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. You're not allowed to go out room for water.
5. 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 EXPLAIN:_@ExceptionHandler is for that purpose. It can be used either in a @Controller or @ControllerAdvice to identify to Spring MVC that the Exception is to be mapped to the method.

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

   T **F** @ResponseStatus Exceptions cannot be handled by @ControllerAdvice
   EXPLAIN:_@ControllerAdvice can handle any exception globally. @ResponseStatus marked exceptions are obviously exceptions and can be handled by @ControllerAdvice.

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

   T **F** Method level authorization can/should be placed in the service layer, thereby ensuring maximum security of resources.

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

   T F **There is a LocaleResolver, AcceptHeaderLocalResolver, just for this purpose. Based on the Language/Localization setting on the client browser, it will set the Locale object on the server.**

5. 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** The multipart resolver needs to be configured.

PART II (15 points): Short Answer Questions

6. (5 points) What are the ways to handle Exception in Spring MVC? Provide simple code to explain.

Exceptions can be handled EITHER individually OR Globally across ALL Controllers with @ControllerAdvice

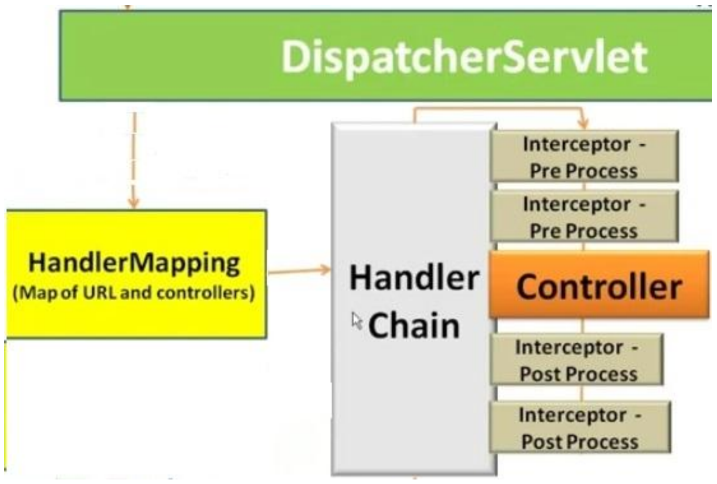
```
@ControllerAdvice
public class ControllerExceptionHandler {

   @ExceptionHandler(ProductNotFoundException.class)
   public ModelAndView handleError(HttpServletRequest req, ProductNotFoundException exception) {
      ModelAndView mav = new ModelAndView();
      mav.addObject("invalidProductId", exception.getProductId());
      mav.addObject("exception", exception);
      mav.addObject("url", req.getRequestURL() + "?" + req.getQueryString());
      mav.setViewName("productNotFound");
      return mav;
   }
}
```

7. (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?

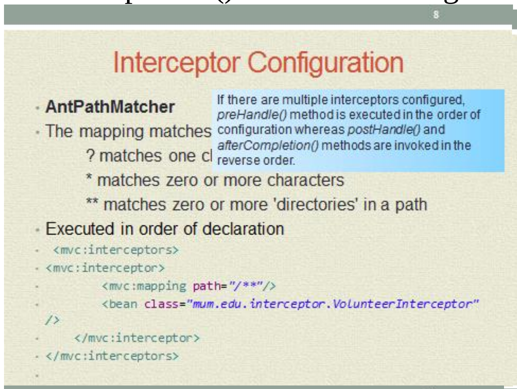
```
<mvc:interceptors>
     <mvc:interceptor>
          <mvc:mapping path="/home/**" />
          <bean class="mum.edu.interceptor.VolunteerInterceptor" />
     </mvc:interceptor>
</mvc:interceptors>
```

Configures Handler interceptor. Based on URL path described. In this case = `"/home/**"` `which means`
`The home URL and ALL urls that are children of home, e.g., /home/kitchen, /home/kitchen/stove`


DispatcherServlet
HandlerMapping
(Map of URL and controllers)
Handler Chain
Interceptor - Pre Process
Interceptor - Pre Process
Controller
Interceptor - Post Process
Interceptor - Post Process

- Part of Spring MVC Handler mapping mechanism
- Fine grained access to the handler/controller
    - preHandle() - before controller execution
    - postHandle() – after controller execution
        - Can expose additional model objects to the view via the given ModelAndView.

