



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.**



Why React?



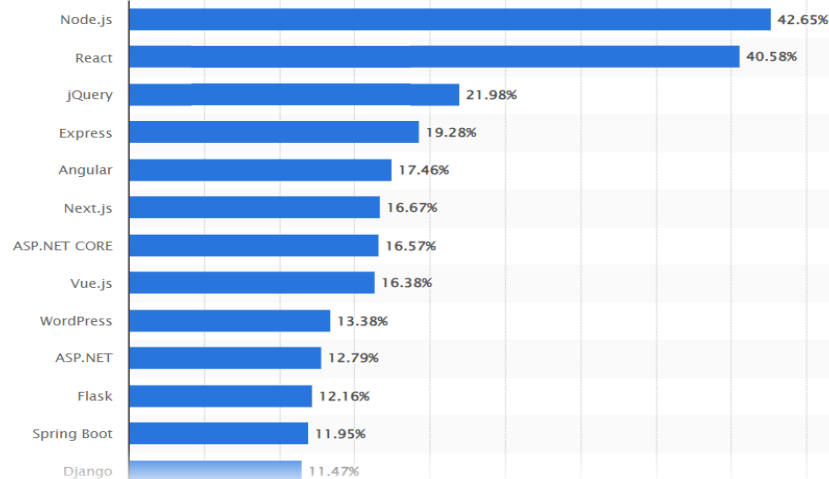
ChatGPT

As of 2024, the most popular framework in the industry, particularly in the realm of web application development, appears to be React.js. While technically a JavaScript library rather than a framework, React.js is extensively used for building interactive user interfaces, especially for single-page applications. Its popularity stems from several factors:



Technology & Telecommunications › Software

Most used web frameworks among developers worldwide, as of 2023



DOWNLOAD



Source

- Show source
- Show publication
- Use Ask S

Release date

June 2023

Region

Worldwide

Survey time

May 8, 2023

Number of

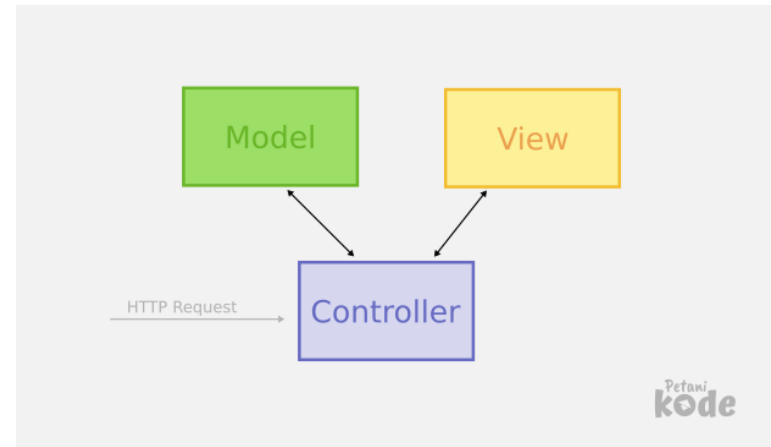
71,802 responses

Special pro

Software dev

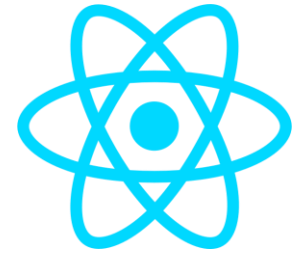
Before React

- 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 focused on 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.

