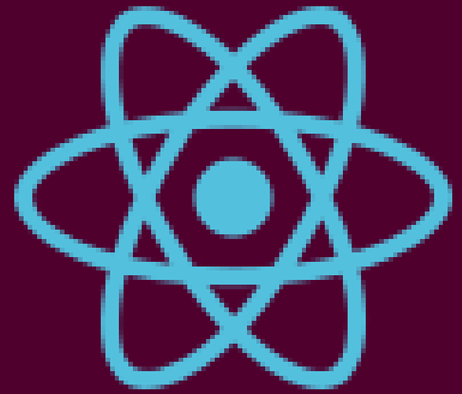
React.js hooks



Feruza Makhmudova

Software Engineer (4+ years prof. experience)

- Mcs in Computer science and software engineering
- **email:** feruza.maxmudova@ventionteams.com

Agenda:

- useEffect
- useContext
- useRef
- useReducer
- Custom hooks
- React.memo
- useMemo & useCallback
- React.lazy & Suspense

useEffect

Handling Side Effects

```jsx
import React, { useState, useEffect } from 'react';

function DataFetcher() {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch('https://api.example.com/data')
      .then(response => response.json())
      .then(data => setData(data));
  }, []); // Empty array means the effect runs only once after the initial render.

  return (
    <div>
      {data ? <pre>{JSON.stringify(data, null, 2)}</pre> : 'Loading...'}
    </div>
  );
}

export default DataFetcher;
```

useContext

Consuming Context

```
import React, { useContext } from 'react';

const ThemeContext = React.createContext('light');

function ThemedComponent() {
  const theme = useContext(ThemeContext); // Consume context value

  return <div>Current theme: {theme}</div>;
}

export default function App() {
  return (
    <ThemeContext.Provider value="dark">
      <ThemedComponent />
    </ThemeContext.Provider>
  );
}
```

useRef

Accessing DOM Elements

```jsx
import React, { useRef } from 'react';

function FocusInput() {
  const inputRef = useRef(null);

  const focusInput = () => {
    inputRef.current.focus(); // Access the input DOM element
  };

  return (
    <div>
      <input ref={inputRef} type="text" />
      <button onClick={focusInput}>Focus Input</button>
    </div>
  );
}

export default FocusInput;
```

useReducer

Advanced State Management

```jsx
import React, { useReducer } from 'react';

const initialState = { count: 0 };

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
    default:
      return state;
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);

  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: 'increment' })}>Increment</button>
      <button onClick={() => dispatch({ type: 'decrement' })}>Decrement</button>
    </div>
  );
}

export default Counter;
```

Custom Hooks

```
import { useState } from 'react';

function useLocalStorage(key, initialValue) {
  const [storedValue, setStoredValue] = useState(() => {
    const item = window.localStorage.getItem(key);
    return item ? JSON.parse(item) : initialValue;
  });

  const setValue = (value) => {
    setStoredValue(value);
    window.localStorage.setItem(key, JSON.stringify(value));
  };

  return [storedValue, setValue];
}
```

Custom Hooks

```
function App() {
  const [name, setName] = useLocalStorage('name', 'John');

  return (
    <div>
      <input value={name} onChange={e => setName(e.target.value)} />
      <p>Hello, {name}</p>
    </div>
  );
}
```

useState	Manage local state in function components
useEffect	Handle side effects such as data fetching, subscriptions, and manual DOM updates.
useContext	Access global data from context without prop drilling.
useRef	Create references to DOM elements or persist mutable values.
useReducer	Manage complex state logic when multiple state variables are involved.
Custom Hooks	Extract and reuse logic between multiple components.

Memoization: React.memo

```jsx
import React from 'react';

const ExpensiveComponent = React.memo(({ data }) => {
  console.log('Rendering ExpensiveComponent');
  return <div>{data}</div>;
});

function ParentComponent() {
  const [count, setCount] = React.useState(0);
  return (
    <div>
      <ExpensiveComponent data="This is expensive" />
      <button onClick={() => setCount(count + 1)}>Increment: {count}</button>
    </div>
  );
}
```

Memoization: useMemo

```jsx
import React, { useMemo, useState } from 'react';

function ExpensiveCalculation(num) {
  console.log('Running expensive calculation');
  return num * 2;
}

function MyComponent() {
  const [number, setNumber] = useState(0);
  const [count, setCount] = useState(0);

  const doubleNumber = useMemo(() => ExpensiveCalculation(number), [number]);

  return (
    <div>
      <input
        type="number"
        value={number}
        onChange={(e) => setNumber(parseInt(e.target.value, 10))}
      />
      <p>Double: {doubleNumber}</p>
      <button onClick={() => setCount(count + 1)}>Increment Count: {count}</button>
    </div>
  );
}
```

Memoization: useCallback

```jsx
import React, { useState, useCallback } from 'react';

function Child({ handleClick }) {
  console.log('Rendering Child');
  return <button onClick={handleClick}>Click me</button>;
}

const MemoizedChild = React.memo(Child);

function ParentComponent() {
  const [count, setCount] = useState(0);

  const handleClick = useCallback(() => {
    console.log('Button clicked');
  }, []); // Only recreated if dependencies change (none in this case)

  return (
    <div>
      <MemoizedChild handleClick={handleClick} />
      <button onClick={() => setCount(count + 1)}>Increment: {count}</button>
    </div>
  );
}
```

React.lazy & Suspense

```jsx
import React, { Suspense } from 'react';

// Lazy load the component
const LazyComponent = React.lazy(() => import('./LazyComponent'));

function App() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <LazyComponent />
      </Suspense>
    </div>
  );
}

export default App;
```

Avoiding Reconciliation

- Avoid **unnecessary re-renders** by using React.memo, useMemo, and useCallback.
- **Immutable state updates**: Always return new objects when updating state rather than mutating the existing state.
- **Use key prop correctly** in lists to help React identify which items have changed.

Windowing/Virtualization

```jsx
import { FixedSizeList as List } from 'react-window';

function VirtualizedList() {
  const items = Array.from({ length: 1000 }, (_, i) => `Item ${i}`);

  return (
    <List
      height={400}
      itemCount={items.length}
      itemSize={35}
      width={300}
    >
      {({ index, style }) => <div style={style}>{items[index]}</div>}
    </List>
  );
}

export default VirtualizedList;
```

Performance Monitoring and Profiling

- **React DevTools Profiler**: Allows you to profile components and see which ones are slow or re-rendering unnecessarily.

devtool demo

Any question?

Thank you very much!