

PRÁTICAS AVANÇADAS EM DESENVOLVIMENTO WEB

Davi Schneid - davi.Schneid@gmail.com

13/08/2024

Agenda

- ▶ NestJS
- ▶ Instalar NestJS
- ▶ Criar novo projeto
- ▶ Criar controllers, services e módulos.

- ▶ Dependências:
 - ▶ `npm install -g @nestjs/cli`
 - ▶ `npm install class-validator`
 - ▶ `npm install class-transformer`
 - ▶ `npm install firebase-admin`
 - ▶ `npm install --save @nestjs/swagger`
 - ▶ `npm install swagger-ui-express`

NestJS

► Benefícios NestJS

► Modularidade

- Organizar código em módulos.
- Facilita manutenção
- Facilita crescimento da aplicação

► Injeção de Dependência

- Facilita a criação e gerenciamento de dependências.

► Suporte para TypeScript

- Construído com TypeScript
- Utiliza os benefícios do TypeScript, tipagem estática, interfaces e etc.

► Filosofia de Design

- Inspirado no Angular.
- Arquitetura de design

► Compatibilidade com Ecossistema Node.js

- Compatível com qualquer pacote Node

NestJS

- ▶ Instalar o NestJS: `npm install -g @nestjs/cli`
- ▶ Abra um novo terminal no VS code e execute o comando

```
● PS C:\Users\Eric> npm install -g @nestjs/cli

added 260 packages in 42s

53 packages are looking for funding
  run `npm fund` for details
```

- ▶ Acessar o diretório das aplicações node:

```
● PS C:\Users\SenacRs\AplicacaoNode\PraticasAvancadasDesenvolvimentoWeb> cd .\PraticasAvancadasDesenvolvimentoWeb\
● PS C:\Users\SenacRs\AplicacaoNode\PraticasAvancadasDesenvolvimentoWeb\PraticasAvancadasDesenvolvimentoWeb> pwd

Path
----
C:\Users\SenacRs\AplicacaoNode\PraticasAvancadasDesenvolvimentoWeb\PraticasAvancadasDesenvolvimentoWeb

● PS C:\Users\SenacRs\AplicacaoNode\PraticasAvancadasDesenvolvimentoWeb\PraticasAvancadasDesenvolvimentoWeb> ls

Directory: C:\Users\SenacRs\AplicacaoNode\PraticasAvancadasDesenvolvimentoWeb\PraticasAvancadasDesenvolvimentoWeb

Mode                LastWriteTime         Length Name
----                -
d----             11/08/2024    21:47             api-web
d----             11/08/2024    20:53             ApiFirebase
d----             11/08/2024    20:53             APIMYSQL
d----             14/08/2024    10:15             ApiRouter
d----             11/08/2024    20:53             ApiSequelize
d----             11/08/2024    20:53             Aula09072024
d----             11/08/2024    20:53             Aula090724
d----             11/08/2024    20:53             Aula160724
d----             14/08/2024    11:19             Material das Aulas
d----             13/08/2024    19:53             Projeto de implementacao
d----             11/08/2024    20:54             react-intro
d----             11/08/2024    20:54             Script BAT
-a---             11/08/2024    20:54          31242 Modelo_Documento.docx
```

NestJS

► No diretório das aplicações Node, executar o comando:

► Criar o projeto nest: nest new ApiNestJS

► Selecionar o package npm: `? Which package manager would you ❤️ to use? npm`

```
PS C:\Users\SenacRs\AplicacaoNode\PraticasAvancadasDesenvolvimentoWeb\PraticasAvancadasDesenvolvimentoWeb> nest new ApiNestJS
⚡ We will scaffold your app in a few seconds..

? Which package manager would you ❤️ to use? npm
CREATE api-nest-js/.eslinttrc.js (688 bytes)
CREATE api-nest-js/.prettierrc (54 bytes)
CREATE api-nest-js/nest-cli.json (179 bytes)
CREATE api-nest-js/package.json (2019 bytes)
CREATE api-nest-js/README.md (3413 bytes)
CREATE api-nest-js/tsconfig.build.json (101 bytes)
CREATE api-nest-js/tsconfig.json (567 bytes)
CREATE api-nest-js/src/app.controller.ts (286 bytes)
CREATE api-nest-js/src/app.module.ts (259 bytes)
CREATE api-nest-js/src/app.service.ts (150 bytes)
CREATE api-nest-js/src/main.ts (216 bytes)
CREATE api-nest-js/src/app.controller.spec.ts (639 bytes)
CREATE api-nest-js/test/jest-e2e.json (192 bytes)
CREATE api-nest-js/test/app.e2e-spec.ts (654 bytes)

✓ Installation in progress... 🍷

🚀 Successfully created project api-nest-js
👉 Get started with the following commands:

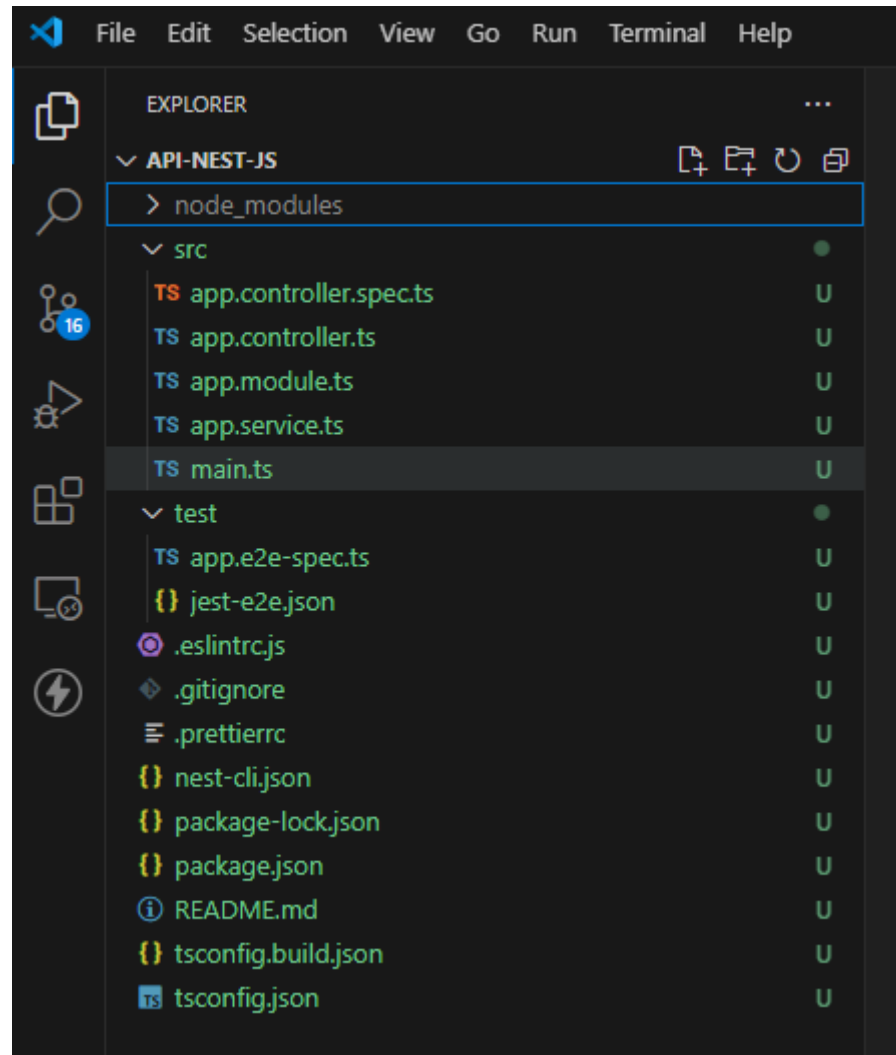
$ cd api-nest-js
$ npm run start

Thanks for installing Nest 🙏
Please consider donating to our open collective
to help us maintain this package.

👉 Donate: https://opencollective.com/nest
```

NestJS

- ▶ Abrir o projeto no Visual Studio Code.
- ▶ A aplicação já vem configurada desta forma




