Introduction to Angular 2 Routing

coryrylan.com/blog/introduction-to-angular-2-routing

Angular 2 brings many improved modules to the Angular ecosystem including a new router called the Component Router. The Component Router is a highly configurable and feature packed router. Features included are standard view routing, nested child routes, named routes, and route parameters. This post we will cover standard routing, route parameters and nested child routes. With these basics we can build a great navigation experience for users that is easy to reason about. This post has been updated to the new RC router (3.0.0-alpha.7) that is documented on the Angular website.

Basic Routing

So now lets take a look at what our first rendered view will look like.

We will start with single app component that has two routes. We will have a home view and a about view. Lets take a look at these two components first.

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

@Component({
  selector: 'app-home',
  template: `<h2>Home</h2>`
})
export class Home { }

@Component({
  selector: 'app-about',
  template: `<h2>About</h2>`
})
export class About { }
```

So we can see that our two components are very simple just displaying some text. Now lets look at our app component and see how we can use the new Router to route between these two components.

```
import { Component } from '@angular/core';
import { ROUTER_DIRECTIVES } from '@angular/router';

@Component({
  selector: 'demo-app',
  template: `
    <a [routerLink]="['/']">Home</a>
    <a [routerLink]="['/about']">About</a>
    <div class="outer-outlet">
      <router-outlet></router-outlet>
    </div>
  `,

  directives: [ROUTER_DIRECTIVES]
})
export class AppComponent { }
```

So lets walk through our app component step by step and see what this code is doing.

```
import { ROUTER_DIRECTIVES } from '@angular/router';
```

So this import is pulling in the Route Directives that will be used on this view. We must list this it our component's `directives` list.

```
@Component({
  selector: 'demo-app',
  template: `
    <a [routerLink]="['/']">Home</a>
    <a [routerLink]="['/about']">About</a>
    <div class="outer-outlet">
      <router-outlet></router-outlet>
    </div>
  `,

  directives: [ROUTER_DIRECTIVES]
})
```

So the first part is the `[routerLink]`. This directive generates our link based on the route path. The second part is the `router-outlet`, this is the location where Angular will insert the component we want to route to on the view.

Next lets take a look at our route config file `routes.app.ts`.

```
import { provideRouter, RouterConfig } from '@angular/router';

import { AboutComponent } from 'app/about.component';
import { HomeComponent } from 'app/home.component';

export const routes: RouterConfig = [
  { path: '', component: HomeComponent }
  { path: 'about', component: AboutComponent }
];

export const APP_ROUTER_PROVIDERS = [
  provideRouter(routes)
];
```

Our route config defines all the routes in our application. The first route is our default home route. The second one is our `AboutComponent`. The path value is the path that we referenced in our template. We export our routes as a `Provider` to bootstrap into our application.

```
import { bootstrap }    from '@angular/platform-browser-dynamic';
import { APP_ROUTER_PROVIDERS } from './app.routes';
import { AppComponent } from './app.component';

bootstrap(AppComponent, [
  APP_ROUTER_PROVIDERS
]);
```

Now that our routes are registered we have our standard routing working. Next lets look at nested routes.

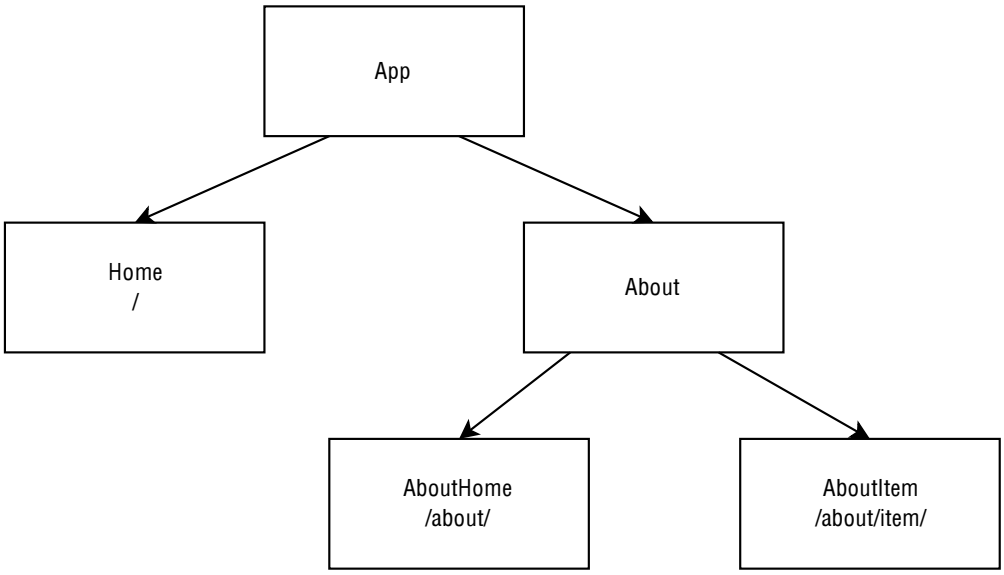
Nested Child Routes

Child/Nested routing is a powerful new feature in the new Angular 2 router. We can think of our application as a tree structure, components nested in more components. We can think the same way with our routes and URLs.

So we have the following routes, `/` and `/about`. Maybe our about page is extensive and there are a couple of different views we would like to display as well. The URLs would look something like `/about` and `/about/item`. The first route would be the default about page but the more route would offer another view with more details.

So lets take a look at what that looks like rendered out.

So as above we can see the About view has its own `router-outlet` highlighted in blue. The about view also has its own links that navigate between two nested about child components. We can think of this as a tree structure.



So lets take a look at our About components.

```
import { Component } from '@angular/core';
import { ROUTER_DIRECTIVES } from '@angular/router';

@Component({
  selector: 'about-home',
  template: `<h3>About Home</h3>`
})
export class AboutHomeComponent { }

@Component({
  selector: 'about-item',
  template: `<h3>About Item</h3>`
})
export class AboutItemComponent { }

@Component({
```

```
    selector: 'app-about',
    template: `
      <h2>About</h2>
      <a [routerLink]="['/about']">Home</a>
      <a [routerLink]="['/about/item']">Item</a>
      <div class="inner-outlet">
        <router-outlet></router-outlet>
      </div>
    `,
    directives: [ROUTER_DIRECTIVES]
})
export class AboutComponent { }
```

So our About template looks very similar to the App component template. Our about component has two child routes, `about/` and `about/item` These pull in two simple components that just display the rendered text above. Notice our route paths start at the root of the about component. The rendered URLs would be `/about/` and `/about/item`. Lets now take a look at the updated route config.

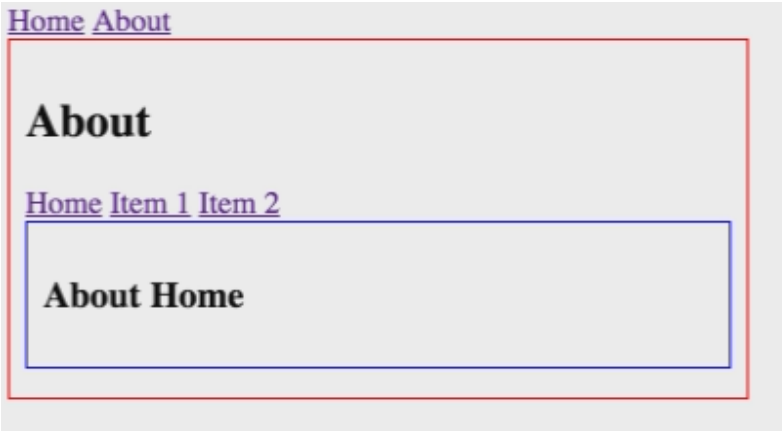
```
import { provideRouter, RouterConfig } from '@angular/router';

import { AboutComponent, AboutHomeComponent, AboutItemComponent } from 'app/about.component';
import { HomeComponent } from 'app/home.component';

export const routes: RouterConfig = [
  { path: '', component: HomeComponent }
  {
    path: 'about',
    component: AboutComponent,
    children: [
      { path: '', component: AboutHomeComponent },
      { path: 'item', component: AboutItemComponent }
    ]
  }
];

export const APP_ROUTER_PROVIDERS = [
  provideRouter(routes)
];
```

Notice our path in our About component is now relative for all the child components. Next we will learn how to dynamically change data in our component via route parameters.

Route Parameters

Building on top of our demo app we are now going to add a component that takes in a route parameter. Route parameters allow us to pass values in our url to our component so we can dynamically change our view content. So

in our example we will have a route that can take an id and then display it on our `AboutItemComponent` component. So lets take a look at what the rendered output would be.



Our URLs would be the following: `/about/`, `/about/item/1`, and `/about/item/2`. We can swap out any number in our URL and our item component can pull that value out and display it in the view. Let's take a look at the code for the root about component.

```
@Component({
    selector: 'app-about',
    template: `
      <h2>About</h2>
      <a [routerLink]="['/about']">Home</a>
      <a [routerLink]="['/about/item', 1]">Item 1</a>
      <a [routerLink]="['/about/item', 2]">Item 2</a>
      <div class="inner-outlet">
        <router-outlet></router-outlet>
      </div>
    `,
    directives: [ROUTER_DIRECTIVES]
})
export class AboutComponent { }
```

So the first part in our about template our new `[routerLink]`'s have a second parameter of the id value ex: `['/about/item', 2]`. This allows us to pass in parameters to the router. For now we are just passing in a number. Now lets take a look at the updated route config.

```
export const routes: RouterConfig = [
  { path: '', component: HomeComponent }
  {
    path: 'about',
    component: AboutComponent,
    children: [
      { path: '', component: AboutHomeComponent },
      { path: 'item/:id', component: AboutItemComponent }
```

```
    ]
  }
];
```

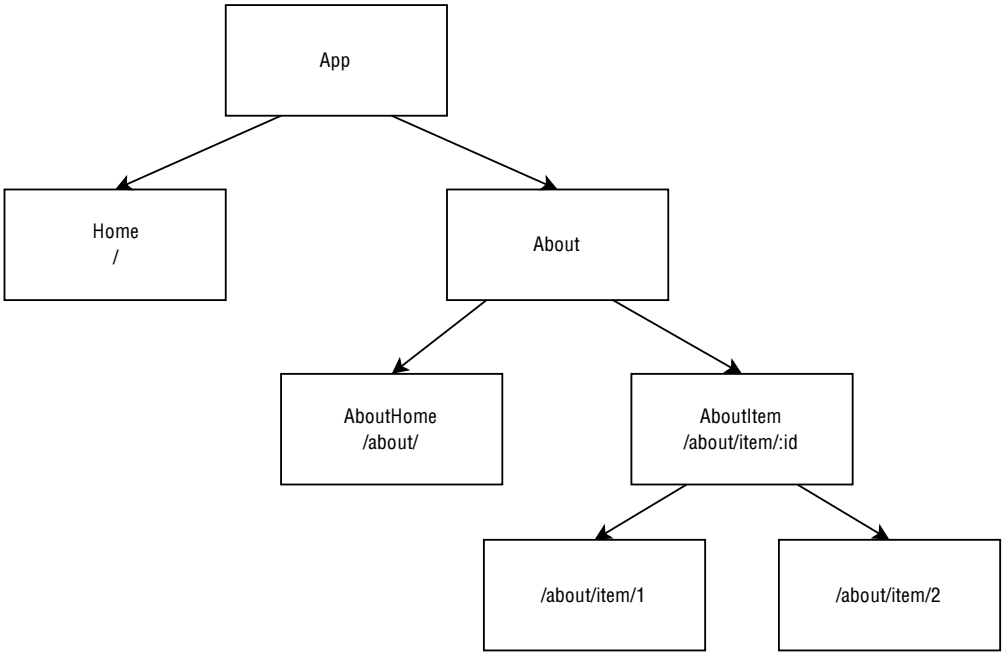
Now taking a look at the item route we now have `'item/:id'` the `:` denotes that this is a route parameter and the r outer should get the value in the URL. So now lets take a look at the `AboutItemComponent` and see how we can get the id from the URL.

```
import { ActivatedRoute } from '@angular/router';

@Component({
  selector: 'about-item',
  template: `<h3>About Item Id: {{id}}</h3>`
})
export class AboutItemComponent {
  id: any;
  paramsSub: any;

  constructor(private activatedRoute: ActivatedRoute) { }

  ngOnInit() {
    this.paramsSub = this.activatedRoute.params.subscribe(params => this.id =
parseInt(params['id'], 10));
  }

  ngOnDestroy() {
    this.paramsSub.unsubscribe();
  }
}
```

We import the `ActivatedRoute` class and inject it into our component. The parameters are wrapped in an Observable that will push the current route parameter value whenever the parameter is updated. We subscribe for any changes. When a new value is recieved we set the value to a property on our template. We could just as easily taken this value as an ID to retrieve some data from a API. We capture the subscription in a property so when the component is destroyed we unsubscribe preventing any memory leaks.

Using Observables to get route params works well when the component persists on the same screen without having to be destroyed and recreated each time. If you are certain your component will be destroyed before a new parameter is updated you can use the `snapshot` api option documented here.

So now lets take a look at our diagram of our application's routes.



Recap

So we learned how to do basic routing between components, child/nested routing and route parameters. With these features mastered you can quickly build large scalable Angular 2 apps in no time. As the new Release Candidate router is documented I will update this post accordingly. Take a look at full working demo in the link below.

Demo