



BUỔI 3 - DATABIDING

ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH
CYBERSOFT.EDU.VN



❑ Layout

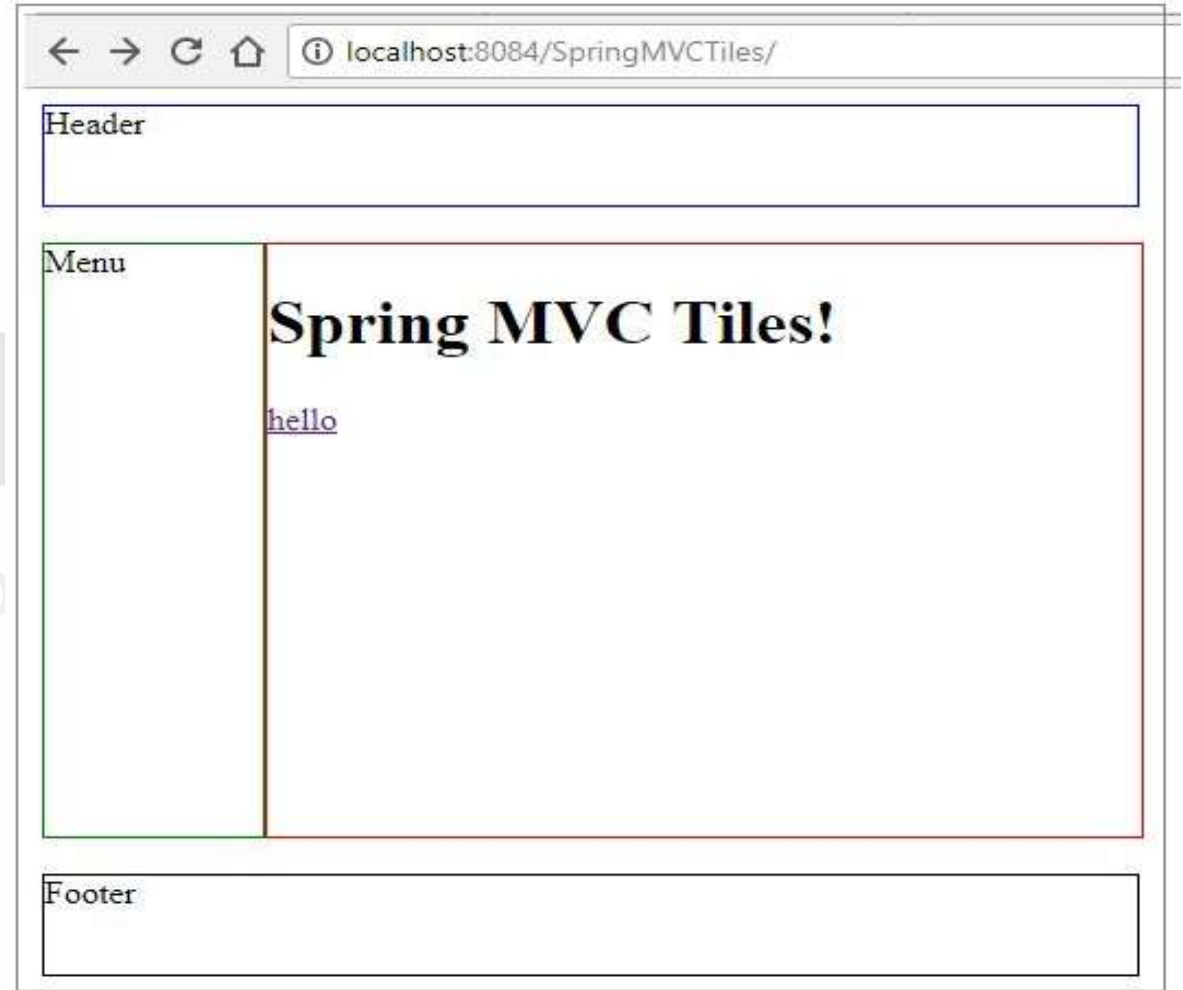
- Sử dụng Apache Tiles
- Cấu hình static resource.
- **Data Binding**
- Data Binding là gì?
- Cách sử dụng.
- Bài tập trên lớp.
- Áp dụng vào dự án.
- Bài về nhà.

❑ Validation

- Validation là gì?
- Validate bằng tay.
- Validate bằng Annotation.
- Hiển thị lỗi validate.
- Custom validation.
- Bài tập trên lớp.
- Áp dụng vào dự án.
- Bài về nhà.

Apache Tiles

- ❑ **Apache Tiles** là là một **framework/engine** thực hiện việc tạo template.
- ❑ **Apache Tiles** là một mã nguồn mở phát triển bởi apache, được xây dựng để **đơn giản hóa việc phát triển các ứng dụng web**, đặc biệt là với mô hình MVC.
- ❑ **Apache Tiles** sẽ giúp tách nhỏ các phần của layout và sử dụng lại chúng.



Cấu hình Apache Tiles



❑ Các bước cấu hình

- ✓ **Bước 1:** Tải thư viện **tiles-extras** về cho dự án.
- ✓ **Bước 2:** Tách các file layout cho dự án (header, footer, navbar,...).
- ✓ **Bước 3:** Tạo file **tiles.xml** nằm trong thư mục WEB-INF (file này giúp ghép nối các file đã tách lại với nhau).
- ✓ **Bước 4:** Tạo file **default.jsp** làm layout chính, khai báo **taglib** của Apache Tiles vào và nhúng lần lượt các file đã tách vào layout chính.
- ✓ **Bước 5:** Cấu hình Apache Tiles cho dự án (xml hoặc java config).

Cấu hình Apache Tiles

Cấu trúc thư mục



Thư viện

```
<dependency>
  <groupId>org.apache.tiles</groupId>
  <artifactId>tiles-extras</artifactId>
  <version>3.0.8</version>
</dependency>
```

Cấu hình Apache Tiles

Định nghĩa layout base dùng chung cho các trang

Khai báo các file static dùng riêng cho từng trang.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tiles-definitions PUBLIC "-//Apache Software Foundation//DTD Tiles Configuration 3.0//EN"
"http://tiles.apache.org/dtds/tiles-config_3_0.dtd">

<tiles-definitions>
  <!-- ***** USER LAYOUT ***** -->
  <definition name="base-user"
    template="/WEB-INF/views/layout/default.jsp">
    <put-attribute name="title" value="" />
    <put-attribute name="header" value="/WEB-INF/views/layout/header.jsp" />
    <put-attribute name="navbar" value="/WEB-INF/views/layout/navbar.jsp" />
    <put-attribute name="body" value="" />
    <put-attribute name="footer" value="/WEB-INF/views/layout/footer.jsp" />
    <put-list-attribute name="styles" />
  </definition>

  <!--===== ADMIN USER =====-->

  <!-- HOME PAGE -->
  <definition name="home" extends="base-user">
    <put-attribute name="title" value="Trang chủ" />
    <put-attribute name="body" value="/WEB-INF/views/home/index.jsp" />
    <put-list-attribute name="styles" inherit="true">
      <add-attribute value="/statics/css/home.css"/>
    </put-list-attribute>
  </definition>

  <!-- USER PROFILE PAGE -->
  <definition name="userProfile" extends="base-user">
    <put-attribute name="title" value="Thông tin cá nhân" />
    <put-attribute name="body" value="/WEB-INF/views/user/profile.jsp" />
  </definition>
</tiles-definitions>
```

Các trang con kế thừa từ layout base

Các trang con kế thừa từ layout base

tiles.xml

FT
RÌNH

Cấu hình Apache Tiles

Khai báo taglib của Apache Tiles

Dùng vòng lặp để in ra đường dẫn tới các file.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles" %>

<tiles:importAttribute name="styles"/>

<!doctype html>
<html lang="en">

<head>
    <title><tiles:getAsString name="title"/></title>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <tiles:insertAttribute name="header" />
    <c:forEach items="${styles}" var="item">
        <link rel="stylesheet" href='<c:url value="${ item }"/>'>
    </c:forEach>
</head>
<body>
    <tiles:insertAttribute name="navbar"/>

    <tiles:insertAttribute name="body"/>

    <tiles:insertAttribute name="footer" />
</body>
```

Import thẻ chứa các file static dùng riêng cho từng trang.

