What is RxJs?

* RxJs (Reactive Extension for JavaScript) is a library for reactive programming using Observable, to Handle Asynchronous events
* **Angular RxJs everywhere:**
  + HTTP Request
  + Form Value Changes
  + Routing Events
  + Component Communication

SwitchMap

It will cancel previous requests and only keep the latest request (Ex. LIVE Search in an Application)

Create service:

* ng g s service/service

import { HttpClient } from '@angular/common/http';

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

@Injectable({

  providedIn: 'root'

})

export class UserService {

  constructor( private http:HttpClient) { }

  serachUsers(query: String){

    return this.http.get<any[]>(`https://jsonplaceholder.typicode.com/users?name\_like=${query}`)

  }

}

Inject the service in any component where we need to implement search functionality

Lets do it in app.component.html

<nav class="navbar navbar-expand-lg bg-body-tertiary">

  <div class="container-fluid">

    <a class="navbar-brand" href="#">Navbar</a>

    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

      <span class="navbar-toggler-icon"></span>

    </button>

    <div class="collapse navbar-collapse" id="navbarSupportedContent">

      <ul class="navbar-nav me-auto mb-2 mb-lg-0">

        <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="directives" href="#">Directives</a>

        </li>

      </ul>

      <form class="d-flex" role="search">

        <input [formControl]="searchControl" class="form-control me-2" type="search" placeholder="Search" aria-label="Search"/>

        <button class="btn btn-outline-success" type="submit">Search</button>

      </form>

    </div>

  </div>

</nav>

<div class="container">

  <li \*ngFor="let u of users">{{u.name}} </li>

</div>

App.component.ts file

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

import { UserService } from './service/user.service';

import { FormControl } from '@angular/forms';

import { debounce, debounceTime, switchMap } from 'rxjs';

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'angular-app';

  searchControl= new FormControl('');

  users:any[]=[];

  constructor(private service:UserService){

    this.searchControl.valueChanges.pipe(

      debounceTime(300),

      switchMap(value=> this.service.serachUsers(value ?? ''))

    ).subscribe(data=>this.users=data)

  }

}

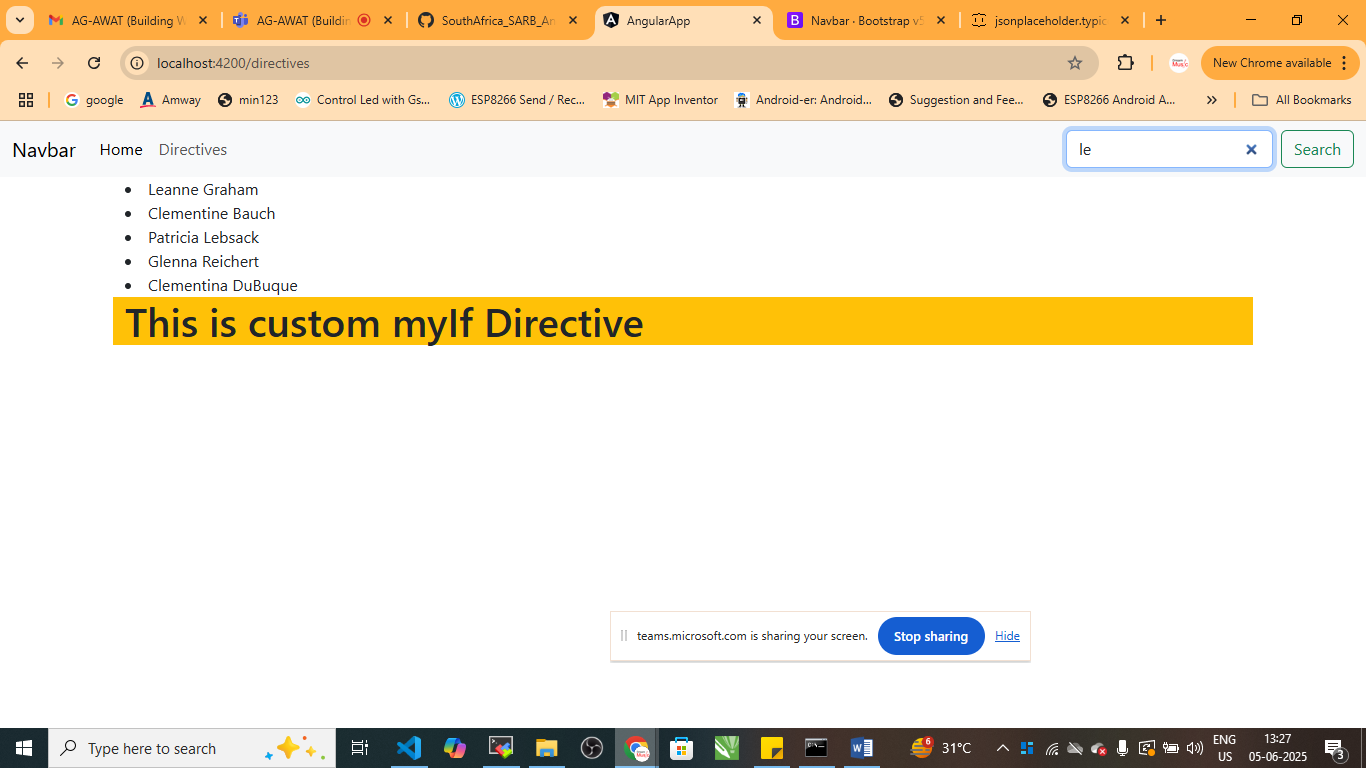
App.module.ts file

imports: [

    ReactiveFormsModule

  ],

Output:



mergeMap

* mergeMap is an RxJS Operator used to flattern and merge inner Observables emiited from a source Observalble
* it subscribe to all innerObservalbles concurrently
* Readl-World UseCase:
* Suppose you have two API’s
  + <https://jsonplaceholder.typicode.com/users>
  + [https://jsonplaceholder.typicode.com/post>userId=ID](https://jsonplaceholder.typicode.com/post%3euserId=ID)

we will get combine data using mergeMap

1. Create method inside your service

getUsers():Observable<any[]>{

    return this.http.get<any[]>('https://jsonplaceholder.typicode.com/users')

  }

  getUserPosts(userId:number):Observable<any[]>{

    return this.http.get<any[]>(`https://jsonplaceholder.typicode.com/posts?userId=${userId}`);

  }

Create user component

* ng g c users

users.component.ts

import { Component, OnInit } from '@angular/core';

import { UserService } from '../service/user.service';

import { from} from 'rxjs';

import {map, mergeMap} from 'rxjs/operators';

@Component({

  selector: 'app-users',

  templateUrl: './users.component.html',

  styleUrls: ['./users.component.css']

})

export class UsersComponent implements OnInit {

  //allPosts:any[]=[];

  userPostList:{name:string,posts:any[]}[]=[];

  constructor(private service:UserService){}

  ngOnInit(): void {

    this.service.getUsers().subscribe(users=>{

      //Create a stream from user array

      from(users).pipe(

        mergeMap(user=>this.service.getUserPosts(user.id).pipe(

          map(posts=>({

            name:user.name,posts

          }))

        ))

      ).subscribe(UserData=>{

        this.userPostList.push(UserData);

      });

    });

  }

}

users.component.html

<div class="container bg-tertuary">

    <h3>User Posts (Via MergeMap)</h3>

    <div \*ngFor="let u of userPostList">

        <h3>{{u.name}}</h3>

        <ul>

            <li \*ngFor="let p of u.posts">

                <strong>{{p.title}}</strong>

                <p>{{p.body}}</p>

            </li>

        </ul>

        <hr>

    </div>

</div>

App.component.html

  <li class="nav-item">

          <a class="nav-link" routerLink="users" href="#">Users</a>

        </li>

App.routing.module.ts

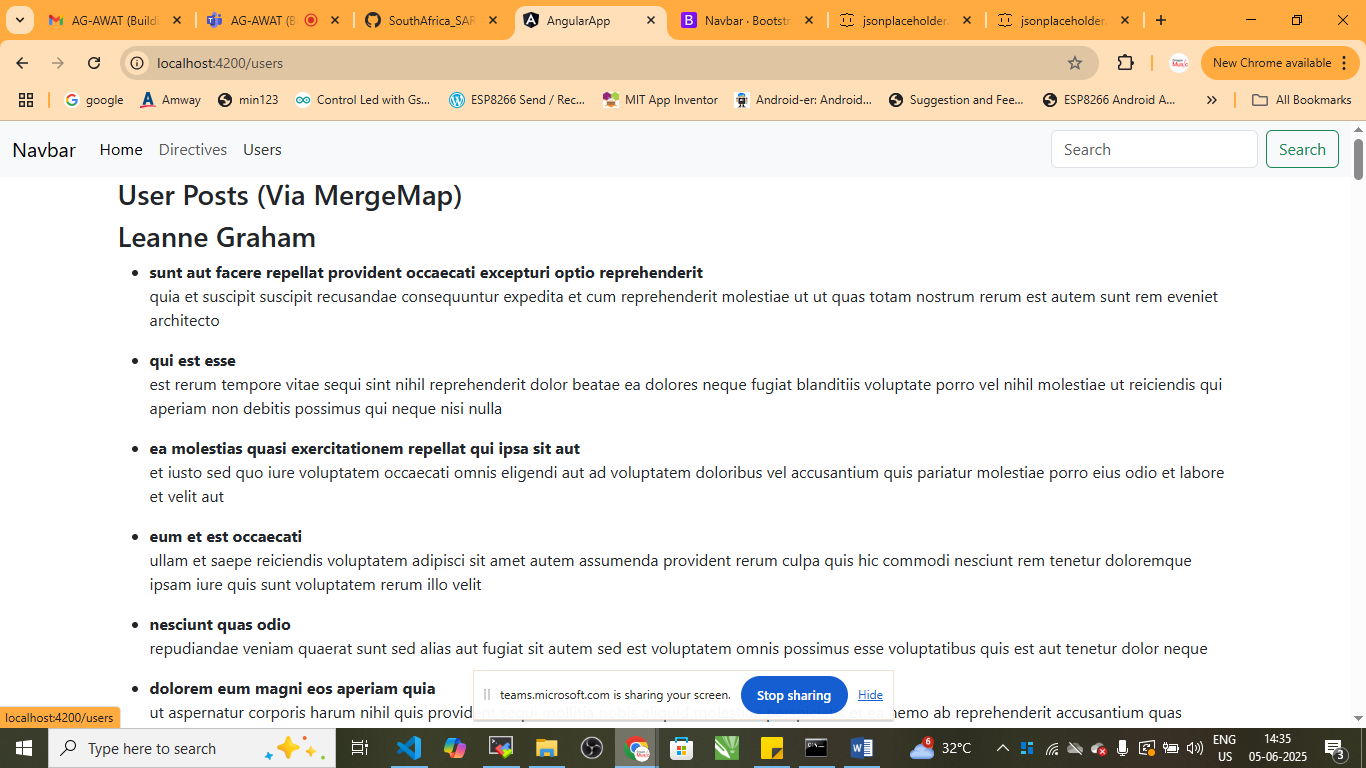
const routes: Routes = [

  {path:"directives",component:DirectiveComponent},

  {path:"users",component:UsersComponent}

];

Output



**combineLatest**

* combine latest is an rxjs operator that combines latest value from multiple observables
* real- world case:
  + when we want to filter the data

user-filter.compoennt.ts file

import { Component, OnInit } from '@angular/core';

import { FormControl } from '@angular/forms';

import { combineLatest, of } from 'rxjs';

import { map, startWith, switchMap } from 'rxjs/operators';

import { UserService } from '../service/user.service';

@Component({

  selector: 'app-user-filter',

  templateUrl: './user-filter.component.html'

})

export class UserFilterComponent implements OnInit {

  searchControl = new FormControl('');

  companyControl = new FormControl('');

  filteredUsers: any[] = [];

  companies: string[] = [];

  constructor(private userService: UserService) {}

  ngOnInit() {

    // Get users first

    this.userService.getUsers().subscribe(users => {

      // Extract unique companies

      this.companies = [...new Set(users.map(u => u.company.name))];

      // Reactive filter setup

      const search$ = this.searchControl.valueChanges.pipe(startWith(''));

      const company$ = this.companyControl.valueChanges.pipe(startWith(''));

      combineLatest([search$, company$])

        .pipe(

          map(([search, company]) =>

            users.filter(user =>

              user.name.toLowerCase().includes((search || '').toLowerCase()) &&

              (!company || user.company.name === company)

            )

          )

        )

        .subscribe(result => {

          this.filteredUsers = result;

        });

    });

  }

}

user.filter.component.html

<h2>Filter Users</h2>

<input [formControl]="searchControl" placeholder="Search by name" />

<select [formControl]="companyControl">

  <option value="">All Companies</option>

  <option \*ngFor="let company of companies" [value]="company">{{ company }}</option>

</select>

<ul>

  <li \*ngFor="let user of filteredUsers">

    👤 {{ user.name }} — {{ user.company.name }}

  </li>

</ul>

