

# Cloud Native Training

## Lab Environment Setup V1

System Requirements:

OS

1. Windows 10 or above
2. Mac OS 10.15 or above

Hardware requirement

1. CPU i5 or above
2. RAM 8GB or above
3. Hard Disk 80GB free or above

Software List

1. JDK 17 or above
2. Apache Maven 3.8.3 or above
3. IDE ( Visual Studio Code )
4. [Pre-Lab] Quarkus Sample Project Hello World
5. (Optional) VMware Player or Virtual Box ( for Kubernetes Installation ) for Ubuntu 18.04.5 (x2 VMs)
5. (Optional) Script for install Kubernetes Cluster(1 Master Node, 1 Workder Node) on Ubuntu 18.04.5

Prepared by :

Felix Tsang  
felix@linux.com

Lab List

## Table of Contents

<b>Environment Setup.....</b>	<b>2</b>
<b>1) JDK Installation .....</b>	<b>2</b>
<b>2) Maven Installation .....</b>	<b>2</b>
<b>3) IDE ( Visual Studio Code ) .....</b>	<b>3</b>
<b>Lab0 Quarkus Sample Project Hello World.....</b>	<b>5</b>
<b>LAB1 URL Parameter .....</b>	<b>8</b>
<b>LAB2 Response with JSON Obj.....</b>	<b>9</b>
<b>LAB3 HTTP Post Consume JSON Obj.....</b>	<b>14</b>
<b>LAB4 JWT .....</b>	<b>15</b>
<b>Appendix.....</b>	<b>21</b>

## Environment Setup

### 1) JDK Installation

**For Mac** (Detailed Procedures ref “Appendix I JDK Installation”)

Install JDK 17 or above by dmg installer

<https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html>

java -version

java version "17.0.1" 2021-10-19 LTS

Java(TM) SE Runtime Environment (build 17.0.1+12-LTS-39)

Java HotSpot(TM) 64-Bit Server VM (build 17.0.1+12-LTS-39, mixed mode, sharing)

**For Windows** (Detailed Procedures ref “Appendix I JDK Installation”)

Install JDK 17

<https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html>

Set Environment Variable

java -version or above

java version "17.0.1" 2021-10-19 LTS

Java(TM) SE Runtime Environment (build 17.0.1+12-LTS-39)

Java HotSpot(TM) 64-Bit Server VM (build 17.0.1+12-LTS-39, mixed mode, sharing)

### 2) Maven Installation

Install Maven (Detailed Procedures ref “Appendix II Maven Installation” )

The installation process of Apache Maven is quite simple, we just need to extract the Maven's zip file and set up maven environment variables.

**For MAC , Install brew**

/bin/bash -c "\$(curl -fsSL

<https://raw.githubusercontent.com/Homebrew/install/master/install.sh>"

brew install maven

mvn -v

Apache Maven 3.8.3 (ff8e977a158738155dc465c6a97ffaf31982d739)

Maven home: /usr/local/Cellar/maven/3.8.3/libexec

Java version: 17.0.1, vendor: Oracle Corporation, runtime:

/Library/Java/JavaVirtualMachines/jdk-17.0.1.jdk/Contents/Home

Default locale: en\_HK, platform encoding: UTF-8

OS name: "mac os x", version: "10.15.7", arch: "x86\_64", family: "mac"

```

admins-iMac:~ admin$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
==> Checking for 'sudo' access (which may request your password)...
Password:
==> This script will install:
/usr/local/bin/brew
/usr/local/share/doc/homebrew
/usr/local/share/man/man1/brew.1
/usr/local/share/zsh/site-functions/_brew
/usr/local/etc/bash_completion.d/brew
/usr/local/Homebrew
==> The Xcode Command Line Tools will be installed.

Press RETURN to continue or any other key to abort:
==> /usr/bin/sudo /usr/sbin/chown -R admin:admin /usr/local/Homebrew
==> Searching online for the Command Line Tools
==> /usr/bin/sudo /usr/bin/touch /tmp/.com.apple.dt.CommandLineTools.installondemand.in-progress
==> Installing Command Line Tools for Xcode-13.2
==> /usr/bin/sudo /usr/sbin/softwareupdate -i Command\ Line\ Tools\ for\ Xcode-13.2
Software Update Tool

Finding available software

Downloading Command Line Tools for Xcode

```

## For Windows

Maven needs JDK to be installed before configuration.

1.Finish JDK setup and set up JAVA\_HOME **Environment Variable** in the previous steps

