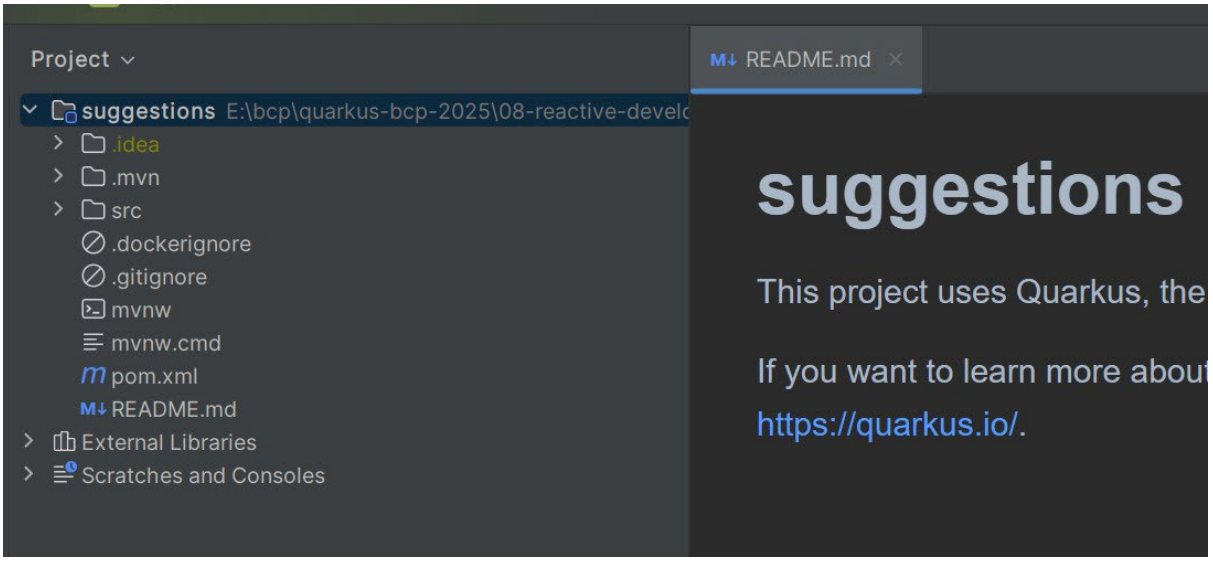
LAB 11: QUARKUS REACTIVE DEVELOP

Autor: José Díaz

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

1. Abre el proyecto **reactive-develop-start**.



Instrucciones

Vamos a crear un API reactiva que implementa un sistema de sugerencias de compras en comercio electrónico. Esta API guardará las sugerencias en una base de datos reactiva para incrementar el rendimiento y reducir la latencia.

La estructura del proyecto contiene los tests para verificar el comportamiento de la aplicación.

1. Configurar el proyecto para desarrollo reactivo

1.1 En una terminal de Windows, navega al directorio del proyecto.

```
cd 08-reactive-develop-start
```

1.2 Agrega al proyecto hibernate-reactive-panache y reactive-pg-client.

```
[student@workstation reactive-develop]$ mvn quarkus:add-extension \
 -Dextension="hibernate-reactive-panache,reactive-pg-client"
...output omitted...
[INFO] [SUCCESS] ... Extension io.quarkus:quarkus-hibernate-reactive-panache has
 been installed
[INFO] [SUCCESS] ... Extension io.quarkus:quarkus-reactive-pg-client has been
 installed
...output omitted...
```

mvn quarkus:add-extension -Dextension=”hibernate-reactive-panache,reactive-pg-client”

1.3 Abre el Proyecto con tu editor, por ejemplo Intellij.

2. Revisa el código a usar.

2.1 Abre la entidad Suggestion para verificar que es una entidad JPA regular y que tiene un constructor para inicializar los valores.

2.2 Abre la clase SuggestionResourceTest y revisa los tests existentes para los métodos del API a construir.

3. Inicia el modo de testing continuo.

3.1 Abre una terminal en tu IDE favorito.

3.2 Inicia Quarkus en modo testing continuo y verifica que los tests están fallando.

```
[student@workstation reactive-develop]$ mvn quarkus:test
...output omitted...
3 tests failed (0 passing, 0 skipped)...
```

4. Agrega el endpoint para crear un suggestion.

4.1 Agrega el endpoint par crear un suggestion en la clase SuggestionResource. Este método debe usar el método PanacheEntity.persist() wrappeado en una llamada con Panache.withTransaction.

```
@POST
public Uni<Suggestion> create( Suggestion newSuggestion ) {
    return Panache.withTransaction( newSuggestion::persist );
}
```

4.2 Verifica que después de salvar el archivo con el nuevo método, el test pasa.

```
...output omitted...
2 tests failed (1 passing, 0 skipped)
```

5. Agrega un endpoint para recuperar un suggestion por ID.

5.1 Agrega el método get en la clase SuggestionResource, usando el método PanacheEntity.findById(). Usa el parámetro para encontrar la entidad en la base de datos.

```
@GET
@Path( "/{id}" )
public Uni<Suggestion> get( Long id ) {
    return Suggestion.findById( id );
}
```

5.2 Guarda y verifica que solo un test esta fallando.

```
...output omitted...
1 tests failed (2 passing, 0 skipped)
```

6. Agrega un método para listar todas las suggestions.

6.1 Agrega un método list a la clase SuggestionResource usando el método GET HTTP en el root path. Usa la siguiente implementación para obtener todos los valores disponibles.

```
@GET
public Multi<Suggestion> list(){
   return Suggestion.<Suggestion>listAll()
         .onItem().transformToMulti(list ->
               Multi.createFrom().iterable(list));
}
```

6.2 Después de salvar el archivo, verifica que todos los tests pasan.

```
...output omitted...
All 3 tests are passing (0 skipped)
```

6.3 Detén el testing continuo presionando la letra **q**.

Felicitaciones, has terminado el laboratorio.

Jose