1. `import React, { useState, useEffect } from 'react';`
    - **Explicação**: Aqui estamos trazendo ("importando") o React e duas ferramentas dele: `useState` e `useEffect`. Imagine que estamos pegando ferramentas de uma caixa de ferramentas que nos ajudarão a construir o nosso aplicativo.
2. `function App() {`
    - **Explicação**: Esta linha começa a definir uma "função" chamada `App`. Pense nisso como se estivéssemos criando uma receita para fazer um bolo. Essa receita nos diz o que o nosso aplicativo vai fazer.
3. `const [city, setCity] = useState('');`
    - **Explicação**: Aqui, estamos criando uma "caixa" (`city`) para guardar o nome da cidade que o usuário vai digitar. `useState('')` diz que essa caixa começa vazia. `setCity` é a ferramenta que usamos para colocar o nome na caixa.
4. `const [weatherData, setWeatherData] = useState(null);`
    - **Explicação**: Esta linha cria outra "caixa" (`weatherData`) para guardar as informações do tempo que pegamos da internet (a previsão do tempo). No começo, essa caixa está vazia (`null`), e `setWeatherData` é a ferramenta para colocar informações na caixa.
5. `const [error, setError] = useState('');`
    - **Explicação**: Esta linha cria uma "caixa" (`error`) para guardar mensagens de erro, caso algo dê errado, como quando a cidade não é encontrada. No começo, está vazia, e `setError` é a ferramenta para colocar uma mensagem lá dentro.
6. `const fetchWeather = async () => {`
    - **Explicação**: Esta linha cria uma função chamada `fetchWeather`, que vai buscar as informações do tempo na internet. `async` significa que essa função pode demorar um pouquinho para terminar, porque precisa esperar pela internet.
7. `try {`
    - **Explicação**: "Tentar" fazer algo. Aqui, estamos dizendo ao nosso código para tentar pegar as informações do tempo. Se algo der errado, ele vai "capturar" o erro e lidar com ele.
8. `const apiKey = 'SUA_CHAVE_DE_API';`
    - **Explicação**: Esta linha guarda a "chave secreta" (`apiKey`) que precisamos usar para acessar a API da WeatherAPI. Essa chave é como uma senha que nos permite usar o serviço deles.
9. `const response = await fetch(`**https://api.weatherapi.com/v1/current.json?key=${apiKey}&q=${city}&lang=pt`);`**

- **Explicação**: Aqui, estamos pedindo ("fetch") as informações do tempo para a API da WeatherAPI. `await` significa que nosso código vai esperar até que a resposta chegue. Estamos dizendo à API: "Me dê o tempo atual para a cidade que está na caixa `city`."

10. `if (!response.ok) {`
    - **Explicação**: Estamos verificando se a resposta foi boa (`ok`). Se não for (`!response.ok`), isso significa que algo deu errado, como a cidade não ser encontrada.
11. `throw new Error('Cidade não encontrada');`
    - **Explicação**: Se algo deu errado, jogamos ("throw") um erro. É como se disséssemos "Ops, não encontramos essa cidade!" e o código vai direto para a parte que lida com erros.
12. `const data = await response.json();`
    - **Explicação**: Quando a resposta chega, pegamos as informações e as transformamos em um formato que o código possa entender (`json`). Guardamos essas informações na "caixa" `data`.
13. `setWeatherData(data);`
    - **Explicação**: Agora, colocamos as informações do tempo na "caixa" `weatherData` para que possamos usá-las mais tarde.
14. `setError('');`
    - **Explicação**: Como tudo deu certo, garantimos que a "caixa" de erros esteja vazia, sem nenhuma mensagem de erro.
15. `} catch (err) {`
    - **Explicação**: Se algo deu errado, o código "pega" (`catch`) o erro. É como dizer: "Se algo der errado, faça isso."
16. `setWeatherData(null);`
    - **Explicação**: Se houve um erro, limpamos a "caixa" `weatherData`, para não mostrar informações erradas.
17. `setError(err.message);`
    - **Explicação**: Colocamos a mensagem de erro na "caixa" `error` para que possamos mostrar ao usuário o que deu errado.
18. `};`
    - **Explicação**: Isso fecha a função `fetchWeather`. Acabamos de dizer ao código o que ele deve fazer para buscar a previsão do tempo.
19. `const handleSearch = (e) => {`
    - **Explicação**: Aqui, criamos outra função chamada `handleSearch`. Ela lida com o que acontece quando o usuário clica no botão de buscar.
20. `e.preventDefault();`
    - **Explicação**: Dizemos ao navegador para não fazer o que ele normalmente faria ao enviar o formulário. Em vez disso, vamos buscar o tempo com a nossa própria função `fetchWeather`.
21. `fetchWeather();`

- **Explicação**: Chamamos a função `fetchWeather` para buscar as informações do tempo. É como se disséssemos: "Agora vá buscar as informações do tempo!"

