

# ReactJS.

## Afa 21

### Fundamentals

# Agenda

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

# ReactJS.

- A Javascript framework for building dynamic Web User Interfaces.
  - 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.**

	<b>Templates</b>	<b>React components</b>
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.**
    - **Other 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, span etc).**
  2. **properties (style, event handler etc).**
  3. **children (0 -> M).**
    - **We never use createElement() directly – far 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 as follows:

```
~ /
```

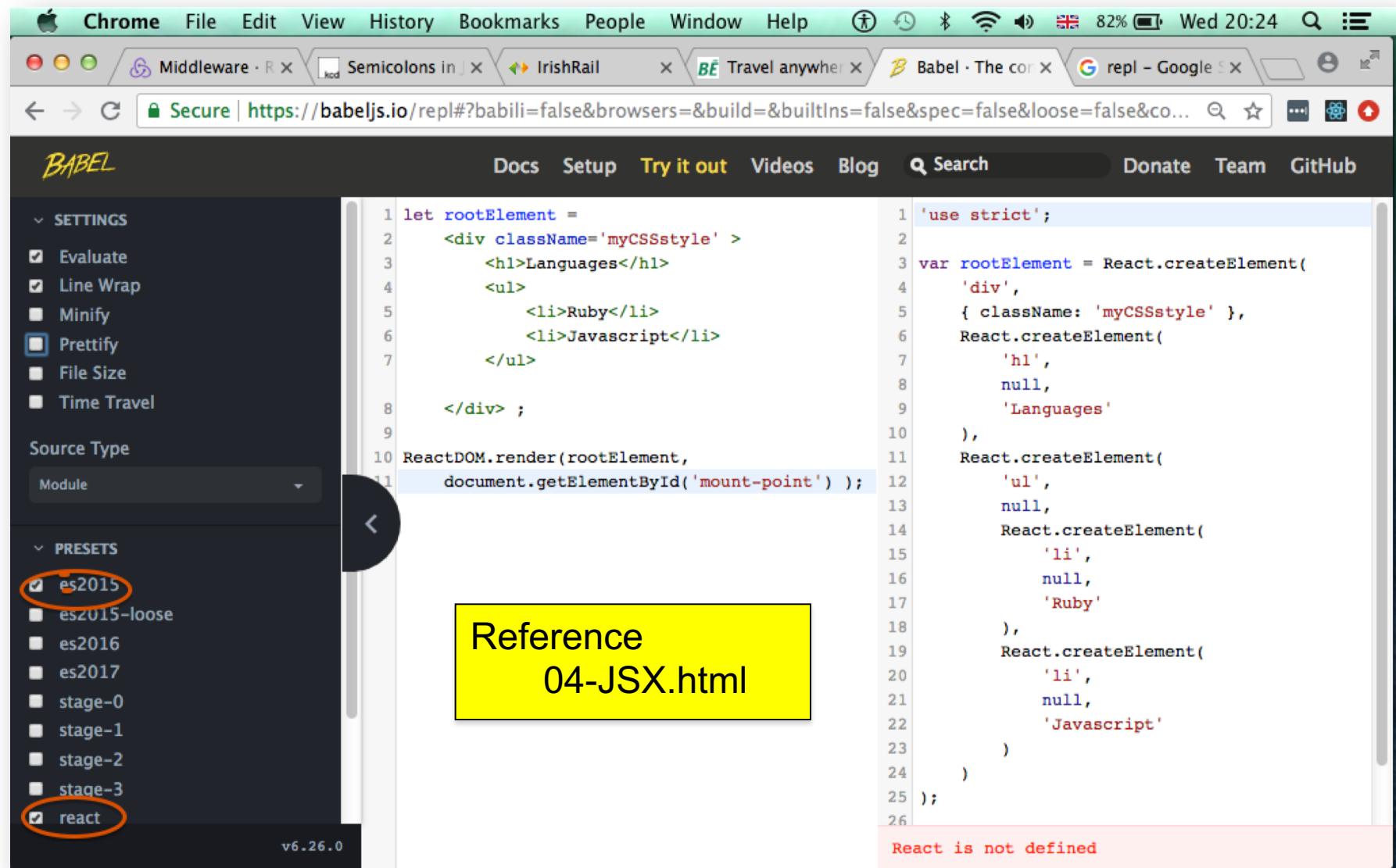
js	node_modules	01-UIDescription.html
02-UIDescription.html	03-JSX-error.html	04-JSX.html
05-simpleComponent.html	package-lock.json	package.json
README	style.css	

**See Archive available with slides.  
Run via Live Server**

# JSX.

- **JSX – JavaScript extension syntax.**
- **Declarative syntax for coding UI descriptions inside JS code**
- **Retains the full power of Javascript.**
- **Allows tight coupling between UI behavior and description.**
- **Must be transpiled (Babel) before being sent to browser.**
  - **Reference** 03-JSX-error.html
  - **Reference** 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 declarative description of the UI 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 UI components.**
  - **Develop component in isolation.**
  - **Leads to more reusable, testable components.**
  - **Quicker development – ignore app-specific dependencies.**



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

A screenshot of a web browser window displaying the Storybook UI. The title bar shows multiple tabs: "Storybook", "Middleware · R", "Semicolons in J", and "Travel anyw". The address bar shows the URL: "localhost:9001/?selectedKind=DynamicLanguages&selectedStory=nor".

The main interface has a header with a search bar labeled "STORYBOOK" and a "Filter" dropdown. Below the header, a sidebar lists component names: "DynamicLanguages", "OtherComponentX", and "OtherComponentY". A red arrow points from the "DynamicLanguages" entry in the sidebar to a list of languages on the right. The right side of the screen displays the content of the "DynamicLanguages" story, which includes the heading "Dynamic Languages" and a bulleted list: "Python" and "Javascript".

**Component names/labels** (blue callout pointing to the sidebar)

**Component Rendering.** (blue callout pointing to the language list)

- > DynamicLanguages →
  - Python
  - Javascript
- > OtherComponentX
- > OtherComponentY



- **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 indents stories inside the containing component:**

A screenshot of the Storybook UI interface. On the left, there's a sidebar titled "STORYBOOK" with a "Filter" input field. A blue callout bubble points to this sidebar with the text "Set of Stories". Below it is a tree view of components:

- DynamicLanguages
  - default
  - boundary
  - exceptional** (highlighted with a red arrow)
- OtherComponentX
- OtherComponentY

On the right, the "exceptional" story is expanded, showing the title "Dynamic Languages" and a list of dynamic languages:

- Python
- Javascript
- Ruby
- PHP
- Groovy

[See more ...](#)

The browser address bar at the top shows the URL: `localhost:9001/?selectedKind=DynamicLanguages&selectedStory=exceptional`.



- Can define component groups for large component libraries
  - helps others understand the catalogue.

The screenshot shows a web browser window titled "Storybook" displaying a component catalog. On the left, there's a sidebar with a "Filter" input field and a tree view of component groups:

- 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 components:

- 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 callback argument of add() method.
- add() returns component instance.

# Grouping stories.

- **Use directory pathname pattern**, group/subgroup/.... e.g.  
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

- Storybook server – performs live re-transpilation and re-loading.

The image shows a Storybook interface with a sidebar and a main content area. The sidebar has a 'STORYBOOK' header, a 'Filter' input field, and two sections: 'Samples' and 'Exercises'. The 'Samples' section contains links to '01 - static component', '02 - JSX embedded variables', '03 - component with props', '04 - Component collection (Iteration)', and '05 - component composition'. The 'Exercises' section contains links to '01 - static component', '02 - JSX embedded variables', '03 - component with props', and '04 - iteration'. The main content area displays a file tree for a 'BASICREACTLAB' project. The tree includes a '.storybook' folder, a 'components' folder containing 'exercises' and 'samples' subfolders, and a 'node\_modules' folder. Under 'samples', there are files named '01\_staticComponent.js', '02\_embeddedVar.js', '03\_props.js', '04\_iteration.js', and '05\_hierarchy.js'. Under 'stories', there are files named 'exercises.stories.js' and 'samples.stories.js'. Other files shown include '.gitignore', 'package-lock.json', and 'package.json'. Callouts with arrows point from the sidebar sections to their corresponding parts in the file tree:

- A blue callout points from the 'Samples' section in the sidebar to the 'samples' folder in the file tree, labeled 'Configuration – boilerplate'.
- A yellow callout points from the 'Exercises' section in the sidebar to the 'exercises' folder in the file tree, labeled 'Lab exercise'.
- A blue callout points from the 'Components' section in the sidebar to the 'components' folder in the file tree, labeled 'Components'.
- A blue callout points from the 'Stories' section in the sidebar to the 'stories' folder in the file tree, labeled 'Stories'.

STORYBOOK

Filter

Samples

01 - static component

02 - JSX embedded variables

03 - component with props

04 - Component collection (Iteration)

05 - component composition

Exercises

01 - static component

02 - JSX embedded variables

03 - component with props

04 - iteration

BASICREACTLAB

- .storybook
- components
  - exercises
  - samples
    - JS 01\_staticComponent.js
    - JS 02\_embeddedVar.js
    - JS 03\_props.js
    - JS 04\_iteration.js
    - JS 05\_hierarchy.js
- node\_modules
- stories
  - JS exercises.stories.js
  - JS samples.stories.js
- .gitignore
- { } package-lock.json
- { } package.json

# 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.**
  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 component design process.**
- **Reference samples/03\_props.js and related story.**

# Aside – Some JS issues

- **Shorthand arrow function – many parts of the syntax is optional for single-parameter and single body statement cases.**
- **The Array map method – returns a new array based on applying the function argument to every element of the source array.**
- **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:** Generate a collection of component/elements instances based on a prop array.
- **Reference** samples/04\_iteration.js

```
▼<div>
  <h1>JS client-side Web</h1>
  ▼<ul> = $0
    ▼<li>
      <a href="https://facebook.github.io/react/">React</a>
    </li>
    ▼<li>
      <a href="https://vuejs.org/">Vue</a>
    </li>
    ▼<li>
      <a href="https://angularjs.org/">Angular</a>
    </li>
  </ul>
</div>
```

Required Real DOM 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>`  
    `<ul>`  
    `. . . . .`  
    `</ul>`  
    `</div>`  
    `) ;`
  - **Must enclose in ( ) when multiline.**

# Component return value.

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

- **Added unnecessary depth to DOM → effected performance.**

- **New solution:**

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

- **<> </> – 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 expanded. The 'Samples' section is selected, and the '05 - component hierarchy' item is highlighted in blue. The main content area displays two sections: 'Ranked client-side frameworks' and 'Ranked Server-side Languages'. The 'Ranked client-side frameworks' section lists React, Vue, and Angular, with a note that data is sourced from npm-stat.com. The 'Ranked Server-side Languages' section lists Javascript, Python, and Java, with a note that data is sourced from StackOverflow. Red circles numbered 1, 2, and 3 are overlaid on the right side of the content blocks, likely indicating the depth of nesting or a specific component ID.

**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).**

# **ES6 top-up.**

# Modularity

- **Split application code into multiple files, termed modules.**
- **Promotes:**
  - **Reusability.**
  - **Understandability.**
  - **Maintainability.**
- **Pre-ES6 provided module system via separate library;**
- **ES6 modules built into language.**

# Modularity (ES6)

- Two options:
  - Default exports;
  - Named exports (also Mixed exports)

```
29 // lib.js
30 export default function() {
31   ...
32 }
33 // -----
34 // bar.js
35 import myFunc from './lib';
36 myFunc();
37
```

```
// lib.js (Named exports)
export let myStr = 'important string';
export const pi = 3.14159526;
export function myFunc() {
  ...
}
export class myClass {
  ...
}
// -----
// bar.js
import {myFunc, myStr} from './lib';
myFunc();
console.log(myStr)
```

```
1 // 3rd party NPM modules
2 // No pathname required      You
3 import React from "react";
4
```

# Destructuring

- Assigning the elements of an array/object to variables using a declarative style rather than an imperative / procedural style..

```
{  
  // without destruction  
  let nums = [10, 11, 12]  
  let v1 = nums[0]  
  let v2 = nums[1]  
  let v3 = nums[2]  
}  
  
{  
  // with destruction  
  let nums = [10, 11, 12]  
  let [v1, v2, v3] = nums  
  console.log(` ${v1} ${v2} ${v3}`) // 10 11 12  
}  
  
{  
  // with destruction  
  let nums = [10, 11, 12, 13, 14, 15];  
  let [,v2,,,v5] = nums  
  console.log(` ${v2} ${v5}`) // 11 14  
  let [v1,,v3,v4] = nums;  
  console.log(` ${v1} ${v3} ${v4}`); // 10 12 13 |  
}
```

# Destructuring

```
29 // OBJECTS
30 {
31     // without destructuring
32     let obj = { alpha: 100, beta: 'enterprise'}
33     let alpha = obj.alpha
34     let beta = obj.beta
35 }
36 {
37     // with
38     let obj = { alpha: 100, beta: 'enterprise'}
39     let {alpha, beta} = obj;
40     console.log(` ${alpha} ${beta}`) // 100 enterprise
41 }
42 {
43     // order is not important
44     let obj = { alpha: 100, beta: 'enterprise'}
45     let { beta, alpha } = obj;
46     console.log(` ${alpha} ${beta}`) // 100 enterprise
47 }
```

```
{
    // isolate particular property
    let obj = { alert: 100, beta: 'enterprise',
    | | | | foo: 'bar' }
    let { foo } = obj
    console.log(`${foo}`) // bar
}

{
    // alternative variable name choices
    let obj = { alpha: 100, beta: 'enterprise'}
    let {alpha: num, beta: word} = obj;
    console.log(` ${num} ${word}`) // 100 enterprise
}

{
    let obj = { alert: 100, beta: 'enterprise',
    | | | | foo: 'bar' }
    let { foo: thing, beta: word} = obj
    console.log(` ${word} ${thing}`) // enterprise bar
}
```

