How to reuse common layouts in Angular using Router

Josip Bojčić Follow
Nov 16, 2018 · 7 min read

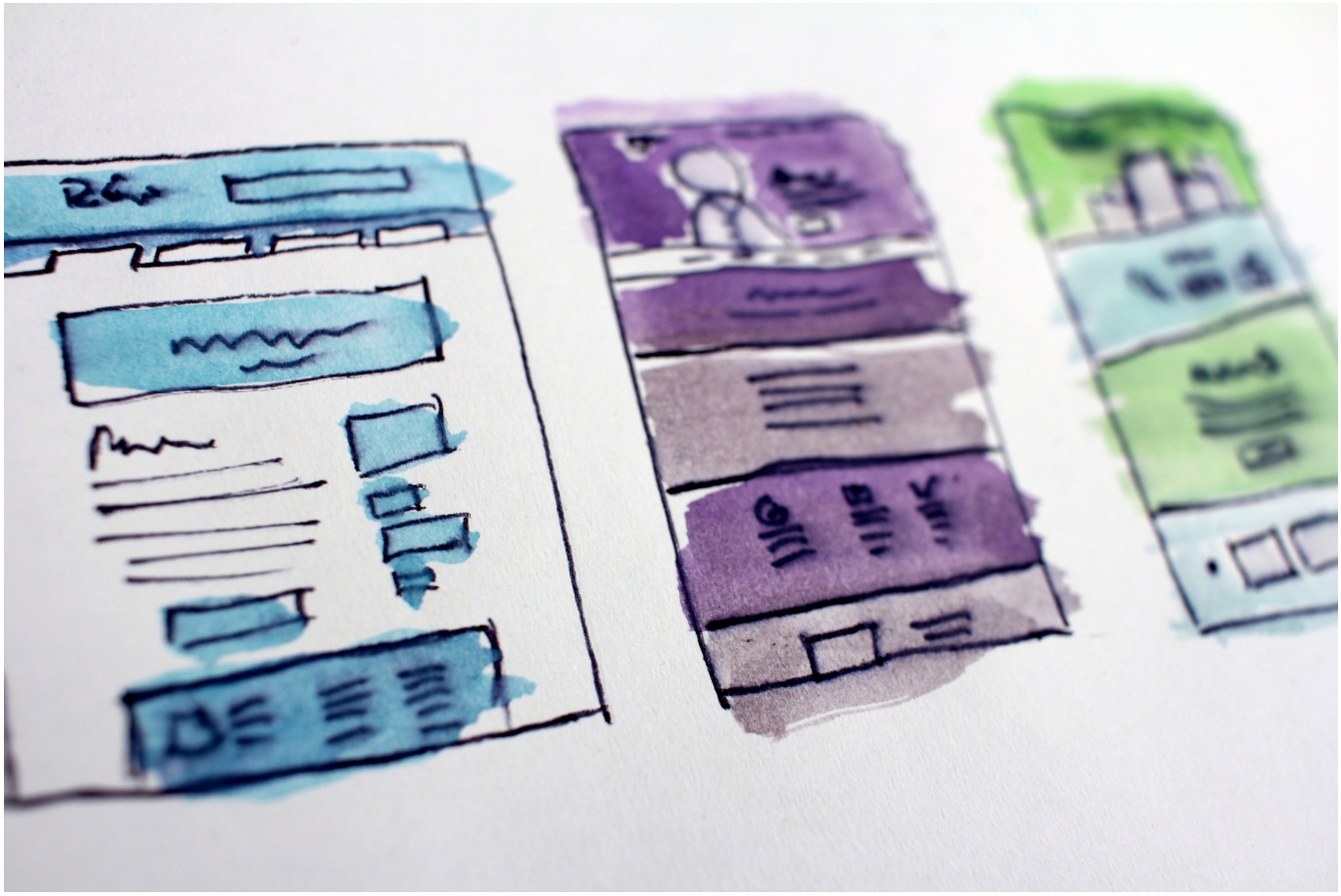


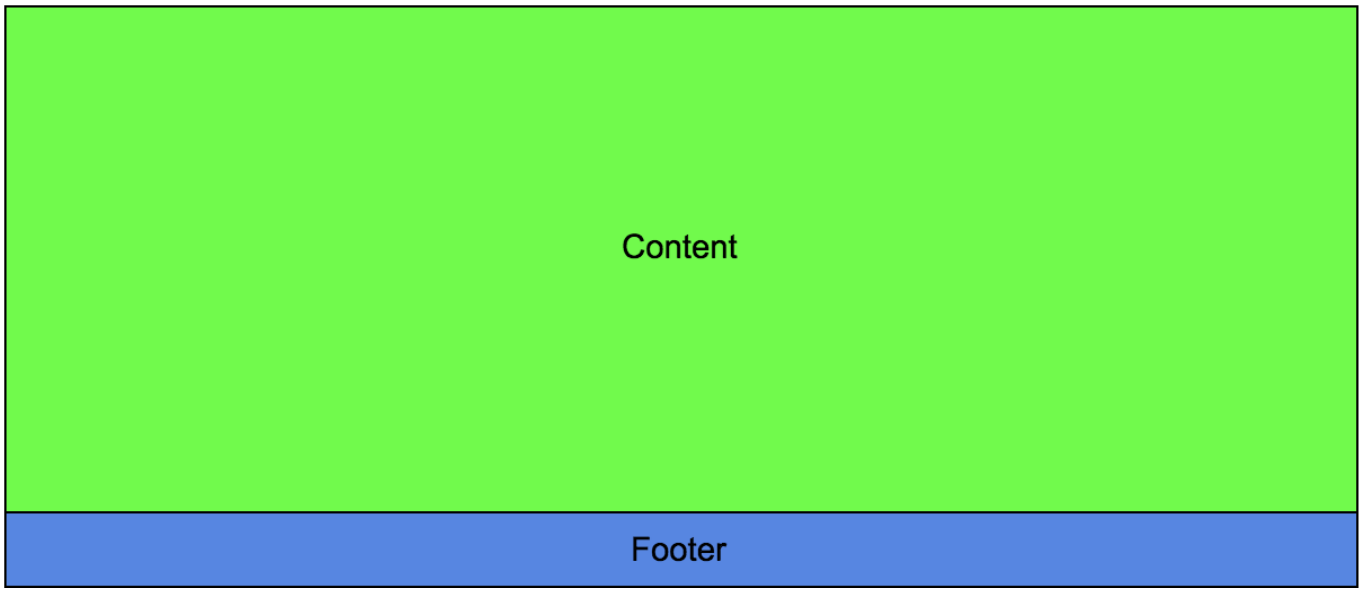
Photo by Hal Gatewood on Unsplash

Most of the web apps I worked on so far, had a design where different pages are using common layout. For example layout which consists of header, footer and sidebar, which are fixed for each page, and the content which varies by the page. Logical idea is to try to extract and reuse common parts. Based on the Angular docs, Pluralsight courses and

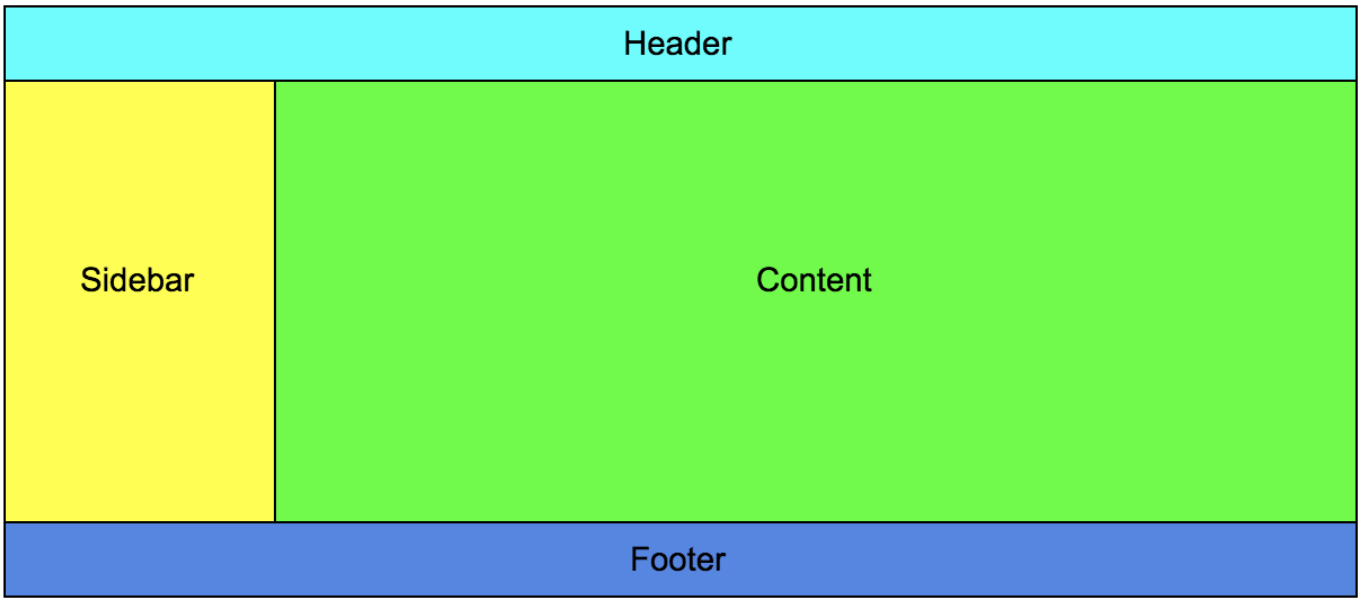
other materials I found, I came up with two possible options. To better explain those options, let's first define example project.

Example Project

Let's say we have a simple app which has 5 different routes/pages (login, registration, dashboard, users, account settings) and two layouts. One layout with content and footer, let's call it **layout 1,** and **layout 2** with header, footer, sidebar and content. Also let's say that login and registration pages have layout 1, while others have layout 2.



Layout 1 — footer only layout



Layout 2 — main layout

The last, we can say that our pages are separate features of the app. Using folder by feature project structure, each of our features will have a separate Angular module with corresponding routing module.

Option 1

(*You can play with it here*)

Layout is defined as a component in a separate module, and use it as a parent component in routing module of each specific feature.

First in the root component template (usually AppComponent) use only `<router-outlet>` like:

```
<router-outlet></router-outlet>
```

Then define `FooterOnlyLayoutComponent` component for layout 1 with following template:

```
<div class="content" fxFlex>
 <router-outlet></router-outlet>
</div>

<app-footer></app-footer>
```

Finally, to use this layout for the login route, route has to be specified like:

```
...

const routes: Routes = [
 {
   path: 'login',
   component: FooterOnlyLayoutComponent,
   children: [
     { path: '', component: LoginComponent },
   ]
 }
];
```

```
@NgModule({
 imports: [RouterModule.forChild(routes)],
 exports: [RouterModule]
})
export class LoginRoutingModule { }
```

This way, when user navigates to `/login`, `FooterOnlyLayoutComponent` will be rendered in the `AppComponent`'s "router slot", while `LoginComponent` will be rendered in the `FooterOnlyLayoutComponent`'s router slot. To make the registration page use `FooterOnlyLayoutComponent`, define the route in the same way while providing registration path and component instead of the login.

For layout 2 component (`MainLayoutComponent`) we have the following template:

```
<app-header fxLayout="column"></app-header>

<div fxLayout="row" fxFlex="100">
 <app-sidebar fxLayout="column" fxFlex="300px"></app-sidebar>
 <div class="content" fxLayout="column" fxFlex>
   <router-outlet></router-outlet>
 </div>
</div>

<app-footer fxLayout="column"></app-footer>
```

To use this layout for the dashboard page, in the dashboard routing module specify route like this:

```
...

const routes: Routes = [
 {
   path: 'dashboard',
   component: MainLayoutComponent,
   children: [
     { path: '', component: DashboardComponent }
   ]
 }
];

@NgModule({
```

```
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class DashboardRoutingModule { }
```

Now, when user navigates to `/dashboard`, `MainLayoutComponent` will be rendered in the `AppComponent`'s "router slot", while `DashboardComponent` will be rendered in the `MainLayoutComponent`'s router slot. To make other pages use this layout, specify their routes in the same way in their corresponding routing modules.

That's it. Now we were able to reuse layout between multiple modules. Login and registration routes are using layout 1 (`FooterOnlyLayoutComponent`), while dashboard, users and account settings routes are using layout 2 (`MainLayoutComponent`).

Issues

Problem with this approach is that layout is unnecessarily recreated on each route change. We can check that by putting console logs in the constructors of the layout, header, footer and sidebar component. If you first go the the `/dashboard` page, check console, and then go to the `/users`, you will see that constructors are called twice.

Other than performance implications this brings another layer of complexity if there is some state that needs to be persisted between routes. Let's say our header has a search input and user typed something in, when he switches to another page, header will be recreated and input cleared. Of course this can be handled by persisting state to some storage but that's still unnecessary complexity.

Option 2 — use lazy loaded modules

(*You can play with it here*)

Define layout as a component in a separate module with routing. Let's call that module `LayoutModule`. Define all feature modules as lazy loaded children modules inside `LayoutModule`.

Again, in the root component template (`AppComponent`) use only `<router-outlet>`. Both layout 1 (`FooterOnlyLayoutComponent`) and layout 2 (`MainLayoutComponent`) have same templates like in the option 1.

Do not import feature modules in the `AppModule`. Instead, we'll import them lazily in the `LayoutRoutingModule`:

```
…

const routes: Routes = [
 {
   path: '',
   redirectTo: '/dashboard',
   pathMatch: 'full'
 },
 {
   path: '',
   component: MainLayoutComponent,
   children: [
     { path: 'dashboard', loadChildren:
'../dashboard/dashboard.module#DashboardModule' },
     { path: 'users', loadChildren:
'../users/users.module#UsersModule' },
     { path: 'account-settings', loadChildren: '../account-
settings/account-settings.module#AccountSettingsModule' },
   ]
 },
 {
   path: '',
   component: FooterOnlyLayoutComponent,
   children: [
     { path: 'login', loadChildren:
'../login/login.module#LoginModule' },
     { path: 'registration', loadChildren:
'../registration/registration.module#RegistrationModule' }
   ]
 },
];

@NgModule({
 imports: [RouterModule.forChild(routes)],
 exports: [RouterModule]
})
export class LayoutRoutingModule { }
```

Lastly, in the routing module of each feature module just use empty path and the component. For example for login, routes would be:

```
const routes: Routes = [
 { path: '', component: LoginComponent }
```

```
];
```

while for the dashboard it's:

```
const routes: Routes = [
 { path: '', component: DashboardComponent }
];
```

and we are done.

Again login and registration are using `FooterOnlyLayoutComponent` while other routes are using `MainLayout`. However this time we avoided recreating layout, header, footer and sidebar on each route change. If you put console logs in the constructors again you will see that now layouts are re-created only when you navigate between routes from different layouts. So if you navigate from `/dashboard` to `/users` layout won't be recreated, while if you go from `/dashboard` to `/login` it will.

Issues

Smaller problem is that all lazy loaded modules and their base paths have to be defined in `LayoutRoutingModule` so it can become messy for larger projects. Bigger issue is that we have to use lazy loading while sometimes maybe you don't want to. It should be possible to reuse layouts similarly without forcing lazy loaded modules. I tried to go around this by specifying loadChildren like this:

```
...

const routes: Routes = [
 {
   path: '',
   redirectTo: '/dashboard',
   pathMatch: 'full'
 },
 {
   path: '',
   component: MainLayoutComponent,
   children: [
     { path: 'dashboard', loadChildren: () => DashboardModule },
     { path: 'users', loadChildren: () => UsersModule },
     { path: 'account-settings', loadChildren: () =>
```

```
    AccountSettingsModule },
      ]
   },
   {
     path: '',
     component: FooterOnlyLayoutComponent,
     children: [
       { path: 'login', loadChildren: () => LoginModule },
       { path: 'registration', loadChildren: () => RegistrationModule }
     ]
   },
  ];

  @NgModule({
   imports: [RouterModule.forChild(routes)],
   exports: [RouterModule]
  })
  export class LayoutRoutingModule { }
```

but this only works if you don't use AOT, which is something we definitely want to use in production (https://github.com/angular/angular-cli/issues/4192).

Another possible solution would be to preload all lazy loaded modules by specifying preload strategy in `AppModule` like:

```
RouterModule.forRoot([], { preloadingStrategy: PreloadAllModules })
```

but with this modules are bundled separately and you end up with multiple files that client needs to fetch which is something you maybe don't want. Also this is not appropriate if you want to lazy load only some specific modules. In that case you may want to write custom preload strategy but you'll still end up with file for each module.

How this was done with AngularJs and UI-Router

(Try it out here)

This was lot easier to achieve with AngularJs and UI-Router, using named views. There we first need to define abstract layout state:

```
$stateRegistry.register({
  name: 'layout',
  abstract: true,
  views: {
    '@': {
      templateUrl: 'layout.html',
    },
    'header@layout': {
      component: 'header'
    },
    'sidebar@layout': {
      component: 'sidebar'
    },
    'content@layout': {
      template: ''
    },
    'footer@layout': {
      component: 'footer'
    }
  }
});
```

then `layout.html`:

```
<div class="flex-column" ui-view="header"></div>

<div class="flex-row flex-100">
 <div class="flex-column" ui-view="sidebar"></div>
 <div class="flex-column flex" ui-view="content"></div>
</div>

<div class="flex-column" ui-view="footer"></app-footer>
```

and then when defining state for actual page you need to use layout state as a parent and override specific named view(s). So login state would be:

```
$stateRegistry.register({
  parent: 'layout',
  name: 'login',
  url: '/login',
  views: {
    'content@layout': {
      component: 'login',
    },
```

```
      'header@layout': {
        component: ''
      },
      'sidebar@layout': {
        template: ''
      }
    }
});
```

while dashboard state would be:

```
$stateRegistry.register({
  parent: 'layout',
  name: 'dashboard',
  url: '/dashboard',
  views: {
    'content@layout': {
      component: 'dashboard',
    }
  }
});
```

To define state for the rest of the pages just follow the same pattern.

Once that is done, let's add `console.log` to the `$onDestroy` hook of the each component and navigate between pages. We can see that header, sidebar and footer are not destroyed when navigating between `/users` and `/dashboard`. Even when we navigate between page with main layout and page with footer only layout we will notice that footer is reused.

Conclusion

Even though it's possible to achieve some kind of layout reuse with Angular router, as described above, both approaches seem a bit "hacky" and painful. It's lot easier to achieve it with UI Router, where we are even able to reuse shared components between different layouts, or React's dynamic routing.

. . .

PS if you know any better way to handle this with Angular router, please share in comments :)

. . .

EDIT:

Option 3

Thanks to Alexander Carls and Lars Gyrup Brink Nielsen, who shared their ideas in the comments, we have an option 3 which solves all the issues mentioned above. Idea is to subscribe to the router events and then on each NavigationEnd event you can show/hide pieces of the layout depending on the route. Examples:

Example 1

Example 2 (with lazy loading)

JavaScript Angular Angular Router Routing