

Instituto Politécnico de Viseu
Escola Superior de Tecnologia e Gestão de Viseu
Departamento de Informática



Relatório do Trabalho Prático

Desenvolvimento de uma API de Gestão de Tarefas

Criar uma API RESTful para a gestão de tarefas

Trabalho elaborado por: Ana Figueiredo - 23223

Docentes das UCS do Trabalho: Carlos Cunha, Pedro Martins e Pedro Batista

Viseu, 2023

Instituto Politécnico de Viseu
Escola Superior de Tecnologia e Gestão de Viseu
Departamento de Informática

Relatório do Trabalho Prático
Curso de Licenciatura em
Tecnologias e Design de Multimédia

Desenvolvimento de uma API de Gestão de Tarefas

Criar uma API RESTful para a gestão de tarefas

Ano Letivo 2023/2024

Trabalho elaborado por: Ana Figueiredo - 23223

Docentes das UCS do Trabalho: Carlos Cunha, Pedro Martins e Pedro Batista

Viseu, 2023

Índice

1. Primeiro título – nível 1 Error! Bookmark not defined.

1.1. Título nível 2 **Error! Bookmark not defined.**

1.1.1. *Título nível 3*..... *Error! Bookmark not defined.*

2. Segundo Título – nível 1..... Error! Bookmark not defined.

Índice de tabelas

Índice de figuras

Figura 1-1. Exemplo de legenda..... **Error! Bookmark not defined.**

1. Introdução

O presente relatório documenta o desenvolvimento de uma API Restful para gestão de tarefas, utilizando a metodologia ágil SCRUM. Este projeto tem como objetivo fornecer uma solução flexível e eficiente para o acompanhamento e organização de tarefas, permitindo os utilizadores criar, atualizar, apagar e listar tarefas de maneira intuitiva e integrada.

O foco principal é a criação da API em Node.js, proporcionando uma arquitetura escalável e fácil integração, além disso, são incorporados requisitos técnicos essenciais, como conformidade com os princípios RESTful, tratamento de erros, autenticação básica para garantir segurança, e a atualização de uma base de dados armazenamento das informações.

O relatório está organizado em seções que refletem as etapas do desenvolvimento, começando pela definição do projeto, criação de diagramas de contexto e containers, elaboração das estruturas de dados JSON, design de JSON Schemas, implementação dos endpoints em Node.js, e culminando na documentação da API em formato OpenAPI. Cada seção fornece insights sobre as decisões de design, os desafios enfrentados e as soluções implementadas.

2. Diagramas

Um diagrama é apresentado como um gráfico ao qual as informações sobre um projeto são apresentadas de forma simplificada e/ou esquematizada. Pode ser simples ou complexo de acordo com a complexidade do projeto e também pode apresentar muitos ou poucos elementos.

2.1. Diagrama de Contexto

O Diagrama de Contexto é denominado como sendo uma visão global do sistema que nos encontramos a desenvolver tendo como objetivos:

- Qual o sistema de software que nos encontramos a desenvolver?
- Quem é que o está a utilizar?
- Como é a integração deste sistema no ambiente?

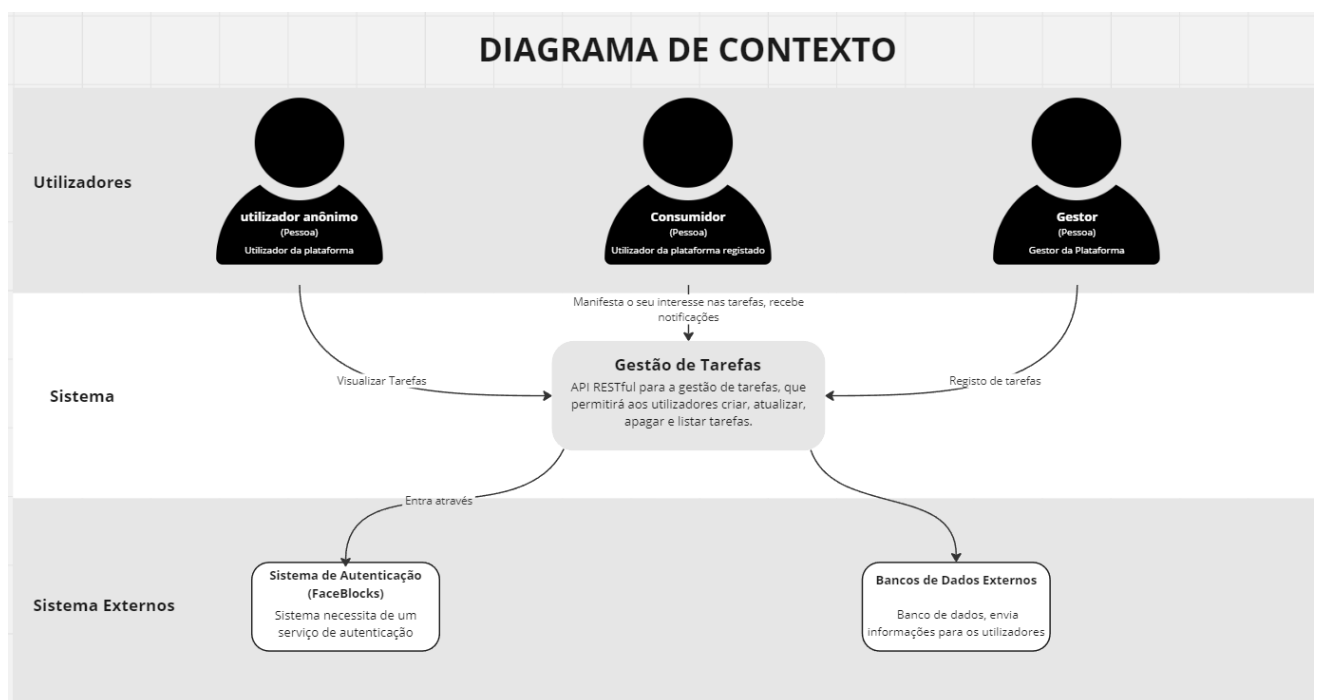


Figura 2-1. Diagrama de Contexto

Numa primeira parte, temos os possíveis utilizadores do sistema. Utilizador Anónimo poderá visualizar a informação disponibilizada, mas não pode registar interesse em produtos. O Consumido, que está registado na plataforma, poderá visualizar a informação disponibilizada e manifestar interesse nas tarefas, e receber. O Gestor, que está registado na plataforma, poderá visualizar, editar e registar a informação disponibilizada na plataforma, das tarefas.

