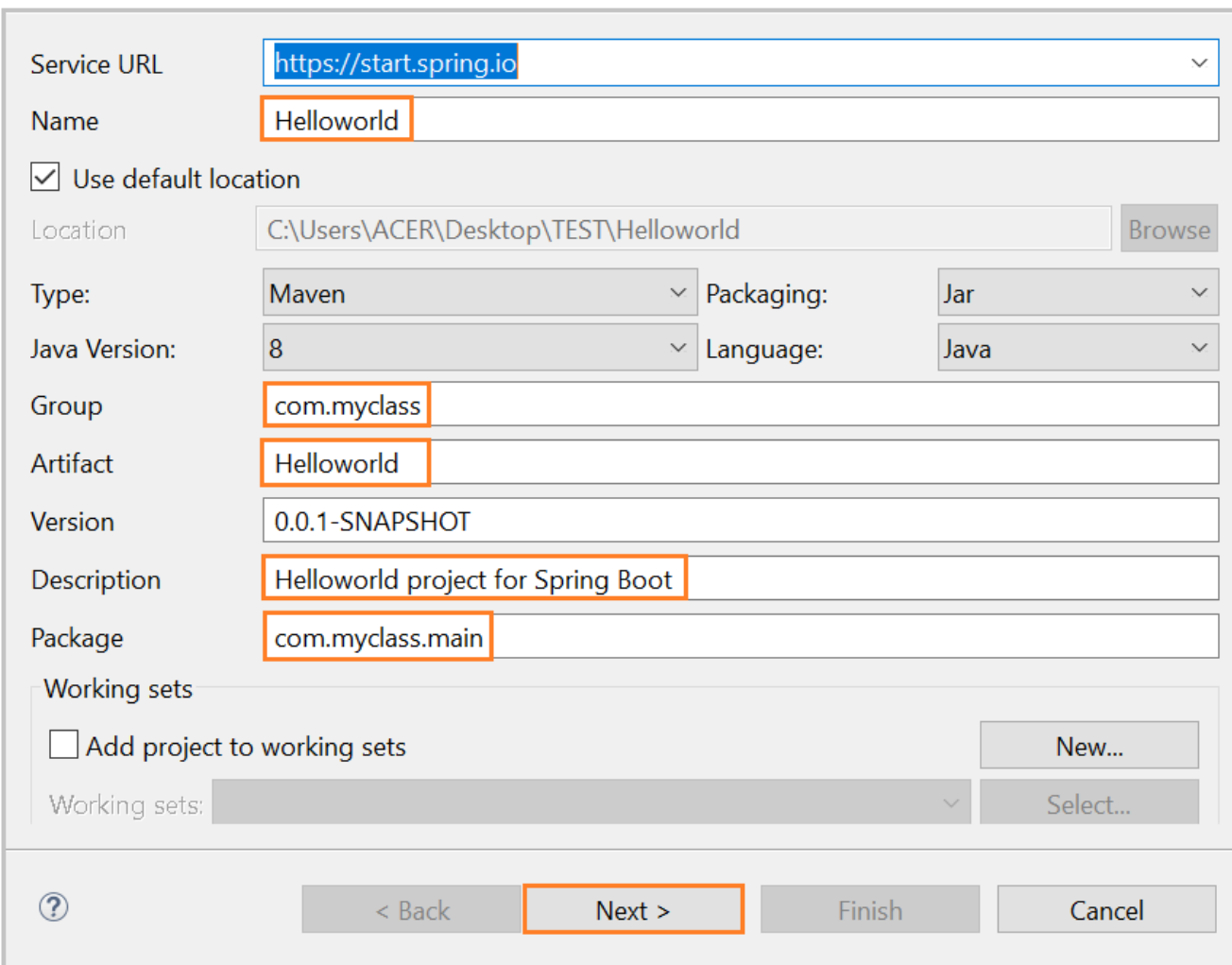

Tạo dự án

❑ Click chuột phải vào **File** → Chọn **New** → Chọn **Spring Starter Project**.



