Software Architecture and Methodologies

Corso di Laurea Magistrale in Ingegneria Informatica

Università degli Studi di Firenze

Online Videogame

Victor Soto Berenguer (7078230)

2022

Index

Introduction

In this project I am extending the application developed for the course of *Ingegneria del Software* of the *Laurea Triennale* as the back end of a Restful architecture.

The project is connected to a **database** in localhost called “assignment-restful-architecture” with **user** “java-client” and **password** “password”.

Tools

The application was developed in Java with the IDE Eclipse (for Enterprise Java and Web developers), as a Maven project.

JPA was used to define how the entities of the Domain Model should be mapped in the database.

CDI was used to manage the lifecycle of different components.

JAX-RS was used to expose endpoint services.

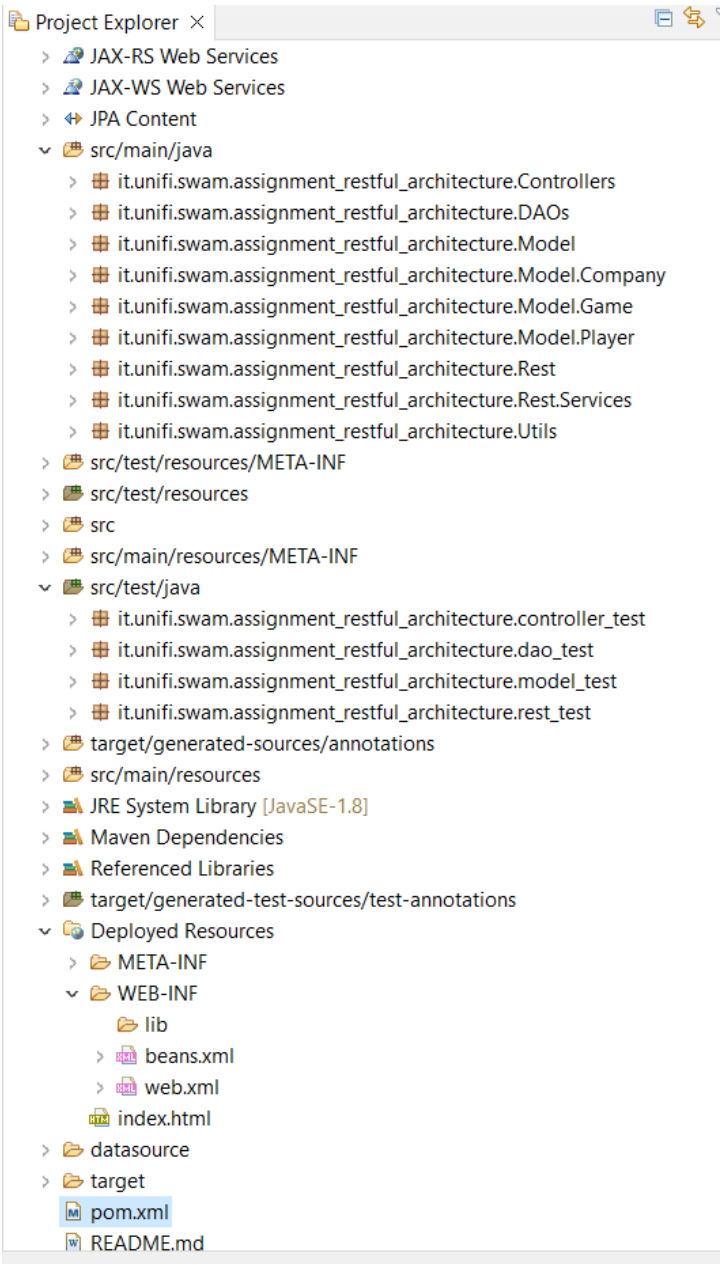
Finally, JUnit was used to perform unit tests, along with several tools like Hamcrest, Rest Assured, or Mockito.

All dependencies included in the pom.xml file are what follows:

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-api</artifactId>
        <version>8.0.1</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.4.13.Final</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <version>1.4.195</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>com.auth0</groupId>
        <artifactId>java-jwt</artifactId>
        <version>3.19.2</version>
    </dependency>
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt</artifactId>
        <version>0.7.0</version>
    </dependency>
    <dependency>
        <groupId>com.lambdaworks</groupId>
        <artifactId>scrypt</artifactId>
        <version>1.4.0</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-crypto</artifactId>
        <version>5.7.1</version>
    </dependency>
</dependencies>
```

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <version>3.12.0</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.6</version>
</dependency>
<dependency>
    <groupId>com.jayway.restassured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>2.9.0</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest-all</artifactId>
    <version>1.3</version>
</dependency>
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>3.9.0</version>
    <scope>test</scope>
</dependency>
<dependency>
    <dependency>
        <groupId>org.json</groupId>
        <artifactId>json</artifactId>
        <version>20220320</version>
    </dependency>
</dependencies>
```

