



LUIZ EDUARDO SANTAFOSTA
MATHEUS HENRIQUE MERLO SEMINATTI

**API REST E GRAPHQL: COMPARANDO AS DUAS TECNOLOGIAS
EM UMA APLICAÇÃO MARKETPLACE.**



LUIZ EDUARDO SANTAFOSTA
MATHEUS HENRIQUE MERLO SEMINATTI

**API REST E GRAPHQL: COMPARANDO AS DUAS TECNOLOGIAS
EM UMA APLICAÇÃO MARKETPLACE.**

Monografia apresentada à UNIP – Universidade Paulista de São José do Rio Preto – para a obtenção do grau de Cientista da Computação sob a orientação do Professor Especialista Thiago Luiz Parrillo Rizzo.



LUIZ EDUARDO SANTAFOSTA
MATHEUS HENRIQUE MERLO SEMINATTI

**API REST E GRAPHQL: COMPARANDO AS DUAS TECNOLOGIAS
EM UMA APLICAÇÃO MARKETPLACE.**

Monografia apresentada à UNIP – Universidade Paulista de São José do Rio Preto – para a obtenção do grau de Cientista da Computação.

Data Avaliação : ____/____/____

Situação: _____

Primeiro examinador
Nome:
Instituição:

Segundo examinador
Nome:
Instituição:

Prof. Orientador
Nome:
Instituição:

Dedicamos nosso trabalho xxxx.

AGRADECIMENTO

“A tecnologia move o mundo.” Steve Jobs

RESUMO

Palavras-chave:..

ABSTRACT

Keywords:

LISTA DE FIGURAS

Figura 1: Como funciona o Protocolo HTTP.	13
Figura 2: Exemplo de requisição HTTP.	14
Figura 3: Exemplo de resposta HTTP.	15
Figura 4: API.	18
Figura 5: Long Polling.....	20
Figura 6: Event Loop Node.js.....	21
Figura 7: Criando um servidor simples.	24
Figura 8: Exemplo de consulta.	25
Figura 9: Exemplo de modelo.	26
Figura 10: Exemplo de consulta com GraphQL.....	27

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO.....	12
1.1 Objetivo geral	12
1.2 Objetivos específicos	12
1.3 Justificativa	12
1.4 Metodologia	12
2 Pesquisa Bibliográfica	13
2.1 HTTP	13
2.1.1 Cliente.....	14
2.1.2 Servidor	14
2.1.3 Requests.....	14
2.1.4 Responses	15
2.1.4 Server Side Rendering e Client Side Rendering.....	16
2.2 API.....	18
2.2.1 API REST	19
2.3 Node.js	19
2.3.1 Assincronicidade	21
2.3.2 Gerenciador de pacotes (NPM)	23
2.4 GraphQL	24
2.5 Banco de dados	28
2.5.1 Tipos de banco de dados.....	28
2.5.2 Linguagem SQL	29
3 Análise e projeto do protótipo – UML	31
3.1 Diagrama de caso de uso	31
3.2 Diagrama de classe	31
3.3 Diagrama de atividade	31
3.4 Diagrama de blocos	31
4 TECNOLOGIAS EMPREGADAS	32
4.1 Visual Studio Code	32
4.2 Google Chrome.....	32
4.3 SQLite.....	33

5 IMPLEMENTAÇÃO	35
CONCLUSÃO (final).....	36
Referências	37

1 INTRODUÇÃO

1.1 Objetivo geral

1.2 Objetivos específicos

1.3 Justificativa

1.4 Metodologia

2 PESQUISA BIBLIOGRÁFICA

Para um melhor entendimento do assunto tratado no projeto, este capítulo abordará então, os diferentes tipos de tecnologias que iremos utilizar no desenvolvimento, a questão do protocolo HTTP, APIs, REST e GraphQL.

2.1 HTTP

Hypertext Transfer Protocol (HTTP) é um protocolo que permite se obter recursos na Web. É o principal responsável pela troca de dados cliente-servidor. Geralmente um navegador Web é o responsável pela reconstrução de um documento que contém diferentes sub-documentos contidos, como por exemplo texto, imagens, vídeos etc.

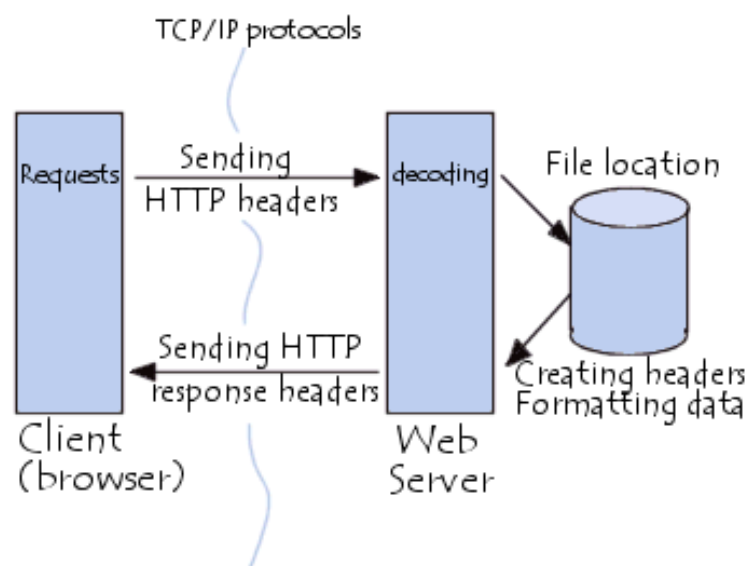


Figura 1: Como funciona o Protocolo HTTP.

Fonte: (CCM, 2017).

Cliente e servidor se comunicam com um certo tipo de mensagem. As mensagens enviadas pelo cliente, normalmente um navegador Web, são chamadas de requests, e as mensagens enviadas como resposta pelo servidor são chamadas de responses.

O HTTP foi projetado no início dos anos 90, e evoluiu ao longo do tempo. É um protocolo de camada de aplicação enviado sobre TCP. Ele é usado além de buscar hipertexto, para buscar imagens, vídeos e pode ser usado para buscar partes de documentos para atualizar páginas da Web.

2.1.1 Cliente

O cliente é qualquer ferramenta que age em nome do usuário. Normalmente é uma função realizada pelo navegador Web, mas também pode ser realizada por programas utilizados por desenvolvedores para realizar testes, por exemplo.

O cliente é sempre a entidade que inicia as requisições, o servidor nunca é responsável por tal.

Para mostrar uma página da Web, por exemplo, o cliente envia uma requisição para buscar o HTML da página, então é realizada uma análise desse arquivo para buscar as requisições adicionais que são necessárias para a apresentação da página, como arquivos CSS, JavaScript, imagens etc. Após isso o navegador interpreta os recursos recebidos e mostra a página da Web completa.