Tạo dự án

❑ Sửa lại thông tin **Name, Group, Description, Package** cho dự án → **Next**



The image shows a dialog box for creating a new project in an IDE. The fields are as follows:

- Service URL: `https://start.spring.io`
- Name: `Helloworld`
- ☒ Use default location
- Location: `C:\Users\ACER\Desktop\TEST\Helloworld` (with a `Browse` button)
- Type: `Maven` (dropdown)
- Packaging: `Jar` (dropdown)
- Java Version: `8` (dropdown)
- Language: `Java` (dropdown)
- Group: `com.myclass`
- Artifact: `Helloworld`
- Version: `0.0.1-SNAPSHOT`
- Description: `Helloworld project for Spring Boot`
- Package: `com.myclass.main`
- Working sets: ☐ Add project to working sets (with a `New...` button)
- Working sets: (dropdown menu) (with a `Select...` button)

At the bottom, there are buttons: `< Back`, `Next >` (highlighted with an orange border), `Finish`, and `Cancel`.

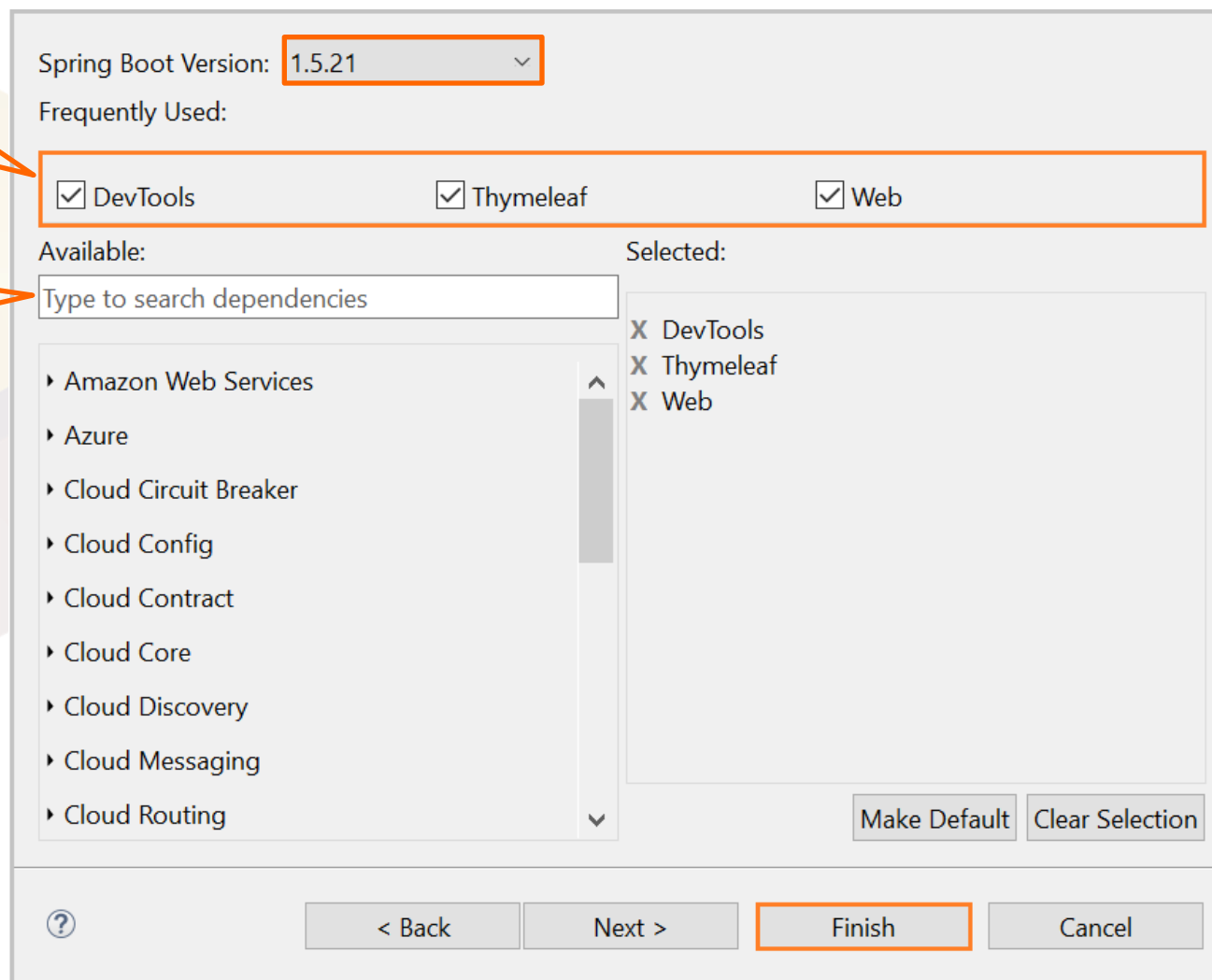
OFT
ÁP TRÌNH

Tạo dự án

❑ Chọn các **dependency** cần thiết cho dự án (Web, Thymeleaf, DevTools).

Tích chọn
dependency

Tìm kiếm
dependency



Spring Boot Version: 1.5.21

Frequently Used:

☒ DevTools ☒ Thymeleaf ☒ Web

Available:

Type to search dependencies

- ▶ Amazon Web Services
- ▶ Azure
- ▶ Cloud Circuit Breaker
- ▶ Cloud Config
- ▶ Cloud Contract
- ▶ Cloud Core
- ▶ Cloud Discovery
- ▶ Cloud Messaging
- ▶ Cloud Routing

Selected:

