

GET EM NODE.JS SEM VALIDAÇÃO DE TOKEN USANDO DESTINATION

Fazendo get em node.js com módulos

Estrutura de Diretórios

```
plaintext Copiar código

project/
|
├─ config/
|   └─ sapConfig.js      # Configuração do destino SAP no SAP BTP
|
├─ controllers/
|   └─ productCharcController.js # Regras de negócio para o endpoint
|
├─ routes/
|   └─ productCharcRoutes.js # Rotas do endpoint
|
├─ server.js            # Arquivo principal para inicializar o servidor
|
├─ package.json         # Dependências do projeto
└─ node_modules/       # Módulos instalados
```

Começamos pela estrutura de diretório

1. Arquivo `config/sapConfig.js`

Configuração para acessar o SAP usando o módulo `sapCfAxios`:

```
javascript Copiar código

const sapCfAxios = require('sap-cf-axios').default;

// Configurando o destination do SAP BTP
const axios = sapCfAxios('SAP_API'); // Substitua 'SAP_API' pelo nome do seu destination

module.exports = axios;
```

3. Arquivo routes/productCharcRoutes.js

Define as rotas que utilizam o controlador:

```
javascript Copiar código  
  
const express = require('express');  
const { getProductCharcValue } = require('../controllers/productCharcController');  
  
const router = express.Router();  
  
// Rota para buscar o valor do produto  
router.get('/product-charc-value', getProductCharcValue);  
  
module.exports = router;
```

4. Arquivo server.js

Configura o servidor principal e registra as rotas:

```
javascript Copiar código  
  
const express = require('express');  
const cors = require('cors');  
const productCharcRoutes = require('./routes/productCharcRoutes');  
  
const app = express();  
  
// Middleware  
app.use(cors()); // Habilita CORS  
app.use(express.json()); // Permite JSON no body  
  
// Rotas  
app.use('/api', productCharcRoutes); // Registra as rotas no caminho '/api'  
  
// Inicializa o servidor  
const PORT = process.env.PORT || 3000;  
app.listen(PORT, () => {  
  console.log(`Servidor rodando na porta ${PORT}`);  
});
```


Explicação

1. Configuração do SAP (`sapConfig.js`):
 - Concentra a configuração do cliente Axios com o destination do SAP BTP.
2. Controlador (`productCharcController.js`):
 - Define a lógica de negócios e a interação com a API SAP.
 - Recebe os dados do request e retorna uma resposta ao cliente.
3. Rotas (`productCharcRoutes.js`):
 - Registra os endpoints e associa ao controlador correspondente.
4. Servidor (`server.js`):
 - Inicializa o servidor e registra as rotas.
 - Middleware para gerenciar CORS e JSON no body.

Como Testar


1. Inicie o servidor:

```
bash  
  
node server.js
```

 Copiar código

2. Acesse o endpoint:

```
bash  
  
GET http://localhost:3000/api/product-charc-value
```

 Copiar código

Não esqueça de instalar as dependências como o

- Express
- Cors
- Axios
- sap-cf-axios
- swagger-ui-express swagger-jsdoc */*Opcional*/*
- npm install bootstrap

2. Arquivo controllers/productCharcController.js

Controlador que implementa a lógica de negócios:

```
javascript Copiar código

const sapAxios = require('../config/sapConfig');

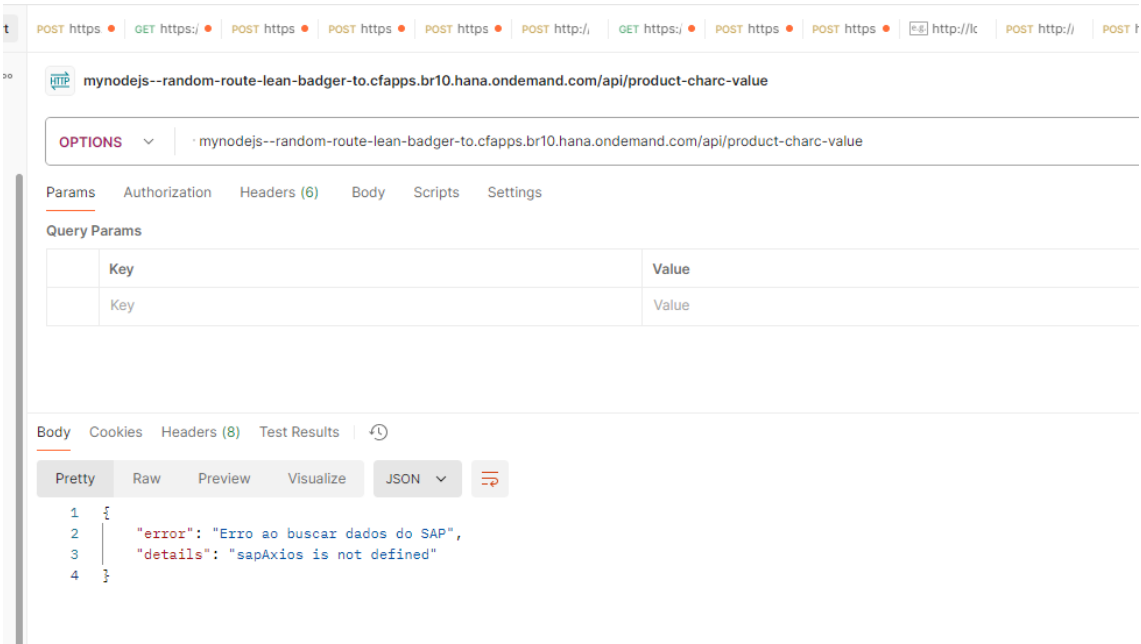
// Função para buscar os dados no SAP
async function getProductCharcValue(req, res) {
  const path = `/sap/opu/odata/sap/API_CLFN_PRODUCT_SRV/A_ProductCharcValue(Product='00000000000000000000')`;

  try {
    const response = await sapAxios.get(path, {
      headers: {
        'sap-client': '110', // Cliente SAP
      },
    });
  } catch (error) {
    console.error('Erro ao buscar dados do SAP:', error.message);
    res.status(500).json({
      error: 'Erro ao buscar dados do SAP',
      details: error.message,
    });
  }
}

module.exports = {
  getProductCharcValue,
};
```

Error

GET EM NODE.JS SEM VALIDAÇÃO DE TOKEN USANDO DESTINATION



Passando o usuário busca os dados organizacionais