FWP
Semester 2, 2022

Week 03

useState, useEffect hooks

Conditional rendering

Forms and

Incorporating data

RMIT UNIVERSITY

Before we start

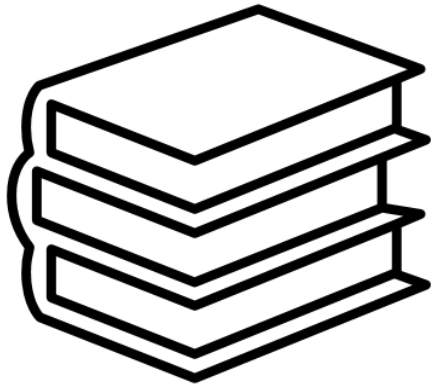
Lectorial

Tutorial/Lab

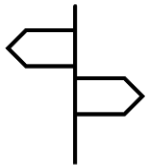
Assignments

Expectations

Introduction



In week 2 we covered



- Components- functional
- Interaction between components and *props*
- Introduction to event handling
- useState hook
- Styling in React apps

Segment 1

useState hook *again*

useEffect hook

More on components:

- **conditional rendering,**
- **form handling**
- **router**

So what is a *hook*?

- **What is a Hook?**
  - A *Hook* is a special function that lets you “hook into” React features. For example, *useState* is a Hook that lets you add React state to function components.
- **When would I use a Hook?**
  - If you write a function component and realize you need to add some state to it, previously you had to convert it to a class. Now you can use a Hook inside the existing function component.

Reference:

https://reactjs.org/docs/hooks-reference.html

State

- ***State: helps a component to become dynamic and interactive***
- **It determines how a component will render and behave**
- **While *props* are used to pass information down the component tree, *state* is used to alter information over time**
  - **Do you remember *props* are readonly!**

State

- ***Think of this scenario***
  - **Whenever a user types something into an HTML input field, the user may want to see this typed information (state) displayed somewhere else in the application.**
  - **Therefore we need some way to change information over time and, what's more important, to notify React to re-render its component(s) again.**
- ***This is where state comes into the picture!!***

Do you remember useState() *hook?*

1. **Call useState() hook to enable state in a functional component.**

2. **The first argument of the useState(initialValue) is the state's initial value.**

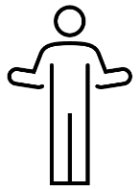
3. **[state, setState] = useState(initialValue) returns an array of 2 items: the state value and a state updater function.**

Do you remember useState() *hook?*

4. **Invoking the state updater function setState(newState) with the new value updates the state.**
   - **Alternatively, you can invoke the state updater with a callback setState(prev => next), which returns the new state based on previous.**

5. **After the state updater is called, React makes sure to re-render the component so that the new state becomes actual.**

useState *hook- when is the state updated?*



- Look at this code snippet

```
const MyComp = () => {
  const [counter, setCounter] = useState(0);
  onClick = () => setCounter(prev => prev + 1);
  return <button onClick={onClick}>Click me</button>
}
```

- Every hook has an update queue.
  - When you call the setCounter function, React doesn't call the updater function immediately, but saves it inside the *queue*, and schedules a re-render.
- The above code is simple but in reality a bigger web application may not be!

useState *hook: when is state updated?*

- **There might be more updates in the code, to this hook *(shown on previous slide*), other hooks, or even hooks in other components in the tree.**
- **All of these updates are queued - nothing is computed immediately.**
- **Finally, React re-renders all components that were scheduled to be rendered, top-down. But the state updates are still not performed.**
- **It's only when useState actually runs, during the render function, that React runs each action in the queue, updates the final state, and returns it back.**
- **This is called *lazy computation* - React will calculate the new state only when it actually needs it.**

useState *hook: when is state updated?*

- So when does React say: "OK, enough queueing, let me do something"?
- Whenever there's an event handler (onClick, onKeyPress, etc.) React runs the provided callback inside a *batch*.
- The batch is synchronous, it runs the callback, and then flushes all the renders that were scheduled

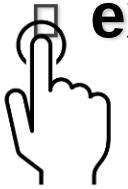
```
const MyComp = () => {
  const [counter, setCounter] = useState(0);

  onClick = () => { // batch starts
    setCounter(prev => prev + 1); // schedule render
    setCounter(prev => prev + 1); // schedule render
  } // only here the render will run
  return <button onClick={onClick}>Click me</button>
}
```

Lectorial Exercise

- if React computes the new state during render time,
  - how can it bail out of render if the state didn't change?

