



Academia Tehnică Militară "Ferdinand I"
Facultatea de Sisteme Informaticе și Securitate Cibernetică

PARADIGME DE PROGRAMARE (ÎN JAVA)

CURS 13 - QUARKUS FRAMEWORK

COL(R) TRAIAN NICULA

TEMATICĂ DE CURS

- Introducere în paradigme de programare (1)
- Programare orientată pe obiecte (OOP) (3)
- Programare funcțională și asincronă (3)
- Programare reactivă (1)
- Servicii web REST (2)
- **Quarkus Framework** (3/4)

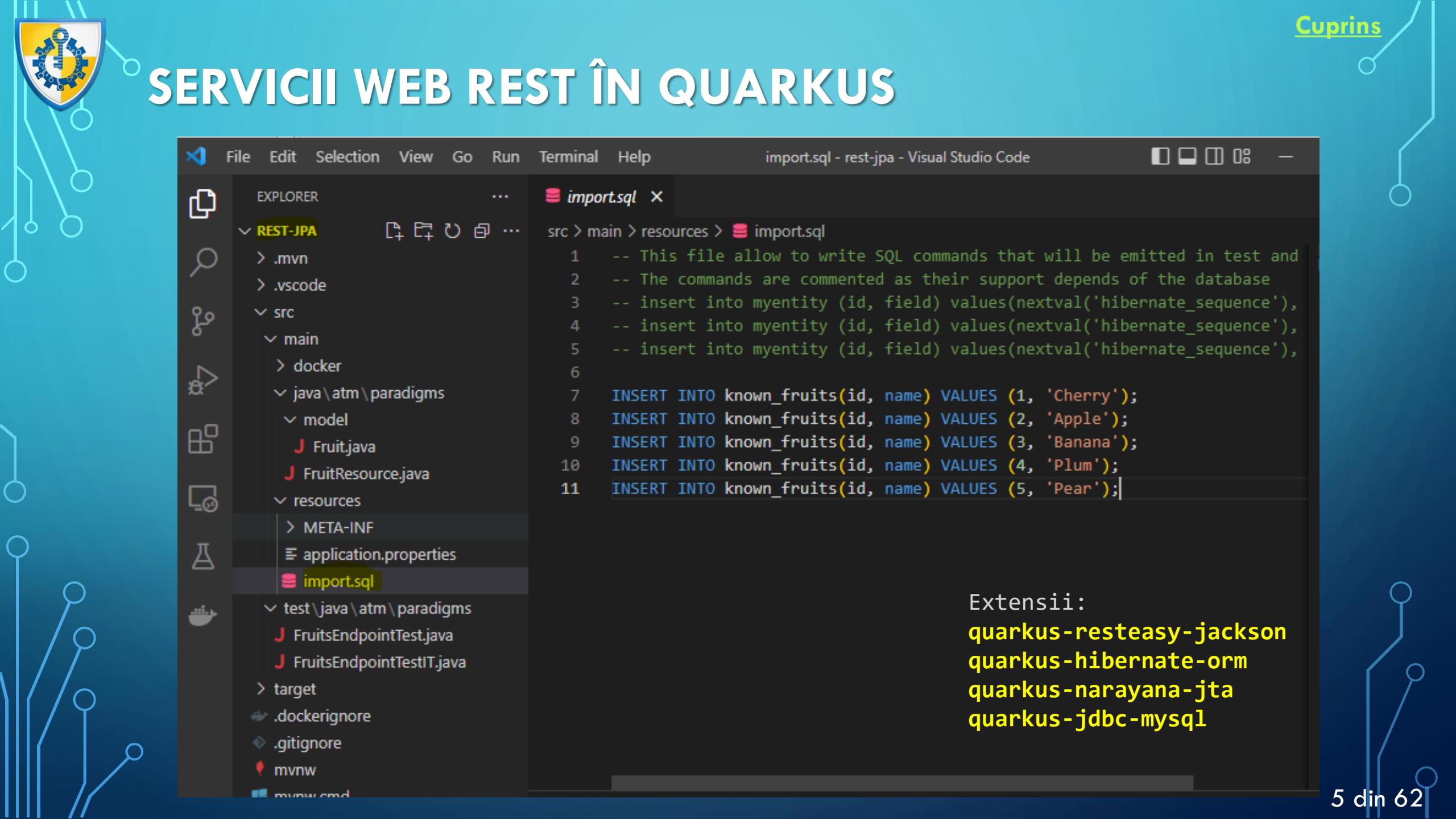
CUPRINS CURS

- Servicii web REST în Quarkus
- Client REST
- Documentare servicii REST
- Arhitectura de securitate
- JSON Web Token
- Bibliografie

SERVICII WEB REST ÎN QUARKUS

- Quarkus utilizează **specificația JAX-RS** pentru a publica servicii web
- Implementarea JAX-RS folosită de Quarkus este **RESTEasy**
- **Toate conceptele și API-urile JAX-RS** prezentate în capitolele 9 și 10 rămân valabile și pentru Quarkus (deși implementarea era Jersey)
- Pentru a uni serviciile web REST cu JPA, JTA și Panache prezentăm 2 proiecte:
- **rest-jpa** – implementează **operații CRUD prin REST folosind JPA** pentru persistență
- **rest-panache** - implementează **operații CRUD prin REST folosind Hibernate ORM cu Panache** pentru persistență
- Ambele proiecte inserează date în baza de date la pornirea aplicației folosind scriptul **import.sql** din **/main/resources**
- **Ambele proiecte expun aceleași endpoint-uri** și au o entitate **Fruit** care mapează tabelul din baza de date **known_fruits**

SERVICIILĂ WEB REST ÎN QUARKUS



A screenshot of the Visual Studio Code interface showing a Java Quarkus project named "REST-JPA". The Explorer sidebar on the left shows the project structure, including ".mvn", ".vscode", "src" (containing "main" with "docker" and "java\atm\paradigms" subfolders, and "model" with "Fruit.java" and "FruitResource.java"), and "resources" (containing "META-INF", "application.properties", and "import.sql"). The "import.sql" file is selected in the Explorer and is open in the main editor area. The editor shows SQL code for inserting five fruit entries into a "known_fruits" table:

```
-- This file allow to write SQL commands that will be emitted in test and
-- The commands are commented as their support depends of the database
-- insert into myentity (id, field) values(nextval('hibernate_sequence'),
-- insert into myentity (id, field) values(nextval('hibernate_sequence'),
-- insert into myentity (id, field) values(nextval('hibernate_sequence'),
-- 
-- INSERT INTO known_fruits(id, name) VALUES (1, 'Cherry');
-- INSERT INTO known_fruits(id, name) VALUES (2, 'Apple');
-- INSERT INTO known_fruits(id, name) VALUES (3, 'Banana');
-- INSERT INTO known_fruits(id, name) VALUES (4, 'Plum');
-- INSERT INTO known_fruits(id, name) VALUES (5, 'Pear');
```

Extensiile disponibile sunt:

- quarkus-resteasy-jackson
- quarkus-hibernate-orm
- quarkus-narayana-jta
- quarkus-jdbc-mysql

SERVICIILĂ WEB REST ÎN QUARKUS

```

@Entity
@Table(name = "known_fruits")
@NamedQuery(name = "Fruits.findAll", query = "SELECT f FROM Fruit f ORDER BY f.name")
public class Fruit {
    @Id
    // avoid PRIMARY KEY violation
    @SequenceGenerator(name = "fruitsSequence", sequenceName = "known_fruits_id_seq",
                       allocationSize = 1, initialValue = 10)
    @GeneratedValue(generator = "fruitsSequence")
    private Long id;
    @Column(length = 40, unique = true)
    private String name;
    public Fruit() {
    }

    public Long getId() {
        return id;
    }

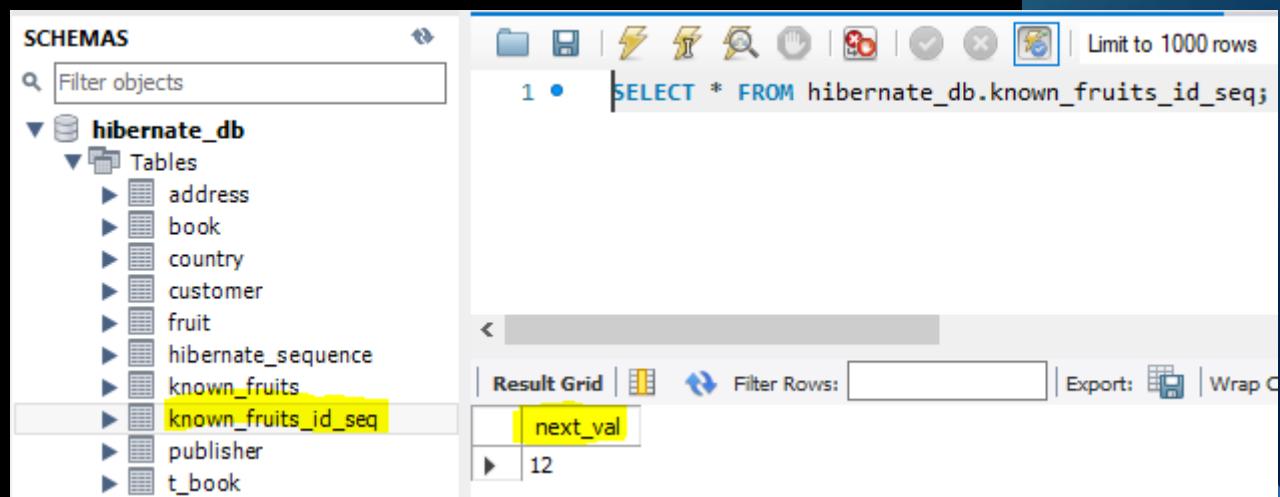
    public void setId(Long id) {
        this.id = id;
    }
}

```

Observăm **@NamedQuery** ca alternativă la interogări dinamice.

