**การยืนยันตัวตน (Authentication)**

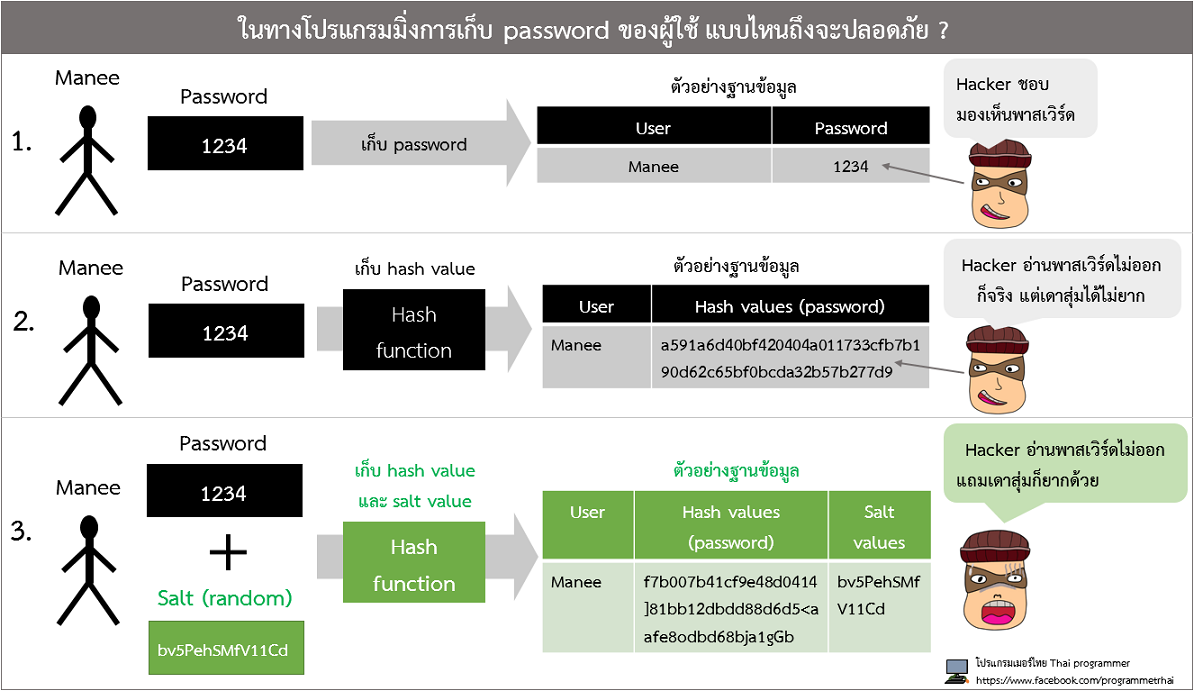
**แบบไม่ใช้ Identiy Framework VS มี Database**

Github : <https://github.com/tee2490/JwtAuthenWithDb>

Youtube : <https://www.youtube.com/watch?v=6sMPvucWNRE&t=7s>

[**https://www.youtube.com/watch?v=6sMPvucWNRE&t=7s**](https://www.youtube.com/watch?v=6sMPvucWNRE&t=7s)

