**Angular 19 CRUD cu .NET9 Web API si AWS DynamoDB**

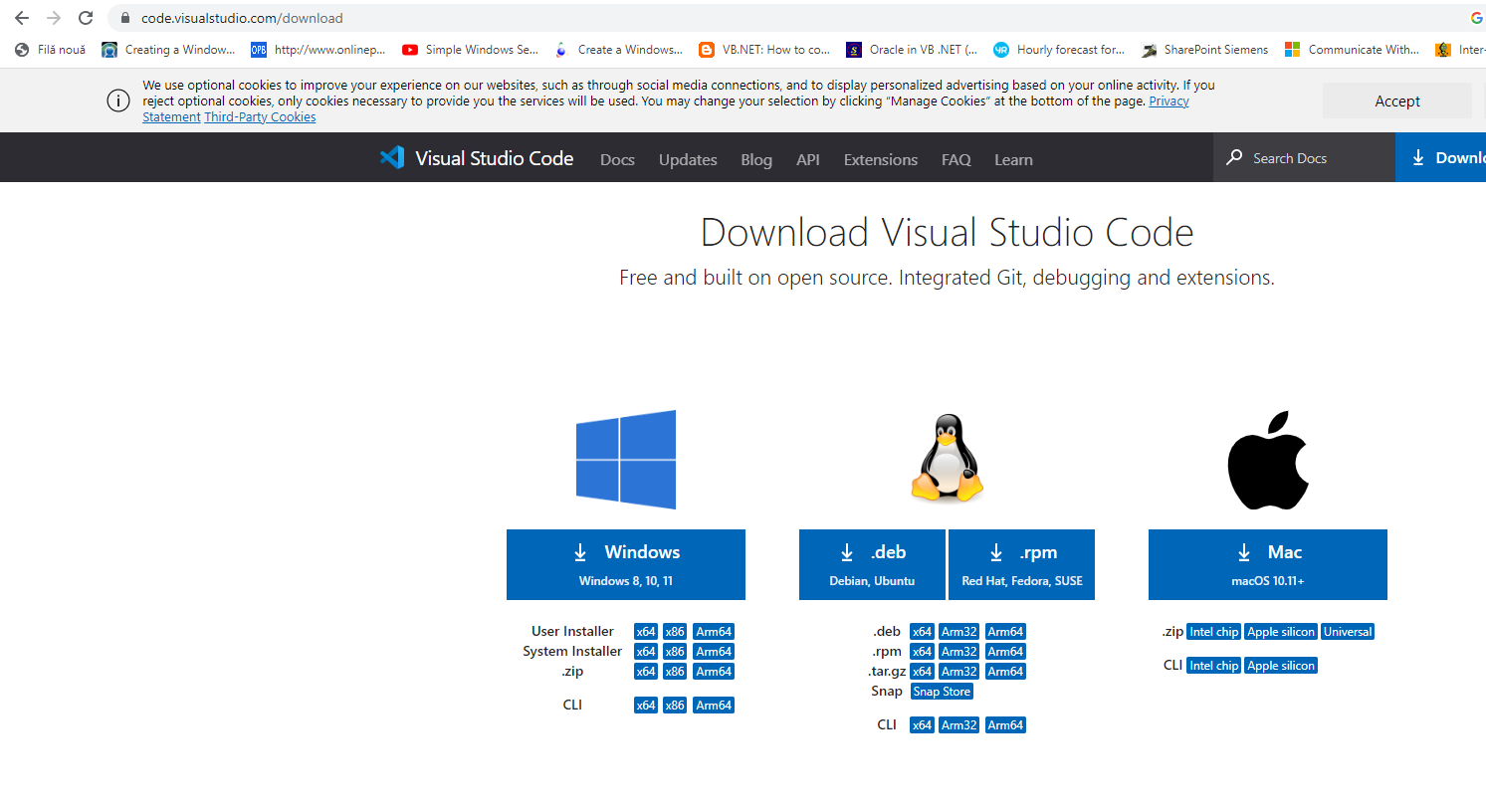
# Scopul laboratorului

Crearea unei aplicatii WEB folosind Angular pentru frontend, .NET9 pentru partea de backend si AWS DynamoDB (NoSQL database) pentru stocarea datelor.

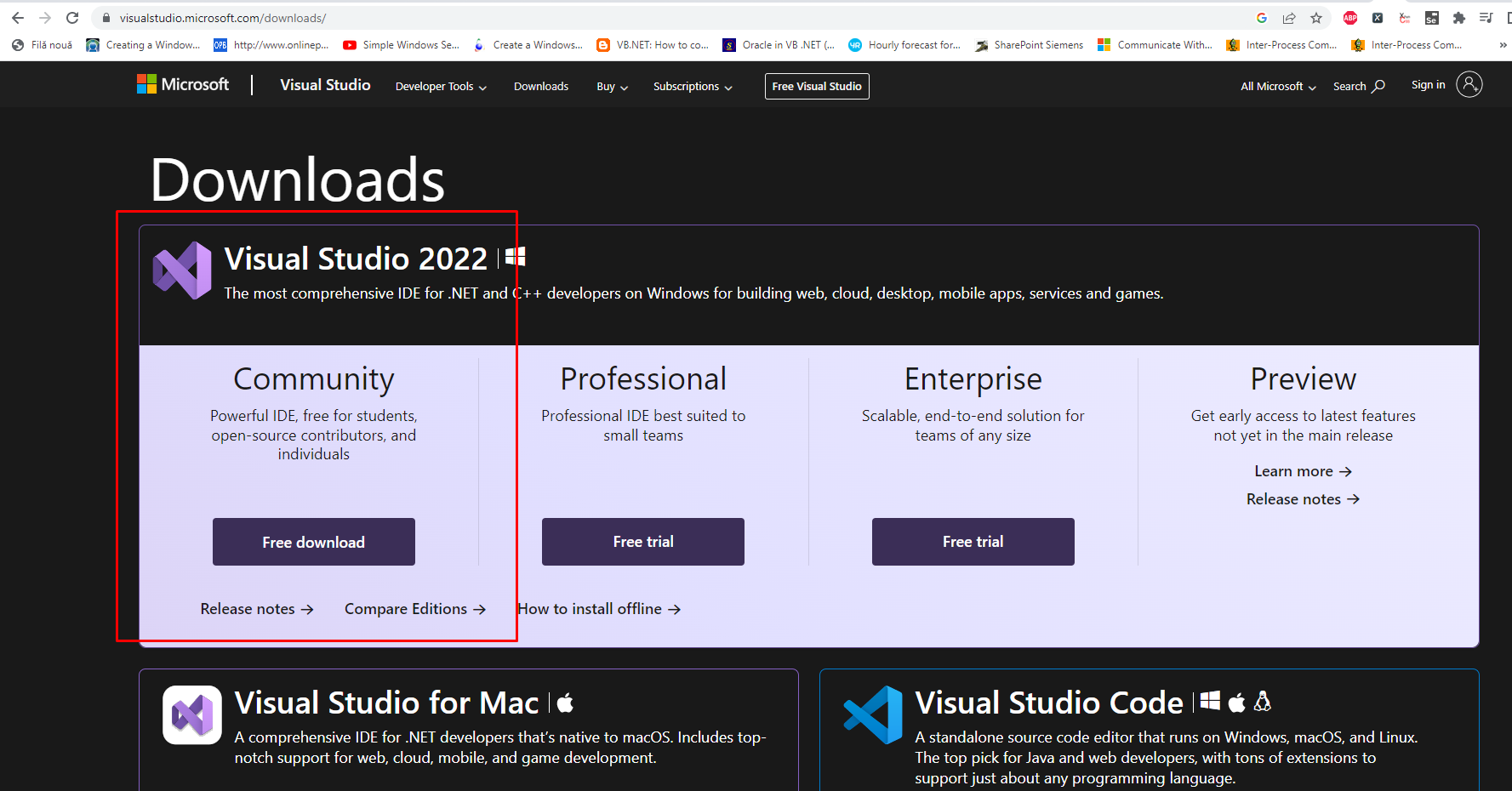
Structura laboratorului:

1. Instalarea pachetelor software necesare.
2. Crearea unei aplicatii .NET care va face publice endpointuri pentru a lista niste produse hardcodate si crearea unei aplicatii angular care va afisa aceste date.
3. Crearea unui cont AWS si conectarea aplicatiei .NET la serviciul AWS.
4. Imbunatatirea atat a aplicatiei Angular cat si a celei de .NET pentru a indeplini functiile de UPDATE, CREATE si DELETE.

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



Pas2: Descarcati Visual Studio (pentru partea de backend)



**Vom instala Visual studio intr-un pas ulterior!**

Pas3: Verificati versiunea de .NET instalata pe calculator

A black background with white text

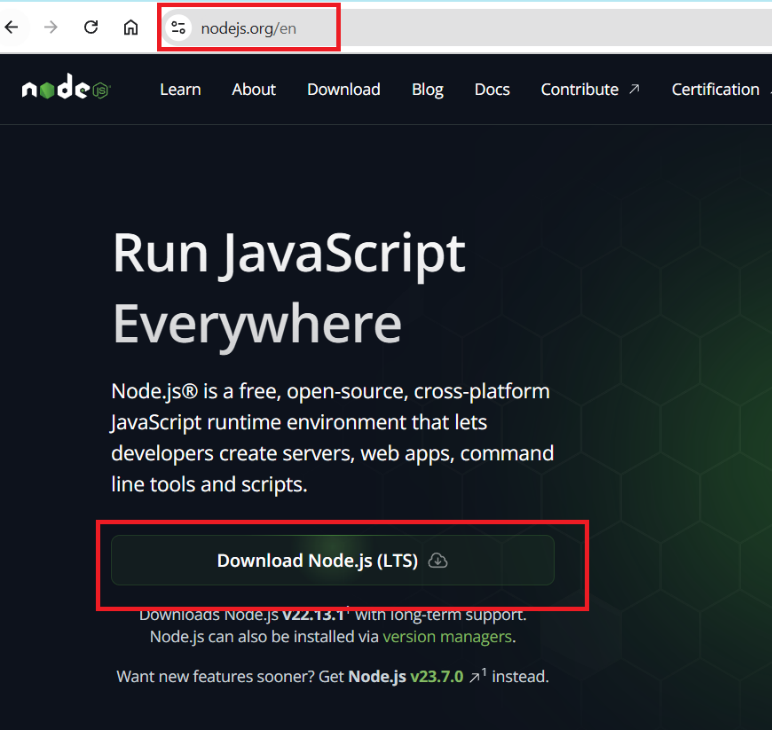
AI-generated content may be incorrect.