Insert thẻ chứa đường dẫn tới các file layout.

default.jsp

DFT
P TRÌNH

Cấu hình Apache Tiles

WebMvcConfig

```
//public void configureViewResolvers(ViewResolverRegistry registry) {  
//    registry.jsp("/WEB-INF/views/", ".jsp");  
//}  
  
@Bean  
public TilesConfigurer tilesConfigurer(){  
    TilesConfigurer tilesConfigurer = new TilesConfigurer();  
    tilesConfigurer.setDefinitions(new String[] {"/WEB-INF/tiles.xml"});  
    tilesConfigurer.setCheckRefresh(true);  
    return tilesConfigurer;  
}  
  
public void configureViewResolvers(ViewResolverRegistry registry) {  
    TilesViewResolver viewResolver = new TilesViewResolver();  
    registry.viewResolver(viewResolver);  
}
```

Tạo bean
tilesConfigurer

Cấu hình lại
ViewResolver

Static Resource

- ❑ Trong Spring khi bạn **phân cấp các URI** thì việc link tới các file resource (css, js, image...) cũng sẽ bị thay đổi.
- ❑ Cần **cố định các file resource bằng 1 đường dẫn cố định** mà tất cả các view đều có thể truy cập qua đường dẫn đó (**Static Resource**).

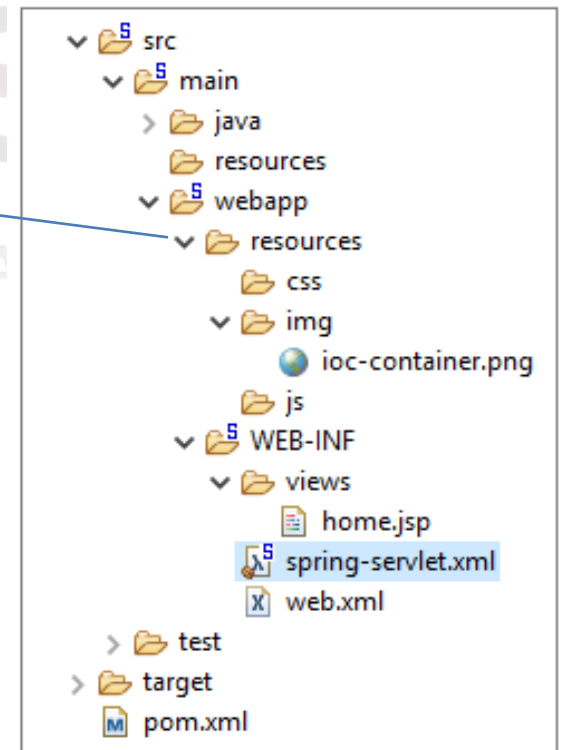
```
<mvc:resources mapping="/statics/**" location="/resources/" />
```

location

Cấu hình resource

mapping

```
<body>
  <img src='<c:url value="statics/img/ioc-container.png" />' />
  <h1>HOME PAGE</h1>
</body>
```



Static Resource Java Config

Cấu hình trong WebMvcConfig

```
public void addResourceHandlers(ResourceHandlerRegistry registry) {  
    registry  
        .addResourceHandler("/statics/**")  
        .addResourceLocations("/resources/");  
}
```

Sử dụng EL binding dữ liệu

- Bạn có thể ràng buộc dữ liệu từ các thuộc tính của bean vào các điều khiển HTML bằng cách sử dụng biểu thức EL.

```
<form action="login.htm" method="post">
  <div>User Name:</div>
  <input name="id" value="${user.id}">

  <div>Password:</div>
  <input name="password" value="${user.password}">

  <hr>
  <button>Login</button>
</form>
```

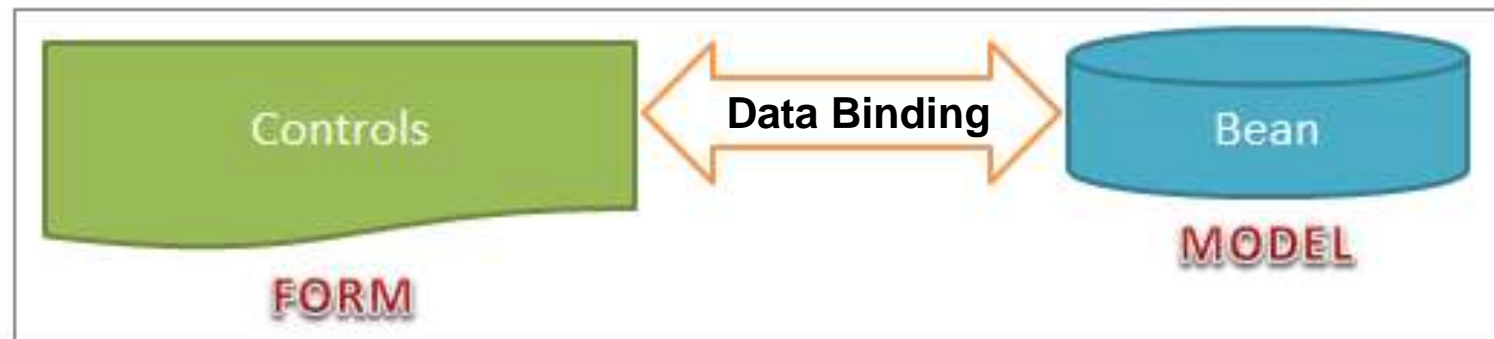
❖ Hạn chế:

- ✓ Phải viết mã trên giao diện, dài dòng, khó quản lý.
- ✓ Đổ dữ liệu vào các List Control trở nên phức tạp và khó khăn
 - ✓ Combox
 - ✓ Listbox
 - ✓ Radiobuttons
 - ✓ Checkboxes
- ✓ Kiểm và thông báo lỗi khó khăn.
- ✓ Phải sử dụng các biến variable local khi set get dữ liệu.

Data Binding

❑ Data binding là gì?

- Data binding (ràng buộc dữ liệu) là kỹ thuật dùng để tạo gắn kết dữ liệu giữa tầng View (giao diện) và dữ liệu trong các bean ở tầng Controller.
- Khi **thay đổi dữ liệu trong bean** thì **dữ liệu trên các điều khiển** cũng thay đổi theo.
- Ràng buộc dữ liệu có thể là **1 chiều** hoặc **2 chiều**:
 - ✓ **Chiều lên**: **chuyển dữ liệu** từ các điều khiển vào các thuộc tính của bean.
 - ✓ **Chiều về**: **hiển thị dữ liệu** từ các thuộc tính của bean lên các điều khiển của form.



Data Binding

- ✓ **Spring MVC** cung cấp **thư viện thẻ** giúp việc buộc dữ liệu từ **bean** vào **các điều khiển** trở nên **dễ dàng** hơn.

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
```

- ✓ Sau khi khai báo thư viện thẻ ngay đầu trang JSP, chúng ta có thể tạo form và ràng buộc dữ liệu.