Na segunda parte, no sistema, onde é feita a Gestão de Tarefas, a tarefa é registada, editada, atualizada, através da disponibilização da informação registada pelo Gestor de Tarefas.

Na terceira parte, os Sistemas Externos recebem as informações disponibilizadas pelo sistema e executam a autenticação para os utilizadores por OAuth, bem como mostram as tarefas disponíveis através do banco de dados.

2.2. Diagrama de Containers

O Diagrama de Containers diz respeito às tecnologias e responsabilidades distribuídas na arquitetura deste diagrama, tendo como principais objetivos:

- Quais são as decisões relacionadas com a escolha de tecnologia?
- Quais são as responsabilidades distribuídas no sistema em questão?
- Como é que os containers comunicam entre si?
- Onde é que os programadores devem desenvolver o código para implementar as funcionalidades?

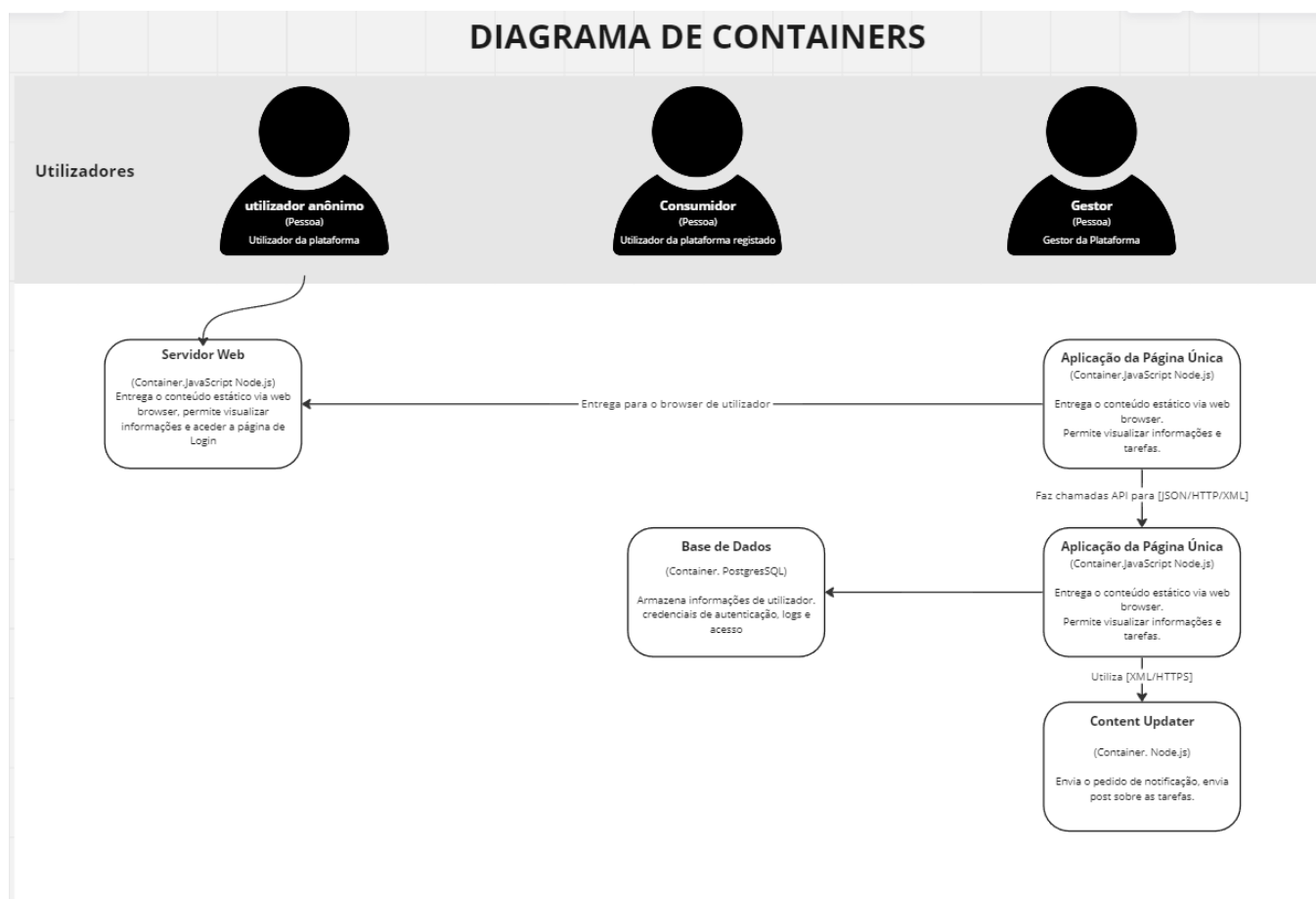


Figura 2-2. Diagrama de Containers

Numa primeira fase, o Servidor Web disponibiliza conteúdo estático, onde os utilizadores têm acesso as informações de produtos e possibilidade de fazer o registo e o login na plataforma.

Tecnologias:

JavaScript, Node.js. Comunicação através de HTTPS.

Na segunda fase, a Aplicação de Página Única faz a entrega de conteúdo dinâmico via web browser, onde os utilizadores registados têm a possibilidade de visualizar informações da conta, registar interesse em produtos e o Gestor de Produto poderá ainda fazer a gestão (inserir, editar e eliminar) dos produtos em promoção.

Através da Aplicação API é feita a gestão de contas de utilizador com recurso a uma Base de Dados Relacional, onde é armazenada as informações dos utilizadores, credências de autenticação, logs de acesso, etc. É disponibilizada ainda a informação para o Sistema de Software FaceBlocks e para o Gateway de SMS.

Tecnologias:

Aplicação de Página Única – JavaScript e React. Comunicação com a Aplicação API através de JSON/HTTP/XML;

