ROTEAMENTO NO EXPRESS.JS

INTRODUÇÃO

O **roteamento** no Express.js é o processo de definir como as requisições HTTP devem ser tratadas pelo servidor. Isso permite criar diferentes endpoints para interagir com uma aplicação web ou API.
Neste material, veremos como organizar as rotas, configurar o Express para receber JSON e criar uma rota de teste.

Requisição	Path	Objetivo
GET	/api/products	Consultar todos produtos
GET	/api/products/4	Consultar produto específico
POST	/api/products	Criar um produto
PUT	/api/products/7	Atualizar um produto específico
DELETE	/api/products/2012	Deletar um produto específico

DEFININDO AS ROTAS

O Express facilita a criação de rotas HTTP para diferentes tipos de requisição (GET, POST, PUT, DELETE, etc).

Rota para a Página Inicial (/)

```
app.get("/", (req, res) => {
  res.send("olá mundo");
});
```

CRIANDO UM SERVIDOR SIMPLES

Podemos criar um servidor Express básico em um único arquivo **server.js**:

```
const express = require("express");
const fs = require("fs");
const path = require("path");

const app = express();
const PORT = 5000;
const filePath = path.join(__dirname, "..", "mocks", "alunos.json");

app.get("/", (req, res) => {
  res.send("Olá mundo");
});

app.get("/alunos", (req, res) => {
  fs.readFile(filePath, "utf-8", (error, data) => {
    if (error) {
      return res.status(400).json({ error: "Erro ao ler arquivo" });
    }
    res.type("text").send(data);
  });
});

app.listen(PORT, () => console.log(`Servidor rodando na porta ${PORT}`));
```

PADRONIZAÇÃO DAS ROTAS

Reduza a necessidade de documentações extensas, utilize paths legíveis ao olho humano, existem diversos tipos de padrões para as rotas, os mais utilizados estão exemplificados abaixo:

Path	Tipo
Plural	/api/products/
Singular	/api/product/
Camel Case	/api/institutionalProducts
Snack Case	/api/institutional_products
Spinal Case	/api/institutional-products

ORGANIZANDO AS PASTAS

```
meu-projeto/
├── server.js
├── src/
│   ├── routes/
│   │   ├── userRoutes.js
│   ├── controllers/
│   │   ├── userController.js
│   ├── services/
│   │   ├── userService.js
│   ├── repositories/
│   │   ├── userRepository.js
│   ├── models/
│   │   ├── userModel.js
├── package.json
```

Para manter um código organizado, seguimos a estrutura da **Layered Architecture**, que separa responsabilidades em diferentes camadas:

- routes/: Define as rotas da aplicação.
- controllers/: Contém a lógica de controle das rotas.
- services/: Implementa as regras de negócio.
- repositories/: Responsável por acessar e manipular os dados no banco.
- models/: Define a estrutura dos dados.

CONTROLLER

Local: (controllers/userController.js)
O Controller vai chamar o Service para tratar as requisições.

```
const userService = require('../services/userService');

exports.getAllUsers = async (req, res) => {
  try {
    const users = await userService.getAllUsers();
    res.json(users);
  } catch (error) {
    res.status(500).send('Error retrieving users');
  }
};
```

SERVICE

Local: (services/userService.js)
O Service vai processar a lógica de negócios e interagir com o Repository para buscar ou manipular os dados.

```
const userRepository = require('../repositories/userRepository');

exports.getAllUsers = async () => {
  return await userRepository.findAllUsers();
};
```

REPOSITORY

Local: (repositories/userRepository.js)
O Repository será responsável pela interação com o banco de dados. Ele vai fornecer funções para buscar ou alterar os dados.

```
// Simulando um banco de dados em memória
const users = [
  { id: 1, name: 'John Doe' },
  { id: 2, name: 'Jane Doe' }
];

exports.findAllUsers = async () => {
  // Aqui você pode fazer a consulta real ao banco de dados
  return users;
};
```

ROUTES

Local: (routes/userRoutes.js)
As Rotas vão apenas redirecionar as requisições para os controllers apropriados.

```
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');

// Rota para pegar todos os usuários
router.get('/', userController.getAllUsers);

// Rota para pegar um usuário por ID
router.get('/:id', userController.getUserById);

module.exports = router;
```

SERVER

Local: Raiz do projeto (junto do package.json)
O arquivo server.js vai configurar o Express e importar as rotas.

```
const express = require('express');
const app = express();
const userRoutes = require('./routes/userRoutes');

// Middleware para receber JSON
app.use(express.json());

// Usando as rotas
app.use('/api/users', userRoutes);

// Iniciando o servidor
app.listen(3000, () => {
  console.log('Servidor rodando na porta 3000');
});
```

RESUMO DA ARQUITETURA EM CAMADAS

1. **Controller**: Lida com a requisição e a resposta, delegando a lógica de negócios para a camada de Service.
2. **Service**: Contém a lógica de negócios e delega a interação com o banco de dados para o Repository.
3. **Repository**: Lida com a persistência dos dados, fazendo operações no banco de dados ou em outras fontes de dados.

Essa organização permite uma boa separação de responsabilidades, tornando o código mais modular, escalável e fácil de manter.

SABER MAIS

TRY...CATCH

TRATAMENTO DE ERROS EM ASYNC/AWAIT

No Express, quando usamos funções **async**, precisamos lidar com erros. Se uma promise falhar dentro de um **async function**, ela lança um erro, que precisa ser tratado para evitar que o servidor quebre.

Exemplo sem Try...Catch (Erro Não Tratado)

```
app.get("/usuarios", async (req, res) => {
  // Se falhar, o erro não será tratado
  const usuarios = await buscarUsuarios();
  res.json(usuarios);
});
```

TRATAMENTO DE ERROS EM ASYNC/AWAIT

Exemplo com Try...Catch (Erro Tratado)

```
app.get("/usuarios", async (req, res) => {
  try {
    const usuarios = await buscarUsuarios();
    res.json(usuarios);
  } catch (error) {
    console.error("Erro ao buscar usuários:", error);
    res.status(500).json({ mensagem: "Erro no servidor" });
  }
});
```

ROTEAMENTO

O Roteamento refere-se a como os endpoints de uma aplicação (URIs) respondem às requisições do cliente. Para uma introdução ao roteamento, consulte Roteamento básico.

SABER MAIS

THANK YOU

@wallace027dev