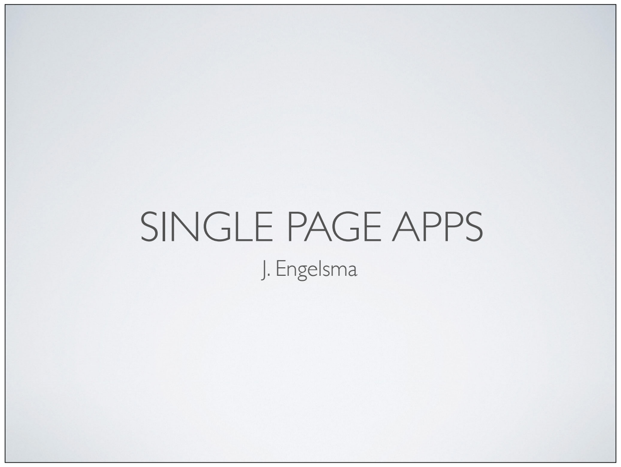
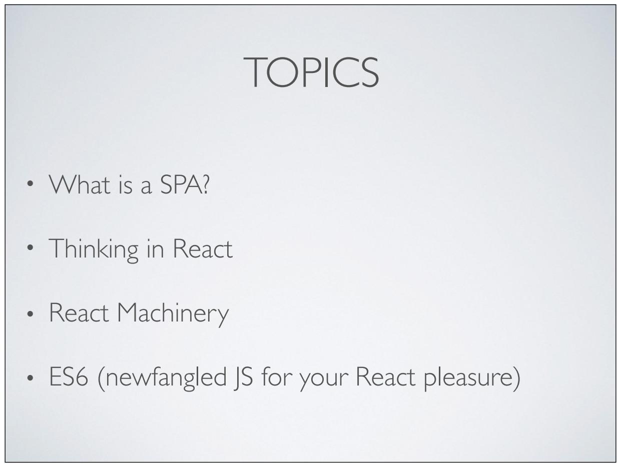


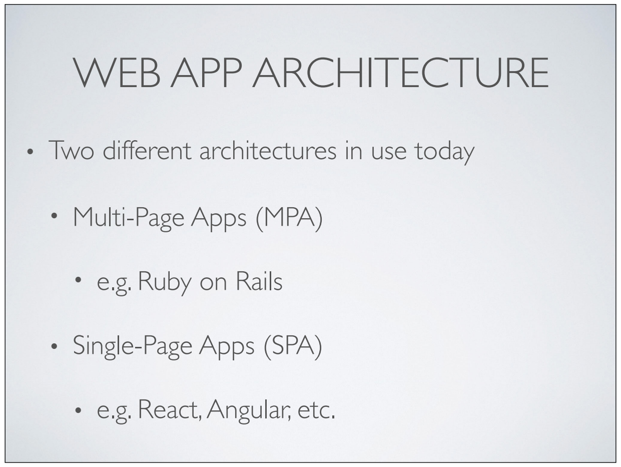
SINGLE PAGE APPS

J. Engelsma



TOPICS

- What is a SPA?
- Thinking in React
- React Machinery
- ES6 (newfangled JS for your React pleasure)



WEB APP ARCHITECTURE

- Two different architectures in use today
  - Multi-Page Apps (MPA)
    - e.g. Ruby on Rails
  - Single-Page Apps (SPA)
    - e.g. React, Angular, etc.

WHAT IS AN MPA

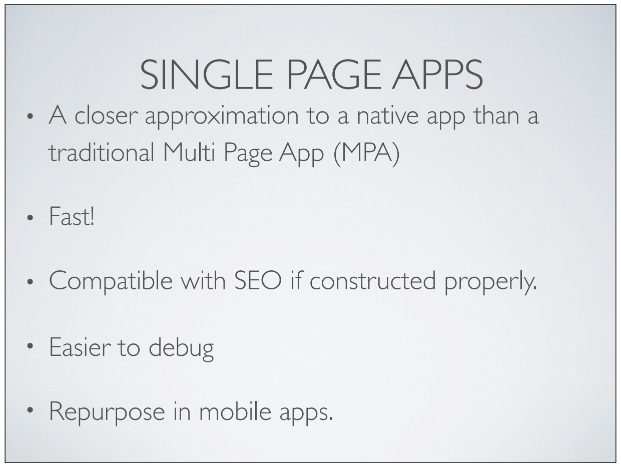
- Each view (e.g. different screen layout) is represented as a single HTML page that is fetched from the web server.
- Views are rendered (often via templates) by code running on the server.
- HTTP fetches involve a mix of HTML and data, though purely AJAX calls are often done by MPAs within a view.

MPA STACKS

- Examples (often use a MVC pattern)
  - Ruby on Rails
  - Django
  - Spring

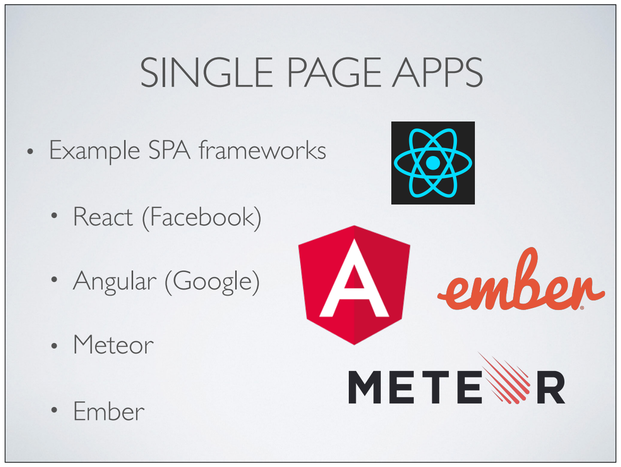
WHAT IS AN SPA?

- SPAs:
  - Load a single HTML page and dynamically update that page as the user interacts.
  - ALL of the user interface is done on the client in JavaScript.
  - HTTP subsequent requests are all data related.



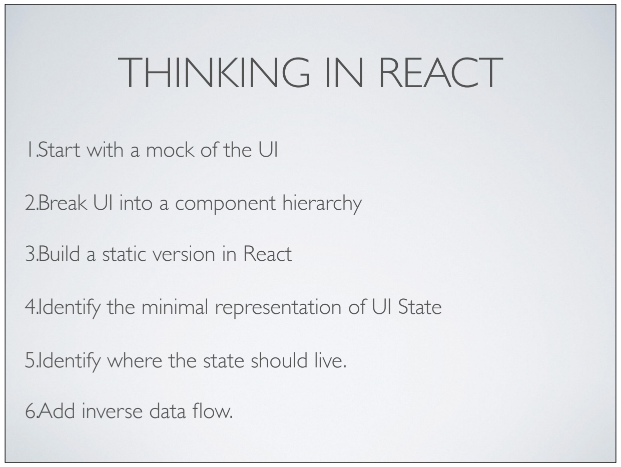
SINGLE PAGE APPS

- A closer approximation to a native app than a traditional Multi Page App (MPA)
- Fast!
- Compatible with SEO if constructed properly.
- Easier to debug
- Repurpose in mobile apps.



SINGLE PAGE APPS

- Example SPA frameworks
  - React (Facebook)
  - Angular (Google)
  - Meteor
  - Ember



THINKING IN REACT

1. Start with a mock of the UI
2. Break UI into a component hierarchy
3. Build a static version in React
4. Identify the minimal representation of UI State
5. Identify where the state should live.
6. Add inverse data flow.

STATE IN REACT

- For each piece of state:
  - id every component that renders something based on state.
  - find a common owner component (e.g. component above the components that need the state)
  - if you can't find a component where it makes sense to own state, then create one!

REACT COMPONENTS

- Pure function based component:
  - Get passed props they need and simply render content based on props passed.

```
const SearchBar = () => {
    return <div> Hello World! </div>;
};
```

REACT COMPONENTS

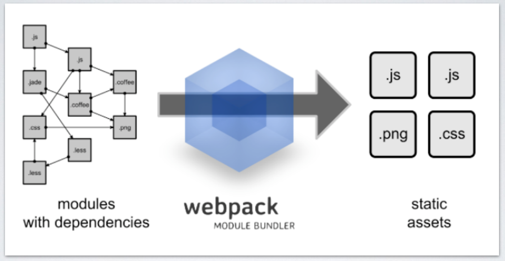
- Controlled component is one where its value set by state.

```
class SearchBar extends Component {
  constructor(props) {
    super(props);
    this.state = { term: '' };
  }

  render() {
    return (
      <div>
        <input
          value={this.state.term}
          onChange={event => this.setState({term: event.target.value })} />
        Value of the input: {this.state.term}
      </div>
    );
  }
}
```

REACT MACHINERY

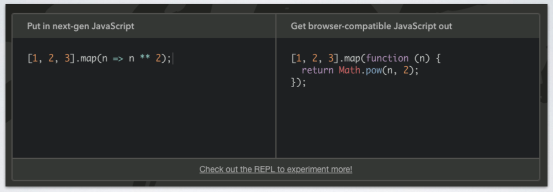
- Webpack
  - open-source JS module bundler
  - takes modules and dependencies and generates static assets.
  - Node based



https://github.com/webpack/docs/wiki/what-is-webpack

REACT MACHINERY

- Babel
  - JavaScript compiler (transpiler)
  - Allows to write "edge" JavaScript and converts it into an older version of JavaScript that most browsers can interpret (ES6 ==> ES5)


Put in next-gen JavaScript
Get browser-compatible JavaScript out

```
[1, 2, 3].map(n => n ** 2);
```

```
[1, 2, 3].map(function (n) {
  return Math.pow(n, 2);
});
```

Check out the REPL to experiment more!

Experiment with Babel.

EMERGING JAVASCRIPT

- React depends on a number of late breaking JavaScript features:
  - Arrow Functions
  - Spread Operators
  - Classes
  - Function context binding

JS: ARROW FUNCTIONS

```
// Old
var lordify = function(firstName, land) {
  return `${firstName} of ${land}`
}

// New
var lordify = (firstName, land) => `${firstName} of ${land}`
```

SPREAD OPERATORS

```
var peaks = ["Tallac", "Ralston", "Rose"]
var canyons = ["Ward", "Blackwood"]
var tahoe = [...peaks, ...canyons]

console.log(tahoe.join(', '))  // Tallac, Ralston, Rose, Ward, Blackwood

var lakes = ["Donner", "Marlette", "Fallen Leaf", "Cascade"]

var [first, ...rest] = lakes

console.log(rest.join(", ")) // "Marlette, Fallen Leaf, Cascade"
```

FUNCTION CONTEXT BINDING

- Many languages (e.g. Java, Ruby) set function context at definition. That is, this/self always point to the object context.
- JavaScript determine function context at the time it is called!
- There are a number of ways we can call a function in JavaScript…

FUNCTION INVOKE PATTERN

- Calling a function directly…
- *this* gets set to a global variable of an environment on which your JavaScript operates
- In browser, it is a *window* global variable.

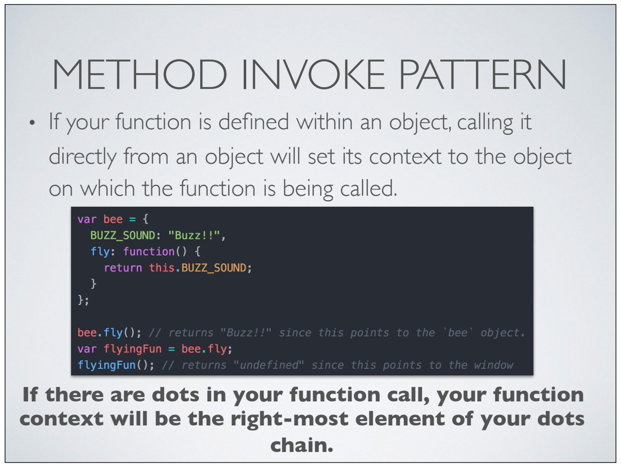
```
// definition of the function
var func = function() {
  // ...
};

func();  // invocation!
```

```
var bee = {
  func: function() { // ... }
};

var fun = bee.func;
fun();
```

If there are no dots in the invocation the context is likely a global window!



METHOD INVOKE PATTERN

- If your function is defined within an object, calling it directly from an object will set its context to the object on which the function is being called.

```
var bee = {
  BUZZ_SOUND: "Buzz!!",
  fly: function() {
    return this.BUZZ_SOUND;
  }
};

bee.fly(); // returns "Buzz!!" since this points to the `bee` object.
var flyingFun = bee.fly;
flyingFun(); // returns "undefined" since this points to the window
```

If there are dots in your function call, your function context will be the right-most element of your dots chain.