

TypeScript, Components, Events, use State, CSS

RUJUAN XING



Configuring TypeScript

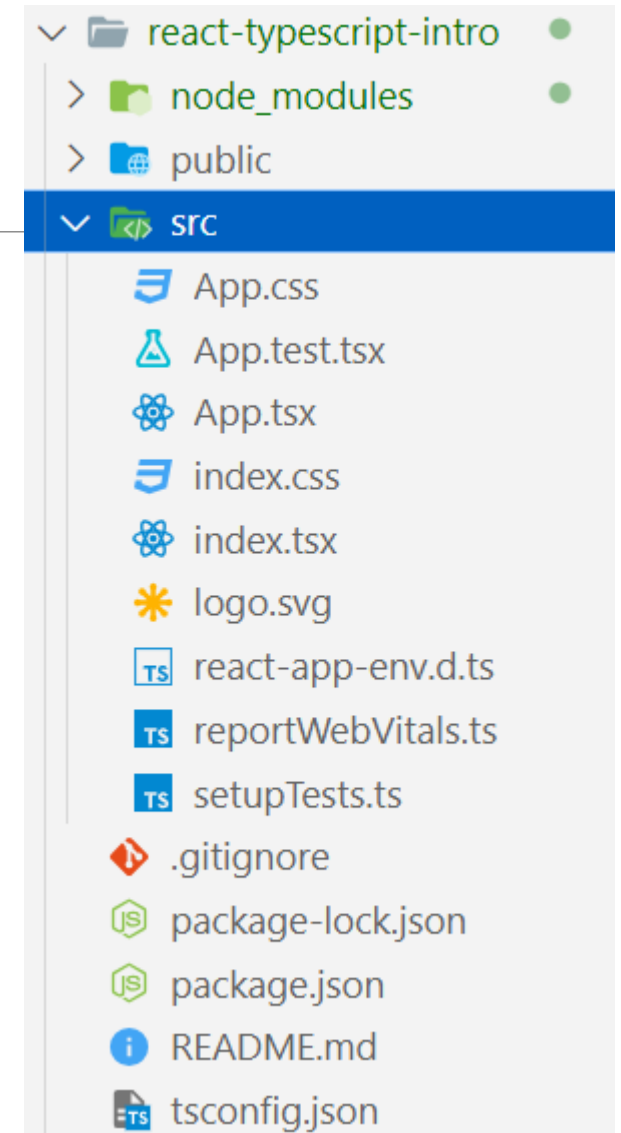
Installation

1. To start a new Create React App project with TypeScript, you can run:

```
npx create-react-app my-app --template typescript
```

2. To add TypeScript to an existing Create React App project, first install it:

```
npm install --save typescript @types/node @types/react @types/react-dom @types/jest
```



TypeScript Configuration File

`tsconfig.json`

- at the root of a project
- Specifies settings that will be used by TypeScript when compiling our code.

Create React App types declaration file

`react-app-env.d.ts`:

- inside `src` folder
- references TypeScript types declarations that are specific to projects started with Create React App.
- Add support for importing resources files such as bmp, gif, etc
- The tripe-slash directive:
 - Referring to a file with some types definitions
 - `node_modules/react-scripts/lib/react-app.d.ts`

.tsx files

The introduction of .tsx also resulted in three additional things:

- JavaScript bundlers like webpack, esbuild, and others can run different plugins for .tsx files
- Test runners like Jest can run a different test environment only for .tsx files
- New language support by code editors

Difference between .ts and .tsx in TypeScript:

The .ts file extension is used when you are creating functions, classes, reducers, etc., that do not require the use of JSX syntax and elements, whereas the .tsx file extension is used when you create a React component and use JSX elements and syntax.