**Hash Password** เข้ารหัสผ่านด้วยเทคนิค Hash+Salt เพื่อให้ค่าแฮชไม่ซ้ำกัน

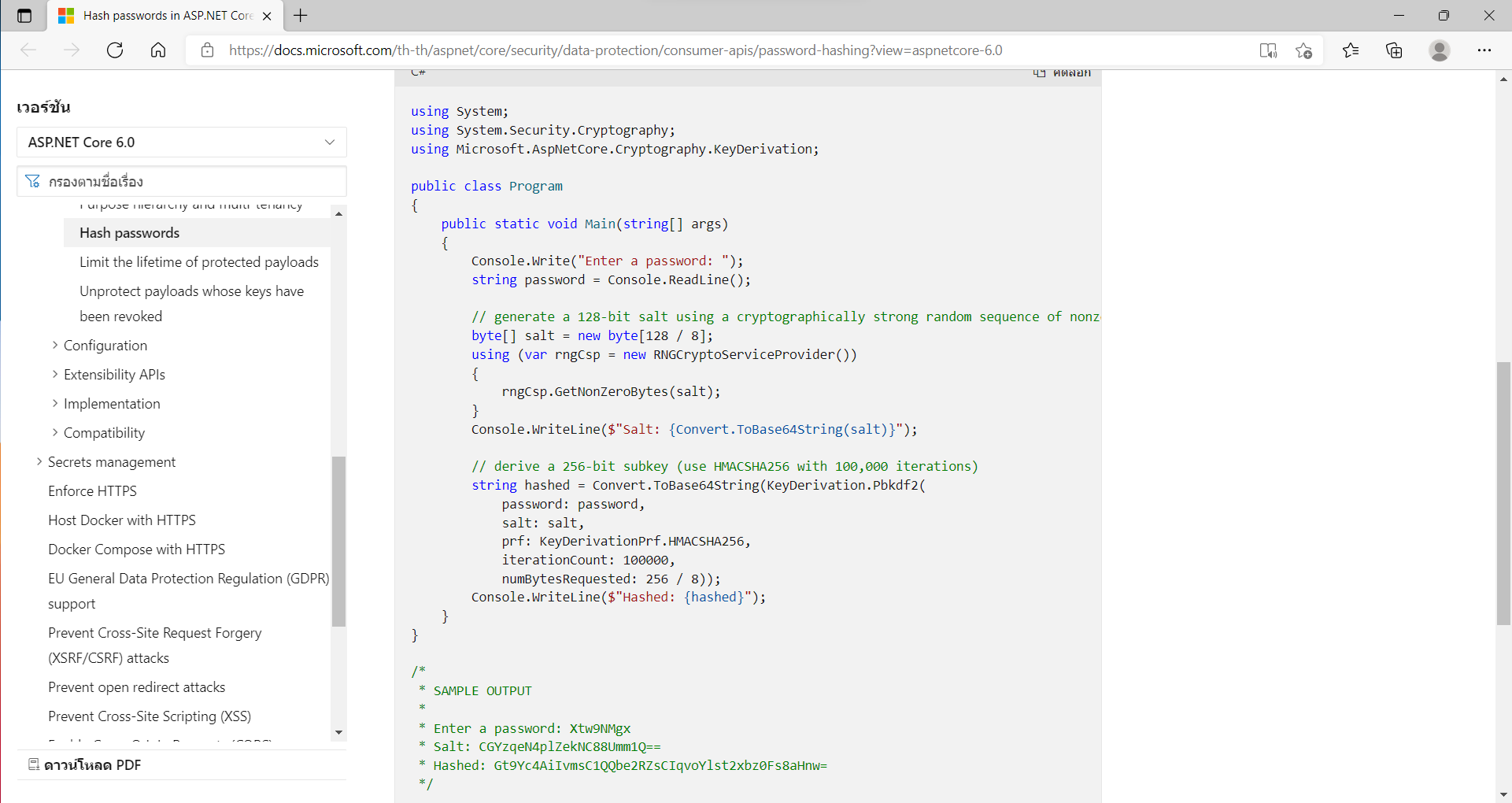
<https://www.patanasongsivilai.com/blog/password-hash-function-salt-values/>

โค้ดการ hash+salt

ค้น hash password.net core ไปดูหลักการ hash ใน help

แบบ 1 <https://docs.microsoft.com/th-th/aspnet/core/security/data-protection/consumer-apis/password-hashing?view=aspnetcore-6.0>

แบบ 2 อันนี้ง่าย Bcrypt <https://jasonwatmore.com/post/2020/07/16/aspnet-core-3-hash-and-verify-passwords-with-bcrypt>

