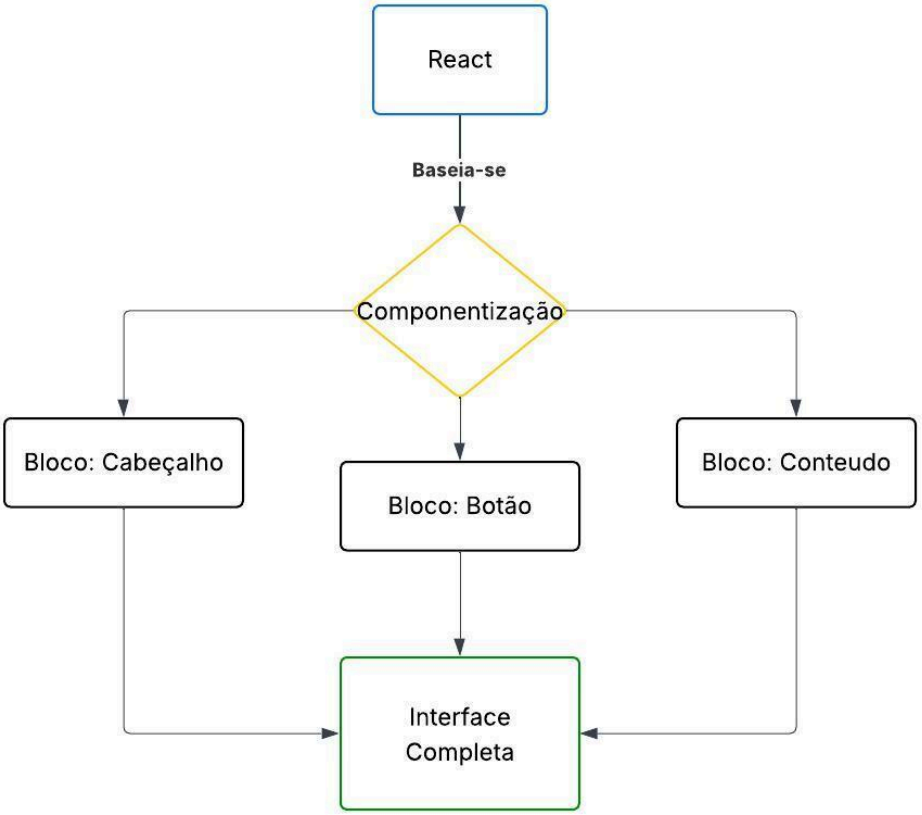
Relatório 4 - Prática: Fundamentos do React com TypeScript e Next.js (II)

