# Uputsvto za kreiranje VSCode .NET8 projekta

## Preduslovi:

-VSCode

<https://code.visualstudio.com/download>

-DotNet SDK

<https://dotnet.microsoft.com/en-us/download>

## Kreiranje kostura .net aplikacije:

Potrebno je otvoriti terminal (cmd/bash/powershell) i navigirati do lokacije gde ćemo kreirati naš projekat.

Komandom **mkdir** NoviProjekat kreiramo folder u koji cemo smestiti našu .NET aplikaciju.

Pre kreiranja aplikacije, možemo se uveriti da smo ispravno instalirali DotNet SDK komandom **dotnet –info**. Ukoliko vidimo ispis na terminalu, možemo da nastavimo dalje.

Na samom početku kreiramo .sln file, tj. solution file kako bi naša aplikacija znala koje fajlove da koristi pri build-u.

**dotnet new sln**

Ukoliko ne unesimo naziv, solution file ce uzeti ime foldera u kome se nalazi, u našem slučaju to će biti NoviProjekat.sln

Kreiraćemo ASP.NET Core Web API projekat, koji kasnije povezujemo sa Angularom. U terminalu koristimo komandu

**dotnet new webapi -n API –use-controllers**

Sledeći korak jeste dodavanje tog fodlera u nas solution fajl.  
**dotnet sln add API**

Pokretanje .net aplikacije se vrši komandom **dotnet run**

# Entity Framework: Automatizacija interakcije sa bazom podataka

## Uvod

Entity Framework je alat za mapiranje objekata na relacijske baze podataka, čiji je zadatak da prevede naš kod u SQL naredbe koje ažuriraju tabele u bazi podataka.

## Poteškoće Pre .NET 3.5

Pre .NET 3.5 verzije, često smo koristili ADO.NET za čuvanje ili dobijanje podataka iz baze podataka. Ovaj proces bio je naporan i sklon greškama jer smo morali pisati SQL kod kao stringove.

## Entity Framework: Automatizacija i Pregled

Microsoft je obezbedio Entity Framework kako bi automatizovao sve aktivnosti vezane za bazu podataka za našu aplikaciju. Entity Framework nam omogućava da pišemo linkovane upite (LINQ upite) za čitanje podataka iz baze, pruža praćenje promena, konkurentnost i automatsko upravljanje transakcijama.

## Glavna Klasa DBContext

Ključna klasa u Entity Framework-u je DBContext, koja služi kao most između naših domenskih ili entitetskih klasa i baze podataka.

## Ključne Funkcionalnosti Entity Framework-a

* Linkovani upiti (LINQ upiti) za rad sa podacima
* Praćenje promena
* Automatsko čuvanje podataka u bazi
* Konkurentnost radi zaštite od istovremenih izmena
* Automatsko upravljanje transakcijama
* Keširanje prvog nivoa radi ubrzanja ponovljenih upita
* Migracije za upravljanje šemom baze podataka

## Mogućnosti Entity Framework-a

Entity Framework nam omogućava da koristimo "code first" pristup za automatsko generisanje šeme baze podataka iz koda, ili "database first" gde se .NET klase generišu na osnovu postojeće baze podataka, i "model first" pristup gde se modeli podataka dizajniraju vizuelno preko alata poput Entity Framework Designer-a.

# SQLite Database

## Uvod

SQLite je popularna baza podataka poznata po svojoj jednostavnosti, performansama i širokoj podršci. Napisana je u programskom jeziku C i često se koristi kao ugrađena biblioteka u aplikacijama.

## Primena u Različitim Industrijama

SQLite se ističe po svojoj primenljivosti u nekoliko najpoznatijih web pregledača, operativnih sistema, mobilnih telefona i drugih ugrađenih sistema. Njegova popularnost proizlazi iz jednostavnosti korišćenja, minimalnih sistemskih zahteva i visokih performansi.

## Podrška u Programskim Jezicima

Mnogi programski jezici podržavaju SQLite biblioteku, a njena sintaksa je slična PostgreSQL-u. Važno je napomenuti da SQLite podrazumevano ne primenjuje proveru tipova, što može biti značajno za programere prilikom manipulacije podacima.

## Karakteristike Baze Podataka

Kao baza podataka, SQLite podržava transakcije i omogućava izolaciju transakcija radi zaštite integriteta podataka. Takođe podržava različite tipove podataka, uključujući tekstualne, numeričke, vremenske i binarne podatke.

## Portabilnost i Integracija

SQLite omogućava portabilnost baza podataka između različitih platformi i operativnih sistema, što olakšava razvoj aplikacija koje zahtevaju lokalno skladištenje podataka. Njegova jednostavnost i efikasnost čine ga popularnim izborom za razvoj aplikacija svih veličina.

# Kontroleri

Kontroleri su klase koje obavljaju logiku za obradu HTTP zahteva. Kontroleri u ASP.NET Core nasleđuju Controller klasu i obično sadrže metode koje se nazivaju akcijama. Akcije su odgovorne za obradu specifičnih vrsta zahteva, kao što su zahtevi za prikazivanje HTML stranica, obrada podataka iz formi ili izvršavanje drugih poslovnih logika.

## Primeri kontrolera

## [HttpGet]

Ovaj atribut se koristi za označavanje metoda u kontroleru koje treba pozvati kada se izvrši HTTP GET zahtev na određeni URL.

## [HttpGet("{id}")]

Ovaj atribut se koristi za definisanje akcije koja će odgovarati HTTP GET zahtevima sa dodatnim segmentom u putanji, u ovom slučaju "id"

## [Route(“[controler]/[action]”)]

Atribut [controller] se koristi u definisanju ruta, kako bi se automatski uklapalo ime kontrolera u putanji. Ovaj atribut se koristi u kombinaciji sa atributima [HttpGet], [HttpPost], itd., kako bi se konfigurisale putanje za određene akcije (metode) unutar kontrolera.

# Uputstvo za CORS (Cross-origin resource sharing)

## Primena CORS-a:

**API** i **Angular** strana imaju različite origine, pa na primer:

* API radi na originu: **http://localhost:5111**
* Angular strana radi na originu: **http://localhost:4200**

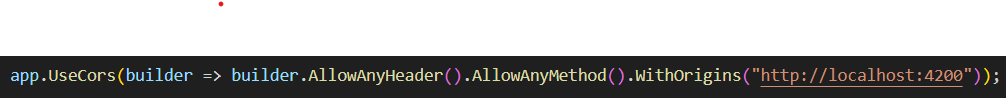
Korisnička strana šalje zahtev API-ju za određene podatke i API odgovara, međutim svaki browser ima mehanizme koji onemogućavaju saobraćaj podataka između različitih origina bez odgovarajućih dozvola, a uloga CORS-a je da nam takvu dozvolu obezbedi.

## Dodavanje podrške za CORS

Kako bi mogli da koristimo CORS potrebno je da definišemo servis koji će predstavljati njegovu podršku, definicija servisa se radi u **Program.cs**-u na sledeći način:

## 

## Definisanje middleware-a za regulaciju saobraćaja

Ovaj middleware se naziva **UseCors** i ima ulogu da definiše koji origini mogu da pristupaju podacima i da koriste njegove usluge. Takođe ima opciju da definiše i koje http zahteve može da šalje korisnička strana (GET,POST,PUT,DELETE):

* Pristup podacima se realizuje tako što se propuštaju zahtevi koji imaju header-e koji su definisani u UseCors-u. U našem slučaju postavljeno je da se propuštaju svi header.
* Takođe postavljeno je da korisnička strana može da pošalje sve vrste http zahteva pomoću metoda **AllowAnyMethod()**
* A **WithOrigins("http://localhost:4200")** služi kako bi naglasili koji origini mogu da pristupaju resursima

Proces instalacije Angular-a i Node-a

# **Linkovi:**

Node.js: https://nodejs.org/

# **Node.js**

# Prvi korak je instalacija node.js-a sa njihove zvanične stranice.Nakon čega će npm(Node Packet Manager) biti automatski dostupan.

**Angular CLI**

Zatim je potrebno instalirati Angular CLI unosom sledeće komande u terminal Visual studio code-a: *npm install -g @angular/cli*

Ova komanda globalno instalira angular CLI na vašem računaru.

