



Clase 45. Programación Backend

***Introducción a frameworks de
desarrollo backend:
Parte I***



OBJETIVOS DE LA CLASE

- Conocer sobre frameworks para desarrollo backend con Node:
 - Adonis
 - Nest

CRONOGRAMA DEL CURSO

Clase 44



GraphQL

Clase 45



**Introducción a
frameworks de
desarrollo backend:
Parte I**

Clase 46



**Introducción a
frameworks de
desarrollo backend:
Parte II**

ADONIS

¿De qué se trata?



- AdonisJS es un framework orientado al desarrollo web, basado en Node.js.
- Si bien cuenta con un “template estándar” con un patrón MVC para iniciar a desarrollar, el framework ofrece una gran adaptabilidad. Mediante el uso de “templates” personalizados permite integrar nuevas soluciones como una configuración API REST.
- Está inspirado en un framework PHP llamado Laravel. Toma prestados los conceptos de inyección de dependencia y proveedores de servicios para escribir un código que sea comprobable en su núcleo.



¿De qué se trata?



- Adonis ya trae incluido:
 - Un CLI
 - Autenticaciones
 - Sistema MVC
 - Soporte de primera clase para tests
 - Un ORM con modelos, migraciones, factorys y seeds.
 - Soporte incluido para internacionalización i18n
 - Login social
 - Sistema de plantillas
 - Validaciones
 - Servidor de sockets y cliente



Algunas características



- Autenticación social a través de Facebook, Google, Github, etc.
- Proveedores de Correo: SMTP, Spark Post, Mailgun y Amazon SES.
- Funciones de seguridad con soporte para CORS, protección de ataques de malware.
- Capa robusta de middlewares para interactuar con las solicitudes HTTP entrantes.
- Plantillas basadas en Edge.
- Soporte para emitir y escuchar eventos en toda la aplicación.
- Soporte incorporado para Redis.



Algunas características



- Carga de archivos segura y directa.
- Poderosa herramienta de línea de comandos llamada Ace.
- Nucleo totalmente extensible.
- Soporte para la internacionalización, puede traducir fácilmente a varios idiomas.
- El motor de base de datos tiene soporte para Query Builder, Lucid ORM, Migrations, Factories y Seeds.
- Pruebas unitarias automatizadas.
- Comunidad solidaria y amigable.



Empezar a usarlo - instalación



- Comenzamos instalando globalmente Adonis CLI:

```
$ npm i --global @adonisjs/cli
```

- Luego, vamos a posicionarnos en la consola en la carpeta donde queramos crear nuestro proyecto con Adonis. Lo crearemos con el comando:

```
$ adonis new my-app-name
```

- Una vez creado nuestro proyecto, ingresamos a la carpeta del mismo e iniciamos el servidor de nuestra app:

```
$ cd my-app-name
```

```
$ adonis serve --dev
```



Empezar a usarlo - instalación



```
C:\Cursos\Coderhouse\CursoBackend\Clase45\Adonis
λ adonis new MiApp

AdonisJS

[1/6] Requirements matched [node & npm]
[2/6] Ensuring project directory is clean [MiApp]
[3/6] Cloned [adonisjs/adonis-fullstack-app]
[4/6] Dependencies installed
[5/6] Environment variables copied [.env]
[6/6] Key generated [adonis key:generate]

Successfully created project
Get started with the following commands

$ cd MiApp
$ adonis serve --dev

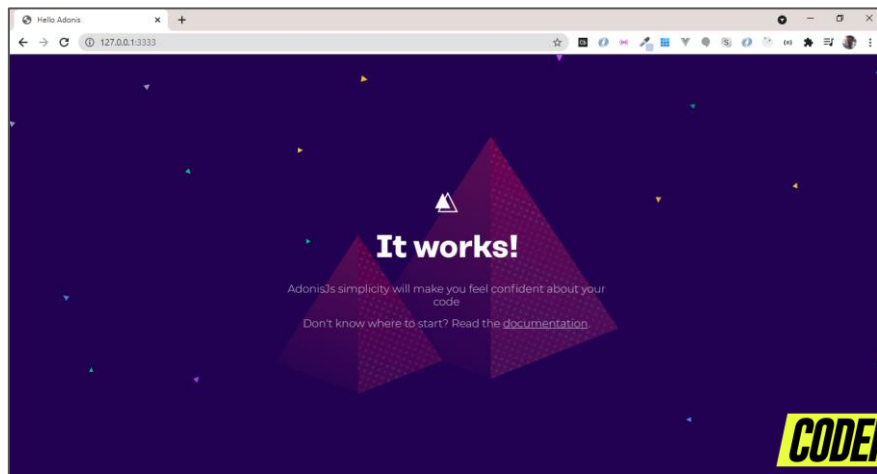
C:\Cursos\Coderhouse\CursoBackend\Clase45\Adonis
λ cd MiApp\

C:\Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp
λ adonis serve --dev

SERVER STARTED
> Watching files for changes...

info: serving app on http://127.0.0.1:3333
|
```

- Como vemos, en este ejemplo creamos un proyecto llamado “MiApp”.
- Al iniciar el servidor, ingresamos en el navegador a la url que nos indica en la consola (<http://127.0.0.1:3333>) y podremos ver la app.

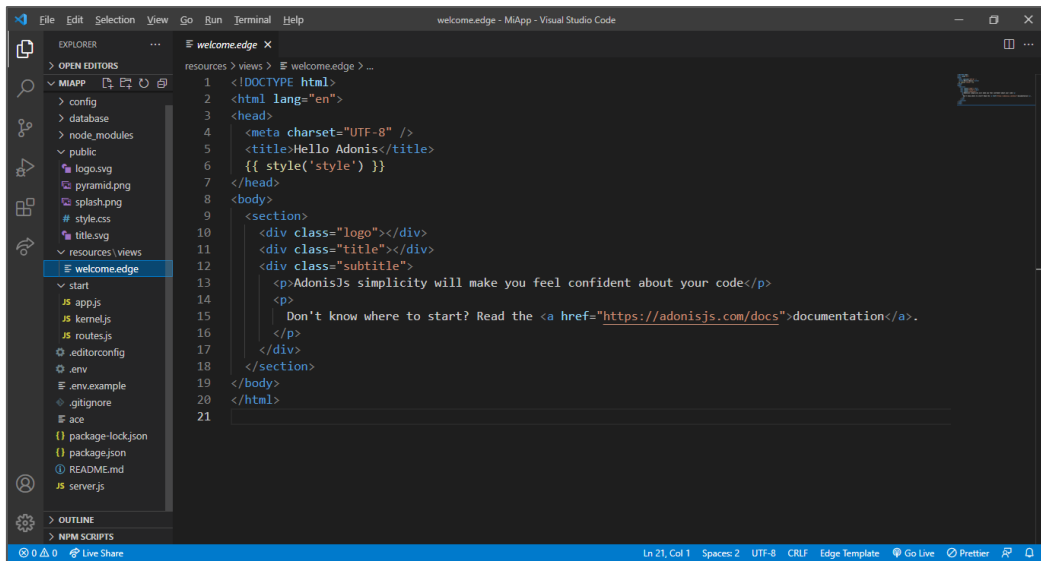




Empezar a usarlo



- En la carpeta *resources*, tenemos un archivo llamado **welcome.edge** que es el que se muestra en el navegador al iniciar el servidor, como vimos anteriormente. El código de este archivo es:



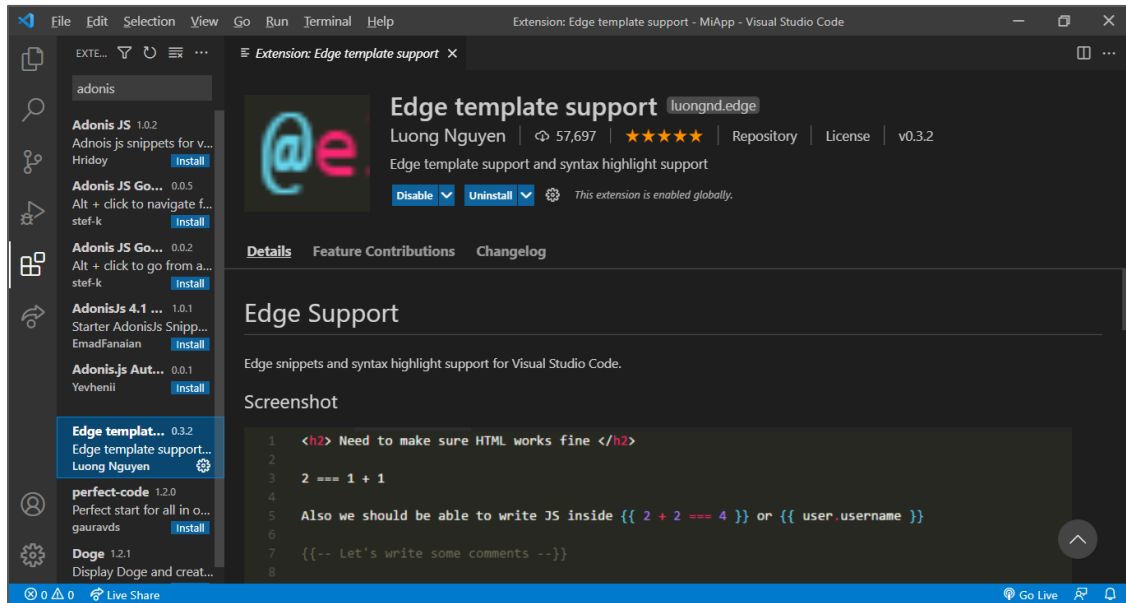
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8" />
5   <title>Hello Adonis</title>
6   {{ style('style') }}
7 </head>
8 <body>
9   <section>
10    <div class="logo"></div>
11    <div class="title"></div>
12    <div class="subtitle">
13      <p>AdonisJs simplicity will make you feel confident about your code</p>
14      <p>
15        Don't know where to start? Read the <a href="https://adonisjs.com/docs">documentation</a>.
16      </p>
17    </div>
18  </section>
19 </body>
20 </html>
21
```



Empezar a usarlo



- Instalamos el siguiente plugin en el Visual Studio Code para que el formato del motor de plantillas edge se represente correctamente en el editor.



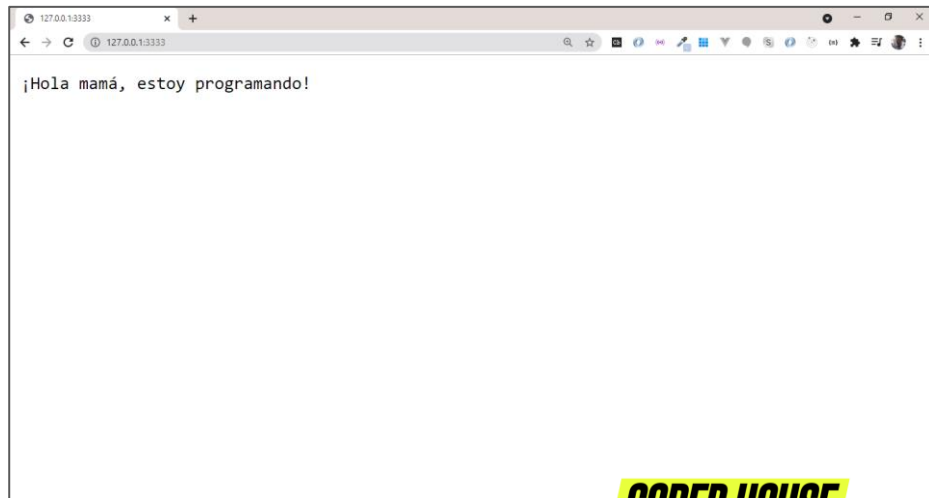


Usando Adonis - Rutas



- En la carpeta *start*, tenemos el archivo de ***routes.js*** en donde podemos definir nuestras rutas. Empezamos definiendo la ruta “/” por GET, que simplemente devuelva un texto. El mismo lo vemos en el navegador.

