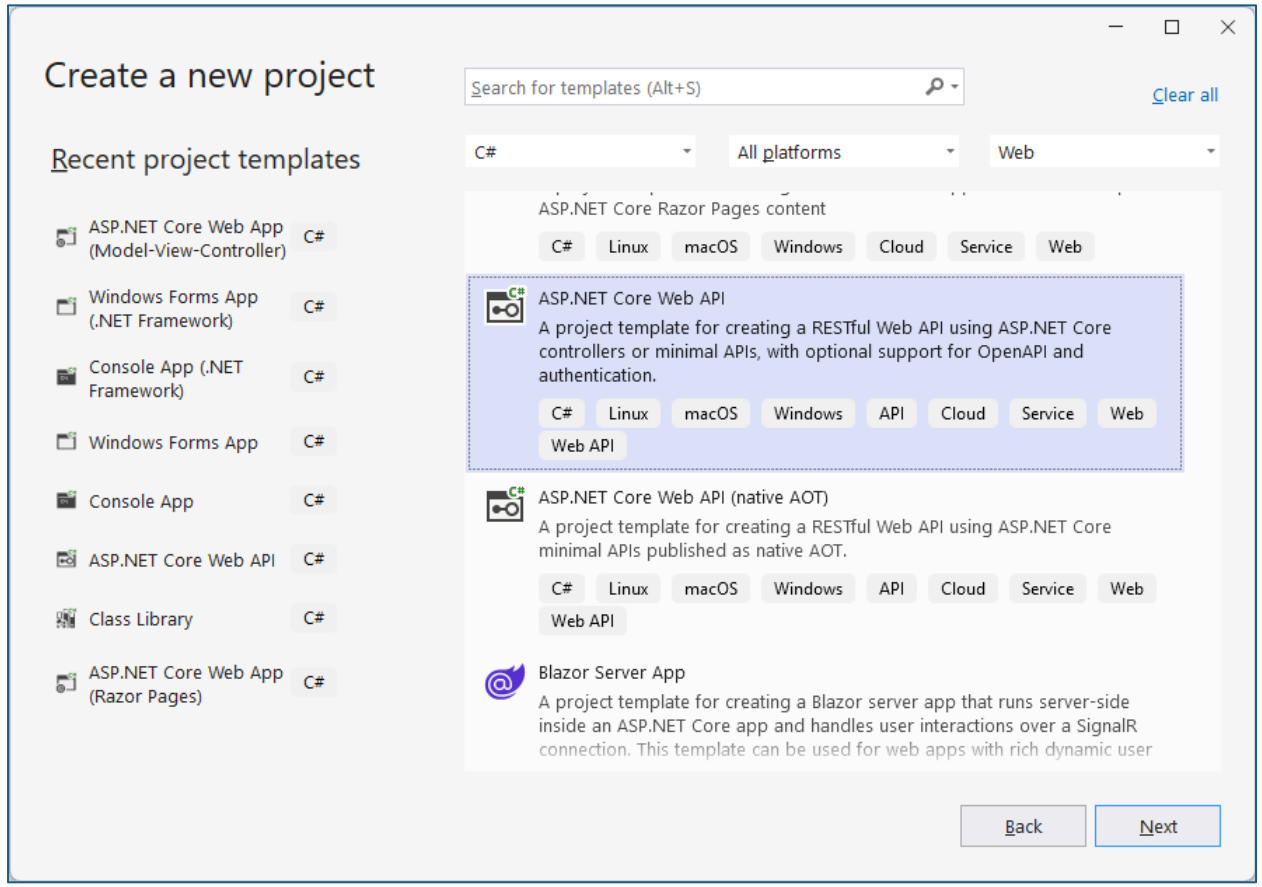
Web API

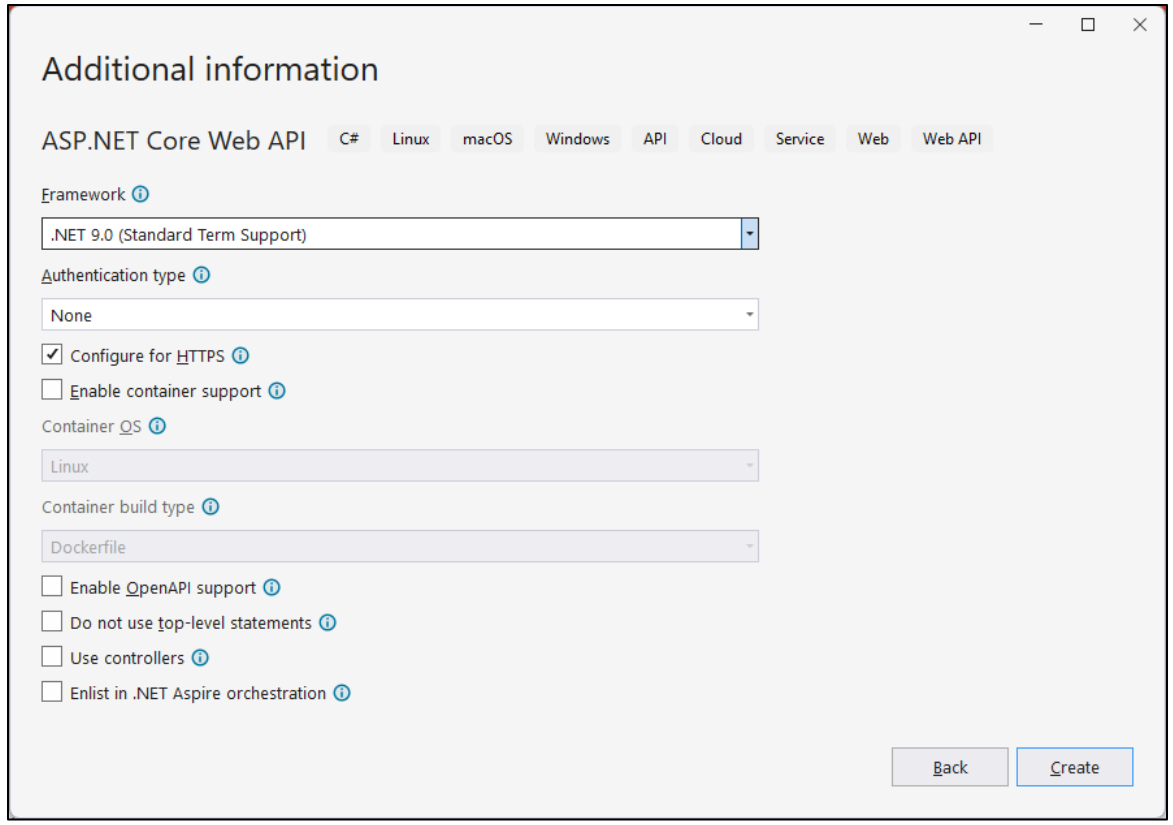
Uvod u Web API

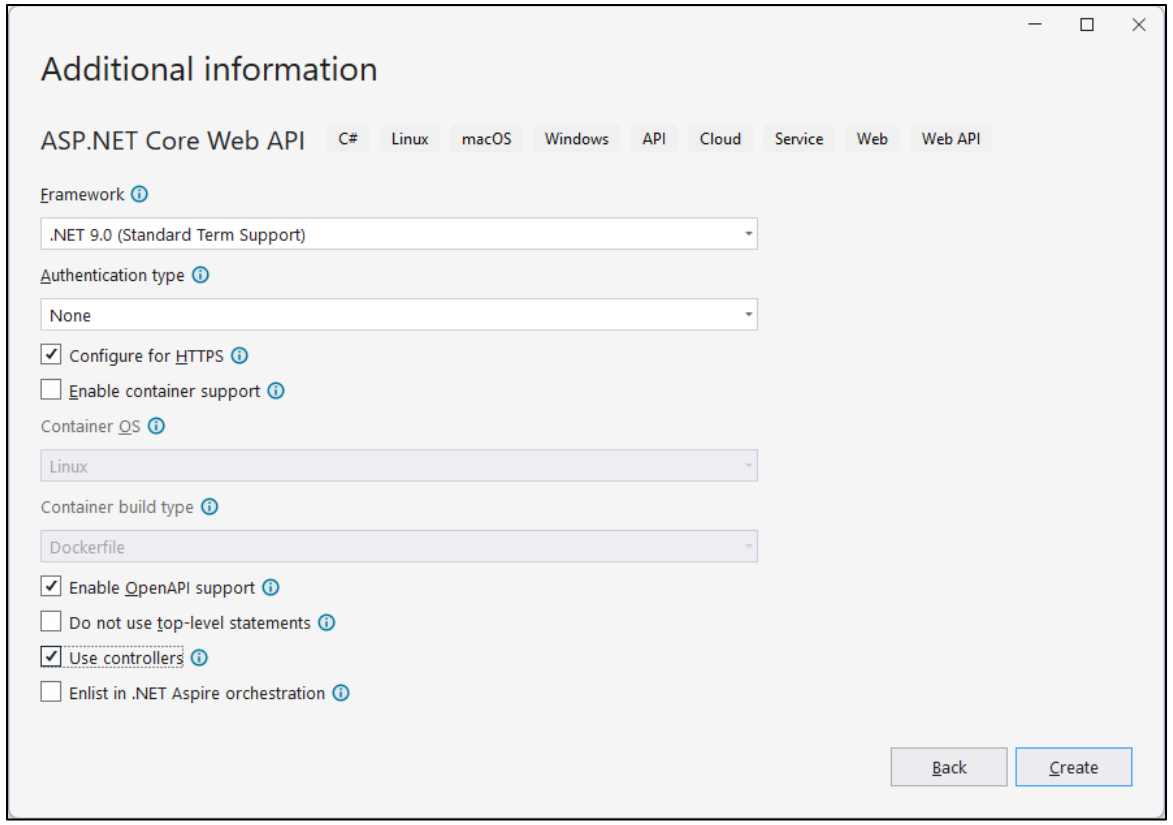
- Web API je serverska klasa koja putem HTTP protokola obezbeđuje podatke i funkcionalnosti drugim aplikacijama
- Klijentske aplikacije šalju HTTP zahteve ka Web API-ju
- Web API obrađuje zahtev, pristupa potrebnim podacima i vraća odgovor klijentu
- Podaci se najčešće razmenjuju u JSON formatu
- Web API omogućava povezivanje web, mobilnih i desktop aplikacija sa serverskim delom sistema

