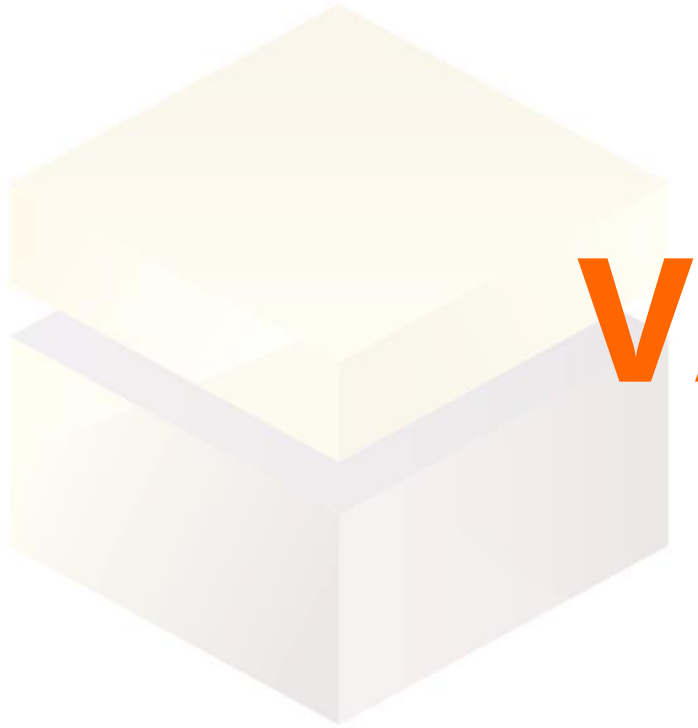


VALIDATION

CYBERSOFT.EDU.VN

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

Nội dung

- ❑ Validation là gì?
- ❑ Validate bằng tay.
- ❑ Validate bằng Annotation.
- ❑ Custom validation.
- ❑ Bài tập trên lớp.
- ❑ Áp dụng vào dự án.
- ❑ Mô hình MVC trong Spring
- ❑ Tổ chức dự án theo mô hình MVC.
- ❑ Phân tầng chức năng.

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

Validation là gì?

- **Validation** (kiểm tra dữ liệu đầu vào) là **xác nhận dữ liệu từ người dùng**. Bất cứ khi nào người dùng gửi dữ liệu vào hệ thống của bạn, thì dữ liệu đó phải cần được xác nhận.
- Dữ liệu vào không hợp lệ sẽ gây các lỗi khó lường. Vì vậy cần kiểm soát dữ liệu vào để **ngăn chặn các cuộc tấn công**, **dữ liệu xấu**, **lỗi của người dùng**.
- Trong **Spring MVC** có hai cách validate:
  - ✓Validation bằng tay.
  - ✓Validation bằng Annotation.

CYBERSOFT
HỌC ĐỂ LÀM CHUYÊN GIA LẬP TRÌNH

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…

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

Bài tập

- **Validate** cho các thuộc tính của User entity và Role entity.
  - **Role**:
    - ✓ **Name**: không được bỏ trống, ít nhất 4 ký tự.
    - ✓ **Description**: Không được bỏ trống.
  - **User**:
    - ✓ **Email**: Không được bỏ trống, định dạng email.
    - ✓ **Password**: Không được bỏ trống, ít nhất 6 ký tự.
    - ✓ **Fullname**: Không được bỏ trống.