"known_fruits_id_seq",

Variantă customizată a adnotării **@GeneratedValue** astfel încât secvența de valori generate pentru PK să înceapă de la 10.



SERVICII WEB REST ÎN QUARKUS

- JAX-RS pune la dispoziție `javax.ws.rs.WebApplicationException` care poate fi generată din metode sau provider-i
- Se folosește pentru a raporta excepțiile la utilizator
- În JAX-RS există o clasă pentru maparea excepțiilor care interceptează excepțiile generate și generează obiectul de răspuns HTTP

`throw new`

`WebApplicationException(Response.Status.NOT_FOUND);`

- Folosind clasa `javax.ws.rs.ext.ExceptionMapper` se pot mapa și customiza excepțiile în mesaje de răspuns HTTP corespunzătoare
- După identificarea clasei de mapare a excepțiilor se invocă metoda `toResponse()`

SERVICII WEB REST ÎN QUARKUS

```
@Path("fruits")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class FruitResource {
    private static final Logger LOGGER = Logger.getLogger(FruitResource.class.getName());
    @Inject
    EntityManager entityManager;

    @GET
    public List<Fruit> get() {
        return entityManager.createNamedQuery("Fruits.findAll", Fruit.class).getResultList();
    }

    @GET
    @Path("{id}")
    public Fruit getSingle(@PathParam("id") Long id) {
        Fruit entity = entityManager.find(Fruit.class, id);
        if (entity == null) {
            throw new WebApplicationException("Fruit with id of " + id + " does not exist.", 404);
        }
        return entity;
    }
}
```

Modul de utilizare a **@NamedQuery** pentru a returna o listă cu toate fructele.

Dacă entitatea **Fruit**, cu **id** obținut din URI client, nu există se generează excepția **WebApplicationException**.

SERVICIİ WEB REST ÎN QUARKUS

```
@POST  
@Transactional  
public Response create(Fruit fruit) {  
    if (fruit.getId() != null) {  
        throw new WebApplicationException("Id was invalidly set on request.", 422);  
    }  
    entityManager.persist(fruit);  
    return Response.ok(fruit).status(201).build();  
}  
  
@PUT  
@Path("{id}")  
@Transactional  
public Fruit update(@PathParam("id") Long id, Fruit fruit) {  
    if (fruit.getName() == null) {  
        throw new WebApplicationException("Fruit Name was not set on request.", 422);  
    }  
    Fruit entity = entityManager.find(Fruit.class, id);  
    if (entity == null) {  
        throw new WebApplicationException("Fruit with id of " + id + " does not exist.", 404);  
    }  
    entity.setName(fruit.getName());  
    return entity;  
}  
...  
...
```

Diferite alte validări pentru operațiile de **CREATE** și **UPDATE** care pot genera generează excepția **WebApplicationException**. Acesta primește ca parametrii un mesaj și un cod de stare.

SERVICIİ WEB REST ÎN QUARKUS

```
@DELETE  
@Path("{id}")  
@Transactional  
public Response delete(@PathParam("id") Long id) {  
    Fruit entity = entityManager.getReference(Fruit.class, id);  
    if (entity == null) {  
        throw new WebApplicationException("Fruit with id of " + id  
                                         + " does not exist.", 404);  
    }  
    entityManager.remove(entity);  
    return Response.status(204).build();  
}  
...
```

SERVICIİ WEB REST ÎN QUARKUS

```
@Provider
public static class ErrorMapper implements ExceptionMapper<Exception> {
    @Inject
    ObjectMapper objectMapper;

    @Override
    public Response toResponse(Exception exception) {
        LOGGER.error("Failed to handle request", exception);
        int code = 500;
        if (exception instanceof WebApplicationException) {
            code = ((WebApplicationException) exception).getResponse().getStatus();
        }
        ObjectNode exceptionJson = objectMapper.createObjectNode();
        exceptionJson.put("exceptionType", exception.getClass().getName());
        exceptionJson.put("code", code);

        if (exception.getMessage() != null) {
            exceptionJson.put("error", exception.getMessage());
        }
        return Response.status(code).entity(exceptionJson).build();
    }
}
```

Interceptează excepțiile generate de **WebApplicationException**, le împachetează într-un mesaj JSON și le trimită în răspuns

SERVICIIL WEB REST ÎN QUARKUS

```
@QuarkusTest
public class FruitsEndpointTest {
    @Test
    public void testFruits() {
        // List all, should have all 3 fruits the database has initially:
        given()
            .when().get("/fruits")
            .then()
            .statusCode(200)
            .body(
                containsString("Cherry"),
                containsString("Apple"),
                containsString("Banana")));
    }

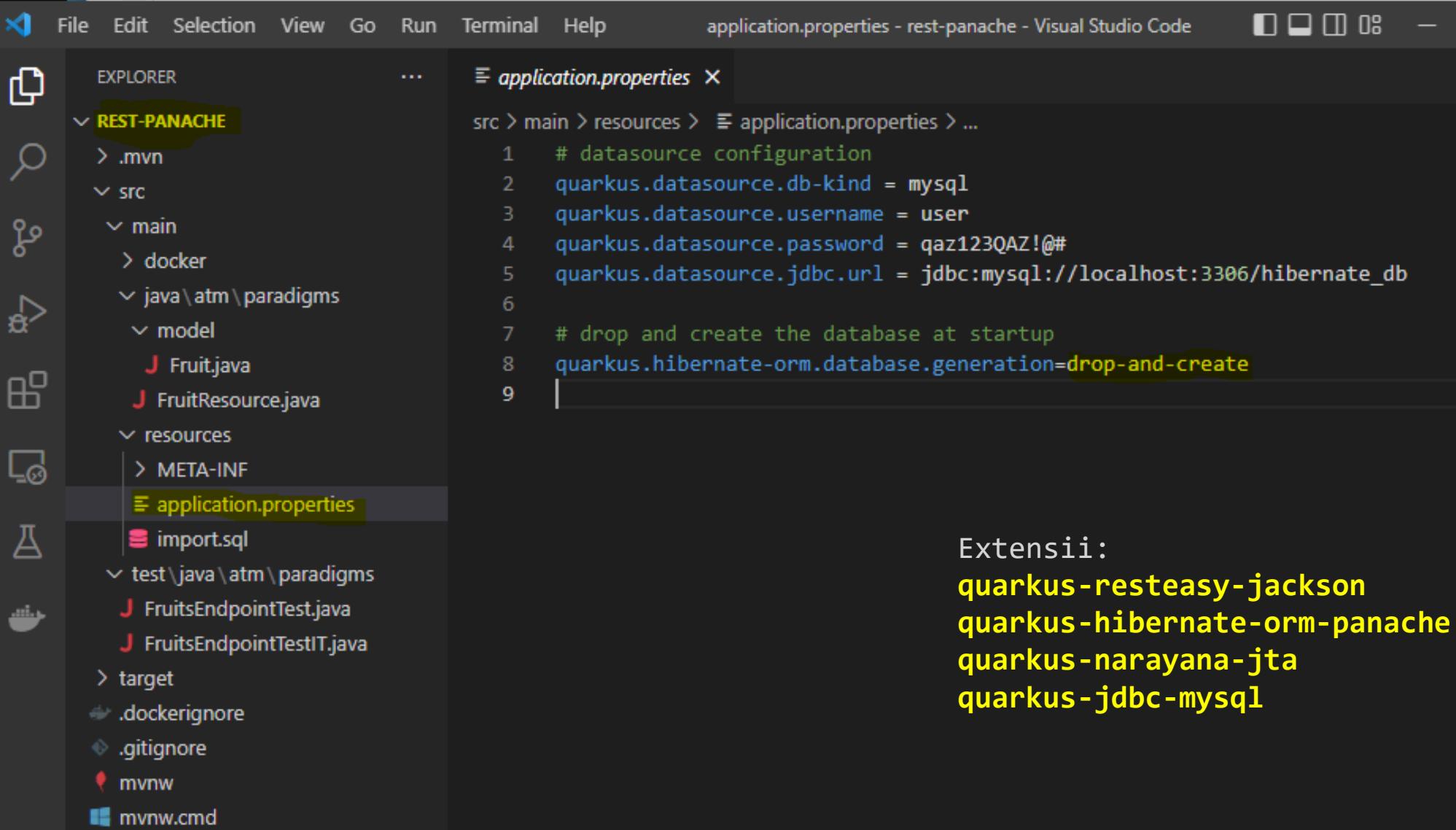
    // Delete the Cherry:
    given()
        .when().delete("/fruits/1")
        .then()
        .statusCode(204);
    ...
}
```

Mod de testare endpoint-uri folosind metodele **Rest Assured**.

