

# **Anexo: FP en Desarrollo de Aplicaciones Web Despliegue de Aplicaciones Web**

---

Creación de una API desarrollada con NestJs y  
Mongo DB y Agregados de Mongo DB

**Carla Adell Michavila**

**14/12/2020**

## Contenido

Instalaciones necesarias.....	2
Creación del proyecto en Nest JS.....	3
Para crear los componentes:.....	5
Comprobación del funcionamiento.....	18
Importar base de datos a Mongo DB .....	21
Agregados de Mongo DB.....	24

## Instalaciones necesarias

- Para instalar **Node JS** [ir aquí](#).
- Para instalar **Nest JS** hace falta ir a la consola del ordenador y ejecutar la orden:

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

*Si quieres saber más sobre NestJS puedes [ir aquí](#).*

- Para instalar **Mongo DB** [ir aquí](#).
- Para instalar **Postman** [ir aquí](#).

## Creación del proyecto en Nest JS

En la terminal del ordenador, nos colocaremos en el directorio en el que queramos crear el proyecto y ejecutaremos la orden:

```
nest new tiendaAnimales
```

A continuación, nos aparecerá la pregunta: de con que paquete queremos trabajar. Elegiremos la opción **npm**, ya que vamos a trabajar con dicha librería de paquetes.

```
> Which package manager would you like to use? (Use arrow keys)  
> npm  
  yarn
```

Después abriremos el proyecto:

```
cd tienda-animales
```

Ya cuando estemos dentro del proyecto, instalaremos los paquetes **npm** necesarios para poder trabajar con MongoDB.

```
npm i mongoose  
npm i class-validator  
npm i @nestjs/mongoose  
npm i @types/mongoose
```

Antes de crear los componentes del proyecto tenemos que tener claro que datos tenemos que almacenar.

En nuestro caso queremos almacenar los siguientes datos para cada tipo de animal:

- Información del animal
  - Nombre
  - Edad
  - Raza
  - Pedigree
  - Fecha de creación de la información
  - Fecha de actualización de la información
  - seguimiento
- Información de los propietarios
  - Primer nombre
  - Segundo nombre
  - Primer apellido

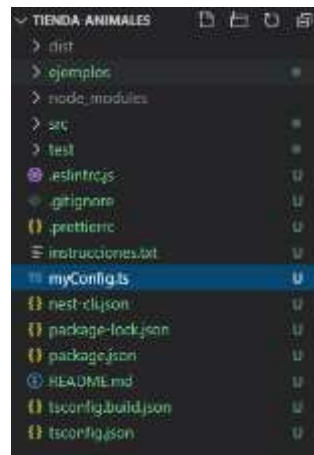
- Segundo apellido
- Edad
- La dirección:
  - Tipo vía
  - Nombre vía
  - Numero
  - CP
  - Localidad
  - Municipio
- Los teléfonos:
  - Móvil
  - Fijo

La aplicación web va a tener una organización de tres módulos en un principio, **database**, **gatos** y **perros**. En **database** es donde conectaremos nuestro backend con la base de datos deseada, en este caso MongoDB.

## Para crear los componentes:

```
nest g mo database
nest g pr database/databaseProviders
```

1. Creamos el archivo **myConfig.ts**, en la raíz del proyecto con el editor que utilizemos:



2. En **myConfig.ts**:

```
1 export const myConfig = {
2   //Puerto
3   mongoServer: 'localhost',
4   //Nombre de nuestra BBDD
5   dataBase: 'tiendaAnimales',
6 }
```

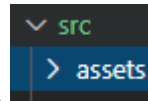
3. En **database-providers.ts**

```
1 import * as mongoose from 'mongoose';
2 import { myConfig } from '../myConfig';
3
4 export const databaseProviders = [
5   {
6     provide: 'DATABASE_CONNECTION',
7     useFactory: (): Promise<typeof mongoose> =>
8       //Se almacenara en el servidor y bbdd seleccionado
9       mongoose.connect(`mongodb://${myConfig.mongoServer}/${myConfig.dataBase}`, {
10        useNewUrlParser: true, useUnifiedTopology: true, useCreateIndex: true,
11        useFindAndModify: false
12      })
13   }
14 ];
```

- **Moongose**, es la clase que se necesita para poder enviar los paquetes a MongoDB.
- `{useNewUrlParser: true, useUnifiedTopology: true, useFindAndModify: false} =>`  
Para quitar unos warning.

4. En **database.module.ts**

```
1 import { Module } from '@nestjs/common';
2 import { databaseProviders } from './database-providers';
3
4 @Module({
5   providers: [...databaseProviders],
6   exports: [...databaseProviders]
7 })
8 export class DatabaseModule { }
```



5. Crear la carpeta **assets** en nuestro proyecto.
6. Crearemos en componente **IReturn**

```
nest g interface assets/interfaces/i-retun
```

#### 7. En **assets/interfaces/i-retun-interface.ts**

```
1 export interface IRetun {
2   readonly msg: string;
3   readonly status: number;
4   readonly data: any;
5   readonly code: string;
6   readonly validRequest: boolean;
7 }
```

- Esta es la estructura del mensaje que devolveremos.
- Las **Interfaces** es donde se declara la estructura de los Objetos. A las interfaces que creamos los podremos una “**I**” delante para indicar que estamos trabajando con interfaces.
- El atributo **readonly**, para que no puedas sobrescribir la información que llega del **Fronted**.

