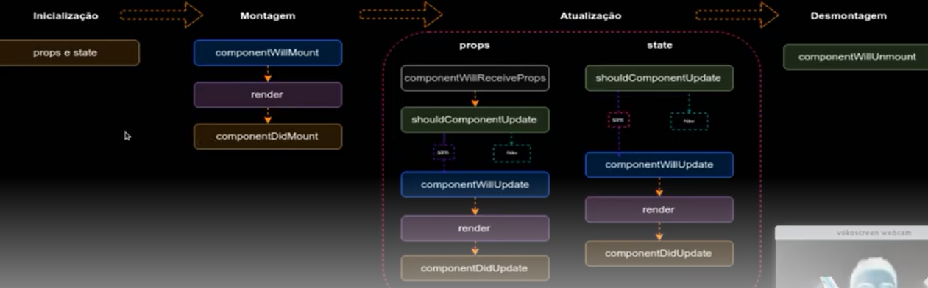
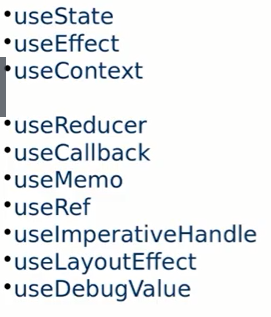
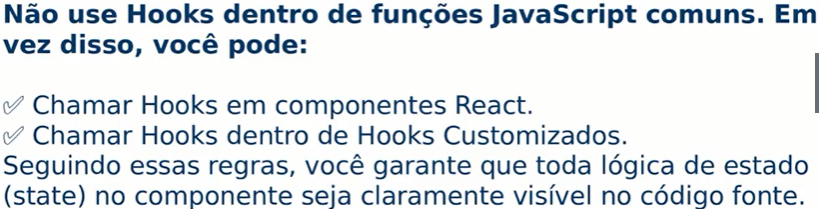
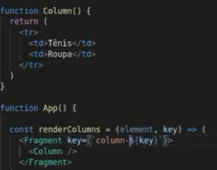
Esse curso fala sobre as praticas avançadas de reactJS, focando no seu ciclo de vida, mais sobre hooks e context API. Nós estamos começando a repetir algumas coisas relacionadas ao ReactJS, então algumas coisas não estão escritas, esse professor também faz muito pelo código, então não tem muito o que fazer além de ver como ele faz as coisas.

Existem quatro estágios importantes sobre o **ciclo de vida** do ReactJS:



Alguns exemplos de hooks:

O **context API** é utilizado para passar valores entre componentes sem ter que passar por um componente intermediario, coisas como **logins**, **temas**, **valores globais**, etc..

**Fragments** permitem agrupar um lista de filhos sem adicionar nós (knoots) extras no DOM. é um padrão comum no react que os componentes possam retornar multiplos elementos, e os fragments permitem exatamente isso.

É muito utilizado ao renderizar elementos em lista sem retornar coisas que não sejam utilizadas.

O grande ponto de usar react é a facilidade de lidar com erros, **um erro de JavaScript não deve ser capaz de quebrar uma aplicação**, para resolver esse problema, o React 16 introduziu um novo conceito chamado **error boundary**. este envelopa a aplicação e “captura” os erros para separar esses dados, permitindo monitorar os erros da aplicação e debugga-los conforme necessário, além de obviamente não permitir o erro de quebrar a aplicação.

O termo **render prop** se refere a uma técnica de compartilhar código entre componentes React, passando um prop cujo valor é uma função. Um componente com uma render prop recebe uma função que retorna um elemento React e a invoca no momento de renderização, não sendo necessário para o componente implementar uma lógica própria.

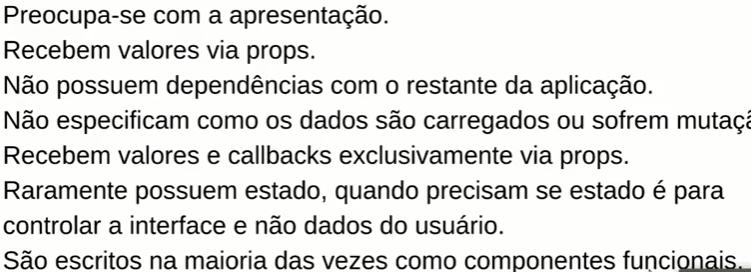
Na medida que a aplicação cresce, muitos bugs podem ser capturados, o react possui algumas habilidades de checkagem de bugs de tipos nativas, se referindo ao termo **typechecking**.

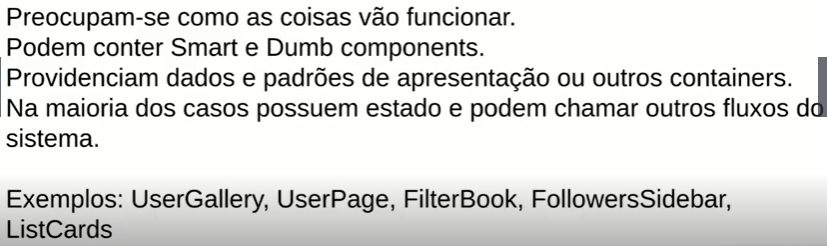
Os **Refs** permitem acessar as árvores do DOM e/ou elementos react. Permitindo assim:

Manipular bibliotecas de terceiros.

Gerenciar inputs de foco, seleção de textos ou reprodução de mídias.

Animações imperativas.

**Dumb components** são componentes de nivel básico, sem regras específicas, não são dependentes de outras partes da aplicação, sendo em geral apenas componentes pequenos por si só e são funcionais. São componentes de apresentação.

**Smarts components** já se preocupam mais com o funcionamento das coisas, providenciam diversos dados importantes para o funcionamento do aplicaativo e consequentemente podem depender de outros componentes, smarts ou dumbs, para seu funcionamento adequado.