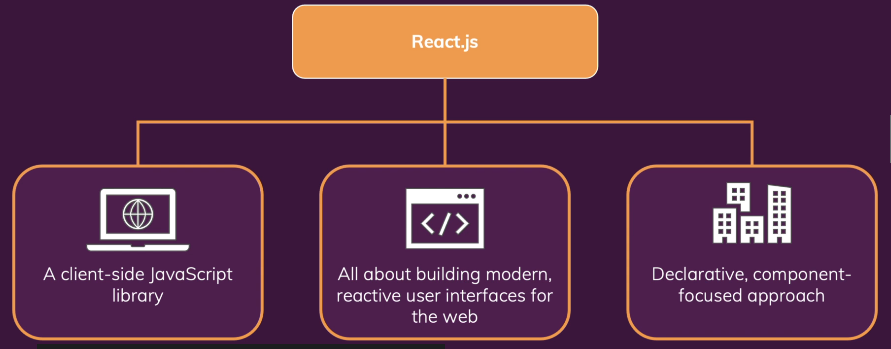
Reacts

2 What is Reacts.js



+Là một thư viện GUI Javascript mã nguồn mở

+Phân loại thành kiểu "V" trong mô hình MVC

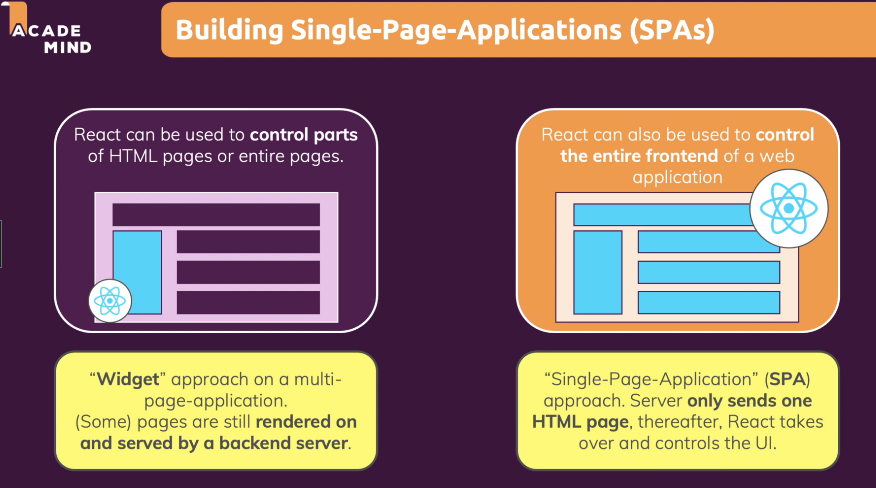
+Cho phép tái sử dụng lại component có cùng chức năng

+Cho phép sữ dụng tiện ích mở rộng kiểu JSX (Kiểu kết hợp giữa HTML và Javascript)

+React sử dụng virtual DOM để tối ưu hóa trang

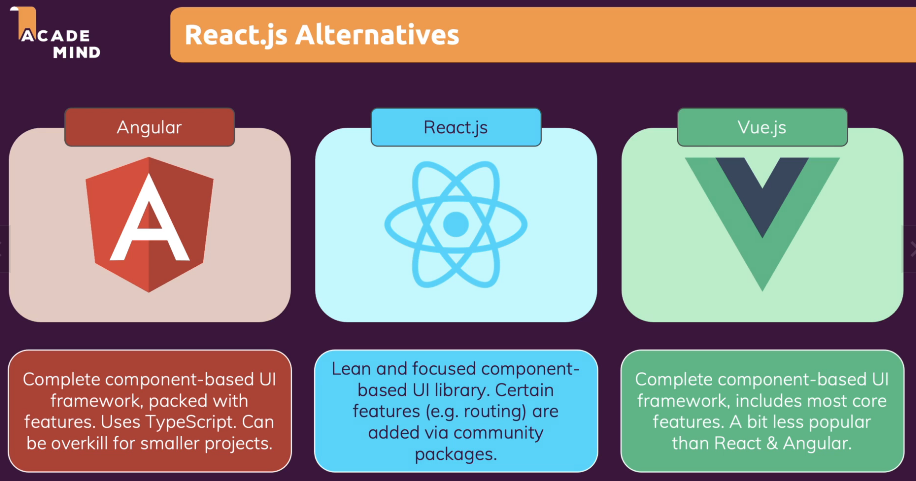
4. Building Single-page Applications

SPAS (Building Single-Page-Application)



+ Không cần phải load lại trang trong quá trình sử dụng trình duyệt (chỉ sữ dụng 1 trang duy nhất)

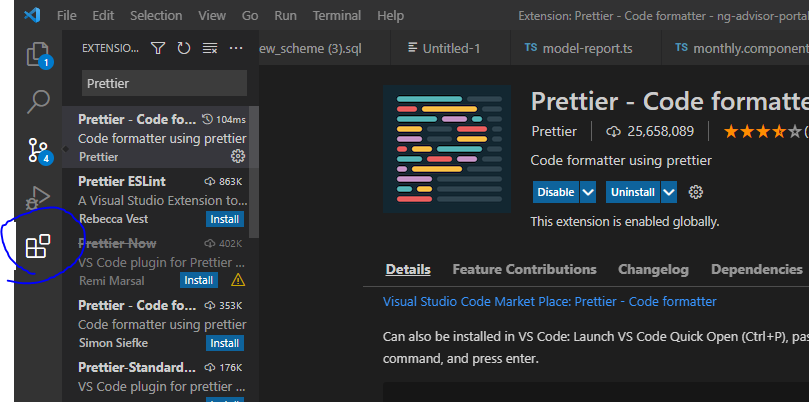
5.Exploring React.js



6. Install extension before code react

+ Prettier: make code more look easily when use div (use shotcut alt + shift + f)

Step 1:

Click 

Then search library Pretties and download

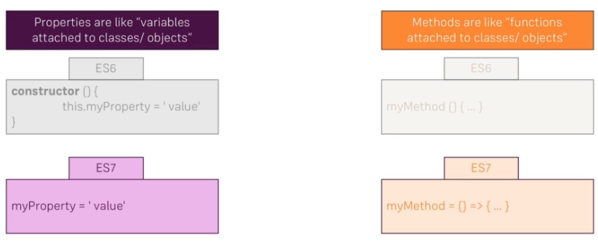
Another library if you want install:

+ Format HTML Format

+ Material Icon Theme

JavaScript Basic

+ ES6 được ra đời vào năm 2015 nên nó có thêm cái tên là ES2015. Tương tự như vậy ES7 hay còn gọi là ES2016.



12. Understanding Let const

Var: Một biến cũ của java script, có thể gán số hay string, có thể thay đổi giá trị

Sau này let và const ra đời thay thế cho Var

* **const** không thể gán giá trị mới cho biến sau khi khai báo.
* **let** có thể gán giá trị mới cho biến sau khi khai báo.

13. Arrow Functions

 Một dạng khai báo hàm rút gọn trong javascript trong JavaScript.

