Photo by Dollar Gill on Unsplash

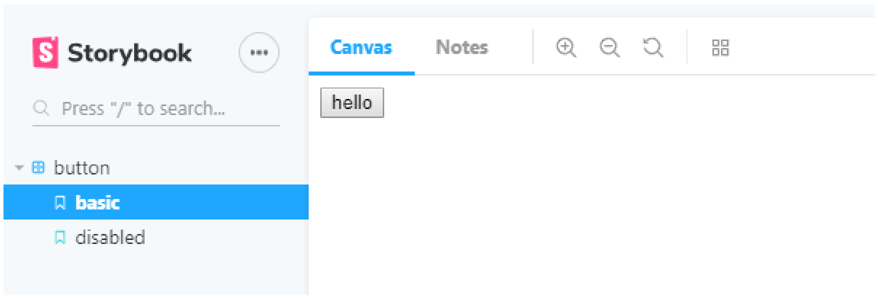
How to Build a Component Library with Angular and Storybook

By visualizing your UI component library both developers and designers become more productive.

Michael Karén Follow
Apr 15 · 7 min read

In this article, I first describe how to build a **component library** with Angular CLI. After the library is done, I write some stories in **Storybook to test and document** the UI components.

Storybook is a UI component explorer that visualizes components in different states. Storybook enables structured UI development, testing, and documentation for the major JavaScript frameworks.



• • •

Before we start

I have a habit of using short hands of script commands and flags so here are a some that I might use in this article:

```
npm install -> npm i
ng test -> ng t
ng generate ->  ng g
ng generate library -> ng g lib
```

Component Library

The goal is to have a component library that we can explore with Storybook. For this, I'm creating a library with a button component.

Workspace

The Angular CLI is constantly getting better and the functionality I'm using here came with versions six and seven of Angular.

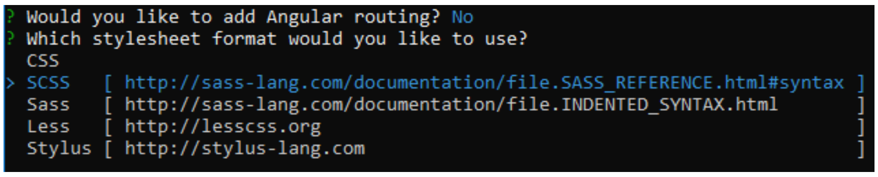
Make sure you have Angular CLI installed globally with:

```
npm i -g @angular/cli@7
```

To begin with I'm going to create an empty workspace called **ng-ui**.

I don't want to have an application and for that the `--create-application` flag is used with the `ng new` command. Setting this to **false** creates an empty workspace with no initial app.

```
ng new ng-ui --create-application=false
```



```
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use?
  CSS
> SCSS   [ http://sass-lang.com/documentation/file.SASS_REFERENCE.html#syntax ]
  Sass   [ http://sass-lang.com/documentation/file.INDENTED_SYNTAX.html       ]
  Less   [ http://lesscss.org                                                 ]
  Stylus [ http://stylus-lang.com                                             ]
```

For deeper knowledge in this topic read this great article by Todd Palmer:

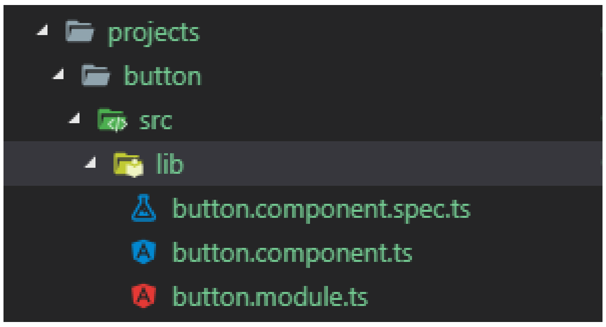
- Angular Workspace: No Application for You!

Libraries

Now let's create a component. Since each component is a library we can create one called button. Documentation for generate library.

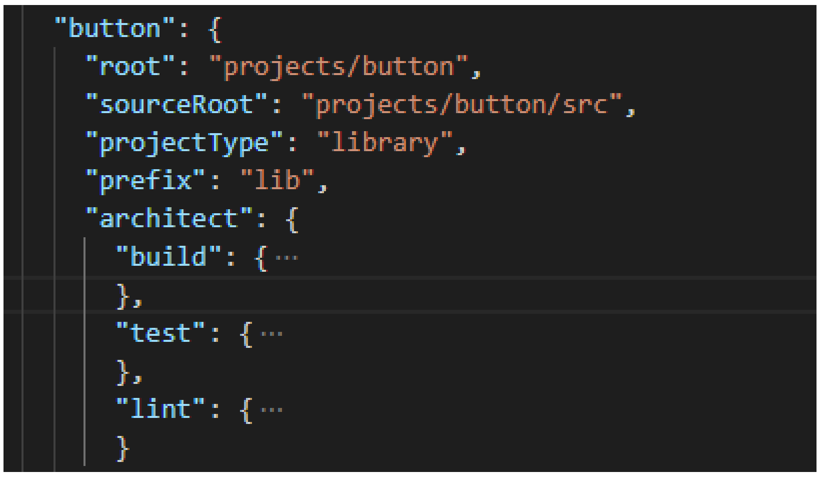
```
ng g lib button -p ui
```

This creates the `projects/button` folder in your workspace, which contains a component and a service inside an NgModule. We don't need the service so we can delete its files.



```
projects
  button
    src
      lib
        button.component.spec.ts
        button.component.ts
        button.module.ts
```

The workspace configuration file, `angular.json`, is updated with a project of type `library`.



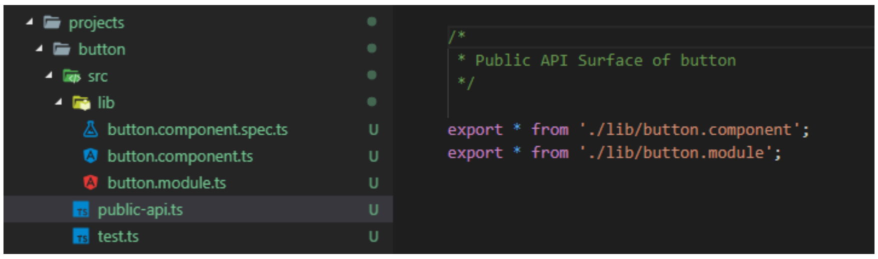
```
"button": {
  "root": "projects/button",
  "sourceRoot": "projects/button/src",
  "projectType": "library",
  "prefix": "lib",
  "architect": {
    "build": {…
    },
    "test": {…
    },
    "lint": {…
    }
```

As we can see, the `architect` has `build` , `test` , and `lint` so we can then use those commands.

```
ng build button
ng test button
ng lint button
```

The public API for your library is maintained in the `public-api.ts` file in your library folder. Anything exported from this file is made public when your library is imported into an application. Use an NgModule to expose services and components.

If you are using an older CLI you might have snake_case instead of kebab-case for this file: `public_api.ts`



```
projects
  button
    src
      lib
        button.component.spec.ts   U
        button.component.ts        U
        button.module.ts           U
      public-api.ts                U
      test.ts                      U
```

```
/*
 * Public API Surface of button
 */

export * from './lib/button.component';
export * from './lib/button.module';
```

I have written two articles that go more into detail about creating libraries, adding useful things to them, and using them in other projects:

- Get Started with your first Angular Library
- Custom Angular Pipes and Dynamic Locale

Button

Let’s change the code of the button component to have two input properties: `text` and `disabled`.

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'lib-button',
  template: `
    <button [disabled]="disabled">{{text}}</button>
  `
})
export class ButtonComponent {
```

Now we can build the button library.

```
ng build button
```

And now let’s also pack it so we can install it in other projects. For this, I’m going to pack it to a tarball. Since we need to be in the button directory when we run `npm pack` we can create a script for it.

```
"scripts": {
  ...
  "pack_button": "cd dist/button && npm pack",
  ...
}
```

Now we can use `npm run pack_button` to get a tarball called `button-0.0.1.tgz.`

Or if you want to publish it to a repository use `npm publish` instead of pack.

For deeper knowledge on building and packaging you can read this great article by Todd Palmer:

- The Angular Library Series—Building and Packaging

If you are into scripting and automation like I am then you can read this article where I explain how to automate your process with multiple libraries**.**

- Script the pain away

Let’s see how we can use our button component in Storybook next.

Storybook is an environment for defining, developing, and testing UI components. It started as react-storybook but it is now open for other frameworks as well. Storybook is open source with a strong community behind it. Hundreds of contributors, tens of thousands of stars on GitHub, and millions of monthly downloads ensures a bright future.

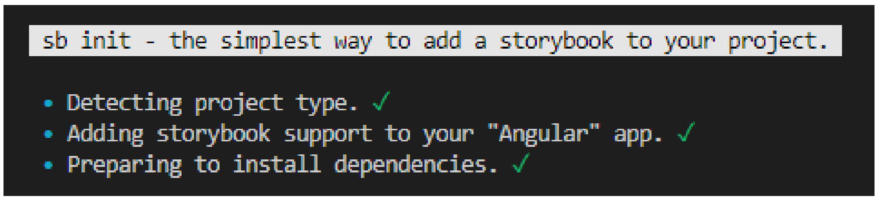
> *It’s open source and totally free to use.*

Setup

Now, let’s add Storybook to our project. It comes with a CLI tool that makes it really easy to add so let’s use it to initialize our project to a storybook project. Here I use npx so I don’t have to install it globally.

```
npx @storybook/cli init
```

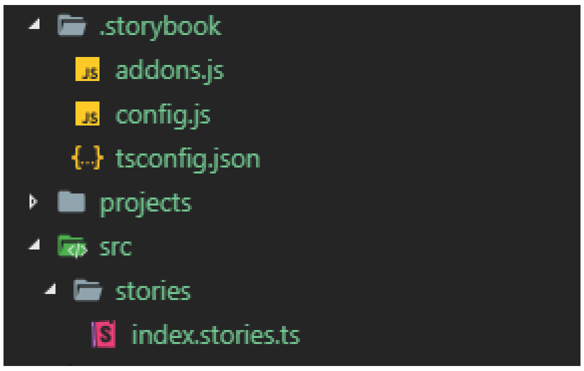
It detects that our project is an Angular project and sets things up accordingly.



```
sb init - the simplest way to add a storybook to your project.

 • Detecting project type. ✓
 • Adding storybook support to your "Angular" app. ✓
 • Preparing to install dependencies. ✓
```

All the necessary configurations, devDependencies, and npm scripts are added to our project. It also created `src/stories/index.stories.ts` with example stories.



```
◢ .storybook
    addons.js
    config.js
    tsconfig.json
▷ projects
◢ src
  ◢ stories
      index.stories.ts
```

Since we didn't have an application in our workspace we have to change the path to the TypeScript configuration file in `storybook/tsconfig.json` so that it instead extends the one in the root folder.

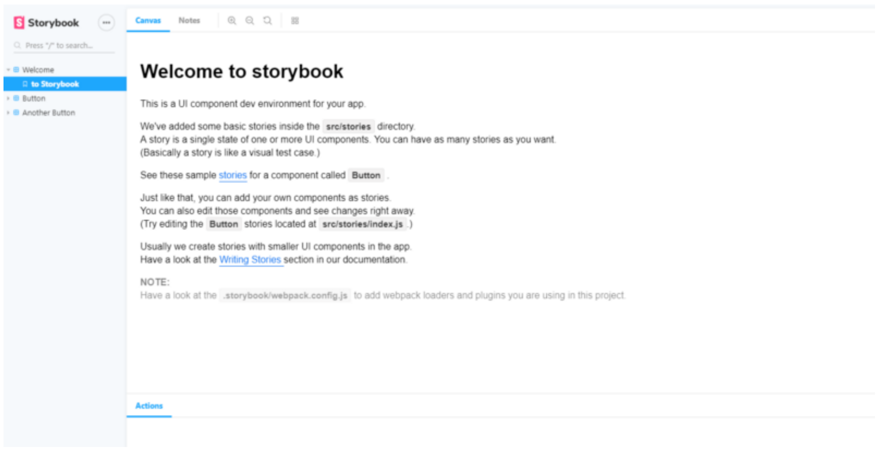
```
"extends": "../tsconfig.json"
```

Startup

We can now run storybook.

```
npm run storybook
```

This will start a local webpack server on port 6006 and you can visit the generated storybook by going to http://localhost:6006/. What you'll see is an example Storybook that was generated from an index.stories.ts file under a stories folder at the root of the project.



Welcome to storybook

This is a UI component dev environment for your app.

We've added some basic stories inside the `src/stories` directory.
A story is a single state of one or more UI components. You can have as many stories as you want.
(Basically a story is like a visual test case.)

See these sample stories for a component called `Button` .

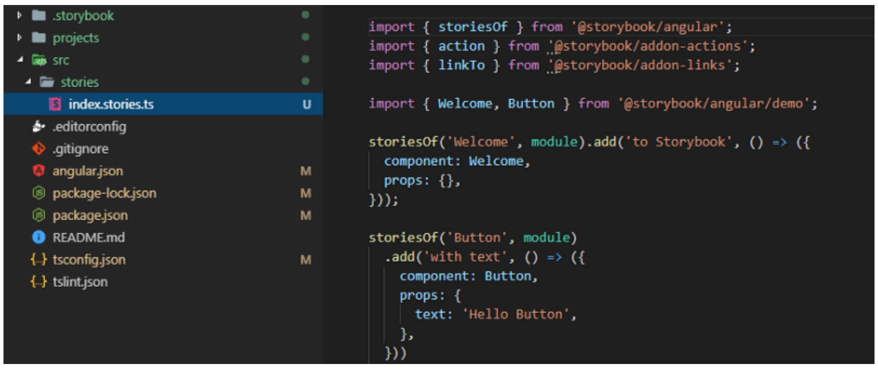
Just like that, you can add your own components as stories.
You can also edit those components and see changes right away.
(Try editing the `Button` stories located at `src/stories/index.js` .)

Usually we create stories with smaller UI components in the app.
Have a look at the Writing Stories section in our documentation.

NOTE:
Have a look at the `.storybook/webpack.config.js` to add webpack loaders and plugins you are using in this project.

You can see the generated sample stories in `/stories/index.stories.ts.`

There are two levels to the tree structure of Storybook. The component is defined by **storiesOf** and we use **add** for showing different views of it. You can add as many stories per component as you need.



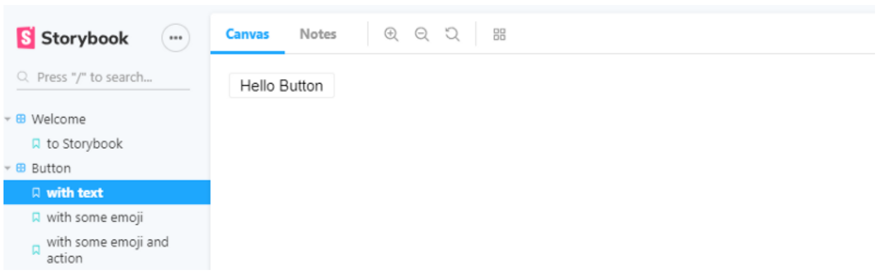
```
import { storiesOf } from '@storybook/angular';
import { action } from '@storybook/addon-actions';
import { linkTo } from '@storybook/addon-links';

import { Welcome, Button } from '@storybook/angular/demo';

storiesOf('Welcome', module).add('to Storybook', () => ({
  component: Welcome,
  props: {},
}));

storiesOf('Button', module)
  .add('with text', () => ({
    component: Button,
    props: {
      text: 'Hello Button',
    },
  }))
```

And we can click to the corresponding button in the Storybook tree view.



Let’s delete these examples and create a story for the button we created in our library. To be able to use it we must first install it from the tarball we created.

Integration

We want to use the button library in our application so we use **npm install** like this:

```
npm i dist/button/button-0.0.1.tgz
```

Now if we open *package.json* we should see that the library is added to the dependencies.

```
"button": "file:dist/button/button-0.0.1.tgz"
```

If you instead published it to a repository you can install it as normal.

```
npm i button
```

Stories

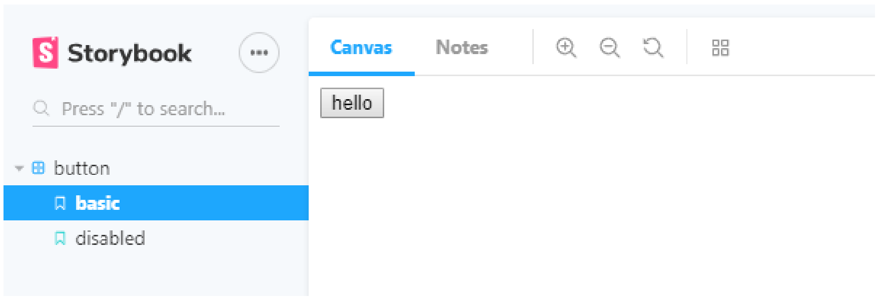
> *Storybook is all about writing stories. A story usually contains a single state of one component, almost like a visual test case.*

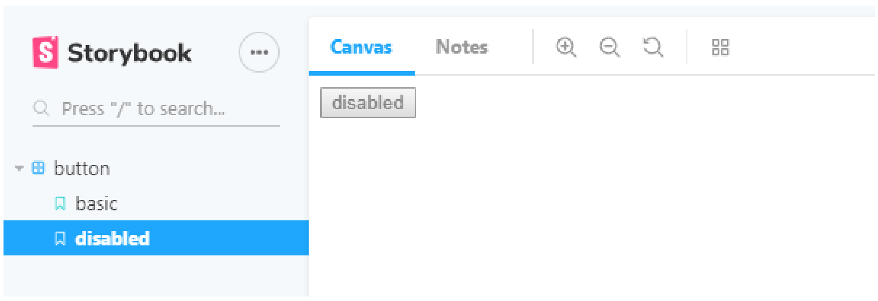
Let’s add a couple of stories to our button component. We have two input properties that we can test. For the first story, we send in ‘hello’ to the text property and get a normal button with hello written on it.

For the second story, we send in true to the disabled property and get a disabled button in the Storybook.

```
import { storiesOf } from '@storybook/angular';
import { ButtonComponent } from 'lib-button';

storiesOf('button', module)
  .add('basic', () => ({
    component: ButtonComponent,
    props: {
      text: 'hello'
    }
  }))
  .add('disabled', () => ({
    component: ButtonComponent,
```





By adding stories for all permutations of the properties of the component we get visual tests for each use-case. We can test and play with our component in an isolated environment. And this also enables designers to work on styling the components that are used in the application.

. . .

Conclusion

We've seen how and why you should consider using Storybook with Angular. By visualizing your UI component library, Storybook can help both developers and designers become more productive. And by writing stories we can also test all use-cases of the components.

And since it is open source and free to use there is no excuse to at least give it a try!

• • •

When you are done it’s time to release the libraries:

- Release management in Angular with Lerna

Resources

- Angular CLI
- Storybook
- Storybook Tutorial