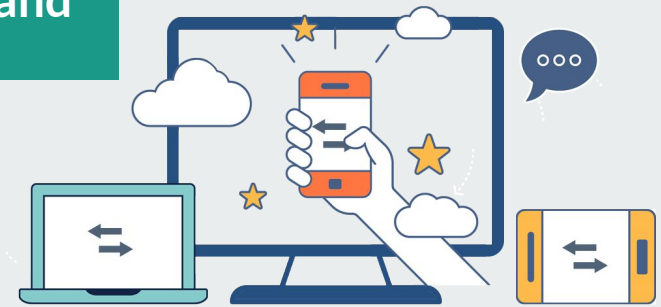
-Lecture 7-

Chapter 4–**Create-React-App, Modules, states and React events.**

Adil **CHEKATI**, PhD

adil.chekati@univ-constantine2.dz



Prerequisites

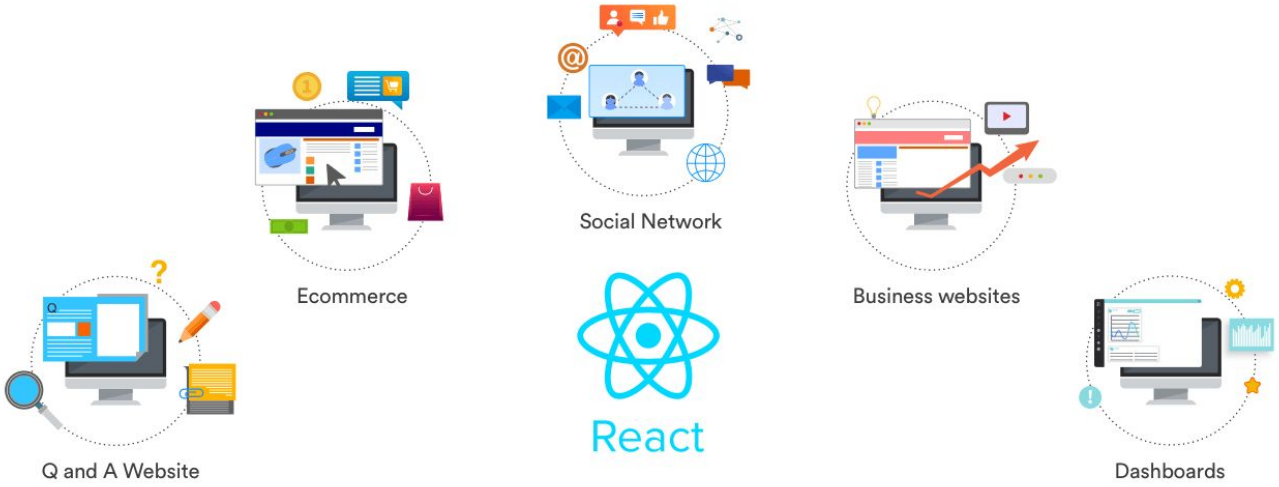
- ❑ Basic Understanding of React Fundamentals (Knowledge)
- ❑ Understanding of JavaScript ES6 Modules (Knowledge)

CRA, Modules, states and React events

Objectives

- ➔ Recall the purpose and advantages of using Create-React-App.
- ➔ Comprehend how ES6 modules improve code maintainability and structure in React projects.
- ➔ Discover React event handling concept and how it enhances user engagement.

