JSX

JSX stands for JavaScript XML and it allows us to write HTML in React.

All the components of react app must have extension jsx.

VITE

Vite optimizes code compilation and execution, which can result in a better end-user experience due to faster loading times and lower resource usage. It is super fast.

Step 1: npm create vite@5.4.0

Project name:

Select a framework: react

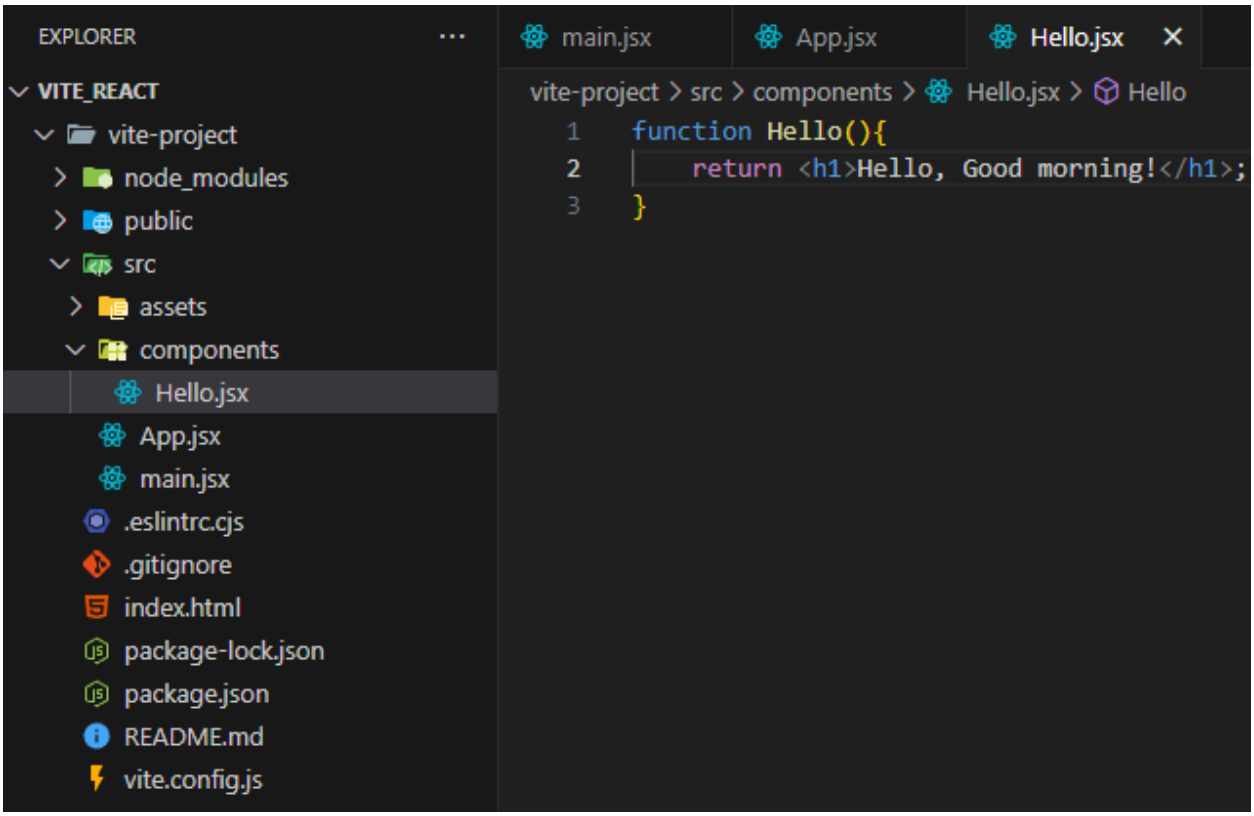
Select a variant: javascript

Step 2: cd directory

Step 3: npm install

Step 4: npm run dev

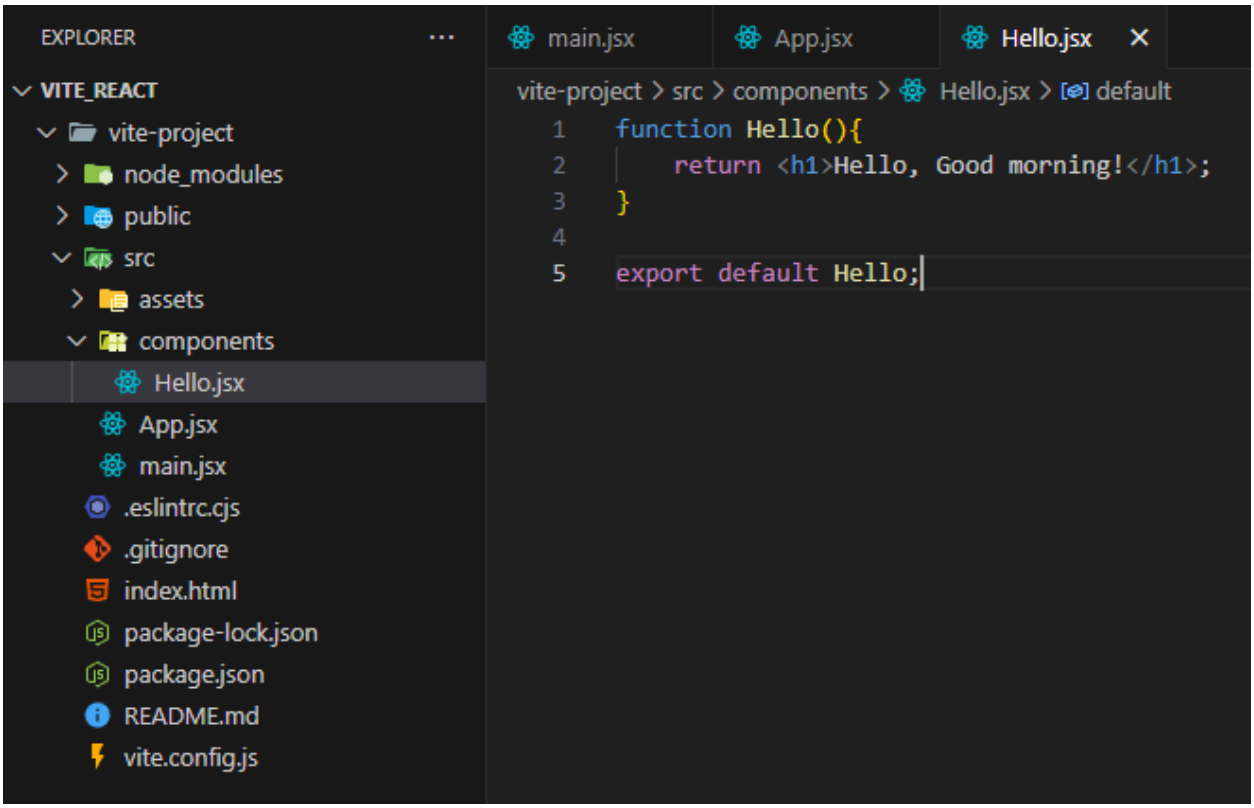
Step 5: ctrl + C to stop the server.