2.Download Maven Zip and unzip to your working folder

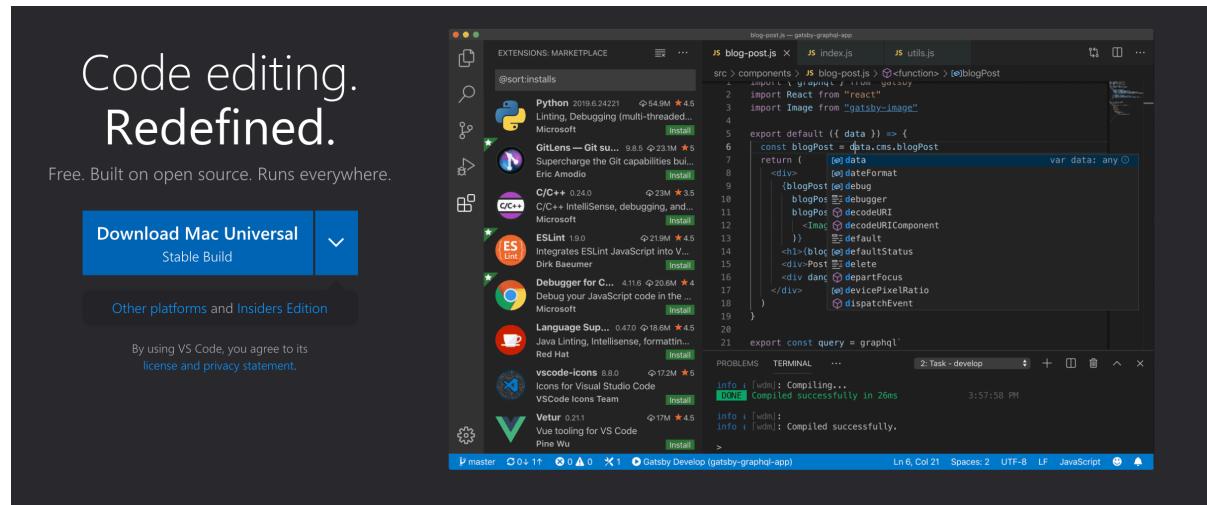
3.Add MAVEN\_HOME Environment Variable

Reference Steps on Appendix Appendix I

## 3) IDE ( Visual Studio Code )

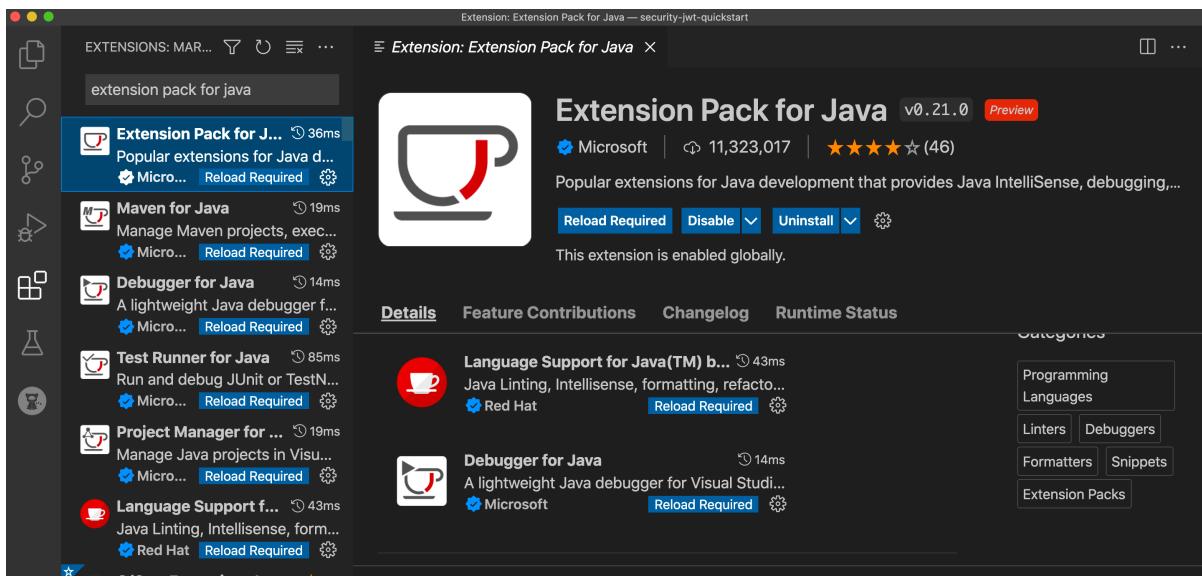
**Download and complete the installation**

<https://code.visualstudio.com/>



## Extension Pack for Java v0.18.6 or above

Navigate to the menu on left hand side and search “extension pack for java” , then click install on the right panel



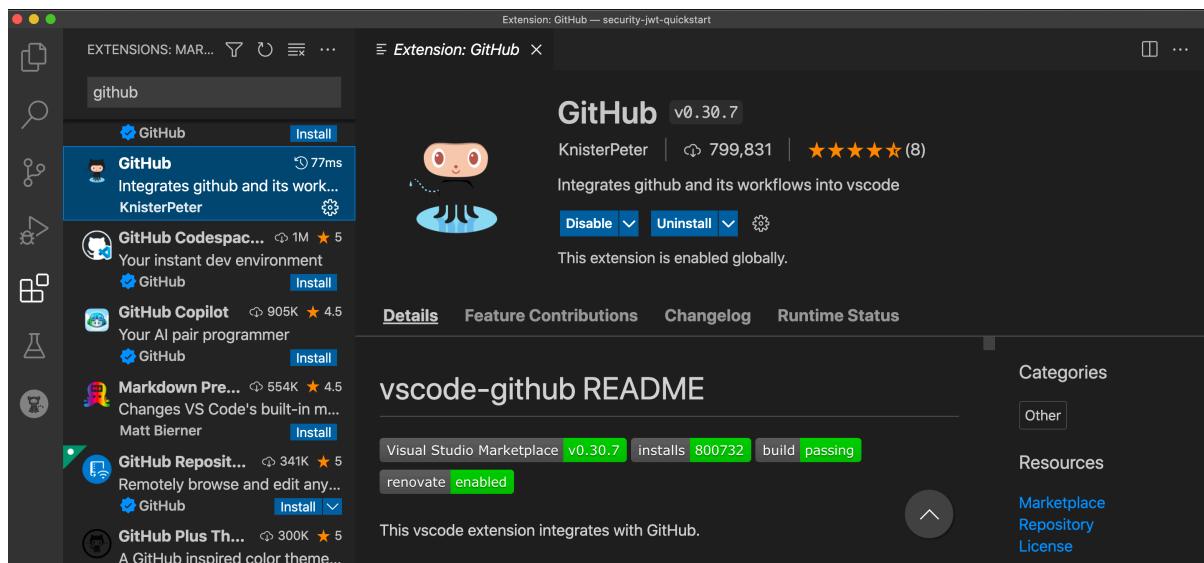
## Install Quarkus Tools for Visual Studio Code

Again, to search “Quarkus” under extension and install the quarkus extation as shown below.



## Install Github for Visual Studio Code

search “Quarkus” under extension and install the quarkus extation as shown below.



## Lab0 Quarkus Sample Project Hello World

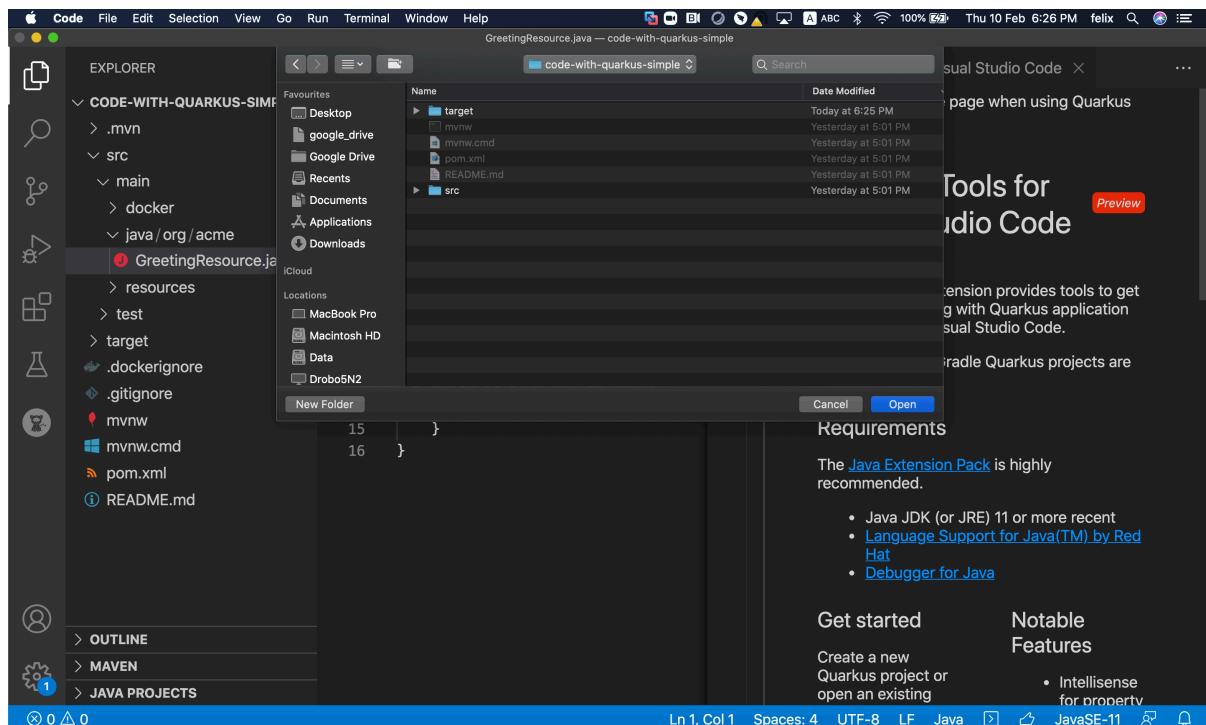
Download Link:

