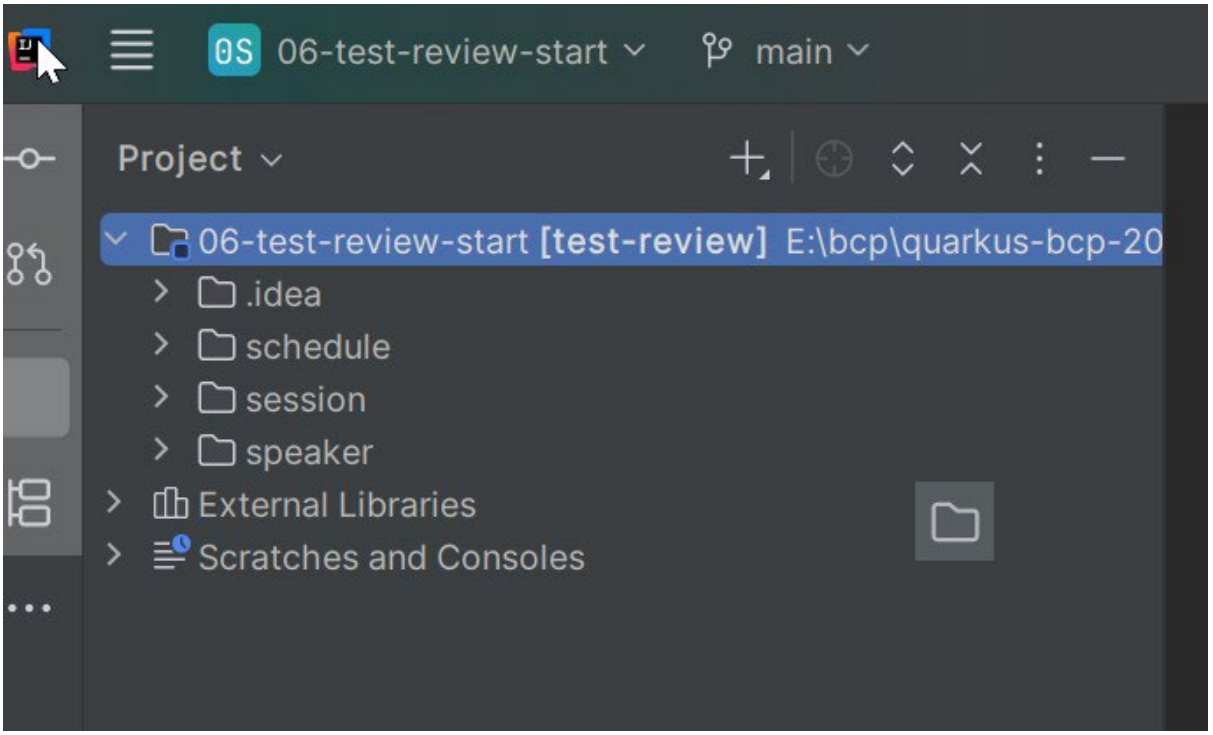
LAB 9: QUARKUS TEST REVIEW

Autor: José Díaz

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

1. Abre el proyecto **test-review**. Encontrarás 3 proyectos: schedule, session y speaker.



Supón que estás probando una aplicación basada en microservicios que implementa un sistema de gestión de conferencias. La aplicación incluye tres microservicios:

schedule
Administra los horarios de la conferencia almacenándolos en una base de datos en memoria H2.

Las pruebas fallan inicialmente porque no se ha configurado el endpoint de prueba HTTP y el Dev Service de H2 no está funcionando.

speaker
Administra los conferencistas almacenándolos en una base de datos en memoria H2. Cuando el servicio se inicia, Quarkus llena la base de datos con datos de prueba.
Las pruebas fallan inicialmente debido a una dependencia faltante y a un escenario de prueba que requiere que la base de datos devuelva una lista vacía de conferencistas.

session
Administra las sesiones de la conferencia almacenándolas en una base de datos PostgreSQL. Este servicio depende del servicio *speaker* para obtener información de los conferencistas.
Las pruebas fallan inicialmente porque Quarkus no puede encontrar la imagen del contenedor de PostgreSQL y porque el servicio *speaker* no es accesible.

