8 - React (State management)

React education, 2024.

MITHRIL

Overview

- React state management
- Zustand
- Hands-on

React state management

React state management

- State
- Component state
- useReducer
- Context API
- State management solutions

State

- In an application, state is an interface between your data from any kind of backend or local change and the representation of this data with UI elements in the frontend.
- State is able to keep data of different components in sync because each state update will re-render all relevant components.
- State can be a medium to communicate between different components aswell.
- In React there are many different paradigms and libraries to keep your state managed.

Component state

- State inside the component:
  - The most straightforward way is to define a state inside the component.
  - The state of a component is like the props which are passed to a component, a plain JS object containing information that influences the way a component is rendered.
  - In comparison to the props, state can be changed by component itself.
  - State API of React is really simple at all and doesn’t add too much complexity to your application.
  - Besides all other state management solutions, the component state is preferred to not be replaced at all because you should always keep your state as close to where it is needed to avoid unnecessary complexity.

MITHRIL

Component state

- It is totally possible to write your application just with **component state**, but as your component dependencies and the size and complexity of your app grows, dealing with state will become more complex.
- In more complex apps, distance between your state locations and your components could increase.
- This leads to props drilling, meaning passing props through components which don’t need the props but their children do.
- We need to avoid this because it increases the complexity in development and also in refactoring.

useReducer

- The useReducer is an alternative to the `useState` hook for managing state in functional components.
- The `useReducer` hook is better suited for managing complex state logic while `useState` is best for simple state changes.

Context API

- Context API provides an internal solution for passing state where it is needed and avoids the possibility of props drilling.
  - If we want to use Context API we must define Provider that provides a certain component state to all Consumers that are located somewhere in the React component tree below the provider.
  - Beneath the state, a provider can also provide functions to manipulate the state within the provided values.
  - The Consumer accepts a render function and is able to access the value prop from the Provider.
- Context API is perfect solution to pass data that is not so complex.

State management solutions

- As you can see, **component state, useReducer** and **Context API** has their own purpose, and both of the solutions are adequate to some specific use cases.
- Context API has a lot of possible use cases, also useReducer is suited for more complex situations of handling state, but there are also libraries that are made only for more complex cases.
- When you need to manage state of applications with higher complexity try to decide between one of the following libraries:
  - Zustand
  - Jotai
  - Redux toolkit (RTK)
  - Recoil
  - TanStack Query

Zustand

Zustand

- A small, fast and scalable bearbones state-management solution using simplified flux principles. Has a comfy API based on hooks, isn't boilerplatey or opinionated.
- Why Zustand over context?
  - Less boilerplate
  - Renders components only on changes
  - Centralized, action-based state management
  - Context is not meant to be used as a choice for state management, but rather as a Provider of some shared App data (Theme, Language, Auth)

MITHRIL

Hands-on

Hands-on

- Find installation script for Zustand on npm.
- Install Zustand in your project.
- Create a simple store for TODOs which will replace previously made Component state.