Dominando APIs:

O Guia Fundamental Para Desenvolvedores

01

Introdução às APIs

01

Neste capítulo introdutório, você será apresentado ao fascinante mundo das APIs *(Application Programming Interfaces)*. Descubra o que são APIs, por que são importantes para os desenvolvedores e como elas facilitam a comunicação entre diferentes sistemas e serviços.

O capítulo inicial deste livro serve como uma porta de entrada para o vasto universo das APIs. Nesta seção introdutória, exploramos o conceito fundamental de APIs, que desempenham um papel essencial no desenvolvimento moderno de software e na integração de sistemas. As APIs são essenciais para a comunicação eficaz entre diferentes componentes de software, permitindo que aplicativos, plataformas e serviços interajam de maneira padronizada e segura. Ao longo deste capítulo, investigaremos não apenas o que são as APIs, mas também por que são tão cruciais para os desenvolvedores e como elas simplificam a complexidade da interação entre sistemas heterogêneos. A compreensão desses conceitos básicos é fundamental para qualquer profissional de tecnologia que deseje criar, integrar ou utilizar software moderno de forma eficaz e eficiente.

As APIs são conjuntos de regras e definições que permitem que diferentes softwares se comuniquem e interajam entre si de maneira predefinida e segura. Elas funcionam como interfaces intermediárias que permitem que uma aplicação utilize os recursos ou serviços de outra aplicação ou sistema de forma controlada, sem precisar conhecer os detalhes internos desse sistema.

Imagine que você está em um restaurante e quer pedir comida, mas você não pode falar com os cozinheiros. O que você faz? Você pede ao garçom para fazer o pedido por você, certo?

Agora, pense nisso em termos de computadores:

- Você é como um aplicativo ou programa que quer usar serviços de outro programa ou sistema (como pedir comida).
- O garçom é como a API. Ele é o intermediário que leva seu pedido à cozinha (ou sistema) e traz a comida de volta para você.
- A cozinha é o sistema ou serviço que realmente prepara a comida, mas você não precisa saber como isso é feito.

Assim como você pode pedir ao garçom um prato específico e ele traz exatamente o que você pediu, você pode usar uma API para solicitar dados ou ações específicas de um sistema ou serviço. Por exemplo, ao usar um aplicativo de previsão do tempo em seu telefone, o aplicativo pode usar uma API para obter os dados meteorológicos atuais de um serviço de previsão do tempo. Você só vê as informações no aplicativo, mas nos bastidores, a API está fazendo a comunicação para obter esses dados para você.

Em resumo, as APIs são como garçons digitais que ajudam diferentes aplicativos a se comunicarem entre si de forma eficiente, permitindo que você (o aplicativo) acesse serviços e informações de outras fontes sem precisar entender os detalhes complicados de como tudo funciona nos bastidores.

Quando você faz um pedido ao garçom, isso é semelhante a fazer uma solicitação a uma API. Você pode solicitar um prato específico (um recurso específico) e talvez especificar como deseja que seja preparado (parâmetros ou dados adicionais na sua requisição).

Assim como o garçom traz de volta o prato que você pediu, a API retorna os dados ou resultados da solicitação que você fez. Pode ser uma resposta com informações, resultados de uma operação ou qualquer outra coisa que você solicitou.

Em termos técnicos, as APIs funcionam com solicitações **HTTP** (ou outros protocolos) que permitem que aplicativos se comuniquem. Por exemplo, ao usar uma API de previsão do tempo, você pode enviar uma solicitação com a localização desejada e receber de volta os dados meteorológicos atuais para essa localização. Da mesma forma, ao integrar um sistema de pagamento em um aplicativo de comércio eletrônico, você usa a API de pagamento para enviar detalhes do pedido e receber uma confirmação de pagamento.

Em resumo, as APIs são fundamentais para a integração de sistemas e serviços, permitindo que aplicativos se comuniquem de forma eficiente e facilitando a criação de soluções mais complexas ao combinar diferentes funcionalidades oferecidas por várias fontes.

