CODER HOUSE

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

CODER HOUSE

CRONOGRAMA DEL CURSO

Clase 44	Clase 45	Clase 46
GraphQL	**Introducción a frameworks de desarrollo backend: Parte I**	**Introducción a frameworks de desarrollo backend: Parte II**

CODER HOUSE

ADONIS

CODER HOUSE

¿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 incluído:
  - 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
```

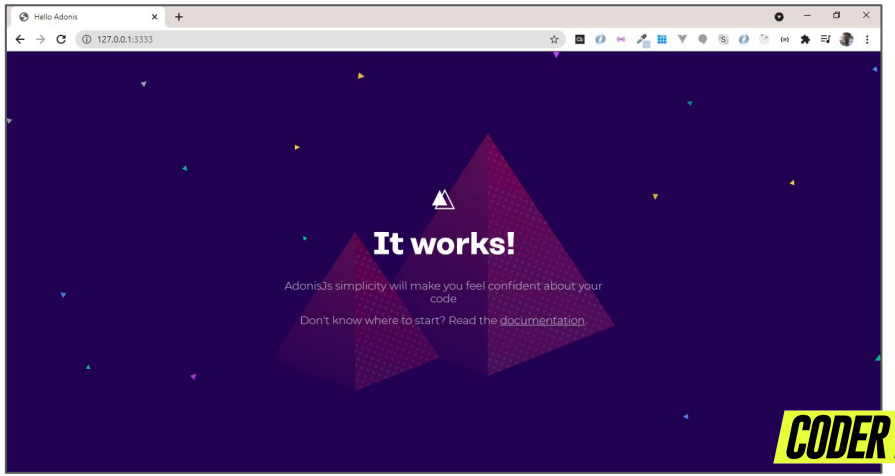
```
$ adonis serve --dev
```

CODER HOUSE

Empezar a usarlo - instalación

```
C:\Cursos\Coderhouse\CursoBackend\Clase45\Adonis
λ adonis new MiApp
   _       _                 _
  /_\   __| |___  _ _  (_)___   | |___
 / _ \ / _` / _ \| ' \ | (_-<   | (_-<
/_/ \_\\__,_\___/|_||_||_/__/   |_/__/

  [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
```

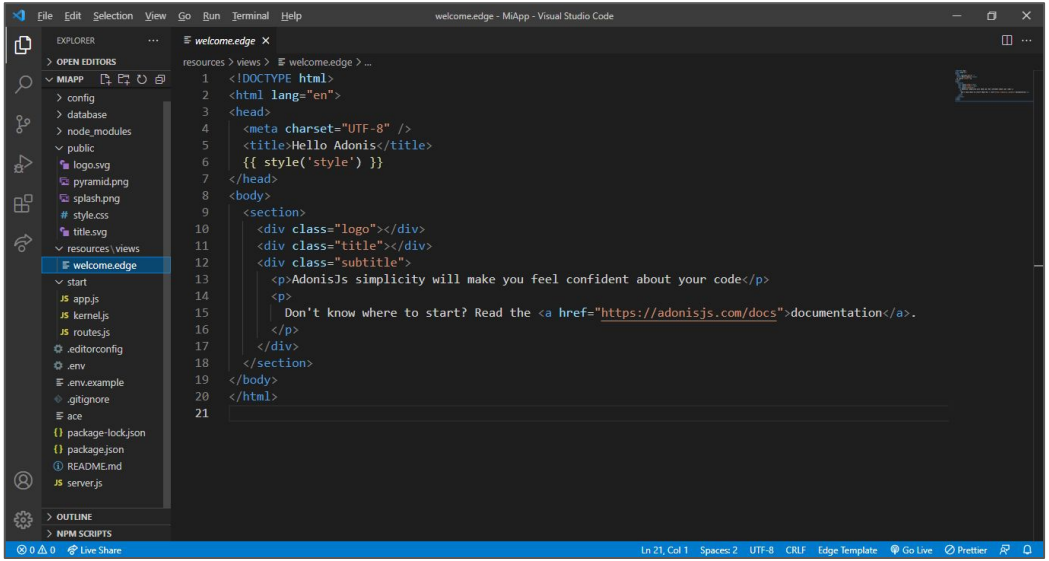
- Como vemos, en este ejemplo creamos un proyecto llamado “MiApp”.
- Al iniciar el servidor, ingresamos en el navegador a la url que nos indica en la consola (http://127.0.0.1:3333) y podremos ver la app.



CODER HOUSE

Empezar a usarlo

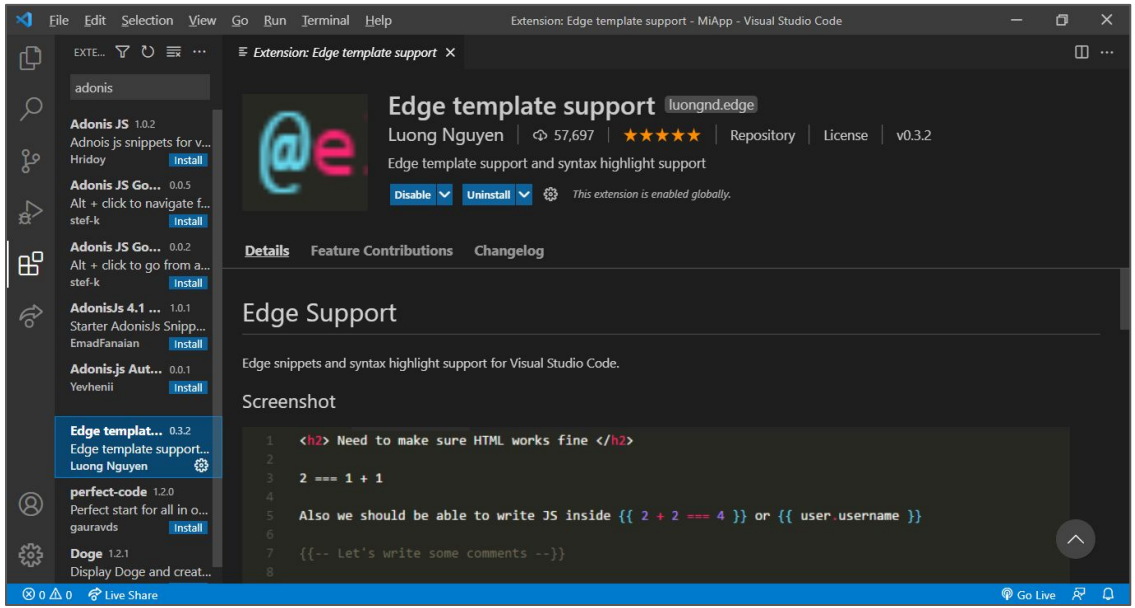
- En la carpeta *resources*, tenemos un archivo llamado ***welcome.edge*** que es el que se muestra en el navegador al iniciar el servidor, como vimos anteriormente. El código de este archivo es:



CODER HOUSE

Empezar a usarlo

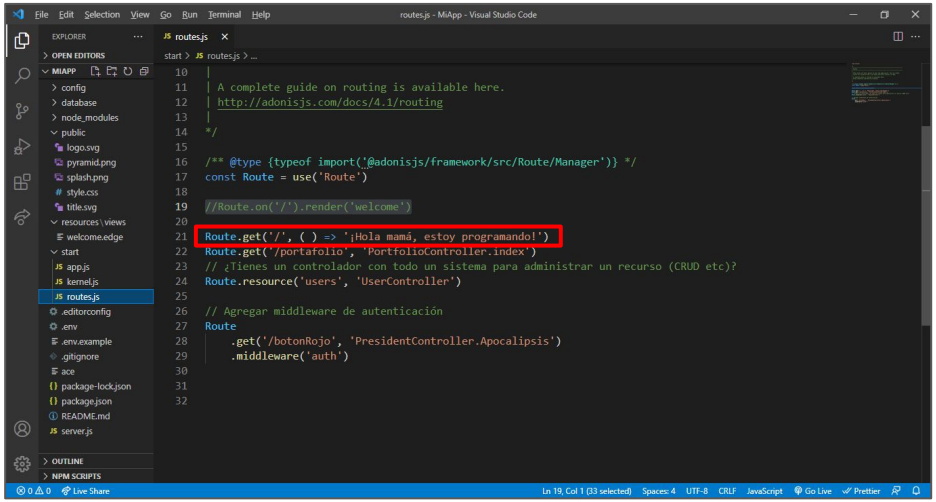
- Instalamos el siguiente plugin en el Visual Studio Code para que que el formato del motor de plantillas edge se represente correctamente en el editor.

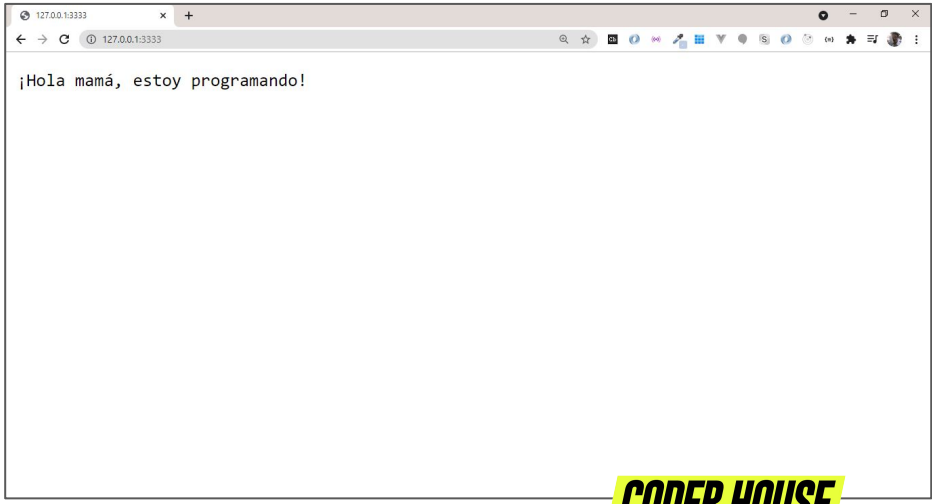


CODER HOUSE

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.

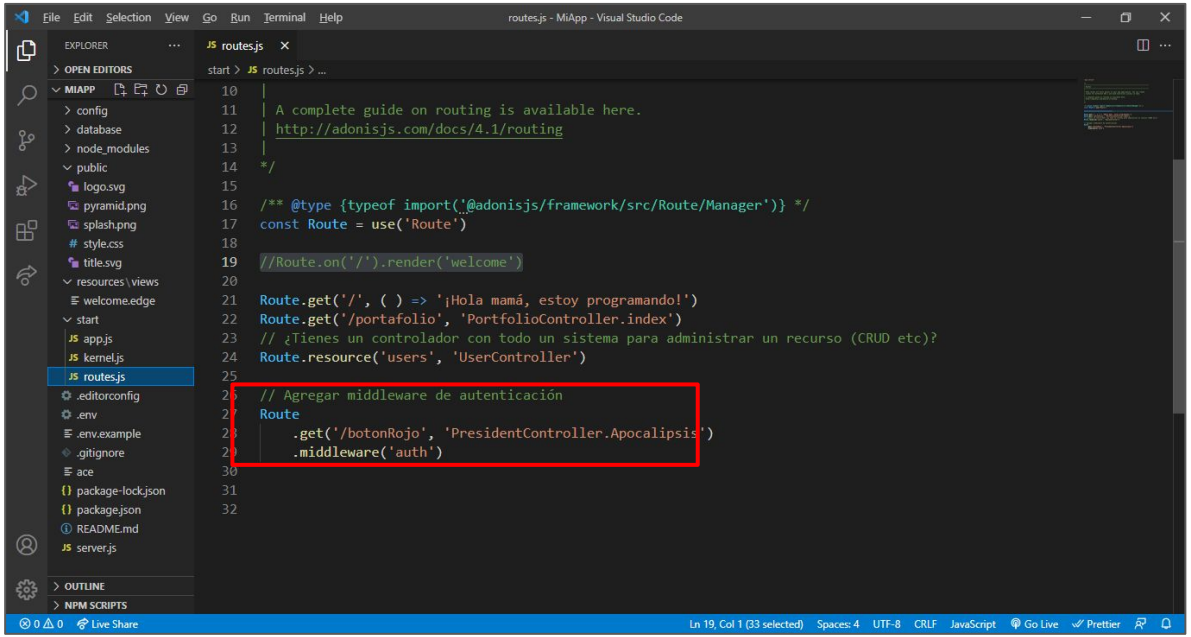




CODER HOUSE

Usando Adonis - Middleware

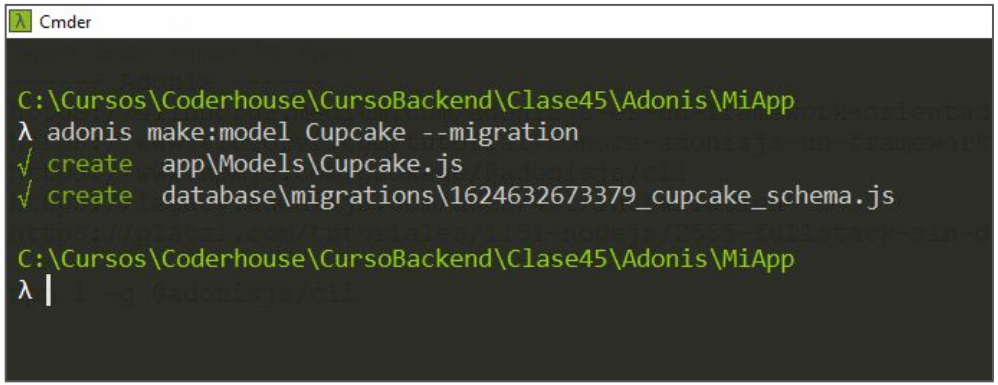
- Los middlewares se definen al encadenar el método ***middleware***, y somos libres de especificar uno o más middleware pasando múltiples argumentos.



