JAVASCRIPT, PUR SUPERANDO I LIMITI HTML, HA DELLE FRAGILITA' : E' UNTYPED E NON HA SVILUPPATO APPIENO IL PARADIGMA DELLA OOP.

INFATTI NON CONSENTE LA CREZIONE DI INTERFACCE, E CONSEGUENTEMENTE NON E'POSSIBILE APPLICARE POLIMORFISMO E IMPLEMENTARE IL PATTERN LOOSE COUPLING.

TALI LIMITI SONO STATI SUPERATI DA TYPESCRIPT, CHE E' UN PLUGIN E QUINDI UNA ESTENSIONE DI JAVASCRIPT.

TYPESCRIPT MANTIENE TUTTE LE CARATTERISTICHE DI JAVASCRIPT E NE SUPERA I LIMITI.

TYPESCRIPT E' UN LINGUAGGIO EVENT DRIVEN, TYPED, ADERENTE AI DUE PARADIGMI OOP (MOLTO DI PIU' RISPETTO A JAVASCRIPT) E FUNZIONALE. TYPESCRIPT CONSENTE DI IMPLEMETARE POLIMORFISMO E LOOSE COUPLING, INFATTI CONSENTE DI CREARE INTERFACCE.

TYPESCRIPT E' UN LINGUAGGIO TRANSPILATO, OVVERO COMPILATO E TRADOTTO IN JAVASCRIPT NATIVO: INFATTI NON ESISTE IN NESSUN BROWSER UN TYPESCRIPT ENGINE, MA SOLO IL JAVASCRIPT ENGINE, PER CUI TYPESCRIPT DEVE ESSERE TRANLATO IN LINGUAGGIO JAVASCRIPT NATIVO CHE VERRA' POI ESEGUITO NEL BROWSER. TYPESCRIPT VIENE TRADOTTO IN JAVASCRIPT DA NODE.JS, TRANSPILER E RUNTIME ENVIRONMENT PER LE APPLICAZIONI ANGULAR CHE USANO COME LINGUAGGIO NATIVO PROPRIO TYPESCRIPT.

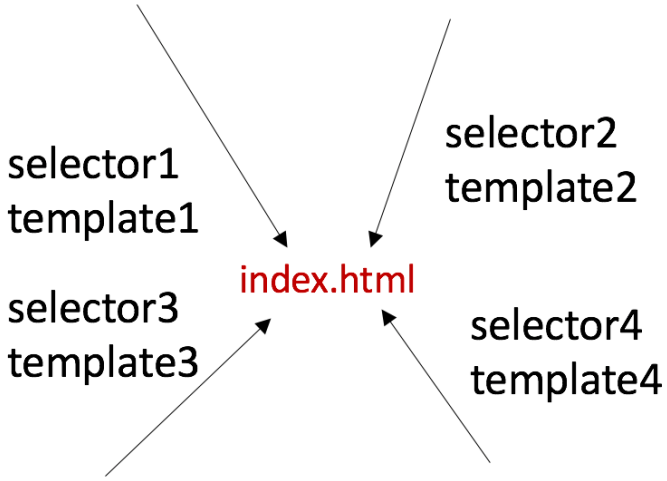
ANGULAR E' UN FRAMEWORK MVC (PROGETTO GOOGLE MICROSOFT) BASATO SU LINGUAGGI COFFESCRIPT E TYPESCRIPT, E IN QUANTO TALE FORNISCE UNA SERIE DI COMPONENTI APPLICATIVE SPECIFICHE COME LA COMPONENT, IL SERVICE E LA PIPE CHE NECESSITANO PER ESSERE ESEGUITE DEL RUNTIME ENVIRONMENT NODE.JS.

ANGULAR E' UN FRAMEWORK MVC IN QUANTO IMPLEMENTA IL PATTERN MVC ASSEGNANDO UNA RESPONSABILITA' ALLE COMPONENTI APPLICATIVE DI PROGETTO:

- **CLASSI MODEL : CLASSI JAVASCRIPT STANDARD, CHE GESTISCONO I DATI (TRAMITE VARIABILI DI ISTANZA E COSTRUTTORE)-→>>RAPPRESENTANO IL MODEL DEL PATTERN MVC**
- **COMPONENT : HANNO 3 "FACCE": CLASSE TYPESCRIPT DECORATA CON DECORATOR @Component; TEMPLATE HTML PER IL RENDERING DELLE INFORMAZIONI; FILE CSS PER LO STYLING. AD OGNI COMPONENT E' ASSOCIATO UN ALIAS CHIAMATO SELECTOR CHE SERVE PER COMPORRE L'APPLICAZIONE ALL'INTERNO DEL FILE index.html -→>>RAPPRESENTANO LA VIEW DEL PATTERN MVC**
- **SERVICE : SONO CLASSI TYPESCRIPT SPECIALI DECORATE CON DECORATOR @Injectable E INIETTABILI NELLE COMPONENT TRAMITE DEPENDENCY INJECTION. ALLE CLASSI SERVICE DEVE ESSERE DEMANDATA LA BUSINESS LOGIC, IN PARTICOLARE CONTROLLI E INTERAZIONI CON APPLICAZIONI DI TERZE PARTI, COME AD ESEMPIO I WEB SERVICE.**

- **PIPE : SONO COMPONENTI TYPESCRIPT PREDEFINITE E CUSTOMIZZABILI, A CUI DEVE ESSERE DEMANDATA LA LOGICA DI TRASFORMAZIONE E CONVERSIONE DEI DATI→>RAPPRESENTANO, INSIEME AI SERVICE, IL CONTROLLER DEL PATTERN MVC.**

LE APPLICAZIONI ANGULAR SONO SPA APPLICATIONS.



L'UNICA PAGINA WEB DELL'APPLICAZIONE E' L'index.html, IN CUI VENGONO INIETTATI I SELECTOR CORRISPONDENTI ALLE COMPONENT, PER FORMARE UN PUZZLE DI TEMPLATE CARICATI TUTTI NELLA STESSA PAGINA. I TEMPLATE SONO PERTANTO PORZIONI DI DOM.

STRUTTURA APPLICAZIONI ANGULAR 2-16

L'APPLICAZIONE E' DIVISA IN MODULI.
UN MODULO E' OBBLIGATORIO: l' AppModule
(CHE CONTIENE LA COMPONENT AppComponent).
ALTRI POSSONO ESSERE EVENTUALMENTE AGGIUNTI.

OGNI MODULO E' UN INSIEME DI COMPONENT, SERVICE E PIPE.

LE COMPONENT E LE PIPE DEVONO ESSERE ELENCATE NELLA VOCE declarations DEL MODULO.

OGNI MODULO PUO' ESPORTARE E IMPORTARE ALTRI MODULI

DEVE ESSERE SEMPRE PRESENTE NELLA STRUTTURA DEL PROGETTO IL FILE package.json, CHE CONTIENE LA LISTA DELLE DIPENDENZE npm (PACKAGE MANAGER FRONT END).

DEVE ESSERE SEMPRE PRESENTE NELLA STRUTTURA DEL PROGETTO IL FILE package-lock.json, CHE CONTIENE LA LISTA DELLE DIPENDENZE npm (PACKAGE MANAGER FRONT END) e IL LINK DELL'NPM REGISTRY DAL QUALE SONO STATE INSTALLATE

DEVE ESSERE PRESENTE LA CARTELLA node_modules NELLA QUALE VENGONO FISICAMENTE COLLOCATE LE DIPENDENZE NPM

DEVE ESSERE PRESENTE IL FILE angular.json NEL QUALE SONO FATTE LE CONFIGURAZIONI PROGETTUALI COME LA PORTA SULLA QUALE VIENE ESEGUITA L'APPLICAZIONE ANGULAR, INTERNZAZIONALIZZAZIONE E COSI' VIA

STRUTTURA APPLICAZIONI ANGULAR 17-18

E' POSSIBILE CREARE MODULI MA NON OBBLIGATORIO

LE COMPONENT POSSONO ESSERE STANDALONE, OVVERO NON INCLUSE NEI MODULI, COSI' COME I SERVICE E LE PIPE

OGNI COMPONENT PUO' IMPORTARE AUTONOMAMENTE MODULI CHE DOVESSERO SERVIRE PER LA SUA IMPLEMENTAZIONE

DEVE ESSERE SEMPRE PRESENTE NELLA STRUTTURA DEL PROGETTO IL FILE package.json, CHE CONTIENE LA LISTA DELLE DIPENDENZE npm (PACKAGE MANAGER FRONT END).

DEVE ESSERE SEMPRE PRESENTE NELLA STRUTTURA DEL PROGETTO IL FILE package-lock.json, CHE CONTIENE LA LISTA DELLE DIPENDENZE
npm (PACKAGE MANAGER FRONT END) e IL LINK DELL'NPM REGISTRY
DAL QUALE SONO STATE INSTALLATE

DEVE ESSERE PRESENTE LA CARTELLA node_modules NELLA QUALE VENGONO FISICAMENTE COLLOCATE LE DIPENDENZE NPM

DEVE ESSERE PRESENTE IL FILE angular.json NEL QUALE SONO FATTE LE CONFIGURAZIONI PROGETTUALI COME LA PORTA SULLA QUALE VIENE ESEGUITA L'APPLICAZIONE ANGULAR, INTERNZAZIONALIZZAZIONE E COSI' VIA

PROGETTO DEMO PRIMO ESEMPIO

ng new demo --standalone false

cartella src/app/model

```
export class Employee {

    firstName: string;

    lastName: string;

    age: number;

    constructor(firstName: string, lastName: string, age: number) {

        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;


    }



}
```

NB: IN UNA CLASSE TYPESCRIPT UNA VARIABILE DI ISTANZA PUO' ESSERE SOLO DICHIARATA NEL CASO IN CUI NE VENGA FATTA ASSEGNAZIONE IN UN COSTRUTTORE. ALTERNATIVAMENTE VA INIZIALIZZATA AL VALORE undefinied, OPPURE VA DICHIARATA COME PARAMETRO OPZIONALE USANDO ? (QUESTION MARK) ; IN QUESTO CASO NON VA ASSEGNATA NEL COSTRUTTORE; ALTERNATIVAMENTE VA USATO IL ! (NOT NULL ASSERTION OPERATOR) INDICANDO IN QUALCHE MODO CHE NEL CORSO DELL'ESECUZIONE DELL'APPLICAZIONE QUELLA VARIABILE NON VARRA' NULL.

cartella src/app/service

ng g s employee

employee.service.ts

```
@Injectable({
  providedIn: 'root'
})
export class EmployeeService {

  employee1: Employee = new Employee("Mario", "Rossi", 34);
  employee2: Employee = new Employee("Giulio", "Verdi", 27);

  employees: Employee[] = [this.employee1, this.employee2];

  constructor() { }

  getEmployees(): Employee[] {

    return this.employees;

  }

}
```