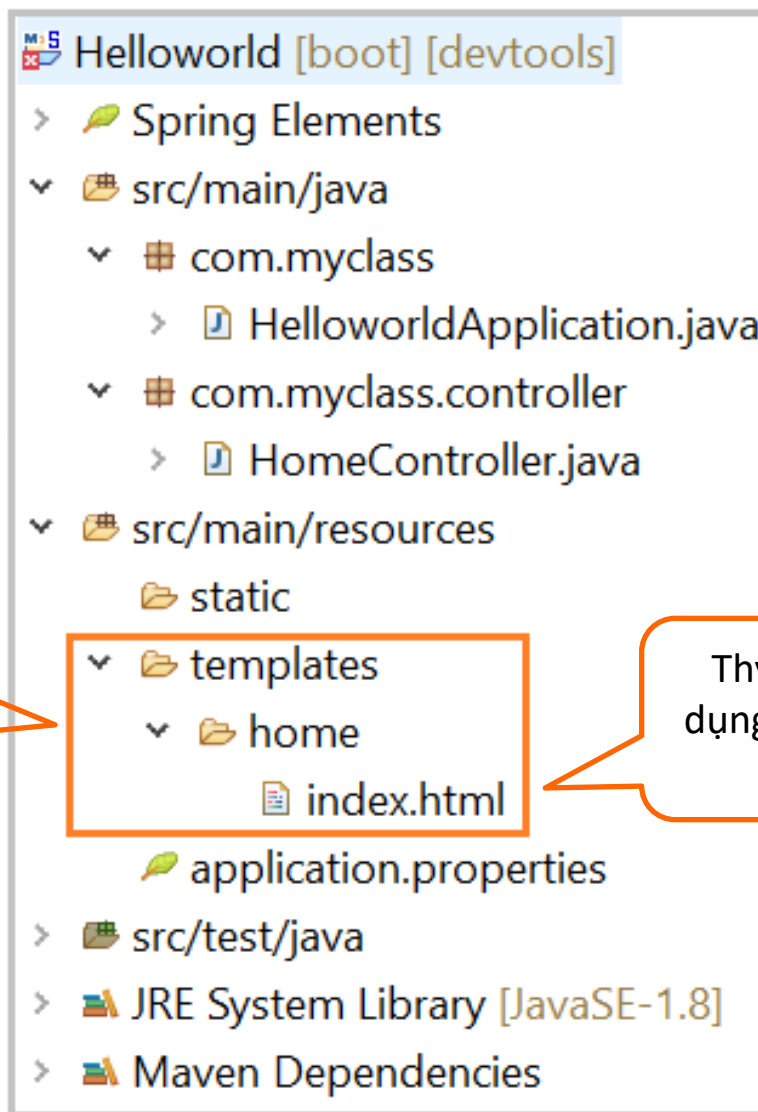
- X DevTools
- X Thymeleaf
- X Web

Make Default Clear Selection

? < Back Next > Finish Cancel

OFT
ÁP TRÌNH

Cấu trúc thư mục



File chính của ứng dụng

Các file view phải đặt trong thư mục **templates**

Thymeleaf sử dụng file có đuôi **.html**

Tạo controller - view

