Bài tập

Nội dung kiến thức

- Cú pháp JSX
- Các kiểu component
- Đối tượng props trong các component
- Đối tượng state trong component
- Xử lý sự kiện trong React
- Kết xuất theo điều kiện
- Nhúng CSS

Nộp bài: Nộp link CodeSandBox theo yêu cầu giáo viên.

Câu hỏi

1. Đoạn mã sau sử dụng JSX để gán một phần tử **div** cho hằng số trong JSX. Thay thế **div** bằng một phần tử **h1** và thêm văn bản **Xin chào JSX!** bên trong nó.

```
const JSX = <div></div>;
```

2. Định nghĩa một hằng số **JSX** mới hiển thị các phần tử sau theo thứ tự:

Một **h1**, một **p** và một danh sách không có thứ tự chứa ba mục **li**. Nội dung các phần tử là tùy ý.

3. Để đặt chú thích (comment) bên trong **JSX**, bạn sử dụng cú pháp **{/ * * /}** để bao quanh văn bản nhận xét. Thêm một nhận xét bên trong phần tử <**div**> của định nghĩa **JSX** sau:

```
const JSX = (
  <div>
    <h1>This is a block of JSX</h1>
    <p>Here's a subtitle</p>
  </div>
);
```

Ví dụ:

```
const JSX = (
   // Toi la comment 1
   <ul>
    {/*Toi la comment 2*/}
      <li>Xin chào JSX!</li>
      <li>Xin chào JSX!</li>
    </ul>
);
```

4. Kết xuất định nghĩa **JSX** trên đến phần tử DOM có ***id = 'jsx-node'*** dùng phương thức ***ReactDOM.render()***.

5. Thay đổi nội dung đoạn mã sau cho chính xác:

```
const JSX = (
  <div lass = "myDIV">
    <h1>This is a block of JSX</h1>
    <p>Here's a subtitle</p>
  </div>
);
```

6. Thay đổi nội dung đoạn mã sau cho chính xác:

```
const JSX = (
  <div>
    <h2>Welcome to React!</h2> <br >
    <p>Be sure to close all tags!</p>
    <hr >
  </div>
);
```

7. Cho hàm

```
const MyComponent = function() {
      // viết code ở đây
}
```

- Hoàn thiện hàm **MyComponent** để trả về một phần tử **div** chứa nội dung bất kỳ.

- Viết lại **MyComponent** dạng hàm mũi tên (arrow function).

8. Viết lại **MyComponent** dạng **class**.

9. Lồng các component sau đây một cách hợp lý:

```
// Stateless Functional Component
const TypesOfFruit = () => {
  return (
    <div>
      <h2>Fruits:</h2>
      <ul>
        <li>Apples</li>
        <li>Blueberries</li>
        <li>Strawberries</li>
        <li>Bananas</li>
      </ul>
    </div>
  );
};
// Stateless Functional Component
const Fruits = () => {
  return (
    <div>
    </div>
  );
};

class TypesOfFood extends React.Component {
  constructor(props) {
    super(props);
```

```
  }

  render() {
    return (
      <div>
        <h1>Types of Food:</h1>
      </div>
    );
  }
};
```

10. Cho đoạn mã định nghĩa các component sau:

```
// Stateless Functional Component
const CurrentDate = (props) => {
 return (
    <div>
      <p>The current date is: </p>
      {/*Viết code ở đây*/}
    </div>
  );
};


class Calendar extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h3>What date is it?</h3>
```

```
        { /* Thay đổi nội dung CurrentDate  */ }
            <CurrentDate />
      </div>
    );
  }
};
```

Truy cập thuộc tính tên **date** thông qua **props** của component **CurrentDate** để hiển thị ngày hiện tại bằng cách gọi hàm **Date()** trong JavaScript.

