**Comandos:**

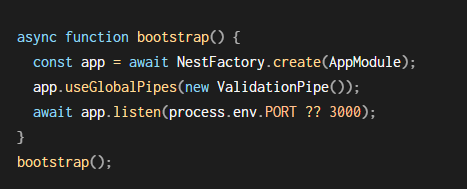
Baixar o nestjs:

npm install -g @nestjs/cli

Criar um arquivo com o nest:

npm nest new task-management-api-2

Dependências para instalar:

* npm install @nestjs/typeorm typeorm mysql2 (typeOrm)
* npm i --save class-validator class-transformer (class validator) e dps colocar no main: app.useGlobalPipes(new ValidationPipe());
* npm install @nestjs/common@latest @nestjs/core@latest (garante que o validationPipe esteja disponível)
* npm install uuid (biblioteca para gerar os uuid)
* npm install bcrypt (biblioteca para criar o hasher)
* npm install --save-dev @types/bcrypt (tipos do hasher no typeScript)
* npm run typeorm migration:generate -- -n AddPasswordToUser (Para gerar uma migration que adiciona algo novo no banco, como uma nova coluna, mas para isso, o TypeORM precisa estar configurado no package.json)
* npm install @nestjs/mapped-types (fornece PartialType pra facilitar criar UpdateDto).

Criar Módulo de User:

nest generate module user

nest generate controller user

nest generate service user

Testar a aplicação:

npm run start:dev

(Ao executar este comando, o nestjs, por padrão, cria um servidor com a porta 3000, ou seja, a rota é http://localhost:3000.

**Explicações:**

O typescript é como um Javascript, mas com maior segurança. Utiliza tipos para indicar as variáveis e se algo for dar errado, ele avisa antes de iniciar o sistema.

Main.ts -> arquivo de entrada

O Module (modulo raiz) é onde são registrados os imports, os controllers e os providers.

No module é onde é feita a conexão com o banco.

Os imports são outros módulos/bibliotecas que a aplicação precisa;

Os Controllers é que lida com as requisições

E os Providers é que lida com as regras de negócio(service)

O controller recebe o que tem que fazer, indica qual é a função do service correspondente pra requisição e chama o service. O service faz as validações e faz as manipulações no banco de dados. Depois o Controller retorna a resposta.

O controller é o garçom (pega o pedido, leva até o chefe e depois traz o pedido ao cliente)

O service é a cozinha/chefe, faz o pedido que chega até ele, podendo ser pelo navegador ou pelo postman, etc.

O nest usa o TypeOrm para conversar com o banco de dados

O arquivo Entity é uma representação da tabela no banco de dados, é a classe user que tem as propriedades do usuário(atributos)

O @Body usado dentro do Controller, serve para quando vamos enviar dados dentro do corpo da requisição. O body(corpo da requisição) é como uma caixa onde fica guardado as informações em um Post ou Put, os quais são enviados ao Service

Se for adicionar uma nova coluna tem que configurar para aceitar nulo, pois no banco, se já houver registros, pode dar problema por não aceitar nulo. Ficando assim: Column({ nullable: true })

Quando usamos “...user” estamos pegando todos os dados que o usuário enviou

Passo a passo:

Fazer a importação do typeOrm

Criar o module de user

Criar o arquivo user.entity.ts (Entity de user) com os atributos do usuário

Conectar a Entity ao module

Métodos do CRUD – No service:

Save

Find

findOneBy

update

Delete

Para que o TypeORM esteja configurado dentro do código, é necessário colocar dentro do package.json:  
"typeorm": "ts-node ./node\_modules/typeorm/cli",

"start": "nest start",

"start:dev": "nest start --watch"

O async e o await são importantes para que tudo ocorra no seu devido momento, pois quando estamos aguardando uma resposta do nest, ele não para até dar a resposta, ele continua rodando. Então quando colocamos o async, indica que a função vai receber o await, e o await faz com que espere o resultado da promise(resposta) sem dar erro.

Se colocar o async sem o await, as coisas não ocorrem na ordem correta, podendo dar erro.

Diferença entre put e patch:

O put muda tudo, mesmo que a gente altere só um campo, ele reescreve tudo. Já o Patch modificada de forma parcial, sem sobrescrever os outros campos.