



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

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.
- Núcleo 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

[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

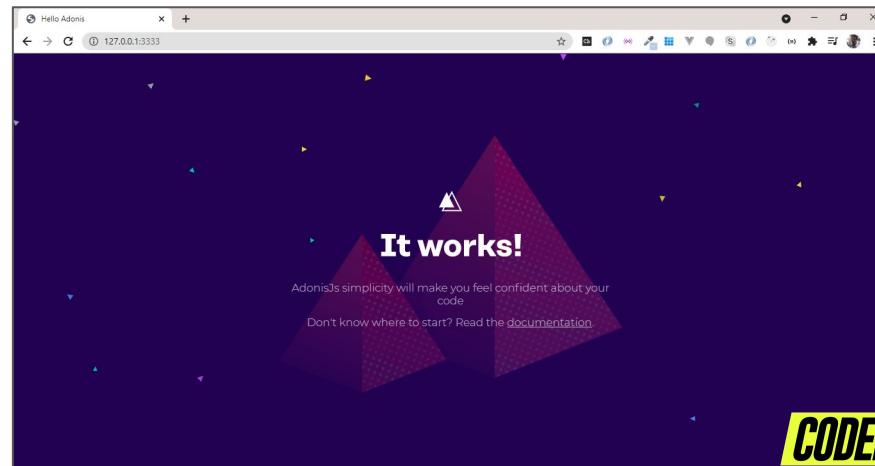
C:\Cursos\Coderhouse\CursoBackend\Clase45\Adonis
λ cd MiApp\

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:

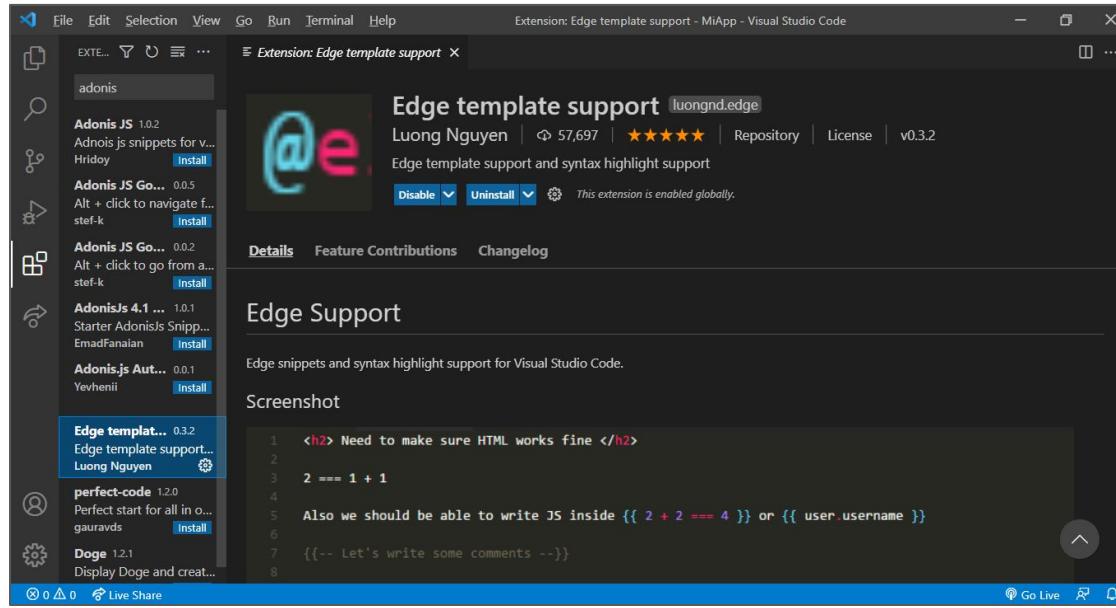
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<title>Hello Adonis</title>
{{ style('style') }}
</head>
<body>
<section>
<div class="logo"></div>
<div class="title"></div>
<div class="subtitle">
<p>Adonis.js simplicity will make you feel confident about your code</p>
<p>Don't know where to start? Read the <a href="https://adonisjs.com/docs">documentation</a>.</p>
</div>
</section>
</body>
</html>
```



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.

The screenshot shows the Visual Studio Code interface with the file `routes.js` open. The code defines a route for the root path ('/').

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

The screenshot shows a browser window at the URL `127.0.0.1:3333`. The page displays the text "¡Hola mamá, estoy programando!"



Usando Adonis - Middleware



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

```
File Edit Selection View Go Run Terminal Help
routes.js - MiApp - Visual Studio Code
EXPLORER routes.js ...
start > routes.js > ...
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('/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('/botonRojo', 'PresidentController.Apocalipsis')
29   .middleware('auth')
30
31
32
```

The screenshot shows the Visual Studio Code interface with the file "routes.js" open. A red box highlights the line ".middleware('auth')". The code defines a route for '/botonRojo' that uses the 'PresidentController.Apocalipsis' action and is protected by the 'auth' middleware.



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:

