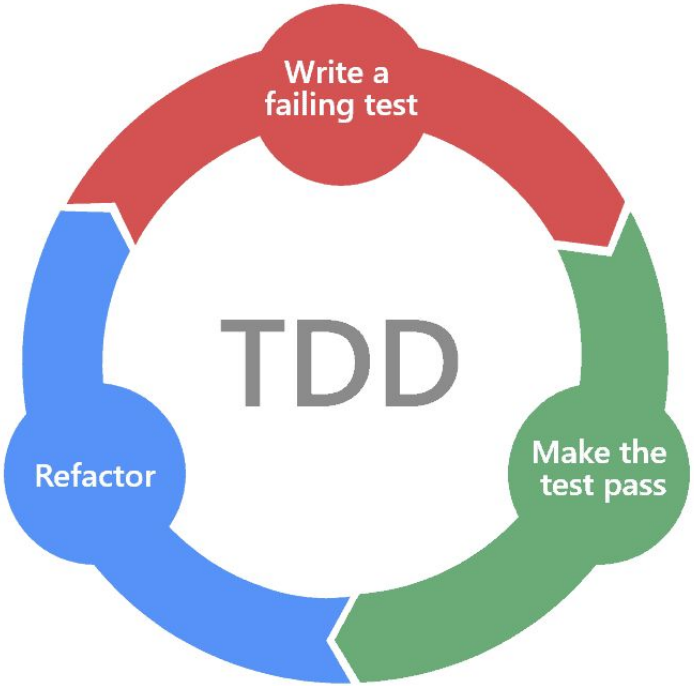
TDD aplicado al desarrollo de APIs

@MercadonaTech

¿Qué es TDD?



¿Por qué TDD?

1. Descubrimiento del sistema.
2. Mejor diseño de aplicación y mejor calidad del código.
3. Documentación detallada del proyecto.
4. Mantenimiento más sencillo y seguro.
5. Testeas comportamiento y te olvidas del resto.

Estructura de un test

Atención! El *naming* es súper importante.

El nombre de un test representa el comportamiento que está comprobando.

Arrange →

Act →

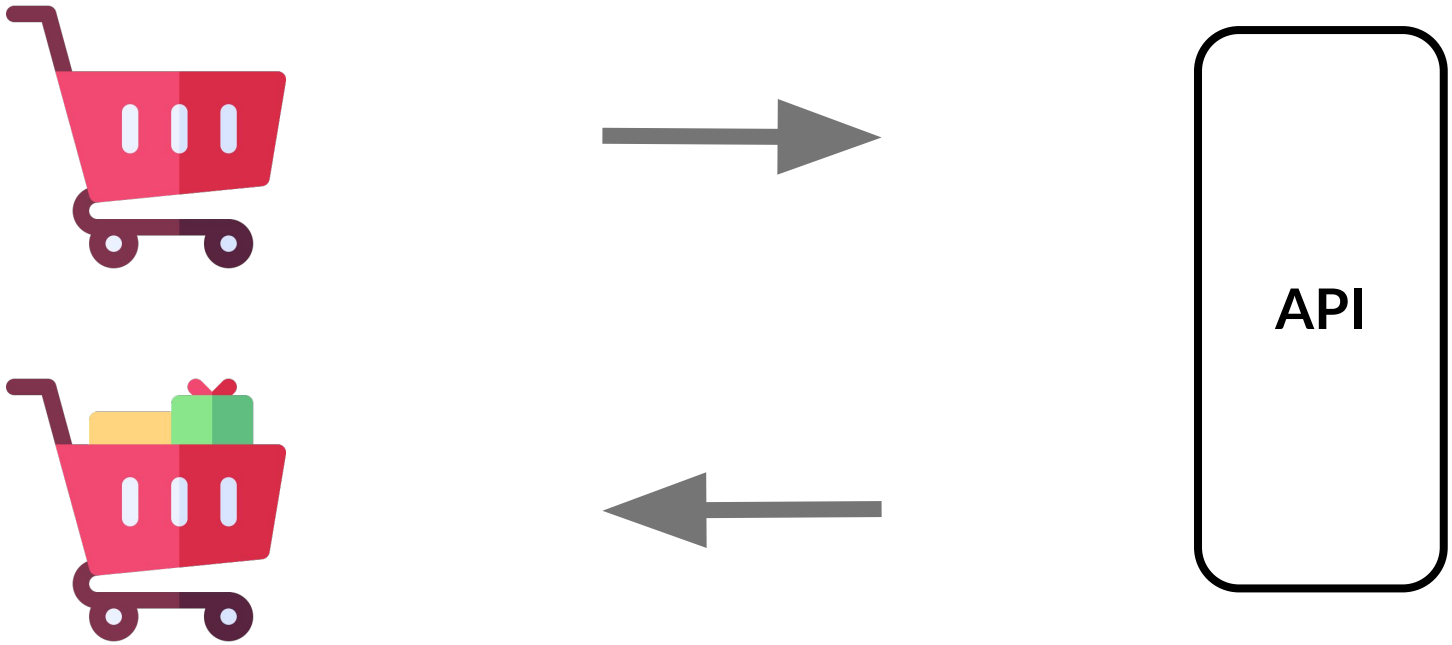
Assert →

```python
def test_return_cart():
  db[1] = {
    "id": 1,
    "lines": [
      {"product": 50776, "quantity": 2},
      {"product": 15778, "quantity": 4}
    ]
  }

  response = client.get("/cart/1")

  assert response.status_code == 200
  assert response.json() == {
    "id": 1,
    "lines": [
      {"product": 50776, "quantity": 2},
      {"product": 15778, "quantity": 4}
    ]
  }
```