SERVICIİ WEB REST ÎN QUARKUS

```
    . . .
    // List all, cherry should be missing now:
    given()
        .when().get("/fruits")
        .then()
        .statusCode(200)
        .body(
            not(containsString("Cherry")),
            containsString("Apple"),
            containsString("Banana")));
    // Create the Pear:
    given()
        .when()
        .body("{\"name\" : \"Fig\"}")
        .contentType("application/json")
        .post("/fruits")
        .then()
        .statusCode(201);
    }
```

SERVICIİ WEB REST ÎN QUARKUS



The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** application.properties - rest-panache - Visual Studio Code.
- Explorer:** Shows the project structure under "REST-PANCHE".
 - src
 - main
 - java\atm\paradigms
 - model
 - Fruit.java
 - FruitResource.java
 - resources
 - META-INF
 - application.properties
 - import.sql
 - test\java\atm\paradigms
 - FruitsEndpointTest.java
 - FruitsEndpointTestIT.java
 - target
 - .dockerignore
 - .gitignore
 - mvnw
 - mvnw.cmd
- Code Editor:** Displays the content of the application.properties file.

```
src > main > resources > application.properties > ...
1 # datasource configuration
2 quarkus.datasource.db-kind = mysql
3 quarkus.datasource.username = user
4 quarkus.datasource.password = qaz123QAZ!@#
5 quarkus.datasource.jdbc.url = jdbc:mysql://localhost:3306/hibernate_db
6
7 # drop and create the database at startup
8 quarkus.hibernate-orm.database.generation=drop-and-create
9 |
```

Extensiî:

quarkus-resteasy-jackson
quarkus-hibernate-orm-panache
quarkus-narayana-jta
quarkus-jdbc-mysql



SERVICIU WEB REST ÎN QUARKUS

```

package atm.paradigms.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;
import io.quarkus.hibernate.orm.panache.PanacheEntityBase;

@Entity
@Table(name = "known_fruits")
public class Fruit extends PanacheEntityBase{
    @Id
    // avoid PRIMARY KEY violation
    @SequenceGenerator(name = "fruitsSequence", sequenceName = "hibernate_sequence",
                       allocationSize = 1, initialValue = 10)
    @GeneratedValue(generator = "fruitsSequence")
    public Long id;

    @Column(length = 40, unique = true)
    public String name;
}
  
```

The screenshot shows a database schema browser with the following details:

- SCHEMAS:** A tree view showing the **hibernate_db** schema containing several tables: address, book, country, customer, fruit, hibernate_sequence, known_fruits, known_fruits_id_seq, publisher, and t_book.
- Tables:** A list of tables under the schema, with **hibernate_sequence** highlighted.
- Result Grid:** A table showing the results of the query `SELECT * FROM hibernate_sequence`. It has one row with the value **next_val** set to **11**.

SERVICIİ WEB REST ÎN QUARKUS

```
@Path("fruits")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class FruitResource {
    private static final Logger LOGGER = Logger.getLogger(FruitResource.class.getName());

    @GET
    public List<Fruit> get() {
        return Fruit.listAll(Sort.by("name"));
    }

    @GET
    @Path("{id}")
    public Fruit getSingle(@PathParam("id") Long id) {
        Fruit entity = Fruit.findById(id);
        if (entity == null) {
            throw new WebApplicationException("Fruit with id of " + id + " does not exist.", 404);
        }
        return entity;
    }
    ...
}
```

Se observă modul de utilizare a metodelor statice aparținând clasei **PanacheEntityBase**.

SERVICIIL WEB REST ÎN QUARKUS

```
@POST  
@Transactional  
public Response create(Fruit fruit) {  
    if (fruit.id != null) {  
        throw new WebApplicationException("Id was invalidly set on request.", 422);  
    }  
    fruit.persist();  
    return Response.ok(fruit).status(201).build();  
}  
  
@PUT  
@Path("{id}")  
@Transactional  
public Fruit update(@PathParam("id") Long id, Fruit fruit) {  
    if (fruit.name == null) {  
        throw new WebApplicationException("Fruit Name was not set on request.", 422);  
    }  
    Fruit entity = Fruit.findById(id);  
    if (entity == null) {  
        throw new WebApplicationException("Fruit with id of " + id + " does not exist.", 404);  
    }  
    entity.name = fruit.name;  
    return entity;  
}
```

Pentru a actualiza o entitate este necesar să i se modifice atrbutele.
Mai departe se ocupă containerul.

SERVICIILĂ WEB REST ÎN QUARKUS

```
    . . .
    @DELETE
    @Path("{id}")
    @Transactional
    public Response delete(@PathParam("id") Long id) {
        Fruit entity = Fruit.findById(id);
        if (entity == null) {
            throw new WebApplicationException("Fruit with id of " + id
                + " does not exist.", 404);
        }
        entity.delete();
        return Response.status(204).build();
    }
```

SERVICIIL WEB REST ÎN QUARKUS

Comenzi *curl* pentru testare endpoint-ui:

```
curl -i -X GET http://localhost:8080/fruits
```

```
curl -i -X GET http://localhost:8080/fruits/1
```

```
curl -i -X POST -H "Content-Type: application/json" -d  
"{"name":"Fig"}" http://localhost:8080/fruits
```

```
curl -i -X PUT -H "Content-Type: application/json" -d  
"{"name":"Orange"}" http://localhost:8080/fruits/2
```

```
curl -i -X DELETE http://localhost:8080/fruits/1
```

SERVICIILĂ WEB REST ÎN QUARKUS

```
Command Prompt

D:\dev\tools\curl\bin>curl -i -X GET http://localhost:8080/fruits
HTTP/1.1 200 OK
Content-Type: application/json
content-length: 121

[{"id":2,"name":"Apple"}, {"id":3,"name":"Banana"}, {"id":1,"name":"Cherry"}, {"id":5,"name":"Pear"}, {"id":4,"name":"Plum"}]
D:\dev\tools\curl\bin>
```

```
Command Prompt

D:\dev\tools\curl\bin>curl -i -X GET http://localhost:8080/fruits/1
HTTP/1.1 200 OK
Content-Type: application/json
content-length: 24

{"id":1,"name":"Cherry"}
D:\dev\tools\curl\bin>
```

SERVICIU WEB REST ÎN QUARKUS

```
Command Prompt
D:\dev\tools\curl\bin>curl -i -X DELETE http://localhost:8080/fruits/1
HTTP/1.1 204 No Content

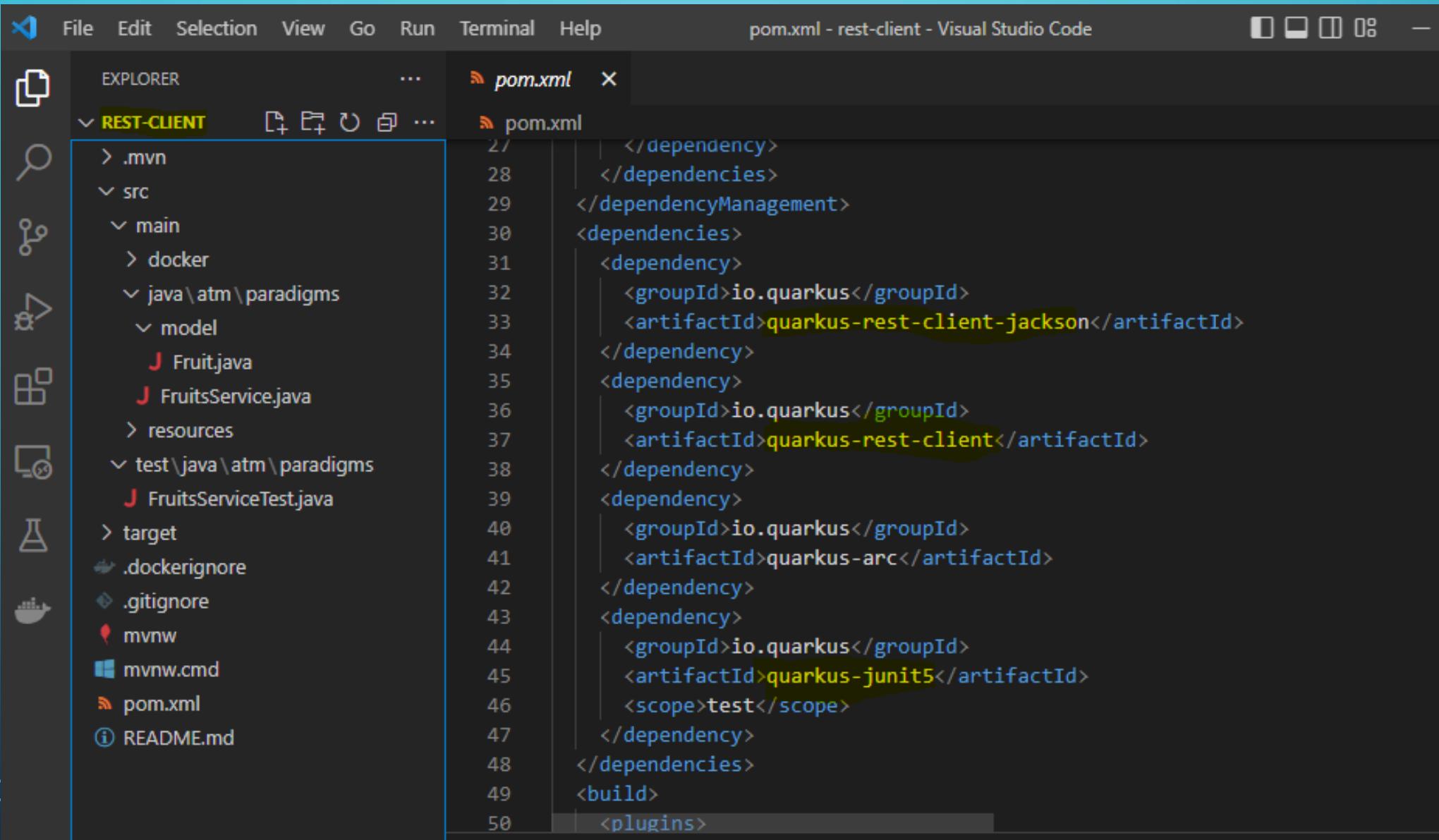
D:\dev\tools\curl\bin>curl -i -X GET http://localhost:8080/fruits/1
HTTP/1.1 404 Not Found
Content-Type: application/json
content-length: 111

{"exceptionType":"javax.ws.rs.WebApplicationException","code":404,"error":"Fruit with id of 1 does not exist."}
D:\dev\tools\curl\bin>
```

