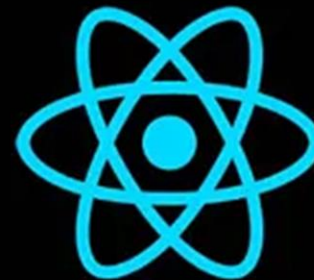


FIAP

RESPONSIVE WEB DEVELOPMENT

NEXT.js



Prof. Alexandre Carlos

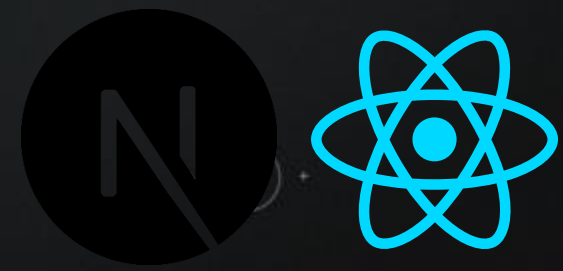
profalexandre.jesus@fiap.com.br

Prof. Luís Carlos

lsilva@fiap.com.br

Prof. Wellington Cidade

profwellington.tenorio@fiap.com.br

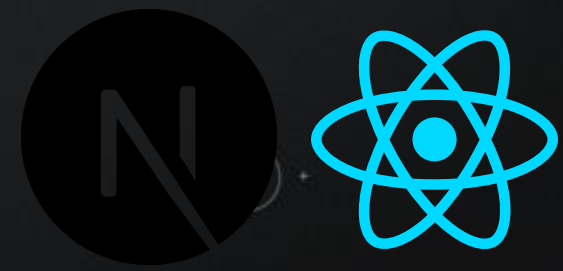


NEXT.JS

FIAP

O QUE É O NEXT.JS?

- △ O Next.js é um framework React para a construção de aplicações web full-stack. Use os Componentes React para criar interfaces de usuário e Next.js para recursos e otimizações adicionais.
- △ Sob o capô, o Next.js também abstrata e configura automaticamente as ferramentas necessárias para o React, como agrupamento, compilação e muito mais. Isso permite que você se concentre na construção do aplicativo em vez de passar tempo com a configuração.
- △ Seja você um desenvolvedor individual ou parte de uma equipe maior, o Next.js pode ajudá-lo a criar aplicativos React interativos, dinâmicos e rápidos.



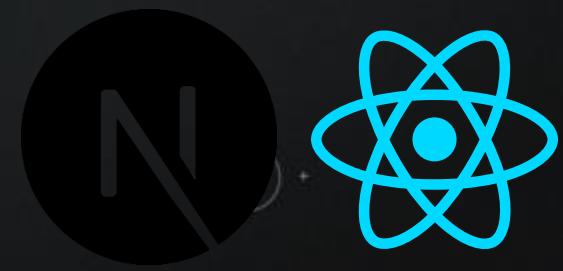
NEXT.JS

FIAP

PRINCIPAIS CARACTERÍSTICAS

△ Alguns dos principais recursos do Next.js incluem:

Característica	Descrição
ROUTING	Um roteamento baseado em sistema de arquivos construído em cima de Server Components que suporta layouts, Nested Routs, loading estates, manipulação de erros (error handling), e muito mais.
RENDERING	Client-side and Server-side rendering, com Client and Server Components. Além disso, otimizado com Renderização Estática e Dinâmica no servidor com Next.js. Streaming em Edge e Node.js runtimes.
DATA FETCHING	Dados simplificados que buscam com async/await em Server Components e fetch API para solicitação de memorização, cache de dados e revalidação.
STYLING	Suporte para seus métodos de estilo preferidos, incluindo Módulos CSS, Tailwind CSS e CSS-in-JS
OPTMIZATIONS	Otimizações de imagem, fontes e scripts para melhorar as principais visualizações e experiência do usuário da Web do seu aplicativo.



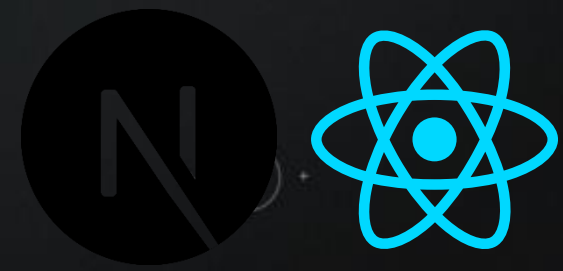
NEXT.JS

FIAP

PRINCIPAIS CARACTERÍSTICAS

△ React Server Components permitem que você escreva uma interface do usuário que pode ser renderizada e, opcionalmente, armazenada em cache no servidor. No Next.js, o trabalho de renderização é dividido por segmentos de rota para permitir streaming e renderização parcial, e existem três estratégias diferentes de renderização de servidor:

- ***Renderização estática***
- ***Renderização dinâmica***
- ***Transmissão***



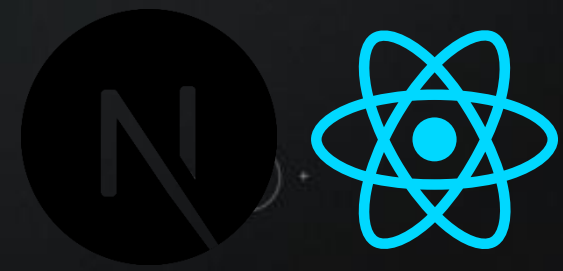
NEXT.JS

FIAP

PRINCIPAIS CARACTERÍSTICAS

△ Benefícios de se usar React Server Components:

- Mover dados pesquisados para o servidor, ou seja, mais perto da fonte de dados. Retirando a carga do cliente, tornando a leitura mais rápida.
- Permite que você mantenha dados confidenciais e lógica no servidor, como tokens e chaves de API, sem o risco de expô-los ao cliente.
- Armazenamento em cache e reutilização de Requests subsequentes e entre usuários. Isso pode melhorar o desempenho e reduzir o custo, reduzindo a quantidade de renderização e busca de dados feitos em cada Request.
- **Tamanhos** de pacote: Permite manter grandes dependências que anteriormente afetaria o tamanho do pacote JavaScript do cliente no servidor. Isso é benéfico para usuários com internet mais lenta ou dispositivos menos poderosos, pois o cliente não precisa baixar, analisar e executar qualquer JavaScript.



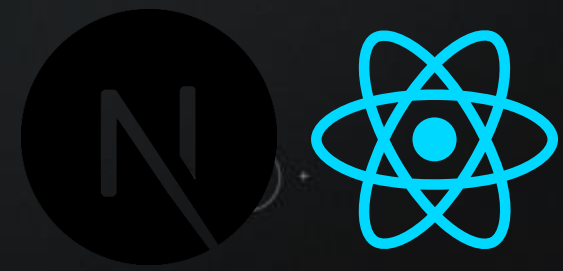
NEXT.JS

FIAP

PRINCIPAIS CARACTERÍSTICAS

△ Benefícios de se usar React Server Components:

- △ **Carga inicial da página (FCP):** No servidor, podemos gerar HTML para permitir que os usuários visualizem a página imediatamente, sem esperar que o cliente baixe, verifique e execute o JavaScript necessário para renderizar a página.
- △ **Search Engine Optimization e Social Network Shareability :** O HTML renderizado pode ser usado por bots de mecanismos de pesquisa para indexar suas páginas e bots de redes sociais para gerar visualizações de cards sociais para suas páginas.
- △ **Streaming:** Os componentes do servidor permitem que você divida o trabalho de renderização em pedaços e transmiti-los ao cliente à medida que eles se tornam prontos. Isso permite que o usuário veja partes da página mais cedo sem ter que esperar que a página inteira seja renderizada no servidor.



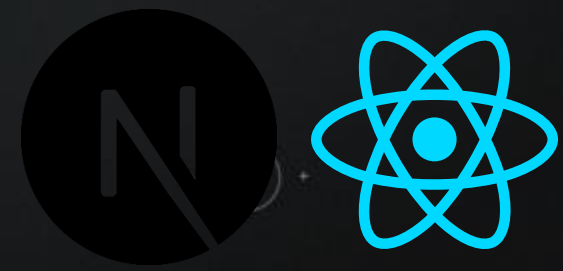
NEXT.JS

FIAP

PRINCIPAIS CARACTERÍSTICAS

△ Benefícios de se usar Client Components:

- △ Permitem que você escreva UI interativa que pode ser renderizada no cliente no momento do pedido. No Next.js, a renderização do cliente é opt-in, o que significa que você precisa decidir explicitamente quais componentes o React deve renderizar no cliente.
- △ **Interatividade:** Podem usar o estado, os efeitos e os ouvintes de eventos, o que significa que podem fornecer feedback imediato ao usuário e atualizar a interface do usuário.
- △ **APIs de navegador:** Têm acesso às APIs do navegador, como a geolocalização ou localStorage, permitindo que você construa UI para casos de uso específicos.



NEXT.JS

FIAP

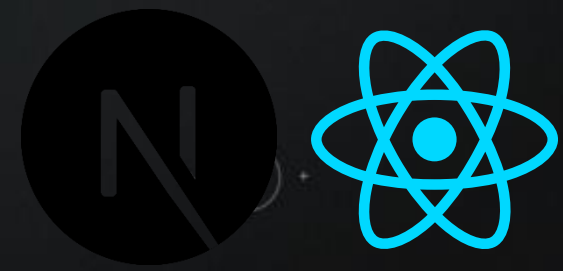
BOYLER-PLATE / HELLO WORLD / INSTALAÇÃO AUTOMÁTICA

- ❖ Recomendamos iniciar um novo aplicativo Next.js usando create-next-app, que configura tudo automaticamente para você.
- ❖ Para criar um projeto, execute:

> Terminal



```
npx create-next-app@latest
```

NEXT.JS

FIAP

BOYLER-PLATE / HELLO WORLD / INSTALAÇÃO AUTOMÁTICA

❖ Na instalação, você verá as seguintes instruções:

>_ Terminal

```
What is your project named? my-app
Would you like to use TypeScript? No / Yes
Would you like to use ESLint? No / Yes
Would you like to use Tailwind CSS? No / Yes
Would you like to use `src/` directory? No / Yes
Would you like to use App Router? (recommended) No / Yes
Would you like to customize the default import alias (@/*)? No / Yes
What import alias would you like configured? @/*
```



NEXT.JS

FIAP

BOYLER-PLATE / HELLO WORLD / INSTALAÇÃO AUTOMÁTICA

❖ Um script será executado e teremos a possibilidade de rodar o projeto e escolher o que queremos que seja instalado inicialmente; Utilize as setas direcionais para escolher a Opção.

```
npm install

D:\projetos\next>npx create-next-app@latest hello_world
✓ Would you like to use TypeScript? ... No / Yes
✓ Would you like to use ESLint? ... No / Yes
✓ Would you like to use Tailwind CSS? ... No / Yes
✓ Would you like to use `src/` directory? ... No / Yes
✓ Would you like to use App Router? (recommended) ... No / Yes
✓ Would you like to customize the default import alias? ... No / Yes
Creating a new Next.js app in D:\projetos\next\hello_world.

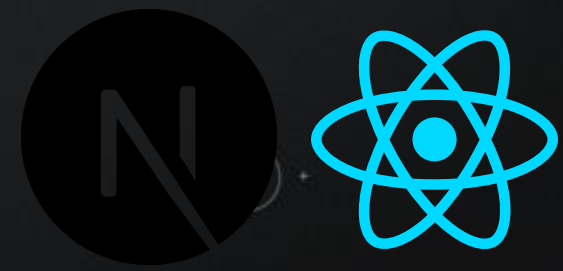
Using npm.

Initializing project with template: app

Installing dependencies:
- react
- react-dom
- next

Installing devDependencies:
- eslint
- eslint-config-next

[Progress Bar] \ reify:is-glob: timing reifyNode:node_modules
```



NEXT.JS

FIAP

BOYLER-PLATE / HELLO WORLD / INSTALAÇÃO AUTOMÁTICA

- ❖ Depois dos prompts anteriores, create-next-app irá criar uma pasta com o nome do seu projeto e vai instalar as dependências necessárias.

É bom saber:

- Next.js agora vem com configuração do TypeScript, ESLint e Tailwind CSS por padrão.
- Você pode, opcionalmente, usar a `src` diretório na raiz do seu projeto para separar o código do aplicativo dos arquivos de configuração.



NEXT.JS

FIAP

BOYLER-PLATE / HELLO WORLD / INSTALAÇÃO AUTOMÁTICA

- ❖ Agora para iniciar o projeto acessamos a pasta com o comando ***cd nome_do_projeto*** e depois iniciamos o projeto com o comando ***npm run dev***

```
Prompt de comando

106 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
Initialized a git repository.

Success! Created hello_world at D:\projetos\next\hello_world

D:\projetos\next>npm run dev
```




NEXT.JS

FIAP

BOYLER-PLATE / HELLO WORLD / INSTALAÇÃO AUTOMÁTICA

- ❖ Agora o projeto está no ar e podemos acessar segurando a **Clique** em cima do endereço que apareceu (***dependendo do seu terminal***), e o conteúdo será apresentado em nosso navegador padrão.

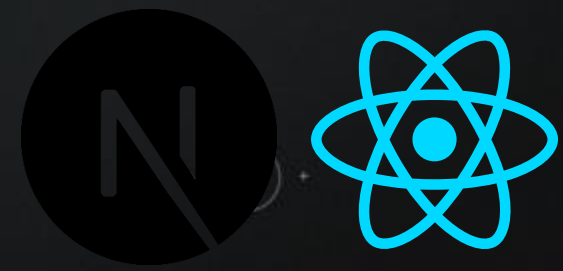
```
npm config get registry

D:\projetos\next\hello_world>npm run dev

> hello_world@0.1.0 dev
> next dev

▲ Next.js 13.5.3
- Local:      http://localhost:3000

✓ Ready in 3.3s
```



NEXT.JS

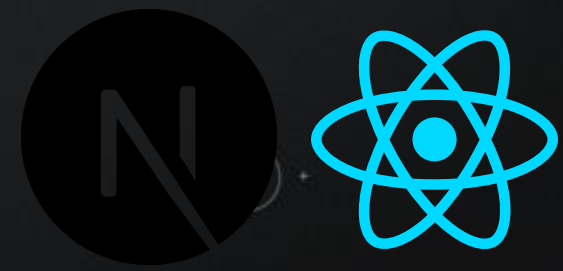
FIAP

BOYLER-PLATE / HELLO WORLD / INSTALAÇÃO AUTOMÁTICA

- ❖ Depois dos prompts anteriores, create-next-app irá criar uma pasta com o nome do seu projeto e vai instalar as dependências necessárias.

É bom saber:

- Next.js agora vem com configuração do TypeScript, ESLint e Tailwind CSS por padrão.
- Você pode, opcionalmente, usar a `src` diretório na raiz do seu projeto para separar o código do aplicativo dos arquivos de configuração.

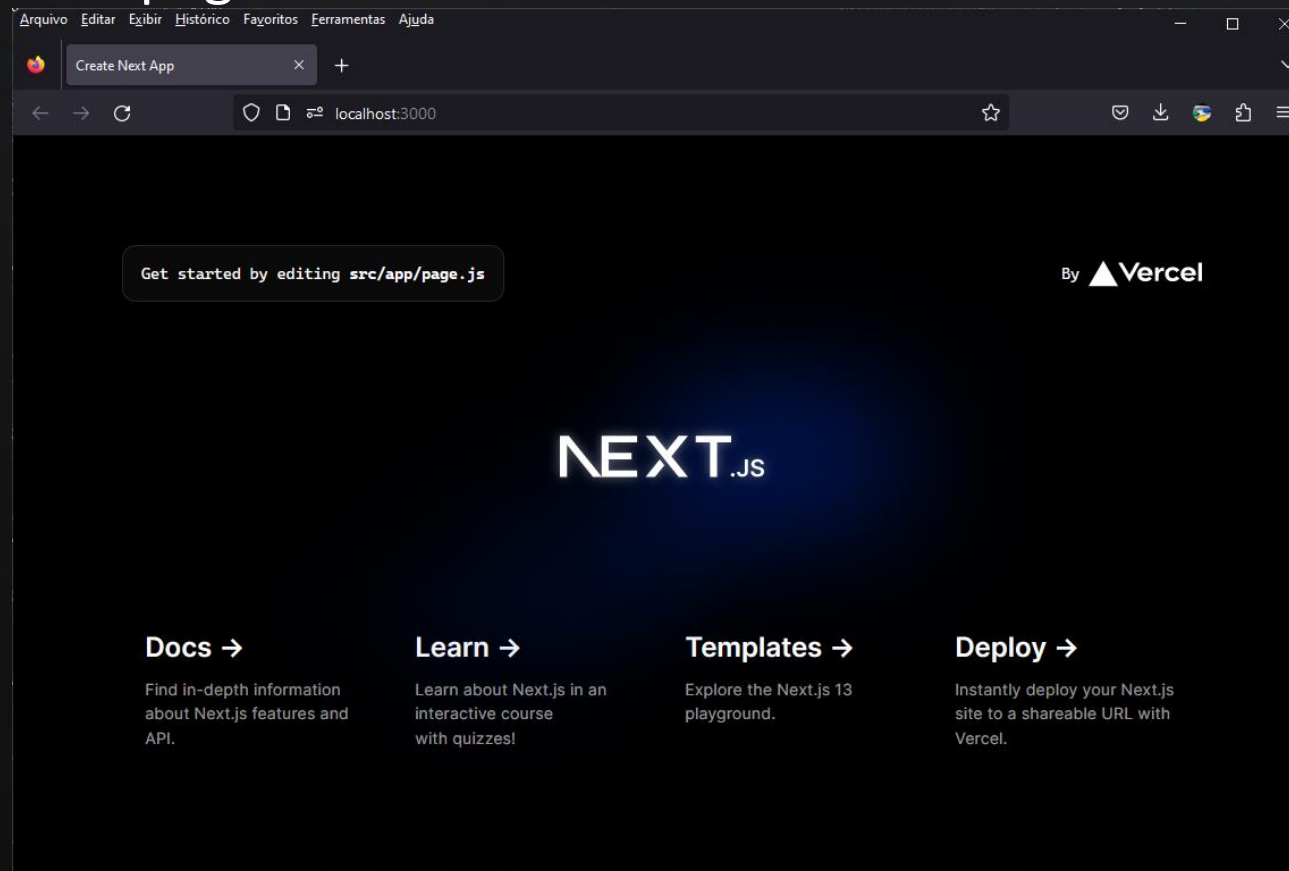


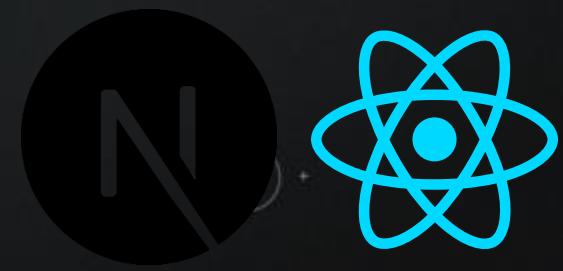
NEXT.JS

FIAP

HELLO WORLD

❖ Agora podemos ver a página rodando:



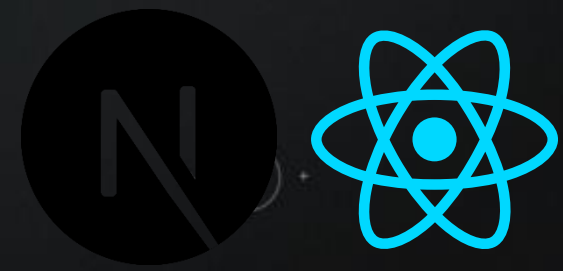


NEXT.JS

FIAAP

VERSIONANDO O PROJETO

- △ Seu projeto está pronto para o trabalhos.
- △ Agora realize a limpeza do projeto, retirando do boyler-plate, os arquivos desnecessários ao início das atividades.
- △ O NEXT.js já inicializa o repositório do GIT, então desde que você já tenha registrado na máquina o seu nome de usuário e email do Github, aproveite para criar seu repositório remoto e realizar o versionamento.
- △ *Se lembre que quando clonar o projeto novamente em qualquer lugar, vai ser necessário rodar o comando `npm install` para que as dependências sejam baixadas novamente.*



NEXT.JS

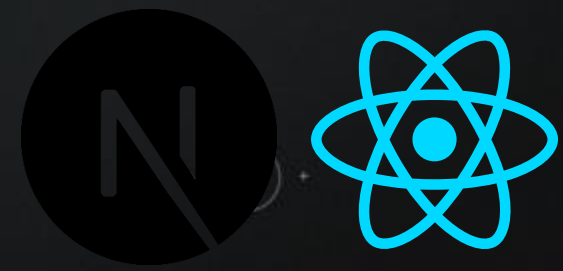
FIAP

FUNDAMENTOS DE ROTEAMENTO

△ O esqueleto de cada aplicação é o roteamento. Estes slides irão apresentá-lo aos **conceitos fundamentais** de roteamento para a web e com o Next.js.

△ Terminologia

- Primeiro, você verá esses termos sendo usados em toda a documentação. Aqui está uma referência rápida:



NEXT.JS

FIAP

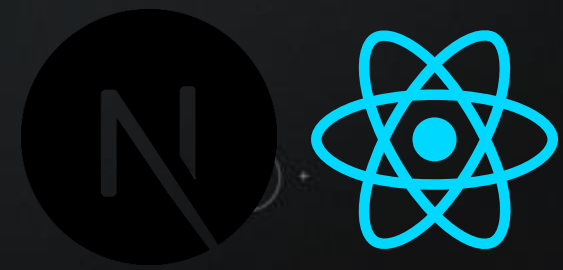
ESTRUTURA DO PROJETO NEXT.JS

△ Esta página fornece uma visão geral da estrutura de arquivos e pastas de um projeto Next.js. Ele cobre arquivos e pastas de nível superior, arquivos de configuração e convenções de roteamento dentro da pasta app os arquivos pages.

△ Os diretórios:

Pastas de nível superior

<code>app</code>	Roteador do aplicativo
<code>pages</code>	Rotações de Páginas
<code>public</code>	Ativos estáticos a serem servidos
<code>src</code>	Pastas de origem de aplicação opcionais

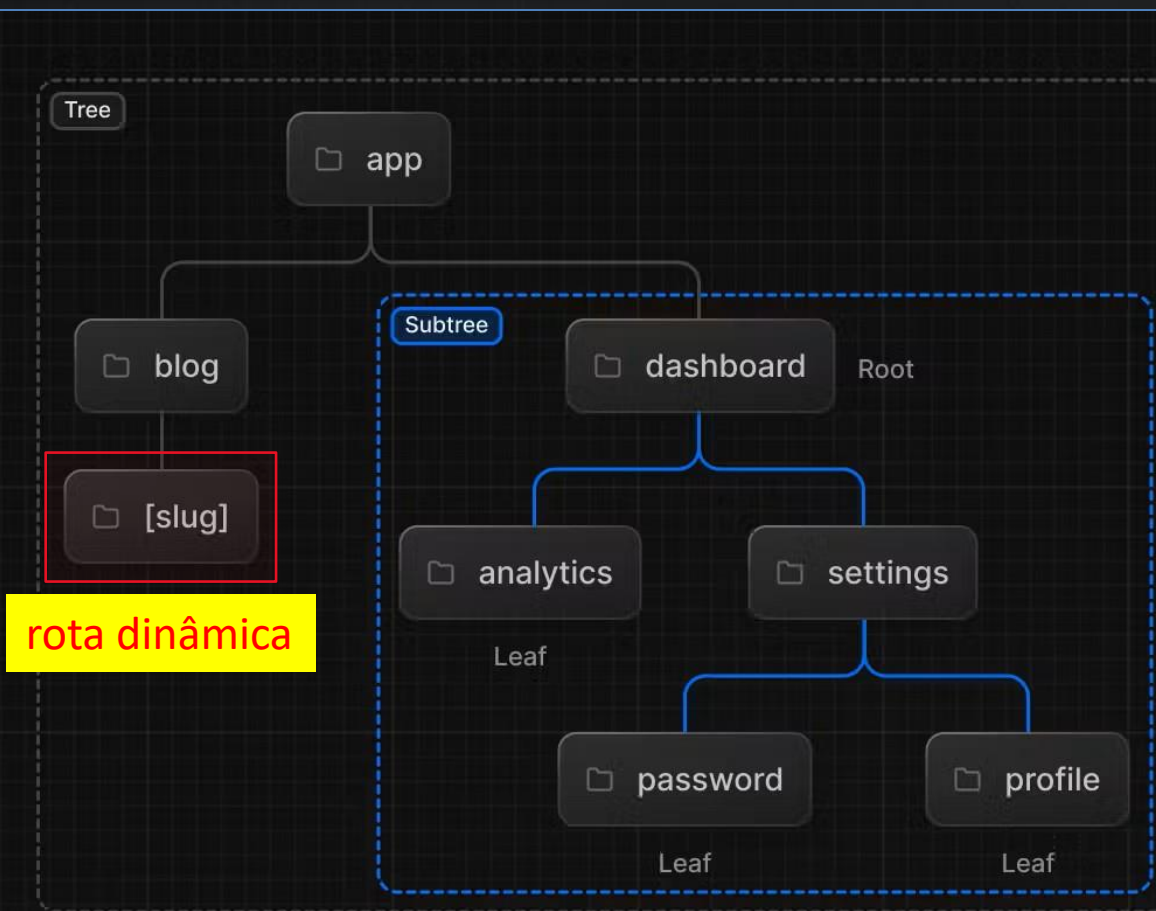


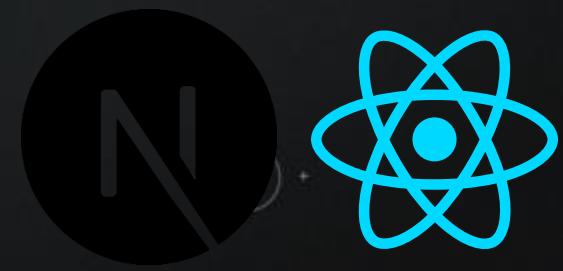
NEXT.JS

FIAP

FUNDAMENTOS DE ROTEAMENTO

△ Tree



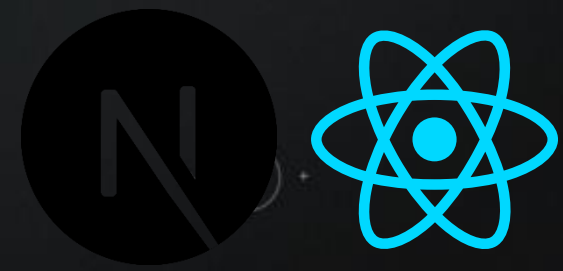


NEXT.JS

FIAP

FUNDAMENTOS DE ROTEAMENTO

- △ **Árvore:** Uma convenção para visualizar uma estrutura hierárquica. Por exemplo, uma árvore componente com componentes pais e filhos, uma estrutura de pastas, etc.
- △ **Subárvore:** Parte de uma árvore, começando em uma nova raiz (primeira) e terminando nas folhas (última).
- △ **Raiz :** O primeiro nó em uma árvore ou subárvore, como um layout de raiz.
- △ **Folha:** Nodes em uma subárvore que não têm filhos, como o último segmento em um caminho de URL.
- △ **Rota Dinâmica:** Uma rota que é criada para receber um parâmetro.

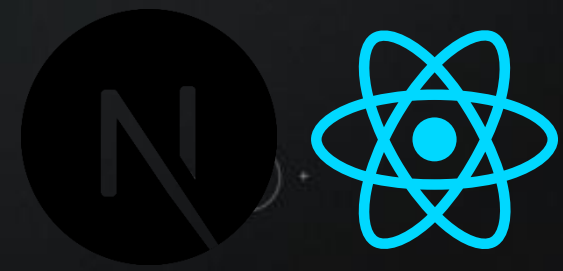


NEXT.JS

FIAP

FUNDAMENTOS DE ROTEAMENTO

- ❖ Uma funcionalidade bem legal do Next é como ele trata o roteamento das páginas. Não é preciso realizar nenhum tipo de configuração ou alguma biblioteca para fazer o tratamento de rotas.
- ❖ Aqui basta criar uma pasta dentro da pasta “app” e dentro desta pasta, você cria o arquivo `pages` e o próprio Next.js vai entender que aquele arquivo “`.jsx`” é uma rota acessível. Esse comportamento torna a criação e roteamento das páginas muito mais facilmente.
- ❖ Ao criar um arquivo dentro da pasta “app”, o Next.js automaticamente irá assumir que o nome daquele arquivo é um endereço acessível da sua aplicação.

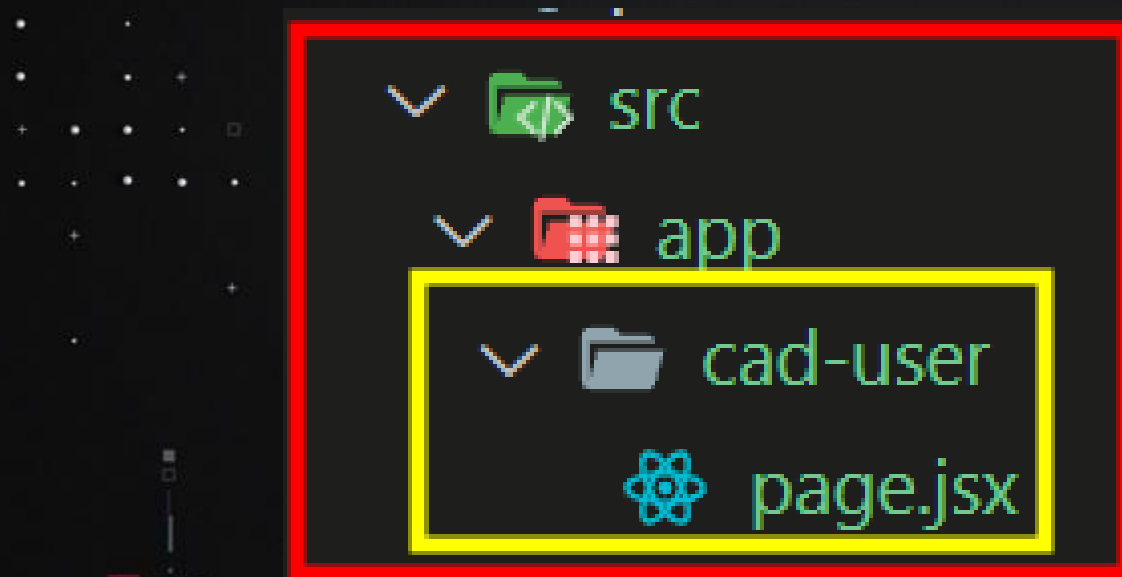


NEXT.JS

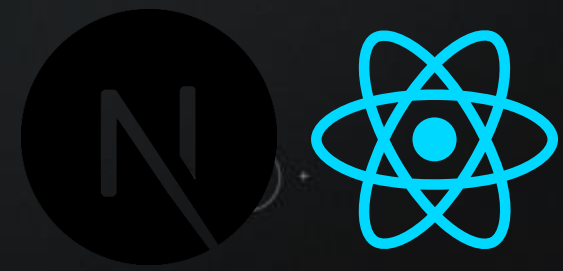
FIAP

FUNDAMENTOS DE ROTEAMENTO

- ❖ Veja nesse exemplo que criamos a pasta ***cad-user*** e dentro dela criamos o arquivo ***page.jsx***, então a rota vai ser: ***/cad-user***



- ❖ O importante é perceber que o arquivo ***page.jsx*** não pode aparecer na chamada deste segmento.



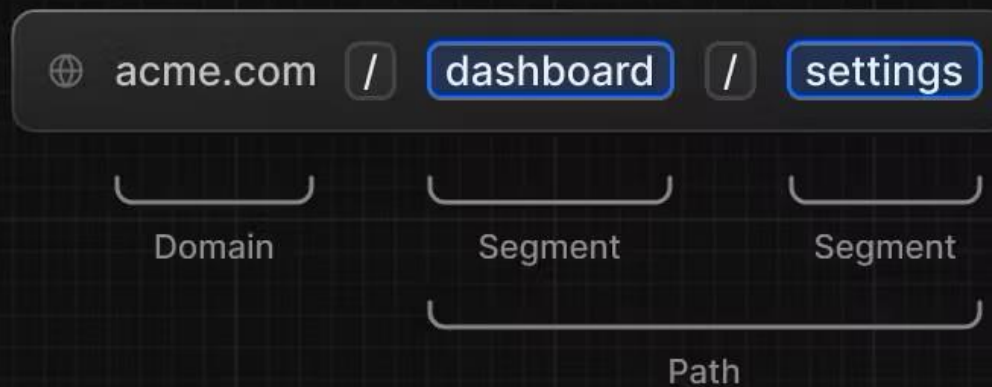
NEXT.JS

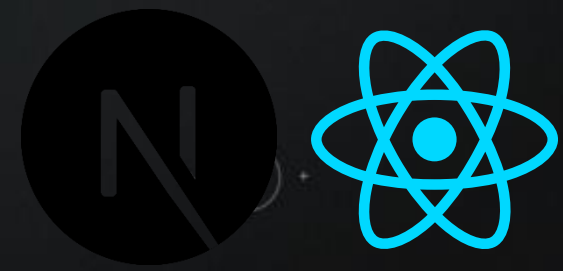
FIAP

FUNDAMENTOS DE ROTEAMENTO

△ **Segmento de URL:** parte do caminho de URL delimitado por barras.

△ **Caminho de URL:** Parte do URL que vem após o domínio (composto por segmentos).





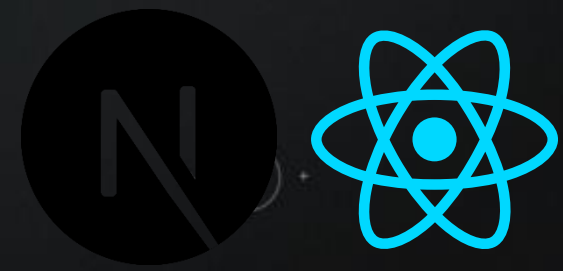
NEXT.JS

FIAAP

ROTAS E ESTRUTURA DE PASTAS

- △ O NEXT trabalha as rotas dentro da pasta app, todo o esqueleto da aplicação é montado ali, por isso o conceito de esqueleto e árvore são tão utilizados na documentação.
- △ **app**: esta pasta contém os arquivos especiais do NEXT, layout e page, estes arquivos com características globais são utilizados para passar as configurações padrão a todas as rotas/páginas criadas, ou seja, a pasta principal da aplicação.


O page faz as vezes de um página inicial, como uma HOME/index e o layout faria o papel do App, trazendo por exemplo um cabeçalho e rodapé, mas isso pode ser sobreposto pelas páginas criadas localmente, sem a necessidade de nenhuma configuração extra. Basta criar sua pasta, colocar seu arquivo page e layout próprio lá.





NEXT.JS

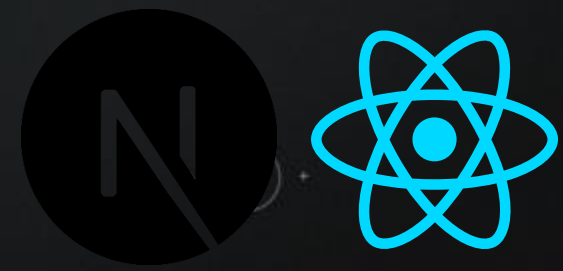
FIAAP

ESTRUTURA DE PASTAS

>  **public**: esta pasta contém arquivos estáticos que podem ser servidos diretamente, como imagens, fontes e outros recursos.

>  **components**: esta pasta é opcional e contém componentes de interface do usuário (UI) reutilizáveis que podem ser usados em todo o aplicativo.

>  **styles**: esta pasta também é opcional e contém estilos globais que podem ser aplicados em todo o aplicativo.



NEXT.JS

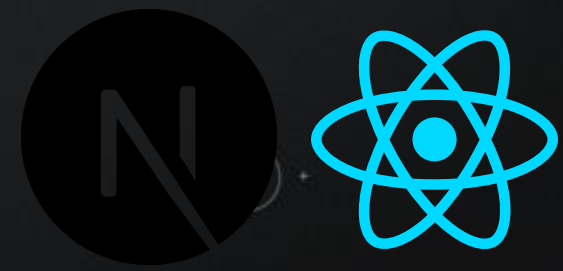
FIAP

ROTAS E ESTRUTURA DE PASTAS

△ Importante saber que:

Por padrão componentes criados dentro da pasta **app**, são considerados React Server Components

△ Esta é uma otimização de desempenho e permite adotá-los facilmente, e você também pode usar os React Client Components.



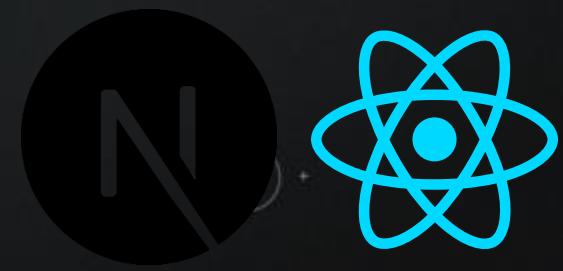
NEXT.JS

FIAP

ROTAS E ESTRUTURA DE PASTAS

△ Importante saber que:

△ Existem otimização de muitos aspectos em relação a processamento no NEXT.js, mas nenhum é mais sentido pelos Devs do que o elemento ``, pois a mudança na utilização de um elemento para fins de ganho de performance, pode parecer um pouco radical, mas isso é explicado em detalhes na documentação e eles tem boas razões para isso: Então ao invés de utilizarmos a tag `` devemos utilizar o componente `<Image/>` que é importado e possui 4 props de preenchimento obrigatórios, segue exemplo:



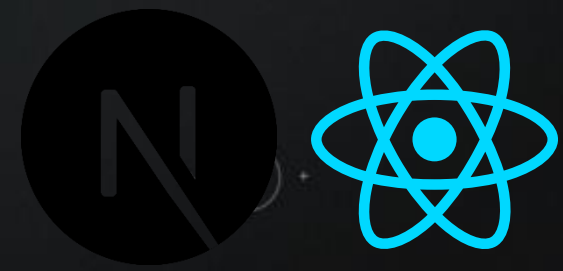
NEXT.JS

FIAP

EXEMPLO / IMAGE COMPONENT

```
import Image from 'next/image'

<figure>
  <Image src="/images/nextjs.png"
    width={500}
    height={500}
    alt="NextJS" />
  <figcaption>NextJS</figcaption>
</figure>
```

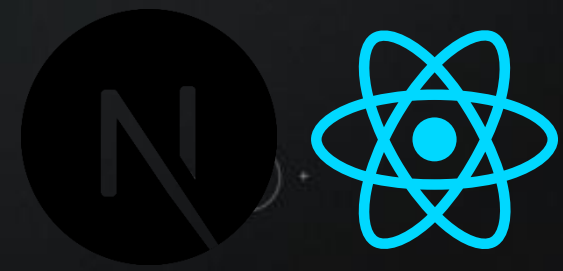


NEXT.JS

FIAP

ROTAS E ESTRUTURA DE PASTAS

- △ Importante saber que:
- △ Além da otimização o framework se interpõem como um firewall para controle das imagens:
- △ Para podermos adicionar imagens externas, ou seja, através de links, devemos adicionar alguns dados desta imagem no arquivo next-config.js.



NEXT.JS

FIAP

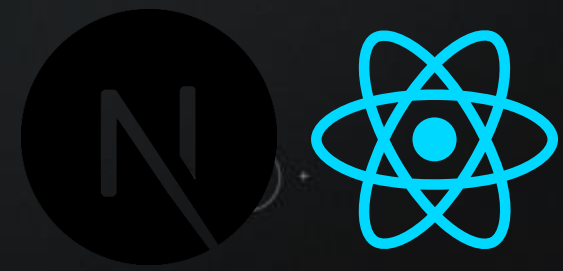
ROTAS E ESTRUTURA DE PASTAS

remotePatterns

Para proteger seu aplicativo de usuários mal-intencionados, é necessária configuração para usar imagens externas. Isso garante que apenas imagens externas da sua conta possam ser veiculadas na API Next.js Image Optimization. Essas imagens externas podem ser configuradas com a `remotePatterns` propriedade do seu `next.config.js` arquivo, conforme mostrado abaixo:

JS next.config.js

```
module.exports = {
  images: {
    remotePatterns: [
      {
        protocol: 'https',
        hostname: 'example.com',
        port: '',
        pathname: '/account123/**',
      },
    ],
  },
}
```



NEXT.JS

FIAP

ROTAS E ESTRUTURA DE PASTAS

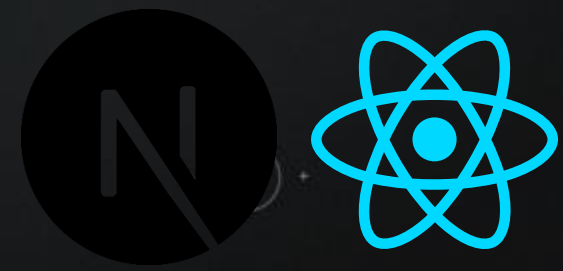
△ Importante saber que:

△ Complementando as rotas temos o componente Link. Anteriormente este componente era utilizado pelo react-router-dom, internamente ainda é, porém com a otimização do NEXT, ele está sendo importado por outro pacote agora e sofreu uma alteração na utilização.

△ A propriedade que recebia o valor de destino era a to, agora é href. Veja o exemplo:

```
import Link from 'next/link'

<Link href="/posts">POST</Link>
```



NEXT.JS

FIAP

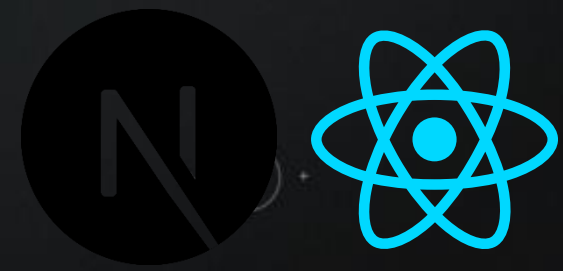
ESTRUTURA DO PROJETO NEXT.JS

△ Para saber mais sobre:

- △ Arquivos de Alto Nível
- △ Convenção de Roteamento
- △ Arquivos de roteamento
- △ Rotas aninhadas
- △ Rotas dinâmicas
- △ Grupos de rotas e pastas privadas
- △ Rotas Paralelas e Interceptadas
- △ Convenções de Arquivo de Metadados
- △ Open Graph e Twitter Imagens

❖ Acesse:

△ <https://nextjs.org/docs/getting-started/project-structure>

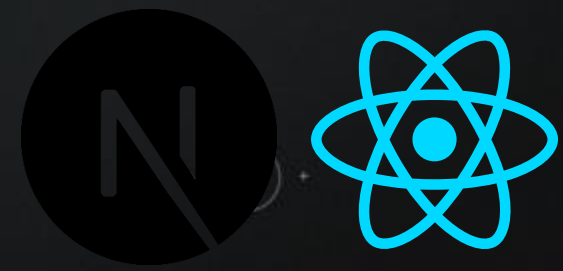


NEXT.JS

FIAP

EXERCÍCIOS / ROTAS

- △ Crie uma pasta chamada **posts** e adicione a sua respectiva página, coloque um título, mude o nome do componente e adicione algum Lorem.
- △ Adicione 03(três), subrotas a rota criada anteriormente, coloque os nomes de:
 - post2, post3 e post4
- △ Crie links, utilizando os elementos Link do NEXT a partir da página Home para cada um dos posts.
- △ Crie links, utilizando os elementos Link do NEXT a partir dos posts para a página Home.
- △ Adicione uma imagem estática dentro de cada post, utilizando o componente Imagem do NEXT.

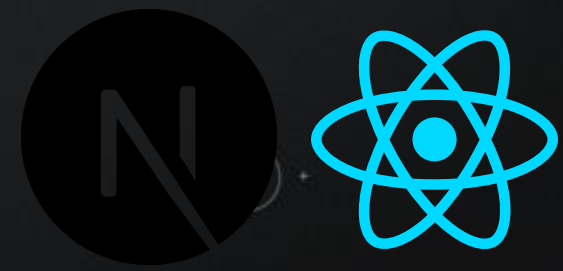


NEXT.JS

FIAP

LAYOUTS

- △ No Next.js, layouts não são um conceito oficial, como em alguns outros frameworks, mas são uma abordagem de organização de componentes que permite criar uma estrutura de página consistente em todo o seu aplicativo. Layouts são componentes React personalizados que envolvem o conteúdo das páginas e fornecem uma maneira de definir a estrutura e o estilo da página de forma uniforme.
- △ A ideia básica é criar componentes de layout que contenham elementos de estrutura comuns a várias páginas do seu aplicativo, como cabeçalhos, rodapés, barras de navegação ou menus laterais. Em seguida, você pode incorporar esses componentes de layout nas páginas específicas do seu aplicativo, permitindo que você reutilize a estrutura em várias partes do site.



NEXT.JS

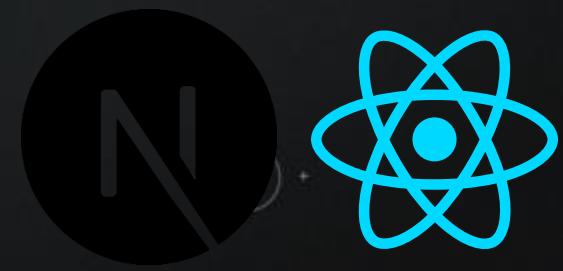
FLIAP

LAYOUTS / Exemplo

```
import Header from './Header';
import Footer from './Footer';

const Layout = ({ children }) => (
  <div>
    <Header />
    {children}
    <Footer />
  </div>
);

export default Layout;
```

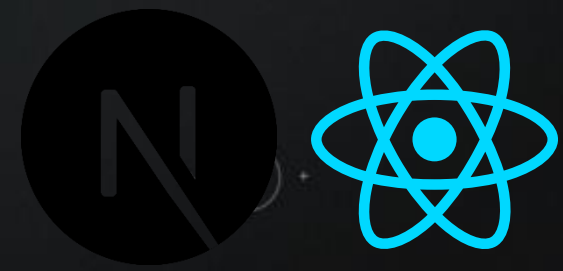


NEXT.JS

FIAP

LAYOUTS

- △ No exemplo anterior foi possível ver que o layout definido é injetado em todas as nossas páginas pelo React. O arquivo layout é um superarquivo e isso faz com que o compilador reconheça e utilize a desestruturação única nos elementos para aplicar o layout padrão. Ou seja um cabeçalho e rodapé.
- △ Mas veja que o cabeçalho e rodapé não estão ali na pasta app, são componentes importados que estão no layout, envolvendo o conteúdo injetado. Esta é uma das técnicas utilizadas com o layout no NEXT.js

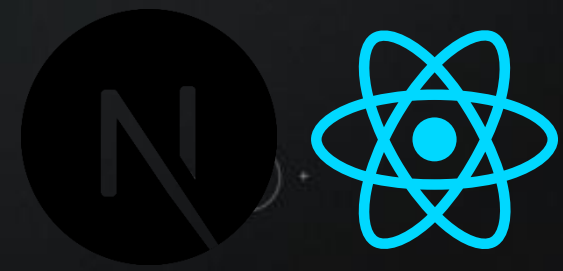


NEXT.JS

FIAP

EXERCÍCIO / LAYOUTS

- △ Dentro da pasta /src, crie uma pasta chamada components.
- △ Utilizando as técnicas aprendidas até o momento, crie dois componentes para representar o Cabeçalho e Rodapé.
- △ Utilize boas práticas de nomenclatura e organização na criação.
- △ Adicione ao Cabeçalho um título como logo e uma lista com 4 links fakes.
- △ O Cabeçalho deve estar estilizado, utilize por enquanto o SASS.
- △ Faça o mesmo para o rodapé.
- △ No rodapé adicione informações de contato e endereço.

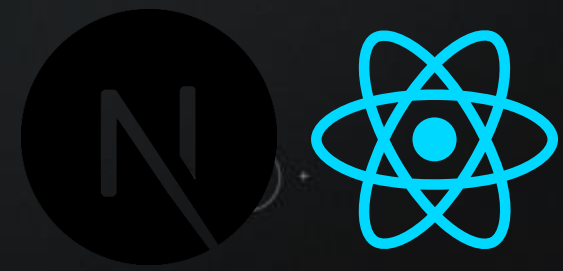


NEXT.JS

FIAP

LAYOUTS/COMPOSTOS

- △ Com o NEXT.js nos temos o poder do root-layout, onde ele se propaga para todas as páginas do projeto.
- △ Mas é certo que nem sempre vamos utilizar apenas o layout principal, em algum momento, vamos precisar de um layout local, mais focado ou único.
- △ Para isso o NEXT.js tem a solução, você só tem que criar um novo arquivo layout dentro da pasta da página(s) que vão receber este layout e esta(s), vai receber todo o estilo do seu layout local e do layout root.
- △ O único cuidado que temos que ter é o de não adicionar aos layouts locais as seguintes tags pois estas não podem se repetir: `<html>` e `<body>`.



NEXT.JS

FIAP

LAYOUTS/COMPOSTOS

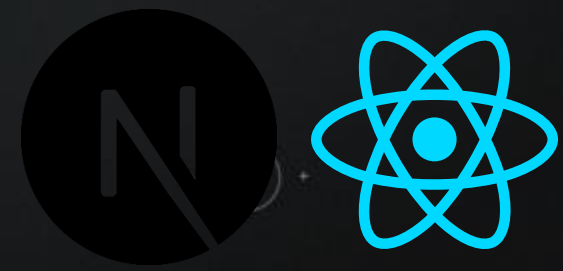
△ Exemplos:.

LAYOUT ROOT

```
> app > layout.jsx > metadata
1 import Cabecalho from '@components/Cab/Cabecalho'
2 import Rodape from '@components/Rod/Rodape'
+ 3 import './base.scss'
4 import { Inter } from 'next/font/google'
5
6 const inter = Inter({ subsets: ['latin'] })
7
8 export const metadata = {
9   title: 'Create',
10  description: 'Generated by create next app',
11 }
12
13 export default function RootLayout({ children }) {
14   return (
15     <html lang="en">
16       <body className={inter.className}>
17         <Cabecalho/>
18         {children}
19         <Rodape/>
20       </body>
21     </html>
22   )
23 }
24
25
```

LAYOUT LOCAL

```
src > app > posts > layout.jsx > PostLayout
1
2
3 export default function PostLayout({ children }) {
4   return (
5     <div>
6       <h1>BELOS POSTS</h1>
7       {children}
8     </div>
9   )
10 }
11
```

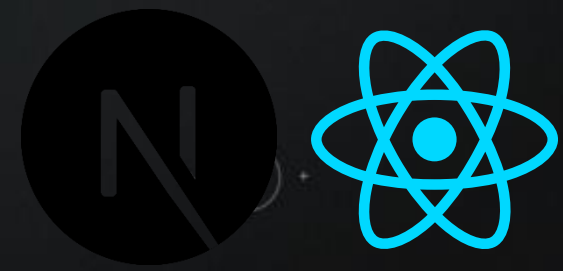


NEXT.JS

FIAAP

ROTAS DINÂMICAS

- △ Já trabalhamos com rotas básicas onde é possível realizar direcionamentos e carregar páginas, agora vamos começar a passar parâmetros.
- △ As rotas dinâmicas são construídas de forma a passar o parâmetro direto dentro do componente que vai receber os dados, de uma forma simples e direta.
- △ Basta criarmos a estrutura e definirmos qual vai ser a rota parametrizada e:
 - 1 - Definir o nome do parâmetro, isso é importante, porque este objeto na rota vai ser criado como uma pasta entre colchetes. Ex: mosfet/**[slug]**/page.jsx.
 - 2 - Veja que no exemplo acima a rota é criada temos o segmento mosfet iniciando, o parâmetro [slug] e o componente que de fato vai carregar e dar destino ao parâmetro page.jsx. Vamos ver um componente criado:



NEXT.JS

FIAP

ROTAS DINÂMICAS / EXEMPLO

> node_modules

> public

✓ src

✓ app

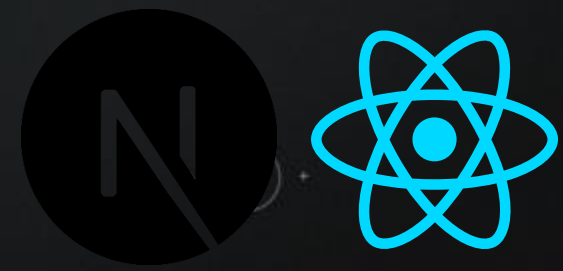
✓ mosfet

✓ [slug]

🌀 page.jsx U

```
2 export default function Produto({params}) {
3   return (
4     <div>
5       <h1>PRODUTOS</h1>
6       <h2>Parâmetro recuperado sempre em formato de String:{params.slug}</h2>
7     </div>
8   )
}
```

ATENÇÃO
Nesta rota dinâmica utilizamos o nome slug, mas é apenas um nome como qualquer outro.

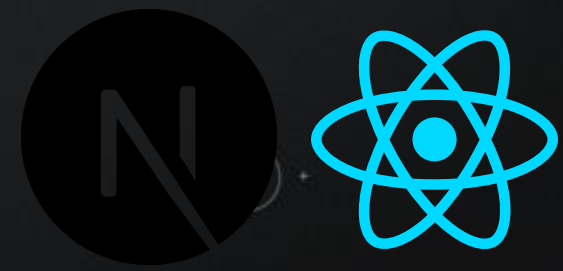


NEXT.JS

FIAP

EXERCÍCIO / ROTAS DINÂMICAS

- △ Crie uma rota chamada produtos.
- △ Adicione um produto de nome item.(Faça o teste com os seguintes parâmetros[camisa, calça, sapato, boné e óculos).
- △ Crie um único card.
- △ Na página quando receber o parâmetro, dependendo de qual parâmetro chegar, carregue neste card a imagem referente ao produto passado.

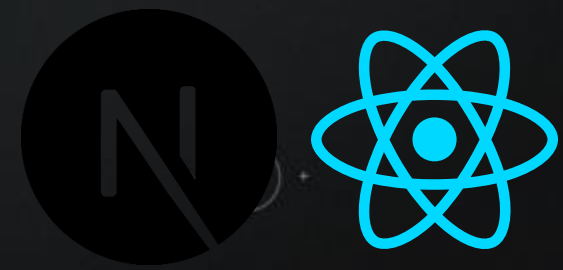


NEXT.JS

FIAPI

ROTAS AGRUPADAS

- △ No segmentos das Rotas podemos nos deparar com algumas situações incomodas.
- △ Por exemplo: Criei o seguimento `/produtos/slug/calca`, `/produtos/slug/camisa`, mas não gostaria que o seguimento (slug), aparecesse, como posso fazer.
- △ Você pode agrupar as ROTAS.
 - `/produtos/(slug)/calca` e ela vai aparecer assim `/produtos/calca`, para todos os segmentos.
- △ É a melhor forma de utilizarmos nossos arquivos de suporte sem a necessidade de expor os segmentos que os acompanham.



NEXT.JS

FIAP

ROTAS AGRUPADAS

✓ rageagain

✓ slug

> cd1

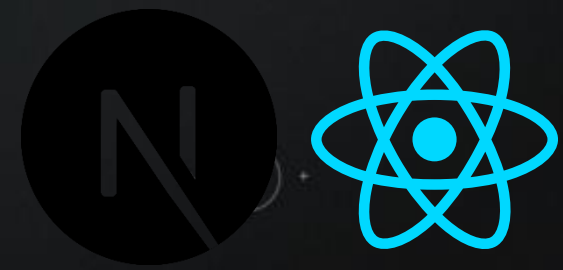
> cd2

> cd3

localhost:3000/rageagain/slug/cd1

Cabecalho

CD1



NEXT.JS

FIAP

ROTAS AGRUPADAS

✓ rageagain

✓ (slug)

> cd1

> cd2

> cd3

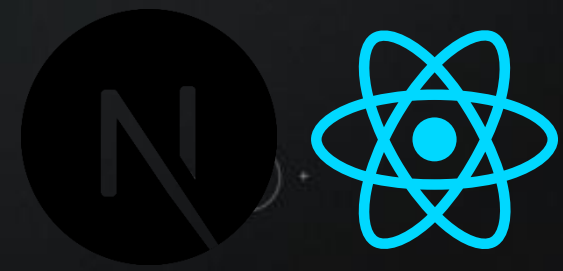
localhost:3000/rageagain/slug/cd1

130%

Cabecalho

404

This page
could not be



NEXT.JS

FIAP

ROTAS AGRUPADAS

✓ rageagain

✓ (slug)

> cd1

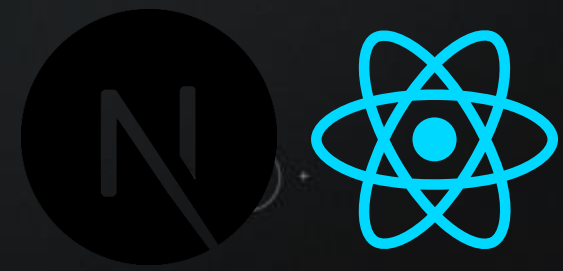
> cd2

> cd3

localhost:3000/rageagain/cd1

Cabecall

CD1

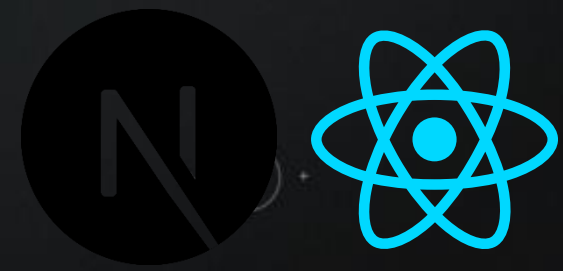


NEXT.JS

FIAP

PÁGINA 404

- △ Como estamos utilizando um framework de baseado em front-end, fatalmente vamos trabalhar e muito com requisições e é comum trabalharmos com todo o tipo de STATUS HTTP:
- △ O mais famoso entre estes é o 404 - NOT-FOUND, de conteúdo ou página não encontrada.
- △ O NEXT.js lida com esse status de uma forma muito natural, se o usuário não configurar o arquivo not-found dentro da pasta/app, ele assumi que deve apresentar suas páginas de erro padrão, agora caso o usuário crie o arquivo e coloque conteúdo, este arquivo passa a ser carregado quando esses erros ocorrem. Vamos ver um exemplo:

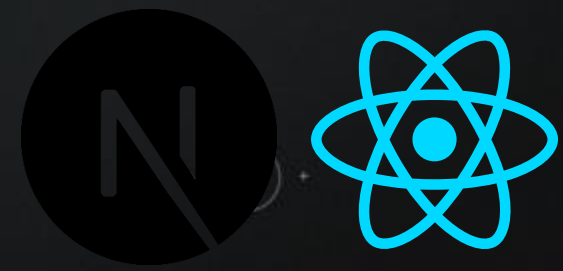


NEXT.JS

FIAP

PÁGINA LOADING

- △ Como estamos utilizando um framework de baseado em front-end, fatalmente vamos trabalhar e muito com requisições e é comum trabalharmos com todo o tipo de STATUS HTTP:
- △ O mais famoso entre estes é o 404 - NOT-FOUND, de conteúdo ou página não encontrada.
- △ O NEXT.js lida com esse status de uma forma muito natural, se o usuário não configurar o arquivo not-found dentro da pasta/app, ele assumi que deve apresentar suas páginas de erro padrão, agora caso o usuário crie o arquivo e coloque conteúdo, este arquivo passa a ser carregado quando esses erros ocorrem. Vamos ver um exemplo:

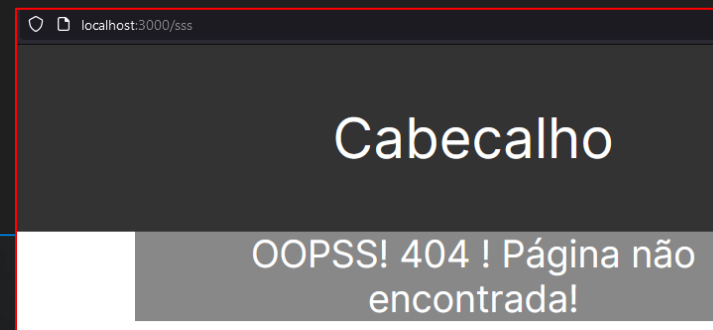


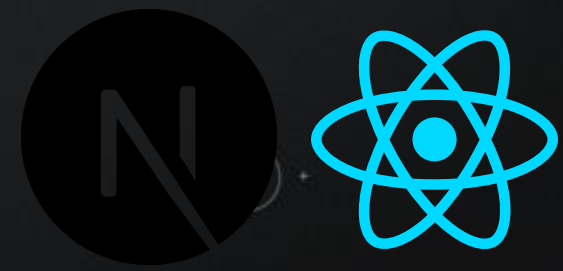
NEXT.JS

FIAP

PÁGINA 404 - NotFound

```
src > app > not-found.jsx > ...  
1  
2   export default function NotFound() {  
3     return (  
4       <div>  
5         OOPSS! 404 ! Página não encontrada!  
6       </div>  
7     )  
8   }
```



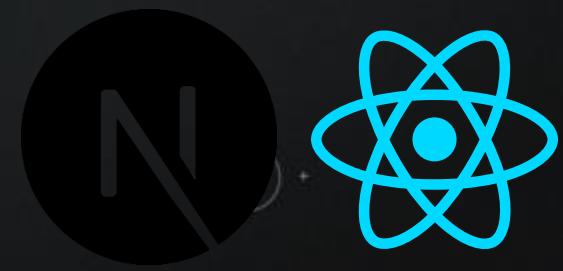


NEXT.JS

FIAP

EXERCÍCIOS / NOT-FOUND + LOADING

- △ 1 – Adicione uma página de erro 404 e adicione uma imagem externa(com link) para representar este STATUS.
- △ 2 – Adicione uma página Loading, faça com que esta apresente o texto (Loading...).
- △ 3 – Ambos os arquivos tenham estilização em SASS.



NEXT.JS

FIAP

NAVEGAÇÃO / REDIRECT

- △ Durante o processamento é possível redirecionar a requisição do usuário para onde quisermos.
- △ Para isso podemos importar a directiva “redirect” e utilizar ela como no exemplo:

Parameters

The `redirect` function accepts two arguments:

```
redirect(path, type)
```

app/team/[id]/page.js

```
import { redirect } from 'next/navigation'

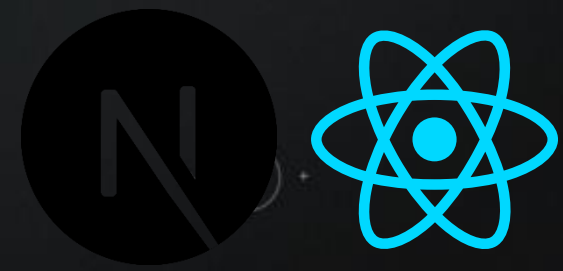
async function fetchTeam(id) {
  const res = await fetch('https://...')
  if (!res.ok) return undefined
  return res.json()
}

export default async function Profile({ params }) {
  const team = await fetchTeam(params.id)
  if (!team) {
    redirect('/login')
  }

  // ...
}
```

ATENÇÃO

Nesta função assíncrona com fetch se verifica o resultado do status da requisição. Caso o resultado seja false a função retorna undefined, caso contrário retorna o json. E podemos avaliar a função, se ela não retornar o esperado, podemos redirecionar a requisição para uma rota específica.



NEXT.JS

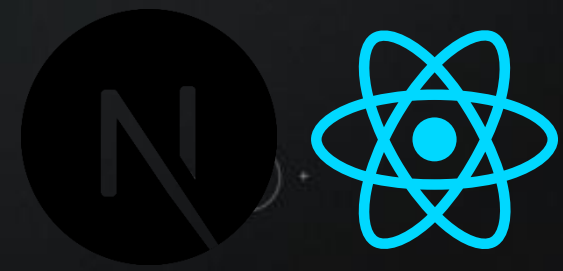
FIAP

Route Handlers / APIs LOCALS

△ Complementando a manipulação de rotas no NEXT.js, apresentamos o RouteHandler, semelhante ao Json-Server, este artefato tem o poder de criar rotas que são arquivos no NEXT.js, mas que podem conter toda a lógica de manipulação não apenas de uma API, seja ela, interna ou externa, local ou remota. Você pode escolher qual a melhor estratégia de gerenciamento destes dados.



- △ Para podermos criar uma API-LOCAL por exemplo devemos criar a seguinte estrutura:
- △ Uma pasta de qualquer nome dentro do diretório */app*.
- △ Dentro do pasta que foi criada, ou você adiciona uma nova pasta que também não é obrigatório, ou já pode criar o arquivo ***route.jsx*** - **Este nome de arquivo é obrigatório.**



NEXT.JS

FIAP

Convenções / APIs LOCAIS

Convenção

Os manipuladores de rota são definidos em um arquivo de nome

`route.js/jsx/tsx` dentro de um outro diretório de qualquer nome dentro de `app`

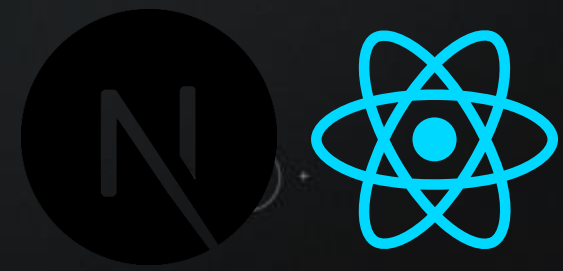
`JS` app/api/route.js

JavaScript



```
export async function GET(request) {}
```

Os manipuladores de rota podem ser aninhados dentro diretório `app`, semelhante aos arquivos `page.js/jsx/tsx` e ao `layout.js/jsx/tsx` mas **não pode** haver um arquivo `route.js/jsx/tsx` no mesmo nível destes mesmos segmentos.



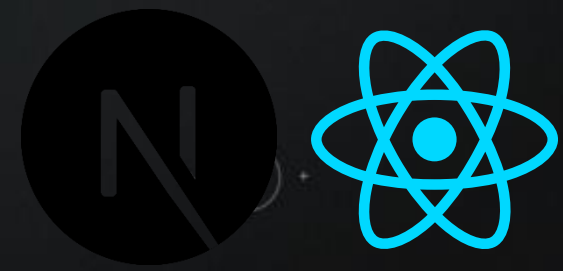
NEXT.JS

FIAP

Route Handlers / APIs LOCAIS

△ Os manipuladores de rota permitem que você crie manipuladores de solicitações personalizados para uma determinada rota usando Request e Response APIs.





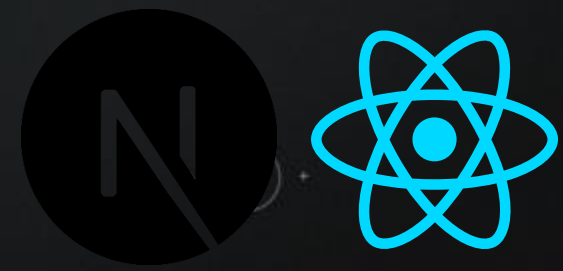
Métodos HTTP Suportados / APIs LOCAIS

Métodos HTTP suportados

Os seguintes [métodos HTTP](#) são suportados: `GET`, `POST`, `PUT`, `PATCH`, `DELETE`, `HEAD`, e `OPTIONS`. Se um método não suportado for chamado, Next.js retornará um `405 Method Not Allowed` response.

Estendido `NextRequest` E a `NextResponse` APIs

Além de suportar a [Request](#) nativa e a [Response](#). Next.js estende-os com `NextRequest` E a `NextResponse` para fornecer helpers convenientes para casos de uso avançados.



NEXT.JS

FIAP

Route Handlers / APIs LOCAIS

△ Depois de criar o arquivo **route.jsx**, você deve criar sua estratégia de dados nele que pode ser diversa: Ex:

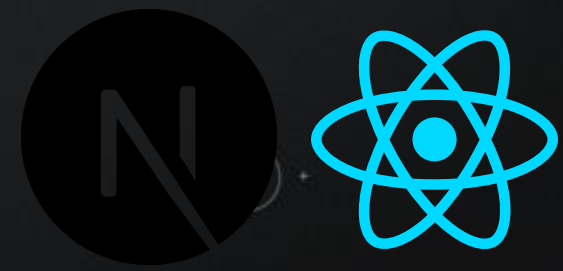
△ Tradicionalmente criamos em formato de função os métodos do CRUD/HTTP. GET/POST/PUT/DELETE.

△ GET:

```
export async function GET(request, { params }) {  
    return NextResponse.json(dados);  
}
```



Esta função já sabe que qualquer requisição realizada para nosso servidor na url:
Localhost:3000/dados



NEXT.JS

FIAP

Comportamento : GET / APIs LOCAIS

Caching

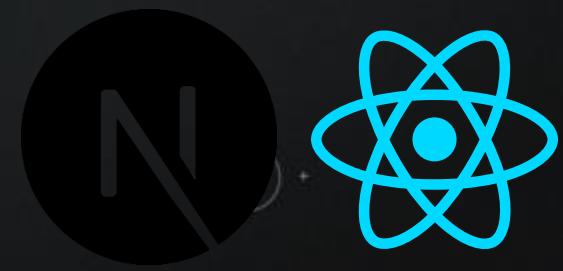
Os manipuladores de rota são armazenados em cache por padrão ao usar o `GET`. O método com o `Response` object.

Js app/items/route.js

JavaScript

```
export async function GET() {
  const res = await fetch('https://data.mongodb-api.com/...', {
    headers: {
      'Content-Type': 'application/json',
      'API-Key': process.env.DATA_API_KEY,
    },
  })
  const data = await res.json()

  return Response.json({ data })
}
```

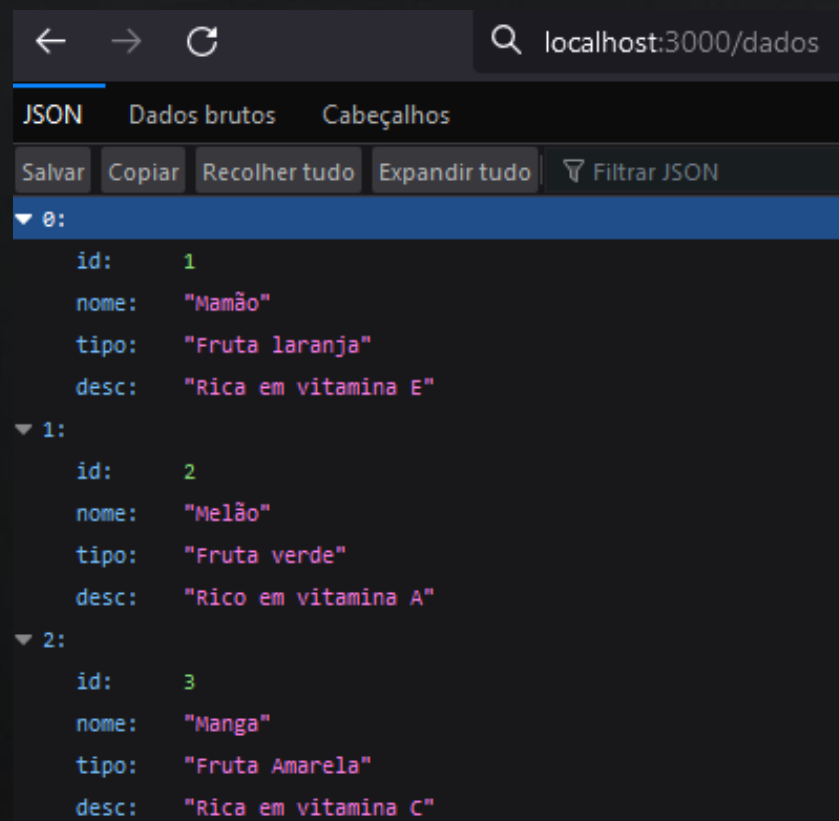


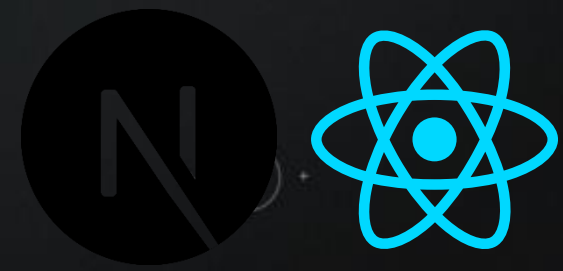
NEXT.JS

FIAP

Route Handlers / APIs LOCAIS

△ Depois de acessar a url vamos ter o seguinte resultado:



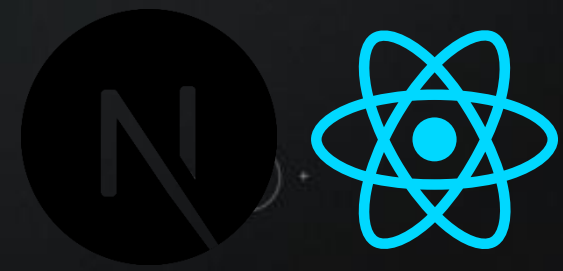


NEXT.JS

FIAP

Route Handlers / APIs LOCAIS

- △ Veja que foi retornado todos os dados de uma lista de objetos, porém caso você queira que seja retornado por exemplo apenas um destes, deve criar uma estratégia para manipular os dados de forma que quando o parâmetro for passado na url, o objeto correto seja retornado na solicitação.
- △ Devemos utilizar a mesma estratégia das rotas dinâmicas [parametro]. Ou seja.
- △ Devemos criar a estrutura de nossas APIs dessa forma:
 - ***/dados/produtos-api/[id]/route.jsx***
- △ Veja que acima foi adicionada uma pasta com [colchetes], que indica parâmetros, ou seja, dessa forma poderemos recuperar esse parâmetro dentro de nossas APIs local.



NEXT.JS

FIAP

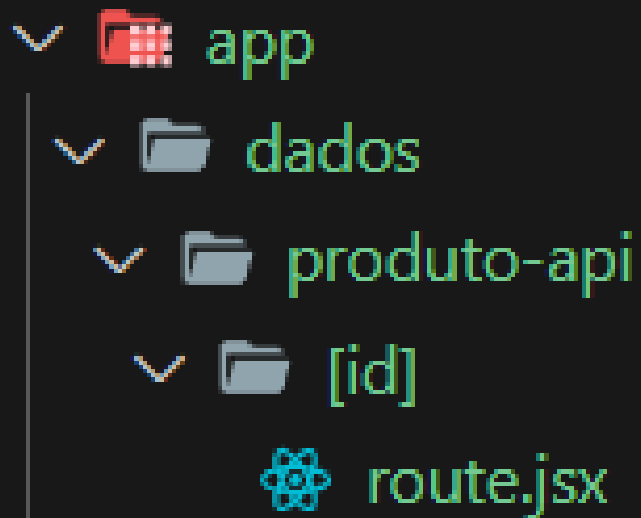
Route Handlers / APIs LOCAIS

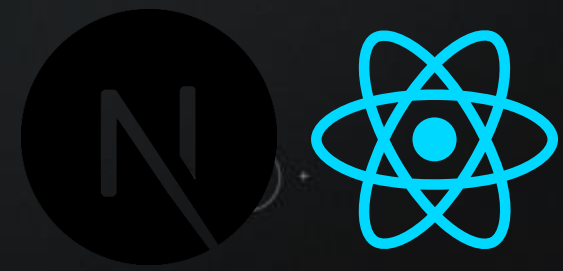
△ Veja um exemplo de como ficaria :

app > dados > produto-api > [id] > route.jsx > GET
`import { NextResponse } from "next/server";`

```
const produtos = [
  { id: 1, nome: "Mamão", tipo: "Fruta laranja", desc: "Rica em vitamina E" },
  { id: 2, nome: "Melão", tipo: "Fruta verde", desc: "Rico em vitamina A" },
  { id: 3, nome: "Manga", tipo: "Fruta Amarela", desc: "Rica em vitamina C" },
];
```

```
export async function GET(request, { params }) {
  //Pegando o id da rota
  const id = params.id;
  if (id > 0) {
    const produto = produtos.find((produto) => produto.id == id);
    //Retornando o produto encontrado
    console.log(produto);
    return NextResponse.json(produto);
  } else {
    //Retornando todos os produtos
    return NextResponse.json(produtos);
  }
}
```





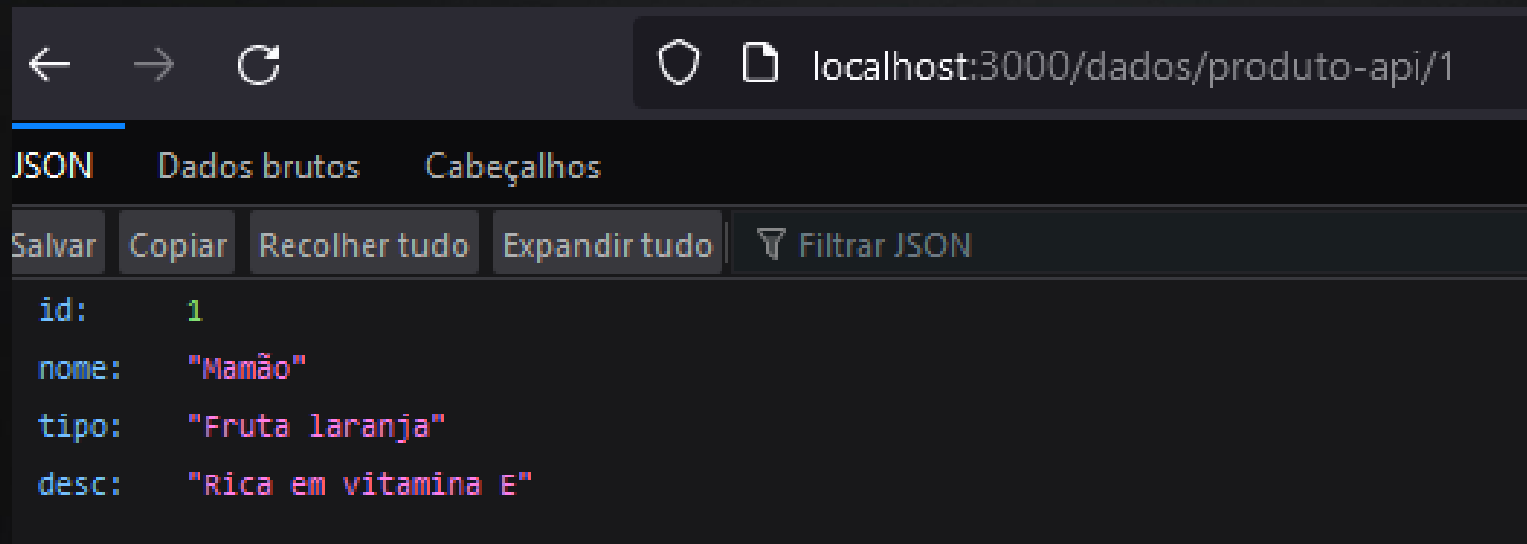
NEXT.JS

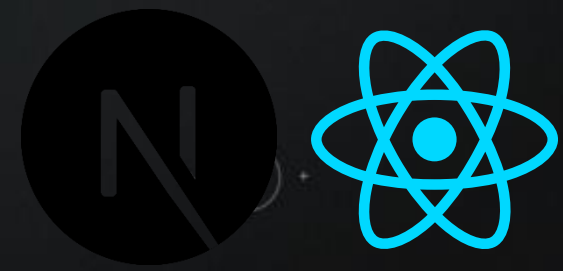
FIAP

Route Handlers / APIs LOCAIS

△ Neste último exemplo foi criado um exemplo de estratégia da seguinte forma:

△ 1 - Se a requisição for realizada passando um valor de `id = 0`, será retornada toda a lista de produtos, caso contrário será retornado somente o produto referente o `id` que foi passado.



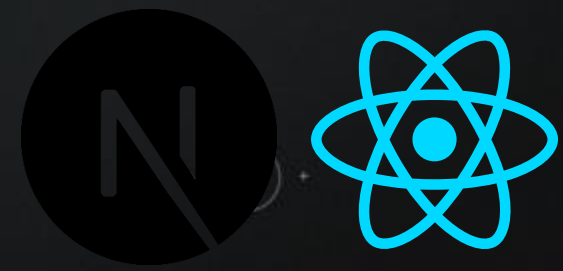


NEXT.JS

FIAP

Route Handlers / APIs LOCAIS

- △ Devemos prestar atenção nos parâmetros que o método GET de route.jsx recebe.
- △ request - Este é o objeto da requisição que carrega muitas informações sobre a solicitação realizada, por exemplo a url de onde partiu.
- △ {params} - Um objeto que chega com todos os parâmetros passados. No caso de request realizados com parâmetros: Ex - /dados/produtos-api/1, podemos recuperar o id, que tem o valor de 1, que vai chegar em {params}, utilizando a terminologia aplicada aos objetos.
- △ params.id
- △ Agora você pode reutilizar este parâmetro para suas estratégias.



NEXT.JS

FIAP

Exemplos Consumo GET Router Handles

OPEN EDITORS

LOJA-APP

- produtos
- (processame...
- [id]
 - page.jsx M

src > app > produtos > (processamento) > [id] > page.jsx > ProdutoID

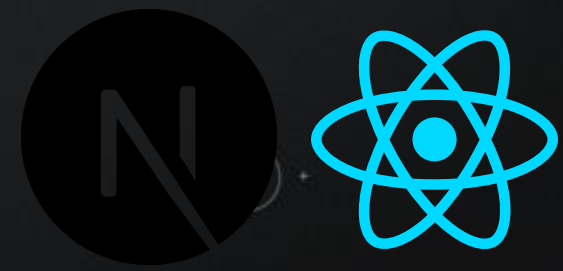
You, 28 seconds ago | 1 author (You)

```
1
2 export default async function ProdutoID({params}) {
3
4   const response = await fetch(`http://localhost:3000/dados/produto-api/${params.id}`);
5   const produto = await response.json();
6
7   return (
8     <div>
9       <h1>Identificação das FRUTAS</h1>
10      <p>Esta página é um exemplo de como receber os parâmetros da api e apresentar um
11      produto selecionado dela:</p>
12      <p>Valor do ID : {params.id}</p>
13      <p>Nome do produto: {produto.nome}</p>
14      <p>Tipo do produto: {produto.tipo}</p>
15      <p>Descrição do produto: {produto.desc}</p>
16    </div>
17  )
18 }
19
```

localhost:3000/produtos/1

HOME NEW MAMÃO

Identificação das FRUTAS
Esta página é um exemplo de como receber os parâmetros da api e apresentar um produto selecionado dela:
Valor do ID : 1
Nome do produto: Mamão
Tipo do produto: Fruta laranja
Descrição do produto: Rica em vitamina E

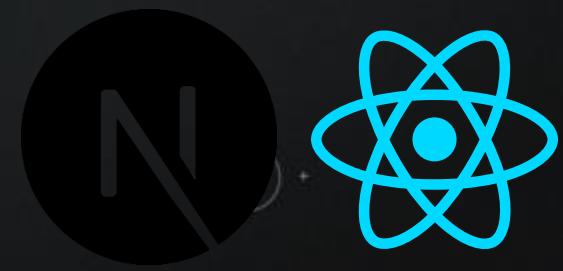


NEXT.JS

FIAP

Route Handlers / APIs LOCAIS

- △ Outro ponto de atenção é que estamos utilizando em nossas funções a marcação de ***async***. Mas porque isso?
- △ Em JavaScript, o conceito "async" se refere a operações assíncronas, ou seja, tarefas que não bloqueiam a execução do código principal. Isso é particularmente útil quando se trabalha com operações demoradas, como a leitura de arquivos, solicitações de rede ou acesso a bancos de dados, pois permite que o código continue a ser executado enquanto a operação assíncrona está em andamento. Isso evita que a interface do usuário trave e melhora a eficiência geral do programa.
- △ Para trabalhar com funções assíncronas em JavaScript, você pode usar a palavra-chave **async** antes de uma função. Por exemplo:



NEXT.JS

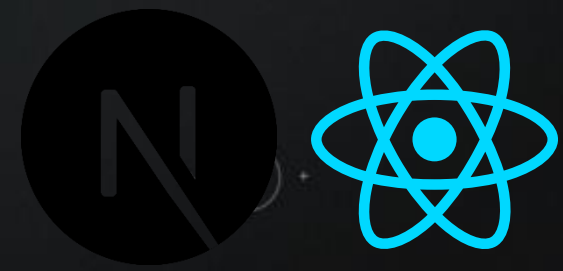
FIAP

Route Handlers / APIs LOCAIS

```
async function minhaFuncaoAssincrona() {  
  // Código assíncrono aqui  
}
```

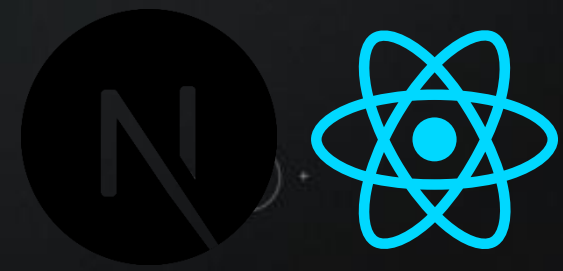
△ Dentro de funções assíncronas, é comum usar a palavra-chave `await` para aguardar a conclusão de uma operação assíncrona. Isso faz com que a função aguarde até que a operação seja concluída antes de continuar. Por exemplo:

```
async function minhaFuncaoAssincrona() {  
  const resultado = await algumaOperacaoAssincrona();  
  // Código a ser executado após a conclusão da operação assíncrona  
}
```



Route Handlers / APIs LOCAIS

- △ É importante observar que apenas funções assíncronas podem usar a palavra-chave `await`. Além disso, as funções assíncronas sempre retornam uma promessa (Promise) que é resolvida com o valor retornado pela função ou rejeitada com qualquer erro lançado.
- △ O uso de funções assíncronas e `await` simplifica o código em comparação com o uso de callbacks ou Promises encadeadas, tornando-o mais legível e fácil de manter. No entanto, é essencial lidar adequadamente com erros em operações assíncronas usando blocos `try...catch` ou tratando as rejeições de Promises para garantir que o programa se comporte corretamente, mesmo em caso de falhas nas operações assíncronas.
- △ **No NEXT.js só poderemos declarar Server Components como `async`, em Client Components, somente em sub-funções poderemos utilizar `async`.**



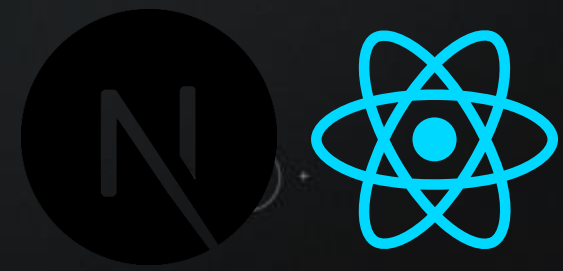
NEXT.JS

FIAP

Route Handlers / APIs LOCAIS

- △ Até agora trabalhamos com o método GET realizando requests para dados locais e se o intuito fosse consumir uma API externa. Vamos ver como seria esse processo.
- △ 1 - Assim como na API-Local, devemos determinar a rota inicial, criando as pastas que vão identificar o contexto de nossos dados.
- △ 2 - Com a rota determinada e criada, vamos criar o arquivo route.jsx e realizar nossa estratégia de recuperação de dados externos.
- △ 3 - Nosso alvo será a api de usuário do Github que fica hospedada no seguinte endereço:

△ URL : <https://api.github.com/users>

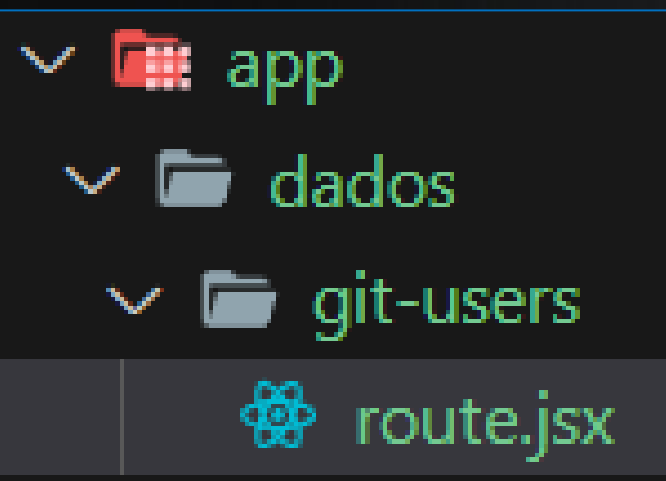


NEXT.JS

FIAP

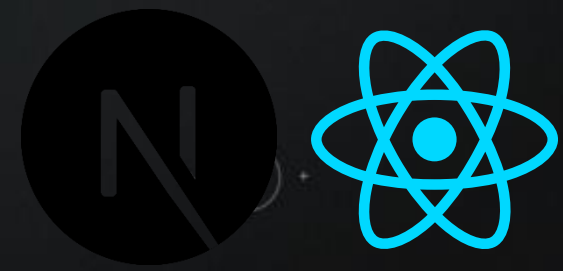
Route Handlers / APIs LOCAIS

△ 4 - Vamos criar a pasta git-users dentro da pasta dados e dentro dela criaremos nosso arquivo ***route.jsx***.



```
src > app > dados > git-users > route.jsx > ...
1   import { NextResponse } from "next/server";
2
3   export async function GET(request, { params }) {
4     const response = await fetch("https://api.github.com/users");
5     const result = await response.json();
6     return NextResponse.json(result);
7   }
```

△ 5 - Assim teremos o segmento : `http://localhost:3000/dados/git-users`, que vai nos devolver todos os dados de forma bruta.



NEXT.JS

FIAP

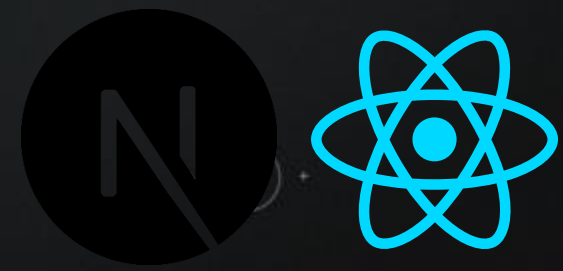
Route Handlers / APIs LOCAIS



△ 6 - Agora vamos criar uma rota de nome git-users para consumirmos os dados desta api que foi gerada em nosso projeto.

△ 7 - E por fim vamos receber os dados da api nessa página.

```
1 import Image from "next/image";
2
3 export default async function GitUsers({ users }) {
4
5
6   const response = await fetch("http://localhost:3000/dados/git-users");
7   const result = await response.json();
8
9   return (
10     <div>
11       <h1>Git Users</h1>
12       <p>Recuperar todos os usuários do Github através da api-interna http://localhost:3000/dados/git-users utilizando fetch. Apresentar apenas os campos login,avatar e url.</p>
13       <div>
14         <ul>
15           {result.map((user) => (
16             <li key={user.id}>
17               <Image src={user.avatar_url} alt={user.login} width={100} height={100}/>
18               <p>{user.login}</p>
19               <a href={user.html_url} target="_blank">Acessar</a>
20             </li>
21           ))}
22         </ul>
23       </div>
24     </div>
25   );
26 }
```



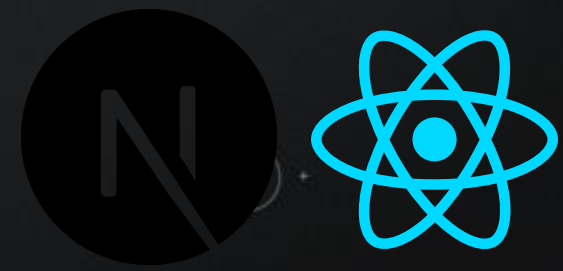
NEXT.JS

FIAP

Route Handlers / APIs LOCAIS

- △ Utilizamos bastante o método GET, agora vamos ver alguns exemplos com o método POST.
- △ Porém agora vamos utilizar um arquivo físico para armazenar nossas informações.
- △ Vamos criar uma pasta de nome base e dentro dela vamos criar um arquivo de nome bd.json.
- △ Vamos adicionar uma lista de produtos ao nosso arquivo de recursos.

```
✓ base  
  { } db.json
```



NEXT.JS

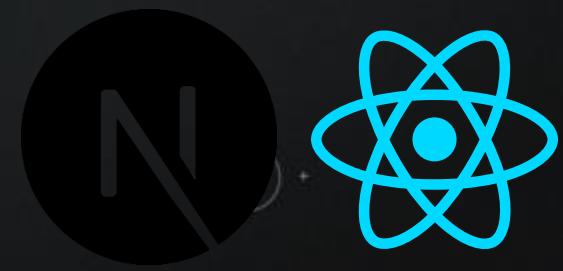
FIAP

Route Handlers / APIs LOCAIS

△ Agora vamos criar uma lista de produtos neste arquivo:

```
src > app > base > {} db.json > ...
```

```
1  {  
2    "produtos": [  
3      {"id": 1, "nome": "Mamão", "tipo": "Fruta laranja", "desc": "Rica em vitamina E"},  
4      {"id": 2, "nome": "Melão", "tipo": "Fruta verde", "desc": "Rico em vitamina A"},  
5      {"id": 3, "nome": "Manga", "tipo": "Fruta Amarela", "desc": "Rica em vitamina C"}  
6    ]  
7  }
```

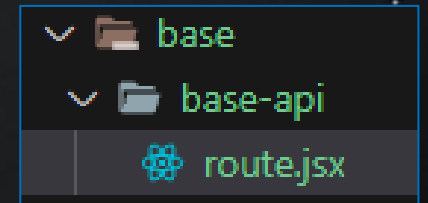


NEXT.JS

FIAP

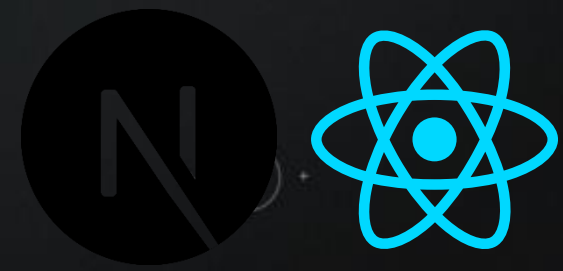
Route Handlers / APIs LOCAIS

- △ E também vamos criar a api responsável por gerenciar estes dados:
- △ Um pasta de nome base-api que pode ser criada dentro da pasta base mesmo e não vamos nos esquecer do arquivo route.jsx.



```
src > app > base > base-api > route.jsx > ...
```

```
1  import { NextResponse } from "next/server";
2
3  export async function GET(request, { params }) {
4      const file = await fs.readFile(process.cwd() + '/src/app/base/db.json', 'utf8');
5      return NextResponse.json(JSON.parse(file));
6  }
```



NEXT.JS

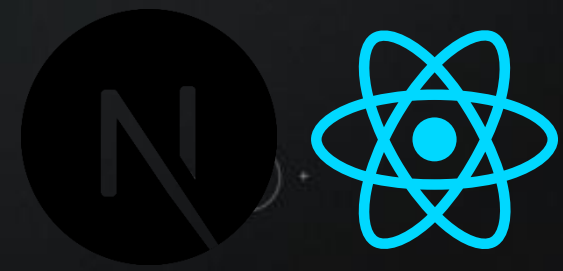
FIAP

Route Handlers / APIs LOCAIS

△ Explicando as funções utilizadas:

- file = Arquivo gerado do FileSistem(fs) em formato de arquivo para escrita e leitura.
- process.cwd() = Função(nativa) do Node.js que retorna o diretório de trabalho e sofre uma concatenação com o restante do caminho até onde o arquivo db.json foi adicionado.
- JSON.parse(file) = Pega o conteúdo do arquivo e transforma em formato de leitura json.

```
src > app > base > base-api > route.jsx > ...
1  import { promises as fs } from 'fs';
2  import { NextResponse } from "next/server";
3
4  export async function GET(request, { params }) {
5      const file = await fs.readFile(process.cwd() +
6          '/src/app/base/db.json', 'utf8');
7      return NextResponse.json(JSON.parse(file));
8  }
```



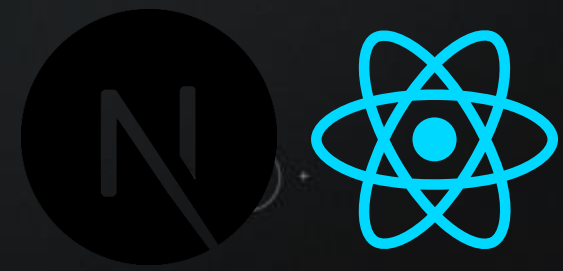
NEXT.JS

FIAP

Route Handlers / APIs LOCAIS

△ E nossa página para apresentar os dados:

△ *Aqui temos que ter atenção no encapsulamento do json por causa do parseamento do file(arquivo), fique atento, com as listas, pois dependendo da estrutura que foi criada a lista pode ter sido encapsulada em mais uma lista.*



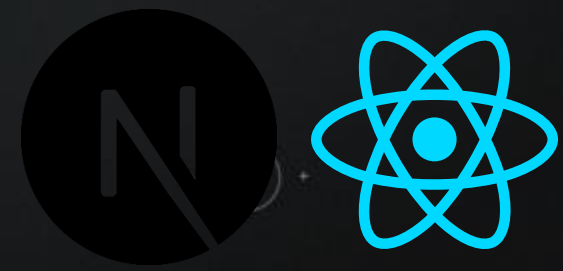
NEXT.JS

FIAP

Route Handlers / APIs LOCAIS

△ E nossa página para apresentar os dados:

```
src > app > dados-json > page.jsx > ...
1  export default async function DadosJson() {
2
3      const response = await fetch("http://localhost:3000/base/base-api");
4      const produtos = await response.json();
5
6      return (
7          <div>
8              <h1>Dados de Produtos recuperados do arquivo Json</h1>
9              <p>Recuperar todos os produtos do Github através da api-interna http://localhost:3000/base/
10             <div>
11                 <ul>
12                     {produtos.produtos.map((produto) => (
13                         <li key={produto.id}>
14                             <p>{produto.id}</p>
15                             <p>{produto.nome}</p>
16                             <p>{produto.tipo}</p>
17                             <p>{produto.desc}</p>
18                         </li>
19                     ))}
20                 </ul>
21             </div>
22         </div>
23     );
24 }
```

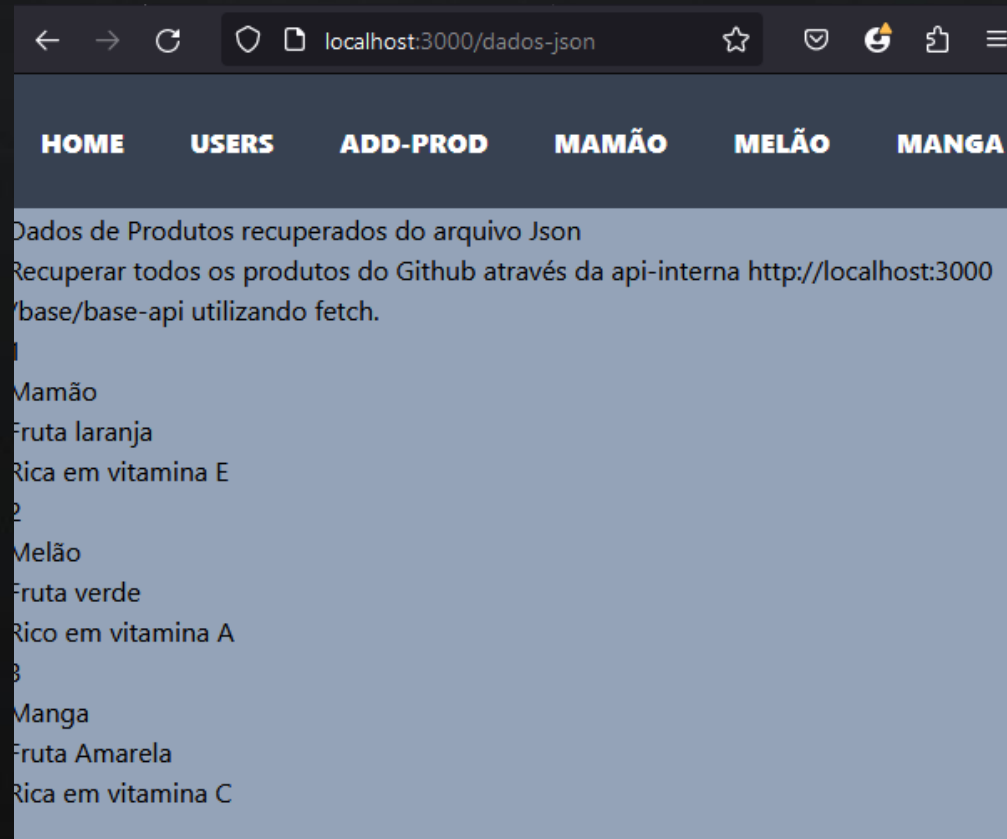


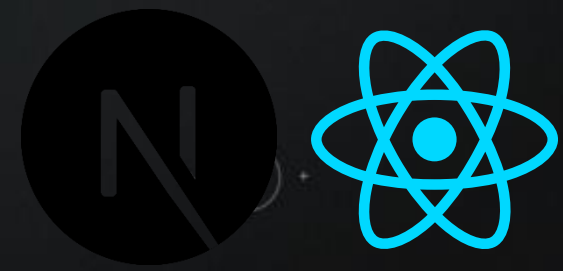
NEXT.JS

FIAP

Route Handlers / APIs LOCAIS

△ E nossa página para apresentar os dados:





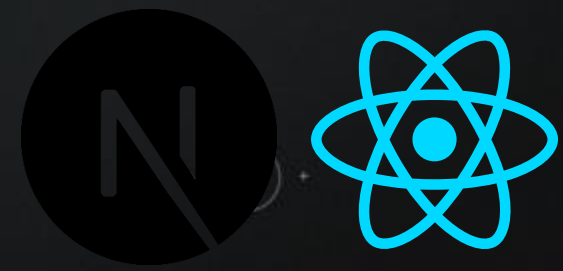
NEXT.JS

FIAP

Route Handlers / APIs LOCAIS

△ Emfim vamos adicionar dados em nossa base-json. Vamos criar um método em route.jsx do tipo **POST**.

```
src > app > dados > produto-api-add > route.jsx > ...
1  import { promises as fs } from 'fs';
2  import { NextResponse } from "next/server";
3
4  export async function POST(request) {
5    //Recuperando os dados do Formulário
6    const { nome, tipo, desc } = await request.json();
7
8    //Recuperando os dados do arquivo JSON em /app/base/db.json;
9    const file = await fs.readFile(process.cwd() + '/src/app/base/db.json', 'utf8');
10   const data = JSON.parse(file);
11
12   //Criando um novo produto
13   const newProduto = {
14     id:data.produtos[data.produtos.length-1].id + 1,
15     nome,
16     tipo,
17     desc,
18   };
19
20   //Adicionando o novo produto ao arquivo JSON em /app/base/db.json;
21   data.produtos.push(newProduto);
22
23   await fs.writeFile(process.cwd() + '/src/app/base/db.json', JSON.stringify(data));
24   return NextResponse.json({ message: "Produto adicionado com sucesso!" });
25 }
```



NEXT.JS

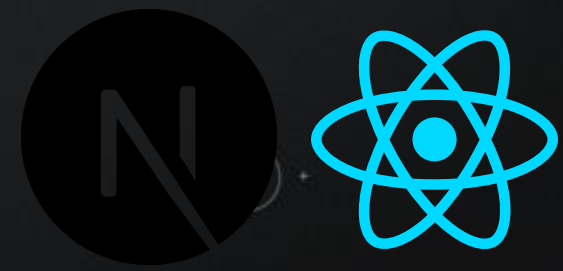
FIAP

Route Handlers / APIs LOCAIS

△ Agora vamos criar o formulário para adicionar os dados na base-json:

△ Veja que este componente foi declarado como “use-client”, ou seja, Client Component.

```
src > app > add-prod > page.jsx > AddProduto
1  "use client"
2  import { useState } from "react";
3
4  export default function AddProduto() {
5
6      const [produto, setProduto] = useState({
7          id: 0,
8          nome: "",
9          tipo: "",
10         desc: "",
11     });
12
13     const handleChange = (e) => {
14         const { name, value } = e.target;
15         setProduto({ ...produto, [name]: value });
16     };
17
18     const handleSubmit = async (e) => {
19         e.preventDefault();
20
21         const response = await fetch("http://localhost:3000/dados/produto-api-add", {
22             method: "POST",
23             headers: {
24                 "Content-Type": "application/json",
25             },
26             body: JSON.stringify(produto),
27         });
28         const result = await response.json();
29         console.log(result);
30     };
}
```



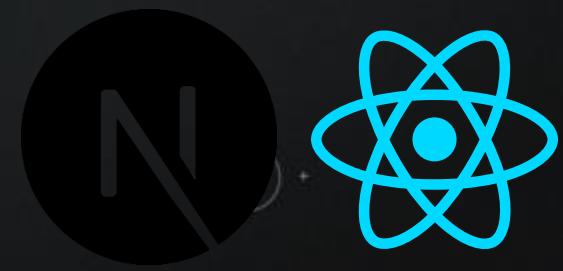
NEXT.JS

FIAP

Route Handlers / APIs LOCAIS

△ Veja que o nosso formulário foi criado utilizando a mesma estratégia de preenchimento de desestruturação com `useState`.

```
src > app > add-prod > page.jsx > ...
31
32   return (
33     <div>
34       <h1>ADD-PROD</h1>
35       <p>Adicione um produto</p>
36
37       <div>
38         <form className="formProd" onSubmit={handleSubmit}>
39           <fieldset>
40             <legend>Novo Produto</legend>
41             <div>
42               <label htmlFor="nome">Nome:</label>
43               <input type="text" id="nome" name="nome" value={produto.nome} onChange={handleChange}/>
44             </div>
45             <div>
46               <label htmlFor="tipo">Tipo:</label>
47               <input type="text" id="tipo" name="tipo" value={produto.tipo} onChange={handleChange}/>
48             </div>
49             <div>
50               <label htmlFor="desc">Descrição:</label>
51               <textarea id="desc" name="desc" value={produto.desc} onChange={handleChange}></textarea>
52             </div>
53             <div>
54               <button>Enviar</button>
55             </div>
56           </fieldset>
57         </form>
58       </div>
59     </div>
60   )
61 }
62 }
```



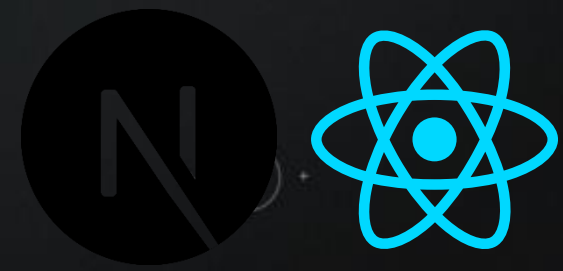
NEXT.JS

FIAP

Route Handlers / APIs LOCAIS

△ Veja que o nosso formulário foi criado utilizando a mesma estratégia de preenchimento de desestruturação com `useState`.

```
src > app > add-prod > page.jsx > ...
31
32   return (
33     <div>
34       <h1>ADD-PROD</h1>
35       <p>Adicione um produto</p>
36
37       <div>
38         <form className="formProd" onSubmit={handleSubmit}>
39           <fieldset>
40             <legend>Novo Produto</legend>
41             <div>
42               <label htmlFor="nome">Nome:</label>
43               <input type="text" id="nome" name="nome" value={produto.nome} onChange={handleChange}/>
44             </div>
45             <div>
46               <label htmlFor="tipo">Tipo:</label>
47               <input type="text" id="tipo" name="tipo" value={produto.tipo} onChange={handleChange}/>
48             </div>
49             <div>
50               <label htmlFor="desc">Descrição:</label>
51               <textarea id="desc" name="desc" value={produto.desc} onChange={handleChange}></textarea>
52             </div>
53             <div>
54               <button>Enviar</button>
55             </div>
56           </fieldset>
57         </form>
58       </div>
59     </div>
60   )
61 }
62 }
```



NEXT.JS

FIAP

Route Handlers / APIs LOCAIS

△ Form Add-Prod

← → ↻ 🔒 📄 localhost:3000/add-prod ☆ 📄 >> ≡

HOME USERS ADD-PROD MAMÃO MELÃO MANGA

ADD-PROD
Adicione um produto

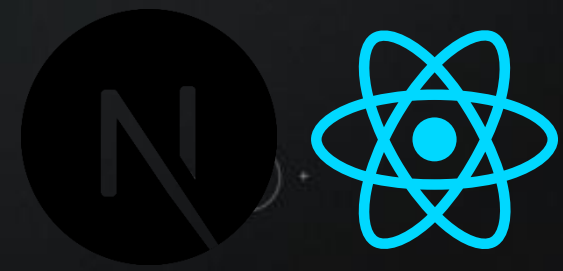
Novo Produto

Nome:

Tipo:

Descrição:

Enviar



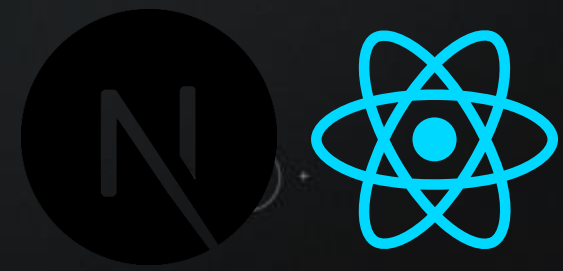
NEXT.JS

FIAP

Route Handlers / APIs LOCAIS

△ Resultado do POST no arquivo json.

```
{
  "produtos": [
    {
      "id": 1,
      "nome": "Mamão",
      "tipo": "Fruta laranja",
      "desc": "Rica em vitamina E"
    },
    {
      "id": 2,
      "nome": "Melão",
      "tipo": "Fruta verde",
      "desc": "Rico em vitamina A"
    },
    {
      "id": 3,
      "nome": "Manga",
      "tipo": "Fruta Amarela",
      "desc": "Rica em vitamina C"
    },
    {
      "id": 4,
      "nome": "Banana",
      "tipo": "Fruta Massiva",
      "desc": "Rica em potássio."
    }
  ]
}
```



NEXT.JS

FIAP

Comportamento : POST / APIs LOCAIS

Da mesma forma, o `POST` O método fará com que o manipulador de rota seja avaliado dinamicamente.

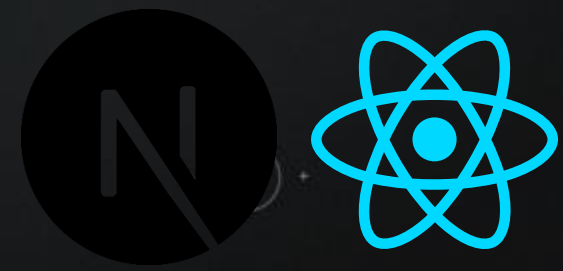
JS app/items/route.js

JavaScript

```
export async function POST() {
  const res = await fetch('https://data.mongodb-api.com/...', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'API-Key': process.env.DATA_API_KEY,
    },
    body: JSON.stringify({ time: new Date().toISOString() }),
  })

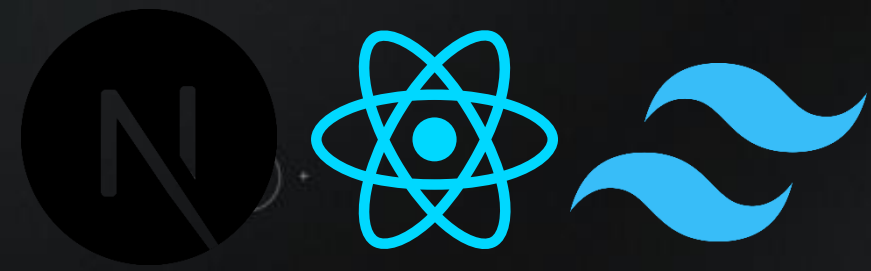
  const data = await res.json()

  return Response.json(data)
}
```



Outros Exemplos GET/POST em Router Handles

```
src > app > dados > route.jsx > GET
27
28   export async function POST(request) {
29     const {nome, email, senha} = await request.json();
30     const id = dados[ dados.length - 1].id + 1;
31     dados.push({id, nome, email, senha});
32     console.log(dados);
33     return Response.json(dados);
34   }
35
36
37   export async function GET() {
38     return NextResponse.json(dados);
39   }
```

NEXT.JS

FIAP

tailwind

△ Aproveitando que o NEXT.js já tem o tailwind integrado por padrão nada melhor do que utilizar este framework de css para trabalhar. Então vamos as configurações iniciais:

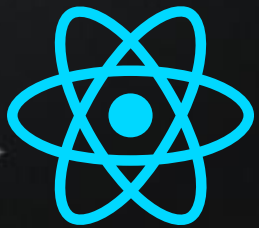
△ 1 - Primeira configuração a ser realizada deve ser a alteração dos prefixos e chamadas ao tailwind pelo preprocessor postcss. Vamos até o arquivo `postcss.config.js` e vamos alterar as seguintes configurações:

Altere o código deste

```
postcss.config.js > ...  
...  
1  module.exports = {  
2    plugins: {  
3      tailwindcss: {},  
4      autoprefixer: {},  
5    },  
6  }
```

```
postcss.config.js > ...  
You, 9 seconds ago | 1 author (You)  
1  module.exports = {  
2    plugins: {  
3      'postcss-import': {},  
4      'tailwindcss/nesting': {},  
5      tailwindcss: {},  
6      autoprefixer: {},  
7    },  
8  }
```

Altere para este!!!



NEXT.JS

FIAP

tailwind

```
src > app > globals.css > ...  
You, 9 seconds ago | 1 author (You)  
1 @tailwind base;  
2 @tailwind components;  
3 @tailwind utilities;  
4 You, 10 hours ago
```

- △ Agora em nosso arquivo principal o `globals.css`, vamos adicionar as `@layers` que representam todas as áreas de nossa aplicação onde devemos aplicar a estilização:
- △ `@base` - Esta camada representa todo o estilo principal, que vai ser aplicado de forma linear no projeto.
- △ `@componentes` - Esta camada compreende os componentes que estão sendo trabalhado. Ex: header, menu, buttons, rodapé.
- △ `@utilities` - Camada responsável por aglutinar todos aqueles elementos que realizam funções diferenciadas: Ex - Darkmode.



OBRIGADO

FIAP

Copyright © 2020 | Professor Titulares

Todos os direitos reservados. Reprodução ou divulgação total ou parcial deste documento, é expressamente proibido sem consentimento formal, por escrito, do professor/autor.

