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

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"

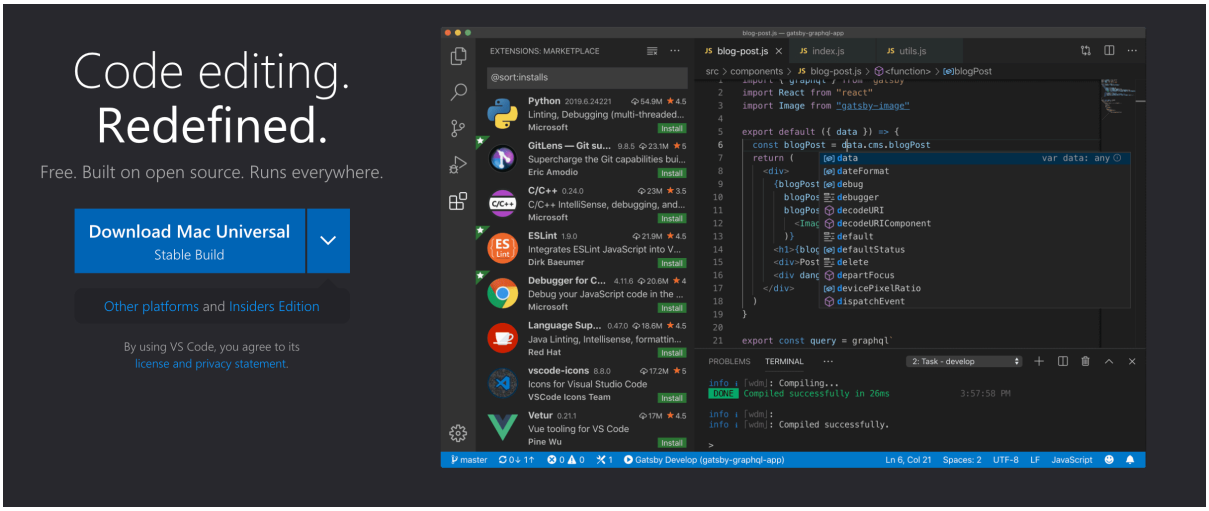
```
admins-iMac:~ admin$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
==> Checking for `sudo` access (which may request your password)...
Password:
==> This script will install:
/usr/local/bin/brew
/usr/local/share/doc/homebrew
/usr/local/share/man/man1/brew.1
/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

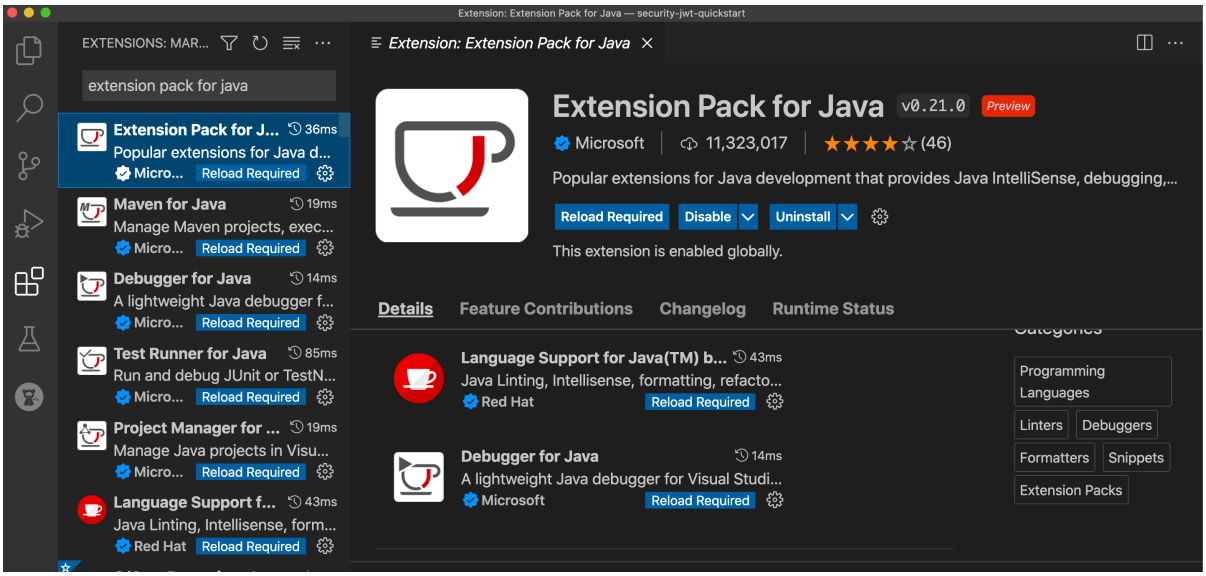
3) IDE (Visual Studio Code)

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

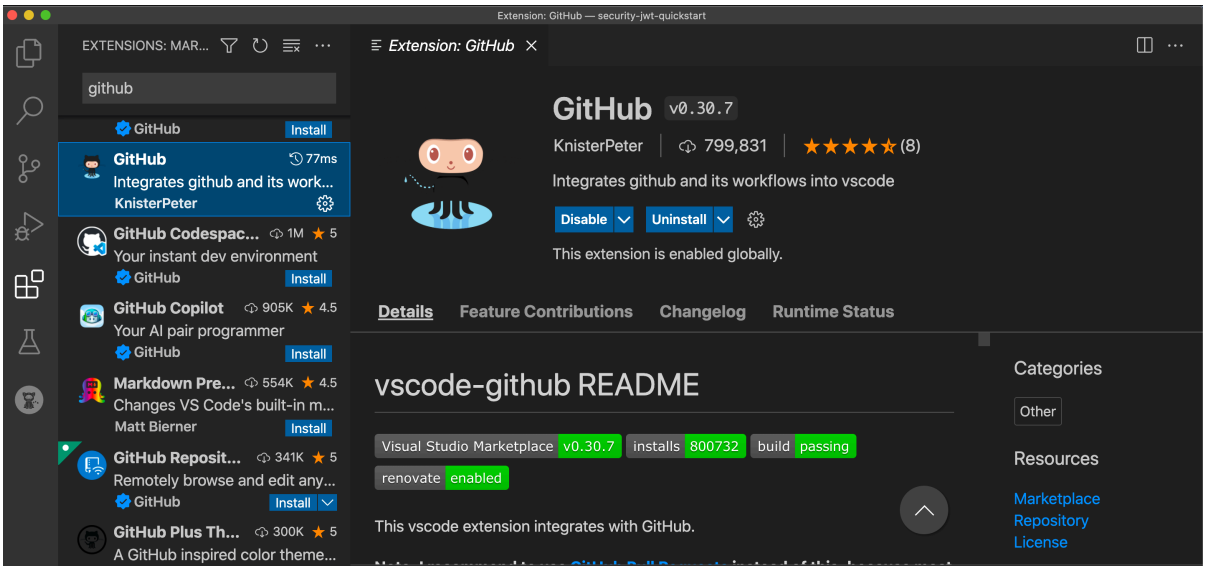


Install Quarkus Tools for Visual Studio Code

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

Install Github for Visual Studio Code

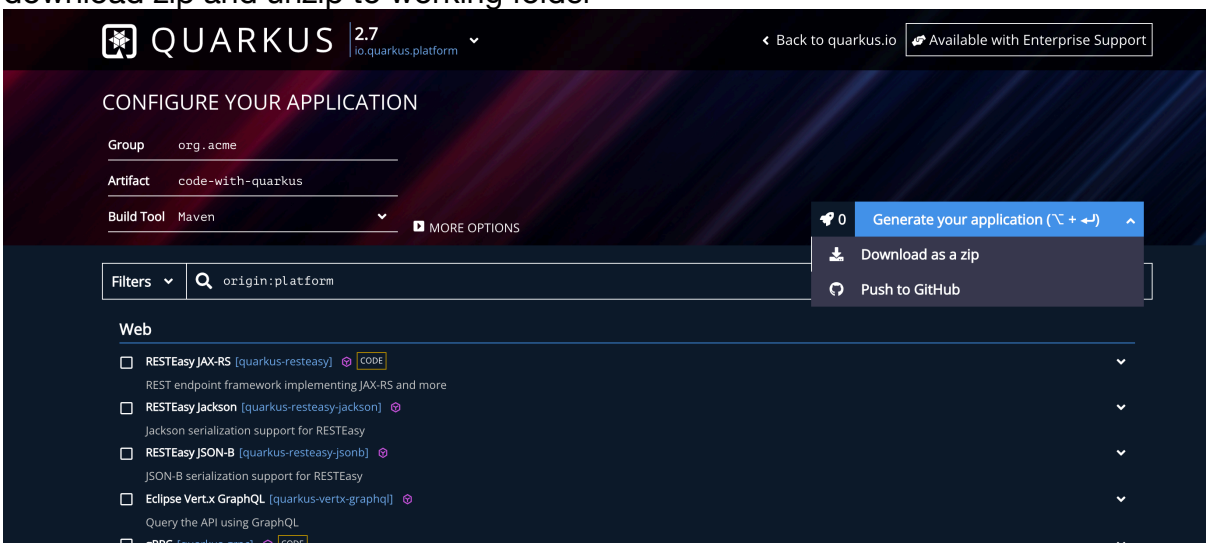
search “Quarkus” under extension and install the quarkus extation as shown below.