CLIENT REST

- În Quarkus implementarea clientilor REST este mult mai simplă folosind adnotări
- Varianta programatică prezentată anterior este încă valabilă prin folosirea clasei **RestClientBuilder**
- Vom prezenta un exemplu de client REST care consumă endpoint-urile puse la dispoziție de serviciul **FruitResource** cu URL `http://localhost:8080/fruits` dezvoltate anterior
- **Clientul REST** în Quarkus are la bază o **interfață** decorată cu **adnotări JAX-RS și MicroProfile**
- Clientul se va ocupa de toată partea de comunicare pe rețea și de serializarea/deserializarea obiectelor
- Adnotarea **@RegisterRestClient** marchează interfața ca și **client REST**
- Exemple din proiectul **rest-client**

CLIENT REST



The image shows a screenshot of the Visual Studio Code (VS Code) interface. The title bar indicates the file is "pom.xml - rest-client - Visual Studio Code". The left sidebar is the Explorer view, showing the project structure:

- .mvn
- src
 - main
 - docker
 - java\atm\paradigms
 - model
 - Fruit.java
 - FruitsService.java
 - resources
 - test\java\atm\paradigms
 - FruitsServiceTest.java
 - target
 - .dockerignore
 - .gitignore
 - mvnw
 - mvnw.cmd
 - pom.xml
 - README.md

The main editor area displays the content of the pom.xml file:

```
27 |     </dependency>
28 |   </dependencies>
29 | </dependencyManagement>
30 | <dependencies>
31 |   <dependency>
32 |     <groupId>io.quarkus</groupId>
33 |     <artifactId>quarkus-rest-client-jackson</artifactId>
34 |   </dependency>
35 |   <dependency>
36 |     <groupId>io.quarkus</groupId>
37 |     <artifactId>quarkus-rest-client</artifactId>
38 |   </dependency>
39 |   <dependency>
40 |     <groupId>io.quarkus</groupId>
41 |     <artifactId>quarkus-arc</artifactId>
42 |   </dependency>
43 |   <dependency>
44 |     <groupId>io.quarkus</groupId>
45 |     <artifactId>quarkus-junit5</artifactId>
46 |     <scope>test</scope>
47 |   </dependency>
48 | </dependencies>
49 | <build>
50 |   <plugins>
```

The artifact IDs "quarkus-rest-client-jackson" and "quarkus-rest-client" are highlighted in yellow.

CLIENT REST

```
package atm.paradigms.model;

public class Fruit {
    private Long id;
    private String name;
    public Fruit() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

POJO corespunzător clasei Fruit.

CLIENT REST

```
@RegisterRestClient(baseUri = "http://localhost:8080/")
public interface FruitsService {

    @GET
    @Path("fruits")
    Set<Fruit> getAllFruits();

    @GET
    @Path("fruits/{id}")
    Fruit getFruitById(@PathParam("id") long id);

    @POST
    @Path("fruits")
    Fruit addFruit(Fruit fruit);

    @PUT
    @Path("fruits/{id}")
    Fruit updateFruit(@PathParam("id") long id, Fruit fruit);

    @DELETE
    @Path("fruits/{id}")
    void deleteFruit(@PathParam("id") long id);
}
```

Interfață care devine client REST
prin decorarea cu adnotări
JAX-RS și **@RegisterRestClient**.

CLIENT REST

```
@QuarkusTest
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class FruitsServiceTest {
    @Inject
    @RestClient
    FruitsService fruitsService;

    @Test
    @Order(1)
    void shouldGetAllFruits(){
        Set<Fruit> fruits = fruitsService.getAllFruits();
        assertEquals(5, fruits.size());
    }

    @Test
    @Order(2)
    void shouldGetOneFruit(){
        Fruit fruit = fruitsService.getFruitById(1);
        assertNotNull(fruit);
    }
    ...
}
```

Clasă de test pentru verificarea clientului REST definit de interfață. La punctul de injecție al interfeței **FruitsService** apare adnotarea **@RestClient**. Metodele de test invocă metodele interfeței **FruitsService**, fără implementare.

CLIENT REST

```
...  
@Test  
@Order(4)  
void shouldUpdateFruit(){  
    Fruit fruit = new Fruit();  
    fruit.setName("Fig");  
    fruit = fruitsService.updateFruit(1, fruit);  
    assertNotNull(fruit.getId());  
}  
  
@Test  
@Order(5)  
void shouldDeleteFruit(){  
    fruitsService.deleteFruit(1);  
    Set<Fruit> fruits = fruitsService.getAllFruits();  
    assertEquals(5, fruits.size());  
}
```

[INFO] Tests run: 5, Failures: 0, Errors: 0,
Skipped: 0, Time elapsed: 3.797 s - in
atm.paradigms.FruitsServiceTest
2022-10-20 13:54:41,728 INFO [io.quarkus]
(main) rest-client stopped in 0.016s
[INFO]
[INFO] Results:
[INFO]
[INFO] **Tests run: 5, Failures: 0, Errors: 0,**
Skipped: 0
[INFO]
[INFO] -----

[INFO] BUILD SUCCESS

DOCUMENTARE SERVICII REST

- **JAX-RS nu specifică modul de documentare a endpoint-urilor expuse pentru a fi consumate de diferiți clienți**
- Din punctul de vedere al dezvoltatorului de microservicii este important să înțeleagă cum să interacționeze cu alte servicii REST
- **Quarkus permite documentarea API prin intermediul specificației OpenAPI**
- **Eclipse MicroProfile OpenAPI pune la dispoziție un set de interfețe și modele de programare care permit producerea de documentație în mod nativ pentru endpoint-urile REST dezvoltate**
- **Eclipse MicroProfile OpenAPI se găsește în pachetul org.eclipse.microprofile.openapi**



DOCUMENTARE SERVICII REST

- Adnotări comune OpenAPI:

Adnotare	Descriere
@APIResponse	Descrie răspunsul generat de endpoint (cod de stare, structură de date etc.)
@Operation	Descrie o operație asupra unei căi
@OpenAPIDefinition	Documentul rădăcină
@Parameter	Numele parametrului metodei
@RequestBody	Scurtă descriere a corpului cererii
@Schema	Definirea tipurilor de date de intrare și ieșire
@Tag	Adăugarea de etichete unui endpoint pentru o descriere suplimentară

- Pentru a folosi OpenAPI se adaugă la proiect extensia
quarkus-smallrye-openapi



DOCUMENTARE SERVICII REST

- Specificația **OpenAPI** (înainte **Swagger**) constă în fișiere pentru descrierea, producerea, consumul și vizualizarea serviciilor web REST
- **Specificația** trebuie scrisă în **format YAML** sau **JSON** și trebuie să conțină:
- **Versiunea OpenAPI** (openapi: 3.0.3)
- Secțiunea **info** (titlu, descriere, versiune)

info:

title: Generated API

version: "1.0"

- Secțiunea **paths** care descrie endpoint-urile

paths:

/authors/{index}:

get:

summary: Returns an author for a given index

- Fișierele sunt generate automat prin scanarea adnotărilor JAX-RS și a obiectelor JAVA



DOCUMENTARE SERVICII REST

- Pentru a demonstra OpenAPI folosim proiectul **openapi-initial**

The screenshot shows the Visual Studio Code (VS Code) interface. On the left, the Explorer sidebar displays the project structure for 'OPENAPI-INITIAL'. The 'src/main/java/atm/paradigms' folder contains 'AuthorResource.java' and 'AuthorResourceTest.java'. The 'src/test/java/atm/paradigms' folder contains 'AuthorResourceIT.java' and 'AuthorResourceTest.java'. Other files shown include '.mvn', '.dockerignore', '.gitignore', 'mvnw', 'mvnw.cmd', and 'README.md'. The 'pom.xml' file is selected in the Explorer and is also open in the main code editor window. The code editor shows the XML configuration for Maven dependencies, specifically for Quarkus and Rest Assured.

