COMPONENTS

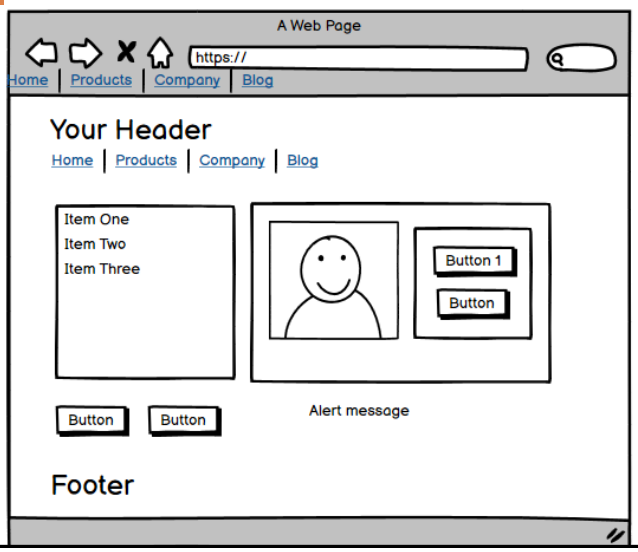
Lect4

1

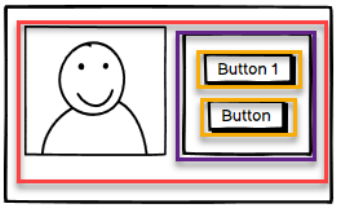
- Components are the core building blocks of React applications.
- React Components are built in one of two ways **functions** or **classes**.
- A React interface can be made-up from a hierarchy of components. You could make a react component with a single component or break it into you smaller more manageable quote blocks.

2

Ui built from a number of components



- Header, Content, lists, avatar Button footer.
- Each component/element can be nested as deep ae needed.



3

- React components return an element, the name of the element is given by the **function** or **class** name.
- Heading.jsx example

```
let Heading = () => {
    return ( <>
        <h1 className="heads">Header section</h1>
        <h3>Menu section</h3>
    </> )
}
export default Heading;
```

4

ny-app > src > JS index.js

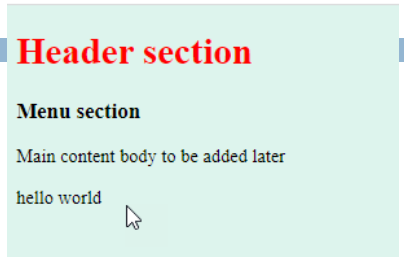
index.js

```
1  // import React from 'react'
2  import ReactDOM  from 'react-dom/client'
3  import App from './App'
4
5  ReactDOM.createRoot(
6      document.getElementById('root')).render(<App />)
7
```

my-app > src > JS App.js > ...

App.js

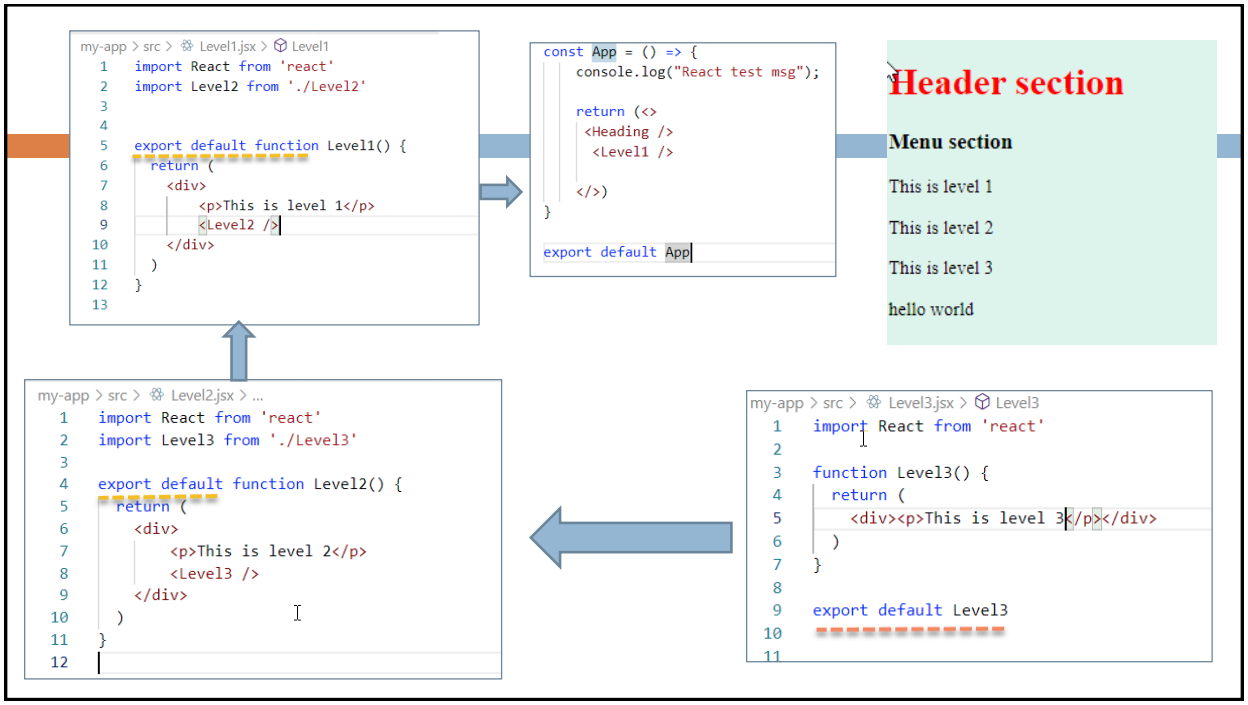
```
1   import Heading from "./Heading"
2
3   const App = () => {
4       console.log("React test msg");
5
6       return (<>
7        <Heading />
8         <p>Main content body to be added later</p>
9       </>)
10  }
11
12  export default App
```



