CADERNO CURSO COMPLETE NEXT.JS DEVELOPER IN 2022 - ZERO TO MASTERY

**SUMÁRIO**

[1 VANTAGENS DO NEXT.JS 3](#_Toc90030254)

[1.1 Diferentes técnicas de renderização: 3](#_Toc90030255)

[1.2 Performance: 3](#_Toc90030256)

[1.3 File based rounting: 3](#_Toc90030257)

[1.4 SEO: 4](#_Toc90030258)

[1.5 Serveless functions: 4](#_Toc90030259)

[2 PRIMEIRO PROJETO 5](#_Toc90030260)

[2.1 Comandos criar projetos Next.js 5](#_Toc90030261)

[2.2 Underscore em arquivos na pasta pages 5](#_Toc90030262)

[2.3 Fast Refresh 6](#_Toc90030263)

[2.4 CSS Modules 6](#_Toc90030264)

[2.5 Head Component 6](#_Toc90030265)

[2.6 Next Routes 7](#_Toc90030266)

[2.6.1 Index route 7](#_Toc90030267)

[2.6.2 Nested routes 8](#_Toc90030268)

[2.6.3 Dynamic routes 9](#_Toc90030269)

[2.6.4 Next/link 9](#_Toc90030270)

[3 Adicionando fonts customizadas no diretório public 10](#_Toc90030271)

[4 Next SEO 11](#_Toc90030272)

[4.1 Pre-rendering 11](#_Toc90030273)

[5 Combinar classes em CSS componentes 12](#_Toc90030274)

# VANTAGENS DO NEXT.JS

O Next.js é um framework baseado no React, que oferece flexibilidade para criar aplicativos escaláveis, que permitem renderizar conteúdo no servidor.

O Next.js oferece diferentes benefícios, veja alguns deles:

## Diferentes técnicas de renderização:

Static site Generation: Permite a renderização do site de forma estática em tempo real.

Server side rendering: Geração de páginas no servidor.

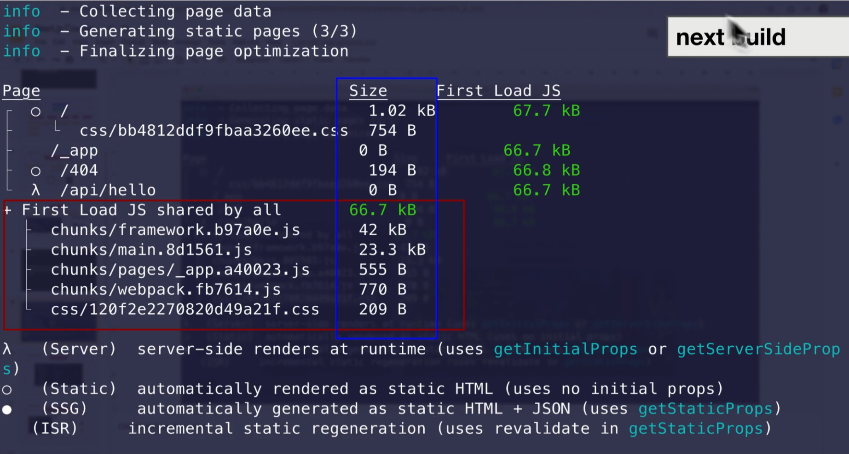
Incremental site regeneration: É uma mistura dos dois modos anteriores.

## Performance:

O Next.js possui uma otimização de performance muito robusta, ele realiza:

* Code Splitting
* Minifying files
* Image optimization
* Pre-fetching assets

O Code Splitting divide o seu app web em pedaços, para que eles sejam renderizados apenas quando forem requisitados na página atual.



Ele realiza também optimização de imagens, com features incluídas no framework ele diminui o tamanho dos arquivos de imagem de acordo com o device utilizado, otimização essa automática.

## File based rounting:

(Entender melhor, voltar e rever)

## SEO:

O Next’js permite a otimização para SEO, é ótimo para lidar com os robôs do Google.

## Serveless functions:

As funções serveless servem para realizar chamadas para API’s nativas dentro do servidor, de forma que estejam ocultas do cliente, por questões de segurança.

# PRIMEIRO PROJETO

## Comandos criar projetos Next.js

// Criar um novo projeto Next

npx create-next-app

Ou

yarn create next-app

// Rodar o projeto Next

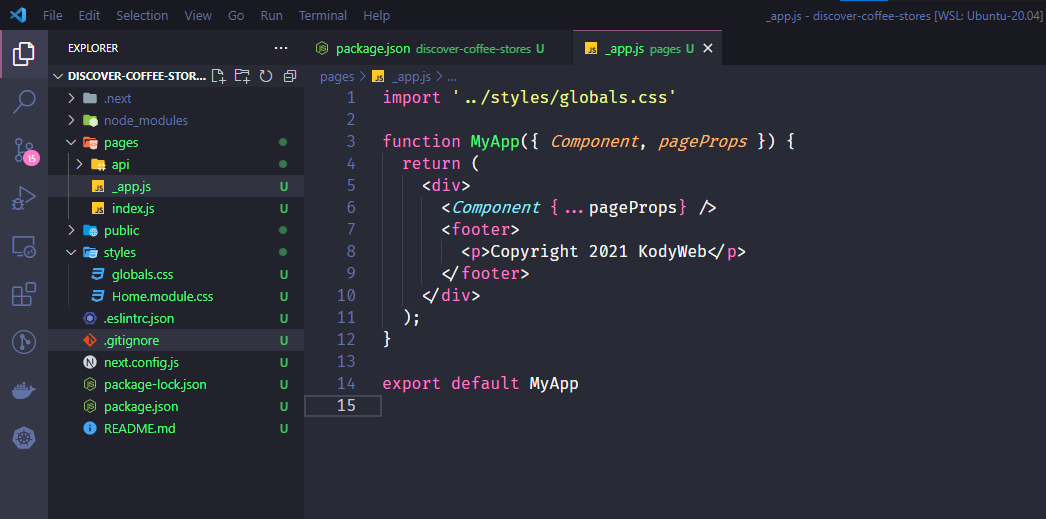
npm run dev

Ou

yarn run dev

## Underscore em arquivos na pasta pages

Ao colocar underscore em nomes de arquivos na pasta pages, esse componente será executado em todas as outras páginas de forma automática, por exemplo, se temos um footer que tem que ser renderizado em todas as páginas convém nomeá-lo com um “\_” para que ele seja repetido em todas as páginas.



## Fast Refresh

O fast-refresh é uma funcionalidade do Next que atualiza a página rapidamente conforme salvamos as alterações no código, sem necessidade de adicionar plugins terceiros, isso já faz parte do Next.

## CSS Modules

CSS Modules trazem uma enorme vantagem no desenvolvimento de styles para o nosso projeto, com eles podemos criar documentos css separados para cada página do site, dando o nome do componente. E uma outra vantagem é que com isso podemos ter nomes de classes iguais em diferentes documentos, por exemplo, temos dois documentos css um com o nome:

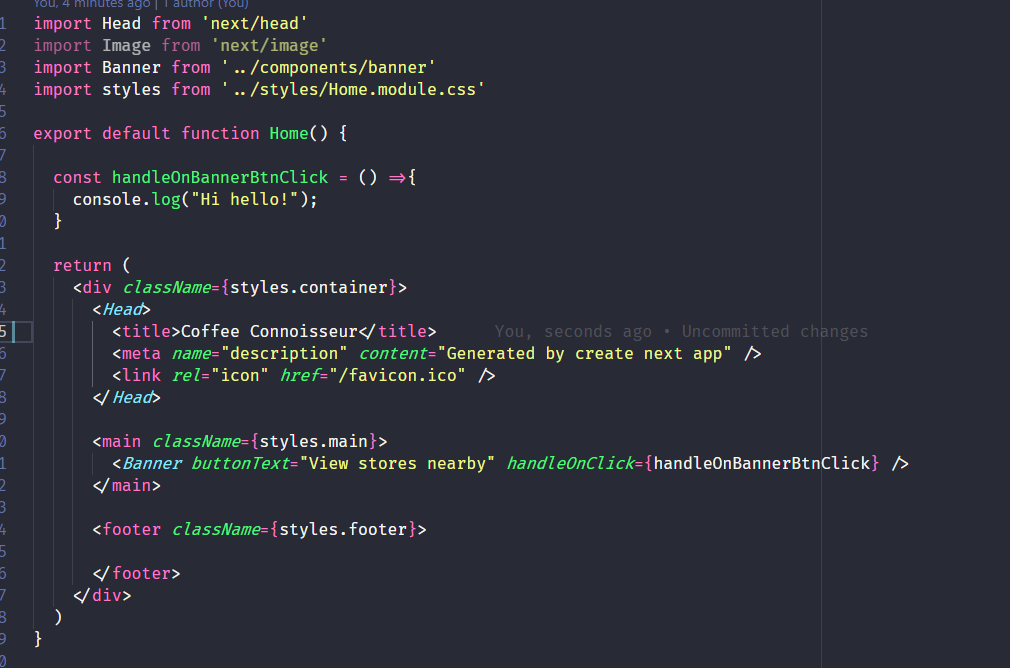
Home.module.css

Main.module.css

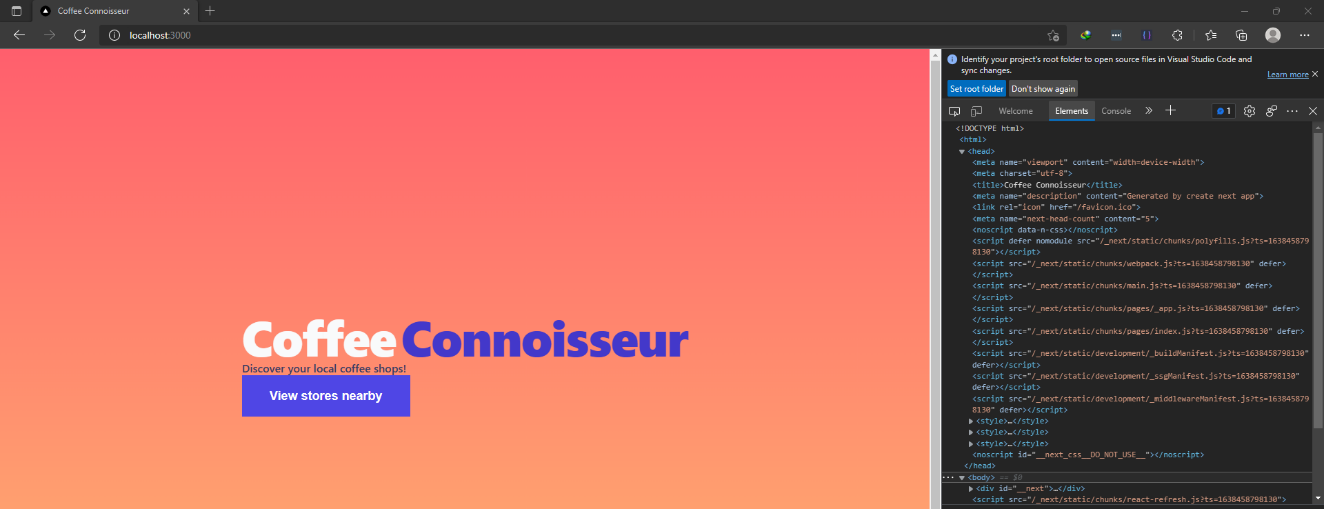
Dentro do Home.module.css temos a classe .container e podemos ter essa mesma classe no Main.module.css também que não haverá conflitos.

## Head Component

O Next possui um componente Head para editar informações do escopo da tag Head, ele gera todos os códigos de forma automática sem a necessidade de escrever as informações das outras tags que ficam dentro do head, veja:



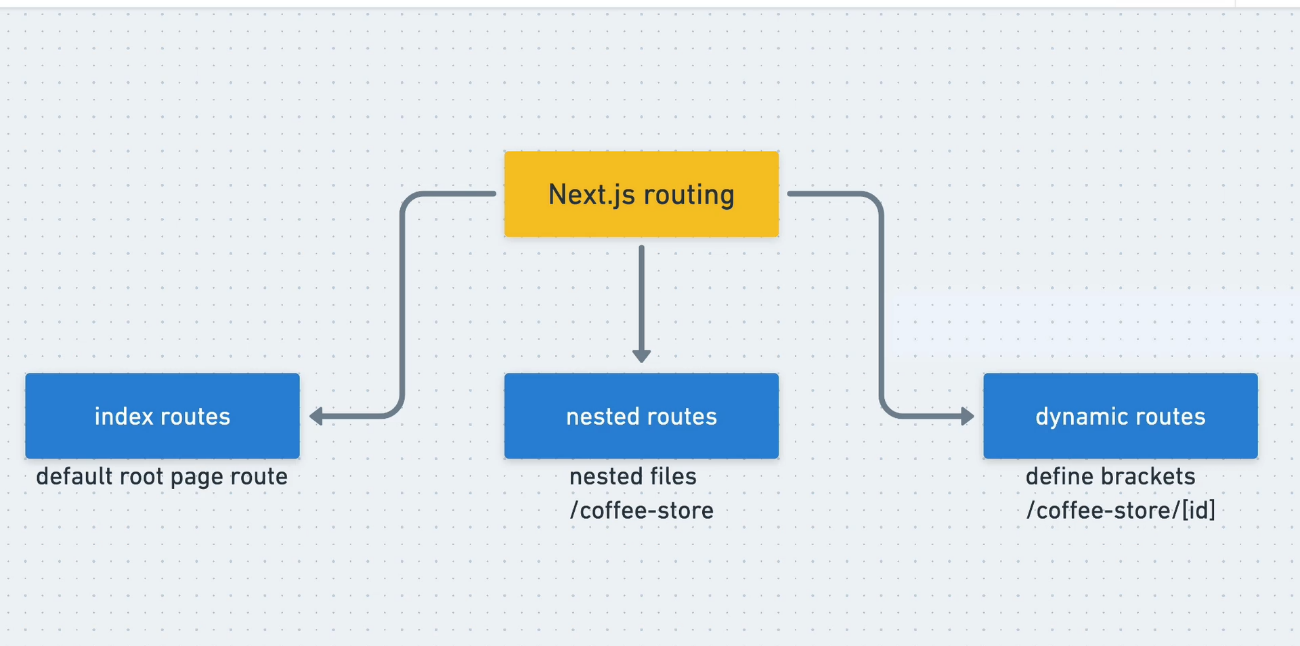
Apenas com esse código ai no Head já é suficiente, veja o código gerado no cliente:



Repare todas as tags geradas de forma automática pelo Next.

## Next Routes

O Next possibilita trabalharmos com routes de diferentes maneiras:

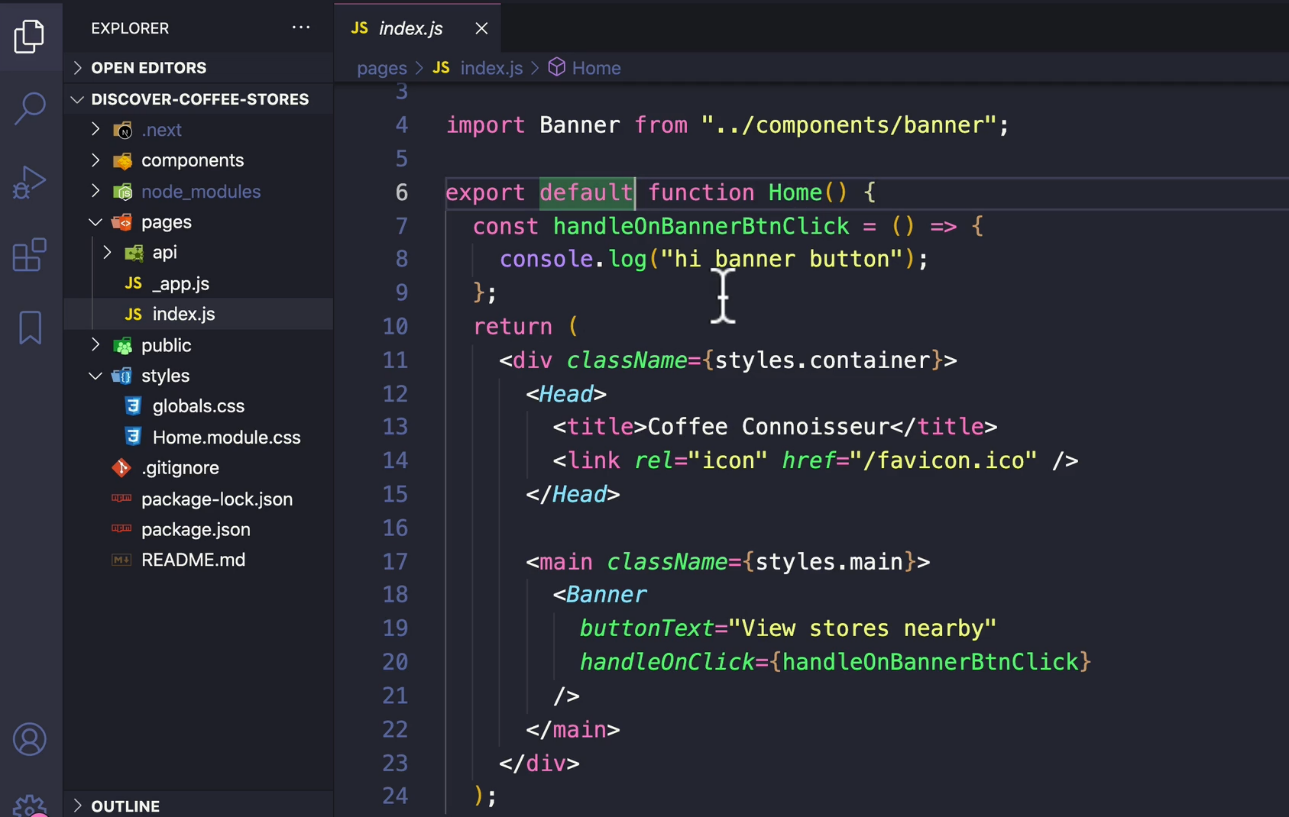


Vamos entender cada uma dessas formas:

### Index route

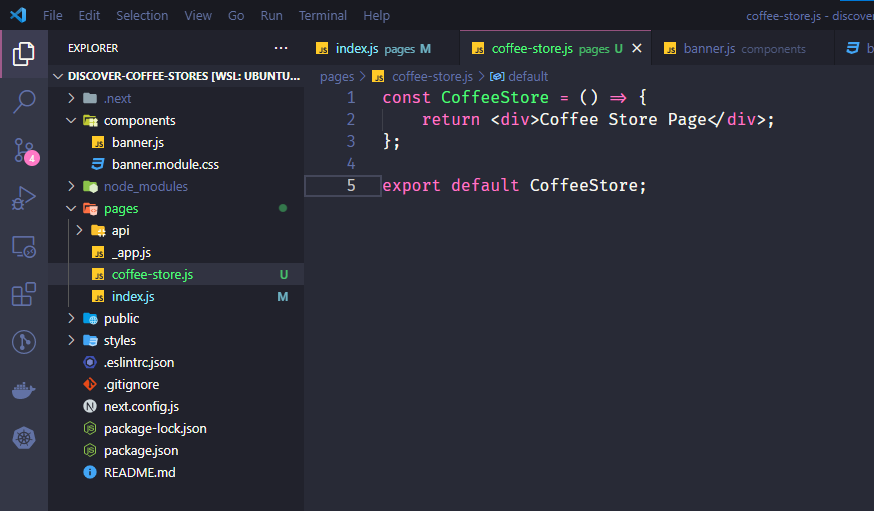
Basicamente nos diretórios das pastas temos um arquivo index.js na pasta pages, esse arquivo é sempre o root exibido por padrão na primeira página do site, é a url padrão, existe 2 regras para routing:

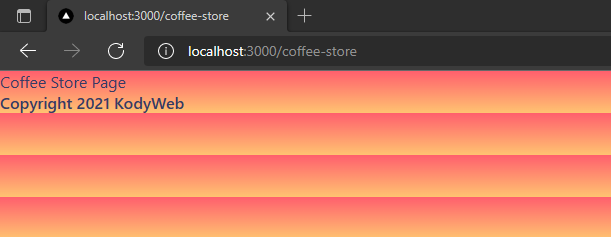
1. A página precisa ser um React componente
2. Componentes precisam ser exportados por default



### Nested routes

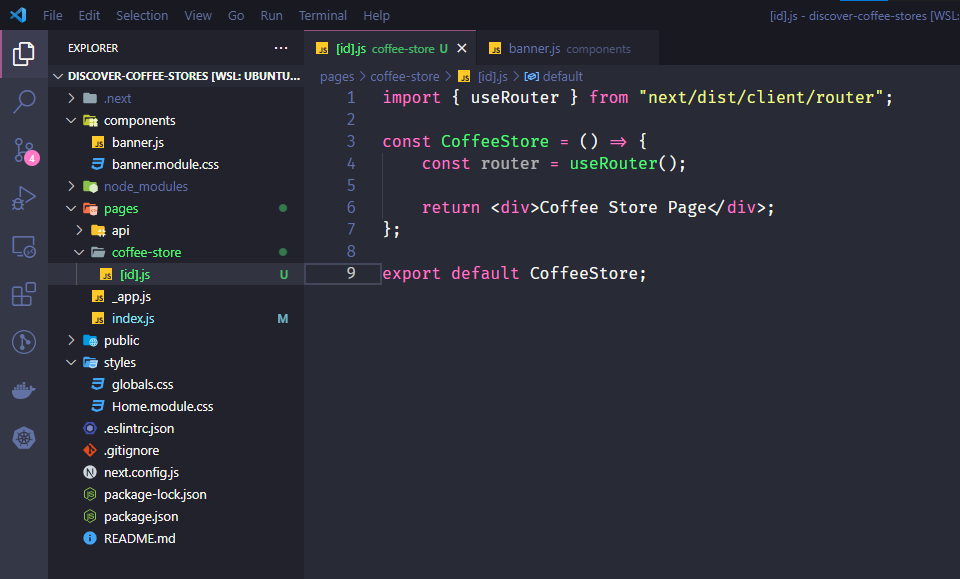
São as páginas que seguem depois da root, criamos um componente e ao adicionar a nova página na url ele fara o link.

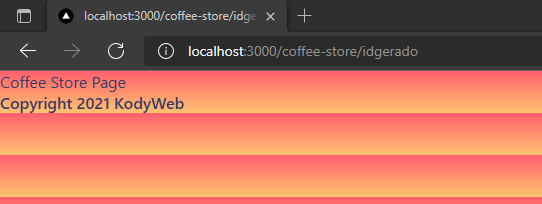




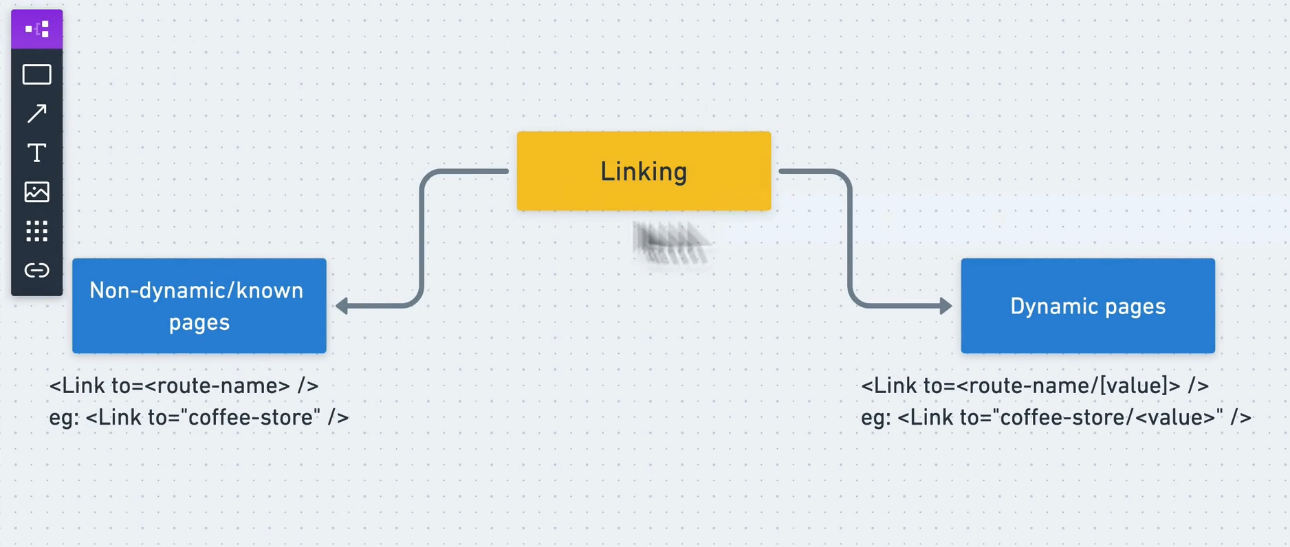
### Dynamic routes

São routes que geram identificadores únicos para uma página.





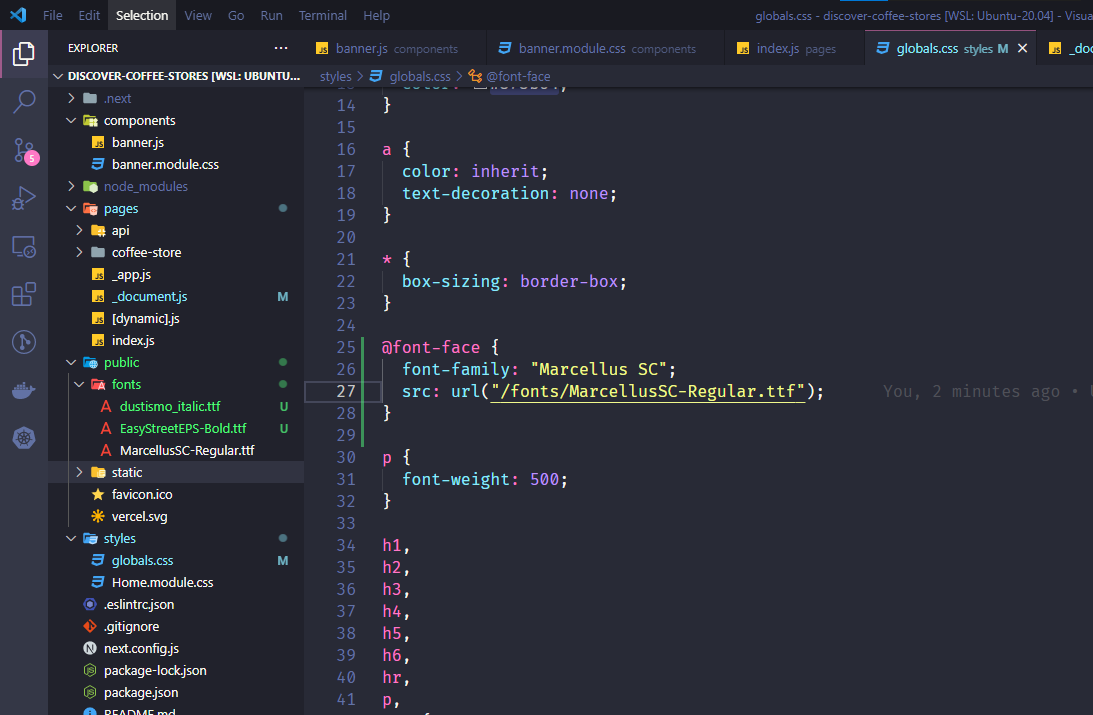
### Next/link



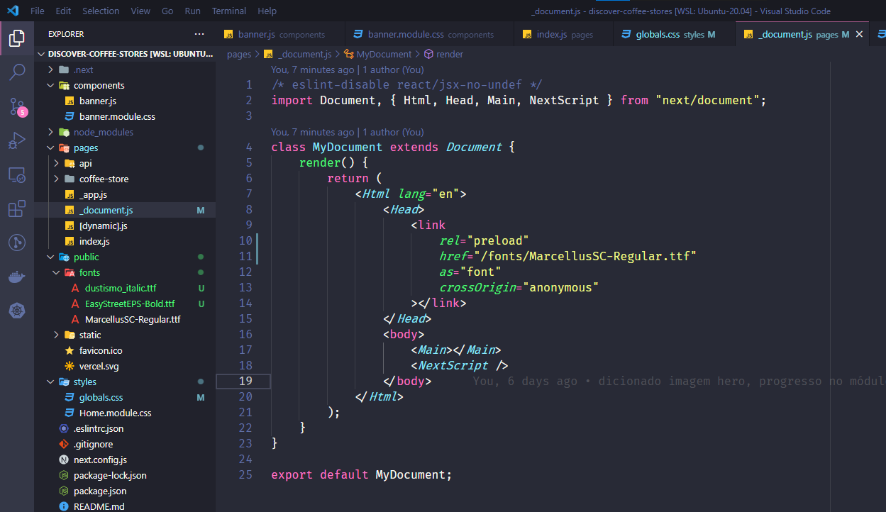
# Adicionando fonts customizadas no diretório public

Podemos adicionar fonts dentro do diretório do Next para podermos utilizar elas de forma mais segura sem depender de serviços online de fonts, com isso nosso projeto fica livre de falhas de design por falha de conexão a serviços como o Google Fonts. Estando ciente de que ter os arquivos de fonts no diretório do projeto é uma boa prática podemos inserir e configurar elas no Next.

Para isso coloque as fonts no diretório public/fonts como na imagem abaixo:

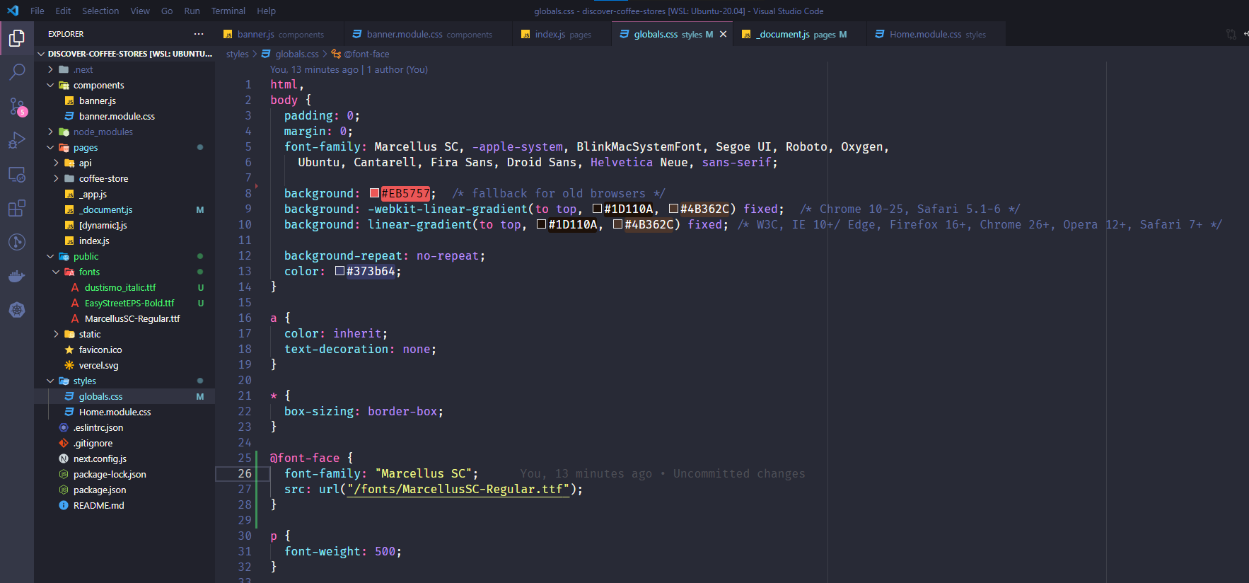


Após ter feito isso, crie um novo documento para ser pré-renderizado pelo Next



Com isso utilizamos uma técnica de pré-loading para que o Next sempre carregue a font no Head do documento HTML. Para passarmos o caminho da font podemos omitir a pasta public do caminho e apenas chamar o caminho fonts diretamente como na imagem acima.

Agora precisamos carregar essa font no documento por intermédio do arquivo global.css, veja:

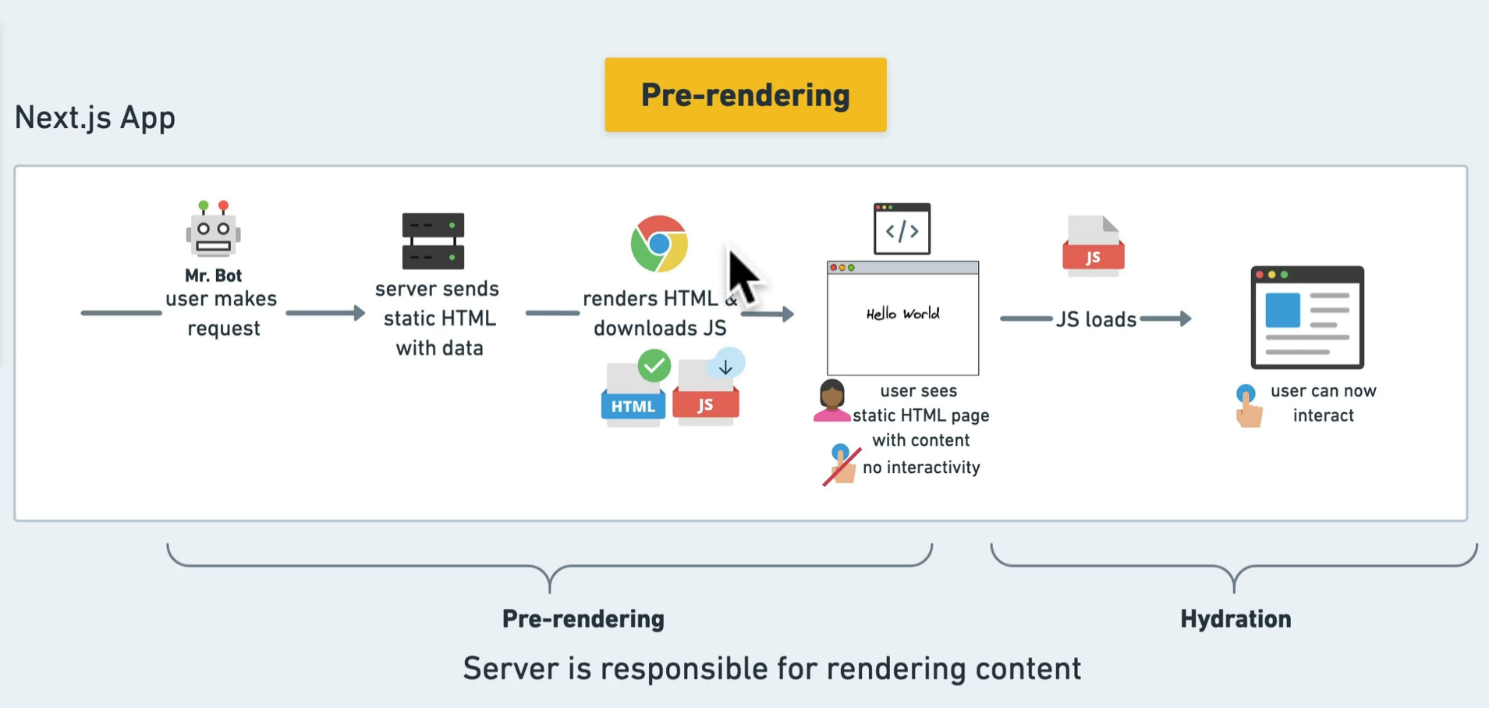


Para isso utilizamos o @font-face do css e damos um nome ancora qualquer a família da font e setamos o caminho da font, assim inserimos dentro da regra css que carrega os presets de font-family da nossa página, que é o body neste caso e adicionamos o nome ancora que criamos na tag @font-face.

# Next SEO

SEO é importante para a indexação do nosso conteúdo nos bancos de dados do Google, para que assim sejamos encontrados nas buscas dos usuários, com o Next podemos tomas as mesmas medidas padrões dentro do documento HTML para otimizar essa busca, cuidando da semântica do HTML e adicionando sempre as meta tags para descrever o conteúdo e também as tags alt de conteúdo de imagens.

## Pre-rendering



O pré-renderindo do Next trabalha de forma que o servidor é responsável por renderizar a aplicação com as tags HTML estáticas todas com conteúdo, desse modo trazendo todo o conteúdo do documento, suas tags, títulos, textos, imagens etc. E depois que o JavaScript renderiza o usuário pode interagir.

Comparando com o React que só vai renderizar o conteúdo HTML depois que o JavaScript terminar de renderizar, no React o JavaScript é responsável por renderizar o conteúdo e não o servidor, o que acontece no React é que o user recebe uma página em branco que não contém HTML.

# Combinar classes em CSS componentes

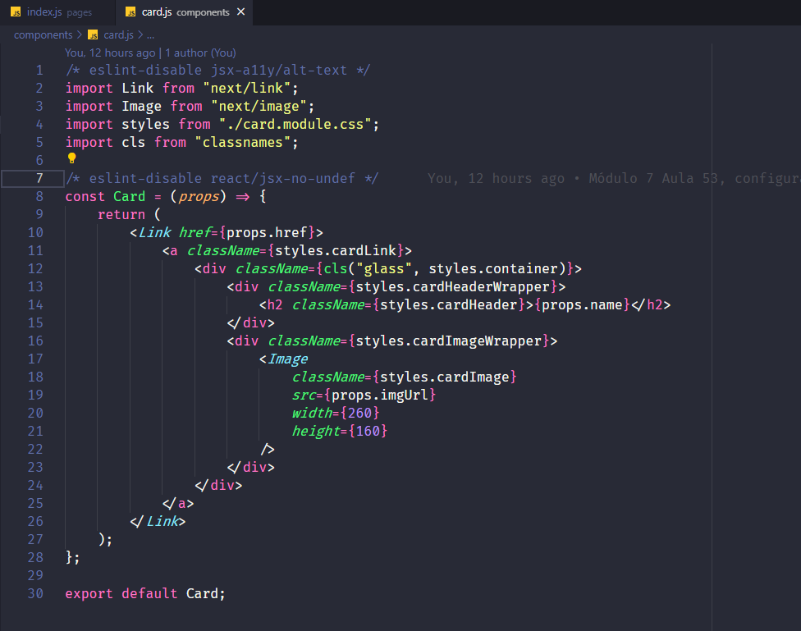
Podemos combinar classes CSS com a extensão classnames, para isso basta instalar ele com:

npm install classnames

ou

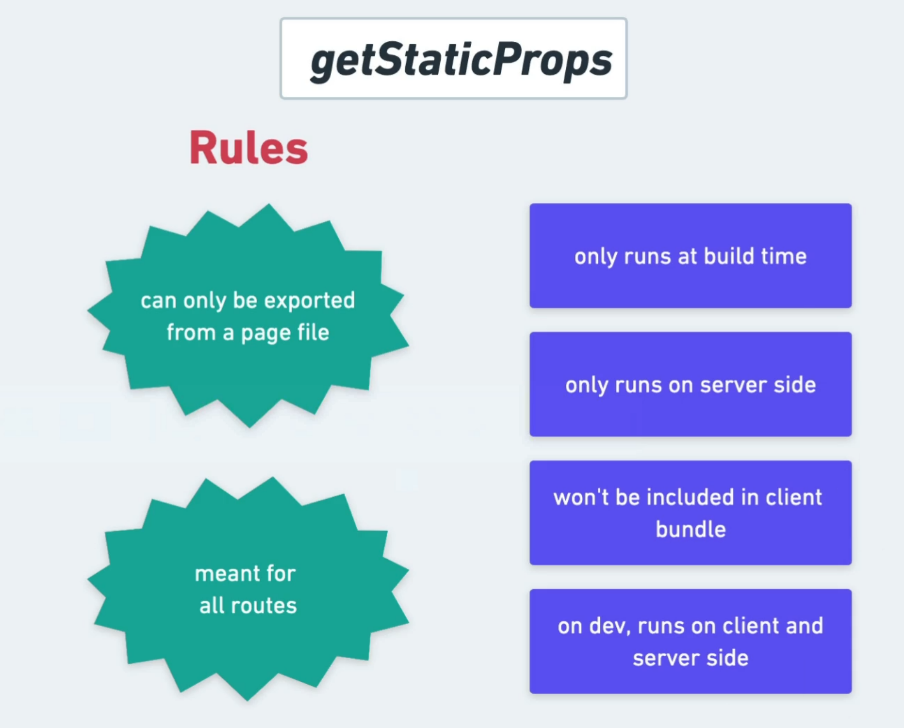
yarn install classnames

Com isso instalamos o pacote classnames e estamos prontos para utilizar no projeto. Para utilizar basta importar o cls no documento e utilizar na chamada de classe onde necessita utilizar mais de uma classe css, veja:

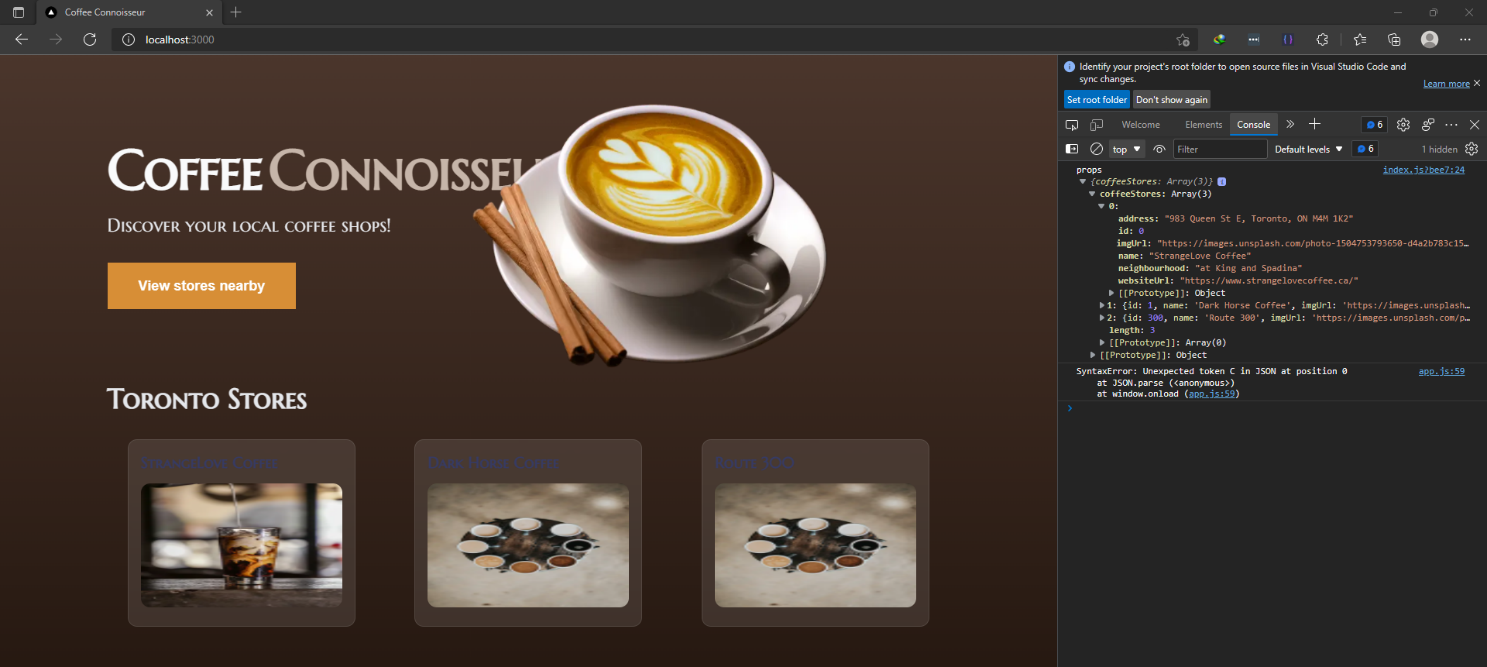


Na linha <div className={cls("glass", styles.container)}> estamos setando 2 classes css em uma mesma chamada utilizando o cls.

# getStaticProps



A função getStaticProps serve para criar funções no lado do servidor, onde não irá aparecer no cliente, todas as funções escritas dentro do escopo do getStaticProps não será visível no cliente. E ela é responsável por pré-renderizar o conteúdo de forma estática, obtido através de uma API, com a pré-renderização o site fica muito mais rápido, e também traz uma certa segurança de impedir que alguns conteúdos e funções fiquem expostas do lado do cliente, veja:



# Condições para exibição de conteúdo

Podemos utilizar um if condicional para exibirmos determinados conteúdo ou não caso não haja dados para serem mostrados, veja o código abaixo:

<main *className*={styles.main}>

        <*Banner* *buttonText*="View stores nearby" *handleOnClick*={handleOnBannerBtnClick} />

        <div *className*={styles.heroImage}>

          <*Image* *src*="/static/hero-image.png" *width*={500} *height*={375}/>

        </div>

        {props.coffeeStores.length > 0 && (

          <>

            <h2 *className*={styles.heading2}>Toronto Stores</h2>

            <div *className*={styles.cardLayout}>

              {props.coffeeStores.map((*coffeeStore*) => {

                return (

                  <*Card*

*key*={coffeeStore.id}

*name*={coffeeStore.name}

*imgUrl*={coffeeStore.imgUrl}

*href*={`/coffee-store/${coffeeStore.id}`}

                  />

                );

              })}

            </div>

          </>

        )}

</main>

Repare a partir desta linha :

Ela é a condição que possibilita a exibição ou não do conteúdo dependendo se houver dados para serem mostrados, caso seja maior do que 0 (tenha conteúdo) será mostrado, caso seja 0 não será exibido nada, por isso a tag <> vazia faz esse trabalho.

# getStaticPaths