```
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

https://github.com/adonisjs/adonisjs-fullstack-tutorial
C:\Cursos\Coderhouse\CursoBackend\Clase45\Adonis\MiApp
λ |
```

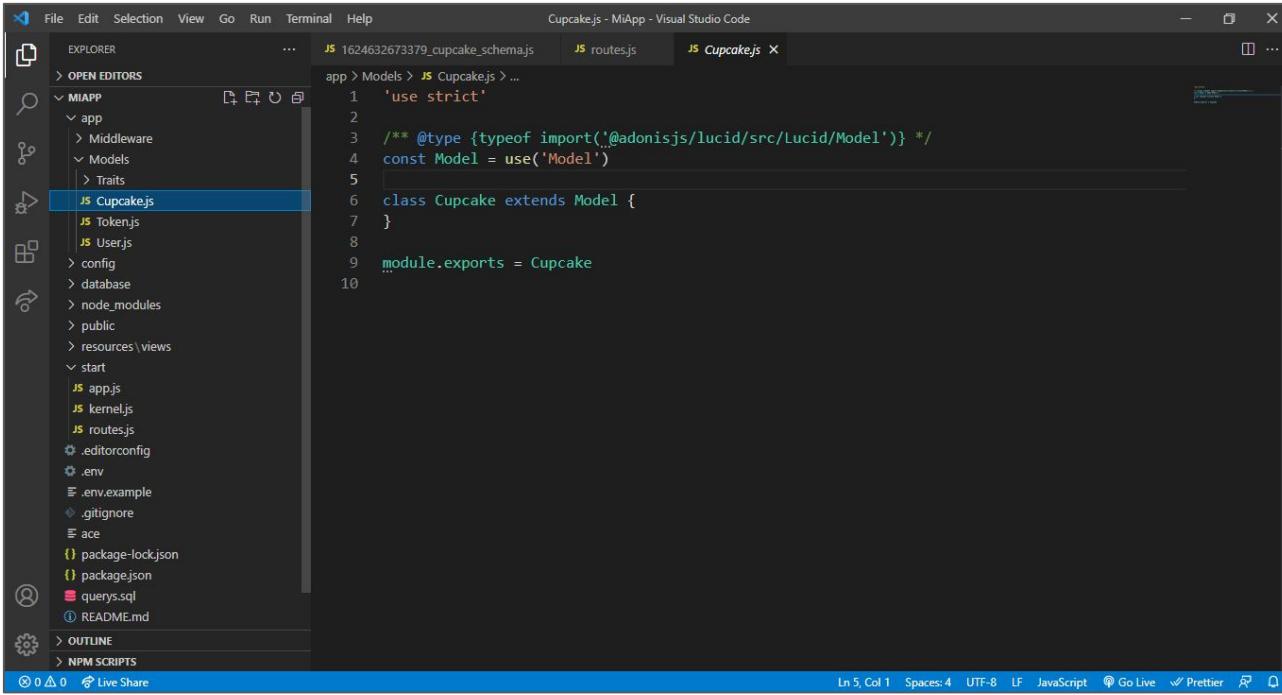
- Este comando nos creará el modelo *Cupcake.js* y el *schema* en la carpeta *migrations*.



Usando Adonis - Modelo



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



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "MIAPP". The "Models" folder is expanded, and "Cupcake.js" is selected, highlighted in blue.
- Editor:** The code for "Cupcake.js" is displayed:

```
'use strict'

/** @type {typeof import('@adonisjs/lucid/src/Lucid/Model')} */
const Model = use('Model')

class Cupcake extends Model {
}

module.exports = Cupcake
```
- Status Bar:** Shows the current file is "Cupcake.js - MiApp - Visual Studio Code", line 5, column 1, with 4 spaces, using UTF-8 LF encoding, and the language is set to JavaScript.
- Bottom Bar:** Includes icons for Live Share, Go Live, Prettier, and other development tools.



Usando Adonis - Modelos y migraciones



- Es esquema que se genera tiene el siguiente código:

The screenshot shows a Visual Studio Code interface with the title bar "1624632673379_cupcake_schema.js - MiApp - Visual Studio Code". The left sidebar shows a project structure for "MIAPP" with files like "app", "config", "database", "migrations" (containing "1503248427885_user.js" and "1503248427886_token.js"), and "resources\views" (containing "welcome.edge"). The main editor area displays the following code:

```
'use strict'

/** @type {import('@adonisjs/lucid/src/Schema')} */
const Schema = use('Schema')

class CupcakeSchema extends Schema {
  up () {
    this.create('cupcakes', (table) => {
      table.increments()
      table.string('name', 80).unique()
      table.string('description', 150)
      table.integer('price')
      table.timestamps()
    })
  }

  down () {
    this.drop('cupcakes')
  }
}

module.exports = CupcakeSchema
```

The status bar at the bottom shows "Ln 12, Col 32 (110 selected)" and other standard VS Code icons.



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).

The screenshot shows a split-screen environment. On the left, the Visual Studio Code interface displays an 'EXPLORER' sidebar with project files like 'MIAPP', 'app', 'config', 'database', 'migrations', and 'public'. A file named '1624632673379_cupcake_schema.js' is open in the main editor area, containing migration code for creating a 'cupcakes' table. On the right, the 'XAMPP Control Panel v3.2.4' window is open, showing the status of various services: Apache (running), MySQL (running, highlighted in green), FileZilla (idle), Mercury (idle), and Tomcat (idle). The MySQL service details show it's running on port 3306. The terminal pane at the bottom of the XAMPP window shows log messages related to the MySQL startup process.



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 a dark theme. The left sidebar displays a file tree for a project named 'MIAPP'. The 'EXPLORER' tab is selected, showing files like 'factory.js', 'node_modules', 'public' (containing 'logo.svg', 'pyramid.png', 'style.css', 'title.svg'), 'resources\views' (containing 'welcome.edge'), 'start' (containing 'app.js', 'kernel.js', 'routes.js'), '.editorconfig', '.env', '.env.example', '.gitignore', 'ace', 'package-lock.json', 'package.json', 'README.md', and 'server.js'. The main editor area shows the contents of the '.env' file:

```
HOST=127.0.0.1
PORT=3333
NODE_ENV=development
APP_URL=http://${HOST}:${PORT}
CACHE_VIEWS=false
APP_KEY=Nyenr0ZHM2KddoAd20L1YDixwBb92Pw9
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_USER=root
DB_PASSWORD=
DB_DATABASE=adonis
SESSION_DRIVER=cookie
HASH_DRIVER=bcrypt
```

At the bottom, the status bar shows: Ln 7, Col 1 (19 selected) Spaces: 4 UTF-8 LF Plain Text Go Live Prettier



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.

The screenshot shows two instances of MySQL Workbench. The left instance displays the 'Welcome to MySQL Workbench' screen with connection information for 'Mi Conexión MariaDB'. The right instance shows the 'query' editor with the following SQL commands:

```
1 • create database adonis;
2 • use adonis;
3 |
```

The 'Output' pane at the bottom shows the results of the executed commands:

Action	Time	Message	Duration / Fetch
create database adonia	1 11:58:13	1 row(s) affected	0.000 sec
use adonia	2 11:58:22	0 row(s) affected	0.000 sec



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*.

```
MySQL Workbench - Mi Conexión MariaDB - Warni... ×
File Edit View Query Database Server Tools Scripting Help
Navigator
SCHEMAS
Tables
adonis
cupcakes
Columns
id
name
description
price
created_at
updated_at
Indexes
Foreign Keys
Triggers
tokens
users
Views
Stored Procedures
Functions
Schemas
Administration
Information
Schema: ecommerce
Result Grid | Filter Rows: [ ] Export: [ ] Wrap Cell Content: [ ]
Field Type Null Key Default Extra
id int(10) unsigned NO PRI auto_increment
name varchar(80)
description varchar(150)
price int(11)
created_at datetime
updated_at datetime
Result 1 ×
Output
Action Output
Time Action Message Duration / Fetch
1 11:58:13 create database adonis 1 row(s) affected 0.000 sec
2 11:58:22 use adonis 0 row(s) affected 0.000 sec
3 12:04:16 desc cupcakes 6 row(s) returned 0.000 sec / 0.000 sec
Object Info Session
```

```
MySQL Workbench - Mi Conexión MariaDB - Warni... ×
File Edit View Query Database Server Tools Scripting Help
Navigator
SCHEMAS
Tables
adonis
cupcakes
Columns
id
name
description
price
created_at
updated_at
Indexes
Foreign Keys
Triggers
tokens
users
Views
Stored Procedures
Functions
Schemas
Administration
Information
Schema: ecommerce
Result Grid | Filter Rows: [ ] Export: [ ] Wrap Cell Content: [ ]
id name description price created_at updated_at
1 Coco Torta de Coco 1234 2023-09-14 11:58:22 2023-09-14 11:58:22
2 Frutilla Torta de Frutilla 2345 2023-09-14 11:58:22 2023-09-14 11:58:22
Result 2 ×
Output
Action Output
Time Action Message Duration / Fetch
4 12:12:25 insert into cupcakes (name,description,price) values ("Coco","Torta de Coco",1234) 1 row(s) affected 0.015 sec
5 12:13:04 insert into cupcakes (name,description,price) values ("Frutilla","Torta de Frutilla",2345) 1 row(s) affected 0.000 sec
6 12:13:59 select * from cupcakes LIMIT 0, 1000 2 row(s) returned 0.000 sec / 0.000 sec
Object Info Session
```



Usando Adonis - Modelo



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

```
File Edit Selection View Go Run Terminal Help
routes.js - MiApp - Visual Studio Code
OPEN EDITORS
> MIAPP
  > app
    > Middleware
    > Models
    > Traits
      JS Cupcakejs
    JS Tokenjs
    JS Userjs
  > config
  > database
  > node_modules
  > public
  > resources\views
  > start
    JS app.js
    JS kernel.js
  JS routes.js
.editorconfig
.env
.env.example
.gitignore
.ace
package-lock.json
package.json
queries.sql
README.md
OUTLINE
NPM SCRIPTS
0 0 Live Share
JS routes.js
JS Cupcakejs
start > JS routes.js > ...
15
16 /**
17  * @type {typeof import('@adonisjs/framework/src/Route/Manager')}
18 */
19 const Route = use('Route')
20
21 //Route.on('/').render('welcome')
22
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 const Cupcake = use('App\Models\Cupcake')
32
33 Route
34   .get('cupcakes', async () => {
35     return await Cupcake.all()
36   })
Ln 35, Col 7 (128 selected) Spaces:4 UTF-8 CRLF JavaScript Go Live Prettier
```



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.

