Estrutura do Projeto

```
Mynodejs/
├── server.js                    # Arquivo principal do servidor
├── config/
│   └── sapAxios.js          # Configuração do axios para o SAP
├── controllers/
│   └── sapController.js     # Lógica de controle para chamadas SAP
└── routes/
    └── sapRoutes.js         # Definição das rotas
```

Passo a Passo

1. **Instale o EJS**: Primeiro, instale o `ejs` para renderizar templates.

   ```
   bash
   Copiar código
   npm install ejs
   ```

2. **Configure o EJS no Express**: No seu `app.js`, configure o EJS como mecanismo de template.
3. **Crie a Rota para a Página**: Crie uma rota para o `index.html` que renderize informações dinâmicas, como a saudação e a URL.
4. **Escreva o Template `index.ejs`**: Em vez de um `index.html` estático, usaremos um `index.ejs` para permitir a inclusão das variáveis dinâmicas.

Código Completo

Estrutura de Arquivos

```
plaintext
Copiar código
project/
├── app.js
├── views/
│   └── index.ejs
└── public/
    └── css/
        └── style.css
```

`app.js`

```
javascript
Copiar código
const express = require('express');
const app = express();
const PORT = process.env.PORT || 8080;

// Define a pasta de views e o mecanismo de template EJS
app.set('view engine', 'ejs');
app.set('views', './views');

// Servindo arquivos estáticos (opcional: para CSS, JS, imagens)
```

```
app.use(express.static('public'));

// Rota para renderizar a página index.ejs com informações dinâmicas
app.get('/', (req, res) => {
  const protocol = req.protocol;
  const host = req.get('host');
  const fullUrl = `${protocol}://${host}`;

  res.render('index', {
    greeting: 'Bem-vindo ao serviço middleware',
    url: fullUrl,
    port: PORT
  });
});

// Inicia o servidor
app.listen(PORT, () => {
  console.log(`Servidor rodando na URL: http://localhost:${PORT}`);
});
```

views/index.ejs

Crie o arquivo `index.ejs` dentro da pasta `views`. Este arquivo atuará como o `index.html`, mas com suporte para variáveis dinâmicas.

```
html
Copiar código
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Página Dinâmica</title>
  <link rel="stylesheet" href="/css/style.css">
</head>
<body>
  <h1><%= greeting %></h1>
  <p>URL do servidor: <%= url %></p>
  <p>Porta do servidor: <%= port %></p>
</body>
</html>
```

Explicação

- **`<%= greeting %>`** e outras variáveis no `index.ejs` são preenchidas dinamicamente com os valores passados na função `res.render`.
- No `app.js`, o `res.render('index', {...})` passa as variáveis `greeting`, `url` e `port` para o template `index.ejs`, que exibe esses valores no HTML.

Resultado

Ao iniciar o servidor (`node app.js`) e acessar a rota principal (`http://localhost:8080`), você verá a página com uma saudação dinâmica, a URL e a porta atual do servidor. Essa abordagem permite flexibilidade para alterar as variáveis e adicionar outros dados dinâmicos conforme necessário.

ara organizar o código em módulos e chamar a página principal a partir de um arquivo `server.js`, podemos estruturar o projeto em três partes principais:

1. **Arquivo `server.js`**: Arquivo principal que inicia o servidor e configura as rotas.
2. **Módulo de Rotas (`routes/mainRoute.js`)**: Onde definimos a rota que renderiza o HTML.
3. **Arquivo de Template (`views/index.ejs`)**: O template da página principal que exibe as informações dinâmicas.

Abaixo está o exemplo completo:

Estrutura de Arquivos

```
plaintext
Copiar código
project/
├── server.js             # Arquivo principal do servidor
├── routes/
│   └── mainRoute.js      # Módulo de rota para a página principal
├── views/
│   └── index.ejs         # Template EJS para a página principal
└── public/
    └── css/
        └── style.css     # Arquivo CSS (opcional)
```

1. `server.js` - Arquivo Principal do Servidor

```
javascript
Copiar código
const express = require('express');
const app = express();
const PORT = process.env.PORT || 8080;

// Configurações do Express para usar o EJS como mecanismo de
visualização
app.set('view engine', 'ejs');
app.set('views', './views');

// Serve arquivos estáticos da pasta "public" (opcional)
app.use(express.static('public'));

// Importa e usa o módulo de rotas
const mainRoute = require('./routes/mainRoute');
app.use('/', mainRoute);

// Inicia o servidor
app.listen(PORT, () => {
  console.log(`Servidor rodando na URL: http://localhost:${PORT}`);
});
```

2. `routes/mainRoute.js` - Módulo de Rotas

Crie o arquivo `mainRoute.js` dentro da pasta `routes`. Esse módulo define a rota principal para renderizar a página `index.ejs` com as informações dinâmicas.

```
javascript
Copiar código
const express = require('express');
const router = express.Router();

// Define a rota principal
router.get('/', (req, res) => {
  const protocol = req.protocol;
  const host = req.get('host');
  const fullUrl = `${protocol}://${host}`;

  // Renderiza o template EJS com variáveis dinâmicas
  res.render('index', {
    greeting: 'Bem-vindo ao serviço middleware',
    url: fullUrl,
    port: process.env.PORT || 8080
  });
});

module.exports = router;
```

3. `views/index.ejs` - Template da Página Principal

Esse arquivo será usado para renderizar o HTML com as variáveis dinâmicas.

```
html
Copiar código
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Página Dinâmica</title>
  <link rel="stylesheet" href="/css/style.css">
</head>
<body>
  <h1><%= greeting %></h1>
  <p>URL do servidor: <%= url %></p>
  <p>Porta do servidor: <%= port %></p>
</body>
</html>
```

Explicação

- **`server.js`**: Configura o servidor e importa o módulo de rotas.
- **`mainRoute.js`**: Define a rota para `"/"` e usa `res.render` para passar variáveis para o `index.ejs`.
- **`index.ejs`**: Exibe as variáveis `greeting`, `url`, e `port`.

Executando o Projeto

Após organizar os arquivos conforme a estrutura acima, execute o projeto com:

```
bash
Copiar código
node server.js
```

Agora, ao acessar `http://localhost:8080`, você verá a página `index.ejs` com as informações dinâmicas exibidas conforme configurado. Essa estrutura modular ajuda a organizar o código e a manter as rotas e a lógica de renderização separadas.

Para usar o Bootstrap em uma aplicação Node.js com Express, você pode adicionar os arquivos do Bootstrap diretamente no projeto ou carregar o Bootstrap a partir de uma CDN. Vou mostrar as duas abordagens.

Opção 1: Usando o Bootstrap via CDN

Essa é a maneira mais fácil e rápida. Basta incluir os links da CDN diretamente no seu arquivo de template HTML, como o `index.ejs`.

Passo a Passo:

1. **Adicione o link da CDN** no cabeçalho do seu arquivo de template `index.ejs` (ou `index.html`):

```
html
Copiar código
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Página com Bootstrap</title>
  <!-- Bootstrap CSS CDN -->
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/boot
strap.min.css" rel="stylesheet">
</head>
<body>
  <div class="container">
    <h1 class="text-center text-primary"><%= greeting %></h1>
    <p class="text-center">URL do servidor: <%= url %></p>
    <p class="text-center">Porta do servidor: <%= port %></p>
  </div>
  <!-- Bootstrap JS Bundle with Popper -->
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootst
rap.bundle.min.js"></script>
</body>
</html>
```

   - Isso carrega o CSS e o JS do Bootstrap diretamente dos servidores da CDN.
2. **Usando o Bootstrap**: Agora você pode usar as classes do Bootstrap nos seus elementos HTML, como `container`, `text-center`, `text-primary`, etc.

Opção 2: Adicionando Bootstrap ao Projeto Localmente

Se preferir baixar os arquivos do Bootstrap para o projeto, siga estas etapas.

Passo a Passo:

1. **Instale o Bootstrap** usando o npm:

```
bash
Copiar código
npm install bootstrap
```

2. **Sirva os arquivos do Bootstrap** no Express. Modifique seu `server.js` para servir arquivos estáticos, incluindo o Bootstrap:

```
javascript
Copiar código
const express = require('express');
const app = express();
const PORT = process.env.PORT || 8080;

// Define a pasta de views e o mecanismo de template EJS
app.set('view engine', 'ejs');
app.set('views', './views');

// Serve a pasta node_modules/bootstrap como arquivos estáticos
app.use('/bootstrap', express.static(__dirname +
'/node_modules/bootstrap/dist'));

// Inicia o servidor
app.listen(PORT, () => {
  console.log(`Servidor rodando na URL:
http://localhost:${PORT}`);
});
```

3. **Inclua o Bootstrap nos templates HTML**:
   - No arquivo `index.ejs` (ou similar), inclua os links para o CSS e o JS do Bootstrap servidos localmente:

```
html
Copiar código
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Página com Bootstrap</title>
  <!-- Bootstrap CSS local -->
  <link href="/bootstrap/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
  <div class="container">
    <h1 class="text-center text-primary"><%= greeting
%></h1>
    <p class="text-center">URL do servidor: <%= url %></p>
    <p class="text-center">Porta do servidor: <%= port
%></p>
  </div>
  <!-- Bootstrap JS local -->
```

```
  <script
src="/bootstrap/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

Agora, você pode usar o Bootstrap nos elementos da página. Com o Bootstrap instalado, o estilo e os componentes da biblioteca estarão disponíveis para estilizar a aplicação.

Para mostrar arquivos da raiz do servidor em node.js

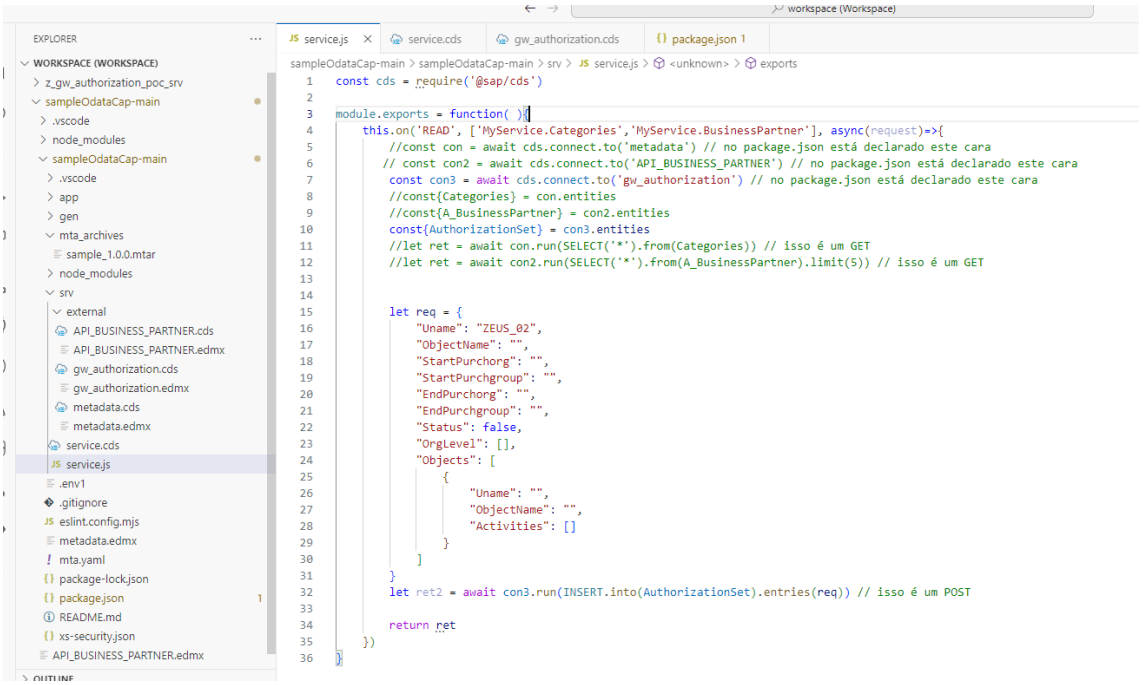
```
const express = require('express');
const path = require('path'); // Adicione esta linha
const app = express();
const PORT = 8081;

// Servir arquivos estáticos da pasta 'public'
app.use('/public', express.static(path.join(__dirname, 'public')));

// Iniciar o servidor
app.listen(PORT, () => {
  console.log(`Servidor rodando em http://localhost:${PORT}`);
});
```

Explicação:

- **`const path = require('path');`**: Importa o módulo `path` que ajuda a manipular caminhos de arquivos no sistema de forma independente do sistema operacional.
- O código **`path.join(__dirname, 'public')`** usa o módulo `path` para garantir que o caminho para a pasta `public` seja construído corretamente, independentemente do sistema operacional.

Agora, o servidor deve funcionar corretamente, e você poderá acessar os arquivos estáticos da pasta `public` usando a URL `http://localhost:8081/public/index.html` (ou qualquer outro arquivo dentro de `public`).

Modulos em Node.js



Em genexus a consulta

Event Start

```
/* Modo SDT com nivel */
&sdtorga.Status      = 'True'
&sdtorga.Erro        = 'OK'

&sdtorga1            = New()
&sdtorga1.EKGRP      = '104'
&sdtorga1.EKORG      = 'C004'
&sdtorga.NivelOrg.Add(&sdtorga1)

&sdtorga1            = New()
&sdtorga1.EKGRP      = '103'
&sdtorga1.EKORG      = 'C004'
&sdtorga.NivelOrg.Add(&sdtorga1)
```

Webeder sistemas

```
        &sdtorga1           = New()
    &sdtorga1.EKGRP      = '102'
    &sdtorga1.EKORG      = 'C004'

        &sdtorga.NivelOrg.Add(&sdtorga1)



    &longvachar = '{"NivelOrg": [{"EKORG": "C004","EKGRP": "103"},{"EKORG":
"C004","EKGRP": "104"},{"EKORG": "C004","EKGRP": "110"}],"Status": "True","Erro":"OK"}'

    &sdtorga.FromJson(&longvachar)

  for &sdtorga1In in &sdtorga.NivelOrg
        &EKGRP =        &sdtorga1In.EKGRP
        &EKORG = &sdtorga1In.EKORG

        GRId1.Load()

        msg(&sdtorga1In.EKGRP)
        msg(&sdtorga1In.EKORG)
    endfor

    msg(&sdtorga.ToJson())

EndEvent
```