*-Lecture 9-*

# Chapter 6–**React Hooks React Hook Forms**

Adil **CHEKATI**, PhD

adil.chekati@univ-constantine2.dz

# Prerequisites

- ❑ Basic Understanding of React Fundamentals (Knowledge)
- ❑ Understanding of Node.js and npm (Application)

# React Hooks & React Hook Forms

## Objectives

- ➔ Define what React Hooks are and their role in functional components.
- ➔ Understand how Hooks can simplify state and context management in React applications.
- ➔ Manage Form data and handle user input effectively.

# Introduction

```
import React, { Component } from 'react'
export default class App extends Component {
  render() {
    return (
      <div>
        <h1>Hello World</h1>
      </div>
    )
  }
}
```

JSX

```
import React from 'react'
const App = () => {
  return (
    <div>
      <h1>Hello World</h1>
    </div>
  )
}
export default App;
```

**Class component** | **Functional component**

Before version 16.8.0, React components were classes. From version 16.8.0, components became functions (the latest is version 18.2.0 of June 2022).

# Introduction

A component is a JavaScript function with the following characteristics:

- Its name must begin with a capital letter.
- It can contain other functions.
- It must return a value (generally in the form of JSX).

# Introduction

## A Typical Function Component in React

React function component.

Which is quite similar to typical JavaScript functions. The only difference is here we are returning HTML code becuase this is JSX(JavaScript XML)

```
import React from "react";

function componenetName() {
    return (
        <h1>I am a function component</h1>
    )
}

export default componentName;
```

Way of telling Babel that we are working with JSX files so that it will not throw error while tranforming code into the React.createElement calls

Exporting the function (component) so that we can use it outside

# Introduction

## Switching from a class component to a function component

- ➔ Class components are well suited to describing both statements and states.
- ➔ Function components can have props as parameters, but they cannot describe states. This is one of the reasons why hooks were created.
- ➔ There are several types of hooks in React: `useState`, `useEffect`, `useContext`, `useRef`, `useReducer`, `useMemo`, `useCallback`.
- ➔ You can also create custom hooks to suit your needs.

