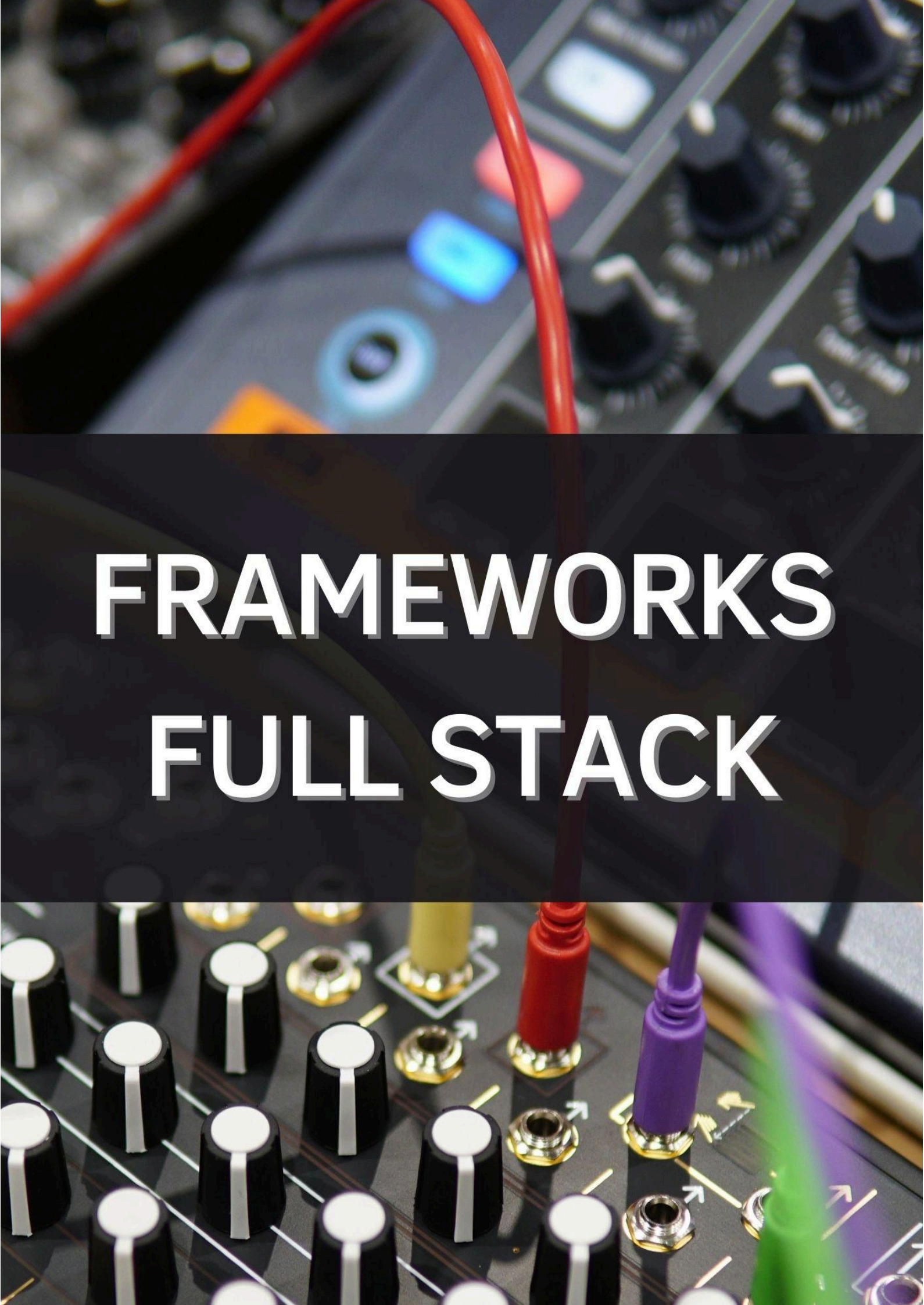


FRAMEWORKS FULL STACK

Texto base

3

Implementação das bases das aplicações a serem construídas

Prof. André Nascimento Maia
Prof. Caio Nascimento Maia

Resumo

Será necessário criar as aplicações base para o aprendizado da criação de uma aplicação utilizando a abordagem full stack framework. Neste texto, você aprenderá, passo-a-passo, como criar uma web application base utilizando o framework Next.js e um web API base utilizando o framework Flask, além de entender os fundamentos básicos sobre as técnicas e tecnologias necessárias para esta atividade.

3.1. Introdução

Nesta parte, o objetivo é criar as bases das duas aplicações que utilizaremos para a construção do produto Impacta Commerce, o e-commerce fictício que será construído durante todo o curso. A primeira aplicação, será a aplicação da camada de *front-end*, uma *web application* utilizando o *framework* Next.js que permitirá aos usuários visualizarem produtos, adicioná-los ao carrinho e executar um pagamento fictício. A segunda aplicação, será uma aplicação da camada de *back-end*, uma API REST construída com o *(micro)framework* Flask, que receberá todos os comandos da *web application* e os processará de acordo com as regras de negócio que forem estipuladas para o *e-commerce*.

3.2. História do Desenvolvimento Web

No início do desenvolvimento *web,* nos anos 1990, o principal objetivo era o compartilhamento de documentos via rede de computadores, sendo esses documentos considerados estáticos, pois os seus usuários finais não conseguiam editar o conteúdo, mas somente lê-los. Esta era da *web* é chamada de *Web* 1.0.