NestJS

- ▶ É possível criar controllers, services e modules com os comandos:
 - ▶ nest generate controller users
 - ▶ nest generate service users
 - ▶ nest generate module users

```
PS C:\Users\SenacRs\AplicacaoNode\PraticasAvancadasDesenvolvimentoWeb\PraticasAvancadasDesenvolvimentoWeb\api-nest-js> nest generate controller users
CREATE src/users/users.controller.ts (103 bytes)
CREATE src/users/users.controller.spec.ts (503 bytes)
UPDATE src/app.module.ts (336 bytes)
PS C:\Users\SenacRs\AplicacaoNode\PraticasAvancadasDesenvolvimentoWeb\PraticasAvancadasDesenvolvimentoWeb\api-nest-js> nest generate service users
CREATE src/users/users.service.ts (93 bytes)
CREATE src/users/users.service.spec.ts (471 bytes)
UPDATE src/app.module.ts (404 bytes)
PS C:\Users\SenacRs\AplicacaoNode\PraticasAvancadasDesenvolvimentoWeb\PraticasAvancadasDesenvolvimentoWeb\api-nest-js> nest generate module users
CREATE src/users/users.module.ts (86 bytes)
UPDATE src/app.module.ts (467 bytes)
PS C:\Users\SenacRs\AplicacaoNode\PraticasAvancadasDesenvolvimentoWeb\PraticasAvancadasDesenvolvimentoWeb\api-nest-js> |
```

NestJS

- Criar o typescript CreateUserDto.ts dentro da pasta users para armazenar os dados do usuário.

```
src > users > TS CreateUser.dto.ts >  CreateUserDto
1   export class CreateUserDto {
2       readonly name: string;
3       readonly age: number;
4   }
```


NestJS

- ▶ Criar as rota no arquivo users.controller.ts
- ▶ Criar rotas get e post

```
src > users > TS users.controller.ts > UsersController > findAll
1  import { Controller, Get, Post, Body, Query } from '@nestjs/common';
2  import { UsersService } from './users.service';
3  import { CreateUserDto } from './CreateUser.dto';
4  import { ApiTags, ApiQuery, ApiResponse, ApiBody } from '@nestjs/swagger';
5
6
7  @ApiTags('users') // Define uma tag para o Swagger
8  @Controller('api/users')
9  export class UsersController {
10
11    constructor(private readonly usersService: UsersService) {}
12
13    @Get()
14    @ApiResponse({ status: 200, description: 'Usuários retornados com sucesso.', type: [CreateUserDto] }) // Define a resposta de sucesso
15    @ApiResponse({ status: 404, description: 'Nenhum usuário encontrado.' }) // Define a resposta de erro
16    async findAll(): Promise<CreateUserDto[]> {
17      return this.usersService.findAllUsers();
18    }
19
20    @Post()
21    @ApiBody({ type: CreateUserDto, description: 'Dados para criar um novo usuário' }) // Define o corpo da requisição
22    @ApiResponse({ status: 201, description: 'Usuário criado com sucesso.' }) // Define a resposta de sucesso
23    @ApiResponse({ status: 400, description: 'Dados inválidos.' }) // Define a resposta de erro
24    async create(@Body() createUserDto: CreateUserDto): Promise<string> {
25      await this.usersService.createUser(createUserDto);
26      return 'User created successfully';
27    }
28
29    // Nova rota de busca por nome
30    @ApiQuery({ name: 'name', required: true, description: 'Nome do usuário para busca' }) // Define o parâmetro de consulta
31    @ApiResponse({ status: 200, description: 'Usuários encontrados com sucesso.', type: [CreateUserDto] }) // Define a resposta de sucesso
32    @ApiResponse({ status: 404, description: 'Usuário não encontrado.' }) // Define a resposta de erro
33    @Get('buscapornome')
34    async searchUsers(@Query('name') name: string): Promise<CreateUserDto[]> {
35      console.log('Name', name);
36      const users = await this.usersService.searchUsersByName(name);
37      return users;
38    }
39  }
40
```