José Carlos Seben de Souza Leite

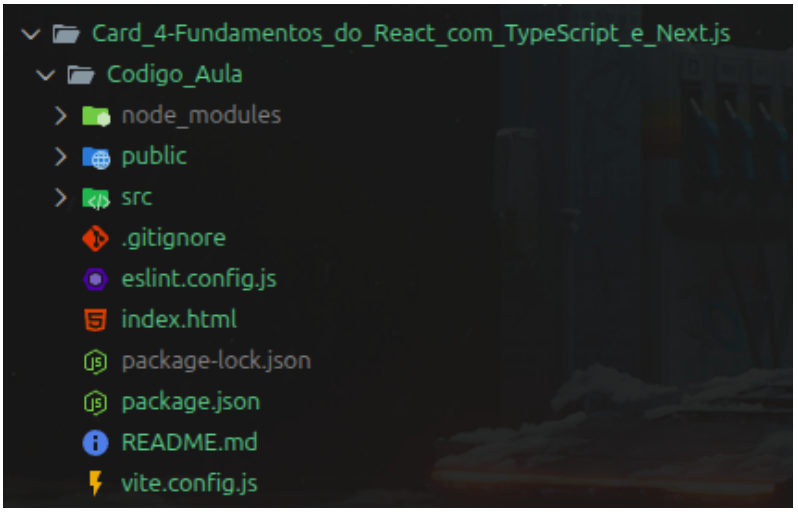
Descrição da atividade

Iniciando este novo card. com um novo conteúdo como no card. anterior fomos apresentados ao Type Script aqui teremos nossa primeira experiência com o React, mas antes de qualquer coisa precisamos saber o que é o React que basicamente é uma biblioteca criada pela meta para ser utilizado no desenvolvimento de interfaces de usuários interativas, para isso ele vem com algo chamado de componentização onde pegamos blocos de códigos separadas para cada coisa como um botão vai ter o código dele e posteriormente se precisarmos podemos reaproveitar esse componente sem precisar o programar novamente do zero, facilitando no desenvolvimento e também no trabalho em equipes por permitir que diferentes desenvolvedores trabalhem em diferentes componentes sem interferir um no trabalho do outro.



Agora saindo um pouco da teoria e começando com a parte prática é inicialmente passado para baixar as coisas como o Node e a NPM no nosso caso já havíamos instalado então podemos prosseguir com o curso com o essencial já instalado podemos iniciar um projeto em React e instalar pôr a própria NPM as dependências que faltavam para conseguirmos dar continuidade ao curso.

Com tudo baixado e o projeto criado teremos uma estrutura semelhante a esta:



Ao explorarmos esse projeto nos deparamos com algo curioso que é quando abrimos o arquivo **index.html** ele esta praticamente vazio comparado com as coisas que estão sendo exibidas na tela quando rodamos o comando “npm run dev” para iniciarmos nosso “site”, ai que nos é explicado com o React nos não adicionamos os textos diretamente com o HTML e o posicionamos na tela como o habitual, com o React nos adicionamos os textos com o próprio Java Script e o HTML “juntos” como nos vai ser mostrado no exemplo a seguir.

Neste exemplo a baixo é uma boa representação de como funciona com o React para inserir texto e, ao mesmo tempo, criar as funções que vão ter influência sobre esses textos.

```jsx
import { useState } from "react";

function App() {
  const [massage, setMassage] = useState("Olá, mundo!");

  return (
    <div>
      <h1>{massage}</h1>
      <button
        onClick={() => {
          setMassage("Olá, fui clicado!");
        }}
      >
        Mudar mensagem
      </button>
    </div>
  );
}

export default App;
```

O que temos aqui é basicamente uma função (relembrando que quando trabalhamos com React uma função basicamente é um componente) que adicionar um texto na tela “Olá, mundo!” e um botão logo abaixo onde quando clicado este botão o texto que estava sendo exibido ira mudar para “Olá, fui clicado!”. E como isso funciona, então basicamente se olharmos na primeira linha estamos importando o userState do próprio React, que permite que componentes “lembrem” de informações e as atualizem na tela sem precisar que nos

recarregamos a tela, assim basicamente causando uma reação do sistema a ação do usuário, com isso já conseguimos ter uma boa ideia de como vai funcionar o React que é basicamente reação.

Dando continuidade ao conteúdo compreendido no vídeo "**Curso de React para Completos Iniciantes**" onde no mesmo tivemos uma experiência completa da criação de uma interface onde o usuário pode interagir, adicionar, excluir e inspecionar tarefas, depois implementamos utilizando a própria memória dos navegadores para manter as alterações que realizamos em nosso gerenciador de tarefas, e para encerrar nos foi mostrado um pouco de como fazer e como funcionam as integrações com APIs onde nesse projeto utilizamos uma que tinha várias tarefas já prontas em um JSON para termos uma ideia de como funciona um projeto real em produção e suas integrações de APIs, e por último é mostrado como fazer o deploy deste projeto, primeiro mostrando alguns comandos básicos do git para versionamento e depois usando o próprio Node para criar os arquivos necessários para deploy.