```
10 start > .js routes.js > ...
11
12 A complete guide on routing is available here.
13 http://adonisjs.com/docs/4.1/routing
14 */
15
16 /** @type {typeof import('@adonisjs/framework/src/Route/Manager')} */
17 const Route = use('Route')
18
19 //Route.on('/').render('welcome')
20
21 Route.get('/', () => '¡Hola mamá, estoy programando!')
22 Route.get('/portafolio', 'PortfolioController.index')
23 // ¡Tienes un controlador con todo un sistema para administrar un recurso (CRUD etc)?
24 Route.resource('users', 'UserController')
25
26 // Agregar middleware de autenticación
27 Route
28   .get('/botonRojo', 'PresidentController.Apocalipsis')
29   .middleware('auth')
30
31
32
```





Usando Adonis - Middleware



- Los middlewares se definen al encadenar el método *middleware*, y somos libres de especificar uno o más middleware pasando múltiples argumentos

```
10 |
11 | A complete guide on routing is available here.
12 | http://adonisjs.com/docs/4.1/routing
13 |
14 | */
15 |
16 | /** @type {typeof import('@adonisjs/framework/src/Route/Manager')} */
17 | const Route = use('Route')
18 |
19 | //Route.on('/').render('welcome')
20 |
21 | Route.get('/', () => '¡Hola mamá, estoy programando!')
22 | Route.get('/portafolio', 'PortfolioController.index')
23 | // ¿Tienes un controlador con todo un sistema para administrar un recurso (CRUD etc)?
24 | Route.resource('users', 'UserController')
25 |
26 | // Agregar middleware de autenticación
27 | Route
28 |   .get('/botonRojo', 'PresidentController.Apocalipsis')
29 |   .middleware('auth')
30 |
31 |
32 |
```



Usando Adonis - Modelos y migraciones



- Podemos crear Modelos, factories, seeds y migraciones de manera muy fácil.
- Vamos de nuevo a la consola, y ponemos el siguiente comando:

```
Cmder

C:\Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp
λ adonis make:model Cupcake --migration
✓ create app\Models\Cupcake.js
✓ create database\migrations\1624632673379_cupcake_schema.js

C:\Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp
λ |
```

- Este comando nos creará el modelo *Cupcake.js* y el *schema* en la carpeta *migrations*.



Usando Adonis - Modelo



- El modelo *Cupcake* lo podemos ver en `app/Models/Cupcake.js`.

```
1  'use strict'
2
3  /** @type {typeof import('@adonisjs/lucid/src/Lucid/Model')} */
4  const Model = use('Model')
5
6  class Cupcake extends Model {
7  }
8
9  module.exports = Cupcake
10
```




Usando Adonis - Modelos y migraciones



- Es esquema que se genera tiene el siguiente código:

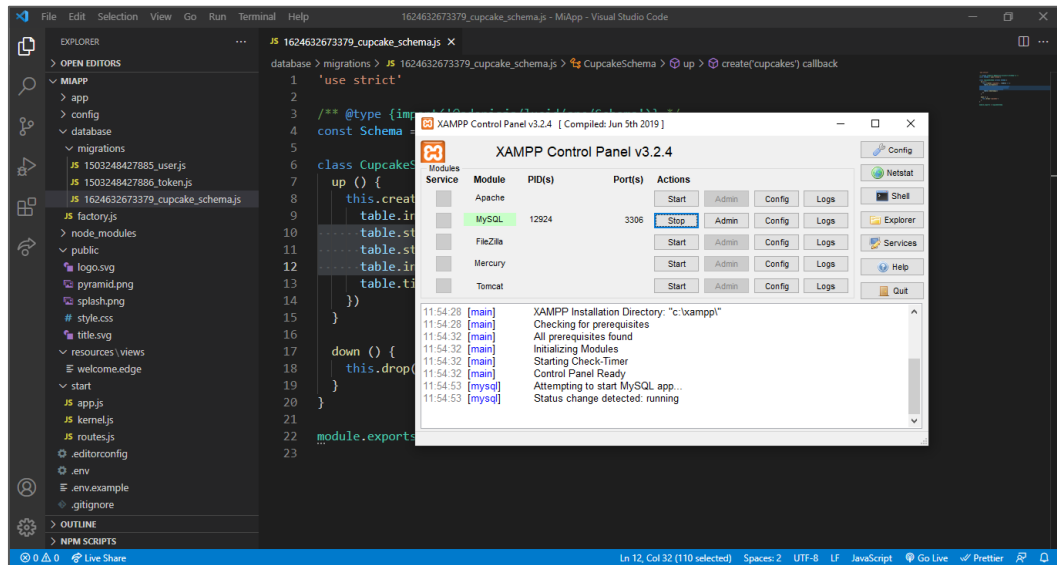
```
1  'use strict'
2
3  /** @type {import('@adonisjs/lucid/src/Schema')} */
4  const Schema = use('Schema')
5
6  class CupcakeSchema extends Schema {
7    up () {
8      this.create('cupcakes', (table) => {
9        table.increments()
10       table.string('name', 80).unique()
11       table.string('description', 150)
12       table.integer('price')
13       table.timestamps()
14     })
15   }
16
17   down () {
18     this.drop('cupcakes')
19   }
20 }
21
22 module.exports = CupcakeSchema
23
```



Usando Adonis - Base de datos



- Para poder correr las migraciones, primero debemos crear la base de datos. En este caso vamos a trabajar con MySQL.
- Para eso, iniciamos nuestro servidor de MySQL (por ejemplo, con XAMPP).





Usando Adonis - Base de datos



- Luego, configuramos el archivo `.env` con las siguientes variables de entorno.

The screenshot shows the Visual Studio Code interface with the `.env` file open. The Explorer sidebar on the left shows the project structure, including files like `factory.js`, `node_modules`, `public`, `logo.svg`, `pyramid.png`, `splash.png`, `style.css`, `title.svg`, `resources\views`, `welcome.edge`, `start`, `app.js`, `kernel.js`, `routes.js`, `.editorconfig`, `.env`, `.env.example`, `.gitignore`, `ace`, `package-lock.json`, `package.json`, `README.md`, and `server.js`. The `.env` file is selected and its content is displayed in the editor. The status bar at the bottom indicates 'Ln 7, Col 1 (19 selected)', 'Spaces: 4', 'UTF-8', 'LF', 'Plain Text', 'Go Live', 'Prettier', and 'Live Share'.

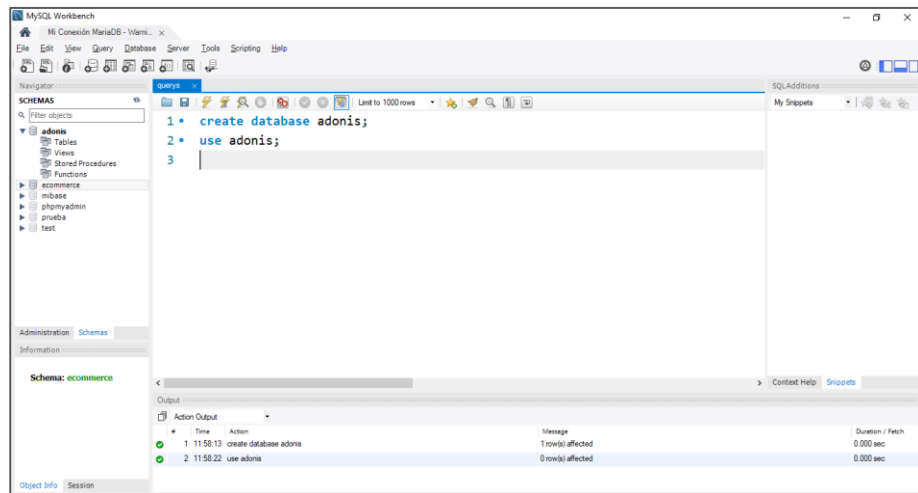
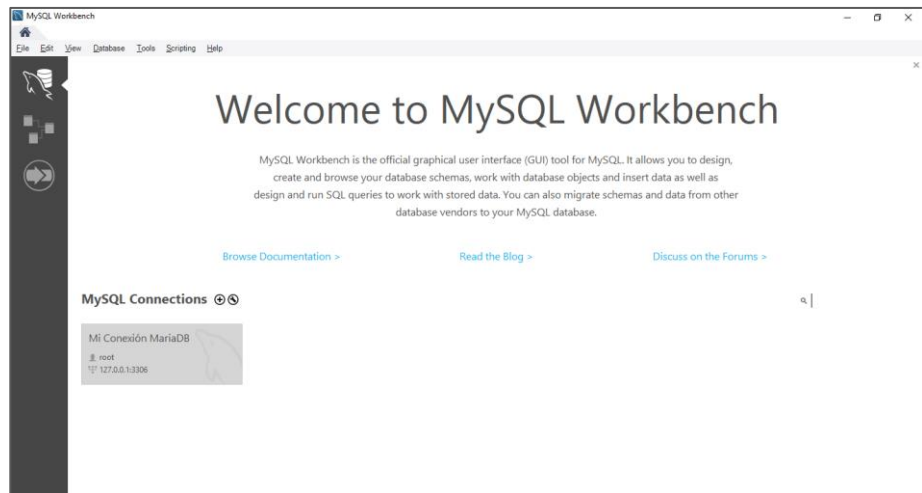
```
.env
1 HOST=127.0.0.1
2 PORT=3333
3 NODE_ENV=development
4 APP_URL=http://${HOST}:${PORT}
5 CACHE_VIEWS=false
6 APP_KEY=Nyenr0ZHM2KddoAd20L1YDixwBb92Pw9
7 DB_CONNECTION=mysql
8 DB_HOST=127.0.0.1
9 DB_PORT=3306
10 DB_USER=root
11 DB_PASSWORD=
12 DB_DATABASE=adonis
13 SESSION_DRIVER=cookie
14 HASH_DRIVER=bcrypt
```



Usando Adonis - Base de datos



- Ahora ya podemos ingresar al Workbench y crear nuestra base de datos.
- Creamos una base de datos llamada Adonis y nos posicionamos sobre ella.





Usando Adonis - Migraciones



- Vamos nuevamente a la consola para correr las migraciones. Primero debemos instalar *mysql* en nuestro proyecto.
- Luego corremos las migraciones con el comando: `adonis migration:run`
- Podemos ver el status de las mismas con el comando: `adonis migration:status`

