# Chapter 09: Lazy Loading, Route Guards & Resolvers

## 🧩 Part 1: Lazy Loading (Modern Angular with Standalone APIs)

### 🚀 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!

### ✅ Lazy Load a Standalone Component

#### CLI Command to Generate Standalone Component:

ng generate component admin-panel --standalone

#### Add Lazy Route in app.routes.ts:

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

export const routes: Routes = [

{

path: 'admin',

loadComponent: () =>

import('./admin-panel/admin-panel.component').then(m => m.AdminPanelComponent)

}

];

### ✅ Lazy Load a Route Group (with children)

{

path: 'reports',

loadChildren: () => import('./reports/reports.routes').then(m => m.REPORT\_ROUTES)

}

#### In reports.routes.ts

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

import { ReportListComponent } from './report-list.component';

import { ReportDetailComponent } from './report-detail.component';

export const REPORT\_ROUTES: Routes = [

{ path: '', component: ReportListComponent },

{ path: ':id', component: ReportDetailComponent }

];

## 🛡️ 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 \*\*.