front-end

prof. Lucas Ferreira

engenharia de software

engenharia de computação

Interagindo com dados online

Temos que integrar o front com o back, certo?

Hello *"back-end"* World

Partindo da ideia de que estamos desenvolvendo a camada de front-end, em algum momento teremos que conectar em algum tipo de "back-end", certo?

Já vimos nas aulas de JavaScript básico algumas forma de fazer isso, inicialmente precisamos que o JavaScript *(seja ele React ou não)*, exerça uma conexão com qualquer "endpoint" on-line, aguarde uma respostas e utilize esses dados como achar melhor.

Normalmente usamos técnicas baseada em **AJAX** e principalmente o método `***fetch***` para fazer essa conexão on-line.

Hello *"back-end"* World

Não há nada de errado com esse princípio, podemos fazer algo assim dentro de um componente por exemplo:

```
const [lista, setLista] = useState([]);

// ao abrir o componente, carregue a lista de uma
// fonte online
useEffect(() => {
  // requisitando a URL
  fetch('https://lista-front-api.burn-cloudflare-account.workers.dev/')
    .then((response) => response.json()) // convertendo a resposta para JSON
    .then((json) => setLista(json)); // setando nossa lista para o React usar
}, []);
```

Hello *"back-end"* World

No código anterior, apesar de tradicional e funcional, perdemos alguns pontos importantes, como:

- Gerenciamento de erros
- Carregamento "em duplicidade"
- Falta de cache interno
- Falta de controle de estado pendente *(loading)*
- Entre outras coisas...

Hello *"back-end"* World

Indicando um caminho mais viável em minha opinião seria o uso de bibliotecas especializadas em consumo de dados conectados em componentes de React.

Por hora vou indicar duas para vocês:

- **TanStack Query** *(antiga react-query)*: https://tanstack.com/query
- **SWR**: https://swr.vercel.app/pt-BR

TanStack Query

Todas as duas bibliotecas são muito boas, fico até em dúvida em qual mostrar para vocês.

Por hora vou optar pela "antiga" React Query, que hoje se chama **TanStack Query** *(vou explicar essa baboseira de nomes)*.

E vamos começar todos por aqui:

https://tanstack.com/query/latest/docs/react/quick-start

TanStack Query

Para fins de testes, vamos retornar ao projeto da lista de compras visto na aula passada:

- **Turma 1-A de Terça-feira**:
  https://stackblitz.com/edit/demo-react-router-1a

- **Turma 1-B de Quinta-feira**:
  https://stackblitz.com/edit/demo-react-router-1b

TanStack Query

Também iremos utilizar um pequeno serviço/api on-line muito simplória que criei para salvarmos os itens da lista de compras on-line.

Este serviço fica no endereço abaixo:

https://lista-front-api.burn-cloudflare-account.workers.dev/?db=seunome

TanStack Query

Para iniciarmos precisamos primeiro instalar o pacote como adicional de nosso projeto com um dos comandos abaixo:

```
$ npm i @tanstack/react-query
# or
$ yarn add @tanstack/react-query
```

Depois iremos seguir o guia oficial para iniciarmos nossa implementação:

https://tanstack.com/query/latest/docs/react/quick-start

Evoluindo nossos projetos

e também o que conhecemos de React

Evoluindo nossos projetos

Primeiro de tudo, gostaria de destacar que nada do que fizemos até aqui em relação ao React.js é "errado".

Projetos 100% front-end e sem mecânicas de servidor complicadas são 100% válidos e tem seu uso garantido.

Porém uma vez que o cenário de front-end tenha avançado no campo de uso de JavaScript e interatividade, fez-se necessário novas mecânicas na "entrega de conteúdo" ao usuário.

Evoluindo nossos projetos

Projetos 100% front-end quando muito evoluídos, ou pesadamente construídos, por vezes podem apresentar respostas lentas de carregamento inicial e também apresentar barreiras de SEO e até impossibilitar acessos em conexões lentas OU com JavaScript desabilitado.