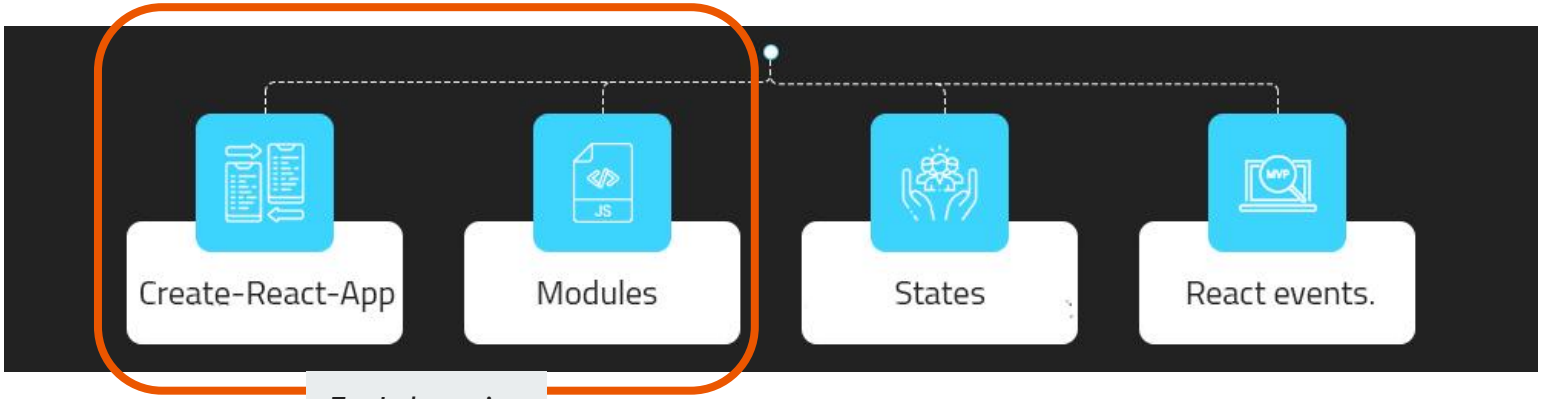
Introduction



*The possibilities that React provides us are **huge**.*
*We can build **almost anything** using it; From the most basic cases like landing pages or blogs to the most complex, like games and e-commerce apps.*

"We can build anything using React"

Introduction



For Lab session

1. Create-React-App

- Create-React-App (CRA) is a command-line tool and build configuration that helps you set up a new React.js web application with **zero configuration**.
- It is an official React project from Facebook and is designed to streamline the process of starting a new React project by providing a **pre-configured** development environment.
- CRA handles all the build tools, dependencies, and build scripts, meaning you don't have to spend time configuring and setting up complex development environments.

*CRA **abstracts** away the complex configuration settings, allowing developers to focus solely on building the application.*

1. Create-React-App

1.1. Features of Create-React-App

React + BABEL + webpack MODULE BUNDLER

a) Zero Configuration

CRA sets up a fully functional React project with all the necessary configurations, dependencies, and scripts out of the box, eliminating the need for manual setup.

b) Webpack Configuration

CRA configures and abstracts away the complex Webpack configuration, giving developers a pre-configured environment ready to use.

c) Babel Configuration

CRA handles the Babel configuration, allowing developers to write code with the latest JavaScript features and have them automatically transpiled to be compatible with all browsers.

1. Create-React-App

1.1. Features of Create-React-App

d) Live Development Server

CRA starts a development server that automatically reloads the application whenever changes are made, providing an efficient development workflow without the need to manually refresh the page.

e) Build Optimizations

CRA optimizes the production build by generating minified and optimized code, ensuring fast and efficient performance.

f) Integrations

CRA supports various integrations and plugins, such as CSS preprocessors, testing frameworks, and analytics tools, making it easy to extend the functionality of your React application.

1. Create-React-App

1.2. Benefits of Using Create-React-App

a) Rapid Development

CRA eliminates the need for setting up complex build configurations, allowing developers to start coding immediately and focus on building the application.

b) Consistent Development Environment

By standardizing the development environment, CRA ensures that all team members are working with the same tools, configurations, and dependencies, promoting collaboration and reducing errors.

c) Easy Updates

CRA simplifies the process of updating to the latest React version. With a single command, you can update the underlying tools and dependencies, ensuring compatibility and taking advantage of new features.