CODER HOUSE

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:

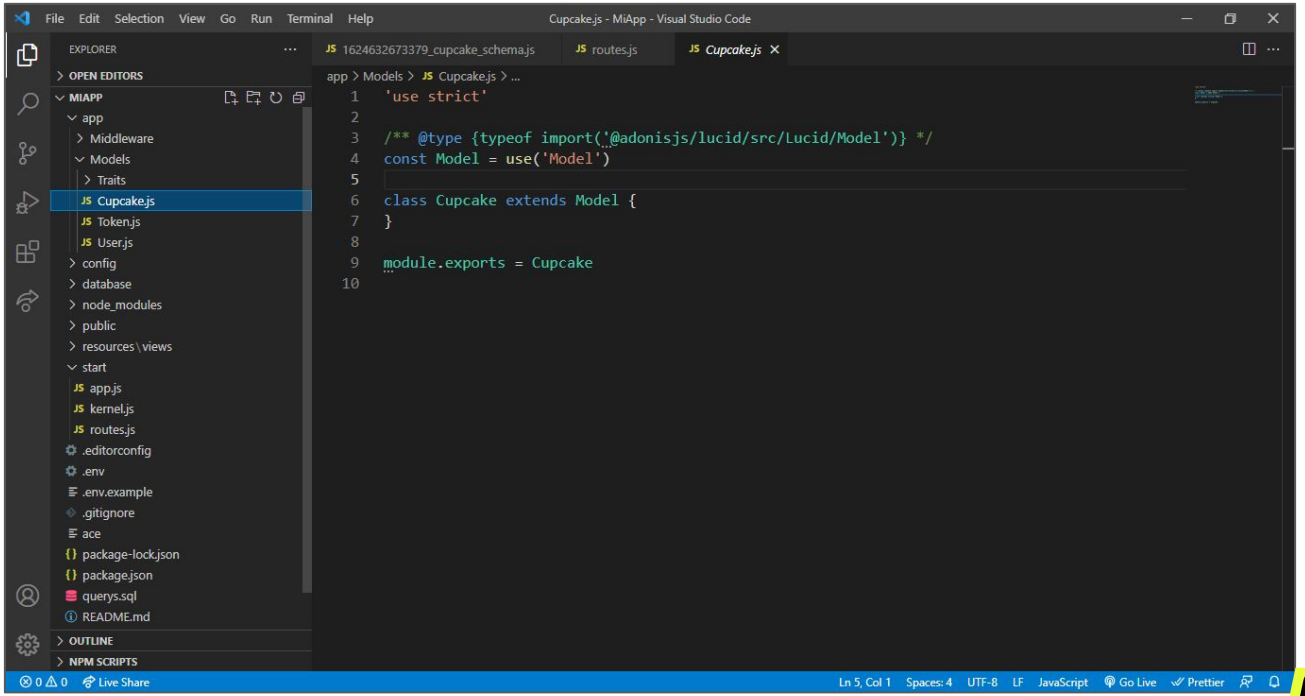


- Este comando nos creará el modelo *Cupcake.js* y el *schema* en la carpeta *migrations*.

CODER HOUSE

Usando Adonis - Modelo

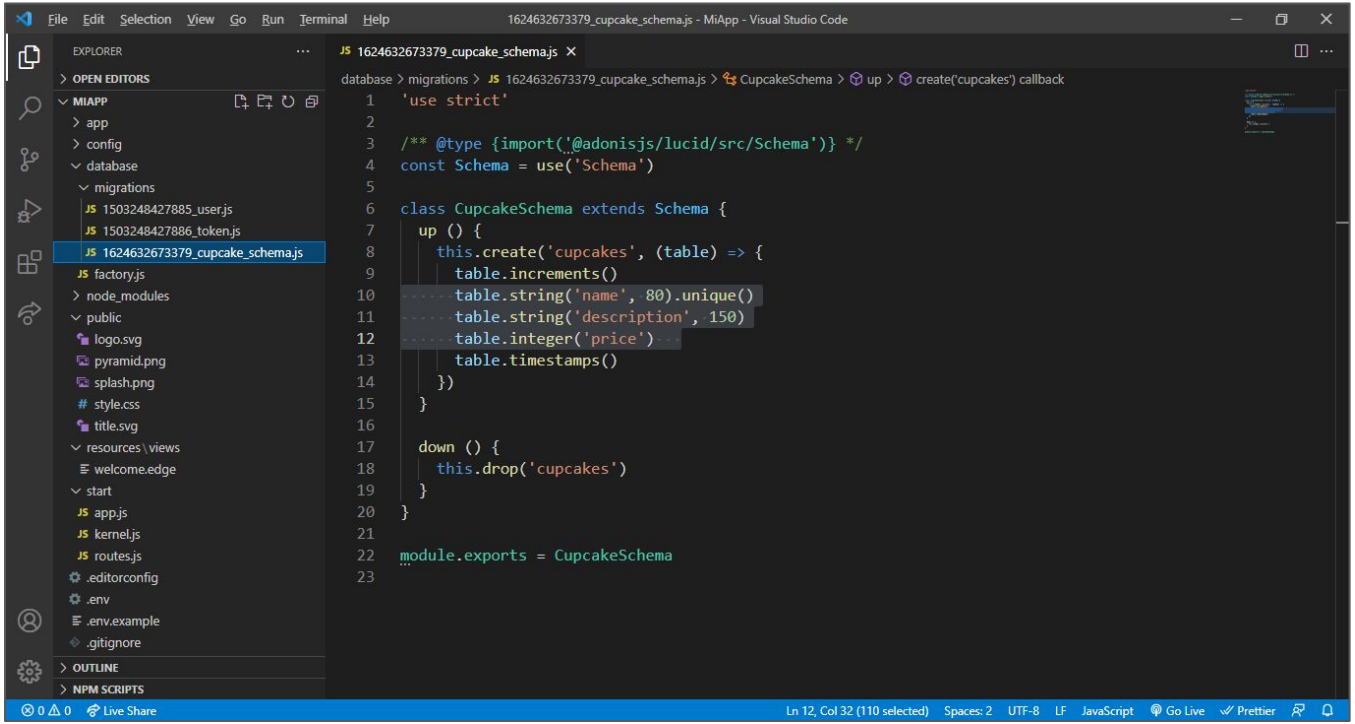
- El modelo *Cupcake* lo podemos ver en app/Models/Cupcake.js.



CODER HOUSE

Usando Adonis - Modelos y migraciones

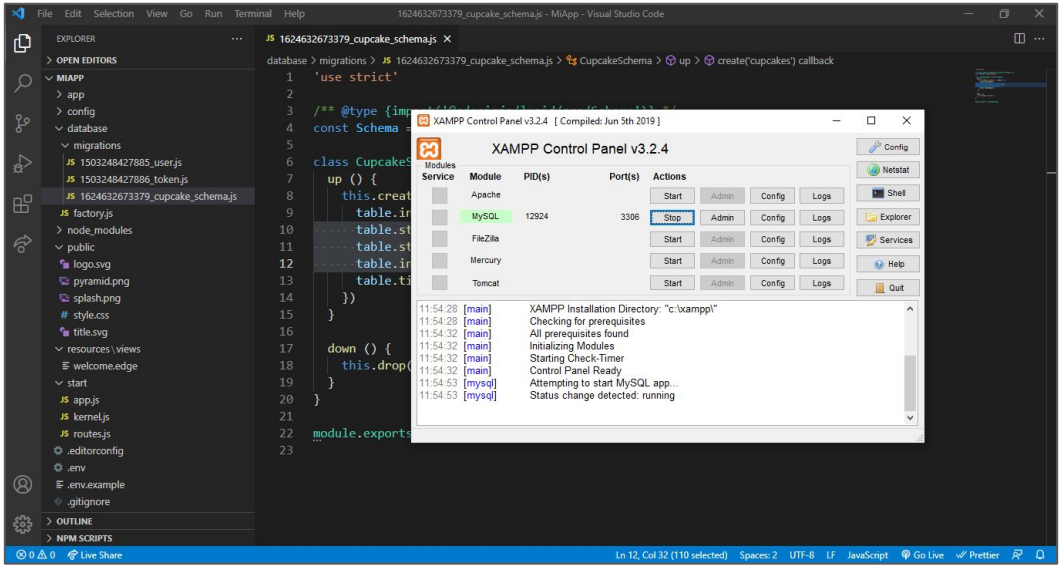
- Es esquema que se genera tiene el siguiente código:



CODER HOUSE

Usando Adonis - Base de datos

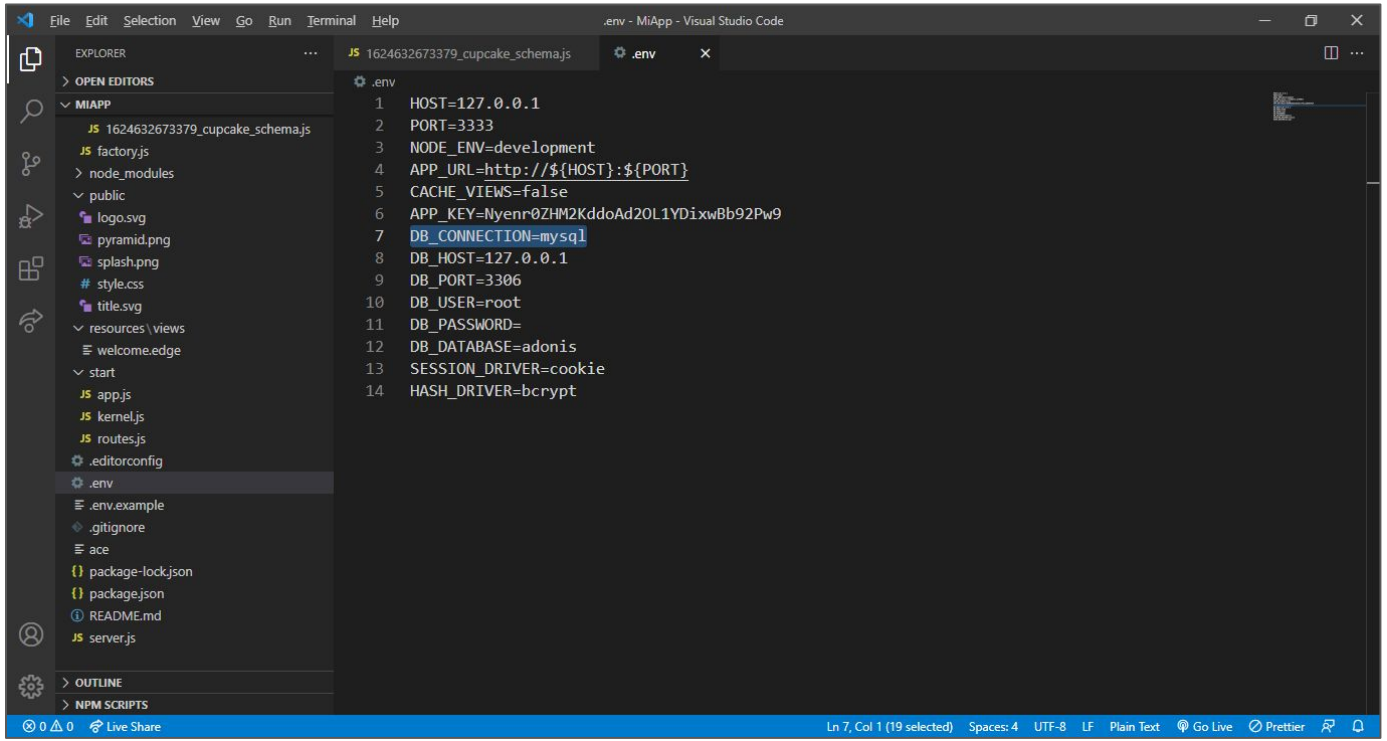
- Para poder correr las migraciones, primero debemos crear la base de datos. En este caso vamos a trabajar con MySQL.
- Para eso, iniciamos nuestro servidor de MySQL (por ejemplo, con XAMPP).



CODER HOUSE

Usando Adonis - Base de datos

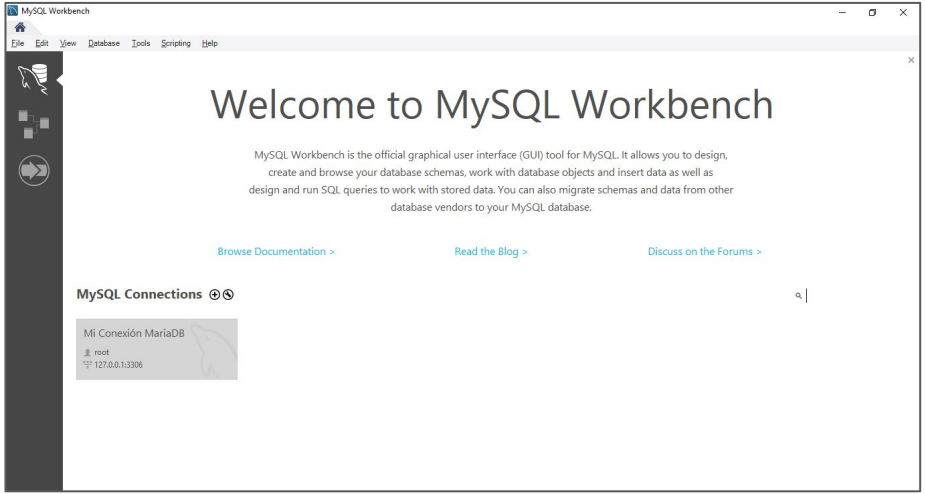
- Luego, configuramos el archivo ***.env*** con las siguientes variables de entorno.

