Create project on git bash go to the directory you want it to be created and run

ng new project\_name

Import project to IntelliJ

File -> New Project from existing sources -> select the path from created project

Add Project to Github

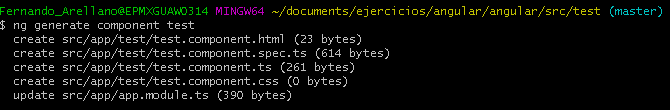
On Intellij VCS-> Import into Version Control -> share project on Github -> fill data and share

Add project into SourceTree

New Tab clicking -> Add -> enter the path where the project is locally

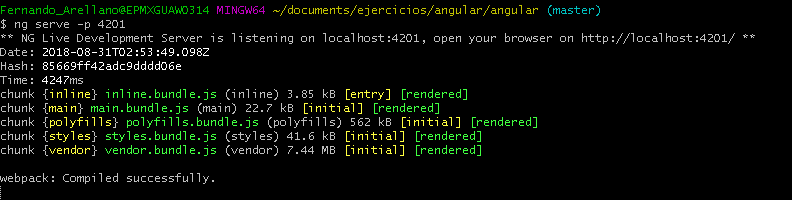
New component

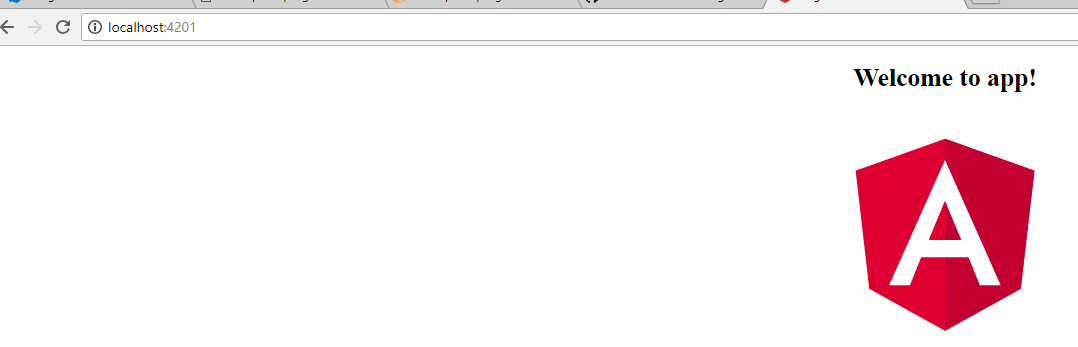
On git bash where the project is located ng generate component component\_name



Run Project

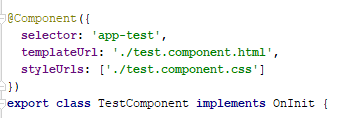
Ng serve -p port\_number





Test created component on app

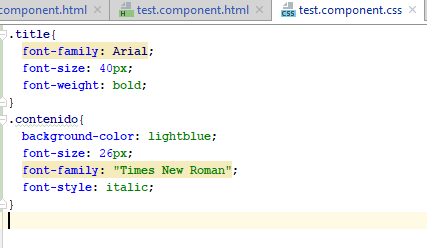
Open app component.html, tak the selector from the created component on the ts file and add it on the app html

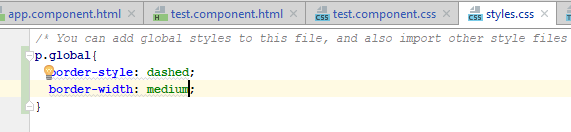


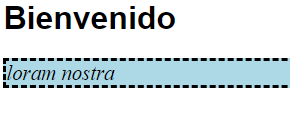


Add Css style to the component

On the css file, normally on the one for the component or there is also another global css







PRACTICA 2

Declare array on Ts

Arreglo: Array<string> = new Array();

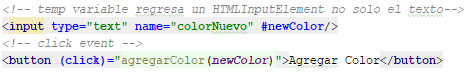
ngFor:



ngIf:

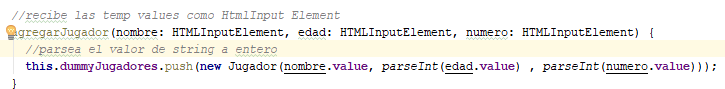


Temp Var:

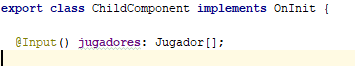




Parse values on tss:

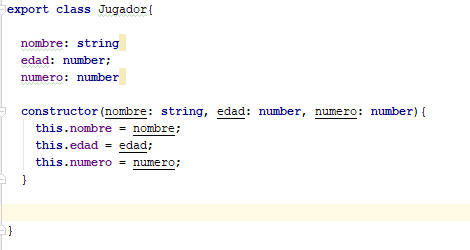


Pass value from parent component to child:



When a new jugador is added its updated and send to the child

Model example:



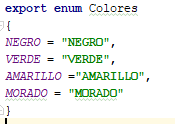
Writing property on String:

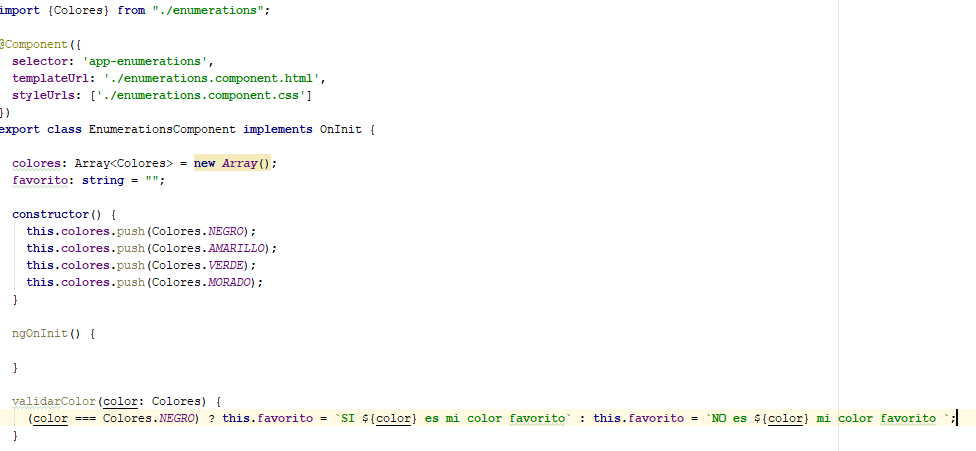




Enum

Import enum class.



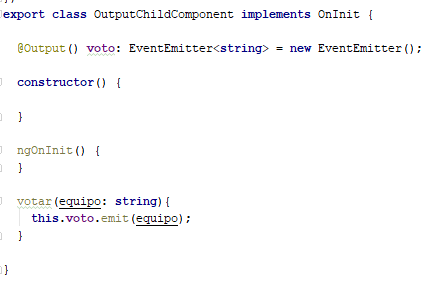


@Output, Event Emitter.

Sends an event from the child to the parent to pass or send values from child to parent



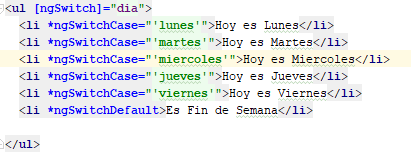
“When there is an event emitted for the voto var on the child component, execute the sumarVoto() method from the parent, pass the value to the parent



Voto property is set as event emitter and when it emits, then the sumarVoto event is executed.

ngSwitch

Choose from few options according from a var





ngClass

adds a class to an element If the condition is true



Adds class resaltarGanador if resaltar is true

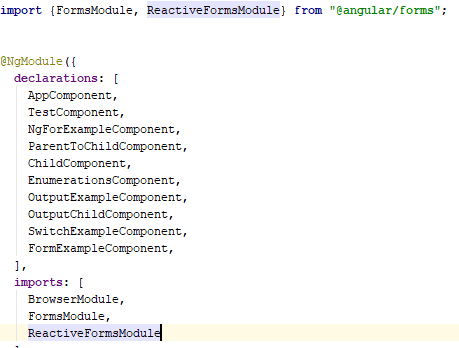
Forms

A FormControl represents a single input field - it is the smallest unit of an Angular form.





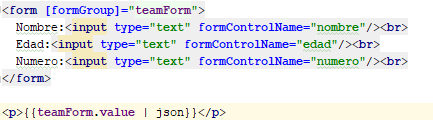
Added formcontrol dayControl and linked It to the input. dayControl is a formControl, to see its value use the value property



Needs to add the import of FormsModule and ReactiveFormsModule in order to use forms in the project

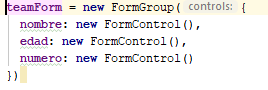
Form Group

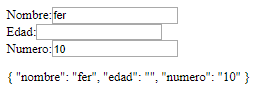
Most forms have more than one field, so we need a way to manage multiple FormControls



Printing team form value as json, if not entered will be null

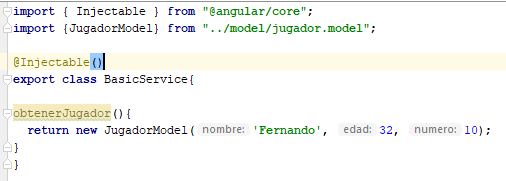
formGroup is linked to the teamForm FormGroup from ts file





Basic Service

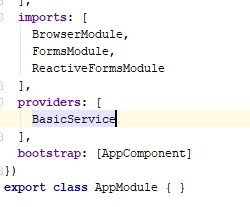
Create ts file for the Service and annotate it with @Inject()



Inject service on the class where its going to be used on the constructor.

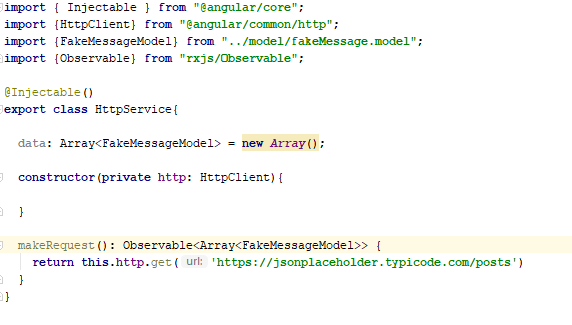


Add service on the providers on app module.

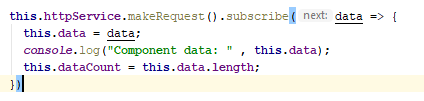


Http Service

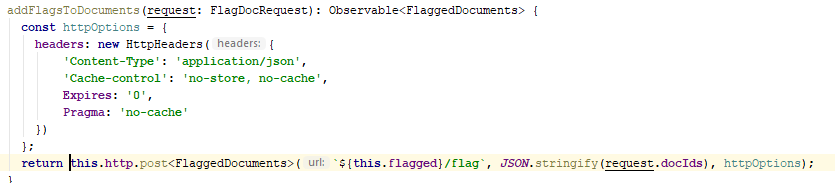
Using the HttpClient from angular/common/http you can make http calls which return and observable. (async calls).



Subscribing to observable and get the data



Call adding header params and body

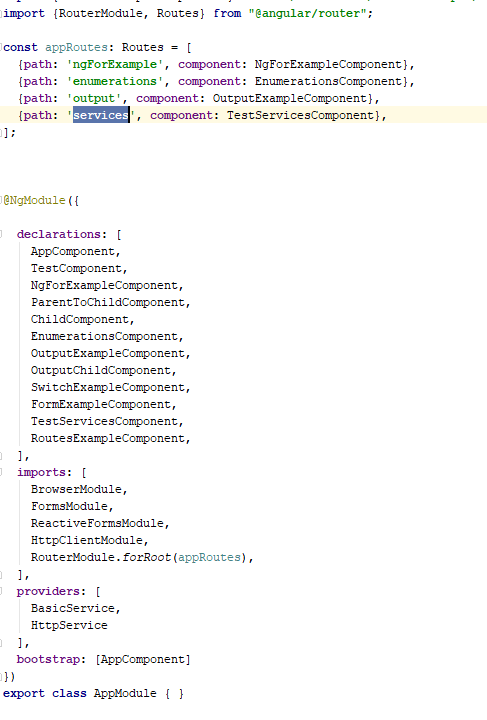


Routes:

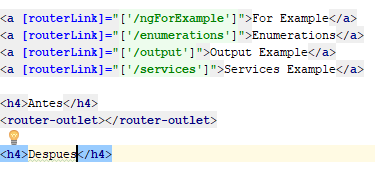
Allows to move among components on different links.

Config.

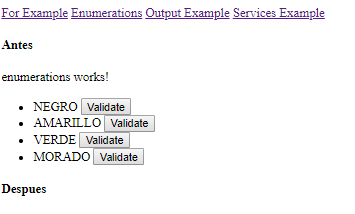
Needs to create the path routes and which component it goes to. Then import the routes.



Then create links which go to the created routes.



<**a [routerLink]="['/ngForExample']"**>For Example</**a**> creates a link that goes to the path ngForExample, from routes and goes to the corresponding component, then it replaces the <**router-outlet**></**router-outlet**> with the content of the component



Path Param,

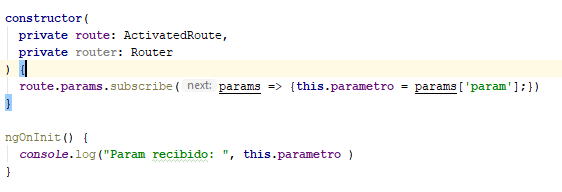
You can get path params from the link and use it on the component.





Declare the path param in the <a> with router link and modify the path of the route to specify it might have a param.

Then on the component you can go through the params received.



Print on console dummyParam (from the routerlink). Router type allows you to be redirected to a component and we could send the params we received.

39 this.router.navigate(['search'], { queryParams: { query: query } })

Observers

Allows to listen on a variable and execute a function when there are changes on it.



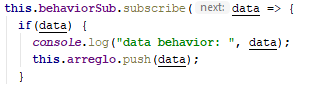
Subscribe to the service will return a observer, we will subscribe to it and when we get the data from the service then it will execute the function which is assigning the data received to the local var

Behavior Subject

Similar to observer allows to trigger the function sending a new value for the subject with the next value (it will replace the actual value of the behavior subject for the one sent on the parameter of next).



Initial value of the subject will be null.



Start the subscription on ng oninit so it will listen for changes.



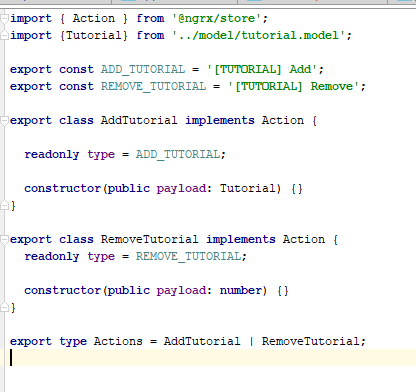
When fillSubject method is called then a new value for behaviorSub is sent which triggers the function on the observer.

Ngrx Example

Action: An action in Ngrx/store is two things:

* A type in the form of a string. It describes what's happening.
* It contains an optional payload of data.

What is going to happen with it depends on the reducer.



Consts used as strings for the action type.

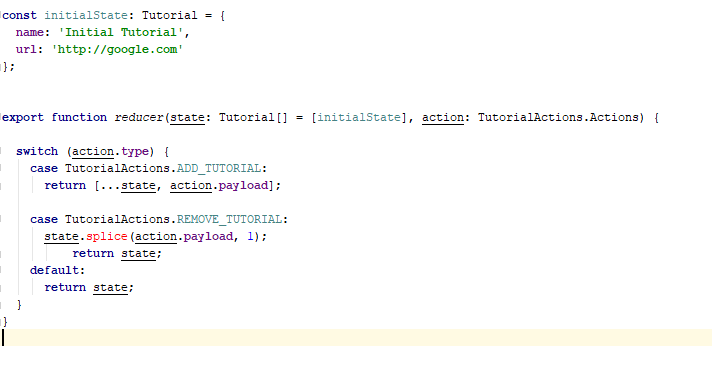
Export available actions which have to specify its type and a constructor (args are optional)

Export type actions then when imported in other file both actions will be available



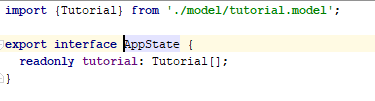


Reducer: A reducer is what takes the incoming action and decides what to do with it. It takes the previous state and returns a new state based on the given action.



Can have an initial state which will be return in case that reducer is selected before any action is executed.

State: state of the application, this will be return by the reducer and made available on the store.

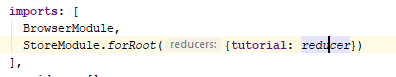


Store is injected to the component with generic type of the wanted state



Config:

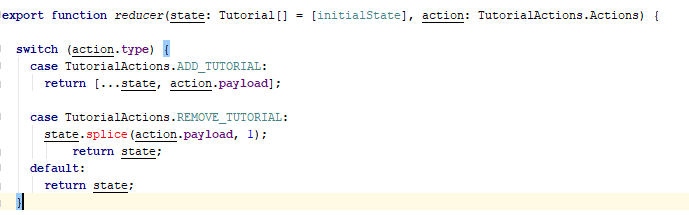




Import the reducers on app module. (name convention, tutorial.reducer.ts would be the file for the tutorial reducer added on the store module for root above.) tutorial should be selected on the store in order to call that reducer and get the reducer from the store:



Read from the store flow:



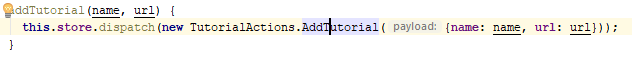
At start reducer is called, it will return the default value of the reducer (will send back the init value of the state)



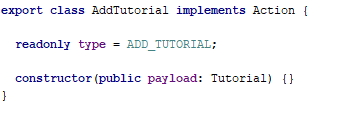
Component constructor calls the reducer tutorial with select, to retrieve info from the store.

Write to the store flow:

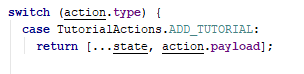
Component method called and it dispatch to the store



On the argument an new action is sent as argument, so it goes to actions to create that action type AddTutorial:

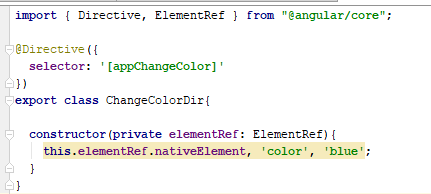


Then the reducer is called and will execute the according method depending on the action type:



New Directive

Annotate it with the @Directive and add a selector (this will be the property used in the components to use the directive ex: \*ngIf)



Use the selector on the component’s html in order to use it

