3.1 Introduction to React JS

React (sometimes called React.js or ReactJS) is a JavaScript library for building a fast and interactive user interface.

It was originated at Facebook in 2011 and allow developers to create sizeable web applications or complex UIs by integrating a small and isolated snippet of code.

In some quarters, React is often called a framework because of its behaviour and capabilities. But technically, it is a library.

Unlike some other frameworks like Angular or Vue, you'll often need to use more libraries with React to form any solution.

3.1.1 Why Use React - Virtual DOM

The two main features that make React more than just a library are JSX and Virtual DOMs.

JSX or JavaScript extension combines HTML syntax with JavaScript making it easier for developers to interact with the browser.(it could be transpiled via Babel)

Virtual DOM is a virtual copy of the DOM tree created by web browsers that React creates for simplifying the process of keeping track of updates in real-time.

Whenever a change occurs in the HTML code, either by user interaction or value updates, the DOM tree has to be rendered again, leading to a lot of time and power consumption. ReactJS comes to our rescue here by use of Virtual DOMs.

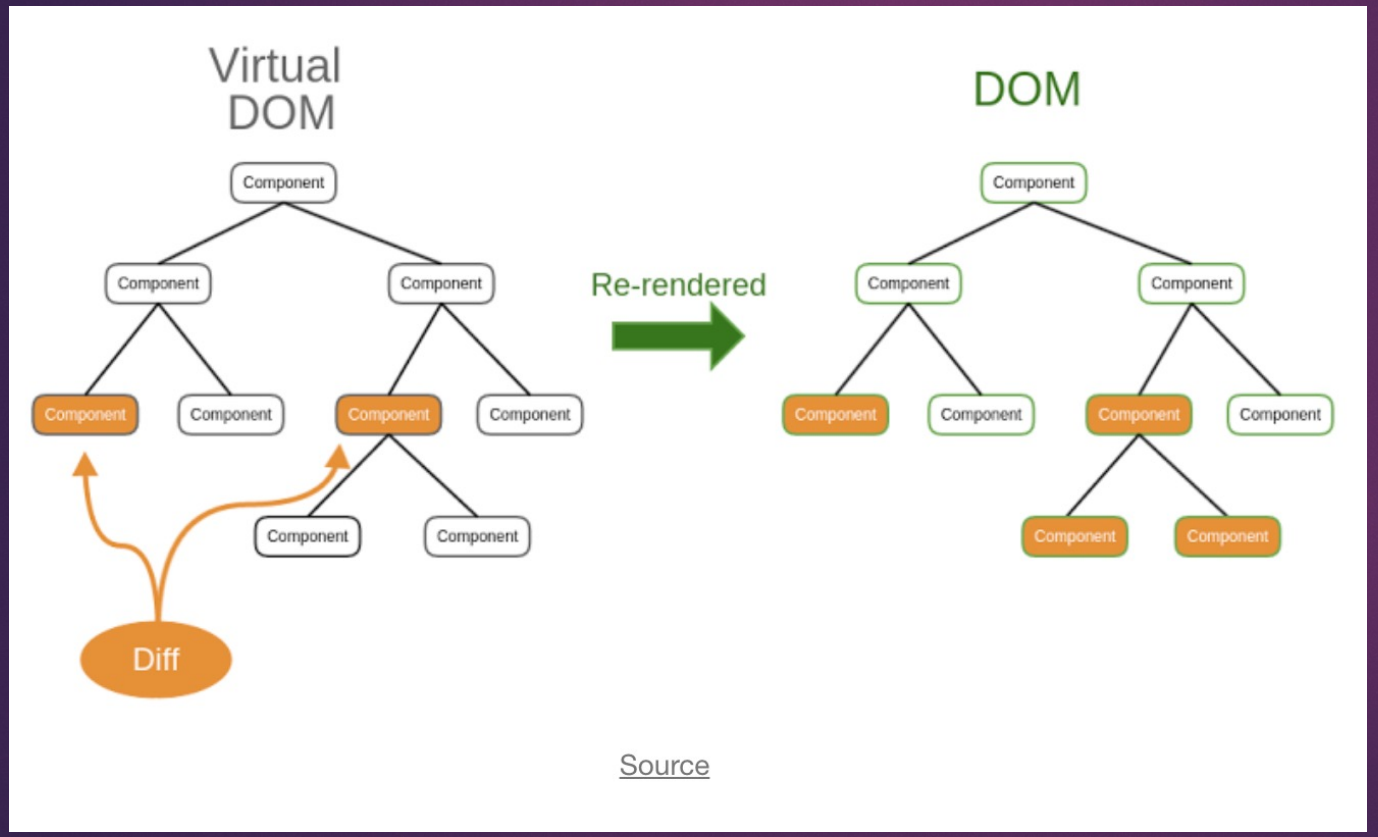
3.1.2 Why Use React - Virtual DOM 2

Updating the virtual DOM is comparatively faster than updating the actual DOM (via js).