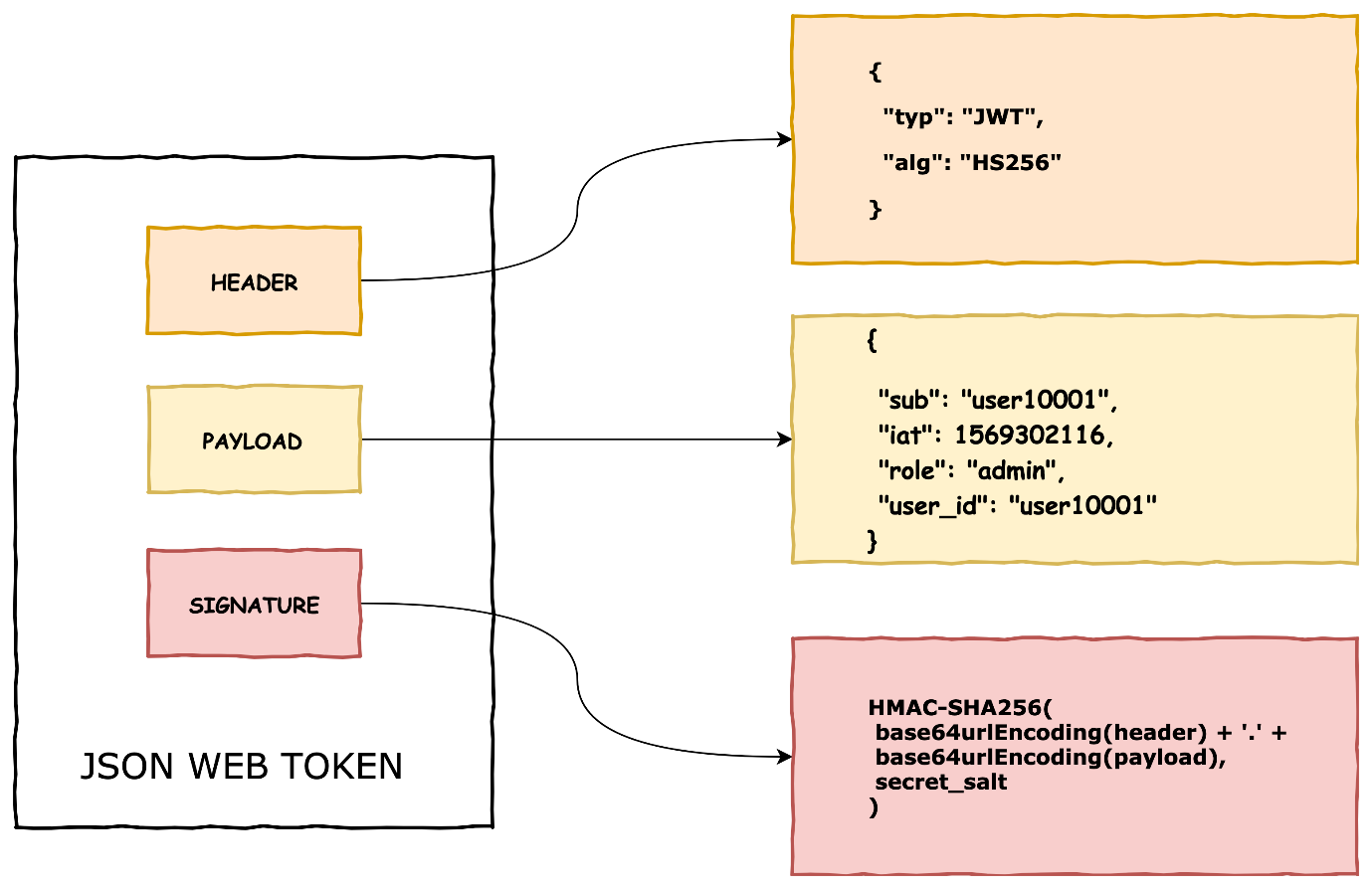


**Token**

Token เป็นรหัสชุดหนึ่งเพื่อระบุตัวตนว่าเป็นใครซึ่งเทียบเท่ากับลายเซ็นต์ เมื่อเข้ารหัสแล้วไม่สามารถเปลี่ยนแปลงแก้ไขได้

JWT (Json Web Token)

JWT เป็นรูปแบบหนึ่งที่ใช้ในการสร้างรหัส token จากข้อมูล JSON Data แล้วทำการเข้ารหัสด้วย Base64Url และถือว่าเป็นมาตรฐานเปิด(RFC 7519) ที่เข้ามาแก้ปัญหาการส่งข้อมูลอย่างปลอดภัยระหว่างกัน โดยถูกออกแบบไว้ว่า จะต้องมีขนาดที่กะทัดรัด (Compact) และเก็บข้อมูลภายในตัว(Self-contained)

