CODER HOUSE

Clase 21. Programación Backend

Trabajo con datos: Diseño de mocks

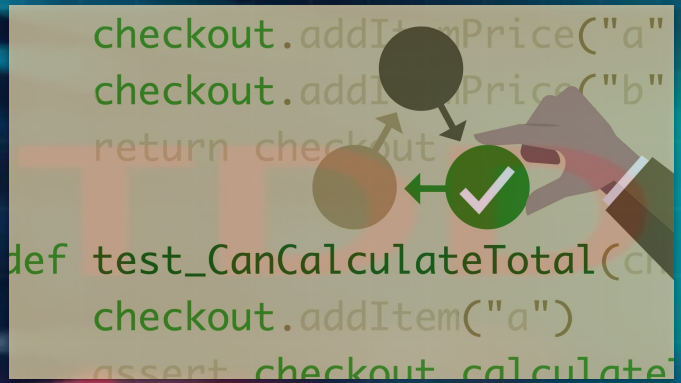
OBJETIVOS DE LA CLASE

- 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	Clase 21	Clase 22
DBaaS	**Trabajo con datos: Mocks**	**Trabajo con datos: Normaliación**

CODER HOUSE



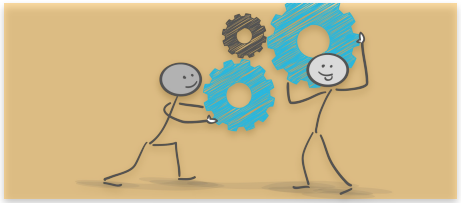
TDD

CODER HOUSE

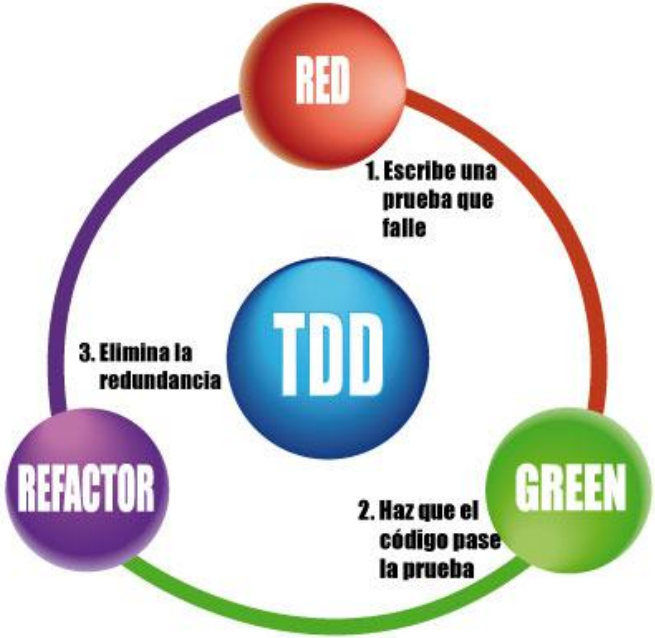
¿Que es TDD?

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.



Esquemas de concepto



CODER HOUSE

Ejemplo TDD: Algoritmo Calculadora

Fuente: Alfred Mut - bolufer blog

CODER HOUSE

1. Supongamos que el cliente nos pide que desarrollemos una **calculadora que sume números**.
2. Acordamos que el **criterio de aceptación** sería que si introduces en la calculadora dos números y le das a la operación de suma, la calculadora te muestra el resultado de la suma en la pantalla.
3. Partiendo de este criterio, comenzamos a **definir** el **funcionamiento del algoritmo** de suma y **convertimos** el **criterio de aceptación** en una **prueba concreta**. Por ejemplo: un algoritmo al que al introducir 3 y 5 devuelve 8:

```
function testSuma() { assertEquals(8, Calculadora.suma(3,5)); }
```

Análisis del último paso

- Es el punto es el más importante del TDD y que supone un cambio de mentalidad: **primero escribo cómo debe funcionar mi programa** y una vez que lo tengo claro, paso a codificarlo.
- Al escribir el test estamos diseñando cómo va a funcionar el software. Para cubrir la prueba vamos a necesitar una clase ‘Calculadora’ con una función que se llame ‘Suma’ y que tenga dos parámetros.
- Esta clase todavía no existe pero **cuando la creemos, sabremos cómo va a funcionar**. Por supuesto, si intentamos pasar este test nos dará un error, porque la clase Calculadora aún no existe.

