08. React
Part3

@Pilot - Christine

Attention!

This training is **interview-driven**.

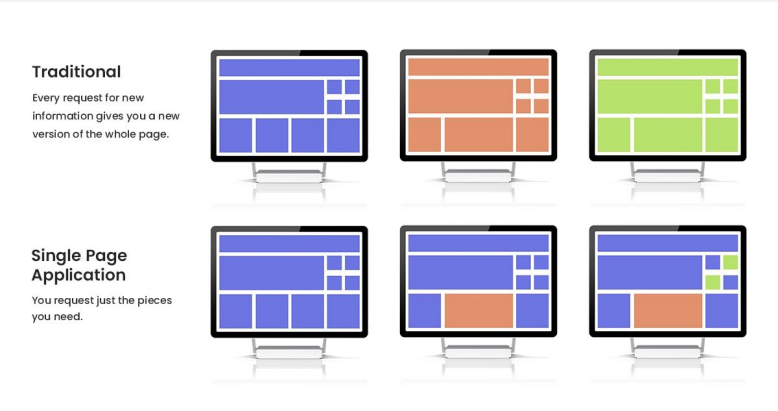
To become a qualified developer, *ChatGPT* and *YouTube* will always be your best friends.

Outline

- SPA
- React Route
- Props drilling
- Context API
- Styles
- Optimize Performance

SPA

- **Single-Page Application (SPA)**: The entire application runs on a single HTML page, and navigation between different "pages" happens without reloading the whole page
  - **Components**: UI is split into reusable components
  - Routing: navigates without a full-page reload
    - The URL dynamically determines which component to render, instead of reloading the HTML page
  - Virtual DOM



React Router (v6)

A router is what lets your app have multiple pages (URLs) without reloading the browser

- **BrowserRouter**: Wraps your app and enables routing functionality
- **Routes and Route**: Define which path shows which component
- **Link**: Used instead of <a> to navigate without reloading the page
- **Nested** route
- URL Parameters (Dynamic routes)
- *

```
function App() {
  return (
    <BrowserRouter>
      {/* Navigation */}
      <nav>
        <Link to="/">Home</Link> |{" "}
        <Link to="/about">About</Link> |{" "}
        <Link to="/contact">Contact</Link>
      </nav>

      {/* Routes */}
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
      </Routes>
    </BrowserRouter>
  );
}
```

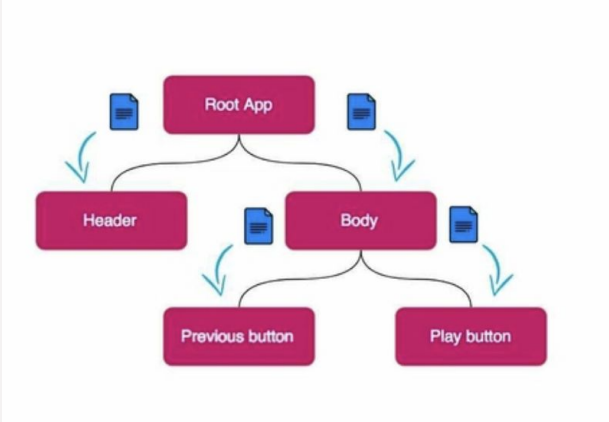
React Router hooks

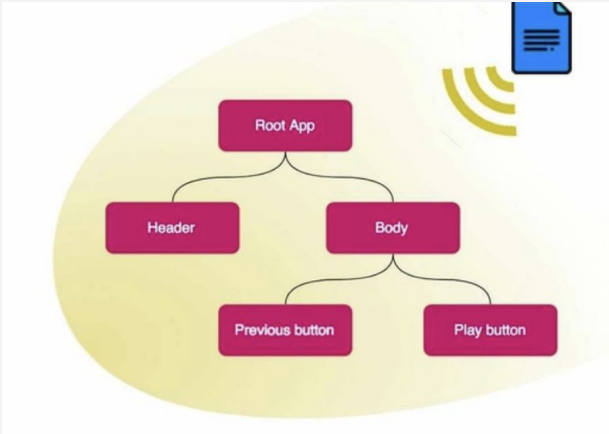
- **useNavigate**: change the URL (go somewhere)
- **useParams:** read URL path params
- **useLocation:** read full location info (path, query, state)

Hook	What it does	What it returns	Typical use cases	Example
`useParams`	Reads **dynamic parameters from the URL path**	An object of key-value pairs (strings)	Detail pages, edit pages, nested resources	`/user/3` → `{ id: "3" }`
`useNavigate`	**Programmatically navigates** to another route	A function (`navigate`)	Redirect after submit, login/logout, back button	`navigate("/home")`
`useLocation`	Provides **full information about the current URL**	A location object (`pathname` , `search` , `state` , `hash`)	Reading query params, tracking navigation source	`location.pathname === "/user/3"`