- afterCompletion() - after rendering the view. Allows for proper resource cleanup.



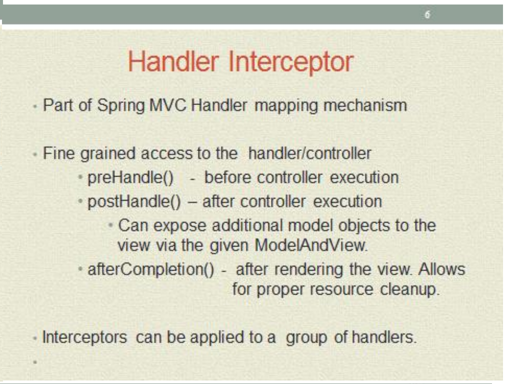
8

Interceptor Configuration

- **AntPathMatcher**
- The mapping matches
  - ? matches one cl
  - * matches zero or more characters
  - ** matches zero or more 'directories' in a path
- Executed in order of declaration

If there are multiple interceptors configured, *preHandle()* method is executed in the order of configuration whereas *postHandle()* and *afterCompletion()* methods are invoked in the reverse order.

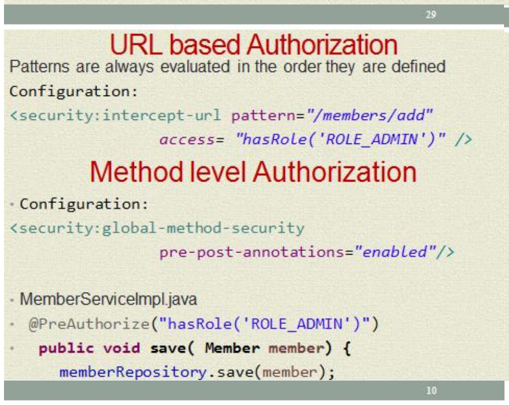
```
 <mvc:interceptors>
<mvc:interceptor>
        <mvc:mapping path="/**"/>
        <bean class="mum.edu.interceptor.VolunteerInterceptor"
/>
    </mvc:interceptor>
</mvc:interceptors>
```



6

Handler Interceptor

- Part of Spring MVC Handler mapping mechanism
- Fine grained access to the handler/controller
  - preHandle()   - before controller execution
  - postHandle() – after controller execution
    - Can expose additional model objects to the view via the given ModelAndView.
  - afterCompletion() - after rendering the view. Allows for proper resource cleanup.
- Interceptors can be applied to a group of handlers.



29

URL based Authorization

Patterns are always evaluated in the order they are defined

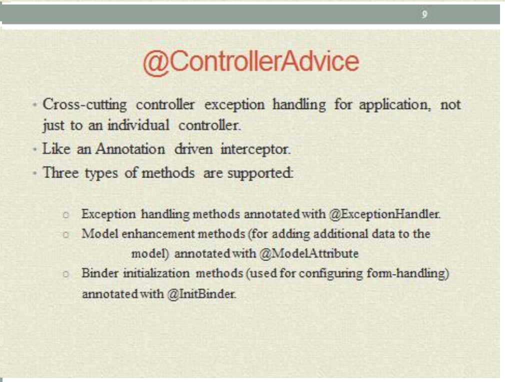
```
Configuration:
<security:intercept-url pattern="/members/add"
               access= "hasRole('ROLE_ADMIN')" />
```

Method level Authorization

```
Configuration:
<security:global-method-security
               pre-post-annotations="enabled"/>
```

- MemberServiceImpl.java

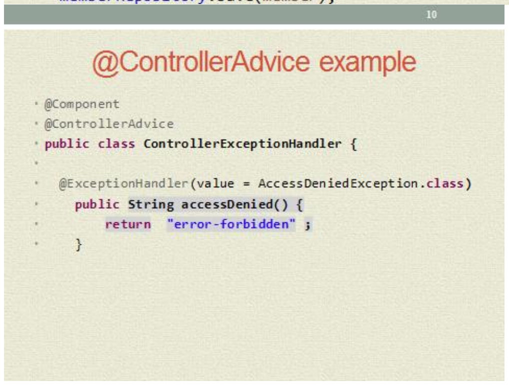
```
 @PreAuthorize("hasRole('ROLE_ADMIN')")
  public void save( Member member) {
    memberRepository.save(member);
```



9

@ControllerAdvice

- Cross-cutting controller exception handling for application, not just to an individual controller.
- Like an Annotation driven interceptor.
- Three types of methods are supported:
  - Exception handling methods annotated with @ExceptionHandler.
  - Model enhancement methods (for adding additional data to the model) annotated with @ModelAttribute
  - Binder initialization methods (used for configuring form-handling) annotated with @InitBinder.



