**🔍 What Is Angular Routing?**

Angular routing is the system that lets you **navigate between different views or components** in a **single-page application (SPA)**, using URLs that **do not reload the page**.

Instead, Angular dynamically updates the **<router-outlet>** based on the route.

**📘 Core Concepts**

**1. Route Definition**

Defined in app-routing.module.ts:

const routes: Routes = [

{ path: '', component: HomeComponent }, // root path

{ path: 'about', component: AboutComponent }, // static path

{ path: 'user/:id', component: UserComponent } // dynamic route

];

You must import RouterModule.forRoot(routes) in AppModule.

**2. Route Parameters**

Dynamic segments using : — e.g., path: 'user/:id'

<a [routerLink]="['/user', 42]">User 42</a>

Access in component:

this.route.snapshot.paramMap.get('id')

**3. Nested (Child) Routes**

Define routes **inside another route**:

const routes: Routes = [

{

path: 'admin',

component: AdminComponent,

children: [

{ path: 'users', component: UsersComponent },

{ path: 'settings', component: SettingsComponent }

]

}

];

You must use a <router-outlet> in AdminComponent to render children.

**4. Default and Wildcard Routes**

{ path: '', redirectTo: 'home', pathMatch: 'full' } // default

{ path: '\*\*', component: NotFoundComponent } // wildcard 404

**🧩 Advanced Features**

**5. Route Guards**

* CanActivate: Prevent access
* CanDeactivate: Prevent leaving
* CanLoad: Prevent loading lazy modules

Apply in route config:

{ path: 'dashboard', canActivate: [AuthGuard], component: DashboardComponent }

**6. Lazy Loading**

Split your app into **feature modules** loaded only when needed.

const routes: Routes = [

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

];

In admin-routing.module.ts:

const routes: Routes = [

{ path: '', component: AdminHomeComponent }

];

Use RouterModule.forChild(routes) in the lazy-loaded module.

**7. Named Outlets**

Enable multiple routes to render in different areas simultaneously.

<router-outlet></router-outlet> <!-- primary -->

<router-outlet name="sidebar"></router-outlet> <!-- named -->

Route config:

{ path: 'chat', component: ChatComponent, outlet: 'sidebar' }

URL:

arduino

CopyEdit

/(primary:home//sidebar:chat)

**8. Route Resolvers**

Pre-fetch data before loading the route.

{

path: 'profile',

component: ProfileComponent,

resolve: { user: UserResolver }

}

In UserResolver:

resolve(): Observable<User> {

return this.userService.getUser();

}

In component:

this.route.snapshot.data['user']

**9. Query Parameters & Fragments**

<a [routerLink]="['/search']" [queryParams]="{ q: 'angular' }" fragment="section1">

Search

</a>

Access in component:

this.route.snapshot.queryParamMap.get('q');

this.route.snapshot.fragment;

**🛠️ Example: Full Routing Module**

const routes: Routes = [

{ path: '', redirectTo: 'home', pathMatch: 'full' },

{ path: 'home', component: HomeComponent },

{

path: 'admin',

loadChildren: () => import('./admin/admin.module').then(m => m.AdminModule),

canLoad: [AdminGuard]

},

{ path: 'login', component: LoginComponent },

{ path: '\*\*', component: NotFoundComponent }

];

In app.module.ts:

imports: [RouterModule.forRoot(routes)]

**🧪 Programmatic Navigation**

constructor(private router: Router) {}

goToDashboard() {

this.router.navigate(['/dashboard']);

}

**🧠 Best Practices**

* Use lazy loading for large apps.
* Protect routes with guards.
* Always define a wildcard route for 404 handling.
* Use resolvers for essential data.
* Prefer [routerLink] over href to avoid full page reload.