Thus, the framework is free to make unnecessary changes to the virtual DOM relatively cheaply.

The framework then finds the differences between the previous virtual DOM and the current one, and only makes the necessary changes to the actual DOM.

3.1.3 Why Use React - Virtual DOM 3



Source

Thanks to virtual DOM, we could rerender the small part of the DOM tree.

3.2 Why Use React - JSX

JSX or JavaScript XML is a syntax extension for JavaScript. Facebook developed it to extend the functionalities of HTML structures into JavaScript. With JSX, there is no requirement for separate HTML and JS codes.

You can use the declarative HTML syntax directly in JavaScript code with ReactJS. Browsers decode HTML documents to display the UI.

//const oneDiv = React is Awesome !

They do so by creating DOM trees, and JavaScript enables us to modify this DOM to create interactive UI. JSX increases the efficiency of manipulating DOMs by manifolds.

3.2.1 Why Use React - Unidirectional Data Flow

In React, information inherited from a parent component is called props. Props are immutable objects whose values can not be modified by the child components due to the unidirectional flow of data in React.

The downward directional binding makes the code stable and consistent as any child components' changes won't affect the siblings or parent components. To modify an object, you just have to update the state. ReactJS will automatically alter the valid details to maintain consistency.

Debugging and error-checking are much more efficient in React due to the unidirectional data binding providing higher control over data access of components.

3.2.2 Why Use React – Other Benefits

- React Hooks (version 16.8)
- React Native
- Redux
- Smooth Learning Curve
- Single Page Application(SPA)….

3.3 Setting up Working Environment

If you have installed NodeJS before

You can use the following command in your terminal:
npx create-react-app my-app

And then you can change your directory to my-app
cd my-app

Or you can use React online playground: codesandbox.io

3.3.1 My first React App

We could build our first react app like this

const App = Hello World !

3.3.1 Working with data

Understanding how the data flows is very crucial to building React component. That brings us to the concept of state and props.

The props (which stands for properties) is one of the two types of "model" data in React. It can be thought of as the attributes in the HTML element. For instance, the attributes – type, checked – in the input tag below are props.

```
<input type="checkbox" checked={true} />
```

Data that is received in the child component becomes read-only and cannot be changed by the child component. This is because the data is owned by the parent component and can only be changed by the same parent component.

3.4 Working with data

Unlike the props, the state data is local and specific to the component that owns it. It is not accessible to any other components unless the owner component chooses to pass it down as props to its child component(s).

There are 2 approach to create local states:

1. Use constructor of a class-based component
2. Use react hook(useState) to create the local state

3.4.1 Class component and function component

Functional Components	Class Components
A functional component is just a plain JavaScript function that accepts props as an argument and returns a React element.	A class component requires you to extend from React. Component and create a render function which returns a React element.
There is no render method used in functional components.	It must have the render() method returning HTML
Also known as Stateless components as they simply accept data and display them in some form, that they are mainly responsible for rendering UI.	Also known as Stateful components because they implement logic and state.
React lifecycle methods (for example, componentDidMount) cannot be used in functional components.	React lifecycle methods can be used inside class components (for example, componentDidMount).

3.4.1 React Component Life cycle

- **Mounting – Birth of your component**
- **Update – Growth of your component**
- **Unmount – Death of your component**

- Mounting – class based component

1. constructor()
2. static getDerivedStateFromProps()
3. render()
4. componentDidMount() // => useEffect

3.4.1 React Component Life cycle

An update can be caused by changes to props or state. These methods are called in the following order when a component is being re-rendered:

- Updating

1. static getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()
5. componentDidUpdate()