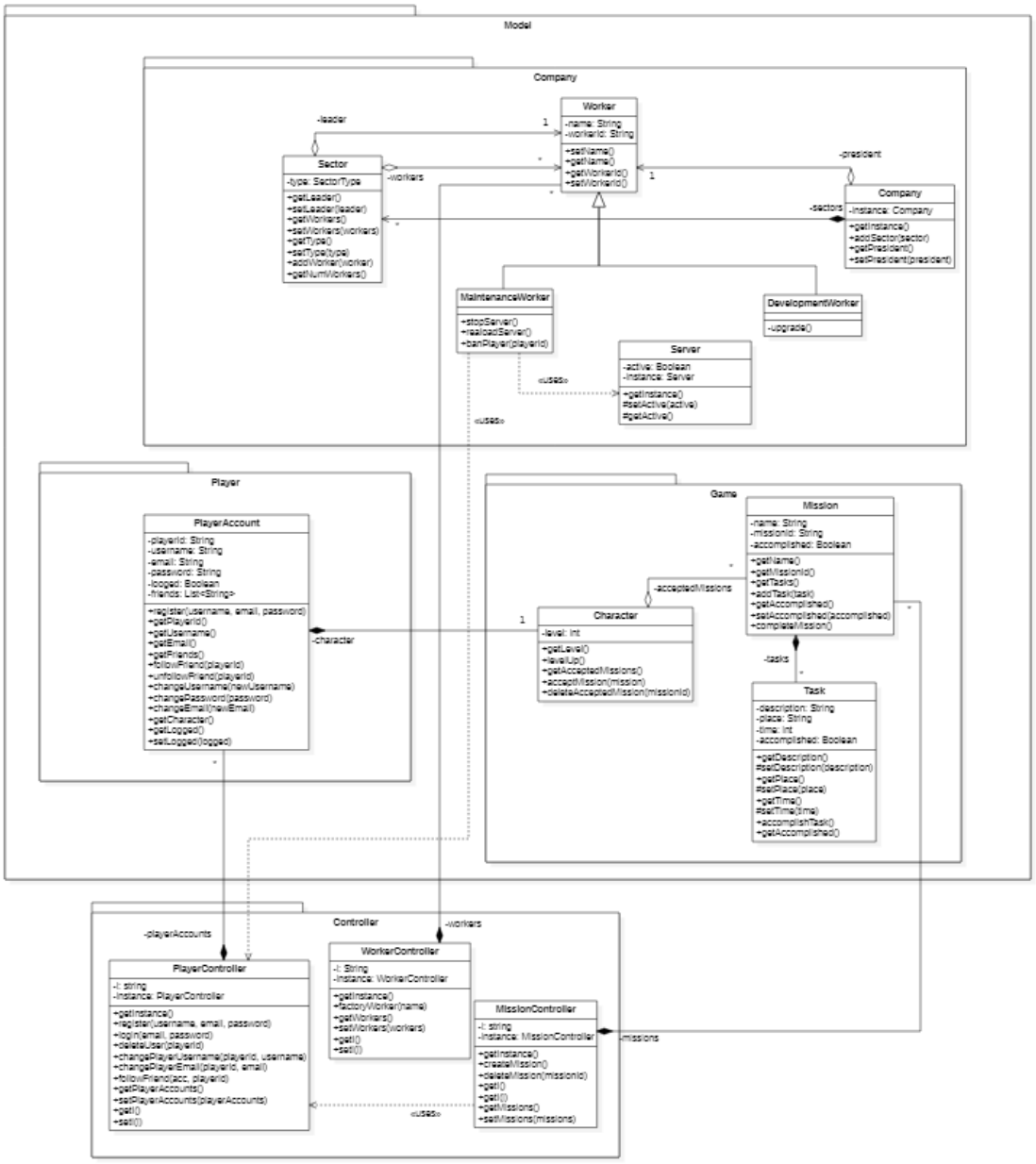
Project Organization



- src/main/java: contains entities of Domain Model, DAOs for each entity, Controllers, Rest Services, as well as Utility Classes.
- src/main/resources/META-INF: contains the persistence.xml file to the MySql database.
- src/test/java: contains the unit tests for the entities, DAOs, Controllers and Endpoints.
- src/test/resources/META-INF: contains the persistence.xml file a simple database in memory to execute some of the tests.
- WEB-INF: contains the beans.xml and web.xml files. The web.xml defines the Roles existing in the application, which are then used to defined who is authorized to do certain operations.

Domain Model

Here it is the Class Diagram presented for the previous assignment, which defines the Structure of the Domain Model (Although with some minor changes/additional elements).



In the package Model we can find the classes ModelFactory and BaseEntity.

Base Entity is an abstract class which only defines the ID and the UUID. All entities will extend this class.

ModelFactory creates new instances of the entities in the DomainModel and generates random UUIDs for them.

```java
public class ModelFactory {
    private ModelFactory() {}

    public static Company company() {
        return new Company(UUID.randomUUID().toString(), null, null);
    }

    public static Worker worker() {
        return new Worker(UUID.randomUUID().toString(), "", "", "");
    }

    public static Worker worker(Worker worker) {
        return new Worker(worker);
    }

    public static Sector sector() {
        return new Sector(UUID.randomUUID().toString(), null);
    }

    public static Server server() {
        return new Server(UUID.randomUUID().toString(), null);
    }

    public static PlayerAccount playeracc() {
        return new PlayerAccount(UUID.randomUUID().toString());
```

```java
@MappedSuperclass
public abstract class BaseEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String uuid;

    protected BaseEntity() {}
    public BaseEntity(String uuid) {
        if(uuid == null) {
            throw new IllegalArgumentException("uuid cannot be null");
        }
        this.uuid = uuid;
    }

    @Override
    public boolean equals(Object obj) {
        if(this==obj) {
            return true;
        }
        if(obj==null) {
            return false;
        }
        if(!(obj instanceof BaseEntity)) {
            return false;
```

In the rest of the packages (Model.*) we can find the entities with the JPA annotations:

Examples:

```java
@Entity
public class Worker extends BaseEntity {
    @Column(nullable = false)
    private String name;

    @Column(nullable = false, unique = true)
    private String email;

    @Column(nullable = false)
    private String password;

    @Enumerated(EnumType.STRING)
    private WorkerRol workerRol;

    Worker() {}

    public Worker(String uuid, String name, String email, String password) {
        super(uuid);
        this.name = name;
        this.email = email;
        this.password = password;
        this.setWorkerRol(null);
    }
```

```java
@Entity
public class Mission extends BaseEntity{
    @Column(nullable = false, unique = true)
    private String name;

    @OneToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinTable(name="mission_tasks",joinColumns=@JoinColumn(name="missionId"),inverseJoinColumns=@JoinCo
            name = "taskId",
            foreignKey = @ForeignKey(
                name = "FK_TASKID",
                foreignKeyDefinition = "FOREIGN KEY (taskId) REFERENCES task(id) ON DELETE CASCADE"
        )))
    @OrderBy
    private List<Task> tasks;

    private Boolean accomplished;

    Mission() {}

    public Mission(String uuid, String name, List<Task> tasks) {
        super(uuid);
        this.name = name;
        this.tasks = tasks;
        this.setAccomplished(false);
    }
```

```java
@Entity
public class Sector extends BaseEntity{
    @OneToOne(cascade = CascadeType.PERSIST)
    @JoinColumn(
            name = "leader",
            foreignKey = @ForeignKey(
                name = "FK_LEADER",
                foreignKeyDefinition = "FOREIGN KEY (leader) REFERENCES worker(id) ON DELETE SET NULL"
        ))
    private Worker leader;

    @OneToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinTable(name="sector_workers",joinColumns=@JoinColumn(
            name = "sectorId",
            foreignKey = @ForeignKey(
                name = "FK_SECTORID",
                foreignKeyDefinition = "FOREIGN KEY (sectorId) REFERENCES sector(id) ON DELETE CASCADE"
        )),inverseJoinColumns=@JoinColumn(
            name = "workerId",
            foreignKey = @ForeignKey(
                name = "FK_WORKERID",
                foreignKeyDefinition = "FOREIGN KEY (workerId) REFERENCES worker(id) ON DELETE CASCADE"
        )))
    @OrderBy
    private List<Worker> workers;
```

Different types of annotations have been used such as: @Column to define characteristics of that column for the table (like not nullable or unique) or annotations for associations such as @OneToMany, @ManyToMany, etc.