NB: ANGULAR CLI CREA DI DEFAULT I SERVICE CON PARAMETRO providedIn : root. QUESTO VUOL DIRE CHE IL SERVICE E' SINGLETON, OVVERO TUTTE LE COMPONENT NELLE QUALI VIENE INIETTATO IL SERVICE, A QUALUNQUE MODULO DELL'APPLICAZIONE APPARTENGONO, USANO LA STESSA ISTANZA DEL SERVIZIO. IL VALORE DI providedIn PUO' ESSERE SETTATO ANCHE A 'platform', IL CHE VUOL DIRE CHE TTTE LE APPLICAZIONI ANGULATCON LO STESSO CONTEXT ROOT, APPARTENENTI ALLO STESSO PORTALE, USANO LA STESSA ISTANZA DEL SERVICE; UN ALTRO PODDIBILE VALORE DI providedIn è 'any', IL CHE VUOL DIRE CHE OGNI MODULO APPARTENENTE AD UNA STESSA APPLICAZIONE USA UNA DIVERSA ISTANZA DEL SERVIZIO. PER DEFAULT, SE NON VIENE FATTA ALCUNA CONFIGURAZIONE CUSTOM, L'ISTANZA DI UN SERVICE VIENE CARICATA SEMPRE IN MODALITA' EAGER, CIOE' IL SERVIZIO VIENE ISTANZIATO ALL'AVVIO DELL'APPLICAZIONE. E' POSSIBILE CONFIGURARE UN SERVICE ANCHE NELLA VOCE providers DELL'app.module.ts. IN QUESTO CASO, IL COMPORTAMENTO E' LO STESSO DI providedIn:root.

app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { EmployeeService } from './service/employee.service';
import { Employee } from './model/employee';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent implements OnInit {

  employees: Employee[] = [];

  constructor(private employeeService: EmployeeService) {

  }

  ngOnInit(): void {

    this.employees = this.employeeService.getEmployees();

  }


}
```

app.component.html

```
<div>

  <ul>

    <li *ngFor="let employee of employees">{{employee.firstName}}
{{employee.lastName}} {{employee.age}}</li>

  </ul>



</div>
```

NB : LO STRING INTERPOLATION E' UN **ONE-WAY BINDING** DALLA COMPONENT AL TEMPLATE (SI OTTIENECON LE CURLY BRACKETS). IN QUESTO CASO VIENE EFFETTUATO UN INFERT TYPE DELLA

VARIABILE employee. *ngFor E' UNA DIRECTIVE, CIOE' UNA PARLOLA CHIAVE CHE CONTIENE GIA' UNA LOGICA (NEL CASO DI *ngFor LOGICA DI ITERAZIONE).

RIMANERE DENTRO LO STESSO PROGETTO PER FARE UN ESEMPIO DI PARENT-CHILD(@INPUT)

CREARE CARTELLA component sotto app

GENERARE UNA NUOVA COMPONENT CON COMANDO ng

ng g c employees

cambiare il codice di app.component.html in:

```
<app-employees [employees]="employees"></app-employees>
```

CODICE employees.component.ts

```
import { Component, Input, OnInit } from '@angular/core';
import { Employee } from '../../model/employee';

@Component({
  selector: 'app-employees',
  templateUrl: './employees.component.html',
  styleUrl: './employees.component.css'
})
export class EmployeesComponent {

  @Input()
  employees : Employee[] = [];

  constructor(){}


}
```

employees.component.html

```
<div>

    <ul>

        <li *ngFor="let employee of employees">{{employee.firstName}}
{{employee.lastName}} {{employee.age}}</li>

    </ul>


</div>
```

NB: @Input E' UN ONE-WAY BINDING DALLA COMPONENT PADRE ALLA COMPONENT FIGLIA. I PASSAGGI SONO I SEGUENTI: LA COMPONENT PADRE TYPESCRIPT BINDA UNA VARIABILE DI ISTANZA AL SUO TEMPLATE TRAMITE SINTASSI SQUARE BRACKETS. LA VARIABILE CHE SI INSERISCE NELLE SQUARE BRACKETS DEL TEMPLATE DELLA COMPONENT PADRE DEVE ESSERE BINDATA CON LA VARIABILE DI ISTANZA USATA SOTTO L'ANNOTATION @Input DELLA COMPONENT TYPESCRIPT FIGLIA.

RIMANERE DENTRO LO STESSO PROGETTO PER FARE UN ESEMPIO DI PARENT-CHILD(@OUTPUT)

creare una nuova component nella cartella component

ng g c send

codice send.component.html

```
<label for="name">Name:</label>
<input type="text" id="name" #name>
<button type="button" (click)="sendName(name.value)">send</button>
```

codice send.component.ts

```
@Component({
  selector: 'app-send',
  templateUrl: './send.component.html',
  styleUrl: './send.component.css'
})
export class SendComponent {


  @Output() name = new EventEmitter<string>();

  sendName(value: string) {
    this.name.emit(value);
  }

}
```

codice app.component.html

```
<app-send (name)="handleName($event)"></app-send>
<app-employees [employees]="employees"></app-employees>
```

codice app.component.ts

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

```
import { EmployeeService } from './service/employee.service';
import { Employee } from './model/employee.model';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent implements OnInit{

  employees:Employee[]=[];

  constructor(private employeeService:EmployeeService){


  }

  ngOnInit(): void {

    this.employees = this.employeeService.getEmployees();
  }


  handleName(name: string) {

    console.log(name);

  }



}
```

NB: @Output E' UN ONE WAY BINDING DALLA COMPONENT FIGLIA ALLA COMPONENT PADRE. GLI STEP SONO I SEGUENTI: 1) IL PRIMO TRIGGER PARTE DAL TEMPLATE DELLA COMPONENT FIGLIA, PER ESEMPIO TRAMITE UNA TEMPLATE REFERENCE CHE VIENE PASSATA IN INPUT AD UNA FUNZIONE SULLA GENERAZIONE DI UN EVENTO (AD ESEMPIO IL CLICK SU UN BUTTON); 2) SUCCESSIVAMENTE LA TEMPLATE REFERENCE VIENE BINDATA TRAMITE LA ANNOTATION @Output NELLA COMPONENT TYPESCRIPT DELLA COMPONENT FIGLIA, CHE DEVE ESSERE DI TIPO EventEmitter; 3)A QUEL PUNTO LA FUNZIONE DELLA COMPONENT FIGLIA TRIGGERATA DAL TEMPLATE EMETTE UN EVENTO; 4) IL TEMPLATE DELLA COMPONENT PADE RICEVE IN INPUT IL BINDING DELLA VARIABILE DI ISTANZA @Output DELLA COMPONENT FIGLIA, E LA CATCHA CON UNA FUNZIONE CHE RICEVE IN INPUT L'EVENTO EMESSO DALLA COMPONENT FIGLIA; LA FUNZIONE DEVE ESSERE DICHIARATA CON LO STESSO NOME NELLA COMPONENT TYPESCRIPT PADRE.

NB: @ViewChild E’ UN ONE WAY BINDING CHE CONSENTE DI EFFETTUARE L’INJECTION DI VARIABILI DI ISTANZA DA UNA COMPONENT TYPESCRIPT FIGLIA IN UNA COMPONENT TYPESCRIPT PADRE, SENZA PASSARE PER IL TEMPLATE.

creare una nuova component nella cartella component

ng g c welcome

codice welcome.component.ts

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

@Component({
  selector: 'app-welcome',
  templateUrl: './welcome.component.html',
  styleUrl: './welcome.component.css'
})
export class WelcomeComponent {

  public messages:string[]=["welcome", "this is my app"];

}
```

cambiare il codice di app.component.ts

```
import { AfterViewInit, ChangeDetectorRef, Component, OnInit, ViewChild } from
'@angular/core';
import { EmployeeService } from './service/employee.service';
import { Employee } from './model/employee.model';
import { WelcomeComponent } from './component/welcome/welcome.component';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent implements OnInit,AfterViewInit{

  @ViewChild(WelcomeComponent)
  welcomeComponent!:WelcomeComponent;

  message1!:string;
  message2!:string;


  employees:Employee[]=[];
```

```
  constructor(private cdr: ChangeDetectorRef,private
employeeService:EmployeeService){


  }

  ngAfterViewInit(): void {


   this.message1=this.welcomeComponent.messages[0];
   this.message2=this.welcomeComponent.messages[1];

   this.cdr.detectChanges();

  }

  ngOnInit(): void {


    this.employees = this.employeeService.getEmployees();

  }

  handleName(name: string) {

    console.log(name);

  }


}
```

NB: E' IMPORTANTE IMPLEMENTARE IL METODO ngAfterViewInit DELL'INTERFACCIA AfterViewInit, IN QUANTO IL DECORATOR @ViewChild VIENE INTERPRETATO DA Node.js QUANDO VIENE INVOCATO QUESTO METODO. RAPPRESENTA UNA BEST PRACTICE INVOCARE IL METODO detectChanges() ALLA FINE DI ngAfterViewInit PER FORZARE NODE.JS A RILEVARE I CAMBIAMENTI.

cambiare il codice di app.component.html

```
<app-welcome></app-welcome>
{{message1}} {{message2}}
<br><br>
<app-send (name)="handleName($event)"></app-send>
<app-employees [employees]="employees"></app-employees>
```

Esempio di Angular Content Projection (ng-content)

CONTENT PROJECTION E' UN ONE WAY BINDING CHE CONSENTE DI INIETTARE PORZIONI DI CONTENUTO DEL TEMPLATE HTML DI UNA COMPONENT FIGLIA NEL TEMPLATE DELLA COMPONENT PADRE (SI OTTIENE TRAMITE TAG ng-content).

CREARE UNA NUOVA COMPONENT SOTTO IL PERCORSO src/app/component

ng g c title

codice title.component.html

```
<div>

    angular-demo

    <ng-content></ng-content>

</div>
```

cambiare il codice di app.component.html

```
<app-welcome></app-welcome>
{{message1}} {{message2}}
<app-title></app-title>
<br><br>
<app-send (name)="handleName($event)"></app-send>
<app-employees [employees]="employees"></app-employees>
```

RECAP ONE WAY BINDING ANGULAR:

INTERPOLATION: NON CONTESTUALIZZABILE NELLA RELAZIONE DI PARENTELA FRA COMPONENT; E' UN ONE WAY BINDING CHE RIMANE CONFINATO IN UNA COMPONENT. SERVE AD EFFETTUARE IL RENDERING DEL CONTENUTO DELLE VARIABILI DI ISTANZA DICHIARATE IN UNA CLASSE COMPONENT ALL'INTERNO DEL SUO STESSO TEMPLATE. SINTASSI USATA. {{}}.

@Input : SI USA PER INVIARE INFORMAZIONI DA UNA COMPONENT PADRE AD UNA COMPONENT FIGLIA. IL PADRE IDENTIFICA UNA CHIAVE INSERITA NELLE [] PER IDENTIFICARE L'INFORMAZIONE CHE SI DEVE INVIARE. IL FIGLIO RECEPISCE TRAMITE DECORATOR **@Input** MATCHANDO IL NOME DELLA CHIAVE. HA SENSO APPLICARE QUESTO BINDING QUANDO UN PADRE DEVE INVIARE DIVERSE INFORMAZIONI A DIVERSE COMPONENT FIGLIE, QUINDI QUANDO LE COMPONENT NON DEVONO CONDIVIDERE LA STESSA INFORMAZIONE.

@Output : SI USA PER INVIARE INFORMAZIONI DA UNA COMPONENT FIGLIA AD UNA COMPONENT PADRE, AD ESEMPIO IL VALORE DI UN CAMPO DI TESTO. IL FIGLIO IDENTIFICA L'INFORMAZIONE TRAMITE UNA TEMPLATE REFERENCE, CHE RAPPRESENTA IL NOME DELL'EVENTO DA EMETTERE E DA MATCHARE TRAMITE IL DECORATOR @Output. IL FIGLIO DEVE CATCHARE IL NOME DELL'EVENTO NEL SUO TEMPLATE, TRAMITE LA SINTASSI ().