```
<form:form action="login.htm" modelAttribute="user">
  <div>User Name:</div>
  <form:input path="id"/>
  <div>Password:</div>
  <form:input path="password"/>
  <hr>
  <button>Login</button>
</form:form>
```

Thẻ trong
thư viện
form

Tên của bean
đặt trong
model

Tên thuộc
tính của
bean

❑ Ưu điểm của Form Spring

- Cung cấp cơ chế **buộc dữ liệu** lên các điều khiển.
- Form **đơn giản, rõ ràng, dễ hiểu**.
- Khi thay đổi **dữ liệu trong bean** thì **dữ liệu trên các điều khiển** cũng thay đổi theo.
- Cấp dữ liệu vào các **List Control** trở nên rất đơn giản.
- Giúp **kiểm tra** và **hiển thị lỗi** một cách dễ dàng.

Data Binding

```
@Controller
@RequestMapping("student")
public class StudentController {

    @GetMapping("add")
    public String add(Model model) {
        model.addAttribute("student", new Student());
        return "student/add";
    }

    @PostMapping("add")
    public String add(Model model,
        @ModelAttribute("student") Student student) {

        model.addAttribute("student", student);
        return "student/add";
    }
}
```

```
public class Student {
    private int id;
    private String name;
    private int age;

    public Student() {}
    public Student(int id, String name, int age) {}

    public int getId() {}
    public void setId(int id) {}
    public String getName() {}
    public void setName(String name) {}
    public int getAge() {}
    public void setAge(int age) {}
}
```

```
<h3 class="text-center">CREATE FORM</h3>
<c:url value="/student/add" var="action"/>
<Mvc:form action="{action}" method="post"
    modelAttribute="student">
    <div class="form-group">
        <label>Id:</label>
        <Mvc:input type="number" path="id" cssClass="form-control" />
        <Mvc:errors path="id" cssClass="text-danger"/>
    </div>
    <div class="form-group">
        <label>Name:</label>
        <Mvc:input type="text" path="name" cssClass="form-control" />
        <Mvc:errors path="name" cssClass="text-danger"/>
    </div>
    <div class="form-group">
        <label>Age:</label>
        <Mvc:input type="text" path="age" cssClass="form-control" />
        <Mvc:errors path="age" cssClass="text-danger"/>
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
</Mvc:form>
```


Data Binding

❑ Các điều khiển FORM của Spring MVC.



Điều khiển Spring	Sinh ra điều khiển HTML
<code><form:form></code>	<code><form></code>
<code><form:input/></code>	<code><input type="text"/></code>
<code><form:textarea/></code>	<code><textarea/></code>
<code><form:checkbox/></code>	<code><input type='checkbox'/></code>
<code><form:radiobutton/></code>	<code><input type='radio'/></code>
<code><form:hidden/></code>	<code><input type='hidden'/></code>
<code><form:password/></code>	<code><input type='password'/></code>
<code><form:button/></code>	<code><button/></code>
<code><form:select/></code>	<code><select/></code>
<code><form:radiobuttons/></code>	Nhóm radio
<code><form:checkboxes/></code>	Nhóm checkbox

FT
TRÌNH

Form:tag

❑ Thẻ Spring <form:tag> có một số thuộc tính thường dùng sau:

- ✓ **cssClass**: thay cho thuộc tính class trong HTML.
- ✓ **disabled**: thay cho thuộc tính disabled trong HTML
- ✓ **readonly**: thay cho thuộc tính readonly trong HTML
- ✓ **cssErrorClass**: cho ra class định dạng thông báo lỗi.

❖ Ví dụ:

```
<form:input path="id" readonly="true"/>
```

```
<form:input path="name" cssClass="form-control">
```

```
<form:errors path="name" cssErrorClass="text-danger">
```

Data Binding - ComboBox

Họ và tên

Nguyễn Văn Tèo

Điểm

9.5

Chuyên ngành

Ứng dụng phần mềm

Update

❑ Để đổ dữ liệu vào ComboBox thì chúng ta cần:

- **Controller:** Phải cung cấp dữ liệu dạng Array, Collection hoặc Map vào model.
- **View:** Sử dụng thẻ select cung cấp bởi thư viện thẻ của Spring MVC.

Đổi từ nhập dữ liệu sang chọn mục trong ComboBox

Data Binding - ComboBox

❑ String[]

```
@ModelAttribute("majors")
public String[] getMajors() {
    String[] majors = {
        "Ứng dụng phần mềm",
        "Thiết kế trang web"
    };
    return majors;
}
```



```
<div>Chuyên ngành</div>
<form:select path="major" items="${majors}"/>
```



```
<select id="major" name="major">
    <option value="Ứng dụng phần mềm">Ứng dụng phần mềm</option>
    <option value="Thiết kế trang web">Thiết kế trang web</option>
</select>
```



Ứng dụng phần mềm ▼
Ứng dụng phần mềm
Thiết kế trang web

Data Binding - ComboBox

❑ Map<String, String>

```
@ModelAttribute("majors")  
public Map<String, String> getMajors() {  
    Map<String, String> majors = new HashMap<>();  
    majors.put("APP", "Ứng dụng phần mềm");  
    majors.put("WEB", "Thiết kế trang web");  
    return majors;  
}
```

> `<div>Chuyên ngành</div>`
`<form:select path="major" items="${majors}" />`

Ứng dụng phần mềm ▾
Ứng dụng phần mềm
Thiết kế trang web

<select id="major" name="major">
 <option value="APP">Ứng dụng phần mềm</option>
 <option value="WEB">Thiết kế trang web</option>
</select>

Data Binding - ComboBox

❑ List<Object>

```
public class Major {  
    private String id;  
    private String name;  
  
    public Major() {}  
    public Major(String id, String name) {}  
  
    public String getName() {}  
    public void setName(String name) {}  
    public String getId() {}  
    public void setId(String id) {}  
}
```



```
@ModelAttribute("majors")  
public List<Major> getMajors() {  
    List<Major> majors = new ArrayList<>();  
    majors.add(new Major("APP", "Ứng dụng phần mềm"));  
    majors.add(new Major("WEB", "Thiết kế trang web"));  
    return majors;  
}
```