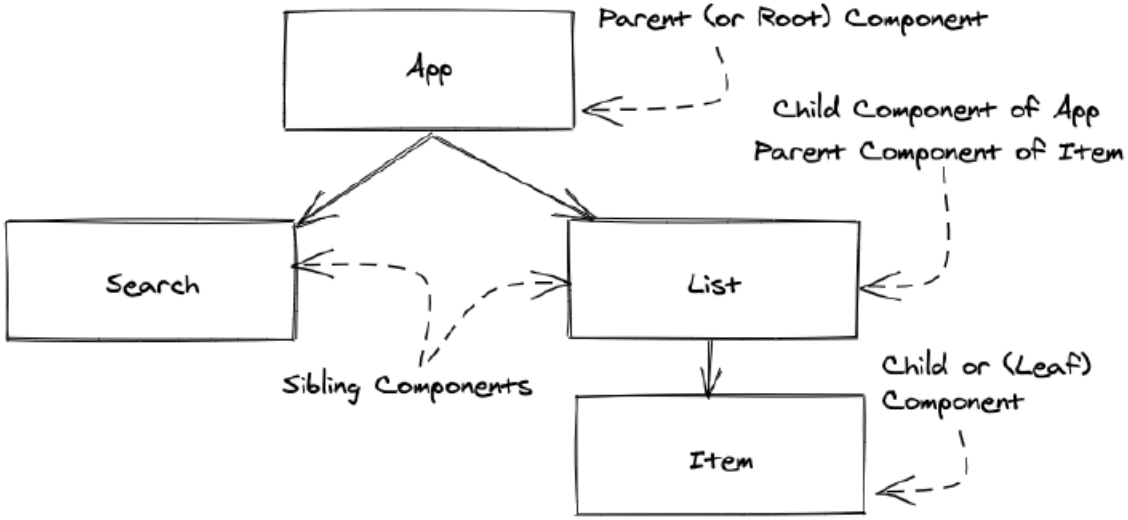
useState is very flexible

- Multiple state variables may be used and updated from within a functional component
- As opposed to strings and numbers, you could also use an object as the initial value passed to useState.
- As opposed to just passing an initial state value, state could also be initialised from a function
- Time to look at a code example
- example1 and example2



Lectorial Exercise

- **Complete the event handlers for example 3 from Lectorial 1.**



A side note: → Redux!

- ***There are external libraries available to impart state in React apps***
- **Redux is one of them!**
- **But this is what Facebook says (https://reactjs.org/docs/faq-state.html)-**

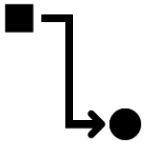
> **Should I use a state management library like Redux or MobX?**
>
> Maybe.
>
> It's a good idea to get to know React first, before adding in additional libraries. You can build quite complex applications using only React.

- **A good reason to redux would be when web application is very large and state management is complex**

useEffect *hook*

- **The Effect Hook lets you perform side effects in function components.**
- ***Side-effects 'eh!!. .do I have to learn something new?***
  - *"There won't be much to learn. In fact, we'll spend most of our time unlearning."* – *Dan Abramov*
- **What are effects?**
- **Examples**
  - **Fetching data**
  - **Reading from local storage Registering and deregistering event listeners**
  - **Manually changing DOM elements …**

Effects



- Effects are always executed after render
- Hooks documentation by Facebook states-
  - Mutations, subscriptions, timers, logging, and other side effects are not allowed inside the main body of a function component (referred to as React's render phase).
  - Doing so will lead to confusing bugs and inconsistencies in the UI.
  - Instead, use *useEffect*.
    - The function passed to useEffect will run after the render is committed to the screen.
- To opt out or skip effects, you have to understand basic JavaScript concepts about values

useEffect *hook*

- This hook accepts a function that contains imperative, possibly effectful code.
  - useEffect(didUpdate);
- Remembering class lifecycle methods, you can think of useEffect Hook as
  - componentDidMount, componentDidUpdate, and componentWillUnmount *combined*.
- Often, effects create resources that need to be cleaned up before the component leaves the screen, such as a subscription or timer ID.
  - To do this, the function passed to useEffect may return a clean-up function

useEffect *hook*

- To understand this hook fully, one needs to explore these scenarios:
  - Running side effects after every render
  - Running an effect only when a component mounts
  - Cleaning up side-effects by returning a function
  - Controlling when an effect runs by specifying dependencies
- Time to look at some of above via examples of this hook
- example4 and example5