**Kreiranje Angular projekta**

Potrebno je u željenom direktorijumu otvoriti terminal u VS code-u i u istom ukucati sledeću komandu: *ng new project*  gde je “project” samo placeholder za željeno ime projekta.

Nakon kreacije projekta potrebno je otvoriti ga unutar VS code-a i pokrenuti ga komandom:

*ng serve*

Ova komanda će pokrenuti lokalni server i pokrenuti Angular aplikaciju. Nakon ovoga moguće je i videti rezultat u browser-u pristupom na link “<https://localhost:4200>” (primer)

# 

# Dodavanje HTTPS-a u Angular korišćenjem mkcert

mkcert je alat za kreiranje locally-trusted development sertifikata.

Instalaciju ćemo izvršiti korišćenjem windows scoop alata.

## Scoop instalacija:

Uneti sledeće komande u powershell:

**Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser**

**Invoke-RestMethod -Uri https://get.scoop.sh | Invoke-Expression**

Nakon toga, mozemo preći na sledeci korak.

## mkcert instalacija:

Uneti sledeće komande u powershell:

**scoop bucket add extras**

**scoop install mkcert**

## Generisanje sertifikata:

Otvorite željeni terminal (cmd, bash, powershell) i navigirajte u direktorijum gde se nalazi Angular projekat. Potrebno je kreirati novi folder u koji ćemo smestiti naše lokalne sertifikate:

**cd client**

**mkdir ssl**

**cd ssl**

Nakon toga, kreiraćemo globalni sertifikat, tj dodaćemo ga u windows trust store:

**mkcert -install**

Dodajemo sertifikate za željeni URL, u našem slučaju to je localhost

**mkcert localhost**

Poslednji korak jeste modifikacija našeg angular.json fajla, kako bismo rekli našem Angular projektu da želimo da koristi SSL.

U “serve”: {} delu unutar vitičastih zagrada, dodaćemo sledeću konfiguraciju:

**"serve": {**

**"options": {**

**"ssl": true,**

**"sslCert": "./ssl/localhost.pem",**

**"sslKey": "./ssl/localhost-key.pem"**

**}**

# Struktura projekta

Struktura projekta u softverskom razvoju igra ključnu ulogu u organizaciji koda i resursa. Ona omogućava efikasno upravljanje, održavanje i skaliranje aplikacije. Osnovni elementi strukture projekta uključuju razdvajanje po slojevima, modularnu organizaciju, funkcionalnu organizaciju, fajl strukturu, konfiguraciju i resurse, testove i dokumentaciju. Razdvajanje po slojevima omogućava jasnu separaciju odgovornosti između različitih delova aplikacije, dok modularna organizacija olakšava ponovno korišćenje, skaliranje i upravljanje zavisnostima. Funkcionalna organizacija omogućava fokusiranje na određene delove aplikacije, dok fajl struktura omogućava organizaciju fajlova prema njihovoj nameni i funkcionalnosti. Konfiguracija, testovi i dokumentacija doprinose efikasnom upravljanju i dokumentovanju projekta. Važno je da struktura projekta bude jasna, organizovana i prilagođena specifičnim zahtevima i tehnologijama koje se koriste u razvoju aplikacije.

Struktura projekta u softverskom razvoju obično se organizuje kroz koncepte modula i komponenti.

## Moduli

Moduli su logičke jedinice organizacije koda koje grupišu srodne funkcionalnosti ili komponente aplikacije. Svaki modul obično sadrži komponente, servise, direktive i druge resurse potrebne za izvršavanje određene funkcionalnosti. Kroz module, aplikacija se deli na manje, lakše upravljive delove, što olakšava održavanje i skaliranje aplikacije. Moduli takođe omogućavaju modularnost i ponovno korišćenje koda, jer se mogu lako uvoziti i koristiti u različitim delovima aplikacije.

## Komponente

Komponente su osnovne građevne jedinice korisničkog interfejsa aplikacije. Svaka komponenta predstavlja jedan deo korisničkog interfejsa, kao što su dugme, forma, meni ili kompleksniji delovi kao što su karte, liste ili obrasci. Komponente sadrže HTML, CSS i JavaScript (ili TypeScript) kod koji definiše izgled i ponašanje korisničkog interfejsa. Kroz komponente, korisnički interfejs se razbija na manje, modularne delove, što olakšava razvoj, testiranje i održavanje aplikacije. Komponente se mogu koristiti višestruko i mogu se ugnježdavati unutar drugih komponenti, omogućavajući modularan pristup izgradnji korisničkog interfejsa.

U praksi, moduli i komponente se često koriste zajedno kako bi se organizovao i strukturirao kod aplikacije. Moduli grupišu srodne komponente i funkcionalnosti, dok komponente predstavljaju konkretnu implementaciju korisničkog interfejsa. Kroz njihovu efikasnu organizaciju i upotrebu, aplikacije postaju lakše za održavanje, proširivanje i razumevanje.

# Angular routes

## Upotreba

## **Angular rute** koriste se za omogućavanje navigacije između različitih delova aplikacije (komponenti) bez osvežavanja celokupne stranice. Kada koristimo Angular rute, možemo definisati kako će se različiti URL-ovi mapirati na određene komponente, omogućavajući korisnicima da prelaze između različitih "stanja" naše aplikacije.

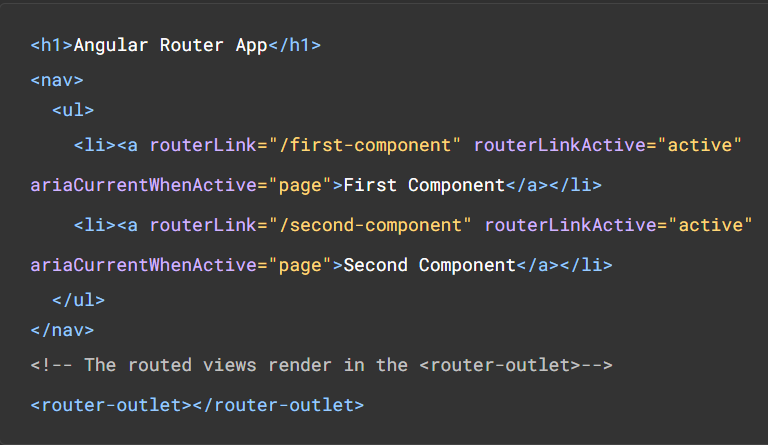
## Definisanje ruta

Svaka ruta u ovom nizu predstavlja jedan JavaScript objekat koji sadrži dva propertija, prvi properti definiše **putanju (URL)** za rutu, dok drugi predstavlja komponentu koju Angular treba da koristi za tu rutu.



## Dodavanje ruta u aplikaciju

1. Prvo se dodaju putanje na dve komponente.
2. Zatim dodeljujemo oznaku sidra na rutu atributa routerLink.
3. Podešavamo vrednost atributa na komponentu koja će se prikazati kada korisnik klikne na link.
4. Zatim ažuriramo šablon komponente da bi smo uključili **<ruter-outlet>**. Ovaj element obaveštava Angular da ažurira prikaz aplikacije sa komponentom za izabranu rutu.



Angular servisi

Klase koje se koriste za organizaciju i deljenje funkcionalnosti izmedju komponenti ili drugih delova Angular aplikacije. Oni omogućuju centralizovano upravljanje podacima,logikom i funkcionalnostima koje se mogu deliti izmedju vise delova aplikacije.

**Kljucni aspekti Angular servisa:**

* Reusability : korišćenjem servisa moguće je ponovno korišćenje istog koda nad različitim delovima aplikacije
* Dependency injection: Angular koristi dependency injection kako bi ubrizgavao servise u komponente ili druge servise. Time se olakšava razmena podataka i funkcionalnosti između delova aplikacije.
* Testing: servisi omogućuju lakše testiranje aplikacije jer mogu biti zamenjeni jednostavnijim servisima tokom faza testiranja. Time se obezbeđuje sigurnije i izolovanije testiranje bez pristupa stvarnim servisima i spoljnim podacima.
* Singletons: injekcijom servisa u bilo koju komponentu/servis angular kreira samo jednu instancu servisa koja se deli kroz celu aplikaciju. Time je omogućeno konzistentno stanje i ponašanje jer se sve vreme radi na jednoj instanci servisa.

Koriscenje servisa u Angular aplikaciji pruža organizovanu i modularnu arhitekturu.Olakšava održavanje koda i čini ga čitljivijim.