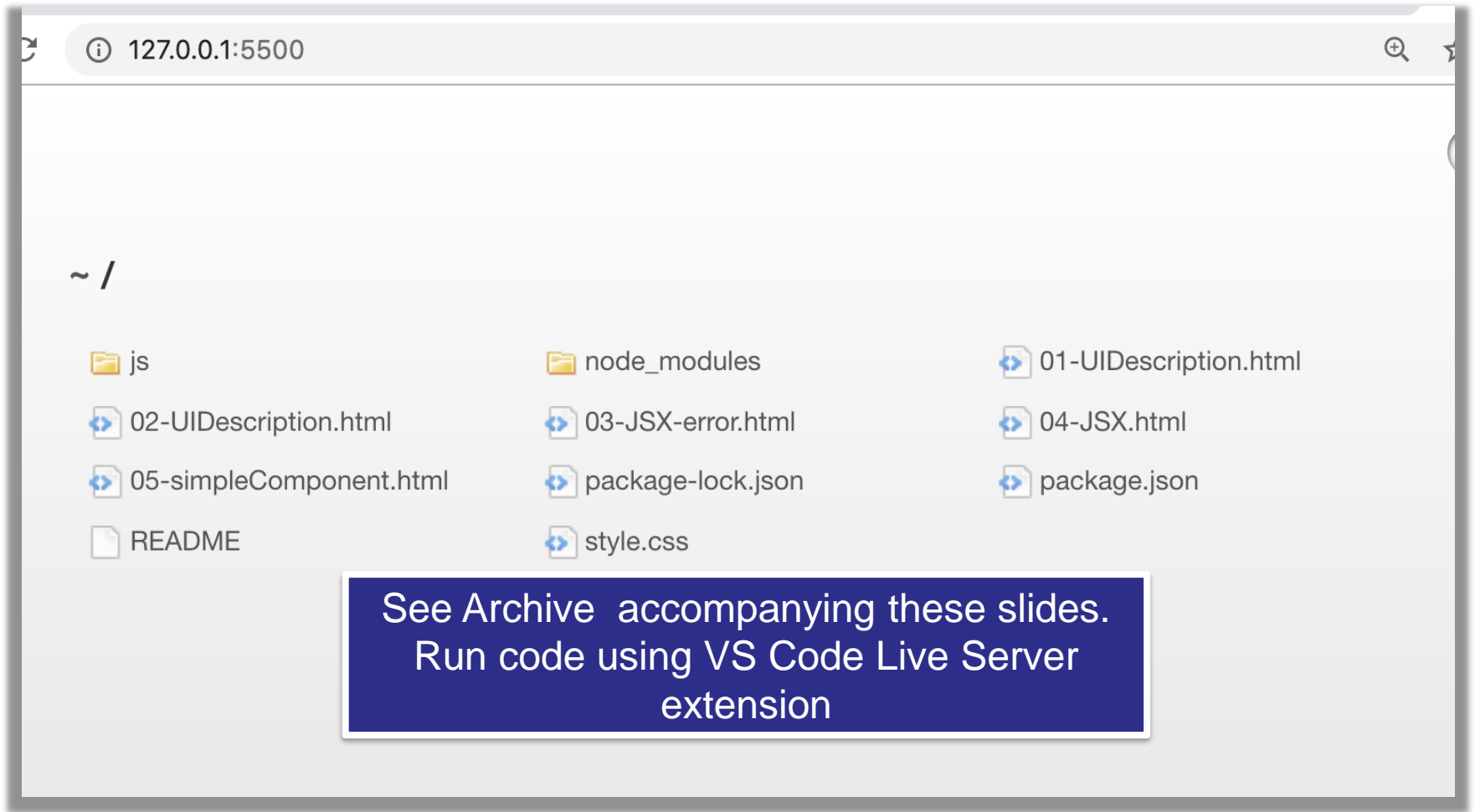
Creating the UI Description.

(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.**
- `ReactDOM.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](#) by 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.

BABEL

Docs Setup **Try it out** Videos Blog Donate Team GitHub

SETTINGS

- ☒ Evaluate
- ☒ Line Wrap
- ☒ Prettify
- ☐ File Size
- ☐ Time Travel

Source Type

Module

TARGETS

defaults, not ie 11, not ie_mob 11

PRESETS

- ☒ react
- ☐ flow
- ☒ typescript
- ☐ stage-3
- ☐ stage-2
- ☐ stage-1
- ☒ stage-0

OPTIONS

```
1 const root = (  
2   <div className="pad">  
3     <h1 className="heading">Languages</h1>  
4     <ul>  
5       <li>Javascript</li>  
6       <li>Java</li>  
7       <li>Python</li>  
8     </ul>  
9   </div>  
10 );  
11 const rootElement = ReactDOM.createRoot(  
12   document.getElementById("mount-point" ) )  
13   rootElement.render(root);  
14
```

Reference
04-JSX.html

```
1 const root = /*#__PURE__*/ React.createElement(  
2   "div",  
3   {  
4     className: "pad"  
5   },  
6   /*#__PURE__*/ React.createElement(  
7     "h1",  
8     {  
9       className: "heading"  
10    },  
11    "Languages"  
12  ),  
13  /*#__PURE__*/ React.createElement(  
14    "ul",  
15    null,  
16    /*#__PURE__*/ React.createElement("li", null,  
17      "Javascript"),  
18    /*#__PURE__*/ React.createElement("li", null,  
19      "Java"),  
20    /*#__PURE__*/ React.createElement("li", null,  
21      "Python")  
22  )  
23 );  
24 const rootElement =  
25 ReactDOM.createRoot(document.getElementById("mount-  
26 point"));  
27 rootElement.render(root);  
28
```

TSX.

- TypeScript XML
- 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 templating with the power of TS.**



Transpiling TSX.

- **Browsers don't do Typescript!**
 - Needs to be Transpiled to Javascript
- **What can we use to Transpile?**
 - 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 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.

Vite

Next Generation Frontend Tooling

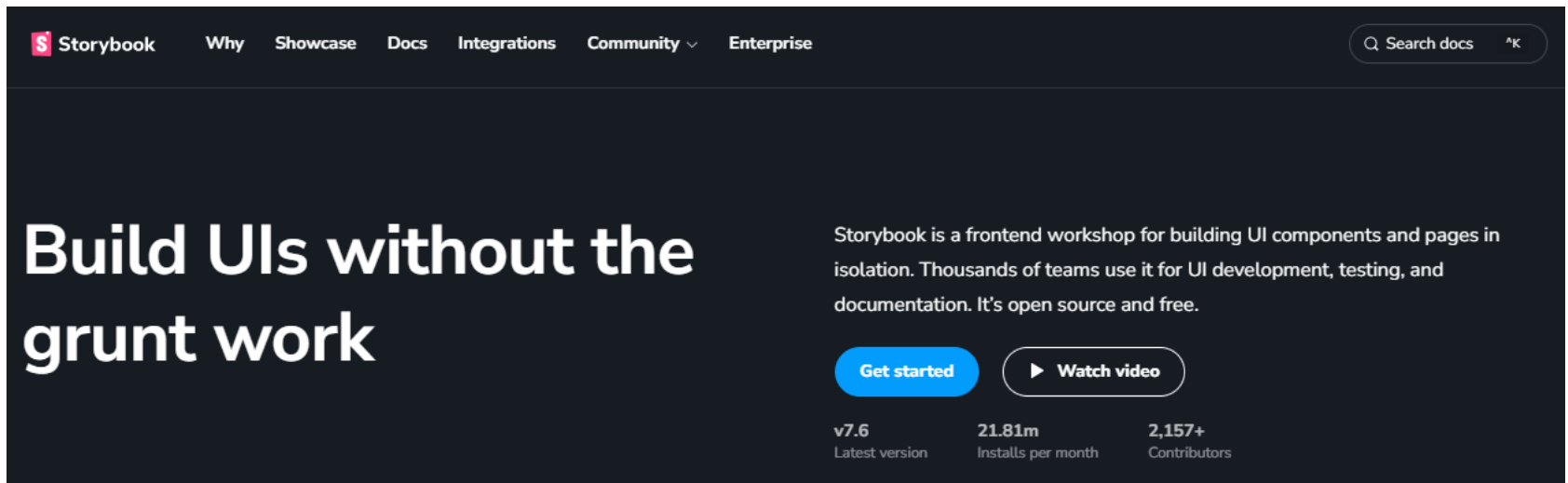
Get ready for a development environment that can finally catch up with you.

[Get Started](#)[Why Vite?](#)[View on GitHub](#)[ViteConf 23!](#)

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

```
    "preview": "vite preview",  
    "storybook": "storybook dev -p 6006",  
    "build-storybook": "storybook build"
```



Storybook

- Storybook User interface.

The screenshot shows the Storybook web application running in a browser. The address bar displays the URL `http://localhost:6006/?path=/story/dynamic-languages--basic`. The interface is divided into three main sections:

- Left Sidebar (Component Catalogue):** Contains the Storybook logo, a search bar labeled "Find components", and a list of components. The "Dynamic Languages" component is highlighted in blue. Below it are sections for "EXAMPLE" (with sub-items: Button, Header, Page) and "EXERCISES".
- Top Bar:** Contains navigation icons (back, forward, search, etc.) and a toolbar with icons for zooming and other actions.
- Main Content Area:** Displays the "Dynamic Languages" component. It features a list of languages: Javascript, Python, Ruby, and PHP. Below this list is a table with columns "Controls", "Actions", and "Interactions". The "Controls" column is active, showing a table with rows for "Name", "languages", and "heading".

Three blue callout boxes with white text provide additional context:

- A red arrow points from the "Dynamic Languages" component in the sidebar to the "Dynamic Languages" component in the main content area.
- A blue callout box labeled "Component Rendering." points to the "Dynamic Languages" component in the main content area.
- A blue callout box labeled "Component arguments and actions (callbacks)" points to the "languages" row in the "Controls" table.

Controls	2	Actions	Interactions
Name			
languages			
heading			



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**



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

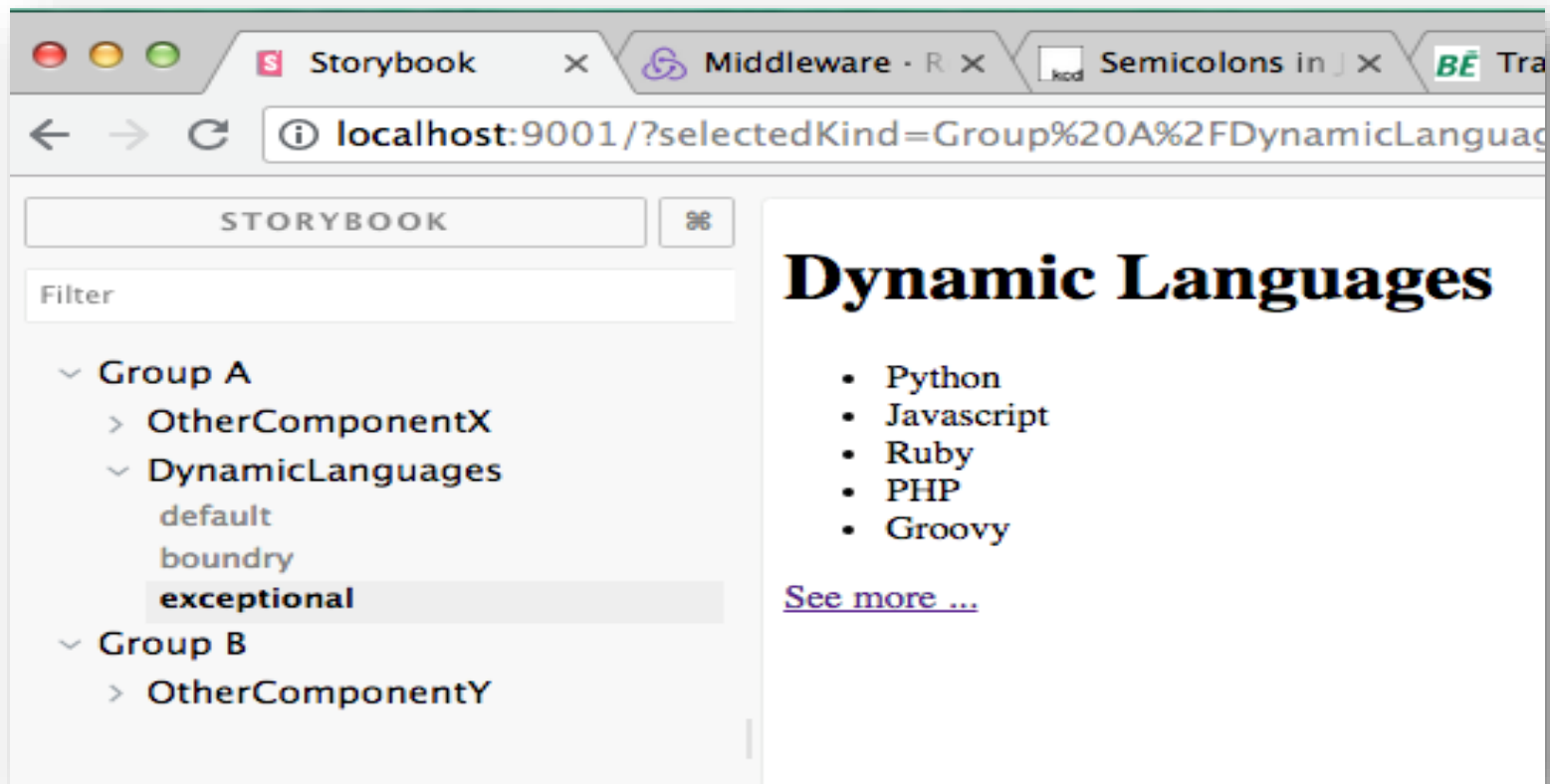
The screenshot shows the Storybook web application running on a browser. The address bar displays the URL `http://localhost:6006/?path=/story/dynamic-languages--exceptional`. The left sidebar contains the Storybook logo, a search bar labeled 'Find components', and a list of component categories: 'Configure your project', 'Dynamic Languages', 'Basic', 'Boundary', and 'Exceptional'. The 'Exceptional' category is currently selected and highlighted in blue. A blue callout box points to this category with the text: 'Set of Component Stories: (i.e. Basic, Boundary, Exceptional)'. The main content area displays the 'Ranked Languages' component, which shows a list of programming languages: Javascript, Python, Ruby, PHP, and Lua. Below the list is a 'See More...' button.

