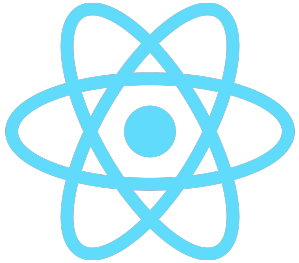


React.js

Lecture 4

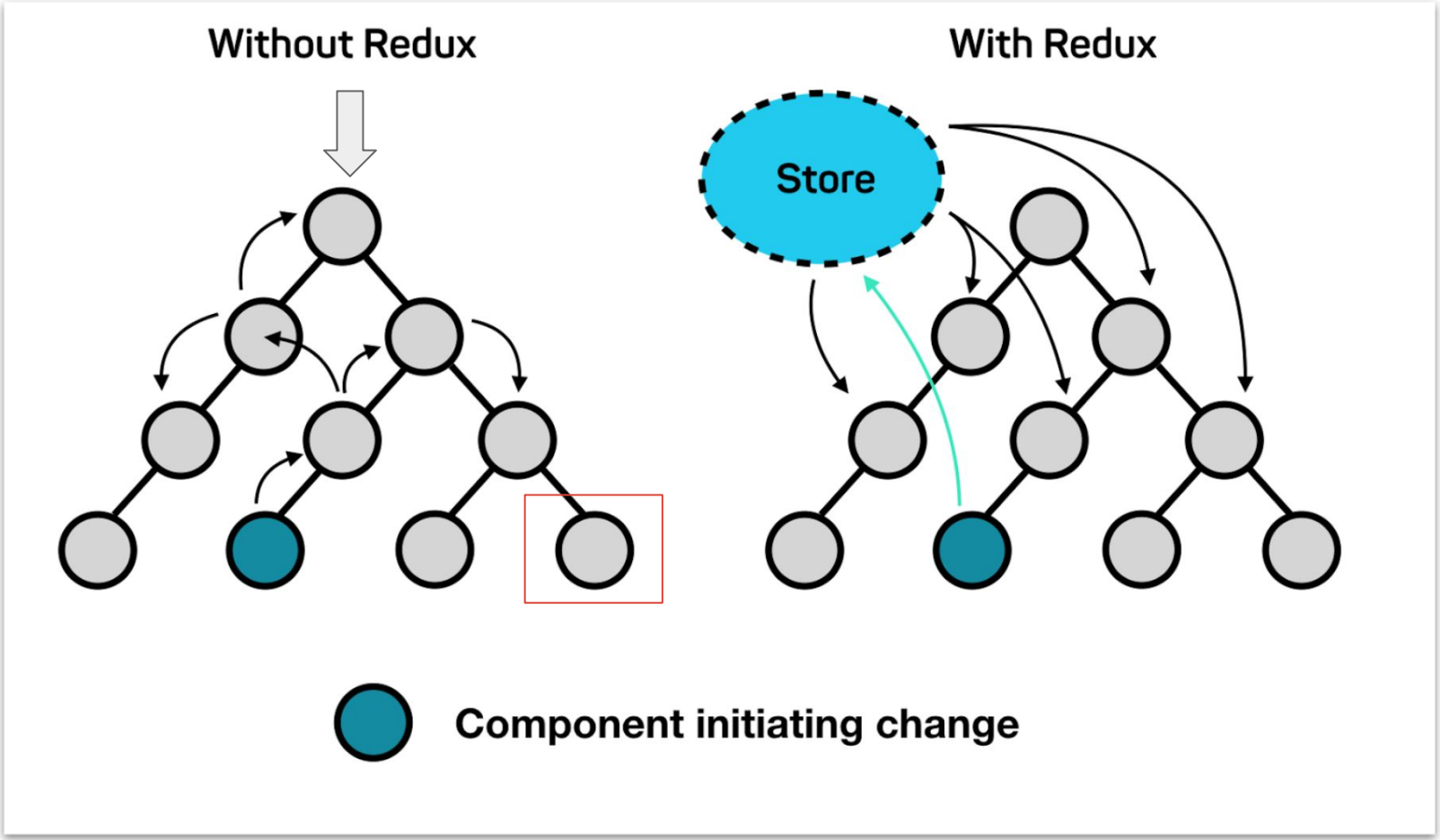
Agenda

- Recap last lecture points
- What is the state Management concept
- What is Redux ?
- Redux toolkit
- Configure redux toolkit
- Useful extensions
- Context API
- Questions!