2.1.2 Servidor

Do outro lado, temos o servidor que devolverá o que for requisitado pelo usuário. O servidor é representado por apenas uma máquina, mas na realidade pode ser uma coleção de servidores dividindo a carga para gerar todo ou parte do que foi solicitado.

2.1.3 Requests

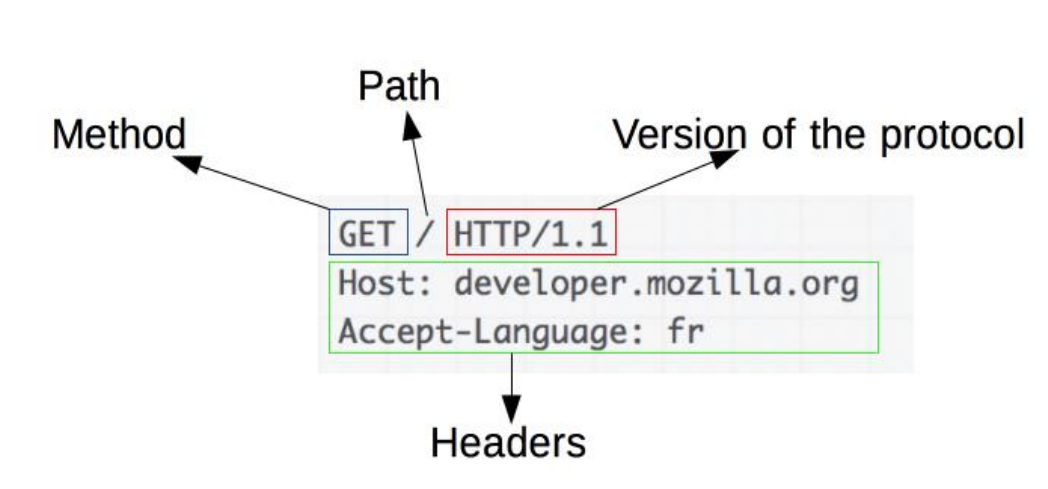


Figura 2: Exemplo de requisição HTTP.
Fonte: (Mozilla, 2020).

As requisições HTTP contam com:

- Um método HTTP, geralmente GET, POST, DELETE e PUT. Normalmente as mais utilizadas são GET para pegar um recurso do servidor e POST para publicar dados de um formulário HTML, por exemplo;
- O caminho (URL) do recurso a ser buscado;
- A versão do protocolo HTTP;
- Cabeçalhos opcionais que contém informações adicionais para os servidores;
- E um corpo de dados para alguns métodos como POST, similar ao corpo das respostas.

2.1.4 Responses

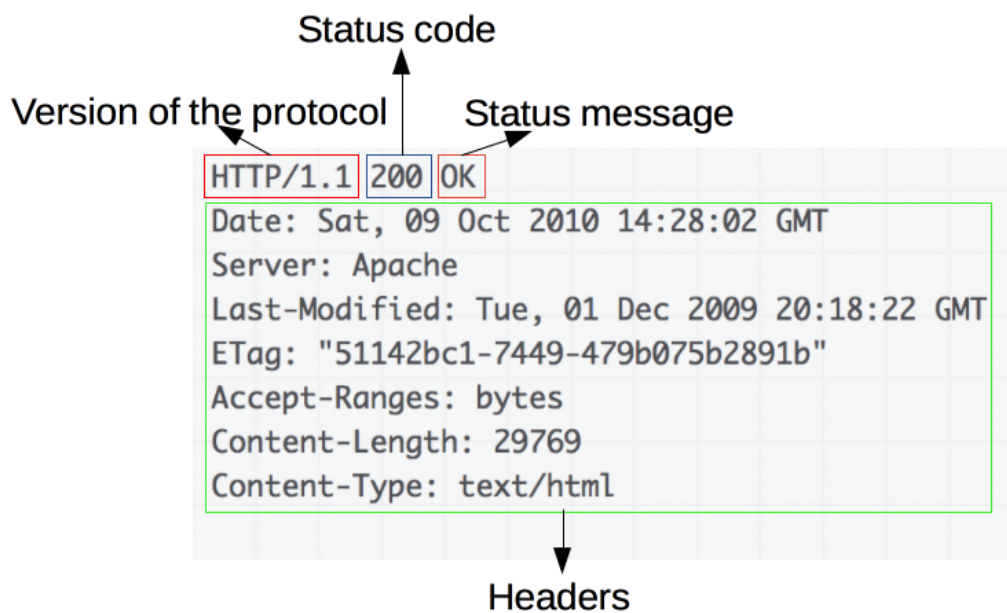


Figura 3: Exemplo de resposta HTTP.

Fonte: (Mozilla, 2020).

As respostas HTTP contam com:

- A versão do protocolo HTTP que elas seguem;
- Um código de status, indicando se foi bem sucedida ou não, e por quê;
- Uma pequena descrição informal sobre o código de status;
- Cabeçalhos HTTP, como os das requisições;
- Opcionalmente, um corpo com dados do recurso requisitado.

2.1.4 Server Side Rendering e Client Side Rendering

No desenvolvimento Web, existem 2 formas principais de entregar conteúdo ao usuário, que são chamadas de: Client Side Rendering (renderização do lado do cliente) e Server Side Rendering (renderização do lado do servidor).

2.1.4.1 Server Side Rendering

A renderização do lado do servidor é o método mais comum para exibir informações na tela. Ele funciona convertendo arquivos HTML no servidor em informações úteis para o navegador. Sempre que você visita um site, seu navegador faz uma solicitação ao servidor que contém o conteúdo do site. Depois que o processamento da solicitação é concluído, seu navegador retorna o HTML totalmente renderizado e o exibe na tela. Se você decidir visitar uma página diferente no site, seu navegador fará novamente uma solicitação para as novas informações. Isso ocorrerá toda vez que você visitar uma página da qual seu navegador não possui uma versão em cache. Não importa se a nova página tem apenas alguns itens que são diferentes da página atual, o navegador solicitará a nova página inteira e renderizará tudo do zero.

2.1.4.2 Client Side Rendering

Quando os desenvolvedores falam sobre renderização do lado do cliente, eles estão falando sobre renderizar conteúdo no navegador usando JavaScript. Portanto, em vez de obter todo o conteúdo do próprio documento HTML, você está obtendo um documento HTML básico com um arquivo JavaScript que renderizará o restante do site usando o navegador.

Essa é uma abordagem relativamente nova para renderizar sites e não se tornou realmente popular até que as bibliotecas JavaScript começaram a incorporá-la em seu estilo de desenvolvimento. Alguns exemplos notáveis são Vue.js, React.js e Angular.js.

