

[Open in app](#)[Get started](#)Ole Ersoy · [Follow](#)

Feb 15, 2019 · 2 min read



Designing Angular Components For External Configuration

Photo by [Federico Vitale](#) on [Unsplash](#)

Scenario

We want users of our component to be able to be able to declare it as an NPM dependency and a design time configuration for it.

To demo this we the Stackblitz `AppComponent` as the component we are depending on, but



[Open in app](#)[Get started](#)

Inside the `AppComponent` create an exported `InjectionToken` key that will be used by the module to configure the provider.

Side Note

What we are doing here is creating a bridge via a shared key value (The `InjectionToken` instance) that allows the `AppModule` to know how to go about providing the configuration object to our component. The `AppModule.providers` array is used to configure the provision of the configuration via the shared key.

Configure the AppModule

Import the `InjectionKey` export from the `AppComponent` into the `AppModule` and use it to create the provider. Here's a stackblitz demo:

The image shows a StackBlitz demo environment. On the left, the `app.module.ts` file is open, showing the following code:

```
21
22 @NgModule({
23   providers: [config, tokenBasedConfig],
24   imports:   [ BrowserModule, FormsModule ],
25   declarations: [ AppComponent ],
26   bootstrap: [ AppComponent ]
27 })
28 export class AppModule { }
29
```

On the right, the application output is displayed, showing the text "Hello Angular" and "Configure the AppComponent". Below the output, there is a "Console" section.

The `AppComponent` will log the injected `config` object.

Summary



[Open in app](#)[Get started](#)

declared in the component and uses it to specify the configuration in the **NgModule.providers** array.

Follow Up

If we wanted to do this more formally we could create a module for the component and place a `forRoot` static method on the module that also performs the configuration of the providers. That way we are symmetric with the approach used for the `RouterModule`.

This is how that's done (We are assuming that we also created a configuration interface called `ConfigurationInterface`):

```
static forRoot(config:ConfigurationInterface): ModuleWithProviders {  
  
  return {  
    ngModule: MyComponentModule  
    provider: [{ provider: tokenConfigKey, config}]  
  }  
}
```