<https://code.quarkus.io/>

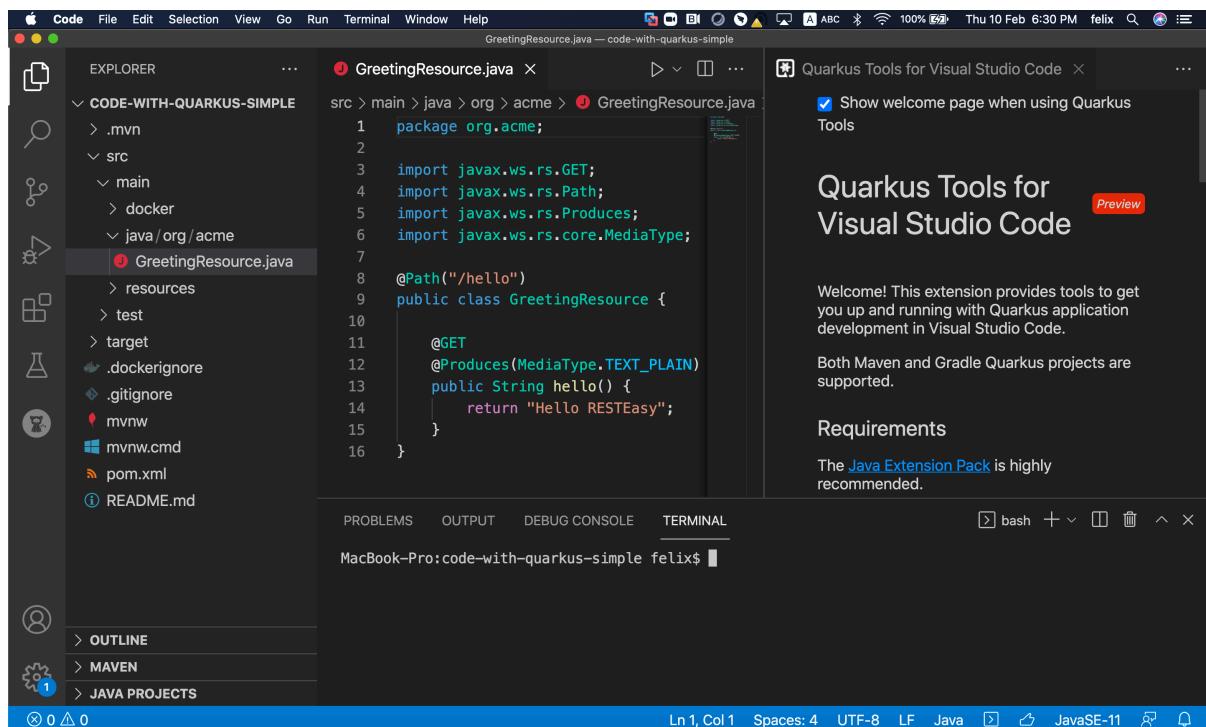
download zip and unzip to working folder

The screenshot shows the Quarkus Platform configuration interface. At the top, there's a header with the Quarkus logo and the text '2.7 io.quarkus.platform'. To the right of the header are links for 'Back to quarkus.io' and 'Available with Enterprise Support'. Below the header, there's a section titled 'CONFIGURE YOUR APPLICATION' with fields for 'Group' (set to 'org.acme'), 'Artifact' (set to 'code-with-quarkus'), and 'Build Tool' (set to 'Maven'). There's also a 'MORE OPTIONS' button and a 'Filters' dropdown with a search bar containing 'origin:platform'. On the right side of the interface, there are three main buttons: 'Generate your application (⌘ + ⌫)' (highlighted in blue), 'Download as a zip', and 'Push to GitHub'. Below these buttons, there's a section titled 'Web' with several expandable options: 'RESTEasy JAX-RS [quarkus-resteasy]' (selected), 'RESTEasy Jackson [quarkus-resteasy-jackson]', 'RESTEasyJSON-B [quarkus-resteasy-jsonb]', and 'Eclipse Vert.x GraphQL [quarkus-vertx-graphql]'. Each option has a small 'Code' link next to it.

Open visual Studio Code IDE and open the folder



Open Terminal under the menu bar.

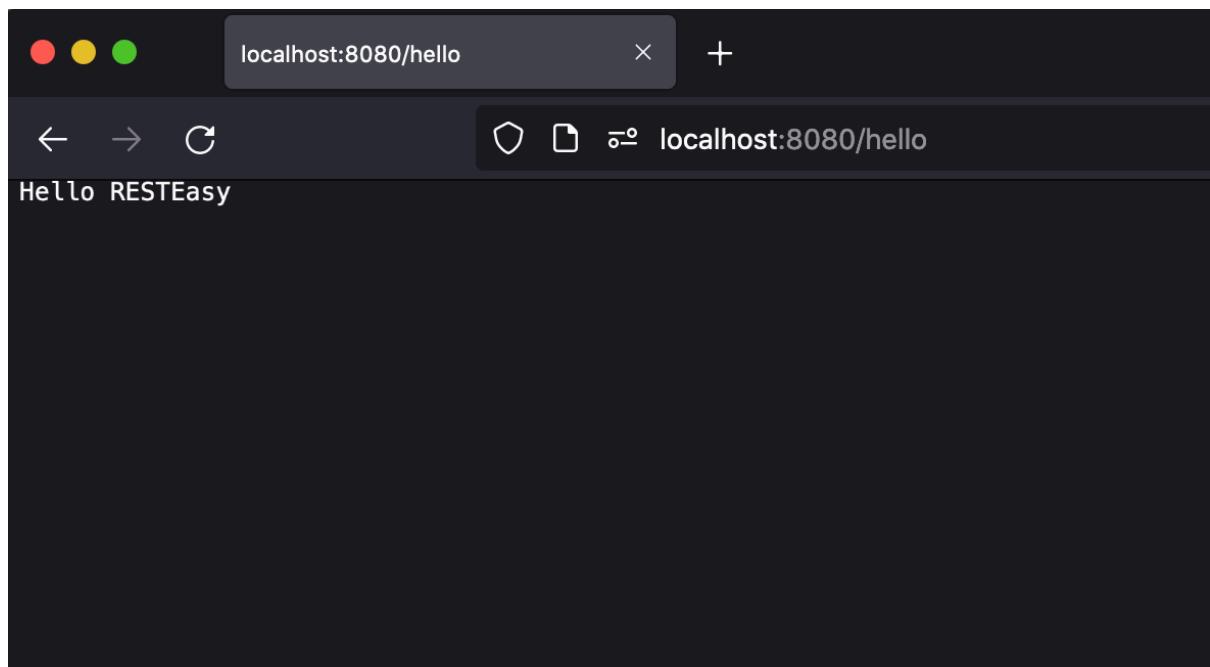


Run the command “mvn quarkus:dev”