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

Validation

- **Kiểm lỗi cho form:**
  - ✓Không để trống tên
  - ✓Không để trống mô tả
  - ✓Điểm ít nhất 4 ký tự

Validate bằng tay

```java
@RequestMapping(value = "", method = RequestMethod.POST)
@ResponseBody
public ResponseEntity add(@RequestBody Role role, BindingResult errors) {
    // Kiểm tra xem thuộc tính name có được nhập chưa
    if(role.getName().length() == 0) {
        errors.rejectValue("name", "role", "Vui lòng nhập tên!");
    }
    // Kiểm tra xem thuộc tính description có được nhập chưa
    if(role.getDescription().length() == 0) {
        errors.rejectValue("description", "role", "Vui lòng nhập mô tả!");
    }

    // Nếu có lỗi xảy ra thì trả về lỗi
    if(errors.hasErrors())
        return new ResponseEntity(errors.getAllErrors(), HttpStatus.BAD_REQUEST);

    // Nếu không có lỗi thì thực hiện thêm mới
    role.setId(UUID.randomUUID().toString());
    listRole.add(role);
    return new ResponseEntity(listRole, HttpStatus.OK);
}
```

Đối số này nên là đối số cuối cùng

rejectValue() cho phép bổ sung thông báo lỗi cho thuộc tính **name** của bean **role**

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

Validate Annotation

- **Spring** hỗ trợ validate các field/thuộc tính khi submit form bằng cách sử dụng các annotation validate của **hibernate-validator**, **javax-validation**.

```
public class Role {

    private int id;

    @NotBlank(message = "Vui lòng nhập tên!")
    @Min(value = 4, message = "Tên ít nhất 4 ký tự!")
    private String name;

    @NotBlank(message = "Vui lòng nhập mô tả!")
    private String description;

}
```

Thư viện sử dụng

```xml
<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 hibernate-validator

Thư viện validattion-api

Validate Annotation

```
@RequestMapping(value = "", method = RequestMethod.POST)
@ResponseBody
public ResponseEntity add(@Valid @RequestBody Role role, BindingResult errors) {

    // Nếu có lỗi xảy ra thì trả về lỗi
    if(errors.hasErrors())
        return new ResponseEntity(errors.getAllErrors(), HttpStatus.BAD_REQUEST);

