Para começar a trabalhar com o NestJS, precisamos instalar a Nest CLI, é uma ferramenta de interface de linha de comando que o ajudará a manter seus Aplicativos Nest, ele auxilia de maneiras diferentes, por exemplo, andaimes do projeto, atendendo aos aplicativos em modo de desenvolvimento e criando o aplicativo para distribuições de produção.

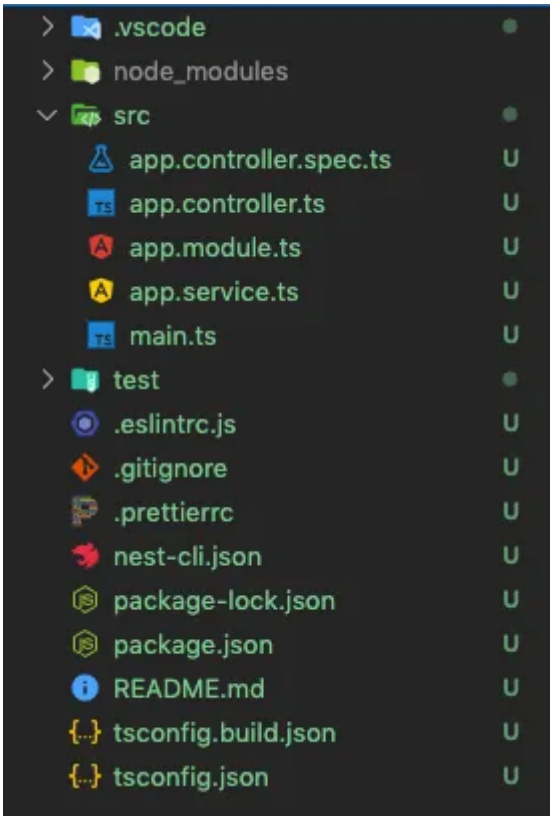
Então, vamos começar a criar nosso projeto com o próximo comando.

$ npm instalar -g ?nestjs/cli

Para criar um projeto com o NestJS CLI, execute o comando a seguir, o comando vai criar um novo diretório com a configuração inicial do núcleo.

$ ninho novo projeto-nome

Podemos encontrar arquivos principais no `src/`Diretório.

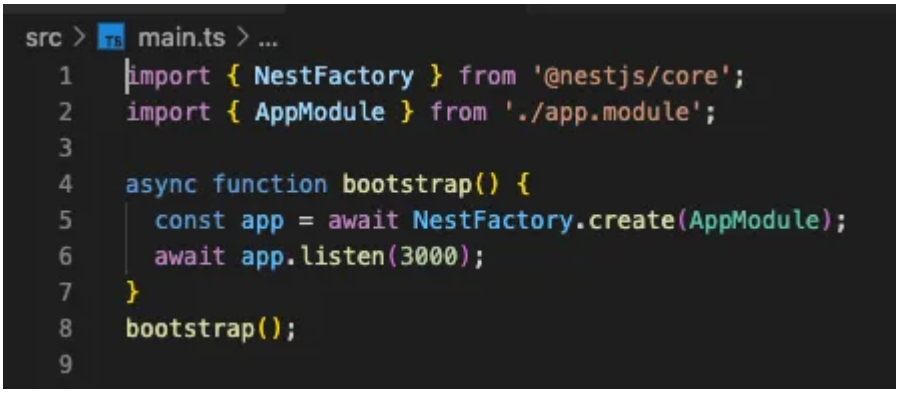


app.controller.ts: Um controlador básico com um caminho simples.

app.service.ts: Um serviço simples.

app.module.ts: O módulo raiz da aplicação.

main.ts: O arquivo de entrada do aplicativo que usa a função principal `NesteFactory para` criar uma instância de aplicação Nest.



Para iniciar o formulário, digite os comandos do ninho.

$ cd projeto-nome
$ npm install
Início da corrida de $ npm

Agora você pode abrir seu navegador e navegar para http://localhost:3000.

ok, agora vamos criar um módulo relacionado a chaves

$ nest generate module chave

O comando adiciona o módulo do chave no módulo de aplicação principal.
Para criar um controlador básico, usamos o próximo comando:

$ nest generate controller /chave/controller

Nosso controlador foi cadastrado no módulo do chave

Um controlador é uma classe simples com o decorador @Controller('chave')que é necessário para definir um controlador e especifica o prefixo chave, o que nos permite agrupar facilmente um conjunto de rotas relacionadas e minimizar o código repetitivo.

Para lidar com os diferentes métodos, o NestJS nos fornece métodos: @Get, @Post, @Put(), @Delete(), @Patch() e há outro decorador que lida com todos eles @All() .

Os serviços são importantes porque são responsáveis pelo armazenamento e recuperação dos dados, o serviço foi projetado para ser utilizado pelo controlador, então vamos criar um serviço básico com o próximo comando.

$ nest generate service /chave/service/chave

Agora dentro do modulo de chaves crie uma pasta chamada module e crie um arquivo chave.module.ts. esse arquivo será responsável pela integridade do banco garantindo seu estrutura

import * as mongoose from 'mongoose';

```
export const ChaveSchema = new mongoose.Schema({
  nome: String,
  situacao: String,
});

export interface Chave extends mongoose.Document {
  nome: string;
  situacao: string;
}
```

agora vamos para o service ja criado que sera responsavel pela regra de negocios e usar o model para definir regras de estrutura de envio

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { Model } from 'mongoose';
import { InjectModel } from '@nestjs/mongoose';
import { Chave } from '../../chave.model/chave.model';

@Injectable()
export class ChaveService {
  constructor(@InjectModel('Chave') private readonly chaveModel: Model<Chave>) {}

  async criarChave(nome: string, situacao: string): Promise<Chave> {
        const chave = new this.chaveModel({ nome, situacao });
        return await chave.save();
  }

  async listarChaves(): Promise<Chave[]> {
        return await this.chaveModel.find().exec();
  }
}
```

agora vamos para o controller já criado que será responsável pelas nossas rotas usara e chamará o service

```
import { Body, Controller, Delete, Get, Param, Post, Put } from '@nestjs/common';
import { ChaveService } from '../service/chave/chave.service';
import { Chave } from '../chave.model/chave.model';

@Controller('chaves')
export class ChaveController {
  constructor(private readonly chaveService: ChaveService) {}

  @Post()
  async criarChave(@Body() data: Chave): Promise<Chave> {
        console.log(data)
        return this.chaveService.criarChave(data.nome, data.situacao);
```

```
  }

  @Get()
  async listarChaves(): Promise<Chave[]> {
        return this.chaveService.listarChaves();
  }

}
```

o módulo de chaves deve ser chamado dentro do módulo principal da aplicação app.module.ts

porem usaremos o “MongooseModule” para garantir a integridade do nosso banco e para informar que vai ser usado o mongo e passar o nome da “tabela” do banco e seu schema e temos que passar os controllers e services

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

import { ChaveModule } from './chave/chave.module';
import { ChaveSchema } from './chave/chave.model/chave.model';
import { ChaveController } from './chave/controller/chave.controller';
import { ChaveService } from './chave/service/chave/chave.service';
import { EmprestimoModule } from './emprestimo/emprestimo.module';
import { EmprestimoSchema } from './emprestimo/emprestimo.model/emprestimo.model';
import { EmprestimoController } from './emprestimo/controller/controller.controller';
import { EmprestimoService } from './emprestimo/service/emprestimo/emprestimo.service';
import { ServidorModule } from './servidor/servidor.module';
import { ServidorSchema } from './servidor/servidor.model/servidor.model';
import { ServidorController } from './servidor/controller/controller.controller';
import { ServidorService } from './servidor/service/servidor/servidor.service';

@Module({
  imports: [
        MongooseModule.forRoot('mongodb://localhost:27017'),
        //ChaveModule,
        MongooseModule.forFeature([{name: 'Chave', schema : ChaveSchema}]),
        MongooseModule.forFeature([{ name: 'Emprestimo', schema: EmprestimoSchema
}]),
        MongooseModule.forFeature([{ name: 'Servidor', schema: ServidorSchema }]),
  ],
  controllers: [AppController,ChaveController,EmprestimoController,ServidorController],
  providers: [AppService,ChaveService,EmprestimoService,ServidorService],

})
export class AppModule {
```

A lógica dos outros módulos(servidor,empréstimo) é a mesma o que vai mudar vai ser em empréstimo que existe ou seja o schema muda, onde você relaciona a outros schema.

```
import * as mongoose from 'mongoose';
import { Chave } from 'src/chave/chave.model/chave.model';
import { Servidor } from 'src/servidor/servidor.model/servidor.model';

export const EmprestimoSchema = new mongoose.Schema({
  datahoraEmprestimo: Date,
  datahoraDevolucao: Date,
  chave: { type: mongoose.Schema.Types.ObjectId, ref: 'Chave' },
 servidorRetirou: { type: mongoose.Schema.Types.ObjectId, ref: 'Servidor' },
  servidorDevolvel: { type: mongoose.Schema.Types.ObjectId, ref: 'Servidor' },
});

export interface Emprestimo extends mongoose.Document {
  datahoraEmprestimo: Date;
  datahoraDevolucao: Date;
  chave: Chave;
  servidorRetirou: Servidor;
  servidorDevolvel: Servidor;
}
```