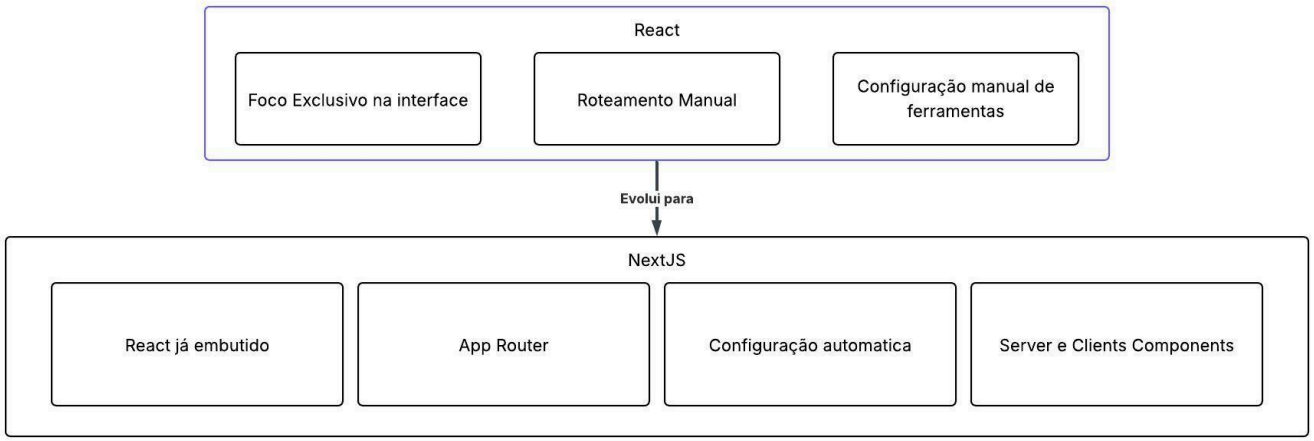
Agora como isso foi feito, basicamente como já havia ressaltado anteriormente por se tratar de um curso para iniciantes do zero o começo foi falando sobre o React para que serve, onde usar e porque usar, após isso quando passamos a para prática desde como criar um projeto utilizando o React, no nosso caso foi passado fazendo com JavaScript mas também pode ser utilizado diretamente com Type Script, após criamos nosso projeto e passarmos algum tempo explorando a organização de pastas e arquivos começamos com o desenvolvimento do nosso gerenciador de tarefas, e é neste desenvolvimento que conseguimos uma base de como funciona o React. Podemos dividir essa implementação em duas partes, primeiro a parte "crua" onde criamos nosso gerenciador, trabalhamos bastante com o React e suas ferramentas, utilizamos diversas bibliotecas diferentes, como, por exemplo, a Tailwind CSS para estilizar, uma biblioteca para gerar IDs, e também uma para ícones que utilizamos nos botões, assim conseguindo ter uma ideia tanto de como funciona as instalações das bibliotecas, como as utilizamos e as renderizamos o que precisamos no decorrer do código e também o que acontece quando instalamos uma biblioteca, e o que acontece quando alguém, por exemplo, clona nosso repositório para sua máquina, basicamente ao instalarmos uma biblioteca seus arquivos e dependências vão para o node_modules a pasta que sempre e configurada o gitignore para a ignorar, pois é uma pasta que tem muitos arquivos e não tem a necessidade de ser comitada, pois as bibliotecas e versões de tudo que utilizarmos no nosso projeto vão estar presente no arquivo "**package.json**" que é basicamente um manual de instruções para quando alguém esta instalando seu projeto em sua máquina ao usar os comandos para instalação o instalador utilizará esse arquivo para saber o que precisa, antes de falar sobre a implementação do próprio React também quero citar os hooks que são uma das principais ferramentas do React e deixa ele mais fácil e funcional no nosso projeto utilizamos principalmente dois deles que foram o useState e o useEffect, começando pelo useState que basicamente serve para criar estados dentro do nosso componente, ou seja, variáveis que quando o seu valor é alterado elas avisam o React para atualizar a tela automaticamente com a nova informação, o que no nosso gerenciador de tarefas foi utilizado para guardar a lista de tarefas e também o texto que digitávamos no input na hora de adicionar uma nova tarefa, e logo em seguida também utilizamos o useEffect que é um hook focado em causar efeitos colaterais na nossa aplicação, no nosso caso utilizamos ele de duas formas diferentes, a primeira foi para que assim que a nossa aplicação abrisse ele buscasse as tarefas já salvas na memória do navegador para não perdermos os dados quando a página recarregasse, e a segunda forma foi para ficar vigiando a nossa lista de tarefas onde toda vez que a gente adicionava, marcava como concluída ou excluía uma tarefa ele percebia essa mudança e atualizava o nosso localStorage salvando os dados mais recentes, mostrando na prática como esses hooks trabalham juntos para manter a interface sempre atualizada e os dados salvos sem a gente precisar ficar manipulando a tela manualmente como fazíamos no JavaScript puro.