Set of Component Stories:
(i.e. Basic, Boundary, Exceptional)



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

```
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

Aside: 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

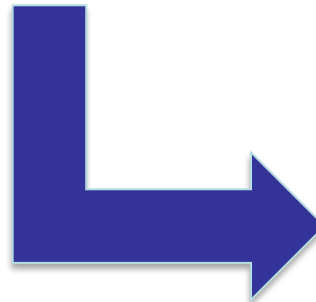
```
interface Person {  
    name: string;  
    age: number;  
}  
  
// This will compile successfully if the object satisfies the Person interface  
let person = {  
    name: "Alice",  
    age: 30,  
    occupation: "Developer"  
} satisfies Person;
```

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;
```



S Storybook

Find components

Configure your project

EXAMPLE

Button

Header

Page

EXERCISES

01 - static component

02 - JSX embedded variable

03 - component with props

04 - iteration

LANGUAGES

Dynamic Languages

SAMPLE

01 - Static Component

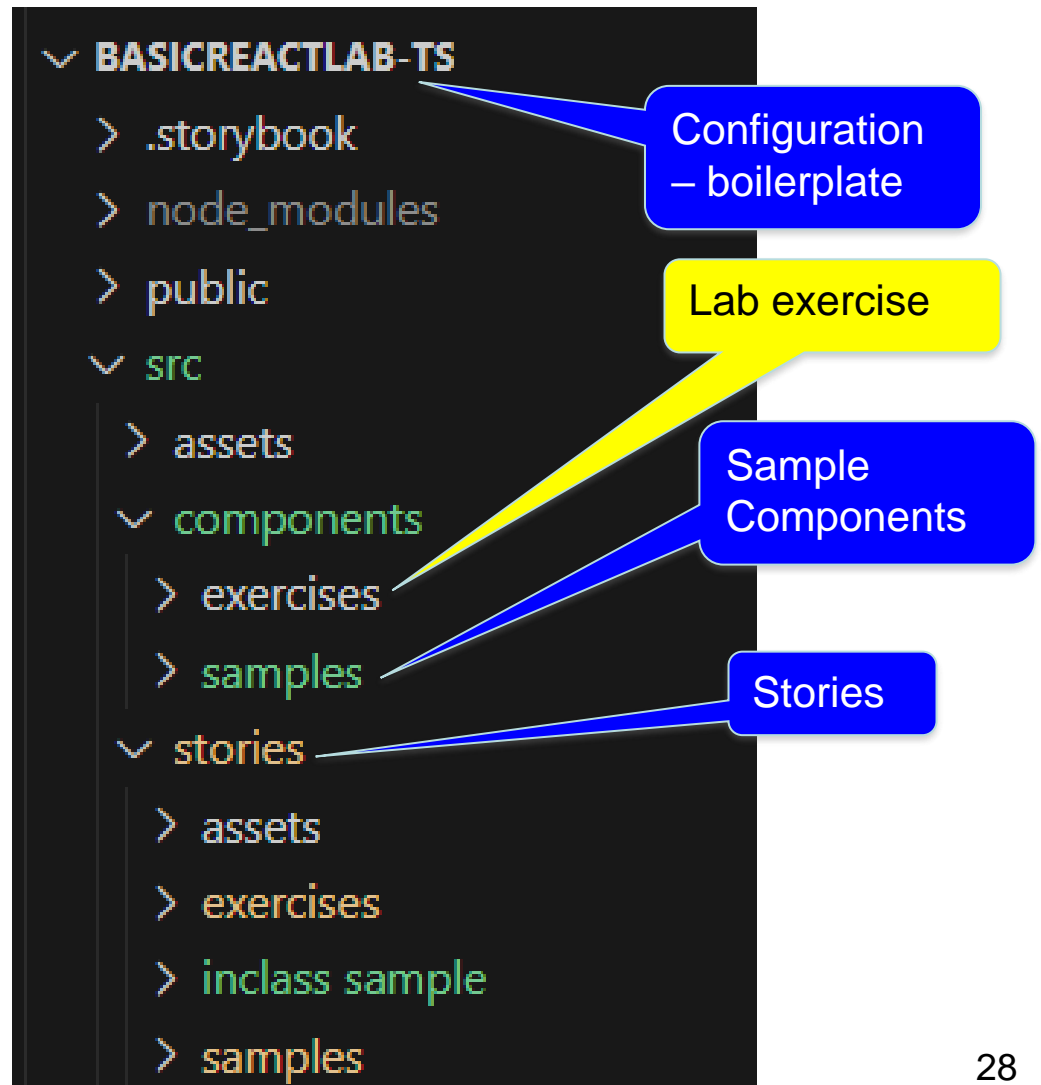
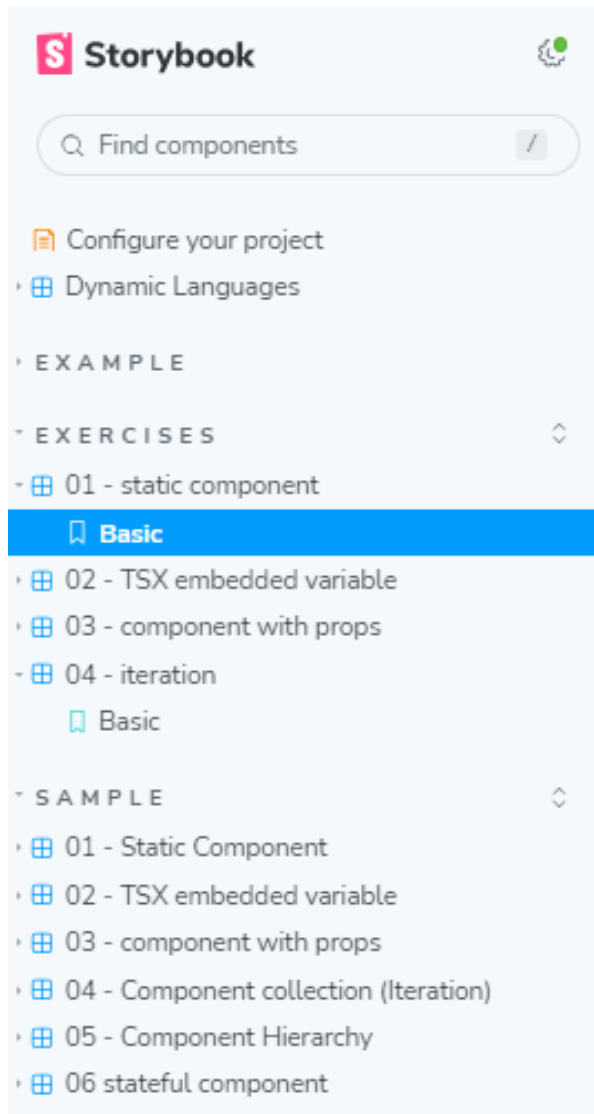
02 - JSX embedded variable

03 - component with props

... 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

```
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.

```
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.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>  
    </>  
  );  
}
```

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 <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.ts.

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](#)

1

2

3

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).**