```
[{"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:

```
File Edit Selection View Go Run Terminal Help routes.js - MiApp - Visual Studio Code routes.js 162463267379_cupcake_schemas.js routes.js ListaCupcakes.edge Cupcake.js routes.js

start > JS 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('/botonRojo', '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
36     return view.render('ListaCupcakes', {cupcakes})
37   })
38
```

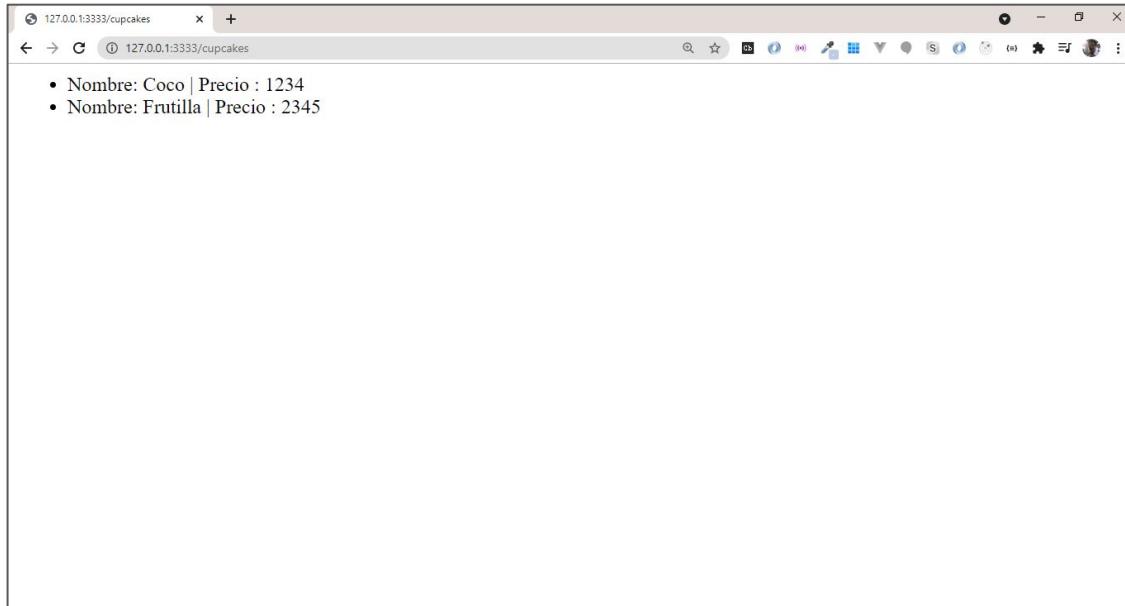
```
File Edit Selection View Go Run Terminal Help ListaCupcakes.edge - MiApp - Visual Studio Code resources > views > ListaCupcakes.edge > ul
resources > views > ListaCupcakes.edge > ul
1 <ul>
2   @each(cupcake in cupcakes)
3     <li>Nombre: {{ cupcake.name }} | Precio : {{ cupcake.price }}</li>
4   @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 - 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.

CODER HOUSE



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

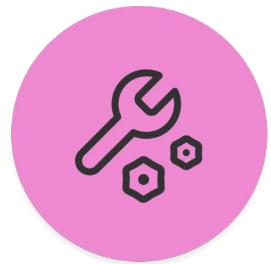
```
// app/Controllers/Http/GetCupcakesController.js
'use strict'

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

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

- 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.

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

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

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
```

The screenshot shows a terminal window titled 'Cmder' running on Windows. The command 'nest new cats-app' is being executed. The output shows the creation of various files and folders, including .eslintrc.js, .prettierrc, nest-cli.json, package.json, README.md, tsconfig.json, tsconfig.app.json, tsconfig.spec.json, app.controller.ts, app.module.ts, app.service.ts, main.ts, and test/app.e2e-spec.ts, test/jest-e2e.json. A question about the package manager is asked, followed by a success message indicating the project was created and providing start commands (cd cats-app, npm run start). A note about donations and a link to the open collective are also present.

```
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 (1976 bytes)
CREATE cats-app/README.md (3339 bytes)
CREATE cats-app/tsconfig.json (97 bytes)
CREATE cats-app/tsconfig.app.json (97 bytes)
CREATE cats-app/tsconfig.spec.json (97 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 (288 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`.

```
λ Cmder
C:\Cursos\Coderhouse\CursoBackend\Clase45\Nest
λ cd cats-app\

C:\Cursos\Coderhouse\CursoBackend\Clase45\Nest\cats-app (master)
λ npm run start:dev
npm 2 - g /usr/local/lib/node_modules/npm/lib/utils/
```

```
λ Cmder
[13:20:37] Starting compilation in watch mode...
[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()`.

```
File Edit Selection View Go Run Terminal Help
main.ts - cats-app - Visual Studio Code
EXPLORER OPEN EDITORS
CATS APP
> dist
> node_modules
src
  TS app.controller.spec.ts
  TS app.controller.ts
  TS app.module.ts
  TS app.service.ts
TS main.ts U
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 }
bootstrap();
```

Ln 1, Col 1 | Spaces: 2 | UTF-8 | LF | TypeScript | Go Live | 4.3.2 | Prettier | Live Share