5

- Each component within an application has a unique name starting in **upper case**.
- The components created use the **default export** statement.
- The file containing the module usually takes the name of the component defined.
- Once the component is imported into another component, the imported component's functionality can be included in your new component
- This promotes reuse and reduces complexity

6



7

- You can nest components as deep as you wish.
- This levels of abstraction decreases application complexity.

8

exercise

- Update <Heading /> to display an image logo and heading on the one line.

9

- React has built-in components for the most commonly used HTML element

footer	nav
form	noscript
h1	object
h2	ol
h3	optgroup
h4	option
h5	output
h6	p
head	param
header	picture
hr	pre
html	progress
i	q
iframe	rp
img	rt
section	ruby
select	s
small	samp
source	script
span	

10

- Most of the HTML attributes are supported in React

```
accept acceptCharset accessKey action allowFullScreen alt async autoComplete
autoFocus autoPlay capture cellPadding cellSpacing challenge charSet checked
cite classID className colSpan cols content contentEditable contextMenu controls
controlsList coords crossOrigin data dateTime default defer dir disabled
download draggable encType form formAction formEncType formMethod formNoValidate
formTarget frameBorder headers height hidden high href hrefLang htmlFor
httpEquiv icon id inputMode integrity is keyParams keyType kind label lang list
loop low manifest marginHeight marginWidth max maxLength media mediaGroup method
min minLength multiple muted name noValidate nonce open optimum pattern
placeholder poster preload profile radioGroup readOnly rel required reversed
role rowSpan rows sandbox scope scoped scrolling seamless selected shape size
sizes span spellCheck src srcDoc srcLang srcSet start step style summary
tabIndex target title type useMap value width wmode wrap
```

11

- Because JSX is an XML markup language, JSX elements can have attributes

```
<body>
    <input type="number" name="" id="" placeholder="0" default="3">
</body>
</html>
```

element attributes

```
import AttributeDemo from "./AttributeDemo.js";

const App = () => {
    console.log("React test msg");

    return (<>
        <AttributeDemo name="Gerard" />

    </>)
}
```

```
export default function AttributeDemo(props) {
  return (
    <div>attributeDemo {props.name}</div>
  )
}
```

React attributes

12

class and for Attributes

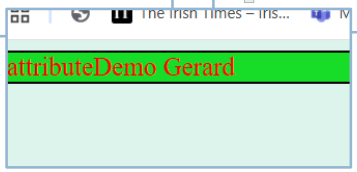
- The class attribute in HTML is **className** in React.
- The for in HTML is **htmlFor** in React

AttributeDemo.js > ...

```
export default function AttributeDemo(props) {
  return (
    <div className="demo" >attributeDemo {props.name}</div>
  )
}
```

App.css > .demo

```
.demo {
    color: red;
    background-color: rgb(23, 221, 40);
    font-size: 20px;
    border: 2px solid black;
}
```

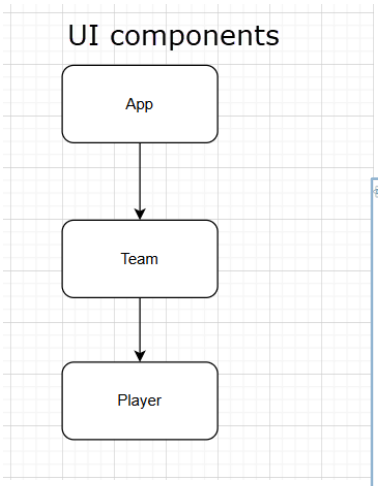


13

- Reacts JSX component element/attribute properties are called props (properties).
- While HTML is case-insensitive, **JavaScript is not.**
- Where ReactJS's JSX has HTML-like stuff embedded in JavaScript, use camelCase for attributes
- React's built-in HTML element components have the same names as elements from HTML5. Using them in your React app causes the equivalent HTML element to be rendered.
- The JSX elements attributes passed onto components are called **props**. Other names can be used but this would be the common React convention

