Luiz Henrique de Oliveira Simionato

Documentação – UC7

**Back-end:**

Primeiramente, será criada uma pasta a qual você deseja criar o back-end, em seguida será feita a instalação das bibliotecas necessárias, usando também o type script que basicamente é uma versão tipificada do java script.

**INSTALAÇÃO BIBLIOTECAS INICIAL**

yarn init -y

Texto

Descrição gerada automaticamente

***---------------------------------------------------------------------------------------***  
***- instalando o typescript***yarn add typescript -D





Com a alteração do nome, é instalado o type Script com dev dependência “-D”

--------------------------------------------------------------------------------------  
***- instalando o express***yarn add express

-----------------------------



Sem o “-D” é uma dependência

yarn add types@express -D



***---------------------------------------------------------------------------------------***  
***- inicializando o typescript***yarn tsc –init

Uma imagem contendo Forma

Descrição gerada automaticamente

Aqui são dois sinais de menos (-)

--------------------------------------------------------------------------------------  
***- instalando o modo de script typescript para subir o servidor***yarn add ts-node-dev -D



***---------------------------------------------------------------------------------------***  
***- instalando tratativas de erros***  
yarn add express-async-errors



***---------------------------------------------------------------------------------------***  
***- liberar para qualquer ip acessar***  
yarn add cors



yarn add @types/cors -D



***---------------------------------------------------------------------------------------***  
***- instalar o Prisma para migrate de base de dados***  
yarn add prisma -D



yarn add @prisma/client



----------------------------------------------------------------------------------------  
npx prisma init

Texto

Descrição gerada automaticamente



Texto

Descrição gerada automaticamente



O Packge.json ficará assim:

Texto

Descrição gerada automaticamente

Toda a aplicação, de início ficará assim:

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

----------------------------------------------------------------------------------------

**INTEGRAÇÃO COM BANCO DE DADOS**

Criar uma tabela no phpMyAdmin(XAMPP):



No env, alterar de postgres para my sql



No arquivo schema.prisma, será feito as tabelas do banco de dados e terá como funcionalidade

*/ É UTILIZADO PARA O USUÁRIO, UTILIZANDO A PASTA CONTROLLER QUE TEM LIGAÇÃO COM O SERVICES*

*// E TUDO ISSO É LIGADO AOU ROUTES, QUE É LIGADO AO SERVER*

*// UTILIZANDO NO SERVICES O PRISMA CLIENT DENTRO DE SRC*

Texto

Descrição gerada automaticamente

O @@map é o que permite o acesso a tabela (DB)



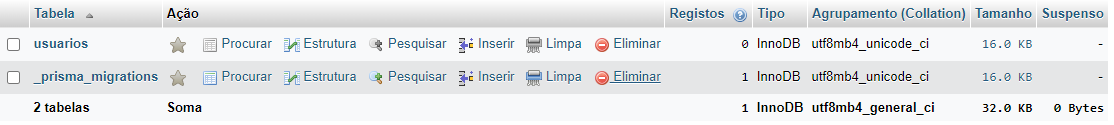
Ao inserir o comando acima na console, será criada a tabela, logo em seguida te perguntará que nome você deseja que a tabela tenha



No prisma será feita a migração e também no banco de dados

Interface gráfica do usuário, Texto

Descrição gerada automaticamente



Na tabela usuários será inserido as informações que você digitou no schema.prisma



Agora, dentro da pasta src, você irá criar uma pasta de nome prisma, criando um index em type script que fará a conexão com o banco de dados.

Interface gráfica do usuário, Texto

Descrição gerada automaticamente com confiança média

**SERVER.TS**

Logo depois você irá criar dentro de src um arquivo chamado server.ts, que irá dar funcionamento a todo o back-end:

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Para dar inicio a aplicação, devemos alterar o caminho do script do package.json e algumas outras coisas adicionais:

Texto

Descrição gerada automaticamente

Logo após você começará a codificar o server.ts com algumas importações para dar toda a funcionalidade:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Quando for importar o express, o typescript vai indicar um erro no “from ‘express’ “, e você fará a leitura do erro.

Texto

Descrição gerada automaticamente

Logo após você irá chamar as funcionalidades dentro disso:

Texto

Descrição gerada automaticamente com confiança média

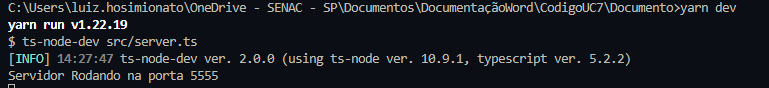
Digitado o código, você o chamará em uma porta:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Tendo aberto a porta de sua preferência, você irá para o próximo passo:





Criar o próximo arquivo.

**Routes.ts**

Dentro de src, você criará o arquivo routes.ts, o chamado rotas.

O arquivo de rotas irá te direcionar, ao local desejado.

Orientar, direcionar, para a pasta “Controller”, aonde dentro de um arquivo em uma outra pasta chamará o método reservado. (().handle)

A funcionalidade das rotas:

O front-end chega com as informações, o controller recebe e direciona ao services, o services devolve ao controller e o controller retorna ao front-end, sendo isso através do arquivo de rotas.

Texto

Descrição gerada automaticamente

Para funcionar você irá chamá-lo no arquivo principal que é o **server.ts**

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente com confiança média

Enfim chegou a parte do controller e o services, aonde o services é a função e o controller da a permissão:

Interface gráfica do usuário, Aplicativo, Teams

Descrição gerada automaticamente

Será a base para o back-end

O controller e o Services sempre se ligam então não serão separadas, sempre unidas uma dependendo da outra.

----------------------------------------------------------------------------------------

**FUNCIONALIDADE INSERT USUÁRIO**

**Controller:**

Interface gráfica do usuário, Texto

Descrição gerada automaticamente



O console.log será importante para saber se o controller está recebendo as informações, enviadas pelo insomnia ou front-end

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Controller: tendo importado o Request e o Response da biblioteca express, irá ter uma classe com o nome do controller que usará o método handle que manipulará os dados recebidos do services, aonde o Request definido pelo req fará a solicitação dos dados enviados pelo insomnia ou pelo front no body e o res sendo a resposta retornará os dados, junto do método execute.

No routes será feito a importação, e chamará o end-point como post para criar, chamando a importação junto do método handle.

Texto

Descrição gerada automaticamente

Sobre a senha, pode ser escrita como password para não confundir lá na frente.

Texto

Descrição gerada automaticamente

Sobre a importância do console.log dita anteriormente, será chamada as informações para verificação se o controller está recebendo. A mesma coisa será feita no services.



**Agora no services:**

Texto

Descrição gerada automaticamente

O prisma client será a conexão com o banco de dados.

O interface vai definir o comportamento de uma classe, sendo ela uma string ou um number

Por enquanto o método execute chamará o console.log

Interface gráfica do usuário, Texto

Descrição gerada automaticamente



Foi feito o console.log no services para verificar se chega as informações.

**FUNCIONALIDADE DO SERVICES**

Campos em branco:

Texto

Descrição gerada automaticamente

Se o email já está cadastrado:

Texto

Descrição gerada automaticamente

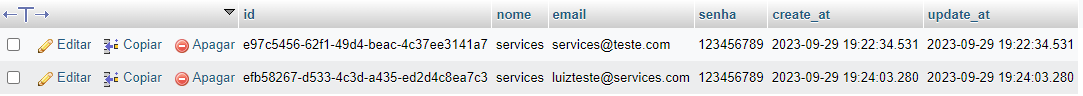
O find first traz apenas um elemento que está dentro de um objeto.

Nessa constante nomeada emailCadastado, tem a funcionalidade da conexão com o banco de dados e a tabela do schema prisma “user”, o Where é pra especificar na tabela o valor pedido. Se o email já foi cadastrado trará o erro.

Texto

Descrição gerada automaticamente

Fazendo a conexão com o banco de dados e a tabela do schema prisma, será criado um usuário, armazenado os valores em data tendo acesso pelo return que é “dados”, tendo sido armazenado o valor da constante “usuário”.



Texto

Descrição gerada automaticamente

Será baixada uma nova biblioteca, que terá como funcionalidade criptografar a senha do usuário, chamada bcrypt js que é o hash 

yarn add @types/bcryptjs -D

Texto

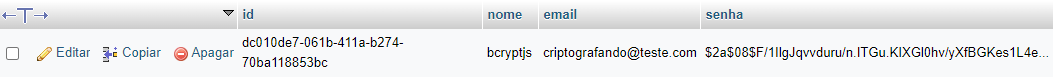
Descrição gerada automaticamente

Tela preta com letras brancas

Descrição gerada automaticamente

O await é pra sincronizar o banco de dados com o back para não haver perda de informação

Tendo feito essa funcionalidade a senha não virá como ela é, mas criptografada.



**Como o controller recebe:**

Texto

Descrição gerada automaticamente

Feito a importação do arquivo do services, criada a constante atribuindo o valor da classe do services que é o “CriarUsuariosServices”, os valores serão armazenados na constante usuários e devolvidos para o insomnia ou front-end, chamando a primeira constante atribuindo o método execute feito no arquivo do services.

-------------------------------------------------------------------------------------------------------------------

**FUNCIONALIDADE SELECT USUÁRIO**

**Controller:**



Texto

Descrição gerada automaticamente

O controller do listar usuários, vai utilizar as informações do services, chamando a funcionalidade e o que o front-end pediu primeiro, sendo armazenado na manipulação de dados que é a função handle.

Texto

Descrição gerada automaticamente

Chamando em formato de get, será feito o end-point com a chamada da classe do controller “ListarUsuariosController” e o método handle.

Texto

Descrição gerada automaticamente

Será feito a importação do prisma client que fará conexão com banco de dados, resgatando as informações da tabela de usuários, através do findMany que irá buscar todos os dados.



Texto

Descrição gerada automaticamente

Logo irá aparecer todos os cadastrados.

**FUNCIONALIDADE APAGAR USUÁRIO**

Tanto no controller como no services, é bom sempre verificar se está recebendo os dados pela console.log, utilizando o end-point com a importação no routes

Logo após ter recebido no controller e no services, você irá criar a funcionalidade no services, chamando a função delete e no “where” chamar por onde você quer deletar o usuário que no meu caso é o ID

Texto

Descrição gerada automaticamente

Basicamente, você vai chamar o await que eu já havia dito para que serve, a conexão com o banco de dados que é o Prisma Client, a tabela usuário do schema e o delete, ai você irá retornar a string sem a necessidade de armazenar em uma constante.

O código é quase a mesma coisa de todos os controller e services, o que muda é a funcionalidade.

Texto

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

**LISTAR USUÁRIO ÚNICO**

No listar usuário único, o primeiro arquivo será no controller, mas a diferença é que iremos receber por params o id, porque o params seleciona uma única informação e por essa informação temos acesso aos dados do usuário.

Tela preta com letras brancas

Descrição gerada automaticamente

A finalidade da criação desse arquivo é para quando eu listar o usuário único eu conseguir alterar por um arquivo futuro.

Texto

Descrição gerada automaticamente



No services, você definirá o comportamento do id como sendo uma string, e dentro do método execute você irá armazenar os dados em uma constante para receber eles no select, retornando a constante. Fazendo a conexão com o banco de dados(prisma Client) e com tabela do usuário(user) iremos utilizar o findUnique que permite recuperar um único registro do banco de dados.

Feito isso, chamaremos no controller a sintaxe e a importação

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**ALTERAR USUÁRIO**

No alterar usuário será feito um console.log no controller e no services normalmente para saber se está recebendo os dados normalmente

Texto

Descrição gerada automaticamente

A finalidade do services será que ao fazer conexão com o banco de dados e a tabela usuários, chamará a função update para alterar. O Where vai chamar o dado que eu quero buscar, sendo esse dado aonde irei conseguir manipular os dados.

Em data, chamando os valores da tabela usuários sendo eles “nome, email” eu vou atribuir a eles um novo valor que terá a função de alterar o registro, retornando em “dados”

Como eu disse sobre o listar usuário único, eu necessito de um front-end para conseguir ter acesso a funcionalidade dele, se não, no alterar eu irei precisar pegar o ID direto do listar usuários que está no insomnia para que assim eu consiga alterar o usuário.

Interface gráfica do usuário, Aplicativo, Teams

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Chamando a importação do services, será feito a sintaxe para trazer no controller a funcionalidade do método execute antes feito.

**AUTENTICAR USUÁRIO**

Basicamente irá fazer a mesma coisa do criar, sendo ele um post, dando console.log para ver se está recebendo tanto no controller como no services

**Sobre a FUNCIONALIDADE NO SERVICES:**

Texto

Descrição gerada automaticamente

Na constante usuário será armazenado uma **busca única** que é pelo **email**, e se o email não for compatível com o registrado virá o **erro.**

Texto

Descrição gerada automaticamente com confiança baixa

 será necessário fazer a importação do compare da biblioteca bcryptjs antes usado para criptografar a senha do usuário,

Basicamente irá comparar com a senha no email cadastrado e se a senha estiver igual irá ser autenticado, se não trará o erro.

Agora iremos utilizar o token, que irá precisar fazer a instalação da biblioteca com dependência e dev dependência.

Texto

Descrição gerada automaticamente

Feito isso, você fará a importação do sign da biblioteca e irá para o site: **MD5 HASH GENERATOR**

No arquivo env, eu tenho que colar essa hash 

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

O MD5 gerou essa Hash, e coloquei no env onde tenho acesso ao **JWT\_SEGREDO,**

Gerando o token para relacionar a tabela de filmes com categoria. Feito isso, irá ser criada a tabela categoria e filmes, para conseguir relacionar

O token, por enquanto é gerado somente na console.log

Tela de celular com aplicativo aberto

Descrição gerada automaticamente



Se eu não estiver com o token ele não autoriza

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Tudo isso vem da funcionalidade do middleware e do autenticar usuário:

**Middleware: isAutenticado.ts**

Texto

Descrição gerada automaticamente

**Autenticar usuário:**

Texto

Descrição gerada automaticamente Texto

Descrição gerada automaticamente

O middleware tem como finalidade fazer a requisição do header para conseguir autorizar a autenticação, ele vai direto para o array e encontra o token e vai buscar o que está armazenado dentro de token, se não virá o erro. A constante sub vai ser o ID do usuário, e a constante irá ter a funcionalidade de buscar o token, o acesso de segurança que é o JWT\_SEGREDO e assim definindo pela interface o ID do usuário como uma string e mostrará o ID na console.

No autenticar Usuário eu precisarei do email e senha correta para gerar o token e conseguir ter acesso aos outros arquivos, sendo eles: Listar, alterar, apagar e criar tendo acesso a todos eles, se não eu não irei conseguir fazer nada no site, vindo o erro 401 “Não autorizado”.

Para testar se o token está funcionando existe um site aonde você coloca o token e o usuário criado aparece com o ID sendo o sub

JSON Web Tokens – jwt.io

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Para que o middleware e o autenticar usuário funcione com o JWT\_SEGREDO que é a segurança do back-end, precisará alterar a linha 85 do tsconfig.json para “strict: false”



Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

-------------------------------------------------------------------------------------------------------------------

**RELACIONAMENTO COM FILMES E CATEGORIAS**

Feito isso, irá ser criada a tabela categoria e filmes, para conseguir relacionar

Texto

Descrição gerada automaticamente

dentro da tabela filmes tendo o categoriaId como string, para fazer referência com o ID da tabela categoria. na tabela categorias chamando a tabela filmes de 1 pra muitos que seria os colchetes [ ]

Criado agora o services e controller de filmes e categoria

Texto

Descrição gerada automaticamente

Com o token eu consigo inserir no categoria ID para fazer o relacionamento

Texto

Descrição gerada automaticamente

**Categoria:**

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Mas para fazer tudo isso eu precisarei estar autenticado

Texto

Descrição gerada automaticamente

Com o ID da categoria, copiando ele e colando no categoria ID eu consigo relacionar o filme com a categoria

Texto

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente com confiança média

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

-------------------------------------------------------------------------------------------------------------------