The Virtual DOM

Up until now you may have been wondering how React can possibly manage to update the state of your components? You may have also heard of this magical sounding thing called the Virtual DOM. Well the Virtual DOM is how React works its magic.

Imagine looking at all of the HTML markup on your page, being able to take a snapshot of it in your memory, and know when any little piece of it has changed. That's essentially what the Virtual DOM is.

When your application is initially rendered into the DOM, with it's initial state, React takes a "snapshot" of this and holds it in memory. This is the Virtual DOM.

Any time a DOM update is triggered, React updates the Virtual DOM to reflect the new state of the app. It then does what's called a DOM diff, and selects only the pieces the DOM that have changed. It compares the Virtual DOM, with the actual DOM, and only updates the parts of the DOM that have changed.

This makes updating the application interface extremely fast and efficient, as we do not have to update the entire page to reflect the new state of the app. Also, we, as developers, do not have to programmatically get the reference to the DOM node that needs to be updated, and then perform the update ourselves. We simply change a variable in the state tree and let React do all the work for us.

Putting it into practice

This is great, but let's see it in action. Do you remember when we created the constructor for our component in the last lesson? We set the initial state of the app:

```
this.state = {
  text: 'Foo is baz and bar'
};
```

You may have noticed that the `textarea` in our component is no longer editable. This is because we made it a controlled component in the last lesson, and we have not implemented an onChange handler to update the state. Let's do that now.

```
render () {
  return (
    <div className='container' style={{ marginTop: '50px' }}>
```

```
      <div className='col-lg-8 col-lg-offset-2 form-group'>
        <textarea
          value={this.state.text}
          onChange={::this.updateText}
          className='form-control'
          style={{ height: '500px', resize: 'none' }}>
        </textarea>
      </div>
      <ReadingTime text={this.state.text} className='col-lg-2 well' />
    </div>
  );
}
```

We've added an onChange handler here to handle when a user types something in the `textarea`. You might be asking yourself, what's this double colon syntax? This is new ES6 syntax for binding a function to the current scope of this. It is analagous to `this.updateText.bind(this)`.

Now that we've added a callback function to our onChange handler, we'll need to implement that function just above the `render` function:

```
updateText(ev) {
  this.setState({ text: ev.target.value });
}
```

This is just a simple function that updates the `state` of our component to reflect the new value of the `textarea`. We'll explore event handlers and changing state in more depth in the next lesson, but for now you can quickly see how React handles changes in component state and updates the DOM automagically.