```
pom.xml
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>io.quarkus</groupId>
            <artifactId>quarkus-smallrye-openapi</artifactId>
        </dependency>
        <dependency>
            <groupId>io.quarkus</groupId>
            <artifactId>quarkus-resteasy</artifactId>
        </dependency>
        <dependency>
            <groupId>io.quarkus</groupId>
            <artifactId>quarkus-arc</artifactId>
        </dependency>
        <dependency>
            <groupId>io.quarkus</groupId>
            <artifactId>quarkus-junit5</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>io.rest-assured</groupId>
            <artifactId>rest-assured</artifactId>
        </dependency>
    </dependencies>
</dependencyManagement>
<dependencies>
    <dependency>
```



DOCUMENTARE SERVICII REST

- Proiectul are o clasă resursă **AuthorResource**

```
package atm.paradigms;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("authors")
@Produces(MediaType.APPLICATION_JSON)
public class AuthorResource {
    String[] scifiAuthors = {"Isaac Asimov", "Nora Jemisin", "Douglas Adams"};

    @GET
    @Path("{idx}")
    public String getAuthor(@PathParam("idx") int idx){
        return scifiAuthors[idx];
    }
}
```



DOCUMENTARE SERVICII REST

- Quarkus are o pagină de start *index.html* aflată în */resources/META-INF* care poate fi personalizată
- Are o consolă de administrare accesibilă la <http://localhost:8080/q/dev/>
- Fișierul YAML autogenerat este accesibil la <http://localhost:8080/q/openapi>
- Interfața Swagger UI pentru testare este disponibilă la <http://localhost:8080/q/swagger-ui/>

openapi: 3.0.3**info:****title:** openapi-initial API**version:** 1.0.0-SNAPSHOT**paths:****/authors/{idx}:****get:****tags:**

- Author Resource

parameters:- **name:** idx**in:** path**required:** true**schema:****format:** int32**type:** integer**responses:****"200":****description:** OK**content:****application/json:****schema:****type:** string



openapi-initial API

1.0.0-SNAPSHOT

OAS3

/q/openapi

Author Resource

GET

/authors/{idx}

Try it out

Parameters

Name

Description

idx * requiredinteger(\$int32)
(path)

idx

Responses

Code

Description

Links

200

OK

No links

Media type

application/json

Controls Accept header.

Example Value | Schema

"string"



DOCUMENTARE SERVICII REST

```
@Path("authors")
@Produces(MediaType.APPLICATION_JSON)
public class AuthorResource {
    String[] scifiAuthors = { "Isaac Asimov", "Nora Jemisin", "Douglas Adams" };

    @GET
    @Path("{idx}")
    @Operation(summary = "Returns an author for a given index")
    @APIResponse(responseCode = "204", description = "Author not found")
    @APIResponse(responseCode = "200", description = "Author returned for a given index")
    public String getAuthor(@PathParam("idx") int idx) {
        return scifiAuthors[idx];
    }
}
```

Cu adnotările **Open API** se pot adăuga mai multe informații la descrierea generată automat a endpoint-ului.

```

---
openapi: 3.0.3
info:
  title: openapi-initial API
  version: 1.0.0-SNAPSHOT
paths:
  /authors/{idx}:
    get:
      tags:
        - Author Resource
      summary: Returns an author for a given index
      parameters:
        - name: idx
          in: path
          required: true
        schema:
          format: int32
          type: integer
      responses:
        "204":
          description: Author not found
        "200":
          description: Author returned for a given index
          content:
            application/json:
              schema:
                type: string

```

```

--- Fruit Resource Fragment
put:
  tags:
    - Fruit Resource
  parameters:
    - name: id
      in: path
      required: true
      schema:
        format: int64
        type: integer
  requestBody:
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Fruit'
  responses:
    "200":
      description: OK
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Fruit'

```



DOCUMENTARE SERVICII REST

The screenshot shows the Swagger UI interface for the **rest-panache API**. The top navigation bar includes the logo, the title "Swagger UI", the URL "/q/openapi", the button "Explore", and the text "rest-panache (powered by Quarkus)". Below the title, it says "rest-panache API" with version "1.0.0-SNAPSHOT" and "OAS3". A green callout box highlights the "Fruit Resource" section, which contains five endpoint definitions:

- GET /fruits**
- POST /fruits**
- GET /fruits/{id}**
- PUT /fruits/{id}**
- DELETE /fruits/{id}**

The "Schemas" section shows a link to "Fruit".

Adăugând extensia Open API la serviciul web **FruitResource se generează interfața **SwaggerUI** pentru testarea endpoint-urilor în browser.**



ARHITECTURA DE SECURITATE

- Interfața ***HttpAuthenticationMechanism*** este folosită de **Quarkus** pentru a **extrage credențialele de autentificare** din cererea HTTP și a le **delega către IdentityProvider** pentru a le **converti în SecurityIdentity**
- **IdentityProvider** verifică credențialele de autentificare și le mapează la **SecurityIdentity** care conține numele utilizator, rolurile, credențialele originale și alte attribute

Mecanisme de autentificare în Quarkus:

- **Basic și Form authentication**
- **WebAuthn authentication**
- **Mutual TLS (mTLS) authentication**
- **OpenID Connect authentication (OAuth 2.0)**



ARHITECTURA DE SECURITATE

- Prezentăm un **exemplu de securizare a endpoint-urilor** care folosește **JPA ca IdentityProvider** ce permite activarea accesului pe roluri (**role-based access control – RBAC**)
- Exemplu se găsește în proiectul **security-jpa**
- **Numele utilizatorilor, parolele criptate și rolul** de care aparțin sunt stocate în baza de date MySql

Proiectul folosește următoarele extensii:

- **quarkus-resteasy**
- **quarkus-hibernate-orm-panache**
- **quarkus-narayana-jta**
- **quarkus-security-jpa**
- **quarkus-jdbc-mysql**

ARHITECTURA DE SECURITATE

```
@Entity  
@Table(name = "test_user")  
@UserDefinition  
public class User extends PanacheEntity {  
    @Username  
    public String username;  
    @Password  
    public String password;  
    @Roles  
    public String role;  
  
    public static void add(String username, String password, String role) {  
        User user = new User();  
        user.username = username;  
        user.password = BcryptUtil.bcryptHash(password);  
        user.role = role;  
        user.persist();  
    }  
}
```

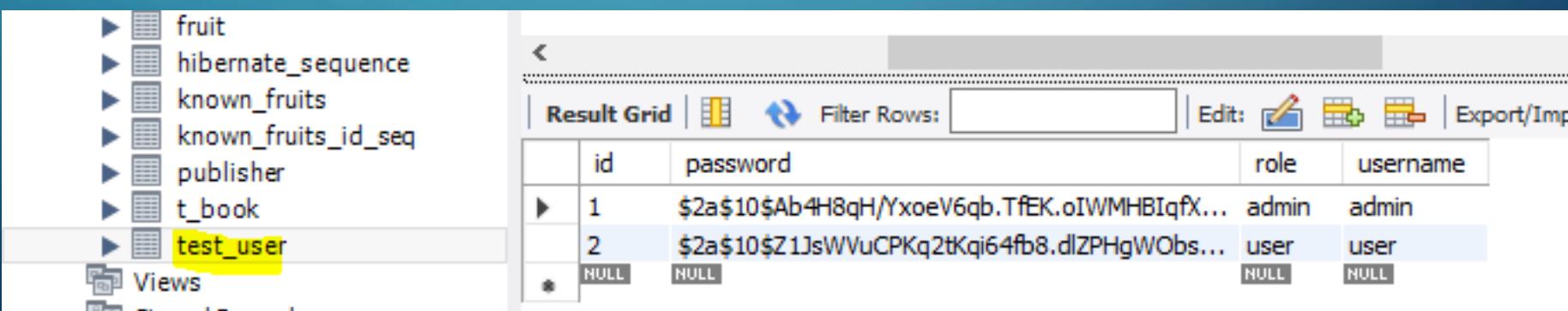
Se creează o entitate **User** care este decorat cu anotări specifice **Security JPA**. Parola este criptată și stocată ca hash.

ARHITECTURA DE SECURITATE

- Bean pentru adăugarea utilizatorilor inițiali în baza de date
- Inițial se sterg toate datele din baza de date