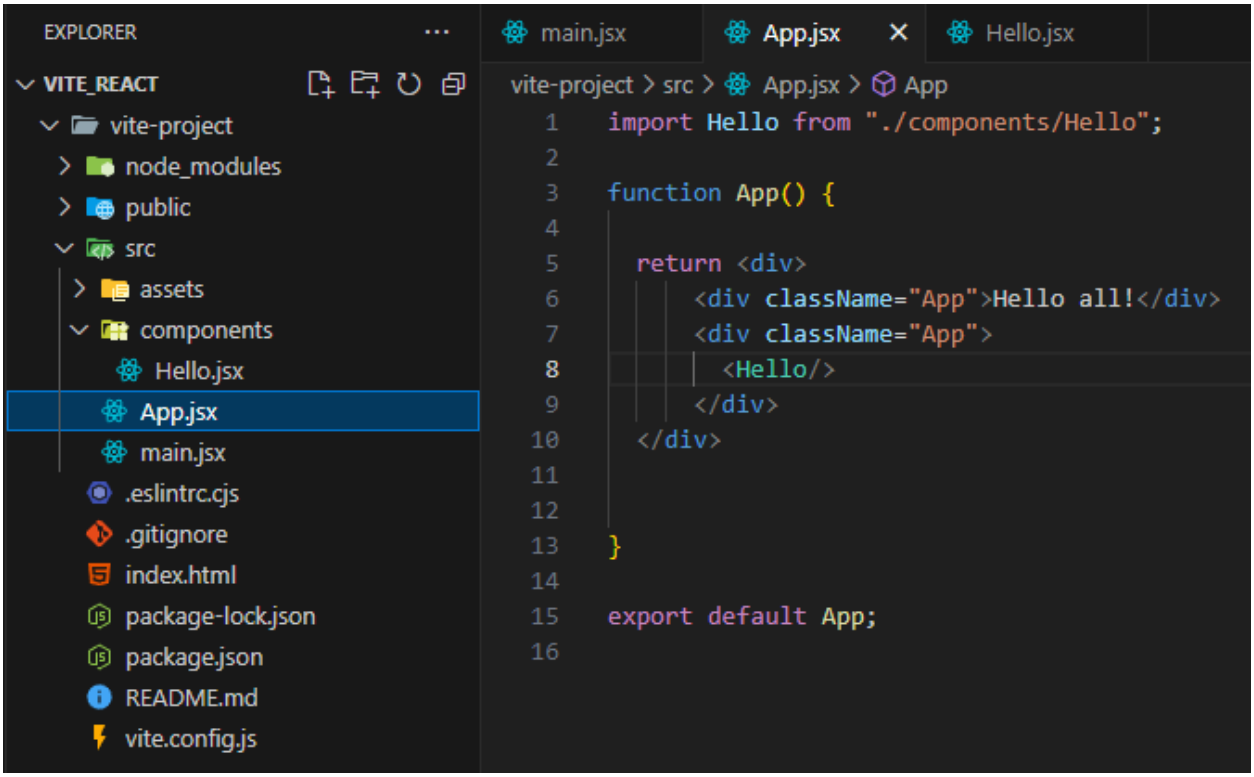


Create a component directory and manage all the components in the same directory.

The component name must be started with a capital letter and also the function inside the file must be of the same name as of the file.

We have to export the function before we import in some file.





Use of jsx.

JSX allows us to write javascript code inside the html. And using jsx, we can only return 1 element. ie. We use div containers.

Reusability of component

It helps in making the application modular.

We can manage each component individually and use any where needed for multiple times.

Abstraction: Hiding away the complexity of codes.

We can use the components as shown below in our application.

```
Hello.jsx    App.jsx
src > App.jsx > App
1   import Hello from "./components/Hello";
2
3   function App() {
4     return (
5       <div>
6         <div className="App">Hello all!</div>
7         <div className="App">
8           <Hello />
9           <Hello />
10          <Hello />
11        </div>
12      </div>
13    );
14  }
15
16  export default App;
17
```

What are props in React?

Props are arguments passed into React components. Props are passed to components via HTML attributes from one component to another. Props stands for properties. It helps use of dynamic components to get different results.

We can pass any data types using props.

Hello.jsx × main.jsx App.jsx

src > components > Hello.jsx > ...

```jsx
const name = "Manish";
function Hello(props) {
  console.log(props);
  return (
    <div>
      <h1>
        {props.message} {props.name}
      </h1>
    </div>
  );
}

export default Hello;
```

Hello.jsx main.jsx App.jsx ×

src > App.jsx > ...

```jsx
import Hello from "./components/Hello";

function App() {
  return (
    <div>
      <div className="App">Hello all!</div>
      <div className="App">
        <Hello />
        <Hello name="Man" message="Hello," />
        <Hello name="Manish" message="Good night!" />
      </div>
    </div>
  );
}

export default App;
```

Strict mode in react

```
Hello.jsx   main.jsx ×   App.jsx
src > main.jsx
1  import React from "react";
2  import ReactDOM from "react-dom/client";
3  import App from "./App.jsx";
4
5  ReactDOM.createRoot(document.getElementById("root")).render(
6    <React.StrictMode>
7      <App />
8    </React.StrictMode>
9  );
10
```

Strict mode is used in dev mode which runs the application 2 times for safety purpose.

It is removed while application goes to production.

Destructuring props

```
Hello.jsx   Hello1.jsx ×   main.jsx   App.jsx
src > components > Hello1.jsx > ...
1  // const name = "Manish";
2  function Hello1(props) {
3    console.log(props);
4    const { name, message } = props;
5    return (
6      <div>
7        <h1>
8          {message} {name}
9        </h1>
10     </div>
11   );
12 }
```

We can directly store the arguments to the variables and use it in the component by destructuring the props.

```
13 function Hello1({ name, message }) {
14   return (
15     <div>
16       <h1>
17         {message} {name}
18       </h1>
19     </div>
20   );
21 }
22
23 export default Hello1;
24
```

We can also directly destructure the props up in the function argument. All the methods give the same result.

Immutability of props

We cannot change the value or reassign value of the variables used to store argument in the components.

Passing Arrays and Object to Components using Props

Hello.jsx | Hello1.jsx | main.jsx | App.jsx | .eslintrc.cjs | Cars.jsx

src > App.jsx > App

```
1  import Hello from "./components/Hello";
2  import Hello1 from "./components/Hello1";
3  import Cars from "./components/Cars";
4
5  function App() {
6    const days = ["Sun", "Mon", "Tue"];
7    const person = {
8      name: "Manish",
9      message: "Hello",
10     days: ["Sun", "Mon", "Tue"],
11   };
12   return (
13     <div>
14       <div className="App">Hello all!</div>
15       <div className="App">
16         <Hello />
17         <Hello name="Man" message="Hello," />
18         {/* <Hello1 name="Manish" message="Good night!" days={days} /> */}
19         <Hello1 person={person} />
20         <Cars />
21       </div>
22     </div>
23   );
24 }
25
26 export default App;
27
```

Hello.jsx | Hello1.jsx | main.jsx | App.jsx | .eslintrc.cjs | Cars.jsx

src > components > Hello1.jsx > ...

```jsx
// }

// function Hello1({ name, message, days }) {
//   return (
//     <div>
//       <h1>
//         {message} {name} {days}
//       </h1>
//     </div>
//   );
// }

// function Hello1(props) {
//   return (
//     <div>
//       <h1>
//         {props.person.message} {props.person.name} {props.person.days}
//       </h1>
//     </div>
//   );
// }
// Destructuring props
function Hello1(person) {
  return (
    <div>
      <h1>
        {person.message} {person.name} {person.days}
      </h1>
    </div>
  );
}
export default Hello1;
```

Using of map function

```jsx
const numbers = [4, 9, 16, 25];
const newArr = numbers.map(Math.sqrt)


const numbers = [65, 44, 12, 4];
const newArr = numbers.map(myFunction)

function myFunction(num) {
  return num * 10;
}
```

`map()` creates a new array from calling a function for every array element.

`map()` does not execute the function for empty elements.

`map()` does not change the original array.

```
Elements  Console  Sources  Network  Performance  Memory  >>  1
top ▼  Filter  Default levels ▼  No Issues  2 hidden

                                                  react-dom_client.js?v=4c5df71a:21549
Download the React DevTools for a better development experience:
https://reactjs.org/link/react-devtools
▶ Object                                                          Hello.jsx:3
▶ Object                                                          Hello.jsx:3
▶ Object                                                          Hello.jsx:3
▶ Object                                                          Hello.jsx:3
▶ Warning: Each child in a list should have a unique   react_jsx-dev-runtime.js?v=4c5df71a:62
"key" prop.

Check the render method of `Cars`. See https://reactjs.org/link/warning-keys for more
information.
    at li
    at Cars
    at div
    at div
    at App
>
```

We can just make ach array item unique by using key as follows:

```
Hello.jsx   Hello1.jsx   main.jsx   App.jsx
src > components > Cars.jsx > ...
1   export default function Cars() {
2     const cars = ["BMW", "TOYOTA", "TATA", "HONDA"];
3     return (
4       <div>
5         {cars.map((car) => (
6           <li key={car}>{car}</li>
7         ))}
8       </div>
9     );
10  }
11
```

Rendering array of Objects

Hello.jsx | Hello1.jsx | main.jsx | App.jsx | .eslintrc.cjs | Cars.jsx

src > components > Cars.jsx > ...

```jsx
export default function Cars() {
  //   const cars = ["BMW", "TOYOTA", "TATA", "HONDA"];
  const cars = [
    { name: "BMW", price: 2000000, emoji: "🚗" },
    { name: "TOYOTA", price: 300000, emoji: "🚗" },
    { name: "TATA", price: 1000000, emoji: "🚗" },
    { name: "HONDA", price: 500000, emoji: "🚗" },
  ];

  return (
    <div>
      {cars.map((car) => (
        // <li key={car}>{car}</li>
        <li key={car.name}>
          {car.name} ${car.price} {car.emoji}
        </li>
      ))}
    </div>
  );
}
```

Rendering components inside a loop

Hello.jsx | Hello1.jsx | main.jsx | App.jsx | Cars.jsx | Car.jsx

src > components > Cars.jsx > Cars > cars.map() callback

```jsx
import Car from "./Car";

export default function Cars() {
  //   const cars = ["BMW", "TOYOTA", "TATA", "HONDA"];
  const cars = [
    { name: "BMW", price: 2000000, emoji: "🚗" },
    { name: "TOYOTA", price: 300000, emoji: "🚗" },
    { name: "TATA", price: 1000000, emoji: "🚗" },
    { name: "HONDA", price: 500000, emoji: "🚗" },
  ];

  return (
    <div>
      <ul>
        {cars.map((car) => (
          // <li key={car}>{car}</li>

          // <li key={car.name}>
          //   {car.name} ${car.price} {car.emoji}
          // </li>

          <Car
            key={car.name}
            name={car.name}
            price={car.price}
            emoji={car.emoji}
          />
        ))}
      </ul>
    </div>
  );
}
```

Hello.jsx | Hello1.jsx | main.jsx | App.jsx

src > components > Car.jsx > Car

```jsx
export default function Car({ name, price, emoji }) {
  return (
    <div>
      {emoji} {name} {price}
    </div>
  );
}
```

Conditionally Rendering JSX & Components

Hello.jsx | Hello1.jsx | main.jsx | App.jsx | Code.jsx

src > components > ConditionalComponent.jsx > ConditionalComponent

```jsx
import Code from "./Code";
import Welcome from "./Welcome";

export default function ConditionalComponent() {
  const display = false;
  if (display) {
    return <Welcome />;
  } else {
    return <Code />;
  }

  //   if (display) {
  //     return (
  //       <div>
  //         <h3>This is conditional statement.</h3>
  //       </div>
  //     );
  //   } else {
  //     return (
  //       <div>
  //         <h3>Code Everyday!</h3>
  //       </div>
  //     );
  //   }
}

```

Hello.jsx | Hello1.jsx | main.jsx

src > components > Code.jsx > Code

```jsx
export default function Code() {
  return (
    <div>
      <h1>CodeEveryday!</h1>
    </div>
  );
}

```

```
Hello.jsx   Hello1.jsx   main.jsx
src > components > Welcome.jsx > Welcome
1  export default function Welcome() {
2    return (
3      <div>
4        <h1>You are welcome.</h1>
5      </div>
6    );
7  }
8
```

It’s not good to have wo return statements in a single component.

Conditional Rendering using Element Variables

```
Hello.jsx   Hello1.jsx   main.jsx   App.jsx   Code.jsx
src > components > ConditionalComponent.jsx > ConditionalComponent
1  import Code from "./Code";
2  import Welcome from "./Welcome";
3
4  export default function ConditionalComponent() {
5    //   let messageOne = <h1>This is message 1</h1>;
6    //   let messageTwo = <h1>This is message 2</h1>;
7    //   const display = true;
8    //   if (display) {
9    //     return messageOne;
10   //   } else {
11   //     return messageTwo;
12   //   }
13
14   //  To handle the multiple return
15   let message;
16   const display = false;
17   if (display) {
18     message = <h1>This is message 1</h1>;
19   } else {
20     message = <h1>This is message 2</h1>;
21   }
22   return message;
23
```

Always use element variables to conditionally render JS elements.

Ternary Operators

```
// Using ternary operator
  const display = false;
  return display ? <h1>Message 1</h1> : <h1>Message 2</h1>;
```

Or we can also use components.

```
  // Using ternary operator
  const display = true;
  return display ? <Welcome /> : <Code />;
```