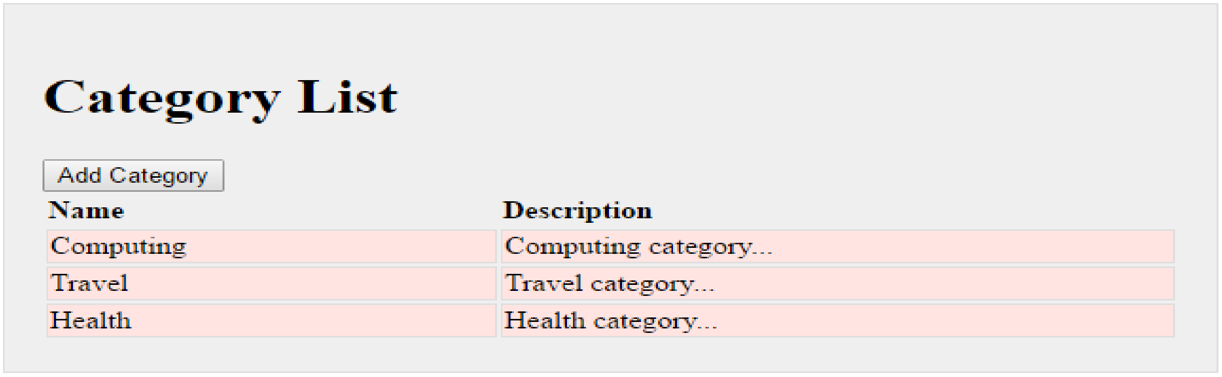
10

@ControllerAdvice example

```
@Component
@ControllerAdvice
public class ControllerExceptionHandler {

  @ExceptionHandler(value = AccessDeniedException.class)
   public String accessDenied() {
       return  "error-forbidden" ;
   }
```

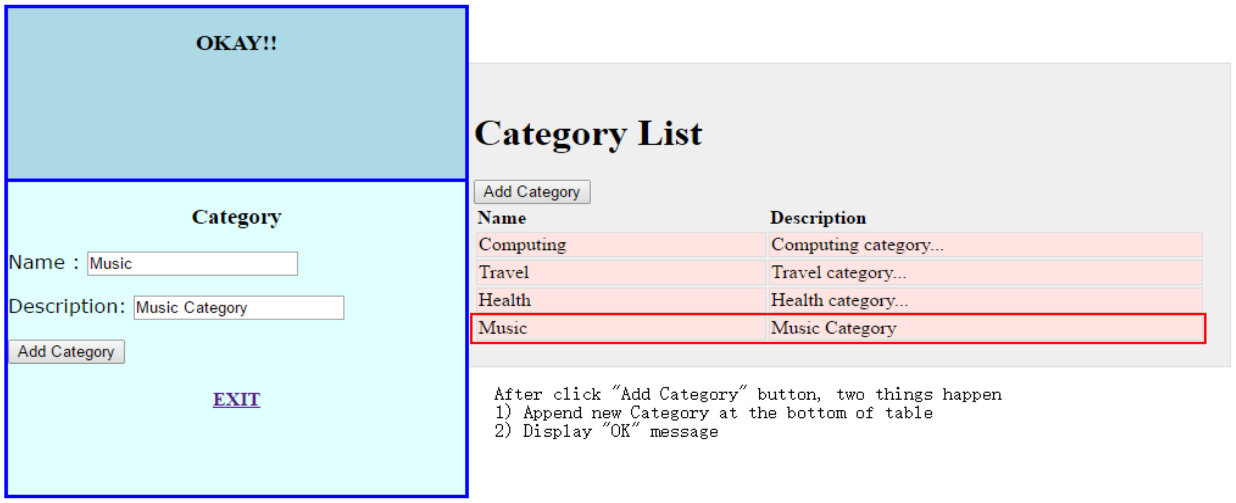
PART III (38 points): Programming Questions

8. (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.



