NestJs

▼ Creation d’un project NestJS

Étape 1: Installer Node.js

- **Téléchargement et Installation**
  - Visitez https://nodejs.org/ pour télécharger l'installateur.
  - Suivez les instructions pour installer Node.js sur votre système.

Étape 2: Initialiser un Nouveau Projet

- **Création d'un Nouveau Dossier**

```
mkdir mon-projet
cd mon-projet
```

- **Initialisation du Projet**

```
nest new nom-du-projet
```

- Répondez aux questions pour créer un fichier `package.json`.

Structure du Projet

- **src/**: Contient le code source de l'application.
  - **app.controller.ts**: Contrôleur de base.
  - **app.module.ts**: Module racine de l'application.
  - **app.service.ts**: Service de base.

        - **main.ts**: Point d'entrée de l'application.
    - **test/**: Contient les tests.
    - **nest-cli.json**: Configuration CLI de NestJS.
    - **package.json**: Dépendances et scripts.

Ajout de Modules, Contrôleurs et Services

Création d'un Module

```
nest g module nom-du-module
```

Création d'un Contrôleur

```
nest g controller nom-du-contrôleur
```

Création d'un Service

```
nest g service nom-du-service
```

▼ Controlleur

Controllers

- **Définition** : Les contrôleurs sont responsables de la gestion des requêtes entrantes et du renvoi des réponses au client.

Création d'un Contrôleur

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

@Controller('nomRoute')
export class NomController {
  @Get()
```

```
  findAll(): string {
    return 'Retourne quelque chose';
  }
}
```

- `@Controller('nomRoute')` : Décorateur pour définir un contrôleur. `'nomRoute'` est le chemin de base.
- `@Get()` : Décorateur pour une méthode qui gère les requêtes GET. Peut être remplacé par `@Post()`, `@Put()`, `@Delete()`, etc.

Paramètres de Route

- **Paramètres statiques** : `/chemin/fixe`
- **Paramètres dynamiques** :

```
@Get(':id')
findOne(@Param('id') id: string): string {
    return `Retourne l'élément ${id}`;
}
```

- `@Param('id')` : Extrait le paramètre `id` de l'URL.

Corps de la Requête (Request Body)

- Pour accéder au corps de la requête :

```
@Post()
create(@Body() createDto: CreateDto): string {
    return 'Ajoute un élément';
}
```

- `@Body()` : Extrait le corps de la requête.

Routage

- **Routage Basique** :

```
@Get('chemin/sous-route')
findInSubRoute(): string {
    return 'Retourne quelque chose depuis une sous-rout
e';
}
```

- **Routage Avancé** :
  - Routage avec des paramètres :

```
@Get('users/:userId/books/:bookId')
findBook(@Param() params): string {
  return `User: ${params.userId}, Book: ${params.boo
kId}`;
}
```

▼ Service

Introduction aux Services

- **Définition** : Les services dans NestJS sont des classes qui encapsulent la logique métier. Ils sont souvent utilisés pour l'accès aux données.
- **Injection de Dépendance** : Les services sont injectables et peuvent être utilisés dans des contrôleurs ou d'autres services.

Création d'un Service

- **Commande CLI** : Utilisez `nest g service <nom_service>` pour générer un service.
- **Structure de Base** :

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

@Injectable()
export class NomService {
  // Méthodes et propriétés
}
```

Injection de Services

- **Dans un Contrôleur** :

```
import { NomService } from './nom.service';

    @Controller('nom')
    export class NomController {
        constructor(private nomService: NomService) {}
}
```

- **Dans un Autre Service** :

```
import { Injectable } from '@nestjs/common';
import { AutreService } from './autre.service';

@Injectable()
export class NomService {
    constructor(private autreService: AutreService) {}
}
```

▼ MongoDB

Recuperer la chaine de connection Mongodb

installation de mongoose:

```
npm install @nestjs/mongoose mongoose
```

mettre la chaine de connection dans le module racine de notre project

```
@Module({
  imports: [
    MongooseModule.forRoot('mongodb+srv://christoloisel:Rose2
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

- Creation du Modele de donnée afin de venir representer ma Data

  fichier model :

  ```
  // Creer mon interface utilisateur
  export interface IUser
  {
      name: string;
      age: number;
  }
  ```

  Creer le Schema qui viendra representer mon modele au seins de MongoDB :

```
import { Schema } from 'mongoose';

export const UserSchema: Schema = new Schema({
    name: { type: String, required: true },
    age: { type: Number, required: true },
});
```

Lier le schema au model pour pouvoir l'injecter dans les services qui devront l'utiliser :

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

@Module({
  imports: [
    MongooseModule.forRoot('mongodb+srv://christoloisel:R
    MongooseModule.forFeature([{ name: 'User', schema: Use
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

Recuperer les instance de mon Model de donner en les injectant dans mes constructeur grace a la balise `InjectModel` :

```
constructor(@InjectModel('User') private readonly userMode
```

Utiliser les fonction de Modification au seins de la BDD MongoDB

- Create :

```
// addUser
    addUser(user : IUser): Promise<IUser>
    {
        const newUser = new this.userModel(user);
        return newUser.save();
    }
```

- Find :

```
// getALLUser
    getAllUser(): Promise<IUser[]>
    {
        // Recuperer les utilisateurs par les parametre du
        // pas de parametre = tous les utilisateurs
        // un exemple de parametre : {name: 'toto'}
        // return this.userModel.find({name: 'toto'}).exec
        // -> retourne tous les utilisateurs qui s'appelle
        return this.userModel.find().exec();
    }

    // getUser
    getUser(id: string): Promise<IUser>
    {
        return this.userModel.findById(id).exec();
    }
```

- Update :

```
// updateUser
    updateUser(id: string, user: IUser): Promise<IUser>
    {
        // L'id en question sera toujours le _id et pas Vo
        return this.userModel.findByIdAndUpdate(id, user,
    }
```

- Delete :

```
// deleteUser
    deleteUser(id: string): Promise<any>
    {
        // L'id en question sera toujours le _id et pas Vo
        return this.userModel.findByIdAndDelete(id).exec(
    }
```

▼ Model