

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!](#)

Storybook

DEVELOPER TOOLS

Introduction to

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

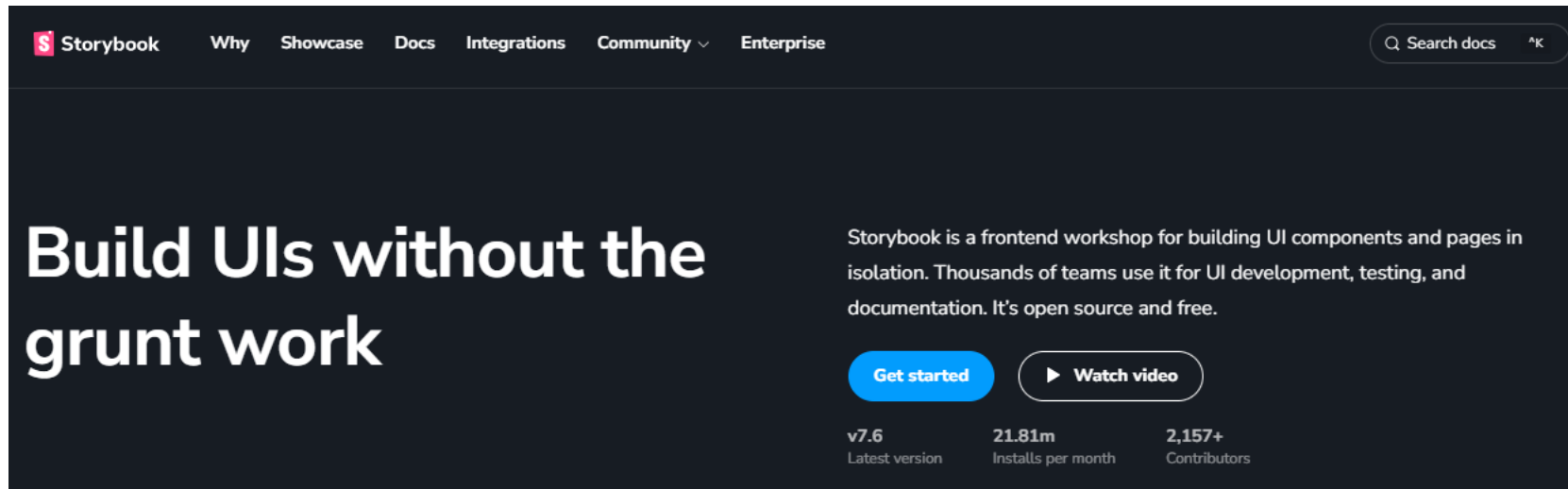
Get ready for a development environment that can finally catch up with you.

[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.





- **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 dev -p 6006",  
    "build-storybook": "storybook build"
```



- **Storybook User interface.**

The screenshot shows the Storybook web application running in a browser. The browser's 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:** Displays the title 'Dynamic Languages' and a list of supported languages: Javascript, Python, Ruby, and PHP.
- Right Panel (Component Arguments and Actions):** Shows the 'Controls' tab with a list of arguments: 'Name', 'languages', and 'heading'. Below this is a table with columns 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 left sidebar to the 'Dynamic Languages' title in the top bar.
- A blue callout box labeled 'Component Rendering.' points to the 'Dynamic Languages' title in the top bar.
- A blue callout box labeled 'Component arguments and actions (callbacks)' points to the 'languages' argument in the 'Controls' tab.



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**

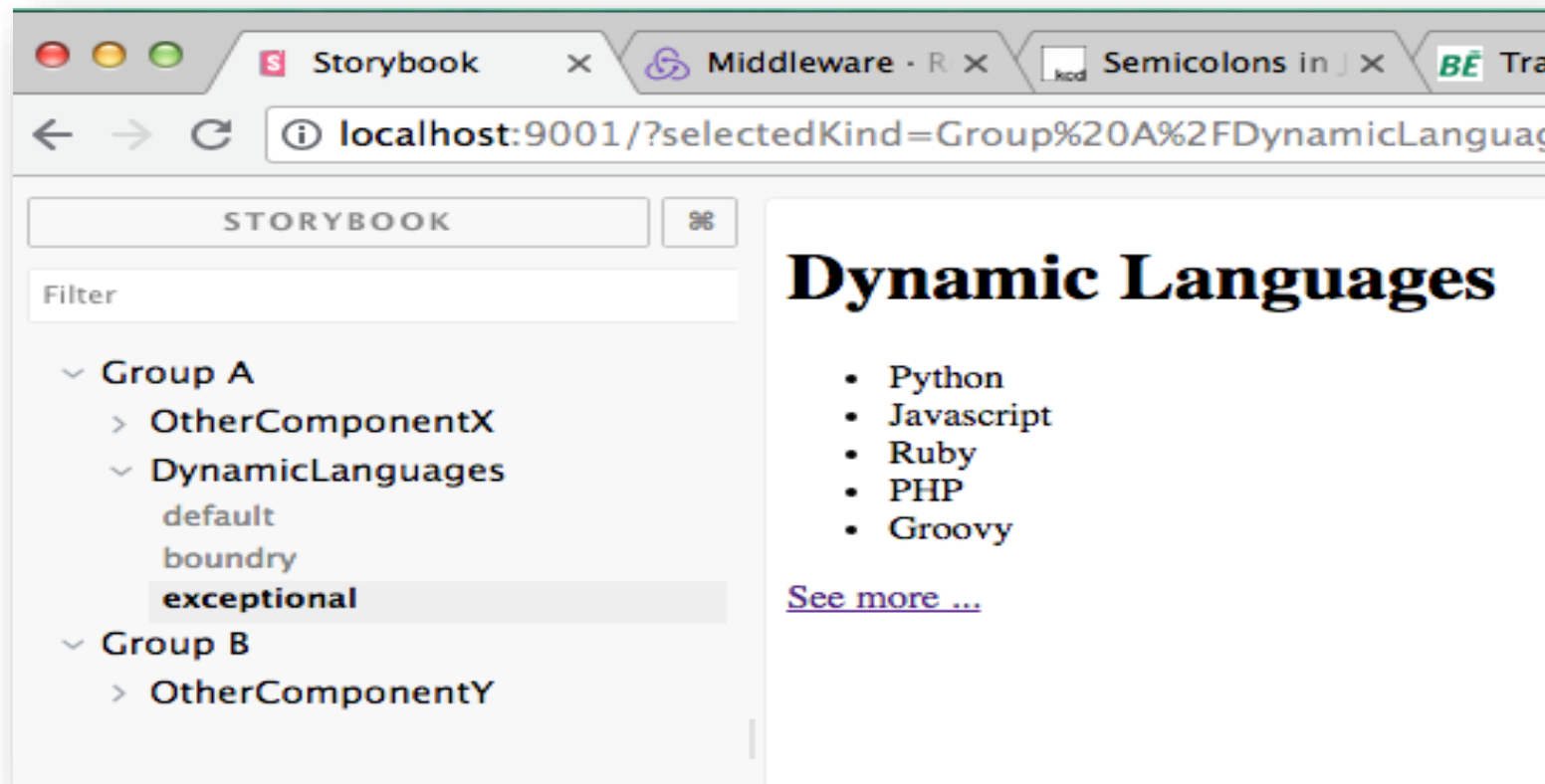


- **List a component's states/stories under its name:**

The screenshot shows a web browser at the URL `http://localhost:6006/?path=/story/dynamic-languages--exceptional`. The Storybook interface includes a sidebar on the left with a search bar and a list of components: 'Configure your project', 'Dynamic Languages' (expanded), 'Basic', 'Boundary', and 'Exceptional' (highlighted in blue). The main content area displays the 'Ranked Languages' component, which lists several programming languages: Javascript, Python, Ruby, PHP, and Lua. A 'See More...' button is located below the list. A blue callout box points to the 'Exceptional' state in the sidebar, containing the text: '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;
```