Props drilling

- **Props (properties)**: how data is passed from a parent component to a child component
- Props are read-only, one-way (from parent → child), and allow components to be reusable and dynamic
- **Props Drilling** happens when you have to pass props through **multiple** layers
    - Middle components don’t need it
    - Context API, State Management Libraries (Redux, Zustand)…





Context API

- Lets you create “global variables” that can be accessed from any component **without prop drilling**
  - createContext
  - Provider (set the value)
  - **useContext** (read the value)
- Use cases
  - Theme (light / dark)
  - Language (i18n)
  - Auth/Permissions

```jsx
const Context = React.createContext();

const MyGrandChild = () => {
  const { fruit, setFruit } = useContext(Context);
  return <button onClick={() => setFruit('banana')}>{fruit}</button>
};

const MyChild = () => {
  return <MyGrandChild />;
};

const MyParent = () => {
  const [fruit, setFruit] = useState('apple');
  return (
    <Context.Provider value={{ fruit, setFruit }}>
      <MyChild />
      {fruit}
    </Context.Provider>
  );
};
```

Styles in React

- Inline Styles
  - Styles are JavaScript **objects**
  - style={{ }} (camelCase)
  - NO :hover, :focus, media queries
- CSS Stylesheets
  - **Global** scope -> override, naming conflicts
- CSS Modules
  - import styles from './xxxx.module.css';
  - local-scoped
  - automatically scope styles by renaming class names at build time (e.g. .button → .Button_button__a1b2c3)

Styles in React

- Styled Components (CSS-in-JS)
  - import styled from 'styled-components';
- Conditional Styles
  - with classnames
- CSS preprocessor: Sass **SCSS**
  - CSS with extra features: Variables, Nesting, Mixins, Functions
- CSS framework: MUI/Tailwind

Optimize Performance

- Measurement
  - React Profiler: unnecessary re-renders
  - Chrome Performance: JS vs layout vs network
- Reduce unnecessary re-renders
  - **React.memo, useCallback, useMemo**
- Handle expensive work & large data
  - useMemo for expensive calculations
  - List virtualization (react-window) for large tables
  - Pagination / infinite scroll
  - React 18: useTransition / useDeferredValue for non-urgent updates

Optimize Performance

- Architecture & bundle size
  - Split large components
  - Code-splitting with React.lazy
    - Lazy loads the component on demand
    - Uses Suspense to show a fallback UI
  - Cache server state (React Query / SWR)
- **Debouncing/Throttling**
  - Debounce API calls (e.g., Input fields, autocomplete, search)

Additional materials

- <Routes> vs <Switch>
- React 18 new features
- SCSS

Homework - Requirement

- 全程recording: 八股闭眼 & 摘耳机, 无AI辅助
- 全程recording: Coding可适当AI辅助, 但需写完(边解释边写), 模拟真实面试情景
- 录音提交地址: https://drive.google.com/drive/folders/1Mrm341uOIS8c2Ty6NwYvvSybHa6hESem?usp=drive_link
- 提交格式: 小组+日期, 例如Group1_12/07/2025
- 提交截止时间: due第二天5PM (周四作业递延到下周一)

Homework - Class Code

- Write the code as taught in class, take a screenshot of your code, and post it in the Wechat group. **It's due the same night.**

Homework

- ❏ Code along with this video:

  https://codetv.dev/series/learn-with-jason/s4/let-s-learn-modern-redux
- ❏ take a screenshot of your code, and post it in the Wechat group

Homework - Short Answer Questions

1. What is React strict mode?
2. What are React 18 new features
3. How to use styles in React?
4. What is lazy loading and what does it help?
5. When coding React, what are some best practices that you keep in mind?
6. What is debouncing?
7. How could you do to improve performance in React?
8. What is Context API?
9. What is prop drilling?
10. Have you built any reusable components? What are best practices?

Homework - Short Answer Questions

11. How to update a component every second?
12. Do Hooks replace render props and higher order components?
13. What are the possible ways of updating objects in state?
14. What is the difference between a `<Link>` and an `<a>` tag?
15. How do you prevent unnecessary re-renders?
16. Why are keys important in lists?
17. How do you optimize React App?
18. What is <Route> used for?
19. How do you navigate using react-router?
20. How do you redirect a user in React Router?

Homework - Short Answer Questions

21. How to acquire the user's current URL params?
22. How do React handle errors?
23. How are we going to render a variable as a react component?
24. What is the lifecycle method `componentDidUpdate` equivalent hook?
25. What is the lifecycle method `componentDidMount` equivalent hook?
26. What is the lifecycle method `componentWillUnmount` equivalent hook?

