CST8277 (24F) Assignment 4: REST ACME Medical

Jakarta EE Group Project

Please read this document carefully, all sections (perhaps multiple times to make sure you find all the places that say you '***must***'). If your submission does not meet the requirements as stated here, you may lose marks even though your program runs. Additionally, this assignment is also a teaching opportunity – **materials presented here will be on the Final Exam!**

Overview

For this group project, ACME Medical Corp. has tasked you to map their medical database schema to Java POJOs. After mapping, ACME Medical Corp. would like you to develop RESTful APIs and expose each POJO as a resource that supports CRUD (create, read, update, delete) operations. ACME Medical Corp. would also like you to write JUnit tests to confirm that all your RESTful APIs are working properly. Finally, you are required to produce a Maven 'Surefire' pass-fail report for all JUnit test cases.

Model Entities

For this project, you need to use JPA annotations to map the Java POJOs to tables in the database. Additionally, a new entity SecurityUser should be mapped to the **SECURITY_USER** table and assigned one of two JEE Security Roles: ***USER_ROLE*** or ***ADMIN_ROLE***. The security will be backed by additional database tables: **SECURITY_ROLE** table and a **USER_HAS_ROLE** join table. This means an additional entity SecurityRole needs to be mapped as well.

Theme for the Group Project

The theme for the Group Project is to bring together everything you have learned this semester:

1. Use JPA to map a database schema to Java POJOs
2. Use session beans for implementing business logic of the application
3. Develop RESTful APIs for the back-end resources
4. Secure the RESTful APIs by adding JEE security roles to control who can invoke which operation
5. Test using JUnit tests – a series of test cases that demonstrate the operation and correctness of the RESTful APIs
6. Use the Maven 'Surefire' plugin to generate a pass-fail report for all test cases

Submission

Your group project submission is to be uploaded to Brightspace/Activities/Assignments. **NOTE: One Brightspace submission is enough for the entire group. Your group also needs to demo the completed project to your respectively lab professor as well and answer your lab professor's questions and clarifications.**

The submission ***must*** include:

- Zip of your project folder (**Note:** Do not reduce anything from the skeleton project but feel free to add and modify the existing annotations and/or code.)
- Updated Postman collection **REST-ACMEMedical-Sample.postman_collection.json** for your project. (**Note:** Feel free to add and modify the REST requests/messages. When submitting your completed work, please include **only** those REST requests that are working properly for your REST API system and remove those REST requests that are not working or not supported by your REST API

system. Your work will be graded based on the REST requests included in your Postman collection.)

- **Style**: Every class file has a multi-line comment block at the top giving the name of the file, your names (authors), and date modified.
    - ***Important*** - The names of all group members ***must*** appear at the top of each and every source code file submitted; otherwise, you will lose marks (up to a score of 0) for the coding portion of the rubric.
- JUnit Test Suite: Test cases that demonstrate the operation and correctness of the RESTful APIs.

Download and Import the Project to Your Eclipse Workspace

1. Download *REST-ACMEMedical-Skeleton.zip*
2. Extract the folder content to some location
3. In Eclipse, go to **File/Import…/Existing Maven Projects**
4. Navigate to your project folder
5. Make sure that there is a check mark beside the project's pom.xml file
6. Click **Finish**

After Importing the Project to Your Eclipse Workspace

Do the following steps to view the TODOs/tasks and hints for this assignment:

1. Go to **Window** menu, select **Preferences**
2. Type "task" on the "type filter text" textbox on the left
3. Under **Java Compiler**, you should see "Task Tags"
4. Select Task Tags on the right, and then click **New…**
5. On the **Tag** text box, type "Hint"
6. On the **Priority** dropdown, select Low and click **OK**
7. Click **Apply** (Note: You have to let Eclipse rebuild your project if it asks.) and then click **Apply and Close**
8. If you don't see the **Tasks** on your Eclipse, do the following:
    a. Go to **Window** menu, select **Show View**, select Other…
    b. Under **General**, select **Tasks** and click **Open**
9. To filter the tasks, do the following:
    a. On the **Tasks** bar, select the filter icon on the right
    b. It should bring up the **Filters** window
    c. Make sure that **Show all items** is not checked
    d. Make sure that **TODOs** is checked
    e. Under Scope, make sure that **On elements in select project** is selected
    f. Then click **New…**, select **New Configuration**, click **Rename**, enter "Hints" on the **Configuration Name** pop-up, and then click **OK**
    g. Again, make sure that **On elements in selected project** is selected for Hints
    h. Then under **Description**, type in "Hint" beside **Text: contains** and click **Apply and Close**
    i. Then make sure that **Show all items** checkbox is checked on the **Filters** window