Ao longo dos anos, o aumento acentuado do uso da *web* por principalmente grandes portais como Yahoo! e UOL, fez com que a necessidade por interatividade aumentasse, o que forçou o avanço para a próxima era da *web*, a era chamada *Web* 2.0.

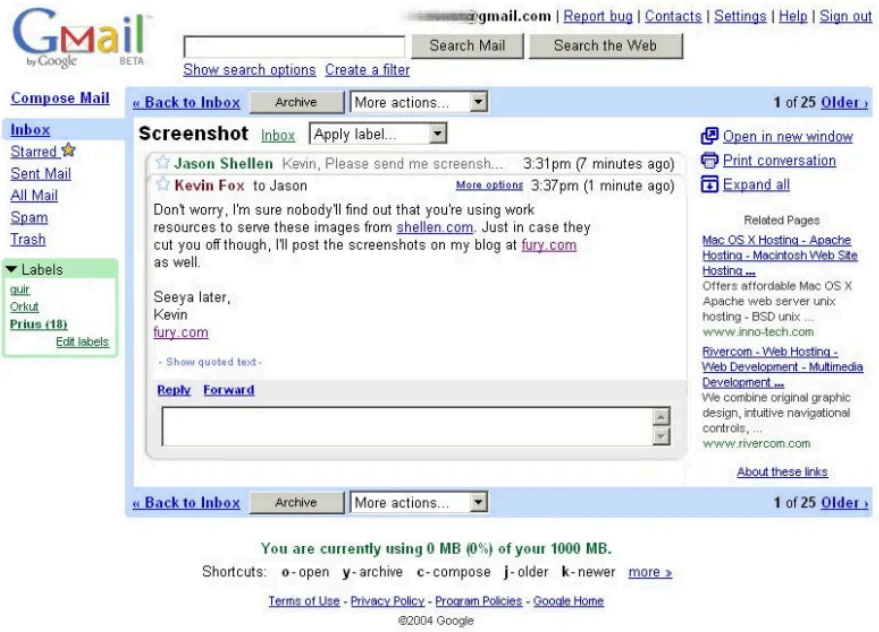
Na era Web 2.0, as páginas servidas não eram mais estáticas, mas sim dinâmicas. O dinamismo ocorria através do processamento das requisições de cada cliente no servidor. A cada requisição, ao invés do servidor entregar um documento específico já pré-processado, o servidor analisa cada requisição, processa a requisição neste momento e gera uma respostas específica. A possibilidade de processar uma requisição no momento que ela chega ao servidor, permitiu que os usuários não somente leiam o conteúdo, mas também que eles enviem dados ao servidor, aumentando a interatividade nos *websites* que utilizam esse modelo dinâmico.

A era da *Web* 2.0 e seu dinamismo permitiu a criação de serviços e aplicações que antes eram exclusivas para *desktop* (que são aplicações nativas para sistemas operacionais como Windows e MacOS), como clientes de e-mail e alguns portais que começaram restringir e processar informações exclusivas de um usuário devidamente identificado e autorizado. Esses novos serviços criados para substituir aplicações *desktop* não eram somente *sites*, mas eram aplicações completas que permitiam tanto leitura quanto entrada de informações pelos usuários através de um navegador utilizando o protocolo HTTP, foi então, que o nome *web application* passou a ser difundido.

Diversos *frameworks* e tecnologias como ASP, PHP, Django, foram criados para auxiliar no processamento de requisições dinâmicas, porém, nem sempre a experiência dos usuários eram as melhores possíveis em aplicações muito complexas e que exigiam grandes quantidades de transmissão de dados entre o cliente e servidor, como os clientes de e-mail. Então, nos anos 2000 um novo padrão começou a ser difundido, chamado de *Single Page Application* (SPA).

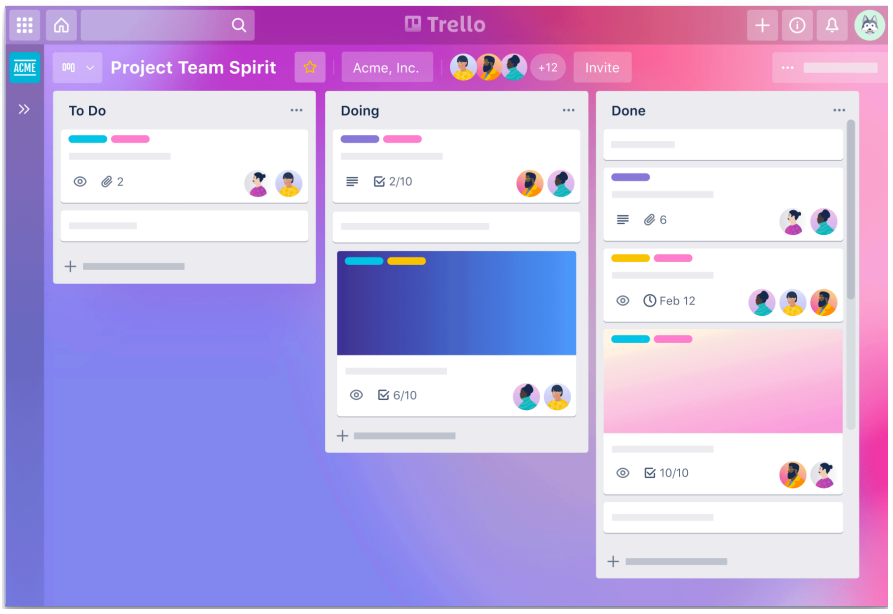
A ideia do SPA é não mais trocar páginas completas entre o cliente e servidor *web* de forma dinâmica, mas obter uma única página e permitir a troca do conteúdo desta página somente dos trechos que são necessários. Um exemplo de SPA é uma *web application* de e-mails, como o Google Gmail (Figura 3.1). Essa *web application* trafega em rede somente o conteúdo que se altera dentro de uma página e garante uma economia significativa de largura de banda de rede, além de garantir maior fluidez e interatividades para os usuários. Outra *web application* que utiliza o padrão SPA é o Trello (Figura 3.2), que ajuda times a organizarem os trabalhos através de quadros *kanban* extremamente flexíveis.

