CS545 Final Exam

### Professor Rujuan Xing

**Student Id: Name:**

The exam takes 2 hours. Total score is 60.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1-5  (5) | 6  (5) | 7  (10) | 5  (10) | 6  (28) | SCI  (2) | Total  (60) |
|  |  |  |  |  |  |  |

Please read the exam policy before you start the exam. Exam Policy:

There is no tolerance policy for exams. **You will be asked to leave the exam room immediately without a warning** once you do the following things which mean you’ll get **NC**.

1. You are caught cheating or trying to cheat.
2. Answers should be written with a Pen or Pencil, but if you want to use a pencil please bring your own eraser and sharpener. You’re not allowed to borrow from other students or proctors during exam.
3. All mobile phones should be turned off and submitted along with your luggage at the beginning of the exam.
4. Restroom policy:
   1. No phones
   2. Only 5 minutes
   3. Not allowed to go to restroom after someone submit paper
5. You’re not allowed to go out room for water.
6. All your answers must be written on your exam paper.

# Please write down your answer clearly. If I cannot read your answer, you’ll not get credit.

Good luck!

## PART I (5 points): True/False Questions

1. @ExceptionHandler declares the class of exception handled by the method.

T F

1. @ResponseStatus Exceptions cannot be handled by @ControllerAdvice.

T F

1. Method level authorization is less important than URL authorization.

T F

1. To determine the language of a user, the Client HTTP Accept-Language header can be used.

T F

1. Spring MVC provides support for file uploading. To enable it, the only thing that needs to be done is to declare the tag with enctype="multipart/form-data" in form.

T F

## PART II (15 points): Short Answer Questions

1. (5 points) What are the ways to handle Exception in Spring MVC? Provide code to explain.
2. (10 points) Cross cutting concerns are an important technology used in Spring & Spring MVC.

Explain the concept of Interceptors used in Spring MVC. What is an interceptor? How are multiple interceptors associated with a resource handled? Explain how interceptors are used for Security authorization and Exception handling.

Below is an example configuration of a custom interceptor to further help in your explanation. Explain the configuration. Where/when [in this example] is the interceptor invoked? What are the methods that VolunteerInterceptor.java must implement?

<mvc:interceptors>

<mvc:interceptor>

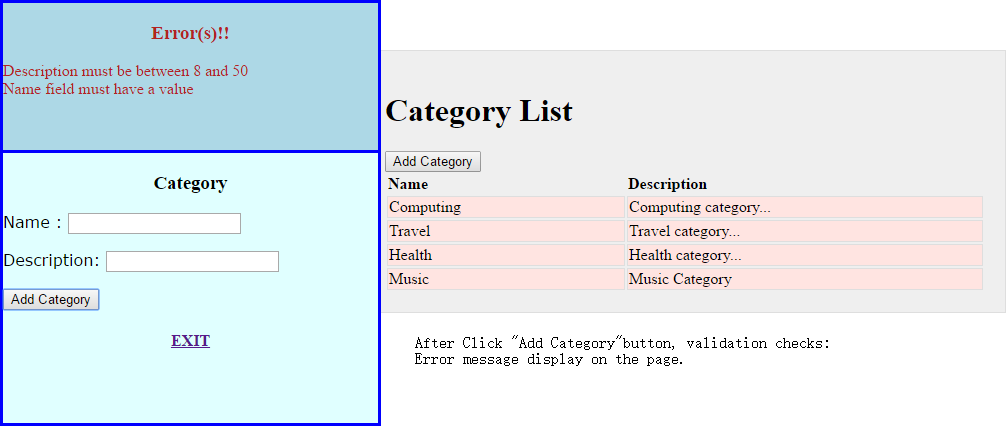
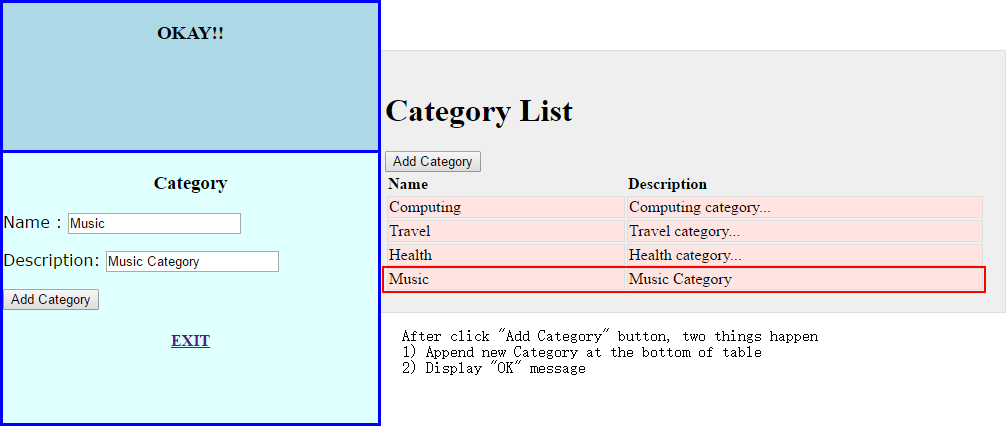
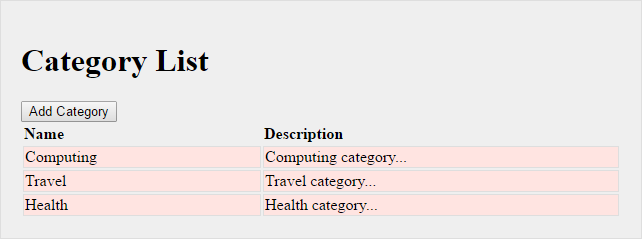
<mvc:mapping path=*"/home/\*\*"* />