NestJS

- ▶ Criar os serviços no arquivo users.service.ts
- ▶ Criar os serviços de busca todos os usuários, busca por nome e cadastrar usuário

```
src > users > TS users.service.ts > UsersService > searchUsersByName
1  import { Injectable } from '@nestjs/common';
2  import { CreateUserDto } from '../CreateUser.dto';
3  import { db } from '../config/firebaseConfig';
4
5  @Injectable()
6  export class UsersService {
7    private readonly collectionName = 'users'; // Nome da coleção no Firestore
8
9    async createUser(createUserDto: CreateUserDto): Promise<void> {
10      const userRef = db.collection(this.collectionName).doc();
11      await userRef.set(createUserDto);
12    }
13
14    async searchUsersByName(name: string): Promise<CreateUserDto[]> {
15      const usersRef = db.collection(this.collectionName);
16      const snapshot = await usersRef
17        .where('name', '>=', name)
18        .where('name', '<=', name + '\uf8ff')
19        .get();
20
21      if (snapshot.empty) {
22        return [];
23      }
24
25      const users: CreateUserDto[] = [];
26      snapshot.forEach(doc => {
27        users.push(doc.data() as CreateUserDto);
28      });
29
30      return users;
31    }
32
33    async findAllUsers(): Promise<CreateUserDto[]> {
34      const usersRef = db.collection(this.collectionName);
35      const snapshot = await usersRef.get();
36
37      if (snapshot.empty) {
38        return [];
39      }
40
41      const users: CreateUserDto[] = [];
42      snapshot.forEach(doc => {
43        users.push(doc.data() as CreateUserDto);
44      });
45
46      return users;
47    }
48  }
49
```

NestJS

- ▶ Configurar o firebase
- ▶ Instalar o firebase na aplicação: `npm install firebase-admin`
- ▶ Criar o arquivo de configuração do firebase na pasta config.

```
src > config > TS firebaseConfig.ts > ...  
1  
2   const path = require('path');  
3   const admin = require('firebase-admin');  
4  
5  
6   const serviceAccount = require(path.join(process.cwd(), 'src/config/serviceAccountKey.json'));  
7  
8  
9  
10  admin.initializeApp({  
11    credential: admin.credential.cert(serviceAccount),  
12    databaseURL: "https://miltalentos-d9445.firebaseio.com"  
13  });  
14  
15  export const db = admin.firestore(); // Para Firestore  
16
```

NestJS

- Na pasta config, adicionar as chaves geradas no Firebase.

```
src > config > {} serviceAccountKey.json > ...
1  {
2    "type": "service_account",
3    "project_id": "miltalentos-d94",
4    "private_key_id": "b3aa97ccc78",
5    "private_key": "-----BEGIN PRIVATE KEY-----",
6    "client_email": "firebase-adminsdk-...",
7    "client_id": "1068823975285069",
8    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
9    "token_uri": "https://oauth2.googleapis.com/token",
10   "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
11   "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/firebase-adminsdk-...",
12   "universe_domain": "googleapis.com"
13 }
14
```

As credencias geradas no firebase

A pasta config deverá ficar assim

```
src
├── config
│   ├── TS firebaseConfig.ts
│   └── {} serviceAccountKey.json
```

NestJS

- Adicionar o Swagger no main.ts

```
src > TS main.ts > ...
1  import { NestFactory } from '@nestjs/core';
2  import { AppModule } from './app.module';
3  import { DocumentBuilder, SwaggerModule } from '@nestjs/swagger';
4
5  async function bootstrap() {
6    const app = await NestFactory.create(AppModule);
7
8    const config = new DocumentBuilder()
9      .setTitle('User API')
10     .setDescription('API para gerenciar usuários')
11     .setVersion('1.0')
12     .build();
13    const document = SwaggerModule.createDocument(app, config);
14    SwaggerModule.setup('api', app, document);
15    await app.listen(3003);
16  }
17
18  bootstrap();
19
```

NestJS

- Usando Thunder, inserir registro o registro no Firebase

The screenshot shows the Thunder client interface. The top bar indicates a POST request to `http://localhost:3003/api/users` with a status of 201 Created, size of 25 Bytes, and time of 1.70 s. The 'Body' tab is selected, showing a JSON payload:

```
{  "id": 9,  "nome": "Davi",  "idade": 30}
```

. The 'Response' tab shows the response: `1 User created successfully`.

