

State & Lifecycle method

Mục tiêu

- Giới thiệu State và cách sử dụng
- Xử lý sự kiện trong ReactJS
- Cơ chế render có điều kiện

State trong ReactJS

Giới thiệu

- State là nơi bạn lưu trữ các giá trị thuộc tính thuộc về component
- Khi state thay đổi, thì components sẽ render lại.

State trong ReactJS

State là gì ?

- State là một object được sử dụng để chứa dữ liệu hoặc thông tin về components, từ đó bạn có thể luân chuyển dữ liệu đến các thành phần trong Component hoặc các Component khác.
- State có thể được thay đổi bất cứ khi nào mong muốn
- Trong các dự án React, state được dùng để phản hồi các yêu cầu từ người dùng, hay lưu trữ một dữ liệu nào đó trong components.

Thao tác với state trong ReactJS

#App.jsx

```
import React from 'react';
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = { header: "Header from state...", content: "Content from state..." }
    render() {return ( <div>
                                <h1>{this.state.header}</h1>
                                <h2>{this.state.content}</h2>
                                </div>); }
  }
export default App;
```

Thao tác với state trong ReactJS

#main.js

```
import React from 'react';
```

```
import ReactDOM from 'react-dom';
```

```
import App from './App.jsx';
```

```
ReactDOM.render(<App />, document.getElementById('app'));
```

Thao tác với state trong ReactJS

Qua ví dụ trên ta sẽ phân tích cách khởi tạo State:

Ta sẽ khởi tạo một state bằng cách gán giá trị cho biến `this.state`

```
this.state = {  
  header: "Header from state...",  
  content: "Content from state..."  
}
```

Và lấy giá trị state bằng `this.state`

```
{this.state.header}  
{this.state.content}
```

Cập nhật State

- State không bao giờ được cập nhật một cách rõ ràng.
- React sử dụng một đối tượng có thể quan sát làm trạng thái quan sát những thay đổi nào được thực hiện đối với trạng thái và giúp thành phần hoạt động tương ứng.
- Ví dụ: nếu cập nhật trạng thái của bất kỳ thành phần nào, trang web sẽ không tự hiển thị lại vì React State sẽ không thể phát hiện các thay đổi được thực hiện.

Cập nhật State

- React cung cấp phương thức `setState ()` của riêng nó. Phương thức `setState ()` nhận một tham số duy nhất và một đối tượng chứa giá trị được cập nhật.
- Sau khi cập nhật xong, phương thức này ngầm gọi phương thức `render ()` để render lại trang.

Cập nhật State

Ví dụ :

```
class Car extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { brand: "Ford", model: "Mustang", color: "red",  
      year: 1964};  
  }  
  changeColor = () => { this.setState({color: "blue"}); }
```

Cập nhật State

```
render() {  
  return ( <div>  
    <h1>My {this.state.brand}</h1>  
    <p> It is a {this.state.color}  
      {this.state.model} from {this.state.year}.  
    </p>  
    <button type="button" onClick={this.changeColor}>  
      Change color</button>  
    </div>  
  );  
}
```

Thao tác với state trong ReactJS

Một số lưu ý khi sử dụng State

- Để việc sử dụng state được dễ dàng và tránh gây ra lỗi ngoài ý muốn , một số lưu ý như sau:
- Bạn nên để cấu trúc dữ liệu của state đơn giản nhất có thể, không nên tạo cấu trúc quá lằng nhằng sẽ khó thao tác và ảnh hưởng hiệu năng
- Không thay đổi state một cách trực tiếp
- State được update không đồng bộ nên bạn cần lưu ý khi sử dụng hàm setState cần giá trị từ state trước

Tổng kết

Qua bài viết này chúng ta đã tìm hiểu:

- Hiểu được khái niệm State
- Thao tác tạo State, cập nhật State
- Một số lưu ý khi sử dụng State

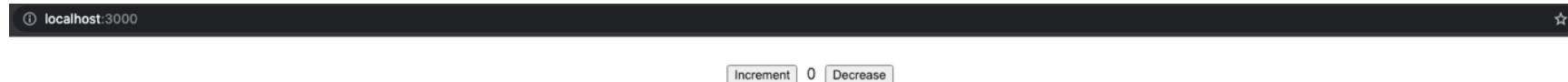
[Thực hành] Đếm số lần click vào Button

Mục tiêu

- Tạo được dự án React JS
- Thao tác với state thông qua event

Mô tả

Viết chương trình cho phép tăng hoặc giảm số thông qua các nút



Xử lý sự kiện trong Reactjs

Giới thiệu

Trong một website việc tương tác giữa người dùng là điều không thể thiếu như click, nhập form,..chúng ta có thể thực hiện bắt các sự kiện này trong React một cách dễ dàng.