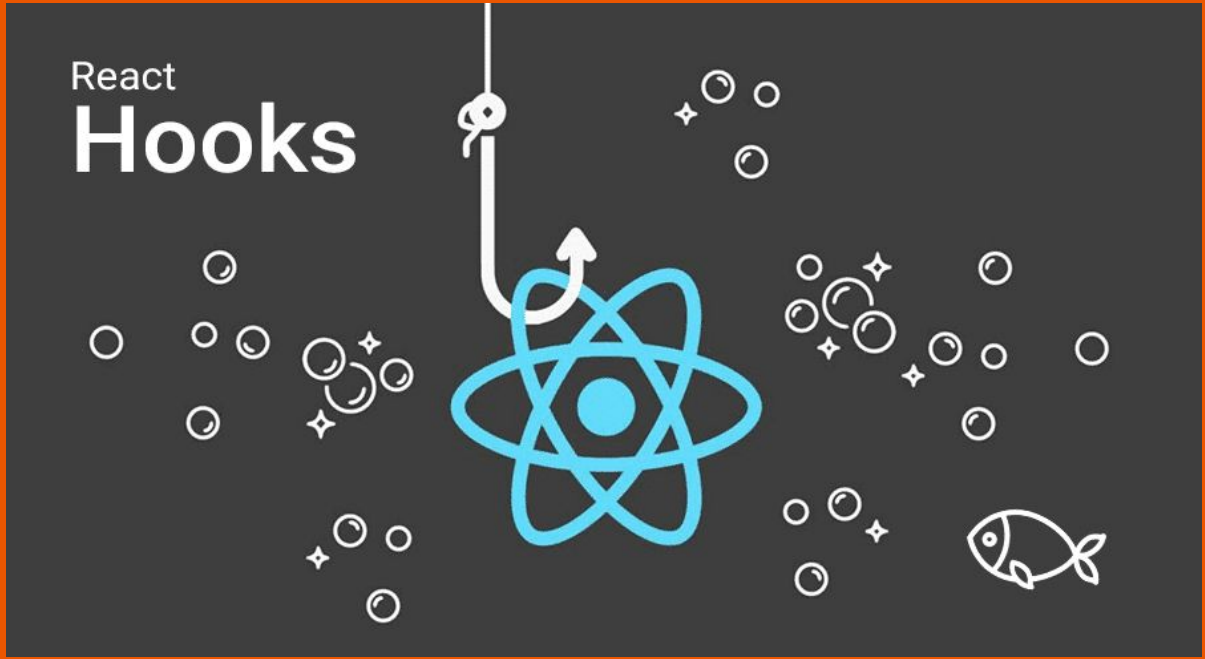
# Quizz

Why didn't state exist in function component before React 16.8.0?

***It was NOT Necessary.***

*- Function components were designed to be simple, reusable building blocks for building React applications, and they did not need access to state.*

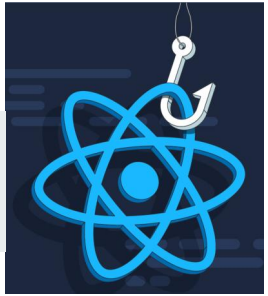
*- With the introduction of hooks, function components can now have access to state and other advanced features, making them more powerful and flexible.*


React
Hooks

# 1.1 What are React Hooks?

- React Hooks are a feature introduced in React version 16.8 that allow developers to **use state** and other React features <u>without writing class components.</u>
- Previously, state management was only possible inside class components using the state property.
- However, with the introduction of React Hooks, functional components can now have their own state and lifecycle methods.

*Simply hooks are features that allow you to "hook into" React state and lifecycle features from function components.*



# 1.1 What are React Hooks?

*Example: The below code segment shows the basic signature for a normal functional component, and we can hook into a particular state using this hook concept.*

```
function Example (props) {
  // Hooks can be used here
  return <div />;
}
```

# 1.2 Why use React Hooks?

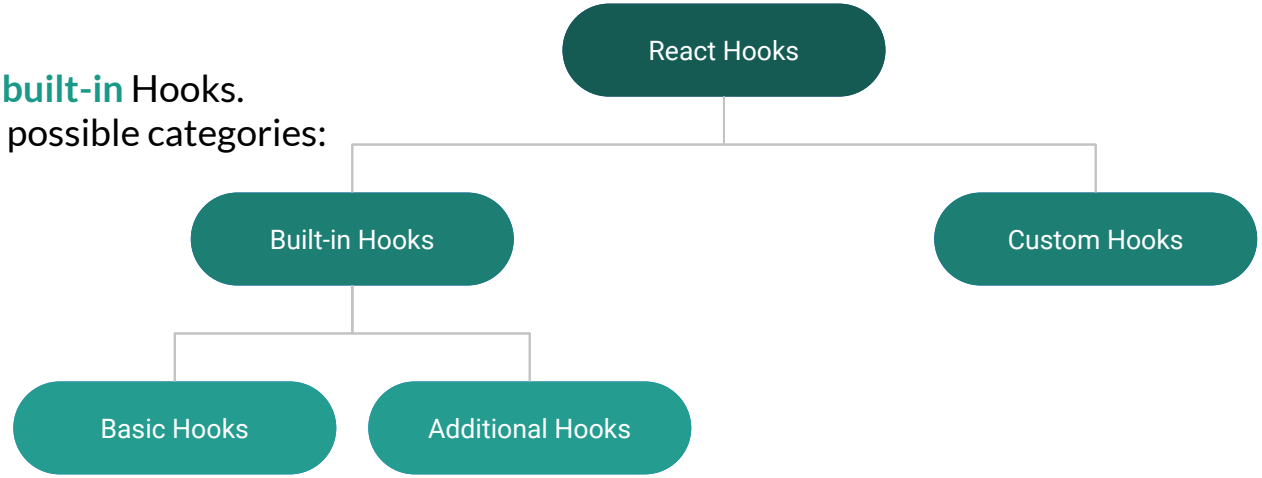
Before React Hooks, complex functionality such as state management, lifecycle methods, and side effects required the use of class components…

=> This often led to components becoming **large** and **difficult to understand**, maintaining a clear and concise codebase was a challenge.