Prós do lado do servidor:

- Os mecanismos de pesquisa podem rastrear o site para melhorar o SEO.
- O carregamento da página inicial é mais rápido.
- Ótimo para sites estáticos.

Contras do lado do servidor:

- Solicitações frequentes do servidor.
- Uma renderização geral lenta da página.
- Recarrega a página inteira.
- Interações de sites não ricos.

Prós do lado do cliente:

- Interações ricas em sites
- Renderização rápida do site após o carregamento inicial.
- Ótimo para aplicativos da web.
- Seleção robusta de bibliotecas JavaScript.

Contras do lado do cliente:

- SEO baixo se não for implementado corretamente.
- O carregamento inicial pode exigir mais tempo.
- Na maioria dos casos, requer uma biblioteca externa.

Pensando em uma aplicação MarketPlace, se faz necessário desenvolver uma aplicação que seja Client Side Rendering, pois os usuários precisam de muitas iterações, conteúdos que são direcionados à determinados tipos de usuários e muitos acessos simultâneos. Uma aplicação como esta proporciona todas essas ações com mais performance, gerando melhores resultados e uma melhor experiência para o usuário.

Mas de onde vêm as informações para o JavaScript renderizar?

Entram em cena então as APIs (Application Programming Interface).

2.2 API

Application Programming Interface (API), é um conjunto de padrões e rotinas de programação para acesso a plataformas baseadas na Web.

As APIs são criadas quando uma empresa quer disponibilizar seu serviço para que os desenvolvedores de outras aplicações associem com o produto que está sendo desenvolvido. Como por exemplo a Google, que disponibiliza a API do Google Maps, que é utilizada por muitos desenvolvedores em projetos Web e em projetos Mobile.

Nos dias atuais, praticamente todo software utiliza algum tipo de API, pois é mais fácil aprender a utilizar novas APIs do que criar uma funcionalidade do zero.

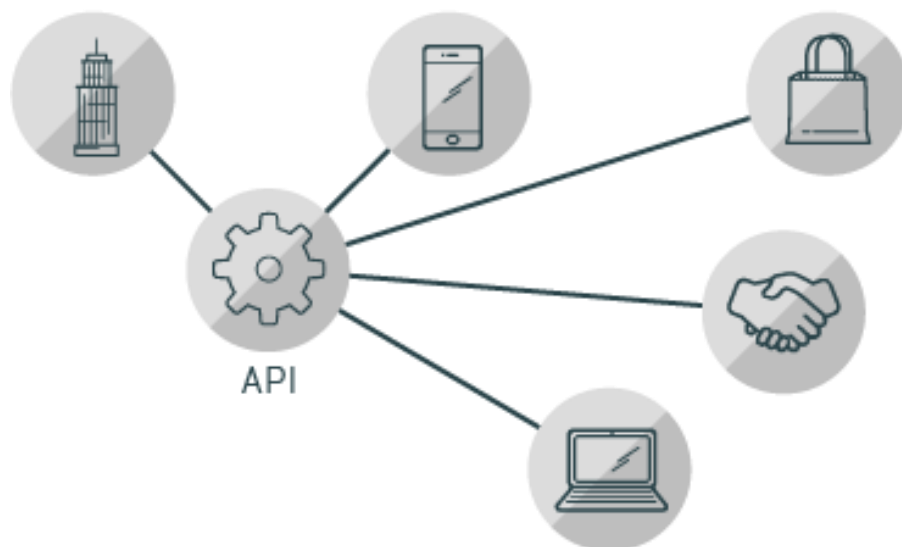


Figura 4: API.
Fonte: (Red Hat, 2020).

Existem, basicamente, três tipos de API:

- Privadas: são utilizadas de forma local entre aplicações de uma empresa;
- De parceiros: são utilizadas entre parceiros de negócios ou para permitir a integração entre softwares diferentes;
- Públicas: podem ser utilizadas livremente. Na maioria das vezes são disponibilizadas por empresas para que os desenvolvedores possam fazer a integração com outras aplicações.

2.2.1 API REST

Uma API REST (Representational State Transfer) utiliza requisições HTTP que são responsáveis pelas operações básicas necessárias para a manipulação dos dados.

Basicamente é um conjunto de regras utilizadas para que as requisições HTTP atendam as diretrizes definidas na arquitetura. As restrições são:

- Cliente-servidor: as aplicações existentes no servidor e no cliente devem ser separadas;
- Sem estado: as requisições são feitas de forma independente. Cada uma realiza apenas uma ação;
- Cache: a API deve utilizar cache para evitar chamadas recorrentes ao servidor;
- Interface uniforme: os recursos devem ser identificados, e sua manipulação deve ser feita por meio de representação com mensagens auto descritivas e utilizar links para navegar pela aplicação.

Quando se fala em API REST, significa utilizar uma API para acessar aplicações backend, de maneira que essa comunicação seja feita com os padrões definidos pela arquitetura REST.

2.3 Node.js

No ano de 2009, Ryan Dahl estava trabalhando em um projeto quando se deparou com um grande problema, descobrir quanto tempo faltava para uma ação de upload ser concluída.

Até então, um método comum utilizado consistia em enviar requisições AJAX a cada determinado intervalo de tempo, perguntando ao servidor quanto do arquivo já tinha sido enviado. Porém, esse método era custoso ao servidor e ao cliente, pois eram feitas muitas requisições até o término do upload. Uma alternativa era o Long Polling, que consistia em fazer uma pergunta ao servidor, porém o servidor segurava a resposta até algum evento acontecer.

O processo de Long Polling é mostrado na figura abaixo.

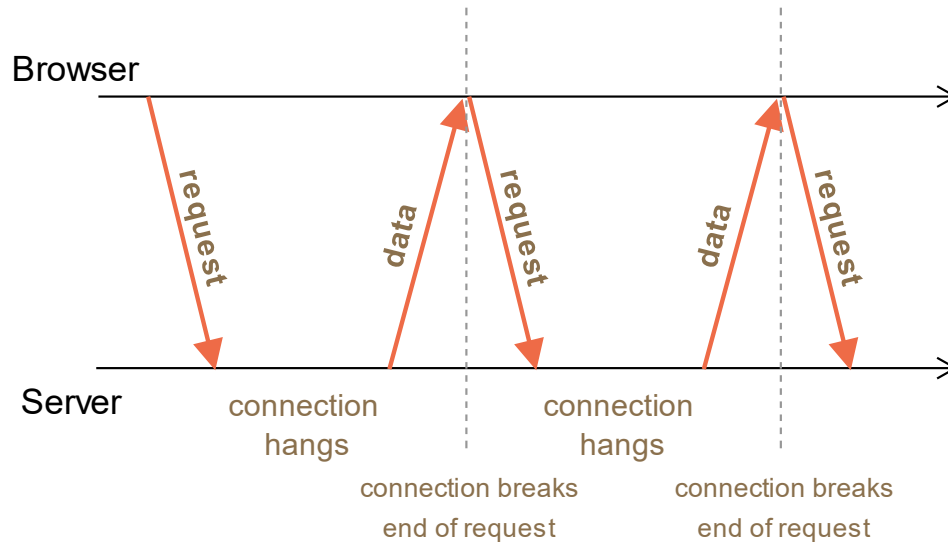


Figura 5: Long Polling.

