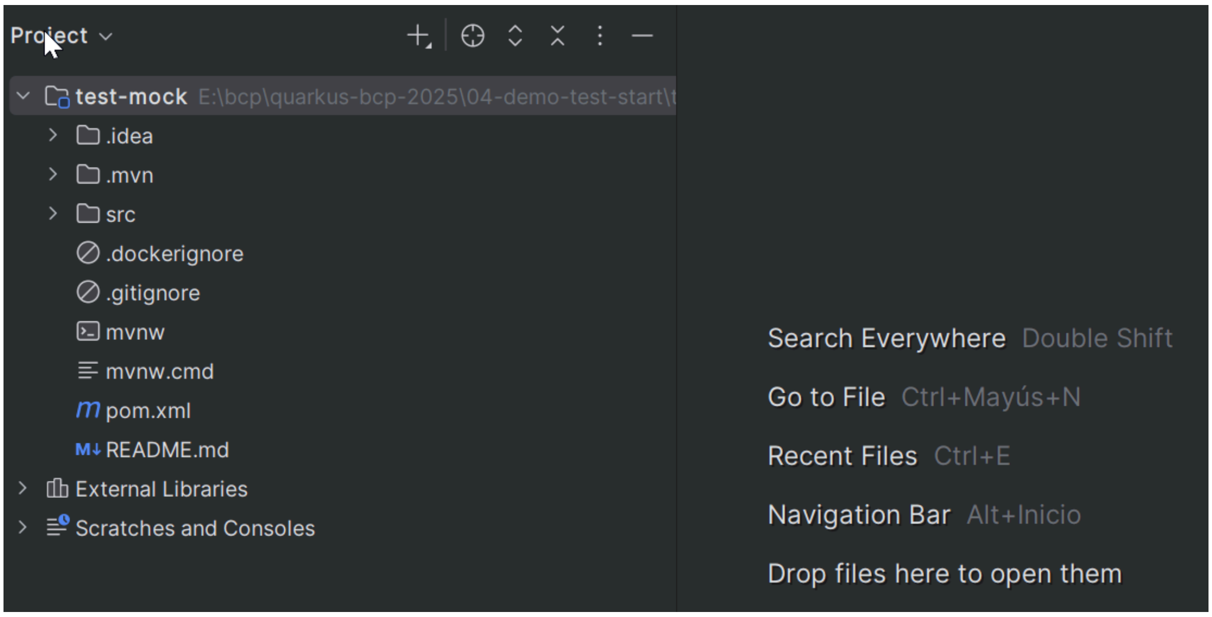
LAB 7: QUARKUS MOCK

Autor: José Díaz

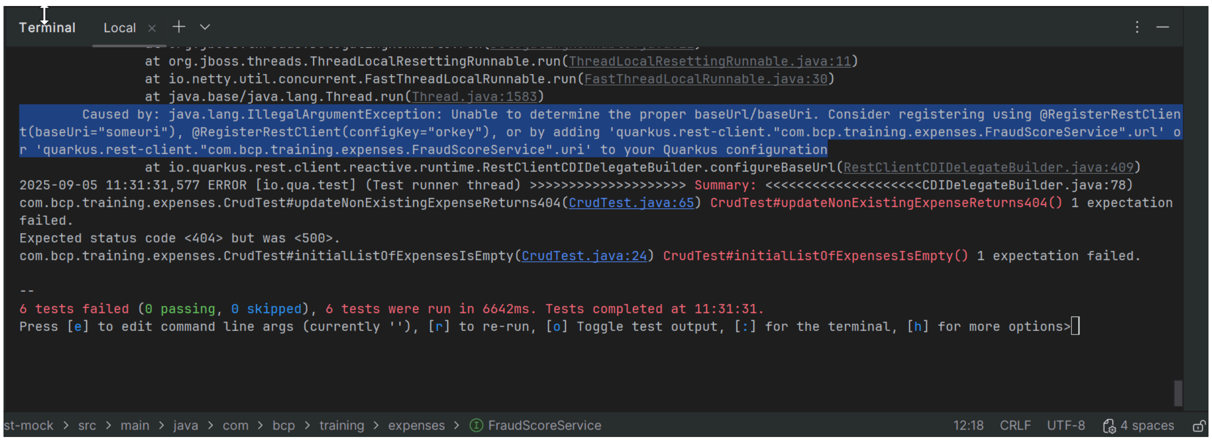
Github Repo: https://github.com/joedayz/quarkus-bcp-2025.git