```
<div>Chuyên ngành</div>  
<form:select path="major" items="${majors}"  
             itemLabel="name" itemValue="id"/>
```



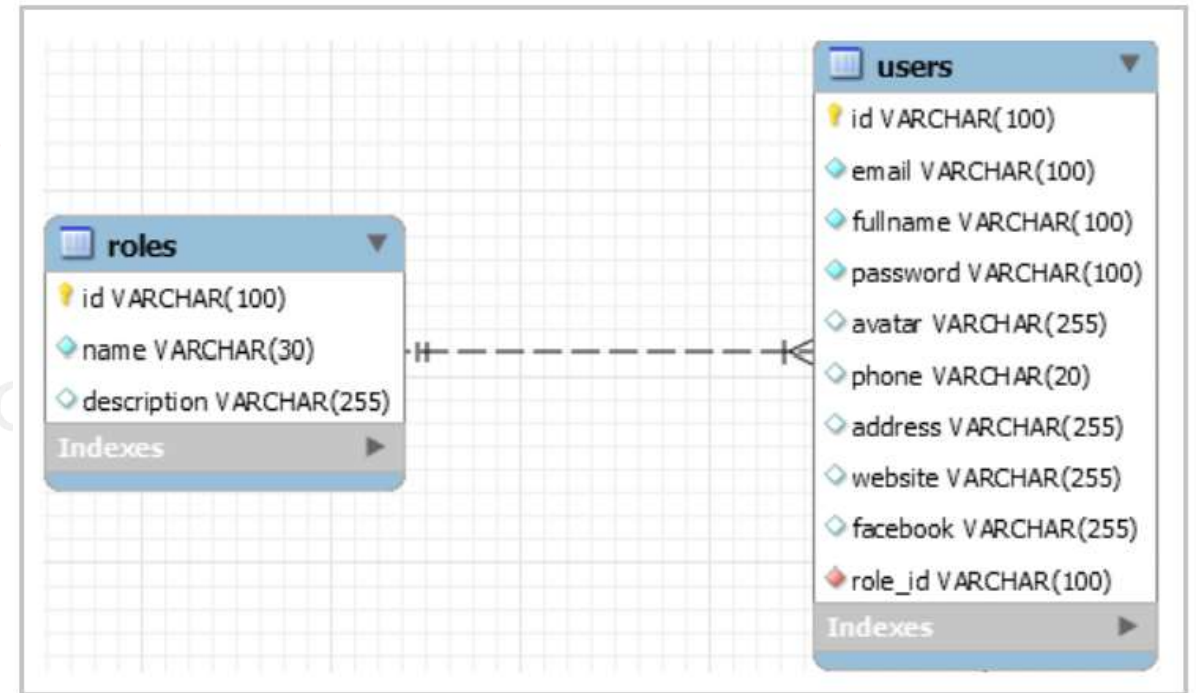
```
<select id="major" name="major">  
    <option value="APP">Ứng dụng phần mềm</option>  
    <option value="WEB">Thiết kế trang web</option>  
</select>
```



Ứng dụng phần mềm ▼
Ứng dụng phần mềm
Thiết kế trang web

Bài tập 1

- ❑ Xây dựng module quản lý người dùng (CRUD) gồm 2 POJO bean có các thuộc tính như hình bên.
- ✓ Tổ chức dự án theo mô hình MVC.
- ✓ Xây dựng các tầng Repository, Service, Controller, View.
- ✓ Sử dụng thư viện thẻ của Spring để thêm mới, cập nhật thông tin.

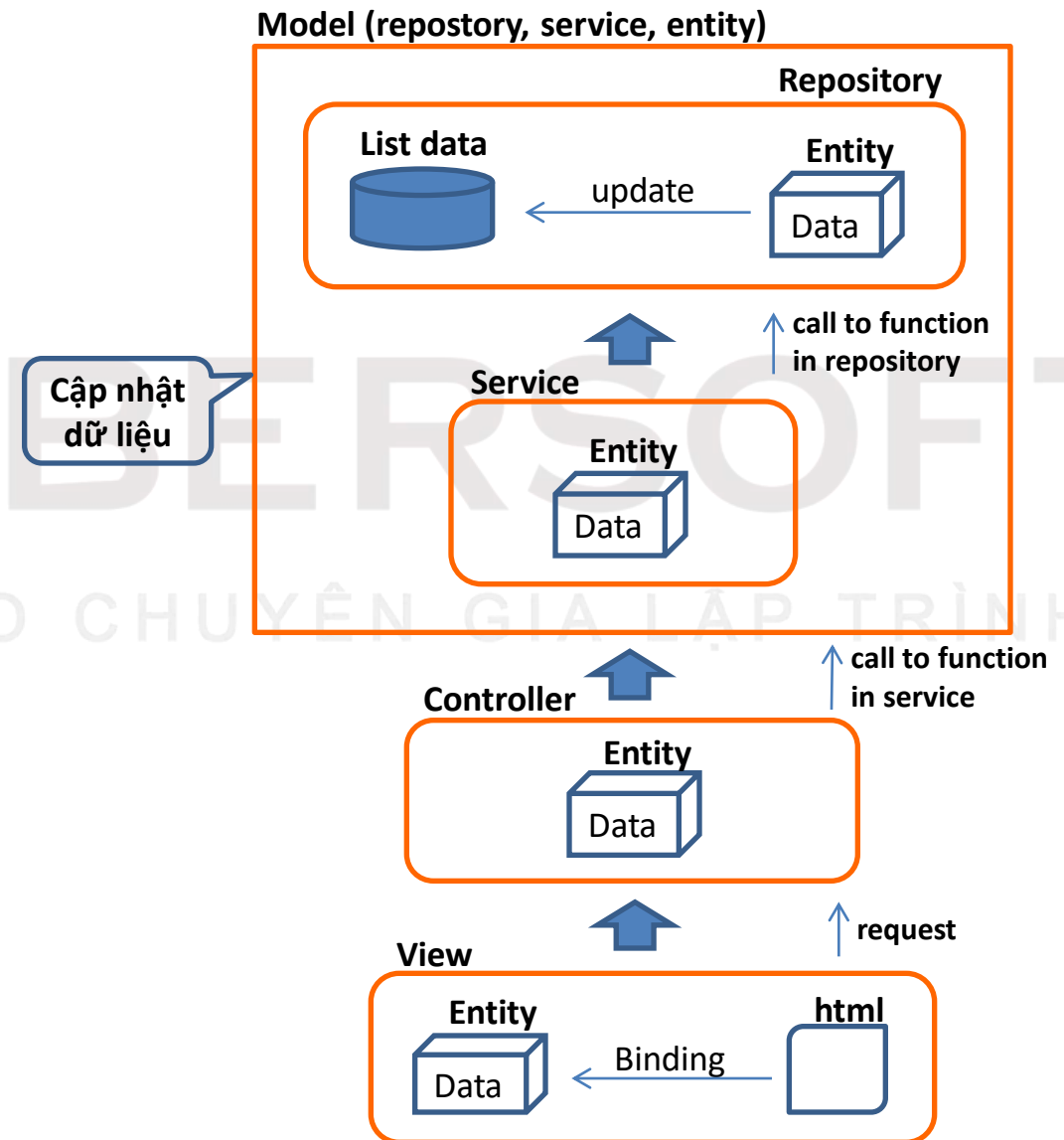
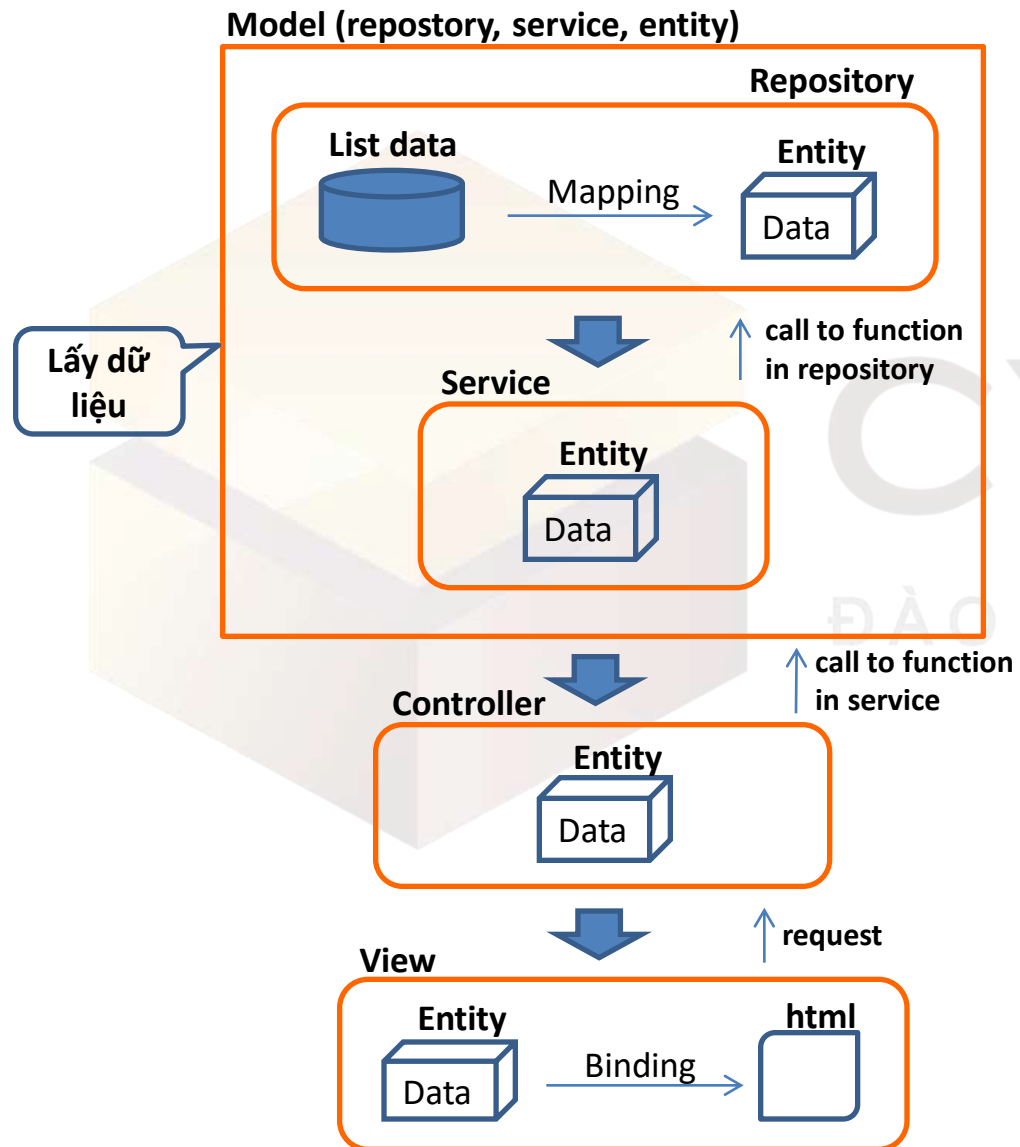


Các bước thực hiện



- ✓ **Bước 1:** Xây dựng cấu trúc thư mục theo mô hình MVC.
- ✓ **Bước 2:** Tạo POJO bean (Role, User).
- ✓ **Bước 3:** Xây dựng tầng Repository (UserRepository, RoleRepository).
 - ✓ UserRepository: Tạo danh sách chứa các User
 - ✓ RoleRepository: Tạo danh sách chứa các Role.
 - ✓ Định nghĩa các phương thức thêm xóa sửa.
- ✓ **Bước 4:** Xây dựng tầng Service().
 - ✓ Định nghĩa các phương thức thêm, xóa, sửa.
- ✓ **Bước 5:** Xây dựng tầng Controller.
- ✓ **Bước 6:** Tạo view.
 - ✓ Sử dụng thư viện thẻ của Spring MVC.

Truyền data giữa các tầng



Cấu trúc thư mục

- ▼ Java Resources
 - ▼ src/main/java
 - > com.myclass.admin.controller
 - > com.myclass.configuration
 - com.myclass.controller
 - > com.myclass.entity
 - ▼ com.myclass.repository
 - > RoleRepository.java
 - > UserRepository.java
 - ▼ com.myclass.repository.impl
 - > RoleRepositoryImpl.java
 - > UserRepositoryImpl.java
 - ▼ com.myclass.service
 - > RoleService.java
 - > UserService.java
 - ▼ com.myclass.service.impl
 - > RoleServiceImpl.java
 - > UserServiceImpl.java
 - > src/main/resources

Role interface