14

Component example




```
import './App.css';
import Team from "./Team";

const App = () => {
    console.log("React test msg");

    return (<>
        <Team />
    </>)
}

export default App
```

```
Team.jsx > Team
import Staff from './Staff';

export default function Team(){
  return(
    <>
      <Staff 1
        manager="Jim McGuinness"
        players={['Patrick McBrearty','Hugh McFadden','Shaun Patton']}/>
      <Staff 2
        manager="Ger Brennan"
        players={['Brian Howard','Paddy Small','Con O Callaghan','AN Other']}/>
    </>
  )
}
```

Attributes/props passed to Staff component

15

Staff.jsx

```
src > Staff.jsx > Staff
3  function Staff(props) {
14   let squad = [];
15   if (props.players) {
16     squad = props.players.map((player, ndx) =>
17       <li key={ndx}> {ndx} {player}</li>
18     )
19   }
20
21   return (
22     <div>
23       <p>Team manager {props.manager}.</p>
24       <p>Key players {props.players[0]} {props.players[1]} {props.players[2]} </p>
25       <ul>
26         {squad}
27       </ul>
28     </div>
29   )
30 }
31
32 export default Staff
```

Team manager Jim McGuinness.

Key players Patrick McBrearty Hugh McFadden Shaun Patton

- 0 Patrick McBrearty
- 1 Hugh McFadden
- 2 Shaun Patton

Team manager Ger Brennan.

Key players Brian Howard Paddy Small Con O Callaghan

- 0 Brian Howard
- 1 Paddy Small
- 2 Con O Callaghan
- 3 AN Other

```
<Staff
  manager="Jim McGuinness"
  players={['Patrick McBrearty','Hugh McFadden','Shaun Patton']}/>
```

16

- The `map()` method creates a new array populated with the results of calling a provided function on every element in the calling array.

```
let squad = [] ;
  if (props.players) {
      squad = props.players.map( (player, ndx) =>
      <li key={ndx}> {ndx} {player}</li>
      )
  }
```

17

- React components can be written in two different ways:
- **Functional Components**: Functional components are simply JavaScript functions

my-app > src > JS quote1.js > default

```
1  const Quote1 = ()=>
2  {
3      return <h1>Function Quote!</h1>;
4  }
5
6  export default Quote1
```

18

- **Class Components**: The class components more complex and not used a much as functional components.
- Both can be used.

```
2   import { Component } from 'react'
3
4   class Quote2 extends Component
5   {
6       render()
7       {
8           return  (<h1>Class Quote message 2!/</h1> )
9       }
10  }
11
12  export default Quote2
```

19

JavaScript Destructuring

- The destructuring assignment is a unique syntax that helps “to unpack” objects or arrays into a group of variables.
- Arrays

```
let arr = [ 1,2, 3, 4];

let a = arr[0];
let b = arr[1];
let c = arr[2];
let d = arr[3];
```

- `// destructuring arrays`
- `let [a,b,c,d] = arr;`

20

- Object destructuring
- let student = {

```
    name: "Fred",
    CAO: "L0002344",
    age: 23
}
let name = student.name;
let CAO = student.CAO;
let age = student.age;
```

```
// destructuring objects
let {name, CAO, age } = student;
console.log(name);
```

21

```
function render(props) {
    var name = props.name;
    var age = props.age;
}

// destructuring functions
function render({name, age}) { }
```

- Destructuring makes the assignment of variables even easier

22

my-app > src > JS Staff.js > Staff

```
export default function Staff(props){

  let squad = [] ;
  if (props.players) {
    squad = props.players.map( (player, ndx) =>
    <li key={ndx}> {ndx} {player}</li>
    )
  }

  return (
  <div>
    <p>Team manager {props.manager}.</p>
    <p>Key players {props.players[0]} {props.players[1]} {props.players[2]} </p>
    <ul>
      {squad}
    </ul>
  </div>
  )

}
```

23

my-app > src > JS Staff.js > ...

```
export default function Staff({manager, players}){

  let squad = [] ;
  if (players) {
    squad = players.map( (player, ndx) =>
    <li key={ndx}> {ndx} {player}</li>
    )
  }

  return (
  <div>
    <p>Team manager {manager}.</p>
    <p>Key players {players[0]} {players[1]} {players[2]} </p>
    <ul>
      {squad}
    </ul>
  </div>
  )
}
```

destructuring props

24

Firsat hook

- To add state to our application's *App* component with the help of **React's state hook**.
- We will look at state and events in more details next week.
- For now we will examine a simple example that changes the state of a components data.