The screenshot shows the Firebase console interface. The breadcrumb navigation indicates the path: `users > ZlpB6aipyCy8T9`. The 'users' collection is selected, and the 'Adicionar documento' (Add document) button is visible. The document details show the following fields: `id: 9`, `idade: 30`, and `nome: "Davi"`.

NestJS

- Usando Thunder, buscar registro por nome no Firebase

The screenshot shows the Thunder client interface. The top bar displays the method 'GET' and the URL 'http://localhost:3003/api/users/buscapornome?name=Joao'. The 'Send' button is visible. Below the URL bar, the 'Body' tab is selected, showing the 'JSON' content type. The response is displayed on the right side, showing a status of '200 OK', a size of '48 Bytes', and a time of '1.94 s'. The response body is a JSON array containing one object with 'name' and 'email' fields.

```
GET http://localhost:3003/api/users/buscapornome?name=Joao Send
Status: 200 OK Size: 48 Bytes Time: 1.94 s

Query Headers 2 Auth Body Tests Pre Run
JSON XML Text Form Form-encode GraphQL Binary
JSON Content Format
1
2
3
4
5
6

Response Headers 6 Cookies Results Docs
1 [
2 {
3   "name": "Joao",
4   "email": "joao.silva@gmail.com"
5 }
6 ]
```

The screenshot shows the Firebase console interface. The breadcrumb navigation indicates the path 'users > 1'. The 'users' collection is selected, and a single document is shown. The document contains two fields: 'email' with the value 'joao.silva@gmail.com' and 'name' with the value 'Joao'. The document ID is 'ZIpb6aipyCy8T9iP5eoE'.

Home > users > 1 Mais no Google Cloud

| (default) | users | 1 |
|-------------------|-----------------------|--|
| + Iniciar coleção | + Adicionar documento | + Iniciar coleção |
| users > | 1 > | + Adicionar campo |
| | 2 | email: "joao.silva@gmail.com" (string) |
| | 3 | name: "Joao" |
| | 4 | |
| | 5 | |
| | 6 | |
| | 7 | |
| | ZIpb6aipyCy8T9iP5eoE | |