HomeController

```
@Controller
public class HomeController {
    @GetMapping("/")
    public String index() {
        return "home/index";
    }
}
```

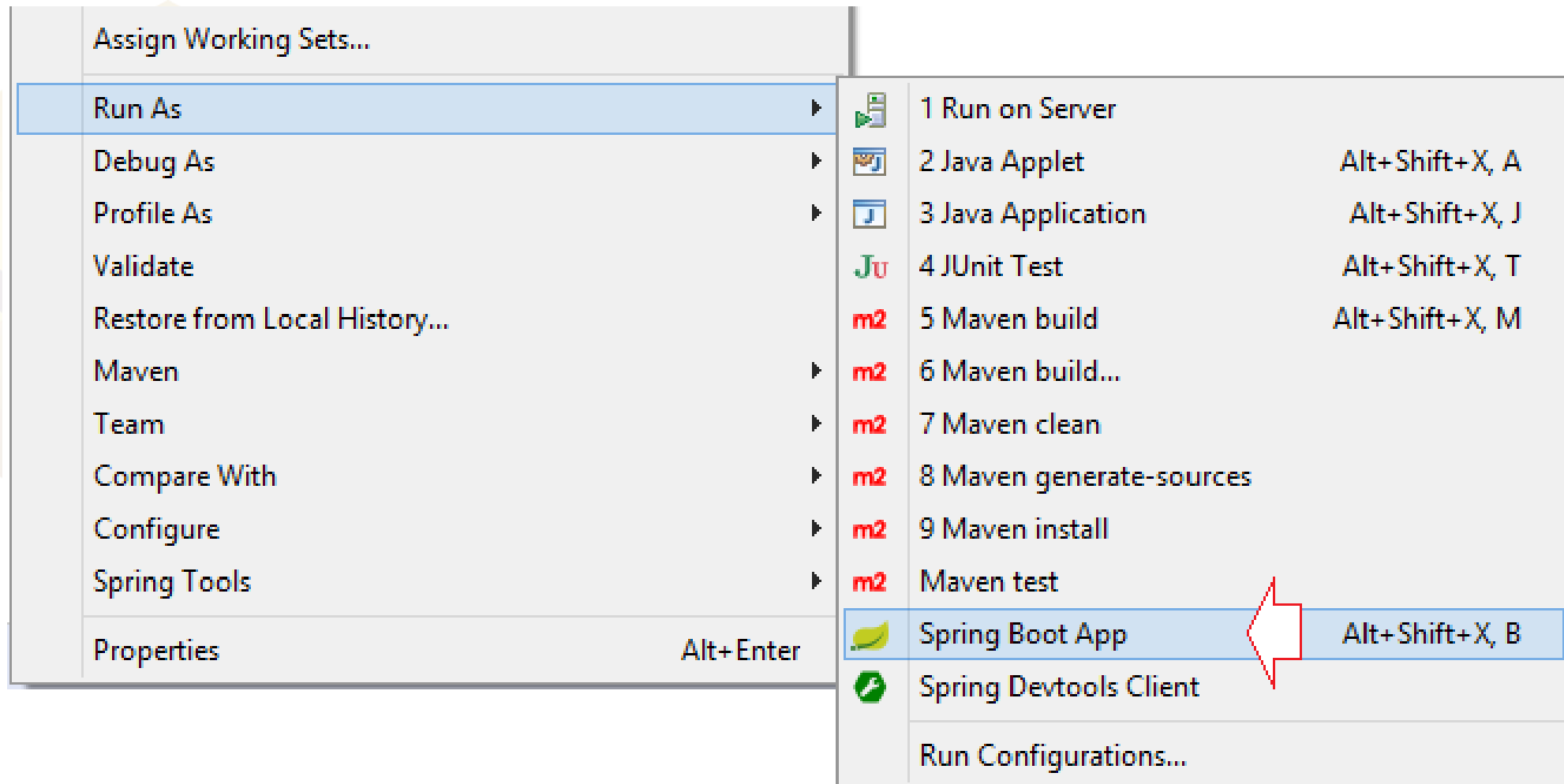
File **html** phải khai báo
xmlns:th=<http://www.thymeleaf.org>
để sử dụng thymeleaf nếu không khi
chạy lên chương trình sẽ báo lỗi.

