

Nest.js : introduction

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

- 1 Introduction
- 2 Installation
- 3 Création d'un nouveau projet
- 4 Structure d'un projet Nest.js

Nest.js

Nest.js ou Nest

- Framework **Node.js**
- Supportant intégralement le langage **TypeScript**
- Permettant de :
 - construire des applications web
 - simplifier la création et la gestion des serveurs **Node.js**
 - faciliter le routage

Nest.js

Autres frameworks **Node.js**

- Ionic
- Express.js
- ...

Nest.js

Pour installer Nest.js, il faut

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

Nest.js

Pour créer un nouveau projet

```
nest new cours-nest
```

© Achref EL MOUELHI ©

Nest.js

Pour créer un nouveau projet

```
nest new cours-nest
```

Ou

```
nest n cours-nest
```

Nest.js

Pour créer un nouveau projet

```
nest new cours-nest
```

Ou

```
nest n cours-nest
```

Répondre à la question suivante

```
Which package manager would you like to use? (npm)
```

Ensuite allez dans le répertoire du projet

```
cd cours-nest
```

© Achref EL MOUELHI ©

Ensuite allez dans le répertoire du projet

```
cd cours-nest
```

Ouvrez Visual Studio Code

```
code .
```

© Achref EL MOU

Ensuite allez dans le répertoire du projet

```
cd cours-nest
```

Ouvrez Visual Studio Code

```
code .
```

Lancez le projet

```
npm run start
```

Ensuite allez dans le répertoire du projet

```
cd cours-nest
```

Ouvrez Visual Studio Code

```
code .
```

Lancez le projet

```
npm run start
```

Et pour avoir plus de détails

```
npm run start:dev
```

Nest.js

Arborescence d'un projet **Nest.js**

- `node_modules` : contenant les fichiers nécessaires de la librairie **Node.js** pour un projet **Nest.js**
- `src` : contenant les fichiers sources de l'application
- `package.json` : contenant l'ensemble de dépendance de l'application
- `test` : contenant les fichiers nécessaires pour lancer les tests
- `dist` : contenant la version à déployer
- `tsconfig.json` : fichier de configuration de **TypeScript**
- `nest-cli.json` : fichier de configuration de **nest-cli**

Que contient `src` ?

- `main.ts` : Le point d'entrée de l'application
- `app.module.ts` : classe correspondante au module principal
- `app.controller.ts` : classe correspondant au contrôleur dans le modèle **MVC**
- `app.service.ts` : classe service
- `app.controller.spec.ts` : contenant le code de test de `app.controller.ts`

Nest.js

Contenu de main.ts

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

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

Contenu de `app.module.ts`

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

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

© Achref EL M...

Contenu de `app.module.ts`

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

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

Explication

- `@Module` : pour déclarer cette classe comme module
- `imports` : dans cette section, on déclare les modules nécessaires pour le module principal
- `providers` : dans cette section, on déclare les services qui seront utilisés dans le module
- `controllers` : dans cette section, on déclare les contrôleurs de ce module

Nest.js

Contenu de `app.service.ts`

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

@Injectable()
export class AppService {
  getHello(): string {
    return 'Hello World!';
  }
}
```

© Activo

Nest.js

Contenu de `app.service.ts`

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

@Injectable()
export class AppService {
  getHello(): string {
    return 'Hello World!';
  }
}
```

Explication

`@Injectable` : pour déclarer cette classe comme un service injectable dans le constructeur

Contenu de `app.module.ts`

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

@Controller()
export class AppController {
  constructor(private readonly appService: AppService) {}

  @Get()
  getHello(): string {
    return this.appService.getHello();
  }
}
```

© Achre

Contenu de `app.module.ts`

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

@Controller()
export class AppController {
  constructor(private readonly appService: AppService) {}

  @Get()
  getHello(): string {
    return this.appService.getHello();
  }
}
```

Explication

- `@Controller` : pour déclarer cette classe comme contrôleur
- `@Get` : pour préciser le verbe **HTTP** permettant l'accès à cette méthode
- Dans le constructeur, le service `AppService` a été injecté