Angular
Lab 9
Routing
Exercises

1 Lab setup

2 Generating a new Angular project

3 Registering routes to use with RouterOutlet

4 Using RouterLink and RouterLink Active to connect to and identify active routes

Q1. Create a new component called Articles and provide some suitable HTML to identify its template

```
ng generate component articles
```

articles.component.html

```
<section>
    <h2>Articles</h2>
    <p>Lots of interesting articles to read on this web site ....</p>
</section>
```

Q2. Provide a link to navigate to this page from the root template and also configure this route appropriately.

app.component.html

```
<ul>
  <li><a href="#" routerLink="./home"
routerLinkActive="activelink">Home</a></li>
  <li><a href="#" routerLink="./contact"
routerLinkActive="activelink">Contact</a></li>
  <li><a href="#" routerLink="./about"
routerLinkActive="activelink">About</a></li>
  <li><a href="#" routerLink="./articles"
routerLinkActive="activelink">Articles</a></li>

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

app.routes.ts

```
import { Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';
import { ContactComponent } from './contact/contact.component';
import { ArticlesComponent } from './articles/articles.component';

export const routes: Routes = [
    {path: 'home', component: HomeComponent},
    {path: 'about', component: AboutComponent},
    {path: 'contact', component: ContactComponent},
    {path:  'articles', component: ArticlesComponent}
  ];
```

5 Route redirecting and wildcard routes

6 Passing route parameters through dynamic routing

Q1. Building on the ArticlesComponent that was created in the solution for Part 4, add in a String array called articleNames in HomeComponent containing a number of random strings, and then bind these strings as a path suffix using the routerLink directive to the end of the `/articles` path that matches to ArticlesComponent, in a similar way that was done for the HeroComponent. Provide a suitable route configuration for dynamic routes so that these strings becomes the route parameters when appended to the end of the `/articles` path.

`home.component.ts`

```
  // Q1
  articleNames = ['superman', 'spiderman','wonder woman','aquaman'];
```

`home.component.html`

```
<!--     Q1 -->
    <ul>
        @for (article of articleNames; track article) {

          <li><a [routerLink]="['/articles', article]">{{article}}</a></li>

        }
    </ul>
```

`app.routes.ts`

```
    // Q1
    { path: 'articles/:article', component: ArticlesComponent},
```

Q2. In ArticlesComponent, retrieve the single route parameter from the path URL using Approach # 2 (The Observables approach) and display in the template.

articles.component.ts

```
export class ArticlesComponent {


  // Q2
  article : string | null = '';

  //Injecting the ActivatedRoute service
  constructor (private route: ActivatedRoute) {  }



  ngOnInit() {

    const paramMap = this.route.paramMap;
    paramMap.subscribe(
      (params) => {
        this.article = params.get('article');
      }
    );

  }

}
```

articles.component.html

```
<section>
    <h2>Articles</h2>
    <p>Lots of interesting articles to read on this web site ....</p>
    <p> The article you chose is about : {{ article }}</p>
</section>
```

7 Passing query parameters

Q1. Building on the ArticlesComponent that was created in the solution for Part 4, add in an array of 3 objects called `articleObjects`, where each object contains a single property `articleName` with a random string value of your choosing (for e.g. `{ articleName : 'Nice article' }`). Bind each of the objects in this array using the *for directive to an <a> tag in the template of ArticlesComponent which links to the `/articles` path, and uses the specific object in the array as a query parameter. You can use the article name as the content of the <a> link itself.

Also ensure that you change the routing configuration so that the `/articles` path no longer needs to receive a route parameter.

`app.routes.ts`

```
    // Q1
    { path: 'articles', component: ArticlesComponent},
```

`contact.component.ts`

```
  // Q1

  articleObjects = [
    {articleName : 'Nice article'},
    {articleName : 'Boring article'},
    {articleName : 'so so article'},
  ]
```

`contact.component.html`

```
    <!-- Q1 -->
    <ul>
        @for (article of articleObjects; track article) {
            <li><a [routerLink]="['/articles']"
[queryParams]="article">{{article.articleName}}</a></li>

        }
    </ul>
```

Q2. In ArticlesComponent, retrieve the single query parameter from the path URL using Approach # 2 (The Observables approach) and display in the template.

articles.component.ts

```
export class ArticlesComponent {


  // Q2
  articleName : string | null = '';

  //Injecting the ActivatedRoute service
  constructor (private route: ActivatedRoute) {  }

  ngOnInit() {

    const queryParamMap = this.route.queryParamMap;
    queryParamMap.subscribe(
      (params) => {
        this.articleName = params.get('articleName') || 'noone';
      }
    );

  }
}
```

articles.component.html

```
<section>
    <h2>Articles</h2>
    <p>Lots of interesting articles to read on this web site ....</p>
    <p> The article you chose is : {{ articleName }}</p>
</section>
```

8 Nested / child routes

Q1. Generate 2 new child components `sadArticle` and `happyArticle`. Make them nested routes of the ArticlesComponent that was created in the solution for Part 4. Provide <a> links to them in the template of ArticlesComponent (there is no need to pass any route or query parameters to them). Ensure that your routing configuration reflects these child components so you can navigate effectively to them and provide a link to the main Articles page from the root template.

```
ng generate component sadArticle
```

```
ng generate component happyArticle
```

`articles.component.html`

```
<section>
    <h2>Articles</h2>
    <p>What kind of articles are you interested in ? </p>

    <ul>
        <li><a routerLink="happy">Happy articles</a></li>
        <li><a routerLink="sad">Sad articles</a></li>
    </ul>

    <router-outlet></router-outlet>

</section>
```

`articles.component.ts`

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

@Component({
  selector: 'app-articles',
  standalone: true,
  imports: [RouterOutlet, RouterLink],
  templateUrl: './articles.component.html',
  styleUrl: './articles.component.css'
})

export class ArticlesComponent {
  // rest of code here is irrelevant
```

app.routes.ts

```
    // Q1
    {path: 'articles', component: ArticlesComponent, children: [
        {path: 'happy', component: HappyArticleComponent},
        {path: 'sad', component: SadArticleComponent},
    ]},
```

app.component.html

```
<ul>
  <li><a href="#" routerLink="./home"
routerLinkActive="activelink">Home</a></li>
  <li><a href="#" routerLink="./contact"
routerLinkActive="activelink">Contact</a></li>
  <li><a href="#" routerLink="./news"
routerLinkActive="activelink">News</a></li>
  <li><a href="#" routerLink="./about"
routerLinkActive="activelink">About</a></li>

  <!-- Q1 -->
  <li><a href="#" routerLink="./articles"
routerLinkActive="activelink">Articles</a></li>
</ul>
```

9 Programmatic navigation and setting page title

Q1. Building on the ArticlesComponent that was created in the solution for Part 4, add a field to obtain input from the user for an article name (implement with NgModel two way binding in the HomeComponent template). Then provide a button, which when clicked on, navigates to the ArticlesComponent view, passing along the article name as a suffix to the `/articles` path portion. Ensure that you have a suitable route configuration for dynamic route for ArticlesComponent.

home.component.ts

```
  // Q1

  articleName = '';
```

```
  navigateToArticle() {
    this.router.navigateByUrl('articles/' + this.articleName);
  }
```

home.component.html

```
    <!-- Q1 -->
    <label for="articlebox">Article name : </label>
    <input type="text" id="articlebox"  [(ngModel)]="articleName">
    <br>
    <br>
    <button type="button"(click)="navigateToArticle()">Show me my article !
</button>
    <br>
    <br>
```

app.routes.ts

```
    // Q1
    { path: 'articles/:article', component: ArticlesComponent},
```

Q2. In ArticlesComponent, retrieve the single route parameter from the path URL using Approach # 2 (The Observables approach) and display in the template.

articles.component.ts

```
export class ArticlesComponent {


  // Q2
  article : string | null = '';

  //Injecting the ActivatedRoute service
  constructor (private route: ActivatedRoute) {  }
```

```
  ngOnInit() {

    const paramMap = this.route.paramMap;
    paramMap.subscribe(
      (params) => {
        this.article = params.get('article');
      }
    );

  }

}
```

articles.component.html

```
<section>
    <h2>Articles</h2>
    <p>Lots of interesting articles to read on this web site ....</p>
    <p> The article you chose is about : {{ article }}</p>
</section>
```