Category List

Name	Description
Computing	Computing category...
Travel	Travel category...
Health	Health category...



OKAY!!

Category

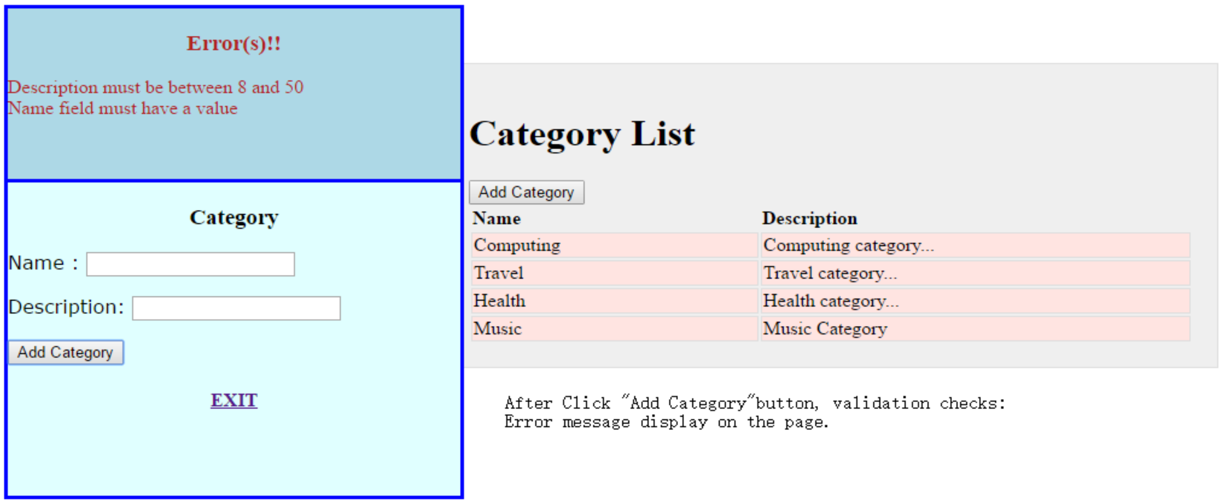
Name : Music

Description: Music Category

Category List

Name	Description
Computing	Computing category...
Travel	Travel category...
Health	Health category...
Music	Music Category

```
After click "Add Category" button, two things happen
1) Append new Category at the bottom of table
2) Display "OK" message
```



Error(s)!!

Description must be between 8 and 50
Name field must have a value

Category

Name :

Description:

Category List

Name	Description
Computing	Computing category...
Travel	Travel category...
Health	Health category...
Music	Music Category

```
After Click "Add Category"button, validation checks:
Error message display on the page.
```

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>
            <th style="width: 40%;">Name</th>
            <th style="width: 60%;">Description</th>
      </tr>
      <c:forEach items="${categories}" var="category">
            <tr>
                  <td>${category.name}</td>
                  <td>${category.description}</td>
            </tr>
      </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({
              type: 'POST',
              url: contextRoot + '/addCategory',
//            url: '/Book5Rest/addCategory',
              dataType: "json",            // Accept header
              data:dataToSend,
              contentType: 'application/json',    // Sends - Content-type
              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(errorObject ){
                     if (errorObject.responseJSON.errorType == "ValidationError") {
                            $('#success').html("");
                            $('#errors').html("");
                            $("#errors").append( '<H3 align="center"> Error(s)!!
<H3>');
                                $("#result").append( '<p>');

