



Clase 21. Programación Backend

Trabajo con datos: Diseño de mocks



OBJETIVOS DE LA CLASE

- Comprender la técnica TDD.
- Entender el concepto de API y Mocking.
- Utilizar Faker.js para la generación de Mocking de datos.
- Realizar un servidor mocking basado en Node.js y Faker.js.

CRONOGRAMA DEL CURSO

Clase 20



DBaaS

Clase 21



**Trabajo con datos:
Mocks**

Clase 22



**Trabajo con datos:
Normaliación**

```
mirror_mod = modifier_ob.  
mirror object to mirror  
mirror_mod.mirror_object =  
operation == "MIRROR_X":  
mirror_mod.use_x = True  
mirror_mod.use_y = False  
mirror_mod.use_z = False  
operation == "MIRROR_Y":  
mirror_mod.use_x = False  
mirror_mod.use_y = True  
mirror_mod.use_z = False  
operation == "MIRROR_Z":  
mirror_mod.use_x = False  
mirror_mod.use_y = False  
mirror_mod.use_z = True
```

```
selection at the end -add  
mirror_ob.select= 1  
mirror_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier  
mirror_ob.select = 0  
bpy.context.selected_obj  
data.objects[one.name].sel  
print("please select exactly
```

```
--- OPERATOR CLASSES ---  
types.Operator):  
X mirror to the selected  
object.mirror_mirror_x"  
mirror X"  
context):  
context.active_object is not
```



```
def test_CanCalculateTotal(c  
checkout.addItem("a")  
assert checkout.calculate
```

TDD

TDD o Test-Driven Development (*desarrollo dirigido por tests*) es una **práctica de programación** que consiste en **escribir primero las pruebas** (generalmente unitarias), **después** escribir el **código fuente** que pase la prueba satisfactoriamente y, **por último**, **refactorizar** el código escrito.

Con esta práctica se consigue entre otras cosas un **código más robusto**, más **seguro, mantenible** y una mayor rapidez en el desarrollo.

TDD

¿QUÉ ES TDD?



Mejoran el código en cualquier momento.

1

La prueba debe fallar: Se muestran los fallos en rojo.



4

Refactoring: Se debe mejorar el código.



La lectura del código será mucho mejor al tener ejemplos de uso (las pruebas).

3

La prueba debe pasar: Las que pasan se muestran en verde.



Los equipos de testing, development y analyst serán más felices y eficientes.



2

Escribir el código mínimo para que la prueba pase



SE COMBINAN 2 METODOLOGÍAS

→ 1. **Test-first development:**
Escribir las pruebas primero.

→ 2. **Refactoring:**
Refactozación de código.

TDD (Test Drive Development) es una metodología de desarrollo, cuyo objetivo es crear primero las pruebas y luego escribir el software.

TDD

Ejemplo TDD: Algoritmo Calculadora

Vamos al IDE... 



***¿Alguna pregunta hasta
ahora?***

API MOCK

- Mocking es la técnica utilizada para **simular objetos en memoria** con la finalidad de poder **ejecutar pruebas unitarias**.
- Los **Mocks** son **objetos preprogramados** con expectativas que forman una especificación de las llamadas que se espera recibir.
- Los Mocks se pueden servir desde un servidor web a través de una **Mock API**.

***¿Qué es
Mocking y
Mock?***

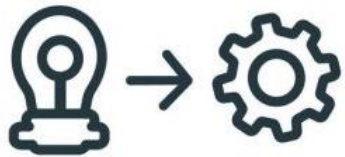
Mocks y TDD 



Utilizando los Mocks en TDD

- Al trabajar con TDD nos encontramos con la **dependencia** de ciertos **elementos** que pueden estar **fuera de contexto** con el sistema que queremos probar.
- Algunas **dependencias** pueden traer **efectos colaterales** sobre el resultado de las pruebas, lo que se traduce en **futuros errores**. Incluso pueden no estar (todavía) implementadas, al estar el sistema en una fase temprana de desarrollo.
- Para resolver este problema, **reemplazamos** las **dependencias** por los **mocks**. Así se devolverán los resultados esperados para hacer las peticiones a dichas dependencias, sin realizar ninguna operación real o compleja.
- Nos podemos valer de un **servidor de mocks** que **imita** el comportamiento de nuestro **servidor real**, devolviendo datos de prueba o datos esperados tras las peticiones que queremos poner a prueba.

Mocks y API 



Mocks implementados en una API

- Los mocks de API son una herramienta muy potente que permite **desarrollar y probar el front-end** como un **componente independiente del back-end**, facilitando y reduciendo tiempos de desarrollo, y aumentando la productividad del equipo.
- Un **mock del servidor** es sumamente útil para el equipo de desarrolladores que trabaja en la interfaz del usuario, ya que responde preguntas triviales y permite avanzar notablemente sin depender del desarrollo del backend
- De esta manera se evita tener que esperar a que el servidor esté terminado para poder empezar a desarrollar el frontend.