3.4.1 React Component Life cycle

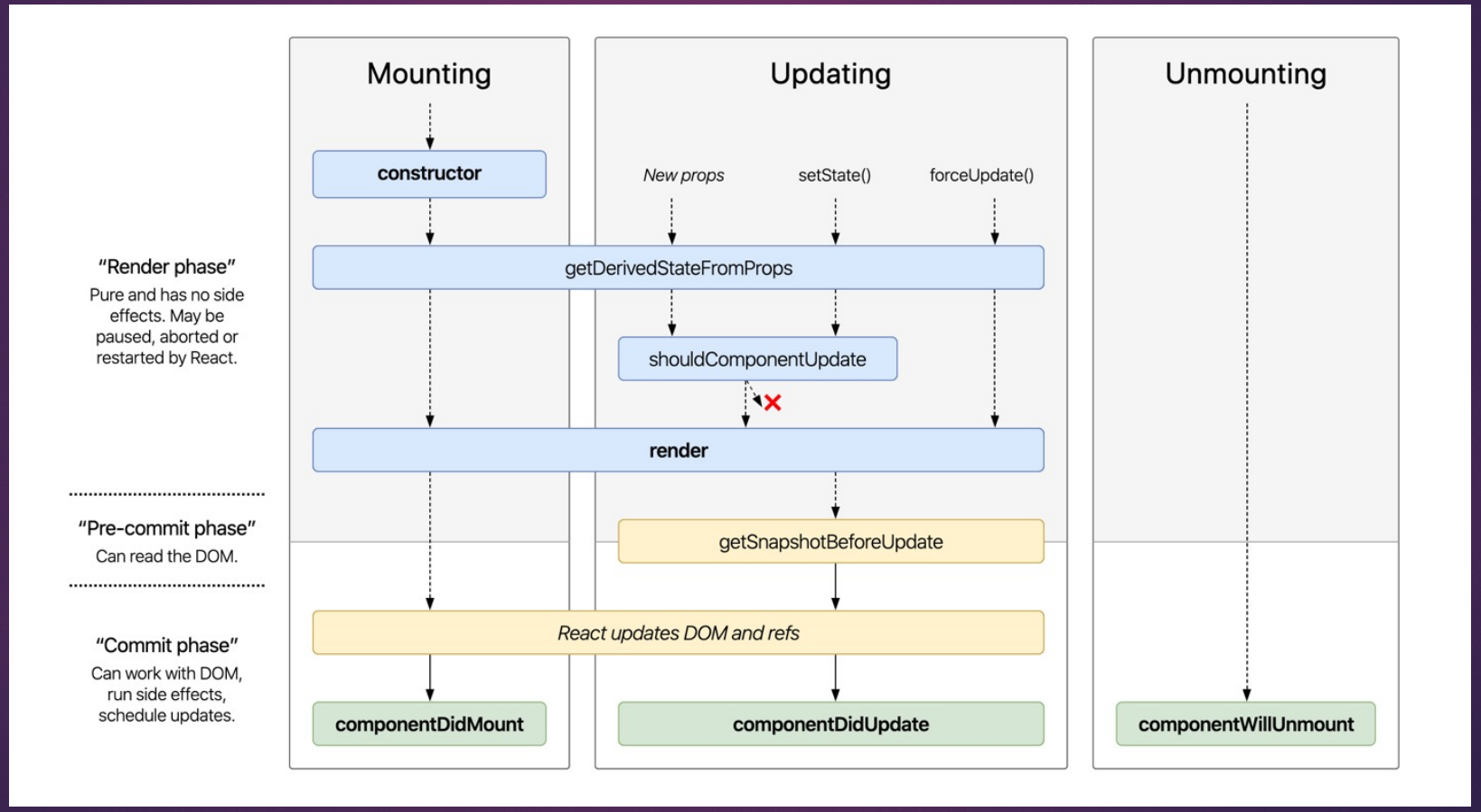
This method is called when a component is being removed from the DOM:

- Unmounting

1. componentWillUnmount()

http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

3.4.2 React Life cycle



3.5 How to design a todo app in React

1. Consider how to set up the state and the format of state
2. Consider the communication between the different Components
3. Design the todo App structure

What are the benefits from implementing a todo app?

Because all the code challenges are the variation of a todo app.

3.5.1 Thinking in React Component

When building an application with React, you build a bunch of independent, isolated and reusable components.

Think of component as a simple function that you can call with some input and they render some output.

And as you can reuse functions, so also you can reuse components, merge them and thereby creating a complex user interface.