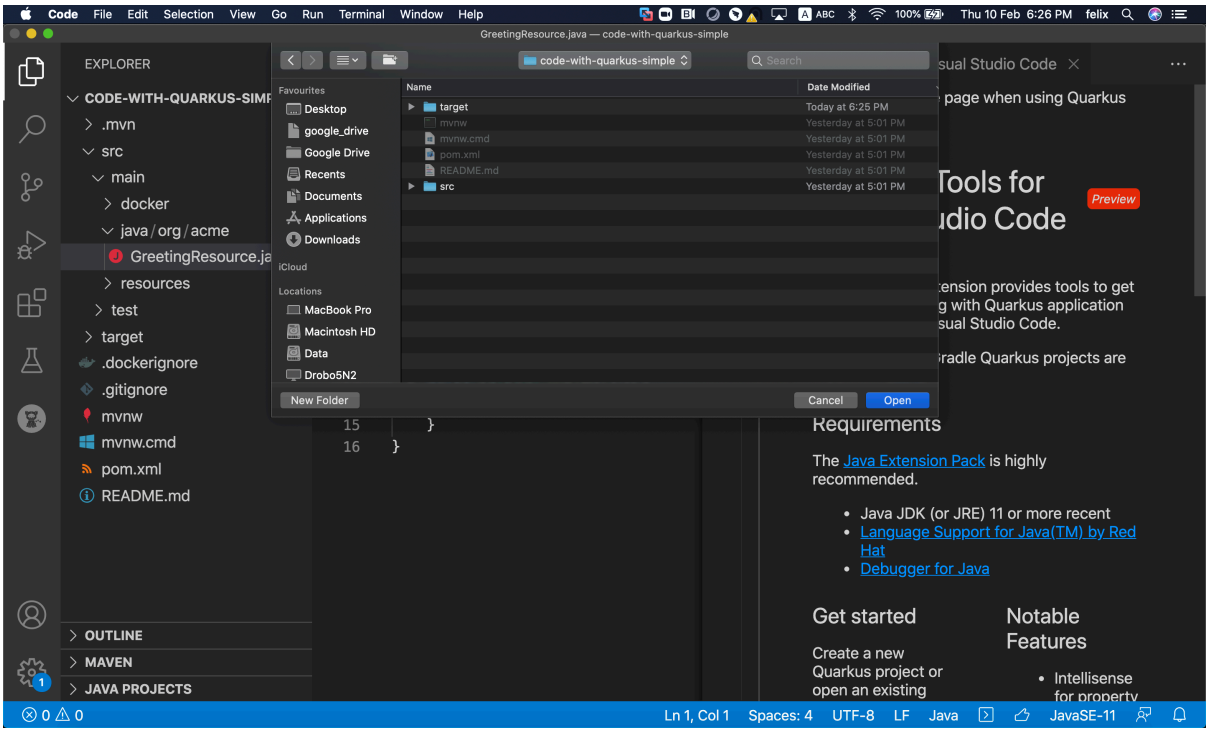
Lab0 Quarkus Sample Project Hello World

Download Link:
https://code.quarkus.io/

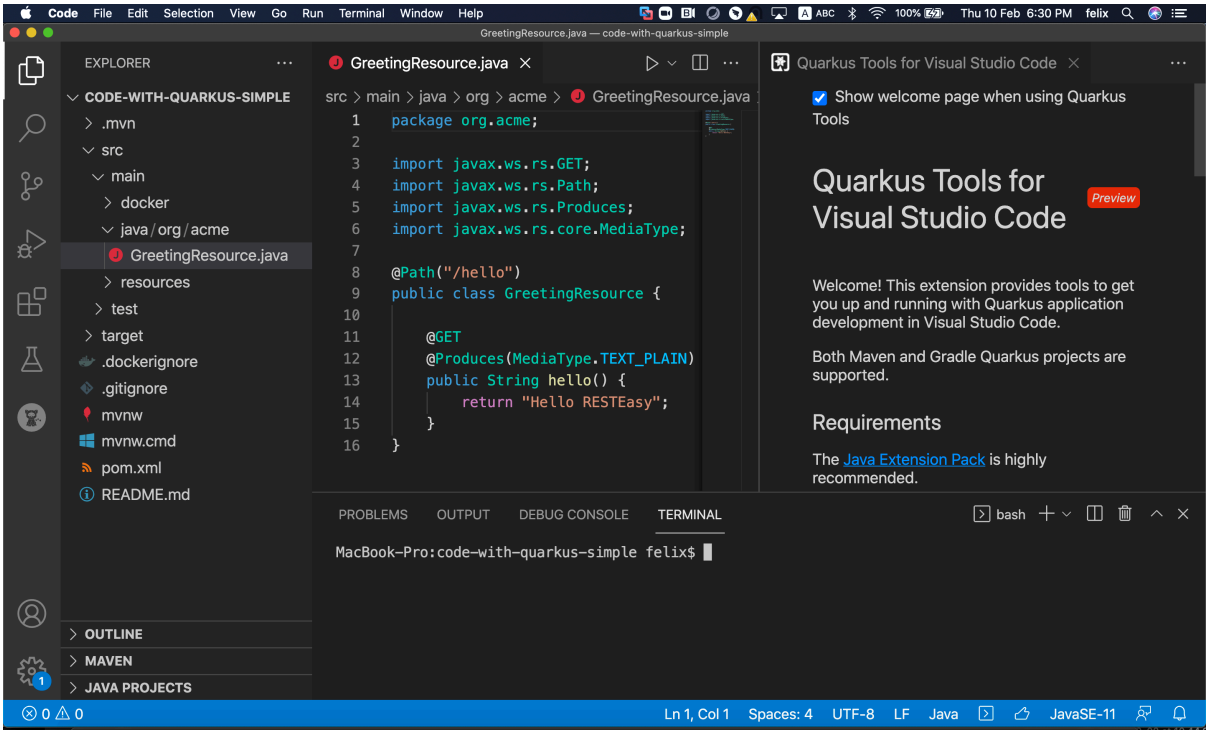
download zip and unzip to working folder



Open visual Studio Code IDE and open the folder



Open Terminal under the menu bar.

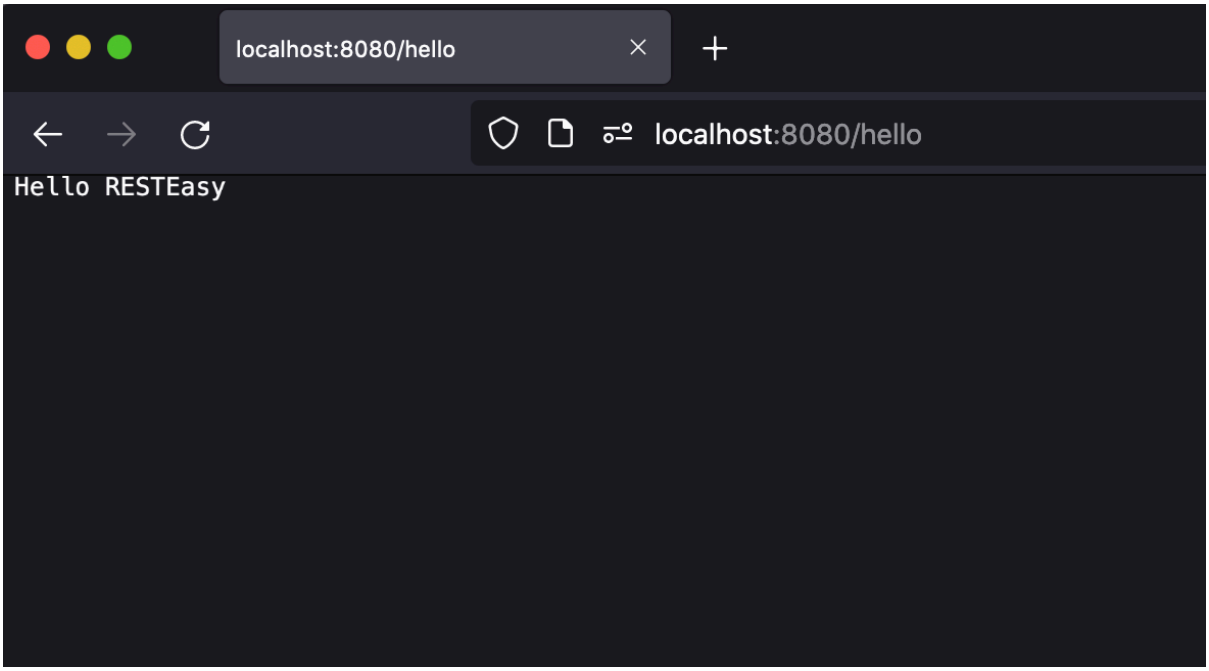


Run the command “mvn quarkus:dev”

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                    bash

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

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                    java

[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
__  ____  __  _____   ___  __ ____  ______
 --/ __ \/ / / / _ | / _ \/ //_/ / / / __/
 -/ /_/ / /_/ / __ |/ , _/ ,< / /_/ /\ \
--\___\_\____/_/ |_/_/|_/_/|_|\____/___/
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>
```

