**Angular 13**

Interpolation (1-demo-app)

1. Used to display dynamic data on a website using ‘{[]}’ (double curly braces).
2. Can be used to perform arithmetic, relational operations inside the braces.
3. Cannot be used to create a new object, to find the type of the object, for assignment etc.

Angular CLI and Important Commands

1. CLI commands can be found @ [Angular - CLI Overview and Command Reference](https://angular.io/cli).
2. CLI can be used to create, build, run and deploy the application.
3. Can also be used to create class, interface, routes etc.
4. Can reduce bugs while creating the above without CLI.
5. To generate:
   1. ng generate class Dummy
   2. ng generate component component-name
   3. ng generate module module-name
   4. ng generate component module-name/component-name
   5. ng generate service service-name
   6. ng generate service module-name/service-name
6. To Generate build:
   1. ng build
   2. creates a ‘dist’ folder inside the project root folder which can then be used to deploy the project.

Components

1. Components are the building block to develop a specific feature or functionality
2. Components after creation can then be added to the app.component.html file to be used or rendered.
3. Components are added using ‘<app-component-name>’.
4. We can change the component name if we change the component selector inside the .component.ts file.
5. We can also change the name of the html and css file for the same component using the .component.ts file.
6. Components should be created in a nested format so that editing and maintaining the component can be easy. For e.g., Header component can contain,
   1. The Logo Component
   2. The Search Bar Component
   3. The Menu Component. The Menu can in turn have multiple nested components such as the profile component, dropdown component or other such components.
7. Nested Components makes it easier to manipulate the positioning, styling and editing the components.

Components with InLine

1. Three Cases for Components with InLine:
   1. InLine Style : Used when css required for the component are few.
   2. InLine Template : Used when html component is not required or used inline.
   3. InLine Style & InLine Template (Most Popular) : Used when html and css component are returned inline.
2. InLine Style
   1. ng g c component-name –inline-style
   2. Will not generate the css component file in the new component.
   3. Style for the components in such case can be added by mentioning the class for the tags inside the html component and defining the css inside the component-name.ts file using the class name.
   4. To reference the class for the html tag we must use `class-name{css;}` (backtick) characters.
3. InLine Template
   1. ng g c component-name –inline-template
   2. Will not generate the html component.
   3. HTML for the page will be passed inline inside the component-name.component.ts file.
   4. Inside the template use `html-code` (backtick) to write the html.
4. InLine Style & Template
   1. ng g c component-name –inline-style –inline-template
   2. Will not generate the html and css component.
   3. Usage of such is the same as above cases.
5. Need For InLine
   1. When the lines of html are few(2 or 3), it is better not to create a file and use css inline.
   2. The same goes for html inline.
   3. When html and css both are few, it is better not to create files for the same and use inline format.
6. The component is used in the app.component.html the same way described above.

Module (2-modules)

1. Major feature of an angular application.
2. Basically, a collection of components, services, pips etc.
3. Modules contain a group of features and functionalities which are related to each other.

Graphical user interface, application

Description automatically generated

1. For e.g., User module can have multiple components such as Login, Registration, Forgot Password etc. It can also have services such as API calls. And can also have other things related to a User.
2. User defined modules must be imported inside the ‘app.module.ts’ before they can be used. Can be imported using:
   1. Import {module-reference} from ‘./module-name/module-name.module’
   2. module-reference refers the name which will be used to call the module components. The import style is same as python import style i.e., <import as name-to-be-used from module-name>.
3. Imported module reference name then must be added to the imports array.
4. To use the component inside this module we must first export the component.
5. This can be accomplished by adding the exports array to the .module.ts file of the module component that is to be used.
6. The module component can be called using the same format described in the component section. The name to be used to call the component can be found inside the .module.ts file of the module.
7. This helps in creating services, routes and helpers etc, which then need to follow the same rules above to be used inside the application.

Buttons (3-buttons)

1. Add a button to the app.component.html
   1. Give the button a display name inside the button tag.
   2. Inside the button body use “(click)=function\_name()” to call the function to be used.
   3. The function then must be created inside the app.component.ts file inside the class.
   4. Make sure the class is exported otherwise the function will not be discovered outside the class.
2. Adding mouseover, mouseleave, keyup, keydown and other such events to the button
   1. These are events that can be added to the body of a tag.
   2. Such features are added to the ag body inside ‘(event)’and can be set equal to a function inside double quotes e.g., <(event) = “function()”>.
   3. Some such events are :
      1. mouseover, mouseleave, mousemove etc.
      2. keyup, keydown, blue, input
      3. click
3. Multiple events can be used inside a single tag.
4. Getting Text Box Value from HTML Input. There are two ways to do this
   1. Get value on keyup
      1. By making an input tag with id, name, placeholder and adding a function to the keyup event we can get the value of the input field.
      2. This value can then be passed to the function and can then be displayed to the screen or the console.
      3. The value of the text filed will be passed by using the id.text-field\_property. Such as box.value.
   2. Get value on button click
      1. The input tag will the defined as above but without the event.
      2. The button will be defined and will contain the click event. This event will call the function and pass the value to the function.
      3. The value of the text filed will be passed by using the id.text-field\_property. Such as box.value.
5. Multiple different properties can be passed to the function inside the button. The attribute properties defined inside the input field can be called inside the button event using “id.(dot)attribute-property”.
   1. E.g., id.value, id.placeholder, id.type etc.
6. There are other ways of getting the value also, but this is a simple way. More complex ways will be discussed further.

Counter (4-counter)

1. Used for increment and decrement. In this tutorial buttons are used to achieve this.
2. Create two buttons in the application. Add a click event to both calling the same function with two different parameters. One param is ‘minus’ the other being ‘plus’.
3. Define variable inside the app.component.ts file to store the count value, say count = 0.
4. Create another function with a parameter to catch the incoming parameter value. Say ‘type : string’.
5. Use if condition to check the check the against the parameter and increment or decrement the value of count.
6. Use interpolation to display the value of count on the page.

Basic Styling (5-basic-style)

1. Styling for the local component is done inside the component.css file inside a component.
2. Adding the style in this file will apply the style to the component.
3. The rules of css styling are followed. Adding class to a tag will allow the class to be called inside the css file and the style can be applied to it there.
4. To add the style to all the components we must add the style to the style.css file inside the root directory.
5. Style can also be added as tags inside the component.html file by creating a style tag under the tag where the style must apply.
6. Style can also be added inline inside the body of the tag.
7. The preference is given in order of “inline style > internal style> component.css > style.css”.

Property Binding

1. Feature of angular; used to update or change the properties of any input element.
2. Let’s assume we want to change the properties of any tag at runtime. Declaring these properties with interpolation can help us achieve that.
3. But interpolation has a flaw that it does not understand or recognise the Boolean values.
4. As a result, the property does not work the way it was intended.
5. To achieve this, we use property binding.
6. Here we define the property inside ‘[]’ brackets and assign them a value using ‘=’ without the interpolation.
7. Works the same as interpolation but also allows the property to understand Boolean values.

Conditional Statements (6-condition-statement)

1. If conditions can be used to make decisions based on certain conditions.
2. They are used in this tutorial to show or hide the text inside the h1 tag.
3. Define a h1 tag calling a function inside app.component.ts where we use two variable to toggle the sow and hide property.
4. Inside the h1 tag the condition needs to be called using the \* (star).
   1. \*ngIf is used toggle the show and hide.
5. ng-template is used to contain tags which should be shown with a condition.
   1. Condition can be set using binding ngIf in the ng template body.
   2. [ngIf] = “condition”

Switch (7-switch)

1. Switch in Angular is preferentially used inside a div tag. Inside the body define a ngSwitch property.
   1. This property will take the name of the variable to use to switch.
2. The div will contain the tags which will be the switch cases. The case will be defined inside the tag body using \*ngSwitchCase.
   1. The case will be enclosed in single quotes inside the double quotes like “’property’”.
3. Default case is defined using \*ngSwitchDefault.

For Loops (8-loops)

1. Angular for loops and JavaScript for loops are two separate things.
2. Loops inside html documents will use angular for loops.
3. Loops inside JavaScript will use JavaScript for loops.
4. Loops are made using \*ngFor = “let var of array” and the body of the tag uses interpolation to display the var in the loop.
   1. If the array multiple key : value pairs, we use array.key inside the interpolation to display it.
   2. If the array has another array, we then use ngFor with let var array.key and the body contains the var inside interpolation to display it to the page.

Style Binding (9-style-binding)

1. Like property binding style binding is used to make the style of a tag dynamic.
2. To style bind use []
   1. Inside use “style.attribute” like “style.color”.
   2. And assign it to the variable to be used to define the colour.
   3. The variable is initialized inside the app.component.ts.
3. Multiple style binding can be used inside the same tag using the same way described here.

Header (10-header)

1. Create a header component and add it to the app inside the app.component.html.
2. Add it by using the selector for the header component.
3. Define a nav tag inside the html of header component. And add image and unordered list. With anchor tags to different locations.
4. Add the css to the header component inside the header.component.css file.
5. This will create the header and add it to the application.
6. The header will have a margin inside the main application.
7. To remove the margin so that the header could take the whole space add margin 0px to the style.css file inside the project root directory.
8. Calling the header again or as many times will add multiple headers from top down and stack them over each other.

Basic Form (11-basic-form)

1. Form are modules which must be imported inside the app.module.ts file and register them under imports.
2. Form is then created inside the html using form tag. The form tag will take id with #id and will be set equal to a value.
3. The form will have a click event which will send the data to function on button click. The parameter will be “id.value”.
4. Inside the app.module.ts, import NgForm to refer to the data caught by the function.
5. Create the function with parameter to catch the data. Assign the type as NgForm
6. Create a new variable outside the function to store the data. The type will be any and the value will be object “{}”.
7. Set this variable equal to received data.
8. This data can then be displayed using interpolation inside the application.
9. To see refer to the project. Inside app.component.html.

Toggle Element (12-toggle-element)

1. Toggle Basically refers to showing and hiding an element with reference to something. In this we use a button to toggle a tag.
2. Create a Boolean inside app.component.ts which will be used to toggle.
3. Set the Boolean to true.
4. Create a tag inside app html page and assign \*ngIf to this Boolean variable.
5. Create a button and on this button call, call a function.
6. This function will then change the value of the Boolean variable using ! (not).
7. For e.g., “this.variable = !this.variable”.

Bootstrap (13-bootstrap)

1. Library used to make a page responsive i.e., the application will now work on any type of device.
2. To add bootstrap run the command “ng add @ng-bootstrap/ng-bootstrap”.
3. This will update the packages and import the bootstrap module.
4. To get the templated for the use of bootstrap goto <https://ng-bootstrap.github.io/#/home>
5. Adding bootstrap is as simple as copying the component code from the website and adding it to the component.html.
6. The application page will display the same for the bootstrap elements.
7. Bootstrap is a library. As displayed on the official page of bootstrap.
8. Alternatives for bootstrap are Material-UI.

Material UI (14-material-ui)

1. Material-UI is a HTML and CSS library, that is used to enhance the application by adding extra properties to elements.
2. Provides better look and feel to our application.
3. To Learn more goto <https://material.angular.io/>
4. To add a component to the application, use the component guide and add the component.
5. After adding the component, it will be displayed to the application but the css will not be working correctly.
6. This happens because right now it is being called as an html tag to register it with the material ui we must import the component class from the Module.
7. This is done inside the app.component.ts file and it is imported the same as above examples.
8. Material UI makes the application slow hence the use of material ui is always recommended using when there is less amount of content to display or in a mobile application.

TODO Application (15-todo-app)

1. Designing a Todo Application.

Data Passing: Parent to Child (16-data-passing-ptc)

1. Passing Data from parent Component to Child Component.
2. Create a component. This will be the child component. Name its child to make it easier.
3. Add the child component to the main page.
4. Create the data to be sent to the child component
5. Create a function to change the value of this variable let’s say using random.
6. Inside the html page add child component and inside the body create a property and assign the variable to it.
7. Create a button with the click event calling the function to update the value.
8. Display the value inside the child element using interpolation.
9. We can send multiple variables / data inside the same component.
10. We can send multiple tag data using the same click button.

Reusable Component (17-reusable-component)

1. Create a component say user-details.
2. Add the component to the application inside a list to render as a list.
3. The UL list will have a \*nfFor which will call the userDetails for each.
4. userDetails is an array of objects created inside the app.component.ts file with some data inside it.
5. While calling the component add inside the body of the component a property say item which will be equal to the var defined inside the for loop.
6. Now the component will be displayed for each row inside the userDetails.
7. Inside the user-details component.ts file import the Input module and catch the incoming data using this.
8. Inside the same file catch the data using : @Input() item : {name : string, email: string} = {name: '', email: ''};
9. Thie body of the item will change according to the data inside the userDetails array.
10. This can then be printed to the user-details component using interpolation.
11. Additionally apply some css inside the user-details-component.css to add css to the component.
12. The result will be the data sent from parent to child component.

Data Passing : Child to Parent (18-data-passing-ctp)

1. Create a Child Component say child.
2. Create a function inside the app.component.ts and pass item of string as parameters. This function will be passed inside the child component to be called and set the value from the child component.
3. We then add the child component to the app and create an event of updateDataEvent and set it equal to the function we created and inside we pass $event as the parameter.
4. This update data event is then called from the child component.
5. To do this we import two modules, the first is Output and the second EventEmitter.
6. We then use “@Output() updateDataEvent = new EventEmitter<string>()” inside the child-component.ts file.
7. Inside the html of child component, we create a text box and a button with click event which is set equal to the updateDataEvent.
8. This event will have to emit the data. We then say updateDataEvent.emit(), inside the emit we pass the parameter of input-id.value.
9. The value is then emitted to the updateDataEvent inside the child.component.ts file. Which is then returned to the calling function inside the app.component.html file and then the property is used to set the data inside the parent component.

Two Way Binding (19-two-way-binding)

1. Two-way binding is inputting and displaying at the same time.
2. Inside the app.component.ts file create a data variable of type any.
3. Inside the app html file create an input field and create an event inside a property. i.e., use [()].
4. Inside the inner brackets use ngModel and assign it to the variable we created inside the app component.ts file.
5. Create a tag inside the main html file and use interpolation to display the data.
6. The data as inputted in the text box will be display in the interpolation. This is two-way binding.

Template Reference Variable (20-template-ref-variable)

1. Template variable is used to get the tag properties.
2. It can get any property of a tag using the id (defined by #) of any tag on which it is called.
3. This value can then be used inside the app.component.ts file.
4. Other than this we can use DOM or listeners.
5. The type to define the value from this tag is HTMLInputElement.

Basics Of Typescript (21-baisc-typescript)

1. We don’t directly use JavaScript inside angular we use typescript which is a superset of JavaScript.
2. Refer to the app.component.ts file of this project to know more.
3. Variables defined in component.ts files should be defined strictly which helps to make sure that the variable are not inferred to another type when setting them using setters and while making use of them using getters.
4. Basics datatypes in TypeScript are:
   1. Number
   2. String
   3. Boolean
   4. Any (Can Accept any type at runtime)
   5. Define 2 types (achieved using pipe ‘|’, say number | string)
5. Type are also inferred according to the data that is stored inside the variable. Storing a number will make the variable type of number and storing a string in the same variable will make the variable of type string.
6. However, this is not recommended.
7. Defining array or object.
8. Typescripts can be used with react, vue and with core js.

Pipes (22-pipes)

1. Pipes are used to transfer the data from one format to another format.
2. For e.g., to make a string from upper case to lower or vice versa, to update date format, to apply percentage on any number or to apply locale such operation and other like it can make use of the pipe.
3. Predefined pipes can be found in the official documentation of angular : [Angular - Transforming Data Using Pipes](https://angular.io/guide/pipes)
4. The app.component.html page shows the basic usage of pipes.
5. Let’s also make a variable toady inside app.component.ts and assign it Date().
6. Let’s call this inside the html page.
7. We can also define custom pipes. Inside app.component.ts.
8. Params with Pipes
   1. Params can be passed for most pipes but not all pipes.
   2. They are passed using ‘:’ character after the pipes being used.
9. Most time saving or reliable pipe is Json pipe
10. Other pipes can be easily understood by looking @ app.component.html.

Custom Pipes (23-custom-pipes)

1. Custom pipes are created by the command “ng generate pipe pipe-name”.
2. Create a pipe to convert USD to INR using “ng g c pipes/USDINR.
3. This will generate the pipe and will place the pipe inside the pipes folder inside the app folder inside the project’s root directory.
4. Inside the ts file inside the pipes folder use the function and change the type of value received from unknow to the value that will be passed to it and change the type of arguments that will be passed to it.
5. The next step is to call the pipe inside a html tag and the see the result.
6. Papameters can be passed using ‘:’.
7. Multiple parameters can also be passed and can be received inside the arguments in the pipes ts file because it contains all the arguments and can be retrieved using array indexing.

Forms In Angular: Major Topic (24-template-forms)

1. Forms in angular are bit different and this also affects the data flow of the forms.
2. Forms are used to capture the data from the user which can then be sent to the database or used directly.
   1. E.g., Login, signup, post, searching etc.
3. Angular cannot directly connect with the database. JavaScript libraries or frameworks cannot directly connect with the database and thus need API’s.
4. This is a standard procedure.
5. There are two types of forms in angular
   1. Template Driven Forms : Most of the work done inside the component html.
   2. Reactive Forms : Mostly work done inside component class or ts file.
6. Data Flow of Forms : Template to Class to Service to Database and the vice versa for retrieving data from the database and setting it in the forms. Diagram

   Description automatically generated
7. This project refers mostly to template driven forms:
8. These forms are suitable for short forms.
9. Template Driven Forms :-
10. Creating a Simple form inside the app.component.html file. Action property is not requiring in angular.
11. We must define ng template for this form.
12. This is done using the #form = “form”.
13. Define an event for the form, (ngSubmit) = “something”.
14. Something in the above point is a function which must be created inside the app.component.ts file which will receive the data from the field. For the moment, let’s say that the data type will be any.
15. And pass the id.value of the forms inside the something function.
16. When we runt his code it shows that the code works and a warning is generated but it is empty, that is because we have not yet bound the input fields to the form yet.
17. The fields can be bound to the form using ngModel.
18. Now the code works fine, and the data is being received by the component.ts file and the warning is shown in the console.
19. As long we are passing the id.value of the form to the button as an event. It will not matter whether the button is placed inside the form or not.
20. Name field is very important for binding he data to the form. Without this property inside a input field it will now work.

Template Form Validations (25-template-form-validation)

1. Create a form to add validations to. For reference the form from the last video has been used here.
2. Inside the text box adding required makes sure that the field is required or is essential to the form.
3. The next step is to add an id to the text field which is set equal to the ngModel. What this does is, it provides with the fields with a few properties which can be used for validations.
4. Create a span with some text for validation and inside the span using \*ngIf we can specify the conditions on which this span should be shown.
5. Here we can provide id.invalid and id.touched i.e., whenever the field is touched but not filled the span will be shown and the second will be shown when the field is invalid, which it is until it is filled.
6. Do the same for the password field. The minimum and maximum length for the password field can also be specified inside the password input tag using minlength and maxlength.
7. We can also disable the button by adding disabled property binding and passing in the formid.invalid as its value.
8. We can also make sure that the user can only enter alphabetic characters inside the name text field by adding the pattern property to the tag. And the value will be set to ‘[a-zA-z]+$’.
9. Custom validations can be applied to the tags using regex inside the pattern property.

Reactive Forms in Angular(26-reactive-forms)

1. The First thing is to make a simple form without any special property inside the app.component.html file.
2. The next step is to import Reactive Form Module inside the app.module.ts file.
3. The next step is to import two more modules inside app.component.ts file so that we can make use of the reactive forms. The two are FormGroup and FormControl.
   1. FormGroup will control the entire form, like validations, submit etc.
   2. FormControl will specifically handle a single input field.
4. So, depending on the number of inputs the number of form control are created inside the class in app.component.ts.
5. First, we assign a name for the form let’s say loginForm = new FormGroup({something}).
6. Inside it we must send an object and inside this object we will send our form controls.
7. The form controls are created by creating a var for each input field creating a new FormControl object for each. As in the projects.
8. We can also pass the default values inside the FormControl instance.
9. After this we can add this loginForm to the form using the formGroup property binding. And add formControlName for both the fields so that they can be identified inside the app.component.ts file.
10. We can use ngSubmit event inside the form body to give it a function to call to submit the values of the input fields.
11. Inside the function we can warn and print loginForm which gives us all the fields and properties associated with it, but we can get the values by using loginForm.value.

Reactive Forms: Validation (27-reactive-validation)

1. Here we will continue to add validation to the form we created in the previous video.
2. Validators are needed to validate the input fields for reactive forms. This can be achieved by importing Validators module inside the app.component.ts file.
3. These can then be added to the FormControl as parameters like, “username : new FormControl('',[Validators.required])”.
4. This will add validation to the field that it is created for but does not allow us to create a tag to display if the validations are not being met inside the html component.
5. To do that we first have to create getters for the fields that we want to check against.
6. The getter function will be created with the name of the variable which defined the field we want to show the message for.
7. Then next part is to create a span inside the html page with \*ngIf pointing to this getter function.
8. Adding .invalid and .touched with && will help us to make sure that this message is only shows when the field is focussed on, and the value is not inputted.
9. Also let’s say that if we want to make sure that only email can be put inside the input field, we add Validator.email inside the field in point 3.
10. We can also disable the button from the id of the form as when the field will be invalid so will be the form.
11. Hence using [disabled] on the button and passing the form id.invalid will help us disable the button.
12. Same thing goes for the password field. We can also add min length using Validators.minLength(length), by replacing the length we can specify the minimum length for the password field.
13. The span for the password field can then be added to the html page and will be rendered when the password field is invalid.
14. Validators can also be used to enter patterns like in the previous tutorial.

Directives (28-angular-directives)

1. Properties like \*ngIf or \*ngFor or \*ngSwitch are referred to as directives.
2. Directives are the classed that can provide some additional feature to html elements.
3. They may override style, change text, display elements multiple time, toggle elements.
4. To create directives, we can use angular cli.
5. To create use “ng g directive directive-name”.
6. We can include the directive inside the tag body using the directive selector, but it will not work yet because we have not defined the directive body.
7. To do so go to the directive file created by the generate command.
8. There import ElementRef and pass the element inside the constructor with type ElementRef.
9. Then inside the constructor assign the style for this element as red.
10. We can use this directive and apply it to other tags as well for e.g., p tag, label tag etc.
11. This is the simplest example for directives.

Routing (29-routing)

1. Create a new angular application and select to add routing in the next prompt. Select to add CSS.
2. Next is to create two components say, home, user and about components.
3. Open app.component.html.
4. Open app-routing.module.ts file. Inside this we will use the Routes array.
5. Inside the array we will create an object for each route.
6. Route will be defined in the following way : “{ component : AboutComponent , path : ‘about’}”.
7. This should work if the same as <http://localhost:4200/about>.
8. Add the same way for other components as well.
9. After adding the routes create links in app.component.html file to get move around with these links. Use the format <a routerLink="">Home</a>
10. By default, the router link for home should be empty hence there is nothing here but for other routes use the path defined before. The name should be the same otherwise it will not work.
11. We can change the component html to render what ever we want based on what we want to route to.
12. We can also pass data using the URL to other pages which can be done using :id in the path of the component in routing file.

Dynamic Routing (30-dynamic-routing)

1. Dynamic routing is used when we want to show the information or data coming from database.
2. Create the dynamic part where we give routeLink with id.
3. Add id param to the path for the user path in app-routing.module.ts.
4. Inside the user component’s ts file import Active Route module and inside the constructor give “private route:ActivatedRoute”. And print this value in the console using warn.
5. We can also create a variable inside the component ts file and assign this value to it and can then display it on the html page using interpolation.

404 Error Page (31-404-page)

1. 404 page is displayed when a user hits any invalid URL.
2. Using the same template from the previous tutorial.
3. Add a component for 404 page.
4. Add the component to the routing file and the path will be “\*\*”.
5. This tells that whenever a page that is not in the routes is called the page served should be this one.
6. Works in the current project file.

Child Routes (32-child-routes)

1. Using the template from the last project we will add child routes to them.
2. Child routes basically refer to the routes / pages inside the components.
3. Child routes are added to this routes file inside the parent route using the keyword children which is an array that contains the component and path associated to the child component.
4. The next step is to create links inside the child component to display the link and routes.
5. We must also define router-outlet to show the page loaded by the router inside the router outlet.
6. We can then use the about link on the home page and go to the about page and from there we can go the child components using the links created inside the homepage.

Services in Angular (33-services)

1. Services basically are classes that may have data, properties or functions and we can use these in different files.
2. For e.g., When we need to share data between two components say Home and User pages, we can use services here.
3. Services are in such ways helpers that can share the same data between components.
4. Services are neither component nor module dependent.
5. Services are created using angular CLI and are usually created inside a folder.
6. Services can then have some data or can be called call API to get some data to return it.
7. They then must be imported inside the .component.ts files of the components they need to be used in.
8. We can then create a variable and use the constructor passing the Service as parameter and assigning the data of this service to this variable.
9. This variable can then be used using \*ngFor and interpolation to display the data to the html pages.
10. Services are independent and used to call API’s.

Calling API in angular Applications (34-api-calling)

1. Creating a new project with a service which will call an api.
2. Import HttpClient inside the service.ts file.
3. Initialize it inside the constructor.
4. And return the data using a function.
5. We then must add the same module to the app.module.ts file and register it.
6. We can then import the service inside the component.ts file to use it inside that component.
7. Check the project to see how the code works.

Post Data in Angular (25-api-post)

1. Create a form to get the data to post.
2. Get the values from the form and send them to the function inside the services.
3. Inside this function using the http create a post request which takes two parameters. The Post URI and the data.
4. The data will be posted. For reference go through the project.