```
C:\Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp
λ npm i mysql
+ mysql@2.18.1
added 5 packages from 11 contributors and audited 513 packages in 2.897s

10 packages are looking for funding
  run `npm fund` for details

C:\Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp
λ adonis migration:run
migrate: 1503248427885_user.js
migrate: 1503248427886_token.js
migrate: 1624632673379_cupcake_schema.js
Database migrated successfully in 319 ms

C:\Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp
λ adonis migration:status
```

| File name | Migrated | Batch |
|------------------------------|----------|-------|
| 1503248427885_user | Yes | 1 |
| 1503248427886_token | Yes | 1 |
| 1624632673379_cupcake_schema | Yes | 1 |

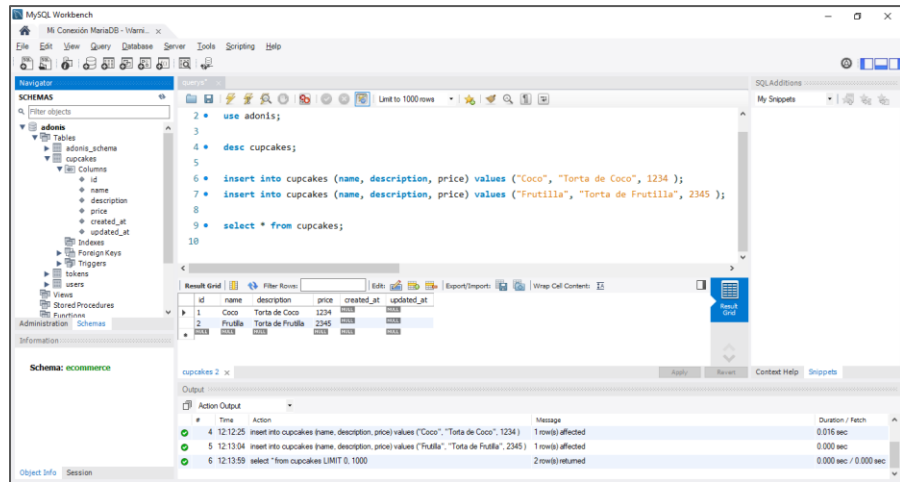
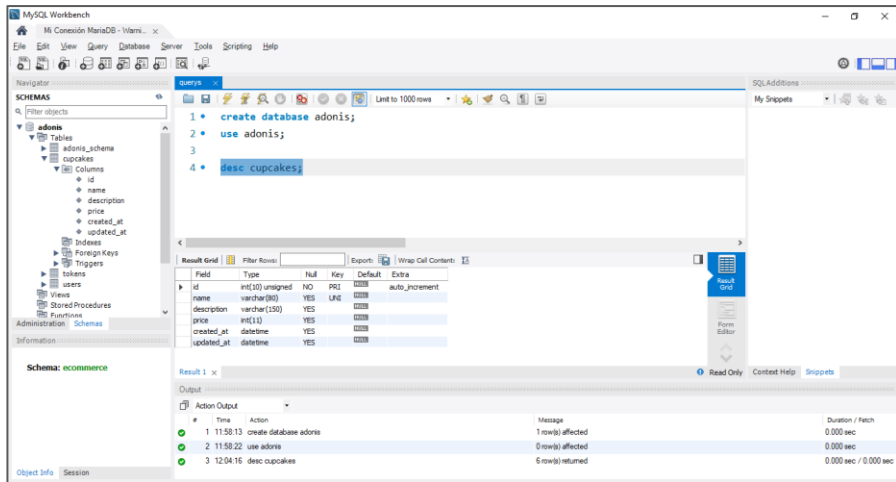
```
C:\Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp
λ
```



Usando Adonis - Modelo



- Vamos ahora de nuevo al Workbench y vamos a insertar dos registros en la tabla *cupcakes*.





Usando Adonis - Modelo



- Vamos ahora nuevamente al archivo de rutas, y agregamos el siguiente código para agregar una ruta para el modelo.

```
start > JS routes.js > ...
14 /
15
16 /** @type {typeof import('@adonisjs/framework/src/Route/Manager')} */
17 const Route = use('Route')
18
19 //Route.on('/').render('welcome')
20
21 Route.get('/', () => '¡Hola mamá, estoy programando!')
22 Route.get('/portafolio', 'PortfolioController.index')
23 // ¿Tienes un controlador con todo un sistema para administrar un recurso (CRUD etc)?
24 Route.resource('users', 'UserController')
25
26 // Agregar middleware de autenticación
27 Route
28   .get('/botonKojro', 'PresidentController.Apocalipsis')
29   .middleware('auth')
30
31 const Cupcake = use('App/Models/Cupcake')
32 Route
33   .get('cupcakes', async () => {
34     return await Cupcake.all()
35   })
36
```



Usando Adonis - Vista del modelo



- Vemos entonces en el navegador, un JSON con todos los registros de la tabla cupcake que tenemos en nuestra base de datos.

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:3333/cupcakes'. The main content area displays a JSON array of two cupcake records. The first record has an id of 1, name 'Coco', description 'Torta de Coco', and price 1234. The second record has an id of 2, name 'Frutilla', description 'Torta de Frutilla', and price 2345. Both records have null values for 'created_at' and 'updated_at'. The browser interface includes a 'View source' link and a settings icon in the top right corner.

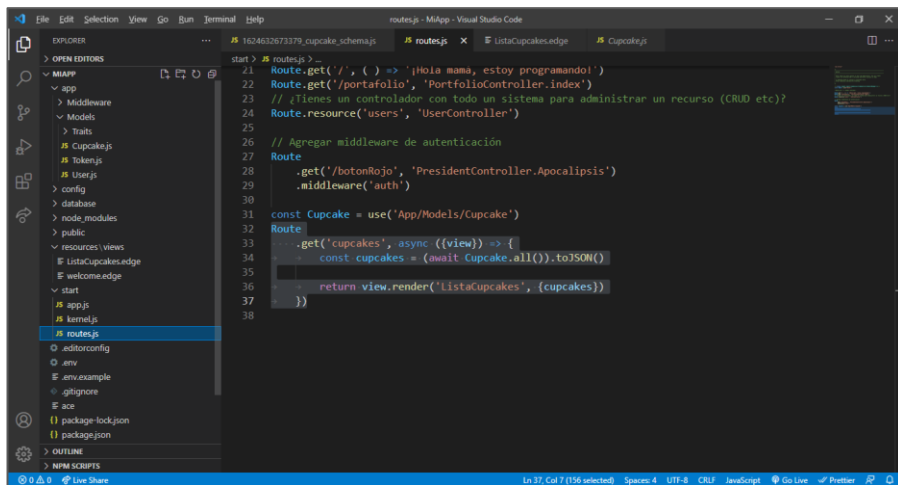
```
[
  - {
    id: 1,
    name: "Coco",
    description: "Torta de Coco",
    price: 1234,
    created_at: null,
    updated_at: null
  },
  - {
    id: 2,
    name: "Frutilla",
    description: "Torta de Frutilla",
    price: 2345,
    created_at: null,
    updated_at: null
  }
]
```




Usando Adonis - vista del modelo

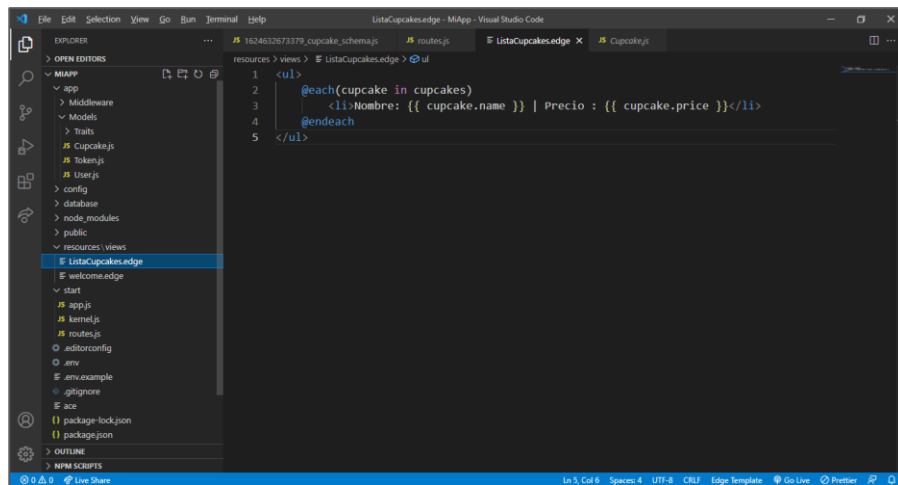


- Para verlo en una plantilla Edge de Adonis debemos llamarla en la ruta y crear el archivo edge en resources/views:



The screenshot shows the Visual Studio Code editor with the `routes.js` file open. The file contains the following code:

```
start > routes.js >
21 Route.get('/', () => '¡Hola mama, estoy programando!')
22 Route.get('/portfolio', 'PortfolioController.index')
23 // ¿Tienes un controlador con todo un sistema para administrar un recurso (CRUD etc)?
24 Route.resource('users', 'UserController')
25
26 // Agregar middleware de autenticación
27 Route
28   .get('/botonMojo', 'PresidentController.Apocalipsis')
29   .middleware('auth')
30
31 const Cupcake = use('App/Models/Cupcake')
32 Route
33   .get('cupcakes', async ({view}) => {
34     const cupcakes = (await Cupcake.all()).toJSON()
35     return view.render('ListaCupcakes', {cupcakes})
36   })
37
38
```



The screenshot shows the Visual Studio Code editor with the `ListaCupcakes.edge` file open. The file contains the following code:

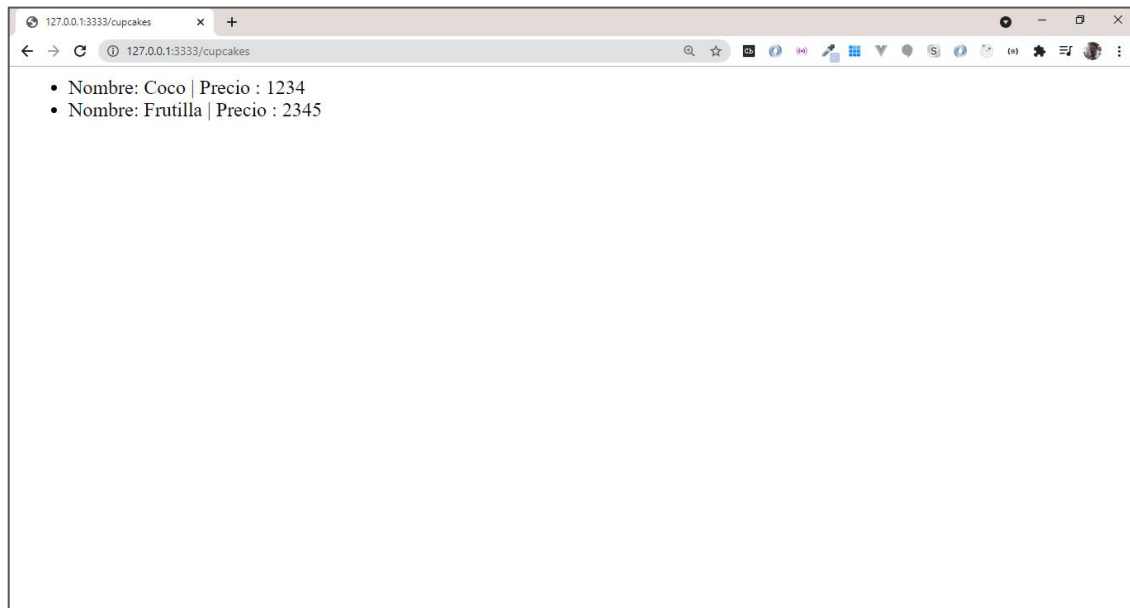
```
resources > views > ListaCupcakes.edge > ul
1 <ul>
2   @each(cupcake in cupcakes)
3     <li>Nombre: {{ cupcake.name }} | Precio : {{ cupcake.price }}</li>
4   @endeach
5 </ul>
```

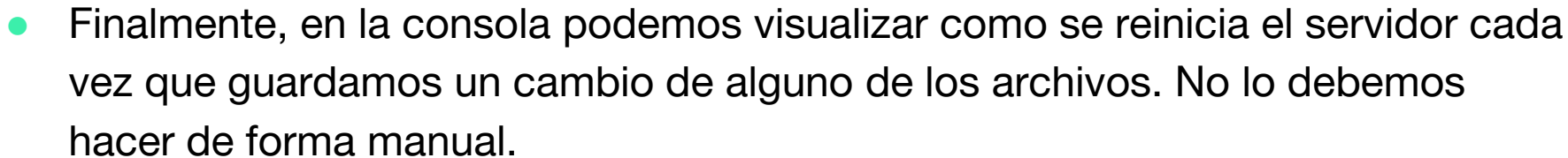


Usando Adonis - vista del modelo



- Con esto, veremos en el navegador una lista de los registros con sus datos, tal como lo configuramos en el archivo `.edge`.







Usando Adonis - vista del modelo



- Alternativamente, también podemos definir el controlador en su propio archivo (dentro de la carpeta “app/Controllers/Http”) y luego importarlo desde el archivo de rutas

```
// app/Controllers/Http/GetCupcakesController.js
'use strict'

const Cupcake = use('App/Models/Cupcake')

class GetCupcakesController {
  async index({ response, view }) {
    const result = await Cupcake.all()
    const cupcakes = result.toJSON()
    return view.render('ListaCupcakes', { cupcakes })
  }
}

module.exports = GetCupcakesController
```

```
// routes.js
Route
  .get('cupcakes', 'GetCupcakesController.index')
```



USANDO ADONIS

Tiempo: 15 minutos

USANDO ADONIS

Desafío
generico



Tiempo: 15 minutos

Crear una aplicación backend, utilizando el framework Adonis, que contenga dos rutas:

1) '/sin-controller' es una ruta gestionada sin controlador.

2) '/con-controller' es una ruta gestionada con controlador.

→ Ambas rutas recibirán una frase por query params y la representarán en una vista edge mediante dos listas ordenadas: una que muestre las palabras en orden y la otra que lo haga en orden invertido.

Para acceder a las query params, invocamos al método .get() del objeto request, disponible entre los argumentos del controlador, junto con view y responses, entre otros.

Ej. entradas:

`http://127.0.0.1:3333/sin-controller?palabras=hola mundo como están todos ustedes!`

`http://127.0.0.1:3333/con-controller?palabras=hola mundo cómo están todos ustedes'`

CODER HOUSE

USANDO ADONIS

Desafío
generico



Tiempo: 15 minutos

- La ruta '/sin-controller' debe realizar el proceso sin utilizar un controlador de ruta, mientras que '/con-controller' si.
- Representar también en la vista, si la respuesta viene de un proceso u otro.



>> Ejemplos de cada ruta.

127.0.0.1:3333/con-controller?palabras=hola%20mundo%20como%20están%20todos%20ustedes!

Con controller

Palabras

1. hola
2. mundo
3. como
4. están
5. todos
6. ustedes!

Palabras Invertidas

1. ustedes!
2. todos
3. están
4. como
5. mundo
6. hola

127.0.0.1:3333/sin-controller?palabras=hola%20mundo%20como%20están%20todos%20ustedes!

Sin controller

Palabras

1. hola
2. mundo
3. como
4. están
5. todos
6. ustedes!

Palabras Invertidas

1. ustedes!
2. todos
3. están
4. como
5. mundo
6. hola



BREAK

¡5/10 MINUTOS Y VOLVEMOS!

NEST



¿De qué se trata?



- **NestJS** es un framework de Node basado tanto en Node como en Express que nos permite construir un backend en TypeScript (o JS) con el patrón MVC.
- Una de sus principales fortalezas es que ofrece un poderoso marco de trabajo similar a otros frameworks MVC existentes en otros lenguajes.
- Cuenta con lenguaje tipado, separación por módulos, generación automática de entidades a partir de una base de datos, ORM, construcción de endpoints a través de decoradores, inyección de dependencias.
- NestJS está influenciado en gran medida por Angular, aprovechando muchos de sus conceptos como son los módulos, los controladores e inyección de dependencias.
- Cuenta con su propia CLI para generar y facilitarnos las tareas de creación de todos estos elementos.



Nest CLI



- Nest CLI es una herramienta de interfaz de línea de comandos que nos ayuda a inicializar, desarrollar y mantener nuestras aplicaciones Nest.
- Ayuda de múltiples formas, incluido el andamiaje del proyecto, el servicio en modo de desarrollo y la construcción y agrupación de la aplicación para la distribución de producción.
- Incorpora patrones arquitectónicos de mejores prácticas para fomentar aplicaciones bien estructuradas.

USANDO NEST



Empezar a usarlo - Instalación



- Comenzamos instalando Nest CLI de forma global con el comando:

```
$ npm install -g @nestjs/cli
```

- Para crear nuestro proyecto en Nest usamos el comando:

```
$ nest new my-nest-project
```

```
Cmder
C:\Cursos\Coderhouse\CursoBackend\Clase45\Nest
λ nest new cats-app
  We will scaffold your app in a few seconds..