Xử lý sự kiện trong Reactjs

Xử lý các sự kiện trong React rất giống với xử lý các sự kiện trên các phần tử DOM. Có một số khác biệt

- Các sự kiện React được đặt tên bằng camelCase, thay vì chữ thường. Ví dụ: onclick -> onClick, onchange -> onChange
- Trong Javascript bạn sẽ xử lý sự kiện trong một hàm, còn trong React bạn sẽ xử lý sự kiện trong một phương thức của Component.

Xử lý sự kiện trong Reactjs

Ví dụ:

```
<button onClick={changeName}>
```

Change Name

```
</button>
```

Trong React, chúng ta không thể trả về false để ngăn chặn hành vi mặc định. Chúng ta phải gọi sự kiện PreventDefault một cách rõ ràng để ngăn chặn hành vi mặc định.

Xử lý sự kiện trong Reactjs

Ví dụ:

```
function ActionLink() {  
  function handleClick(e) {  
    e.preventDefault();  
    console.log('You had clicked a Link.');  }  
  return (  
    <a href="#" onClick={handleClick}>  
      Click_Me  
    </a>  
  );  
}
```

This trong React

- Đối với các phương thức trong React, từ khóa this đại diện cho component sở hữu phương thức.
- Bạn phải cẩn thận về ý nghĩa của this trong những callback JSX. Trong JavaScript, những phương thức của class mặc định không bị ràng buộc.
- Nếu bạn quên ràng buộc và truyền nó vào onClick, this sẽ có giá trị là undefined khi phương thức này được thực thi.
- Đó là lý do tại sao bạn nên sử dụng các hàm arrow function. Với các hàm arrow function, điều này sẽ luôn đại diện cho đối tượng đã xác định.

This trong React

Ví dụ:

```
class Football extends React.Component {  
  shoot = () => { alert(this);  
    /*Từ khóa 'this' đề cập đến đối tượng thành phần*/  
  }  
  render() {  
    return (<button onClick={this.shoot}>Take the shot!</button>);  
  }  
}  
  
ReactDOM.render(<Football />, document.getElementById('root'))
```

Truyền đối số

Nếu bạn muốn gửi các tham số vào một trình xử lý sự kiện, bạn có hai tùy chọn:

- Tạo một arrow function ẩn danh

Ví dụ:

```
class Football extends React.Component {  
  shoot = (a) => { alert(a);  
  }  
  render() {  
    return (<button onClick={() => this.shoot("Goal")}>Take the  
shot!</button>);  
  }  
}  
ReactDOM.render(<Football />, document.getElementById('root'));
```

Truyền đối số

- Hoặc ràng buộc trình xử lý sự kiện với this

Ví dụ:

```
class Football extends React.Component {  
  shoot(a) { alert(a);}  
  render() {  
    return (  
      <button onClick={this.shoot.bind(this, "Goal")}>  
        Take the shot!</button>;  
    )  
  }  
}  
  
ReactDOM.render(<Football />, document.getElementById('root'));
```

Truyền đối số

- Lưu ý:

Đối với trường hợp 2, nếu bạn gửi đối số mà không sử dụng phương thức liên kết, (`this.shoot (this, "Goal")`) thay vì `this.shoot.bind (this, "Goal")`)

Hàm sẽ được thực thi khi trang được tải thay vì đang đợi nút được bấm.

Tổng kết

Qua bài viết này chúng ta đã tìm hiểu:

- Sự kiện trong React
- This trong React
- Truyền đối số vào trình xử lý sự kiện

Component Life Cycle trong ReactJS

Giới thiệu

- Mỗi thành phần React đều có một vòng đời của riêng nó,
- Vòng đời của một thành phần có thể được định nghĩa là một loạt các phương thức được gọi trong các giai đoạn khác nhau của sự tồn tại của thành phần đó