State Management

What is the state management?

React applications are built using components, and they manage their state internally, and it works well for applications with few components, but when the application grows bigger, the complexity of managing states shared across components becomes difficult.


Without Redux
With Redux
Store
Component initiating change

Redux

Redux is a predictable state container and the state of your application is kept in a store, and each component can access any state that it needs from this store.

Redux is not only for react, It could be used with Angular and vue or JS too, it's a JS library.

We will be using the latest version of redux which is Redux toolkit

To Install Redux toolkit you will need to run the following commands:

- **npm install @reduxjs/toolkit react-redux**

https://redux.js.org/introduction/why-rtk-is-redux-today

What is Redux ?

The first thing to ask is, "what is Redux?"

Redux is really:

- A single store containing "global" state
- Dispatching plain object actions to the store when something happens in the app
- Pure reducer functions looking at those actions and returning immutably updated state

Redux Toolkit is designed to solve those problems!

- Redux Toolkit simplifies store setup down to a single clear function call while retaining the ability to fully configure the store's options if you need to.

- Redux Toolkit eliminates the need to write any action creators or action types by hand.

- Redux Toolkit makes it easy to write a Redux feature's code in one file, instead of spreading it across multiple separate files.

Why should I use Redux?

- **Centralized state management:** With Redux, you can maintain the state of your entire application in a single store, making it easier to manage and access data across components.
- **Predictable state updates:** Redux has a clear flow of data, which means changes to the state can only happen when you create an action and send it through Redux. This makes it easy to understand how your application's data will change in response to user actions.
- **Easier debugging:** With Redux DevTools, you have a clear record of all the changes to your application's state. This makes locating and fixing issues in your code easier, saving you time and effort in the debugging process.
- **Better performance:** By minimizing the number of state updates and reducing the need for prop drilling, Redux helps improve your application's performance.

How Redux works :

The whole global state of your app is stored in an object tree inside a single store. The only way to change the state tree is to create an action, an object describing what happened, and dispatch it to the store. To specify how state gets updated in response to an action, you write pure reducer functions that calculate a new state based on the old state and the action.

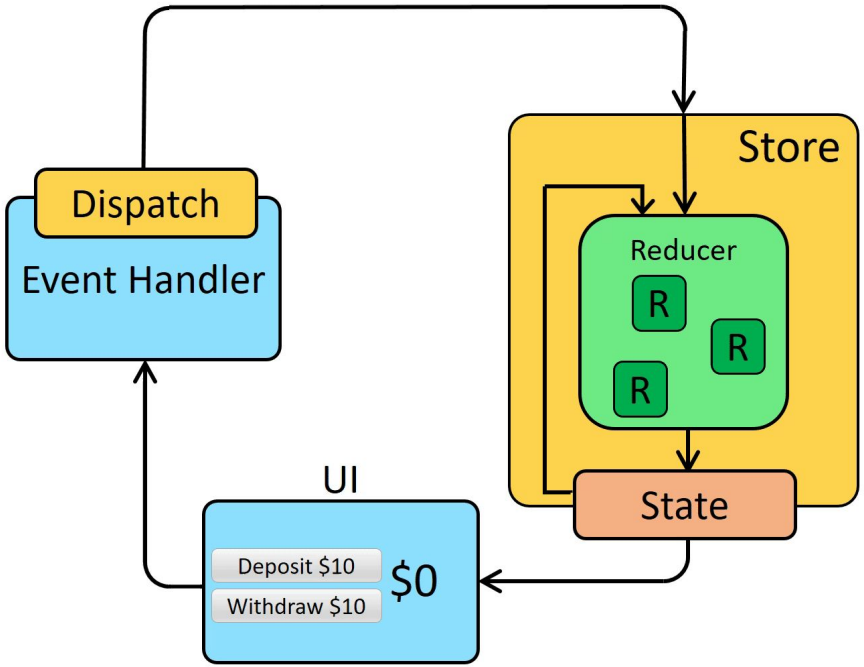
https://www.freecodecamp.org/news/redux-and-redux-toolkit-for-beginners/