Agora que já falamos sobre os hooks e as bibliotecas podemos falar sobre o porquê chamei esta primeira fase da implementação de crua, basicamente quando trabalhamos com React sua funcionalidade mais básica é a componentização para evitar repetir códigos, poupar trabalho e padronizar o projeto, coisa que nesta primeira parte praticamente não fizemos, onde criamos tudo de forma "bruta" repetindo código para componentes iguais e coisas assim, e eu

não estou falando que isso foi ruim muito pelo contrário com essa implementação em duas partes onde na primeira criamos tudo de forma bruta e não componentizada e na segunda refinamos esse projeto e o componentizamos ficou clara a vantagem de utilizar a componentização e a diferença de a utilizar e não utilizar, e com essa parte de repetirmos várias vezes códigos iguais conseguimos criar uma base forte e melhor compreender o React que diferente do JavaScript padrão onde temos os arquivos separados HTML, CSS, e Java Script e precisamos passar escrevendo em cada um e arrumando as coisas em tres arquivos diferentes para fazer um componente aqui fazemos um praticamente um arquivo de texto onde criamos o componente, o estilizamos e também adicionamos funcionalidade com Java Script como nesse caso dentro do próprio componente.

Começando Next.JS

Começando com os conteúdos sobre Next começamos com o vídeo "**Dominando Next JS completo do zero**" que diferente do anterior que é um vídeo prático sobre a programação e como utilizar a biblioteca React para desenvolver um sistema simples de gerenciamento de tarefas aqui é uma explicação e passagem quase geral sobre o framework Next.js.



O decorrer do vídeo é até que parecido começamos com a diferença do Next para o React sendo basicamente o React uma biblioteca que utilizando dentro de Next que é um framework que muitas coisas que precisávamos fazer manualmente no React aqui teremos algumas ferramentas para nos ajudar, como o sistema de roteamento baseado em arquivos (File System Routing) e a capacidade de realizar renderização do lado do servidor utilizado para melhorar o desempenho da aplicação, ao fazer a instalação do Next já podemos ver algumas diferenças como no procedimento de instalação manual onde escolhemos o que instalar, varias bibliotecas que tivemos que instalar manualmente já pede se você quer fazer a instalação das mesmas como **Tailwind CSS**, **App Router** o novo sistema de roteamento, **ESLint** para padronização do código e o uso de TypeScript. Além disso, destaca-se que o Next organiza automaticamente o projeto com uma estrutura de pastas otimizada (como a pasta src/app), onde cada diretório criado torna-se automaticamente uma rota da aplicação.