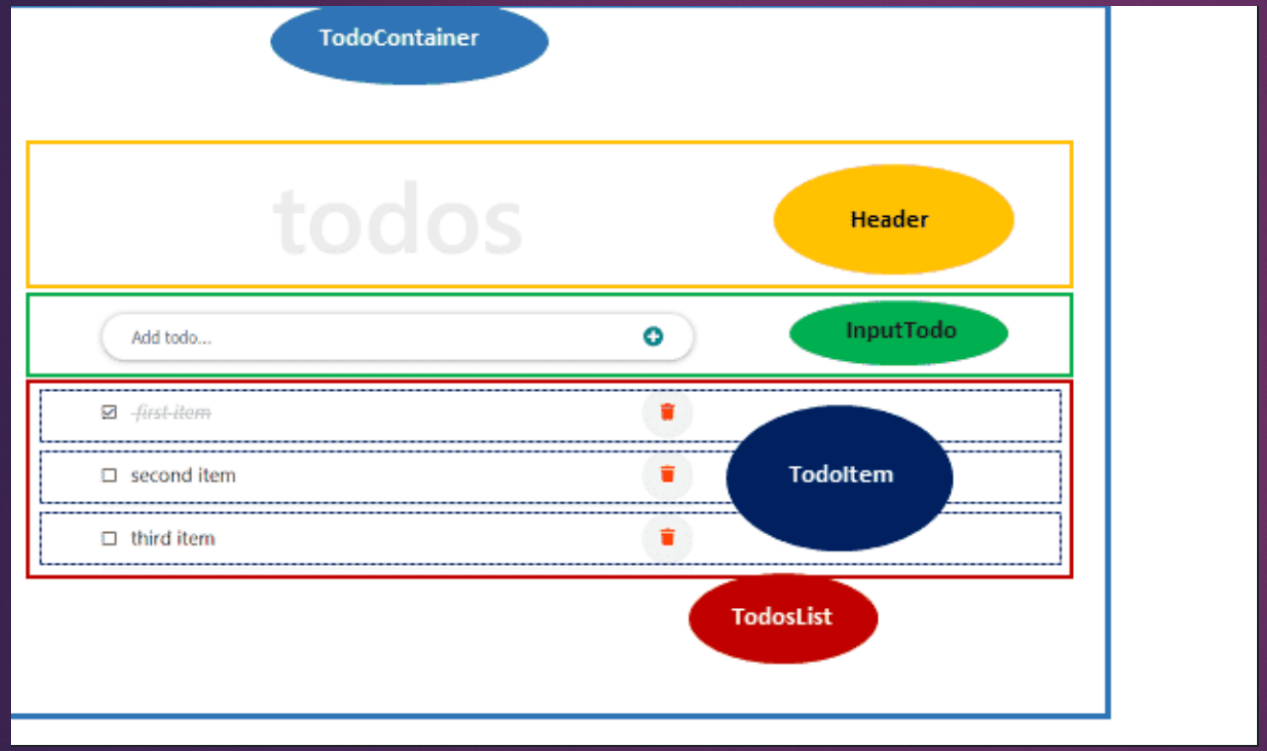
3.5.2 Thinking in React Component 2

To build this type of React app or any complex app (even as complex as Twitter), the very first thing to do is to split and decompose the UI design into a smaller and isolated unit as outlined in the image.

Where each of these units can be represented as a component which we can build in isolation and then later merge to form a complete UI.

Question: How to break down the todo App ?

3.5.3 Example : Break down the todos App



1. Todo Cotainer
   1. Header
   2. InputTodo
   3. TodoList
      1. TodoItem

3.5.4 Example : Break down the todos App

- The parent component (also known as the root component), label TodoContainer, holds all the other components (known as children components).

- The Header component renders the header contents
- The InputTodo component accepts the user's input.

- the TodosList component renders the todos list, the TodoItem component takes care of each of the todos items

- If you want, you can further decompose the TodoItem into smaller components: textContent and delete todo button

3.6 React Hooks

- In React(before version 16.8.0), class components are stateful and smart, being state is attached to it and kept persistent through renders. Then functional components were stateless and dumb components, having nor state nor logic attached to it.
- From React's version 16.8.0, Hooks are introduced to make functional components more useful.
- React Hooks enable functional components to attach local state with it, and React will preserve this state between re-renders. Thereby this will allow React to be used without classes.