Existem vários tipos de APIs, cada um com suas características específicas e usos adequados. Abaixo estão alguns dos tipos mais comuns de APIs:

- **APIs da Web (Web APIs):** As APIs da Web são interfaces expostas na web que permitem que aplicativos se comuniquem com serviços online. Elas são acessadas por meio de URLs e respondem a solicitações utilizando os protocolos HTTP ou HTTPS. As APIs da Web geralmente retornam dados no formato JSON ou XML.
- **APIs RESTful (RESTful APIs):** As APIs RESTful seguem os princípios do estilo arquitetural REST (Representational State Transfer). Elas são baseadas em recursos (como URLs) e usam métodos HTTP (GET, POST, PUT, DELETE) para realizar operações nos recursos. As APIs RESTful são amplamente utilizadas na web devido à sua simplicidade, escalabilidade e facilidade de integração.
- **SOAP (Simple Object Access Protocol):** SOAP é um protocolo de comunicação baseado em XML usado para trocar informações estruturadas em redes. As APIs SOAP definem uma estrutura rigorosa para mensagens, incluindo cabeçalhos e corpos XML. Elas oferecem suporte a recursos avançados como transações e segurança, sendo frequentemente usadas em ambientes corporativos.

GraphQL: GraphQL é uma linguagem de consulta desenvolvida pelo Facebook que permite aos clientes solicitar apenas os dados necessários e especificar a estrutura dos dados desejados. Diferentemente das APIs RESTful, onde os endpoints retornam conjuntos de dados predefinidos, o GraphQL permite consultas flexíveis e personalizadas, reduzindo a sobrecarga de múltiplas chamadas de API.

APIs de Bibliotecas ou SDKs (Software Development Kits): Além das APIs voltadas para comunicação entre sistemas, existem APIs que são disponibilizadas como bibliotecas ou SDKs para facilitar o uso de funcionalidades específicas em linguagens de programação. Por exemplo, uma API de mapa pode ser fornecida como um SDK para integrar mapas em aplicativos móveis.

APIs de Serviços Externos: Muitos serviços externos oferecem APIs para permitir a integração de seus recursos em aplicativos de terceiros. Por exemplo, serviços de pagamento, serviços de mídia social, serviços de entrega de e-mails, entre outros, disponibilizam APIs para permitir interações programáticas com seus sistemas.3

Esses são alguns dos tipos principais de APIs, cada um com suas próprias características e casos de uso específicos. A escolha do tipo de API adequado depende dos requisitos do projeto, da natureza dos dados a serem manipulados e das preferências de design arquitetural.

Protocolos HTTP e HTTPS

HTTP *(Hypertext Transfer Protocol)* e HTTPS *(HTTP Secure)* são protocolos de comunicação usados para transferir dados na web. HTTP é usado para comunicação não criptografada, enquanto HTTPS usa criptografia SSL/TLS para segurança adicional.

JSON e XML

JSON (JavaScript Object Notation) e XML (eXtensible Markup Language) são dois formatos de dados amplamente utilizados para representar e transmitir informações estruturadas na comunicação entre sistemas computacionais.

JSON é um formato leve e fácil de ler para troca de dados entre sistemas. Ele é baseado em uma estrutura de pares chave-valor e é comumente usado para transmitir dados na web.

```json
{
  "nome": "João",
  "idade": 30,
  "cidade": "São Paulo"
}
```

Neste exemplo, temos um objeto JSON com três pares chave-valor: *"nome"* com o valor *"João"*, *"idade"* com o valor *30* e *"cidade"* com o valor *"São Paulo"*.

XML é uma linguagem de marcação que define regras para codificar documentos em um formato legível tanto para humanos quanto para máquinas. Ele é mais extenso e verboso que o JSON, mas é altamente flexível e pode ser usado para representar uma ampla variedade de estruturas de dados.

```xml
<pessoa>
  <nome>João</nome>
  <idade>30</idade>
  <cidade>São Paulo</cidade>
</pessoa>
```

Neste exemplo, temos um documento XML que descreve um elemento **<pessoa>** com três elementos filhos: **<nome>**, **<idade>** e **<cidade>**, cada um contendo um valor específico.

