How to Dynamically Create a Component in Angular

In this article, we will learn to create a component dynamically. You may need to load a component dynamically in various scenarios, such as wanting to show a popup modal.

Let us assume that we have a component as listed below, which we will load dynamically.

```
import { Component, Input } from '@angular/core';
@Component({
    selector: 'app-message',
    template: `<h2>{{message}}</h2>
`
})
export class MessageComponent {
    @Input() message: string;
}
```

To load `MessageComponent` dynamically you need a container. Let us say that we want to load `MessageComponent` inside `AppComponent`. We need a container element in the `AppComponent`.

The template for `AppComponent` is as below:

```
<div style="text-align:center">
     <h1>
         Welcome to {{ title }}!
     </h1>
     <template #messagecontainer>
     </template>
 </div>
```

As you can see, we have an entry point template or a container template in which we will load `MessageComponent` dynamically.

In AppComponent, we need to import the following:

1. `ViewChild`, `ViewContainerRef`, and `ComponentFactoryResolver` from `@angular/core`.
2. `ComponentRef` and `ComponentFactory` from `@angular/core`.
3. `MessageComponent` from `message.component`.

After importing the required elements, `AppComponent` will look like the following code block:

```
import {
    Component,
    ViewChild,
    ViewContainerRef,
    ComponentFactoryResolver,
    ComponentRef,
    ComponentFactory
} from '@angular/core';
import { MessageComponent } from './message.component';
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html'
})
export class AppComponent {
    title = 'app';
}
```

We can access template using `ViewChild` inside the Component class. The template is a container in which we want to load the component dynamically. Therefore, we have to access template with `ViewConatinerRef`.

`ViewContainerRef` represents a container where one or more views can be attached. This can contain two types of views.

Host Views are created by instantiating a component using `createComponent` and Embedded Views are created by instantiating an Embedded Template using`createEmbeddedView`. We will use Host Views to dynamically load `MessageComponent`.

Let us create a variable called `entry` which will refer to the template element. In addition, we have injected `ComponentFactoryResolver` services to the component class, which will be needed to dynamically load the component.

```
export class AppComponent {
    title = 'app';
    @ViewChild('messagecontainer', { read: ViewContainerRef }) entry: ViewC
ontainerRef;
    constructor(private resolver: ComponentFactoryResolver) { }
}
```

Keep in mind that the `entry`variable, which is a reference to a template element, has an API to create components, destroy components, etc.

To create a component, let us first create a function. Inside the function, we need to perform the following tasks:

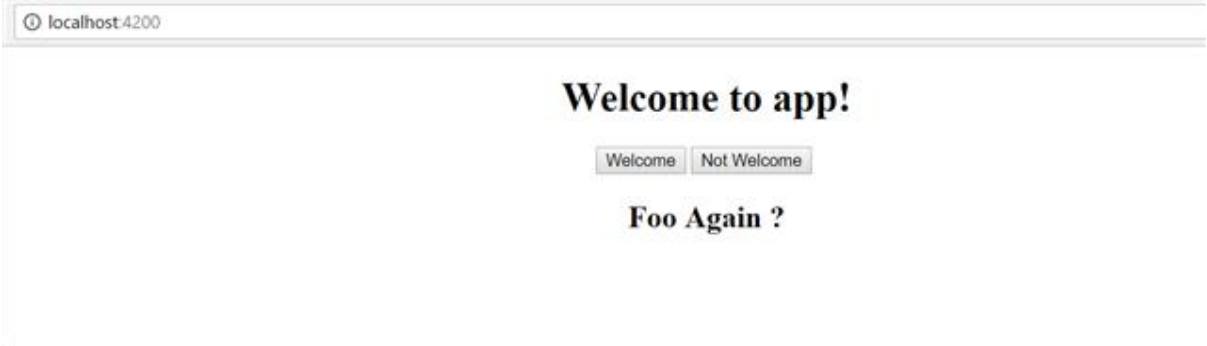
- Clear the container.
- Create a factory for `MessageComponent`.
- Create a component using the factory.
- Pass the value for `@Input` properties using a component reference instance method.

Putting everything together, the `createComponent` function will look like the below code:

```
createComponent(message) {
    this.entry.clear();
    const factory = this.resolver.resolveComponentFactory(MessageComponent)
;
    const componentRef = this.entry.createComponent(factory);
    componentRef.instance.message = message;
}
```

We can call the `createComponent` function on click event attached to the button. Let us put two buttons in the template and call the `createComponent` function when a button is clicked.

In the output, you can see that component is getting loaded dynamically on the click of the button.



As you click on the buttons, the component will be reloaded with a different message. You can destroy a component using the `destroy` method on the `componentRef`.

```
destroyComponent() {
    this.componentRef.destroy();
}
```

You can either destroy a dynamically loaded component by manually calling the function or by putting it inside the `ngOnDestroy()` lifecycle hook of the component so that when the host component is destroyed, the dynamically loaded component will also be destroyed.

Putting everything together, `AppComponent` will look like the code below:

```
import {
    Component,
    ViewChild,
    ViewContainerRef,
    ComponentFactoryResolver,
    ComponentRef,
    ComponentFactory
} from '@angular/core';
import { MessageComponent } from './message.component';
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html'
})
export class AppComponent {
    title = 'app';
    componentRef: any;
    @ViewChild('messagecontainer', { read: ViewContainerRef }) entry: ViewC
ontainerRef;
    constructor(private resolver: ComponentFactoryResolver) { }
    createComponent(message) {
        this.entry.clear();
        const factory = this.resolver.resolveComponentFactory(MessageCompon
ent);
        this.componentRef = this.entry.createComponent(factory);
        this.componentRef.instance.message = message;
    }
    destroyComponent() {
        this.componentRef.destroy();
    }
}
```

At this point, when running the application, you will get an error because we have not set the `entryComponents` in `AppModule`. We can set that as shown in the listing below:

```
import { AppComponent } from './app.component';
import { MessageComponent } from './message.component';
@NgModule({
    declarations: [
        AppComponent, MessageComponent
    ],
    imports: [
        BrowserModule
    ],
    providers: [],
    bootstrap: [AppComponent],
    entryComponents: [MessageComponent]
})
export class AppModule { }
```

This is all you need to do to load a component dynamically in Angular.