    // Nếu không có lỗi thì thực hiện thêm mới
    role.setId(UUID.randomUUID().toString());
    listRole.add(role);
    return new ResponseEntity(listRole, HttpStatus.OK);
}
```

- ✓ Chỉ cần bổ sung **@Validated** hoặc **@Valid** 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

- Chỉ cần thêm **@Valid** hoặc **@Validated** trước đối tượng cần kiểm tra để cho Spring xác nhận đối tượng được theo dõi (Subscribe).
- **BindingResult** là đối tượng của Spring giữ kết quả của việc xác nhận và ràng buộc và chứa các lỗi có thể đã xảy ra. Các **BindingResult** phải **để ngay sau khi đối tượng được xác nhận** tức là đặt ngay **@ModelAttribute** nếu không Spring sẽ không xác nhận đối tượng và ném một ngoại lệ.
- **Spring** tự động đọc và kích hoạt các Annotation nếu bạn đã kích hoạt “annotation-driven”.

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.*

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

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

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.

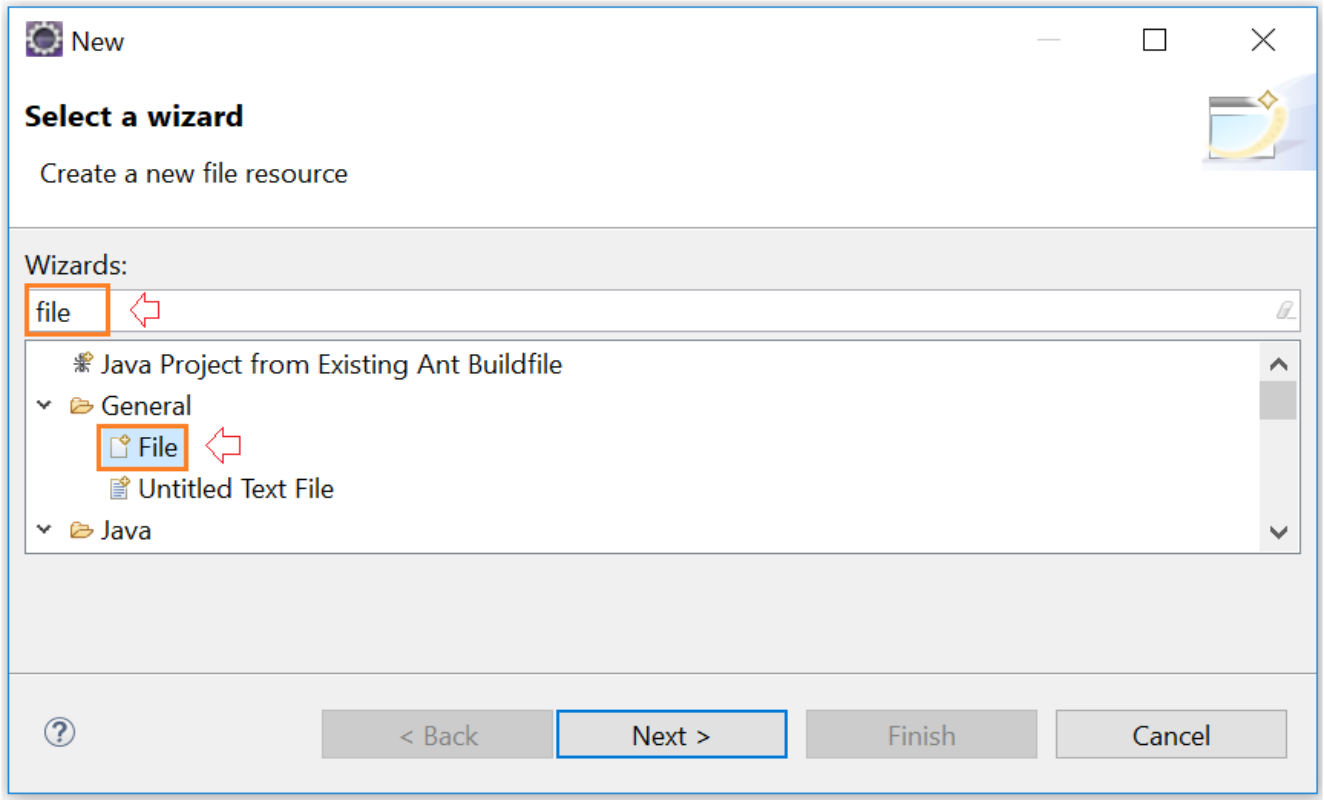
✓ **Bước 3**: Tạo bean messageResource để đọc thông báo lỗi (Khai báo trong file WebMvcConfig).

✓ **Bước 4**: Xóa các message trong Entity class.

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

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.