Components Intro

Module vs Component

Module:

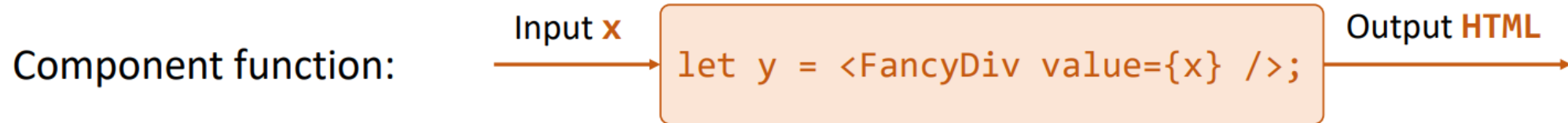
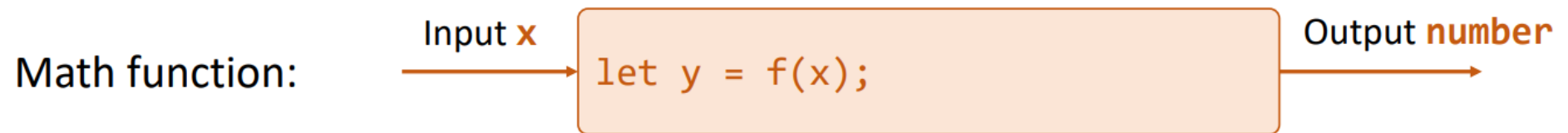
- JavaScript modules allow you to break up your code into separate files.
- Example: 1 giant js file -> separate into multiple JS files(a.js, b.js, c.js, etc)

Components:

- let you split the UI into independent, reusable pieces, and think about each piece in isolation.
- Example: not only js file is splited, all resources such as html, css, js, videos, images, etc

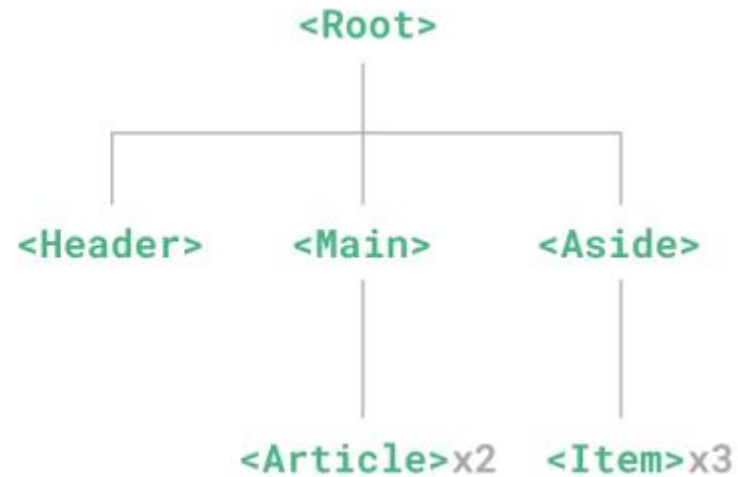
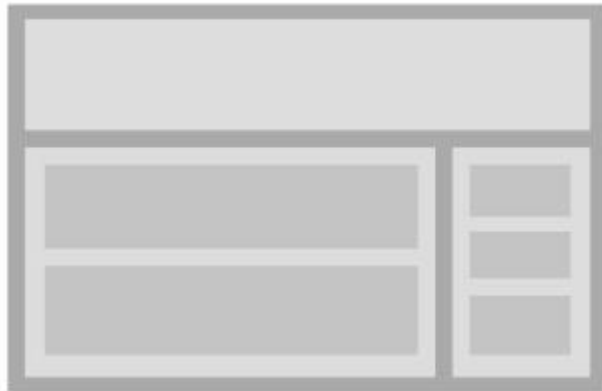
Components

Components are functions for user interfaces



Components

A component is a part of the user interface that can have its own logic and appearance. Components can be nested within each other and reused multiple times.



Component-based development allows developers to build a complete and large-scale application like assembling building blocks.

Types of Components in React

Two types of Components:

- Function Component
 - Function components are defined as functions and are used for simple, stateless presentation components.
 - With the introduction of React Hooks, functional components have become even more powerful and can manage state and side effects as well.
 - The name of function component **must start with capital case**.
- Class Component
 - A class component must include the extends `React.Component` statement. This statement creates an inheritance to `React.Component`, and gives your component access to `React.Component`'s functions.
 - The component also requires a `render()` method, this method returns JSX.
 - **In older React code bases, you may find Class components primarily used. It is now suggested to use Function components along with Hooks, which were added in React 16.8.**

Function Component Demo

The simplest way to define a component is to write a JavaScript function:

- Function name must start with capital case
- `this` keyword inside function component is undefined.

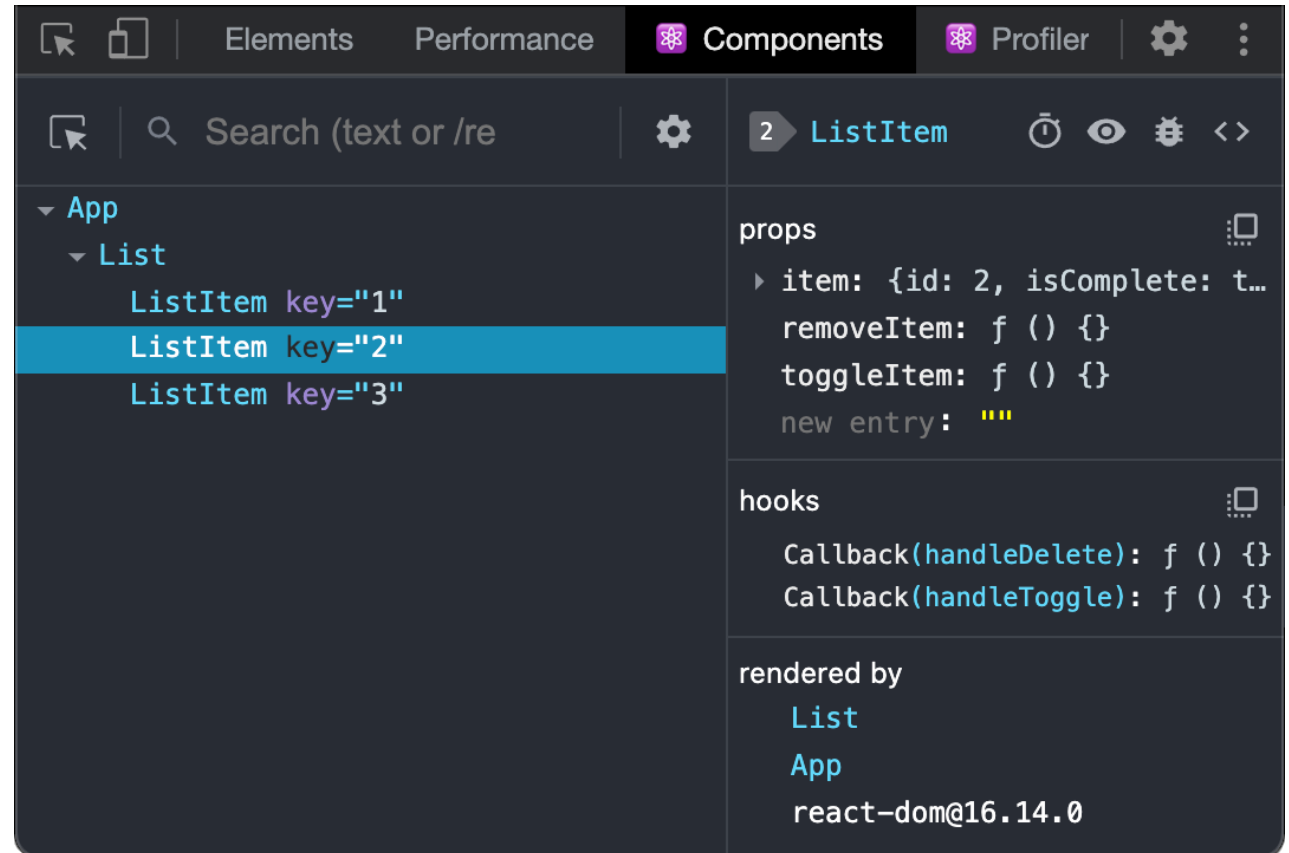
```
function App() {  
  return (  
    <div className="App">  
      <header className="App-header">  
        Learn React  
      </header>  
    </div>  
  );  
}
```

React Developer Tools

Use React Developer Tools to inspect React components, edit props and state, and identify performance problems.

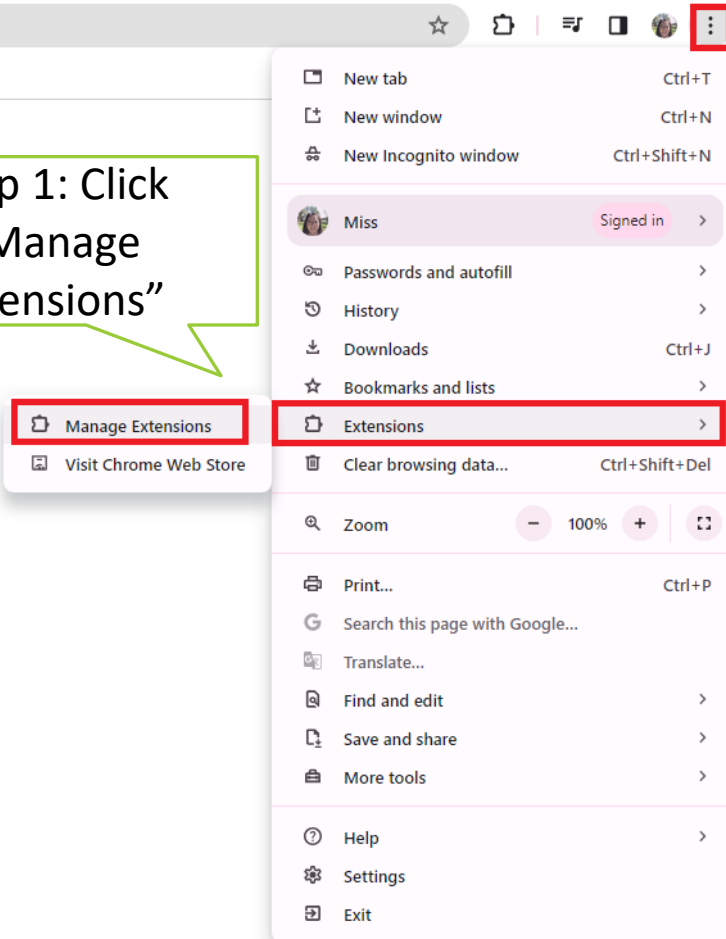
[Install for Chrome](#) [Install for Firefox](#)
[Install for Edge](#)

if you visit a website **built with React**, you will see the *Components* and *Profiler* panels.

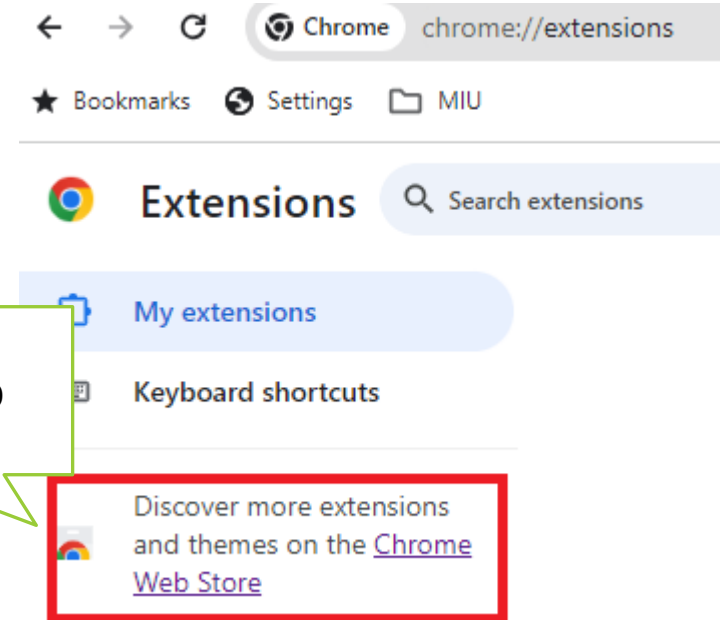


React Developer Tools Installation

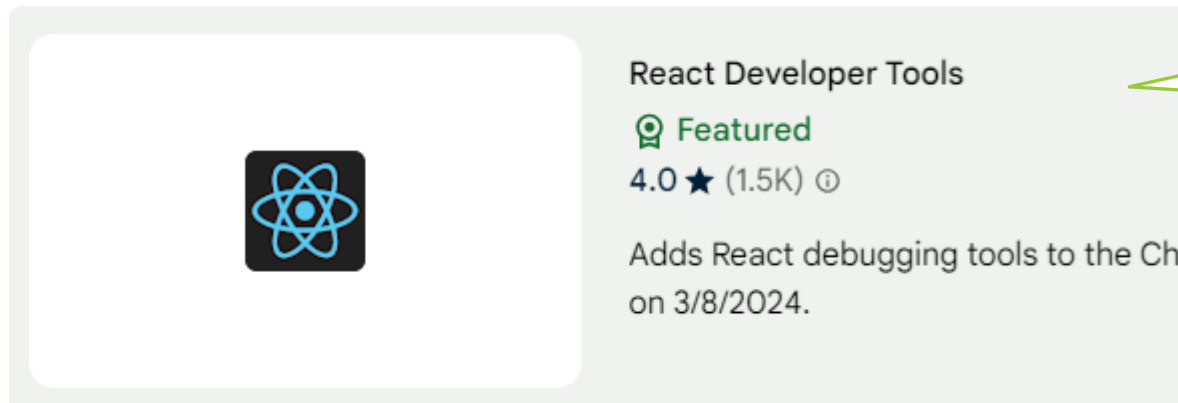
Step 1: Click
“Manage
Extensions”



Step 2: Click
“Chrome Web
Store”



React Developer Tools Installation (Cont.)



Step 3: Search
"React" and show
results

Step 4: Click "Add
to Chrome"



Responding to Events

Responding to Event

React lets you add event handlers to your JSX. Event handlers are your own functions that will be triggered in response to interactions like clicking, hovering, focusing form inputs, and so on.

1. Register through `onEvent` property, be aware of the camel case used here.
 - a. React customizes `onEvent`, not using the original DOM event.
 - b. Events in React are handled through event delegation (delegated to the outermost element of the component)
2. Can get the DOM object via `event.target`. Don't overuse Refs.

Example

```
function App() {  
  
  const clickHandler = () => {  
    console.log('Button is clicked!!!');  
  }  
  
  return (  
    <button onClick={clickHandler}>Click Me</button>  
  );  
}
```

Note:

- `clickHandler` is a function, doesn't matter using function declaration or arrow function
- Register `clickHandler` as a function reference, not call the function

Example: get event object

Set parameter 'e' in the event callback function

```
function App() {  
  const clickHandler = (e: MouseEvent<HTMLButtonElement>):void => {  
    console.log('Button is clicked!!!', e);  
  }  
  
  return (  
    <button onClick={clickHandler}>Click Me</button>  
  );  
}
```

Example: Pass custom parameters

Transform event binding into arrow function syntax, and pass arguments when executing the `clickHandler` to handle actual business logic.

```
function App() {  
  
  const clickHandler = (name: string):void => {  
    console.log('Button is clicked!!!', name);  
  }  
  
  return (  
    <button onClick={() => clickHandler('MIU')}>Click Me</button>  
  );  
}
```

Example: Pass Event object and Custom Parameters together

Be Careful: The order of the parameters

```
function App() {  
  const clickHandler = (name: string, e: MouseEvent<HTMLButtonElement>):void => {  
    console.log('Button is clicked!!!', name);  
  }  
  
  return (  
    <button onClick={e => clickHandler('MIU', e)}>Click Me</button>  
  );  
}
```

Hooks: useState

REACT

Hooks

Hooks are functions that allow functional components to "hook into" React state and lifecycle features.

There are many hooks:

1. State Hooks
2. Ref Hooks
3. Effect Hooks
4. Context Hooks
5. ...

Component Cores: State

The state is a built-in React object that is used to contain data or information about the component.

A component's **state** can change over time; **whenever it changes, the component re-renders. (Data Driven View)**

- A state can be modified based on user action or network changes
- Every time the state of an object changes, React re-renders the component to the browser
- The state object is initialized in the constructor
- The state object can store multiple properties using Object.

```
▼ MyComponent ⓘ  
  ▶ context: {}  
  ▶ props: {}  
  ▶ refs: {}  
  ▶ state: null  
  ▶ updater: {isMounted: f, enqueue  
  ▶ _reactInternalInstance: {_proc  
  ▶ _reactInternals: FiberNode {ta  
    isMounted: (...)  
    replaceState: (...)  
  ▶ [[Prototype]]: Component
```


State Hook - useState

- `useState` is a React Hook that lets you add a state variable to your component.

```
const [state, setState] = useState(initialState);
```

- `initialState`: The value you want the state to be initially. It can be a value of any type, but there is a special behavior for functions. This argument is ignored after the initial render.
 - If you pass a function as `initialState`, it will be treated as an initializer function. It should be pure, should take no arguments, and should return a value of any type. React will call your initializer function when initializing the component, and store its return value as the initial state.
- `useState` returns an array with exactly two values:
 1. The current state. During the first render, it will match the `initialState` you have passed.
 2. The `set` function that lets you update the state to a different value and trigger a re-render.
- The convention is to name state variables like `[something, setSomething]` using array destructuring.

useState Demo

In React, state is considered to be read-only. We should always replace it instead of modifying it directly. Modifying state directly does not trigger view updates.


```
import { useState } from 'react';

function App() {

  let [count, setCount] = useState<number>(0);

  const clickHandler = () => {
    count++;
    console.log(count);
  }

  return (
    <div>
      {count}
      <button onClick={clickHandler}>Click Me</button>
    </div>
  );
}
```




```
import { useState } from 'react';

function App() {

  let [count, setCount] = useState<number>(0);

  const clickHandler = () => {
    setCount(++count);
    console.log(count);
  }

  return (
    <div>
      {count}
      <button onClick={clickHandler}>Click Me</button>
    </div>
  );
}
```




useState Demo: object

For object type state variables, pass a completely new object to the set method for modification

```
function App() {  
  
  interface UserFormState {  
    email: string;  
    password: string;  
    username: string;  
  }  
  
  const [userForm, setUserForm] = useState<UserFormState>({  
    email: '',  
    password: '',  
    username: 'MIU'  
  });  
  
  const handleChangeName = () {  
    userForm.username = 'MSD';  
  }  
  
  return (  
    <div>  
      {userForm.username}  
      <button onClick={handleChangeName}>Change Name</button>  
    </div>  
  );  
}
```

```
const handleChangeName = () => {  
  setUserForm({  
    ...userForm,  
    username: 'MSD'  
  })  
}
```



useState & TypeScript

1. Typically, React automatically infers the type based on the default value passed to useState, so explicit type annotations are not required.
 - `const [value, toggle] = useState(false);`
 - value: boolean
 - toggle: boolean
2. useState is a generic function that can take specific custom types as arguments.

```
type User = {  
  name: string,  
  age: number  
}
```

```
const [user, setUser] = useState<User>();
```

- Restricting the initial value parameter of the useState function to be of type: `User | (() => User)`
- Restricting the parameter of the setUser function to be of type: `User | (() => User) | undefined`

useState & TypeScript (Cont.)

3. When we are unsure about the initial value of a state, setting `useState`'s initial value to `null` is a common practice, and explicit annotation can be done by using a specific type union with `null`.

```
type User = {  
  name: string,  
  age: number  
}
```

```
const [user, setUser] = useState<User | null>(null);
```

- Restricting the initial value parameter of the `useState` function to be either `User` or `null`.
- Restricting the parameter type of the `setUser` function to be `User` or `null`.

Main Point

React Hooks, introduced in React 16.8, are a set of functions that enable developers to add state and side-effect handling to functional components. This innovation allows for the management of component logic without the need for class components. By using hooks like `useState` and `useEffect`, developers can achieve cleaner, more readable code and better component reusability. Hooks have become a cornerstone of modern React development, simplifying state management and making it more approachable for both new and experienced developers.

The existence of a unified field of all the laws of nature, from which everything in the universe originates. It is described as a field of pure consciousness and the source of all knowledge and creativity.