@ViewChild: SI USA QUANDO PIU' COMPONENT PADRE VOGLIONO EREDITARE, CONDIVIDENDOLO, LO STESSO SET DI INFORMAZIONE DA UNA COMPONENT FIGLIA CHE FUNGE DA CENTRALIZZATRICE. E' UN BINDING TRA LA COMPONENT TYPESCRIPT FIGLIA E LA COMPONENT TYPESCRIPT PADRE.

PROJECT CONTENTION: SI USA QUANDO UNO STESSO CONTENUTO HTML E' DA REPLICARE IN PIU' TEMPLATE DELL'APPLICAZIONE. IN TAL CASO E' CONVENIENTE DEFINIRE LA STRUTTURA HTML NEL TEMPLATE FIGLIO E FARLA EREDITARE DA TUTTI I PADRI CHE DEVONO REPLICARE PER L'APPUNTO QUELLA STRUTTURA NEL PROPRIO TEMPLATE. E' UN BINDING TRA TEMPLATE DELLA COMPONENT FIGLIA E TEMPLATE DELLA COMPONENT PADRE.

PIPE

NB: LE PIPE IN ANGULAR SONO DEGLI STRUMENTI SINTATTICI CHE CONSENTONO DI IMPLEMENTARE LOGICHE DI CONVERSIONE E/O TRASFORMAZIONE DATI.

ESISTONO PIPE DI DUE TIPOLOGIE:

- BUILT-IN PIPE (PIPE PREDEFINITE DAL FRAMEWORK ANGULAR)

- CUSTOM PIPE (POSSONO ESSERE CREATE TRAMITE ANGULAR CLI ED ELENCATE NELLA VOCE DECLARATIONS DI UN MODULO COME LE COMPONENT)

LE PIPE, INSIEME AI SERVICE ANGULAR, DEVONO ESSERE RICONDOTTE ALLA BUSINESS LOGIC, E QUINDI RIENTRANO NELL'IMPLEMENTAZIONE DEL CONTROLLER LAYER RISPETTO AL PATTERN MVC.

LE PIPE, COME LE CLASSI MODEL, SONO ISTANZIABILI A MANO E NON SI INIETTANO.

CREAZIONE DI UNA COMPONENT SOTTO PERCORSO src/component per esemplificare le PIPE

ng g c pipejourney

codice pipejourney.component.ts

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

@Component({
  selector: 'app-pipejourney',
  templateUrl: './pipejourney.component.html',
  styleUrl: './pipejourney.component.css'
})
export class PipejourneyComponent {
```

```
  today:number = Date.now();

}
```

codice pipejourney.component.html

```
<p>{{today}}</p>
```

codice app.component.html

```
<app-welcome></app-welcome>
{{message1}} {{message2}}
<app-title></app-title>
<br><br>
<app-send (name)="handleName($event)"></app-send>
<app-employees [employees]="employees"></app-employees>
<app-pipejourney></app-pipejourney>
```

cambiare il codice di pipejourney.component.html in

```
<p>{{today | date:'fullDate'}}</p>
```

cambiare il codice in

```
<p>{{today | date:'dd/MM/yyyy'}}</p>
```

cambiare codice app.component.ts

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

@Component({
  selector: 'app-pipejourney',
  templateUrl: './pipejourney.component.html',
  styleUrl: './pipejourney.component.css'
})
export class PipejourneyComponent {

  today:number = Date.now();
  message:string = "Hello World";

}
```

cambiare il codice di app.pipejourney.html in:

```
<p>{{today | date:'dd/MM/yyyy'}}</p>
```

```
<p>{{message | uppercase}}</p>
<p>{{message | lowercase}}</p>
```

cambiare il codice di pipejourney.component.ts in:

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

@Component({
  selector: 'app-pipejourney',
  templateUrl: './pipejourney.component.html',
  styleUrl: './pipejourney.component.css'
})
export class PipejourneyComponent {
```

cambiare il codice di pipejourney.component.ts in

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

@Component({
  selector: 'app-pipejourney',
  templateUrl: './pipejourney.component.html',
  styleUrl: './pipejourney.component.css'
})
export class PipejourneyComponent {

  today:number = Date.now();
  message:string = "Hello World";
  amount:number = 1234.89;

}
```

cambiare il codice di pipejourney.component.html in

```
<p>{{today | date:'dd/MM/yyyy'}}</p>
<p>{{message | uppercase}}</p>
<p>{{message | lowercase}}</p>
<p>{{amount | currency:'USD'}}</p>
```

cambiare il codice di pipejourney.component.ts

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

@Component({
  selector: 'app-pipejourney',
  templateUrl: './pipejourney.component.html',
  styleUrl: './pipejourney.component.css'
```

```
})
export class PipejourneyComponent {

  today:number = Date.now();
  message:string = "Hello World";
  amount:number = 1234.89;

  person = {

    "firstName" : "Marco",
    "lastName" : "Rossi",
    "age" :21

  }

}
```

cambiare il codice di pipejourney.component.html in:

```
<p>{{today | date:'dd/MM/yyyy'}}</p>
<p>{{message | uppercase}}</p>
<p>{{message | lowercase}}</p>
<p>{{amount | currency:'USD'}}</p>
<p>{{person}}</p>
```

cambiare il codice di pipejourney.component.html in:

```
<p>{{today | date:'dd/MM/yyyy'}}</p>
<p>{{message | uppercase}}</p>
<p>{{message | lowercase}}</p>
<p>{{amount | currency:'USD'}}</p>
<p>{{person | json}}</p>
```

ESEMPIO DI CUSTOM PIPE

creazione di una custom pipe (CREAZIONE DI UNA CARTELLA pipe sotto /src/app)

porsi nella cartella pipe e lanciare il seguente comando:

ng g p custom

scrivere il seguente codice in custompipe.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'custompipe'
```

```
})
export class CustomPipe implements PipeTransform {

  transform(value:number,exponent:number): number {

    return Math.pow(value,exponent);
  }

}
```

In pipejourney.component.ts fare l'import della custom pipe

```
import { CustompipePipe } from '../custompipe.pipe';
```

cambiare il codice di pipejourney.component.ts in:

```
import { Component } from '@angular/core';
import { CustompPipe } from '../custompipe.pipe';

@Component({
  selector: 'app-pipejourney',
  templateUrl: './pipejourney.component.html',
  styleUrl: './pipejourney.component.css'
})
export class PipejourneyComponent {

  today:number = Date.now();
  message:string = "Hello World";
  amount:number = 1234.89;

  person = {

    "firstName" : "Marco",
    "lastName" : "Rossi",
    "age" :21

  }

   custompipe:CustomPipe = new CustomPipe();
  numberWithExponent : number = this.custompipe.transform(3,8);

}
```

cambiare il codice di pipejourney.component.html

```
<p>{{today | date:'dd/MM/yyyy'}}</p>
<p>{{message | uppercase}}</p>
<p>{{message | lowercase}}</p>
```

```
<p>{{amount | currency:'USD'}}</p>
<p>{{person | json}}</p>
<p>{{numberWithExponent}}</p>
```

MODULI ANGULAR

creazione di una cartella module sotto src/app

porsi in src/app/module

eseguire il comando ng g m user per creare un nuovo modulo

eseguire il comando ng g c user/user per creare una componente user dentro il modulo user

codice user.component.ts interna al modulo user

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

@Component({
  selector: 'app-user',
  templateUrl: './user.component.html',
  styleUrl: './user.component.css'
})
export class UserComponent {

  user = {

    "username" :"myUsername"

  }

}
```

codice user.component.html interna al modulo user

```
<p>{{user | json}}</p>
```

codice user.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { UserComponent } from './user/user.component';



@NgModule({
  declarations: [
    UserComponent
  ],
  imports: [
```

```
    CommonModule
  ],
  exports:[
    UserComponent
  ]
})
export class UserModule { }
```

Importare UserModule in app.module.ts

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { EmployeesComponent } from './component/employees/employees.component';
import { SendComponent } from './component/send/send.component';
import { WelcomeComponent } from './component/welcome/welcome.component';
import { LabelComponent } from './component/label/label.component';
import { PipejourneyComponent } from
'./component/pipejourney/pipejourney.component';
import { CustompipePipe } from './component/custompipe.pipe';
import { UserModule } from './module/user/user.module';


@NgModule({
  declarations: [
    AppComponent,
    EmployeesComponent,
    SendComponent,
    WelcomeComponent,
    PipejourneyComponent,
    CustomPipe,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    UserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

modificare il codice di app.component.html

```
<app-welcome></app-welcome>
```

```
{{message1}} {{message2}}
<app-title></app-title>
<br><br>
<app-send (name)="handleName($event)"></app-send>
<app-employees [employees]="employees"></app-employees>
<app-pipejourney></app-pipejourney>
<app-user></app-user>
```

NB: PER INIETTARE LA COMPONENT DI UN MODULO A ALL'INTERNO DELLA COMPONENT DI UN MODULO B E' NECESSARIO ESPORTARE LA COMPONENT DAL MODULO A E IMPORTARE IL MODULO A NEL MODULO B.

PER DEFAULT, node.js EFFETTUA CARICAMENTO EAGER PER I MODULI, OVVERO ALLO START UP DELL'APPLICAZIONE I MODULI VENGONO CARICATI TUTTI.

VIENE CONSIDERATA UNA BEST PRACTICE CARICARE IN MODALITA' LAZY LOADING I MODULI, OVVERO CARICARLI QUANDO STRETTAMENTE NECESSARIO, CIOE' QUANDO VENGONO INVOCATE LE COMPONENT IN ESSE CONTENUTE. PER CHIEDERE A NODE.JS DI CARICARE IN MODALITA' LAZY LOADING UN MODULO, OCCORRE APPLICARE LA LOGICA NEL FILE app.routing.module INSERENDO NELLE ROTTE DI NAVIGAZIONE LA PAROLA CHIAVE loadChildren SEGUITA DAI : e da una ARROW ANONYMOUS FUNCTION CHE IMPORTA IL MODULO.

PER IL CARICAMENTO LAZY LOADING DEL MODULO app.routing.module SCRIVERE IL SEGUENTE COCODICE NEL FILE app-routing.module.ts.

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

const routes: Routes = [
  {
    path: 'user',
    loadChildren: () => import('./module/user/user.module').then(m => m.UserModule)
  }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

CHANGE DETECTION

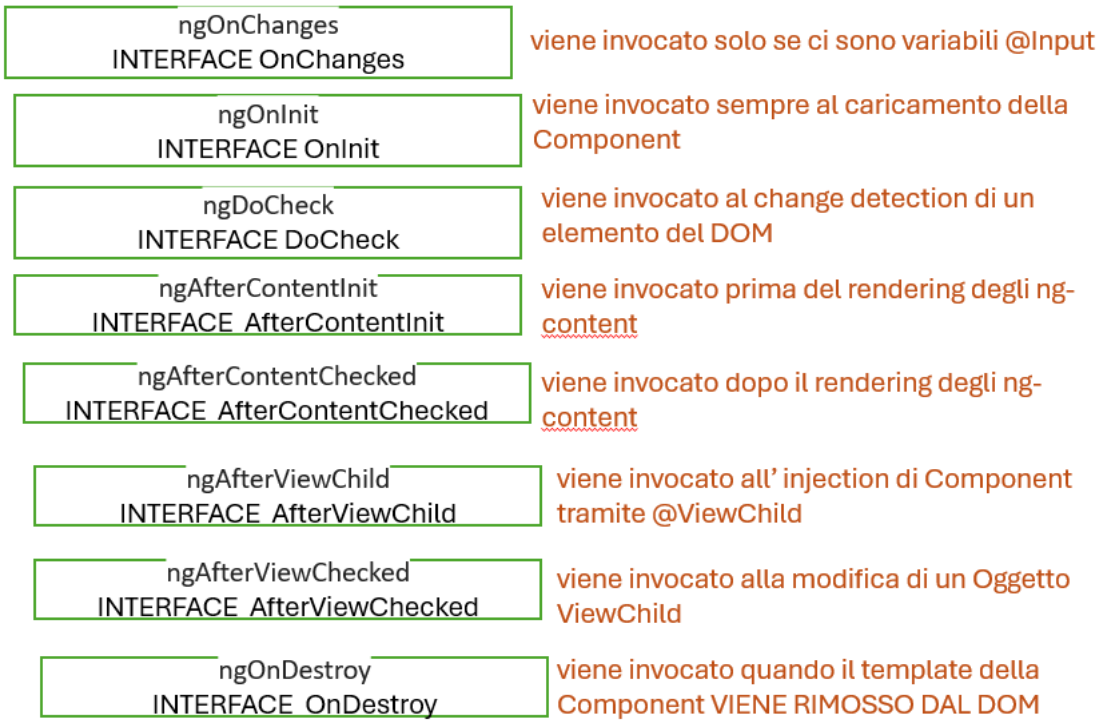
PER DEFAULT node.js EFFETTUA UN CARICAMENTO TOTALE DEL DOM, ANCHE SE L'AZIONE INTERATTIVA DELL'UTENTE COINVOLGE UNA O UN SOTTOINSIEME DI PORZIONI DEL DOM. ESEMPIO: SE NELL'index.html VIENE DIGITATO UN CAMPO DI TESTO DA PARTE DELL'UTENTE E TUTTO IL RESTO

RIMANE INVARIATO,Node.js RICARICA COMUNQUE TUTTA LA STRUTTURA DEL DOM. **VIENE CONSIDERATA UNA BEST PRACTICE FAR RICARICARE "SOLO CIO' CHE ESATTAMENTE SERVE", QUINDI LA PORZIONE DEL DOM DIRETTAMENTE INTERESSATA**. AL FINE DI OTTENERE QUESTO EFFETTO, E' NECESSARIO IMPOSTARE UNA STRATEGIA DI CHANGE DETECTION PER LA COMPONENT COINVOLTA CHIAMATA OnPush (LA STRATEGIA VA SETTATA NEL DECORATOR @Component DELLA CLASSE TYPESCRIPT). CON QUESTA STRATEGIA, SI OTTENGONO PERFORMANCE QUASI PARAGONABILI ALLA TECNOLOGIA VIRTUAL DOM DI REACT.

ESEMPIO PER LA COMPONENT SendComponent

```
Component({
  selector: 'app-send',
  templateUrl: './send.component.html',
  styleUrl: './send.component.css',
changeDetection: ChangeDetectionStrategy.OnPush
})
```

ANGULAR COMPONENT LIFECYCLE



APPLICAZIONE academy_restful_web_service_consumer

L'Applicazione è sviluppata nell'ultima versione di Angular (con le standalone Components).

STRUTTURA DI UN PROGETTO ANGULAR 17/18.

package.json (contiene la lista delle dipendenze installate tramite npm) – non c'è nessuna variazione rispetto alle versioni precedenti di Angular.

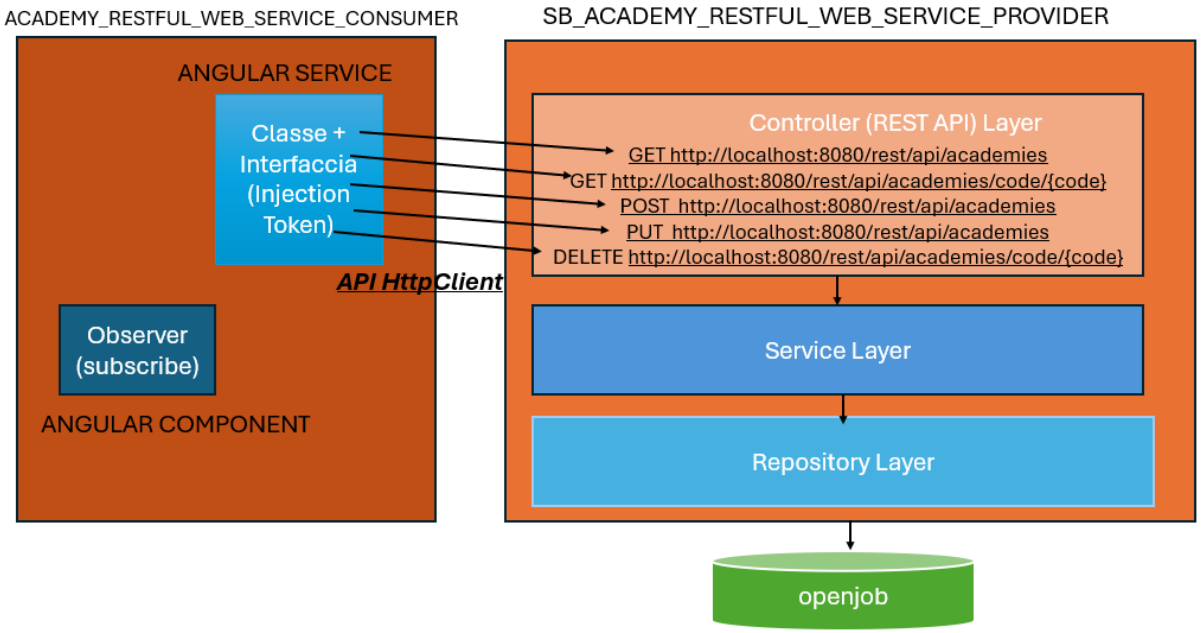
package-lock.json (contiene la lista delle dipendenze installate tramite npm e relativi link dell'npm registry dai quali le dipendenze vengono installate) – non c'è nessuna variazione rispetto alle versioni precedenti di Angular.

angular.json (è un file nel quale si possono fare configurazioni progettuali) – non c'è nessuna variazione rispetto alle versioni precedenti di Angular.

main.ts (file nel quale Node.js legge che allo start del progetto deve caricare l'AppComponent e deve esaminare il file app.config.ts) – c'è una variazione rispetto alle versioni precedenti di Angular, in cui il main.ts indicava a Node.js la necessità di caricare allo start il modulo AppModule. Nelle nuove versioni di Angular l'AppModule non è più obbligatorio. Le Component possono essere inserite in dei moduli,ma possono essere standalone, ed importare autonomamente moduli a seconda delle loro esigenze.

app.config.ts – non esisteva nelle versioni precedenti alla 17 di Angular. E' un file nel quale si può comunicare a Node.js di valutare la presenza di alcune API usate nel progetto, come ad esempio HttpClient per effettuare chiamate a Servizi rest.

ARCHITETTURA APPLICATIVA FULLSTACK



COMPONENT ANGULAR REALIZZATE

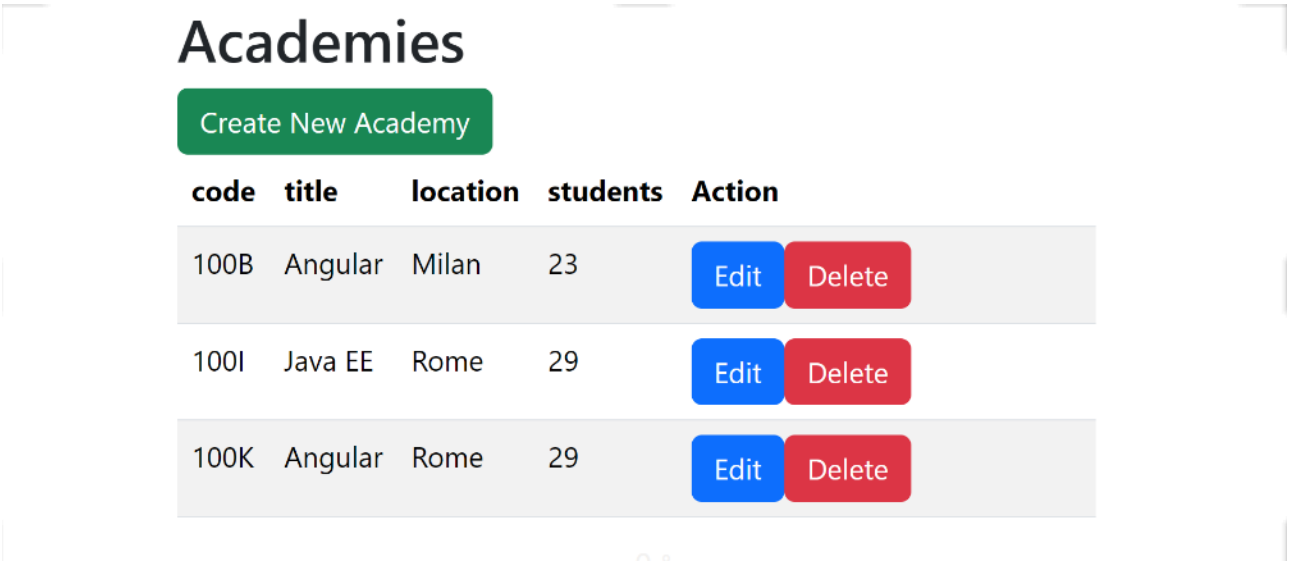


Figura 1 - Index Component (visualizza la lista delle Academies)

Figura 2 – CreateComponent (Component per l'inserimento di una nuova Academy)

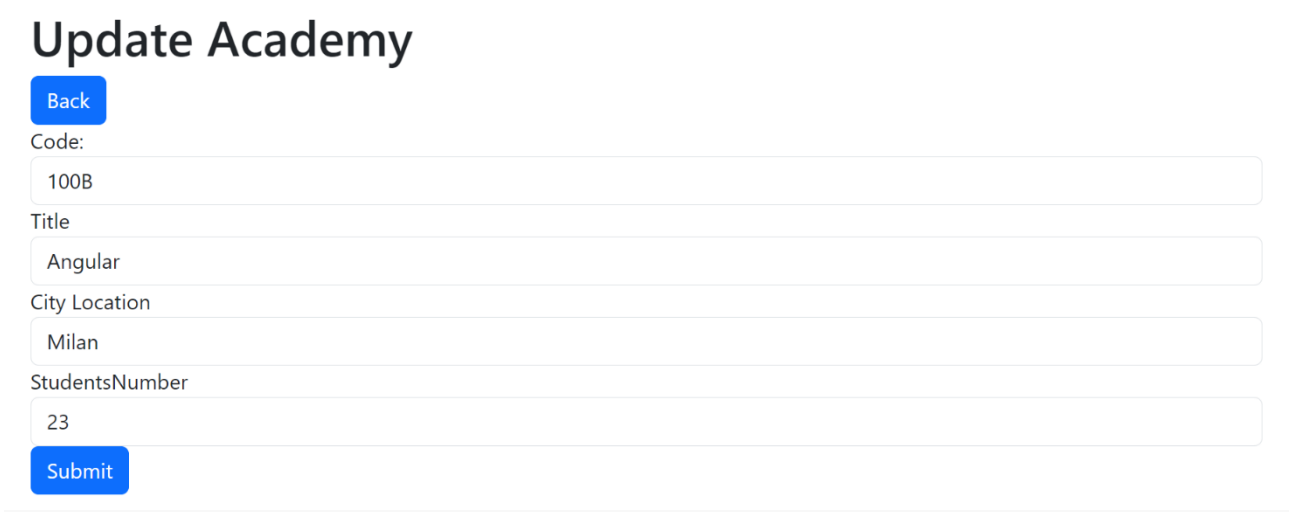


Figura 3 – EditComponent (Component per la modifica di una Academy già esistente)

STEP PROGETTUALI

1.CREAZIONE PROGETTO : ng new academy_restful_web_service_consumer

2. INSTALLAZIONE DIPENDENZA BOOTSTRAP

```
npm install bootstrap -save
```

3. CONFIGURAZIONE DIPENDENZA BOOTSTRAP IN angular.json

```
"node_modules/bootstrap/dist/css/bootstrap.min.css"
```

4. GENERAZIONE DI UN MODULO

```
ng generate module academy
```

5. GENERAZIONE DELLE COMPONENT DI PROGETTO

ng g c academy/index

ng g c academy/create

ng g c academy/edit

6. CONFIGURAZIONE DELLE ROTTE DI PROGETTO IN app.routes.ts

```
import { Routes } from '@angular/router';
import { IndexComponent } from './academy/index/index.component';
import { CreateComponent } from './academy/create/create.component';
import { EditComponent } from './academy/edit/edit.component';

export const routes: Routes = [
    { path: 'academy', redirectTo: 'academy/index', pathMatch: 'full'},
    { path: 'academy/index', component: IndexComponent },
    { path: 'academy/create', component: CreateComponent },
    { path: 'academy/:code/edit', component: EditComponent }
];
```

7. IMPLEMENTAZIONE INTERFACCIA MODELLO (SOTTO PACKAGE model academy.model.ts)

```
export interface Academy {

    code:string;
    title:string;
    cityLocation:string;
    studentsNumber:number;


}
```

8. CREAZIONE DI UNA CLASSE SERVICE (ng g s academy/academy) academy.service.ts

9. IMPLEMENTAZIONE DI UNA INTERFACCIA SERVICE (academyI.service.ts)

```
import { Observable } from "rxjs";
import { Academy } from "./model/academy.model";
import { InjectionToken } from "@angular/core";

export const academy_service_token = new
InjectionToken<AcademyServiceI>('academy_service_token');

export interface AcademyServiceI {


  getAcademies():Observable<any>;

  getAcademyByCode(code:string):Observable<any>;

  saveAcademy(academy:Academy):Observable<any>;

  updateAcademy(academy:Academy):Observable<any>;

  removeAcademy(code:string):Observable<any>;

}
```

10. IMPLEMENTAZIONE DELLA CLASSE SERVICE

```
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { catchError, Observable, throwError } from 'rxjs';
import { Academy } from './model/academy.model';
import { AcademyServiceI } from './academyI.service';

@Injectable({
  providedIn: 'root'
})
export class AcademyService implements AcademyServiceI{
```

```
  private apiURL = "http://localhost:8080/rest/api/academies";


  httpOptions = {
    headers: new HttpHeaders({
      'Content-Type': 'application/json'
    })
  }

  constructor(private httpClient: HttpClient) { }


  getAcademies(): Observable<any> {

    return this.httpClient.get(this.apiURL)

    .pipe(
      catchError(this.errorHandler)
    )
  }

  getAcademyByCode(code:string): Observable<any> {

    return this.httpClient.get(this.apiURL + '/code/' + code)

    .pipe(
      catchError(this.errorHandler)
    )
  }


  saveAcademy(academy:Academy): Observable<any> {

    return this.httpClient.post(this.apiURL,JSON.stringify(academy),
this.httpOptions)

    .pipe(
      catchError(this.errorHandler)
    )
  }

  updateAcademy(academy:Academy): Observable<any> {

    return this.httpClient.put(this.apiURL,JSON.stringify(academy),
this.httpOptions)

    .pipe(
      catchError(this.errorHandler)
    )
```

```
  }


  removeAcademy(code:string):Observable<any>{

    return this.httpClient.delete(this.apiURL + '/code/' + code, this.httpOptions)

    .pipe(
      catchError(this.errorHandler)
    )
  }


  errorHandler(error:any) {

    let errorMessage = '';

    if(error.error instanceof ErrorEvent) {
      errorMessage = error.error.message;
    } else {
      errorMessage = `Error Code: ${error.status}\nMessage: ${error.message}`;
    }
    return throwError(()=>new Error(errorMessage));
 }

}
```

NB : HttpClient è un Service built-in fornito da Angular per effettuare chiamate REST, e in quanto tale è iniettabile nel costruttore dei Service Custom. I metodi post, put, delete e get di HttpClient hanno come tipo di ritorno Observable. Observable è una API asincrona rxjs, libreria di front end trasversale, utilizzabile in qualunque applicazione e tecnologia Front End. Observable implementa il pattern Observer. Il Pattern Observer contempla la presenza di due attori applicativi : Observer, che si sottoscrivere per ricevere la notifica di eventi e li attende in modalità asincrona, e Observable, che notifica gli eventi. Observable ha un ciclo di vita rappresentato da 3 stati : NEXT (la chiamata REST è stata effettuata, ma la risposta non è ancora completata) ; COMPLETED (la risposta è stata completata con successo); ERROR (l'esito della risposta è negativo). Ogni Component che desidera invocare un Observable si deve sottoscrivere per farlo invocando il metodo subscribe Typescript.

11. IMPLEMENTAZIONE DELLA COMPONENT TS IndexComponent

```
import { Component, inject, Inject, OnInit } from '@angular/core';
import { Academy } from '../model/academy.model';
import { academy_service_token, AcademyServiceI } from '../academyI.service';
import { AcademyService } from '../academy.service';
import { CommonModule } from '@angular/common';
```

```
import { RouterModule } from '@angular/router';

@Component({
  selector: 'app-index',
  standalone: true,
  providers: [{ provide: academy_service_token, useClass: AcademyService }],
  imports: [CommonModule,RouterModule],
  templateUrl: './index.component.html',
  styleUrl: './index.component.css'
})
export class IndexComponent implements OnInit {


  academies: Academy[] = [];

  private academyService = inject<AcademyServiceI>(academy_service_token);

  constructor() {

  }

  getAcademies(): void {

    this.academyService.getAcademies().subscribe({
      next: (res) => {
        this.academies = res;
        console.log('Data fetched successfully', res);
      },
      error: (err) => {
        console.error('Error fetching data', err);
      }
    });

  }


  ngOnInit(): void {

   this.getAcademies();


  }

  removeAcademy(code: string) {

    this.academyService.removeAcademy(code).subscribe(res => {
      console.log(res.data);
      this.getAcademies();
    });
```

```
  }


}
```

12. IMPLEMENTAZIONE DELLA COMPONENT HTML index.component.html

```
<div class="container">
    <h1>Academies</h1>

    <a href="#" routerLink="/academy/create/" class="btn btn-success">Create New
Academy</a>

    <table class="table table-striped">
        <thead>
              <tr>
                <th>code</th>
                <th>title</th>
                <th>location</th>
                <th>students</th>
                <th width="250px">Action</th>
              </tr>
        </thead>
        <tbody>
              <tr *ngFor="let academy of academies">
                <td>{{ academy.code }}</td>
                <td>{{ academy.title }}</td>
                <td>{{ academy.cityLocation }}</td>
                <td>{{ academy.studentsNumber }}</td>
                <td>
                  <a href="#" [routerLink]="['/academy/', academy.code, 'edit']"
class="btn btn-primary">Edit</a>
                    <button type="button" (click)="removeAcademy(academy.code)"
class="btn btn-danger">Delete</button>
                </td>
              </tr>
        </tbody>
    </table>
 </div>
```

13. IMPLEMENTAZIONE DELLA COMPONENT HTML create.component.html

```
<div class="container">
    <h1>Save New Academy</h1>
```

```
    <form [formGroup]="form" (ngSubmit)="submit()">

        <div class="form-group">
            <label for="code">Code:</label>
            <input
                formControlName="code"
                id="code"
                type="text"
                class="form-control">
        </div>

        <div class="form-group">
            <label for="body">Title</label>
            <input
                formControlName="title"
                id="title"
                type="text"
                class="form-control">
        </div>

        <div class="form-group">
            <label for="cityLocation">City Location</label>
            <input
                formControlName="cityLocation"
                id="cityLocation"
                type="text"
                class="form-control">
        </div>

        <div class="form-group">
            <label for="studentsNumber">Students Number</label>
            <input
                formControlName="studentsNumber"
                id="studentsNumber"
                type="number"
                class="form-control">
        </div>

        <button class="btn btn-primary" type="submit">Submit</button>
    </form>
</div>
```

14. IMPLEMENTAZIONE DELLA COMPONENT TS create.component.ts

```
import { Component, inject, OnInit } from '@angular/core';
import { FormControl, FormGroup, FormsModule, ReactiveFormsModule, Validators }
from '@angular/forms';
import { academy_service_token, AcademyServiceI } from '../academyI.service';
import { AcademyService } from '../academy.service';
```

```
import { Router } from '@angular/router';
import { CommonModule } from '@angular/common';
import { Academy } from '../model/academy.model';


@Component({
  selector: 'app-create',
  standalone: true,
  providers: [{ provide: academy_service_token, useClass: AcademyService }],
  imports: [CommonModule,ReactiveFormsModule],
  templateUrl: './create.component.html',
  styleUrl: './create.component.css'
})
export class CreateComponent implements OnInit{


  academy! : Academy;

  private academyService = inject<AcademyServiceI>(academy_service_token);
  private router: Router = new Router;
  form!: FormGroup;

  ngOnInit(): void {

      this.form = new FormGroup({
        code: new FormControl(''),
        title: new FormControl(''),
        cityLocation: new FormControl(''),
        studentsNumber: new FormControl(0)
      });

    }

    submit() {

      this.academyService.saveAcademy(this.form.value).subscribe((res: any) => {
        console.log('Academy created successfully!');
        this.router.navigateByUrl('academy/index');
      })

    }




  }
```

15. IMPLEMENTAZIONE DELLA COMPONENT TS edit.component.ts

```
import { Component, inject, OnInit } from '@angular/core';
import { Academy } from '../model/academy.model';
import { CommonModule } from '@angular/common';
import { academy_service_token, AcademyServiceI } from '../academyI.service';
import { AcademyService } from '../academy.service';
import { ActivatedRoute, Router } from '@angular/router';
import { FormControl, FormGroup, FormsModule, ReactiveFormsModule, Validators }
from '@angular/forms';
import { BrowserModule } from '@angular/platform-browser';

@Component({
  selector: 'app-edit',
  standalone: true,
  providers: [{ provide: academy_service_token, useClass: AcademyService }],
  imports: [CommonModule,ReactiveFormsModule],
  templateUrl: './edit.component.html',
  styleUrl: './edit.component.css'
})
export class EditComponent implements OnInit {

  private academyService = inject<AcademyServiceI>(academy_service_token);

  academy!: Academy;

  code!: string;

  form!: FormGroup;

  constructor(private route: ActivatedRoute, private router: Router) {

  }

  ngOnInit(): void {

    this.code = this.route.snapshot.params['code'];
    this.academyService.getAcademyByCode(this.code).subscribe((data) => {
      this.academy = data;
      console.log(this.academy);

    });

    this.form = new FormGroup({
      code: new FormControl(''),
      title: new FormControl(''),
      cityLocation: new FormControl(''),
      studentsNumber: new FormControl()
    });
```

```
  }

  submit() {

    this.academyService.updateAcademy(this.form.value).subscribe((res: any) => {
      console.log('Academy updated successfully!');
      this.router.navigateByUrl('academy/index');
    })

  }



}
```

16. IMPLEMENTAZIONE DEL TEMPLATE edit.component.html

```
<div class="container">
    <h1>Update Academy</h1>


    <form [formGroup]="form" (ngSubmit)="submit()">

        <div class="form-group">
            <label for="code">Code:</label>
            <input
                formControlName="code"
                id="code"
                type="text"
                readonly
                [(ngModel)]="academy.code"
                class="form-control">
        </div>

        <div class="form-group">
            <label for="body">Title</label>
            <input
                formControlName="title"
                id="title"
                type="text"
                [(ngModel)]="academy.title"
                class="form-control">
        </div>

        <div class="form-group">
            <label for="cityLocation">City Location</label>
            <input
                formControlName="cityLocation"
```

```
                id="cityLocation"
                type="text"
                [(ngModel)]="academy.cityLocation"
                class="form-control">
        </div>

        <div class="form-group">
            <label for="studentsNumber">StudentsNumber</label>
            <input
                formControlName="studentsNumber"
                id="studentsNumber"
                type="number"
                [(ngModel)]="academy.studentsNumber"
                class="form-control">
        </div>

        <button class="btn btn-primary" type="submit">Submit</button>
    </form>
</div>
```

[(NgModel)] è UN TWO WAY BINDING CHE CONSENTE DI RIFLETTERE DATI DI UN OGGETTO TYPESCRIPT TRA UN TEMPLATE E UNA CLASSE COMPONENT E VICEVERSA.

LE FORM ANGULAR ALL'INTERNO DELLE QUALI VIENE USATO L'[(NgModel)] VENGONO TERMINOLOGICAMENTE CHIAMATE DAL FRAMEWORK TEMPLATE DRIVEN FORMS.

17.CONFIGURAZIONE app.component.html

```
<router-outlet></router-outlet>
```

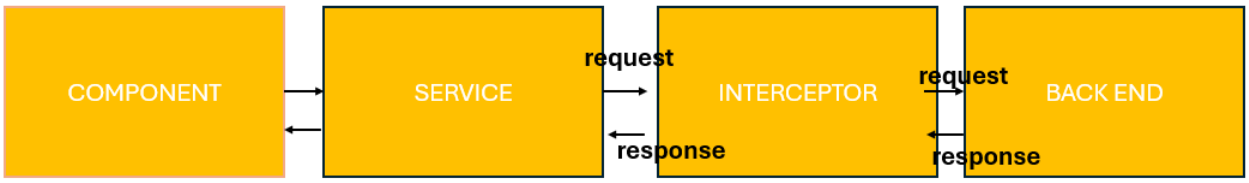
19. CONFIGURAZIONE app.config.ts

```
import { ApplicationConfig } from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';
import { provideAnimations } from '@angular/platform-browser/animations';

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

export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes), provideAnimations(), provideHttpClient()]
};
```

20. RUN DELL'APPLICAZIONE: npm start

21. AGGIUNTA DI UN INTERCEPTOR ANGULAR ALL'INTERNO DELL'APPLICAZIONE

GLI INTERCEPTOR SONO FUNZIONI PREDEFINITE ANGULAR (CUSTOMIZZABILI) CHE POSSONO ESSERE UTILIZZATE PER INTERCETTARE EVENTI DI RICHIESTA E/O RISPOSTA TRA L'APPLICAZIONE ANGULAR E APPLICAZIONI DI TERZE PARTI (APPLICAZIONI BACK END).



CREAZIONE INTERCEPTOR ALL'INTERNO DEL PERCORSO src/app/module

ng g interceptor logger

CODICE loggerinterceptor.interceptor.ts

```
import { HttpInterceptorFn } from '@angular/common/http';

export const loggerInterceptor: HttpInterceptorFn = (req, next) => {
  console.log(req.urlWithParams);
  return next(req);
};
```

CAMBIARE IL CODICE DEL FILE app.config.ts

```
import { ApplicationConfig, provideZoneChangeDetection } from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';
import { provideAnimations } from '@angular/platform-browser/animations';
import { provideHttpClient, withFetch, withInterceptors } from '@angular/common/http';
import { loggerInterceptor } from './module/loggerinterceptor.interceptor';

export const appConfig: ApplicationConfig = {
```

providers: [provideRouter(routes), provideAnimations(), provideHttpClient(withInterceptors([loggerInterceptor]), withFetch())

]

};

FARE IL RUN DELL'APPLICAZIONE path http://localhost:8080/rest/api/academies

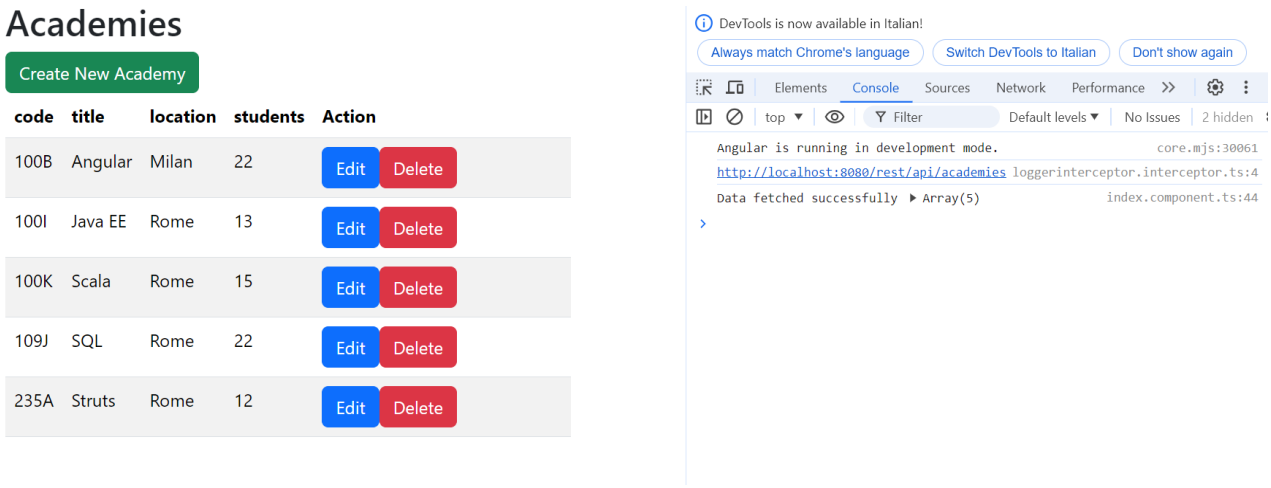


Figura 4 - Catch della url di chiamata verso il Servizio Rest

APPLICAZIONE academy_restful_web_service_consumer

L'Applicazione è sviluppata nell'ultima versione di Angular (con le standalone Components).

STRUTTURA DI UN PROGETTO ANGULAR 17/18.

package.json (contiene la lista delle dipendenze installate tramite npm) – non c'è nessuna variazione rispetto alle versioni precedenti di Angular.

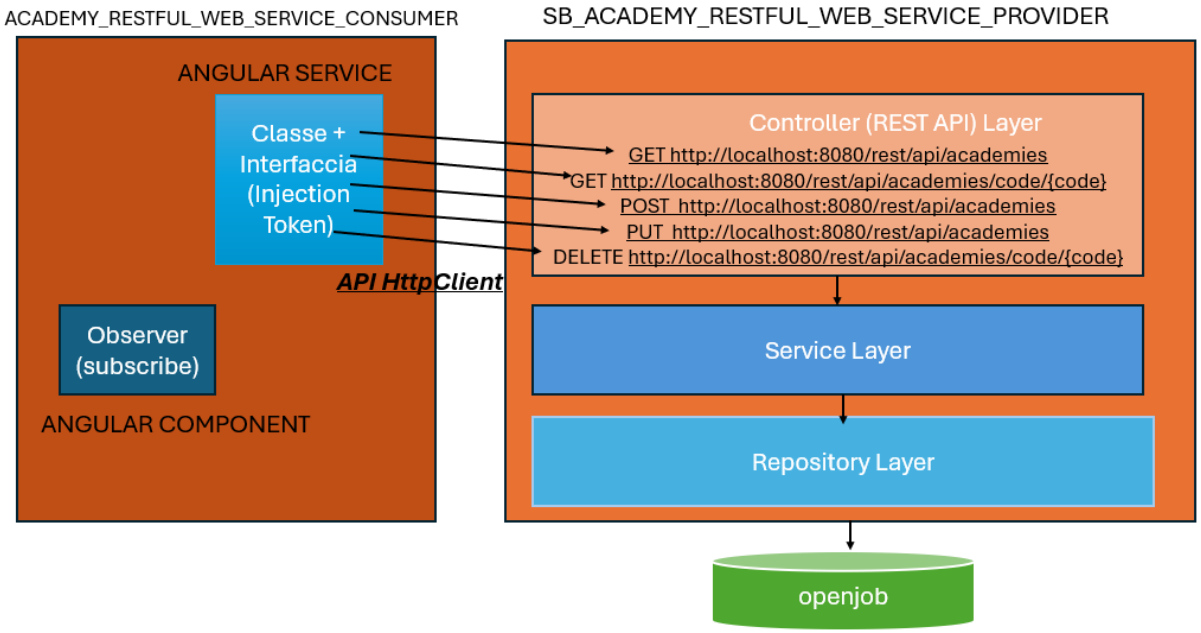
package-lock.json (contiene la lista delle dipendenze installate tramite npm e relativi link dell'npm registry dai quali le dipendenze vengono installate) – non c'è nessuna variazione rispetto alle versioni precedenti di Angular.

angular.json (è un file nel quale si possono fare configurazioni progettuali) – non c'è nessuna variazione rispetto alle versioni precedenti di Angular.

main.ts (file nel quale Node.js legge che allo start del progetto deve caricare l'AppComponent e deve esaminare il file app.config.ts) – c'è una variazione rispetto alle versioni precedenti di Angular, in cui il main.ts indicava a Node.js la necessità di caricare allo start il modulo AppModule. Nelle nuove versioni di Angular l'AppModule non è più obbligatorio. Le Component possono essere inserite in dei moduli,ma possono essere standalone, ed importare autonomamente moduli a seconda delle loro esigenze.

app.config.ts – non esisteva nelle versioni precedenti alla 17 di Angular. E' un file nel quale si può comunicare a Node.js di valutare la presenza di alcune API usate nel progetto, come ad esempio HttpClient per effettuare chiamate a Servizi rest.

ARCHITETTURA APPLICATIVA FULLSTACK



COMPONENT ANGULAR REALIZZATE

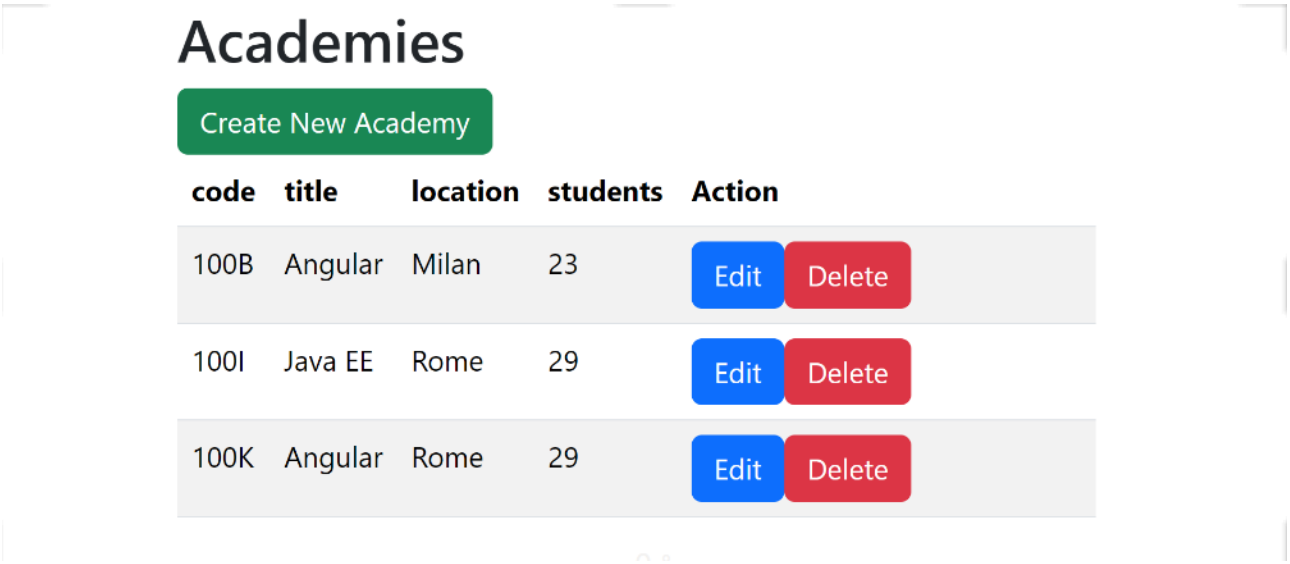


Figura 1 - Index Component (visualizza la lista delle Academies)

Figura 2 – CreateComponent (Component per l'inserimento di una nuova Academy)

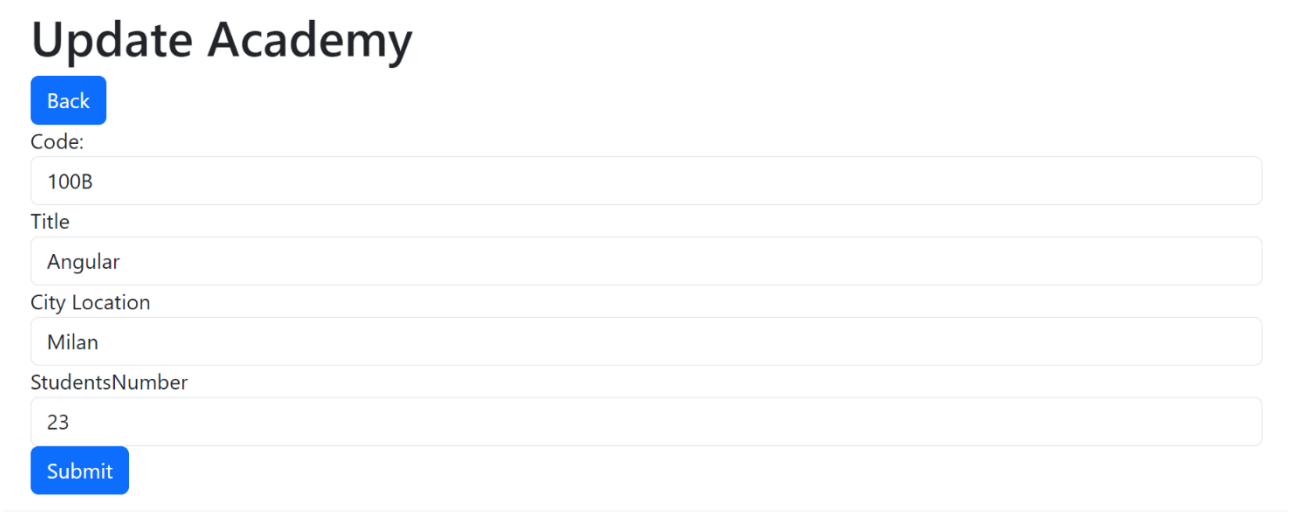


Figura 3 – EditComponent (Component per la modifica di una Academy già esistente)

STEP PROGETTUALI

1.CREAZIONE PROGETTO : ng new academy_restful_web_service_consumer

2. INSTALLAZIONE DIPENDENZA BOOTSTRAP

```
npm install bootstrap -save
```

3. CONFIGURAZIONE DIPENDENZA BOOTSTRAP IN angular.json

```
"node_modules/bootstrap/dist/css/bootstrap.min.css"
```

4. GENERAZIONE DI UN MODULO

```
ng generate module academy
```

5. GENERAZIONE DELLE COMPONENT DI PROGETTO

ng g c academy/index

ng g c academy/create

ng g c academy/edit

6. CONFIGURAZIONE DELLE ROTTE DI PROGETTO IN app.routes.ts

```
import { Routes } from '@angular/router';
import { IndexComponent } from './academy/index/index.component';
import { CreateComponent } from './academy/create/create.component';
import { EditComponent } from './academy/edit/edit.component';

export const routes: Routes = [
    { path: 'academy', redirectTo: 'academy/index', pathMatch: 'full'},
    { path: 'academy/index', component: IndexComponent },
    { path: 'academy/create', component: CreateComponent },
    { path: 'academy/:code/edit', component: EditComponent }
];
```

7. IMPLEMENTAZIONE INTERFACCIA MODELLO (SOTTO PACKAGE model academy.model.ts)

```
export interface Academy {

    code:string;
    title:string;
    cityLocation:string;
    studentsNumber:number;


}
```

8. CREAZIONE DI UNA CLASSE SERVICE (ng g s academy/academy) academy.service.ts

9. IMPLEMENTAZIONE DI UNA INTERFACCIA SERVICE (academyI.service.ts)

```
import { Observable } from "rxjs";
import { Academy } from "./model/academy.model";
import { InjectionToken } from "@angular/core";

export const academy_service_token = new
InjectionToken<AcademyServiceI>('academy_service_token');

export interface AcademyServiceI {


  getAcademies():Observable<any>;

  getAcademyByCode(code:string):Observable<any>;

  saveAcademy(academy:Academy):Observable<any>;

  updateAcademy(academy:Academy):Observable<any>;

  removeAcademy(code:string):Observable<any>;

}
```

10. IMPLEMENTAZIONE DELLA CLASSE SERVICE

```
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { catchError, Observable, throwError } from 'rxjs';
import { Academy } from './model/academy.model';
import { AcademyServiceI } from './academyI.service';

@Injectable({
  providedIn: 'root'
})
export class AcademyService implements AcademyServiceI{
```

```
  private apiURL = "http://localhost:8080/rest/api/academies";


  httpOptions = {
    headers: new HttpHeaders({
      'Content-Type': 'application/json'
    })
  }

  constructor(private httpClient: HttpClient) { }


  getAcademies(): Observable<any> {

    return this.httpClient.get(this.apiURL)

    .pipe(
      catchError(this.errorHandler)
    )
  }

  getAcademyByCode(code:string): Observable<any> {

    return this.httpClient.get(this.apiURL + '/code/' + code)

    .pipe(
      catchError(this.errorHandler)
    )
  }


  saveAcademy(academy:Academy): Observable<any> {

    return this.httpClient.post(this.apiURL,JSON.stringify(academy),
this.httpOptions)

    .pipe(
      catchError(this.errorHandler)
    )
  }

  updateAcademy(academy:Academy): Observable<any> {

    return this.httpClient.put(this.apiURL,JSON.stringify(academy),
this.httpOptions)

    .pipe(
      catchError(this.errorHandler)
    )
```

```
  }


  removeAcademy(code:string):Observable<any>{

    return this.httpClient.delete(this.apiURL + '/code/' + code, this.httpOptions)

    .pipe(
      catchError(this.errorHandler)
    )
  }


  errorHandler(error:any) {

    let errorMessage = '';

    if(error.error instanceof ErrorEvent) {
      errorMessage = error.error.message;
    } else {
      errorMessage = `Error Code: ${error.status}\nMessage: ${error.message}`;
    }
    return throwError(()=>new Error(errorMessage));
 }

}
```

NB : HttpClient è un Service built-in fornito da Angular per effettuare chiamate REST, e in quanto tale è iniettabile nel costruttore dei Service Custom. I metodi post, put, delete e get di HttpClient hanno come tipo di ritorno Observable. Observable è una API asincrona rxjs, libreria di front end trasversale, utilizzabile in qualunque applicazione e tecnologia Front End. Observable implementa il pattern Observer. Il Pattern Observer contempla la presenza di due attori applicativi : Observer, che si sottoscrivere per ricevere la notifica di eventi e li attende in modalità asincrona, e Observable, che notifica gli eventi. Observable ha un ciclo di vita rappresentato da 3 stati : NEXT (la chiamata REST è stata effettuata, ma la risposta non è ancora completata) ; COMPLETED (la risposta è stata completata con successo); ERROR (l'esito della risposta è negativo). Ogni Component che desidera invocare un Observable si deve sottoscrivere per farlo invocando il metodo subscribe Typescript.

11. IMPLEMENTAZIONE DELLA COMPONENT TS IndexComponent

```
import { Component, inject, Inject, OnInit } from '@angular/core';
import { Academy } from '../model/academy.model';
import { academy_service_token, AcademyServiceI } from '../academyI.service';
import { AcademyService } from '../academy.service';
import { CommonModule } from '@angular/common';
```

```
import { RouterModule } from '@angular/router';

@Component({
  selector: 'app-index',
  standalone: true,
  providers: [{ provide: academy_service_token, useClass: AcademyService }],
  imports: [CommonModule,RouterModule],
  templateUrl: './index.component.html',
  styleUrl: './index.component.css'
})
export class IndexComponent implements OnInit {


  academies: Academy[] = [];

  private academyService = inject<AcademyServiceI>(academy_service_token);

  constructor() {

  }

  getAcademies(): void {

    this.academyService.getAcademies().subscribe({
      next: (res) => {
        this.academies = res;
        console.log('Data fetched successfully', res);
      },
      error: (err) => {
        console.error('Error fetching data', err);
      }
    });

  }


  ngOnInit(): void {

   this.getAcademies();


  }

  removeAcademy(code: string) {

    this.academyService.removeAcademy(code).subscribe(res => {
      console.log(res.data);
      this.getAcademies();
    });
```

```
  }


}
```

12. IMPLEMENTAZIONE DELLA COMPONENT HTML index.component.html

```
<div class="container">
    <h1>Academies</h1>

    <a href="#" routerLink="/academy/create/" class="btn btn-success">Create New
Academy</a>

    <table class="table table-striped">
        <thead>
              <tr>
                <th>code</th>
                <th>title</th>
                <th>location</th>
                <th>students</th>
                <th width="250px">Action</th>
              </tr>
        </thead>
        <tbody>
              <tr *ngFor="let academy of academies">
                <td>{{ academy.code }}</td>
                <td>{{ academy.title }}</td>
                <td>{{ academy.cityLocation }}</td>
                <td>{{ academy.studentsNumber }}</td>
                <td>
                  <a href="#" [routerLink]="['/academy/', academy.code, 'edit']"
class="btn btn-primary">Edit</a>
                    <button type="button" (click)="removeAcademy(academy.code)"
class="btn btn-danger">Delete</button>
                </td>
              </tr>
        </tbody>
    </table>
 </div>
```

13. IMPLEMENTAZIONE DELLA COMPONENT HTML create.component.html

```
<div class="container">
    <h1>Save New Academy</h1>
```

```
    <form [formGroup]="form" (ngSubmit)="submit()">

        <div class="form-group">
            <label for="code">Code:</label>
            <input
                formControlName="code"
                id="code"
                type="text"
                class="form-control">
        </div>

        <div class="form-group">
            <label for="body">Title</label>
            <input
                formControlName="title"
                id="title"
                type="text"
                class="form-control">
        </div>

        <div class="form-group">
            <label for="cityLocation">City Location</label>
            <input
                formControlName="cityLocation"
                id="cityLocation"
                type="text"
                class="form-control">
        </div>

        <div class="form-group">
            <label for="studentsNumber">Students Number</label>
            <input
                formControlName="studentsNumber"
                id="studentsNumber"
                type="number"
                class="form-control">
        </div>

        <button class="btn btn-primary" type="submit">Submit</button>
    </form>
</div>
```

14. IMPLEMENTAZIONE DELLA COMPONENT TS create.component.ts

```
import { Component, inject, OnInit } from '@angular/core';
import { FormControl, FormGroup, FormsModule, ReactiveFormsModule, Validators }
from '@angular/forms';
import { academy_service_token, AcademyServiceI } from '../academyI.service';
import { AcademyService } from '../academy.service';
```

```
import { Router } from '@angular/router';
import { CommonModule } from '@angular/common';
import { Academy } from '../model/academy.model';


@Component({
  selector: 'app-create',
  standalone: true,
  providers: [{ provide: academy_service_token, useClass: AcademyService }],
  imports: [CommonModule,ReactiveFormsModule],
  templateUrl: './create.component.html',
  styleUrl: './create.component.css'
})
export class CreateComponent implements OnInit{


  academy! : Academy;

  private academyService = inject<AcademyServiceI>(academy_service_token);
  private router: Router = new Router;
  form!: FormGroup;

  ngOnInit(): void {

      this.form = new FormGroup({
        code: new FormControl(''),
        title: new FormControl(''),
        cityLocation: new FormControl(''),
        studentsNumber: new FormControl(0)
      });

    }

    submit() {

      this.academyService.saveAcademy(this.form.value).subscribe((res: any) => {
        console.log('Academy created successfully!');
        this.router.navigateByUrl('academy/index');
      })

    }




  }
```

15. IMPLEMENTAZIONE DELLA COMPONENT TS edit.component.ts

```
import { Component, inject, OnInit } from '@angular/core';
import { Academy } from '../model/academy.model';
import { CommonModule } from '@angular/common';
import { academy_service_token, AcademyServiceI } from '../academyI.service';
import { AcademyService } from '../academy.service';
import { ActivatedRoute, Router } from '@angular/router';
import { FormControl, FormGroup, FormsModule, ReactiveFormsModule, Validators }
from '@angular/forms';
import { BrowserModule } from '@angular/platform-browser';

@Component({
  selector: 'app-edit',
  standalone: true,
  providers: [{ provide: academy_service_token, useClass: AcademyService }],
  imports: [CommonModule,ReactiveFormsModule],
  templateUrl: './edit.component.html',
  styleUrl: './edit.component.css'
})
export class EditComponent implements OnInit {

  private academyService = inject<AcademyServiceI>(academy_service_token);

  academy!: Academy;

  code!: string;

  form!: FormGroup;

  constructor(private route: ActivatedRoute, private router: Router) {

  }

  ngOnInit(): void {

    this.code = this.route.snapshot.params['code'];
    this.academyService.getAcademyByCode(this.code).subscribe((data) => {
      this.academy = data;
      console.log(this.academy);

    });

    this.form = new FormGroup({
      code: new FormControl(''),
      title: new FormControl(''),
      cityLocation: new FormControl(''),
      studentsNumber: new FormControl()
    });
```

```
  }

  submit() {

    this.academyService.updateAcademy(this.form.value).subscribe((res: any) => {
      console.log('Academy updated successfully!');
      this.router.navigateByUrl('academy/index');
    })

  }



}
```

16. IMPLEMENTAZIONE DEL TEMPLATE edit.component.html

```
<div class="container">
    <h1>Update Academy</h1>


    <form [formGroup]="form" (ngSubmit)="submit()">

        <div class="form-group">
            <label for="code">Code:</label>
            <input
                formControlName="code"
                id="code"
                type="text"
                readonly
                [(ngModel)]="academy.code"
                class="form-control">
        </div>

        <div class="form-group">
            <label for="body">Title</label>
            <input
                formControlName="title"
                id="title"
                type="text"
                [(ngModel)]="academy.title"
                class="form-control">
        </div>

        <div class="form-group">
            <label for="cityLocation">City Location</label>
            <input
                formControlName="cityLocation"
```

```
                id="cityLocation"
                type="text"
                [(ngModel)]="academy.cityLocation"
                class="form-control">
        </div>

        <div class="form-group">
            <label for="studentsNumber">StudentsNumber</label>
            <input
                formControlName="studentsNumber"
                id="studentsNumber"
                type="number"
                [(ngModel)]="academy.studentsNumber"
                class="form-control">
        </div>

        <button class="btn btn-primary" type="submit">Submit</button>
    </form>
</div>
```

[(NgModel)] è UN TWO WAY BINDING CHE CONSENTE DI RIFLETTERE DATI DI UN OGGETTO TYPESCRIPT TRA UN TEMPLATE E UNA CLASSE COMPONENT E VICEVERSA.

LE FORM ANGULAR ALL'INTERNO DELLE QUALI VIENE USATO L'[(NgModel)] VENGONO TERMINOLOGICAMENTE CHIAMATE DAL FRAMEWORK TEMPLATE DRIVEN FORMS.

17.CONFIGURAZIONE app.component.html

```
<router-outlet></router-outlet>
```

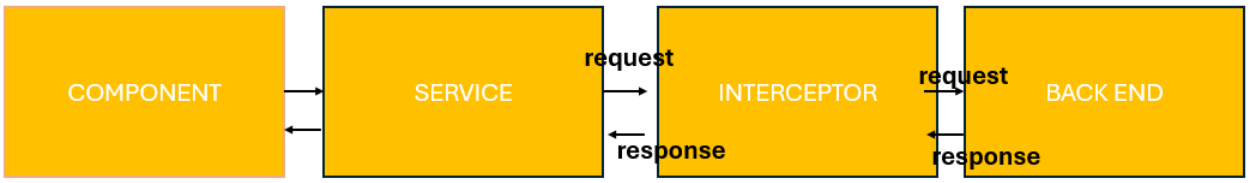
19. CONFIGURAZIONE app.config.ts

```
import { ApplicationConfig } from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';
import { provideAnimations } from '@angular/platform-browser/animations';

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

export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes), provideAnimations(), provideHttpClient()]
};
```

20. RUN DELL'APPLICAZIONE: npm start

21. AGGIUNTA DI UN INTERCEPTOR ANGULAR ALL'INTERNO DELL'APPLICAZIONE

GLI INTERCEPTOR SONO FUNZIONI PREDEFINITE ANGULAR (CUSTOMIZZABILI) CHE POSSONO ESSERE UTILIZZATE PER INTERCETTARE EVENTI DI RICHIESTA E/O RISPOSTA TRA L'APPLICAZIONE ANGULAR E APPLICAZIONI DI TERZE PARTI (APPLICAZIONI BACK END).



CREAZIONE INTERCEPTOR ALL'INTERNO DEL PERCORSO src/app/module

ng g interceptor logger

CODICE loggerinterceptor.interceptor.ts

```
import { HttpInterceptorFn } from '@angular/common/http';

export const loggerInterceptor: HttpInterceptorFn = (req, next) => {
  console.log(req.urlWithParams);
  return next(req);
};
```

CAMBIARE IL CODICE DEL FILE app.config.ts

```
import { ApplicationConfig, provideZoneChangeDetection } from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';
import { provideAnimations } from '@angular/platform-browser/animations';
import { provideHttpClient, withFetch, withInterceptors } from '@angular/common/http';
import { loggerInterceptor } from './module/loggerinterceptor.interceptor';

export const appConfig: ApplicationConfig = {
```

providers: [provideRouter(routes), provideAnimations(), provideHttpClient(withInterceptors([loggerInterceptor]), withFetch())

]

};

FARE IL RUN DELL'APPLICAZIONE path http://localhost:8080/rest/api/academies

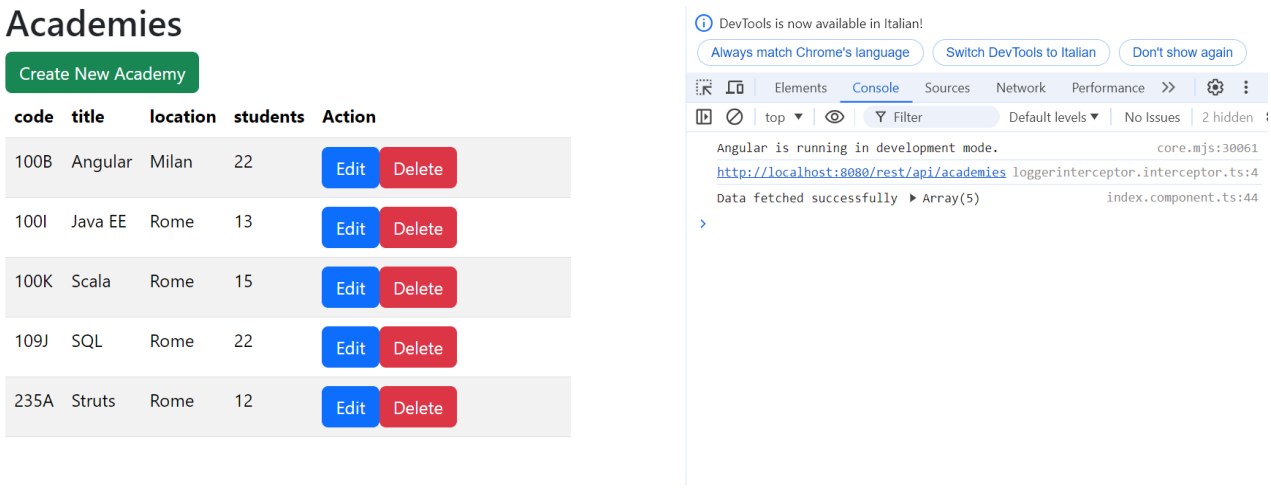


Figura 4 - Catch della url di chiamata verso il Servizio Rest