- ❑ React Hooks solved this problem by allowing us to write more concise and readable code.
- ❑ Hooks enable us to separate concerns and reuse logic across different components effectively.
- ❑ Additionally, they **enhance** code reusability and **simplify** testing, making the development process more efficient and maintainable.

# 1.3 React Hook Types

- React provides some **built-in** Hooks.
- Can be divided into 3 possible categories:




- useState()
- useEffect()
- useContext()

- useReducer()
- useMemo()
- useCallback()
- useImperativeHandle()
- useDebugValue()
- useRef()
- useLayoutEffect()

# 1.3 React Hook Types

Basic Hooks

- **useState()**

The **`useState`** hook allows functional components to have their own state.

```
class Sample extends React.Component {
  constructor(props) {
    super(props);
   this.state = {
     age: 0
   };
 }
```

*In a class, by setting **`this.state`** to { age: 0 } in the constructor, we initialize the age state to 0.*

```
import React, { useState } from 'react';

function Sample() {
  // Initialize a new state variable called  "age"
  const [age, setAge] = useState(0);
}
```

*We **can't** use this.state inside the functional component. we call the **`useState`** hook directly.*

# 1.3 React Hook Types

Basic Hooks

- **useState()**

*example:*

```
import React, { useState } from 'react';

   function Sample() {
     const [age, setAge] = useState(0);

     return (
       <div>
         <p>You clicked {age} times</p>
         <button onClick={() => setAge(age + 1)}>
         Click me
        </button>
      </div>
    );
   }
```

# 1.3 React Hook Types

Basic Hooks

- **useEffect()**

❑ The **`useEffect`** hook allows functional components to handle side effects and perform actions after rendering.

❑ It takes two parameters: a function to be executed, and an optional array of dependencies.

❑ The function inside **`useEffect`** runs after the component renders, and it can perform actions such as fetching data, subscribing to events, or modifying the DOM.

```
1  useEffect(() => {
2      // side effect
3  })
```

# 1.3 React Hook Types

Basic Hooks

- **useEffect()**

➔ Basically, You can tell React that your component needs to do something after rendering by using this hook.

➔ After performing the DOM updates, React will remember the feature you passed and later call it. When our component is rendered, it will remember the effect we used previously and run our effect after updating the DOM.

# 1.3 React Hook Types

Basic Hooks

- **useEffect()**

*Example:*

```
import React, { useState, useEffect } from 'react';

function Sample() {
  const [rate, setRate] = useState(0);
  useEffect(() => {
    document.title = `You clicked ${rate} times`;
  });

  return (
    <div>
      <p>Hey!! you  have clicked {rate} times</p>
      <button onClick={() => setRate(rate + 1)}>
        Click
      </button>
    </div>
  );
}
```

# 1.3 React Hook Types

Basic Hooks

- **UseContext()**

➔ The **`useContext`** hook is a way to manage state globally.

➔ It can be used together with the useState Hook to share state between deeply nested components more easily than with useState alone.

➔ The **`useContext`** hook enables components to consume values from React Context API without the need for a Context.

➔

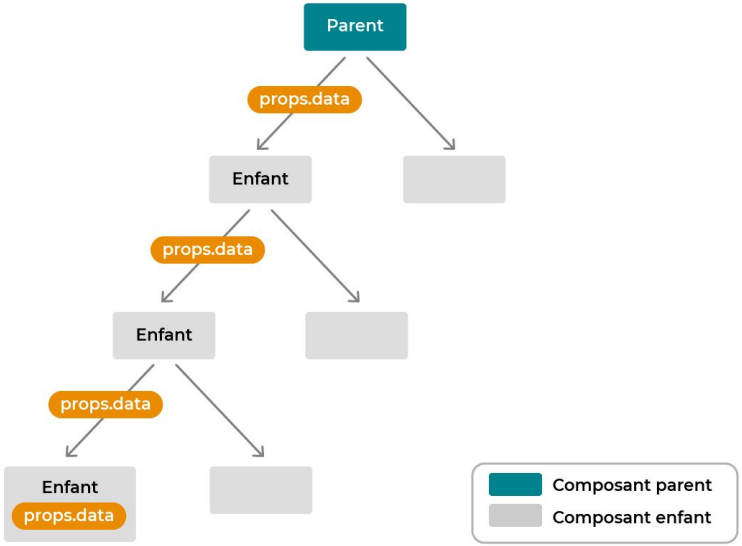
➔ Using the context in React requires 3 simple steps:

