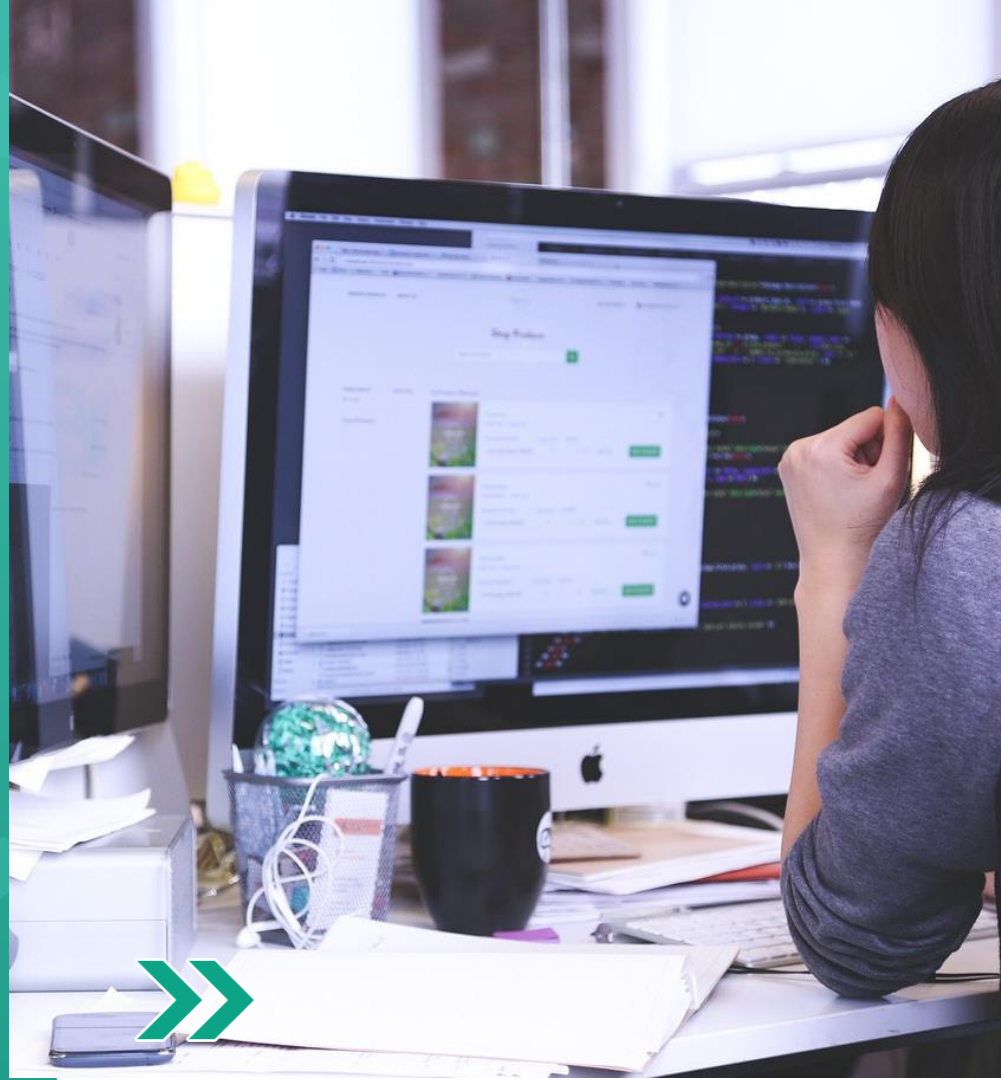
escola britânica de artes criativas & tecnologia

Profissão: Engenheiro Front-End

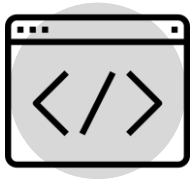
escola britânica de artes criativas & tecnologia

BOAS PRÁTICAS



Introdução ao React

- Conheça o JSX
- Crie componentes
- Manipule estado e eventos
- Utilize a função useEffect
- Renderize listas



Confira boas práticas da comunidade de Front-End por assunto relacionado às aulas.