NestJS

- Usando Thunder, buscar todos os registros do Firebase

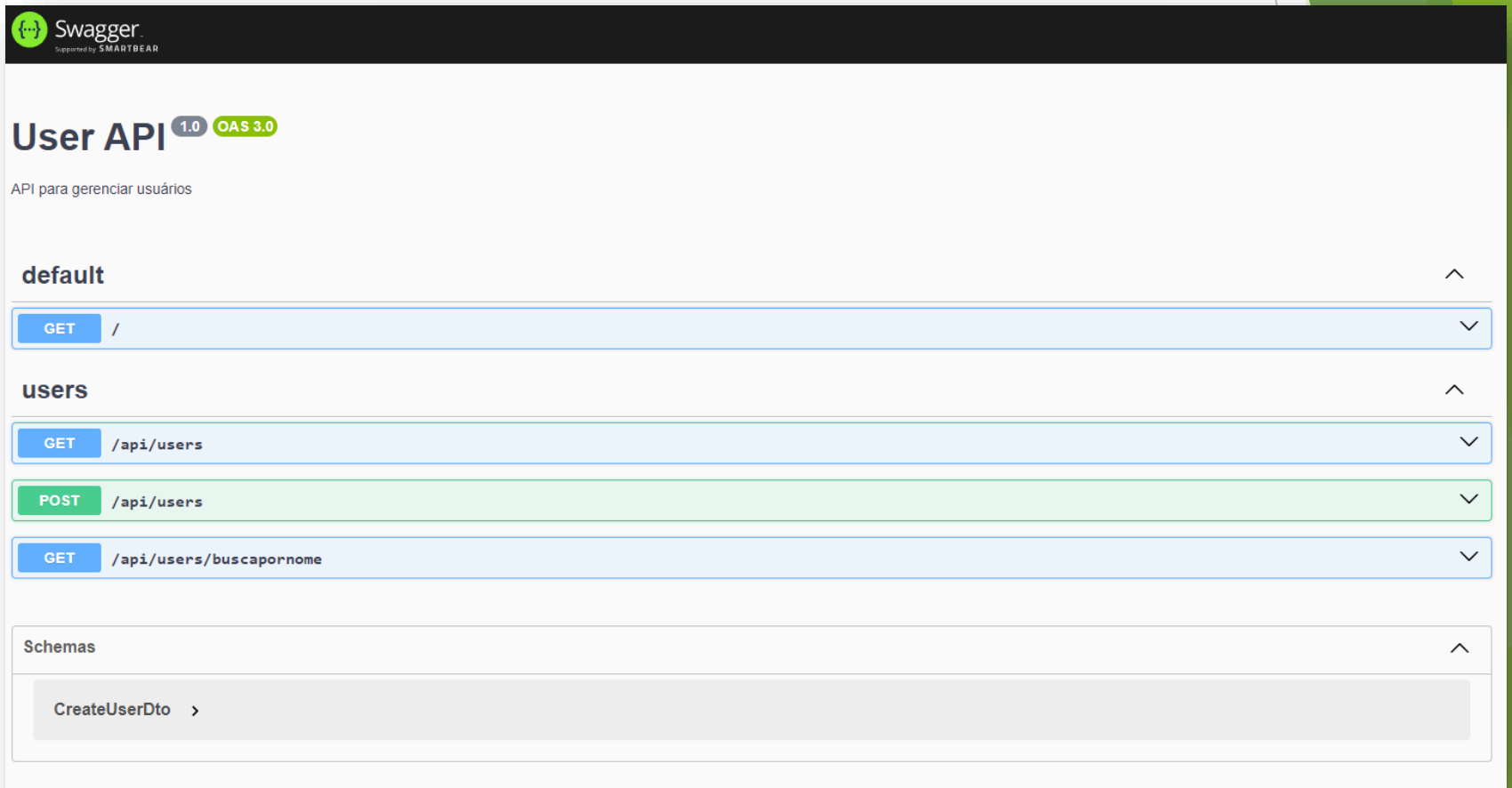
The screenshot displays the Thunder client interface. The top bar shows the method 'GET' and the URL 'http://localhost:3003/api/users/'. The 'Send' button is visible. Below the URL bar, the 'Body' tab is selected, showing the 'JSON' content type. The JSON content is displayed in a text area, and the 'Format' button is visible. The right panel shows the 'Response' tab, displaying the status '200 OK', size '460 Bytes', and time '1.27 s'. The response body is a JSON array of user records, including names, emails, and timestamps.

```
1 [
2   {
3     "name": "Joao",
4     "email": "joao.silva@gmail.com"
5   },
6   {
7     "name": "Davi2",
8     "email": "davi.schneid@gmail.com"
9   },
10  {
11    "name": "Davi2",
12    "email": "davi.schneid@gmail.com"
13  },
14  {
15    "name": "TESTE",
16    "email": "TESTE"
17  },
18  {
19    "name": "TESTE",
20    "email": "TESTE"
21  },
22  {
23    "data": {
24      "_seconds": 1723062281,
25      "_nanoseconds": 866000000
26    },
27    "name": "TESTE FIRE BASE",
28    "email": "teste@gmail.com"
29  },
30  {
31    "data": {
32      "_seconds": 1723062310,
33      "_nanoseconds": 619000000
34    },
35    "name": "FIRE BASE",
36    "email": "teste@gmail.com"
37  },
38  {
39    "idade": 30,
40    "nome": "Davi",
41    "id": 9
42  }
43 ]
```


NestJS

- ▶ Testando pelo Swagger.

 localhost:3003/api#/



The image shows the Swagger UI interface for a NestJS application. The header includes the Swagger logo and the text "Supported by SMARTBEAR". The main title is "User API" with a version "1.0" and "OAS 3.0" specification. Below the title, it says "API para gerenciar usuários". The interface is divided into sections: "default", "users", and "Schemas". The "default" section shows a GET endpoint at "/". The "users" section shows three endpoints: GET "/api/users", POST "/api/users", and GET "/api/users/buscapornome". The "Schemas" section shows a "CreateUserDto" schema.

Swagger
Supported by SMARTBEAR

User API 1.0 OAS 3.0

API para gerenciar usuários

default

GET /

users

GET /api/users

POST /api/users

GET /api/users/buscapornome

Schemas

CreateUserDto >