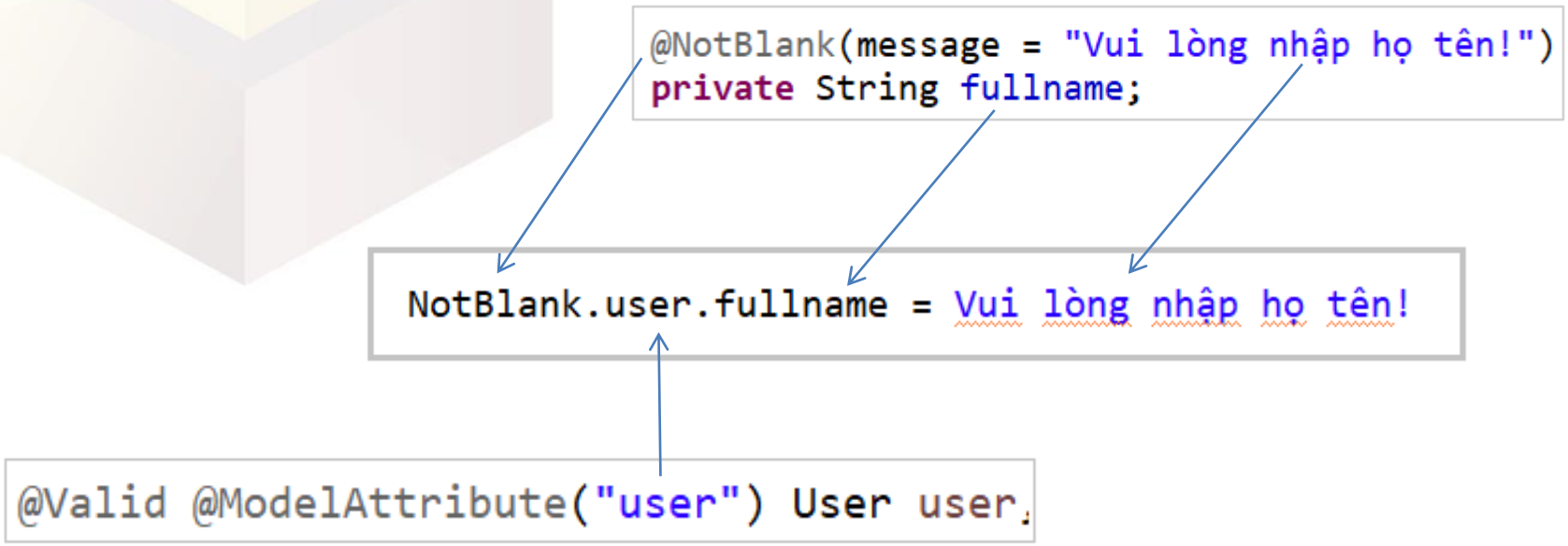


messages.properties

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



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.

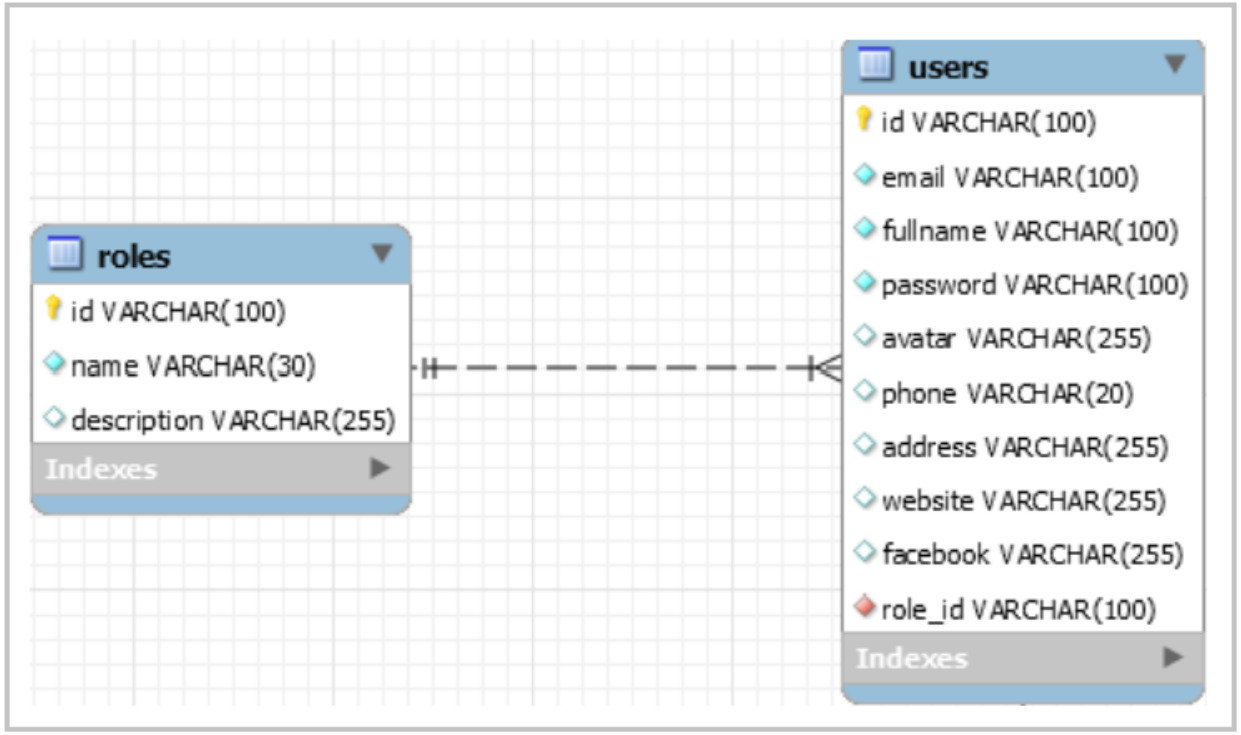
Sử dụng định dạng tiếng Việt.

Bài tập

- Xây dựng module (CRUD) quản lý người dùng và quản lý quyền 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.

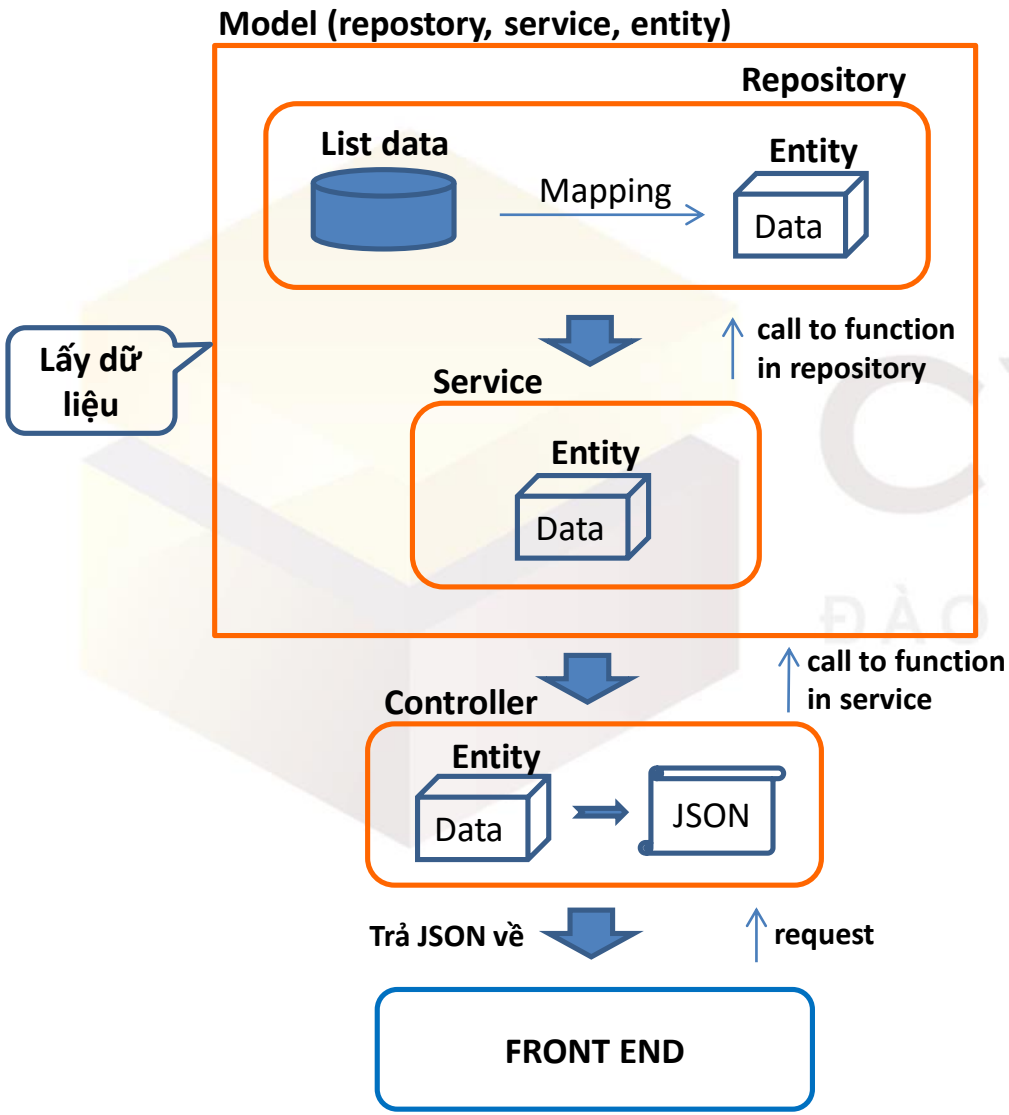


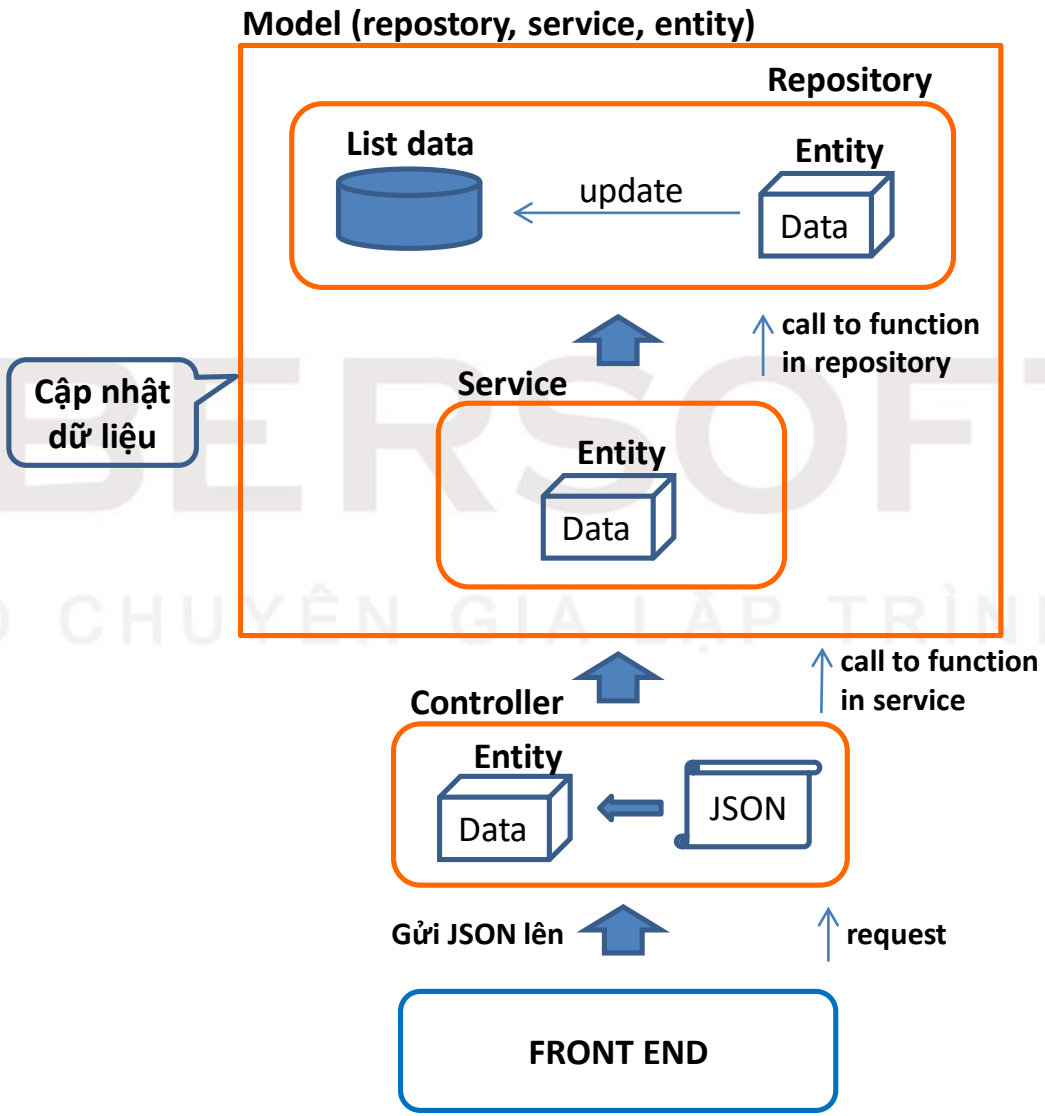