ส่วนประกอบ JWT

ประกอบด้วย 3 ส่วน ดังนี้

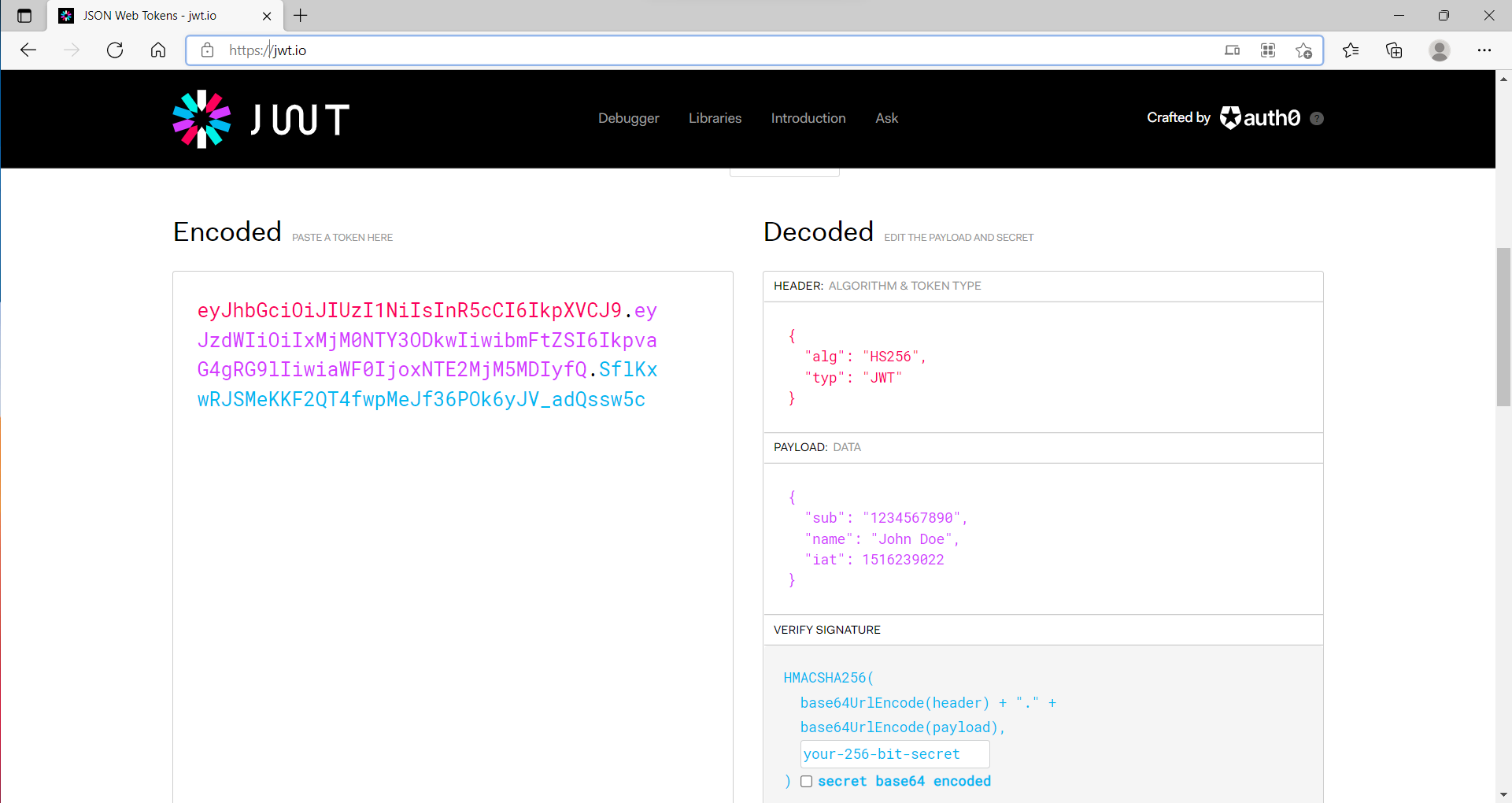
1.Header คือข้อมูล metadata ของ token ซึ่งบอกว่า เป็น type และใช้ algorithm อะไร

2.Body หรือ Payload หรือ Claims คือ ข้อมูลทั้งหมดที่เอาไว้ sign token

3.Signature : ส่วนสำคัญของข้อมูล เป็นการรวมกันของ Header และ Body ใช้ algorithm และ secret key ในการ sign

อ่านเพิ่มเติม <https://devahoy.com/blog/2016/07/understanding-token-and-jwt-create-authentication-with-hapijs/>

