MIT em Desenvolvimento Full Stack

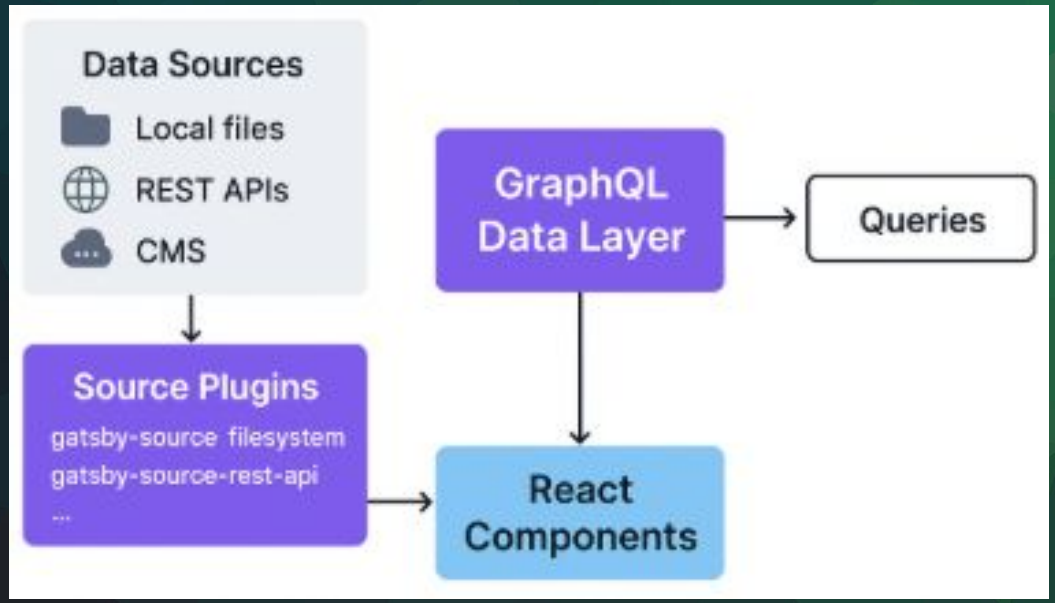
Front-end Jamstack com Gatsby

Agenda

Aula 7: Integração de Dados em Gatsby.

- Aprofundando GraphQL Data Layer.
- Convenções do Gatsby.
- GraphQL: Language Feature Support.

Aprofundando GraphQL Data Layer

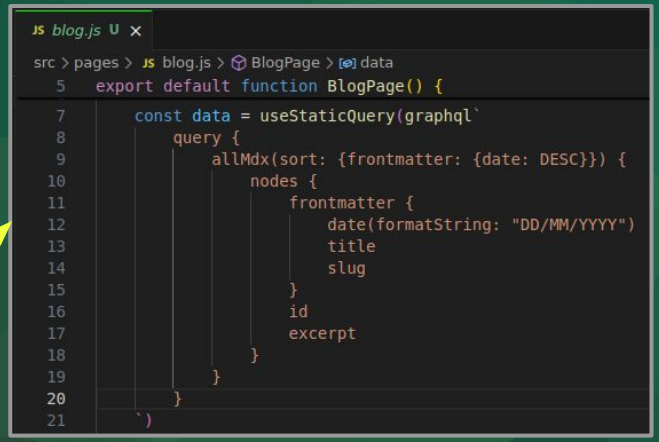


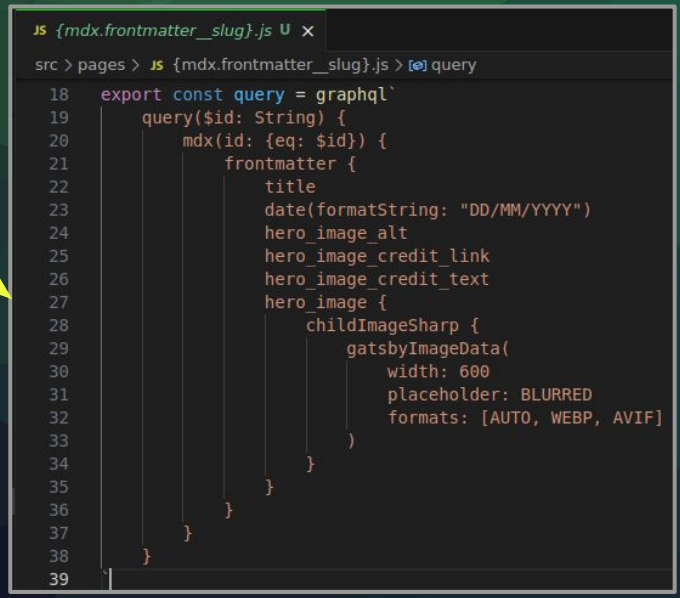
GraphQL Data Layer é um sistema unificado de acesso a dados.

