Getting Start with React

Agenda

- Principle & Components
- Refs & DOM
- Context & Redux
- Best Practice with Components
- Styling Components
- Useful External Libraries
- Create Project and Packing
- Optimizing Performance
- Addition: React Hooks
- Addition: Useful Tools

Demos

- https://codesandbox.io/s/restless-shadow-42bmk?fontsize=14&hidenavigation=1&theme=dark
- https://codesandbox.io/s/nostalgic-lake-b0syk?fontsize=14&hidenavigation=1&theme=dark

Principle

- ***View = fn(data)***
- A specific view can be rolled out with a data
- Once data changed, component re-render

```
import React from 'react'

const Hello = props => {
  return <h1>Hello, {props.name}</h1>
}

ReactDOM.render(<Hello name={"React"}/>, element)
```

Principle

- JSX: syntactic suger for *React.createElement(component, props, …children)*

```
<MyButton color="blue" shadowSize={2}>
  Click Me
</MyButton>
```

Will be compiled by babel into:

```
React.createElement(
  MyButton,
  {color: 'blue', shadowSize: 2},
  'Click Me'
)
```

Principle

- React elements: plan objects

```
<div className="header">Header</div>
<Hello name={"React"}/>
```

- *ReactDOM.render*: create and mount DOMs into root DOM

```
ReactDOM.render(rootElement , document.getElementById('root'))
```

Components

- React Components: reusable pieces of UI
- Function Component & Class Component
- Function Component is ***render function*** in Class Component

```
const Hello = props => {
  return <h1>Hello, {props.name}</h1>
}
```

```
class ClassHello extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>
  }
}
```

- Instantiate the component as a “renderable” React Element

```
<Hello name={"React"}/>
```

Demo 1

Components

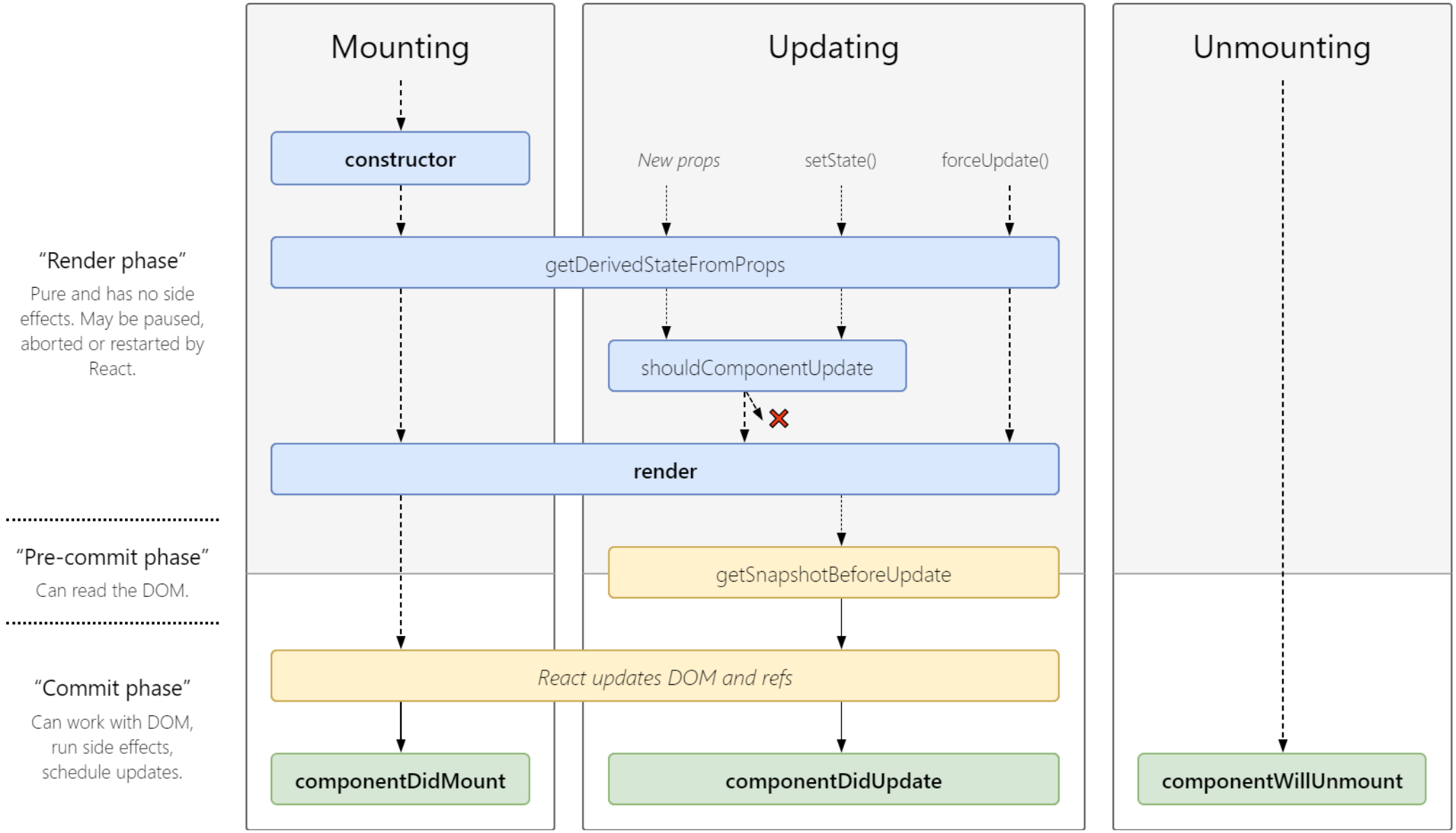
- Props and State
- Props and state are immutable in a render cycle
- Use ***setState()*** and ***getDerivedStateFromProps()*** to update state
- *SetState()* is shallow merge
- Props are read-only, ***do not update props in components***
- Props could be anything: string, object, function, even a react node

```
class ParentComponent extends React.Component {
  state = {
    name: 'React'
  }

  render() {
    const header = <p>{this.state.name}</p>
    const onNameChangeChange = e => {
      this.setState({
        name: e.target.value
      })
    }
    return (
      <ChildComponent
        name={this.state.name}
        header={header}
        onNameChangeChange={onNameChangeChange}
      />
    )
  }
}
```

```
const ChildComponent = props => {
  return (
    <div>
      {props.header}
      <input value={props.name}
             onChange={props.onNameChange}
      />
    </div>
  )
}
```

Demo 2



Component Lifecycles

```
class Clock extends React.Component {
  state = {date: new Date()}

  componentDidMount() {
    this.timerID = setInterval(
      () =>
        this.setState({
           date: new Date()
        }),
      1000
    )
  }

  componentWillUnmount() {
    clearInterval(this.timerID)
  }

  render() {
    return <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
  }
}
```

Event Handling

- Use an arrow function in the callback *(different callbacks will be created in each time the component renders)*
- Add a method on class to be an event handler *(recommend using the class field syntax)*

```
class EventHandling extends React.Component {
  onButtonClick = e => {
    alert(e.target.innerHTML)
  }
  render() {
    return <button onClick={this.onButtonClick}>Click Me!</button>
  }
}
```

Demo 3

Refs & DOM

- *Ref*: a container for mutable data
- Normally used to store an instance of component or a DOM element
- Function components can not be refed *(they don't have instance)*
- *React.forwardRef()*
- Callback Refs https://reactjs.org/docs/refs-and-the-dom.html#callback-refs

```
React.createRef() // => { current: null }

render() {
  return (
    <input type="text" ref={this.textInput} />
  )
}
```

```
class CustomTextInput extends React.Component {
  textInput = React.createRef()

  focusTextInput = () => {
    if (this.textInput.current != null) {
      this.textInput.current.focus()
    }
  }

  render() {
    return (
      <div>
        <input type="text" ref={this.textInput} />
        <button onClick={this.focusTextInput}>Focus</button>
      </div>
    )
  }
}
```

Demo 4

Context

- Share data in component tree without having to pass props down

```
// create context outside of components
const UserContext = React.createContext({user: null, onUserChange: () => {}})

render() {
  const contextValue = {
    user: this.state.user,
    onUserChange: this.onUserChange
  }
  return (
    // place context provider in root of component tree
    <UserContext.Provider value={contextValue}>
      {this.props.children}
    </UserContext.Provider>
  )
}
```

Demo 5

Context

- Get context with Context.Consumer component

```
const NavBar = () => (
  <UserContext.Consumer>
    {({user}) => (
      <nav>
        {user && user.username ? `Welcome ${user.username}` : 'Please Login'}
      </nav>
    )}
  </UserContext.Consumer>
)
```

Context

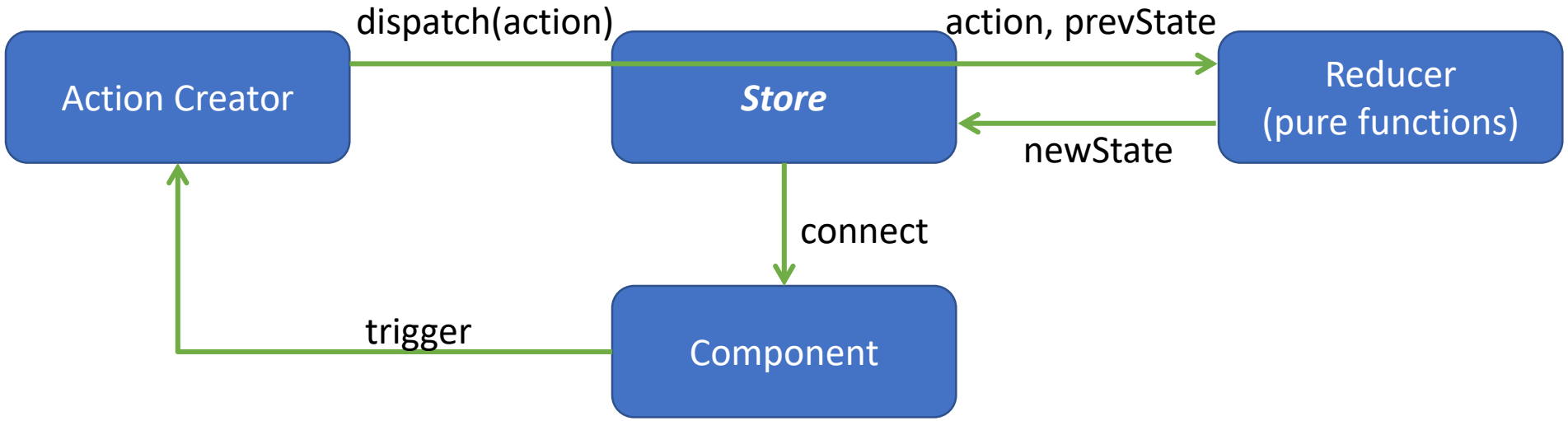
- Set *Class.contextType* and get context from *this.context* in class component

```
class Body extends React.Component {
  // this is class-level property
  static contextType = UserContext

  render() {
    const {user, onUserChange} = this.context
    return (
      <input
        value={user && user.username}
        onChange={e => onUserChange({username: e.target.value})}
      />
    )
  }
}
```

Redux

- Redux: A Predictable State Container for JS Apps
- Normally used to store data needs to be stored and shared globally
- Changes are made with pure functions
- Basic example: https://redux.js.org/introduction/getting-started#basic-example



Redux

- Connect react component with React-Redux

```
const Root = ({children}) => {
  return <Provider store={store}>{children}</Provider>
}
```

- Create stateMapper and ActionMapper

```
const mapStateToProps = count => ({count})
const mapActionToProps = dispatch =>
  bindActionCreators({
    incrementAction,
    incrementWithNumberAction
  }, dispatch)
```

Demo 6

Redux

- Connect component

```
const ConnectedCounter = connect(
  mapStateToProps,
  mapActionToProps
)(Counter)
```

- Get states, dispatch function, actionCreators from props

```
const Counter = ({ count, dispatch, incrementAction, incrementWithNumberAction })
  => {
    // render logics
  }
```

Demo 6

Redux

- Use redux-thunk for async logic
- Normally use for load data from server, or one action with multiple dispatch
- An action creator that returns a function to perform asynchronous dispatch

```
const incrementWithDelay =
  () => (dispatch, getState) => {
    setTimeout(() => {
      dispatch(incrementAction())
    }, 1000)
}
```

Best Practice with Components

Use Higher-Order Components (HOC)

- A higher-order component is a function that takes a component and returns a new component
- Props Proxy: add or filter component props
- Render Hijacking

Best Practice with Components

Use Higher-Order Components (HOC)

- Props Proxy

```
const Navbar = ({username}) => {
  return <header>{username}</header>
}

const withGlobalUsername = WrappedComponent => {
  return props => {
    const {username} = getUserContext()
    return <WrappedComponent {...props} username={username} />
  }
}

const ConnectedNavbar = withGlobalUsername(Navbar)
```

Best Practice with Components

Use Higher-Order Components (HOC)

- Render Hijacking

```
const loginRequired = WrappedComponent => {
  class LoginRequiredWrapper extends React.Component {
    render() { return (
      <AuthContext.Consumer>
        {({isLogin}) => {
          if (!isLogin) { return <p>Access Declined</p> }
          return <WrappedComponent {...this.props} />
        }}
      </AuthContext.Consumer>
    )}
  }
  return LoginRequiredWrapper
}
```

Best Practice with Components

Use Higher-Order Components (HOC)

- Render Hijacking

```
const loginRequired = WrappedComponent => {
  class LoginRequiredWrapper extends WrappedComponent {
    render(){
      if(!this.props.isLogin){
        return <div>Access Declined</div>
      }
      super.render()
    }
  }
  return LoginRequiredWrapper
}
```

Best Practice with Components

Fetch async data

- Fetch data in *componentDidMount()*
- Why? No matter fetch data in what stage, the view will re-render at least twice.
- If you really need to pre-fetch data, you can call the async function in render *(this may cause duplicate http requests, and don't forget to add condition)*

```
class Todos extends React.Component {
  state = { todos: null }

  fetchTodos = () => {
   fetch('https://jsonplaceholder.typicode.com/todos?size=5').then(res => res.json())
    .then(todos => { this.setState({todos: todos}) })
  }

  componentDidMount() {
    this.fetchTodos()
  }

  render() {
    const {todos} = this.state
    if (todos === null) { return <div>Loading...</div> }
    return (<ul>
      {this.state.todos.slice(0, 5).map(item => <li key={item.id}>{item.title}</li>)}
    </ul>)
  }
}
```

Demo 7

Best Practice with Components

State derived from props/state

- Calculate derived state in *getDerivedStateFromProps*

```
class Child extends React.Component {
  state = {
    name: Child.getName(this.props.firstName, this.props.lastName)
  }

  static getName = (firstName, lastName) => {
    if (!firstName && !lastName) return 'Please Input Your name'
    return `${firstName} ${lastName}`
  }

  static getDerivedStateFromProps(props, prevState) {
    const name = Child.getName(props.firstName, props.lastName)
    if (name !== prevState.name) {return {name}}
    return null
  }

  render() {
    return <h3>{this.state.name}</h3>
  }
}
```

Demo 8

Best Practice with Components

Event Listener/Subscription

- Add event listeners/subscriptions in *componentDidMount* *(or ComponentDidUpdate, otherwise DOMs may not be accessible)*
- Remove event listeners/subscriptions in *componentWillUnmount*

Best Practice with Components

External function calls

- Call external function (side effects, mutations) in *componentDidMount* or *componentDidUpdate*

```
class ExampleComponent extends React.Component {
  componentDidUpdate(prevProps, prevState) {
    if (this.state.article.title !== prevState.article.title) {
      updateDocumentTitle(this.state.article.title)
    }
  }
}
```

Best Practice with Components

Fetch async data when props change

- Save the required props field in state
- Capture props change and update flag in *getDerivedStateFromProps*
- Determining whether to fetch data and call async fetch data method in *componentDidUpdate*

See: Demo9

Best Practice with Components

Store external data/form in redux

- Store external data in redux store
- Control data-fetching flow in actions and redux store
- Trigger action when state/props change
- *Advantages: easy to control data flow, time travel, separating data and view*
- *Disadvantage: development efforts, not all data needs time travel*

See: Redux-time-travel-demo

Demo *Redux-time-travel-demo*

Best Practice with Components

Stateful & Stateless components

- Reduce stateful components as possible
- Only one component of a business module contains business-related states *(Container Component)*
- Use function components in other components as possible, they only render views based on props, use state only when need to control the style. *(View Component)*
- *Advantages: easy to manage states, higher performance in views, easy to optimizing*

Styling Components

- Inline
- Use CSS and className
- CSS Modules
- CSS in JS

Styling Components

- Import CSS file
- Add className on elements

```
import React from 'react'
import './styles.css'

const ClassNameStyle = () => {
  return <div className="header">Header</div>
}
```

Demo 10

Styling Components

- Create style.***module***.css
- Import CSS file with name
- Add className on elements
- https://webpack.js.org/loaders/css-loader/#modules

```
import React from 'react'
import styles from './styles.module.css'

const CssModuleStyle = () => {
  return <div className={styles.header}>Header</div>
}
```

Demo 10

Styling Components

- Create style.***module***.css
- Import CSS file with name
- Add className on elements
- *Advantages: avoid style issue with same class name*
- https://webpack.js.org/loaders/css-loader/#modules

Styling Components

- "CSS-in-JS" refers to a pattern where CSS is composed using JavaScript instead of defined in external files.
- *Styled-components*: https://github.com/styled-components/styled-components
- *Advantages: flexible, generate style from params*

```
import React from 'react'
import styled from 'styled-components'

const StyledHeader = styled.header`
font-size: 24px;
font-weight: bold;
`
const Presentation = () => <StyledHeader>StyledHeader</StyledHeader>
```

Demo 10

Useful External Libraries

- Redux, react-redux, redux-thunk, @reduxjs/tookit
- Immer
- React-router-dom
- React-helmet, react-side-effect
- Classnames
- @welldone-software/why-did-you-render
- Prop-types
- Jest, enzyme

Create Project and Packing

- Packing with *webpack* https://github.com/facebook/create-react-app/blob/master/packages/react-scripts/config/webpack.config.js
- *create-react-app* (facebook official) https://www.github.com/facebook/create-react-app
- *react-scripts* (inside cra) https://github.com/facebook/create-react-app/tree/master/packages/react-scripts
- Lightly customize:
  - creact-app-rewired https://github.com/timarney/react-app-rewired/
  - *customize-cra* *(relies on react-app-rewired)* https://github.com/arackaf/customize-cra
- Manually configurate: https://create-react-app.dev/docs/alternatives-to-ejecting/
  - Fork create-react-app and configurate yourself
  - Run `react-scripts eject`

Optimizing Performance

Reduce re-render times

- Implement *shouldComponentUpdate (Class Components)* or use *React.memo (Function Components)*
- Use *React.PureComponent (shallow compare props and state)*

Optimizing Performance

Code Splitting

- Use *import()* syntax to dynamic import packages

```
foo = () => {
  import('lodash').then(_ => {
    _.get(this.state.foo, 'bar’)
  })
}
```

- Use *React.lazy* https://reactjs.org/docs/code-splitting.html#reactlazy

Optimizing Performance

Virtualize Long lists and Lazy Load Resources

- React-virtualized https://github.com/bvaughn/react-virtualized

Others

- https://reactjs.org/docs/optimizing-performance.html

Addition: React Hooks

- Hooks let you use state and other React features in function components
- Make logic in component easier to understand *(No lifecycles, more like functional programming)*
- Can NOT completely replace class components

Addition: React Hooks

- *React.useState*

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {caount} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Demo 11

Addition: React Hooks

- *React.useEffect:* run your “effect” function after flushing changes to the DOM
- Add deps as second parameter *(empty list for run once)*
- Return a function in callback and react will run it on component unmount

```
// Similar to componentDidMount and componentDidUpdate:
useEffect(() => {
  // Update the document title using the browser API
  document.title = `You clicked ${count} times`
}, [count])
```

Demo 11

Addition: React Hooks

- Return a function in callback and react will run it on component unmount

```
function FriendStatus(props) {
  const [isOnline, setIsOnline] = useState(null);

  function handleStatusChange(status) { setIsOnline(status.isOnline); }

  useEffect(() => {
    ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
    };
  });

  if (isOnline === null) { return 'Loading...’; }
  return isOnline ? 'Online' : 'Offline';
}
```

Addition: React Hooks

- *React.useContext:* subscribe to React context without introducing nesting

```
function Example() {
  const locale = useContext(LocaleContext);
  const theme = useContext(ThemeContext);
  // ...
}
```

Addition: React Hooks

- React.useRef : get refs in function components

```
function TextInputWithFocusButton() {
  const inputEl = useRef(null)
  const onButtonClick = () => {
    // `current` points to the mounted text input element
    inputEl.current.focus()
  }
  return (
    <>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>Focus the input</button>
    </>
  )
}
```

Demo 11

Addition: React Hooks

- Create a custom hook:

```
const useInputValue = initValue => {
  const [value, setValue] = useState(initValue)
  const handleInputChange = e => {
    setValue(e.target.value)
  }
  return [value, handleInputChange]
}

const CustomInput = () => {
  const [value, setValue] = useInputValue(undefined)
  return <>
      <input value={value} onChange={setValue} />
      <p>you input: {value}</p>
  </>
}
```

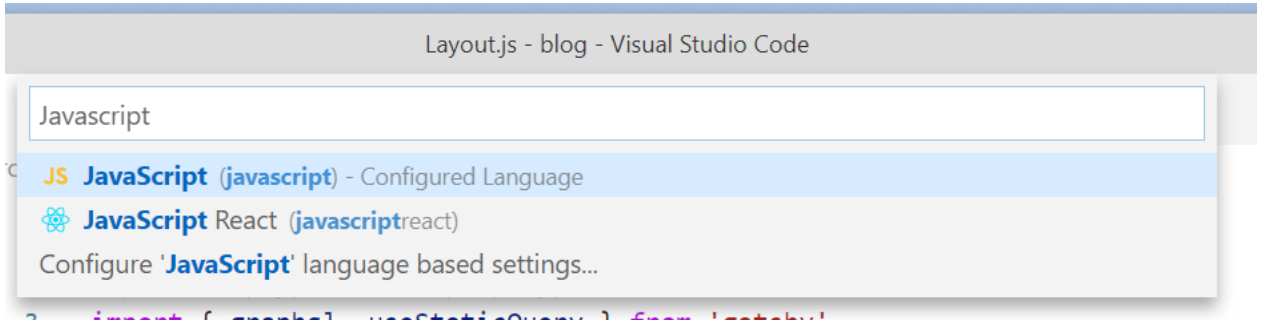
Demo 11

Addition: React Hooks

- React.useReducer
- React.useCallback
- React.useMemo
- React.useImperativeHandle
- React.useLayoutEffect
- React.useDebugValue
- https://reactjs.org/docs/hooks-intro.html

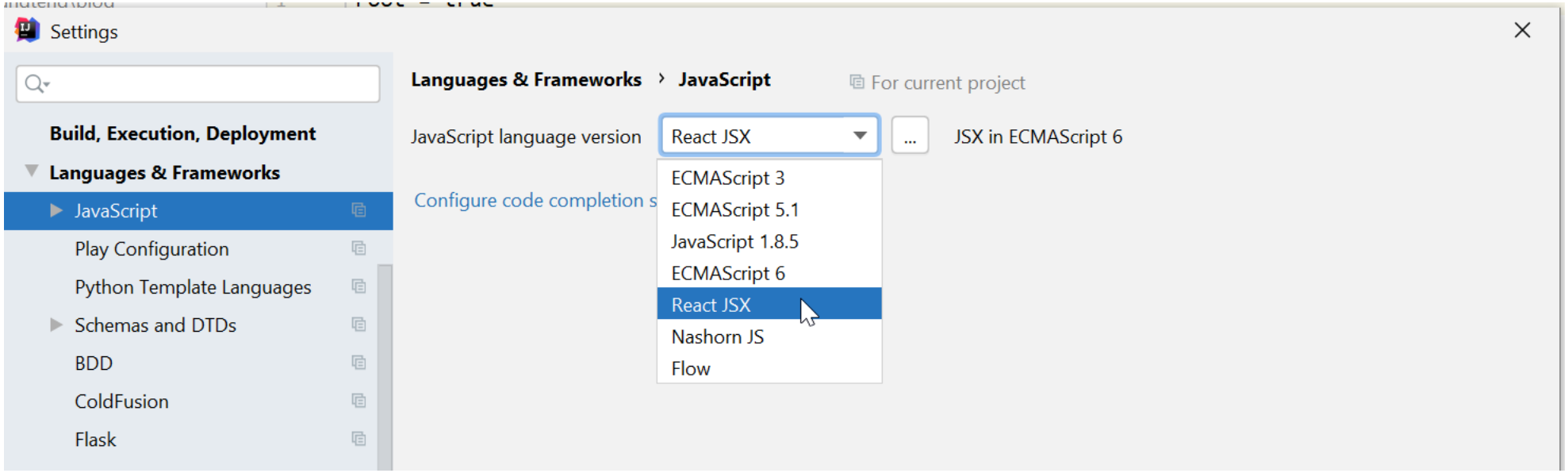
Addition: Useful Tools

- Development: IDEA / VSCode *(build-in support)*



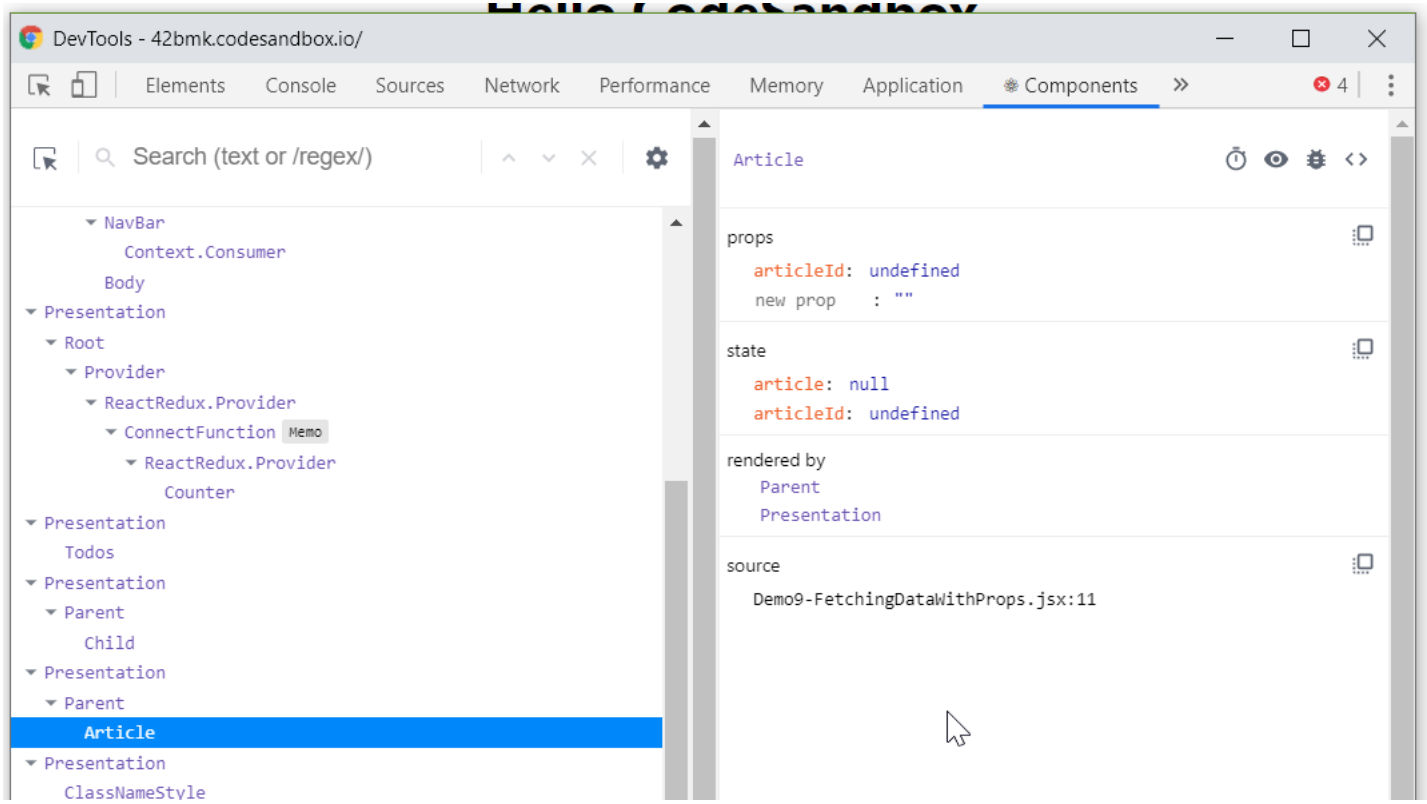
Addition: Useful Tools

- Development: IDEA / VSCode *(build-in support)*



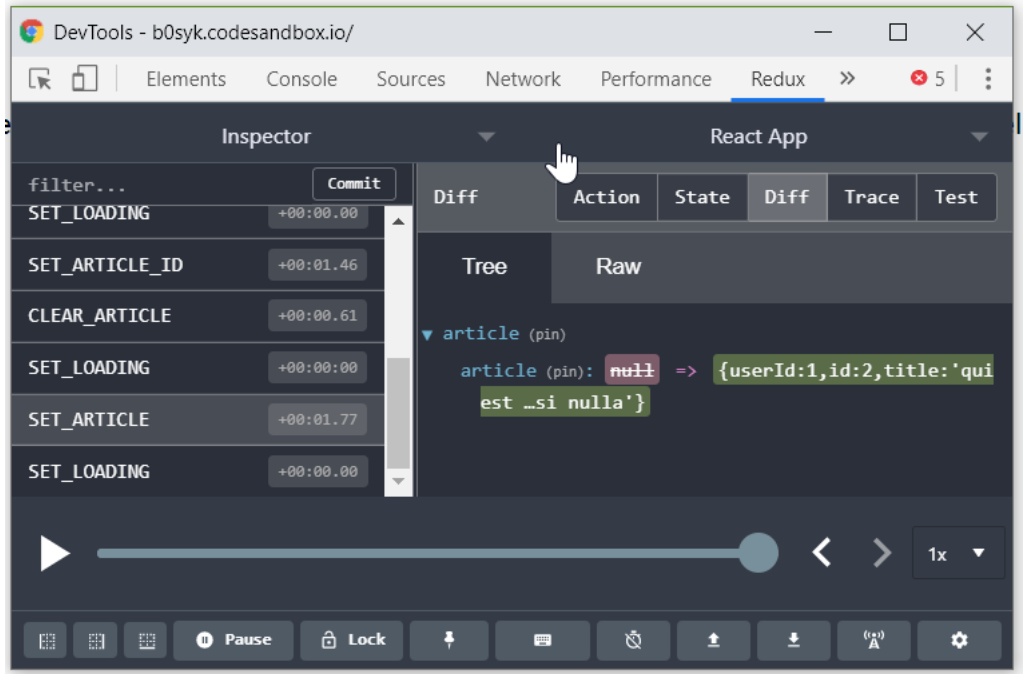
Addition: Useful Tools

- React DevTools
  - https://github.com/facebook/react/tree/master/packages/react-devtools-extensions



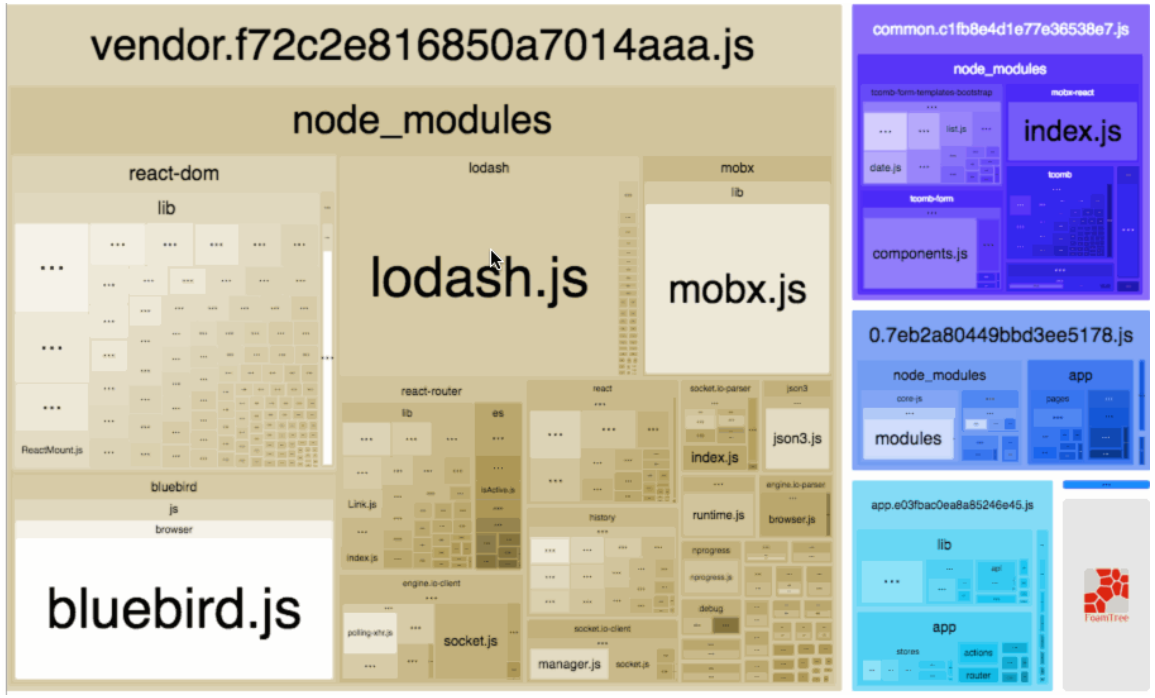
Addition: Useful Tools

- Redux DevTools
  - https://github.com/zalmoxisus/redux-devtools-extension



Addition: Useful Tools

- Webpack Bundle Analyzer
  - https://github.com/webpack-contrib/webpack-bundle-analyzer



Links

- https://reactjs.org/docs/getting-started.html
- https://create-react-app.dev/
- https://github.com/enaqx/awesome-react

Thanks