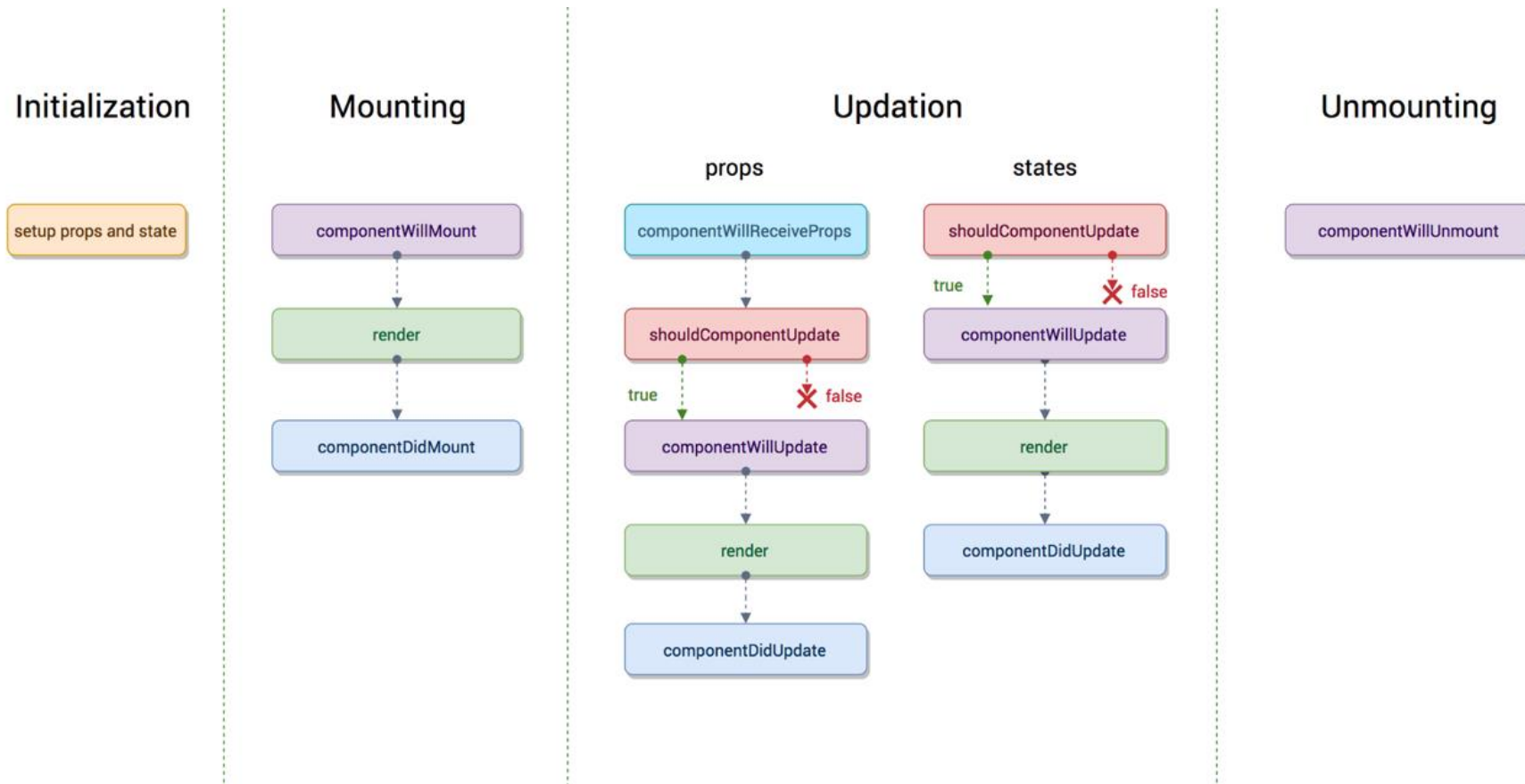
Component Life Cycle

Một thành phần React có thể trải qua bốn giai đoạn trong vòng đời của nó như sau.

- **Initialization:** Đây là giai đoạn mà thành phần được xây dựng với các Props đã cho và trạng thái mặc định. Điều này được thực hiện trong phương thức khởi tạo Component Class
- **Mounting:** Giai đoạn này được thực hiện sau khi quá trình initialization(khởi tạo) được hoàn thành.
Nó thực hiện nhiệm vụ chuyển virtual DOM (DOM ảo) trong React thành DOM và hiển thị trên trình duyệt.
- **Updating:** là giai đoạn trạng thái của một thành phần được cập nhật và ứng dụng được render lại.

Component Life Cycle

- **Unmounting:** là bước cuối cùng của vòng đời thành phần, nơi thành phần được xóa khỏi trang.



