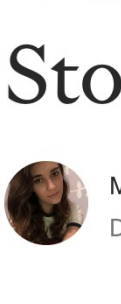
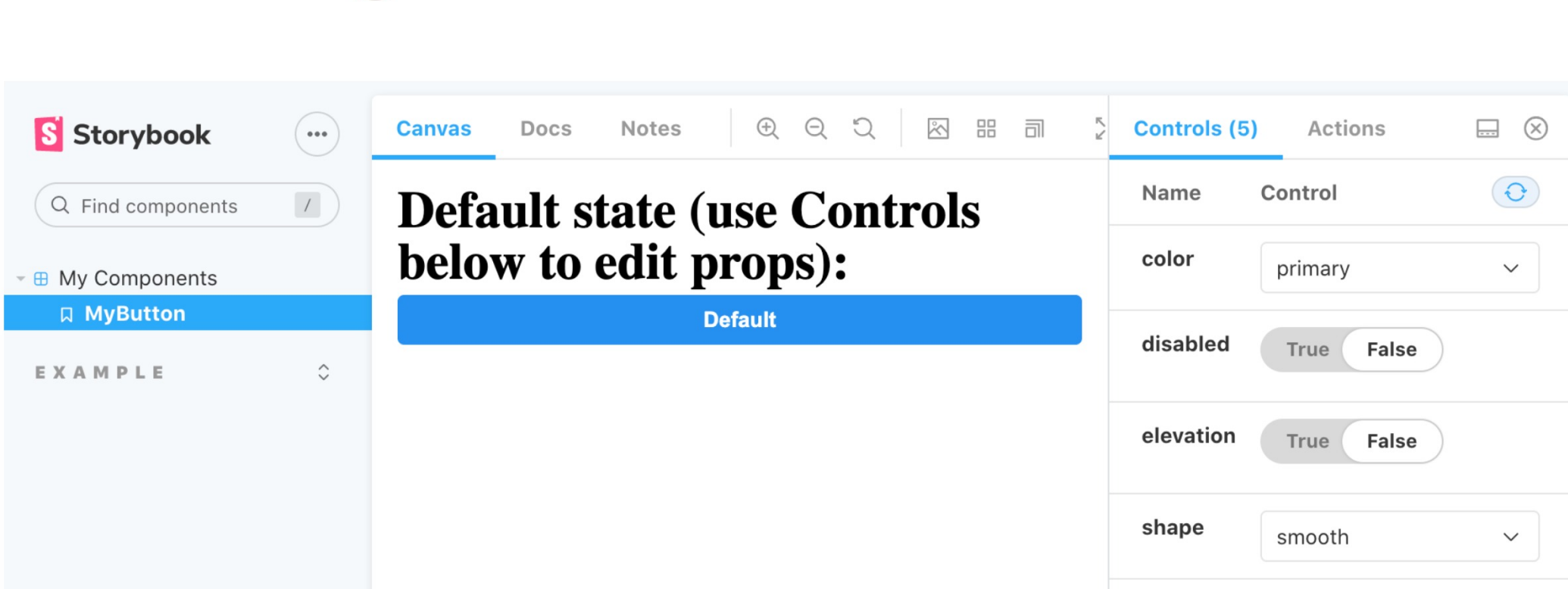
This is your last free member-only story this month. Sign up for Medium and get an extra one

# Automated Stories With Storybook and StencilJS



Magaly Muñoz Follow
Dec 3 · 5 min read ★



Default state (use Controls below to edit props):

Default

color primary
disabled True False
elevation True False
shape smooth
size medium

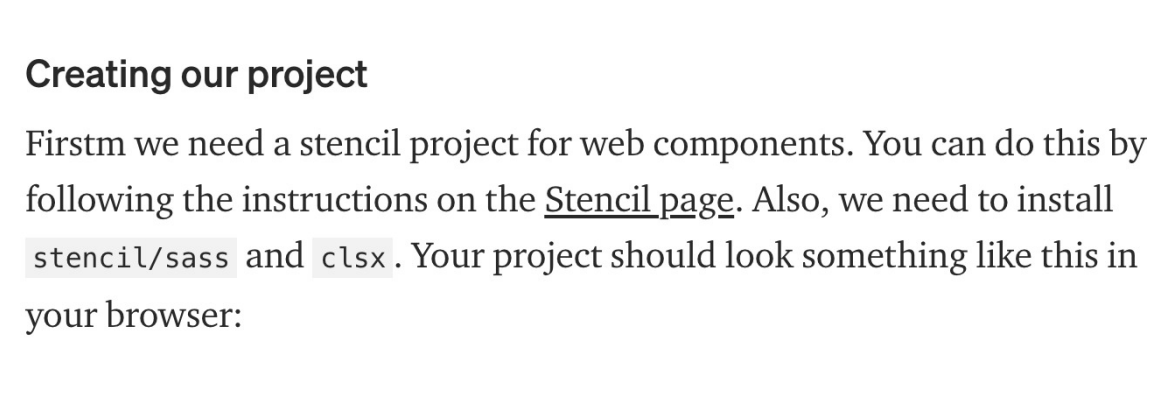
Autogenerated story

Stencil is an incredible tool that helps you create web components, which you can use with Vanilla Javascript and some frameworks like React, Vue, Angular … , and combined with storybook you can offer an incredible tool for all our community, in this article, I will show you how to save some time and create all your stories in an automatic way.

> ***Note:*** *For this project, we are assuming that you have some basic knowledge of StencilJS*

### Creating our project

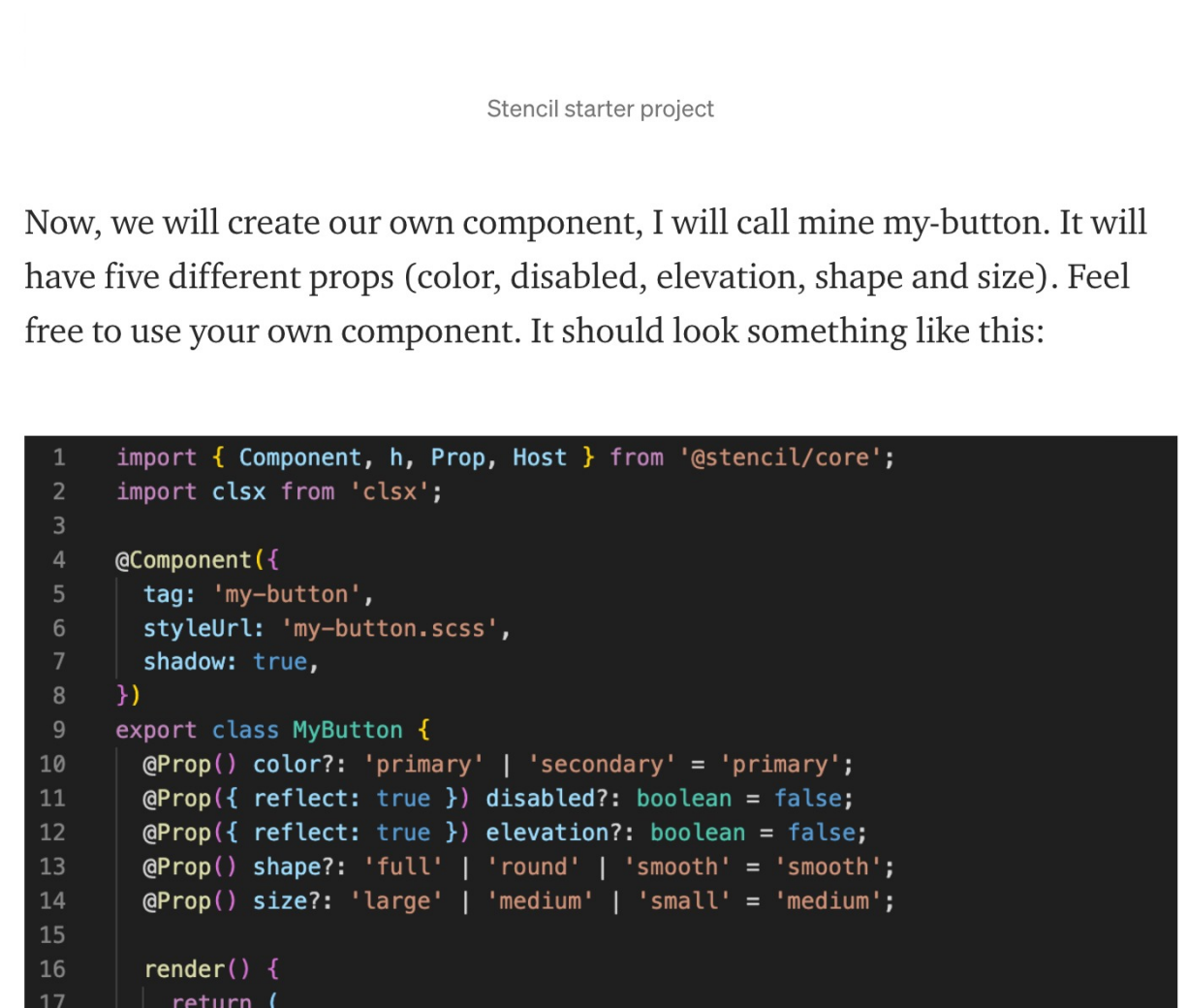
Firstm we need a stencil project for web components. You can do this by following the instructions on the Stencil page. Also, we need to install `stencil/sass` and `clsx`. Your project should look something like this in your browser:



Hello, World! I'm Stencil 'Don't call me a framework' JS

Stencil starter project

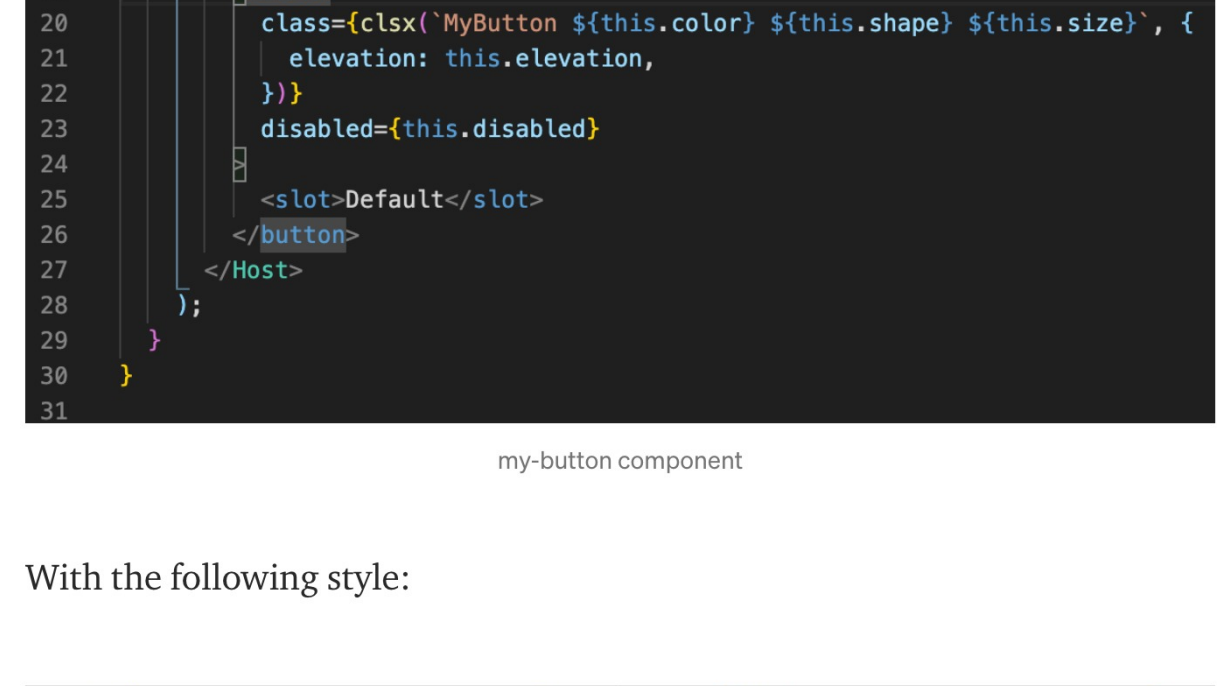
Now, we will create our own component, I will call mine my-button. It will have five different props (color, disabled, elevation, shape and size). Feel free to use your own component. It should look something like this:



```
import { Component, h, Prop, Host } from '@stencil/core';
import clsx from 'clsx';

@Component({
  tag: 'my-button',
  styleUrl: 'my-button.scss',
  shadow: true,
})
export class MyButton {
  @Prop() color?: 'primary' | 'secondary' = 'primary';
  @Prop({ reflect: true }) disabled?: boolean = false;
  @Prop({ reflect: true }) elevation?: boolean = false;
  @Prop() shape?: 'full' | 'round' | 'smooth' = 'smooth';
  @Prop() size?: 'large' | 'medium' | 'small' = 'medium';

  render() {
    return (
      <Host>
        <button
          class={clsx(`MyButton ${this.color} ${this.shape} ${this.size}`, {
            elevation: this.elevation,
          })}
          disabled={this.disabled}
        >
          <slot>Default</slot>
        </button>
      </Host>
    );
  }
}
```

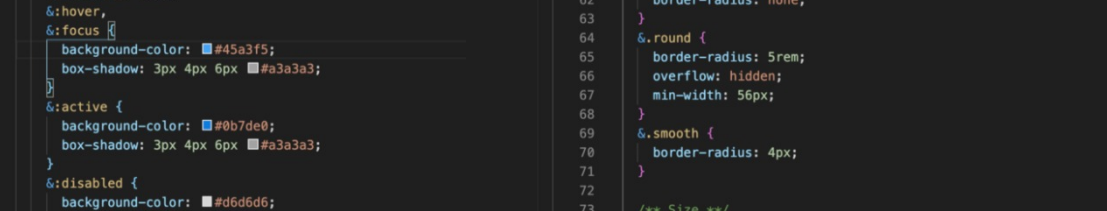
my-button component

With the following style:



```
:host {
  display: block;
}

.MyButton {
  width: 100%;
  font-weight: 600;
  cursor: pointer;
  outline: none;
  display: flex;
  justify-content: center;
  align-items: center;

  /** Colors **/
  &.primary {
    background: #1b8df2 0% 0% no-repeat padding-box;
    border: none;
    color: white;
    transition: 0.3s;
    &:hover,
    &:focus {
      background-color: #45a3f5;
      box-shadow: 3px 4px 6px #a3a3a3;
    }
    &:active {
      background-color: #0b7de0;
      box-shadow: 3px 4px 6px #a3a3a3;
    }
    &:disabled {
      background-color: #d6d6d6;
      color: #a3a3a3;
      cursor: not-allowed;
      box-shadow: none;
    }
  }
  &.secondary {
    background: transparent;
    border: 1px solid #1b8df2;
    border-radius: 4px;
    color: #1b8df2;
    transition: 0.3s;
    &:hover,
    &:focus {
      background-color: #7fbef5;
      color: #45a3f5;
    }

    &:active {
      background-color: #45a3f5;
      color: #45a3f5;
    }
    &:disabled {
      border-color: #d6d6d6;
      color: #a3a3a3;
      background-color: #f7f7f7;
      cursor: not-allowed;
    }
  }

  /** Shape **/
  &.full {
    border-radius: none;
  }
  &.round {
    border-radius: 5rem;
    overflow: hidden;
    min-width: 56px;
  }
  &.smooth {
    border-radius: 4px;
  }

  /** Size **/
  &.large {
    padding: 12px;
  }
  &.medium {
    padding: 8px;
  }
  &.small {
    padding: 4px;
  }

  /** Elevation **/
  &.elevation {
    box-shadow: 3px 4px 6px rgba(0, 0, 0, $alpha: 0.3);
  }
}
```

Once you have it, use it on your `index.html`, and see the results. If you would have used my example you should see something like this:



my-button component with the default props

Great!! Now we have a component to start working on.

. . .

### Installing and configuring storybook

Let's start with the storybook configuration, since the stencil option for web components is still in alpha, we will install storybook for HTML. Use the following command on your project to install storybook:

```
npx sb init
```

> ***Note:*** *For this project, we are using @storybook/html version ^6.1.2, down versions will not work*

Once storybook is up and running, add the next dependencies:

```
npm install npm-run-all @storybook/addon-essentials
@storybook/addon-notes @storybook/addon-viewport case --save-dev
```

We will use addon-essentials to install controls, so that anyone can interact dynamically with our components; addon-notes to charge the documentation generated automatically by stencil; addon-viewport (optional) to check our components on different screen sizes; case and npm-run-all will be used for our script to generate each story, we will cover them later. Now let's go to the `.storybook` directory and modify the `main.js` file as follows:

```
const path = require('path');

module.exports = {
  stories: ['../src/**/*.stories.mdx', '../src/**/*.stories.@(js|jsx|ts|tsx)'],
  addons: ['@storybook/addon-links', '@storybook/addon-essentials', '@storybook/addon-vi
  webpackFinal: async (config, { configType }) => {
    // `configType` has a value of 'DEVELOPMENT' or 'PRODUCTION'
    // You can change the configuration based on that.
    // 'PRODUCTION' is used when building the static version of storybook.

    config.devServer = {
      watchContentBase: true,
      contentBase: path.join(__dirname, 'src'),
      historyApiFallback: true,
    };

    config.module.rules[0].use[0].options.sourceType = 'unambiguous';

    config.module.rules.push({
      test: /.\.stories\.tsx$/,
      exclude: /(src)/,
      use: 'raw-loader',
    });
    config.resolve.extensions.push('.stories.tsx');

    config.resolve.alias = {
      '@src': path.resolve(__dirname, '../dist/collection'),
    };

    // Return the altered config
    return config;
  },
};
```

main.js hosted with ❤ by GitHub view raw

.storybook/main.js

We have changed the default storybook webpack configuration to charge our automated stories in a correct way. Now, we need to build the last version of our components in order for them to work in our stories. For this purpose, we need to add the next scripts to our `package.json`:

```
"storybook.run": "start-storybook -p 6006 -s dist",
"storybook": "npm-run-all --parallel build storybook.run",
"build-storybook": "build-storybook -c ./.storybook -o ../public"
```

If you run storybook now, you should have a readme file created by stencil inside each component, we will use them in our automated stories. To do this first we have to create a `global.d.ts` file inside your src directory with the next line inside:

```
declare module '*.md';
```

global.d.ts hosted with ❤ by GitHub view raw

global.d.ts

### Our first story

Our next step is to create our first story. Inside our my-button component directory, create a file called `my-button.stories.tsx`, add the next code:

```
import readme from './readme.md';

export default {
  title: 'MyButton',
  /**
   * Notes is an optional string which will be displayed in the "Notes"
   * tab for your component. It is recommended to use the generated readme,
   * however any string should suffice. This supports markdown.
   */
  notes: readme,
};
```

my-button.stories.tsx hosted with ❤ by GitHub view raw

my-button.stories.tsx

Here, we are importing our readme file to show it as the documentation, later we will see how storybook will manage it.

### Creating our script to automate stories

Now, let's start working on the creation of our automated stories. For this purpose, we will create a script dedicated to this. Inside your `.storybook` directory, create a `stories` directory with a file called `automatedStories.js`. From now on we will be adding a lot of code inside this file, so let's start. Our first step is to create the function that will be in charge of receiving each component and story created to complete with our base render. It should look like this:

```
import { storiesOf } from '@storybook/html';

/**
 * Iterates all of the stencil contexts and build a "config" object
 * which is used to generate the individual stories.
 */
function buildStencilStories(name, componentsCtx, storiesCtx) {
  const configs = buildGeneratorConfigs(componentsCtx, storiesCtx);

  const stories = storiesOf(name, module);

  Object.keys(configs)
    .map(comp => configs[comp])
    .forEach(config =>
      typeof config === 'function'
        ? // If the config is a function, call it with the stories context.
          // The function is responsible for calling stories.add(...) manually.
          config(stories)
        : createStencilStory(config, stories),
    );
}

export default buildStencilStories;
```

automatedStories.js hosted with ❤ by GitHub view raw

.storybook/stories/automatedStories.js (part 1)

This function receives our configs and checks if we have an object (which is the way we will create the stories we want to be processed by our script) or a function. In case it is a function, we will let storybook manage it, but the other way we want to use our own script. Now, let's add our configs generator:

```
import path from 'path';

/**
 * Given a module, iterates over the exports and returns the first
 * one which looks like a stencil component (using duck typing).
 */
function getComponentFromExports(_module) {
  const key = Object.keys(_module).find(exportKey => {
    const _export = _module[exportKey];
    // does it quack like a stencil class component?
    if (_export.prototype && _export.is && _export.encapsulation) {
      return true;
    }
  });

  return _module[key];
}

/**
 * Cleans the notes, which should be in markdown format.
 * The markdown parser used by the notes addon is not the best, so
 * we have to fix some issues before rendering.
 */
function cleanNotes(notes) {
  if (notes) {
    // replaces "\|" with "` `" so property tables to break
    return notes.replace(/\\\|/g, '` `');
  }
}

// Gets all stories and check for specific configuration to add to each story
function buildGeneratorConfigs(componentsCtx, storiesCtx) {
  const componentRoutes = componentsCtx.keys();
  const storyRoutes = storiesCtx.keys();

  return componentRoutes.reduce((obj, compRoute) => {
    const _module = componentsCtx(compRoute);
    const Component = getComponentFromExports(_module);
    const dirName = '/' + path.basename(path.dirname(compRoute)) + '/';
    const storyRoute = storyRoutes.find(k => k.indexOf(dirName) > -1);

    if (!Component) {
      console.warn(`Couldn't load component ${compRoute}`);
      return obj;
    }

    if (storyRoute) {
      const _export = storiesCtx(storyRoute).default;

      // If the default export is a function, then that function should
      // be used to create the story. It will be passed the "stories" object
      // where it should call stories.add(...) manually.
      if (typeof _export === 'function') {
        return Object.assign(obj, {
          [Component.name]: _export,
        });
      }

      return Object.assign(obj, {
        [Component.name]: {
          Component,
          states: _export.states,
          args: _export.args,
          argTypes: _export.argTypes,
          notes: cleanNotes(_export.notes),
        },
      });
    }

    return Object.assign(obj, {
      [Component.name]: {
        Component,
      },
    });
  }, {});
}
```

automatedStories.js hosted with ❤ by GitHub view raw

.storybook/stories/automatedStories.js (part 2)

Here, we are getting all our Stencil Components and object for each of them who will have our custom configuration. This configuration includes our *Component; states*, which will be use to write extta renders we want to add to our story; *args*, which will include our props; *argTypes*, which will specify the control (check addon-controls) we want to use for each arg; and *notes* which will load our readme files. We have to clean our notes because addon-notes has some issues rendering tables.

Now that we have our configs covered, we can write our function to create stories, add the next code:

```
/**
 * Generates DOM nodes from states to render.
 */
function createNodes(el, elements) {
  if (elements && elements.length > 0) {
    elements.forEach(({ tag, innerText, props, children }) => {
      let childEl = document.createElement(tag);
      childEl.innerHTML = innerText;
      if (props) {
        Object.keys(props).forEach(prop => {
          if (props[prop]) {
            childEl.setAttribute(prop, props[prop]);
          } else {
            childEl.removeAttribute(prop);
          }
        });
      }
      createNodes(childEl, children);
      el.appendChild(childEl);
    });
  }
}

/**
 * Generates an interactive controls-enabled story for a stencil Component.
 * For any additional states, a static rendering is generated with
 * the given state.
 */
function createStencilStory({ Component, notes, states, args = {}, argTypes = {} }, stor
  // It is important that the main container element
  // is NOT created inside of the render function below!!
  const mainEl = document.createElement('div');
  const controls = getPropsWithControlValues(Component, { args, argTypes });
  const storyOpts = notes ? { notes, args: controls.args, argTypes: controls.argTypes }
  const tag = Component.is;

  // Clone the "states" array and add the default state first
  states = states && states.length ? states.slice(0) : [];
  states.unshift({
    title: 'Default state (use Controls below to edit props):',
    tag: Component.is,
    props: {},
    children: [{ tag: 'span', innerText: 'Default' }],
  });

  // Create the story with all of the states
  stories.add(
    Component.name,
    args => {
      mainEl.innerHTML = '';
      // First, add the controls-enabled props to the default state.
      // This MUST be done inside this render function!!
      states[0].props = { ...args };
      states[0].argTypes = controls.argTypes;

      // Next, render each state. Only the first one is interactive (with controls).
      // This is sort of a light-weight "chapters" addon because the community
      // "chapters" addon only works with react :/
      states.forEach(({ title, description, props, children }) => {
        const containerEl = document.createElement('div');
        const componentEl = document.createElement(String(tag));

        if (props) {
          Object.keys(props).forEach(prop => {
            const propKebab = Case.kebab(prop);
            if (props[prop]) {
              componentEl.setAttribute(propKebab, props[prop]);
            } else {
              componentEl.removeAttribute(propKebab);
            }
          });
        }

        if (children) {
          createNodes(componentEl, children);
        }

        containerEl.innerHTML = getStencilTemplate({
          title,
          description,
          tag,
          props,
          args,
          children,
        });

        containerEl.querySelector(`.placeholder`).appendChild(componentEl);
        mainEl.appendChild(containerEl);
      });

      return mainEl;
    },
    storyOpts,
  );
}
```

automatedStories.js hosted with ❤ by GitHub view raw

.storybook/stories/automatedStories.js (part 3)

Here, we are adding each stories, creating first our automated story. We are getting our props from our controls (I will explain how to do this later in our *getPropsWithControlValues*), adding them for our first story. We are adding a children with the createNodes function so our component is not empty inside, notice that this is optional, and creating our component element to send them to our special template, to create our template add the next function:

```
/**
 * Template used to render a single stencil component.
 */
function getStencilTemplate({ title, description }) {
  let template = `
          <div class="component-area">
            <h2>${title}</h2>
            ${description ? '<p>' + description + '</p>' : ''}
            <div class="placeholder">
              <!-- the component will be inserted here -->
            </div>
          </div>
      `;

  return template;
}
```

automatedStories.js hosted with ❤ by GitHub view raw

.storybook/stories/automatedStories.js (part 4)

Notice that you have to add our args, argTypes and notes to our story options, so they connect to our story.

Now, let's see how to get our controls generated checking each of the stencil properties' types, the format for generate the controls can be check here:

```
import Case from 'case';

/**
 * Given a property (from stencil Component.properties) and an optional
 * controlOptions object generates a control which can be used to
 * dynamically update the properties of the component.
 */
function getControlForProp(prop, controlOptions) {
  let defaultVal = '';
  let control = {
    defaultValue: defaultVal,
    control: { type: 'text' },
  };

  // control options have to be defined using camelCase
  const propCamel = Case.camel(prop.attribute);
  const argsOption = controlOptions.args[propCamel] || controlOptions.args[prop.attribut
  const argTypesOptions = controlOptions.argTypes[propCamel] || controlOptions.argTypes[

  // if control options are defined, use those
  if (argTypesOptions) {
    control = argTypesOptions;
  }
  // otherwise, implicitly create controls based on prop type or attribute name
  else if (/^(?:number|boolean|object)$/i.test(prop.type)) {
    control = { control: { type: prop.type.toLowerCase() } };
  } else if (/^(?:string)$/i.test(prop.type)) {
    if (!/^(?:string|number|boolean|object)$/i.test(prop.complexType.original)) {
      const arrOptions = prop.complexType.original.split(' | ');
      const selectOptions = arrOptions.map(o => (o.match(/('(\w|-)+')/g) ? o.replace(/'|

      control = {
        control: {
          type: 'select',
          options: selectOptions,
        },
      };
    }
  } else if (prop.attribute.indexOf('date') !== -1) {
    control = {
      control: {
        type: 'date',
      },
    };
    defaultVal = new Date();
  }

  if (argsOption) {
    defaultVal = argsOption;
  } else if (prop.defaultValue) {
    try {
      defaultVal = prop.defaultValue;

      if (typeof defaultVal === 'string') {
        defaultVal =
          /('\w+')/g.test(defaultVal) || /('')/g.test(defaultVal) ? (/('')/g.test(defaul
      }
    } catch (e) {
      defaultVal = typeof prop.defaultValue === 'string' ? prop.defaultValue : undefined
    }
  }

  return { default: defaultVal, control: { ...control, defaultValue: defaultVal } };
}

/**
 * Given a stencil Component and control options, returns an dictionary of
 * all the properties and default values.
 */
function getPropsWithControlValues(Component, controlOptions) {
  let controls = { args: {}, argTypes: {} };
  Object.keys(Component.properties || {}).forEach(key => {
    const property = Component.properties[key];

    // normalize older "attr" into newer "attribute" property
    if (property.hasOwnProperty('attr')) {
      property.attribute = property.attr;
    }

    if (property.hasOwnProperty('attribute')) {
      const control = getControlForProp(property, controlOptions);
      controls = {
        args: { ...controls.args, [key]: control.default },
        argTypes: { ...controls.argTypes, [key]: control.control },
      };
    }
  });

  return controls;
}
```

automatedStories.js hosted with ❤ by GitHub view raw

.storybook/stories/automatedStories.js (part 5)

### Rendering our stories

Now, with all of this set we have our automatedStories script finished, our last step is to use this script. Storybook has the file called `preview.js` to control the way stories are rendered, find/create it inside our `.storybook` directory, we will modify it as follows to load the stories the way we want to.

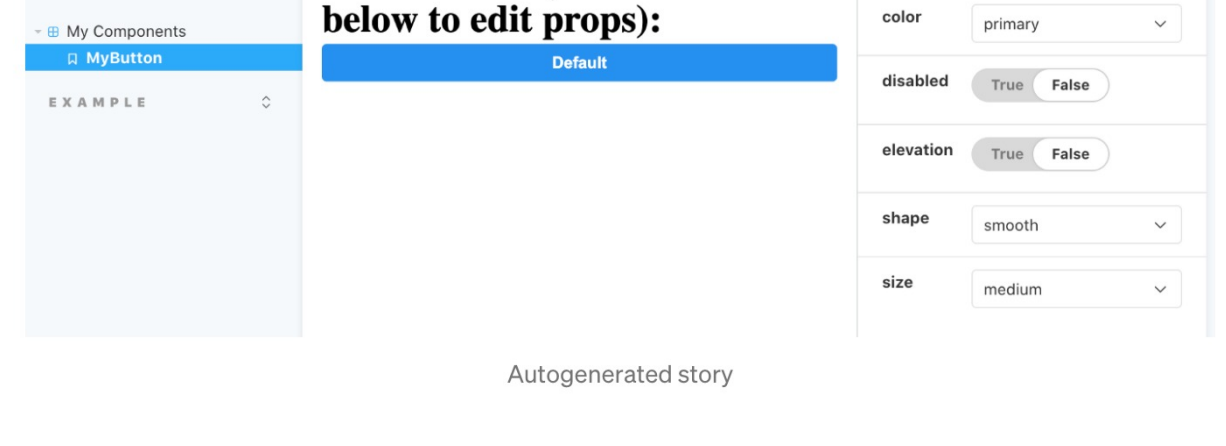
```
import { configure } from '@storybook/html';
import buildStencilStories from './stories/automatedStories';

const loader = require('../loader/index.cjs.js');

const COLLECTIONS = [
  {
    name: 'My Components',
    componentsCtx: require.context('../dist/collection', true, /\/components\/([^/]+)\/\
    storiesCtx: require.context('../src', true, /\.stories\.tsx$/),
  },
];

function loadStories() {
  loader.defineCustomElements(window);
  COLLECTIONS.forEach(({ name, componentsCtx, storiesCtx }) => {
    buildStencilStories(name, componentsCtx, storiesCtx);
  });
}

configure(loadStories, module);
```

preview.js hosted with ❤ by GitHub view raw

.storybook/preview.js

Let's analyze what we are doing, we are creating our collection (My Components) and getting all our previously build components and all our created stories. Once this is done, we are creating a function that will call each collection and use our script (*buildStencilStories*) to generate each story automatically.

Now, add the stories file to each component following the example metioned before. Run storybook and see all your stories with its own controls generated automatically. The final result should look as follows:



Default state (use Controls below to edit props):

Default

color primary
disabled True False
elevation True False
shape smooth
size medium

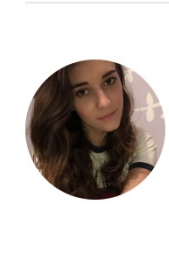
Autogenerated story

If you are having problems reproducing this, you can check the original code on the repo (https://github.com/MagalyMJ/stencil-automated-stories).

Public domain.

Stenciljs Storybook

53 claps 1 response



WRITTEN BY
**Magaly Muñoz**
Follow