3.6.1 React Hooks – Hooks Rules

1. Don’t call inside loops
2. Don’t call inside conditions
3. Don’t call inside nested function
4. Always use hook at top of function
5. Only call hooks from react function

3.6.2 React Hooks – Hooks list

- There are a lot of React hooks, but you should use some of them in the real interviews only.

Basic Hooks
- useState
- useEffect
- useContext

Additional Hooks
- useReducer
- useCallback
- useMemo
- useRef
- useImperativeHandle
- useLayoutEffect
- useDebugValue

3.6.3 React Hooks - State Hook

```
Count  => this.state = { count: 0 }  setCount => this.setState({count: 10})

const  Example = () => {
      const [count, setCount] = useState(0);
      return (
            <div>
                  <p>You clicked {count} times</p>
                  <button onClick={() => setCount(count + 1)}> Add </button>
                  <button onClick={() => setCount(count - 1)}> Remove </button>
            </div>
      );
}
```

3.6.4 React Hooks - Effect Hook

Effect hook here, is used to perform side effects in functional components.

From familiar user cases, it could be considered combination of methods: componentDidMount, componentDidUpdate, and componentWillUnmount.

So when React renders (updates the DOM including first render) component, this hook becomes applied. Typical use cases for effect hooks are data fetching, managing subscriptions and dealing with manual DOM changes.

3.6.5 React Hooks - Effect Hook 2

- DOM change but for the initial page load
 => componentDidMount

```
useEffect(() => {
      document.title = `You clicked ${count} times`;
}, []);
```

- This effect hook acts for each render, so can replace lifecycle methods such as componentDidMount & componentDidUpdate

```
useEffect(() => {
      document.title = `You clicked ${count} times`;
}, [count]);
```

3.6.6 React Hooks - Effect Hook 3

- Data fetching

```
useEffect(() => {
      const async fetchData() {
            const response = await fetchMockAPI();
            setData({...data, arr: response});
      }
      fetchData();
}, []);
```

3.6.7 React Hooks - Effect Hook 4

In hooks it's called cleanup, and if function is returned from effect hook (optional) it'll be used for cleanup when unmounting this functional component.

```
const [timer, setTimer] = useState(10);
useEffect(() => {
      if (timer === 0) {
            return;
      }
      const newTimer = setInterval(() => {
            setTimer(timer - 1);
      }, 1000);

      return () => clearInterval(newTimer);
}, [timer]);
```

3.6.9 React Hooks - Ref Hook

Reference hook can be used to refer React element created by render method. When there are DOM changes, ref's .current value will be up to date.

```
const inputElement = useRef(null);
const onButtonClick = () => {
      inputElement.current.focus();
};

<input ref={inputElement} type="text" />
```

3.6.10 React Hooks - useMemo hook

Pass a “create” function and an array of dependencies. useMemo will only recompute the memoized value when one of the dependencies has changed. This optimization helps to avoid expensive calculations on every render.