Bình thường chúng ta khai báo hàm dạng:

function dude() {

console.log('hey dude')

}

Nhưng đối với ES6 chúng ta có thể khai báo hàm như sau

const dude = () => {

console.log('hey dude')

}

Đối với hàm có tham số:

const dude = (name, age) => { console.log(name + ' is ' + age) }

Đối với những hàm có một tham số thì ta có thể bỏ dấu ngoặc đơn đi:

const dude = name => { console.log('hey ' + name) }

Với hàm mũi tên trong ES6, giá trị của this chính là this trong tầm vực gần nhất với nó

function App() {

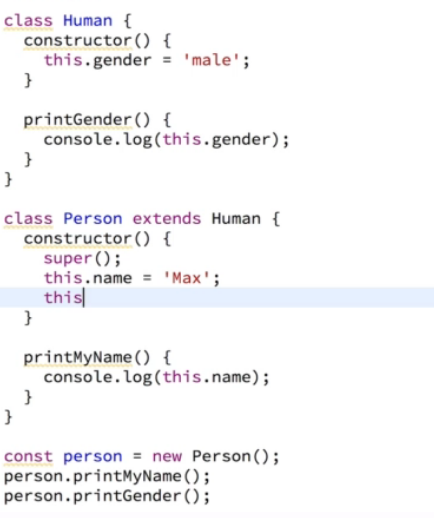
this.count = 0

setInterval(() => console.log(this.count++), 1000)

}

14. Export & Imports

+ Khi có nhiều file js và muốn sử dụng một số chức năng của js

15. Understanding Classes

Hoạt động giống như java

+Bao gồm thuộc tính và phương thức

+Hỗ trợ kế thừa class

16. Classes, Properties &methods



17. Spread & Rest Operators

Spread:

+Xài theo ký hiệu dấu ba chấm

+Ra đời ES6

+Cho phép:

* Sao chép một mảng
* Tách hoặc kết hợp một hay nhiều mảng
* Sử dụng mảng như danh sách các argument
* Thêm một item vào một list
* Thao tác với state trong React
* Kết hợp các objects

Ví dụ nếu ko có spread:

Problem

Math.max(1,3,5) // output: 5

Math.max([1,3,5]) // output: NaN.

Chúng ta mong muốn rằng hàm này sẽ hiểu rằng chúng ta truyền vào 3 tham số riêng biệt, và tìm số lớn nhất trong 3 số này (như cách viết trong dòng thứ 2). Tất nhiên là nếu chúng ta viết như vậy thì hàm sẽ không hiểu được rồi, và sẽ cho ra output là NaN

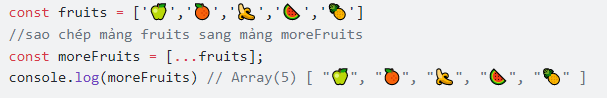
Solved

Math.max(...[1,3,5]) // output: 5

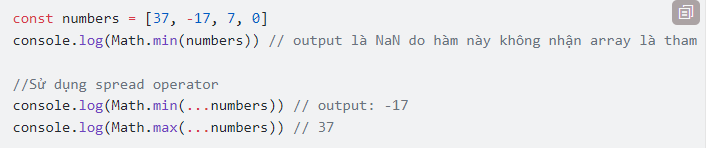
* Trong trường hợp này, spread operator đã mở rộng (spread) mảng 3 phần tử thành 3 tham số riêng biệt.

Một số ví dụ khác về spread

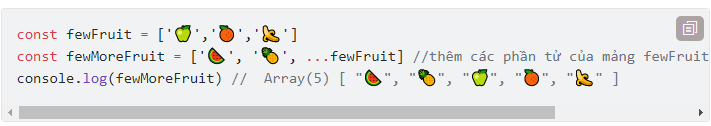
1. Sao chép mảng



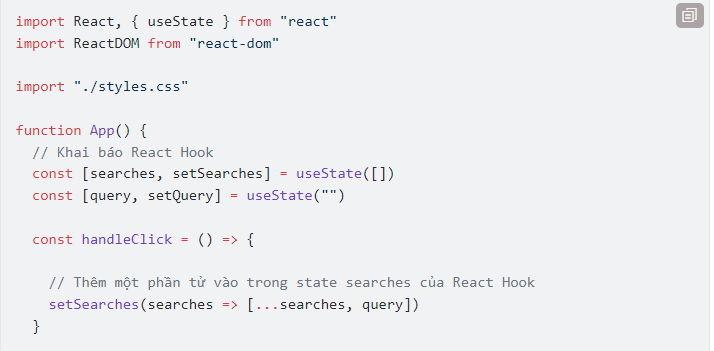
1. Sử dụng mảng vào tham số



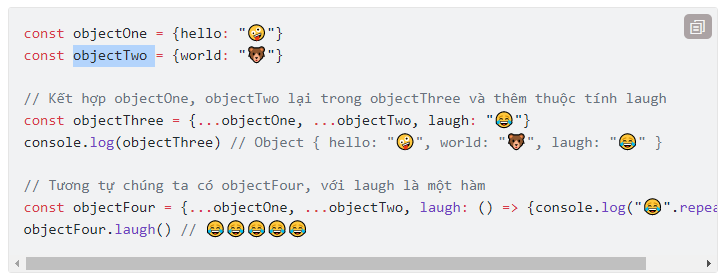
1. Thêm phần tử vào mảng



1. Thao tác với state

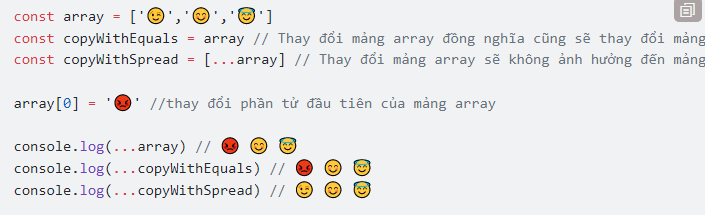


1. Kết hợp 2 hay nhiều object với nhau



Một số lưu ý khi sử dụng spread

Chúng sẽ tạo ra một tham chiếu mới, sau đó sao chép giá trị của tham chiếu cũ vào tham chiếu mới này. Khi đó, mọi thao tác làm thay đổi tham chiếu cũ sẽ không ảnh hưởng đến mảng được sao chép, điều mà sẽ xảy ra nếu chúng ta sao chép mảng sử dụng phương thức gán =



18. Destructing

Là một cú pháp cho phép gán các thuộc tính của một Object hoặc một Array

### Object Destructuring

const person = {

name: 'Dam SamNang',

age: 29,

location: {

country: 'Vietnam'

company: 'Framgia VN'

}

};

# cách không dùng destructuring object

const ten = person.name;

const tuoi = person.age;

const nuoc = person.location.country;

const congty = person.location.company;

# cách dùng destructuring object

const { name, age } = person;

console.log(`${name} is ${age}`);

=> Dam SamNang is 29

const { country, company } = person.location;

console.log(`${company} is located in ${country}`);

=> Framgia VN is located in Vietnam

const { otherPropertise, name, age } = person;

console.log(otherPropertise);

=> otherPropertise sẽ trả về bằng undefined

* Cách thay đổi tên destructuring object, có cách viết như sau:

const { propertise\_name: new\_name\_propertise } = object;

Vi dụ:

const { name: ten } = person;

console.log(name) # sẽ có lỗi không tìm thấy name

console.log(ten) # Dam SamNang

* Cách gán giá trị bắt đâu (Default value), có cách viết như sau:

const { propertise\_name = default\_value } = object;

Vi dụ:

const person = {

age: 29,

location: {

city: 'Hanoi'

company: 'Framgia'

}

};

const { name = 'Unknow' } = person;

console.log(name);

=> Unknow

* Cách viết default value và thay đổi tên của propertise, có cách viết như sau:

const { propertise\_name: new\_name\_propertise = default\_value } = object

Vi dụ:

const person = {

age: 29,

location: {

city: 'Hanoi'

company: 'Framgia'

}

};

const { name: firstName = 'Unknow' } = person;

console.log(firstName);

=> Unknow

### Array Destructuring

* ***Array Destructuring***: có ý tưởng giống nhau nhưng khác viết.Cách viết array destructuring sẽ giúp chúng ta dễ hiểu biết càng rõ hơn về thuộc tính của array mà chúng ta đang dùng đó là có mục đích như thể nào.

# không sử dùng array destructuring

const colors = ['blue', 'red', 'pink'];

console.log(`These colors : ${colors[1]} and ${colors[2]}.`);

These colors: red and pink.

# không sử dùng array destructuring

const do = colors[1];

const hong = colors[2];

console.log(`These colors : ${do} and ${hong}.`);

These colors: red and pink.

# sử dùng array destructuring

const [blue, red, pink] = colors;

console.log(`These colors : ${red} and ${pink}.`);

These colors: red and pink.

* **Array destructuring không đây đủ**: đối với array cùng cho phép chúng ta destructuring gán gía trị đối với trường hợp như sau: Vi dụ:

const colors = ['blue', 'red', 'pink'];

const [, red] = colors;

console.log(red); // kết quả: red

const [, , pink] = colors;

console.log(pink); // kết quả: pink

const [, , , otherColor] = colors;

console.log(otherColor); // kết quả: undefined

* **Array destructuring set default value**: chúng ta có thể gán giá trị bán đầu, có cách viết như sau:

const [ object\_name = default\_value ] = array;

Vi dụ:

const colors = [];

const [pink = 'pink'] = colors;

console.log(pink); // kết quả: pink

## Template literals

Nếu không dùng ES6 thì

function myDetails(name, age, city) {

console.log(name + ' is ' + age + ' and lives in ' + city) // khá khó nhìn và cũng khó code

}

Còn đối với ES6 thì

function myDetails(name, age, city) {

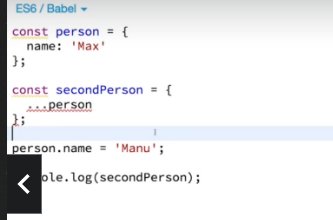
console.log(`${name} is ${age} and lives in ${city}`)

}

19. Reference and primitive types refresher

+Không dùng kiểu copy secondPerson = person; vì khi thay đổi person thi secondPerson sẽ thay đổi theo

Solved (nên dung Spread để ko bị ảnh hưởng bởi object cũ)



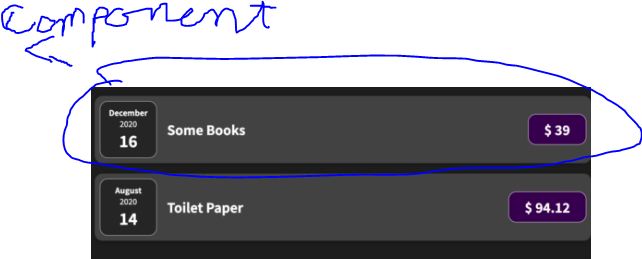
File Tóm tắt javascript đã gửi trong email

* File next-gen-summary.docs

25. What are component

What is component ?

+Cho phép tái sử dụng lại component có cùng chức năng



* Make reusability, no repeat html, css, no too many things in one and the same place

26. How is a component build and declarative way

What is declarative way

=> Bạn có thể hình dung là với Imperative Programming thì bạn quan tâm tới việc làm thế nào để giải quyết bài toán còn Declarative Programming quan tâm tới đầu vào và đầu ra của bài toán.

ví dụ viết theo Imperative Programming sau:

let array = [1, 2, 3, 4, 5, 6]

var reduced = 0

var filtered = []

for element in array {

reduced += element

if element % 2 == 1 {

filtered.append(element)

}

}

Và khi viết lại với Declarative Programming:

let numbers = [1, 2, 3, 4, 5, 6]

let sum = reduce(numbers, 0, +)

let odds = filter(numbers, { $0 % 2 == 1})

Rõ ràng code trở lên mạch lạc và rõ ràng hơn rất nhiều.

27.Create a new react project

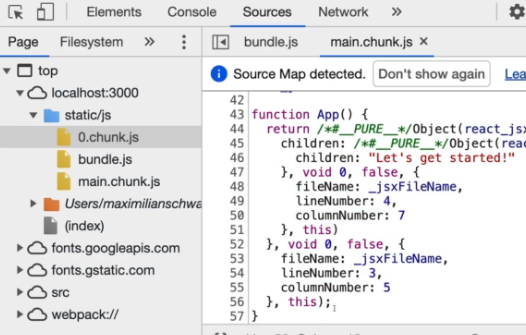
+Install nodejs <https://nodejs.org/en/download/> = > tsl 16.18.0

+Link git <https://github.com/academind/react-complete-guide-code/tree/03-react-basics-working-with-components/code>

+Then install npx create-react-app react-complete-guide -> cd react-compelete-build -> npm start

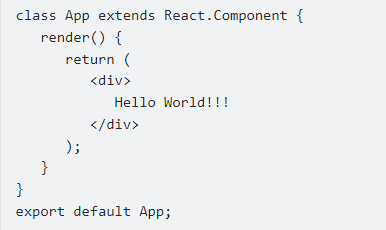
31. JSX

It transform you file html to JSX when run in brower

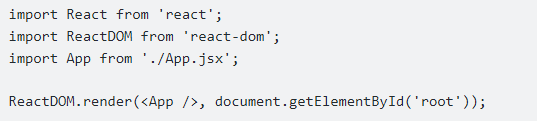


32. How React Works

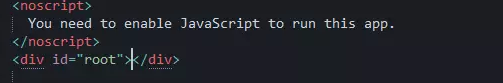
Step 1 Xây dựng App.js trước



Step 2 sau đó ở index.js nó gọi App thông qua ReactDOM.render

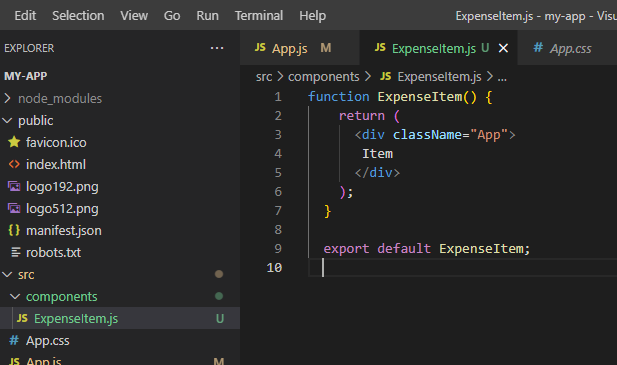


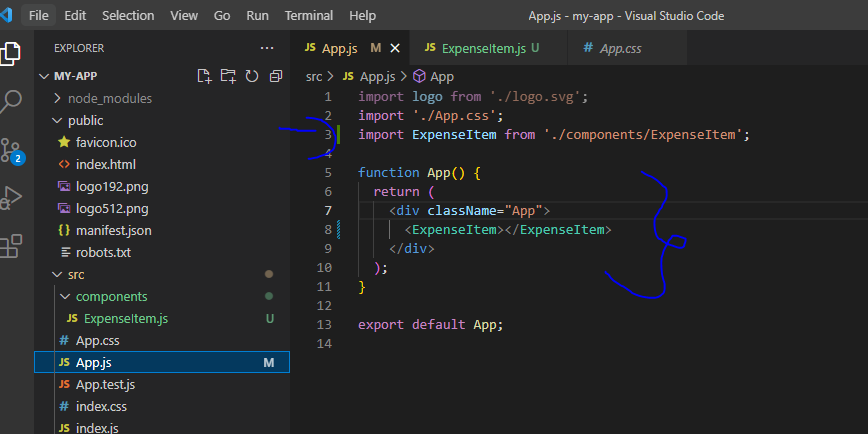
Step 3 sau đó react sẽ tìm id qua root, qua file index.html như hình



33.Buildindg a first custom component

+ Component just a JavaScript function return html





Guild to build

Step 1: run npx create-react-app my-app

Step 2: create folder components and js like picture

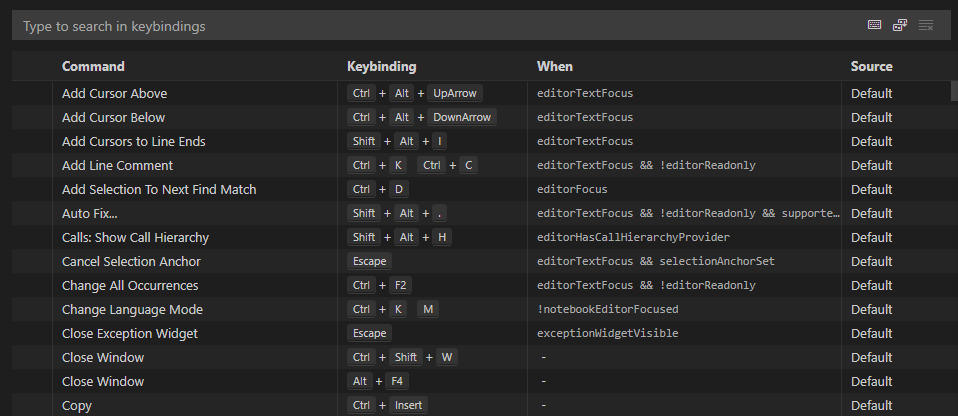
(remember import React)

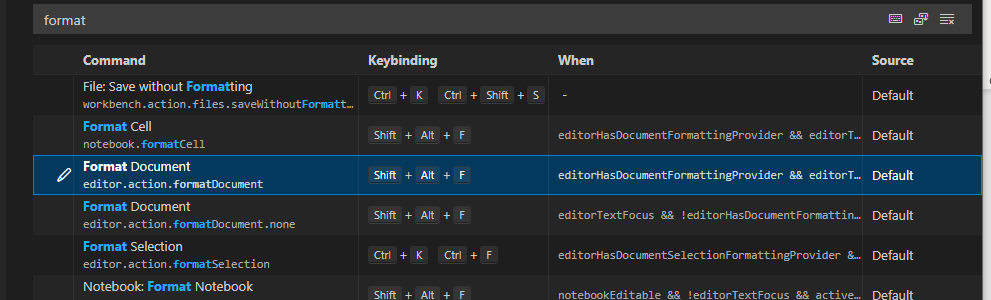
Step 3: run command ‘npm start’

34. Shorcut keyboard

File -> Prefenrence -> keyboard

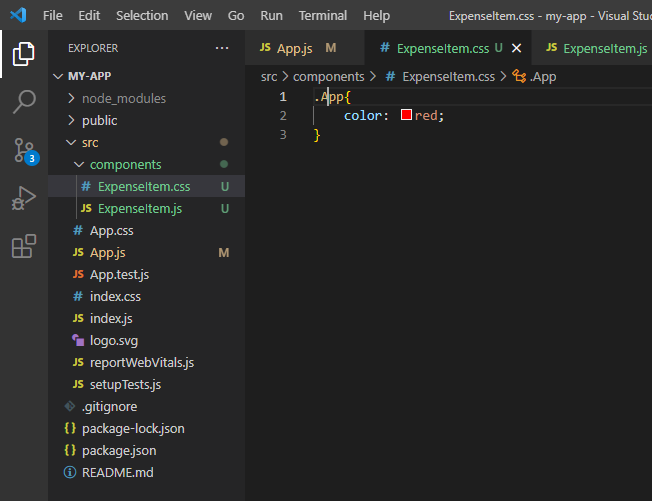
Alt + Shift + F: giúp clear code



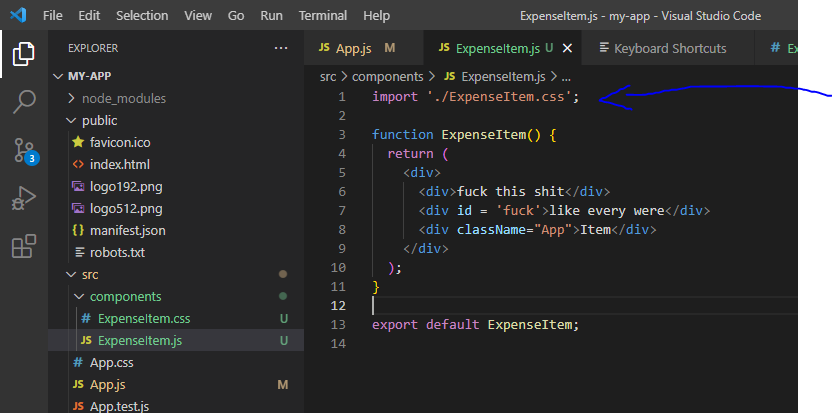


35. Adding Basic CSS Styling

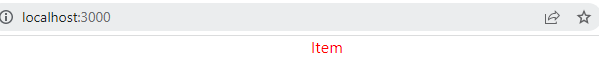
Step 1



Step 2



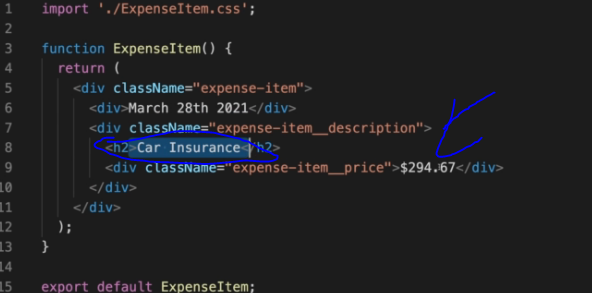
Result



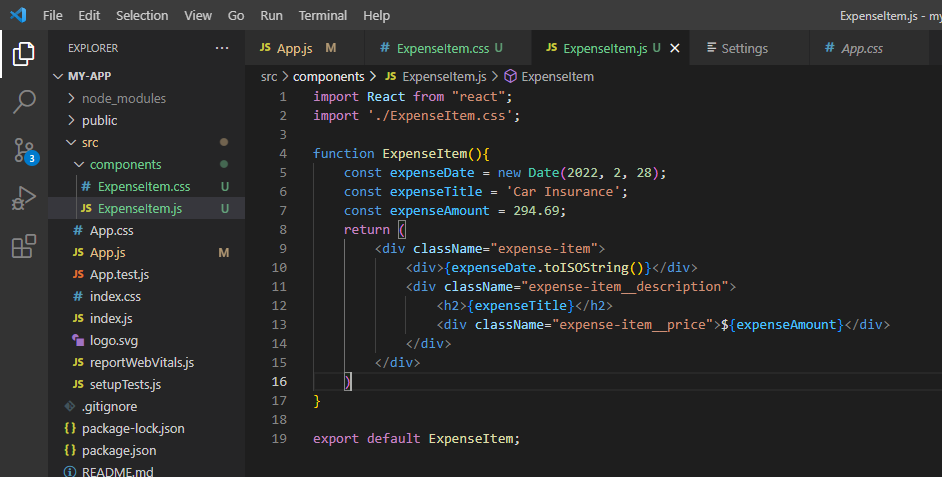
36. Outputting Dynamic Data & Working With Expressions in JSX

Add css by this link to this ExpenseItem.css : <https://github.com/academind/react-complete-guide-code/blob/03-react-basics-working-with-components/code/05-outputting-dynamic-data/src/components/ExpenseItem.css>

All data below hardcode



Should Fix like this



Guild

Step 0: Copy css to ExpenseItem.css

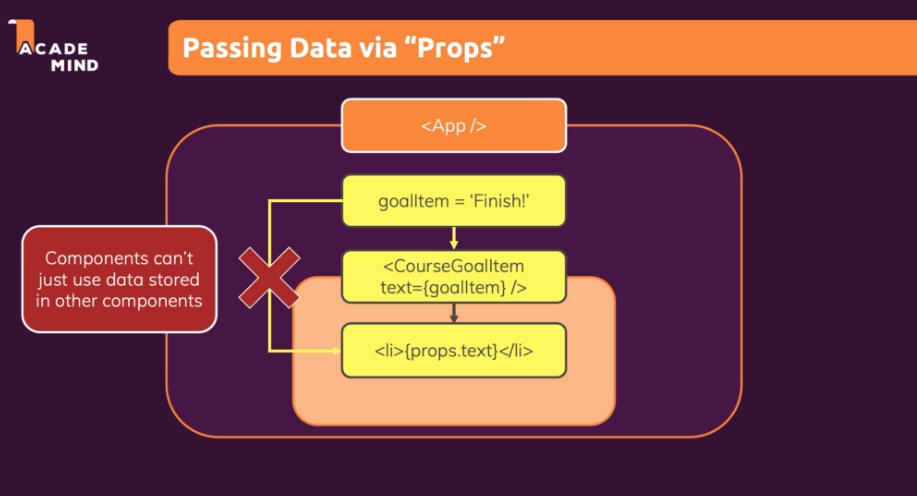
<https://github.com/academind/react-complete-guide-code/blob/03-react-basics-working-with-components/code/05-outputting-dynamic-data/src/components/ExpenseItem.css>

Step 1: Copy like image

Step 2: Done

37. Passing Data via ‘Props’

Problem: Component can’t just use data stored in other components



Props

+Allow transform data from component to another component

In App.js declare list object

Step 1:

import "./App.css";

import ExpenseItem from "./components/ExpenseItem";

function App() {

  const expenses = [

    {

      id: "e1",

      title: "Toilet Paper",

      amount: 94.12,

      date: new Date(2020, 7, 14),

    },

    { id: "e2", title: "New TV", amount: 799.49, date: new Date(2021, 2, 12) },

    {

      id: "e3",

      title: "Car Insurance",

      amount: 294.67,

      date: new Date(2021, 2, 28),

    },

    {

      id: "e4",

      title: "New Desk (Wooden)",

      amount: 450,

      date: new Date(2021, 5, 12),

    },

  ];

  return (

    <div>

      <h2>Let's get started</h2>

      <ExpenseItem

        title={expenses[0].title}

        amount={expenses[0].amount}

        date={expenses[0].date}

      ></ExpenseItem>

      <ExpenseItem

        title={expenses[1].title}

        amount={expenses[1].amount}

        date={expenses[1].date}

      ></ExpenseItem>

      <ExpenseItem

        title={expenses[2].title}

        amount={expenses[2].amount}

        date={expenses[2].date}

      ></ExpenseItem>

      <ExpenseItem

        title={expenses[3].title}

        amount={expenses[3].amount}

        date={expenses[3].date}

      ></ExpenseItem>

    </div>

  );

}

export default App;

Step 2: add ExpenseItem.js like this

import './ExpenseItem.css';

function ExpenseItem(props) {

  return (

    <div className='expense-item'>

      <div>{props.date.toISOString()}</div>

      <div className='expense-item\_\_description'>

        <h2>{props.title}</h2>

        <div className='expense-item\_\_price'>${props.amount}</div>

      </div>

    </div>

  );

}

export default ExpenseItem;

Step 3 define css

.expense-item {

    display: flex;

    justify-content: space-between;

    align-items: center;

    box-shadow: 0 2px 8px rgba(0, 0, 0, 0.25);

    padding: 0.5rem;

    margin: 1rem 0;

    border-radius: 12px;

    background-color: #4b4b4b;

  }

  .expense-item\_\_description {

    display: flex;

    flex-direction: column;

    gap: 1rem;

    align-items: flex-end;

    flex-flow: column-reverse;

    justify-content: flex-start;

    flex: 1;

  }

  .expense-item h2 {

    color: #3a3a3a;

    font-size: 1rem;

    flex: 1;

    margin: 0 1rem;

    color: white;

  }

  .expense-item\_\_price {

    font-size: 1rem;

    font-weight: bold;

    color: white;

    background-color: #40005d;

    border: 1px solid white;

    padding: 0.5rem;

    border-radius: 12px;

  }

  @media (min-width: 580px) {

    .expense-item\_\_description {

      flex-direction: row;

      align-items: center;

      justify-content: flex-start;

      flex: 1;

    }

    .expense-item\_\_description h2 {

      font-size: 1.25rem;

    }

    .expense-item\_\_price {

      font-size: 1.25rem;

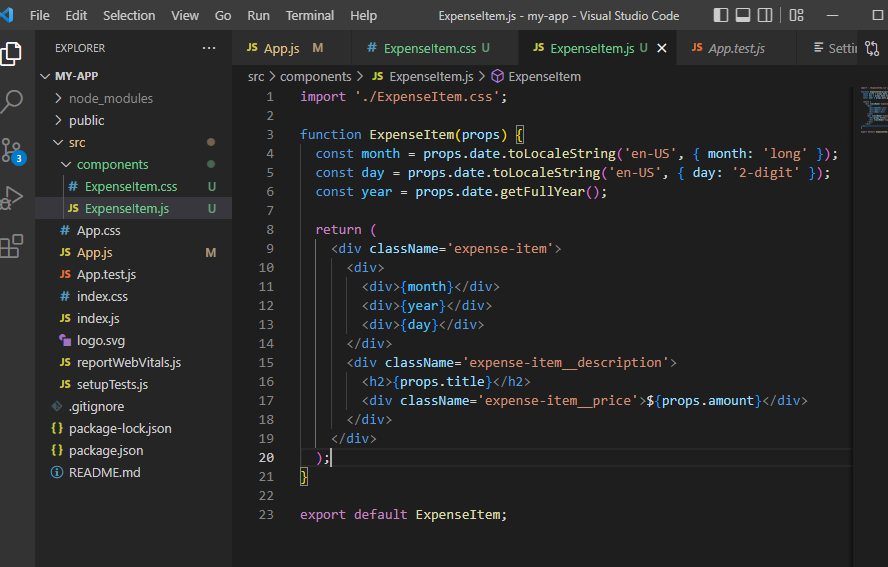
      padding: 0.5rem 1.5rem;

    }

  }

38. Adding ‘Normal’ Javascript logic to components

+Bạn có thể sử dụng các function của javascript như bình thường (ToLocaleString)



Guild:

Copy in your code

import './ExpenseItem.css';

function ExpenseItem(props) {

  const month = props.date.toLocaleString('en-US', { month: 'long' });

  const day = props.date.toLocaleString('en-US', { day: '2-digit' });

  const year = props.date.getFullYear();

  return (

    <div className='expense-item'>

      <div>

        <div>{month}</div>

        <div>{year}</div>

        <div>{day}</div>

      </div>

      <div className='expense-item\_\_description'>

        <h2>{props.title}</h2>

        <div className='expense-item\_\_price'>${props.amount}</div>

      </div>

    </div>

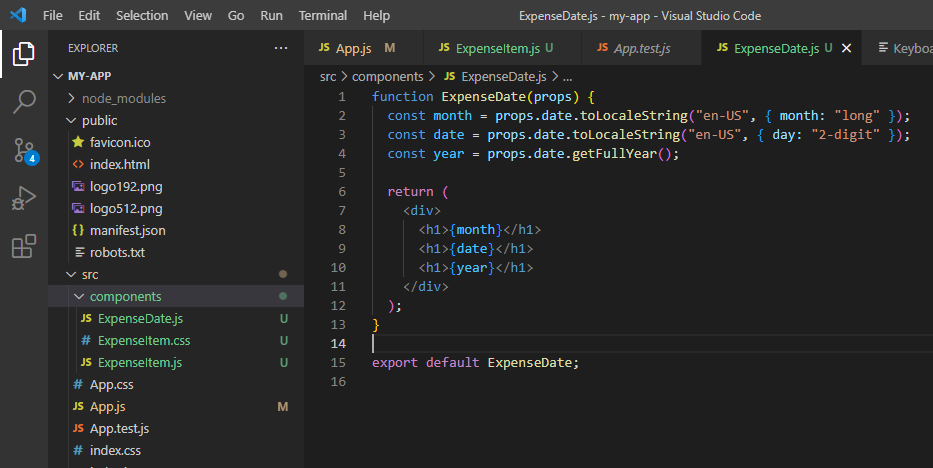
  );

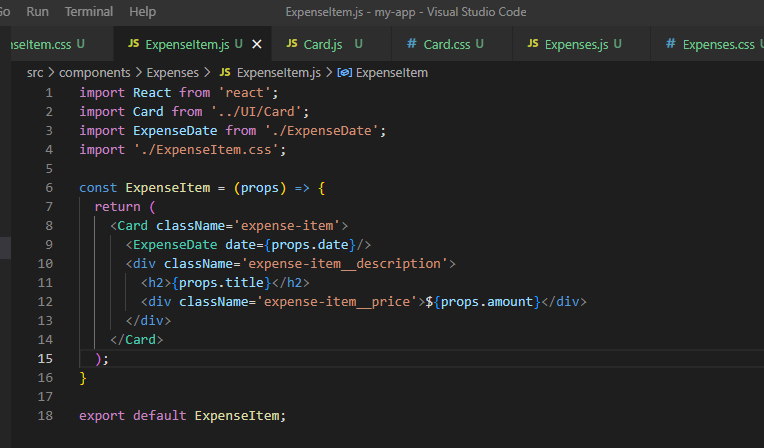
}

export default ExpenseItem;

39. Splitting Components into multiple components

+ Make code more clear





40. The Concept of ‘Composition’ (“Children Props”)

## Child component

* Bây giờ chúng ta có component <Dad /> chứa nhiều component <Son /> như này:

<Dad>

<Son />

<Son />

<Son />

</Dad>

* Ở đây ta có 3 component <Son />, chúng sẽ là props.children của component <Dad />. Để render ra các children ta có thể viết như này:

class Dad extends Component {

render() {

return (

<div>{this.props.children}</div>

);

}

}

* Đây là cách viết thông thường nhất, sử dụng expression container (thuật ngữ của dấu {} trong JSX). Hoặc nếu bạn không muốn trả về 3 component <Son /> đó thì bạn có thể trả về bất cứ gì bạn muốn như này:

class Dad extends Component {

render() {

return (

<h1>Xin chào viblo</h1>

);

}

}

* Trong trường hợp này thì bất kể children của component <Dad /> là gì thì nó vẫn chỉ render ra dòng chữ Xin chào viblo.
* JSX cũng tự động loại bỏ các kí tự khoảng trắng đầu dòng và cuối dòng cũng như các dòng trống. Nó chuyển tất cả trở thành 1 dấu cách. Nên dù bạn viết như sau:

<h1>Xin chào viblo</h1>

<h1>Xin chào

viblo</h1>

<h1>

Xin chào viblo

</h1>

thì cũng đều ra một kết quả như nhau.

### Function as a child

* Ta có thể sử dụng bất cứ Javascript expression nào như là 1 children, kể cả các function.
* Xem ví dụ sau:

\*\* Đây là chỗ gọi component <Dad />:

<Dad>

{() => <h1>Xin chào viblo</h1>}

</Dad>

\*\* Và component <Dad /> sẽ như này:

class Dad extends Component {

render() {

return this.props.children();

}

}

* Do children của component <Dad /> là 1 function nên nếu ta chỉ sử dụng cách gọi children thông thường là this.props.children thì component của ta sẽ trắng trơn, ta sẽ phải sử dụng this.props.children()

### Looping over children

* 2 hàm chúng ta thường gặp là React.Children.map và React.Children.forEach. Chúng hoạt động giống như khi chúng ta dùng map và forEach đối với mảng. Chỉ khác là React.Children cho phép chúng ta thao tác với cả object hay là function tương tự như vậy.
* Xem ví dụ dưới đây để hiểu rõ hơn React.Children hoạt động như nào:

<Dad>

<h1>First message</h1>

<h2>Second message</h2>

</Dad>

* Giờ ở component <Dad /> chúng ta muốn hiển thị ra 2 thẻ h1 và h2 ta có 2 cách như sau:

return (

<div>

{React.Children.map(this.props.children, child => {

return child;

})}

</div>

);

hoặc là:

return (

<div>

{this.props.children.map(child => {

return child;

})}

</div>

);

* 2 cách trên đều cho ra kết quả như nhau tuy nhiên nếu ta gọi component <Dad /> như thế này:

<Dad>

<h1>First message</h1>

<h2>Second message</h2>

{() => <h3>Third message</h3>}

</Dad>

* Nếu gọi như này thì khi ta sử dụng this.props.children sẽ raise lên warning là Functions are not valid as a React child còn sử dụng React.Children.map thì không.

### Counting children

* Bởi vì children của React có thể là bất cứ gì nên việc đếm số lượng children cũng dễ nhầm lẫn.
* Ví dụ như sau:

<Dad>

<h1>First message</h1>

<h2>Second message</h2>

{() => <h3>Third message</h3>}

</Dad>

* Khi ta sử dụng this.props.children.length thì trả về kết quả là 3, nhưng khi sử dụng React.Children.count(this.props.children) thì lại trả về kết quả là 2.
* Thêm ví dụ nữa như này:

<Dad>

"First message"

</Dad>

* Trong trường hợp này thì nếu ta sử dụng this.props.children.length thì sẽ ra kết quả là 15 bởi nó nghĩ đây là 1 chuỗi nhưng React.Children.count(this.props.children) thì trả ra kết quả đúng là 1.
* Vì vậy ta phải lưu ý khi sử dụng 1 trong 2 cách này tùy vào từng trường hợp để cho kết quả chính xác.

### 2.2. Enforcing a single child

* Bây giờ nếu chúng ta muốn component chỉ có duy nhất một children là function ta có thể làm như sau:

Dad.propTypes = {

children: PropTypes.func.isRequired

}

* Nhưng cách trên sẽ raise lên một warning Failed prop type: Invalid prop children of type string supplied to Dad, expected function., thay vào đó ta có thể dùng cách này

class Dad extends Component {

render() {

return React.Children.only(this.props.children)();

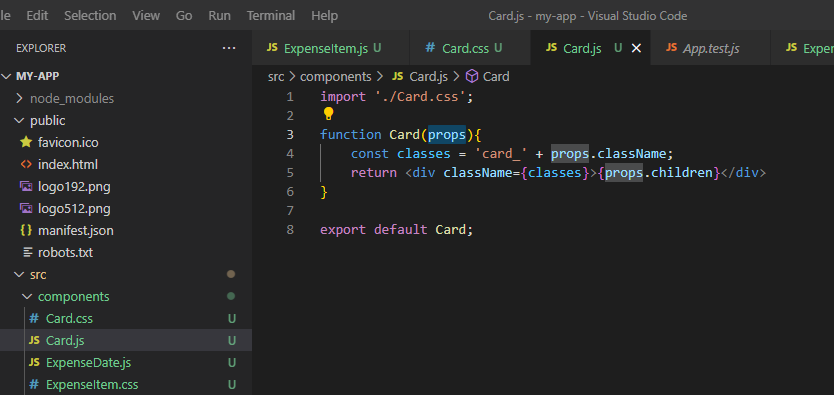
}

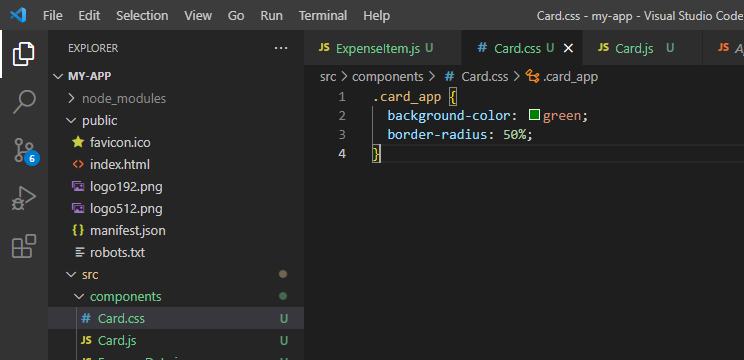
}

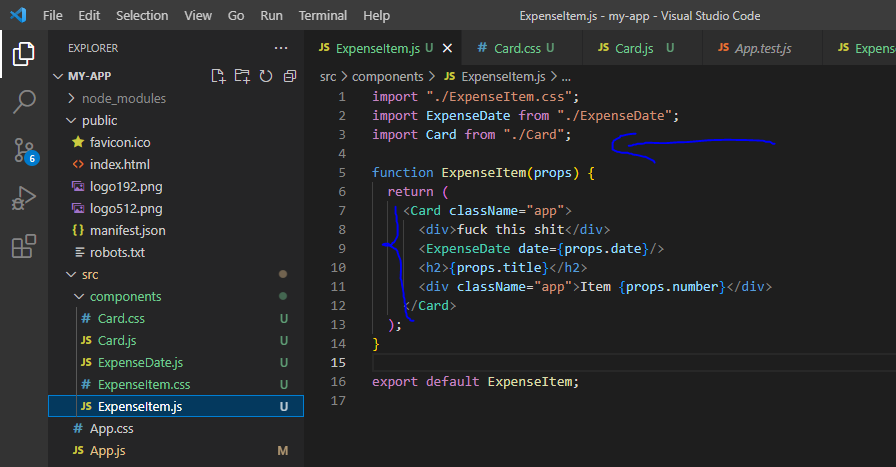
* Nó sẽ trả về duy nhất một child của this.props.children. Nếu this.props.children có nhiều hơn 1 child nó sẽ raise lên error React.Children.only expected to receive a single React element child. Nó sẽ rất tốt trong việc ràng buộc dữ liệu để tránh những lập trình viên ẩu vào làm component trở lên lộn xộn.

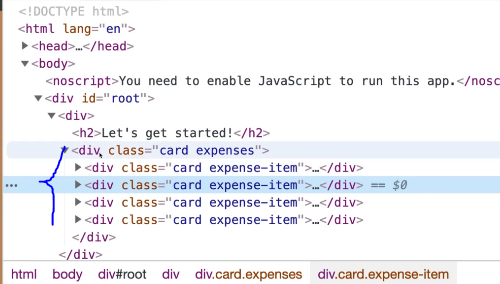
## 3. Editing children

* Chúng ta có thể render tùy ý các component như là children của component <Dad />, tuy nhiên ta vẫn có thể điều chỉnh chúng tùy ý từ component <Dad />.
* Bây giờ tôi sẽ lấy một ví dụ khác:
  + Ta có một form cho người dùng đăng kí thành viên, trong đó có một form input để người dùng nhập giới tính của mình, vậy nó sẽ có 3 component radio button gồm Nam, Nữ và Không xác định. Giả sử ta sẽ render nó như này:
* <FormGroup>
* <RadioButton value="Nam" />
* <RadioButton value="Nữ" />
* <RadioButton value="Không xác định" />
* </FormGroup>
  + Nhưng nó sẽ không chạy đúng bởi cả 3 ô input đều có thể được chọn do nó chưa có chung 1 attribute name, ta có thể làm thủ công như này:
* <FormGroup>
* <RadioButton name="gioitinh" value="Nam" />
* <RadioButton name="gioitinh" value="Nữ" />
* <RadioButton name="gioitinh" value="Không xác định" />
* </FormGroup>
  + Như vậy thì nó hoạt động đúng. Nhưng cách này không phải cái tôi muốn nói ở đây, chúng ta có cách khác để làm việc này mà không phải pass props qua từng component con như vậy. Ta có thể làm việc này từ component <FormGroup />.
  + Ta sẽ sử dụng hàm hỗ trợ của React có tên là cloneElement. Ý tưởng là ta sẽ clone lại các children của this.props.children và tạo 1 element mới với thêm 1 props là name. Ta sẽ làm như sau:
* import React, {Component} from 'react';
* export default class FormGroup extends Component {
* renderChildren() {
* return React.Children.map(this.props.children, child => {
* return React.cloneElement(child, {
* name: this.props.name
* })
* });
* }
* render() {
* return (
* <div>
* {this.renderChildren()}
* </div>
* );
* }
* }
  + Và khi gọi component <FormGroup /> ta sẽ truyền như sau:
* <FormGroup name="gioitinh">
* <RadioButton value="Nam" />
* <RadioButton value="Nữ" />
* <RadioButton value="Không xác định" />
* </FormGroup>
  + Cách này sẽ rất có hiệu quả khi children của FormGroup có rất nhiều phần tử và không được render trực tiếp như này.









Guild

Step 1: add Card.js and Card.css

import './Card.css';

function Card(props) {

  const classes = 'card ' + props.className;

  return <div className={classes}>{props.children}</div>;

}

export default Card;

Card.css

.card {

    border-radius: 12px;

    box-shadow: 0 1px 8px rgba(0, 0, 0, 0.25);

  }

Step 2: Add Expenses.js and Expenses.css

import ExpenseItem from './ExpenseItem';

import Card from './Card';

import './Expenses.css';

function Expenses(props) {

  return (

    <Card className="expenses">

      <ExpenseItem

        title={props.items[0].title}

        amount={props.items[0].amount}

        date={props.items[0].date}

      />

      <ExpenseItem

        title={props.items[1].title}

        amount={props.items[1].amount}

        date={props.items[1].date}

      />

      <ExpenseItem

        title={props.items[2].title}

        amount={props.items[2].amount}

        date={props.items[2].date}

      />

      <ExpenseItem

        title={props.items[3].title}

        amount={props.items[3].amount}

        date={props.items[3].date}

      />

    </Card>

  );

}

export default Expenses;

then

.expenses {

    padding: 1rem;

    background-color: rgb(31, 31, 31);

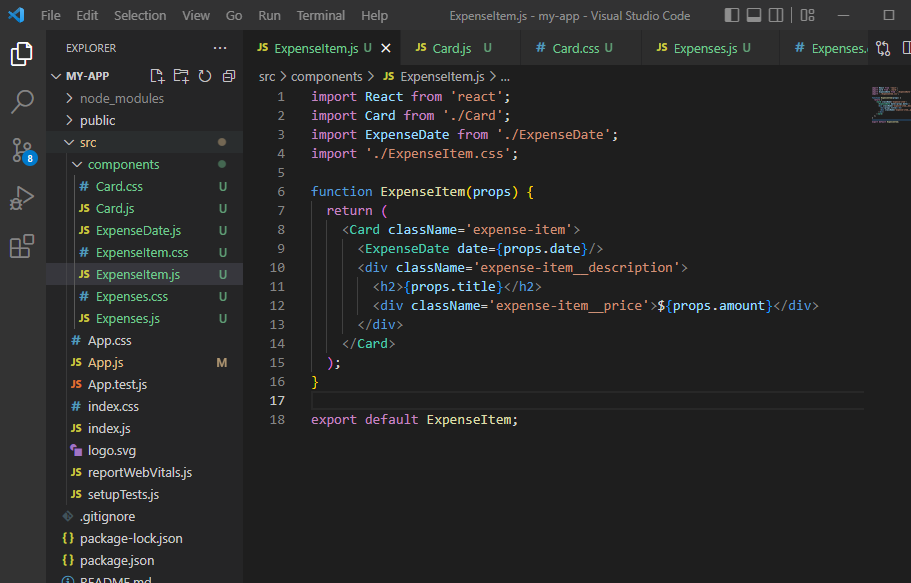
    margin: 2rem auto;

    width: 50rem;

    max-width: 95%;

  }

Step 3 Change ExpenseItem.js like this



Step 4: change App.js like this

import "./App.css";

import Expenses from "./components/Expenses";

function App() {

  const expenses = [

    {

      id: "e1",

      title: "Toilet Paper",

      amount: 94.12,

      date: new Date(2020, 7, 14),

    },

    { id: "e2", title: "New TV", amount: 799.49, date: new Date(2021, 2, 12) },

    {

      id: "e3",

      title: "Car Insurance",

      amount: 294.67,

      date: new Date(2021, 2, 28),

    },

    {

      id: "e4",

      title: "New Desk (Wooden)",

      amount: 450,

      date: new Date(2021, 5, 12),

    },

  ];

  return (

    <div>

      <h2>Let's get started</h2>

      <Expenses items={expenses} />

    </div>

  );

}

export default App;

Summary

+ Know react core & JSX.

+ Working component, data.

+ Declare array

+ Know how organize code, shortcut keyboard on visual

42. A Closer Look At JSX

+Write JSX instead write html

Solution 1



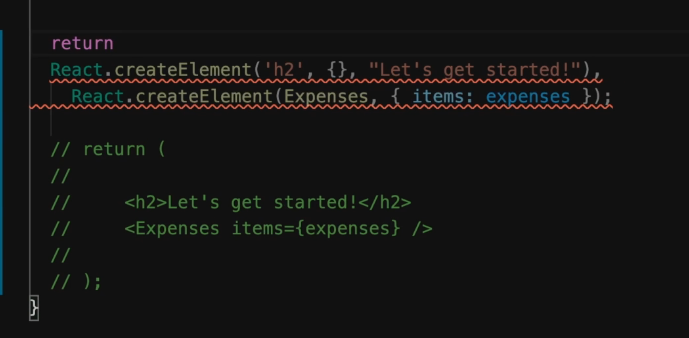
* Hard to maintain

Solution 2

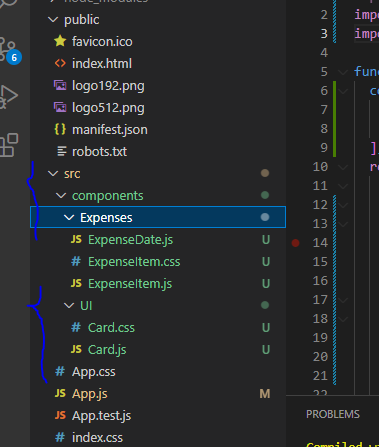
Import React from ‘react’;

Then write html like normal

Why we should write div around all html, because if we write jsx, jsx only return one element



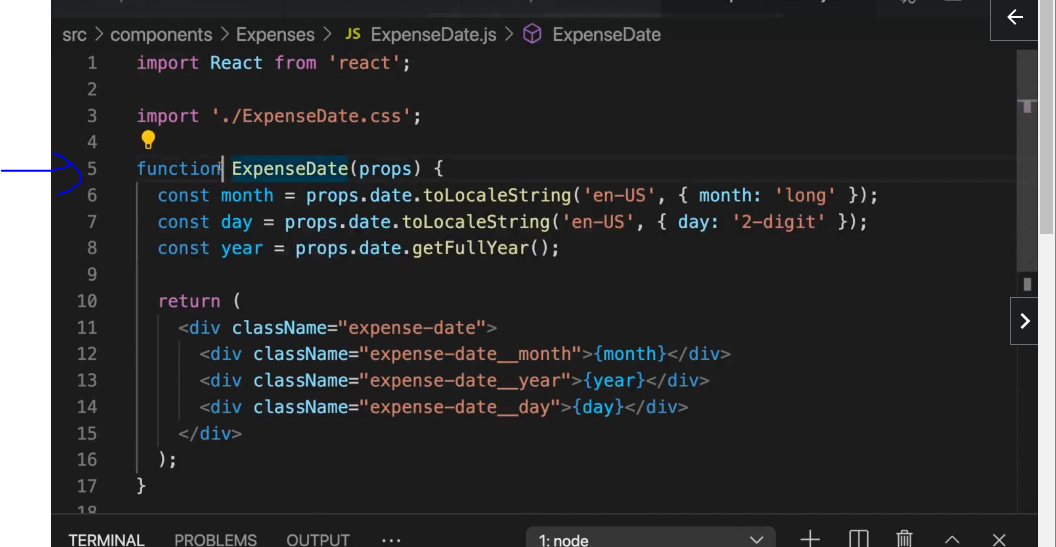
43. Organizing Component Files



44. An Alternative Function Syntax

Change function to typescript like this

From this



To this