Escritura del código y prueba

- Ahora pasamos a **escribir el código** de la clase. Será fácil, porque ya sabemos exactamente cómo se va a comportar:

```
class Calculadora {
    static suma (a, b) {
        const c = a + b;
        return c;
    }
}
```

- **Ejecutamos la prueba** y ya tenemos el código funcionado con la prueba pasada.

Refactorización

- Una vez que todo esté funcionando, pasamos a refactorizar y a eliminar código duplicado. Este ejemplo es sencillo y en un caso real no haríamos tantos pasos para algo tan evidente, pero el código mejorado podría ser:

```
class Calculadora {
    static suma (a, b) {
        return a+b;
    }
}
```

- En ejemplos más complejos, deberíamos buscar código duplicado y agruparlo en funciones, o utilizar la herencia o el polimorfismo.

Detalles a tener en cuenta

- Es importante pasar todos los **tests después de refactorizar** por si nos olvidado de algo.
- Ahora deberíamos volver al punto 3 con **tests más complicados** y repetir el proceso. Por ejemplo, podríamos hacer que el algoritmo admita sumar números decimales.
- Esta forma de trabajar es también muy buena para **entender el código**. La calidad del diseño de un software está también relacionada con el conocimiento del equipo de desarrollo en relación al dominio en cuestión.

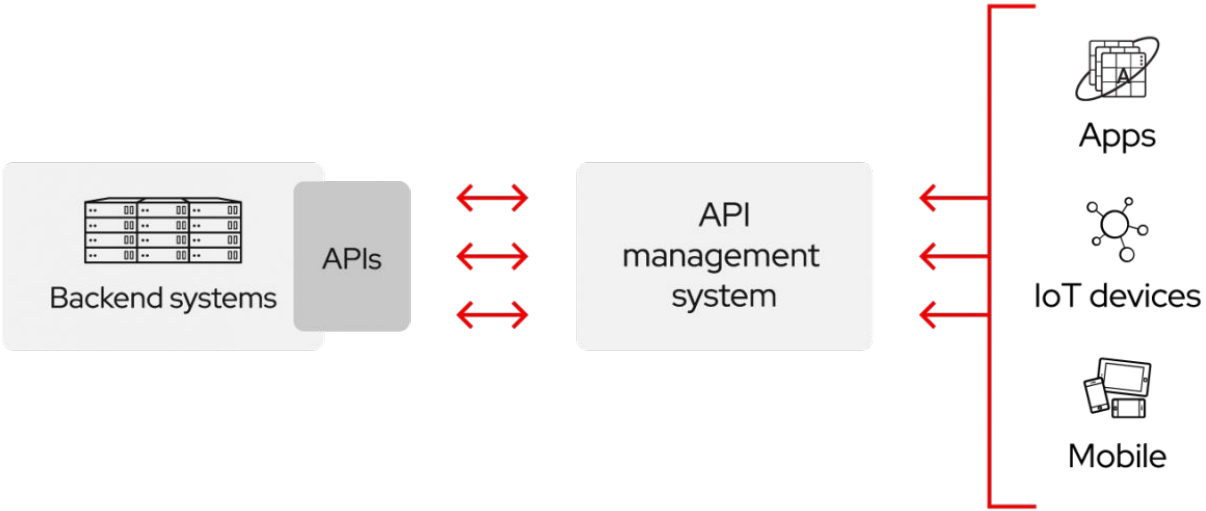
El mock de APIs

CODER HOUSE

API
Application Programming Interface

¿Qué es una API?

API significa **interfaz de programación de aplicaciones**. Es un conjunto de definiciones y protocolos que se utilizan para desarrollar e **integrar el software de las aplicaciones**.



