Contents

React

- Component based
- JS/TS function with JSX/TSX Expression
- Virtual DOM for optimized rendering

JSX

- Always needs **single parent element** (`<> </>` if necessary)
- `null` is valid JSX
- `{fn}` -> *fn* is replaced by evaluation result

Attributes:

- `className="validJS"`
- `htmlFor="string"` <- for is a reserved JS keyword
- `onClick={ () => fn() }`

Conditional Rendering:

- `{ bool ? "string" : "string" }` -> ternary expression

- `{ bool && "string"}` -> basically if statement

Map:

```
const items = ["Apple", "Banana", "Cherry"];
return (
  <ul>
    {items.map((item, index) => (
      <li key={index}>{item}</li>
    ))}
  </ul>
);
```

Inline CSS:

Indirect:

```
const style = { color: "red", fontSize: "20px" };
const element = <h1 style={style}>Styled Text</h1>;
```

Direct:

```
<h1 style={{ color: "blue", backgroundColor: "yellow" }}>Hello</h1>
```

Components

- Can manage its internal state
- Rerendered upon changes
- Props can be used to pass data between components

Early Return

```
if (todos.length === 0) {
    return ( // after this the next return will not be executed
        <>nothing to see here</>
    );
}
return ( <div></div>) // <- default return, won't render if todos.length === 0
```

Load Function on Component Load

```
doSmth();
// OR
if (todos.length === 0) {
    doSmth();
}
```

Subcomponent

Simple

```
import TodoList from './components/TodoList';

function App() {
    return (
        <div className="container">
           <TodoList />
        </div>
    );
}
```

Complex Possible props: - Functions - States - State setter - JSX

Pass props:

```
import React, { useState } from 'react';
import TodoItems from './TodoItems';
import AddTodoForm from './AddTodoForm';

function TodoList() {
  const [todos, setTodos] = useState(todosInit);

  // pass function as prop
  const doSmth = () => {
    console.log("I am doing something");
  }

  return (
    <div className="todo-list">
      <h2>My Todo List</h2>
      here we want AddTodoForm component
      <hr />
      <TodoItems todos={todos} setTodos={setTodos} onClick={doSmth}>
        {/* here we want to pass some children /}
        <p>Some children</p>
      </TodoItems>
      {/ A prop defines a call parameter for a component which is passed through a
      HTML attribute when the component is used in JSX code */}
      <AddTodoForm todos={todos} setTodos={setTodos} />
    </div>
  );
}

export default TodoList;
```

Use props:

```
function TodoItems({ todos, setTodos, onClick, children }) {
    const handleDone = (todo) => {
        const updatedTodos = todos.map(t => {
            if (t.id === todo.id) {
                return { ...t, status: "closed" };
            }
            return t;
        });
        setTodos(updatedTodos);
    };

    const openTodos = todos.filter(todo => todo.status === "open");
    const todoList = openTodos.map(todo => (
        <li key={todo.id}>
            { todo.description }
            <button className="done" onClick={() => handleDone(todo)}>done</button>
        </li>
    ));

    return (
      <>
        <h3>Current Todos</h3>
        <ul> {todoList} </ul>

        Alternatively, you can directly map over the todos prop:
        <ul>
            {openTodos.map(todo => (
                <li key={todo.id}>{ todo.description }</li>
            ))}
        </ul>
        {children}
        <button onClick={onClick}>Click me</button>
      </>
    ); // render root element with heading and list of todos
// OR
function TodoItems(props) {
    const todos = props.todos;
    return(
        <>
            {props.children}
            <button onClick={props.onClick}>Click me</button>
        </>
    );
}
```

Wrapper

Definition

```
const Wrapper = ({ children }) => {
    return <div className="wrapper">{children}</div>;
};
```

Usage

```
const App = () => {
    return (
        <Wrapper>
            <h1>Hello, World!</h1>
            <p>This is inside the wrapper.</p>
        </Wrapper>
    );
};

export default App;
```

Hooks

useState

State management.