11. Cho các component sau:

```
// Stateless Functional Component
const List = (props) => {
  { /* Thêm thuộc tính mảng tasks trong cặp {}  */ }
  return <p>{ }</p>
};

class ToDo extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h1>To Do Lists</h1>
        <h2>Today</h2>
     { /* Thêm thuộc tính tasks đến List gồm 2 giá trị taskA và taskB */ }
        <List />
        <h2>Tomorrow</h2>
  { /* Thêm thuộc tính tasks đến List gồm 3 giá trị taskC, taskD, taskE */ }
        <List />
```

```
      </div>
    );
  }
};
```

Có các component **List** và **ToDo**. Khi hiển thị từng **List** từ **ToDo**, hãy chuyển thuộc tính ***tasks*** được gán cho một mảng công việc cần làm. Sau đó, truy cập mảng ***tasks*** này trong **List**, hiển thị giá trị của nó trong phần tử **p**. Sử dụng hàm ***join (",")*** để hiển thị mảng ***tasks*** trong phần tử **p** dưới dạng danh sách được phân tách bằng dấu phẩy. **List** của *Today* nên có ít nhất 2 nhiệm vụ và *Tomorrow* nên có ít nhất 3 nhiệm vụ sao cho khi kết xuất sẽ trông như sau:

To Do Lists

Today

task A, task B

Tomorrow

task C, task D, task E

12. Component **StatefulComponent** đang cố gắng hiển thị thuộc tính **firstName** từ trạng thái của nó. Tuy nhiên, không có trạng thái nào được xác định. Khởi tạo thành phần với trạng thái dùng **state** trong phương thức khởi tạo (constructor) và gán tên của bạn cho một thuộc tính của **firstName**. Quá trình này gọi là tạo một component trạng thái (a stateful component):

```
// A Stateful Component
class StatefulComponent extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
```

```
      <div>
      {/*Kết xuất state đến giao diện người dùng (UI)*/}
        <h1>{this.state.firstName}</h1>
      </div>
    );
  }
};
```

13. Chúng ta có thể kết xuất trạng thái (state) của một component đến giao diện (UI) bằng 2 cách:

- Cách 1: Dùng trực tiếp đối tượng **state** như câu 12

- Cách 2: Khai báo một biến trong phương thức render() phía trên lệnh **return** và gán thuộc tính của đối tượng **state** đến biến này.

Hãy sử dụng cách 2 bằng cách khai báo một biến tên **firstName** trong component **StatefulComponent**.

Ví dụ

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'freeCodeCamp'
    }

  }

  render() {

    const name = this.state.name;

    return (
```

```
      <div>
        <h1>{name}</h1>
      </div>
    );
  }
};
```

14. Cho component sau:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'Initial State'
    };
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    // Viết mã ở đây
   }
  render() {
    return (
      <div>
        <button onClick={this.handleClick}>Click Me</button>
        <h1>{this.state.name}</h1>
      </div>
    );
  }
};
```

Viết mã trong phương thức *handleClick* để khi nhấn chuột vào button sẽ thay đổi nội dung thuộc tính **name** của **state** từ ***Initial State*** thành ***React Rocks***.

Gợi ý: Dùng phương thức ***setState()***.

15. Cho component sau

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      text: "Hello"
    };
  }
  handleClick() {
    this.setState({
      text: "You clicked!"
    });
  }
  render() {
    return (
      <div>
                <button>Click Me</button>
                <h1>{this.state.text}</h1>
      </div>
    );
  }
};
```

Thay đổi nội dung bên trong **MyComponent** để khi nhấn chuột vào button sẽ thực hiện phương thức *handleClick* và hiển thị thông điệp ***You clicked!*** Dùng theo 2 cách:

- Hàm mũi tên

- Phương thức ***bind***

Gợi ý: Tham khảo **câu 14** và
https://hoctructuyencntt.github.io/React/Bai03.html#muc3_2

16. Cho component sau:

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  render() {
    return (
      <div>
        <button className='inc' onClick={this.increment}>Increment!</button>
        <button className='dec' onClick={this.decrement}>Decrement!</button>
        <button className='reset' onClick={this.reset}>Reset</button>
        <h1>Current Count: {this.state.count}</h1>
      </div>
    );
  }
};
```

Counter theo dõi giá trị count của state, các button gọi các phương thức *increment()* để tăng giá trị count lên 1, *decrement()* giảm giá trị count xuống 1 và *reset()* gán giá trị count trở lại 0.

Viết các phương thức này trong component Counter theo 2 cách:

- Dùng ***bind***
- Dùng hàm mũi tên

17. Cho component sau:

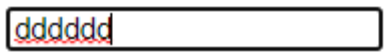
```
class ControlledInput extends React.Component {
```

```
  constructor(props) {
    super(props);
    this.state = {
      input: ''
    };
  }
  render() {
    return (
      <div>
        <input />
        <h4>Controlled Input:</h4>
        <p>{this.state.input}</p>
      </div>
    );
  }
};
```

Kết xuất component này đến trình duyệt sẽ như sau:

Controlled Input:

Thêm một **input** đến component và xử lý sự kiện ***onChange*** để khi người dùng gõ nội dung trong input sẽ hiển thị nội dung này như sau:



Controlled Input:

dddddd

Gợi ý các bước thực hiện:

- Tạo ra một hàm **handleChange(event)** thay đổi giá trị thuộc tính **input** thành *event.target.value*. Hàm này có thể được định nghĩa dùng *hàm mũi tên* hay *bind*

- Bước kế tiếp thêm thuộc tính **value** và **onChange** đến **input** trong đó **value** nhận giá trị là thuộc tính *input* từ **state** và **onChange** nhận giá trị là hàm *handleChange*

18. Cho component sau:

```
class MyForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      input: '',
      submit: ''
    };

  }
  handleChange(event) {
    this.setState({

    });
  }
  handleSubmit(event) {
    event.preventDefault()
    this.setState({

    });
  }
  render() {
    return (
      <div>
        <form onSubmit={}>
```

```
          <input
            value={this.state.input}
            onChange={} />
          <button type='submit'>Submit!</button>
        </form>
        <h1>{this.state.submit}</h1>
      </div>
    );
  }
};
```

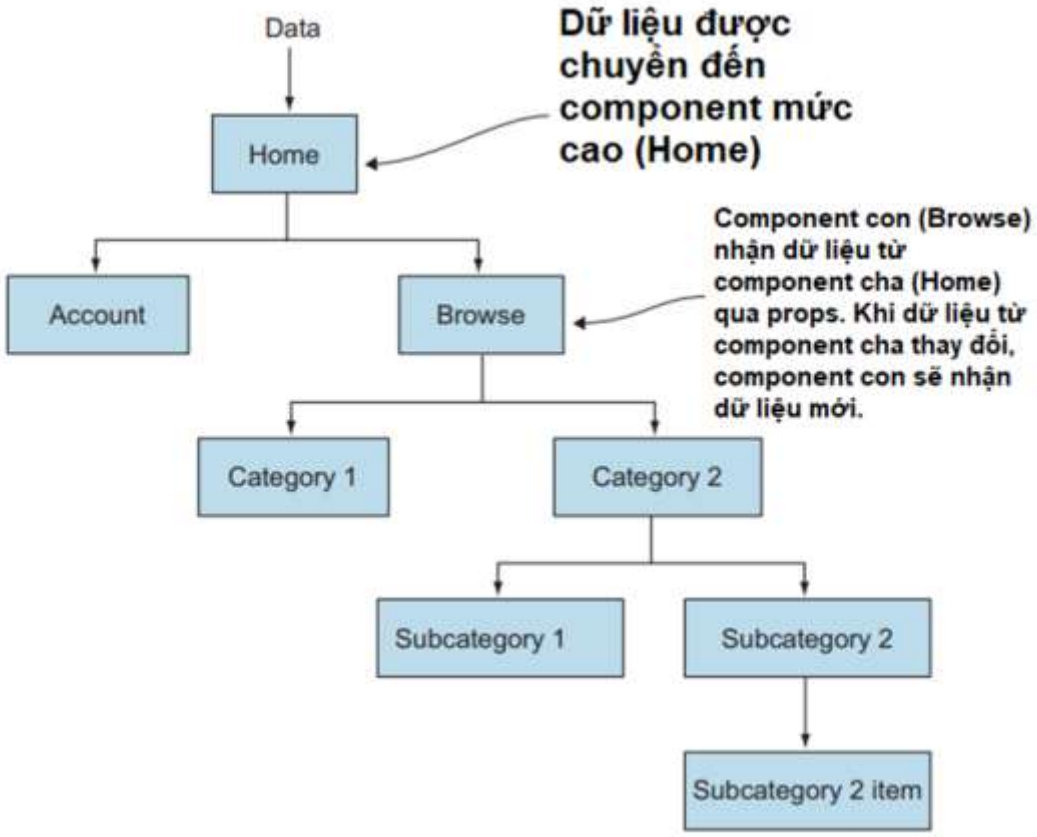
Hoàn thiện component trên để khi kết xuất đến trình duyệt web:

Nhập nội dung bất kỳ và nhấn Submit

minh

19. Có hai đặc điểm quan trọng trong quản lý trạng thái của React:

- Đầu tiên là luồng dữ liệu một chiều (One-way Dataflow). Dữ liệu chuyển theo một hướng xuống cây các component của ứng dụng, từ component cha có trạng thái đến các component con. Các component con chỉ nhận dữ liệu trạng thái mà chúng cần (xem hình).



- Thứ hai là các ứng dụng có trạng thái phức tạp có thể được chia nhỏ thành một vài hoặc có thể là một component trạng thái duy nhất. Phần còn lại của các componnet chỉ đơn giản là nhận trạng thái từ component cha dưới dạng ***props*** và hiển thị giao diện người dùng từ trạng thái đó. Nguyên tắc tách logic trạng thái khỏi logic giao diện người dùng là một trong những nguyên tắc chính của React. Khi nó được sử dụng đúng cách, nó làm cho việc thiết kế các ứng dụng phức tạp, có trạng thái dễ quản lý hơn nhiều.

Cho các component MyApp và NavBar như sau:

```
class MyApp extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'CamperBot'
    }
  }
```

```
  render() {
    return (
       <div>
         <Navbar />
       </div>
    );
  }
};
class Navbar extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
    <div>
      <h1>Hello, my name is: </h1>
    </div>
    );
  }
};
ReactDOM.render(<MyApp />, document.getElementById('root'));
```

Thêm nội dung thích hợp đến **MyApp** và **NavBar** để khi kết xuất **MyApp** đến trình duyệt sẽ hiển thị

Hello, my name is: CamperBot

20. Cho component sau:

```
class Colorful extends React.Component {
  render() {
```

```
    return (
      <div>Big Red</div>
    );
  }
};
```

Định dạng Big Red thành màu đỏ, cỡ chữ 72px và đóng khung viền có độ dày 2px, màu viền blue, kiểu solid.

21. Cho component sau:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      display: true
    }
    this.toggleDisplay = this.toggleDisplay.bind(this);
  }
  toggleDisplay() {
    this.setState((state) => ({
      display: !state.display
    }));
  }
  render() {
    return (
       <div>
         <button onClick={this.toggleDisplay}>Toggle Display</button>
         <h1>Displayed!</h1>
       </div>
    );
  }
```

```
};
```

Thêm nội dung cần thiết đến MyComponent để khi kết xuất đến trình duyệt sẽ:

Displayed!

Nhấn nút *Toggle Display* chữ **Displayed!** sẽ biến mất

Nhấn lại nút Toggle Display một lần nữa

Displayed!