ARQUIVOS E TIPOS

SAE COMP

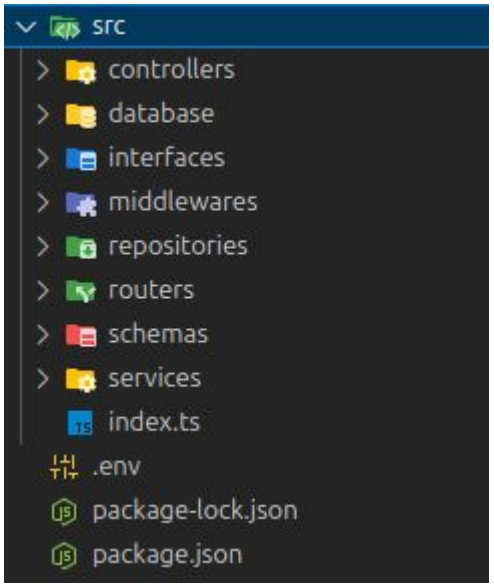
Instalando dependências

Neste projeto será usada as seguintes dependências:

```
npm init -y
npm install express
npx tsc --init
npm install typescript dotenv pg cors zod
npm install --save-dev @types/dotenv @types/react @types/react-dom @types/pg @types/cors
```

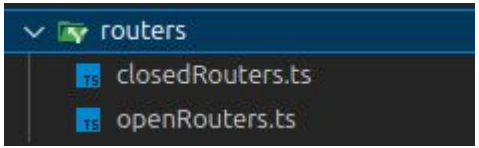
Organização das pastas

Para facilitar a organização de código e a modularização, são definidas pastas onde se coloca arquivos que realizam funções específicas



Routers

Os routers são redirecionadores de tráfego para cada rota. No exemplo, todas as rotas que se iniciam com /api serão redirecionadas para o openRouter. Para acessar uma rota dele deve-se usar /api/products e não apenas /products.




```ts
// router que pode ser acessado abertamente
app.use('/api', openRouter);

// router que precisa de autenticação
app.use('/admin', closedRouter);
```

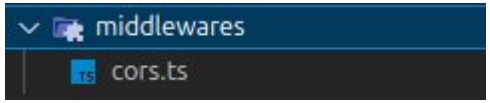
index.ts

```ts
import express from 'express';
import getProducts from '../controllers/getProducts';
import createProduct from '../controllers/createProduct';

const openRouter = express.Router();

openRouter.get('/products/:id', getProducts);

openRouter.post('/products', createProduct);


export default openRouter;
```

openRouters.ts

Middlewares

Os middlewares são chamados em todas as rotas antes de serem chamadas as funções que vão responder ao request. No exemplo, é utilizado um middleware pronto de cors, que impede o acesso da API por sites desconhecidos.



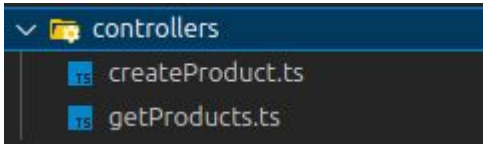

```ts
// permite apenas body do tipo JSON e coloca content-type como JSON
app.use(express.json());

// impede que outros sites utilizaem a API
app.use(corsMiddleware);
```

index.ts

```ts
import cors from 'cors';

const allowedOrigins = [
    "http://localhost:8080",
    "http://localhost:5173",
    "http://localhost:5500"
];

const corsMiddleware = cors({
    origin: (origin, callback) => {
        if (!origin || allowedOrigins.includes(origin)) {
            return callback(null, true);
        }
        return callback(new Error('Not allowed by CORS'));
    }
});

export default corsMiddleware;
```

Controllers

Os controllers são as funções que respondem a requisições de rota. Elas que devem ser colocadas em cada rota dos routers. Essas funções sempre seguem o mesmo padrão com req e res como argumentos.




```ts
openRouter.get('/products/:id', getProducts);

openRouter.post('/products', createProduct);
```

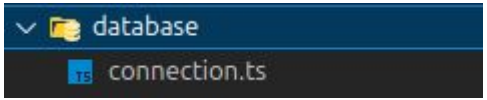
openRouters.ts

```ts
import { Request, Response } from "express";
import convertUsd from "../services/convertUsd";
import productsData from "../repositories/productsData";
import { getProductsParamSchema, getProductsQuerySchema } from "../schemas/getProducts.schema";
import { z } from 'zod'

const getProducts = async (req: Request, res: Response): Promise<void> => {
    try {
        // pega parâmetro da URL /products/:id
        const { id } = getProductsParamSchema.parse(req.params);

        // pega query da URL /products/:id?moeda=usd
        const { moeda } = getProductsQuerySchema.parse(req.query);

        // chama a função do repositories para realizar a query
        const produto = await productsData(id);

        if (!produto) {
            res.status(404).json({ message: 'Produto não encontrado' });
            return;
        }
        if (moeda == 'usd') {
            produto.preco_do_produto = convertUsd(produto.preco_do_produto);
        }
        res.json(produto);
    } catch (error) {
        console.log(error);
        if (error instanceof z.ZodError) { // erro de tipo no zod
            res.status(400).json({ error: error.errors });
            return;
        };
        res.status(500).json({ error: 'Internal server error' });
    }
}

export default getProducts;
```

