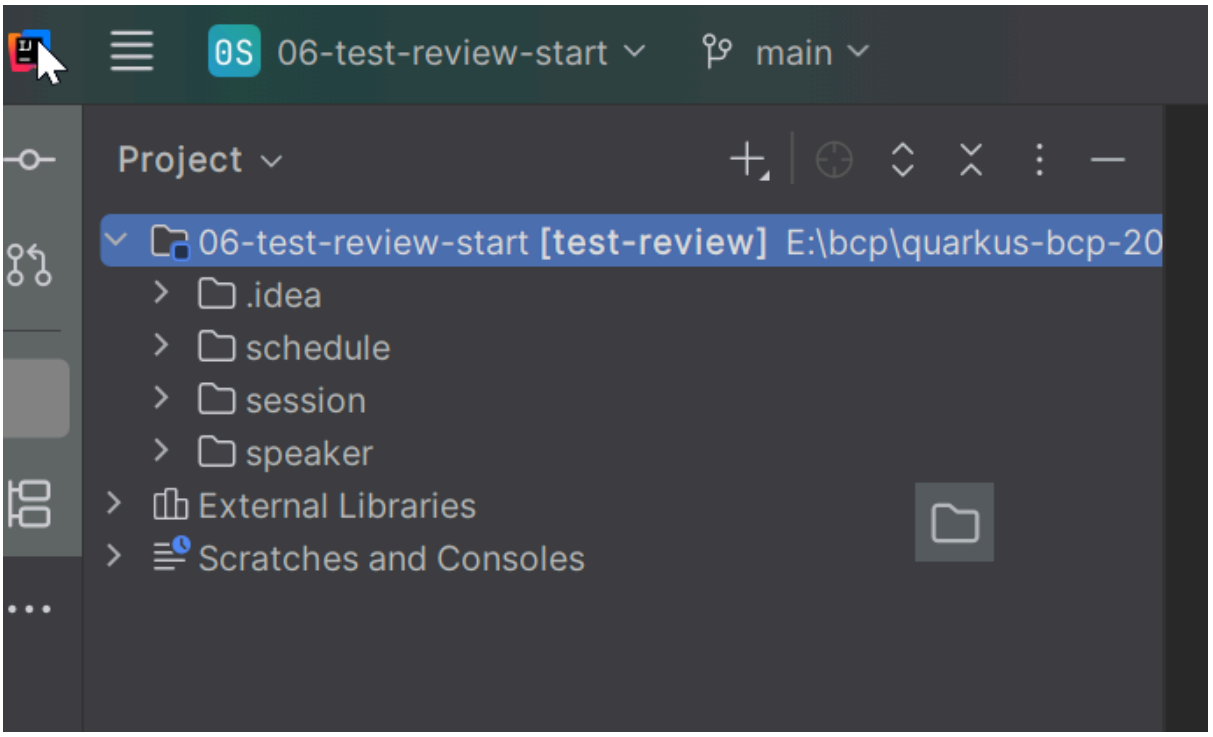
LAB 9: QUARKUS TEST REVIEW

Autor: José Díaz
Apoyo: Juan Ramirez

Github Repo: https://github.com/joedayz/quarkus-bcp-2025.git

1. Abre el proyecto **test-review**.



Instructions

Assume you are testing a microservices-based application that implements a conference management system. The application includes three microservices:

`schedule`
: Manages conference schedules by storing them in an H2 in-memory database.

  Tests initially fail because the HTTP test endpoint is not set and the H2 Dev Service is not working.

`speaker`
: Manages conference speakers by storing them in an H2 in-memory database. When the service starts, Quarkus populates the database with test data.

  Tests initially fail because of a missing dependency and a test scenario that requires the database to return an empty list of speakers.

`session`
: Manages conference sessions by storing them in a PostgreSQL database. This service depends on the `speaker` service to gather speaker information.

  Tests initially fail because Quarkus cannot find the PostgreSQL container image, and because the `speaker` service is not reachable.

You must make the tests pass in each of the three services.

1. Open the `schedule` project and fix the `ScheduleResourceTest` class. Convert the tests of this class into Quarkus tests and make the tests use the base URL of the `ScheduleResource` class.

2. The tests of the `schedule` service still fail due to a database connection error. Quarkus does not activate Dev Services for H2 because the application configuration contains H2 connection properties. Modify the configuration file, so that the H2 connection property does not apply to the `test` profile.

3. Change to the `speaker` service and inject the missing `DeterministicIdGenerator` dependency in the `SpeakerResourceTest` class. To inject this dependency, you must also update the `DeterministicIdGenerator` class to be a singleton mock bean.