```
public interface RoleRepository {  
    public List<Role> findAll();  
    public Role findById(String id);  
    public boolean save(Role model);  
    public boolean delete(String id);  
}
```

User interface

```
public interface UserRepository {  
    public List<User> findAll();  
    public User findById(String id);  
    public boolean save(User model);  
    public boolean delete(String id);  
}
```

UserRepository

Đánh dấu đây là
bean thuộc tầng
Repository

Đánh dấu phương
thức init của bean,
chạy ngay sau
phương thức khởi tạo

```
@Repository
public class UserRepositoryImpl implements UserRepository{

    private List<User> users;

    @PostConstruct
    public void init() {
        users = new ArrayList<User>();
    }

    public List<User> findAll() {
        return users;
    }

    public User findById(String id) {
        for(User user: users) {
            if(user.getId().equals(id))
                return user;
        }
        return null;
    }

    public boolean save(User model) {
        return users.add(model);
    }

    public boolean delete(String id) {
        User user = this.findById(id);
        return users.remove(user);
    }
}
```

RoleRepository

Đánh dấu đây là
bean thuộc tầng
Repository

Đánh dấu phương
thức init của bean,
chạy ngay sau
phương thức khởi tạo

```
@Repository
public class RoleRepositoryImpl implements RoleRepository{

    private List<Role> roles;

    @PostConstruct
    public void init() {
        roles = new ArrayList<Role>();
        Role role = new Role();
        role.setId(UUID.randomUUID().toString());
        role.setName("ROLE_ADMIN");
        role.setDescription("Quản trị hệ thống");
        roles.add(role);

        role = new Role();
        role.setId(UUID.randomUUID().toString());
        role.setName("ROLE_USER");
        role.setDescription("Thành viên");
        roles.add(role);
    }

    public List<Role> findAll() {
        return roles;
    }

    public Role findById(String id) {}

    public boolean save(Role model) {}

    public boolean delete(String id) {}
}
```


RoleService

Đánh dấu đây
là bean thuộc
tầng Service

Random chuỗi
để làm id

```
@Service
public class RoleServiceImpl implements RoleService{

    @Autowired
    private RoleRepository roleRepositoryImpl;

    public List<Role> findAll() {
        return roleRepositoryImpl.findAll();
    }

    public Role findById(String id) {
        return roleRepositoryImpl.findById(id);
    }

    public boolean insert(Role model) {
        model.setId(UUID.randomUUID().toString());
        return roleRepositoryImpl.save(model);
    }

    public boolean update(Role model) {
        return roleRepositoryImpl.save(model);
    }

    public boolean delete(String id) {
        return roleRepositoryImpl.delete(id);
    }
}
```

interface

```
public interface RoleService {
    public List<Role> findAll();
    public Role findById(String id);
    public boolean insert(Role model);
    public boolean update(Role model);
    public boolean delete(String id);
}
```

YÊN GIA LẬP TRÌNH

UserService

```
@Service
public class UserServiceImpl implements UserService{

    @Autowired
    private UserRepository userRepositoryImpl;

    public List<User> findAll() {
        return userRepositoryImpl.findAll();
    }

    public User findById(String id) {
        return userRepositoryImpl.findById(id);
    }

    public boolean insert(User model) {
        model.setId(UUID.randomUUID().toString());
        return userRepositoryImpl.save(model);
    }

    public boolean update(User model) {
        return userRepositoryImpl.save(model);
    }

    public boolean delete(String id) {
        return userRepositoryImpl.delete(id);
    }
}
```

interface

```
public interface UserService {
    public List<User> findAll();
    public User findById(String id);
    public boolean insert(User model);
    public boolean update(User model);
    public boolean delete(String id);
}
```

CHUYÊN GIA LẬP TRÌNH

UserController

Inject
UserService
để sử dụng

Sử dụng đối tượng
Model để chuyển
dữ liệu từ
Controller đến view

```
@Controller
@RequestMapping("admin/user")
public class AdminUserController {
    @Autowired
    private UserService userServiceImpl;

    @Autowired
    private RoleService roleServiceImpl;

    @GetMapping("")
    public String index(Model model) {
        model.addAttribute("users", userServiceImpl.findAll());
        return "userList";
    }

    @GetMapping("add")
    public String add(Model model) {
        model.addAttribute("user", new User());
        model.addAttribute("roles", roleServiceImpl.findAll());
        return "userAdd";
    }

    @PostMapping("add")
    public String postAdd(@ModelAttribute("user") User user) {
        userServiceImpl.insert(user);
        return "redirect:/admin/user";
    }

    @GetMapping("edit/{id}")
    public String edit(@PathVariable String id, Model model) {
        model.addAttribute("user", userServiceImpl.findById(id));
        model.addAttribute("roles", roleServiceImpl.findAll());
        return "userEdit";
    }
}
```

Sử dụng @ModelAttribute
để binding dữ liệu từ view
gửi lên Controller

SOFT
IA LẬP TRÌNH

User Add View

```
<c:url value="/admin/user/add" var="action"/>
<mvc:form action="{ action }" method="post" modelAttribute="user">
  <div class="form-group">
    <label>Họ tên</label>
    <mvc:input type="text" cssClass="form-control" path="fullname" />
  </div>
  <div class="form-group">
    <label>Email</label>
    <mvc:input type="text" cssClass="form-control" path="email" />
  </div>
  <div class="form-group">
    <label>Mật khẩu</label>
    <mvc:input type="password" cssClass="form-control" path="password" />
  </div>
  <div class="form-group">
    <label>Loại tài khoản</label>
    <mvc:select path="roleId" items="{ roles }" cssClass="form-control"
      itemLabel="description" itemValue="id"/>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</mvc:form>
```

Sử dụng
modelAttribute để
binding dữ liệu

User List View