Conheça o JSX

Ao usar o **bundle** no React, você pode considerar as seguintes dicas para otimizar o desempenho e a eficiência do seu aplicativo.

- **Otimização de imagens:**
  As imagens costumam ser arquivos grandes e podem afetar o tamanho do bundle. Para otimizar o carregamento de imagens, você pode usar técnicas como compressão de imagens, uso de formatos mais eficientes (como WebP) e carregamento assíncrono de imagens.

- **Análise de dependências:**
  Ao configurar seu bundle, é importante analisar as dependências do seu projeto. Certifique-se de que está incluindo apenas as bibliotecas e dependências necessárias no seu bundle final. Evite incluir bibliotecas não utilizadas, pois isso pode aumentar o tamanho do bundle e afetar negativamente o desempenho.

Conheça o JSX

Ao usar o **bundle** no React, você pode considerar as seguintes dicas para otimizar o desempenho e a eficiência do seu aplicativo.

- **Cache de assets:**
  Configurar o cache de assets corretamente pode melhorar significativamente o desempenho do seu aplicativo. Ao atribuir um hash único ao nome do arquivo de cada versão do bundle, você pode aproveitar o cache do navegador, permitindo que os arquivos sejam armazenados localmente pelos usuários entre as visitas ao site.

- **Monitoramento e testes:**
  Ao fazer alterações no seu bundle ou em configurações de empacotamento, é importante realizar testes e monitorar o desempenho do seu aplicativo. Use ferramentas de monitoramento de desempenho para identificar gargalos, tempos de carregamento lentos ou problemas de tamanho do bundle.

Conheça o JSX

Separamos algumas dicas para aproveitar ao máximo a funcionalidade do método createRoot():

- **Separe o código inicial:**
  Ao usar o createRoot(), você pode dividir seu código em uma parte inicial e uma parte assíncrona, carregando os módulos menos urgentes de forma tardia. Isso ajuda a melhorar o tempo de carregamento inicial do seu aplicativo.

- **Priorize a renderização:**
  O createRoot() permite que você agende a renderização dos seus componentes, permitindo que você defina prioridades e melhore o desempenho do aplicativo.

- **Teste e monitore o desempenho:**
  Assim como em qualquer alteração significativa no seu aplicativo, é importante realizar testes e monitorar o desempenho ao usar o createRoot(). Verifique se não há regressões de desempenho ou problemas de renderização. Use ferramentas de monitoramento e perfilamento para identificar gargalos e otimizar ainda mais o desempenho do seu aplicativo.

Conheça o JSX

Acompanhe agora algumas dicas para usar fragmentos no React.

- **Substitua elementos pai desnecessários:**
Quando você precisa renderizar vários elementos adjacentes sem um elemento pai adicional na árvore de elementos do DOM, em vez de criar um elemento ou qualquer outro elemento extra, use fragmentos para envolver os elementos filhos e evitar a criação de nós HTML desnecessários.

- **Chaves (keys) em fragmentos:**
Se você estiver usando um fragmento com uma lista de elementos que precisam ser renderizados com uma iteração, certifique-se de adicionar uma chave (key) exclusiva a cada elemento. Elas ajudam o React a identificar e rastrear elementos individualmente para melhorar a eficiência das atualizações do DOM.

- **Limitações no acesso a propriedades especiais:**
Como os fragmentos não são elementos reais, eles não suportam o acesso a propriedades especiais, como ref. Se você precisar acessar uma ref dentro de um fragmento, pode usar uma abordagem alternativa, envolvendo o fragmento em um componente ou usando a propriedade children para obter acesso ao elemento interno.

Conheça o JSX

Acompanhe agora algumas dicas para usar fragmentos no React.