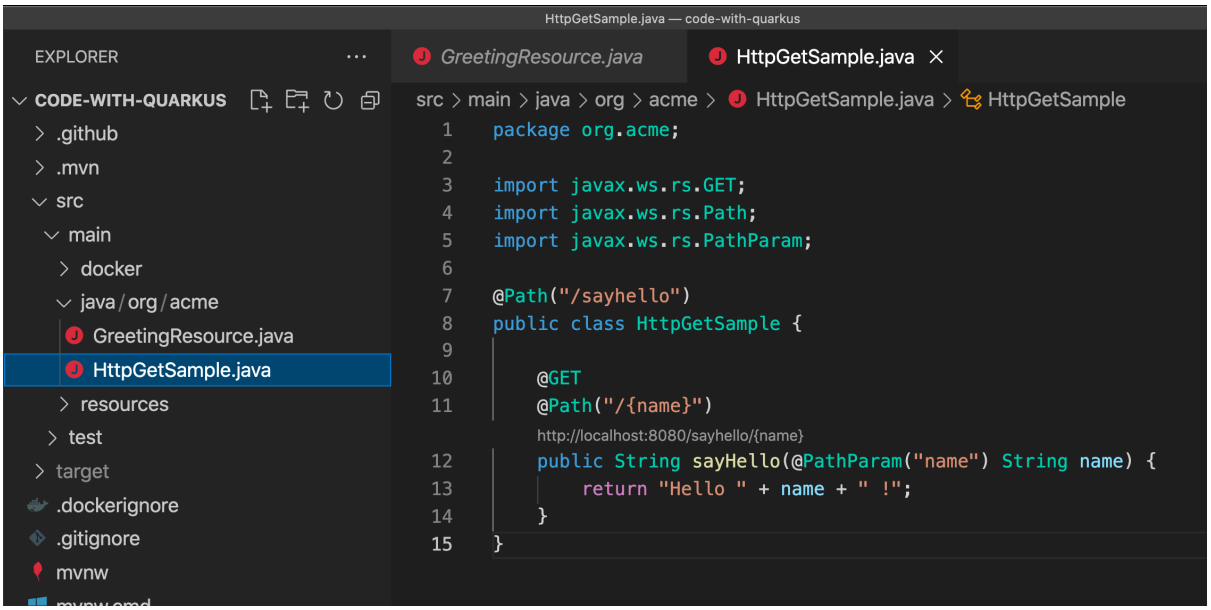
Test with browser : http://localhost:8080/hello