Em vez de você buscar dados diretamente de APIs, arquivos ou bancos de dados no seu código React, o Gatsby coleta todas essas fontes e centraliza tudo em um grafo de dados interno acessível via GraphQL.

1. Durante o build o **Gatsby** executa todos os plugins como **gatsby-source-filesystem**, **gatsby-source-contentful**, **gatsby-source-firestore** etc.
2. Cada plugin coleta dados de sua fonte, como arquivos, CMS, API, etc.
3. Esses dados são convertidos em nós **GraphQL** e salvos num banco interno, usando o sistema de cache do **Gatsby**.
4. O **Gatsby** então constrói um esquema **GraphQL** automático, baseado nos tipos de dados coletados.
5. A partir daí, você pode fazer consultas **GraphQL** diretamente dentro dos componentes, usando **pageQuery** ou **useStaticQuery** para pegar os dados como se todos viessem de uma única fonte.


gatsby-config.js M
gatsby-config.js > ...
/**
* @type {import('gatsby').GatsbyConfig}
*/
module.exports = {
siteMetadata: {
title: `Estudos do Jamstack com Gatsby`,
siteUrl: `https://www.infnet.edu.br`,
},
plugins: [
{
resolve: "gatsby-source-filesystem",
options: {
name: `blog`,
path: `${__dirname}/blog`,
},
},
"gatsby-plugin-mdx",
"gatsby-plugin-image",
"gatsby-plugin-sharp",
"gatsby-transformer-sharp",
],
}


blog.js U
src > pages > blog.js > BlogPage > data
export default function BlogPage() {
const data = useStaticQuery(graphql`
query {
allMdx(sort: {frontmatter: {date: DESC}}) {
nodes {
frontmatter {
date(formatString: "DD/MM/YYYY")
title
slug
}
id
excerpt
}
}
}
`)


{mdx.frontmatter__slug}.js U
src > pages > {mdx.frontmatter__slug}.js > query
export const query = graphql`
query($id: String) {
mdx(id: {eq: $id}) {
frontmatter {
title
date(formatString: "DD/MM/YYYY")
hero_image_alt
hero_image_credit_link
hero_image_credit_text
hero_image {
childImageSharp {
gatsbyImageData(
width: 600
placeholder: BLURRED
formats: [AUTO, WEBP, AVIF]
)
}
}
}
}
}

	Consulta Estática → `useStaticQuery`	Consulta Dinâmica → `pageQuery`
Onde usar	Em qualquer componente React	Somente em componentes de página
Quando roda	Em tempo de build	Em tempo de build
Dados dependem da rota	❌ Não	✅ Sim
Sintaxe	Hook React	Exportação de constante `query`
Ideal para	Dados globais (site info, layout)	Dados específicos por página (posts, produtos)
Dica	Use useStaticQuery para dados que se repetem em todas as páginas - layout, rodapé, navbar.	Use pageQuery para dados específicos de cada página - artigo, produto, autor, etc.

Quando você cria arquivos dentro de src/pages/, o Gatsby:

1. Detecta automaticamente esses arquivos (React components);
2. Cria uma page query context para cada um;
3. Gera as rotas automaticamente com base no nome do arquivo (ex: src/pages/about.js → /about);
4. Constrói o data layer (via GraphQL) usando as fontes declaradas no gatsby-config.js.

O que acontece quando aparece uma rota `{mdx.frontmatter__slug}.js`?

1. Consulta Interna: O Gatsby primeiro olha o tipo de nó que você especificou (neste caso, mdx). Ele faz uma consulta interna para encontrar todos os nós do tipo mdx (semelhante ao que você fez no seu blog.js).
2. Criação de Páginas (Loop): Para cada nó mdx que ele encontra, o Gatsby decide criar uma página web.
3. Definição da URL: Ele usa a segunda parte do nome do arquivo, frontmatter__slug, para determinar qual será a URL da página. Ele extrai o valor do campo slug dentro do frontmatter de cada nó.
4. Injeção de Contexto (aqui está o $id): Ao criar a página para um nó mdx específico (por exemplo, o nó com slug: "meu-primeiro-post"), o Gatsby automaticamente passa o id desse nó específico como uma variável de "contexto" para o template.

O uso de dois sublinhados (__) é a sintaxe que o Gatsby escolheu para representar o aninhamento de objetos (object nesting) em GraphQL.

Vamos quebrar o nome do arquivo: {mdx.frontmatter__slug}.js

{mdx...}: O mdx antes do ponto (.) é o Tipo de Nó (o tipoDeNó). Você está dizendo ao Gatsby: "Olhe para todos os nós do tipo mdx".

...frontmatter__slug}: A parte depois do ponto é o Caminho do Campo (o campo).

Se o slug estivesse diretamente na raiz do mdx (o que não é comum), o nome do arquivo seria {mdx.slug}.js.

No entanto, o seu slug está dentro do objeto frontmatter.

```
mdx {
  info {
    meta {
      url_slug
    }
  }
}
```

```
{mdx.info__meta__url_slug}.js
```

Convenções do Gatsby

Convenção	Onde é Usada	Propósito (O que faz)
`src/pages/qualquer-nome.js`	Diretório src/pages/	**Criar Páginas Estáticas:** Automaticamente cria uma página estática. Ex: sobre.js $\rightarrow$ /sobre/
`{tipoDeNo.campo}.js`	Diretório src/pages/	**Criar Páginas Dinâmicas:** (File System Route API). Cria uma página para *cada* nó GraphQL encontrado. Ex: {mdx.frontmatter__slug}.js
`export default ...`	Arquivos de página/template	**Definir o Componente da Página:** É o componente React que será renderizado como o corpo (HTML) da página.
`export const query`	Apenas em templates de página	**Buscar Dados da Página (Dinâmica):** É a "query de página" que aceita variáveis (como $id) para buscar dados *específicos* para aquela página.
`export const Head`	Arquivos de página/template	**Definir o <head> da Página:** (API Head). Permite definir o <title>, metatags, etc., de forma otimizada para a página.
`useStaticQuery`	*Dentro* de qualquer componente	**Buscar Dados (Estática):** Um Hook do React para buscar dados que *não* dependem de variáveis de página (ex: buscar *todos* os posts).

	Onde é Usada	Propósito (O que faz)
`gatsby-config.js`	Raiz do projeto	**Configuração Principal:** Define metadados do site e, o mais importante, ativa e configura os **plugins** do Gatsby.
`gatsby-node.js`	Raiz do projeto	**API de "Build" (Node.js):** Permite criar páginas programaticamente (função createPages), modificar o GraphQL (onCreateNode), etc.
`gatsby-browser.js`	Raiz do projeto	**API do Navegador (Cliente):** Permite "interceptar" eventos do lado do cliente, como navegação (onRouteUpdate) ou adicionar wrappers globais.
`gatsby-ssr.js`	Raiz do projeto	**API de Renderização no Servidor (SSR):** Permite modificar o HTML estático final durante a "build" (ex: adicionar scripts/estilos ao <body> com onRenderBody).

GraphQL: Language Feature Support

Extensão: GraphQL: Language Feature Support

GraphQL: Language Feature Support

GraphQL Foundation graphql.org | 2.540.185 | (43)

GraphQL LSP extension that adds autocompletion, validation, go to definition, hov...

Desabilitar | Desinstalar | Atualização Automática

DETALHES | RECURSOS | LOG DE MUDANÇAS | DEPENDÊNCIAS

Changelog | Discord Channel | LSP Server Docs

GraphQL extension for VSCode built with the aim to tightly integrate the GraphQL Ecosystem with VSCode for an awesome developer experience.

`.graphql`, `.gql` file extension support and `gql`/`graphql` tagged template literal support for tsx, jsx, ts, js

- syntax highlighting (provided by `vscode-graphql-syntax`)
- autocomplete suggestions
- validation against schema
- snippets (interface, type, input, enum, union)
- hover support
- go to definition support (input, enum, type)
- outline support

Getting Started

> **This extension requires a graphql-config file.**

Marketplace

Identific...	graphql.vscode-graphql
Versão	0.13.2
Publicado	7 anos atrás
Último Lançam...	7 meses atrás

Categorias

Programming Languages

Linters

Recursos

Repositório

Problemas

Licença

JS .graphqlrc.js U

JS .graphqlrc.js > ...

```
module.exports = {
  schema: ".cache/schema.graphql",
  documents: ["src/**/*.{js,jsx,ts,tsx}"],
};
```

JS blog.js U

src > pages > JS blog.js > TypeScript > BlogPage > data

```
export default function BlogPage() {
    const data = useStaticQuery(graphql`
        query {
            allMdx(sort: {frontmatter: {date: DESC}}) {
                nodes {
                    frontmatter {
                        date(formatString: "DD/MM/YYYY")
                        title
                        slug
                    }
                    id
                    excerpt
                }
            }
        }
    `)
    const posts = data.allMdx.nodes;
```

JS {mdx.frontmatter__slug}.js U

src > pages > JS {mdx.frontmatter__slug}.js > TypeScript > query

```
import { graphql } from "gatsby";
import Layout from "../componentes/layout";
import { GatsbyImage, getImage } from "gatsby-plugin-image";
            <h2>{data.mdx.frontmatter.title}</h2>

export const query = graphql`
    query($id: String) {
        mdx(id: {eq: $id}) {
            frontmatter {
                title
                date(formatString: "DD/MM/YYYY")
                hero_image_alt
                hero_image_credit_link
                hero_image_credit_text
                hero_image {
                    childImageSharp {
                        gatsbyImageData(
                            width: 600
                            placeholder: BLURRED
                            formats: [AUTO, WEBP, AVIF]
                        )
                    }
                }
            }
        }
    }
`
```