25

Staff.jsx > Staff

```
import { useState } from 'react'  1

function Staff(props) {

  const [theManager, setManager] = useState("Jim McGuinness");  2

  function handleClick() {
    let name = props.manager === "Jim McGuinness" ? "Donegal manager" : "Dublin manager";
    console.log(name);
    setManager(name)
  }

  let squad = [];
  if (props.players) {
    squad = props.players.map((player, ndx) =>
      <li key={ndx}> {ndx} {player}</li>
    )
  }

  return (
    <div>
      <p>Team manager {theManager}.</p>  4
      <p>Key players {props.players[0]} {props.players[1]} {props.players[2]} </p>
      <ul>
        {squad}
      </ul>
      <button onClick={handleClick} >Change Manager</button>  3
    </div>
  )
}
```

Component: **Staff** (functional React component)
Imports the **useState** hook from React
Receives props:

```
manager (string)
players (array of player names)
```

26

State Management

```
1  import { useState } from 'react'
2
3  function Staff(props) {
4
5    const [theManager, setManager] = useState("Jim McGuinness");
6
```

- `theManager`: current manager value displayed in the UI
- `setManager`: function to update manager state

27

Event Handling

- `handleClick` function: Checks the value of props.manager
- If "Jim McGuinness", sets new state "Donegal manager“
- Otherwise sets "Dublin manager“
- conditional logic + state update

```
const [theManager, setManager] = useState("Jim McGuinness");


function handleClick() {
  let name = props.manager === "Jim McGuinness" ? "Donegal manager" : "Dublin manager";
  console.log(name);
  setManager(name)
}
```

28

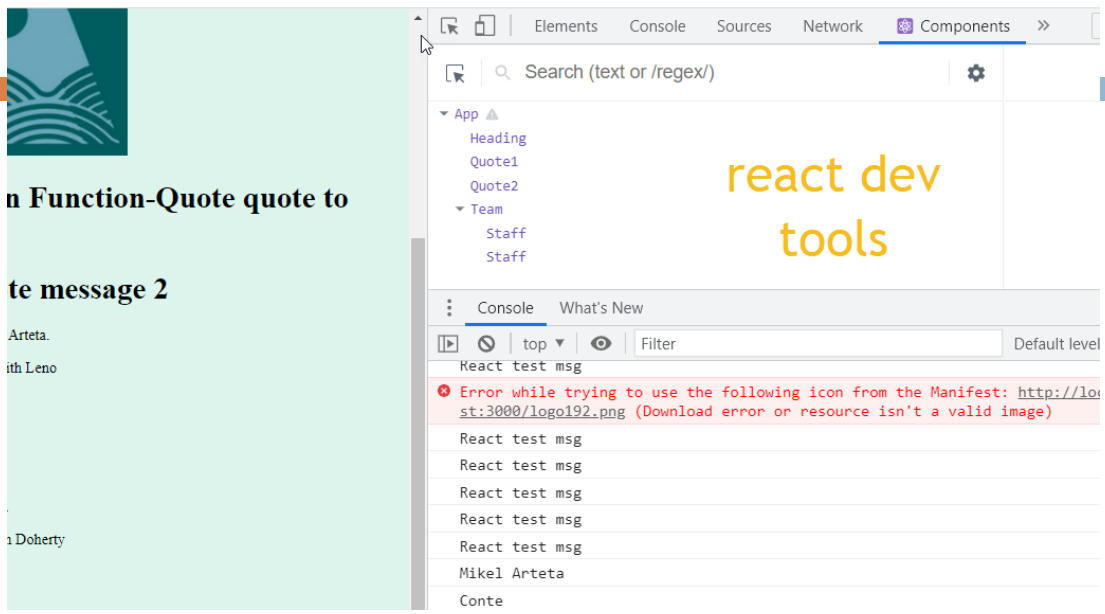
Rendering Players

- Uses `props.players` to generate a dynamic list

```
let squad = [];
if (props.players) {
  squad = props.players.map((player, ndx) =>
    <li key={ndx}> {ndx} {player}</li>
  )
}

return (
  <div>
    <p>Team manager {theManager}.</p>
    <p>Key players {props.players[0]} {props.players[1]} {props.players[2]} </p>
    <ul>
      {squad}
    </ul>
    <button onClick={handleClick} >Change Manager</button>
  </div>
)
```

29



30

Summary

- Reviewed components and elements.
- Demonstrated and tested how to use React's HTML elements.
- Investigates how to pass data between components with props.
- Reviewed the differences between writing class and function components.
- How to change component state.

31