

**SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.  
(AFFILIATED TO SAURASHTRA UNIVERSITY)**



**Lt. Shree Chimanbhai Shukla**

**MSCIT SEM-3 ANGULARJS**

**Shree H.N.Shukla college2  
vaishali nagar  
Near Amrapali Under Bridge,  
Raiya road  
Rajkot  
Ph No:-0281 2440478**

**Shree H.N.Shukla college3  
vaishali nagar  
Near Amrapali Under Bridge,  
Raiya road  
Rajkot  
Ph No:-0281 2440478**

**SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.**  
**(AFFILIATED TO SAURASHTRA UNIVERSITY)**

## Unit : 4

### Routing & Wrap Up

- Understanding the need of a Router
- Setting Up and implementing Routes
- Navigating to Router Links
- Understanding Router Paths
- Styling Active Router Links
- Understanding Navigation Paths
- Styling Router Links
- Navigating Dynamically
- Using Relative Paths
- Passing Parameters to Routes and fetching route parameters
- Fetching route parameters in a Reactive Way
- Passing query parameters and fragments
- Understanding Nested Routes
- Redirecting and Wildcard routes
- Wrap Up

#### What is a router?

A **router** is a device that shares a single internet connection with multiple wired devices. It serves as a hub and firewall, providing multiple ports for computers, game consoles, media streamers, and more. It also has a port dedicated to communicating with a modem or ONT.

A **Wi-Fi router** (or wireless router) is a device that shares a single internet connection with multiple wired and wireless devices. A wireless router may have up to eight external antennas, while other models pack the antennas inside the chassis.

A **wireless gateway** is a device that functions as a cable or DSL modem **and** a router. It typically includes several ports on the back for wired connections. The Wi-Fi antennas are internal or external, depending on the model.

Think of a router as a small computer dedicated to relaying only network traffic. It has a processor, system memory, two storage devices with the startup configuration and the diagnostic software, and flash-based storage for the operating system

# **SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.**

## **(AFFILIATED TO SAURASHTRA UNIVERSITY)**

(firmware). There are several ports on the back for wired Ethernet connections and possibly one or two USB ports for network-shared storage.

### **Why do we need a router?**

You need a router to share a single internet connection to multiple devices. Internet providers generally issue only one IP address to the first device connected to its modem or ONT. Think of that address as your internet mailing address—without it, you can't receive or send data across the internet.

Because your provider issues only one public address, the router is technically the only device that can send and receive data. To share the connection, it assigns a private address to each of your devices, and sends and receives internet data on their behalf.

A router is also required if you want to manage all your networked devices from one central point.

### **❖ 7 Steps To Get Started With React Routing**

React is one of the most used JavaScript-based front-end framework right now. As routing is a common task when implementing React web application this article runs you through the process of setting up routing with React router in your React project quickly.

React Router is a fully-featured client and server-side routing library for React. The project's website is available at <https://reactrouter.com/>:

Activating React Router for your React web application is easy and just comprises few steps. Let's start the process by setting up a new fresh React project first.

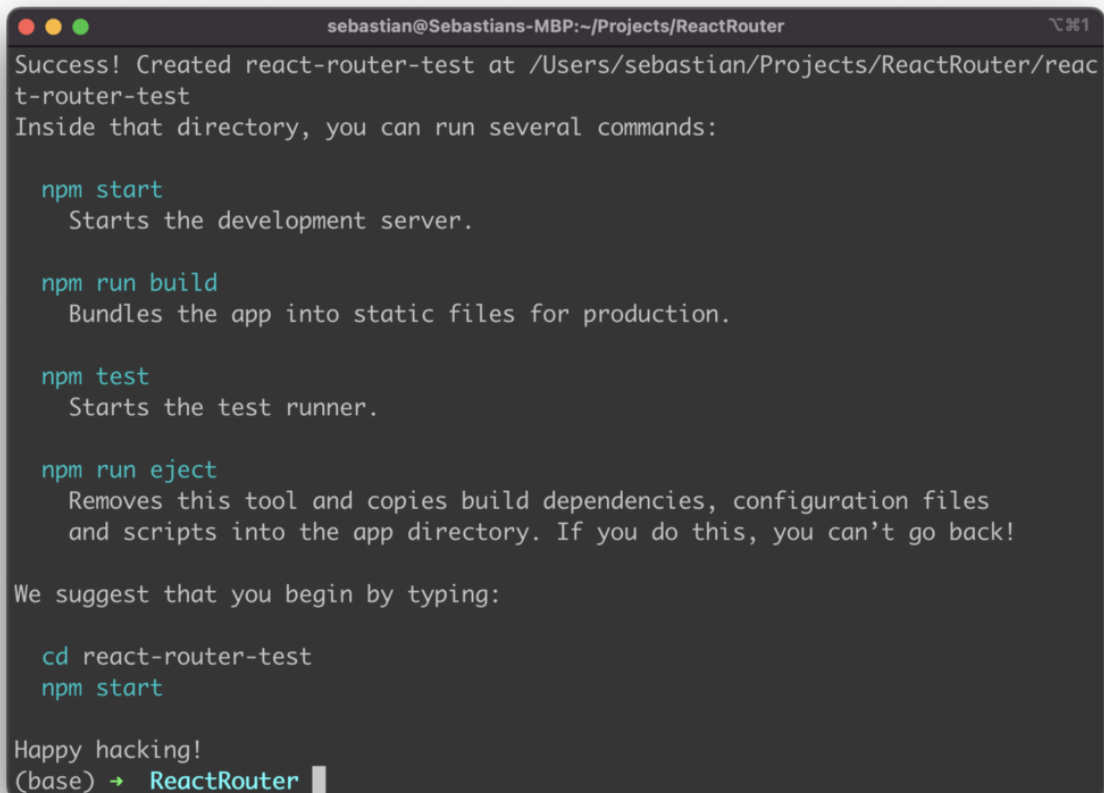
#### **Step 1: Setting Up A React Project**

The easiest way to setup a new React project is to use the create-react-app script in the following way:

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT. (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
$ npx create-react-app react-router-test
```

Once the command has finished downloading and installing the new React project in folder react-router-test you should see the following output on the command line:

A terminal window screenshot with a dark background. The title bar shows 'sebastian@Sebastians-MBP:~/Projects/ReactRouter'. The output text is as follows:

```
Success! Created react-router-test at /Users/sebastian/Projects/ReactRouter/react-router-test
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd react-router-test
  npm start

Happy hacking!
(base) → ReactRouter
```

## Step 2: Installing React Router

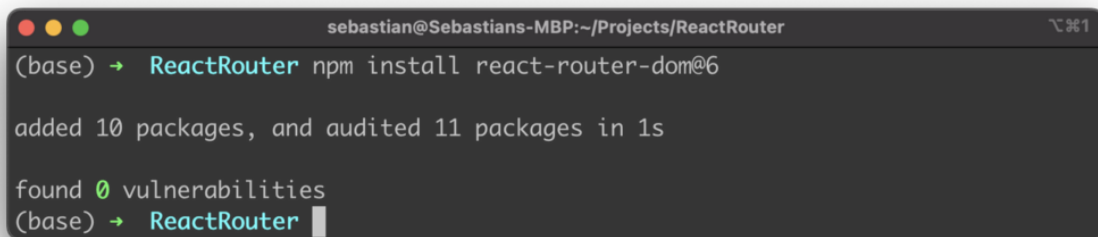
Now we have a new React project folder react-router-test. Change into this new folder by typing in:

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

Then use the Node.js Package Manager [command](#) line tool NPM to install the React Router package:

Here you can see the output on the command line when executing this command:



```
sebastian@Sebastians-MBP:~/Projects/ReactRouter
(base) → ReactRouter npm install react-router-dom@6

added 10 packages, and audited 11 packages in 1s

found 0 vulnerabilities
(base) → ReactRouter
```

### Step 3: Use BrowserRouter Component To Activate The Router

In the project folder open file index.js and change the default implementation to the following:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import { BrowserRouter } from "react-router-dom";
const root = ReactDOM.createRoot(
  document.getElementById('root')
);

root.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

Here we're importing the BrowserRouter component from package react-router-dom. We use BrowserRouter as the top level element in the call of root.render. In the next step we'll embed the route configuration of our app inside the **<BrowserRouter>** element right here.

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

### Step 4: Configure Routes

For adding the route configuration we need extend the import statement to also import Routes and Route. By using these two components we'll add the route configuration in the following way:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';

import { BrowserRouter, Routes, Route } from "react-router-dom";

import App from './App';
import Projects from "./routes/projects";
import About from "./routes/about";

const root = ReactDOM.createRoot(
  document.getElementById('root')
);

root.render(
  <BrowserRouter>
    <Routes>
      <Route path="/" element={<App />} />
      <Route path="projects" element={<Projects />} />
      <Route path="about" element={<About />} />
    </Route>
  </Routes>
</BrowserRouter>
);
```

Inside **<BrowserRouter>** we need to add a **<Routes>** element which is containing the route configuration as single **<Route>** elements as childs.

First a **<Route>** element is added to cover the default route of the app (/) and connect this route to the output of App component. Inside this route configuration two route elements are included which are connecting route /projects to the output of component Projects and route /about to the output of component About.

### Step 5: Implement Components For Routes

Next we need to implement the two React components which have been connected to the routes: Projects and About.

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

*src/routes/projects.jsx:*

```
export default function Projects() {  
  return (  
    <main style={{ padding: "1rem 0" }}>  
      <h2>Projects</h2>  
    </main>  
  );  
}
```

*src/routes/about.jsx:*

```
export default function About() {  
  return (  
    <main style={{ padding: "1rem 0" }}>  
      <h2>About</h2>  
    </main>  
  );  
}
```

## Step 6: Using Link And Outlet Component

Now let's bring everything together and open file App.js which contains the default implementation of the main React application component App. Change the default implementation to the code which is available in the following listing:

```
import './App.css';  
  
import { Link, Outlet } from "react-router-dom";  
  
function App() {  
  return (  
    <div className="App">  
  
      <h1>Learn React Router</h1>  
  
      <br/>  
      <Link to="/">Home</Link> {" | "  
      <Link to="/projects">Projects</Link> {" | "  
      <Link to="/about">About</Link>  
      <br/>  
      <Outlet />  
  
    </div>  
  );  
}
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
}
```

```
export default App;
```

Two components are imported from the react-router-dom package: Link and Outlet.

The Link component is used to output links to the routes of our application. The target route is specified by using the to property.

The Outlet component is used to specify where the output of child routes should be inserted.

### Step 7: Try it out!

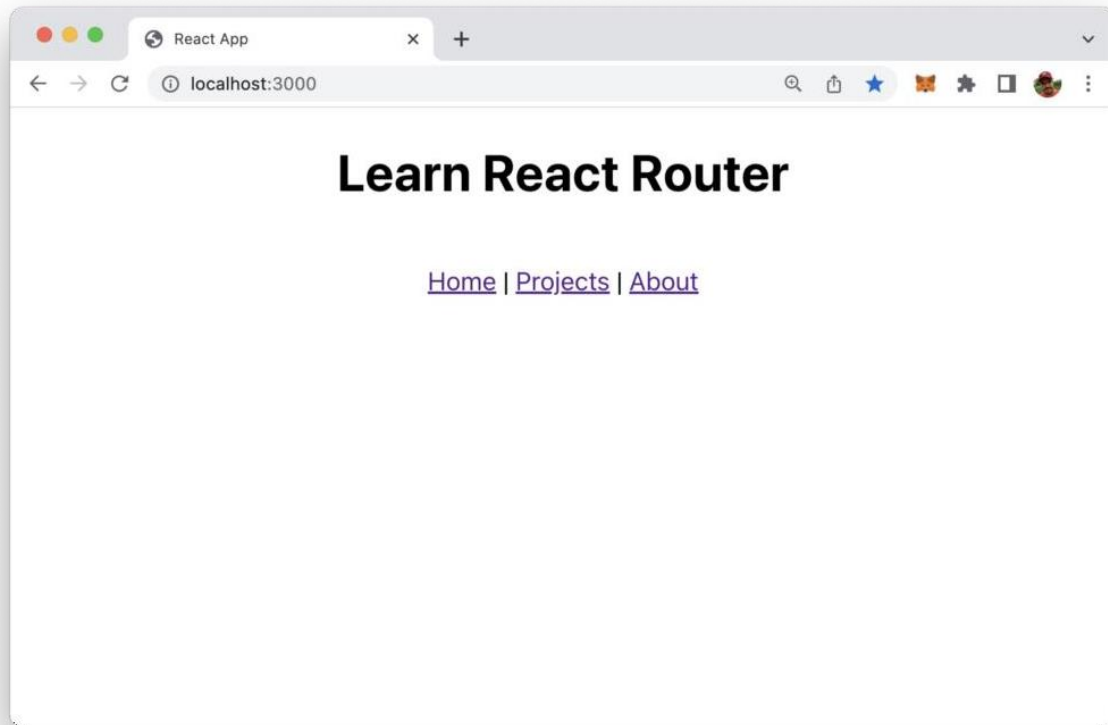
Finally let's start the development web server and try out if everything works.  
Type in

```
$ npm start
```

The web server start and the React application open in the browser, so that you should see the following output:

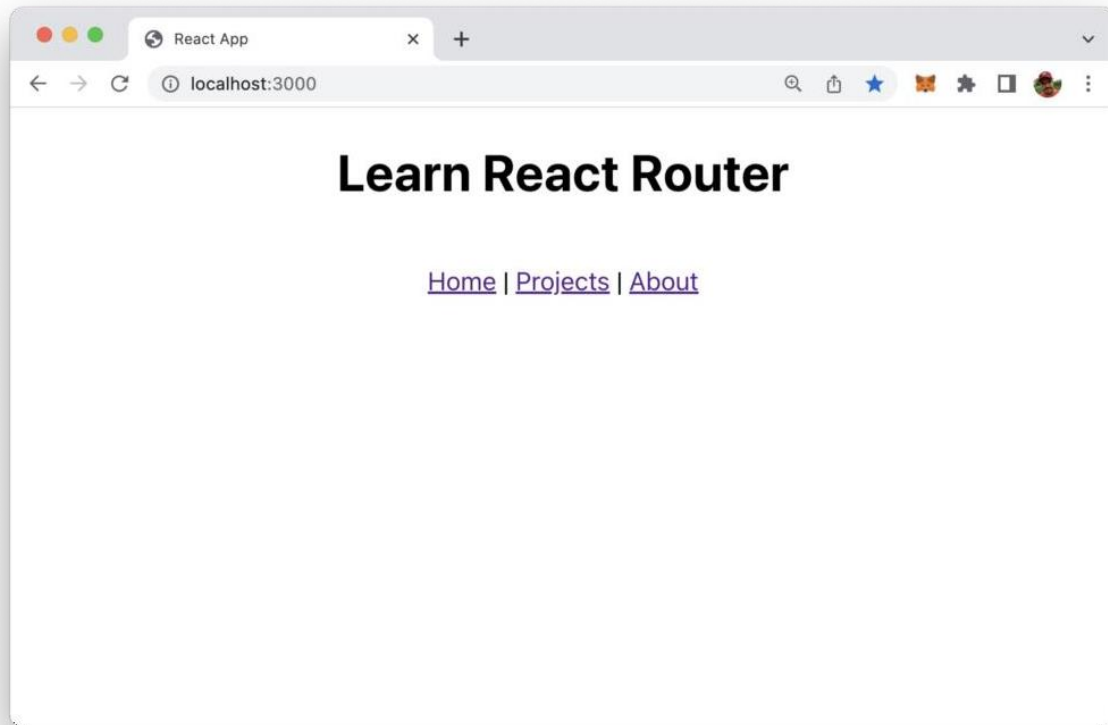


**SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.  
(AFFILIATED TO SAURASHTRA UNIVERSITY)**



Clicking on the link Projects takes you directly to the /projects route of our application and displays the output which is coming from Projects components without reloading the page:

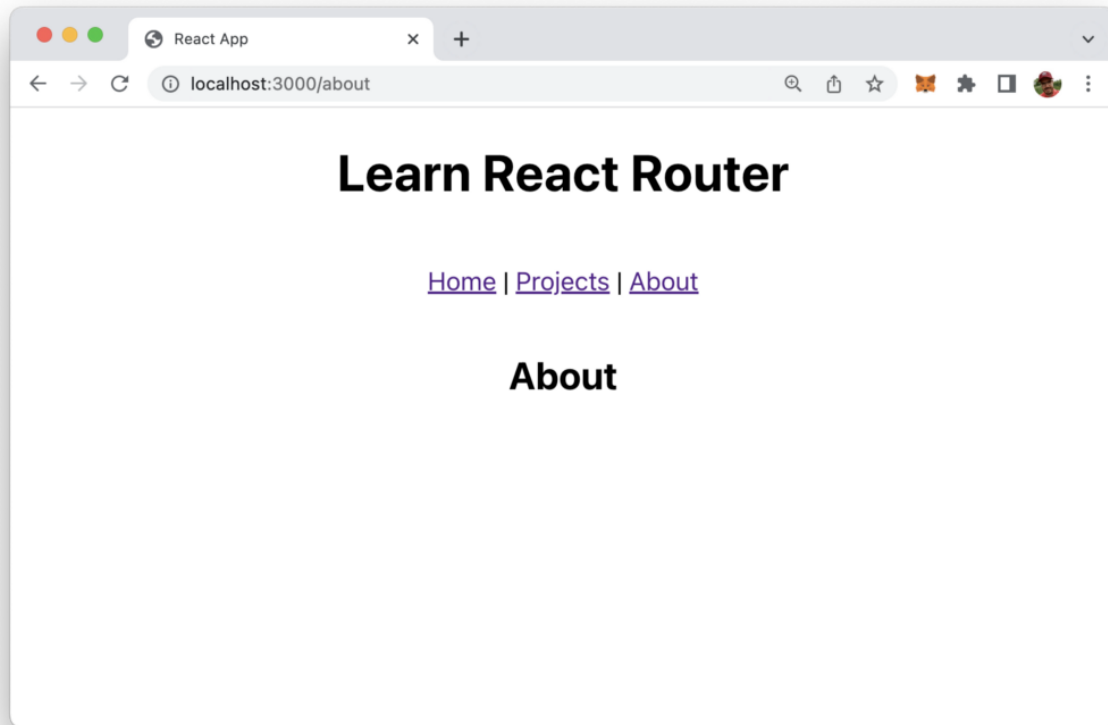
**SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.  
(AFFILIATED TO SAURASHTRA UNIVERSITY)**



Clicking on the About link shows you the output from About component on route /about:

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)



React Route is working as expected. Feel free to play around with the project and start adding additional routes to your web application.

### Navigating to Router Links

**Angular Router: NavigationUsing RouterLink, Navigate, or NavigateByUrl**

#### Introduction

In Angular, RouterLink is a directive for navigating to a different route *declaratively*. Router.navigate and Router.navigateByUrl are two methods available to the Router class to navigate *imperatively* in your component classes.

Let's explore how to use RouterLink, Router.navigate, and Router.navigateByUrl.

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

### Using RouterLink

A typical link in HTML looks like this:

```
<a href="/example">  
  Example HTML link.  
</a>
```

Copy

This example link will direct the user to the `/example` page.

However, a single page application (SPA) does not have different pages to link to. Instead, it has different *views* to display to the user. To allow a user to navigate and change the view, you will want to use the `RouterLink` directive instead of `href`:

```
<a routerLink="/users/sammy">  
  Link that uses a string.  
</a>
```

Copy

This `RouterLink` example will direct the user to the component at `/users/sammy`.

The different URL segments can also be passed in an array like this:

```
<a [routerLink]="['/', 'users', 'sammy']">  
  Link that uses an array.  
</a>
```

Copy

These two examples are formatted differently but will be directed to the same component at `/users/sammy`.

### Relative Paths

You can use `'../'` to go up to higher levels in the navigation using something like this:

```
<a [routerLink]="['../', 'posts', 'sammy']">  
  Link that goes up a level.  
</a>
```

Copy

In that example, if the user is at `/users/sammy`, the navigation will change to `/posts/sammy`.

It is possible to prepend the first URL segment with a `./`, `../`, or no leading slash.

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

### Parameters

You can pass in parameters to the navigation with an object in the list of URL segments:

```
<a [routerLink]="['/', 'users', { display: 'verbose'}, 'sammy']">
```

Link with parameter.

```
</a>
```

Copy

In that example, the Router will navigate to /users;display=verbose/sammy.

### Named Outlets

You can tell the Router what to place in a named outlet with something like this:

```
<a [routerLink]="['/', 'users', { outlets: { side: ['sammy'] } } ]">
```

Link with a side outlet.

```
</a>
```

Copy

In that example, the sammy segment will be placed in the outlet named side.

It is also possible to tell the Router what to place in multiple named outlets with something like this:

```
<a [routerLink]="['/', 'users', { outlets: { side: ['sammy'], footer: ['sharks'] } } ]">
```

Link with a side and footer outlets.

```
</a>
```

Copy

In this example, the sammy segment will be placed in the outlet named side and the sharks segment will be placed in the outlet named footer.

### Using Router

There are two methods available on Angular's Router class to navigate imperatively in your component classes: Router.navigate and Router.navigateByUrl. Both methods return a promise that resolves to true if the navigation is successful, null if there's no navigation, false if the navigation fails, or is completely rejected if there's an error.

To use either method, you'll first want to make sure that the Router class is injected into your component class.

First, import Router into your component class:

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

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

Copy

## SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT. (AFFILIATED TO SAURASHTRA UNIVERSITY)

Then, add `Router` to the dependencies:

```
@Component({
  // ...
})
export class AppComponent {

  constructor(private router: Router) {
    // ...
  }

  // ...
}
```

Copy

Now, you can use `Router.navigate` or `Router.navigateByUrl`.

### [Router.navigate](#)

You pass in an array of URL segments to `Router.navigate`.

Here's a basic example using the `Router.navigate` method:

```
goPlaces() {
  this.router.navigate(['/', 'users']);
}
```

Copy

And here's an example demonstrating how `Router.navigate` is *thenable*:

```
goPlaces() {
  this.router.navigate(['/', 'users'])
    .then(nav => {
      console.log(nav); // true if navigation is successful
    }, err => {
      console.log(err) // when there's an error
    });
}
```

Copy

If `Router.navigate` is successful in this example, it will display `true`.

If `Router.navigate` is unsuccessful in this example, it will display an error.

### [Router.navigateByUrl](#)

`Router.navigateByUrl` is similar to `Router.navigate`, except that a string is passed in instead of URL segments. The navigation should be absolute and start with a `/`.

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

Here's a basic example using the `Router.navigateByUrl` method:

```
goPlaces() {  
  this.router.navigateByUrl('/users;display=verbose/sammy');  
}
```

Copy

In this example, `Router.navigateByUrl` will navigate to `/users;display=verbose/sammy`.

### Conclusion

In this article, you learned about navigation in Angular applications. You were introduced to `RouterLink`, `Router.navigate`, and `Router.navigateByUrl`.

If you'd like to learn more about Angular, check out [our Angular topic page](#) for exercises and programming projects.

## What is path in Angular routing?

The path refers to the part of the URL that determines a unique view that should be displayed, and component refers to the Angular component that needs to be associated with a path. Based on a route definition that we provide (via a static `RouterModule`).

### ❖ How to get current route URL in Angular

get current route Url in Angular

[Arunkumar Gudelli](#)

Last updated on Feb 11, 2021 2 min read

Steps to get current route URL in Angular.

1. Import `Router, NavigationEnd` from '@angular/router' and inject in the constructor.
2. Subscribe to the `NavigationEnd` event of the router.
3. Get the current route url by accessing `NavigationEnd`'s `url` property.

Now we will take an example and understand it further.

I have created an Angular app which contains three routes. About, Service and Dashboard.

```
import { Component } from '@angular/core';  
import { Router, NavigationEnd } from '@angular/router';
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
@Component({
  selector: 'my-app',
  templateUrl: './app.component.html',
  styleUrls: [ './app.component.css' ]
})
export class AppComponent {

  name = 'Get Current Url Route Demo';
  currentRoute: string;

  constructor(private router: Router){
    console.log(router.url);

    router.events.filter(event => event instanceof NavigationEnd)
      .subscribe(event =>
        {
          this.currentRoute = event.url;
          console.log(event);
        });
  }
}
```

I have created a variable called `currentRoute`, to display current router url value in the component HTML file.

In the subscribe event of `NavigationEnd`, I am updating `currentRoute` value.

```
<ul>
  <li routerLinkActive="active">
    <a [routerLink]="['/about']">About</a>
  </li>
  <li routerLinkActive="active">
    <a [routerLink]="['/service']">Service</a>
  </li>
  <li routerLinkActive="active">
    <a [routerLink]="['/dashboard']">Dashboard</a>
  </li>
</ul>
```

The current Route Is {{ currentRoute }}

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

Here is the demo.

You might get current route by accessing `router.url` as well.



# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

But If you are navigating the routes with Hash location strategy, the `router.url` is always return “/”.

So it's advisable to listen for `NavigationEnd` event of router to get the current route url in Angular.

*NavigationEndEvent*

### RouterLinkActive DIRECTIVE

Tracks whether the linked route of an element is currently active, and allows you to specify one or more CSS classes to add to the element when the linked route is active.

Exported from

- [RouterModule](#)

Selectors

- [\[routerLinkActive\]](#)

Properties

Property	Description
----------	-------------

links:	
<a href="#">QueryList</a> < <a href="#">RouterLink</a> >	

isActive	<b><i>Read-Only</i></b>
----------	-------------------------

@ <a href="#">Input</a> () routerLinkActiveOptions: { exact: boolean; }   <a href="#">IsActiveMatchOptions</a>	
---	--

Options to configure how to determine if the router link is active. These options are passed to the <a href="#">Router.isActive()</a> function. See also: <ul style="list-style-type: none"><li>• <a href="#">Router#isActive</a></li></ul>
--

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

### Property

### Description

@[Input\(\)](#)

ariaCurrentWhenActive?:  
'page' | 'step' | 'location' |  
'date' | 'time' | true | false

Aria-current attribute to apply when the router link is active.  
Possible values: 'page' | 'step' | 'location' | 'date' | 'time' | true | false.  
See also:

- [aria-current](#)

@[Output\(\)](#)

isActiveChange:  
[EventEmitter](#)<boolean>

### Read-Only

You can use the output isActiveChange to get notified each time the link becomes active or inactive.

Emits: true -> Route is active false -> Route is inactive

```
content_copy<a
  routerLink="/user/bob"
  routerLinkActive="active-link"
```

```
(isActiveChange)="this.onRouterLinkActive($event)">Bob</a>
```

@[Input\(\)](#)

[routerLinkActive](#): string |  
string[]

### Write-Only

Template variable references

### Identifier

### Usage

[routerLinkActive](#)

#myTemplateVar=["routerLinkActive"](#)

## Description

Use this directive to create a visual distinction for elements associated with an active route. For example, the following code highlights the word "Bob" when the router activates the associated route:

```
content_copy<a routerLink="/user/bob" routerLinkActive="active-link">Bob</a>
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

Whenever the URL is either '/user' or '/user/bob', the "active-link" class is added to the anchor tag. If the URL changes, the class is removed.

You can set more than one class using a space-separated string or an array. For example:

```
content_copy<a routerLink="/user/bob" routerLinkActive="class1 class2">Bob</a>
```

```
<a routerLink="/user/bob" [routerLinkActive]="['class1', 'class2']">Bob</a>
```

To add the classes only when the URL matches the link exactly, add the option exact: true:

```
content_copy<a routerLink="/user/bob" routerLinkActive="active-link"
[routerLinkActiveOptions]={exact:
true}>Bob</a>
```

To directly check the isActive status of the link, assign the [RouterLinkActive](#) instance to a template variable. For example, the following checks the status without assigning any CSS classes:

```
content_copy<a routerLink="/user/bob" routerLinkActive #rla="routerLinkActive">
Bob {{ rla.isActive ? '(already open)' : '' }}
</a>
```

You can apply the [RouterLinkActive](#) directive to an ancestor of linked elements. For example, the following sets the active-link class on the <div> parent tag when the URL is either '/user/jim' or '/user/bob'.

```
content_copy<div routerLinkActive="active-link" [routerLinkActiveOptions]={exact: true}>
<a routerLink="/user/jim">Jim</a>
<a routerLink="/user/bob">Bob</a>
</div>
```

The [RouterLinkActive](#) directive can also be used to set the aria-current attribute to provide an alternative distinction for active elements to visually impaired users.

For example, the following code adds the 'active' class to the Home Page link when it is indeed active and in such case also sets its aria-current attribute to 'page':

```
content_copy<a routerLink="/" routerLinkActive="active" ariaCurrentWhenActive="page">Home
Page</a>
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)



### Adding navigation

This guide builds on the first step of the Getting Started tutorial, [Get started with a basic Angular app](#).

At this stage of development, the online store application has a basic product catalog.

In the following sections, you'll add the following features to the application:

- Type a URL in the address bar to navigate to a corresponding product page
- Click links on the page to navigate within your single-page application
- Click the browser's back and forward buttons to navigate the browser history intuitively

---

### Associate a URL path with a component

The application already uses the Angular [Router](#) to navigate to the ProductListComponent. This section shows you how to define a route to show individual product details.

1. Generate a new component for product details. In the terminal generate a new product-details component by running the following command:

```
content_copyng generate component product-details
```

2. In app.module.ts, add a route for product details, with a path of products/:productId and ProductDetailsComponent for the component.  
src/app/app.module.ts

```
content_copy@NgModule({  
  imports: [  
    BrowserModule,  
    ReactiveFormsModule,  
    RouterModule.forRoot([  
      { path: '', component: ProductListComponent },  
      { path: 'products/:productId', component: ProductDetailsComponent },  
    ])  
  ],  
  declarations: [  

```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
AppComponent,  
TopBarComponent,  
ProductListComponent,  
ProductAlertsComponent,  
ProductDetailsComponent,  
],
```

3. Open product-list.component.html.
4. Modify the product name anchor to include a [routerLink](#) with the product.id as a parameter.  
src/app/product-list/product-list.component.html

```
content_copy<div *ngFor="let product of products">
```

```
<h3>
```

```
<a
```

```
  [title]="product.name + ' details'"
```

```
  [routerLink]="['/products', product.id]">
```

```
    {{ product.name }}
```

```
</a>
```

```
</h3>
```

```
<!-- ... -->
```

```
</div>
```

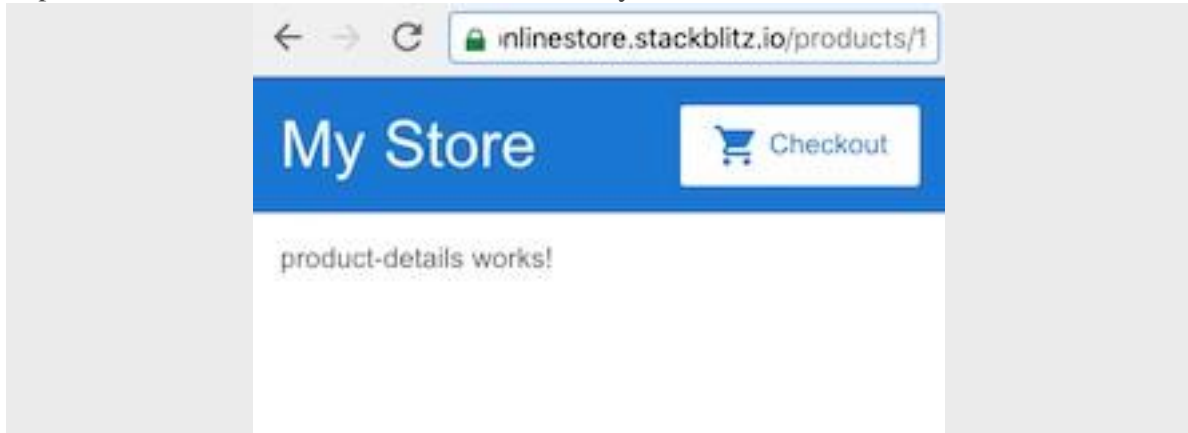
The [RouterLink](#) directive helps you customize the anchor element. In this case, the route, or URL, contains one fixed segment, /products. The final segment is variable, inserting the id property of the current product. For example, the URL for a product with an id of 1 would be similar to <https://getting-started-myfork.stackblitz.io/products/1>.

5. Verify that the router works as intended by clicking the product name. The application should display the ProductDetailsComponent, which currently says "product-details works!"

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

Notice that the URL in the preview window changes. The final segment is products/# where # is the number of the route you clicked.



### View product details

The ProductDetailsComponent handles the display of each product. The Angular Router displays components based on the browser's URL and [your defined routes](#).

In this section, you'll use the Angular Router to combine the products data and route information to display the specific details for each product.

1. In product-details.component.ts, import [ActivatedRoute](#) from @angular/router, import [OnInit](#) from @angular/core, and the products array from ../products.  
src/app/product-details/product-details.component.ts

```
content_copyimport { Component, OnInit } from '@angular/core';  
  
import { ActivatedRoute } from '@angular/router';
```

```
import { Product, products } from '../products';
```

2. Define the product property. The implements [OnInit](#) statement indicates that the class implements the [OnInit](#) interface, requiring the implementation of the ngOnInit method for initialization tasks when the component is created.  
src/app/product-details/product-details.component.ts

```
content_copyexport class ProductDetailsComponent implements OnInit {  
  
  product: Product | undefined;  
  
  /* ... */
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
}
```

3. Inject [ActivatedRoute](#) into the constructor() by adding private route: [ActivatedRoute](#) as an argument within the constructor's parentheses.  
src/app/product-details/product-details.component.ts

```
content_copyexport class ProductDetailsComponent implements OnInit {
```

```
product: Product | undefined;
```

```
constructor(private route: ActivatedRoute) { }
```

```
}
```

[ActivatedRoute](#) is specific to each component that the Angular Router loads. [ActivatedRoute](#) contains information about the route and the route's parameters. By injecting [ActivatedRoute](#), you are configuring the component to use a service. The [Managing Data](#) step covers services in more detail.

4. In the ngOnInit() method, extract the productId from the route parameters and find the corresponding product in the products array.  
src/app/product-details/product-details.component.ts

```
content_copyngOnInit() {
```

```
// First get the product id from the current route.
```

```
const routeParams = this.route.snapshot.paramMap;
```

```
const productIdFromRoute = Number(routeParams.get('productId'));
```

```
// Find the product that correspond with the id provided in route.
```

```
this.product = products.find(product => product.id === productIdFromRoute);
```

```
}
```

The route parameters correspond to the path variables you define in the route. To access the route parameters, we use route.snapshot, which is the [ActivatedRouteSnapshot](#) that contains information about the active route at that particular moment in time. The URL that matches

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

the route provides the productId . Angular uses the productId to display the details for each unique product.

5. Update the ProductDetailsComponent template to display product details with an `*ngIf`. If a product exists, the `<div>` renders with a name, price, and description.  
src/app/product-details/product-details.component.html

```
content_copy<h2>Product Details</h2>
```

```
<div *ngIf="product">
```

```
<h3>{{ product.name }}</h3>
```

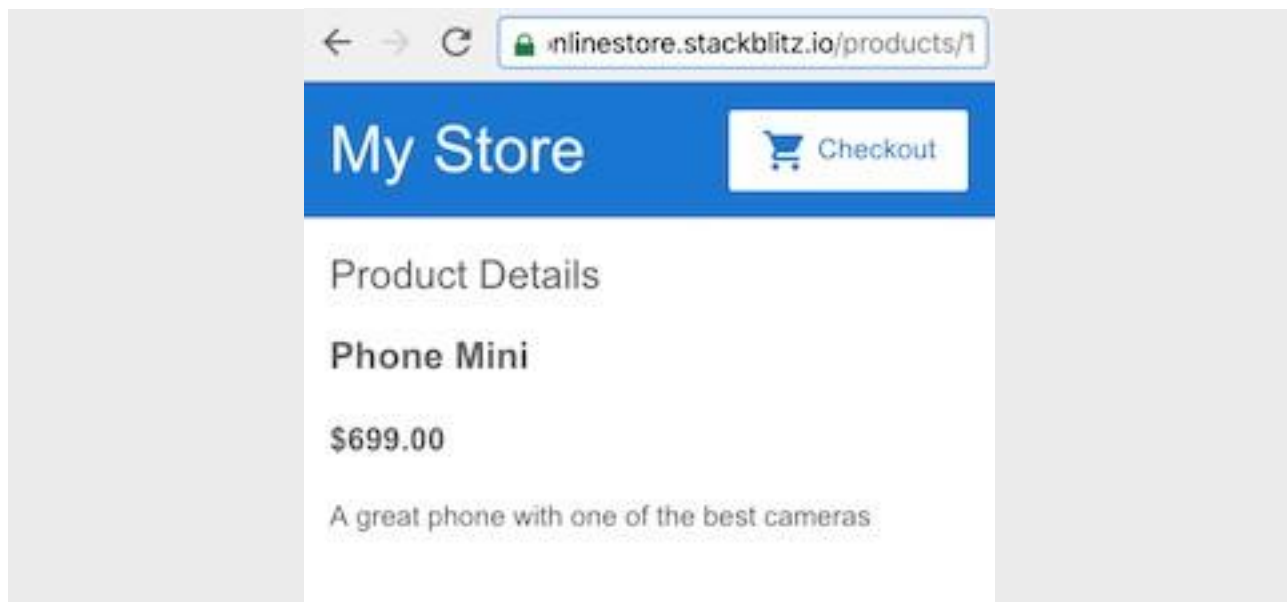
```
<h4>{{ product.price | currency }}</h4>
```

```
<p>{{ product.description }}</p>
```

```
</div>
```

The line, `<h4>{{ product.price | currency }}</h4>`, uses the `currency` pipe to transform `product.price` from a number to a currency string. A pipe is a way you can transform data in your HTML template. For more information about Angular pipes, see [Pipes](#).

When users click on a name in the product list, the router navigates them to the distinct URL for the product, shows the ProductDetailsComponent, and displays the product details.



For more information about the Angular Router, see [Routing & Navigation](#).



# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)



### Styling Active Router Link in Angular

- Styling the active router link in angular allows the user to differentiate between the active router link and inactive router links. Angular provides a special mechanism to work with active router links.

#### Approach:

- Create the Angular app to be used.
- Create the header component that contains the navigation links.
- Then apply the “**routerLinkActive**” on each router link and provide the CSS class to this property. Here we have created the “**active**” class in CSS file.
- Provide the { **exact : true** } to the root route to avoid multiple active router links.

#### Syntax:

```
<a routerLink="/" routerLinkActive="active" >Home</a>
```

**Example:** We have created the header component with specified routes.

- header.component.html

```
<span>

<ul>

  <li><a routerLink="/" routerLinkActive="active">

    Home

  </a></li>

  <li><a routerLink="/products"

    routerLinkActive="active">Products

  </a></li>

  <li><a routerLink="/about"
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
routerLinkActive="active">About Us

</a></li>

<li><a routerLink="/contact"

routerLinkActive="active">Contact Us

</a></li>

</ul>

</span>

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

Here we have provided the “**routerLinkActive**” which is routing functionality that automatically activate the current route, and we have to provide the CSS class as well. Here in **routerLinkActive** = “**active**” active is a CSS class that automatically applied to the activated route.

- header.component.css

```
.active{

background: 'white'

}
```

But here, it still causes an issue our Home route is always active even we navigate to some other route the reason behind this is the way “*routerLinkActive*” works. The home route works on “*localhost:4200/*” and other routes are “*localhost:4200/about*” so “*routerLinkActive*” finds “*localhost:4200/*” inside every other route and the Home router link is always active to deal with this angular provide another directive called *routerLinkActiveOptions*.

**Updated**

- header.component.htm

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
<span>

<ul>

  <li><a routerLink="/" routerLinkActive="active"

    [routerLinkActiveOptions]={exact:true}>Home

  </a></li>

  <li><a routerLink="/products"

    routerLinkActive="active">Products

  </a></li>

  <li><a routerLink="/about"

    routerLinkActive="active">About Us

  </a></li>

  <li><a routerLink="/contact"

    routerLinkActive="active">Contact Us

  </a></li>

</ul>

</span>

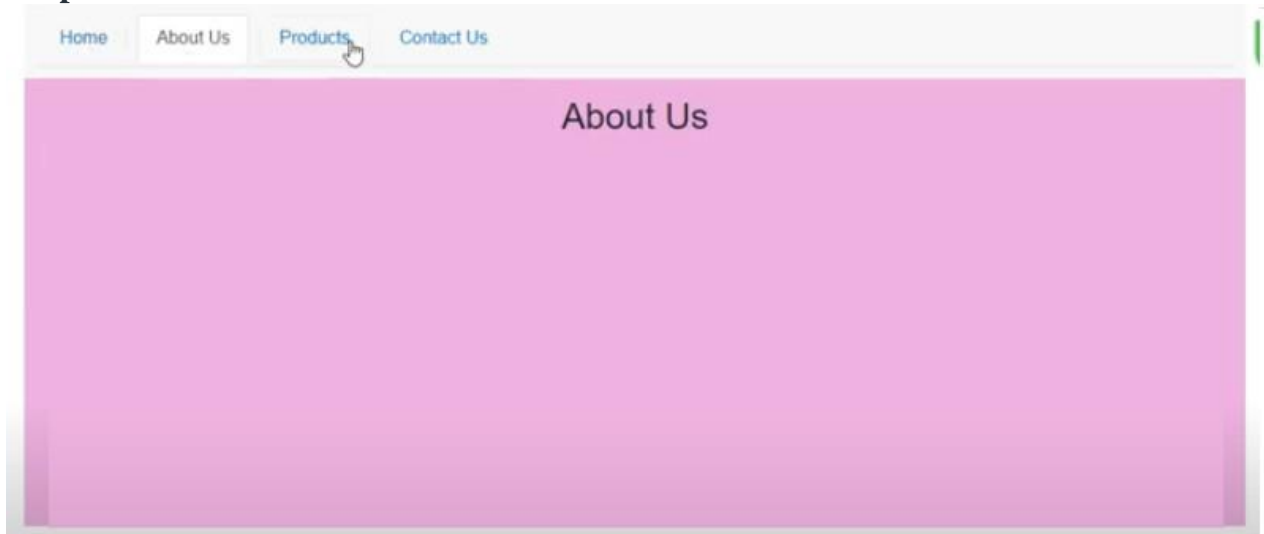
<router-outlet></router-outlet>
```

So *routerLinkActiveOptions* allow only the exact path match as an active route for the Home component.

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

Output:



## Navigating to Routes from Code in Angular

### Introduction

Hello friends! As you know, Angular is a single-page application, and it doesn't have multipage HTML pages to create different routes, so we require a client-side routing solution. The Angular Router helps us to create client-side routing in a few simple steps, and then we can render using [routerLink](#). Today, I will walk you through different options to navigate from components to various other components using code so you can navigate the user dynamically using different conditions.

### What We Will Cover

- The need for dynamic routing
- Making routes with different components
- Navigating between components using routes from code

### The Need for Dynamic Routing

Let's take a real-life and straightforward example. Suppose you have multiple roles in your application, and depending on the role, you have to decide whether or not users are authorized to

## SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT. (AFFILIATED TO SAURASHTRA UNIVERSITY)

access your application. If they have the authority, you'll navigate them to the home page of the application; otherwise, you'll take them to a different page, preferably with a 403 status code.

In this scenario, you will need to decide the route in one hit at the time of form submission on the login page. So what are you going to do?

Here dynamic routing comes into the picture. First, you will check the condition, and depending on that condition, you will dynamically decide the routes where a user will be sent.

Let's get started with the next section by making routes in [app-routing.module.ts](#).

## Making Routes with Different Components

Now we're going to make some of the components that will help us to understand Angular Router more clearly. I am going to create three components: [FirstComponent](#), [SecondComponent](#), and [FirstChildComponent](#), respectively. [FirstChildComponent](#) will be used in the [FirstComponent](#) as a child component.

The following commands will create the components.

```
1ng g c first --skipTests=true
2ng g c second --skipTests=true
3ng g c first-child --skipTests=true
```

shell

We have successfully created the components; now, we will map it with different URI. After successfully configuring [app-routing.module.ts](#), it will look as follows.

```
1import { NgModule } from '@angular/core';
2import { Routes, RouterModule } from '@angular/router';
3import { SecondComponent } from './second/second.component';
4import { FirstComponent } from './first/first.component';
5import { FirstChildComponent } from './first-child/first-child.component';
6
7const routes: Routes = [
8 {
9   path: 'first',
10  component: FirstComponent,
11  children: [
12 {
```

**SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.**  
**(AFFILIATED TO SAURASHTRA UNIVERSITY)**

```
13   path: 'first-child',
14   component: FirstChildComponent
15 }
16 ]
17 },
18 {
19   path: 'second',
20   component: SecondComponent
21 },
22 {
23   path: '',
24   redirectTo: '/first',
25   pathMatch: 'full'
26 }
27];
28
29@NgModule({
30 imports: [RouterModule.forRoot(routes)],
31 exports: [RouterModule]
32})
33export class AppRoutingModule { }
```

ts

In the routes file, we have created two main routes, `/first` and `/second`, and a default route that will redirect to `/first`. Mainly, it is for redirecting the user to the home page initially. We have also created children routing to explain the dynamic navigation for the parent-child relationship.

It's time to write the chunks of code for navigating between the component using routes.

## Navigate Between the Components using Routes from Code

I am not going to write a condition for now; rather, I will make a couple of buttons in [app.component.html](#). On click, it will navigate to different components.

```
1<div>
2  <button type="button" class="btn" (click)="navigateToFirst()">First</button>
3  <button type="button" class="btn" (click)="navigateToSecond()">Second</button>
4</div>
5
6<router-outlet></router-outlet>
```

html

As you can see, we have successfully added two buttons in [app.component.html](#), and on click events methods are going to call [navigateToFirst\(\)](#) and [navigateToSecond\(\)](#), respectively. These methods will call the [navigate\(\)](#) method of the [Router](#) class to navigate to another view. So let's add these functions to our [app.component.ts](#) file.

```
1import { Component } from '@angular/core';
2import { Router } from '@angular/router';
3
4@Component({
5  selector: 'app-root',
6  templateUrl: './app.component.html',
7  styleUrls: ['./app.component.scss']
8})
9export class AppComponent {
10
11  constructor(private _router: Router) { }
12
13  navigateToFirst() {
```

## SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT. (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
14  this._router.navigate(['first'])
15  }
16  navigateToSecond() {
17    this.router.navigateByUrl('/second')
18  }
19
20}
```

ts

Here we have injected the router class in the constructor that enables us to navigate from one view to another. It has two methods, `navigate` and `navigateByUrl`, that navigate the routes. They are similar; the only difference is that the `navigate` method takes an array that joins together and works as the URL, and the `navigateByUrl` method takes an absolute path. For example, if we have to navigate to `/students/1`, we can navigate to this URL in two ways.

1. `this._router.navigate(['students',1])`
2. `this._router.navigateByUrl('/students/1')`

On button click, you can add some condition to navigate the user on different conditions.

Now I am going to apply some of the CSS to beautify our application. I'll put it in the `style.scss` file.

```
1.btn{
2  color: white;
3  background-color: blue;
4  border: none;
5  padding: .5em 2em;
6  border-radius: 2em;
7  box-shadow: 0 0 2px 2px;
8  font-size: 18px;
9  margin-left: 5px;
10 margin-right: 5px;
```



## SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT. (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
11 outline: none;
12 &:hover{
13     background-color: gray;
14     cursor: pointer;
15 }
16}
```

SCSS

I have used SCSS here. If you're not familiar with it, you can read about it [here](#). You can use CSS, too, if you're not comfortable with SCSS.

Our output will look like this.



first works!

Now It's time to bring the `FirstChildComponent` into the picture. We'll make a button like `app.component.html` in `first.component.html`.

```
1<p>first works!</p>
2<button type="button" class="btn" (click)="navigateToFirstChild()">First</button>
3<router-outlet></router-outlet>
```

html

After making changes in `first.component.ts`, it will look like the above snippet in which we have a button that will call a function on click event, which will then navigate it and show the child view.

Now we're going to make a function in `first.component.ts` to navigate to `FirstChildComponent`.

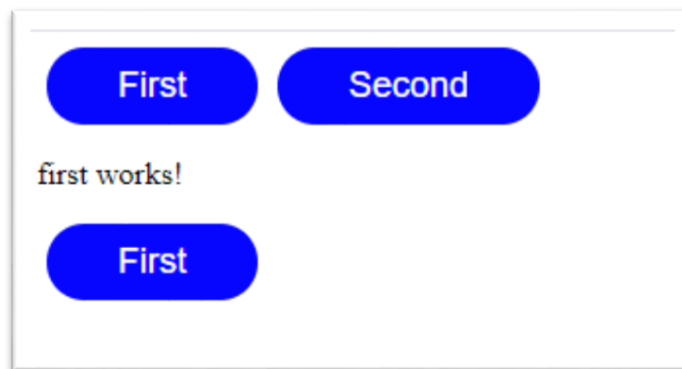
```
1import { Component, OnInit } from '@angular/core';
2import { Router, ActivatedRoute } from '@angular/router';
3
4@Component({
5  selector: 'app-first',
6  templateUrl: './first.component.html',
7  styleUrls: ['./first.component.scss']
8})
```

## SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT. (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
8})
9export class FirstComponent implements OnInit {
10
11  constructor(
12    private _router: Router,
13    private _activatedRoute: ActivatedRoute
14  ) { }
15
16  navigateToFirstChild() {
17    this._router.navigate(["first-child"], {relativeTo: this._activatedRoute}) 18
18  }
19
20}
```

ts

Note: In the `navigate()` method, I have passed an extra parameter. It is a kind of object known as `NavigateExtras`. We have used `relativeTo` of `NavigateExtras`, and in the value, we have given the instance of `ActivatedRoute`. So you don't need to provide the whole URL in the array, and it will automatically extend the current parameter after the existing URL. Finally, our output looks like this.



# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)



## Angular.js: using relative paths

One of the issues I had with Angular.js (but also with other JavaScript frameworks) is that URL's you create on the client are relative to the server root address. This is fine as long as your application is hosted on the server root(e.g. [www.myservername.com](http://www.myservername.com)), but the moment you start using virtual directories(e.g. [www.myservername.com/virtualdirectoryname](http://www.myservername.com/virtualdirectoryname) ), you're into trouble.

Let's have a look for example to the following Angular service:

```
"use strict";
(function (app) {
    var applicationsService = function ($http) {
        var getApplications = function () {
            return $http.get("/api/applications");
        };
        return {
            getApplications: getApplications,
        };
    };
    applicationsService.$inject = ["$http"];
    app.factory("applicationsService", applicationsService);
})(angular.module("maintenanceApp"));
```

The problem is that when you host this service in a virtual directory, the HTTP call will be send to <http://www.myservername.com/api/applications> and not to [www.myservername.com/virtualdirectoryname/api/applications](http://www.myservername.com/virtualdirectoryname/api/applications) as you would expect.

There a multiple ways to solve this issue. I did it by rendering the rootUrl inside my ASP.NET Razor view. This url is generated by the server and embedded in the page:

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link id="linkRoot" href="/" />
</head>
```

Inside my Angular app, I added some code that read this value from the html and register it as a constant value:

```
var maintenanceApp = angular.module('maintenanceApp', ['ngRoute']);
maintenanceApp
  .config(['$provide', function ($provide) {
    var rootUrl = $("#linkRoot").attr("href");
    $provide.constant('rootUrl', rootUrl);
  }])
```

This allows me to inject this value anywhere I need it, let's have a look at the refactored service:

```
"use strict";

(function (app) {
  var applicationsService = function ($http, rootUrl) {

    var getApplications = function () {
      return $http.get(rootUrl+ "/api/applications");
    };

    return {
      getApplications: getApplications,
    };
  };
  applicationsService.$inject = ["$http", "rootUrl"];
  app.factory("applicationsService", applicationsService);
}(angular.module("maintenanceApp"));
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

### Passing Parameters to Routes and fetching route parameters

In this post we'll see how to pass and retrieve route parameters in Angular routing.

- Table of contents
- Route parameters in Angular
- Passing Route parameters in Angular
- Retrieving route parameters in Angular
- Route parameters in Angular example

#### ❖ Route parameters in Angular

In our web apps we do need to navigate to a specific resource. For example we want the details of a specific account number then we can navigate to that account number by using URL- /account/A1001 or to another account by using URL- /account/A1002.

It is not practical to hardcode the path segment for each account number, you will use a route parameter instead that acts as a placeholder. Value passed for the placeholder becomes the value of the route parameter.

In a route that takes a parameter, route parameter is specified by prefixing it with a colon. So the route definition will be like- /route/:routeparam

You can pass more than one route parameter too- /route/:param1/:param2/:param3

For example if you have to give route definition for a route- /account/A1001 where /A1001 part represents an account number and should be passed as a route parameter.

```
{path: 'account/:acctno', component: COMPONENT_NAME}
```

#### ❖ Passing Route parameters in Angular

1. You can pass route parameters with RouterLink directive. For a dynamic link, pass an array of path segments (which includes path and route parameters). For example

```
[router Link]="['/account', accountnumber]"
```

generates a link to /account/A1001 or to /account/A1002 based on what is passed as value for the string variable accountnumber.

2. You can also pass route parameters programmatically using Router.navigate() method. For example onAccountClick() method is called with account number as argument and then it creates a URL with route parameter using navigate() method. Here account number gets added as a route parameter to the current route.

```
onAccountClick(accountNo: string){  
  this.router.navigate([accountNo], {relativeTo:this.route});  
}
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

### ➤ Retrieving route parameters in Angular

To extract the parameter, params observable of the ActivatedRoute interface is used and we subscribe to it so that when ever there is a change in the route parameter it is extracted into a variable.

```
this.route.params.subscribe((params: Params)=> this.acctNo = params['acctno']);
```

Here we have passed the account parameter from the route using acctNo

```
this.acctNo = this.route.snapshot.params['acctno'];
```

### ➤ Route parameters in Angular example

In the example we show user a list of account numbers. Then the details of the account number, that is clicked, are showed using a separate component. In this scenario you can pass the clicked account number as route parameter.

Here are the route definitions for the routes. Here AccountsComponent displays all the account numbers for a user and AccountComponent shows details for the selected account number.

```
const routes: Routes = [  
  {path: 'home', component: HomeComponent},  
  {path: 'account', component: AccountsComponent},  
  {path: 'account/:acctno', component: AccountComponent},  
  {path: 'service', component: ServiceComponent},  
  {path: '', redirectTo: '/home', pathMatch: 'full'}  
];
```

As you can see there is a route with route parameter- {path: 'account/:acctno', component: AccountComponent}

Code for menu and adding link for routes is done in the app.component.html template itself for this example.

```
<nav class="navbar navbar-expand-md bg-dark navbar-dark">  
  <div class="container-fluid">  
    <div class="collapse navbar-collapse" id="collapsibleNavbar">  
      <ul class="nav navbar-nav">  
        <li class="nav-item" routerLinkActive="active">  
          <a class="nav-link" routerLink="/home">Home</a>  
        </li>  
        <li class="nav-item" routerLinkActive="active">
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
<a class="nav-link" router Link="/account">Accounts</a>
</li>
<li class="nav-item" router LinkActive="active">
  <a class="nav-link" router Link="/service">Services</a>
</li>
</ul>
</div>
</div>
</nav>
<div class="container">
  <div class="row"><p></p></div>
  <div class="row">
    <div class="col-md-12">
      <router-outlet></router-outlet>
    </div>
  </div>
</div>
```

### Components

We'll concentrate here on AccountsComponent and AccountComponent for the code of other components please refer- [Angular Routing Concepts With Example](#).

#### AccountsComponent (accounts.component.ts)

In AccountsComponent we have a `routerLink` property and a method `onAccountClick()` to navigate to another route.

```
import { Component } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';

@Component({
  selector: 'app-accounts',
  templateUrl: './accounts.component.html'
})
export class AccountsComponent {
  accounts = ['A1001', 'A1002'];
  constructor(private router: Router, private route: ActivatedRoute) {}
  onAccountClick(account: string){
    this.router.navigate([account], {relativeTo:this.route});
  }
}
```

Two classes Router class and ActivatedRoute class are injected into the component.

- **Router class** has `navigate()` method using which we navigate to a URL dynamically.

## SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT. (AFFILIATED TO SAURASHTRA UNIVERSITY)

- **ActivatedRoute** class provides access to information about the current route.

This line of code instructs Angular to navigate to the path which is relative to current route (localhost:4200/account in our example) and adds the value of account to it making it a route in this format - http://localhost:4200/account/ACCOUNT\_NUMBER

```
this.router.navigate([account], {relativeTo:this.route});
```

Since we already have a route definition with a route parameter {path: 'account/:acctno', component: AccountComponent} which matches any route in this format http://localhost:4200/account/ACCOUNT\_NUMBER so that's how AccountComponent gets called.

### accounts.component.html

```
<div class= "row">
  <div class="col-xs-4 col-md-6">
    <h2>Account Numbers</h2>
    <div class="list-group">
      <a [routerLink]="''
        (click)="on AccountClick(account)"
        class="list-group-item"
        *ngFor="let account of accounts">
        {{ account }}
      </a>
    </div>
  </div>
</div>
```

If you want to use RouterLink directive then your template can be written as-

```
<div class= "row">
  <div class="col-xs-4 col-md-6">
    <h2>Account Numbers</h2>
    <div class="list-group">
      <a [routerLink]="['/account', account]" class="list-group-item"
        *ngFor="let account of accounts">
```



## SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT. (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
    {{ account }}  
  </a>  
</div>  
</div>  
</div>
```

In this case onAccountClick() method can be removed from typescript code as navigation is configured in the template itself.

### AccountComponent (account.component.ts)

In this component we simulate a scenario where we have details for all the account numbers and we have to get the details for the account number which is sent as route parameter.

To extract the parameter route.params observable is used and we subscribe to it so that when ever there is a change the parameter is extracted into a acctNo variable. Using the fetched account number we find the related object in the array using the find() method.

```
import { Component, OnInit } from '@angular/core';  
import { ActivatedRoute, Params } from '@angular/router';  
  
@Component({  
  selector: 'app-account',  
  templateUrl: './account.component.html'  
})  
export class AccountComponent implements OnInit {  
  acctNo: string;  
  account: {accountnumber: string, type: string, balance: number};  
  constructor(private route: ActivatedRoute) { }  
  accountDetails = [  
    {  
      accountnumber: 'A1001',  
      type: 'Saving',  
      balance: 22000  
    },  
    {  
      accountnumber: 'A1002',  
      type: 'Checking',  
      balance: 1000  
    }  
  ];  
}
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
ngOnInit() {  
  //this.acctNo = this.route.snapshot.params['acctno'];  
  this.route.params.subscribe((params: Params)=> this.acctNo = params['acctno']);  
  this.account = this.accountDetails.find(e=>e.accountnumber === this.acctNo);  
}  
}
```

Note that you can also extract parameter from the route using the current snapshot of the route. But route.params observable is preferred.

```
this.acctNo = this.route.snapshot.params['acctno'];
```

### account.component.html

```
<h2>Account Details</h2>  
<div class="row">  
  <div class="col-xs-6">  
    <label>Account Number: </label> {{ account.accountnumber }}  
  </div>  
</div>  
<div class="row">  
  <div class="col-xs-6">  
    <label>Account Type: </label> {{ account.type }}  
  </div>  
</div>  
<div class="row">  
  <div class="col-xs-6">  
    <label>Balance: </label> {{account.balance}}  
  </div>  
</div>
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT. (AFFILIATED TO SAURASHTRA UNIVERSITY)



## Passing query parameters and fragments

### Via: HTML file

to pass query params, we will add [queryParams] attribute it takes an object. As

Shown in the Picture, and for the fragment, we take fragment property and write our fragment.

```
dynamic-component.component.html X
1 <!-- <a (click)="randomPassUrl()">Random dynamic url</a>
2 <hr />
3 {{ id }}
4 <hr />
5 directly using
6 {{ route.snapshot.params.id }} -->
7
8 <hr />
9 <div
10   class="box"
11   [routerLink]="['/dynamic', id]"
12   [queryParams]="{ allowEdit: '1' }"
13   fragment="leadme">
14   Params by tag
15 </div>
16 <div class="box" (click)="passQuery()">pass dynamically function</div>
```

dynamic-component.component.ts

For programmatically navigate,

## SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT. (AFFILIATED TO SAURASHTRA UNIVERSITY)

we use the router from @angular/router and inject it to the constructor same.

```
dynamic-componnet.component.ts X
1  import { Component, OnDestroy, OnInit } from '@angular/core';
2  import { Router, ActivatedRoute } from '@angular/router';
3  @Component({
4    selector: 'app-dynamic-componnet',
5    templateUrl: './dynamic-componnet.component.html',
6    styleUrls: ['./dynamic-componnet.component.css'],
7  })
8  export class DynamicComponnetComponent implements OnInit, OnDestroy {
9    constructor(public route: ActivatedRoute, private r: Router) { }
10   id: any;
11   observeParam: any;
12   ngOnInit() {
13     this.id = this.route.snapshot.params.id;
14     this.observeParam = this.route.params.subscribe((data) => {
15       this.id = data.id;
16     });
17   }
18   passQuery() {
19     this.r.navigate(['/dynamic', Math.random() * 1000], {
20       queryParams: { allowedit: Math.random() * 10 },
21       fragment: 'loading'
22     });
23     console.log(this.observeParam)
24   }
25 }
```

dynamic-component.component.ts

We use navigate function to pass our path, query params, and fragment.

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT. (AFFILIATED TO SAURASHTRA UNIVERSITY)

## Made it Simple:

The router navigate function takes two arguments.

- The first argument takes an array and we specify our paths in the array
- The second one is Object, **In key queryParams: we pass our query params in the object as in key and value.**
- In fragment, we specify our fragment.

## OUTPUT

on clicking of params by tag it passes through the tag, and on clicking on pass dynamically

function. It will the value dynamically of query params and fragments.



# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)



### Nested Routes

In this video I'm using an online editor called Plunker to write and run Angular code. The book and code has since been updated to use StackBlitz instead. To understand more about why and the differences between read this.

- **PARAMETERISED ROUTES**
- **ROUTER GUARDS**

1. Learning Objectives
2. Goal
3. Setup
4. Route Configuration
5. Relative Routes
6. Child Routes
7. Parent Route Parameters
8. Summary
9. Listing



### Learning Objectives

---

- How to configure child routes.
- How to define paths relative to the current path or the root of the application.
- How to get access to parent parameterised route params.

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

### ❖ Goal

---

The goal for this lecture is to change our iTunes search app so that when we click on a search result we are taken to an *Artist* page.

This *Artist* page has two more menu items, *Tracks* and *Albums*. If we click on *Tracks* we want to show the list of tracks for this artist, if we click on *Albums* we want to show the list of albums for this artist.

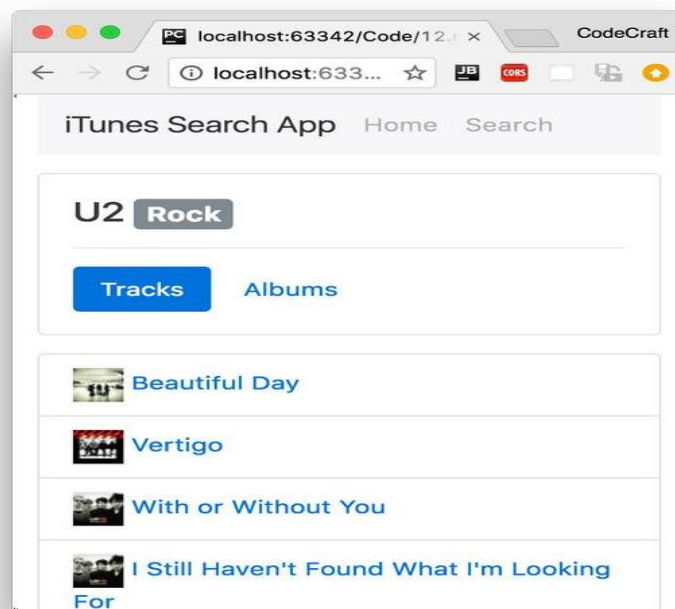


Figure 1. Artist Track Listing Page

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT. (AFFILIATED TO SAURASHTRA UNIVERSITY)

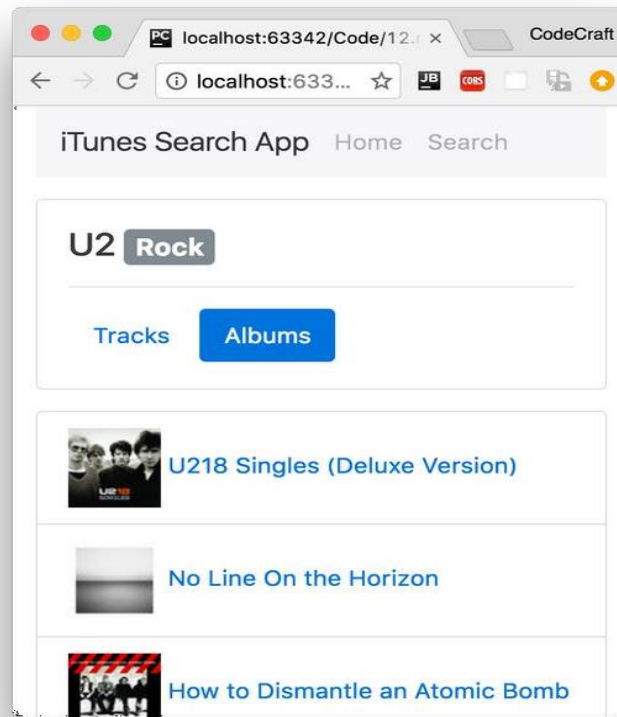


Figure 2. Artist Album Listing Page

We can get information about an Artist using the iTunes API by passing in the id of an artist

<https://itunes.apple.com/lookup?id=16586443>

A list of tracks for that artist by additionally passing in **entity=song**:

<https://itunes.apple.com/lookup?id=16586443&entity=song>

A list of albums for that artist by additionally passing in **entity=album**:

<https://itunes.apple.com/lookup?id=16586443&entity=album>

The concept of having *two* sets of menu items. A top level between *Home*, *Search* and *Artist* and a second level under *Artist* between *Tracks* and *Albums* is called *Nested Routing* and it's the topic of this lecture.



# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

### Setup

We've added 3 more components to our sample project and included them in our `NgModule` declarations:

1. `ArtistComponent` — This shows some details about an *Artist* and contains either the `ArtistTrackListComponent` or `ArtistAlbumListComponent`.
2. `ArtistTrackListComponent` — This will show a track listing for the current *Artist*.
3. `ArtistAlbumListComponent` — This will show an Album listing for the current *Artist*.

For now each component's template just has a heading with the name of the component, like so:

### TypeScript

```
Copy@Component({
  selector: 'app-artist',
  template: `
    <h1>Artist</h1>
  `,
})
class ArtistComponent {
}
```

### TypeScript

```
Copy@Component({
  selector: 'app-artist-track-list',
  template: `
    <h1>Artist Track Listing</h1>
  `,
})
class ArtistTrackListComponent {
}
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT. (AFFILIATED TO SAURASHTRA UNIVERSITY)

## TypeScript

```
Copy@Component({
  selector: 'app-artist-album-list',
  template: `
<h1>Artist Album Listing</h1>
`
})
class ArtistAlbumListComponent {
}
```

## ❖ Route Configuration

We need another route for our **ArtistComponent**, we want to pass to this component an **artistId** so it needs to be a *parameterised route*, the **artistId** is mandatory so we are *not* using optional params.

## TypeScript

```
Copyconst routes: Routes = [
  {path: '', redirectTo: 'home', pathMatch: 'full'},
  {path: 'find', redirectTo: 'search'},
  {path: 'home', component: HomeComponent},
  {path: 'search', component: SearchComponent},
  {path: 'artist/:artistId', component: ArtistComponent}, (1)
  {path: '**', component: HomeComponent}
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT. (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
];
```

Add a path of `'artist/:artistId'` for `ArtistComponent`

In our search results listing we add a `routerLink` to the result item so it navigates to our `Artist` page.

## HTML

Copy `<div class="list-group">`

```
<a [routerLink]="['artist', track.artistId]" (1)
  class="list-group-item list-group-item-action"
  *ngFor="let track of itunes.results">
  
  {{ track.name }} <span class="text-muted">by</span> {{ track.artist }}
</a>
</div>
```

Add `routerLink` directive.



## Relative Routes

---

## SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT. (AFFILIATED TO SAURASHTRA UNIVERSITY)

If we ran the above code you'll notice that searching for U2 transforms the URL to:

```
Copy/#/search;term=U2
```

and then if we click on a *U2* song the URL *incorrectly* becomes

```
Copy/#/search;term=U2/artist/78500
```

And we don't get shown the Artist page.

The URL looks incorrect, it's a concatenation of the search URL with the artist URL. We should be expecting a URL like so:

```
Copy/#/artist/78500
```

That's because when we navigated to

```
Copy[routerLink]="['artist', item.artistId]"
```

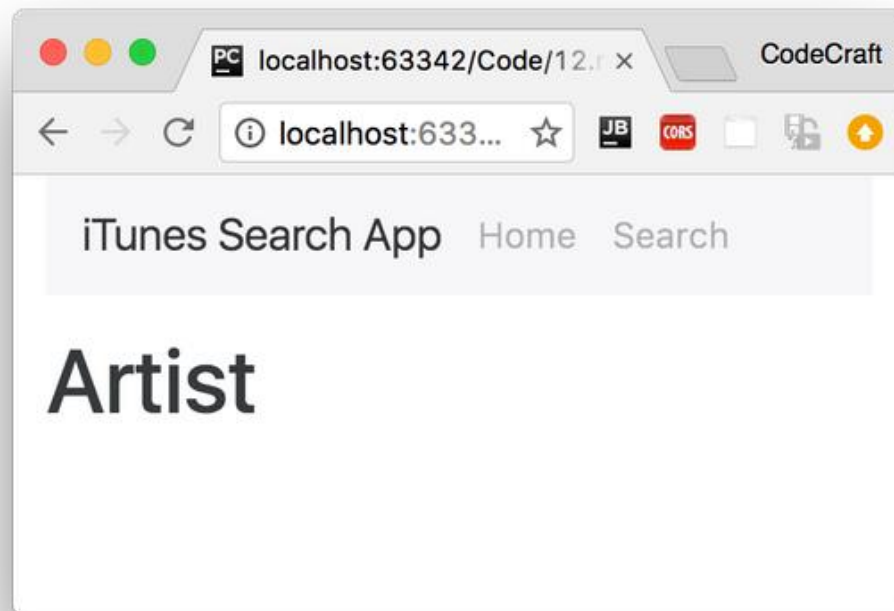
we did so *relative* to the current URL and the current URL at the time was the search URL.

What we want to do is to navigate relative to the `_root_` of our application, relative to `/` — we can do that simply by pre-pending `/` to our path, like so:

```
Copy[routerLink]="['/artist', item.artistId]"
```

## SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT. (AFFILIATED TO SAURASHTRA UNIVERSITY)

Now when we navigate we do so *relative* to the *root* URL and when we click the first U2 song the URL changes to `/#/artist/78500` and we are shown the artist page.



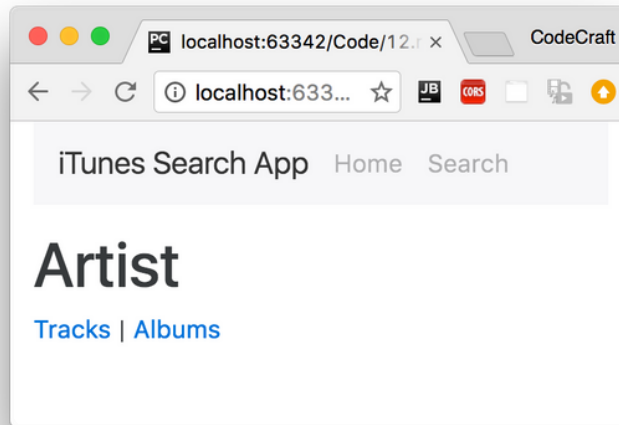
### ❖ Child Routes

---

When the Artist page is shown we want there to be two menu options, one called *Tracks* and the other called *Albums*, like so:

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)



When a user clicks on *Tracks* underneath I would like to show a list of all the tracks this artist has released, if the user clicks on *Albums* then I want to show a list of all the albums they have released.

Explained in terms that the *Router* would understand:

*“Inside the **ArtistComponent** I want to **conditionally** show either the **AristtAlbumsListComponent** or the **ArtistTrackListComponent** depending on which menu item the user has selected”.*

Let me first define the route configuration, each route can have a property called **children** where you can define the child routes of this route. I’m going to add my routes for **AristtAlbumsListComponent** and **ArtistTrackListComponent**, like so:

## TypeScript

```
Copyconst routes: Routes = [  
  {path: '', redirectTo: 'home', pathMatch: 'full'},  
  {path: 'find', redirectTo: 'search'},  
  {path: 'home', component: HomeComponent},  
  {path: 'search', component: SearchComponent},
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
{
  path: 'artist/:artistId',
  component: ArtistComponent,
  children: [
    {path: '', redirectTo: 'tracks'}, (1)
    {path: 'tracks', component: ArtistTrackListComponent}, (2)
    {path: 'albums', component: ArtistAlbumListComponent}, (3)
  ]
},
{path: '**', component: HomeComponent}
];
```

If a user navigates to say `/artist/1234` it will redirect to `/artist/1234/tracks` (since we would *at least* want one of either the track or album components to be shown).

This route matches a URL like `/artist/1234/tracks`.

This route matches a URL like `/artist/1234/albums`.

Now I can add the menu items to my artist template, like so:

## HTML

Copy `<h1>Artist</h1>`

```
<p>
  <a [routerLink]="['tracks']">Tracks</a> |
  <a [routerLink]="['albums']">Albums</a>
</p>
```

### Important

Notice that the paths in the `routerLink` directives above *don't* start with `/` so they are *relative* to the *current URL* which is the `/artist/:artistId` route.

A more explicit way of saying you want to route relative to the current URL is to prepend it with `./` like so:

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

### HTML

```
Copy<h1>Artist</h1>

<p>

  <a [routerLink]="['./tracks']">Tracks</a> |

  <a [routerLink]="['./albums']">Albums</a>

</p>
```

#### Note

Pre-pending with `./` clearly expresses our intent that the path is relative so let's use this syntax instead.

The final thing we need to do is to tell Angular where we want the `ArtistTrackListComponent` and `ArtistAlbumListComponent` components to be injected into the page, we do that by adding in another `router-outlet` directive in our Artist template like so:

### HTML

```
Copy<h1>Artist</h1>

<p>

  <a [routerLink]="['./tracks']">Tracks</a> |

  <a [routerLink]="['./albums']">Albums</a>

</p>

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

Now we have two `router-outlets` one nested inside another Angular figures out *which* outlet to insert the component in by the *nesting level* of the route and the router outlet.

## ❖ Parent Route Parameters



# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

To query the list of tracks for an Artist from the iTunes API the `ArtistTrackListComponent` needs the `artistId`.

As a reminder the route configuration for our `ArtistTrackListComponent` is

### JSON

```
Copy {path: 'tracks', component: ArtistTrackListComponent}
```

and the route configuration for its parent is

### JSON

```
Copy {  
  path: 'artist/:artistId',  
  component: ArtistComponent,  
  children: [  
    {path: '', redirectTo: 'tracks'},  
    {path: 'tracks', component: ArtistTrackListComponent},  
    {path: 'albums', component: ArtistAlbumListComponent},  
  ]  
}
```

The *parent route* has the `:artistId` as a route param, however if we injected `ActivatedRoute` into our child `ArtistTrackListComponent` and tried to print out the params surprisingly we just get an empty object printed out.

### TypeScript

```
Copy class ArtistTrackListComponent {  
  constructor(private route: ActivatedRoute) {  
    this.route.params.subscribe(params => console.log(params)); // Object {}  
  }  
}
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
}
```

The reason for this is that `ActivatedRoute` only passes you the parameters for the *current* component's route and since the route for `ArtistTrackListComponent` doesn't have any route parameters it gets passed nothing, we want to get the params for the *parent* route.

We can do this by calling `parent` on our `ActivatedRoute` like so:

### TypeScript

```
Copyclass ArtistTrackListComponent {  
  constructor(private route: ActivatedRoute) {  
    this.route.parent.params.subscribe(params => console.log(params)); // Object {artistId: 12345}  
  }  
}
```

This returns the params for the parent route.



### Summary

---

We can nest routes, this means that for a given URL we can render a tree of components.

We do this by using multiple `router-outlet` directives and configuring child routes on our route configuration object.

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

Next up we will learn about protecting access to different routes via the use of *Router Guards*.

### *Note*

The application above is complete in terms of child routing but still needs rounding out in terms of making the other API requests for tracks and albums and prettying up the template HTML.

You won't learn anything else about routing from going through the process of finishing off the app, but for your interest i've provided the full listing below.

As with all our examples they are for illustrative purposes only, to follow the official style guides we should be putting each component in a separate file and our HTTP request should all be wrapped in a separate service.

I will however leave that as an exercise for the reader if they so wish.

## ❖ Listing

### *Listing 1. main.ts*

#### TypeScript

```
Copyimport { NgModule, Component, Injectable } from "@angular/core";
import { BrowserModule } from "@angular/platform-browser";
import { platformBrowserDynamic } from "@angular/platform-browser-dynamic";
import { ReactiveFormsModule, FormControl, FormsModule } from "@angular/forms";
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
import {
  HttpClientJsonpModule,
  HttpClientModule,
  HttpClient
} from "@angular/common/http";
import { Routes, RouterModule, Router, ActivatedRoute } from "@angular/router";

class SearchItem {
  constructor(
    public name: string,
    public artist: string,
    public link: string,
    public thumbnail: string,
    public artistId: string
  ) {}
}

@Injectable()
class SearchService {
  apiRoot: string = "https://itunes.apple.com/search";
  results: SearchItem[];

  constructor(private http: HttpClient) {
    this.results = [];
  }

  search(term: string) {
    return new Promise((resolve, reject) => {
      this.results = [];
      let apiURL = `${this.apiRoot}?term=${term}&media=music&limit=20`;
      this.http
        .jsonp(apiURL, "callback")
        .toPromise()
        .then(
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
res => {  
  // Success  
  this.results = res.results.map(item => {  
    return new SearchItem(  
      item.trackName,  
      item.artistName,  
      item.trackViewUrl,  
      item.artworkUrl30,  
      item.artistId  
    );  
  });  
  resolve();  
},  
msg => {  
  // Error  
  reject(msg);  
}  
);  
});  
}  
}
```

```
@Component({  
  selector: "app-search",  
  template: `<form class="form-inline">  
    <div class="form-group">  
      <input type="search"  
        class="form-control"  
        placeholder="Enter search string"  
        #search>  
    </div>  
    <button type="button"  
      class="btn btn-primary"  
      (click)="onSearch(search.value)">
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
Search
</button>
</form>

<hr />

<div class="text-center">
  <p class="lead"
    *ngIf="loading">Loading...</p>
</div>

<div class="list-group">
  <a [routerLink]="['/artist', track.artistId]"
    class="list-group-item list-group-item-action"
    *ngFor="let track of itunes.results">
    
    {{ track.name }} <span class="text-muted">by</span> {{ track.artist }}
  </a>
</div>
,
})
class SearchComponent {
  private loading: boolean = false;

  constructor(
    private itunes: SearchService,
    private route: ActivatedRoute,
    private router: Router
  ) {
    this.route.params.subscribe(params => {
      console.log(params);
      if (params["term"]) {
        this.doSearch(params["term"]);
      }
    })
  }
}
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
});  
}  
  
doSearch(term: string) {  
  this.loading = true;  
  this.itunes.search(term).then(_ => (this.loading = false));  
}  
  
onSearch(term: string) {  
  this.router.navigate(["search", { term: term }]);  
}  
}  
  
@Component({  
  selector: "app-home",  
  template: `  
<div class="jumbotron">  
  <h1 class="display-3">iTunes Search App</h1>  
</div>  
  `,  
})  
class HomeComponent {}  
  
@Component({  
  selector: "app-header",  
  template: `  <a class="navbar-brand"  
    [routerLink]="['home']">iTunes Search App  
  </a>  
  <ul class="nav navbar-nav">  
    <li class="nav-item"  
      [routerLinkActive]="['active']">  
      <a class="nav-link"  
        [routerLink]="['home']">Home
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
</a>
</li>
<li class="nav-item"
  [routerLinkActive]="['active']">
  <a class="nav-link"
    [routerLink]="['search']">Search
  </a>
</li>
</ul>
</nav>
`
})
class HeaderComponent {
  constructor(private router: Router) {}

  goHome() {
    this.router.navigate([""]);
  }

  goSearch() {
    this.router.navigate(["search"]);
  }
}

@Component({
  selector: "app-artist-track-list",
  template: `
<ul class="list-group">
  <li class="list-group-item"
    *ngFor="let track of tracks">
    
    <a target="_blank"
      href="{{track.trackViewUrl}}">{{ track.trackName }}
    </a>
  </li>
</ul>
  `
})
```



# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
        </li>
    </ul>
    `
  })
  class ArtistTrackListComponent {
    private tracks: any[];

    constructor(private http: HttpClient, private route: ActivatedRoute) {
      this.route.parent.params.subscribe(params => {
        this.http
          .jsonp(
            `https://itunes.apple.com/lookup?id=${
              params["artistId"]
            }&entity=song`,
            "callback"
          )
          .toPromise()
          .then(res => {
            console.log(res);
            this.tracks = res.results.slice(1);
          });
      });
    }
  }

  @Component({
    selector: "app-artist-album-list",
    template: `<ul class="list-group">
      <li class="list-group-item"
        *ngFor="let album of albums">
        
        <a target="_blank"
          href="{{ album.collectionViewUrl }}">{{ album.collectionName }}
        </a>
      </li>
    </ul>`
  })
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
        </li>
    </ul>
    `
  })
  class ArtistAlbumListComponent {
    private albums: any[];

    constructor(private http: HttpClient, private route: ActivatedRoute) {
      this.route.parent.params.subscribe(params => {
        this.http
          .jsonp(
            `https://itunes.apple.com/lookup?id=${
              params["artistId"]
            }&entity=album`,
            "callback"
          )
          .toPromise()
          .then(res => {
            console.log(res);
            this.albums = res.results.slice(1);
          });
      });
    }
  }

  @Component({
    selector: "app-artist",
    template: `<div class="card">
      <div class="card-block">
        <h4>{{ artist?.artistName }} <span class="tag tag-default">{{ artist?.primaryGenreName }}</span></h4>
        <hr />
        <footer>
          <ul class="nav nav-pills">
            <li class="nav-item">
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
<a class="nav-link"
  [routerLinkActive]="['active']"
  [routerLink]="['./tracks']">Tracks
</a>
</li>
<li class="nav-item">
  <a class="nav-link"
    [routerLinkActive]="['active']"
    [routerLink]="['./albums']">Albums
  </a>
</li>
</ul>
</footer>
</div>
</div>

<div class="m-t-1">
  <router-outlet></router-outlet>
</div>
`
  })
class ArtistComponent {
  private artist: any;

  constructor(private http: HttpClient, private route: ActivatedRoute) {
    this.route.params.subscribe(params => {
      this.http
        .jsonp(
          `https://itunes.apple.com/lookup?id=${params["artistId"]}`,
          "callback"
        )
        .toPromise()
        .then(res => {
          console.log(res);
        })
    })
  }
}
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
this.artist = res.results[0];
console.log(this.artist);
});
});
}
}

@Component({
  selector: "app",
  template: `
    <app-header></app-header>
    <div class="m-t-1">
      <router-outlet></router-outlet>
    </div>
  `
})
class AppComponent {}

const routes: Routes = [
  { path: "", redirectTo: "home", pathMatch: "full" },
  { path: "find", redirectTo: "search" },
  { path: "home", component: HomeComponent },
  { path: "search", component: SearchComponent },
  {
    path: "artist/:artistId",
    component: ArtistComponent,
    children: [
      { path: "", redirectTo: "tracks", pathMatch: "full" },
      { path: "tracks", component: ArtistTrackListComponent },
      { path: "albums", component: ArtistAlbumListComponent }
    ]
  },
  { path: "**", component: HomeComponent }
];
```

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
@NgModule({
  imports: [
    BrowserModule,
    ReactiveFormsModule,
    FormsModule,
    HttpClientModule,
    HttpClientModule,
    RouterModule.forRoot(routes, { useHash: true })
  ],
  declarations: [
    AppComponent,
    SearchComponent,
    HomeComponent,
    HeaderComponent,
    ArtistAlbumListComponent,
    ArtistTrackListComponent,
    ArtistComponent
  ],
  bootstrap: [AppComponent],
  providers: [SearchService]
})
class AppModule {}

platformBrowserDynamic().bootstrapModule(AppModule);
```

## Angular Basics: Router Links and Wildcard Routing in Angular

### ❖ What Is Routing?

In the last article, we looked at how Angular makes it easy to bring in the navigation to your application through the router module. We also saw how routing can be set up in Angular with ease. Now that we have set up routing, there are a few more things we can do with the routing module.

### ❖ What We Are Building

Today we are building a simple navbar component with navigation in a single-page application (SPA) and a wildcard page to guide users every time they enter a wrong URL. We are going to continue from the last post, so download the source file [from here](#) into your machine.

### ❖ Setting Up

Open the new file in VS Code and inside the terminal run the command below:

### ❖ `npm install`

## SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT. (AFFILIATED TO SAURASHTRA UNIVERSITY)

This ensures all the node modules and dependencies needed for the project are properly installed. Your folder should have an app component with two child components: about and contact. The app component.html file should look like this:

```
<div class="container">
<ul class="nav justify-content-center">
  <li class="nav-item">
    <a class="nav-link active" aria-current="page" href="/">Home</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="/about">About</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="/contact">Contact</a>
  </li>
</ul>
<router-outlet></router-outlet>
</div>
```

### HTML

You can save your work and run the dev server to see that it all works well in the browser at localhost:4200.

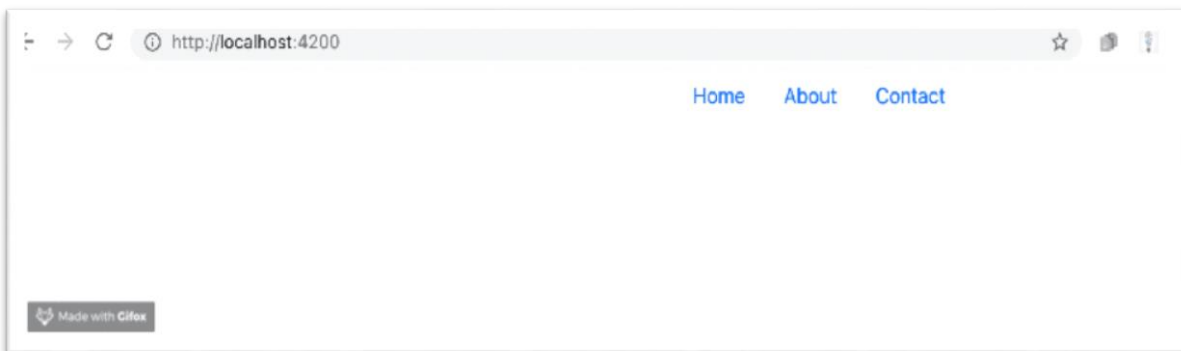
ng serve

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

### Single-Page Applications (SPAs)

A single-page web application is one that does not have to reload as you interact with it and navigate from one view to another—rather it dynamically displays views as requested. The first thing you notice with the application we have now is that every new click reloads the entire application, so it is not an SPA.



Router links are element properties provided by the router module that makes a link initiate navigation. You can liken it to href for anchor tags, so in our case, we replace the href tags with router links.

```
<div class="container">
<ul class="nav justify-content-center">
  <li class="nav-item">
    <a class="nav-link active" aria-current="page" href="/">Home</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" routerLink="/about">About</a>
  </li>
  <li class="nav-item">
```

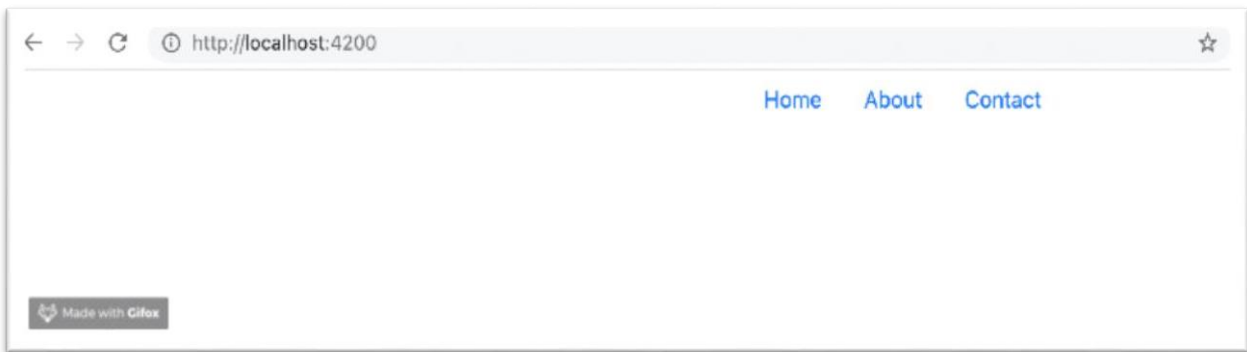


# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
<a class="nav-link" routerLink="/contact">Contact</a>
</li>
</ul>
<router-outlet></router-outlet>
</div>
```

HTML



### ❖ Wildcards

Now that we have set up routes, what happens if a user types in the wrong URL? A good application should always gracefully handle scenarios like this, where a user enters a URL that does not exist. Angular helps you do this with ease. You can set up a wildcard route to that effect.

Once you set up a wildcard route, it just tells the router to select this route anytime a requested URL does not match the specified routes in the app module. The syntax for defining a wildcard route looks like this:

```
{ path: '**', component: WildcardComponent }
```

TypeScript

Let us see it in action with our application. Generate a new component, and call it Page404.

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

### ng generate component page404

Now that you have created your wild card component, let us make it really clear. Open the page404 component.html file and change the paragraph content inside the html file from “page404 works” to “This URL you just entered is incorrect, kindly try again.”

```
<p>This URL you just entered is incorrect, kindly try again.</p>
```

HTML

Open the app module file and copy the code block below inside it:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { AppComponent } from './app.component';
import { AboutComponent } from './about/about.component';
import { ContactComponent } from './contact/contact.component';
import { Page404Component } from './page404/page404.component';
const routes: Routes = [
  { path: 'about', component: AboutComponent },
  { path: 'contact', component: ContactComponent },
  { path: '**', component: Page404Component }
];
@NgModule({
  declarations: [
    AppComponent,
    AboutComponent,
    ContactComponent,
    Page404Component
```

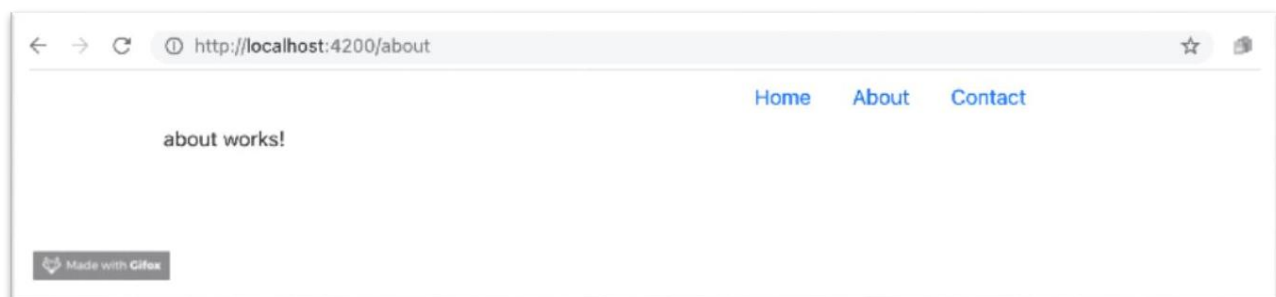
# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

```
],  
imports: [  
  BrowserModule,  
  RouterModule.forRoot(routes)  
],  
providers: [],  
bootstrap: [AppComponent]  
}))  
export class AppModule { }
```

TypeScript

Now if you try putting in any wrong URL, this is what you get:



# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

### □ Redirecting the Routes

Besides creating a separate wildcard component like we have done, you can just redirect the route to go to another route like the Homepage or the Contact page whenever a user enters an incorrect URL.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { AppComponent } from './app.component';
import { AboutComponent } from './about/about.component';
import { ContactComponent } from './contact/contact.component';
import { Page404Component } from './page404/page404.component';
const routes: Routes = [
  { path: 'about', component: AboutComponent },
  { path: 'contact', component: ContactComponent },
  { path: '**', redirectTo: '/contact' }
];
@NgModule({
  declarations: [
    AppComponent,
    AboutComponent,
    ContactComponent,
    Page404Component
  ],
  imports: [
    BrowserModule,
    RouterModule.forRoot(routes)
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

TypeScript

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

Now when you save the files, you will see that every incorrect URL redirects to the Contact page

### **wrap up**

- **USER INTERACTION & OUTPUTS**
- **ACTIVITY**

In this quick start section you created your first Angular Application!

You should now have at least a top-level view of what an Angular application is, how it functions and how to go about architecting it.

We covered:

### **Environment**

How to use [stackblitz.com](https://stackblitz.com) to write Angular apps in a browser

### **Components**

Which let us extend the HTML language to create new tags and forms the basis of an Angular application.

### **Bootstrapping**

How to actually make Angular load and take control of a HTML page.

### **Binding**

String interpolation with `{{ }}` and both input property binding as well as output event binding.

### **NgFor**

# SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

## (AFFILIATED TO SAURASHTRA UNIVERSITY)

How to render multiple elements using the built-in **NgFor** directive.

### Data Modelling

We touched on **data modelling**, by creating our own Joke class which encapsulated all our data and logic relate to a joke.

### Template Local Variables

Capturing user input from users by adding **#** variables to input controls.

### Architecture

We started to see how we go about building Angular apps by wiring together *inputs* and *outputs* in order to glue Components together.

We'll be going into *much* more detail into each of these topics, and others, in future chapters.

However since Angular is based on TypeScript it makes sense to get a good understanding of those features before we dive into the rest of this course so that's the topic of the next section.