CODER HOUSE

API: Conceptos

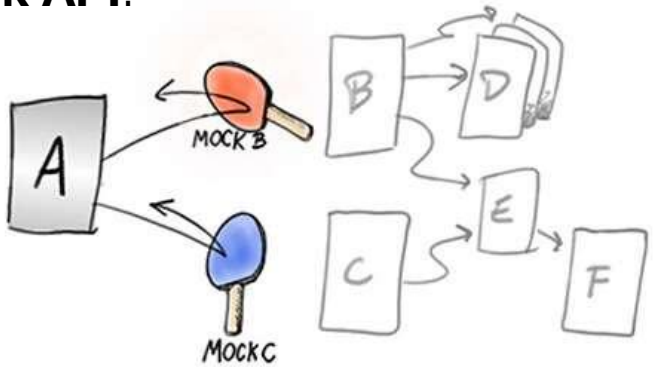
- Permiten que sus **productos y servicios se comuniquen con otros** sin necesidad de saber cómo están implementados.
- Esto **simplifica el desarrollo de las aplicaciones** y permite ahorrar tiempo y dinero.
- **Otorgan flexibilidad**, proporcionan **oportunidades de innovación** y simplifican el diseño, la administración y el uso de las aplicaciones.
- Esto es ideal al momento de diseñar herramientas y productos nuevos (o gestionar los actuales).

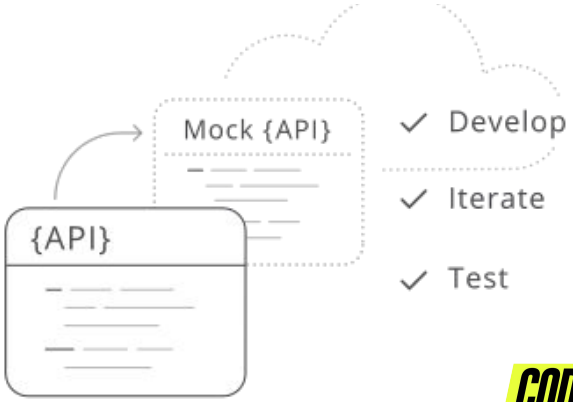
API: Conceptos

Las API se consideran como **contratos**, con documentación que representa un **acuerdo entre las partes**: si una de las partes envía una solicitud remota con cierta estructura, esa misma estructura determinará cómo responderá el software de la otra parte.

¿Qué es Mocking y 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**.





CODER HOUSE

Mocks y TDD

CODER HOUSE

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

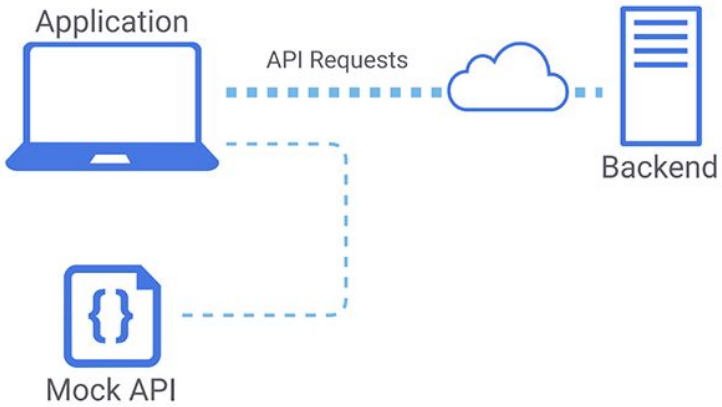
CODER HOUSE

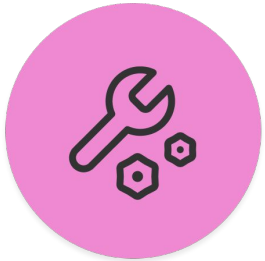
Mocks implementados en una API

- 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.

Mocks implementados en una API

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.



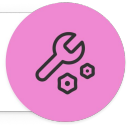


Random array

Tiempo: 10 minutos

CODER HOUSE

Desafío generico



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: ''
}
```

2) 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']
```

3) Con cada request se obtendrán valores diferentes.

