CS142 Section 4

Introduction to React

MERN Stack

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

Agenda

- Review ReactJS
  - Structure
  - JSX (element creation,)
  - Input binding
  - Component lifecycle
- Project 4 Tips

HTML

- Loads compiled javascript
- Provide container   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
     conopnment 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>
```

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'),
);
```

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;
```

Use JSX to Create Elements

Ben

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”.

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>YO!</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>YO!</strong>}
        </div>
        );
}
```

https://reactjs.org/docs/conditional-rendering.html

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.

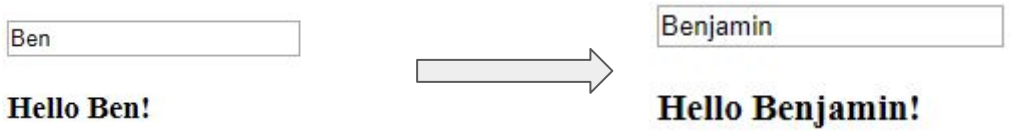
Component State and Input Handling

The input field is binded 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.Compone
    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

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>
        );
}
```

https://reactjs.org/docs/handling-events.html

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.Componen
    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.

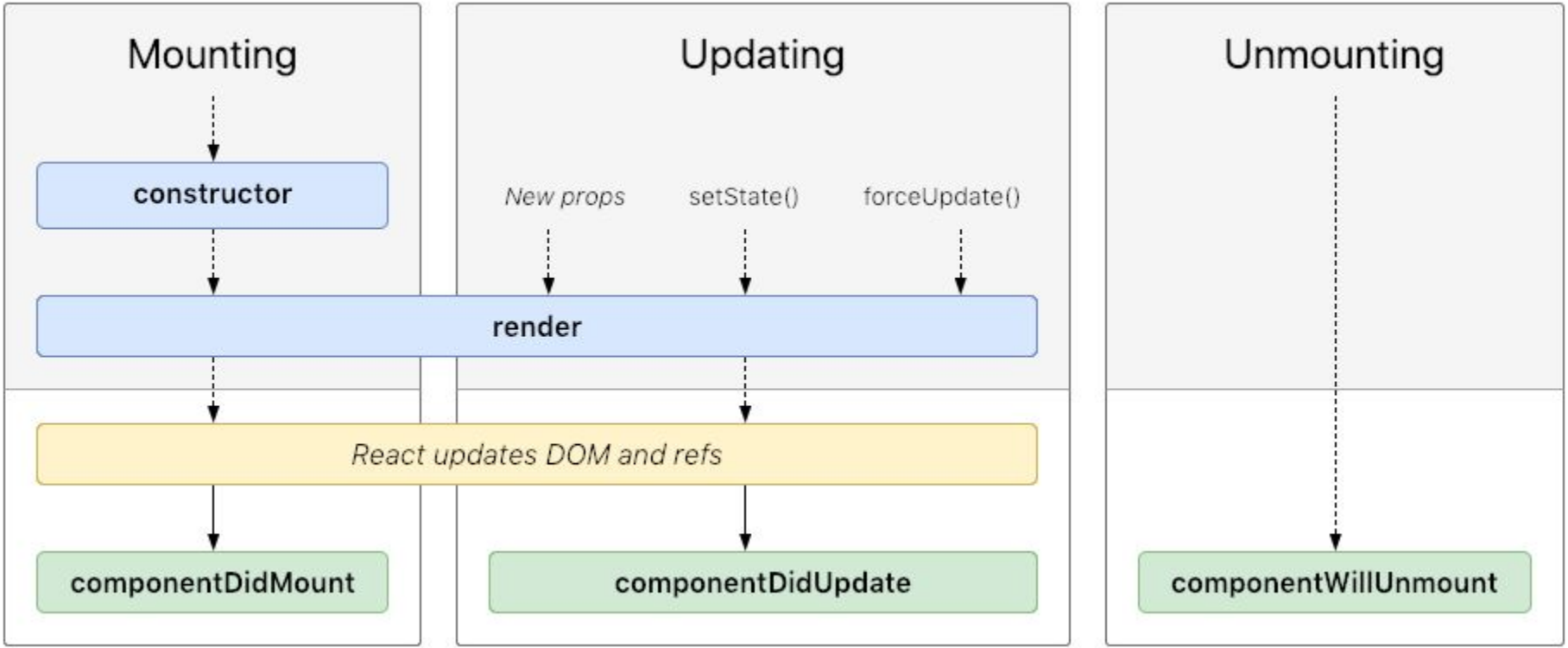
```
constructor(props) {
    super(props); // Must run the constructor of React.Compone
    this.state = {
       name: "Ben"
    };
}
handleChange(event){
    this.setState({name: event.target.value});
}
render(){
    return (
        <div>
            <input type="text" value={this.state.name}
   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



http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/ https://reactjs.org/docs/state-and-lifecycle.html

Getting Started with Project 4

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

- Will be covered in next Monday'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

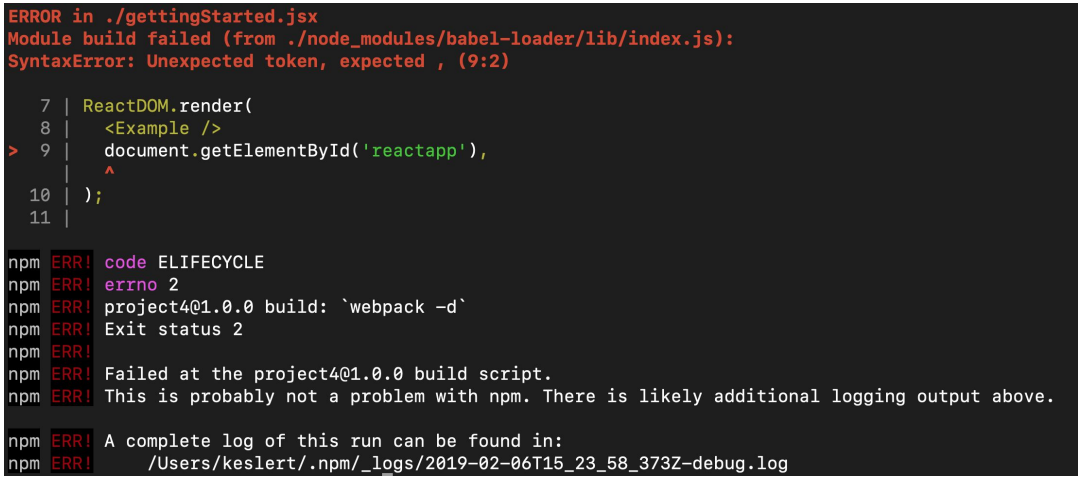
“:book” and “:chapter” will be passed to the component, and you can access it in the component class with calls like this.props.params.book.

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.



[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*

http://web.stanford.edu/class/cs142/lectures/ReactJS.pdf

Getting started

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