How Redux works

There is a central store that holds the entire state of the application. Each component can access the stored state without having to send down props from one component to another.

There are three building parts:

- actions
- store
- reducers



How Redux works :

configureStore accepts a single configuration object parameter.

configureStore call automatically does all the usual setup work you'd have done manually:

- The slice reducers were automatically passed to combineReducers()
- The redux-thunk middleware was automatically added
- Dev-mode middleware was added to catch accidental mutations
- The Redux DevTools Extension was automatically set up
- The middleware and DevTools enhancers were composed together and added to the store

https://redux.js.org/introduction/why-rtk-is-redux-today#what-does-redux-toolkit-do

How Redux works :

Create Redux Store:

```
import { configureStore } from '@reduxjs/toolkit';


export default configureStore({
  reducer: {},
});
```

How Redux works : What is Redux Slice...

- A slice is the portion of Redux code that relates to a specific set of data and actions within the store‘s state.
- Redux Toolkit has a function called createSlice, which takes care of the work of generating action type strings, action creator functions, and action objects. All you have to do is define a name for this slice, write an object that has some reducer functions in it, and it generates the corresponding action code automatically.
- createSlice automatically generates action creators with the same names as the reducer functions we wrote, It also generates the slice reducer function that knows how to respond to all these action types
- Think of the Redux store as a cake, where each slice represents a specific piece of data in the store. By creating a slice, you can define the behaviour of the state in response to particular actions using reducer functions.

https://redux.js.org/tutorials/essentials/part-2-app-structure#creating-slice-reducers-and-actions

How Redux works :

Create Slice

```
export const counter = createSlice({
  name: counter,
  initialState: [],
  reducers: {},
});
```

How Redux works :

Create Slice : Reducer

```
reducers: {
    increaseCounter: (state, action) => {
      state.counter = action.payload;
  },
```

The **createSlice** function automatically generates action creators and action types based on the names of the reducer functions you provide. So you don't have to define the action creators yourself manually.

How Redux works :

Create Slice : Export action and reducer

```
// this is for dispatch in component
export const { increaseCounter} = todoSlice.actions;

// this is for configureStore
export default increaseCounter.reducer;
```

How Redux works : Wrap application with Redux

This makes your whole app could access the Redux. Wrap your <App /> component with the Provider and include the store that you made recently.

Syntax:

```
<Provider store={store}>
    <App />
</Provider>
```

Read And Update Store in Functional Components

Redux Hooks

UseSelector()

To read and get different store values

Syntax :

```
const state = useSelector(state => state)
```

UseDispatch()