```
@Singleton
public class Startup {
    @Transactional
    public void loadUsers(@Observes StartupEvent evt) {
        // reset and load all test users
        User.deleteAll();
        User.add("admin", "admin", "admin");
        User.add("user", "user", "user");
    }
}
```

Clasă **Singleton** care primește evenimentul de pornire a aplicației în metoda **loadUsers()** și reinițializează tabelul cu utilizatori.



The figure consists of two parts. On the left, a schema browser shows a list of tables: fruit, hibernate_sequence, known_fruits, known_fruits_id_seq, publisher, t_book, and test_user. The 'test_user' table is highlighted with a yellow box. On the right, a result grid displays the data from the 'test_user' table. The grid has columns: id, password, role, and username. It contains two rows:

	id	password	role	username
▶	1	\$2a\$10\$Ab4H8qH/YxoeV6qb.TfEK.oIWMHBIqfx...	admin	admin
*	2	\$2a\$10\$Z1JsWVuCPKq2tKqi64fb8.dlZPHgWObs...	user	user

ARHITECTURA DE SECURITATE

```
ackage atm.paradigms;

import javax.annotation.security.PermitAll;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("api/public")
public class PublicResource {
    @GET
    @PermitAll
    @Produces(MediaType.TEXT_PLAIN)
    public String publicResource() {
        return "public";
    }
}
```

Endpoint cu acces public marcat cu **@PermitAll**.

ARHITECTURA DE SECURITATE

```
package atm.paradigms;

import javax.annotation.security.RolesAllowed;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("api/admin")
public class AdminResource {
    @GET
    @RolesAllowed("admin")
    @Produces(MediaType.TEXT_PLAIN)
    public String adminResource() {
        return "admin";
    }
}
```

Endpoint cu acces bazat pe roluri `@RolesAllowed("admin")`.

ARHITECTURA DE SECURITATE

```
package atm.paradigms;

import javax.annotation.security.RolesAllowed;

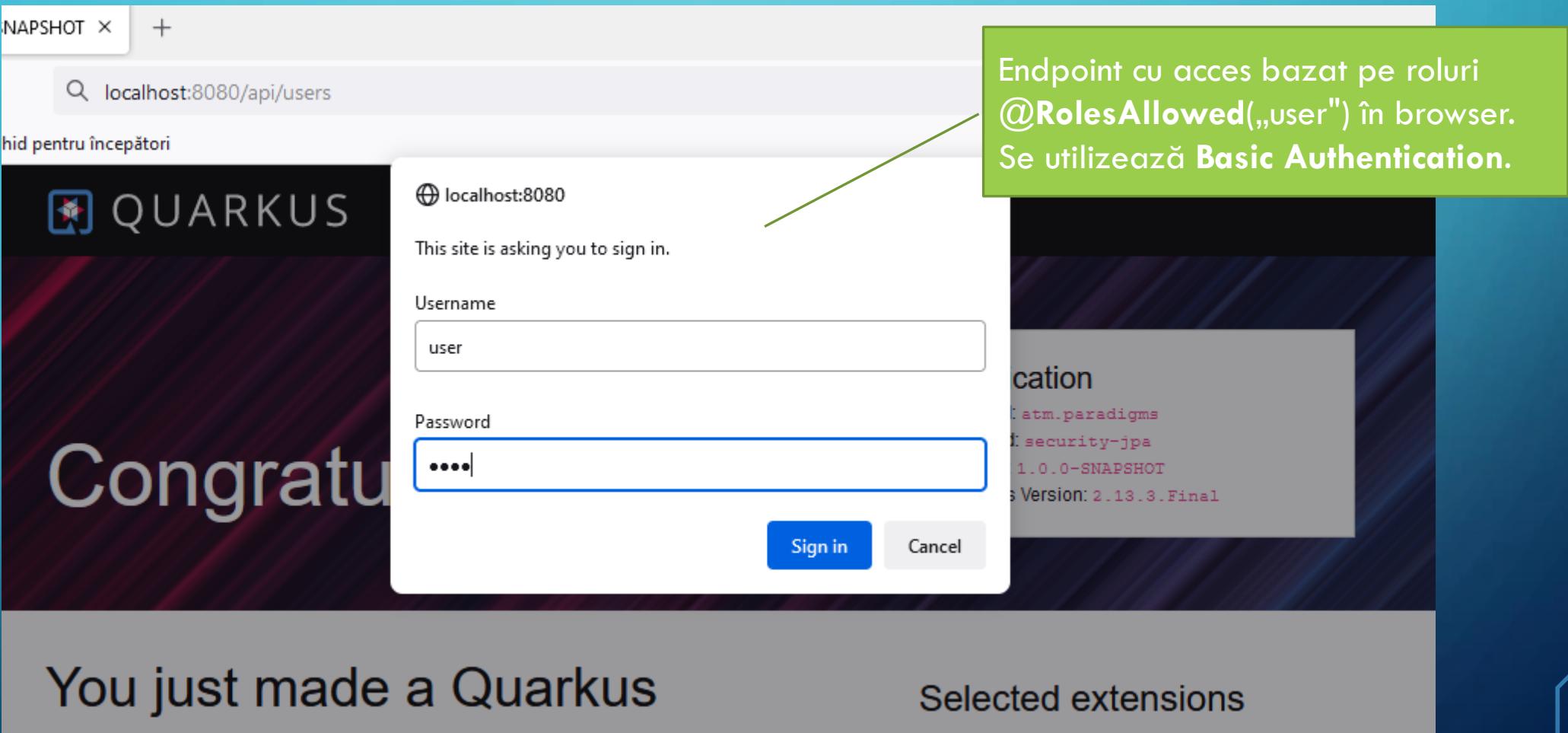
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.SecurityContext;