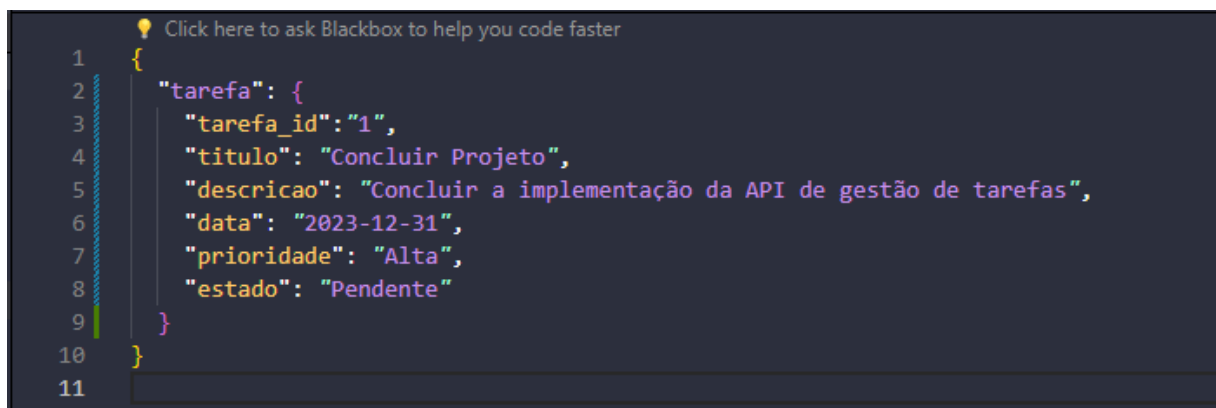
Aplicação API – Node.js e JavaScript; Comunicação com a Base de Dados Relacional PostgreSQL através de SQL/JDBC pela porta 3000, com o Gateway de SMS através de JSON/HTTP e com o FaceBlocks através de XLM/HTTPS.

3. Criação das estruturas de dados JSON

Cada estrutura de dados JSON aqui apresentada é cuidadosamente concebida para suportar operações específicas, tais como adicionar/editar tarefas, apagar tarefas, gerar notificações, autenticar utilizadores, registar novos utilizadores e listar tarefas. Cada estrutura reflete os requisitos funcionais da API, fornecendo a informação necessária para realizar operações semânticas e significativas.

Ao criar essas estruturas de dados, é essencial considerar a flexibilidade e expansibilidade do sistema. A API deve ser capaz de acomodar potenciais mudanças nos requisitos futuros sem comprometer a integridade do sistema. A escolha de campos específicos, identificadores únicos e a inclusão de dados relevantes são fundamentais para garantir a eficácia dessas estruturas.

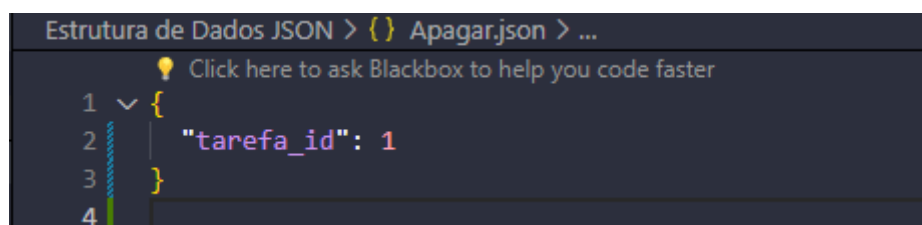
3.1. Adicionar & Editar



```
1 {  
2   "tarefa": {  
3     "tarefa_id": "1",  
4     "titulo": "Concluir Projeto",  
5     "descricao": "Concluir a implementação da API de gestão de tarefas",  
6     "data": "2023-12-31",  
7     "prioridade": "Alta",  
8     "estado": "Pendente"  
9   }  
10 }  
11
```

Figura 3-1. Código

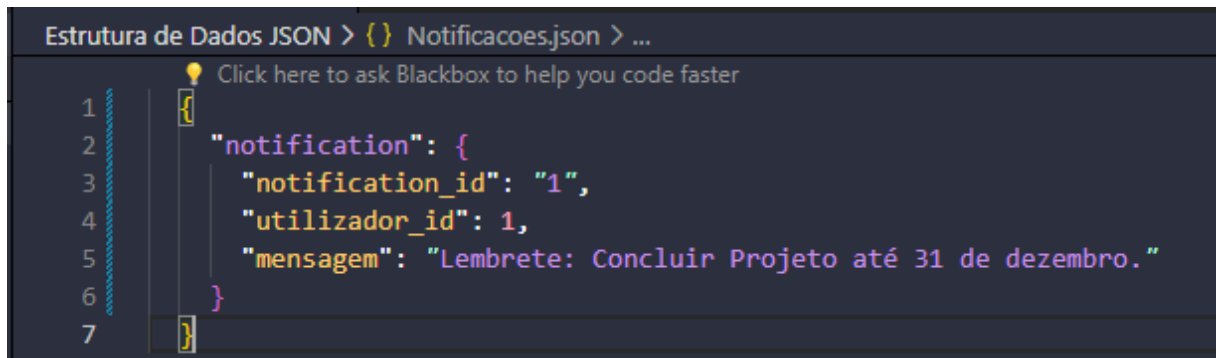
3.2. Apagar



```
Estrutura de Dados JSON > {} Apagar.json > ...  
1 {  
2   "tarefa_id": 1  
3 }  
4
```

Figura 3-2. Código

3.3. Notificações

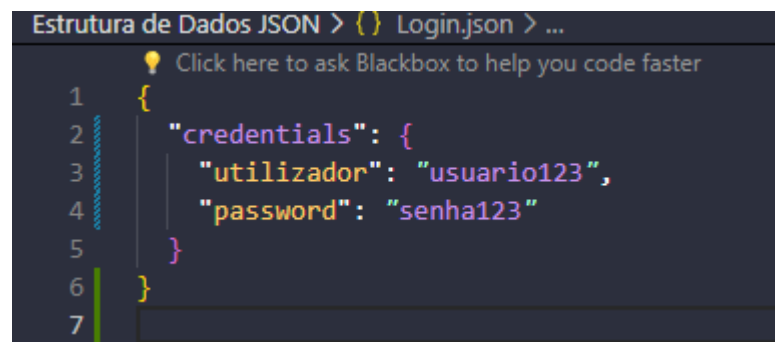


The screenshot shows a VS Code editor window with the title 'Estrutura de Dados JSON > {} Notificacoes.json > ...'. The code is as follows:

```
1 {  
2   "notification": {  
3     "notification_id": "1",  
4     "utilizador_id": 1,  
5     "mensagem": "Lembrete: Concluir Projeto até 31 de dezembro."  
6   }  
7 }
```