El siguiente módulo **Gatos**, gestionará toda la información que queremos almacenar de dicha especie.

```
nest g mo gatos

nest g cl gatos/dto/createGatoDTO
nest g cl gatos/dto/updateGatoDTO
nest g interface gatos/interfaces/i-gato
nest g cl gatos/schemas/gatoSchema
nest g pr gatos/providers/gato
nest g s gatos/services/gato
nest g co gatos/api/gato
```

Primero crearemos los **DTO**, estos son la información que le enviaremos al **Fronted**.

#### 8. En **dto/create-gato-dto.ts**

```
1 export class CreateGatoDto {
2   nombre: string;
3   edad: number;
4   raza: string;
5   pedigree: boolean;
6   created_at : Date;
7   updated_at :Date;
8   seguimiento: boolean;
9   propietario:{
10    primer_nombre: string;
11    segundo_nombre: string
12    primer_apellido: string;
13    segundo_apellido: string;
14    edad: number;
15    direccion: {
```

```

16         tipo_via: string;
17         nombre_via: string;
18         numero: number;
19         cp: number;
20         localidad: string;
21         municipio: string;
22     };
23     telefonos: {
24         movil: number;
25         fijo: number;
26     }
27 }
28 }

```

## 9. En `dto/update-gato-dto.ts`

```

1  export class UpdateGatoDto {
2      _id: string;
3      nombre: string;
4      edad: number;
5      raza: string;
6      pedigree: boolean;
7      created_at : Date;
8      updated_at :Date;
9      seguimiento: boolean;
10     propietario:{
11         primer_nombre: string;
12         segundo_nombre: string
13         primer_apellido: string;
14         segundo_apellido: string;
15         edad: number;
16         direccion: {
17             tipo_via: string;
18             nombre_via: string;
19             numero: number;
20             cp: number;
21             localidad: string;
22             municipio: string;
23         };
24         telefonos: {
25             movil: number;
26             fijo: number;
27         }
28     }
29 }

```

- En la estructura de **Update**, ponemos en **id** como opcional ya que **MongoDB** por defecto le añade su propio **id**, en este caso un **string**.

Luego creamos la **Interfaz**, como ya hemos comentado antes, aquí declaramos la estructura del Objeto que queremos crear.

## 10. En `interfaces/i-gato.interfaces.ts`

```

1  import { Document } from 'mongoose';
2
3  export interface IGato extends Document {
4      readonly nombre: string;
5      readonly edad: number;
6      readonly raza: string;
7      readonly pedigree: boolean;
8      readonly created_at : Date,
9      readonly updated_at :Date,
10     readonly seguimiento: boolean;
11     propietario:{
12         readonly primer_nombre: string;
13         readonly segundo_nombre: string
14         readonly primer_apellido: string;

```



```

15     readonly segundo_apellido: string;
16     readonly edad: number;
17     direccion: {
18         readonly tipo_via: string;
19         readonly nombre_via: string;
20         readonly numero: number;
21         readonly cp: number;
22         readonly localidad: string;
23         readonly municipio: string;
24     };
25     telefonos: {
26         readonly movil: number;
27         readonly fijo: number;
28     }
29 }
30 }

```

Después creamos los **Schemas**, que son los que se encargan de mandarle la información a MongoDB, se le mandan los Objetos.

### 11. En **schemas/gato-schema.ts**

```

1  import * as mongoose from 'mongoose';
2
3  export const GatoSchema = new mongoose.Schema ({
4      nombre: { type: String, required: true },
5      edad: { type: Number, required: true },
6      raza: { type: String, default: 'Desconocido' },
7      seguimiento: { type: Boolean, default: true },
8      created_at : { type: Date, default: Date.now },
9      updated_at : { type: Date , default:null},
10     propietario:{
11         primer_nombre: { type: String},
12         segundo_nombre: { type: String},
13         primer_apellido: { type: String},
14         segundo_apellido: { type: String},
15         edad: { type: Number},
16         email: { type: String},
17         direccion: {
18             tipo_via: { type: String},
19             nombre_via: { type: String},
20             numero: { type: Number},
21             cp: { type: Number},
22             localidad: { type: String },
23             municipio: { type: String},
24         },
25         telefonos: {
26             movil: { type: Number},
27             fijo: { type: Number}
28         }
29     }
30 });

```

En **providers** indicaremos el nombre que le queremos dar a la colección, y lo que nos permite trabajar con diferentes colecciones en nuestra base de datos.

### 12. En **providers/ gato.ts**

```

1  import { Connection } from 'mongoose';
2  import { GatoSchema } from '../schemas/gato-schema';
3
4  export const gatoProviders = [
5      {
6          provide: 'GATO_MODEL',
7          useFactory: (connection: Connection) =>
8              connection.model('Gato', GatoSchema),
9          inject: ['DATABASE_CONNECTION'],
10     },
11 ];

```

En **services** crearemos las funciones que necesitamos, en este caso crearemos el **CRUD**, *Create, Read, Update y Delete*.

