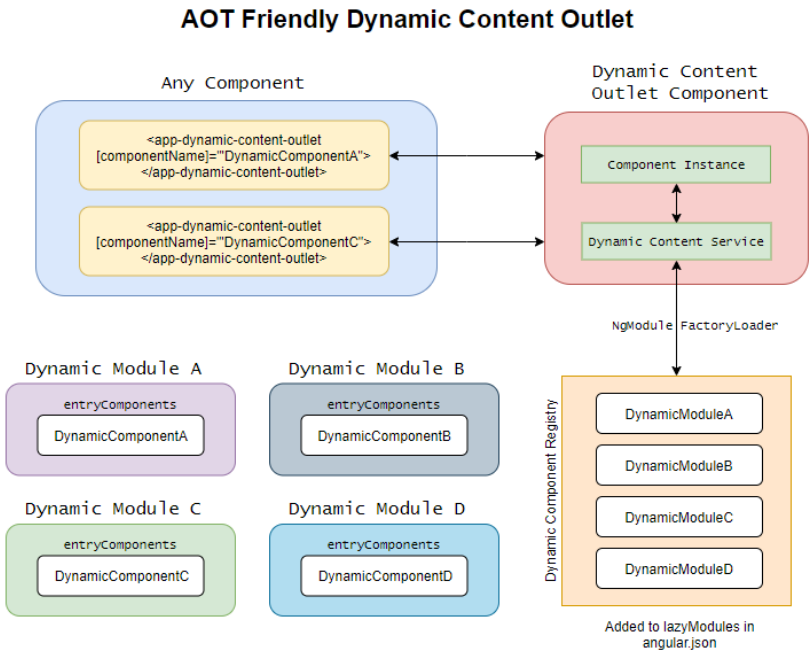
Building an AOT Friendly Dynamic Content Outlet in Angular

Wes Grimes Follow
Jan 21 · 8 min read



Overview — Dynamic Content Outlet

Have you ever needed to dynamically load content or components in your Angular applications? How about in a way that the built-in structural directives—*ngIf* and *ngSwitch*—just don't provide? Are you also in need of the optimization benefits of using Ahead-of-Time compilation?

Well, I have good news for you...(And no you don't have to be Chuck Norris!) If you stay tuned, I will help you get a solution up and

running that will provide a solid way to choose from and load dynamically, at run-time, a set of predefined modules & components in your application.

> *This article is assumes you are building an Angular 6+ application generated using the Angular CLI. For information on using the Angular CLI check out the official documentation.*

> *This arose out of a business need for the company that I work for. What's important to note here is that many articles and examples exist on loading content dynamically in Angular, but none that I found worked reliably when compiling Angular with the `—prod` or `—aot` flags enabled. The good news is that what I describe in this article works fantastically with Ahead-of-Time compiling.*

What We're Going To Do

We're going to build a special module with a dynamic component outlet that can be included and used anywhere in your application. The only requirement is that you register, upfront, an array mapping your dynamic components to their parent modules. You will also add these modules to the `lazyModules` property in your `angular.json` file. By doing so, the compiler will pre-compile these modules. The compiler then splits them off into separate minified chunks and makes them available to the SystemJS loader at runtime, with AOT.

• • •

Let's Build Our Dynamic Content Outlet

Assuming that you have an existing Angular 6+ CLI generated project let's run through the following steps to scaffold the necessary parts that make up this new Dynamic Content Outlet.

Generate the Dynamic Content Outlet Module

Generate a new module named **DynamicContentOutletModule** by running the following command in your shell of choice:

```
$ ng g m dynamic-content-outlet
```

We will come back later to this module and wire things up.

Build the Dynamic Content Outlet Registry

Create a new file underneath the newly created folder `src/app/dynamic-content-outlet` named `dynamic-content-outlet.registry.ts`. This will serve as the placeholder for array mapping component name to module path and module name. For now, it will be an empty array as follows.

```
interface RegistryItem {
  componentName: string;
  modulePath: string;
  moduleName: string;
}
```

```
/**
* A registry array of Component Name to details
* that must be updated with each new component
* that you wish to load dynamically.
*/
```

```
export const DynamicContentOutletRegistry: RegistryItem[] =
[];
```

Build the Dynamic Content Outlet Error Component

Create a new file underneath the folder ***src/app/dynamic-content-outlet/dynamic-content-outlet-error.component.ts***. This will serve as the component to be rendered anytime an error occurs attempting to load a dynamic component. You can customize the **template** property to use any custom styles or layout that you may have. The **errorMessage** input must stay the same and will be fed with the actual details of the error that occurred while attempting to dynamically render your component.

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-dynamic-content-outlet-error-componen
  template: `
    <div>{{ errorMessage }}</div>
  `
})
export class DynamicContentOutletErrorComponent {
```

src/app/dynamic-content-outlet/dynamic-content-outlet-error.component.ts

Build the Dynamic Content Outlet Service

Create a new file underneath the folder `src/app/dynamic-content-outlet/dynamic-content-outlet.service.ts`

- This service encapsulates the logic that loads dynamic components using SystemJS and renders them into the Dynamic Content Outlet.
- It uses the `DynamicContentOutletRegistry` to lookup the module by `componentName`.
- It also makes use of a new `static` property that we will add later on to each module we wish to dynamically load named `dynamicComponentsMap`. This allows us to get the type literal for the given `componentName` so that the `resolveComponentFactory` can instantiate the correct component. You might ask why we didn't just add a fourth property to the `DynamicContentOutletRegistry`, well this is because if we import the type in the registry, then it defeats the purpose of lazy loading these modules as the the type will be included in the main bundle.
- If an error occurs, a `DynamicContentOutletErrorComponent` is rendered instead with the error message included.

```
import {
  ComponentFactoryResolver,
  ComponentRef,
  Injectable,
  Injector,
  NgModuleFactoryLoader,
  Type
} from '@angular/core';
import { DynamicContentOutletErrorComponent } from './
import { DynamicContentOutletRegistry } from './dynami

type ModuleWithDynamicComponents = Type<any> & {
  dynamicComponentsMap: {};
};

@Injectable()
export class DynamicContentOutletService {
  constructor(
    private componentFactoryResolver: ComponentFactory
    private moduleLoader: NgModuleFactoryLoader,
    private injector: Injector
  ) {}

  async GetComponent(componentName: string): Promise<C
    const modulePath = this.getModulePathForComponent(

    if (!modulePath) {
      return this.getDynamicContentErrorComponent(
        `Unable to derive modulePath from component: $
      );
    }

    try {
      const moduleFactory = await this.moduleLoader.lo
      const moduleReference = moduleFactory.create(thi
      const componentResolver = moduleReference.compon

      const componentType = (moduleFactory.moduleType
        .dynamicComponentsMap[componentName];

      const componentFactory = componentResolver.resol
        componentType
      );
      return componentFactory.create(this.injector);
```

```
45      } catch (error) {
46        console.error(error.message);
47        return this.getDynamicContentErrorComponent(
48          `Unable to load module ${modulePath}.
49                  Looked up using component: ${component
50            error.message
```

src/app/dynamic-content-outlet/dynamic-content-outlet.service.ts

Build the Dynamic Content Outlet Component

Create a new file underneath the folder `src/app/dynamic-content-outlet/dynamic-content-outlet.component.ts`. This component takes an input property named `componentName` that will call the `DynamicContentOutletService.GetComponent` method passing into it `componentName`. The service then returns an instance of that rendered and compiled component for injection into the view. The service returns an error component instance if the rendering fails for some reason. The component listens for changes via the `ngOnChanges` life-cycle method. If the `@Input() componentName: string;` is set or changes it automatically re-renders the component as necessary. It also properly handles destroying the component with the `ngOnDestroy` life-cycle method.

```
import {
  Component,
  ComponentRef,
  Input,
  OnChanges,
  OnDestroy,
  ViewChild,
  ViewContainerRef
} from '@angular/core';
import { DynamicContentOutletService } from './dynamic

@Component({
  selector: 'app-dynamic-content-outlet',
  template: `
    <ng-container #container></ng-container>
  `
})
export class DynamicContentOutletComponent implements
  @ViewChild('container', { read: ViewContainerRef })
  container: ViewContainerRef;

  @Input() componentName: string;

  private component: ComponentRef<{}>;

  constructor(private dynamicContentService: DynamicCo

  async ngOnChanges() {
    await this.renderComponent();
  }

  ngOnDestroy() {
    this.destroyComponent();
```

src/app/dynamic-content-outlet/dynamic-content-outlet.component.ts

Finish Wiring Up Parts To The Dynamic Content Outlet Module

Make sure your `src/app/dynamic-content-outlet/dynamic-content-outlet.module.ts` file looks like the following:

```
import { CommonModule } from '@angular/common';
import {
  NgModule,
  NgModuleFactoryLoader,
  SystemJsNgModuleLoader
} from '@angular/core';
import { DynamicContentOutletErrorComponent } from './
import { DynamicContentOutletComponent } from './dynam
import { DynamicContentOutletService } from './dynamic

@NgModule({
  imports: [CommonModule],
  declarations: [
    DynamicContentOutletComponent,
    DynamicContentOutletErrorComponent
  ],
  exports: [DynamicContentOutletComponent],
```

src/app/dynamic-content-outlet/dynamic-content-outlet.module.ts

. . .

Let's Use Our New Dynamic Content Outlet

Phew! Take a deep breath and grab a cup of coffee (french press fair trade organic dark roast). The hard work is behind you. Next we will go through the process of actually putting this new module into play!

For any component that you would like dynamically rendered you need to do the following four steps. ***These steps must be followed exactly.***

1. Prepare your module for dynamic import

- Confirm that the component is listed in the `entryComponents` array in the module that the component is a part of.
- Add to the module, a new `static` property called `dynamicComponentsMap`. This allows us to get the type literal for the given `componentName` so that the `resolveComponentFactory` can instantiate the correct component.

> *You might ask why we didn't just add a fourth property to the* `DynamicContentOutletRegistry` *named* `componentType` *; well this is because if we import the type in the registry, then it defeats the purpose of lazy loading these modules as the the type will be included in the main bundle.*

A prepared module might look as follows:

```
import { CommonModule } from '@angular/common';
import { NgModule } from '@angular/core';
import { DynamicMultipleOneComponent } from './dynamic
import { DynamicMultipleTwoComponent } from './dynamic

@NgModule({
  declarations: [MySpecialDynamicContentComponent],
  imports: [CommonModule],
  entryComponents: [MySpecialDynamicContentComponent]
})
export class MySpecialDynamicContentModule {
```

2. Add your dynamic component(s) to the registry

For any component that you would like dynamically rendered, add a new entry to the `DynamicContentOutletRegistry` array in `src/app/dynamic-content-outlet/dynamic-content-outlet.registry.ts`.

The following properties must be filled out:

- **componentName**: This should match exactly the name of the Component you wish to load dynamically.
- **modulePath**: The absolute path to the module containing the component you wish to load dynamically. This is only the path to the module and does NOT include moduleName after a `#`.
- **moduleName**: This is the exact name of the module.

Example Component Mapping

```
{
  componentName: 'MySpecialDynamicContentComponent',
  modulePath: 'src/app/my-special-dynamic-content/my-
special-dynamic-content.module',
  moduleName: 'MySpecialDynamicContentModule'
},
```

3. Add your dynamic modules to the lazyModules array

In your `angular.json` update the `projects > ** > architect > build > options > lazyModules` array and add an item for each module that

you added to the registry in order for the Angular AOT compiler to detect and pre-compile your dynamic modules. If you have multiple projects in a folder, make sure you add this for the correct project you are importing and using dynamic modules in. The updated file will look similar to this:

```
{
  ...
  "projects": {
    "angular-dynamic-content": {
      ...
      "architect": {
        "build": {
          "builder": "@angular-devkit/build-
angular:browser",
          "options": {
            ...
            "lazyModules": ["src/app/my-special-dynamic-
content/my-special-dynamic-content.module"]
          },
        }
      }
    }
  }
}
```

Wire up the Dynamic Content Outlet Module

Up to this point you have created your dynamic content outlet module and registered your components to be available in the outlet. The last thing we need to do is wire up our new `DynamicContentOutletModule` to be used in our application. In order to do so you need to:

1. Add your new `DynamicContentOutletModule` to the `imports` array of any feature module or the main `AppModule` of your Angular application.

Example of addition to the `imports` array

```
@NgModule({
...
imports: [
   ...
   DynamicContentOutletModule
  ],
  ...
})
```

```
export class AppModule {}
```

2. Add the following tag to the template of the parent component that you would like to render the dynamic content in:

```
<app-dynamic-content-outlet [componentName]="'MyComponent'">
</app-dynamic-content-outlet>
```

This is very similar in nature to Angular's built-in `<router-outlet></router-outlet>` tag.

3. Happy `ng serve --prod` ing!

Conclusion

Hopefully you have found this solution helpful. Here is the full GitHub repository example for you to clone and play around with. PR's are welcome, appreciated, encouraged and accepted!

Real-World Complex Example

If you are interested in a more in-depth real-world example, then check out the Github Repository which will demonstrate the following:

- Dynamic modules with multiple components
- Demonstrating the use of on-the-fly component changes
- Demonstrating that the scoped styles are loaded dynamically for each component

GitHub Repository Example

Additional Resources

I would highly recommend enrolling in the Ultimate Angular courses. It is well worth the money and I have used it as a training tool for new Angular developers. Follow the link below to signup.

Ultimate Angular™ | Expert-led online Angular and TypeScript training courses

Learn latest Angular, TypeScript, through to NGRX and beyond. Become an Angular...

ultimatecourses.com

• • •

Special Thanks

I want to take a moment and thank all those I was able to glean this information from. I did not come up with all this on my own, but I was able to get a working solution by combining parts from each of these articles!

Dynamically Loading Components with Angular CLI

blog.angularindepth.com

Here is what you need to know about dynamic components in Angular

Create Angular components dynamically like a pro

blog.angularindepth.com

The Need for Speed: Lazy Load Non-Routable Modules in Angular 🏎

As you probably know, Angular comes with a functionality that allows you to lazy load...

netbasal.com

Also, a huge thank you to Medium reader ivanwonder and Github user Milan Saraiya for pointing this out and providing a fork example of resolution.

ivanwonder (@ivanwond) | Twitter

The latest Tweets from ivanwonder (@ivanwond). The world is so beautiful...

twitter.com

milansar—Overview

milansar has 6 repositories available. Follow their code on GitHub.

github.com