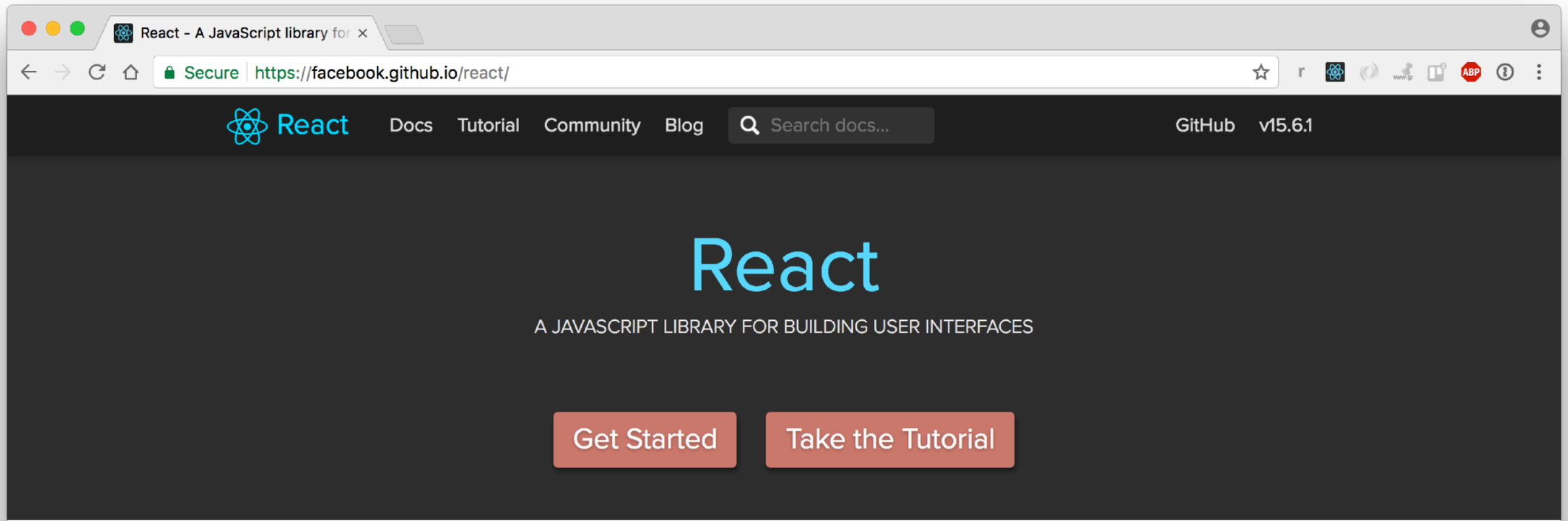
React I

A JS library to build views

What is React?

A **JavaScript library** used to **produce HTML**
from a series of dynamic **components**



Boilerplate

Start from https://github.com/lewagon/react-boilerplate

```
cd ~/code/<github_nickname>
git clone git@github.com:lewagon/react-boilerplate.git react-giphy
cd react-giphy
yarn install
rm -rf .git
git init
git add .
git commit -m "Start new project from lewagon/react-boilerplate"
yarn start
```

Component

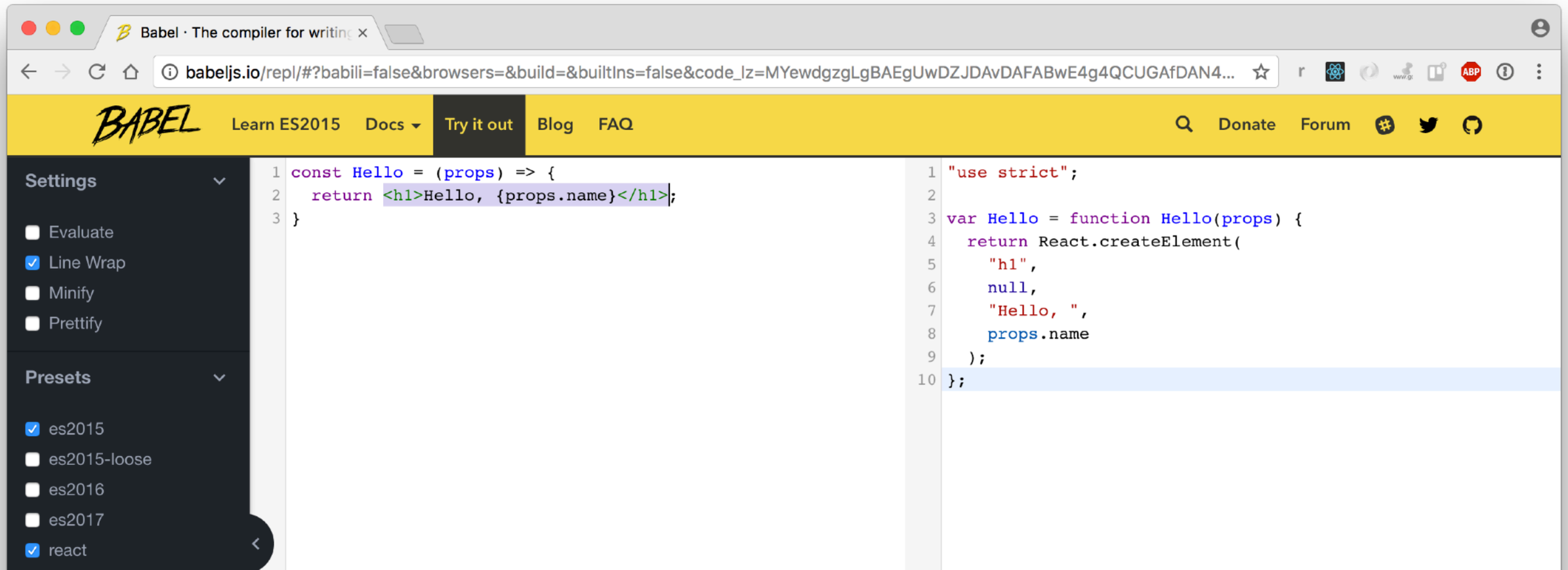
A collection of **JavaScript functions**
Some JavaScript that **returns HTML**
We'll use the **jsx syntax extension** for that

```
// src/index.js
import React from 'react';
import ReactDOM from 'react-dom';

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

const container = document.getElementById('root');
ReactDOM.render(<Hello name="Boris" />, container);
```

JSX

Tag syntax **looking like** HTML
Inject **code** with **{}**
Produces the expected **HTML**

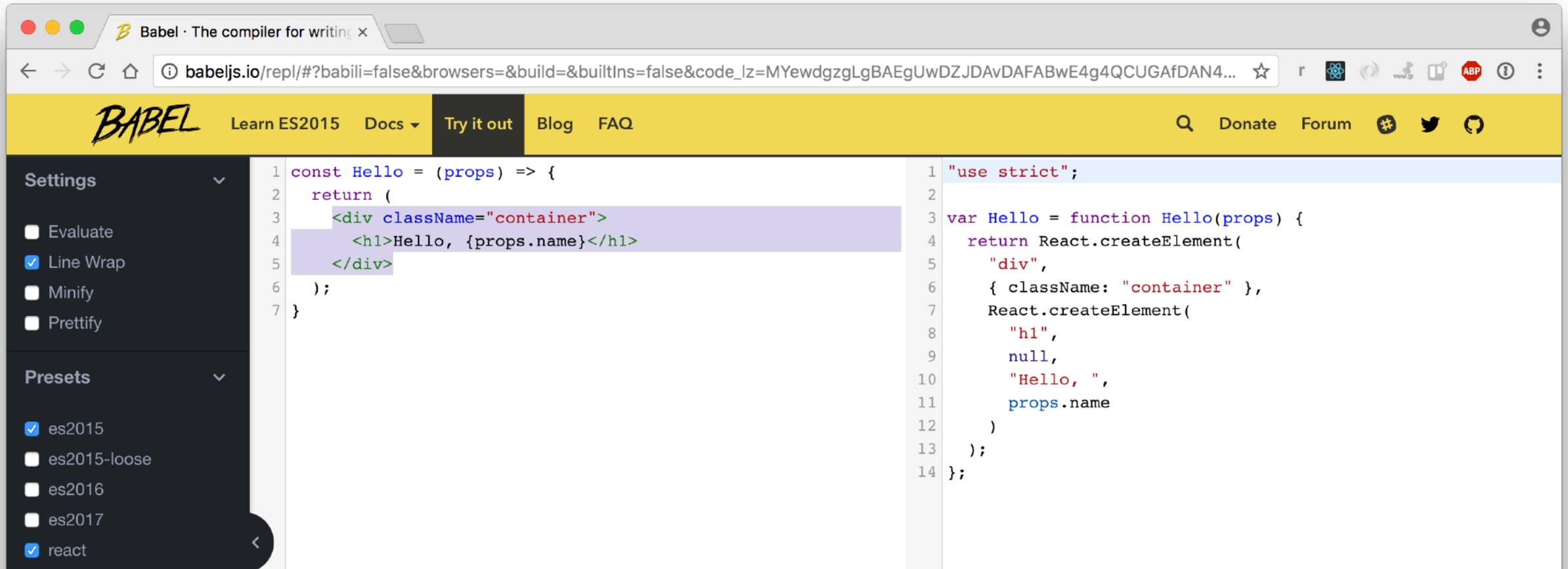
⚠️ **babel-preset-react** is needed by webpack to **transpile** jsx into js



JSX

Multiline possible with **( )**

⚠️ **`class`** HTML attribute becomes **`className`**



Props

A **JavaScript object**
Stores the component's **properties**
Immutable during the component's lifespan

```
// src/index.js

// [...]
// This component is a stateless (or a functional) component
const Hello = (props) => {
  return <h1>Hello, {props.name}</h1>;
}

const helloBoris = <Hello name="Boris" />;
// the value "Boris" won't change during the time helloBoris is
// displayed in the DOM. "Boris" is an intrinsic property of the
// `helloBoris` instance of the `Hello` component.
```

State

Also a **JavaScript object**

Stores the component's **current local state**

Can change during the component's lifespan

```
// src/index.js
import React, { Component } from 'react';

class Hello extends Component { // A stateful component needs to
  constructor(props) {          // be promoted into a class
    super(props);

    this.state = { clicked: false }; // defines initial state
  }

  render() {
    return (
      <h1 className={this.state.clicked ? "clicked" : ""}>
        Hello, {this.props.name}
      </h1>
    );
  }
}
```

Render

Every class-based component **must have** a **render** method

```
// src/index.js
import React, { Component } from 'react';

class Hello extends Component { // A stateful component needs to
  constructor(props) {          // be promoted into a class
    super(props);

    this.state = { clicked: false }; // defines initial state
  }

  render() {
    return (
      <h1 className={this.state.clicked ? "clicked" : ""}>
        Hello, {this.props.name}
      </h1>
    );
  }
}
```

Events

Always use **`this.setState()`** to update state
Anytime the **`state`** changes, **`render()`** is called back

```
// src/index.js
import React, { Component } from 'react';

class Hello extends Component {
  // [...]

  handleClick() {
    this.setState({ clicked: true });
  }

  render() {
    return (
      <h1 className={this.state.clicked ? "clicked" : ""}
          onClick={this.handleClick}>
        Hello, {this.props.name}
      </h1>
    );
  }
}
```

Events

To bind **`this`** with arrow function callbacks in classes:

```
// src/index.js
import React, { Component } from 'react';

class Hello extends Component {
  // [...]
 handleClick = () => {
    this.setState({ clicked: true });
  }
  // [...]
}
```

Add **babel-plugin-transform-class-properties**

```
yarn add babel-plugin-transform-class-properties --dev
```

```
# .babelrc
{
  "presets": [ "es2015", "react" ],
  "plugins": [ "transform-class-properties" ]
}
```

Wrap-up

A **component** is responsible for producing **one piece of HTML**

Static data is carried by the component's **`props`**
Mutable data is carried by the component's **`state`**
State and **events handling** make **interactivity** possible

Stateless components can be coded as **functions**
Stateful components need to be promoted as **classes**

Anytime a component's **`state`** **changes** (`this.setState( )`),
its **`render( )`** method is called back

Good to know

ALWAYS use **`this.setState()`** to update a component's state

```
// src/index.js
// [...]
class Hello extends Component {
  // [...]
 handleClick = () => {
    this.state.clicked = true; 😱⛔☠️
    this.setState({ clicked: true }); //
  }
  // [...]
}
```

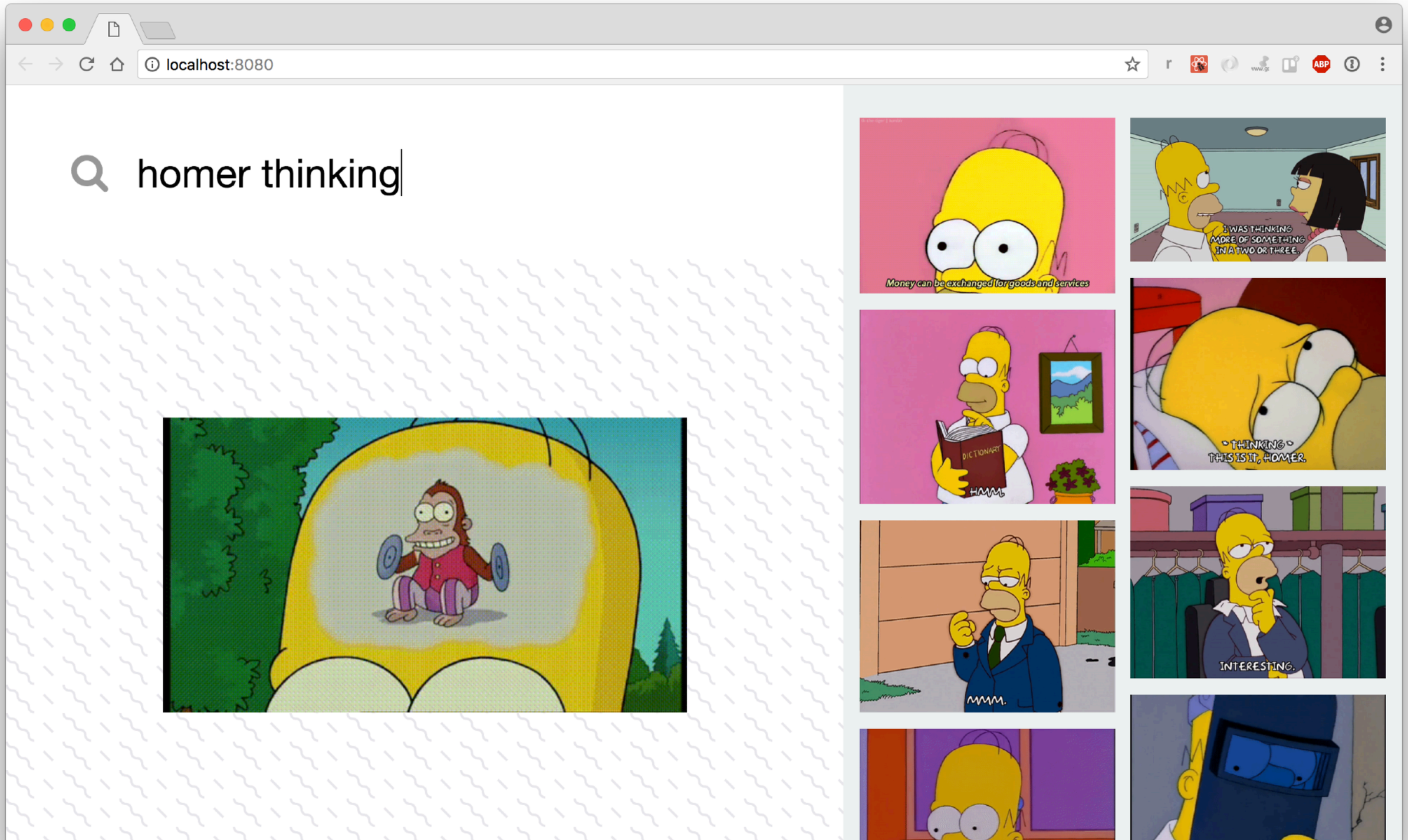
Never try to mutate `this.state` directly

Our first React App

Sublime Configuration

- Install **Babel** from Package manager
- Add this key bindings

Let's build an App!



Boilerplate

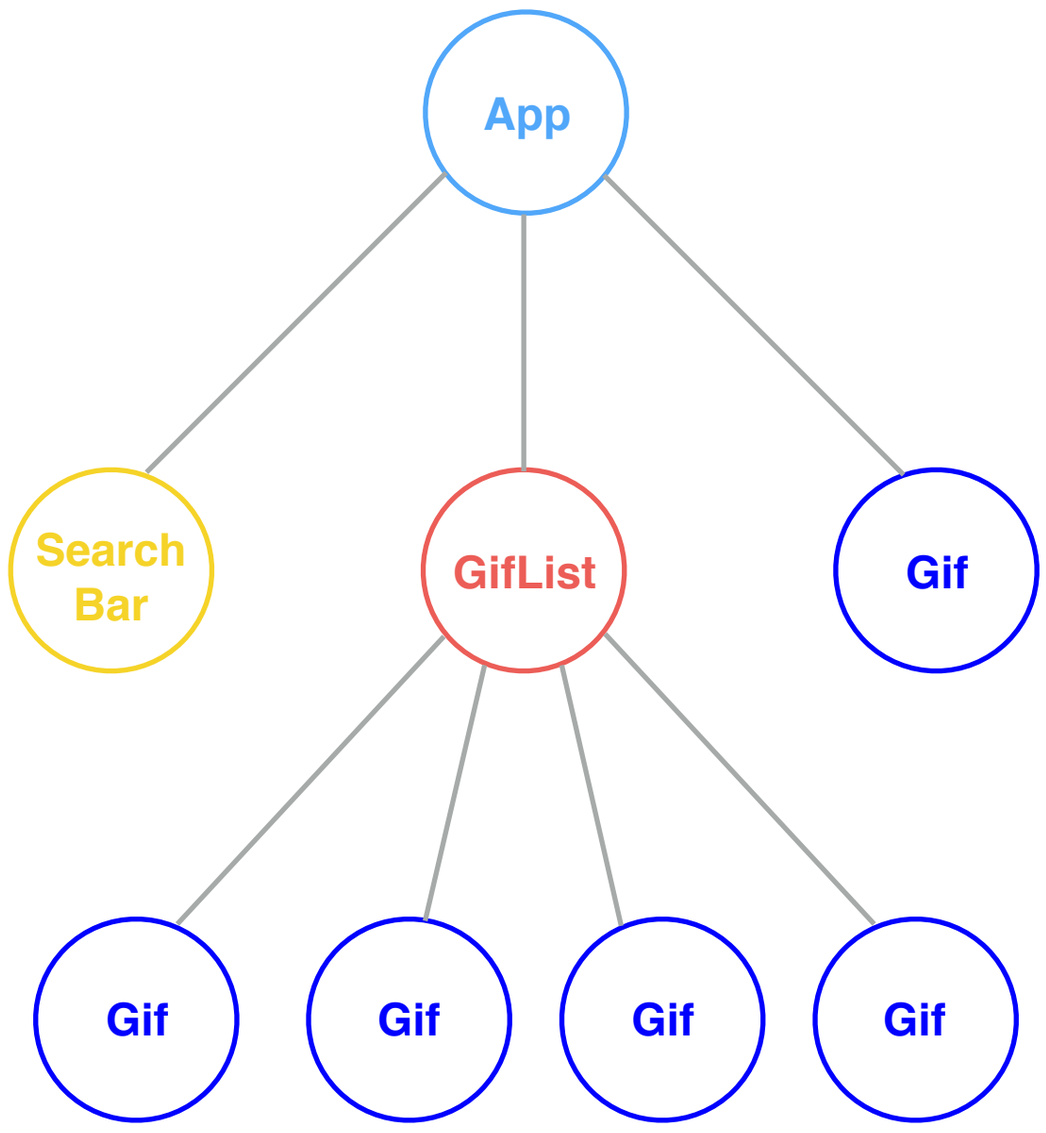
CSS: https://gist.github.com/ssaunier/dbf2b76987ec62258d7ad51f0162a0ed

\+ https://www.bootstrapcdn.com/fontawesome/ in index.html

Components


localhost:8080
SearchBar
homer thinking
GifList
Money can be exchanged for goods and services
I WAS THINKING MORE OF SOMETHING IN A TWO OR THREE.
DICTIONARY
HMM.
"THINKING"
THIS IS IT, HOMER.
MMM.
INTERESTING.
Gif
App

Components



Modeling data

Think **immutable** first. Model data with **`props`**.

A **`Gif`** is characterised by its **`id`** (string)
A **`GifList`** is characterised by a **list of gifs** (array)
`SearchBar` has no props.
Neither does **`App`**.

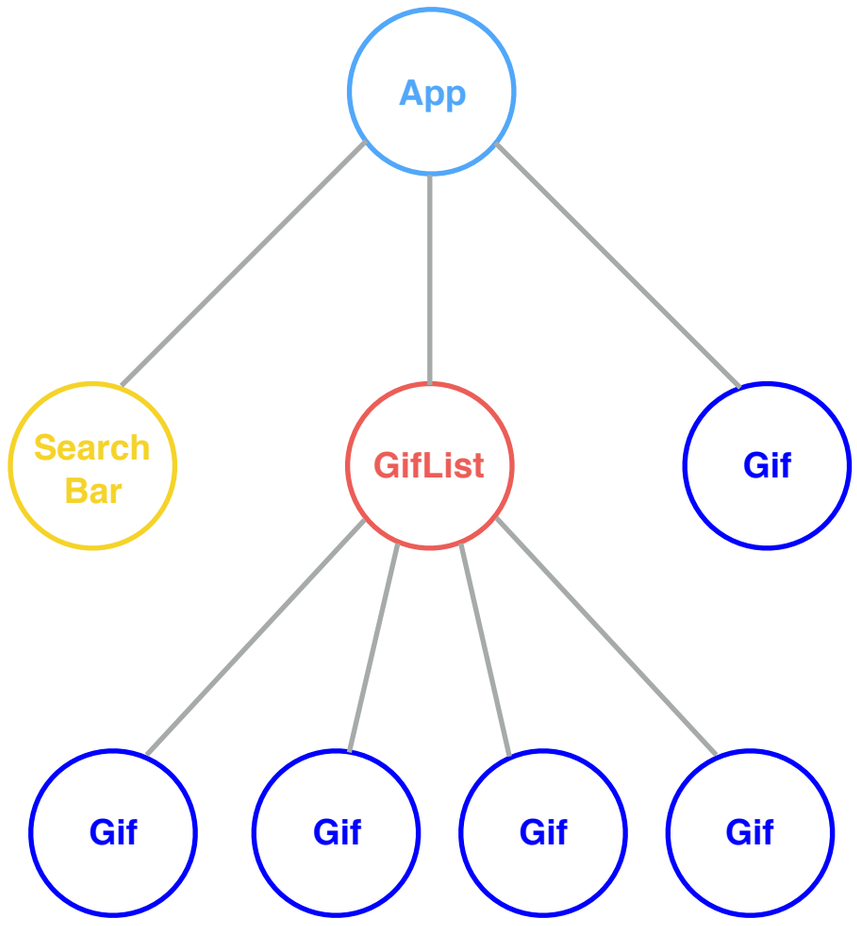
```
// Gif's props:
{ id: 'xT9IgDEI1iZyb2wqo8' }

// GifList's props:
{ gifs: [ ... ] }
```

Modeling data

Then think in terms of **events**.

Data which **needs to change** after an event
must be carried by some component's **`state`**.

Downward data-flow



When a component's state changes, its **`render()`** method is called back.

This triggers the **re-rendering** of **all of its children**

Event n°1

Here, when a user **types a query**, the **list of gifs** should **update**.

The **first parent** component aware of **both the query and the list of gifs** should carry this piece of state.

Here, **App** should carry the **`gifs`** piece of state

Event n°2

When a user **clicks on a gif** from the **list**,
it should be featured on the main scene.

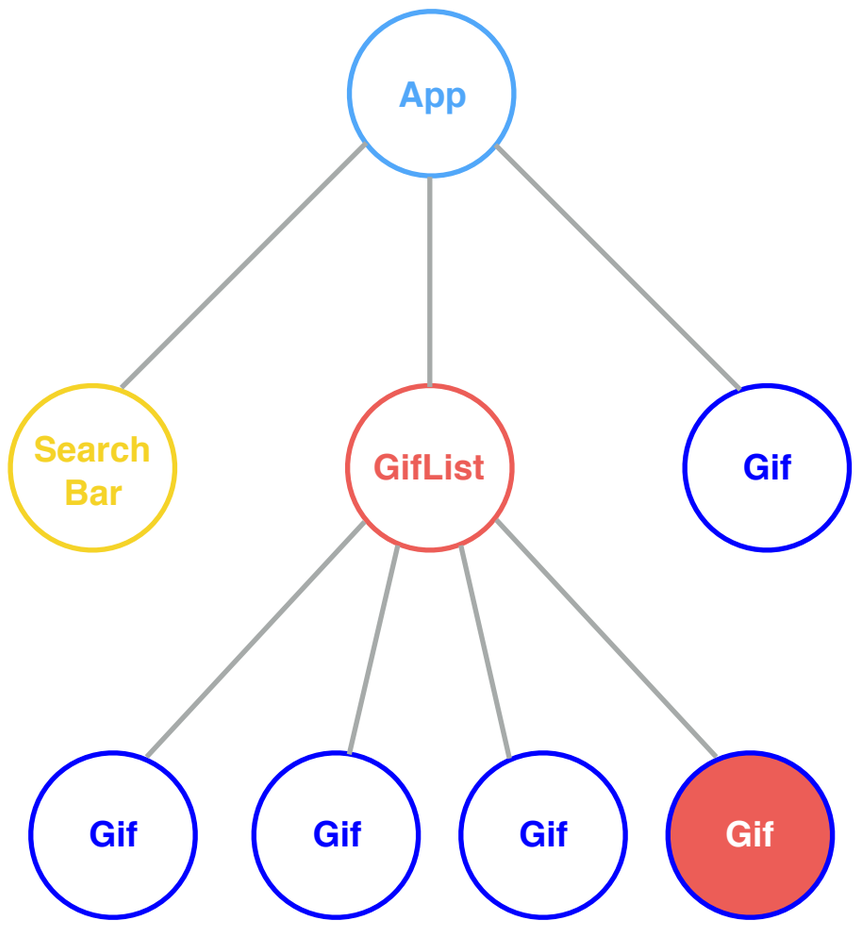
The **first parent** component aware of **both the list of gifs and the selected one** should carry this piece of state.

Here, **App** should carry the **`selectedGif`** piece of state

App's state

```
// App's state:
{
  gifs: [],
  selectedGifId: null
}
```

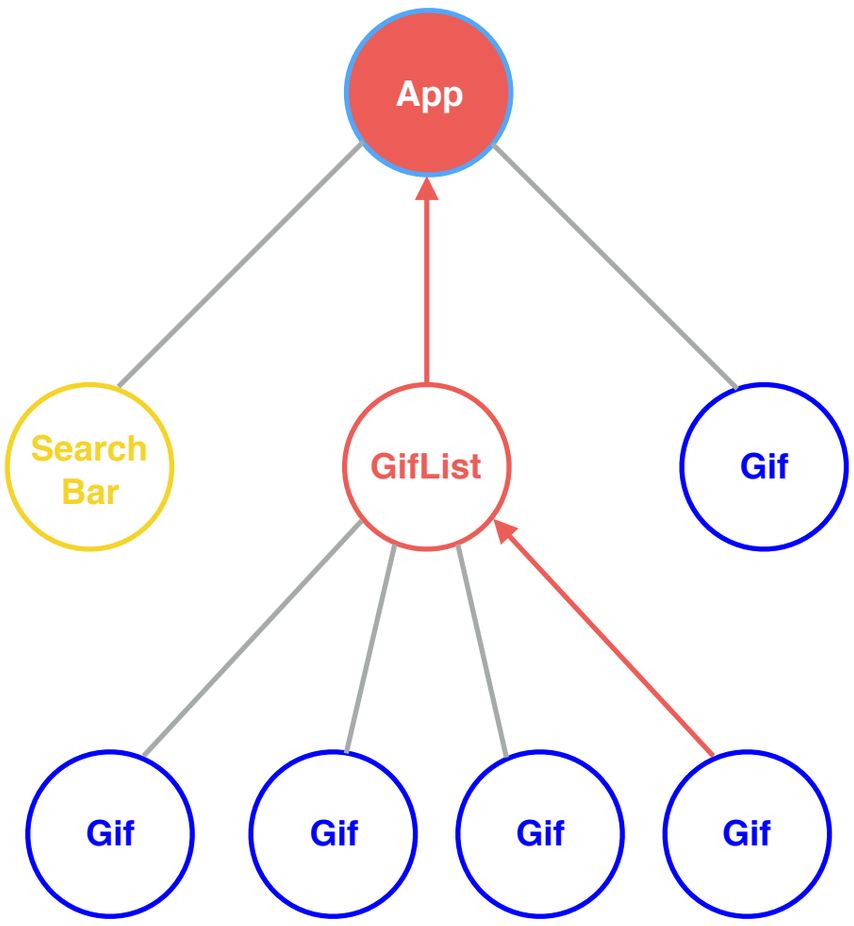
Strategy



we'll handle **events** at the **local component level**

```
onClick={this.props.handleClick}
```

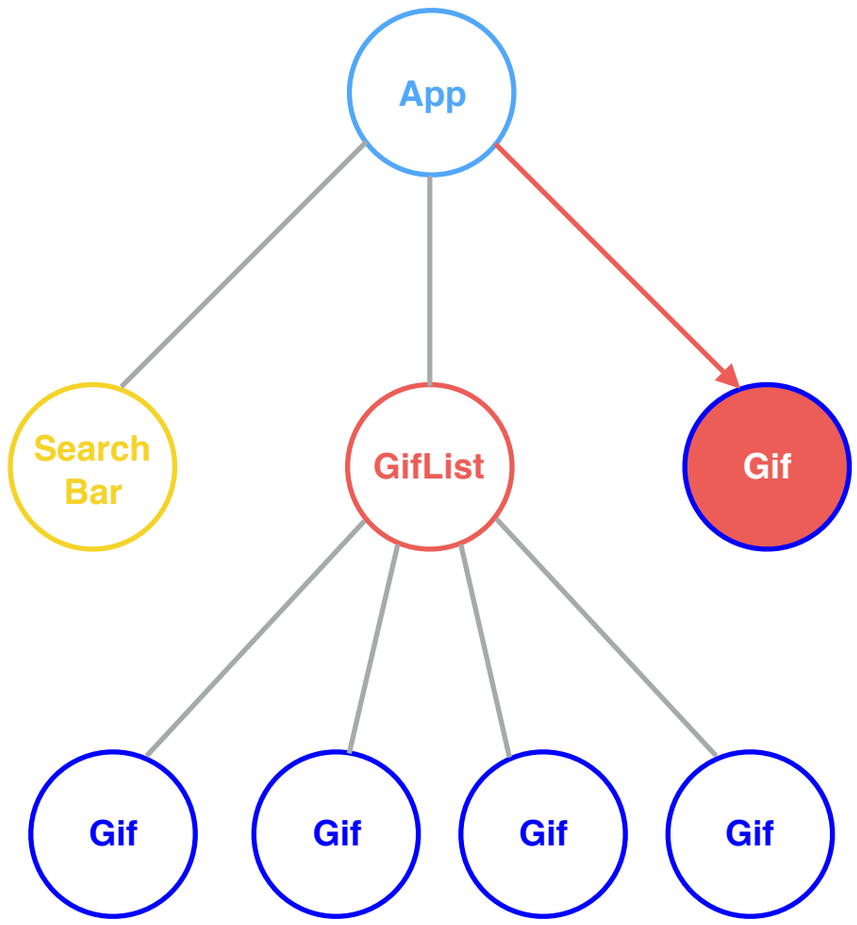
Strategy




```
this.setState({ selectedGifId: …})
```

update the **`state`** at the **first possible parent component level**, triggering its **`render( )`**

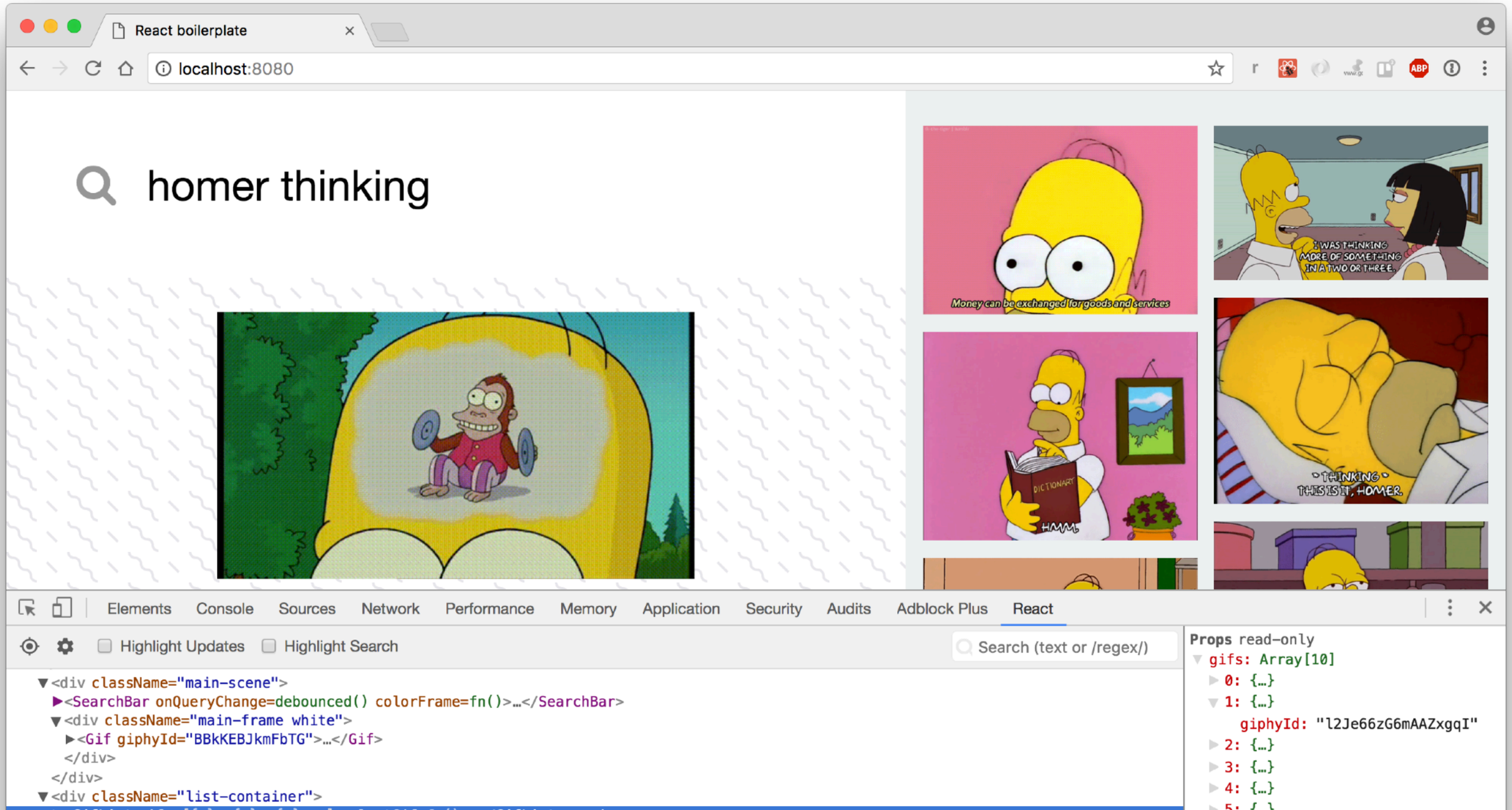
Strategy



flowing new **`props`** down to its children components who **re-render** as well

Tools

Download Chrome's React developer Tools



Your turn!