Figura 3-3. Código

3.4. Login

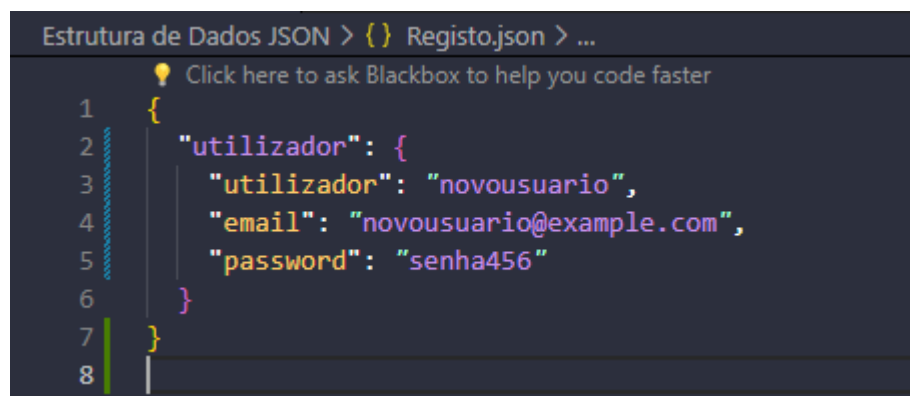


The screenshot shows a VS Code editor window with the title 'Estrutura de Dados JSON > {} Login.json > ...'. The code is as follows:

```
1 {  
2   "credentials": {  
3     "utilizador": "usuario123",  
4     "password": "senha123"  
5   }  
6 }  
7
```

Figura 3-4. Código

3.5. Registo



The screenshot shows a VS Code editor window with the title 'Estrutura de Dados JSON > {} Registo.json > ...'. The code is as follows:

```
1 {  
2   "utilizador": {  
3     "utilizador": "novousuario",  
4     "email": "novousuario@example.com",  
5     "password": "senha456"  
6   }  
7 }  
8
```

Figura 3-5. Código

3.6. Utilizadores

```
Estrutura de Dados JSON > {} Utilizadores.json > ...  
Click here to ask Blackbox to help you code faster  
1 {  
2   "utilizador_id": 123,  
3   "utilizador": "joao",  
4   "email": "utilizador@email.com",  
5   "nif": "200300400",  
6   "cargo": "Colaborador"  
7 }  
8
```

Figura 3-6. Código

3.7. Listar Tarefas

```
Estrutura de Dados JSON > {} Listar_Tarefas.json > ...  
Click here to ask Blackbox to help you code faster  
1 {  
2   "filter": {  
3     "utilizador_id": 1,  
4     "estado": "Em Progresso",  
5     "prioridade": "Alta"  
6   }  
7 }  
8
```

Figura 3-7. Código

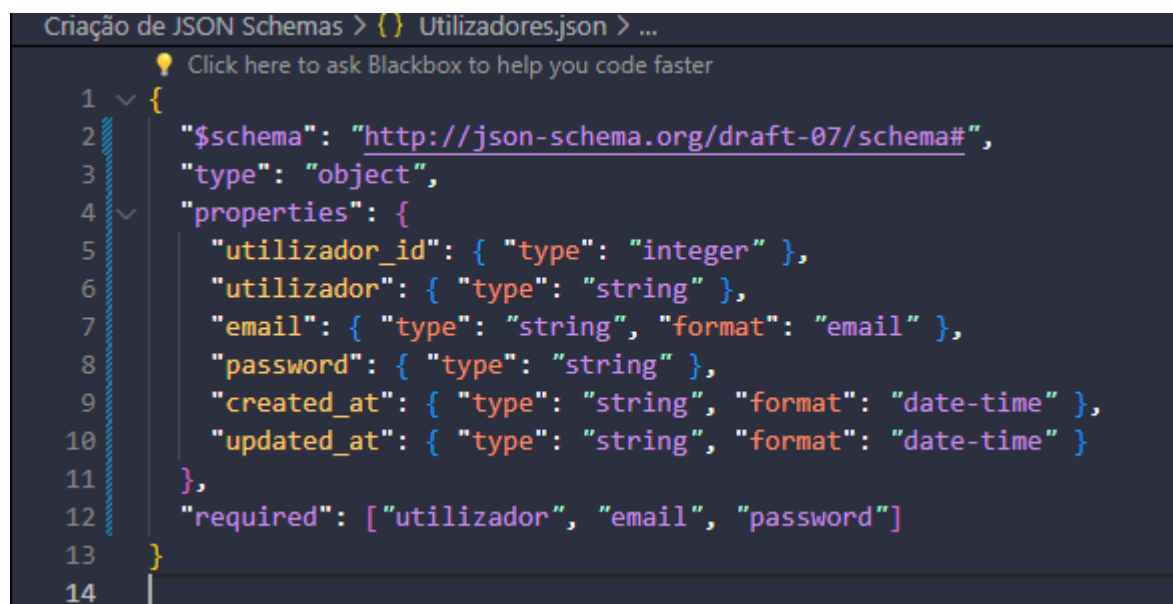
4. Criação de JSON Schemas

O objetivo primordial dessa etapa é definir normas claras e específicas para cada estrutura de dados JSON envolvida nas operações da API. Os JSON Schemas oferecem uma descrição formal da estrutura e dos tipos de dados esperados, permitindo a validação eficaz dos dados recebidos e enviados pela API.

Neste relatório, cada JSON Schema é elaborado de acordo com as operações específicas, como adicionar/editar tarefas, apagar tarefas, gerar notificações, autenticar utilizadores, registar novos utilizadores e listar tarefas. A estrutura destes esquemas reflete os requisitos funcionais da API, promovendo a coerência e integridade dos dados.

A compreensão e implementação eficaz destes JSON Schemas são cruciais para a interoperabilidade da API, assegurando que as partes envolvidas possam comunicar-se de maneira eficiente, evitando erros e inconsistências nos dados. Este relatório oferecerá uma visão detalhada de cada JSON Schema, destacando a lógica subjacente e como essas estruturas contribuem para a robustez e confiabilidade global da API de Gestão de Tarefas.