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
MacBook-Pro:code-with-quarkus-simple felix$ mvn quarkus:dev
```

```
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 2 source files to /Volumes/Data/quarkus/code-with-quarkus-simple/target/test-classes
Listening for transport dt_socket at address: 5005
[INFO] [Quarkus Main Thread] code-with-quarkus 1.0.0-SNAPSHOT on JVM (powered by Quarkus 2.7.1.Final) started in 1.719s. Listening on: http://localhost:8080
[INFO] [Quarkus Main Thread] Profile dev activated. Live Coding activated.
[INFO] [Quarkus Main Thread] Installed features: [cdi, resteasy, smallrye-context-propagation, vertx]
-- Tests paused
Press [r] to resume testing, [o] Toggle test output, [:] for the terminal, [h] for more options>
```

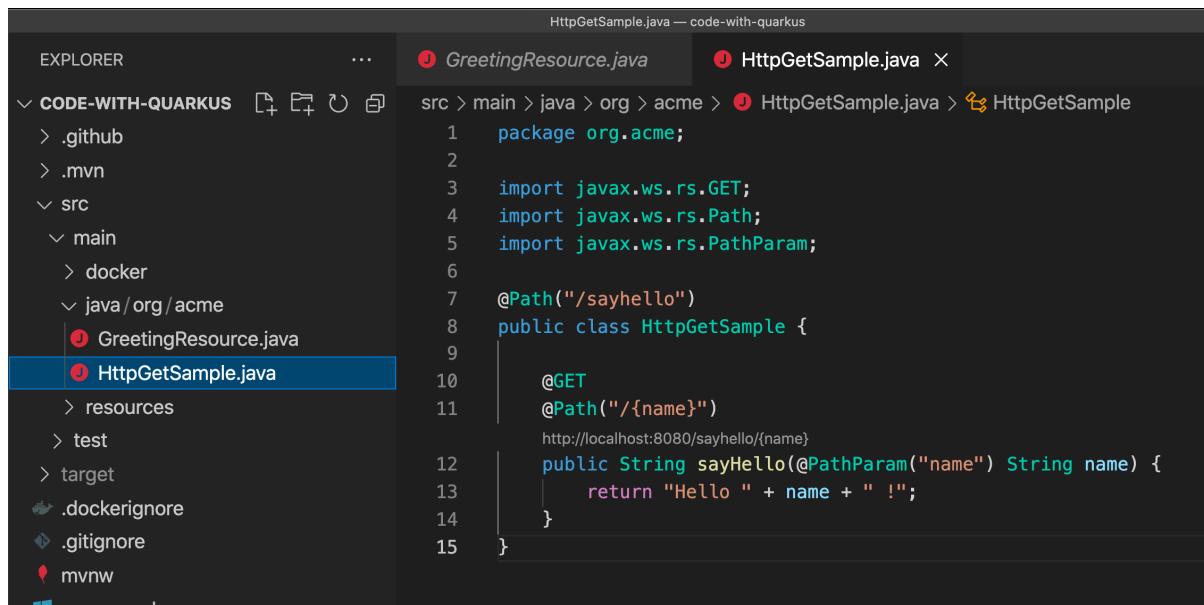
Test with browser : <http://localhost:8080/hello>



## LAB1 URL Parameter

Create an other class named “HttpGetSample”

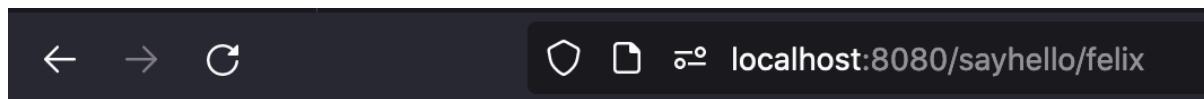
Use PathParam mapper to retrieve the URL parameter



The screenshot shows a code editor with the following structure:

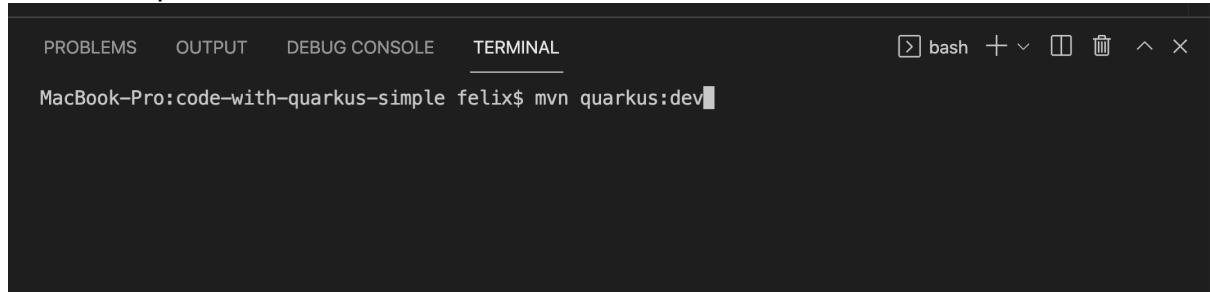
- EXPLORER**: Shows the project structure: CODE-WITH-QUARKUS, .github, .mvn, src, main, docker, java/org/acme (containing GreetingResource.java), and HttpGetSample.java.
- GreetingResource.java**: A Java file with annotations @Path("/sayhello") and @GET.
- HttpGetSample.java**: A Java file containing the following code:

```
1 package org.acme;
2
3 import javax.ws.rs.GET;
4 import javax.ws.rs.Path;
5 import javax.ws.rs.PathParam;
6
7 @Path("/sayhello")
8 public class HttpGetSample {
9
10    @GET
11    @Path("/{name}")
12    http://localhost:8080/sayhello/{name}
13    public String sayHello(@PathParam("name") String name) {
14        return "Hello " + name + " !";
15    }
}
```

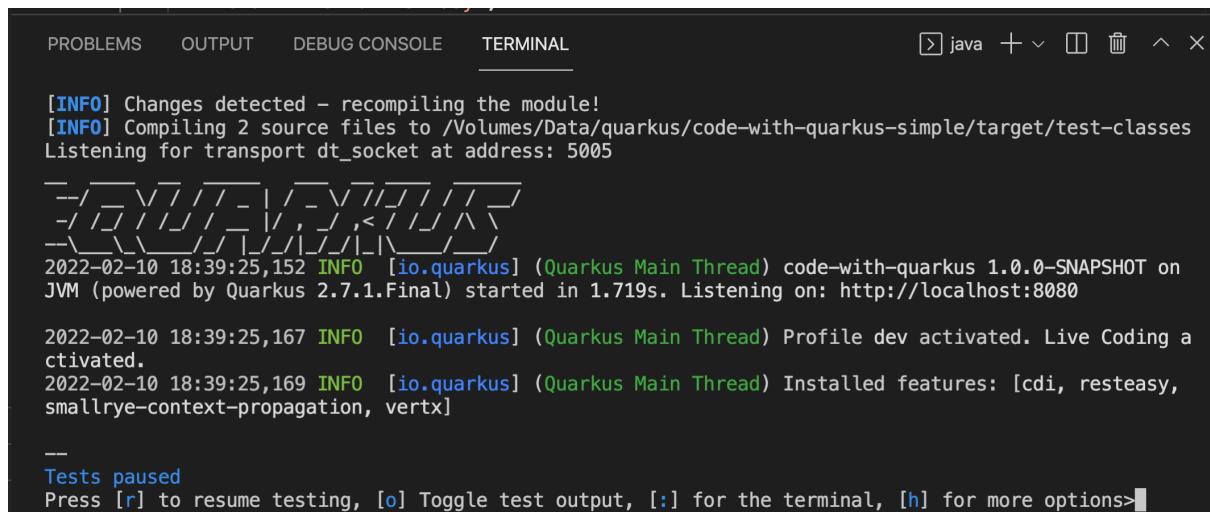


Hello felix !

Run mvn quarkus:dev

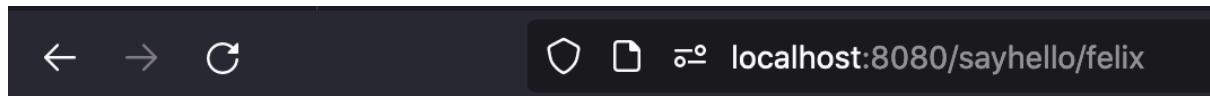


```
MacBook-Pro:code-with-quarkus-simple felix$ mvn quarkus:dev
```



```
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 2 source files to /Volumes/Data/quarkus/code-with-quarkus-simple/target/test-classes
Listening for transport dt_socket at address: 5005
--/--\v/\_/_/_|/_\v/_/_/_/_/
--/_/_/_/_/_/_\_,_,</_/_/\_\
--\_\_\_/_/_/_/_/_/_/_/_/_/_\_
2022-02-10 18:39:25,152 INFO [io.quarkus] (Quarkus Main Thread) code-with-quarkus 1.0.0-SNAPSHOT on
JVM (powered by Quarkus 2.7.1.Final) started in 1.719s. Listening on: http://localhost:8080
2022-02-10 18:39:25,167 INFO [io.quarkus] (Quarkus Main Thread) Profile dev activated. Live Coding a
ctivated.
2022-02-10 18:39:25,169 INFO [io.quarkus] (Quarkus Main Thread) Installed features: [cdi, resteasy,
smallrye-context-propagation, vertx]
--
Tests paused
Press [r] to resume testing, [o] Toggle test output, [:] for the terminal, [h] for more options>
```

Test from Brower locally

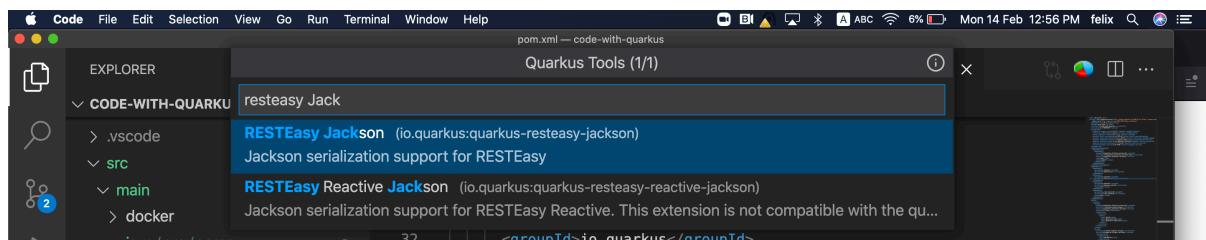
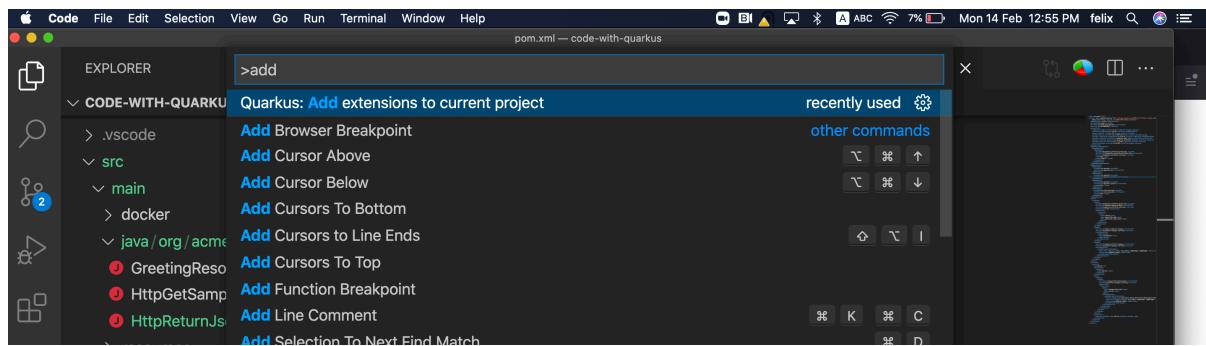


Hello felix !

## LAB2 Response with JSON Obj

Importing the RESTEasy/JAX-RS and [Jackson](#) extensions

Click “View” > “Command Palette”, type “add” and choose “Quarkus: Add extensions to current project”



and press enter to import the dependency.

Or

(Manual method) add the dependency

```

> test
> target
.dockerignore
.gitignore
mvnw
mvnw.cmd
pom.xml
      34   </dependency>
      35   <dependency>
      36     <groupId>io.quarkus</groupId>
      37     <artifactId>quarkus-resteasy</artifactId>
      38   </dependency>
      39   <dependency>
      40     <groupId>io.quarkus</groupId>
      41     <artifactId>quarkus-resteasy-jackson</artifactId>
      42   </dependency>

```

open pom.xml and add the dependency

```

<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-resteasy-jackson</artifactId>
</dependency>

```

Create a class named “Lab2HttpReturnJsonObj” and a Student class for the example of converting the Obj to JSON response.

Student.java

```

package org.acme;

public class Student {
    private String name;
    private String gender;
    private String year;

    public Student(String name, String gender , String year) {
        this.name = name;
        this.gender = gender;
    }
}

```

```

        this.year = year;
    }

    public String getGender() {
        return gender;
    }
    public String getYear() {
        return year;
    }
    public void setYear(String year) {
        this.year = year;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setGender(String gender) {
        this.gender = gender;
    }
}

```

### Lab2HttpReturnJsonObj.java

```

package org.acme;

import javax.enterprise.context.ApplicationScoped;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/getStudent")
@Produces(MediaType.APPLICATION_JSON)
@ApplicationScoped
public class Lab2HttpReturnJsonObj {
    @GET
    public Student sayHello() {
        Student s = new Student("M", "Felix T", "year 12");
        return s;
    }
}

```

The screenshot shows the VS Code interface with the following details:

- EXPLORER** sidebar: Shows the project structure under "CODE-WITH-QUARKUS". The "src/main/java/org/acme" folder contains several Java files: Lab0GreetingResource.java, Lab1HttpGetSample.java, Lab2HttpReturnJsonObj.java, Lab2HttpReturnJsonObjList.java, and Student.java. The "Student.java" file is currently selected.
- Editor Area**: The code for **Student.java** is displayed:

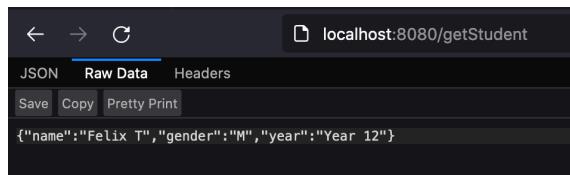
```
1 package org.acme;
2
3 public class Student {
4     private String name;
5     private String gender;
6     private String year;
7
8     public Student(String name, String gender , String year) {
9         this.name = name;
10        this.gender = gender;
11        this.year = year;
12    }
13
14    public String getGender() {
15        return gender;
16    }
17    public String getYear() {
18        return year;
19    }
20    public void setYear(String year) {
21        this.year = year;
22    }
23    public String getName() {
24        return name;
25    }
26    public void setName(String name) {
27        this.name = name;
28    }
29    public void setGender(String gender) {
30        this.gender = gender;
31    }
32}
33}
```

The screenshot shows the VS Code interface with the following details:

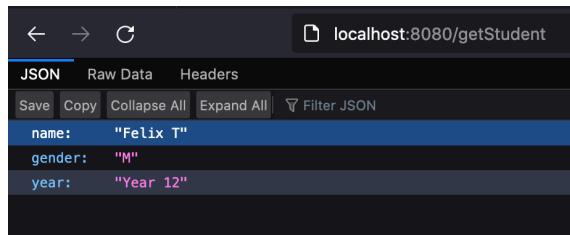
- EXPLORER** sidebar: Shows the project structure under "CODE-WITH-QUARKUS". The "src/main/java/org/acme" folder contains several Java files: Lab0GreetingResource.java, Lab1HttpGetSample.java, Lab2HttpReturnJsonObj.java, Lab2HttpReturnJsonObjList.java, and Student.java. The "Lab2HttpReturnJsonObj.java" file is currently selected.
- Editor Area**: The code for **Lab2HttpReturnJsonObj.java** is displayed:

```
1 package org.acme;
2
3 import javax.enterprise.context.ApplicationScoped;
4 import javax.ws.rs.GET;
5 import javax.ws.rs.Path;
6 import javax.ws.rs.Produces;
7 import javax.ws.rs.core.MediaType;
8
9 @Path("/getStudent")
10 @Produces(MediaType.APPLICATION_JSON)
11 @ApplicationScoped
12 public class Lab2HttpReturnJsonObj {
13     @GET
14     http://localhost:8080/getStudent
15     public Student sayHello() {
16         Student s = new Student("M","Felix T","year 12");
17         return s;
18     }
19 }
```

Test the JSON response



A screenshot of a browser developer tools JSON tab titled "localhost:8080/getStudent". The tab has tabs for "JSON", "Raw Data", and "Headers". Below the tabs are buttons for "Save", "Copy", and "Pretty Print". The JSON content is displayed as a single-line string: {"name": "Felix T", "gender": "M", "year": "Year 12"}.



A screenshot of a browser developer tools JSON tab titled "localhost:8080/getStudent". The tab has tabs for "JSON", "Raw Data", and "Headers". Below the tabs are buttons for "Save", "Copy", "Collapse All", "Expand All", and "Filter JSON". The JSON content is displayed with expanded properties: name: "Felix T", gender: "M", year: "Year 12".

## LAB3 HTTP Post Consume JSON Obj

Create “Lab3HttpPostAndConsumeJsonObj.java” and add the consume annotation

The screenshot shows a code editor with two panes. The left pane displays the project structure:

```
CODE-WITH-QUARKUS-LAB
├── .mvn
└── src
    └── main
        ├── docker
        └── java/org/acme
            ├── GreetingResource.java
            ├── Lab0GreetingResource.java
            ├── Lab1HttpGetSample.java
            ├── Lab2HttpReturnJsonObj.java
            ├── Lab2HttpReturnJsonObjList.java
            ├── Lab3HttpPostAndConsumeJsonObj.java
            └── Student.java
        └── resources
    └── test
    └── target
└── .dockerignore
```

The right pane shows the code for `Lab3HttpPostAndConsumeJsonObj.java`:

```
import javax.enterprise.context.ApplicationScoped;
import javax.ws.rs.Consumes;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/addStudent")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
@ApplicationScoped
public class Lab3HttpPostAndConsumeJsonObj {

    @POST
    public List<Student> addStudents(List<Student> student) {
        return student;
    }
}
```

Eg : POST Student Obj and convert to Java Student Obj

The screenshot shows a Postman request configuration:

- Method: POST
- URL: `http://localhost:8080/addStudent`
- Body tab selected
- JSON selected under the dropdown
- Body content:

```
[{"name": "M", "gender": "Felix", "year": "Year 12"}, {"name": "F", "gender": "Fanny", "year": "Year 11"}]
```

The response view shows the JSON output:

```
[{"name": "M", "gender": "Felix", "year": "Year 12"}, {"name": "F", "gender": "Fanny", "year": "Year 11"}]
```

```
curl --location --request POST 'http://localhost:8080/addStudent' \
--header 'Content-Type: application/json' \
--data-raw '[{"name": "M", "gender": "Felix", "year": "Year 12"}, {"name": "F", "gender": "Fanny", "year": "Year 11"}]'
```

## LAB4 JWT

### Add extention

```
smallrye-jwt  
smallrye-jwt-build  
resteas  
resteasy-jackson
```

```
./mvnw quarkus:add-extension -Dextensions="resteas,resteasy-jackson,smallrye-jwt,smallrye-jwt-build"
```

### pom.xml

```
....  
<dependency>  
    <groupId>io.smallrye</groupId>  
    <artifactId>smallrye-jwt-build</artifactId>  
</dependency>  
  
<dependency>  
    <groupId>io.quarkus</groupId>  
    <artifactId>quarkus-resteasy-jackson</artifactId>  
</dependency>  
  