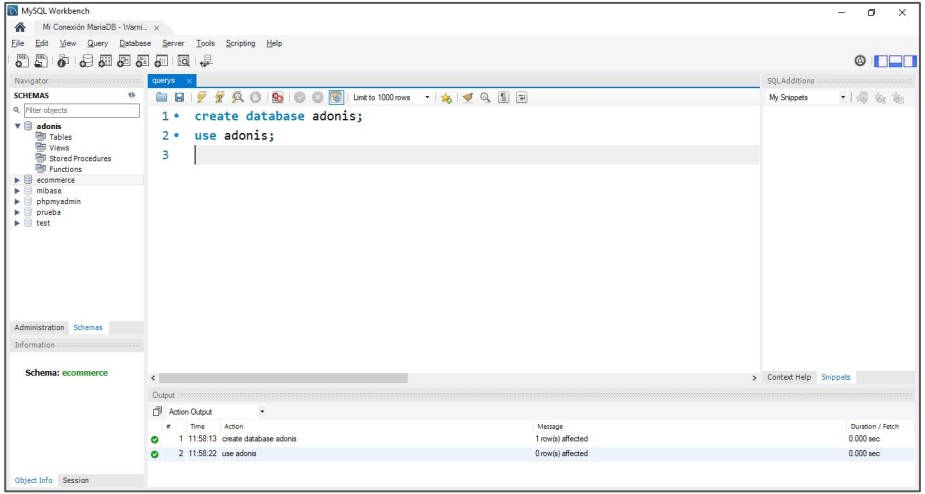


CODER HOUSE

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.

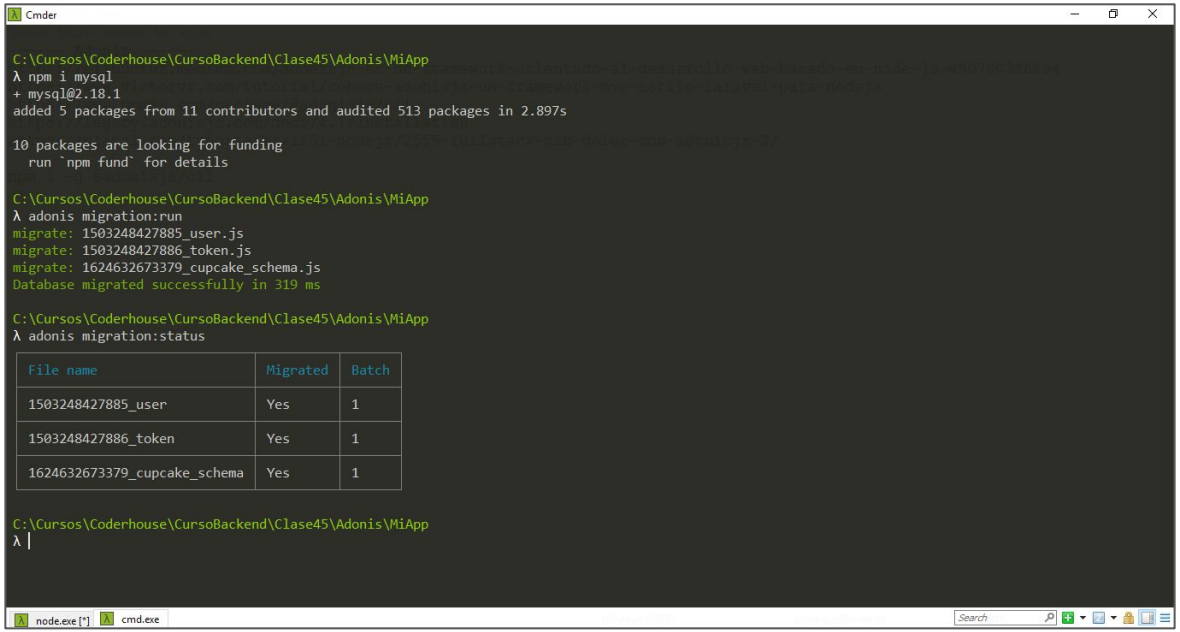




CODER HOUSE

Usando Adonis - Migraciones

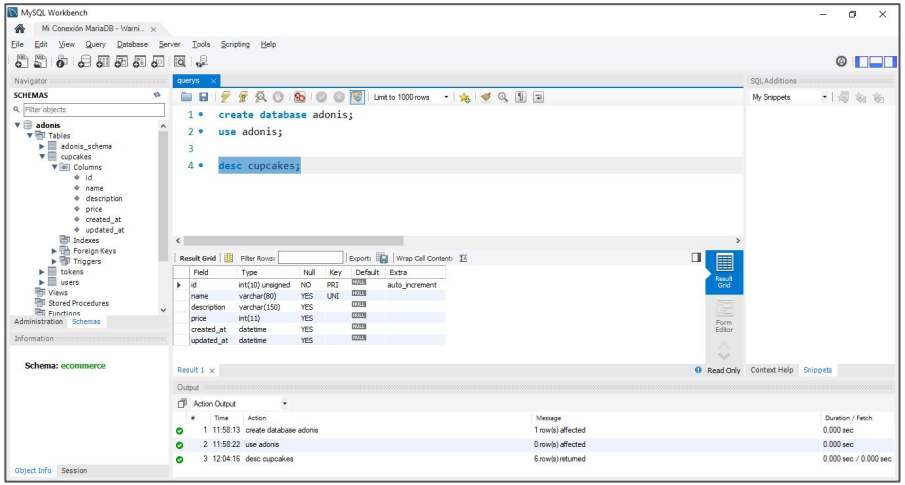
- Vamos nuevamente a la consola para correr las migraciones. Primero debemos instalar *mysql* en nuestro proyecto.
- Luego corremos las migraciones con el comando: *adonis migration:run*
- Podemos ver el status de las mismas con el comando: *adonis migration:status*

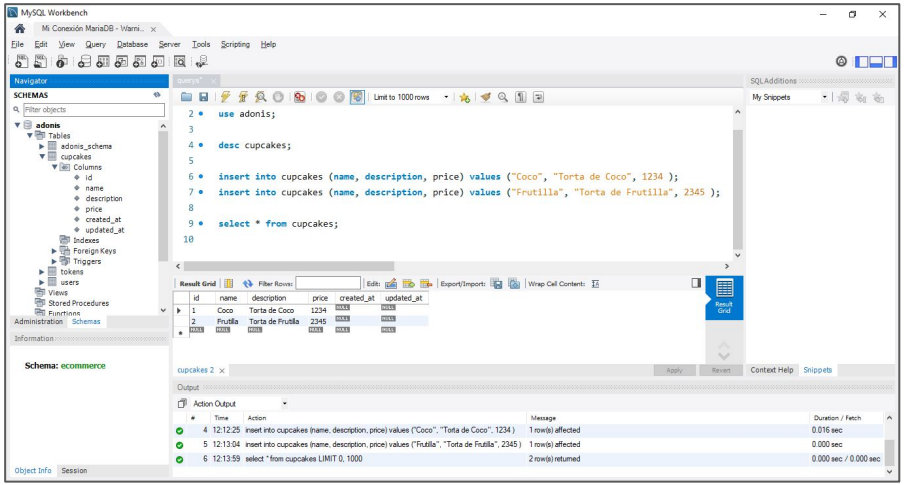


CODER HOUSE

Usando Adonis - Modelo

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

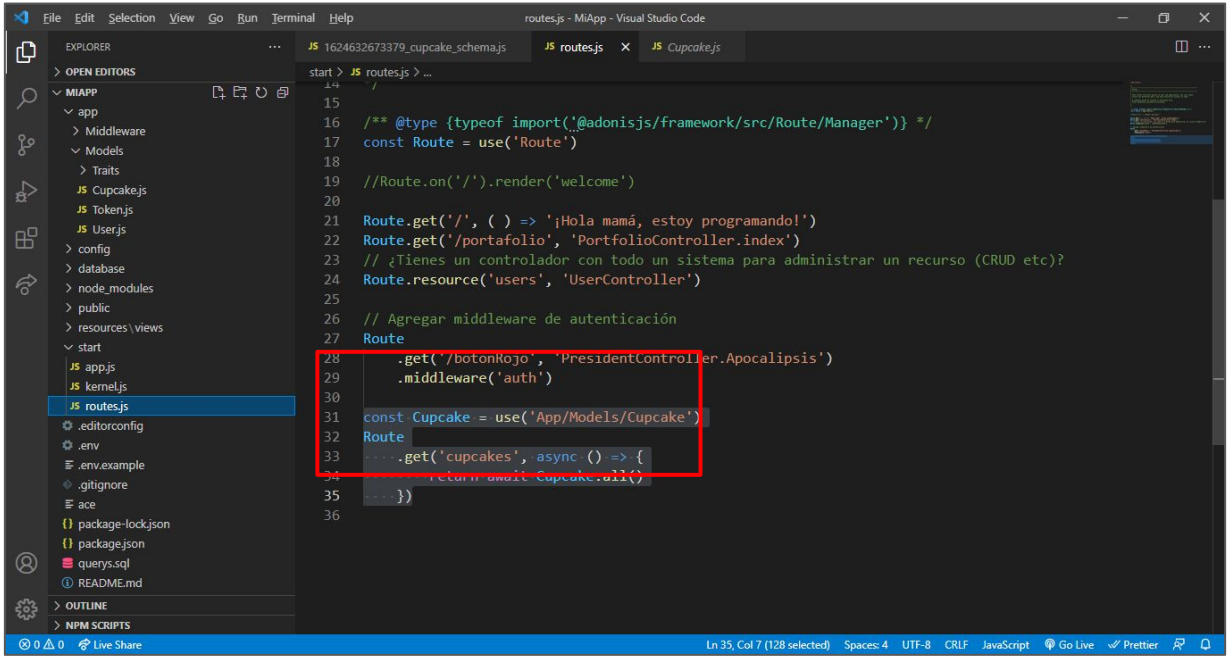




CODER HOUSE

Usando Adonis - Modelo

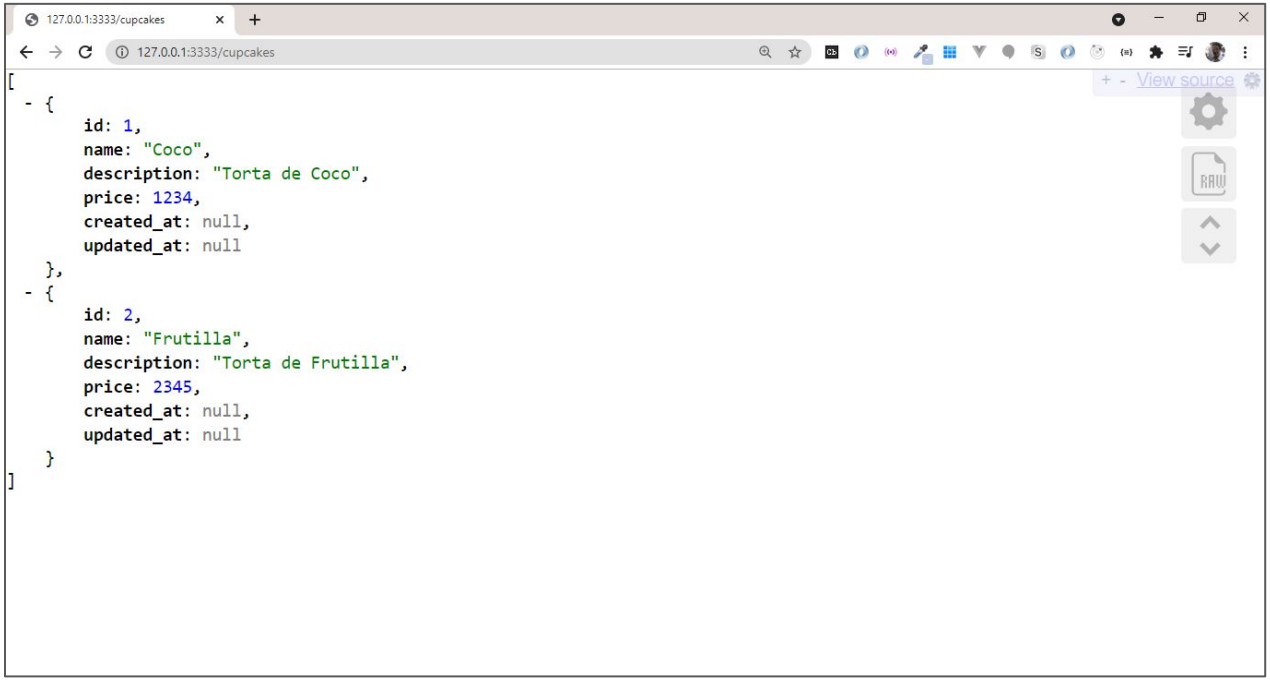
- Vamos ahora nuevamente al archivo de rutas, y agregamos el siguiente código para agregar una ruta para el modelo.



CODER HOUSE

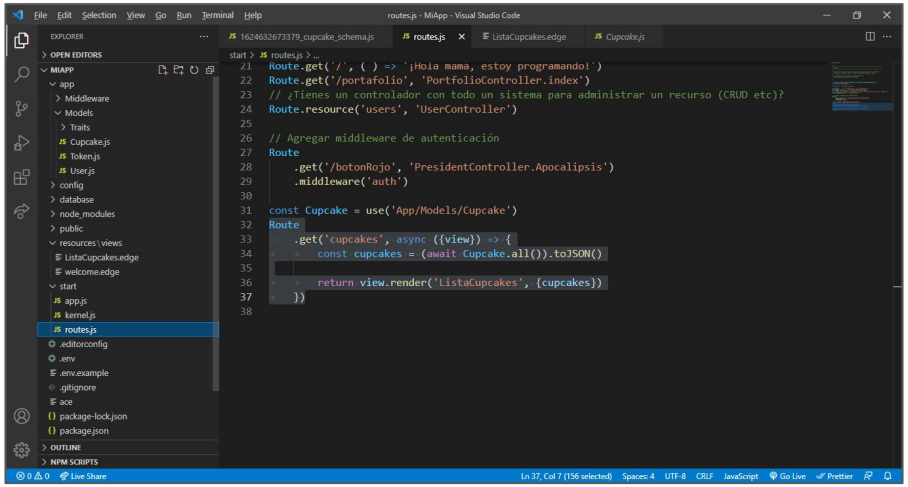
Usando Adonis - Vista del modelo

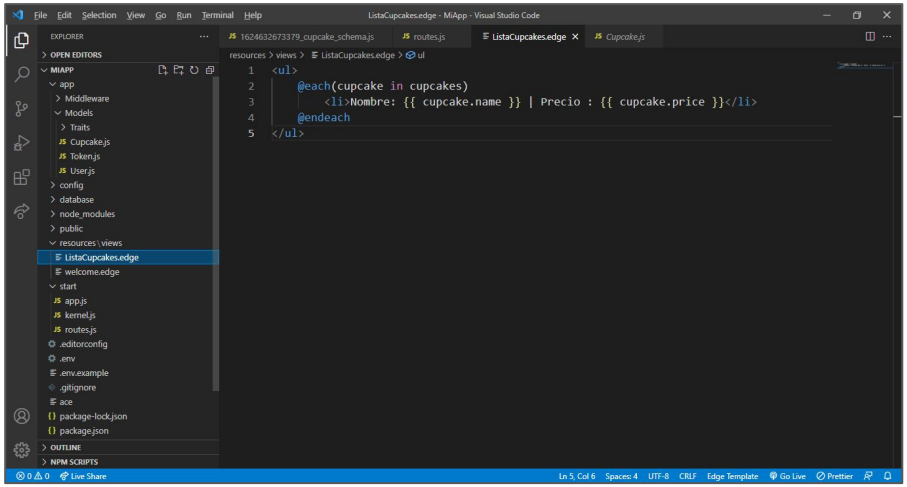
- Vemos entonces en el navegador, un JSON con todos los registros de la tabla cupcake que tenemos en nuestra base de datos.