Fonte: (JAVASCRIPT.INFO, 2020).

Dahl queria implementar um servidor que englobasse esse conceito. Um ano antes, em 2008, a empresa Google havia lançado o navegador Google Chrome, que contava com o V8, um poderoso interpretador JavaScript desenvolvido por ela.

Aproveitando as novas ferramentas disponíveis, o grande crescimento no mercado, e por não necessitar de bibliotecas ou meios para se implementar assincronismo, escolheu JavaScript como linguagem. O que também evitou a necessidade de criação e gerenciamento de threads, perfeito para manter a simplicidade que ele desejava.

JavaScript era simples e leve, permitia ao servidor liberar o uso da CPU pois tinha suporte a operações não-bloqueantes, executava em uma única thread, diminuía o uso da CPU e memória, e era orientada a eventos, o que era essencial para implementar o Long Polling. Com isso Dahl começou a implementar seu projeto, que foi crescendo, ganhando padrões e hoje é o servidor Web completo que todos conhecem, o Node.js.

2.3.1 Assincronicidade

O maior benefício do Node.js, é a facilidade de se implementar assincronismo.

Graças a Libuv, uma biblioteca open source criada originalmente para o Node.js que implementa um Event Loop com todos os recursos necessários, o assincronismo não precisa ser gerenciado pelo desenvolvedor, a própria biblioteca gerencia as threads e os processos para que tudo corra perfeitamente.

O Node.js oferece 3 maneiras de se implementar assincronismo: Callbacks, Promises e utilizando Async/Await (a partir da ES2017).

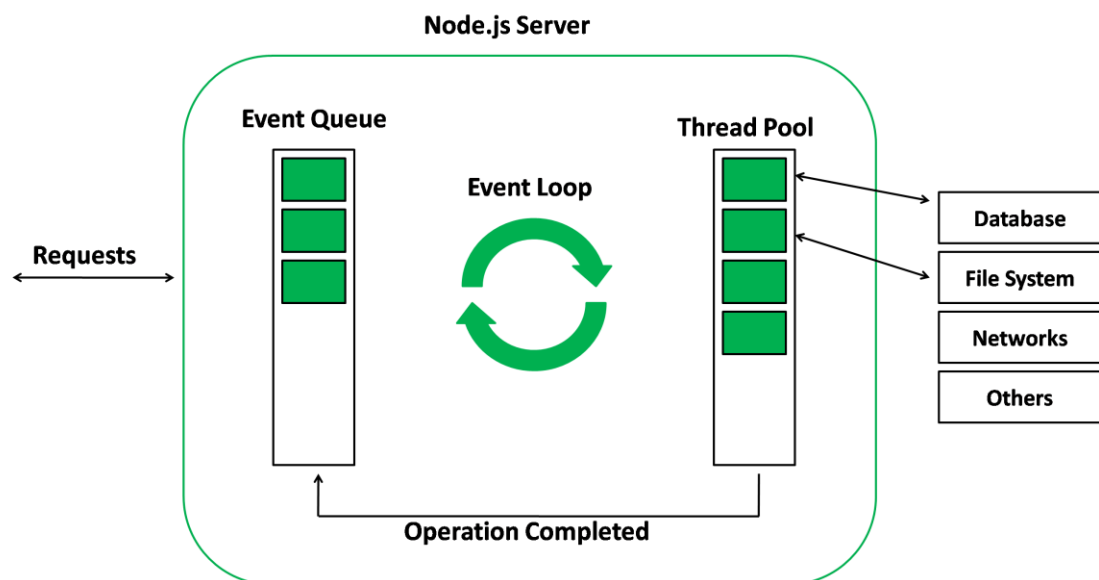


Figura 6: Event Loop Node.js.
Fonte: (GeeksforGeeks, 2020).

2.3.1.1 Callbacks

Callbacks são funções passadas como parâmetro para outra função. A função que o recebe pode executar a callback quando ela achar necessário, normalmente depois de um intervalo de tempo.

Algumas funções internas utilizam callback, como setInterval e setTimeout.

A função `setTimeout` é o melhor exemplo possível para se entender esse conceito. Caso essa função fosse síncrona, a execução do restante do código ficaria parada por um grande tempo, dependendo do tempo limite que for definido. Mas como ela é assíncrona, recebe uma função `callback` que será executada após o tempo limite terminar, fazendo assim que o código continue sendo executado sem nenhum bloqueio.

Outro exemplo também é o método `.addEventListener()`, onde é passado como primeiro parâmetro o tipo de evento que será ouvido, como o “click” por exemplo, e o segundo parâmetro é um `callback` que será executada somente quando o evento for acionado.

No entanto, utilizar muitos `call-backs` aninhados pode deixar tudo muito complicado para o desenvolvedor e acabar resultando em um “callback hell”. Como uma opção de substituto aos `callbacks`, foi introduzido o conceito de `promise`.

2.3.1.2 Promises

`Promise` nada mais é do que uma sintaxe mais agradável de `callback`, foi desenvolvida para ajudar o desenvolvedor a evitar erros como o “callback hell”.

Uma `promise` pode ter 3 estados:

- **Pendente:** é o primeiro estado, imediatamente quando a `promise` é criada, e permanece nesse estado até avançar para resolvida ou rejeitada.
- **Resolvida:** é quando a operação é concluída com sucesso. Normalmente uma `promise` resolvida retorna algum tipo de dado com ela.
- **Rejeitada:** é quando a operação falha. Normalmente uma `promise` rejeitada retorna algum tipo de erro com ela.

As `promises` vem integradas com um método `.then()`, que recebe uma função de retorno como parâmetro. Essa função recebe todos os dados e só será executado quando a `promise` estiver resolvida.

No caso da `promise` ser rejeitada, há o método `.catch()`, que será executado sempre que houver rejeição.

Os métodos `.then()` e `.catch()` dão à `promise` a aparência de um bloco de código `try/catch`.

2.3.1.3 Async/Await

Diferente das funções normais, funções declaradas com a palavra-chave `async` permitem se utilizar outra palavra-chave dentro do escopo da função: `await`.

