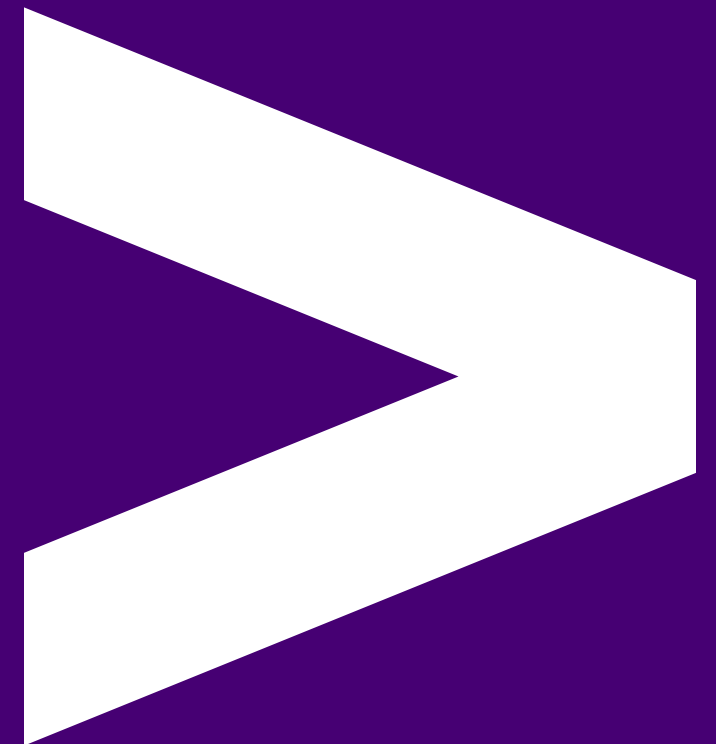


React.js 1

Components



Overview

- Introducing React
- JSX / TSX
- The React component
- Testing a component

Objectives

- Understand the problem React is aiming to solve
- Understand what a React component is and how to define one
- Become familiar with the parent->child component pattern
- Understand how to pass data from parent to child via props
- Understand how to test a component renders correctly

The React sessions

We have 4 React sessions:

- Components: Introduction to React; Components & JSX/TSX; Testing Components (this one)
- State: Virtual DOM; Data Flow; State Management with the `useState` React Hook
- Side Effects: Component Lifecycle; Fetching Data with the `useEffect` React Hook; Testing User Events
- Routing: React Router; Building & Deployment

What is React?

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex UIs from small and isolated pieces of code called “components”.

Created by Facebook. First released: 2013.

reactjs.org

Similar Tools

- Angular (Google)
- Vue.js
- Svelte
- Backbone.js
- Knockout.js

Use Cases

- Traditional websites
- Frontend web Single Page Apps (SPA)
- Isomorphic / Progressive Web Apps (PWAs)
- Native mobile apps (React Native)
- Server Side Rendered web apps (SSR)

Why do we need these tools?

In the early days, the web was primarily text based.

Today, we have a variety of rich user experiences all running in the browser across multiple platforms and devices.

As developers, this means there's a huge amount of complexity we have to organise, manage, and control.

Tools such as React have been created to make this task easier.

Separation of Concerns

In the "old" days it was best practice to keep HTML (markup) and JavaScript (logic) separate (separation of technologies).

React's approach is to achieve separation of concerns instead by using loosely coupled units called “components” that contain both markup and logic.

Components can then be composed to make complex UIs.

React Concepts

- Composition of components
- Unidirectional Data Flow
- The Virtual DOM

Bootstrap a React App

There is a tool called [vite](#) which can be used to bootstrap a react app and get developing quickly. Do this in your [react-01-components/examples](#) folder:

```
mkdir demo-projects
cd demo-projects
npm create vite@latest
# type in your react app-name
# <select React>
# <select Typescript + SWC>
cd <app-name>
npm install
npm run dev
```

React Components

Components are the core building blocks of a React application. They help us break down our UI into independent, reusable pieces that can be worked on separately. These pieces can then be composed together to create our application.

Example: bbc.co.uk

A React Component Example in Practice

Example: [skyscanner.net](https://www.skyscanner.net)
[The calendar component](#)

Create a React Component

A React component can be written as a typescript function with a capitalized name, this is called a function component:

```
interface GreetingProps = {  
  name: string  
}  
  
const Greeting = (props: GreetingProps) => {  
  return <h1>Hello {props.name}</h1>  
}
```

You can use a class to define a component (less common nowadays). The equivalent component written as a class component:

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

Breakdown of a React Component

```
interface GreetingProps = {  
  name: string  
}  
  
const Greeting = (props: GreetingProps) => {  
  return <h1>Hello {props.name}</h1>  
}
```

- Name must be capitalized (to differentiate it from an HTML tag)
- Accepts a single `props` (properties) parameter (which is an object)
- Returns a React element (written in JSX)

JSX

Here is an example of JSX:

```
const element = <h1>Hello world!</h1>
```

`<h1>Hello world!</h1>` is neither a string nor HTML.

JSX is a syntax extension to JavaScript that can be used generate React elements that describe what the UI should look like.

When we use JSX with TypeScript we call it TSX, but it works the same way.

We'll refer to TSX for the rest of this session.

TSX is compiled to "vanilla" JavaScript (just like normal TypeScript).

Expressions in TSX

TSX comes with the full power of TypeScript, so we can embed any valid TypeScript expression in TSX by putting it in curly braces in the TSX:

```
const name = "Alice"  
const element = <h1>Hello {name}!</h1>
```

TSX is a TypeScript Expression

TSX comes with the full power of TypeScript, so we can use TSX as an expression too: we can assign it to variables and return it from functions.

```
function getGreeting(user?: string): JSX.Element {  
  if (user) {  
    return <h1>Hello {user}</h1>  
  }  
  return <h1>Hello stranger</h1>  
}
```

TSX conditional

TSX comes with the full power of TypeScript, so we can write arbitrarily complex logic to conditionally render our page content.

```
return loading ? (  
  <div>Loading...</div>  
) : (  
  <div>  
    <p>Some page content</p>  
    <div>Some data that was loaded</div>  
  </div>  
)
```

Emoji Check:

How do you feel about TSX and the concept that it is an extension of TypeScript?

1. 🥲 Haven't a clue, please help!
2. 😞 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 😊 Yes, with team collaboration could try it
5. 😄 Yes, enough to start working on it collaboratively

TSX Gotcha

Sibling TSX elements/components must be wrapped in an enclosing tag.
The following is invalid and will error:

```
// cannot do this
return (
  <h1>Header</h1>
  <p>Some paragraph text</p>
)
```

TSX Wrapping Possible Solution

We can wrap the sibling elements in any tag or component (for example, a `div`), but this will add another node to the DOM which we may not want.

```
return (  
  <div>  
    <h1>Header</h1>  
    <p>Some paragraph text</p>  
  </div>  
)
```

TSX Fragment

Instead we can use a TSX fragment `<>` to group a list of children without adding extra nodes to the DOM.

```
return (  
  <>  
    <h1>Header</h1>  
    <p>Some paragraph text</p>  
  </>  
)
```

Lists and Keys

If you have a list of sibling elements that are the same you will need to provide a unique "key" prop. Keys help React identify which items have changed, have been added or removed.

Use a string that uniquely identifies the item among its siblings.

```
const shoppingListItems = shoppingList.map((item) => (  
  <li key={item.id}>{item.text}</li>  
))  
return <ul>{shoppingListItems}</ul>
```


TSX Summary

- TSX is a syntax extension to JavaScript and therefore comes with the full power of TypeScript (conditionals, iteration, functions)
- Components must return a single root tag, component or fragment
- Add a **key** prop when rendering multiples of the same component

Emoji Check:

How do feel about the usage of TSX in React?

1. 🥲 Haven't a clue, please help!
2. 😞 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 😊 Yes, with team collaboration could try it
5. 😄 Yes, enough to start working on it collaboratively

Breakdown of a React Component - recap

```
const Greeting = (props: GreetingProps) => {  
  return <h1>Hello {props.name}</h1>  
}
```

- Name must be capitalized (to differentiate it from an HTML tag)
- Accepts a single `props` (properties) parameter (which is an object)
- Returns a React element (written in TSX)

Components Demo

Let's have a look at some components in action.

```
cd to ./examples/component-demo and run npm install, then npm run dev
```

The Child Component

```
interface GreetingProps = {  
    name: string  
    isBirthday: boolean  
}  
  
const Greeting = (props: GreetingProps) => {  
    return (  
        <div>  
            <h1>Hello {props.name}</h1>  
            {props.isBirthday && <p>Happy Birthday to you!</p>}  
        </div>  
    )  
}
```

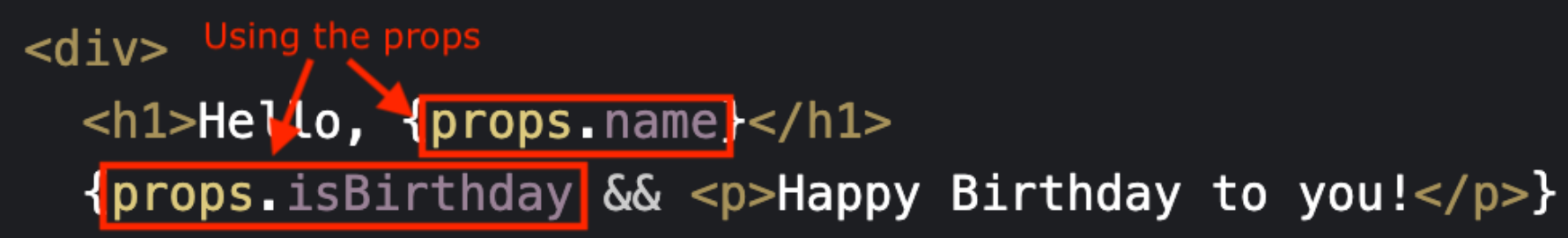
Property Types

We need to give our properties a type, so that the rest of our program knows what to do with them.

We can use an interface to do this:

```
interface GreetingProps {  
  name: string,  
  isBirthday: string  
}  
  
const Greeting = (props: GreetingProps) => {  
  // Component contents...  
}
```

This is useful as TypeScript will tell us if we are using the wrong values in our props

```
const Greeting = (props) => {  
  return (  
    <div> Using the props  
    <h1>Hello, {props.name}</h1>  
    {props.isBirthday} && <p>Happy Birthday to you!</p>  
  </div>  
);  
};
```

The Parent Component

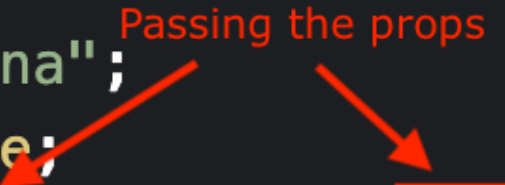
```
const App = () => {  
  const firstName = "Fiona"  
  const isBirthday = true  
  return <Greeting name={firstName} isBirthday={isBirthday} />  
}
```



```
const App = () => {  
  const firstName = "Fiona";  
  const isBirthday = true;  
  return <Greeting name={firstName} isBirthday={isBirthday} />;  
};
```

Passing the props

Text



The diagram consists of two red arrows originating from the text 'Passing the props'. One arrow points to the `name={firstName}` prop in the JSX, and the other points to the `isBirthday={isBirthday}` prop. Additionally, the word 'Text' is written above the `name={firstName}` prop.

Emoji Check:

How do you feel about passing props to a component?

1. 🥲 Haven't a clue, please help!
2. 😞 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 😊 Yes, with team collaboration could try it
5. 😄 Yes, enough to start working on it collaboratively

Children

`props.children` is a special prop that allows us to access a component's children, i.e. anything contained within its `<Component></Component>` tags. This is a useful technique when a component doesn't know about their children ahead of time.

[Codepen \[props.children example\]](#)

React Component Summary

- Can be defined as functions (more common nowadays) or classes
- Name must start with a capital letter (**PascalCase**)
- Receive arbitrary inputs (called props)
- Return React elements describing the UI
- Components can compose other components to create component trees
- Data is passed down the tree as **props** from parent to child

Emoji Check:

How do you feel about React components?

1. 🥲 Haven't a clue, please help!
2. 😞 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 😊 Yes, with team collaboration could try it
5. 😄 Yes, enough to start working on it collaboratively

Exercise - Components - 20mins

Instructor to distribute exercise:

See [./exercises/react-posts/README.md](#)

React Testing

`vite` does not set up Jest and React Testing Libraries by default. We can install the necessary packages by doing the following:

```
npm install -D jest ts-jest \
  "@testing-library/jest-dom" \
  "@testing-library/react" \
  "@testing-library/user-event" \
  "@types/jest";
```

continued next slide...

React Testing

You then need to create a file called `jest.config.ts` with the following contents:

```
export default {
  preset: 'ts-jest',
  testEnvironment: 'jest-environment-jsdom',
  transform: {
    '^.+\\.tsx?$': 'ts-jest'
    // process `*.tsx` files with `ts-jest`
  },
  moduleNameMapper: {
    '\\.(gif|ttf|eot|svg|png)$': '<rootDir>/test/__mocks__/'
  },
}
```


React Testing

Jest is a test runner that lets us run our tests.

React Testing Library provides utilities that facilitate querying the DOM the same way a user would.

It encourages you to test your React apps in the same way a user would use them, that is, finding elements by logical type (combo box, text box) or by their text.

This is a key principle of React Testing Library - the user does not know about `ids`, or `classes` so we should not test by those. It is "opinionated" about interacting with pages like a user does.

Rendering a Component

Say we have a component like this:

```
interface GreetingProps {  
  // Properties  
}  
  
const Greeting = (props: GreetingProps) => {  
  return (  
    <div>  
      <h1>Hello {props.name}</h1>  
      {props.isBirthday && <p>Happy Birthday to you!</p>}  
    </div>  
  )  
}
```

What might we want to test?

What to test?

We want to test:

- What happens when we pass it a `name` prop i.e. `Alice`
- What happens when we pass it no `name` prop
- What happens when we pass it a truthy `isBirthday` prop
- What happens when we pass it a falsy `isBirthday` prop

Render the component

```
it("should render a personalised welcome message", () => {  
  // Arrange + Act  
  render(<Greeting name={"Alice"} isBirthday={true} />)  
  // Assert  
  const foundElement = screen.getByText(/Hello/)  
  expect(foundElement).toHaveTextContent("Hello Alice")  
})
```

The `/Hello/` is a regex matcher.

Notice that the way we are finding the element under test is by its text content and not by class, id or element type.
This mirrors how a user interacts with the page: the user scans for text and is not aware of implementation details like `ids` and `classes`.
To reiterate, this is the "opinionated" bit of the React Testing philosophy.

Render without a result

```
it("do not render a birthday message if not a birthday", () =>
  render(<Greeting name={"Alice"} isBirthday={false} />)
  expect(screen.queryByText(/Birthday/)).toBeNull()
})
```

In the first test we used `getByText` but in this one we used `queryByText` which will return null if not found.

Testing Components Demo

Let's have a look at some component testing in action.

```
cd to ./examples/component-demo and run npm test
```

Some other queries we can use are:

- `getBy...` - fails the test if it can't be found
- `queryBy...` - returns null if it can't be found
- `await findBy...` - waits for the element to appear

These all fail the test if they find multiple matches. If you expect many matches, there are array equivalents:

- `getAllBy...` - fails the test if none are found
- `queryAllBy...` - returns empty array if none are found
- `await findAllBy...` - waits for at least one element to appear

[Cheetsheet](#)

Jest DOM Assertions

Testing Library also provides custom Jest matchers:

```
import "@testing-library/jest-dom/extend-expect"
```

- toBeDisabled
- toBeEnabled
- toBeEmptyDOMElement
- toBeInTheDocument
- toBeInvalid
- toBeVisible
- toContainElement
- toContainHTML

...and many more: [Testing Library Jest Matchers](#)

Emoji Check:

How do you feel about testing a React component?

1. 🥲 Haven't a clue, please help!
2. 😞 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 😊 Yes, with team collaboration could try it
5. 😄 Yes, enough to start working on it collaboratively

Exercise - Write a Component Test - 20mins

Instructor to distribute exercise:

See [./exercises/react-posts-testing/README.md](#)

Emoji Check:

How do you feel about the exercise to write a React component test?

1. 🥲 Haven't a clue, please help!
2. 😞 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 😊 Yes, with team collaboration could try it
5. 😄 Yes, enough to start working on it collaboratively

Quiz time

Which of these is the correct way to define a React component?

- A. `const InfoBox = (props) => <div>{props.text}</div>`
- B. `const InfoBox = (text) => <div>{text}</div>`
- C. `const infoBox = (props) => <div>{props.text}</div>`
- D. `const infoBox = (text) => <div>{text}</div>`

Quiz time

Which of these is the correct way to define a React component?

- A. `const InfoBox = (props) => <div>{props.text}</div>`
- B. `const InfoBox = (text) => <div>{text}</div>`
- C. `const infoBox = (props) => <div>{props.text}</div>`
- D. `const infoBox = (text) => <div>{text}</div>`

Answer: A

Quiz time

Which of these is the correct way to pass the following data `const infoText = "Some great info"` to a child component?

- A. `<InfoBox props={text: infoText}>`
- B. `<InfoBox text=infoText>`
- C. `<InfoBox props={infoText}>`
- D. `<InfoBox text={infoText}>`

Quiz time

Which of these is the correct way to pass the following data `const infoText = "Some great info"` to a child component?

- A. `<InfoBox props={text: infoText}>`
- B. `<InfoBox text=infoText>`
- C. `<InfoBox props={infoText}>`
- D. `<InfoBox text={infoText}>`

Answer: D

Exercise - Todo App - 30mins

Instructor to distribute exercise:

See [./exercises/react-todo/README.md](#)

Overview - recap

- Introducing React
- JSX / TSX
- The React component
- Testing a component

Objectives - recap

- Understand the problem React is aiming to solve
- Understand what a React component is and how to define one
- Become familiar with the parent->child component pattern
- Understand how to pass data from parent to child via props
- Understand how to test a component renders correctly

Emoji Check:

On a high level, do you think you understand the main concepts of this session? Say so if not!

1. 😓 Haven't a clue, please help!
2. 😞 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 😊 Yes, with team collaboration could try it
5. 😄 Yes, enough to start working on it collaboratively