Debes lograr que las pruebas pasen en cada uno de los tres servicios.

1. Abre el proyecto **schedule** y corrige la clase ScheduleResourceTest.
   Convierte las pruebas de esta clase en pruebas de Quarkus y haz que las pruebas usen la URL base de la clase ScheduleResource.

2. Las pruebas del servicio **schedule** aún fallan debido a un error de conexión con la base de datos.
   Quarkus no activa Dev Services para H2 porque la configuración de la aplicación contiene propiedades de conexión de H2.
   Modifica el archivo de configuración de modo que la propiedad de conexión H2 no se aplique al perfil de pruebas.

3. Cambia al servicio **speaker** e inyecta la dependencia faltante DeterministicIdGenerator en la clase SpeakerResourceTest.
   Para inyectar esta dependencia, también debes actualizar la clase DeterministicIdGenerator para que sea un *singleton mock bean*.

4. Actualiza la prueba SpeakerResourceTest#testListEmptySpeakers del servicio **speaker** para preparar un escenario en el que Speaker#listAll devuelva una lista vacía.
   Debido a que la base de datos se llena inicialmente, debes *mockear* la entidad Panache Speaker y el método Speaker#listAll para que retornen una lista vacía.

5. Abre el microservicio **session** y corrige la configuración de Dev Services.
   Usa registry.ocp4.example.com:8443/redhat-training/do378-postgres:14.1 como la imagen de PostgreSQL para Dev Services.

6. La prueba testGetSessionWithSpeaker del servicio **session** cubre código que envía una solicitud HTTP al servicio **speaker**. La prueba falla porque el otro servicio no es accesible.
   En particular, la parte del código que hace la solicitud HTTP es el método SpeakerService#getById.
   Corrige la prueba *mockeando* este método. El método *mockeado* debe

devolver un conferencista que cumpla con las expectativas de la prueba.

Solución:

Debes hacer que las pruebas pasen en cada uno de los tres servicios.

1.0 Abre el proyecto schedule y corrige la clase ScheduleResourceTest. Convierte las pruebas de esta clase en pruebas de Quarkus y haz que las pruebas usen la URL base de la clase ScheduleResource.

1.1. Navega al directorio del servicio **schedule**.

Línea de Comando: [student@workstation ~]$ cd ~/06-test-review-start/schedule

1.2: Abre el proyecto con tu editor de elección, como VSCode o Intellij.

1.3: Verifica que cuatro pruebas estén fallando.

Salida de Línea de Comandos: [student@workstation schedule]$ mvn test

```
[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: Abre el archivo src/test/java/com/redhat/training/conference/ScheduleResourceTest.java y agrega las anotaciones QuarkusTest y TestHTTPEndpoint. TestHTTPEndpoint debe usar el endpoint de la clase ScheduleResource.

Fragmento de Código:
...salida omitida...

@QuarkusTest
@TestHTTPEndpoint(ScheduleResource.class)
public class ScheduleResourceTest {
salida omitida...

1.5: Verifica que las pruebas sigan fallando debido a un error diferente.

Salida de Línea de Comandos:
[student@workstation schedule]$ **mvn test**
...salida omitida...