<https://jwt.io> เว็ปทดสอบ JWT



ขั้นตอนการใช้ Token

1. Login เข้าระบบไปยังฐานข้อมูล
2. สร้าง Token
3. ใช้ Token โดยไม่ต้องไปดึงข้อมูลมาจากฐานข้อมูลอีก

**การสร้างฐานข้อมูลแบบ Code First**

รูปแบบคำสั่งการสร้างตารางในฐานข้อมูล ef migration

<https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli>

Add a migration

dotnet ef migrations add InitialCreate

Create/Upate database and schema (latest migration)

dotnet ef database update

OR The following updates your database to a given migration:

dotnet ef database update AddNewTables

\*สีแดงคือชื่อ Migration ที่ต้องการกำหนด หรือจัดการ

ตัวอย่าง

การสร้างตารางในฐานข้อมูล

dotnet ef migrations add InitialCreate เริ่มสร้างกำหนดชื่อ migration ว่า InitialCreate

dotnet ef migrations remove ลบการ migration ตัวล่าสุดออก

dotnet ef database update การอัพเดทฐานข้อมูลด้วยตัว migration ล่าสุด

การสร้างตารางโดยระบุชื่อ context

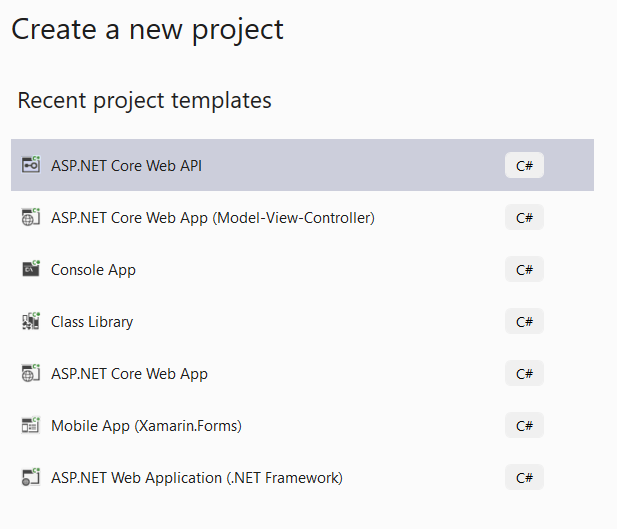
dotnet ef migrations add CreateIdentitySchema --context ชื่อContext

dotnet ef database update --context ชื่อContext

dotnet ef migrations remove -c IdentityContext

\*หมายเหตุ การระบุ --context กรณีมี dbcontext มากกว่า 1 ตัว

สร้างโปรเจค Web API เพื่อให้ง่ายต่อการทดสอบ ชื่อ JwtAuthen



ติดตั้ง Nugets : Entity Framework (แพคเกตสำหรับสร้างฐานข้อมูล)

Microsoft.EntityFrameworkCore

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Design

ติดตั้ง Nugets สำหรับเข้ารหัส(HashPassword)

