Routing

Routing

- Serve webpages based upon the requested url
- In Angular applications URLs are not serve from the server and never reload the page, every time user requests for a new page
- The URLs are strictly local in the browser and serve the page form local
- Every time when you request a Url, the Angular router navigates to the new component and renders its template and updates the history and URL

Routing Configuration

- **Routes** describe the routes

```
const routes: Routes = [
    { path: '', component: DatabindingComponent },
    { path: 'about/:id', component: AboutComponent },
    { path: 'notfound', component: NotfoundComponent },
    { path: '**', redirectTo: 'notfound' }
];
```

- **RouterOutlet** is where the router render the component

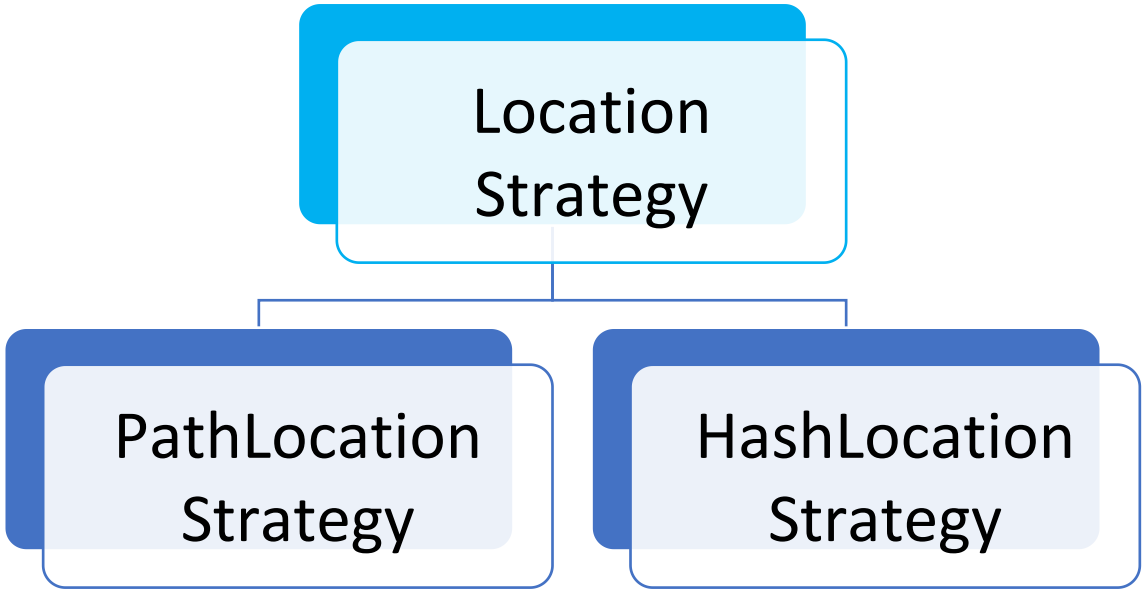
```
<div class="container">
  <router-outlet></router-outlet>
</div>
```

- **RouterLink** a link to a route name

```
<a [routerLink]="['/about','4']">About</a>
```

Location Strategies

- Defines how the URL/Request is being resolved
- Also determines how your URL will look like
- Angular supports two Location Strategies



PathLocation Strategy

- Here, URL looks like http://localhost:4200/product
- Supports Server-Side Rendering
- SEO Friendly
- Older Browsers don’t support
- For Configuration use
  - <base href="/"> at main page
  - RouterModule.forRoot(appRoutes) in routing module

HashLocation Strategy

- Here, URL looks like *http://localhost:4200/#/product*
- Supported by all browsers
- For Configuration use
  - <base href="/"> at main page
  - RouterModule.forRoot(appRoutes, {useHash: true}) in routing module

Route Parameters

- Used to pass data through route paths
- Mainly there are following ways :
  - Required Parameters
  - Optional Parameters
  - Query Parameters

Required Parameters

- Setup

```
{ path: 'product/:id', component: ProductComponent }
```

- Activate

```
<a [routerLink]="['/product',4]">Product</a>
```

```
this.router.navigate(['product', 4]);
```

- Retrieve

```
this.route.params.subscribe((p) => {
  this.id = p.id;
});
```

Optional Parameters

- Setup

```
{ path: 'product', component: ProductComponent }
```

- Activate

```
<a [routerLink]="['/product',{id:1,name:'books'}]">Product</a>
```

```
this.router.navigate(['product', { id: 1, name: 'books' }]);
```

- Retrieve

```
this.route.params.subscribe((p) => {
  this.id = p.id;
  this.name = p.name;
});
```

Query Parameters

- Setup

```
{ path: 'product', component: ProductComponent }
```

- Activate

```
<a [routerLink]="['/product']" [queryParams]="{id:1,name:'books'}">Product</a>
```

```
this.router.navigate(['product', { queryParams: { id: 1, name: 'books' } }]);
```

- Retrieve

```
this.route.queryParams.subscribe((p) => {
  this.id = p.id;
  this.name = p.name;
});
```

Child/Nested Routes

- Child/Nested routes is a powerful feature in Angular router to render child/nested components using route
- Children option is used to define child/nested routes

```
const routes: Routes = [
    { path: '', component: DatabindingComponent },
    {
        path: 'about', component: AboutComponent, children: [
            { path: 'profile', component: ProfileComponent } ]
    }
];
```

Setup Child Routes

```
<a [routerLink]="['profile']">Show Profile</a>
<router-outlet></router-outlet>
```

Activate Child Routes

DotNetTricks

Lazy Loading

- Lazy loading is a technique that allows you to load Angular components asynchronously when a specific route is activated
- Improves initial performance during the initial load, especially if you have many components with complex routing
- Lazy module loading is useful for developing more than one area(modules) in an application

Configuring Lazy Loading

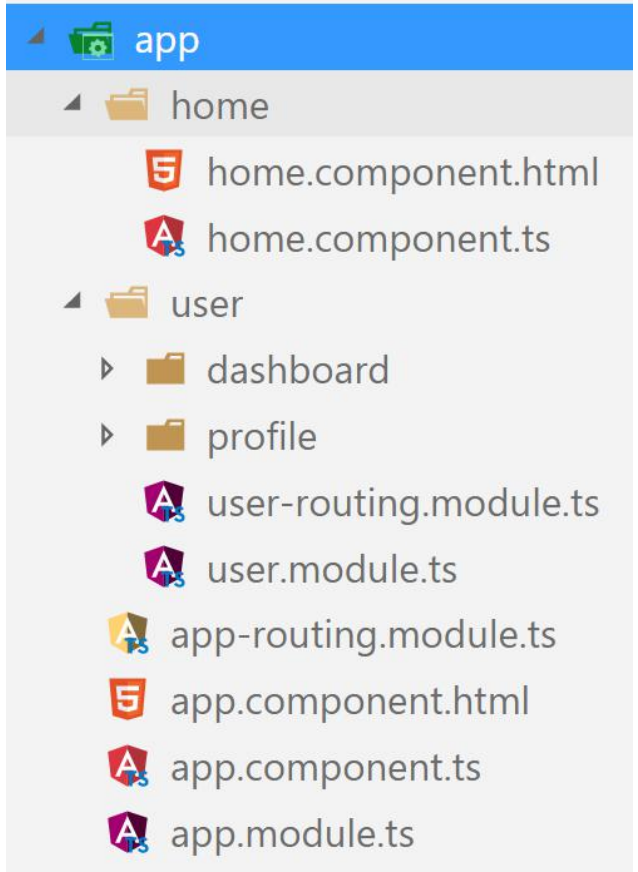
Step 1 : Use loadChildren

```
{ path: 'admin', loadChildren: () => import('./admin/admin.module').then(mod => mod.AdminModule) }
```

```
const routes: Routes = [
  { path: 'eager', component: HomeComponent },
  { path: 'lazy', loadChildren: './user/user.module#UserModule' }
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Step 2 : Define routes for lazy load module

```
const routes: Routes = [
  { path: '', component: DashboardComponent }
];
@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class UserRoutingModule { }
```



Angular Route Guards

- Route guards are functions which are called when router tries to activate or deactivate certain routes
- Guards control how the user navigates in Angular App
- Using route guards you can stop a user from visiting certain routes
- Used to protect routes from unauthorize access
- A route can have multiple guards and the guards are checked in the order they were added to the route

Route Guard Types

- Angular supports following route guards:
  - CanActivate - Checks the route access for a user
  - CanActivateChild - Checks the child routes access for a user
  - CanDeactivate - Checks to see if a user can leave a route
  - CanLoad - Checks to see if a user can route to a module that lazy loaded
  - Resolve - Allows to get route data before a route activation