Wykorzystanie hooków w React z TypeScript

1. useState

Hook `useState` pozwala na dodanie stanu do komponentu funkcyjnego.

Przykład:

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

const Counter: React.FC = () => {
  const [count, setCount] = useState<number>(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
};

export default Counter;
```

2. useEffect

Hook `useEffect` pozwala na wykonywanie efektów ubocznych w komponentach funkcyjnych.

Przykład:

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

const Timer: React.FC = () => {
  const [seconds, setSeconds] = useState<number>(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setSeconds(prevSeconds => prevSeconds + 1);
    }, 1000);

    return () => clearInterval(interval);
  }, []);

  return <div>Seconds: {seconds}</div>;
};

export default Timer;
```

3. useContext

Hook `useContext` pozwala na korzystanie z kontekstu w komponentach funkcyjnych.

Przykład:

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

interface UserContextType {
  name: string;
  age: number;
}

const UserContext = createContext<UserContextType | undefined>(undefined);

const UserProvider: React.FC = ({ children }) => {
  const user = { name: 'John Doe', age: 30 };

  return <UserContext.Provider value={user}>{children}</UserContext.Provider>;
};

const UserProfile: React.FC = () => {
  const user = useContext(UserContext);

  if (!user) {
    return <div>No user found</div>;
  }

  return (
    <div>
      <p>Name: {user.name}</p>
      <p>Age: {user.age}</p>
    </div>
  );
};

const App: React.FC = () => (
  <UserProvider>
    <UserProfile />
  </UserProvider>
);

export default App;
```

4. useReducer

Hook `useReducer` jest alternatywą dla `useState`, która jest bardziej odpowiednia dla złożonej logiki stanu.

Przykład:

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

interface State {
  count: number;
}

type Action = { type: 'increment' } | { type: 'decrement' };

const initialState: State = { count: 0 };

const reducer = (state: State, action: Action): State => {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
    default:
      return state;
  }
};

const Counter: React.FC = () => {
  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;
```

5. useRef

Hook `useRef` pozwala na tworzenie referencji, które mogą przechowywać wartości między renderami bez powodowania ponownego renderowania.

Przykład:

```tsx
import React, { useRef, useEffect } from 'react';

const TextInputWithFocusButton: React.FC = () => {
  const inputEl = useRef<HTMLInputElement>(null);

  const onButtonClick = () => {
    if (inputEl.current) {
      inputEl.current.focus();
```

```
    }
  };

  return (
    <div>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>Focus the input</button>
    </div>
  );
};

export default TextInputWithFocusButton;
```

To są podstawowe przykłady użycia hooków w React z TypeScript. Każdy z tych hooków ma swoje specyficzne zastosowania i może być używany w różnych scenariuszach w zależności od potrzeb aplikacji.