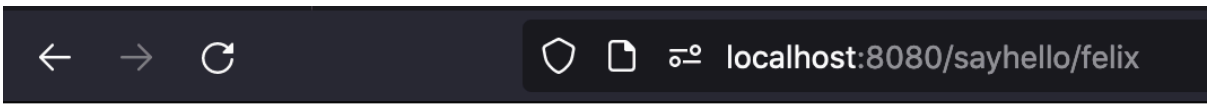


LAB1 URL Parameter

Create an other class named “HttpGetSample”

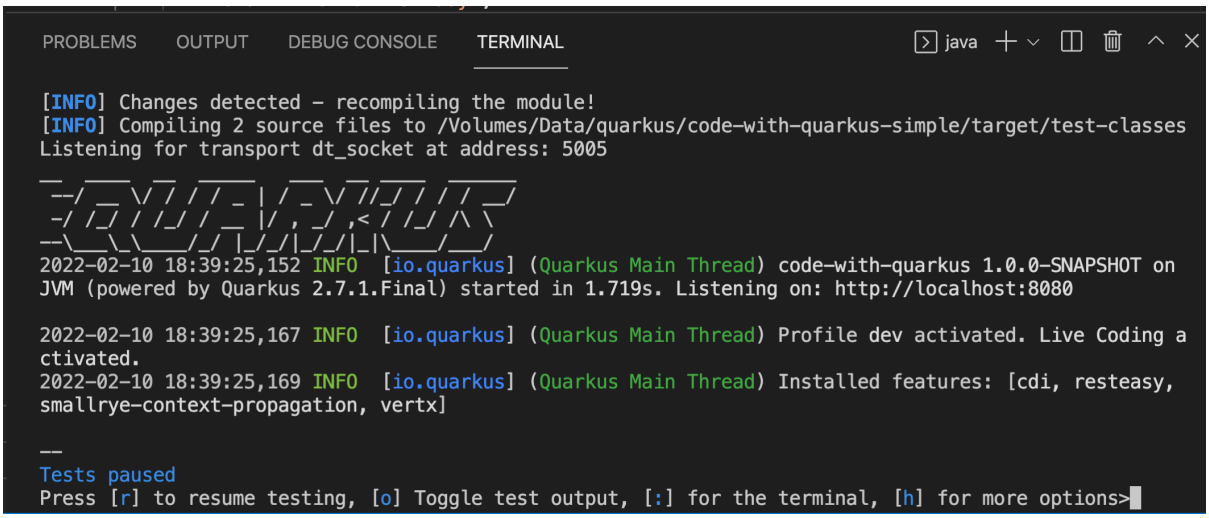
Use PathParam mapper to retrieve the URL parameter



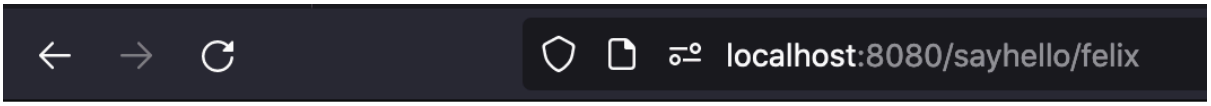


Hello felix !