- **Aninhamento de fragmentos:**
  Você pode aninhar fragmentos dentro de outros fragmentos para criar estruturas mais complexas, se necessário. Isso é útil quando você precisa agrupar elementos em vários níveis sem adicionar elementos HTML adicionais. No entanto, tenha em mente que o aninhamento excessivo de fragmentos pode tornar o código mais difícil de ler e entender. Use-o com moderação e priorize a legibilidade.

- **Considerações de renderização condicional:**
  Quando você precisa renderizar condicionalmente fragmentos, você pode usar operadores ternários ou curto-circuitos lógicos para envolver os fragmentos conforme necessário. Isso permite renderizar diferentes conjuntos de elementos com base em condições específicas.

Crie componentes

Acompanhe algumas dicas para usar componentes no React de maneira eficiente.

- **Composição de componentes:**
  Aproveite a capacidade de composição de componentes no React para criar hierarquias complexas de componentes. Um componente pode conter outros componentes, permitindo que você construa uma estrutura hierárquica de elementos reutilizáveis. Pense em como seus componentes podem ser combinados para criar uma interface de usuário mais poderosa.

- **Testabilidade:**
  Projetar seus componentes de forma que sejam facilmente testáveis é uma prática recomendada. Separe a lógica de negócios da lógica de renderização, permitindo que você teste a funcionalidade interna dos componentes sem a necessidade de simular o ambiente do React.

Crie componentes

Acompanhe algumas dicas para usar componentes no React de maneira eficiente.

- **Estilização:**
  Utilize abordagens de estilização, como CSS-in-JS (por exemplo, styled-components) ou pré-processadores de CSS (por exemplo, Sass), para melhorar a organização e reutilização de estilos em seus componentes. Isso pode ajudar a evitar conflitos de estilos e facilitar a manutenção do código.

- **Performance:**
  Ao criar componentes, leve em consideração a performance. Evite renderizações desnecessárias, use a reconciliação do React de forma eficiente e otimize o tempo de renderização. Use ferramentas de análise de desempenho para identificar gargalos e melhorar a eficiência do seu aplicativo.

- **Documentação e padronização:**
  Documente seus componentes para que outros desenvolvedores possam entender seu propósito, uso e comportamento. Além disso, estabeleça padrões de nomenclatura, estrutura e estilização dos componentes em seu projeto para manter a consistência e facilitar a colaboração entre a equipe.

Crie componentes

As **props** são a principal maneira de comunicar informações de um componente pai para um componente filho. Certifique-se de projetar suas props de forma clara e consistente, garantindo que sejam imutáveis e bem documentadas. Acompanhe algumas dicas sobre as propriedades.

- **Nomeie suas props com clareza:**
  Dê nomes significativos às suas props para que fique claro qual é a finalidade e o conteúdo que elas representam. Isso facilita a compreensão e o uso dos componentes por outros desenvolvedores.

- **Documente suas props:**
  Documente as props dos seus componentes para que outros desenvolvedores saibam como usá-las corretamente. Descreva o tipo de dado esperado, o propósito e qualquer valor padrão associado a cada prop.

- **Defina defaultProps:**
  Use a propriedade defaultProps para definir valores padrão para suas props. Isso é útil quando você deseja fornecer valores pré-definidos para suas props, caso não sejam especificadas pelo componente pai.

Crie componentes

As **props** são a principal maneira de comunicar informações de um componente pai para um componente filho. Certifique-se de projetar suas props de forma clara e consistente, garantindo que sejam imutáveis e bem documentadas. Acompanhe algumas dicas sobre as propriedades.

- **Verifique as props:**
  Se você espera que uma prop seja de um determinado tipo ou requerida, pode usar a biblioteca PropTypes ou TypeScript para fazer uma verificação de tipo nas props. Isso ajuda a identificar erros de tipo e a garantir que as props sejam passadas corretamente.

- **Mantenha suas props imutáveis:**
  As props no React devem ser tratadas como somente leitura e imutáveis. Não modifique diretamente as props recebidas em um componente. Em vez disso, se você precisar alterar o valor de uma prop, crie um estado local no componente e use-o para rastrear as alterações.

