# Chapter 09: Lazy Loading, Route Guards & Resolvers

## ****What are Child Routes?**** Child routes (or nested routes) allow you to define routes inside a parent route. They are useful for building layouts where certain components (like headers or sidebars) remain constant, while only a part of the view updates.

### What is Lazy Loading?

**Lazy loading** means loading components or routes only when they are needed — not at the start. This reduces the initial bundle size and improves performance.

### ✅ When to Use It

* Large applications with multiple features
* Feature areas like dashboard, admin panel, reports, etc.
* Anything that’s not immediately required on startup

### Setup Lazy Loading in Angular 17+

Angular 17+ uses **loadComponent** or **loadChildren** with **standalone components** — no NgModules required!

Great! Let see how to define and configure **child routes** with **lazy loaded** component:

* Lazy loading studedit component
* Components like studlist, studadd, and studedit
* Dedicated routing file: stud.route.ts
* Integration with main app.routes.ts

**Folder Structure**

src/

app/

stud/

studlist/

studlist.component.ts

studadd/

studadd.component.ts

studedit/

studedit.component.ts

stud.route.ts

app.routes.ts

app.component.ts

**Step 1: Create Standalone Components**

Each component should be created as **standalone**:

ng g component stud/studlist --standalone

ng g component stud/studadd --standalone

ng g component stud/studedit --standalone

**Step 2: Define Routes in stud.route.ts**

**We have to create this file manually in stud folder**

// src/app/stud/stud.route.ts

import { Routes } from '@angular/router';

import { StudlistComponent } from './studlist/studlist.component';

import { StudaddComponent } from './studadd/studadd.component';

export const studRoutes: Routes = [

{ path: '', component: StudlistComponent },

{ path: 'add', component: StudaddComponent },

{ path: 'edit/:id', loadComponent: () =>

import('./studedit/studedit.component').then(m => .StudeditComponent)}

];

* '' is the default child path (/stud)
* edit/:id uses lazy loading with loadComponent, **it will not be imported**

**Step 3: Add Child Routes to app.routes.ts**

// src/app/app.routes.ts

import { Routes } from '@angular/router';

import { studRoutes } from './stud/stud.route';

export const appRoutes: Routes = [

{ path: 'stud', children: studRoutes }

];

**Step 4: Add RouterLinks in StudlistComponent**

<!-- src/app/stud/studlist/studlist.component.html -->

<h2>Student List</h2>

<ul>

<li><a [routerLink]="['/stud/add']">Add Student</a></li>

<li><a [routerLink]="['/stud/edit', 1]">Edit Student #1</a></li>

</ul>

## What is loadChildren?

loadChildren is used in Angular routing to **lazy-load an entire set of routes** (usually for a feature or section like staff, admin, products, etc.).

### Key Benefits:

* Improves **initial load performance**
* Downloads only the routes and components **when needed**

### How it works with Standalone Components

In Angular 14+, loadChildren supports **lazy-loading a route definition (array of Routes)** from a .ts file.

#### ✅ Example

{

path: 'staff',

loadChildren: () =>

import('./staff/staff.routes').then(m => m.staffRoutes)

}

* This tells Angular:
  + When the user navigates to /staff…
  + Load the staff.routes.ts file dynamically.
  + Use the exported staffRoutes array for routing.

## What is a Shared Layout Component?

### Definition:

A **shared layout component** acts as a **visual wrapper** or **parent layout** for a group of child routes.  
It typically contains:

* A **common layout** (e.g., nav, sidebar, header)
* A <router-outlet> to render child components

### Example

// staff.routes.ts

{

path: '',

loadComponent: () =>

import('./layout/layout.component').then(m => m.LayoutComponent),

children: [

{

path: '',

loadComponent: () =>

import('./dashboard/dashboard.component').then(m => m.DashboardComponent)

},

{

path: 'members',

loadComponent: () =>

import('./members/members.component').then(m => m.MembersComponent)

}

]

}

This means:

1. When someone visits /staff or /staff/members, Angular:
   * Loads layout.component.ts first.
   * Inserts the matched child component (dashboard, members, etc.) into **<router-outlet> inside the layout**.

### Layout Component Template Example:

<!-- layout.component.html -->

<h2>Staff Portal</h2>

<nav>

<a routerLink="/staff">Dashboard</a> |

<a routerLink="/staff/members">Members</a>

</nav>

<hr />

<router-outlet></router-outlet>

* The <router-outlet> is **where Angular renders matched child routes**
* The nav, header, etc., are always shown as **shared UI**

Excellent! Let's now **include CanDeactivate** and **replace AdminComponent with ProfileComponent** for a more personal experience. Here's the **full updated Angular 19 standalone project**.

## ✅ Updated Goal

An app with:

* **LoginComponent** – login page.
* **ProfileComponent** – parent layout for authenticated area.
* **AddPaymentComponent** – has a form, protected with CanDeactivate.
* **ListPaymentComponent** – simple list.
* Guards:
  + CanActivate → for /profile
  + CanActivateChild → for /profile/payment/\*
  + CanDeactivate → for /profile/payment/add if form is dirty

## 📦 Updated File Structure

src/app/

│

├── guards/

│ ├── auth.guard.ts (CanActivate)

│ ├── child.guard.ts (CanActivateChild)

│ └── deactivate.guard.ts (CanDeactivate)

│

├── login.component.ts

├── profile.component.ts

├── payment/

│ ├── add-payment.component.ts

│ └── list-payment.component.ts

│

├── auth.service.ts

├── form-state.service.ts (for CanDeactivate)

├── app.routes.ts

├── app.component.ts

└── main.ts

## 🧱 Code Files

### ✅ AuthService

// src/app/auth.service.ts

import { Injectable } from '@angular/core';

@Injectable({ providedIn: 'root' })

export class AuthService {

private isLoggedInValue = false;

login() { this.isLoggedInValue = true; }

logout() { this.isLoggedInValue = false; }

isLoggedIn() { return this.isLoggedInValue; }

}

### 📝 FormStateService (for unsaved form tracking)

// src/app/form-state.service.ts

import { Injectable } from '@angular/core';

@Injectable({ providedIn: 'root' })

export class FormStateService {

hasUnsavedChanges = false;

}

### 🔐 Guards

#### CanActivate

// src/app/guards/auth.guard.ts

import { CanActivateFn, Router } from '@angular/router';

import { inject } from '@angular/core';

import { AuthService } from '../auth.service';

export const canActivateGuard: CanActivateFn = () => {

const auth = inject(AuthService);

const router = inject(Router);

if (auth.isLoggedIn()) return true;

alert('Login required');

router.navigate(['/login']);

return false;

};

#### CanActivateChild

// src/app/guards/child.guard.ts

import { CanActivateChildFn, Router } from '@angular/router';

import { inject } from '@angular/core';

import { AuthService } from '../auth.service';

export const canActivateChildGuard: CanActivateChildFn = () => {

const auth = inject(AuthService);

const router = inject(Router);

if (auth.isLoggedIn()) return true;

alert('Access denied');

router.navigate(['/login']);

return false;

};

#### CanDeactivate

// src/app/guards/deactivate.guard.ts

import { CanDeactivateFn } from '@angular/router';

import { inject } from '@angular/core';

import { FormStateService } from '../form-state.service';

export const canDeactivateGuard: CanDeactivateFn<any> = () => {

const formState = inject(FormStateService);

return !formState.hasUnsavedChanges || confirm('You have unsaved changes. Leave anyway?');

};

### 🔑 LoginComponent

// src/app/login.component.ts

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

import { Router } from '@angular/router';

import { AuthService } from './auth.service';

@Component({

selector: 'app-login',

standalone: true,

template: `

<h2>Login</h2>

<button (click)="login()">Login</button>

`,

})

export class LoginComponent {

constructor(private auth: AuthService, private router: Router) {}

login() {

this.auth.login();

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

}

}

### 👤 ProfileComponent (replaces AdminComponent)

// src/app/profile.component.ts

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

import { RouterModule } from '@angular/router';

@Component({

selector: 'app-profile',

standalone: true,

imports: [RouterModule],

template: `

<h2>My Profile</h2>

<nav>

<a routerLink="payment/add">Add Payment</a> |

<a routerLink="payment/list">List Payments</a>

</nav>

<router-outlet></router-outlet>

`,

})

export class ProfileComponent {}

### 💳 AddPaymentComponent (uses CanDeactivate)

// src/app/payment/add-payment.component.ts

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

import { FormStateService } from '../form-state.service';

@Component({

selector: 'app-add-payment',

standalone: true,

template: `

<h3>Add Payment</h3>

<input placeholder="Amount" (input)="onInput()" />

<button (click)="submit()">Submit</button>

`,

})

export class AddPaymentComponent {

constructor(public formState: FormStateService) {}

onInput() {

this.formState.hasUnsavedChanges = true;

}

submit() {

alert('Payment submitted!');

this.formState.hasUnsavedChanges = false;

}

}

### 📋 ListPaymentComponent

// src/app/payment/list-payment.component.ts

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

@Component({

selector: 'app-list-payment',

standalone: true,

template: `

<h3>Payment List</h3>

<ul>

<li>Payment #1 - ₹500</li>

<li>Payment #2 - ₹1200</li>

</ul>

`,

})

export class ListPaymentComponent {}

### 🚦 Routing Setup

// src/app/app.routes.ts

import { Routes } from '@angular/router';

import { LoginComponent } from './login.component';

import { ProfileComponent } from './profile.component';

import { AddPaymentComponent } from './payment/add-payment.component';

import { ListPaymentComponent } from './payment/list-payment.component';

import { canActivateGuard } from './guards/auth.guard';

import { canActivateChildGuard } from './guards/child.guard';

import { canDeactivateGuard } from './guards/deactivate.guard';

export const routes: Routes = [

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

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

{

path: 'profile',

component: ProfileComponent,

canActivate: [canActivateGuard],

canActivateChild: [canActivateChildGuard],

children: [

{ path: 'payment/add', component: AddPaymentComponent, canDeactivate: [canDeactivateGuard] },

{ path: 'payment/list', component: ListPaymentComponent },

]

},

{ path: '\*\*', redirectTo: 'login' },

];

### ⚙️ main.ts

import { bootstrapApplication } from '@angular/platform-browser';

import { AppComponent } from './app.component';

import { provideRouter } from '@angular/router';

import { routes } from './app.routes';

bootstrapApplication(AppComponent, {

providers: [provideRouter(routes)],

});

### 📌 AppComponent

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

import { RouterModule } from '@angular/router';

@Component({

selector: 'app-root',

standalone: true,

imports: [RouterModule],

template: `

<h1>Payment Portal</h1>

<router-outlet></router-outlet>

`,

})

export class AppComponent {}

## ✅ Test the Flow

| **Route** | **Guard Type** | **Behavior** |
| --- | --- | --- |
| /profile | CanActivate | ❌ Redirects to /login if not logged in |
| /profile/payment/add | CanActivateChild | ❌ Redirects to /login if not logged in |
| /profile/payment/add | CanDeactivate | 🔄 Prompt before leaving if unsaved changes exist |
| /profile/payment/list | Open after login | ✅ Works fine after logging in |
| /login → login() |  | ✅ Logs in and redirects to /profile |

## ✅ Summary

You now have a fully working standalone Angular 19 app with:

* **Login system**
* **Protected parent + child routes**
* **Unsaved form detection**
* **Standalone components (no modules!)**
* **Three types of guards**: CanActivate, CanActivateChild, CanDeactivate

Would you like me to generate a GitHub repo, a downloadable zip, or help you add Resolve or CanLoad too?

## 🛡️ Part 2: Route Guards — Protecting Routes

### 🧠 What are Route Guards?

Guards are functions that **control access to routes**.

They answer questions like:

* “Is the user logged in?”
* “Does the user have permission to view this page?”
* “Are there unsaved changes we should warn the user about?”

### ✅ Types of Route Guards

| **Guard** | **Purpose** |
| --- | --- |
| canActivate | Allow/deny access to route |
| canMatch | Decide if a route matches before activation |
| canDeactivate | Warn or confirm before leaving the page |
| resolve | Preload data before route loads |

### ✅ Writing a canActivate Guard (Functional, Angular 15+)

import { inject } from '@angular/core';

import { CanActivateFn } from '@angular/router';

import { AuthService } from './auth.service';

export const authGuard: CanActivateFn = () => {

const auth = inject(AuthService);

return auth.isLoggedIn();

};

#### Using the Guard in a Route

{

path: 'dashboard',

loadComponent: () =>

import('./dashboard.component').then(m => m.DashboardComponent),

canActivate: [authGuard]

}

### ✅ canDeactivate Guard (Prevent Leaving Page)

import { CanDeactivateFn } from '@angular/router';

export const confirmLeave: CanDeactivateFn<any> = (component) => {

return confirm('You have unsaved changes. Leave page?');

};

{

path: 'edit',

component: EditFormComponent,

canDeactivate: [confirmLeave]

}

### ✅ canMatch — Block route matching early

import { CanMatchFn } from '@angular/router';

export const roleGuard: CanMatchFn = () => {

const userRole = 'admin'; // Assume fetched

return userRole === 'admin';

};

## 🧾 Part 3: Route Resolvers — Preloading Data

### 🎯 What is a Resolver?

Resolvers fetch data **before** a route is activated. Perfect for:

* Product details
* Profile data
* Dashboard metrics

### ✅ Creating a Resolver (Standalone)

import { inject } from '@angular/core';

import { ResolveFn } from '@angular/router';

import { ProductService } from './product.service';

export const productResolver: ResolveFn<any> = (route) => {

const productService = inject(ProductService);

const id = route.paramMap.get('id')!;

return productService.getProduct(id);

};

### ✅ Using it in a Route

{

path: 'product/:id',

component: ProductDetailComponent,

resolve: {

product: productResolver

}

}

### Inside ProductDetailComponent:

constructor(private route: ActivatedRoute) {

this.route.data.subscribe(data => {

this.product = data['product'];

});

}

## 🧪 Real-World Scenario: Admin Section

### Goal:

* /admin should be **lazy-loaded**
* Protected with a **login guard**
* Use a **resolver** to fetch stats before showing the dashboard

### ✅ Route Setup:

{

path: 'admin',

loadComponent: () =>

import('./admin/admin-dashboard.component').then(m => m.AdminDashboardComponent),

canActivate: [authGuard],

resolve: { stats: adminStatsResolver }

}

## 🧠 Summary

| **Feature** | **Description** |
| --- | --- |
| Lazy Loading | Load only when needed (routes/components) |
| Guards | Control access and flow (canActivate, etc.) |
| Resolver | Preload data before showing a route |
| Functional APIs | All guards and resolvers use inject() |

## 📘 Practice Tasks

1. Lazy load a new route /analytics.
2. Add a canActivate guard to it that blocks non-logged-in users.
3. Use a resolver to load analytics data before route activation.
4. Test the fallback route using \*\*.