BCrypt.Net-Next

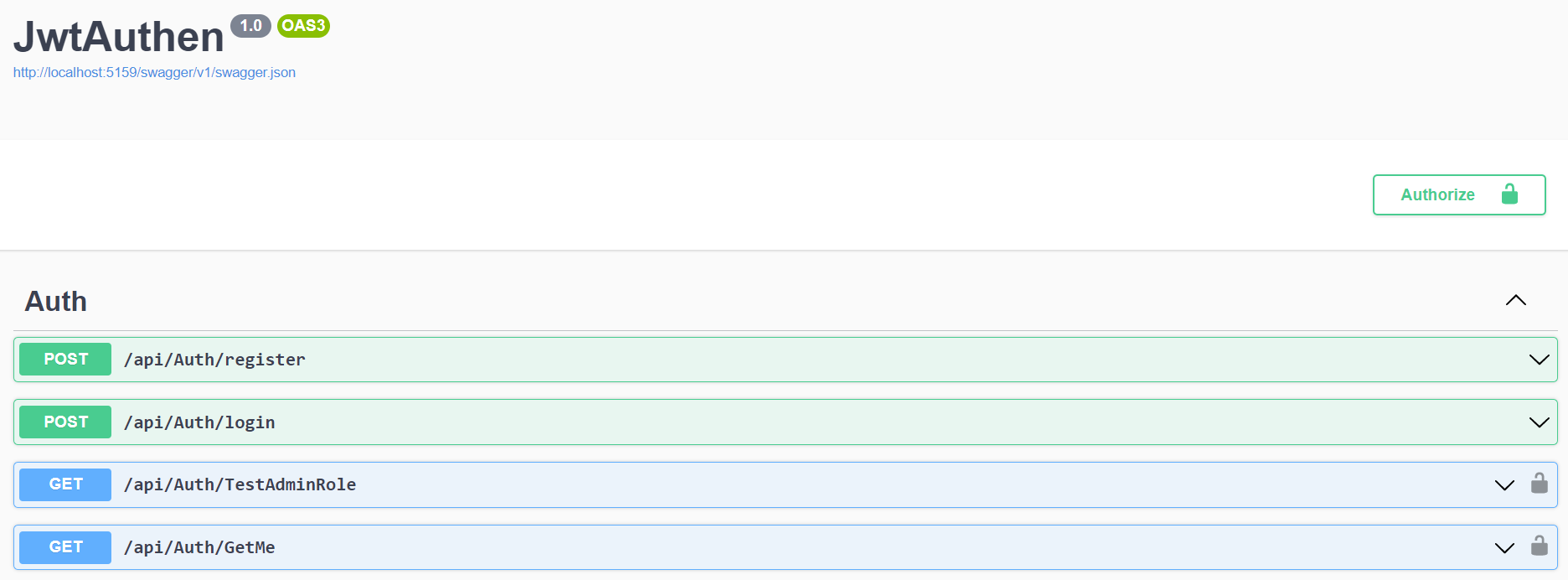
ติดตั้ง Nugets สำหรับจัดการ Token (authenticationScheme)

Microsoft.AspNetCore.Authentication.JwtBearer

ติดตั้ง Nugets สำหรับจัดการ Swaggle (Authorize)

Swashbuckle.AspNetCore.Filters

ผลลัพธ์ที่ต้องการ Register, Login , TestAdminRole, GetMe



Register ลงทะเบียนผู้ใช้ใหม่

Login เข้าสู่ระบบ และ ได้รับ Token

TestAdminRole ยืนยันตัวตนด้วย Role

GetMe ยืนยันตัวตนทั่วไป

หมายเหตุ

การใส่ Token ผ่าน Swaggle ให้ทำดังนี้

Bearer <ตามด้วย Token>

การ Logout

ไม่สามารถลบ Token ฝั่ง Backend

การ Logout ต้องจัดการที่ฝั่ง Frontend

สร้าง Models

Models\User.cs

public class User

{

public int Id { get; set; }

public string Username { get; set; } = string.Empty;

public string PasswordHash { get; set; } = string.Empty;

public int RoleId { get; set; }

[JsonIgnore]

public Role Role { get; set; }

}

Models\Role.cs

public class Role

{

public int Id { get; set; }

public string Name { get; set; }

}

Models\RegisterDto.cs

public class RegisterDto

{

public required string Username { get; set; }

public required string Password { get; set; }

public required int RoleId { get; set; }

}

Models\LoginDto.cs

public class LoginDto

{

public required string Username { get; set; }

public required string Password { get; set; }

}

Data\DataContext.cs

public class DataContext : DbContext

{

public DataContext(DbContextOptions options) : base(options) { }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

base.OnConfiguring(optionsBuilder);

optionsBuilder.UseSqlServer("Server=.\\SqlExpress; Database=TestXXX; Trusted\_Connection=True; TrustServerCertificate=True");

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Entity<Role>().HasData(

new Role() {Id=1, Name = "Admin" },

new Role() {Id=2, Name = "User" });

}

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

public DbSet<Role> Roles { get; set; }

}

Program.cs (DI)

builder.Services.AddDbContext<DataContext>();

สร้าง Database ด้วย Dotnet Entity Framework

* rebuild โปรเจคให้เรียบร้อยก่อน
* dotnet ef migrations add InitialCreateDB -o Data/Migrations ระบุโฟลเดอร์จัดเก็บ/optional
* dotnet ef database update ยืนยันการสร้างฐานข้อมูล

appsettings.json

{

"AppSettings": {

"Token": "my top secret key"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

สร้างโฟลเดอร์ Services สำหรับการ Authen

Services\IAuthenService.cs

public interface IAuthenService

{

Task<User> Register(RegisterDto request);