                                var errorList = errorObject.responseJSON.errors;
                             $.each(errorList,  function(i,error) {
                                   $("#errors").append( error.message + '<br>');
                            });
                             $("#errors").append( '</p>');
                             $('#result').show();
                     }
                     else {
                            alert(errorObject.responseJSON.errors(0));   // "non"
Validation Error
                     }
              }
       });
}
```

CategoryController.java

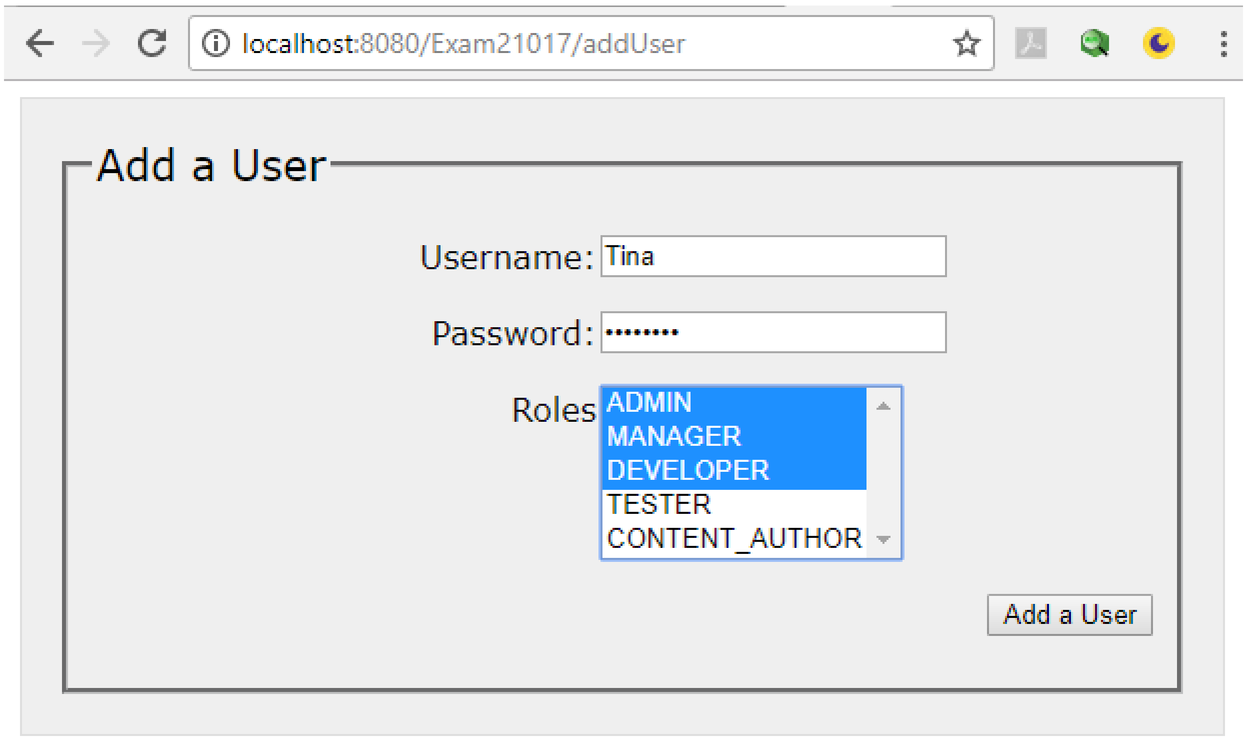
```
@Controller
public class CategoryController {

    @Autowired
    private CategoryService categoryService;

    @RequestMapping(value = "/addCategory", method =
RequestMethod.POST)
    public @ResponseBody Category saveCategory( @Valid  @RequestBody
Category category) {