Figura 3.1: Gmail em 2004. Uma captura de tela criada pelo designer Kevin Fox



Fonte: Fox, Kevin, 2004

Figura 3.2: Captura de tela da web application Trello



Fonte: Trello, 2022

Após o lançamento de uma das primeiras grandes *web applications* nos anos 2000, o Google Gmail, diversos *frameworks* Javascript e CSS foram criados e a quantidade de *web applications* aumentou consideravelmente.

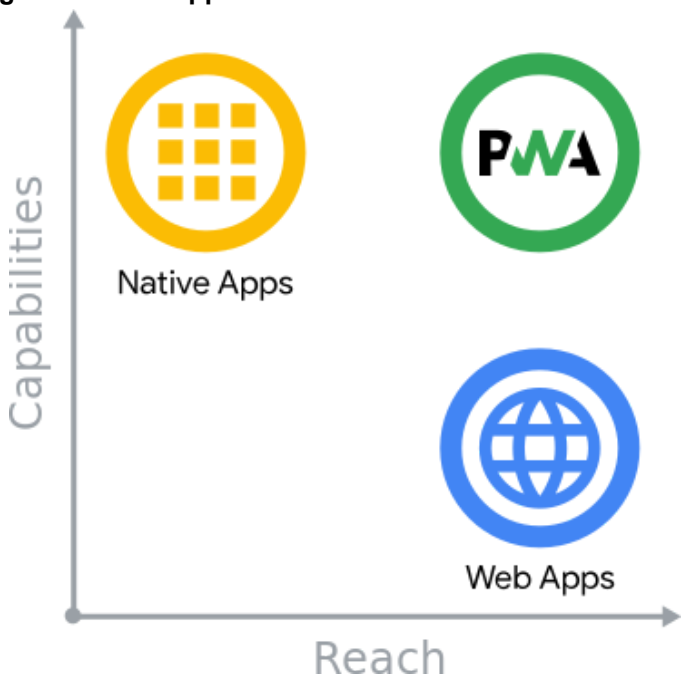
Atualmente, os *frameworks* Javascript que se destacam são React.js, Vue.js e Angular.js.

3.3. Web applications

Uma *web application* é uma aplicação desenvolvida para ser executada dentro de um navegador *web* utilizando Javascript, HTML e CSS. *Web applications* permitem grande alcance de usuários pelo fato da *world wide web* ser descentralizada e não necessitar de instalação de aplicações específicas para cada um dos sistemas operacionais (Nations, 2021). Grandes esforços são feitos por diversas empresas para que as capacidades das *web applications* se expandam unindo todo o alcance inigualável da *web* com as capacidades de *native applications*. Esses esforços se traduzem nas chamadas *Progressive Web Applications* (PWAs).

PWAs são aplicações que combinam as capacidades de *native applications* com as *web applications* (Google, 2020). *Native applications* têm acessos exclusivos ao sistema operacional e consequentemente ao dispositivo em execução, como o acesso ao microfone, câmeras, leitura e escrita no sistema de arquivos. Já as *web applications* têm um grande alcance de usuários, porque não obrigam que os usuários instalem uma aplicações específica e na maioria das vezes não precisam conceder permissões intrusivas, os usuários precisam apenas de acesso a internet e um navegador *web* que atualmente está disponível em quase todos os sistemas operacionais *desktop* e móvel (Figura 3.3).

Figura 3.3: Capacidade vs. alcance de aplicações de plataformas específicas, web application, e progressive web applications.



Fonte: Google, 2020

O que vamos construir é uma *web application*, porém, com algumas características que se aproximam de uma PWA, como: layout responsivo e funcionamento *off-line* trabalhando com *caches* locais no browser.

Neste texto vamos construir as bases para uma *web application* de *e-commerce* que chamaremos de Impacta Commerce, que terá algumas telas para simular um fluxo simples de experiência de compra de produtos *on-line* para um usuário. Como esta é uma aplicação de exemplo, suspenderemos alguns rigores na programação porque o objetivo principal é nos atermos à experimentação dos *frameworks* para a criação de uma *web application* utilizando tecnologias modernas, produtivas e de alta qualidade.

A *web application* seguindo o modelo SPA que criaremos utilizará o *framework* Next.js e consequentemente o *framework* React.js.

