React

The library for web and native user interfaces

Learn React API Reference

Introduction

Agenda

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

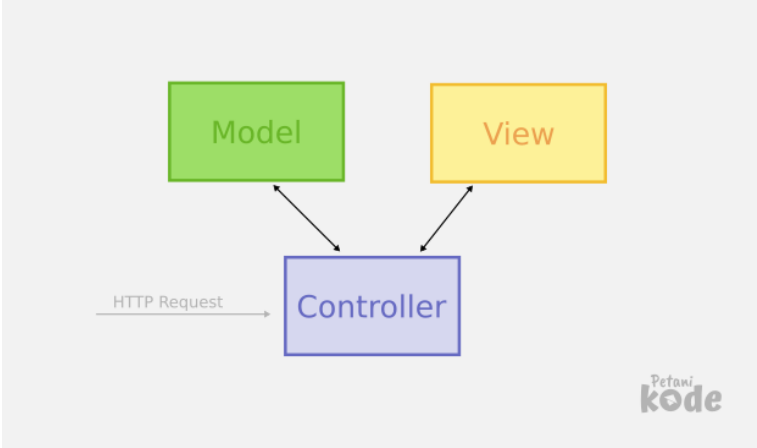
React

- **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.**

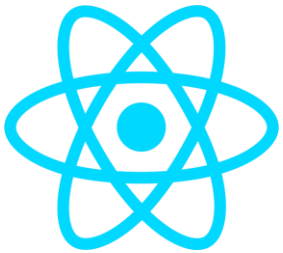
Before React

- 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”.**



This Photo by Unknown Author is licensed under CC BY

React 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 behaviour (JS)
  - **Two aspects are tightly coupled and co-located.**
    - Pre-React frameworks decoupled them.
  - **Benefits:**

    1. Improved Composition.

    2. Greater Reusability.

This Photo by Unknown Author is licensed under CC BY-NC-ND

Creating the UI Description.

(Vanilla React)

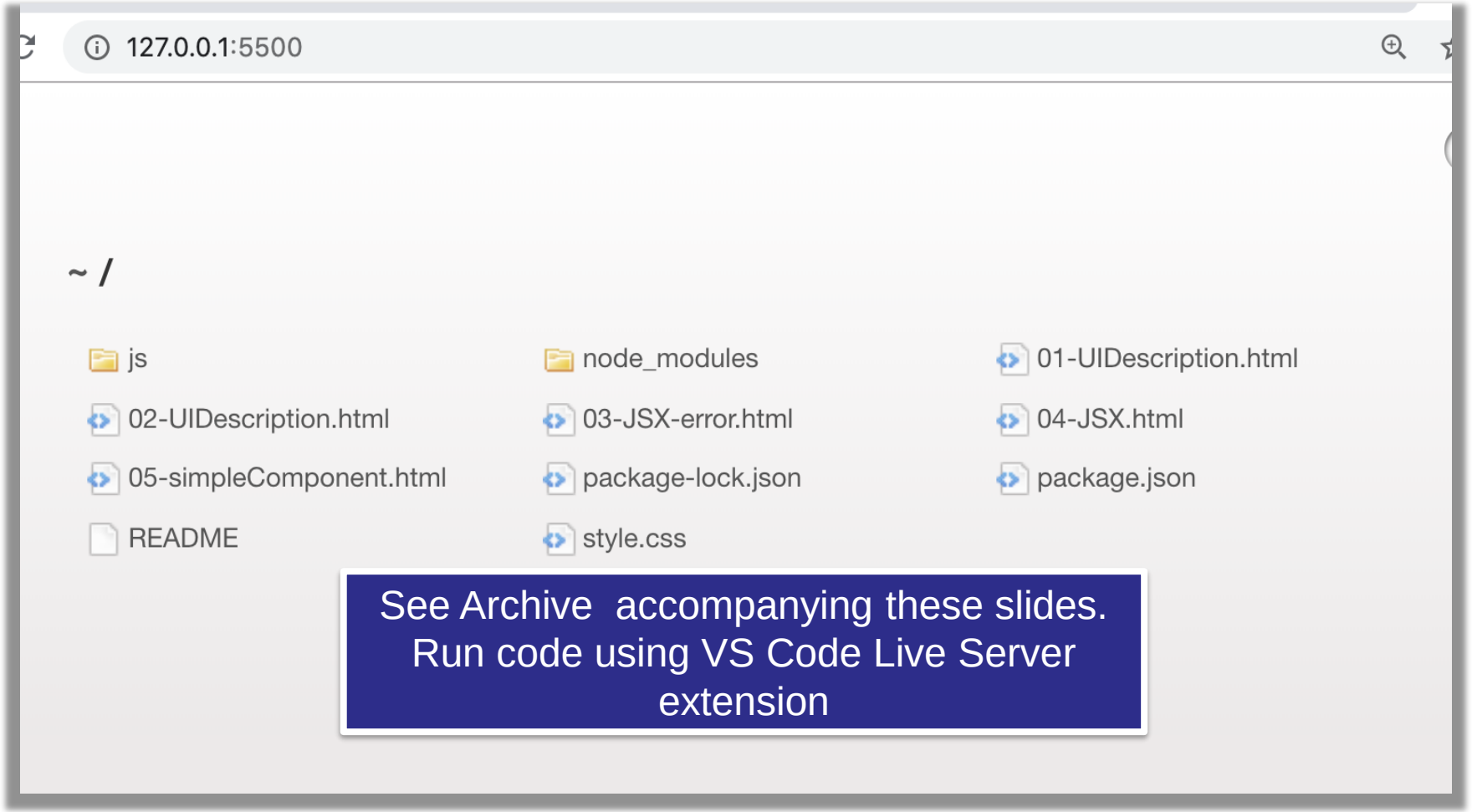
- React.createElement() **– create a HTML element.**
- ReactDOM.createRoot() – **which existing DOM node to attach the created element.**

- React.createElement() **arguments**:
  1. **type (h1, div, button etc).**
  2. **properties (style, event handler etc).**
  3. **children (0 -> M).**
  - **We don’t use createElement() directly – too cumbersome.**

- ReacrDOM.createRoot () **arguments:**
  1. **DOM node on which to** mount **a new element**.