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:

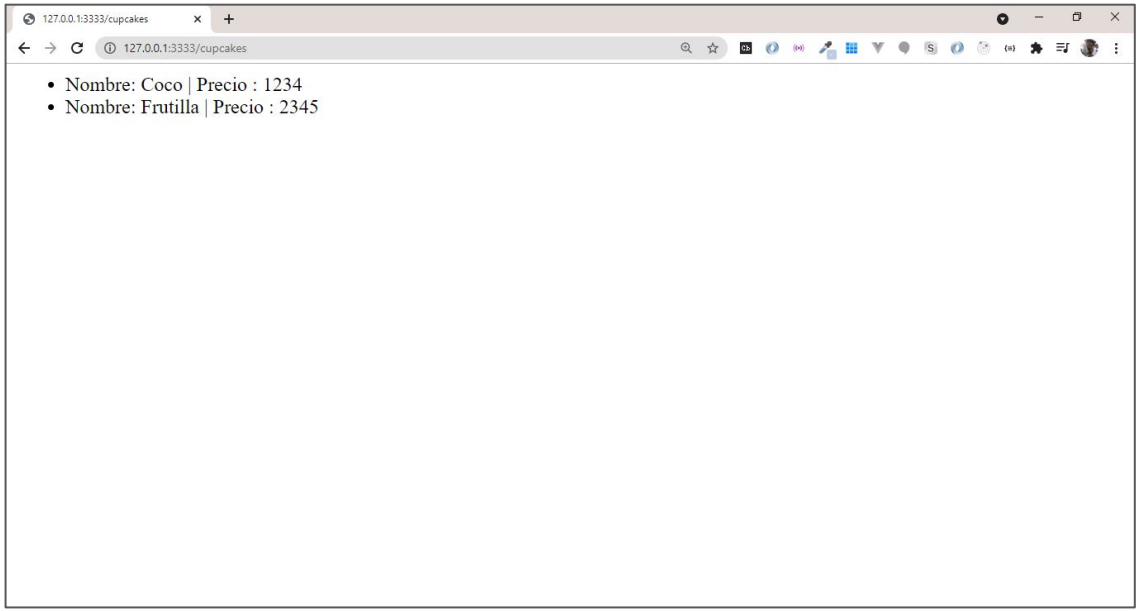




CODER HOUSE

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 - consola

- Finalmente, en la consola podemos visualizar como se reinicia el servidor cada vez que guardamos un cambio de alguno de los archivos. No lo debemos hacer de forma manual.

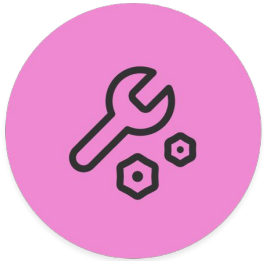
```
Cmder
C:\Cursos\Coderhouse\CursoBackend\Clase45\Adonis
λ cd MiApp\

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
changed c:Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp\start\routes.js
info: serving app on http://127.0.0.1:3333
changed c:Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp\app\Models\Cupcake.js
changed c:Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp\database\migrations\1624632673379_cupcake_schema.js
info: serving app on http://127.0.0.1:3333
changed c:Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp\database\migrations\1624632673379_cupcake_schema.js
info: serving app on http://127.0.0.1:3333
changed c:Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp\package.json
changed c:Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp\package-lock.json
info: serving app on http://127.0.0.1:3333
changed c:Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp\start\routes.js
info: serving app on http://127.0.0.1:3333
changed c:Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp\start\routes.js
info: serving app on http://127.0.0.1:3333
changed c:Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp\start\routes.js
info: serving app on http://127.0.0.1:3333
changed c:Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp\start\routes.js
info: serving app on http://127.0.0.1:3333
changed c:Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp\start\routes.js
info: serving app on http://127.0.0.1:3333
changed c:Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp\start\routes.js
info: serving app on http://127.0.0.1:3333

node.exe   cmd.exe
```

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

CODER HOUSE

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 respones, entre otros.

Ej. entradas:

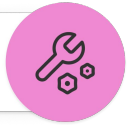
http://127.0.0.1:3333/sin-controller?palabras=hola mundo como están todos ustedes!

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

USANDO ADONIS

Tiempo: 15 minutos

Desafío generico

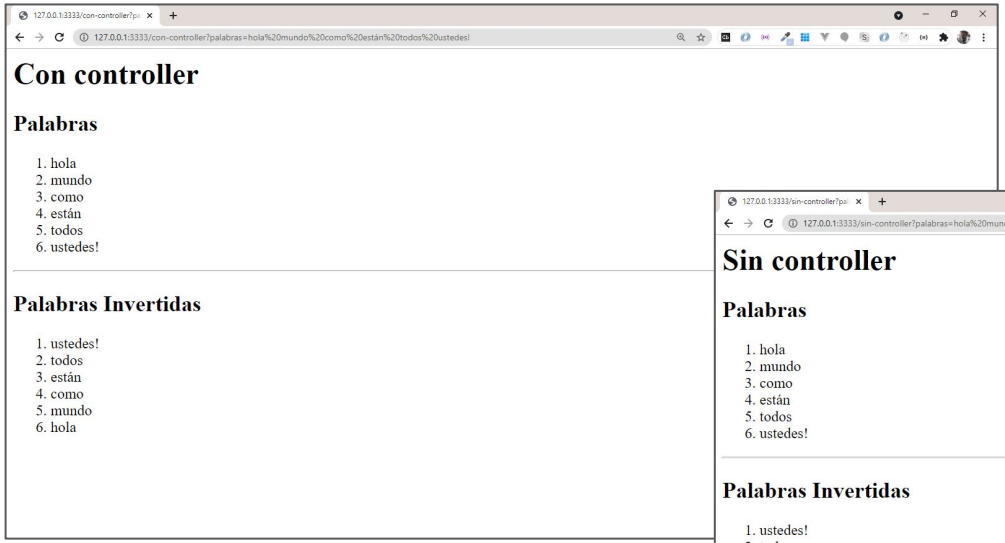


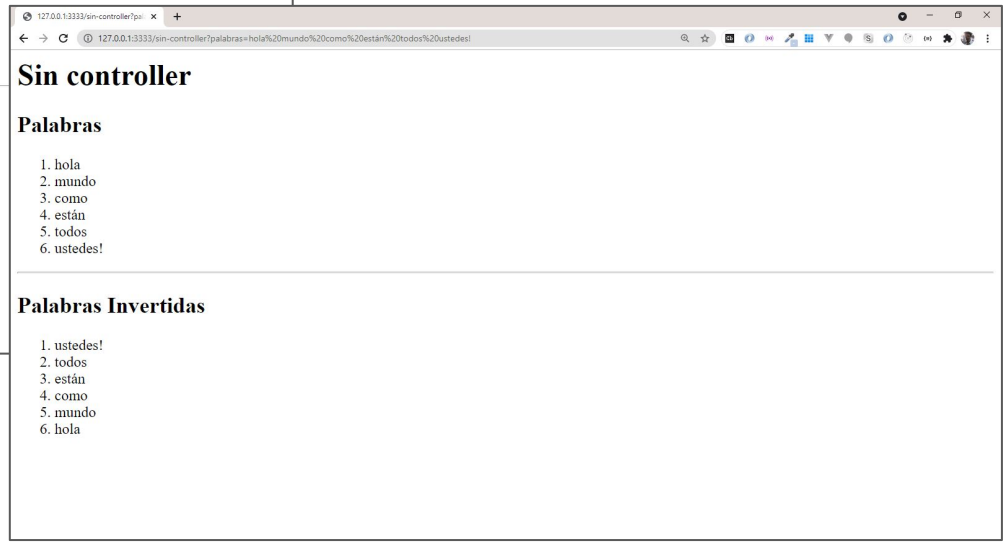
- ➔ La ruta '/sin-controller' debe realizar el proceso sin utilizar un controlador de ruta, mientras que '/con-controller' si.
- ➔ Representar también en la vista, si la respuesta viene de un proceso u otro.

CODER HOUSE

Desafío
generico

>> Ejemplos de cada ruta.





CODER HOUSE

BREAK

¡5/10 MINUTOS Y VOLVEMOS!

NEST

CODER HOUSE

¿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

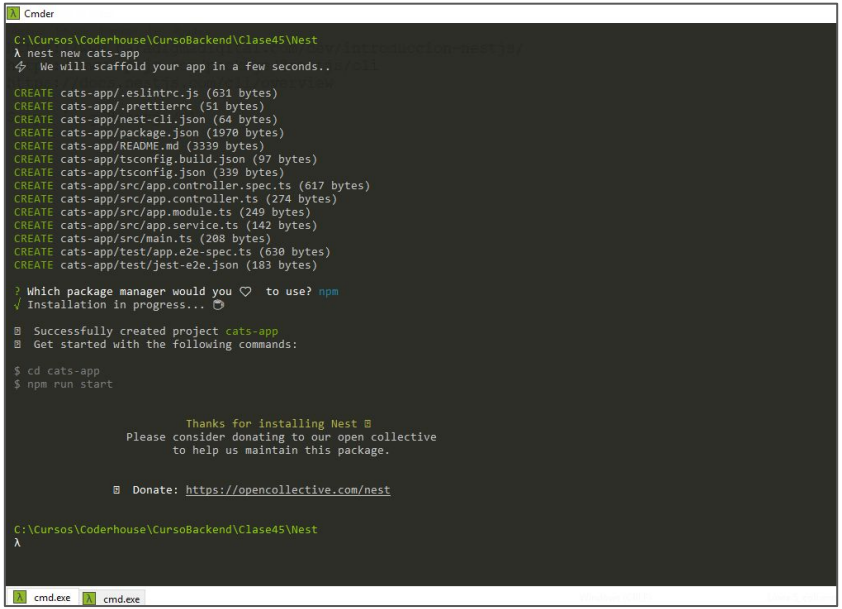
Empezar a usarlo - Instalación

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

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

- Para crear nuestro proyecto en Nest usamos el comando:

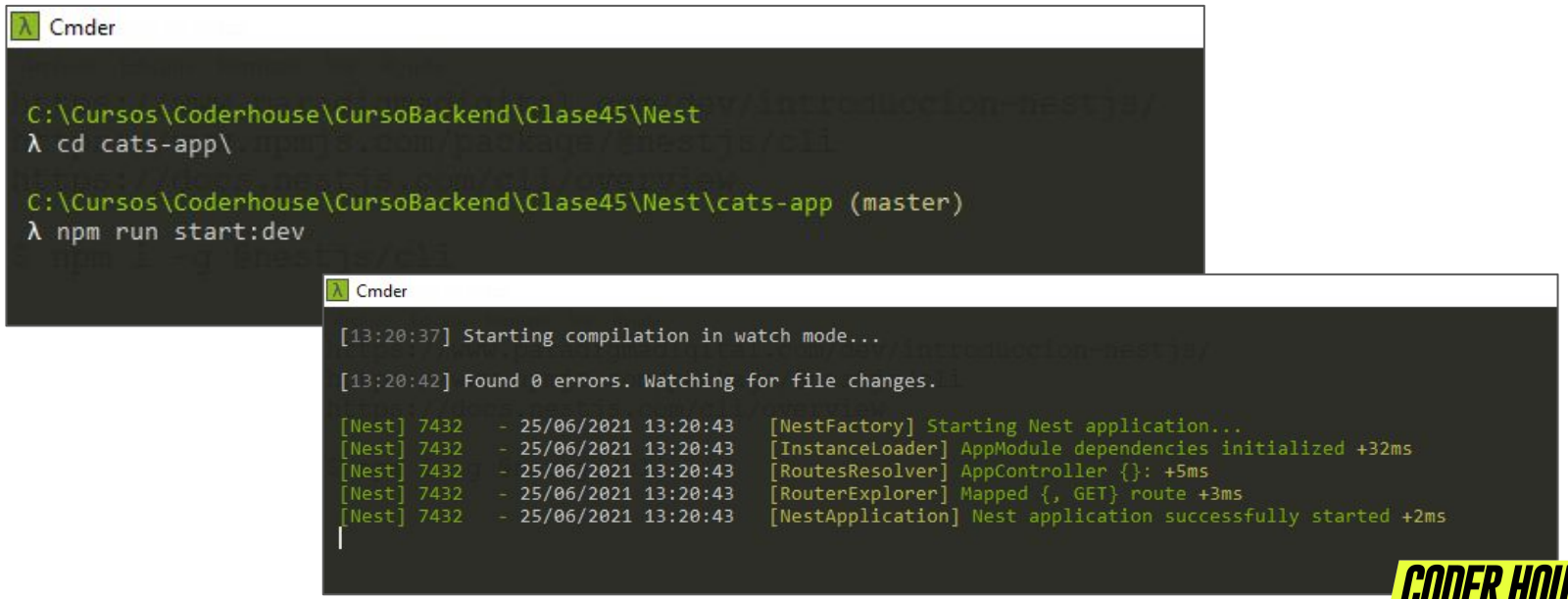
```
$ nest new my-nest-project
```



CODER HOUSE

Empezar a usarlo - Iniciar servidor

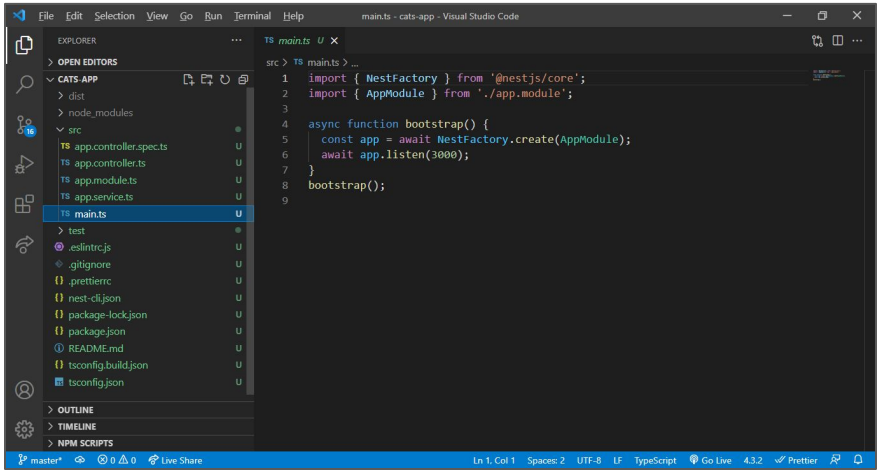
- Para iniciar el servidor, ingresamos a la carpeta creada del proyecto y utilizamos el comando: *npm run start:dev*.