1. Create-React-App

1.2. Benefits of Using Create-React-App

d) Community Support

Create-React-App is an official React project with a large and active community. This means you can find extensive documentation, tutorials, and community support to help you solve problems and learn best practices.

e) Pre-configured Best Practices

CRA enforces a set of best practices for React development. It sets up a proper project structure, includes recommended dependencies, and encourages the use of modern practices, ensuring your codebase follows industry standards.

1. Create-React-App

1.3. Setting up

- ➔ CRA is written in Node and requires Node 6+Node.js®.
- ➔ To use it, you need to install Node first, then install it.

Setup details to be explained in the Lab session.

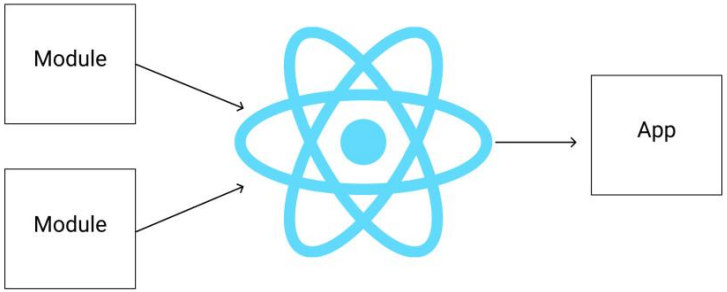
2. React modules

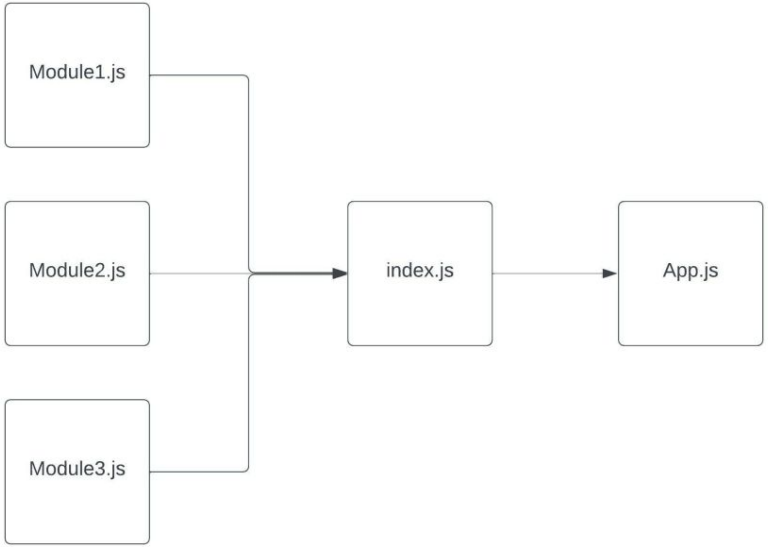
- React Modules are **reusable** pieces of code that encapsulate a set of functionalities and components in the React ecosystem.
- React Modules are designed to promote code **reusability**, **maintainability**, and **organization** in large-scale applications.

*In React, modules are typically written using a combination of **JavaScript**, **JSX** (a syntax extension for JavaScript), and **CSS**. They can contain various elements such as: components, helper functions, stylesheets, and utility classes.*

2. React modules

➔ React lets you put your files into a directory and then create an `index.js` file that exports all.





2. React modules

2.1. Creating and Using React Modules

- ➔ To create a React Module, you start by defining the required components, functions, and styles.
- ➔ You can package these elements into a single reusable module that can be imported and used by other parts of the application

More Details to be explained in the Lab session.