Kreiranje WebApi aplikacije Visual Studio



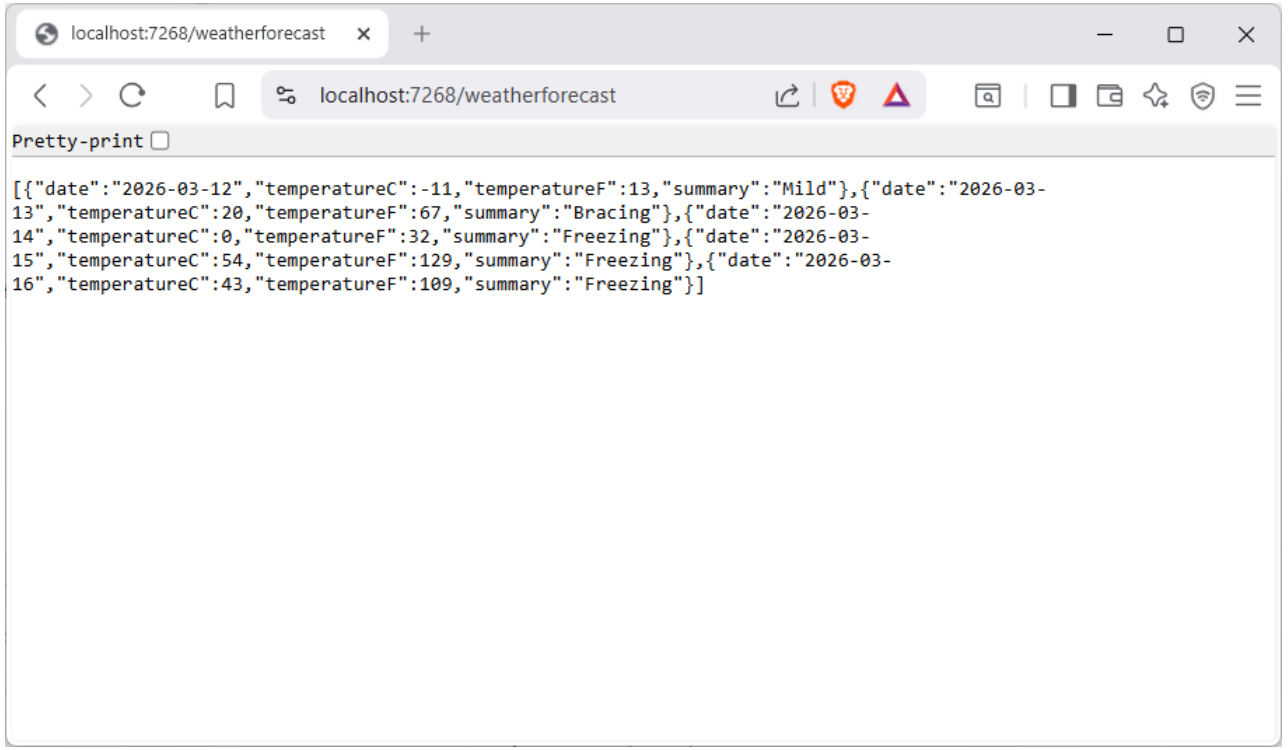
Kreiranje WebApi aplikacije Visual Studio





Test API

https://localhost:7268/weatherforecast

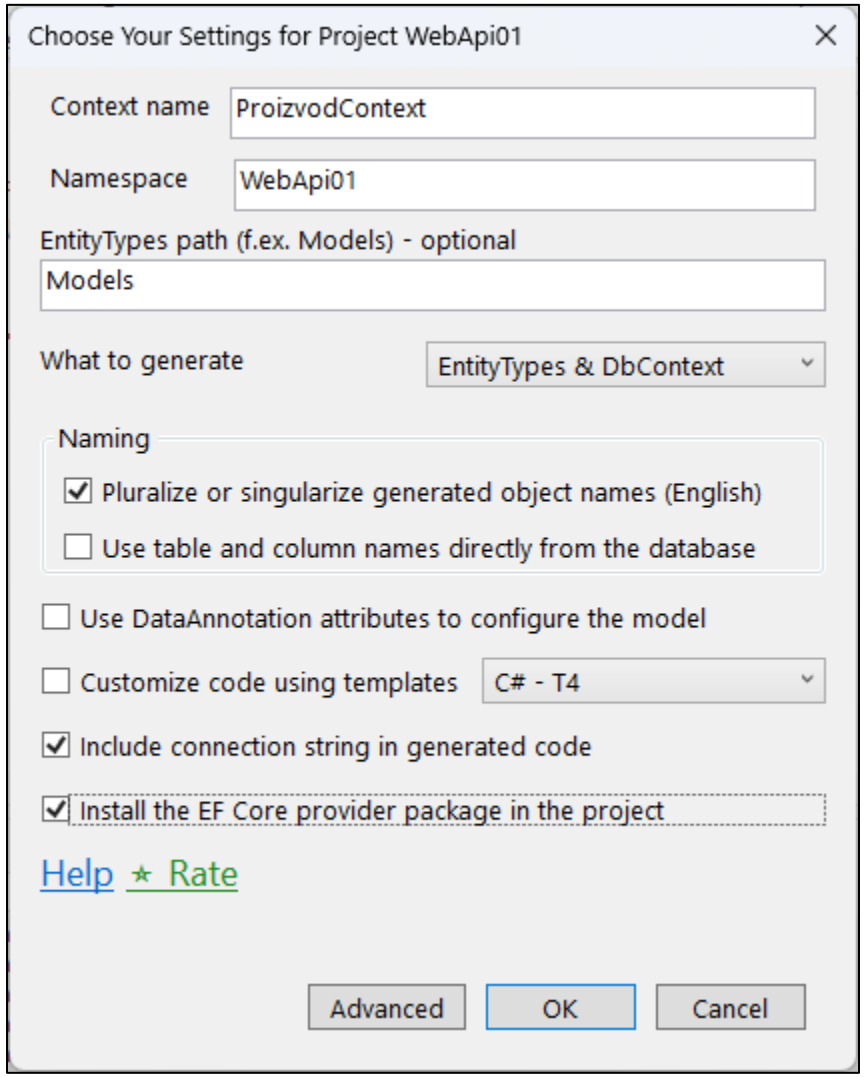


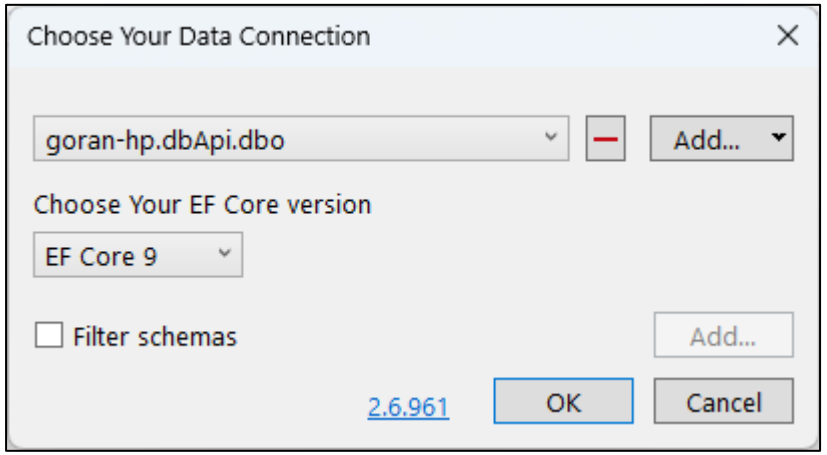
Fajl appSettings.json

```
{
  "$schema": "https://json.schemastore.org/launchsettings.json",
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": false,
      "applicationUrl": "http://localhost:5000/",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "https": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": false,
      "applicationUrl": "https://localhost:5001;http://localhost:5002",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

- Web API aplikacija sluša zahtjeve na definisanim portovima
- Port 5000 koristi se za HTTP komunikaciju
- Portovi 5001 i 5002 koriste se za HTTPS komunikaciju

EF Core Power Tools





Model klasa