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



2. Examinar las clases y ejecutar con: mvn quarkus dev

3. Presionar la tecla **“r”** para ver el error de FraudScoreService:


Terminal
Local
st-mock › src › main › java › com › bcp › training › expenses › FraudScoreService
12:18 CRLF UTF-8 4 spaces

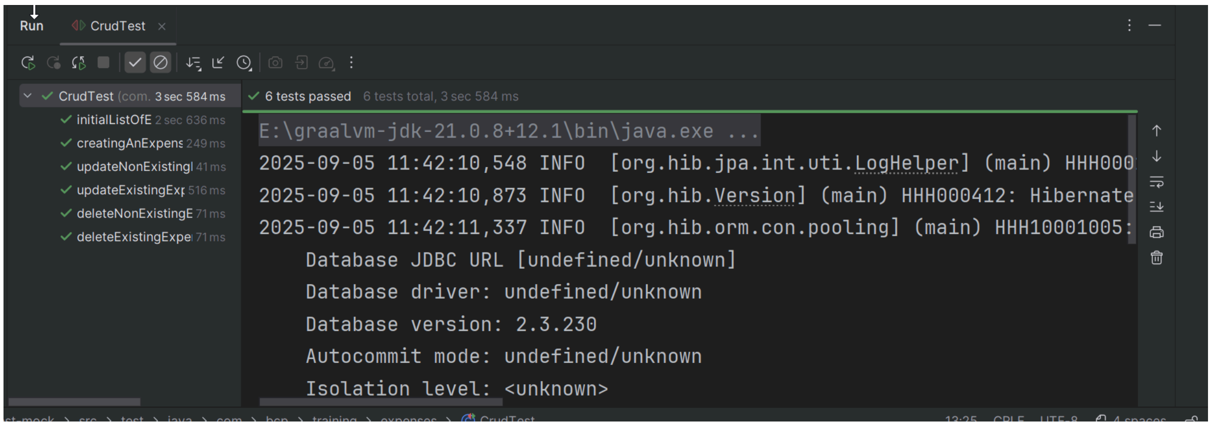
```
                at org.jboss.threads.ThreadLocalResettingRunnable.run(ThreadLocalResettingRunnable.java:11)
                at io.netty.util.concurrent.FastThreadLocalRunnable.run(FastThreadLocalRunnable.java:30)
                at java.base/java.lang.Thread.run(Thread.java:1583)
        Caused by: java.lang.IllegalArgumentException: Unable to determine the proper baseUrl/baseUri. Consider registering using @RegisterRestClien
t(baseUri="someuri"), @RegisterRestClient(configKey="orkey"), or by adding 'quarkus.rest-client."com.bcp.training.expenses.FraudScoreService".url' o
r 'quarkus.rest-client."com.bcp.training.expenses.FraudScoreService".uri' to your Quarkus configuration
                at io.quarkus.rest.client.reactive.runtime.RestClientCDIDelegateBuilder.configureBaseUrl(RestClientCDIDelegateBuilder.java:409)
2025-09-05 11:31:31,577 ERROR [io.qua.test] (Test runner thread) >>>>>>>>>>>>>>>>>>>> Summary: <<<<<<<<<<<<<<<<<<<<CDIDelegateBuilder.java:78)
com.bcp.training.expenses.CrudTest#updateNonExistingExpenseReturns404(CrudTest.java:65) CrudTest#updateNonExistingExpenseReturns404() 1 expectation
failed.
Expected status code <404> but was <500>.
com.bcp.training.expenses.CrudTest#initialListOfExpensesIsEmpty(CrudTest.java:24) CrudTest#initialListOfExpensesIsEmpty() 1 expectation failed.

--
6 tests failed (0 passing, 0 skipped), 6 tests were run in 6642ms. Tests completed at 11:31:31.
Press [e] to edit command line args (currently ''), [r] to re-run, [o] Toggle test output, [:] for the terminal, [h] for more options>
```

