# Templates and Databinding

SWE 432, Fall 2017

Design and Implementation of Software for the Web

GEORGE MASON UNIVERSITY

# Today

- What are templates?
- What are frontend components?
- How can I use these with React?

# What’s wrong with this code?

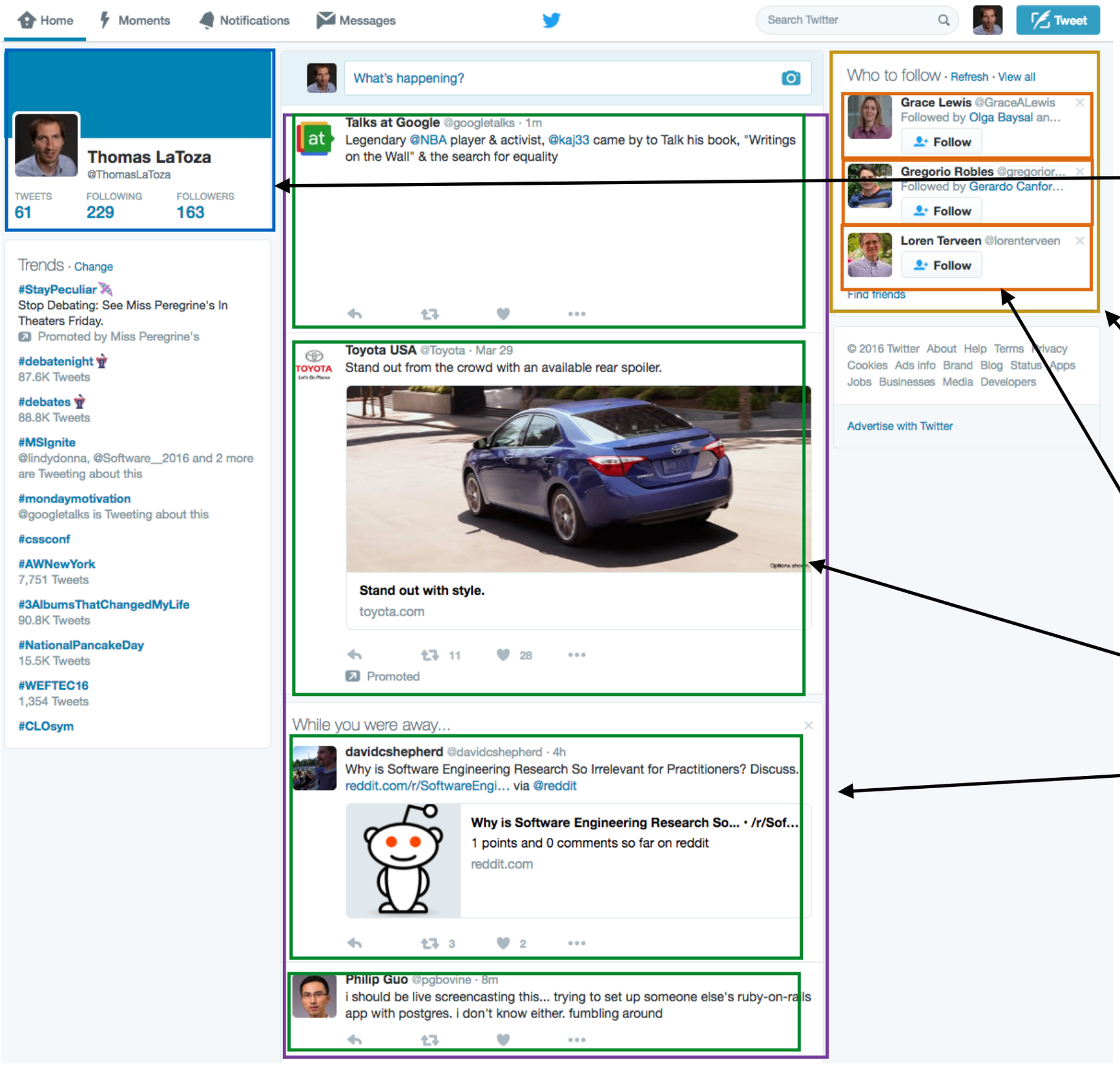
```
function createItem(value, key)
{
    document.getElementById('todoItems').innerHTML +=
            '<div class="todoItem" data-index="' + key
           + '"><input type="text" onchange="itemChanged(this)" value='
   + value + '"><button onclick="deleteItem(this.parentElement)">&#x2716;</button></div>'
        ;
    }
```

# What’s wrong with this code?

```
function createItem(value, key)
{
    document.getElementById('todoItems').innerHTML +=
            '<div class="todoItem" data-index="' + key
           + '"><input type="text" onchange="itemChanged(this)" value="'
   + value + '"><button onclick="deleteItem(this.parentElement)">&#x2716;</button></div>'
        ;
}
```

No syntax checking

# Anatomy of a Non-Trivial Web App




User profile widget

Who to follow widget

Follow widget

Feed widget

Feed item widget

# Typical properties of web app UIs

- Each widget has both visual presentation & logic
  - e.g., clicking on follow button executes some logic related to the containing widget
  - Logic and presentation of individual widget strongly related, loosely related to other widgets
- Some widgets occur more than once
  - e.g., Follow widget occurs multiple times in Who to Follow Widget
  - Need to generate a copy of widget based on data
- Changes to data should cause changes to widget
  - e.g., following person should update UI to show that the person is followed. Should work even if person becomes followed through other UI
- Widgets are hierarchical, with parent and child
  - Seen this already with container elements in HTML…

# Idea 1: Templates

```
 document.getElementById('todoItems').innerHTML +=
         '<div class="todoItem" data-index="' + key
         + '"><input type="text" onchange="itemChanged(this)" value="'
+ value + '"><button onclick="deleteItem(this.parentElement)">&#x2716;</button></div>';
```

- Templates describe repeated HTML through a single common representation
  - May have variables that describe variations in the template
  - May have logic that describes what values are used or when to instantiate template
  - Template may be instantiated by binding variables to values, creating HTML that can be used to update DOM

# Templates with template literals

```
document.getElementById('todoItems').innerHTML +=
        `<div class="todoItem" data-index="${key}">
             <input type="text" onchange="itemChanged(this)" value="${value}">
             <button onclick="deleteItem(this.parentElement)">&#x2716;</button>
         </div>`;
```

- Template literals reduce confusion of nested strings

# Server side vs. client side

- Where should template be instantiated?

- *Server-side* frameworks: Template instantiated on server
  - Examples: JSP, ColdFusion, PHP, ASP.NET
  - Logic executes on server, generating HTML that is served to browser
- *Front-end* framework: Template runs in web browser
  - Examples: React, Angular, Meteor, Ember, Aurelia, …
  - Server passes template to browser, browser generates HTML on demand

```
<html>
<head><title>First JSP</title></head>
<body>
  <%
    double num = Math.random();
    if (num > 0.95) {
  %>
      <h2>You'll have a luck day!</h2><p>(<%= num %>)</p>
  <%
    } else {
  %>
      <h2>Well, life goes on ... </h2><p>(<%= num %>)</p>
  <%
    }
  %>
```

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="John" />,
  mountNode
);
```

# Server side vs. client side

- Server side
  - Oldest solution.
  - True when “real” code ran on server, Javascript
- Client side
  - Enables presentation logic to exist entirely in browser
  - e.g., can make call to remote web service, no need for server to be involved
  - (What we’ve looked at in this course).

# Logic

- Templates require combining logic with HTML
  - Conditionals - only display presentation if some expression is true
  - Loops - repeat this template once for every item in collection
- How should this be expressed?
  - Embed code in HTML (ColdFusion, JSP, Angular)
  - Embed HTML in code (React)

# Embed code in HTML

```
<cfcomponent name = "PersonalChef">
    <cffunction name = "makeToast" returnType = "component">
        <cfargument name = "color" required="yes">

        <cfset this.makeToast = "Making your toast #arguments.color#!" />
        <cfreturn this />
    </cffunction>
</cfcomponent>
```

```
<html>
<head><title>First JSP</title></head>
<body>
  <%
    double num = Math.random();
    if (num > 0.95) {
  %>
      <h2>You'll have a luck day!</h2><p>(<%= num %>)</p>
  <%
    } else {
  %>
      <h2>Well, life goes on ... </h2><p>(<%= num %>)</p>
  <%
    }
  %>
```

- Template takes the form of an HTML file, with extensions
  - Custom tags (e.g., <% %>) enable logic to be embedded in HTML
  - Uses another language (e.g., Java, C) or custom language to express logic
  - Found in frameworks such as PHP, Angular, ColdFusion, ASP, …

# Embed HTML in code

- Template takes the form of an HTML fragment, embedded in a code file
  - HTML instantiated as part of an expression, becomes a value that can be stored to variables
  - Uses another language (e.g., Javascript) to express logic
  - This course: **React**

# Templates enable HTML to be rendered multiple times

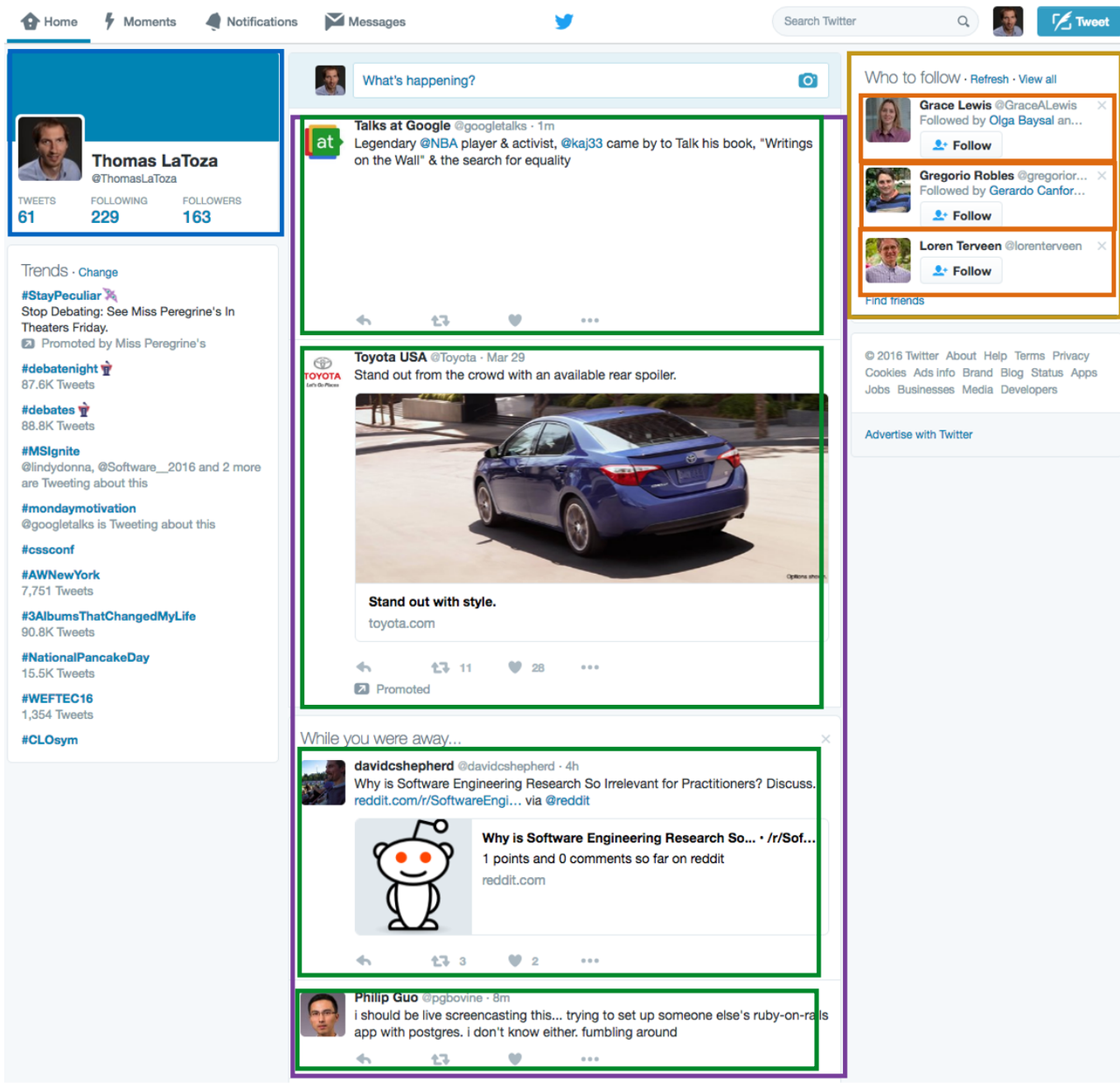
- Rendering takes a template, instantiates the template, outputs HTML

- Logic determines which part(s) of templates are rendered

- Expressions are evaluated to instantiate values
  - e.g., { this.props.name }
  - Different variable values ==> different HTML output

# Idea 2: Components

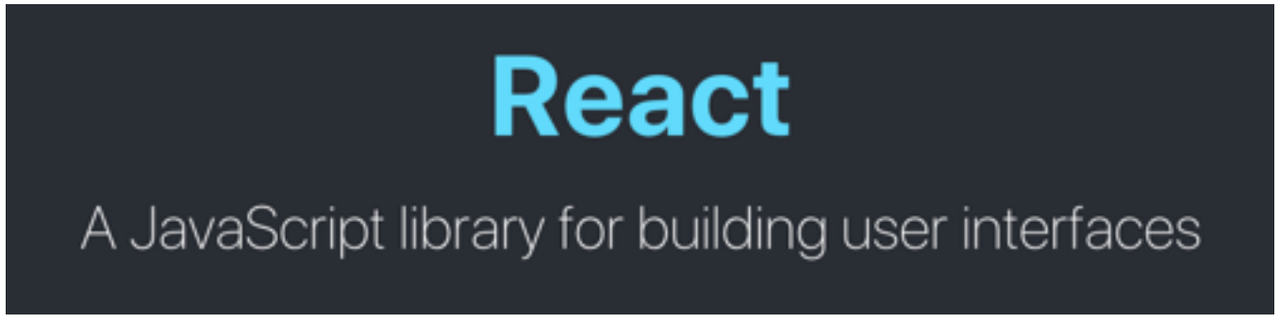
- Web pages are complex, with lots of logic and presentation
- How can we organize web page to maximize modularity?
- Solution: Components
  - Templates that correspond to a specific widget
  - Encapsulates related logic & presentation using language construct (e.g., class)



# Components

- Organize related logic and presentation into a single unit
  - Includes necessary *state* and the logic for updating this state
  - Includes presentation for *rendering* this state into HTML
  - Outside world *must* interact with state through accessors, enabling access to be controlled
- Synchronizes state and visual presentation
  - Whenever state changes, HTML should be rendered again
- Components instantiated through custom HTML tag

# React: Front End Framework for Components




- Originally build by Facebook
- Opensource frontend framework
- Powerful abstractions for describing frontend UI components
- Official documentation & tutorials
  - https://reactjs.org/

# Example

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello world!
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage/>, mountNode
);
```

## "Declare a HelloMessage component"

Declares a new component with the provided functions.

## "Return the following HTML whenever the component is rendered"

Render generates the HTML for the component. The HTML is dynamically generated by the library.

## "Render HelloMessage and insert in mountNode"

Instantiates component, replaces mountNode innerHTML with rendered HTML. Second parameter should always be a DOM element.

# Example - Properties

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="John" />,
  mountNode
);
```

## “Read this.props.name and output the value”

Evaluates the expression to a value.

## “Set the name property of HelloMessage to John”

Components have a this.props collection that contains a set of properties instantiated for each component.

# Embedding HTML in Javascript

```
return <div>Hello {this.props.name}</div>;
```

- HTML embedded in JavaScript
  - HTML can be used as an expression
  - HTML is checked for correct syntax
  - Can use { expr } to evaluate an expression and return a value
    - e.g., { 5 + 2 }, { foo() }
  - Output of expression is HTML

# JSX

- How do you embed HTML in JavaScript and get syntax checking??
- Idea: extend the language: JSX
  - Javascript language, with additional feature that expressions may be HTML
  - Can be used with ES6 or traditional JS (ES5)
- It's a new language
  - Browsers *do not* natively run JSX
  - If you include a JSX file as source, you will get an error

# Transpiling

- Need to take JSX code and trans-compile ("transpile") to Javascript code
  - Take code in one language, output *source* code in a second language
- Where to transpile?
  - Serverside, as part of build process
    - Fastest, least work for client. Only have to execute transpiring once.
  - Clientside, through library.
    - Include library that takes JSX, outputs JS.
    - Easy. Just need to include a script.

# Babel

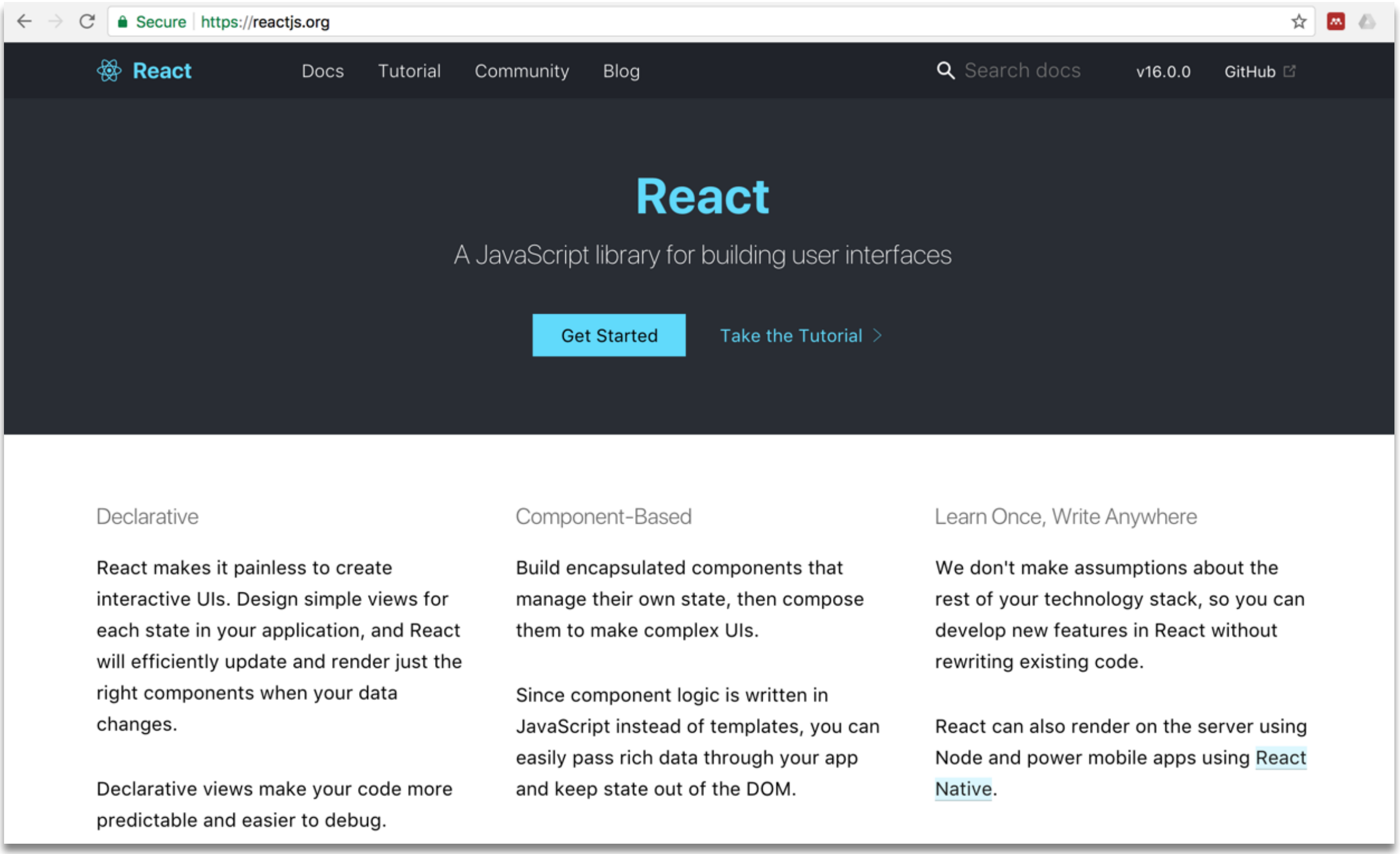
```
<script src="https://cdnjs.com/libraries/babel-core/
5.8.34"> </script>

<script type="text/babel">
//JSX here
</script>
```

**Babel client side**

- Transpiler for Javascript
- Takes JSX (or ES6) and outputs traditional Javascript (a.k.a ES5)
- Can use server side or client side
- Using Babel serverside: https://facebook.github.io/react/docs/language-tooling.html

https://babeljs.io/


Secure https://reactjs.org
React
Docs
Tutorial
Community
Blog
Search docs
v16.0.0
GitHub
React
A JavaScript library for building user interfaces
Get Started
Take the Tutorial
Declarative
React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.
Declarative views make your code more predictable and easier to debug.
Component-Based
Build encapsulated components that manage their own state, then compose them to make complex UIs.
Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.
Learn Once, Write Anywhere
We don't make assumptions about the rest of your technology stack, so you can develop new features in React without rewriting existing code.
React can also render on the server using Node and power mobile apps using React Native.

Secure | https://codepen.io/gaearon/pen/gWWZgR?editors=1010

Tic Tac Toe

A PEN BY Dan Abramov PRO

Fork | Settings | Change View

HTML

```
<div id="errors" style="
  background: #c00;
  color: #fff;
  display: none;
  margin: -20px -20px 20px;
  padding: 20px;
  white-space: pre-wrap;
"></div>
<div id="root"></div>
<script>
window.addEventListener('mousedown',
function(e) {
  document.body.classList.add('mouse-
navigation');
  document.body.classList.remove('kbd-
navigation');
});
window.addEventListener('keydown',
function(e) {
  if (e.keyCode === 9) {
    document.body.classList.add('kbd-
```

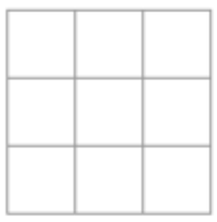
CSS

JS (Babel)

```
function Square(props) {
  return (
    <button className="square" onClick={props.onClick}>
      {props.value}
    </button>
  );
}

class Board extends React.Component {
  renderSquare(i) {
    return (
      <Square
        value={this.props.squares[i]}
        onClick={() => this.props.onClick(i)}
      />
    );
  }

  render() {
    return (
      <div>
```



Next player: X

1. Go to game start

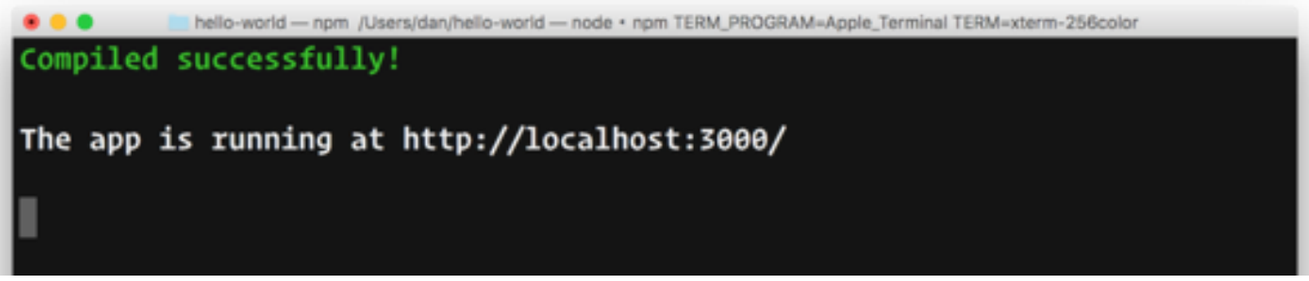
- Pastebin sites such as JSFiddle and codepen.io work with React
- Just need to include React first

# Create React App

```
npm install -g create-react-app

create-react-app my-app
cd my-app/
npm start
```

Then open http://localhost:3000/ to see your app.
When you're ready to deploy to production, create a minified bundle with `npm run build`.




- Sets up Babel as well as other tools.
- Should create an app as a **subfolder** of your express backend.
  - Will now have **two** package.json files: one for your frontend, one for your backend

https://github.com/facebookincubator/create-react-app

# Proxy Server for Dev

- Will use a proxy server to have two separate web servers running, one for front end one for back end
- Add to your **client** package.json

```
    "eject": "react-scripts eject"
    },
    "proxy": "http://localhost:5000"
}
```

- Update **server** port in server.js to 5000

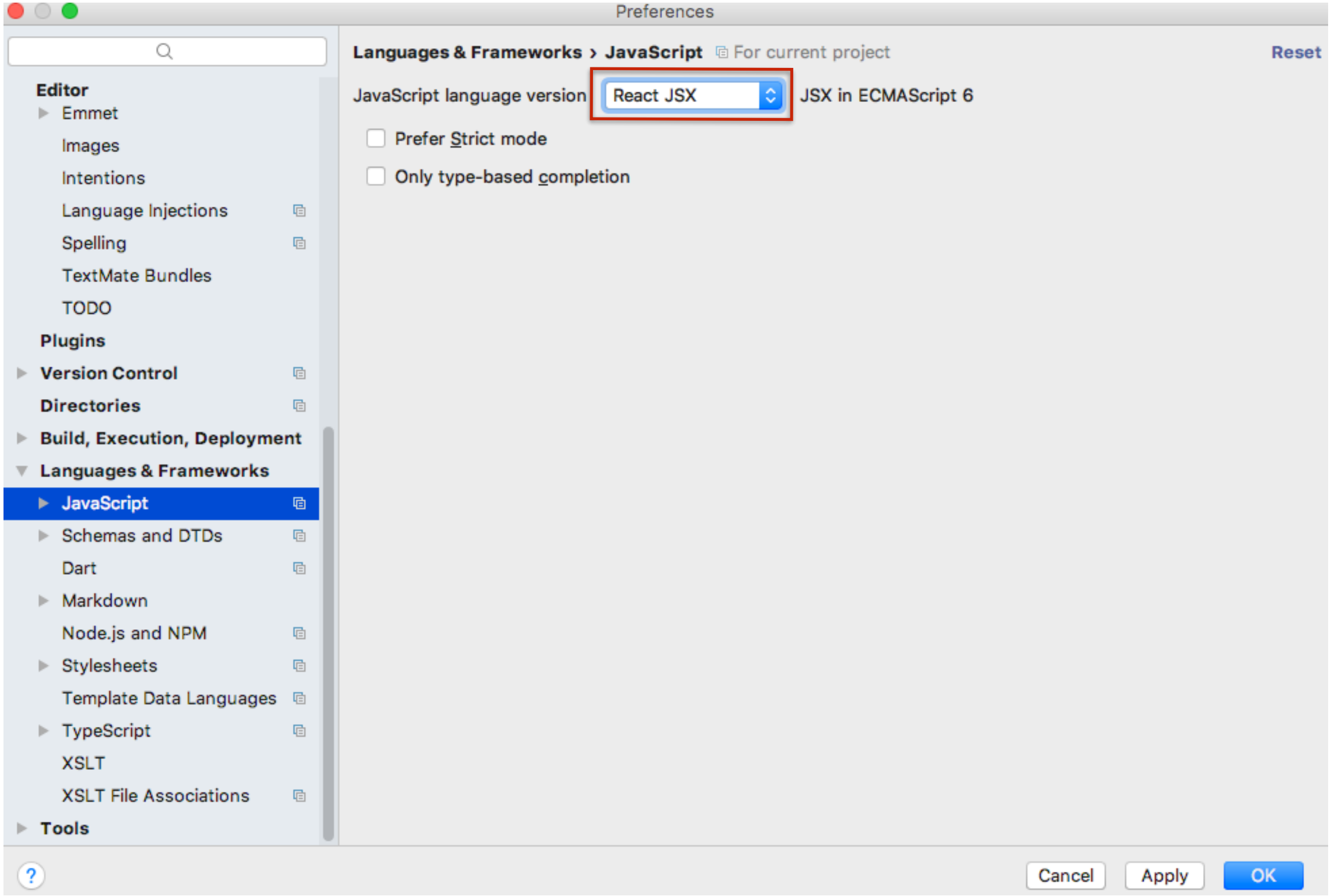
```
app.listen(5000, function () {
    console.log('Example app listening on port 5000!')
});
```

- https://daveceddia.com/create-react-app-express-backend/
- This won't work in deployment, need a different approach (See next slide)

# Deploying to heroku

- Need to add a build command
- https://daveceddia.com/create-react-app-express-production/
  - Start with "Create the React App" section

# Setting Language in Webstorm



# Demo: Create React App

- Example of create react app
- Overview of default files and folder structure

# Demo: Generating lists from fetched data

# Readings for next time

- More on React
  - https://reactjs.org/docs/conditional-rendering.html
  - https://reactjs.org/docs/lists-and-keys.html
  - https://reactjs.org/docs/forms.html