Routing in Angular

By Default, when we run the Angular app let say on port 4200, we get our application running at http://localhost:4200

This is the only path our browser understands, and hence it can't serve different destinations on different paths such as http://localhost:4200/items. But what if we have multiple paths in the same application?

We will need to tell our browser about all the destination and paths to reach then initially.

That's where the **JavaScript Router** comes in action. Means application state will be changed whenever browser URL changes.

In Angular, these routers are written in **@angular/router** package.

How to implement routing?

To tell the paths and destinations to our browser, we need to import the **RouterModule** and **Routes** from **@angular/router** into **app.module.ts** file.

```
import { RouterModule, Routes } from '@angular/router';
```

After this, you can define an array of your paths and destination pages.

```
const routes: Routes = [
{
   path: '',
   component: UserDashboardLayoutComponent,
   children: [
    {
     path: 'wizard',
     loadChildren: './wizard/wizard.module#WizardModule',
     canActivate: [AuthGuard],
     data: { roles: ['Pro', 'Free'] }
    },
    {
     path: 'dashboard',
     loadChildren: './dashboard/dashboard.module#DashboardModule',
     canActivate: [AuthGuard],
     data: { roles: ['Pro', 'Free'] }
    },
}];
```

Lastly, we need to pass the routes array in the **RouterModule**.

```
RouterModule.forRoot(routes);
```

The complete **app.module.ts** file will look like this:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { RouterModule, Routes} from '@angular/router';
import { AppComponent } from './app.component';

const routes: Routes = [
{
   path: '',
   component: UserDashboardLayoutComponent,
   children: [
    {
      path: 'wizard',
      loadChildren: './wizard/wizard.module#WizardModule',
      canActivate: [AuthGuard],
      data: { roles: ['Pro', 'Free'] }
    },
    {
      path: 'dashboard',
      loadChildren: './dashboard/dashboard.module#DashboardModule',
      canActivate: [AuthGuard],
      data: { roles: ['Pro', 'Free'] }
    },
}];

@NgModule({
 declarations: [
   AppComponent,
 ],
 imports: [
   BrowserModule,
   RouterModule.forRoot(routes)
 ],
 providers: [],
 bootstrap: [AppComponent]
})
export class AppModule { }
```

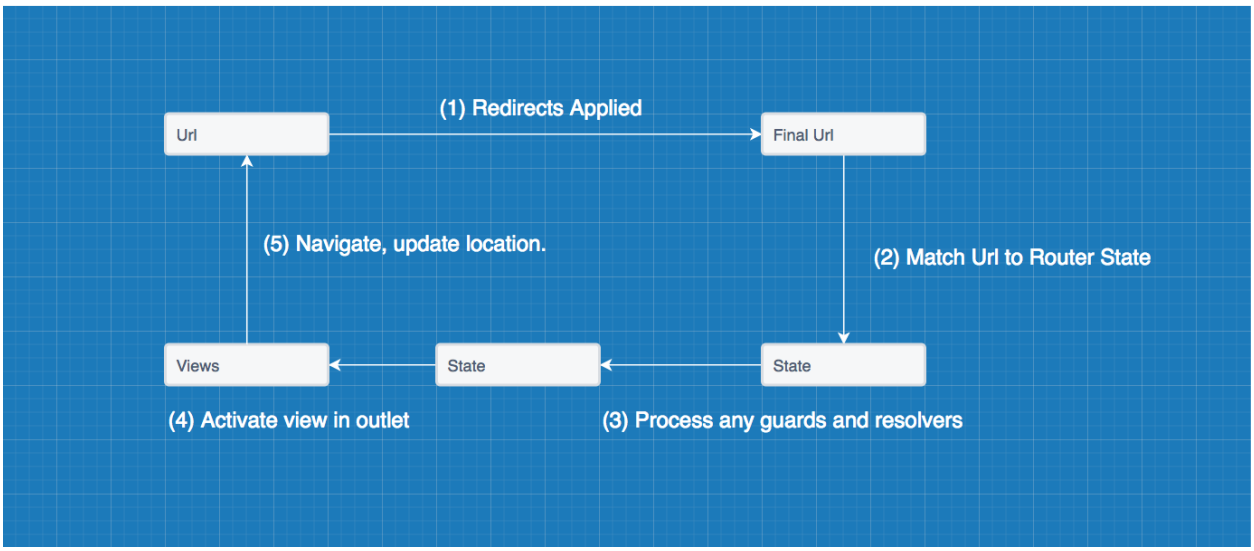
Our Routing configuration is set up, but we need to tell the Angular Application where to place our rendered **AppComponent**.

For that, we will use the **router-outlet**.

Finally, we will write **<router-outlet></router-outlet>** in **app.component.html**. This will ensure that the DOM of the component should be placed here.

```
<div id="wrapper">
  <div class="clearfix"></div>
  <app-header></app-header>
  <app-Menu></app-Menu>
  <!-- MAIN -->
  <div class="main">
    <router-outlet>
    </router-outlet>
  </div>
  <!-- END MAIN -->
  <div class="clearfix"></div>
</div>
```

The lifecycle of a Router



During this **navigation cycle, the router emits a series of events**. Some noteworthy events during this cycle are:

NavigationStart	Represents the start of a navigation cycle.
NavigationCancel	For instance, a guard refuses to navigate to a route.
RoutesRecognized	When a URL has been matched to a route.
NavigationEnd	Triggered when navigation ends successfully.

```
ngAfterViewInit() {
  this.router.events.subscribe(
    (event) => {
      if (event instanceof NavigationStart) {
        popupFunctionObject.showLoader();
      }
      else if (event instanceof NavigationEnd || event instanceof NavigationCancel) {
        popupFunctionObject.hideLoader();
      }
    },
    (error: any) => {
      popupFunctionObject.hideLoader();
    }
  );
 }
```

Lazy Loading Modules

As an application grows over time, more and more of its functionality will be encapsulated in separate feature modules.

It is better to load these modules on demand whenever a user navigates to them, and it is through lazy loading that the Angular router achieves this.

Route Parameters

```
export const routes: Routes = [
 { path: '', redirectTo: 'product-list', pathMatch: 'full' },
 { path: 'product-list', component: ProductList },
 { path: 'product-details/:id', component: ProductDetails }
];
```

```
localhost:4200/product-details/5
```

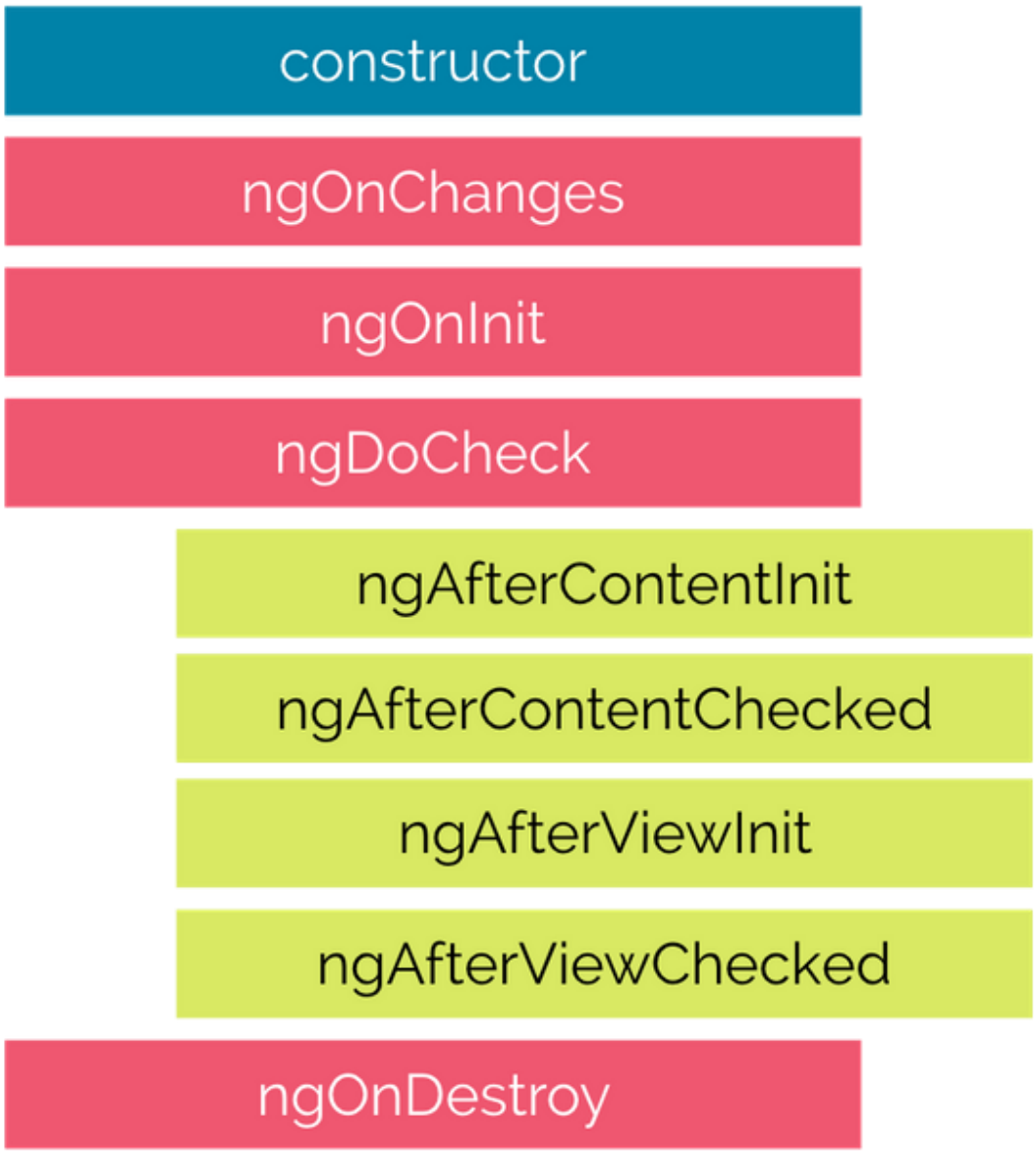
```
ngOnInit() {
 this.sub = this.route.params.subscribe(params => {
   this.id = +params['id']; // (+) converts string 'id' to a number

   // TODO: Add method to be called.
 });
}
```

Angular component lifecycle hooks

Understand the different phases an Angular component goes through from being created to being destroyed.

Know the order in which the different phases happen and what triggers each phase.

The hooks are executed in this order:



constructor

This is invoked when Angular creates a component or directive by calling new on the class.

```
constructor(
   private subscriptionService: SubscriptionService,
   private manageMyAccountService: ManageMyAccountService,
   private errorMessage: ErrorMessage,
   private commonService: CommonService,
   private router: Router,
   private resolver: ComponentFactoryResolver,
   private cdr: ChangeDetectorRef,
   private addressFormat: AddressFormatPipe,
  ) { }
```

ngOnChanges

Invoked every time there is a change in one of th *input* properties of the component.

```
export class OrderFormPayComponent implements OnInit, OnChanges {
  @ViewChild('container', { read: ViewContainerRef }) entry: ViewContainerRef;
  @Input() shoppingCart: ShoppingCart;
  @Input() selectedSubscriptionPriceLink: SubscriptionPriceLink;
  @Input() allSubscriptionPriceLink: SubscriptionPriceLink[] = [];
  @Input() userBillingDetail: UserBillingDetail;
  @Input() isMonthly: boolean;
  @Input() selectedTabIndex: number;
  @Input() invoiceAmount: number;
  @Output() onCancel = new EventEmitter();
  @Output() nextPrevClick = new EventEmitter<number>();
```

```
ngOnChanges(changes: SimpleChanges) {
   if (changes['invoiceAmount'] && this.invoiceAmount) {
     let obj = {
       'invoiceAmountTemp': this.invoiceAmount
     };
     let copiedinvoiceAmount = Object.assign({}, obj);
     this.invoiceAmountTemp = copiedinvoiceAmount.invoiceAmountTemp;
     this.cdr.markForCheck();
   }
```

```
  if (changes['userBillingDetail']) {
    this.formatAddress();
  }

  if (changes['isMonthly']) {
    this.isMonthly = changes.isMonthly.currentValue;
  }
 }
```

ngOnInit

Invoked when given component has been initialized. This hook is only called once after the first **ngOnChanges**

```
ngOnInit() {
  this.promoDiscount = new Discount();
  this.paymentRequest = new PaymentRequest();
  this.userInviteAndCredit = new UserInviteAndCredit();
  this.getUserCreditDetail();
  this.getUserDownPaymentDetail();
 }
```

ngDoCheck

Invoked when the change detector of the given component is invoked. It allows us to implement our own change detection algorithm for the given component.

```
@Component({
 selector: 'app-order-form-pay',
 templateUrl: './order-form-pay.component.html',
 styleUrls: ['./order-form-pay.component.css'],
 changeDetection: ChangeDetectionStrategy.OnPush
})
```

```
export class AComponent {
 @Input() o;

 // store previous value of `id`
 id;
```

```
 constructor(private cd: ChangeDetectorRef) {}

 ngOnChanges() {
  // every time the object changes
  // store the new `id`
  this.id = this.o.id;
 }

 ngDoCheck() {
  // check for object mutation
  if (this.id !== this.o.id) {
   this.cd.markForCheck();
  }
 }
}
```

ngAfterContentInit

Invoked after Angular performs any content projection into the components. Called once after the first ngDoCheck().

ngAfterContentChecked

Invoked each time the content of the given component has been checked by the change detection mechanism of Angular. Called after the ngAfterContentInit() and every subsequent ngDoCheck()

ngAfterViewInit

Invoked when the component's view has been fully initialized. Called once after the first ngAfterContentChecked().

ngAfterViewChecked

Invoked each time the view of the given component has been checked by the change detection mechanism of Angular. Called after the ngAfterViewInit and every subsequent ngAfterContentChecked().

ngOnDestroy

This method will be invoked just before Angular destroys the component.
Use this hook to unsubscribe observables and detach event handlers to avoid memory leaks.

```
subs = new Subscription();
```

```
subscribeDropZone() {
   this.subs.add(this.dndService.dropModel(this.model).subscribe(
    (target) => {
      if (target.classList.contains('contactListDrop')) {
        this.onDrop(item);
      }
      else if (target.classList.contains('contactDeleteDrop')) {
        this.onDeleteContact(item);
      }
      else if (target.classList.contains('contactEmailDrop')) {
        this.onEmailModalClick(item);
      }
      else if (target.classList.contains('contactSmsDrop')) {
        this.onSmsModalClick(item);
      }
      else if (target.classList.contains('sharingContact')) {
        this.onShareContact(item);
      }
    })
  );
 }
```

```
ngOnDestroy() {
   // destroy all the subscriptions at once
   this.subs.unsubscribe();
}
```