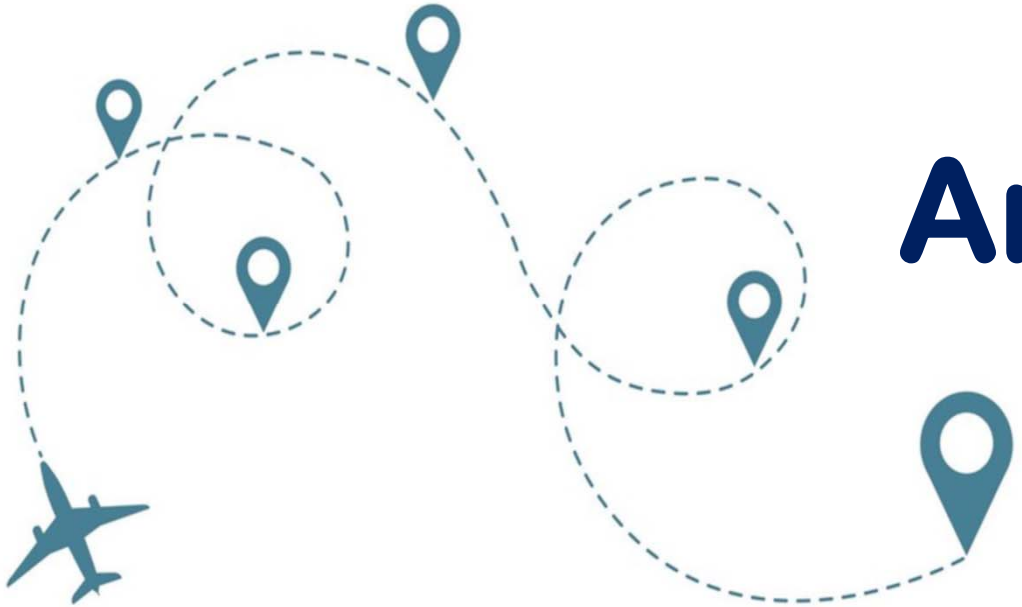
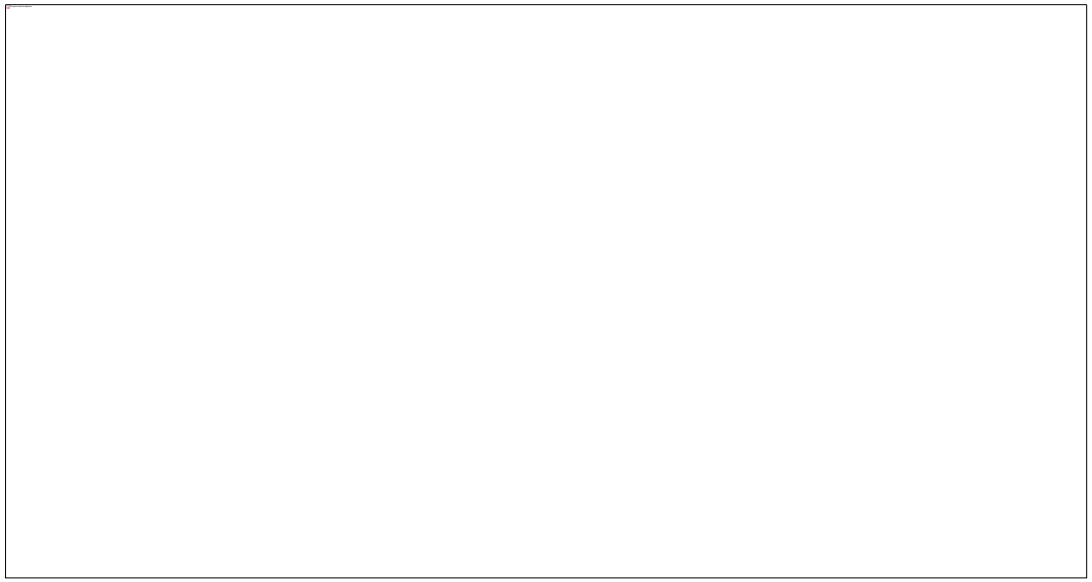


Angular Routing

Aitrich

Contents



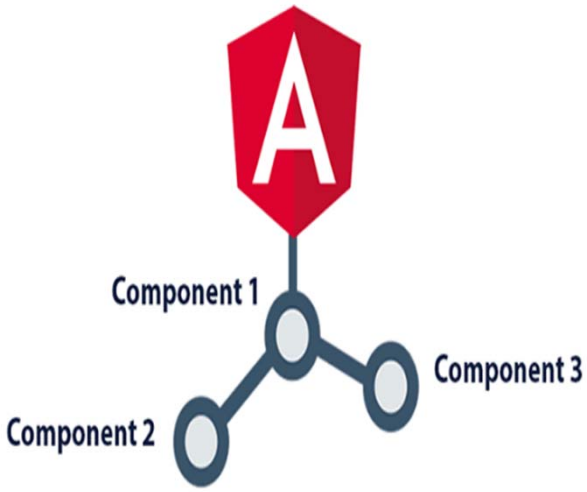
- ROUTER
- HOW DOES ROUTING WORKS
- CONFIGURING ROUTES
- SETTING UP ROUTEROUTLET
- NAVIGATION WITH ROUTERLINK
- ROUTE GUARDS

Aitrich

Introduction

In web development, Angular's Router plays a crucial role in creating smooth and interactive single-page applications (SPAs). SPAs allow content to be dynamically loaded and updated without reloading the entire page.

Router

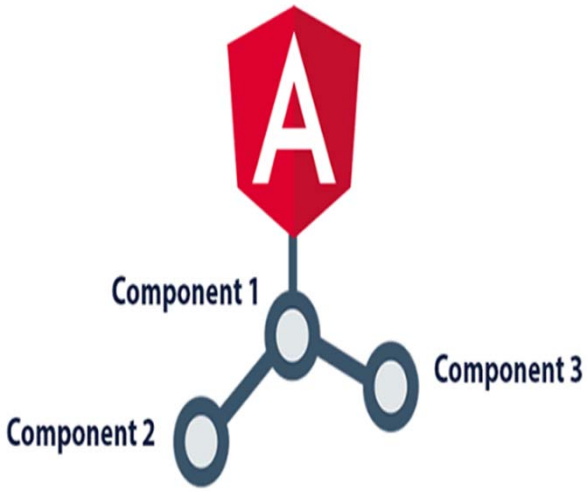


Angular Router is a powerful tool that facilitates navigation in your web application by interpreting changes in the browser's URL as instructions to update the view. Instead of traditional full-page reloads, the Router dynamically loads and updates content, providing a seamless user experience.

Slide 4

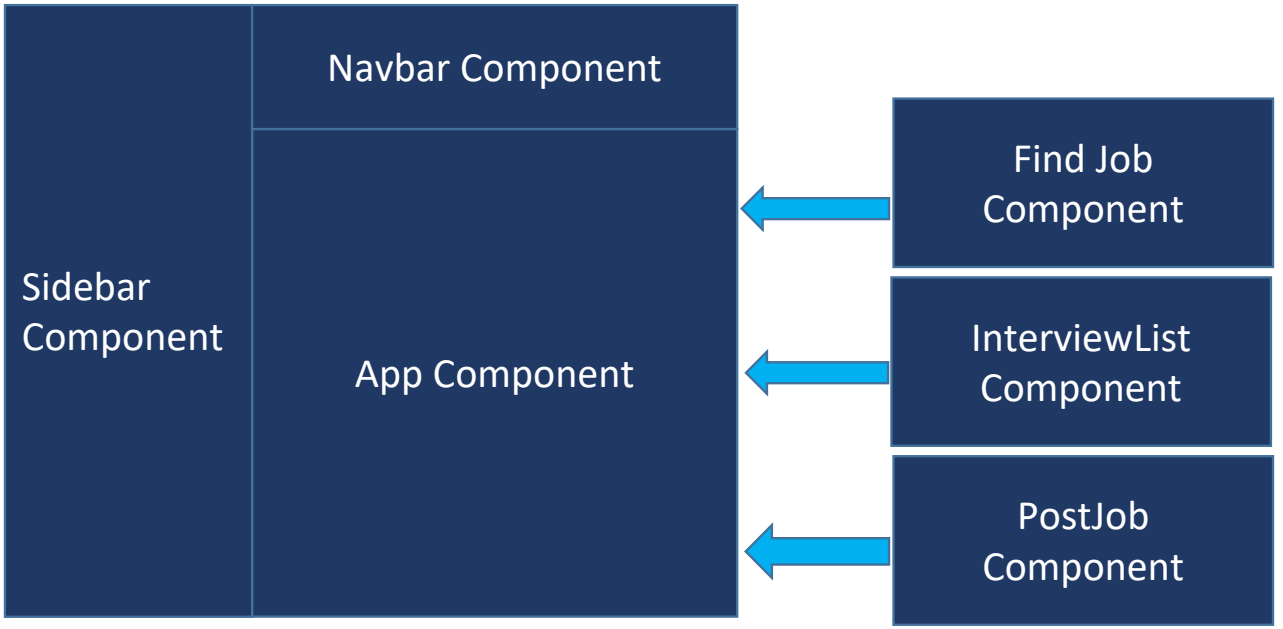
1 Aitrich 005, 8/10/2023

Routing

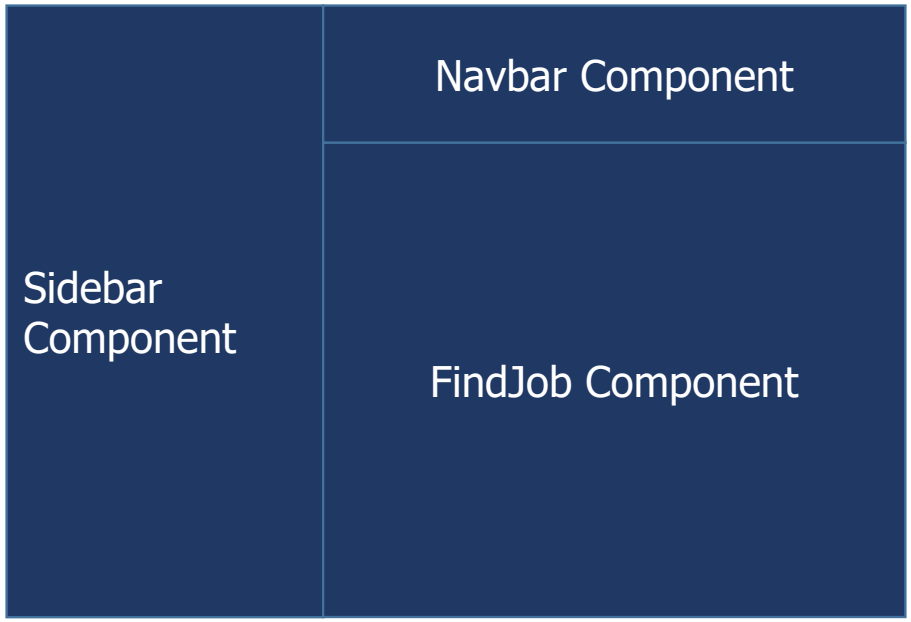


Routing is the system that directs internet traffic, similar to how road signs and maps guide drivers on the road, ensuring data packets find their way from one computer to another smoothly and efficiently.

Example


Navbar Component
Sidebar Component
App Component
Find Job Component
InterviewList Component
PostJob Component

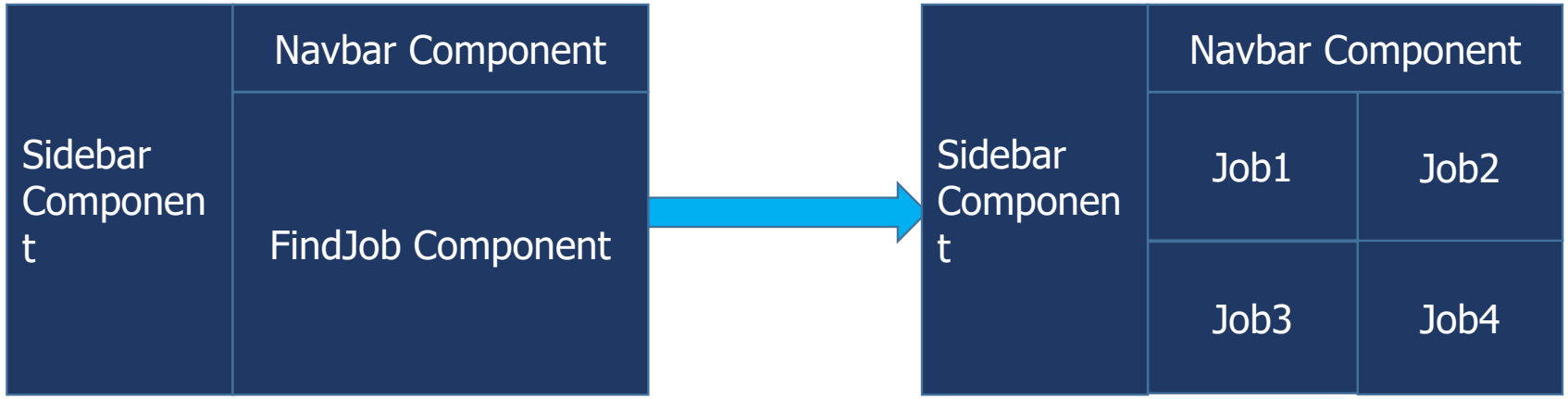
Example



While navigating through Single Page Application…

Example

If you reload a part (FindJob), it only renders that component without fetching the whole page (i.e. Navbar & SideBar) from the server.



How Does Route Work?

Sender and Receiver: Imagine you're sending a letter to a friend. You know where your friend lives (the destination), and you're at your house (the source). In the world of computers, your "house" is your device, like a computer or a smartphone.

Addressing: Just like you need your friend's address to send them a letter, your device needs the address of the website or server you want to communicate with. This address is called an IP address. It's like the street address for a house.

Router: Now, you give your letter to the postal service. In the internet world, your letter (data packet) goes to a router. A router is like a postal sorting center. It knows where to send your data packet based on the IP address it's headed to.

Routing Decision: The router checks its routing table, which is like a map of the internet. It decides the best path for your data packet to reach its destination. This path might involve multiple routers, just like your letter might pass through multiple postal sorting centers before reaching your friend.

Transmission: Once the router decides the path, it forwards your data packet to the next router along the way. This process continues until your data packet reaches its destination.

Receiver: Finally, your data packet arrives at the destination device, like your friend's house. The device there processes the packet, just like your friend reads the letter you sent.

Response: If your friend wants to reply, the process works in reverse. Their device becomes the source, and the data packet travels back to you following a similar routing process.

Configuring Routes

Configuring routes in Angular involves setting up the routes in your application's routing module and defining the corresponding components to be displayed when navigating to those routes. Here's a step-by-step guide:

Create Routing Module:

If you haven't already, create a routing module for your Angular application. You can generate it using Angular CLI:

```
ng generate module app-routing --flat --module=app
```

This command creates an **app-routing.module.ts** file in your project's root directory.

Configuring Routes

Import Required Modules:

- Inside your **app-routing.module.ts** file, import **RouterModule** and **Routes** from **@angular/router**:

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

Define Routes:

• Define an array of route objects in the **Routes** constant. Each route object should have a **path** and a **component**:

```
const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: 'contact', component: ContactComponent },
  // Add more routes as needed
];
```

Add RouterModule to NgModule:

• In the **@NgModule** decorator's **imports** array, import **RouterModule** and pass the routes array to the **forRoot()** method:

```
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
```

Add Router Outlet:

- In your main application component's template (usually **app.component.html**), add a **<router-outlet></router-outlet>** tag.
- This is where Angular will render the matched component for each route:

```
<router-outlet></router-outlet>
```

Navigate Between Routes:

- Use the **routerLink** directive in your templates to navigate between routes. For example:

```
<a routerLink="/home">Home</a>
<a routerLink="/about">About</a>
<a routerLink="/contact">Contact</a>
```

Create a Routing Module with Components

Run the Angular CLI command to generate the component with routing:

```
ng generate component job --routing
```

Configuring Routes

```
const routes: Routes = [
  { path: '', redirectTo: '/FindJob', pathMatch: 'full' },
  { path: 'Jobs', component: FindJobComponent },
  { path: 'PostJob', component: PostJobComponent },
  { path: 'Interview', component: InterviewListComponent }
];
```

- **const routes: Routes = [...]**: This declares an array named **routes** of type **Routes**. In Angular, **Routes** is an array type provided by **@angular/router** that holds route definitions.
- **{ path: '', redirectTo: '/FindJob', pathMatch: 'full' }**: This defines the default route. When the application is accessed at the root URL (**/**), it redirects to the **'/FindJob'** route.

Configuring Routes

- **path**: Specifies the URL path segment for the route. In this case, it's an empty string, indicating the root URL.
- **redirectTo**: Specifies the URL to redirect to if the path matches.
- **pathMatch: 'full'**: This option ensures that the redirection occurs only when the entire URL matches the specified path.
- **{ path: 'Jobs', component: FindJobComponent }**: This defines a route for the **'/Jobs'** path. When the URL matches **'/Jobs'**, Angular will display the **FindJobComponent**.
- **component**: Specifies the Angular component to be rendered when this route is activated. In this case, it's the **FindJobComponent**.

Child Routes

- Child routes allow you to create nested views within a parent component.
- Instead of putting all the functionality in a single component, you can use child routes to organize and modularize the user interface.

```
const routes: Routes = [
  {
    path: 'dashboard',
    children: [
      { path: 'jobs', component: JobListComponent },
      { path: 'postJob', component: PostJobComponent },
      { path: '', redirectTo: 'jobs', pathMatch: 'full' },
    ],
  },
  // ... other routes
];
```

Lazy Loading Modules

Lazy loading is a technique used in Angular to improve the performance of your application by loading modules or components only when they are needed. Instead of loading the entire application upfront, Angular loads parts of the application on-demand as the user navigates through it.

```
const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', loadChildren: () => import('./home/home.module').then(m => m.HomeModule) },
  { path: 'about', loadChildren: () => import('./about/about.module').then(m => m.AboutModule) },
];
```

Lazy Loading Modules

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

path: Specifies the URL path segment for the route. Here, it's **/home**.

•**loadChildren**: Specifies a function that returns a promise. This function is called to load the module lazily when the route is activated.

•**() => import('./home/home.module').then(m => m.HomeModule)**: This function uses dynamic import syntax to asynchronously import the **HomeModule**. Once the module is loaded, it returns a promise that resolves to the **HomeModule**.

•**m => m.HomeModule**: This is a function that extracts the **HomeModule** from the loaded module and returns it. The **HomeModule** is then used by Angular to render the components associated with this route.

Lazy Loading Modules

Lazy Loaded Module (Home.module.ts):

Create a separate module for the "Job List" section. This module will contain the components, services, and other resources specific to job listings.

When the user clicks on the "Job List" link, Angular will load the JobListingsModule only when needed, reducing the initial load time of your application.

The JobListComponent will be displayed within the main content area.

Navigation with RouterLink

- RouterLink is a built-in Angular directive used for creating navigation links within your application.It allows you to navigate between different views/components defined by routes
- RouterLink simplifies navigation and ensures consistent behavior across your application.It automatically generates correct links based on the configured routes.
- RouterLink is applied to HTML anchor (<a>) tags to create clickable navigation links.It binds to the specified route path, enabling navigation when the link is clicked.

Navigation with RouterLink

```
<!-- Creating navigation links using RouterLink -->

<a routerLink="/home">Home</a>
<a routerLink="/about">About</a>
<a routerLink="/contact">Contact</a>
```

Generating Links with Parameters:

Angular replaces parameter values when generating the link.

```
<!-- Creating a link with route parameter -->

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

Route Guards

Route guards are mechanisms in Angular that allow you to control and manage access to routes based on certain conditions. They provide an extra layer of security and control over navigation within your application

Types of Route Guards:

1. **CanActivate:**

   Determines whether a user is allowed to navigate to a particular route. Useful for scenarios like authentication and authorization checks before accessing a route.

Route Guards

2. CanDeactivate

Controls whether a user is allowed to leave a route and discard changes made to a form or data.Useful for confirming navigation away from unsaved data.

3. Resolve

Fetches data required by a component before activating the route. Ensures that the component has the necessary data before rendering

4. CanLoad

Determines whether a feature module (with lazy loading) should be loaded or not. Useful for delaying the loading of certain modules until specific conditions are met.

Questions???