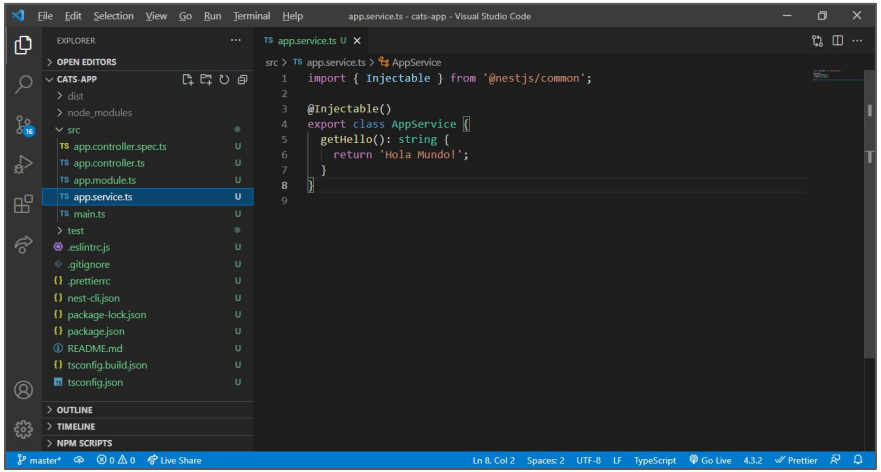


CODER HOUSE

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()*.



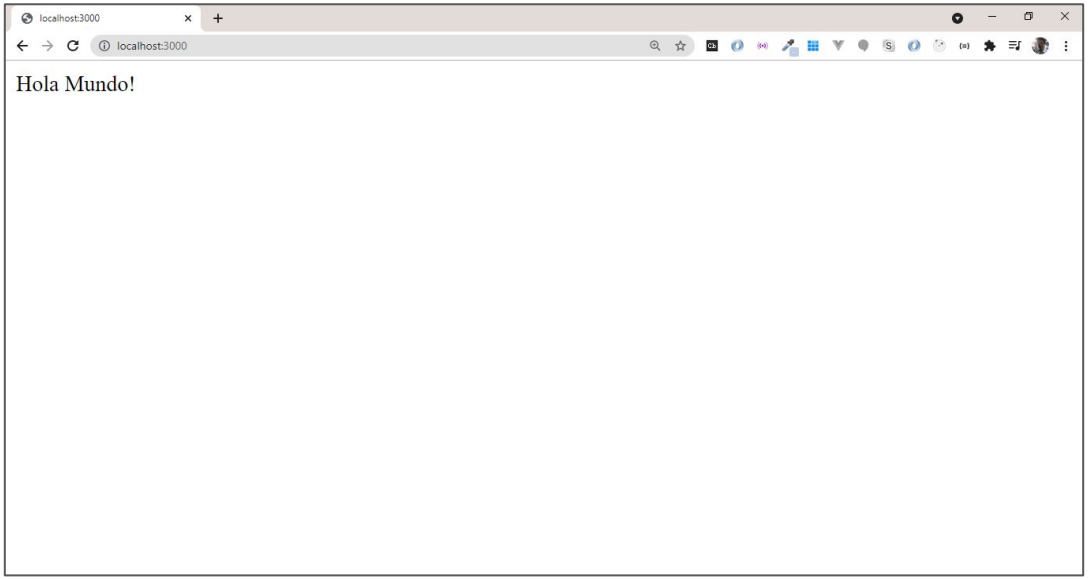


Para quienes la conozcan, verán que la estructura de carpetas es casi igual a la se crea en Angular

CODER HOUSE

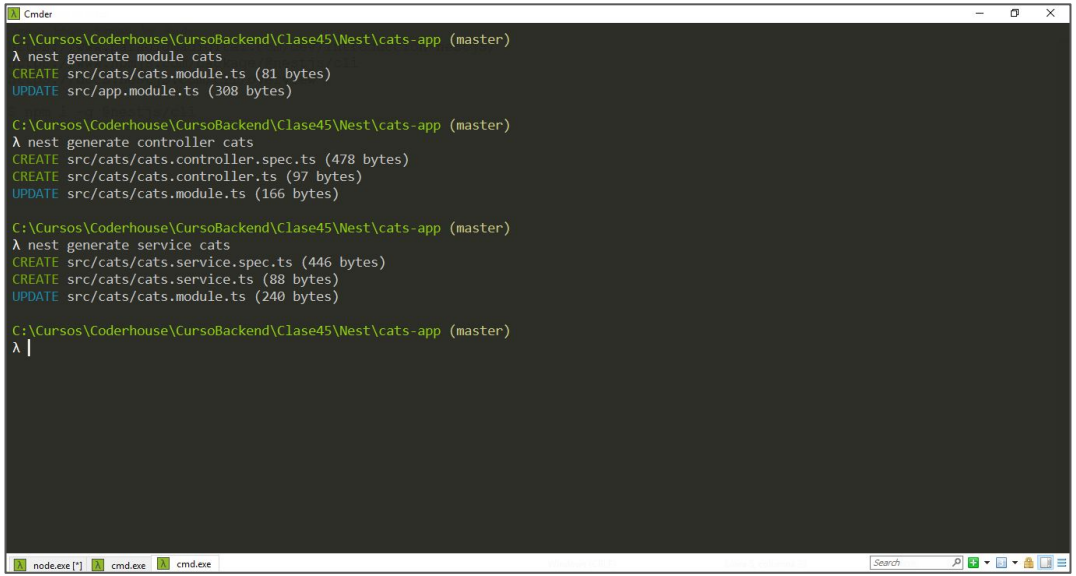
Empezar a usarlo - Iniciar servidor

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



Crear módulo, controlador y servicio

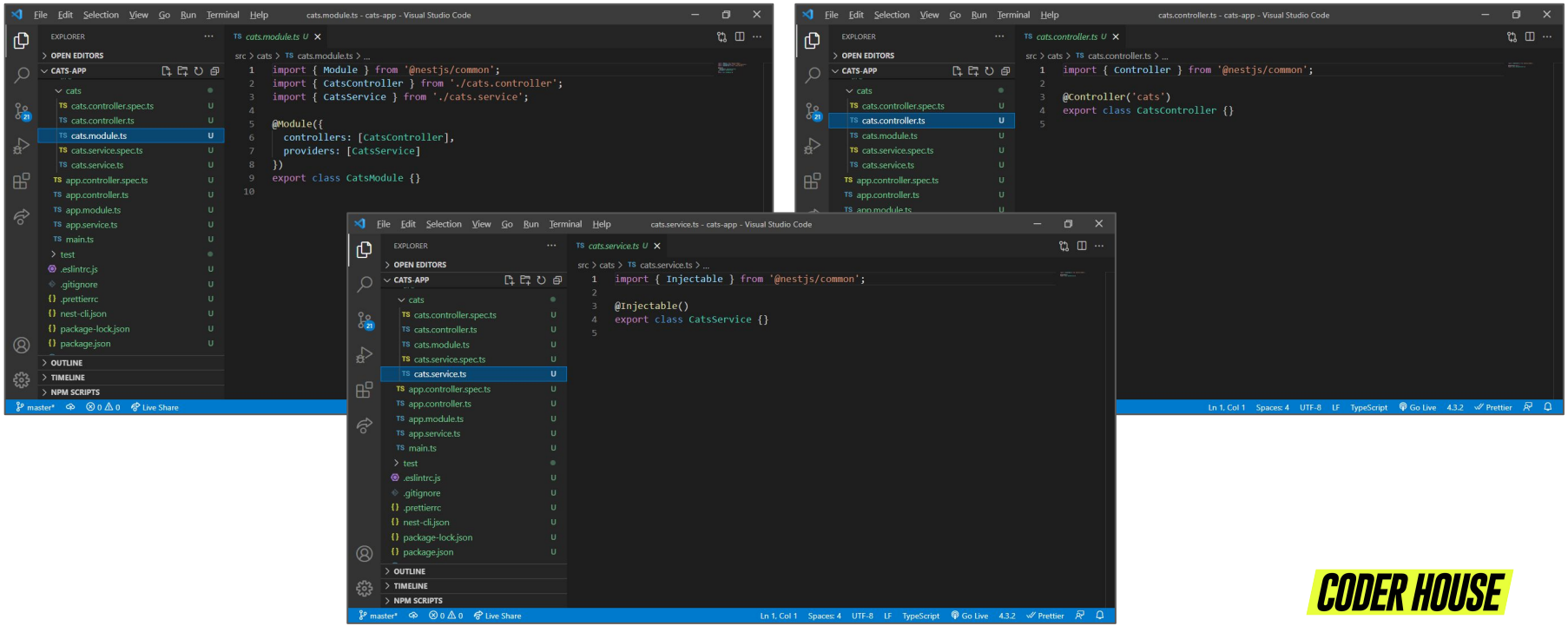
- Para crear un nuevo módulo usamos el comando: *nest generate module <name>*
- Para crear un nuevo controlador usamos: *nest generate controller <name>*
- Para crear un nuevo servicio usamos: *nest generate service <name>*.



CODER HOUSE

Crear módulo, controlador y servicio

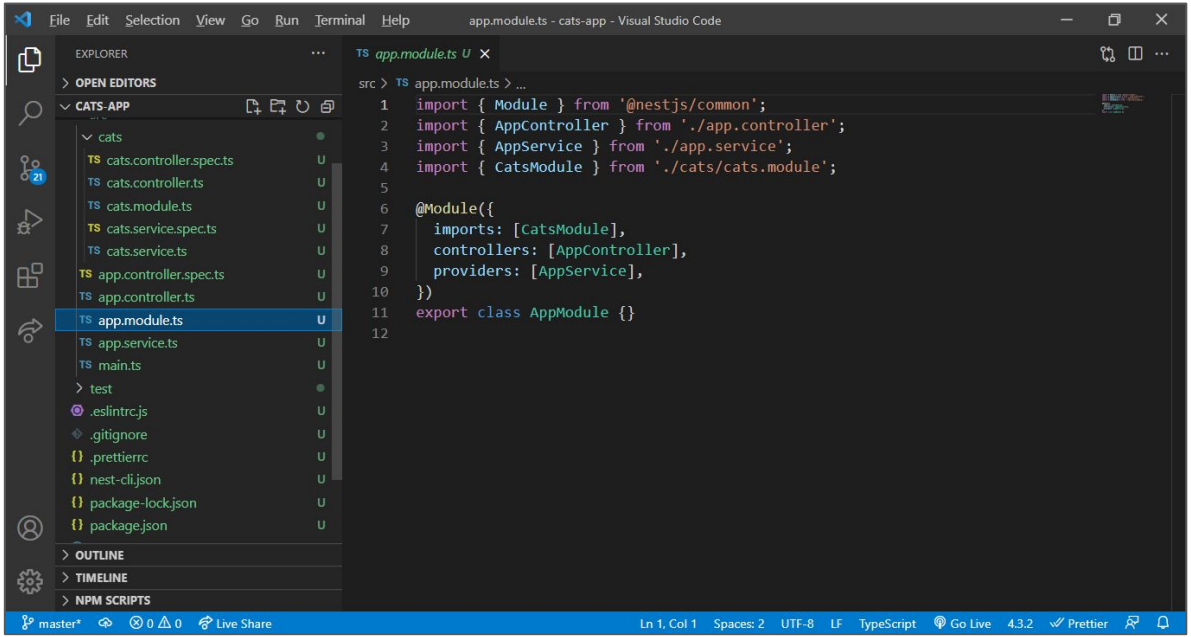
- Se crean los siguientes archivos dentro de una carpeta con el nombre del módulo:



CODER HOUSE

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:



CODER HOUSE

Interface y DTO

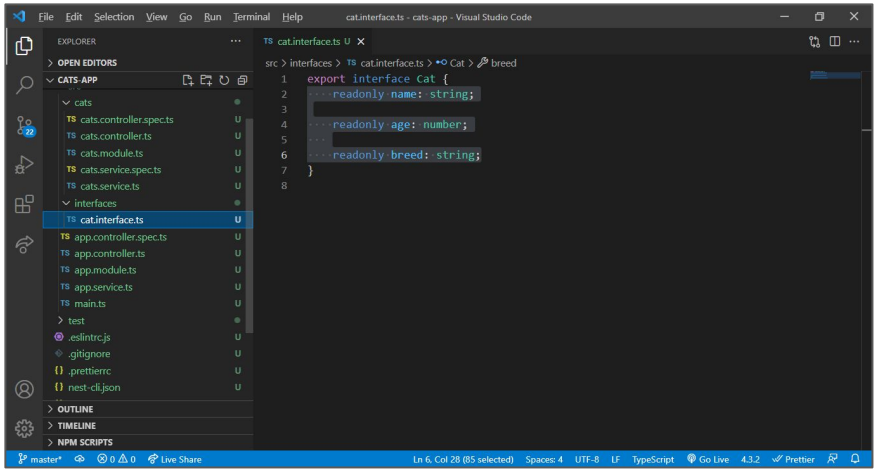
- Antes de empezar a crear endpoints y añadir lógica, creamos dos elementos que vamos a necesitar. Uno de ellos es una ***Interface*** de cat y el otro es el DTO para la creación del objeto tipo Cat. Los creamos con los comandos que vemos.