@Path("api/users")
public class UserResource {

    @GET
    @RolesAllowed("user")
    @Produces(MediaType.TEXT_PLAIN)
    public String showUser(@Context SecurityContext securityContext) {
        return securityContext.getUserPrincipal().getName();
    }
}
```

Endpoint cu acces bazat pe roluri
`@RolesAllowed(“user”)`.
Folosește adnotarea `@Context`
pentru a injecta `SecurityContext`
pentru a extrage numele utilizatorului.

ARHITECTURA DE SECURITATE



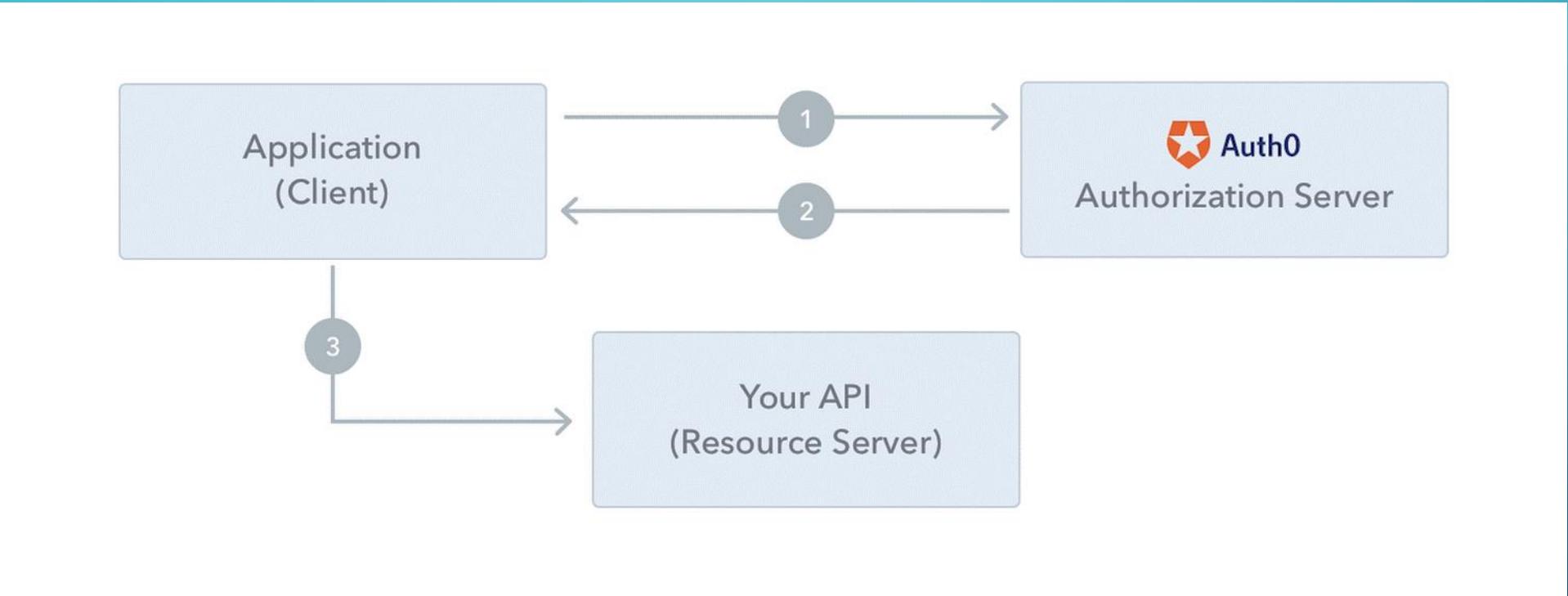


JSON WEB TOKEN

- **JSON Web Token (JWT)** este un standard deschis care definește o **modalitate compactă și autosuficientă** pentru a **transmite informații ca obiecte JSON**
- **Informațiile transmise pot fi verificate** deoarece sunt semnate digital cu **un secret (HMAC)** sau cu **o pereche de chei public/privată (RSA)**
- **Signed token** permite **verificarea integrității datelor** conținute (claims). Există și **token criptat**
- Dacă **token-ul este semnat** folosind **o pereche de chei public/privată**, semnătura certifică că doar de **partea care deține cheia privată o putea semna**
- **JWT** este folosit în principal **pentru autorizare**. O dată de **utilizatorul este logat**, **cererile ulterioare vor include token-ul** permitând acestuia să acceseze resurse
- **Single Sign On (SSO)** se bazează pe JWT



JSON WEB TOKEN



1. Clientul cere autorizarea la serverul de autorizare (OpenID Connect)
2. După autorizare serverul trimite clientului un token pentru acces la aplicație
3. Folosind token-ul clientul are acces la resursele protejate



JSON WEB TOKEN

- **Structura JWT:** *header, payload, signature* separare cu .
- **Header-ul** este codificat cu *Base64Url* și constă din:
 - *tipul de token* care este JWT
 - *algoritm de semnare* (HMAC SHA256 or RSA)
- **Payload-ul** este codificat cu *Base64Url* și constă din **date (claims) predefinite, publice și private**
- **Signature** se creează din **header-ul, payload-ul codificate, un secret** folosind algoritmul de semnare
- **Semnătura** se folosește și pentru **a verifica că mesajul nu a fost modificat**
- Pentru a accesa o resursă protejată **clientul trimite JWT într-un header-ul Authorization cu schema Bearer**

Authorization: Bearer <token>



Encoded

PASTE A TOKEN HERE

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3Mi0iJodHRwczovL2V4YW1wbGUuY29tL2lzc3VlciIsInVwbii6Impkb2VAcXVhcmt1cy5pbysIsImdyb3VwcyI6WyJVc2VyIiwiQWRtaW4iXSwiYmlydGhkYXRlIjoiMjAwMS0wNy0xMyIsImlhdCI6MTY2NjYwNTM4MywiZXhwIjoxNjY2NjA1NjgzLCJqdGkiOiJkZTF1M2Q4Yy02ZDVkLTRkMTctYjUxMy1kNzljOWRj0DczMzIifQ.cw2JWp6svQDUgypVMPydljtSUU9ySVR3Tco7hRXE6NnSBkhwlRK-KNj2EmNIgiUMZ2kjpti0yI7LLPsx3-bdCZesDw18I2XXI6h09DwQVkdB4Aohi1njmnMpVVTxTqQaWR9_058yipecrVpLckhZJrzhc_HvXxAtwABcONXIL-7-0P3T4BwoBVPUK2oXU-C1CQn2Hq_lclrGQ0YHjJHRuTqF1aXrg9e8rAUD4nIP07mupc_HRQi-QKV0xDngMjy4kZXb2cZji6c76PRqI1i9fBWY7KqX10Jdr0NY3Hu3TdPRYQInWcxs1IUvNjB9J7wI9eJu-gHLiASpiomTCCG0Q|
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "typ": "JWT",  
  "alg": "RS256"  
}
```

PAYOUT: DATA

```
{  
  "iss": "https://example.com/issuer",  
  "upn": "jdoe@quarkus.io",  
  "groups": [  
    "User",  
    "Admin"  
  ],  
  "birthdate": "2001-07-13",  
  "iat": 1666605383,  
  "exp": 1666605683,  
  "jti": "de1e3d8c-6d5d-4d17-b513-d79c9dc87332"  
}
```

VERIFY SIGNATURE

RSASHA256(
 base64UrlEncode(header) + "." +
 base64UrlEncode(payload),

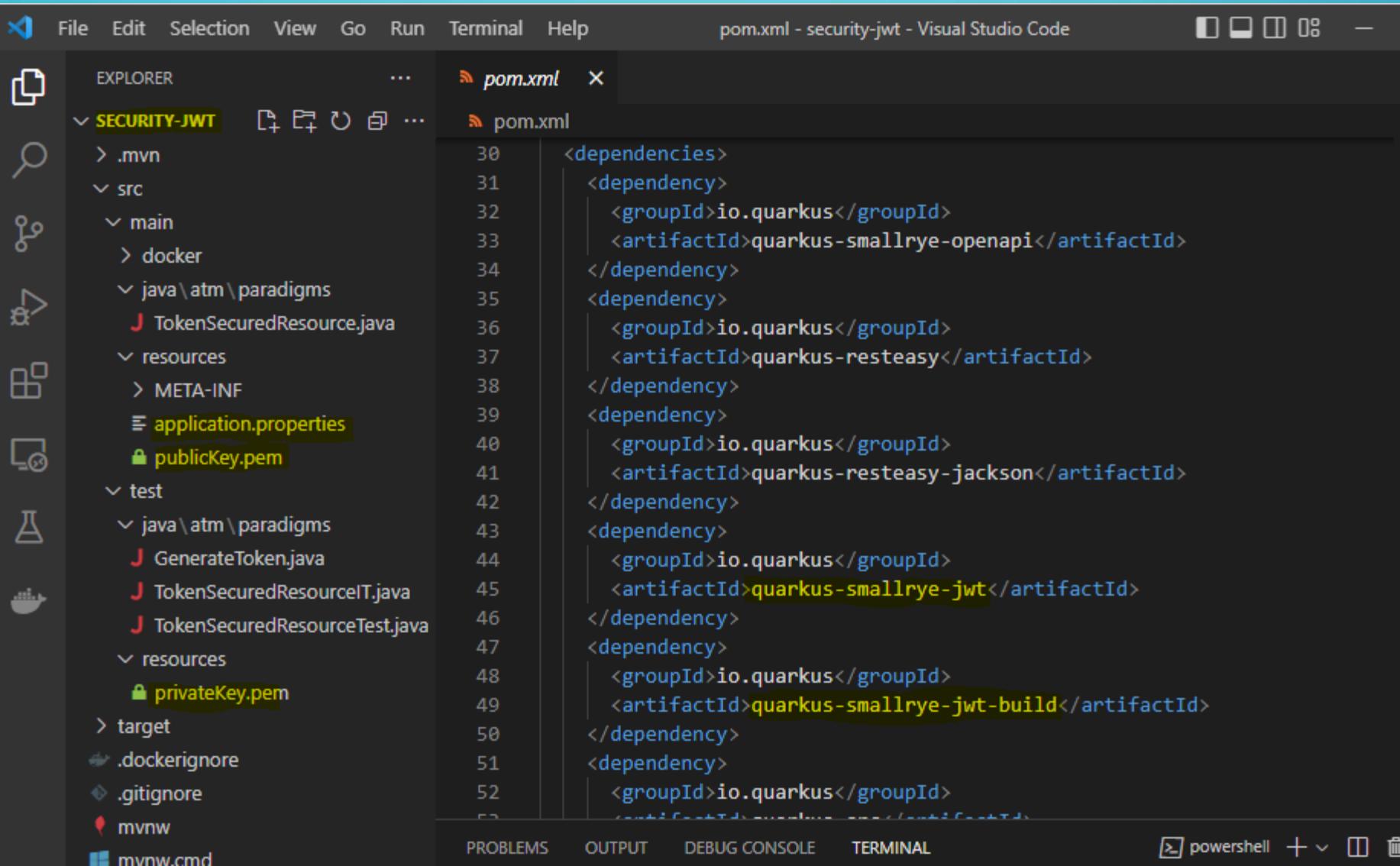
Public Key in SPKI, PKCS #1,
 X.509 Certificate, or JWK string
 format.

Private Key in PKCS #8, PKCS #1,
 or JWK string format. The key
 never leaves your browser.

Cuprins

JSON WEB TOKEN

- Demonstrarea modului de utilizare a **SmallRye JWT** pentru a **verifica token-uri** și a **asigura acces securizat la endpoint-uri** se găsește în proiectul **security-jwt**



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "SECURITY-JWT". Key files include `pom.xml`, `application.properties`, `publicKey.pem`, `privateKey.pem`, and several Java test files like `GenerateToken.java`, `TokenSearchedResourceT.java`, and `TokenSearchedResourceTest.java`.
- Code Editor:** Displays the `pom.xml` file with the following dependency declarations:

```
<dependencies>
    <dependency>
        <groupId>io.quarkus</groupId>
        <artifactId>quarkus-smallrye-openapi</artifactId>
    </dependency>
    <dependency>
        <groupId>io.quarkus</groupId>
        <artifactId>quarkus-resteasy</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-smallrye-jwt-build</artifactId>
    </dependency>
    <dependency>
        <groupId>io.quarkus</groupId>
```
- Bottom Bar:** Shows tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL, along with icons for powershell, terminal, and file operations.