          categoryService.save(category);
          return category;
    }

}
```

9. (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`
      a. There should be a method that finds a User by username. The name should be `LookupTheUserThruUsername`.
   5. Complete the `RoleRepository`
      a. 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`

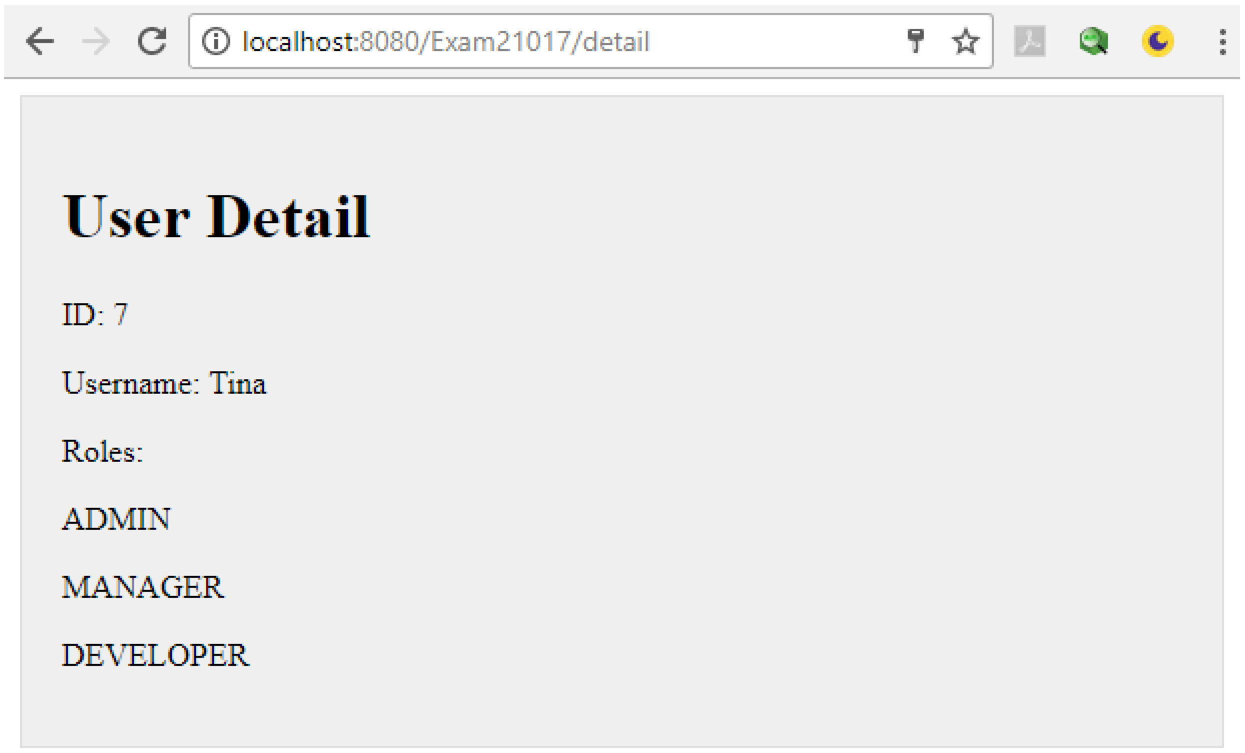


localhost:8080/Exam21017/addUser

Add a User

Username:	Tina
Password:	••••••••
Roles	ADMIN, MANAGER, DEVELOPER (selected); TESTER; CONTENT_AUTHOR

Detail.jsp



User Detail

ID: 7

Username: Tina

Roles:

ADMIN

MANAGER

DEVELOPER

```
@Entity
public class User implements Serializable {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	@Column(name = "UID")
	private long id;

	@Column(name = "USERNAME")
	private String username;

	@Column(name = "PASSWORD")
	private String password;

	@ManyToMany(cascade = { CascadeType.MERGE })
	private List<Role> roles = new ArrayList<>();
}
```

```
@Entity
public class Role implements Serializable{

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	@Column(name = "RID")
	private long id;

	private String role;
}

@Controller
public class UserController {

	@Autowired
	private UserService userService;

	@Autowired
	private RoleService roleService;

	@RequestMapping(value="/addUser", method=RequestMethod.GET)
	public String getAddUserForm(@ModelAttribute("user") User user,
Model model) {
		List<Role> roles = roleService.findAll();
		model.addAttribute("roles", roles);
		return "UserForm";
	}

	@RequestMapping(value="/addUser", method=RequestMethod.POST)
	public String processAddUserForm(@ModelAttribute("user") User
user, BindingResult result,
			RedirectAttributes redirectAttributes, Model model) {
		User savedUser = userService.save(user);
		redirectAttributes.addFlashAttribute("user", savedUser);
		return "redirect:/detail";
	}

