

SOEN 487: Web Services and Applications

Lab Instructions

Token Authentication for JAX-RS

March 8, 2022

1 General Information

Lab Date: Wednesday, March 9th, 2022.

Your lab instructor will provide you with instructions on how to do this lab activity.

2 Introduction

The purpose of this lab is implement token authentication on an existing JAX-RS API. This lab will be performed individually, and at the end, you will demonstrate your results as outlined, to your lab instructor.

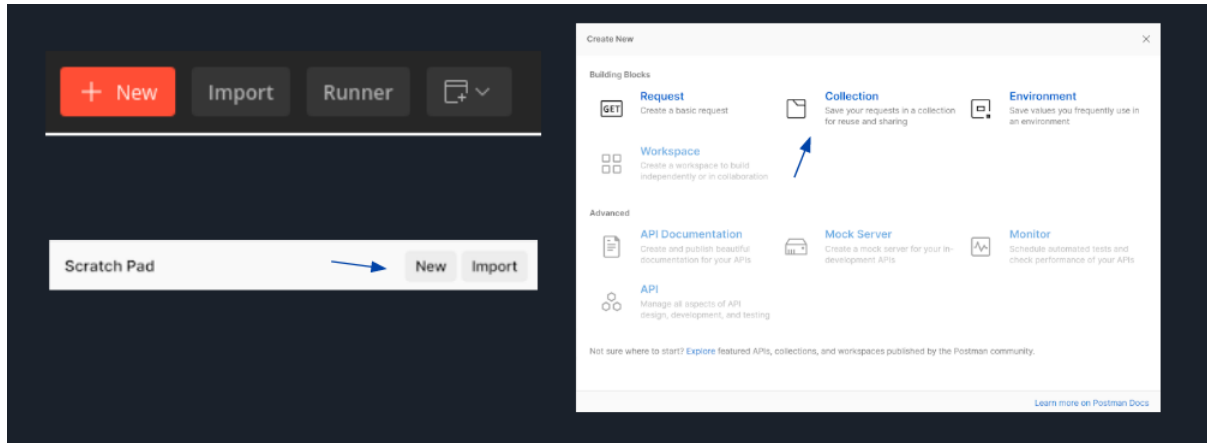
Note: To complete this lab, you must have completed Lab03.

3 Project Setup

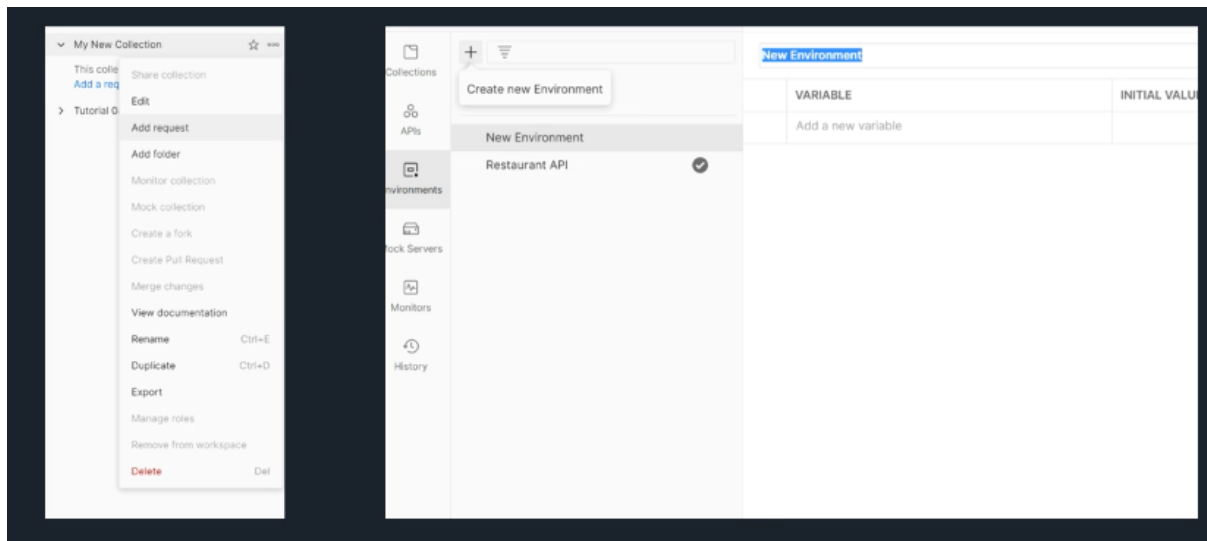
To get started with this lab, simply open your maven project from your third lab.