Ambos JSON e XML têm seus usos adequados dependendo do contexto e dos requisitos do sistema, sendo escolhidos com base na necessidade de simplicidade, flexibilidade ou compatibilidade com padrões existentes.

02

Consumindo APIs

Neste capitulo você aprenderá a como consumir (ou usar) uma API em seu próprio código. Veremos exemplos práticos de como fazer solicitações HTTP para uma API e como analisar e manipular dados de resposta.

Vamos utilizar a API REST do *JSONPlaceholder,* que é uma ferramenta gratuita e simples para fazer testes de APIs e não requer chave ou autenticação. Usaremos o endpoint */posts* para fazer uma requisição e obter uma lista de posts.

Exemplo em JavaScript

Primeiramente certifique-se de que tem um ambiente de desenvolvimento JavaScript configurado. Você pode usar o console do navegador ou um editor de código como *VS Code*.

Observe o código a seguir:

```
// URL da API
const url = 'https://jsonplaceholder.typicode.com/posts';

// Função para fazer a requisição HTTP usando a Fetch API
fetch(url)
  .then(response => {
    // Verifica se a resposta foi bem-sucedida
    if (!response.ok) {
      throw new Error('Erro na requisição');
    }
    // Converte a resposta para JSON
    return response.json();
  })
  .then(data => {
    // Manipula os dados de resposta
    console.log('Lista de posts:', data);
  })
  .catch(error => {
    // Manipula erros da requisição
    console.error('Erro:', error);
  });
```

A execução desse script gera uma lista de objetos contendo todos os posts vindo do endpoint */posts*.

Explicação:

URL da API: Define a URL do endpoint /posts do JSONPlaceholder.

fetch(url): Faz uma requisição HTTP GET para a URL especificada.

response.json(): Converte a resposta da API de JSON para um objeto JavaScript.

Manipulação de dados: imprime a lista de posts.

Tratamento de erros: Captura e exibe quaisquer erros que ocorram durante o processo.

Esse foi um exemplo básico de uso de API utilizando JavaScript. Primeiro definimos a URL de destino e utilizamos a função fetch para puxarmos os dados que a plataforma disponibiliza, verificamos alguns casos de possíveis erros que poderiam acontecer no processo e manipulamos a saída do script. Esses passos podem ser feitos utilizando qualquer linguagem de programação de sua preferência.

Exemplo em Python

Agora vamos repetir o mesmo exemplo usando a linguagem Python. Para realizar a execução correta do script, primeiro você precisa da biblioteca *requests* instalada em seu ambiente.

- Abra o terminal e execute o comando: pip install requests

Agora reescreva o código a seguir:

```python
import requests

# URL da API
url = 'https://jsonplaceholder.typicode.com/posts'

try:
    # Fazendo a requisição HTTP GET
    response = requests.get(url)
    response.raise_for_status()  # Verifica se a resposta foi bem-sucedida

    # Converte a resposta para JSON
    data = response.json()

    # Manipula os dados de resposta
    print('Lista de posts:', data)

except requests.exceptions.HTTPError as err:
    # Manipula erros da requisição
    print(f'Erro na requisição: {err}')
```

Explicação:

URL da API: Define a URL do endpoint /posts do JSONPlaceholder.

requests.get(url): Faz uma requisição HTTP GET para a URL especificada.

response.json(): Converte a resposta da API de JSON para um dicionário Python.

Manipulação de dados: imprime a lista de posts.

Tratamento de erros: Captura e exibe quaisquer erros HTTP que ocorram durante o processo.

Nestes exemplos, vimos como consumir uma API REST pública usando JSONPlaceholder em JavaScript e Python. Aprendemos a fazer uma requisição HTTP, converter a resposta para um formato utilizável e manipular os dados recebidos. Esses passos são fundamentais para integrar dados de APIs em suas próprias aplicações de forma eficiente.

Vamos modificar o exemplo em JavaScript anterior para fazer uma requisição ao endpoint que retorna apenas um post específico. Vamos usar o endpoint /posts/1 da API JSONPlaceholder para obter o post com ID 1.

```
// URL da API para obter o post com ID 1
const url = 'https://jsonplaceholder.typicode.com/posts/1';

// Função para fazer a requisição HTTP usando a Fetch API
fetch(url)
  .then(response => {
    // Verifica se a resposta foi bem-sucedida
    if (!response.ok) {
      throw new Error('Erro na requisição');
    }
    // Converte a resposta para JSON
    return response.json();
  })
  .then(data => {
    // Manipula os dados de resposta
    console.log('Post:', data);
    console.log(`Post ID: ${data.id}, Título: ${data.title}, Corpo: ${data.body}`);
  })
  .catch(error => {
    // Manipula erros da requisição
    console.error('Erro:', error);
  });
```

Explicação:

- Definimos a URL *https://jsonplaceholder.typicode.com/posts/1*, que é o endpoint específico para obter o post com ID 1 da API JSONPlaceholder.
- **fetch(url):** A função fetch é usada para fazer uma requisição HTTP GET para a URL especificada.
- **.then(response => { ... }):** Esta função é chamada quando a requisição é completada. O parâmetro response contém a resposta da API.
- **response.ok:** Verifica se a resposta foi bem-sucedida (código de status HTTP na faixa 200-299). Se não for bem-sucedida, lança um erro.
- **response.json():** Converte a resposta da API de JSON para um objeto JavaScript. Isso retorna uma Promise, que é resolvida com o conteúdo JSON do corpo da resposta.
- **.then(data => { ... }):** Esta função é chamada quando a conversão JSON é completada. O parâmetro data contém os dados do post.
- **console.log('Post:', data):** Imprime o objeto completo do post no console.
- **console.log(Post ID: ${data.id}, Título: ${data.title}, Corpo: ${data.body}):** Imprime o ID, título e corpo do post no console de forma mais legível.
- **.catch(error => { ... }):** Esta função é chamada se qualquer erro ocorrer durante a requisição ou a conversão JSON. O parâmetro error contém informações sobre o erro.
- **console.error('Erro:', error):** Imprime uma mensagem de erro no console.

Este script em JavaScript faz uma requisição a uma API REST pública para obter um único post com ID 1. O uso da Fetch API facilita a requisição HTTP, a conversão da resposta em JSON e a manipulação dos dados retornados. Além disso, o tratamento de erros garante que qualquer problema durante o processo seja capturado e exibido de forma adequada.

Tente fazer o mesmo para o script em Python!

03

Criando suas próprias APIs

Neste capitulo vamos dar um passo além e aprender a criar nossas próprias APIs. Descubra como projetar endpoints RESTful, criar documentação clara para desenvolvedores, e implementar sua API usando linguagens populares como Python ou Node.js.

A importância de saber criar e usar suas próprias APIs

No desenvolvimento de software moderno, a habilidade de criar APIs (Application Programming Interfaces) é fundamental para programadores back-end. As APIs atuam como pontes entre diferentes sistemas e aplicações, permitindo que eles se comuniquem e compartilhem dados de maneira eficiente e segura. Aprender a criar suas próprias APIs oferece inúmeras vantagens e abre diversas possibilidades para a construção de software robusto e escalável.

APIs são essenciais para a comunicação entre diferentes sistemas e serviços. Elas permitem que diferentes partes de um sistema, ou mesmo sistemas completamente distintos, troquem informações de maneira padronizada. Por exemplo, um aplicativo móvel pode usar uma API para interagir com um servidor que armazena dados do usuário. Sem APIs, essa comunicação seria complexa e propensa a erros.