Các bước thực hiện

- ✓ **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.
  - ✓ Đinh 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.

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

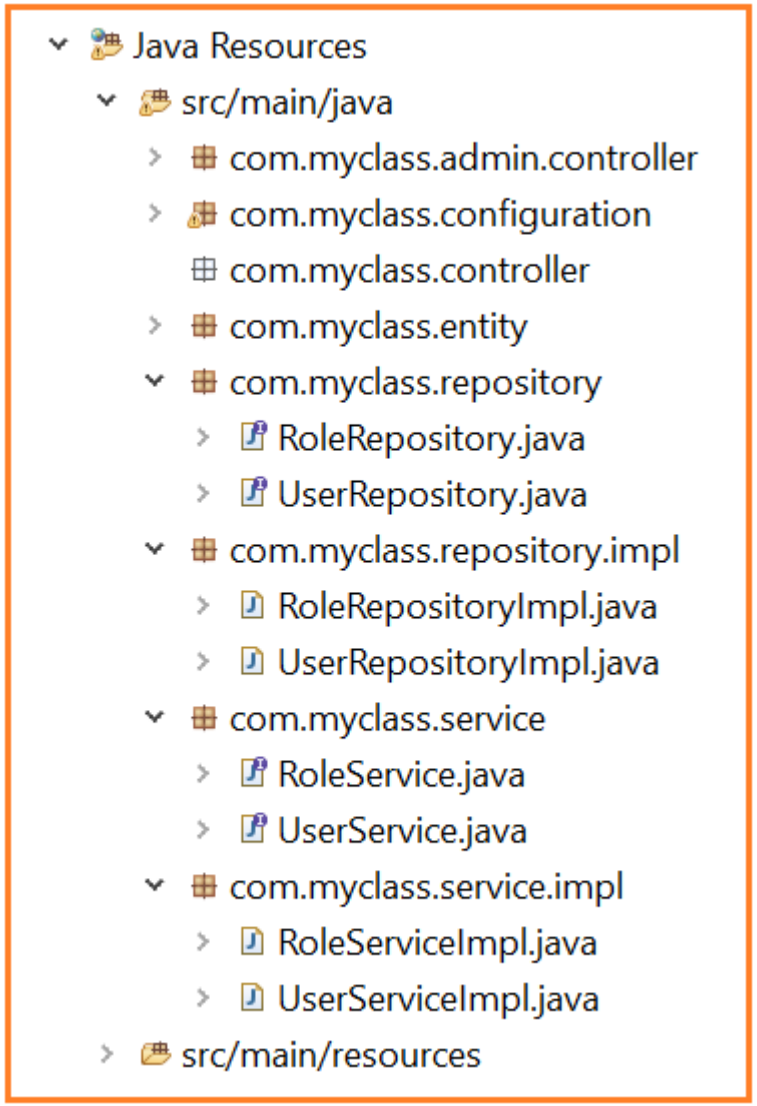
Truyền data giữa các tầng





CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

Cấu trúc thư mục



Role interface

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

User interface