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

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

  const handleClick = (event) => {
    event.preventDefault();
    setCount(count + 1);
  };

  return (
    <div className="container">
      <p>Count: {count}</p>
      <button onClick={handleClick}>Increment</button>
    </div>
  );
}

export default Counter;
```

useEffect

useEffect is a React Hook that lets you synchronize a component with an external system reading asynchronous data (e.g. fetching data from an API)

Single Run

```
import { useState, useEffect } from "react";

const [queryResult, setQueryResult] = useState([]);
const [isLoading, setIsLoading] = useState(false);

async function fetchData() {
    try {
        setIsLoading(true);
        const response = await fetch(`${URL}${searchTerm}`);
        const json = await response.json();
        setIsLoading(false);
        setQueryResult(json.hits);
    } catch (err) {}
}

// will be executed once, direct fetchData() call would lead to a infinity loop
useEffect(() => {
    fetchData();
}, []);
```

On State Change

```
import { useState, useEffect } from "react";

const MultiDependencyExample = () => {
  const [count, setCount] = useState(0);
  const [name, setName] = useState("");

  useEffect(() => {
    console.log(`Count: ${count}, Name: ${name}`);
  }, [count, name]); // Runs when either count or name updates
};
```

useContext

Global state management. Usually, you will pass information from a parent component to a child component via props. But passing props can be inconvenient if you have to pass them through many components (prop drilling). Context lets the parent component make some information available to any component in the tree below it—no matter how deep—without passing it explicitly through props.

Provider

```
import { useState, useContext } from "react";

function useContextProvider() {
    const UserContext = createContext();
    const [user, setUser] = useState("Jesse Hall");

    return (
        <UserContext.Provider value={user}>
            <h1>{Hello ${user}!}</h1>
            <UseContextComponent />
        </UserContext.Provider>
    )
}
```

Usage

```
import { useContext } from "react";

function UseContextComponent() {
    const user = useContext(UserContext);

    return (
        <>
            <h1>Component 2</h1>
            <h2>{`Hello ${user} again!`}</h2>
        </>
    );
}
```

useRef

useRef is a React Hook that lets you reference a value that's not needed for rendering.

```
import { useRef } from "react";

const inputRef = useRef();

    const send = () => {
        inputRef.current.focus(); // set scroll focus
    }

    function UseRefComponent() {
        return (
            <>
```

```
                <input ref={inputRef} />
                <button onClick={send}>Send</button>
            </>
        )
    }
```

useReducer

Components with *many state updates* spread across *many event handlers* can get overwhelming. For these cases, you can *consolidate all the state update logic* outside your component in a single function, called a **reducer**.

Simple

```
import { useReducer } from "react";

const reducer = (state, action) => {
  switch (action.type) {
    case "increment":
      return { count: state.count + 1 };
    case "decrement":
      return { count: state.count - 1 };
    case "reset":
      return { count: 0 };
    default:
      return state;
  }
};

const Counter = () => {
  const [state, dispatch] = useReducer(reducer, { count: 0 });

  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: "increment" })}>+</button>
      <button onClick={() => dispatch({ type: "decrement" })}>-</button>
      <button onClick={() => dispatch({ type: "reset" })}>Reset</button>
    </div>
  );
};

export default Counter;
```

Complex

```
import { useReducer } from "react";

function todosReducer(todos, action) { // todos is the current state
    switch(action.type) {
      case 'addTodo': {
        const newTodo = {
          id: Date.now(),
          description: action.description,
          status: "open",
        };
        console.log(adding new todo: id=${newTodo.id}, description = ${action.description});
        return([...todos, newTodo]);
      }
      case 'done': {
        return todos.map((t) => {
          if (t.id === action.todo.id) {
            t.status = 'closed';
            return t;
          }
          return t;
        });
      }
      default: {
        throw Error('Unknown action: ' + action.type);
      }
    }
}

function UseReducerComponent() {
    const [todos, dispatch] = useReducer(todosReducer, todosInit);

    const onAddTodo = (description) => {
        dispatch({
            type: 'addTodo', // action type
            description: description // additional data
        });
    }

    const onDone = (todo) => {
        dispatch({
            type: 'done', // action type
            todo: todo // additional data
        });
    }

    return (
```

```
        <div>
            {todos.map((todo) => (
                <div key={todo.id}>
                    <span>{todo.description}</span>
                    <button onClick={() => onDone(todo)}>Done</button>
                </div>
            ))}
        </div>
    )
}
```

useCallback / memo

useCallback is a React Hook that lets you cache a function definition between re-renders. **memo** lets you skip re-rendering a component when its props are unchanged

- components are rerendered when a state changes
- rerender: rerun component code
- sometimes this slows app down due to expensive calculations
- only use the hooks if encountering performance issues because adds complexity