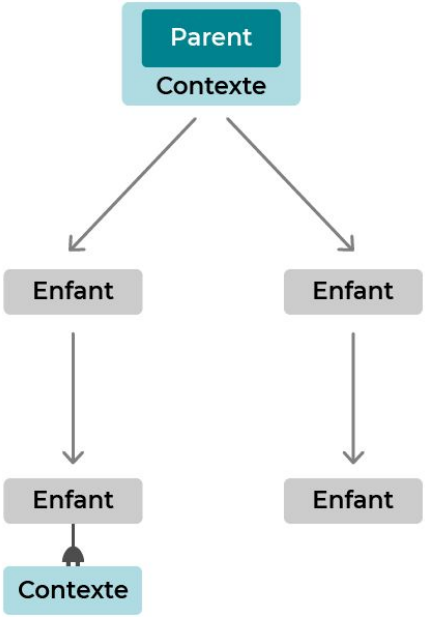
- Creating the context.
- Providing the context.
- Consuming the context.

# 1.3 React Hook Types

Basic Hooks

- **UseContext()**



# 1.3 React Hook Types

Basic Hooks

- **useEffect()**

*Example:*

- Providing the context:

```
import { Context } from './context';

function Main() {
  const value = 'My Context Value';
  return (
    <Context.Provider value={value}>
      <MyComponent />
    </Context.Provider>
  );
}
```

- Creating the context: **`context.js`**

```
// context.js
import { createContext } from 'react';

export const Context = createContext('Default Value');
```

- Consuming the context

```
import { useContext } from 'react';
import { Context } from './context';

function MyComponent() {
  const value = useContext(Context);

  return <span>{value}</span>;
}
```

# 1.3 React Hook Types

## Custom Hooks

- ➔ In addition to the basic hooks provided by React, you can also create custom hooks to encapsulate reusable logic.
- ➔ Custom hooks are **regular JavaScript functions** that utilize one or more of the basic hooks.
- ➔ By creating custom hooks, you can extract common logic and share it across multiple components in a more organized and reusable manner.

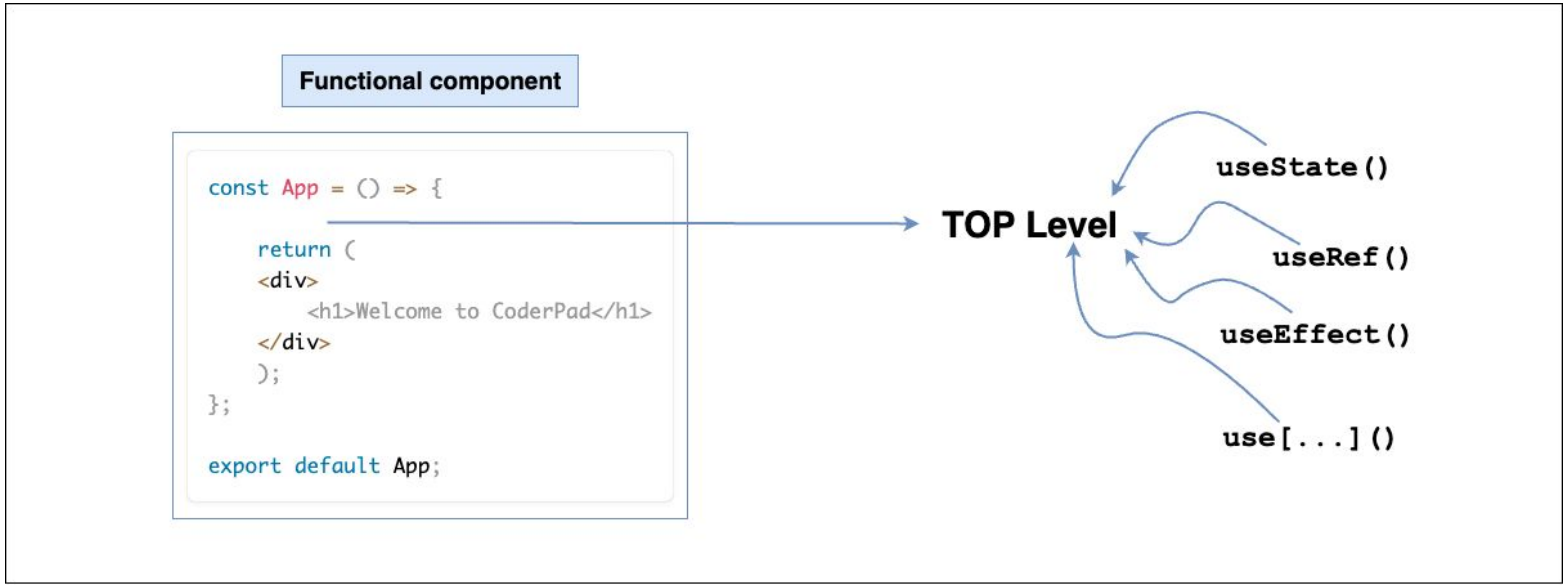
*Learn more about custom Hooks and how to create yours in: the React documentation.*

