Standalone Components

CHEATSHEET

Alain Chautard | @AlainChautard

Since Angular 14, Angular developers can write components, pipes, and directives that do not belong to any specific modules. Those are called **standalone** features. We will refer to them as standalone components throughout this document, but remember that those can also be standalone pipes and standalone directives.

1. How to create a standalone feature with the Angular CLI?

```
ng generate component MyComponentName --standalone
```

It also works with **`ng generate`** pipe and **`ng generate`** directive.

2. How to turn an existing component into a standalone component?

Use the **`standalone: true`** decorator option, then remove your component class from its current module:

```
@Component{(
  selector: 'app-hello',
  standalone: true,
  template: '...'
 )}
export class HelloComponent
```

3. How to add template dependencies to a standalone component?

If your component has template dependencies (pipes, directives, or other components), you can import those using their module or individually if they are standalone components using the **`imports`** array:

```
@Component{(
  selector: 'my-app',
  standalone: true,
  imports: [CommonModule, HelloComponent],
  template: `
   <h1 *ngIf="name">Hello from {{name}}!</h1>
    <app-hello></app-hello>
  `
 )}
export class HelloComponent
```

4. How to bootstrap an entire application with standalone components?

Use the **bootstrapApplication** function (from **@angular/platform-browser**) and pass the application root standalone component as a parameter:

```
bootstrapApplication(App);
```

5. How to configure dependency injection for standalone components?

Use the array of providers in your standalone component decorator:

```
@Component{(
  selector: 'my-app',
  standalone: true,
  imports: [CommonModule, HelloComponent],
  providers: [{provide: UserService}],
```

If you run a module-less application with a standalone component, you can also configure your providers when calling the **bootstrapApplication** function:

```
bootstrapApplication(App, {providers: [{provide:
UserService}] });
```

6. How to include services from existing modules?

If you need to use the providers config from existing modules with your standalone components, you can use the **`importProvidersFrom`** function (from **`@angular/core`**):

```
  providers: [
    importProvidersFrom(AdminModule, CommonModule)
  ]
```

7. How to use standalone components with the Angular router?

When bootstrapping a standalone component, you can also use a module-less approach to your routing config using the **`provideRouter`** function (from **`@angular/router`**):

```
const appRoutes: Routes = [
// Routes go here
];

bootstrapApplication(Appp, {
  providers: [provideRouter(appRoutes)]
});
```

8. How to use lazy-loading with standalone components?

You can lazy-load a standalone component using the **`loadComponent`** option in your route config:

```
{
  path: 'test',
  loadComponent: () => import("src/app/hello/hello.component")
  .then(c => c.HelloComponent)
}
```

If your standalone component is the **default export** of its source file (**`export default class HelloComponent`**), then the above syntax can be simplified into:

```
{
  path: 'test',
  loadComponent: () => import("src/app/hello/hello.component")
}
```

9. How to use lazy-loading with multiple standalone components at once?

You can create a route file without any **`RouterModule`** using this approach, which only works if all routed components are standalone components;

```
// In the main route config:

export const ROUTES: Route[] = [
  {path: 'app', loadChildren: () => import('./routes')
  .then(mod => mod.ALL_ROUTES)},
// ...
];
```

```
// In ./routes.ts

export const ALL_ROUTES: Route[] = [
  {path: 'hello', component: HelloComponent},
  {path: '', component: AppComponent},
// ...
]
```

10. How to provide scoped services for a given route?

You can add an array of **`providers`** to add providers that are scoped services for that specific route and all child components under it:

```
{
  path: 'test',
  providers: [UserService],
  children: [
    {path: 'hello', component: HelloComponent},
    {path: 'admin', component: AdminComponent},
  ],
},
```

11. How to support both standalone components and modules at the same time?

If you're a library author, you can give the option to access your components as standalone or as part of a module. All you have to do is use the **`standalone: true`** option in your components' decorator and then make your components part of the module that exports your feature:

```
@NgModule({
  imports: [HelloComponent, AdminComponent],
  exports: [HelloComponent, AdminComponent],
})
export class HelloModule {}
```

Such components can be imported in two different ways as dependencies:

- By importing **`HelloModule`**
- By importing **`HelloComponent`** or **`AdminComponent`** individually in their dependencies