3. React States

- State refers to the data that represents the current state of the user interface.
- It helps to create dynamic and interactive components by allowing them to hold and manipulate data.
- Proper management of state is **crucial** for building scalable and maintainable React applications.

state is used for internal communication inside a Component

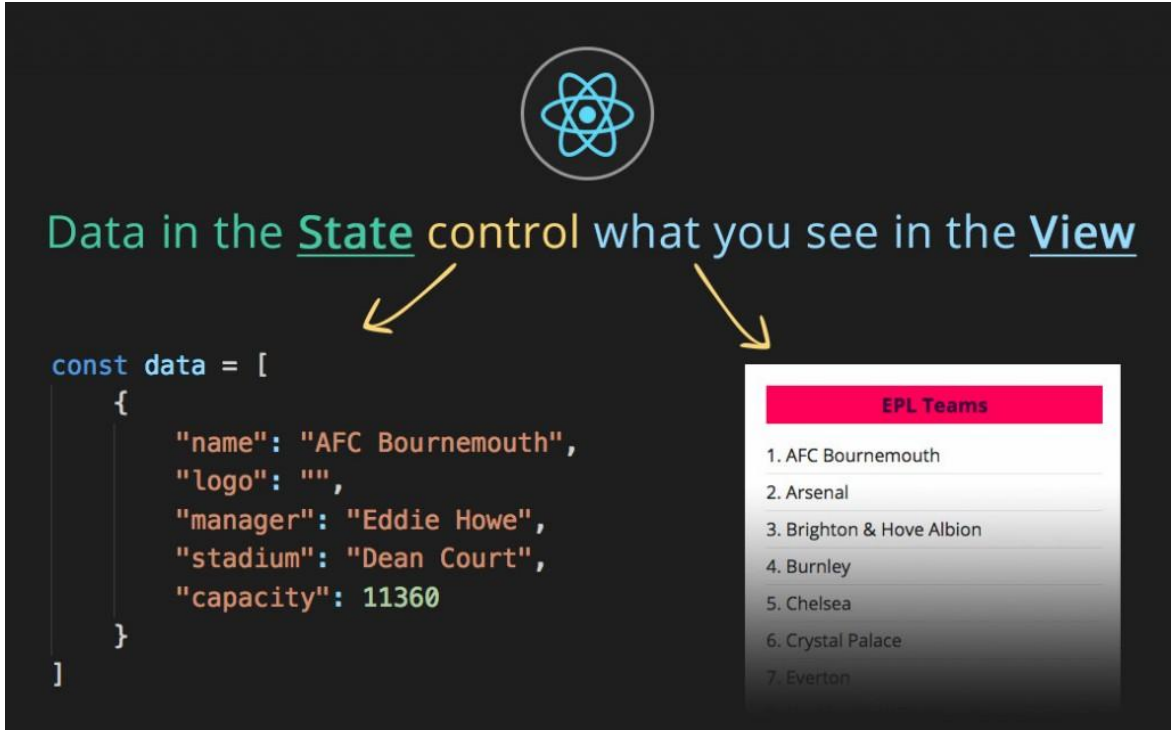
3. React States

3.1 What is a React state?

- ➔ In React, a state is an object, i.e. a set of pairs (`keys:values`) representing an attribute of a component instance.
- ➔ It is **not mandatory** for a component.
- ➔ You can package these elements into a single reusable module that can be imported and used by other parts of the application

3. React States

3.1 What is a React state?



3. React States

3.2 Initial state

- If a component needs to have a State attribute, it **must be initialized** when the component is created in the constructor function.
- The constructor is not mandatory for a stateless component, but in the case of a stateful component, you'll **need** a standard React **constructor**.

```
constructor(props) { super(props);
    this.state = {
    /* initial state values */ };
}
```

3. React States

3.2 Initial state

```
constructor(props) {
    super(props);
    this.state = {
    /* initial state values */ };
}
```

- ➔ `constructor` takes one argument, `props`.
- ➔ You need to call `super(props)` at the start of the constructor, which registers the class as a React component.
- ➔ Instance methods provide access to :
  - ◆ state properties with `this.state`
  - ◆ props properties with `this.props`

3. React States

Example of a component with states:

```
class Game extends React.Component {

    constructor(props) {
    super(props);
    this.state = {
        player: 'Riad',
        score: 0
    };
    }

    render() {
    return (
        <div>
        <h1>Football</h1>
        <p>Current Player: {this.state.player}</p>
        <p>Score: {this.state.score}</p>
        </div>
        );
    }
}
```