¿Qué vamos a desarrollar?



Iteración 0

Un endpoint GET /cart/1 que recibimos una lista con los productos de ese carrito.

Test en rojo

Test en verde

Refactor!

Iteración 0

Un endpoint GET /cart/1 que recibimos una lista con los productos de ese carrito.

Test en rojo

Test en verde

Refactor!

```
def test_return_cart():
  db[1] = {
    "id": 1,
    "lines": [
      {"product": 50776, "quantity": 2},
      {"product": 15778, "quantity": 4}
    ]
  }

  response = client.get("/cart/1")

  assert response.status_code == 200
  assert response.json() == {
    "id": 1,
    "lines": [
      {"product": 50776, "quantity": 2},
      {"product": 15778, "quantity": 4}
    ]
  }
```

Iteración 0

Un endpoint GET /cart/1 que recibimos una lista con los productos de ese carrito.

Test en rojo

Test en verde

Refactor!

```
~/WorkshopTDD/src$ pytest
========================================= test session starts ==========================================
platform linux -- Python 3.8.12, pytest-7.1.3, pluggy-0.13.1
rootdir: /home/runner/WorkshopTDD/src, configfile: pytest.ini
plugins: anyio-3.6.1
collected 1 item

carts/tests.py F                                                                            [100%]

=============================================== FAILURES ===============================================
_____________________________________ test_return_cart _____________________________________

    def test_return_cart():
      response = client.get("/cart/1")

>     assert response.status_code == 200
E     assert 404 == 200
E      +  where 404 = <Response [404]>.status_code

carts/tests.py:10: AssertionError
======================================= short test summary info ========================================
```

Iteración 0

Un endpoint GET /cart/1 que recibimos una lista con los productos de ese carrito.

Test en rojo

Test en verde

Refactor!

```python
@app.get("/cart/{cart_id}")
def cart(cart_id: int):

    return {}
```

Iteración 0

Un endpoint GET /cart/1 que recibimos una lista con los productos de ese carrito.

Test en rojo

Test en verde

Refactor!

```
~/WorkshopTDD/src$ pytest
========================================= test session starts ==========================================
platform linux -- Python 3.8.12, pytest-7.1.3, pluggy-0.13.1
rootdir: /home/runner/WorkshopTDD/src, configfile: pytest.ini
plugins: anyio-3.6.1
collected 1 item

carts/tests.py F                                                                              [100%]

=============================================== FAILURES ===============================================
_____________________________________ test_return_cart ______________________________________

    def test_return_cart():
      response = client.get("/cart/1")

      assert response.status_code == 200
>     assert response.json() == {"id": 1, "lines": [{"product": 50776, "quantity": 2}, {"product": 1
5778, "quantity": 4}]}
E     AssertionError: assert {} == {'id': 1, 'li...uantity': 4}]}
E       Right contains 2 more items:
E       {'id': 1,
E        'lines': [{'product': 50776, 'quantity': 2},
E                  {'product': 15778, 'quantity': 4}]}
E       Use -v to get more diff

carts/tests.py:11: AssertionError
======================================= short test summary info ========================================
FAILED carts/tests.py::test_return_cart - AssertionError: assert {} == {'id': 1, 'li...uantity':
========================================== 1 failed in 0.82s ===========================================
~/WorkshopTDD/src$
```

Iteración 0

Un endpoint GET /cart/1 que recibimos una lista con los productos de ese carrito.