Como no Next graças ao Router ao criamos pastas dentro da estrutura src/app e dentro criamos um arquivo **page.tsx** assim criamos uma nova rota (que seria uma nova tela) graças a essa funcionalidade criar rotas é muito fácil, e tem algo que deixa isso melhor ainda solto dentro da pasta src/app temos um arquivo chamado **layout.tsx** que é o arquivo de renderização geral, e com ele podemos fazer componentes que aparecem em todas as telas, por exemplo, criamos um cabeçalho neste arquivo e criamos várias outras paginas esse cabeçalho por estar neste arquivo geral da pasta vai permitir que o cabeçalho apareça em todas as paginas, exceto se ao criamos a nova pagina coloquemos algo onde iria ficar esse componente geral, ai o que vai ser mostrado é o componente da rota e não o geral, mas no

caso de não ter nada o componente geral ira aparecer, sendo isso muito útil para padronização e impedindo que nos precise-se mos ficar carregando aquele mesmo componente em todas as paginas que criamos se necessário.

Outro diferencial do Next são os Server Components que permite que componentes ou até paginas sejam carregadas no próprio servidor com a restrição que elas sejam páginas ou componentes estáticos, assim mandando uma carga menor de JavaScript para o cliente ter de carregar assim deixando o site mais leve para o cliente permitindo um carregamento mais rápido e melhorando a experiência do mesmo, já quando os componentes forem interativos ou animados precisamos utilizar os clients components que são os componentes renderizados pelo cliente esses não tendo restrições e neles podemos usar as ferramentas do próprio React como **useState** que em server components são “bloqueados”. Além da leveza, essa estrutura de Server Components simplifica drasticamente como consumimos dados de APIs, no React, precisaríamos obrigatoriamente de ferramentas como **useEffect** e **useState** para buscar e armazenar os dados após a página carregar, mas no Next conseguimos transformar o componente em uma função assíncrona **async** e realizar o **fetch** dos dados diretamente no corpo do código com o **await**. Isso torna o código muito mais limpo e legível, pois os dados já chegam prontos do servidor para o componente, além disso, temos como deixar dados que dependam de resposta de APIs para carregarem separados do restante da página para casos em que a resposta possa demorar mais que o normal, o restante do site carregar e ficar algo personalizado como uma mensagem “carregando…” ou alguma coisa do tipo no local onde precisa da resposta da API prejudicando menos a experiência do usuário, do que depender do site inteiro não carregar ou demorar.

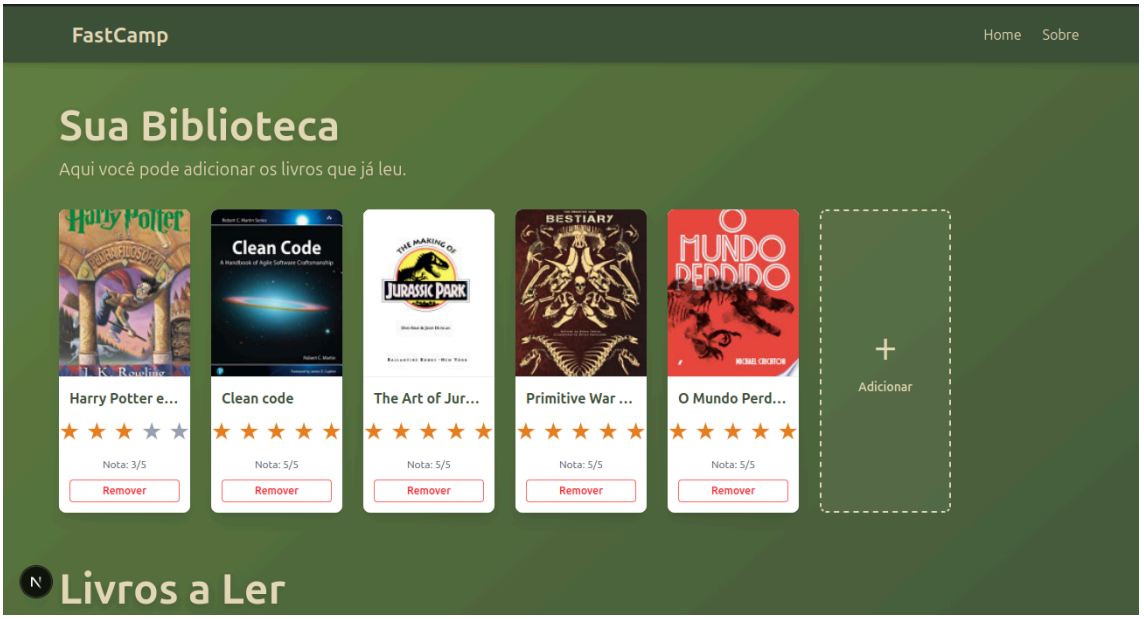
Também para lidarmos com sistemas mais complexos que contam com diversas páginas temos algo chamado rotas dinâmicas no Next que nos livra de precisarmos criar manualmente diversas páginas uma por uma, para evitar esse trabalho manual criamos as rotas dinâmicas que são paginas que vão receber, por exemplo, um [id] e seus dados vão ser carregados dinamicamente com base nesse identificador passado na URL. Isso significa que, ao acessarmos um endereço como /posts/1 ou /posts/2, o Next entende que o valor entre colchetes é um parâmetro, permitindo que o componente busque as informações específicas daquele item, assim poupando trabalho manual e aumentando a escalabilidade de nosso sistema. Por fim, o framework também simplifica a gestão de SEO através do objeto de metadados. De forma muito prática, podemos exportar uma constante chamada metadata dentro de cada página para definir o título e a descrição que aparecerão nos motores de busca e nas abas do navegador. Essa funcionalidade permite que cada rota, inclusive as dinâmicas, tenha informações personalizadas para o Google, garantindo que o site seja bem indexado e mais fácil de ser encontrado pelos usuários.

