

<https://codesandbox.io/s/lecture-14-9hy2zy>

Topics

- Conditional Rendering
- Forms
- HTTP Requests
- High-Order Components
- Pure Components
- `lazy` & `React.Suspense`
- `React.StrictMode`
- Component Library

Conditional Rendering

- `if` / `switch` statement
- `&&` operator
- `ternary` operator

Forms in React

- Controlled Components
- `onSubmit` event
- `event.preventDefault()`
- validation

HTTP Requests in Class Components

- `fetch` API / `axios`
- `componentDidMount`

High-Order Components

- A function that takes a component and returns a new component
- Used to share common functionality between components
- Example:
 - `withRouter` from `react-router-dom`
 - `connect` from `react-redux`

Pure Components

PureComponent.jsx
SkipRender.jsx

- Shallow comparison of props and state
- `shouldComponentUpdate` has to be implemented by yourself
- Consider using the built-in `PureComponent` instead of writing `shouldComponentUpdate()` by hand.

lazy & React.Suspense

- Code splitting
- `React.Suspense` is a component that lets you wait for some code to load and declaratively specify a loading state (like a spinner) while we're waiting.

Strict Mode

- Strict mode can't automatically detect side effects for you, but it can help you spot them by making them a little more deterministic.
- recognize unsafe lifecycles
 - `UNSAFE_componentWillMount`
- legacy string ref API usage
- legacy context API
- warn about deprecated `findDOMNode` usage
- **only** in development mode

Strict Mode (cont'd)

- Detecting unexpected side effects
 - React does work in two phases: render and commit.
 - Render phase includes some lifecycle methods and `setState` updater functions (first argument)
 - Strict mode helps with problems in render phase by intentionally **“double rendering”** the component tree

Component Library

- [React Bootstrap](#)
- [Material UI](#)
- [Ant Design](#)
- [Semantic UI](#)
- [Chakra UI](#)

Material UI

- Install
 - v4: `npm install @material-ui/core`
 - v5: `npm install @mui/material @emotion/react @emotion/styled`
- <https://github.com/mui/material-ui/tree/master/examples/material-ui-cra>
- [Templates](#)

Ant Design

- Install: `npm install antd`
- <https://ant.design/components/overview>
- <https://codesandbox.io/s/antd-reproduction-template-forked-vrgn4p?file=/index.js>

Different Types of Components

- **Function Components** vs Class Components
- Presentational Components vs Container Components
- Stateless Components vs Stateful Components

Hooks

- What are hooks?
 - Generally speaking, hooks are functions registered for a specific time and will be called when that time comes along with system processing.
 - A way to use state and other React features without writing a class (function components **ONLY**)
- Why hooks?
- How to use hooks?

Why Hooks?

- Pure function -> **F(state) = UI**
- Function components
 - No **constructor**
 - No **this** keyword (no binding)
 - No **lifecycle** methods
 - No **render** method
- Class components
 - data and behavior are not organized in a single place

How to use Hooks?

- `useState` - manage state
- `useEffect` - manage side effects
- `useRef` - access DOM nodes / store mutable values
- `useMemo` - optimize expensive calculations
- `useCallback`
- `useContext`
- `useReducer`
- Custom hooks

useState

- `const [state, setState] = useState(initialState);`
- `setState` can be called with a new value or a function
- `setState` does not automatically merge update objects
- `setState` is asynchronous
- `setState` has no callback function as the second argument

useEffect

- `useEffect` is a hook that lets you perform side effects in function components
- `useEffect` runs
 - after the first render (componentDidMount)
 - after every update (componentDidUpdate)
 - after specific values have changed (componentDidUpdate)
 - before unmounting (componentWillUnmount)

When to use `useEffect` ?

- Data fetching
- Setting up a subscription
- Manually changing the DOM
- Reading from local storage
- Cleaning up before component unmounts

Hooks Rules

- Only call hooks at the top level
- Only call hooks from React function components
- Don't call hooks inside loops, conditions, or nested functions

Custom Hooks

- A custom hook is a JavaScript function whose name starts with `use` and that may call other hooks
- Custom hooks are a convention that naturally follows from the design of hooks, rather than a React feature