LAB 14: QUARKUS SECURE JWT

Autor: José Díaz

Github Repo: https://github.com/joedayz/quarkus-bcp-2025.git

Abre el proyecto **secure-jwt-start**.

Instructions

1. Open the expenses application.

1.1. Navigate to the `~/D0378/secure-jwt` directory.

```
[student@workstation ~]$ cd ~/D0378/secure-jwt
```

1.2. Open the project with an editor, such as VSCodium or vim.

```
[student@workstation secure-jwt]$ codium .
```

2. Explore the code.

2.1. Inspect the `JwtResource` class. The `/jwt/{username}` endpoint generates a JWT for a given user.

> **Important**
> This endpoint does not require a password for the sake of simplicity. In production use cases, do not issue JWTs without authenticating users first.

2.2. Inspect the `UserResource` class. The `/user/expenses` endpoint returns the expenses of the authenticated user.

2.3. Inspect the `AdminResource` class. The `/admin/expenses` endpoint lists all the expenses. Only users with the `ADMIN` role should have permissions to use this endpoint.

2.4. Inspect the `src/main/resources/application.properties` file. The file configures the properties required to build JWTs by using the SmallRye JWT generation API.

2.5. Inspect the `JwtGeneratorTest` class. This test class verifies the groups assigned to the JWTs of regular users and administrators. Regular users must belong to the `USER` group. Administrators must belong to both the `USER` and `ADMIN` groups. This class also verifies that the JWTs for regular users include the `iss`, `sub`, `upn`, `aud`, and `locale` claims.

2.6. Inspect the `UserResourceTest` class. This test class verifies that the `/user/expenses` endpoint is secured. Unauthenticated users must not be able to access the `/user/expenses` endpoint. Authenticated regular users must get a list of their expenses.

2.7. Inspect the `AdminResourceTest` class. This test class verifies that the `/admin/expenses` endpoint is secured. Unauthenticated and regular users must not be able to access the `/admin/expenses` endpoint. Administrators must get a list of all expenses.

3. Run all the tests and verify that nine of them fail.

```
[student@workstation secure-jwt]$ mvn test
...output omitted...
[ERROR]  AdminResourceTest.guestsCannotListExpenses:26 1 expectation failed.
Expected status code <401> but was <200>.

[ERROR]  AdminResourceTest.regularUsersCannotListExpenses:39 1 expectation failed.
Expected status code <403> but was <200>.

[ERROR]  UserResourceTest.guestsCannotListExpenses:26 1 expectation failed.
Expected status code <401> but was <200>.

[ERROR]  JwtGeneratorTest.adminJwtBelongsToAdminGroup:83 JWT groups for admin do
 not contain ADMIN ...
[ERROR]  JwtGeneratorTest.adminJwtBelongsToUserGroup:74 JWT groups for admin do
 not contain USER ...
[ERROR]  JwtGeneratorTest.userJwtBelongsToUserGroup:29 JWT groups for regular user
 do not contain USER ...
[ERROR]  JwtGeneratorTest.userJwtContainsAudienceClaim:74 JWT 'aud' claim not set
 as expected ...
[ERROR]  JwtGeneratorTest.userJwtContainsLocaleClaim:65 JWT 'locale' claim not set
 as expected ...
[ERROR]  JwtGeneratorTest.userJwtContainsSubjectClaim:38 JWT 'sub' claim not set
 as expected ...

[ERROR] Tests run: 13, Failures: 9, Errors: 0, Skipped: 0
...output omitted...
```

4. Complete the code to create JWTs that contain the required claims.

4.1. Open the `JwtGenerator` class and modify the `generateJwtForRegularUser` method. This method generates JWTs for regular users. Add the `sub`, `aud`, and `locale` claims, as follows:

```
public static String generateJwtForRegularUser(String username) {
    return Jwt.issuer( ISSUER )
            .upn( username + "@example.com" )
            .subject(username)
            .audience("expenses.example.com")
            .claim("locale", "en_US")
            .sign();
}
```

4.2. Complete the `generateJwtForRegularUser` method by adding a group for regular users. Set the `groups` claim to contain the `USER` group.

```
public static String generateJwtForRegularUser(String username) {
    return Jwt.issuer( ISSUER )
            .upn( username + "@example.com" )
            .subject(username)
            .audience("expenses.example.com")
            .claim("locale", "en_US")
            .groups( new HashSet<>( Arrays.asList( "USER" ) ) )
            .sign();
}
```

4.3. In the same class, complete the `generateJwtForAdmin` method. This method generates JWT for administrators. Set the groups claim to contain the `USER` and `ADMIN` groups.

```
public static String generateJwtForAdmin(String username) {
    return Jwt.issuer( ISSUER )
            .upn( username + "@example.com" )
            .subject( username )
            .claim( "locale", "en_US" )
            .groups( new HashSet<>( Arrays.asList( "USER", "ADMIN" ) ) )
            .sign();
}
```

4.4. Run the tests in the `JwtGeneratorTest` class and verify that eight tests pass.

```
[student@workstation secure-jwt]$ mvn test -Dtest="JwtGeneratorTest"
...output omitted...
[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0
...output omitted...
```

5. Secure the `/user/expenses` endpoint. Allow access to the `USER` role.

5.1. Annotate the `UserResource` class to secure its endpoints. Restrict the access to the `USER` role.

```
@Path( "/user" )
@RolesAllowed({ "USER" })
public class UserResource {
    ...implementation ommitted...
}
```

5.2. Run the tests in the `UserResourceTest` class and verify that two tests pass.

```
[student@workstation secure-jwt]$ mvn test -Dtest="UserResourceTest"
...output omitted...
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
...output omitted...
```

6. Secure the `/admin/expenses` endpoint. Allow access to the `ADMIN` role.

6.1. Annotate the `AdminResource` class and secure its endpoints. Restrict access to the `ADMIN` role.

```
@Path( "/admin" )
@RolesAllowed({ "ADMIN" })
public class AdminResource {
    ...implementation ommitted...
}
```

6.2. Run the tests in the `AdminResourceTest` class and verify that three tests pass.

```
[student@workstation secure-jwt]$ mvn test -Dtest="AdminResourceTest"
...output omitted...
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0
...output omitted...
```

7. Verify that all tests pass.

```
[student@workstation secure-jwt]$ mvn test
...output omitted...
[INFO] Tests run: 13, Failures: 0, Errors: 0, Skipped: 0
...output omitted...
```

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish secure-jwt
```

This concludes the section.

Solución:

enjoy!

Jose