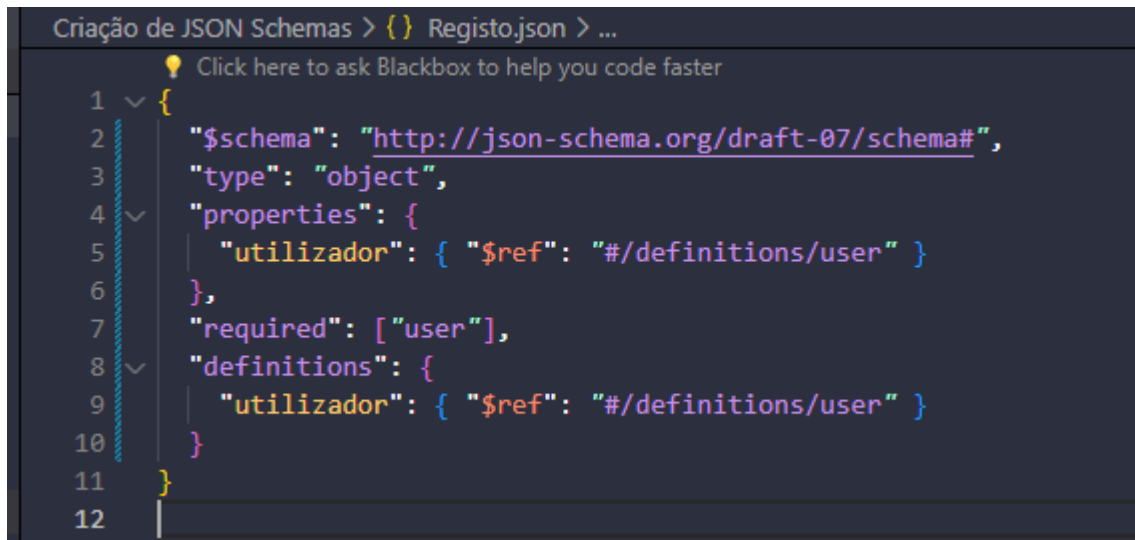
4.1. Utilizadores



```
Criação de JSON Schemas > {} Utilizadores.json > ...
Click here to ask Blackbox to help you code faster
1 {
2   "$schema": "http://json-schema.org/draft-07/schema#",
3   "type": "object",
4   "properties": {
5     "utilizador_id": { "type": "integer" },
6     "utilizador": { "type": "string" },
7     "email": { "type": "string", "format": "email" },
8     "password": { "type": "string" },
9     "created_at": { "type": "string", "format": "date-time" },
10    "updated_at": { "type": "string", "format": "date-time" }
11  },
12  "required": ["utilizador", "email", "password"]
13 }
14
```

Figura 4-1. Código

4.2. Registo



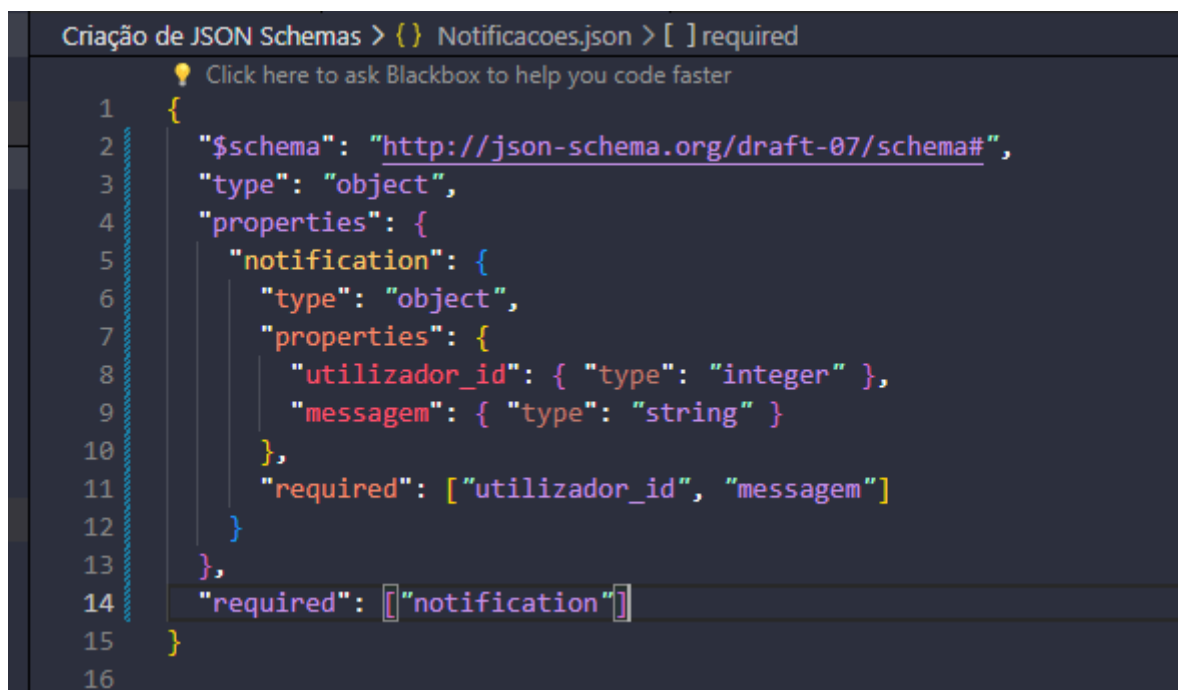
Criação de JSON Schemas > {} Registo.json > ...

Click here to ask Blackbox to help you code faster

```
1 {
2   "$schema": "http://json-schema.org/draft-07/schema#",
3   "type": "object",
4   "properties": {
5     "utilizador": { "$ref": "#/definitions/user" }
6   },
7   "required": ["user"],
8   "definitions": {
9     "utilizador": { "$ref": "#/definitions/user" }
10  }
11 }
12
```

Figura 4-2. Código

4.3. Notificações



Criação de JSON Schemas > {} Notificacoes.json > [] required

Click here to ask Blackbox to help you code faster

```
1 {
2   "$schema": "http://json-schema.org/draft-07/schema#",
3   "type": "object",
4   "properties": {
5     "notification": {
6       "type": "object",
7       "properties": {
8         "utilizador_id": { "type": "integer" },
9         "mensagem": { "type": "string" }
10      },
11       "required": ["utilizador_id", "mensagem"]
12     }
13   },
14   "required": ["notification"]
15 }
16
```

Figura 4-3. Código

4.4. Login

```
Criação de JSON Schemas > {} Login.json > [ ] required
Click here to ask Blackbox to help you code faster
1  {
2    "$schema": "http://json-schema.org/draft-07/schema#",
3    "type": "object",
4    "properties": {
5      "credentials": {
6        "type": "object",
7        "properties": {
8          "utilizador": { "type": "string" },
9          "password": { "type": "string" }
10       },
11      "required": ["username", "password"]
12    },
13  },
14  "required": ["credentials"]
15 }
16
```

Figura 4-4. Login

4.5. Apagar

```
Criação de JSON Schemas > {} Apagar.json > ...
Click here to ask Blackbox to help you code faster
1  {
2    "$schema": "http://json-schema.org/draft-07/schema#",
3    "type": "object",
4    "properties": {
5      "tarefa_id": { "type": "integer" }
6    },
7    "required": ["task_id"]
8  }
9
```

Figura 4-5. Código

4.6. Adicionar & Editar



```
Criação de JSON Schemas > {} Adicionar_Editar.json > ...
Click here to ask Blackbox to help you code faster

1  {
2    "$schema": "http://json-schema.org/draft-07/schema#",
3    "type": "object",
4    "properties": {
5      "tarefa_id": { "type": "string" },
6      "titulo": { "type": "string" },
7      "descricao": { "type": "string" },
8      "data": { "type": "string", "format": "date" },
9      "prioridade": { "type": "string", "enum": ["Baixa", "Média", "Alta"] },
10     "estado": { "type": "string", "enum": ["Pendente", "Em Progresso", "Concluída"] }
11   },
12   "required": ["tarefa_id", "titulo", "descricao", "data", "prioridade", "estado"],
13   "additionalProperties": false
14 }
```

Figura 4-6. Código

5. Criação da documentação da API em OpenAPI

Antes de começar a documentar uma API, precisamos perceber o que é uma API e como funciona. Isto torna a documentação muito mais fácil. Então, o que significa uma API?

