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.**
- ReacrDOM.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

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

JSX.

- **JSX – JavaScript extension syntax.**
- **<u>Declarative</u> <u>syntax</u> 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.

Chrome File Edit View History Bookmarks People Window Help 82% Wed 20:24

Middleware · R | Semicolons in | IrishRail | Travel anywhe | Babel · The co | repl – Google

Secure | https://babeljs.io/repl#?babili=false&browsers=&build=&builtIns=false&spec=false&loose=false&co...

BABEL Docs Setup Try it out Videos Blog Search Donate Team GitHub

SETTINGS
- [x] Evaluate
- [x] Line Wrap
- [ ] Minify
- [ ] Prettify
- [ ] File Size
- [ ] Time Travel

Source Type
Module

PRESETS
- [x] es2015
- [ ] es2015-loose
- [ ] es2016
- [ ] es2017
- [ ] stage-0
- [ ] stage-1
- [ ] stage-2
- [ ] stage-3
- [x] react

v6.26.0

```
let rootElement =
    <div className='myCSSstyle' >
        <h1>Languages</h1>
        <ul>
            <li>Ruby</li>
            <li>Javascript</li>
        </ul>

    </div> ;

ReactDOM.render(rootElement,
    document.getElementById('mount-point') );
```

```
'use strict';

var rootElement = React.createElement(
    'div',
    { className: 'myCSSstyle' },
    React.createElement(
        'h1',
        null,
        'Languages'
    ),
    React.createElement(
        'ul',
        null,
        React.createElement(
            'li',
            null,
            'Ruby'
        ),
        React.createElement(
            'li',
            null,
            'Javascript'
        )
    )
);
```

React is not defined

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 **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 <u>HTML tag.</u>**

  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.**

STORYBOOK

- **Installation:**

  $ npm install @storybook/react

- **Tool has two aspects:**

  1. **The server:**

     $ ./node_modules/.bin/***start-storybook -p 6006 -c ./.storybo***ok

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

  2. **The User interface.**

STORYBOOK

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

STORYBOOK

- **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

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

STORYBOOK

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

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('boundry',
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

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.**

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

Aside – Some JS issues

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

```
const demo = <div>
                <h1>Something</h1>
                <h2>Something else</h2>
            </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>
          <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 tag.**

Component return value.

- **Old solution:**

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

- **Adds unnecessary depth to DOM → effects 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 <u>a hierarchy of components</u>.

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

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)**

```
// lib.js
export default function() {
  ...
}
// ------------------------
// bar.js
import myFunc from './lib';
myFunc();
```

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

```
// 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)
```

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

```
// OBJECTS
{
    // without destructuring
    let obj = { alpha: 100, beta: 'enterprise'}
    let alpha = obj.alpha
    let beta = obj.beta
}
{
    // with
    let obj = { alpha: 100, beta: 'enterprise'}
    let {alpha, beta} = obj;
    console.log(`${alpha} ${beta}`) // 100 enterprise
}
{
    // order is not important
    let obj = { alpha: 100, beta: 'enterprise'}
    let { beta, alpha } = obj;
    console.log(`${alpha} ${beta}`) // 100 enterprise
}
```

```
{
    // 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
}
```