CREATE cats-app/.eslintrc.js (631 bytes)
CREATE cats-app/.prettierrc (51 bytes)
CREATE cats-app/nest-cli.json (64 bytes)
CREATE cats-app/package.json (1970 bytes)
CREATE cats-app/README.md (3339 bytes)
CREATE cats-app/tsconfig.build.json (97 bytes)
CREATE cats-app/tsconfig.json (339 bytes)
CREATE cats-app/src/app.controller.spec.ts (617 bytes)
CREATE cats-app/src/app.controller.ts (274 bytes)
CREATE cats-app/src/app.module.ts (249 bytes)
CREATE cats-app/src/app.service.ts (142 bytes)
CREATE cats-app/src/main.ts (208 bytes)
CREATE cats-app/test/app.e2e-spec.ts (630 bytes)
CREATE cats-app/test/jest-e2e.json (183 bytes)

? Which package manager would you like to use? npm
✓ Installation in progress...

  Successfully created project cats-app
  Get started with the following commands:

$ cd cats-app
$ npm run start

  Thanks for installing Nest
  Please consider donating to our open collective
  to help us maintain this package.

  Donate: https://opencollective.com/nest

C:\Cursos\Coderhouse\CursoBackend\Clase45\Nest
λ
```



Empezar a usarlo - Iniciar servidor



- Para iniciar el servidor, ingresamos a la carpeta creada del proyecto y utilizamos el comando: `npm run start:dev`.

```
Cmdr
C:\Cursos\Coderhouse\CursoBackend\Clase45\Nest\dev/introduccion-nestjs/
λ cd cats-app\ npmjs.com/package/enestjs/cli
https://docs.nestjs.com/cli/overview
C:\Cursos\Coderhouse\CursoBackend\Clase45\Nest\cats-app (master)
λ npm run start:dev
npm i -g @nestjs/cli
```

```
Cmdr
[13:20:37] Starting compilation in watch mode... /introduccion-nestjs/
[13:20:42] Found 0 errors. Watching for file changes.

[Nest] 7432 - 25/06/2021 13:20:43 [NestFactory] Starting Nest application...
[Nest] 7432 - 25/06/2021 13:20:43 [InstanceLoader] AppModule dependencies initialized +32ms
[Nest] 7432 - 25/06/2021 13:20:43 [RoutesResolver] AppController {}: +5ms
[Nest] 7432 - 25/06/2021 13:20:43 [RouterExplorer] Mapped {, GET} route +3ms
[Nest] 7432 - 25/06/2021 13:20:43 [NestApplication] Nest application successfully started +2ms
```



Empezar a usarlo - Archivos



- Tendremos el archivo **main.ts**, que es el encargado de la entrada de la aplicación. En él se crea una instancia de la aplicación Nest y se establece el puerto en el que se va a ejecutar la aplicación.
- También nos encontramos con el archivo **app.service.ts**, que implementa el método `getHello()`.

```
src > TS main.ts > ...
1 import { NestFactory } from '@nestjs/core';
2 import { AppModule } from './app.module';
3
4 async function bootstrap() {
5   const app = await NestFactory.create(AppModule);
6   await app.listen(3000);
7 }
8 bootstrap();
9
```

```
src > TS app.service.ts > AppService
1 import { Injectable } from '@nestjs/common';
2
3 @Injectable()
4 export class AppService {
5   getHello(): string {
6     return 'Hola Mundo!';
7   }
8 }
9
```

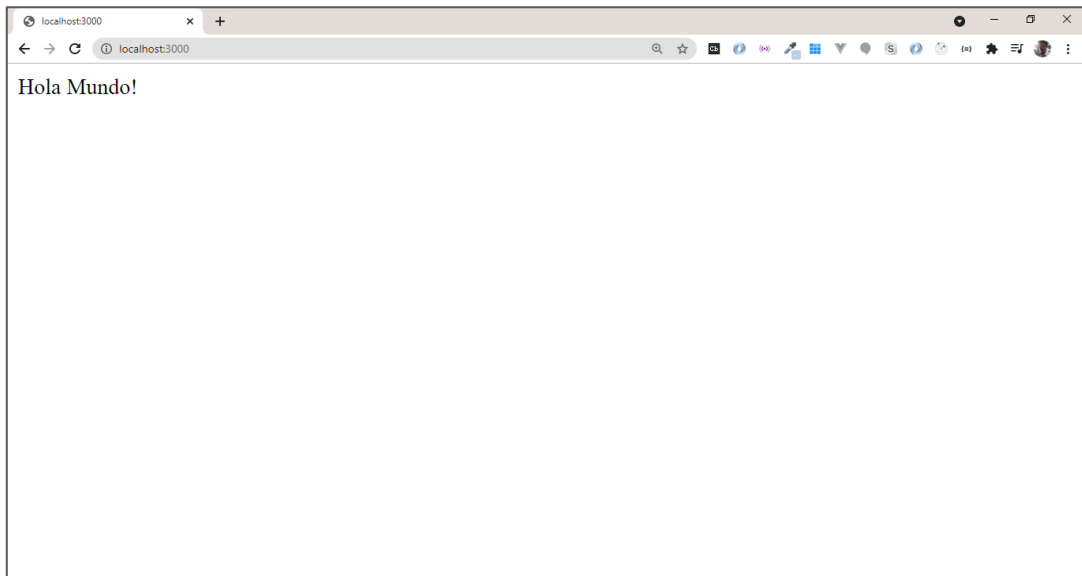
Para quienes la conozcan, verán que la estructura de carpetas es casi igual a la se crea en Angular



Empezar a usarlo - Iniciar servidor



- Entrando en el navegador a <http://localhost:3000> podemos ver la respuesta del método *getHello*.





Crear módulo, controlador y servicio



- Para crear un nuevo módulo usamos el comando: `nest generate module <name>`
- Para crear un nuevo controlador usamos: `nest generate controller <name>`
- Para crear un nuevo servicio usamos: `nest generate service <name>`.

```
C:\Cursos\Coderhouse\CursoBackend\Clase45\Nest\cats-app (master)
λ nest generate module cats
CREATE src/cats/cats.module.ts (81 bytes)
UPDATE src/app.module.ts (308 bytes)

C:\Cursos\Coderhouse\CursoBackend\Clase45\Nest\cats-app (master)
λ nest generate controller cats
CREATE src/cats/cats.controller.spec.ts (478 bytes)
CREATE src/cats/cats.controller.ts (97 bytes)
UPDATE src/cats/cats.module.ts (166 bytes)

C:\Cursos\Coderhouse\CursoBackend\Clase45\Nest\cats-app (master)
λ nest generate service cats
CREATE src/cats/cats.service.spec.ts (446 bytes)
CREATE src/cats/cats.service.ts (88 bytes)
UPDATE src/cats/cats.module.ts (240 bytes)

C:\Cursos\Coderhouse\CursoBackend\Clase45\Nest\cats-app (master)
λ |
```



Crear módulo, controlador y servicio



- Se crean los siguientes archivos dentro de una carpeta con el nombre del módulo:

```
src > cats > TS cats.module.ts > ...
1 import { Module } from '@nestjs/common';
2 import { CatsController } from './cats.controller';
3 import { CatsService } from './cats.service';
4
5 @Module({
6   controllers: [CatsController],
7   providers: [CatsService]
8 })
9 export class CatsModule {}
10
```

```
src > cats > TS cats.controller.ts > ...
1 import { Controller } from '@nestjs/common';
2
3 @Controller('cats')
4 export class CatsController {}
5
src > cats > TS cats.service.ts > ...
1 import { Injectable } from '@nestjs/common';
2
3 @Injectable()
4 export class CatsService {}
5
```



Crear módulo, controlador y servicio



- Se actualizó el archivo ***app.module.ts***, que es nuestro módulo raíz. Lo que se hizo en este proceso fue importar el módulo, controlador y servicio que hemos creado al módulo general de la aplicación:

```
src > TS app.module.ts > ...
1  import { Module } from '@nestjs/common';
2  import { ApplicationController } from '../app.controller';
3  import { AppService } from '../app.service';
4  import { CatsModule } from './cats/cats.module';
5
6  @Module({
7    imports: [CatsModule],
8    controllers: [AppController],
9    providers: [AppService],
10 })
11 export class AppModule {}
12
```



Interface y DTO



- Antes de empezar a crear endpoints y añadir lógica, creamos dos elementos que vamos a necesitar. Uno de ellos es una **Interface** de cat y el otro es el DTO para la creación del objeto tipo Cat. Los creamos con los comandos que vemos.