CODER HOUSE

FAKER.js

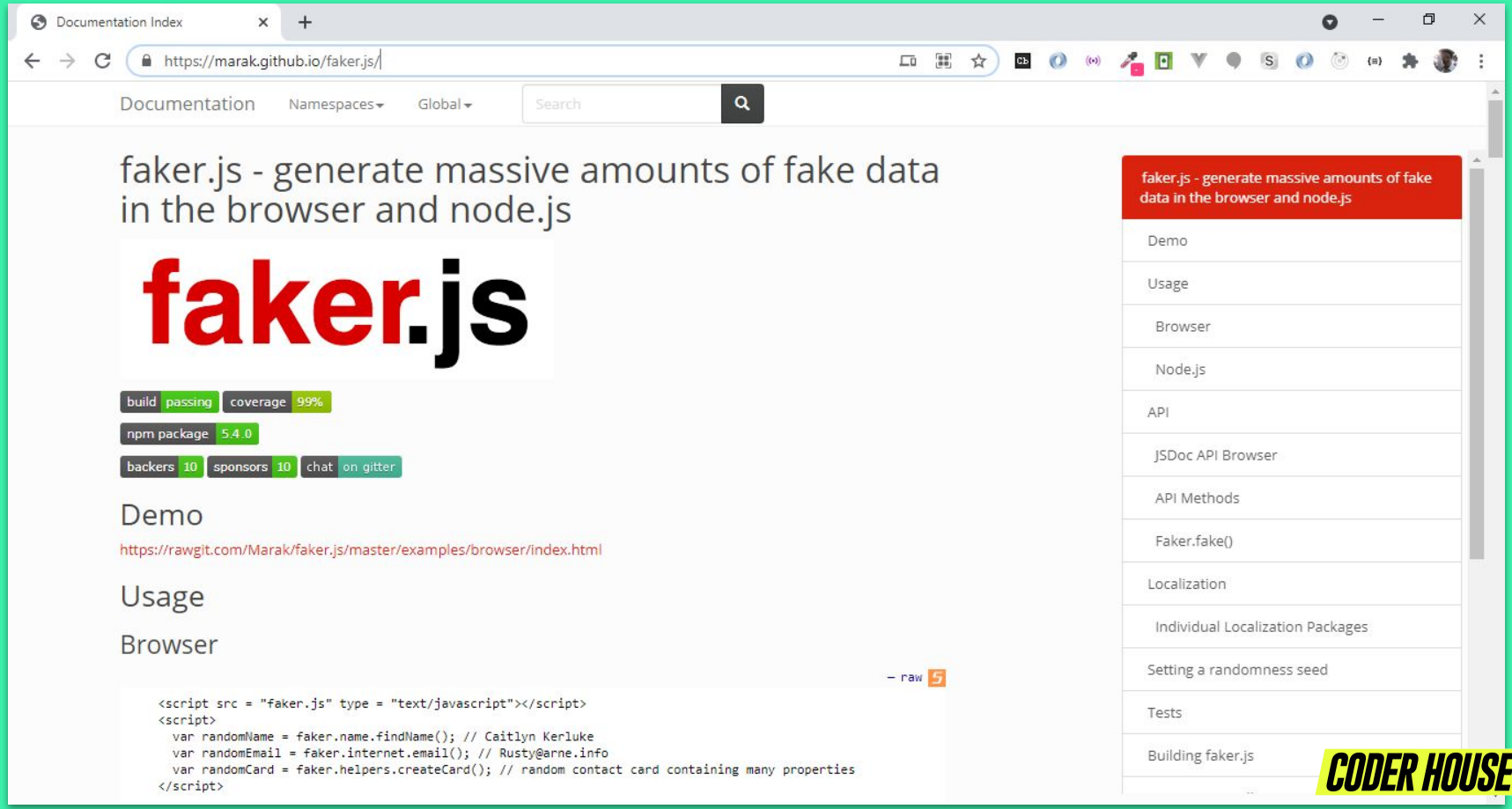
CODER HOUSE

FAKER.js

faker.js

- 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**
- **A continuación veremos un ejemplo de uso**