- **Evite props excessivamente longas ou complexas:**
  Tente manter suas props simples e diretas. Props excessivamente longas ou complexas podem dificultar a compreensão e o uso do componente. Considere dividir propriedades complexas em props menores, se possível.

Crie componentes

As **props** são a principal maneira de comunicar informações de um componente pai para um componente filho. Certifique-se de projetar suas props de forma clara e consistente, garantindo que sejam imutáveis e bem documentadas. Acompanhe algumas dicas sobre as propriedades.

- **Passe funções como props:**
  Além de passar dados, você também pode passar funções como props para permitir que os componentes interajam entre si. Isso é particularmente útil para permitir que os componentes filhos notifiquem ou enviem dados de volta ao componente pai.

- **Props desestruturadas:**
  Ao receber props em um componente, use a desestruturação (destructuring) para acessar facilmente as props individuais. Isso torna o código mais limpo e legível.

- **Evite props excessivamente aninhadas:**
  Evite props excessivamente aninhadas que podem dificultar a leitura e manutenção do código. Se você perceber que suas props estão ficando muito complexas, considere repensar a estrutura do seu componente e possivelmente dividir em componentes menores.

Manipule estados e eventos

Acompanhe algumas dicas para trabalhar com **estado** (state) no React de maneira eficiente.

escola britânica de artes criativas & tecnologia

- **Inicialize o estado adequadamente:**
  Inicialize o estado no construtor do componente ou usando a sintaxe de inicialização de estado (useState no caso de componentes funcionais). Certifique-se de que o estado esteja configurado corretamente antes de ser usado no componente.

- **Atualize o estado corretamente:**
  Ao atualizar o estado, leve em consideração que o React pode realizar atualizações de estado de forma assíncrona. Portanto, ao fazer uma atualização com base no estado atual, é recomendável usar a função de callback na chamada de atualização de estado. Isso garante que você esteja atualizando o estado com base no valor mais recente.

- **Componente de estado único:**
  Se você perceber que vários componentes precisam compartilhar o mesmo estado, considere centralizar o estado em um componente superior. Isso permite que você compartilhe e sincronize o estado entre os componentes usando props e callbacks.

Manipule estados e eventos

Acompanhe algumas dicas para trabalhar com **estado** (state) no React de maneira eficiente.

- **Levante o estado para o componente pai quando necessário:**
  Se vários componentes precisam compartilhar o mesmo estado e a comunicação entre eles se torna complexa, considere levantar o estado para o componente pai. Isso simplifica a comunicação e garante que o estado seja gerenciado em um único local.

- **Use bibliotecas de gerenciamento de estado:**
  Para aplicativos maiores e mais complexos, considere o uso de bibliotecas de gerenciamento de estado como Redux ou MobX. Essas bibliotecas fornecem recursos avançados para gerenciar e sincronizar o estado em todo o aplicativo.

- **Mantenha o estado imutável:**
  Não modifique diretamente o estado do componente. Em vez disso, use as funções de atualização de estado fornecidas pelo React, como setState (componentes baseados em classe) ou o método retornado pelo useState (componentes funcionais). Essas funções garantem que as atualizações de estado sejam feitas de maneira imutável, preservando a integridade do estado.

Manipule estados e eventos

Funções puras referem-se a funções que seguem o conceito de programação funcional. Nem todas as partes do código React podem ser funções puras, especialmente aquelas envolvidas em interações com o DOM, como manipulação de eventos ou atualizações assíncronas. No entanto, adotar o princípio de funções puras sempre que possível pode tornar seu código mais legível, testável e otimizável. Acompanhe:

- **Não têm efeitos colaterais:**
  Funções puras não causam efeitos colaterais observáveis fora do seu escopo. Isso significa que elas não modificam dados externos, não interagem com o DOM, não fazem chamadas a APIs externas ou têm qualquer comportamento que possa afetar o ambiente externo. Elas recebem argumentos como entrada e retornam um valor consistente com base nesses argumentos, sem afetar o estado global.

