**Configuring single-spa**

The single-spa root config consists of the following:

1. The root HTML file that is shared by all single-spa applications.
2. The JavaScript that calls [singleSpa.registerApplication()](https://single-spa.js.org/docs/api.html" \l "registerapplication).

Your root config exists only to start up the single-spa applications.

## Index.html file[​](https://single-spa.js.org/docs/configuration#indexhtml-file)

See [this example root config](https://github.com/polyglot-microfrontends/root-config/blob/master/src/index.ejs) for what a root HTML file looks like.

**You do not have to use SystemJS when using single-spa**, but many examples and tutorials will encourage you to do so because it allows you to [independently deploy](https://single-spa.js.org/docs/separating-applications.html) your applications.

## Registering applications[​](https://single-spa.js.org/docs/configuration#registering-applications)

You must register [applications](https://single-spa.js.org/docs/building-applications) with single-spa so it knows how and when to initiate, load, mount, and unmount each application. Registration most commonly occurs inside of the single-spa config but does not have to. Note that if an application is registered from within another application, no hierarchy will be maintained between the applications. Instead, the applications will be siblings and will be mounted and unmounted according to their own activity functions.

In order to register an application, call the registerApplication function. Example:

*// single-spa-config.js*  
*import* { registerApplication, start } *from* 'single-spa';  
  
*// Simple usage*  
registerApplication(  
 'app2',  
 () => *import*('src/app2/main.js'),  
 (location) => location.pathname.startsWith('/app2'),  
 { some: 'value' }  
);  
  
*// Config with more expressive API*  
registerApplication({  
 name: 'app1',  
 app: () => *import*('src/app1/main.js'),  
 activeWhen: '/app1',  
 customProps: {  
 some: 'value',  
 }  
});  
  
start();

Copy

### Using arguments[​](https://single-spa.js.org/docs/configuration#using-arguments)

#### Application name[​](https://single-spa.js.org/docs/configuration#application-name)

The first argument to registerApplication must be a string name.

#### Loading Function or Application[​](https://single-spa.js.org/docs/configuration#loading-function-or-application)

The second argument to registerApplication must be either a function that returns a promise [loading function](https://single-spa.js.org/docs/configuration#loading-function) or the resolved Application.

##### Application as second argument[​](https://single-spa.js.org/docs/configuration#application-as-second-argument)

Optionally for the second argument you can use the resolved Application, consisting of an object with the lifecycle methods. This allows you import the Application from another file or define applications inline in your single-spa-config

*const* application = {  
 bootstrap: () => Promise.resolve(), *//bootstrap function*  
 mount: () => Promise.resolve(), *//mount function*  
 unmount: () => Promise.resolve(), *//unmount function*  
}  
registerApplication('applicationName', application, activityFunction)

Copy

##### Loading function[​](https://single-spa.js.org/docs/configuration#loading-function)

The second argument to registerApplication must be a function that returns a promise (or an ["async function"](https://ponyfoo.com/articles/understanding-javascript-async-await)). The function will be called with no arguments when it's time to load the application for the first time. The returned promise must be resolved with the application. The most common implementation of a loading function is an import call: () => import('/path/to/application.js')

#### Activity function[​](https://single-spa.js.org/docs/configuration#activity-function)

The third argument to registerApplication must be a pure function, the function is provided window.location as the first argument, and returns a truthy value whenever the application should be active. Most commonly, the activity function determines if an application is active by looking at window.location/the first param.

Another way of looking at this is that single-spa is a top-level router that has a lot of applications that have their own sub-router.

single-spa will call each application's activity function under the following scenarios:

* hashchange or popstate event
* pushState or replaceState is called
* [triggerAppChange](https://single-spa.js.org/docs/api#triggerappchange) api is called on single-spa
* Whenever the checkActivityFunctions method is called

#### Custom props[​](https://single-spa.js.org/docs/configuration#custom-props)

The optional fourth argument to registerApplication is [custom props](https://single-spa.js.org/docs/building-applications/#custom-props) that are passed to the application's single-spa lifecycle functions. The custom props may be either an object or a function that returns an object. Custom prop functions are called with the application name and current window.location as arguments.

Create a new app --- (ng new my-app)

Switch to directory

Add Single spa –(ng add single-spa-angular)

Changing port manually ,

1 Angular.json --->deploy url

2.package .json 🡪 port no change accordingly .

ng new schedule-control --routing --prefix=schedule-control to create new app

Error Invalid configuration object. Webpack has been initialized using a configuration object that does not match the API schema.

Graphical user interface, text, application, email

Description automatically generated