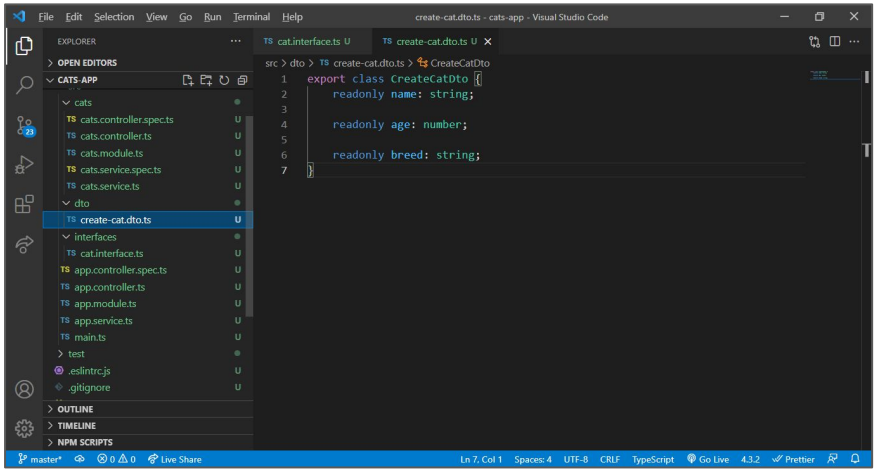
```
C:\Cursos\Coderhouse\CursoBackend\Clase45\Nest\cats-app (master)
λ nest generate interface interfaces/cat
CREATE src/interfaces/cat.interface.ts (24 bytes)

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
```

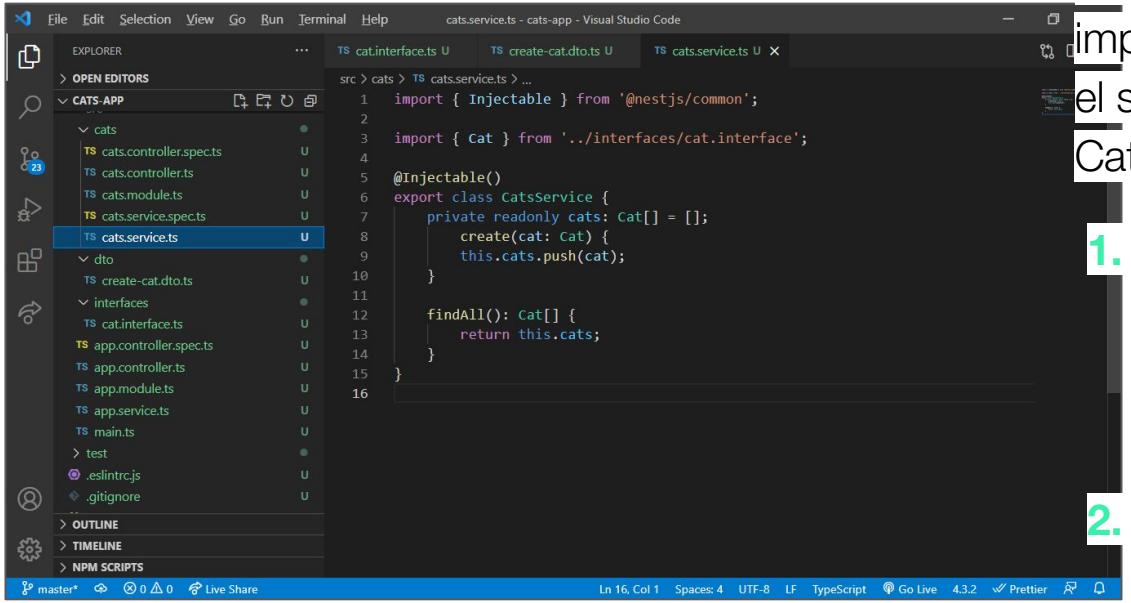
Interface y DTO

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





Implementación de un servicio



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.

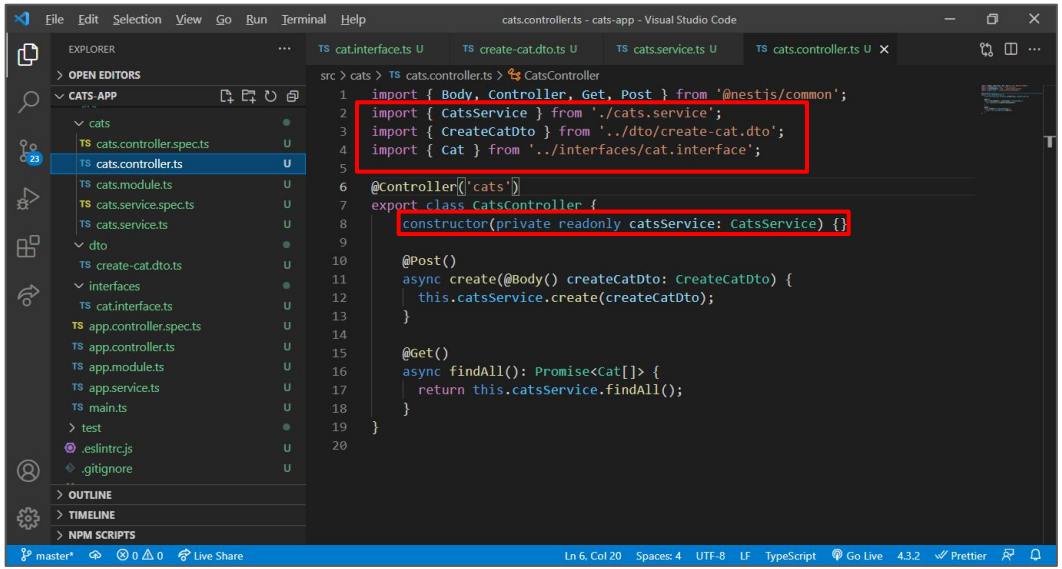
2. Luego, definimos el método *findAll* que nos devuelve el array con la información de los gatos almacenada.

CODER HOUSE

Implementación de un controlador

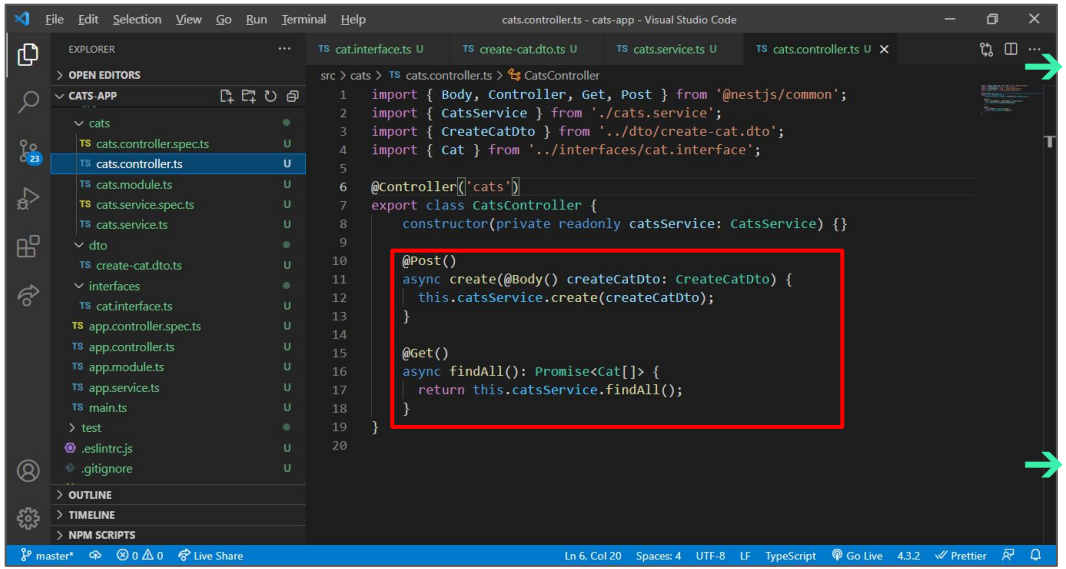
4. Ahora vamos a la parte del controlador. Importamos en el mismo la Interface *Cat*, el DTO *CreateCatDto* y el servicio *CatService*.

5. Agregamos una instancia del servicio en el constructor para poder utilizarlo en el controlador.



CODER HOUSE

Implementación de un controlador



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

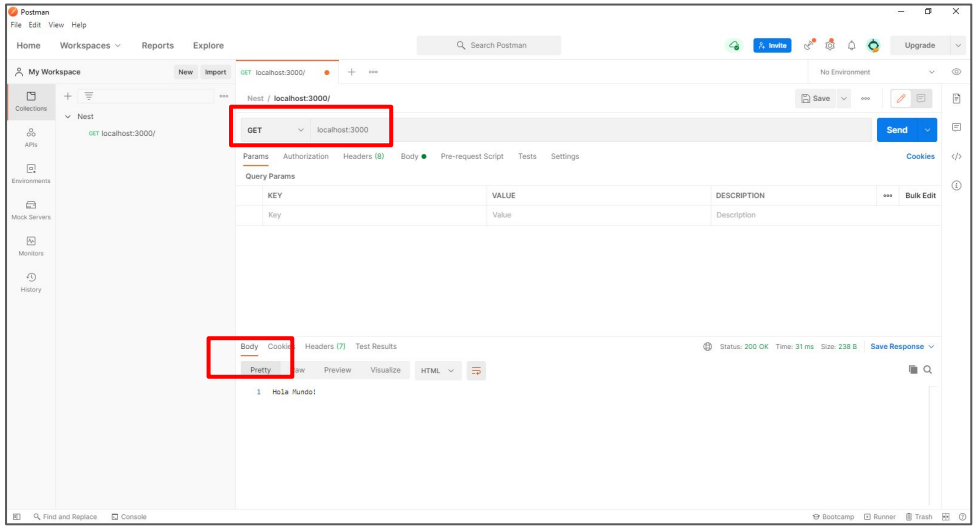
En el método *create*, indicamos a través de la etiqueta decoradora el tipo de acción que realiza (POST). Por parámetros le indicamos que va a recibir un objeto de tipo @Body, que a su vez es el que vamos a enviar al servicio para crear el objeto tipo Cat.

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*.

CODER HOUSE

Consumir la API con Postman

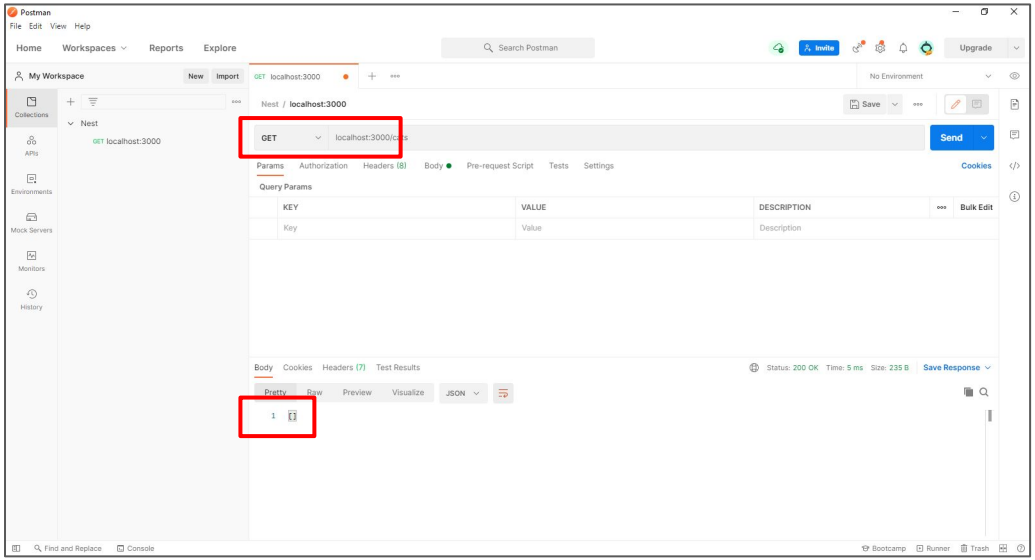
- Levantamos nuestra app con el comando: *npm run start*.
- Vamos a Postman e ingresamos la petición por GET de http://localhost:3000.
- Vemos que obtenemos el “Hola Mundo” que configuramos en el *app.service.ts*.