Task One – Finish the JPA Annotations for Entities

1. First of all, finish all the TODOs in the models (i.e. Physician, Patient, MedicalCertificate, MedicalTraining, MedicalSchool, PublicSchool, PrivateSchool, etc.); you may use the Medicine and Prescription entities as guides/references in completing the other entities/models
2. As you complete the JPA annotations on the entities, refer to the provided ER diagram (ACMEMedical-Diagram.pdf) to understand how the entities and tables are related to each other; you must annotate the classes/entities based on how they are related as shown in ACMEMedical-Diagram.pdf

Task Two – Finish Custom Authentication Mechanism

In the starter code, you will find code similar to the JEE security demo 'REST-Demo-Security':

```
@ApplicationScoped
public class CustomAuthenticationMechanism implements
HttpAuthenticationMechanism {

    @Inject
    protected IdentityStore identityStore;

---
@ApplicationScoped
@Default
public class CustomIdentityStore implements IdentityStore {

    @Inject
    protected CustomIdentityStoreJPAHelper jpaHelper;
---
@Singleton
public class CustomIdentityStoreJPAHelper {

     private static final Logger LOG = LogManager.getLogger();

     @PersistenceContext(name = PU_NAME)
     protected EntityManager em;

     public SecurityUser findUserByName(String username) {
          LOG.debug("find a User By the Name={}", username);
          SecurityUser user = null;
          //TODO:  ...
```

The **TODO** here is to make the custom authentication mechanism actually use the database and must be done in the **CustomIdentityStoreJPAHelper** class.

Task Three – Relationship Between SecurityUser and Physician

One of the tasks to be done is to map a 1:1 relationship between a SecurityUser and a Physician. Please see the TODO inside SecurityUser class. This is done so that when the custom authentication mechanism successfully resolves the Principal (SecurityUser implements the Principal interface), it can be inject'd into your code – then the developer can un-wrap the object and find the SecurityUser inside and from there access the related Physician as was done in getPhysicianById() in PhysicianResource:

```
@Inject
protected SecurityContext sc;

@GET
@RolesAllowed({ADMIN_ROLE, USER_ROLE})
@Path(RESOURCE_PATH_ID_PATH)
public Response getPhysicianById(@PathParam(RESOURCE_PATH_ID_ELEMENT)
int id) {
     LOG.debug("try to retrieve specific physician " + id);
     Response response = null;
     Physician physician = null;

     if (sc.isCallerInRole(ADMIN_ROLE)) {
          physician = service.getPhysicianById(id);
          response = Response.status(physician == null ?
Status.NOT_FOUND : Status.OK).entity(physician).build();
     } else if (sc.isCallerInRole(USER_ROLE)) {
          WrappingCallerPrincipal wCallerPrincipal =
(WrappingCallerPrincipal) sc.getCallerPrincipal();
          SecurityUser sUser = (SecurityUser)
wCallerPrincipal.getWrapped();
          physician = sUser.getPhysician();
          if (physician != null && physician.getId() == id) {
               response =
Response.status(Status.OK).entity(physician).build();
          } else {
               throw new ForbiddenException("User trying to access
resource it does not own (wrong userid)");
          }
     } else {
          response = Response.status(Status.BAD_REQUEST).build();
     }
     return response;
}
```

There is no requirement to create an (administrative) API for creating SecurityUser's and SecurityRole's – you may populate the **SECURITY_USER**, **SECURITY_ROLE** and **USER_HAS_ROLE** tables using 'raw' SQL (or use MySQL Workbench).

Task Four and Lesson 1 – Building a REST API

We need to build JAX-RS REST'ful resources for our model objects/entities (remember the security requirements from Task Three):

```java
@Path(PHYSICIAN_RESOURCE_NAME)
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public class PhysicianResource {

	private static final Logger LOG = LogManager.getLogger();

	@EJB
	protected ACMEMedicalService service;

	...

	@GET
	@RolesAllowed({ADMIN_ROLE, USER_ROLE})
	@Path(RESOURCE_PATH_ID_PATH)
	public Response getPhysicianById(@PathParam(
RESOURCE_PATH_ID_ELEMENT) int id) {
		...
	}

	...
}
```

The main focus is on C-R-U-D:

- Q1: What REST request/message creates a Physician?
  - What endpoint API should we send the above message to?
- Q2: What REST request/message relates a Medicine to a Physician?
  - What endpoint API should we send the above message to?
- .. you get the idea (!)