Test en rojo

Test en verde

Refactor!

```python
@app.get("/cart/{cart_id}")
def cart(cart_id: int):

    return db[cart_id]
```

Iteración 0

Un endpoint GET /cart/1 que recibimos una lista con los productos de ese carrito.

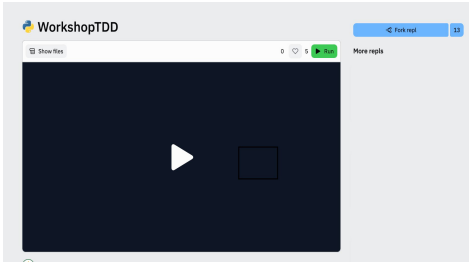
Test en rojo

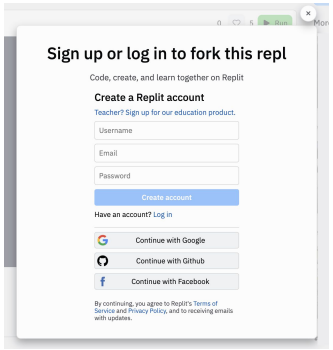
Test en verde

Refactor!

Entorno Repl.it

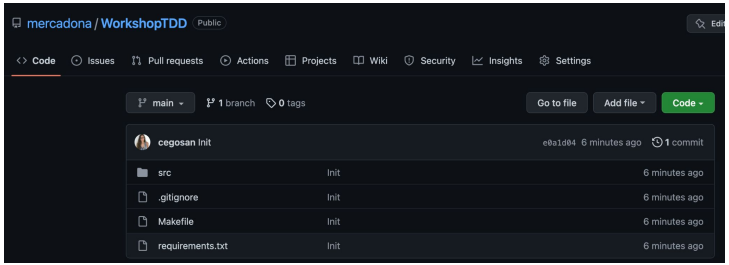
https://replit.com/@mercadonatech/WorkshopTDD





Repositorio

https://github.com/mercadona/WorkshopTDD



Ejecutamos:

- make install
- cd src/
- pytest

Iteración 1

Un endpoint POST /cart/1/products/add al que enviarle el producto que queremos añadir.

Test en rojo

Test en verde

Refactor!

```
{
  'product': 50776
}
```

>

```
{
  'id': 1,
  'lines': [
    {
      'product': 50776
    }
  ]
}
```

Iteración 1.2

Modificar el endpoint para poder aumentar la cantidad de un producto ya existente.

Test en rojo

Test en verde

Refactor!


```
{
  'product': 50776,
  'quantity': 2
}
```

›

```
{
  'id': 1,
  'lines': [
    {
      'product': 50776,
      'quantity': 7
    }
  ]
}
```

Iteración 2

Ahora, añadimos otro endpoint POST /cart/1/products/substract para poder quitar un producto de nuestro carrito.

Test en rojo

Test en verde

Refactor!


```
{
  'product': 15778,
  'quantity': 1
}
```

```
{
  'id': 1,
  'lines': [
    {
      'product': 50776,
      'quantity': 5
    }
  ]
}
```

Iteración 2.2

Modificamos el endpoint para poder reducir la cantidad de un producto de nuestro carrito.

Test en rojo

Test en verde

Refactor!


```
{
  'product': 50776,
  'quantity': 2
}
```

›

```
{
  'id': 1,
  'lines': [
    {
      'product': 50776,
      'quantity': 3
    }
  ]
}
```

Iteración 2.3

Modificamos el endpoint para que no se pueda reducir más de la cuenta una cantidad de un producto ya existente.

Test en rojo

Test en verde

Refactor!


```
{
  'product': 50776,
  'quantity': 5
}
```

Iteración 3

Ahora, queremos otro endpoint POST /cart/1/clear para poder quitar todos los productos de nuestro carrito.

Test en rojo

Test en verde

Refactor!


```
{
  'id': 1,
  'lines': []
}
```

Iteración 4

Queremos modificar todos los endpoints anteriores para que nos devuelvan el total de la compra.

Test en rojo

Test en verde

Refactor!


```
{
  'id': 1,
  'lines': [
    {
      'product': 50776,
      'quantity': 5,
      'price': 100
    },
    {
      'product': 15778,
      'quantity': 1,
      'price': 150
    }
  ],
  'total': 650
}
```

Iteración 5. Abstraer la lógica

Test en rojo

Test en verde

Refactor!

- Los tests de la vista se simplifican.
- Incrementas la velocidad del testing (adiós ciclos request-response).
- Sabes lo que hace tu aplicación con solo ver los tests de las acciones.
- Facilidad de desacople y aplicar patrones.
- Incorporación de nueva gente.