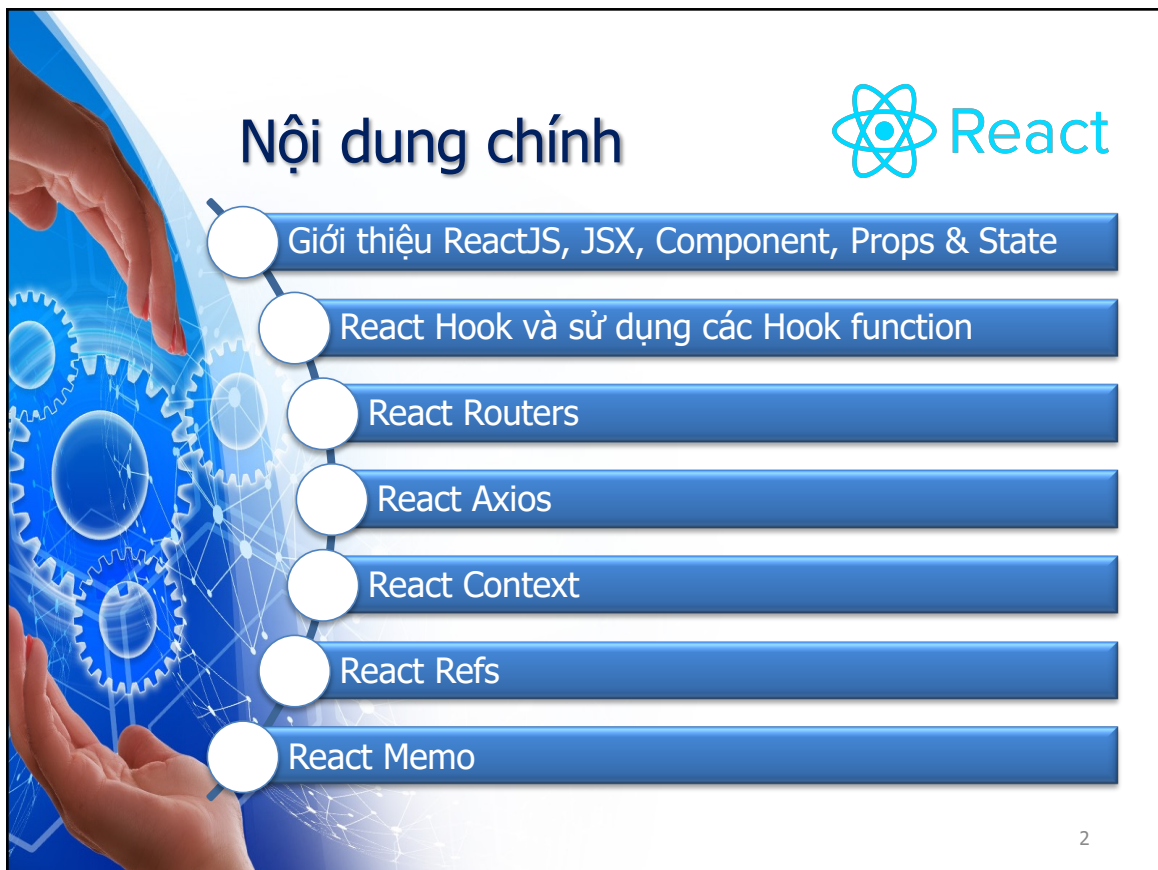




1



2



Giới thiệu ReactJS

- ReactJS là một **thư viện Javascript** được phát triển bởi **Facebook** cho phép phát triển các thành phần UI **dựa trên component** có khả năng **tái sử dụng cao**.
- ReactJS sử dụng cho cả **phía client** (client-side) và **phía server** (server-side).
- ReactJS **tập trung giải quyết phần View** trong mô hình MVC (Model-View-Controller).



Cài đặt môi trường

- Cài đặt NodeJS
 - <https://nodejs.org/en/download/>
- Cài đặt Yarn

```
npm install -g yarn
```
- Tạo ứng dụng ReactJS đầu tiên

```
yarn create react-app course-app
```
- Chạy ứng dụng

```
yarn start
```

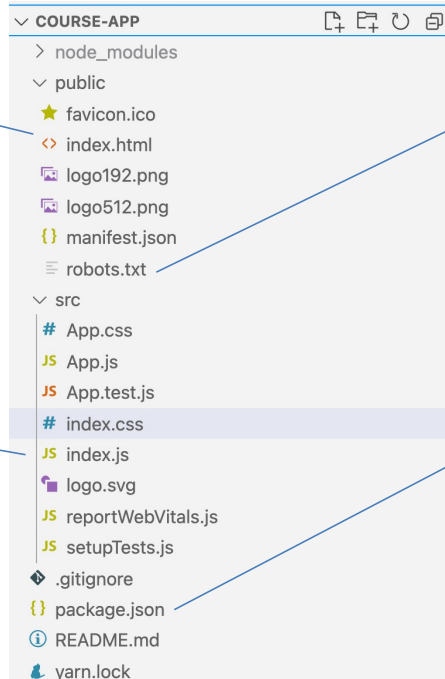
Cài đặt môi trường

index.html là trang HTML duy nhất

Chương trình bắt đầu từ index.js

Tập tin robots.txt nói cho crawler của search engine những tập tin nào được phép truy cập

Nằm trong thư mục gốc của project là tập tin cấu hình của npm, chứa thông tin ứng dụng, các dependency cần cài cho project.



Virtual DOM

- ReactJS sử dụng **DOM ảo** (virtual DOM) để cải thiện hiệu năng của các ứng dụng Web thay vì sử dụng DOM thật truyền thống.
- Virtual DOM có đầy đủ đặc tính của DOM.
- Khi một node trên Virtual DOM thay đổi, nó tìm node thay đổi bằng cách **so sánh sự khác nhau** giữa DOM và Virtual DOM, node tìm thấy sẽ được cập nhật mà **không ảnh hưởng** đến các node khác trên DOM.



JSX

- JSX (Javascript XML) cho phép **viết HTML bên trong Javascript** và chuyển các thẻ HTML thành các thành phần (component) của React.
- Biểu thức có thể được viết giữa cặp dấu ngoặc nhọn **{}**.
- Các thuộc tính các thẻ HTML trong JSX đặt theo quy tắc **lowerCamelCase**, ví dụ `className`.
- Ta có thể trả ra nhiều thành phần, nhưng chúng phải được **bọc (wrap) trong một thành phần container**.



JSX

- React **không bắt buộc** sử dụng JSX, nhưng JSX giúp **dễ dàng** xây dựng component hơn và thực thi **nhANH hơn** JS do nó đã được tối ưu khi chuyển thành mã nguồn JS.
- Các lỗi đều có thể phát hiện lúc biên dịch nên giúp chương trình **an toàn hơn**.
- Việc tập trung component giúp chương trình **dễ thiết kế template, dễ tái sử dụng, bảo trì và mở rộng**.



Component

- Trong ReactJS, view chia thành nhiều thành phần nhỏ khác nhau gọi là **component**.
- Component giúp việc **tái sử dụng** hiệu quả và **quản lý, bảo trì** được thực hiện dễ dàng hơn.
- Component giống như hàm **trả về các thành phần HTML**, có hai loại component là class component và function component.



Component

- Component dạng function

```
function MyComponent(props) {  
  return <h1>Hello {props.name}</h1>  
}
```

- Component dạng class

```
class MyComponent extends React.Component {  
  render() {  
    return <h1>Hello {this.props.name}</h1>  
  }  
}
```

```
let h = <MyComponent name="Thanh">  
ReactDOM.render(h, document.getElementById("root"))
```



Component

- Một component trong react phải kế thừa (extends) thành phần **React.Component** và bắt buộc có phương thức **render()** trả về HTML.
- Phương thức **constructor()** trong component sẽ được **gọi đầu tiên** khi gọi sử dụng component, các thuộc tính của component thường được khởi tạo trong phương thức này.
- Các component được **sử dụng tương tự như các thẻ** HTML thông thường.



Component

```
import React from 'react'

class UserLogin extends React.Component {
  render() {
    return (
      <div>
        <input type="text" placeholder="Username" />
        <input type="password" placeholder="Password" />
        <input type="button" value="Login"/>
      </div>
    )
  }
}

export default UserLogin
```



Fragment

- Trong các component khi trả ra nhiều thành phần HTML thì **phải bọc** trong một container như div. Tuy nhiên trong vài tình huống, điều này rất bất tiện cho lập trình viên.
- Sử dụng **React Fragment** cho phép **gom nhóm** các thành phần con mà **không cần sử dụng** node phụ container bọc lại.
- Hơn nữa, sử dụng Fragment giúp chương trình **thực thi nhanh hơn** và **tiết kiệm bộ nhớ**.



Fragment

- Ví dụ

```
class Hello extends React.Component {  
  render() {  
    return (  
      <React.Fragment>  
        <h1>Test 1</h1>  
        <h2>Test 2</h2>  
      </React.Fragment>  
    )  
  }  
}
```

Fragment

- Viết ngắn gọn, tuy nhiên cú pháp này **không cho khai báo thuộc tính** trong fragment.

```
class Hello extends React.Component {  
  render() {  
    return (  
      <>  
        <h1>Test 1</h1>  
        <h2>Test 2</h2>  
      </>  
    )  
  }  
}
```

Props

- Props là **các đối số được truyền vào** các component giống như các thuộc tính.
- Giá trị của props **tại component con không được thay đổi**.
- Nếu component có constructor thì props phải được truyền vào **constructor** và truyền cho React.Component bằng phương thức **super**.

```
class UserInfo extends React.Component {  
  constructor(props) {  
    super(props)  
  }  
}
```


- Component lấy danh sách user và sử dụng component UserList để hiển thị.

```
class UserInfo extends React.Component {  
  render() {  
    let users = [{  
      name: "Nguyen Van A",  
      country: "Vietnam"  
    }]  
    return (  
      <table>  
        {users.map(user => <UserList user={user} />)}  
      </table>  
    )  
  }  
}
```

Dương Hữu Thành

17

- Component hiển thị danh sách user

```
import React from 'react'  
class UserList extends React.Component {  
  render() {  
    return (  
      <tr>  
        <td>{this.props.user.name}</td>  
        <td>{this.props.user.country}</td>  
      </tr>  
    )  
  }  
}  
export default UserList
```

Dương Hữu Thành

18

18

Props

- Sử dụng cú pháp **spread** truyền props vào component.
- Ví dụ ta có component

```
class Welcome extends React.Component {  
  render() {  
    return (  
      <h1>Welcome {this.props.firstName}  
{this.props.lastName}</h1>  
    )  
  }  
}
```

Props

- Sử dụng cú pháp **spread** truyền props.

```
class DemoApp extends React.Component {  
  render() {  
    let data = {  
      firstName: 'Thanh',  
      lastName: 'Duong'  
    }  
    return (  
      <>  
        <Welcome {...data} />  
      </>  
    )  
  }  
}
```

State

- Đối tượng state **lưu giá trị các thuộc tính** của component.
- State **chỉ có hiệu lực trong phạm vi component**, giá trị state **có thể thay đổi** bất cứ khi nào bằng phương thức **this.setState()**.
- Khi đối tượng state thay đổi thì component sẽ **nạp lại** (re-render).
- Truy cập thuộc tính của đối tượng state:
 - **this.state.propertyName**

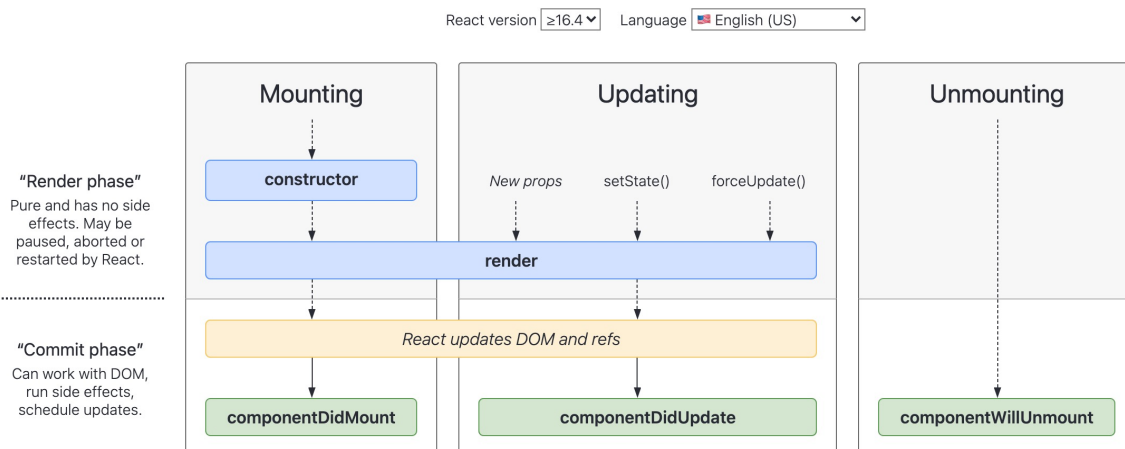
```
class DemoApp extends React.Component {
  constructor() {
    super()
    this.state = { "name": "" }
  }
  changeName = (event) => {
    this.setState({ "name": event.target.value })
  }
  render() {
    return (
      <div>
        <input type="text" value={this.state.name}
          onChange={this.changeName} />
        <h1>Hello {this.state.name}</h1>
      </div>
    )
  }
}
```

Hello Thanh D

Vòng đời React

- Mỗi component của React có 3 giai đoạn chính
 - Mounting: khi **đặt một component** vào DOM.
 - Updating: khi **component được update**. Một component được update khi có thay đổi trong props hoặc state.
 - Unmounting: khi một **component được gỡ** khỏi DOM.

Vòng đời React





Mounting

- Có 4 phương thức sẽ được gọi:
 - **constructor()** gọi khi component được tạo.
 - **render()** là phương thức bắt buộc và trả ra HTML cho DOM.
 - **static getDerivedStateFromProps()** gọi ngay trước khi render các thành phần trong DOM.
 - **componentDidMount()** gọi sau khi component được render.



Mounting

- Ví dụ

```
class Hello extends React.Component {  
  static getDerivedStateFromProps() {  
    console.log("Calling before rendering")  
  }  
  componentDidMount() {  
    console.log("Calling after rendering")  
  }  
  render() {  
    console.log("Calling for rendering")  
    return (  
      <h1>Test 1</h1>  
    )  
  }  
}
```




Updating

- Có 5 phương thức được gọi:
 - **static getDerivedStateFromProps()** gọi khi component được cập nhật.
 - **shouldComponentUpdate()** trả về giá trị boolean cho biết React có tiếp tục render hay không.
 - **render()** gọi khi component cập nhật render lại HTML cho DOM.
 - **getSnapshotBeforeUpdate()** được gọi trước khi props và state được cập nhật.
 - **componentDidUpdate()** gọi sau khi được cập nhật.



Unmounting

- React có một phương thức có sẵn được gọi khi component được unmounted:
 - **componentWillUnmount()**



React Hooks

- Hooks là đặc trưng mới hỗ trợ **từ phiên bản React 16.8** cho phép sử dụng state và các đặc trưng của React mà **không cần tạo class**.
- Hook là **các hàm** móc (hook) vào state và các đặc trưng trong vòng đời của React bằng component function.
- Hook cũng giống các hàm JS nhưng cần tuân thủ quy tắc: **chỉ gọi hook ở top level và từ các React function**.



Hook State

- Hook sử dụng **hàm useState()** để thiết lập và tìm kiếm state.

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

- Phương thức này trả về hai giá trị:
 - Giá trị 1: giá trị hiện tại của state.
 - Giá trị 2: hàm dùng để cập nhật state.

```
const [value, setValue] = useState(initValue)
```

Hook State

- Ví dụ

```
import React, {useState} from 'react'
export default function DemoApp() {
  const [number, setNumber] = useState(0.9999)
  return (
    <>
      <p>Random number: {number}</p>
      <button onClick={() => setNumber(
        Math.random().toFixed(4))}>Random</button>
    </>
  )
}
```

Hook State

- Hook `useState()` có thể sử dụng theo dõi giá trị chuỗi, số, luận lý, đối tượng hoặc các sự kết hợp giữa chúng.
- Ta có thể **tạo nhiều hook state** cho các giá trị riêng biệt

```
const Test = () => {
  const [name, setName] = useState("Thanh")
  const [year, setYear] = useState(2022)

  return <h1>{name} - {year}</h1>
}
```

Hook State

- Trường hợp hàm có quá nhiều state, ta có thể khai báo **một state cho một đối tượng**.

```
const Test = () => {  
  const [person, setPerson] = useState({  
    name: "Thanh",  
    year: 2021  
  })  
  ...  
  return (  
    <>  
      <h1>{person.name} - {person.year}</h1>  
      <button onClick={update}></button>  
    </>  
  )  
}
```

Hook State

- Chú ý khi cập nhật state thì **toàn bộ state sẽ được ghi đè**. Khi đó, cú pháp spread operator sẽ hữu ích:

```
const Test = () => {  
  ...  
  const update = () => {  
    setPerson(prev => {  
      return { ...prev, year: 2022 }  
    })  
  }  
  ...  
}
```



Hook Effect

- Hooks effect **thực hiện các hành động trên function component** mà không sử dụng các phương thức trong vòng đời của React component.
- Hooks effect **tương đương** với các phương thức `componentDidMount()`, `componentDidUpdate()`, `componentWillUnmount()`.
- Phương thức áp dụng: **`useEffect()`**



Hook Effect

- Hàm `useEffect()` có **hai tham số**, trong đó tham số thứ hai là tùy chọn.

```
useEffect(<function>, <dependency>)
```

- Nếu `<dependency>` là:
 - **một mảng rỗng** thì `useEffect()` chỉ chạy một lần đầu tiên khi `render()`.
 - **một giá trị** thì `useEffect()` chỉ được thay đổi khi giá trị này bị thay đổi.



Hook Effect

```
useEffect(() => {  
    // Được gọi mỗi khi render  
})
```

```
useEffect(() => {  
    // Được gọi lần đầu tiên khi render  
}, [])
```

```
useEffect(() => {  
    // Được gọi lần đầu tiên khi render  
    // và mỗi khi prop, state trong tham số thứ hai  
    // thay đổi  
}, [prop, state])
```



Hook Effect

```
import React, {useState, useEffect} from 'react'  
export default function DemoApp() {  
    const [number, setNumber] = useState(0.9999)  
    useEffect(() => {  
        console.log(number)  
    })  
    return (  
        <>  
            <p>Random number: {number}</p>  
            <button onClick={() => setNumber(  
                Math.random().toFixed(4))}>Random</button>  
        </>  
    )  
}
```



Hook Effect

- Một vài xử lý effect **cần giải phóng** khi không sử dụng nữa để giảm bớt rò rỉ bộ nhớ (leak memory).
- Để làm được điều này, ta **trả về một hàm ở cuối** của useEffect().



Hook Effect

- Ví dụ

```
function Timer() {  
  const [count, setCount] = useState(0)  
  
  useEffect(() => {  
    let timer = setTimeout(() => {  
      setCount(count => count + 1)  
    }, 1000)  
  
    return () => clearTimeout(timer)  
  }, [])  
  
  return <h1>{count}</h1>  
}
```



Tạo hook tùy chỉnh

- Khi ta có một xử lý nào đó được sử dụng ở nhiều component thì ta có thể tạo một hook function để có thể **dễ dàng tái sử dụng** hiệu quả.
- Quy ước tên hook **bắt đầu bằng "use"**, chẳng hạn `useData`, `useFetch`.



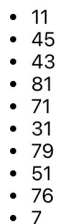
Tạo hook tùy chỉnh

```
const useRange = (n) => {  
  const [data, setData] = useState([])  
  
  useEffect(() => {  
    setData(() => {  
      let d = []  
      for (let i = 0; i < n; i++)  
        d.push(parseInt(Math.random() * 100))  
  
      return d;  
    })  
  }, [n])  
  
  return [data]  
}
```

Tạo hook tùy chỉnh

- Sử dụng hook `useRange()`

```
const App = () => {  
  const [data] = useRange(10)  
  
  return (  
    <ul>{data.map(v => <li>{v}</li>)}</ul>  
  )  
}
```



- 11
- 45
- 43
- 81
- 71
- 31
- 79
- 51
- 76
- 7

React Events

- Tên các sự kiện trong React được đặt theo quy tắc **lowerCamelCase** như `onClick`, `onChange`.
- Ví dụ

```
const Home = () => {  
  const show = () => {  
    alert("Welcome!!!")  
  }  
  return (  
    <button onClick={show}>Click me!!!</button>  
  )  
}
```



React Events

- Truyền đối số vào hàm xử lý sự kiện

```
const Home = () => {  
  const hello = (name) => {  
    alert("Hello " + name)  
  }  
  
  return (  
    <button onClick={() => hello("Thanh")}>  
      Click me!!!  
    </button>  
  )  
}
```



React Forms

- Trong React, dữ liệu form thường được xử lý bởi các component khi tất cả dữ liệu lưu trong **state**.
- Ta có thể truy cập giá trị các trường bằng lệnh:
 - **event.target.value**


```
const Home = () => {
  const [name, setName] = useState("")

  const inputChange = (event) => {
    setName(event.target.value)
  }

  const submit = (event) => {
    event.preventDefault()
    alert(`Hello ${name}`)
  }

  return (
    <form onSubmit={submit} >
      <input value={name} onChange={inputChange} />
      <input type="submit" value="Hello" />
    </form>
  )
}
```



React CSS

- Trong React, CSS thường được viết trong component, ta có thể tạo các đối tượng chứa thông tin thiết lập CSS và gán cho thuộc tính style của các thành phần HTML.
- Chú ý: tên thuộc tính CSS viết trong component phải dùng theo quy tắc **lowerCamelCase**, chẳng hạn thuộc tính **background-color** phải viết là **backgroundColor**.

- Ví dụ

```
const Hello = () => {  
  const myStyle = {  
    color: "white",  
    backgroundColor: "darkblue"  
  }  
  return (  
    <>  
      <h1 style={{color:"red"}}>Test 1</h1>  
      <h2 style={myStyle}>Test 2</h2>  
    </>  
  )  
}
```

- Ta cũng có thể **viết CSS trong tập tin .css riêng** biết và import vào component.

```
import React from 'react'  
import './Style.css'  
const Hello = () => {  
  return (  
    <>  
      <h1>Test 1</h1>  
    </>  
  )  
}
```

Style.css

```
h1 {  
  font-family: Tahoma;  
  font-size: 48px;  
  font-weight: bold;  
  color: blue  
}
```



React Bootstrap

- Cách đơn giản nhất là **sử dụng CDN**, sử dụng thẻ `<link>` trong `<head>` của **index.html** để chèn Bootstrap CDN vào.
- Import JS liên quan bằng thẻ `<script>` nếu cần.
- Lấy thông tin Bootstrap CDN tại getbootstrap: <https://getbootstrap.com/>



React Bootstrap

- React Bootstrap thay thế cho Bootstrap javascript, các **component của bootstrap được xây dựng thành các React component không cần** một số thư viện liên quan như jQuery.
- Cài đặt

```
npm install react-bootstrap bootstrap@4.6.0
```

- Import component sử dụng cú pháp như sau:

```
import { Container, Row, Col } from "react-bootstrap"
```



React Bootstrap

- Ví dụ sử dụng hệ thống lưới của bootstrap

```
<Container fluid>
  <Row>
    <Col xs={12} md={8}>xs=12 md=8</Col>
    <Col xs={6} md={4}>xs=6 md=4</Col>
  </Row>
  <Row>
    <Col xs={6}>xs=6</Col>
    <Col xs={6}>xs=6</Col>
  </Row>
</Container>
```

Email address

Password

React Bootstrap

- Ví dụ sử dụng form

```
<Form>
  <Form.Group controlId="email">
    <Form.Label>Email address</Form.Label>
    <Form.Control type="email" />
  </Form.Group>
  <Form.Group controlId="password">
    <Form.Label>Password</Form.Label>
    <Form.Control type="password" />
  </Form.Group>
  <Button variant="primary" type="submit">
    Submit
  </Button>
</Form>
```

React Bootstrap

```
const CommonModal = (props) => {
  return (
    <Modal show={props.show}>
      <Modal.Header>
        <Modal.Title>WELCOME</Modal.Title>
      </Modal.Header>
      <Modal.Body>
        <h1>WELCOME TO OUR MODAL</h1>
      </Modal.Body>
      <Modal.Footer>
        <Button variant="info"
          onClick={props.toggle}>Thoát</Button>
      </Modal.Footer>
    </Modal>
  )
}
```

Dương Hữu Thành

55

React Bootstrap

```
const Parent = () => {
  • const [show, setShow] = useState(false)

  const toggleModal = (event) => {
    setShow(s => !s)
  }

  return (
    <>
      <Button onClick={toggleModal}
        variant="info">Show Modal</Button>
      <CommonModal show={show}
        toggle={toggleModal} />
    </>
  )
}
```

Dương Hữu Thành

56

56

React Font Awesome

- Cài đặt

```
npm install --save @fortawesome/fontawesome-svg-core
npm install --save @fortawesome/free-solid-svg-icons
npm install --save @fortawesome/react-fontawesome
npm install --save @fortawesome/free-brands-svg-icons
npm install --save @fortawesome/free-regular-svg-icons
```

- Import một số icon

```
import { FontAwesomeIcon }
  from '@fortawesome/react-fontawesome'
import { faApple }
  from '@fortawesome/free-brands-svg-icons'
import { faSmile }
  from '@fortawesome/free-solid-svg-icons'
```

Dương Hữu Thành

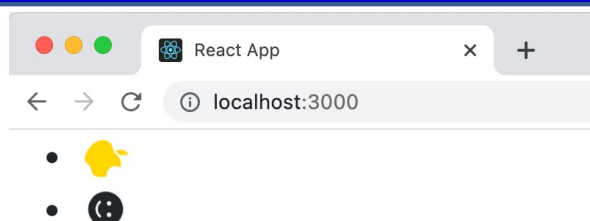
57

57

React Font Awesome

- Sử dụng

```
<ul>
  <li><FontAwesomeIcon icon={faApple}
    color="gold"
    size="lg" spin /></li>
  <li><FontAwesomeIcon icon={faSmile}
    size="12px"
    rotation="90" /></li>
</ul>
```



Dương Hữu Thành

58

58



React animate.css

- Cài đặt

```
npm install --save animate.css
```

- Sử dụng cơ bản

```
import 'animate.css/animate.css'
function Demo() {
  return (
    <div className="animate__animated animate__bounce">
      </div>
    )
  }
}
```



React Router

- React Router dùng chủ yếu phát triển các ứng dụng Web dạng **Single Page**. Nó cho phép **chuyển qua lại giữa các component mà không cần nạp lại trang**. React có 3 gói cho routing
 - **react-router**: cung cấp các thành phần routing chính cho ứng dụng.
 - **react-router-native**: dùng cho các ứng dụng di động.
 - **react-router-dom**: dùng cho các ứng dụng Web.



React Router

- Trong tài liệu này sử dụng React Router V6
- Cài đặt

```
npm install react-router-dom@6
```

- **BrowserRouter**: xử lý các URL động.
- **Route**: chỉ định các component được render dựa trên path chỉ định.
- **Routes**: chứa các Route để render các component khi path khớp.



React Router

- **Link**: tạo liên kết đến URL chỉ định nhưng không nạp lại trang.
- **NavLink**: tương tự Link nhưng cho phép thêm style cho active link.
- **Redirect**: dùng chuyển hướng đến route khác trong ứng dụng, nhưng vẫn duy trì URL cũ.

React Router

- Ví dụ: import các component trong App.js

```
import {
  BrowserRouter,
  Route,
  Routes,
  Link
} from 'react-router-dom';
import Home from './Home'
import Category from './Category'
```

React Router

```
const App = () => {
  return (
    <BrowserRouter>
      <ul>
        <li><Link to="/home">Home</Link></li>
        <li><Link to="/category">Category</Link></li>
      </ul>
      <Routes>
        <Route path="/home" element={<Home />} />
        <Route path="/category" element={<Category />} />
        <Route path="/admin/*" element={<Admin />} />
      </Routes>
    </BrowserRouter>
  )
}
```

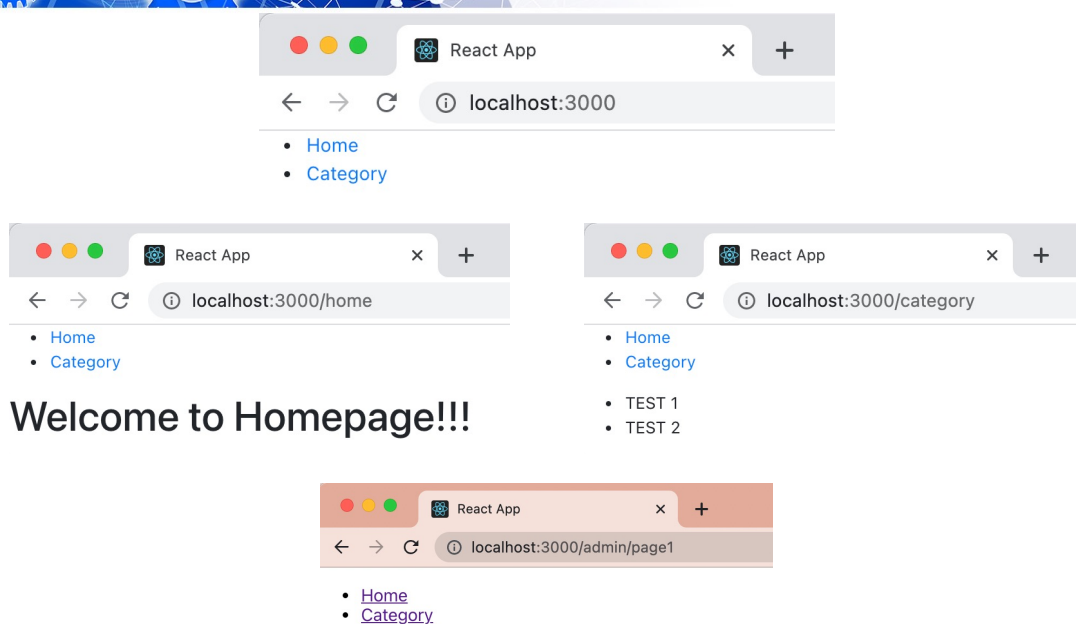
React Router

```
const Home = () => {  
  return (  
    <h1>Welcome to Homepage!!!</h1>  
  )  
}
```

```
const Category = () => {  
  return (  
    <ul>  
      <li>TEST 1</li>  
      <li>TEST 2</li>  
    </ul>  
  )  
}
```

```
const Admin = () => {  
  return <h1>Admin Pages</h1>  
}  
  
export default Admin
```

React Router



Nested Routes

- Các route có thể lồng nhau (nested routes)

```
const App = () => {  
  return (  
    <BrowserRouter>  
      <Routes>  
        <Route path="/admin" element={<Admin />}>  
          <Route path="product" element={<AdminProduct />} />  
          <Route path="stats" element={<AdminStats />} />  
        </Route>  
      </Routes>  
    </BrowserRouter>  
  )  
}
```

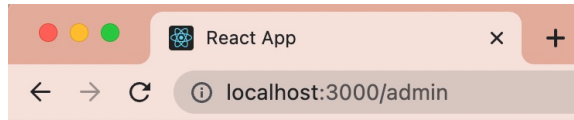
Nested Routes

```
import { Outlet } from "react-router-dom"  
  
const Admin = () => {  
  return <>  
    <h1>Admin Pages</h1>  
    <Outlet />  
  </>  
}
```

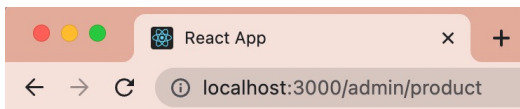
```
const AdminProduct = () => {  
  return <h1>Admin Product</h1>  
}
```

```
const AdminStats = () => {  
  return <h1>Admin Stats</h1>  
}
```


Nested Routes

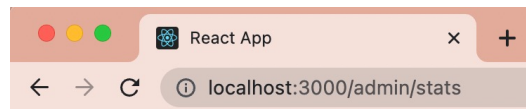


Admin Pages



Admin Pages

Admin Product



Admin Pages

Admin Stats

useParams và useSearchParams

- Định nghĩa router **có tham số**.
 - **useParams()** đọc tham số trên đường dẫn.
 - **useSearchParams()** đọc tham số truy vấn.

```
const App = () => {  
  return (  
    <BrowserRouter>  
      <Routes>  
        <Route path="/category/:categoryId"  
          element={<CategoryDetail />} />  
      </Routes>  
    </BrowserRouter>  
  )  
}
```

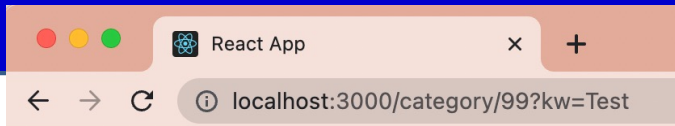
useParams và useSearchParams

- Lấy giá trị tham số trong CategoryDetail

```
import { useParams, useSearchParams } from "react-router-dom"

const CategoryDetail = () => {
  let { categoryId } = useParams()
  let [q] = useSearchParams()

  return <ul>
    <li>CateId: {categoryId}</li>
    <li>Keyword: {q.get("kw")}</li>
  </ul>
}
```



- Cateld: 99
- Keyword: Test

Dương Hữu Thành

71

useNavigate

- Hook **useNavigate()** truy cập đối tượng navigate.

```
import { useNavigate } from "react-router-dom"

const Home = () => {
  const nav = useNavigate()

  const go = () => nav("/category/99")

  return (
    <button onClick={go}>Go to details</button>
  )
}
```

Dương Hữu Thành

72

72



React Axios

- Axios là một **HTTP client** làm việc dựa trên promise cung cấp các API dễ dàng sử dụng cho phía trình duyệt và NodeJS.
- Cài đặt react axios
 - Dùng npm

```
npm install axios
```

- Dùng yarn

```
yarn add axios
```



React Axios

- Sử dụng axios trong React.
- Tạo API.js

```
import axios from 'axios'

export let endpoints = {
  'categories': '/categories/',
  'login': '/o/token/'
}

export default axios.create({
  baseURL: "https://localhost:8000"
})
```

React Axios

```
const DemoApp = () => {
  const [cates, setCates] = useState([])

  useEffect(() => {
    const loadCates = async () => {
      let res = await API.get(endpoints["categories"])
      setCates(res.data)
    }

    loadCates()
  }, [])

  return <ul>
    {cates.map(it => <li>{it.name}</li>)}
  </ul>
}
```

React Axios

- Cài đặt React Axios

```
npm install react-axios
```

- Sử dụng React Axios

```
<Request instance={axios.create({})}
  method="" url="" data={}
  params={} config={} debounce={200}
  debounceImmediate={true}
  isReady={true} onSuccess={(response)=>{}}
  onLoading={()=>{}} onError=(error)=>{ } />
```



React Axios

- **instance**: thể hiện của axios.
- **url** (bắt buộc): endpoint API.
- **method** (bắt buộc): phương thức HTTP request.
- **data**: dữ liệu trong request body.
- **params**: dữ liệu trên query string.
- **config**: thông tin cầu hình cho axios.
- **debounce**: khoảng thời gian tối thiểu giữa các sự kiện request.



React Axios

- **onSuccess**: sự kiện được gọi khi request thực hiện thành công.
- **onError**: sự kiện được gọi khi request thực hiện có lỗi.
- **onLoading**: thực hiện khi bắt đầu request.



React Cookies

- React-cookies cho phép ta dễ dàng tương tác với cookie với React.
- Cài đặt

```
npm install react-cookies --save
```

- Import sử dụng

```
import cookies from 'react-cookies'
```



React Cookies

- **cookies.load(name)**: nạp giá trị một cookie.
- **cookies.loadAll()**: nạp tất cả các cookies.
- **cookies.select([regex])**: tìm cookie có tên khớp với một biểu thức chính quy.
- **cookies.save(name, value, [options])**: lưu một cookie.
- **remove(name, [options])**: xóa một cookie.



React Cookies

- Một số option quan trọng khi lưu/xoá cookie:
 - **path** → string: đường dẫn được truy cập cookie.
 - **expires** → object: thời điểm hết hạn cookie
 - **maxAge** → number:
 - **domain** → string: tên miền cho cookie.
 - **secure** → boolean: thiết lập true nếu muốn cookie chỉ được truy cập qua https.
 - **httpOnly** → boolean: thiết lập true nếu muốn cookie chỉ được truy cập trên server.



React Cookies

- Đăng nhập và lưu access token trong cookie

```
API.post(endpoints["login"], {
  'username': this.state.username,
  'password': this.state.password,
  'grant_type': 'password',
  'client_id': '...',
  'client_secret': '...'
}).then(res => {
  cookies.save("access-token",
    res.data.access_token)
  this.props.history.push("/")
}).catch(err => console.error(err))
```



React Context

- Khi sử dụng props để truyền dữ liệu qua **nhiều cấp** component con sẽ **khá phức tạp**.
- React Context là cách **thức quản lý state toàn cục**, thường sử dụng kết hợp useState chia sẻ state giữa các component lồng nhau (nested component) giúp dễ dàng **truyền dữ liệu trong toàn ứng dụng**.
- Sử dụng hàm **createContext** tạo context và **Context Provider** bọc các component con sử dụng state của context.



React Context

```
import { createContext, useState } from "react"

export const MyContext = createContext()

const Parent = () => {
  const [name, setName] = useState("Thanh")
  return (
    <MyContext.Provider value={name}>
      <h1>Hello {name} (Parent)</h1>
      <Child1 />
    </MyContext.Provider>
  )
}

const Child1 = () => {
  return <Child2 />
}
```

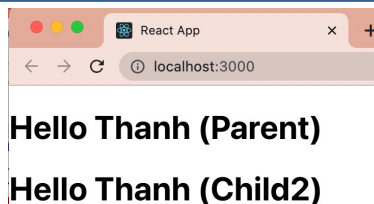
React Context

- Trong các component con truy cập sử dụng context thông qua hook **useContext**.

```
import { useContext } from "react"

const Child2 = () => {
  const name = useContext(MyContext)

  return <h1>Hello {name} (Child2)</h1>
}
```



useReducer

- Hook useReducer sử dụng tương tự useState cho phép tùy chỉnh logic của state. Hook này có hai tham số

```
useReducer(<reducer>, initState)
```

- Hàm reducer chứa logic của state.
- Giá trị <initState> khởi động cho state.
- Hook này trả về trạng thái hiện tại và phương thức dispatch.

useReducer

```
import { useReducer } from "react"

const DemoApp = () => {
  const [value, dispatch] = useReducer(reducer, 'Unknown')

  return (
    <div>
      <h1>{value}</h1>
      <button onClick={() => dispatch(helloViAction())}>
        Vietnamese
      </button>
      <button onClick={() => dispatch(helloEnAction())}>
        English
      </button>
    </div>
  )
}
```

React Refs

- Hook useRef cho phép duy trì các giá trị giữa các lần render.
- Ta có thể sử dụng nó để
 - lưu trữ các giá trị có thể thay đổi (mutal value) để không render lại component khi thay đổi chúng.
 - truy cập trực tiếp vào thành phần DOM.

React Refs

```
const App = () => {
  const [v, setV] = useState(0)
  const count = useRef(0)

  useEffect(() => {
    count.current = count.current + 1
  })

  return (
    <>
      <input value={v}
        onChange={event => setV(event.target.value)} />
      <h1>Render count: {count.current}</h1>
    </>
  )
}
```

89

Dương Hữu Thành

89

```
const App = () => {
  const name = useRef()

  const hello = () => {
    if (name.current.value === '')
      name.current.style.border = '1px solid red'
    else {
      name.current.style.border = ''
      alert('Hello ' + name.current.value)
    }
  }

  return (
    <>
      <input ref={name} />
      <button onClick={hello}>Show</button>
    </>
  )
}
```

90

Dương Hữu Thành

90

- Minh họa upload tập tin

```
const Register = () => {  
  const avatar = useRef()  
  register = (event) => { ... }  
  return (  
    <form onSubmit={register}>  
      <input type="file" ref={avatar} />  
      <input type="submit" value="Đăng ký" />  
    </form>  
  )  
}
```

- Phương thức register

```
register = (event) => {  
  event.preventDefault()  
  
  const form = new FormData()  
  form.append('avatar', avatar.current.files[0])  
  
  API.post('/upload', form, {  
    headers: { 'Content-Type': 'multipart/form-data' }  
  }).then(res => {  
    console.info(res)  
  }).catch(err => console.error(err))  
}
```


React Memo

- Sử dụng memo báo React **bỏ qua render** lại một component khi props của nó không thay đổi.
- Nó có thể **giúp cải thiện hiệu năng** nếu đó là các component phức tạp. Ví dụ component sau:

```
const ToDoItems = ({items}) => {  
  console.info(Math.random())  
  return (  
    <>  
      {items.map((item) => <h1>{item}</h1>)}  
    </>  
  )  
}  
export default ToDoItems
```

Dương Hữu Thành

93

93

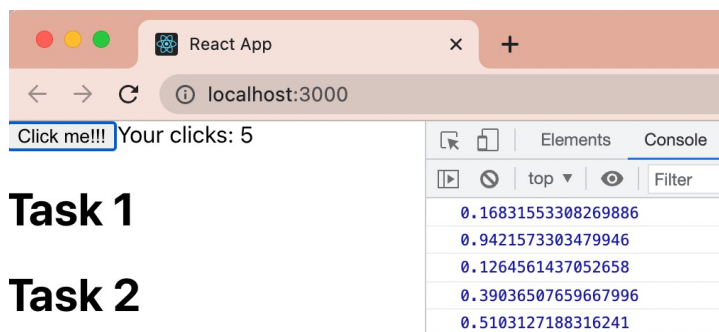
React Memo

```
const App = () => {  
  const [count, setCount] = useState(0)  
  const [items, setItems] = useState(["Task 1", "Task 2"])  
  
  return (  
    <>  
      <div>  
        <button value={count}  
          onClick={() => setCount(c => c + 1)}>  
          Click me!!!</button>  
        Your clicks: {count}  
      </div>  
      <ToDoItems items={items} />  
    </>  
  )  
}  
export default App
```

94

React Memo

- Khi click vào button thì component `ToDoItems` cũng **bị render lại** mặc dù **props của nó không thay đổi**.



Task 1

Task 2

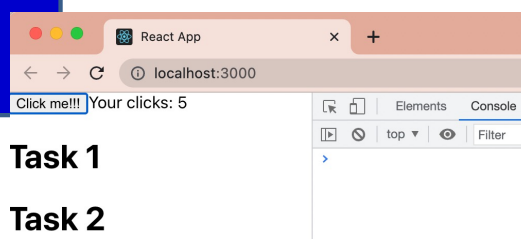
React Memo

- Ta giải quyết vấn đề bằng cách **sử dụng memo** khi export component.
- Chạy lại chương trình và bấm nút "Click me!!!" thì component `ToDoItems` không bị render lại.

```
import { memo } from "react"

const ToDoItems = ({items}) => {
  ...
}

export default memo(ToDoItems)
```



Task 1

Task 2

- Bổ sung component `ToDoItems` như sau:

```
import { memo } from "react"

const ToDoItems = ({items, addItem}) => {
  console.info(Math.random())
  return (
    <>
      {items.map((item) => <h1>{item}</h1>)}
      <button onClick={addItem}>Add Item</button>
    </>
  )
}

export default memo(ToDoItems)
```

- Bổ sung component `App` như sau:

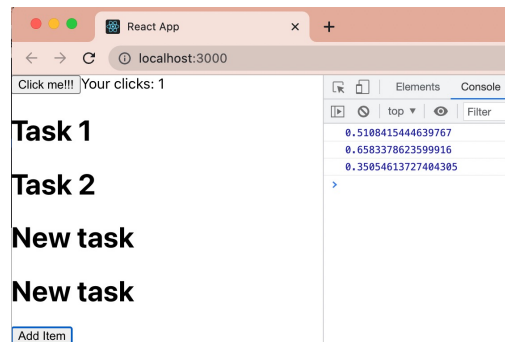
```
const App = () => {
  ...
  const addItem = () => {
    setItems((items) => [...items, "New task"])
  }

  return (
    <>
      ...
      <ToDoItems items={items} addItem={addItem} />
    </>
  )
}
```

useCallback

- Khi đó click vào button "Click me!!!", ta thấy component `ToDoItems` vẫn bị render lại do khi component nạp lại thì hàm `addItem()` cũng được tạo lại.

→ Sử dụng Hook `useCallback`



useCallback

- Hook `useCallback()` trả về hàm callback nhớ (**memorized callback function**) và hạn chế việc tạo lại hàm khi không cần thiết. Nó chỉ chạy khi một trong các dependency của nó cập nhật.
- Ta sử dụng hàm `addItem` trong component `App`

```
const App = () => {  
  ...  
  const addItem = useCallback(() => {  
    setItems((items) => [...items, "New task"])  
  }, [items])  
  ...  
}
```



useMemo

- Hook **useMemo** trả về giá trị nhớ (**memorized value**), hook này chỉ được chạy khi một trong các dependency của nó được cập nhật.
- Các hook **useMemo** và **useCallback** giống nhau, chỉ khác ở điểm **useMemo** trả về giá trị nhớ, còn **useCallback** trả về hàm nhớ.



useMemo

- Ví dụ ta có hàm như sau:

```
const expensiveCal = (n) => {  
  console.info(Math.random())  
  
  let sum = 0  
  for (let i = 1; i <= n; i++)  
    sum += i  
  
  return sum  
}
```

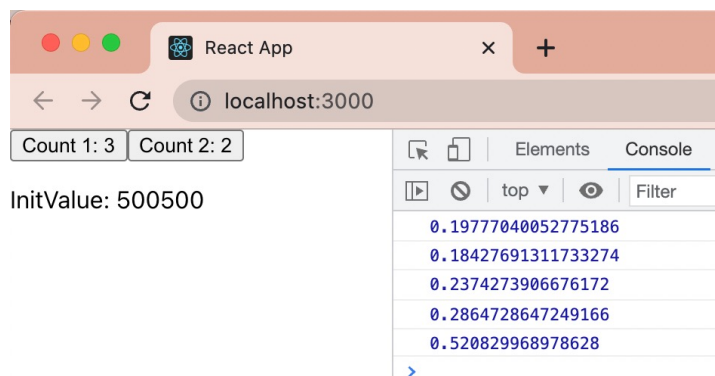
useMemo

```
const App = () => {
  const [count1, setCount1] = useState(0)
  const [count2, setCount2] = useState(0)
  const initValue = expensiveCal(1000)

  return (
    <>
      <button onClick={() => setCount1(c => c + 1)}
        value={count1}>Count 1: {count1}</button>
      <button onClick={() => setCount2(c => c + 1)}
        value={count2}>Count 2: {count2}</button>
      <p>InitValue: {initValue}</p>
    </>
  )
}
```

useMemo

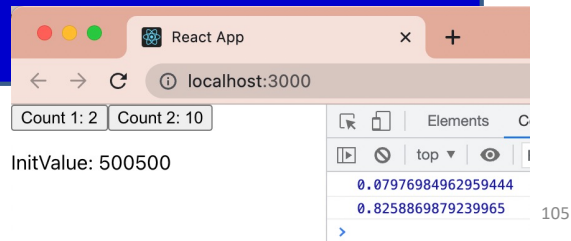
- Ta nhận xét khi click vào nút thứ 1 hay thứ 2 thì hàm expensiveCal() cũng **bị gọi lại**.
- Điều này có thể ảnh hưởng hiệu năng nếu hàm này phức tạp.



useMemo

- Để giải quyết vấn đề này ta sử dụng useMemo, giả sử ta mong muốn hàm chỉ được gọi khi count1 có thay đổi.

```
import { useMemo } from 'react';
const App = () => {
  ...
  const initValue = useMemo(()
    => expensiveCal(1000), [count1])
  ...
}
```



Dương Hữu Thành

105

Q&A

ThS. Dương Hữu Thành,
Khoa CNTT, Đại học Mở TP.HCM,
thanh.dh@ou.edu.vn.

106

106