TECHNOLOGY

Caltech | Center for Technology & Management Education

Full Stack Java Developer

simplilearn

Angular

TECHNOLOGY

Angular Service Layer

Learning Objectives

By the end of this lesson, you will be able to:

- Learn Angular HTTP clients with various requests methods
- Describe HTTP Headers with Cross-Origin Resource Sharing
- Understand how to set up server communication in Angular
- Illustrate Async Pipe
- Define Angular services and discuss how to create them

A Day in the Life of a Full Stack Developer

You are working in an organization and have been assigned a website project. You are trying to develop and host a website on the organization's server, where you want to restrict resources from similar domains.

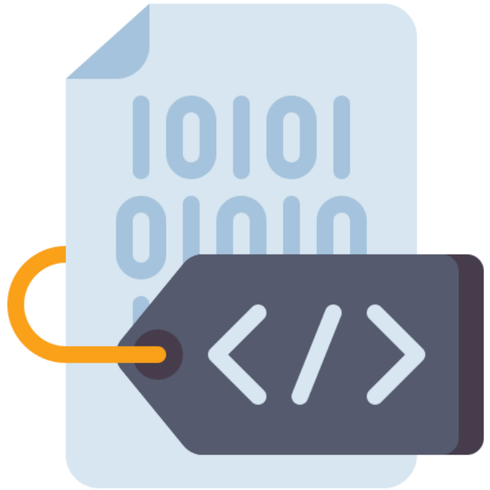
In any case, if you want to access these resources, you need to request the resources from another domain outside the domain from which the first resource was requested.

This can be done with the help of CORS. To do so, you need to explore the Angular HTTP client, CORS, and async pipe.

Basic Routing and Routing Parameters

Basic Routing and Routing Parameters

In Single Page Applications (SPA), all application's functions exist within a single HTML web page.

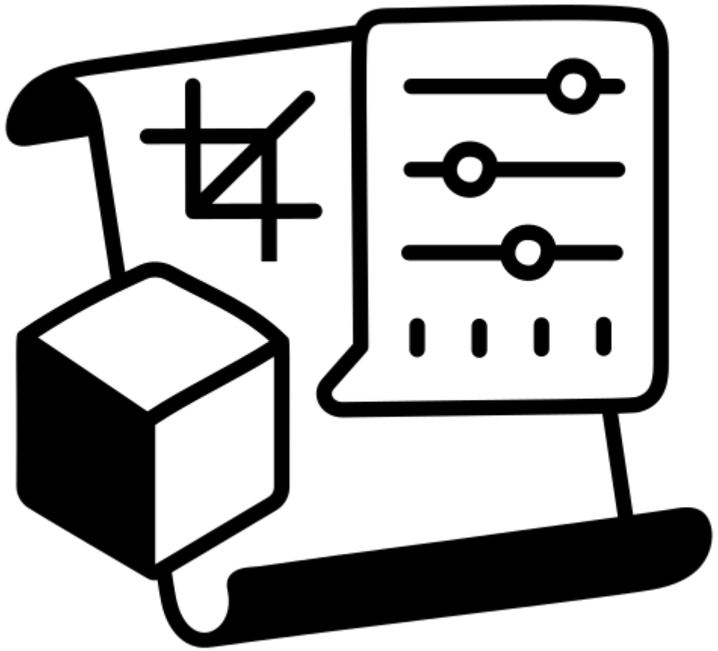


Basic Routing and Routing Parameters

Angular routing helps the user navigate from one component to another.

Angular Routing

Using routing, the user can configure the routes of their own application in an app module file.

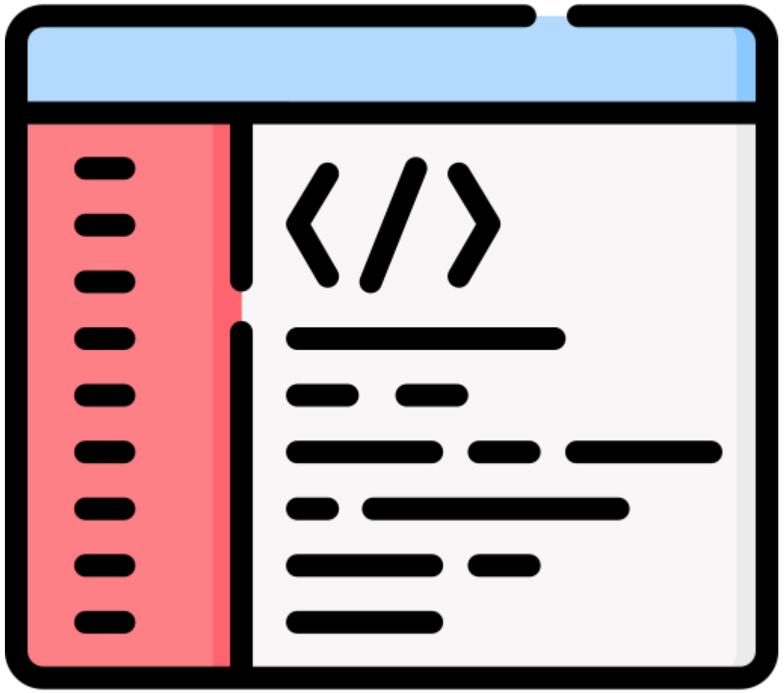


Angular Routing

<router-outlet> is an element where each component will be rendered to correspond to the Angular route and act as a placeholder for the component.

Angular Routing

The RouterLink directive is used to navigate to the route in the application and creates a link like an HTML anchor tag.



Angular Routing

Example: Creating routes in the router module file

Syntax:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { AccountCompomnent } from `./account.component';
import { HomeCompomnent } from `./home.component';
import { ServiceComponent } from `./service.component';
const routes: Routes = [
      { path: `', redirectTo:`/home',pathMatch: `full' },
      { path: `home', component: HomeComponent },
      { path: `account', component: AccountComponent },
      { path: `service', component: ServiceComponent }
};
@NgModule({
imports: [RouterModule.forRoot(routes)],
exports : [RouterModule]
})
export class AppRoutingModule {}
```

Angular Routing

Example: Importing the AppRouting module from the app module file of the Angular app

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';

@NgModule({
  imports: [
      BrowserModule,
      AppRoutingModule,
],
Bootstrap: [ AppComponent ]
})
export class AppModule {}
```

Angular Routing

Create a navbar to navigate between the configured routes in the application

Bootstrap can be used for creating a good navbar for the application in app.component.html.

Angular Routing

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"><a
Class="nav-link" routerLink="/home">Accounts</a><li>
        <li class="nav-item" routerLinkActive="active"><a
Class="nav-link" routerLink="/home">Services</a><li>
        <li class="nav-item" routerLinkActive="active"><a
     </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>
```

Angular Routing: Rules

Angular uses the first-match strategy while navigating a specific route.

1. Static routes should be placed first.
2. Empty-path route should match the default route.
3. The wildcard route should be placed last because it matches every route.
3. Programmatic navigation should be used to navigate from one route to another for events, like buttons, clicks, links, and other dynamic routes.

Angular Routing

Syntax:

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

@Component({
   selector: 'app-home',
   templateUrl: './home.component.html'
})
export class HomeComponent implements OnInit {
   userId = 'UID0023';
   constructor(private router: Router) {}

   ngOnInit() {
  }
navigateToAccount() {
this.router.navigate(['/account']) ;
```

Path Redirection

Path Redirection

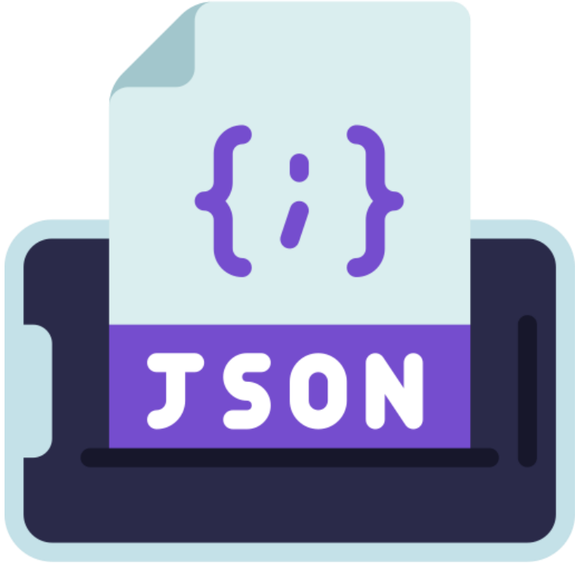
Path redirection is used to redirect the user from one component to another component using Angular routing in the application.

- Consider the path from where the user in the application needs to be redirected
- Bind the component with the route from where the redirection should happen
- Use the pathMatch value to tell the router to navigate when routes match completely

Angular HTTP Client

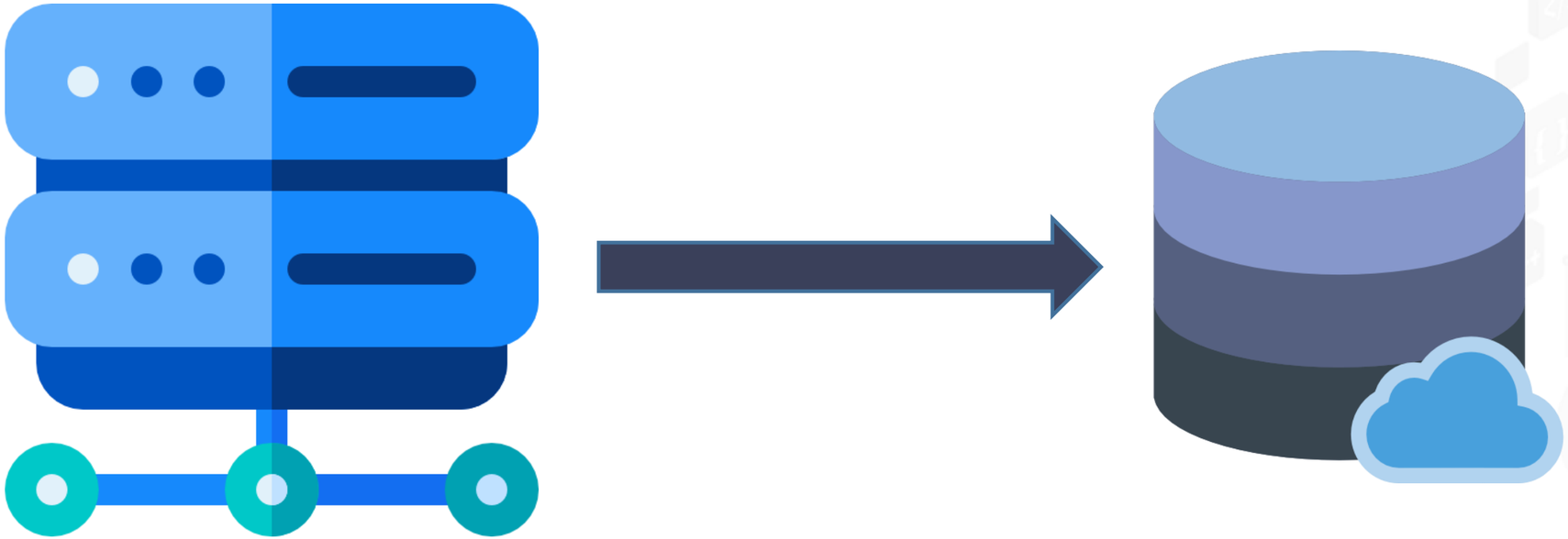
Angular HTTP Client

The Hypertext Transfer Protocol is used for transmitting hypermedia documents like HTML or data over the network.



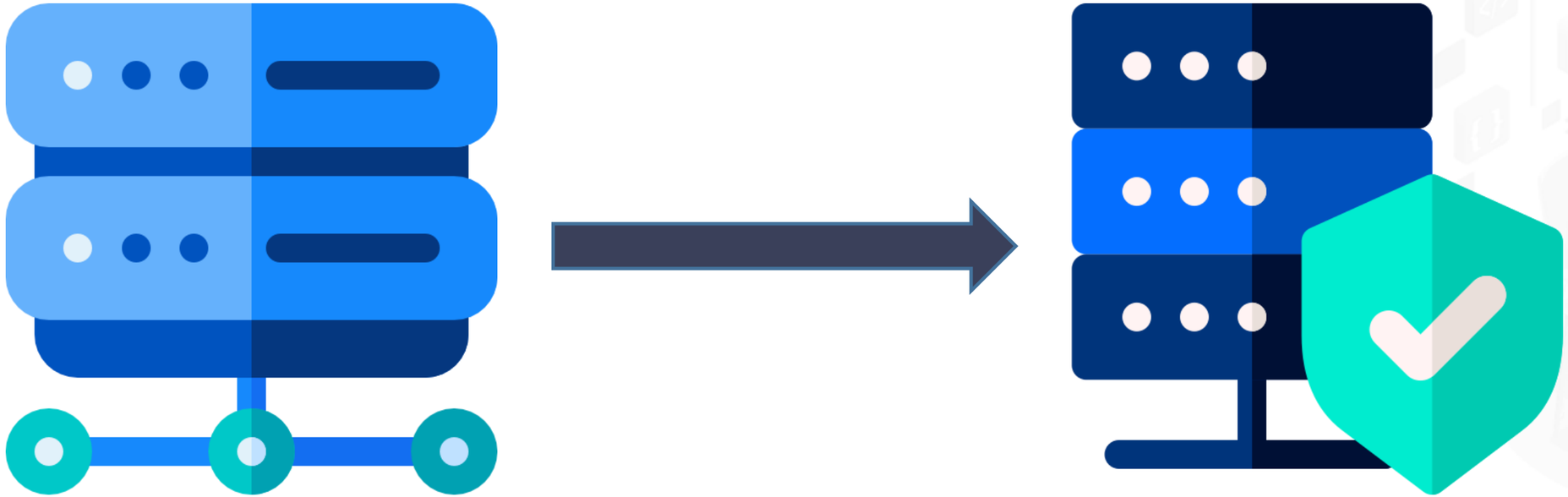
Angular HTTP Client

An HTTP Request is sent from a client to the server to get particular data from the database.



Angular HTTP Client

The HTTP Response is sent from the server to the client application which requested the data.



Angular HTTP Client

GET

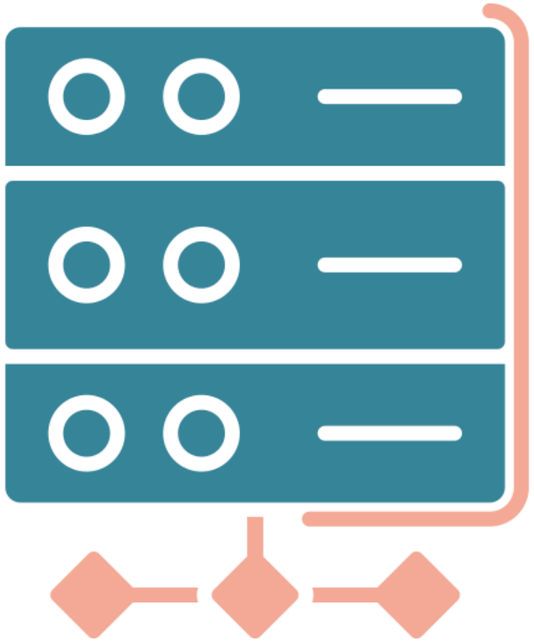
POST

PUT

PATCH

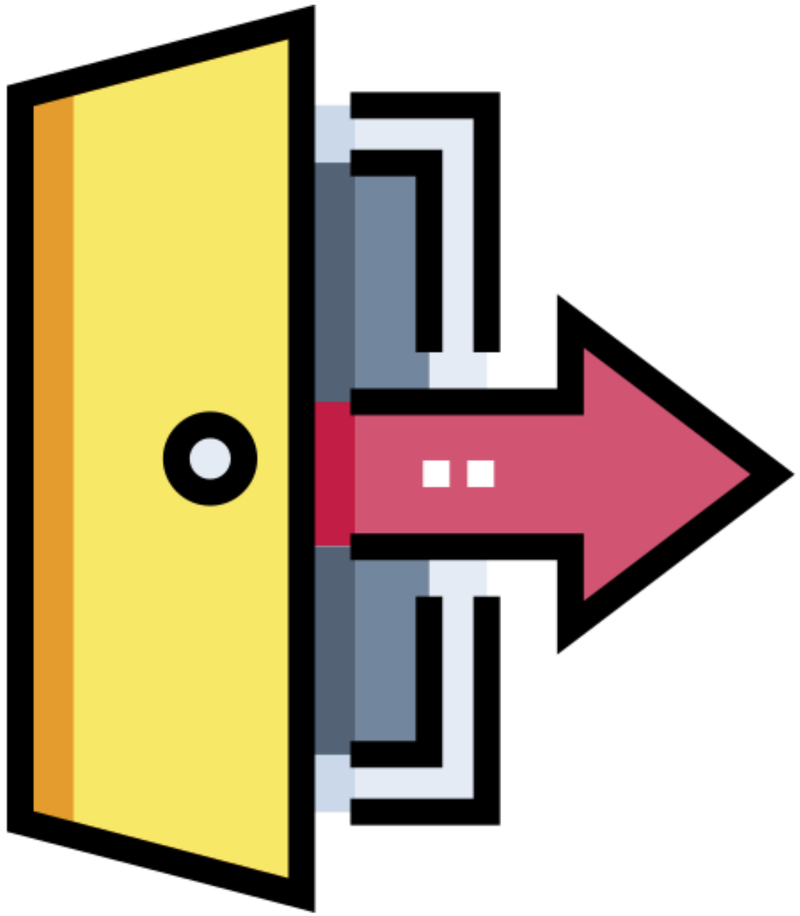
DELETE

Used to request data from the server, and it is the most used request method



Angular HTTP Client

The GET Method can:



- Be cached
- Remain in the browser history
- Be bookmarked
- Have length restrictions
- Be used to request data (not modify)

Angular HTTP Client

Example:

```
Request URL:
https://newsapi.org/v2/everything?q=tesla&from=2021-10-
24&sortBy=pubshiedAt
Request Method: GET
Status Code: 200
Remote Address: 104.26.12.149:443
Referrer Policy: strict-origin-when-cross-origin
```

Angular HTTP Client

- GET
- POST
- PUT
- PATCH
- DELETE

Used to send data from the client to the server for creating and updating operations on records

Angular HTTP Client

POST requests:

- Hold no restrictions on data length
- Are never cached
- Cannot be bookmarked
- Do not remain in the browser history

Angular HTTP Client

Example:

```
POST /users/add-user HTTP/1.1
Host: localhost:3000
Content-Type: application/x-www-form-urlencoded
Content-Length: 71

firstName=ABC&lastName=Singh&email=abc@40example.com
com&password=manjot123
```

Angular HTTP Client

GET

POST

PUT

PATCH

DELETE

Modifies an existing record or creates a new record, if required, on the server, for the future

A similar request is received once or several times in rows

No effect on the server and leaves it in the same state

Angular HTTP Client

Example:

```
PUT /users/update-user/619de842a94b322af548155a
HTTP/1.1
Host: localhost:3000
Content-Type: application/x-www-form-urlencoded
Content-Length: 32

firstName=ABCDEF&lastName=GHIJKL
```

Angular HTTP Client

- GET
- POST
- PUT
- PATCH
- DELETE

Used to modify existing records on the server partially

Angular HTTP Client

Example:

```
PATCH/file.txt HTTP/1.1
Host: www.example.com
Content-Type: application/example
If-Match: "e0023aa4e"
Content-Length: 100

[description of changes]
```

Angular HTTP Client

GET

POST

PUT

PATCH

DELETE

Deletes a record from the server

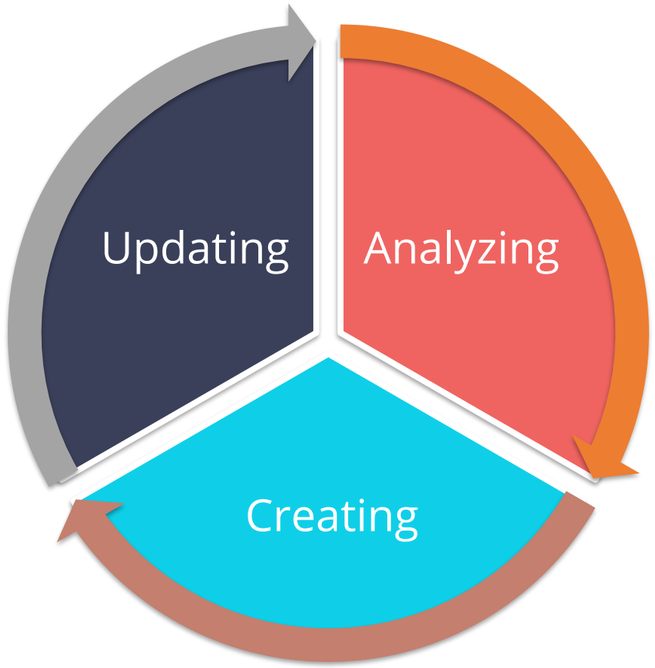
Angular HTTP Client

Example:

```
DELETE /users/delete-user/619de842a94b322af548155a
HTTP/1.1
Host: localhost:3000
```

HTTP Headers

HTTP headers are used to send additional information with an HTTP request or send a response in the form of JSON or any other type.



HTTP Headers

HTTP headers consist of the case-sensitive name followed by a colon (:) and then by its required value without whitespace.

HTTP Headers

HTTP headers are grouped into these categories:

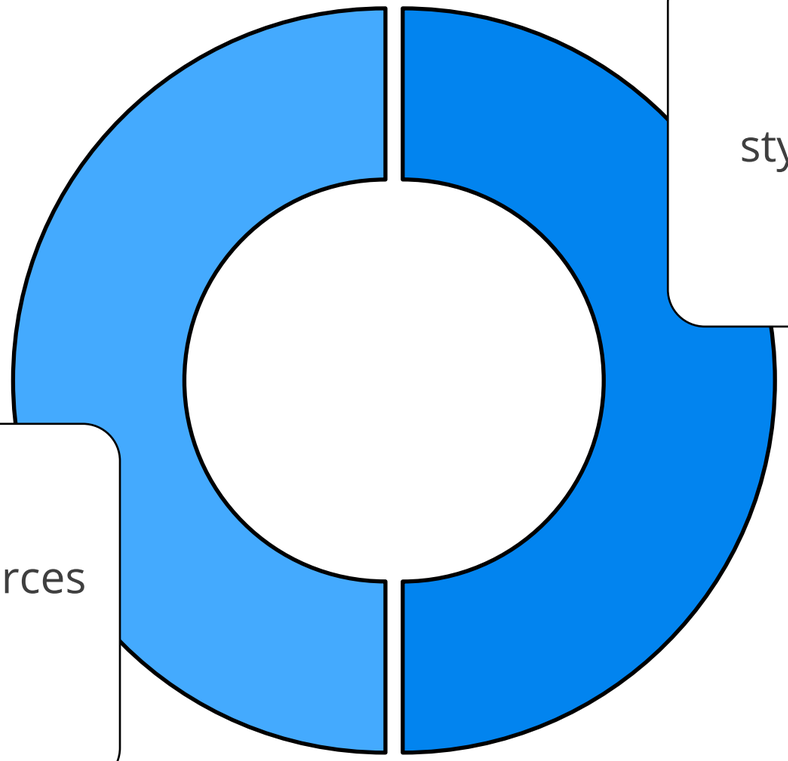
- Payload header
- Request header
- Representation header
- Response header

Cross-Origin Resources Sharing (CORS)

CORS helps to request restricted resources from another domain outside the domain from which the first resource was requested.

Cross-Origin Resources Sharing (CORS)

It acts as a communication medium between the server and a client application for resource exchange.



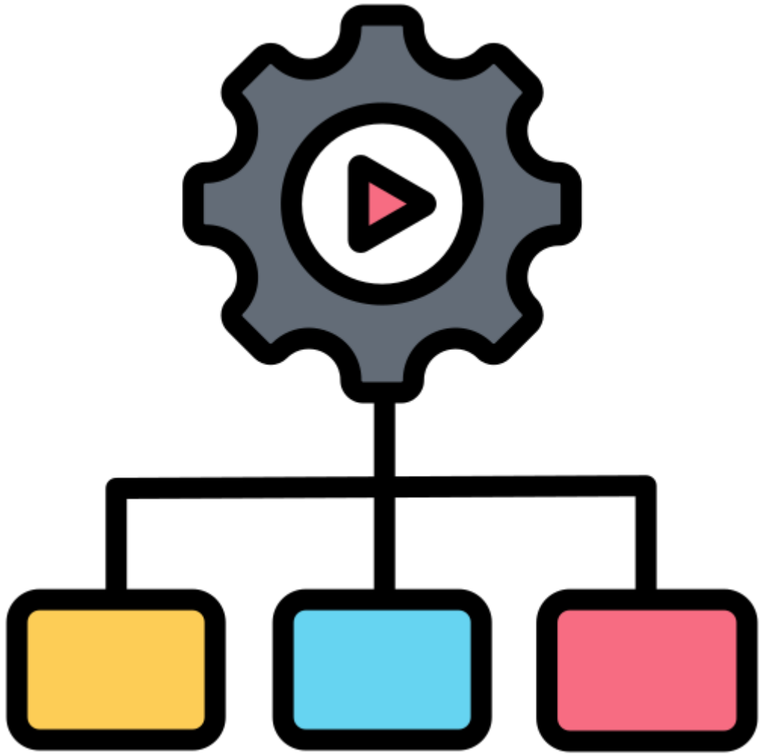
Web pages freely embed stylesheets, scripts, iframes, and videos.

Web pages do not embed resources from another domain.

Angular HTTP Client Module

Angular HTTP Client Module

Import HttpClientModule from "@angular/common/http" in the "app.module.ts"



Angular HTTP Client Module

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';

@NgModule({
   imports: [
       BrowserModule,
      // import HttpClientModule after BrowserModule.
       HttpClientModule,
      ],
      declaration: [
        AppComponent,
      ],
      bootstrap: [ AppComponent ]
})
export class AppModule {}
```

Angular HTTP Client Module

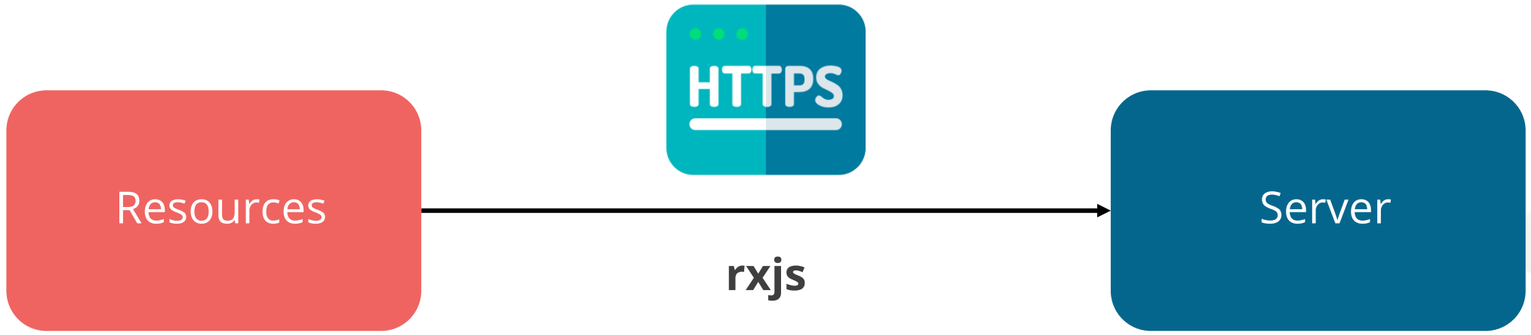
After importing HttpClientModule in the app module file, inject the HttpClient in the service or component of an application class.

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

@Injectable()
export class FetchNewsService {
  constructor(private http: HttpClient) {}
}
```

Angular HTTP Client Module

The HTTP client uses observables from the "rxjs" package for all transactions that request resources to the server.



Angular HTTP Client Module

The first method that can be used is the GET method.

```
this.httpClient.get <{ key1: any[], key2: any[] }>(
      'http://localhost:3000/attainments/dbehjbdwu7w'
}.toPromise()
.then((value)=> {
      //code to be executed on success
      console.log(value);
},    (error)=> {
      // code to be executed on error
      console.log(error);
});
```

Angular HTTP Client Module

Next, the POST method is used:

```
this.httpClient.post (
'http://localhost:3000/course-outcomes/add-co',
{ …values }
)
      .toPromise()
      .then((value)=> {
      //code to be executed on success
      console.log(value);
},    (err)=> {
      // code to be executed on error
      console.log(">>> error", err);
});
```

Angular HTTP Client Module

Next, the PUT method is used:

```
this.httpClient.put (
'http://localhost:3000/course-outcomes/update-co/sdfghjkgf3456',
{ …values }
)
      .toPromise()
      .then((value)=> {
      //code to be executed on success
      console.log(value);
},    (err)=> {
      // code to be executed on error
      console.log(">>> error", err);
});
```

Angular HTTP Client Module

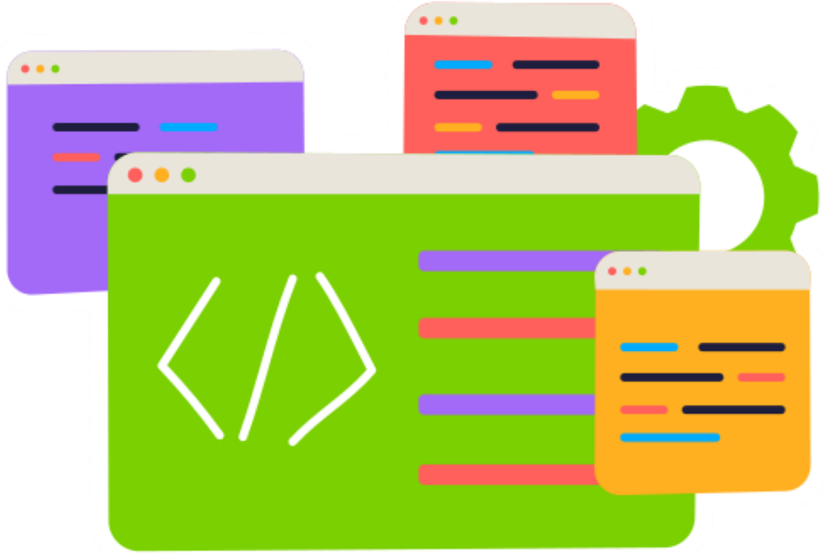
Finally, the DELETE method is used:

```
this.httpClient.put (
'http://localhost:3000/course-outcomes/delete-co/sdfghjkgf3456',
)
      .toPromise()
      .then((value)=> {
      //code to be executed on success
      console.log(value);
},    (err)=> {
      // code to be executed on error
      console.log(">>> error", err);
});
```

Async Pipe

Async Pipe

The Async pipe is different from other built-in pipes which are available in Angular.



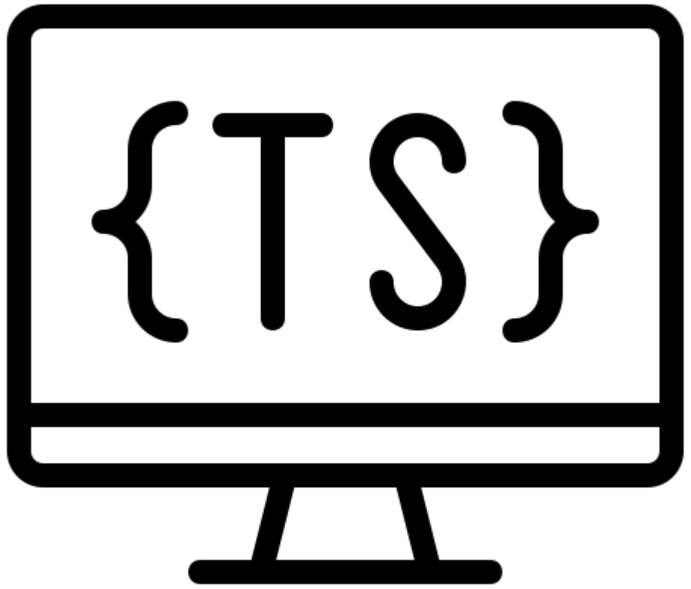
Async Pipe

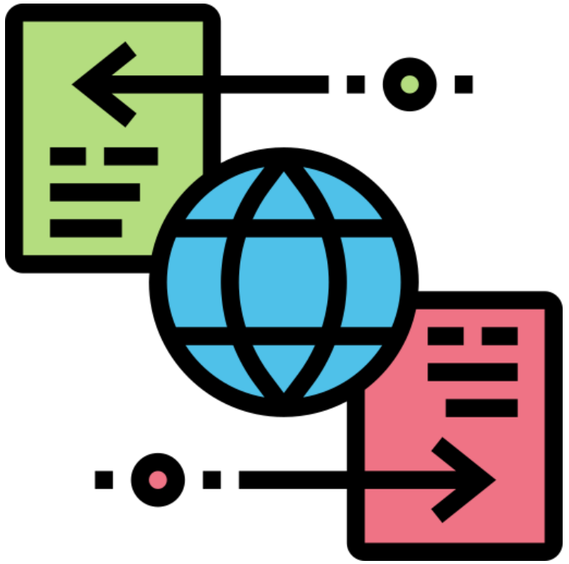
It is used to subscribe to an Observable or a Promise and return the latest value from them.

Automatically unsubscribes when an angular component is destroyed and avoids a memory leak.

Async Pipe: Introduction

Define additional code to unsubscribe to an Observable or a Promise manually in the TypeScript file





Advantages of Async Pipe

- It makes it easier to get values from the Observables or Promise directly.
- It calls the then method of the Observable or a Promise automatically.
- It subscribes to or unsubscribes from the Observable or Promise automatically.

Async Pipe

Example:

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

@Component({
   selector: 'app-root',
   templateUrl: './app.component.html',
})
export class AppComponent{
   name: string;
   constructor(){
      this.getValue().then(v=> this.name=v);
   }

   getValue(){
      return new Promise<string>((resolve, reject)=> {
       setTimeout(()=>resolve("ABC Technologies"), 2000);
       });
    }
}
```

Async Pipe: Example

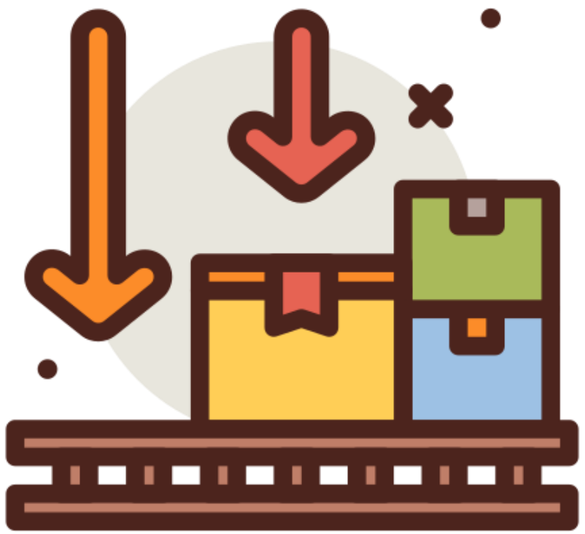
Using Template (app.component.html):

Syntax:

```
<div class="container">
{{name}}
</div>
```

Using Async Pipe

Avoid calling the then() method of Promise to get value from it.



Using Async Pipe

Using the Component class (app.component.ts):

Syntax:

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

@Component({
   selector: 'app-root',
   templateUrl: './app.component.html',
})
export class AppComponent{
   name: string;
   constructor(){
      this.promise()= this.getNameValue();
   }

   getNameValue(){
      return new Promise<string>((resolve, reject)=> {
        setTimeout(()=>resolve("ABC Technologies"), 2000);
        });
    }
}
```

Using Async Pipe

Template (app.component.html):

Syntax:

```
<div class="container">
{{ namePromise | async }}
</div>
```

Async Pipe with Observables

Directly using Observable: Component class (app.component.ts):

Syntax:

```
import { Component } from '@angular/core';
import { Component ,OnDestroy, OnInit } from '@angular/core';
import { Observable, Observer, Subscription } from 'rxjs';

@Component({
   selector: 'app-root',
   templateUrl: './app.component.html',
})
export class AppComponent implements OnInit, OnDestroy{
   counter: number;
   observable: Observable<number>;
   subscription: Subscription;
   ngOnInit() {
      this.observable = new Observable ((observer:
Observer<number>)
```

Async Pipe with Observables

Using the Template (app.component.html):

Syntax:

```
<div class="container">
{{ counter }}
</div>
```

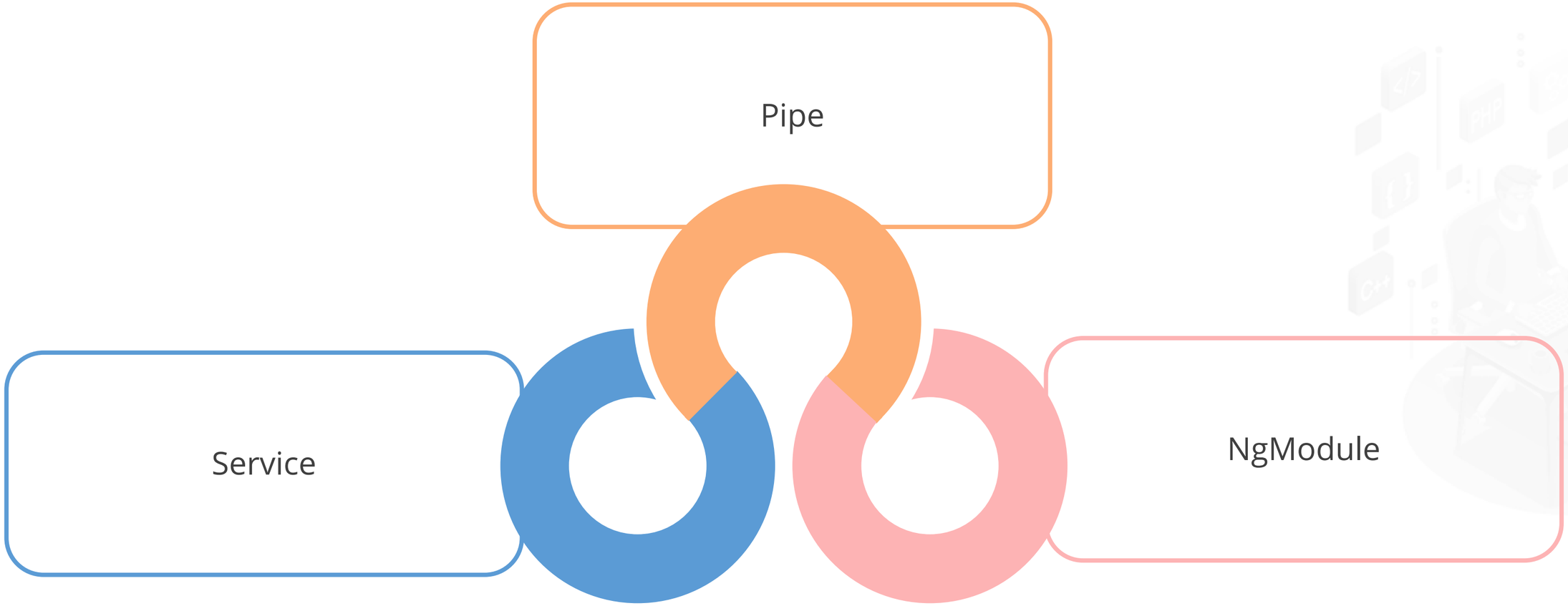
Angular Custom Services: Injectable Decorator

Injectable Decorator

The Injectable Decorator is closely wired into the Angular framework and is used everywhere to provide new components with services.

Injectable Decorator

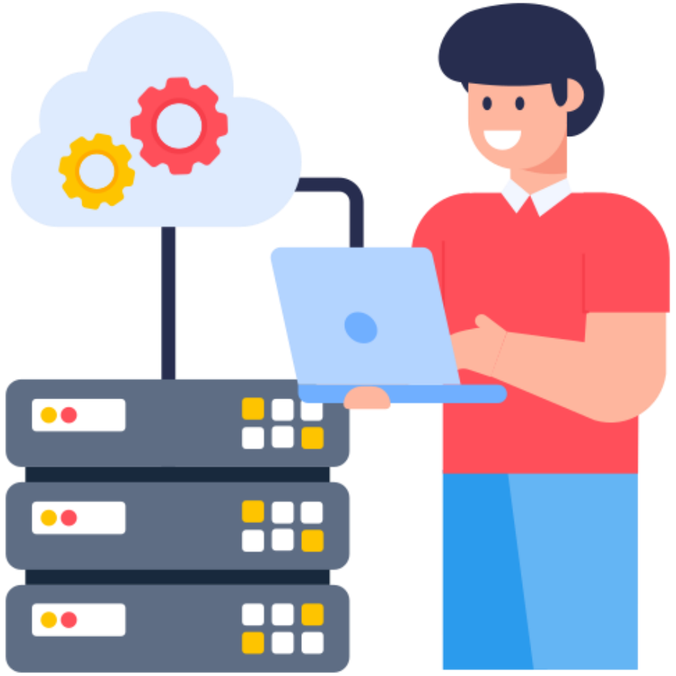
It is used to provide metadata which enables Angular to inject it into a component as a dependency.



Injectable Decorator

Angular automatically creates an injector during the bootstrap process. It:

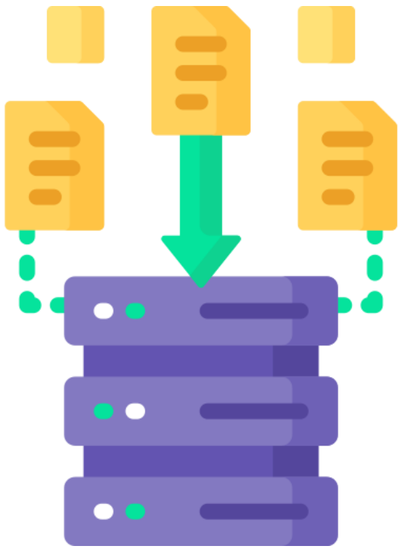
Creates dependencies



Maintains a container of dependency

Injectable Decorator

A provider is an object that tells an injector how to obtain or create a dependency.



It determines which services or other dependencies that component needs by looking at the constructor parameter types.

Injectable Decorator

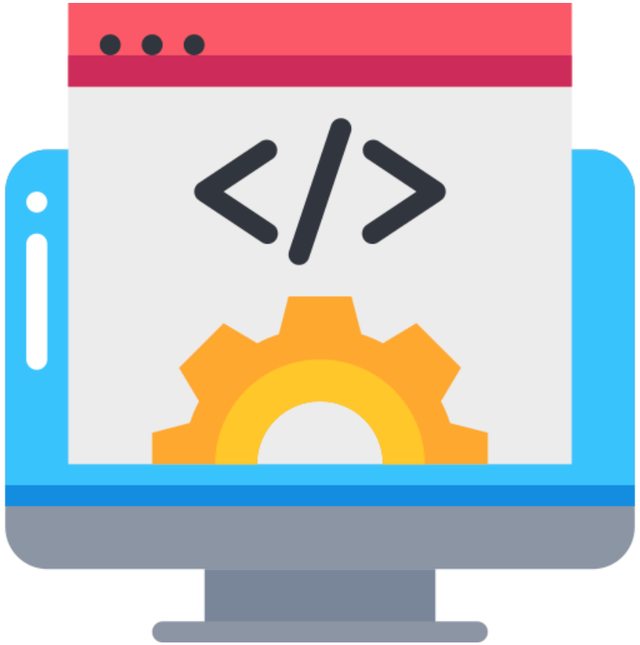
For example, the constructor of TopListComponent needs HeroService:

```
src/app/top-list.component.ts (constructor)
Constructor(private service: TopService) {}
```

Injectable Decorator

Angular checks if the injector has any existing instances of that service.

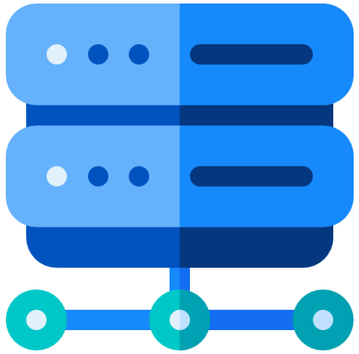
The injector makes a requested service instance using the registered provider and adds it to the injector before returning the service to Angular.



Angular Custom Services: Fetching Data

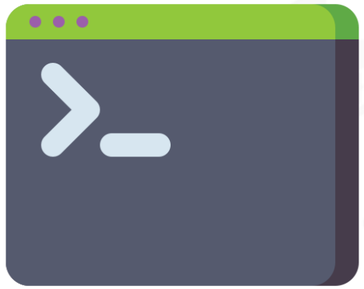
Fetching Data

Any logic like fetching data can be delegated to the service in Angular.



Server calls

User input validations



Console to login

Advantages of Services

- Components are lean because they have code related to user experience.
- Application Logic is written into service classes, which can be made available for any component in Angular.

Fetching Data

A service file can be created using this command:

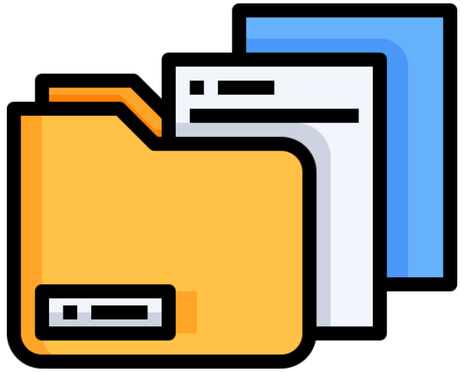
```
ng generate service <service-name>
```

```
ng g s <service-name>
```

```
ng g s services/logger
```

Fetching Data

Angular CLI creates a logger.service.ts, which will be under the app folder. This file contains a:



@Injectable Decorator

Provider with the root injector

Fetching Data

Syntax:

```
@Injectible({
  provideIn: "root"
})
```

Fetching Data

Add the code for logging to the console to give the output in the browser console in the service class:

Syntax:

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

@Injectible({
   provideIn: 'root',
})
export class LoggerService {
   constructor() {}
   log(msg: string) { console.log(msg); }
   error(msg:sring) { console.error(msg);
```

Fetching Data

Creating a component user.component.ts where this LoggerService can be used

Syntax:

```
import { Component, OnInit} from '@angular/core';
import { User } from './user.model';
import { LoggerService } from '../services/longer.service';

@Component({
   selector: 'app-user',
   templateUrl: './',
   styleUrls: ['./user.component.css',
})
export class UserComponent implements OnInit{
   users: User[];
   constructing(private loggerService: LoggerService) {
     //Adding user instance in the users array
         this.users = [
             new Users('John', 27, new Date('2018-03-25')),
             new Users('Jack', 22, new Date('2020-05-09')),
             new Users('Dave', 28, new Date('2019-10-21')),
         ];
         this.loggerService.log('Total Users: $(this.users.length)');}
ngOnInit(): void { }
```

Key Takeaways

- In SPA, all application's functions exist within a single HTML web page.
- The RouterLink directive is used to navigate to the route in the application and creates a link like an HTML anchor tag.
- The Hypertext Transfer Protocol is used for transmitting hypermedia documents like HTML or data over the network.
- CORS acts as a communication medium between the server and a client application for resource exchange.
- Async Pipe is used to subscribe to an Observable or a Promise and return the latest value from them.

TECHNOLOGY

Thank You

simplilearn