Boa noite professor, tudo bem?
Aluno: Bruno Cardoso da Silva

Nesse trabalho da A3 eu fiquei responsável por todo o backend, que foi feito com TS.

Nosso projeto consiste em um “spotify”, onde o usuário ADMIN pode utilizar o CRUD completo de artistas e álbuns e o USER pode somente ver as informaçẽos cadastradas.

Vou explicar o código seguindo por pastas, da forma como elas estão no código:

Config:

A pasta de config, como o nome sugere, é uma pasta que tem arquivos de configuração, o arquivo que tem nela é o auth.ts que vai conter uma constante com dois dados em um objeto, o chave secreta JWT e em quanto tempo ela expira.

Controllers:

Cada entidade tem um controlador específico, basicamente todos fazem a mesma coisa, que é ter um méotodo para chamar o serviço e retornar a mensagem de sucesso. O controller sempre lança uma mensagem de sucesso pois os erros acontecem sempre no serviço. O controlador de album e artista são bem semelhantes, eles tem métodos de getAll, getById, create, update e delete, bem tranquilo e acho que não precisa explicar linha por linha do código, mas, basicamente, o controlador vai chamar o serviço que vai chamar o repositório e verificar se tudo está correto, caso sim, ele passa tudo e volta por controlador, enviando a mensagem de sucesso pro usuário.

O arquivo de auth.controller.ts é o mais simples, por chamar apenas os métodos e enviar o que receber no corpo da requisição. São antes dos controladores que o middlewares vão trabalhar.

DataBase:

Possui o arquivo de conexão do prisma, ORM do banco de dados, com o bdd de fato para realizar as operações necessárias dentro dos repositórios.

Middlewares:

A parte mais dificil e legal do codigo vai ta aqui. Fiz alguns middlewares que são bastante uteis:

O ensureAuthenticated.ts vai garantir que o usuário estará autneitcado ao tentar acessar uma rota privada, primeiro verifica se tem auth.header na req, que se não tiver quer dizer que nao tem token. Depois pega o token, que vem no formato Bearer Token, então ali fiz um split pra ficar somente com o token, depois é conferido se tem o arquivo de configuração JWT, que está na pasta CONFIG, esse arquivo vai ter os dados do token JWT que vai bater com o token do usuário que estamos autenticando. Depois usamos um método da lib do JWT que verifica se o token é valido, e esses as Unknown e as TokenPayLoad são necessários para que o TS não fique reclamando dizendo que falta tipagem, de fé eu não sei muito bem o por que precisa disso, mas aprendi que precisa então sempre uso. Depois adiciona o usuário na

requisição e bola pra frente. Essa adição do usuário na requisição precisa de outro arquivo, que ta dentro da pasta TYPES, que tipa a requisição do express adicionando um usuário, aí o TS nao reclama dizendo que não existe propriedade user dentro da requisição.

O errorHandling.ts vai tratar os erros, se for um erro do ZOD, biblioteca de validação, ele retorna um erro personalizado com o que foi retornado no erro de validação do prórpio zod, tem mais alguns métodos, mas todos relacionados a erros, então se tiver um código http especifico podemos retornar um erro especifico para aquele código, depende muito de aplicação pra aplicação.

O validateSchema.ts é meu middleware favorito por que ele faz a validação de todos os schemas que foram feitos e utilizados no código. Na pasta SCHEMA tem uma interface/type para cada operação com cada entidade, ou seja, ao adicionar um usuário novo precisa ter uma interface dizendo o que deve conter no corpo da requisição para que tenham todos os dados necessários. Esse validateSchema valida se os dados recebidos dentro do body condizem com o que era esperado a partir do schema, essa parte do typescript é muito legal e é minha parte favorita do código tipar tudo que eu consigo e verificar com um middleware.

O verifyUserAUthorization.ts é um middleware que verifica o papel do usuário, então colocamos sempre como parâmetro um array com todos os papéis que podem acessar aquela rota, tipo ([“ADMIN”]) ou se for uma rota que vários papéis podem acessar ([“P1”, “P2”]). Esse middleware é dinâmico com base no que colocamos dentro do array, então sempre verifica tudo antes de deixar passar.

Repositories:

Essa pasta tem um arquivo pra cada entidade, e cada arquivo tem méotods de busca no banco de dados, na real não só de busca né, tanto pra fazer adição, alteração e delete de dados no banco, essa camada que é chamada pelos serviços para que eles se comuniquem com o banco de dados.

Routes:

Assim como os repositórios, é uma das pastas com os arquivos mais simples, somente vamos ter um arquivo por entidade, com a URL pra acessar, middlewares necessários para verificar e qual controlador cada rota chama. Pra centralizar o uso das rotas eu criei um index.ts que tem todas as rotas e exporta tudo como routes, pra facilitar o uso dentro do app.ts

Schemas:

COmo comentei antes, os schemas tem interfaces/types das entidades para garantir que cada entidade receba exatamente o que é necessário para atualizar, criar, deletar e ver cada dado do bdd. Muito importante usar isso com TS, já que a a principal funcionaldiade do TS, que é tipar.

Services:

A camada que tem as regras de negócio de cada entidade, então se uma entidade precisa ter um dado, o service que vai verificar se o dado chegou certo, se já existe algum dado igual no banco pra não ter duplicidade o service vê também. No service é aonde tem mais incidencia de uso do AppError, que é uma classe criada para jogar erros pro usuário, então eu instancio ela toda vez que faço uma validação e vejo que se for algo negativo, jogo um erro. Essa classe foi criada na pasta Utils e é um arquivo bem simples mas muito útil, eu sempre passo AppErro(“mensagem de rro”, 500) depois da virgula posso passar sempre um código HTTP que vai ser o código do erro, assim como a mensagem. Muito útil e facilita jogar erros novos.

Os demais arquivos são arquivos de configuração que dependem de cada projeto, então acredito que não cabe explicar dentro aqui.

Eu sei que posso não ter explicado muito detalhado o que eu fiz em cada arquivo, mas ia estender muito o arquivo de relatório. Realmente entendi o que eu fiz e gosto muito de trabalhar com esse tipo de desenvolvimento mais organizado, Não sou muito fã do front, como deu pra ver que eu mexi muito pouco nesse projeto com o front kkkkk. Mas, já o back, eu sou viciado, gosto muito de tipar tudo, middlewares e tudo mais que vai fazer com que minha API seja segura para que o usuário tenha seus dados safe.