

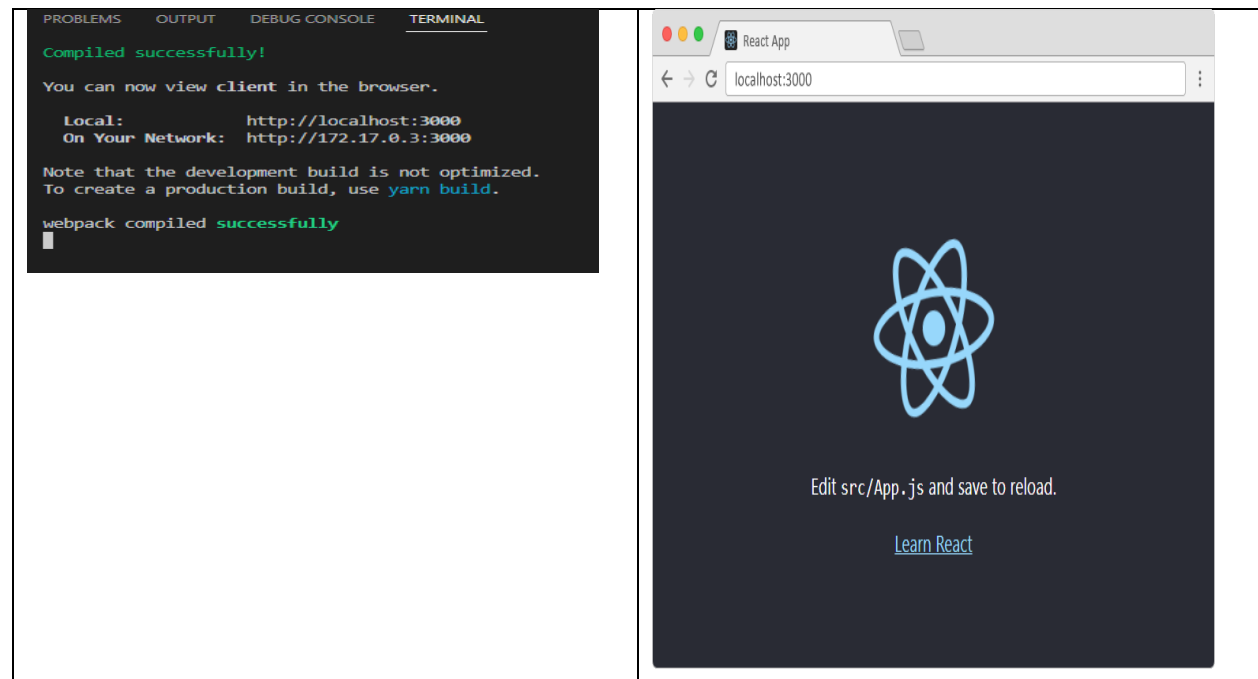
Contents

1. Xây dựng ứng dụng đầu tiên.....	2
2. React component.....	2
Class component.....	3
Functional component.....	3
3. Sử dụng git và push Project lên kho lưu trữ Của github	4
4. Importing and Exporting Components.....	5
Import.....	5
Export.....	5
5. State and prop	6
State	6
Prop.....	7
6. Một số hook cơ bản.....	10
useEffect	10
useRef	14
Bài tập 1:.....	15
Bài tập 2.....	15
7. Single Page Application (SPA)	18
8. Routing trong ReactJS	19
a. Các component cơ bản	19
Ví dụ	20
Chú ý:.....	20
Cách khác:.....	25
Bài tập	26
9. Controlled Forms	27
Đặc điểm quan trọng của Controlled Forms	27
Lợi ích của controlled forms	28
Bất lợi của controlled forms.....	28
10. Uncontrolled Forms	29
Đặc điểm quan trọng của Uncontrolled Forms	29
Lợi ích của Uncontrolled forms	30
Bất lợi của Uncontrolled forms.....	30
11. Template cho app	30
12. Redux	32
13. Redux form	32
14. JSON Server.....	32

15. React Animation..... 32

1. Xây dựng ứng dụng đầu tiên

npx create-react-app my-app
Khởi chạy project npm start
Runs the app in the development mode. Open http://localhost:3000 to view it in your browser. The page will reload when you make changes. You may also see any lint errors in the console.



2. React component

React component hay component là một block code độc lập, có khả năng tái sử dụng. Việc chia UI thành các component giúp cho việc tổ chức và quản lý code dễ dàng hơn

React component bao gồm 2 loại:

- **Class component**
- **Functional component**

Class component

Đặc điểm

Các Class component đơn giản là những **class ES6** kế thừa từ class tên 'Component' của React

Mọi Class component sẽ **phải chứa method render()**, nơi **return một JSX template** hoặc **null**

Trong docs mới nhất của React, Class component đã được đưa vào danh mục **'Legacy React APIs'** và đội ngũ phát triển React cũng khuyến khích việc sử dụng Functional Component để thay thế. Tuy nhiên, Class component hiện vẫn sẽ được hỗ trợ sử dụng.

<https://react.dev/reference/react/Component>

Syntax

```
import React, { Component } from "react";
class ClassComponent extends Component {
  render() {
    return (
      <div>
        <h1>Hello Class component!!</h1>
      </div>
    );
  }
}
export default ClassComponent;
```

Functional component

Đặc điểm

Xuất hiện từ phiên bản React 16.8

Functional component được tạo bởi một hàm JavaScript, return một JSX template hoặc null

Vì được tạo bởi một hàm JavaScript, ta có thể viết Functional component dưới dạng JS function thông thường hay ES6 arrow function

Với việc đội ngũ phát triển React đang chú trọng phát triển các hooks và cách viết, triển khai code khá dễ hiểu, Functional component được khuyến khích sử dụng

Syntax

JS function

```
import React from "react";
const FunctionComponent = () => {
  return (
    <div>
      <h1>Function
      component example!!!</h1>
    </div>
  );
}
export default
FunctionComponent;
```

```
import React from "react";
function FunctionComponent{
  return (
    <div>
      <h1>Function
      component example!!!</h1>
    </div>
  );
}
export default
FunctionComponent;
```

3. Sử dụng git và push Project lên kho lưu trữ Của github

git init (để khởi tạo git)

Branch

Khi sử dụng Git, bạn có thể tạo ra nhiều nhánh (branch) khác nhau. Câu lệnh Git này dùng để kiểm tra branch hiện tại:

git branch

Để tạo mới một branch:

git branch <name_branch>

Xóa một branch:

git branch -d <name_branch>

Để đưa subsidiary vào branch master, thì trước hết sẽ di chuyển đến branch master. Sau đó làm các bước tiếp theo:

git checkout master git branch new-branch git checkout new-branch git add . git commit -m "first-commit" git checkout master git merge new-branch	Để cập nhật tất cả các file thì ta sử dụng câu lệnh: git add . Còn muốn cập một file nào đó thì sử dụng câu lệnh: git add .<tên file> sử dụng câu lệnh Commit để ghi lại việc thay đổi và đẩy thông tin thay đổi lên Local Repository: git commit -m 'Commit 1'
---	--


Tạo Repository: my-app

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Owner *

 anhttv20

Repository name *

/ my-app

✔ my-app is available.