<dependency>  
    <groupId>io.quarkus</groupId>  
    <artifactId>quarkus-smallrye-jwt</artifactId>  
</dependency>  
<dependency>  
    <groupId>io.quarkus</groupId>  
    <artifactId>quarkus-resteasy</artifactId>  
</dependency>  
....
```

Open the `src/main/java/org/acme/security/jwt/TokenSecuredResource.java` file and see the following content:

Path `/secured/permit-all` , to display the SecurityContext info.

Path `/secured/helloadmin`, to allow the role “admin” to access the function

### TokenSecuredResource.java

```
package org.acme.security.jwt;  
  
import javax.annotation.security.PermitAll;  
import javax.annotation.security.RolesAllowed;  
import javax.enterprise.context.RequestScoped;  
import javax.inject.Inject;  
import javax.ws.rs.GET;  
import javax.ws.rs.InternalServerErrorException;  
import javax.ws.rs.POST;  
import javax.ws.rs.Path;  
import javax.ws.rs.Produces;  
import javax.ws.rs.core.Context;  
import javax.ws.rs.core.MediaType;  
import javax.ws.rs.core.SecurityContext;  
  
import org.eclipse.microprofile.jwt.JsonWebToken;
```

```

@Path("/secured")
@RequestScoped
public class TokenSecuredResource {

    @Inject
    JsonWebToken jwt;

    @GET
    @Path("permit-all")
    @PermitAll
    @Produces(MediaType.TEXT_PLAIN)
    public String hello(@Context SecurityContext ctx) {
        return getResponseString(ctx);
    }

    @GET
    @Path("roles-allowed")
    @RolesAllowed({ "User", "Admin" })
    @Produces(MediaType.TEXT_PLAIN)
    public String helloRolesAllowed(@Context SecurityContext ctx) {
        return getResponseString(ctx) + ", birthdate: " + jwt.getClaim("birthdate").toString() + ", exp:" +
            jwt.getClaim("exp").toString();
    }

    private String getResponseString(SecurityContext ctx) {
        String name;
        if (ctx.getUserPrincipal() == null) {
            name = "anonymous";
        } else if (!ctx.getUserPrincipal().getName().equals(jwt.getName())) {
            throw new InternalServerErrorException("Principal and JsonWebToken names do not match");
        } else {
            name = ctx.getUserPrincipal().getName();
        }
        return String.format("hello + %s," +
            " + isHttps: %s," +
            " + authScheme: %s," +
            " + hasJWT: %s",
            name, ctx.isSecure(), ctx.getAuthenticationScheme(), hasJwt());
    }

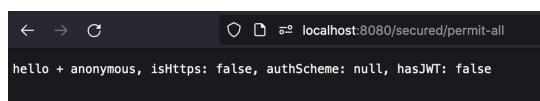
    private boolean hasJwt() {
        return jwt.getClaimNames() != null;
    }

    @GET
    @POST
    @Path("helloworld")
    @RolesAllowed({ "Admin" })
    @Produces(MediaType.TEXT_PLAIN)
    public String helloworld(@Context SecurityContext ctx) {
        return "I am admin. " + getResponseString(ctx) + ", birthdate: " + jwt.getClaim("birthdate").toString() + ", exp:" +
            jwt.getClaim("exp").toString();
    }

}

```

Run mvn quarkus:dev to test  
<http://localhost:8080/secured/permit-all>



POSTMAN

<http://localhost:8080/secured/helloadmin>

>> 401 Unauthorized

Untitled Request

The screenshot shows the Postman interface with a GET request to <http://localhost:8080/secured/helloadmin>. The 'Params' tab is selected, showing a single entry: 'Key' (Value: Value) and 'Description' (Value: Description). The 'Body' tab is selected, showing a JSON response with one item: { "error": "401 Unauthorized" }. The 'Headers' tab shows two entries: 'Content-Type' (Value: application/json) and 'Date' (Value: Sun, 21 Oct 2018 10:45:14 GMT). The 'Tests' tab contains a single test script: `console.log(response.data.error);`. The 'Settings' tab is visible at the bottom. A status bar at the bottom right indicates 'Status: 401 Unauthorized' and 'Time: 14 ms'. A tooltip for the '401 Unauthorized' error message is displayed, stating: 'Similar to 403 Forbidden, but specifically for use when authentication is possible but has failed or not yet been provided. The response must include a WWW-Authenticate header field containing a challenge applicable to the requested resource.'

Open the `src/main/java/org/acme/security/jwt/GenerateToken.java` file and see the following content:

### GenerateToken.java

```
package org.acme.security.jwt;

import java.util.Arrays;
import java.util.HashSet;

import javax.enterprise.context.RequestScoped;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

import io.smallrye.jwt.build.Jwt;
import org.acme.security.UserLoginBean;
import org.eclipse.microprofile.jwt.Claims;

@Path("/authen")
@Produces(MediaType.APPLICATION_JSON)
@RequestScoped
public class GenerateToken {
    /**
     * Generate JWT token
     */
    public static void main(String[] args) {
        String token =
            Jwt.issuer("https://felixtsang.com/issuer")
                .upn("felix@linux.com")
```

```

// .preferredUserName("felix")
    .groups(new HashSet<>(Arrays.asList("User", "Admin")))
    .claim(Claims.birthdate.name(), "2001-07-13")
    // .claim(Claims.exp.name(), "2022-07-13")
    .sign();
System.out.println(token);
}

public static String getJwtToken() {
String token =
Jwt.issuer("https://felixtsang.com/issuer")
.upn("felix@linux.com")
.preferredUserName("felix")
.claim("name", "nathan")
.groups(new HashSet<>(Arrays.asList("User", "Admin")))
.claim(Claims.birthdate.name(), "2001-07-13")
.claim("userid", "123")
.sign();
return token;
}

@POST
@Produces(MediaType.APPLICATION_JSON)
@Path("login")
public UserLoginBean login() {

UserLoginBean userLoginBean = new UserLoginBean();
userLoginBean.setJwtToken(getJwtToken());

return userLoginBean;
// return getJwtToken();
}

@GET
@Produces(MediaType.APPLICATION_JSON)
@Path("logindemo")
public UserLoginBean loginDemo() {

UserLoginBean userLoginBean = new UserLoginBean();
userLoginBean.setJwtToken(getJwtToken());
userLoginBean.setUserId("1");
userLoginBean.setUserName("Felix");

return userLoginBean;
// return getJwtToken();
}
}

```

## Generate the private.key to decryt the jwt token and public to encrypt the jwt

### Generating Keys with OpenSSL

It is also possible to generate a public and private key pair using the OpenSSL command line tool.

openssl commands for generating keys

```
openssl genrsa -out rsaPrivateKey.pem 2048
openssl rsa -pubout -in rsaPrivateKey.pem -out publicKey.pem
```

An additional step is needed for generating the private key for converting it into the PKCS#8 format.

openssl command for converting private key

```
openssl pkcs8 -topk8 -nocrypt -inform pem -in rsaPrivateKey.pem -outform pem -out
privateKey.pem
```

You can use the generated pair of keys insted of the keys used in this quickstart.

Or generate the key pairs online if no openssl command installed

[https://cryptotoools.net/rsagen](https://cryptotools.net/rsagen)

### application.properties