Run mvn quarkus:dev



Test from Brower locally

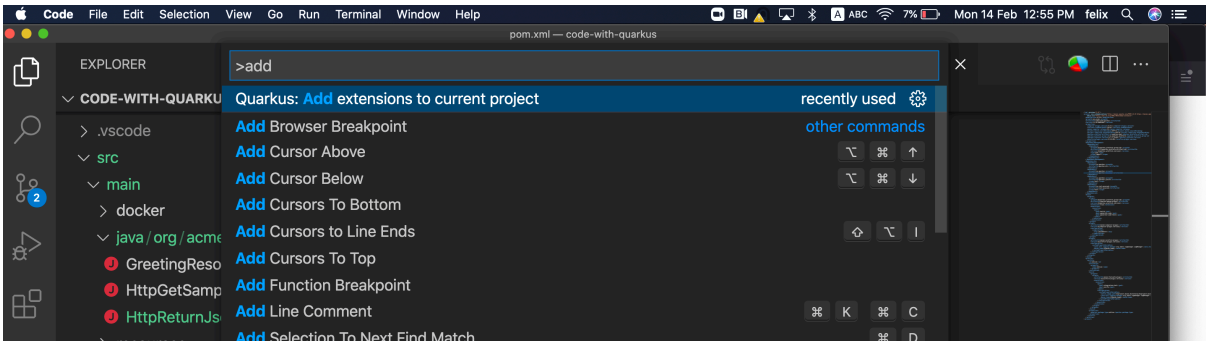


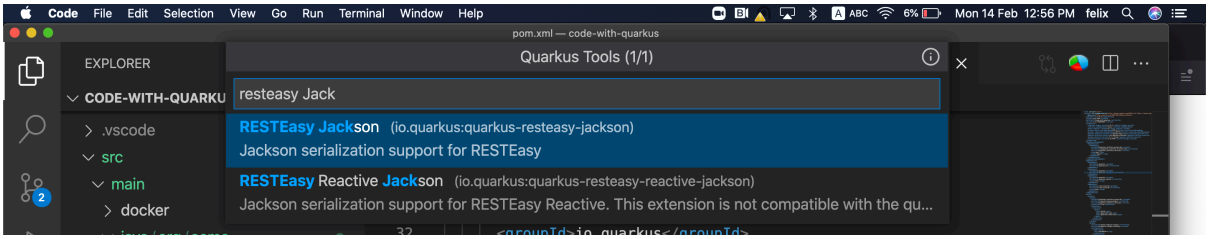
Hello felix !

LAB2 Response with JSON Obj

Importing the RESTEasy/JAX-RS and Jackson extensions

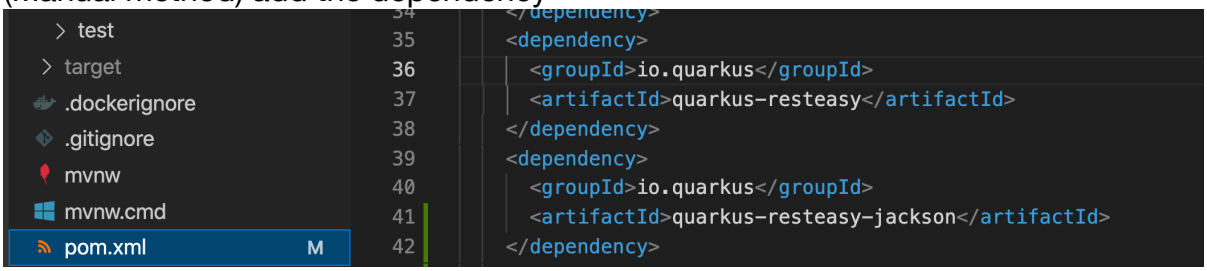
Click “View” > “Command Palette” , type “add” and choose “Quarkus: Add extensions to current project”





and press enter to import the dependency.

Or
(Manual method) add the dependency



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.