```
namespace WebApi01.Models;

public partial class Proizvod
{
    public int ProizvodId { get; set; }

    public string Naziv { get; set; }

    public decimal Cena { get; set; }

    public string Opis { get; set; }
}
```

Kontekst klasa

```
namespace WebApi01.Models;

public partial class ProizvodContext : DbContext
{
    public ProizvodContext()
    {
    }

    public ProizvodContext(DbContextOptions<ProizvodContext> options)
        : base(options)
    {
    }

    public virtual DbSet<Proizvod> Proizvodi { get; set; }

}
```

Fajl appSettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=GORAN-HP;Initial Catalog=dbApi;
     Integrated Security=True;Encrypt=False"
  }
}
```

Registracija DbContext servisa

```
using Microsoft.EntityFrameworkCore;
using WebApi01.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring OpenAPI at https://aka.ms/aspnet/openapi
builder.Services.AddOpenApi();
var connectionString =
builder.Configuration.GetConnectionString("DefaultConnection");
builder.Services.AddDbContext<ProizvodContext>(options =>
    options.UseSqlServer(connectionString));

var app = builder.Build();
```

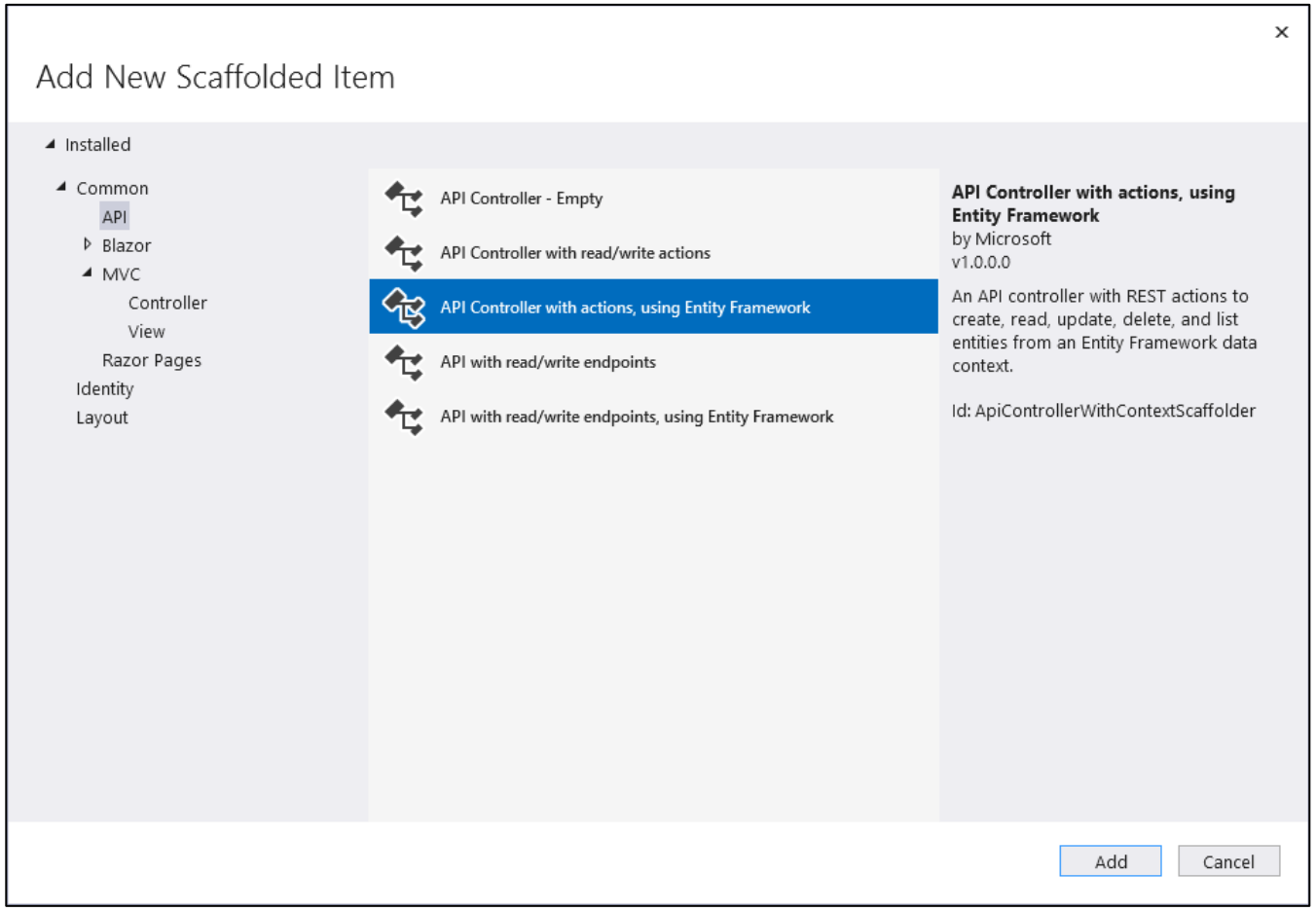
Web API Controller

- Kontroler predstavlja klasu koja obrađuje HTTP zahteve klijenta
- Kontroleri u ASP.NET Web API-ju nasleđuju baznu klasu **ControllerBase**
- Bazna klasa **ControllerBase** obezbeđuje osnovne funkcionalnosti za rad sa HTTP zahtevima i odgovorima
- Metode kontrolera odgovaraju HTTP operacijama
- Najčešće HTTP operacije su GET, POST, PUT i DELETE
- Kontroler obrađuje zahtev i vraća rezultat klijentskoj aplikaciji
- Web API omogućava komunikaciju između različitih aplikacija putem interneta

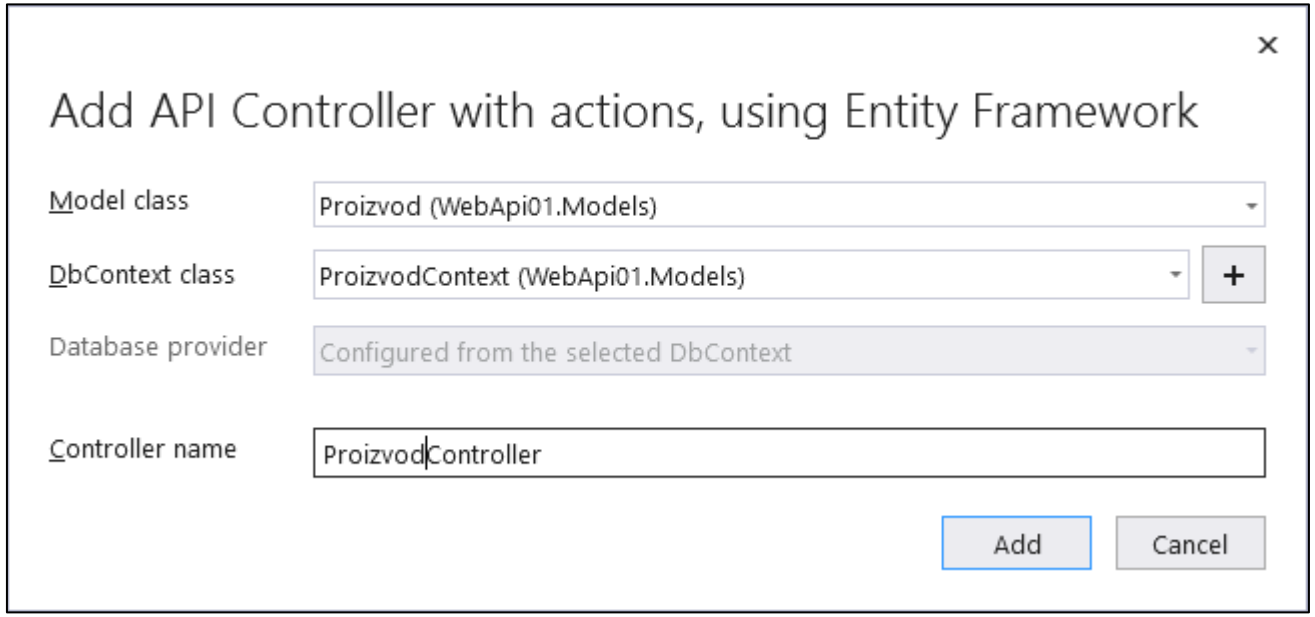
HTTP metode u Web API-ju

- GET – koristi se za preuzimanje podataka sa servera
- POST – koristi se za kreiranje novog zapisa ili resursa
- PUT – koristi se za izmenu postojećeg zapisa na serveru
- DELETE – koristi se za brisanje postojećeg zapisa
- Podaci između klijenta i servera u Web API-ju najčešće se razmenjuju u JSON formatu
- Ove metode zajedno omogućavaju **CRUD** operacije nad podacima

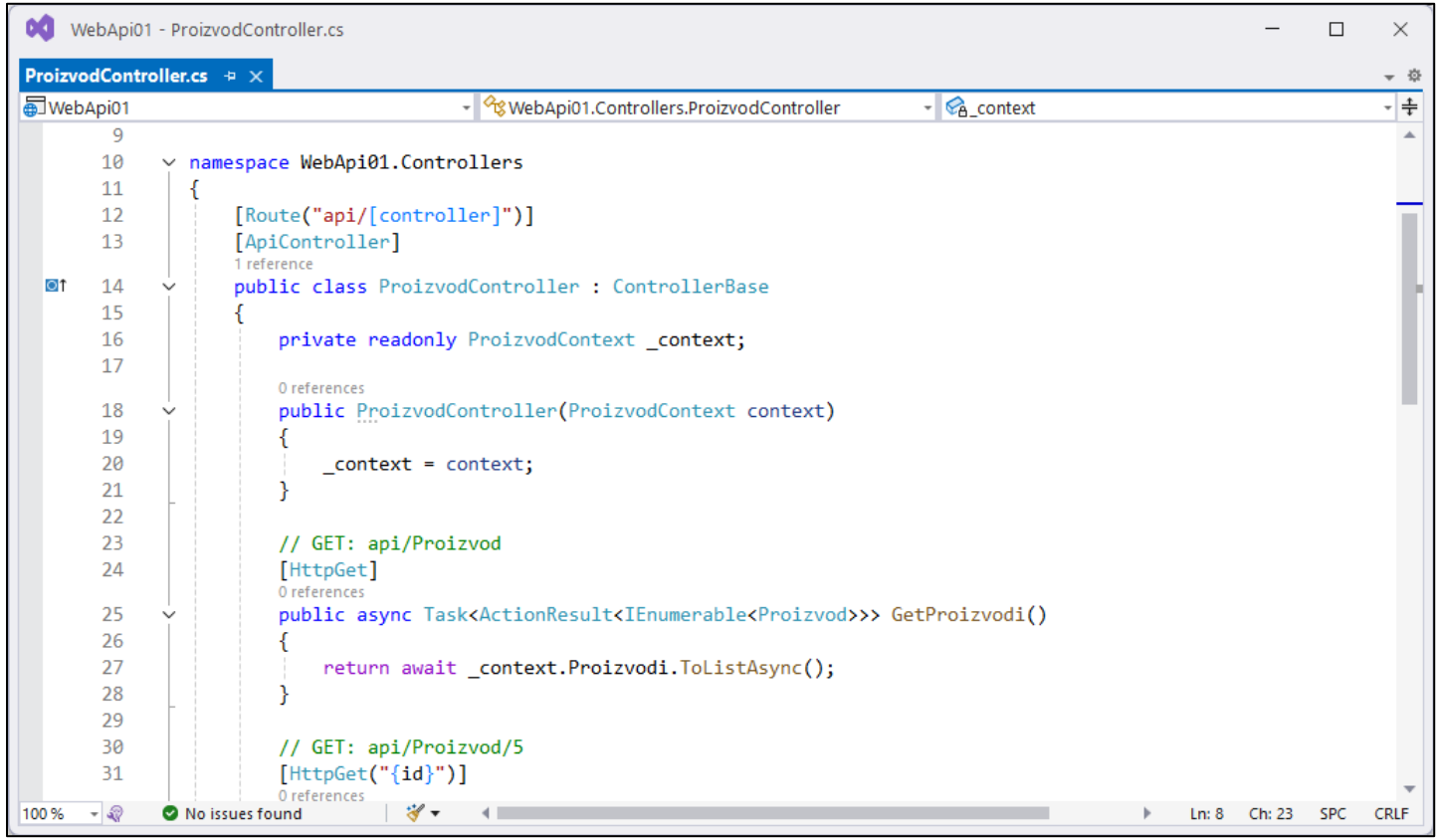
Kreiranje Web API api kontrolera



Kreiranje Web API api kontrolera



Web API kontroler



- Atribut [ApiController] označava da klasa predstavlja Web API kontroler
- Atribut [Route("api/[controller]")] definiše adresu putem koje se kontroler poziva
- Naziv kontrolera ProizvodController određuje deo URL adrese api/proizvod

GET metoda u Web API-ju

- GET metoda se koristi za preuzimanje podataka sa servera
- Ne menja podatke na serveru
- Najčešće se koristi za dobijanje jednog ili više zapisa
- Podaci se klijentu vraćaju u JSON formatu
- Primer: dohvat liste proizvoda ili pojedinačnog proizvoda prema ID-ju

GET metoda u Web API-ju

```
// GET: api/Proizvod
[HttpGet]
public async Task<ActionResult<IEnumerable<Proizvod>>> GetProizvodi()
{
    return await _context.Proizvodi.ToListAsync();
}
```

```
[HttpGet]
public ActionResult<IEnumerable<Proizvod>> GetProizvodi()
{
    return _context.Proizvodi.ToList();
}
```

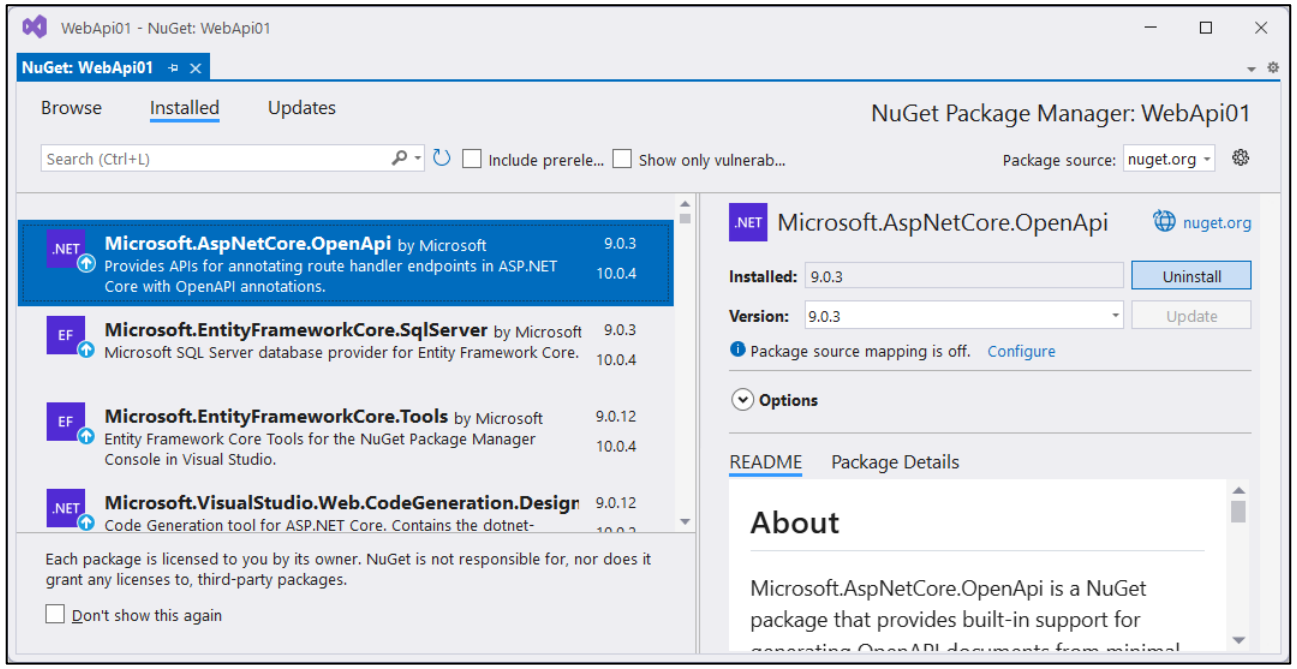
- Atribut [HttpGet] označava da metoda odgovara na GET HTTP zahtev
- URL za poziv metode je api/proizvod
- Metoda vraća listu objekata Proizvod iz baze podataka
- U prvom primeru koristi se asinhroni pristup (async i await) za efikasniji rad servera
- U drugom primeru prikazana je sinhrona verzija metode
- Obe metode vraćaju podatke klijentu u JSON formatu

GET metoda za preuzimanje jednog resursa

```
// GET: api/Proizvod/5
[HttpGet("{id}")]
public async Task<ActionResult<Proizvod>> GetProizvod(int id)
{
    var proizvod = await _context.Proizvodi.FindAsync(id);

    if (proizvod == null)
    {
        return NotFound();
    }

    return proizvod;
}
```

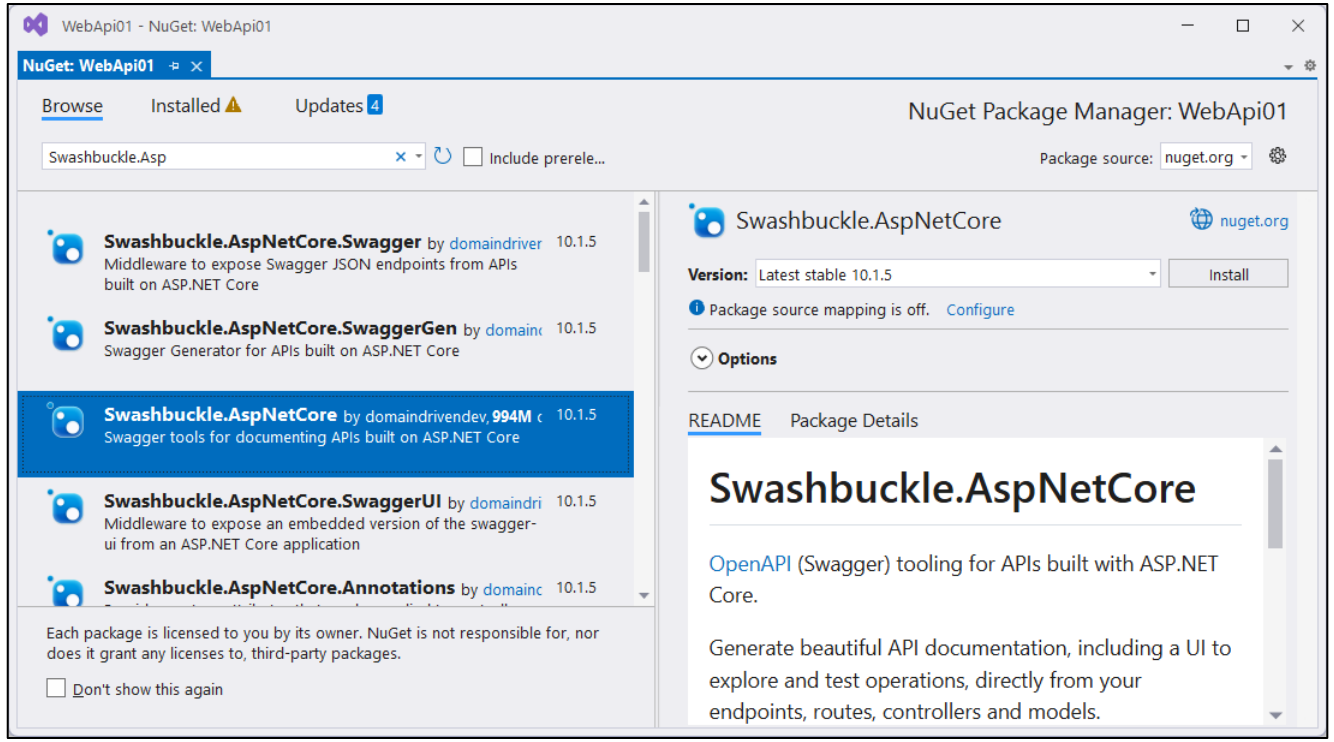
- Metoda vraća jedan proizvod prema ID-ju
- URL poziv: api/proizvod/{id}
- Atribut [HttpGet("{id}")] omogućava prosleđivanje ID-ja kroz URL
- Metoda pretražuje bazu pomoću FindAsync(id)
- Ako proizvod ne postoji vraća se NotFound() (404)

Deinstalacija OpenAPI paketa



- Paket Microsoft.AspNetCore.OpenApi uklanja se iz projekta
- Ovaj paket omogućava osnovnu OpenAPI podršku u ASP.NET Core aplikaciji
- Deinstalira se jer zbog korišćenja **Swagger-a** za dokumentaciju API-ja
- **Swagger** je alat koji omogućava pregled i testiranje API metoda u browseru
- Paket se uklanja preko NuGet Package Manager-a u Visual Studio-u

Instalacija Swagger paketa



- U projekat se instalira paket **Swashbuckle.AspNetCore**
- Paket omogućava generisanje dokumentacije za Web API
- Swagger omogućava pregled i testiranje API metoda u browseru
- Paket se instalira preko **NuGet Package Manager**-a u Visual Studio-u

Konfiguracija Swagger-a u Program.cs

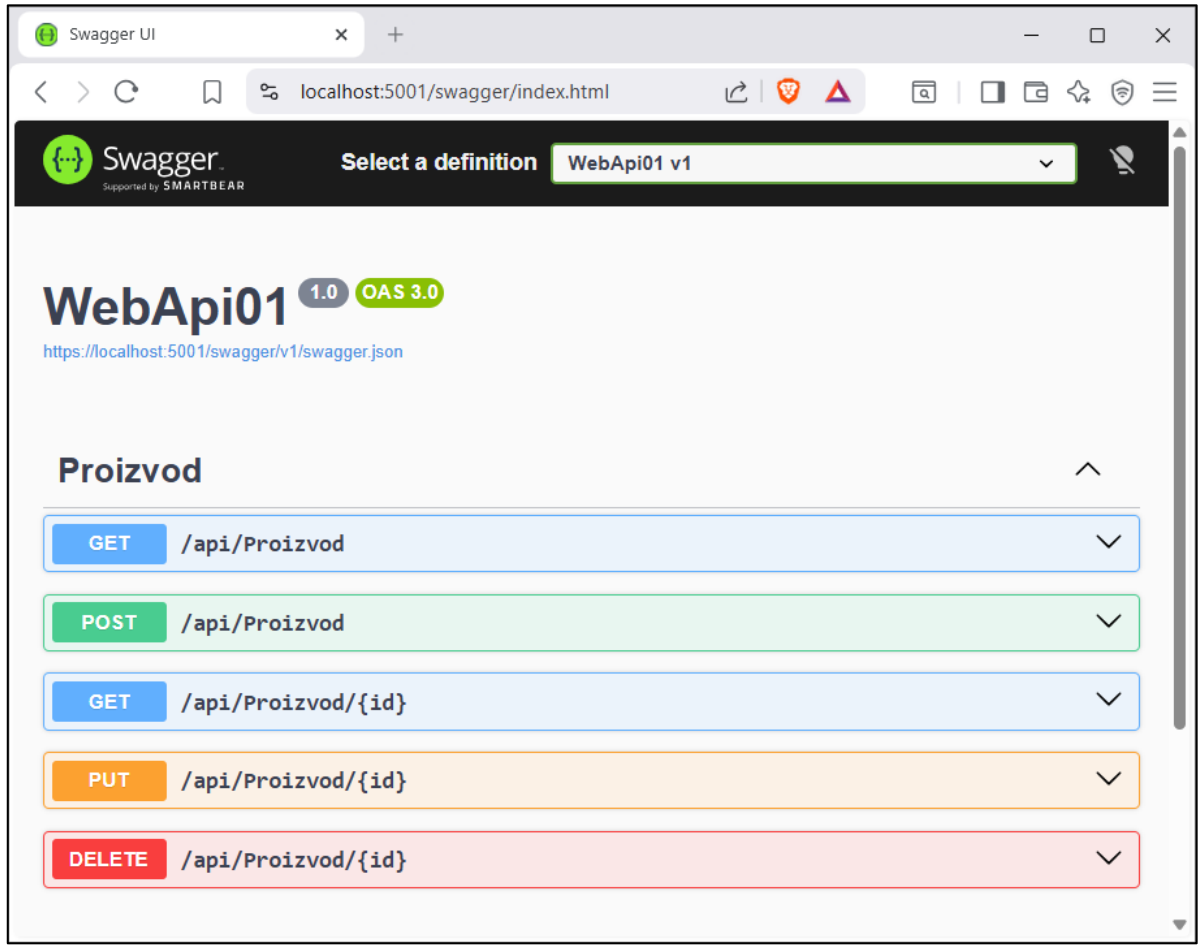
- U datoteci Program.cs vrši se konfiguracija servisa i middleware komponenti aplikacije
- Servis za otkrivanje API kontrolera i njihovih HTTP metoda registruje se pozivom metode **AddEndpointsApiExplorer**()
- Servis za generisanje Swagge dokumenta registruje se pozivom metode **AddSwaggerGen**()
- Middleware komponenta za generisanje Swagger JSON dokumenta uključuje se pozivom metode **UseSwagger**()
- Middleware komponenta za prikaz Swagger korisničkog interfejsa u browseru uključuje se pozivom metode **UseSwaggerUI**()

Program.cs

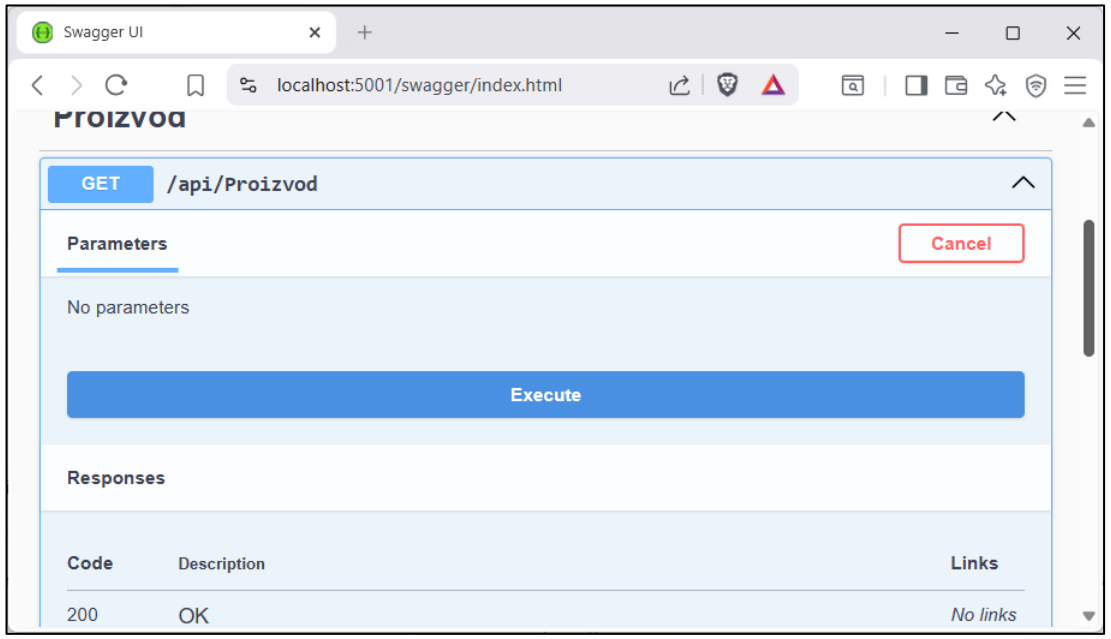
```
using Microsoft.EntityFrameworkCore;
using WebApi01.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring OpenAPI at https://aka.ms/aspnet/openapi
//builder.Services.AddOpenApi();
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
builder.Services.AddDbContext<ProizvodContext>(options =>
    options.UseSqlServer(connectionString));

builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    //app.MapOpenApi();
     app.UseSwagger();
     app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

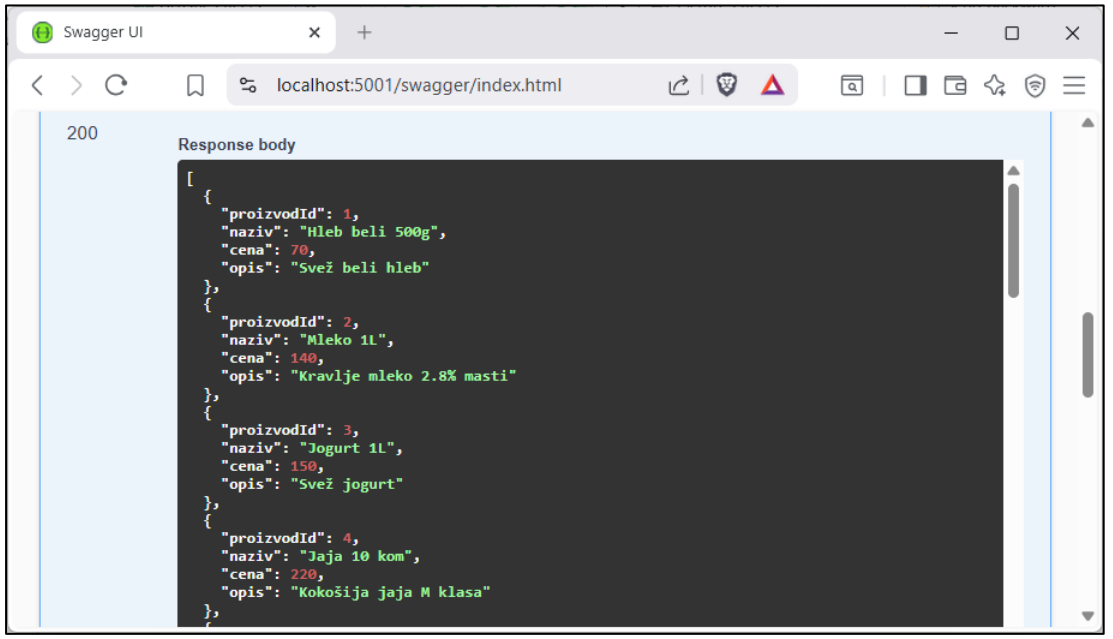
Testiranje Web API-ja pomoću Swagger UI

http://localhost:5000/swagger/index.html

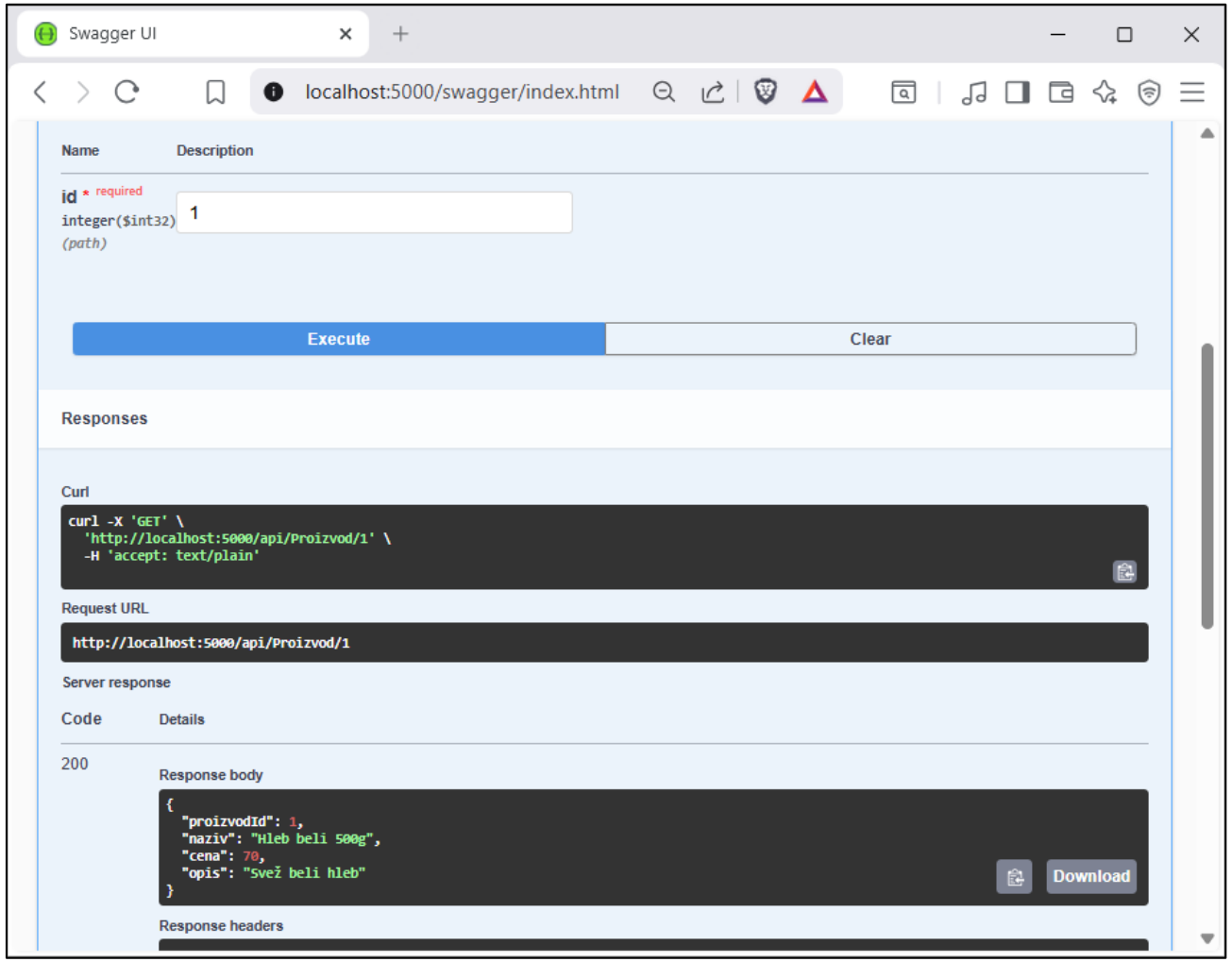


Testiranje GET zahteva

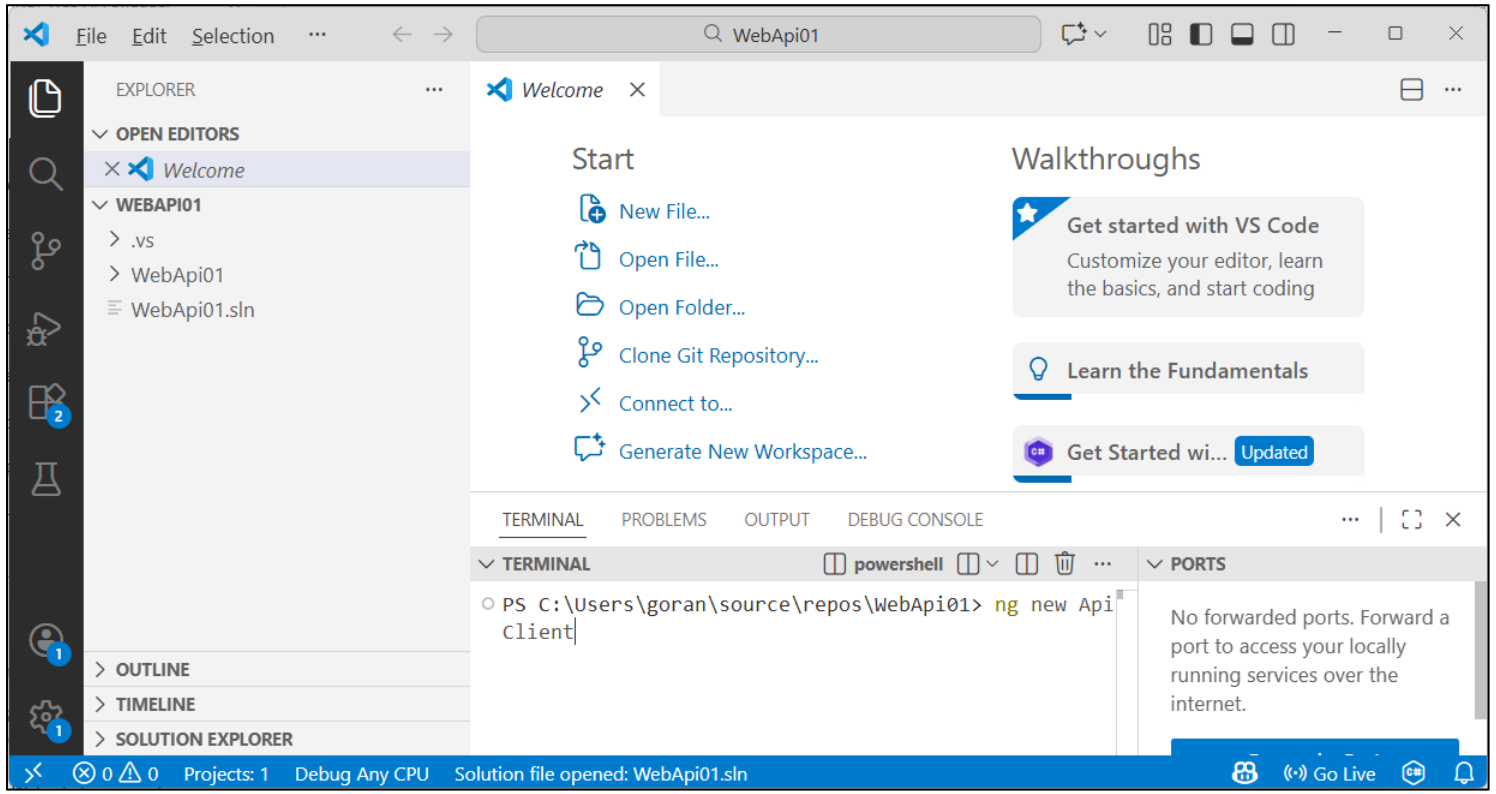




Testiranje GET zahteva koji vraća jedan resurs



Kreiranje Angular aplikacije



- Angular aplikacija kreira se u folderu repos/WebApi01
- U istom projektu nalazi se i serverski dio ASP.NET Core Web API-ja
- Komanda ng new ApiClient pravi novi Angular projekat
- Angular klijent komunicira sa ASP.NET Core Web API aplikacijom

CORS

- CORS (Cross-Origin Resource Sharing) je sigurnosni mehanizam u browseru koji kontroliše pristup resursima između različitih web aplikacija
- Angular aplikacija radi na portu 4200, a ASP.NET Core Web API na portu 5000
- Browser prepoznaje aplikacije na portovima 4200 i 5000 kao različite
- Zbog toga je potrebno omogućiti CORS komunikaciju na backend strani

Omogućavanje CORS zateva u Web Api -ju

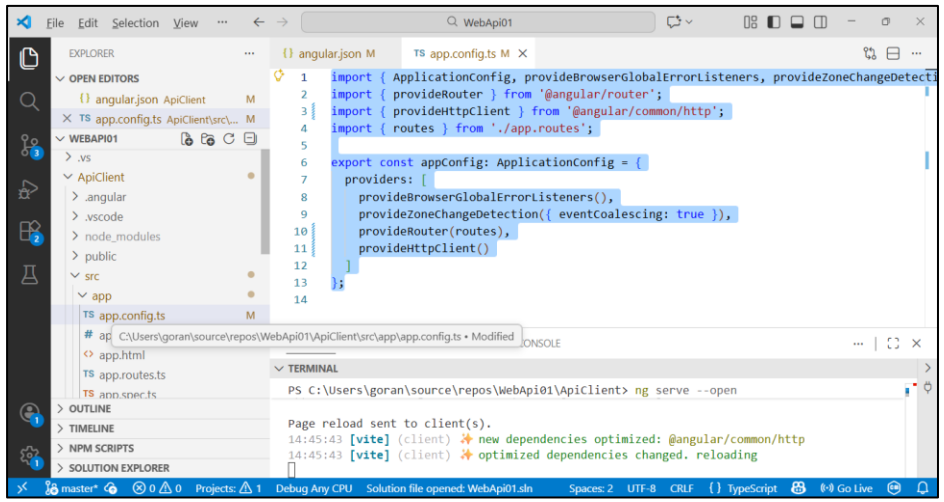
```
builder.Services.AddCors(options =>
{
    options.AddPolicy("AngularPolicy", policy =>
    {
        policy.WithOrigins("http://localhost:4200")
              .AllowAnyHeader()
              .AllowAnyMethod();
    });
});
var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    //app.MapOpenApi();
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseCors("AngularPolicy");

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Omogućavanje CORS zateva u Web API -ju

- CORS se konfiguriše u Program.cs fajlu
- AddCors() definiše pravila pristupa (policy)
- WithOrigins određuje dozvoljeni frontend (http://localhost:4200)
- AllowAnyHeader i AllowAnyMethod dozvoljavaju sve header-e i metode
- UseCors aktivira definisanu politiku u aplikaciji
- CORS mora biti uključen pre autentikacije i autorizacije
- Omogućava komunikaciju između Angular aplikacije i Web API-ja

Osnovna konfiguracija Angular aplikacije

```
import { ApplicationConfig,
provideBrowserGlobalErrorListeners,
provideZoneChangeDetection } from '@angular/core';
import { provideRouter } from '@angular/router';
import { provideHttpClient } from '@angular/common/http';
import { routes } from './app.routes';


export const appConfig: ApplicationConfig = {
  providers: [
    provideBrowserGlobalErrorListeners(),
    provideZoneChangeDetection({ eventCoalescing: true }),
    provideRouter(routes),
    provideHttpClient()
  ]
};
```



Osnovna konfiguracija Angular aplikacije

- Konfiguracija aplikacije definiše se kroz app.config.ts
- provideRouter omogućava rad sa rutama u aplikaciji
- provideHttpClient omogućava slanje HTTP zahteva ka backendu
- provideZoneChangeDetection optimizuje detekciju promena
- Globalni servisi se registruju kroz providers niz
- routes definiše navigaciju između komponenti

Kreiranje interfejsa

```
ng g interface models/proizvod
```

```
export interface Proizvod {
  proizvodId: number;
  naziv: string;
  cena: number;
  opis: string;
}
```

Kreiranje servisa sa GET metodama

```
ng g s services/proizvodService
```

```
@Injectable({
  providedIn: 'root'
})
export class ProizvodService {
  private apiUrl = '/api/proizvod';

  constructor(private http: HttpClient) { }

  getProizvodi(): Observable<Proizvod[]> {
    return this.http.get<Proizvod[]>(this.apiUrl);
  }

  getProizvod(id: number): Observable<Proizvod> {
  return this.http.get<Proizvod>(`${this.apiUrl}/${id}`);
}
}
```

Komponenta sa proizvodima

```
ng g component components/proizvodiComponent
```

```
export class ProizvodiComponent {
  naslov = 'Lista proizvoda';
  proizvodi: Proizvod[] = [];
  selektovanProizvod?: Proizvod;
  constructor(private proizvodService: ProizvodService) {}

  ngOnInit(): void {
    this.proizvodService.getProizvodi().subscribe(data => {
      this.proizvodi = data;
    });
  }

  ucitajProizvod(id: number): void {
    this.proizvodService.getProizvod(id).subscribe(p => {
      this.selektovanProizvod = p;
    });
  }
}
```

Šablon komponente sa proizvodima

```
<div class="container mt-4">


  <div class="d-flex justify-content-between align-items-center mb-3">
    <h2 class="mb-0">Lista proizvoda</h2>
    <a routerLink="/login" class="btn btn-primary">Dodaj proizvod</a>
  </div>


  <table class="table table-striped table-bordered table-hover">
    <thead class="thead-dark">
      <tr>
        <th>ID</th>
        <th>Naziv</th>
        <th>Cena</th>
        <th>Opis</th>
      </tr>
    </thead>
```

```
<tbody>
      @for (p of proizvodi; track p.proizvodId) {
        <tr (click)="ucitajProizvod(p.proizvodId)"
            [class.table-primary]="p.proizvodId === selektovanProizvod?.proizvodId">
          <td>{{ p.proizvodId }}</td>
          <td>{{ p.naziv }}</td>
          <td>{{ p.cena }}</td>
          <td>{{ p.opis }}</td>
        </tr>
      }
    </tbody>
  </table>


  @if (selektovanProizvod) {
    <div class="card mt-3">
      <div class="card-body">
        <h5 class="card-title">Detalji proizvoda</h5>
        <p><strong>ID:</strong> {{ selektovanProizvod.proizvodId }}</p>
        <p><strong>Naziv:</strong> {{ selektovanProizvod.naziv }}</p>
        <p><strong>Cena:</strong> {{ selektovanProizvod.cena }}</p>
        <p><strong>Opis:</strong> {{ selektovanProizvod.opis }}</p>
      </div>
    </div>
  }
</div>
```

Ruta za prikaz svih proizvoda

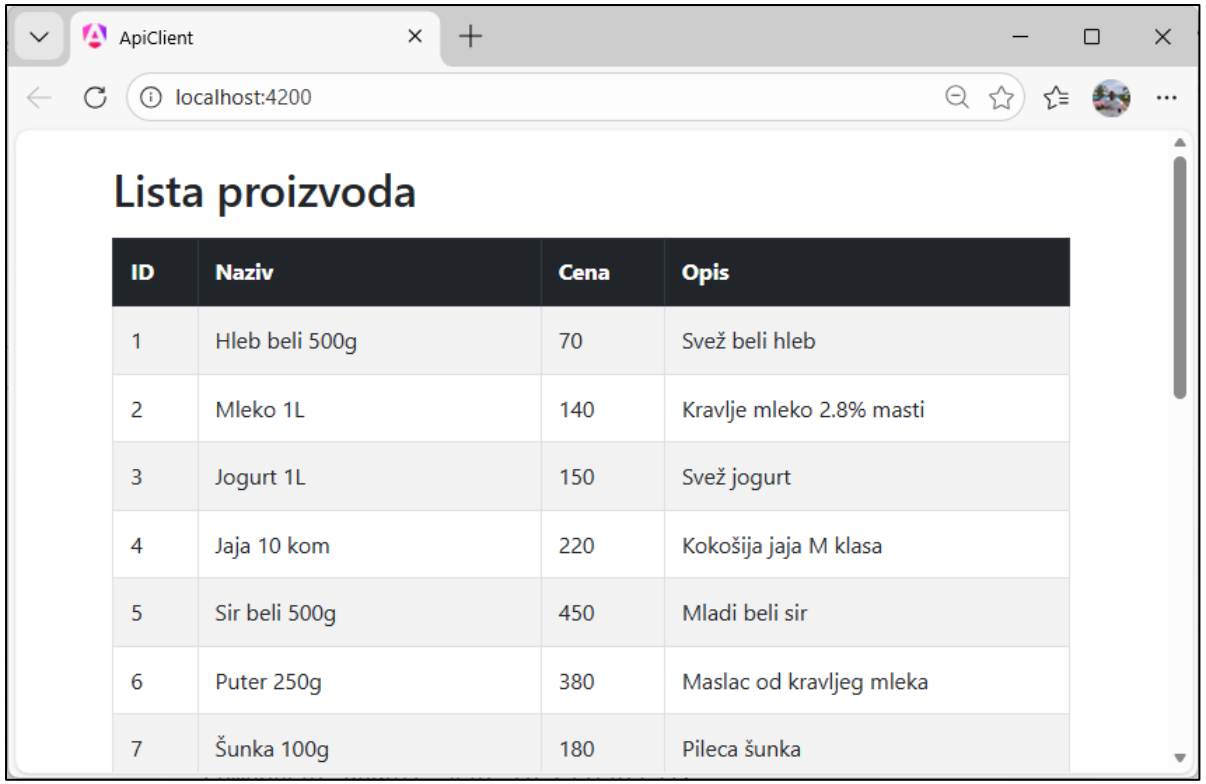
```
import { AddProizvodComponent } from './components/add-proizvod-
component/add-proizvod-component';
import { LoginComponent } from './components/login-
component/login-component';
import { ProizvodiComponent } from './components/proizvodi-
component/proizvodi-component';


export const routes: Routes = [
  { path: '', component: ProizvodiComponent }
];
```

Glavna komponenta app.html

```
@Component({
  selector: 'app-root',
  imports: [RouterOutlet],
  templateUrl: './app.html',
  styleUrl: './app.css'
})
export class App {
}
```

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

Prikaz podataka



Pitanje 1

Šta je Web API?

a. Serverska aplikacija koja putem HTTP protokola prima zahteve klijenata i vraća podatke najčešće u JSON format
b. Klijentska komponenta koja prikazuje podatke u browseru i upravlja izgledom korisničkog interfejsa
c. Deo severske aplikacije koji služi konvertovanje podataka iz baze u JSON format

Odgovor: a

Pitanje 2

Koja bazna klasa se najčešće koristi za kreiranje Web API kontrolera u ASP.NET Core aplikaciji?

a. Controller
b. ControllerBase
c. HttpController

Odgovor: b

Pitanje 3

Koja je uloga atributa [ApiController] u ASP.NET Core Web API kontroleru?

a. Označava da klasa predstavlja Web API kontroler
b. Definiše URL adresu preko koje se kontroler poziva
c. Pokreće Web API aplikaciju na određenom portu

Odgovor: a

Pitanje 4

U Web API aplikaciji za kontroler ProizvodController definisan je atribut [Route("api/[controller]")]. Kako se poziva GET metoda ovog kontrolera?

a. api/proizvodcontroller
b. proizvod/get
c. api/proizvod

Odgovor: c

Pitanje 5

U Web API aplikaciji postoji kontroler ProizvodController, a GET metoda ima atribut [HttpGet("{id}")]. Kojom rutom se poziva proizvod sa ID vrednošću 5?

a. api/proizvod/id/5
b. api/proizvod/5
c. proizvod/get/5

Odgovor: b

Pitanje 6

Koja je uloga Swagger-a u Web API aplikaciji?

a. Omogućava generisanje dokumentacije, pregled i testiranje Web API metoda u browseru
b. Omogućava čuvanje podataka u bazi i izvršavanje SQL upita nad tabelama
c. Omogućava prikaz korisničkog interfejsa klijentske aplikacije i upravljanje HTML stranicama

Odgovor: a

Pitanje 7

Šta je CORS(Cross-Origin Resource Sharing)?

a. Mehanizam koji automatski pretvara HTTP odgovore u JSON format
b. Sigurnosni mehanizam u browseru koji kontroliše pristup resursima između različitih web aplikacija
c. Servis klijentske aplikacije koji služi za prikaz podataka dobijenih sa servera

Odgovor: b

Pitanje 8

Koja HTTP metoda se u Web API-ju koristi za izmenu postojećeg resursa na serveru?

a. PUT
b. GET
c. POST

Odgovor: a