

ReactJS (v18+).

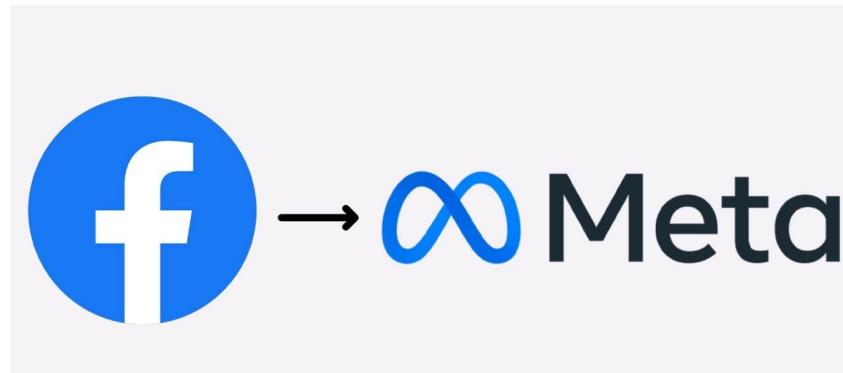
Fundamentals

Agenda

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

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.**
 - **Pre-React frameworks decoupled them.**
 - **Benefits:**
 - 1. Improved Composition.**
 - 2. Greater Reusability.**

Creating the UI description.

(Vanilla React)

- **React.createElement()** – **create a HTML element.**
- **ReactDOM.createRoot()** – **which existing DOM node to attach the created element.**
- **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.**
- **ReacrDOM.createRoot () arguments:**
 1. **DOM node on which to mount a new element.**

Code Demos.

(See lecture archive)

A screenshot of a web browser window displaying a file directory listing at the URL 127.0.0.1:5500. The directory structure shown is:

```
~ /  
  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
```

Below the file listing, a blue callout box contains the text:

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

UI description implementation

(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

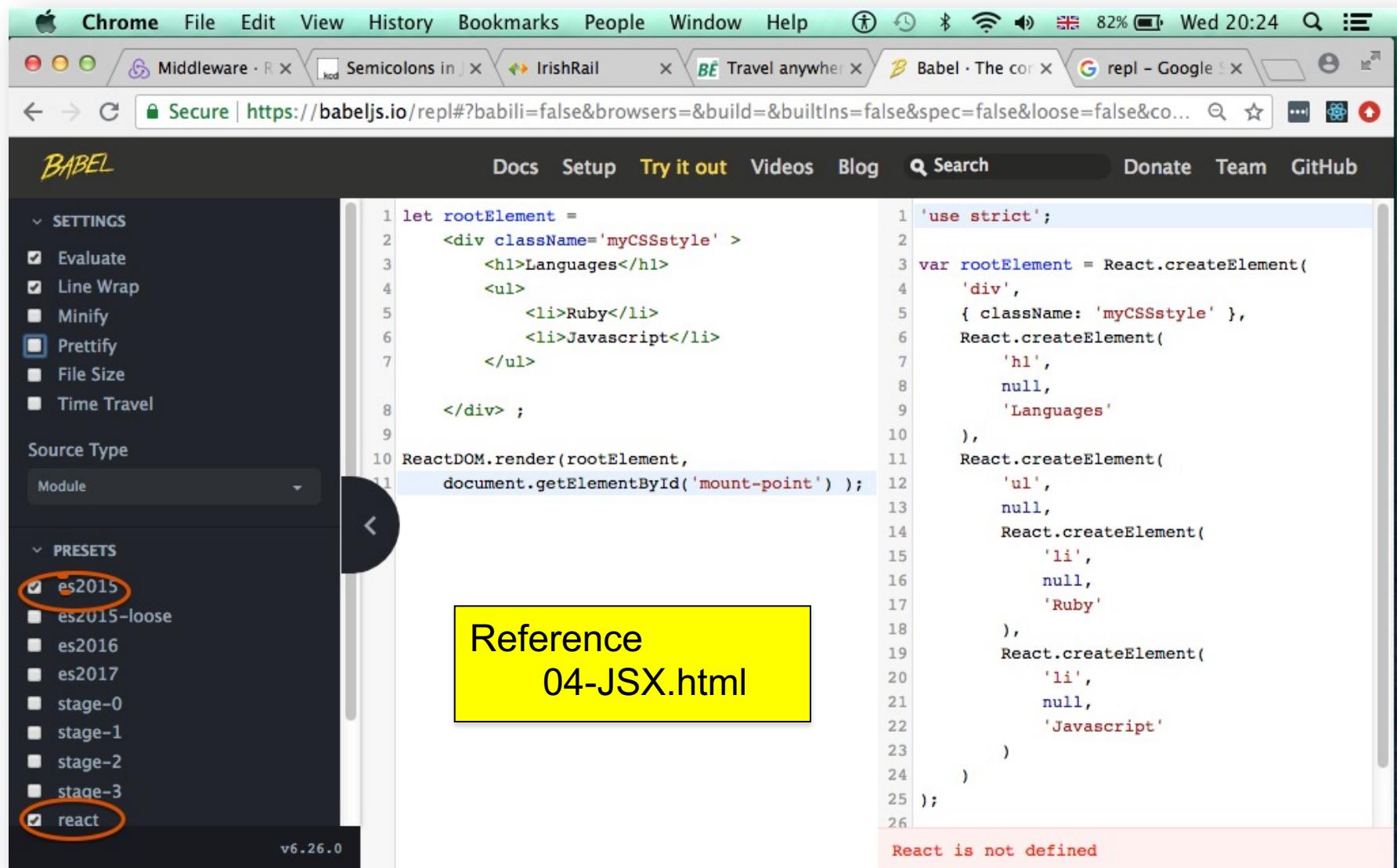
Declarative programming is a programming paradigm ... that expresses the logic of a computation without describing its control flow.

UI description implementation

(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 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 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.
 - The **Vite library**.
- **How?**
 1. **Manually, via REPL or command line.**
 - When experimenting only.
 2. **Using an instrumented web server – Vite library instrumentation.**
 - Ideal for development.
 3. **Using bundler tools as part of the build process – Vite again.**
 - Production standard.

React Components.

- **We develop COMPONENTS.**
 - A JS function **that returns a UI description**, i.e. JSX.
- **We reference a component like a HTML tag.**
 - e.g.

```
const rootElement =  
  ReactDOM.createRoot(document.getElementById("mount-point"));  
rootElement.render( <DynamicLanguages/> );
```

- **Reference 05-simpleComponent.html**

React Developer tools.

- Vite - **Features:**
 - **Scaffolding/Generator.**
 - **Development web server: auto-transpilation on file change + live reloading (HMR – Hot Module Replacement).**
 - **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
- **The tool has two aspects:**
 1. **A web server.**
\$./node_modules/.bin/*start-storybook -p 6006 -c ./storybook*
 - **Performs live re-transpilation and re-loading.**
 2. **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" with a search icon. Below it is a "Filter" input field. On the left, there's a sidebar with a list of components: "DynamicLanguages", "OtherComponentX", and "OtherComponentY". A red arrow points from the "DynamicLanguages" item in the sidebar to a list of language options on the right. The right side displays the "Dynamic Languages" component with two bullet points: "Python" and "Javascript". A blue callout bubble points to this list with the text "Component Rendering.". A blue callout bubble also points to the sidebar with the text "Component catalogue".



- **What is a Story?**
- **A component may have several STATES → State affects how it renders.**
 - **Each state case termed a STORY.**
 - **Stories are a design consideration.**
- **EX.: DynamicLanguages component.**
 - **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 with the URL `localhost:9001/?selectedKind=DynamicLanguages&selectedStory=exceptional`. The page title is "STORYBOOK". On the left, there is a sidebar with a "Filter" input field and a list of components: "DynamicLanguages", "OtherComponentX", and "OtherComponentY". The "DynamicLanguages" item is expanded, showing three story types: "default", "boundary", and "exceptional". A blue speech bubble labeled "Set of Stories" points to the "exceptional" story. 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 is also a "See more ..." link.

Set of Stories

Dynamic Languages

- Python
- Javascript
- Ruby
- PHP
- Groovy

[See more ...](#)



- Define component groups when component catalogue is large.
 - helps others team members with searching.

The screenshot shows a web browser window titled "Storybook" with the URL "localhost:9001/?selectedKind=Group%20A%2FDynamicLanguages". The interface has a sidebar on the left labeled "STORYBOOK" with a "Filter" input field. The sidebar contains a tree view of component groups:

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

The main content area is titled "Dynamic Languages" and lists the following components:

- 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)**
- **1 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 = () => {
  .....
};
```

default export; Metadata; How Storybook lists components.

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

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

(See lab exercise)

The image shows two side-by-side views. On the left is the Storybook interface, displaying a sidebar with 'EXERCISES' and 'SAMPLE' sections. Under 'SAMPLE', 'Basic' is selected, showing sub-options like '02 - JSX embedded variable'. On the right is a file explorer showing a project structure for 'BASICREACTLAB'. The structure includes a '.storybook' file, a 'node_modules' folder, a 'src' directory containing 'components' (with 'exercises' and 'samples' subfolders) and 'stories' (with 'exercises' and 'samples' subfolders). Below 'src' are files: '01_staticComponent.js', '02_embeddedVar.js', '03_props.js', '04_iteration.js', '05_hierarchy.js', and '06_state.js'. At the bottom are '.gitignore', 'package-lock.json', and 'package.json'. Callout boxes point from specific parts of the interface to corresponding files in the file explorer:

- A blue callout points from the 'Basic' section in the Storybook sidebar to the 'Basic' file in the file explorer.
- A yellow callout points from the '02 - JSX embedded variable' option in the Storybook sidebar to the '02_embeddedVar.js' file in the file explorer.
- A blue callout points from the '01 - static component' section in the Storybook sidebar to the '01_staticComponent.js' file in the file explorer.
- A yellow callout points from the '03 - component with props' option in the Storybook sidebar to the '03_props.js' file in the file explorer.
- A blue callout points from the '04 - iteration' option in the Storybook sidebar to the '04_iteration.js' file in the file explorer.
- A blue callout points from the '06 stateful component' option in the Storybook sidebar to the '06_state.js' file in the file explorer.
- A yellow callout points from the 'Lab exercise' section in the Storybook sidebar to the '05_hierarchy.js' file in the file explorer.
- A blue callout points from the 'Configuration – boilerplate' section in the Storybook sidebar to the '.storybook' file in the file explorer.
- A blue callout points from the 'Lab exercise' section in the Storybook sidebar to the '05_hierarchy.js' file in the file explorer.
- A blue callout points from the 'Sample Components' section in the Storybook sidebar to the 'samples' folder in the file explorer.
- A blue callout points from the 'Stories' section in the Storybook sidebar to the 'stories' folder in the file explorer.

JSX - embedded variables.

- Use {} to dereference variable embedded in JSX.
 - Curly braces can contain any valid JS expression.
- Reference src/components/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. **Part of a component's design.**
- **Reference** `src/components/samples/03_props.js` (**and related story**).

Aside.

- 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:** Generate an array of (similar) component from a data array.
- **Reference** src/components/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 → affects performance.**
- **Alternative solution:**

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

# Component *Hierarchy*.

All React application are designed as a hierarchy of components.

- Components have children – nesting.
- Ref. src/components/samples/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 interface, pointing to the 'Ranked client-side frameworks' section, the 'Ranked Server-side Languages' section, and the '05 - component hierarchy' item in the sidebar respectively.

Storybook

Press "/" to search...

Exercises

Samples

- 01 - static component
- 02 - JSX embedded variables
- 03 - component with props
- 04 - Component collection (Iteration)
- 05 - component hierarchy**

## Ranked client-side frameworks

- React
- Vue
- Angular

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

## Ranked Server-side Languages

- Javascript
- Python
- Java

Data sourced from [StackOverflow](#)

# React version 18

- **Released March 2022.**
- One significant breaking change.

```
You, 1 second ago | 1 author (You)
1 import React from 'react'
2 import ReactDOM from 'react-dom'
3 import App from './App'
4 You, 5 months ago • Completed app but using Bootstrap
5 ReactDOM.render(<App />, document.getElementById('root'));
```

```
You, 1 second ago | 1 author (You)
1 import React from 'react' React 18
2 import { createRoot } from 'react-dom/client'
3 import App from './App'
4 You, 3 weeks ago • Initialize project using Create React App
5 const rootElement = createRoot(document.getElementById('root'))
6 rootElement.render(<App />);
```

# 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).**