index.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>home page</title>
</head>
<body>
    <h1>HOME PAGE</h1>
</body>
</html>
```

Chạy Spring Boot Project

❑ Click chuột phải vào **project** → Chọn **Run As** → Chọn **Spring Boot App**.



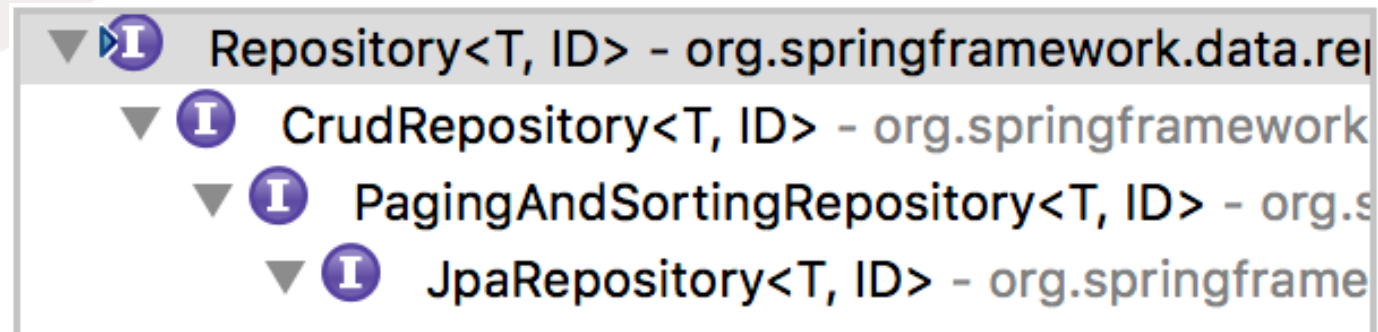
Spring Data JPA

- ❑ **Spring Data JPA** là một module nhỏ trong **Spring Data** project.
- ❑ **Spring Data JPA** được tạo ra với mục đích giảm thiểu các đoạn code lặp đi lặp lại nhiều lần khi tương tác với cơ sở dữ liệu.
- ❑ **Spring Data** định nghĩa một interface chính có tên là **Repository** nằm trong module **Spring Data Common**.
- ❖ Interface này sử dụng 2 generic type:
 - ✓ **T** là entity class mà repository sẽ quản lý.
 - ✓ **ID** là kiểu dữ liệu của ID của entity class mà repository quản lý.

