

# Introduction to React

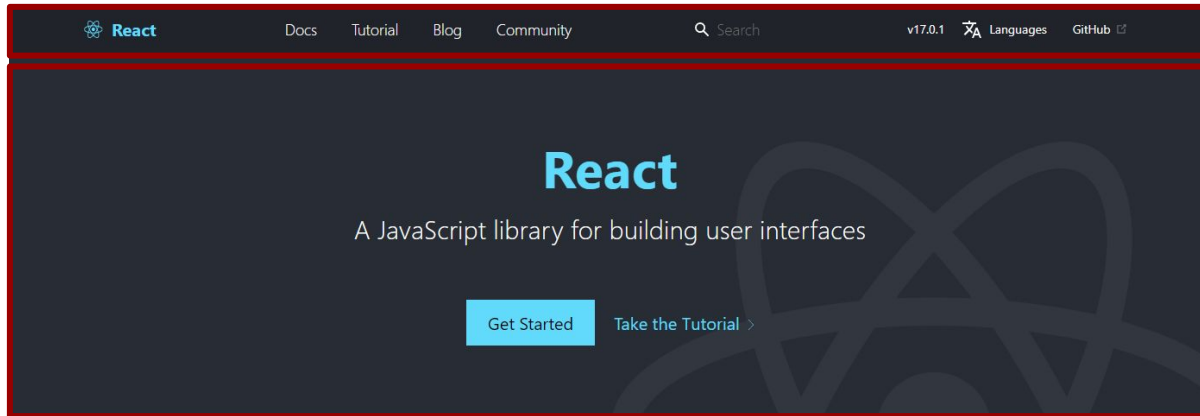
CS142 Section 4

# Agenda

- Review ReactJS
  - Structure
  - JSX
  - Input binding
  - Component lifecycle
  - Hooks (new!)
- Project 4 Tips

# What is React?

React is a JavaScript Library for building user interfaces. It lets you build complex UIs from small and isolated pieces of code called “components”



## Declarative

React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.

Declarative views make your code more predictable and easier to debug.

## Component-Based

Build encapsulated components that manage their own state, then compose them to make complex UIs.

Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.

## Learn Once, Write Anywhere

We don't make assumptions about the rest of your technology stack, so you can develop new features in React without rewriting existing code.

React can also render on the server using Node and power mobile apps using [React Native](#).

# HTML

- Loads compiled javascript
- Provide container `<div id=reactapp></div>` for the app to load at

```
<!doctype html>
<!-- React.js applications are written in JavaScript but you need an
      HTML document to load the JavaScript. This HTML loads the React.js
      cononpment Example -->
<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>CS142 Class Project</title>
    <!-- Insert the model data for the Example widet into the DOM -->
    <script src="modelData/example.js"></script>
  </head>
  <body>
    <div id="reactapp"></div>
    <script src="compiled/gettingStarted.bundle.js"></script>
  </body>
</html>
```

# Example Component

A simple example of a React component.

constructor is called when `<Example />` is passed to `ReactDOM.render`.

render is called after constructor is called.

```
import React from 'react';
import './Example.css';

class Example extends React.Component {
  constructor(props) {
    super(props); // Must run the constructor of React.Component first
    this.state = {
      name: "Ben"
    };
  }

  render(){
    return (
      <div>
        My name: {this.state.name}
      </div>
    );
  }
}

export default Example;
```

# Basic Structure of a React Component

```
1  import React, { Component } from 'react';
2  import './App.css';
3
4  class App extends Component {
5    render() {
6      return (
7        <div className="App">
8          <p>This is the day of joy</p>
9        </div>
10      );
11    }
12  }
13
14  export default App;
15
```

# ReactDOM.render

- 1st parameter: React element to be loaded (in this case an “Example” component)
- 2nd parameter: the HTML element where the provided React element is loaded

```
import React from 'react';
import ReactDOM from 'react-dom';
import './styles/main.css';

import Example from './components/example/Example';
import Header from './components/header/Header';

ReactDOM.render(
  <Example />,
  document.getElementById('reactapp'),
);
```

# Use JSX to Create Elements

JSX is syntactic sugar

```
<p className="myName">Ben</p>
```

is equivalent to:

```
React.createElement("p", {className: "myName"}, "Ben")
```

Note: reserved keywords such as “**class**” and “**for**” need to be changed to “**className**” and “**htmlFor**”.

Other DOM attribute names are also adapted to match JS API names. E.g. **onclick** becomes **onClick**



# Evaluating Values Using {} in JSX

`{this.state.name}` will display as “Ben” in the webpage.

You can pass any JavaScript *expression* to `{}` in JSX.

```
import React from 'react';
import './Example.css';

class Example extends React.Component {
  constructor(props) {
    super(props); // Must run the constructor of React.Component first
    this.state = {
      name: "Ben"
    };
  }

  render(){
    return (
      <div>
        My name: {this.state.name}
      </div>
    );
  }
}

export default Example;
```

# Conditional render in JSX

if statements and for loops are not expressions in JavaScript, so they can't be used in JSX directly.

However, there can be workarounds.

```
render(){  
  let message;  
  if (this.state.name === "Ben"){  
    message = <p>Hello!</p>;  
  } else{  
    message = <strong>Y0!</strong>;  
  }  
  return (  
    <div>  
      I have a message for you: {message}  
    </div>  
  );  
}
```

```
render(){  
  return (  
    <div>  
      I have a message for you:  
      {this.state.name === "Ben" ? <p>Hello!</p> : <strong>Y0!</strong>}  
    </div>  
  );  
}
```

# Iteration in JSX

What if you would like to display an array of elements?

You can do something like this.

In React, you need to specify “key” attribute for each list item. List item’s “key” need to be unique among its siblings.

```
render(){  
  let data = ["Ben", "John", "Andrew"];  
  let myList = [];  
  for (let i = 0; i < data.length; i++){  
    myList.push(<li key={i}>data[i]</li>);  
  }  
  return (  
    <ul>{myList}</ul>  
  );  
}
```

```
render(){  
  let data = ["Ben", "John", "Andrew"];  
  return (  
    <ul>{data.map((d, index) => <li key={index}>{d}</li>)}</ul>  
  );  
}
```

Do note that I am using index as key. This is acceptable if list item has no unique id and the order of items stays unchanged.

# Component State and Input Handling

The input field is bound to `this.state.name`. When you change the input field, it also changes the state value.

`setState` updates state value of the component and calls render again.

Tip: do not call `setState` in render!

```
constructor(props) {  
  super(props); // Must run the constructor of React.Component  
  this.state = {  
    name: "Ben"  
  };  
}  
  
handleChange(event){  
  this.setState({name: event.target.value});  
}  
  
render(){  
  return (  
    <div>  
      <input type="text" value={this.state.name}  
        onChange={(event) => this.handleChange(event)} />  
      <h3>Hello {this.state.name}!</h3>  
    </div>  
  );  
}
```

Hello Ben!



Hello Benjamin!

# Handling Events

Beware of “this” keyword!

This does not work because “this” before setState will be undefined when handleChange is called!

```
handleChange(event){
  this.setState({name: event.target.value});
}
render(){
  return (
    <div>
      <input type="text" value={this.state.name}
        onChange={this.handleChange}/>
      <h3>Hello {this.state.name}!</h3>
    </div>
  );
}
```

# Handling Events - Workaround 1 - Binding

Use bind to create a bound version of the method to preserve this in the new scope.

```
constructor(props) {  
  super(props); // Must run the constructor of React.Component  
  this.state = {  
    name: "Ben"  
  };  
  this.handleChangeBound = this.handleChange.bind(this);  
}  
handleChange(event){  
  this.setState({name: event.target.value});  
}  
render(){  
  return (  
    <div>  
      <input type="text" value={this.state.name}  
        onChange={this.handleChangeBound}/>  
      <h3>Hello {this.state.name}!</h3>  
    </div>  
  );  
}
```

# Handling Events - Workaround 2 - Arrow Function

Use arrow function in JSX to avoid creating additional scope at runtime so that the “this” value of the enclosing scope is used.

```
constructor(props) {  
  super(props); // Must run the constructor of React.Component  
  this.state = {  
    name: "Ben"  
  };  
}  
  
handleChange(event){  
  this.setState({name: event.target.value});  
}  
  
render(){  
  return (  
    <div>  
      <input type="text" value={this.state.name}  
        onChange={(event) => this.handleChange(event)}/>  
      <h3>Hello {this.state.name}!</h3>  
    </div>  
  );  
}
```

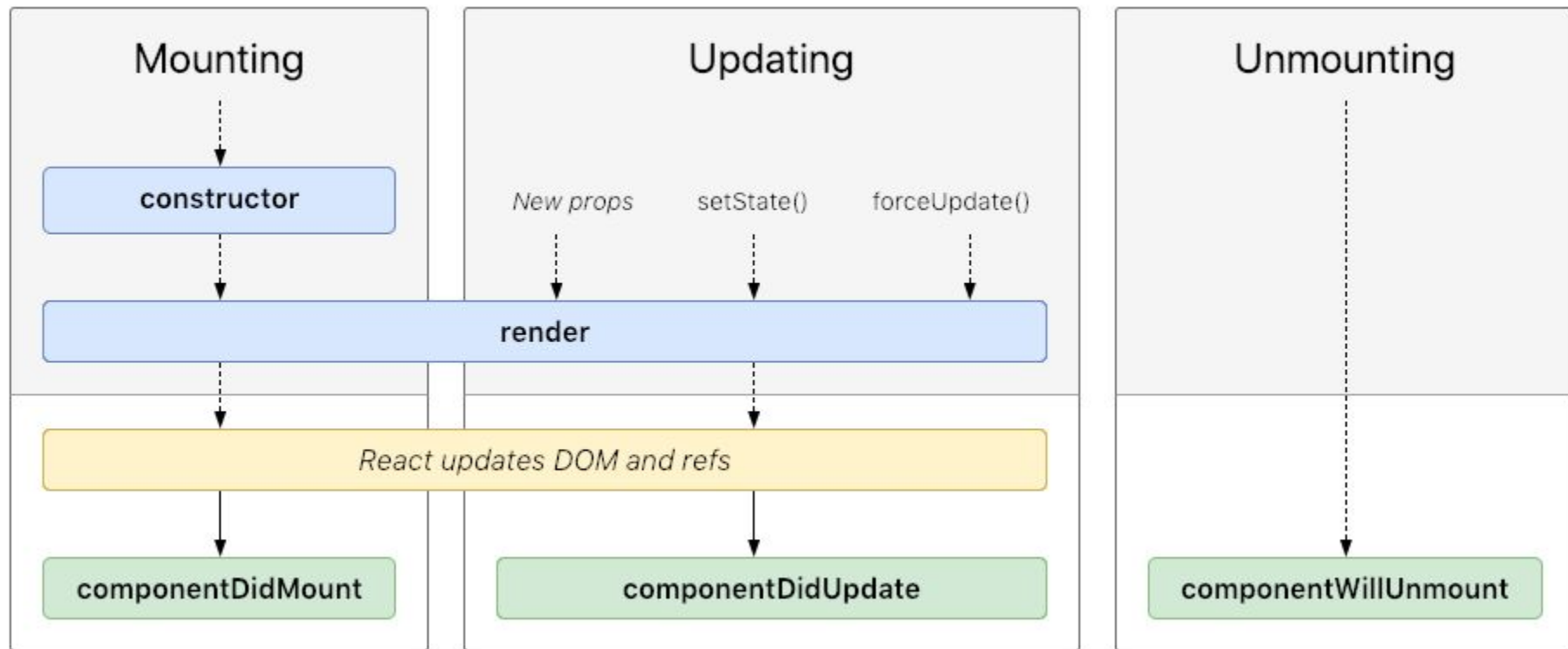
# Handling Events - Workaround 3 - Public Class Field

Define `handleChange` function using public class field syntax and arrow function to correct bind to class.

```
handleChange = (event) => {  
  this.setState({name: event.target.value});  
}  
render(){  
  return (  
    <div>  
      <input type="text" value={this.state.name}  
        onChange={this.handleChange}/>  
      <h3>Hello {this.state.name}!</h3>  
    </div>  
  );  
}
```



# Component lifecycle and methods



# Since React 16.8 - React Hooks

let you use state and other React features *without* writing a class (and use function components instead)

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

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

# useState Hook

useState returns a pair: the current state value and a function that lets you update it

```
function ExampleWithManyStates() {  
  // Declare multiple state variables!  
  const [age, setAge] = useState(42);  
  const [fruit, setFruit] = useState('banana');  
  const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);  
  // ...  
}
```

# useEffect Hook

- Used to perform side effects. You tell React that your component needs to do something after every render.
- You can think of useEffect Hook as componentDidMount, componentDidUpdate, and componentWillUnmount combined
- If your effect returns a function, React will run it when it is time to clean up

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

# Hook Pitfalls

- Only call Hooks at the top level. Don't call Hooks inside loops, conditions, or nested functions.
  - Why?
    - Hooks don't know how and where they are used. They only remember the **order** they are called
    - Imagine an array of state storage. Each useState is like fetching the next element in the array.
    - If you wrap hooks in if-else block, some hooks would be skipped, causing the next hook to retrieve the incorrect element in the array.
- Only call Hooks from React function components. Don't call Hooks from regular JavaScript functions.
  - There is just one other valid place to call Hooks — your own custom Hooks. Check the documentation for more!

# Getting Started with Project 4

# Using a build workflow

Installation: after unzipping the zip file to project 4 directory, run

`npm install`

To compile your code, run

`npm run build`

To run the server, run

`node webServer.js`

Then access the website at `http://localhost:3000`

# Problem 1 - 3: ReactJS Component

- Problem 1:
  - Input binding with state
  - Find model data location
- Problem 2:
  - More input binding
  - Filtering state: involves list iteration in JSX
  - CSS styling is necessary
- Problem 3:
  - Personalization of header layout
  - You need a new header component
  - CSS styling is necessary



## Problem 4 - Dynamic Switching

- Your website should enable the user to freely switch between two different components
- Consider using state as indicator for which view to show, controlling the render method

# Problem 5 - Single Page Application

- Covered in today's lecture
- A lot of hints are given in the spec

Sample usage:

```
<Route path="/myPage" component={MyComponent} />
```

With this line, Route will render MyComponent if URL is matched (e.g. on local server the URL is `http://localhost:3000/myPage`).

# More on Routing: Passing Parameters

```
<Route  
  path="/Book/:book/ch/:chapter"  
  component={BookChapterComponent}  
>
```

An example of link that causes this component to mount:

`http://localhost:3000/Book/3/ch/4`

“:book” and “:chapter” will be passed to the component, and you can access it in the component class with calls like `this.props.match.params.book`.

# Debugging

Two places for debugging: Inspector Tools & Terminal

When webpack experiences errors when bundling your code, it will show these errors in the terminal.

When React experiences errors at runtime in the browser, these errors will appear in the inspector tools console.

```
ERROR in ./gettingStarted.jsx
Module build failed (from ./node_modules/babel-loader/lib/index.js):
SyntaxError: Unexpected token, expected , (9:2)

   7 |   ReactDOM.render(
   8 |     <Example />
>  9 |     document.getElementById('reactapp'),
      |     ^
  10 |   );
  11 |

npm ERR! code ELIFECYCLE
npm ERR! errno 2
npm ERR! project4@1.0.0 build: `webpack -d`
npm ERR! Exit status 2
npm ERR!
npm ERR! Failed at the project4@1.0.0 build script.
npm ERR! This is probably not a problem with npm. There is likely additional logging output above.

npm ERR! A complete log of this run can be found in:
npm ERR!     /Users/keslert/.npm/_logs/2019-02-06T15_23_58_373Z-debug.log
```

[React Developer Tools - Google Chrome](#)

# Model-view-controller (MVC)

- Key insight: the logic that manages data from your application should be separate from the logic that manages how the data is displayed
- Model
  - storing, filtering, deleting, creating, modifying your data
- View
  - displays the data
  - HTML templates
- Controller
  - fetches data from appropriate model, depending on request
  - passes model data to appropriate view, depending on request
  - JavaScript
- Project 4 focuses on understanding the *view and controller*

# MERN Stack

- MongoDB
- Express.js
- **React**
- Node.js

# Getting started

[Link to code](#) (install and run via [instructions in project details](#))