CFP
Programador full-stack

MVC

¿Qué es MVC?

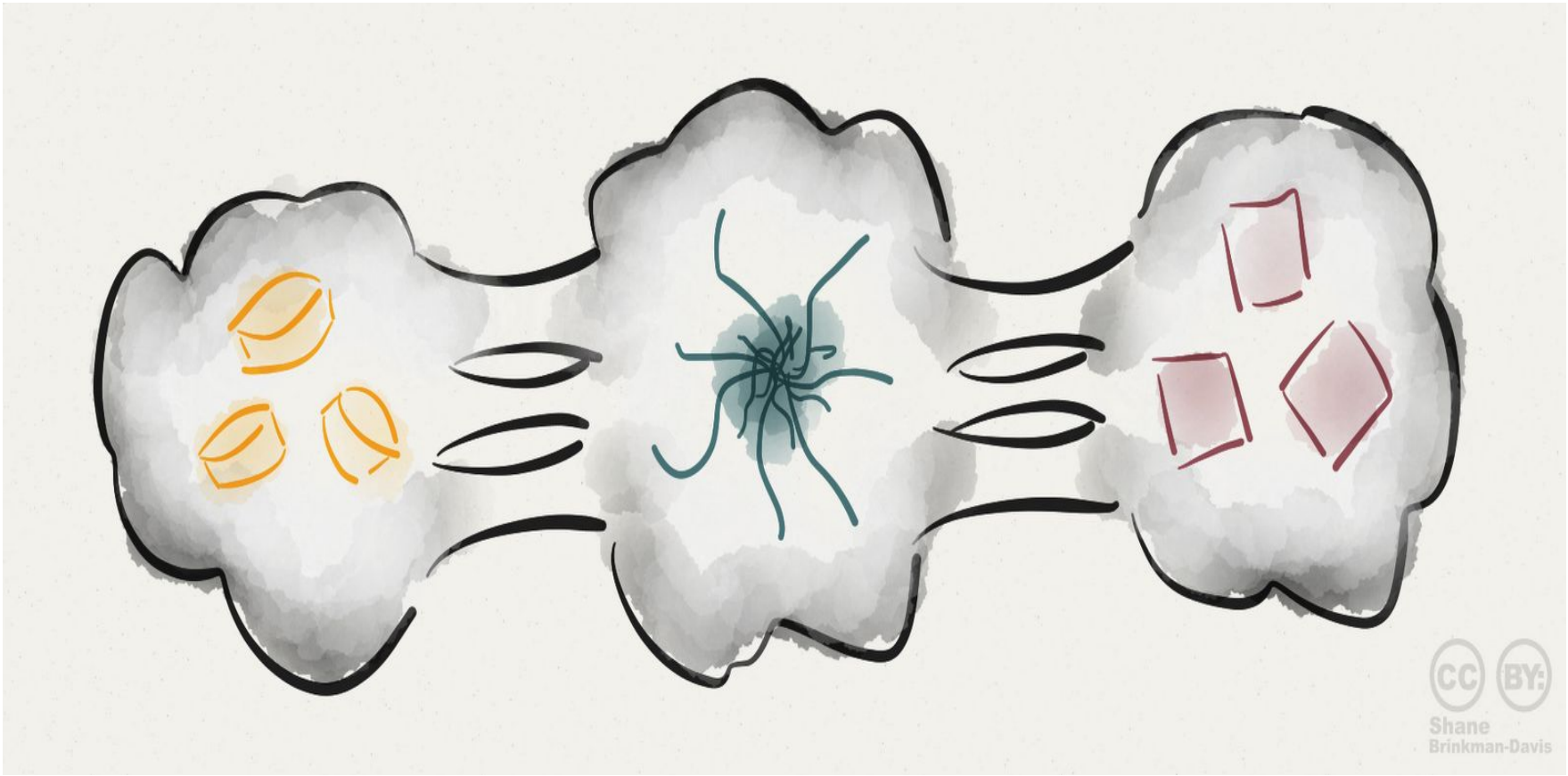
- **M**odel **V**iew **C**ontroller
- Patrón de *diseño*
- Utilizado en ampliamente en la industria


¿Qué es MVC?