Entities Worker and PlayerAccount make use of the library BScrypt to encrypts the passwords and/or verify them.

DAOs

It is defined a DAO for each entity in the Domain Model.
These DAOs have an EntityManager injected through the annotation @PersistenceContext which they use to perform several operations like persist, find, delete, or even different queries.
Example:

```java
@SessionScoped
@Default
public class WorkerDAO implements Serializable {
    static final long serialVersionUID = 1L;

    @PersistenceContext
    private EntityManager entityManager;

    public void save(Worker worker) {
        if(worker.getId() != null) {
            entityManager.merge(worker);
        }else {
            entityManager.persist(worker);
        }
    }

    public void delete(Worker worker) {
        this.entityManager.remove(
                this.entityManager.contains(worker) ? worker :
                this.entityManager.merge(worker));
    }

    public Worker findById(Long id) {
        return entityManager.find(Worker.class, id);
    }

    public List<Worker> getAllWorkers() {
        return this.entityManager.createQuery(
                "FROM Worker", Worker.class)
                .getResultList();
    }

    public List<Worker> findWorkerByName(String name) {
        List<Worker> result = this.entityManager.createQuery(
                "FROM Worker WHERE name = :name", Worker.class)
        .setParameter("name", name)
        .getResultList();

        if(result.isEmpty()) {
            return null;
        }else {
            return result;
        }
    }

    public Worker getWorkerByEmail(String email) {
        List<Worker> result = this.entityManager.createQuery(
                "FROM Worker WHERE email = :email", Worker.class)
        .setParameter("email", email)
        .getResultList();

        if(result.isEmpty()) {
            return null;
        }else {
            return result.get(0);
        }
    }
```

Controllers

Controllers can create new instances of entities of the DomainModel and also make use of the previously defined DAOs to perform operations with these instances.

DAOs are injected into the Controllers with the CDI annotation @Inject.

Example:

```
@SessionScoped
@Named
public class WorkerController implements Serializable{
    private static final long serialVersionUID = 1L;

    @Inject
    WorkerDAO workerDao;

    public Worker getById(Long id) {
        if(id==null) {
            throw new IllegalArgumentException("Id cannot be null");
        }else {
            return workerDao.findById(id);
        }
    }

    public Worker saveWorker(Worker worker, Boolean encrypted) {
        Worker workerToPersist = ModelFactory.worker();

        workerToPersist.setName(worker.getName());
        workerToPersist.setEmail(worker.getEmail());
        workerToPersist.setPassword(worker.getPassword(), encrypted);

        workerToPersist.setWorkerRol(worker.getWorkerRol());

        workerDao.save(workerToPersist);
        return workerToPersist;
    }

    public void updateWorker(Worker workerToUpdate, Worker updates, Boolean encrypted) {
        if(updates.getName()!=workerToUpdate.getName() && updates.getName()!=null) {
            workerToUpdate.setName(updates.getName());
        }

        if(updates.getEmail()!=workerToUpdate.getEmail() && updates.getEmail()!=null) {
            workerToUpdate.setEmail(updates.getEmail());
        }

        if(updates.getPassword()!=workerToUpdate.getPassword() && updates.getPassword()!=null) {
            workerToUpdate.setPassword(updates.getPassword(), encrypted);
        }

        if(updates.getWorkerRol()!=workerToUpdate.getWorkerRol() && updates.getWorkerRol()!=null) {
            workerToUpdate.setWorkerRol(updates.getWorkerRol());
        }

        workerDao.save(workerToUpdate);
    }

    public void delete(Worker worker) {
        workerDao.delete(worker);
    }

    public List<Worker> getAll() {
        return workerDao.getAllWorkers();
    }

    public Worker getWorke
```

```
public Worker getWorkerByEmail(String email) {
    return workerDao.getWorkerByEmail(email);
}
```

Endpoints

In the package Rest we can find some useful classes that will be used later for the Endpoints:

- MyRestApplication: defines the path of the endpoints.

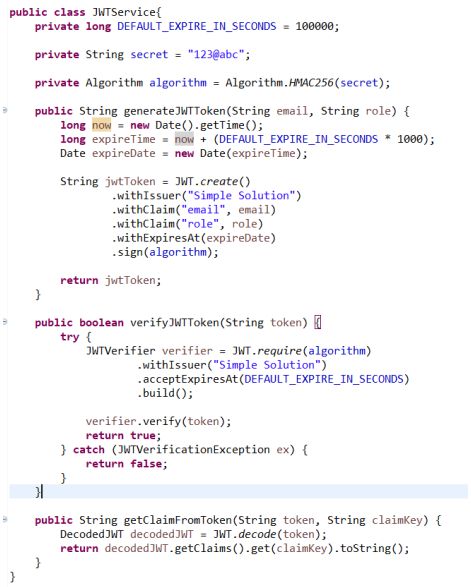
MyRestApplication.java

```
1 package it.unifi.swam.assignment_restful_architecture.Rest;
2
3 import javax.ws.rs.ApplicationPath;
5
6 @ApplicationPath("/rest")
7 public class MyRestApplication extends Application {
8 }
```

- ResponseFilter: allows CORS among other uses.

```java
package it.unifi.swam.assignment_restful_architecture.Rest;

import java.io.IOException;

@Provider
public class ResponseFilter implements ContainerResponseFilter {
    @Override
    public void filter(ContainerRequestContext requestContext,
            ContainerResponseContext responseContext) throws IOException {

        responseContext.getHeaders().add("Access-Control-Allow-Origin", "*");
        responseContext.getHeaders().add("Access-Control-Allow-Credentials", "true");
        responseContext.getHeaders().add("Access-Control-Allow-Methods", "GET, POST, DELETE, PUT");

        //allows CORS (CROSS-ORIGIN RESOURCE SHARING)
    }
}
```

- JWTService: defines method to create JWT Tokens or verify a token that was passed as a parameter