Great repository names are short and memorable. Need inspiration? How about [potential-barnacle](#)?

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

...or create a new repository on the command line

```
echo "# my-app" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/anhttv20/my-app.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/anhttv20/my-app.git
git branch -M main
git push -u origin main
```

4. Importing and Exporting Components

Module

Một module là một file. Hay là “One script is one module”. Những module có thể load nhiều function bởi hai keywords đặc biệt đó là Import và Export. Và đặc biệt module này có thể gọi và sử dụng một module khác.

Import

Cho phép import các functionality từ các module khác. Export: Khai báo những variables hoặc function cho phép những module khác truy cập và sử dụng

Export

Có 2 loại export đó là named và default:

Named Export: Trong JavaScript ES6, named export được sử dụng để xuất nhiều thứ từ một module bằng cách thêm keyword export vào khai báo của chúng. Những thứ được export sẽ được phân biệt bằng tên. Sau đó import những thứ chúng ta cần sử dụng bằng cách bao quanh chúng cặp dấu ngoặc nhọn { }. Tên của module đã nhập phải giống với tên của module đã xuất.

Default Export: Trong Javascript ES6 chỉ cho phép xuất một mặc định cho mỗi file. Default Export có thể cho một function, class hoặc một object.

Không thể export khi định nghĩa tên: export default const name = 'value';

```
import React from "react";
import Taylor from '../assets/images/Taylor.png'
export function Profile() {
  return (
    <img
      src={Taylor}
      alt="Taylor Swift"
      width="200px" height="200px"
    />
  );
}
export default function Gallery() {
  return (
    <section>
```

```

        <h1>Amazing singers</h1>
        <Profile />
        <Profile />
        <Profile />
    </section>
);
}

import Gallery from './components/Gallery';
import { Profile } from './components/Gallery'

```

5. State and prop

State

Được sử dụng để quản lý dữ liệu của riêng component chứa nó

Thay đổi được giá trị thông qua hook useState (Functional Component), khi state thay đổi giá trị, Component chứa state sẽ được re-render

Syntax:

```

import { useState } from "react";
.....
const [state, setState] = useState([giá trị khởi tạo]);

```

Ví dụ bộ đếm (counter) trong StateExample.jsx, ta có 1 state mang tên 'counter', phương thức cập nhật lại giá trị của state này là 'setCounter' và state này có giá trị ban đầu = 0

```
const [counter, setCounter] = useState(0);
```

Khi thực hiện bấm nút Increase, function handleIncrease sẽ được gọi ở onClick của button, setCounter sẽ được thực hiện và thay đổi tăng state counter, dẫn tới việc Component StateExample được render lại và số tương ứng sẽ xuất hiện

Trường hợp tương tự xảy ra với nút Decrease

```

const handleIncrease = () => {
    setCounter(counter + 1);
};
const handleDecrease = () => {
    setCounter(counter - 1);
};

```

Một số css

box.css	stateExample.css
<pre> .box-wrapper { width: 100px; height: 100px; margin: 10px; } //propExample.css .box-container { display: flex; flex-direction: row; justify-content: center; } </pre>	<pre> .state-example-wrapper { margin-bottom: 40px; } .counter-text { font-size: 30px; } .state-change-btn { font-size: 15px; padding: 5px 10px; margin: 10px; } </pre>

Demo như sau

State Example

Counter: 5

Decrease

Increase

```
StateExample.jsx
import React, { useState } from "react";
import "../assets/css/stateExample.css";
const StateExample = () => {
  const [counter, setCounter] = useState(0);
  const handleIncrease = () => {
    setCounter(counter + 1);
  };
  const handleDecrease = () => {
    setCounter(counter - 1);
  };

  return (
    <div className="state-example-wrapper">
      <h1>State Example</h1>
      <h2 className="counter-text">Counter: {counter}</h2>
      <button className="state-change-btn"
onClick={handleDecrease}>
        Decrease
      </button>
      <button className="state-change-btn"
onClick={handleIncrease}>
        Increase
      </button>
    </div>
  );
};
export default StateExample;
```

Prop

Được sử dụng để truyền dữ liệu giữa các Component của React theo hướng từ Component cha sang Component con dưới dạng một object.

Không thể thay đổi được, prop là dạng dữ liệu 'chỉ đọc' tức là props không thể thay đổi ở Component con, nếu cần có sự thay đổi thì phải thay đổi ở Component cha sau đó truyền lại xuống.

Ví dụ trong Component cha PropExample.jsx, ta truyền dữ liệu là màu của box xuống Component con là Box.jsx

```
<Box boxColor="green" />
```

Component Box sẽ nhận được prop và dữ liệu sẽ được sử dụng để set màu như trong style ở dưới.

```
import React from "react";
import "../assets/css/box.css";
const Box = (props) => {
  return (
    <div>
      <div className="box-wrapper"
        style={
          { backgroundColor: props.boxColor }
        }>
      </div>
    </div>
  );
}
```

Ta hoàn toàn có thể thử các giá trị màu khác để nhận được các box với màu tương ứng

```
...
<Box boxColor="black" />
<Box boxColor="blue" />
...
```

Để ví dụ cho trường hợp **prop không thể thay đổi được**, ta sẽ thay đổi màu của box bằng cách đặt thêm một button 'Change prop color', khi click ta sẽ thực hiện set lại giá trị boxColor của prop

```
const Box = (props) => {
  return (
    <div>
      <div className="box-wrapper"
        style={
          { backgroundColor: props.boxColor }
        }>
      </* button được thêm ở đây */>
      <button
        onClick={() => {
          props.boxColor = "yellow";
        }}
      >
        Change prop color
      </button>
    </div>
  </div>
  );
}
```

Một lỗi sẽ được hiển thị thông báo ta không thể gán giá trị ở đây là boxColor (prop) từ đối tượng 'không thể mở rộng'

Uncaught runtime errors:

ERROR

State Example

```
Cannot add property boxColor, object is not extensible
TypeError: Cannot add property boxColor, object is not extensible
    at onClick (http://localhost:3000/static/js/bundle.js:169:26)
    at HTMLUnknownElement.callCallback (http://localhost:3000/static/js/bundle.js:11224:18)
    at Object.invokeGuardedCallbackDev (http://localhost:3000/static/js/bundle.js:11268:20)
    at invokeGuardedCallback (http://localhost:3000/static/js/bundle.js:11325:35)
    at invokeGuardedCallbackAndCatchFirstError (http://localhost:3000/static/js/bundle.js:11339:29)
    at executeDispatch (http://localhost:3000/static/js/bundle.js:15483:7)
    at processDispatchQueueItemsInOrder (http://localhost:3000/static/js/bundle.js:15509:11)
    at processDispatchQueue (http://localhost:3000/static/js/bundle.js:15520:9)
```