CODER HOUSE

Consumir la API con Postman

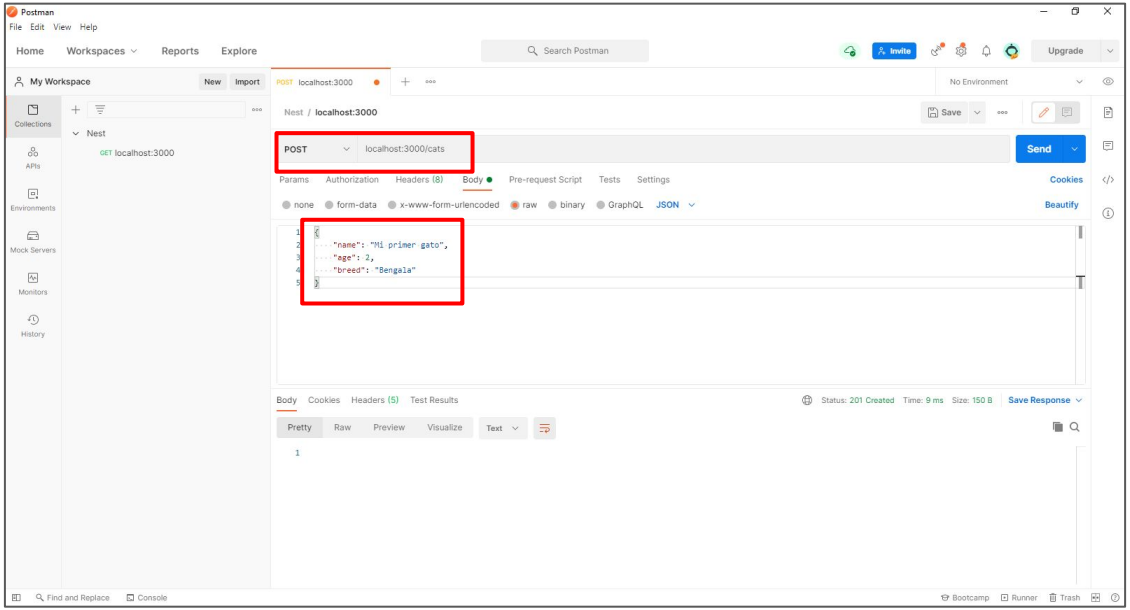
- Probamos ahora la ruta de “/cats” por GET con la url http://localhost:3000/cats.
- Vemos que obtenemos un array vacío ya que todavía no tenemos información de gatos creados.



CODER HOUSE

Consumir la API con Postman

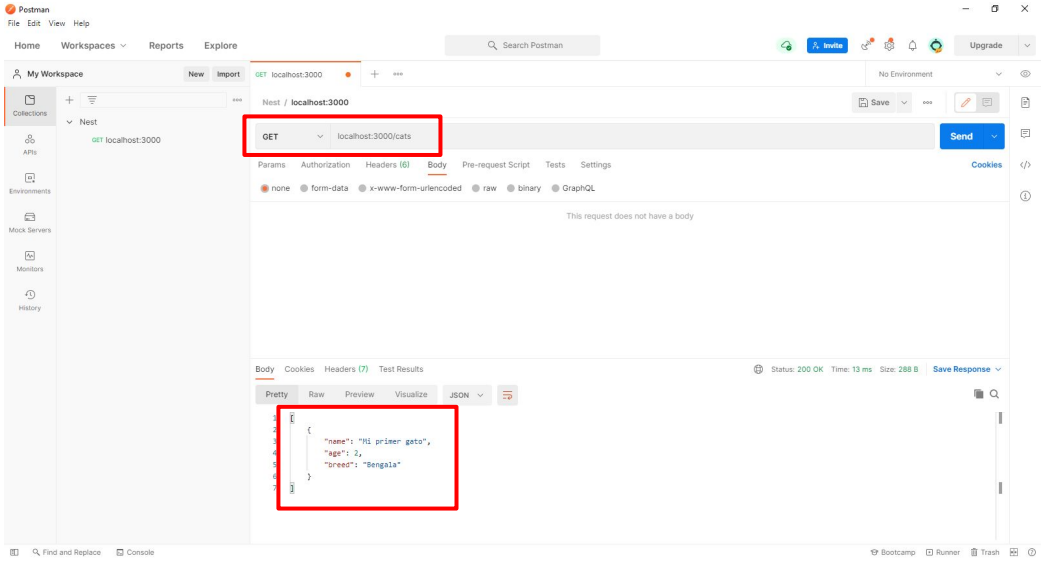
- Creamos un gato nuevo con la ruta de “/cats” por POST con la url http://localhost:3000/cats.
- Los atributos de gato son: *name*, *age* y *breed*. Se los pasamos como JSON.



CODER HOUSE

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).



CODER HOUSE

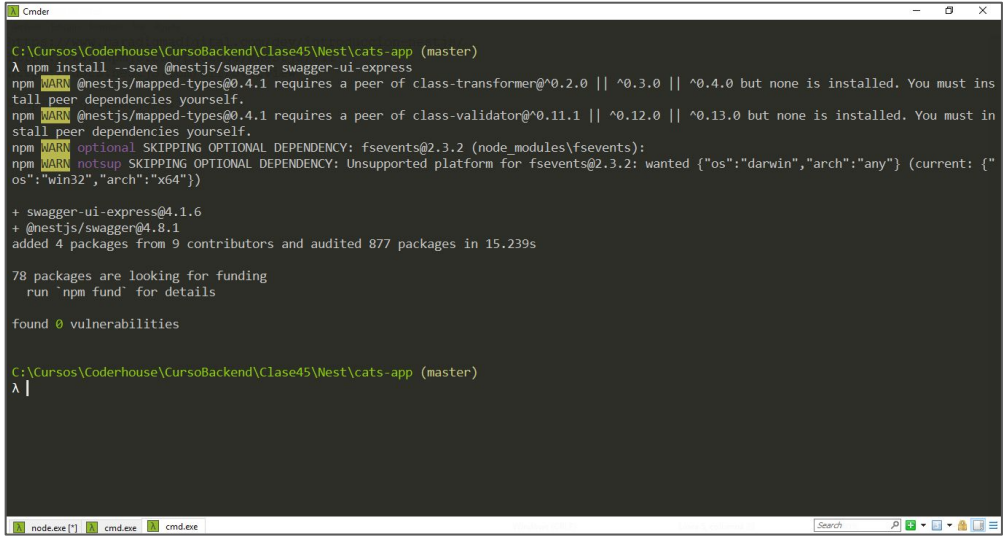
INCORPOREMOS SWAGGER!

Instalación de Swagger

- Instalamos Swagger con el comando:

  npm install --save @nestjs/swagger swagger-ui-express

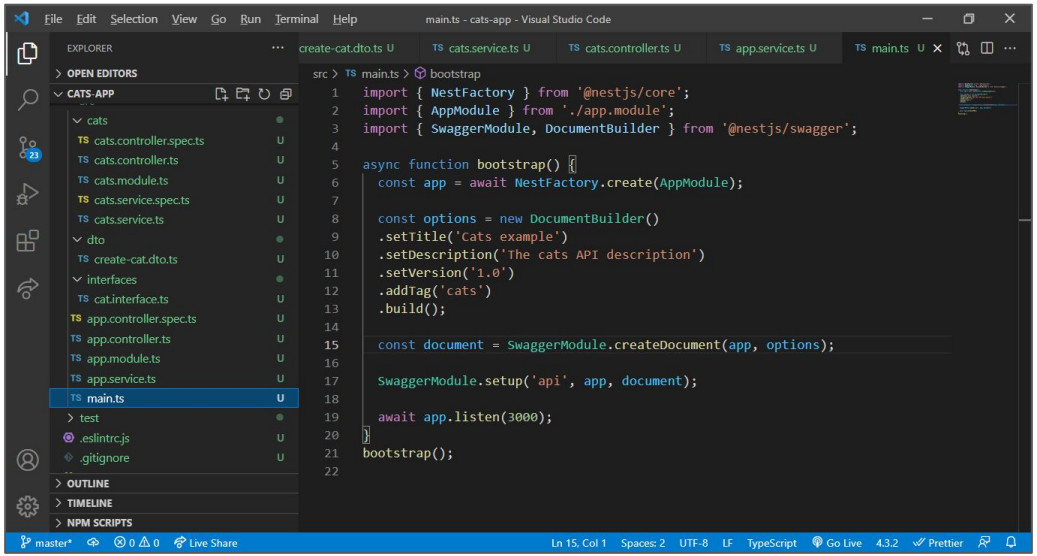
- De esta forma se instala el módulo de nestjs/swagger y swagger-ui-express para poder empezar a usar esta librería.



CODER HOUSE

Implementación de Swagger

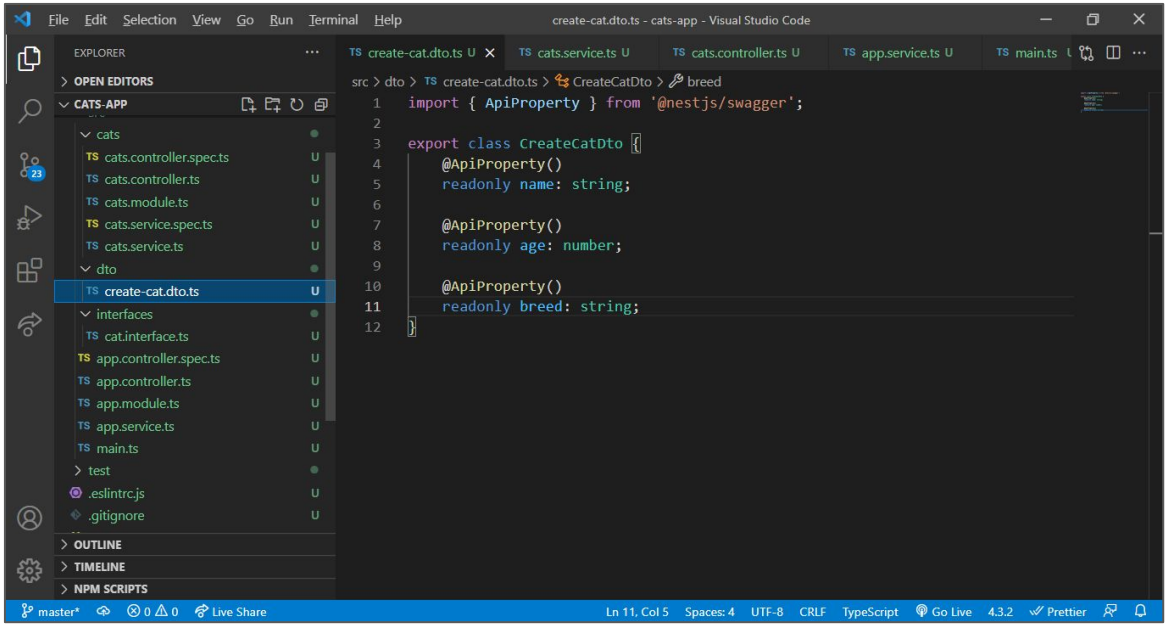
- Vamos ahora al archivo ***main.ts*** y especificamos la configuración de Swagger (título, descripción, versión…).
- Finalmente, se inyecta esta API Swagger en la aplicación y le damos una ruta de acceso, en este caso "*api*".



CODER HOUSE

Implementación de Swagger

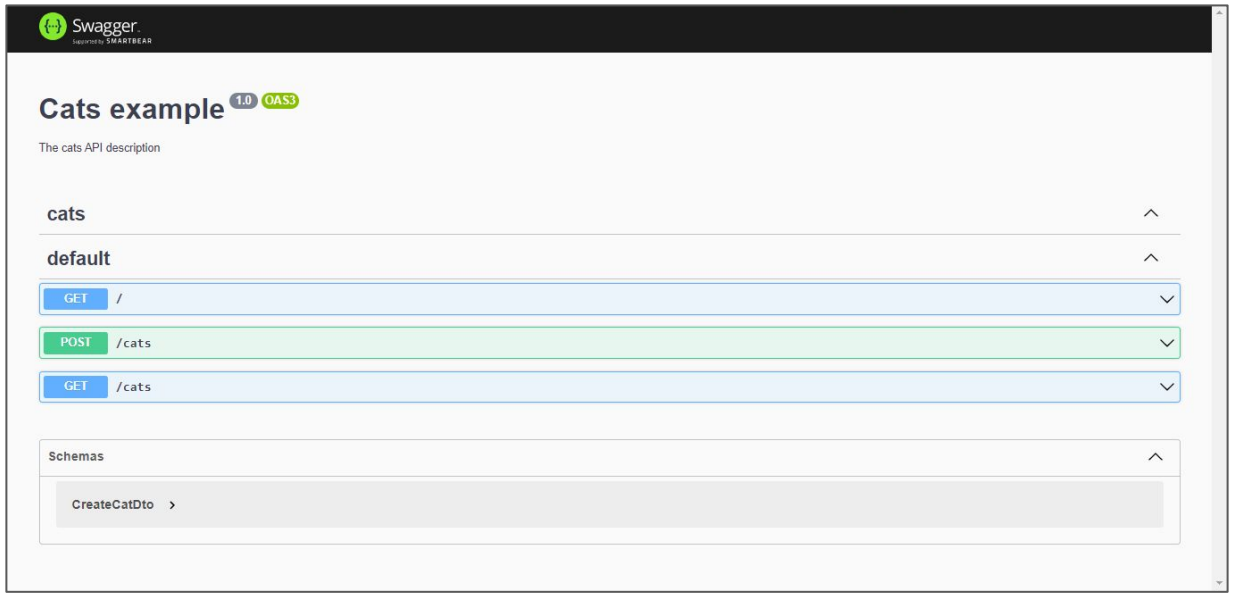
- Por último, agregamos una etiqueta decoradora básica @ApiProperty en el DTO ***CreateCatDto*** para que se muestre en Swagger sus propiedades bien definidas.