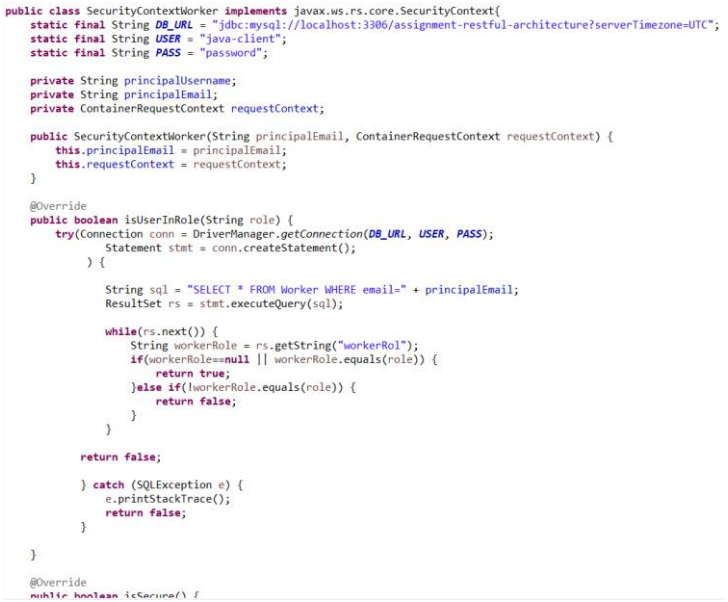


```java
public class JWTService{
    private long DEFAULT_EXPIRE_IN_SECONDS = 100000;

    private String secret = "123@abc";

    private Algorithm algorithm = Algorithm.HMAC256(secret);

    public String generateJWTToken(String email, String role) {
        long now = new Date().getTime();
        long expireTime = now + (DEFAULT_EXPIRE_IN_SECONDS * 1000);
        Date expireDate = new Date(expireTime);

        String jwtToken = JWT.create()
                .withIssuer("Simple Solution")
                .withClaim("email", email)
                .withClaim("role", role)
                .withExpiresAt(expireDate)
                .sign(algorithm);

        return jwtToken;
    }

    public boolean verifyJWTToken(String token) {
        try {
            JWTVerifier verifier = JWT.require(algorithm)
                    .withIssuer("Simple Solution")
                    .acceptExpiresAt(DEFAULT_EXPIRE_IN_SECONDS)
                    .build();

            verifier.verify(token);
            return true;
        } catch (JWTVerificationException ex) {
            return false;
        }
    }

    public String getClaimFromToken(String token, String claimKey) {
        DecodedJWT decodedJWT = JWT.decode(token);
        return decodedJWT.getClaims().get(claimKey).toString();
    }
}
```

- SecurityRequestFilter: makes sure that Authorization is performed for every request. When performing login or singup of a worker or player we will receive a JWT Token which we will have to pass as Authorization Header for every request. In this case the Authorization is of type Bearer.

```java
@Provider
@Priority(Priorities.AUTHENTICATION)
public class SecurityRequestFilter implements ContainerRequestFilter{

    @Override
    public void filter(ContainerRequestContext requestContext) throws IOException {
        String authorizationHeader = requestContext.getHeaderString(HttpHeaders.AUTHORIZATION);
        if(authorizationHeader == null || authorizationHeader.isEmpty()) {
            if(requestContext.getUriInfo().getPath().contains("login")) {
                System.out.println("Performing login");
            }else if(requestContext.getUriInfo().getPath().contains("signup")) {
                System.out.println("Performing signup");
            }else {
                //Without and AUTHORIZATION Header, a client is considered UNAUTHORIZED
                requestContext.abortWith(
                        Response.status(Response.Status.UNAUTHORIZED)
                        .entity("This Request is UNAUTHORIZED!")
                        .type("text/plain")
                        .build()
                );
            }

        }else {
            String email;
            if(authorizationHeader.startsWith("Bearer") || authorizationHeader.startsWith("BEARER")) {
                String token = authorizationHeader.substring("Bearer".length()).trim();
                JWTService jwtService = new JWTService();
                Boolean verified = jwtService.verifyJWTToken(token);

                if(verified) {
                    email = jwtService.getClaimFromToken(token, "email");
                    String role = jwtService.getClaimFromToken(token, "role");
                    System.out.println(role);
                    System.out.println(role.equals("User"));
                    System.out.println(role=="User");
                    System.out.println(role.toLowerCase().trim().equals("User".toLowerCase().trim()));
                    System.out.println(role.replace('"', ' ').trim().equals("User".replace('"', ' ').trim()))
                    if(role.replace('"', ' ').trim().equals("User".replace('"', ' ').trim())) {
                        System.out.println("test encuentra al usuario y lo mete en context");
                        requestContext.setSecurityContext(new SecurityContextPlayer(email, requestContext));
                    }else {
                        requestContext.setSecurityContext(new SecurityContextWorker(email, requestContext));
                    }
```

- SecurityContextPlayer/SecurityContextWorker: after the Token is passed and verified in the SecurityRequestFilter, a Player or Worker (depending on what the user of the Token is) is set. With this class and the annotation @RolesAllowed in the Endpoints, the methods will only be performed if the role of the user identified in the Token is allowed for that method.
  Players have role “User” whereas Workers can have different roles like “Development”, “Mainteinance” or “All” if no role has been set when creating that worker.
  The method isUserInRole is the one used automatically along with the @RolesAllowed annotation.



```
public class SecurityContextWorker implements javax.ws.rs.core.SecurityContext{
    static final String DB_URL = "jdbc:mysql://localhost:3306/assignment-restful-architecture?serverTimezone=UTC";
    static final String USER = "java-client";
    static final String PASS = "password";

    private String principalUsername;
    private String principalEmail;
    private ContainerRequestContext requestContext;

    public SecurityContextWorker(String principalEmail, ContainerRequestContext requestContext) {
        this.principalEmail = principalEmail;
        this.requestContext = requestContext;
    }

    @Override
    public boolean isUserInRole(String role) {
        try(Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
                Statement stmt = conn.createStatement();
            ) {

                String sql = "SELECT * FROM Worker WHERE email=" + principalEmail;
                ResultSet rs = stmt.executeQuery(sql);

                while(rs.next()) {
                    String workerRole = rs.getString("workerRol");
                    if(workerRole==null || workerRole.equals(role)) {
                        return true;
                    }else if(!workerRole.equals(role)) {
                        return false;
                    }
                }

            return false;

            } catch (SQLException e) {
                e.printStackTrace();
                return false;
            }

    }

    @Override
    public boolean isSecure() {
```

Finally, in the package Rest.Services we can find the different endpoints for each entity. These endpoints use annotations like @Path or @PathParam to specify the path to access that endpoint and/or take the param in the path to use it in the method. The basic methods are @GET, @POST, @PUT, @DELETE.