Component Life Cycle

Initialization

Ở giai đoạn này, React Component sẽ tiến hành khởi tạo các state, props hay các câu lệnh được khởi tạo trong constructor()

Ví dụ :

```
class App extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      website: 'Học ReactJS'  
    };  
  }  
}
```

Component Life Cycle

Mounting

- Giai đoạn của vòng đời thành phần khi quá trình khởi tạo thành phần hoàn tất và thành phần được gắn kết trên DOM và hiển thị lần đầu tiên trên trang web.
- Bây giờ React tuân theo một thủ tục mặc định trong Quy ước đặt tên của các hàm được xác định
- Trong đó các hàm chứa “Will” biểu thị trước một số giai đoạn cụ thể và “Did” đại diện sau khi hoàn thành giai đoạn đó.

Component Life Cycle

Mounting

- **componentWillMount():** hàm này được gọi ngay trước khi thành phần được gắn trên DOM, tức là hàm này được gọi một lần trước khi hàm render () được thực thi lần đầu tiên.
- **componentDidMount():** hàm này được gọi ngay sau khi thành phần được gắn trên DOM, tức là hàm này được gọi một lần sau khi hàm render () được thực thi lần đầu tiên

Component Life Cycle

Mounting

Ví dụ:

```
class Demo extends Component {  
  constructor(props) {  
    super(props);  
    // Don't do this!  
    this.state = { color: 'green' };  
  }  
  componentWillMount() { console.log("componentWillMount da chay")}  
  componentDidMount() { console.log("componentDidMount da chay")}
```

Component Life Cycle

```
render() {  
  console.log("Ham render da duoc chay");  
  return (  
    <div>  
      <button onClick={() => this.setState({color :  
'aaaaa'}})}>Submit</button>  
      <p>{this.state.color}</p>  
    </div>  
  )  
}
```


Component Life Cycle

Updating

- Đây là giai đoạn sau giai đoạn initialization (khởi tạo) , mount (render lần đầu),... .
- Trong giai đoạn này, dữ liệu của các phần (props & state) sẽ được cập nhật để đáp ứng với các sự kiện của người dùng như click, gõ, v.v.

Component Life Cycle

Updating

- **componentWillReceiveProps():** Hàm này được gọi trước khi một thành phần bắt đầu truyền props
- **shouldComponentUpdate:** trả về giá trị true hoặc false. Điều này sẽ xác định xem thành phần sẽ được cập nhật hay không. Giá trị mặc định là True
- **componentWillUpdate :** được gọi ngay trước khi render
- **componentDidUpdate :** được gọi sau khi render

Component Life Cycle

Unmounting

Đây là giai đoạn cuối cùng của vòng đời của thành phần, là giai đoạn ngắt kết nối thành phần khỏi DOM.

- **componentWillUnmount** được gọi sau khi component được ngắt kết nối khỏi dom.

Component Life Cycle

Ví dụ tổng hợp các bước

#File App.jsx

```
import React from 'react';  
class App extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {data: 0}  
    this.setNewNumber = this.setNewNumber.bind(this)  
  };  
  setNewNumber() {  
    this.setState({data: this.state.data + 1})  
  }  
}
```

Component Life Cycle

```
render() {  
  return (  
    <div>  
      <button onClick = {this.setNewNumber}>INCREMENT</button>  
      <Content myNumber = {this.state.data}></Content>  
    </div>  
  );  
}
```

Component Life Cycle

```
class Content extends React.Component {  
  componentWillMount() { console.log('Component WILL MOUNT!') }  
  componentDidMount() { console.log('Component DID MOUNT!') }  
  componentWillReceiveProps(newProps) { console.log('Component WILL  
RECIEVE PROPS!') }  
  shouldComponentUpdate(newProps, newState) { return true;}  
  componentWillUpdate(nextProps, nextState) {console.log('Component WILL  
UPDATE!');}  
  componentDidUpdate(prevProps, prevState) {console.log('Component DID  
UPDATE!')}  
  componentWillUnmount() {console.log('Component WILL UNMOUNT!')}
```

Component Life Cycle

```
render() {  
  return (  
    <div>  
      <h3>{this.props.myNumber}</h3>  
    </div>  
  );  
}  
}  
  
export default App;
```

Component Life Cycle

#File main.js

```
import React from 'react';
```

```
import ReactDOM from 'react-dom';
```

```
import App from './App.jsx';
```

```
ReactDOM.render(<App/>, document.getElementById('app'));
```

```
setTimeout(() => {
```

```
  ReactDOM.unmountComponentAtNode(document.getElementById('app'));},  
  10000);
```