```
mp.jwt.verify.publickey.location=publicKey.pem  
mp.jwt.verify.issuer=https://felixsang.com/issuer  
quarkus.native.resources.includes=publicKey.pem  
smallrye.jwt.sign.key.location=privateKey.pem  
quarkus.http.cors=true  
quarkus.http.cors.origins=http://localhost,http://127.0.0.1  
quarkus.http.cors.method=GET,POST
```

/src/main/resources/

### application.properties

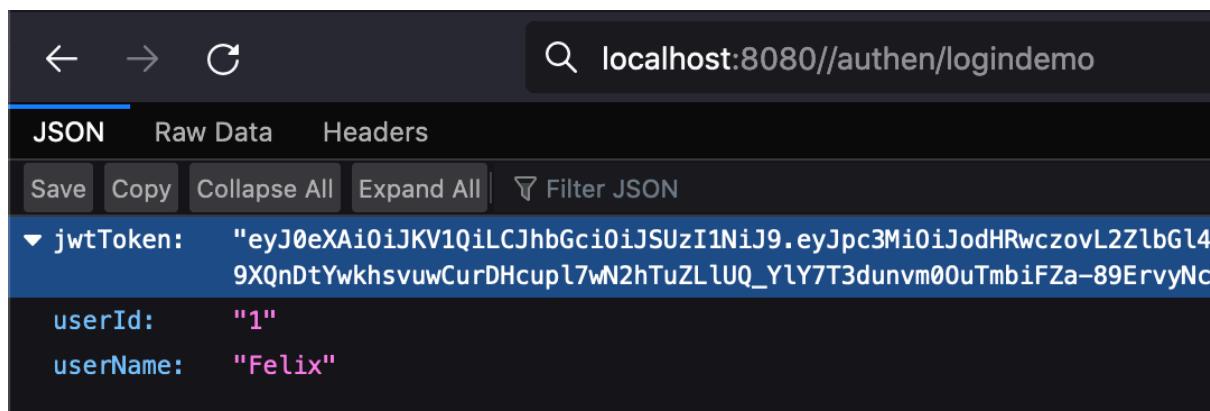
privateKey.pem

publicKey.pem

rsaPrivateKey.pem

1. generate the jwt token

localhost:8080//authen/logindemo



The screenshot shows a Postman interface with the URL `localhost:8080//authen/logindemo` in the search bar. The response is displayed in JSON format:

```
jwtToken: "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJodHRwczovL2ZlbGl49XQnDtYwkhsvuwCurDHcupl7wN2hTuZLlUQ_YLY7T3dunvm0OuTmbiFZa-89ErvyNc"
userId: "1"
userName: "Felix"
```

copy the jwtToken and use the postman to verify

under the postman

1. select POST method
2. `http://localhost:8080/secured/helloadmin`
3. Authorizatoin > Type > “Bearer Token” > patse the token on right hand side

POST http://localhost:8080/secured/helloadmin

**Authorization**

Bearer Token

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

**Token**

eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3Mi0iJodHRwczovL2ZlbGl4dHNhbmcuY29tL2lzC3VlcIIsInVwbii6ImZlbGl4QGxpbnV4LmNvbSIisInByZWZlcnJlZF91c2VybmfTzSI6ImZlbGl4IiwbmfTzSI6Im5hdGhhbiIsImdyb3VwcyI6WyJvc2VyIiwiQWRtaW4iXSwiYmlydGhkYXRlIjoiMjAwMS0wNy0xMyIsInVzZXJpZCI6IjEyMyIsImlhdCI6MTY0NDkyMjQ4MCwiZXhwIjoxNjQ0OTIyNzgwLCJqdGki0iI5MmVhZDc4Yi04MzRmLTQyZGYtODNiOC04MDJlMmM3NmJkYTcif0.t3RWRF7y6wwt--9XQnDtYwkhsuwCurDHcupl7wN2hTuZLlUQ\_YLY7T3dunvm00uTmbiFZa-89ErvyNcE\_vfSwalz6mwJs4gToKE7zQNwg-N\_kY2CtVMnqGJd0RlVLFEGCx5H-7kpF6yeYZCvlMCIo8-5JONGiS5AQVAH2mg4cL7P9P1lfNZyuPqNETqd\_R\_14Feo\_n8NV98-j3A3SjNdlj7zY6xo4\_rw7cpkoAkalCSZAtcJDbizT-2ssCzSPc6BeNFTm4GM2zA3DPB7E0PucbKgevQ8pFMJyCXxiA1mb83QZinqzSr5nzZ4UsXAt-lSiZ\_rCDa5X9lkeeWf2Uyjg"

Status: 200 OK Time: 10 ms Size: 204 B Save

raw curl like the following:

```
curl --location --request POST 'http://localhost:8080/secured/helloadmin' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3Mi0iJodHRwczovL2ZlbGl4dHNhbmcuY29tL2lzC3VlcIIsInVwbii6ImZlbGl4QGxpbnV4LmNvbSIisInByZWZlcnJlZF91c2VybmfTzSI6ImZlbGl4IiwbmfTzSI6Im5hdGhhbiIsImdyb3VwcyI6WyJvc2VyIiwiQWRtaW4iXSwiYmlydGhkYXRlIjoiMjAwMS0wNy0xMyIsInVzZXJpZCI6IjEyMyIsImlhdCI6MTY0NDkyMjQ4MCwiZXhwIjoxNjQ0OTIyNzgwLCJqdGki0iI5MmVhZDc4Yi04MzRmLTQyZGYtODNiOC04MDJlMmM3NmJkYTcif0.t3RWRF7y6wwt--9XQnDtYwkhsuwCurDHcupl7wN2hTuZLlUQ_YLY7T3dunvm00uTmbiFZa-89ErvyNcE_vfSwalz6mwJs4gToKE7zQNwg-N_kY2CtVMnqGJd0RlVLFEGCx5H-7kpF6yeYZCvlMCIo8-5JONGiS5AQVAH2mg4cL7P9P1lfNZyuPqNETqd_R_14Feo_n8NV98-j3A3SjNdlj7zY6xo4_rw7cpkoAkalCSZAtcJDbizT-2ssCzSPc6BeNFTm4GM2zA3DPB7E0PucbKgevQ8pFMJyCXxiA1mb83QZinqzSr5nzZ4UsXAt-lSiZ_rCDa5X9lkeeWf2Uyjg'"
```

## Appendix

### Appendix I JDK Installation

For Mac

1.The detailed procedures of install JDK

<https://devwithus.com/install-java-jdk#mac>

For Windows OS

2.The detailed procedures of install JDK

<https://devwithus.com/install-java-windows-10/>

### Appendix II Maven Installation

For Mac

1.The detailed procedures of **How to Install Maven on Mac**

<https://devwithus.com/install-maven-mac/>

For Windows OS

2.The detailed procedures of **How to Install Maven on windows**

<https://devwithus.com/install-maven-windows/>