Passo 1. criar a pasta estoque em Documentos

Passo 2. criar a pasta backend em estoque

Passo 3. na pasta backend, digite os comandos abaixo:

Passo 4. npm init -y

Passo 5. npm install typescript -D

Passo 6. npx tsc --init

Passo 7. npm install tsx -D

Passo 8. crie arquivo src/server.ts

Passo 9. no package.json, alterar test para dev: tsx watch src/server.ts

Passo 10. crie arquivo src/routes.ts

Passo 11. npm install fastify

Passo 12. altere o arquivo routes.ts para

import { FastifyInstance } from 'fastify'

export async function AppRoutes(app: FastifyInstance) {

}

Passo 13. npm install @fastify/cors

Passo 14. altere o arquivo server.ts para

import Fastify from 'fastify'

import cors from '@fastify/cors'

import { AppRoutes } from './routes'

// instanciar o objeto da classe Fastify

const app = Fastify()

// registra o uso do cors

app.register(cors) // permite que qualquer IP utilize as rotas

// registra todas as rotas no servidor HTTP

app.register(AppRoutes)

// vamos subir o servidor, vamos executá-lo, ele ficará ouvindo

// e aguardando as requisições

app.listen({

port: 3333

})

.then(() => {

console.log('HTTP server running and listening requests')

})

Passo 15. npm install prisma -D

Passo 16. npm install @prisma/client

Passo 17. npx prisma init --datasource-provider SQLite

Passo 18. alterar o arquivo prisma/schema.prisma para

generator client {

provider = "prisma-client-js"

}

datasource db {

provider = "sqlite"

url = env("DATABASE\_URL")

}

model User {

id Int @id @default(autoincrement())

username String

password String

email String

created\_at DateTime

products Product[]

controls Control[]

@@map("user")

}

model Product {

id Int @id @default(autoincrement())

name String

description String

quantity Float

price Float

created\_at DateTime

userId Int

user User @relation(fields: [userId], references: [id])

controls Control[]

@@map("product")

}

model Control {

id Int @id @default(autoincrement())

type String

quantity Float

price Float

created\_at DateTime

userId Int

user User @relation(fields: [userId], references: [id])

productId Int

product Product @relation(fields: [productId], references: [id])

@@map("control")

}

Passo 19. npx prisma migrate dev

Passo 20. coloque o nome da migração como product, user and control

Passo 21. npm install zod

Passo 22. crie o arquivo src/lib/prisma.ts com o conteúdo

// importa o prisma

import { PrismaClient } from '@prisma/client'

// instanciar o objeto da classe PrimaClient

export const prisma = new PrismaClient()

Passo 23. npm install dayjs

Passo 24. abrir o arquivo src/routes.ts e insira:

import { FastifyInstance } from 'fastify'

import {z} from 'zod'

import {prisma} from './lib/prisma'

import dayjs from 'dayjs'

export async function AppRoutes(app: FastifyInstance) {

    // rota para criar um user

    app.post('/user', async (request) => {

        const postBody  = z.object({

                username: z.string(),

                password: z.string(),

                email: z.string()

            })

        const {username, password, email} = postBody.parse(request.body)

        const created\_at = dayjs().startOf('day').toDate() // sem hora, minuto e segundo

        const newUser = await prisma.user.create({

            data: {

                username,

                password,

                email,

                created\_at

        }

        })

        return newUser

    })

     // rota para recuperar um user

     app.post('/user/login', async (request) => {

        const postBody  = z.object({

                username: z.string(),

                password: z.string(),

            })

        const {username, password } = postBody.parse(request.body)

        const user = await prisma.user.findMany({

            where: {

                username: username,

                password: password

            }

        })

        return user

    })

    // define uma rota que consulta todos os usuários cadastrados no banco de dados

    app.get('/users', async () => {

        const users = await prisma.user.findMany()

        return users

    })

    // define uma rota que consulta todos os produtos cadastrados no banco de dados

        app.get('/products', async () => {

            const products = await prisma.product.findMany()

            return products

        })

// rota para criar um produto

    // define uma rota que cria um produto no banco de dados, usando o verbo post, com um usuário

    app.post('/product', async (request) => {

        // recupera os dados do corpo da requisição

        const createProductBody = z.object({

            name: z.string(),

            description: z.string(),

            quantity: z.number(),

            price: z.number(),

            userId: z.number()

        })

        const {name, description, quantity, price, userId} = createProductBody.parse(request.body)

        // insere o produto no banco de dados

        // recupera a data atual - de hoje

        const today = dayjs().startOf('day').toDate() // sem hora, minuto e segundo

        let newProduct = await prisma.product.create({

            data: {

                name,

                description,

                quantity,

                price,

                created\_at: today,

                userId

            }

        })

        return newProduct

    })

// recupera todos os produtos de um usuário

    app.get('/products/:userId', async (request) => {

        const userIdParams = z.object({

            userId: z.string()

        })

        const {userId} = userIdParams.parse(request.params)

        const products = await prisma.product.findMany({

            where: {

                userId: Number(userId)

            }

        })

        return products

    })

    app.patch('/product/compra', async (request) => {

        const compraBody = z.object({

            id: z.number(),

            userId: z.number(),

            quantity: z.number(),

            price: z.number()

        })

        const {id, userId, quantity, price} = compraBody.parse(request.body)

        let productUpdated = await prisma.product.update({

            where: {

                id: id

            },

            data: {

                quantity: {

                    increment: quantity

                }

            }

        })

        const today = dayjs().startOf('day').toDate() // sem hora, minuto e segundo

        await prisma.control.create({

            data: {

                type: "C",

                quantity,

                price: price,

                created\_at: today,

                userId,

                productId: id

            }

        })

        return productUpdated

    })

    //Passo 5. Lista os controls

app.get('/controls', async () => {

        const controls = await prisma.control.findMany()

        return controls

    })

   // Passo 6. Lista os controls de um usuário

app.get('/controls/:userId', async (request) => {

        const userIdParams = z.object({

            userId: z.string()

        })

        const {userId} = userIdParams.parse(request.params)

        const controls = await prisma.control.findMany({

            where: {

                userId: Number(userId)

            }

        })

        return controls

    })

   // Passo 7. Realiza uma venda de um usuário

// rota pra atualizar a quantidade em estoque - venda

    app.patch('/product/venda', async (request) => {

        const vendaBody = z.object({

            id: z.number(),

            x: z.number(),

            userId: z.number(),

            price: z.number()

        })

        const {id, x, userId, price} = vendaBody.parse(request.body)

        let resp = await prisma.product.updateMany({

            where: {

                id: id,

                quantity: {

                    gte: x

                }

            },

            data: {

                quantity: {

                    decrement: x

                }

            }

        })

    //  return resp.count

        if ((resp.count) > 0){

            const today = dayjs().startOf('day').toDate() // sem hora, minuto e segundo

            await prisma.control.create({

                data: {

                    type: "V",

                    quantity: x,

                    price: price,

                    created\_at: today,

                    userId,

                    productId: id

                }

            })

            return 1 // indica que a venda foi realizada

        }

        else {

            return 0 // indica que a venda não foi realizada

        }

    })

    // rota para remover um produto, usando o verbo delete

    app.delete('/product/:id', async (request) => {

        // recupera o id para remoção

        const idParam = z.object({

            id: z.string()

        })

        const {id} = idParam.parse(request.params)

        // remove o produto

        let productDeleted = await prisma.product.delete({

            where: {

                id: Number(id)

            }

        })

        return productDeleted

    })

}