Faker website: https://marak.github.io/faker.js/



CODER HOUSE

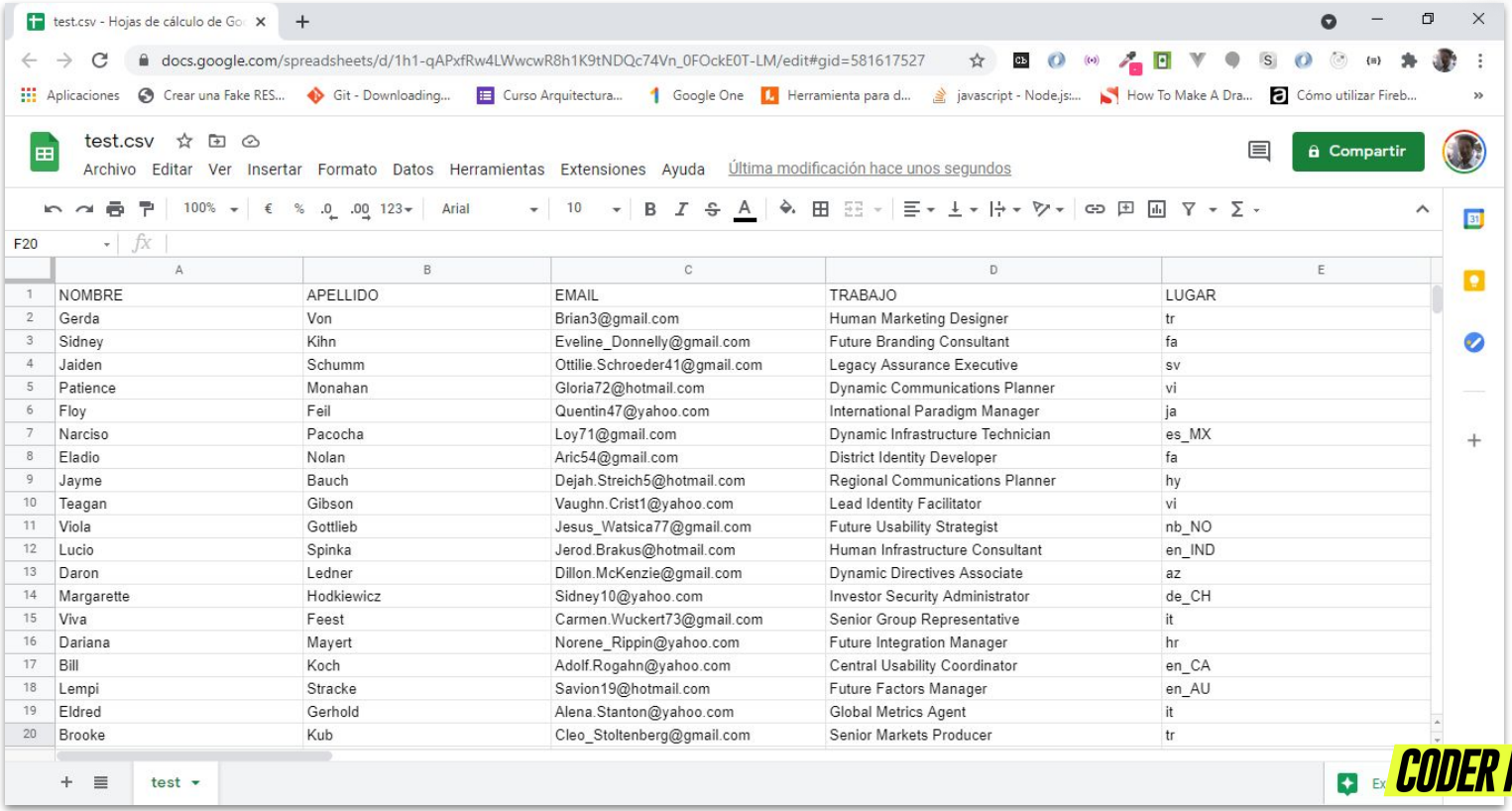
FAKER.js: ejemplo de uso

Este código nos permite crear un archivo test.csv con información de relleno

```
const faker = require('faker')
const fs = require('fs')

var str = 'NOMBRE;APELLIDO;EMAIL;TRABAJO;LUGAR\r\n'

for(let i=0; i<100; i++) {
    str += faker.name. firstName ( ) +
            ';' + faker.name.lastName() +
            ';' + faker.internet.email() +
            ';' + faker.name. jobTitle() +
            ';' + faker.random.locale() +
            '\r\n'
}

fs.writeFile ( './test.csv' , str, function ( err ) {
    if( err ) console.log( err );
    console.log( 'archivo guardado' )
})
```

CODER HOUSE

FAKER.js: ejemplo de uso



test.csv - Hojas de cálculo de Go

docs.google.com/spreadsheets/d/1h1-qAPxfRw4LWwcwR8h1K9tNDQc74Vn_0FOckE0T-LM/edit#gid=581617527