Quando essa função é chamada, ela retorna uma `promise` pendente. Se a função retornar um valor, ela resolve a `promise` com esse valor, se gerar um erro, ela rejeita a `promise` com o erro gerado.

A palavra-chave `await` é utilizada para tornar síncronas as funções que normalmente são assíncronas em síncronas. Normalmente é utilizada quando há uma função que retorna uma `promise` e é necessário pausar a execução da função assíncrona até que a chamada seja concluída.

Essas palavras-chaves podem ajudar muito o desenvolvedor dependendo de suas necessidades.

2.3.2 Gerenciador de pacotes (NPM)

Node Package Manager (NPM) é o gerenciador de pacotes nativo do Node.js. Consiste em três componentes distintos: o site, a interface de linha de comando e o registro.

O site é usado para descobrir pacotes, configurar perfis e gerenciar a experiência com o NPM. A interface de linha de comando é executada no terminal do sistema e é usada para o desenvolvedor interagir com o NPM. O registro é um banco de dados de software JavaScript e as metainformações que o cercam.

2.3.3 Servidor integrado

Diferente de outras linguagens de programação para Web onde temos que configurar um servidor externo, o Node.js já traz um embutido, que consegue dar suporte até aplicações de médio porte sem grandes dificuldades.

Com poucas linhas de código conseguimos iniciar um servidor na porta que desejamos.


```
var http = require('http');
...
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Olá Mundo\n');
}).listen(3000, '127.0.0.1');
```

Figura 7: Criando um servidor simples.

Fonte: Elaborado pelos autores.

2.4 GraphQL

GraphQL é uma linguagem de consulta de dados e especificação desenvolvida internamente pelo Facebook em 2012 antes de ser disponibilizada publicamente em 2015. Ela oferece uma alternativa às arquiteturas baseadas em REST com o objetivo de aumentar a produtividade do desenvolvedor e minimizar a quantidade de dados transferidos.

Segundo a própria documentação, GraphQL é uma linguagem de consulta para APIs e um tempo de execução para atender a essas consultas com seus dados existentes. GraphQL fornece uma descrição completa e compreensível dos dados em sua API, dá aos clientes o poder de pedir exatamente o que precisam e nada mais, torna mais fácil evoluir APIs ao longo do tempo e permite ferramentas poderosas de desenvolvedor.

Como assim os clientes podem pedir exatamente o que precisam? Isso mesmo!

Veremos um exemplo a seguir:

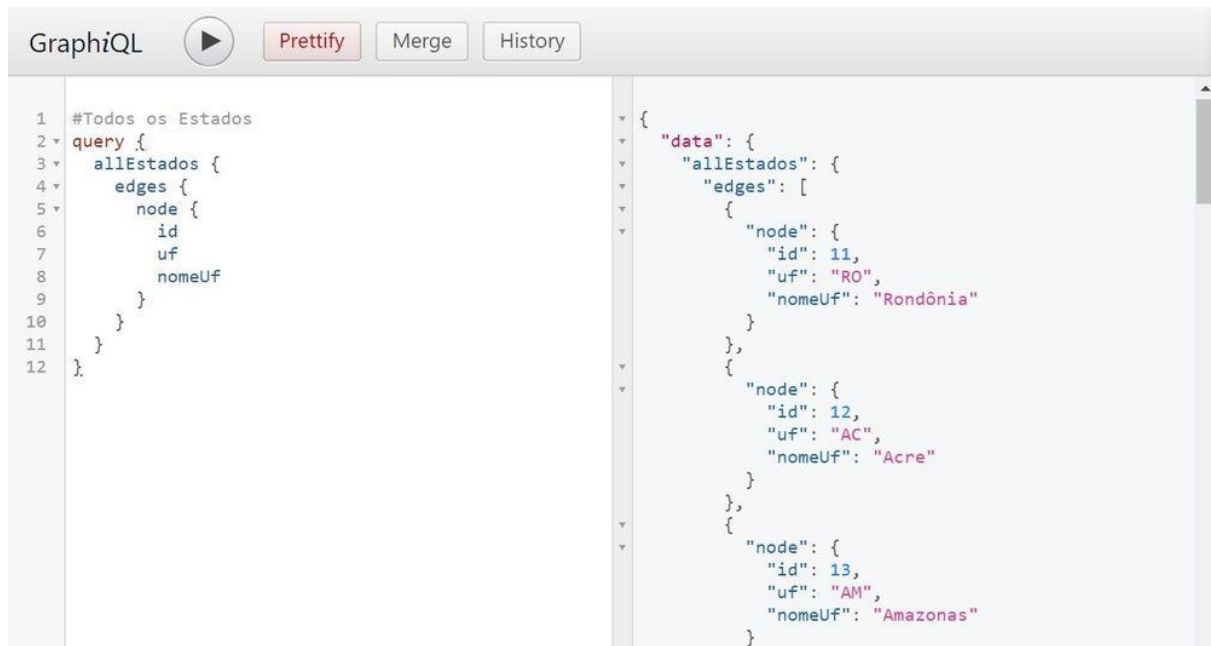


Figura 8: Exemplo de consulta.

Fonte: Elaborado pelos autores.

Do lado esquerdo vemos um JSON (JavaScript Object Notation) que é enviado no corpo da requisição, e do lado direito temos a resposta dessa requisição. Na requisição, é solicitado todos os estados e os seguintes dados de cada estado: id, uf, nomeUf. Na resposta, temos exatamente isso: um JSON contendo todos os estados e as informações requisitadas de cada estado. Se quiséssemos apenas o id e a uf de cada estado, precisaríamos apenas desconsiderar do exemplo anterior o nomeUf, e ele deixaria de aparecer em nossa resposta.

Isso permite uma liberdade maior para os clientes, e segundo a própria documentação, os aplicativos que usam GraphQL são rápidos e estáveis porque controlam os dados que obtêm, não o servidor.

Estas requisições feitas com GraphQL são todas do tipo POST, e o que vai mudar de uma para a outra é apenas o corpo da requisição, ou seja, nossa query.

Esta query precisa seguir um modelo, que é chamado de Type.

Abaixo temos dois exemplos de definição de Type:

```
type User {
  uuid: ID!
  name: String!
  cpf: String
  zipcode: String
  address: Address
}

type Address {
  street_address: [String!]!
  city: String!
  state: String!
}
```

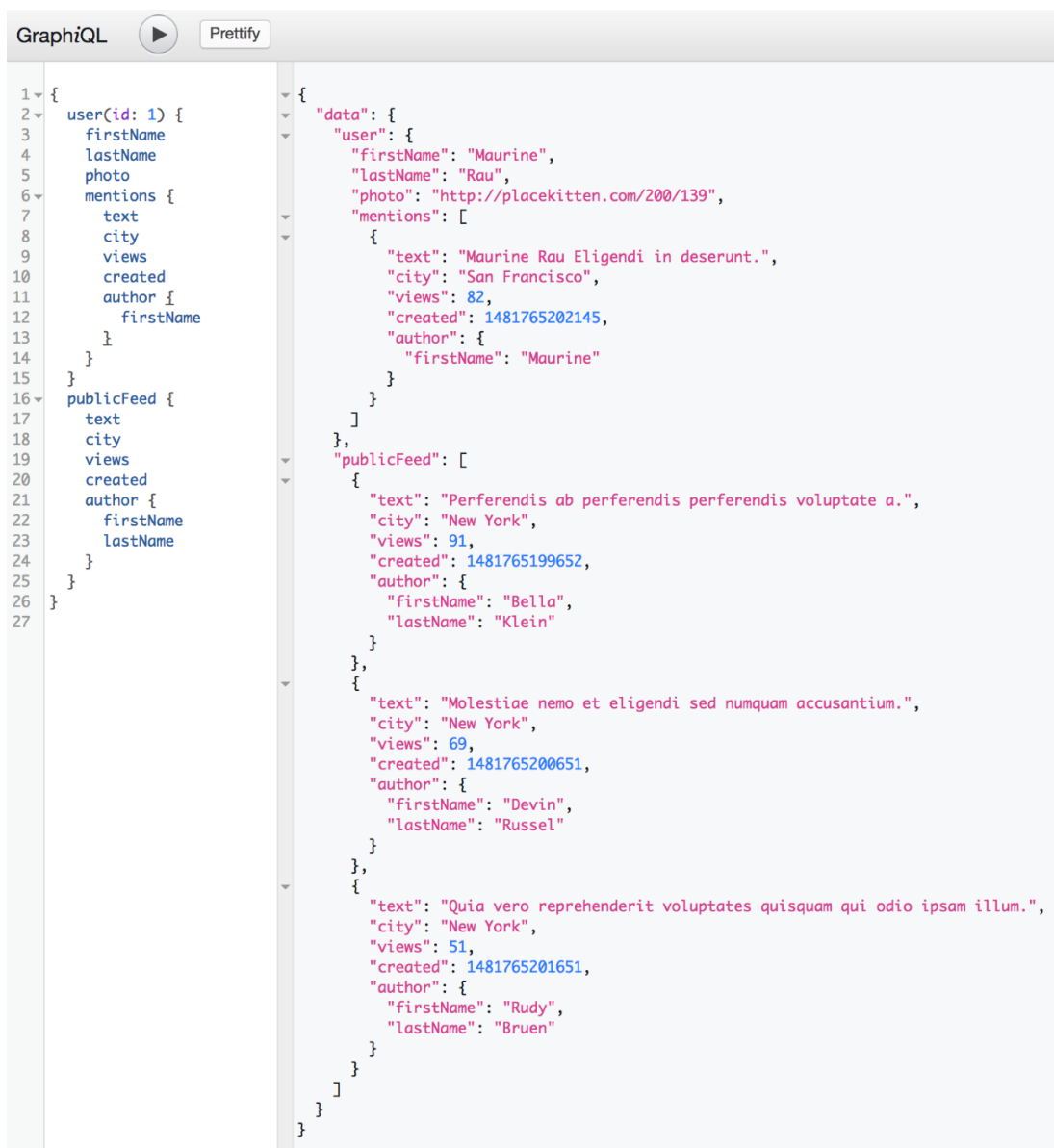
Figura 9: Exemplo de modelo.

Fonte: Elaborado pelos autores.

Como vemos, temos dois Types, User e Address. Em User, podemos notar que existe um relacionamento com Address, pois foi definida uma propriedade com o tipo Address. Para montarmos uma query solicitando todas as informações de um User com todas as informações de seu Address, podemos fazer da seguinte forma:

```
query{
  user{
    uuid,
    name,
    cpf,
    zipcode,
    address{
      street_address,
      city,
      state
    }
  }
}
```

Ainda segundo a documentação, as consultas GraphQL acessam não apenas as propriedades de um recurso, mas também seguem suavemente as referências entre eles. Enquanto as APIs REST típicas requerem o carregamento de vários URLs, as APIs GraphQL obtêm todos os dados de que seu aplicativo precisa em uma única solicitação. Abaixo temos um exemplo do que em REST seriam necessárias duas requisições para termos como resposta os mesmos resultados:



The image shows a screenshot of the GraphQL Playground interface. On the left, there is a query editor with line numbers 1 through 27. The query is as follows:

```
1 {
2   user(id: 1) {
3     firstName
4     lastName
5     photo
6     mentions {
7       text
8       city
9       views
10      created
11      author {
12        firstName
13      }
14    }
15  }
16  publicFeed {
17    text
18    city
19    views
20    created
21    author {
22      firstName
23      lastName
24    }
25  }
26 }
27
```

On the right, there is a JSON response editor showing the result of the query. The response is a JSON object with a "data" field containing the results for the user and the public feed.

```
{
  "data": {
    "user": {
      "firstName": "Maurine",
      "lastName": "Rau",
      "photo": "http://placekitten.com/200/139",
      "mentions": [
        {
          "text": "Maurine Rau Eligendi in deserunt.",
          "city": "San Francisco",
          "views": 82,
          "created": 1481765202145,
          "author": {
            "firstName": "Maurine"
          }
        }
      ]
    },
    "publicFeed": [
      {
        "text": "Perferendis ab perferendis perferendis voluptate a.",
        "city": "New York",
        "views": 91,
        "created": 1481765199652,
        "author": {
          "firstName": "Bella",
          "lastName": "Klein"
        }
      },
      {
        "text": "Molestiae nemo et eligendi sed numquam accusantium.",
        "city": "New York",
        "views": 69,
        "created": 1481765200651,
        "author": {
          "firstName": "Devin",
          "lastName": "Russel"
        }
      },
      {
        "text": "Quia vero reprehenderit voluptates quisquam qui odio ipsam illum.",
        "city": "New York",
        "views": 51,
        "created": 1481765201651,
        "author": {
          "firstName": "Rudy",
          "lastName": "Bruen"
        }
      }
    ]
  }
}
```

Figura 10: Exemplo de consulta com GraphQL.

Fonte: Elaborado pelos autores.

Isso torna a aplicação mais rápida, pois conseguimos fazer mais do que uma consulta em apenas uma requisição.

2.5 Banco de dados

Boa parte dos motivos para se ter um banco de dados está ligado às necessidades e exigências dos consumidores. Eles desejam ter experiências relevantes e que vão além de comprar um item e pagá-lo no caixa.

A personalização do atendimento é muito mais abrangente que isso e pode começar, até mesmo, na seleção de produtos do varejo. Outros motivos que fortalecem sua estratégia são:

- permitir o acesso facilitado dos dados da empresa: seja por não ter uma equipe específica para gestão e análise, seja por ter diversas áreas e filiais que podem usufruir da ferramenta. Um banco de dados garante o compartilhamento de seu acesso e utilização simplificada;
- criar análises e comparativos entre seus dados: o que aumenta consideravelmente seu nível de acerto são os dados estruturados dos bancos, que podem criar relatórios e informações estratégicas para as tomadas de decisão,
- facilitar atualizações e melhorias das informações: com um sistema para gerenciamento de dados, a curadoria dos registros é facilitada. O gestor identifica aqueles que podem ser relevantes para o negócio e outros que podem ser descartados, como antigos parâmetros ou leis que foram modificadas.

A importância do banco de dados é tão indiscutível quanto seu crescimento exponencial. A cada interação entre empresa e cliente, são gerados milhares de registros. Quando organizados, eles podem trazer *insights* fundamentais para a gestão do varejo. Esse crescimento é tão impactante, que a análise de todos esses dados pode ser considerada extremamente complexa.

2.5.1 Tipos de banco de dados

Os bancos de dados podem ser aplicados em diversas áreas de mercado. Além disso, têm algumas tipificações. Para complementar essa análise, é importante entender que eles podem ser do tipo relacional ou não.

2.5.1.1 Relacional

Nesse caso, os registros são tabelados e estruturados em colunas e linhas que podem se relacionar. Tipo muito comum em sistemas de ERP e CRM. Com eles, é possível criar relações valiosas para a tomada de decisão.

Um exemplo seria comparar informações sobre o volume de vendas na loja física e o período de uma campanha promocional na TV. O impacto nos resultados, positivos ou negativos, pode conduzir gestores a *insights* importantes, como o quanto seu público é sensível àquela mídia.

2.5.1.2 Não relacional

Os dados não relacionais não são tão versáteis como os primeiros para análises comparativas, mas são igualmente importantes. Os seus formatos são menos convencionais, como as imagens publicadas nas redes sociais.

Apesar de terem um formato não relacional, também podem ser avaliados. Por exemplo, o número de fotografias tiradas e publicadas nas redes sociais como um sinal de aprovação da marca.

Para o desenvolvimento do projeto foi escolhido o tipo de banco de dados relacional, pois se faz necessários relações entre os dados.

2.5.2 Linguagem SQL

A linguagem SQL implementa os conceitos definidos no modelo relacional, um modelo largamente aceito e recomendado.

A utilização deste standard internacional reduz as incompatibilidades entre os sistemas e evita que se opte por arquiteturas proprietárias que implicam maiores custos de desenvolvimento e maior esforço financeiro e humano por parte dos intervenientes. Com a linguagem SQL é possível:

- Criar, alterar e remover todas as componentes de uma base de dados, como tabelas, campos, views, índices etc.;
- Inserir, alterar e apagar dados;

- Interrogar a base de dados, obtendo como resposta o conjunto de registos que obedece às condições indicadas;
- Controlar o acesso dos utilizadores à base de dados e as operações a que cada um deles pode ter acesso;
- Obter a garantia da consistência e integridade dos dados. De notar que a linguagem SQL realiza o conjunto das tarefas enunciadas através de uma linguagem simples, de fácil aprendizagem e implementação.

3 ANÁLISE E PROJETO DO PROTÓTIPO – UML

3.1 Diagrama de caso de uso

A x mostra o diagrama que compõe os atores do sistema.

Figura : Diagrama de casos de uso.
Fonte: elaborado pelos autores

3.2 Diagrama de classe

A figura x mostra o diagrama contendo as entidades do sistema.

Figura: Diagrama de classes.
Fonte: elaborado pelos autores.

3.3 Diagrama de atividade

A figura x apresenta o diagrama de atividades realizadas pelo sistema.

Figura: Diagrama de atividades.
Fonte: elaborado pelos autores.

3.4 Diagrama de blocos

A x apresenta através do diagrama de blocos, a arquitetura do sistema.

Figura: Diagrama de bloco.
Fonte: elaborado pelos autores

4 TECNOLOGIAS EMPREGADAS

Para o desenvolvimento do projeto algumas ferramentas foram necessárias e, por conta disso, este capítulo apresenta as principais ferramentas envolvidas na construção do protótipo.

4.1 Visual Studio Code

Lançado em 2015 pela Microsoft, o Visual Studio Code, é um editor de código destinado ao desenvolvimento de aplicações Web. É uma ferramenta leve e multiplataforma que atende a uma enorme gama de projetos, como ASP.NET e Node.js.

Além de gratuito, trata-se de um editor open source, com seu código disponibilizado no GitHub, permitindo contribuições com seu desenvolvimento e a criação de novas funcionalidades e extensões.

Ao contrário do que o nome indica, o Visual Studio Code não é uma versão do Visual Studio, ele é um editor de código semelhante ao Atom, Sublime Text e tantos outros disponíveis para utilização.

Ele conta com a opção de personaliza-lo de acordo com o gosto de quem vai utilizá-lo, adicionar extensões para facilitar o desenvolvimento e deixar o ambiente mais agradável para o desenvolvedor.

4.2 Google Chrome

O Google Chrome é um dos navegadores mais populares atualmente. Multiplataforma e open source, ele está disponível tanto para desktops de diferentes sistemas operacionais, como para a plataforma mobile.

Como o próprio nome já diz, o produto desenvolvido pela Google teve sua primeira versão gratuita liberada em 2008, e recebe atualizações constantemente.

Por ser open source o Chrome conta com diferentes versões, como a Portable, que não exige a instalação na máquina, pois pode ser executada diretamente de um pen drive por exemplo.

O navegador possui versões de 32 e 64 bits. Segundo a Google, é recomendada a instalação da versão em 64 bits, pois traz melhor desempenho, estabilidade e

segurança, além da melhor renderização de páginas que pode ser até duas vezes mais rápida do que na versão de 32 bits.

O Chrome também conta com uma loja virtual chamada Chrome Web Store, que reúne em um só lugar os add-ons e plugins criados para o navegador. Ela se assemelha com a App Store e a Google Play, porém as ferramentas ali encontradas são compatíveis somente com o Chrome.

4.3 SQLite

Dentre tantos mecanismos de banco de dados que existem, como Microsoft SQL Server, Oracle, MySQL, optamos por utilizar o SQLite.

Segundo o próprio site do SQLite, ele trata-se de uma biblioteca de linguagem C que implementa um mecanismo de banco de dados SQL pequeno, rápido, independente, de alta confiabilidade e de recursos completos. SQLite é o mecanismo de banco de dados mais usado no mundo. Está integrado em todos os telefones celulares e na maioria dos computadores, e vem integrado em inúmeros outros aplicativos que as pessoas usam todos os dias.