API significa Application Program Interface (interface de programa de aplicação). Opera como uma interface entre dois programas ou dois sistemas. Também pode ser um software que cria uma ligação entre duas aplicações diferentes. Quando uma organização decide disponibilizar alguns dados publicamente, publica na forma de endpoints para que outros consigam extrair esses dados.

Application Program Interface (API)

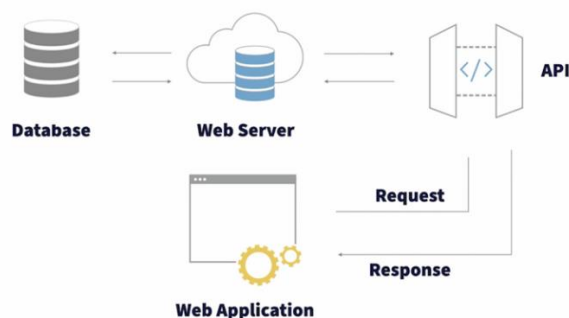


Figura 5-1. API

5.1. Integrando documentação com ferramentas de API

Com a ferramenta Swagger, a documentação pode ser gerada automaticamente a partir do próprio código. Isto ajuda a manter a documentação atualizada com as alterações mais recentes no código.

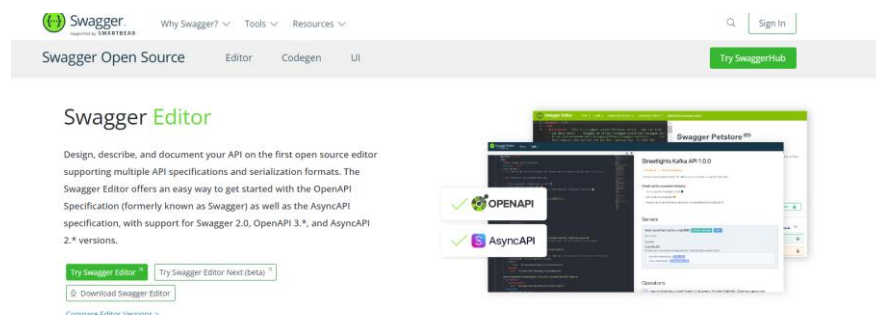


Figura 5-2. Swagger

5.2. Ficheiro em YAML

Neste ponto irão estar presentes pequenos excertos do código realizado em YAML para a resolução do Sprint 4, que consiste na criação de uma API consoante todo o trabalho realizado anteriormente tal como XML e Schemas Json.

```
1 openapi: 3.0.0
2 info:
3   title: API de Gestão de Tarefas
4   description: Documentação da API de Gestão de Tarefas
5   version: 1.0.0
6
7 paths:
8   /tarefas:
9     get:
10      summary: Listar todas as tarefas
11      responses:
12        '200':
13          description: Sucesso
14          content:
15            application/json:
16              example:
17                - id: 1
18                  title: "Concluir Projeto"
19                  description: "Finalizar o trabalho prático de API de Gestão de Tarefas"
20                  due_date: "2023-11-30"
21                  priority: "Alta"
22                  status: "Pendente"
23
24      post:
25        summary: Criar uma nova tarefa
26        requestBody:
27          required: true
28          content:
29            application/json:
30              example:
31                title: "Nova Tarefa"
32                description: "Descrição da nova tarefa"
33                due_date: "2023-12-01"
34                priority: "Média"
35                status: "Pendente"
36        responses:
37          '201':
38            description: Tarefa criada com sucesso
39            content:
40              application/json:
41                example:
42                  id: 2
43                  title: "Nova Tarefa"
44                  description: "Descrição da nova tarefa"
45                  due_date: "2023-12-01"
46                  priority: "Média"
47                  status: "Pendente"
48
49      /tarefas/{id}:
50        parameters:
51          - name: id
52            in: path
53            required: true
```

Aqui podemos observar um exemplo de uma função Post que vai criar uma tarefa à base de dados através de um formulário introduzido por um Schema.

```

49 - /tarefas/{id}:
50 -   parameters:
51 -     - name: id
52 -       in: path
53 -       required: true
54 -       description: ID da tarefa
55 -       schema:
56 -         type: integer
57 -
58 -   get:
59 -     summary: Obter detalhes de uma tarefa específica
60 -     responses:
61 -       '200':
62 -         description: Sucesso
63 -         content:
64 -           application/json:
65 -             example:
66 -               id: 1
67 -               title: "Concluir Projeto"
68 -               description: "Finalizar o trabalho prático de API de Gestão de Tarefas"
69 -               due_date: "2023-11-30"
70 -               priority: "Alta"
71 -               status: "Pendente"
72 -
73 -   put:
74 -     summary: Atualizar uma tarefa existente
75 -     requestBody:
76 -       required: true
77 -       content:
78 -         application/json:
79 -           example:
80 -             title: "Tarefa Atualizada"
81 -             description: "Nova descrição da tarefa"
82 -             due_date: "2023-12-02"
83 -             priority: "Alta"
84 -             status: "Em Progresso"
85 -     responses:
86 -       '200':
87 -         description: Tarefa atualizada com sucesso
88 -         content:
89 -           application/json:
90 -             example:
91 -               id: 1
92 -               title: "Tarefa Atualizada"
93 -               description: "Nova descrição da tarefa"
94 -               due_date: "2023-12-02"
95 -               priority: "Alta"
96 -               status: "Em Progresso"
97 -
98 -   delete:
99 -     summary: Apagar uma tarefa existente
100 -     responses:

```

Pode-se observar-se um Schema que vai realizar o formulário e especificar os dados necessários para criar um user e assim ser possível a utilização de métodos como POST, GET, etc...

6. Implementação dos endpoints em Node.JS

Criamos a nossa API com recurso ao ficheiro disponibilizado na nossa conta do Swagger, com tudo devidamente configurado, e usamos o Node.js como base do nosso servidor local.