```
<a href='<c:url value="/admin/user/add"/>'
  class="btn btn-sm btn-primary mb-2">Thêm mới</a>
<table class="table table-bordered">
  <thead>
    <tr>
      <th>STT</th>
      <th>Email</th>
      <th>Họ tên</th>
      <th>#</th>
    </tr>
  </thead>
  <tbody>
    <c:forEach items="${users}" var="item" varStatus="loop">
      <tr>
        <td>${ loop.index + 1 }</td>
        <td>${ item.email }</td>
        <td>${ item.fullname }</td>
        <td>
          <a href='<c:url value="/admin/user/edit/${item.id}"/>'
            class="btn btn-sm btn-info">Sửa</a>
          <a href='<c:url value="/admin/user/delete/${item.id}"/>'
            class="btn btn-sm btn-danger">Xóa</a></td>
        </tr>
      </c:forEach>
    </tbody>
  </table>
```

FT
RINH

❑ Validation

- **Validation** kiểm tra dữ liệu đầu vào.
- Dữ liệu vào không hợp lệ sẽ **gây các lỗi khó lường**. Vì vậy việc kiểm soát dữ liệu vào luôn đóng vai trò quan trọng của ứng dụng.
- Trong **Spring MVC** có hai cách validate:
 - ✓ Validation bằng tay.
 - ✓ Validation bằng Annotation.

❑ Các lỗi thường gặp

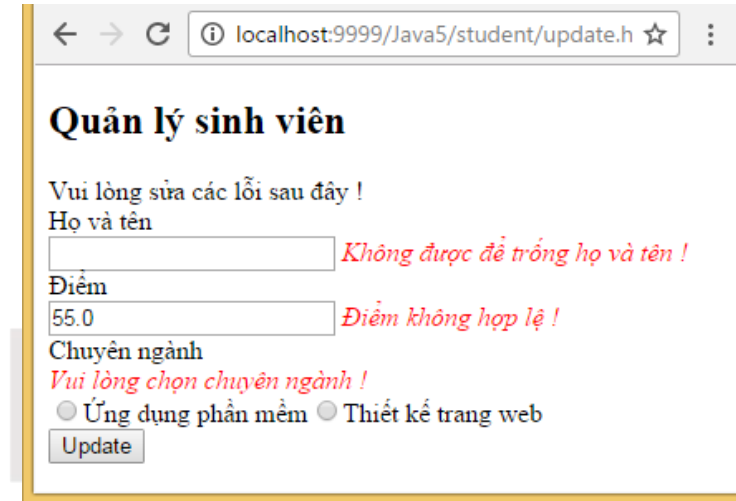
- Để trống ô nhập...
- Không đúng định dạng email, creditcard, url...
- Sai kiểu số nguyên, số thực, ngày giờ...
- Giá trị tối thiểu, tối đa, trong phạm vi...
- Không giống mật khẩu, đúng captcha, trùng mã
- Không như mong đợi của việc tính toán nào đó...

Validation

❑ Kiểm lỗi cho form:

- ✓ Không để trống họ và tên
- ✓ Không để trống điểm
- ✓ Điểm phải có giá trị từ 0 đến 10
- ✓ Phải chọn ngành

Nhập sai



Quản lý sinh viên

Vui lòng sửa các lỗi sau đây !

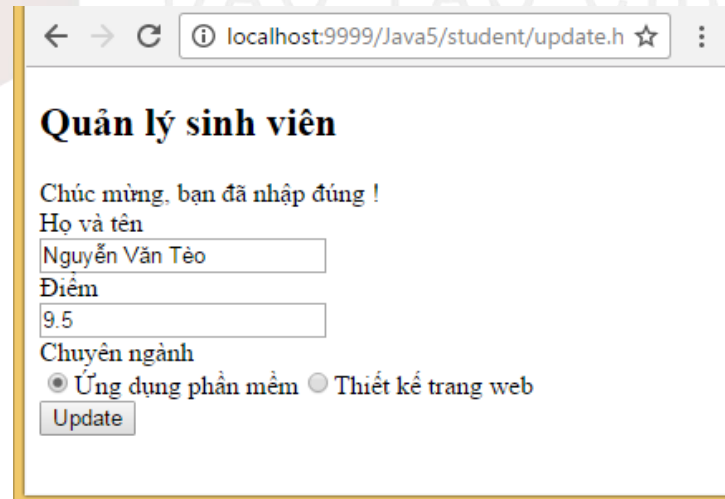
Họ và tên
 Không được để trống họ và tên !

Điểm
 Điểm không hợp lệ !

Chuyên ngành
Vui lòng chọn chuyên ngành !
☒ Ứng dụng phần mềm ☐ Thiết kế trang web

Update

Nhập đúng



Quản lý sinh viên

Chúc mừng, bạn đã nhập đúng !

Họ và tên

Điểm

Chuyên ngành
☒ Ứng dụng phần mềm ☐ Thiết kế trang web

Update

Kiểm lỗi bằng tay

```
@RequestMapping(value="validate1", method=RequestMethod.POST)
public String validate1(ModelMap model,
    @ModelAttribute("student") Student1 student, BindingResult errors) {
    if(student.getName().trim().length() == 0){
        errors.rejectValue("name", "student", "Vui lòng nhập họ tên !");
    }
    if(student.getMark() == null){
        errors.rejectValue("mark", "student", "Vui lòng nhập điểm !");
    }
    else if(student.getMark() < 0 || student.getMark() > 10){
        errors.rejectValue("mark", "student", "Điểm không hợp lệ !");
    }
    if(student.getMajor() == null){
        errors.rejectValue("major", "student", "Vui lòng chọn một chuyên ngành !");
    }
    if(errors.hasErrors()){
        model.addAttribute("message", "Vui lòng sửa các lỗi sau đây !");
    }
    else{
        model.addAttribute("message", "Chúc mừng, bạn đã nhập đúng !");
    }
    return "student1";
}
```

Đối số này nên là
đối số cuối cùng

rejectValue() cho phép
bổ sung thông báo lỗi
cho thuộc tính **mark** của
bean **student**

Phương thức
hasErrors() cho
biết có thông báo
lỗi nào hay không?

Hiển thị lỗi

```
{message}
<form:form action="student/validate1.htm" modelAttribute="student">
  <div>Họ và tên</div>
  <form:input path="name"/>
  <form:errors path="name"/>

  <div>Điểm</div>
  <form:input path="mark"/>
  <form:errors path="mark"/>

  <div>Chuyên ngành</div>
  <form:errors path="major" element="div"/>
  <form:radiobuttons path="major" items="{majors}"
    itemLabel="name" itemValue="id"/>

  <div>
    <button>Validate 1</button>
  </div>
</form:form>
```

Hiển thị lỗi thuộc tính **name** của bean student

Thuộc tính **element** chỉ ra thẻ chứa thông báo lỗi. Mặc định là

Thư viện sử dụng

Thư viện
hibernate-validator

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${spring.version}</version>
  </dependency>
  <dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>1.1.0.Final</version>
  </dependency>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>5.4.2.Final</version>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
  </dependency>
</dependencies>
```

Thư viện
validation-api

SOFT
IA LẬP TRÌNH

Validate Annotation

- ❑ Nạp các annotation lỗi vào các thuộc tính lớp **bean** được sử dụng để nhận dữ liệu form.

```
public class Student2 {  
    @NotBlank(message="Không được để trống họ và tên !")  
    private String name;  
  
    @NotNull(message="Không được để trống điểm !")  
    @DecimalMin(value="0", message="Điểm không hợp lệ !")  
    @DecimalMax(value="10", message="Điểm không hợp lệ !")  
    private Double mark;  
  
    @NotNull(message="Vui lòng chọn chuyên ngành !")  
    private String major;  
  
    getters/setters  
}
```

- ❑ **Annotation** kiểm lỗi:

- ✓ @NotBlank: kiểm trường name rỗng.
- ✓ @NotNull: kiểm trường mark và major null.
- ✓ @DecimalMin(), @DecimalMax(): kiểm khoảng số thực.

Validate Annotation

```
@RequestMapping(value="validate2", method=RequestMethod.POST)  
public String validate2(ModelMap model,  
    @Validated @ModelAttribute("student") Student2 student, BindingResult errors) {  
    if(errors.hasErrors()){  
        model.addAttribute("message", "Vui lòng sửa các lỗi sau đây !");  
    }  
    else{  
        model.addAttribute("message", "Chúc mừng, bạn đã nhập đúng !");  
    }  
    return "student2";  
}
```

- ✓ Chỉ cần bổ sung **@Validated** trước bean nhận dữ liệu form thì các thuộc tính của bean sẽ được kiểm lỗi theo các luật đã nạp vào các trường bean.