```
C:\Cursos\Coderhouse\CursoBackend\Clase45\Nest\cats-app (master)
λ nest generate interface interfaces/cat
CREATE src/interfaces/cat.interface.ts (24 bytes)
C:\Cursos\Coderhouse\CursoBackend\Clase45\Nest\cats-app (master)
λ
```

```
C:\Cursos\Coderhouse\CursoBackend\Clase45\Nest\cats-app (master)
λ mkdir src\dto && touch src\dto\create-cat.dto.ts
C:\Cursos\Coderhouse\CursoBackend\Clase45\Nest\cats-app (master)
λ
```



Interface y DTO



- Se crean entonces los archivo de interface y DOT correspondientes a los comandos:

```
src > interfaces > TS catInterface.ts U X
1 export interface Cat {
2   readonly name: string;
3
4   readonly age: number;
5
6   readonly breed: string;
7 }
8
```

```
src > dto > TS create-cat.dto.ts U X
1 export class CreateCatDto {
2   readonly name: string;
3
4   readonly age: number;
5
6   readonly breed: string;
7 }
```



Implementación de un servicio



```
1 import { Injectable } from '@nestjs/common';
2
3 import { Cat } from '../interfaces/cat.interface';
4
5 @Injectable()
6 export class CatsService {
7   private readonly cats: Cat[] = [];
8   create(cat: Cat) {
9     this.cats.push(cat);
10  }
11
12   findAll(): Cat[] {
13     return this.cats;
14   }
15 }
16
```

Vamos ahora a implementar dos métodos en nuestro servicio. Para eso, importamos nuestra Interface de Cat en el servicio y definimos un array de tipo Cat.

1. Definimos el método `create` que recibe un parámetro de tipo Cat y se ocupa de almacenar en el array los gatos creados.
2. Luego, definimos el método `findAll` que nos devuelve el array con la información de los gatos



Implementación de un controlador



- Ahora vamos a la parte del controlador. Importamos en el mismo la Interface *Cat*, el DTO *CreateCatDto* y el servicio *CatService*.
- Agregamos una instancia del servicio en el constructor para poder utilizarlo en el controlador.

```
src > cats > TS cats.controller.ts > CatsController
1 import { Body, Controller, Get, Post } from '@nestjs/common';
2 import { CatsService } from '../cats.service';
3 import { CreateCatDto } from '../dto/create-cat.dto';
4 import { Cat } from '../interfaces/cat.interface';
5
6 @Controller('cats')
7 export class CatsController {
8   constructor(private readonly catsService: CatsService) {}
9
10  @Post()
11  async create(@Body() createCatDto: CreateCatDto) {
12    | this.catsService.create(createCatDto);
13  }
14
15  @Get()
16  async findAll(): Promise<Cat[]> {
17    | return this.catsService.findAll();
18  }
19 }
20
```




Implementación de un controlador



```
1 import { Body, Controller, Get, Post } from '@nestjs/common';
2 import { CatsService } from '../cats.service';
3 import { CreateCatDto } from '../dto/create-cat.dto';
4 import { Cat } from '../interfaces/cat.interface';
5
6 @Controller('cats')
7 export class CatsController {
8   constructor(private readonly catsService: CatsService) {}
9
10  @Post()
11  async create(@Body() createCatDto: CreateCatDto) {
12    this.catsService.create(createCatDto);
13  }
14
15  @Get()
16  async findAll(): Promise<Cat[]> {
17    return this.catsService.findAll();
18  }
19 }
20
```

6. Definimos ahora nuestros métodos de crear y de buscar gatos.

→ En el método `create`, indicamos a través de la etiqueta decoradora el tipo de acción que realiza (POST). Por parámetros le indicamos que va a recibir un objeto de tipo `@Body`, que a su vez es el que vamos a enviar al servicio para crear el objeto tipo `Cat`.

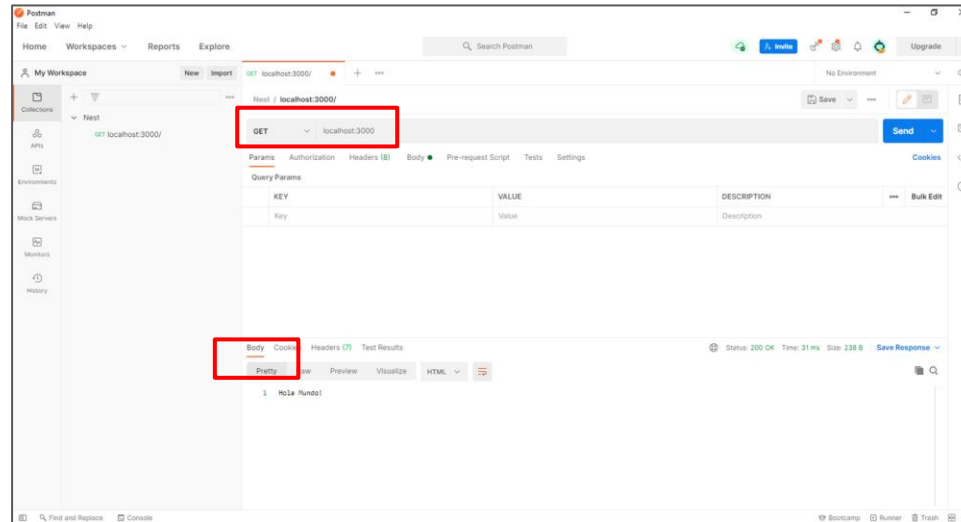
→ En el método `findAll`, le indicamos que vamos a devolver una promesa de array de tipo `Cat`. En este caso, solo debemos hacer un *return* de la llamada al servicio del método `findAll`.



Consumir la API con Postman



- Levantamos nuestra app con el comando: `npm run start`.
- Vamos a Postman e ingresamos la petición por GET de <http://localhost:3000>.
- Vemos que obtenemos el “Hola Mundo” que configuramos en el `app.service.ts`.

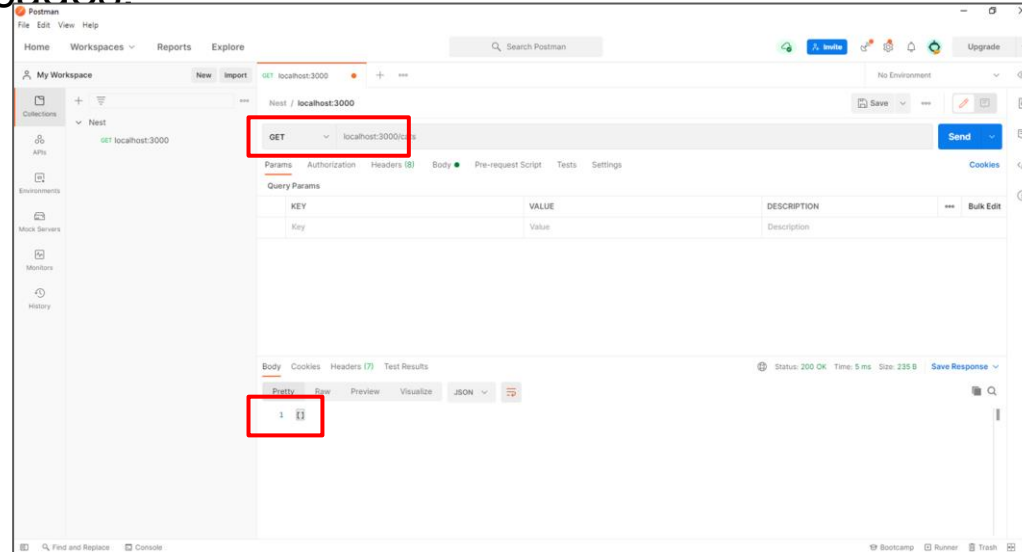




Consumir la API con Postman



- Probamos ahora la ruta de “/cats” por GET con la url <http://localhost:3000/cats>.
- Vemos que obtenemos un array vacío ya que todavía no tenemos información de gatos creados.

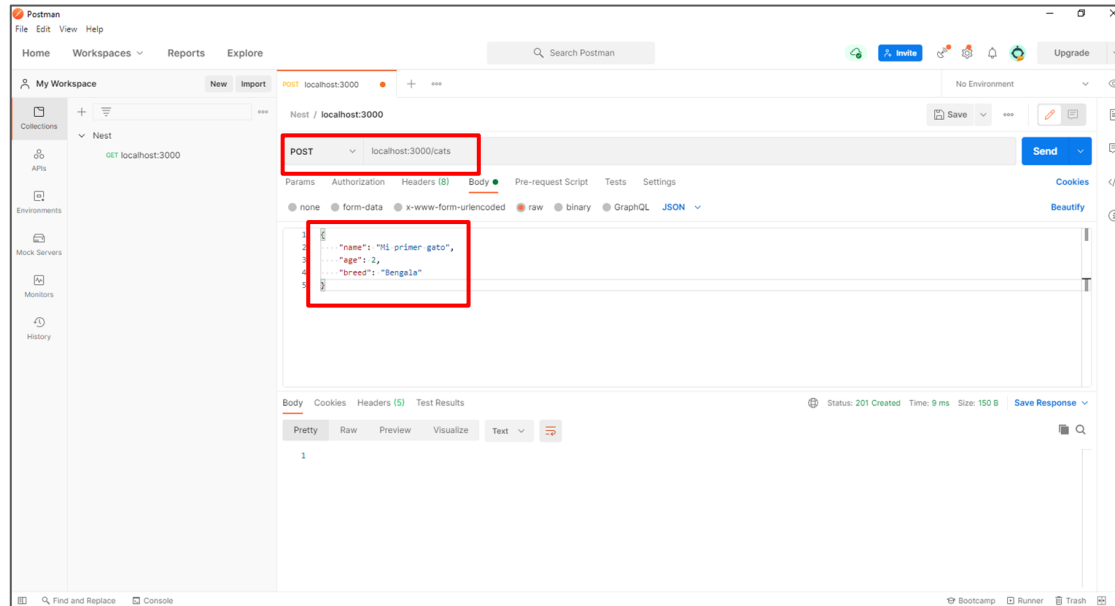




Consumir la API con Postman



- Creamos un gato nuevo con la ruta de “/cats” por POST con la url <http://localhost:3000/cats>.
- Los atributos de gato son: *name*, *age* y *breed*. Se los pasamos como JSON.

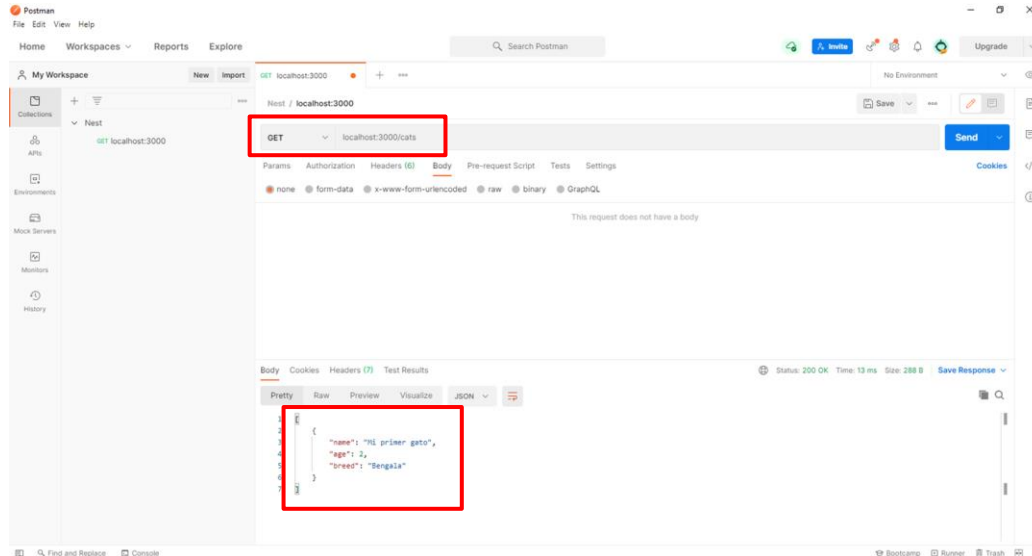




Consumir la API con Postman



- Nuevamente probamos la ruta de “/cats” por GET con la url <http://localhost:3000/cats>.
- Vemos que ahora nos trae la información del gato que creamos (se ejecuta el método findAll que creamos antes).



INCORPOREMOS SWAGGER!



Instalación de Swagger



- Instalamos Swagger con el comando:

```
npm install --save @nestjs/swagger swagger-ui-express
```
- De esta forma se instala el módulo de nestjs/swagger y swagger-ui-express para poder empezar a usar esta librería.

```
C:\Cursos\Coderhouse\CursoBackend\Clase45\Nest\cats-app (master)
λ npm install --save @nestjs/swagger swagger-ui-express
npm WARN @nestjs/mapped-types@0.4.1 requires a peer of class-transformer@^0.2.0 || ^0.3.0 || ^0.4.0 but none is installed. You must ins
tall peer dependencies yourself.
npm WARN @nestjs/mapped-types@0.4.1 requires a peer of class-validator@^0.11.1 || ^0.12.0 || ^0.13.0 but none is installed. You must in
stall peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"
os":"win32","arch":"x64"})

+ swagger-ui-express@4.1.6
+ @nestjs/swagger@4.8.1
added 4 packages from 9 contributors and audited 877 packages in 15.239s

78 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Cursos\Coderhouse\CursoBackend\Clase45\Nest\cats-app (master)
λ |
```



Implementación de Swagger



- Vamos ahora al archivo *main.ts* y especificamos la configuración de Swagger (título, descripción, versión...).
- Finalmente, se inyecta esta API Swagger en la aplicación y le damos una ruta de acceso, en este caso “api”.

