# Minimal API v VS Code

1. Izdelaš nov projekt (v terminal-u)

dotnet new web -o TodoApi

cd TodoApi

code -r ../TodoApi

1. Napisano dobiš že kodo v Program.cs

var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

app.MapGet("/", () => "Hello World!");

app.Run();

1. Dodaš še v terminalu

dotnet dev-certs https –trust

1. Poženeš aplikacijo
2. Dodamo pakete spet v terminalu

dotnet add package Microsoft.EntityFrameworkCore.InMemory

dotnet add package Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore

1. Dodamo razred

public class Todo

{

public int Id { get; set; }

public string? Name { get; set; }

public bool IsComplete { get; set; }

}

1. In razred

using Microsoft.EntityFrameworkCore;

class TodoDb : DbContext

{

public TodoDb(DbContextOptions<TodoDb> options)

: base(options) { }

public DbSet<Todo> Todos => Set<Todo>();

}

1. Koda za Program.cs

using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<TodoDb>(opt => opt.UseInMemoryDatabase("TodoList"));

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

var app = builder.Build();

app.MapGet("/todoitems", async (TodoDb db) =>

await db.Todos.ToListAsync());

app.MapGet("/todoitems/complete", async (TodoDb db) =>

await db.Todos.Where(t => t.IsComplete).ToListAsync());

app.MapGet("/todoitems/{id}", async (int id, TodoDb db) =>

await db.Todos.FindAsync(id)

is Todo todo

? Results.Ok(todo)

: Results.NotFound());

app.MapPost("/todoitems", async (Todo todo, TodoDb db) =>

{

db.Todos.Add(todo);

await db.SaveChangesAsync();

return Results.Created($"/todoitems/{todo.Id}", todo);

});

app.MapPut("/todoitems/{id}", async (int id, Todo inputTodo, TodoDb db) =>

{

var todo = await db.Todos.FindAsync(id);

if (todo is null) return Results.NotFound();

todo.Name = inputTodo.Name;

todo.IsComplete = inputTodo.IsComplete;

await db.SaveChangesAsync();

return Results.NoContent();

});

app.MapDelete("/todoitems/{id}", async (int id, TodoDb db) =>

{

if (await db.Todos.FindAsync(id) is Todo todo)

{

db.Todos.Remove(todo);

await db.SaveChangesAsync();

return Results.NoContent();

}

return Results.NotFound();

});

app.Run();

1. Dodamo še kodo za uporabo swaggerja (ni nujno, lahko testiraš s postmanom)

dotnet add package NSwag.AspNetCore

using NSwag.AspNetCore;

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddOpenApiDocument(config =>

{

config.DocumentName = "TodoAPI";

config.Title = "TodoAPI v1";

config.Version = "v1";

});

Kodo dodamo pred var app = builder.Build();

Nato pa še kodo

if (app.Environment.IsDevelopment())

{

app.UseOpenApi();

app.UseSwaggerUi(config =>

{

config.DocumentTitle = "TodoAPI";

config.Path = "/swagger";

config.DocumentPath = "/swagger/{documentName}/swagger.json";

config.DocExpansion = "list";

});

}

Aplikacijo poženemo, vtipkamo naslov localhost:xxxx/swagger in nato testiramo metodo POST (try it out). Vnesemo podatke v obliki

{

"name":"walk dog",

"isComplete":true

}

In izberemo gumb Execute. Testiramo še vse ostale API metode.

1. Dodamo lahko tudi grupiranje, (MapGroup in uporabimo metode namesto lambda izrazov 🡪 glej v ToDoApi
2. Uporaba skrivanja določenih podatkov (preprečevanje over postinga)

Narediš nov razred, ki služi samo prenašanju po omrežju. Na primer, če imaš ToDo

public class Todo

{

public int Id { get; set; }

public string? Name { get; set; }

public bool IsComplete { get; set; }

public string? Secret { get; set; }

}

Dodaš

public class TodoItemDTO

{

public int Id { get; set; }

public string? Name { get; set; }

public bool IsComplete { get; set; }

public TodoItemDTO() { }

public TodoItemDTO(Todo todoItem) =>

(Id, Name, IsComplete) = (todoItem.Id, todoItem.Name, todoItem.IsComplete);

}

V tem primeru Secret ne prenaša okoli

1. Delo z bazo SQLLite:
   1. dotnet add package Microsoft.EntityFrameworkCore.Sqlite --version 8.0
   2. dotnet tool install --global dotnet-ef
   3. dotnet add package Microsoft.EntityFrameworkCore.Design --version 8.0
   4. Takoj na začetku dodaj kodo var connectionString = builder.Configuration.GetConnectionString("Pizzas") ?? "Data Source=Pizzas.db";
   5. builder.Services.AddSqlite<PizzaDb>(connectionString);
   6. Poženi projekt bo napaka
   7. dotnet ef migrations add InitialCreate
   8. dotnet ef database update

# Avtentikacija z JWT-ji

Recimo, da imamo spodnje podatke

{

"alg": "HS256",

"typ": "JWT"

}

{

"sub": "1234567890",

"name": "John Doe",

"iat": 1516239022

}

Te podatke bi radi kodirane poslali po omrežju. V ta namen uporabimo JWT, ki nam podatke kodira v obliki

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.

eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.C0gpNoQg2qEZSU\_51jGdXC43Fxmn1mvNqWTAi56E6j4

V kodiranih podatkih imamo 3 dele:

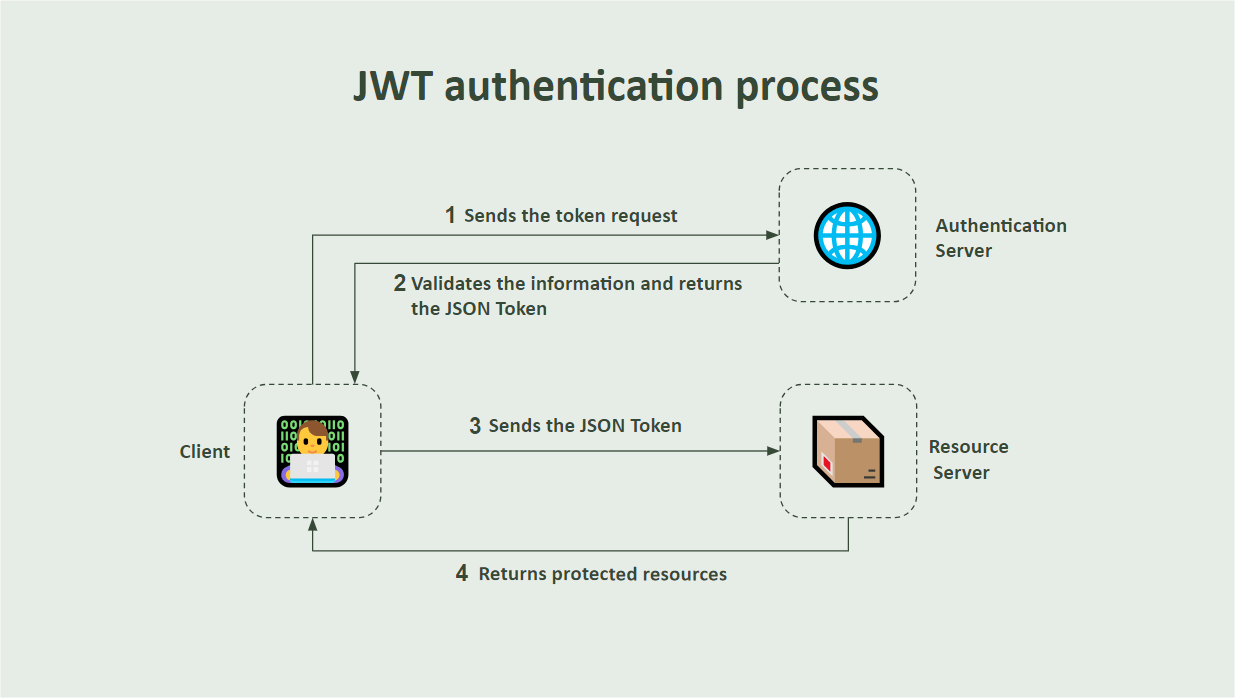
Glava = header, ki je del do prve . V tem delu so podatki o tipu žetona (pri nas JWT) in o kodiranju (HMAC SHA256 ali RSA)

Telo =payload (koristni tovor), kjer imamo kodirane poljubne podatke

Podpis=Signature Če želite ustvariti podpisni del, morate vzeti kodirano glavo, kodirano koristno vsebino, skrivnost in algoritem, določen v glavi, ter ga podpisati.

Preverite <https://jwt.io/>

Kako poteka avtentikacija s pomoćjo JWT-ja



dotnet new web -o AuthCore

cd AuthCore

dotnet add package Swashbuckle.AspNetCore

dotnet add package Microsoft.AspNetCore. Authentication.JwtBearer

Dodamo skrivnost, ki bo del našega JWT-ja (navadno je nimamo v kodi, ampak nekje skrito)

namespace AuthCore.Helpers{

public static class AuthSettings

{

public static string PrivateKey { get; set; } = "MIICWwIBAAKBgHZO8IQouqjDyY47ZDGdw9jPDVHadgfT1kP3igz5xamdVaYPHaN24UZMeSXjW9sWZzwFVbhOAGrjR0MM6APrlvv5mpy67S/K4q4D7Dvf6QySKFzwMZ99Qk10fK8tLoUlHG3qfk9+85LhL/Rnmd9FD7nz8+cYXFmz5LIaLEQATdyNAgMBAAECgYA9ng2Md34IKbiPGIWthcKb5/LC/+nbV8xPp9xBt9Dn7ybNjy/blC3uJCQwxIJxz/BChXDIxe9XvDnARTeN2yTOKrV6mUfI+VmON5gTD5hMGtWmxEsmTfu3JL0LjDe8Rfdu46w5qjX5jyDwU0ygJPqXJPRmHOQW0WN8oLIaDBxIQQJBAN66qMS2GtcgTqECjnZuuP+qrTKL4JzG+yLLNoyWJbMlF0/HatsmrFq/CkYwA806OTmCkUSm9x6mpX1wHKi4jbECQQCH+yVb67gdghmoNhc5vLgnm/efNnhU";

}

}

Ustvarimo uporabnika

namespace AuthCore.Models{

public class User

{

public Guid Id {get;set;}

public string Name{get;set;}

public string Email{get;set;}

public string Password{get;set;}

public string[] Roles{get;set;}

}

}

Nato izdelamo servis

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using AuthCore.Helpers;

using AuthCore.Models;

using Microsoft.IdentityModel.Tokens;

namespace AuthCore.Services;

public class AuthService

{

public string GenerateToken(User user) //naša metoda

{

var handler = new JwtSecurityTokenHandler();

//razred J wtSecurityTokenHandler je za generiranje,

//preverjanje in upravljanje z žetoni

//razred ima CreateToken() metodo in WriteToken(token) metodo

var key = Encoding.ASCII.GetBytes(AuthSettings.PrivateKey);

//poišče ključ in ga spremeni v zaporedje bytov

var credentials = new SigningCredentials(

new SymmetricSecurityKey(key),

SecurityAlgorithms.HmacSha256Signature);

//ustveri objekt za podpis po metodi, ki smo jo določili

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = GenerateClaims(user),

Expires = DateTime.UtcNow.AddMinutes(15),

SigningCredentials = credentials,

};

//opis đetona vsebuje podatke potrebne za ustvarjanje žetona

var token = handler.CreateToken(tokenDescriptor);

return handler.WriteToken(token);

}

private static ClaimsIdentity GenerateClaims(User user) //generira objekt ClainsIdentity za določenega userja

{

var claims = new ClaimsIdentity();

claims.AddClaim(new Claim(ClaimTypes.Name, user.Email));

foreach (var role in user.Roles)

claims.AddClaim(new Claim(ClaimTypes.Role, role));

return claims;

}

}

Koda za testiranje

using AuthCore.Models;

using AuthCore.Services;

//using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddTransient<AuthService>();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddOpenApiDocument(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "AuthCore API", Version = "v1" });

});

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI(c =>

{

c.SwaggerEndpoint("/swagger/v1/swagger.json", "AuthCore API V1");

});

}

app.MapPost("/authenticate", (User user, AuthService authService)

=> authService.GenerateToken(user));

app.Run();

Testiramo, da v naslovni vrstici dodamo localhost:xxxxx/swagger in preizkusimo metodo post na podatkih

{

"email": "johnsmith@samplemail.com",

"roles": [

"admin”]

}

A screenshot of a computer

Description automatically generated

Će prenesemo rezulta na levo stran jwt.io (Encoded), dobimo za rezultat kaj smo vnesli.

Do tukaj smo izdelali žeton. Dodati moramo še kodo, ki ta žeton uporabi za avtentikacijo in avtorizacijo.

Dodamo kar manjka:

....

# Zapisovanje uporabnikov v bazo in uporaba JWT avtentikacije

dotnet add package Microsoft.EntityFrameworkCore.Sqlite

dotnet add package Microsoft.EntityFrameworkCore.Design

dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer

dotnet add package Microsoft.IdentityModel.Tokens

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using System.ComponentModel.DataAnnotations;

public class User

{

public int Id { get; set; }

[Required]

public string Username { get; set; }

[Required]

public string PasswordHash { get; set; }

}

public class ApplicationDbContext : DbContext

{

public DbSet<User> Users { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlite("Data Source=users.db");

}

}

dotnet ef migrations add InitialCreate

dotnet ef database update

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<ApplicationDbContext>();

builder.Services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = "yourissuer",

ValidAudience = "youraudience",

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("yoursecretkey"))

};

});

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

app.UseAuthentication();

app.UseAuthorization();

app.MapPost("/register", async (User user, ApplicationDbContext db, IPasswordHasher<User> passwordHasher) =>