### 13. En **services/ gato.service.ts** => las funciones **CRUD**

```

1  import { Inject, Injectable } from '@nestjs/common';
2  import { Model } from 'mongoose';
3  import { IRetun } from 'src/assets/interfaces/i-retun.interface';
4  import { CreateGatoDto } from 'src/gatos/dto/create-gato-dto';
5  import { UpdateGatoDto } from 'src/gatos/dto/update-gato-dto';
6  import { IGato } from 'src/gatos/interfaces/i-gato.interface';
7
8  @Injectable()
9  export class GatoService {
10   /*****
11    * @param IGato
12    * @returns
13    *****/
14   constructor(
15     @Inject('GATO_MODEL')
16     private readonly gatoModel: Model<IGato>,
17   ) { }
18
19   /*****
20    * @param CreateGatoDto
21    * @returns Promise<IRetun>
22    *****/
23   async create(createGatoDto: CreateGatoDto): Promise<IRetun> {
24     const existGato = await this.gatoModel.exists({ nombre: createGatoDto.nombre });
25     //Se llama a la promesa
26     const myPromise = new Promise<IRetun>((resolve, reject) => {
27       if (!existGato) {
28         new this.gatoModel(createGatoDto).save().then(saved => {
29           resolve({ msg: 'Gato creado', status: 400, data: saved, code: '400', validRequest:
30 true});
31         });
32       }
33       else {
34         resolve({ msg: 'El nombre de gato ya existe', status: 500, data: undefined, code: '500',
35 validRequest: false });
36       }
37     });
38     return myPromise;
39   }
40
41   /*****
42    * @param
43    * @returns Promise<IRetun>
44    *****/
45   async findAll(): Promise<IRetun> {
46     const myPromise = new Promise<IRetun>((resolve, reject) => {
47       this.gatoModel.find().exec().then(r => {
48         resolve({ msg: 'Gatos:', status: 400, data: r, code: '400', validRequest: true });
49       });
50     });
51     return myPromise;
52   }
53
54   /*****
55    * @param id
56    * @returns Promise<IRetun>
57    *****/
58   async delete(id: string): Promise<IRetun> {
59     const exist = await this.gatoModel.exists({ _id: id });
60     const myPromise = new Promise<IRetun>((resolve, reject) => {
61       if (!exist) {
62         resolve({ msg: 'El gato no existe', status: 500, data: undefined, code: '500',
63 validRequest: false });
64       }
65       else {
66         this.gatoModel.deleteOne({ _id: id }).exec();
67         // this.gatoModel.updateMany
68

```

```

69         resolve({ msg: 'El gato fue eliminado', status: 400, data: id, code: '400',
70         validRequest: true });
71     }
72 });
73     return myPromise;
74 }
75
76 /*****
77  * @param UpdateGatoDto
78  * @returns Promise<IRetun>
79  *****/
80     async update(updateGatoDto: UpdateGatoDto): Promise<IRetun> {
81         if(updateGatoDto._id === undefined || updateGatoDto._id === '' || updateGatoDto._id.trim()
82         === ''){
83             return new Promise<IRetun>((resolve,reject) => {
84                 resolve({ msg: 'Falta id', status: 400, data: undefined, code: '405', validRequest:
85                 false});
86             })
87         }
88         const exist = await this.gatoModel.exists({ _id: updateGatoDto._id });
89         const myPromise = new Promise<IRetun>((resolve, reject) => {
90             if (!exist) {
91                 resolve({ msg: 'El gato no existe', status: 500, data: undefined, code: '500',
92                 validRequest: false });
93             }
94             else {
95                 updateGatoDto.updated_at = new Date;
96                 this.gatoModel.findOneAndUpdate({_id:updateGatoDto._id},updateGatoDto,{ new:
97                 true}).exec();
98                 resolve({ msg: 'El gato existe', status: 400, data: updateGatoDto, code: '400',
99                 validRequest: true });
100             }
101         });
102         return myPromise;
103     }
104 }

```

En el **controller** creamos los **endpoint** de nuestra API.

#### 14. En `api/gato/gato.controller.ts`

```

1  import { Body, Controller, Delete, Get, Param, Post } from '@nestjsjs/common';
2  import { CreateGatoDto } from 'src/gatos/dto/create-gato-dto';
3  import { UpdateGatoDto } from 'src/gatos/dto/update-gato-dto';
4  import { GatoService } from 'src/gatos/services/gato/gato.service';
5
6  //Aqui es donde indicamos la URL
7  @Controller('api/tiendaAnimal/v0/gatos')
8  export class GatoController {
9      /*****
10       * @param GatoService
11       * @returns
12       *****/
13       constructor(private gatoService: GatoService) {}
14
15       /*****
16       * @param CreateGatoDto
17       * @returns create()
18       *****/
19       @Post('create')
20       create(@Body() gatoDetalle: CreateGatoDto) {
21           //se llama a la promesa
22           return this.gatoService.create(gatoDetalle).then(r => {
23               return r;
24           });
25       }
26
27       /*****
28       * @param
29       * @returns findAll()
30       *****/
31       @Get('readAll')

```

```

32   readAll() {
33       return this.gatoService.findAll();
34   }
35
36   /*****
37    * @param id
38    * @returns delete()
39    *****/
40   @Delete('delete/:id')
41   delete(@Param('id') id: string) {
42       return this.gatoService.delete(id).then(r => {
43           return r;
44       });
45   }
46
47   /*****
48    * @param UpdateCatDto
49    * @returns update()
50    *****/
51   @Post('update')
52   update(@Body() gatoDetalle: UpdateGatoDto) {
53       return this.gatoService.update(gatoDetalle).then(r => {
54           return r;
55       });
56   }
57 }

```

## 15. En `gatos.modules.ts`

```

1   import { Module } from '@nestjs/common';
2   import { gatoProviders } from '../providers/gato';
3   import { GatoService } from '../services/gato/gato.service';
4   import { GatoController } from '../api/gato/gato.controller';
5   import { DatabaseModule } from 'src/database/database.module';
6
7   @Module({
8       imports: [DatabaseModule],
9       providers: [...gatoProviders, GatoService],
10      controllers: [GatoController]
11  })
12  export class GatosModule {}

```

Ahora crearemos el modelo de los **Perros**:

```

nest g mo perros
nest g cl perros/dto/createPerroDTO
nest g cl perros/dto/updatePerroDTO
nest g interface perros/interfaces/i-perro
nest g cl perros/schemas/perroSchema
nest g pr perros/providers/perro
nest g s perros/services/perro
nest g co perros/api/perro