# 1.3 React Hook Types

Custom Hooks

# 1.4 Rules of React Hooks

- Only call Hooks at the top level of your functional component




*you must not be calling them within loops, conditions, and nested functions.*

# 1.4 Rules of React Hooks

- Only call Hooks from React functions.

```
Error: Invalid hook call.      VM757 react_devtools_backend.js:3973
Hooks can only be called inside
of the body of a function component. This could happen for one of the
following reasons:
1. You might have mismatching versions of React and the renderer (such
as React DOM)
2. You might be breaking the Rules of Hooks
3. You might have more than one copy of React in the same app
See https://reactjs.org/link/invalid-hook-call for tips about how to
debug and fix this problem.
```

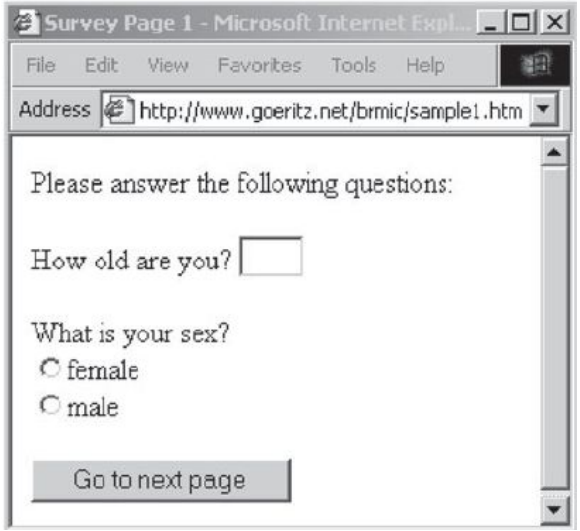
# React Hook Form

# 2. React forms

A classic HTML form (**`<form>`** tag) is used to obtain information from the user.

Usually, the information entered (input) is sent to a server so that it can be processed (to update or check the presence in a database).

The standard way of responding to the submission of an HTML form is to send the information entered to a server and the browser will load a new page sent by the  the server.

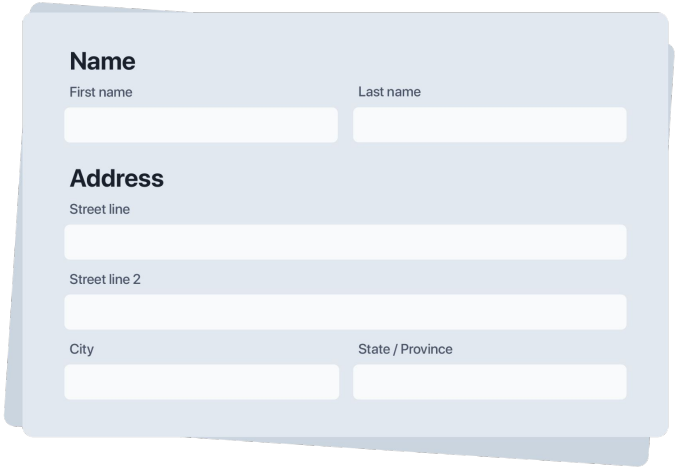



*React, being a popular JavaScript library for building user interfaces, provides effective ways to handle form creation and management.*

