TUTORIAL NESTJS

# Creating a REST API application

## Running NestJS in Development Mode

*// Development mode:*

$ npm run start:dev

## Creating a Basic Controller

*// Generate a Controller*

$ nest generate controller

*// shorthand: $ nest g co*

*// shorthand: $ nest g co module/user*

*/\* CoffeesController \*/*

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

@Controller('coffees')

export class CoffeesController {

  // /coffees

  @Get()

  findAll(): string {

    return 'Return all coffees';

  }

  // /coffees/flavor

  @Get('flavor')

  findFlavor(): string {

    return 'Return all coffees flavor';

  }

}

## Use Route Parameters

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

@Controller('coffees')

export class CoffeesController {

  // /coffees/:id

  @Get(':id')

  // findOne(@Param('id') id: string) {

  //   return `Return coffee flavor by ID #${id}`;

  // }

  findOne(@Param() params) {

    return `Return coffee flavor by ID #${params.id}`;

  }

}

## Handling Request Body / Payload

### Get

import {

  Controller,

  Get,

  Param

} from '@nestjs/common';

@Controller('coffees')

export class CoffeesController {

  @Get()

  findAll() {

    return 'This action returns all coffees';

  }

  @Get(':id')

  findOne(@Param('id') id: string) {

    return `This action returns #${id} coffee`;

  }

}

### Create

Graphical user interface, text, application, email

Description automatically generated

import { Controller, Post, Body } from '@nestjs/common';

@Controller('coffees')

export class CoffeesController {

  // coffees

  @Post()

  // create(@Body('name') body){

  //   return body;

  // }

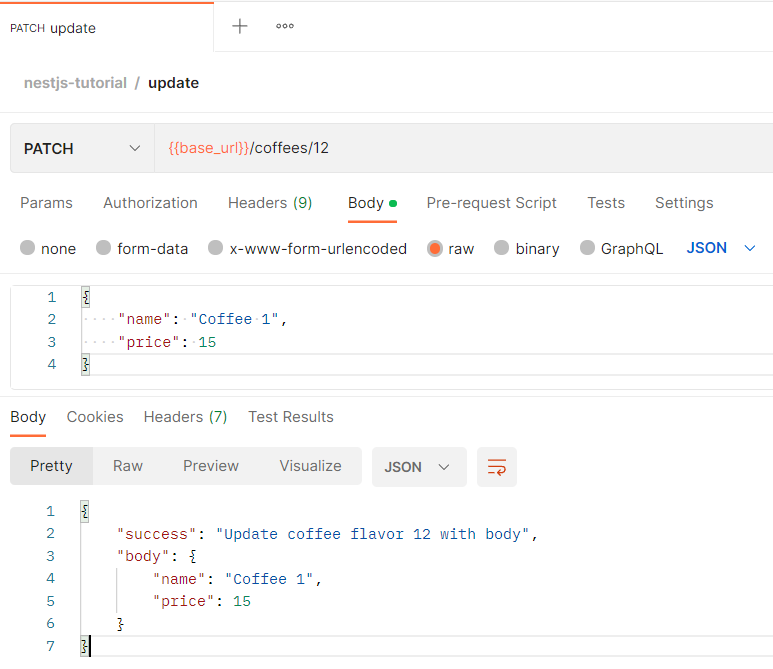
  create(@Body() body){

    return body.name;

  }

}

### Update



import { Controller, Res, Patch, Body, HttpStatus } from '@nestjs/common';

@Controller('coffees')

export class CoffeesController {

  // coffees/:id

  @Patch(':id')

  update(@Param('id') id: string, @Body() body, @Res() res) {

    res.status(HttpStatus.OK).json({success: `Updated coffee flavor ${id} with body`, body: body});

  }

}

### Delete

Graphical user interface, text, application, email

Description automatically generated

import { Controller, Res, Param, Delete, HttpStatus } from '@nestjs/common';

@Controller('coffees')

export class CoffeesController {

  // coffees/:id

  @Delete(':id')

  delete(@Param('id') id: string, @Res() res) {

    res.status(HttpStatus.OK).json(`Deleted coffee flavor ${id}`);

  }

}

## Response Status Codes

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

@Controller('coffees')

export class CoffeesController {

  // /coffees

  @Get()

  findAll(@Res() res) {

    // res.status(200).send('Return all coffees');

    res.status(HttpStatus.OK).send('Return all coffees');

  }

}

## Creating a Basic Service

### Create a service

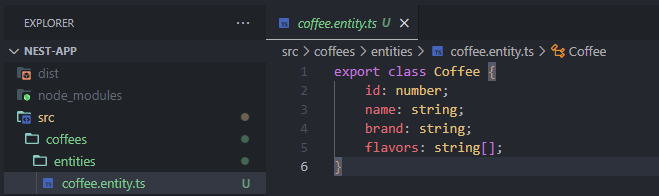
// Generate a Service with the Nest CLI

nest generate service coffees

// shorthand: nest g s coffees

### Create an entity schema

Tạo schema các bảng cơ sở dữ liệu.



### Create methods for the entity (service)

*coffees.service.ts*

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

import { Coffee } from './entities/coffee.entity';

@Injectable()

export class CoffeesService {

    private coffees: Coffee[] = [

        {

            id: 1,

            name: 'Coffee 1',

            brand: 'Trung Nguyen',

            flavors: ['chocolate', 'vanilla']

        },

        {

            id: 2,

            name: 'Coffee 2',

            brand: 'Thai Nguyen',

            flavors: ['chocolate', 'vanilla']

        }

    ];

    findAll(){

        return this.coffees

    }

    findOne(id: number){

        return this.coffees.find(item => item.id === +id);

    }

    create(createCoffeeDto: any){

        this.coffees.push(createCoffeeDto)

    }

    update(id: number, updateCoffeeDto: any){

        const existingCoffee = this.findOne(id)

        if(existingCoffee){

            const foundIndex = this.coffees.findIndex(x => x.id === +id);

            return this.coffees[foundIndex] = updateCoffeeDto

        }

    }

    delete(id: number){

        const coffeeIndex = this.coffees.findIndex(coffee => coffee.id === +id)

        if(coffeeIndex >= 0){

           return this.coffees.filter(obj => obj.id !== +id);

        }

    }

}

### Using service: import to controller.ts

*coffees.controler.ts*

import { Controller, Get, Res, Param, Post, Patch, Delete, Body, HttpStatus } from '@nestjs/common';

import { CoffeesService } from './coffees.service';

@Controller('coffees')

export class CoffeesController {

  constructor(private readonly coffeesService: CoffeesService){}

  // /coffees

  @Get()

  findAll(@Res() res) {

    res.status(HttpStatus.OK).json(this.coffeesService.findAll());

  }

  // /coffees/:id

  @Get(':id')

  findOne(@Param() params, @Res() res) {

    res.status(HttpStatus.OK).json(this.coffeesService.findOne(params.id));

  }

  // coffees

  @Post()

  create(@Body() body, @Res() res){

    res.status(HttpStatus.OK).json(this.coffeesService.create(body));

  }

  // coffees/:id

  @Patch(':id')

  update(@Param('id') id: string, @Body() body, @Res() res) {

    res.status(HttpStatus.OK).json(this.coffeesService.update(+id, body));

  }

  // coffees/:id

  @Delete(':id')

  delete(@Param('id') id: string, @Res() res) {

    res.status(HttpStatus.OK).json(this.coffeesService.delete(+id));

  }

}

### Send End-User Error Messages

*coffees.service.ts*

import { Injectable, NotFoundException } from '@nestjs/common';

import { Coffee } from './entities/coffee.entity';

@Injectable()

export class CoffeesService {

    findOne(id: number){

        const coffee = this.coffees.find(item => item.id === +id);

        if(!coffee){

            throw new NotFoundException(`Coffee ${id} not found!`)

        }

        return coffee

    }

}

These are exposed from the @nestjs/common package, and represent many of the most common HTTP exceptions:

* BadRequestException
* UnauthorizedException
* NotFoundException
* ForbiddenException
* NotAcceptableException
* RequestTimeoutException
* ConflictException
* GoneException
* HttpVersionNotSupportedException
* PayloadTooLargeException
* UnsupportedMediaTypeException
* UnprocessableEntityException
* InternalServerErrorException
* NotImplementedException
* ImATeapotException
* MethodNotAllowedException
* BadGatewayException
* ServiceUnavailableException
* GatewayTimeoutException
* PreconditionFailedException

Graphical user interface, text, application, email

Description automatically generated

## Encompass Business-Domain in Modules

// Generate a Nest Module with the Nest CLI

nest g module {name}

// shorthand: nest g mo coffees

* ***controllers***: route for module.
* ***exports***– export list providers within this current module that should be **made available anywhere** this module is imported
* ***imports***- Just as we saw in the AppModule, gives us list *OTHER*modules. Any exported providers of these imported modules are now fully available here.
* ***providers****-*list our **services**. Any providers here will be available only within “THIS” module itself, unless added to the exports.

*coffees.module.ts*

import { Module } from '@nestjs/common';

import { CoffeesController } from './coffees.controller';

import { CoffeesService } from './coffees.service';

@Module({

    controllers: [CoffeesController],

    providers: [CoffeesService]

})

export class CoffeesModule {}

*app.module.ts*

import { Module } from '@nestjs/common';

import { AppController } from './app.controller';

import { AppService } from './app.service';

import { CoffeesModule } from './coffees/coffees.module';

@Module({

  imports: [CoffeesModule],

  controllers: [AppController],

  providers: [AppService],

})

export class AppModule {}

## Introduction to Data Transfer Objects (DTO)

### Using and create DTO

Using define object to create and update by a module.

/\*\*

 \* Generate a DTO class with the Nest CLI

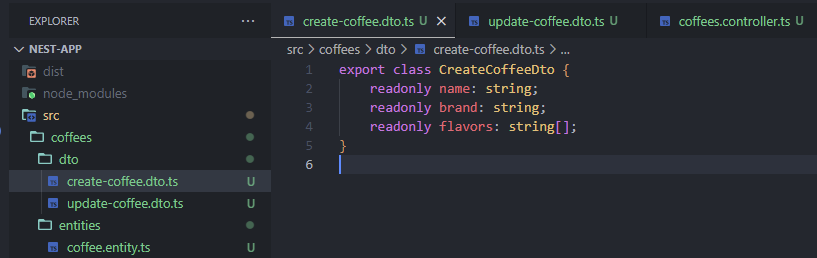
 \* --no-spec (no test file needed for DTO's)

 \*/

nest g class coffees/dto/create-coffee.dto --no-spec

### Create-Object DTO

*create-coffee.dto.ts*



### Update-Object DTO

*update-coffee.dto.ts*

A screenshot of a video game

Description automatically generated

### Import DTO to CONTROLLER: create and update

import { Controller, Res, Param, Post, Patch, Body, HttpStatus } from '@nestjs/common';

import { CoffeesService } from './coffees.service';

import { CreateCoffeeDto } from './dto/create-coffee.dto';

import { UpdateCoffeeDto } from './dto/update-coffee.dto';

@Controller('coffees')

export class CoffeesController {

  constructor(private readonly coffeesService: CoffeesService){}

  @Post()

  create(@Body() createCoffeeDto: CreateCoffeeDto, @Res() res){

    res.status(HttpStatus.OK).json(this.coffeesService.create(createCoffeeDto));

  }

  @Patch(':id')

  update(@Param('id') id: string, @Body() updateCoffeeDto: UpdateCoffeeDto, @Res() res) {

    res.status(HttpStatus.OK).json(this.coffeesService.update(+id, updateCoffeeDto));

  }

}

## Validate Input Data with Data Transfer Objects

### Install and config

// Install needed dependencies

npm i class-validator class-transformer

// Apply the ValidationPipe globally in our main.ts file

import { ValidationPipe } from '@nestjs/common';

app.useGlobalPipes(new ValidationPipe());

// Implement validation rules in our CreateCoffeeDto

import { IsString } from 'class-validator';

export class CreateCoffeeDto {

  @IsString()

  readonly name: string;

  @IsString()

  readonly brand: string;

  @IsString({ each: true })

  readonly flavors: string[];

}

// Install @nestjs/mapped-types

npm i @nestjs/mapped-types

/\* UpdateCoffeeDto - FINAL CODE  \*/

// PartialType: su dung de gan validate rule UpdateCoffeeDto giong voi CreateCoffeeDto

import { PartialType } from '@nestjs/mapped-types';

import { CreateCoffeeDto } from './create-coffee.dto';

export class UpdateCoffeeDto extends PartialType(CreateCoffeeDto) {}

### Note important

* By default, **every path parameter** and **query parameter** comes **over the network as a string**. So that, all path parameter and query parameter from **controller** is string 🡪 transfer to **service** is string. If value service require number 🡪 parseInt() or (+string, example: +id).
* To **auto stranform to type variable define in service**, using transfrom: true.

app.useGlobalPipes(new ValidationPipe({transform: true}));

### Example

*Before not to use*

findOne(id: number){

   const coffee = this.coffees.find(item => item.id === +id);

}

*After using*

findOne(id: number){

   const coffee = this.coffees.find(item => item.id === id);

}

## Handling Malicious Request Data (mã độc)

To remove malicious request data, do:

app.useGlobalPipes(

   new ValidationPipe({

     whitelist: true,            // config to remove invalid data object send to server then do action (ex: insert valid data to db)

     forbidNonWhitelisted: true  // config to not allow to do action (ex: insert invalid data to db) and then message to user

   })

);

# Add PostgreSQL with TypeORM