```java
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

```java
@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

```java
@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

```java
@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

```java
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);
}
```

UserService

```java
@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

```java
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

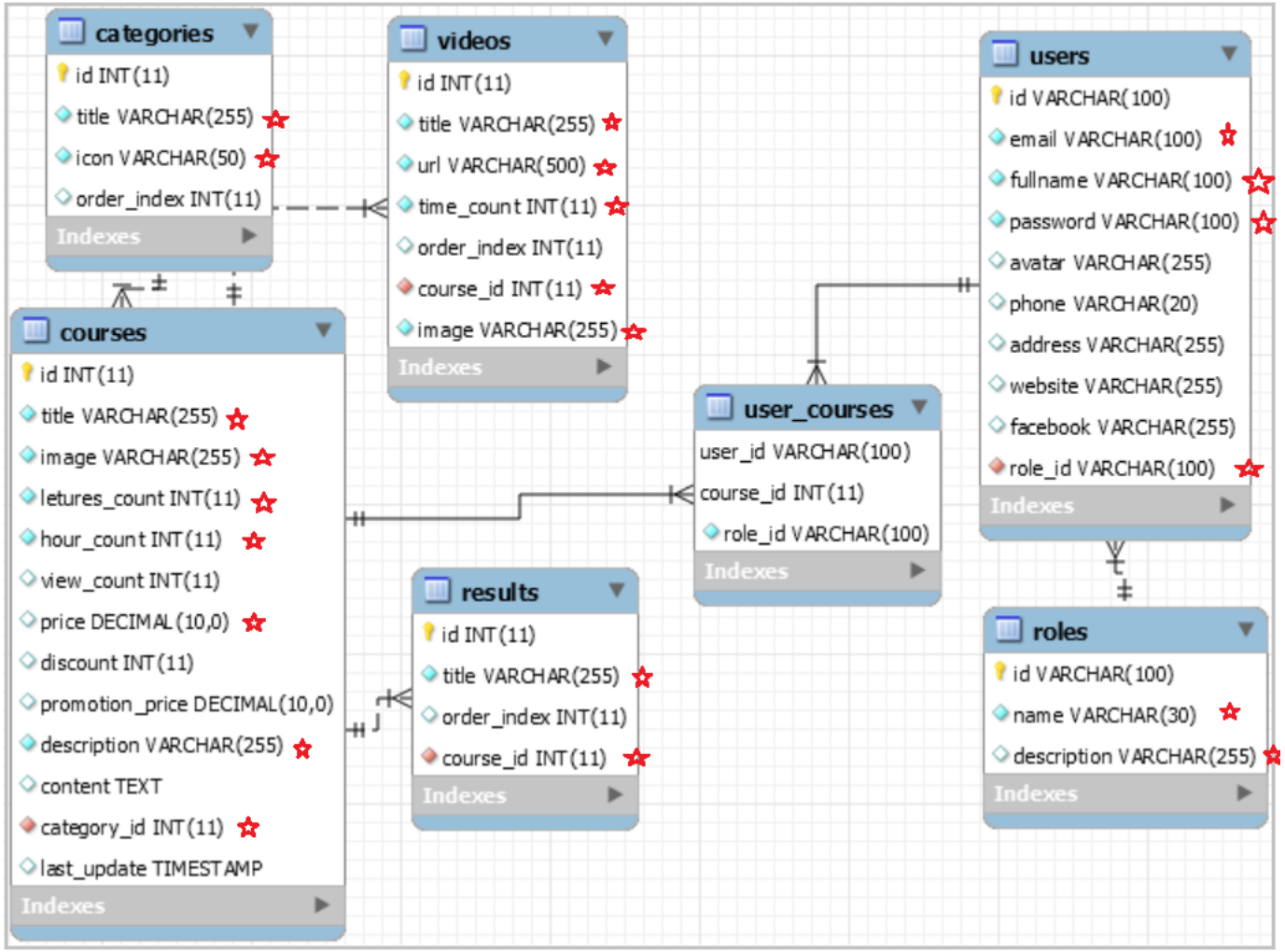
Sử dụng đối tượng Model để chuyển dữ liệu từ Controller đến view

```java
@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

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.



Tổng kết

- ❑ Validation là gì?
- ❑ Validate bằng tay.
- ❑ Validate bằng Annotation.
- ❑ Custom validation.
- ❑ Bài tập trên lớp.
- ❑ Áp dụng vào dự án.
- ❑ Mô hình MVC trong Spring
- ❑ Tổ chức dự án theo mô hình MVC.
- ❑ Phân tầng chức năng.

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH