Uma Análise Comparativa de Frameworks para Contratos de API: OpenAPI, Spring Cloud Contract e GraphQL

Beatriz R. Navarro, Wylliams B. Santos

CESAR School

Caixa Postal 50030-390 – Recife – PE – Brasil

brn@cesar.school, wbs@cesar.school

Introdução

Um desafio recorrente para as equipes de desenvolvimento de software, diante da variedade de frameworks disponíveis, é a escolha de quais destes serão utilizados durante o desenvolvimento do projeto. Nem sempre se opta por um único framework; frequentemente, o sistema é construído em partes que empregam abordagens diferentes. Contudo, a decisão sobre a melhor opção, dentre as diversas disponíveis, representa um grande desafio.

Diante desse contexto desafiador, este artigo propõe uma comparação entre três frameworks destinados, principalmente, à implementação de contratos de APIs: OpenAPI, Spring Cloud Contract e GraphQL. O intuito é simplificar a busca por literatura comparativa e, por consequência, orientar a escolha da melhor opção para cada necessidade e tipo específico de projeto.

Definições e Conceitos

API

É um acrônimo para **A**pplication **P**rogram **I**nterface ou **I**nterface de **P**rogramação de **A**plicações em português. Segundo (MUNIZ et al., 2019) APIs "são mecanismos que permitem que dois componentes de software se comuniquem usando um conjunto de definições e protocolos". Essas interfaces podem ser web, que são as mais amplamente conhecidas, porém existem outros tipos como APIs de Banco de dados, APIs de Sistema Operacional, APIs de Hardware etc.

A Figura 1 apresenta uma metáfora para o funcionamento de uma API.

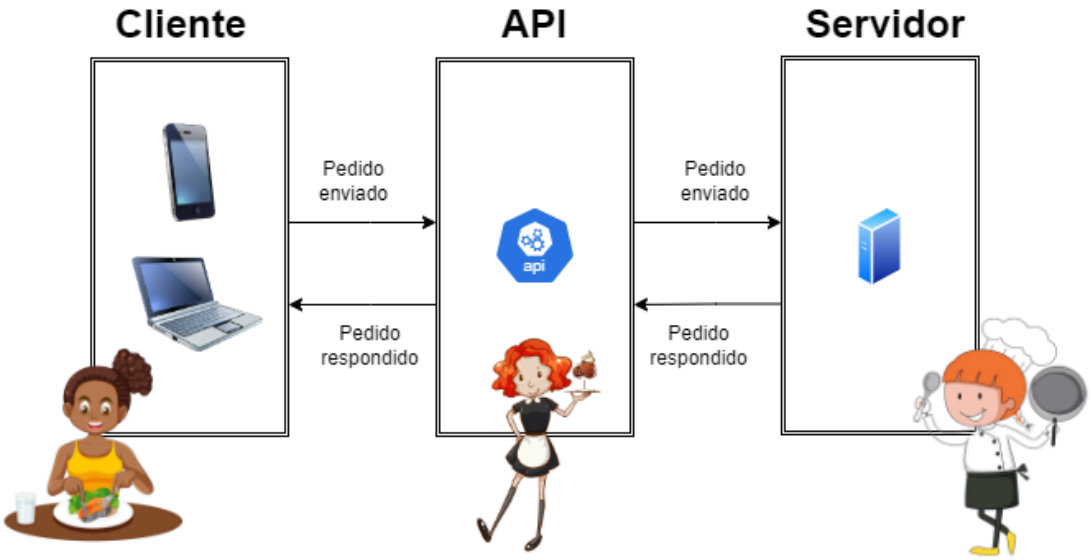

Figura 1 - Uma metáfora para APIs
Fonte: Esta capa foi projetada usando recursos do Freepik.

Contrato

Contrato é um acordo realizado entre duas ou mais partes que precisam trabalhar em conjunto. São regras definidas que facilitam a utilização de algum produto e/ou sistema; bem como o seu próprio desenvolvimento. Qualquer alteração em uma dessas regras deverá ser comunicada a todas as partes envolvidas, caso contrário esse contrato foi quebrado.

Framework

Existem diversas definições para o conceito de framework disponíveis na literatura, todas complementares e não excludentes. Segundo (Govoni, 1999) Framework é "uma abstração de uma coleção de classes, interfaces e padrões dedicados a resolver um grupo de problemas através de uma arquitetura flexível e extensível."

REST

É um acrônimo para **RE**presentational **S**tate **T**ransfer ou **T**ransferência de **E**stado **R**epresentativo em português. É um protocolo de desenvolvimento de serviços web que foi descrito na tese de doutorado de Roy Fielding em 2000. O protocolo REST, entre outras coisas, preza pelo

- Uso adequado dos métodos HTTP;
- Uso de status de sucesso e falha padronizados;
- Utilização otimizada de cabeçalhos HTTP;
- Interligação eficiente entre os recursos disponibilizados pelo serviço.

SOAP

É um acrônimo para **S**imple **O**bject **A**ccess **P**rotocol ou **P**rotocolo **S**imples de **A**cesso a **O**bjetos, segundo (RAMALHO, 2013) SOAP é "um protocolo de comunicação simples (pouco exigente), baseado em XML e destinado à troca de informação tipada e

estruturada entre aplicações distribuídas e descentralizadas". Frequentemente utilizado em projetos onde segurança é extremamente importante.

XML

É o acrônimo para e**X**tensible **M**arkup **L**anguage ou **L**inguagem **E**xtensível de **M**arcação em português. É definida como uma metalinguagem que marca partes de um texto, geralmente considerada uma opção mais verbosa que outras linguagens como JSON ou YAML. Na Figura 2 vemos um exemplo de xml.

Input XML

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <row>
    <author>Mark Haddon</author>
    <language>English</language>
    <link>https://www.goodreads.com/book/show/1618.The_Curious_Incident_of_the_Dog_in_the_Night_Time</link>
    <pages>226</pages>
    <title>The Curious Incident of the Dog in the Night-Time</title>
    <year>2004</year>
  </row>
  <row>
    <author>Marie Benedict</author>
    <language>English</language>
    <link>https://www.goodreads.com/book/show/39971465-the-only-woman-in-the-room</link>
    <pages>312</pages>
    <title>The Only Woman in the Room</title>
    <year>2019</year>
  </row>
</root>

```

Figura 2 - Exemplo XML

Yaml

É um acrônimo para "YAML Ain't Markup Language" ou "não é uma linguagem de marcação" em português, ou seja, não é uma linguagem de marcação como o HTML usado para criação de conteúdo. É uma linguagem desenvolvida para trabalhar com dados de forma concisa, clara e amigável, facilitando a compreensão por humanos. Em arquivos YAML a correta indentação é essencial. Na Figura 3 vemos um exemplo de Yaml.

Output

```yaml
- author: Mark Haddon
  language: English
  link: >-
    https://www.goodreads.com/book/show/1618.The_Curious_Incident_of_the_Dog_in_the_Night_Time
  pages: 226
  title: The Curious Incident of the Dog in the Night-Time
  year: 2004
- author: Marie Benedict
  language: English
  link: https://www.goodreads.com/book/show/39971465-the-only-woman-in-the-room
  pages: 312
  title: The Only Woman in the Room
  year: 2019

```

Figura 3 - Exemplo YAML

JSON

É um acrônimo para **J**ava**S**cript **O**bject **N**otation ou **N**otação de **O**bjetos **J**ava**S**cript em português. É uma linguagem hierárquica, autodescritiva, de fácil entendimento e que

possibilita a manipulação de dados de forma simples. Na Figura 4 vemos um exemplo de JSON.


