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;

  // }

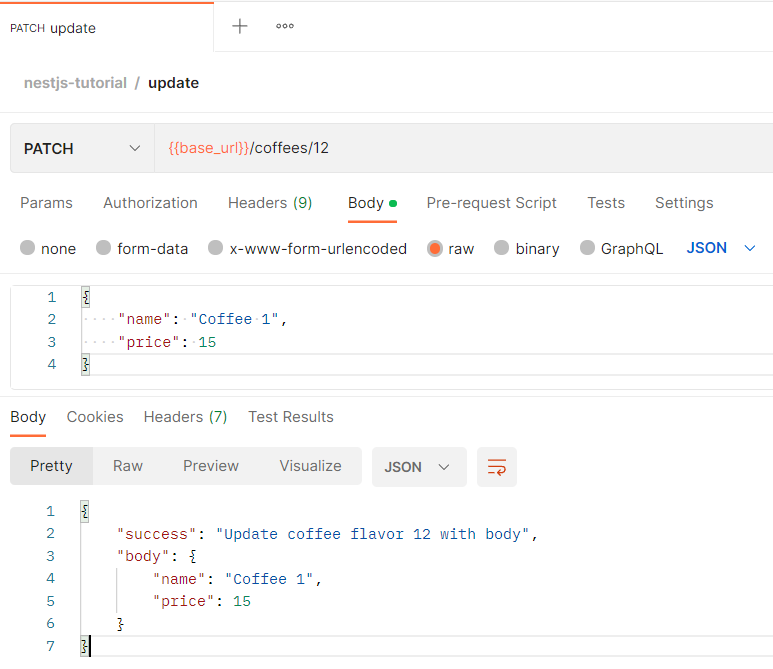
  create(@Body() body){

    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 (item.service.ts)

item.entity.ts

item.service.ts

item.controller.ts

import

### Create a service

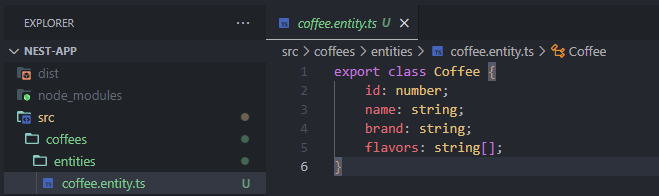
// Generate a Service with the Nest CLI

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, sử dụng cho quá trình create and update by a module. Hệ thống sẽ tự động chuyển kiểu dữ liệu từ string (client gửi lên) về kiểu dữ liệu đã định nghĩa trong Object DTO.

/\*\*

 \* Generate a DTO class with the Nest CLI

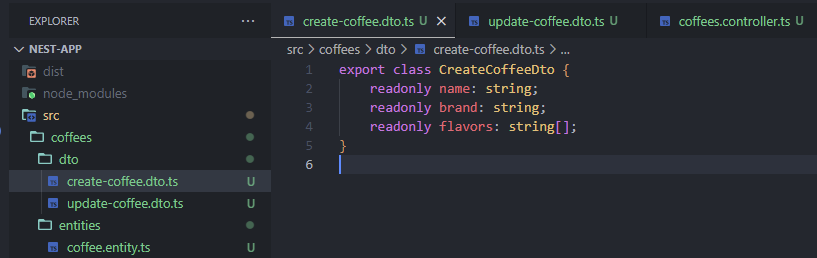
 \* --no-spec (no test file needed for DTO's)

 \*/

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) {}

### Auto transform Param / query

* 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

   })

);

# Pipes

## Use

Sử dụng cho validate dữ liệu client gửi lên trước khi đưa vào xử lý.

* ValidationPipe
* ParseIntPipe
* ParseFloatPipe
* ParseBoolPipe
* ParseArrayPipe
* ParseUUIDPipe
* ParseEnumPipe
* DefaultValuePipe
* ParseFilePipe

**Ví dụ**



**Khi chưa sử dụng pipes**

@Get(':id')

  async findOne(@Param('id') id: number, @Res() res) {

    const coffee = await this.coffeesService.findOne(id)

    res.status(HttpStatus.OK).json(coffee);

  }

A screenshot of a test results

Description automatically generated

**Sau khi sử dụng pipes**

A computer code with colorful text

Description automatically generated

A screenshot of a test results

Description automatically generated

## ValidationPipe

Sử dụng để validate data từ client gửi lên trước khi đưa vào sử lý.

**Cài đặt**

npm i --save class-validator class-transformer

**Định nghĩa Vlidation pipe on DTO cho một entity cụ thể**

import { IsString, IsNotEmpty } from 'class-validator'

export class CreateCoffeeDto {

    @IsString()

    @IsNotEmpty()

    readonly name: string;

    @IsString()

    readonly brand: string;

    @IsString({each: true})

    readonly flavors: string[];

}

**Validate before into controller**

 @UsePipes(new ValidationPipe())

  @Post()

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

    const coffee = await this.coffeesService.create(createCoffeeDto)

    res.status(HttpStatus.CREATED).json(coffee);

  }

**Ví dụ**

A screenshot of a computer

Description automatically generated

A screenshot of a computer code

Description automatically generated

**Truyền đúng**

A screenshot of a computer

Description automatically generated

# Add PostgreSQL with TypeORM

## Install docker

Link: <https://docs.docker.com/desktop/install/windows-install>

/\* YAML docker-compose.yml configuration file \*/

version: "3"

services:

  db:

    image: postgres

    restart: always

    ports:

      - "5432:5432"

    environment:

       POSTGRES\_PASSWORD: pass123

// Start containers in detached / background mode

docker-compose up -d

// Stop containers

docker-compose down

## Introducing the TypeORM Module

### Install typeORM postgres, configModule

// Install neccessary TypeORM dependencies

pnpm install @nestjs/typeorm typeorm pg

// Install config to using process.env

npm i --save @nestjs/config

### Connect to DB

/\* AppModule - FINAL CODE \*/

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

import { ConfigModule } from '@nestjs/config';

import { TypeOrmModule } from '@nestjs/typeorm';

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

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

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

@Module({

  imports: [

ConfigModule.forRoot(),

    TypeOrmModule.forRoot({

      type: 'postgres',                           // type of our database

      host: process.env.HOST,                     // database host

      port: parseInt(process.env.POSTGRES\_PORT),  // database port

      username: process.env.POSTGRES\_USER,        // username

      password: process.env.POSTGRES\_PASSWORD,    // user password

      database: process.env.POSTGRES\_DB,          // name of database

      autoLoadEntities: true,                     // models will be loaded automatically

      synchronize: true                           // your entities will be synced with the database(recommended: disable in prod)

    }),

CoffeesModule,

  ],

  controllers: [AppController],

  providers: [AppService],

})

export class AppModule {}

## Creating a TypeORM Entity

### Create

import { Entity, PrimaryGeneratedColumn, Column } from "typeorm";

@Entity('coffees') // sql table = 'coffee'

export class Coffee {

    @PrimaryGeneratedColumn()   // Auto increment primary key

    id: number;

    @Column()

    name: string;

    @Column()

    brand: string;

    @Column('json', {nullable: true})   // accept null value

    flavors: string[];

}

### Tạo bảng

|  |  |
| --- | --- |
| **Item** | Ý nghĩa |
| @Entity() | Định nghĩa thực thể (Entity), bảng |
| @PrimaryGeneratedColumn() | Cột primary và tự động tăng dạng số nguyên |
| @PrimaryGeneratedColumn("uuid") | Cột tự động tăng dạng uuid |
| @Column()  @Generated("uuid")  uuid: **string** | Cột tự động tăng dạng uuid |
| @PrimaryColumn() | Primary column |
| @Column("geometry")  point: Point | **import** { Point, LineString, MultiPoint  } **from** "typeorm"  Điểm |
| @Column("geometry")  linestring: LineString | Đường |
| @Column("geometry", {  spatialFeatureType: "MultiPoint",  srid: 4326,  })  multiPointWithSRID: MultiPoint | Vùng |
| @Column("int") or @Column({ **type**: "int" }) | Dạng số |
| @Column("varchar", { length: 200 }) |  |
| @Column({ **type**: "int", width: 200 }) |  |
| @Column("text") | Cột, dạng text |
| @Column({ unique: **true** }) | Cột unique |
| @Column()  @Generated() | Cột, tự động tăng số |
| @Column({ primary: **true** }) | Cột primary |
| @CreateDateColumn(({ type: 'timestamptz', nullable: true });  createDate: Date; | Cột lưu thời gian create |
| @UpdateDateColumn({ type: 'timestamptz', nullable: true });  updateDate: Date; | Cột lưu thời gian update |
| @Entity()  @Index(["firstName", "lastName"], { unique: **true** })  **export** **class** User {} | Index nhiều cột trong bảng |
| @Column()  @Index() | Index một cột |

List of available options in ColumnOptions:

* type: ColumnType - Column type. One of the type listed [above](https://typeorm.io/entities#column-types).
* name: string - Column name in the database table. By default the column name is generated from the name of the property. You can change it by specifying your own name.
* length: number - Column type's length. For example if you want to create varchar(150) type you specify column type and length options.
* nullable: boolean - Makes column NULL or NOT NULL in the database. By default column is nullable: false.
* default: string - Adds database-level column's DEFAULT value.
* primary: boolean - Marks column as primary. Same if you use @PrimaryColumn.
* unique: boolean - Marks column as unique column (creates unique constraint).

Column type for postgres

int, int2, int4, int8, smallint, integer, bigint, decimal, numeric, real, float, float4, float8, double, character, varchar, character, char, text, citext, bit, timestamptz, timestamp, timestamp without time zone, timestamp with time zone, date, time, time without time zone, time with time, zone, interval, bool, boolean, enum, point, line, path, polygon, tsvector, tsquery, uuid, xml, json, jsonb, numrange, geometry, geography, cube, ltree

### Using entity: import into module

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

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

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

import { TypeOrmModule } from '@nestjs/typeorm';

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

@Module({

    imports: [TypeOrmModule.forFeature([Coffee])],

    controllers: [CoffeesController],

    providers: [CoffeesService]

})

export class CoffeesModule {}

## Using Repository to Access Database

### Connect to database

#### .env

HOST=localhost

POSTGRES\_DB=nestjs

POSTGRES\_PORT=5432

POSTGRES\_USER=postgres

POSTGRES\_PASSWORD=1

To load process environment variables using package:

Ref: <https://docs.nestjs.com/techniques/configuration>

npm i --save @nestjs/config

#### main.ts

*app.module.ts*

import { TypeOrmModule } from '@nestjs/typeorm';

import { ConfigModule } from '@nestjs/config';       // for using process.env

@Module({

  imports: [

    ConfigModule.forRoot(),

    TypeOrmModule.forRoot({

      type: 'postgres',                           // type of our database

      host: process.env.HOST,                     // database host

      port: parseInt(process.env.POSTGRES\_PORT),  // database port

      username: process.env.POSTGRES\_USER,        // username

      password: process.env.POSTGRES\_PASSWORD,    // user password

      database: process.env.POSTGRES\_DB,          // name of database

      autoLoadEntities: true,                     // models will be loaded automatically

      synchronize: true                           // your entities will be synced with the database(recommended: disable in prod)

    })

],

})

export class AppModule {}

### CRUD

#### coffees.service.ts

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

import { InjectRepository } from '@nestjs/typeorm';

import { Repository } from 'typeorm';

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

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

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

@Injectable()

export class CoffeesService {

    constructor(

        @InjectRepository(Coffee)

        private readonly coffeeRepository: Repository<Coffee>

    ) {}

    async findAll(){

        return await this.coffeeRepository.find()

    }

    async findOne(id: number){

        const coffee = await this.coffeeRepository.findOne({ where: { id: +id } });

        if(!coffee){

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

        }

return coffee

    }

    async create(createCoffeeDto: CreateCoffeeDto){

        const coffee = this.coffeeRepository.create(createCoffeeDto);

        return await this.coffeeRepository.save(coffee);

    }

    async update(id: number, updateCoffeeDto: UpdateCoffeeDto){

        const coffee = await this.coffeeRepository.preload({

            id: id,

            ...updateCoffeeDto

        })

        if(!coffee){

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

        }

        return this.coffeeRepository.save(coffee)

    }

    async delete(id: number){

        const coffee = await this.findOne(id)

        return this.coffeeRepository.remove(coffee)

    }

}

#### coffees.controller.ts

import { Controller, Get, Res, Param, Post, Patch, Delete, 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){}

  // /coffees

  @Get()

  async findAll(@Res() res) {

    const coffee = await this.coffeesService.findAll()

    res.status(HttpStatus.OK).json(coffee);

  }

  // /coffees/:id

  @Get(':id')

  async findOne(@Param('id') id: number, @Res() res) {

    const coffee = await this.coffeesService.findOne(id)

    res.status(HttpStatus.OK).json(coffee);

  }

  // coffees

  @Post()

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

    const coffee = await this.coffeesService.create(createCoffeeDto)

    res.status(HttpStatus.CREATED).json(coffee);

  }

  // coffees/:id

  @Patch(':id')

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

    const coffee = await this.coffeesService.update(id, updateCoffeeDto)

    res.status(HttpStatus.OK).json(coffee);

  }

  // coffees/:id

  @Delete(':id')

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

    const coffee = await this.coffeesService.delete(id)

    res.status(HttpStatus.OK).json(coffee);

  }

}

## Create a Relation between two Entities

### One to One

Mỗi user có một profile

*user.entity.ts*

import {

    Entity,

    PrimaryGeneratedColumn,

    Column,

    OneToOne,

    JoinColumn,

} from "typeorm"

import { Profile } from "./Profile"

@Entity()

export class User {

    @PrimaryGeneratedColumn()

    id: number

    @Column()

    name: string

@OneToOne(() => Profile, (profile) => profile.user)

  @JoinColumn()

    profile: Profile

}

Giải thích: User có quan hệ với bảng Profile qua cột profile.user

import { Entity, PrimaryGeneratedColumn, Column, OneToOne } from "typeorm"

import { User } from "./User"

@Entity()

export class Profile {

    @PrimaryGeneratedColumn()

    id: number

    @Column()

    gender: string

    @Column()

    photo: string

    @OneToOne(() => User, (user) => user.profile)

    user: User

}

### Many to one / One to Many

Một user có nhiều photo

*user.entity.ts*

import { Entity, PrimaryGeneratedColumn, Column, OneToMany } from "typeorm"

import { Photo } from "./Photo"

@Entity()

export class User {

    @PrimaryGeneratedColumn()

    id: number

    @Column()

    name: string

    @OneToMany(() => Photo, (photo) => photo.user)

    photos: Photo[]

}

**Lưu ý:** nhiều pho to 🡪 photo để ở dạng mảng Photo[].

*Photo.entity.ts*

import { Entity, PrimaryGeneratedColumn, Column, ManyToOne } from "typeorm"

import { User } from "./User"

@Entity()

export class Photo {

    @PrimaryGeneratedColumn()

    id: number

    @Column()

    url: string

    @ManyToOne(() => User, (user) => user.photos)

    user: User

}

### Many to many (n-n)

Tạo lớp class flavor có quan hệ n-n với coffee

$ nest g class coffees/entities/flavor.entity --no-spec

*flavor.entity.ts*

import { Entity, PrimaryGeneratedColumn, Column, ManyToMany } from "typeorm";

import { Coffee } from "./coffee.entity";

@Entity()

export class Flavor {

    @PrimaryGeneratedColumn()   // Auto increment primary key

    id: number;

    @Column()

    name: string;

    @ManyToMany(

        type => Coffee,

        (coffee) => coffee.flavors

    )

    coffees: Coffee[]

}

*coffee.entiti.ts*

import { Entity, PrimaryGeneratedColumn, Column, CreateDateColumn, UpdateDateColumn, JoinTable, ManyToMany } from "typeorm";

import { Flavor } from "./flavor.entity";

@Entity() // sql table = 'coffee' | 'coffee'

export class Coffee {

    @PrimaryGeneratedColumn()   // Auto increment primary key

    id: number;

    @Column()

    name: string;

    @Column()

    brand: string;

    @JoinTable()

    @ManyToMany(

        type => Flavor,

        (flavor) => flavor.coffees

    )

    flavors: Flavor[];

    @CreateDateColumn({ type: 'timestamptz', nullable: true })

    date\_created: Date;

    @UpdateDateColumn({ type: 'timestamptz', nullable: true })

    date\_updated: Date;

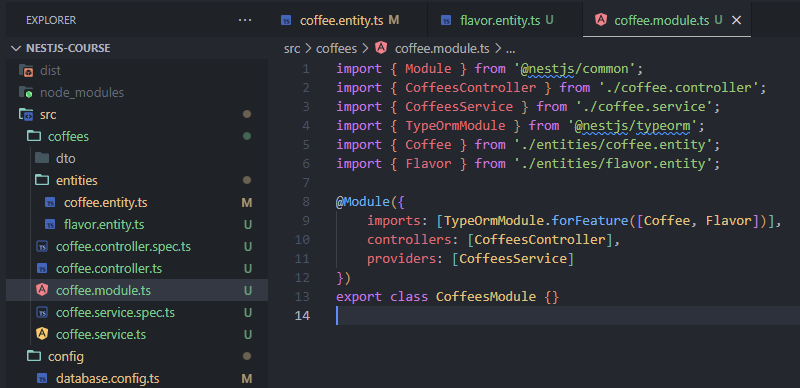
}

**Giải thích**

Coffee.entiti định nghĩa mối quan hệ @JoinTable giữa hai bảng coffee và flavor, mối quan hệ nhiều (n) – nhiều (n) (@ManyToMany).

* Bảng coffee: cột flavors, có kiểu dữ liệu Flavor (type => Coffee), quan hệ với bảng (flavor), thông qua cột (=> flavor.coffees).
* Bảng flavor: cột coffees, có kiểu dữ liệu Coffee (type => Coffee), quan hệ với bảng (coffee), thông qua cột (=> coffee.flavors).

Sau khi thiết lập mối quan hệ, cần import Entiry flavor và Coffee vào Coffee.module



Khi load code dev, bảng quan hệ **coffee\_flavors\_flavor** sẽ được typeorm tự tạo ra trên DB. Quan hệ hai bảng được xác định qua ID bảng. Cột flavors định nghĩa trong coffee.entiti.ts và flavor.entiti.ts không có trên DB, chỉ sử dụng để rằng buộc quan hệ giữa hai bảng.

Graphical user interface, text, application

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  |  |  |

### Select query with relation table

Để query dữ liệu hai bảng quan hệ, cần truyền tham số relations: { column\_relation: true }

async findAll(){

        return await this.coffeeRepository.find({

            relations: {

                flavors: true

            }

        })

    }

### Using Cascading Inserts and Updates

Khai báo cascade trong coffee.entiti.ts.

*coffee.entiti.ts*

@JoinTable()

    @ManyToMany(

        type => Flavor,

        (flavor) => flavor.coffees,

        {

            cascade: true,  // 👈 or optionally just insert ['insert'] or update ['update']

        }

    )

    flavors: Flavor[];

Khi insert và update Coffee chưa thuộc tính Flavor, thì Flavor sẽ tự động được thêm vào bảng flavor (khi chưa tồn tại flavor\_name, nếu đã tồn tại thì không thêm vào).

*coffee.service.ts*

private async preloadFlavorByName(name: string): Promise<Flavor> {

        const existingFlavor = await this.flavorRepository.findOne({ where: { name } }); // 👈 notice the "where"

        console.log('existingFlavor', existingFlavor);

        if (existingFlavor) {

          return existingFlavor;

        }

        return this.flavorRepository.create({ name });

    }

CRUD

async create(createCoffeeDto: CreateCoffeeDto){

        const flavors = await Promise.all(  // [ Flavor { id: 2, name: 'flavors 2' }, Flavor { name: 'flavors 4' } ], nếu flavor đã tồn tại trả về id, name; nếu chưa thì trả về name;

            createCoffeeDto.flavors.map(name => this.preloadFlavorByName(name))

        )

        const coffee = await this.coffeeRepository.create(

            {

                ...createCoffeeDto,

                flavors

            }

        );

        return this.coffeeRepository.save(coffee);

    }

    async update(id: number, updateCoffeeDto: UpdateCoffeeDto){

        const flavors = await Promise.all(

            updateCoffeeDto.flavors.map(name => this.preloadFlavorByName(name)),

        )

        const coffee = await this.coffeeRepository.preload({

            id: id,

            ...updateCoffeeDto,

            flavors

        })

        if(!coffee){

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

        }

        return this.coffeeRepository.save(coffee)

    }

    async delete(id: number){

        const coffee = await this.findOne(id)

        if(!coffee){

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

        }

        return this.coffeeRepository.remove(coffee)

    }

Create

|  |  |
| --- | --- |
| **Bảng** | **Tác động** |
| coffee | Thêm record mới |
| coffee\_flavors\_flavor | Typeorm tự động thêm |
| flavor | Nếu flavor\_name đã tồn tại, không thêm; Chưa tồn tại, thêm mới. |

Update

|  |  |
| --- | --- |
| **Bảng** | **Tác động** |
| coffee | Update lại thuộc tính theo id-coffee |
| coffee\_flavors\_flavor | Typeorm tự động update theo flavor-name cho row id-coffee truyền vào. |
| flavor | Nếu flavor\_name đã tồn tại, không thêm; Chưa tồn tại, thêm mới. |

Delete

|  |  |
| --- | --- |
| **Bảng** | **Tác động** |
| coffee | Xóa theo id-coffee |
| coffee\_flavors\_flavor | Typeorm tự động xóa row theo id-coffee truyền vào |
| flavor | Không tác động gì |

## Use Transactions

Sử dụng trong CRUD, thực hiện nhóm câu lệnh, nếu lệnh 1, 2,… thành công, nhưng lệnh 3,4, … thất bại thì hệ thống tự động restore lại trạng thái ban đầu.

Các lệnh sử dụng với transaction là CRUD;

### Tạo class event

Sử dụng để lưu event CRUD

$ nest g class events/entities/event.entity --no-spec

import { Column, Entity, PrimaryGeneratedColumn, CreateDateColumn } from "typeorm";

@Entity()

export class Event {

    @PrimaryGeneratedColumn()

    id: number;

    @Column()

    type: string;

    @Column()

    name: string;

    @Column('json')

    payload:  Record<string, any>;

    @CreateDateColumn({ type: 'timestamptz', nullable: true })

    date\_created: Date;

}

### Import class event

import { Event } from '../events/entities/event.entity';

@Module({

    imports: [TypeOrmModule.forFeature([Coffee, Flavor, **Event**])],

    controllers: [CoffeesController],

    providers: [CoffeesService]

})

export class CoffeesModule {}

### Apply to coffee.service

async createCoffeeWithTransaction(coffee: Coffee) {

        const queryRunner = this.dataSource.createQueryRunner();

        await queryRunner.connect();

        await queryRunner.startTransaction();

        try {

            // Save coffee

            const coffee\_save = await queryRunner.manager.save(coffee);

            // Save event

            const recommendEvent    =  new Event();

            recommendEvent.name     = 'create\_coffee';

            recommendEvent.type     = 'coffee';

            recommendEvent.payload  = { coffeeId : coffee\_save.id };

            await queryRunner.manager.save(recommendEvent);

            // commit transaction now

            await queryRunner.commitTransaction();

            // return coffee\_save

        } catch (error) { // since we have errors let's rollback changes we made

            await queryRunner.rollbackTransaction();

        } finally { // release query runner commit which is manually created

            await queryRunner.release();

        }

    }

## Indexs

### Index one column

import { Column, Index } from "typeorm";

@Entity() // sql table = 'coffee' | 'coffee'

export class Coffee {

@Column()

@Index()

name: string;

}

### Index multi column

import { Entity, Column, Index } from "typeorm";

import { Flavor } from "./flavor.entity";

@Index(["name", "brand"])

@Entity()

export class Coffee {

    @PrimaryGeneratedColumn()

    id: number;

    @Column()

    name: string;

    @Column()

    brand: string;

}

# Thuật ngữ, khái niệm

## Closures

**Closures** là functions lồng nhau.

Graphical user interface, text, application

Description automatically generated

Để thực hiện function **displayName()** bên trong thì **makeFunc()().**

Hàm bên trong displayName() có thể nhận được tham số bên ngoài function (name).

## Decorator

<https://www.typescriptlang.org/docs/handbook/decorators.html>

**Decorator** là kiểu cú pháp khai báo hàm (@function).

Luôn đi kèm với biến, class, method.

Được gọi lúc runtime (runtime is the period of time when a program is running. It begins when a program is opened (or executed) and ends with the program is quit or closed).

Có nhiệm vụ thay đổi hoặc bổ sung cho đối tượng được decorate.

@Entity()

export class Coffee {

@Column()

@Index()

name: string;

}

1. The expressions for each decorator are evaluated top-to-bottom.
2. The results are then called as functions from bottom-to-top.

@classDecorator

class Bird {

@propertyDecorator

name: string;

@methodDecorator

fly(

@parameterDecorator

meters: number

) {}

@accessorDecorator

get egg() {}

}

**Ví dụ**

Text

Description automatically generated

Text

Description automatically generated

**Giải thích**

* target: cho biết Decorator tác động đến Class nào
* propertyName: cho biết tên hàm Decorator tác động vào (hình dưới là hàm add)
* descriptor: descriptor.value chính là hàm Decorator tác động, nhận giá trị trả về từ hàm này (hàm add). Dựa vào descriptor để tác động đến hàm add để custom lại và truyền tham số cho hàm này.

## Functional composition

Là function bao bọc function khác.

function doBusinessJob(arg) {

console.log('do my job')

return arg

}

function logDecorator(job) { console.log('start my job') }

**Viết function composition**

logDecorator(doBusinessJob)

**Viết dưới dạng Decorator**

@logDecorator

doBusinessJob(arg)

**Kết quả chạy**

Graphical user interface, text, application, email

Description automatically generated

**Ý nghĩa**

Hàm doBusinessJob sẽ chay trước và trả về kết quả cho hàm logDecorator sử dụng.