Redux

"Redux is a predictable state container for JavaScript apps“

It is for JavaScript apps

It is a state container

It is predictable

Redux is for JavaScript applications

Redux is not tied to React

Can be used with React, Angular, Vue or even vanilla JavaScript

Redux is a library for JavaScript applications

Redux is a state container

Redux stores the state of your application

Consider a React app - state of a component

State of an app is the state represented by all the individual components of that app

Redux will store and manage the application state

LoginFormComponent

```
state = {
  username: ‘ ’,
  password: ‘ ’,
  submitting: false
}
```

UserListComponent

```
state = {
  users: [ ]
}
```

Application

```
state = {
  isUserLoggedIn: true,
  username: ‘Vishwas’,
  profileUrl: ‘ ’,
  onlineUsers: [ ],
  isModalOpened: false
}
```

Redux is predictable

Predictable in what way?

Redux is a state container

The state of the application can change

Ex: todo list app – item (pending) → item (completed)

In redux, all state transitions are explicit and it is possible to keep track of them

The changes to your application's state become predictable

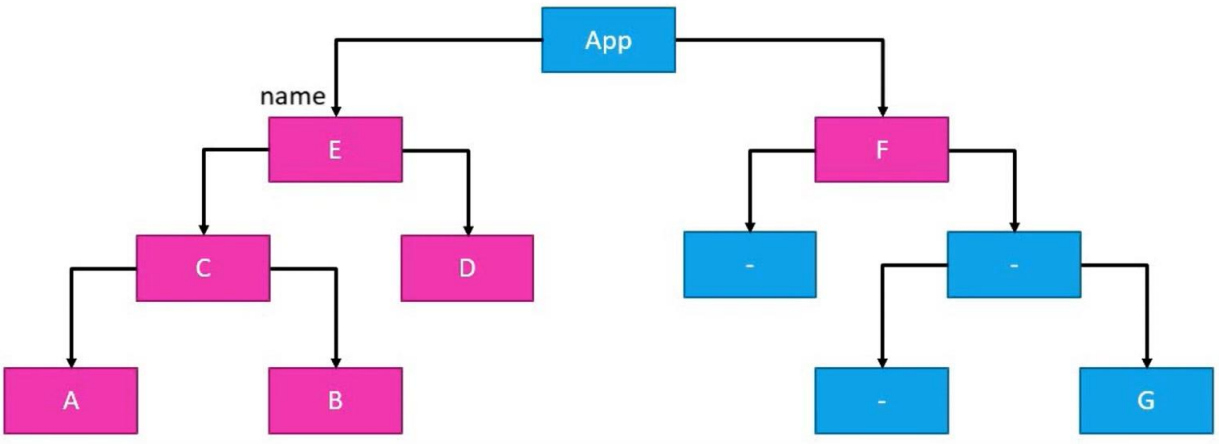
React + Redux?

Why would we want to use redux in a react application?

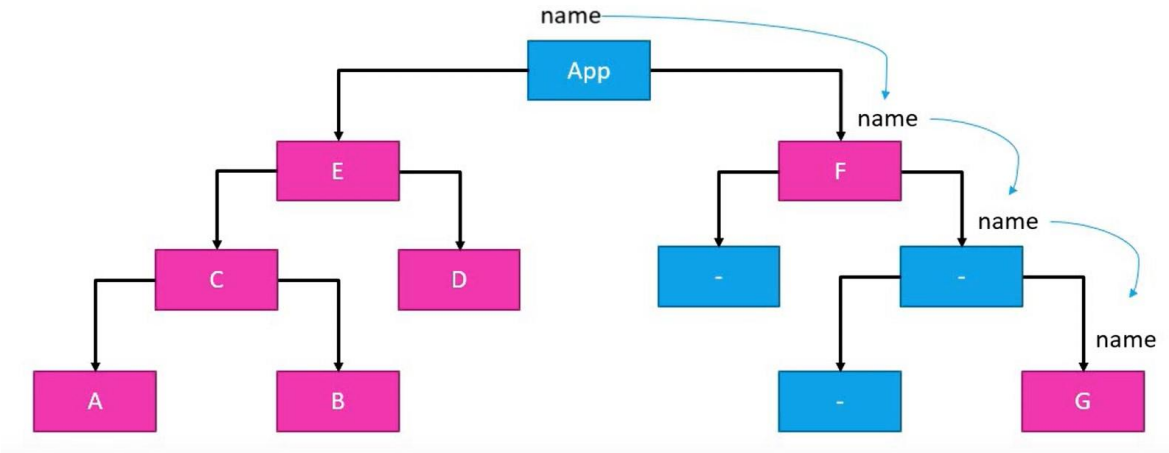
Components in React have their own state

Why do we need another tool to help manage that state?