Tổng kết

Qua bài viết này chúng ta đã tìm hiểu:

- Hiểu được vòng đời của React component
- Nắm được các hàm trong vòng đời trong React component

Render có Điều Kiện(Conditional rendering)

Giới thiệu

- Trong ReactJs, đôi khi bạn có một số component và tùy thuộc vào từng điều kiện ví dụ như trạng thái của state, props,... mà bạn muốn hiển thị một hoặc một số component nào đó.
- Khi đó bạn có thể sử dụng Conditional rendering để render ra component mà bạn mong muốn.

Render có Điều Kiện

Conditional rendering

- Trong React, Conditional rendering đề cập đến quá trình cung cấp các phần tử và thành phần dựa trên các điều kiện nhất định.
- Có nhiều cách để sử dụng render có điều kiện trong React. Như với hầu hết mọi thứ trong lập trình, một số tùy thuộc vào vấn đề bạn đang cố gắng giải quyết.

Render có Điều Kiện

Viết câu lệnh if...else trong React

Tạo một component và thêm một số phương thức như sau

```
class App extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {text: '', inputText: '', mode:'view'};  
  
    this.handleChange = this.handleChange.bind(this);  
    this.handleSave = this.handleSave.bind(this);  
    this.handleEdit = this.handleEdit.bind(this);  
  }  
}
```

Render có Điều Kiện

```
handleChange(e) {  
  this.setState({ inputText: e.target.value });  
}  
  
handleSave() {  
  this.setState({text: this.state.inputText, mode: 'view'});  
}  
  
handleEdit() {  
  this.setState({mode: 'edit'});  
}  
}
```

Render có Điều Kiện

Để render ra các phương thức, ta kiểm tra các thuộc tính mode

```
render () {  
  if(this.state.mode === 'view') {  
    return (  
      <div>  
        <p>Text: {this.state.text}</p>  
        <button onClick={this.handleClick}>  
          Edit  
        </button>  
      </div>  
    );  
  }
```

Render có Điều Kiện

```
} else {  
    return (  
        <div>  
            <p>Text: {this.state.text}</p>  
            <input onChange={this.handleChange}  
                value={this.state.inputText}  
            />  
            <button onClick={this.handleSave}>Save</button>  
        </div>  
    );  
}
```

Render có Điều Kiện

Bạn cũng có thể sử dụng câu lệnh **switch**

```
switch(this.state.mode) {
```

```
  case 'a':
```

```
    // ...
```

```
  case 'b':
```

```
    // ...
```

```
  case 'c':
```

```
    // ...
```

```
  default:
```

```
    //
```

```
}
```


Render có Điều Kiện

Sử dụng render với Null

Nếu bạn muốn ẩn một component, bạn có thể cho phương thức render trả về null

```
render() {  
  if(this.props.number % 2 == 0) {  
    return (<div>  
      <h1>{this.props.number}</h1>  
      </div>);  
  } else {return null;}  
}
```

Ví dụ trên cho thấy kết quả sẽ luôn trả về là số chẵn, nếu không phải số lẻ kết quả là null

Render có Điều Kiện

Toán tử ba ngôi trong React(Ternary operator)

Thay vì sử dụng if...else, ta có thể sử dụng toán tử ba ngôi theo cú pháp

`condition ? expr_if_true : expr_if_false`

Render có Điều Kiện

Toán tử ba ngôi trong React (Ternary operator)

Ví dụ

```
// ...
```

```
return ( <div>  
  <p>Text: {this.state.text}</p>  
  { view ? null : (<p>  
    <input onChange={this.handleChange}  
      value={this.state.inputText} /> </p>)  
  }  
  </div>  
);
```

Render có Điều Kiện

Toán tử AND (&&)

- Toán tử bậc ba có một trường hợp đặc biệt mà nó có thể được đơn giản hóa. Khi bạn muốn hiển thị một cái gì đó hoặc không có gì, bạn chỉ có thể sử dụng toán tử &&.
- Nếu biểu thức đầu tiên được đánh giá là false (false &&...), thì không cần thiết phải đánh giá biểu thức tiếp theo vì kết quả sẽ luôn là false

Render có Điều Kiện

Ví dụ:

```
return (  
  <div>  
    { showHeader && <Header /> }  
  </div>  
);
```

Nếu showHeader là true, thì component <Header /> sẽ được trả về bởi biểu thức. Nếu showHeader là false, component <Header /> sẽ bị bỏ qua và <div> trống sẽ được trả về.

