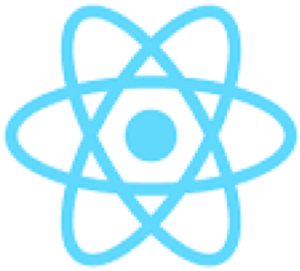


ReactJS.
Afa 21

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.

	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:**

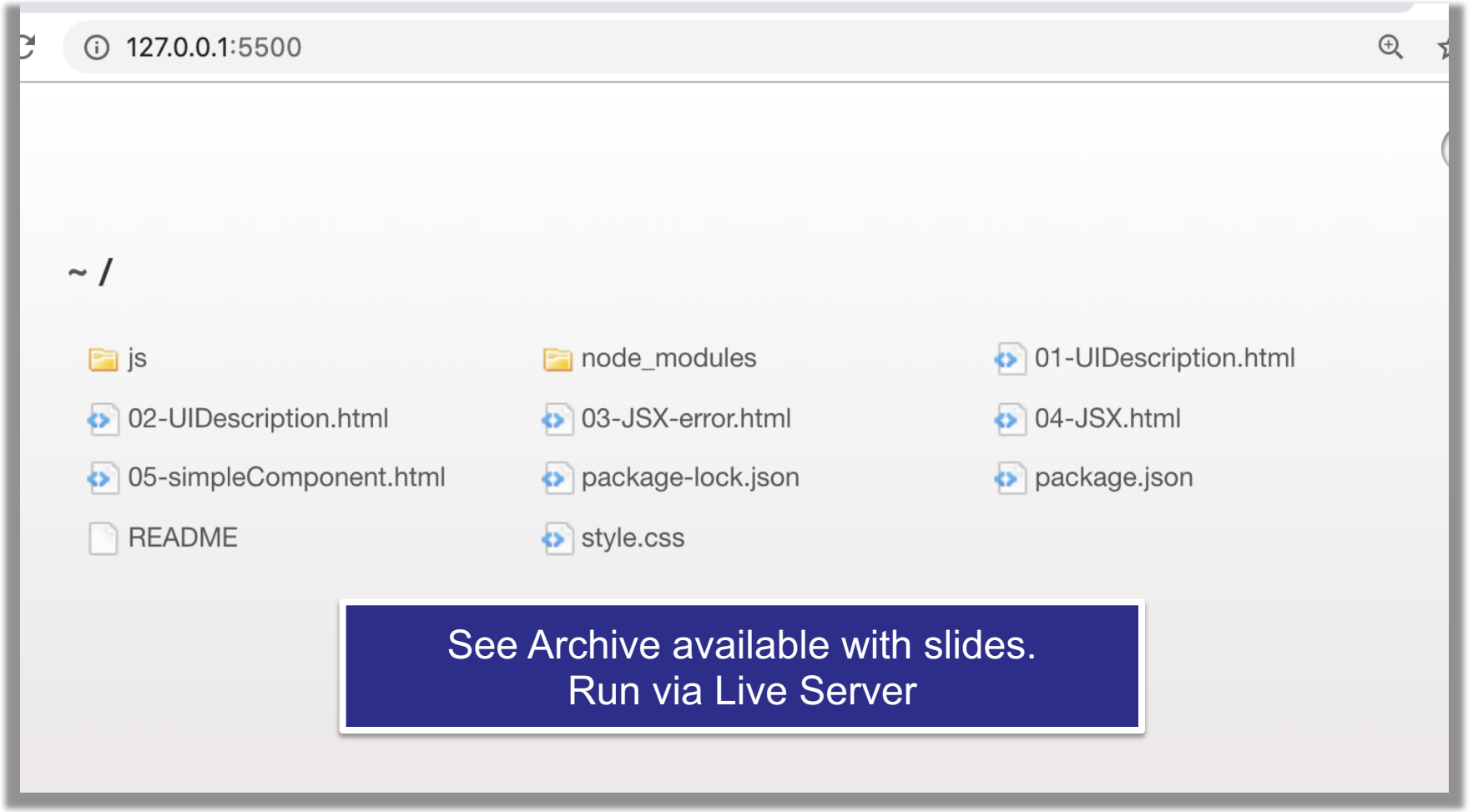
  UI description (HTML) + UI behavior (JS)
  - **Two aspects are tightly coupled and co-located.**
    - **Other frameworks decoupled them.**
  - **Benefits:**
    1. **Improved Composition.**
    2. **Greater Reusability.**
    3. **Better Performance.**

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.**
- 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


127.0.0.1:5500
~ /
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.

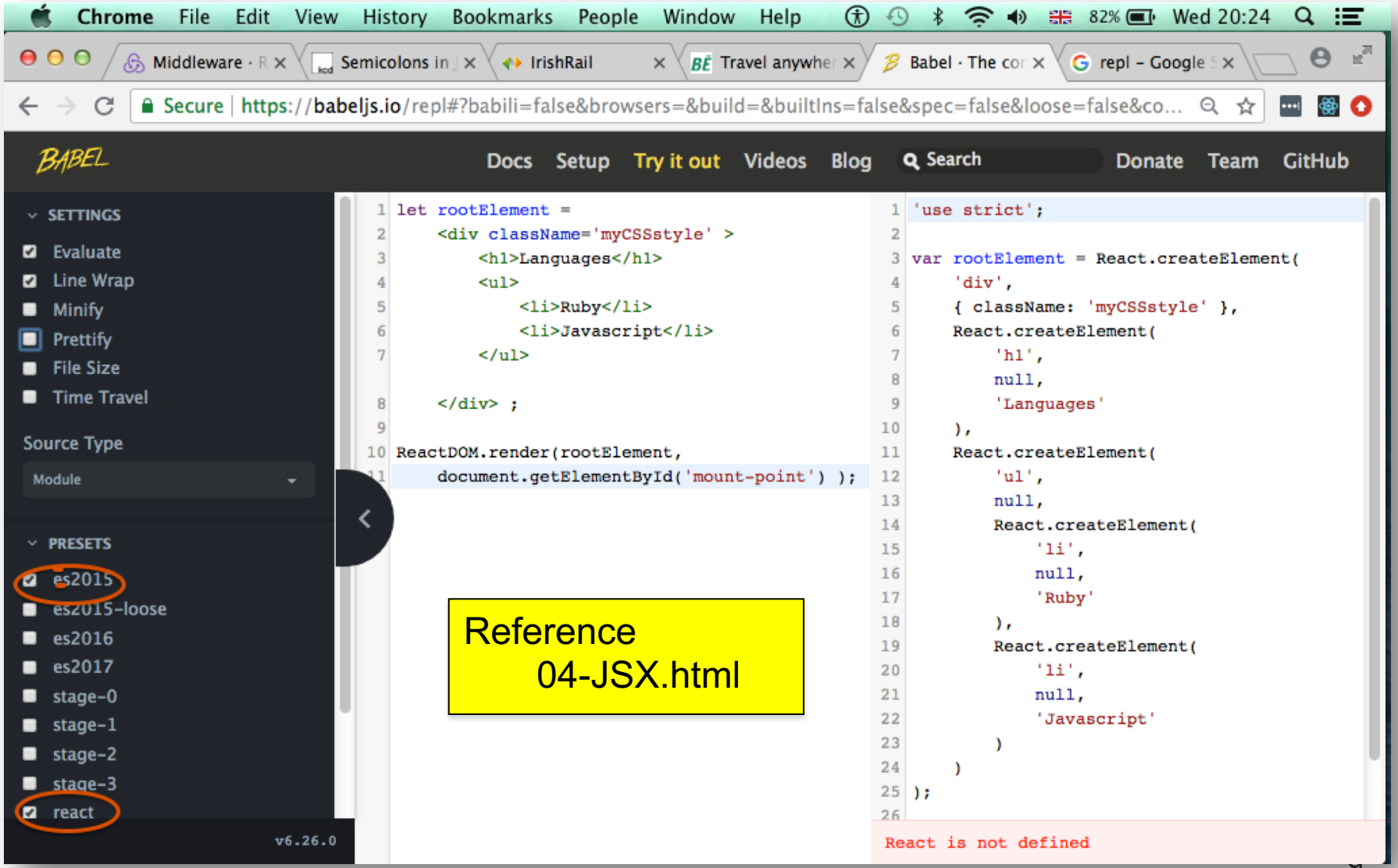
- **JSX – JavaScript extension syntax.**

- **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

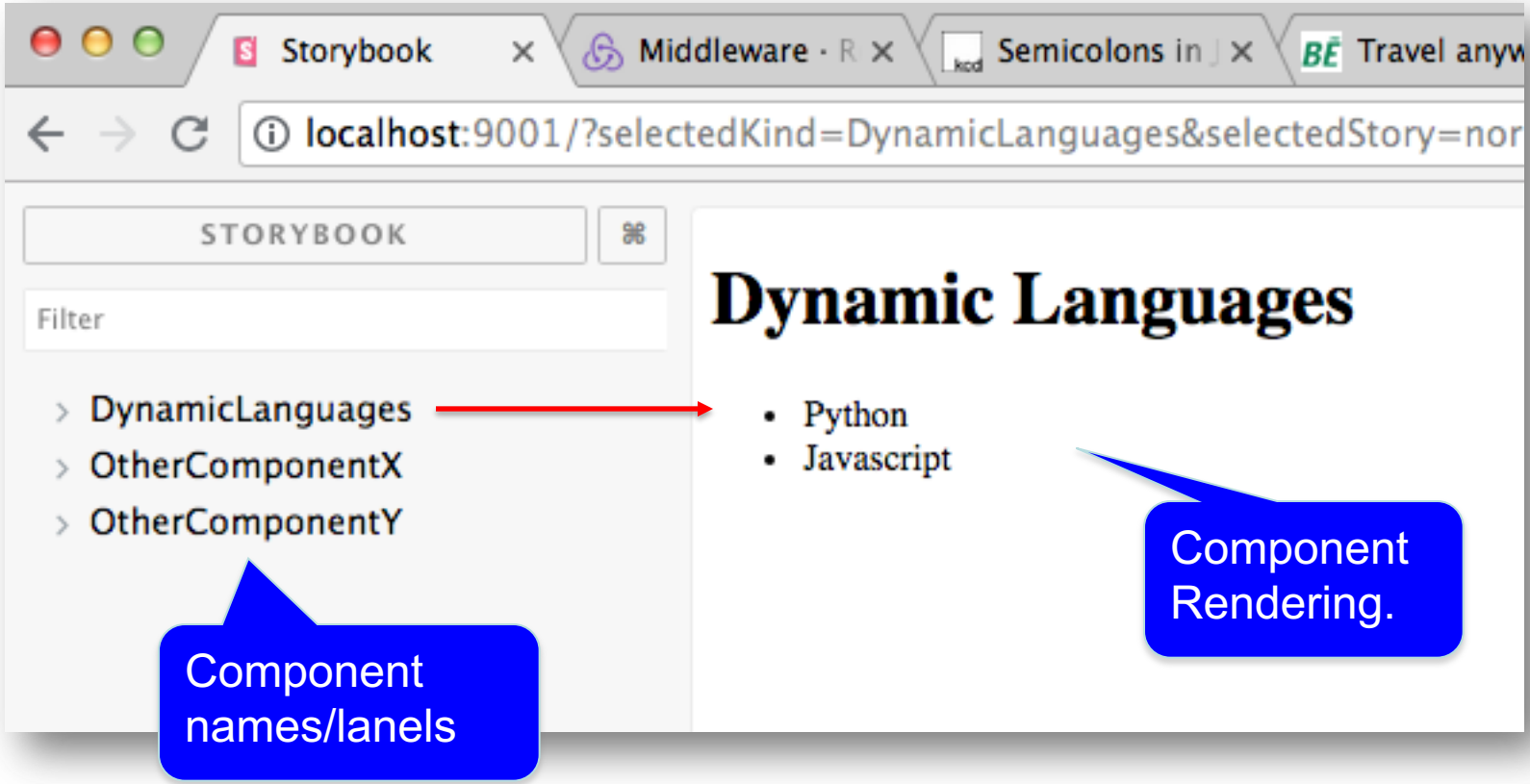
React Developer tools.

- 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

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



STORYBOOK

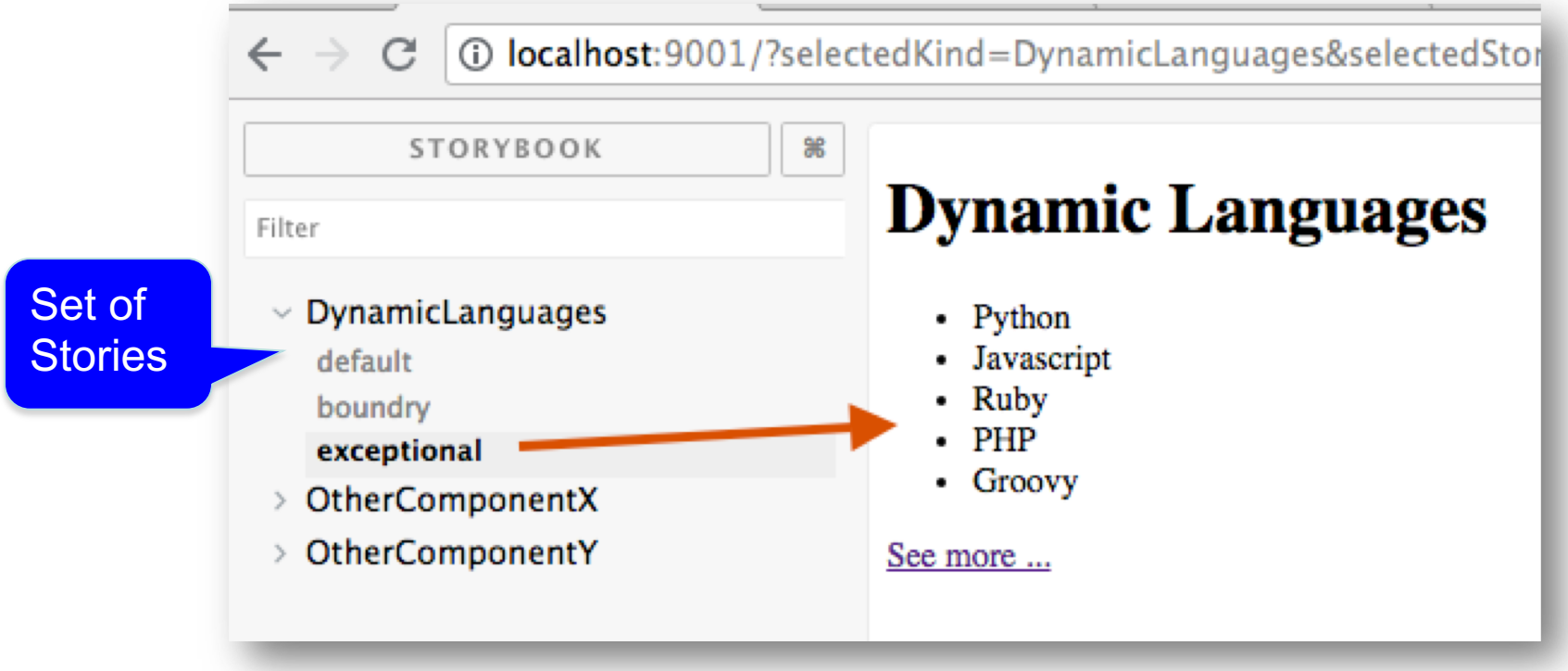
- **What is a Story?**

- **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.**

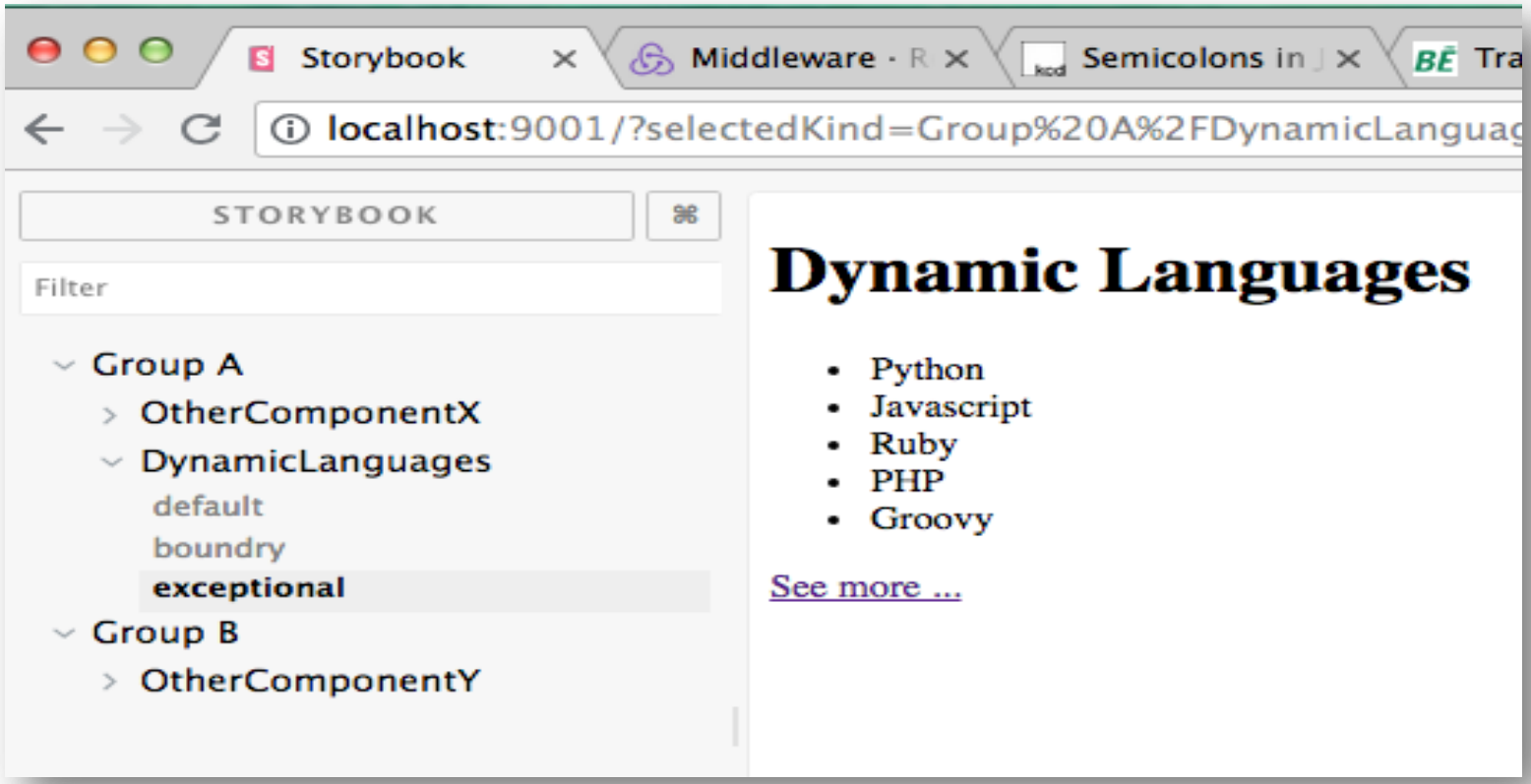
STORYBOOK

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



STORYBOOK

- **Can define component** groups f**or large component libraries**
  - **helps others understand the catalogue**.



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

Grouping stories.

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

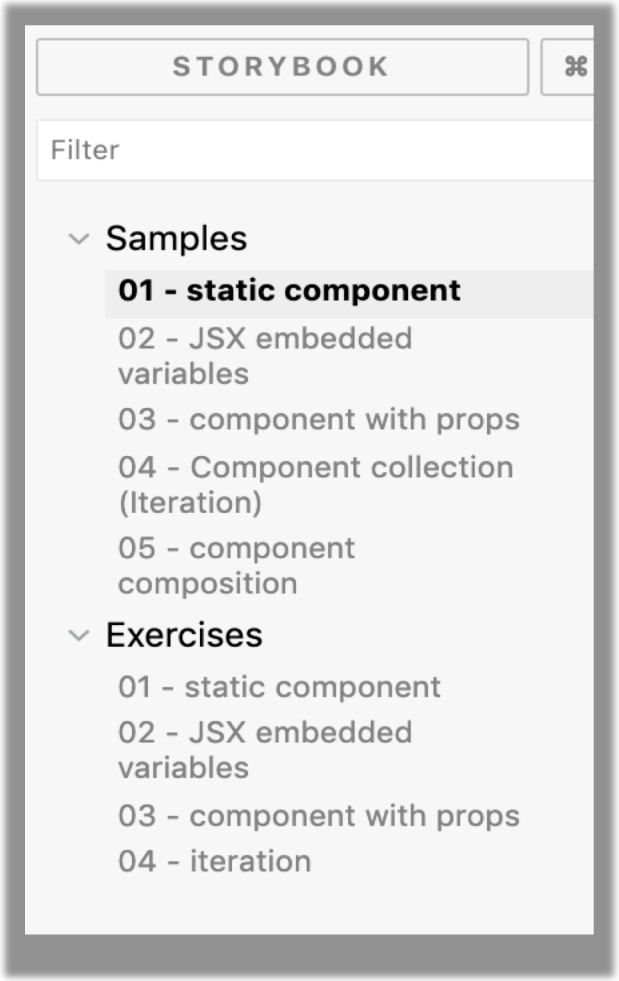
```
storiesof('Group A/Component 1')
    .add('…'), () => {……}
    .add('…'), () => {……}
storiesof('Group A/Component 2')
   .add('…'), () => {……}
   .add('…'), () => {……}
storiesof('Group B/Component X')
    .add('…'), () => {……}
   .add('…'), () => {……}
   .add('…'), () => {……}
```

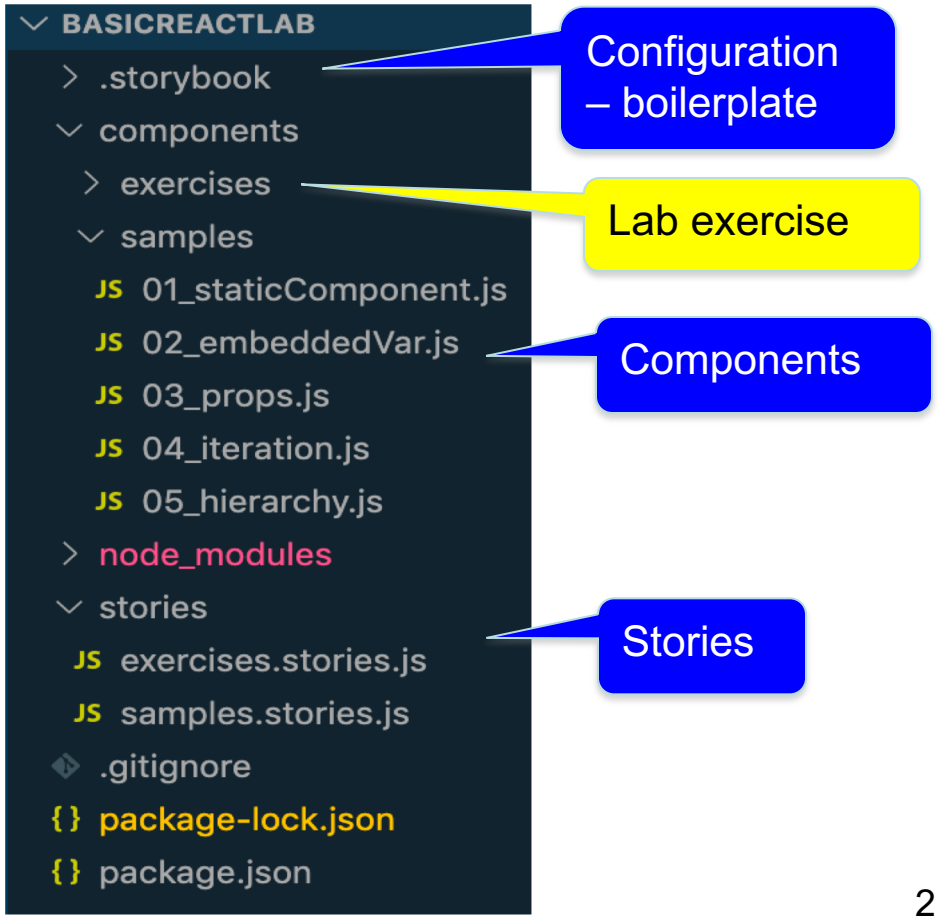
- **Lots of flexibility with grouping approach.**

… back to components . . .

Demo Samples

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





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

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>
    ) ;
```

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

- **New solution:**

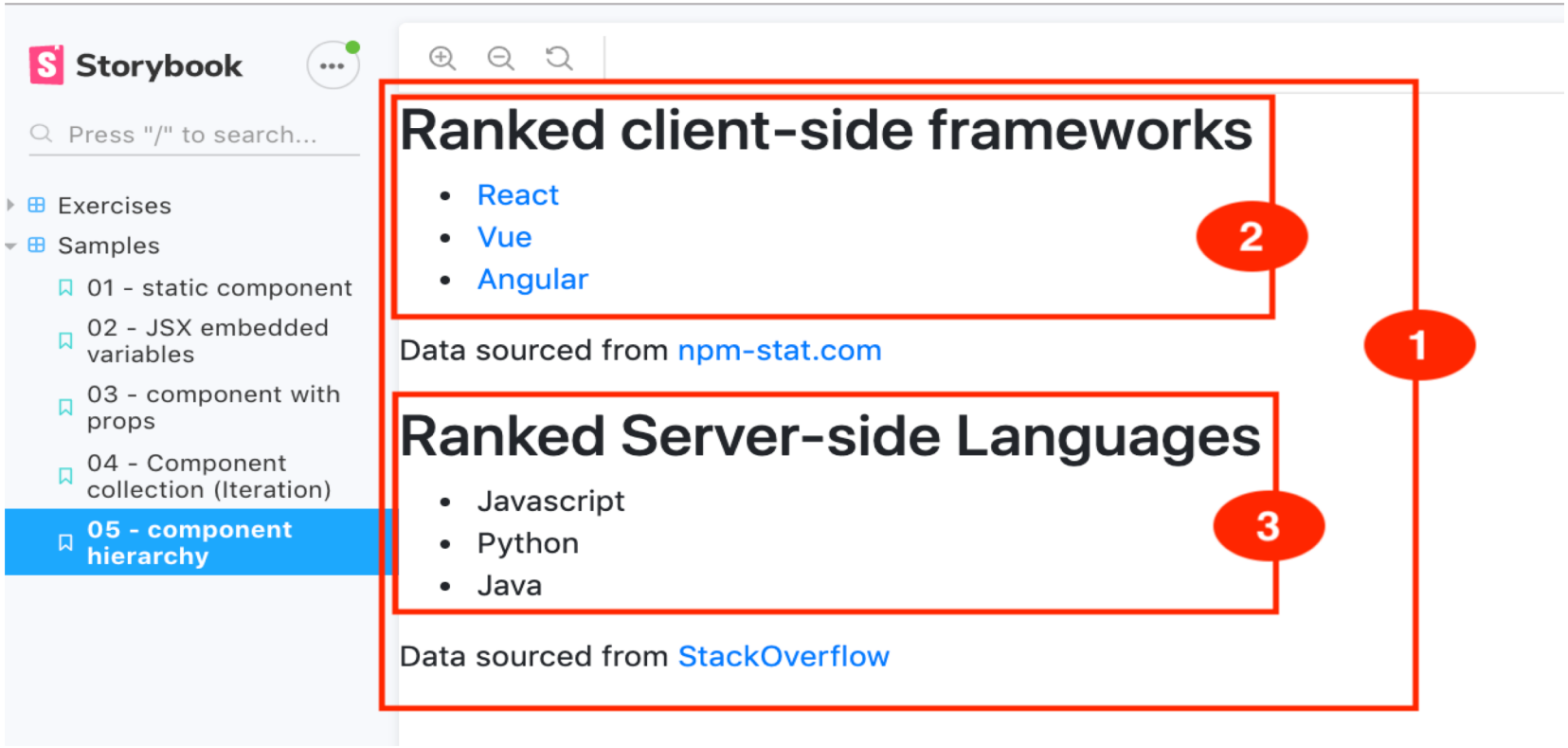
```
return (
    <>
      <h1>{this.props.type}</h1>
      <ul>
          . . . . . .
      </ul>
    </>
    ) ;
```

- **<> </> – 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.



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
```

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

```
// 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

```
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
}
```