Ao criar APIs, os programadores podem modularizar funcionalidades específicas de um software, tornando-as reutilizáveis em diferentes partes do mesmo projeto ou até mesmo em projetos diferentes. Isso leva a uma maior eficiência no desenvolvimento, já que componentes bem definidos podem ser facilmente adaptados e reutilizados. APIs permitem que aplicações se integrem facilmente com serviços de terceiros, como gateways de pagamento, serviços de envio de e-mails, plataformas de redes sociais, e muitos outros. Por exemplo, uma loja online pode usar uma API de pagamento para processar transações de maneira segura e eficiente.

Criando uma API em Node.js com Express

Neste exemplo, vamos criar uma API simples usando o framework Express em Node.js. A API permitirá realizar operações CRUD (Create, Read, Update, Delete) em uma coleção de "posts". Vamos detalhar cada passo do processo e mostrar como rodar o código.

- Certifique-se de que o Node.js e o npm estão instalados no seu sistema. Você pode baixá-los do site oficial Node.js: *https://nodejs.org*

- Crie um novo diretório para o projeto e inicialize um novo projeto Node.js.

```
$ mkdir my-api
$ cd my-api
$ npm init -y
```

- Instale o Express.

```
$ npm install express
```

- Crie um arquivo chamado index.js dentro do diretório atual (my-api).

Escrevendo o código passo a passo:

- **Importando o express**

```
const express = require('express');
const app = express();
const port = 3000;
```

Importamos o Express e criamos uma instância da aplicação Express. Definimos a porta em que o servidor vai rodar.

- **Criando middleware para JSON**

```
app.use(express.json());
```

Usamos o middleware express.json() para permitir o parsing de corpos de requisições em JSON.

- ## Criando uma lista de posts

```
let posts = [
  {
    id: 1,
    title: 'Primeiro Post',
    content: 'Conteúdo do primeiro post'
  },
  {
    id: 2,
    title: 'Segundo Post',
    content: 'Conteúdo do segundo post'
  }
];
```

Definimos uma lista de posts como exemplo. Estes serão manipulados através das operações de CRUD da nossa API.

- ## Criando a rota POST (/posts)

```
app.post('/posts', (req, res) ⇒ {
  const newPost = {
    id: posts.length + 1,
    title: req.body.title,
    content: req.body.content
  };

  posts.push(newPost);
  res.status(201).json(newPost);
});
```

Esta rota cria um novo post. Os dados do novo post são enviados no corpo da requisição.

- ## Criando a rota GET (/posts)

```
app.get('/posts', (req, res) ⇒ {
  res.json(posts);
});
```

Esta rota retorna a lista de todos os posts.

- ## Criando a rota GET (/posts/:id)

```
app.get('/posts/:id', (req, res) => {
  const post = posts.find(p => p.id == req.params.id);
  if (post) {
    res.json(post);
  } else {
    res.status(404).send('Post não encontrado');
  }
});
```

Esta rota retorna um post específico com base no ID.

- ## Criando a rota PUT (/posts/:id)

```
app.put('/posts/:id', (req, res) => {
  const post = posts.find(p => p.id == req.params.id);
  if (post) {
    post.title = req.body.title;
    post.content = req.body.content;
    res.json(post);
  } else {
    res.status(404).send('Post não encontrado');
  }
});
```

Esta rota atualiza um post específico com base no ID.

- ## Criando a rota DELETE (/posts/:id)

```
app.delete('/posts/:id', (req, res) => {
  const postIndex = posts.findIndex(p => p.id == req.params.id);
  if (postIndex !== -1) {
    posts.splice(postIndex, 1);
    res.status(204).send();
  } else {
    res.status(404).send('Post não encontrado');
  }
});
```

Esta rota deleta um post específico com base no ID.

- ## Iniciando o servidor

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

Com estes passos, você criou uma API RESTful completa em Node.js usando Express, que suporta operações CRUD básicas.