Frente a estes e muitos outros cenários, uma vez que o React.js seja uma opção que a maioria dos desenvolvedores de front-end não pensam em trocar, o time principal do projeto junto de outros grandes players de mercado, começou a buscar alternativas de "agilizar" e "melhorar" a "entrega" de conteúdos e elementos desenvolvidos em React.

Evoluindo nossos projetos

O primeiro passo foi possibilitar a renderização de React também no servidor *(não só no navegador do usuário)*. Chamamos isso, inicialmente, de **SSR** *(server side rendering)*.

Só com esse pequeno avanço, conseguimos entregar uma resposta inicial ao usuário já adiantada, não mais uma tela em branco esperando um "calhamaço" de JS ser carregado antes de criar algo interativo.

Na sequência o próximo caminho buscado foi criar interações mais complexas ao ponto de facilitar a conversa entre o front-end e o back-end na troca de dados dinâmicos, como dados de usuários, listas de itens, formulários e tc.

Frameworks completos para React

Essa busca pela constante evolução levou a criação de grandes frameworks completos, disponíveis de uso open-source e todos baseados *(ou em torno)* do React.js.

Novamente, o React em si e o que aprendemos, **não está errado**. O conceito de componentização *(através de suas funções)* e também de inteligência *(hooks)* está em dia e será usado em outros cenários.

Mas dada a necessidade de evoluir para "algo mais" surgiram frameworks super interessantes como **Gatsby**, **Remix.run** e **Next.js**.

Frameworks completos para React

Vou acessar o site de cada projeto e falar brevemente sobre eles, mas para nosso conteúdo em sala de aula iremos adotar o **Next.js** como opção principal para finalizar nosso conteúdo de React.

Acredito ser melhor explicá-lo pois ele é mais popular e têm mais opções de "vagas" disponíveis no mercado de trabalho.

Next.js

"The React Framework"

Disclaimer

Assim com o Vite.js e o CRA, que não se tratam de uma "nova linguagem" ou "tecnologia diferenciada", o Next.js *(mesmo sendo um framework)* é considerado uma ***FERRAMENTA*** que "orbita" o ecossistema do React.js.

Então todo o nosso conhecimento de ***Components***, ***Hooks***, ***Context***, ***States***, ***Reducers***, ***Events*** e etc. se aplica diretamente em qualquer projeto, seja um projeto que use um setup 100% customizado, ou use Vite.js ou use até Next.js.

Apresentando: *Next.js*

Apresentado como *"the React Framework"*, o **Next.js** disponibiliza um setup completo para solucionar problemas como:

- Code splitting *(carregamento parcial de acordo com os recurso solicitados)*
- Versões do projeto otimizados para produção
- Pré-renderização de páginas estáticas e otimização para SEO
- Integração contínua com o back-end/server-side em um único projeto

Uma vez que ele resolve no mínimo todos os problemas acima, já podemos colocar o Next.js como uma solução completa em torno do React.js, caracterizando assim um framework *(e não apenas uma biblioteca que busca resolver apenas uma tarefa)*.

Apresentando: *Next.js*

Outras vantagens de usar ***Next.js*** em um projeto:

- Um sistema de navegação entre páginas já de fábrica
- Pré-renderização, seja geração de conteúdo estático (SSG) OU renderização em tempo real (SSR)
- Suporte de fábrica a CSS e SASS *(assim com o Vite.js/CRA)*
- Suporte a fast refresh/reload *(assim com o Vite.js/CRA)*
- Possibilidade embutir rotas exclusivas para execução no servidor (APIs por exemplo) com funções simplificadas
- É totalmente extensível e customizável

Iniciando um projeto

com Next.js

Setup / Primeiro projeto

Se sua máquina estiver preparada para rodar CRA (create react app), com Node.js e etc, ela também já está preparada para rodar projetos baseados em Next.js.

Para iniciar seu **primeiro projeto** navegue até a pasta alvo e rode o comando abaixo:

```
npx create-next-app@latest meu-projeto-next --use-npm
```

O termo "*meu-projeto-next*" é o nome da pasta do novo projeto a ser criado. E o parâmetro (opcional) `--use-npm` é uma forma de optar obrigatoriamente pelo npm ao invés de outro gerenciador de pacotes *(como Yarn por exemplo)*.