Next.js é um *framework* que combina diversas funcionalidades importantes para o ambiente de produção, além de garantir uma ótima e produtiva experiência de desenvolvimento *web*. Por padrão, o Next.js inclui diversas funcionalidades que são utilizadas por diversas grandes empresas de tecnologia, aumentando a qualidade da *web application* que está sendo construída desde o início. Algumas das principais funcionalidades são:

- **Renderização híbrida**. Utilizando o *framework* é possível pré-renderizar as páginas em tempo de *build* ou em tempo de requisição, aumentando a disponibilidade e fluidez na interatividade do usuário.
- **Suporte a Typescript**. Typescript é uma linguagem de programação criada pela Microsoft que, principalmente, adiciona tipagem forte ao Javascript e permite melhor integração com ferramentas de desenvolvimento, garantindo grande produtividade ao processo de desenvolvimento de software e melhor manutenibilidade. O Next.js permite utilizar Typescript sem nenhuma configuração adicional, bastando apenas que você nomeie os arquivos como *.ts ao invés de *.js;
- **Fast refresh**. Permite *feedback* instantâneo durante as alterações no ambiente de desenvolvimento. Toda vez que a *web application* estiver em executando em modo de desenvolvimento local e um alteração em qualquer arquivo for aplicada, essa alteração será refletida automaticamente no ambiente da *web application* em execução, criando o *loop* de desenvolvimento mais produtivo e claro para o desenvolvedor.
- **Otimização de imagens**. Aumenta a performance da *web application* sempre servindo o tamanho correto da imagem para cada dispositivo, aumentando significativamente o tempo de carregamento das páginas e garantindo uma melhor experiência e ranqueamento dos sites de busca como Google Search e Bing.

Além das funcionalidades destacadas, o Next.js possui diversas outras funcionalidades e facilidades, como: roteamento de páginas baseadas no sistema de arquivos; padrões para internacionalização de conteúdo; suporte para SASS e CSS globais; tags de cabeçalho customizadas por página.

O *framework* Next.js é amplamente utilizado pela indústria e está presente em diversos ambientes e cargas de uso diferentes através de *web applications* de grandes empresas de tecnologia, como TikTok, Netflix, Twitch, Github, Hulu, Nike, e diversas outras.

3.3.1 Primeiros passos com Next.js

Para iniciarmos a criação da nossa *web application* com Next.js, é necessário que você tenha instalado em seu ambiente o Node.js na versão 10.3 ou superior.

Verifique a versão atual do Node.js instalada a partir do comando a seguir.

```
1. $ node --version
```

No nosso caso, a versão utilizada é a 12.18.3. Então, o retorno do comando anterior é:

```
1. $ v12.18.3
```

Se você estiver utilizando Windows, recomendamos que você instale e utilize o *Git For Windows* (*Git for Windows Application*, 2007), para ter acesso aos comandos *Bash* e suporte a comandos específicos no formato Unix.

Caso ainda não tenha instalado o Node.js, siga as instruções em https://nodejs.org/ para instalação.

Agora que temos todos os pré-requisitos instalados, vamos criar uma aplicação de exemplo utilizando o Next.js.

Através do comando *npx create-next-app* você criará o conjunto de arquivos iniciais para um projeto Next.js usando NPM, que é o gerenciador de pacotes padrão do para Node.js. Então, execute os comandos abaixo:

```
1. $ mkdir -r ~/workspaces/impacta/
2. $ cd ~/workspaces/impacta/
3. $ npx create-next-app --use-npm impacta-commerce-webapp
```

Os comandos anteriores estão numerados linha a linha. A linha de número 1 cria uma pastas chamadas *impacta* dentro de uma pasta chamadas *workspaces* no diretório *home* para usuário corrente do sistema operacional. Após criar a pasta *impacta*, a linha 2 executa o comando para navegar para o diretório criado. Na linha 3, criaremos a nossa web application utilizando Next.js.

Ao final do terceiro comando, diversas informações serão impressas, mas para termos certeza que ele foi executado com sucesso, a mensagem a seguir deve estar presente no terminal.

```
1. $ Success! Created impacta-commerce-webapp at …
```

O último comando executado criará uma nova aplicação chamada *impacta-commerce-webapp* em uma pasta de mesmo nome com todos os arquivos básico e iniciais para configuração de uma aplicação mínima Next.js, e já com alguns arquivos de páginas, APIs, estáticos e estilos CSS iniciados para imediata execução da aplicação localmente.

A estrutura de arquivos criados deve ser conforme a seguir:

```
.
├── README.md
├── next.config.js
├── package-lock.json
├── package.json
├── node_modules
│   ├── …
├── pages
│   ├── _app.js
│   ├── api
│   │   └── hello.js
│   └── index.js
├── public
│   ├── favicon.ico
│   └── vercel.svg
└── styles
    ├── Home.module.css
    └── globals.css
```

O arquivo *README.md*, contém instruções básicas sobre a aplicação criada. Incluindo instruções de como executar a aplicação e referências para tutorias de documentações sobre Next.js.

Os arquivos *next.config.js, package.json* e *package-lock.json* contém configurações do *framework* Next.js, configurações de pacotes Node.js utilizando o NPM e a exata árvore de dependências entre os pacotes gerados na pasta *node_modules*, respectivamente.

Alguns outros diretórios também foram criados. O diretório *pages* contém os arquivos de extensão *.js* que representam as páginas *web* e o diretório chamado *api* contém todos os arquivos *.js* para suportar as rotas de API da *web application*. Já o diretório *public* contém os arquivos que serão utilizados como estáticos para a aplicação. São arquivos de imagens, ícones, fontes, e qualquer outro arquivo que são para somente leitura. O diretório *styles* contém arquivos de extensão *.css* para aplicação de folhas de estilos específicas a algum componente ou globais para todo a *web application*.

