

ReactJS.

Fundamentals

Agenda

- **Background.**
- **The V in MVC**
- **JSX (JavaScript Extension Syntax).**
- **Developer tools.**
- **React Component basics.**
- **(Material Design.)**

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.**
- Templating – **widespread use in the V layer.**
 - **React favored a component model.**

	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

Components

- **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.**

Creating the UI description

- React.createElement() – **create a HTML element.**
- ReactDOM.render() – **attach an element to the DOM.**
- React.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.**

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
```

Below the file tree, a blue callout box contains the text: "See Archive accompanying these slides."

UI description implementation

(Vanilla React - the imperative way)

- **See the demos:**
 - **Ref.** 01-UIDescription.html.
 - **Nesting createElement() calls - Ref.** 02-UIDescription.html
-

Imperative programming is a programming paradigm that uses statements that change a program's state.

Declarative programming is a programming paradigm that expresses the logic of a computation without describing its control flow.

UI description implementation

(Developer-friendly - the declarative way)

- **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 to vanilla form before being sent to browser.**
 - **The Babel tool**
- **Reference** 03-JSX-error.html and 04-JSX.html

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

The screenshot shows a Chrome browser window with the Babel REPL interface. The URL in the address bar is [https://babeljs.io/repl#?babili=false&browsers=&build=&builtIns=false&spec=false&loose=false&code=%0D%0Alet%20rootElement%20=%0D%0A%20%20%20%20%3Cdiv%20className='myCSSstyle'%20%3E%0D%0A%20%20%20%20%20%20%20%20%3Chl>Languages%3C/hl>%0D%0A%20%20%20%20%20%20%20%20%3Cul%3E%0D%0A%20%20%20%20%20%20%20%20%20%20%20%3Cli>Ruby%3C/li%3E%0D%0A%20%20%20%20%20%20%20%20%20%20%20%3Cli>Javascript%3C/li%3E%0D%0A%20%20%20%20%20%20%20%20%20%20%3C/ul%3E%0D%0A%20%20%20%20%3C/div%3E%20;%0D%0A%20%20%20%20ReactDOM.render\(rootElement,%0D%0A%20%20%20%20%20%20%20%20document.getElementById\('mount-point'\)\)%20\);](https://babeljs.io/repl#?babili=false&browsers=&build=&builtIns=false&spec=false&loose=false&code=%0D%0Alet%20rootElement%20=%0D%0A%20%20%20%20%3Cdiv%20className='myCSSstyle'%20%3E%0D%0A%20%20%20%20%20%20%20%20%3Chl>Languages%3C/hl>%0D%0A%20%20%20%20%20%20%20%20%3Cul%3E%0D%0A%20%20%20%20%20%20%20%20%20%20%20%3Cli>Ruby%3C/li%3E%0D%0A%20%20%20%20%20%20%20%20%20%20%20%3Cli>Javascript%3C/li%3E%0D%0A%20%20%20%20%20%20%20%20%20%20%3C/ul%3E%0D%0A%20%20%20%20%3C/div%3E%20;%0D%0A%20%20%20%20ReactDOM.render(rootElement,%0D%0A%20%20%20%20%20%20%20%20document.getElementById('mount-point'))%20);)

The interface includes a sidebar with **SETTINGS** (Evaluate, Line Wrap, Minify, Prettify, File Size, Time Travel), **Source Type** (Module, Script), and **PRESETS** (es2015, es2015-loose, es2016, es2017, stage-0, stage-1, stage-2, stage-3, react). The **react** preset is highlighted with a red oval.

The main area displays two blocks of code. The left block is JSX:

```
1 let rootElement =
2   <div className='myCSSstyle' >
3     <hl>Languages</hl>
4     <ul>
5       <li>Ruby</li>
6       <li>Javascript</li>
7     </ul>
8   </div> ;
9
10 ReactDOM.render(rootElement,
11   document.getElementById('mount-point'));
```

The right block is the transpiled JavaScript:

```
1 'use strict';
2
3 var rootElement = React.createElement(
4   'div',
5   { className: 'myCSSstyle' },
6   React.createElement(
7     'hl',
8     null,
9     'Languages'
10   ),
11   React.createElement(
12     'ul',
13     null,
14     React.createElement(
15       'li',
16       null,
17       'Ruby'
18     ),
19     React.createElement(
20       'li',
21       null,
22       'Javascript'
23     )
24   );
25 );
26
27 React is not defined
```

A yellow box at the bottom left contains the text "Reference 04-JSX.html".

JSX.

- **HTML-like markup.**
 - It's actually XML code.
- Some minor HTML tag attributes differences, e.g. `className` (`class`), `htmlFor` (`for`).
- Allows UI description to 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. Using specially instrumented web server during development mode - the Webpack library..**
 - 3. Using bundler tools as part the build process before deployment – Webpack again.**

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(<ComponentX />, . . .)`
- **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 three aspects:**
 1. **An API/library.**
 2. **A web server.**
 \$./node_modules/.bin/**start-storybook -p 6006 -c ./storybook**
 - **Performs live re-transpilation and re-loading.**
 3. **Web browser user interface.**



- **Storybook User interface.**

A screenshot of a web browser window showing the Storybook user interface. 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 content area has a header "STORYBOOK" and a "Filter" input field. On the left, there is a "Component catalogue" sidebar with a list of components: "DynamicLanguages", "OtherComponentX", and "OtherComponentY". A red arrow points from the "DynamicLanguages" item in the catalogue to a list of "Python" and "Javascript" items on the right. A blue callout bubble points from the "Component catalogue" sidebar towards the "Component Rendering" text on the right. The main content area displays the title "Dynamic Languages".

STORYBOOK

Filter

> DynamicLanguages →

- Python
- Javascript

Component catalogue

Component Rendering.

Dynamic Languages



- **What is a Story?**
 - From the User Story idea in User Requirements Analysis.
- **A component may have several STATES → State effects how it renders.**
 - Each state case is termed a STORY.
 - Stories are a design consideration.
- EX.: DynamicLanguages component – its states might be:
 - Default – 5 or less 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

- List a component's states/stories under its name:

The screenshot shows a web browser window for the URL `localhost:9001/?selectedKind=DynamicLanguages&selectedStory=exceptional`. The browser title bar says "STORYBOOK". On the left, there's a sidebar with a "Filter" input field and a list of components: "DynamicLanguages" (expanded), "OtherComponentX", and "OtherComponentY". Under "DynamicLanguages", the "exceptional" story is selected, highlighted with a blue background. A blue speech bubble labeled "Set of Stories" points to this selection. An orange arrow points from the "exceptional" story in the sidebar to the list of languages on the right. The main content area has a large heading "Dynamic Languages" and a bulleted list of languages: Python, Javascript, Ruby, PHP, and Groovy. There's also a "See more ..." link.

Set of Stories

Dynamic Languages

- Python
- Javascript
- Ruby
- PHP
- Groovy

[See more ...](#)



- We can also define groups for large component libraries.
 - helps others team members with searching.

A screenshot of a web browser displaying the Storybook interface at `localhost:9001/?selectedKind=Group%20A%2FDynamicLanguages`. The browser has several tabs open, including "Storybook", "Middleware - R", "Semicolons in J", and "Tr".

The main content area shows a sidebar with a tree view of components:

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

To the right of the sidebar, the title "Dynamic Languages" is displayed in a large, bold font. Below it is a bulleted list of dynamic languages:

- Python
- Javascript
- Ruby
- PHP
- Groovy

[See more ...](#)

Aside – JS Modularity.

- Split application code into multiple files, termed **modules**.
- Reusability - make modules available to other modules..
- Pre-ES6 provided module system via separate library;
- ES6 modules built into language.
 - Two main 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)
```

Writing stories

- **.stories.js file extension (convention).**
- **One Stories file per component.**

```
import React from "react";
import DynamicLanguages from "../components/dynamicLanguages";

export default {
  title: "Dynamic Languages",
  component: DynamicLanguages,
};

export const Default = () => {
  const list = ["Javascript", "Python", "Java", "C#"];
  return <DynamicLanguages languages={list} />;
};

export const Exceptional = () => {
  .....
};

export const Error = () => {
  .....
};
```

A screenshot of a code editor showing a file named 'DynamicLanguages.stories.js'. The code defines three stories: 'Default', 'Exceptional', and 'Error'. The 'Default' story uses a named export and metadata to identify the component and its title. A callout box points to this section with the following notes:

- default export.
- Metadata.
- Used by Storybook UI

The 'Default' story uses a function export to implement the story. A red curly brace groups the 'Default' and 'Exceptional' stories. A callout box points to this group with the following notes:

- Story implemented as a function.
- Named exports.
- 3 stories for this component

Writing stories (the old way)

- 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/dynam:
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   . . . .
```

Grouping stories.

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

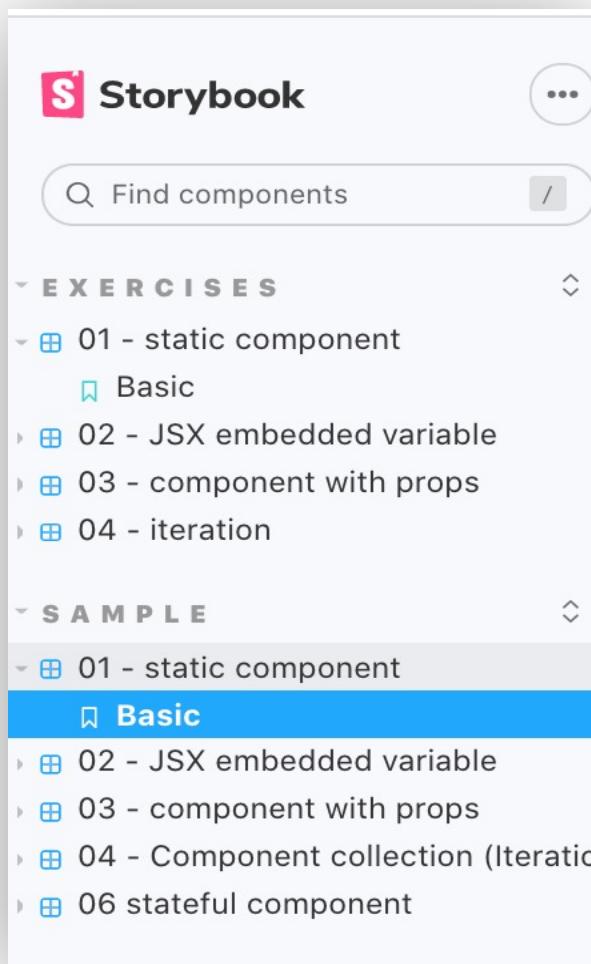
```
export default {  
  title: "Group A/ Component 1",  
  component: Component1,  
};  
  
... stories ...
```

```
export default {  
  title: "Group A/ Component 2",  
  component: Component2,  
};  
  
... stories ...
```

```
export default {  
  title: "Group B/ Component X",  
  component: Component1,  
};  
  
... stories ...
```

... back to components . . .

Demo Samples



The screenshot shows a file explorer window displaying a React project structure. The structure includes:

- BASICREACTLAB**
 - .storybook
 - node_modules
 - src
 - components
 - exercises
 - samples
 - 01_staticComponent.js
 - 02_embeddedVar.js
 - 03_props.js
 - 04_iteration.js
 - 05_hierarchy.js
 - 06_state.js
 - stories
 - exercises
 - samples
 - .gitignore
 - package-lock.json
 - package.json

Annotations with callouts point to specific files and folders:

 - A blue callout points to ".storybook" with the text "Configuration – boilerplate".
 - A yellow callout points to "01_staticComponent.js" with the text "Lab exercise".
 - A blue callout points to "01_staticComponent.js" with the text "Sample Components".
 - A blue callout points to "stories" with the text "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.

- **We 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. **Choosing appropriate props is a design issue.**
- **Reference** samples/03_props.js (and related story).

Aside – Some JS features

- 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 features

- 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 features.

- 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> ;
```

- Why?

```
const demo = React.createElement(
  "div",
  null,
  React.createElement("h1", null, "Something"),
  React.createElement("p", null, "Some text ...")
);
```

Component collection - Iteration

- **Use case:** We want to generate an array of (similar) component from a data array.
- **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 <MyComponent prop1={.....} prop2={.....} /> ;
 2. return (

```
<div>
  <h1>{this.props.type}</h1>
  <MyComponent prop1={.....} prop2={.....} />
  <p>
    . . .
  </p>
</div>
```

) ;
- **Must enclose in () when multiline.**

Component return value.

- **Must return only ONE element.**
- **Error Examples:**
 - ```
return (
 <h1>{this.props.type}</h1>
 <MyComponent prop1={.....} prop2={.....} />
 <p>

 </p>
) ;
```
  - **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></h1>
 <MyComponent />
 <p> </p>
 </div>
) ;
```
- **Adds unnecessary depth to DOM → effects performance.**
- **Alternative solution:**

```
return (
 <>
 <h1></h1>
 <MyComponent />
 <p> </p>
</>
) ;
```
- **<> </> – special React element, termed a Fragment.**
  - **No DOM presence.**

# Component *Hierarchy*.

All React application are 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. 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 boxes, likely indicating a nesting or parent-child relationship structure.

**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).**