Homework - Coding Questions

1. Toggle

- Create a parent component and child component. The parent component has a property that contains a list of two languages: [“JavaScript”, “Java”]. By default, it displays the first language, “JavaScript”. This property is sent to the child component.
- The child component renders a button that can be clicked to toggle the view (updates the parent component). It should toggle from “JavaScript” to “Java”, or vice versa.
- Do it again using the Context API.

Homework - Coding Questions

2. Build a functional component with React hooks:

- Render a Todo List page.
- The page should contain a text input and an Add button:
  - When the input is not empty, clicking Add should create a new todo at the top of the list.
  - Pressing Enter should also submit.
  - After submission, clear the input.
- Each todo item should display:
  - The text content
  - An Edit button (switch to editable state, save with Enter, cancel with Esc)
  - A Delete button (remove the todo; optional: confirm before delete)

Homework - Coding Questions

Build a functional component with React hooks:

- Support toggle completed state (checkbox or button). Completed items should have different styling (e.g., strikethrough or opacity).
- Use a stable key when rendering list items (e.g., id instead of array index).
- Implement input validation and error handling:
  - Prevent adding empty/whitespace-only todos.
  - Apply the same validation when editing.
  - Show user-friendly feedback for errors (e.g., toast, text message).
- Use clear state management (e.g., todos, inputValue, editingId).

Homework - Coding Questions

Build a functional component with React hooks:

- Extra Credit
  - Create a custom hook (e.g., useTodos()) to encapsulate add/edit/delete logic.
  - Use useReducer to manage todos with actions (ADD, DELETE, TOGGLE, EDIT).
  - Improve accessibility: add aria-labels for buttons/inputs and support keyboard navigation.

Homework - Coding Questions

Build a functional component with React hooks:

- Suggested Data Structure
- Tips:
  - Use crypto.randomUUID() or libraries like ulid for IDs.
  - For edit mode: controlled input + editingId or inline form.
  - Filters can be managed with state for advanced cases.

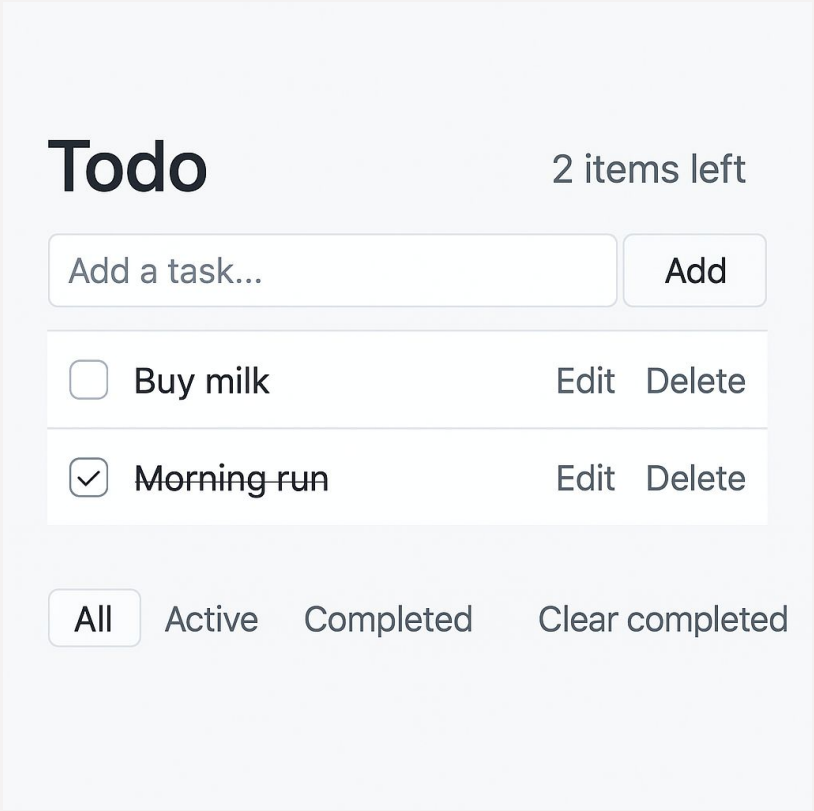
```
{
  "id": "uuid-or-ulid",
  "text": "Buy milk",
  "completed": false,
  "updatedAt": 1736033112000
}
```

Homework - Coding Questions

Build a functional component with React hooks:

- Acceptance Criteria
  - Can add a new todo when input is not empty. Input clears after add.
  - Can delete, edit, and toggle completed state. Edit supports Enter (save) and Esc (cancel).
  - List rendering uses stable keys, no console warnings.
  - User feedback provided (disabled buttons, error messages, etc).
  - Code is well-structured: components, hooks, or reducer (if Extra Credit is attempted).

Homework - Coding Questions



@Pilot - Christine

Homework - Coding Questions

- Edit mode