To Update store values and dispatch actions

Syntax :

```
const dispatch = useDispatch();

dispatch(action());
```

Useful extensions

- React Developer Tools
- Redux DevTools

Context

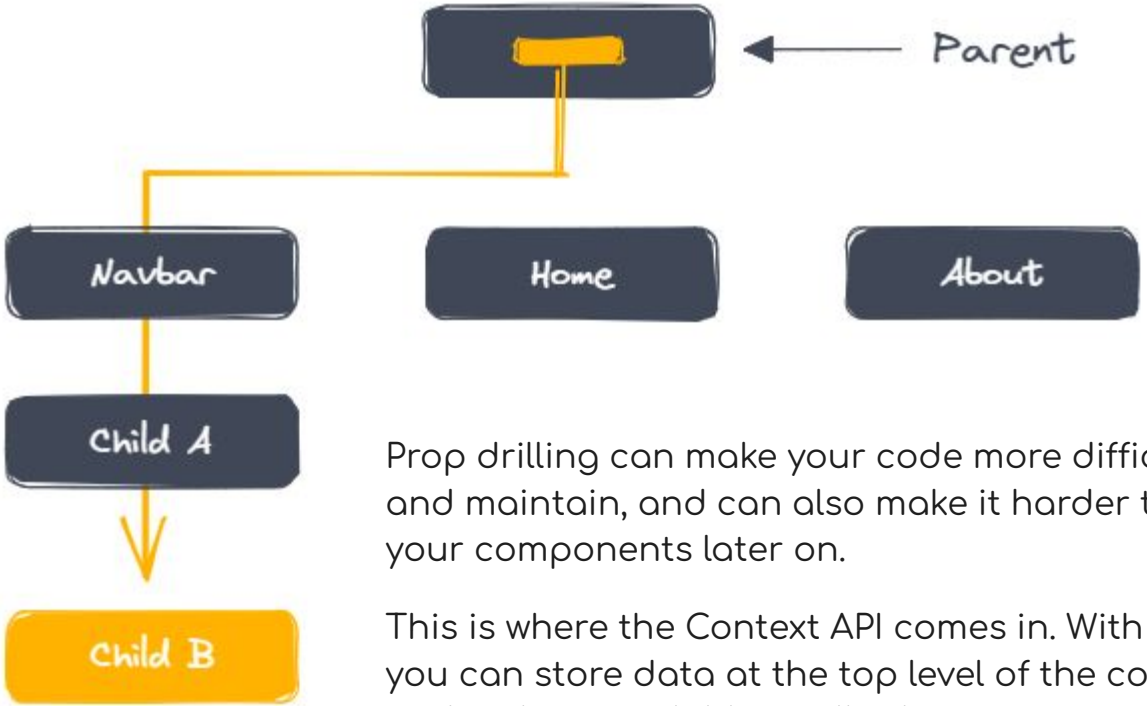
Context provides a way to pass data through the component tree without having to pass props down manually at every level.

When to Use Context

Context is designed to share data that can be considered “global” for a tree of React components, such as the current authenticated user, theme, or preferred language.

https://www.freecodecamp.org/news/context-api-in-react/

Passing Props ..



Prop drilling can make your code more difficult to read and maintain, and can also make it harder to refactor your components later on.

This is where the Context API comes in. With Context API, you can store data at the top level of the component tree and make it available to all other components that need it without passing props.

Context VS Redux

- If you are using Redux only to avoid passing props down to deeply nested components, then you could replace Redux with the Context API. It is exactly intended for this use case.

- On the other hand, if you are using Redux for everything else (handling your application's logic outside of your components, centralizing your application's state, using Redux DevTools to track when, why, and how your application's state changed, or using plugins such as Redux Form, Redux Thunk, etc...), then there is absolutely no reason for you to abandon Redux.

  The Context API doesn't provide any of this.

Context : Create context

Create context file config :

```
import React from "react";

export const ContextFeature = React.createContext();
```

Context : Use context

Wrap the components that needs to have the provider values with context provider:

```
const [value, setValue] = useState(initial value);

<Context.Provider  value={{ value, setValue}}>
        <Component />
</Context.Provider>
```

Context : Use context

useContext Hook

In child component you can use and set context value using useContext hook

const {value , setValue} = useContext(contextFeature)

Then you can use the value and its setter function to update context value

Check the following for class components with context : Link

Thank you.

