React Foundation

Module 4: Advancing Components

Azat Mardan @azat_co

Node University

Before we move on: Did you make improvements to Timer? If not, go back and implement them now.

Props Features

Default Props

Use `defaultProps` class attribute/property to set props on `this.props` if they are not set by the parent.

Default Props Example

```
class Button extends React.Component {
  render(){
    return <button >{this.props.buttonLabel}</button>
  }
}
Button.defaultProps = {
  buttonLabel: 'lorem ipsum'
}
```

Parent With a Missing Props

This parent component `Content` is missing props on 3 Button components:

```
class Content extends React.Component {
  render() {
    return (
      <div>
        <Button buttonLabel="Start"/>
        <Button />
        <Button />
        <Button />
      </div>
    )
  }
}
```

Prop Types

Prop Types

You can set the prop types on React.js classes. If the type doesn't match and you're in development mode, then you'll get a warning in the console.

Note: React.js suppresses this warning in production mode (more on the dev vs. prod later).

Front-end Validation Warning

Warning: Never rely on the front-end user input validation. Use it only for better User Experience (UX) and check everything on the server-side.

Development vs. Production

The way React.js team defines the development mode is when you're using un-minified version, and the production mode is when you're using minified version.

We provide two versions of React: an uncompressed version for development and a minified version for production. The development version includes extra warnings about common mistakes, whereas the production version includes extra performance optimizations and strips all error messages.

Validating Props

Use the `propTypes` property with the object that has props as keys and types as values. React.js types are in the React.PropTypes object. For example:

```
>> React.PropTypes.string

>> React.PropTypes.number

>> React.PropTypes.bool

>> React.PropTypes.object
```

Prop Type Example

This class will have an optional `title` prop of the string type:

```
class Button extends React.Component {
  //...
}
Button.propTypes = {
  title: React.PropTypes.string
}
```

Required Prop Type

To make a prop required just add `isRequired` to the type. This class will have a `handler` prop of function type required:

```
class Button extends React.Component {
  //...
}
Button.propTypes = {
  handler:  React.PropTypes.func.isRequired
}
```

Prop Types Demo

The example in the `module2/prop-types` folder will produce these warnings:

```
Warning: Failed propType: Required prop `handler` was not specified in `Button`. Check the render method of `Content`.
Warning: Failed propType: Invalid prop `title` of type `number` supplied to `Button`, expected `string`. Check the render method of `Content`.
```

Only the unminifed version of React.js shows the warnings—development mode.

Custom Validation

Just return an instance of `Error`. For example, this code validate email with Regular Expression:

```
email(props, propName, componentName) {
  let emailRegularExpression = /^([\w-]+(?:\.[\w-]+)*)@((?:[\w-]+\.)*\w[\w-]{0,66})\.([a-z]{2,6}(?:\.[a-z]{2})?)$/i
  if (!emailRegularExpression.test(props[propName])) {
    return new Error('Email validation failed!')
  }
}
```

Additional Prop Types

There are many additional types and helper methods. Please refer to the documentation:

https://facebook.github.io/react/docs/reusable-components.html#prop-validation

Try the default and prop type demo: http://plnkr.co/edit/wYOMF9?p=preview

Higher-Order Components

```
const LoadWebsite = (Component) => {
  class _LoadWebsite extends React.Component {
    constructor(props) {
      super(props)
      this.state = {label: 'Run'}
      this.state.handleClick = this.handleClick.bind(this)
    }
    // ...
    render() {
      console.log(this.state)
      return <Component {...this.state} {...this.props} />
    }
  }
  return _LoadWebsite
}
```

Rendering Children

Children Components

Instance A:

```
<Content>
  <h1>React.js</h1>
  <p>Rocks</p>
</Content>
```

Instance B:

```
<Content>
  <img src="https://facebook.github.io/react/img/logo.svg"/>
</Content>
```

Children Prop

There's an easy way to render all the children with `{this.props.children}`.

Children Prop Example

For example, we add a `div` and pass along children elements:

```
class Content extends React.Component {
  render() {
    return (
      <div>
        {this.props.children}
      </div>
    )
  }
}
```

Parent

The parent has children `<h1>` and `<p>`:

```
ReactDOM.render(
  <Content>
    <h1>React.js</h1>
    <p>Rocks</p>
  </Content>,
  document.getElementById('content')
)
```

Children is an Array

Children is an Array if n>1. You can access individual elements link this:

```
{this.props.children[0]}
{this.props.children[1]}
```

Children Truthy Check

There's only one element, this.props.children is NOT an array. Use `React.Children.count(this.props.children)` to get the accurate count.

Style Attribute

CSS Style Attribute

You can set the style attribute using JS object literal or JSON and camel case (backgroundImage instead of background-image). For example, the first {} is for object and the second {} is for rendering:

```
<div style={{borderColor: 'blue', fontFamily: 'Arial'}}>
```

Style with Object

Of course, we can define the style as an object and use it in JSX with { }:

```
class Content extends React.Component {
  render() {
    let style = {border: '1px solid blue'}
    return (
      <div style={style}>
        <h1>Hello!</h1>
      </div>
    )
  }
}
```

`componentDidMount()`

The `componentDidMount()` method is invoked when component is inserted into the DOM. You can use this method to perform operations, and/or send AJAX/XHR requests.

`componentDidMount()` Example

Print DOM:

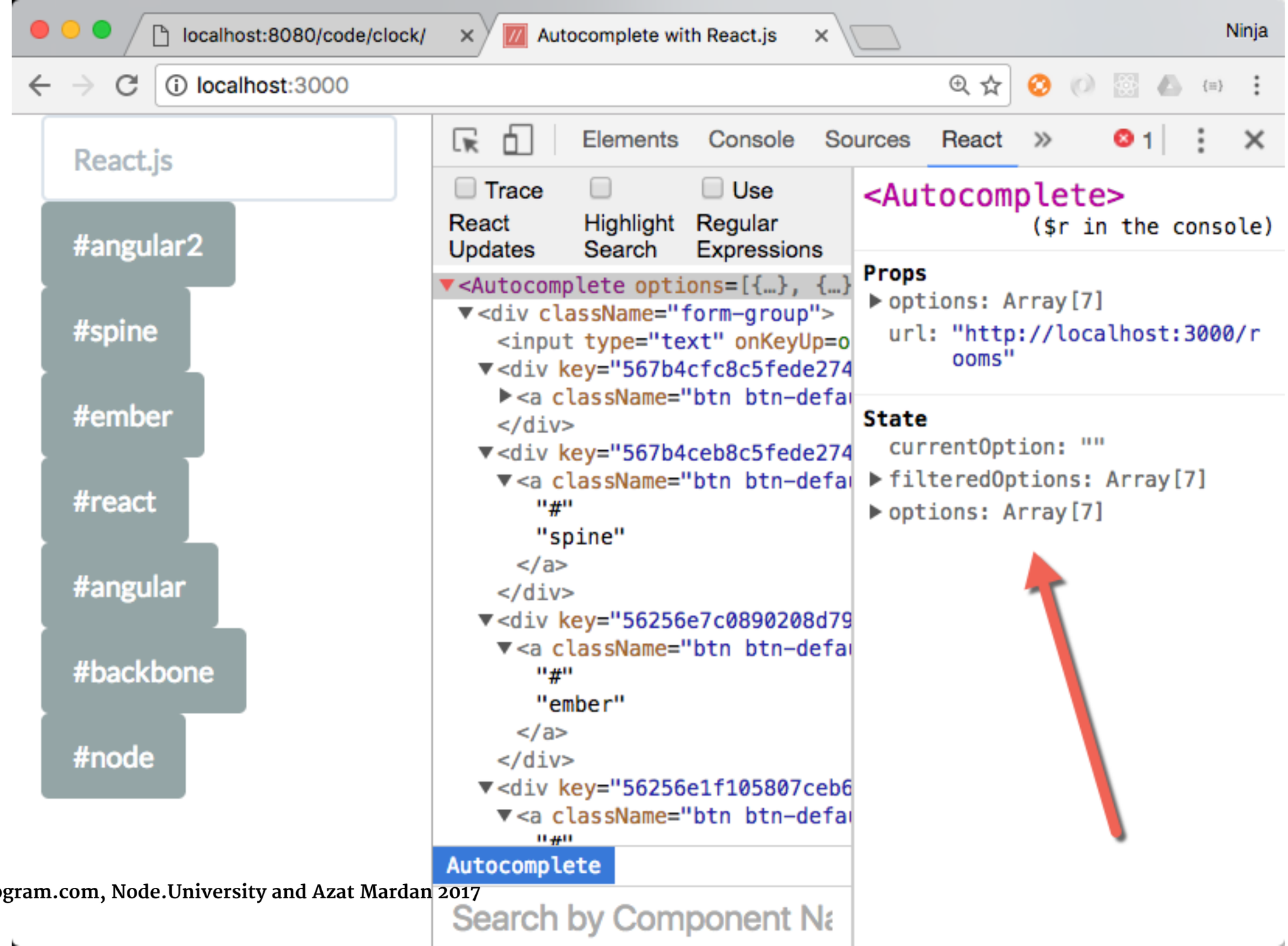
```
class Content extends React.Component {
  componentDidMount() {
    console.log(ReactDOM.findDOMNode(this))
  }
  render() {
    return (
      <div/>
    )
  }
}
```

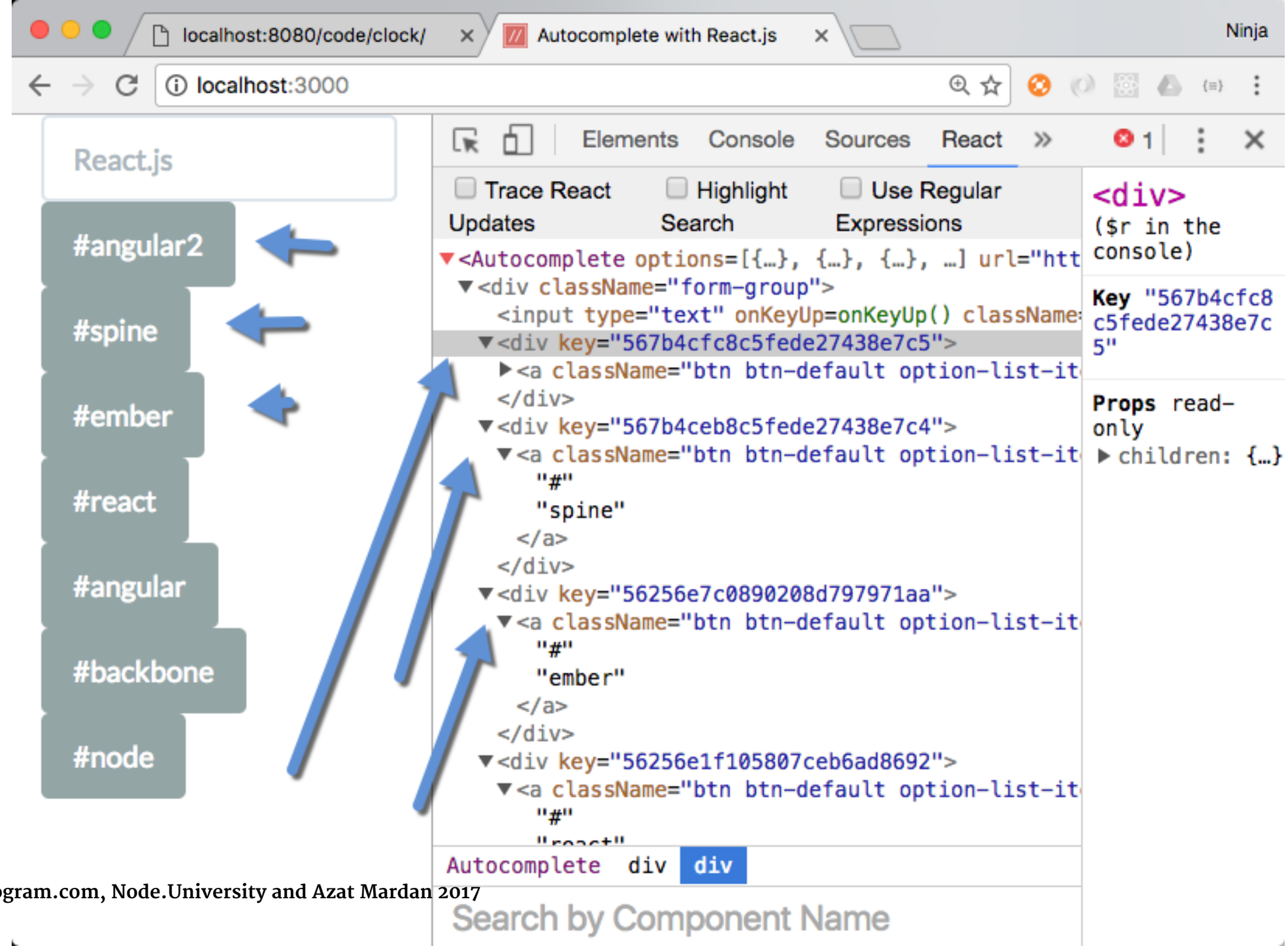
Let's pull the data from the server!

☁️

Autocomplete Project

1. Data: Express, MongoDB, Universal JS
2. Setup: JSX, npm, Babel and Webpack





To run the project:

```
$ npm install
$ npm start
```

Navigate to http://localhost:3000

Demo

Project: Autocomplete

Source code: `code/autocomplete`

Workshop: Autocomplete 🔨💻😁

1. Make it work (`mongod`?)
2. Add remove/delete/x icon/button to *each* chat room in views
3. Add a REST endpoint to delete
4. Add AJAX/XHR call to remove message (pass ID in the URL as as DELETE /rooms/:id)
5. Deploy to cloud: Heroku, now.sh, AWS, etc.