# 2. React forms

- Since most React applications are single page applications (SPAs), or web applications that load a single page through which new data is displayed dynamically, you won't submit the information directly from the form to a server.

- Instead, you'll capture the form information on the client-side and send or display it using additional JavaScript code.



# 2. React forms

- With its extensive collection of built-in hooks, React provides several features and techniques for creating and managing forms, including **state management**, **event handling**, and **form validation**.

# 2. React forms

*Example:*

```
import {useState} from 'react';

export default function  ControlledComponent()  {
  const  [inputValue, setInputValue] =  useState('');

  const  handleChange = (event) => {
    setInputValue(event.target.value);
  };

return  (
<form>
  <label>Input Value:
  <input  type="text"  value={inputValue} onChange={handleChange} />
  </label>
  <p>Input Value: {inputValue}</p>
</div>
)};
```

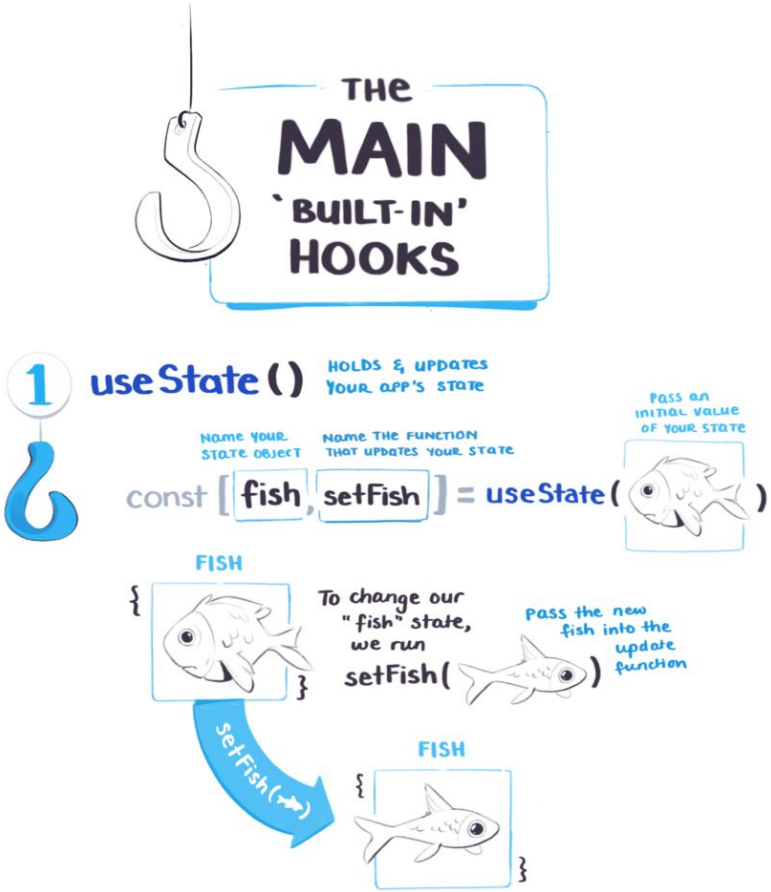
# 2. React forms

*Example:*


- ➔ As the user types into the input field, the **`handleChange`** function updates the state variable using the "**`setInputValue`**" function. The component is then re-rendered, and the input field's value attribute is updated to reflect the new value of inputValue.
- ➔ The value of the input field and the text displayed below it are always in sync, making it a controlled component.