Validate Annotation



Annotation	Ý nghĩa	Ví dụ
NotBlank	Chuỗi không rỗng	@NotBlank()
NotNull	Không cho phép null	@NotNull()
NotEmpty	Chuỗi /tập hợp không rỗng	@NotEmpty()
Length	Độ dài chuỗi	@Length(min=5, max=10)
Max	Giá trị số nguyên tối đa	@Max(value="10")
Min	Giá trị số nguyên tối thiểu	@Min(value="0")
Size, Range	Phạm vi số nguyên tối	@Size(min=0, max=10)
DecimalMax	Giá trị số thực tối đa	@DecimalMin(value="5.5")
DecimalMin	Giá trị số nguyên tối thiểu	@DecimalMax(value="9.5")
Future	Thời gian trong tương lai	@Future()
Past	Thời gian trong quá khứ	@Past()
Pattern	So khớp biểu thức chính qui	@Pattern(regexp="[0-9]{9,10}")
Email	Đúng dạng email	@Email()
CreditCardNumber	Đúng dạng số thẻ tín dụng	@CreditCardNumber()
URL	Đúng dạng url	@URL()
SafeHtml	Không được chứa thẻ HTML	@SafeHtml()

Custom Validator

✓ Spring cho phép lập trình viên **tùy biến validate dữ liệu của form** thông qua implement **interface Validator**.

❑ Các bước thực hiện

✓ **Bước 1:** Tạo class implement từ interface **Validator**.

✓ **Bước 2:** Trong phương thức **supports()** của class vừa tạo, kiểm tra đối tượng truyền vào có phải là class hay không.

✓ **Bước 3:** Phương thức **validate()** kiểm tra dữ liệu nhập vào của trường bất kỳ, inject lỗi nếu như có lỗi.

✓ **Bước 4:** **Inject** class vừa tạo vào Controller cần sử dụng validation.

❖ **Note:** *Class validator vừa tạo phải đánh dấu **@Component** để khai báo lớp này là một bean.*

Custom Validator

Kiểm tra đối tượng truyền vào

```
@Component
public class RegisterValidator implements Validator{

    public boolean supports(Class<?> clazz) {
        // Kiểm tra đối tượng truyền vào có phải là class không
        return RegisterUserDto.class.equals(clazz);
    }

    public void validate(Object target, Errors errors) {
        // Ép kiểu đối tượng truyền vào
        RegisterUserDto user = (RegisterUserDto) target;
        // Kiểm tra thuộc tính null hoặc rỗng
        if(user.getConfirm() == null || user.getConfirm().length() == 0) {
            // add thông báo yêu cầu nhập
            errors.rejectValue("confirm", "Vui lòng nhập lại mật khẩu!");
        }
        // So khớp mật khẩu và nhập lại mật khẩu
        else if (!user.getConfirm().equals(user.getPassword())) {
            // add thông báo lỗi
            errors.rejectValue("confirm", "Mật khẩu không khớp!");
        }
    }
}
```

Kiểm tra lỗi của thuộc tính trong class

Custom Validator

Inject class
validator vào
để sử dụng

```
@Controller
@RequestMapping("/account")
public class AccountController {

    @Autowired
    RegisterValidator validator;

    @PostMapping("/register")
    public String postRegister(Model model,
        @ModelAttribute @Valid RegisterUserDto account,
        BindingResult errors) {
        validator.validate(account, errors);
        if(errors.hasErrors()) {
            return "account/register";
        }
    }
}
```

Gọi hàm
validate() để
kiểm tra dữ liệu

FT
TRÌNH

Bài tập 2

❑ Sử dụng lại bài tập 1 validate cho các trường của User entity và Role entity.

❖ **User entity:**

- ✓ NotBlank: Email, password, fullname.
- ✓ Email: email.
- ✓ MinLength: fullname, password.

❖ **Role entity:**

- ✓ NotBlank: name.

Các bước thực hiện



- ✓ **Bước 1:** Tải thư viện validation cho dự án.
- ✓ **Bước 2:** Thêm các annotation bắt lỗi nhập liệu vào từng thuộc tính cho lớp User entity và Role entity.
- ✓ **Bước 3:** Cập nhật lại Controller.
- ✓ **Bước 4:** Thêm các thẻ errors của thư viện Spring MVC cho các trường cần validate dữ liệu trong form.
- ✓ **Bước 5:** Run...

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

Entity class

User entity

```
public class User {  
    @Id  
    private String id;  
    @NotBlank(message = "Vui lòng nhập email!")  
    @Email(message = "Email không đúng định dạng!")  
    private String email;  
    @NotBlank(message = "Vui lòng nhập họ tên!")  
    private String fullname;  
    @NotBlank(message = "Vui lòng nhập mật khẩu!")  
    private String password;  
    private String avatar;  
    private String phone;  
    private String address;  
    private String website;  
    private String facebook;  
    @NotBlank(message = "Vui lòng chọn loại người dùng!")  
    private String roleId;  
}
```

Role entity

```
public class Role {  
    @Id  
    private String id;  
    @NotBlank(message = "Vui lòng nhập tên!")  
    private String name;  
    private String description;  
  
    public Role() {  
    }  
  
    public Role(String id, String name, String description) {  
        super();  
        this.id = id;  
        this.name = name;  
        this.description = description;  
    }  
}
```

Controller class

Cập nhật lại
phương thức
add()

```
@Controller
@RequestMapping("admin/user")
public class UserController {

    @Autowired
    private UserService userServiceImpl;

    public String index() {}

    public String add(Model model) {}

    @PostMapping("add")
    public String add(Model model, @Valid @ModelAttribute("user") User user,
        BindingResult errors) {
        if(errors.hasErrors()) {
            return "userAdd";
        }
        userServiceImpl.add(user);
        return "redirect:/admin/user";
    }

    public String edit(@PathVariable String id, Model model) {}

    @PostMapping("edit")
    public String edit(Model model, @Valid @ModelAttribute("user") User user,
        BindingResult errors) {
        if(errors.hasErrors()) {
            return "userEdit";
        }
        userServiceImpl.update(user);
        return "redirect:/admin/user";
    }
}
```

Cập nhật lại
phương thức
edit()