NOTE: There are also 2 **TODO**s inside **ACMEMedicalService.java** which your group need to complete.

Swagger and Postman

Before writing your JUnit tests, you can use Swagger or Postman to test your REST APIs.

You should do the tutorial at https://app.swaggerhub.com/help/tutorials/openapi-3-tutorial to become familiar with how to use the editor and add to the .yaml document.

Or you can use https://www.postman.com/downloads/.

Task Five and Lesson 2 – Securing REST Endpoints

You need to put JEE security annotations on your REST'ful resources to enforce the following rules:

- Only a user with the SecurityRole 'ADMIN_ROLE' can get the list of all physicians.
- A user with either the role 'ADMIN_ROLE' or 'USER_ROLE' can get a specific physician. However, there is logic inside the **getPhysicianById** method that disallows a 'USER_ROLE' user from getting a physician that is not linked to the SecurityUser.
- Only a user with the SecurityRole 'ADMIN_ROLE' can add a new physician.
- Any user can retrieve the list of MedicalTraining and MedicalSchool.
- Only an 'ADMIN_ROLE' user can apply CRUD to one or all MedicalCertificate.
- Only a 'USER_ROLE' user can read their own MedicalCertificate.
- Only an 'ADMIN_ROLE' user can associate a Medicine and/or Patient to a Physician.
- Only an 'ADMIN_ROLE' user can delete any entities.
- **Q3:** Based on these rules, what role should be allowed to add new MedicalTraining? New MedicalSchool? etc.

Task Six and Lesson 3 - Building JUnit Tests

For Java JAX-RS resources (e.g. PhysicianResource), there is a Client API to remotely invoke behavior on the REST'ful resources (https://javaee.github.io/tutorial/jaxrs-client.html).
[Note: In TestACMEMedicalSystem.java, the first test-case **test01_all_physicians_with_adminrole** is implemented below. It uses JUnit 5's @BeforeAll and @BeforeEach annotations to make things more neat and tidy.

```
@Test
public void test01_all_physicians_with_adminrole() throws
JsonMappingException, JsonProcessingException {
    Response response = webTarget
        //.register(userAuth)
        .register(adminAuth)
        .path(PHYSICIAN_RESOURCE_NAME)
        .request()
        .get();
    assertThat(response.getStatus(), is(200));
    List<Physician> physicians = response.readEntity(new
GenericType<List<Physician>>(){});
    assertThat(physicians, is(not(empty())));
    assertThat(physicians, hasSize(1));
}
```

Remember, negative testing is also useful, i.e. for example:

```
assertThat(response.getMediaType(), is(not(MediaType.APPLICATION_XML));
```

You ***must*** build a collection with 50 (minimum) tests to various REST'ful URI endpoints of your ACME Medical application, testing the full C-R-U-D lifecycle of the entities, building associations between entities … all using REST requests/messages.

Fetch Strategy

Fetch should always be lazy. Because of this, you will sometimes get **LazyInitializationException**. The way to solve this is to use "fetch" in your named queries as explained below.

Normally, you will have basic named queries like these:

```
@NamedQuery(name = MedicalSchool.ALL_MEDICAL_SCHOOLS_QUERY_NAME, query =
"SELECT distinct ms FROM MedicalSchool ms")
@NamedQuery(name = MedicalSchool.SPECIFIC_MEDICAL_SCHOOL_QUERY_NAME, query =
"SELECT distinct ms FROM MedicalSchool ms WHERE ms.id = :param1")
```

With join fetch, you will grab the entity from the DB and grab the dependencies as well. You can have multiple join fetches to fetch multiple dependencies.

```
@NamedQuery(name = MedicalSchool.ALL_MEDICAL_SCHOOLS_QUERY_NAME, query =
"SELECT distinct ms FROM MedicalSchool ms LEFT JOIN FETCH ms.medicalTrainings")
@NamedQuery(name = MedicalSchool.SPECIFIC_MEDICAL_SCHOOL_QUERY_NAME, query =
"SELECT distinct ms FROM MedicalSchool ms LEFT JOIN FETCH ms.medicalTrainings WHERE
sc.id=:param1")
```

Security Users

Role	User	Pass
Admin	admin	admin
User	cst8277	8277

Running the Skeleton

Unzip the skeleton project provided and then open it with your Eclipse. **Do not put your code in any shared drive like OneDrive.** This project will make many files in the background and having it synched will be major problem.

