**NOTE 3C:**

Q: What’s Lifecycle in React, and what are some LifeCycle Events?

A: 1) Lifecycle events: special methods each component can have that allows us to hook into the views (UI) when specific conditions happen (time like when the conditions are being created and inserted into the DOM, when the component receives new props etc.)

2) \_componentWillMount: invoked immediately BEFORE the component is INSERTED in the DOM

\_componentDidMount: invoked immediately AFTER the component is INSERTED in the DOM

\_componentWillUnmount: invoked immediately BEFORE a component is REMOVED from the DOM

\_componentWillReceiveProps: invokved whenever the component is about to receive brand new props

Q: Should we put code like Ajax requests in the render() method? Why/ why not?

A: No, because the render method should only be used to render the component and shouldn’t do anything else

Put Ajax requests and the like into LifeCycle Events

**componentDidMount:**

Q: How does it work?

A: If I want to make an Ajax request with React, make it in the componentDidMount like this:

componentDidMount() {

fetchUser().then(user => this.setState({

name: user.name,

age: user.age

}))

render(){

* When the component is at the view, componentDidMount is revoked. It will then initiate an Ajax request. When the request is finished and data is retrieved, setState is called with the newly requested data => sets off render and updates the UI
* This is a good place to instantiate the network request

\_A more in-depth example:

**import** React, { Component } **from** 'react';

**import** fetchUser **from** '../utils/UserAPI';

**class** **User** **extends** **Component** {

constructor(props) {

**super**(props)

**this**.state = {

name: '',

age: ''

}

}

componentDidMount() {

fetchUser().then((user) => **this**.setState({

name: user.name,

age: user.age

}))

}

render() {

**return** (

<div>

<p>Name: {this.state.name}</p>

<p>Age: {this.state.age}</p>

</div>

)

}

}

**export** **default** User;

You'll notice that this component has a componentDidMount() lifecycle event. This component seems pretty straightforward, but let's walk through the order of how it works:

1. The render() method is called which updates the page with a <div> that has one paragraph for the name and one paragraph for the age. What's important to realize is that this.state.name and this.state.age are empty strings (at first), so the name and age *don't actually display*
2. Once the component has been mounted, the componentDidMount() lifecycle event occurs
   * The fetchUser request from the UserAPI is run which sends a request to the user database
   * When the data is returned, setState() is called and updates the name and age properties
3. Since the state has changed, render() gets called again. This re-renders the page, but now this.state.name and this.state.age have values

Let's use componentDidMount() to fetch real users from a server in our Contacts app!

Q: Analyze this piece of code:

import \* as ContactsAPI from './utils/ContactsAPI'

Class App extends Component {

state ={

contacts: []

}

componentDidMount(){

ContactsAPI.getAll().then((contacts) =>{

this.setState({ contacts })

})

}

A: So instead of hard coding in the data, React makes an Ajax request to the server which already has the .getAll method in ComponentDidMount.

Now the code highlighted in yellow is actually this this.setState({ contacts: contacts }) because we name the key as contacts and the value is also contacts. In ES6, when the key and value have the same name, I can just leave on

2 reasons not to make Ajax request in the render method is because the render method should be a pure function and I won’t have complete control over when the render method will be involed (cuz it’s Ajax)

Q: Before our data was able to delete the contact on the front end but it still exists in the backend. How to delete data from the backend?

A: In the backend as defined by ContactsAPI.js, there’s a remove command. If I put that command into the removeContact method like this:

removeContact = (contact) =>{

this.setState((state) => ({

contacts: state.contacts.filter((c) => c.id !== contact.id)

}))

ContactsAPI.remove(contact)

}

If I lose all the contacts then just restart the backend. It’ll make a new default list

**RECAP:**

\_3 kinds of LifeCycle Events:

**Adding to the DOM**

These lifecycle events are called when a component is being added to the DOM:

constructor()

componentWillMount()

render()

componentDidMount()

**Re-rendering**

These lifecycle events are called when a component is re-rendered to the DOM

componentWillReceiveProps()

shouldComponentUpdate()

componentWillUpdate()

render()

componentDidUpdate()

**Removing from the DOM**

This lifecycle event is called when a component is being removed from the DOM

componentWillUnmount()