```
package org.springframework.data.repository;  
  
import org.springframework.stereotype.Indexed;  
  
* Central repository marker interface. Captures  
@Indexed  
public interface Repository<T, ID> {  
  
}
```

Spring Data JPA

- ❑ Spring cung cấp một số **interface** hỗ trợ làm việc với CSDL:
- ✓ **CrudRepository** extend từ interface **Repository** hỗ trợ create, read, update, delete.
- ✓ **PagingAndSortingRepository** extend từ **CrudRepository** hỗ trợ phân trang và sắp xếp.
- ✓ **JpaRepository** extend từ **PagingAndSortingRepository** hỗ trợ crud, phân trang và sắp xếp.



JpaRepository interface

```
@NoRepositoryBean
public interface JpaRepository<T, ID>
    extends PagingAndSortingRepository<T, ID>, QueryByExampleExecutor<T> {

    List<T> findAll();

    List<T> findAll(Sort sort);

    List<T> findAllById(Iterable<ID> ids);

    <S extends T> List<S> saveAll(Iterable<S> entities);

    void flush();

    <S extends T> S saveAndFlush(S entity);

    void deleteInBatch(Iterable<T> entities);

    void deleteAllInBatch();

    T getOne(ID id);

    @Override
    <S extends T> List<S> findAll(Example<S> example);

    @Override
    <S extends T> List<S> findAll(Example<S> example, Sort sort);
}
```

Kế thừa JpaRepository

- ❑ Khi muốn sử dụng các phương thức đã **có sẵn** trong interface **JpaRepository** lập trình viên chỉ cần định nghĩa một interface bất kỳ và extend interface **JpaRepository**.

❖ Ví dụ:

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.myclass.entity.Product;

@Repository
public interface ProductRepository extends JpaRepository<Product, Integer>{

}
```

❑ Thư viện

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
```

❑ Khai báo application.properties

```
#----- DATABASE -----
spring.datasource.url=jdbc:mysql://localhost:3306/spring_boot_ticket
spring.datasource.username=root
spring.datasource.password=123456
spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
```


Spring Data Query Creation

- ❑ **Query Creation** là một cơ chế cho phép tạo ra các câu query theo tên **method** trong **Spring Data JPA**.
- ❑ Chỉ cần đặt tên đúng chuẩn, **spring** sẽ tự động sinh ra các câu truy vấn tương ứng.
- ❖ Quy tắc đặt tên:
 - ✓ Tên method phải bắt đầu bằng các từ như **findBy...**, **readBy...**, **queryBy...**, **countBy...** và **getBy...** (Dấu ... là tên thuộc tính entity mà các bạn muốn tìm).
 - ✓ Nếu muốn giới hạn số record trả về thì có thể thêm **First** hoặc **Top** kèm theo **số record** trả về vào trước từ **By** (Nếu ko truyền số record thì mặc định chỉ có 1 record được trả về).

Ví dụ Query Creation

❖ Lấy một **Student** theo thuộc tính **Name**.

```
public Student findFirstByName(String name);
```

❖ Đếm số **Student** theo thuộc tính **Age**.

```
public Integer countByAge(int age);
```

❖ Lấy danh sách **Student** theo thuộc tính **Name** và **Email**.

```
public List<Student> findByNameAndEmail(String name, String email);
```

❑ Chúng ta còn có thể order kết quả trả về bằng cách thêm từ **OrderBy...Asc** (tăng dần) hoặc **OrderBy...Desc** (giảm dần) vào cuối tên phương thức.