Primeiro projeto

O comando anterior irá baixar todo o necessário dentro da pasta *node_modules* de nosso novo projeto para que o projeto rode perfeitamente tendo suporte a React e tudo mais que o *Next.js* nos oferece.

Após isso podemos navegar até a pasta de nosso projeto, e rodar o comando de início:

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

Isto irá disponibilizar o nosso projeto para acesso em tempo real no endereço:

http://localhost:3000

Primeiro projeto

Assim como no Vite.js, teremos uma nova página de "bem vindo ao next.js" a disposição.

Aí de agora em diante podemos começar a editar o projeto e visualizarmos em tempo real *(a medida que vamos editando e salvando)* as mudanças em nosso navegador.

Sistema de páginas

ou como funcionam as rotas "mágicas"

Páginas em um projeto Next.js

Logo de cara quando abrimos a pasta de nosso projeto Next.js em um editor de código, podemos perceber que o Next.js sugere uma organização de arquivos um pouco mais "complexa" do que o Vite.js/CRA.

Diferente do Vite.js em que temos a liberdade de organizar quase tudo do nosso jeito *(nomes de arquivos, pastas e etc)* dentro da pasta **src**, o Next.js nos dá um pouco menos de liberdade, em troca de fazermos menos configurações manuais.

Sendo assim, deixo logo de cara que: todas as telas "brutas" ficam dentro da pasta ***"pages"*** e tudo que for de estilo/css deve ficar dentro da pasta ***"styles".*** O restante de arquivos como imagens, vídeos, fontes e etc devem ficar dentro da pasta ***"public"***.

Páginas em um projeto Next.js

Dentro de sua "inteligência" o Next.js organiza nossas rotas/URLs de cada página/tela de seguinte forma:

Se criarmos um arquivo de uma tela em ***"pages/teste.js"***, poderemos acessar este mesmo arquivo em nosso navegador através do endereço: http://localhost:3000/teste.

Caso precisarmos de uma "árvore de links" mais detalhada, por exemplo um CRUD contendo criação, listagem e edição, podemos seguir este esquema:

- ***"pages/clientes/index.js"*** -> para a listagem -> http://localhost:3000/clientes
- ***"pages/clientes/create.js"*** -> para criação de um novo cliente -> http://localhost:3000/clientes/create

Páginas em um projeto Next.js

Como podemos ver, basta criar o arquivo com o nome certo, no lugar certo, que uma "nova tela" passa a existir.

Não precisamos configurar rotas, nem adicionar um *React Router* em nossa aplicação, o Next.js adianta tudo isso para nós.

E antes de avançarmos, caso precise editar globalmente alguma parte da aplicação *(ex: criar um topo ou rodapé que apareça em todas as páginas)* podemos editar o arquivo ***"pages/_app.js"***.

Páginas em um projeto Next.js

Para criar um link *(ou botão)* de navegação que jogue de uma página para outra, é indicado que usemos o component **Link** *(assim como no react-router)* só que neste caso o mesmo virá do próprio Next.js, e deverá conter sempre uma tag **<a>** dentro dele *(ou outra tag clicável)*:

```
import Link from 'next/link';

return (
  <Link href="/clientes/create">
    <a>Adicionar novo Cliente</a>
  </Link>
);
```

Páginas em um projeto

Caso tenhamos a necessidade de mudar de página de forma programática, podemos usar o hook "***useRouter***" também disponibilizado pelo próprio Next.js:

```
import { useRouter } from 'next/router'

export default function IndexPage() {
  const router = useRouter();

  const handleClick = (event) => {
    event.preventDefault()
    router.push("/clientes/create");
  };

  return (
    <div>
      Hello World.{" "}
      <button onClick={handleClick} type="button">
        Criar novo Cliente
      </button>
    </div>
  );
};
```

Rotas dinâmicas

ou quando as coisas precisam escalar

Rotas dinâmicas

Já vimos que criar páginas "estáticas" é bem fácil.