4 Postman Collections

To create a Postman collection, simply click on new from your Scratch Pad or Workspace on the top left and select Collection.



You can then then name your collection and start adding requests. You should also create an Environment which will hold important variables.



5 Creating SSL Certificate

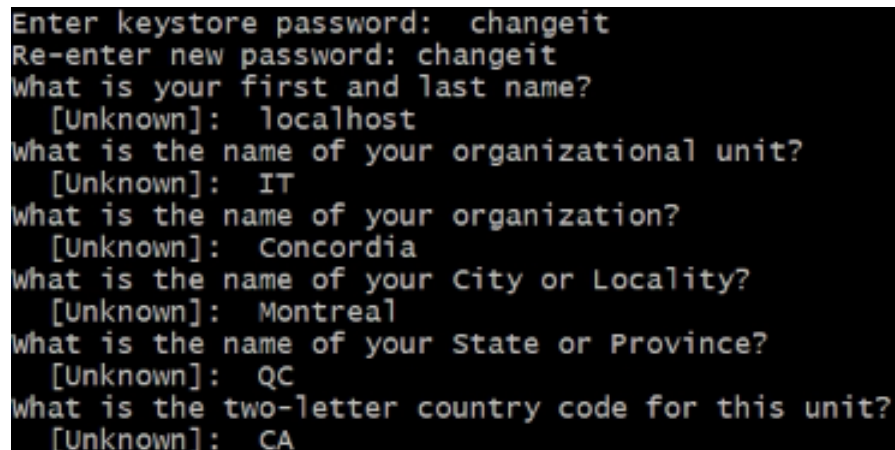
We will be using Java's keytool to create the self signed certificates. Make sure the JDK bin directory is included in your Path. For example:

```
C:\Program Files\Java\jdk1.8.0_281\bin
```

In the project directory, open Git Bash or terminal on MacOS and run the following command:

```
keytool -genkey -keyalg RSA -keystore localhost.jks -alias localhost -ext  
SAN=dns:localhost
```

Here is an example of what to enter for the questions asked:



```
Enter keystore password: changeit  
Re-enter new password: changeit  
What is your first and last name?  
[Unknown]: localhost  
What is the name of your organizational unit?  
[Unknown]: IT  
What is the name of your organization?  
[Unknown]: Concordia  
What is the name of your City or Locality?  
[Unknown]: Montreal  
What is the name of your State or Province?  
[Unknown]: QC  
What is the two-letter country code for this unit?  
[Unknown]: CA
```

Note: Make sure you use the same password in the code as well.

After the localhost.jks file has been generated, run the following two commands, to generate public keys for chrome and curl:

```
keytool -importkeystore -srckeystore localhost.jks -destkeystore localhost.p12  
-srcstoretype JKS -deststoretype PKCS12 -srcstorepass changeit -deststorepass  
changeit -srcalias localhost -destalias localhost -srckeypass changeit  
-destkeypass changeit
```

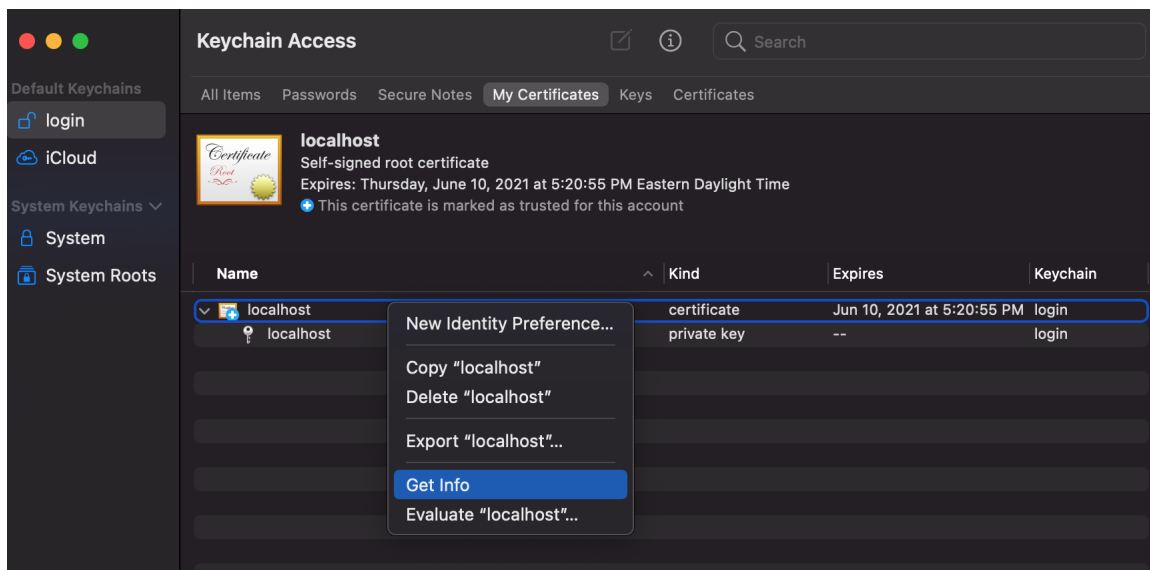
Gitbash (Win):

```
winpty openssl pkcs12 -in localhost.p12 -clcerts -nokeys -out localhost.crt
```

