COMPONENT TECHNOLOGIES FOR THE VIEW

OVERVIEW

- Web Components
- React.js

WEB DEVELOPMENT

In many instances you're either copying huge chunks of HTML out of some doc and then pasting that into your app (Bootstrap, Foundation, etc.), or you're sprinkling the page with jQuery plugins that have to be configured using JavaScript. It puts us in the rather unfortunate position of having to choose between bloated HTML or mysterious HTML, and often we choose ***both****.*

A Guide to Web Components

BETTER WEB DEVELOPMENT

HTML would be ...

- ... *expressive* enough to create complex UI widgets
- ... *extensible* so that we could fill in any gaps with our own tags

This is finally possible Web Components

WEB COMPONENTS

- Bundle markup and styles into custom HTML elements
- Fully encapsulate all of their HTML and CSS
- Introduced by Alex Russell at Fronteers Conference 2011

EXAMPLE: IMAGE SLIDER

```
<div id="slider">
  <input checked="" type="radio" name="slider" id="slide1" selected="false">
  <input type="radio" name="slider" id="slide2" selected="false"> ...
  <div id="slides">
    <div id="overflow">
      <div class="inner">
        <img src="images//rock.jpg">
        <img src="images/grooves.jpg">...
      </div>
    </div>
  </div>
  <label for="slide1"></label>
  <label for="slide2"></label>...
</div>
```

codepen.io/robdodson/pen/rCGvJ

EXAMPLE: BETTER IMAGE SLIDER

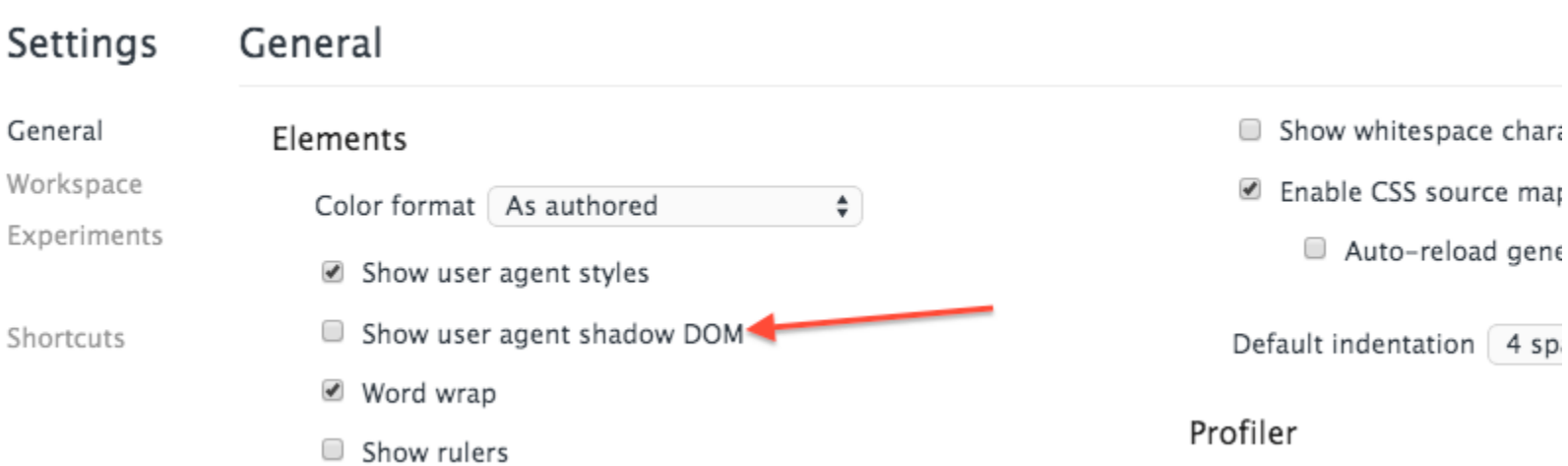
```
<img-slider>
  <img src="images/sunset.jpg" alt="a dramatic sunset">
  <img src="images/arch.jpg" alt="a rock arch">
  <img src="images/grooves.jpg" alt="some neat grooves">
  <img src="images/rock.jpg" alt="an interesting rock">
</img-slider>
```

THE VIDEO ELEMENT

```
<video src="./foo.webm" controls></video>
```

- There’s a play button, a scrubber, timecodes, a volume slider
- A way to build the *video* element from these parts was needed
- Browser makers created a secret place: the **Shadow DOM**

SHADOW DOM


Settings
General
General
Workspace
Experiments
Shortcuts
Elements
Color format
As authored
Show user agent styles
Show user agent shadow DOM
Word wrap
Show rulers
Show whitespace chara
Enable CSS source map
Auto-reload gene
Default indentation
4 sp
Profiler

THE VIDEO ELEMENT

nts | Network | Sources | Timeline | Profiles | Resources | Audits | Console

```
<video id="video" controls preload="none" poster="http://media.w3.org/2010/05/sintel/
poster.png">
  #shadow-root (user-agent)
    <div>
      <div>
        <div>
          <input type="button">
          <input type="range" step="any" max="0">
           <div style="display: none;">0:00</div>
           <div>0:00</div>
          <input type="button">
          <input type="range" step="any" max="1" style="display: none;">
          <input type="button" style="display: none;">
          <input type="button" style="display: none;">
```

TEMPLATES

- New `template` element
- Not rendered on the page until it is activated using JavaScript

```
<template>
  <h1>Hello there!</h1>
  <p>This content is top secret :)</p>
</template>
```

EXAMPLE: IMAGE SLIDER

Put all of its HTML and CSS into a `template`

```
<template>
  <style>...</style>
  <div id="slider">
    <input checked="" type="radio" name="slider" id="slide1" selected="false">
    ...
  </div>
</template>
```

SHADOW DOM

Select an element and call its *createShadowRoot* method

```
<!-- HTML -->
<div class="container"></div>
```

```
// JavaScript
var host = document.querySelector('.container');
var root = host.createShadowRoot();
root.innerHTML = '<p>How <em>you</em> doin?</p>'
```

SHADOW HOST

- Element that *createShadowRoot* is called on
- The only piece visible to the user
- The place where the element is supplied with content
- Example: the *video* element is the shadow host

```
<video>
  <source src="trailer.mp4" type="video/mp4">
  <source src="trailer.webm" type="video/webm">
  <source src="trailer.ogv" type="video/ogg">
</video>
```

SHADOW ROOT

- Document fragment returned by *createShadowRoot*
- It and its descendants are hidden from the user
- But they’re what the browser will actually render

SHADOW BOUNDARY

- Separates CSS in the parent document from the shadow DOM
- Separates JS in the parent document from the shadow DOM

EXAMPLE: IMAGE SLIDER

```
<template>
  <!-- Full of slider awesomeness -->
</template>

<div class="img-slider"></div>
```

```
// Add the template to the Shadow DOM
var tmpl = document.querySelector('template');
var host = document.querySelector('.img-slider');
var root = host.createShadowRoot();
root.appendChild(document.importNode(tmpl.content, true));
```

codepen.io/robdodson/pen/GusaF (Chrome)

EXAMPLE: IMAGE SLIDER

- Open problem: Image paths are hard coded in the template
- To pull items into the shadow DOM use the *content* tag
- It projects elements from the shadow host into the shadow DOM
- These projections are known as *insertion points*

```
<template>
  ...
  <div class="inner">
    <content select="img"></content>
  </div>
</template>
```

EXAMPLE: IMAGE SLIDER

```
<div class="img-slider">
  <img src="images/rock.jpg" alt="an interesting rock">
  <img src="images/grooves.jpg" alt="some neat grooves">
  <img src="images/arch.jpg" alt="a rock arch">
  <img src="images/sunset.jpg" alt="a dramatic sunset">
</div>
```

SHADOW DOM CSS

New pseudo classes and elements for the Shadow DOM

- `::shadow`
  Selects shadow trees inside of an element
- `:host`
  Selects a shadow host element
- `:host-context`
  Shadow host based on a matching parent element
- `::content`
  Selects distributed nodes inside of an element

robdodson.me/shadow-dom-css-cheat-sheet/

CUSTOM ELEMENT

- Its name must contain a hyphen
- Its prototype must extend *HTMLElement*
- The method *createdCallback* creates the Shadow DOM
- Register the new element with *document.registerElement*

EXAMPLE: IMAGE SLIDER

```
// Grab our template full of slider markup and styles
var tmpl = document.querySelector('template');

// Create a prototype for a new element that extends HTMLElement
var ImgSliderProto = Object.create(HTMLElement.prototype);

// Setup our Shadow DOM and clone the template
ImgSliderProto.createdCallback = function() {
  var root = this.createShadowRoot();
  root.appendChild(document.importNode(tmpl.content, true));
};

// Register our new element
var ImgSlider = document.registerElement('img-slider', {
  prototype: ImgSliderProto
});
```

BROWSER SUPPORT

- Web Comonents were introduced in 2011
- By now, 4 years on, Web Components should be everywhere
- In reality only Chrome has *some version* of Web Components
  caniuse.com/#search=Web%20Components
- Firefox: set *dom.webcomponents.enabled* to true for some support
- Reason: vendors couldn’t agree
- Web Components were a Google effort

WEB COMPONENT POLYFILLS

- Polymer: Google's Web Component library
  www.polymer-project.org
- X-Tag: Mozilla's alternative
  www.x-tags.org

THINK ABOUT / DISCUSS

Will Web Components be the future of Web application development?

REACT.JS

REACT.JS POPULARITY

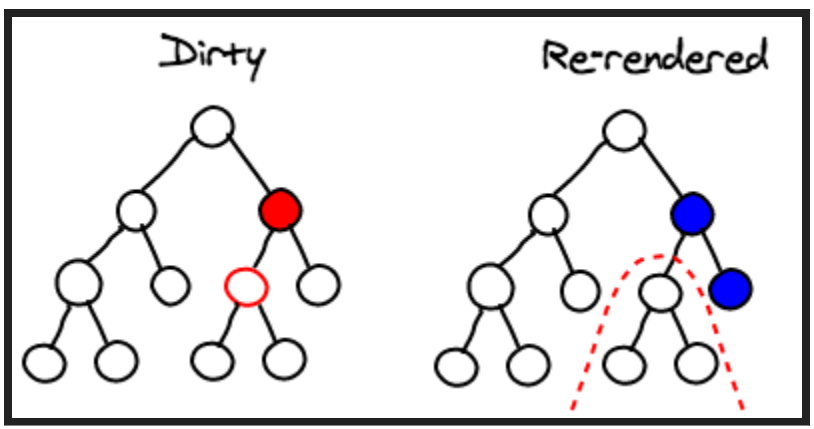
WHAT IS IT?

- React is the "View" in the application
- It is not a framework, it is mainly a concept
- Helps you organize your templates in components
- Virtual DOM makes DOM understandable and controllable
- Virtual DOM makes the rendering fast

Short: We have components and fast rendering

VIRTUAL DOM

- Data changes tracked using an observer model
- React builds a tree representation of the DOM in memory
- It calculates which DOM element should change using a diffing algorithm



VIRTUAL DOM

- Whole application can be re-rendered when data changes
- Managing applications state much simpler
- Alternative: Many small updates using direct DOM manipulation (jQuery ...)

RENDER ON THE SERVER

- Fake DOM representation in memory
- Can also be rendered on the server

COMPONENT-DRIVEN DEVELOPMENT

- Power of thinking in smaller pieces
- Work with less responsibility
- Makes things easier to understand, to maintain and to test
- Design your components to be responsible for only one thing

↓ example ↓


Search...
Only show products in stock
Name
Price
Sporting Goods
Football
$49.99
Baseball
$9.99
Basketball
$29.99
Electronics
iPod Touch
$99.99
iPhone 5
$399.99
Nexus 7
$199.99

FIRST EXAMPLE

```
// ES5
var HelloComponent = React.createClass({
  render: function() {
    return <div>Hello {this.props.name}</div>;
  }
});

// ES6
class HelloComponent extends React.Component {
  render() {
    return <div>Hello {this.props.name}</div>;
  }
}
```

↓ ?? ↓

Something is strange here, isn’t it?

JAVASCRIPT AND JSX

- Component is a mix of JS and HTML code
- Idea: Everything belonging to the component is in one place
- Pure JS notation:

```
render () {
  return React.createElement("div", null, "Hello ",
    this.props.name);
}
```

JAVASCRIPT AND JSX

- JSX is a XML-like syntax extension to ECMAScript
- JSX and HTML syntax are similar but with some differences
- Example: HTML class attribute is called *className* in JSX
- Has to be compiled to JS

https://babeljs.io/repl/

JSX VS. TEMPLATES

- Difficult to create complex UIs with template languages
- Separation of concerns – where to separate?
- Markup and display logic both share the same concern
- Bundled in components with JSX
- Techniques for reuse code and functional concepts (map, filter) are available

PROPERTIES IN A COMPONENT HIERARCHY

```
class UserName extends React.Component {
  render() {
    return <div>name: {this.props.name}</div>;
  }
}
class User extends React.Component {
  render() {
    return <div>
        <h1>City: {this.props.user.city}</h1>
        <UserName name={this.props.user.name} />
      </div>;
  }
}

var user = { name: 'John', city: 'San Francisco' };
React.render(<User user={user} />, mountNode);
```

A SIMPLE EXAMPLE: COMMENTS

Source: Tutorial from React Website
facebook.github.io/react/docs/tutorial.html

- To start, we put everything in one file
- jQuery ist optional for now

EXAMPLE: HTML

```
<!-- index.html -->
<!DOCTYPE html>
<html>
  <head>
    <title>Hello React</title>
    <script src="https://cdnjs.cloudflare.com/.../react.js"></script>
    <script src="https://cdnjs.cloudflare.com/.../JSXTransformer.js"></script>
    <script src="https://cdnjs.cloudflare.com/.../jquery.min.js"></script>
  </head>
  <body>
    <div id="content"></div>
    <script type="text/jsx">
      // Your code here
    </script>
  </body>
</html>
```

COMPONENT STRUCTURE

- CommentBox
  - CommentList
    - Comment
  - CommentForm

STATIC COMMENT BOX

```
var CommentBox = React.createClass({
  render: function() {
    return (
      <div className="commentBox">
        Hello, world! I am a CommentBox.
      </div>
    );
  }
});
React.render(
  <CommentBox />,
  document.getElementById('content')
);
```

STATIC COMMENT BOX

- *React.createClass()* creates a new React component
- Method render() returns a tree of React components
- The tags are React *div* components, not DOM nodes
- The root component is passed to *React.render()*

ADD FURTHER COMPONENTS

```
var CommentList = React.createClass({
  render: function() {
    return (
      <div className="commentList">
        Hello, world! I am a CommentList.
      </div>
    );
  }
});

var CommentForm = React.createClass({
  render: function() {
    return (
      <div className="commentForm">
        Hello, world! I am a CommentForm.
      </div>
    );
  }
});
```

USE NEW COMPONENTS IN COMMENTBOX

```
var CommentBox = React.createClass({
  render: function() {
    return (
      <div className="commentBox">
        <h1>Comments</h1>
        <CommentList />
        <CommentForm />
      </div>
    );
  }
});
```

MAKE COMMENT COMPONENT DYNAMIC

```
var Comment = React.createClass({
  render: function() {
    return (
      <div className="comment">
        <h2 className="commentAuthor">
          {this.props.author}
        </h2>
        {this.props.children}
      </div>
    );
  }
});
```

MAKE COMMENT COMPONENT DYNAMIC

- Expressions in braces are evaluated in JSX
- Properties passed from the parent component are available through `this.props`
- Any nested elements are passed as `this.props.children`

PASS DATA TO SUB-COMPONENT

```
var CommentList = React.createClass({
  render: function() {
    return (
      <div className="commentList">
        <Comment author="Pete Hunt">This is one comment</Comment>
        <Comment author="Jordan Walke">This is *another*
          comment</Comment>
      </div>
    );
  }
});
```

- Properties can be passed via attributes
- Nested elements are also passed (two text nodes in the example)

USING MARKDOWN

- Markdown formatting is possible using the marked library
- Conversion:

```
var rawMarkup = marked(this.props.children.toString(), {sanitize: true});
```

- The generated HTML code can be inserted in the component:

```
<span dangerouslySetInnerHTML={{__html: rawMarkup}} />
```

- Important: Make sure that no dangerous code (e.g., script) can be inserted

JSON DATA

Comments are now dynamically generated from JSON data

```
var data = [
  {author: "Pete Hunt", text: "This is one comment"},
  {author: "Jordan Walke", text: "This is *another* comment"}
];

React.render(
  <CommentBox data={data} />,
  document.getElementById('content')
);

// In CommentBox:
<CommentList data={this.props.data} />
```

JSON DATA

```
var CommentList = React.createClass({
  render: function() {
    var commentNodes = this.props.data.map(function (comment) {
      return (
        <Comment author={comment.author}>
          {comment.text}
        </Comment>
      );
    });
    return (
      <div className="commentList">
        {commentNodes}
      </div>
    );
  }
});
```

FETCHING DATA FROM THE SERVER

```
React.render(
  <CommentBox url="comments.json" />,
  document.getElementById('content')
);
```

FETCHING DATA FROM THE SERVER

- Fetching data is delegated to the *CommentBox* component
- CommentBox should first be displayed with an empty comment list
- When data from the server arrives, the comment list is updated
- Consequently, we need mutable state to implement this
- Problem: *this.props* is immutable
- Solution: use *this.state*

MUTABLE STATE

```
var CommentBox = React.createClass({
  getInitialState: function() {
    return {data: []};
  },
  render: function() {
    return (
      <div className="commentBox">
        <h1>Comments</h1>
        <CommentList data={this.state.data} />
        <CommentForm />
      </div>
    );
  }
});
```

MUTABLE STATE

- The method *getInitialState* executes once during the lifecycle of the component
- Another method *componentDidMount* initiates the Ajax request

MUTABLE STATE

```
var CommentBox = React.createClass({
  // ...
  componentDidMount: function() {
    $.ajax({
      url: this.props.url,
      dataType: 'json',
      cache: false,
      success: function(data) {
        this.setState({data: data});
      }.bind(this),
      error: function(xhr, status, err) {
        console.error(this.props.url, status, err.toString());
      }.bind(this)
    });
  },
  //...
});
```

CHANGING STATE

- Using *this.setState()* causes re-rendering to take place
- This implementation loads the comments once after displaying the comment box
- Easy to replace this with a polling updater (or e.g., Websockets)

(cf. facebook.github.io/react/docs/tutorial.html)

ADDING NEW COMMENTS

Adding the form is straightforward:

```
var CommentForm = React.createClass({
  render: function() {
    return (
      <form className="commentForm" onSubmit={this.handleSubmit}>
        <input type="text" placeholder="Your name" ref="author" />
        <input type="text" placeholder="Say something..." ref="text" />
        <input type="submit" value="Post" />
      </form>
    );
  }});
```

ADDING NEW COMMENTS

- The *onSubmit* handler calls *handleSubmit*
- This method has yet to be implemented:

```
handleSubmit: function(e) {
   e.preventDefault();
   var author = React.findDOMNode(this.refs.author).value.trim();
   var text = React.findDOMNode(this.refs.text).value.trim();
   if (!text || !author) {
     return;
   }
   // TODO: send request to the server
   React.findDOMNode(this.refs.author).value = '';
   React.findDOMNode(this.refs.text).value = '';
   return;
 }
```

ADDING NEW COMMENTS

- The *ref* attribute is used to assign a name to a child component
- This component is then referenced with *this.refs*
- *React.findDOMNode(component)* returns the native browser DOM element

ADDING NEW COMMENTS

- To do: send request to the server and update comment list
- *CommentBox* is extended by a *handleCommentSubmit* method
- The reference to the method is handed over to the *CommentForm* component:

```
<CommentForm onCommentSubmit={this.handleCommentSubmit} />
```

- In the *CommentForm handleSubmit* method:

```
this.props.onCommentSubmit({author: author, text: text});
```

- This way the author and text data is passed up to the *CommentBox*

ADDING NEW COMMENTS

```
var CommentBox = React.createClass({
  // ...
  handleCommentSubmit: function(comment) {
    $.ajax({
      url: this.props.url,
      dataType: 'json',
      type: 'POST',
      data: comment,
      success: function(data) {
        this.setState({data: data});
      }.bind(this),
      error: function(xhr, status, err) {
        console.error(this.props.url, status, err.toString());
      }.bind(this)
    });
  }
  // ...
});
```

OPTIMIZATION: OPTIMISTIC UPDATES

- Instead of waiting for the Ajax call to return, comment list can be updated immediately

```
handleCommentSubmit: function(comment) {
  var comments = this.state.data;
  var newComments = comments.concat([comment]);
  this.setState({data: newComments});
  $.ajax(
    // ...
  );
  // ...
}
```

SUMMARY: MAIN CONCEPTS

- Clear and simple flow of data: data is passed down and events flow up
- Data is passed down using properties
- While properties should never be changed, state is mutable
- State is owned by a component and can be passed down using properties
- Also to be passed down: Functions to change the state
- It’s best to keep most of your components stateless

SERVER SIDE RENDERING

- You could render the top-level app component with the initial state on the server
- Browser gets HTML-code and can render the complete page immediately

Example:

- jlongster.com/s/bloop/app3/
- Konsole:

```
Bloop.renderComponentToString(Toolbar({ username: 'foo' }))
```

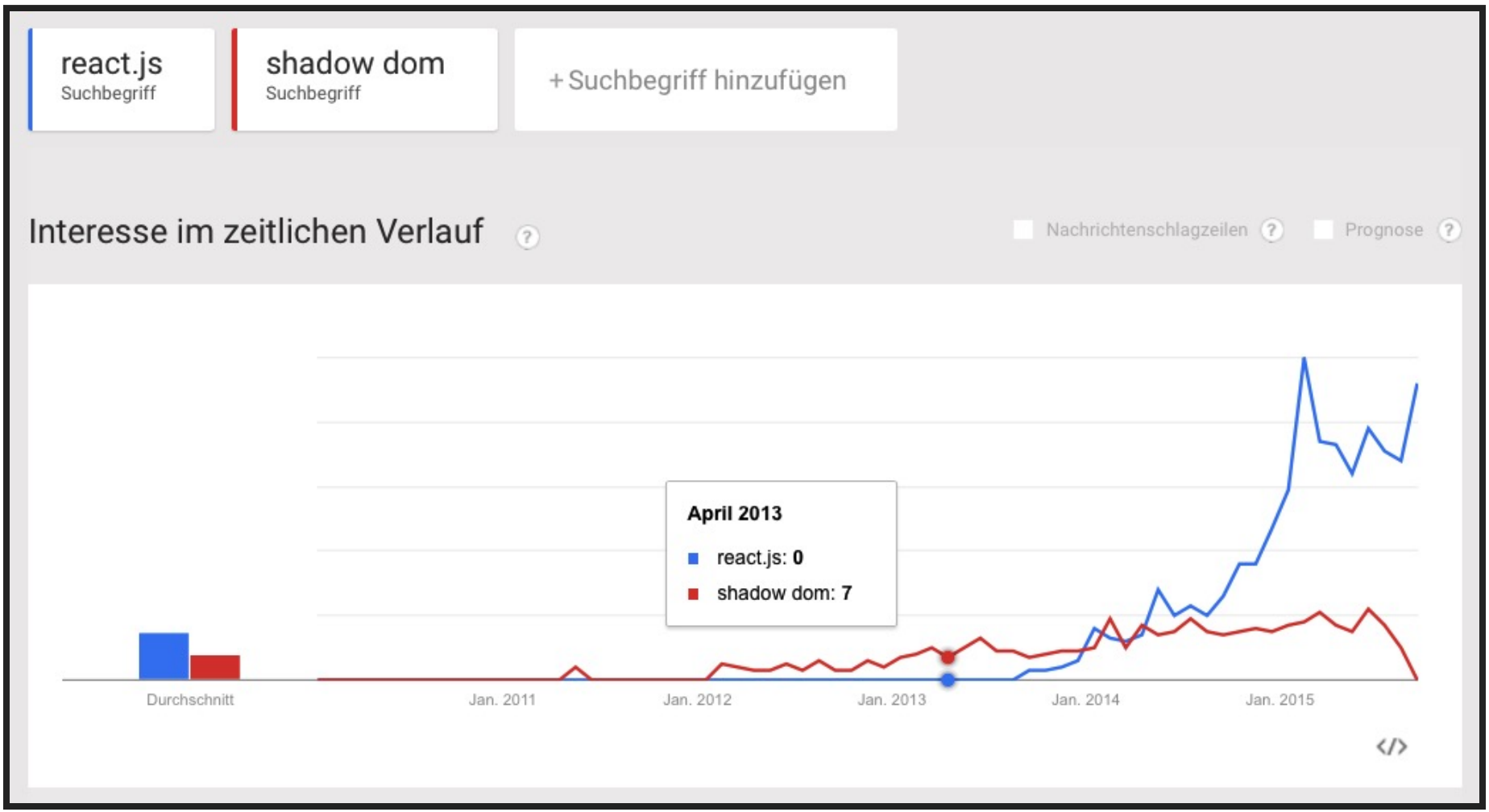
RENDERING WHAT IS VISIBLE

- Example: jlongster.com/s/bloop/app4/
- Code gist.github.com/jlongster/3f32b2c7dce588f24c92#file-e-optimized-list-js

THINK ABOUT / DISCUSS

- What are the key differences between Web Components and React.js?
- Which one – if any – will be the future of Web application development?

GOOGLE TRENDS



READING MATERIAL, SOURCES

READING MATERIAL

- CSS-Tricks: A Guide to Web Components
  css-tricks.com/modular-future-web-components/
- The React.js Way: Getting Started Tutorial
  blog.risingstack.com/the-react-way-getting-started-tutorial/

FURTHER READING

- The state of Web Components (Mozilla) hacks.mozilla.org/2015/06/the-state-of-web-components/
- The Extensible Web Manifesto extensiblewebmanifesto.org

WEB COMPONENTS SPECIFICATIONS

- HTML5 template element
  www.w3.org/TR/html5/scripting-1.html#the-template-element
- Shadow DOM
  w3c.github.io/webcomponents/spec/shadow/
- HTML Imports
  w3c.github.io/webcomponents/spec/imports/
- CSS Scoping Module Level 1
  drafts.csswg.org/css-scoping/
- Custom Elements
  w3c.github.io/webcomponents/spec/custom/

SOURCES

- CSS-Tricks: A Guide to Web Components
  css-tricks.com/modular-future-web-components/
- The React.js Way: Getting Started Tutorial
  blog.risingstack.com/the-react-way-getting-started-tutorial/
- The Secrets of React's Virtual DOM
  fluentconf.com/fluent2014/public/schedule/detail/32395
- JSX Compiler
  facebook.github.io/react/jsx-compiler.html

↓ more ↓

- OSCON 2014: How Instagram.com Works; Pete Hunt www.youtube.com/watch?v=VkTCL6Nqm6Y
- Removing User Interface Complexity, or Why React is Awesome jlongster.com/Removing-User-Interface-Complexity,-or-Why-React-is-Awesome