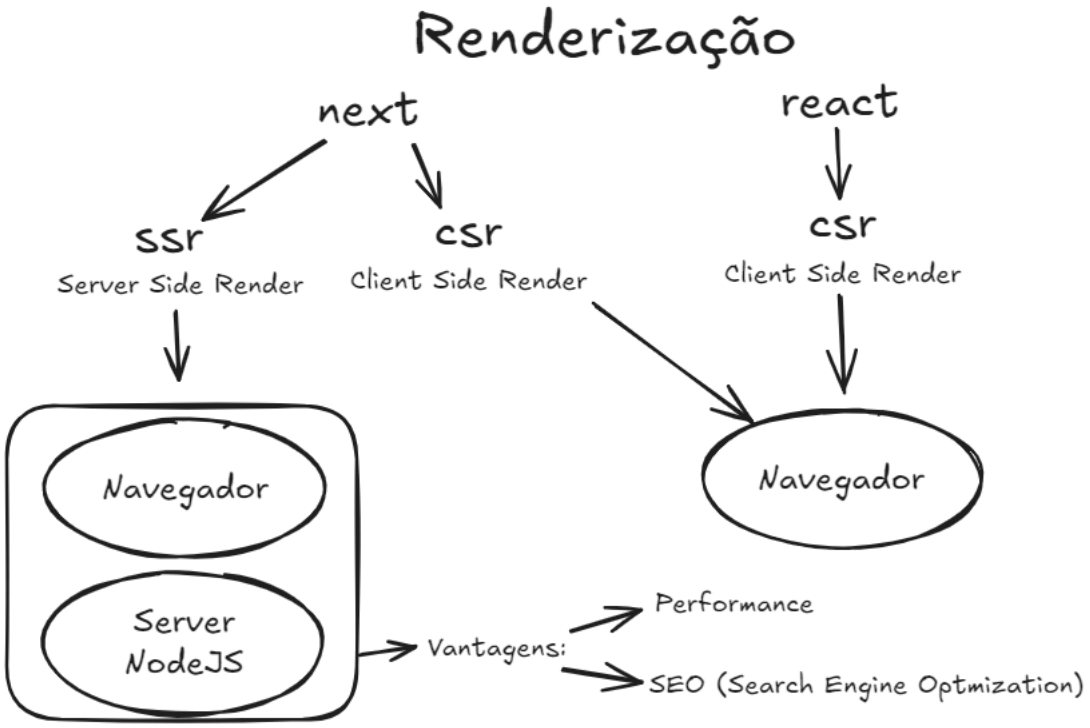
O que é o Next.js?

Next.js é um framework de desenvolvimento baseado em **React** que permite criar aplicativos web e sites otimizados, incluindo suporte para renderização no lado do servidor (SSR), geração de sites estáticos (SSG), APIs integradas e muito mais. Criado pela **Vercel**, o Next.js fornece uma estrutura completa e otimizada para a construção de aplicações React, lidando automaticamente com configurações de Webpack, Babel e roteamento.

Principais Recursos do Next.js

1. **Renderização no Lado do Servidor (SSR)**:
   - No Next.js, é fácil configurar páginas para serem renderizadas no servidor. Isso significa que o HTML da página pode ser gerado no servidor antes de ser enviado ao cliente, o que melhora o desempenho e SEO. Basta usar a função getServerSideProps.
2. **Geração Estática (SSG)**:
   - Com o Next.js, você pode gerar páginas estáticas durante o build (compilação) da aplicação, usando a função getStaticProps. Isso é útil para conteúdo que não muda frequentemente, garantindo velocidade e SEO aprimorado.
3. **Incremental Static Regeneration (ISR)**:
   - O Next.js permite atualizar páginas estáticas após a publicação sem precisar reconstruir todo o site. É uma forma híbrida que permite manter a velocidade do SSG com dados mais recentes.
4. **Roteamento Automático**:
   - Next.js possui um sistema de roteamento automático baseado em arquivos. Toda página criada na pasta /pages é automaticamente transformada em uma rota acessível.
5. **API Routes**:
   - Você pode criar APIs RESTful diretamente no projeto Next.js dentro da pasta /pages/api. Isso elimina a necessidade de configurar um servidor separado para gerenciar APIs, ideal para microsserviços simples.
6. **Suporte Nativo ao CSS e CSS Modules**:
   - O Next.js suporta estilos CSS diretamente e permite a modularização de estilos usando **CSS Modules**, além de oferecer suporte a SASS e Styled Components.

Entendendo as diferenças na renderização



Instalando e Configurando o Next.js

Para iniciar um novo projeto Next.js, você precisa ter o **Node.js** instalado. Em seguida, pode criar um projeto Next.js usando **npx**:

```
    npx create-next-app meu-projeto
```

O next poderá fazer algumas perguntas, abaixo deixo sugestão de como responder:

```
create-next-app@15.0.1
Ok to proceed? (y) y
√ What is your project named? ... my-primary-next
√ 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 your code inside a `src/` directory? ... No / Yes
√ Would you like to use App Router? (recommended) ... No / Yes
√ Would you like to use Turbopack for next dev? ... No / Yes
√ Would you like to customize the import alias (@/* by default)? ... No / Yes
```

Isso cria um novo projeto com uma estrutura básica de pastas e configurações predefinidas. Após a instalação, navegue até o diretório do projeto e execute-o com:

```
cd meu-projeto
npm run dev
```

Agora você terá o Next.js rodando em http://localhost:3000

Estrutura Básica de um Projeto Next.js

- **/pages**: Onde as páginas são definidas. Cada arquivo é uma rota.
- **/public**: Armazena ativos públicos como imagens e fontes.
- **/styles**: Guarda os arquivos de estilo (CSS ou SASS).
- **next.config.js**: Arquivo de configuração para personalizações avançadas.

Fluxo da aula

Start do projeto Next
Explicando as pastas iniciais do projeto
Explicando como funciona roteamento no Next
Explicando como funciona roteamento dinâmico
Explicando rotas aninhadas (pasta dentro de pasta com roteamento dinâmico)
Rotas 404
Explicando o arquivo Layout (cabeçalho e rodapé)
Entendendo Layouts aninhados (arquivo layout.tsx dentro das pastas da página)
{ children }: { children: React.ReactNode }
Título e metadata

```
export const metadata: Metadata = {
	title: {
		absolute: ‘’, (ignora qualquer informação de template ou default)
		default: ‘’, (se nao tiver algo no template)
		template: ‘%s | alguma coisa fixa’
	},
	description: 'Confira todos os posts',
};
```

Componente Link (permite navegação sem recarregar a página)

```
<Link href="/about">
	<a className={router.pathname === '/about' ? 'active' : ''}>Sobre</a>
</Link>
```

Links Ativos (‘use client’)
Arquivo Template (se utilizar ele não preserva o estado dos componentes e variáveis já o arquivo Layout preserva)
Route Groups (se criar uma pasta (auth) com duas pastas login e register dentro, por ela estar entre parênteses vai poder acessar /login e /register direto)

Manipulação de Erros (criar arquivo error.tsx)
Route Handlers (arquivo route.tsx dentro da pasta)

```
// app/api/posts/route.js
export async function GET() {
	return new Response(JSON.stringify([{ id: 1, title: 'Primeiro Post' }]), { status:
	200, headers: { 'Content-Type': 'application/json' }, });
}

export async function POST(request) {
	const data = await request.json();
	return new Response(JSON.stringify({ message: 'Post criado', data }), { status:
201 });
}
```

Manipulando rotas POST

```
import { useState } from 'react';
export default function CreatePost() {
	const [title, setTitle] = useState('');
	const handleSubmit = async (event) => {
		event.preventDefault();
		await fetch('/api/posts', {
			method: 'POST',
			headers: { 'Content-Type': 'application/json' },
			body: JSON.stringify({ title }), });
		};
	return (
		<form onSubmit={handleSubmit}>
			<label>Título do Post</label>
			<input value={title} onChange={(e) => setTitle(e.target.value)} />
			<button type="submit">Criar Post</button>
		</form>
	);
}
```

Manipulação de Rotas Dinâmicas (PATCH e DELETE)

```
// app/api/posts/[id]/route.js
export async function PATCH(request, { params }) {
	const data = await request.json();
	return new Response(JSON.stringify({ message: `Post ${params.id}
	atualizado`, data }), { status: 200 });
}

// app/post/[id]/edit/page.js
import { useState } from 'react';
import { useRouter } from 'next/router';
```

```
export default function EditPost() {
	const router = useRouter();
	const { id } = router.query;
	const [title, setTitle] = useState('');
	const handleEdit = async (e) => {
		e.preventDefault();
		await fetch(`/api/posts/${id}`, { method: 'PATCH', headers: {
		'Content-Type': 'application/json' }, body: JSON.stringify({ title }), });
	};
	return (
		<form onSubmit={handleEdit}>
			<label>Editar Título do Post</label>
		<input value={title} onChange={(e) => setTitle(e.target.value)} />
		<button type="submit">Salvar Alterações</button> </form> ); }
```