Sobre o artigo “**O que é Next.js? Uma Análise do Popular Framework JavaScript**”, ele basicamente aborda vários conceitos que já expliquei anteriormente sobre o vídeo, tratando das vantagens de utilizar o Next para criar aplicações tanto em pequena quanto em grande escala. O texto define o framework como uma "camada" superior ao React que simplifica a criação de aplicações web rápidas e prontas para produção, funcionando como uma ferramenta completa para o desenvolvimento de sites.

Atividade Pratica

Para desenvolvimento da atividade pratica que seria o código pessoal englobando os materiais vistos no decorrer do card. desenvolvi um sistema de biblioteca simples, utilizando Next, React e Type Script, onde no mesmo você adiciona os livros que já leu e pode dar uma nota para o menso, ou também adicionar os livros em uma lista de desejos que são livros que você ainda tem interesse em ler, para armazenar os dados estamos usando o armazenamento do próprio navegador ensinado no curso de react, e para testar a navegação de páginas fiz algo simples, uma página de sobre onde tem apenas um texto qualquer, feito apenas para praticar a navegação e criação de páginas no Next, o site foi desenvolvido componentizado

criando componentes e depois reutilizando os mesmos, e como nos dois vídeos foi feito a implementação de APIs também queria fazer para ter uma ideia de como funciona, então utilizei uma API do Google Books para quando você adicionar um livro se a API conseguir localizar no card do livro ficara com a capa dele, só é um pouco chata que precisa escrever o nome exato para ela localizar o livro, mas está funcionando, alguns livros para teste são **Harry Potter e a Pedra Filosofal, 1984…**



Dificuldades

Acho que a maior dificuldade no desenvolvimento deste card foi entender o conceito e aplicação de Server Client e Server Host, tirando isso não houve nenhuma outra dificuldade real a não ser falta de prática resolvida justamente praticando.

Conclusões

A conclusão que pode ser tirada é a versatilidade que o Next nos proporciona para o desenvolvimento web, facilitando e automatizando alguns processos manuais chatos do React.

Referencias

Videos:

Curso de React para Completos Iniciantes [2026]

Dominando Next JS completo do zero 🔥

Artigo:

https://kinsta.com/pt/blog/next-js/