<bean class=*"mum.edu.interceptor.VolunteerInterceptor"* />

</mvc:interceptor>

</mvc:interceptors>

## PART III (38 points): Programming Questions

1. (10 Points) This problem involves a Restful Web Service implementation. The Restful service is responsible for the “Add Category” function on the Category List Page. Tasks:
   1. Complete the CategoryController
   2. Complete the AJAX function.

# DomainErrors.java

This is the Object that is the error payload sent in response to the AJAX call JUST like the Lab in class. It contains a String errorType and a list of error messages. As a reminder, here is how you load it up with errors on the server:

DomainErrors errors = **new** DomainErrors(); errors.setErrorType("CategoryValidationError"); **for** (FieldError fieldError : fieldErrors) {

DomainError error =

### new

DomainError(messageAccessor.getMessage(fieldError)); errors.addError(error);

}

**CategoryList.jsp –** you might need this file for reference in your AJAX success and error callback function.

<table style="width: *100%*;">

<tr>

</tr>

<th style="width: *40%*;">Name</th>

<th style="width: *60%*;">Description</th>

<c:forEach items=*"*${categories}*"* var=*"category"*>

<tr>

</tr>

<td>${category.name}</td>

<td>${category.description}</td>

</c:forEach>

</table>

<!-- Success - or Validation errors -->

<div id=*"result"* style="display:*none*" >

<p id=*"success"* ></p>

<p id=*"errors"* ></p>

</div>

**Main.js**

**var** contextRoot = "/" + window.location.pathname.split('/')[1];

**function** categorySubmit() {

**var** dataToSend = JSON.stringify(serializeObject($('#categoryForm')));

$.ajax({

**//code to make AJAJ Call**

**type: ‘POST’,**

**data:** dataToSend,

url: contextRoot + ‘/addcategory’,

success : **function**( category ) {

$('#success').html("");

$('#errors').html("");

$("#success").append('<H3 align="center"> OKAY!! <H3>');

$('#result').show();

**var** newRow = '<tr><td>' + category.name +

'</td><td>'+ category.description + '</td></tr>';

$('table tbody').append(newRow);

},

error : **function**( xmlresponse ) {

**if** ( xmlresponse.responseJSON.errorType= ‘CategoryValidationError’ ) {

$('#success').html("");

$('#errors').html("");

$("#errors").append('<H3 align="center"> Error(s)!! <H3>');

$("#result").append('<p>');

**var** errorList = xmlresponse.responseJSON.errors;

**//code to loop errorList**

**var errorMsg = ‘’;**

**$.each(errorList, function(i, error){**

**errorMsg + = error.message + ‘<br/>’;**

**});**

$("#errors").append(errorMsg);

$("#errors").append('</p>');

$('#result').show();

} **else** {

alert(errorObject.responseJSON.errors(0));

}}});}

# CategoryController.java

@Controller

**public class** CategoryController {

@Autowired

**private** CategoryService categoryService; //assume has save() method.

@PostMapping(“/addCategory”)

@ResponseBody

**public Category** saveCategory( @Valid @RequestBody Category category ) {

categoryService.save(category);

### return category ;

}

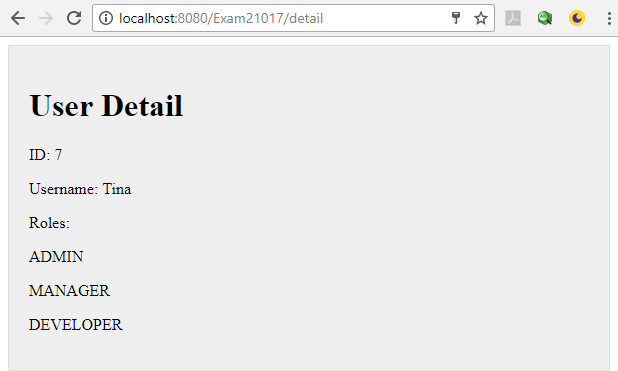
}

1. (28 points) This is the “full stack” problem AKA “tall thin person” implementation. It is expected that you will use “good coding practices” as discussed in class.