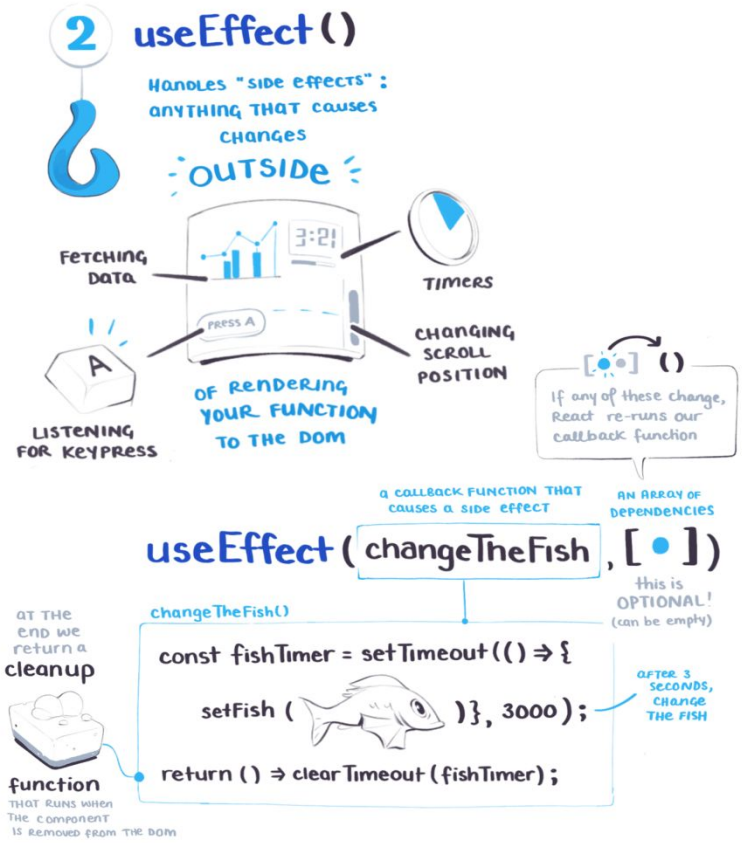
# Recap

# 2. Recap


THE
MAIN
'BUILT-IN'
HOOKS
1
useState()
HOLDS & UPDATES YOUR APP'S STATE
Name YOUR STATE OBJECT
Name THE FUNCTION THAT UPDATES YOUR STATE
Pass an INITIAL VALUE OF YOUR STATE
const [fish, setFish] = useState( )
FISH
To change our "fish" state, we run
setFish( )
Pass the new fish into the update function
setFish( )
FISH

# 2. Recap

*More details:*
*[Illustrated notes on building custom React hooks]*



# Lab Exercises Submission Guidelines

- **Deadline:**
  At the end of each Lab session (no later than Saturday at 23:59)
  To: adil.chekati@univ-constantine2.dz

- **Link to be submitted:**
  Github repository link.

# Textbook

➔ All academic materials will be available on:

Google Drive.
E-learning platform of Constantine 2 University.
Google Classroom.

aoa5lne

Google Classroom

SCAN ME!

# References

➔ **Book:**
Stoyan Stefanov -*React Up and Running: Building Web Applications - 2nd edition (2021).*


➔ **Online Resource:**
Illustrated notes on building custom React hooks
(https://maggieappleton.com/customhooks )

How to Build Forms in React
(https://www.freecodecamp.org/news/how-to-build-forms-in-react/ )

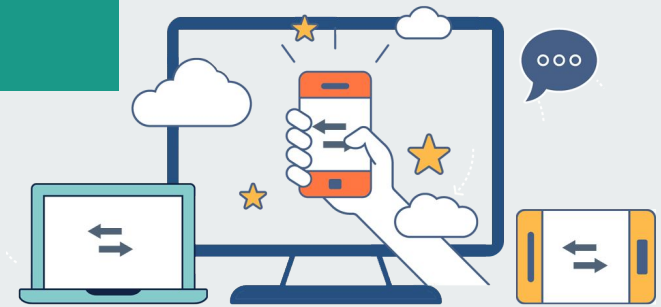
# Next Lecture

*-Lecture 10-*

Chapter 7– **React Router**

Adil **CHEKATI**, PhD

adil.chekati@univ-constantine2.dz



# Questions, & comments...

adil.chekati@univ-constatine2.dz