

Angular Material : first, Angular REFRESH

Angular Refresh:

one single HTML page (index.html) which is continuously re-rendered by Angular. Angular is the app that I deployed to the server

ng serve: launch a server that host the app. All the content is rendered dynamically through javascript (view source of the page, you will see on the button js scripts, the CLI inject them in the build process)

Decorator: Add metadata

main.ts: the first file that runs (it's in ts but it will compile to js)

@NgModule: Decorator that turn a class into a module. It has some backbone declarations! All the components I'm using in this module

imports: import other modules that add more functionality to my module

providers: for services

bootstrap: must exist only in the root module but not in others

Component: A ts class, may be empty, that has the @Component decorator

(click) = "blabla" : Event Binding ()

not a string, it's TS code, a ref to a function or a statement/assignment

event binding let you react for events created by the user

Property Binding []

property: Attribute of an object. Which object?

Every HTML element has underlying DOM element which is just a JS object with properties. Ex:

the underlying object of <button> has a property disabled

Using <button disabled> just sets the property's value, it's static

but if I want to set it dynamically I must use property binding

<button [disabled] = "isDisabled"

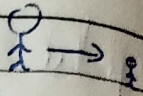
a property in the ts class

Two Way Binding []

You need to import FormsModule and add it to the list of imports for it to work

#used on inputs typically. You bind to the ngModel directive. The purpose is to render the text inside the input dynamically, it binds the text inside the input to a variable and update it which each KeyStroke

Html: [(ngModel)] = "var"

@Input: Communication between Components 
use it before a property to class, that comes from outside. ex:
Input is used to receive data from the outside

Child Component (TS)

`@Input() product: string;`

Here

(HTML) Parent Component
`<app-child [product]="var">`
inject this

@Output:

Emitting an event in the child and reacting to it at the parent component for doing so, in the child component I need 3 ingredients:

- 1) `@Output()`
 - 2) `EventEmitter` so I can launch an event
 - 3) method to emit the event
- to be able to pass the event to the outside → `@Output()`

Child Component (TS)

- 1) `@Output event = new EventEmitter();`
- 2) `onEmit() { this.event.emit(); }`

Parent Component

`<app-child (event)="react20">`

Forms

A more powerful way to handle inputs. To use the Template Form → import `FormModule`
Angular's Forms vs HTML Forms

- Action: is removed, we don't want it. We use instead `(ngSubmit)=""`
- button: type: submit. It will submit the form and send request to the server. We don't want this. Angular will take care of it, it will detect the `<form>` and manage everything behind the scenes:
 - 1) it will prevent the submission
 - 2) manage the validity/state of the form
- every input need to be registered so Angular will be aware of it. Done with `<input ... ngModel>`. Angular will keep track of it and its value
- input name: `<input name="">` to identify the input cause can be more than 1.

Behind the scene

Angular creates a JS object storing all the info about the form. You can get access to this info by placing and `id` and using the following syntax
`<form (ngSubmit)=" " #identifier="ngForm">`

Form Object

By passing the form ID (see **previous** page) to the method that is referenced in my Submit, I can get access to the Form object in the TS class

is the form valid? `check if (form.valid)`

wanna get access to a value? `form.value.name`

`onClick(form)`

→ `<input name="name">`

name of the control in the template

Services

A TS class that can be injected to every component and can be used by it. After injecting the service Angular will understand that it needs to provide an instance of the service when the component is generated. But you need also to tell Angular where/how to look/search. This can be done by:

1) including the service class in `app.module.ts` → `providers: []`

2) using Annotation on the service class → `@Injectable`

3) inside the `@Component` → `providers: []` annotation. It will be provided only to this component

RxJS: Subjects

Data changes, to reflect it in my app without RxJS I would probably need to make new calls to the API every second to see if something is changed.

With RxJS I can use Subject and Subscription

Subscription to a subject ~~is~~ give me updates on any changes, will inform about.

~~To the subscribe function has as an arg the emitted data~~

To the subscribe method pass a function

`subscribe (data emitted) => code to execute with each trigger/change`

After subscribing it's a good practice to unsubscribe on destroy. It clears resources and prevent memory leaks

Router

Our angular app can have multiple pages (illusion) even though only 1 page is sent from the server. The other pages are just simulated

For routing to a component, ~~it doesn't~~ we don't use its selector `<app-comp>`

where to render the component? → `<router-outlet>`