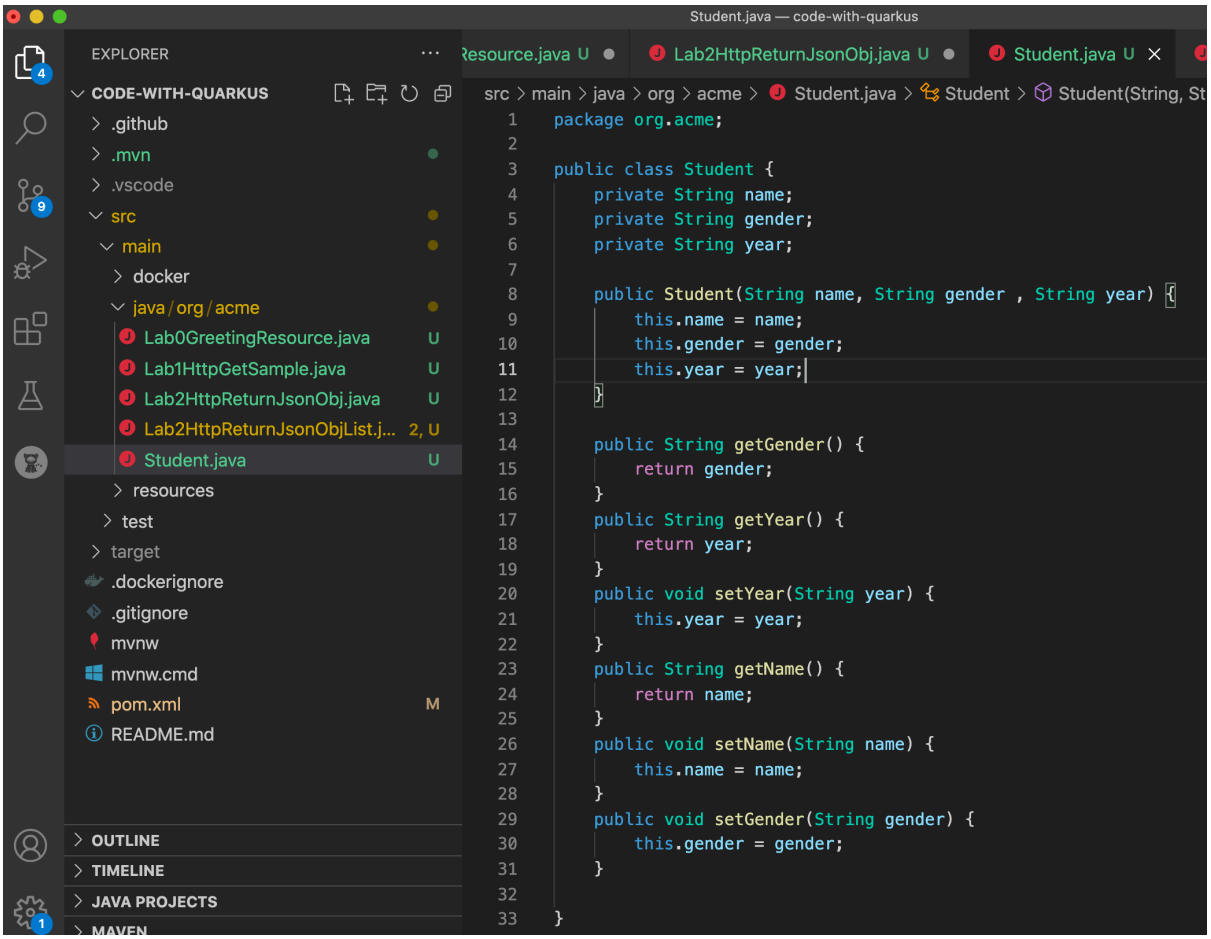
Student.java

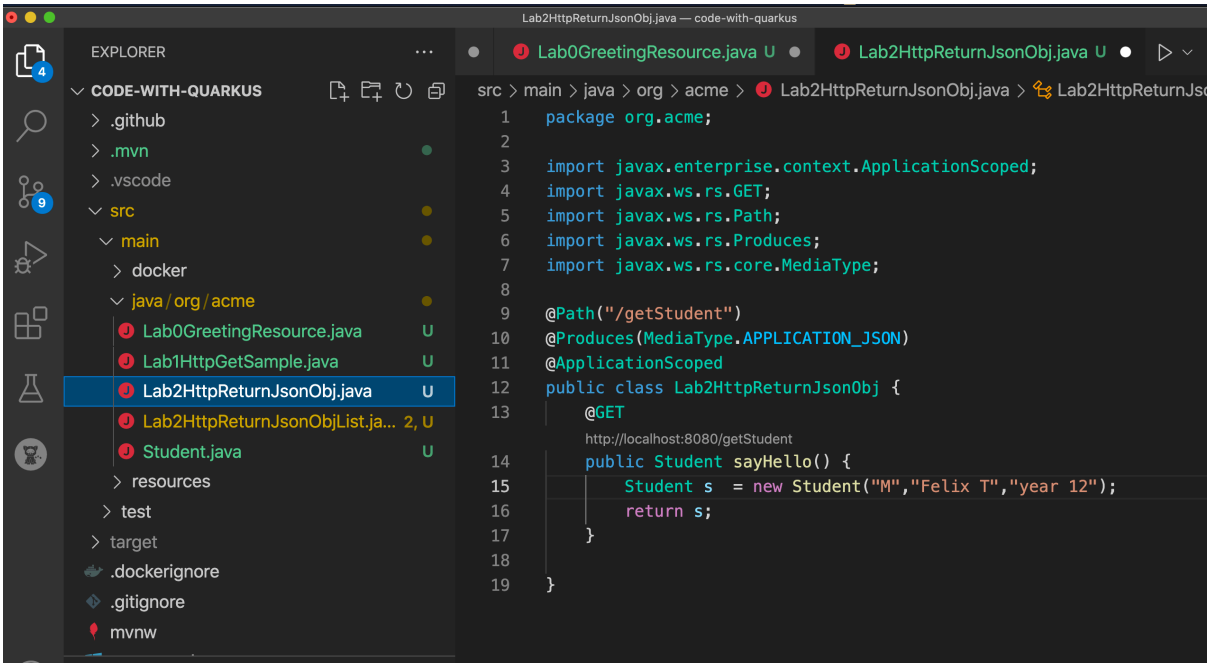
```
package org.acme;

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;
    }

}
```

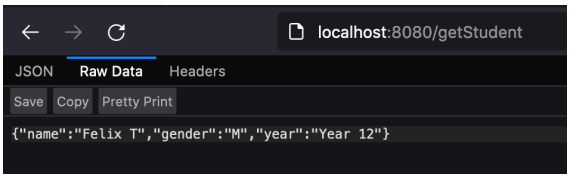
Lab2HttpReturnJsonObj.java

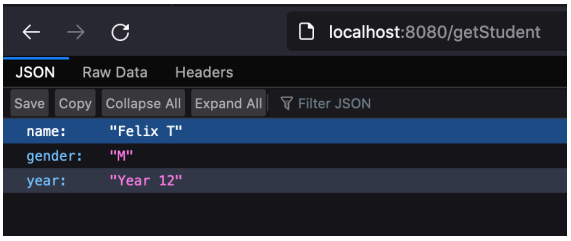
```
package org.acme;

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;
    }

}
```