La **mock API** debe estar **bien diseñada y documentada**. Si hay errores en la especificación, habrá disparidad en el comportamiento de los mocks, causando que el frontend no termine de encajar cuando se haga el cambio al backend real.

Mocks implementados en una API





***¿Alguna pregunta hasta
ahora?***



Random array

Tiempo: 10 minutos



- 1) Desarrollar un servidor basado en Node.js y express que para la **ruta '/test'** responda con un **array** de 10 objetos, con el siguiente formato:

```
{  
  nombre: '',  
  apellido: '',  
  color: ''  
}
```

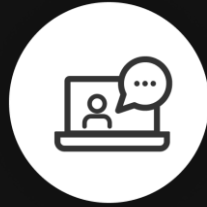
- 1) Los objetos generados tendrán un **valor aleatorio** para cada uno de sus campos. El valor será obtenido de los siguientes arrays:

```
const nombres = ['Luis', 'Lucía', 'Juan', 'Augusto', 'Ana']
```

```
const apellidos = ['Pieres', 'Cacurri', 'Bezzola', 'Alberca', 'Mei']
```

```
const colores = ['rojo', 'verde', 'azul', 'amarillo', 'magenta']
```

- 1) Con cada request se obtendrán valores diferentes.



***¿Alguna pregunta hasta
ahora?***

FAKER.js




CODER HOUSE

- Faker.js es una **librería Javascript** que nos permite **generar** varios tipos de **datos aleatorios** como nombres, dirección de correo electrónico, perfil de avatar, dirección, cuenta bancaria, empresa, título del trabajo y mucho más.
- Faker.js se puede **utilizar dentro** de un proyecto **Node.js** para generar un mocking de datos para ser servidos desde un proyecto implementado con **Express**.
- Se instala en un proyecto Node.js es a través del comando **npm i @faker-js/faker**
- **A continuación veremos un ejemplo de uso**

***¿Qué es
Mocking y
Mock?***

Faker website: <https://fakerjs.dev/>


 **Faker**


[Guide](#) [API](#) [Ecosystem](#) [About](#) [v7.6.0](#) [🌙](#) [🐦](#) [💬](#) [🔄](#)

Faker

Generate massive amounts of fake (but realistic) data for testing and development.


[Get Started](#) [View on GitHub](#)






Products

Generate Prices, Product Names, Adjectives, and Descriptions.




Finance

Create stubbed out Account Details, Transactions, and Crypto Addresses.




Addresses

Generate valid Addresses, Zip Codes, Street Names, States, and Countries!




Hacker Jargon

"Try to reboot the SQL bus, maybe it will bypass the virtual application!"



Time-based Data

Past, present, future, recent, soon... whenever!



Localization

Set a locale to generate realistic looking Names, Addresses, and Phone Numbers.

Released under the MIT License.
Copyright © 2022-present Faker.

Vamos al IDE... 



***¿Alguna pregunta hasta
ahora?***

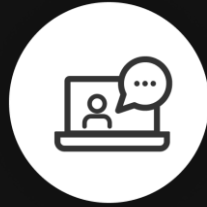


Random array con Faker

Tiempo: 10 minutos



- Reformar el ejercicio anterior utilizado **Faker** para generar los objetos con **datos aleatorios en español**.
- A la ruta '/test' se le podrá pasar por query params la **cantidad de objetos** a generar.
- *Ej: '/test?cant=30'*.
- De no pasarle ningún valor, producirá 10 objetos.
- Incorporarle **id** a cada uno de los objetos generados en forma incremental, empezando por 1.



***¿Alguna pregunta hasta
ahora?***



BREAK

¡10 MINUTOS Y VOLVEMOS!

API Mock con Node.js

***Proyecto ejemplo en vivo:
Servidor Mock REST API
con Node.js y Faker.js***



Acciones del servidor

- El proyecto tiene cinco rutas:
 - **POST /api/usuarios/popular?cant=n** : si no específico cant me genera 50 objetos mock
 - **GET /api/usuarios/:id?** : con id me trae un mock; sin id devuelve todos los mocks
 - **POST /api/usuarios** : incorpora un nuevo mock
 - **PUT /api/usuarios/:id** : actualiza un mock total o parcialmente por campo
 - **DELETE /api/usuarios/:id** : borra un mock específico
- Los usuarios tienen: nombre, email, website, e imagen.
- Cada una puede generar, listar, incorporar, actualizar y borrar mock
- Los datos son persistentes en memoria.

Vamos al IDE... 

¿PREGUNTAS?



¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- TDD
- Mock de APIs
- Faker.js
- Servidor Mock API con Node.js



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDOLAEDUCACIÓN