REACT ROCKETSEAT IGNITE LAB - 2

19 DE JUNHO DE 2022 → Apresentação do Evento:

Ferramentas: React (Vite), TypeScript, Tailwind, GraphQL (REST - API), Apolo (Client) GraphQL, GraphCMS (painel de administração para os dados - front), GraphQL CodeGen.

Projeto: **Plataforma de Evento → Cadastro e acesso a um conteúdo (evento) em video.**

20 DE JUNHO DE 2022 → Aula 1 → Setup Inicial

21 DE JUNHO DE 2022 → Aula 2 → Estrutura visual do Projeto

22 DE JUNHO DE 2022 → Aula 3 → Roteamento e Player de Aula

23 DE JUNHO DE 2022 → Aula 4 → Inscrição via GraphQL

24 DE JUNHO DE 2022 → Aula 5 → Deploy, CodeGen

GraphQL, GraphCMS → será o banco de dados da aplicação

React:

- Biblioteca para fazer a construção de componentes
- Componentes são funções que retornam algum tipo de HTML
- Unindo os componentes, forma a interface do app

tsx → TypeScript + JSX (XML dentro de JS)

dom → árvore de elementos dentro do HTML → WEB

Extensões do VSCode:

GraphQl

Tailwind CSS intelliSense

PostCSS Language Support

Ferramentas e Criação:

Criando o projeto com Vite: (pode ser feito com o Yarn)

- `npm create vite@latest`
- Nome do projeto: event-platform
- Seleção do framework: reactTs
- `cd event-platform`
  `npm install`
  `npm run dev`

Instalando o Tailwind:

- `npm i tailwindcss postcss autoprefixer -D`
  - postcss → automatizar ferramentas dentro do css (Vite utiliza)
- `npx tailwindcss init -p`
- Dentro do arquivo tailwind.config.js → content: []
  - Passar quais arquivos vou utilizar o tailwind
  - `content:[`
    `'./src/**/*.tsx'`

`]`

- Deletar todos os arquivos src/css
- Criar pasta src/styles
  - Criar arquivo global.css
    - setup do tailwind:

      @tailwind base;

      @tailwind components;

      @tailwind utilities;
- Importar esse arquivo de CSS no src/main.tsx

- Adiciona classes utilitárias no código - estilização com `className="`*ctrl+space mostra todas as classes*`"` - interface declarativa;

GraphCMS: (Content Management Sistem)

Acessar:

https://app.graphcms.com/

Clonar o projeto pronto do Ignite

Gerencia de conteúdo da aplicação → conteúdo rotativo → Headless CMS: traz apenas o painel de ADMIN (dados fornecidos através de uma API REST oi GraphQL →

React que consome essa API do CMS → chamadas

Projeto →

Schema →

models →

Entidade (Tabela) → Ex: Aula |

Campos: Informações que serão armazenadas → Ex: dentro da tabela Aula: Titulo | Descrição | Video

Slug → link adaptado da barra de navegação

Content → cadastrar novos dados nas Entidades

API Playground (como o Postman) → testar chamadas da API → cria as API de forma automatizada

SLQ:

query (buscar dados):

`query {`

Quais dados eu quero buscar - (na esquerda, estarão todos os dados da API)

`}`

mutation (criar, alterar ou deletar dados)

Como integrar o projeto front-end com a API GraphCMS →

- projeto
  - → configurações do projeto
  - → API Acess
  - → Content API (url de acesso aos conteúdos da API)
    - Quem tiver essa URL, consegue acessar os dados conforme as permissões
  - Precisa ajustar as permissões (Create Permission)
    - state : published → não acessar os rascunhos cadastrados
  - Copia a URL e utilizar o Apollo

Apollo:

Requisições na API (como o Axios) → identifica duas requisições iguais.

- Instala o Apolo e o GraphQL :

  `npm i @apollo/client graphql`

- Cria uma pasta scr/lib
  - criar arquivo apollo.ts → Arquivo de configuração do apollo: **ou no arquivo .env.local → mais seguro → ver no assunto Mutations - variável ambiente**

(mais abaixo)

```
import { ApolloClient, InMemoryCache } from "@apollo/client";

export const client = new ApolloClient({
  uri: "*****",
  cache: new InMemoryCache(),
});
```

- Como fazer a Requisição dentro do React App.tsx:
  - Com o useEffect:

```
import gql from "graphql-tag";
import { useEffect } from "react";
import { client } from "./lib/apollo";

const GET_LESSONS_QUERY = gql`
  query {
    lessons {
      id
      title
    }
  }
`;

function App() {
  useEffect(() => {
    client
      .query({
        query: GET_LESSONS_QUERY,
      })
      .then((response) => {
        console.log(response.data);
      });
  }, []);

  return <h1 className="text-5xl font-bold">Hello World</h1>;
}

export default App;
```

  - Com o useQuery: Hook do React
    - Arquivo App.tsx:

```
import { gql, useQuery } from "@apollo/client";

const GET_LESSONS_QUERY = gql`
  query {
    lessons {
      id
      title
    }
  }
`;

function App() {
  const { data } = useQuery(GET_LESSONS_QUERY);
  console.log(data);

  return <h1 className="text-5xl font-bold">Hello World</h1>;
}

export default App;
```

- Arquivo main.tsx:

```
import { ApolloProvider } from "@apollo/client";
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
import { client } from "./lib/apollo";
import "./styles/global.css";

ReactDOM.createRoot(document.getElementById("root")!).render(
  <React.StrictMode>
    <ApolloProvider client={client}>
      <App />
    </ApolloProvider>
  </React.StrictMode>
);
```

→ Componentes:

- funcionarem de forma isolada e separadas → pedacinhos que, quando juntos, formam a interface.
- Criar um componente quando algo se repete muitas vezes ou quando é possível desacoplar algo (algo que não interfere em mais de um lugar da interface/ Projeto)

- Criar pasta: src/ components
- Tudo no App.tsx é para tudo da aplicação → criar pasta src/pages para separar as páginas

Mudar cores do Tailwind:

- Dentro do arquivo tailwind.config.js →

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: ["./src/**/*.tsx"],
  theme: {
    extend: {
      fontFamily: {
        sans: "Roboto, sans-serif",
      },
      colors: {
        green: {
          300: "#00B37E",
          500: "#00875F",
          700: "#015F43",
        },
        blue: {
          500: "#81D8F7",
        },
        orange: {
          500: "#FBA94C",
        },
        red: {
          500: "#F75A68",
        },
        gray: {
          100: "#E1E1E6",
          200: "#C4C4CC",
          300: "#8D8D99",
          500: "#323238",
          600: "#29292E",
          700: "#121214",
          900: "#09090A",
        },
      },
    },
  },
  plugins: [],
};
```

Mudar fonte e avisar Tailwind:

- importar a fonte no link do index.html do projeto
- Dentro do arquivo tailwind.config.js → theme → extend → `fontFamily: { sans: "Roboto, sans-serif"` (exemplo acima)

Ícones:

Biblioteca de ícones:

phosphoricon.com

instalar no projeto: `npm i phosphor-react`

no arquivo do componente: `import { CheckCircle } from "phosphor-react";`

no componente, usar como `<CheckCircle/>`

Datas:

Formatação de datas no React

Biblioteca `npm i date-fns`

```
import { isPast, format } from "date-fns";
import ptBR from "date-fns/locale/pt-BR";


export function Lesson(props: LessonProps) {
  const isLessonAvailable = isPast(props.availableAt);
  const availableDateFormatted = format(
    props.availableAt,
    "EEEE • 'd' de 'MMMM' • 'k'h'mm",
    { locale: ptBR }
  );
```

- verifica se a data ja passou, se esta no passado.
- formata a data
- traduz para pt-BR

Player video:

Formatação de video

Biblioteca vimejs.com

`npm i @vime/core @vime/react` não funciona no rect 18

`npm i @vime/core @vime/react —-force`

```
import { DefaultUi, Player, Youtube } from "@vime/react";

import "@vime/core/themes/default.css";

export function Video() {
  return (
    <div className="flex-1">
      <div className="bg-black flex justify-center">
        <div className="h-full w-full max-w-[1100px] max-h-[60vh] aspect-video">
          <Player>
            <Youtube videoId="Ox_zb2cs9zM" />
            <DefaultUi />
          </Player>
        </div>
      </div>
```

Background com tailwind:

dentro de Tailwind-config

```
theme: {
    extend: {
      backgroundImage: {
        blur: "url(/src/assets/blur-background.png)",
      },
```

no componente:

```
export function Subscribe() {
  return <div className="min-h-screen bg-blur"></div>;
}
```

→ Propriedades:

- Componentes maleáveis, que recebam mais de um conteúdo → condicionais a alguma informação → propriedades são formas de alterar comportamento ou visual de componente, baseado em informação que enviamos à ele.

→ no typescript: ideal criar uma interface de types, com as informações que serão alteradas na aplicação

ex:

```
interface LessonProps {
title: string;
slug: string;
avaliableAt: Date;
type: "live" | "class";
}
```

→ passar a props como parâmetro da função do componente (para ser usado dentro da função):

ex: `export function Lesson( props : LessonProps )`

ex dentro da função: `<strong > { props .title} </strong>`

```
import { CheckCircle, Lock } from "phosphor-react";
import { isPast, format } from "date-fns";
import ptBR from "date-fns/locale/pt-BR";
interface LessonProps {
title: string;
slug: string;
availableAt: Date;
type: "live" | "class";
}
export function Lesson( props : LessonProps ) {
const isLessonAvailable = isPast( props .availableAt);
const availableDateFormatted = format(
props .availableAt,
"EEEE • 'd' de 'MMMM' • 'k'h'mm",
{ locale: ptBR }
```

`);`

→ Roteamento

Aplicação ter permissão para ter várias páginas (acessíveis por endereços diferentes)

mudança da URL

- Biblioteca: React Router DOM `npm i react-router-dom`
- Criar um arquivo src/**Router.tsx** na raiz do projeto (junto com App)

```
import { Route, Routes } from "react-router-dom";
import { Event } from "./pages/Event";

export function Router() {
  return (
    <Routes>
      <Route path="/" element={<h1>Home</h1>} />
      <Route path="/event" element={<Event />} /> //importa o componente
    </Routes>
  );
}
```

path=”” → qual caminho que eu quero que a rota pertença

element={} → qual elemento eu quero mostrar em tela quando a pessoa estiver nessa rota

- No arquivo **App.jsx**:

OBS: Trouxe o Apollo Provides pra esse arquivo porque ele precisa ficar em volta da aplicação

```
import { ApolloProvider } from "@apollo/client";
import { client } from "./lib/apollo";
import { Router } from "./Router";
import { BrowserRouter } from "react-router-dom";
function App() {
  return (
    <ApolloProvider client={client}>
      <BrowserRouter>
        <Router />
      </BrowserRouter>
```

```
    </ApolloProvider>
  );
}

export default App;
```

- No arquivo **main.jsx:**

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
import "./styles/global.css";

ReactDOM.createRoot(document.getElementById("root")!).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

- Nos componentes, o `<a href="">` que altera a rota muda para `<Link to={``}>`

```
< Link  to ={`/event/lesson/ ${ props .slug} `}  className ="group">
```

- na page:

```
import { useParams } from "react-router-dom";
```

```
const { slug } = useParams<{ slug:  string  }>();
```

→ Mutations

Qualquer operação que façamos com GraphQL, que não seja a leitura de dados

Permissões do GraphCMS: → No projeto CMS → configurações

- Public Content API → Somente para leitura de dados
- Permanent auth token → Escritas de dados
  - → na criação (management API → Selecionar NO → assim não criará novos schemas)
  - → na criação → **Would you like us to initialize some defaults? yes**
  - Create permission → seleciona a tabela → create

- copia o Token value *******
- Token → variáveis que queremos esconder → variáveis ambiente
- Variáveis ambiente no Vite:
  - criar arquivo .env.local
    - **VITE**_API_ACCESS_TOKEN = **********
  - Adiciona o arquivo .env.local no git.ignore
  - No arquivo Apollo.ts:

```
import { ApolloClient, InMemoryCache } from "@apollo/client";

export const client = new ApolloClient({
  uri: import.meta.env.VITE_API_URL,
  headers: {
    Autorization: `Bearer
${import.meta.env.VITE_API_ACCESS_TOKEN}`,
  },
  cache: new InMemoryCache(),
});
```

```
import { gql, useMutation } from "@apollo/client";
import { useState, FormEvent } from "react";
import { useNavigate } from "react-router-dom";
import { Logo } from "../components/Logo";

const CREATE_SUBSCRIBER_MUTATION = gql`
  mutation CreateSubscriber($name: String!, $email: String!) {
    createSubscriber(data: { name: $name, email: $email }) {
      id
    }
  }
`;

export function Subscribe() {
  const navigate = useNavigate();

  const [name, setName] = useState("");
  const [email, setEmail] = useState("");
  const [createSubscriber] = useMutation(CREATE_SUBSCRIBER_MUTATION);

  async function handleSubscribe(event: FormEvent) {
    event?.preventDefault();
    await createSubscriber({
      variables: {
```

```
        name,
        email,
      },
    });

    navigate("/event");
  }
```

→ CodeGen:

Ferramenta para GraphQL dentro do ecossistema do REACT

https://www.graphql-code-generator.com/

https://www.graphql-code-generator.com/docs/guides/react#apollo-and-urql

Vamos utilizar para identificar erros nas requisições da API / Não precisar criar sempre query e interface de tipagem

A tipagem já existe dentro do GraphCMS

O GraphQL Code Generator é uma ferramenta baseada em plug-ins que ajuda você a tirar o melhor proveito da sua pilha GraphQL.

Do back-end ao front-end, o GraphQL Code Generator automatiza a geração de:

- **Consultas digitadas, mutações e, assinaturas**

  para React, Vue, Angular, Next.js, Svelte, se você estiver usando Apollo Client, URQL ou React Query.

- **Resolvedores GraphQL digitados**

  , para qualquer servidor Node.js (GraphQL Yoga, Módulos GraphQL, TypeGraphQL ou Apollo) ou Java GraphQL.

- **SDKs Node.js totalmente tipados,**

  suporte Apollo Android

A força de seus tipos de aplicativos front-end é baseada em seus tipos de dados. Qualquer erro em seus tipos de dados mantidos manualmente ondula em muitos de seus componentes.

Por esse motivo, automatizar e gerar a digitação de suas operações do GraphQL melhorará a experiência do desenvolvedor e a estabilidade de sua pilha.

- Comandos no projeto:

`npm i @graphql-codegen/typescript @graphql-codegen/typescript-operations @graphql-codegen/typescript-react-apollo -D`

`npm install @graphql-codegen/cli -D`

- criar arquivo de configuração **codegen.yml** → na raiz, junto com index.html

```
schema: URL DA API DO GRAPHQL
documents: "./src/graphql/**/*.graphql"
generates:
  ./src/graphql/generated.ts:
    plugins:
      - typescript
      - typescript-operations
      - typescript-react-apollo
    config:
      reactApolloVersion: 3
      withHooks: true
      # criará um hook automatico para cara query ou mutation
      withHOC: false
      withComponent: false
```

- Criar uma pasta scr/graphql
- Criar pastas: scr/graphql/queries

  scr/graphql/mutations
- Retirar todas as queries e mutations (e as interfaces referentes) do projeto e colocar em arquivos .graphql, dentro destas pastas. Todas precisarão ter nome.

- no package.json:

```
"scripts": {
    "dev": "vite",
    "build": "tsc && vite build",
    "preview": "vite preview",
    "codegen": "graphql-codegen"
  },
```

- No terminal `npm run codegen` → vai criar o arquivo de output → toda a API traduzida para Typescript
- Ajustar o código, substituindo as queries e mutations com as funções use(Hook das queries e mutations que criamos)

→ Deploy :

- Repositório GitHub

gh repo create (criar repositorio no github)

commit

push

- Deploy no Vercel.com
- importa o projeto
- Em FrameWork: vite
- Em Environment Variables: as variaveis de ambiente que criamos
- Deploy
- deixar as imagens sempre em uma pasta public (junto ao src)