```
public List<Student> findByNameOrderByIdDesc(String name);
```

```
public List<Student> findByAgeOrderByNameAsc(int age);
```

Đặt tên phương thức

TỪ KHÓA	VÍ DỤ	JPQL SNIPPET
And	findByLastnameAndFirstname	... where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	... where x.lastname = ?1 or x.firstname = ?2
Between	findByStartDateBetween	... where x.startDate between 1? and ?2
LessThan	findByAgeLessThan	... where x.age < ?1
GreaterThan	findByAgeGreaterThan	... where x.age > ?1
After	findByStartDateAfter	... where x.startDate > ?1
Before	findByStartDateBefore	... where x.startDate < ?1
IsNull	findByAgeIsNull	... where x.age is null
IsNotNull,NotNull	findByAge(Is)NotNull	... where x.age not null

Đặt tên phương thức

TỪ KHÓA	VÍ DỤ	JPQL SNIPPET
Like	findByFirstnameLike	... where x.firstname like ?1
NotLike	findByFirstnameNotLike	... where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	... where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	... where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstnameContaining	... where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	... where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	... where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	... where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> age)	... where x.age not in ?1
True	findByActiveTrue()	... where x.active = true
False	findByActiveFalse()	... where x.active = false

Spring data @Query

- ❑ Một số hạn chế gặp phải của các phương thức Spring Data:
 - ✓ Các phương thức có sẵn khi kế thừa các interface **JpaRepository** và **CrudRepository** không đáp ứng được yêu cầu.
 - ✓ Đặt tên theo chuẩn **Query Creation** quá dài hoặc tối nghĩa (Ví dụ bạn muốn truy vấn theo 5 điều kiện thì tên method của bạn sẽ gồm 5 điều kiện đó => quá dài).
- **Spring JPA** cung cấp annotation **@Query** để khai báo các query cho các method trong interface kế thừa từ Repository.

```
@Repository
public interface CustomerRepository extends JpaRepository<Customer, Integer> {

    @Query("SELECT e FROM Customer e ORDER BY e.name DESC")
    List<Customer> findAllOrderByNameDesc();
}
```

Truyền tham số

❑ Truyền theo thứ tự

```
@Repository
public interface CustomerRepository extends JpaRepository<Customer, Integer> {

    @Query("SELECT e FROM Customer e WHERE e.name = ?1 AND e.address = ?2")
    List<Customer> findByNameAndAddress(String name, String address);

}
```

❑ Sử dụng @Param

```
@Repository
public interface CustomerRepository extends JpaRepository<Customer, Integer> {

    @Query("SELECT e FROM Customer e WHERE e.name = :name AND e.address = :address")
    List<Customer> findByNameAndAddress(@Param("name") String name, @Param("address") String address);

}
```


Kết quả trả về

❑ Trả về một List

```
@Repository
public interface CustomerRepository extends JpaRepository<Customer, Integer> {
    @Query("SELECT e FROM Customer e WHERE e.name = :name")
    List<Customer> findByName(@Param("name") String name);
}
```

❑ Trả về một Stream

- ❖ Trường hợp kết quả của câu query có quá nhiều bản ghi, thay vì phải tạo rất nhiều đối tượng để lưu các bản ghi đó ta sẽ tạo ra 1 stream, và nó chỉ thực sự tạo ra các đối tượng khi bạn sử dụng đến phần tử bên trong stream.

```
@Repository
public interface CustomerRepository extends JpaRepository<Customer, Integer> {
    @Query("SELECT e FROM Customer e WHERE e.name = :name")
    Stream<Customer> findByName(@Param("name") String name);
}
```

Kết quả trả về

❑ Trả về một đối tượng

- ❖ Sử dụng khi chắc chắn kết quả của câu query sẽ trả về 1 đối tượng, nếu câu query trả về nhiều hơn 1 đối tượng thì sẽ xảy ra lỗi.

```
@Repository
public interface CustomerRepository extends JpaRepository<Customer, Integer> {
    @Query("SELECT e FROM Customer e WHERE e.name = :name")
    Customer findByName(@Param("name") String name);
}
```

❑ Trả về một Page

- ❖ Page áp dụng cho trường hợp muốn phân trang

```
@Repository
public interface CustomerRepository extends JpaRepository<Customer, Integer> {
    @Query("SELECT e FROM Customer e WHERE e.name = :name")
    Page<Customer> findByName(@Param("name") String name, Pageable pageable);
}
```


Ví dụ

```
@Repository
public interface CustomerRepository extends JpaRepository<Customer, Integer> {

    @Query("SELECT e FROM Customer e ORDER BY e.name DESC")
    List<Customer> findAllOrderByNameDesc();

    @Query(value = "SELECT e.* FROM customer e ORDER BY e.name DESC", nativeQuery = true)
    List<Customer> findAllOrderByNameDescNative();