Agora, vamos executar a aplicação localmente em modo de desenvolvimento para visualizar os resultados, através do comando npm.

```
1. $ npm run dev
```

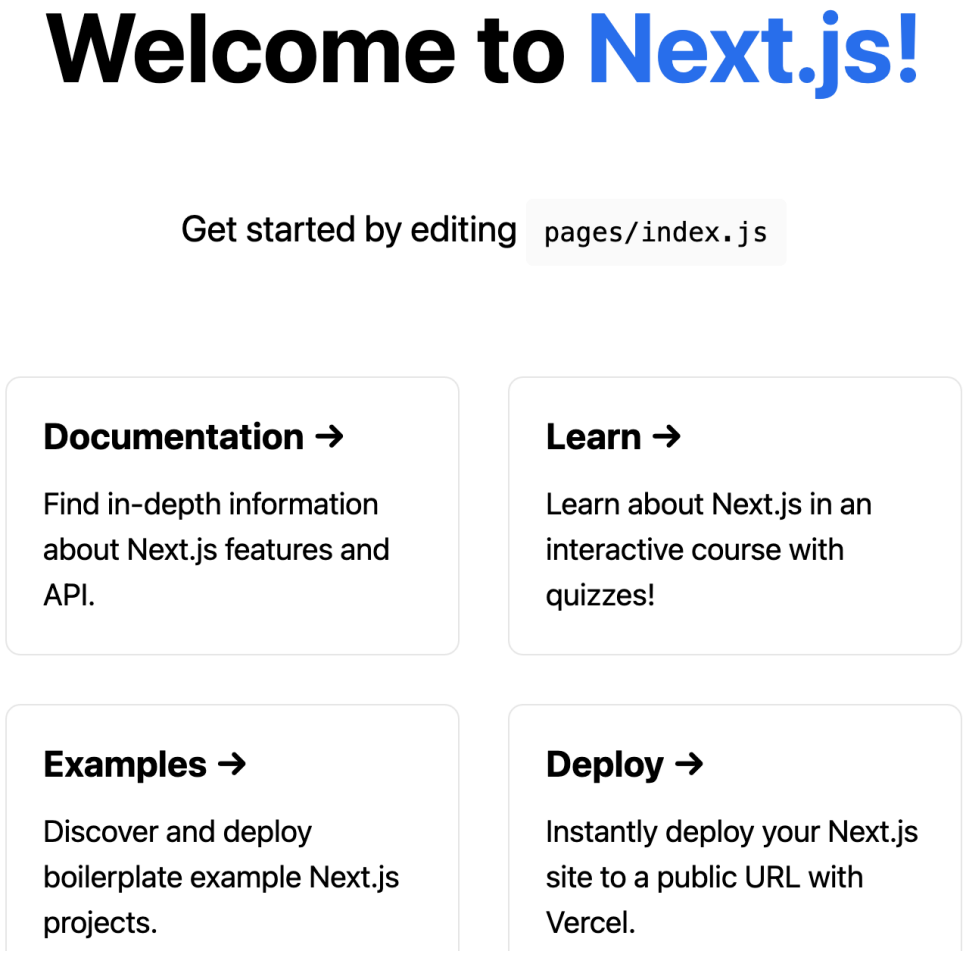
O resultado do comando deve ser conforme a seguir:

```
1. > dev
2. > next dev
3.
4. ready - started server on 0.0.0.0:3000, url: http://localhost:3000
5. (node:5601) ExperimentalWarning: The ESM module loader is
experimental.
6. event - compiled client and server successfully in 1995 ms (125
modules)
```

Essa saída significa que um servidor foi iniciado localmente pelo terminal e responderá na porta 3000, conforme descrito na linha 4. Esse é o endereço que pode ser utilizado para navegar e visualizar a sua nova *web application* utilizando Next.js.

Abra o seu navegador e navegue para o endereço http://localhost:3000. O resultado deve ser igual a Figura 3.4.

Figura 3.4: Página inicial da *web application* impacta-commerce-webapp



Fonte: do Autor, 2022.

Para fazermos o nosso primeiro teste de edição, abra o arquivo em *pages/index.js*, procure o texto *Welcome to Next.js!*, e substitua-o conforme instruções a seguir.

Antes:

```
Welcome to <a href="https://nextjs.org">Next.js!</a>
```

Depois:

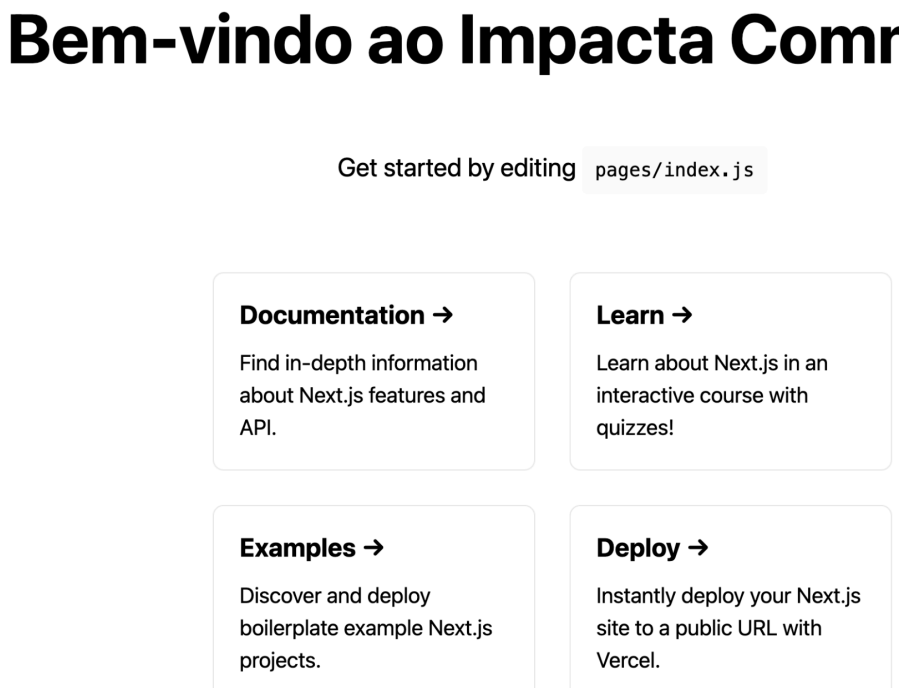
```
Bem-vindo ao Impacta Commerce!
```

Após salvar o arquivo, duas coisas aconteceram. A primeira, duas novas mensagens aparecerão no terminal, informando que o código está sendo compilado imediatamente e uma outra informando que a compilação foi executada com sucesso em um tempo específico.

```
1. wait  - compiling...
2. event - compiled client and server successfully in 217 ms (135 modules)
```