```
// example unnecessary rerendering: parent state changes without effecting children
// -> all children rerender nonetheless
const [count, setCount] = useState(0);
const [todos, setTodos] = useState([]);

const increment = () => {
    setCount((c) => c + 1);
};

const addTodo = () => {
    setTodos((t) => [...t, "New Todo"]);
};

function TodoList() {
    return (
        <>
            <Todos todos={todos} addTodo={addTodo} />

            Count: {count}
            <button onClick={increment}>+</button>
            {/* causes Todos to rerender
                state change -> rerender -> rerun code
                    -> addTodo reference changes -> Todos (child) rerenders
            */}
```

```
        </>
    )
}

function Todos() {
    return (
        <></>
    )
}
```

Solution 1 **memo**: only rerenders Todos if props change

```
function Todos() {
    return (
        <></>
    )
}
export default memo(Todos);
```

Solution 2 **useCallback**: aches the reference to the arrow function and always returns the cached reference

```
const addTodo = useCallback(() => {
    setTodos((t) => [...t, "New Todo"]);
}, [todos]);
```

Forms

Simple

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

function AddTodoForm({ todos, setTodos }) {
    const [description, setDescription] = useState("");

    const handleDescriptionChange = (e) => {
        const value = e.target.value;
        setDescription(value);
    };

    const handleAdd = () => {
        e.preventDefault();
        const newTodo = {
           id: Date.now(),
           description: description,
           status: "open",
        };
```

```
        setTodos([...todos, newTodo]);
        setDescription("");
    };


    return (
      <>
        <label htmlFor="new-todo-id">new todo:</label>
        <input type="text" id="new-todo-id"
            value={description}
            onChange={handleDescriptionChange}
        />
        <button type="button" onClick={handleAdd}>add</button>
      </>
    );
}

export default AddTodoForm;
```

Complex

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

function AddTodoForm({ todos, setTodos }) {
    // for more complex forms
    const [form, setForm] = useState(
        { description: "", status: "open" }
    );

    const handleDescriptionChange = (e) => {
        const value = e.target.value;
        setForm({ ...form, description: value });
    };

    const handleAdd = (e) => {
        e.preventDefault();
        const newTodo = {
            id: Date.now(),
            description: form.description,
            status: form.status,
         };
         setTodos([...todos, newTodo]);
         // reset form if needed
    };

    return (
```

```
      <>
        <form onSubmit={handleAdd}>
            <input type="text" id="new-todo-id"
                value={description}
                onChange={handleDescriptionChange}
            />
            <button type="submit">add</button>
        </form>
      </>
    );
}

export default AddTodoForm;
```

Router

- Allows for client side navigation
- `npm install react-router-dom`

`Router`: Wraps the app and provides routing functionality `Route`: renders a component based on the current URL `Routes`: renders the first matching route `Link`: navigates to a different URL `Outlet`: renders the child route

```
import {
    BrowserRouter as Router,
    Routes, Route, Link, Outlet,
    useNavigate, useParams
} from "react-router-dom";


function App() {
    const navigate = useNavigate();

    const navigateHome = () => {
        navigate('/');
    };

    return (
        <Router>
            <ul>
                <li><Link to="/">Home</Link></li>
                <li><Link to="/about">About</Link></li>
            </ul>

            Alternative to Link:
            <button onClick={navigateHome}></button>
```

```
            <Routes>
                <Route path="/" element={<Home />} />
                <Route path="/about" element={<About />} />

                <Route path="/optional" element={<ProtectedRoute />}>
                    <Route path="/dashboard" element={<DashboardComponent />} />
                    <Route path="/profile:id" element={<ProfileComponent />} />
                </Route>
            </Routes>
        </Router>
    )
}

function ProtectedRoute() {
    // check auth
    return (
        <div>
            <h1>Protected Route</h1>
            <Outlet /> {/* renders the child routes */}
        </div>
    )
}

function ProfileComponent() {
    const { id } = useParams();
    return (
        <h1>{`Profile ${id}`}</h1>
    )
}

}
```