**REACT**

Principais premissas:

1. Componetização – Reaproveitar os componentes (ícones diferentes por exemplo).
2. Gerenciamento de estado – React é reativo (JS é imperativo).

Utilizando NPM Create React app, escreva o comando no terminal:

* npx create-react-app ./ para instalar o React caso já esteja dentro da pasta desejada.
* npx create-react-app projeto(*isso é o nome da pasta)* –template typescript –use-npm
  + Esse comando irá criar um projeto React com Typescript utilizando npm com Create React App
* cd projeto
  + Irá entrar na pasta do projeto
* npm start
  + Irá rodar o projeto
* O comando *npx*, diferente do comando *npm*,apenas executa um pacote, sem que precisemos instalá-lo na nossa máquina. Como o create-react-app normalmente é usado somente no começo para criar o projeto, o ideal é executar com *npx* pois isso fará com que você sempre execute a última versão do pacote.
* *Create-react-app* é o nome do pacote que queremos executar.
* *Projeto* é o nome do projeto que queremos, assim como o nome da pasta.
* *--template typescrip* o Create React App nos permite usar vários templates, tendo como sintaxe o –template, que diz pro terminal que logo após iremos dizer qual template gostaríamos de usar, e o nome do template em si. O Create React App já tem o template *typescript* criado, mas caso tenha algum template que queira usar e não existe, você também pode criar o seu.
* *--use-npm* caso o *yarn* esteja instalado na sua máquina, o Create React App prioriza a utilização dele, e isso irá gerar um *yarn.lock* dentro da nossa aplicação, e nós queremos utilizar o npm ao invés do yarn, por isso precisamos usar o comando *–use-npm* se quisermos garantir que o Create React App utilize o npm para instalar as dependências necessárias e também garantir que ele gere o *package-lock.json* ao invés do *yarn.lock*

Don’t Repeat Yourself: Princípio da componetização para não reescrever códigos.

Single Responsability Principal: Deixar responsabilidades para cada componentes de forma independente.

OBS: Caso tenha fechado o projeto e queira retornar no terminal, basta escrever os códigos:

1. npm alura-studies(nome da pasta/arquivo)
2. npm start

­

Uma das vantagens de se utilizar o React é por conta da Componetização no qual podemos reutilizar as diferentes partes do layout do projeto de maneira mais simples do que se tivéssemos que reescrever a árvore HTML novamente. Também permite a composição de diversos componentes para a criação de um comportamento complexo, facilitando a compreensão do código. Como cada componente tem um nome que a equipe desenvolvendo o sistema definiu é possível abstrairmos partes do código para que a comunicação entre o time melhore.

**Elementos** são os menores blocos de construção de aplicativos React. São objetos simples e compõem os componentes.

**State** serve para guardar valores/estado que podem vir a mudar com a interação do usuário com o componente, tendo efeito na renderização do mesmo. Pode ser passado como *props*. É uma forma de salvar informações que serão observadas pelo Virtual DOM.

* Stateful – usa estados. Criado usando classe
* Stateless – não usa estados

**Props** são valores para a configuração de um componente, essas *props* são passadas pelo elemento acima (que utiliza o componente que irá receber) e são imutáveis. São utilizados para a comunicação de componentes. São propriedades que passamos de um componente para outro quando o componente filho precisa de uma informação que o componente pai possui.

**Key** é um atributo especial que você precisa definir ao criar listas de elementos.

**componentDidMount( )** esse método é chamado assim que o componente é criado e está pronto. É dentro dele que devemos iniciar chamadas para API ou executar código que tem efeito colateral. No caso, é executado depois que a saída do componente é renderizada no DOM.