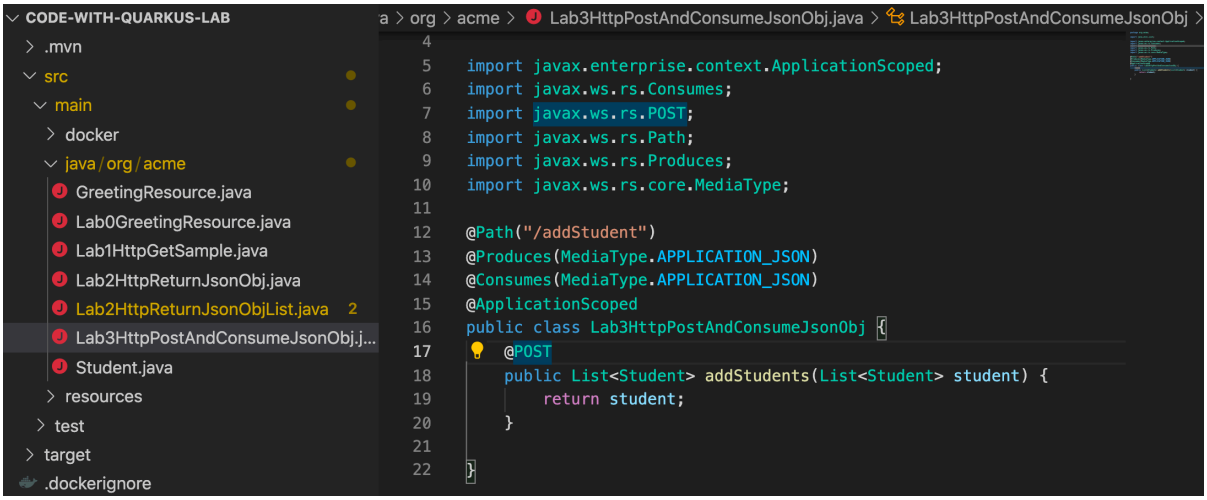
Test the JSON response


localhost:8080/getStudent
JSON
Raw Data
Headers
Save
Copy
Pretty Print
{"name":"Felix T","gender":"M","year":"Year 12"}

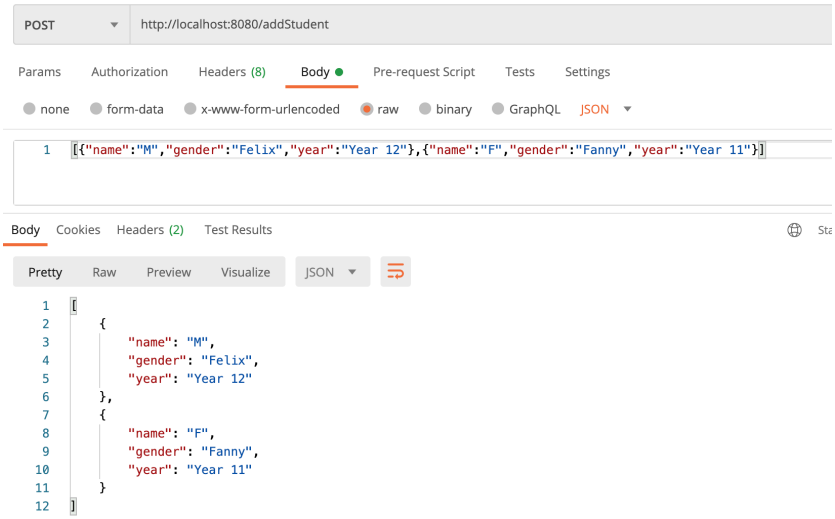

localhost:8080/getStudent
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON
name: "Felix T"
gender: "M"
year: "Year 12"

LAB3 HTTP Post Consume JSON Obj

Create “Lab3HttpPostAndConsumeJsonObj.java” and add the consume annotation



Eg : POST Student Obj and convert to Java Student Obj




```
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
resteasy
resteasy-jackson
```

```
./mvnw quarkus:add-extension -Dextensions="resteasy,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.

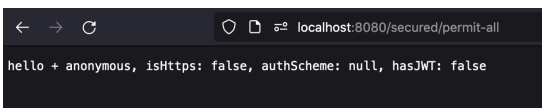
Path /secured/helloadmin, to allow the role “admin” to access the function

`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;
```

