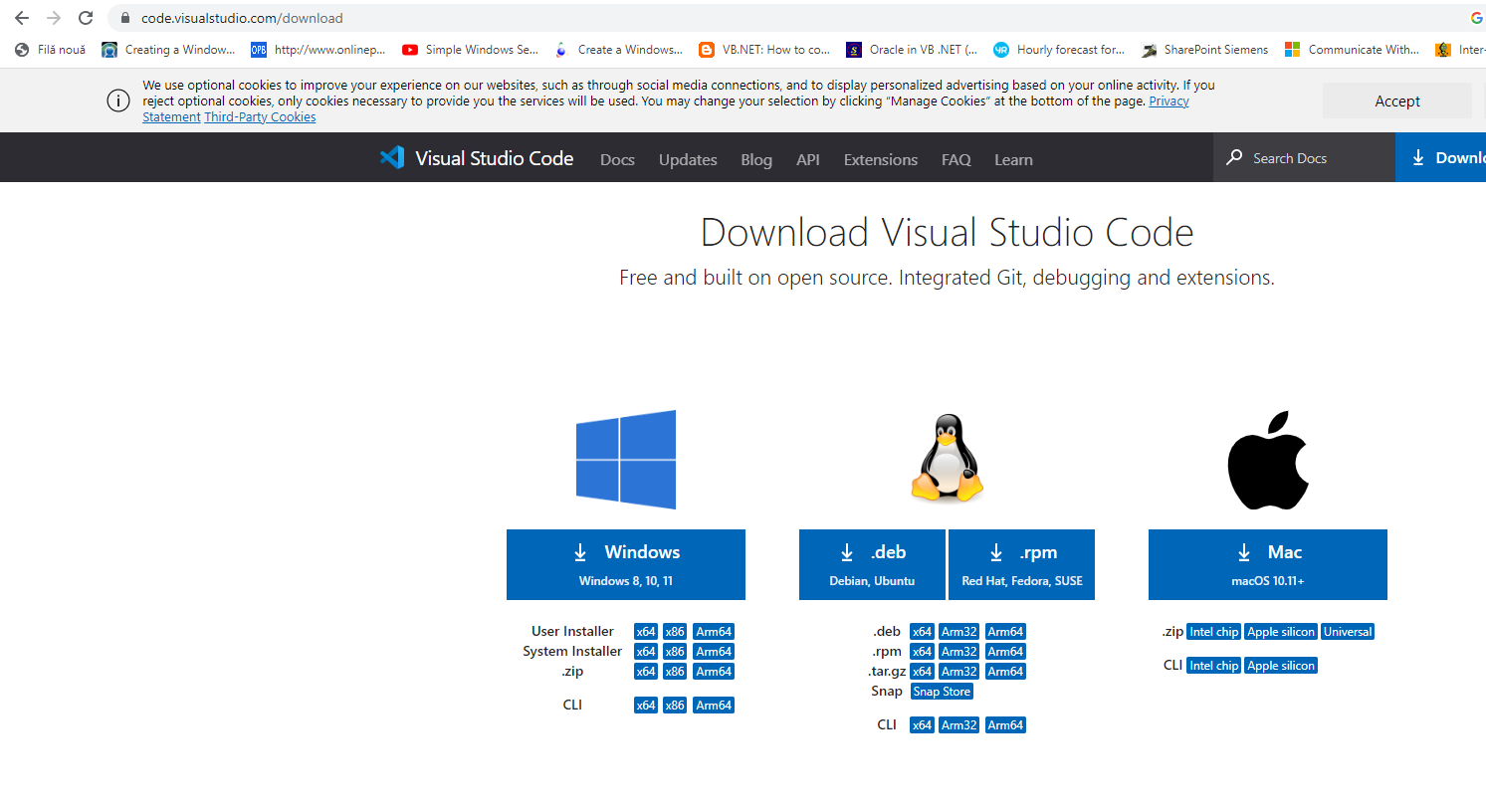
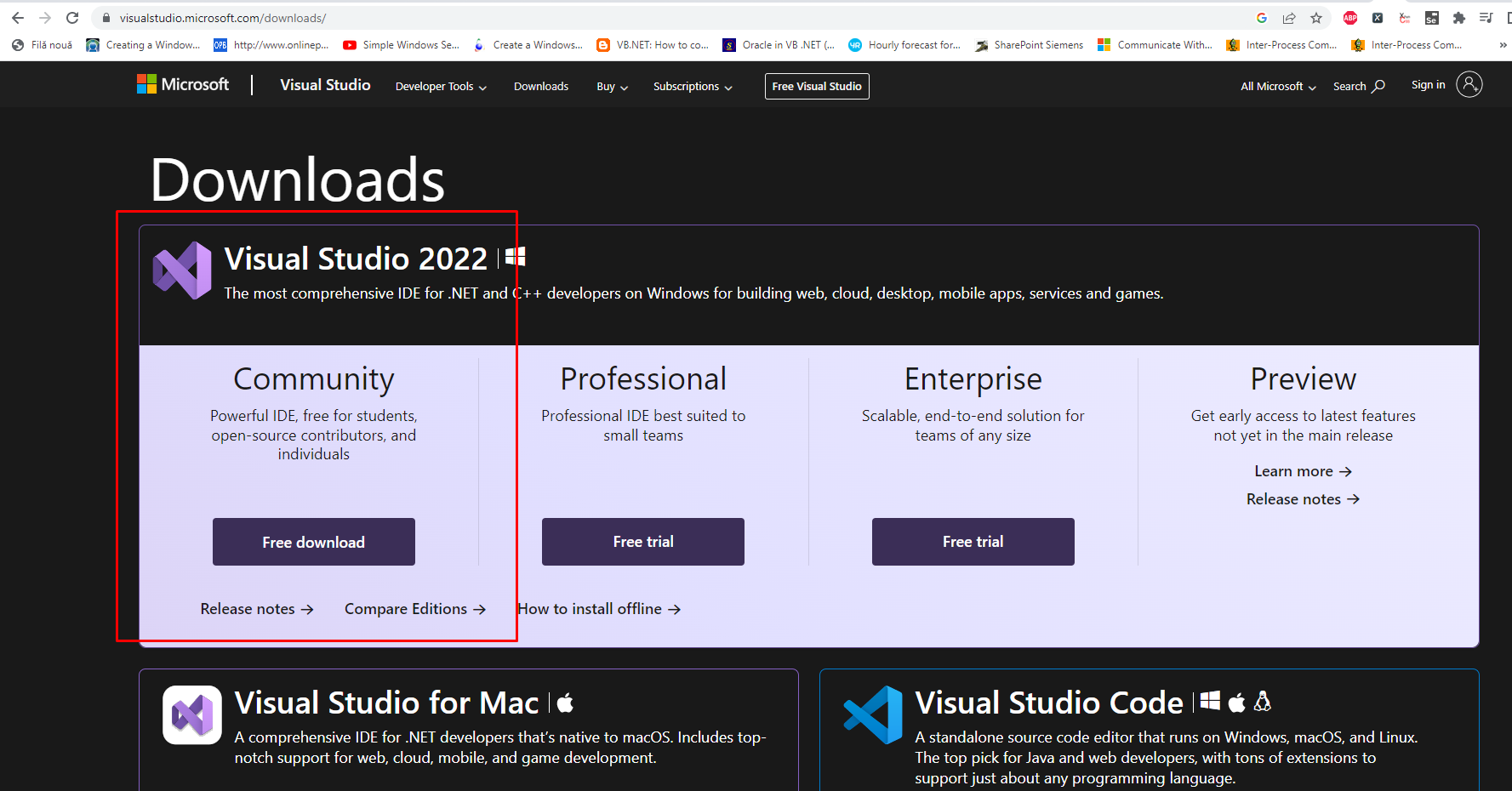
**Angular 14 CRUD cu .NET6 Web API si Entity Framework Core**

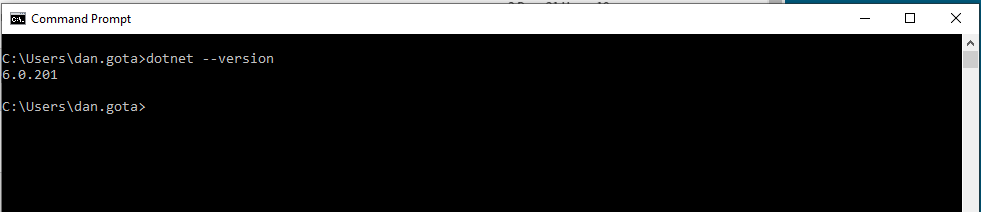
Pas1: Download Visual Studio Code ( pentru partea de frotend)



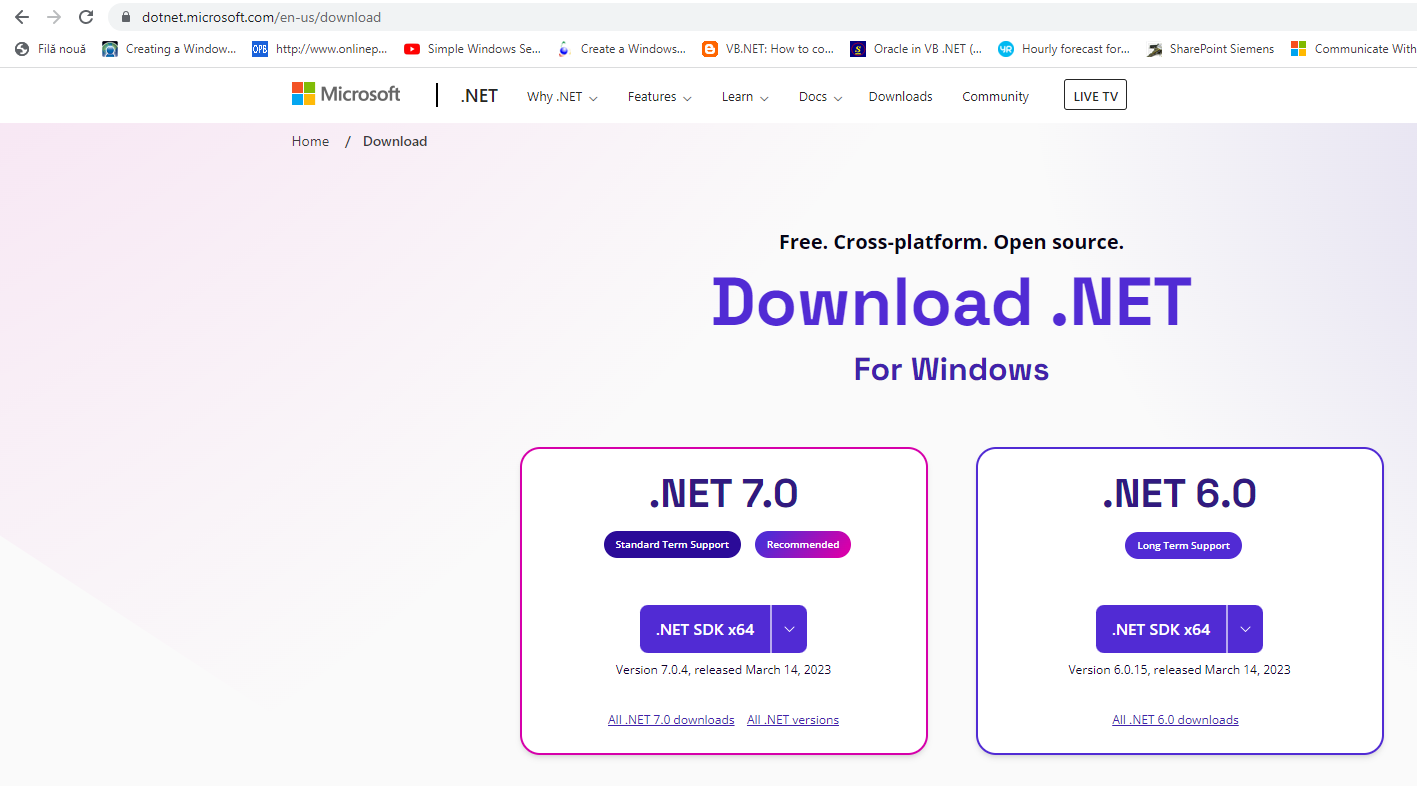
Pas2: Descarcati Visual Studio (pentru artea de backend)



Pas3: Verificati versiunea de .NET instalata pe calculator



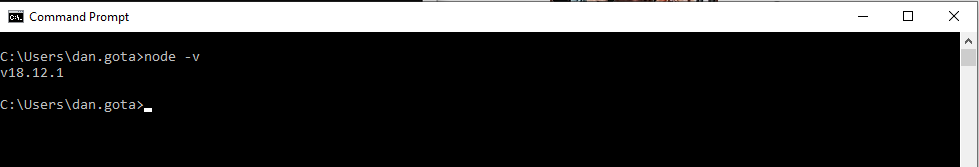
Pas2: Instalati ultima versiune de .NET



Pas4: Instalati nodeJS

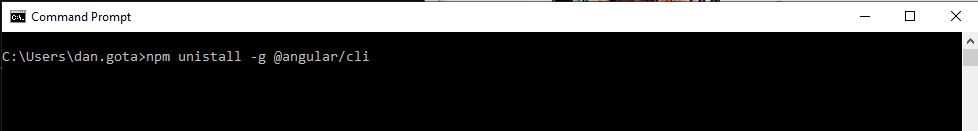


Pas5: Verificare versiune nodeJS



Pas6: Instalare Angular CLI

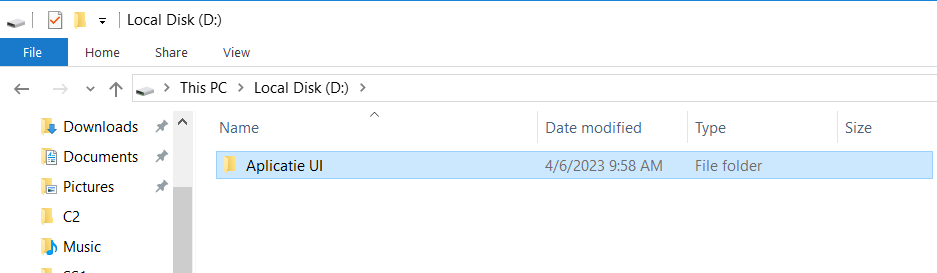
Mai intai dezinstalati versiunea curenta de Angular



Mai apoi instalati global ultima varianta de Angular



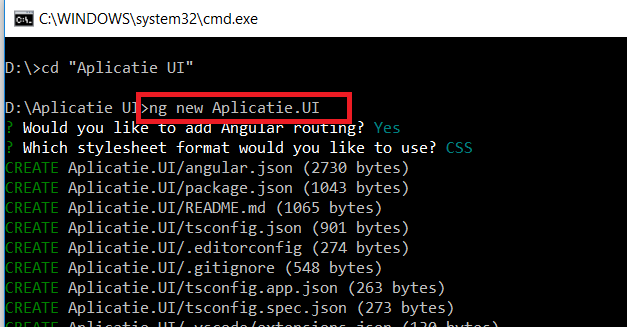
Pas7: Creati pe disc un folder denumit Aplicatie UI



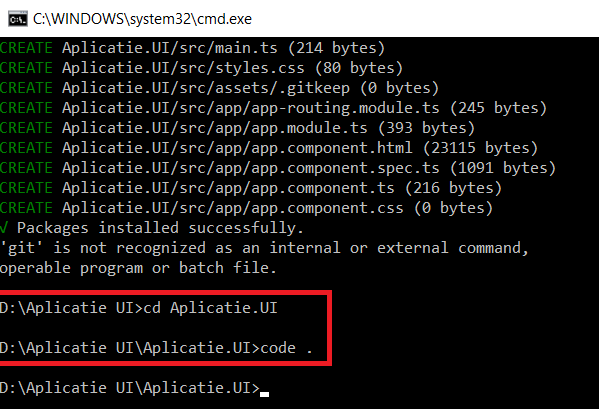
Pas8: Creati in aceasta locatie un proiect angular denumit Aplicatie.UI direct din consola de comanda, utilizand comanda: ng new Aplicatie.UI

**Selectati mai apoi optiunile: Would you like to add Angular routing? YES**

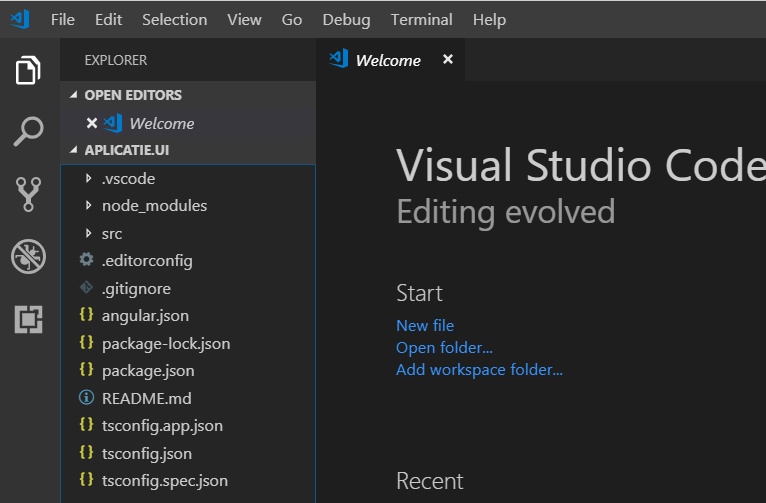
**Which style sheet format would you like to use? CSS**



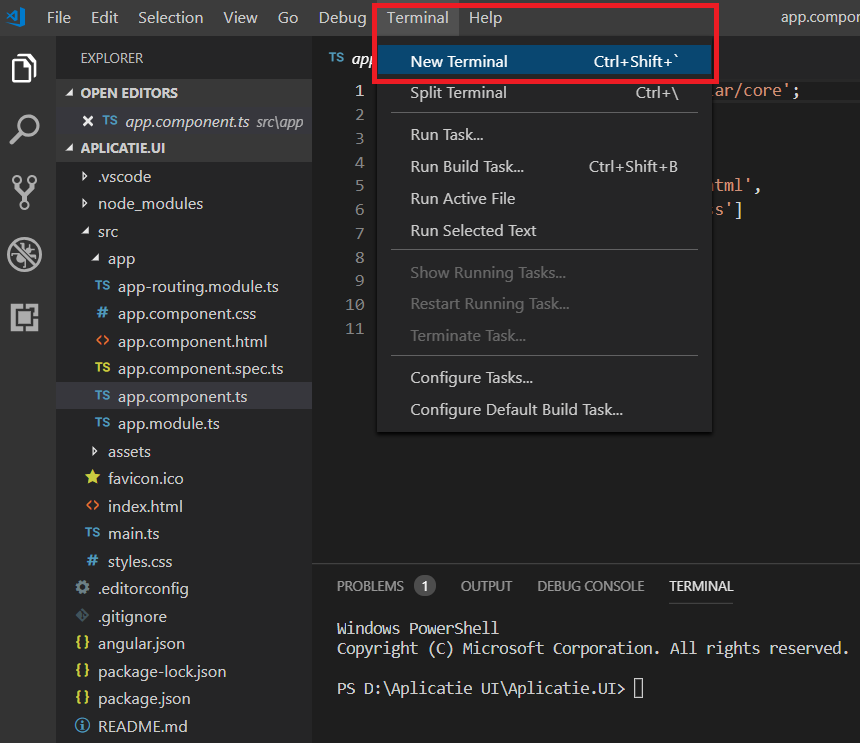
Pas9: Deschideti proiectul de UI cu Visual Studio Code prin scrierea comenzii **code .** in linia de comanda



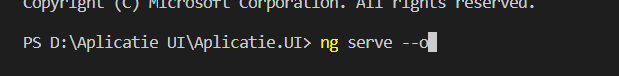
Ruland aceasta comanda o sa se deschida aplicatia de UI in aplicatia Visula Studio Code.



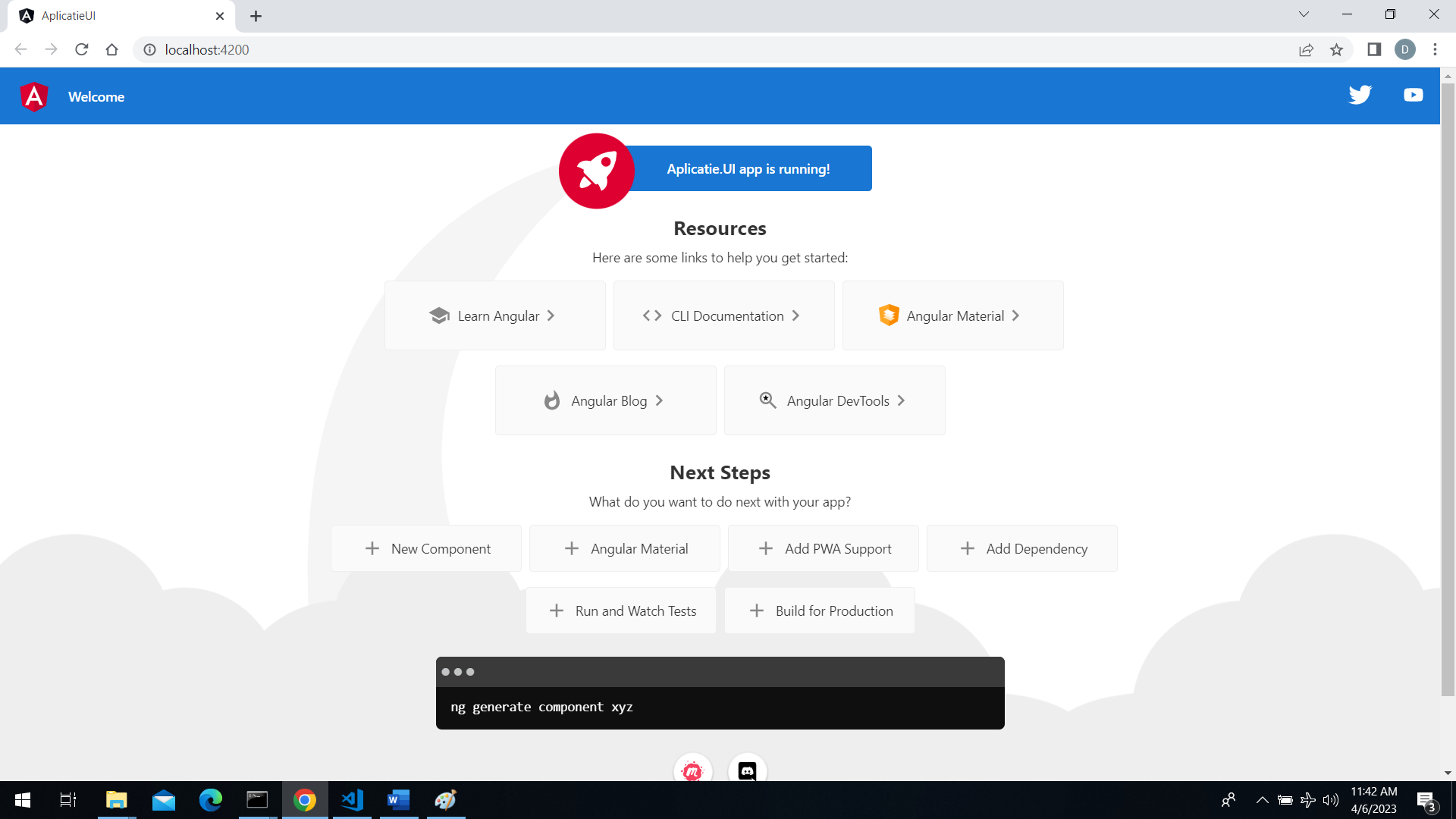
In cele ce urmeaza o sa rulam aplicatia pentru a observa ce avem implicit in aplicatia creata. Pentru a face acest lucru trebuie sa activam o fereastra de terminal la fel ca in imaginea de mai jos:



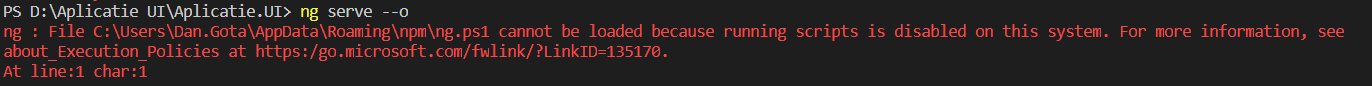
Mai apoi, in terminal, vom scrie urmatoarea comanda: **ng serve - -o** pentru a deschide aplicatie intr-un browser.

****

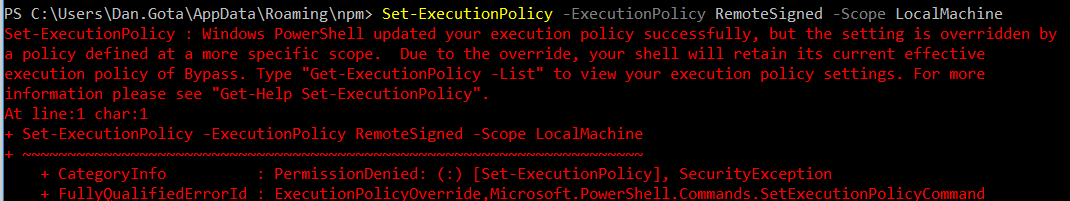
**Rezultatul este urmatorul:**



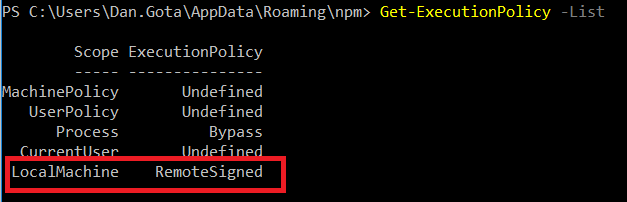
Daca la rularea comenzii ng serve - - o apare urmatoarea eroare:



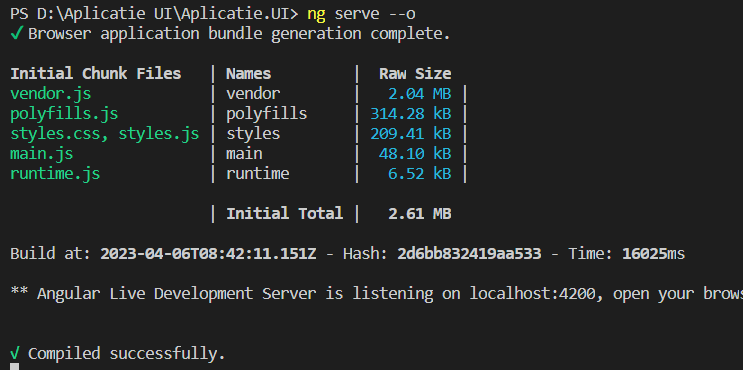
Aceasta se rezolva deschizand in modul administrator Command Prompt si Ruland urmatoarea comanda:



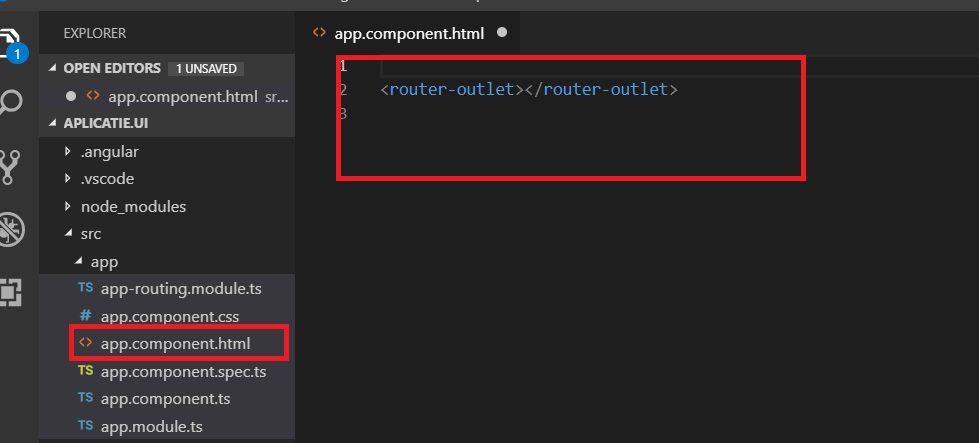
Mai apoi verificati faptul ca pe masina locala s-a setat correct politica de executie:



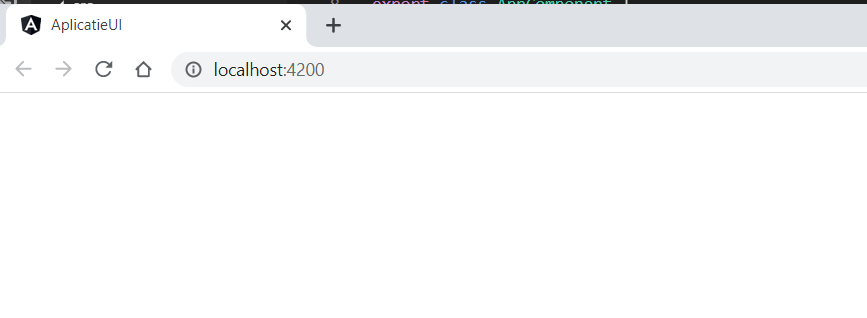
Mai apoi se va rula cu success comanda ng serve - -o.



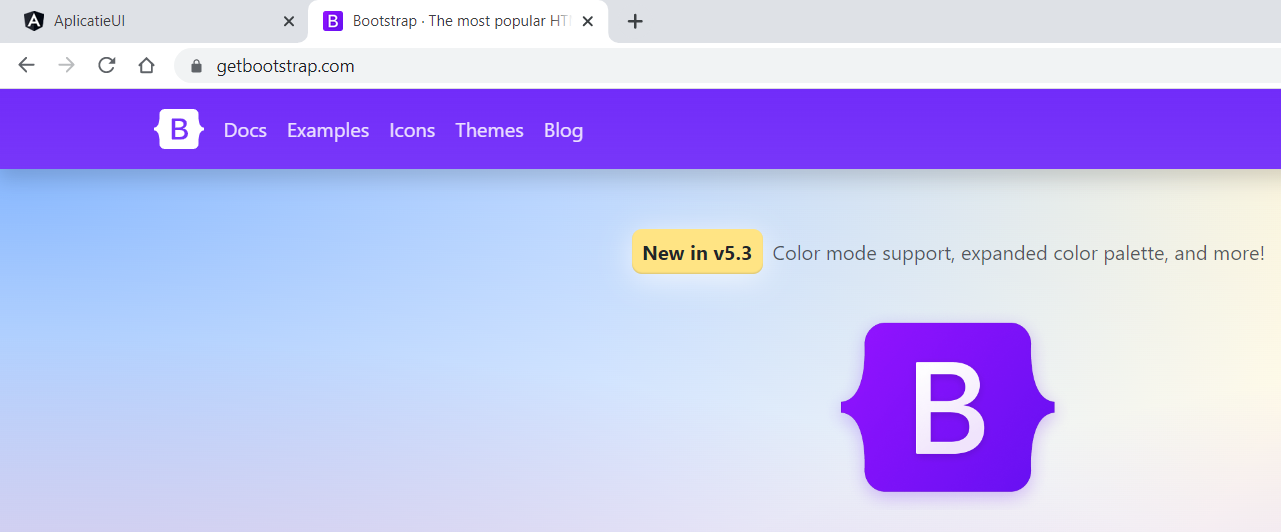
In cele ce urmeaza vom curata aplicatia noastra deoarece aceasta contine doar cod html static. Pentru curatare o sa mergem in proiect la fisierul app.component.html si vom sterge intre continutul in afara de tagul router-outlet, necesar pentru aplciarea routarilor.

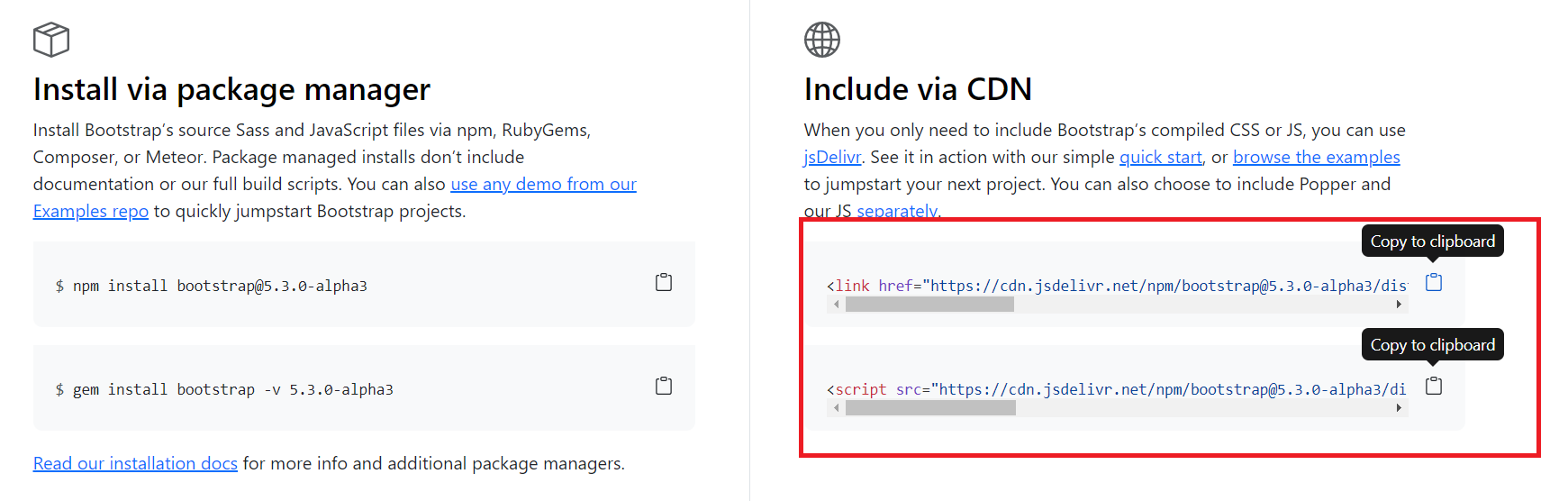


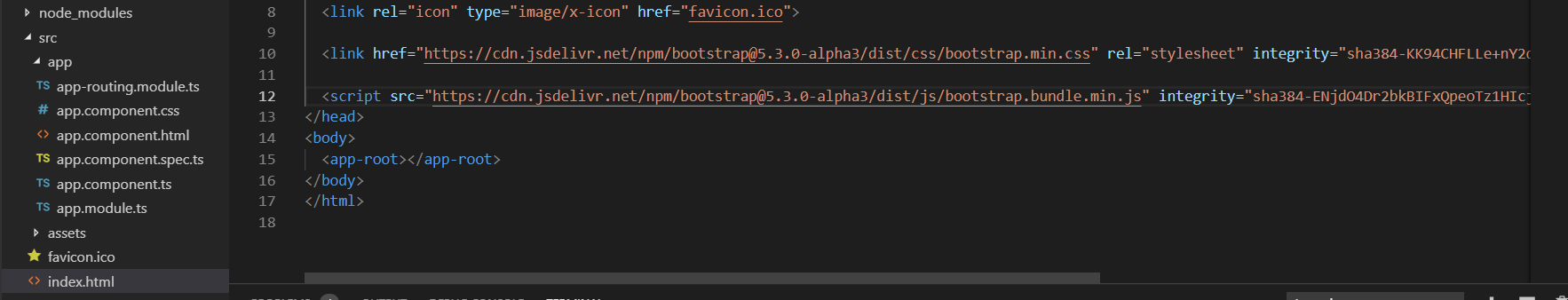
Rezultatul este urmatorul:



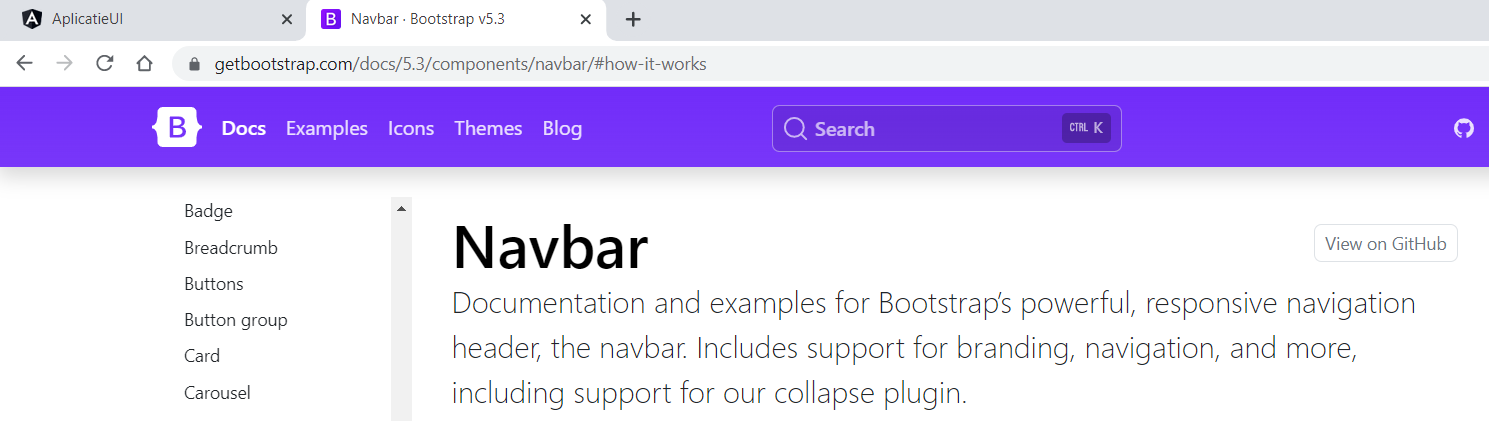
In cele ce urmeaza vom instala in proiect Boostrap pentru a putea utiliza componentele si stilurile acestuia in proiect. Pentru a instala boostrap in aplicatie vom naviga la site-ul boostrap de mai jos:

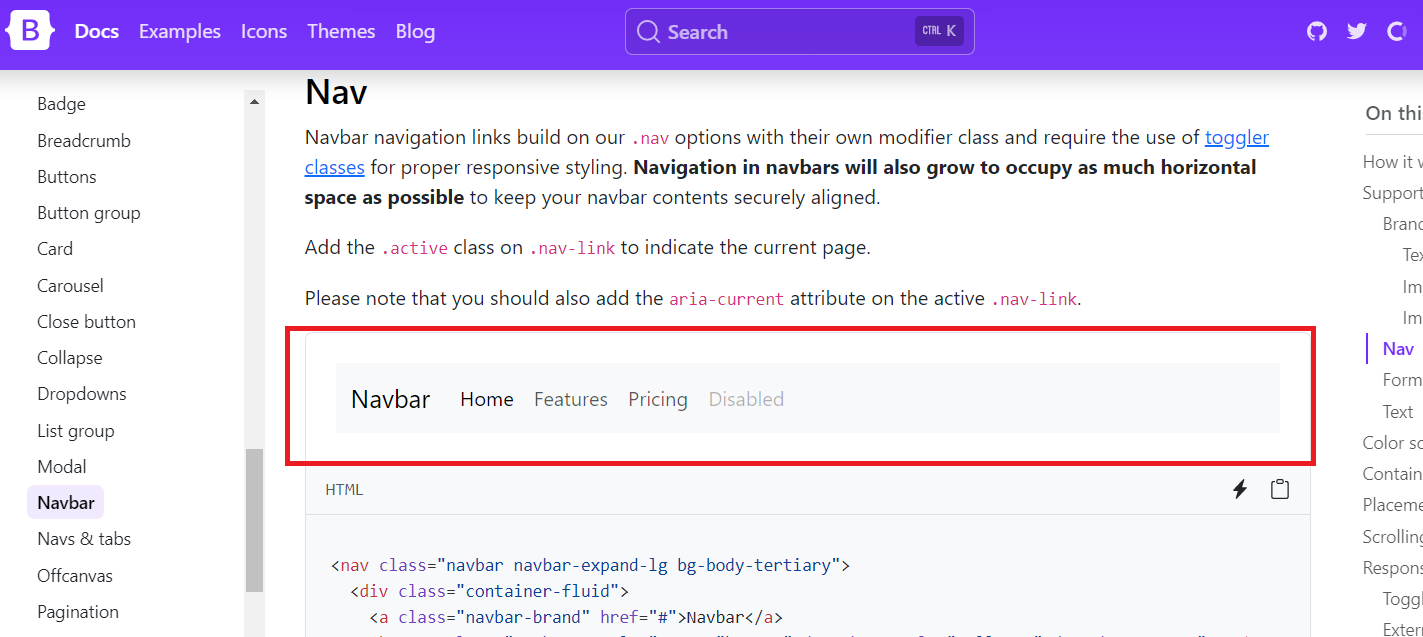


Vom copia mai apoi cele doua taguri in fisierul index.html din proiectul nostrum.



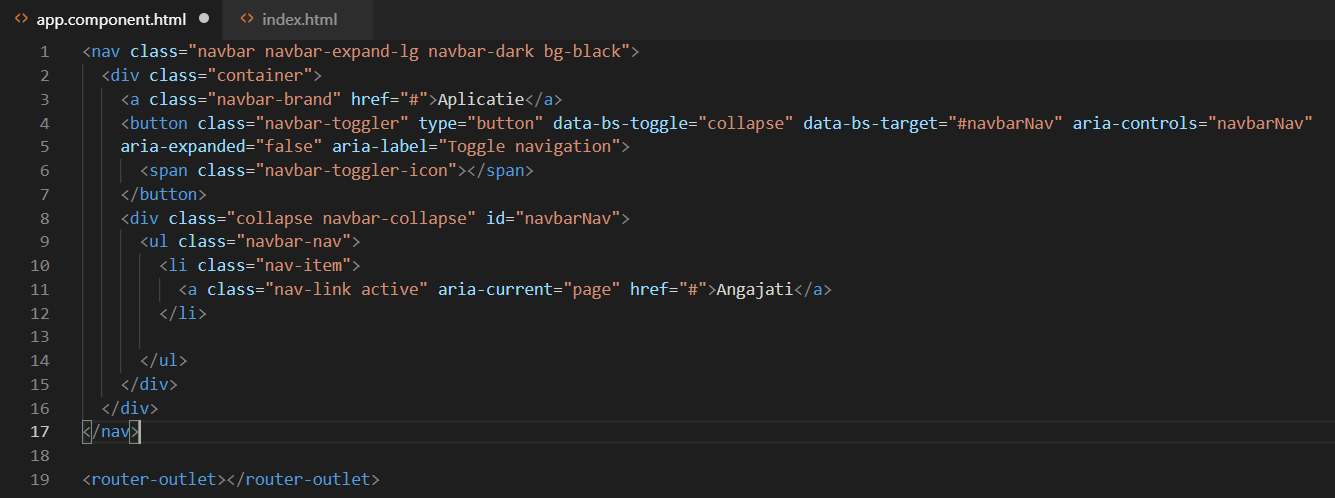
In cele ce urmeaza dorim sa cream o bara de navigare pentru proeictul nostru. Vom accesa aceasta componenta din site-ul boostrap preluand cea mai simpla componenta de navigare de pe site.



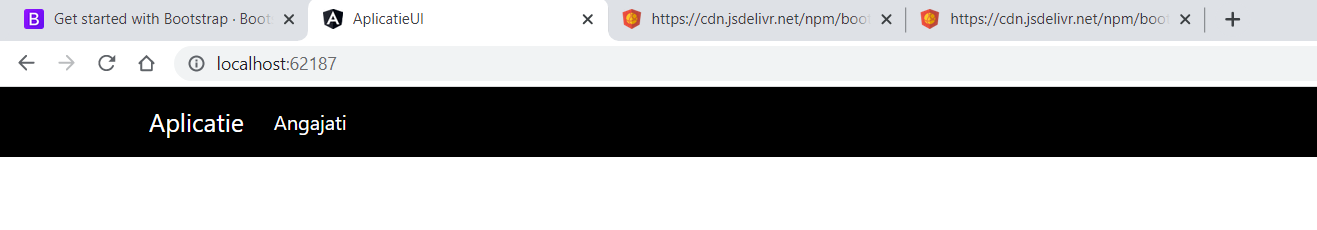


Pentru a include bara de navigare de mai sus in proiectul nostru este necesar sa copiem codul HTML in fisierul app.component.html, deasupra tagului de routare.

Modificati codul html copiat sa arata ca in imaginea de mai jos:



Rezultatul o sa fie urmatorul:



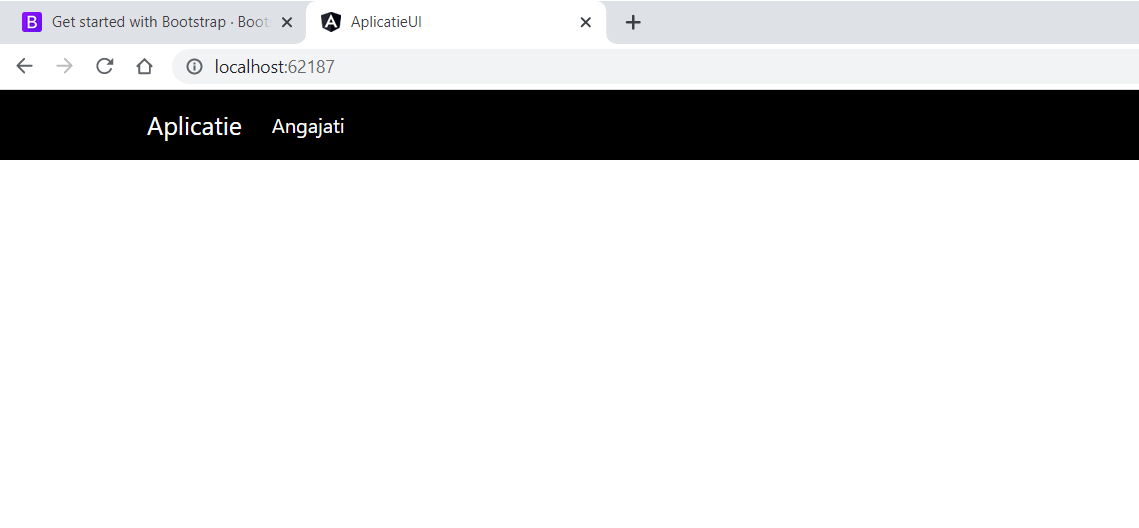
Urmatorul cod trebuie sa fie in app.component.html

|  |
| --- |
| <nav class="navbar navbar-expand-lg navbar-dark bg-black">  <div class="container">  <a class="navbar-brand" href="#">Aplicatie</a>  <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav"  aria-expanded="false" aria-label="Toggle navigation">  <span class="navbar-toggler-icon"></span>  </button>  <div class="collapse navbar-collapse" id="navbarNav">  <ul class="navbar-nav">  <li class="nav-item">  <a class="nav-link active" aria-current="page" href="#">Angajati</a>  </li>    </ul>  </div>  </div>  </nav>  <router-outlet></router-outlet> |

Urmatorul cod trebuie sa fie in index.html

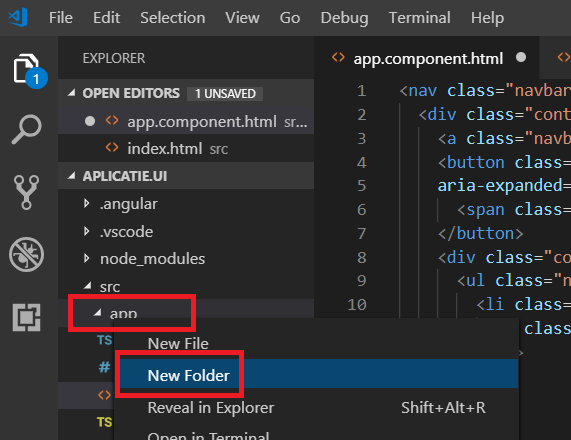
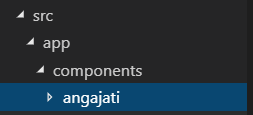
|  |
| --- |
| <!doctype html>  <html lang="en">  <head>  <meta charset="utf-8">  <title>AplicatieUI</title>  <base href="/">  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="icon" type="image/x-icon" href="favicon.ico">    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css"  rel="stylesheet" integrity="sha384-KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJOZ"  crossorigin="anonymous">  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/js/bootstrap.bundle.min.js"  integrity="sha384-ENjdO4Dr2bkBIFxQpeoTz1HIcje39Wm4jDKdf19U8gI4ddQ3GYNS7NTKfAdVQSZe"  crossorigin="anonymous"></script>    </head>  <body>  <app-root></app-root>  </body>  </html> |

Asadar pana in acest moment am realizat un proiect Angular in care am adaugat scripturile de Boostrap si avem o pagina de angajati care este goala.



Ceea ce ne propunem in continuare este sa adaugam un table pentru angajati in pagina de Angajati. Pentru aceasta o sa facem o componenta noua in aplicatie. Pentru aceasta trebuie sa efectuam urmatorii pasi:

Pas1: In folderul de app se vor crea urmatoarele foldere: Components si Angajati, unde angajati este un subfolder pentru folderul Components.

Pas2: Deschidem un nou terminal in Visual Studio Code si navigam spre folderul de Angajati

Text

Description automatically generated

Pas3: Rulam comanda **ng g c angajati-list** pentru a genera componenta angajati. In comanda g=generate si c =componenta

Text

Description automatically generated

Rezultatul executiei comenzii de mai sus este urmatorul:

Text

Description automatically generated

Undem ai jos o sa observam ca avem o noua componenta create cu toate elementele care o compun

Graphical user interface

Description automatically generated with medium confidence

Pas4: Trebuie sa adaugam componenta nou creata in pagina angajati. Pentru aceasta este necasar sa deschidem fisierul app-routing.module.ts iar in contanta routes trebuie sa adaugam un nou obiect, la fel ca in captura de ecran de mai jos:

Text

Description automatically generated

Rezultatul este urmatorul:

Graphical user interface, text, application, chat or text message

Description automatically generated

Practic se va afisa in pagina implicita tot ceea ce este in fisierul html al componentei.

Graphical user interface, application

Description automatically generated

Functionalitatea corecta este ca in momentul in care efectuez click pe Angajati atunci sa fie afisata componenta si nu cand incarc pagina de home. Pentru aceasta specificat o noua ruta in lista de rute si de asemenea o sa fim nevoiti sa modificam in fisierul html al paginii unde avem navigarea catre Angajati sa modificam valoare href si sa o inlocuim cu routeLink=”angajati”, la fel ca in capturile de mai jos.

Text

Description automatically generatedText

Description automatically generated

Vom modifica lista de angajati astfel incat sa avem afisata o lista de angajati sub forma tabelara. Pentru aceata vom efectua urmatorii pasi.

Pas1: Cream un director numit models in proeictul nostru, care va contine un model denumit angajati in care vom declara cateva proprietati pentru modelul nostru.

Graphical user interface, text

Description automatically generated

Pas2: Populam lista cu valori hardcodate. In componenta angajati-list vom crea lista de angajati cu valori hardcodate exact ca in imaginea de mai jos:

Text

Description automatically generated

Pentru simplificare codul din componenta andgajati-list este cel de mai jos:

|  |
| --- |
| import { Component } from '@angular/core';  import{ Angajat } from 'src/app/models/angajat.model'  @Component({  selector: 'app-angajati-list',  templateUrl: './angajati-list.component.html',  styleUrls: ['./angajati-list.component.css']  })  export class AngajatiListComponent {  angajati: Angajat[] = [  {  id:"1",  name:"Nume angajat1",  email:"email@angajat1",  phone: 998877665,  department:"Departament angajat1",  salary:1000  },  {  id:"2",  name:"Nume angajat2",  email:"email@angajat2",  phone: 88554477,  department:"Departament angajat2",  salary:100  },  {  id:"3",  name:"Nume angajat3",  email:"email@angajat3",  phone: 5544778899,  department:"Departament angajat3",  salary:1500  }  ];  } |

Codul pentru modelul de angajat este urmatorul:

|  |
| --- |
| export interface Angajat{  id: string;  name: string;  email:string;  phone:number;  salary:number;  department:string;  } |

Pas3: In componenta angajati in fisierul html o sa adaugam un tabel care va afisa lista noastra creata anterior. Vom adauga urmatorul cod in angajati-list.component.html:

|  |
| --- |
| <div class="container my-5">  <h1 class="mb-3">Angajati</h1>  <table class="table" \*ngIf="angajati && angajati.length >0">  <thead>  <tr>  <th>Id</th>  <th>Name</th>  <th>Email</th>  <th>Phone</th>  <th>Salary</th>  </tr>  </thead>  <tbody>  <tr \*ngFor= "let angajat of angajati">  <td>{{angajat.id}}</td>  <td>{{angajat.name}}</td>  <td>{{angajat.email}}</td>  <td>{{angajat.phone}}</td>  <td>{{angajat.salary}}</td>  </tr>  </tbody>  </table>  <p \*ngIf="!angajati || angajati.length <= 0">Nu sunt angajati</p>  </div> |

Text

Description automatically generated

Rezultatul final este urmatorul:

Graphical user interface, text, application

Description automatically generated

Haideti in cele ce urmeaza sa cream proiectul de API ca sa preluam datele intr-un mod automat si nu hardcodat.

Pentru aceasta deschidem instanta de Visual Studio si vom crea un proiect ASP.NET Core Web API.

Pentru instalarea VS2022 community edition, cea care este si gratuita, o sa fie necesar sa instalam pachetele bifate in cele 2 imagini de mai jos:

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Vom crea mai apoi in folderul de proiect un nou director denumit Aplicatie API.

Graphical user interface, text, application, table, email

Description automatically generated

Vom cream ai apoi un proiect nou in Visual Studio de tipul ASP.NET Core Web API

Text

Description automatically generated

Vom denumi proiectul Aplicatie.API si vom selecta locatia de salvare folderul Aplicatie API creat in pasul anterior.

Graphical user interface, text

Description automatically generated

Vom merge mai departe pe optiunea NEXT cu optiunile selectate mai jos si vom Crea proiectul.

A screenshot of a computer

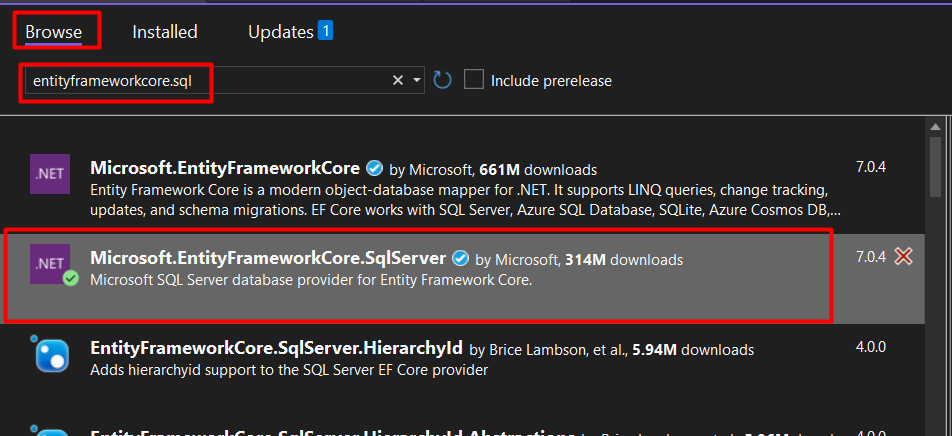
Description automatically generated with medium confidence

In proiectul rezultat trebuie sa instalam doua pachete prin optiunea Manage NuGet Packages pe care o vom accesa efectuand click dreapta pe proiect.

A screenshot of a computer

Description automatically generated with medium confidence

In zona de Browse vom scrie entityframeworkcore.sql si vom instala in proiectul nostru al doilea pachet, cel marcat in imaginea de mai jos.



Mai apoi vom cauta si instala pachetul entityframeworkcore.tools

A screenshot of a computer

Description automatically generated with medium confidence

Haideti in cele ce urmeaza sa cream un folder Models in proiect, in interiorul caruia sa efectuam operatiile de CRUD pentru baza nostra de date.

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Clasa nou creata va contine cele 6 proprietati ale clasei din Angular si anume Id, Name, Email, Phone, Salary, Department.

Clasa va avea urmatorul cod:

|  |
| --- |
| public class Angajat  {  public Guid Id { get; set; }  public string Name { get; set; }  public string Email { get; set; }  public long Phone { get; set; }  public long Salary { get; set; }  public string Department { get; set; }  }  } |



In continuare vom crea Contextul prin intermediul caruia ne vom conecta la baza de date. Pentru aceasta trebuie create un nou folder denumit Data si in interiorul acestuia o sa creati o noua clasa denumita AplicatieDbContext in care o sa adaugati urmatorul cod:

|  |
| --- |
| using Aplicatie.API.Models;  using Microsoft.EntityFrameworkCore;  namespace Aplicatie.API.Data  {  public class AplicatieDbContext : DbContext  {  public AplicatieDbContext(DbContextOptions options) : base(options)  {  }  //definire proprietate care va crea tabel Angajati in baza de date  public DbSet<Angajat> Angajati { get; set; }  }  } |

Pentru a crea stringul de conexiune trebuie sa deschidem Microsoft SQL Server Management Studio si sa obtinem numele serverului. Atentie!!! La dvoastra este cu siguranta o alta valoare:

Graphical user interface, text, application

Description automatically generated

Revenim in Visual Studio iar in fisierul appsettings.json o sa cream un nou field modificand fisierul astfel:

Graphical user interface, text

Description automatically generated

|  |
| --- |
| {  "Logging": {  "LogLevel": {  "Default": "Information",  "Microsoft.AspNetCore": "Warning"  }  },  "AllowedHosts": "\*",  "ConnectionStrings": {  "AngajatiConnectionString": "server=DESKTOP-0RJ742B;Database=AngajatiDb;Trusted\_Connection=True;TrustServerCertificate=True"  }  } |

Mai departe vom face un dependency injection al contextului nostru in serviciul de baze de date in care vom avea ca optiuni sa foloseasca connection stringul creat anterior. Asadar clasa Program.cs o sa aiba urmatorul cod adaugat:

|  |
| --- |
| builder.Services.AddDbContext<AplicatieDbContext>(options =>  options.UseSqlServer(builder.Configuration.GetConnectionString("AngajatiConnectionString"))); |

A screenshot of a computer

Description automatically generated

Totul este in acest moment setat pentru a crea baza de date cu ajutorul Entity Framework Core. Pentru a face acest lucru o sa deschidem consola de NuGet Package Manager:

A screenshot of a computer

Description automatically generated with medium confidence

Vom scrie urmatoarea comanda in consola deschisa:

Graphical user interface, text

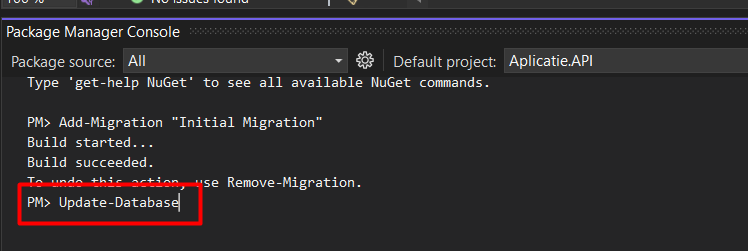
Description automatically generated

Rezultatul rularii acestei comenzi va duce la crearea unui nou folder Migrations in solutia noastra. Acesta va contine o clasa care se vede in imaginea de mai jos, care practic va contine toate comenziile necesare pentru realizarea tabelei Angajati, pe baza clasei create de noi in folderul Models

Text

Description automatically generated with medium confidence

In acest moment daca mergem in SQL Server Management Studio nu o sa avem nici o baza de date creata cu nici un tabel. Pentru aceasta o sa rulam/executam o a doua comanda in consola.



In acest moment daca mergem in SQL Server Management Studio vom observa ca s-a create o baza de date si un tabel conform cerintelor noastre.

Graphical user interface, application

Description automatically generated

In cele ce urmeaza vom prezenta realizarea controller-ului prin intermediul caruia vom realiza operatiile de CRUD pe tabela noastra.

In folderul Controllers vom adauga un nou controller denumit AngajatiController cu optiunea de EmptyController. Forma initiala a controller-ului este ca cea de mai jos. O sa fie necesar sa modificam structura controller-ului astfel:

1. Decoram controllerul cu atributul [ApiController] astfel vom specifica ca acesta nu o sa aiba nici un view.
2. Vom decora metoda cu atributul [HttpGet] specificand ca aceasta metoda o sa produca un Get pe toti angajatii.
3. Aceeasi clasa o sa o decoram cu atributul [Route(“api/[controller]”)] pentru a specifica ruta de url ca va face ca acest controller sa raspunda
4. Schimbam numele metodei din controller cu GetAllAngajati in care vom utiliza contextual pentru a comunica cu baza de date prin Entity Framework. Controller-ul o sa aiba forma din imaginea de mai jos. Mai jos, de asemenea, este prezentat tot codul pentru Controllerul creat.

Text

Description automatically generated

|  |
| --- |
| using Aplicatie.API.Data;  using Microsoft.AspNetCore.Mvc;  using Microsoft.EntityFrameworkCore;  namespace Aplicatie.API.Controllers  {  [ApiController]  [Route("api/[controller]")]  public class AngajatiController : Controller  {  private readonly AplicatieDbContext \_aplicatieDbContext;  public AngajatiController(AplicatieDbContext aplicatieDbContext)  {  \_aplicatieDbContext= aplicatieDbContext;  }  [HttpGet]  public async Task<IActionResult> GetAllAngajati()  {  var angajati = await \_aplicatieDbContext.Angajati.ToListAsync();  return Ok(angajati);  }  }  } |

Daca rulam aplicatia o sa observam ca se va deschide o aplicatie Swagger care prezinta care sunt API-urile disponibile spre apelare.

Graphical user interface, application

Description automatically generated

In imaginea de mai sus am executat un astfel de call si am primit ca raspuns Code 200, adica success. Chiar daca in acest moment nu avem angajati in tabela, raspunsul a fost success.

Haideti sa cream o noua metoda de adaugare angajat. Aceasta o sa fie o metoda de tip HttpPost. Metoda o sa fie cea descrisa mai jos:

|  |
| --- |
| [HttpPost]  public async Task<IActionResult> AddAngajat([FromBody] Angajat angajatRequest)  {  angajatRequest.Id = Guid.NewGuid();  await \_aplicatieDbContext.Angajati.AddAsync(angajatRequest);  await \_aplicatieDbContext.SaveChangesAsync();  return Ok(angajatRequest);  } |

Cum verificam faptul ca functionalitatea de adaugare merge correct?

Rulam aplicatia si facem un call in Swagger.

Graphical user interface, application, email

Description automatically generated

Background pattern

Description automatically generated

Daca verificam dupa executia call-ului in baza de date, putem observa faptul ca avem o intrare in tabela noastra.

Graphical user interface, text, application

Description automatically generated

Pana in acest moment am realizat metodele pentru operatiile de CREATE si READ. Haideti sa ne intoarcem inapoi in aplicatia Angular si sa modificam lista statica sa aduca inregistrarile din baza de date.

Ne aflam in componenta angajati-list unde in acest moment avem o lista statica de angajati. Text

Description automatically generated

Pentru a comunica cu API-ul nostru extern o sa ne folosim de servicii. Pentru a crea un nou serviciu o sa cream un nou folder in app, denumit services. Dorim mai apoi sa cream un nou serviciu ruland comanda

**ng g s angajati**  fiind localizati in folderul services in momentul rularii acestei comenzi.

Text

Description automatically generated

Dupa rularea cu success a acestei comenzi o sa avem adaugat un nou serviciu in folderul de servicii. Prin intermediul acestui serviciul vom comunica cu API-ul care adauga sau returneaza inregistrarile angajati din baza de date.

Text

Description automatically generated

Haideti sa adaugam o noua metoda care va apela aceste API-uri. Inainte de acest lucru vom adauga in tsconfig.json urmatorul cod:

Text

Description automatically generated

Astfel o sa facem accesibila comunicatia cu exteriorul printr-un client https. Mai jos aveti codul care este scris in serviciul angajati pentru a prelua toti angajati din tabela bazei de date.

|  |
| --- |
| import { Injectable } from '@angular/core';  import {HttpClient} from '@angular/common/http';  import { Angajat } from '../models/angajat.model';  //import {Observable} from 'rxjs';  @Injectable({  providedIn: 'root'  })  export class AngajatiService {  baseApiUrl: string = "https://localhost:7138/"  constructor(private http:HttpClient) { }  getAllAngajati(){  return this.http.get<Angajat[]>(this.baseApiUrl + '/api/angajati');  }  } |

In componenta noastra unde avem lista hardcodata de angajati, vom sterge aceasta lista si vom avea codul de mai jos

|  |
| --- |
| import { Component, OnInit } from '@angular/core';  import{ Angajat } from 'src/app/models/angajat.model'  @Component({  selector: 'app-angajati-list',  templateUrl: './angajati-list.component.html',  styleUrls: ['./angajati-list.component.css']  })  export class AngajatiListComponent {  angajati: Angajat[] = [];  ngOnInit():void{    }  } |

Daca vom face refresh la aplicatie acum, o sa observam ca nu vom mai avea tabelul afisat, ci o sa fie afisat paragraful care ne atentioneaza ca nu avem angajati disponibili spre afisare.

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Corectare eroare CORS Cross Origin Request Blocked

Browserul nu permite ca de la portul 4200 sa mergem la cel de la API. Vrem ca API-ul sa spuna cine poate sa consume requesturile noastre.

Graphical user interface, application

Description automatically generated

Asadar in Visual Studio proiectul de API, in Program.cs trebuie sa dam enable la CORS.

Text

Description automatically generated

Dupa ce am dat enable la CORS specificand ca orice header, orice metoda din orice origine poate sa faca call la API, raspunsul cand executam refresh la pagina Angular este urmatorul:

Graphical user interface, text, application

Description automatically generated

Acest raspuns este vizibil numai in Consola de dezvoltator din Chrome. Ce trebuie sa facem pentru a vedea acest angajat in pagina de Angajati? O sa ne intoarcem in componenta Angajati-list si unde avem acum scrierile in consola, o sa modificam codul astfel:

Text

Description automatically generated

Rezultatul obtinut este urmatorul:

Text

Description automatically generated with low confidence

Haideti in cele ce urmeaza sa apelam API-ul de adaugare Angajat. Pentru acesta efectuam urmatorii pasi:

Pas1: Deschidem o noua fereastra terminal

Pas2: Navigam in sectiunea components\angajati

Text

Description automatically generated

Pas3: Rulam comanda ng g c add-angajati

Text

Description automatically generated

Pas4: Adaugam o noua ruta pentru aceasta componenta, astfel ca in app-rounting.module.ts vom adauga o noua ruta care apelata va afisa componenta create anterior.

Text

Description automatically generated

Codul necesar pentru noua ruta este cel de mai jos:

|  |
| --- |
| import { NgModule } from '@angular/core';  import { RouterModule, Routes } from '@angular/router';  import { AngajatiListComponent } from './components/angajati/angajati-list/angajati-list.component';  import { AddAngajatiComponent } from './components/angajati/add-angajati/add-angajati.component';  const routes: Routes = [      {      path:'angajati',      component:AngajatiListComponent    },    {      path:'angajati/add',      component:AddAngajatiComponent    }  ];  @NgModule({    imports: [RouterModule.forRoot(routes)],    exports: [RouterModule]  })  export class AppRoutingModule { } |

Pas5: Adaugam o noua intrare in meniul de navigare a aplicatiei, codul este cel de mai jos:

|  |
| --- |
| <nav class="navbar navbar-expand-lg navbar-dark bg-black">    <div class="container">      <a class="navbar-brand" href="#">Aplicatie</a>      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav"      aria-expanded="false" aria-label="Toggle navigation">        <span class="navbar-toggler-icon"></span>      </button>      <div class="collapse navbar-collapse" id="navbarNav">        <ul class="navbar-nav">          <li class="nav-item">            <a class="nav-link active" aria-current="page" routerLink="angajati">Angajati</a>          </li>          <li class="nav-item">            <a class="nav-link active" aria-current="page" routerLink="angajati/add">Adaugare Angajati</a>          </li>          </ul>      </div>    </div>  </nav>  <router-outlet></router-outlet> |

Rezultatul asteptat este urmatorul:

Graphical user interface, text, application

Description automatically generated

Haideti in cele ce urmeaza sa modificam aceasta componenta sa nu afiseze ce avem mai sus, ci sa afiseze un formular cu ajutorul caruia sa putem adauga noi angajati in tabela bazei de date. Pentru aceasta vom accesta vom deschide in Visual Studio Code componenta add-angajati.component.html si vom adauga urmatorul cod:

|  |
| --- |
| <div class="container my-5">      <h1 class="mb-3">Adaugare Angajat Nou</h1>      <div class="row">          <div class="col-6">              <form>                  <div class="mb-3">                    <label for="name">Nume</label>                    <input type="text" class="form-control" id="name">                  </div>                    <div class="mb-3">                      <label for="email">Email</label>                      <input type="email" class="form-control" id="email">                    </div>                    <div class="mb-3">                      <label for="telefon">Telefon</label>                      <input type="number" class="form-control" id="telefon">                    </div>                    <div class="mb-3">                      <label for="salar">Salar</label>                      <input type="number" class="form-control" id="salar">                    </div>                    <div class="mb-3">                      <label for="departament">Departament</label>                      <input type="text" class="form-control" id="departament">                    </div>                  <button type="submit" class="btn btn-primary">Save</button>                </form>          </div>      </div>    </div> |

Rezultatul este cel de mai jos:

Graphical user interface, application

Description automatically generated

In acest moment urmeaza sa modificam acest formular astfel incat sa facem legaturile cu API-ul de adaugare. Pentru aceasta o sa facem urmatoarele operatii.

1. Vom adauga tagul form formularului nostru

A screenshot of a computer

Description automatically generated with medium confidence

1. Datorita faptului ca folosim valoare ngForm pentru tageul form, este necesar sa agaugam modului de Forms in fisierul app-module.ts

Text

Description automatically generated

1. Mai apoi trebuie sa cream un model pentru a putea lega campurile din formulat la proprietatiile modelului. Pentru aceasta vom accesa add-angajati.component.ts si vom crea o noua variabila publica

Text

Description automatically generated

1. Trebuie sa legat campurile din formular cu proprietatiile noului obiect. Aceasta legatura se face dupa modelul urmator. Campul din form o sa aiba un noua proprietate [(ngModel)]=obiect.proprietate. Dupa modificarile efectuate in form, acesta va avea urmatoarea forma:

|  |
| --- |
| <div class="container my-5">      <h1 class="mb-3">Adaugare Angajat Nou</h1>      <div class="row">          <div class="col-6">              <form #form="ngForm" (ngSubmit)="addAngajat()">                  <div class="mb-3">                    <label for="name">Nume</label>                    <input type="text" class="form-control" id="name"                    name ="name" [(ngModel)]="addAngajatRequest.name">                  </div>                    <div class="mb-3">                      <label for="email">Email</label>                      <input type="email" class="form-control" id="email"                      name ="email" [(ngModel)]="addAngajatRequest.email">                    </div>                    <div class="mb-3">                      <label for="telefon">Telefon</label>                      <input type="number" class="form-control" id="telefon"                      name="telefon" [(ngModel)]="addAngajatRequest.phone">                    </div>                    <div class="mb-3">                      <label for="salar">Salar</label>                      <input type="number" class="form-control" id="salar"                      name="salar" [(ngModel)]="addAngajatRequest.salary">                    </div>                    <div class="mb-3">                      <label for="departament">Departament</label>                      <input type="text" class="form-control" id="departament"                      name="departament" [(ngModel)]="addAngajatRequest.department">                    </div>                  <button type="submit" class="btn btn-primary">Save</button>                </form>          </div>      </div>    </div> |

1. Avand elementele legate de proprietary, urmeaza sa setam ca atunci cand formularul este salvat sa apelam o actiune din fisierul de typescript.

Codul pentru formular:  <form #form="ngForm" (ngSubmit)="addAngajat()">

Metoda addAngajat() o sa fie adaugata in fisierul add-angajat.component.ts. Pentru moment aceasta o sa fie goala sau cel mult va afisa rezultatul in consola.

Text

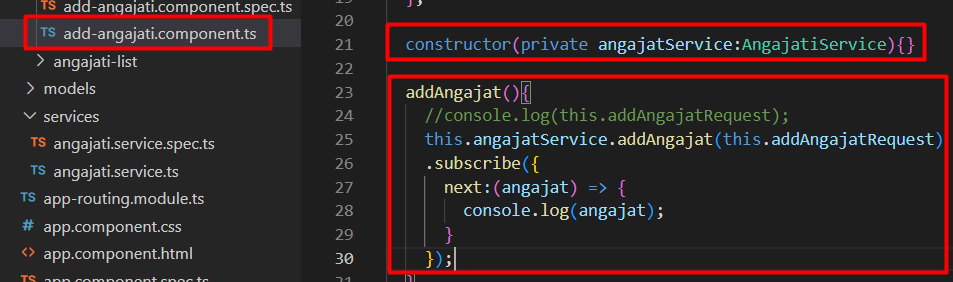
Description automatically generated

Aceasta metoda va apela o metoda din serviciul existent in aplciatie. Aceasta va transmite datele catre API-ul create in Visual Studio. Haideti in cele ce urmeaza sa cream o noua metoda in acest serviciu. Pentru acesta o sa accesam folderul services si vom accesa angajati.service.ts.

Text

Description automatically generated

Pentru a consuma aceasta metoda din serviciu, vom adauga urmatorul cod in metoda addAngajat din add-angajat.component.ts



Rezultatul este urmatorul:

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

Timeline

Description automatically generated with medium confidence

Pentru a ne redirecta spre pagina de angajati in momentul in care s-a efectuat o inserare cu success, nu vom mai afisa raspunsul in consola si o sa adaugam urmatorul cod in componenta add-angajat.component.ts. Vom injecta in controller un obiect de tip Router si vom scrie o comanda de navigate in caz de success. Codul pentru add-angajati.component.ts este urmatorul:

|  |
| --- |
| import { Component } from '@angular/core';  import { Angajat } from 'src/app/models/angajat.model';  import { AngajatiService } from 'src/app/services/angajati.service';  import {Router} from '@angular/router'  @Component({    selector: 'app-add-angajati',    templateUrl: './add-angajati.component.html',    styleUrls: ['./add-angajati.component.css']  })  export class AddAngajatiComponent {    addAngajatRequest: Angajat={      id:'',      name:'',      email:'',      phone:0,      salary:0,      department:''    };    constructor(private angajatService:AngajatiService, private router: Router){}    addAngajat(){      //console.log(this.addAngajatRequest);      this.angajatService.addAngajat(this.addAngajatRequest)      .subscribe({        next:(angajat) => {          //console.log(angajat);        this.router.navigate(['angajati']);        }      });    }  } |

Urmatorul pas este sa implementam un buton de vizualizare, modificare si stergere a unei inregistrari. Pentru functionalitatea de editare vom crea o noua componenta in fisierul de components al aplicatiei.

Text

Description automatically generated

Vom adauga o noua ruta pentru a lega componenta de viitorul element de navigare din meniu:

Text

Description automatically generated

Vom adauga in tabelul de angajati, pentru fiecare inregistrare, un buton care ne va afisa componenta de editare:

Text

Description automatically generated

Astfel tabelul se va modifica precum evidentiat in captura de ecran de mai jos:

Graphical user interface, text, application

Description automatically generated

Daca vom efectua click pe unul din angajati o sa fim rutati catre pagina de editare pentru angajatul respectiv. In acest moment o sa fie afisata componenta goala, doar cu paragraful standard, la fel ca in imaginea de mai jos:

Graphical user interface, text, application, website

Description automatically generated

Pentru a popula fisierul html al acestei component o sa copiem din fisierul html al add-angajat si vom efectua unele modificari. Codul din pagina html de editare o sa fie cel de mai jos:

|  |
| --- |
| <div class="container my-5">      <h1 class="mb-3">Editare Angajat</h1>      <div class="row">          <div class="col-6">              <form #form="ngForm">                  <div class="mb-3">                      <label for="id">Id</label>                      <input type="text" class="form-control" id="id" readonly                      name ="id" [(ngModel)]="detaliiAngajat.id">                    </div>                  <div class="mb-3">                    <label for="name">Nume</label>                    <input type="text" class="form-control" id="name"                    name ="name" [(ngModel)]="detaliiAngajat.name">                  </div>                    <div class="mb-3">                      <label for="email">Email</label>                      <input type="email" class="form-control" id="email"                      name ="email" [(ngModel)]="detaliiAngajat.email">                    </div>                    <div class="mb-3">                      <label for="telefon">Telefon</label>                      <input type="number" class="form-control" id="telefon"                      name="telefon" [(ngModel)]="detaliiAngajat.phone">                    </div>                    <div class="mb-3">                      <label for="salar">Salar</label>                      <input type="number" class="form-control" id="salar"                      name="salar" [(ngModel)]="detaliiAngajat.salary">                    </div>                    <div class="mb-3">                      <label for="departament">Departament</label>                      <input type="text" class="form-control" id="departament"                      name="departament" [(ngModel)]="detaliiAngajat.department">                    </div>                  <button type="submit" class="btn btn-primary">Save</button>                </form>          </div>      </div>    </div> |

Text

Description automatically generated

Metoda de get din webAPI este urmatoarea(ATENTIE: este creata in Visual Studio)

|  |
| --- |
| [HttpGet]  [Route("{id:Guid}")]  public async Task<IActionResult> GetAngajat([FromRoute]Guid id)  {  var angajat = await \_aplicatieDbContext.Angajati.FirstOrDefaultAsync(x => x.Id == id);  if(angajat == null)  {  return NotFound();  }    return Ok(angajat);  } |

Vom apela aceasta metoda prin intermediul unui metode continuta de serviciul de comunicare din aplicatia Angular. Asadar in angajati.service.ts vom crea o noua metoda care va avea urmatoarea forma.

|  |
| --- |
| getAngajat(id: string):Observable<Angajat>{      return this.http.get<Angajat>(this.baseApiUrl+'api/angajati/'+ id);    } |

Vom consuma aceasta metoda din serviciu in componenta edit-angajat.component.ts astfel:

Text

Description automatically generated

Mai jos aveti tot codul din aceasta componenta:

|  |
| --- |
| import { Component } from '@angular/core';  import { ActivatedRoute } from '@angular/router';  import { Angajat } from 'src/app/models/angajat.model';  import { AngajatiService } from 'src/app/services/angajati.service';  @Component({    selector: 'app-edit-angajat',    templateUrl: './edit-angajat.component.html',    styleUrls: ['./edit-angajat.component.css']  })  export class EditAngajatComponent {    detaliiAngajat: Angajat={      id: '',      name:'',      email:'',      phone:0,      salary:0,      department:''    };    constructor(private route:ActivatedRoute, private angajatiService:      AngajatiService){  }      ngOnInit(): void{      this.route.paramMap.subscribe({        next:(params)=>{          const id = params.get('id');          if(id) {            //call api            //aici facem call catre functia din webAPI care sa ne returneze datele            //angajatului cu id=ul = xxxxxxxxxxxxxxxxxxxxxxx            this.angajatiService.getAngajat(id)            .subscribe({              next: (response)=>{                this.detaliiAngajat=response;              }            })          }        }      })    }  } |

Rezultatul este urmatorul:

Graphical user interface, application

Description automatically generated

Graphical user interface

Description automatically generated

Formularul este acum creat si completat. Pentru a pune in evidenta functionalitatea de UPDATE o sa fie necesar sa legat o actiune de butonul SAVE din formularul precedent creat. Pentru aceasta o sa decoram formularul cu atributul (ngSubmit)=”metodaUpdate()” lucru care nu face altceva decat sa specifice faptul ca in momentul salvarii acestui formular, sa se apeleze metoda specificata.

Formularul din html o sa contina randul de mai jos ca modificare:

|  |
| --- |
| <form #form="ngForm" (ngSubmit)="updateAngajat()"> |

Metoda de update este cea de mai jos si este inclusa in componenta edit-angajat:

|  |
| --- |
| import { Component } from '@angular/core';  import { ActivatedRoute, Router } from '@angular/router';  import { Angajat } from 'src/app/models/angajat.model';  import { AngajatiService } from 'src/app/services/angajati.service';  @Component({    selector: 'app-edit-angajat',    templateUrl: './edit-angajat.component.html',    styleUrls: ['./edit-angajat.component.css']  })  export class EditAngajatComponent {    detaliiAngajat: Angajat={      id: '',      name:'',      email:'',      phone:0,      salary:0,      department:''    };    constructor(private route:ActivatedRoute, private angajatiService:      AngajatiService, private router: Router){  }    ngOnInit(): void{      this.route.paramMap.subscribe({        next:(params)=>{          const id = params.get('id');          if(id) {            //call api            //aici facem call catre functia din webAPI care sa ne returneze datele            //angajatului cu id=ul = xxxxxxxxxxxxxxxxxxxxxxx            this.angajatiService.getAngajat(id)            .subscribe({              next: (response)=>{                this.detaliiAngajat=response;              }            })          }        }      })    }    updateAngajat(){      this.angajatiService.updateAngajat(this.detaliiAngajat.id, this.detaliiAngajat)      .subscribe({        next:(response) =>{          this.router.navigate(['angajati']);        }      });    }  } |

Metoda din serviciul care comunica cu API-ul este cea de mai jos:

|  |
| --- |
| updateAngajat(id:string, updateAngajat: Angajat): Observable<Angajat>{      return this.http.put<Angajat>(this.baseApiUrl+'api/angajati/'+id,      updateAngajat);    } |

Metoda de WebAPI de update este urmatoarea:

|  |
| --- |
| [HttpPut]  [Route("{id:Guid}")]  public async Task<IActionResult> UpdateAngajat([FromRoute] Guid id, Angajat angajatDeModificat)  {  var angajat = await \_aplicatieDbContext.Angajati.FindAsync(id);  if (angajat == null)  {  return NotFound();  }  angajat.Name = angajatDeModificat.Name;  angajat.Email = angajatDeModificat.Email;  angajat.Salary = angajatDeModificat.Salary;  angajat.Phone = angajatDeModificat.Phone;  angajat.Department = angajatDeModificat.Department;  \_aplicatieDbContext.SaveChangesAsync();  return Ok(angajat);  } |

Rezultatul dupa aceste modificari este urmatorul

Graphical user interface, text, application

Description automatically generated 

Pentru ultima operatie, cea de delete codul pentru fiecare din solutii este urmatorul:

Codul pentru WebAPI, metoda de detele este urmatorul:

|  |
| --- |
| [HttpDelete]  [Route("{id:Guid}")]  public async Task<IActionResult> DeleteAngajat([FromRoute] Guid id)  {  var angajat = await \_aplicatieDbContext.Angajati.FindAsync(id);  if (angajat == null)  {  return NotFound();  }  \_aplicatieDbContext.Remove(angajat);  await \_aplicatieDbContext.SaveChangesAsync();  return Ok(angajat);  } |

Metoda de comunicare cu WebAPI, din serviciul Angular este urmatoarea:

|  |
| --- |
| deleteAngajat(id: string): Observable<Angajat>{      return this.http.delete<Angajat>(this.baseApiUrl+'api/angajati/'+ id);    }  } |

Metoda de adaugat in componenta de edit-angajat cea care va face call la metoda de mai sus din serviciu , este urmatoarea:

|  |
| --- |
| deleteAngajat(id: string){      this.angajatiService.deleteAngajat(id)      .subscribe({        next:(response) =>{          this.router.navigate(['angajati']);        }      });    } |

Codul care se afiseaza in componenta html este urmatorul:

A picture containing text

Description automatically generated

|  |
| --- |
| <div class="container my-5">      <h1 class="mb-3">Editare Angajat</h1>      <div class="row">          <div class="col-6">              <form #form="ngForm" (ngSubmit)="updateAngajat()">                  <div class="mb-3">                      <label for="id">Id</label>                      <input type="text" class="form-control" id="id" readonly                      name ="id" [(ngModel)]="detaliiAngajat.id">                    </div>                  <div class="mb-3">                    <label for="name">Nume</label>                    <input type="text" class="form-control" id="name"                    name ="name" [(ngModel)]="detaliiAngajat.name">                  </div>                    <div class="mb-3">                      <label for="email">Email</label>                      <input type="email" class="form-control" id="email"                      name ="email" [(ngModel)]="detaliiAngajat.email">                    </div>                    <div class="mb-3">                      <label for="telefon">Telefon</label>                      <input type="number" class="form-control" id="telefon"                      name="telefon" [(ngModel)]="detaliiAngajat.phone">                    </div>                    <div class="mb-3">                      <label for="salar">Salar</label>                      <input type="number" class="form-control" id="salar"                      name="salar" [(ngModel)]="detaliiAngajat.salary">                    </div>                    <div class="mb-3">                      <label for="departament">Departament</label>                      <input type="text" class="form-control" id="departament"                      name="departament" [(ngModel)]="detaliiAngajat.department">                    </div>                  <button type="submit" class="btn btn-primary">Save</button>                  <button type="button" class="btn btn-danger"                  (click)="deleteAngajat(detaliiAngajat.id)">Delete</button>                </form>          </div>      </div>  </div> |

Rezultatul este urmatorul:

Graphical user interface, text, application

Description automatically generated

Rezultatul este faptul ca inregistrarea de mai sus s-a sters:

Graphical user interface, text, application, email

Description automatically generated