	@RequestMapping("/detail")
	public String detail() {
		return "Detail";
	}

	}
```

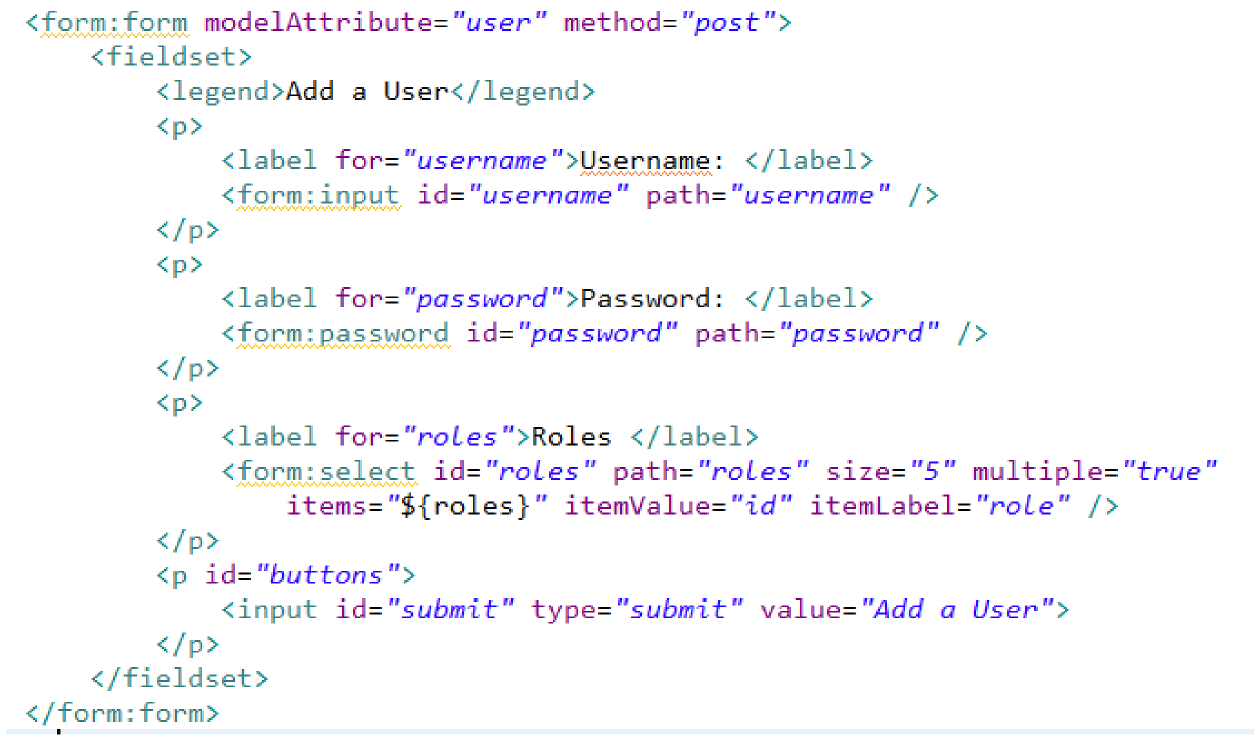
```java
//Code completed for this
interface
public interface UserService {
	User save(User user);
}
```

```java
//Code completed for this
interface
public interface RoleService {
	List<Role> findAll();

	Role get(long id);
}
```

```java
@Service
public class UserServiceImpl implements UserService {

	@Autowired
	private UserRepository userRepository;

	@Override
	public User save(User user) {
		return userRepository.save(user);
	}

}


@Service
public class RoleServiceImpl implements RoleService {

	@Autowired
	private RoleRepository roleRepository;

	@Override
	public List<Role> findAll() {
		return (List<Role>) roleRepository.findAll();
	}

	@Override
	public Role get(long id) {
		return roleRepository.findOne(id);
	}

}
```

```
@Repository
public interface UserRepository extends CrudRepository<User, Long>{

LookupTheUserThruUsername
}


@Repository
public interface RoleRepository extends CrudRepository<Role, Long>{
    getRolesStartWithA()
}
```

UserForm.jsp



```
<form:form modelAttribute="user" method="post">
    <fieldset>
        <legend>Add a User</legend>
        <p>
            <label for="username">Username: </label>
            <form:input id="username" path="username" />
        </p>
        <p>
            <label for="password">Password: </label>
            <form:password id="password" path="password" />
        </p>
        <p>
            <label for="roles">Roles </label>
            <form:select id="roles" path="roles" size="5" multiple="true"
                items="${roles}" itemValue="id" itemLabel="role" />
        </p>
        <p id="buttons">
            <input id="submit" type="submit" value="Add a User">
        </p>
    </fieldset>
</form:form>
```

10. (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).