Segment 2

More on components:

conditional rendering,

form handling

router

Data storage

Conditional rendering

- You can create distinct components that encapsulate behaviour you need.
  - Then, you can render only some of them, depending on the state of your application.
- Conditional rendering in React works the same way conditions work in JavaScript.
  - Use JavaScript operators like if or the conditional operator to create elements representing the current state, and let React update the UI to match them.
- It is a very handy concept when you create a web app

Conditional rendering- *various ways*

- if…else statement
- switch statement
- if...else with element variable
- Ternary operator
- Logical && operator

Conditional rendering

- Time to look at examples
- We will borrow on our understanding of event handling from week 2 lectorial
- example6 and example7

Form handling

- ***Form handling is slightly different in React***
- **You need to know the concept of *controlled components***
- **In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input.**
- **In React, we will handle forms with *useState* hook**
- **An input form element whose value is controlled by React is called a "controlled component".**

example8

Self-exercise

- **Why not style the form using react bootstrap module**
- ***Start here:***
  - ***https://react-bootstrap.github.io/forms/overview/***

Router

- Routers help create and navigate between the different URLs that make up your web application.
- They allow your user to move between the components of your app i.e. ***redirect to another page/component***
- React Router library is build a declarative routing system for your application
- Install as
  - npm install react-router-dom
- It installs version 6.0
- Official page:
  - https://reactrouter.com/docs/en/v6/getting-started/overview

Router

- React-Router library
  - https://reactrouter.com/docs/en/v6/getting-started/overview
- Time for an example
- example9

- We will look at advanced routing next week

Data storage

- In any web application, you will need to store data one way or the other
- Examples- store user details, product preferences, items bought, etc.
- The data storage may be temporary or persistent
- We will learn the data storage via the databases in the latter half of this course where we will use MySQL database *(more for assignment 2)*.
- But before that, there are other ways of storing data
  - Data structures in JS
  - HTML5's localStorage object

  You will need these for assignment 1

Data storage with data structures

- Time to look at an example – here we will use a data structure and localStorage to store the data.

example10

  - Simple login-logout system
  - It will also be covered in this week's lab

exampl10- *crib notes*

- Uses some concepts that will be covered in week 03
  - Forms in React
  - Conditional rendering of components
  - React Router dom:
    - npm install react-router-dom
  - React router has been used to control which component is shown based on the current page, with location and navigation support
  - The router, switch and route paths can be found in App.js. The links using these paths can be found in Navbar.js.

Example10- *crib notes*

- The logged in username state is stored in the App parent component and is passed down to components that reference this state as props / properties.
- To modify the state functions implemented within the App component called loginUser and logoutUser are implemented. These functions are passed as props / properties to the Navbar and Login components; the Navbar component uses logoutUser and Login component uses loginUser.
- Lastly there is some conditional rendering is included in the Home and Navbar components to render different output if the user is logged in or not.
- The user data can be found in src/data/repository.js

Assignment 1 *discussion*

- It is online
- Deadline: *Check Canvas*
- Worth 25%
- To be completed individually or in a group of 2
- Partial React website prototype
- Based on week(s) 1-4
- DO NOT WRITE CLASS COMPONENTS- you will get a ZERO for the whole assignment.
- Discussion..

Assignment 1 *DEMO*

- **COMPULSORY! ➔ NO DEMO means NO MARKS**
- **A 20 minutes session where you run the website on your machine and show it to the marker**
  - **You bring student card!**
  - **Face to face at the marking venue**
  - **Via Teams with <u>camera on all the time</u>**
- **Marker will ask you questions regarding your code, website etc**
- **You will be marked on the website on the spot**
  - **Marker will check code in free time and finalise marks**

Next week

- More on React components
  - Components interacting
- useEffect hook,
- custom hooks,
- Routing again and
- form handling yet again