```

Y seguiremos los pasos mencionados con el módulo anterior:

## 16. En `dto/create-perro-dto.ts`

```

1   export class CreatePerroDto{
2       nombre: string;
3       edad: number;
4       raza: string;
5       pedigree: boolean;
6       created_at : Date;
7       updated_at :Date;

```

```

8     seguimiento: boolean;
9     propietario:{
10         primer_nombre: string;
11         segundo_nombre: string;
12         primer_apellido: string;
13         segundo_apellido: string;
14         edad: number;
15         direccion: {
16             tipo_via: string;
17             nombre_via: string;
18             numero: number;
19             cp: number;
20             localidad: string;
21             municipio: string;
22         };
23         telefonos: {
24             movil: number;
25             fijo: number;
26         }
27     }
28 }

```

### 17. En `dto/update-perro-dto.ts`

```

1  export class UpdatePerroDto{
2      _id: string;
3      nombre: string;
4      edad: number;
5      raza: string;
6      pedigree: boolean;
7      created_at : Date;
8      updated_at :Date;
9      seguimiento: boolean;
10     propietario:{
11         primer_nombre: string;
12         segundo_nombre: string;
13         primer_apellido: string;
14         segundo_apellido: string;
15         edad: number;
16         direccion: {
17             tipo_via: string;
18             nombre_via: string;
19             numero: number;
20             cp: number;
21             localidad: string;
22             municipio: string;
23         };
24         telefonos: {
25             movil: number;
26             fijo: number;
27         }
28     }
29 }

```

### 18. En `interfaces/i-perro.interfaces.ts`

```

1  import { Document } from 'mongoose';
2
3  export interface IPerro extends Document {
4      readonly nombre: string;
5      readonly edad: number;
6      readonly raza: string;
7      readonly pedigree: boolean;
8      readonly created_at : Date,
9      updated_at :Date,
10     readonly seguimiento: boolean;
11     propietario:{
12         readonly primer_nombre: string;
13         readonly segundo_nombre: string;
14         readonly primer_apellido: string;
15         readonly segundo_apellido: string;
16         readonly edad: number;
17         direccion: {
18             readonly tipo_via: string;

```

```

19         readonly nombre_via: string;
20         readonly numero: number;
21         readonly cp: number;
22         readonly localidad: string;
23         readonly municipio: string;
24     };
25     telefonos: {
26         readonly movil: number;
27         readonly fijo: number;
28     }
29 }
30 }

```

## 19. En **schemas/perro-schema.ts**

```

1  import * as mongoose from 'mongoose';
2
3  export const PerroSchema = new mongoose.Schema ({
4      nombre: { type: String, required: true},
5      edad: { type: Number, required: true },
6      raza: { type: String, default: 'Desconocido' },
7      seguimiento: { type: Boolean, default: true },
8      created_at : { type: Date, default: Date.now },
9      updated_at : { type: Date , default:null},
10     propietario:{
11         primer_nombre: { type: String},
12         segundo_nombre: { type: String},
13         primer_apellido: { type: String},
14         segundo_apellido: { type: String},
15         edad: { type: Number},
16         email: { type: String},
17         direccion: {
18             tipo_via: { type: String},
19             nombre_via: { type: String},
20             numero: { type: Number},
21             cp: { type: Number},
22             localidad: { type: String },
23             municipio: { type: String},
24         },
25         telefonos: {
26             movil: { type: Number},
27             fijo: { type: Number}
28         }
29     }
30 });

```

## 20. En **providers/perro.ts**

```

1  import { Connection } from "mongoose";
2  import { PerroSchema } from "../schemas/perro-schema";
3
4  export const perroProviders = [
5      {
6          provide: 'PERRO_MODEL',
7          useFactory: (connection: Connection) =>
8              connection.model('Perro', PerroSchema),
9          inject: ['DATABASE_CONNECTION'],
10      },
11  ];

```

## 21. En **services/perro.service.ts**

```

1  import { Inject, Injectable } from '@nestjs/common';
2  import { Model } from 'mongoose';
3  import { IRetun } from 'src/assets/interfaces/i-retun.interface';
4  import { CreatePerroDto } from 'src/perros/dto/create-perro-dto';
5  import { UpdatePerroDto } from 'src/perros/dto/update-perro-dto';
6  import { IPerro } from 'src/perros/interfaces/i-perro.interface';
7
8  @Injectable()
9  export class PerroService {
10      /*****
11