Code Demos.

(See lecture archive)



TypeScript with React

- **Used to add type definitions to JavaScript codebases.**
- **TypeScript supports JSX (=>TSX)**
- **Include in your React project using @types/react and @types/react-dom**
- **Needs to be Transpiled to Javascript to run in Browser/Client.**

This Photo b Unknown Author is licensed under CC BY

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.

\+ *focuses on describing how a program operates, step by step.*

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

\+ *Focuses on what the result should be without specifying how it should achieve the results*

UI description implementation

(the declarative way)

- **TSX – TypeScript XML.**

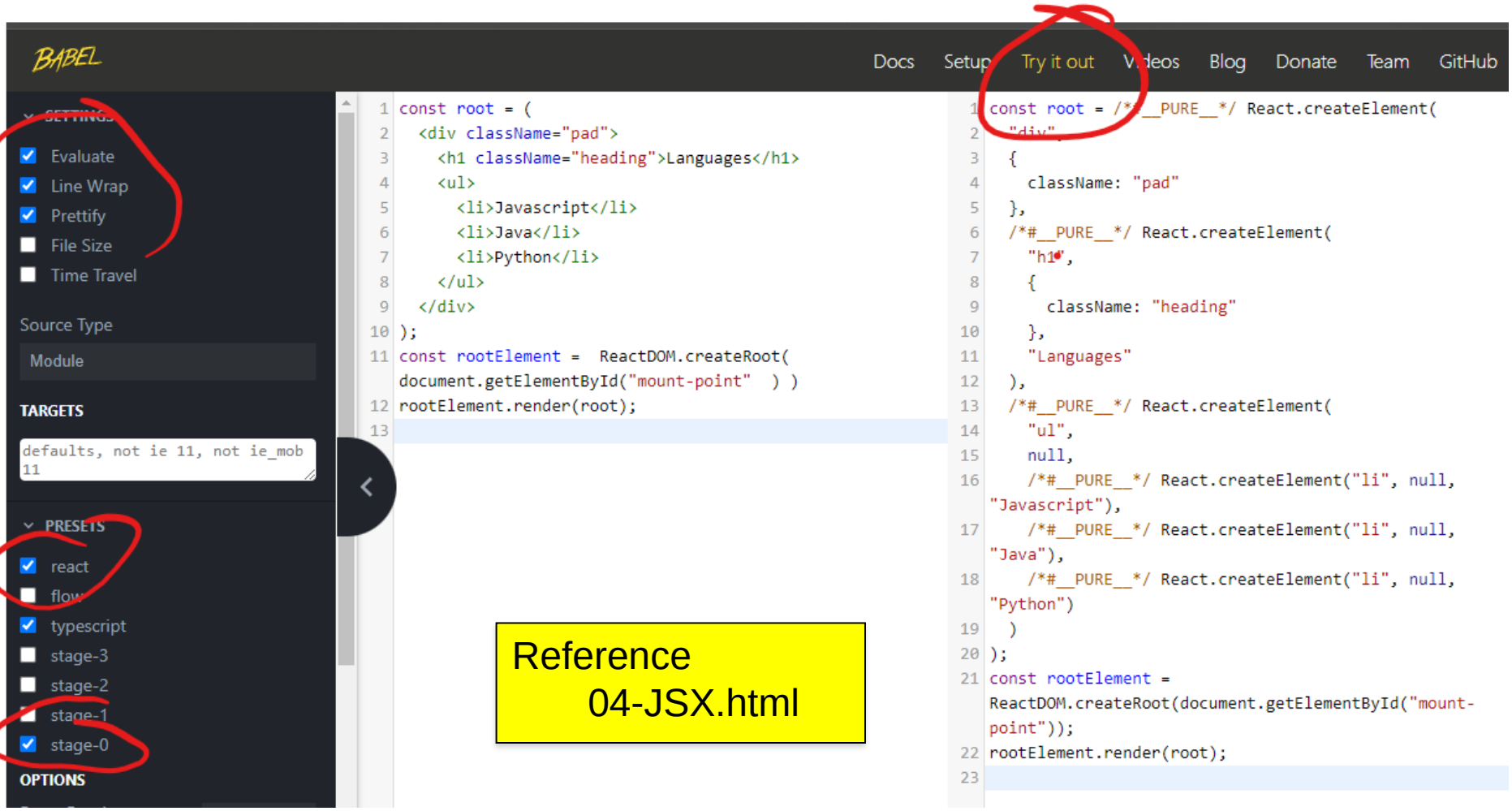
- Declarative syntax for coding UI descriptions.

- Retains the full power of Typescript.
- Allows tight coupling between UI behavior and UI description.

- **However, must be transpiled before being sent to browser.**

- **Reference** 03-JSX-error.html and 04-JSX.html

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



Reference
04-JSX.html

TSX.

- 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 TypeScript.
- **Combines the ease-of-use of templates with the power of TS.**

Transpiling TSX.

- **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 TS function that returns a UI description, i.e. TSX**.**

- **We reference a component like a <u>HTML tag.</u>**
  e.g.

