Properties and State

1. Types for properties
2. Working with classes
3. Stateless func components
4. State management

1. Types for Properties

- Problem statement
- Solution - React property types
- How to specify property types

Problem Statement

- JavaScript is a dynamically typed language
  - There's no "compiler" to check that you've assigned the correct type of value to a variable
  - The first you know about a type problem is when the app crashes!
- This is unsatisfactory
  - It makes your code potentially untrustworthy
  - It puts a lot more pressure on your rigour during testing

The Solution - React Property Types

- React allows you to specify the type for properties
  - `React.PropTypes.number`
  - `React.PropTypes.string`
  - `React.PropTypes.bool`
  - `React.PropTypes.array`
  - `React.PropTypes.object`
  - `React.PropTypes.func`
- To use these type flag, you'll need to include the following script file:

```
https://unpkg.com/prop-types/prop-types.js
```

How to Specify Property Types

- When you define a component, specify the types for all its properties
  - How to do this depends on how you define the component (class or stateless functional component)
- You can specify if a property is required/optional
  - You can also specify default values for the optional ones
- We'll investigate all these techniques in this chapter

2. Working with Classes

- Specifying property types
- Required props and defaults
- Defining a custom validator

Specifying Property Types

- In your class, define a static `propTypes` field
  - Specify name and type for the component's properties

```
class Person extends React.Component {
  static propTypes = {
    name: PropTypes.string,
    age: PropTypes.number,
    isWelsh: PropTypes.bool,
    skills: PropTypes.array
  }
  …
}
```

propertyTypes2.html

Specifying Required Props and Defaults

```
class Person extends React.Component {
  static propTypes = {
    name: PropTypes.string.isRequired,
    age: PropTypes.number.isRequired,
    isWelsh: PropTypes.bool,
    skills: PropTypes.array
  }
  static defaultProps = {
    isWelsh: false,
    skills: []
  }
}
```

propertyTypes2.html

Defining a Custom Validator (1 of 2)

- You can define a custom validator for a property
  - E.g. a regular expression pattern for postal codes
  - E.g. the allowed range of values for a number
  - E.g. the maximum number of elements in an array
- See example on next slide

Defining a Custom Validator (2 of 2)

```
const isValidAge = (props, propName) => {
  if (typeof props[propName] !== 'number')
    throw new Error('Must be a number')
  if (props[propName] > 120)
    throw new Error('Max value is 120')
}

class Person extends React.Component {
  static propTypes = {
    name: PropTypes.string.isRequired,
    age: isValidAge,
    isWelsh: PropTypes.bool,
    skills: PropTypes.array
  } …
```

propertyTypes3.html

3. Stateless Func Comps

- Overview
- Key syntactic differences

Overview

- The previous section showed how to do the following for a component class:
  - Specify property types for the component
  - Specify required properties, and default values for others
  - Define custom validation
- You can do all this for functional components too
  - See `propertyTypes4.html`
  - The following slides summarize the key points …

Key Syntactic Differences

- To define property types:
  - Define a static `propTypes` field and specify prop types
  - For required properties, append `isRequired`
  - For optional properties, specify default in function sig
- To define a custom validator:
  - Same as for classes

4. State Management

- Recap component properties
- Fixed vs. mutable state
- Complete example

Recap Component Properties

- We've seen how to pass properties into a component
  - E.g. `Person` component in `propertyTypes1.html`
  - Receives properties `name`, `age`, `isWelsh`, `skills`

```
class Person extends React.Component {
  render() {
    const {name,age,isWelsh,skills}=this.props
    return ( … some elements … )
  }
}
```

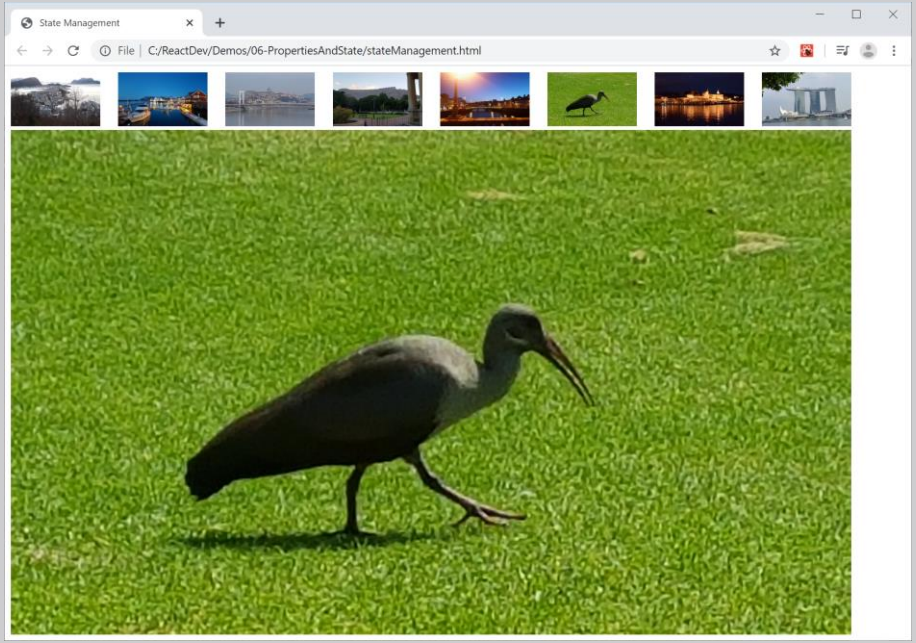
```
<Person name="John Evans"
        age={21}
        isWelsh={true}
        skills={[…]} />
```

Fixed Properties vs. Mutable State

- When you pass properties to a component, they're fixed - you can't change their values thereafter
- What if the component needs to hold mutable state?
  - E.g. add items to an array, update a timestamp, etc.
- Mutable state is available via `this.state`
  - You can put anything into `this.state`, typically in ctor
  - You can modify state via `this.setState(state)`

Complete Example

- See `stateManagement.html`



Summary

- Types for properties
- Working with classes
- Stateless func components
- State management