22. `return (`
    - **Explicação**: Estamos começando a dizer o que o aplicativo vai mostrar na tela. Tudo o que vem depois deste `return` será mostrado ao usuário.
23. `<div className="app-container">`
    - **Explicação**: Criamos uma "caixa" chamada `div` para segurar tudo o que vamos mostrar no nosso aplicativo. Essa "caixa" tem uma classe (`className`) chamada `app-container`, que ajuda a estilizar a aparência.
24. `<h1>Previsão do Tempo</h1>`
    - **Explicação**: Esta linha coloca um título grande no topo da página que diz "Previsão do Tempo".
25. `<form onSubmit={handleSearch}>`
    - **Explicação**: Criamos um formulário (`form`). Quando o usuário envia (submit) esse formulário, chamamos a função `handleSearch`.
26. `<input type="text" value={city} onChange={(e) => setCity(e.target.value)} placeholder="Digite o nome da cidade" />`
    - **Explicação**: Criamos uma caixa de texto (`input`) onde o usuário pode digitar o nome da cidade. `onChange` diz ao código para atualizar a "caixa" `city` sempre que o usuário digitar algo novo.
27. `<button type="submit">Buscar</button>`
    - **Explicação**: Criamos um botão (`button`) com o texto "Buscar". Quando o usuário clica nesse botão, ele envia o formulário e aciona a função `handleSearch`, que vai buscar as informações do tempo.
28. `</form>`
    - **Explicação**: Fecha a "caixa" do formulário que começamos anteriormente. Tudo o que está dentro desse formulário é o que será enviado quando o usuário clicar no botão "Buscar".
29. `{error && <p>{error}</p>}`
    - **Explicação**: Aqui estamos dizendo: "Se houver uma mensagem de erro (se `error` não estiver vazia), mostre essa mensagem em um parágrafo (`<p>`)". O `&&` é como um "se" — ele só mostra o que está depois dele se `error` tiver alguma coisa.
30. `{weatherData && (`
    - **Explicação**: Semelhante à linha anterior, mas para os dados do tempo. Se houver dados de tempo (`weatherData` não estiver vazio), mostramos essas informações na tela.
31. `<div className="weather-info">`
    - **Explicação**: Criamos uma "caixa" (`div`) para segurar as informações do tempo. Esta "caixa" tem a classe `weather-info`, que pode ser usada para estilizar o conteúdo.

32. `<h2>{weatherData.location.name}</h2>`
    - **Explicação**: Mostra o nome da cidade (`weatherData.location.name`) em um título menor (`<h2>`). É como o cabeçalho para as informações do tempo.
33. `<p>Temperatura: {weatherData.current.temp_c}°C</p>`
    - **Explicação**: Mostra a temperatura atual (`weatherData.current.temp_c`) em um parágrafo (`<p>`). Aqui, `temp_c` é a temperatura em graus Celsius.
34. `<p>Condições: {weatherData.current.condition.text}</p>`
    - **Explicação**: Mostra uma descrição das condições climáticas (`weatherData.current.condition.text`) em um parágrafo (`<p>`). Por exemplo, pode mostrar algo como "Ensolarado" ou "Nublado".
35. `<img src={weatherData.current.condition.icon} alt={weatherData.current.condition.text} />`
    - **Explicação**: Mostra uma imagem (`<img>`) que representa as condições climáticas atuais. O `src` é o endereço da imagem que vem da API (`weatherData.current.condition.icon`), e o `alt` é o texto que descreve a imagem, útil para acessibilidade.
36. `</div>`
    - **Explicação**: Fecha a "caixa" que criamos para as informações do tempo.
37. `</div>`
    - **Explicação**: Fecha a "caixa" principal (`app-container`) que segura todo o conteúdo do aplicativo.
38. `export default App;`
    - **Explicação**: Diz ao React para usar essa função (`App`) como o componente principal do aplicativo. É como dizer "Aqui está o meu aplicativo, e é isso que eu quero mostrar".

Resumo

- **Importação**: Pegamos as ferramentas necessárias.
- **Função `App`**: É a receita para nosso aplicativo.
- **Estados**: Caixas para guardar a cidade, dados do tempo e erros.
- **Função `fetchWeather`**: Busca os dados do tempo e lida com erros.
- **Função `handleSearch`**: Lida com o envio do formulário e chama `fetchWeather`.
- **JSX**: O que será mostrado na tela, incluindo o formulário e as informações do tempo.

Espero que essa explicação ajude a entender cada parte do código! Se precisar de mais alguma coisa, estou aqui para ajudar. Boa sorte na sua prova! 🌟