getProducts.ts

Database

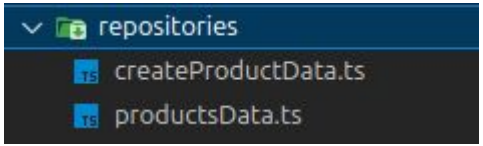
Aqui ficam as configurações para a conexão com o banco de dados.




```ts
import { Pool } from 'pg';


const poolConfig = {
  host: process.env.DB_HOST ?? '',
  port: Number(process.env.DB_PORT) ?? 5432,
  user: process.env.DB_USER ?? '',
  password: process.env.DB_PASSWORD ?? '',
  database: process.env.DB_NAME ?? '',
  ssl: false,
  max: 6
};

const pool = new Pool(poolConfig);

export default pool;
```

Repositories

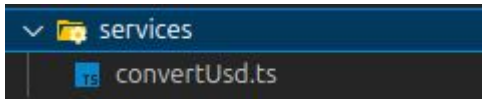
Esse é o lugar onde se faz as queries com o banco de dados. As suas funções devem apenas aceitar dados que serão utilizados na query. Deve-se evitar realizar manipulação de dados aqui.




```ts
import pool from "../database/connection";
import { IProductsData } from "../interfaces/productsData.interface";

const dbQuery = `
select
nome,
descricao,
preco::float as preco_do_produto

from produtos
where id = $1
`;


const productsData = async (id: number): Promise<IProductsData> => {
    const { rows } = await pool.query(dbQuery, [id]);
    const produto = rows[0];
    return produto;
}

export default productsData;
```

Services

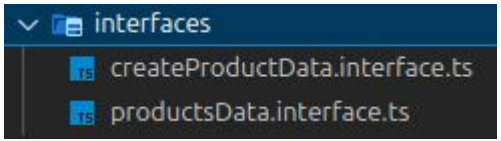
Os services são funções de manipulação de dados. Elas são utilizadas dentro dos controllers para alterar dados necessários que vieram do db.




```ts
const convertUsd = (currentValue: number): number => {
    const newValue = currentValue * 6;
    return newValue;
}

export default convertUsd;
```

Interfaces

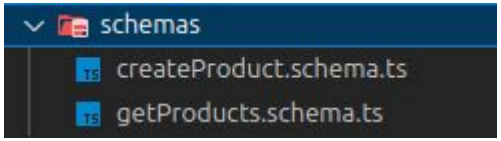
A pasta interfaces serve para definir os tipos (interfaces) que são utilizados em comunicações internas entre funções.




```ts
export interface IProductsData {
    nome: string;
    descricao: string;
    preco_do_produto: number;
};
```

Schemas

Os schemas são verificações de tipos para dados externos como body, query e params de um request. Vamos utilizar a biblioteca zod para isso. Deve-se criar um objeto zod que define todos os parâmetros que devem ser verificados.




```ts
import { z } from 'zod';

export const getProductsParamSchema = z.object({
    id: z.coerce.number(),
});

export type IGetProductsParamSchema = z.infer<typeof getProductsParamSchema>;

export const getProductsQuerySchema = z.object({
    moeda: z.coerce.string().optional(),
});

export type IGetProductsQuerySchema = z.infer<typeof getProductsQuerySchema>;
```

Verificação de tipos

Em cada controller que recebe dados externos, deve ser feita a verificação dos tipos utilizando o schema do zod. Caso o tipo seja incorreto, ele vai abrir uma exception que diz o campo errado.

```
// pega parâmetro da URL /products/:id
const { id } = getProductsParamSchema.parse(req.params);

// pega query da URL /products/:id?moeda=usd
const { moeda } = getProductsQuerySchema.parse(req.query);
```

```
catch (error) {
        console.log(error);
        if (error instanceof z.ZodError) { // erro de tipo no zod
            res.status(400).json({ error: error.errors });
            return;
        };
        res.status(500).json({ error: 'Internal server error' });
    }
```

ATIVIDADES

Atividade 1

Fazer um sistema de cadastro de tarefas

Implementar uma API no express que salve todos os seus dados no postgres utilizando verificação de tipos e organização de arquivos com as rotas:

- GET /tarefas -> Listar todas as tarefas
- POST /tarefas -> Adicionar uma nova tarefa
- PATCH /tarefas/:id -> Marcar tarefa como concluída
- GET /tarefas/status/:status -> Listar tarefas por status

MUITO OBRIGADO


@saecomp.ec


saecomp@usp.br


saecomp.github.io


Prédio da Engenharia de Computação,
Campus 2, USP São Carlos

SAECOMP