```
File Edit Selection View Go Run Terminal Help
app.service.ts - cats-app - Visual Studio Code
EXPLORER OPEN EDITORS
CATS APP
> dist
> node_modules
src
  TS app.controller.spec.ts
  TS app.controller.ts
  TS app.module.ts
  TS app.service.ts U
TS app.service.ts U
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 }
```

Ln 1, Col 2 | Spaces: 2 | UTF-8 | LF | TypeScript | Go Live | 4.3.2 | Prettier | Live Share

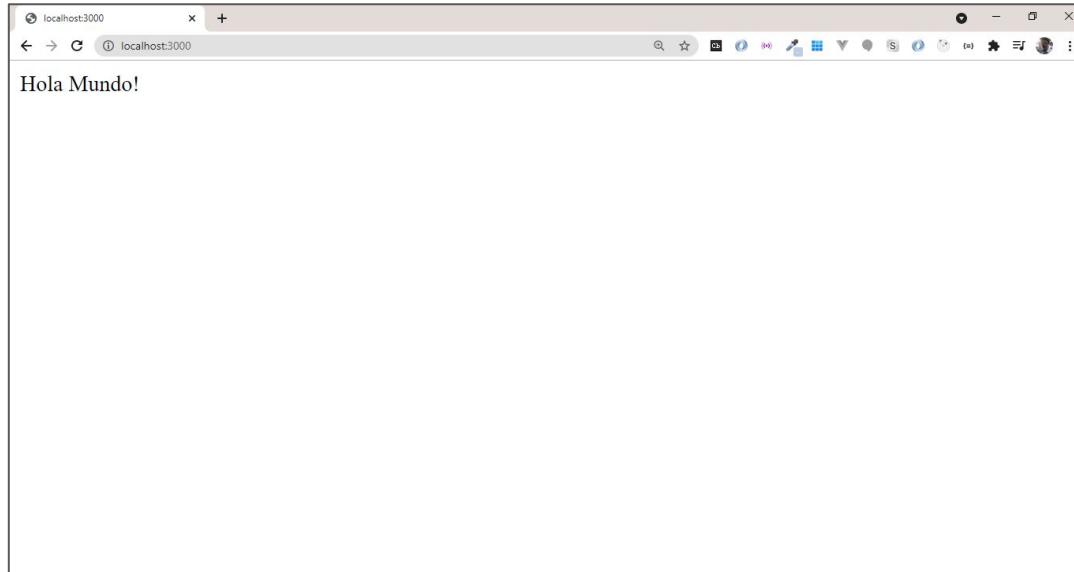
Para quienes la conoczan, verán que la estructura de carpetas es casi igual a la se crea en Angular



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 |
```

The screenshot shows a terminal window titled "Cmder" with a dark theme. It displays three separate command-line sessions, each starting with the command "nest generate". In the first session, "nest generate module cats" is run, creating a new file "cats.module.ts" and updating "app.module.ts". In the second session, "nest generate controller cats" is run, creating "cats.controller.spec.ts" and "cats.controller.ts" files, and updating "cats.module.ts". In the third session, "nest generate service cats" is run, creating "cats.service.spec.ts" and "cats.service.ts" files, and updating "cats.module.ts". The terminal window has a title bar, a menu bar, and a status bar at the bottom showing "node.exe ["] cmd.exe cmd.exe".



Crear módulo, controlador y servicio



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

The screenshot shows three instances of Visual Studio Code side-by-side, each displaying a different file from a Nest.js application structure:

- cats.module.ts**:

```
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 {}
```
- cats.controller.ts**:

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

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

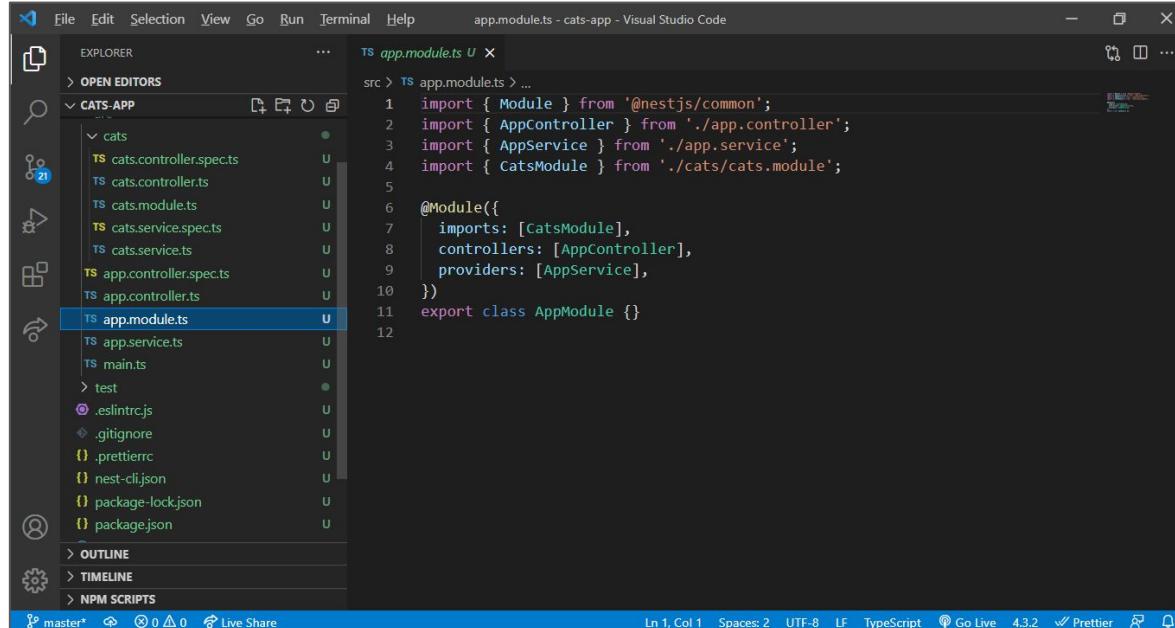
The application structure visible in the Explorer pane includes: CATS-APP, cats, TS cats.controller.specs, TS cats.controller.ts, TS cats.module.ts, TS cats.service.ts, TS app.controller.specs, TS app.controller.ts, TS app.module.ts, TS app.service.ts, TS main.ts, > test, .eslintrc.js, .gitignore, .prettierrc, nest-cli.json, package-lock.json, package.json.