3. React States

3.3 Change of state

➔ `this.setState()` is React's built-in method for changing the state of a component.

```
this.setState({
    playerName: "Riad",
    score: 2 })
```

3. React States

3.3 Change of state

The built-in method `this.setState` :

- ➔ Can be called from any instance method **except** the constructor.
- ➔ Has for parameter an object showing state changes.
- ➔ The **only** method that allows you to change the state of a component:
  You can't directly access and modify the state of a component without this method.
- ➔ Properties not specified in state changes are not modified.
- ➔ Is **asynchronous!**
  - ◆ The state of the component may change.
  - ◆ React controls when the change of state actually takes place, for performance reasons.
- ➔ Components are re-rendered when their stat changes.

3. React States

Example of state change:

```
class Rando extends React.Component {
    constructor(props) {
    super(props);
        this.state = { num: 0
    }; this.makeTimer();
    }

 makeTimer() {
 setInterval(() => {
    this.setState({
    num: Math.floor(Math.random() * this.props.maxNum) });
    }, 1000);
 }

 render() {
    return <h1>Random number: {this.state.num}</h1>;
 }
 }
```

3. React States

3.4 Managing Complex State

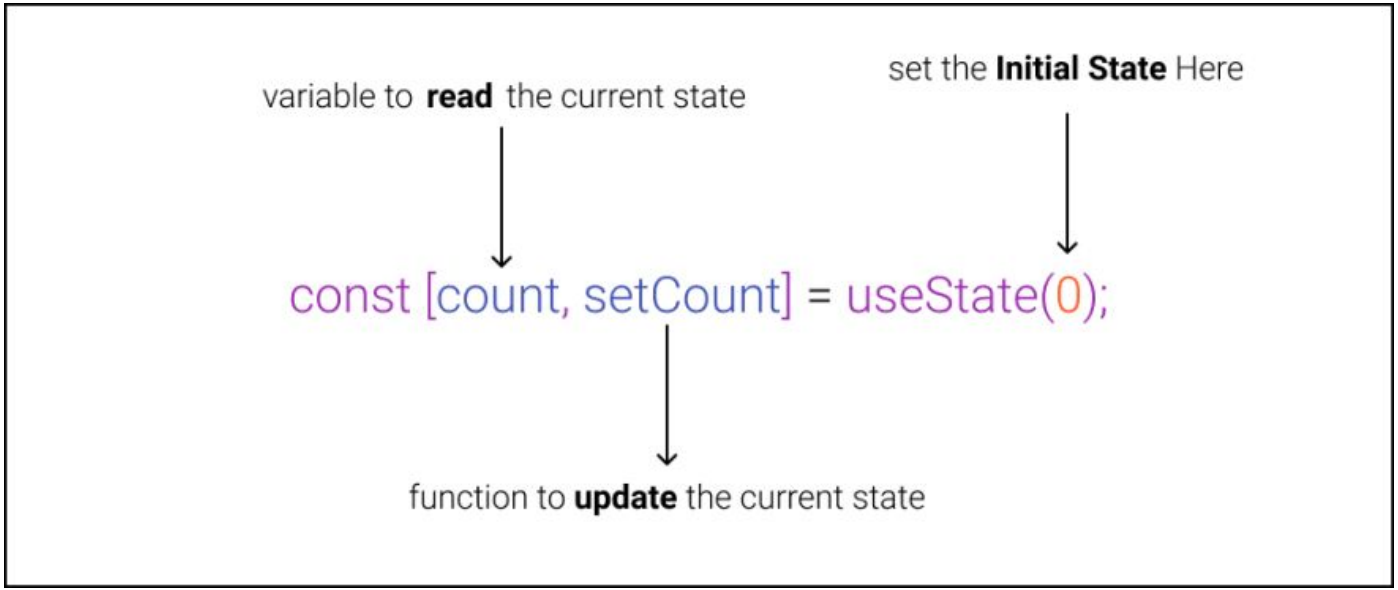
- Before the release of **React Hooks**, we could only manage our state using class components. The release of **React Hooks** gave start to a new era in the React community.