We could take the search box App as an example

https://codesandbox.io/s/use-memo-test-nwpzw?file=/src/App.js:767-775

3.6.11 Todo App Implementation

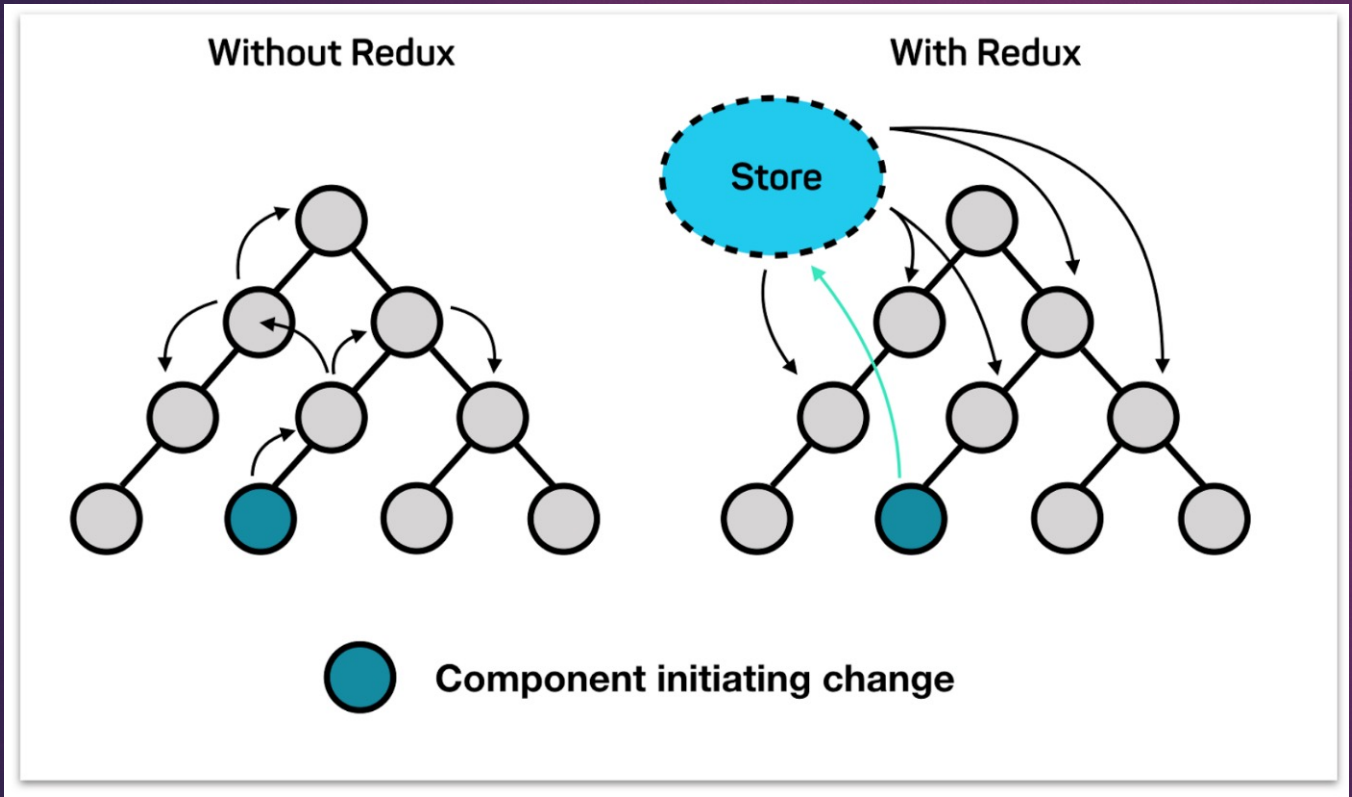
Let's create a Todo App via React

3.7 Introduction to Redux

Now that we have a basic understanding of React, props, and state — let's do a shallow dive into Redux. One of the most fundamental benefits of using Redux is to help manage state for larger apps.

Instead of passing around all of your state/props of components, the state is stored in a separate location — aptly named the store.

3.7.1 Introduction to Redux 2



The picture to illustrate the basic Principles of redux.

3.7.2 Introduction to Redux 3

This is a visual representation of passing state/props with React vs Redux.

With Redux and the state being held within the store, all of the components can make requests to the store and access the state without having to go through parent components.

With Redux comes two new functionalities — actions and reducers.

3.7.3 Introduction to Redux 4

Actions can be considered payloads that send information from your application to the store. This is the only way the store can be updated — via actions.

Actions are an object and require a type as well as the information it is delivering.

```
export const addTodo = (dispatch) => (value) => {
      dispatch({
            type: "ADD",
            payload: {
                  todoContent: value,
                  isCompleted: false
            }
      });
};
```

3.7.3 Introduction to Redux 5

A reducer is at its foundation a function that accepts the current state as well as the action to return the new state

Reducers are functions that actually return the new, updated state to the store.

```
export const reducer = (state = [], { type, payload }) => {
        switch (type) {
                case "ADD":
                        return [...state, { ...payload }];
        }
};
```

Dispatch action from a component -> reducer received the action and update the state -> components which connect to the store will update

3.7.3 Introduction to Redux 6

A real example for implement a todo via redux

1. Create the store first
2. Implement the action creator
3. Implement the reducer

FYI: you can use the react-redux hook in your code(useDispatch, and useSelector)
useDispatch
This hook returns a reference to the dispatch function from the Redux store. You may use it to dispatch actions as needed

useSelector
Allows you to extract data from the Redux store state, using a selector function.

3.8 Introduction to Redux Thunk

What is a "thunk"?

The word "thunk" is a programming term that means "a piece of code that does some delayed work".

Rather than execute some logic now, we can write a function body or code that can be used to perform the work later.

3.8.1 Introduction to Redux Thunk

Why Use Thunks ?