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:



The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** app.module.ts - cats-app - Visual Studio Code.
- Explorer Panel:** Shows the project structure under 'CATS-APP'. The 'cats' folder contains several files: cats.controller.spec.ts, cats.controller.ts, cats.module.ts, cats.service.spec.ts, cats.service.ts, app.controller.spec.ts, and app.controller.ts. The 'app.module.ts' file is currently selected and highlighted in blue.
- Code Editor:** Displays the content of the app.module.ts file:

```
src > TS app.module.ts > ...
1 import { Module } from '@nestjs/common';
2 import { AppController } 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 {}
```
- Bottom Status Bar:** master*, Live Share, Ln 1, Col 1, Spaces: 2, UTF-8, LF, TypeScript, Go Live, 43.2, Prettier.



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:

```
File Edit Selection View Go Run Terminal Help cat.interface.ts - cats-app - Visual Studio Code

EXPLORER
> OPEN EDITORS
  ✓ CATS-APP
    ✓ cats
      TS cat.controller.spec.ts
      TS cats.controllers.ts
      TS cats.module.ts
      TS cats.service.specs
      TS cats.service.ts
    < interfaces
      TS cat.interface.ts
    > test
      TS .eslintrc.js
      TS .gitignore
      TS prettierc
      TS nest-cli.json
  > OUTLINE
  > TIMELINE
  > NPM SCRIPTS

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

L 1 2 3 4 5 6 7
```

The screenshot shows the Visual Studio Code interface with the 'cat.interface.ts' file open. The code defines an interface 'Cat' with three readonly properties: 'name', 'age', and 'breed'. The 'interfaces' folder is expanded in the Explorer sidebar.

```
File Edit Selection View Go Run Terminal Help create-cat.dto.ts - cats-app - Visual Studio Code

EXPLORER
> OPEN EDITORS
  ✓ CATS-APP
    ✓ cats
      TS cat.controller.spec.ts
      TS cats.controllers.ts
      TS cats.module.ts
      TS cats.service.specs
      TS cats.service.ts
    < dto
      TS create-cat.dto.ts
    > interfaces
      TS cat.interface.ts
      TS app.controller.spec.ts
      TS app.controller.ts
      TS app.module.ts
      TS app.service.ts
    > test
      TS .eslintrc.js
      TS .gitignore
      TS prettierc
      TS nest-cli.json
  > OUTLINE
  > TIMELINE
  > NPM SCRIPTS

TS create-cat.dto.ts U
src > dto > TS create-cat.dto.ts > CreateCatDto
1  export class CreateCatDto {
2    ...
3    ...
4    ...
5    ...
6    ...
7  }

L 1 2 3 4 5 6 7
```

The screenshot shows the Visual Studio Code interface with the 'create-cat.dto.ts' file open. The code defines a class 'CreateCatDto' with five readonly properties: 'name', 'age', 'breed', 'category', and 'size'. The 'dto' folder is expanded in the Explorer sidebar.



Implementación de un servicio



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "CATS-APP". The file "cats.service.ts" is selected.
- Code Editor:** Displays the code for "cats.service.ts":

```
src > cats > TS cats.service.ts > ...
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 }
```

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.



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.

```
File Edit Selection View Go Run Terminal Help
cats.controller.ts - cats-app - Visual Studio Code
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 catService: CatsService) {}
9
10   @Post()
11   async create(@Body() createCatDto: CreateCatDto) {
12     this.catService.create(createCatDto);
13   }
14
15   @Get()
16   async findAll(): Promise<Cat[]> {
17     return this.catService.findAll();
18   }
19 }
20
master* ⚡ ⚡ 0 △ 0 Live Share
Ln 6. Col 20 Spaces: 4 UTF-8 LF TypeScript Go Live 4.3.2 Prettier
```



Implementación de un controlador



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

```
File Edit Selection View Go Run Terminal Help
cats.controller.ts - cats-app - Visual Studio Code
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 }
```

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`.

The screenshot shows the Postman application window. In the top navigation bar, 'File', 'Edit', 'View', 'Help' are visible. Below the bar, there are tabs for 'Home', 'Workspaces', 'Reports', and 'Explore'. A search bar says 'Search Postman'. On the left sidebar, 'My Workspace' is selected, showing 'Collections' (with '+ New' and 'Import' buttons), 'APIs' (with a 'GET localhost:3000/' entry), 'Environments', 'Mock Servers', 'Monitors', and 'History'. The main workspace shows a 'Nest / localhost:3000/' collection. Under it, a 'GET localhost:3000/' request is listed. This request has 'Params', 'Authorization', 'Headers (0)', 'Body', 'Pre-request Script', 'Tests', and 'Settings' tabs. The 'Body' tab is selected and contains a table with one row: KEY 'Key', VALUE 'Value', and DESCRIPTION 'Description'. Below the table are 'Cookies' and 'Headers' tabs. The 'Body' tab also has 'Pretty' and 'Raw' options. At the bottom of the request card, there are buttons for 'Send' and 'Save'. The status bar at the bottom shows 'Status: 200 OK Time: 31 ms Size: 238 B Save Response'. The bottom right corner of the status bar has a trash bin icon. The footer of the Postman window includes 'Find and Replace', 'Console', and icons for 'Bootcamp', 'Runner', 'Trash', and a help icon.