Pas2: Instalati ultima versiune de .NET

A screenshot of a computer

AI-generated content may be incorrect.

Pas4: Instalati nodeJS



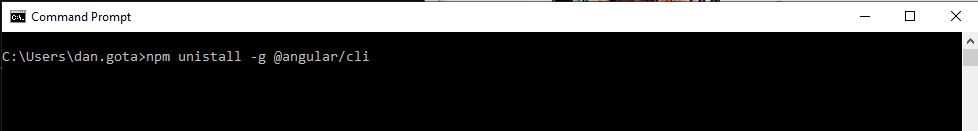
Pas5: Verificare versiune nodeJS

A black background with white text

AI-generated content may be incorrect.

Pas6: Instalare Angular CLI

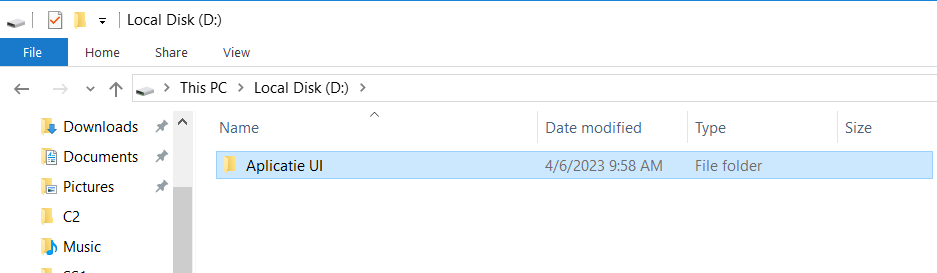
Mai intai dezinstalati versiunea curenta de Angular



Mai apoi instalati global ultima varianta de Angular



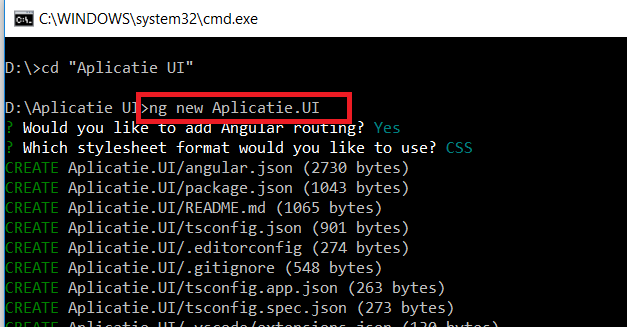
Pas7: Creati pe disc un folder denumit Aplicatie UI



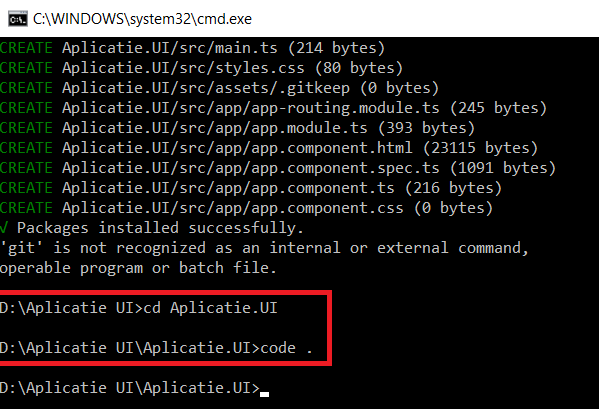
Pas8: Creati in aceasta locatie un proiect angular denumit Aplicatie.UI direct din consola de comanda, utilizand comanda: ng new Aplicatie.UI

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

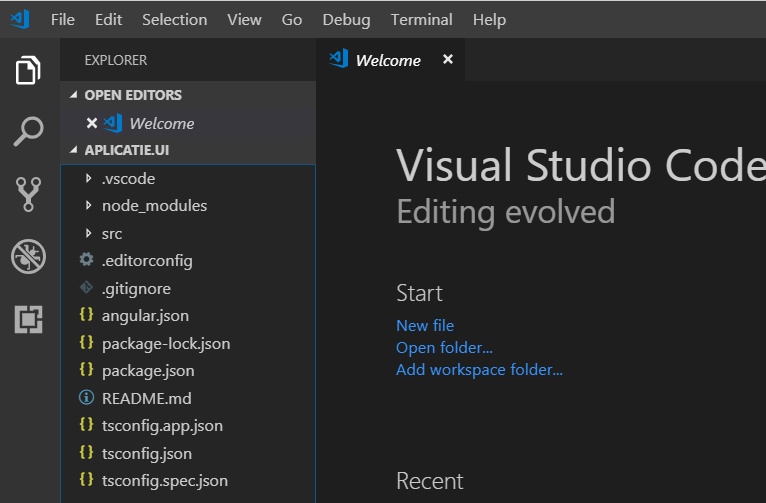
**Which style sheet format would you like to use? CSS**



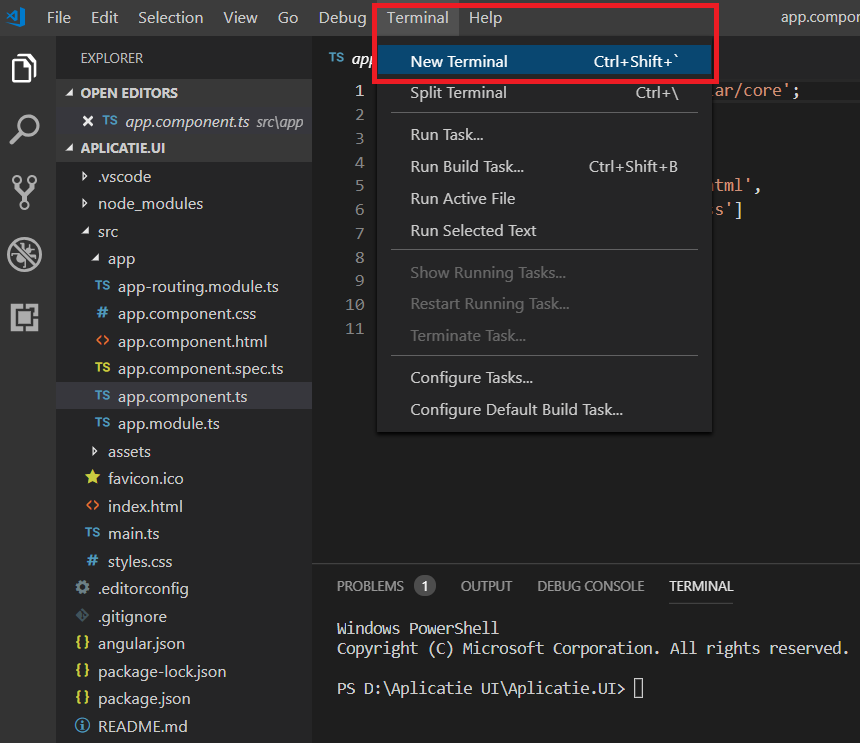
Pas9: Deschideti proiectul de UI cu Visual Studio Code prin scrierea comenzii **code .** in linia de comanda



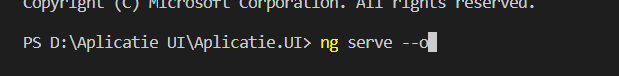
Ruland aceasta comanda o sa se deschida aplicatia de UI in aplicatia Visula Studio Code.



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.

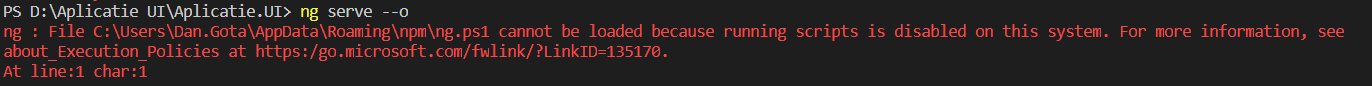
****

**Rezultatul este urmatorul:**

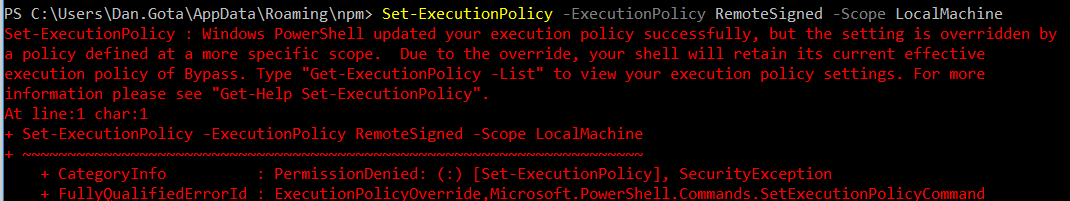
**A screenshot of a computer

AI-generated content may be incorrect.**

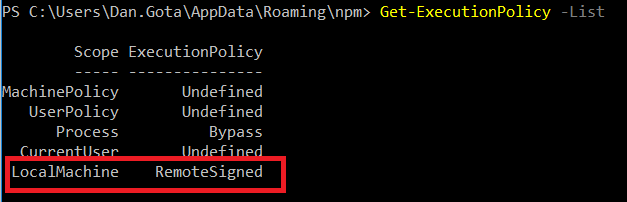
Daca la rularea comenzii ng serve - - o apare urmatoarea eroare:



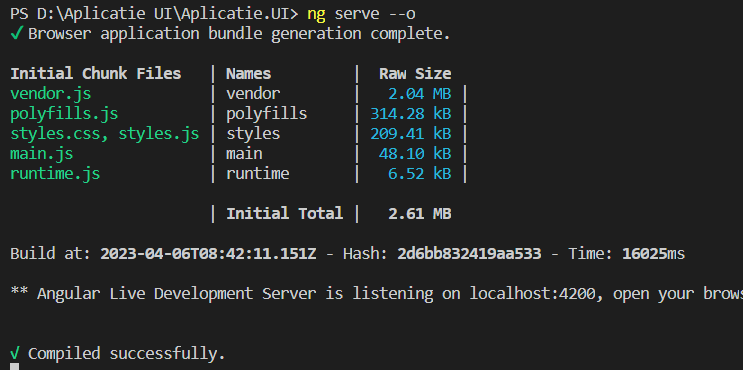
Aceasta se rezolva deschizand in modul administrator Command Prompt si Ruland urmatoarea comanda:



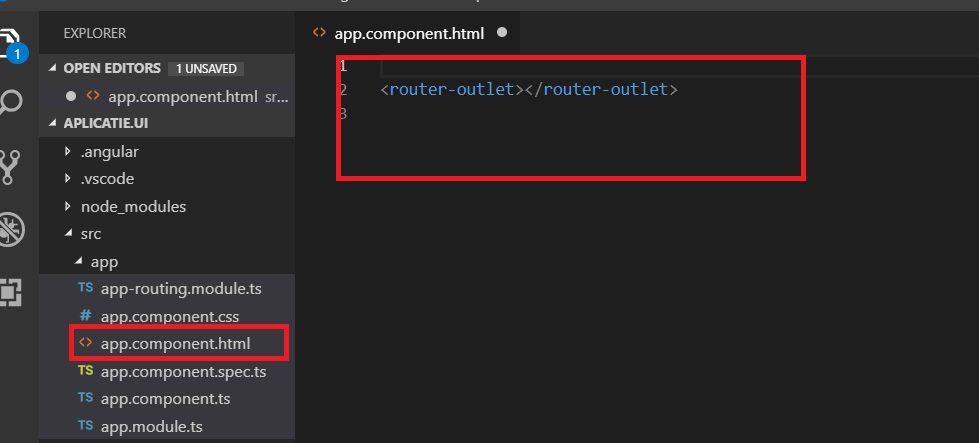
Mai apoi verificati faptul ca pe masina locala s-a setat correct politica de executie:



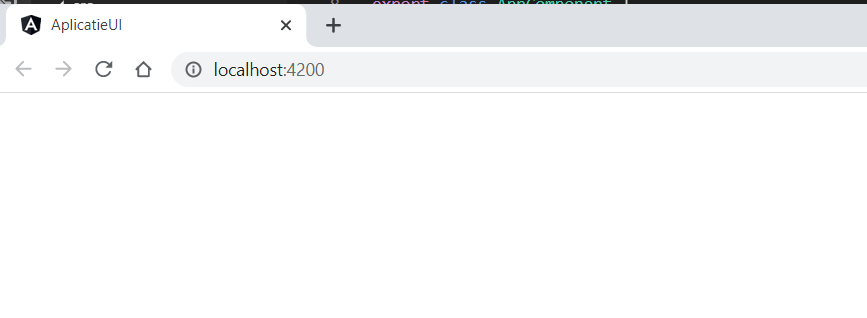
Mai apoi se va rula cu success comanda ng serve - -o.



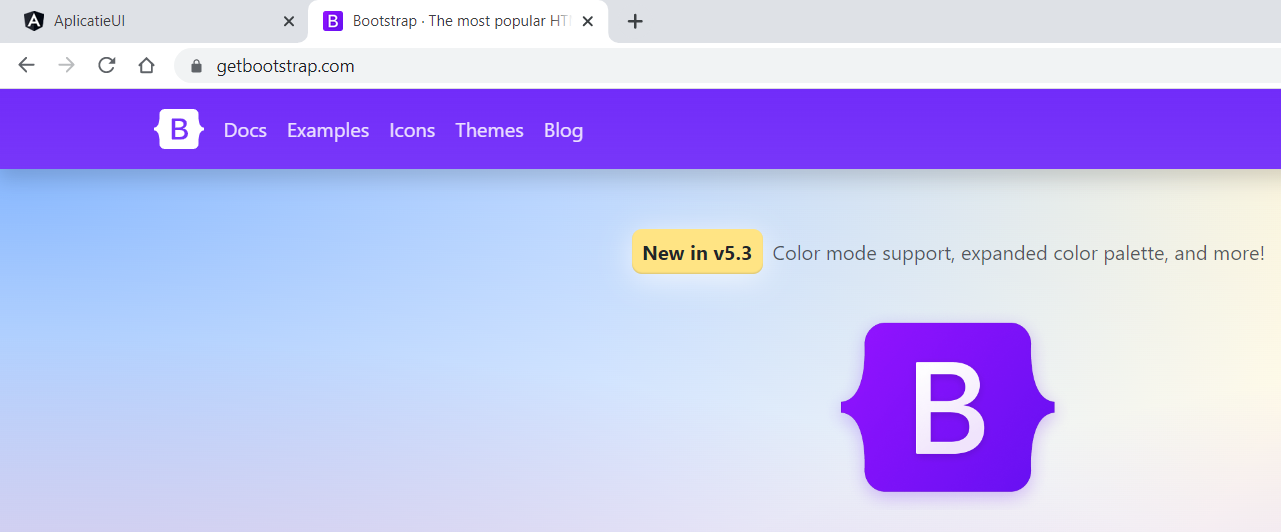
In cele ce urmeaza vom curata aplicatia noastra deoarece aceasta contine doar cod html static. Pentru curatare o sa mergem in proiect la fisierul app.component.html si vom sterge intre continutul in afara de tagul router-outlet, necesar pentru aplciarea routarilor.

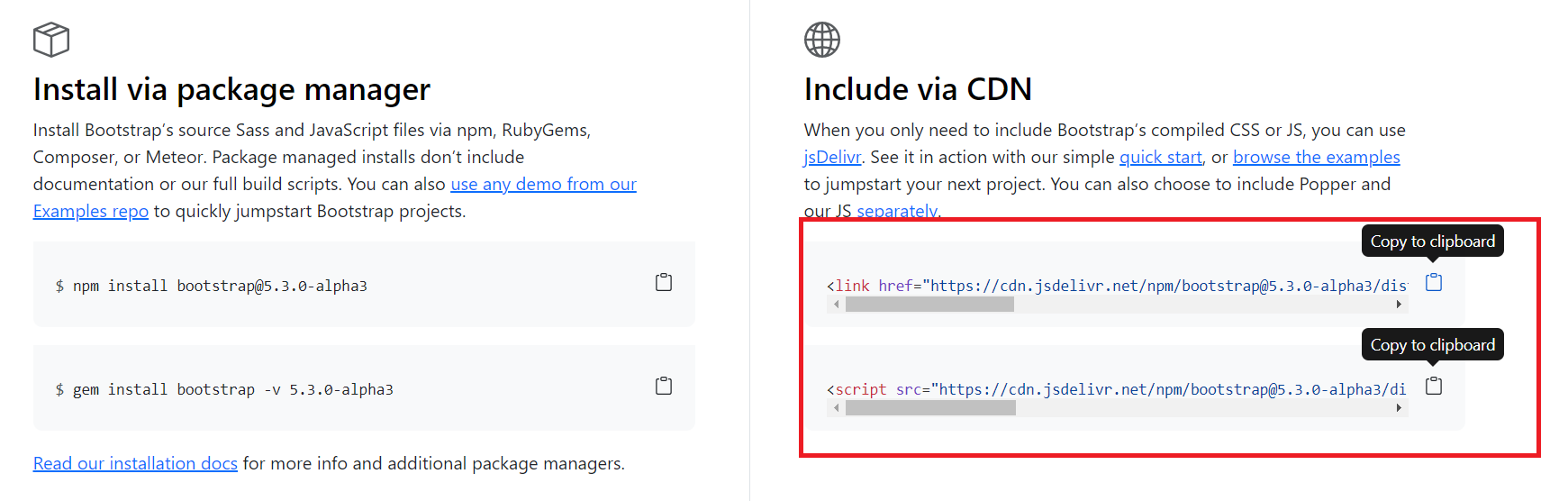


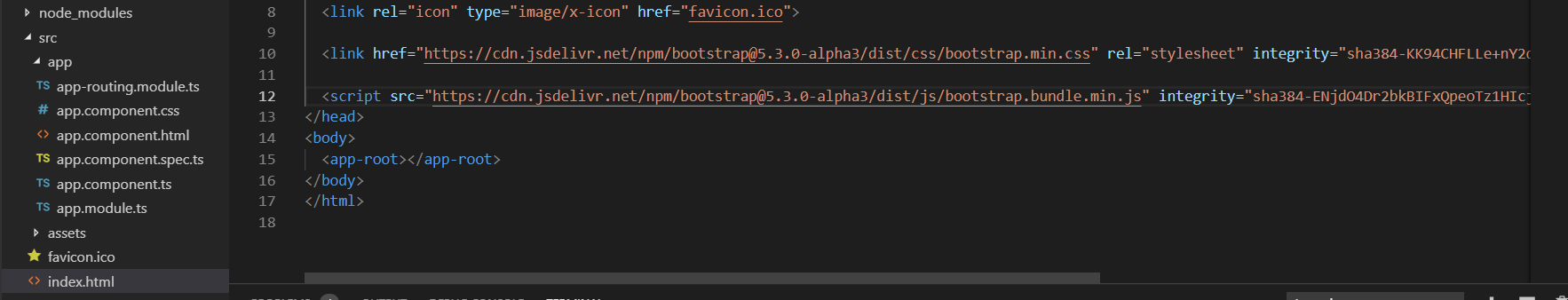
Rezultatul este urmatorul:



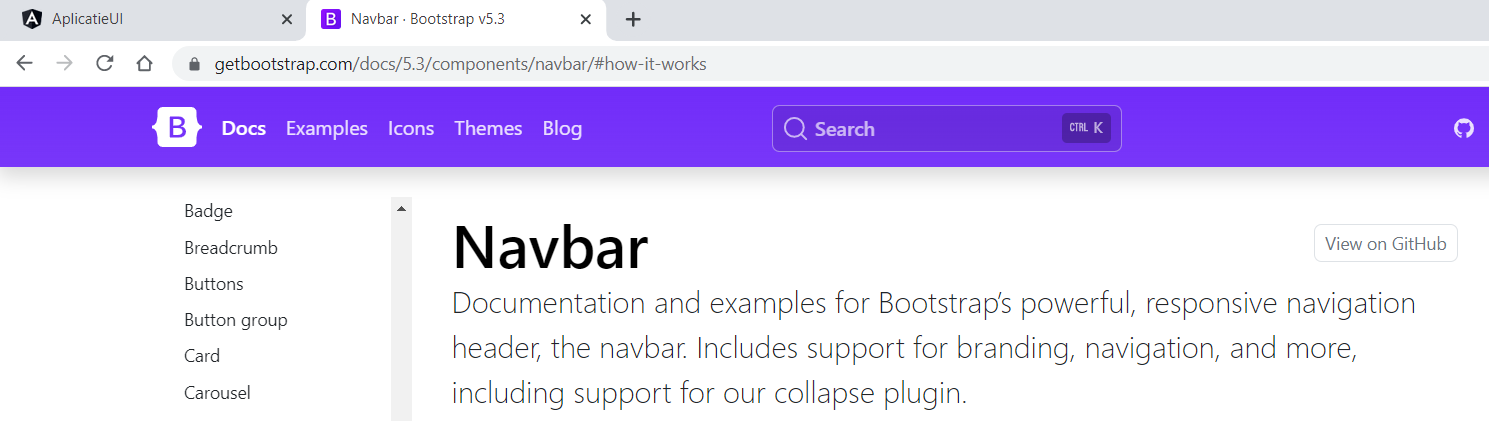
In cele ce urmeaza vom instala in proiect Boostrap pentru a putea utiliza componentele si stilurile acestuia in proiect. Pentru a instala boostrap in aplicatie vom naviga la site-ul boostrap de mai jos:

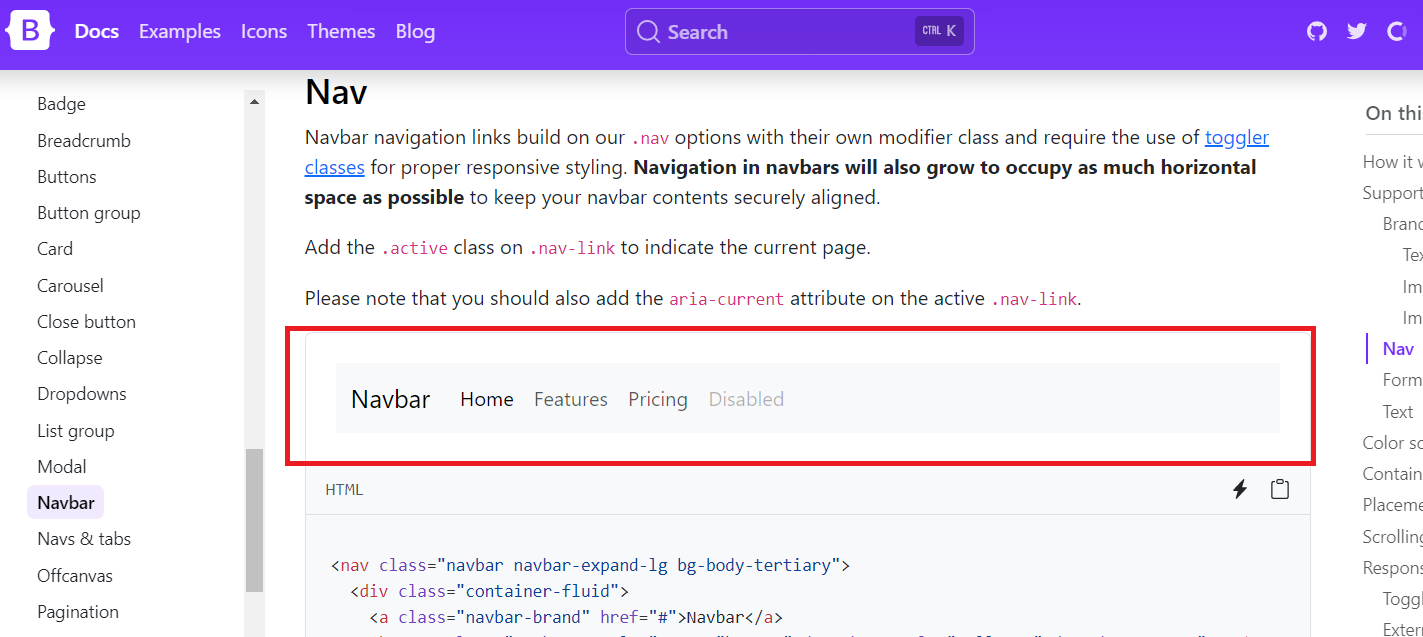


Vom copia mai apoi cele doua taguri in fisierul index.html din proiectul nostru.



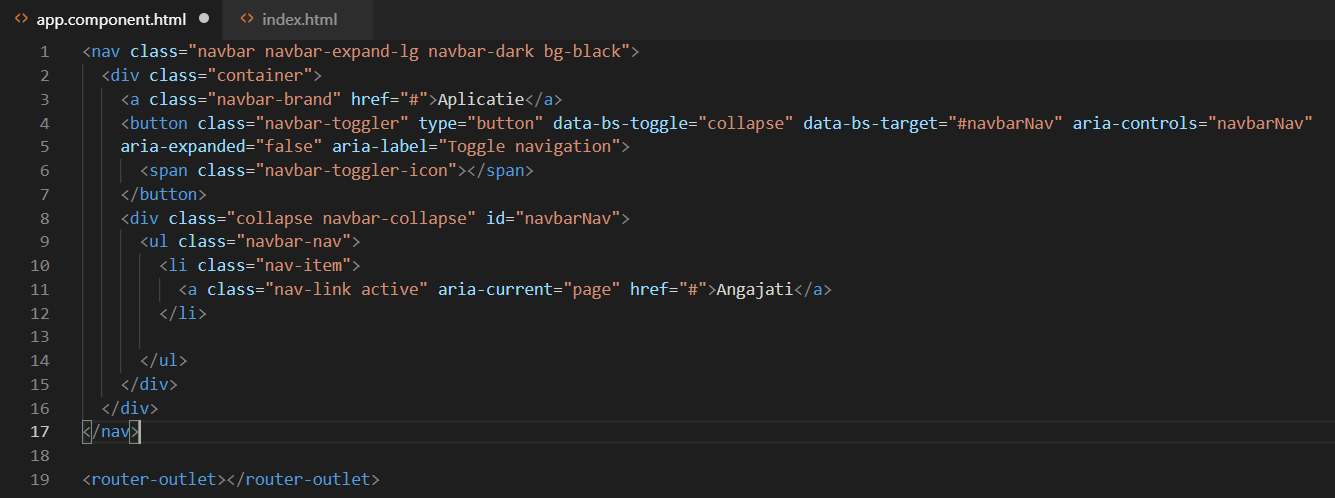
In cele ce urmeaza dorim sa cream o bara de navigare pentru proeictul nostru. Vom accesa aceasta componenta din site-ul boostrap preluand cea mai simpla componenta de navigare de pe site.



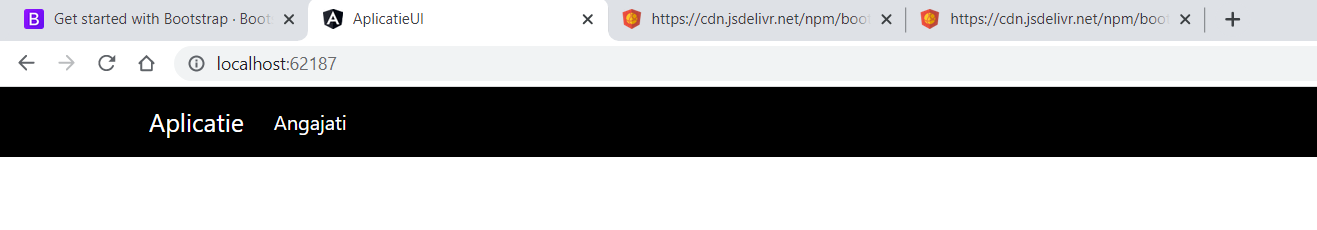


Pentru a include bara de navigare de mai sus in proiectul nostru este necesar sa copiem codul HTML in fisierul app.component.html, deasupra tagului de routare.

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="#">Products</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@5.3.3/dist/css/bootstrap.min.css"    rel="stylesheet"    integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"    integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"    crossorigin="anonymous"></script>    </head>  <body>    <app-root></app-root>  </body>  </html> |

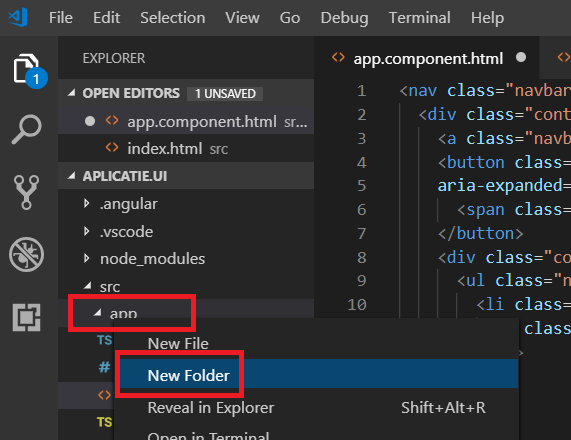
Asadar pana in acest moment am realizat un proiect Angular in care am adaugat scripturile de Boostrap si avem o pagina de produse care este goala.

A screenshot of a computer

AI-generated content may be incorrect.

Ceea ce ne propunem in continuare este sa adaugam un tabel pentru produse in pagina de Produse. 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 Products, unde Products este un subfolder pentru folderul Components.

 A screenshot of a computer

AI-generated content may be incorrect.

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

A screen shot of a computer

AI-generated content may be incorrect.

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

A screen shot of a computer screen

AI-generated content may be incorrect.

Rezultatul executiei comenzii de mai sus este urmatorul:

A screen shot of a computer screen

AI-generated content may be incorrect.

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

A screenshot of a computer

AI-generated content may be incorrect.

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

A screenshot of a computer

AI-generated content may be incorrect.

Rezultatul este urmatorul:

A screenshot of a computer

AI-generated content may be incorrect.

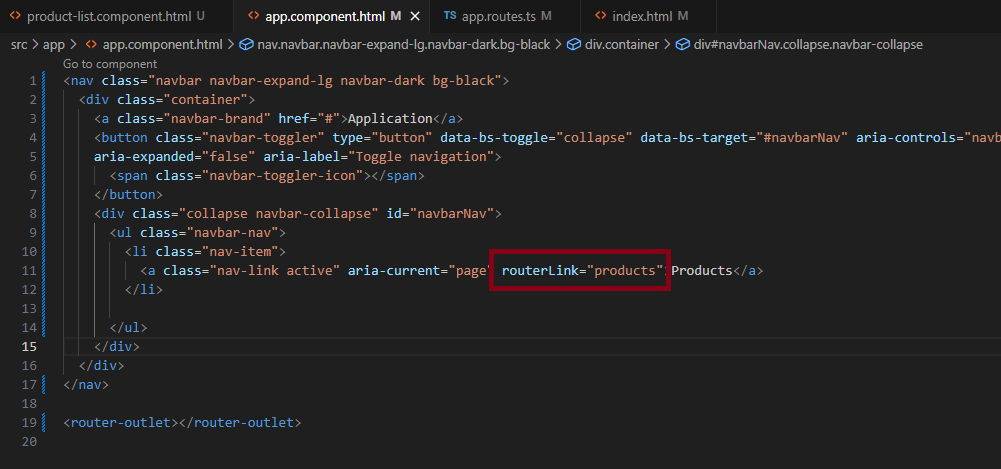
Practic se va afisa in pagina implicita tot ceea ce este in fisierul html al componentei.

A screenshot of a computer

AI-generated content may be incorrect.

Functionalitatea corecta este ca in momentul in care efectuez click pe Products 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 Products sa modificam valoare href si sa o inlocuim cu routeLink=”products”, la fel ca in capturile de mai jos.

A screen shot of a computer

AI-generated content may be incorrect.

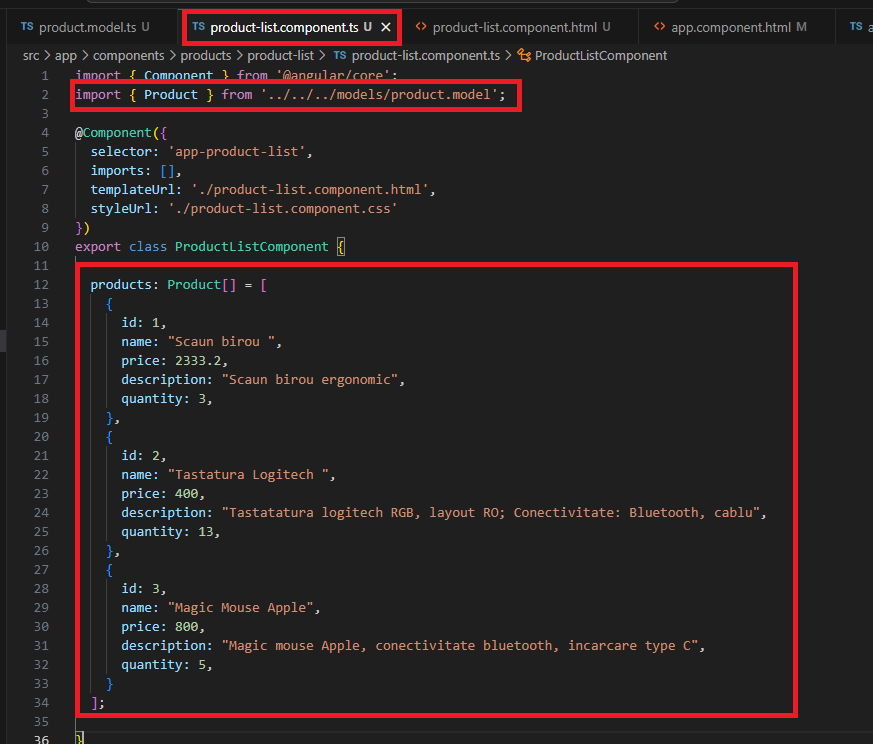
Vom modifica lista de produse astfel incat sa avem afisata o lista de produse sub forma tabelara. Pentru aceata vom efectua urmatorii pasi.

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

A screenshot of a computer

AI-generated content may be incorrect.

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



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

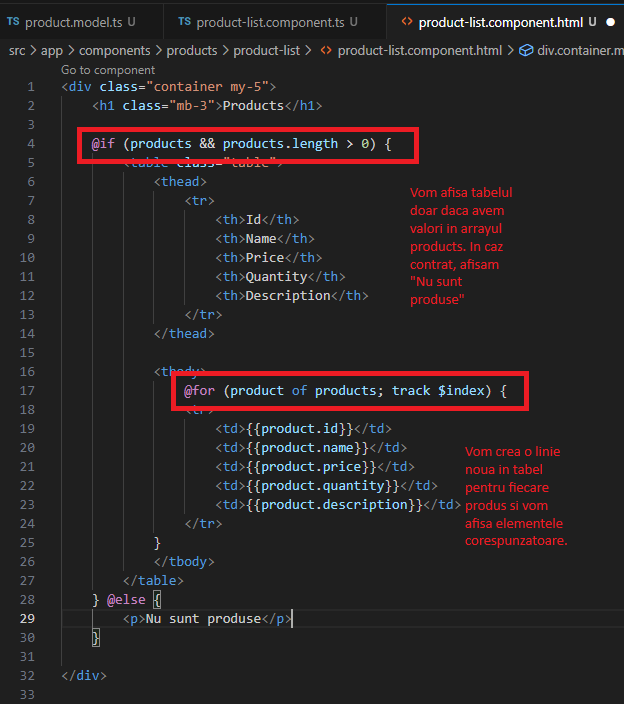
|  |
| --- |
| import { Component } from '@angular/core';  import { Product } from '../../../models/product.model';  @Component({    selector: 'app-product-list',    imports: [],    templateUrl: './product-list.component.html',    styleUrl: './product-list.component.css'  })  export class ProductListComponent {    products: Product[] = [      {        id: 1,        name: "Scaun birou ",        price: 2333.2,        description: "Scaun birou ergonomic",        quantity: 3,      },      {        id: 2,        name: "Tastatura Logitech ",        price: 400,        description: "Tastatatura logitech RGB, layout RO; Conectivitate: Bluetooth, cablu",        quantity: 13,      },      {        id: 3,        name: "Magic Mouse Apple",        price: 800,        description: "Magic mouse Apple, conectivitate bluetooth, incarcare type C",        quantity: 5,      }    ];  } |

Codul pentru modelul de **Product** este urmatorul:

|  |
| --- |
| export interface Product {      id: number;      name: string;      price: number;      description: string;      quantity: number;  } |

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

|  |
| --- |
| <div class="container my-5">      <h1 class="mb-3">Products</h1>      @if (products && products.length > 0) {          <table class="table">              <thead>                  <tr>                      <th>Id</th>                      <th>Name</th>                      <th>Price</th>                      <th>Quantity</th>                      <th>Description</th>                  </tr>              </thead>                <tbody>                  @for (product of products; track $index) {                  <tr>                      <td>{{product.id}}</td>                      <td>{{product.name}}</td>                      <td>{{product.price}}</td>                      <td>{{product.quantity}}</td>                      <td>{{product.description}}</td>                  </tr>              }              </tbody>          </table>      } @else {          <p>Nu sunt produse</p>      }    </div> |



Rezultatul final este urmatorul:

A screenshot of a computer

AI-generated content may be incorrect.

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

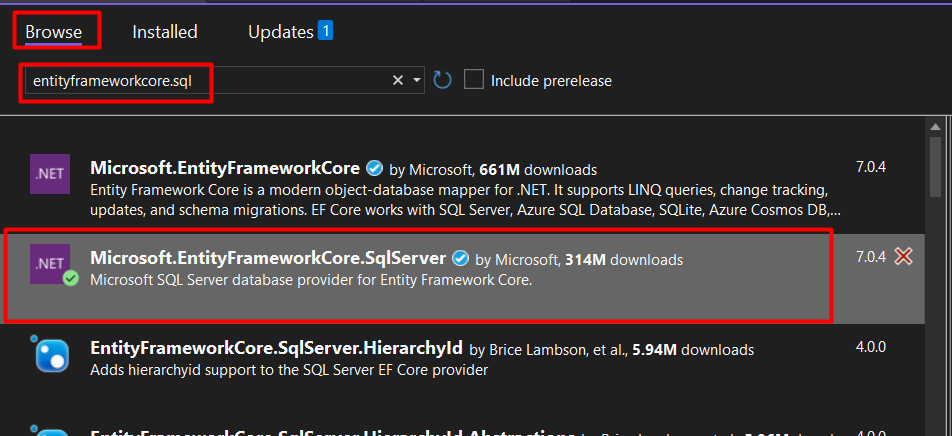
AI-generated content may be incorrect.

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

AI-generated content may be incorrect.

Clasa nou creata va contine cele 5 proprietati ale clasei din Angular si anume Id, Name, Price, quantity, Description.

Clasa va avea urmatorul cod:

|  |
| --- |
| public class Product  {  public Guid Id { get; set; }  public required string Name { get; set; }  public long Price { get; set; }  public int Quantity { get; set; }  public required string Description { get; set; }  } |

A screen shot of a computer

AI-generated content may be incorrect.

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

A screen shot of a computer program

AI-generated content may be incorrect.

In folderul Controllers vom adauga un nou controller denumit ProductController cu optiunea de EmptyController. Forma initiala a controller-ului este ca cea de mai sus. 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 toate produsele.
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 in GetAllProducts. Controller-ul o sa aiba forma din imaginea de mai jos. Pentru moment vom reuturna niste valori hardcodate. Mai jos, de asemenea, este prezentat tot codul pentru Controllerul creat.

|  |
| --- |
| using Aplicatie.API.Data;  using Microsoft.AspNetCore.Mvc;  using AplicatieAPI.Models;  using Microsoft.AspNetCore.Mvc;  namespace AplicatieAPI.Controllers  {  [ApiController]  [Route("api/[controller]")]  public class ProductController : Controller  {    [HttpGet]  public async Task<IActionResult> GetAllProducts()  {  var products = new List<Product>  {  new Product { Id = Guid.NewGuid(), Name = "Laptop", Price = 120000, Quantity = 5, Description = "High-performance laptop" },  new Product { Id = Guid.NewGuid(), Name = "Smartphone", Price = 60000, Quantity = 10, Description = "Latest model smartphone" },  new Product { Id = Guid.NewGuid(), Name = "Headphones", Price = 15000, Quantity = 15, Description = "Noise-canceling headphones" }  };  return Ok(products);  }  }  } |

Daca rulam aplicatia selectand HTTP in loc de HTTPS, o sa observam ca se va deschide o aplicatie Swagger care prezinta care sunt API-urile disponibile spre apelare.



In imaginea de mai sus am executat un astfel de call si am primit ca raspuns Code 200, adica success, si vom vedea lista de produse pe care am definit-o in controller.

Pana in acest moment am realizat metodele pentru operatia de READ. Haideti sa ne intoarcem inapoi in aplicatia Angular si sa modificam lista statica sa aduca inregistrarile definite in controller.

Ne aflam in componenta product-list unde in acest moment avem o lista statica de produse. A screenshot of a computer program

AI-generated content may be incorrect.

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 product**  fiind localizati in folderul services in momentul rularii acestei comenzi.

A screen shot of a computer screen

AI-generated content may be incorrect.

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 products.

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 **ProductService** pentru a prelua toate produsele.

|  |
| --- |
| import { HttpClient } from '@angular/common/http';  import { Injectable } from '@angular/core';  import { Product } from '../models/product.model';  @Injectable({    providedIn: 'root'  })  export class ProductService {    baseApiUrl: string ="https://localhost:5069/"    constructor(private httpService: HttpClient) { }    getAllProducts() {      return this.httpService.get<Product[]>(this.baseApiUrl + '/api/product');    }  } |

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

|  |
| --- |
| import { Component } from '@angular/core';  import { Product } from '../../../models/product.model';  @Component({    selector: 'app-product-list',    imports: [],    templateUrl: './product-list.component.html',    styleUrl: './product-list.component.css'  })  export class ProductListComponent {    products: Product[] = [];    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 produse disponibile spre afisare.

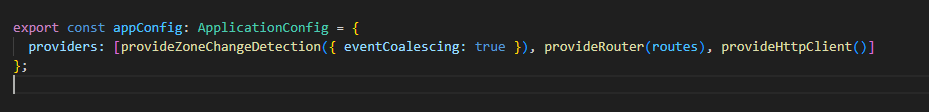
A screen shot of a computer

AI-generated content may be incorrect.

A computer screen shot of a program code

AI-generated content may be incorrect.

In fisierul **app.config.ts** va trebuit sa adaugam provideHttpClient() ca in imaginea de mai jos



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

In cazul in care primim o eroare legata de certificatul SSL, putem porni aplicatia de Backend cu HTTP in loc de HTTPS si requestul il vom face din Angular prin protocolul HTTP.

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

A computer screen with text

AI-generated content may be incorrect.

Rezultatul obtinut este urmatorul:

A screenshot of a computer

AI-generated content may be incorrect.

In momentul de fata avem un API functional care ne returneaza datele de la backend. In continuare vom crea o baza de date in AWS DynamoDB (NoSQL database) pe care vom aplica operatii CRUD.

## Bazele de date NoSQL

**NoSQL** (Not Only SQL) este o categorie de baze de date concepute pentru a gestiona volume mari de date nestructurate sau semi-structurate, oferind scalabilitate și performanță ridicată. Acestea sunt utilizate în special în aplicații moderne, cum ar fi rețele sociale, big data și IoT.

Avantaje:

* Permite stocarea flexibila a datelor (Schema se poate modifica usor)
* Performanta ridicata (se pot procesa volume mari de date)
* Disponibilitate mare (poate porocesa date concurent)
* Scalabilitate

Dezavantaje:

* Dificil de garantat consistenta datelor
* Lipsa relatiilor din SQL
* Lipsa unor standarde generale
* Greu de aplicat principiile ACID

### Tipuri de baze de date NoSQL:

**Documente** – Stochează date în format JSON/BSON (ex. **MongoDB**).  
**Coloane** – Optimizează datele în coloane (ex. **Cassandra**).  
**Cheie-Valoare** – Date simple accesate printr-o cheie (ex. **Redis, DynamoDB**).  
**Grafuri** – Gestionare de relații complexe între date (ex. **Neo4j**).

In cele ce urmeaza, vom folosi AWS DynamoDB, care este o baza de date de tip cheie-valoare. Pentru inceput va trebuie sa ne cream un cont gratuit pe platforma AWS Console.

A screenshot of a sign up form

AI-generated content may be incorrect.

Dupa ce ne facem contul, consola ar trebui sa arate dupa cum urmeaza: A screenshot of a computer

AI-generated content may be incorrect.

In acest pas trebuie sa avem grija ca suntem pe regiunea buna. In exemplul de astazi vom folosi Frankfurt.

A screenshot of a computer

AI-generated content may be incorrect.

Vom cauta in search DynamoDB si vom da click pe Create table.

Vom crea o tabela noua cu urmatoarele specificatii:

A screenshot of a computer

AI-generated content may be incorrect.

Dupa ce dam click pe create table, vom fi redirectionati catre overview ul tabelelor, care va arata ca in imaginea de mai jos. Important este sa avem status Active:

A screenshot of a computer

AI-generated content may be incorrect.

Vom da apoi click pe numele tabelului (products) si vom crea o noua intrare in acesta dand click pe actions si apoi pe create item. Vom folosi apoi butonul de add attribute pentru a adauga toate campurile pe care le avem in modelul nostru.

A screenshot of a computer

AI-generated content may be incorrect.Daca vom merge acum pe explore items, vom vedea noua intrare in tabel pe care tocmai am adaugat-o:

A screenshot of a computer

AI-generated content may be incorrect.

Acum va trebui sa conectam aplicatia noastra de backend la AWS pentru a putea citi si scrie in aceasta baza de date.

## Conexiunea cu baza de date

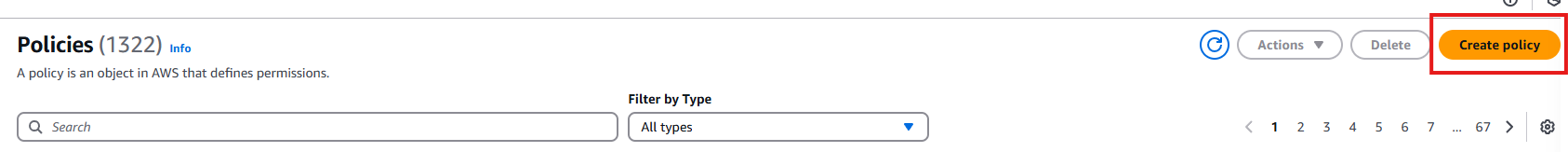
Pentru a ne putea conecta la baza de date din cloud, vom avea nevoie sa cream un nou user si sa ii dam permisiunile necesare.

Din consola vom cauta IAM (Identity and Access Manager) si ar trebui sa ajungem pe pagina de mai jos:

A screenshot of a computer

AI-generated content may be incorrect.

Prima data vom crea accesul nevesar navigand pe **policies** si vom da click pe Create policy in coltul din dreapta sus.



Vom cauta dynamoDb in services.

A screenshot of a computer

AI-generated content may be incorrect.

In urmatorul ecran vom bifa All pentru actions si resources ca in maginea de mai jos. Astfel vom da access la toate actiunile pentru DynamoDB.

A screenshot of a computer

AI-generated content may be incorrect.

Vom da apoi next si in urmatorul ecran vom seta numele si vom da click pe **Create policy**.

A screenshot of a computer

AI-generated content may be incorrect.

Acum va trebui sa cream un nou user si ii asignam permisiunile pe care tocmai le-am setat. Pentru asta vom naviga la pagina de users din meniul din stanga si vom da click pe **Create user.**

A screenshot of a computer

AI-generated content may be incorrect.

In urmatorul ecran ii vom atribui un nume si vom da next.

A screenshot of a computer

AI-generated content may be incorrect.

In urmatorul ecran vom atasa permisiunile create dand click pe Attach policies directly si vom cauta dupa numele pe care l-am atribuit anterior, in cazul meu, dynamo-lab. Voi selecta aceste permisiuni si voi da next.

A screenshot of a computer

AI-generated content may be incorrect.

Urmatorul ecran este doar de verificare si ar trebui sa arate in felul urmator. Vom da click pe create user.

A screenshot of a computer

AI-generated content may be incorrect.

Urmatorul pas este sa configuram userul create astfel incat sa il putem folosi in aplicatia noastra. Astfel vom naviga pe Users -> Security credentials si vom crea un nou Access Key.

A screenshot of a computer

AI-generated content may be incorrect.

Vom selecta in urmatorul ecran **Other** si vom da next. Apoi vom seta o descriere sugestiva si vom crea cheia de acces.

A screenshot of a computer screen

AI-generated content may be incorrect.

In urmatorul pas este foarte important sa descarcam fisierul generat!

A screenshot of a computer

AI-generated content may be incorrect.

Ne vom intoarce in Visual Studio, vom naviga pe NutGetManager si vom instala urmatoarele pachete:

A screenshot of a computer

AI-generated content may be incorrect.A computer screen shot of a computer code

AI-generated content may be incorrect.

Dupa acest pas, daca vom merge in fisierul AplicatieUI.cs ar trebui sa arate ca in imaginea de mai jos:

A screen shot of a computer program

AI-generated content may be incorrect.

Odata ce avem aceste pachete instalate, vom naviga in fisierul Program.cs unde vom adauga urmatorul cod:

A screen shot of a computer program

AI-generated content may be incorrect.

Pentru a continua setarile pentru AWS, vom naviga in fisierul appsettings.json si vom adauga urmatoarele randuri. Region va fi eu-central-1 deoarece aceasta este regiunea corespunzatoare pentru Frankfurt.

A screen shot of a computer

AI-generated content may be incorrect.



{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"AWS": {

"Profile": "default",

"Region": "eu-central-1"

}

}

Acum ne vom intoarce in fisierul ProductController.cs si va trebuit sa adaugam conexiunea catre AWS. Pentru aceasta vom folosi AccessKey si SecretKey pe care le-am salvat mai devreme.

Prima data va trebuit sa definim conexiunea catre baza de date folosind AmazonDynaoDBClient ca in exemplul de mai jos:

private static readonly AmazonDynamoDBClient client = new AmazonDynamoDBClient("your-access-key", "your-secret-key", Amazon.RegionEndpoint.EUCentral1);

private static readonly DynamoDBContext context = new DynamoDBContext(client);

Aici vom inlocui cheile cu valorile descarcate anterior. Apoi vom adauga o noua metoda de tip get care ne va returna datele din baza de date create in AWS, pe care o vom numi GetAllProductsFromAWS().

Pentru a citi datele din tabela noastra, vom folosi metoda ScanAsync si vom folosi filtrul „default“ pentru a returna toate datele din tabela.

A screenshot of a computer program

AI-generated content may be incorrect.

In continuare vom crea un folder nou numit entities la acelasi nivel cu Controllers si Models, unde vom adauga o noua clasa denumita ProductEntity:

A screenshot of a computer

AI-generated content may be incorrect.

Clasa noastra va avea urmatoare structura:

A screen shot of a computer program

AI-generated content may be incorrect.

using Amazon.DynamoDBv2.DataModel;

namespace AplicatieAPI.Entities

{

[DynamoDBTable("products")]

public class ProductEntity

{

[DynamoDBHashKey("id")]

public string? Id { get; set; }

[DynamoDBProperty("name")]

public required string Name { get; set; }

[DynamoDBProperty("price")]

public long Price { get; set; }

[DynamoDBProperty("quantity")]

public int Quantity { get; set; }

[DynamoDBProperty("description")]

public required string Description { get; set; }

}}

Acum vom porni aplicatia de backend si vom verifica daca aplicatia ne returneaza datele corecte. Daca testam folosim swagger, ar trebui sa primim urmatorul rezultat:

A screenshot of a computer

AI-generated content may be incorrect.

Pentru a vedea datele din tabela noastra pe frontend, va trebui sa modificam aplicatia pentru a ne apela noua metoda get. Va trebui sa ne intoarcem in proiectul **Aplicatie.UI** mergem in fisierul **product.service.ts**  unde vom adauga **/aws** pe ruta noastra.

A screen shot of a computer

AI-generated content may be incorrect.

Dupa ce salvam si rulam aplicatia prin comanda **ng serve** va trebui sa vedem intrarea din baza noastra de date in tabel:

A screenshot of a computer

AI-generated content may be incorrect.

Acum vom continua cu celelalte operatiuni, anume: CREATE, UPDATE si DELETE.

## Create

Pentru functia de create vom crea o componenta nou ace va continue un formular prin intermediul caruia vom trimite datele la aplicatia noastra de Backend si mai apoi la AWS pentru a le salva in tabela.

Pentru acesta efectuam urmatorii pasi:

* Deschidem o noua fereastra terminal
* Nagivam in sectiunea /**products**

A black screen with white text

AI-generated content may be incorrect.

* Rulam comanda **ng g c add-product**

A screenshot of a computer screen

AI-generated content may be incorrect.

* Adaugam o noua ruta pentru aceasta componenta, astfel ca in **app-routes.ts** vom adauga o noua ruta care apelata va afisa componenta create anterior.

A screenshot of a computer program

AI-generated content may be incorrect.

//////////////////////////////////////////////////////////////////////////////////

Pas5: Adaugam o noua intrare in meniul de navigare a aplicatiei din **app.component.html**, codul este cel de mai jos:

|  |
| --- |
| <nav class="navbar navbar-expand-lg navbar-dark bg-black">    <div class="container">      <a class="navbar-brand" href="#">Application</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="products" >Products</a>          </li>          <li class="nav-item">            <a class="nav-link active" aria-current="page" routerLink="products/add">Add product</a>          </li>          </ul>      </div>    </div>  </nav>  <router-outlet></router-outlet> |

Pas6: Vom naviga in fisierul **app.component.ts** si vom adauga RouterModule in lista de imports, deoarece fara acest modul nu va functiona routerLink. Va trebui sa adaugam acest import oriunde vom folosi routerLink.

Rezultatul asteptat este urmatorul:

A black and white rectangular object with white text

AI-generated content may be incorrect.

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 produse in tabela bazei de date. Pentru aceasta vom accesta vom deschide in Visual Studio Code componenta add-product.component.html si vom adauga urmatorul cod:

|  |
| --- |
| <div class="container my-5">      <h1 class="mb-3">Adaugare Produs Nou</h1>      <div class="row">          <div class="col-6">              <form>                  <div class="mb-3">                    <label for="name">Name</label>                    <input type="text" class="form-control" id="name">                  </div>                    <div class="mb-3">                      <label for="price">Price</label>                      <input type="email" class="form-control" id="price">                  </div>                  <div class="mb-3">                      <label for="quantity">Quantity</label>                      <input type="number" class="form-control" id="quantity">                  </div>                  <div class="mb-3">                      <label for="description">Description</label>                      <input type="number" class="form-control" id="description">                  </div>                  <button type="submit" class="btn btn-primary">Save</button>                </form>          </div>      </div>    </div> |

Rezultatul este cel de mai jos:

A screenshot of a computer

AI-generated content may be incorrect.

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 naviga in fisierul **add-product.component.ts** si vom importa modulul **ReactiveFormsModule**

A screen shot of a computer screen

AI-generated content may be incorrect.

1. Vom crea apoi scheletul formularului nostru ca in imaginea de mai jos:

A screen shot of a computer code

AI-generated content may be incorrect.

Dupa cum se poate observa, trebuie sa importam FormGroup, FormControl si Validators. Dupa cum se poate observa am adaugat cateva validatoare **required** si cateva pentru dimensiune sau suma. Aceste validatoare nu ne vor permite sa cream un produs cat timp nu sunt indeplinite aceste cerinte.

1. Urmatorul pas este sa atribuim fiecarui input legatura catre formularul nostru. Vom folosi tagul de **formControlName** ca in exemplul de mai jos:

A screen shot of a computer program

AI-generated content may be incorrect.

1. Ne ramane sa definim ce sa se intample cand apasam pe butonul save. Astfel vom adauga pe tagul de form o actiune pe (ngSubmit) ca in exemplu:



1. Ne vom intoarce in fisierul **add-product.component.ts** si vom define metoda noastra:
2. createProduct() {
3. if (this.productForm.valid) {
4. console.log('Form Data:', this.productForm.value);
5. } else {
6. console.log('Form is invalid');
7. }
8. }

6. Vom merge in browser si vom testa aplicatia noastra. Vom apasa F12 pentru a deschide dev-tools si vom naviga pe console. Daca completam toate fieldurile (LUATI IN CALCUL VALIDARILE) si dam click pe save, ar trebui sa vedem in consola datele introduse.

A screen shot of a computer

AI-generated content may be incorrect.

7. In acest moment daca vom introduce date invalide, de exemplu daca lasam liber un field, in consola se va printa “Form is invalid”.

Pentru simplicitate aici aveti codul fisierului add-product.component.ts

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

import {ReactiveFormsModule, FormGroup, FormControl, Validators} from '@angular/forms';

@Component({

  selector: 'app-add-product',

  imports: [ReactiveFormsModule],

  templateUrl: './add-product.component.html',

  styleUrl: './add-product.component.css'

})

export class AddProductComponent {

  productForm = new FormGroup({

    name: new FormControl('', [Validators.required, Validators.minLength(3)]),

    price: new FormControl('', [Validators.required]),

    quantity: new FormControl('', [Validators.required, Validators.min(1)]),

    description: new FormControl('', [Validators.required, Validators.minLength(10)]),

  });

  createProduct() {

    if (this.productForm.valid) {

      console.log('Form Data:', this.productForm.value);

    } else {

      console.log('Form is invalid');

    }

  }

}

In cele ce urmeaza vom trimite datele mai departe la backend, dar prima data va trebui sa ne definim endpointul pe care sa il apelam. Va trebui sa mergem in **Visual Studio in Aplicatie.API.**

Vom naviga apoi in fisierul **ProductController.cs** si vom adauga urmatoarea metoda

[HttpPost("create")]

public async Task<IActionResult> CreateProduct([FromBody] ProductEntity productRequest)

{

productRequest.Id = Guid.NewGuid().ToString();

await context.SaveAsync(productRequest);

return Ok(productRequest);

}

Dupa cum se poate observa, setam noi id-ul de fiecare data cand cream un produs nou.

Acum ne vom intoarce in aplicatia noastra angular si vom apela metoda pe care tocmai am creat-o.

1. Vom naviga in fisierul produc.service.ts
2. Vom adauga o noua metoda createProduct()

A screen shot of a computer code

AI-generated content may be incorrect.

1. Va fi nevoie sa folosim metoda .subscribe() pentru a complete apelarea metodei.

A screen shot of a computer program

AI-generated content may be incorrect.

1. Daca deschidem dev-tools putem observa ce se afiseaza si vom vedea id-ul create si returnat de backend.



## Update

Pentru a adauga functionalitatea de UPDATE va trebui sa ne cream metoda HTTP in controllerul nostru in aplicatia backend si apoi va trebui sa modificam aplicatia de frontend pentru a adauga un formular asemanator celui de creare produs, prin care sa putem edita un produs si salva modificarile. Prima data vor incepe cu partea de backend.

Vom naviga in **ProductController**.cs si vom adauga urmatoarele metode(una pentru update si una pentru a returna datele obiectului pe care vrem sa il editam):

[HttpPut("{id}")]

public async Task<IActionResult> UpdateProduct(string id, [FromBody] ProductEntity productRequest)

{

var product = await context.LoadAsync<ProductEntity>(id);

if (product == null) return NotFound();

product.Name = productRequest.Name;

product.Quantity = productRequest.Quantity;

product.Price = productRequest.Price;

product.Description = productRequest.Description;

await context.SaveAsync(product);

return Ok(productRequest);

}

[HttpGet("{id}")]

public async Task<IActionResult> GetProductById(string id)

{

var product = await context.LoadAsync<ProductEntity>(id);

if (product == null) return NotFound();

return Ok(product);

}

Prin prima metoda, vom da get la produsul nostru dupa id, deoarece vrem sa verificam ca exista inainte sa ii face update. In cazul in care nu exista vom returna “NotFound()” in caz contrar vom face update si vom returna obiectul cu noile proprietati suprascrise.

Acum ne vom intoarce in aplicatia de Angular si va trebui sa ne cream noua componenta de editare produs. Vom face asta urmarind urmatorii pasi:

1. Vom deschide o noua fereastra de terminal si vom naviga in folderul **products** folosind comanda

**Cd src/app/components/products**

1. Vom rula comanda **ng g c update-product**

A screen shot of a computer screen

AI-generated content may be incorrect.

1. Vom adauga noua ruta in fisierul **app.routes.ts** pentru update ca in imaginea de mai jos:

A computer screen shot of text

AI-generated content may be incorrect.

1. Vom adauga acum o iconita de edit pe care cand o vom apasa ne vom ruta userul catre pagina de edit. Astfel vom naviga in fisierul **product-list.component.html** si vom adauga urmatorul buton:

A screen shot of a computer program

AI-generated content may be incorrect.

1. Pentru a putea folosi routerLink, va trebui sa importam **RouterModule** in **product-list.component.ts.**

A screen shot of a computer program

AI-generated content may be incorrect.

1. Acum va trebui sa cream formularul efectiv in componenta pe care tocmai am creat-o. Vom naviga in **update-product.component.html** si vom copia tot codul din fisierul **add-product.component.html.**

Aveti mai jos tot codul din acest fisier:

<div class="container my-5">

    <h1 class="mb-3">Modificare Produs cu id-ul: {{product.id}}</h1>

    <div class="row">

        <div class="col-6">

            <form [formGroup]="productForm" (ngSubmit)="updateProduct(product.id)">

                <div class="mb-3">

                  <label for="name">Name</label>

                  <input type="text" class="form-control" id="name" formControlName="name">

                </div>

                <div class="mb-3">

                    <label for="price">Price</label>

                    <input type="number" class="form-control" id="price" formControlName="price">

                </div>

                <div class="mb-3">

                    <label for="quantity">Quantity</label>

                    <input type="number" class="form-control" id="quantity" formControlName="quantity">

                </div>

                <div class="mb-3">

                    <label for="description">Description</label>

                    <input type="text" class="form-control" id="description" formControlName="description">

                </div>

                <button type="submit" class="btn btn-primary">Save</button>

              </form>

        </div>

    </div>

</div>

1. Vom naviga acum in fisierul **update-product.component.ts** unde va trebui sa adaugam mai multe functionalitati:
   1. Va trebui sa ne cream formularul nostru (vom importa Reactive Forms module) si vom declara forma formularului si vom pastra aceleasi validari.

A screen shot of a computer code

AI-generated content may be incorrect.

* 1. Vom adauga si structura produsului ca in imaginea de mai sus.
  2. Va trebui sa citim din ruta noastra id ul produsului si sa facem un get request pentru a putea popula inputurile cu datele existente.

A screen shot of a computer program

AI-generated content may be incorrect.

Momentan inca nu avem definita functia getProductById din service, deci este normal sa fie subliniat cu rosu.

In constructor luam id-ul din ruta noastra si apoi apelam metoda de get si popuam obiectul nostru **product** cu rezultatul returnat de backend.

* 1. Apelam functia **resetForm** cu datele primite de la backend si populam fiecare camp cu datele respective.
  2. Adaugam metoda de **updateProduct** care va fi apelata cand apasam pe butonul save

A screen shot of a computer program

AI-generated content may be incorrect.

1. Vom naviga in fisierul **product.service.ts** unde vom adauga noile metode de **getProductById** si **updateProduct**

A screen shot of a computer program

AI-generated content may be incorrect.

Acum daca vom rula atat aplicatia de backend cat si cea de frontned (ng serve) ar trebui sa avem functionalitatea de edit.

A group of black and orange text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Aveti atasat codul pentru fisierul **update-product.component.ts**

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

import {ReactiveFormsModule, FormGroup, FormControl, Validators} from '@angular/forms';

import { ProductService } from '../../../services/product.service';

import { ActivatedRoute, Router } from '@angular/router';

import { Product } from '../../../models/product.model';

@Component({

  selector: 'app-update-product',

  imports: [ReactiveFormsModule],

  templateUrl: './update-product.component.html',

  styleUrl: './update-product.component.css'

})

export class UpdateProductComponent {

    productForm = new FormGroup({

      name: new FormControl('', [Validators.required, Validators.minLength(3)]),

      price: new FormControl('', [Validators.required]),

      quantity: new FormControl('', [Validators.required, Validators.min(1)]),

      description: new FormControl('', [Validators.required, Validators.minLength(10)]),

    });

    product: Product = {

        id: '',

        name: '',

        quantity: 0,

        price: 0,

        description: ''

    };

    constructor(

      private service: ProductService,

      private route:  ActivatedRoute

    ) {

      this.route.paramMap.subscribe({

        next: (params) => {

          const id = params.get('id');

          if (id) {

            this.service.getProductById(id).subscribe({

              next: (response) => {

                this.product = response;

                this.resetForm(response);

              }

            });

          }

        }

      });

    }

    resetForm(product: Product) {

      this.productForm.get('name')?.setValue(product.name);

      this.productForm.get('price')?.setValue(product.price.toString());

      this.productForm.get('quantity')?.setValue(product.quantity.toString());

      this.productForm.get('description')?.setValue(product.description);

    }

    updateProduct(productId: string | undefined) {

      if (productId && this.productForm.valid) {

        this.service.updateProduct(productId, {

          ...this.productForm.value

          } as any

        ).subscribe({

          next:(updatedProduct) => { console.log(`Product with id ${updatedProduct.id} was updated!`)},

          error: (error) => { console.log(error) }

        });

      } else {

        console.log('Form is invalid');

      }

    }

}

Iar aici codul fisierului **product.service.ts**

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

import { Injectable } from '@angular/core';

import { Product } from '../models/product.model';

@Injectable({

  providedIn: 'root'

})

export class ProductService {

  baseApiUrl: string ="http://localhost:5069";

  constructor(private httpService: HttpClient) { }

  public getAllProducts() {

    return this.httpService.get<Product[]>(this.baseApiUrl + '/api/Product/aws');

  }

  public getProductById(id: String) {

    return this.httpService.get<Product>(this.baseApiUrl + '/api/Product/' + id);

  }

  public createProduct(productData: Product) {

    return this.httpService.post<Product>(this.baseApiUrl + '/api/Product/create', productData);

  }

  public deleteProduct(id: string) {

    return this.httpService.delete(this.baseApiUrl + '/api/Product/delete/' + id);

  }

  public updateProduct(productId: string, productData: Product) {

    return this.httpService.put<Product>(this.baseApiUrl + '/api/Product/' + productId, productData);

  }

}

## Delete

Pentru a adauga si functionalitatea de DELETE, va trebui sa parcurgem aceeasi pasi. Prima data vom merge in Visual Studio in ApplicatieAPI unde va trebui sa adaugam in controllerul nostru o metoda de delete.

Vom naviga in **ProductController**.cs si vom adauga urmatoarea metoda:

[HttpDelete("delete/{id}")]

public async Task<IActionResult> DeleteProduct( string id)

{

var product = await context.LoadAsync<ProductEntity>(id);

if (product == null)

{

return NotFound();

}

await context.DeleteAsync(product);

return Ok(product);

}

Acum vom porni aplicatia backend si vom merge in Visual Studio Code pentru a implementa si pe aplicatia de angular functia de delete. Vom urma urmatorii pasi:

1. Va trebui sa adaugam in fisierul **product.service.ts** apelarea metodei noastre de **DELETE**.

A screen shot of a computer program

AI-generated content may be incorrect.

1. Vom naviga in **index.html** si vom adauga urmatoarea linie:

 <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">

1. Vom adauga o noua iconita in tabel care va actiona ca un buton de delete. Pentru asta vom naviga in fisierul **product-list.component.ts** si vom adauga o coloana noua ca in imagine:

A screen shot of a computer program

AI-generated content may be incorrect.

1. Vom adauga metoda de **onDelete()** in fisierul product-list.component.ts

  onDelete(productId?: string): void {

    if (productId !== undefined) {

      this.service.deleteProduct(productId).subscribe({

        next:(product) => { this.products = this.products.filter(p => p.id != productId);},

        error: (error) => { console.log(error) }

      });

    }

  }

Dupa cum puteti observa aici reasignam variabilei **this.products** noua lista. Acesta nu este un best practice, este doar pentru a vedea modificarile si in lista noastra in timp real. O abordare mai buna ar fi sa ne folosim in continuare de RxJs pentru a crea un subject care sa ne faca update la lista noastra de produse la fiecare delete.

1. Acum putem rula atat aplicatia de backend cat si cea de frontend pentru a testa ca functioneaza codul scris.