Để thay đổi được màu của box, ta sẽ phải thực hiện ở Component cha PropExample.jsx:

Ta đặt thêm 2 button ở Component cha lần lượt là 'Change to red' và 'Change to yellow' và dùng 1 state để quản lý màu (Cần state thay vì biến thường để re-render Component và thấy được sự thay đổi)

Ban đầu, với giá trị mặc định của state boxColor là green, Box sẽ được render và hiển thị với màu xanh tương ứng

Khi click 1 trong 2 button trên, state boxColor sẽ được thay đổi, đồng nghĩa với việc Component cha là PropExample chứa state sẽ được re-render, Box lúc này cũng sẽ được render lại với giá trị boxColor mới tương ứng với nút vừa bấm

```
import React, { useState } from "react";
import "../assets/css/propExample.css";
import Box from "./Box";
const PropExample = () => {
  const [boxColor, setBoxColor] = useState("green");

  const handleChangeToRed = () => {
    setBoxColor("red");
  };

  const handleChangeToYellow = () => {
    setBoxColor("yellow");
  };
  return (
    <div>
      <h1>Prop Example</h1>
      <div className="box-container">
        { /* <Box boxColor="green" /> */ }
        <Box boxColor={boxColor} />
      </div>
      <button className="state-change-btn"
onClick={handleChangeToRed}>
        Change to red
      </button>
      <button className="state-change-btn"
onClick={handleChangeToYellow}>
        Change to yellow
    </div>
  );
};
```

```

        </button>
      </div>
    );
  };
  export default PropExample;

```

6. Một số hook cơ bản

useState:

Dùng để khai báo state của component (Đã đề cập phía trên)

useEffect

Được sử dụng để xử lý các side effects:

- Thao tác API
- Thao tác với DOM
- Thêm, xóa event listeners ('click', 'scroll', ...)
- setTimeout, setInterval

Syntax

```

import {useEffect} from 'react'
...
useEffect(callback, dependency array nếu có);

```

Ví dụ về các trường hợp của dependency array

- Trong trường hợp dependency array không xuất hiện, useEffect sẽ chạy với mỗi lần component re-render
- Trong trường hợp dependency array rỗng, useEffect sẽ chạy đúng một lần khi component render lần đầu tiên
- Trong trường hợp dependency array có giá trị, useEffect sẽ chạy khi component render và một trong các giá trị truyền vào có sự thay đổi

effectExample.css	refExample.css
<pre> .effect-example { margin-top: 20px; } </pre>	<pre> .increaseBtn { font-size: 30px; width: fit-content; } .count-text-wrapper { margin-bottom: 50px; } </pre>

Ví dụ 1:

```

EffectExample.jsx
import React, { useEffect, useState } from "react";
const EffectExample = () => {
  const [countA, setCountA] = useState(0);

```

```

const [countB, setCountB] = useState(0);
useEffect(() => {
  console.log("Không có dependency array");
});
useEffect(() => {
  console.log("Dependency array rỗng");
}, []);
useEffect(() => {
  console.log("Dependency array phụ thuộc vào countA");
}, [countA]);
useEffect(() => {
  console.log("Dependency array phụ thuộc vào countA và countB");
}, [countA, countB]);
const handleIncreaseCountA = () => {
  setCountA(countA + 1);
};
const handleIncreaseCountB = () => {
  setCountB(countB + 1);
};
return (
  <div className="effect-example">
    <div className="count-text-wrapper">
      <h1>Count A: {countA}</h1>
      <h1>Count B: {countB}</h1>
    </div>
    <div>
      <button onClick={handleIncreaseCountA}>Increase Count A</button>
      <button onClick={handleIncreaseCountB}>Increase Count B</button>
    </div>
  </div>
);
};
export default EffectExample;

```

Ví dụ 2: ban đầu

```

import React, { useState } from 'react';

export const EffectDemo = () => {
  //State
  const [fullName, setFullName] = useState({ name: 'name',
familyName: 'family' });
  const [title, setTitle] = useState('useEffect() in Hooks');

  return (
    <div>
      <h1>Title: {title}</h1>
      <h3>Name: {fullName.name}</h3>
    </div>
  );
};

```

```
        <h3>Family Name: {fullName.familyName}</h3>
      </div>
    );
  };
};
```

Sau đó khai báo useEffect()

```
import React, { useState, useEffect } from 'react';

export const EffectDemo = () => {
  //State
  const [fullName, setFullName] = useState({ name: 'name',
familyName: 'family' });
  const [title, setTitle] = useState('useEffect() in Hooks');

  //useEffect
  useEffect(() => {
    setFullName({ name: 'TrungHC', familyName: 'HCT' });
  });
  return (
    <div>
      <h1>Title: {title}</h1>
      <h3>Name: {fullName.name}</h3>
      <h3>Family Name: {fullName.familyName}</h3>
    </div>
  );
};
```

Sau đó

```
useEffect(() => {
  console.log('useEffect has been called!');
  setFullName({ name: 'TrungHC', familyName: 'HCT' });
}, [fullName.name]);
```

Như vậy hàm useEffect() chỉ được gọi 2 lần: 1 lần khi render components, 1 lần khi set name thành "TrungHC".

Vậy nếu chúng ta muốn hàm useEffect() chỉ gọi 1 lần khi render components (tương đương với componentDidMount) thì như thế nào? Trong trường hợp này ta chỉ cần truyền tham số thứ 2 của useEffect() là 1 hàm rỗng []

```
useEffect(() => {
  console.log('useEffect has been called!');
  setFullName({ name: 'TrungHC', familyName: 'HCT' });
}, []);
```

Ví dụ về clean up function

Khi truyền callback vào useEffect, callback có thể return một clean up function. Clean up function mang vai trò 'dọn dẹp' những gì đã thực thi trong useEffect nếu cần thiết

Ở ví dụ có bộ đếm (component Counter) tăng theo từng giây sử dụng setInterval và một button ẩn hiện bộ đếm này. Trong một số trường hợp ta cần bộ đếm bị ẩn đi (hay component Counter bị loại bỏ khỏi view),

ta sẽ muốn bộ đếm này dừng lại để tránh một số hành vi không mong muốn. Việc này sẽ được xử lý ở clean up function bằng cách clearInterval

```
EffectIntervalExample.jsx

import React, { useState } from "react";
import Counter from "../Counter";

const EffectIntervalExample = () => {
  const [showCounter, setShowCounter] = useState(true);

  const toggleCounter = () => {
    setShowCounter((prevShowCounter) => !prevShowCounter);
  };

  return (
    <div>
      {showCounter && <Counter />}
      <button onClick={toggleCounter}>Toggle Counter</button>
    </div>
  );
};

export default EffectIntervalExample;
```

```
Counter.jsx

import React, { useEffect, useState } from "react";

const Counter = () => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setCount((prevCount) => prevCount + 1);
      console.log("Increase count");
    }, 1000);

    return () => {
      clearInterval(interval);
    };
  }, []);

  return (
    <div>
      <h1>Count: {count}</h1>
    </div>
  );
};

export default Counter;
```

Trong trường hợp không có clean up function hoặc clean up function không xử lý việc clearInterval, kể cả khi ẩn bộ đếm, ta vẫn sẽ thấy trong console liên tục in ra 'Increase count' => Đây là điều ta không mong muốn xảy ra

useRef

Được sử dụng để

- Lưu trữ tham chiếu giữa các lần re-render của component
- Truy xuất đến thành phần DOM

Việc thay đổi giá trị của useRef không gây ra sự re-render của component như useState, vì vậy, các biến liên quan đến hiển thị UI thì ta nên sử dụng useState, còn lại có thể sử dụng useRef

Syntax

```
import {useRef} from "react"
...
const ref = useRef([Giá trị khởi tạo]);
```

Ví dụ 1: lưu trữ tham chiếu giữa các lần re-render của component

```
RefExample.jsx
import React, { useRef, useState } from "react";
import "../assets/css/refExample.css";

const RefExample = () => {
  const countRef = useRef(0);
  const countObj = {
    current: 0,
  };

  const [count, setCount] = useState(0);

  const increaseCount = () => {
    countRef.current++;
    countObj.current++;
    setCount(count + 1);
  };

  console.log("count", count);
  console.log("countRef", countRef);
  console.log("countObj", countObj);

  return (
    <button className="increaseBtn" onClick={increaseCount}>
      Rerender
    </button>
  );
};

export default RefExample;
```

Sử dụng 1 state count và với mỗi lần bấm nút, state này sẽ bị thay đổi dẫn tới việc component RefExample sẽ được re-render

Quan sát giá trị của countRef và countObj :

- Với countRef, giá trị current sẽ bị thay đổi sau mỗi lần component re-render do giá trị trước đó vẫn được lưu lại
- Với countObj, giá trị current sẽ giữ nguyên do mỗi lần component re-render thì giá trị của các object thường như này sẽ được khởi tạo lại hoàn toàn

Ví dụ 2: truy xuất đến thành phần DOM:

Trong trường hợp này, ref chính là một element trên DOM, hay cụ thể hơn là input, chúng ta hoàn toàn có thể thao tác với element này như lấy giá trị của input hay click button để focus input này

```
RefDomExample.jsx
import React, { useRef } from "react";

const RefDomExample = () => {
  // Thường khi dùng ref truy xuất DOM giá trị khởi tạo sẽ là null
  const inputRef = useRef(null);

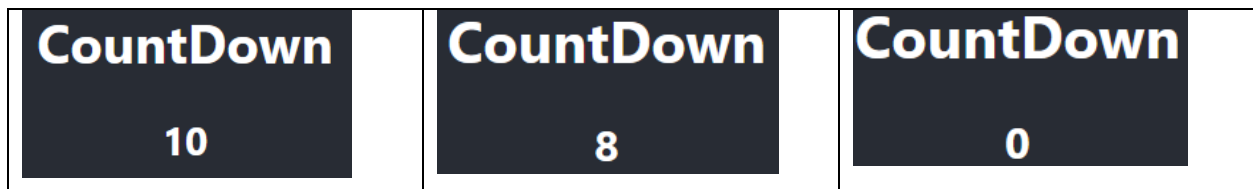
  const getInputValue = () => {
    console.log("Giá trị của input:", inputRef.current.value);
  };

  const focusInput = () => {
    inputRef.current.focus();
  };

  return (
    <div>
      <input type="text" ref={inputRef} />
      <button onClick={getInputValue}>Log Input Value</button>
      <button onClick={focusInput}>Focus Input</button>
    </div>
  );
};

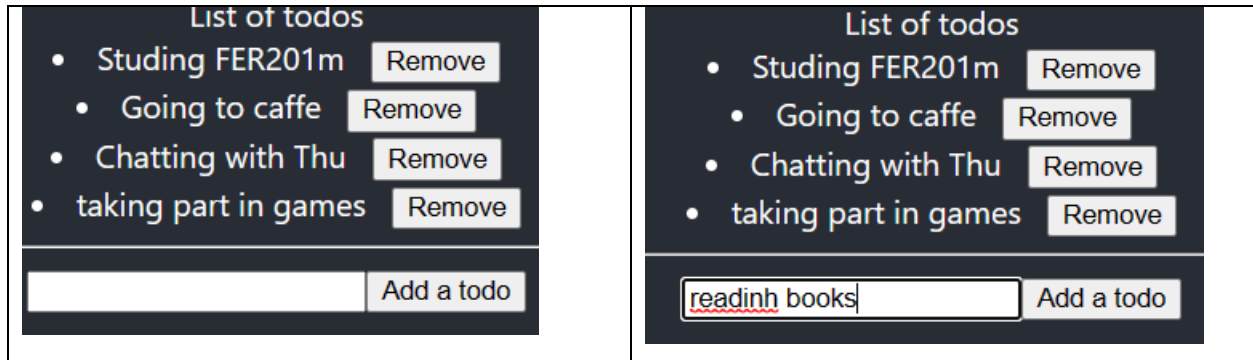
export default RefDomExample;
```

Bài tập 1: Viết Countdown có dạng như sau:



Bài tập 2: Viết 1 List of todos (có Thêm, xóa, hiển thị)

Hiển thị	Thêm
----------	------



Bài 1:

```

Timer
import React, { useState, useEffect } from 'react';

const Timer = (props) => {
  const [count, setCount] = useState(10);

  useEffect(() => {
    if (count === 0) {
      props.onTimesup();
      return;
    }
    let timer = setInterval(() => {
      setCount(count - 1);
    }, 1000);

    return () => {
      clearInterval(timer)
    }
  }, [count]);
  return (
    <div><h1>CountDown</h1>
      <h2>{count}</h2>
    </div>
  )
}
export default Timer;

const onTimesup = () => {
  // alert('times up')
}

<Timer onTimesup={onTimesup} />

```

Bài 2:

```

ListTask
export const ListTask =
[
  { id: 'todo1', title: 'Studing FER201m', type: 'me' },
  { id: 'todo2', title: 'Going to caffe', type: 'me' },

```



```
{ id: 'todo3', title: 'Chatting with Thu', type: 'thu' },
{ id: 'todo4', title: 'taking part in games', type: 'all'
}
```

Task

[illegible]

App.js

```
import Task from './components/ex/Task'
import { ListTask } from './components/ex/ListTask';
import { useState, useEffect } from 'react';
```

```
const [todos, setTodos] = useState(ListTask);
const [input, setInput] = useState('');
useEffect(() => {
  // console.log('use effect todos')
}, [todos]);
const handleClick = (event) => {
  if (!input) {
    alert('empty input')
    return;
  }
  //hook not merge state
  //...spread syntax array js
  let newTodo = {
    id: Math.floor((Math.random() * 100000) + 1),
```

```
        title: input,
        type: 'me'
      }
      setTodos([...todos, newTodo])
      setInput('')
    }

    const handleChangeInput = (event) => {
      setInput(event.target.value)
    }

    const deleteDataTodo = (id) => {
      let currentTodos = todos;
      currentTodos = currentTodos.filter(item => item.id !== id)
      setTodos(currentTodos)
    }
  }
}
```

```
<Task
  todos={todos}
  title={'List of todos'}
  deleteDataTodo={deleteDataTodo}
/>
<input type="text" value={input}
  onChange={ (event) => handleChangeInput(event) } />
<button type="button"
  onClick={ (event) => handleEventClick(event) }>Add a
todo</button>
```

7. Single Page Application (SPA)

Hiện nay khi lướt web, ta sẽ thường bắt gặp 2 dạng Web Application:

- Single Page Application (SPA): Facebook, Youtube, Netflix, ...
- Multi Page Application (MPA): Các trang báo ở Việt Nam như Dantri, Vnexpress, ...

Vậy điểm khác biệt cơ bản giữa chúng là gì?

Với SPA:

- Thông thường, chỉ 1 HTML Page và Content của trang web được trình duyệt tải về khi người dùng truy cập trang web
- Việc định tuyến (Routing) khi người dùng truy cập URL nào đó được thực hiện bởi JavaScript
- Cho người dùng cảm giác web giống như một ứng dụng di động, mượt mà hơn khi sử dụng
- Giảm thiểu gánh nặng xử lý cho Server
- Phân tách rõ ràng FrontEnd (Client) và BackEnd (Server)
- Không tốt cho SEO

Với MPA:

- Mỗi HTML Page, Content của trang web sẽ gắn với một URL, khi người dùng truy cập trình duyệt sẽ tải về các file tương ứng với URL do server cung cấp
- Việc định tuyến được thực hiện bởi Server

- Tốt cho SEO

Ví dụ chúng ta có 2 trang là /user và /product

- Với SPA: Trình duyệt chỉ tải 1 lần 1 file HTML và content của toàn bộ trang web, việc hiển thị UI tương ứng với URL sẽ do JavaScript đảm nhiệm, nếu URL là /user JavaScript sẽ chỉ render phần UI của /user

- Với MPA: Trình duyệt sẽ tải file HTML và content do server trả ra tương ứng với URL, có nghĩa là trình duyệt sẽ tải file HTML và content tương ứng khi truy cập /user và tiếp tục tải file HTML và content tương ứng khi truy cập /product

Tips: Khi chuyển qua lại giữa các tab, navigation, ... của một trang web và thấy URL trình duyệt thay đổi nhưng trang web không reload, đó có thể là dấu hiệu của SPA

8. Routing trong ReactJS

Khi truy cập 1 trang web. Nhập đường dẫn (url) sẽ load(render) ra các trang(Component) tương ứng. Để làm được điều này ta cần quản lý routing trong ứng dụng web của chúng ta. Trong reactjs ta có thể dùng thư viện react-router-dom để làm việc này.

Để cài đặt ta sử dụng lệnh:

```
yarn add react-router-dom
```

Hoặc

```
npm i react-router-dom
```

a. Các component cơ bản

- **Router:** Component bao bọc tất cả component khác của routing. Có rất nhiều loại router:
 - Browser Router: Thông dụng nhất, sử dụng HTML5 history API để đồng bộ UI và URL
 - Hash Router: Loại routing có dấu # ở đầu, công dụng để thấy nhất đó là giúp việc deploy trên những trang có nguồn tài nguyên được chia sẻ giữa nhiều người dùng như Github Pages đơn giản hơn
 - Static Router: Thường sử dụng cho Server Side Rendering
- **MemoryRouter:** Thường sử dụng khi Testing, thanh địa chỉ sẽ không được cập nhật URL mà giữ trong memory (Hữu ích khi sử dụng React Native)
- **Route:** Component này dùng để render component nếu đường dẫn url trùng với path props của Route component.
- **Switch:** Bao ngoài Route, chỉ render Route đầu tiên match với path URL
- **Redirect:** Điều hướng từ một path này qua path khác, thường đi kèm với điều kiện cụ thể (VD: người dùng chưa login thì khi truy cập web sẽ redirect qua path **/unauthorized**)
- **Link:** Thẻ chuyển hướng tới path tương ứng
- **NavLink:** Tương tự Link nhưng có thêm activeClassName (Hữu ích trong việc CSS tab nào đang active)

Ví dụ

Có <BrowserRouter> bao bọc tất cả component, nếu không có sẽ xảy ra lỗi.

```
function App() {  
  return (  
    <div className="App">  
      <BrowserRouter></BrowserRouter>  
    </div>  
  )  
}
```

```

    <Route path="/test" component={<div>test</div>} />
  </div>
);
}

```

Uncaught runtime errors:

ERROR

```

Invariant failed: You should not use <Route> outside a <Router>
    at invariant (http://localhost:3000/static/js/bundle.js:45213:9)
    at http://localhost:3000/static/js/bundle.js:36647:86
    at updateContextConsumer (http://localhost:3000/static/js/bundle.js:27583:23)
    at beginWork (http://localhost:3000/static/js/bundle.js:27956:18)
    at HTMLUnknownElement.callCallback (http://localhost:3000/static/js/bundle.js:12919:18)
    at Object.invokeGuardedCallbackDev (http://localhost:3000/static/js/bundle.js:12963:20)
    at invokeGuardedCallback (http://localhost:3000/static/js/bundle.js:13020:35)
    at beginWork$1 (http://localhost:3000/static/js/bundle.js:32894:11)
    at performUnitOfWork (http://localhost:3000/static/js/bundle.js:32141:16)
    at workLoopSync (http://localhost:3000/static/js/bundle.js:32064:9)

```

Ví dụ: cơ bản về React Router, ta có các Components bao gồm:

- Một header để điều hướng trang web
- Tab đang active sẽ có màu vàng, tab không active sẽ có màu xanh
- Click tab sẽ render ra UI tương ứng, ngoại trừ tab Unauthorized, khi click tab này sẽ tự động chuyển hướng đến Redirected Page nhờ component Redirect
- Page Product sẽ có 1 vài product, khi click sẽ chuyển đến trang tương ứng và hiện ra id của product

Chú ý:

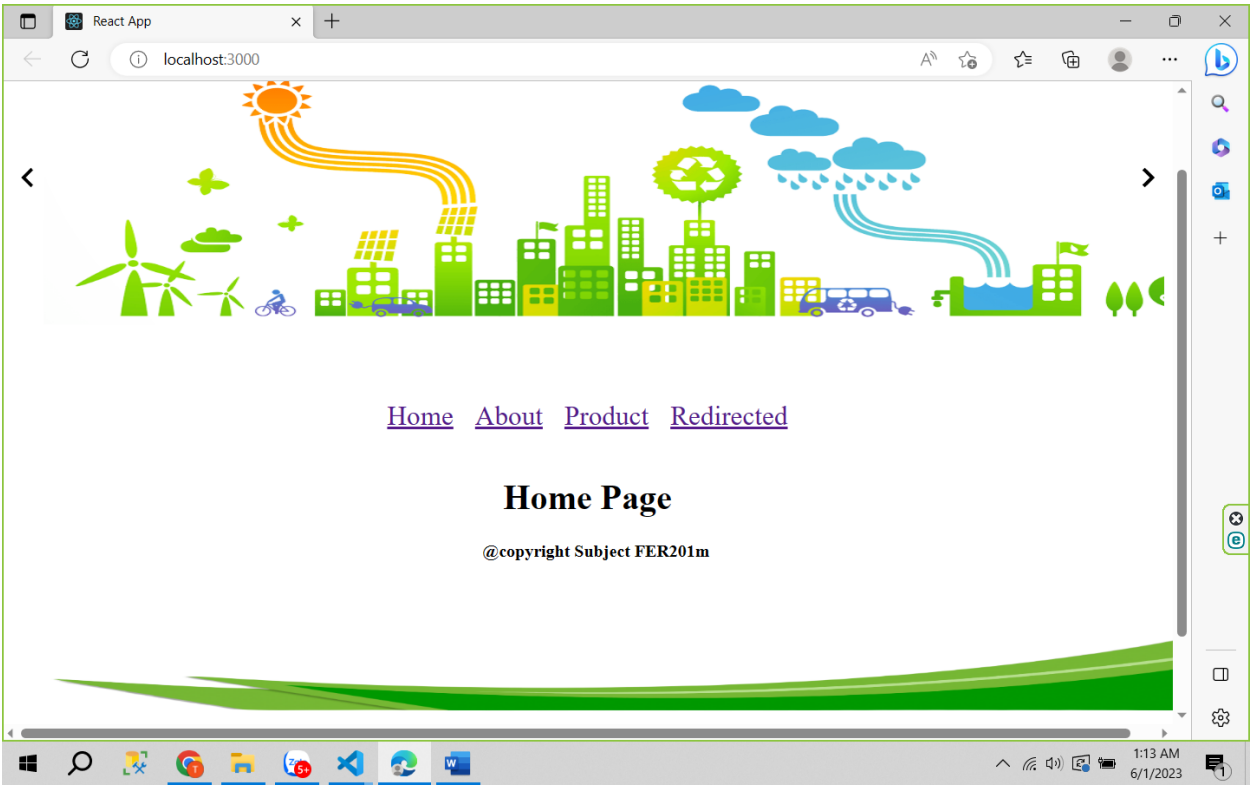
<Switch> (react-router v5) thì dùng từ khoá 'exact' ở 2 path / và /home vì khi không có từ khóa 'exact', khi qua path /home, nó sẽ match ngay với path / ở Route, khiến cho HomePage được render thay vì AboutPage. Còn khi thay bằng <Routes> (react-router v6) thì không xảy ra như vậy.

Ví dụ:

Switch	Routes
<pre> import {BrowserRouter, Switch, Route} from 'react-router-dom' function App() { return(<BrowserRouter> <Switch> <Route exact path = "/> <Main Page /> </Route> <Route path = "/user"> <User /> </Route> </pre>	<pre> import { BrowserRouter, Routes, Route} from 'react-router-dom'; function App() { return (<BrowserRouter> <Routes> <Route path="/" element={<Main Page />} /> <Route path="user/" element={<User />} /> </pre>

<pre> </Switch> </BrowserRouter>); }</pre>	<pre> </Routes> </BrowserRouter>); }</pre>
--	--

Thực hiện để có kết quả như sau:



Các Components:

AboutPage	HomePage
<pre>import React from "react"; const AboutPage = () => { return (<div> <h1>About Page</h1> </div>); }; export default AboutPage;</pre>	<pre>import React from "react"; const HomePage = () => { return (<div> <h1>Home Page</h1> </div>); }; export default HomePage;</pre>
RedirectedPage	UnauthorizedPage
<pre>import React from "react"; const RedirectedPage = () => { return (<div></pre>	<pre>import React from "react"; import { Redirect } from "react-router";</pre>

<pre> <h1>Redirected Page</h1> </div>); }; export default RedirectedPage; </pre>	<pre> const UnauthorizedPage = () => { return <Redirect to="/redirected" />; }; export default UnauthorizedPage; </pre>
<pre> ProductPage import React from "react"; import { Link } from "react- router-dom"; const ProductPage = () => { return (<div style={{ display: "flex", flexDirection: "column" }}> <Link to="/product/1">Product 1</Link> <Link to="/product/2">Product 2</Link> <Link to="/product/3">Product 3</Link> </div>); }; export default ProductPage; </pre>	<pre> ProductDetailPage import React from "react"; import { useParams } from "react-router"; const ProductDetailPage = () => { const { id } = useParams(); return <div>Đây là Product ID: {id}</div>; }; export default ProductDetailPage; </pre>
<pre> App.css .App { text-align: center; } .header { list-style: none; display: flex; justify-content: center; flex-direction: row; padding: 0; } </pre>	<pre> .header-item { margin: 20px 10px; font-size: 25px; } .active-header-item { color: rgb(255, 196, 0); } </pre>

Banner:

<pre> banner.css img { height: 40vh; margin-left: 5vh; width: 180vh; } .each-fade { margin-top: 10vh; margin-bottom: 5vh; } </pre>

```

}

import React from "react";
import { Fade } from 'react-slideshow-image';
import 'react-slideshow-image/dist/styles.css';

import './banner.css'
const fadeImages = [
  {
    url: '/images/slide_1.png',
    caption: 'Slide 1'
  },
  {
    url: '/images/slide_2.png',
    caption: 'Slide 2'
  },
  {
    url: '/images/slide_3.png',
    caption: 'Slide 3'
  },
];

const Banner = () => {
  return (
    <div>
      <Fade>
        {fadeImages.map((fadeImage, index) => (
          <div className="each-fade" key={index}>
            <div className="image-container">
              <img src={fadeImage.url} />
            </div>
          </div>
        ))}
      </Fade>
    </div>
  )
}

export default Banner;

```

```
npm i react-slideshow-image -S
```

Chân trang

```

footer.css

footer {
  display: flex;
  align-items: center;
  position: relative;
  margin-top: -9rem;
  margin-bottom: -2rem;
}

.f-content {

```

```

    position: absolute;
    display: flex;
    align-items: center;
    justify-content: center;
    flex-direction: column;
    width: 100%;
    gap: 4rem;
  }

import React from 'react'
import Wave from '../images/wave.png'
import './footer.css'

const Footer = () => {
  return (
    <div className="footer">
      <div className="f-content">
        <span style={{ fontWeight: 'bold' }}>@copyright
Subject FER201m</span>
      </div>
      <img src={Wave} alt=""
        style={{ width: '100%' }}
      />
    </div >
  )
}

export default Footer

```

Các phần tiếp:

```

App
import './App.css';
import React from "react";

import Banner from './components/Banner';
import AboutPage from './components/AboutPage';
import HomePage from './components/HomePage';
import ProductPage from './components/ProductPage';
import RedirectedPage from './components/RedirectedPage';
import { NavLink, Routes, Route } from 'react-router-dom';

function App() {
  return (
    <div className="App">
      <Banner />
      <ul className="header">
        <li className="header-item">
          <NavLink to="/" activeClassName="active-header-item">
            Home
          </NavLink>

```



```

        </li>
        <li className="header-item">
            <NavLink to="/about" activeClassName="active-header-
item">
                About
            </NavLink>
        </li>
        <li className="header-item">
            <NavLink to="/product" activeClassName="active-header-
item">
                Product
            </NavLink>
        </li>

        <li className="header-item">
            <NavLink
                to="/redirected"
                activeClassName="active-header-item"
            >
                Redirected
            </NavLink>
        </li>
    </ul>
    <Routes>
        <Route path="/" element={<HomePage />} />
        <Route path="/home" element={<HomePage />} />
        <Route path="/about" element={<AboutPage />} />
        <Route path="/product" element={<ProductPage />} />
        <Route path="/redirected" element={<RedirectedPage />} />
    </Routes>
</div>
);
}
export default App;

```

Cách khác:

```

Routes.js
import { createBrowserRouter } from "react-router-dom";
import Home from "../pages/Home/Home";
import SignIn from "../pages/SignIn/SignIn";
import SignUp from "../pages/SignUp/SignUp";

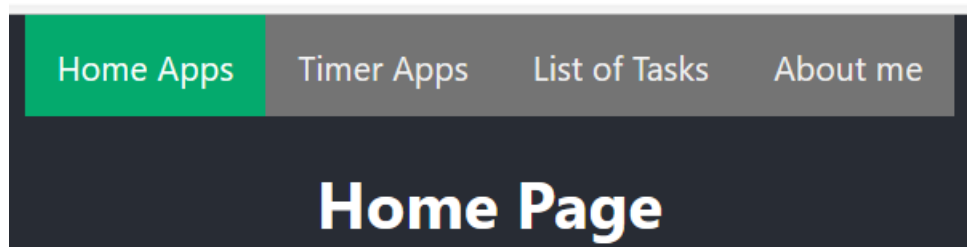
const Routes = createBrowserRouter([
    {
        path: "/",
        element: <Home/>
    },
    {
        path: "/login",
        element: <SignIn />
    },
]);

```

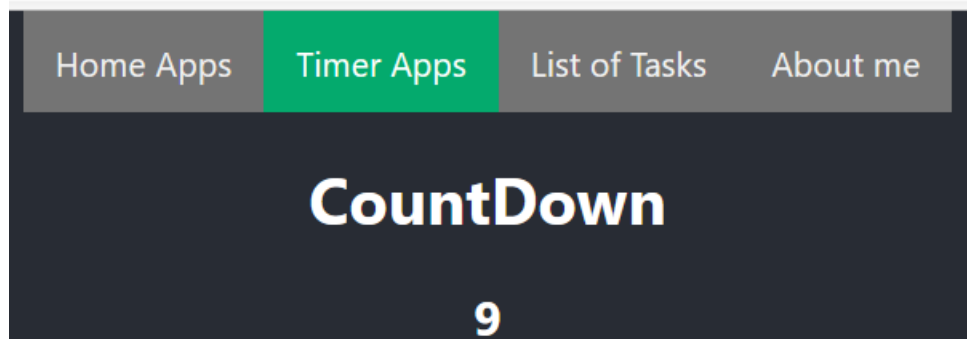
```
    {
      path: "/signup",
      element: <SignUp />
    }
  ]);

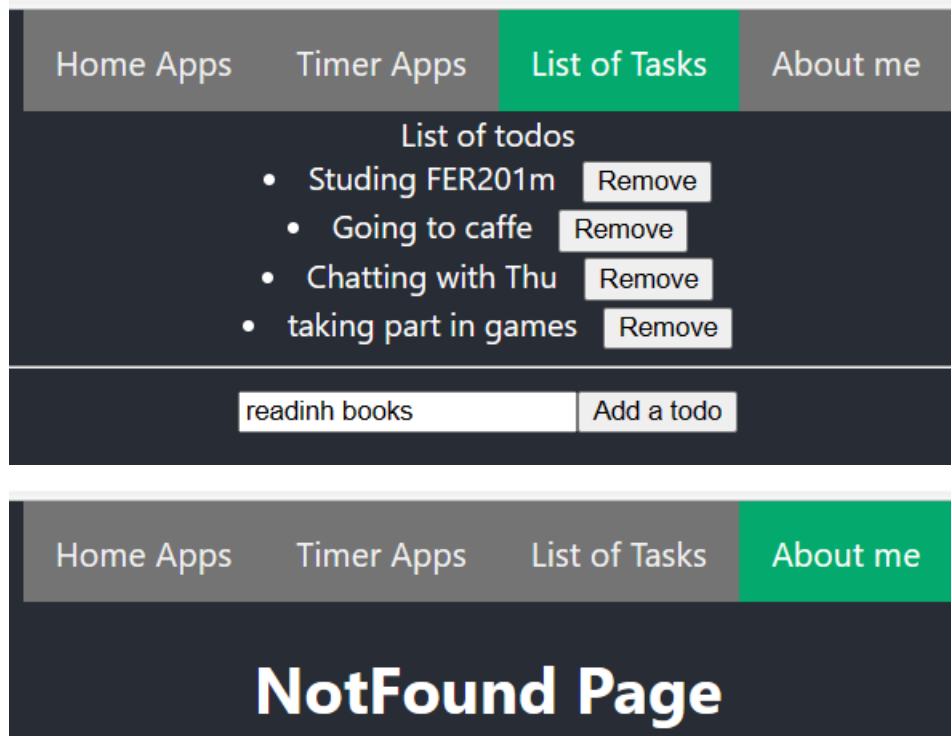
export default Routes;
App
import { RouterProvider } from "react-router-dom";
import routes from "../Router";
const App = () => {
  return (
    <RouterProvider router={routes} />
  );
};
export default App;
```

Bài tập: Viết 1 app có dạng như sau



Các chức năng tiếp theo lấy từ bài tập 1 và 2:





9. Controlled Forms

Trong React.js, có hai phương pháp để xử lý biểu mẫu: biểu mẫu kiểm soát (controlled forms) và biểu mẫu không kiểm soát (uncontrolled forms). Sự khác biệt chính giữa chúng nằm ở cách quản lý và cập nhật dữ liệu biểu mẫu.

Biểu mẫu kiểm soát là các biểu mẫu mà dữ liệu biểu mẫu được quản lý bởi các **React components**. Trong **Controlled Forms**, các phần tử form input (ví dụ: **input fields**, **checkboxes**, **selects**) được liên kết với **component's state**, và bất kỳ thay đổi nào trong các phần tử input đều kích hoạt việc cập nhật **component's state**. Sau đó, **components** được cập nhật và hiển thị dữ liệu biểu mẫu đã được cập nhật.

Đặc điểm quan trọng của Controlled Forms

- Dữ liệu biểu mẫu được lưu trữ trong component's state.
- Các trường nhập liệu nhận giá trị từ state.
- Các thay đổi trong các trường nhập liệu được xử lý bởi các trình xử lý sự kiện, cập nhật state với các giá trị mới.
- Component được cập nhật với dữ liệu biểu mẫu đã được cập nhật.

Ví dụ 1:

```
const [inputa, setInputa] = useState(0);
const handleOnChangeInputa = (e) => {
  setInputa(e.target.value)
}
<div>
  <div>
```

```
<label htmlFor="inputa">Input number 1:</label>
<input type="number" value={inputa}
  onChange={(e) => handleOnChangeInputa(e)} />
</div>
<h2>Number 1: {inputa}</h2>
</div>
```

Ví dụ 2:

```
.table-student {
  width: 100%;
  border: 1px solid #ccc;
}
.head-student {
  font-weight: 600;
}
.head-student tr,
.body-student tr {
  display: flex;
  align-items: center;
  padding-left: 20px;
}
.head-student td,
.body-student td {
  display: flex;
  align-items: center;
  justify-content: center;
  padding: 12px 0;
  flex: 1;
}
.body-student tr:nth-child(2n+1) {
  background-color: #ddd;
}
```

Lợi ích của controlled forms

- Có đầy đủ kiểm soát về dữ liệu biểu mẫu và có thể thực hiện validations, transformations, hoặc xử lý các logic khác trước khi cập nhật state hoặc gửi biểu mẫu.
- Có thể dễ dàng triển khai các tính năng như validations form, hoặc phản hồi thời gian thực cho người dùng.
- Dữ liệu biểu mẫu có thể dễ dàng điều chỉnh hoặc xử lý trước khi gửi đến máy chủ.

Bất lợi của controlled forms

- Controlled forms có thể yêu cầu nhiều code và xử lý sự kiện hơn so với Uncontrolled forms.
- Nếu có một biểu mẫu lớn với nhiều trường nhập liệu, việc quản lý state cho mỗi trường có thể trở nên phức tạp.

10.Uncontrolled Forms

Uncontrolled Forms là biểu mẫu trong đó dữ liệu biểu mẫu được xử lý bởi các phần tử DOM, thay vì được điều khiển bởi các React component. Trong Uncontrolled Forms, các trường nhập liệu không được liên kết một cách rõ ràng với component's state. Thay vào đó, chúng ta sử dụng các tham chiếu (refs) để truy cập giá trị trường nhập liệu khi cần thiết.

Đặc điểm quan trọng của Uncontrolled Forms

- Dữ liệu biểu mẫu được quản lý trực tiếp bởi các phần tử DOM.
- Trường nhập liệu được truy cập bằng cách sử dụng tham chiếu (refs) để lấy giá trị khi cần thiết.
- Không có quản lý state rõ ràng cho dữ liệu biểu mẫu.
- Các thay đổi trong trường nhập liệu không được theo dõi hoặc xử lý một cách rõ ràng bởi các React components.

Ví dụ:

```
import React, { useRef } from "react";

function UncontrolledForms() {
  const nameInputRef = useRef();
  const emailInputRef = useRef();
  const passwordInputRef = useRef();
  const handleSubmit = (e) => {
    e.preventDefault();
    // Access form data using refs
    const name = nameInputRef.current.value;
    const email = emailInputRef.current.value;
    const password = passwordInputRef.current.value;
    // Perform form submission logic here
    console.log(name, email, password);
    alert(name + " " + email + " " + password);
    // Reset the form
    e.target.reset();
  };
  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" ref={nameInputRef} />
      </label>
      <br />
      <label>
        Email:
        <input type="email" ref={emailInputRef} />
      </label>
      <br />
      <label>
        Password:
        <input type="password" ref={passwordInputRef} />
      </label>
    <br />
  )
}
```

```
        <button type="submit">Save</button>
      </form>
    );
  }
  export default UncontrolledForms;
```

Lợi ích của Uncontrolled forms

- Biểu mẫu không được điều khiển đơn giản hơn và yêu cầu ít code hơn so với controlled forms.
- Chúng hữu ích cho các biểu mẫu đơn giản không yêu cầu quản lý state phức tạp.
- Dễ dàng truy cập trực tiếp giá trị trường nhập liệu khi cần thiết, ví dụ như trong quá trình gửi biểu mẫu.

Bất lợi của Uncontrolled forms

- Bị hạn chế về việc kiểm soát dữ liệu biểu mẫu và không thể dễ dàng thực hiện kiểm tra hoặc biến đổi dữ liệu trước khi gửi biểu mẫu.
- Có thể khó khăn khi triển khai các tính năng biểu mẫu nâng cao như validations thời gian thực hoặc tương tác giữa các trường phụ thuộc.
- Việc thao tác hoặc xử lý dữ liệu biểu mẫu trước khi gửi nó đến máy chủ có thể đòi hỏi các bước bổ sung.

11.Template cho app

Cài đặt

Cài Bootstrap 4

```
npm i bootstrap@4.6.0
npm i reactstrap react react-dom
npm i popper.js@1.16.1
```

Cấu hình sử dụng Bootstrap 4

index.js

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

Nếu sử dụng:

```
import { Navbar, NavbarBrand } from 'reactstrap';
Ví dụ:
<Navbar dark color="primary">
  <div className="container">
    <NavbarBrand href="/">Ristorante Con
  </div>
</Navbar>
```

```
import { Card, CardImg, CardImgOverlay, CardText, CardBody,
  CardTitle } from 'reactstrap';
```

Ví dụ:

```
renderDish(dish) {
  if (dish !== null)
    return(
```

```
        <Card>
          <CardImg top src={dish.image} alt={dish.name}
/>
          <CardBody>
            <CardTitle>{dish.name}</CardTitle>
            <CardText>{dish.description}</CardText>
          </CardBody>
        </Card>
      );
    } else
      return (
        <div></div>
      );
    }
    const menu = this.props.dishes.map((dish) => {
      return (
        <div className="col-12 col-md-5 m-1">
          <Card key={dish.id}
            onClick={() => this.onDishSelect(dish)}>
            <CardImg width="100%" src={dish.image}
alt={dish.name} />
            <CardImgOverlay>
              <CardTitle>{dish.name}</CardTitle>
            </CardImgOverlay>
          </Card>
        </div>
      );
    });
```

Sử dụng: Font Awesome Icons and Bootstrap-Social

Npm i font-awesome@4.7.0

Npm i [bootstrap-social@5.1.1](#)

npm i react-slideshow-image -S

Cấu hình : index.js

```
import 'font-awesome/css/font-awesome.css';
```

```
import 'bootstrap-social/bootstrap-social.css';
```

Cài đặt React Router

npm i react-router-dom

12.Redux

13.Redux form

14.JSON Server

15.React Animation