- The most-used hook in React is `useState`.

- `useState` hook is used inside a functional component and that will make the component associated with that state in particular.

3. React States

3.4 Managing Complex State



3. React States

Example of useState:

```
1  import React, { useState } from "react";
2
3  const App = () => {
4    const [count, setCount] = useState(0);
5
6    const handleIncrement = () => setCount(count + 1);
7    const handleDecrement = () => setCount(count - 1);
8    const handleReset = () => setCount(0);
9    return (
10     <div>
11       <p>Count: {count}</p>
12       <button onClick={handleIncrement}>Increment</button>
13       <button onClick={handleDecrement}>Dcrement</button>
14       <button onClick={handleReset}>Reset</button>
15     </div>
16   );
17 }
```

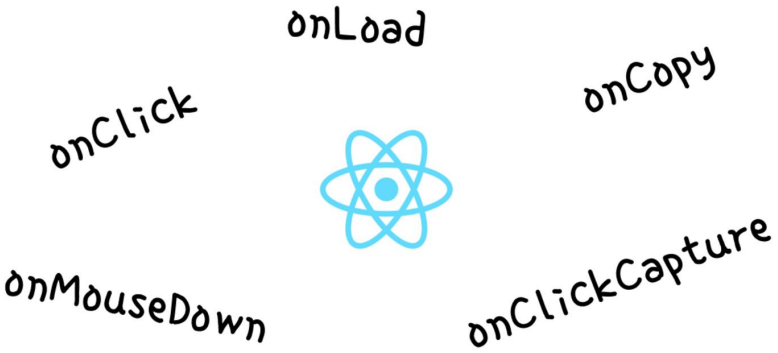
3. React States

3.4 Managing Complex State

- ➔ In more complex React applications, state can become more than just a single value. It can be an object or an array that holds multiple values.
- ➔ `useState` **Not** for Complex State Management
- ➔ To manage complex state, we can use the useState hook multiple times or combine multiple state variables into a single object.

4. React events

- The state of a component usually changes in response to an event.
- In React, each JSX element has built-in attributes representing categories of events frequently encountered in the browser (the same events encountered in HTML).
- They are represented in camel-case notation, like `onClick`, and take callback functions like event-handle.



4. React events

4.1 React synthetic events

- These are "**advanced**" React events built on top of the basic browser events.
- They are called synthetic events, but in practice they behave in the same way (named in camel-case, with callback function as event-handler).
- For further details, see the React documentation for all supported event types.

4. React events

4.2 React handling

- In React, event handling is quite similar to traditional JavaScript event handling.
- However, instead of using inline event handlers or `addEventListener()` method, React uses a **declarative approach** to handle events.

- To handle events in React, you need to:
  - ➔ Define an event handler function.
  - ➔ Attach the event handler to the desired element or component.

4. React events

4.2 React handling

- ➔ In React, event handlers are defined as functions within the component's class or functional component.

- ➔ Usually named using the `handle` prefix followed by the event name. *For example*, to handle a click event, you might name the function handleClick .

4. React events

4.2 React handling

```
class MyComponent extends React.Component {
    handleClick() {
    // Event handling logic goes here.
    }

    render() {
    return <button onClick={this.handleClick}>Click me</button>;
    }
   }
```

handleClick is the event handler function for the onClick event of the button element. Inside the function, you can write the logic to be executed when the event occurs.

4. React events

4.3 Binding Event Handlers

Problem with `this`

Here **`this`** is not recognized (undefined)!

- ➔ Who calls **`handleClick`**?
  - React, after click (normally)
- ➔ What's he calling her about?
  - React doesn't "remember" the instance on which to call **`handleClick`**
  - The method was called "out of context".
- ➔ What should we do?
  - Use method binding via `.bind()`

Correction of the problem

To solve this problem, we need to bind the handleClick instance method in the constructor as follows: (other binding solutions exist)

4. React events

4.3 Binding Event Handlers

Correction of the problem

To solve this problem, we need to bind the **`handleClick`** instance method in the constructor:

```
1   class MyComponent extends React.Component {
2     constructor(props) {
3       super(props);
4       this.handleClick = this.handleClick.bind(this);
5     }
6
7     handleClick() {
8       // Event handling logic goes here.
9     }
10
11    render() {
12      return <button onClick={this.handleClick}>Click me</button>;
13    }
14  }
```

(other binding solutions exist)

4. React events

Complete example: Click Rando

```
class ClickRandom extends React.Component {
  constructor(props) {
  super(props);
  this.state = { num: 0 };
  this.handleClick = this.handleClick.bind(this); }

  setRandom() {
      this.setState({
      num: Math.floor(Math.random() * this.props.maxNum)
      });
      }
  handleClick(evt) {
      this.setRandom();
  }
  render() {
      return (
      <div>
          <h1> Number = {this.state.num}</h1>
          <br/>
          <button onClick = {this.handleClick}> Generate Random </button> </div>
      );
  }
  }
```

4. React events

- ➔ *Handling events in React is crucial for creating **interactive** and **responsive** user Interfaces.*
- ➔ *By defining event handler functions and attaching them to elements or components, you can respond to various user actions.*
- ➔ *Remember to **bind** the this value correctly, access the event object.*
- ➔ *Incorporating event handling into your React components will enhance the interactivity and usability of your applications.*

5. List of components and keys

- ➔ You can create collections of components and include them in JSX using { }.
- ➔ When producing a collection of components, it is necessary to assign each component a **unique key** in the list.

```
class Comp extends React.Component {
    render() {
    let stars = [2,1,6,9].map((e,i) =>
    <li> value is {e}</li> );

    return(
    <div>
        <ul>
        { stars }
        </ul>
    </div>)
    }
}
```

This code will produce a warning: ***`Warning: Each child in an array or iterator should have a unique "key" prop.`***

5. List of components and keys

Fixing: A key attribute must be defined to identify elements that have been added, deleted or modified. Typically, a key identifies a single element, such as a database key. Otherwise, the key is simply a number in the list (or an index of the browsed table).

```
class Comp extends React.Component {
    render() {
    let stars = [2,1,6,9].map((e,i) =>
    <li key ={i}> value is {e}</li> );

    return(
    <div>
        <ul>
        { stars }
        </ul>
    </div>)
    }
}
```

(other binding solutions exist)

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:**
Zac Gordon - *React Explained (2021)*

MOOC
React Udacity NanoDegree
https://www.udacity.com/course/react-nanodegree--nd019 on Udacity

Online Resource:
React.js official documentation
(https://react.dev/learn)

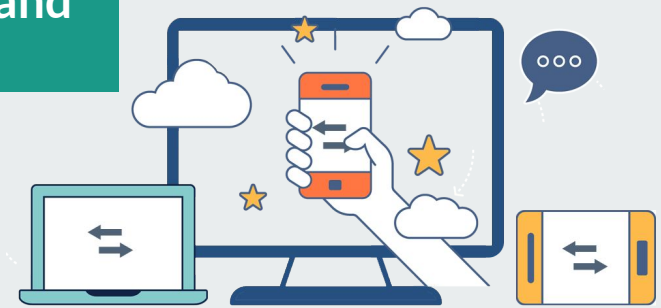
Next Lecture

-Lecture 8-

Chapter 4–**Create-React-App, Modules, states and React events.**

Adil **CHEKATI**, PhD

adil.chekati@univ-constantine2.dz



Questions, & comments...

adil.chekati@univ-constatine2.dz