Project Assignment
Paynet REST API Security workshop Oct - Nov 2021

1 Project outline

You are required to implement a security architecture to authorize and authenticate users accessing the REST service that you developed in your first assignment. There are basically 2 parts to this project:

- Developing a Spring MVC app that will allow users to register an account with the system in and then sign in using a customized login form. This app will also allow users to request a new JWT that will provide different levels of access to the REST service.

- Securing the existing REST service from the first assignment, so that it validates a JWT placed in the Authorization Bearer header of incoming requests. From this JWT, it determines the identity of the sender as well as specific resources of the REST Service that a given user is allowed to access.

2 Functional specification

2.1 Registration system MVC app

This is a Spring MVC app that can be implemented using either JSP or a templating engine such as Thymeleaf.

A backend database table will be provided to persist user info with the following fields: `first name, last name, email, password, roleMVC, roleREST`. This can be done using Spring Data JPA with a MySQL table

The app home page should provide a means for selecting:
- Register new account
- Login with existing account or social media

On the register new account page, provide appropriate fields to obtain the following items from a new user: `first name, last name, email, password`
Basic validation can be performed: ensuring that email is in a valid format, password is correct (i.e. providing a field to retype password and compare with original version, etc).

When the information is validated, it is saved to the backend database table.
- The `roleMVC` is used to designate the role of this user account for the purposes of authorization in the Spring Security framework for accessing resources in this particular app. There are two possible values: USER and ADMIN. By default, it is set to USER.

- The `roleREST` is used to designate the role of this user account for the purposes of authorization in the Spring Security framework for accessing resources in the REST service. There are two possible values: BASIC and FULL. By default, it is set to BASIC.

If an existing user account already exists, an error is flagged and the user is requested to enter details again. User accounts are differentiated from each other based on their `email` fields: so it is possible to have two unique accounts with the same `first name` and `last name` as long as their `email` fields are different.

At the customized login page, the user is provided an option to login using an existing registered account (where the username used for login is the `email`).

If the user provides an email that does not match any email field in the existing user accounts in the backend database table, she is warned that her account does not exist and is provided an option to register a new account. If on the other hand, the email is valid but the password is incorrect, the user is redirected back again to the customized login page to login again with a suitable customized error message.

The customized login page also provides an option to login via any other OAuth2 / OIDC provider of your choice (e.g. Google, GitHub, Facebook, etc). If this option is selected, the email of the user registered with that particular OAuth2 / OIDC provider (which will be available at the end of the OAuth2 workflow) is used to create a new user account on the backend database table (if there isn't already an existing one).

Once a user is successfully logged in, they have an option to request a new JWT to access the REST service. The following are the standard and custom claims to be provided in the new JWT.

```
{

  "iat": time at which JWT was issued
  "exp": expiration time for JWT
  "sub": email address of user
  "iss": marveluniverse.com.my
  "role": roleREST from the user account

}
```

The JWT is signed with a secret key stored in a keystore that is made available to the Spring MVC app. This same keystore will also be made available to the Spring REST service

The JWT is displayed on a particular page from which the user can copy and paste it to a separate text editor to be used later.

There are two specific roles (stored in `roleMVC`) available for enforcing access control in this app: USER and ADMIN. All users creating a new account (or authenticating via an OIDC / OAuth2 provider) are assigned the USER role by default.

One (or more) ADMIN accounts are pre-created by default in the user database table (you can hardcode them in the start-up logic of the app).

ADMIN accounts are allowed to perform the following:

- Delete an existing normal user account

- View the info of all existing user accounts (Note that since passwords are stored encoded in the database table, the ADMIN will only be able to view this encoded version and not the actual plaintext password)
- Assign a new value to the `roleREST` in any of the user accounts. The other possible value (besides the default of BASIC) is FULL.
- Set a new expiration date for all JWTs issued from the app (for e.g. expiration will be 1 day from the time that the JWT was issued)

All of these functionalities can be either provided in:

- separate views (JSP or Thymeleaf templates) that are only accessible in this app to the ADMIN user
- same views that are accessible to all users, but which display different content depending on the role of the authenticated user

Implement HTTPS (standard server-side authentication only) to protect all communication between the browser and the MVC app. Generate all the necessary keystore keys and certificates to implement this.

2.2 Securing the REST service

The REST API service that you created in your first project will be extended to incorporate the necessary authentication / authorization abilities using the JWT that the user obtained from the registration system. This API service can be implemented as a separate project running on a different embedded Tomcat server from the registration system MVC app.

The user will arrange for the submission of the JWT in the Authorization Bearer header of all requests sent to the REST service. This can be accomplished either by:

- Using the Postman client to manually create the HTTP requests and configuring the JWT in the outgoing request accordingly
- Creating a REST Template client that programmatically includes the JWT in the Authorization Bearer header

You can implement both approaches for more practice, or use only the Postman client, if you lack time.

The REST service will now incorporate the Spring Security framework functionality to:

- Extract the JWT from the Authorization header of the incoming request
- Determine whether the JWT is valid (i.e. that the signature on it is valid and the expiration date has not yet been surpassed)
- Enforce access control based on the value of the `role` claim in the JWT (this contains the `roleREST` from the user account which has two possible values: BASIC and FULL.

The REST service that you created will have a certain number of API endpoints. At your own discretion, enforce authorization constraints so that only a certain number of API endpoints are available for access to users with a JWT with a `role` claim of BASIC, while all other API endpoints are available to users a JWT with a `role` claim of FULL.

An example is as follows:

Method	Sample endpoints	Access available to
GET	`/api/shops`	BASIC, FULL
GET	`/api/shops/{id}`	BASIC, FULL
POST	`/api/shops`	BASIC, FULL
PUT	`/api/shops/{id}`	FULL only
DELETE	`/api/shops/{id}`	FULL only
GET	`/api/shops?key1=val1`	BASIC, FULL
GET	`/api/shops?limit=X&page=Y`	BASIC, FULL

Optionally, include appropriate custom error messages to be returned to the client if

- no JWT is available in the Authorization header of an incoming request
- the JWT is invalid (signature cannot be verified or JWT has expired)
- a request is being made to an API endpoint which the user's JWT `role` claim is not permitted to access

If you have time, you can optionally consider implementing more interesting access control scenarios, for e.g.

a) If your REST API exposes information about countries, you could include a `location` claim in the user JWT. Then based on the value of this claim, you could filter the set of results returned from a GET request that should return info about all countries.

b) You could log all the different API endpoint requests that the user made and store it in the user's account (in the registration system) for viewing and analysis by the ADMIN account. This is part of auditing, which is an important third component to supplement Authentication and Authorization and is also necessary if billing is to performed on the number or frequency of requests such as in the case of commercial API platforms such as RapidAPI.

3 Submission requirements

You can submit this project either by:

- Uploading a zip of your Spring project folders (in the event you are using different projects for your client and server) to a file-sharing cloud service like Google Drive or DropBox and email me me the link to download
- Putting your code up on a GitHub code repo and emailing me the link to it

You should provide basic documentation on your project with information on:

- A high level overview of the particular approach that you have taken to implement the relevant security functionality (a description of between 0.5 - 1 page is adequate). This can be included as a Microsoft Word document or text file in the root of your Spring Boot project folder, or emailed to me separately.
- Any additional info required for me to configure and start the apps properly. For e.g. any keystores I need to generate (in the event you have not placed your keystore in your project folder), any API keys for remote REST services that may need to be placed in the `application.properties` for the app to run correctly, . Otherwise, I will just assume I can import your project as a Maven project into STS and then run it directly from the Boot Dashboard without any further modifications.