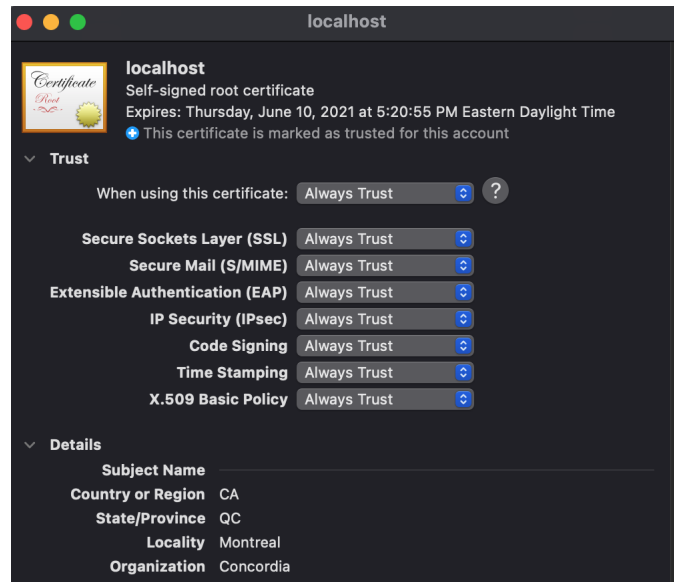
MacOS:

```
openssl pkcs12 -in localhost.p12 -clcerts -nokeys -out localhost.crt
```

To add the p12 public key to Chrome, look for Manage Certificates in the settings and import it as part of the Trusted Root Certification Authorities. On MacOS you will need to add it to the keychain and make the OS trust the key. To do this go to keychain Access, then Go to My Certificates, Right click on the certificate and go to Get Info:



Then, click on Trust to expand the trust options and then change "When using this certificate:" to Always Trust and click save:



Finally, you will have to make some changes for grizzly to publish the server using https.

Take a look at the code in the Main class of the finished branch of tutorial 8:

```
https://github.com/SOEN487/T07/tree/finished/src/main/java/com/example/rest/Main.java
```

To test with curl, you may need to add the `--cacert` option, eg:

```
curl --cacert localhost.crt https://localhost:8443/restaurant/customerform
```

6 Instructions

- Create a User POJO class that has three class variables: **username**, **password**¹, **token** and implement a token generation method similar to the tutorial, to use for login.
- Create a UserRest class that implements four REST endpoints:
 1. createUser²
 2. login
 3. logout
 4. validateToken
- Update the existing **Bookstore** REST implementation so that any methods that modify the API expect an authentication token and validate it using the UserRest API.
- Run the API and check that user creation, login and logout work properly and that all methods that modify the API check for the token. You can test using curl, Postman or Java.
- Generate keys as shown in the tutorial and alter your API to work with https.

The **UserRest** class should hold an array with the users and a data structure of your choice to store the tokens and their duration. Tokens should last for **30 minutes** after which time if a request is sent, it should be denied and the token should be deleted. You may copy the **MyResponse** class from the tutorial to simplify reading REST responses.

Note: createUser is currently an **insecure**³ method because it allows public registration. Normally we would protect this method with email validation or only allow admins to register.

¹See best practices on how to secure and protect password

²See the note below

³For simplicity the createUser has been provided without any security precautions so that it can easily be implemented during the lab time. Please note that this is a bad practice and must be avoided in real applications. Normally createUser may only be provided to 1) **admins** only, or 2) available via a self registration process protected by an approval process (i.e. via email verification, or similar identification).

Bonus

- Implement two types of token timeouts: The primary one checks the entire duration of the token (when created) whereas the second one check when the token was last accessed. For example, a token should last for a total of **30 minutes**, but if no request is made for **1 minute**⁴, the token should be considered expired.

References

1. Token Authentication Info

<https://www.okta.com/identity-101/what-is-token-based-authentication>

⁴This 1 minute timeout is to facilitate the testing purposes and in reality is normally set to a much greater value