- # Código completo

```
// index.js

const express = require('express');
const app = express();
const port = 3000;

app.use(express.json()); // Middleware para permitir JSON no corpo das requisições

let posts = [
  { id: 1, title: 'Primeiro Post', content: 'Conteúdo do primeiro post' },
  { id: 2, title: 'Segundo Post', content: 'Conteúdo do segundo post' }
];

// Create - POST /posts
app.post('/posts', (req, res) => {
  const newPost = {
    id: posts.length + 1,
    title: req.body.title,
    content: req.body.content
  };
  posts.push(newPost);
  res.status(201).json(newPost);
});

// Read - GET /posts
app.get('/posts', (req, res) => {
  res.json(posts);
});

// Read - GET /posts/:id
app.get('/posts/:id', (req, res) => {
  const post = posts.find(p => p.id == req.params.id);
  if (post) {
    res.json(post);
  } else {
    res.status(404).send('Post não encontrado');
  }
});

// Update - PUT /posts/:id
app.put('/posts/:id', (req, res) => {
  const post = posts.find(p => p.id == req.params.id);
  if (post) {
    post.title = req.body.title;
    post.content = req.body.content;
    res.json(post);
  } else {
    res.status(404).send('Post não encontrado');
  }
});

// Delete - DELETE /posts/:id
app.delete('/posts/:id', (req, res) => {
  const postIndex = posts.findIndex(p => p.id == req.params.id);
  if (postIndex !== -1) {
    posts.splice(postIndex, 1);
    res.status(204).send();
  } else {
    res.status(404).send('Post não encontrado');
  }
});

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

- ## Como rodar o código

Salve o código acima no arquivo index.js. No terminal, navegue até o diretório do projeto e execute o seguinte comando: node index.js.

Use ferramentas como Postman ou Insomnia, ou mesmo o cURL no terminal, para fazer requisições aos endpoints da API.

1. Criando um novo post (POST /posts)

Para criar um novo post, você usará o método HTTP POST com um corpo JSON. Aqui está o comando cURL:

```
curl -X POST http://localhost:3000/posts \
     -H "Content-Type: application/json" \
     -d '{"title": "Novo Post", "content": "Conteúdo do novo post"}'
```

- **-X POST:** Especifica o método HTTP POST.
- **-H "Content-Type: application/json":** Define o cabeçalho para indicar que o corpo da requisição é JSON.
- **-d '{"title": "Novo Post", "content": "Conteúdo do novo post"}':** É o corpo da requisição em JSON.

2. Listando todos os posts (GET /posts)

Para listar todos os posts, você usará o método HTTP GET. Aqui está o comando cURL:

```
curl http://localhost:3000/posts
```

- Este comando simples faz uma requisição GET para a URL especificada e retorna a lista de posts.

3. Obtendo um post específico (GET /posts/:id)

Para obter um post específico com base no ID, você usará o método HTTP GET com o ID do post na URL. Aqui está o comando cURL:

```
curl http://localhost:3000/posts/1
```

- Substitua 1 pelo ID do post que você deseja recuperar.

4. Atualizando um post (PUT /posts/:id)

Para atualizar um post específico com base no ID, você usará o método HTTP PUT com um corpo JSON. Aqui está o comando cURL:

```
curl http://localhost:3000/posts
```

- **-X PUT:** Especifica o método HTTP PUT.
- **-H "Content-Type: application/json":** Define o cabeçalho para indicar que o corpo da requisição é JSON.
- **-d '{"title": "Post Atualizado", "content": "Conteúdo atualizado"}':** O corpo da requisição em JSON.
- Substitua 1 pelo ID do post que você deseja atualizar.

5. Deletar um post (DELETE /posts/:id)

Para deletar um post específico com base no ID, você usará o método HTTP DELETE. Aqui está o comando cURL:

```
curl -X DELETE http://localhost:3000/posts/1
```

- **-X DELETE:** Especifica o método HTTP DELETE.
- Substitua 1 pelo ID do post que você deseja deletar.

Com esses comandos cURL, você pode manipular a API que criamos em Node.js usando Express, realizando operações de criação, leitura, atualização e deleção de posts.

Como exercício. Tente replicar ou consumir essa API com a linguagem de programação que você domina!

Segue um exemplo utilizando Python:

```python
import requests

# URL base da API
BASE_URL = 'http://localhost:3000/posts'

def get_all_posts():
    response = requests.get(BASE_URL)
    if response.status_code == 200:
        posts = response.json()
        print("Lista de todos os posts:")
        for post in posts:
            print(f"ID: {post['id']}, Título: {post['title']}")
    else:
        print(f"Erro ao obter posts: {response.status_code}")

def get_post_by_id(post_id):
    url = f"{BASE_URL}/{post_id}"
    response = requests.get(url)
    if response.status_code == 200:
        post = response.json()
        print(f"Detalhes do post ID {post_id}:")
        print(f"Título: {post['title']}")
        print(f"Conteúdo: {post['content']}")
    else:
        print(f"Erro ao obter o post: {response.status_code}")

if __name__ == "__main__":
    # Obter todos os posts
    get_all_posts()

    # Obter um post específico pelo ID
    post_id = 1
    get_post_by_id(post_id)
```

04

Estudos de caso e exemplos do mundo real

Como você já imagina, as empresas desenvolvem e fornecem APIs para permitir que outras empresas, desenvolvedores e serviços interajam com seus sistemas de maneira controlada e segura. Essas APIs são projetadas para expor funcionalidades específicas do software ou dados da empresa, facilitando a integração com outros sistemas.

O Google fornece APIs para diversos serviços, como Google Maps, YouTube, e Google Drive, permitindo que desenvolvedores integrem funcionalidades de localização, vídeos e armazenamento em nuvem em seus próprios aplicativos. O X (Twitter) disponibiliza uma API que permite a criação de aplicações que podem ler e publicar tweets, gerenciar seguidores e acessar dados de usuários. Empresas como Uber e Lyft utilizam APIs de mapas (Google Maps) e APIs de pagamento (Stripe) para fornecer um serviço completo que inclui navegação e processamento de pagamentos.

Em resumo, as APIs são fundamentais para o funcionamento eficiente e inovador das empresas modernas.

Consumindo a API do GitHub

Vamos criar um script em Python que usa a API do GitHub para buscar dados de um usuário, incluindo a foto de perfil, nome, nickname, e repositórios. Usaremos a biblioteca requests para fazer as requisições HTTP.

Se você ainda não tem a biblioteca requests instalada, você pode instalá-la usando o seguinte comando:

```
$ pip install requests
```

O código a seguir faz a busca dos dados do usuário *Linus Torvalds e exibe no terminal.*

```python
import requests

def get_github_user_info(username):
    # URL da API para obter informações do usuário
    user_url = f"https://api.github.com/users/{username}"
    # URL da API para obter repositórios do usuário
    repos_url = f"https://api.github.com/users/{username}/repos"

    # Fazendo a requisição para obter as informações do usuário
    user_response = requests.get(user_url)
    if user_response.status_code == 200:
        user_data = user_response.json()
        print(f"Foto de perfil: {user_data['avatar_url']}")
        print(f"Nome: {user_data.get('name', 'N/A')}")
        print(f"Nickname: {user_data['login']}")
    else:
        print(f"Erro ao obter informações do usuário: {user_response.status_code}")
        return

    # Fazendo a requisição para obter os repositórios do usuário
    repos_response = requests.get(repos_url)
    if repos_response.status_code == 200:
        repos_data = repos_response.json()
        print("Repositórios:")
        for repo in repos_data:
            print(f"- {repo['name']}")
    else:
        print(f"Erro ao obter repositórios do usuário: {repos_response.status_code}")

if __name__ == "__main__":
    # Substitua 'torvalds' pelo nome de usuário do GitHub que você deseja consultar
    username = "torvalds"
    get_github_user_info(username)
```

Este script simples permite que você use a API do GitHub para buscar e exibir informações básicas sobre qualquer usuário do GitHub. É uma maneira prática de começar a trabalhar com APIs e entender como fazer requisições HTTP para obter dados.

Podemos ir além e exibir esses mesmos recursos buscados usando python, porém vamos criar uma página HTML simples que faz uma requisição à API do GitHub e exibe os dados do usuário, como foto de perfil, nome, nickname e repositórios, usando JavaScript.

Arquivo index.html