When running your code, you might be getting some errors that make no sense. Like Eclipse saying you have to import while it is already imported. You can do the following to help the situation. You might have to do one or all of these steps many times during your project:

- Go to Project → Clean.
- Update your Maven project. Right-click Project/Maven/Update.
- Clean and build your Maven project.
- Remove the application from Payara and run it again.
- Restart your Payara server.
- Manually delete the target folder in your project.

Requirement Summary

1. Make a resource for all tables.
    a. Physician is given as an example.
    b. Each resource needs to support CRUD operations.
    c. Resource is **not** needed for security_user, security_role, and user_has_role.
    d. Resource is needed for the following classes/entities:
        i. MedicalTraining
        ii. Patient
        iii. Prescription
        iv. MedicalCertificate
        v. Medicine
        vi. Physician
        vii. MedicalSchool
    e. Import and use the provided Postman collection **REST-ACMEMedical-Sample.postman_collection.json** to test your REST endpoints. (**Note:** Feel free to add and modify the REST requests/messages. Please include **only** those REST requests that are working properly for your REST API system and remove those REST requests that are not working or not supported by your REST API system. When submitting your completed work, please include **only** those REST requests that are working properly for your REST API system and remove those REST requests that are not working or not supported by your REST API system. Your work will be graded based on the REST requests included in your Postman collection.)
2. Update your entities with appropriate Jackson annotations similar to Lab 4.
    a. Examples of all you need is in the code already.
    b. Use @JsonIgnore if you need to remove a field from being processed by Jackson. For example, medical school does not need to display all the medical trainings.
    c. Use @JsonSerialize if you like to create a custom serialization of your entity. For example, when creating JSON for medical school we just need to see the count of medical trainings.
    d. If you need access to your lazy fetched objects, you need to create a namedQuery with left join fetch. For example, we need all medical schools with their medical training counts. "SELECT distinct ms FROM MedicalSchool ms **LEFT JOIN FETCH ms.medicalTrainings**".
    e. What exactly needs to be displayed is up to you. Display enough meaningful information in your JSON.
3. Create JUnit tests for your REST APIs.
    a. Minimum of 50 JUnit tests.
    b. Use the Client API to test your code.
    c. Remember to run your application on the server first before running any JUnit tests.
    d. Tests the roles and CRUD operations.
    e. Generate a **Maven surefire report** that summarizes your test results.
        i. To build your project and run all of the test cases, right-click on your project, choose **Run As**, select **Maven build…**, on the **Goals** text box type:
        clean install test surefire-report:report site -DgenerateReports=true
        ii. The above will generate *target/site/surefire-report.html* file which will show the results of the test cases when opened with a web browser

IMPORTANT: Please create a **ReadMe.doc** inside of your project and put the names of all members of the group. Please also specify how the work was divided among the members of the group (that is, who did what). Finally, the entire group should grade each individual member of the group based on the member's contribution to the group work. For example, the below table shows the contribution of each member and how the entire group has graded each member of the group (please see next page):

Member	**Contributions**	**Average Peer Grade (Grade Provided by Group #1)**
John Smith	Wrote 15 JUnit tests, absent from a lot of group meetings, …	60%
Mary Lee	Completed Security User & Roles annotations + wrote 24 JUnit tests, acts as group leader, …	100%
Martha Snow	Wrote Physician, Medicine, MedicalSchool REST resources	90%
Emmanuel Gray	Wrote MedicalCertificate, MedicalTraining resources	80%

Submit a zip file containing your entire project folder to Brightspace. You should name your zip file as: <Lastname1>-<Lastname2>-<Lastname3>-<Lastname4>-Assignment4.zip. For example: **Gray-Lee-Smith-Snow-Assignment4.zip**. Submit only one zip file under your group number on Brightspace.

Finally, please note that your group needs to demo your project to your respective lab professor in order to receive credit for this assignment. That is, no demo would result in a grade of 0 for your group project.

– end –