4. Update the `SpeakerResourceTest#testListEmptySpeakers` test of the `speaker` service to prepare a scenario in which the `Speaker#listAll` returns an empty list. Because the database is initially populated, you must mock the Panache `Speaker` entity and the `Speaker#listAll` method to return an empty list.

5. Open the `session` microservice and fix the Dev Services configuration, use `registry.ocp4.example.com:8443/redhattraining/do378-postgres:14.1` as the PostgreSQL Dev Services image.

6. The `testGetSessionWithSpeaker` test of the `session` covers code that sends an HTTP request to the `speaker` service. The test fails because the other service is not reachable.

   In particular, the piece of code that makes the HTTP request is the `SpeakerService#getById` method. Fix the test by mocking this method. The mocked method must return a speaker that meets the test expectations.

Solución:

You must make the tests pass in each of the three services.

1. Open the `schedule` project and fix the `ScheduleResourceTest` class. Convert the tests of this class into Quarkus tests and make the tests use the base URL of the `ScheduleResource` class.

    1.1. Navigate to the `schedule` service directory.

```
[student@workstation ~]$ cd ~/D0378/test-review/schedule
```

1.2. Open the project with your editor of choice, such as VSCodium or vim.

1.3. Verify that four tests are failing.

```
[student@workstation schedule]$ mvn test
...output omitted...
[ERROR] Errors:
[ERROR]   ScheduleResourceTest.testAdd:41 » Connect Connection refused
[ERROR]   ScheduleResourceTest.testAllSchedules:50 » IllegalState This ...
[ERROR]   ScheduleResourceTest.testRetrieve:28 » Connect Connection refused
[ERROR]   ScheduleResourceTest.testRetrieveByVenue:61 » IllegalState This ...
[INFO]
[ERROR] Tests run: 4, Failures: 0, Errors: 4, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
...output omitted...
```

1.4. Open the `src/test/java/com/redhat/training/conference/ScheduleResourceTest.java` file and add the `QuarkusTest` and `TestHTTPEndpoint` annotations. The `TestHTTPEndpoint` must use the endpoint of the `ScheduleResource` class.

```
...output omitted...

@QuarkusTest
@TestHTTPEndpoint(ScheduleResource.class)
public class ScheduleResourceTest {
...output omitted...
```

1.5. Verify that the tests still fail due to a different error.

```
[student@workstation schedule]$ mvn test
...output omitted...
ERROR [org.hib.eng.jdb.spi.SqlExceptionHelper] (main) Connection is broken:
 "`java.net.UnknownHostException`: schedules-db.conference.svc.cluster.local:
 schedules-db.conference.svc.cluster.local" [...]
...output omitted...
[ERROR] Errors:
[ERROR]   ScheduleResourceTest.testAllSchedules:51 » Persistence
 org.hibernate.exception...
[ERROR]   ScheduleResourceTest.testRetrieveByVenue:62 » Persistence
 org.hibernate.except...
[INFO]
[ERROR] Tests run: 4, Failures: 2, Errors: 2, Skipped: 0
...output omitted...
```

2. The tests of the `schedule` service still fail due to a database connection error. Quarkus does not activate Dev Services for H2 because the application configuration contains H2

connection properties. Modify the configuration file, so that the H2 connection property does not apply to the `test` profile.

2.1. Open the `src/main/resources/application.properties` file. The H2 database is configured to use a specific connection URL, which causes the deactivation of H2 Dev Services. Use the `%prod.` prefix to indicate that the H2 connection URL is only for production.

```
%prod.quarkus.datasource.jdbc.url = jdbc:h2:tcp://schedules-
db.conference.svc.cluster.local/~/schedules
```

2.2. Verify that all the tests pass.

```
[student@workstation schedule]$ mvn test
...output omitted...
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
...output omitted...
```

3. Change to the `speaker` service and inject the missing `DeterministicIdGenerator` dependency in the `SpeakerResourceTest` class. To inject this dependency, you must also update the `DeterministicIdGenerator` class to be a singleton mock bean.

