Single page application, component based

1)download node js

2)commond prompt:npm –v

## 3) 1. Install Angular CLI (only once per machine)

npm install -g @angular/cli

👉 Installs Angular CLI globally so you can use ng commands anywhere.

npm hosts reusable code/libraries as packages that you can publish and install for others to use.

## 4) Check Angular Version

ng version or ng v

5) <https://github.com/learningPartner/angular20_tutorial> EMPTY

## 2. Create a New Angular App

ng new my-app

* my-app → your project name
* It will ask:
  + **Add Angular routing?** (y/n) → Choose y if you want routing.
  + **Which stylesheet format?** → Choose CSS, SCSS, etc.

## 📌 3. Navigate into the Project Folder

cd my-app

## 📌 4. Run the Development Server

ng serve

or with a specific port:

ng serve --open --port=4201

* --open → auto opens browser.
* Default port = **4200**.

Package.json

contains entries of all packages used in angular project some dependencies(packages for development) and dev(developer) dependencies(packages required to run application locally) are there,all the packages are downloaded from npm and stored in node module folder

scripts: u can add ur own too

angular.json

architectural file/configuration file---settings holds entire files

tsconfig.json

complier setting file

in public folder:all images

index.html:single

inside body whole application will run

entery

main.ts:executation point in {entery point in angular.json}

in main.ts there is reference for App,appconfig

Component=>(page) will contain minimum three files

Ts,html,css   
app.config.ts{providers array} is replacement app.module

App.route.ts:to handle page based navigation as single page application we use routing library

To create another page u create new component

Under app create folder components:open integrated terminal:create new components  
now how all 3 files of component linked   
using component decoratior:meta data for class Admin:ex

Imports in app.ts type class name of components

Standlone:whatever it needs it has all, component by default alone from angular 17

Creating variable

Not let,var

Rendering data in html how?{{variable}}=interplolation

### 🔹 1. Initialize Git in your folder

git init

👉 Creates a new Git repository inside your folder.

### 🔹 2. Add a remote repository (only once per project)

git remote add origin https://github.com/USERNAME/REPO\_NAME.git

👉 Replace with your GitHub repo link.  
(If remote already exists, skip this.)

### 🔹 3. Check remote added (optional)

git remote -v

### 🔹 4. Stage all files

git add .

👉 Adds all files to staging area.  
(Use git add fileName for specific files.)

### 🔹 5. Commit changes

git commit -m "First commit"

👉 Always give a meaningful commit message.

### 🔹 6. Push to remote (first time with branch setup)

git branch -M main

git push -u origin main

👉 -u sets upstream so next time you can just do git push.

### 🔹 7. For next pushes (after making changes)

git add .

git commit -m "Updated something"

git push

✅ That’s it. From now on, only steps **7** are needed for updates.

## Ep 3:

Data binding

One way dat binding=>

Interpolation,property binding:dynamic,event binding

Two way data binding=> using ngmodel

New component databinding;

All class has acces to constructor

Create variable

CLICK,CHANE,SCROOLL,MOUSE EVENT

For ngModel import formsModule,earlier this was not the case[(ngModel)]=”variable name”gift box

Change of variable value everywhere

Events()-🡪doubt

## Ep:4

what is signal,computed signal,change signal value,read signal value  
change detection cycle-🡪3rd party library zoone.js  
signal does not need  
somehow like rxjx not replacement  
future possible signal based form

## how do you create state or variable?

Normal signal

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

courseName=signal<string>("Angular");

  cousreDuration=signal("15 viseos");

we access signal as method

courseName=signal<string>("Angular");

  courseDuration=signal("15 videos");

  courseDetails=computed(()=>this.courseName()+" "+this.courseDuration());

  constructor(){

    setTimeout(()=>{

      this.courseName.set("reacts");

    },5000)

  }

<p>signal-ex works!{{courseName()}} {{courseDuration()}}</p>

<p>{{courseDetails()}}</p>

computed signal no set

## Ep:5

CONTROL FLOW STATEMENT

@if  
@else

@else if

@for

@if within for

Importance of track $index

## Ep:6

install bootstrap

npm I bootstrap

“./node\_modules/bootstrap/dist/css/bootstrap.min.css” in angular.json add

Routing

Create routes

What is router outlet

How to navigate

What is routerlink

what is npm what is its use

"styles": [

              "./node\_modules/bootstrap/dist/css/bootstrap.min.css",

              "src/styles.css",

            ]

"src/styles.css",

Always at end??

Any changes in angular.json always compile ng s

-------------------------------------------------------------------------------------------------------------------------------------------------------

In app.route.ts you are going to create routes

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

export const routes: Routes = [];

app.config

import { ApplicationConfig, provideBrowserGlobalErrorListeners, provideZoneChangeDetection } from '@angular/core';

import { provideRouter } from '@angular/router';

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

export const appConfig: ApplicationConfig = {

  providers: [

    provideBrowserGlobalErrorListeners(),

    provideZoneChangeDetection({ eventCoalescing: true }),

    provideRouter(routes)

  ]

};

Route is nothing but an object of type routes

Instead of href we are going to use routerlink which is directive

imports: [ControlFlow,RouterLink], in app.ts

imports: [User, RouterLink, RouterOutlet],

<nav class="navbar bg-dark navbar-dark navbar-expand-sm">

<div class="container-fluid">

  <!-- Links -->

  <ul class="nav navbar-nav" style="visibility: visible;">

    <li class="nav-item">

      <a class="nav-link active" routerLink="admin">Admin</a>

    </li>

    <li class="nav-item">

      <a class="nav-link" routerLink="control-flow">ControlFlow</a>

    </li>

    <li class="nav-item">

      <a class="nav-link" routerLink="signal-ex">SignalEx</a>

    </li>

    <li class="nav-item">

      <a class="nav-link" routerLink="databinding">Databinding</a>

    </li>

  </ul>

</div>

</nav>

<table>

    <tr>

      <app-user></app-user>

    </tr>

</table>

<router-outlet></router-outlet>

14:00

## Ep:7

Directive(something that adds extra behavior to html tag)

Attribute directive  
ngClass(add dynamic class to an element) and ngStyle,ngModel  
Structural Directive(\*ngIf \*ngFor)

Components are standalone they should be imported there ngClass and ngStyle both should be imported

  imports: [NgClass],

<div class="col-6" [ngClass]="divClass()">

U can create custom directive too

 imports: [NgClass,NgStyle]

<div class="col-6" [ngStyle]="{'color':divColor?'red':'green'}">

## Ep:8

Component life cycle and pipe

Difference between constructor and ngOninit

Pipes ?  
default pipes??

Here’s your cleaned-up version without timestamps:

So this is the eighth episode of Angular 20 tutorial. Till now, basic topics are done. Now we are slowly moving towards the advanced concepts. Project creation, component, variable declaration, data binding, signal, control flow statement, attribute directive, and routing are done. From here, component lifecycle events and pipes remain.

### Lifecycle Events

A component is nothing but a page. When that component renders, certain things happen behind the scenes:

* Before component loading
* When component loads (like loading images or external content)
* Before component gets destroyed (like navigating to another page)

We can write code during these phases. These are not just click or change events but events happening at the component level. These are called lifecycle events.

We are working with TypeScript, which has classes and constructors. An intro interview question is the difference between **constructor** and **ngOnInit**.

#### Constructor

The constructor is a function that executes automatically whenever an instance of the class is created. So, when we render a component, the class instance is created and the constructor executes first. After that, lifecycle events follow.

#### ngOnInit

To use lifecycle events, we need to implement them. For example, implement **OnInit** and define ngOnInit().

constructor() {

console.log("Constructor called");

}

ngOnInit() {

console.log("ngOnInit called");

}

When you navigate to the component, you’ll see the constructor executes first, then ngOnInit().

#### Other Lifecycle Events

* **ngOnChanges** → Triggers when input-bound values change (better understood with reusable components).
* **ngAfterContentInit**
* **ngAfterContentChecked**
* **ngAfterViewInit**
* **ngAfterViewChecked**
* **ngOnDestroy**

Example usage with console logs shows the sequence:

1. Constructor
2. ngOnInit
3. ngAfterContentInit
4. ngAfterContentChecked
5. ngAfterViewInit
6. ngAfterViewChecked
7. ngOnDestroy (when navigating away or destroying component)

#### Practical Usage

* **ngOnInit** → Most used, typically for API calls and subscriptions.
* **ngAfterViewInit** → Used when working with @ViewChild or DOM elements.
* **ngOnDestroy** → Used to unsubscribe or clean up resources before leaving a component.

Most of the time, developers use ngOnInit, ngAfterViewInit, and ngOnDestroy. Others are rarely needed.

### Pipes

Pipes are used to **format data** before displaying it in templates. For example, converting to uppercase, lowercase, formatting dates, etc.

#### Example

firstName: string = "chan";

courseName: string = "Angular 20 tutorial";

<p>Original: {{ firstName }}</p>

<p>Uppercase: {{ firstName | uppercase }}</p>

<p>Lowercase: {{ courseName | lowercase }}</p>

<p>Title Case: {{ courseName | titlecase }}</p>

#### Array with Slice Pipe

rollNumberList = [11, 12, 13, 14, 15, 16, 17];

<p>Original: {{ rollNumberList }}</p>

<p>Sliced: {{ rollNumberList | slice:0:3 }}</p>

Output: [11, 12, 13]

#### JSON Pipe

For objects, Angular prints [object Object]. Using the JSON pipe makes it readable:

studentObj = { name: "Jet", city: "Pune", state: "MH" };

<p>{{ studentObj }}</p>

<p>{{ studentObj | json }}</p>

#### Date Pipe

currentDate: Date = new Date();

<p>Original: {{ currentDate }}</p>

<p>Formatted: {{ currentDate | date:'dd-MM-yyyy' }}</p>

<p>Month Name: {{ currentDate | date:'dd-MMM-yyyy' }}</p>

<p>Full Month: {{ currentDate | date:'dd-MMMM-yyyy' }}</p>

* dd → day
* MM → month number
* MMM → short month name
* MMMM → full month name
* yy or yyyy → year

This is one of the most commonly used pipes.

  imports: [NgClass,NgStyle,UpperCasePipe],

Got it 👍 Let’s break down **JSON in JavaScript** step by step.

## 1. What is JSON?

* **JSON = JavaScript Object Notation**
* A lightweight format for storing and exchanging data.
* Looks like JavaScript objects but is always **text (string)**.

👉 Example of JSON data:

{

"name": "Anusha",

"age": 25,

"isStudent": false,

"skills": ["HTML", "CSS", "JavaScript"]

}

## 2. JSON vs JavaScript Object

* **JavaScript Object** → actual object in memory.
* **JSON** → string representation of that object.

// JavaScript Object

let user = { name: "Anusha", age: 25 };

// JSON (string)

let jsonUser = '{"name":"Anusha","age":25}';

## 3. Converting between JSON and Object

JavaScript has two built-in methods:

### ✅ Object → JSON string

let person = { name: "Anusha", city: "Bangalore" };

let jsonStr = JSON.stringify(person);

console.log(jsonStr);

// Output: {"name":"Anusha","city":"Bangalore"}

### ✅ JSON string → Object

let jsonData = '{"name":"Anusha","city":"Bangalore"}';

let obj = JSON.parse(jsonData);

console.log(obj.name);

// Output: Anusha

## 4. JSON Rules

* Keys must be in **double quotes** "".
* Values can be:
  + string "text"
  + number 123
  + boolean true/false
  + null
  + object {}
  + array []

❌ JSON does **not** allow:

* Single quotes ' '
* Functions
* undefined

## 5. Where is JSON used?

* Sending data between frontend and backend (APIs).
* Storing configuration (package.json, angular.json, etc).
* Local storage / session storage in browsers.

👉 Quick Demo:

// Sending user info as JSON

let user = { name: "Anusha", skills: ["Angular", "React"] };

let jsonStr = JSON.stringify(user);

console.log("To Server:", jsonStr);

// Suppose server replies with JSON string

let serverReply = '{"status":"success","id":101}';

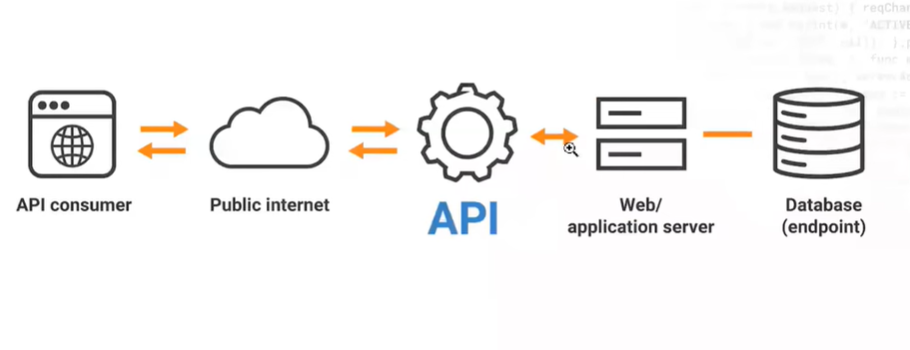
let replyObj = JSON.parse(serverReply);

console.log("From Server:", replyObj.status); // success

<https://angular.dev/guide/templates/pipes>

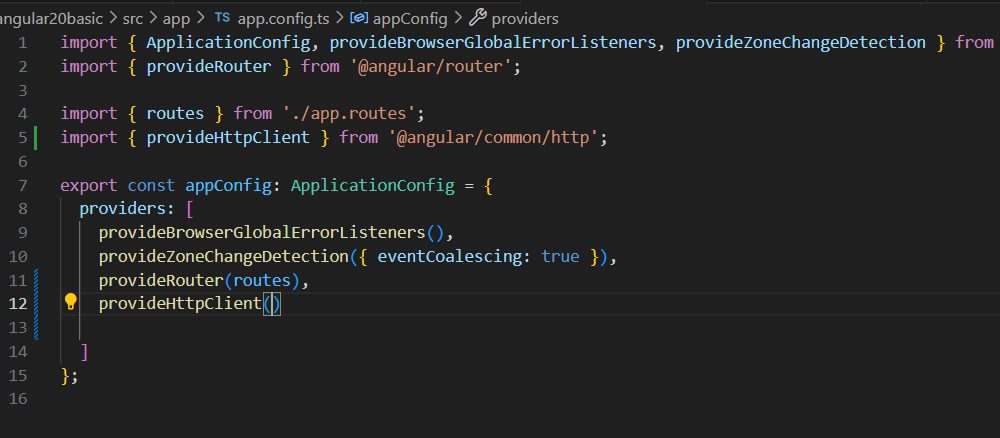
## ep:9

all the data will have api call,  
api is the mediator which gets data from the database  
  
what is api?url  
what are get,post(add new record),put(update existing record),delete method  
how to intergrate api in angular app,using httpclient library  
json server api/mock api  
real api  
httpclient library//rxjs async calls

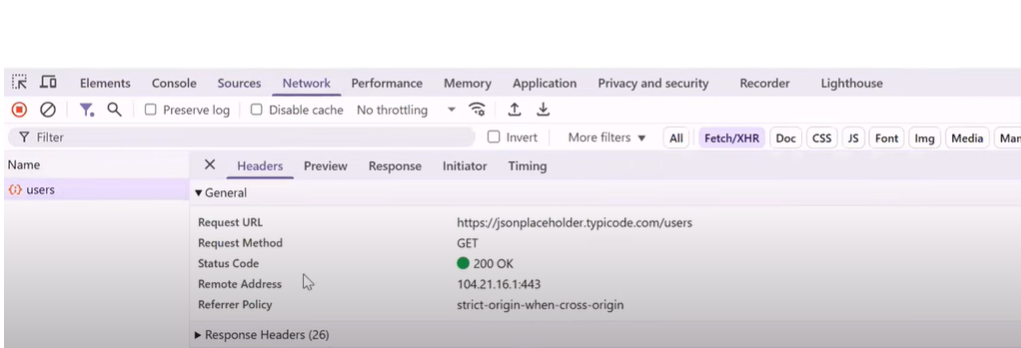


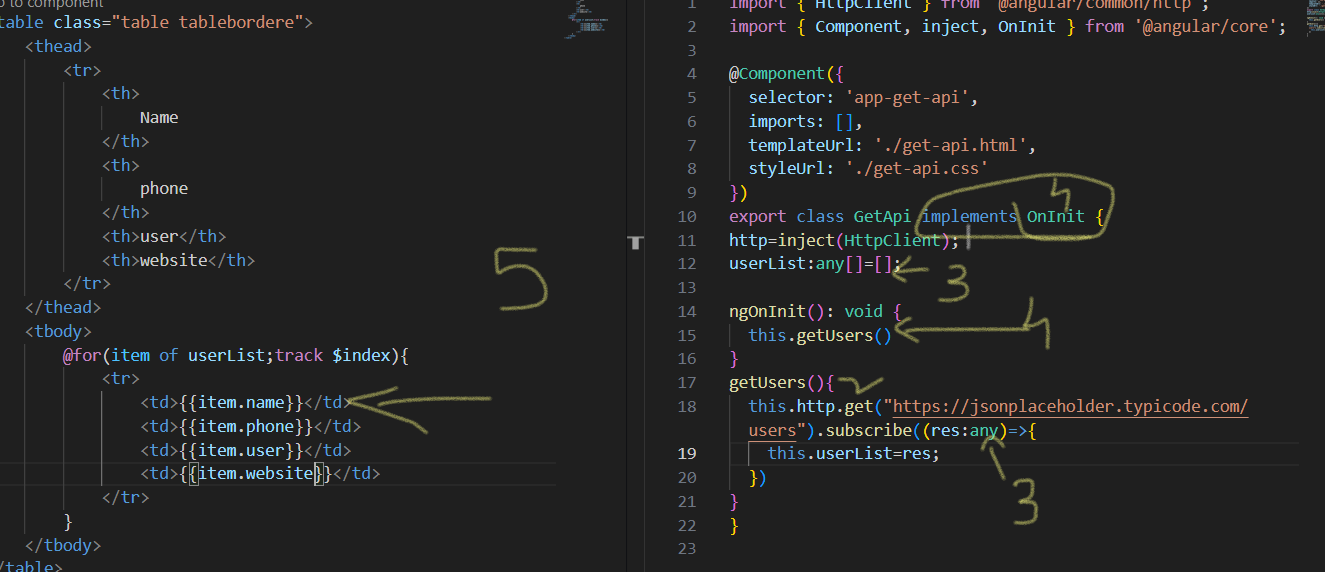
Api is a separate team:they will provide api url

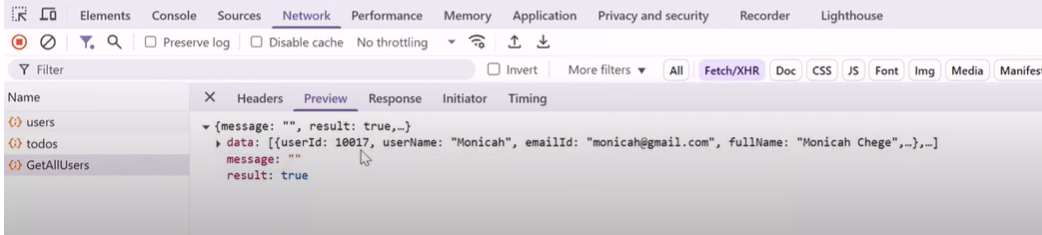
Mock:json placeholder  
json:array of objects



From angular 16

Dependency injection  
  


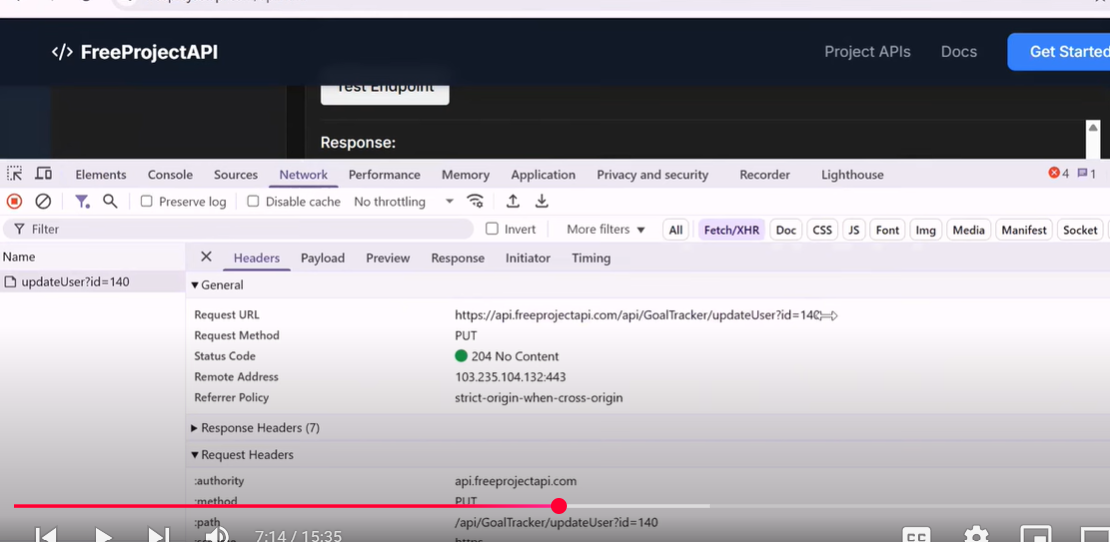
https://freeprojectapi.com/  




-----------------------------------------------------------------------------------------------------------------------------------------------------------  
ep:10

Post api=>template form

What involves in curd application reading data,creating,updating existing record,displaying data

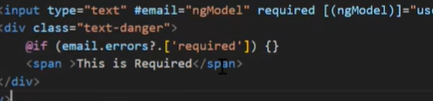


## Ep:15: Reactive Form in angular

Create reactive form   
bind form with template  
read form value  
crud operation with reactive form

## Ep:16:form validation

Template form:  
form validation  
create #email=”ngModel”-🡪variable



email regex pattern for angular  
form button disabled

<form  #formValue="ngForm">

<input name="emailId" pattern="[a-zA-Z0-9.-\_]{1,}@[a-zA-Z.-]{2,}[.]{1}[a-zA-Z]{2,}" #emailId="ngModel" required type="text" [(ngModel)]="userObj.emailId" class="form-control">

Name attribute must



## Ep:17

What are services in angular  
how to create service  
how to inject service  
scenarios to use services  
  
git is nothing but a class,some code repeatedly used in most component,and put is wherevere required  
reusability  
  
create some utility,storing api call function,array sorting

Component has component directive/decorator  
pipe has pipe decorator  
service has injectable decorator  
helps understand extra information about class  
httpclient is also a service  
  
get function returns observable type of data

## Ep:18

**Resource & rxResource API**

* What Are Resource API in Angular?
* What Are rxResource API in Angular?
* How to Do API integration using Resource API?
* New Feature from Angular 17..

# Resource API & RxResource API

Till now, we have covered:

* Data binding
* Routing
* API call integration with HttpClient
* Template forms & Reactive forms
* Services

Now, moving into **advanced Angular topics**:

From Angular 17 onwards, we got a new feature called **Resource API**.

Just like we already use **RxJS + HttpClient** for async API calls, Angular also provides **Resource API** with two variants:

1. **Resource** – works with **Promise** (uses fetch)
2. **RxResource** – works with **RxJS + HttpClient**

## 1. Resource API (Promise-based)

* Works with **Promises**, so internally we often use fetch.
* Example: Fetching user data from **JSONPlaceholder** API.

### Code Example (resource with fetch):

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

import { resource } from '@angular/core/rxjs-interop';

@Component({

selector: 'app-users',

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

})

export class UsersComponent {

userData = resource({

loader: () => fetch('https://jsonplaceholder.typicode.com/users')

.then(res => res.json())

});

}

### HTML Example (display data):

<div \*ngIf="userData.hasValue(); else loading">

<ul>

<li \*ngFor="let user of userData.value">

{{ user.name }} - {{ user.email }}

</li>

</ul>

</div>

<ng-template #loading>

<p>Loading users...</p>

</ng-template>

### Notes:

* userData.hasValue() → returns true once data is available.
* userData.value → contains the response data.
* userData.reload() → can trigger the API call again.

#### Example reload after 6 seconds:

constructor() {

setTimeout(() => {

this.userData.reload();

}, 6000);

}

## 2. RxResource API (RxJS-based)

* Works with **RxJS + HttpClient**.
* More Angular-native way compared to fetch.
* Loader property (in older versions) is now replaced with a **stream**.

### Code Example (RxResource with HttpClient):

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

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

import { rxResource } from '@angular/core/rxjs-interop';

@Component({

selector: 'app-users-rx',

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

})

export class UsersRxComponent {

http = inject(HttpClient);

userList = rxResource({

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

});

}

### HTML Example:

<div \*ngIf="userList.hasValue(); else loadingRx">

<ul>

<li \*ngFor="let user of userList.value">

{{ user.name }} - {{ user.email }}

</li>

</ul>

</div>

<ng-template #loadingRx>

<p>Loading users with RxResource...</p>

</ng-template>

## Key Features of Resource & RxResource

* .hasValue() → check if data is available.
* .value → access the response.
* .reload() → reloads the resource (like triggering API again).
* .isLoading → returns true/false if request is in progress.

## Interview Pointers

* **Resource API** → Promise-based, uses fetch.
* **RxResource API** → RxJS-based, uses HttpClient.
* **Difference** → Resource = Promise, RxResource = Observable.
* Both provide useful state properties: hasValue, value, isLoading, reload.
* Introduced around Angular 17+, so it’s a **modern feature** often asked in interviews.

✅ So now you know **3 ways to call APIs in Angular**:

1. HttpClient (traditional & most common)
2. Resource (Promise-based)
3. RxResource (RxJS-based, recommended)

## Ep:17

Login page in Angular

Website login—some pages can be accessed without login,ex:amazon

Admin app login---only login page visible

Login with hard core credentials  
  
hard code some user credentials and allow user to login with that only  
local storage🡪sign up(store in local storage and use that only to login) and login page

For testing machine test

Login with api intergration

Login with api---return user data

Login with api 🡪on login whatever user credentials u passed is correct token based(jwt token)

Hard core login:

On login button has to change:

On loging in and page reload the log out button is shown

What is subject behaviour?