Front End III

Glossário de revisão

1) Hook: Um Hook é uma função especial que permite 'se conectar' às características do React. Os Hooks são basicamente funções, e permitem aos componentes funcionais incorporar características do React, antes restritas apenas a componentes de classes. Entre outros utilitários, eles fornecem aos componentes um estado interno e ciclo de vida. Ainda (e não menos importante), resolvem problemas que o React enfrentou durante anos. Os Hooks são uma forma simplificada de inserir características do React - por exemplo: estado, ciclo de vida, contexto, referências em componentes funcionais - sem alterar o funcionamento e as bases dele, como a importância dos componentes, o fluxo de dados e as props. O genial dos Hooks é que eles permitem extrair lógica de um componente para sua reutilização e compartilhamento.

2) Problemas dos componentes de classe:

- A lógica de estado não é fácil de se reutilizar.
- Os componentes complexos são difíceis de se compreender.
- A implementação de classes pode ser confusa e entediante.

3) Surgimento dos Hooks: no mês de outubro do ano 2018 - na React Conf realizada em Henderson, Nevada, EUA - Sophie Alpert e DAN Abramov, integrantes da equipe do Facebook responsável pelo desenvolvimento do React (naquele momento) apresentaram uma nova característica revolucionária para o mundo do front-end: React Hooks. Mais tarde, em fevereiro de 2019, essa proposta se tornou realidade e o React liberou a sua versão 16.8, incluindo a API de Hooks.

4) Benefícios hooks:

- Menos quantidade de código.

- Código mais organizado.
- Uso de funções reutilizáveis.
- Mais facilidade para testar.
- Não chama o super () em um construtor de classes.
- Não trabalha com this e bind no uso de manipuladores.
- Simplifica a vida e o ciclo de vida.
- O estado local está dentro do escopo dos handlers e as funções de efeitos secundários.
- Componentes mais reduzidos, que tornam mais fácil o trabalho do React.

5) Convívio entre componentes de Classe e Hooks: no React convivem ambos os tipos de programação. Ou seja, na mesma aplicação podem conviver componentes de ambos os tipos, para dar tempo a uma mudança gradual de mentalidade. A modo de exemplo desta decisão, o Facebook mantém uma imensa quantidade de código escrito em classes. Por fim, devemos esclarecer que não há uma liberdade absoluta entre ambos os tipos de componentes, pois não é possível usar Hooks dentro de um componente de classe.

6) Hooks nativos

useState	Declara variáveis de estado e as atualiza.
useEffect	Permite incluir e lidar com efeitos secundários.
useContext	Permite criar um fornecedor de dados globais para que eles possam ser consumidos pelos componentes inseridos pelo fornecedor.
useReducer	É uma variação de useState. O seu uso é conveninete quando nos deparamos com uma grande quantidade de partes de estado.
useCallback	Devolve um callback que é memorizado a fim de evitar renderizações desnecessárias.
useMemo	Devolve um valor memorizado, e ainda é útil para evitar renderizações desnecessárias. Otimiza o desempenho.
useRef	Permite utilizar programação imperativa, e devolve um objeto mutável cujas alterações de valor não irão renderizar o componente novamente. Contrariamente, ele irá se manter persistente durante toda a vida do componente.
useImperativeHandle	Customiza o valor de instância que expõe os componentes

	pais quando se usa ref.
useLayoutEffect	É parecido com o useEffect, mas ele é disparado de forma síncrona após todas as mutações do DOM.
useDebugValue	Permite aplicar um rótulo nos Hooks customizados, visível através de React DevTools.

7) useState: a partir do lançamento de Hooks, os componentes funcionais podem declarar e atualizar o seu estado interno através do hook useState, e é uma função que cria uma variável que permite armazenar uma parte de estado interno e, ao mesmo tempo, atualizar esse valor para um componente funcional.

8) Estrutura de useState: recebe um argumento e devolve um array com 2 elementos. O primeiro deles será o valor da variável, e o segundo, a função para alterá-la.

9) Funcionamento de useState: quando o React renderiza ou re-renderiza o nosso componente, ele solicita ao React o último valor de estado, a fim de desenhar a interface de usuário, assim como a função, para atualizar o valor de estado. Se a função para modificar o valor for executada, o React irá re-renderizar a interface, assim como comparar a interface de usuário gerada com a versão armazenada, atualizando a tela da forma mais eficiente.

10) Ciclo useState:

1. O React atravessa a árvore de componentes, montando-os para desenhar a UI, e passará as props correspondentes dependendo do caso.
2. Cada componente invoca o useState pela primeira vez, solicitando ao useState o valor inicial, e o useState, por sua vez, o solicitará ao React.
3. O React retorna tanto o valor atual quanto a função de atualização.
4. O componente configura os controladores de eventos de usuário.
5. O componente utiliza o valor de estado obtido para declarar a UI.
6. O React atualiza o DOM com as alterações necessárias.
7. Quando o controlador de eventos invoca a função de atualização, um evento é ativado e o controlador é executado. Através do handler, a função de atualização é invocada para alterar o valor do estado.
8. O React altera o valor do estado conforme o valor passado pela função de atualização.
9. O React chama o componente.

10. O componente chama o useState novamente, mas sem considerar o argumento passado inicialmente.
11. O React retorna o valor de estado atual e a função de atualização novamente.
12. O componente cria uma nova versão do manipulador de eventos e utiliza o novo valor de estado.
13. O componente devolve o seu UI com o novo valor de estado.
14. O React atualiza o DOM com as alterações necessárias.

11) Eventos no React: possibilitam aos usuários interagirem com a camada visual e de dados da aplicação. A que chamamos de eventos de usuário? Por exemplo, quando clicamos em um botão, alteramos um input (change) ou fazemos hover sobre uma imagem, entre outras muitas opções. São declarados nas tags JSX no render do componente.

12) Manipulador de eventos: é uma função ou método que, assim que o evento for disparado, executará as instruções declaradas no seu interior.

13) SyntheticEvent: fazendo uma pequena aproximação do assunto, ao instanciar um manipulador de eventos (ao disparar um evento), uma instância de SyntheticEvent será passada para ele. Qual é a utilidade disso? Simplificando, poderíamos dizer que ele fornece um contêiner que não se importará nem um pouco com qual navegador está sendo usado, permitindo assim que os eventos funcionem da mesma forma em todos os navegadores.

14) Componentes controlados: tanto os componentes controlados pelo React quanto os não controlados são controlados pelo DOM.