vivasoft

Learnathon 2.0 - React Class 4

...

Ekhlas Mridha | Software Engineer (L-1) | Vivasoft Limited

Learnathon - 2.0

vivasoft

Data fetching

Ways of making API request to fetch data in react:

- Using “fetch” api

  - It’s a browser api to make an api request.

- Using a third party library, for example: “axios”.

vivasoft

Examples

```
let responseData = await fetch("https://jsonplaceholder.typicode.com/todos")
      .then((res) => res.json())
      .catch((err) => console.log(err));
```

```
import axios from "axios";

let responseData = axios
      .get("https://jsonplaceholder.typicode.com/todos")
      .then((res) => res?.data);
```

vivasoft

Handle API response data with state

We can store the response data from an API request in a state using the useState() hook or any kind of state management tool.

```
// .....
const [dataList, setDataList] = useState<TodoItemType[]>([]);

const fetchUserData = () => {
  let responseData = fetch("https://jsonplaceholder.typicode.com/todos")
    .then((res) => res.json())
    .then(res=>{
        setDataList(res);
    })
    .catch((err) => console.log(err));
};
// ....
```

vivasoft

Context API

Context API allows data to be passed through a component tree without having to pass props manually at every level

- Context API allows us to maintain a global state
- It is used to avoid props drilling too.

```
// Code example of using Context API to manage global state
const MyContext = createContext();

const MyProvider = ({ children }) => {
  const [globalState, setGlobalState] = useState(initialState);

  return (
    <MyContext.Provider value={{ globalState, setGlobalState }}>
      {children}
    </MyContext.Provider>
  );
};
```

vivasoft

Best Practices and Performance Optimization

- **Reconciliation**
  - React uses Virtual DOM (VDOM) to improve performance. It’s a copy of the real DOM. React compares the previous VDOM with the new one, identifies the differences, and updates only the parts of the DOM that have changed. Efficient reconciliation is crucial for a performant React application.

vivasoft

Best Practices and Performance Optimization

- **React.memo()**
  - It is a higher-order component that memoizes the rendering of a functional component. It prevents unnecessary re-renders by comparing the current props with the previous ones

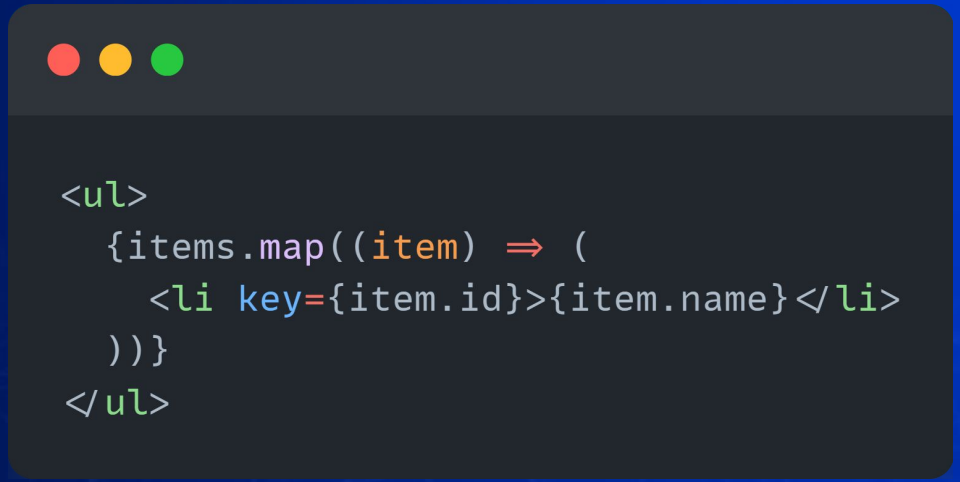
```
import React from 'react';

const MemoizedComponent = React.memo(function MyComponent(props) {
  // Component logic
});
```

vivasoft

Best Practices and Performance Optimization

- **Use Key Prop in Lists**
  - When rendering lists of components, always provide a unique key prop. React uses keys to identify elements efficiently during reconciliation.



```
<ul>
  {items.map((item) => (
    <li key={item.id}>{item.name}</li>
  ))}
</ul>
```

vivasoft

Best Practices and Performance Optimization

- **Use Code Splitting / lazy loading**
  - Use React.lazy() function to lazy load your code. This allows you to split your application into smaller chunks, loading only the code needed for a specific route or feature.

- Optimize Expensive Renders with **useMemo** and **useCallback**

vivasoft

Best Practices

- **Code organization**
  - Divide components based on functionality.
  - Separate presentational components from container/logic components.
  - Use a consistent folder structure for better navigation.
  - Group related components, styles, and assets in dedicated folders.
  - Consider modularizing reusable components

vivasoft

Best Practices

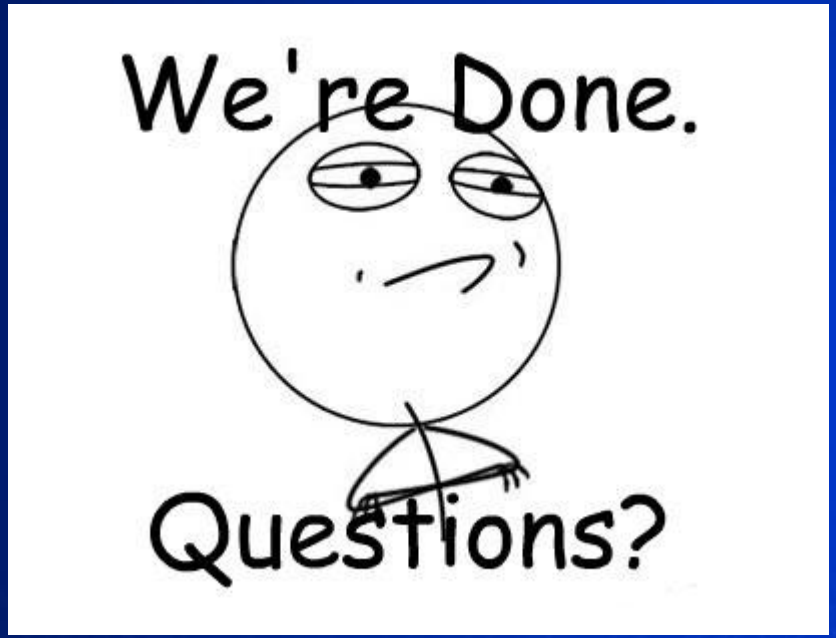
```
// An example of organizing your folder and components
src/
|-- components/
|   |-- DataList/
|   |   |-- Table.js
|   |   |-- Table.css
|-- layouts
|   |--AdminLayout.js
|-- utils/
|   |-- api.js
|   |-- helpers.js
|-- App.js
|-- index.js
```

vivasoft

Best Practices

- **Naming Conventions**:
  - Adopt a consistent and meaningful naming convention for files and variables.
  - Use camelCase for variables and PascalCase for component names.
  - Enhances code readability and maintainability.

vivasoft



vivasoft

Teamathon - 2.0