Aplicaciones | Crear una Fake RES... | Git - Downloading... | Curso Arquitectura... | Google One | Herramienta para d... | javascript - Node.js:... | How To Make A Dra... | Cómo utilizar Fireb...

test.csv

Archivo Editar Ver Insertar Formato Datos Herramientas Extensiones Ayuda Última modificación hace unos segundos

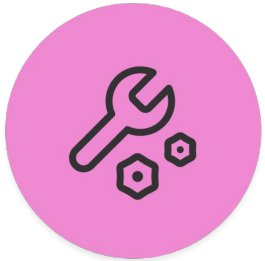
Compartir

F20

	A	B	C	D	E
1	NOMBRE	APELLIDO	EMAIL	TRABAJO	LUGAR
2	Gerda	Von	Brian3@gmail.com	Human Marketing Designer	tr
3	Sidney	Kihn	Eveline_Donnelly@gmail.com	Future Branding Consultant	fa
4	Jaiden	Schumm	Ottilie.Schroeder41@gmail.com	Legacy Assurance Executive	sv
5	Patience	Monahan	Gloria72@hotmail.com	Dynamic Communications Planner	vi
6	Floy	Feil	Quentin47@yahoo.com	International Paradigm Manager	ja
7	Narciso	Pacocha	Loy71@gmail.com	Dynamic Infrastructure Technician	es_MX
8	Eladio	Nolan	Aric54@gmail.com	District Identity Developer	fa
9	Jayme	Bauch	Dejah.Streich5@hotmail.com	Regional Communications Planner	hy
10	Teagan	Gibson	Vaughn.Crist1@yahoo.com	Lead Identity Facilitator	vi
11	Viola	Gottlieb	Jesus_Watsica77@gmail.com	Future Usability Strategist	nb_NO
12	Lucio	Spinka	Jerod.Brakus@hotmail.com	Human Infrastructure Consultant	en_IND
13	Daron	Ledner	Dillon.McKenzie@gmail.com	Dynamic Directives Associate	az
14	Margarette	Hodkiewicz	Sidney10@yahoo.com	Investor Security Administrator	de_CH
15	Viva	Feest	Carmen.Wuckert73@gmail.com	Senior Group Representative	it
16	Dariana	Mayert	Norene_Rippin@yahoo.com	Future Integration Manager	hr
17	Bill	Koch	Adolf.Rogahn@yahoo.com	Central Usability Coordinator	en_CA
18	Lempi	Stracke	Savion19@hotmail.com	Future Factors Manager	en_AU
19	Eldred	Gerhold	Alena.Stanton@yahoo.com	Global Metrics Agent	it
20	Brooke	Kub	Cleo_Stoltenberg@gmail.com	Senior Markets Producer	tr

test

CODER HOUSE



Random array con Faker

Tiempo: 10 minutos

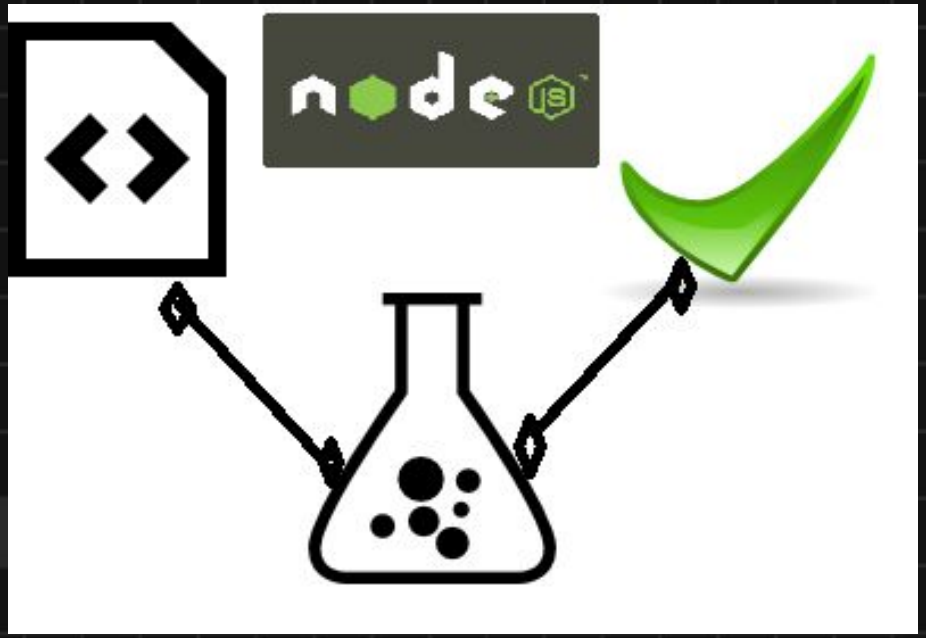
CODER HOUSE

- 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.