```json
[
  {
    "author": "Mark Haddon",
    "language": "English",
    "link": "https://www.goodreads.com/book/show/1618.The_Curious_Incident_of_the_Dog_in_the_Night_Time",
    "pages": 226 ,
    "title": "The Curious Incident of the Dog in the Night-Time",
    "year": 2004
  },
  {
    "author": "Marie Benedict",
    "language": "English",
    "link": "https://www.goodreads.com/book/show/39971465-the-only-woman-in-the-room",
    "pages": 312 ,
    "title": "The Only Woman in the Room",
    "year": 2019
  }
]
```


Figura 4 - Exemplo JSON

APIs RESTful

Uma API é considerada restful quando implementa os princípios do REST. No REST são implementadas múltiplas URLs, cada uma com sua responsabilidade única na manipulação dos dados. Utiliza o protocolo HTTP e é um tipo de arquitetura conhecida como stateless, ou seja, não armazena o estado da requisição, cada solicitação é única, independente e contém toda a informação necessária.

Anatomia de uma requisição HTTP

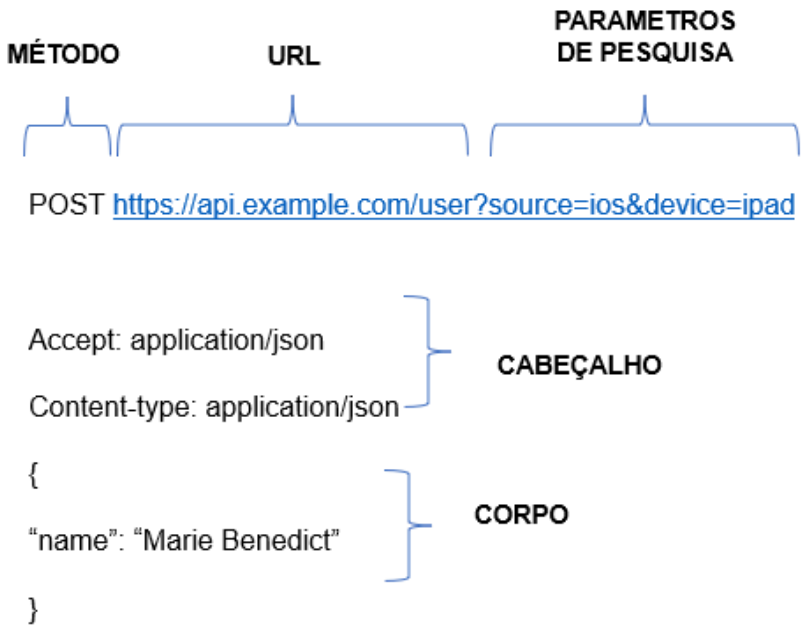

Figura 5 - Anatomia de uma requisição HTTP

Método HTTP	Operação
GET	Leitura
POST	Criação
DELETE	Exclusão
PUT	Atualização completa
PATCH	Atualização parcial

Principais Status Codes HTTP	
2XX	Sucesso - 200, 201, 202
3XX	Redirecionamento – 301, 302
4XX	Falha no cliente – 400, 401, 403, 404
5XX	Falha no servidor – 500, 503

Segurança

Em APIs Restful é possível a implementação de diferentes tipos de autenticação. Importante lembrar que existem dois conceitos comumente confundidos, são eles: autenticação e autorização. Quando um usuário está autenticado significa que seu usuário e senha, ou seu token de acesso, foram validados e estão corretos. Quando o usuário e/ou o token possui autorização, significa que ele tem acesso a algum recurso específico do sistema. Por exemplo, eu posso logar em um sistema com sucesso (autenticação), porém ao tentar acessar algum relatório recebo o erro 403 – Forbidden, pois não tenho autorização para acessar aquele recurso específico.

Como já mencionado existem diferentes tipos de autenticação para APIs restful, as principais são:

- Basic Auth
- API Key
- OAuth

Frameworks

OpenAPI (Swagger)

Inicialmente “Swagger” era uma especificação de como criar o arquivo de definição de uma API, porém após o lançamento da versão 2.0 ele tornou-se o “Open API Specification (OAS)”.

Hoje o Swagger é um conjunto de ferramentas que utilizam a especificação OpenAPI, entretanto ambos os termos ainda são usados indistintamente.

A especificação OpenAPI é um framework baseado em REST, portanto seguirá os conceitos definidos na seção anterior. Utiliza dados estruturados para a criação do arquivo de definição da API, podendo ser JSON ou YAML, sempre respeitando o formato chave-valor.

Algumas chaves importantes

Configuração		
Chave	**Tipo**	**Descrição**
host	String	nome do host (domínio). Exemplos: "api.example.com" ou "api.example.com:8080"
basePath	String	Representa um prefixo comum a todos os endpoints. Exemplo: “/v1”
schemes	Array de strings	Define os protocolos de transferência aceitos pela API. Exemplos:[“https”, “http”]
consumes	Array de strings	Define o tipo de conteúdo consumido pela API. Aceito como entrada de dados. Exemplos: ["application/json", "application/xml"].
produces	Array de strings	Define o tipo de conteúdo utilizado pela API em suas respostas. Exemplos: ["application/json", "application/xml"].

Requests		
Chave	**Tipo**	**Descrição**
$ref	String	Referência para uma estrutura/componente no mesmo arquivo YAML
name	String	Nome do parâmetro
In	String	Indica onde o parâmetro será disponibilizado. Exemplos: body, formdata, path, header, query.
Required	true ou false	Indica se o parâmetro é obrigatório ou não.
type	String	integer, string, boolean, number, object, array, null.

Segurança		
Chave	**Tipo**	**Descrição**
type	String	Define o tipo de segurança a ser implementado. Exemplos: basic, oauth2, apiKey,

Erros		
Chave	**Tipo**	**Descrição**
message	String	Mensagem descritiva sobre o erro que ocorreu.

Exemplo arquivo de definição de API:

```yaml
openapi: 3.0.0
info:
  title: Minha Biblioteca API
  description: API para consulta de livros na biblioteca
  version: 1.0.0
servers:
  - url: https://api.minhabiblioteca.com/v1
```

```yaml
paths:
  /livros:
    get:
      summary: Retorna uma lista de livros
      description: |
        Retorna uma lista de livros disponíveis na biblioteca.
      security:
        - basicAuth: []
      responses:
        '200':
          description: OK. Lista de livros recuperada com sucesso.
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/LivrosResponse'
        '401':
          description: Não autorizado. Falha na autenticação.
        '403':
          description: Proibido. O usuário não tem permissão para acessar este livro.
        '404':
          description: Não encontrado. O livro solicitado não foi encontrado.
        '500':
          description: Erro interno do servidor. Tente novamente mais tarde.
components:
  securitySchemes:
    basicAuth:
      type: http
      scheme: basic
  schemas:
    Livro:
      type: object
      properties:
        id:
          type: integer
          format: int64
          description: ID único do livro
        titulo:
          type: string
          description: Título do livro
        autor:
          type: string
          description: Autor do livro
        ano_publicacao:
          type: integer
          description: Ano de publicação do livro
      example:
        id: 1
        titulo: "Memórias Póstumas de Brás Cubas"
        autor: "Machado de Assis"
        ano_publicacao: 1881
    LivrosResponse:
      type: object
      properties:
        livros:
          type: array
```

```yaml
      items:
        $ref: '#/components/schemas/Livro'
  example:
    livros:
      - id: 1
        titulo: "Memórias Póstumas de Brás Cubas"
        autor: "Machado de Assis"
        ano_publicacao: 1881
      - id: 2
        titulo: "A Hora da Estrela"
        autor: "Clarice Lispector"
        ano_publicacao: 1977
```

Melhores práticas no OpenAPI

- Adição de descrições claras e concisas para a API;
- Adição de descrições claras e concisas para as operações;
- Adição de descrições claras e concisas para os parâmetros;
- Documentação dos erros que podem acontecer;
- Definição de um formato de resposta único;
- Adição de links para documentação e ajuda externa, caso existam;
- Reutilização dos componentes criados;
- Prevenção de código duplicado utilizando referências ($ref);
- Inclusão de exemplos de requests e responses;
- Utilização dos status code de forma apropriada;
- Prevenção de uso do mesmo status code para diferentes contextos.

Spring Cloud Contract

É uma biblioteca que implementa o padrão Consumer-Driven Contracts (CDC). Neste padrão teremos sempre dois atores principais, o Produtor e o Consumidor.

O arquivo de definição de contrato no Spring pode ser escrito em Groovy ou em YAML, e a partir dele são gerados os seguintes artefatos:

- **JSON stub definitions:** arquivo que irá definir a estrutura aceita pela API;
- **Messaging routes:** quando usamos um sistema baseado em mensageria é possível configurar como as mensagens serão gerenciadas. Basicamente, “Messaging routes” são as configurações que podem ser aplicadas as mensagens;
- **Acceptance tests:** São os testes que vão ser executados para definir se o sistema está fazendo o que deveria fazer da melhor forma possível (seguindo o contrato do ponto de vista do usuário/consumidor).

Contratos Orientados pelo Consumidor (CDC)

Neste padrão os consumidores são responsáveis pela definição do contrato, enquanto os produtores devem garantir o cumprimento do contrato.

Exemplo de DSL em Groovy:

```
import org.springframework.cloud.contract.spec.Contract

Contract.make {
  description 'Retorna uma lista de livros'

  request {
    method 'GET'
    url '/livros'
    headers {
      contentType 'application/json'
      header('Authorization', 'Basic dXNlcjpwYXNzd29yZA==')
    }
  }

  response {
    status 200
    headers {
      contentType 'application/json'
    }
    body([
      {
        id: $(regex("[0-9]+")),
        titulo: 'Memórias Póstumas de Brás Cubas',
        autor: 'Machado de Assis',
        anoPublicacao: 1881
      },
      {
        id: $(regex("[0-9]+")),
        titulo: 'A Hora da Estrela',
        autor: 'Clarice Lispector',
        anoPublicacao: 1977
      }
    ])
  }

  response {
    status 401
    headers {
      contentType 'application/json'
    }
    body([
      erro: 'Unauthorized',
      mensagem: 'Credenciais inválidas'
    ])
  }

  response {
    status 404
    headers {
      contentType 'application/json'
    }
    body([
      erro: 'Not Found',
```

```
        mensagem: 'Recurso não encontrado'
      ])
  }
}
```

Exemplo de stub:

```
{
 "endpoints": [
  {
   "method": "GET",
   "path": "/books",
   "request": {
    "headers": {
     "Authorization": "Basic dXNlcm5hbWU6cGFzc3dvcmQ="
    }
   },
   "response": {
    "status": 200,
    "headers": {
     "Content-Type": "application/json"
    },
    "body": [
     {
      "id": 1,
      "title": "Memórias Póstumas de Brás Cubas",
      "author": "Machado de Assis",
      "published_year": 1881
     },
     {
      "id": 2,
      "title": "A Hora da Estrela",
      "author": "Clarice Lispector",
      "published_year": 1977
     }
    ]
   }
  },
  {
   "method": "GET",
   "path": "/books",
   "request": {
    "headers": {
     "Authorization": "Basic incorrect_credentials"
    }
   },
   "response": {
    "status": 401,
    "headers": {
     "Content-Type": "application/json"
    },
    "body": {
     "error": "Unauthorized",
     "message": "Credenciais de autenticação inválidas."
```

```
    }
   }
  },
  {
   "method": "GET",
   "path": "/books",
   "request": {
    "headers": {
     "Authorization": "Basic dXNlcm5hbWU6cGFzc3dvcmQ="
    }
   },
   "response": {
    "status": 403,
    "headers": {
     "Content-Type": "application/json"
    },
    "body": {
     "error": "Forbidden",
     "message": "Você não tem permissão para acessar este recurso."
    }
   }
  },
  {
   "method": "GET",
   "path": "/books/nonexistent",
   "request": {
    "headers": {
     "Authorization": "Basic dXNlcm5hbWU6cGFzc3dvcmQ="
    }
   },
   "response": {
    "status": 404,
    "headers": {
     "Content-Type": "application/json"
    },
    "body": {
     "error": "Not Found",
     "message": "O recurso solicitado não foi encontrado."
    }
   }
  },
  {
   "method": "GET",
   "path": "/books/error",
   "request": {
    "headers": {
     "Authorization": "Basic dXNlcm5hbWU6cGFzc3dvcmQ="
    }
   },
   "response": {
    "status": 500,
    "headers": {
     "Content-Type": "application/json"
    },
    "body": {
```

```
      "error": "Internal Server Error",
      "message": "Ocorreu um erro interno no servidor."
    }
   }
  }
 ]
}
```

Melhores práticas no Spring

- Utilização do formato de DSL do Spring Cloud Contract (Groovy);
- Adoção de uma abordagem orientada a testes criando stubs para testes automatizados.

GraphQL

GraphQL é tanto um estilo arquitetural como uma linguagem criada para interagir com serviços, solicitando dados através do frontend/cliente para o backend/servidor. Criada pelo Facebook justamente para solucionar os problemas encontrados pelas redes sociais ao tentar interagir com bancos de dados relacionais e com o modelo arquitetural REST.

Nessa abordagem existe apenas uma URL. No body da requisição é enviado uma query GraphQL para recuperação dos dados específicos solicitados pelo frontend. Evitando, portanto, transferência de dados que não serão utilizados e tornando a comunicação mais rápida e eficiente.

Alguns conceitos importantes no GraphQL

Conceito	Descrição
Query	Utilizada para a busca dos dados no backend.
Mutation	Utilizada para realizar alterações nos dados do backend.
Schema	Descreve os tipos de dados disponíveis na API. Basicamente o contrato da API GraphQL.
Type	Representa os tipos de dados disponíveis em uma API. Tipos podem ser criados pelo desenvolvedor (Ex.: Livro) ou podem ser usados os tipos básicos do GraphQL (Tipo Scalar: String, Int, Boolean, Float e ID)
Resolver	São funções que determinam como os dados serão recuperados.

Exemplo de DSL, Query e Response:

```
#DSL
type Query {
  livros: [Livro!]!
}

type Livro {
  id: ID!
  titulo: String!
  autor: String!
  anoPublicacao: Int!
}

schema {
  query: Query
}

#QUERY
query {
  livros {
    id
    titulo
    autor
    anoPublicacao
  }
}

#RESPONSE
{
  "data": {
    "livros": [
      {
        "id": "1",
        "titulo": "Memórias Póstumas de Brás Cubas",
        "autor": "Machado de Assis",
        "anoPublicacao": 1881
      },
      {
        "id": "2",
        "titulo": "A Hora da Estrela",
        "autor": "Clarice Lispector",
        "anoPublicacao": 1977
      }
    ]
  }
}
```

Melhores práticas no GraphQL

- Utilização dos recursos de validação e documentação disponíveis pelos tipos predefinidos do GraphQL;
- Utilização do versionamento na URL e/ou em campos específicos para evitar a quebra de contratos com clientes que utilizam versões anteriores;

- Utilização de limites na complexidade das consultas;
- Adição de paginação no retorno para consultas muito grandes;
- Utilização de cache para consultas frequentes.

Comparação

	OpenAPI	Spring Cloud Contract	GraphQL
Arquitetura	RESTful	RESTful	Alternativa ao REST
Formatos suportados	YAML/JSON	Groovy/YAML	SDL (Schema Definition Language)
Documentação Automática	Sim	Não	Não
Adoção	Amplamente adotado	Amplamente adotado para aplicações Java	Adoção crescente
Versionamento suportado	Sim	Sim	Não convencional – através de namespaces ou aliases
Testes de contrato	Suporte nativo	Usado em conjunto com ferramentas específicas	GraphQL Inspector
Facilidade de uso	Fácil aprendizado.	Requer conhecimento em Spring.	Requer conhecimento de estruturação de dados.
Banco de dados suportados	SQL, NoSQL	SQL, NoSQL	SQL, NoSQL
Cache	Fácil implementação	Fácil implementação	Difícil implementação

Cenários de uso recomendados

	OpenAPI	Spring Cloud Contract	GraphQL
Cenários recomendados	- Desenvolvimento de APIs RESTful; - Projetos que necessitam de documentação automatizada de APIs; - Projetos com requisitos bem definidos e fixos.	- Testes de contrato em APIs RESTful para garantir a consistência entre produtores e	- APIs com necessidade de consultas complexas e flexíveis;

		consumidores de serviços; - Projetos que necessitem de fácil geração de stubs para simulação de APIs durante o desenvolvimento de clientes consumidores; - Verificação de integração entre diferentes componentes de sistemas distribuídos.	- Aplicações com necessidade de evolução constante e consultas personalizadas. - Projetos com equipes de Frontend e Backend em times distribuídos sem muito contato.

Cenários de uso não recomendados

	OpenAPI	Spring Cloud Contract	GraphQL
Cenários não recomendados	- Aplicações que requerem flexibilidade extrema na estrutura da API, pois o OpenAPI é mais adequado para definir contratos rígidos; - Projetos que não necessitam de uma documentação detalhada da API ou que possuem requisitos de documentação mínimos.	- Projetos pequenos ou simples, pois pode adicionar complexidade desnecessária; - Necessário que o projeto esteja em ambiente Spring para funcionar completamente; - Equipes sem experiência no framework Spring.	- Times e/ou projetos pequenos; - GraphQL utiliza o método POST para as consultas, portanto para projetos que fazem uso dos benefícios de cache disponíveis nos navegadores na utilização de métodos GETs, o uso de GraphQL pode significar uma grande desvantagem.

Conclusão

Ao analisar os três frameworks – OpenAPI, GraphQL e Spring Cloud Contract - fica claro que cada um possui seus pontos fortes e fracos. A escolha adequada depende do contexto específico da aplicação.

Conforme detalhado na análise comparativa, o OpenAPI destaca-se por sua simplicidade, forte adoção na comunidade e foco em documentação. Por outro lado, GraphQL oferece uma abordagem mais robusta e escalável, adequada para projetos que demandam buscas mais específicas e mais performáticas. Enquanto isso, o Spring Cloud Contract, dentro do seu ecossistema específico de desenvolvimento, oferece uma opção também RESTful e com um forte foco em testes de contrato.

Portanto, ao considerar qual framework adotar, é crucial avaliar as necessidades e requisitos do projeto em questão. Cada contexto apresenta desafios e necessidades únicas, e a seleção do framework mais adequado pode aumentar a eficiência e a eficácia do desenvolvimento e testes da API.

Referências

GRAPHQL. Disponível em: https://graphql.org/. Acesso em: 01 mar. 2024.

MUNIZ, Antonio et al. JORNADA API NA PRÁTICA. 2019. Disponível em: https://painel.livros.leiamais.uol.com.br/meus-livros/132660/650582. Acesso em: 01 mar. 2024.

OPENAPI SPECIFICATION. Disponível em: https://www.openapis.org/. Acesso em: 01 mar. 2024.

RAMALHO, José Carlos. Engenharia Web. Dep. de Informática, Universidade do Minho. jcr@di.uminho.pt. Disponível em: https://www.di.uminho.pt/~jcr/LIVROS/EngWeb-OpenBook/. Acesso em: 01 mar. 2024.

RFC 7807: Problem Details for HTTP APIs. Disponível em: https://datatracker.ietf.org/doc/html/rfc7807. Acesso em: 01 mar. 2024.

SPRING CLOUD CONTRACT. Disponível em: https://spring.io/projects/spring-cloud-contract/. Acesso em: 01 mar. 2024.

YAML. YAML Version 1.2 Specification. Disponível em: https://yaml.org/spec/1.2.2/#chapter-1-introduction-to-yaml. Acesso em: 01 mar. 2024.