    @Query("SELECT e FROM Customer e WHERE e.name = ?1")
    List<Customer> findByName(String name);

    @Query("SELECT e FROM Customer e WHERE e.name = :name AND e.address = :address")
    List<Customer> findByNameAndAddress(@Param("name") String name, @Param("address") String address);

    @Query("SELECT e FROM Customer e WHERE e.name like ?1")
    List<Customer> findByNameLike(String name);

}
```

Phân trang

- ❑ Để phân trang trong spring data jpa, trong method truy vấn, thay vì trả về đối tượng Stream hay một List thì ta **trả về đối tượng Page**.

```
@Query("SELECT e FROM cinemas e")  
public Page<Cinema> findAllPaging(Pageable pageable);
```

- ❖ Trong đó:
 - ✓ **Pageable** sẽ chứa các thông tin phân trang như **số phần tử được lấy, vị trí trang được lấy**.
 - ✓ **Page** sẽ chứa kết quả trả về gồm **số phần tử, danh sách các phần tử,...**

Phân trang

❑ Để **cung cấp thông tin phân trang** chúng ta phải tạo ra một biến có kiểu **Pageable** và sử dụng **PageRequest** để khởi tạo nó.

❖ Ví dụ lấy danh sách với thông tin là trang thứ nhất và lấy 10 phần tử, ta làm như sau:

```
Pageable pageable = new PageRequest(1, 10);
```

❑ Ngoài ra bạn cũng có thể **sắp xếp các phần tử trong Page** bằng cách thêm tham số **Sort** vào trong **Pageable**:

```
Sort sort = new Sort("name");  
Pageable pageable = new PageRequest(1, 10, sort);
```

Ví dụ phân trang

❑ Lấy thông tin phân trang của rạp chiếu phim.

❖ Repository

```
@Query("SELECT e FROM cinemas e")  
public Page<Cinema> findAllPaging(Pageable pageable);
```

❖ Service

```
@Override  
public Page<Cinema> findAllPaging(int pageIndex, int pageSize) {  
    Pageable pageable = new PageRequest(pageIndex, pageSize);  
    return cinemaRepository.findAllPaging(pageable);  
}
```

Ví dụ phân trang

❖ Controller

```
@GetMapping("paging/{pageIndex}/{pageSize}")
@ResponseBody
public ResponseEntity<Page<Cinema>> paging(
    @PathVariable int pageIndex,
    @PathVariable int pageSize) {
    Page<Cinema> cinemas = cinemaService.findAllPaging(pageIndex - 1, pageSize);
    if(cinemas.getSize() == 0) {
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    }
    return new ResponseEntity<Page<Cinema>>(cinemas, HttpStatus.OK);
}
```

❖ Trong đó:

- ✓ **pageIndex**: Vị trí trang được lấy.
- ✓ **pageSize**: Số phần tử được lấy.

Truy vấn Join

❑ Lấy dữ liệu từ nhiều bảng bằng câu lệnh JOIN

✓ Bước 1: Tạo lớp DTO

✓ Bước 2: Truy vấn dữ liệu dùng **Data @Query**

Lớp DTO

```
public UserDto(String email, String fullname, String avatar, int roleId, String roleName) {  
    super();  
    this.email = email;  
    this.fullname = fullname;  
    this.avatar = avatar;  
    this.roleId = roleId;  
    this.roleName = roleName;  
}
```

Truy vấn

```
@Repository  
public interface UserRepository extends JpaRepository<User, String>{  
  
    @Query("SELECT new UserDto ( u.email, u.fullname, u.avatar, u.roleId, r.name ) FROM User u JOIN u.Role r")  
    List<UserDto> findAllUserRole();  
}
```

Tổng kết

- ☐ Tìm hiểu tổng quan về Spring Boot.
- ☐ Biết cách tạo dự án bằng Spring Boot.
- ☐ Spring JPA là gì?
- ☐ Sử dụng Spring Data Query Creation.
- ☐ Sử dụng Spring Data @Query.

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH