

ReactJS.

Fundamentals

Agenda

- **Background.**
- **JSX (JavaScript Extension Syntax).**
- **Developer tools.**
 - **Storybook.**
- **Component basics.**

ReactJS.

- **A Javascript framework for building dynamic Web User Interfaces.**
 - **A Single Page Apps technology.**
 - **Open-sourced in 2012**



- **Client-side framework.**
 - **More a library than a framework.**

Before ReactJS.

- MVC pattern – **The convention for app design. Promoted by market leaders**, e.g. **AngularJS (1.x), EmberJS, BackboneJS**.
- **React is not MVC, just V.**
 - **It challenged established best practice (MVC).**
- Templating – **widespread use in the V layer.**
 - **React based on components.**

	Templates	(React) Components
Separation of concerns	Technology (JS, HTML)	Responsibility
Semantic	New concepts and micro-languages	HTML and Javascript
Expressiveness	Underpowered	Full power of Javascript

ReactJS

- **Philosophy:** *Build components, not templates.*
- **All about the User Interface (UI).**
 - **Not about business logic or the data model (Mvc)**
- **Component - A unit comprised of:**
 - UI description (HTML) + UI behavior (JS)*
 - **Two aspects are tightly coupled and co-located.**
 - **Pre-React frameworks decoupled them.**
 - **Benefits:**
 - 1. Improved Composition.**
 - 2. Greater Reusability.**
 - 3. Better Performance.**

Creating the UI description

- React.createElement() – **create a HTML element.**
- ReactDOM.render() – **attach an element to the DOM.**
- createElement() **arguments:**
 1. **type (h1, div, button etc).**
 2. **properties (style, event handler etc).**
 3. **children (0 -> M).**
 - **We never use createElement() directly – too cumbersome.**
- ReactDOM.render() **arguments:**
 1. **element to be displayed.**
 2. **DOM node on which to mount the element.**
- **Ref. 01-UIDescription.html.**
- **Nesting createElement() calls - Ref. 02-UIDescription.html**

Code Demos

A screenshot of a web browser window displaying a file directory listing. The address bar shows the URL `127.0.0.1:5500`. The page content is a file tree structure:

```
~ /  
  js  
  02-UIDescription.html  
  05-simpleComponent.html  
  README  
  node_modules  
  03-JSX-error.html  
  package-lock.json  
  style.css  
  01-UIDescription.html  
  04-JSX.html  
  package.json
```

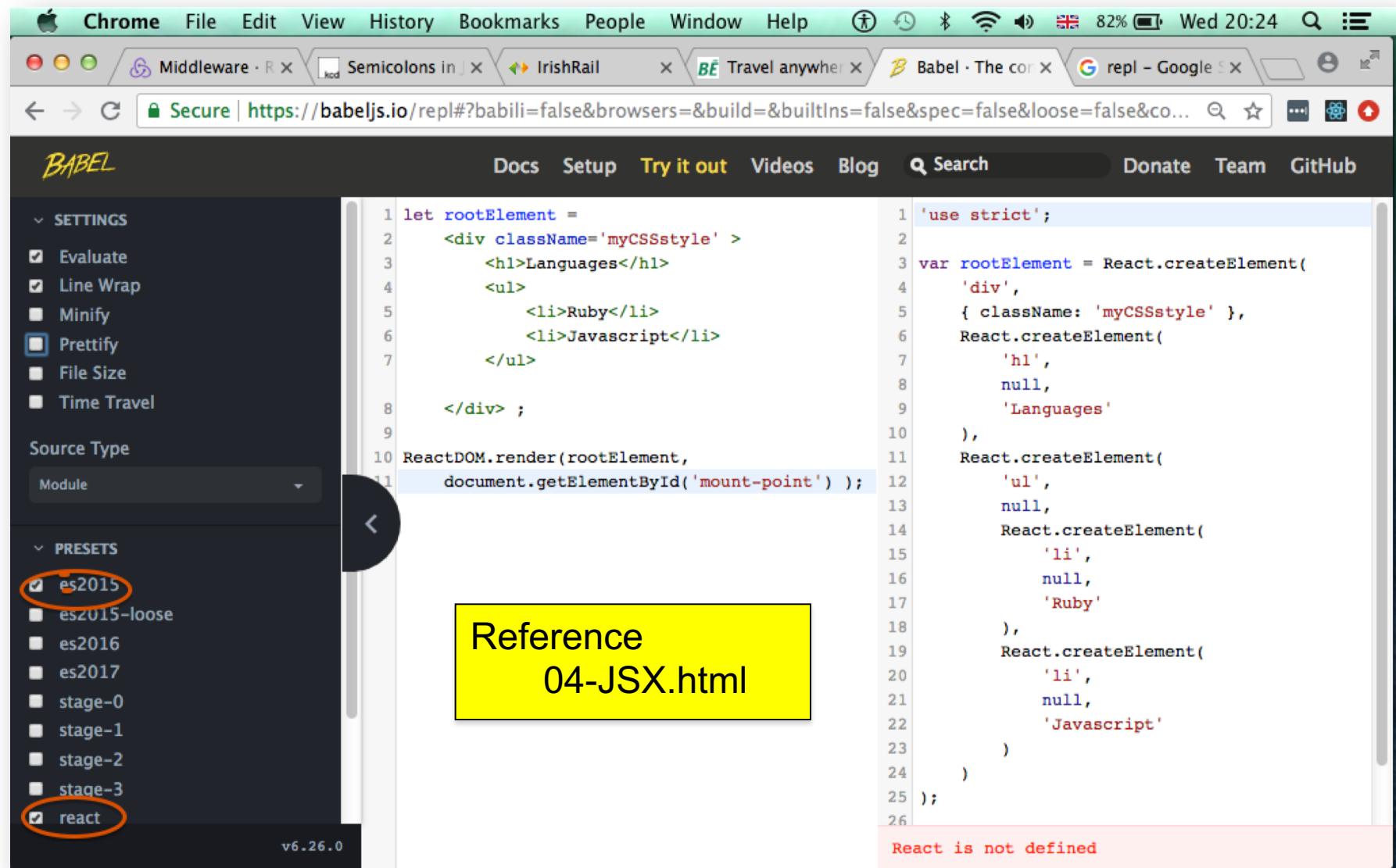
Overlaid on the bottom right of the file tree is a blue rectangular box containing white text:

See Archive accompanying these slides.
Run code using VS Code Live Server extension

JSX.

- **JSX – JavaScript extension syntax.**
- **Declarative syntax for coding UI descriptions.**
- **Retains the full power of Javascript.**
- **Allows tight coupling between UI behavior and UI description.**
- **Must be transpiled before being sent to browser.**
 - **The Babel tool**
- **Reference** 03-JSX-error.html and 04-JSX.html

REPL (Read-Evaluate-Print-Loop) transpiler.



JSX.

- **HTML-like markup.**
 - **It's actually XML code.**
- **Some minor HTML tag attributes differences, e.g. className (class), htmlFor (for).**
- **Allows UI description be coded in a declarative style and be inlined in JavaScript.**
- **Combines the ease-of-use of templates with the power of JS.**

Transpiling JSX.

- **What?**
 - The Babel platform
- **How?**
 - 1. Manually, via REPL or command line.**
 - When experimenting only.
 - 2. During development by the web server (using special tooling, i.e. Webpack).**
 - 3. Before deployment as part of the build process for an app.**

React Components.

- **We develop COMPONENTS.**
 - A JS function that **returns** a UI description, i.e. JSX.
- **Can reference a component like a HTML tag.**
e.g. ReactDOM.render(<ComponentName />, . . .)
- **Reference** 05-simpleComponent.html

React Developer tools.

- **create-react-app (CRA) - Features:**
 - **Scaffolding/Generator.**
 - **Development web server: auto-transpilation on file change + live reloading.**
 - **Builder: build production standard version of app, i.e. minification, bundling.**
- **Storybook - Features:**
 - **A development environment for React components.**
 - **Allows components be developed in isolation.**
 - **Promotes more reusable, testable components.**
 - **Quicker development – ignore app-specific dependencies.**



- **Installation:**

```
$ npm install @storybook/react
```

- **Tool has two aspects:**

1. **The server:**

```
$ ./node_modules/.bin/start-storybook -p 6006 -c ./storybook
```

- **Performs live re-transpilation and re-loading.**

2. **The User interface.**



- **Storybook UI (User interface).**

A screenshot of a web browser window titled "Storybook". The address bar shows the URL "localhost:9001/?selectedKind=DynamicLanguages&selectedStory=nor". The main content area displays a component catalog titled "Dynamic Languages". The catalog lists three items: "DynamicLanguages", "OtherComponentX", and "OtherComponentY". A red arrow points from the "DynamicLanguages" item to a bulleted list on the right: "Python" and "Javascript". A blue callout bubble labeled "Component catalogue" points to the left side of the catalog. Another blue callout bubble labeled "Component Rendering." points to the right side of the catalog.

STORYBOOK

Filter

Dynamic Languages

> DynamicLanguages

> OtherComponentX

> OtherComponentY

- Python
- Javascript

Component catalogue

Component Rendering.



- **What is a Story?**
- **A component may have several STATES → State effects how it renders.**
 - **Each state case termed a STORY.**
 - **Stories are a design consideration.**
- **EX.: DynamicLanguages component.**
 - **States might be:**
 - Default – **less than 5 languages** → **Render full list**
 - Boundary – **empty list** → **Render 'No languages' message**
 - Exceptional – **More than 5 languages** → **Render first 5 and a 'See More...' link to display next 5.**



- **Storybook UI – List a component's states/stories under its name:**

A screenshot of a web browser displaying the Storybook UI. The URL in the address bar is `localhost:9001/?selectedKind=DynamicLanguages&selectedStory=default`. The page title is "STORYBOOK". On the left, there is a sidebar with a "Filter" input field and a tree view of components. A blue speech bubble points to the "exceptional" story under the "DynamicLanguages" component, with the text "Set of Stories". An orange arrow points from the "exceptional" story to the list of stories on the right. The main content area displays the title "Dynamic Languages" and a list of stories: Python, Javascript, Ruby, PHP, and Groovy. There is also a "See more ..." link.

Set of Stories

- Python
- Javascript
- Ruby
- PHP
- Groovy

[See more ...](#)



- Define component groups when component library is large.
 - helps others understand the structure.

The screenshot shows a web browser window titled "Storybook" displaying a component library. The left sidebar, titled "STORYBOOK", contains a tree view of components:

- Group A
 - OtherComponentX
 - DynamicLanguages
 - default
 - boundary
 - exceptional** (highlighted in grey)
- Group B
 - OtherComponentY

The main content area has a title "Dynamic Languages" and a list of dynamic languages:

- Python
- Javascript
- Ruby
- PHP
- Groovy

[See more ...](#)

Writing stories

- Fluent-style syntax for writing stories.
 - Method chaining programming style.

```
1 import React from 'react';
2 import { storiesOf } from '@storybook/react';
3 import DynamicLanguages from '../components/dynamicLanguages';
4
5 storiesOf('DynamicLanguages', module)
6   .add('default',
7     () => {
8       let languages = ['Python', 'Javascript', 'Ruby']
9       return <DynamicLanguages list={languages} />
10    }
11  )
12  .add('boundary',
13    () => . . . . .
14  )
15  .add('exceptional',
16    () => . . . . .
17  )
18
19 storiesOf('OtherComponentX', module)
20   .add('state 1',
21     () => . . . . .
22  )
23  . . . . .
```

3 stories/states for DynamicLanguages component

- Story coded in a callback argument of add() method.
- add() must return a component instance.

Grouping stories.

- **Use directory pathname symbol (/) to indicate component grouping (i.e. group/subgroup/....). EX.:**

storiesof('Group A/Component 1')

.add('...'), () => {.....}

.add('...'), () => {.....}

storiesof('Group A/Component 2')

.add('...'), () => {.....}

.add('...'), () => {.....}

storiesof('Group B/Component X')

.add('...'), () => {.....}

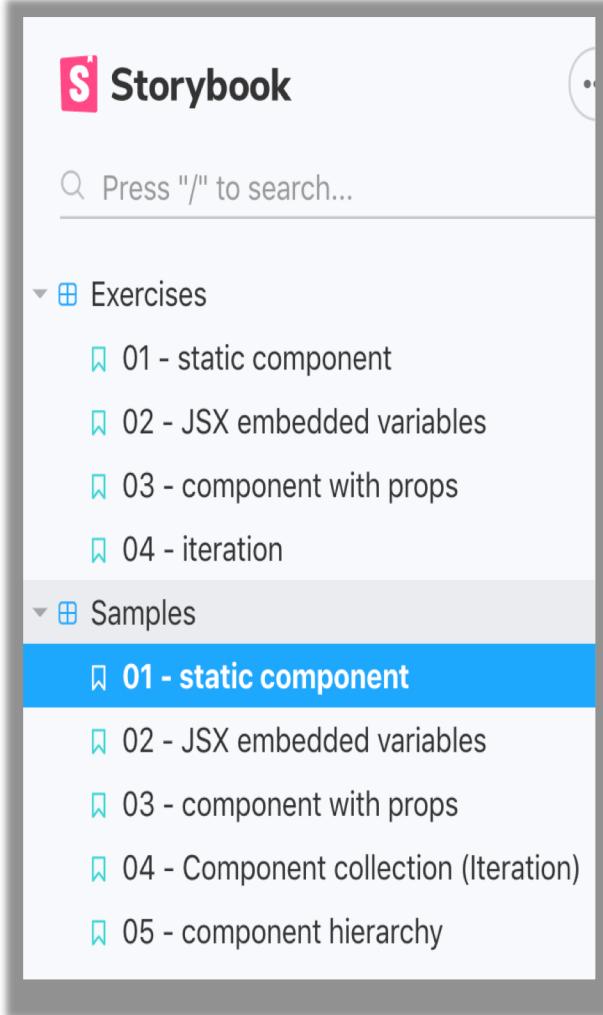
.add('...'), () => {.....}

.add('...'), () => {.....}

- **Lots of flexibility with grouping approach.**

... back to components . . .

Demo Samples



The screenshot shows a file tree for a project named "BASICREACTLAB". The tree includes the following files and folders:

- .storybook
- components
 - exercises
 - samples
 - 01_staticComponent.js
 - 02_embeddedVar.js
 - 03_props.js
 - 04_iteration.js
 - 05_hierarchy.js
- node_modules
- stories
 - exercises.stories.js
 - samples.stories.js
- .gitignore
- package-lock.json
- package.json

Configuration
– boilerplate

Lab exercise

Sample
Components

Stories

JSX embedded variables.

- Dereference variable embedded in JSX using {} braces.
 - Braces can contain any valid JS expression.
- Reference samples/02_embeddedVariables.js

```
JS 02_embeddedVar.js ×
components > samples > JS 02_embeddedVar.js > ...
1 import React from "react";
2
3 const Demo = () => {
4   const languages = ["Go", "Julia", "Kotlin"];
5   const header = "Modern";
6   return (
7     <div>
8       <h1>{` ${header} Languages`} </h1>
9       <ul>
10         <li>{languages[0]}</li>
11         <li>{languages[1]} </li>
12         <li>{languages[2]} </li>
13       </ul>
14     </div>
15   );
16 };
17
18 export default Demo
```

Reusability.

- **Achieve reusability through parameterization.**
- **props – Component properties / attribute / parameters.**
 1. **Passing props to a component:**
`<CompName prop1Name={value} prop2Name={value} . . . />`
 2. **Access inside component via props object:**
`const ComponentName = (props) => {
 const p1 = props.prop1Name

}`
 3. **Props are Immutable.**
 4. **Part of the a component's design.**
- **Reference samples/03_props.js (and related story).**

Aside – Some JS issues

- When an arrow function has only ONE statement, which is its return value, then you may omit:
 - Body curly braces; ‘return’ keyword.

```
const increment = (num) => {  
    return num + 1  
}
```

```
const increment = (num) => num + 1
```

Aside – Some JS issues

- The **Array map method** – returns a new array based on applying the function argument to each element of the source array.

```
1 let frameworks = [
2   {name: 'React', url : 'https://facebook.github.io/react/'},
3   {name: 'Vue', url : 'https://vuejs.org/'},
4   {name: 'Angular', url : 'https://angularjs.org/'}
5 ];
6 const names = frameworks.map((f,index) => `${index+1}. ${f.name}`)
7 console.log(names)
8 // [ '1. React', '2. Vue', '3. Angular' ]
9
```

Aside – Some JS issues

- We can assign a single JSX element to a variable.

```
9
0 const demo = <div>
1           <h1>Something</h1>
2           <h2>Something else</h2>
3           </div> ;
```

Component collection - Iteration

- **Use case:** A component prop array used to generate a collection of JSX elements.
- **Reference** samples/04_iteration.js

```
▼<div id="root">
  <h2>Most Popular client-side frameworks</h2> == $0
  ▼<ul>
    ▼<li>
      <a href="https://facebook.github.io/react/">React</a>
    </li>
    ▼<li>
      ▶<a href="https://vuejs.org/">...</a>
    </li>
    ▼<li>
      ▶<a href="https://angularjs.org/">...</a>
    </li>
  </ul>
</div>
```

Required HTML produced by component.
(From Chrome Dev Tools)

Component return value.

- **Examples:**
 1. `return <h1>Something</h1> ;`
 2. `return <MyComponent prop1={.....} prop2={.....} /> ;`
 3. `return (`
 `<div>`
 `<h1>{this.props.type}</h1>`
 ``
 `.`
 ``
 `</div>`
 `) ;`
 - **Must enclose in () when multiline.**

Component return value.

- **Must return only ONE element.**
- **Error Examples:**
 - ```
return (
 <h1>{this.props.type}</h1>

 . . .

);
```
  - **Error** – ‘Adjacent JSX elements must be wrapped in an enclosing tag’
  - **Solution: Wrap elements in a <div> tag.**

# Component return value.

- **Old solution:**

```
return (
 <div>
 <h1>{this.props.type}</h1>

 </div>
) ;
```

- **Adds unnecessary depth to DOM → effects performance.**

- **New solution:**

```
return (
 <>
 <h1>{this.props.type}</h1>

 </>
) ;
```

- **<> </> – special React element.**
  - **No DOM presence.**

# Component *Hierarchy*.

*A React application is designed as a hierarchy of components.*

- Components have children – nesting.
- Ref. 05\_hierarchy.js.

The screenshot shows the Storybook interface with the sidebar navigation open. The 'Samples' section is expanded, and the '05 - component hierarchy' item is selected, highlighted with a blue background. The main content area displays two sections: 'Ranked client-side frameworks' and 'Ranked Server-side Languages'. The 'Ranked client-side frameworks' section contains a list of three items: React, Vue, and Angular. The 'Ranked Server-side Languages' section contains a list of three items: Javascript, Python, and Java. Red circles with numbers 1, 2, and 3 are overlaid on the right side of the content, corresponding to the three items in each section. Below each section, there is a note indicating the data source: 'Data sourced from npm-stat.com' for the client-side frameworks and 'Data sourced from StackOverflow' for the server-side languages.

**Ranked client-side frameworks**

- React
- Vue
- Angular

Data sourced from [npm-stat.com](#)

**Ranked Server-side Languages**

- Javascript
- Python
- Java

Data sourced from [StackOverflow](#)

# Summary.

- **JSX.**
  - UI description and behaviour **tightly coupled**.
  - **Can embed variables/expressions with braces.**
- **All about components.**
  - A function that takes a **props argument** and returns a **single JSX element** .
  - **Components can be nested.**
- **Storybook tool.**
  - **Develop components in isolation.**
  - **Story – the state (data values) of a component can effect its rendering (and behaviour).**