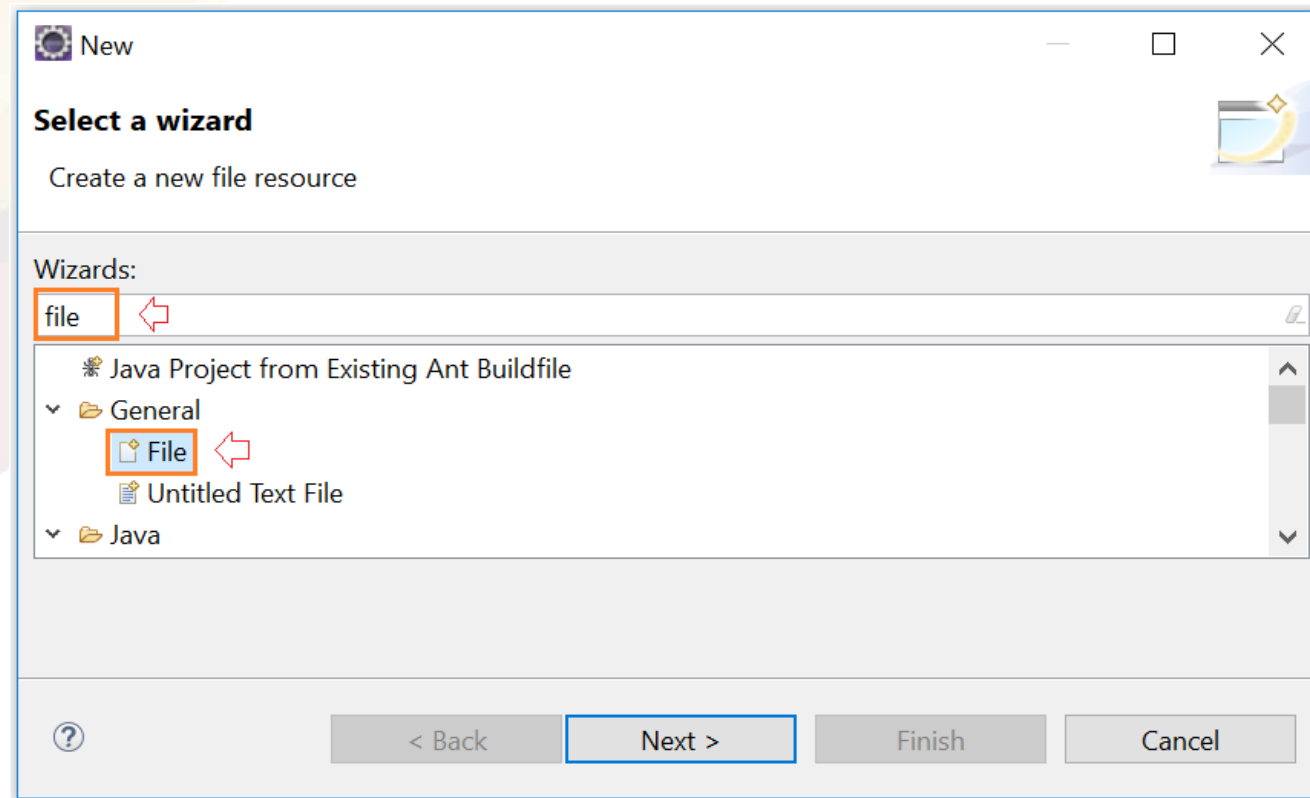
OFT
ÁP TRÌNH

Message Properties

- ❑ Để tránh hash code các thông báo lỗi, chúng ta sẽ sử dụng file **properties** khai báo các thông báo lỗi và sử dụng bean **MessageResource** để đọc các đoạn văn bản đó.
- ❑ Các bước thực hiện:
 - ✓ **Bước 1:** Tạo file message.properties (nằm trong thư mục src/main/resource).
 - ✓ **Bước 2:** Khai báo các thông báo lỗi của từng thuộc tính trong file vừa tạo.
 - ✓ **Bước 3:** Tạo bean messageResource để đọc thông báo lỗi (Khai báo trong file WebMvcConfig).
 - ✓ **Bước 4:** Xóa các message trong Entity class.

Tạo file properties

- ❑ Click chuột phải vào src/main/resource → New → Other.
- ❑ Trong cửa sổ hiện ra search từ khóa “file” → Chọn File → Chọn Next → Đặt tên file là **messages.properties**.



messages.properties

message.properties

```
NotBlank.user.email = Vui lòng nhập email!  
Email.user.email = Email không đúng định dạng!  
NotBlank.user.fullname = Vui lòng nhập họ tên!  
NotBlank.user.password = Vui lòng nhập mật khẩu!  
  
NotBlank.role.name = Vui lòng nhập tên!
```

❑ Cấu trúc message

```
@NotBlank(message = "Vui lòng nhập họ tên!")  
private String fullname;
```

```
NotBlank.user.fullname = Vui lòng nhập họ tên!
```

```
@Valid @ModelAttribute("user") User user;
```

MessageResource bean

MessagesResource Bean

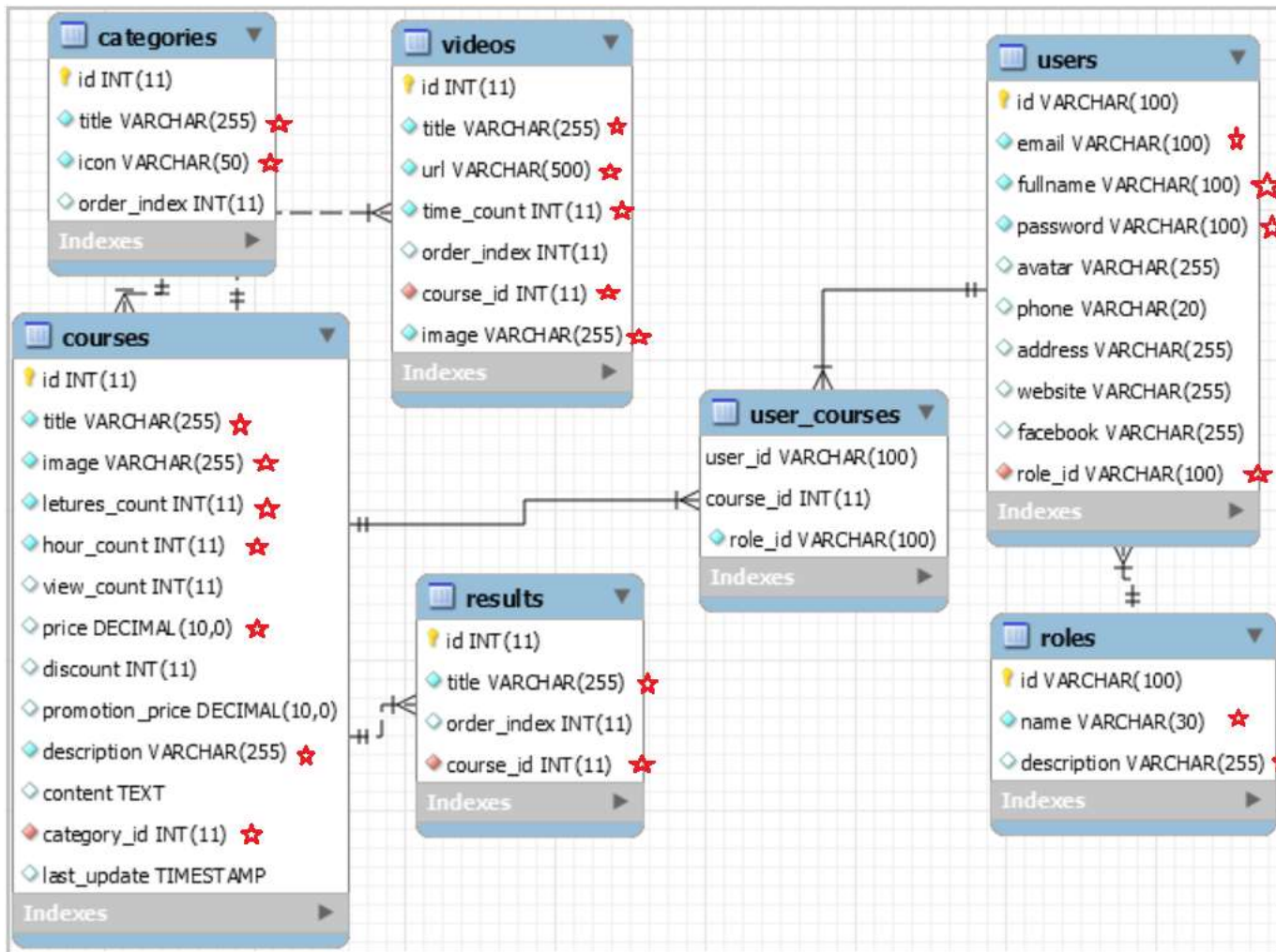
```
@Bean
public MessageSource messageSource() {
    ReloadableResourceBundleMessageSource source =
        new ReloadableResourceBundleMessageSource();
    source.setBasename("classpath:messages");
    source.setDefaultEncoding("UTF-8");
    return source;
}
```

Khai báo tên
resources.

Sử dụng định
dạng tiếng Việt.

Bài tập về nhà

- ❑ Hoàn thành các entity còn lại trong dự án.
- ✓ Viết CRUD theo mô hình MVC đã học.
- ✓ Validate dữ liệu cho các trường được đánh dấu.
- ❖ **Note:** user_account chưa cần làm.



☐ Layout

- ✓ Sử dụng Apache Tiles
- ✓ Cấu hình static resource.

☐ Data Binding

- ✓ Data Binding là gì?
- ✓ Cách binding dữ liệu.

☐ Validation

- ✓ Validation là gì?
- ✓ Validate bằng tay.
- ✓ Validate bằng Annotation.
- ✓ Hiển thị lỗi validate.
- ✓ Custom validation.