CODER HOUSE

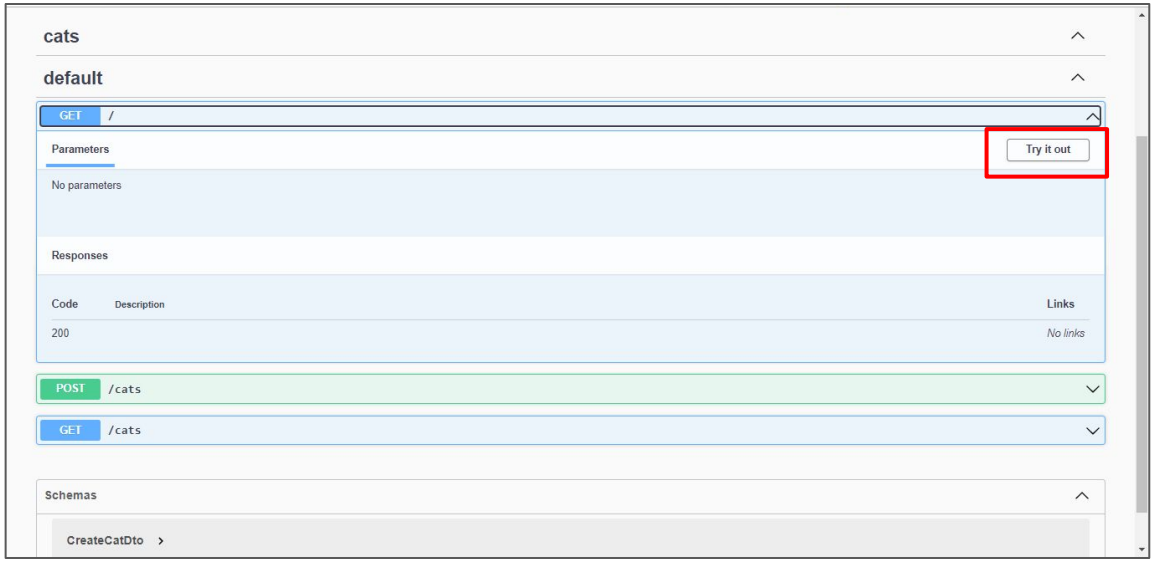
Implementación de Swagger

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



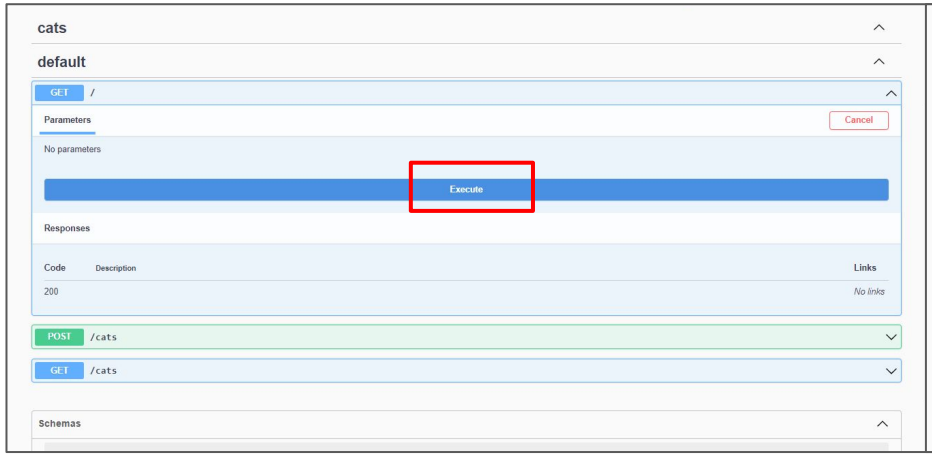
Implementación de Swagger

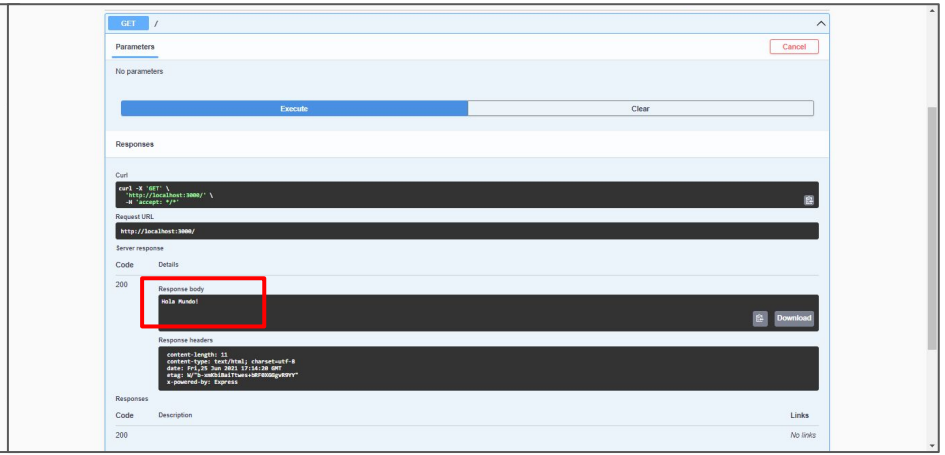
- Si vamos al primer endpoint, que es la ruta home por GET (“/”), vemos lo siguiente.
- Para empezar a ejecutar, clickeamos en el botón “Try it out”.



Implementación de Swagger

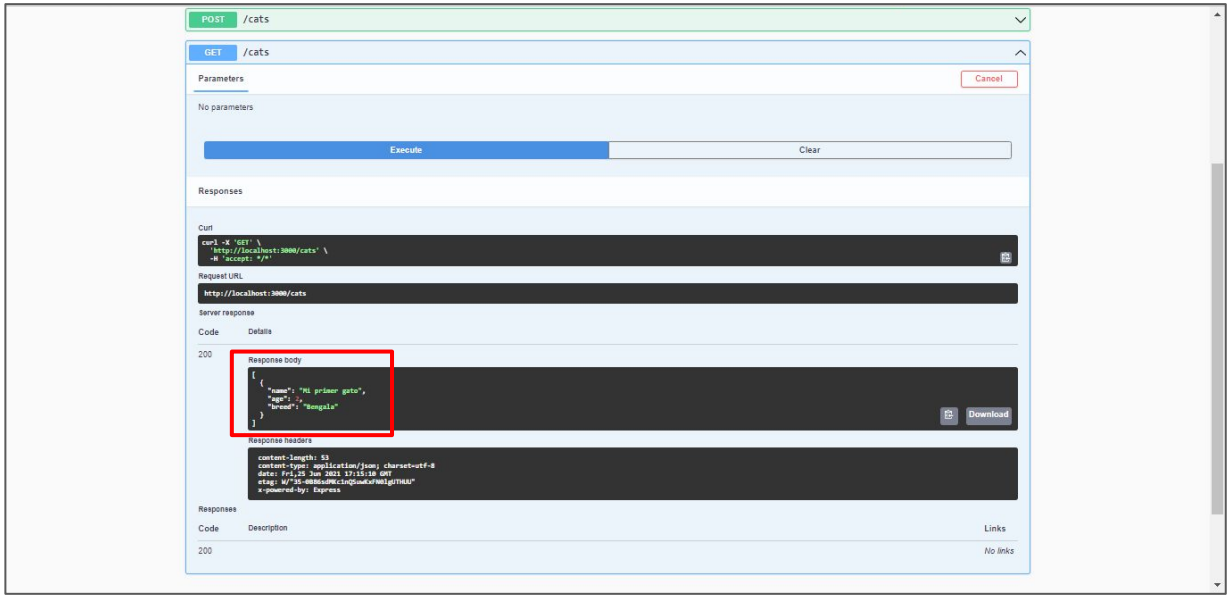
- Clickeando en el botón “Execute” se ejecuta el endpoint y obtenemos la respuesta en “Response body” que en este caso era “Hola Mundo”.





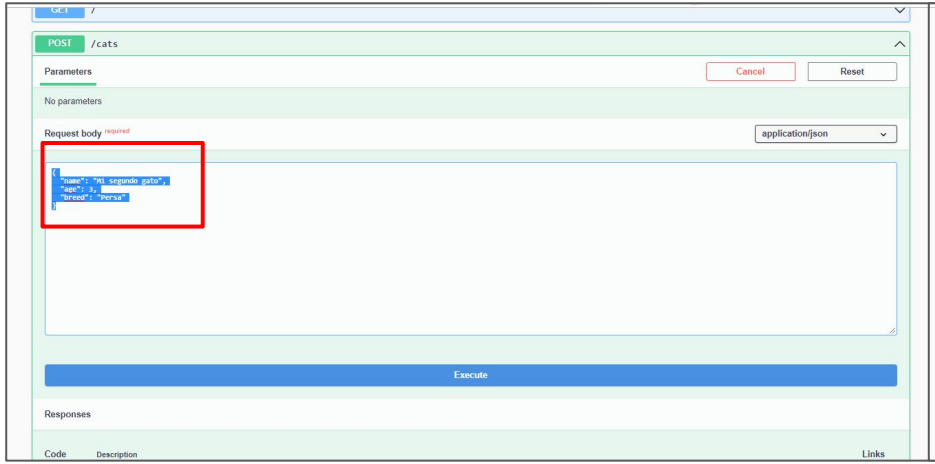
Implementación de Swagger

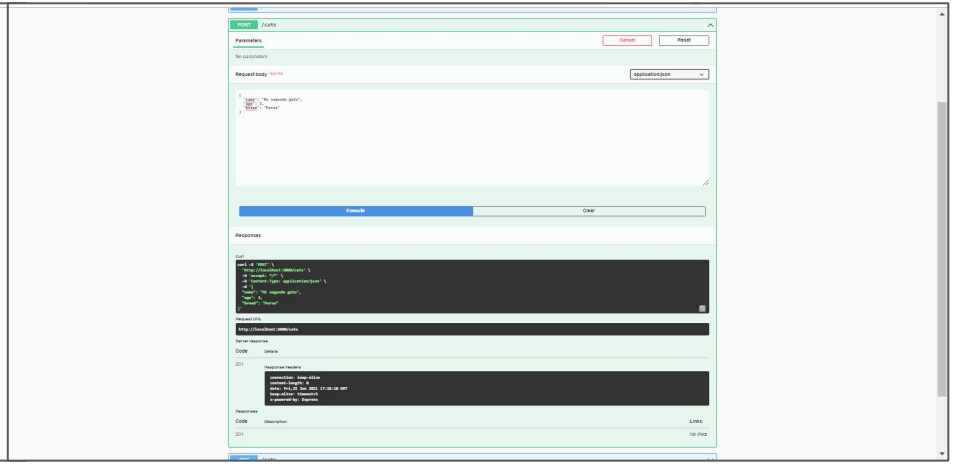
- Si ahora ejecutamos el endpoint por GET de “/cats” obtenemos la información del gato que agregamos antes por Postman.



Implementación de Swagger

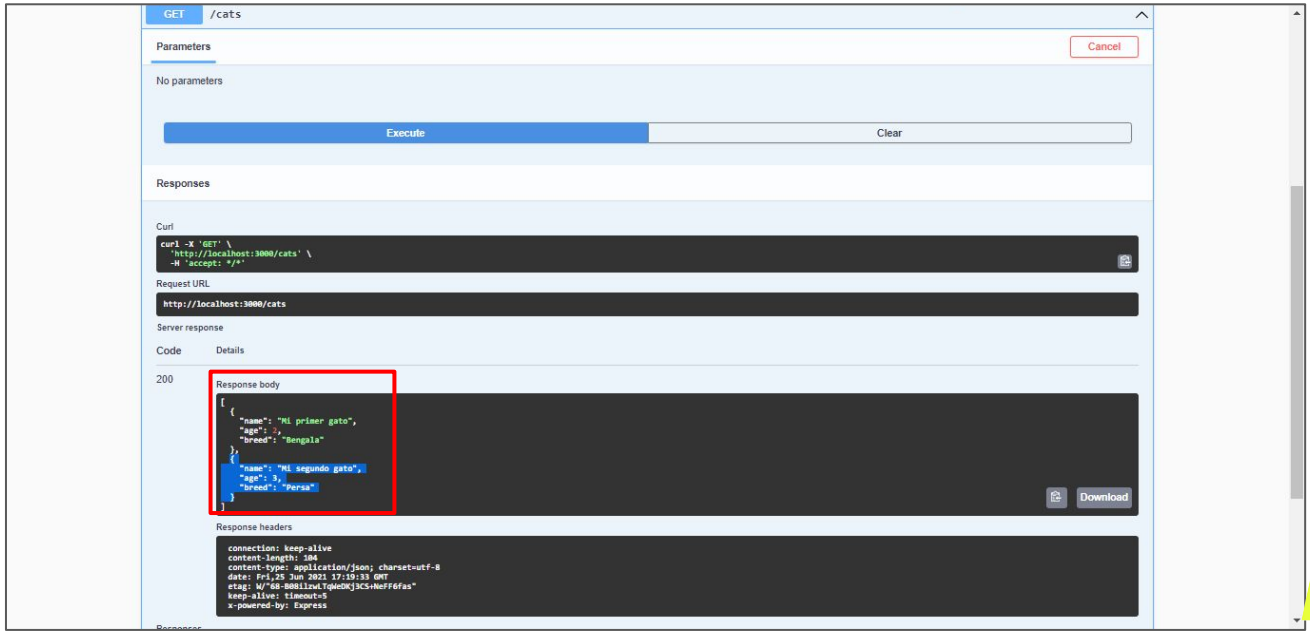
- Podemos agregar otro gato con el endpoint por POST de “/cats”. A este debemos pasarle el JSON con los datos y lo ejecutamos.

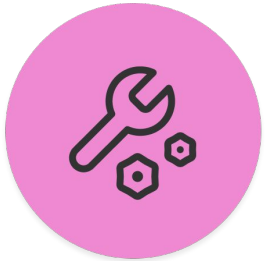




Implementación de Swagger

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





USANDO NEST

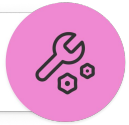
Tiempo: 15 minutos

CODER HOUSE

USANDO NEST

Tiempo: 15 minutos

Desafío generico



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.

CODER HOUSE

¿PREGUNTAS? 

¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Adonis
- Nest

OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDOLAEDUCACIÓN

CODER HOUSE