- Separar responsabilidades
  - **Modelo**: Acceso a datos
  - **Vista**: Interfaz de usuario (Front End)
  - **Controlador**: Logica del negocio

Tengo una aplicación

Separo en partes

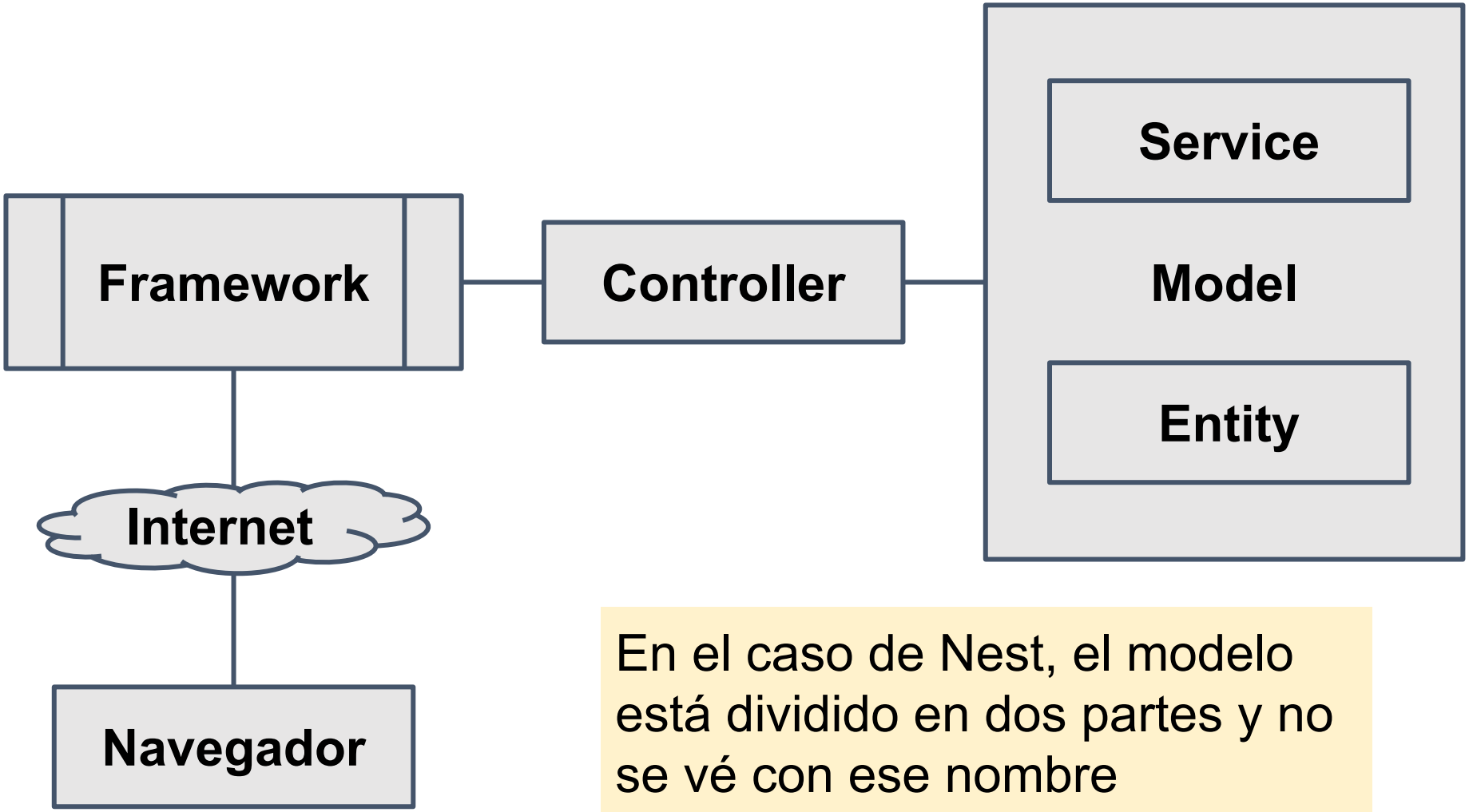


Vista - Responsabilidades

- Se encarga de la presentación los datos
- Ej: Comunicar información al usuario
  - Imágenes, fuentes, estilos, etc. (Front end)
- En el caso de una API (devuelven JSON) generlamente es código estándar que no hacemos nosotros
- O sea, si generaramos HTML va aca

Modelo - Responsabilidades

- Proteger y persistir los datos del usuario
- Asegurar la integridad y consistencia de datos
- Proveer métodos para
  - Consultar Datos
  - Modificar / Borrar Datos
- En este componente manejamos el guardado de datos y todo lo relacionado a lógica específica



En el caso de Nest, el modelo está dividido en dos partes y no se vé con ese nombre

Service

Un servicio en NestJS es responsable de almacenar y devolver datos. En general, son usados por los controladores, aunque un servicio puede llamar a otro servicio.

Es donde está implementada la lógica de negocio. En el módulo de POO se correspondería con la clase gestora.

La lógica de negocio NUNCA debe ir en el controller.

Controller

Se encarga de recibir los pedidos al backend.

Controla el ruteo, y maneja los detalles de la comunicación en sí, delegando tareas a otras partes más específicas. Por ejemplo, puede delegar controles de tipos de los parámetros y de lógica de negocio.

En una aplicación hay varios controladores, y cada uno tiene varias rutas relacionadas entre sí.

Parámetros en la URL

Por ejemplo, si tenemos un endpoint como

/mostrar/2

Que devuelve "ud ingresó un 2 como parámetro"
(y sirve para /mostrar/lo-que-sea)

En el controller podemos poner

@Get(':id')

También es posible tener más de un parámetro:

@Get(':arg1/fijo/:arg2/:arg3/')

Parametros en la URL: Ejercicio

Hacer una calculadora que funcione con urls y permita calcular el resultado de sumar, restar, multiplicar o dividir dos números. Ejemplos:

.../sumar/2/3 (devuelve 5)

.../multiplicar/2/3 (devuelve 6)

Parametros en la URL: Solucion - controlador

```
import { Controller, Get, Param } from '@nestjs/common';
import { CalcularService } from './calcular.service';


@Controller('calcular')
export class CalcularController {
    constructor (private calcularService : CalcularService) {}

    @Get(':oper/:ope1/:ope2')
    ejecutar(@Param('oper') oper, @Param('ope1') ope1, @Param('ope2') ope2) : string {
            let num1=parseInt(ope1);
            let num2=parseInt(ope2);
            return this.calcularService.getResultado(oper, num1, num2);
    }
}
```

Parametros en la URL: Solucion - servicio

```
import { Injectable } from '@nestjs/common';

@Injectable()
export class CalcularService {
	public getResultado(ope : string, op1 : number, op2 : number) : any  {
		let resultado=null;
		switch (ope) {
			case '+':
				resultado = { "resultado" : `${op1+op2}` };
				break;
			case '-':
				resultado = { "resultado" : `${op1-op2}` };
				break;
			case '*':
				resultado = { "resultado" : `${op1*op2}` };
				break;
			case '÷':
				resultado = { "resultado" : `${op1/op2}` };
				Break;
			case '^':
				resultado = { "resultado" : `${Math.pow(op1,op2)}` };
				Break;
		}
		return resultado;
	}
}
```

Inyección de dependencias

Las dependencias son servicios o objetos que una clase necesita para lograr sus objetivos. Dependency injection (DI) es un patrón de diseño a nivel de código que permite aumentar la modularidad del código.

Con DI la clase pregunta por sus dependencias al exterior (al framework) en lugar de crearlas ella misma.

Inyección de dependencias

Esto permite que en diferentes lugares el framework le pase diferentes subclases. La clase que las no necesita saber que tipo le pasaron específicamente

Esto permite hacer aplicaciones más flexibles, eficientes, robustas, testeables y mantenibles.

Este patrón permite que los objectos creen relaciones entre ellos, pero que el “cableado ("wiring up") de las instancias sea manejado por el framework.

Inyección en NestJS

Providers: Es el nombre que le da a las cosas inyectables.

```
import { Injectable } from '@nestjs/common';

@Injectable()
export class ProductoService {
```

Usar una dependencia inyectada

Para usar dependencias inyectadas, declaro que las necesito (“las pido”) en el constructor.

Esto se llama “constructor-based injection”.

```
@Controller('productos')
export class ProductoController {
    constructor(private productoService: ProductoService) {}

    @Get()
    public getProducto(): string {
        return this.productoService.getProducto()
    }
```

CFP Programador full-stack

Programar un back-end

Desarrollo Full Stack

Al programar algo full stack al surgir un error es difícil encontrar dónde viene.

Por eso es importante programar cada parte por separado, front y back end.

Es útil tener herramientas para probar cada una por separado:

- Mocks: permiten usar el front end sin el back end real
- Postman: permite hacer pedidos al a API sin usar el front end

Programar una API

- Como vamos a programar la API necesitamos algo para consumirla (llamarla/invocarla)
- Si programamos el Javascript y hay un error, no sabemos si el error está en el JS o en el PHP
- Vamos a usar una herramienta para invocar al servicio web

POSTMAN

Advanced REST client

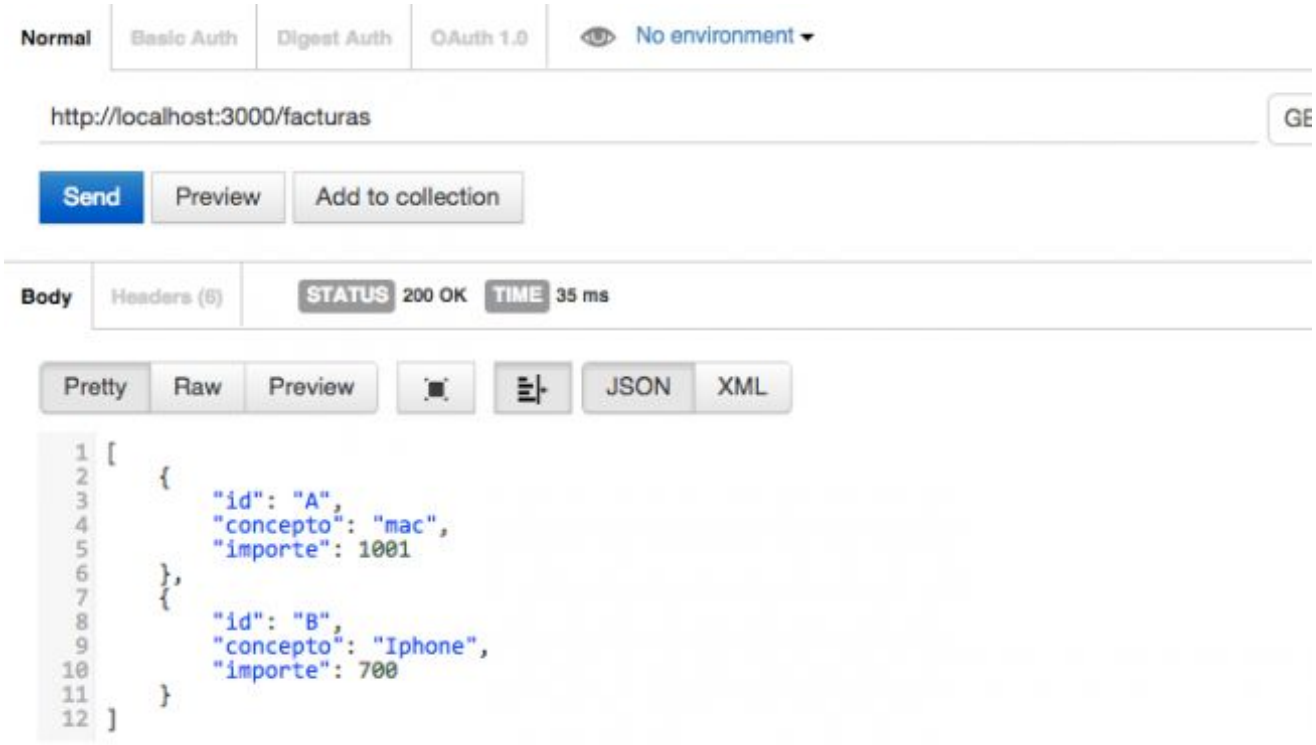
Postman

- Permite construir y gestionar peticiones a servicios REST (POST, GET, Etc.).
- Definir la petición que a realizar.

- Le damos enviar y la petición será lanzada contra nuestro servidor. Espera la respuesta (XML/JSON/Texto).

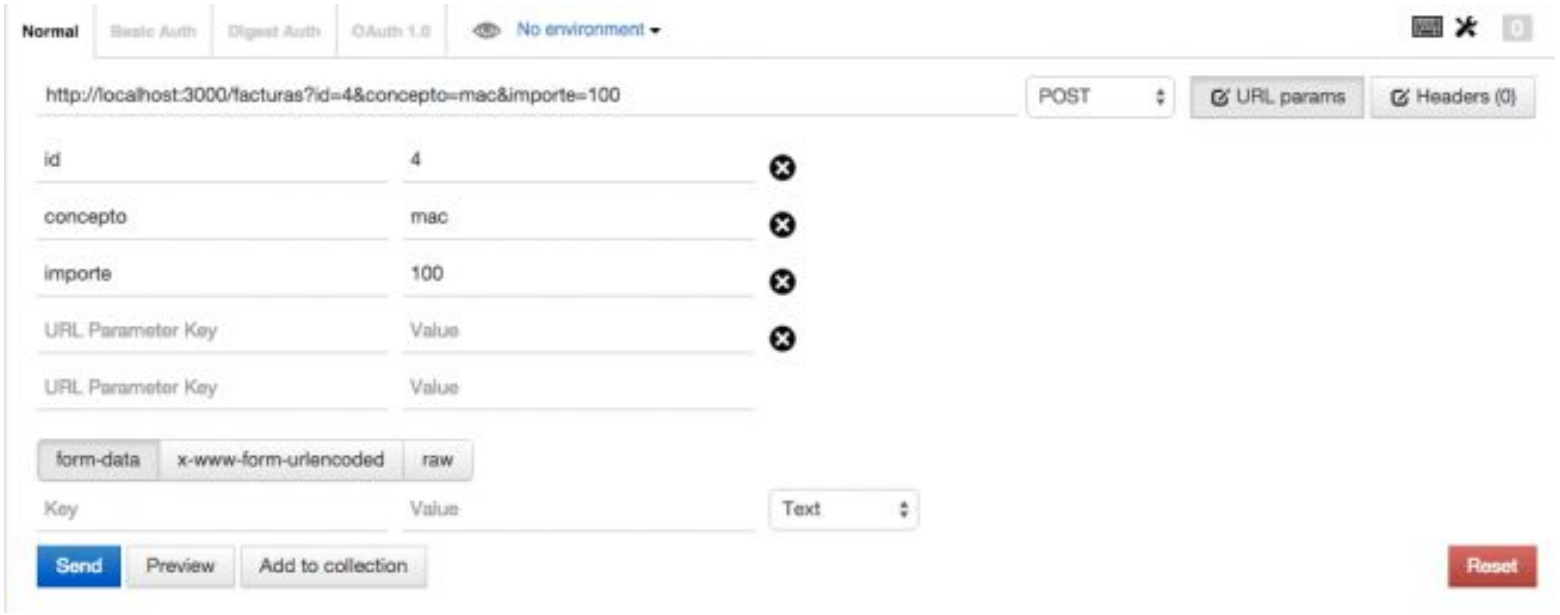
Postman

- Captura las respuestas y muestra el resultado de una forma clara y ordenada.

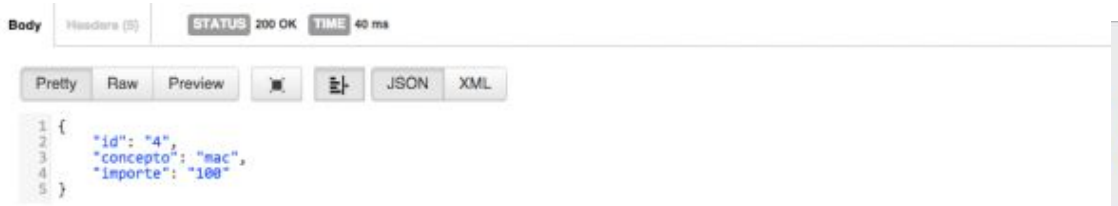


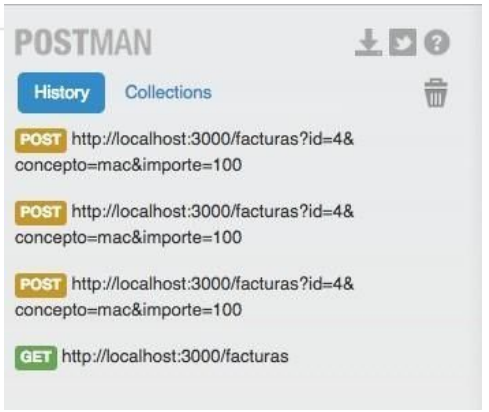
Postman

- Puede realizar peticiones de tipo GET o configurar a medida peticiones de tipo POST
- Mostrará el resultado igual que en el caso anterior