TASKS:

* 1. Define necessary attributes in User.java and Role.java according to screenshots. Annotate them for ORM (think about their relationship). No need to add setters and getters. Each User has a List of <Role>. You many add more properties/attributes in User or Role classes.
  2. Complete the UserController. Follow best practices and make sure the URLs matche the screenshots.
  3. Complete&Annotate the UserServiceImpl & RoleServiceImpl. You need to call methods from each repository to do actual Save or Update functionality.
  4. Complete the UserRepository
     1. There should be a method that finds a User by username. The name should be LookupTheUserThruUsername.
  5. Complete the RoleRepository
     1. Define a class level declared query which finds all ROLES that start with letter “A”. The name should be GetRolesStartWithA.
  6. Implement UserForm.jsp using Spring Form Tag

### UserForm.jsp

**Detail.jsp**

**@Entity**

**@Table(name=”user”)**

**public class** User **implements** Serializable {

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

### private long id;

@Column(name=”uname”)

@Size(min=’4’, max=’10’, message={user.username})

**private** String username;

@Column(name=”pwd”)

@NotNull

**private** String password;

@ManyToMany

Set<Role> roles;

}

@Entity

**public class** Role **implements** Serializable{

@Id

@GeneratedValue(….)

### private long id;

**private** String role;

}

@Controller

**public class** UserController {

@Autowired

UserService userService;

@Autowired

RoleService roleService;

@GetMapping(“/addUser”)

**public** String getAddUserForm( Model model ) {

model.addAttribute(“newUser”, new User());

// get Roles

model.addAttribute(“roles”, roleService.findAll());

return “UserForm”

}

@PostMapping(“/addUser”)

**public** String processAddUserForm(@Valid @ModelAttribute(“newUser”)User user, BindingResult result, RedirectAttribute ra ){

if(result.hasErrors()){

return “UserForm”;

}

userService.save(user);

ra.addFlashAttribute(user);

return “redirect:/detail”;

}

@GetMapping(“/detail”)

**public** String detail( ) {

return “userdetail”;

}

}

//Code completed for this interface

**public interface** UserService { User save(User user);

//Code completed for this interface

**public interface** RoleService { List<Role> findAll();

}

Role get(**long** id);

}

@Service

**public class** UserServiceImpl **implements** UserService {

@Autowired

UserRepository userRepository;

@Override

**public** User save(User user) {

return userRepository.save(user);

}

}

@Service

**public class** RoleServiceImpl **implements** RoleService {

@Autowired

RoleRepository roleRepository;

@Override

**public** List<Role> findAll() {

return roleRepository.findAll();

}

@Override

**public** Role get(**long** id) {

**return roleRepository.findById(id).get();**

}}

@Respository

**public interface** UserRepository extends CrudRepository<User, Long> {

}

@Respository

**public interface** RoleRepository extends CrudRepository<Role, Long> {

@Query(value=”select r from Role r where r.role like ?1%”)

List<Role> getRolesStartWithA(String letter)

@Query(value=”select r from Role r where r.role like :letter%”)

List<Role> getRolesStartWithA(@Param(“letter”) String abc)

@Query(value=”select \* from role where role like :letter%”, nativeQuery=true)

List<Role> getRolesStartWithA(@Param(“letter”) String abc)

@Query(value=”select \* from role where role like ?1%”, nativeQuery=true)

List<Role> getRolesStartWithA(String abc)

}

### UserForm.jsp

< form:form modelAttribute=” newUser” >

<fieldset><legend>Add a User</legend>

<p>

</p>

<p>

</p>

<label for=*"username"*>Username: </label>

< form:input path=”username” >

<label for=*"password"*>Password: </label>

< form:input path=”password” >

<p>

<label for=*"roles"*>Roles </label>

//write code here to generate multi-selector

<form:select path=”roles” items=”${roles}” multiple=”true” itemLabel=”role” itemValue=”id” />

</p>

<p id=*"buttons"*>

<input id=*"submit"* type=*"submit"* value=*"Add a User"*>

</p>

</fieldset>

</ form:form >

1. (2 Points) Write one or two paragraphs relating a point from the course to a principle from SCI. (No Credit if copy from main points chart. You have to use your own words).