```

```

12  * @param IPerro
13  * @returns
14  *****/
15  constructor(
16    @Inject('PERRO_MODEL')
17    private readonly perroModel: Model<IPerro>,
18  ) { }
19
20  /*****
21  * @param CreatePerroDto
22  * @returns Promise<IRetun>
23  *****/
24  async create(createPerroDto: CreatePerroDto): Promise<IRetun> {
25    const existPerro = await this.perroModel.exists({ nombre: createPerroDto.nombre});
26    //Se llama a la promesa
27    const myPromise = new Promise<IRetun>((resolve, reject) => {
28      if (!existPerro) {
29        new this.perroModel(createPerroDto).save().then(saved => {
30          resolve({ msg: 'Perro creado', status: 400, data: saved, code: '400', validRequest:
31 true});
32        });
33      }
34      else {
35        resolve({ msg: 'El nombre de perro ya existe', status: 500, data: undefined, code: '500',
36 validRequest: false });
37      }
38    });
39    return myPromise;
40  }
41
42  /*****
43  * @param
44  * @returns Promise<IRetun>
45  *****/
46  async findAll(): Promise<IRetun> {
47    const myPromise = new Promise<IRetun>((resolve, reject) => {
48      this.perroModel.find().exec().then(r => {
49        resolve({ msg: 'Perros:', status: 400, data: r, code: '400', validRequest: true});
50      });
51    });
52    return myPromise;
53  }
54
55  /*****
56  * @param id
57  * @returns Promise<IRetun>
58  *****/
59  async delete(id: string): Promise<IRetun> {
60    const exist = await this.perroModel.exists({ _id: id });
61    const myPromise = new Promise<IRetun>((resolve, reject) => {
62      if (!exist) {
63        resolve({ msg: 'El perro no existe', status: 500, data: undefined, code: '500',
64 validRequest: false });
65      }
66      else {
67        this.perroModel.deleteOne({ _id: id }).exec();
68        resolve({ msg: 'El perro fue eliminado', status: 400, data: id, code: '400',
69 validRequest: true });
70      }
71    });
72    return myPromise;
73  }
74
75  /*****
76  * @param UpdatePerroDto
77  * @returns Promise<IRetun>
78  *****/
79  async update(updatePerroDto: UpdatePerroDto): Promise<IRetun> {
80
81
82
83
84
85

```

```

96     if(updatePerroDto._id === undefined || updatePerroDto._id === '' || updatePerroDto._id.trim()
97     === ''){
98         return new Promise<IRetun>((resolve,reject) => {
99             resolve({ msg: 'Falta id', status: 400, data: undefined, code: '405',  validRequest:
100             false});
101         })
102     }
103     const exist = await this.perroModel.exists({ _id: updatePerroDto._id });
104     const myPromise = new Promise<IRetun>((resolve, reject) => {
105         if (!exist) {
106             resolve({ msg: 'El perro no existe', status: 500, data: undefined, code: '500',
107             validRequest: false });
108         }
109         else {
110             updatePerroDto.updated_at = new Date;
111             this.perroModel.findOneAndUpdate({_id:updatePerroDto._id},updatePerroDto,{ new:
112             true}).exec();
113             resolve({ msg: 'El perro existe', status: 400, data: updatePerroDto, code: '400',
114             validRequest: true });
115         }
116     });
117     return myPromise;
118 }

```

## 22. En api/perro/perro.controller.ts

```

1  import { Body, Controller, Delete, Get, Param, Post } from '@nestjs/common';
2  import { CreatePerroDto } from 'src/perros/dto/create-perro-dto';
3  import { UpdatePerroDto } from 'src/perros/dto/update-perro-dto';
4  import { PerroService } from 'src/perros/services/perro/perro.service';
5
6  @Controller('api/tiendaAnimal/v0/perros')
7  export class PerroController {
8      /*****
9       * @param PerroService
10      * @returns
11      *****/
12      constructor(private perroService: PerroService) {}
13
14      /*****
15       * @param CreatePerroDto
16       * @returns create()
17       *****/
18      @Post('create')
19      create(@Body() perroDetalle: CreatePerroDto) {
20          //se llama a la promesa
21          return this.perroService.create(perroDetalle).then(r => {
22              return r;
23          });
24      }
25
26      /*****
27       * @param
28       * @returns findAll()
29       *****/
30      @Get('readAll')
31      readAll() {
32          return this.perroService.findAll();
33      }
34
35      /*****
36       * @param id
37       * @returns delete()
38      *****/
39
40

```



```

41     *****/
42     @Delete('delete/:id')
43     delete(@Param('id') id: string) {
44         return this.perroService.delete(id).then(r => {
45             return r;
46         });
47     }
48 }
49
50 /*****
51  * @param UpdateCatDto
52  * @returns update()
53  *****/
54 @Post('update')
55 update(@Body() perroDetalle: UpdatePerroDto) {
56     return this.perroService.update(perroDetalle).then(r => {
57         return r;
58     });
59 }
60 }

```

### 23. En **perros.modules.ts**

```

1  import { Module } from '@nestjs/common';
2  import { perroProviders } from '../providers/perro';
3  import { PerroService } from '../services/perro/perro.service';
4  import { PerroController } from '../api/perro/perro.controller';
5  import { DatabaseModule } from 'src/database/database.module';
6
7  @Module({
8      imports: [DatabaseModule],
9      providers: [...perroProviders, PerroService],
10     controllers: [PerroController]
11 })
12 export class PerrosModule {}

```

### Por último, en **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      //Para que se comuniquen con el frontend
8      app.enableCors();
9  }
10 bootstrap();

```

Usamos los **Cors** porque trabajamos con dos aplicaciones por separado, una para el **Backend** y otra para el **Frontend**. Esto solo nos permiten trabajar con los métodos *POST/GET/DELETE*.

Y compilamos el proyecto:

```
npm run start
npm run strat:debug //se compila automáticamente con los cambios
```

```
> nest start
[Nest] 1936 - 20/11/2020 15:01:51 [NestFactory] Starting Nest application...
[Nest] 1936 - 20/11/2020 15:01:51 [InstanceLoader] AppModule dependencies initialized +50ms
[Nest] 1936 - 20/11/2020 15:01:51 [InstanceLoader] DatabaseModule dependencies initialized +21ms
[Nest] 1936 - 20/11/2020 15:01:51 [InstanceLoader] PerrosModule dependencies initialized +10ms
[Nest] 1936 - 20/11/2020 15:01:51 [InstanceLoader] GatosModule dependencies initialized +0ms
[Nest] 1936 - 20/11/2020 15:01:51 [InstanceLoader] PajarracosModule dependencies initialized +0ms
[Nest] 1936 - 20/11/2020 15:01:51 [InstanceLoader] RoedoresModule dependencies initialized +1ms
[Nest] 1936 - 20/11/2020 15:01:51 [RoutesResolver] AppController {}: +5ms
[Nest] 1936 - 20/11/2020 15:01:51 [RouterExplorer] Mapped {/, GET} route +2ms
[Nest] 1936 - 20/11/2020 15:01:51 [RoutesResolver] PerroController {/api/tiendaAnimal/v0/perros}: +1ms
[Nest] 1936 - 20/11/2020 15:01:51 [RouterExplorer] Mapped {/api/tiendaAnimal/v0/perros/create, POST} route +2ms
[Nest] 1936 - 20/11/2020 15:01:51 [RouterExplorer] Mapped {/api/tiendaAnimal/v0/perros/readAll, GET} route +1ms
[Nest] 1936 - 20/11/2020 15:01:51 [RouterExplorer] Mapped {/api/tiendaAnimal/v0/perros/delete/:id, DELETE} route +3ms
[Nest] 1936 - 20/11/2020 15:01:51 [RouterExplorer] Mapped {/api/tiendaAnimal/v0/perros/update, POST} route +0ms
[Nest] 1936 - 20/11/2020 15:01:51 [RoutesResolver] GatoController {/api/tiendaAnimal/v0/gatos}: +1ms
[Nest] 1936 - 20/11/2020 15:01:51 [RouterExplorer] Mapped {/api/tiendaAnimal/v0/gatos/create, POST} route +0ms
[Nest] 1936 - 20/11/2020 15:01:51 [RouterExplorer] Mapped {/api/tiendaAnimal/v0/gatos/readAll, GET} route +1ms
[Nest] 1936 - 20/11/2020 15:01:51 [RouterExplorer] Mapped {/api/tiendaAnimal/v0/gatos/delete/:id, DELETE} route +0ms
[Nest] 1936 - 20/11/2020 15:01:51 [RouterExplorer] Mapped {/api/tiendaAnimal/v0/gatos/update, POST} route +1ms
```

## Comprobación del funcionamiento

Vamos a comprobar el funcionamiento de los **entrypoints** con **Postman**:

Para todos los **endpoint** hay que declararles las cabeceras.

- KEY: Content-Type
- VALUE: application/json



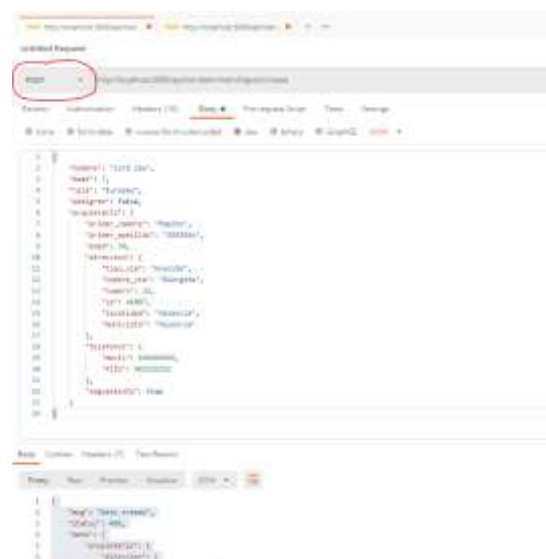
24. **Create** del gato => Dirección: <http://localhost:3000/api/tiendaAnimal/v0/gatos/create>

Estructura que le mandamos:

```
{
  "nombre": "Lord Zas",
  "edad": 7,
  "raza": "Europeo",
  "pedigree": false,
  "propietario": {
    "primer_nombre": "Pepito",
    "primer_apellido": "XXXXXXx",
    "edad": 30,
    "direccion": {
      "tipo_via": "Avenida",
      "nombre_via": "Giorgeta",
      "numero": 32,
      "cp": 46007,
      "localidad": "Valencia",
      "municipio": "Valencia"
    },
    "telefonos": {
      "movil": 666666666,
      "fijo": 962525252
    },
    "seguimiento": true
  }
}
```

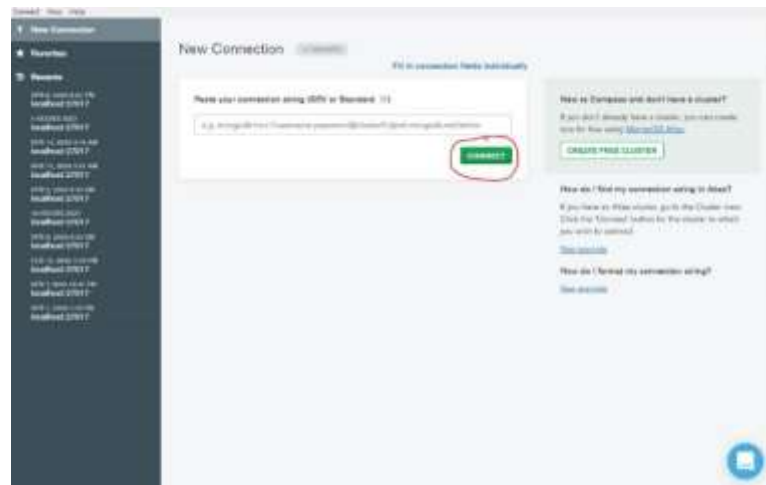
Lo que nos responde:

```
1 {
2   "msg": "Gato creado",
3   "status": 400,
4   "data": {
5     "propietario": {
6       "direccion": {
7         "tipo_via": "Avenida",
8         "nombre_via": "Giorgeta",
9         "numero": 32,
10        "cp": 46007,
11        "localidad": "Valencia",
12        "municipio": "Valencia"
13      },
14      "telefonos": {
15        "movil": 666666666,
16        "fijo": 962525252
17      },
18      "primer_nombre": "Pepito",
19      "primer_apellido": "XXXXXXx",
20      "edad": 30
21    },
22    "raza": "Europeo",
23    "seguimiento": true,
24    "updated_at": null,
25    "id": "5fbbb9cf2153fd4d4fc7b53a",
26    "nombre": "Lord Zas",
27    "edad": 7,
28    "created_at": "2020-11-23T13:31:59.051Z",
29    "_v": 0
30  },
31  "code": "400",
32  "validRequest": true
33 }
```

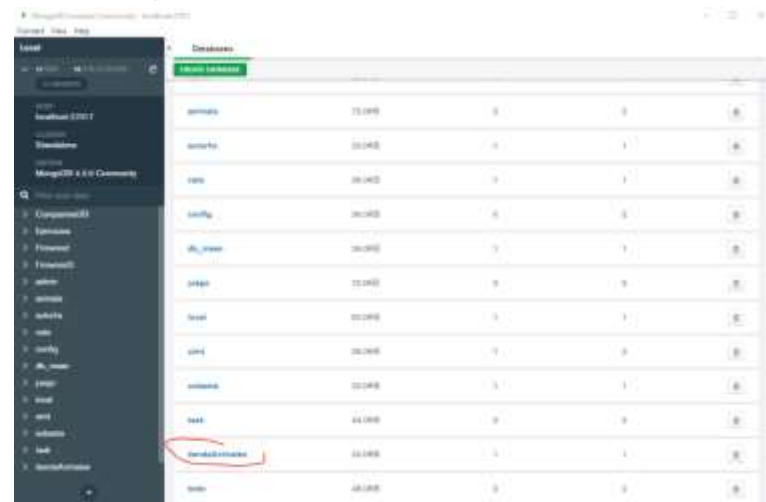


Para acabar de ver que te ha creado bien el objeto, nos vamos a **MongoDB Compass**


a. Nos conectamos al localhost



b. Entramos a la colección, en nuestro caso **tiendaAnimales**





c. Nos tienen que aparecer la colección **gatos**

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
gatos	1	3770 B	3770 B	1	20.0 KB	

**IMPORTANTE:** Para comprobar **update** y **delete**, necesitaremos copiar el **id** de uno de los objetos que tenemos en la colección.

25. **Create** del perro => Dirección: <http://localhost:3000/api/tiendaAnimal/v0/perros/create>

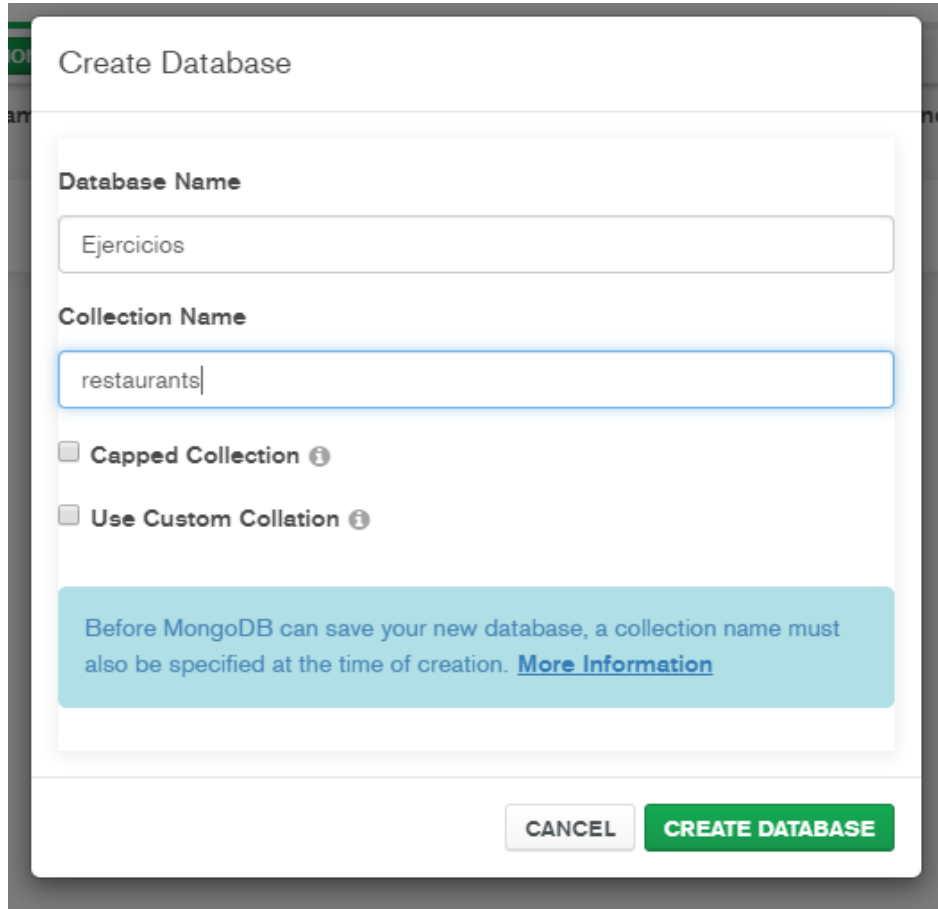
Si seguimos los pasos mencionados anteriormente, MongoDB creará de forma automática otra colección en nuestra base de datos.

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
<a href="#">gatos</a>	1	377.0 B	377.0 B	1	20.0 KB	
<a href="#">perros</a>	1	376.0 B	376.0 B	1	20.0 KB	

**NOTA:** Prueba a comprobar el funcionamiento del resto de **entypoints**.

## Importar base de datos a Mongo DB

1. Entramos en MongoDB Compass.
2. Creamos la base de datos **Agregados** con colección **NotasAlumnos**.



Create Database

Database Name

Ejercicios

Collection Name

restaurants

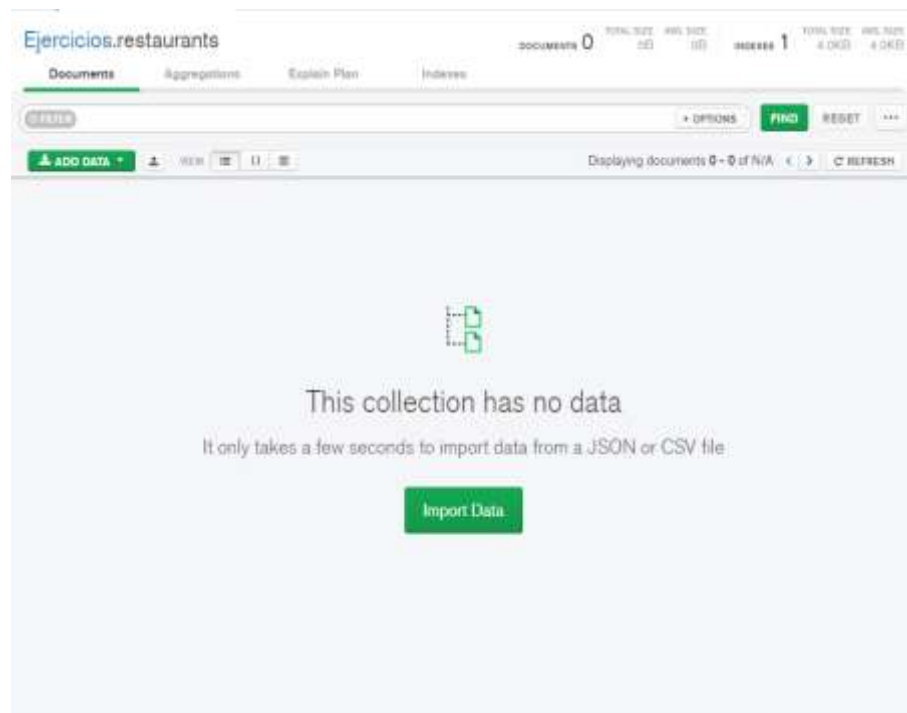
☐ Capped Collection ⓘ

☐ Use Custom Collation ⓘ

Before MongoDB can save your new database, a collection name must also be specified at the time of creation. [More Information](#)

CANCEL CREATE DATABASE

3. Importamos los datos de **NotasAlumnos.csv**



Select Input File Type

JSON

CSV

Options

Select delimiter 

COMMA

☒ Ignore empty strings

☐ Stop on errors

<input checked="" type="checkbox"/> pr4 Number	<input checked="" type="checkbox"/> pr5 Number	<input checked="" type="checkbox"/> participacion Double	<input checked="" type="checkbox"/> promedio Double	<input checked="" type="checkbox"/> notaf Double
0	1	0.714852074	empty string	empty string
2	2	0.619768014	empty string	empty string
3	2	0.528018833	empty string	empty string
2	3	0.233489696	empty string	empty string
0	2	0.40959215	empty string	empty string
2	0	0.486871558	empty string	empty string
0	1	0.565694253	empty string	empty string
0	3	0.506507787	empty string	empty string
1	0	0.437199528	empty string	empty string
1	3	0.850869247	empty string	empty string

CANCEL

IMPORT

Quedaría así:

```
_id: ObjectId("5fd15c02a98d4041f05e6bae")
idalumno: 1
pr1: 0
pr2: 1
pr3: 0
pr4: 0
pr5: 1
participacion: 0.714852074
promedio: 0
notaf: 0
```

---

```
_id: ObjectId("5fd15c02a98d4041f05e6baf")
idalumno: 2
pr1: 0
pr2: 0
pr3: 0
pr4: 2
pr5: 2
participacion: 0.619768014
promedio: 0
notaf: 0
```

---

```
_id: ObjectId("5fd15c02a98d4041f05e6bb0")
idalumno: 3
pr1: 1
pr2: 1
pr3: 0
pr4: 3
pr5: 2
participacion: 0.528018833
promedio: 0
notaf: 0
```



## Agregados de Mongo DB

1. Abrir Mongo en cmd
2. En el general: mongo
3. show dbs
4. use Agregados

### Sacar la media de cada parcial:

```
db.NotasAlumnos.aggregate([{$group:{ _id: "media", mediaPr1: {$avg:$pr1}}}]
```

```
db.NotasAlumnos.aggregate([{$group:{ _id: "media", mediaPr1: {$avg:$pr1},  
mediaPr2:{$avg:$pr2}}}]
```

### La suma total de todos los parciales por alumno:

```
db.NotasAlumnos.updateMany({},  
  [  
    { $set: { sumaTotal : {$sum :{$sum: ['$pr1','$pr2', '$pr3', '$pr4', '$pr5']}} } }  
  ]  
)
```

### El promedio de cada alumno:

```
db.NotasAlumnos.updateMany({},  
  [  
    { $set: { promedio :{$avg : ['$pr1','$pr2', '$pr3', '$pr4', '$pr5']}} } }  
  ]  
)
```

### Nota final, sumando el promedio con la participación:

```
db.NotasAlumnos.updateMany({},  
  [  
    { $set: { notaf : {$sum :{$sum: ['$promedio','$participacion']}} } }  
  ]  
)
```