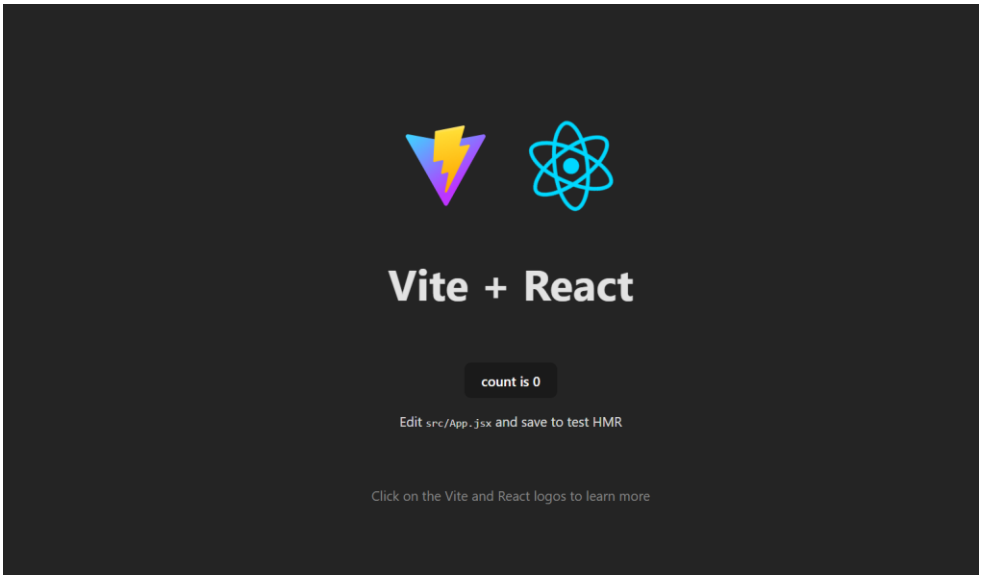
Getting Started with Vite

- Vite is a modern build tool and development server designed to provide a fast and efficient development experience for web projects. It was created by Evan You, the creator of Vue.js, but it is framework-agnostic and can be used with various front-end frameworks such as Vue, React, Svelte, and others. Here are some key features and aspects of Vite:
  - **Instant Server Start:** Vite uses native ES modules in the browser to start the development server almost instantly, without the need for bundling.
  - **Hot Module Replacement (HMR):** Vite offers fast and efficient HMR, enabling a smooth development experience by updating modules in the browser without requiring a full page reload.
  - **Optimized Build:** Vite uses Rollup under the hood for production builds, providing optimized and efficient bundling of your application.
  - **Rich Plugin Ecosystem:** Vite supports Rollup plugins and has a growing ecosystem of its own plugins to extend its functionality.
  - **Modern JavaScript:** Vite is designed to leverage modern JavaScript features and browser capabilities, making it suitable for contemporary development practices.
  - **Support for Various Frameworks:** While it was initially built with Vue.js in mind, Vite has plugins and configurations for other frameworks like React, Preact, Svelte, and more.
  - **Static Asset Handling:** Vite provides an efficient way to handle static assets, such as images and CSS, during development and production builds.
- Overall, Vite is known for significantly improving the development workflow by reducing build times and enhancing the responsiveness of the development server. It has become a popular choice for developers looking for a fast, modern, and flexible build tool for their web projects.

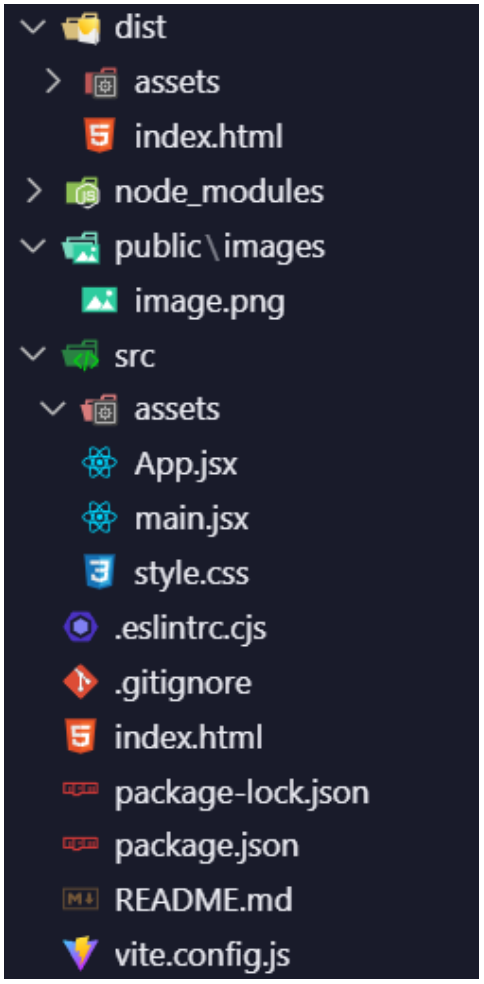
Using Vite

- Use the following commands to set up React + Vite :
  - **npm create vite@latest**
  - **cd my-project**
  - **npm install**
  - **npm run dev**

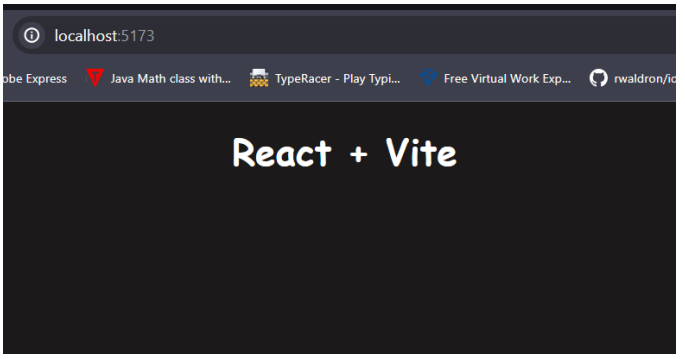
- Now you will see a page running on localhost port 5173



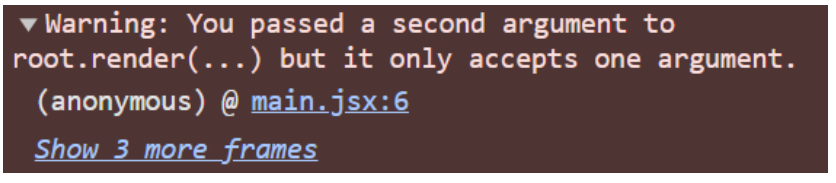
- Now to initiate our project we have to remove some unwanted files which loading the page like this, remove some files and make the dir structure like this:



- Modify App.jsx and main.jsx according to convenience. Note : In vite, it is file naming convention that the first letter of file must be capital, same applies to function naming also.



- **Remember root.render() method always accept only 1 argument (in the from of function).**



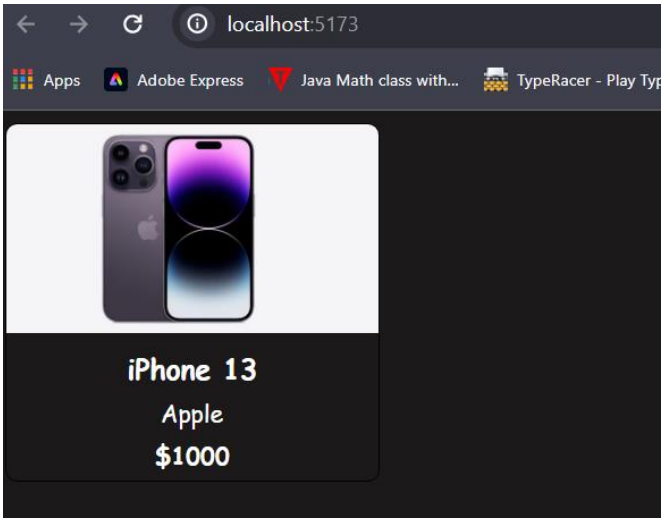
Rendering Multiple Elements

As **root.render()** method only accept one argument, so we can Render Multiple elements enclosing in a div.

Note : In react using Vite we generally don't write any code in main.jsx, we write most of the code in App.jsx

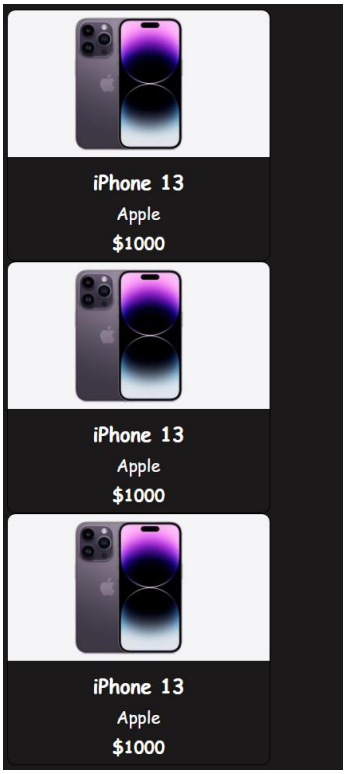
First create a simple element using Javascript Embedded html like this

```
// Let's create a Card element
const Card = (
  <div className="card">
    <img
      src="https://akm-img-a-in.tosshub.com/indiatoday
      alt="iphone"
    />
    <div className="card-content">
      <h3>iPhone 13</h3>
      <p>Apple</p>
      <p>
        <b>$1000</b>
      </p>
    </div>
  </div>
);
```



We can also render multiple elements by enclosing them in array, if we use other than erray then it'll show error :

```
// Render the AppWithCards component into the root DOM node
ReactDOM.createRoot(document.getElementById("root")).render([Card, Card, Card]);
```



Also this will show error like this to give each card a unique identity:

```
▶Warning: Each child in a list   react-dom_client.js?v=1c80f8f7:519
should have a unique "key" prop. See
https://reactjs.org/link/warning-keys for more information.
```

Let's see this React Element in console (Remember we can't directly use it in console of browser as it's a module and we can't directly access them, just use **"console.log(Card)"** in code to see it):

```
main.jsx:24
▼ {$$typeof: Symbol(react.element), type: 'div', key: null, ref: nul
  l, props: {…}, …} i
    $$typeof: Symbol(react.element)
    key: null
  ▼ props:
    ▼ children: Array(2)
      ▶ 0: {$$typeof: Symbol(react.element), type: 'img', key: null,
      ▶ 1: {$$typeof: Symbol(react.element), type: 'div', key: null,
        length: 2
      ▶ [[Prototype]]: Array(0)
      className: "card"
    ▶ [[Prototype]]: Object
    ref: null
    type: "div"
    _owner: null
  ▶ _store: {validated: false}
    _self: undefined
  ▶ _source: {fileName: 'C:/Web Development/React Js/04 Getting Start
  ▶ [[Prototype]]: Object
```

❖ Now we can see that React element ways comes with key property and different props, etc. Also we render multiple elements using array then we can't modify the details and also can't pass the unique key.
For that we uses Card() as function to pass arguments also. Let's first pass key argument:

```jsx
function Card(key) {
  return (
    <div className="card" key={key}>
      <img
        src="https://akm-img-a-in.tosshub.com/indiatoday/
        alt="iphone"
      />
      <div className="card-content">
        <h3>iPhone 13</h3>
        <p>Apple</p>
        <p>
          <b>$1000</b>
        </p>
      </div>
    </div>
  );
}
```

```
{$$typeof: Symbol(react.element), type: 'div', key: '1', ref: nul
l, props: {…}, …}
  $$typeof: Symbol(react.element)
  key: "1"
▶ props: {className: 'card', children: Array(2)}
  ref: null
  type: "div"
  _owner: null
▶ _store: {validated: false}
  _self: undefined
▶ _source: {fileName: 'C:/Web Development/React Js/04 Getting Start
▶ [[Prototype]]: Object
```

❖ Now let's use **dummyJSON** to fetch the product using API, before using it first log it in console and see the properties which we can enter:

```jsx
// Get all Products using dummyJSON
fetch("https://dummyjson.com/products")
  .then((res) => res.json())
  .then(console.log);
```

```
▼ products: Array(30)
  ▼ 0:
      availabilityStatus: "Low Stock"
      brand: "Essence"
      category: "beauty"
      description: "The Essence Mascara Lash Princess is a pop
    ▶ dimensions: {width: 23.17, height: 14.43, depth: 28.01}
      discountPercentage: 7.17
      id: 1
    ▶ images: ['https://cdn.dummyjson.com/products/images/beau
    ▶ meta: {createdAt: '2024-05-23T08:56:21.618Z', updatedAt:
      minimumOrderQuantity: 24
      price: 9.99
      rating: 4.94
      returnPolicy: "30 days return policy"
    ▶ reviews: (3) [{…}, {…}, {…}]
      shippingInformation: "Ships in 1 month"
      sku: "RCH45Q1A"
      stock: 5
    ▶ tags: (2) ['beauty', 'mascara']
      thumbnail: "https://cdn.dummyjson.com/products/images/be
      title: "Essence Mascara Lash Princess"
      warrantyInformation: "1 month warranty"
      weight: 2
```

❖ Here we can use id as key, image for image tag, price, title and thumbnail, etc. We can access these properties by using "dot" operator, first let's change declaration of function card() so that it can take and use all the passed values.

```jsx
function Card(key, title, image, brand, price) {
  return (
    <div className="card" key={key}>
      <img src={image} alt={title} />
      <div className="card-content">
        <h3>{title}</h3>
        <p>{brand}</p>
        <p>
          <b>{price}</b>
        </p>
      </div>
    </div>
  );
}
```

In react **hooks** are used to enable functional components, we use **useState** and **useEffect** to have state and side-effects, respectively.

❖ **useState :** The useState hook is used to add state to a functional component. It returns a state variable and a function to update that state. Here's the basic syntax and an example of how it can be used:

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

function Counter() {
  // Declare a state variable named "count" and a function "setCount" to update it
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

In this example: useState(0) initializes the state variable count to 0. setCount is a function that updates the state variable count.

❖ **useEffect :** The useEffect hook allows us to perform side effects in functional components. Side effects include data fetching, setting up subscriptions, and manually changing the DOM. It is similar to lifecycle methods componentDidMount, componentDidUpdate, and componentWillUnmount in class components. Here's the basic syntax and an example:

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

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

  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;

    // Optionally, you can return a cleanup function
    return () => {
      console.log('Cleanup if necessary');
    };
  }, [count]); // Only re-run the effect if count changes

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

In this example: useEffect runs after every render by default. However, the second argument [count] tells React to only re-run the effect if count changes. The cleanup function (optional) is useful for cleaning up subscriptions or other side effects to prevent memory leaks.

React Components

- In React, components are the building blocks of the user interface. They are JavaScript functions or classes that optionally accept inputs (known as "props") and return React elements that describe what should appear on the screen. There are two main types of components in React: functional components and class components.

- **Functional Components :** Functional components are simple JavaScript functions that accept props as an argument and return React elements. They are often preferred for their simplicity and ease of use, especially with the introduction of hooks like useState and useEffect.

```jsx
import React from 'react';

function Greeting(props) {
  return <h1>Hello, {props.name}</h1>;
}

export default Greeting;
```

With hooks, functional components can manage state and side effects:

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

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

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  }, [count]);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
}

export default Counter;
```

- **Class Components :** Class components are ES6 classes that extend from React.Component and must define a render method that returns a React element. They can have state and lifecycle methods, but with the advent of hooks, they are used less frequently in new React applications.

```jsx
import React, { Component } from 'react';

class Greeting extends Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}

export default Greeting;
```

This is final code which using React Component Card and returning it using App().

```jsx
function Card(props) {
  const { id, title, thumbnail, brand, price } = props;
  return (
    <div className="card" key={id}>
      <img src={thumbnail} alt={title} />
      <div className="card-content">
        <h3>{title}</h3>
        <p>{brand}</p>
        <p>
          <b>{price}</b>
        </p>
      </div>
    </div>
  );
}

function App() {
  const [products, setProducts] = useState([]); // State to hold the list of products

  // useEffect hook to fetch product data from the API when the component mounts
  useEffect(() => {
    fetch("https://dummyjson.com/products")
      .then((res) => res.json())
      .then((data) => setProducts(data.products)); // Update the state with the fetched products
  }, []); // Empty dependency array ensures this effect runs only once when the component mounts

  return (
    <>
      <div className="container">
        {/* Map through the products array and render a Card component for each product */}
        {products.map((product) => (
          <Card
            key={product.id}
            id={product.id}
            title={product.title}
            thumbnail={product.thumbnail}
            brand={product.brand}
            price={product.price}
          />
        ))}
      </div>
    </>
  );
}
```