React.JS Workshop

Demo App
React Docs
Life-Cyle Methods
React Hooks

By Daniel Mburu

Topics

- **Introduction to React.JS**
- **Component Life-Cycle Methods**
  - What are they?
  - Best Practices.
  - Common Pitfalls.
- **React Hooks**
  - Best Practices.
  - Common Pitfalls.
- **React Libraries**
  - Why we need them.
  - How to choose a library for your react-app.

Intro. to React.JS

- A JavaScript Framework
- Uses a virtual DOM to enhance performance
- Uses an efficient diff algorithm to compare versions of the virtual DOM
- Updates and patches to virtual DOM are sent to the real DOM

Component Life-Cycle Methods

- **Life Cycle**
  - Mounting.
  - Updating.
  - UnMounting.
- **Special functions in a react component that are called at specific instances of a component's life-cycle.**

Component Life-Cycle Methods

- **Mounting**
  - constructor()
  - Static getDerivedStateFromProps()
  - render()
  - componentDidMount()

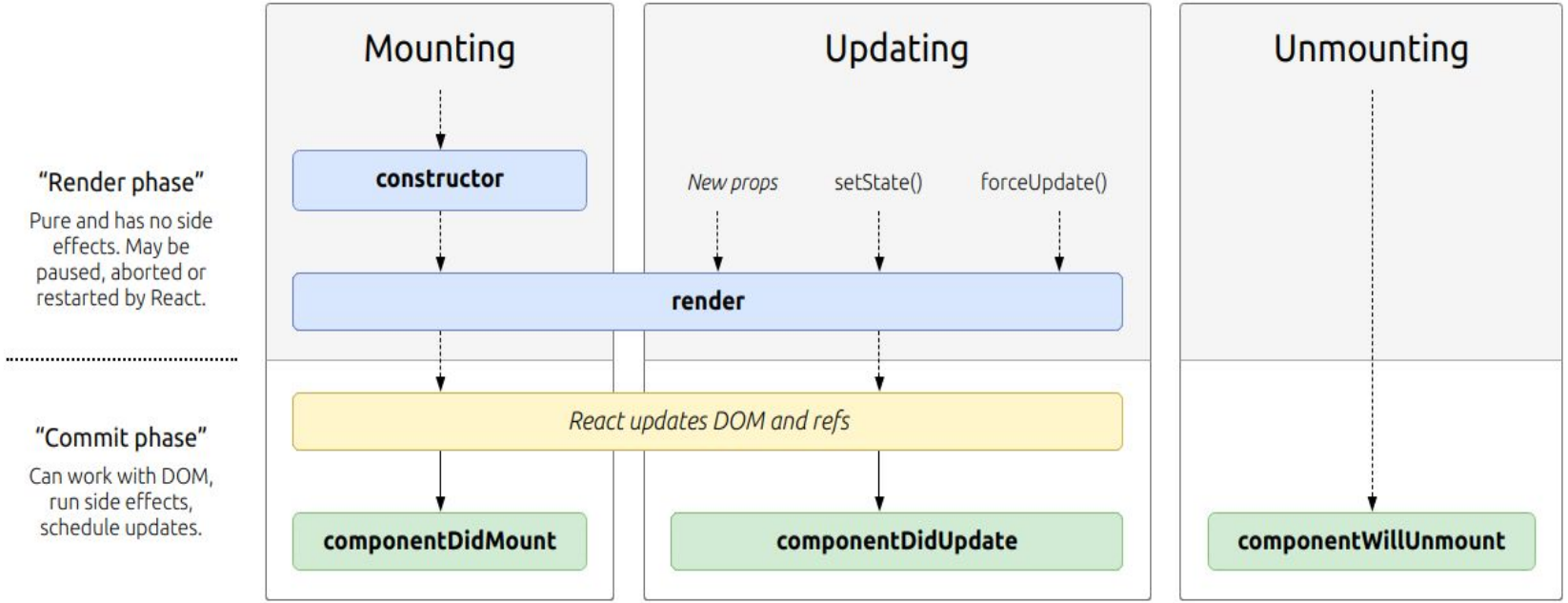
Component Life-Cycle Methods

- **Updating**
  - static getDerivedStateFromProps()
  - shouldComponentUpdate()
  - render()
  - getSnapshotBeforeUpdate()
  - componentDidUpdate()

Component Life-Cycle Methods

- **UnMounting**
  - componentWillUnmount()

Component Life-Cycle Methods



Component Life-Cycle Methods

➔ Common Pitfalls

- Use constructor only if you need to set initial state of the component or require method binding.
- Call “super(props)” before any other statement inside the constructor. Otherwise, this.props will be undefined in the constructor.
- Never use “setState()” in the render-phase (constructor, render()..etc). Only use “setState()” in commit phase.

React Hooks

- New feature in react 16.8
- They allow normal functions to use state and other react features that would normally be accessed only by writing a react class
  - Life cycle methods
  - setState()
- Basically, no more React Classes!!! 🎉🎉🎉

React Libraries, Patterns and Paradigms

- **There are numerous libraries, coding styles, patterns, and paradigms in React that can be used and still help achieve the same level of efficiency and functionality in apps.**
- **The most important this is to maintain common patterns across the team.**
  - High maintainability of existing codebase.
  - Faster debugging.
  - Easy to re-use existing functionality in new modules/apps.
  - Easy to onboard new team members.
  - Higher team productivity.
  - Easy to develop tech. Standards for the team.

Q&A