- **Produzem o mesmo resultado para as mesmas entradas:**
  Uma função pura sempre retorna o mesmo resultado quando chamada com os mesmos argumentos. Isso é importante para a previsibilidade e testabilidade do código.

Manipule estados e eventos

Funções puras referem-se a funções que seguem o conceito de programação funcional. Nem todas as partes do código React podem ser funções puras, especialmente aquelas envolvidas em interações com o DOM, como manipulação de eventos ou atualizações assíncronas. No entanto, adotar o princípio de funções puras sempre que possível pode tornar seu código mais legível, testável e otimizável. Acompanhe:

No contexto do React, é uma boa prática buscar funções puras sempre que possível, especialmente ao lidar com componentes funcionais e hooks. Seguem algumas razões pelas quais as funções puras são valorizadas no React:

1. Facilidade de teste: Funções puras são mais fáceis de testar, pois seu resultado é previsível e depende apenas dos argumentos fornecidos. Isso facilita a criação de testes unitários independentes.
2. Rastreabilidade: Quando um componente é composto por funções puras, é mais fácil rastrear e entender como os dados fluem por meio da aplicação. A lógica de processamento fica claramente definida na função pura, tornando o código mais legível e de fácil manutenção.
3. Performance e otimizações: Funções puras podem ser otimizadas mais facilmente, pois seu resultado depende apenas de suas entradas e não há efeitos colaterais para lidar. O React pode realizar otimizações como memoização (memoization) ou evitar renderizações desnecessárias com base na premissa de que funções puras retornarão o mesmo resultado para as mesmas entradas.

Utilize a função useEffect

Acompanhe algumas dicas sobre o uso do useEffect.

- **Compreenda o ciclo de vida do componente:**
O useEffect é uma forma de lidar com efeitos colaterais em componentes funcionais. Ele é executado após cada renderização do componente, incluindo a primeira renderização e todas as atualizações subsequentes. Tenha em mente que o useEffect não substitui completamente os métodos do ciclo de vida dos componentes de classe, mas pode ser usado para realizar tarefas semelhantes.

- **Especifique as dependências corretamente:**
O useEffect recebe um array de dependências como segundo argumento. Essas dependências indicam quais variáveis devem ser observadas para disparar o efeito. É importante especificar as dependências corretamente para evitar que o efeito seja executado desnecessariamente ou que falhe em ser executado quando necessário. Se nenhum array de dependências for fornecido, o efeito será executado após cada renderização.

Utilize a função useEffect

Acompanhe algumas dicas sobre o uso do useEffect.

escola britânica de artes criativas & tecnologia

- **Cuidado com atualizações infinitas:**
  Se você não especificar as dependências corretamente, pode acabar em um loop de atualizações infinitas. Por exemplo, se você usar useEffect e não fornecer um array de dependências, o efeito será executado a cada renderização, causando um ciclo infinito. Certifique-se de revisar e entender quais dependências são relevantes para o seu efeito.

- **Limpeza de efeitos:**
  O useEffect pode retornar uma função de limpeza opcional, que será executada quando o componente for desmontado ou antes de executar o próximo efeito. Essa função é útil para cancelar assinaturas de eventos, limpar temporizadores ou executar qualquer limpeza necessária para evitar vazamentos de memória ou comportamentos inesperados. Retorne uma função de limpeza no useEffect quando for necessário realizar essas ações.

Utilize a função useEffect

Acompanhe algumas dicas sobre o uso do useEffect.

- **Efeitos assíncronos:**
  Se o efeito envolver operações assíncronas, como chamadas a APIs externas, você pode usar funções assíncronas dentro do useEffect ou chamar uma função assíncrona definida separadamente. Lembre-se de que o useEffect não pode ser assíncrono por si só.

- **Considere efeitos específicos:**
  Em vez de usar um único useEffect para tratar vários casos, considere dividir a lógica em múltiplos efeitos. Isso ajuda a manter o código mais organizado e modular, tornando mais fácil de entender e testar.