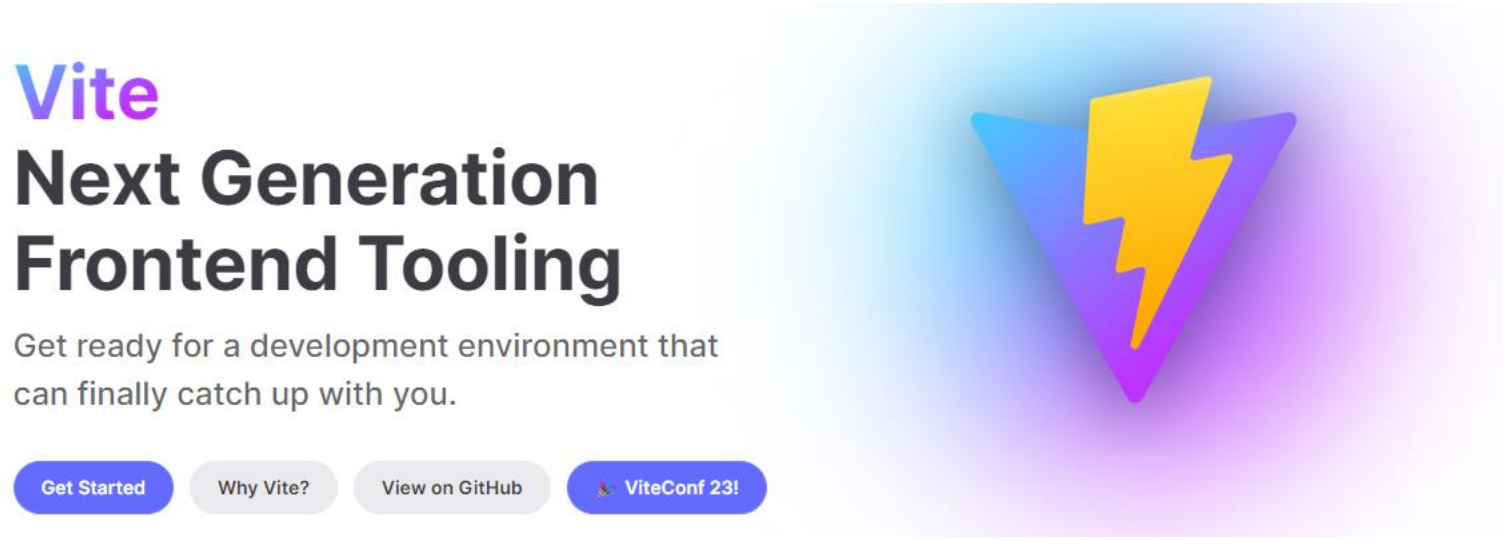
```
const rootElement =
    ReactDOM.createRoot(document.getElementById("mount-point"));
rootElement.render( <DynamicLanguages/> );
```

- **Reference** 05-simpleComponent.html

React Developer Tools - Vite

Features:

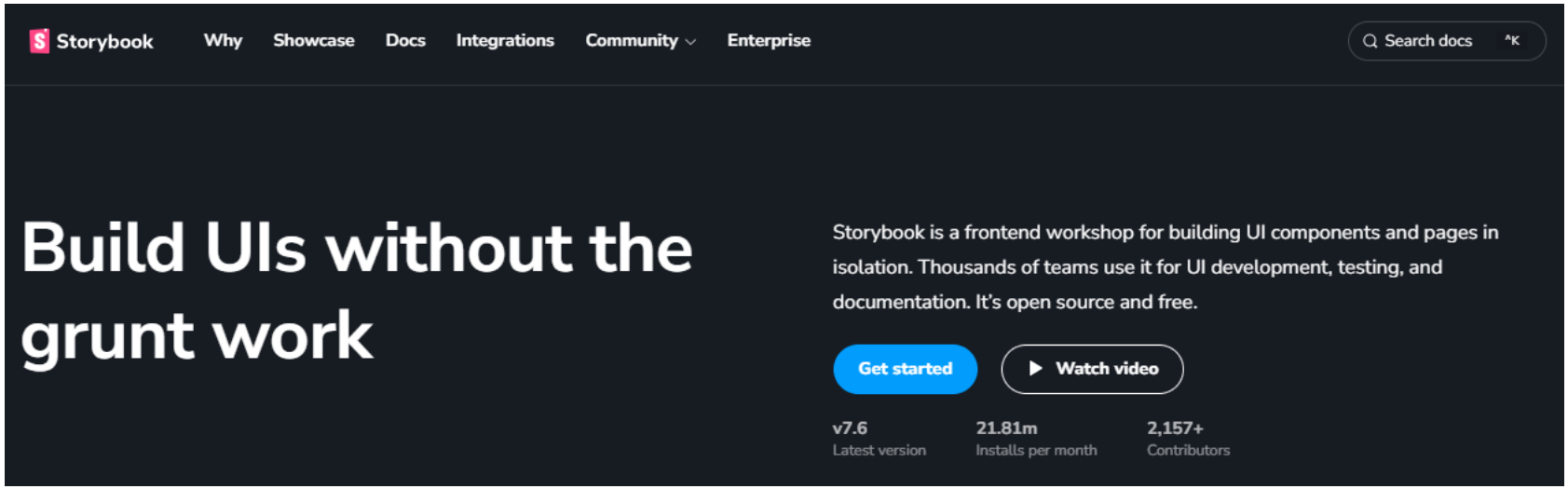
- Scaffolding/Generator.
- Development web server: auto-transpilation on file change + live reloading (HMR – Hot Module Replacement).
- Builder: build production standard version of app, i.e. minification, bundling.



React Developer Tools - 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.



Introduction to

Storybook

Storybook

- **Installation:**

  $ npm install @storybook/react

- **The tool has two aspects:**

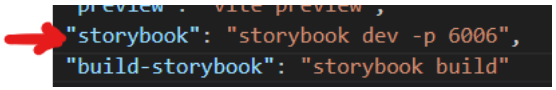
  1. **A web server.**

     $ `./node_modules/.bin/start-storybook -p 6006 -c ./.storybook`

     • Performs live re-transpilation and re-loading.

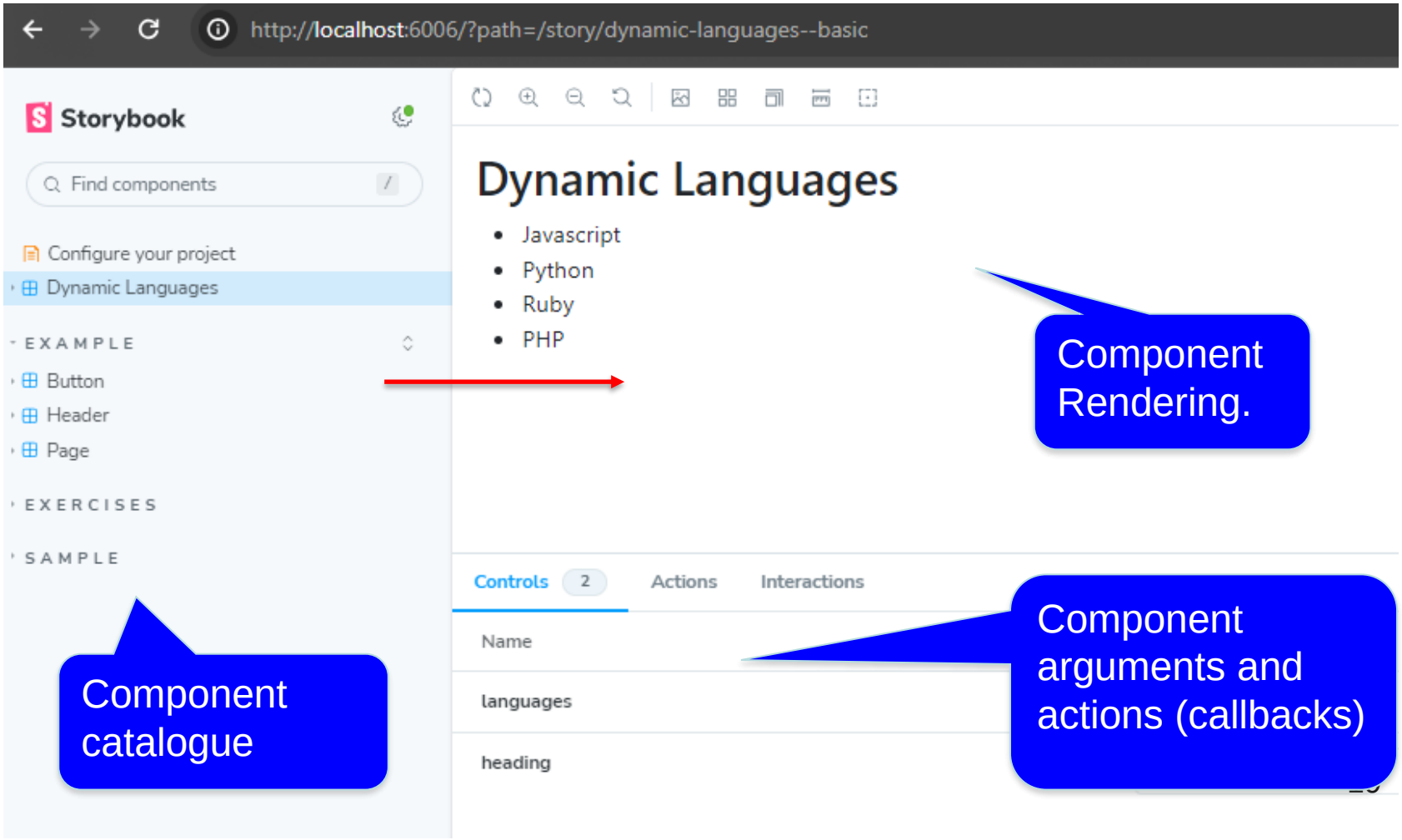
  2. **Web browser user interface.**

- Start up using package.json script



Storybook

- **Storybook User interface.**



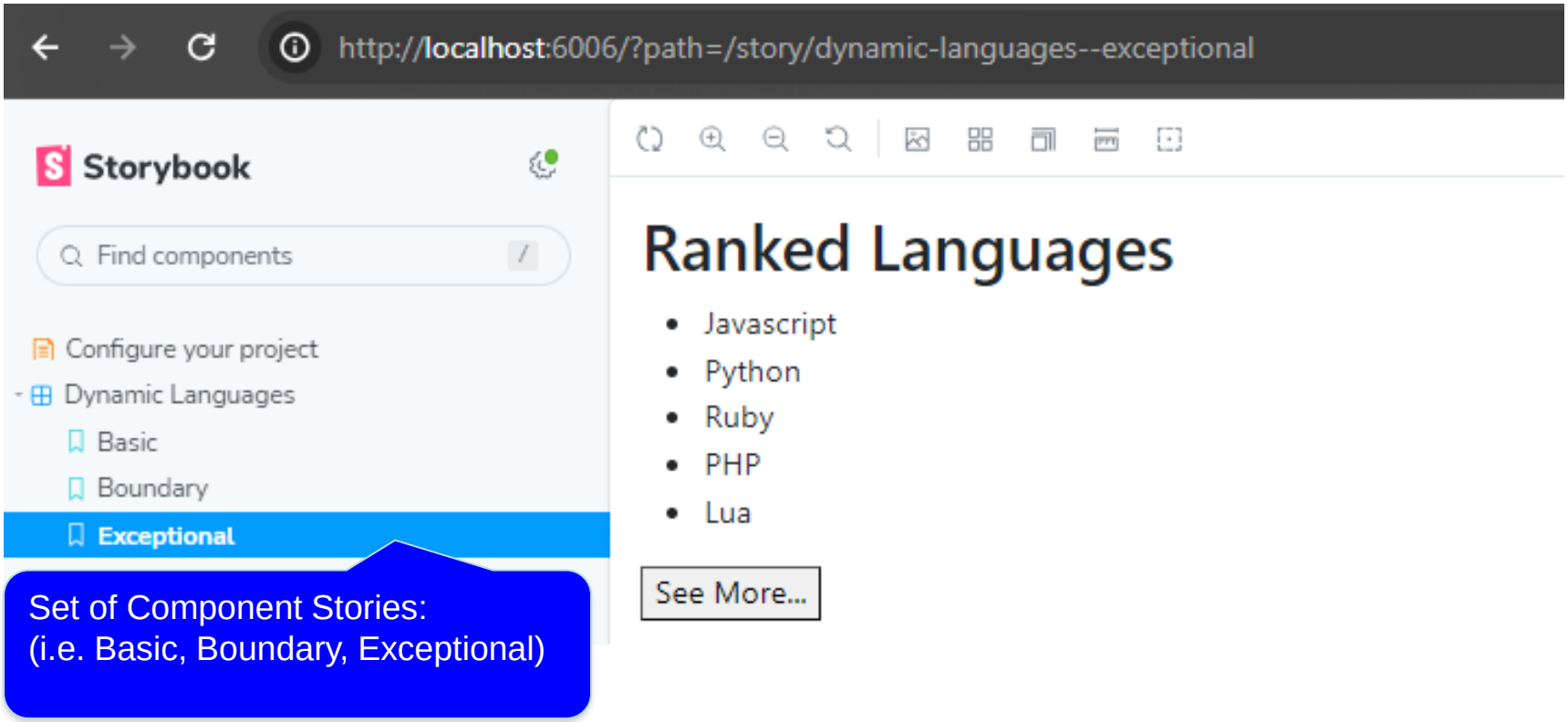
Storybook

What is a Story?

- A component may have several **STATES**
  - State affects how it renders.
- **Each state case termed a "STORY"**
- Example: 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.**
- Stories are a **design consideration** written in **Component Story Format (CSF)**
  - **They are functions that describes how to render components**

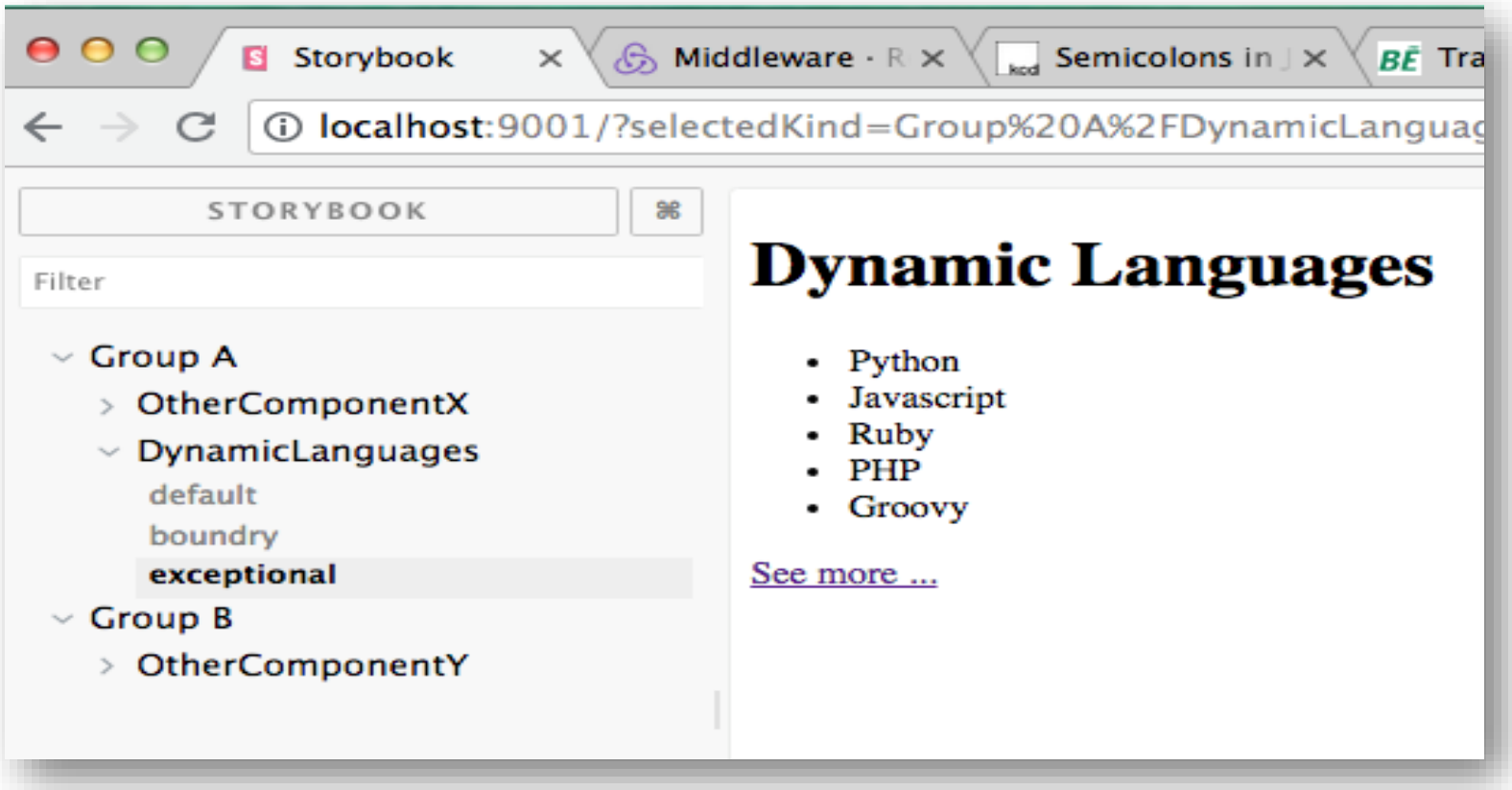
Storybook

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



Storybook

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



Writing stories

- .stories.ts **file extension (convention)**
- **1 Stories file per component**

```ts
import type { Meta, StoryObj } from '@storybook/react';
import DynamicLanguages from '../../components/samples/inclass_example';

const meta = {
  title: 'Dynamic Languages',
  component: DynamicLanguages,
} satisfies Meta<typeof DynamicLanguages>;

export default meta;
type Story = StoryObj<typeof meta>;

const list=["Javascript", "Python", "Ruby", "PHP"];
const emptyList: string[]=[];
const exceptionalList=[...list,  "Lua", "Perl", "Groovy",  "Lua", "Erlang", "Clojure"]

export const Basic: Story = {
  args:{
    languages: list,
    heading: "Dynamic"}
};

export const Boundary: Story = {···
};

export const Exceptional: Story = {···
};
```

default export; Metadata; How Storybook lists components.

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

Asside: The satisfies Operator

- TypeScript 4.9 introduces a new operator, **`satisfies`**
- Allows you to check that an expression matches a particular type.
- Used in Stories to ensure an object conforms to a type without casting

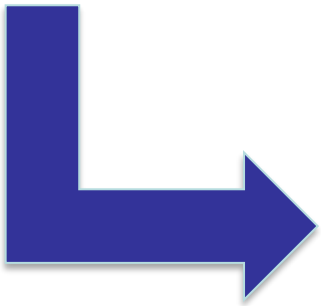
```
interface Person {
    name: string;
    age: number;
}

// This will compile successfully if the object satisfies the Person interface
let person = {
    name: "Alice",
    age: 30,
    occupation: "Developer"
} satisfies Person;
```

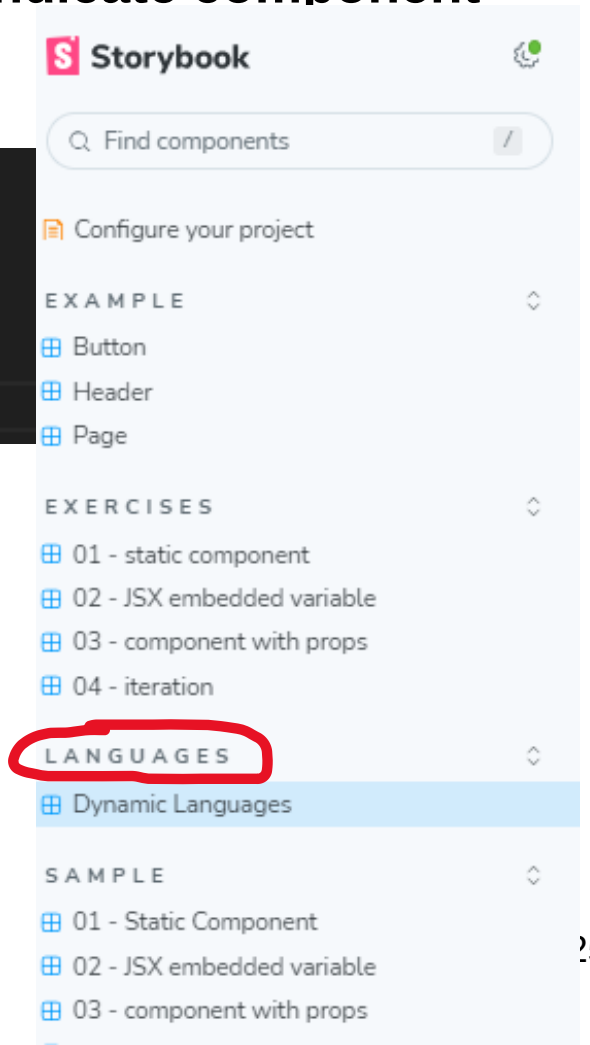
Would cause a compile error if age was missing

Grouping stories.

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

```
const meta = {
  title: 'Languages/ Dynamic Languages',
  component: DynamicLanguages,
} satisfies Meta;
```