4. Para que el CrudTest.java funcione, agregar en tu properties:

```
quarkus.rest-client."com.bcp.training.expenses.FraudScoreService".url=http://localhost:9080
```

5. Ejecuta la prueba y deberías tener este resultado:



```
CrudTest (com. 3 sec 584 ms
  initialListOfE 2 sec 636 ms
  creatingAnExpens 249 ms
  updateNonExistingI 41 ms
  updateExistingExp 516 ms
  deleteNonExistingE 71 ms
  deleteExistingExpe 71 ms

6 tests passed   6 tests total, 3 sec 584 ms

E:\graalvm-jdk-21.0.8+12.1\bin\java.exe ...
2025-09-05 11:42:10,548 INFO  [org.hib.jpa.int.uti.LogHelper] (main) HHH000
2025-09-05 11:42:10,873 INFO  [org.hib.Version] (main) HHH000412: Hibernate
2025-09-05 11:42:11,337 INFO  [org.hib.orm.con.pooling] (main) HHH10001005:
	Database JDBC URL [undefined/unknown]
	Database driver: undefined/unknown
	Database version: 2.3.230
	Autocommit mode: undefined/unknown
	Isolation level: <unknown>
```

6. Ahora probemos ServiceMockTest.java

Implementar:

```java
@InjectMock
ExpenseService mockExpenseService;
```

```java
    @Test
    public void creatingAnExpenseReturns400OnInvalidAmountAndType() {
        Mockito.when(
                mockExpenseService.meetsMinimumAmount(Mockito.anyDouble())
        ).thenReturn(false);

        given()
                .body(CrudTest.generateExpenseJson(
                        "",
                        "Expense 1",
                        "CASH"
                        , 99999
                ))
                .contentType(ContentType.JSON)
                .when()
                .post("/expenses")
                .then().statusCode(400);
    }
```

7. Implementar el Mock de ExpenseService:

```java
@Mock
@ApplicationScoped
public class ExpenseServiceMock extends
ExpenseService {

   @Override
   public boolean exists(UUID uuid) {
       return
!uuid.equals(UUID.fromString(CrudTest.NON_EXISTI
NG_UUID));
   }
}
```

8. Ahora vamos a implementar PanacheMock

```java
@QuarkusTest

public class PanacheMockTest {
   @Test
   public void listOfExpensesReturnsAnEmptyList(){

Mockito.when(Expense.listAll()).thenReturn(Collect
ions.emptyList());


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

9. Hay que crear un PanacheMock

```
@BeforeAll
public static void setup(){
   PanacheMock.mock(Expense.class);
}
```

10. Volvemos a probar el test y debería funcionar.

11. Vamos a probar el `RestClientMockTest`
12. Inyectar el mock del RestClient

```
@QuarkusTest
public class RestClientMockTest {



   @InjectMock
   @RestClient
   FraudScoreService fraudScoreService;
}
```

13. Implementar el método

```
@Test
public void highFraudScoreReturns400(){
   Mockito.when(

fraudScoreService.getByAmount(Mockito.anyDouble())
   ).thenReturn(new FraudScore(500));
   given()
           .body(
                   CrudTest.generateExpenseJson(
                           "",
                           "Expense 1"
                           ,
                           "CASH"
                           ,50000
                   )
           ).contentType(ContentType.JSON)
           .when()
           .post("/expenses/score")
           .then().statusCode(400);


}
```

14. Implementar el SpyTest

```
@QuarkusTest
public class SpyTest {
```

```java
   @InjectSpy
   ExpenseService expenseService;
```

15. Implementar el método que valida la llamada (1 sola vez) al método list

```java
@Test
public void listOfExpensesCallsExpensesList() {
   given()
           .when()
           .get("/expenses")
           .then()
           .statusCode(200)
           .body("$.size()", is(0));
   Mockito.verify(expenseService,
Mockito.times(1)).list();
}
```

enjoy!

Jose