- **Depuração de efeitos:**
  Para depurar e entender melhor o comportamento dos efeitos, você pode usar a ferramenta de desenvolvedor do React para observar quando eles são executados ou usar console.log dentro do useEffect para rastrear os valores das dependências e verificar se o efeito está sendo chamado corretamente.

Utilize a função useEffect

Acompanhe algumas dicas sobre o uso do useEffect.

- **Tratamento de erros:**
  Se ocorrerem erros dentro do useEffect, eles normalmente serão capturados pelo React e exibidos no console do navegador. No entanto, se você precisar lidar com erros de forma personalizada, pode usar um bloco try-catch dentro do efeito ou uma função de tratamento de erros separada.

- **Evite efeitos desnecessários:**
  Ao utilizar o useEffect, tente otimizar seu código para evitar efeitos desnecessários. Por exemplo, use uma lógica condicional dentro do efeito para verificar se é realmente necessário executar determinadas ações, como chamar uma API, com base nas mudanças nas dependências.

Utilize a função useEffect

Outras funções hook bastante utilizadas são as destacadas a seguir.

- **useContext:**
  O Hook useContext permite que você acesse um contexto definido com o React.createContext() em um componente funcional. Ele retorna o valor atual do contexto.

- **useMemo:**
  O Hook useMemo permite que você memorize um valor calculado e evite recomputá-lo em cada renderização do componente, a menos que as dependências fornecidas mudem. Isso é útil para otimizar o desempenho quando há cálculos caros ou operações intensivas.

- **useRef:**
  O Hook useRef retorna um objeto mutável com uma propriedade .current. Você pode usá-lo para manter valores que persistem entre renderizações, como referências a elementos DOM ou qualquer outro valor que precise ser mantido sem acionar uma nova renderização.

Utilize a função useEffect

Outras funções hook bastante utilizadas são as destacadas a seguir.

- **useCallback:**
  O Hook useCallback retorna uma versão memoizada de uma função. Isso é útil para evitar a criação de novas instâncias de função a cada renderização, especialmente quando essa função é passada como prop para componentes filhos.

- **useReducer:**
  O Hook useReducer é uma alternativa ao uso de useState para gerenciar o estado de um componente. Ele aceita um reducer (função de atualização do estado) e um estado inicial, retornando o estado atual e uma função para despachar ações para atualizar o estado.

Renderize listas

O **map** é frequentemente usado para renderizar listas de elementos de forma dinâmica.
Acompanhe algumas dicas para usar o método map() no React de maneira eficiente.

- **Transforme dados antes de renderizar:**
  Além de simplesmente renderizar elementos, você pode usar o map() para transformar os dados antes de renderizá-los. Por exemplo, você pode mapear um array de objetos para um array de componentes personalizados, passando propriedades específicas para cada componente.

- **Extraia lógica complexa para funções separadas:**
  Se a lógica dentro do map() ficar muito complexa, é uma boa prática extrair essa lógica para funções separadas. Isso ajuda a manter o código mais legível e facilita a reutilização da lógica em outros lugares, se necessário.

Renderize listas

O **map** é frequentemente usado para renderizar listas de elementos de forma dinâmica.
Acompanhe algumas dicas para usar o método map() no React de maneira eficiente.

- **Manipule valores vazios ou nulos:**
  Se o array que você está mapeando pode conter valores vazios ou nulos, você pode adicionar uma lógica de tratamento para lidar com esses casos. Por exemplo, você pode retornar uma mensagem de "lista vazia" ou renderizar um componente de carregamento enquanto aguarda os dados.

- **Combine map() com outros métodos de array:**
  O método map() pode ser combinado com outros métodos de array, como filter() e sort(), para realizar operações mais avançadas em seus dados antes de renderizá-los. Essa combinação permite filtrar, classificar e transformar os dados de maneira flexível.

escola britânica de artes criativas & tecnologia

Bons estudos!