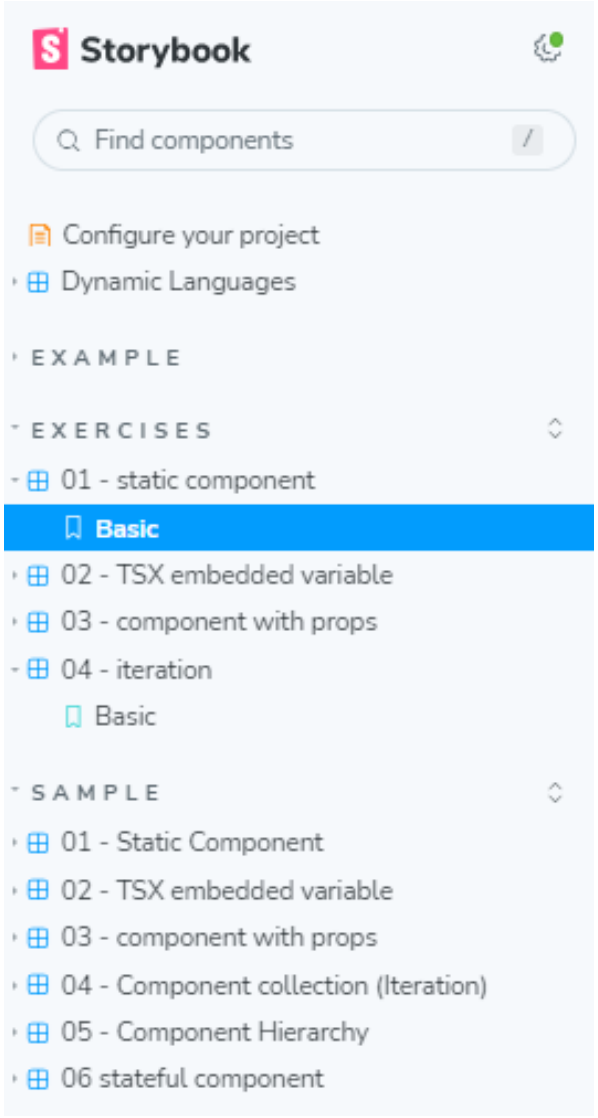


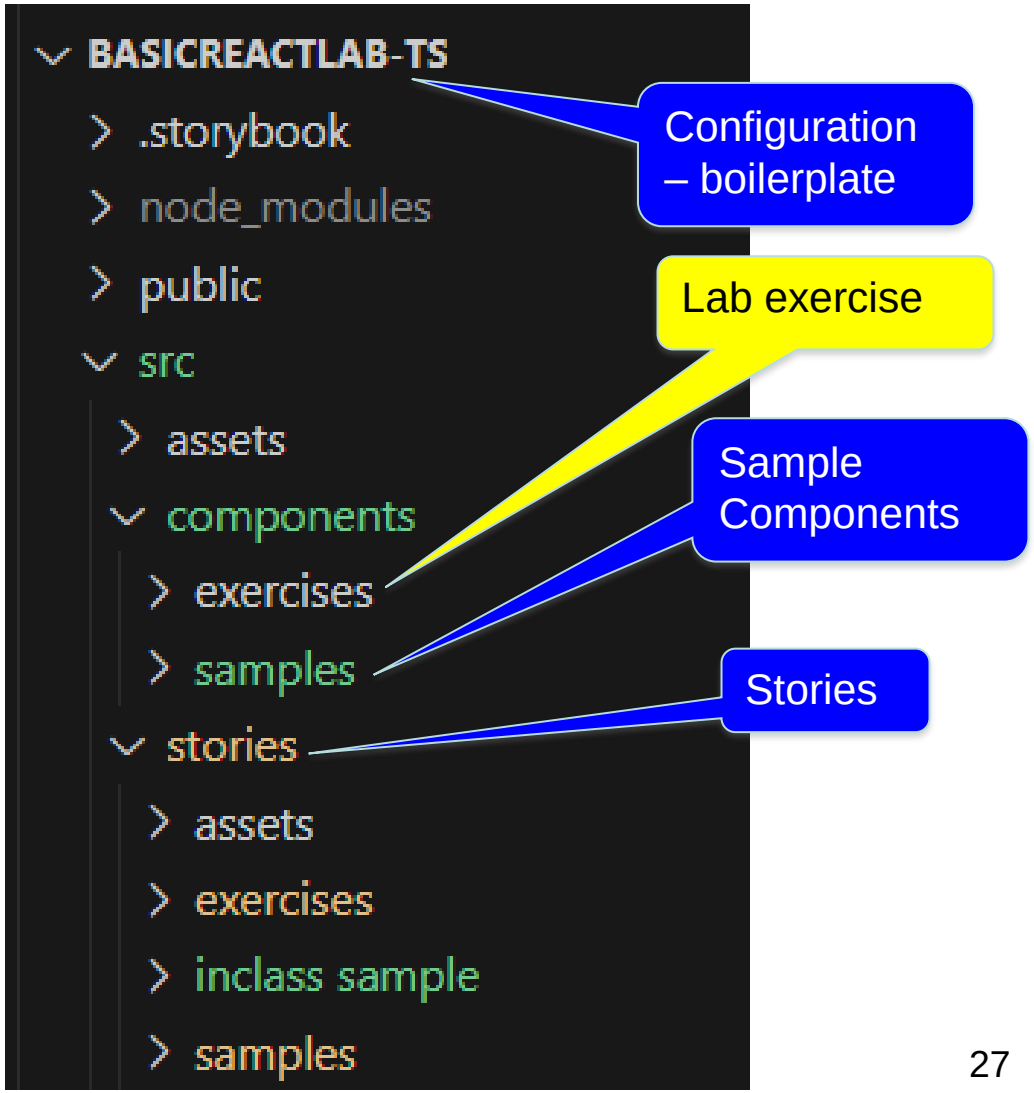


… back to components . . .

Demo Samples

(See lab exercise)





TSX - embedded variables.

- **Use { } to reference variable embedded in TSX.**
  - **Curly braces can contain any valid TS expression.**
- **Reference** src/components/samples/02_embeddedVariables.tsx

```tsx
import React from "react";

const Demo: React.FC = () => {
    const languages: string[] = ["Go", "Julia", "Kotlin"];
    const header: string = "Modern";

    return (
        <div>
            <h1>{`${header} Languages`}</h1>
            <ul>
                {languages.map(language => (
                    <li key={language}>{language}</li>
                ))}
            </ul>
        </div>
    );
};

export default Demo;
```

Reusability.

- We achieve reusability through **parameterisation.**
- **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 React.FC<PropInterface> = (props) => {
  const p1 = props.prop1Name
    ........
```

  3. Props are Immutable.
  4. Part of a component's design.
- Reference **src/components/samples/03_props.tsx (**and related story).

Aside.

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

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

```tsx
const Demo: React.FC<Frameworks>  = (props) => {
  const list = props.frameworks.map((f, index) =>
    <li key={index}>
      <a href={f.url}> {f.name} </a>
    </li>
  );
  return (
    <>
      <h2>{props.type}</h2>
      <ul>{list}</ul>
    </>
  );
}

export de
```

map used to to create a new array based on the frameworks array passed in through the `props` object.

```
▼<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 tag.**

Component return value.

- **Old solution:**

```
return (
    <div>
        <h1> ……</h1>
        <MyComponent …… />
        <p> …….. </p>
     </div>
    ) ;
```

- **Adds unnecessary depth to DOM → affects performance.**

- **Alternative solution:**

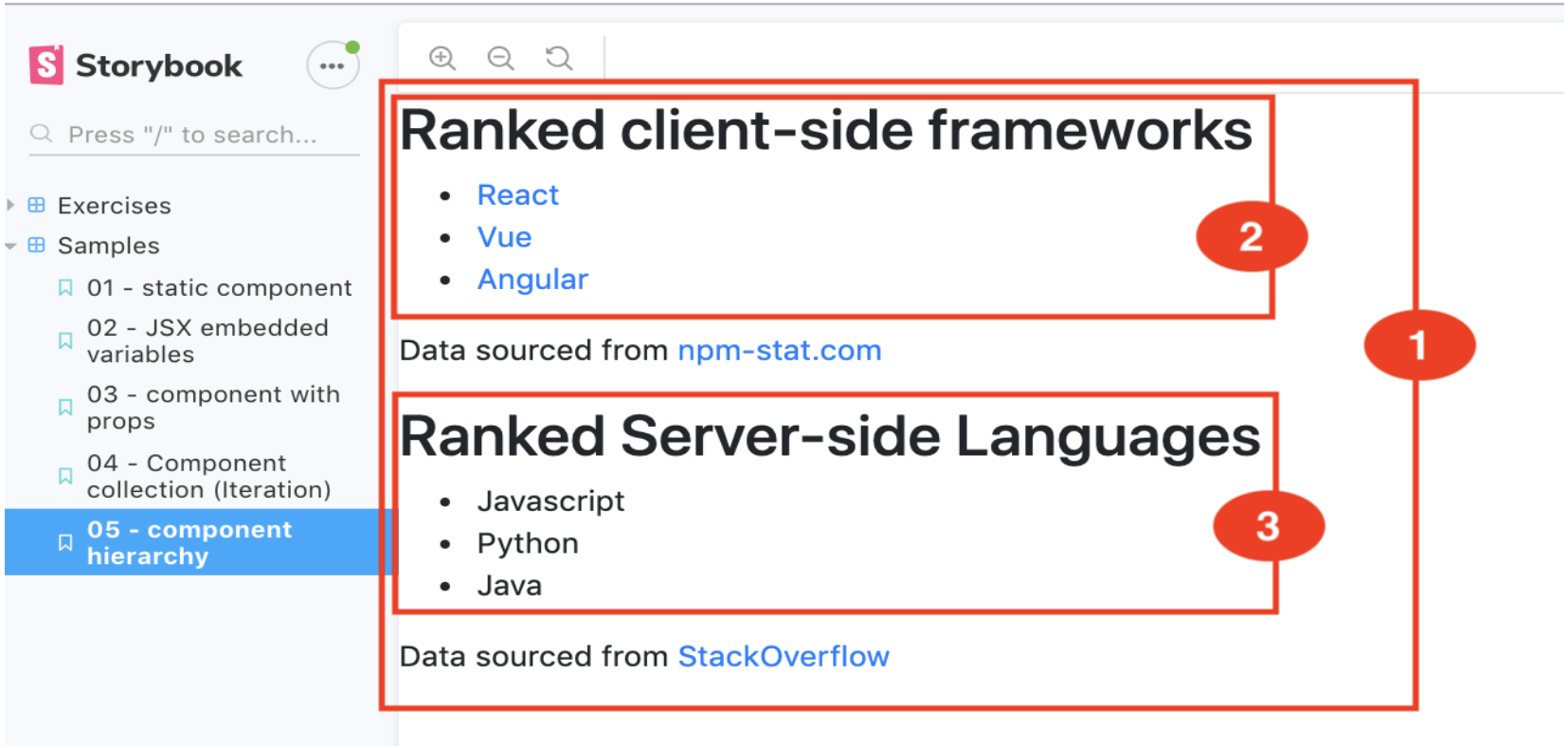
```
return (
    <>
        <h1> ……</h1>
        <MyComponent …… />
        <p> …….. </p>
     </>
    ) ;
```

- **<> </> – 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.ts.



Summary.

- **TSX.**
  - **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 TSX element .**
  - **Components can be nested.**
- **Storybook tool.**
  - **Develop components in isolation.**
  - **Story – the state (data values) of a component can affect its rendering (and behaviour).**

Material UI.

A 3rd party component library to build high quality digital UIs

Material Design.

- Material (Design) is a design system created by Google to help teams build high-quality digital experiences for Android, iOS, and web.
- A visual language that synthesizes classic principles of good design with the innovation and possibility of technology and science.
- Inspired by**:**
  - the physical world and its textures, including how they reflect light and cast shadows.
  - the study of paper and ink.
- Material is a metaphor.
  - Material surfaces reimagine the mediums of paper and ink.

Material Components.

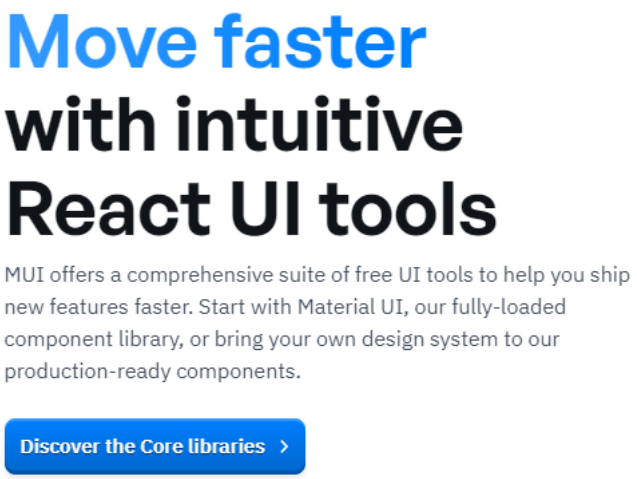
- Material Components **are interactive building blocks for creating a digital user interface.**
- They cover a range of interface needs, including:
  1. **Display:** Placing and organising content using components like cards, lists, and grids.
  2. **Navigation:** Allowing users to move through an application using components like navigation drawers and tabs.
  3. **Actions:** Allowing users to perform tasks using components such as the floating action button.
  4. **Input:** Enter information or make selections using components like text fields and selection controls.
  5. **Communication:** Alerting users to key information and messages using snackbars, banners and dialogues.

Theming.

- **Material Design does not mean copy Google design.**

- Material Theming makes it easy to customize Material Design to match the look and feel of your brand, with built-in support and guidance for customizing colors, typography styles, and corner shape.
- **Colour** - Material's colour system is an organised approach to applying colour to a UI. Global colour styles have semantic names and defined usage in components – primary, secondary.
- **Typography** - The Material type system provides 13 typography styles for everything from headlines to body text and captions.
  - Each style has a clear meaning and intended application within an interface.

Material UI



- MUI is a React component library based on the Material Design system.
- Its React components include: `<Card />`, `<Box />`, `<Grid />`, `<Menu />`, `<Button />`, `<Icon />`, `<Snackbar />`, `<Typography />` ……
- Build your own design system or start with Material Design.
- The CSS-in-JS model.

CSS-in-JS

- **Plain CSS**

```
.my-header {
   background-color: lightblue;
   padding: 10px;
}
```

- **----------**

```
import ‘app.css’

<header
     className=”my-header">
         ……..
</header>
```

- **CSS-in-JS**

```
const myHeader = {
        backgroundColor: "lightblue",
        padding: "10px”
     };
  . . . . . .
<header style={myHeader}>
                  …….. </header>
```

Must be CamelCase

Many Alternatives

- **Libraries:**
  - **Tailwind**
  - **Chakera**
  - **React-Bootstrap**

- **Design Systems:**
  - **ANT**

- **CSS models:**
  - **CSS modules**
  - **BEM**