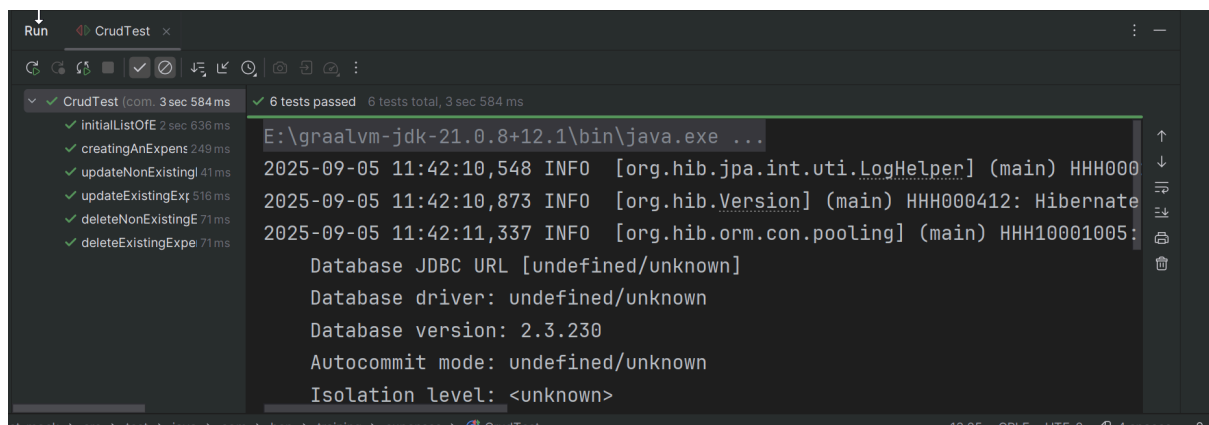


4. Para que el CrudTest.java funcione, agregar en tu properties:

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

5. Ejecuta la prueba y deberías tener este resultado:



```
Run CrudTest x
6 tests passed 6 tests total, 3 sec 584 ms
initialListOfE 2 sec 636 ms
creatingAnExpense 249 ms
updateNonExistingExpense 41 ms
updateExistingExpense 516 ms
deleteNonExistingExpense 71 ms
deleteExistingExpense 71 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:

```
@InjectMock
ExpenseService mockExpenseService;
```

```
@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:

```
@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

```
@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 {
```

```
@InjectSpy
```

```
ExpenseService expenseService;
```

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

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

enjoy!

Jose