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.

The screenshot shows the Postman application window. In the center, there's a request card for a GET method to the URL `localhost:3000/cats`. Below the URL, under the 'Params' tab, there's a table with one row: 'Key' (Value) and 'Value' (Description). At the bottom of the screen, the 'Body' section shows a JSON response with the number '1' highlighted by a red box. The status bar at the bottom indicates a successful 200 OK response with a size of 235 B.



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.

The screenshot shows the Postman application interface. On the left, the sidebar includes 'My Workspace' (Collections, APIs, Environments, Mock Servers, Monitors, History), 'File', 'Edit', 'View', 'Help', and a search bar. The main area shows a 'Nest' collection with a single 'POST localhost:3000' request. The 'Body' tab is selected, showing the JSON payload:

```
{  
  "name": "Mi primer gato",  
  "age": 2,  
  "breed": "Bengala"  
}
```

The 'Body' tab also displays the response status: 'Status: 201 Created Time: 9 ms Size: 150 B'. At the bottom, there are tabs for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'Text'. The footer includes 'Find and Replace', 'Console', and navigation buttons for 'Bootcamp', 'Runner', 'Trash'.



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).

The screenshot shows the Postman application interface. On the left, there's a sidebar with icons for Collections, APIs, Environments, Mock Servers, Monitors, and History. The main workspace shows a 'Nest / localhost:3000' collection. A specific request is selected: a 'GET' request to 'localhost:3000/cats'. The 'Body' tab is active, showing a JSON response:

```
[{"name": "Mi primer gato", "age": 2, "breed": "Sengala"}]
```

The status bar at the bottom indicates 'Status: 200 OK'.

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)
$ npm install --save @nestjs/swagger swagger-ui-express
npm WARN @nestjs/mapped-types@4.1 requires a peer of class-transformer@^0.2.0 || ^0.3.0 || ^0.4.0 but none is installed. You must install peer dependencies yourself.
npm WARN @nestjs/mapped-types@4.1 requires a peer of class-validator@^0.11.1 || ^0.12.0 || ^0.13.0 but none is installed. You must install 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)
x |
```



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”.

```
File Edit Selection View Go Run Terminal Help
main.ts - cats-app - Visual Studio Code
EXPLORER
OPEN EDITORS
CATS-APP
  > cats
    TS cats.controller.specs.ts
    TS cats.controller.ts
    TS cats.module.ts
    TS cats.service.specs.ts
    TS cats.service.ts
    TS create-cat.dto.ts
  > dto
    TS create-cat.dto.ts
  > interfaces
    TS cat.interface.ts
  > test
    TS app.controller.specs.ts
    TS app.controller.ts
    TS app.module.ts
    TS app.service.ts
    TS main.ts
    .eslintrc.js
    .gitignore
  > OUTLINE
  > TIMELINE
  > NPM SCRIPTS
src > TS main.ts > bootstrap
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();
```

Ln 15, Col 1 Spaces: 2 UTF-8 LF TypeScript Go Live 4.3.2 Prettier



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.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "CATS-APP". The "cats" folder contains files like "controller.spec.ts", "controller.ts", "module.ts", "service.spec.ts", "service.ts", and "main.ts". The "dto" folder contains "create-cat.dto.ts", which is currently selected and highlighted in blue. Other files in "dto" include "interfaces", "cat.interface.ts", "controller.spec.ts", "controller.ts", "module.ts", "service.ts", and "main.ts".
- Code Editor:** Displays the content of "create-cat.dto.ts". The code defines a class "CreateCatDto" with three properties: "name" (string), "age" (number), and "breed" (string), each annotated with `@ApiProperty()`.
- Bottom Status Bar:** Shows the current file is "create-cat.dto.ts" in a "cats-app" workspace, with line 11, column 5, 4 spaces, and using UTF-8 encoding. It also indicates the file is a TypeScript file.
- Bottom Right Logo:** A yellow "CODER HOUSE" logo.

```
import { ApiProperty } from '@nestjs/swagger';

export class CreateCatDto {
    @ApiProperty()
    readonly name: string;

    @ApiProperty()
    readonly age: number;