**JSX** é o código HTML dentro do JS.

OBS: shft+alt+F faz a formatação das tags no VS Code

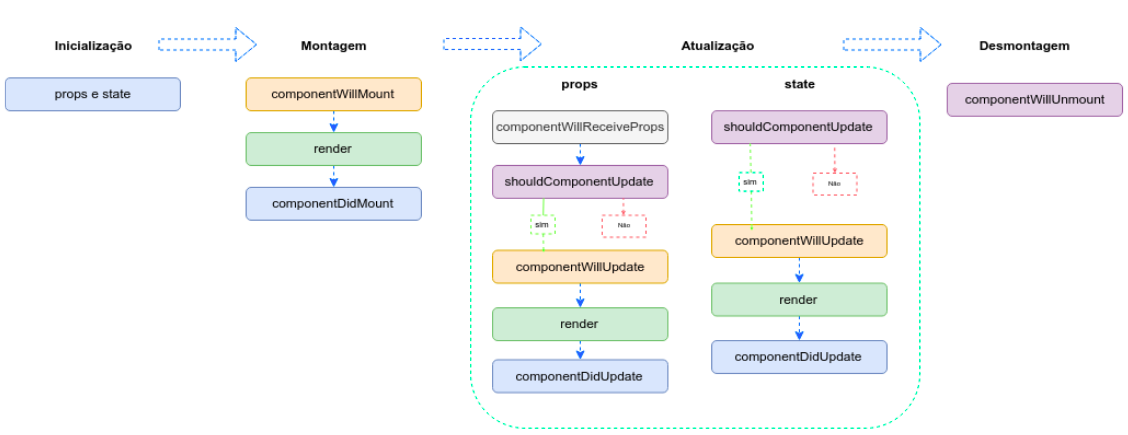
**Styled components:** Aplicação de CSS no JS

Para instalação, digite no Node: npm install --save styled-components

**Redux:**

Para instalação, digite no Node: npm install react-redux

**Ciclo de vida dos Componentes:**



**Hooks**são funções externas que você pode importar para dentro do seu componente.

1. useState
2. useEffect – espera receber uma função como primeiro parâmetro e um array no segundo parâmetro. Esse segundo parâmetro é quem irá definir quando o *useEffect* irá renderizar novamente. Caso o segundo parâmetro seja um array vazio, siginifca que o useEffect será renderizado somente a primeira vez, visto que o segundo parametro nunca irá sofrer alteração, pois é um array vazio (esse exemplo é utilizado na chamada de uma API).
3. useRef – tem um objetivo similar ao de uma variável, porém a diferença é que com o *useRef* seu valor é guardado e o componente não é re-renderizado. Sendo assim, a principal utilização do*useRef* é quando queremos guardar um valor de uma variável sem ser necessário sua renderização
4. useContext – tem o intuito de passar propriedades de elementos pais para elementos de hierarquias mais baixas, sem a necessidade de passar de elemento para elemento, como se fosse um estado global. Por exemplo: um componente A possui uma propriedade e quer passar para o componente D (seu “bisneto”). A maneira tradicional seria passar a propriedade seguindo a cadeia hierárquica A -> B -> C -> D, porém em um projeto grande pode acabar ficando complexo e confuso demais. Através do *useContext*, usando o “context.provider” torna-se possível passar a propriedade direto de A -> D ou para qualquer elemento que esteja abaixo em sua hierarquia.
5. useReducer – tem um objetivo similar ao *useState* porém serve para realizar funcionalidades mais complexas. Através do *useReducer* não é necessário mais utilizar vários *useStates* pois você consegue passar todos os estados em apenas um *useReducer*
6. useParams – é utilizado quando queremos passar um caminho dinâmico na url. Por exemplo: *path=”/cocktail/:id”* nesse caso o *useParams* será utilizado para o *id*  pois esse será diferente de acordo com a busca do usuário.
7. useCallback
8. useMemo
9. useImperativeHandle
10. useLayoutEffect
11. useDebugValue

**.env** para utilizar o dotEnv no *vite*, é necessário instalar primeiro no Node o ***npm i dotenv*** e em seguida, dentro do arquivo ***vite.config.js*** irá inserir o ***loadEnv*:**

import { defineConfig, loadEnv } from 'vite'

**Renderização condicional:** funciona da mesma forma que condições funcionam em Javascript. Pode utilizar o operador lógico “&&” como um “*if inline*” ou também é possível utilizar operador ternário *(condição ? true : false)*. PS: Toda vez que uma condição tornar-se muito complexa, é sinal de que deve-se extrair componentes.