```html
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>GitHub API</title>
</head>
<body>
    <h1>Buscando dados de usuários do Github</h1>
    <input type="text" id="username" placeholder="Entre com um nome de usuário" />
    <button onclick="getGithubUserInfo()">Buscar informações</button>

    <div id="user-info">
        <img id="avatar" src="" alt="Profile Picture" width="150" style="display:none"/>
        <p><strong>Nome:</strong> <span id="name"></span></p>
        <p><strong>Nickname:</strong> <span id="nickname"></span></p>
        <h3>Repositórios:</h3>
        <ul id="repos"></ul>
    </div>

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

- Um campo de entrada (input) para digitar o nome de usuário do GitHub.
- Um botão (button) para iniciar a busca de informações do usuário.
- Um div (div) para exibir as informações do usuário (foto de perfil, nome, nickname e repositórios).

Arquivo script.js

```
async function getGithubUserInfo() {
  const username = document.getElementById('username').value;
  const userUrl = `https://api.github.com/users/${username}`;
  const reposUrl = `https://api.github.com/users/${username}/repos`;

  try {
    const userResponse = await fetch(userUrl);
    if (userResponse.ok) {
      const userData = await userResponse.json();
      document.getElementById('avatar').src = userData.avatar_url;
      document.getElementById('avatar').style.display = 'block';
      document.getElementById('name').textContent = userData.name || 'N/A';
      document.getElementById('nickname').textContent = userData.login;
    } else {
      alert('Usuário não encontrado!');
      return;
    }

    const reposResponse = await fetch(reposUrl);
    if (reposResponse.ok) {
      const reposData = await reposResponse.json();
      const reposList = document.getElementById('repos');
      reposList.innerHTML = '';
      reposData.forEach(repo => {
        const listItem = document.createElement('li');
        listItem.textContent = repo.name;
        reposList.appendChild(listItem);
      });
    } else {
      alert('Erro ao buscar repositórios!');
    }
  } catch (error) {
    console.error('Error:', error);
  }
}
```

- A função getGithubUserInfo é chamada quando o botão é clicado.
- A função usa fetch para fazer requisições GET às URLs da API do GitHub para obter as informações do usuário e seus repositórios.
- Se a resposta da API for bem-sucedida, os dados do usuário são exibidos na página.
- A imagem do perfil é exibida apenas se a requisição for bem-sucedida.

Como usar

- Salve o código HTML acima em um arquivo chamado index.html.
- Abra o arquivo index.html em um navegador.
- Digite um nome de usuário do GitHub no campo de entrada e clique no botão "Get User Info".
- As informações do usuário, incluindo a foto de perfil, nome, nickname e a lista de repositórios, serão exibidas na página.

Conclusão

Ao longo deste ebook, você embarcou em uma jornada pelo vasto e dinâmico mundo das APIs. Desde os conceitos básicos até a criação de suas próprias APIs, você agora possui uma compreensão sólida de como essas interfaces podem transformar a maneira como você desenvolve e integra aplicações.

Lembre-se de que o aprendizado contínuo é fundamental. O universo das APIs está em constante evolução, com novas tecnologias, padrões e práticas emergindo regularmente. Mantenha-se atualizado com as tendências e continue explorando novas formas de utilizar APIs para maximizar o potencial de seus projetos.

OBRIGADO POR LER ATÉ AQUI

Este material foi desenvolvido e gerado por IA com o apoio e cuidado humano para fins didáticos.

github.com/ianpatricck