BREAK

¡5/10 MINUTOS Y VOLVEMOS!

Servidor Mock API con Node.js



CODER HOUSE

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

CODER HOUSE

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 mocks.
- Los datos son persistentes en memoria.

Ejemplo en vivo: server

```
import express, { json } from 'express'
import UsuariosRouter from './router/usuarios.js'

const app = express()

app.use(json())

app.use('/api/usuarios', new UsuariosRouter())

const PORT = 8080
const server = app.listen(PORT, () => {
   console.log(`Servidor escuchando en el puerto ${PORT}`)
})
server.on('error', error => console.log(`Error en servidor: ${error}`))
```

CODER HOUSE

Ejemplo en vivo: router

```
import express from 'express'
import ApiUsuariosMock from '../api/usuarios.js'

class UsuariosRouter extends express.Router {
   constructor() {
       super()

       const apiUsuarios = new ApiUsuariosMock()

       this.post('/popular', async (req, res, next) => {
           try {
               res.json(await apiUsuarios.popular(req.query.cant))
           } catch (err) {
               next(err)
           }
       })

       //...
```

Ejemplo en vivo: router

```
//...
this.get('/', async (req, res, next) => {
    try {
        res.json(await apiUsuarios.listarAll())
    } catch (err) {
        next(err)
    }
})
this.get('/:id', async (req, res, next) => {
    try {
        res.json(await apiUsuarios.listar(req.params.id))
    } catch (err) {
        next(err)
    }
})
//...
```

CODER HOUSE

Ejemplo en vivo: router

```
//...
this.post('/', async (req, res, next) => {
    try {
        res.json(await apiUsuarios.guardar(req.body))
    } catch (err) {
        next(err)
    }
})
this.put('/:id', async (req, res, next) => {
    try {
        res.json(await apiUsuarios.actualizar({ ...req.body, id: req.params.id }))
    } catch (err) {
        next(err)
    }
})
//...
```

Ejemplo en vivo: router

```
//...
this.delete('/:id', async (req, res, next) => {
    try {
        res.json(await apiUsuarios.borrar(req.params.id))
    } catch (err) {
        next(err)
    }
})
//...
```

CODER HOUSE

Ejemplo en vivo: router

```
        //...
        this.use((err, req, res, next) => {
            const erroresNoEncontrado = [
                'Error al listar: elemento no encontrado',
                'Error al actualizar: elemento no encontrado',
                'Error al borrar: elemento no encontrado'
            ]

            if (erroresNoEncontrado.includes(err.message)) {
                res.status(404)
            } else {
                res.status(500)
            }
            res.json({ message: err.message })
        })
    }
}

export default UsuariosRouter
```

Ejemplo en vivo: api de prueba

```
import ContenedorMemoria from '../contenedores/ContenedorMemoria.js'
import { generarUsuario } from '../utils/generadorDeUsuarios.js'

class ApiUsuariosMock extends ContenedorMemoria {
    constructor() {
        super()
    }

    popular(cant = 50) {
        const nuevos = []
        for (let i = 1; i <= cant; i++) {
            const nuevoUsuario = generarUsuario()
            const guardado = this.guardar(nuevoUsuario)
            nuevos.push(guardado)
        }
        return nuevos
    }
}

export default ApiUsuariosMock
```

Ejemplo en vivo: generador de usuarios de prueba

```
import faker from 'faker'
faker.locale = 'es'

function generarUsuario() {
 return {
   nombre: faker.name.findName(),
   email: faker.internet.email(),
   website: faker.internet.url(),
   image: faker.image.avatar(),
 }
}

export { generarUsuario }
```

CODER HOUSE

¿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

CODER HOUSE