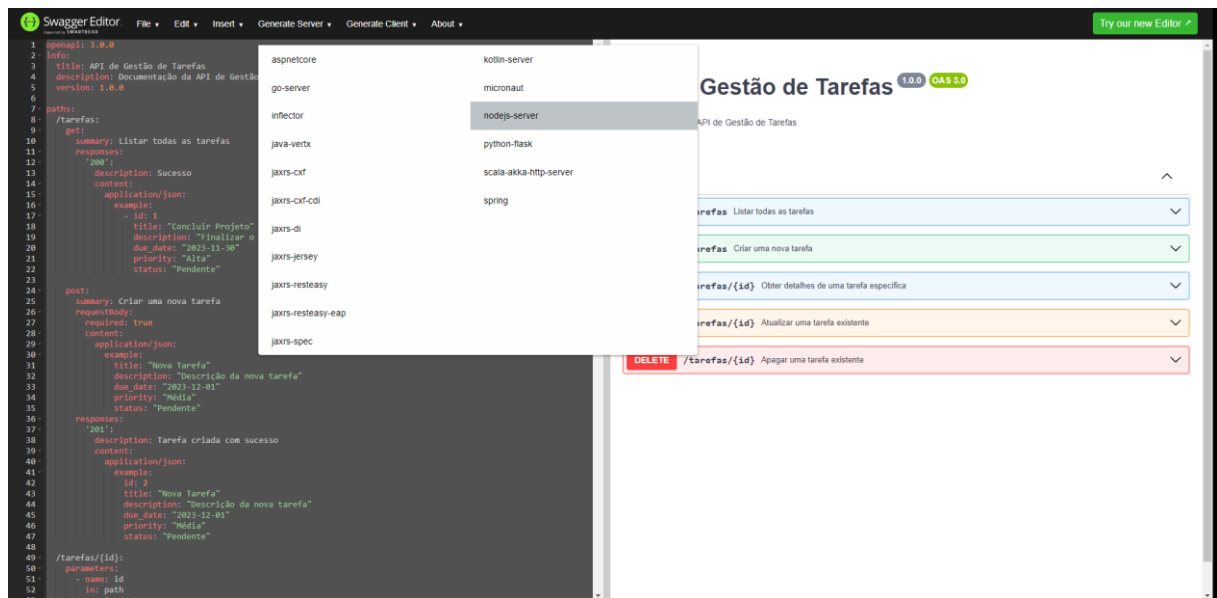


Figura 6-1. Exportação do swagger

Depois de descarregado o ficheiro do Swagger e descompactado na pasta procedemos à instalação do node.js.

6.1. Instalação

Comecei por fazer download da versão mais recente do Node.js e de acordo com o sistema operativo em que estamos a desenvolver a API em: <https://nodejs.org/en/download/>

Através do “Terminal” do sistema operativo, posicionámo-nos dentro da pasta criada anteriormente, com os ficheiros disponibilizados pelo Swagger já estão configurados para o node.js-server, basta inicializar o serviço, que automaticamente faz a instalação dos ficheiros necessários para arrancar.

```
PS C:\Users\Utilizador 1\Desktop\TrabalhoAI3_23223\Trabalho-AI3> npm run start
```

Figura 6-2. Código no Terminal

Mensagem de retorno:

```
YOUR MODEL DISABLED
Your server is listening on port 8080 (http://localhost:8080)
Swagger-ui is available on http://localhost:8080/docs
```

Figura 6-3. Mensagem

Podemos agora aceder à documentação da api através do endereço indicado: <http://localhost:8080/docs/>

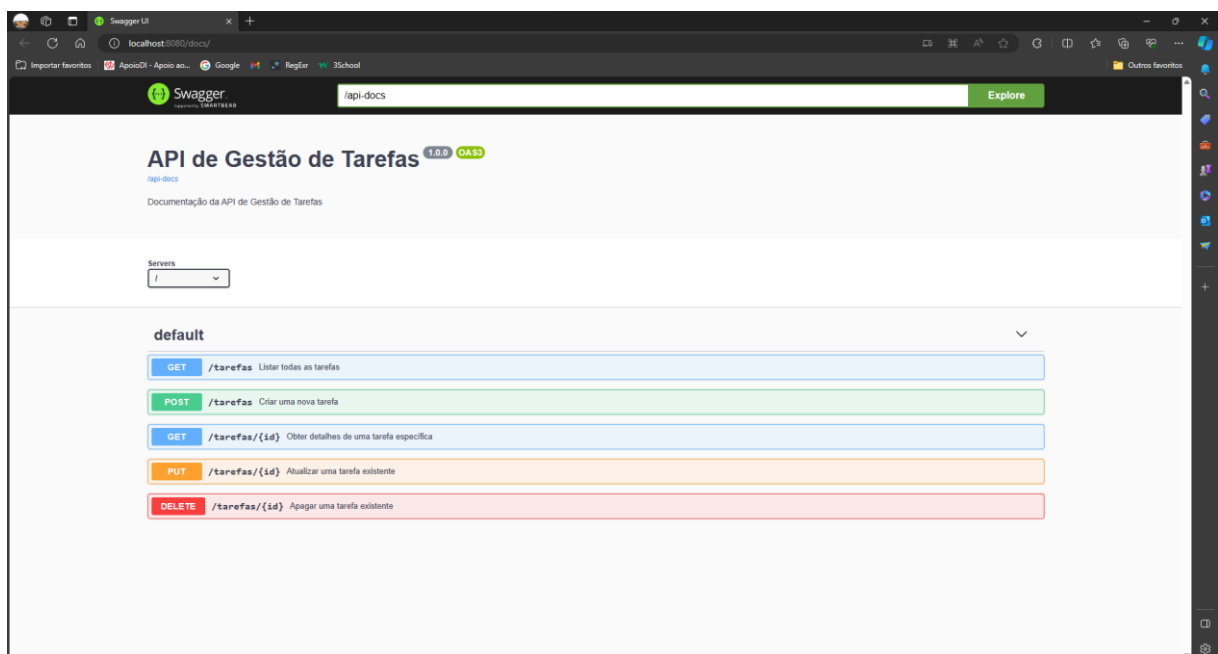


Figura 6-4. Documentação da API

Durante o último sprint, fiz correções e atualizações nos diagramas, no swagger e ao YAML.

