TP-NESTJS-CRUD

1-Create New Project

nest new project-name

2-Add database mysql or you can use phpMyAdmin

Docker

docker-compose.yml

```
version: "3.8"
services:
  mysql:
    image: mysql:8.0
    container_name: mysql_db
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: db_crud
      MYSQL_USER: user_crud
      MYSQL_PASSWORD: root
    volumes:
      - ./mysql:/var/lib/mysql
    ports:
      - "3307:3306"
```

.gitignore

```
mysql
```

```
docker compose up -d
```

3-SETUP PROJECT RESOURCES

sh

```
nest g resource cats --no-spec
```

sh

```
yarn add class-validator class-transformer -SE
```

main.ts

ts

```
import { ValidationPipe } from "@nestjs/common";
import { NestFactory } from "@nestjs/core";
import { AppModule } from "./app.module";
```

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.setGlobalPrefix("api/v1");

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

  await app.listen(3000);
}
bootstrap();
```

src\cats\cats.controller.ts

ts

```
@Get(':id')
findOne(@Param('id') id: number) {
  return this.catsService.findOne(id);
}

@Patch(':id')
update(@Param('id') id: number, @Body() updateCatDto: UpdateCatDto) {
  return this.catsService.update(id, updateCatDto);
}

@Delete(':id')
remove(@Param('id') id: number) {
  return this.catsService.remove(id);
}
```

4-Environment Variables

sh

```
yarn add @nestjs/config -DE
```

src\app.module.ts
ts
```
import { Module } from "@nestjs/common";
import { CatsModule } from "./cats/cats.module";
import { ConfigModule } from "@nestjs/config";

@Module({
  imports: [ConfigModule.forRoot({ isGlobal: true }), CatsModule],
  controllers: [],
  providers: [],
})
export class AppModule {}
```

5- ADD TypeORM

sh
```
yarn add @nestjs/typeorm typeorm mysql2
```

src\app.module.ts
ts
```
import { Module } from "@nestjs/common";
import { TypeOrmModule } from "@nestjs/typeorm";
import { CatsModule } from "./cats/cats.module";

@Module({
  imports: [
    CatsModule,
    TypeOrmModule.forRoot({
      type: "mysql",
      host: "localhost",
      port: 3307,
      username: "user_crud",
      password: "root",
      database: "db_crud",
      autoLoadEntities: true,
      synchronize: true,
    }),
  ],
  controllers: [],
  providers: [],
})
export class AppModule {}
```

6-Repository pattern

The Repository Pattern is an architectural design pattern used in software development to separate data access logic and business logic. This pattern promotes abstraction and modularity, making code easier to maintain and extensibility.
T he main purpose of the Repository Pattern is to provide an interface between the business logic layer (services, controllers, etc.) and the data access layer (database, external APIs, etc.). This way, the business logic does not need to worry about the details of how data is accessed and stored, making the code more decoupled and flexible.
Features and benefits of the Repository Pattern:

- Data Source Abstraction : The repository pattern abstracts data access, meaning that the business logic code does not need to worry about whether the data comes from a database, an API, or any other source.
- Code Reuse : By providing a common interface to access data, it is easier to reuse code in different parts of the application.
- Separation of responsibilities : The repository pattern clearly divides responsibilities between business logic and data access logic, making the code easier to maintain and understand.
- Facilitates unit testing : By using interfaces to represent repositories, you can easily create test implementations (mocks) to isolate business logic during unit testing.
- Improves security : By controlling access to data through the repository pattern, it is easier to apply access controls and security measures.

TypeORM Repository pattern

TypeORM supports the repository design pattern, so each entity has its own repository. These repositories can be obtained from the database data source.
src\cats\entities\cat.entity.ts
ts

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

@Entity()
export class Cat {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  name: string;

  @Column()
  age: number;

  @Column()
```

```
  breed: string;

  @DeleteDateColumn()
  deletedAt: Date;
}
```

src\cats\cats.module.ts

```
ts
import { Module } from "@nestjs/common";
import { TypeOrmModule } from "@nestjs/typeorm";
import { CatsController } from "./cats.controller";
import { CatsService } from "./cats.service";
import { Cat } from "./entities/cat.entity";

@Module({
  imports: [TypeOrmModule.forFeature([Cat])],
  controllers: [CatsController],
  providers: [CatsService],
})
export class CatsModule {}
```

src\cats\cats.service.ts

```
ts
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { CreateCatDto } from './dto/create-cat.dto';
import { UpdateCatDto } from './dto/update-cat.dto';
import { Cat } from './entities/cat.entity';

@Injectable()
export class CatsService {
  constructor(
    @InjectRepository(Cat)
    private catsRepository: Repository<Cat>,
  ) {}

  create(createCatDto: CreateCatDto) {
    return 'This action adds a new cat';
  }

  findAll() {
    return this.catsRepository.find();
  }

  findOne(id: number) {
    return this.catsRepository.findOneBy({ id });
  }
```

```
  update(id: number, updateCatDto: UpdateCatDto) {
    return `This action updates a #${id} cat`;
  }

  remove(id: number) {
    return this.catsRepository.delete(id);
  }
}
```

src\cats\dto\create-cat.dto.ts

```
ts
import { IsInt, IsOptional, IsPositive, IsString } from "class-validator";

export class CreateCatDto {
  @IsString()
  name: string;

  @IsInt()
  @IsPositive()
  age: number;

  @IsString()
  @IsOptional()
  breed: string;
}
```

CREATE CRUD WITH REPOSITORY

src\cats\cats.service.ts

```
ts
import { Injectable } from "@nestjs/common";
import { InjectRepository } from "@nestjs/typeorm";
import { Repository } from "typeorm";
import { CreateCatDto } from "./dto/create-cat.dto";
import { UpdateCatDto } from "./dto/update-cat.dto";
import { Cat } from "./entities/cat.entity";

@Injectable()
export class CatsService {
  constructor(
    @InjectRepository(Cat)
    private catsRepository: Repository<Cat>
  ) {}

  async create(createCatDto: CreateCatDto) {
```

```
    const cat = this.catsRepository.create(createCatDto);
    return await this.catsRepository.save(cat);
  }

  async findAll() {
    return await this.catsRepository.find();
  }

  async findOne(id: number) {
    return await this.catsRepository.findOneBy({ id });
  }

  async update(id: number, updateCatDto: UpdateCatDto) {
    return await this.catsRepository.update(id, updateCatDto);
  }

  async remove(id: number) {
    return await this.catsRepository.softDelete(id);
  }
}
```

In soft delete , a special column in the table is used to mark records as "deleted" without physically removing them from the database. This column, usually called deletedAt (or whatever name you specified), stores the date and time that the soft delete was performed. Records marked with a value in this column are considered "deleted" and are not visible in regular queries, but still exist in the database and can be retrieved or restored if necessary. In soft remove , "deleted" records are moved to another special table (usually a historical or archive table) instead of simply marking them in the original table. This way, deleted records are still kept in the database, but in a different table, which helps keep the original table lighter and improves performance on regular queries.

GENERATE RELATIONS WITH TYPEORM

Relationship Type	Description	Decorator
One-to-one	Each row in the main table has one and only one associated row in the external table.	@OneToOne()
One-to-many	Each row in the main table has one or more related rows in the external table.	@OneToMany()

Many-to-one	Each row in the main table has one or more related rows in the external table.	@ManyToOne()
Many-to-many	Each row in the main table has many related rows in the external table and each record in the external table has many related rows in the main table.	@ManyToMany()

Many-to-one and One-to-many

sh
nest g resource breeds --no-spec

In this relationship, one breed can be associated with many cats. In the Breed entity, we use @OneToMany()to establish the relationship with the Cat entity. The cats property in the Breed entity will be a collection of cats that are associated with that breed.
src\breeds\entities\breed.entity.ts
ts

```
import { Cat } from "src/cats/entities/cat.entity";
import {
  Column,
  DeleteDateColumn,
  Entity,
  OneToMany,
  PrimaryGeneratedColumn,
} from "typeorm";

@Entity()
export class Breed {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({ unique: true })
  name: string;

  @OneToMany(() => Cat, (cat) => cat.breed)
  cats: Cat[];

  @DeleteDateColumn()
  deletedAt: Date;
}
```

In this relationship, several cats can belong to a single breed. In the Cat entity, we use @ManyToOne()to establish the relationship with the Breed entity. A cat will have a single breed, identified by the breed property on the Cat entity.
src\cats\entities\cat.entity.ts

ts
```ts
import { Breed } from "src/breeds/entities/breed.entity";
import {
  Column,
  DeleteDateColumn,
  Entity,
  ManyToOne,
  PrimaryGeneratedColumn,
} from "typeorm";

@Entity()
export class Cat {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  name: string;

  @Column()
  age: number;

  @ManyToOne(() => Breed, (breed) => breed.id, {
    // cascade: true,
    eager: true, // para que traiga las raza al hacer un findOne
  })
  breed: Breed;

  @DeleteDateColumn()
  deletedAt: Date;
}
```

-Create Breeds

src\breeds\dto\create-breed.dto.ts
ts
```ts
import { IsString, MinLength } from "class-validator";

export class CreateBreedDto {
  @IsString()
  @MinLength(1)
  name: string;
}
```

src\breeds\breeds.service.ts
ts
```ts
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
```

```
import { Repository } from 'typeorm';
import { CreateBreedDto } from './dto/create-breed.dto';
import { UpdateBreedDto } from './dto/update-breed.dto';
import { Breed } from './entities/breed.entity';

@Injectable()
export class BreedsService {
  constructor(
    @InjectRepository(Breed)
    private readonly breedsRepository: Repository<Breed>,
  ) {}

  async create(createBreedDto: CreateBreedDto) {
    const breed = this.breedsRepository.create(createBreedDto);
    return await this.breedsRepository.save(breed);
  }

  async findAll() {
    return await this.breedsRepository.find();
  }
```

Communication between modules

src\breeds\breeds.module.ts
ts

```
import { Module } from "@nestjs/common";
import { TypeOrmModule } from "@nestjs/typeorm";
import { BreedsController } from "./breeds.controller";
import { BreedsService } from "./breeds.service";
import { Breed } from "./entities/breed.entity";

@Module({
  imports: [TypeOrmModule.forFeature([Breed])],
  controllers: [BreedsController],
  providers: [BreedsService],
  exports: [TypeOrmModule],
})
export class BreedsModule {}
```

src\cats\cats.module.ts
ts

```
import { Module } from "@nestjs/common";
import { TypeOrmModule } from "@nestjs/typeorm";
import { BreedsModule } from "src/breeds/breeds.module";
import { BreedsService } from "src/breeds/breeds.service";
import { CatsController } from "./cats.controller";
import { CatsService } from "./cats.service";
```

```
import { Cat } from "./entities/cat.entity";

@Module({
  imports: [TypeOrmModule.forFeature([Cat]), BreedsModule],
  controllers: [CatsController],
  providers: [CatsService, BreedsService],
  exports: [],
})
export class CatsModule {}
```

Add Cat with Breed

src\cats\cats.service.ts
ts

```
import { BadRequestException, Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Breed } from 'src/breeds/entities/breed.entity';
import { Repository } from 'typeorm';
import { CreateCatDto } from './dto/create-cat.dto';
import { UpdateCatDto } from './dto/update-cat.dto';
import { Cat } from './entities/cat.entity';

@Injectable()
export class CatsService {
  constructor(
    @InjectRepository(Cat)
    private catsRepository: Repository<Cat>,

    @InjectRepository(Breed)
    private breedsRepository: Repository<Breed>,
  ) {}

  async create(createCatDto: CreateCatDto) {
    const breed = await this.breedsRepository.findOneBy({
      name: createCatDto.breed,
    });

    if (!breed) {
      throw new BadRequestException('Breed not found');
    }

    const cat = this.catsRepository.create({
      name: createCatDto.name,
      age: createCatDto.age,
      breed,
    });
    return await this.catsRepository.save(cat);
```

```
  }
```

Update BREAD

src\cats\cats.service.ts

ts

```
async update(id: number, updateCatDto: UpdateCatDto) {
  const cat = await this.catsRepository.findOneBy({ id });

  if (!cat) {
    throw new BadRequestException('Cat not found');
  }

  let breed;
  if (updateCatDto.breed) {
    breed = await this.breedsRepository.findOneBy({
      name: updateCatDto.breed,
    });

    if (!breed) {
      throw new BadRequestException('Breed not found');
    }
  }

  return await this.catsRepository.save({
    ...cat,
    ...updateCatDto,
    breed,
  });
}
```