{

user.PasswordHash = passwordHasher.HashPassword(user, user.PasswordHash); // Note: PasswordHash is misused here, you should separate plain text password input

db.Users.Add(user);

await db.SaveChangesAsync();

return Results.Ok();

});

app.MapPost("/login", async (User user, ApplicationDbContext db, IPasswordHasher<User> passwordHasher) =>

{

var existingUser = await db.Users.FirstOrDefaultAsync(u => u.Username == user.Username);

if (existingUser != null && passwordHasher.VerifyHashedPassword(existingUser, existingUser.PasswordHash, user.PasswordHash) == PasswordVerificationResult.Success)

{

var tokenHandler = new System.IdentityModel.Tokens.Jwt.JwtSecurityTokenHandler();

var key = Encoding.ASCII.GetBytes("yoursecretkey");

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(new Claim[]

{

new Claim(ClaimTypes.Name, user.Username)

}),

Expires = DateTime.UtcNow.AddDays(7),

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)

};

var token = tokenHandler.CreateToken(tokenDescriptor);

return Results.Ok(new { Token = tokenHandler.WriteToken(token) });

}

return Results.Unauthorized();

});

app.Run();

{ "username": "yourusername", "passwordHash": "yourpassword" // Note: This should be a plain text password in practice }

# Še druga izvedba

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = "yourissuer",

ValidAudience = "youraudience",

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("yoursecretkey"))

};

});

builder.Services.AddAuthorization();

builder.Services.AddSingleton<IPasswordHasher<User>, PasswordHasher<User>>();

builder.Services.AddScoped<UserService>();

var app = builder.Build();

// Configure the HTTP request pipeline.

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.MapPost("/register", async (UserDto userDto, UserService userService) =>

{

var user = new User

{

Username = userDto.Username,

PasswordHash = userService.HashPassword(userDto.Password)

};

userService.CreateUser(user);

return Results.Ok();

});

app.MapPost("/login", async (LoginDto loginDto, UserService userService) =>

{

var user = userService.GetUserByUsername(loginDto.Username);

if (user != null && userService.VerifyPassword(loginDto.Password, user.PasswordHash))

{

var token = userService.GenerateJwtToken(user);

return Results.Ok(new { Token = token });

}

return Results.Unauthorized();

});

app.Run();

public record UserDto(string Username, string Password);

public record LoginDto(string Username, string Password);

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.IdentityModel.Tokens;

using System;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using System.Threading.Tasks;

public class UserService

{

private readonly ApplicationDbContext \_context;

private readonly IPasswordHasher<User> \_passwordHasher;

private readonly IConfiguration \_configuration;

public UserService(ApplicationDbContext context, IPasswordHasher<User> passwordHasher, IConfiguration configuration)

{

\_context = context;

\_passwordHasher = passwordHasher;

\_configuration = configuration;

}

public void CreateUser(User user)

{

\_context.Users.Add(user);

\_context.SaveChanges();

}

public User GetUserByUsername(string username)

{

return \_context.Users.SingleOrDefault(u => u.Username == username);

}

public string HashPassword(string password)

{

return \_passwordHasher.HashPassword(null, password);

}

public bool VerifyPassword(string password, string hashedPassword)

{

return \_passwordHasher.VerifyHashedPassword(null, hashedPassword, password) == PasswordVerificationResult.Success;

}

public string GenerateJwtToken(User user)

{

var tokenHandler = new JwtSecurityTokenHandler();

var key = Encoding.ASCII.GetBytes(\_configuration["Jwt:Key"]);

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(new Claim[]

{

new Claim(ClaimTypes.Name, user.Username)

}),

Expires = DateTime.UtcNow.AddHours(1),

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)

};

var token = tokenHandler.CreateToken(tokenDescriptor);

return tokenHandler.WriteToken(token);

}

}

using Microsoft.EntityFrameworkCore;

public class ApplicationDbContext : DbContext

{

public DbSet<User> Users { get; set; }

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)

{

}

}

public class User

{

public int Id { get; set; }

public string Username { get; set; }

public string PasswordHash { get; set; }

}

# Net Maui

Napake, poskusi iz Visual studio

Vsak nov projekt?

dotnet build -t:InstallAndroidDependencies -f:net8.0-android -p:AndroidSdkDirectory="/Users/barbarap/Library/Developer/Xamarin/android-sdk-macosx/cmdline-tools/7.0/bin

" -p:JavaSdkDirectory="/Library/Java/JavaVirtualMachines/microsoft-11.jdk/Contents/Home

" -p:AcceptAndroidSDKLicenses=True

V VS Code moraš to napisati

/Users/barbarap/Library/Developer/Xamarin/android-sdk-macosx