A segunda, a página do endereço http://localhost:3000/ será atualizada automaticamente no navegador com o novo conteúdo. Agora, a página deve estar conforme a Figura 3.5.

Figura 3.5: Página inicial da *web application* impacta-commerce-webapp após alteração.



Fonte: do Autor, 2022.

Pronto, a sua primeira *web application* usando Next.js foi criada com sucesso e está pronta para ser alterada conforme as especificações e necessidades de negócios.

3.4 API REST

Web application frequentemente necessitam acessar serviços externos, chamados de *Web Services*, para enviar e-mail, executar pagamentos, armazenar, processar e

consultar informações específicas necessárias para o negócio, fora diversas outras operações. Nesta seção, veremos como construir uma API REST para o Impacta Commerce.

3.4.1 O que são APIs REST?

Application Programming Interfaces (API), ou em português, Interfaces de Programação de Aplicação, são um conjunto de operações em alto nível de abstração que permitem manipulação de um determinado serviço através de programação, ou seja, são operações que não são executadas manualmente, mas automaticamente através de uma programação.

Alguns exemplos de APIs são: APIs para manipulação do sistema de arquivos e janelas dos sistemas operacionais (Windows, Linux). Em ambos os casos, existem inúmeras rotinas executadas internamente pelo sistema operacional para garantir que um arquivo seja criado, escrito ou lido, bem como inúmeras operações para exibir uma janela em um dos monitores disponíveis para exibição no seu computador. Porém, quando criamos uma aplicação e precisamos escrever um arquivo ou desenhar uma tela com determinados conteúdos internos, não precisamos saber quais são essas operações internas dos sistemas operacionais (e cada sistema operacional tem o seu conjunto de operações específicas para cada propósito), precisamos apenas, conhecer as interfaces de alto nível de abstração e executá-las através de uma programação para o arquivo seja criado ou a janela exibida.

As API de sistemas operacionais, são APIs de baixo nível e só podem ser executadas localmente no sistema operacional, através de protocolos específicos de comunicação local no sistema operacional. Porém, quando criamos uma *web application*, a execução dos serviços sempre será remota, onde a *web application* local faz uma requisição de execução em um serviço remoto que pode estar disponível em qualquer lugar do planeta. Esses procedimentos são chamados de *Remote Procedure Call* (RPC). Dessa forma, *web applications* precisam utilizar APIs que suportem um protocolo baseado em HTTP, que é o protocolo de comunicação suportado pelos navegadores.

APIs que utilizam o protocolo de comunicação baseado em HTTP, são chamadas de *Web* APIs ou *Web Services*. Existem diversos protocolos de comunicação sobre o protocolo HTTP, um dos mais conhecidos e utilizados é o protocolos *Simple Object Access Protocol* (SOAP), que se baseia em XML para formatar mensagens (*Web Services Architecture*, 2004). Porém, o SOAP tem sido utilizado em cenários mais específicos, que precisam de um protocolo formal e bem especificado para comunicação. Outra alternativa muito utilizada atualmente no desenvolvimento de aplicações web, é o estilo arquitetural chamado *Representational State Transfer* (REST) (Fielding, Roy Thomas, 2000), já que é mais simples de entender, utiliza apenas a tecnologia e os padrões HTTP e HTTPS e qualquer cliente ou até mesmo servidor tem grande suporte a este protocolo (*REST E SOAP: Usar Um Dos Dois Ou Ambos?*, 2013).

As APIs REST são *Web* APIs que utilizam o estilo arquitetural REST como protocolo de comunicação. Embora o REST seja apenas uma abordagem e não oferece algumas facilidades presente no SOAP, como contratos formais de comunicação, operações baseadas em estado (operações *stateful*), procedimentos e chamadas

assíncronas, o REST é muito utilizado pela simplicidade e grande flexibilidade para criar *web services* que executam operações não baseadas em estados (operações *stateless*), operações que podem ser cacheadas, e principalmente onde existe uma limitação de largura de banda, porque utiliza geralmente JSON como conteúdo que é muito menos verboso do que o conteúdo em XML trafegado pelo SOAP (*Topics Understanding Enterprise Integration REST Vs. SOAP*, 2019).

Para a construção da *web application* do Impacta Commerce, criaremos uma API REST para listar os produtos disponíveis em estoque, armazenar os carrinhos de compras e executar uma simulação de pagamento de um carrinho de compras. Esta API REST será criada utilizando a linguagem de programação Python e o *framework* Flask.

3.4.2 Primeiros passos com Flask

O Flask é um *framework* Python para auxiliar na criação de *web applications* ou *web services* baseados em REST. Ele é caracterizado como um *microframework*, que significa que o seu código é simples e carrega por padrão apenas as funcionalidades essenciais para o que se propõe, mas pode ser facilmente estendido para incorporar mecanismos de acesso a bancos de dados ou validações através de outros *frameworks* já existentes.

Para criar a nossa API REST que será utilizada pela *web application impacta-commerce-webapp* é necessário ter pelo menos o Python 3.6 instalado em seu sistema operacional. É recomendado ter a versão mais atual possível. Verifique se o Python está instalado utilizando o comando a seguir.

```
1. python3 --version
```

Se o Python estiver instalado adequadamente, o comando retornará a versão instalada em sua máquina do Python 3. No nosso caso, o retorno foi:

```
1. Python 3.8.9
```

Isso significa que temos o Python na versão 3.8.9 instalada na máquina local.

Assim como na nossa *web application* em Next.js, na nossa API REST também teremos dependências com bibliotecas *open-source*. Para gerenciá-la adequadamente é recomendada a criação de ambientes virtuais para garantir que não tenhamos incompatibilidades de versões quando precisamos construir e entregar a aplicação em ambientes diferentes, como ambiente local ou ambiente de produção.

Para criação do projeto base, crie um diretório chamado *impacta-commerce-webapi* no mesmo diretório do projeto *impacta-commerce-webapp* e navegue para o novo diretório criado. Então, se você criou o diretório *impacta-commerce-webapp* dentro do diretório *~/workspaces/impacta/*, isso significa que você também deve criar o diretório *impacta-commerce-webapi* dentro de *~/workspaces/impacta/*. Esse requisito é apenas para facilitar nossa organização e localização, mas nada impede que você utilize qualquer outra estrutura de diretórios que achar mais adequada, porém, todas as referências e instruções a seguir serão baseadas nesses caminhos. Veja a seguir os comandos que devem ser executados no terminal para a criação da estrutura indicada e o ambiente virtual python.

```
1. $ mkdir impacta-commerce-webapi
2. $ cd impacta-commerce-webapi
3. $ python3 -m venv venv
4. $ . venv/bin/activate
```

O resultado final dos comando deve ser:

```
1. (venv)
```

Isso significa que o diretório e o ambiente virtual foram criados com sucesso e também que o ambiente virtual chamado *venv* está ativo.

Agora, vamos instalar a dependência do *framework* Flask utilizando o comando *pip*. O comando *pip* é o comando de terminal do PyPA, um instalador de pacotes Python. Se você não tem o comando *pip* disponível em seu terminal, veja como instalá-lo em https://pip.pypa.io/en/stable/installation/.

```
1. $ pip install Flask
```

Alguns pacotes serão baixados para sua máquina e todos serão descritos na saída do comando executado anteriormente. É importante que a mensagem de sucesso esteja presente.

```
1. Successfully installed Flask-2.0.2…
```

No nosso caso, a versão do pacote instalado para o Flask foi 2.0.2. É recomendado que você utilize a mesma versão ou superior.

Uma aplicação Flask deve conter minimamente a dependência para o *framework* e uma rota com um retorno padrão. Vamos criar um arquivo chamado *hello.py* contendo o código mínimo para executar a nossa *web api* e abri-lo no Visual Studio Code (VSCode). No nosso caso, vamos utilizar o VSCode como nosso *Integrated Development Environment* (IDE) tanto para desenvolvimento da *web api* quanto para a *web application.*

```
1. $ touch hello.py
2. $ code hello.py
```

Com o arquivo aberto no VSCode, adicione o código a seguir e salve o arquivo.

```
1. from flask import Flask
2. app = Flask(__name__)
3. @app.route("/")
4. def hello_world():
5.     return "<p>Hello, World!</p>"
```

No terminal, execute o comando conforme abaixo.

```
1. $ export FLASK_APP=hello
2. $ flask run
```

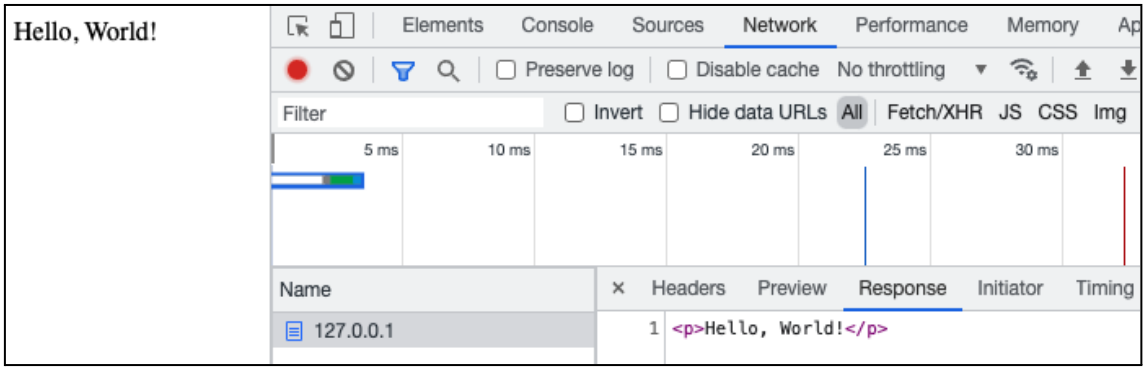
A primeira linha informa ao terminal via variável de ambiente que o nome do arquivo de inicialização da aplicação utilizando Flask é *hello*. A segunda linha, inicializa a aplicação. Caso você tenha algum problema com o comando *flask run*, utilize *python -m flask*.

Como saída para o comando *flask run* algumas instruções serão impressas, mas a mais importante é a informação de que o servidor está em execução, informando o endereço e a porta. No nosso caso o resultado foi:

```
1. * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Navegue para o endereço http://127.0.0.1:5000/, que foi informado no terminal, e veja que o resultado é justamente o que escrevemos no arquivo *hello.py*, Figura 3.6.

Figura 3.6: Resultado do HTML renderizado (à esquerda) e processamento da requisição como HTML não renderizado (à direita)



Fonte: do Autor, 2022.

Aqui finalizamos a criação do projeto base da nossa API REST impacta-commerce-webapi que será utilizada pela *web application* chamada impacta-commerce-webapp.

3.5 GitHub

Futuramente, ambas as aplicações devem ser entregues em produção ou em ambientes públicos para que os clientes consigam acessá-las através da internet. Também pode ser necessário que outros desenvolvedores participem da criação de novas funcionalidades ou dêem manutenção nas funcionalidades existentes. Para facilitar ambos os casos, vamos utilizar repositórios Git para todos o código fonte construído e armazená-los no Github, que é uma plataforma para hospedagem de código-fonte e arquivos gratuita que utiliza Git.

3.5.1 Repositório Git para a Web Application

Crie um repositório chamado *impact-commerce-webapp* seguindo as instruções em https://docs.github.com/pt/get-started/quickstart/create-a-repo.

Antes de enviarmos os arquivos, precisamos informar ao repositório Git quais arquivos ele não deve manter uma trilha; quais arquivos ele deve ignorar. Para isso, criaremos um arquivo oculto chamado *.gitignore*, cujo o conteúdo deve ser o conteúdo gerado pelo link a seguir:

```
https://www.toptal.com/developers/gitignore/api/linux,windows,macos,visualstudiocode,sublimetext,nextjs,node,sass,python,flask,react.
```

Após ter criado o repositório no Github e salvado o arquivo *.gitignore* no diretório da aplicação *impacta-commerce-webapp*, dentro deste diretório execute os comandos a seguir substituindo o *{endereço-do-seu-repo}* pelo endereço do seu repositório criado no Github.

```
1. $ git init
2. $ git add .
3. $ git commit -m "Aplicação base"
4. $ git remote add origin https://github.com/{endereço-do-seu-repo}
5. $ git branch -M main
6. $ git push -u origin main
```

Esses comandos devem inicializar o repositório Git, adicionar todos os arquivos ao repositório, adicionar um endereço de repositório Git remoto de origem, criar a *branch main* e enviar todo o código para o Github.

Ao final da execução do comando verifique se todos os arquivos foram enviados para o Github visitando a página do seu repositório através do navegador.

3.5.2 Repositório Git para a Web API

Os mesmos procedimentos executados em 2.5.1 Repositório Git para a Web Application, também devem ser executados para a aplicação *web api*. Porém, criando o seu próprio repositório no Github e substituindo de forma adequada os nomes e endereços para a *web api*.

3.6. Considerações finais

Se todos os passos foram executados conforme descritos, duas aplicações base foram criadas: uma aplicação *web application* para *front-end* utilizando o *framework* Next.js e outra aplicação web api para *back-end* utilizando o *framework* Flask.

Essa arquitetura é base para resolução de diversos problemas do mundo real utilizando soluções *web*.

Referências

Dougherty, D. **World Wide Web and Its Journey from Web 1.0 to Web 4.0**. Disponível em: http://ijcsit.com/docs/Volume%205/vol5issue06/ijcsit20140506265.pdf. Acesso em: 12 Jan. 2022.

Git for Windows Application (2007). Git for Windows. Disponível em: https://gitforwindows.org/. Acesso em 14 jan. 2022.

Google (2020). **What are Progressive Web Apps?** web.dev. Disponível em: https://web.dev/what-are-pwas/. Acesso em: 12 jan. 2022.

McCracken, H. (2014, April 1). **How Gmail Happened:** The Inside Story of Its Launch 10 Years Ago. Time Magazine. Disponível em: https://time.com/43263/gmail-10th-anniversary/. Acesso em: 14 jan. 2022.

Nations, D. (2021, June 24). **What Is a Web Application?** Lifewire. Disponível em: https://www.lifewire.com/what-is-a-web-application-3486637. Acesso em: 14 jan. 2022.

Trello: O Trello ajuda os times a agilizar o trabalho. Disponível em: https://trello.com/pt-BR. Acesso em: 14 jan. 2022

REST e SOAP: Usar um dos dois ou ambos? (2013, October 2). InfoQ. Disponível em: https://www.infoq.com/br/articles/rest-soap-when-to-use-each/. Acesso em: 14 jan. 2022.

Fielding, Roy Thomas. **Architectural Styles and the Design of Network-based Software Architectures**. Doctoral dissertation, University of California, Irvine, 2000. Disponível em: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. Acesso em: 14 jan. 2022.

Topics Understanding enterprise integration REST vs. SOAP. (2019, April 8). Red Hat. Disponível em: https://www.redhat.com/en/topics/integration/whats-the-difference-between-soap-rest . Acesso em 14 jan. 2022.

Web Services Architecture. (2004, February 11). W3C. Disponível em: https://www.w3.org/TR/ws-arch/#SOAP. Acesso em: 14 jan. 2022.

Flask: Web Development, one drop at a time. Disponível em: https://flask.palletsprojects.com/en/2.0.x/. Acesso em: 10 Dez de 2021.

Node.js: a JavaScript runtime built on Chrome's V8 JavaScript Engine. Disponível em: https://nodejs.org/en/. Acesso em: 10 Dez de 2021.