Porém em algum momento teremos que escalar isso para um sistema de CRUD mais dinâmico, uma vez que não poderemos criar um arquivo .js cara cada subitem de nosso sistema. Por exemplo, em aulas passadas eu dei a ideia de termos uma listagem de produtos.

A ideia é ter uma única página que receba um parâmetro ***(ex: id do produto)*** e se adapte para cada detalhes/preços e etc de um produto selecionado.

Rotas dinâmicas

Complementando, para fazer essa tarefa de rotas dinâmicas no motor clássico do Next.js, precisaremos seguir um caminho um pouco mais "seguro" para definir quais produtos *(e seus parâmetros)* são válidos de receberem URLs dinâmicas.

Uma vez que essa ***página comum*** terá que servir dinamicamente para todo e qualquer produto que venha a existir, precisaremos criar uma rota/página que receba parâmetros em seu endereço a fim de descobrir qual produtos queremos visualizar ou editar *(por exemplo)*.

Rotas dinâmicas

Para cada "rota dinâmica" teremos que criar uma página .js baseada em **um nome meio estranho**.

Por exemplo, vamos supor que queiramos editar ou visualizar um produto, sendo assim criarmos um arquivo assim:

pages/produtos/edit/[id].js -> que nos daria uma rota mais ou menos assim -> ***localhost/produtos/edit/103***

Como podemos perceber esse nome estranho do arquivo "**[id].js**" é justamente o parâmetro dinâmico que vai ser inserido dentro da página para adaptarmos a mesma para cada único cliente (*ou outra informação)*.

Rotas dinâmicas

No caso da rota *localhost/produtos/edit/103* podemos receber o I**D de produto 103** dentro de nosso componente da seguinte forma:

```
export default function ProdutoEdit({ id }) {
  return <p>Editando: {id}</p>;
}

export async function getStaticPaths() {
  const paths = [{ params: { id: "103" } }];
  return {
    paths,
    fallback: false,
  };
}

export async function getStaticProps({ params }) {
  return {
    props: {
      id: params.id,
    },
  };
}
```

Outras ideias

para se fazer com Next.js

Criação de APIs

Vale a pena dar uma olhada na documentação básica caso queiramos fazer nossas próximas APIs:

https://nextjs.org/learn/basics/api-routes/creating-api-routes

Formulários

Também temos um guia simples de como lidar com formulários no sistema clássico de rotas/pages do Next.js:

https://nextjs.org/docs/pages/building-your-application/data-fetching/building-forms

Deploy do projeto

O Next.js oferece um jeito "fácil" de botarmos um projeto no ar oficialmente caso queiramos usar os servidores da **Vercel** *(dona do Next.js)*.

Segue a documentação simplificada do processo:

https://nextjs.org/learn/basics/deploying-nextjs-app/github

FOCO NO ABP!

vamos para mais um *checkpoint*

Não podemos esquecer do ABP!

Em nossa última movimentação oficial do ABP solicitei que após todas as equipes formadas e os temas definidos que fizessem um **repositório oficial no GITHUB** para cada projeto/trabalho.

De hoje em diante teremos mais 2 interações importantes antes da entrega oficial do trabalho:

Preciso de mais um checkpoint em cada ABP até o **domingo dia 11/06**.

O que é um checkpoint?

A maioria das equipes ainda está com seus repositórios vazios, praticamente só tem o README explicando o tema do trabalho.

Como eu não quero que vocês deixem tudo para última hora, cada checkpoint deve representar uma evolução significativa no código do projeto, uma grande "evolução".

Por exemplo, o **primeiro checkpoint** *(domingo, dia 11/06)* pode ser a criação estrutural da base do projeto e o início de algumas telas e sistema de navegação (para quem ainda não fez nada).

E o **segundo checkpoint** seria estar com o trabalho "quase lá" digamos que algo entre 80% a 90% do que vocês imaginam apresentar até o dia final de entrega do ABP.

obrigado 🚀