3.1. Navigate to the `speaker` service directory.

```
[student@workstation schedule]$ cd ~/D0378/test-review/speaker
```

3.2. Open the project with your editor of choice.

3.3. Verify that the tests fail due to a compilation error.

```
[student@workstation speaker]$ mvn test
...output omitted...
[ERROR] COMPILATION ERROR :
[INFO] -------------------------------------------------------------
[ERROR] /.../SpeakerResourceTest.java:[25,9] cannot find symbol
  symbol:   variable idGenerator
  location: class com.redhat.training.speaker.SpeakerResourceTest
[INFO] 1 error
[INFO] -------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
...output omitted...
```

3.4. Open the `src/test/java/com/redhat/training/speaker/SpeakerResourceTest.java` file and inject the `DeterministicIdGenerator` bean in the `idGenerator` field.

```
...output omitted...

@QuarkusTest
public class SpeakerResourceTest {

  @Inject
  DeterministicIdGenerator idGenerator;

  ...output omitted...
}
```

3.5. Open the `src/test/java/com/redhat/training/speaker/DeterministicIdGenerator.java` file and identify the class as a mock singleton bean.

```
...output omitted...

@Mock
@Singleton
public class DeterministicIdGenerator implements IdGenerator {
    ...output omitted...
}
```

3.6. Verify that the tests pass partially. One test passes, but the `testListEmptySpeakers` test method is still failing because the database is not empty.

```
[student@workstation speaker]$ mvn test
[ERROR] Failures:
[ERROR]   SpeakerResourceTest.testListEmptySpeakers:49 1 expectation failed.
JSON path size() doesn't match.
Expected: is 0
  Actual: 5

[INFO]
[ERROR] Tests run: 2, Failures: 1, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
```

4. Update the `SpeakerResourceTest#testListEmptySpeakers` test of the `speaker` service to prepare a scenario in which the `Speaker#listAll` returns an empty list. Because the database is initially populated, you must mock the Panache `Speaker` entity and the `Speaker#listAll` method to return an empty list.

4.1. Mock the Panache entity to simulate a scenario where the speakers database is empty.

```
@Test
public void testListEmptySpeakers() {
    PanacheMock.mock( Speaker.class );
    Mockito.when( Speaker.listAll() )
            .thenReturn( Collections.emptyList() );

    given()
    .when()
        .get( "/speaker" )
    .then()
        .statusCode( 200 )
        .body( "size()", is( 0 ) );
}
```

4.2. Verify that the two tests pass.

```
[student@workstation speaker]$ mvn test
...output omitted...
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
...output omitted...
```

5. Open the `session` microservice and fix the Dev Services configuration, use `registry.ocp4.example.com:8443/redhattraining/do378-postgres:14.1` as the PostgreSQL Dev Services image.

5.1. Navigate to the `session` service directory.

```
[student@workstation speaker]$ cd ~/DO378/test-review/session
```

5.2. Open the project with your editor of choice.

5.3. Verify that the tests fail due to a container image error.

```
[student@workstation session]$ mvn test
...output omitted...
Caused by: org.testcontainers.containers.ContainerLaunchException: Container
 startup failed
Caused by: org.testcontainers.containers.ContainerFetchException: Can't get Docker
 image: ...
...output omitted...
[ERROR] Errors:
[ERROR]   SessionResourceTest.testGetSessionWithSpeaker » Runtime
 java.lang.RuntimeExcep...
[INFO]
[ERROR] Tests run: 2, Failures: 0, Errors: 2, Skipped: 1
[INFO]
```

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
```

5.4. Open the `src/main/resources/application.properties` file and fix the `quarkus.datasource.devservices.image-name` property.

```
quarkus.datasource.devservices.image-name
 = registry.ocp4.example.com:8443/redhattraining/do378-postgres:14.1
```

5.5. Run the tests. Verify that the `testGetSessionWithSpeaker` method fails.

```
[student@workstation session]$ mvn test
...output omitted...
[ERROR] Failures:
[ERROR]   SessionResourceTest.testGetSessionWithSpeaker:53 1 expectation failed.
JSON path speaker.firstName doesn't match.
Expected: Pablo
  Actual: null

[INFO]
[ERROR] Tests run: 2, Failures: 1, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
...output omitted...
```

6. The `testGetSessionWithSpeaker` test of the `session` covers code that sends an HTTP request to the `speaker` service. The test fails because the other service is not reachable.

In particular, the piece of code that makes the HTTP request is the `SpeakerService#getById` method. Fix the test by mocking this method. The mocked method must return a speaker that meets the test expectations.

6.1. Open the `src/test/java/com/redhat/training/conference/session/SessionResourceTest.java` file. In the `testGetSessionWithSpeaker()` method, mock the speaker service to return a speaker.

```
@Test
public void testGetSessionWithSpeaker () {

    int speakerId = 12;

    Mockito.when(
        speakerService.getById(Mockito.anyInt())
    ).thenReturn(
        new Speaker(speakerId, "Pablo", "Solar")
    );

    ...output omitted...
}
```

6.2. Run the tests and verify that they pass.

```
[student@workstation session]$ mvn test
...output omitted...
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
...output omitted...
```

Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation test-review]$ lab grade test-review
```

Finish

Run the **lab finish** command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish test-review
```

This concludes the section.

enjoy!

Jose