O formato de arquivo SQLite é estável, multiplataforma e compatível com versões anteriores e os desenvolvedores se comprometem a mantê-lo assim até pelo menos o ano 2050. Arquivos de banco de dados SQLite são comumente usados como contêineres para transferir conteúdo rico entre sistemas e como um formato de arquivamento de longo prazo para dados. Existem mais de 1 trilhão de bancos de dados SQLite em uso ativo.

Ao contrário da maioria dos outros bancos de dados SQL, o SQLite não tem um processo de servidor separado. O SQLite lê e grava diretamente em arquivos de disco comuns. Um banco de dados SQL completo com várias tabelas, índices, gatilhos e visualizações está contido em um único arquivo de disco. O formato do arquivo de banco de dados é multiplataforma - você pode copiar livremente um banco de dados entre sistemas de 32 e 64 bits ou entre as arquiteturas big-endian e little-endian. Esses recursos tornam o SQLite uma escolha popular como formato de arquivo de aplicativo. Os arquivos de banco de dados SQLite são um formato de armazenamento recomendado pela Biblioteca do Congresso dos EUA.

A base de código SQLite é suportada por uma equipe internacional de desenvolvedores que trabalham em SQLite em tempo integral. O código-fonte do SQLite é de

domínio público e pode ser usado gratuitamente por todos para qualquer propósito, mas também está disponível para suporte profissional.

5 IMPLEMENTAÇÃO

CONCLUSÃO (FINAL)

REFERÊNCIAS

MDN WEB DOCS. **Uma visão geral do HTTP**. Disponível em:

<<https://www.developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview>>. Acesso em: 22 de ago. 2020.

CCM. **O protocolo HTTP**. Disponível em: <https://img-21.ccm2.net/JY7b3wt_LUqWROfMf6OqiRRu444=/b712b72d2699457190ce05622277ff4a/cm-encyclopedia/internet-images-comm.gif>.

Acesso em: 22 de ago. 2020.

MDN WEB DOCS. **Exemplo de uma requisição HTTP**. Disponível em:

<https://media.prod.mdn.mozit.cloud/attachments/2016/08/09/13687/5d4c4719f4099d5342a5093bdf4a8843/HTTP_Request.png>. Acesso em: 22 de ago. 2020.

MDN WEB DOCS. **Exemplo de uma resposta HTTP**. Disponível em:

<https://media.prod.mdn.mozit.cloud/attachments/2016/08/09/13691/58390536967466a1a59ba98d06f43433/HTTP_Response.png>. Acesso em: 22 de ago. 2020.

FREECODECAMP. **Client-side vs. server-side rendering: why it's not all black and white**. Disponível em: <<https://www.freecodecamp.org/news/what-exactly-is-client-side-rendering-and-hows-it-different-from-server-side-rendering-bd5c786b340d/>>.

Acesso em: 24 de ago. 2020.

CANALTECH. **O que é API?** Disponível em:

<<https://canaltech.com.br/software/o-que-e-api/>>. Acesso em: 25 de ago. 2020.

RED HAT. **O que são APIs?** Disponível em:

<https://www.redhat.com/cms/managed-files/what-are-apis-370x226_0.png>. Acesso em: 25 de ago. 2020.

REST API TUTORIAL. **What is REST**. Disponível em:

<<https://restfulapi.net/>>. Acesso em: 25 de ago. 2020.

ROCK CONTENT. **Entenda o que é Rest API e a importância dele para o site da sua empresa**. Disponível em: <<https://rockcontent.com/br/blog/rest-api/>>. Acesso em: 25 de ago. 2020.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. p. 76-105. Dissertation (Doctoral Degree in Philosophy in Information and Computer Science) – University of California, Irvine. 2000.

NODEJS. **About Node.js**. Disponível em: <<https://nodejs.org/en/about/>>. Acesso em: 25 de ago. 2020.

WANDSCHNEIDER, Marc. **Learning Node.js: A Hands-On Guide to Building Web Applications in JavaScript**. 1. ed, Pearson Education, 2013.

GITCONNECTED. Polling in JavaScript. Disponível em:
<<https://levelup.gitconnected.com/polling-in-javascript-ab2d6378705a>>. Acesso em: 29 de ago. 2020.

JAVASCRIPT.INFO. Long Polling. Disponível em:
<<https://javascript.info/long-polling>>. Acesso em: 29 de ago. 2020.

V SCHOOL. Understanding Asynchronicity. Disponível em:
<<https://coursework.vschool.io/asynchronicity/>>. Acesso em: 30 de ago. 2020.

IMASTERS. Node.js: o que é esse Event Loop, afinal? Disponível em:
<<https://imasters.com.br/front-end/node-js-o-que-e-esse-event-loop-afinal>>. Acesso em: 30 de ago. 2020.

GEEKSFORGEEKS. Working of the Event Loop. Disponível em:
<<https://media.geeksforgeeks.org/wp-content/uploads/20200224050909/nodejs2.png>>. Acesso em: 30 de ago. 2020.

NODEJS. What is npm? Disponível em:
<<https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/>>. Acesso em: 30 de ago. 2020.

NPM DOCUMENTATION. About npm. Disponível em:
<<https://docs.npmjs.com/about-npm/>>. Acesso em: 30 de ago. 2020.

GRAPHQL. A query language for your API. Disponível em:
<<https://graphql.org/>>. Acesso em: 01 de set. 2020.

GRAPHQL FOUNDATION. What is GraphQL? Disponível em:
<<https://foundation.graphql.org/>>. Acesso em: 01 de set. 2020.

TOTVS. A importância de um banco de dados corporativo. Disponível em:
<<https://www.totvs.com/blog/negocios/banco-de-dados/>>. Acesso em: 01 de set. 2020.

MIRANDA, William. O que é um Banco de Dados e Qual a sua importância para uma Empresa. LinkedIn. Disponível em: <<https://www.linkedin.com/pulse/o-que-%C3%A9-um-banco-de-dados-e-qual-sua-import%C3%A2ncia-para-william-miranda/>>. Acesso em: 01 de set. 2020.

DAMAS, Luís. SQL. 14. ed, FCA, 2013.

DEVMEDIA. Introdução ao Visual Studio Code. Disponível em:
<<https://www.devmedia.com.br/introducao-ao-visual-studio-code/34418>>. Acesso em: 07 de set. 2020.

TECHTUDO. Chrome: o navegador completo e gratuito do Google. Disponível em:
<<https://www.techtudo.com.br/tudo-sobre/google-chrome.html>>. Acesso em: 07 de set. 2020.

SQLITE. What is SQLite. Disponível em:
<<https://www.sqlite.org/index.html>>. Acesso em: 07 de set. 2020.