6.2. Testes

Utilizando a aplicação “Postman”, procedi à realização de testes para verificar se a API estava devidamente configurada e documentada, como poderá ser verificado nas imagens seguintes:

6.2.1. Tarefas

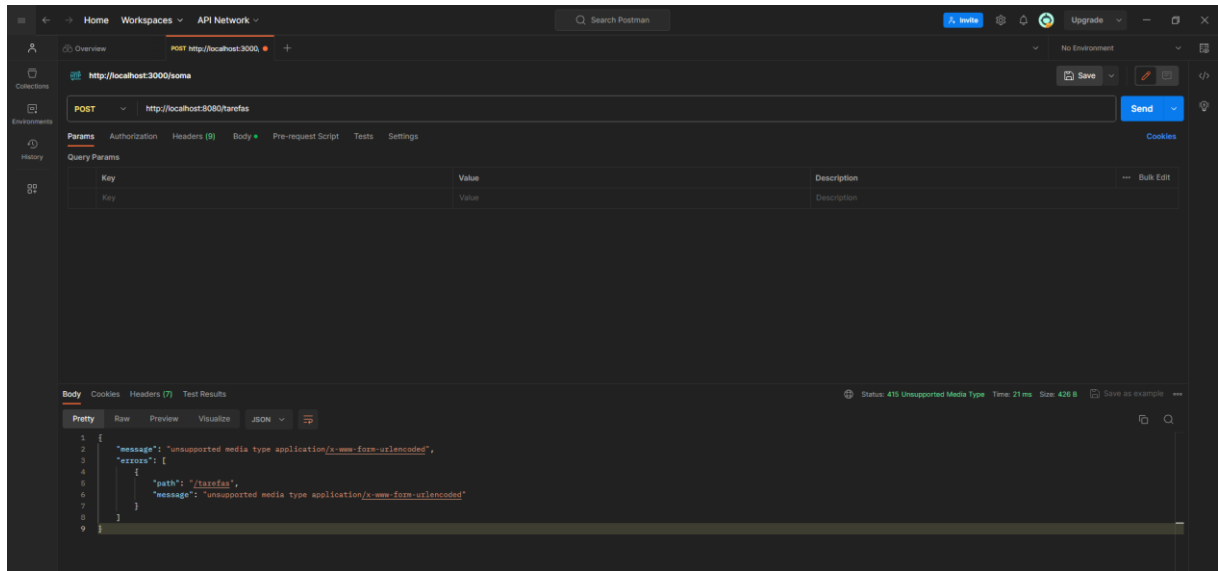


Figura 6-5. Tarefas

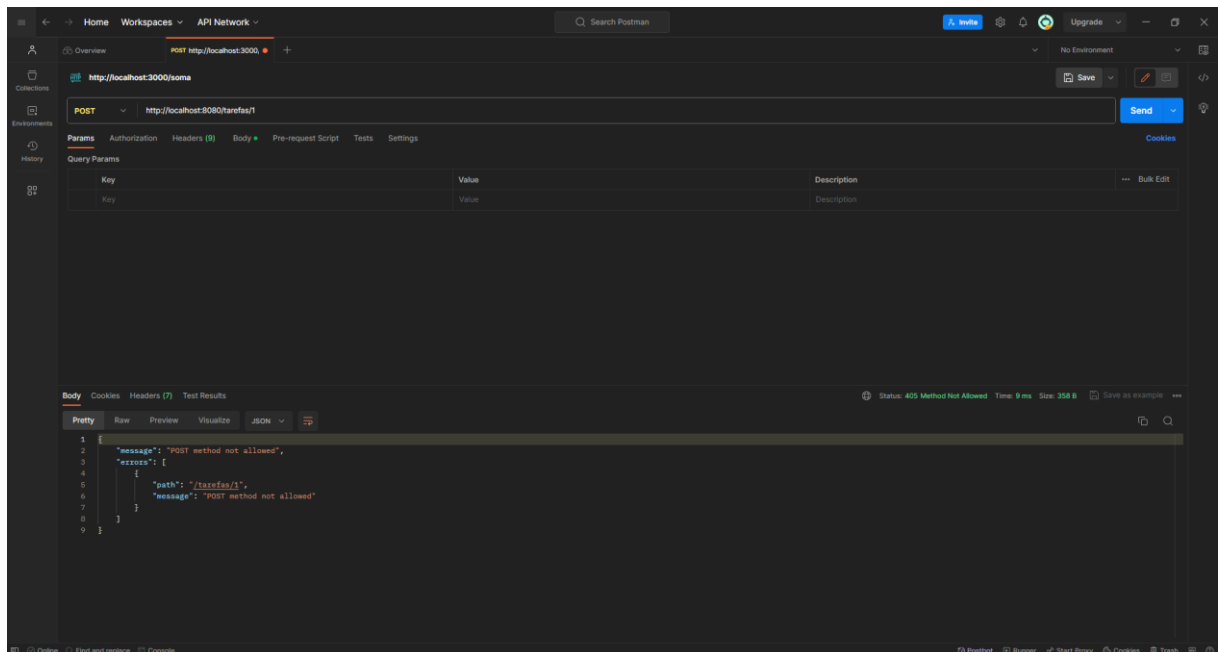


Figura 6-6. Id de Tarefas

7. Conclusão

O trabalho prático de desenvolvimento da API de Gestão de Tarefas representa um marco significativo na jornada do projeto. Ao longo deste percurso, foi criada uma infraestrutura flexível, segura e eficiente para a gestão de tarefas, utilizando as melhores práticas e tecnologias disponíveis.

A definição da arquitetura da API, criando diagramas de contexto e containers que proporcionam uma visão abrangente do sistema. Esses diagramas são fundamentais para compreender quem utiliza o sistema, como ele se integra no ambiente e quais são as tecnologias e responsabilidades distribuídas.

O segundo sprint concentrou-se na criação das estruturas de dados JSON, essenciais para a troca de informações entre o cliente e o servidor. Cada estrutura foi cuidadosamente projetada para suportar as operações fundamentais da API, como adicionar, editar, apagar e listar tarefas. A criação dos JSON Schemas proporciona uma camada adicional de validação e consistência aos dados.

Os restantes sprints foram dedicados à documentação da API em OpenAPI, utilizando o Swagger Editor para garantir a precisão e legibilidade. Cada operação foi detalhadamente descrita, e os JSON Schemas foram integrados para fornecer uma referência clara e abrangente. A consideração das regras REST e dos cabeçalhos HTTP importantes reforça a qualidade e segurança da API.

Em conclusão, o projeto está agora bem encaminhado, com uma arquitetura sólida, estruturas de dados bem definidas e uma documentação detalhada. A colaboração contínua entre as equipes de desenvolvimento e os stakeholders será crucial para ajustes futuros, mas este trabalho estabeleceu uma base robusta para o sucesso contínuo da API de Gestão de Tarefas. Este projeto não é apenas sobre código e tecnologia, mas também sobre a criação de soluções que melhoram a eficiência e organização no gerenciamento de tarefas, promovendo o sucesso do utilizador final.