    @ApiProperty()
    readonly breed: string;
}
```



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>.

The screenshot shows the Swagger UI interface for a 'Cats example' API. At the top, it says 'The cats API description'. Below that, there's a section for 'cats' which contains a 'default' endpoint. Under 'default', there are three methods listed: a 'GET' method for '/' (which is collapsed), a 'POST' method for '/cats' (which is expanded, showing its schema), and another 'GET' method for '/cats' (which is collapsed). At the bottom, there's a 'Schemas' section containing a 'CreateCatDto' entry.



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”.

The screenshot shows the Swagger UI interface for a 'cats' API. At the top, there's a navigation bar with 'cats' and 'default'. Below it, the 'GET /' endpoint is highlighted with a blue bar. The 'Parameters' section shows 'No parameters'. The 'Responses' section lists a single entry for 'Code 200'. The 'Links' section shows 'No links'. At the bottom of the main panel, there are two more endpoints: 'POST /cats' (highlighted with a green bar) and 'GET /cats'. The 'Schemas' section at the bottom contains a 'CreateCatDto' entry. On the right side of the main panel, there's a vertical sidebar with collapse/expand arrows. A red box highlights the 'Try it out' button in the 'Responses' section of the main panel.



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”.

The image shows two screenshots of the Swagger UI interface. On the left, the 'cats' API is displayed with a 'GET /' endpoint. A red box highlights the 'Execute' button. On the right, the execution details for the '/cats' endpoint are shown. A red box highlights the 'Hola Mundo!' response body. Both screenshots include a 'Cancel' button in the top right corner.

Cats API Overview:

- Default:** GET /
 - No parameters
 - Responses
 - Code: 200
- POST /cats**
- GET /cats**

Execution Details:

GET /

Parameters: No parameters

Responses:

Code: 200 Details: Response body: Hola Mundo!

Request URL: curl -X 'GET' 'http://localhost:3000/' \ -H 'accept: */*' Request Headers: Content-Length: 13 Content-Type: text/html; charset=UTF-8 Date: Fri, 28 Jun 2021 17:14:26 GMT Server: Apache/2.4.41 (Ubuntu) X-Powered-By: Express



Implementación de Swagger



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

The screenshot shows the Postman application interface. At the top, there's a green header bar with the text "POST /cats". Below it, a modal window is open for the "GET /cats" operation. The modal includes fields for "Parameters" (set to "No parameters") and "Responses". Under "Responses", there's a "Curl" section with a command line, a "Request URL" field containing "http://localhost:3000/cats", and a "Server response" section. The "Server response" section shows a status code of 200. The "Response body" field contains the following JSON data, which is highlighted with a red rectangle:

```
[{"name": "Mi primer gato", "age": 2, "breed": "Kangala"}]
```

Below the response body, the "Response headers" section displays the following information:

```
content-length: 53
Content-Type: application/json; charset=utf-8
date: Fri, 25 Jun 2021 17:15:18 GMT
etag: W/"35-0BMsPcNgGaudxHfIgTHAU"
x-powered-by: Express
```

At the bottom of the modal, there are "Code" and "Description" fields, and a "Links" section indicating "No links".



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.

The screenshot shows the Swagger UI interface for a POST request to the '/cats' endpoint. The 'Request body' field is highlighted with a red box and contains the following JSON:

```
{"name": "Mi segundo gato", "age": 2, "breed": "Persa"}
```

Below the request body, there is a large empty text area for the response. At the bottom of the interface, there is a prominent blue 'Execute' button.

The screenshot shows the results of executing the POST request to '/cats'. The 'Responses' section displays the following information:

- Raw:**

```
curl -X POST 'http://127.0.0.1:5000/cats' -H 'Content-Type: application/json' -d '{"name": "Mi segundo gato", "age": 2, "breed": "Persa"}'
```
- HTTP Headers:**

```
HTTP/1.1 201 Created
```
- Code:** 201
- Description:** Response headers
- Raw Response Body:**

```
{\"id\": 2, \"name\": \"Mi segundo gato\", \"age\": 2, \"breed\": \"Persa\"}
```
- Code:** 201
- Description:** Response



Implementación de Swagger

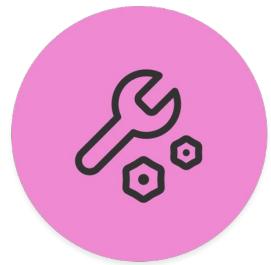
- Finalmente, si volvemos a ejecutar el endpoint “/cats” por GET, obtendremos la información de los dos gatos que tenemos creados.

The screenshot shows the Swagger UI interface for a 'GET /cats' endpoint. The 'Responses' section displays a successful 200 status code response. The 'Response body' field contains the following JSON data, which is highlighted with a red box:

```
{  
  "name": "Mi primer gato",  
  "age": 3,  
  "breed": "Bengala"  
,  
  "name": "Mi segundo gato",  
  "age": 3,  
  "breed": "Persa"  
}
```

Below the response body, the 'Response headers' section shows the following metadata:

```
connection: keep-alive  
content-length: 184  
content-type: application/json; charset=utf-8  
date: Mon, 17 Oct 2016 17:19:33 GMT  
etag: W/"68-BE8811zwLQgIeKj3C5-NeFF6fas"  
keep-alive: timeout=5  
x-powered-by: Express
```



USANDO NEST

Tiempo: 15 minutos



USANDO NEST

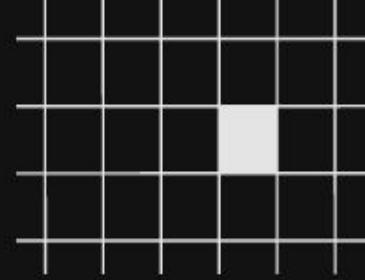
Tiempo: 15 minutos

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?

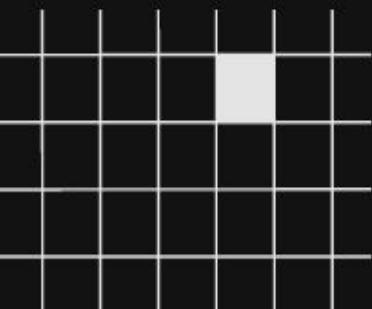




¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Adonis
- Nest





OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDOLAEDUCACIÓN