Web API autentifikacija

Tok autentikacije u Web API + Angular aplikaciji

- Korisnik unosi email i lozinku na login formi
- Angular šalje POST zahtev ka Web API-ju: /api/Auth/login
- Backend proverava podatke korisnika
- Ako su podaci ispravni, backend generiše JWT token
- Token se vraća klijentu i čuva u localStorage
- Kod narednih zahteva token se automatski dodaje u autorizacioni heder
- Svaki naredni zahtev šalje token u autorizacionom hederu
- Backend validira token pre obrade zahteva
- Ako je token validan, zahtev se izvršava
- Ako token nije validan ili ne postoji, vraća se 401 Unauthorized

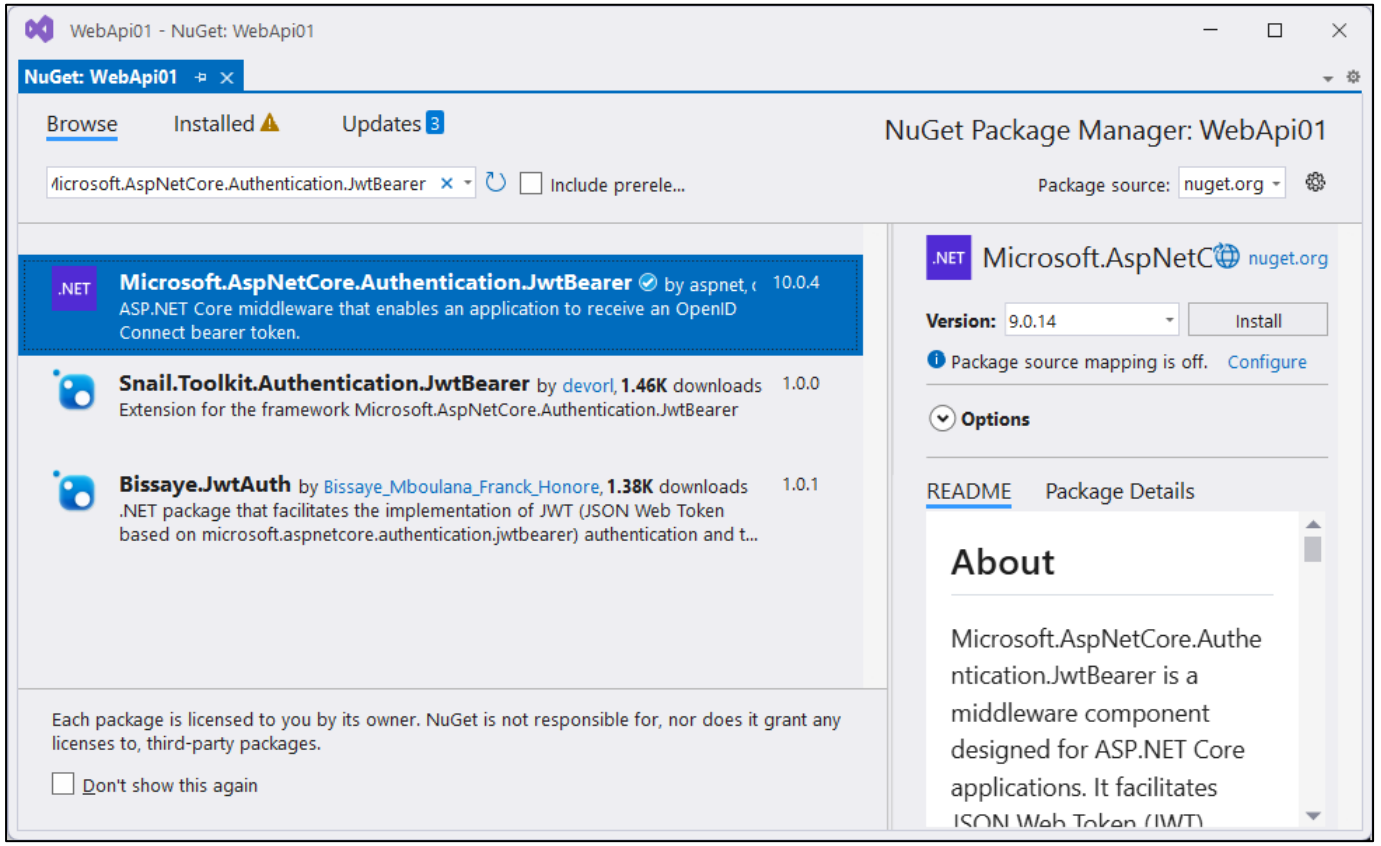
appSettings.json

```
"Jwt": {
  "Key": "OvoJeMojTajniKljucZaJwtToken123456789",
  "Issuer": "WebApi01",
  "Audience": "WebApi01Users",
  "DurationInMinutes": 60
}
```

- Konfiguracija za generisanje i validaciju JWT tokena
- Key – tajni ključ za potpisivanje tokena
- Issuer – aplikacija koja izdaje token
- Audience – aplikacija ili korisnici kojima je token namenjen
- DurationInMinutes – vreme važenja tokena

Instalacija JWT biblioteke

Microsoft.AspNetCore.Authentication.JwtBearer



Konfiguracija autentikacije u ASP.NET Core

```
AuthenticationBuilder ab = builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme =
    JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
});
```

- Autentikacija se konfiguriše pomoću AddAuthentication() metode
- Postavlja se podrazumevana autentikaciona šema
- JwtBearerDefaults.AuthenticationScheme definiše korišćenje JWT tokena
- DefaultAuthenticateScheme određuje način identifikacije korisnika
- DefaultChallengeScheme definiše ponašanje kada korisnik nije autentifikovan
- Ova konfiguracija omogućava korišćenje JWT autentikacije u aplikaciji

Konfiguracija JWT validacije u ASP.NET Core

```
 var jwtSettings = builder.Configuration.GetSection("Jwt");
 var key = jwtSettings["Key"];

ab.AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = jwtSettings["Issuer"],
            ValidAudience = jwtSettings["Audience"],
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(key!))
        };
    });

builder.Services.AddAuthorization();
```

JSON Web Token (JWT)

- JWT predstavlja digitalni token za autentikaciju korisnika
- Sadrži informacije o korisniku (npr. username, role)
- Token generiše server nakon uspešnog logina
- Klijent čuva token (npr. u localStorage)
- Token se šalje uz svaki zahtev ka serveru
- Server proverava validnost tokena

Konfiguracija JWT validacije u ASP.NET Core

- JWT podešavanja se učitavaju iz appsettings.json
- Key predstavlja tajni ključ za potpisivanje tokena
- AddJwtBearer() konfiguriše validaciju JWT tokena
- ValidateIssuer i ValidateAudience proveravaju izvor i namenu tokena
- ValidateLifetime proverava da li je token istekao
- ValidateIssuerSigningKey osigurava integritet tokena
- IssuerSigningKey definiše ključ za verifikaciju potpisa
- ValidIssuer i ValidAudience moraju odgovarati vrednostima u tokenu

Middleware za autentikaciju i autorizaciju

```
app.UseHttpsRedirection();

app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();

app.Run();
```

- UseAuthentication se dodaje tek nakon konfiguracije JWT autentikacije
- Bez njega token se ne proverava u zahtevima
- UseAuthorization proverava dozvole pristupa resursima
- UseAuthentication mora ići pre UseAuthorization

POST metoda API kontrolera bez autorizacije

```
[HttpPost]
public async Task<ActionResult<Proizvod>> PostProizvod(Proizvod proizvod)
{
    _context.Proizvodi.Add(proizvod);
    await _context.SaveChangesAsync();

    return CreatedAtAction("GetProizvod", new { id = proizvod.ProizvodId }, proizvod);
}
```

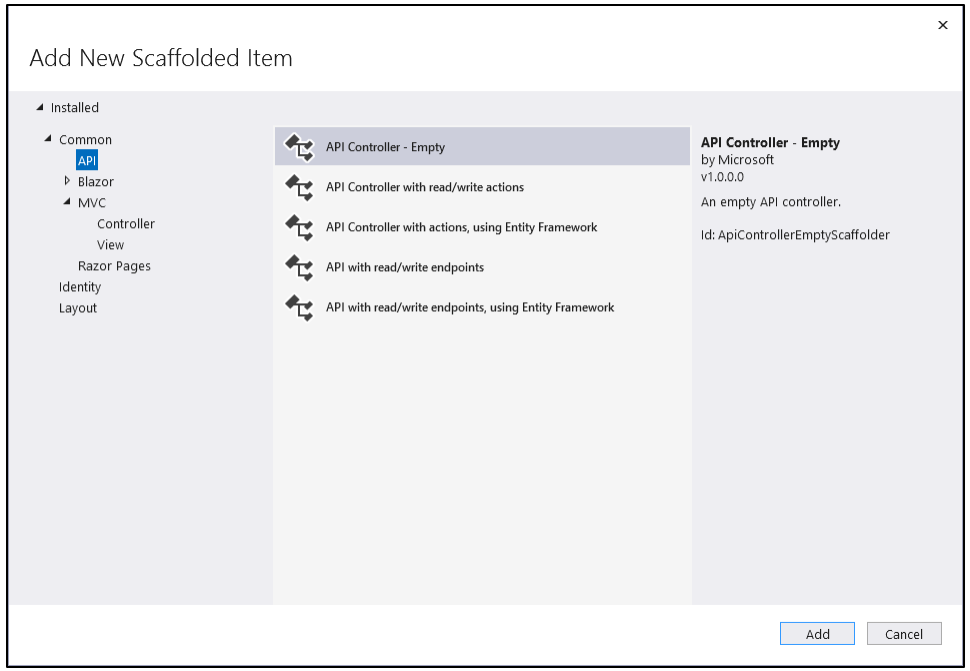
- HttpPost označava endpoint za kreiranje resursa
- Metoda prima objekat Proizvod iz zahteva
- Podaci se čuvaju u bazi pomoću DbContext-a
- SaveChangesAsync upisuje podatke u bazu
- CreatedAtAction vraća status 201 Created i proizvod u response-u
- Endpoint je dostupan bez autentikacije
- Svaki korisnik može dodati proizvod bez ograničenja

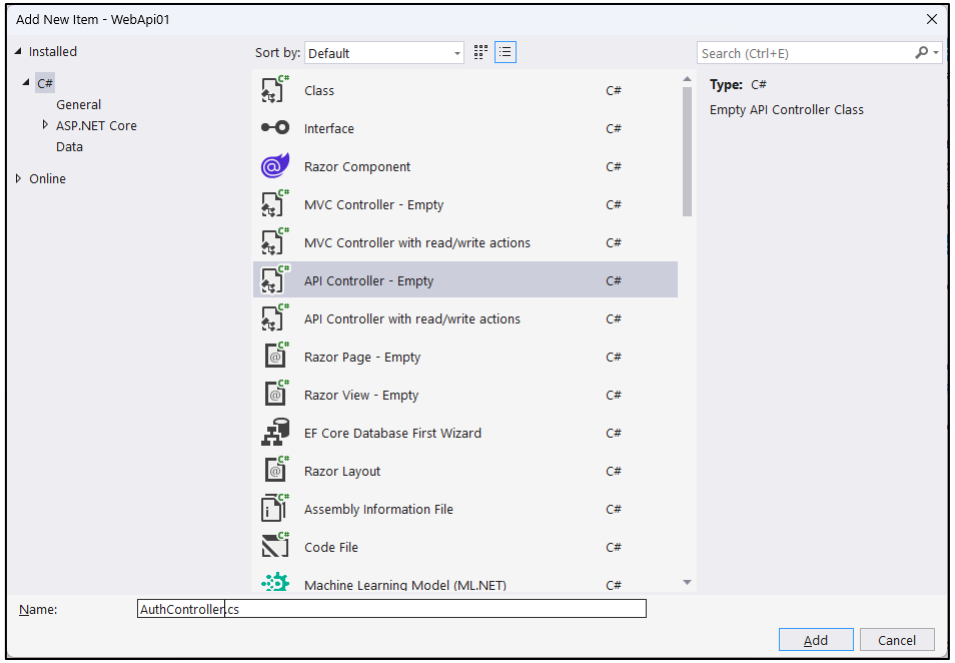
LoginModel-model za autentikaciju korisnika

```
namespace WebApi01.Models
{
    public class LoginModel
    {
        public string Username { get; set; } = string.Empty;
        public string Password { get; set; } = string.Empty;
    }
}
```

LoginModel je objekat koji nosi korisničke kredencijale ka serveru

Api kontroler za autentifikaciju





- Dodaje se novi API Controller (Empty)
- Naziv kontrolera: AuthController
- Služi za logovanje i generisanje JWT tokena
- Prima LoginModel kao ulaz
- Vraća token nakon uspešne prijave

Biblioteke za JWT autentikaciju

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using WebApi01.Models;
```

- System.IdentityModel.Tokens.Jwt za kreiranje JWT tokena
- Microsoft.IdentityModel.Tokens za validaciju tokena
- System.Security.Claims za korisničke podatke u tokenu
- System.Text za rad sa ključem (encoding)

Učitavanje konfiguracije u autentifikacioni kontroler

```
private readonly IConfiguration _configuration;

public AuthController(IConfiguration configuration)
{
    _configuration = configuration;
}
```

- **IConfiguration** omogućava pristup podešavanjima aplikacije iz appsettings.json
- omogućava čitanje vrednosti kao što su **JWT Key, Issuer i Audience**

Login metoda -validacija korisnika i priprema tokena

```
public IActionResult Login([FromBody] LoginModel model)
{
    if (model.Username != "admin" || model.Password != "admin123")
    {
        return Unauthorized("Pogresno korisnicko ime ili lozinka");
    }
    var claims = new[]
    {
    new Claim(ClaimTypes.Name, model.Username),
    new Claim(ClaimTypes.Role, "Admin")
     };
     // nastavak ovde
 }
```

- Metoda Login autentifikaciong kontrolera prima podatke za prijavu kroz LoginModel objekat iz HTTP zahteva
- Proverava se korisničko ime i lozinka
- Ako podaci nisu tačni vraća se Unauthorized (401) odgovor
- Nakon uspešne prijave kreiraju se **claims** (informacije o korisniku)
- Claims sadrže podatke kao što su ime korisnika i njegova uloga
- Ovi podaci se kasnije koriste za generisanje JWT tokena i kontrolu pristupa API metodama

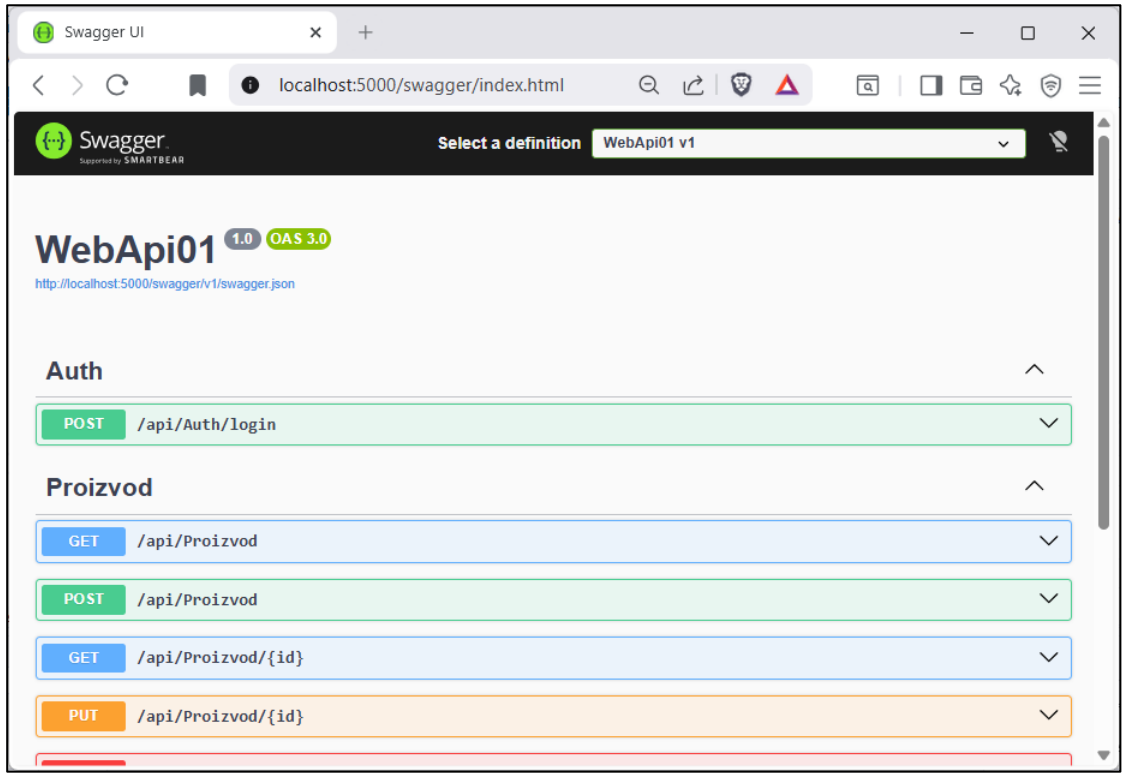
Login metoda -generisanje JWT tokena

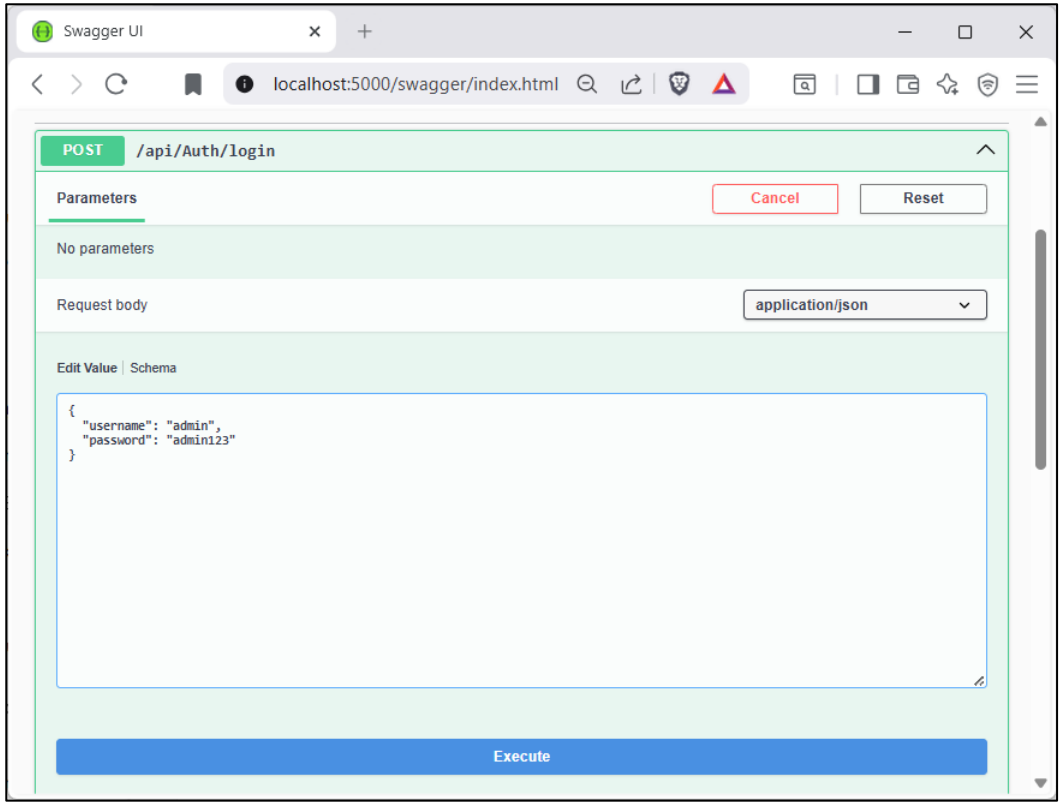
```
var key = new SymmetricSecurityKey(
    Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]!));

var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(
    issuer: _configuration["Jwt:Issuer"],
    audience: _configuration["Jwt:Audience"],
    claims: claims,
    expires: DateTime.Now.AddMinutes(
        Convert.ToDouble(_configuration["Jwt:DurationInMinutes"])),
    signingCredentials: creds
);

var jwt = new JwtSecurityTokenHandler().WriteToken(token);

return Ok(new
{
    token = jwt
});
```

- Ključ za potpisivanje se učitava iz konfiguracije (Jwt:Key)
- Kreiraju se **SigningCredentials** čime se definiše kako server potpisuje JWT token
- Formira se JWT token sa podacima: issuer, audience i claims
- Definiše se vreme isteka tokena (**DurationInMinutes**)
- Token se pretvara u string pomoću **JwtSecurityTokenHandler**
- Generisani token se vraća klijentu kao odgovor API-ja

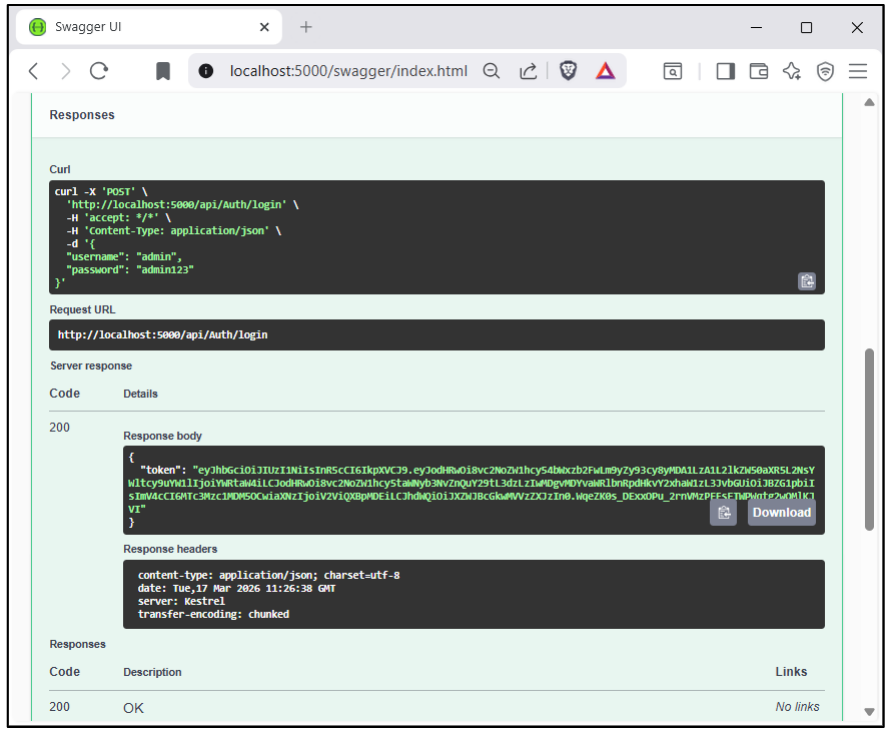
Testiranje login-a u Swaggeru

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





Testiranje login-a i dobijanje JWT tokena



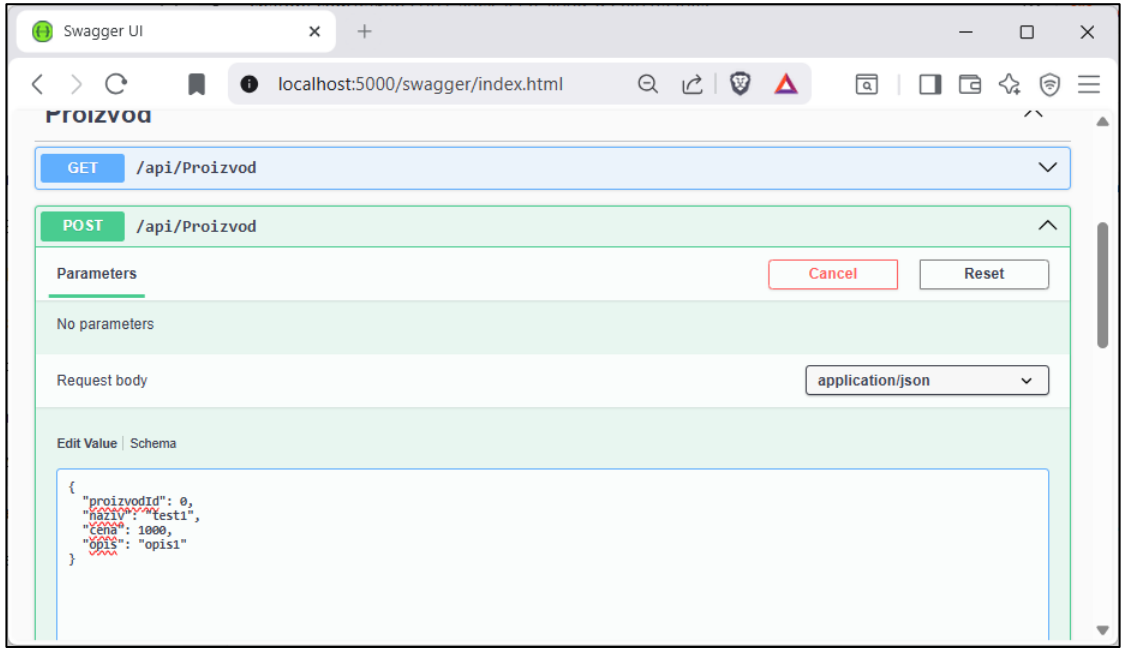
Korisnik šalje login podatke, a API nakon uspešne autentifikacije vraća JWT token koji se koristi za pristup zaštićenim metodama

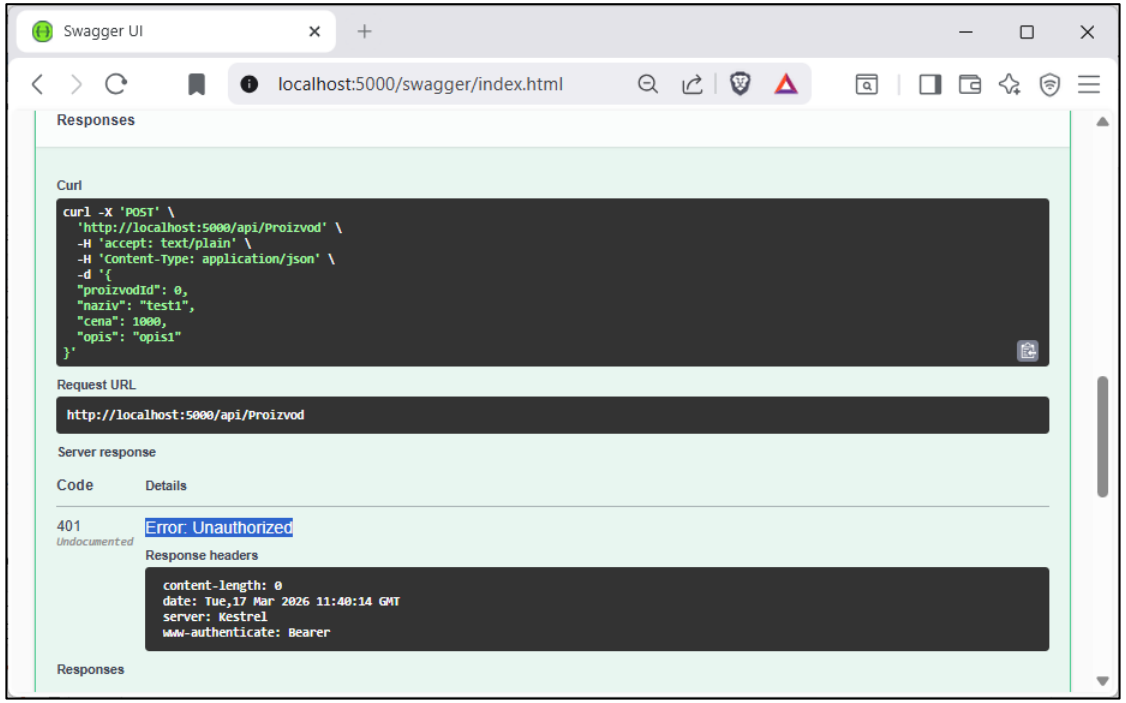
Autorizacija POST, PUT i DELETE metode

```
[Authorize]
[HttpPost]
public async Task<ActionResult<Proizvod>> PostProizvod(Proizvod proizvod)
{
    _context.Proizvodi.Add(proizvod);
    await _context.SaveChangesAsync();

    return CreatedAtAction("GetProizvod", new { id = proizvod.ProizvodId }, proizvod);
}
```

- Atribut [**Authorize**] ograničava pristup metodi samo prijavljenim korisnicima
- Nezaštićene metode mogu ostati javne, dok su POST, PUT i DELETE zaštićene
- Korisnik mora poslati važeći JWT token da bi izvršio izmenu podataka
- Bez tokena ili sa neispravnim tokenom API vraća 401 Unauthorized

Testiranje zaštićenih metoda bez tokena





- Pokušaj poziva zaštićene metode bez autentifikacije
- API odbija zahtev jer nedostaje token
- Server vraća odgovor **401 Unauthorized**
- Pristup POST, PUT i DELETE metodama nije dozvoljen bez validnog tokena
- Potvrda da je zaštita API-ja uspešno implementirana

Kreiranje interfejsa u folderu models

Fajl login.ts

```
export interface LoginRequest {
  username: string;
  password: string;
}


export interface LoginResponse {
  token: string;
}
```

Angular servis za autentifikaciju

ng g s services/authService

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { LoginRequest, LoginResponse } from '../models/login';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root',
})
export class AuthService {
  private apiUrl = 'http://localhost:5000/api/Auth';
  constructor(private http: HttpClient) {}

  login(data: LoginRequest): Observable<LoginResponse> {
    return this.http.post<LoginResponse>(`${this.apiUrl}/login`, data);
  }
}
```

Metoda login Angular servisa

- Servis komunicira sa backend API-jem za autentifikaciju korisnika
- Koristi **HttpClient** za slanje HTTP zahteva
- Definiše osnovni URL za Auth API
- Metoda login() šalje korisničke podatke na backend
- Koristi tipove **LoginRequest** i **LoginResponse**
- Backend vraća JWT token za autorizaciju daljih zahteva

Komponenta za logovanje

```
ng g c  components/loginComponent
```

```
import { Component } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { AuthService } from '../../services/auth-service';
@Component({
  selector: 'app-login-component',
  imports: [FormsModule],
  templateUrl: './login-component.html',
  styleUrl: './login-component.css',
})
export class LoginComponent {
  username: string = '';
  password: string = '';


 constructor(private authService: AuthService) {}
```

Komponenta za logovanje

```
onSubmit() {
  const data = {
    username: this.username,
    password: this.password
  };

  this.authService.login(data).subscribe({
    next: (response) => {
      console.log('TOKEN:', response.token);
    },
    error: (err) => {
      console.log('GRESKA', err);
    }
  });
}
```

Šablon komponente za logovanje

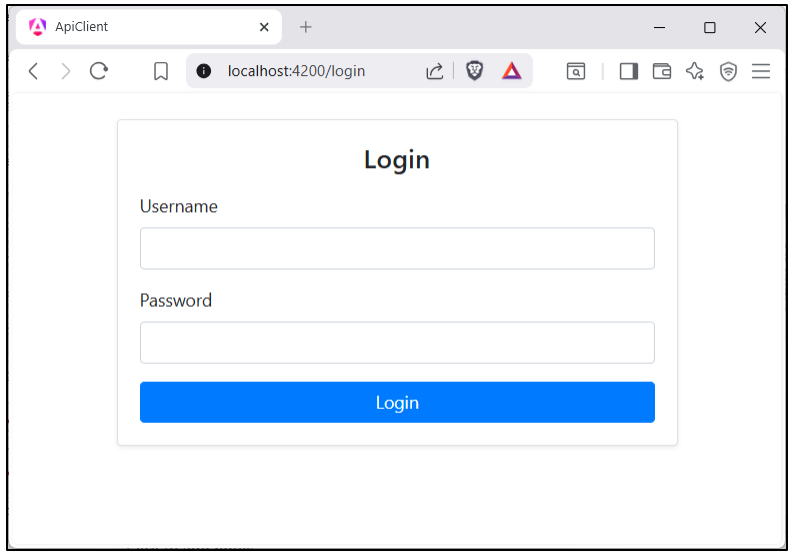
```
<h2 class="mb-4">Login</h2>


<form #loginForm="ngForm" (ngSubmit)="onSubmit()">


  <div class="form-group row">
    <label class="col-sm-2 col-form-label">Username</label>
    <div class="col-sm-4">
      <input type="text" class="form-control" name="username" [(ngModel)]="username" required>
    </div>
  </div>


  <div class="form-group row">
    <label class="col-sm-2 col-form-label">Password</label>
    <div class="col-sm-4">
      <input type="password" class="form-control" name="password" [(ngModel)]="password" required>
    </div>
  </div>


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


</form>
```

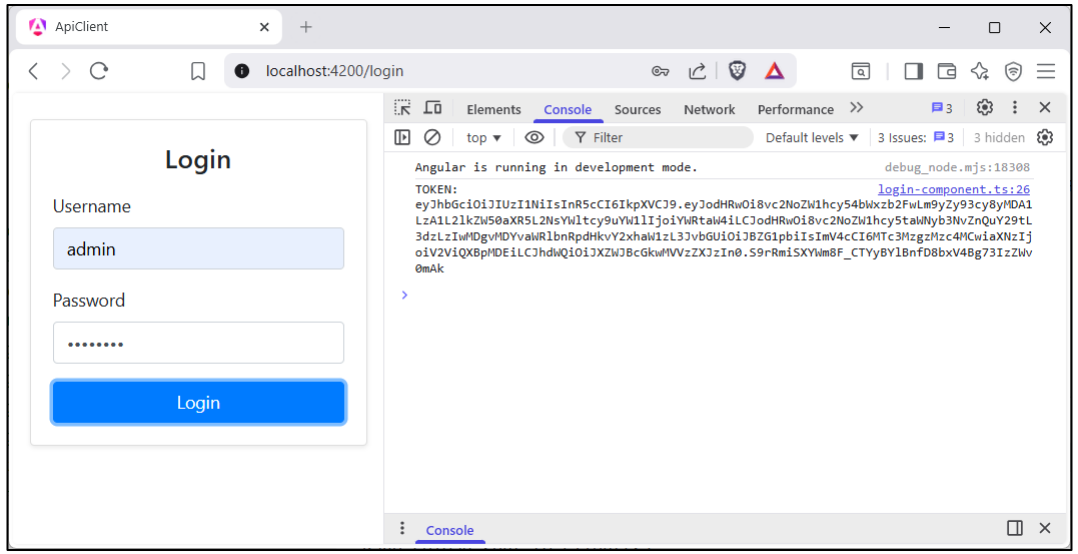
Definisanje rute za logovanje

app.routes.ts

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

Generisanje token





Čuvanje tokena u lokalnom skladištu browsera(AuthService klasa)

auth-service.ts

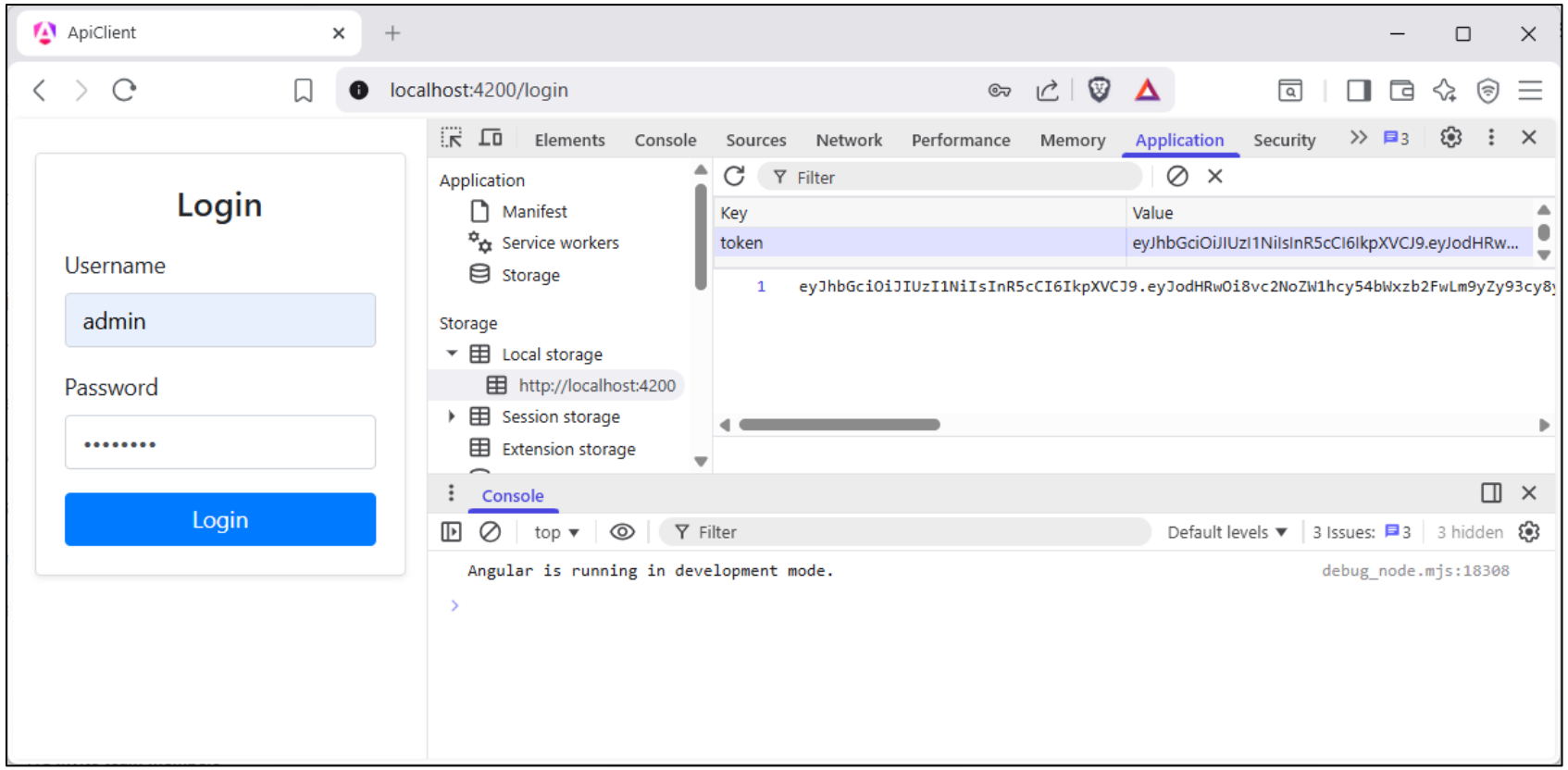
```
saveToken(token: string) {
    localStorage.setItem('token', token);
  }
```

- Lokalno skladište čuva token u browseru i ostaje dostupno i posle osvežavanja stranice
- localStorage čuva podatke kao par ključ–vrednost
- I ključ i vrednost u lokalnom skladištu su tipa string.
- Token se najčešće upisuje metodom localStorage.setItem('token', token).

Login komponenta

```
onSubmit() {
  const data = {
    username: this.username,
    password: this.password
  };

  this.authService.login(data).subscribe({
    next: (response) => {
      console.log('TOKEN:', response.token);
      this.authService.saveToken(response.token);
    },
    error: (err) => {
      console.log('GRESKA', err);
    }
  });
}
```

Prikaz lokalnog skladišta sa tokenom



Authorization header i Bearer token

- Authorization header predstavlja HTTP zaglavlje koje se koristi za slanje podataka za autentikaciju serveru
- HTTP header je dodatna informacija koja se šalje uz svaki zahtev
- Authorization header služi za identifikaciju korisnika
- **Bearer** je tip autentikacije kod koga se za pristup koristi token
- Token predstavlja jedinstveni identifikator korisnika
- Token se šalje u formatu: **Authorization: Bearer <token>**
- Server proverava token pre obrade zahteva
- Bez validnog tokena server vraća 401 Unauthorized

Dodavanje autorizacionog header-a u Angularu

- **Interceptor** je servis koji presreće HTTP zahteve i odgovore
- Nakon uspešnog login-a, server vraća token u odgovoru
- Token se čuva u localStorage za kasnije zahteve
- Interceptor čita token iz localStorageAko token postoji, automatski ga dodaje u Authorization header
- Tako zaštićeni zahtevi sadrže token bez ručnog dodavanja header-a u svakoj metodi

HTTP Interceptor u Angular aplikaciji

- HTTP **interceptor** predstavlja mehanizam u Angular framework-u koji omogućava presretanje i modifikaciju svih HTTP zahteva i odgovora na centralizovan način, pre nego što budu poslati serveru ili vraćeni aplikaciji
- Presreće sve HTTP zahteve i odgovore
- Centralizuje logiku HTTP komunikacije
- Dodaje Authorization (JWT -JSON Web Token) header
- Koristi se za obradu grešaka

Implementacija HTTP Interceptora

- Interceptor se generiše pomoću Angular CLI komande
- Implementira se kao funkcija (**HttpInterceptorFn**)
- Presreće svaki HTTP zahtev
- Proverava postojanje JWT tokena u localStorage
- Dodaje autorizacioni header ukoliko token postoji
- Prosleđuje modifikovan zahtev dalje u pipeline-u

Kreiranje interceptora

ng g interceptor interceptors/auth --skip-tests --flat

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

export const authInterceptor: HttpInterceptorFn = (req, next) => {
  const token = localStorage.getItem('token');

  if (!token) {
    return next(req);
  }

  const authReq = req.clone({
    setHeaders: {
      Authorization: `Bearer ${token}`
    }
  });
  return next(authReq);
};
```

Kreiranje interceptora

- **localStorage**.getItem('token') preuzima JWT token sa klijentske strane
- Uslov if (!token) omogućava prolaz zahteva bez autentikacije
- **HttpRequest** objekat je immutable i ne može se direktno menjati
- Metoda **clone**() se koristi za kreiranje nove instance zahteva
- **setHeaders** dodaje Authorization header u novi zahtev
- Format header-a mora biti **Bearer <token>** prema JWT standardu
- **next(authReq)** prosleđuje modifikovan zahtev sledećem handler-u u pipeline-u

Registracija HTTP Interceptora u Angular aplikaciji

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

import { routes } from './app.routes';
import { authInterceptor } from './interceptors/auth-interceptor';

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

Komponenta za dodavanje proizvoda

ng g component components/add-proizvod-component

```
import { Component } from '@angular/core';
import { Proizvod } from '../../models/proizvod';
import { FormsModule } from '@angular/forms';

@Component({
  selector: 'app-add-proizvod-component',
  imports: [FormsModule],
  templateUrl: './add-proizvod-component.html',
  styleUrl: './add-proizvod-component.css',
})
export class AddProizvodComponent {
proizvod: Proizvod = {
  proizvodId: 0,
  naziv: '',
  cena: 0,
  opis: ''
};
sacuvaj(): void {

}
}
```

Definisanje rute za dodavanje proizvoda

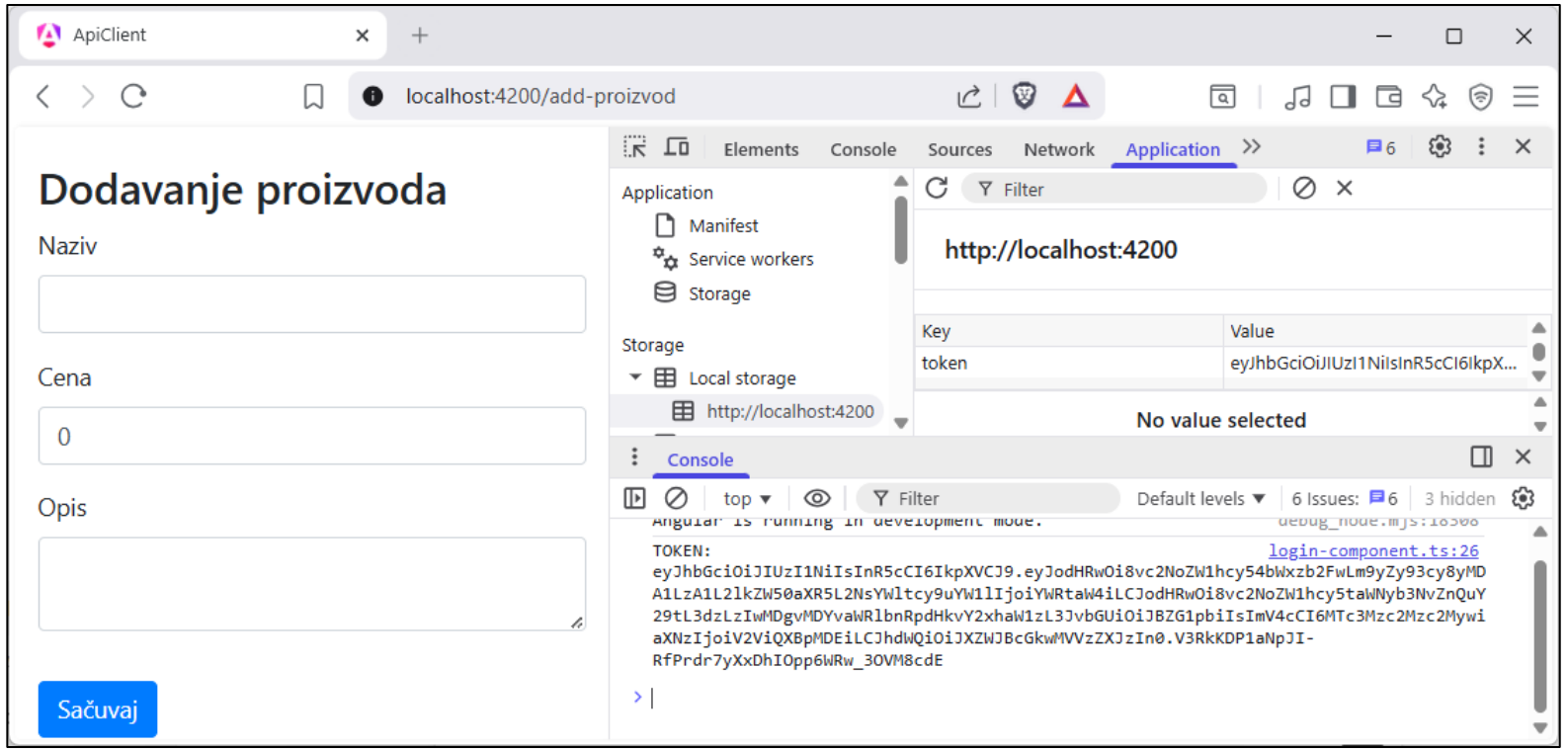
```
import { Routes } from '@angular/router';
import { AddProizvodComponent } from './components/add-proizvod-
component/add-proizvod-component';
import { LoginComponent } from './components/login-component/login-
component';
import { ProizvodiComponent } from './components/proizvodi-
component/proizvodi-component';


export const routes: Routes = [
  { path: '', component: ProizvodiComponent },
  { path: 'login', component: LoginComponent },
  { path: 'add-proizvod', component: AddProizvodComponent }
];
```

Komponenta za logovanje

```
export class LoginComponent {
  username: string = '';
  password: string = '';

 constructor(private authService: AuthService, private router: Router) {}

  onSubmit() {
  const data = {
    username: this.username,
    password: this.password
  };

  this.authService.login(data).subscribe({
    next: (response) => {
      console.log('TOKEN:', response.token);
      this.authService.saveToken(response.token);
       this.router.navigate(['/add-proizvod']);
    },
    error: (err) => {
      console.log('GRESKA', err);
    }
  });
}
```

Tek nakon uspešnog logovanja moguće je dodavati novi proizvod

Uspesno logovanje



Route Guard u Angular aplikaciji

- **Route guard** predstavlja mehanizam u Angularu koji omogućava kontrolu pristupa rutama na osnovu definisanih uslova pre nego što se ruta aktivira
- Koristi se za zaštitu ruta u aplikaciji
- Omogućava kontrolu navigacije između stranica
- Proverava uslove pre učitavanja komponente
- Često se koristi za autentikaciju korisnika
- Može sprečiti pristup ili izvršiti preusmeravanje
- Implementira se pomoću **CanActivate** interfejsa ili funkcionalnog pristupa

Kreiranje garda

ng g guard guards/auth --skip-tests

```
PS C:\Users\goran\source\repos\WebApi01\ApiClient> ng g guard guards/auth --sk
ip-tests
? Which type of guard would you like to create?
>◉ CanActivate
 ○ CanActivateChild
 ○ CanDeactivate
 ○ CanMatch
```

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


export const authGuard: CanActivateFn = (route, state) => {
  return true;
};
```

Route Guard

```
import { inject } from '@angular/core';
import { CanActivateFn, Router } from
'@angular/router';


export const authGuard: CanActivateFn = () => {
  const router = inject(Router);
  const token = localStorage.getItem('token');

  if (token) {
    return true;
  }

  return router.createUrlTree(['/login']);
};
```

Logika auth guarda

- Guard se implementira kao funkcija tipa **CanActivateFn**
- inject(Router) omogućava pristup **Router** servisu bez konstruktora
- localStorage.getItem('token') proverava da li korisnik ima JWT token
- Ako token postoji, ruta se dozvoljava (return true)
- Ako token ne postoji, korisnik se preusmerava na login stranu
- router.createUrlTree(['/login']) definiše cilj preusmeravanja
- **Guard** se izvršava pre aktivacije rute

ProizvodService dodavanje POST metode

proizvod-service.ts

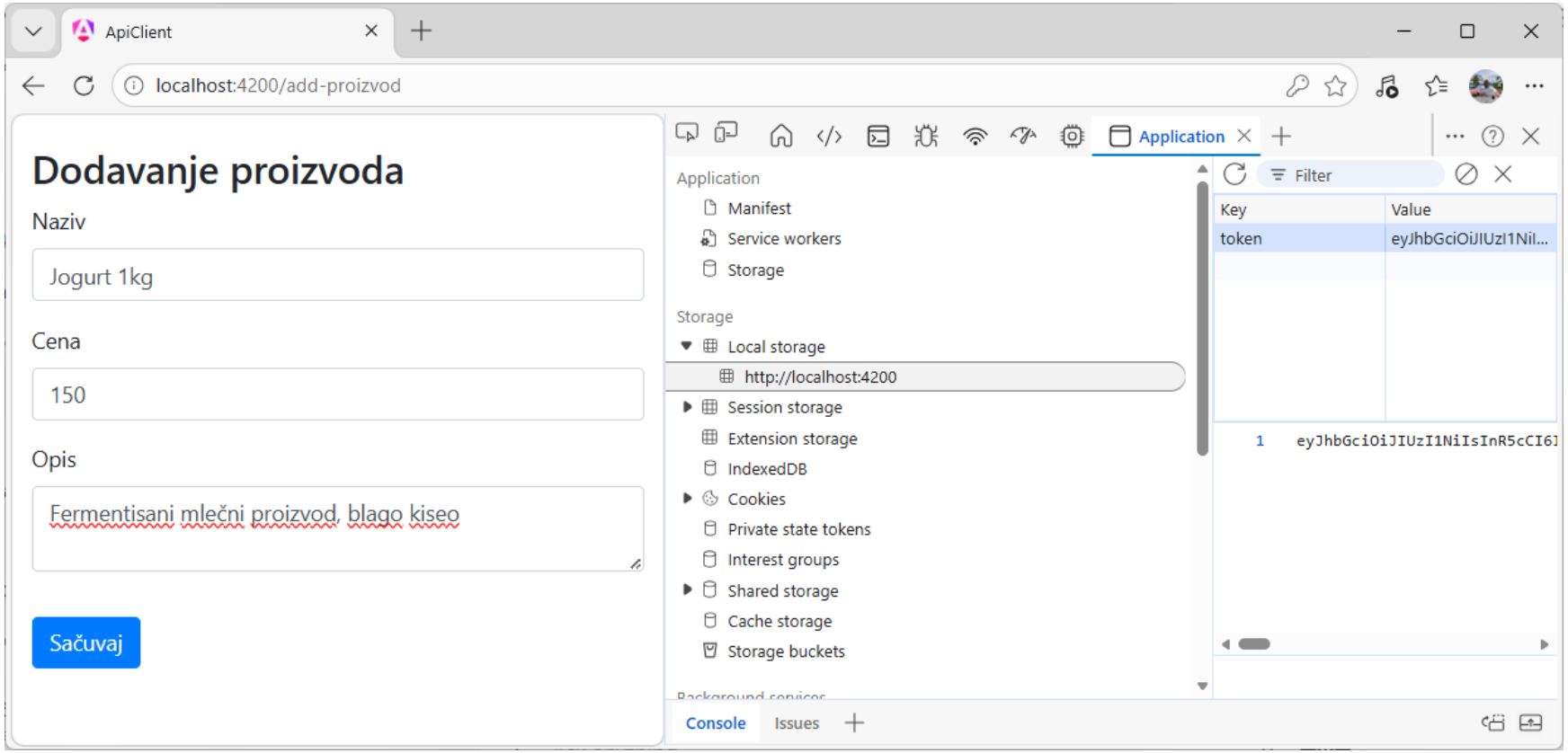
```
export class ProizvodService {

  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}`);
}
addProizvod(proizvod: Proizvod): Observable<Proizvod> {
  return this.http.post<Proizvod>(this.apiUrl, proizvod);
}
}
```

Komponenta add-proizvod

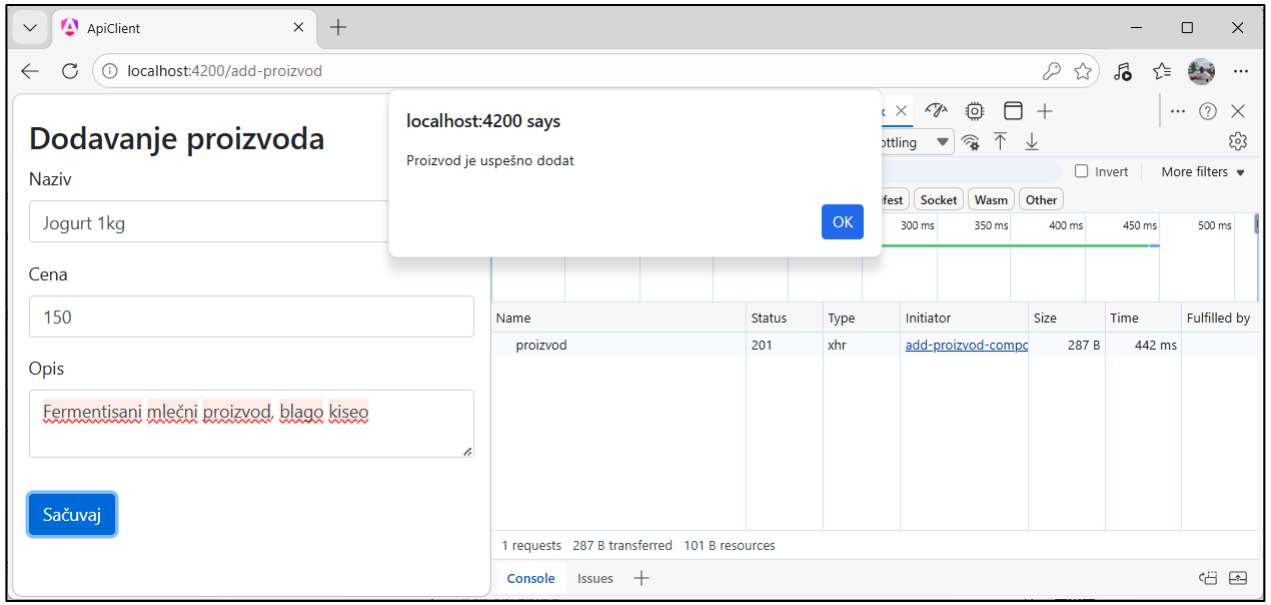
add-proizvod-component.ts

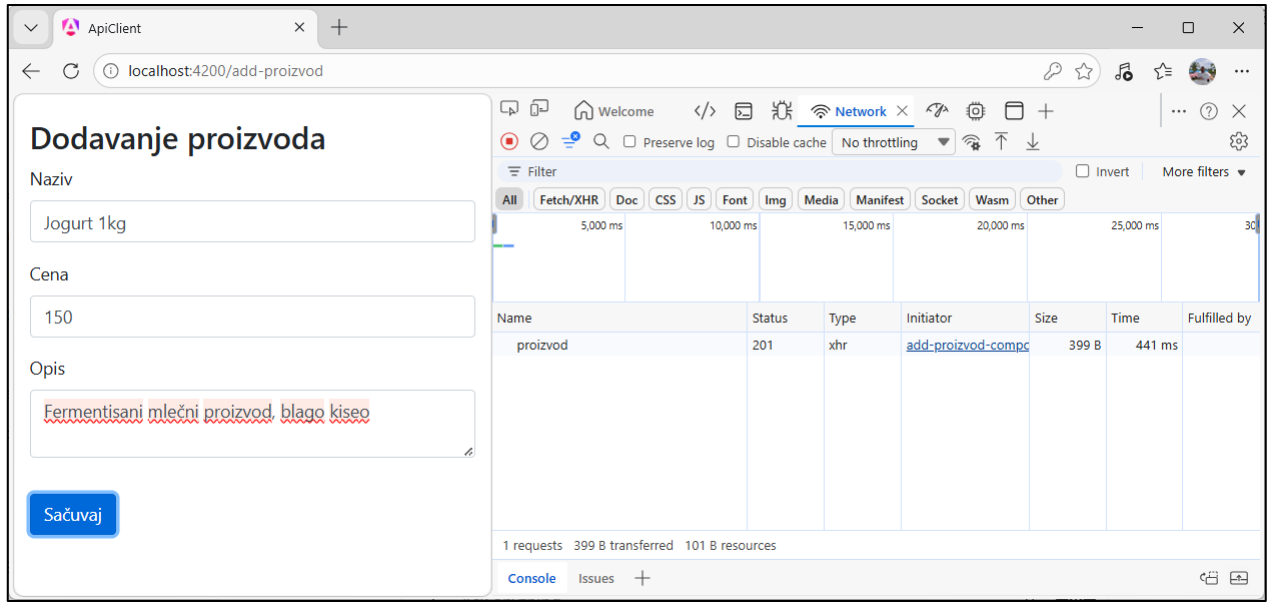
```
export class AddProizvodComponent {
proizvod: Proizvod = {
  proizvodId: 0,
  naziv: '',
  cena: 0,
  opis: ''
};

constructor(private proizvodService: ProizvodService) {}
sacuvaj(): void {
  this.proizvodService.addProizvod(this.proizvod).subscribe({
    next: (res) => {
      console.log('Dodat proizvod:', res);
      alert('Proizvod je uspešno dodat');
    },
    error: (err) => {
      console.error('Greška:', err);
      alert('Greška pri dodavanju proizvoda');
    }
  });
}
```

Unos podataka



Uspešan unos proizvoda





Uspešan unos proizvoda

- POST zahtev ka API-ju je uspešno izvršen
- Backend vraća status 201 (Created)
- JWT token je automatski dodat putem interceptora
- Podaci sa forme su uspešno poslati serveru
- Server je sačuvao novi proizvod u bazi
- Angular aplikacija dobija potvrdu o uspešnom unosu

JWT zaštita u Angular + ASP.NET aplikaciji

- JWT autentikacija omogućava sigurnu komunikaciju između klijenta i servera
- Token se generiše nakon uspešnog logina
- Angular čuva token u localStorage
- HTTP interceptor automatski dodaje token u svaki zahtev
- Route guard štiti pristup zaštićenim rutama
- ASP.NET backend validira token pre obrade zahteva
- Zaštićene operacije (POST, PUT, DELETE) zahtevaju autentikaciju

Pitanje 1

Šta je JWT token?

a. Bezbednosni ključ koga generise angular komponenta
b. Digitalni token koga generise angular servis
c. Digitalni token koji server izdaje korisniku nakon uspešnog logovanja

Odgovor: c

Pitanje 2

Čemu služi vrednost Key u JWT konfiguraciji?

a. Za određivanje aplikacije ili korisnika kojima je token namenjen
b. Za podešavanje vremena važenja tokena
c. Za potpisivanje i proveru integriteta tokena

Odgovor: c

Pitanje 3

Ko potpisuje JWT token prilikom njegovog generisanja??

a. Browser korisnika pomoću podataka iz login forme
b. Server pomoću tajnog ključa iz JWT konfiguracij
c. Interceptor pomoću autorizacionog header-a

Odgovor: b

Pitanje 4

Gde Angular aplikacija čuva JWT token nakon uspešnog logovanja?

a. u lokalnom skladištu browsera
b. u HTTP interceptoru
c. U appsettings.json fajlu

Odgovor: a

Pitanje 5

Šta je glavna uloga HTTP interceptora u aplikaciji koja koristi autentikaciju?

a. Da spreči izvršavanje svih GET zahteva u aplikaciji
b. Da centralizovano izmeni odlazne HTTP zahteve dodavanjem autorizacionog hedera
c. Da generiše serverski token

Odgovor: b

Pitanje 6

Čemu služi route guard u Angular aplikaciji?

a. Da proveri uslov pre aktivacije određene rute
b. Da promeni HTTP odgovor sa servera
c. Da automatski ubaci autorizacioni heder u HTTP zahtev

Odgovor: a

Pitanje 7

Šta označava izraz Bearer u autorizacionom header-u?

a. Deo tokena u kome se čuva vreme isteka i korisnička uloga
b. Tip autentikacije kod kog se uz zahtev šalje token koji server proverava
c. Da server ne treba da proverava autorizacioni heder

Odgovor: b

Pitanje 8

Koji podaci se najčešće navode u konfiguraciji za JWT token?

a. Tajni ključ, izdavalac tokena, kome je token namenjen i vreme važenja
b. Korisničko ime, lozinka, korisnička uloga i status prijave
c. Authorization header, Bearer prefiks, HTTP zahtev

Odgovor: a

Pitanje 9

Koja je uloga appSettings.json fajla u konfiguraciji JWT autentikacije?

a. U njemu se čuva generisani JWT token svakog prijavljenog korisnika
b. U njemu se čuvaju podešavanja potrebna za generisanje i validaciju JWT tokena
c. U njemu se definišu parametri HTML forme za unos korisničkog imena i lozinke

Odgovor: b

Pitanje 10

Zašto se zaštićene rute koriste u Angular aplikacijama?

a. Da se spreči pristup određenim delovima aplikacije neprijavljenim korisnicima
b. Da bi se svi HTTP zahtevi izvršavali bez provere korisnika
c. Da bi se izbegla upotreba Angular servisa

Odgovor: a

Pitanje 11

Kako se omogućava korišćenje HTTP interceptora u Angular aplikaciji?

a. Poziva se u svakoj komponenti pre slanja HTTP zahteva
b. Registruje se u konfiguraciji aplikacije kroz podešavanje HTTP klijenta sa interceptorima
c. Čuva se u localStorage zajedno sa JWT tokenom

Odgovor: b

Pitanje 12

Koja tvrdnja najbolje opisuje konfiguraciju JWT autentikacije u Program.cs fajlu?

a. AddAuthentication() određuje šemu autentikacije, a AddJwtBearer() definiše parametre za proveru tokena
b. AddAuthentication() generiše token, a AddJwtBearer() ga čuva u lokalnom skladištu browsera
c. AddJwtBearer() čuva token u browseru a AddAuthentication() dodaje autorizacioni heder

Odgovor: a