Could we dispatch an async action to reducer ?

We said earlier that reducers must always follow some special rules

- They should only calculate the new state value based on the state and action arguments
- They are not allowed to modify the existing state. Instead, they must make immutable updates, by copying the existing state and making changes to the copied values.
- They must not do any asynchronous logic or other "side effects"

3.8.2 Introduction to Redux Thunk

Side Effect:

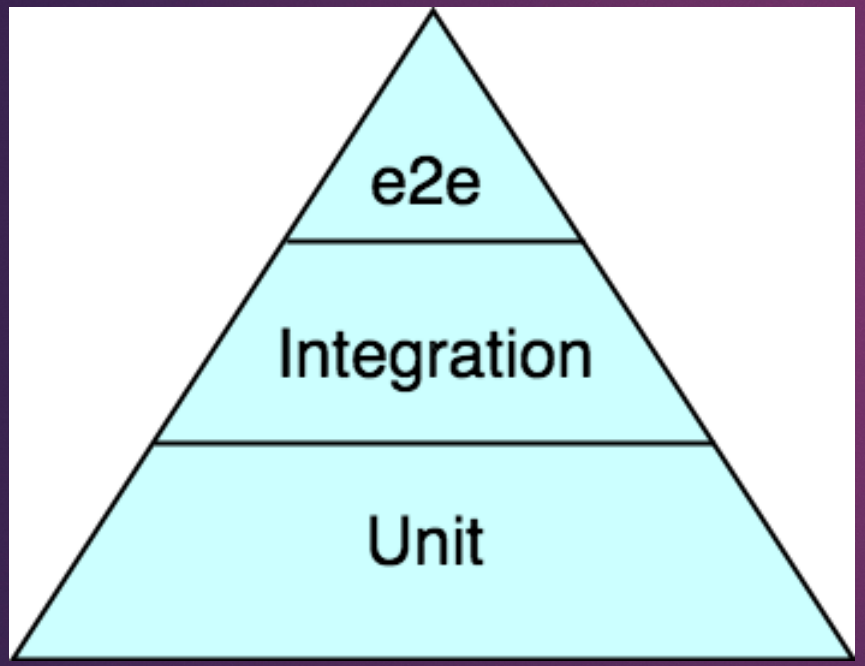
- Logging a value to the console
- Saving a file
- Setting an async timer
- Making an AJAX HTTP request
- Modifying some state that exists outside of a function, or mutating arguments to a function
- Generating random numbers or unique random IDs (such as Math.random() or Date.now())

3.8.3 Introduction to Redux Thunk

How to sync up data with the backend ?

Let's modify our todo App via Redux Thunk.

3.9 Introduction to Test



- E2E
- Integration
- Unit

3.9.1 Introduction to Test

1. Unit test

Simple JS unit testing will take a function, monitor output and return its behavior.

JavaScript Unit Testing is a testing method in which JavaScript test code written for a web page or web application module is combined with HTML as an inline event handler and executed in the browser to test if all functionalities work fine

Test runner: Jest

3.9.2 Introduction to Test

2. Integration test

- By definition, integration tests test the interactions between the various components of an application. For a React application, this means testing
- interactions between React components, typically performed via calling prop functions such as <Component onClick={onClickHandler}>
- manipulation of component state
- direct manipulation of the DOM in React lifecycle methods

Airbnb's Enzyme JS and react-testing-library.

3.9.3 Introduction to Test

3. E2E(End to End) Test

End-to-end testing is a technique that tests the entire software product from beginning to end to ensure the application flow behaves as expected. It defines the product's system dependencies and ensures all integrated pieces work together as expected.

3.9.4 Introduction to Test

4. Automation Test

Automated tests are programs that automate the task of testing your software, and they matter because they give developers quick feedback on errors. They interface with applications to perform actions and compare the actual result with the expected output you've previously defined. Thus, automated tests enable fast error feedback on the code developed, making the development process more reliable. In other words, developers can see errors quickly and fix them before software goes to production.

Recommended Framework: Nightwatch
https://nightwatchjs.org/

3.9.5 Introduction to Test

Let's look at some unit test examples in our todo App

3.10 Summary

1. React component life cycle(concept)
2. React Hooks(code challenge)
3. How to communication with backend(code challenge)
4. Redux(concept)
5. Practice implementing some Apps, Todos, search bar, calendar (within 40 minutes)