Also, annotations @Consumes and @Produces defines what content is expected at the request or at the response.

The endpoints for Worker and PlayerAccount have login and signup methods, which returns the Token that will be used in subsequent requests.

Login methods used BScrypt library to check if the encrypted password coincides with the one introduced by the user.

Examples:

```java
@Path("/workerendpoint")
public class WorkerEndpoint {

    @Inject
    WorkerController workerController;

    @GET
    @Path("/ping")
    public Response ping() {
        return Response.ok().entity("WorkerEndpoint is ready").build();
    }

    @POST
    @Path("/login")
    @Consumes({ MediaType.APPLICATION_JSON})
    @Produces({MediaType.APPLICATION_JSON})
    public Response login(String json) {
        ObjectMapper mapper = new ObjectMapper();
        try {
            if(json.isEmpty()) {
                return Response.status(Status.BAD_REQUEST).entity(mapper.writeValueAsString("Email and Password cannot be null")).build();
            }

            Worker worker = mapper.readValue(json, Worker.class);
            if(worker.getEmail()==null||worker.getPassword()==null) {
                return Response.status(Status.BAD_REQUEST).entity(mapper.writeValueAsString("Email and Password cannot be null")).build();
            }

            JSONObject obj = new JSONObject(json);
            String password = obj.getString("password"); //mapper already encrypts the password so it is not useful because then we will compare two different encrypted passwords

            Worker retrievedWorker = workerController.getWorkerByEmail(worker.getEmail());
            if(retrievedWorker==null) {
                return Response.status(Status.NOT_FOUND).entity(mapper.writeValueAsString("No Worker found with this email")).build();
            }else if(!BCrypt.checkpw(password, retrievedWorker.getPassword())) {
                return Response.status(Status.BAD_REQUEST).entity(mapper.writeValueAsString("Incorrect password")).build();
            }

            JWTService jwtService = new JWTService();
            String role;
            if(retrievedWorker.getWorkerRol()==null) {
                role = "All";
            }else {
                role = retrievedWorker.getWorkerRol().toString();
            }
            String token = jwtService.generateJWTToken(retrievedWorker.getEmail(), role);
            return Response.ok(token, MediaType.TEXT_PLAIN).build();

            } catch (IOException e) {
               e.printStackTrace();
```

```java
    @GET
    @Produces({MediaType.APPLICATION_JSON})
    @RolesAllowed({"All","Developer","Mainteinance"})
    public Response getAllWorkers() {
        ObjectMapper mapper = new ObjectMapper();
        try {
            List<Worker> workers = workerController.getAll();
            String json = mapper.writeValueAsString(workers);
            return Response.ok(json, MediaType.APPLICATION_JSON).build();

        } catch (JsonProcessingException e) {
            e.printStackTrace();
            return Response.serverError().entity("An unexpected error ocurred").build();
        }
    }

    @GET
    @Path("/{id}")
    @Produces({MediaType.APPLICATION_JSON})
    @RolesAllowed({"All","Developer","Mainteinance"})
    public Response getById(@PathParam("id") Long id) {
        ObjectMapper mapper = new ObjectMapper();
        try {
            if(id == null) {
                return Response.serverError().entity(mapper.writeValueAsString("The id cannot be null")).build();
            }
            Worker worker = workerController.getById(id);
            if(worker == null) {
                return Response.ok(mapper.writeValueAsString("No object found with this id: " + id),MediaType.APPLICATION_JSON).status(Status.NOT_FOUND).build();
            }else {
                String json = mapper.writeValueAsString(worker);
                return Response.ok(json, MediaType.APPLICATION_JSON).build();
            }
        } catch (JsonProcessingException e) {
            e.printStackTrace();
            return Response.serverError().entity("An unexpected error ocurred").build();
        }
    }
```

```java
    @DELETE
    @Path("/{id}/{friendId}")
    @Produces({MediaType.APPLICATION_JSON})
    @Transactional
    @RolesAllowed({"All","Developer","User"})
    public Response unfollowAccount(@PathParam("id") Long id, @PathParam("friendId") Long friendId) {
        ObjectMapper mapper = new ObjectMapper();
        try {
            PlayerAccount playeracc = playeraccController.getById(id);
            PlayerAccount playeracc2 = playeraccController.getById(friendId);
            if(playeracc == null) {
                return Response.ok(mapper.writeValueAsString("No player found with this ID: " + id),MediaType.APPLICATION_JSON).status(Status.NOT_FOUND).build();
            }else if(playeracc2 == null) {
                return Response.ok(mapper.writeValueAsString("No player found with this ID: " + friendId),MediaType.APPLICATION_JSON).status(Status.NOT_FOUND).build();
            }else {
                Boolean unfollowed = playeraccController.unfollowAccount(playeracc,playeracc2);
                if(unfollowed==false) {
                    return Response.status(Status.BAD_REQUEST).entity(mapper.writeValueAsString("Player with ID " + id + " is not following player with ID " + friendId)).build();
                }else {
                    return Response.ok().entity(mapper.writeValueAsString("Player with ID " + id + " unfollowed player with ID " + friendId)).build();
                }
            }
        } catch (JsonProcessingException e) {
            e.printStackTrace();
            return Response.serverError().entity("An unexpected error ocurred").build();
        }
    }

    @PUT
    @Path("/{id}/character")
    @Produces({ MediaType.APPLICATION_JSON})
    @Transactional
    @RolesAllowed({"All","Developer","User"})
    public Response createCharacter(@PathParam("id") Long id) {
        ObjectMapper mapper = new ObjectMapper();
        try {
            PlayerAccount playeracc = playeraccController.getById(id);
            if(playeracc == null) {
                return Response.ok(mapper.writeValueAsString("No player found with this ID: " + id),MediaType.APPLICATION_JSON).status(Status.NOT_FOUND).build();
            }else {
                Character persistedCharacter = playeraccController.createCharacter(playeracc);
                return Response.ok().entity(mapper.writeValueAsString("Player with ID " + id + " has created Character with ID " + persistedCharacter.getId() + " correctly")).build();
            }
            } catch (IOException e) {
               e.printStackTrace();
               return Response.serverError().entity("An unexpected error ocurred").build();
            }
    }
```

Jackson Framework has been used to parse a Json as an instance of entities or to parse instances/text as Json.

Testing

To perform the unit tests, some test suites containing test cases have been defined with the use of JUnit annotations like @Test. Other useful annotations worth noticing are @BeforeClass which executes a method before all the test cases in the test suite, @Before which is similar to BeforeClass but instead of executing the method before all the tests only once, it executes the method before each test case, and the equivalents @AfterClass and @After. These methods are the setup and teardown methods performed before and after the tests.

Model Test

The objective of these tests is to check that initialization and identity/equality comparisons between Objects work correctly.

To do this, first we define a simple class called “FakeBaseEntity” that extends BaseEntity. This is useful to avoid the problem of BaseEntity being an abstract class.

```java
package it.unifi.swam.assignment_restful_architecture.model_test;

import it.unifi.swam.assignment_restful_architecture.Model.BaseEntity;

//solves the problem of having BaseEntity abstract
public class FakeBaseEntity extends BaseEntity {
    public FakeBaseEntity (String uuid) {
        super(uuid);
    }
}
```

Now we can define the test suite BaseEntityTest.

```java
//We test correct initialization as well as identity and equality between Objects
public class BaseEntityTest {
    private FakeBaseEntity entity1;
    private FakeBaseEntity entity2;
    private FakeBaseEntity entity3;


    //Before = BeforeEach of junit5 (in junit4)
    @Before
    public void setup() {
        System.out.println("Perform the setup...");
        String uuid1 = UUID.randomUUID().toString();
        String uuid2 = UUID.randomUUID().toString();

        entity1 = new FakeBaseEntity(uuid1);
        entity2 = new FakeBaseEntity(uuid2);
        entity3 = new FakeBaseEntity(uuid1);
    }

    //Expected exception when trying to create an object without uuid
    //In juni5 exists Assertions.assertThrows(...)
    @Test(expected = IllegalArgumentException.class)
    public void testNullUUID() {
        System.out.println("Perform testNullUUID");
        new FakeBaseEntity(null);
    }

    @Test
    public void testEquals() {
        System.out.println("Perform testEquals");
        assertEquals(entity1, entity1); //Check Identity
        assertEquals(entity1, entity3); //Check Equality
        assertNotEquals(entity1, entity2); //Check Not Equality
    }
}
```

DAO Test

These tests make use of a simple database in memory defined in the persistence.xml of test/resources/META-INF.

It consists of an abstract class called JPATest that contains the methods of setup and teardown. These methods create an EntityManager manually through an EntityManagerFactory, which will be used later for the tests to perform operations with this database. Furthermore, it contains the method init(), which will be defined specifically for each concrete DAOTest.

```java
//includes the four annotations of junit (@Before, @BeforeClass, @After, @AfterClass)
//initializes the EntityManager and factory
public abstract class JPATest {
    private static EntityManagerFactory entityFactory;
    protected EntityManager entityManager;

    //creates EntityManagerFactory
    //once for every Test Suite (costful operation)
    @BeforeClass
    public static void setupEM() {
        System.out.println("Creates EntityManagerFactory");
        //not real DB system, "in memory" to give the DAO something to work with
        entityFactory = Persistence.createEntityManagerFactory("test");
    }

    //initializes EntityManager and calls init() method
    //Performed before each single TestCase
    @Before
    public void setup() throws IllegalAccessException {
        System.out.println("Creates EntityManager");
        entityManager = entityFactory.createEntityManager();
        entityManager.getTransaction().begin(); //starts cleaning transaction

        //cleans DB keeping the tables
        String sql = "SET FOREIGN_KEY_CHECKS = 0";
        entityManager.createNativeQuery(sql).executeUpdate();

        sql = "TRUNCATE TABLE server";
        entityManager.createNativeQuery(sql).executeUpdate();

        sql = "TRUNCATE TABLE sector;";
        entityManager.createNativeQuery(sql).executeUpdate();

        sql = "TRUNCATE TABLE worker;";
        entityManager.createNativeQuery(sql).executeUpdate();

        sql = "SET FOREIGN_KEY_CHECKS = 1";
        entityManager.createNativeQuery(sql).executeUpdate();


        entityManager.getTransaction().commit(); //closes cleaning transaction
        entityManager.getTransaction().begin(); //starts transaction for custom init
        System.out.println("Calls method init");
        //this method is abstract and it is specialized in each concrete class
        init();
        entityManager.getTransaction().commit();
        entityManager.clear();
        entityManager.getTransaction().begin();
        System.out.println("Setup completed");
    }

    //Closes the transaction of the EntityManager
    //Performed after each single TestCase
    @After
    public void close() {
        if(entityManager.getTransaction().isActive()) {
            entityManager.getTransaction().rollback();
            //If there is active transaction, perform a rollback
        }
        System.out.println("Closes EntityManager");
        entityManager.close();
    }

    //Closes the EntityManagerFactory
    //Performed after every TestSuite
    @AfterClass
    public static void tearDownDB() {
        System.out.println("Closes EntityManagerFactory");
        entityFactory.close();
    }

    //Abstract method that will be defined for each ConcreteClass extending JPATest
    protected abstract void init()
        throws IllegalAccessException;
```

Once we have defined this abstract class, we can make each concrete class to test the different DAOs. For example:

```java
public class WorkerDAOTest extends JPATest {
    private Worker worker;
    private WorkerDAO workerDao;

    //concrete init() method
    @Override
    protected void init() throws IllegalAccessException {
        System.out.println("Start init custom for WorkerDAOTest");
        worker = ModelFactory.worker();
        worker.setName("testWorker1");
        worker.setEmail("testWorker1@gmail.com");
        worker.setPassword("testWorker1",false);
        entityManager.persist(worker); //Persisted manually, without using the DAO
        //this is done to test the retrieve afterwards
        workerDao = new WorkerDAO();
        FieldUtils.writeField(workerDao,"entityManager",entityManager,true);
    }

    //retieve test
    //check that the entity retrieved is the same as the one defined in the init()
    @Test
    public void testFindById() {
        System.out.println("Perform testFindById in WorkerDAOTest");
        Worker result = workerDao.findById(worker.getId()); //It has an ID since it has been persisted in the init()
        assertEquals(worker.getId(), result.getId());
        assertEquals(worker.getName(), result.getName());
    }


    //In this case we check persisting the data through the DAO
    @Test
    public void testSave() {
        System.out.println("Perform testSave in WorkerDAOTest");
        Worker workerToPersist = ModelFactory.worker();
        workerToPersist.setName("testWorker2");
        workerToPersist.setEmail("testWorker2@gmail.com");
        workerToPersist.setPassword("testWorker2",false);
        workerDao.save(workerToPersist);
        Worker manuallyRetrievedWorker =  entityManager.
                createQuery("FROM Worker WHERE uuid = :uuid", Worker.class)
                .setParameter("uuid", workerToPersist.getUuid())
                .getSingleResult();
        assertEquals(workerToPersist, manuallyRetrievedWorker);
    }

    @Test
    public void testDelete() {
        System.out.println("Perform testDelete in WorkerDAOTest");
        Worker workerToDelete = ModelFactory.worker();
        workerToDelete.setName("testWorker3");
        workerToDelete.setEmail("testWorker3@gmail.com");
        workerToDelete.setPassword("testWorker3",false);
        entityManager.persist(workerToDelete);
        workerDao.delete(workerToDelete);
        List<Worker> manuallyRetrievedWorker =  entityManager.
                createQuery("FROM Worker WHERE uuid = :uuid", Worker.class)
                .setParameter("uuid", workerToDelete.getUuid())
                .getResultList();
        assertTrue(manuallyRetrievedWorker.isEmpty());
    }
```

```java
    @Test
    public void testFindWorkerByName() {
        System.out.println("Perform testFindWorkerByName in WorkerDAOTest");
        Worker worker1 = ModelFactory.worker();
        worker1.setName("Victor");
        worker1.setEmail("testemail@gmail.com");
        worker1.setPassword("testworker1",false);
        Worker worker2 = ModelFactory.worker();
        worker2.setName("Victor");
        worker2.setEmail("testemail2@gmail.com");
        worker2.setPassword("testworker2",false);
        entityManager.persist(worker1);
        entityManager.persist(worker2);
        List<Worker> result = workerDao.findWorkerByName("Victor");
        assertEquals(result.size(), 2);
    }

    @Test
    public void testGetWorkerByEmail() {
        System.out.println("Perform testGetWorkerByEmail in WorkerDAOTest");
        Worker workerToPersist = ModelFactory.worker();
        workerToPersist.setEmail("testgetbyemail@gmail.com");
        workerToPersist.setName("testgetbyemail1");
        workerToPersist.setPassword("testgetbyemail1",false);

        Worker workerToPersist2 = ModelFactory.worker();
        workerToPersist2.setEmail("testgetbyemail2@gmail.com");
        workerToPersist2.setName("testgetbyemail2");
        workerToPersist2.setPassword("testgetbyemail2",false);

        entityManager.persist(workerToPersist); //cascade is applied so the Worker is also persisted
        entityManager.persist(workerToPersist2);
        Worker retrievedWorker = workerDao.getWorkerByEmail("testgetbyemail@gmail.com");
        assertEquals(retrievedWorker.getEmail(), "testgetbyemail@gmail.com");
    }

    @Test
    public void testGetByRol() {
        System.out.println("Perform testGetByRol in WorkerDAOTest");
        Worker worker1 = ModelFactory.worker();
        worker1.setName("Victor");
        worker1.setEmail("testemail@gmail.com");
        worker1.setPassword("testworker1",false);
        worker1.setWorkerRol(WorkerRol.Developer);
        Worker worker2 = ModelFactory.worker();
        worker2.setName("Victor");
        worker2.setEmail("testemail2@gmail.com");
        worker2.setPassword("testworker2",false);
        worker2.setWorkerRol(WorkerRol.Mainteinance);

        entityManager.persist(worker1);
        entityManager.persist(worker2);
        List<Worker> result = workerDao.getWorkersByRol(WorkerRol.Developer);
        assertEquals(result.size(), 1);
    }
```

These classes test the methods defined in each of the different DAOs, using the assert methods that JUnit provides. Test Cases are independent between each other (they don't depend on the result of a previous Test Case).

Controller Test

These classes test the different Controllers. To do this, apart from JUnit methods and annotations, it is also used the Mockito Framework to "mock" the behavior of the DAOs inside the Controllers.

For Example:

```java
public class PlayerAccountControllerTest {
    private PlayerAccountController playerController;
    private PlayerAccountDAO playerDao;
    private PlayerAccount fakePlayer;

    @Before
    public void setup() throws IllegalAccessException {
        playerController = new PlayerAccountController();

        playerDao = mock(PlayerAccountDAO.class); //Mock instance (defines behaviour manually)

        fakePlayer =  ModelFactory.playeracc();
        fakePlayer.setUsername("Fake Player Username");
        fakePlayer.setEmail("fakeplayeremail@gmail.com");
        fakePlayer.setPassword("Fake Player Password", false);

        FieldUtils.writeField(playerController, "playeraccDao", playerDao, true); //Mock instance injected in the controller manually
    }

    @Test
    public void testGetPlayer() {
        when(playerDao.findById(1L)).thenReturn(fakePlayer);

        PlayerAccount retrievedPlayer = playerController.getById(1L);
        assertEquals(retrievedPlayer.getUsername(), fakePlayer.getUsername());
        assertEquals(retrievedPlayer.getEmail(), fakePlayer.getEmail());
        assertEquals(retrievedPlayer.getPassword(), fakePlayer.getPassword());
        assertEquals(retrievedPlayer, fakePlayer);
    }

    @Test
    public void testGetAllPlayers() {
        PlayerAccount fakePlayer2 =  ModelFactory.playeracc();
        fakePlayer2.setUsername("Fake Player Username2");
        fakePlayer2.setEmail("fakeplayeremail2@gmail.com");
        fakePlayer2.setPassword("Fake Player Password2", false);
        List<PlayerAccount> players = new ArrayList<PlayerAccount>();
        players.add(fakePlayer);
        players.add(fakePlayer2);
        when(playerDao.getAllPlayers()).thenReturn(players);

        List<PlayerAccount> retrievedPlayers = playerController.getAll();
        assertEquals(retrievedPlayers.size(), 2);
        assertEquals(retrievedPlayers, players);
        assertEquals(retrievedPlayers.get(0), fakePlayer);
        assertEquals(retrievedPlayers.get(1), fakePlayer2);
    }

    @Test
    public void testUpdatePlayer() {
        PlayerAccount newPlayer = ModelFactory.playeracc();
        newPlayer.setUsername("Updated Username Player");

        playerController.updatePlayerAccount(fakePlayer, newPlayer,true);
        assertEquals(fakePlayer.getUsername(), newPlayer.getUsername());
        assertEquals(fakePlayer.getUsername(), "Updated Username Player");
    }
```

```java
    @Test
    public void testFollowFriend() {
        PlayerAccount fakePlayer2 =  ModelFactory.playeracc();
        fakePlayer2.setUsername("Fake Player Username2");
        fakePlayer2.setEmail("fakeplayeremail2@gmail.com");
        fakePlayer2.setPassword("Fake Player Password2", false);

        playerController.followAccount(fakePlayer, fakePlayer2);
        assertEquals(fakePlayer.getFriends().size(), 1);
        assertEquals(fakePlayer.getFriends().get(0), fakePlayer2);
    }

    @Test
    public void testUnfollowFriend() {
        PlayerAccount fakePlayer2 =  ModelFactory.playeracc();
        fakePlayer2.setUsername("Fake Player Username2");
        fakePlayer2.setEmail("fakeplayeremail2@gmail.com");
        fakePlayer2.setPassword("Fake Player Password2", false);

        playerController.followAccount(fakePlayer, fakePlayer2);
        assertEquals(fakePlayer.getFriends().size(), 1);
        assertEquals(fakePlayer.getFriends().get(0), fakePlayer2);

        playerController.unfollowAccount(fakePlayer, fakePlayer2);
        assertEquals(fakePlayer.getFriends().size(), 0);
    }
```

