React Component Lifecycle

Goals

- Describe what component lifecycle is
- Contrast methods for mounting, updating and unmounting
- Overview the less commonly used lifecycle methods

React Component Lifecycle

Every component comes with methods that allow developers to update application state and reflect the changes to the UI before/after key react “events”.

- There are three main phases to know about:
  - mounting
  - updating
  - unmounting

> **Note: Some things have changed!**
>
> React 16 deprecates several lifecycle methods. If you’ve been using React for a while, you may notice several differences.

Mounting

constructor()

Often used for initializing state or binding event handlers to class instance.

```
class MyComponent extends Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0,
      value: 'Hey There!',
    };

    this.handleClick = this.handleClick.bind(this);
  }
}
```

render()

After the constructor, React calls render(). It tells React what should be displayed. React updates the DOM to match the output of render().

componentDidMount()

- This method runs after the component is mounted
- “Mounting” is the first time the component is rendered to DOM.
- This is a good place to load any data via AJAX or set up subscriptions/timers.
- Calling ***setState()*** here will trigger re-render, so be cautious.

- Let’s start a timer when Clock instance is first rendered to the DOM
- ***componentDidMount()*** method runs after the component has been rendered.

```
class Clock extends Component {
  componentDidMount() {
    this.timerID = setInterval(() => {
      this.tick();
    }, 1000);
  }

  // ...
}
```

componentDidMount is also quite useful for making AJAX requests when the component is mounted

```
class GitHubUserInfo extends Component {
  componentDidMount() {
    axios.get('https://api.github.com/users/facebook')
         .then(response => {
             let user = response.data
             this.setState({ user });
         });
   }

   // ...
}
```

We can also make ***componentDidMount*** an ***async*** function:

```
class GitHubUserInfo extends Component {
  async componentDidMount() {
    let response = await axios.get(
        'https://api.github.com/users/elie');
    let user = response.data
    this.setState({ user });
  }

  // ...
}
```

Updating

This a suitable place to implement any side effect operations.

- syncing up with ***localStorage***
- auto-saving
- updating DOM for uncontrolled components

componentDidUpdate()

This method is called after every render occurs.

You can do a comparison between the previous and current props and state:

```
componentDidUpdate(prevProps, prevState) {
  // you can call setState here as well if you need!
}
```

Unmounting

componentWillUnmount()

When component is unmounted or destroyed, this will be called.

This is a place to do some clean up like:

- Invalidating timers
- Canceling network request
- Removing event handlers directly put on DOM
- Cleaning up subscriptions

Calling ***setState*** here is useless — there will be no re-rendering after this!

- Remember our timer from above?
- We want to clear that timer whenever the Clock is removed.
- This is called “unmounting” in React.

```
class Clock extends Component {
  componentDidMount() {
    this.timerID = setInterval(() => {
      this.tick()
    }, 1000);
  }

  componentWillUnmount() {
    clearInterval(this.timerID);
  }

  // ...
}
```

Visualizing Component Lifecycle

React Lifecycle Methods <http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/>