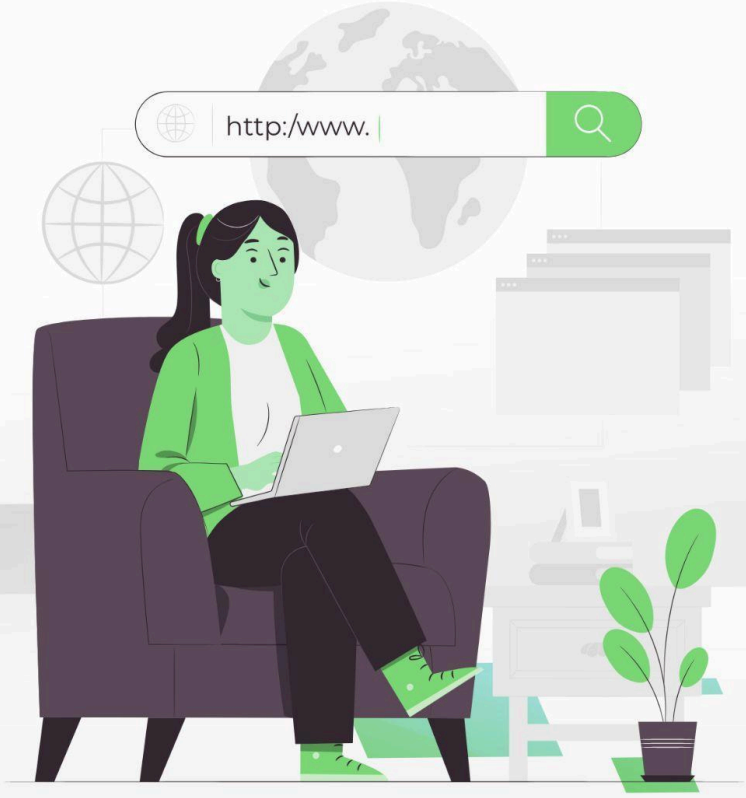
Angular Routing

Everything you need to know



Abdelfattah Ragab

Angular Routing

Everything you need to know

Abdelfattah Ragab

Introduction

Welcome to the book “Angular Routing”.
In this book, I explain everything you need to know about Angular routing.
Routing helps you to change what the user sees in a single-page app.
In this book, you will learn how to implement common routing tasks. You will learn how to set up routes, retrieve route information, display 404 pages, prevent unauthorized access, and much more.
By the end of this book, you will be confident working with routing in your Angular application and be able to handle all kinds of scenarios.
Let us get started.

Configuration

If you open the Angular application in Visual Studio Code, you can find the **app.config.ts** file in the **src/app** folder of the application.
In the `providers` array, you must specify the routes as follows: `provideRouter(routes)`. This is what it looks like:

```
import { ApplicationConfig,
provideZoneChangeDetection } from
'@angular/core';
import { provideRouter } from
'@angular/router';

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

export const appConfig:
ApplicationConfig = {
  providers:
[provideZoneChangeDetection({
eventCoalescing: true }),
provideRouter(routes)]
};
```

The `routes` array is defined in the **app.routes.ts** file. It will be empty at first:

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

export const routes: Routes = [];
```

`<router-outlet />`

The router outlet is a directive that acts as a placeholder that Angular fills dynamically based on the current router state.
If you open the **app.component.html** file, you will find it in the last line of the file. Normally we remove all the code above this line when we start a new application. We remove everything in the **app.component.html** file and leave only one line: the `<router outlet />`.
If you want to add a header to the application, you can add it above this line, and if you want to add a footer, you need to add it after this line. The router outlet line is the place where the dynamic component is placed based on the current router state.

How the router works

Angular routing is very simple, you just have to add a new entry to the routes array. That's it.
If you enter this path in the address bar, the desired component will be displayed in the `<router-outlet />` placeholder.
Let us see this in action by adding a new entry to the routes array in the **app.routes.ts** file.

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

```
import { HomeComponent } from
'./pages/home/home.component';

export const routes: Routes = [{ path:
'home', component: HomeComponent }];
```

Now, run run the application and write the following url in the address bar:
http://localhost:4200/**home**

You will find a page saying “home works!”, the content of the home page.
Don’t forget to remove everything from the app.component.html page except the `<router-outlet />`

Default route

When you run the application, a blank page is displayed.
I want it to default to the home page.
I will add a new empty path to the route array as follows:

```
import { Routes } from
'@angular/router';
import { HomeComponent } from
'./pages/home/home.component';

export const routes: Routes = [
```

```
  { path: '', redirectTo: 'home',
pathMatch: 'full' },
  { path: 'home', component:
HomeComponent },
];
```

redirectTo

We use the `redirectTo` property to redirect from one route to another.
When the user tries to access the base url, `http://localhost:4200`, it will redirect to the "home" entry, `http://localhost:4200/home`, and load the `HomeComponent`.

pathMatch

The `pathMatch` property can take two values: `full` or `prefix`.
The `pathMatch: 'full'` option is crucial here; it tells the router to redirect only if the entire URL path matches the empty string.
If you set `pathMatch: 'prefix'` for the empty path, any route starting with the empty string (which is all routes) would trigger the redirect, which is not normally the desired behavior.

Not Found

Add a new entry to the routes array as follows:

```
import { Routes } from
'@angular/router';
import { HomeComponent } from
'./pages/home/home.component';
import { NotFoundComponent } from
'./pages/not-found/not-found.component';

export const routes: Routes = [
  { path: '', redirectTo: 'home',
pathMatch: 'full' },
  { path: 'home', component:
HomeComponent },
  { path: 'not-found', component:
NotFoundComponent },
];
```

When you visit the URL `http://localhost:4200/not-found`, the `NotFoundComponent` is loaded.
You can fill the `NotFoundComponent` with any content to inform the user that the page they are looking for does not exist.

Wildcard route

A wildcard route is defined with the syntax `**` in the routing configuration. This route should usually be at the

end of your route definitions to ensure that only routes that were not found by any of the previous routes are included. The routes are matched in the order in which you define them in the array.

```
import { Routes } from
'@angular/router';
import { HomeComponent } from
'./pages/home/home.component';
import { NotFoundComponent } from
'./pages/not-found/not-found.component';

export const routes: Routes = [
  { path: '', redirectTo: 'home',
pathMatch: 'full' },
  { path: 'home', component:
HomeComponent },
  { path: 'not-found', component:
NotFoundComponent },
  {
    path: '**',
    redirectTo: 'not-found',
  },
];
```

As a final step, I redirect all routes to the `not-found` route.

title

When defining routes, you can specify page titles. It is strongly recommended that you do this as it helps to

assess accessibility. If you are using Lighthouse to test your application.

```
{ path: 'home', component:
HomeComponent, title: 'Home - Mini Shop'
},
{ path: 'about', component:
AboutComponent, title: 'About - Mini
Shop' },
{ path: 'contact', component:
ContactComponent, title: 'Contact - Mini
Shop' },
```

Guards

Guards allow developers to control access to routes in an Angular application based on certain conditions.

canActivate

canActivate allows developers to define logic that determines whether a user can navigate to a specific route.

Let us assume we have the "Add products" page, which should only be accessible to administrators.

```
import { Routes } from
'@angular/router';
import { HomeComponent } from
'./pages/home/home.component';
```