Endpoint Test

These classes test the Rest Endpoints. It uses JUnit methods as well as the REST Assured library.

Example:

```
17 public class ServerEndpointTest {
18     private final static String baseURL = "assignment-restful-architecture/rest/";
19     private static String token;
20     private static Worker worker;
21 
22     @BeforeClass
23     public static void setup() throws IllegalAccessException {
24         RestAssured.baseURI = "http://localhost/";
25         RestAssured.port = 8080;
26         Populate.truncate();
27         Populate.populate();
28 
29         worker = ModelFactory.worker();
30         worker.setEmail("testdeveloper@gmail.com");
31         worker.setPassword("testdeveloper");
32         Response response = given()
33                 .contentType("application/json")
34                 .body(worker)
35                 .when().post(baseURL + "workerendpoint/login");
36         token = response.getBody().asString(); //token of an existing worker in the database with role Developer
37     }
38 
39     @AfterClass
40     public static void teardown() {
41         Populate.truncate();
42     }
43 
44     @Test
45     public void getServerByIdTest() {
46         Response response  = given().header("Authorization", "Bearer "+ token).pathParam("id","1").get(baseURL + "serverendpoint/" + "{id}");
47         response.then().statusCode(200)
48         .body("name",org.hamcrest.Matchers.equalTo("test server"));
49     }
50 
51     @Test
52     public void addServerTest() {
53         Server server =  ModelFactory.server();
54         server.setName("test server 2");
55 
56         Response response = given()
57         .header("Authorization", "Bearer "+ token)
58         .contentType("application/json")
59         .body(server)
60         .when().post(baseURL + "serverendpoint/");
61 
62         response.then().statusCode(201);
63     }
```

```
    @Test
    public void updateServer() {
        Server server =  ModelFactory.server();
        server.setName("test server 3");

        Response response = given()
        .header("Authorization", "Bearer "+ token)
        .contentType("application/json")
        .body(server)
        .when().post(baseURL + "serverendpoint/");

        response.then().statusCode(201);

        Long id = Long.valueOf(response.body().asString().replace("Object created with ID: ", "").replace('"', ' ').trim());

        server =  ModelFactory.server();
        server.setName("test server 3 update");

        response = given()
        .header("Authorization", "Bearer "+ token)
        .contentType("application/json")
        .body(server)
        .when().put(baseURL + "serverendpoint/" + id);

        response.then().statusCode(204);

        response  = given().header("Authorization", "Bearer "+ token).pathParam("id",id).get(baseURL + "serverendpoint/" + "{id}");
        response.then().statusCode(200)
        .body("name",org.hamcrest.Matchers.equalTo("test server 3 update"));
    }

    @Test
    public void deleteServer() {
        Server server =  ModelFactory.server();
        server.setName("test server 4");

        Response response = given()
                .header("Authorization", "Bearer "+ token)
                .contentType("application/json")
                .body(server)
                .when().post(baseURL + "serverendpoint/");

        response.then().statusCode(201);

        Long id = Long.valueOf(response.body().asString().replace("Object created with ID: ", "").replace('"', ' ').trim());

        response = given()
                .header("Authorization", "Bearer "+ token)
                .when().delete(baseURL + "serverendpoint/" + id);

                response.then().statusCode(200);

                response  = given().header("Authorization", "Bearer "+ token).pathParam("id",id).get(baseURL + "serverendpoint/" + "{id}");
                response.then().statusCode(404);
    }
```

```
@Test
public void activateServer() {
    Server server =  ModelFactory.server();
    server.setName("test server 5");
    server.setActive(false);

    Response response = given()
            .header("Authorization", "Bearer "+ token)
            .contentType("application/json")
            .body(server)
            .when().post(baseURL + "serverendpoint/");

    response.then().statusCode(201);

    Long id = Long.valueOf(response.body().asString().replace("Object created with ID: ", "").replace('"', ' ').trim());

    response = given()
            .header("Authorization", "Bearer "+ token)
            .when().put(baseURL + "serverendpoint/" + id + "/deactivate");

    response.then().statusCode(403); //forbidden por Worker of type Developer
}
```

The setup method uses truncate() and populate() from the Populate class in Utils to introduce some initial data in the database. Then a login is performed with the worker defined in the populate method. This is done because Endpoints are protected, so we need to perform the login to obtain the JWT Token to be able to make use of the endpoint operations.

In the test cases REST Assured is used to perform the operations defined in the endpoints, passing in the header the token received after logging.

References

[1] Java Authentication with JSON Web Tokens (JWTs) - https://www.baeldung.com/java-json-web-tokens-jjwt

[2] Jason Web Tokens (JWT) - https://jwt.io/

[3] Java: Create a Secure Password Hash - https://howtodoinjava.com/java/java-security/how-to-generate-secure-password-hash-md5-sha-pbkdf2-bcrypt-examples/

[4] BCrypt Docs - https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/crypto/bcrypt/BCrypt.html