Lap

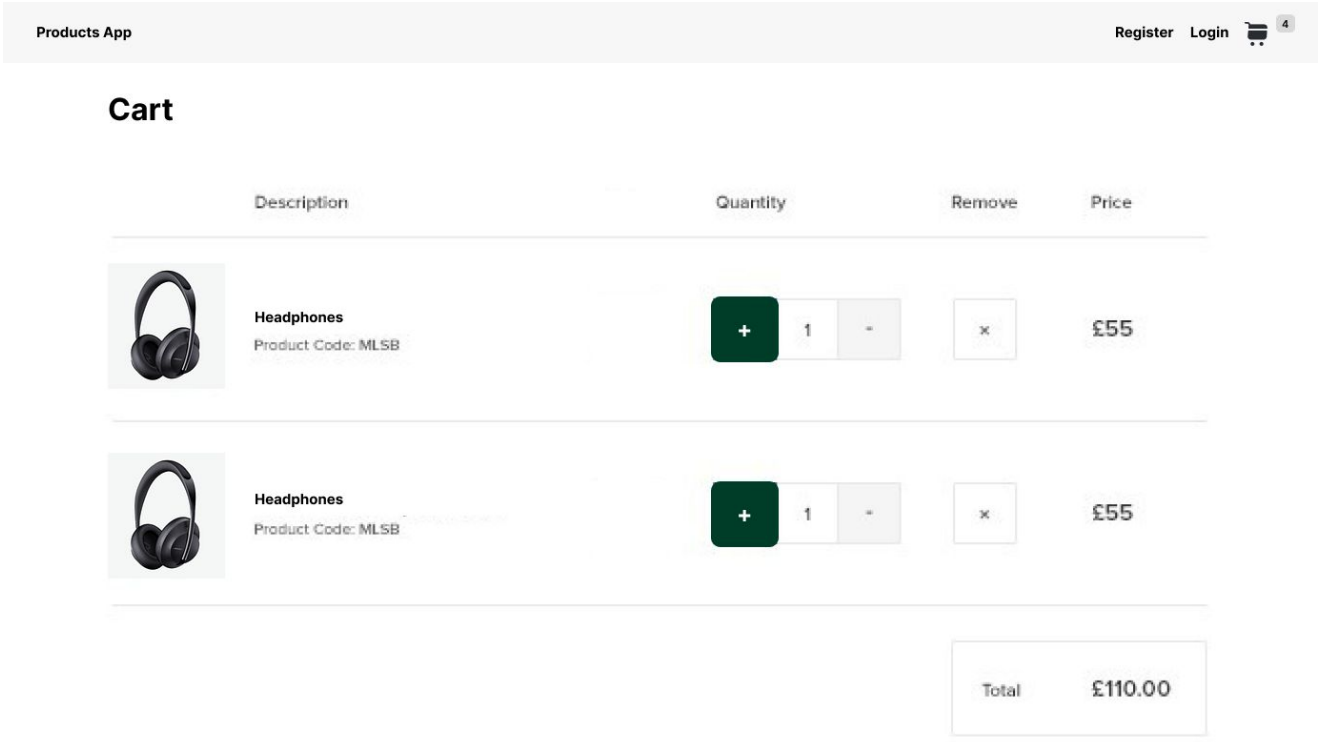
Ecommerce : Cart

Create **a redux cycle** to add products to the cart:

As a user, I would like to:

- Show count value of added items on cart icon on Navbar
- When users click on the cart icon, it shows the following:
  - If counter = 0 : show in shopping cart page (Empty cart)
  - If count > 0 : show items count in page
  - Show selected products in cart page with option to +/- item count and remove items from cart

Ecommerce : Cart



Ecommerce App

Using context:

Create a dropdown in the navbar with languages like (ar, en) and save the language value in a context. When changing the app language, you should change the app direction based on the current language.

- If lang is ‘ar’ => direction would be RTL
- If lang is ‘en’ => direction would be LTR