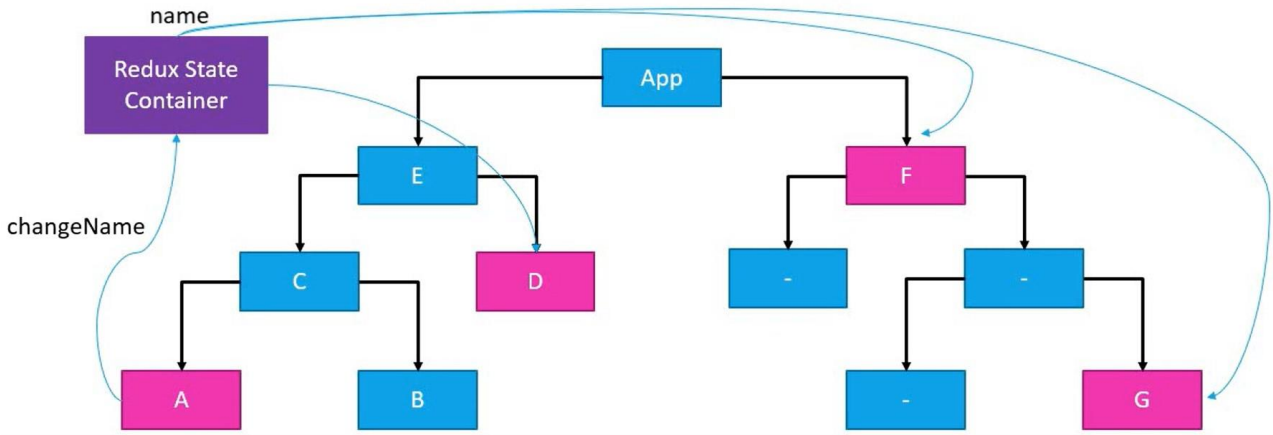
State in a React App



State in a React App



React + Redux



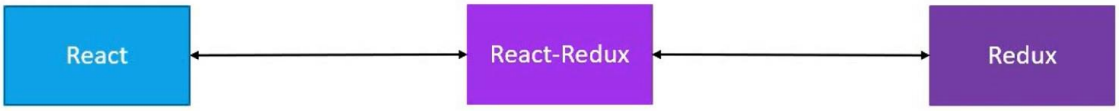
Do we really have a problem?

React context – Prevents prop drilling

useContext + useReducer ?

React-Redux

React-Redux is the official Redux UI binding library for React



Example of Redux + Reaxt (cake shop)

Three Core Concepts

Cake Shop

Entities

Shop – Stores cakes on a shelf
Shopkeeper – At the front of the store
Customer – At the store entrance

Activities

Customer – Buy a cake
Shopkeeper – Remove a cake from the shelf
– Receipt to keep track

Cake Shop Scenario	Redux	Purpose
Shop	Store	Holds the state of your application
Intention to BUY_CAKE	Action	Describes what happened
Shopkeeper	Reducer	Ties the store and actions together

A **store** that holds the state of your application.

An **action** that describes the changes in the state of the application.

A **reducer** which actually carries out the state transition depending on the action.

Three Principles

First Principle

"***The state of your whole application is stored in an object tree within a single store***"

Maintain our application state in a single object which would be managed by the Redux store

Cake Shop –

Let's assume we are tracking the number of cakes on the shelf

```
{
  numberOfCakes: 10
}
```

Second Principle

"The only way to change the state is to emit an action, an object describing what happened"

To update the state of your app, you need to let Redux know about that with an action

Not allowed to directly update the state object

Cake Shop

Let the shopkeeper know about our action – BUY_CAKE

```
{
  type: BUY_CAKE
}
```

Third Principle

"To specify how the state tree is transformed by actions, you write pure reducers"

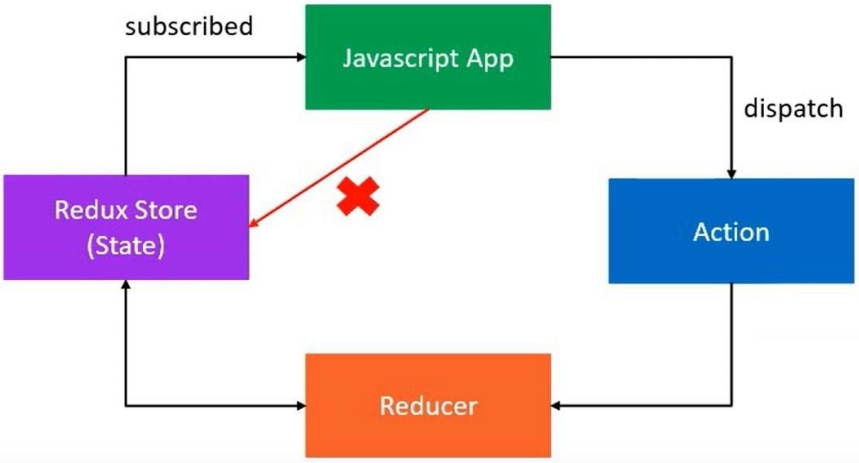
Reducer - (previousState, action) => newState