```
1 import { NestFactory } from '@nestjs/core';
2 import { AppModule } from './app.module';
3 import { SwaggerModule, DocumentBuilder } from '@nestjs/swagger';
4
5 async function bootstrap() {
6   const app = await NestFactory.create(AppModule);
7
8   const options = new DocumentBuilder()
9     .setTitle('Cats example')
10    .setDescription('The cats API description')
11    .setVersion('1.0')
12    .addTag('cats')
13    .build();
14
15   const document = SwaggerModule.createDocument(app, options);
16
17   SwaggerModule.setup('api', app, document);
18
19   await app.listen(3000);
20 }
21 bootstrap();
22
```




Implementación de Swagger



- Por último, agregamos una etiqueta decoradora básica `@ApiProperty` en el DTO **CreateCatDto** para que se muestre en Swagger sus propiedades bien definidas.

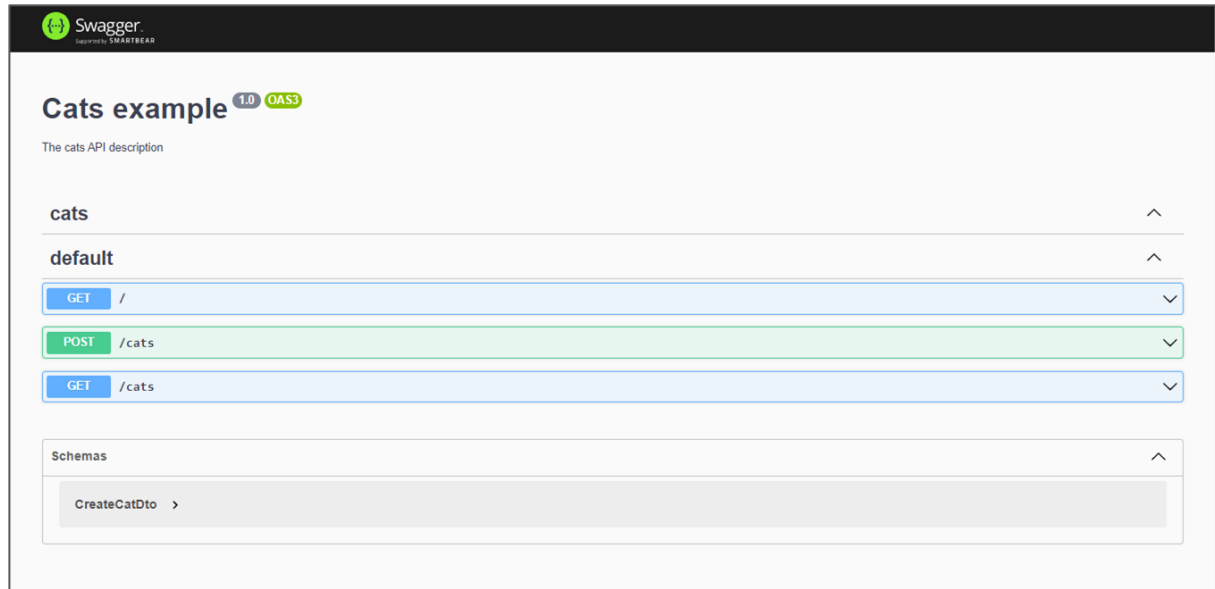
```
src > dto > TS create-cat.dto.ts > CreateCatDto > breed
1  import { ApiProperty } from '@nestjs/swagger';
2
3  export class CreateCatDto {
4    @ApiProperty()
5    readonly name: string;
6
7    @ApiProperty()
8    readonly age: number;
9
10   @ApiProperty()
11   readonly breed: string;
12 }
```



Implementación de Swagger



- Ahora entonces iniciamos nuevamente el servidor, e ingresamos desde el navegador a la ruta “/api” con la URL <http://localhost:3000/api>.

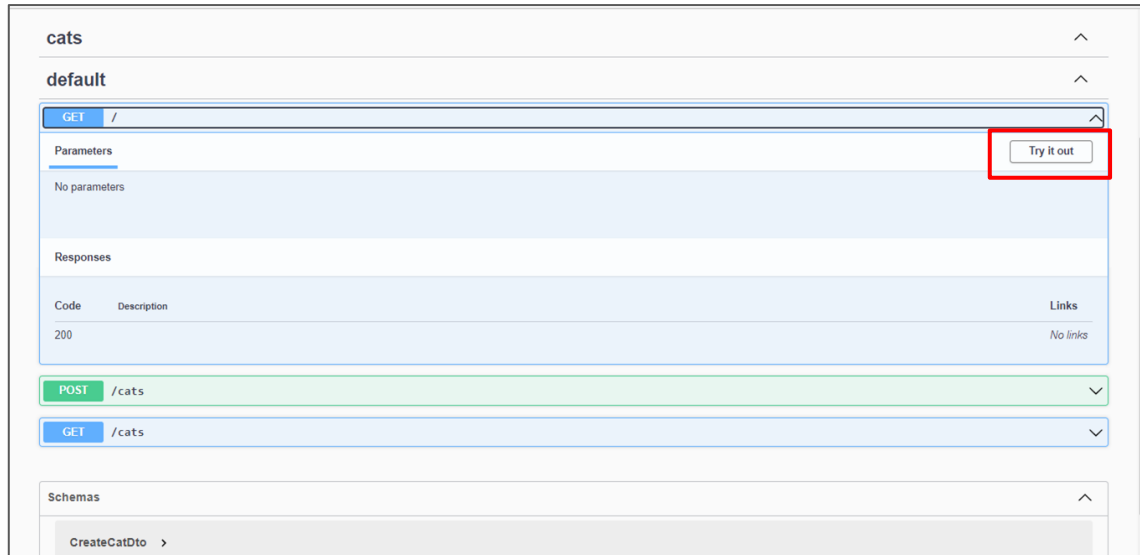




Implementación de Swagger



- Si vamos al primer endpoint, que es la ruta home por GET (“/”), vemos lo siguiente.
- Para empezar a ejecutar, clickeamos en el botón “Try it out”.

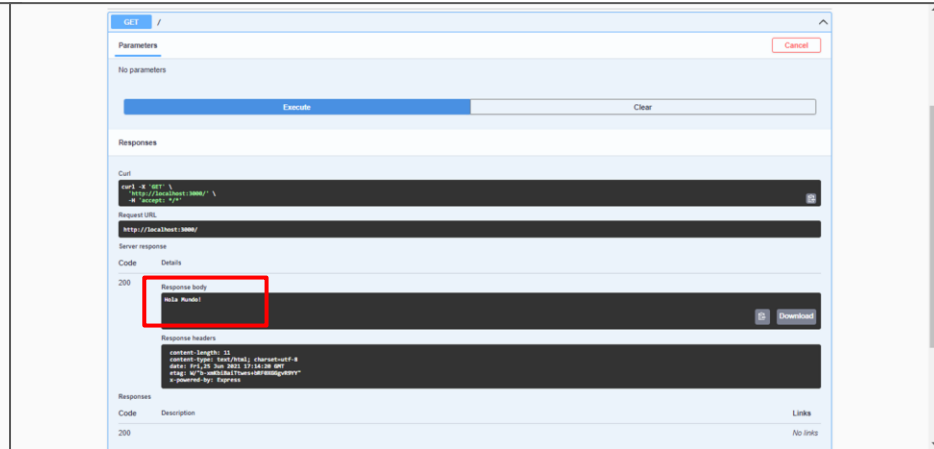
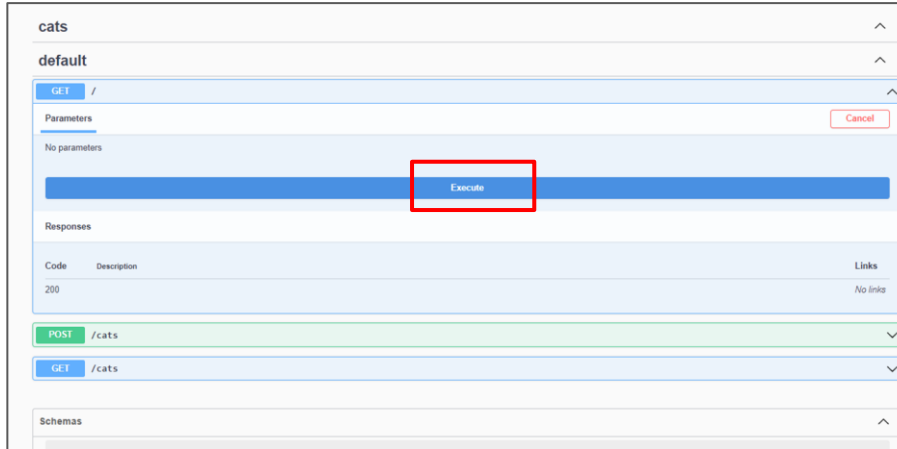




Implementación de Swagger



- Clickeando en el botón “Execute” se ejecuta el endpoint y obtenemos la respuesta en “Response body” que en este caso era “Hola Mundo”.

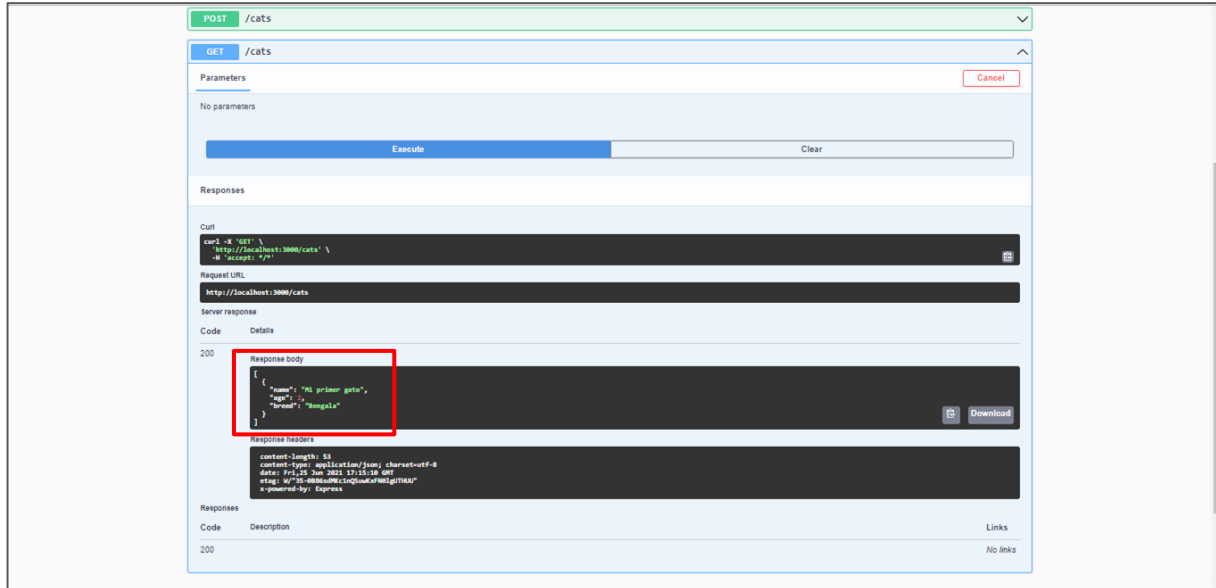




Implementación de Swagger



- Si ahora ejecutamos el endpoint por GET de “/cats” obtenemos la información del gato que agregamos antes por Postman.

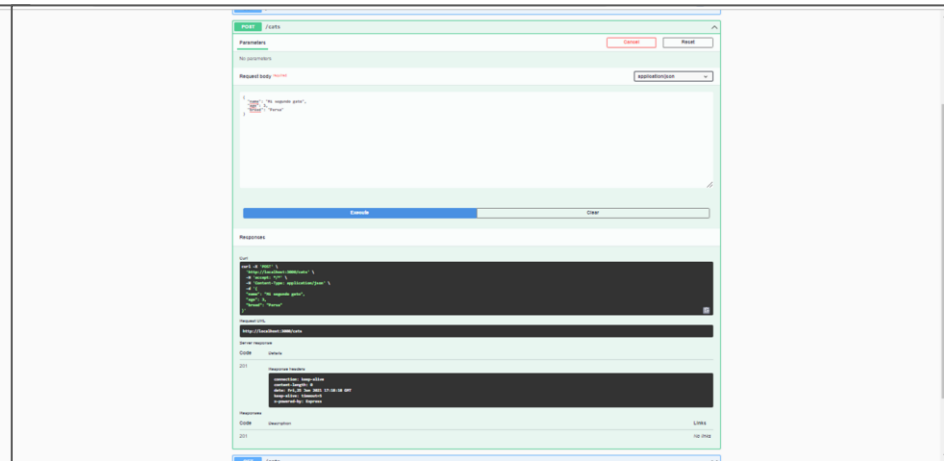
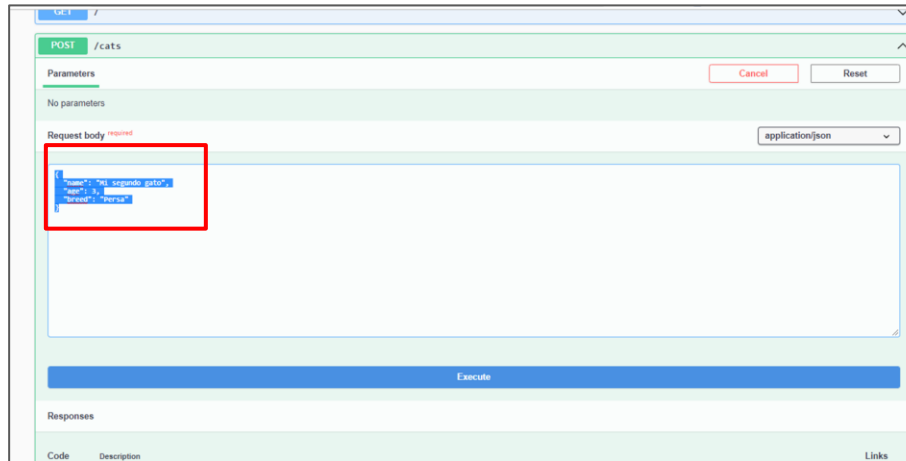




Implementación de Swagger



- Podemos agregar otro gato con el endpoint por POST de “/cats”. A este debemos pasarle el JSON con los datos y lo ejecutamos.





Implementación de Swagger



- Finalmente, si volvemos a ejecutar el endpoint “/cats” por GET, obtendremos la información de los dos gatos que tenemos creados.

The screenshot displays the Swagger UI for the GET /cats endpoint. The interface includes a 'Parameters' section with a 'Cancel' button, an 'Execute' button, and a 'Clear' button. The 'Responses' section shows the response for the 200 status code, which is a JSON array of two cat objects. The response body is highlighted with a red box.

```
curl -X 'GET' \
  'http://localhost:3000/cats' \
  -H 'accept: */*'

Request URL
http://localhost:3000/cats

Server response
Code    Details
200

Response body
[
  {
    "name": "El primer gato",
    "age": 1,
    "breed": "Bengala"
  },
  {
    "name": "El segundo gato",
    "age": 3,
    "breed": "Persa"
  }
]

Response headers
connection: keep-alive
content-length: 184
content-type: application/json; charset=utf-8
date: Fri, 25 Jun 2021 17:19:33 GMT
etag: W/"68-0001lmTqMcKj3CS+NeFF6fas"
keep-alive: timeout=5
x-powered-by: Express
```



USANDO NEST

Tiempo: 15 minutos

USANDO NEST

Desafío
generico



Tiempo: 15 minutos

Realizar una aplicación backend que utilice el framework Nest, donde dispondremos de un recurso llamado autos, que me permita por ruta get listar los autos almacenados.

- Disponer de una ruta post para agregar un auto con su marca, modelo y año.
- Incorporar varios autos a través de postman y verificar que ellos estén cargados. La persistencia se realizará en memoria.
- Incorporar Swagger y realizar estas mismas acciones a través de dicha interfaz.

¿PREGUNTAS?





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Adonis
 - Nest
- 



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN