Nest.js : API REST

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en Programmation par contrainte (IA)
Ingénieur en Génie logiciel

elmouelhi.achref@gmail.com

Plan

Plan

Nest.js

Pour créer un nouveau module

```
nest generate module module-name
```

Nest.js

Pour créer un nouveau module

```
nest generate module module-name
```

Ou

```
nest g mo module-name
```

Nest.js

Pour créer un nouveau module

```
nest generate module module-name
```

Ou

```
nest g mo module-name
```


Pour le placer dans un répertoire `services`

```
nest g mo modules/module-name
```

Nest.js

Créons un module `personne`

```
nest g mo modules/personne
```

Nest.js

Créons un module `personne`

```
nest g mo modules/personne
```

Résultat

```
CREATE src/modules/personne/personne.module.ts (85 bytes)
UPDATE src/app.module.ts (332 bytes)
```

Nest.js

Explication

- Un fichier `personne.module.ts` a été créé dans `src/modules`
- Ce module a été déclaré dans `app.module.ts`


```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { PersonneModule } from './modules/personne/personne.module';

@Module({
  imports: [PersonneModule],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

Nest.js

Pour créer une classe

```
nest generate class class-name
```

Nest.js

Pour créer une classe

```
nest generate class class-name
```

Ou le raccourci

```
nest g cl class-name
```

Nest.js

Pour créer une classe

```
nest generate class class-name
```

Ou le raccourci

```
nest g cl class-name
```


Pour créer une classe sans la placer dans un répertoire portant le même nom

```
nest g cl class-name --flat
```

Nest.js

Pour créer une classe sans le fichier `spec`

```
nest g cl entities/class-name --no-spec --flat
```

Nest.js

Pour créer une classe sans le fichier `spec`

```
nest g cl entities/class-name --no-spec --flat
```

Créons une classe `Personne` **dans le module** `personne`

```
nest g cl modules/personne/personne.entity --no-spec --flat
```

Nest.js

Pour créer une classe sans le fichier `spec`

```
nest g cl entities/class-name --no-spec --flat
```

Créons une classe `Personne` **dans le module** `personne`

```
nest g cl modules/personne/personne.entity --no-spec --flat
```


Résultat

```
CREATE src/modules/personne/personne.entity.ts (31 bytes)
```

Nest.js

Remplaçons le contenu généré dans `personne.entity.ts`

```
export class PersonneEntity { }
```

Nest.js

Remplaçons le contenu généré dans `personne.entity.ts`

```
export class PersonneEntity { }
```

Par

```
export class Personne {
    num: number;
    nom: string;
    prenom: string;
}
```

Nest.js

Pour créer un nouveau service

```
nest generate service service-name
```

Nest.js

Pour créer un nouveau service

```
nest generate service service-name
```

Ou

```
nest g s service-name
```

Nest.js

Pour créer un nouveau service

```
nest generate service service-name
```

Ou

```
nest g s service-name
```

Pour créer un service sans le placer dans un répertoire portant le même nom

```
nest g s service-name --flat
```

Nest.js

Pour créer un nouveau service sans le fichier `.spec`

```
nest g s service-name --no-spec --flat
```

Nest.js

Pour créer un nouveau service sans le fichier `.spec`

```
nest g s service-name --no-spec --flat
```

Créons un service `personne` **dans le module** `personne`

```
nest g s modules/personne/personne --no-spec --flat
```

Nest.js

Résultat

```
CREATE src/modules/personne/personne.service.ts (92 bytes)
UPDATE src/modules/personne/personne.module.ts (171 bytes)
```

Nest.js

Résultat

```
CREATE src/modules/personne/personne.service.ts (92 bytes)
UPDATE src/modules/personne/personne.module.ts (171 bytes)
```


Explication

- Un fichier `personne.service.ts` a été créé dans `src/services`
- Ce service a été déclaré dans `personne.module.ts`

```
import { Module } from '@nestjs/common';
import { PersonneService } from './personne.service';

@Module({
  providers: [PersonneService]
})
export class PersonneModule {}
```

Nest.js

Contenu de `personne.service.ts`

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

@Injectable()
export class PersonneService {}
```

Nest.js

Mettons à jour le contenu de `personne.service.ts`

```
import { Injectable } from '@nestjs/common';
import { Personne } from './personne.entity';

@Injectable()
export class PersonneService {

    personnes: Personne[] = [
        { num: 1, nom: 'Wick', prenom: 'John' },
        { num: 2, nom: 'Dalton', prenom: 'Jack' },
        { num: 3, nom: 'Maggio', prenom: 'Carol' }
    ]

}
```

Nest.js

Ajoutons aussi les méthodes suivantes

```
import { Injectable } from '@nestjs/common';
import { Personne } from './personne.entity';

@Injectable()
export class PersonneService {

    personnes: Personne[] = [
        { num: 1, nom: 'Wick', prenom: 'John' },
        { num: 2, nom: 'Dalton', prenom: 'Jack' },
        { num: 3, nom: 'Maggio', prenom: 'Carol' }
    ]

    findAll(): Array<Personne> {
        return this.personnes;
    }

    findOneById(num: number): Personne {
        return this.personnes.find(elt => elt.num == num);
    }

    addPerson(p: Personne): void {
        this.personnes.push(p);
    }
}
```

Nest.js

Pour créer un nouveau contrôleur

```
nest generate controller controller-name
```

Nest.js

Pour créer un nouveau contrôleur

```
nest generate controller controller-name
```

Ou

```
nest g co controller-name
```

Nest.js

Pour créer un nouveau contrôleur

```
nest generate controller controller-name
```

Ou

```
nest g co controller-name
```


Pour créer un contrôleur sans le placer dans un répertoire portant le même nom

```
nest g co controller-name --flat
```

Nest.js

Pour créer un nouveau contrôleur sans le fichier `.spec`

```
nest g co controllers/controller-name --no-spec --flat
```

Nest.js

Pour créer un nouveau contrôleur sans le fichier `.spec`

```
nest g co controllers/controller-name --no-spec --flat
```

Créons un contrôleur `personne`

```
nest g co modules/personne/personne --no-spec --flat
```

Nest.js

Résultat

```
CREATE src/modules/personne/personne.controller.ts (105 bytes)
UPDATE src/modules/personne/personne.module.ts (268 bytes)
```

Nest.js

Résultat

```
CREATE src/modules/personne/personne.controller.ts (105 bytes)
UPDATE src/modules/personne/personne.module.ts (268 bytes)
```


Explication

- Un fichier `personne.controller.ts` a été créé dans `modules/personne`
- Ce contrôleur a été déclaré dans `personne.module.ts`

```
@Module({
  providers: [PersonneService],
  controllers: [PersonneController]
})
export class PersonneModule {}
```

Nest.js

Contenu de `personne.controller.ts`

```
import { Controller } from '@nestjs/common';

@Controller('personne')
export class PersonneController {}
```

Nest.js

Injectons `PersonneService` **dans** `PersonneController`

```
import { Controller } from '@nestjs/common';
import { PersonneService } from './personne.service';

@Controller('personne')
export class PersonneController {

    constructor(private personneService: PersonneService) { }

}
```

Nest.js

Définissons une méthode dans le contrôleur qui permet de retourner la liste de personnes

```
import { Controller } from '@nestjs/common';
import { PersonneService } from './personne.service';

@Controller('personne')
export class PersonneController {

    constructor(private personneService: PersonneService) { }

    findAll() {
        return this.personneService.findAll();
    }

}
```

Nest.js

Spécifions le verbe HTTP permettant l'accès à cette méthode

```
import { Controller, Get } from '@nestjs/common';
import { PersonneService } from './personne.service';

@Controller('personne')
export class PersonneController {

    constructor(private personneService: PersonneService) { }

    @Get()
    findAll() {
        return this.personneService.findAll();
    }
}
```

Nest.js

Spécifions le verbe HTTP permettant l'accès à cette méthode

```
import { Controller, Get } from '@nestjs/common';
import { PersonneService } from './personne.service';

@Controller('personne')
export class PersonneController {

    constructor(private personneService: PersonneService) { }

    @Get()
    findAll() {
        return this.personneService.findAll();
    }
}
```


Pour tester, allez à l'URL `localhost:3000/personne` et vérifiez la récupération des trois personnes au format **JSON**

Nest.js

Pour retourner une personne selon l'identifiant, il faut définir ce dernier comme paramètre en le préfixant par :

```
import { Controller, Get } from '@nestjs/common';
import { PersonneService } from './personne.service';

@Controller('personne')
export class PersonneController {

    constructor(private personneService: PersonneService) { }

    @Get()
    findAll() {
        return this.personneService.findAll();
    }

    @Get(':id')
    findById(id: number) {
        return this.personneService.findOneById(id);
    }
}
```

Nest.js

Pour retourner une personne selon l'identifiant, il faut définir ce dernier comme paramètre en le préfixant par :

```
import { Controller, Get } from '@nestjs/common';
import { PersonneService } from './personne.service';

@Controller('personne')
export class PersonneController {

    constructor(private personneService: PersonneService) { }

    @Get()
    findAll() {
        return this.personneService.findAll();
    }

    @Get(':id')
    findById(id: number) {
        return this.personneService.findOneById(id);
    }
}
```

En allant à l'URL `localhost:3000/personne`, one ne récupère pas la personne.

Nest.js

Pour récupérer le paramètre, il faut indiquer à nest qu'il s'agit d'un paramètre avec le décorateur `@Param`

```
import { Controller, Get, Param } from '@nestjs/common';
import { PersonneService } from './personne.service';

@Controller('personne')
export class PersonneController {

    constructor(private personneService: PersonneService) { }

    @Get()
    findAll() {
        return this.personneService.findAll();
    }

    @Get(':id')
    findById(@Param('id') id: number) {
        return this.personneService.findOneById(id);
    }

}
```

Nest.js

Pour récupérer le paramètre, il faut indiquer à nest qu'il s'agit d'un paramètre avec le décorateur `@Param`

```
import { Controller, Get, Param } from '@nestjs/common';
import { PersonneService } from './personne.service';

@Controller('personne')
export class PersonneController {

    constructor(private personneService: PersonneService) { }

    @Get()
    findAll() {
        return this.personneService.findAll();
    }

    @Get(':id')
    findById(@Param('id') id: number) {
        return this.personneService.findOneById(id);
    }

}
```

Pour tester, allez à l'URL `localhost:3000/personne/1` et vérifiez la récupération des la personne ayant le `num` 1 au format **JSON**

Nest.js

Remarque

Pour un meilleur affichage de données au format **JSON** avec **Google Chrome**, installez l'extension **Json Formatter**.

Jersey

Utilisation de **Postman** pour tester : simulateur d'un client **REST**

- Aller sur internet et chercher **Postman**
- Télécharger puis installer l'application **Postman**
- Démarrer l'application

Jersey

Utilisation de **Postman** pour tester : simulateur d'un client **REST**

- Aller sur internet et chercher **Postman**
- Télécharger puis installer l'application **Postman**
- Démarrer l'application

Il existe plusieurs autres tel que ARC (pour Advanced REST Client)...

Nest.js

Pour tester avec **Postman**

- Dans la liste déroulante, choisir `GET`
- Saisir l'URL de notre web service : `http://localhost:3000/personne/1`
- Cliquer sur `Send`

Nest.js

Spécifions le verbe HTTP permettant l'accès à cette méthode

```
import { Controller, Get, Param, Post } from '@nestjs/common';
import { PersonneService } from './personne.service';
import { Personne } from './personne.entity';

@Controller('personne')
export class PersonneController {

    constructor(private personneService: PersonneService) { }

    @Get()
    findAll() {
        return this.personneService.findAll();
    }

    @Get(':id')
    findById(@Param('id') id: number) {
        return this.personneService.findOneById(id);
    }

    @Post()
    create(personne: Personne) {
        console.log(personne)
        this.personneService.addPerson(personne);
        return personne;
    }
}
```

Nest.js

Pour tester avec **Postman**

- Dans la liste déroulante, choisir `POST`
- saisir l'URL vers notre web service`http://localhost:3000/personne`
- Ensuite cliquer sur `Body`, cocher `raw`, choisir `JSON (application/json)` et saisir des données sous format `json` correspondant à l'objet personne à ajouter

```
{
    "nom": "Morena",
    "prenom": "Andreas"
}
```

- Cliquer sur `Send`

Nest.js

Résultat

- Erreur et la personne n'a pas été ajoutée dans la base de données
- Message affiché dans la console : `undefined`

Nest.js

Pour récupérer l'objet défini dans le corps de la requête `HTTP` et assurer le binding avec l'objet défini comme paramètre de la méthode `create()`, on doit utiliser le décorateur `@Body`

```
import { Body, Controller, Get, Param, Post } from '@nestjs/common';
import { PersonneService } from './personne.service';
import { Personne } from './personne.entity';

@Controller('personne')
export class PersonneController {

    constructor(private personneService: PersonneService) { }

    @Get()
    findAll() {
        return this.personneService.findAll();
    }

    @Get(':id')
    findById(@Param('id') id: number) {
        return this.personneService.findOneById(id);
    }

    @Post()
    create(@Body() personne: Personne) {
        console.log(personne)
        this.personneService.addPerson(personne);
        return personne;
    }
}
```

Nest.js

Quels paquets ?

- `mysql2` : pour le driver **MySQL**
- `typeorm` : **ORM** pour **Node.js**

Nest.js

Pour installer `mysql2`

```
npm install --save mysql2
```

Nest.js

Pour installer `mysql2`

```
npm install --save mysql2
```

Pour installer `sequelize`

```
npm install --save typeorm
```

Nest.js

Définissons `Personne` comme entité

```
import { Entity, Column, PrimaryGeneratedColumn } from 'typeorm';

@Entity()
export class Personne {

    @PrimaryGeneratedColumn()
    num: number;

    @Column()
    nom: string;

    @Column()
    prenom: string;
}
```

Nest.js

Créons un module `database`

```
nest g mo modules/database
```

Nest.js

Créons un module `database`

```
nest g mo modules/database
```

Résultat

```
CREATE src/modules/database/database.module.ts (85 bytes)
UPDATE src/app.module.ts (417 bytes)
```

Nest.js

Contenu de `database.module.ts`

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

@Module({
  providers: []
})
export class DatabaseModule {}
```

Nest.js

Créons un provider `database`

```
nest g pr modules/database/database.provider --flat --no-spec
```

Nest.js

Créons un provider `database`

```
nest g pr modules/database/database.provider --flat --no-spec
```

Résultat

```
CREATE src/modules/database/database.provider.ts (93 bytes)
UPDATE src/modules/database/database.module.ts (174 bytes)
```

Nest.js

Nouveau contenu de `database.module.ts`

```
import { Module } from '@nestjs/common';
import { DatabaseProvider } from './database.provider';

@Module({
  providers: [DatabaseProvider]
})
export class DatabaseModule {}
```

Nest.js

Modifions le contenu de `database.providers.ts` en ajoutant les données de connexion à la base de données

```
import { DataSource } from 'typeorm';
import { Personne } from '../personne/personne.entity';

export const databaseProviders = [
  {
    provide: 'DATA_SOURCE',
    useFactory: async () => {
      const dataSource = new DataSource({
        type: 'mysql',
        host: 'localhost',
        port: 3306,
        username: 'root',
        password: '',
        database: 'cours_nest',
        entities: [Personne],
        synchronize: true,
      });

      return dataSource.initialize();
    },
  },
];
```

Nest.js

Mettons à jour le contenu de `database.module.ts`

```
import { Module } from '@nestjs/common';
import { databaseProviders } from './database.provider';

@Module({
  providers: [...databaseProviders],
  exports: [...databaseProviders],
})
export class DatabaseModule {}
```

Nest.js

Déclarons `database.module.ts` **dans** `personne.module.ts`

```
import { DataSource } from "typeorm";
import { Module } from '@nestjs/common';
import { PersonneController } from './personne.controller';
import { PersonneService } from './personne.service';

@Module({
  providers: [PersonneService],
  controllers: [PersonneController]
})
export class PersonneModule { }
```

Nest.js

Définissons un provider (de repository) pour l'entité `Personne` que nous appelons `personne.provider.ts`

```
import { DataSource } from "typeorm";
import { Personne } from "./personne.entity";

export const personneProvider = [
    {
        provide: 'PERSONNE_REPOSITORY',
        useFactory: (dataSource: DataSource) => dataSource.
          getRepository(Personne),
        inject: ['DATA_SOURCE'],
    },
];
```

Nest.js

Définissons un provider (de repository) pour l'entité `Personne` que nous appelons `personne.provider.ts`

```
import { DataSource } from "typeorm";
import { Personne } from "./personne.entity";

export const personneProvider = [
    {
        provide: 'PERSONNE_REPOSITORY',
        useFactory: (dataSource: DataSource) => dataSource.
          getRepository(Personne),
        inject: ['DATA_SOURCE'],
    },
];
```

Le repository associé à l'entité `Personne` aura comme nom `PERSONNE_REPOSITORY`.

Nest.js

Déclarons ce provider dans `personne.module.ts`

```
import { Module } from '@nestjs/common';
import { PersonneController } from './personne.controller';
import { PersonneService } from './personne.service';
import { personneProvider } from './personne.provider';
import { DatabaseModule } from '../database/database.module';

@Module({
    imports: [DatabaseModule],
    providers: [PersonneService, ...personneProvider],
    controllers: [PersonneController]
})
export class PersonneModule { }
```

Nest.js

Pour utiliser le repository dans le service, il faut l'injecter

```
import { Inject, Injectable } from '@nestjs/common';
import { Personne } from './personne.entity';
import { Repository } from 'typeorm';

@Injectable()
export class PersonneService {

    constructor(
        @Inject('PERSONNE_REPOSITORY')
        private personneRepository: Repository<Personne>
    ) { }

    async findAll() {
        return this.personneRepository.find();
    }

    async findOneById(num: number) {
        return this.personneRepository.findOneBy({ num: num });
    }

    async addPerson(p: Personne) {
        return this.personneRepository.save(p);
    }
}
```

Nest.js

Commençons par installer les dépendances suivantes

```
npm i --save class-validator class-transformer
```

Nest.js

Activons la validation de données dans `main.ts`

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

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.useGlobalPipes(new ValidationPipe());
  await app.listen(3000);
}
bootstrap();
```

Nest.js

Ajoutons les validateurs suivants aux attributs de la classe `Personne`

```
import { IsNotEmpty, MaxLength, MinLength } from 'class-validator';
import { Entity, Column, PrimaryGeneratedColumn } from 'typeorm';

@Entity()
export class Personne {

    @PrimaryGeneratedColumn()
    num: number;

    @MinLength(3)
    @MaxLength(30)
    @Column()
    nom: string;

    @Column()
    @IsNotEmpty()
    prenom: string;
}
```

Jersey

Autres décorateurs de validation pour le type `number`

- `@Min()` et `@Max()`
- `@IsDivisibleBy(num)`
- `@IsPositive()` et `@IsNegative()`

Jersey

Autres décorateurs de validation pour le type `string`

- `@Length()`
- `@Contains()` et `@NotContains()`
- `@IsAlpha()`
- `@IsAlphanumeric()`
- `@IsISBN()`
- `@IsIBAN()` et `@IsBIC()`
- `@IsUppercase()` et `@IsLowercase()`
- `@Matches()`
- ...

Jersey

Autres décorateurs (commun) de validation

- `@Equals()` et `@NotEquals()`
- `@@IsIn(values: any[])` et `@IsNotIn(values: any[])`
- `@IsDefined()`
- `@IsNotEmpty()`
- ...

Jersey

Autres décorateurs de validation

- `@ValidateIf(arrow_function)`
- `@IsBoolean()`
- `@IsEmail()`
- `@IsDate()`
- `@MinDate()` et `@MaxDate()`

Jersey

Autres décorateurs de validation

- `@ValidateIf(arrow_function)`
- `@IsBoolean()`
- `@IsEmail()`
- `@IsDate()`
- `@MinDate()` et `@MaxDate()`

Liste complète (documentation officielle)

https://www.npmjs.com/package/class-validator

Nest.js

Pour tester avec **Postman**

- Dans la liste déroulante, choisir `POST`
- saisir l'URL vers notre web service`http://localhost:3000/personne`
- Ensuite cliquer sur `Body`, cocher `raw`, choisir `JSON (application/json)` et saisir des données sous format `json` correspondant à l'objet personne à ajouter

```
{
    "nom": "Do",
    "prenom": "John"
}
```

- Cliquer sur `Send`

Nest.js

En envoyant la requête précédente, vérifiez qu'on reçoit la réponse suivante

```
{
    "statusCode": 400,
    "message": [
        "nom must be longer than or equal to 3
           characters"
    ],
    "error": "Bad Request"
}
```

Nest.js

Personnalisons les messages d'erreur

```
import { IsNotEmpty, MaxLength, MinLength } from 'class-validator';
import { Entity, Column, PrimaryGeneratedColumn } from 'typeorm';

@Entity()
export class Personne {

    @PrimaryGeneratedColumn()
    num: number;

    @MinLength(3, { message: 'Ce champ doit contenir au moins 3 caractères' })
    @MaxLength(30, { message: 'Ce champ doit contenir au plus 30 caractères' })
    @Column()
    nom: string;

    @Column()
    @IsNotEmpty({ message: 'Ce champ ne doit pas être vide' })
    prenom: string;
}
```

Nest.js

En envoyant la requête précédente, vérifiez qu'on reçoit le message d'erreur personnalisé

```
{
    "statusCode": 400,
    "message": [
        "Ce champ doit contenir au moins 3 caractè
          res"
    ],
    "error": "Bad Request"
}
```

Nest.js

On peut intégrer le nom du champ et la valeur reçue dans le message d'erreur

```
import { IsNotEmpty, MaxLength, MinLength } from 'class-validator';
import { Entity, Column, PrimaryGeneratedColumn } from 'typeorm';

@Entity()
export class Personne {

    @PrimaryGeneratedColumn()
    num: number;

    @MinLength(3, { message: '$property est trop court. Minimum autorisé $constraint1 caractè
       res, mais valeur reçue $value' })
    @MaxLength(30, { message: '$property est trop long. Maximum autorisé $constraint1 caractè
       res, mais valeur reçue $value' })
    @Column()
    nom: string;

    @Column()
    @IsNotEmpty({ message: '$property ne doit pas être vide' })
    prenom: string;
}
```

Nest.js

En envoyant la requête précédente, vérifiez qu'on reçoit le message d'erreur personnalisé

```
{
    "statusCode": 400,
    "message": [
        "nom est trop court. Minimum autorisé 3
           caractères, mais valeur reçue Do"
    ],
    "error": "Bad Request"
}
```

Nest.js

Pour désactiver l'affichage des messages d'erreur, on modifie `main.ts`

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

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.useGlobalPipes(
    new ValidationPipe({
      disableErrorMessages: true,
    }),
  );
  await app.listen(3000);
}
bootstrap();
```

Nest.js

Problématique

- Si nous envoyons une requête **HTTP** de type **GET** à l'URL `http://localhost:3000/personne/1`, nous obtiendrons un objet **JSON** contenant des informations sur la personne ayant le numéro `1`.
- Si nous demandons des informations sur une personne qui n'existe pas (par exemple numéro `10000`), nous n'obtiendrons pas de réponse mais un statut **200 OK**.
- Ce type de retour peut avoir des graves conséquences au récepteur.
- Nous préférons générer une exception pour un tel cas.

Nest.js

Modifions la méthode `findById`

```
@Get(':id')
async findById(@Param('id') id: number) {
    const p = await this.personneService.findOneById(id)
    if (p == null || p == undefined) {
        throw new HttpException(
            `Aucune personne avec l'identifiant ${id}`,
            HttpStatus.NOT_FOUND
        );
    }
    return p
}
```

Nest.js

En envoyant la requête précédente, vérifiez qu'on reçoit la réponse suivante

```
{
    "statusCode": 404,
    "message": "Aucune personne avec l'identifiant 1000"
}
```