Tổng kết

Qua bài học chúng ta đã tìm hiểu:

- Render có điều kiện trong React
- Các cách Conditional rendering sử dụng trong React

[Thực hành] Đổi màu nền component

Mục tiêu

- Tạo được dự án React JS
- Thao tác với state thông qua componentDidMount

Mô tả

Viết chương trình cho phép tăng hoặc giảm số thông qua các nút



[Thực hành]Hiện thông báo

Mục tiêu

- Tạo được dự án React JS
- Thao tác với state thông qua componentWillUnmount

Mô tả

Viết chương trình hiện thông báo trước khi ẩn component



[Thực hành] Kiểm tra User Login/Logout

Mục tiêu

- Tạo được dự án React JS
- Thao tác với state thông qua event
- Giao tiếp giữa các components
- Thực hành với cơ chế condition rendering
- **Mô tả**

Viết chương trình cho phép Login/ Logout

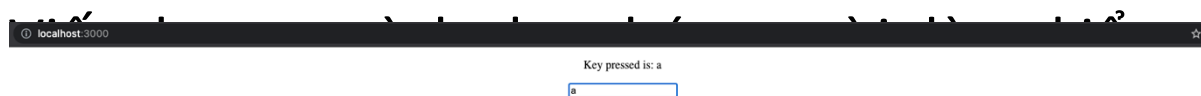


[Bài tập] Kiểm tra phím được nhập

Mục tiêu

- Tạo được dự án React JS
- Thao tác với state thông qua event
- Thực hành condition rendering

- **Mô tả**



ra phím được nhập

[Bài tập] Xử lý sự kiện Login/ Logout

Mục tiêu

- Tạo được dự án React JS
- Thao tác với state thông qua event
- Giao tiếp giữa các components
- Thao tác với componentWillUnmount

Mô tả

\ phép Login/ Logout với Bootstrap form



[Bài tập] Ứng dụng quản lý công việc Todo App

Mục tiêu

- Tạo được dự án React JS
- Thao tác với state thông qua event

Mô tả



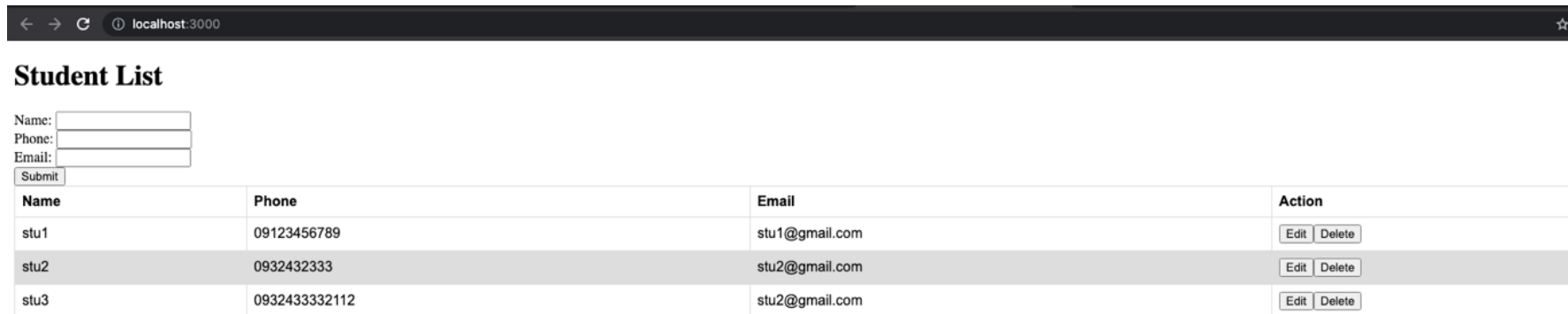
[Bài tập] Ứng dụng quản lý sinh viên

Mục tiêu

- Tạo được dự án React JS
- Sử dụng được hàm `React.createElement`

Mô tả

Viết chương trình thể hiện được bảng thông tin của các sinh viên trong lớp học



Name	Phone	Email	Action
stu1	09123456789	stu1@gmail.com	<button>Edit</button> <button>Delete</button>
stu2	0932432333	stu2@gmail.com	<button>Edit</button> <button>Delete</button>
stu3	093243332112	stu2@gmail.com	<button>Edit</button> <button>Delete</button>