Task<String> Login(LoginDto request);

Object GetMe();

}

Services\AuthenService.cs

public class AuthenService : IAuthenService

{

private readonly DataContext dataContext;

private readonly IConfiguration configuration;

private readonly IHttpContextAccessor httpContextAccessor;

public AuthenService(DataContext dataContext, IConfiguration configuration, IHttpContextAccessor httpContextAccessor)

{

this.dataContext = dataContext;

this.configuration = configuration;

this.httpContextAccessor = httpContextAccessor;

}

public async Task<User> Register(RegisterDto request)

{

var user = await dataContext.Users

.FirstOrDefaultAsync(x => x.Username == request.Username);

if (user != null) { return null; }

string passwordHash

= BCrypt.Net.BCrypt.HashPassword(request.Password);

var User = new User()

{

Username = request.Username,

PasswordHash = passwordHash,

RoleId = request.RoleId,

};

await dataContext.Users.AddAsync(User);

await dataContext.SaveChangesAsync();

return User;

}

public async Task<string> Login(LoginDto request)

{

var user = await dataContext.Users.Include(x => x.Role)

.FirstOrDefaultAsync(x => x.Username == request.Username);

if (user == null) return "User not found.";

if (!BCrypt.Net.BCrypt.Verify(request.Password, user.PasswordHash))

{

return "Wrong password.";

}

string token = CreateToken(user);

return token;

}

private string CreateToken(User user)

{

List<Claim> claims = new List<Claim> {

new Claim(ClaimTypes.Name, user.Username),

new Claim(ClaimTypes.Role, user.Role.Name),

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(

configuration.GetSection("AppSettings:Token").Value!));

var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha512Signature);

var token = new JwtSecurityToken(

claims: claims,

expires: DateTime.Now.AddDays(1),

signingCredentials: creds

);

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

return jwt;

}

public Object GetMe()

{

var username = string.Empty;

var role = string.Empty;

if (httpContextAccessor.HttpContext != null)

{

username = httpContextAccessor.HttpContext.User.FindFirstValue(ClaimTypes.Name);

role = httpContextAccessor.HttpContext.User.FindFirstValue(ClaimTypes.Role);

}

return new { username, role };

}

}

Program.cs (DI)

builder.Services.AddSwaggerGen(options =>

{

options.AddSecurityDefinition("oauth2", new OpenApiSecurityScheme

{

In = ParameterLocation.Header,

Name = "Authorization",

Type = SecuritySchemeType.ApiKey

});

options.OperationFilter<SecurityRequirementsOperationFilter>();

});

builder.Services.AddAuthentication().AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuerSigningKey = true,

ValidateAudience = false,

ValidateIssuer = false,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(

builder.Configuration.GetSection("AppSettings:Token").Value!))

};

});

builder.Services.AddScoped<IAuthenService,AuthenService>();

builder.Services.AddHttpContextAccessor();

AuthController.cs

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace JwtAuthen.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class AuthController : ControllerBase

{

private readonly IAuthenService authenService;

public AuthController(IAuthenService authenService)

{

this.authenService = authenService;

}

[HttpPost("register")]

public async Task<IActionResult> Register(RegisterDto request)

{

var user = await authenService.Register(request);

if (user == null) return BadRequest("User already exists");

return Ok(user);

}

[HttpPost("login")]

public async Task<IActionResult> Login(LoginDto request)

{

var result = await authenService.Login(request);

return Ok(result);

}

[HttpGet("TestAdminRole"), Authorize(Roles = "Admin")]

public IActionResult test()

{

return Ok("Authorize Success");

}

[HttpGet("GetMeByContext"), Authorize]

public IActionResult GetMe()

{

var result = authenService.GetMe();

return Ok(result);

}

[HttpGet("GetMeInBaseController"), Authorize]

public IActionResult GetMyName()

{

//ใช้ภายใต้ ControllerBase แสดงการใช้หลายวิธี

//var userName = User?.Identity?.Name;

var userName = User.FindFirstValue(ClaimTypes.Name);

var roles = User.FindFirstValue(ClaimTypes.Role);

//กรณีมีหลาย Role

//var roleClaims = User.FindAll(ClaimTypes.Role);

//var roles = roleClaims.Select(x => x.Value).ToList();

return Ok(new { userName, roles });

}

}

}