```
[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...
```

2.0 Las pruebas del servicio schedule siguen fallando debido a un error de conexión de base de datos. Quarkus no activa Dev Services para H2 porque la configuración de la aplicación contiene H2. Modifica el archivo de configuración, de manera que las propiedades de conexión H2 no apliquen al profile test.

2.1 Abre el archivo src/main/resources/application.properties. La base de datos H2 está configurada para usar una URL de conexión específica, lo que causa la desactivación de H2 Dev Services. Usa el prefijo %prod. para indicar que la URL de conexión H2 es solo para producción."

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

2.2 Verifica que todas las pruebas pasen.

Salida de Línea de Comandos: [student@workstation schedule]$ mvn test
...salida omitida...
[INFO] Pruebas ejecutadas: 4, Fallos: 0, Errores: 0, Omitidas: 0
[INFO]
[INFO]
[INFO] CONSTRUCCIÓN EXITOSA
[INFO]
...salida omitida...

3.0 Cambia al servicio speaker e inyecta la dependencia

faltante DeterministicIdGenerator en la clase SpeakerResourceTest. Para inyectar esta dependencia, también debes actualizar la clase DeterministicIdGenerator para que sea un bean mock singleton.

3.1 Navega al directorio del servicio speaker.

Salida de Línea de Comandos:
[student@workstation schedule]$ cd ~/D0378/06-test-review-start/speaker

3.2 Abre el proyecto con tu editor de elección.

3.3 Verifica que las pruebas fallen debido a un error de compilación.

Salida de Línea de Comandos:
[student@workstation speaker]$ mvn test

```
[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 Abre el archivo src/test/java/com/redhat/training/speaker/SpeakerResourceTest.java e inyecta el bean DeterministicIdGenerator en el campo idGenerator.

```
@QuarkusTest
public class SpeakerResourceTest {

  @Inject
  DeterministicIdGenerator idGenerator;

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

3.5 Abre el archivo src/test/java/com/redhat/training/speaker/DeterministicIdGenerator.java e identifica la clase como un bean mock singleton.

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

3.6 Verifica que las pruebas pasen parcialmente. Una prueba pasa, pero el método de prueba testListEmptySpeakers sigue fallando porque la base de datos no está vacía.

```
[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. Actualiza la prueba SpeakerResourceTest#testListEmptySpeakers del servicio speaker para preparar un escenario en el que Speaker#listAll devuelva una lista vacía. Debido a que la base de datos está inicialmente poblada, debes simular la entidad Panache Speaker y el método Speaker#listAll para devolver una lista vacía.

4.1 Simula la entidad Panache para simular un escenario donde la base de datos de speakers está vacía.

```
@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 Verifica que los 2 tests pasen.

```
[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. Abre el microservicio session y repara la configuración de Dev Service, usa postgres:14.1 como la imagen de PostgreSQL Dev Services.

5.1 Ve al directorio de servicio session

Salida de Línea de Comandos:
[student@workstation schedule]$ cd ~/D0378/06-test-review-start/session

5.2 Abre el proyecto con tu editor preferido.

5.3 Verifica que las pruebas fallen debido a un error en la imagen del contenedor.

```
[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]
```

5.4 Abre el src/main/resources/application.properties y modifica la propiedad quarkus.datasource.devservices.image-name.

```
quarkus.datasource.devservices.image-name=postgres:14.1
```

5.5 Ejecuta las pruebas. Verifica que el método testGetSessionWithSpeaker falla.

```
[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. El test testGetSessionWithSpeaker de session cubre el Código que envía un request HTTP al servicio speaker. El test falla porque el otro servicio no está disponible.

   El código que hace el request HTTP es el método SpeakerSErvice#getById. Repara el test mockeando este método. El método mockeado debe retornar un speaker que cumple con las expectivas del test.

   6.1 Abre el src/test/java/com/bcp/training/conference/session/SessionResourceTest.java. En el método testGetSessionWithSpeaker(), mockea el servicio speaker para retornar un speaker.

```
@Test
public void testGetSessionWithSpeaker(){
    int speakerId = 12;

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

    … Código omitido …
}
```

6.2 Ejecuta las pruebas y verifica que los tests pasen.

```
[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...
```

Has finalizado el review.

Felicitaciones.

José