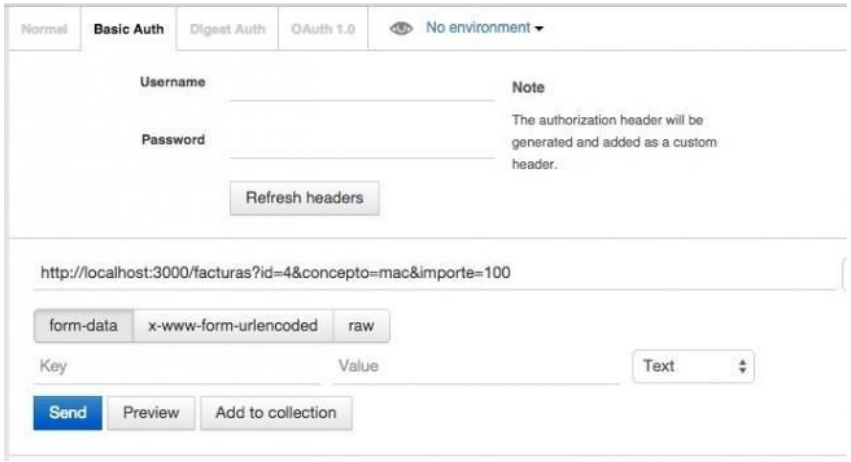


Postman





- Hay disponible una barra de historial.



- Permite gestionar funcionalidades como la autenticación.

CFP Programador full-stack

Guardar los datos

Entidad

Creemos una clase Producto que nos guarde nuestros datos:

```
Producto
    nombreProducto: string
    precio: number
```

Archivo

Vamos a tener nuestros datos guardados en un CSV (productos.csv)

```
producto_0,45
producto_1,17
producto_2,33
producto_3,91
producto_4,13
producto_5,6
```

Service

La clase producto.service.ts debe tener:

Un método load() que carga los productos del archivo

```
    private loadProductos(): void {
        let archivo = fs.readFileSync('productos.csv', 'utf8');
        const elementos = archivo.split('\n')
                    .map(p => p.replace('\r', '')).map(p => p.split(','));
        this.listaProductos = [];
        for (let i = 0; i < elementos.length; i++) {
            let producto = new Producto(elementos[i][0],
parseInt(elementos[i][1]));
            this.listaProductos.push(producto);
        }
    }
```

Service

La clase producto.service.ts debe tener:

Un método para devolver el arreglo de productos

```
public getProductos(): Producto[] {
    return this.listaProductos;
}
```

Controller

La clase producto.service.ts debe tener:

Un método para devolver el arreglo de productos

```
@Controller('productos')
export class ProductoController {
   constructor(private productoService: ProductoService) {}

   @Get()
   public getProductos(): Producto[] {
       return this.productoService.getProductos();
   }
}
```

Resultado GET

CFP
Programador full-stack

GET Individual

GET Individual

La API tiene que permitir traer un solo producto:

/productos/:ID

Service

La clase producto.service.ts debe tener:

Un método para devolver el arreglo de productos

```
public getProducto(index: number): Producto {
    // Más adelante agregar manejo de status code
    if (index < 0 || index >= this.listaProductos.length)
        return null;


    return this.listaProductos[index];
}
```

Controller

A la clase producto.service.ts le agregamos un método para devolver UN SOLO productos

```
import { Controller, Get, Param } from '@nestjs/common';

...

@Get(':index')
public getProducto(@Param('index') index): Producto {
    return this.productoService.getProducto(parseInt(index));
}
```

Resultado GET Individual

Changes

https://github.com/jdottori/nest-cfp-demo/commit/a28060fed1d29b901a04271f7fc2abc4baa97a70