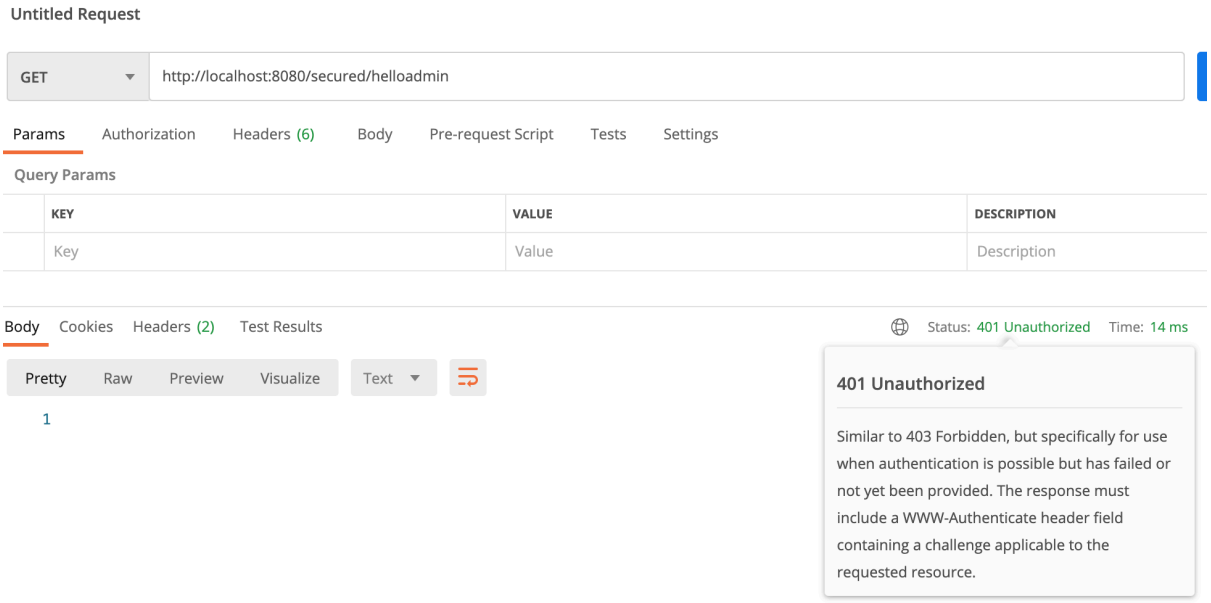
```
@Path("/secured")
@RequestScoped
public class TokenSecuredResource {

    @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("helloadmin")
    @RolesAllowed({ "Admin" })
    @Produces(MediaType.TEXT_PLAIN)
    public String helloadmin(@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



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://cryptotools.net/rsagen

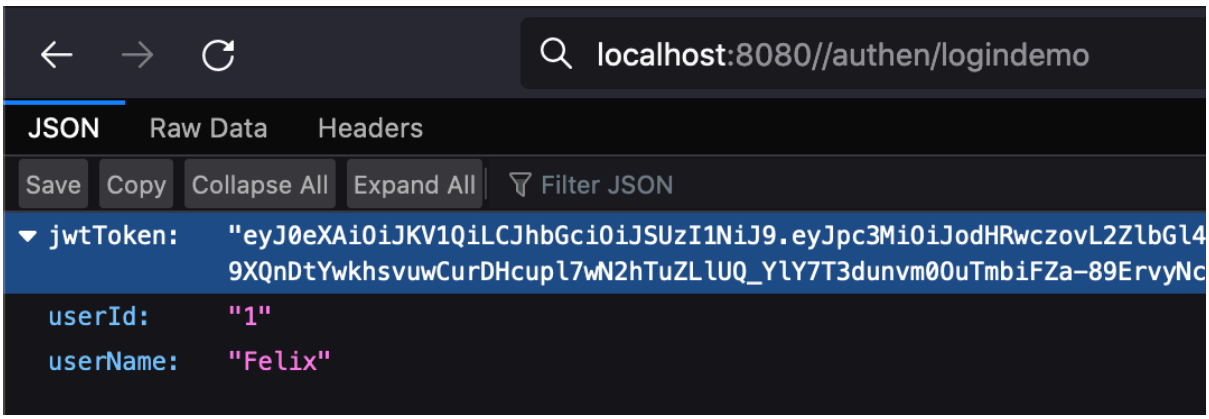
application.properties

```
mp.jwt.verify.publickey.location=publicKey.pem
mp.jwt.verify.issuer=https://felixtsang.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/

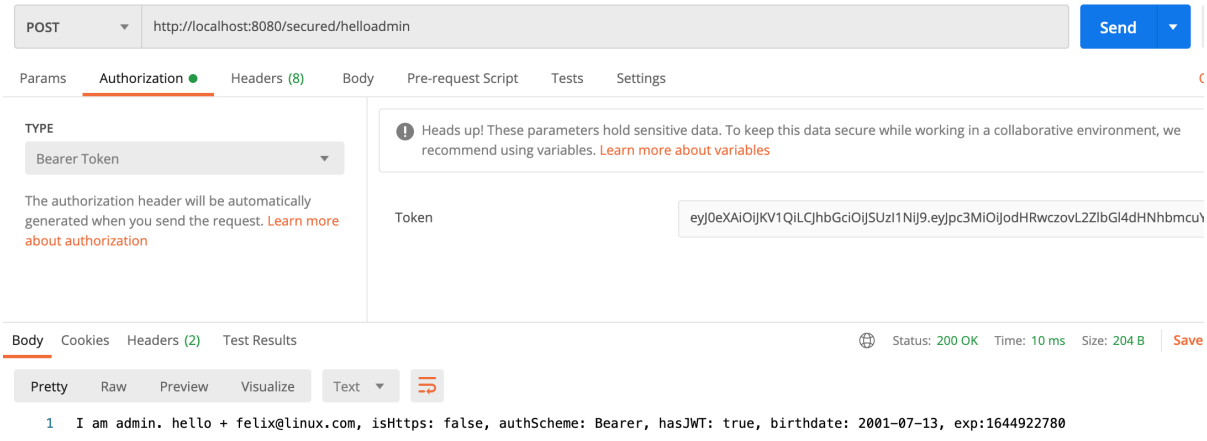
application.properties
privateKey.pem
publicKey.pem
rsaPrivateKey.pem

1. generate the jwt token
   localhost:8080//authen/logindemo



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



raw curl like the following:

```
curl --location --request POST 'http://localhost:8080/secured/helloadmin' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJodHRwczovL2ZlbGl4dHNhbmcuY29tL2lzc3VlciIsInVwbiI6ImZlbGl4QGxpbnV4LmNvbSIsInByZWZlcnJlZF91c2VybmFtZSI6ImZlbGl4Iiwibmftzsi6Im5hdGhhbiIsImdyb3VwcyI6WyJVc2VyIiwiQWRtaW4iXSwiYmlydGhkYXRlIjoiMjAwMS0wNy0xMyIsInVzZXJpZCI6IjEyMyIsImlhdCI6MTY0NDkyMjQ4MCwiZXhwIjoxNjQ0OTIyNzgwLCJqdGkiOiI5MmVhZDc4Yi04MzRmLTQyZGYtODNiOC04MDJlMmM3NmJkYTcifQ.t3RWRF7y6wwt--
9XQnDtYwkhsvuwCurDHcupl7wN2hTuZLlUQ_YlY7T3dunvm0OuTmbiFZa-
89ErvyNcE_vfSwalz6mwJs4gToKE7zQNWg-N_kY2CtVMnqGJd0RlVLFEGCx5H-7kpF6yeYZCvlMCIo8-
5JONGiS5AQVAH2mg4cL7P9P1lfNZyuPqNEtqd_R_14Feo_n8NV98-
j3A3SjNdlj7zY6xo4_rw7cpkoAkalCSZAtcJDbizT-
2ssCzSPc6BeNFTm4GM2zA3DPB7EOPucbKgevQ8pFMJyCXxiA1mb83QZinqzSr5nzZ4UsXAt-
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/