Cake Shop

Reducer is the shopkeeper

```
const reducer = (state, action) => {
  switch (action.type) {
    case BUY_CAKE: return {
      numOfCakes: state.numOfCakes - 1
    }
  }
}
```

Three Principles Overview



Actions

The only way your application can interact with the store

Carry some information from your app to the redux store

Plain JavaScript objects

Have a 'type' property that indicates the type of action being performed

The 'type' property is typically defined as string constants

Reducers

Specify how the app's state changes in response to actions sent to the store

Function that accepts state and action as arguements, and returns the next state of the application

(previousState, action) => newState

Redux Store

One store for the entire application

Responsibilities –

- Holds application state
- Allows access to state via ***getState()***
- Allows state to be updated via ***dispatch(action)***
- Registers listeners via ***subscribe(listener)***
- Handles unregistering of listeners via the function returned by subscribe(listener)

Example of Redux + Reaxt (cakes & iceCreams)

Cakes & Ice Creams!

Cake shop

Cakes stored on the shelf

Shopkeeper to handle BUY_CAKE from customer

Sell ice creams!

Ice Creams stored in the freezer

New shopkeeper to handle BUY_ICECREAM from customer

Middleware

Is the suggested way to extend Redux with custom functionality

Provides a third-party extension point between dispatching an action, and the moment it reaches the reducer

Use middleware for logging, crash reporting, performing asynchronous tasks etc

Actions

Synchronous Actions

As soon as an action was dispatched, the state was immediately updated.

If you dispatch the BUY_CAKE action, the numOfCakes was right away decremented by 1.

Same with BUY_ICECREAM action as well.

Async Actions

Asynchronous API calls to fetch data from an end point and use that data in your application.

Our Application

Fetches a list of users from an API end point and stores it in the redux store.

State ?

Actions ?

Reducer ?

State

```
state = {
  loading: true,
  data: [ ],
  error: ' '
}
```

loading - Display a loading spinner in your component
data - List of users
error – Display error to the user

Actions

FETCH_USERS_REQUEST – Fetch list of users

FETCH_USERS_SUCCESS – Fetched successfully

FETCH_USERS_FAILURE – Error fetching the data

Reducers

case: **FETCH_USERS_REQUEST**

loading: true

case: **FETCH_USERS_SUCCESS**

loading: false

users: data (from API)

case: **FETCH_USERS_FAILURE**

loading: false

error: error (from API)

Async action creators

axios

Requests to an API end point

redux-thunk

Define async action creators

Middleware

React Redux + Hooks

React Redux pattern

Action creators, reducers, provide the store and connect the components.

Components can access state and dispatch actions

React Hooks

React Redux v7.1

Subscribe to store and dispatch actions without connect( )

Actions

Synchronous Actions

As soon as an action was dispatched, the state was immediately updated.

If you dispatch the BUY_CAKE action, the numOfCakes was right away decremented by 1.

Same with BUY_ICECREAM action as well.

Async Actions

Asynchronous API calls to fetch data from an end point and use that data in your application.

Our Application

Fetches a list of users from an API end point and stores it in the redux store.

State ?

Actions ?

Reducer ?

State

```
state = {
  loading: true,
  data: [ ],
  error: ' '
 }
```

loading - Display a loading spinner in your component
data - List of users
error – Display error to the user

Actions

FETCH_USERS_REQUEST – Fetch list of users

FETCH_USERS_SUCCESS – Fetched successfully

FETCH_USERS_FAILURE – Error fetching the data

Reducers

case: **FETCH_USERS_REQUEST**
loading: true

case: **FETCH_USERS_SUCCESS**
loading: false
users: data (from API)

case: **FETCH_USERS_FAILURE**
loading: false
error: error (from API)

Summary

React is a library used to build user interfaces

Redux is a library for managing state in a predictable way in JavaScript applications

React-redux is a library that provides bindings to use React and Redux together in an application

NOTE: You can find code for above example in my github page. You can clone and use it.

Happy coding!

Link: https://github.com/mamokebe/ReactReduxPractice.git