

Pipes in Angular



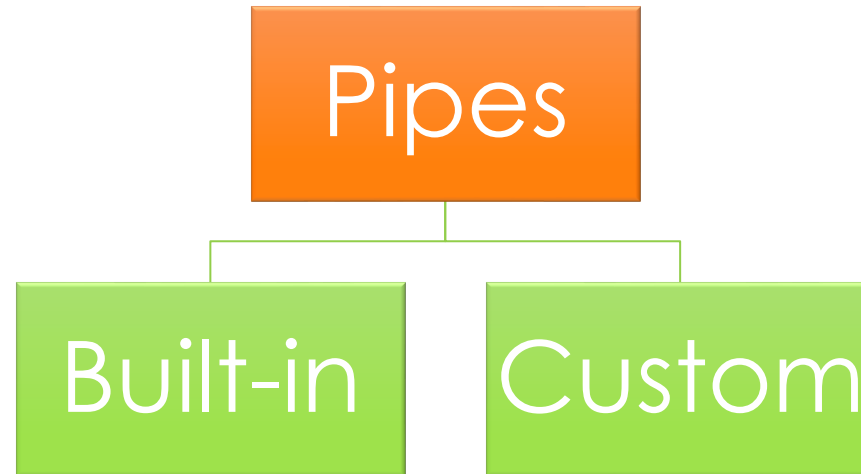
By Sonam Soni(B.E.I.T.)
JAVA FSD/MERN/MEAN (Expert & Instructor)



PIPES

- A. In Angular, **pipes** are a feature used to transform or format data displayed in the view.
- B. They are simple functions that accept input, process it, and return a transformed version of that input.
- C. Pipes are particularly useful for formatting strings, dates, numbers, and more in an Angular template.

TYPES OF PIPES



Angular comes with a set of built-in pipes, such as `DatePipe`, `CurrencyPipe`, `UpperCasePipe`, which cover common data transformation needs.

You Can Create Your Own Pipe to Handle Specific data, Like Highlighting AnyTag or Decorate Data Using Pipes

USE OF PIPES



Transforming Data



Formatting Numbers



Handling Async Data

WORKING OF INBUILT-PIPES

```
<p><b>UPPERCASE: </b>{{name | uppercase}}</p>
```

```
<p><b>pi: </b>{{pi | number:'5.5-5'}}</p>
```

```
<p><b>pi: </b>{{pi | number:'.10-10'}}</p>
```

```
<p><b>USD: </b>{{a | currency:'USD':true : '3.3-3'}}</p>
```

WORKING OF INBUILT-PIPES

```
<div style="background-color: lightblue;">
  <div class="container">
    <h2>1. Inbuilt Pipes</h2>
    <hr>
    <p><b>UPPERCASE: </b>{{name | uppercase}}</p>
    <hr>
    <p><b>UPPERCASE: </b>{{name | lowercase}}</p>
  </div>
</div>

<div style="background-color: lightgrey;">
  <div class="container">
    <h2>Number Pipes</h2>
    <hr>
    <p><b>pi: </b>{{pi | number:'5.5-5'}}</p>
    <hr>
    <p><b>pi: </b>{{pi | number:'3.3-3'}}</p>
    <hr>
    <p><b>pi: </b>{{pi | number:'3.5-5'}}</p>
    <hr>
    <p><b>pi: </b>{{pi | number:'.10-10'}}</p>
  </div>
</div>
```

```
<div style="background-color: lightseagreen;">
  <div class="container">
    <h2>Currency Pipes</h2>
    <hr>
    <p><b>USD: </b>{{a | currency:'USD':true :'3.3-3'}}</p>
    <hr>
    <p><b>USD: </b>{{a | currency:'USD':false :'3.3-3'}}</p>
    <hr>
    <p><b>EURO: </b>{{b | currency:'EURO':false}}</p>
    <hr>
    <p><b>EURO: </b>{{b | currency:'EURO':true}}</p>
    <hr>
    <p><b>RAND: </b>{{b | currency:'RAND':true}}</p>
    <hr>
    <p><b>INR: </b>{{a | currency:'INR':true:'4.4-4'}}</p>
  </div>
</div>
```

WORKING OF INBUILT-PIPES

```
<div style="background-color: ■yellow;">
  <div class="container">
    <h2>Date Pipes</h2>
    <hr>
    <p><b>Today is: </b>{{today}}</p>

    <hr>
    <p><b>Today Date is: </b>{{today | date}}</p>

    <hr>
    <p><b>Today Date in dd/MM/yyyy is: </b>{{today | date:'dd/MM/yyyy'}}</p>
  </div>
</div>

<div style="background-color: ■yellowgreen;">
  <div class="container">
    <h2>JSON Pipes</h2>
    <hr>
    <p><b>Without Pipes: </b>{{object}}</p>
    <hr>
    <p><b>With Pipes: </b>{{object | json}}</p>
  </div>
</div>
```


LAB-1 FORMAT THE GIVEN DATA WITH PIPES

```
name:string="My Name is Sonam And i am a JAVA FSD Trainer";  
pi:number=3.1415927;  
a=0.12345;  
b=9876.54321;  
today=new Date();  
object={name:"Sonam Soni",email:"sonam@gmail.com",address:"India"}  
  
//custom pipe  
  
custom:string="Welcome to the World of CustomPipes";
```

```
custom:string="Welcome to the World of CustomPipes";
```


WORKING OF CUSTOM-PIPES

1. Lets assume you want to split the string and rejoin with some special character, so we can create the custom pipes using which we can use it globally anywhere in any component
2. Create New File under app Folder > `CustomPipes.ts`

```
import { Pipe, PipeTransform } from "@angular/core";

@Pipe({name: 'changestring'})
export class CustomPipes implements PipeTransform{
    transform(value: string, character: string) {
        return value.split(character).join("-")
    }
}
```

WORKING OF CUSTOM-PIPES

Declare your pipes in `app.module.ts` file

```
@NgModule({  
  declarations: [  
    AppComponent,  
    PipesComponent,  
    CustomPipes,  
    HomeComponent,
```

Use like this in any of the component you like

```
<p><b>Custom String: </b>{{custom | changestring:' '}}</p>
```

LAB-2 FORMAT DATA WITH CUSTOM PIPES

```
<h2>2. Custom Pipe</h2>
<hr>
<p><b>Original String: </b>{{custom}}</p>

<hr>
<p><b>Custom String: </b>{{custom | changestring:' '}}</p>

<hr>
<p><b>Custom String: </b>{{custom | uppercase | changestring:' '}}</p>
```

```
<b><p>Custom String: </p>{{custom | uppercase | changestring:' '}}</b>
</b>
```

Services in Angular



By Sonam Soni(B.E.I.T.)
JAVA FSD/MERN/MEAN (Expert & Instructor)



SERVICES IN ANGULAR

- In Angular, services can be used as **singletons** to share data between components and maintain a shared state across the application.
- By default, services in Angular are singleton instances when provided at the root level (providedIn: 'root').
- This allows for effective **data sharing** between components.
- ng generate service data-sharing

CODE FOR SERVICES

```
import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class DataSharingService {
  // Use a BehaviorSubject to allow subscription to data changes
  private messageSource = new BehaviorSubject<string>('Initial Message');
  currentMessage = this.messageSource.asObservable();

  constructor() { }

  // Method to update the message
  changeMessage(message: string) {
    this.messageSource.next(message);
  }
}
```


COMPONENT COMMUNICATION

```
@Component({
  selector: 'app-first-component',
  templateUrl: './first-component.component.html',
})
export class FirstComponent {
  newMessage: string = "Hello from First Component";

  constructor(private dataSharingService: DataSharingService) {}

  sendMessage() {
    // Update the shared message
    this.dataSharingService.changeMessage(this.newMessage);
  }
}
```

READ DATA

```
@Component({
  selector: 'app-second-component',
  templateUrl: './second-component.component.html',
})
export class SecondComponent implements OnInit {
  message: string = '';

  constructor(private dataSharingService: DataSharingService) {}

  ngOnInit(): void {
    // Subscribe to the shared data
    this.dataSharingService.currentMessage.subscribe(message => {
      this.message = message;
    });
  }
}
```



SHARE DATA ACROSS COMPONENTS

OBSERVABLES

- In Angular, **Observables** from RxJS (Reactive Extensions for JavaScript) are widely used to handle asynchronous data streams such as HTTP requests, user input events, or real-time updates.
- **Operators** like map, subscribe, catchError, and retry allow us to transform, handle, and control these streams effectively.
- Let's generate one service to use this operators.
- Ng g s data --skip-tests

DATA SERVICE

```
@Injectable({
  providedIn: 'root',
})
export class DataService {
  private apiUrl = 'https://api.example.com/data';

  constructor(private http: HttpClient) {}

  getData(): Observable<any> {
    return this.http.get(this.apiUrl).pipe(
      retry(3), // Retry the request up to 3 times in case of failure
      map(response => {
        // Transform the response data
        return response['data'].map(item => ({ ...item, newField: 'newValue' }));
      }),
      catchError(error => {
        console.error('Request failed:', error);
        return of([]); // Return an empty array as a fallback in case of an error
      })
    );
  }
}
```

USE SERVICE IN COMP

```
@Component({
  selector: 'app-data-component',
  template: `
    <div *ngFor="let item of data">{{ item.name }}</div>
  `,
})
export class DataComponent implements OnInit {
  data: any[] = [];
  ⚡
  constructor(private dataService: DataService) {}

  ngOnInit(): void {
    this.dataService.getData().subscribe(
      data => (this.data = data), // Handle data
      error => console.error('Error:', error), // Handle error
      () => console.log('Completed') // Handle completion
    );
  }
}
```

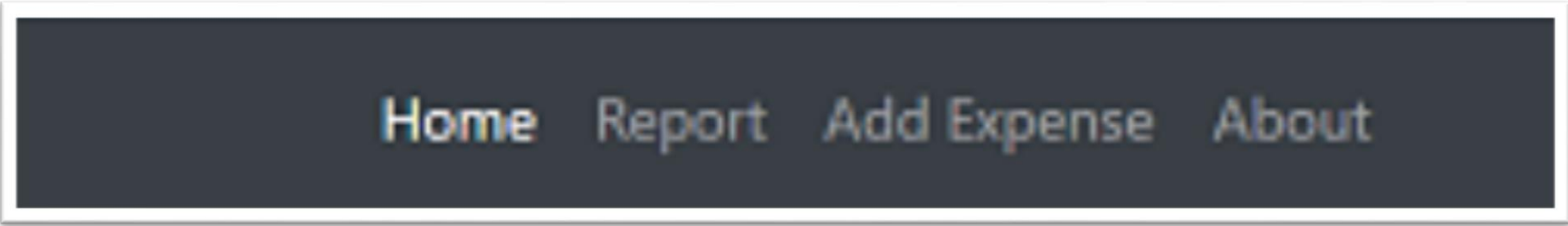

Routing in Angular



By Sonam Soni(B.E.I.T.)
JAVA FSD/MERN/MEAN (Expert & Instructor)

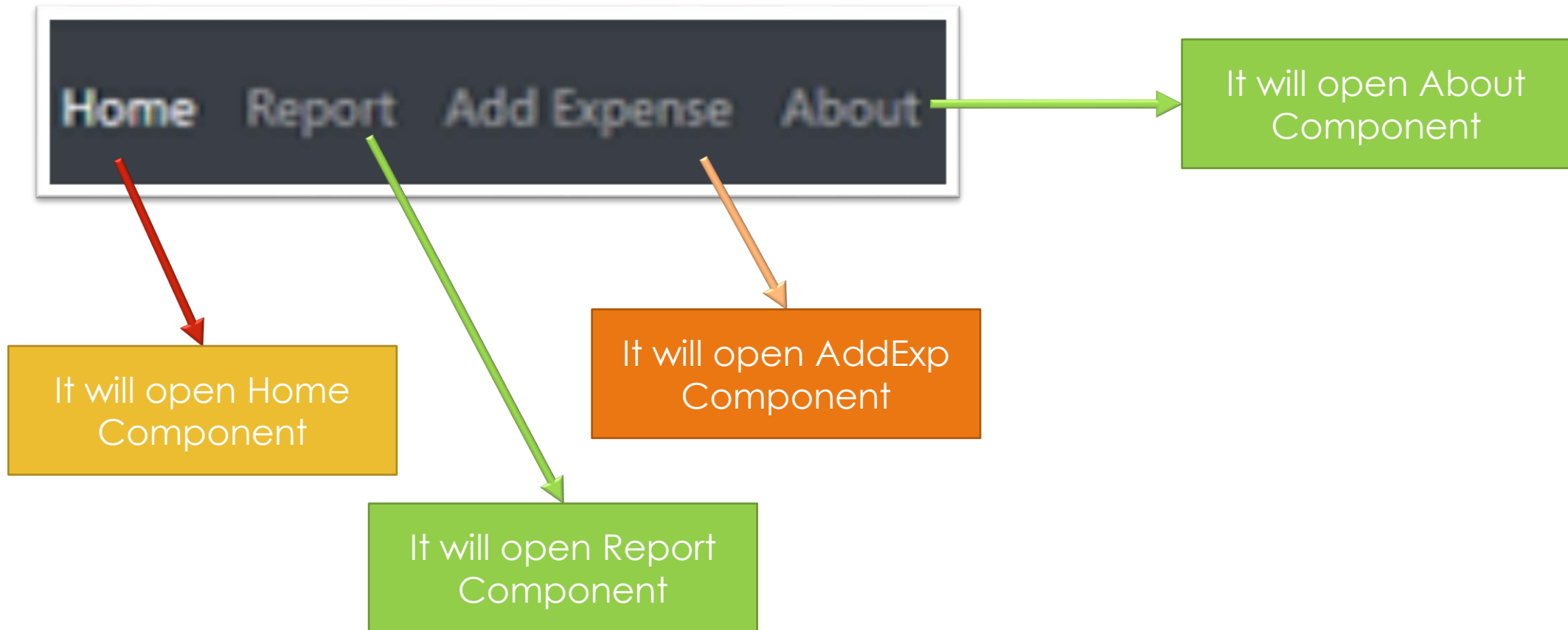
NEED OF ROUTING?

- Usually we have different pages in application.
- How do we have different pages in SPA?
- Here routing comes in a picture.



Home Report Add Expense About

HOW ANGULAR ROUTING WORKS?





HOW TO ADD ROUTING MODULE

- Import RouterModule.
- Ng new routing-app –routing
- This will add routing module by default your application

TO WORK WITH ROUTING

- Understand 3 things:
 1. Create routes in **app-routing.module.ts** file
 2. Create link using **routerLink** attribute not href
 3. To render the component must add **<router-outlet></router-outlet>**

LET'S UNDERSTAND ROUTING

```
const routes: Routes = [  
  {path: '',redirectTo: '/home',pathMatch: 'full'},  
  {path: 'home',component: HomeComponent},  
  {path: 'login',component: LoginComponent},  
  {path: 'register',component: RegisterComponent},  
  {path: 'lifecycle',component: LifecycleComponent},  
  {path: 'events',component: EventsComponent},  
  {path: 'loginForm',component: LoginformComponent},  
  {path: 'registerForm',component: RegisterformComponent},  
]
```



```
<div>
  <h1>App Component</h1>
  <nav>
    <a routerLink="home">Home</a> |
    <a routerLink="login">Login</a> |
    <a routerLink="register">Register</a> |
    <a routerLink="products">Products Page</a> |
    <a routerLink="dashboard">Dashboard</a> |
    <a routerLink="lifecycle">lifecycle</a> |
    <a routerLink="events">Events</a> |
    <a routerLink="loginForm">loginForm</a> |
    <a routerLink="registerForm">Registration</a> |
  </nav>
</div>
<app-lifecycle [data]="parenData"></app-lifecycle>
<router-outlet></router-outlet>
```

CHILD ROUTE

```
{path: 'dashboard', component: DashboardComponent, children: [  
  {path: '', redirectTo: 'child-a', pathMatch: 'full'},  
  {path: 'child-a', component: ChildAComponent},  
  {path: 'child-b', component: ChildBComponent}  
]},
```

USE CHILD ROUTE COMPONENT

```
<h3>Dashboard Component</h3>
<nav>
  <ul>
    <li>
      <a routerLink="child-a">Overview</a>
    </li>
    <li>
      <a routerLink="child-b">Technical Specification</a>
    </li>
  </ul>
</nav>
<div>
  <router-outlet></router-outlet>
</div>
```

DYNAMIC ROUTE

```
{path: 'products/details/:id', component: DetailsComponent},
```

```
export class DetailsComponent implements OnInit {  
  
  product: any = null;  
  constructor(private route: ActivatedRoute) { }  
  
  ngOnInit(): void {  
    const id = Number(this.route.snapshot.paramMap.get('id'));  
    console.log(id);  
    this.product = productsData.find(item => item.id === id);  
  }  
}
```

PROTECTED ROUTE

- Create one Service
- Auth.service.ts
- Create Guard:
- ng generate guard auth

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

@Injectable({
  providedIn: 'root'
})
export class AuthService {

  private loggedIn=false;
  isLoggedIn():boolean{
    return this.loggedIn;
  }
  login():void{
    this.loggedIn=true;
  }
  logout():void{
    this.loggedIn=false
  }
  constructor() { }
}
```

AUTH GUARD FILE

```
@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  //injected Service which we have created
  //injected router Dependency for programatic routing
  constructor(private authService:AuthService,private router:Router){}
  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> {

    if(this.authService.isLoggedIn()){
      return true;
    }else{
      this.router.navigate(['/login']);
      return false;
    }
  }
}
```


MAKE PROTECTED ROUTE

```
{path: 'products', component: ProductsComponent, canActivate: [AuthGuard]},  
  
{path: 'products/details/:id', component: DetailsComponent,  
  canActivate: [AuthGuard]  
},  
{path: 'dashboard', component: DashboardComponent, children: [  
  {path: '', redirectTo: 'child-a', pathMatch: 'full'},  
  {path: 'child-a', component: ChildAComponent},  
  {path: 'child-b', component: ChildBComponent}  
], canActivate: [AuthGuard]},
```

CANACTIVATECHILD

- The CanActivateChild guard is used to control access to child routes.

```
{path: 'dashboard',  
  component: DashboardComponent,  
  canActivate: [AuthGuard],  
  canActivateChild: [AuthGuard],  
  children: [  
    {path: '', redirectTo: 'child-a', pathMatch: 'full'},  
    {path: 'child-a', component: ChildAComponent},  
    {path: 'child-b', component: ChildBComponent}  
  ],  
}
```

IMPLEMENT LOGIC IN AUTH GUARD

```
export class AuthGuard implements CanActivate, CanActivateChild {  
  //injected Service which we have created  
  //injected router Dependency for programatic routing  
  constructor(private authService: AuthService, private router: Router) {}  
  
  canActivateChild(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):  
    | return this.canActivate(route, state); // Same logic for child routes  
  }
```



CANDEACTIVATE

- The CanDeactivate guard is used to prevent users from leaving a route
- for example, preventing them from losing unsaved changes.
- Create a new CanDeactivate guard.
- `ng generate guard can-deactivate`

```
import { Injectable } from '@angular/core';
import { CanDeactivate } from '@angular/router';
import { Observable } from 'rxjs';

export interface CanComponentDeactivate {
  canDeactivate: () => Observable<boolean> | Promise<boolean> | boolean;
}

@Injectable({
  providedIn: 'root',
})
export class CanDeactivateGuard implements CanDeactivate<CanComponentDeactivate> {
  canDeactivate(component: CanComponentDeactivate): Observable<boolean>
    | Promise<boolean> | boolean {
    return component.canDeactivate ? component.canDeactivate() : true;
  }
}
```

```
export class DashboardComponent implements CanComponentDeactivate {  
  unsavedChanges = true; // Let's assume there are unsaved changes  
  
  canDeactivate(): boolean {  
    if (this.unsavedChanges) {  
      return confirm('You have unsaved changes! Do you really want to leave?');  
    }  
    return true;  
  }  
  
  logout() {  
    this.unsavedChanges = false; // Reset unsaved changes on logout  
  }  
}
```


PROTECT ROUTE USING DEACTIVATE

```
{path: 'dashboard',  
  component: DashboardComponent,  
  canActivate: [AuthGuard],  
  canActivateChild: [AuthGuard],  
  canDeactivate: [CanDeactivateGuard],  
  // Protect the home route with CanDeactivate  
  children: [  
    {path: '', redirectTo: 'child-a', pathMatch: 'full'},  
    {path: 'child-a', component: ChildAComponent},  
    {path: 'child-b', component: ChildBComponent}  
  ],  
},
```

FEATURE MODULE

- **Feature Module** is a way to organize and encapsulate specific functionality in a separate module, keeping the codebase modular and easy to manage.
- This allows you to split large applications into smaller, more manageable pieces.
- ng generate module customer –routing
- Define components in Module
- ng generate component customer/customer-list
- ng generate component customer/customer-detail

FEATURE MODULE

- Create one module called: ng g m customer

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

const routes: Routes = [{ path: '', component: CustomerComponent }];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class CustomerRoutingModule { }
```

CUSTOMER.MODULE.TS

```
import { CustomerRoutingModule } from './customer-routing.module';
import { CustomerComponent } from './customer.component';
import { CustomerListComponent } from './customer-list/customer-list.component';

@NgModule({
  declarations: [
    CustomerComponent,
    CustomerListComponent
  ],
  imports: [
    CommonModule,
    CustomerRoutingModule
  ]
})
export class CustomerModule { }
```

- Add in app.module.ts in imports.
- Add routes in customer routing

```
const routes: Routes = [  
  { path: '', component: CustomerComponent },  
  { path: 'customers', component: CustomerListComponent },  
];  
  
@NgModule({  
  imports: [RouterModule.forChild(routes)],  
  exports: [RouterModule]  
})  
export class CustomerRoutingModule { }
```

LAZY LOADING

- Lazy loading of component in app.module.ts file
- To optimize performance, you can lazy load the feature module so it's only loaded when the user navigates to a route that uses the module.

```
{ path: 'customer',  
  loadChildren: () =>  
    import('./customer/customer.module').then(m => m.CustomerModule) }  
//Lazy loading
```




PROVIDEHTTPCLIENT

- In Angular 15, the way to configure the HttpClient has evolved with the introduction of the **Standalone APIs**.
- The new provideHttpClient method replaces the traditional method of importing HttpClientModule in the NgModule.
- This new API provides a more flexible and modern approach to integrating the HttpClient.
- The provideHttpClient function allows you to configure and register the HttpClient service in your Angular application.



IMPLEMENTATION

- Step 1: Install HttpClient Module
- `ng add @angular/common`
- Configure `provideHttpClient` in `main.ts`
- Instead of importing the `HttpClientModule` in an `NgModule`, you can directly provide the HTTP client service in your **`main.ts`** using `provideHttpClient`.

```
import { bootstrapApplication } from '@angular/platform-browser';
import { provideHttpClient } from '@angular/common/http';
import { AppComponent } from '../app/app.component';

bootstrapApplication(AppComponent, {
  providers: [provideHttpClient()] // Register HttpClient globally
})
.catch(err => console.error(err));
```