JSON WEB TOKEN

- Informațiile pentru configurarea de securitate se găsesc în fișierul ***application.properties***

```
# Public verification key
mp.jwt.verify.publickey.location=publicKey.pem
quarkus.native.resources.includes=publicKey.pem
# Required issuer
mp.jwt.verify.issuer=https://example.com/issuer

# Private signing key
smallrye.jwt.sign.key.location=privateKey.pem
```

- Se setează locația cheii primare. Specificația **MicroProfile JWT RBAC** cere ca token-urile să fie semnate cu algoritmul **RSA256**. Cheia publică RSA se folosește pentru verificarea semnăturii
- Pentru generarea semnăturii este nevoie și de cheia privată pereche aflată în locația **/src/test/resources/privateKey.pem**



JSON WEB TOKEN

- În mod normal **token-ul** se obține de la un **manager de identitate** precum **Keycloak**
- Pentru a demonstra **securitatea JWT** în **Quarkus** generăm propriile token-uri
- **JWT token** are un număr de date (**claims**):
- **iss** – **emitentul token-ului**. Pentru ca token-ul să fie valid valoare **trebuie să se potrivească** cu valoarea proprietății **mp.jwt.verify.issuer** de pe server
- **upn** – definit de specificația **MicroProfile JWT RBAC** ca **Principal** (nume utilizator)
- **group** - rolurile asociate cu JWT
- **birthday** – dată customizată



JSON WEB TOKEN

- Clasa de test generează și tipărește la consolă 3 JWT corespunzătoare rolurilor: **User + Admin, Admin, User**
- Aceste token-uri vor fi folosite pentru accesarea endpoint-urilor REST securizate

```
@QuarkusTest
public class GenerateToken {
    /**
     * Generate JWT token
     */
    @Test
    public void getUserAdminToken() {
        String token = Jwt.issuer("https://example.com/issuer")
            .upn("jdoe@quarkus.io")
            .groups(new HashSet<>(Arrays.asList("User", "Admin")))
            .claim(Claims.birthdate.name(), "2001-07-13")
            .sign();
        System.out.println("User Admin:\n\n" + token);
        assertNotNull(token);
    }
    ...
}
```



JSON WEB TOKEN

[Cuprins](#)

```
@Test
public void getAdminToken() {
    String token = Jwt.issuer("https://example.com/issuer")
        .upn("jdoe@quarkus.io")
        .groups("Admin")
        .claim(Claims.birthdate.name(), "2001-07-13")
        .sign();
    System.out.println("Admin:\n\n" + token);
    assertNotNull(token);
}

@Test
public void getUserToken() {
    String token = Jwt.issuer("https://example.com/issuer")
        .upn("jdoe@quarkus.io")
        .groups("User")
        .claim(Claims.birthdate.name(), "2001-07-13")
        .sign();
    System.out.println("User:\n\n" + token);
    assertNotNull(token);
}
```



JSON WEB TOKEN

[Cuprins](#)

- Clasa resursă *TokenSecuredResource* folosește JWT API pentru accesarea datele din token (claims) trimis de client

```
@Path("secured")
@RequestScoped
public class TokenSecuredResource {
    @Inject
    JsonWebToken jwt;
    @Inject
    @Claim(standard = Claims.birthdate)
    String birthdate;

    @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();
    }
}
```

Injectează **JsonWebToken** ce va fi folosit pentru a accesa datele din token-ul trimis de client.
Se injectează claim-ul customizat **birthday**.
Observăm adnotările de securitate JAX-RS.

JSON WEB TOKEN

```
@GET  
@Path("roles-allowed-admin")  
@RolesAllowed("Admin")  
@Produces(MediaType.TEXT_PLAIN)  
public String helloRolesAllowedAdmin(@Context SecurityContext ctx) {  
    return getResponseString(ctx) + ", birthdate: " + birthdate;  
}  
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;  
}
```

Metoda **getAsString()** folosește **SecurityContext** și **JWT API** pentru a extrage și afișa datele conținute în token.



JSON WEB TOKEN

- Clasă de test pentru verificarea endpoint-urilor REST. Sunt generate tokenuri pentru a le accesa

```
-----  
 @Test  
 public void testHelloRolesAllowedAdminOnlyWithUserRole() {  
     Response response = given().auth()  
         .oauth2(generateValidUserToken())  
         .when()  
         .get("/secured/roles-allowed-admin").andReturn();  
  
     response.then().statusCode(403);  
 }  
-----  
 static String generateValidUserToken() {  
     return Jwt.upn("jdoe@quarkus.io")  
         .issuer("https://example.com/issuer")  
         .groups("User")  
         .claim(Claims.birthdate.name(), "2001-07-13")  
         .sign();  
 }
```



JSON WEB TOKEN

- Pornim aplicația în mod dezvoltare cu comanda **quarkus dev**
- În **consolă** se **apasă o apoi r** pentru a executa testele și a afișa la consolă rezultatele. Se afișează token-urile

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
java + ▾ ⌂ ⌂ ... ^ ×

eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJodHRwczovL2V4YW1wbGUuY29tL2lzc3VlcIsInVwbii6Impkb2VAcXhcm1cy5pbysImdyb3WcyI6WyJvc2VyIl0sImJpcnRoZGF0ZSI6IjIwMDEtMDctMTMiLCJpYXQiOjE2NjY20DYxNzksImV4cCI6MTY2NjY4NjQ3OSwianRpIjoiNTBkY2VjMTItNjZjNy00NjcxLWJiN2ItMGI3MmJiOWYyNDg0In0.hSJTCAFO6yZWCa_qSQ2eCEyu75T6h60j3WsSTITu56eC8AXeJmItgR40T8Nm0W3zd3VqkcQoM10ri4js4FhogJWjm60YmkUjMnY7cKwXE9WBbinejp2VqGedw1leyG3yNjhJoc4UeV3ZrT1UGQ1hX4ENoRr0MkfX8qkuBo6c_M5PF863N4NiB-Mfn3No9W8qj9qw1Uz58u06jQFIjD7RnpBKJjXJG2dRnIFYF0oVeEq-T5bA7xCuvvHL_MMcdfdd42JN0T46rYHJBHwkoC6GGqa6hM3D2eydbD5gJupi0n6RXD9uWpmpZfI1bz3c9YyHpIuIw5K6oGqg8X6-VKxB58RCSw

2022-10-25 11:22:59,348 INFO [io.qua.test] (Test runner thread) Running 3/12. Running: atm.paradigms.GenerateToken#getUserAdminToken()
```

- Se accesează **Swagger UI** la <http://localhost:8080/q/swagger-ui/>
- Se **apasă butonul Authorize** și se **copiază token-ul** dorit din consola Quarkus, apoi se testează endpoint-ul



JSON WEB TOKEN

[Cuprins](#)

Swagger UI

/q/openapi

Explore

security-jwt (powered by Quarkus)

security-jwt API

/q/openapi

1.0.0-SNAPSHOT

Token Secured Resource

GET /secured/deny-all

GET /secured/permit-all

GET /secured/roles-allowed

GET /secured/roles-allowed-admin

Available authorizations

SecurityScheme (http, Bearer)

Authentication

Value:

V5K6oGq8X6-VKxBS8RCSwI

Authorize



Authorize

Close



GET /secured/roles-allowed

Parameters Cancel

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/secured/roles-allowed' \
  -H 'accept: text/plain' \
  -H 'Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJodHRwczovL2V4YW1wbGUuY29tL2lzc3VlciIsInVwbii6Impkb2VAcXVhcmt1cy5pbysImdyb3WcycI6MyJVc2VyI10' < /dev/null > response.txt
```

Request URL

```
http://localhost:8080/secured/roles-allowed
```

Server response

Code	Details
200	<p>Response body</p> <pre>hello + jdoe@quarkus.io, isHttps: false, authScheme: Bearer, hasJWT: true, birthdate: 2001-07-13</pre> <p>Copy Download</p> <p>Response headers</p> <pre>content-length: 96 content-type: text/plain; charset=UTF-8</pre>

GET /secured/roles-allowed-admin



Parameters

[Cancel](#)

No parameters

[Execute](#)

[Clear](#)

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8080/secured/roles-allowed-admin' \
-H 'accept: text/plain' \
-H 'Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJodHRwczovL2V4YW1wbGUuY29tL2lzc3VlcIisInVwbii6Impkb2VAcXhcmt1cy5pbysImdyb3VwcyI6WyJVc2VyIi0e...' < >
```

Request URL

<http://localhost:8080/secured/roles-allowed-admin>

Server response

Code

Details

401

Error: Unauthorized

Response headers

```
content-length: 0
www-authenticate: Bearer
```

BIBLIOGRAFIE

- **Understanding Quarkus**, Antonio Goncalves, Independently Published, 2020
- **Quarkus Cookbook**, Alex Soto Bueno and Jason Porter, O'Reilly Media, 2020
- **Beginning Quarkus Framework**, Tayo Koleoso, Apress, 2020
- [Quarkus Guides –Latest](#)
- [JSON Web Token](#)