Best Practice “Create Web API: RESTFUL API”

Dalam Membuat Web API dapat menggunakan dotnet 5.0. Berikut Aplikasi yang diperlukan dalam serangkaian pembuatan web API menggunakan dotnet 5.0:

1. .NET 5.0:

* .Net SDK 5.0.406
* ASP.NET Core Runtime 5.0.15
* .NET Desktop Runtime 5.0.15
* .NET Runtime 5.0.15

(<https://dotnet.microsoft.com/en-us/download/dotnet/5.0>)

1. Visual Studio Code: Aplikasi IDE/editor kode selama pembuatan web API

(<https://code.visualstudio.com/docs/?dv=win>)

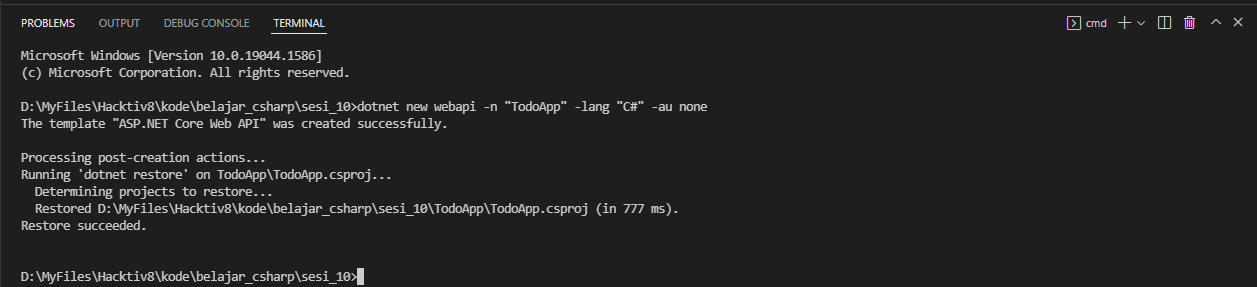
1. Insomnia: aplikasi untuk mencoba API

(<https://insomnia.rest/download>)

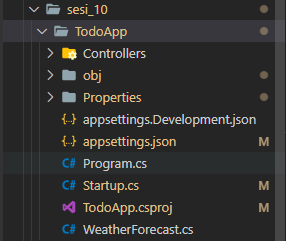
1. Navicat: aplikasi untuk membuka database termasuk SQLite
2. Membuat dan Menjalankan ‘Web API TodoApp’ dari dotnet 5.0
3. Membuat Web API dapat dilakukan dengan menjalankan syntax pada Folder Project

Syntax:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Syntax | dotnet new webapi | -n “TodoApp” | -lang “C#” | -au none |
| Keterangan | (1) | (2) | (3) | (4) |
| Keterangan:   1. Membuat project baru dotnet dengan template webapi 2. Penamaan project 3. Penggunaan bahasa C# 4. Authorization = none | | | | |

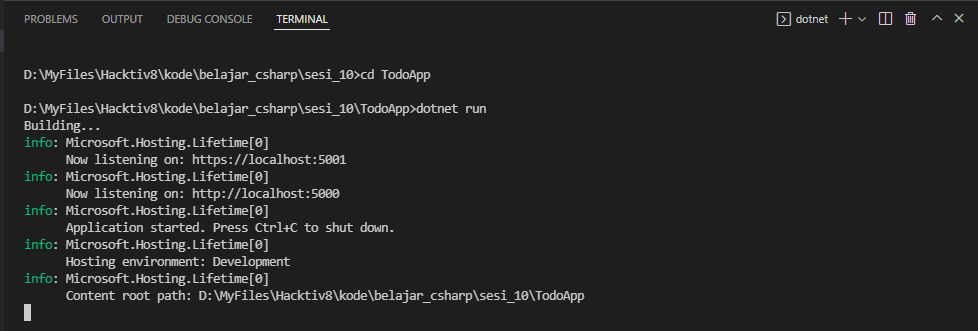


Dengan menjalankan syntax di atas akan mengenerate file sebagai berikut:



Setelah itu masuk ke dalam folder project TodoApp (1) dan mencoba menjalankannya (2)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Syntax 1 | cd TodoApp |  |  |  |
| Syntax 2 | dotnet run |  |  |  |



Berdasarkan Hasil running syntax, aplikasi berjalan pada host ‘http://localhost:5001’ Berikut hasilnya:

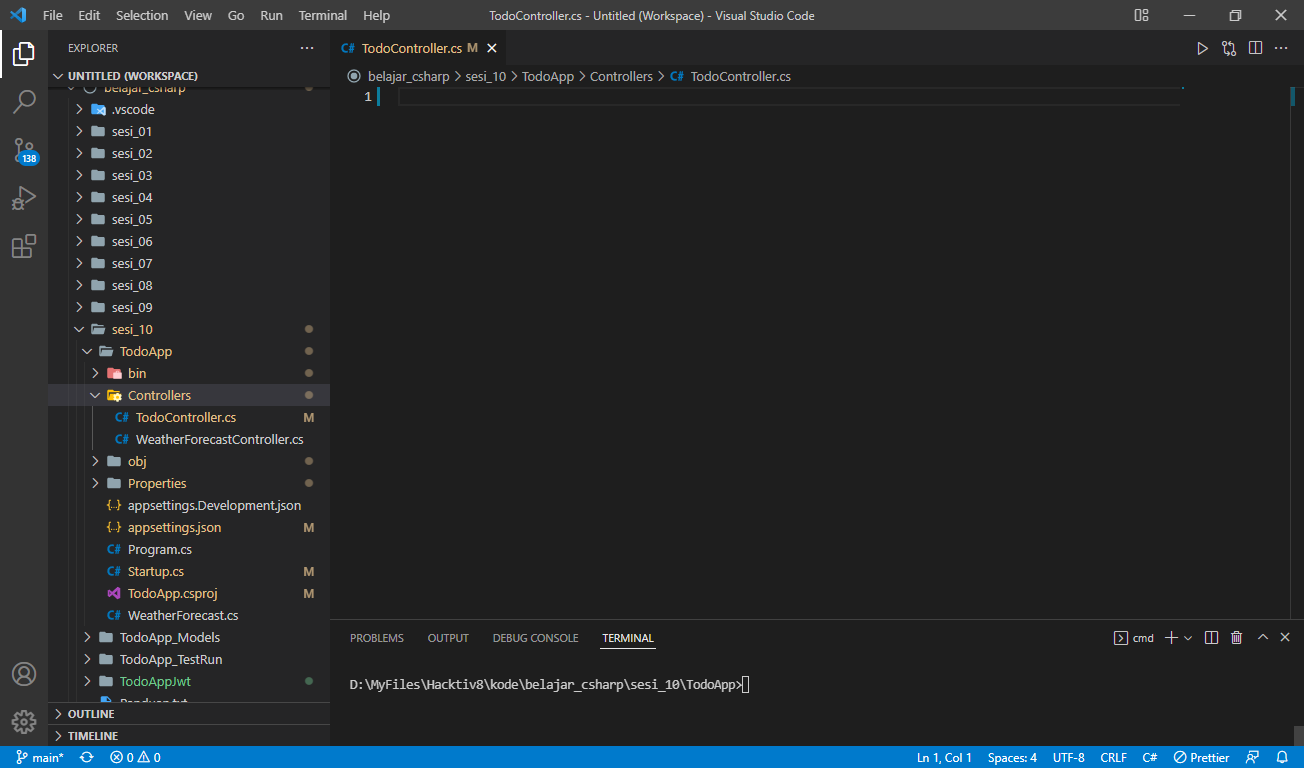


Untuk menghentikannya dapat menekan ‘Ctrl+C’. Dengan ini Project web API TodoApp sudah siap untuk dilakukan tahap selanjutnya.

1. Menambahkan Controller Todo pada Project TodoApp sebagai TestRun

Dalam folder project TodoApp sudah terdapat folder controller maka dari itu dapat langsung menambahkan file controller pada folder ini

1. Pada ‘TodoApp>Controllers’, tambahkan ‘TodoController.cs’.



1. Lalu isi dengan kode seperti dibawah ini

using Microsoft.AspNetCore.Mvc;

namespace TodoApp.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

    public class TodoController: ControllerBase

    {

        [Route("TestRun")]

        [HttpGet]

        public ActionResult TestRun(){

            return Ok("success");

        }

    }

}

Penjelasan Library:

|  |  |
| --- | --- |
| Library | using Microsoft.AspNetCore.Mvc; |
| Kode | [Route("api/[controller]")]  [ApiController]  [Route("TestRun")]  [HttpGet]  Ok("success"); |

Berdasarkan

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

Maka rute berada pada <http://localhost/api/todo> dimana todo ini merupakan nama controller berdasarkan ‘TodoController.cs’

Sedangkan setelah itu terdapat lagi fungsi Route

[Route("TestRun")]

Akan mendefinisikan rute pada <http://localhost/api/todo/testrun>

[HttpGet]

public ActionResult TestRun(){

return Ok("success");

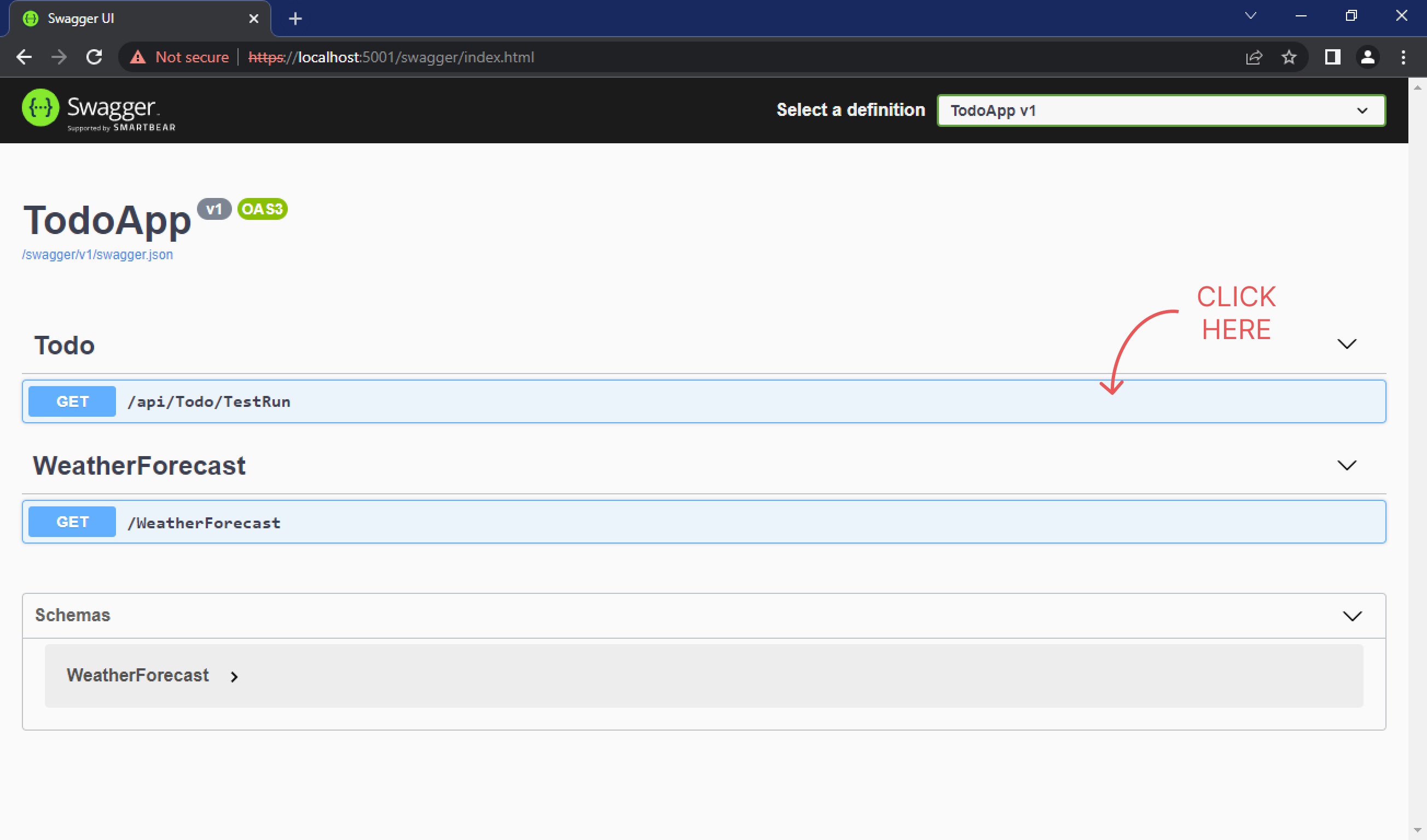
}

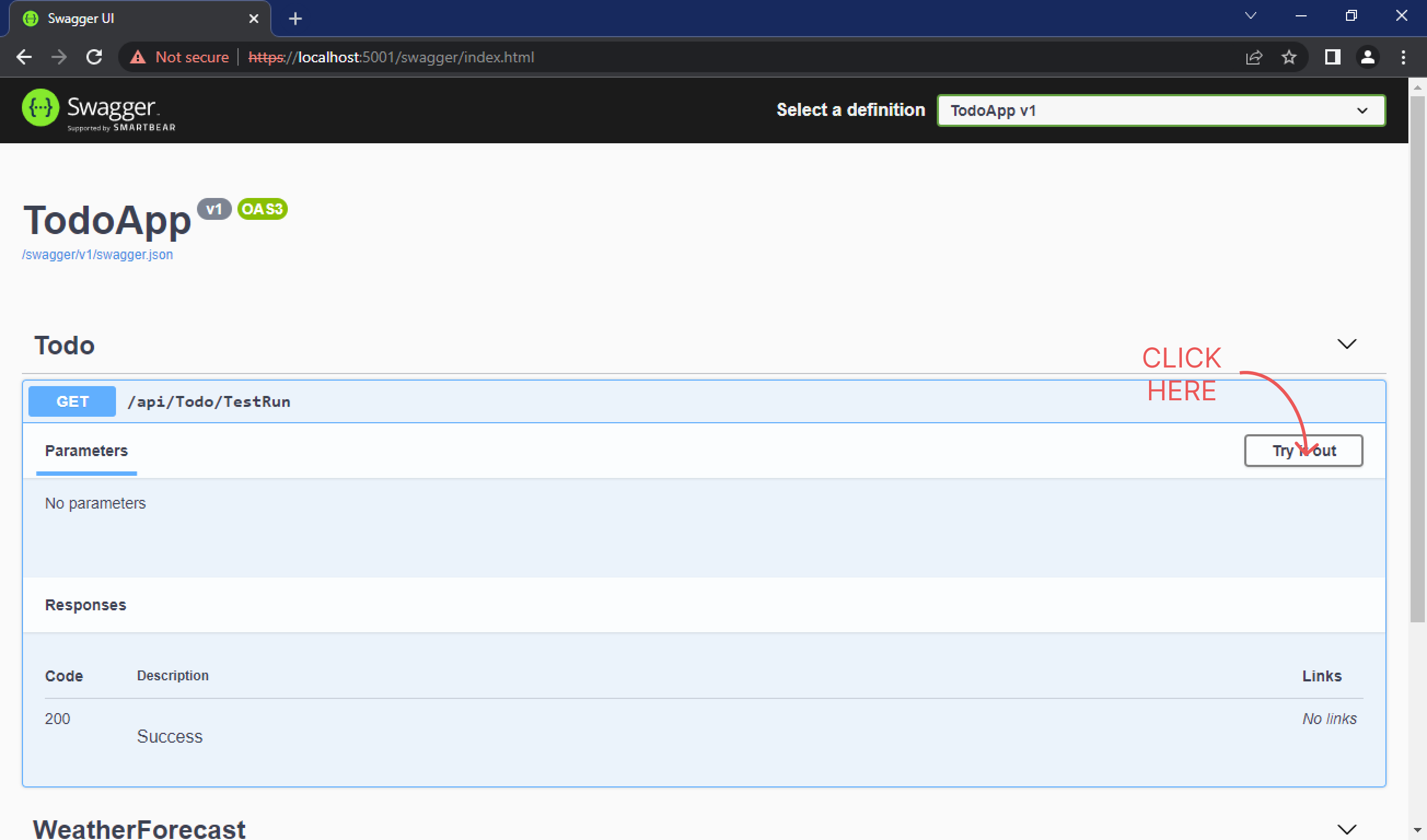
TestRun() akan berjalan pada <http://localhost/api/todo/testrun> dengan method ‘GET’

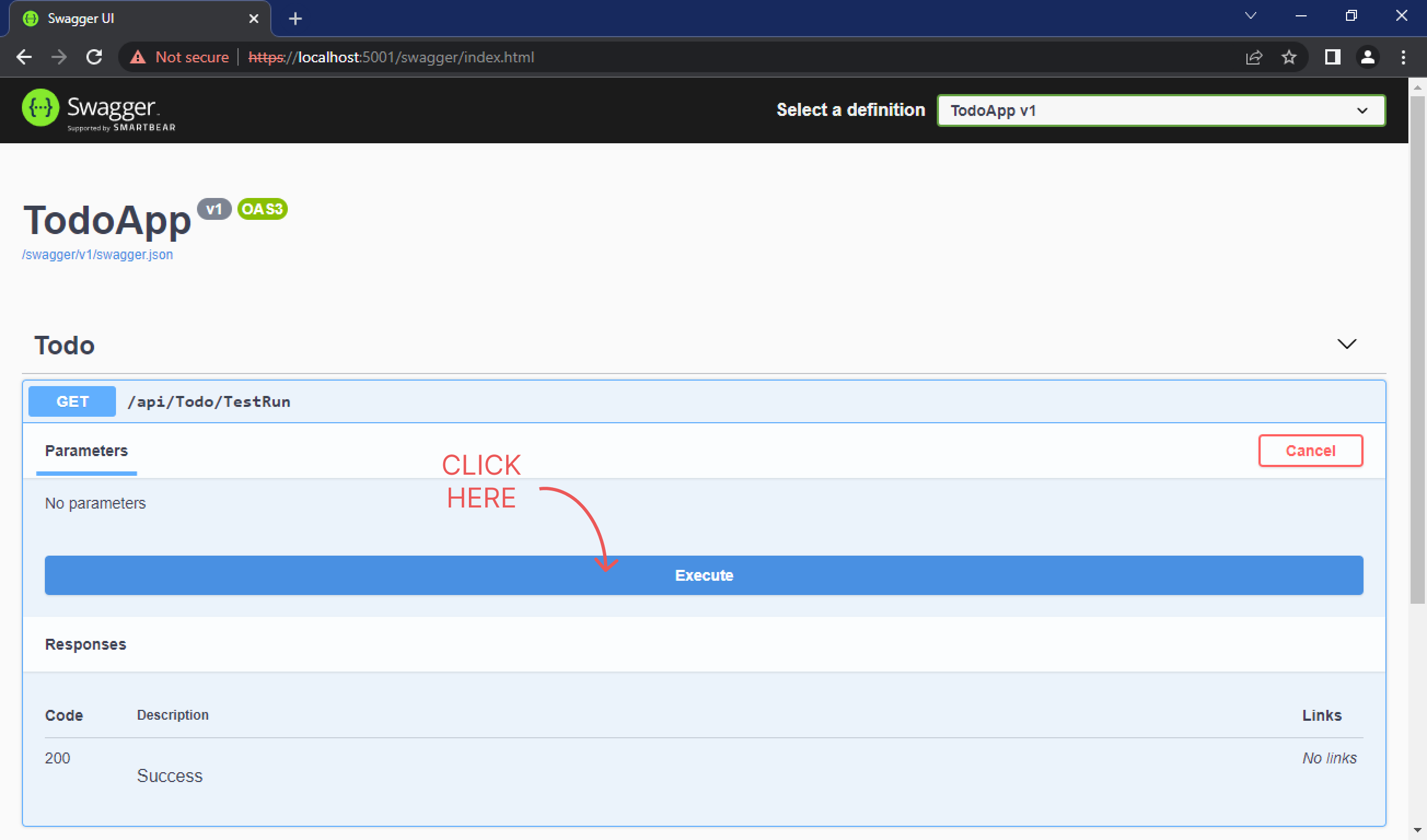
return Ok("success");

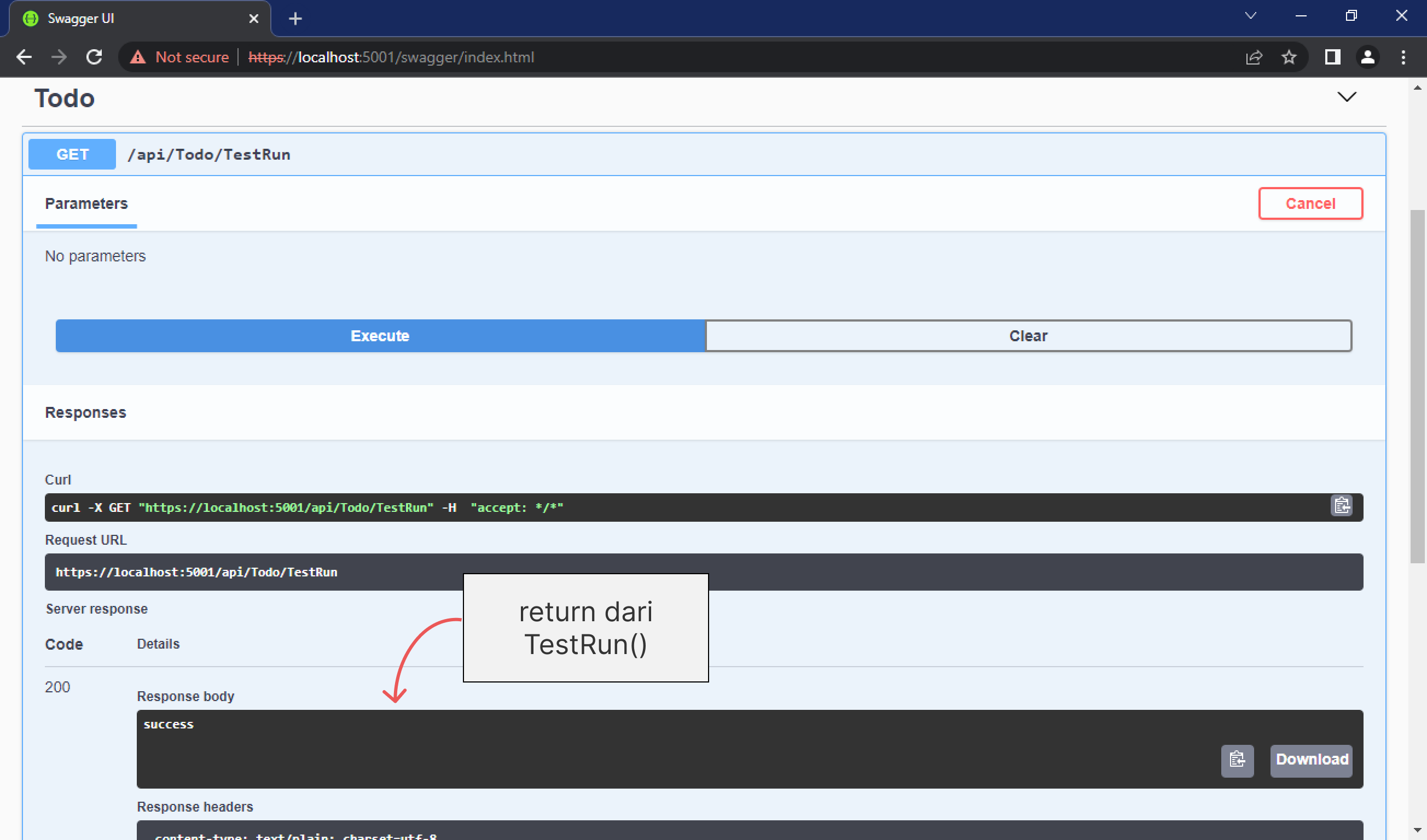
merupakan Method dari [ApiController]

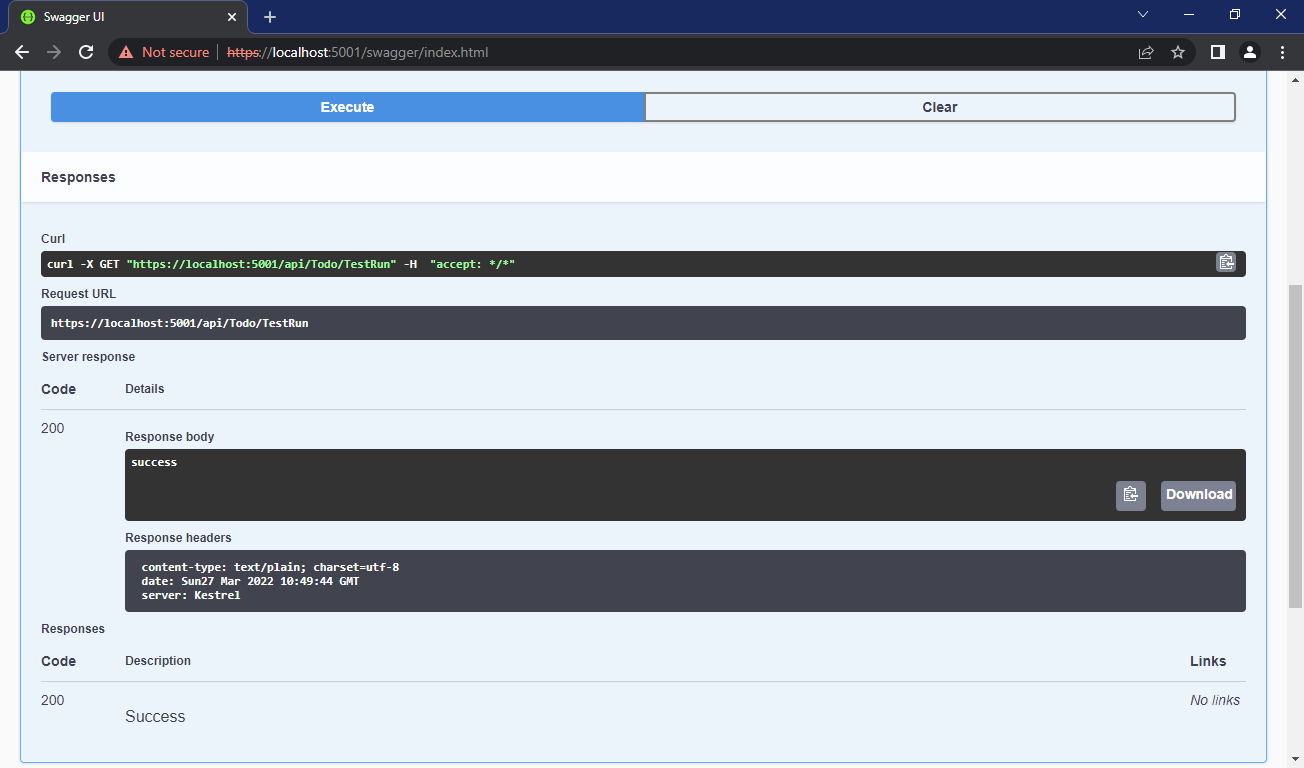
Sebagai hasil running :











Dengan ini TodoController berfungsi dengan baik dan siap untuk tahap selanjutnya.

1. Migrasi dan Update Database SQLite ‘Items’ sebagai database ‘Web API TodoApp’

Pada TodoApp ini akan menggunakan SQLite sebagai database. Di dalam database akan ada tabel Item yang berisi data:

Id INT

Title STRING

Description STRING

Done BOOL

Dikarenakan menggunakan SQLite dan migrations maka diperlukan package tambahan:

dotnet add package Microsoft.EntityFrameworkCore.Sqlite -v 5.0.15

dotnet add package Microsoft.EntityFrameworkCore.Tools -v 5.0.15

1. Pada ‘TodoApp’ tambahkan folder **‘Models’**.
2. Pada ‘TodoApp>Models’ tambahkan file **‘ItemData.cs’**, di dalam file ini definisikan

class ItemData

namespace TodoApp.Models

{

    public class ItemData

    {

        public int Id {get; set;}

        public string Title {get; set;}

        public string Description {get; set;}

        public bool Done {get; set;}

    }

}

Langkah ini diperlukan untuk mendefinisikan class ItemData yang nantinya akan dimanfaatkan sebagai object data untuk Item

1. Pada ‘TodoApp’ tambahkan folder ‘Data’
2. Pada ‘TodoApp>Data’ tambahkan file ‘ApiDBContext.cs’

using Microsoft.EntityFrameworkCore;

using TodoApp.Models;

namespace TodoApp.Data

{

    public class ApiDbContext : DbContext

    {

        public virtual DbSet<ItemData> Items {get;set;}

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

        {

        }

    }

}

* DbContext diambil dari Microsoft.EntityFrameworkCore
* ItemData diambil dari TodoApp.Models

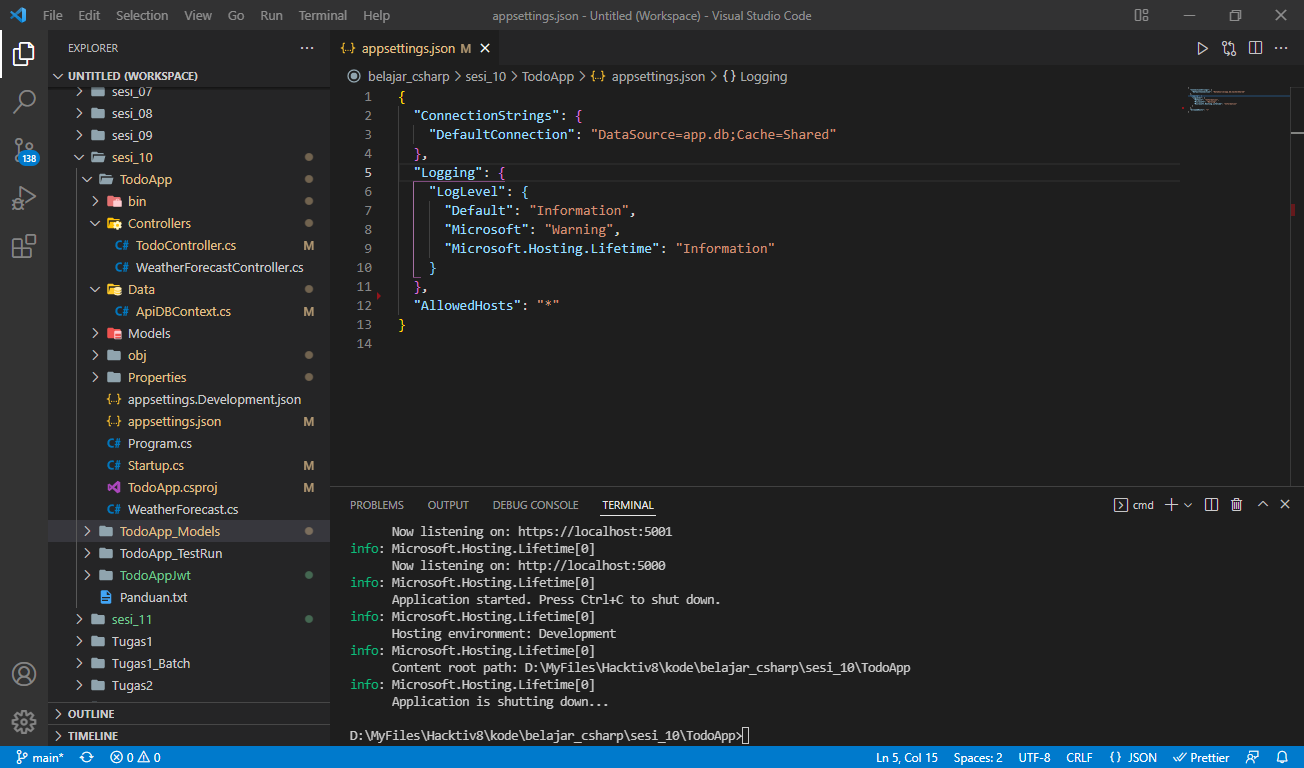
1. Pada appsettings.json tambahkan

"ConnectionStrings": {

    "DefaultConnection": "DataSource=app.db;Cache=Shared"

  },

Jadi seperti ini :



1. Pada ‘startup.cs’ pada bagian atas tambahkan

using Microsoft.EntityFrameworkCore;

untuk menggunakan Microsoft.EntityFrameworkCore.Tools dan Microsoft.EntityFrameworkCore.Sqlite

1. method ‘ConfigurationServices’ tambahkan

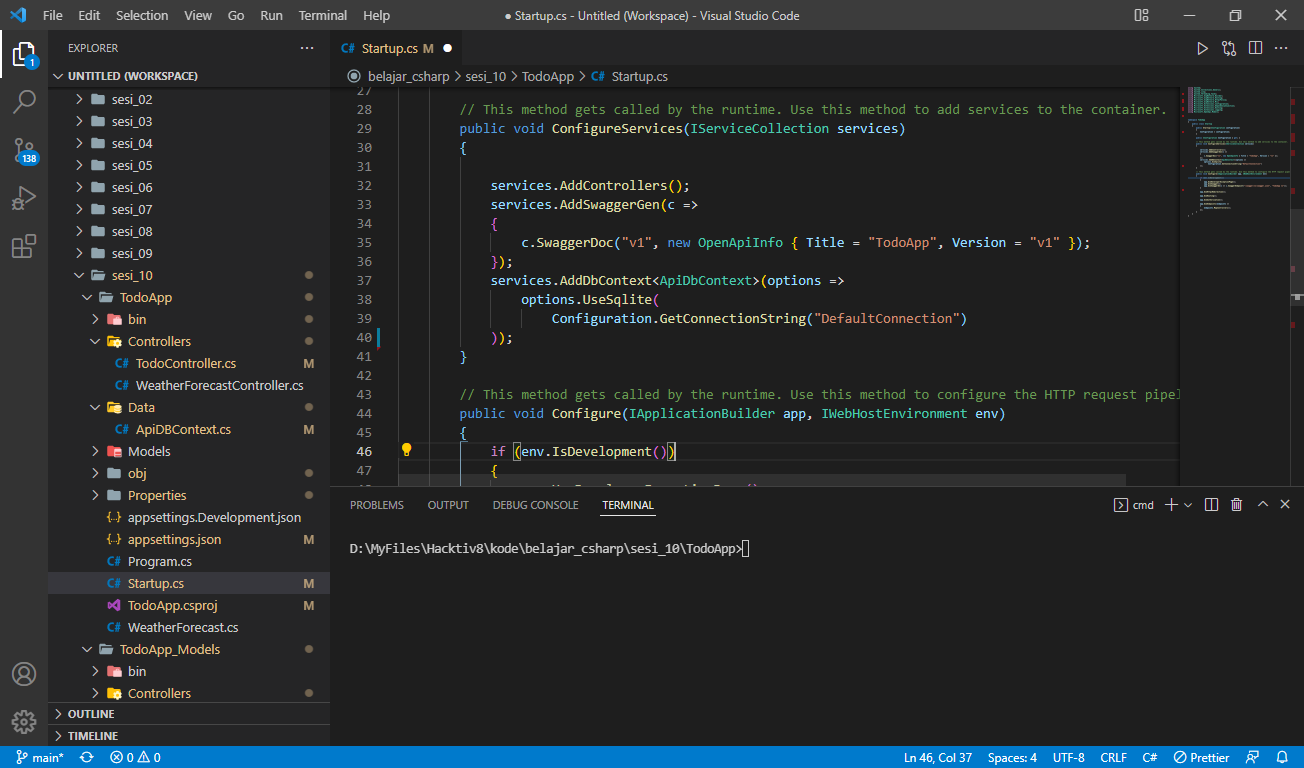
services.AddDbContext<ApiDbContext>(options =>

                options.UseSqlite(

                    Configuration.GetConnectionString("DefaultConnection")

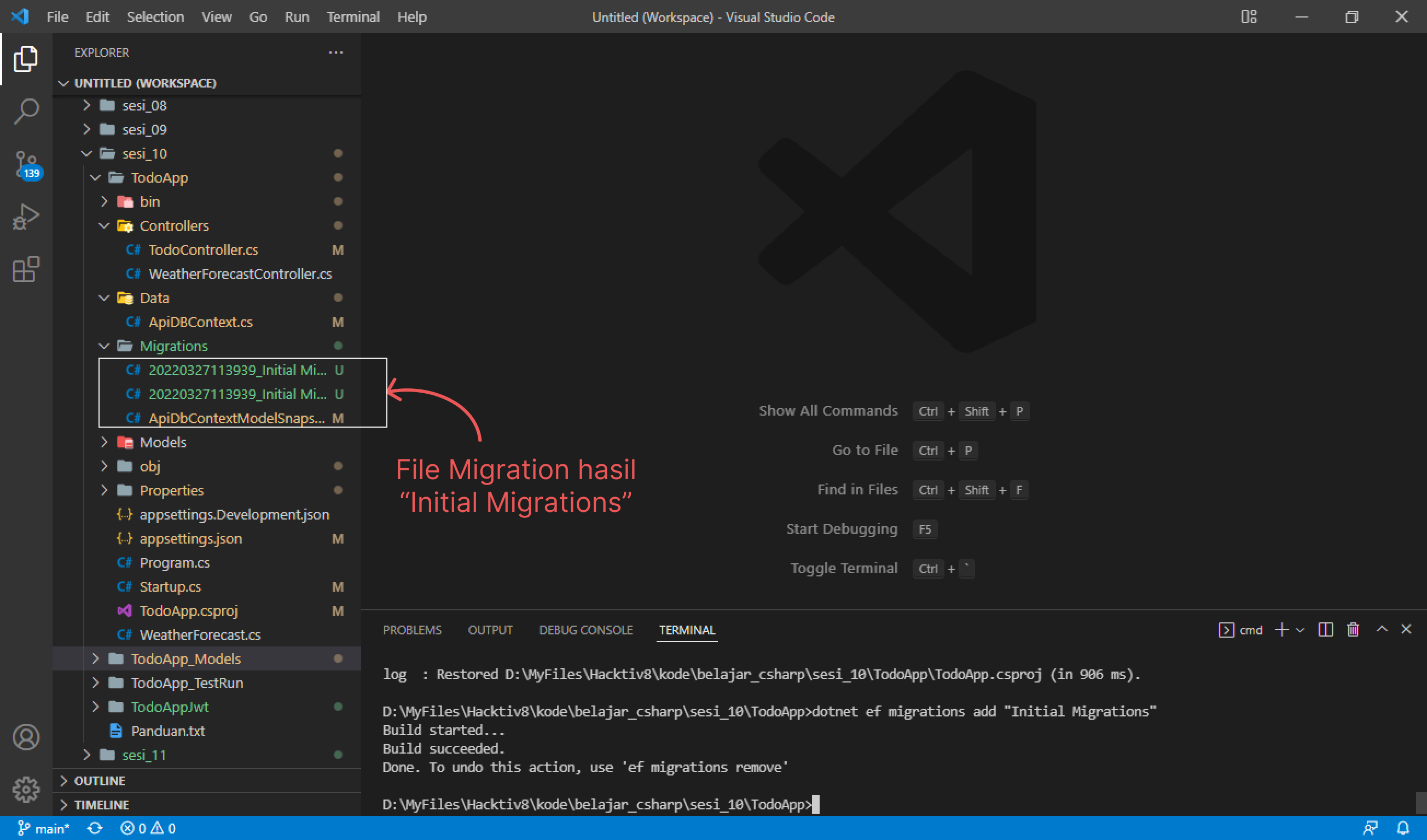
            ));

Menjadi seperti ini:



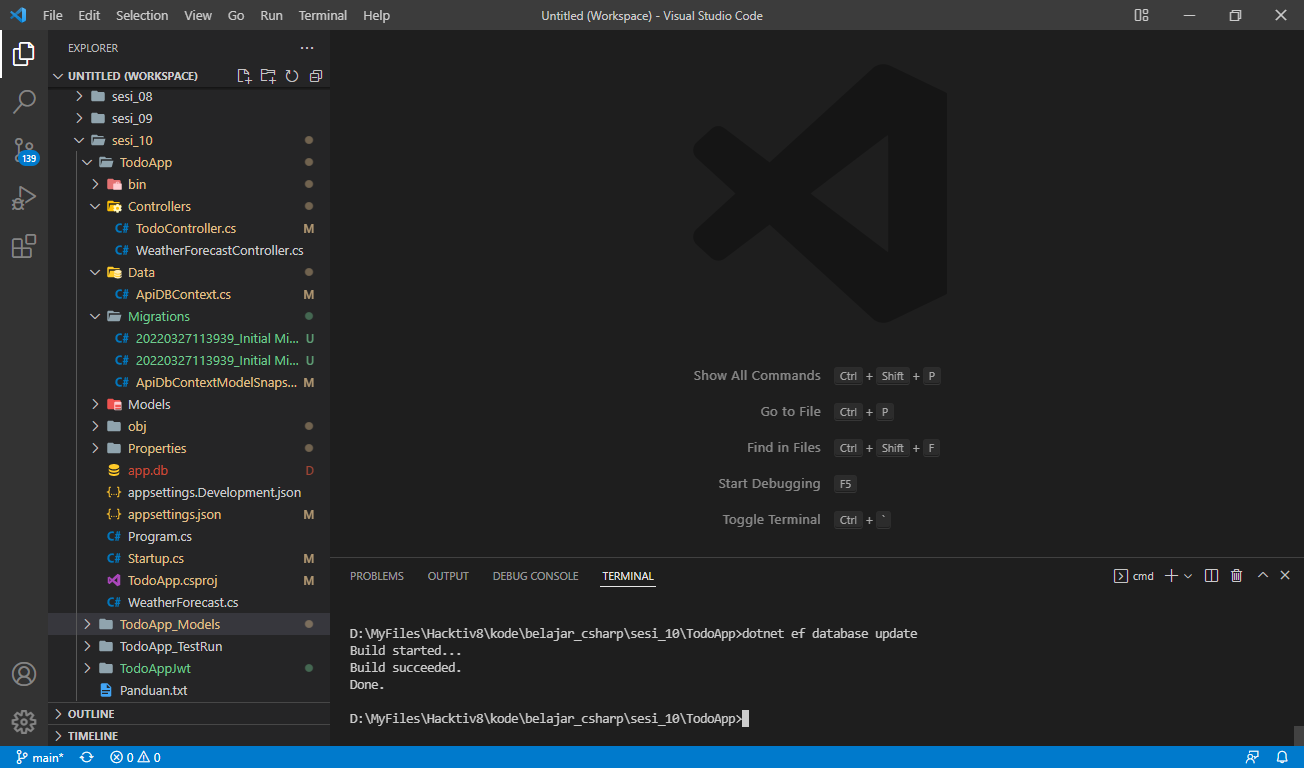
1. Pada terminal jalankan untuk membuat migrations

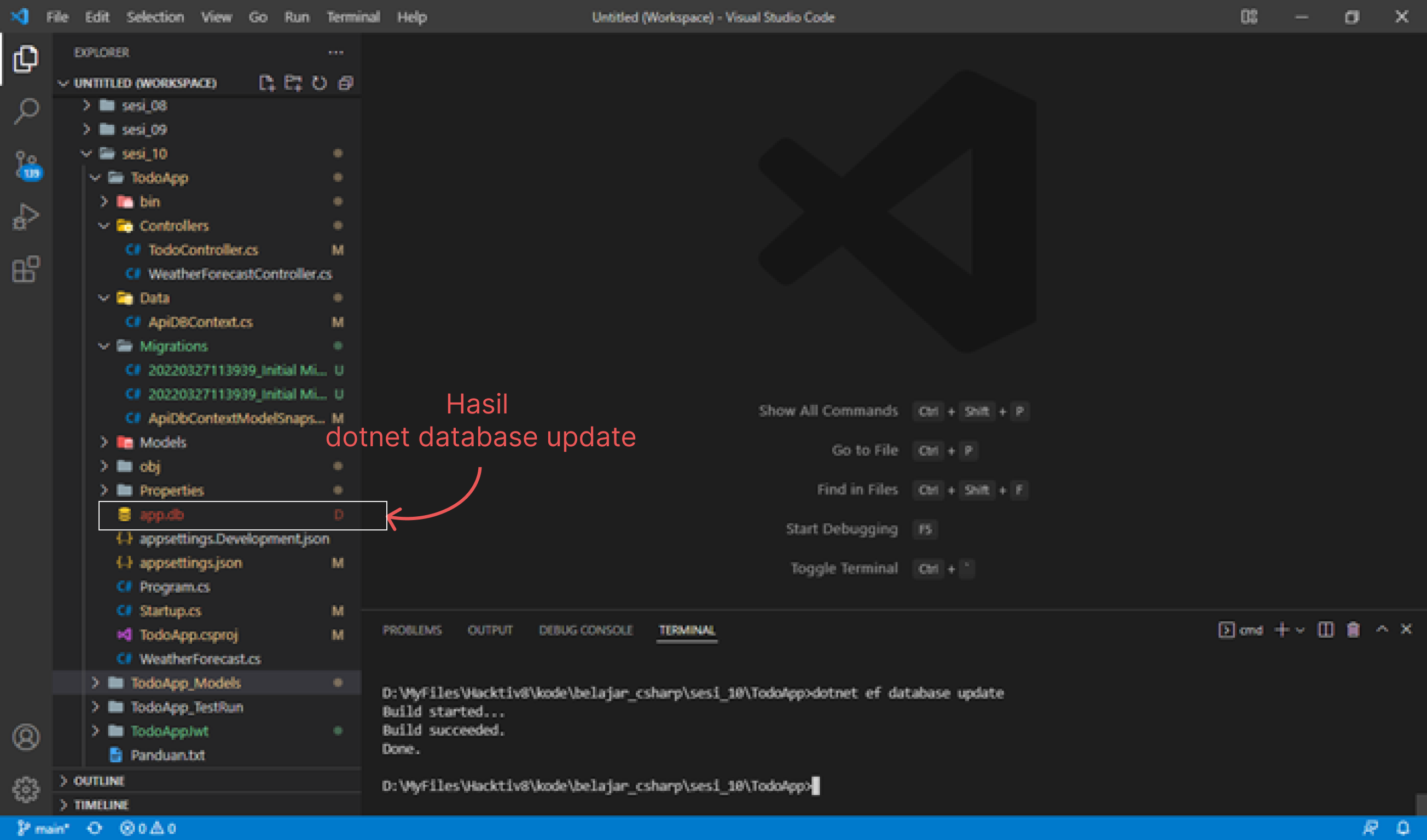
dotnet ef migrations add "Initial Migrations"

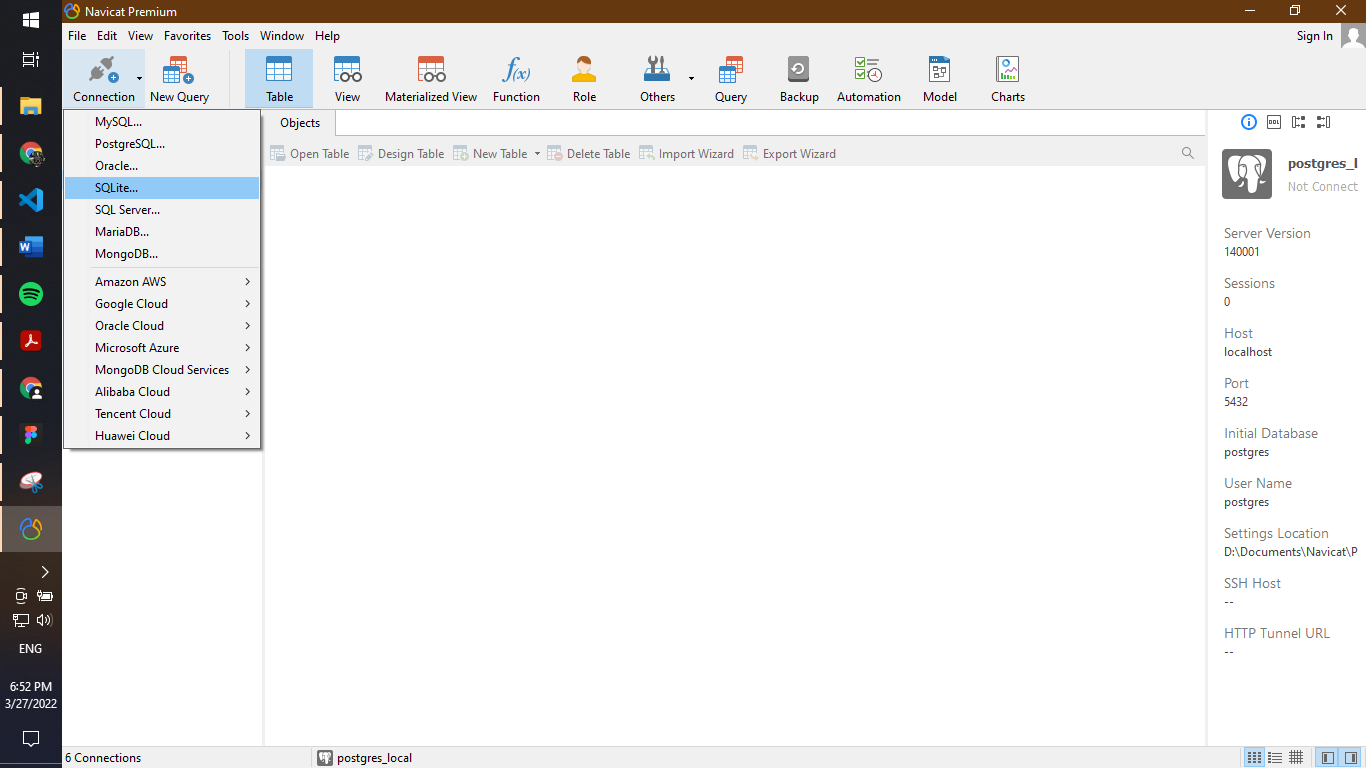


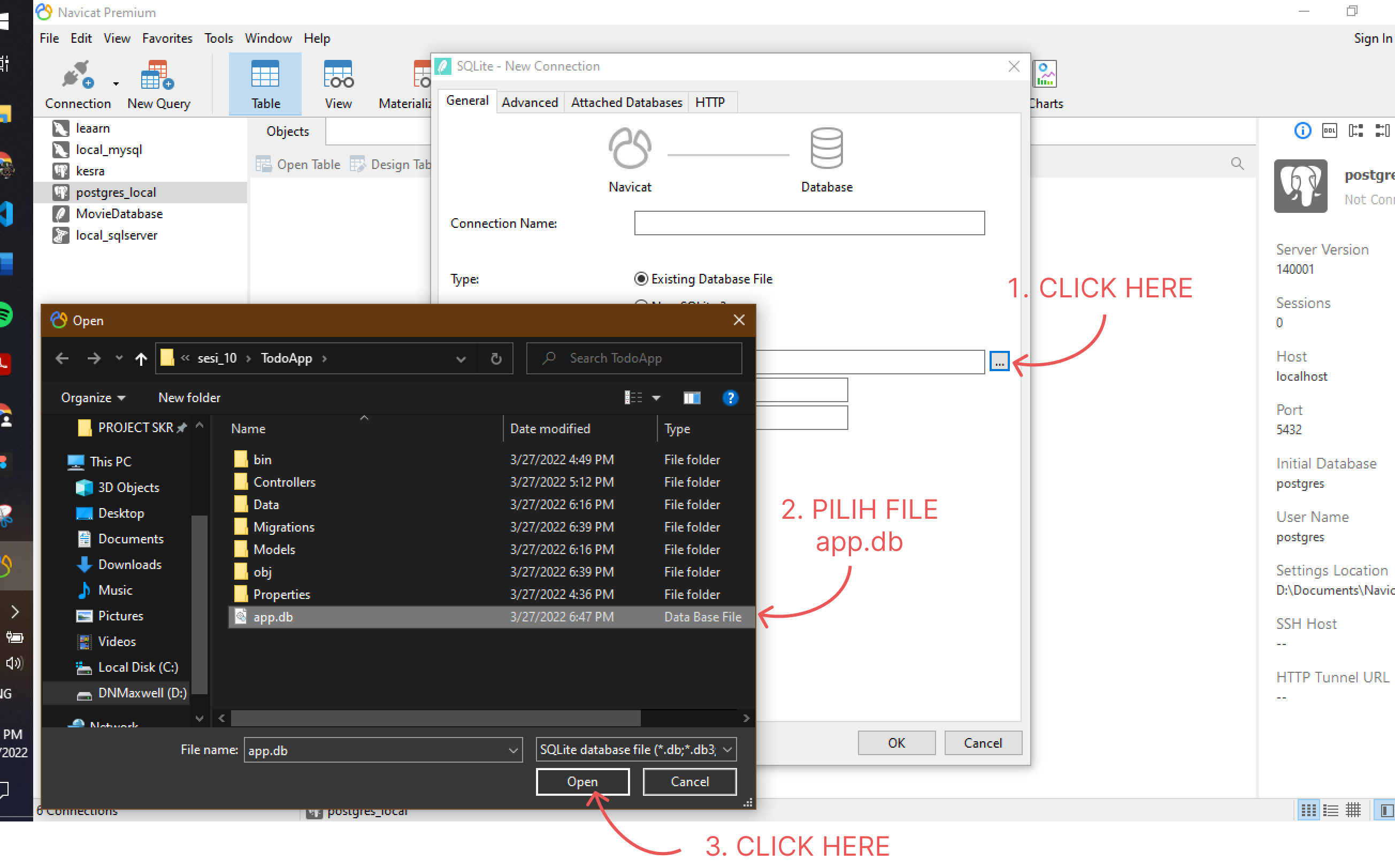
1. Lalu jalankan

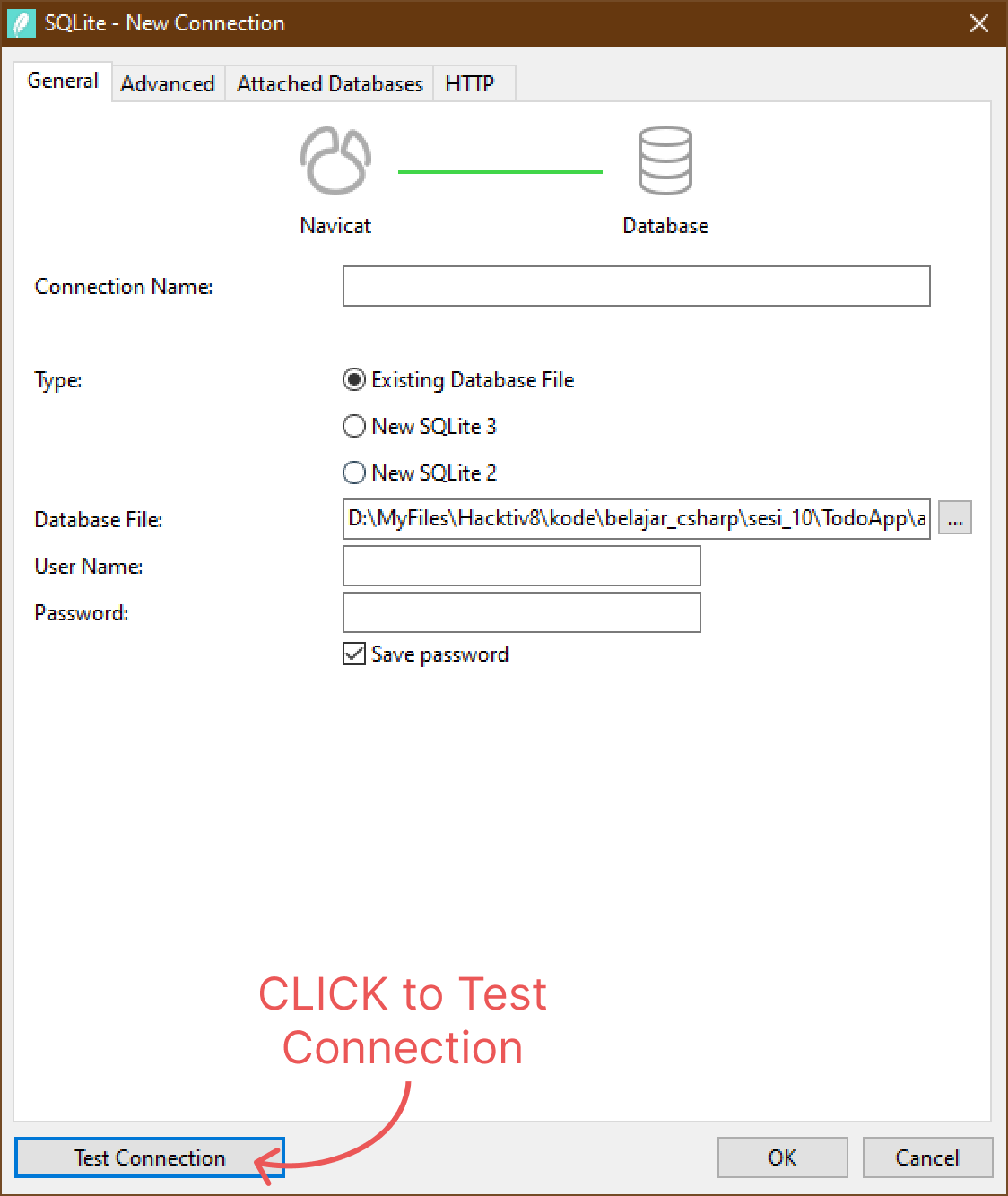
dotnet ef database update



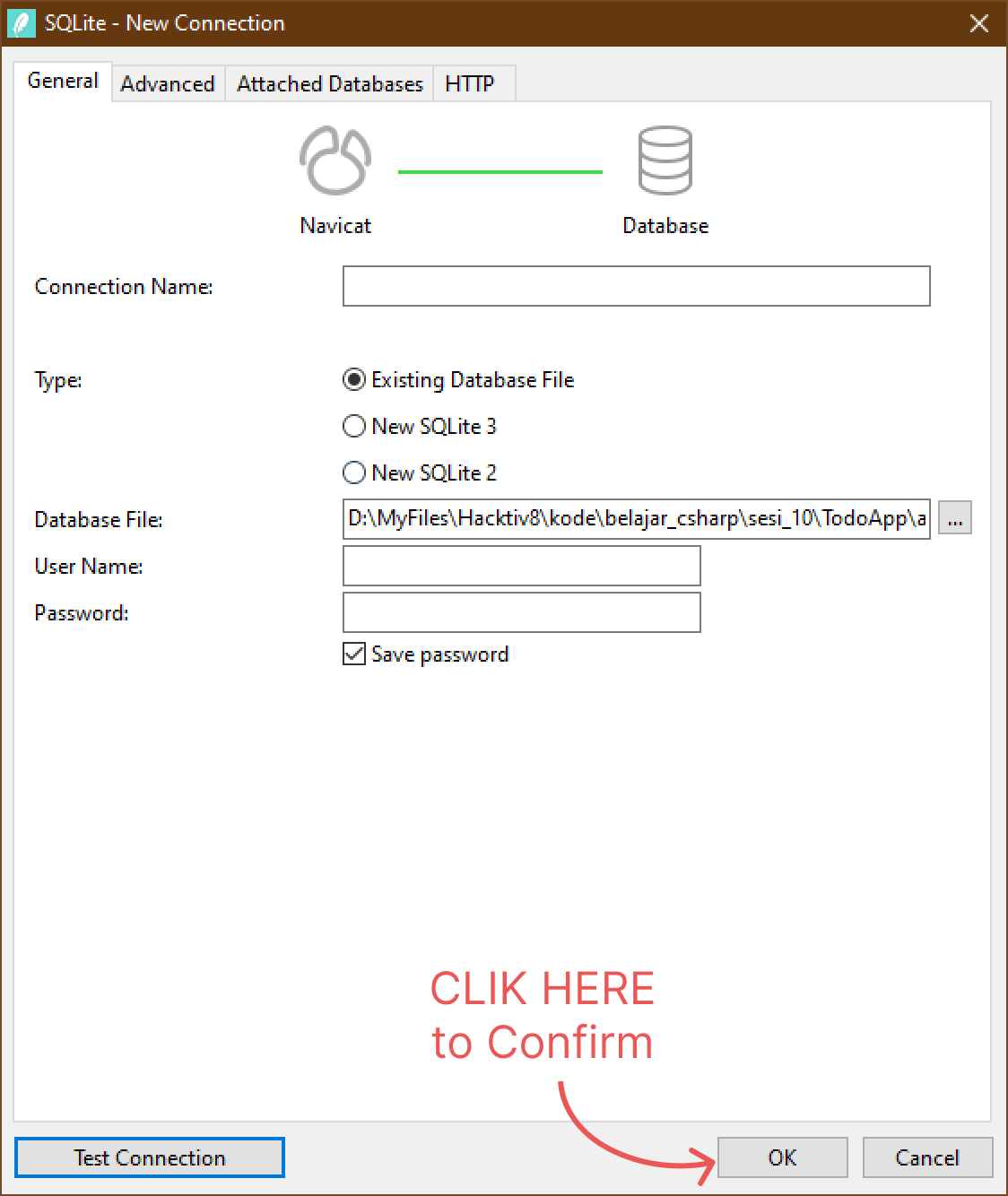


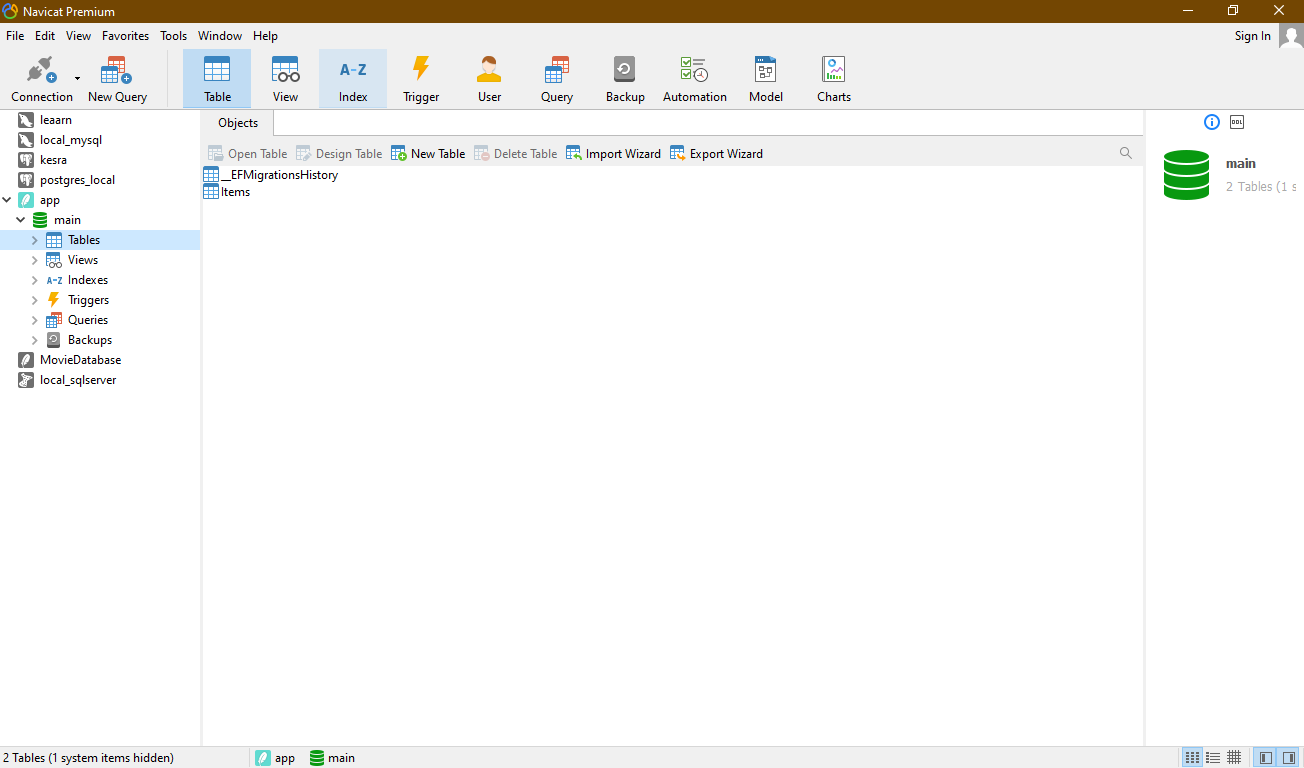


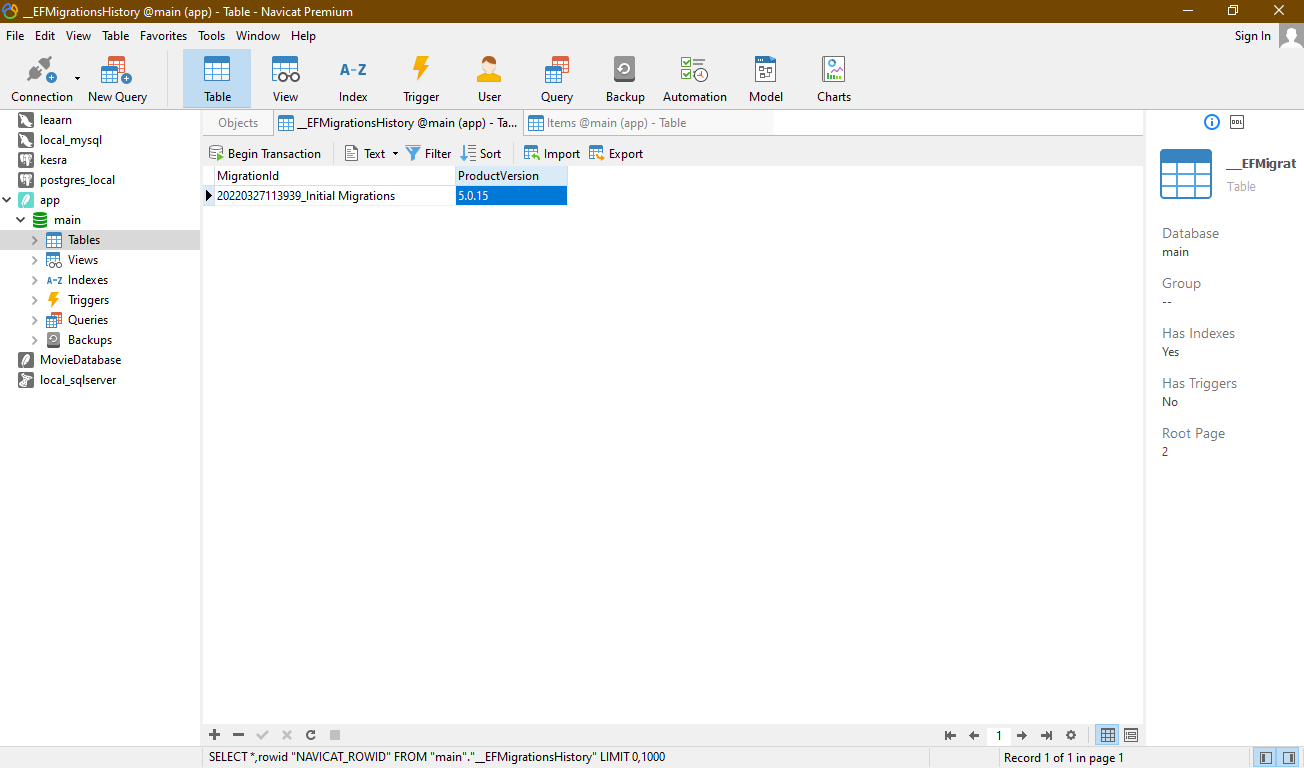


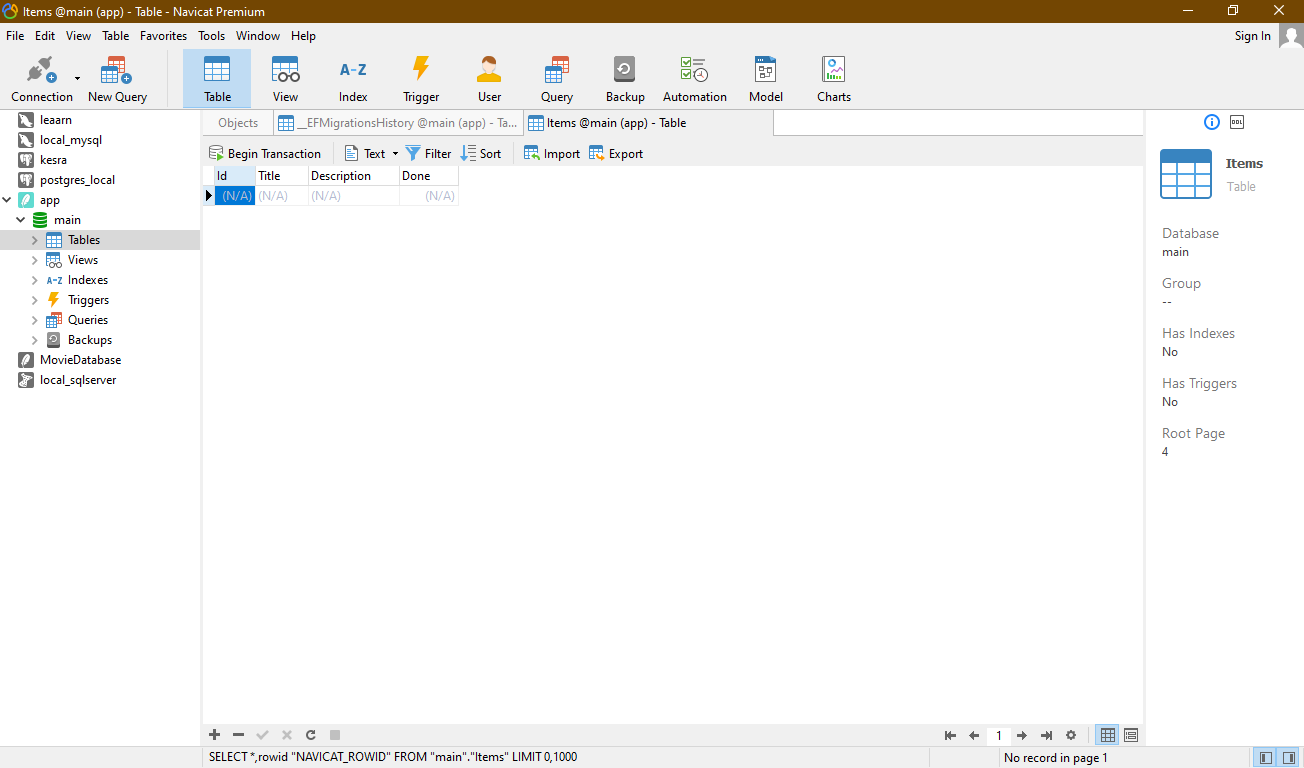












1. Penambahan Model GET dan POST pada ‘Web API TodoApp’
2. Pada **‘Controller>TodoController’**, bagian atas (include), tambahkan

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using TodoApp.Data;

using TodoApp.Models;

System.Threading.Tasks untuk Task

Microsoft.EntityFrameworkCore untuk sqlite

1. Pada **‘Controller>TodoController’**, bagian method TodoController: ControllerBase

private readonly ApiDbContext \_context;

        public TodoController(ApiDbContext context){

            \_context = context;

        }

        [HttpGet]

        public async Task<IActionResult> GetItems(){

            var items = await \_context.Items.ToListAsync();

            return Ok(items);

        }

        [HttpPost]

        public async Task<IActionResult> CreateItem(ItemData data){

            if(ModelState.IsValid){

                await \_context.Items.AddAsync(data);

                await \_context.SaveChangesAsync();

                return CreatedAtAction(nameof(GetItems), new {id = data.Id}, data);

            }

                return new JsonResult("Something went wrong") {StatusCode = 500};

        }

        [HttpGet("{id}")]

        public async Task<IActionResult> GetItems(int id){

            var items = await \_context.Items.FirstOrDefaultAsync(x=> x.Id == id);

            if(items== null)

                return NotFound();

            return Ok(items);

        }

        [HttpPut("{id}")]

        public async Task<IActionResult> UpdateItem(int id, ItemData item)

        {

            if(id!=item.Id){

                return BadRequest();

            }

            var existItem = await \_context.Items.FirstOrDefaultAsync(x=> x.Id == id);

            if(existItem == null)

                return NotFound();

            existItem.Title = item.Title;

            existItem.Description = item.Description;

            existItem.Done = item.Done;

            await \_context.SaveChangesAsync();

            return NoContent();

        }

        [HttpDelete("{id}")]

        public async Task<IActionResult> DeleteItem(int id){

            var existItem = await \_context.Items.FirstOrDefaultAsync(x=> x.Id == id);

            if(existItem == null)

                return NotFound();

            \_context.Items.Remove(existItem);

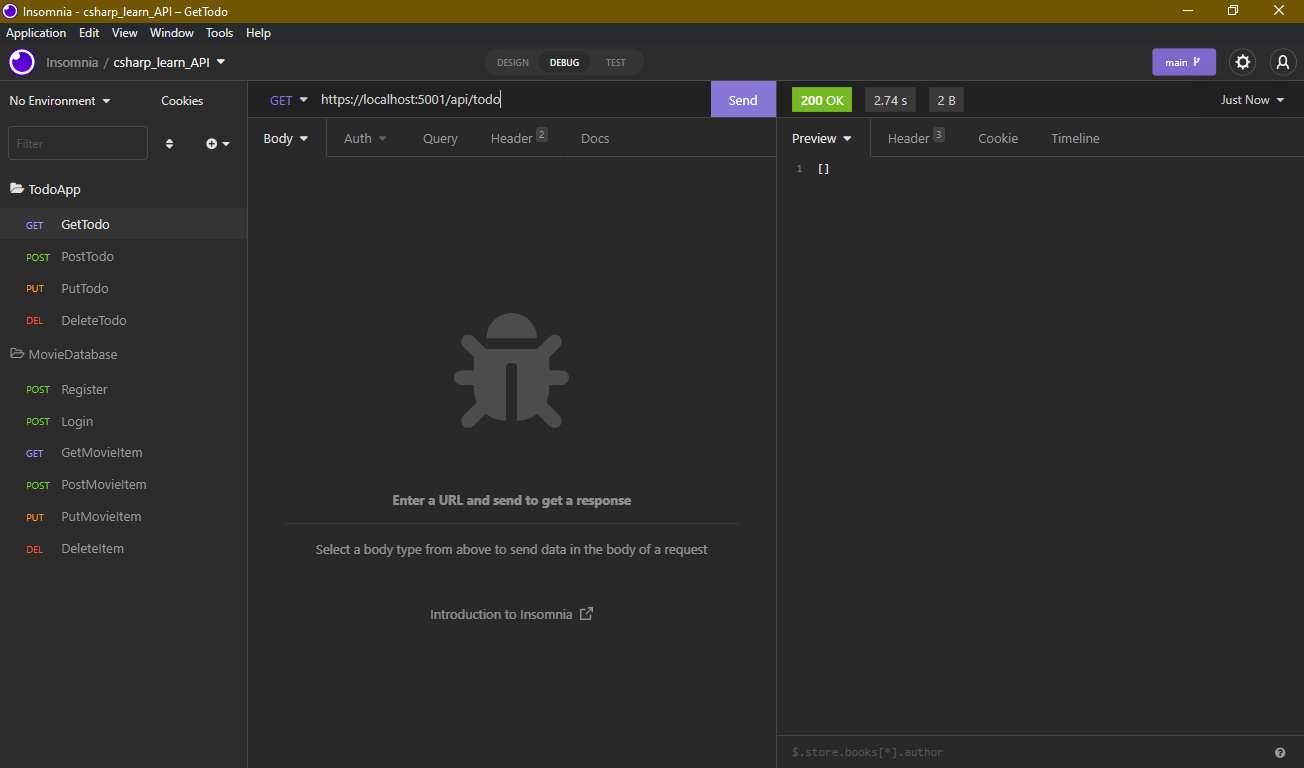
            return Ok(existItem);

        }



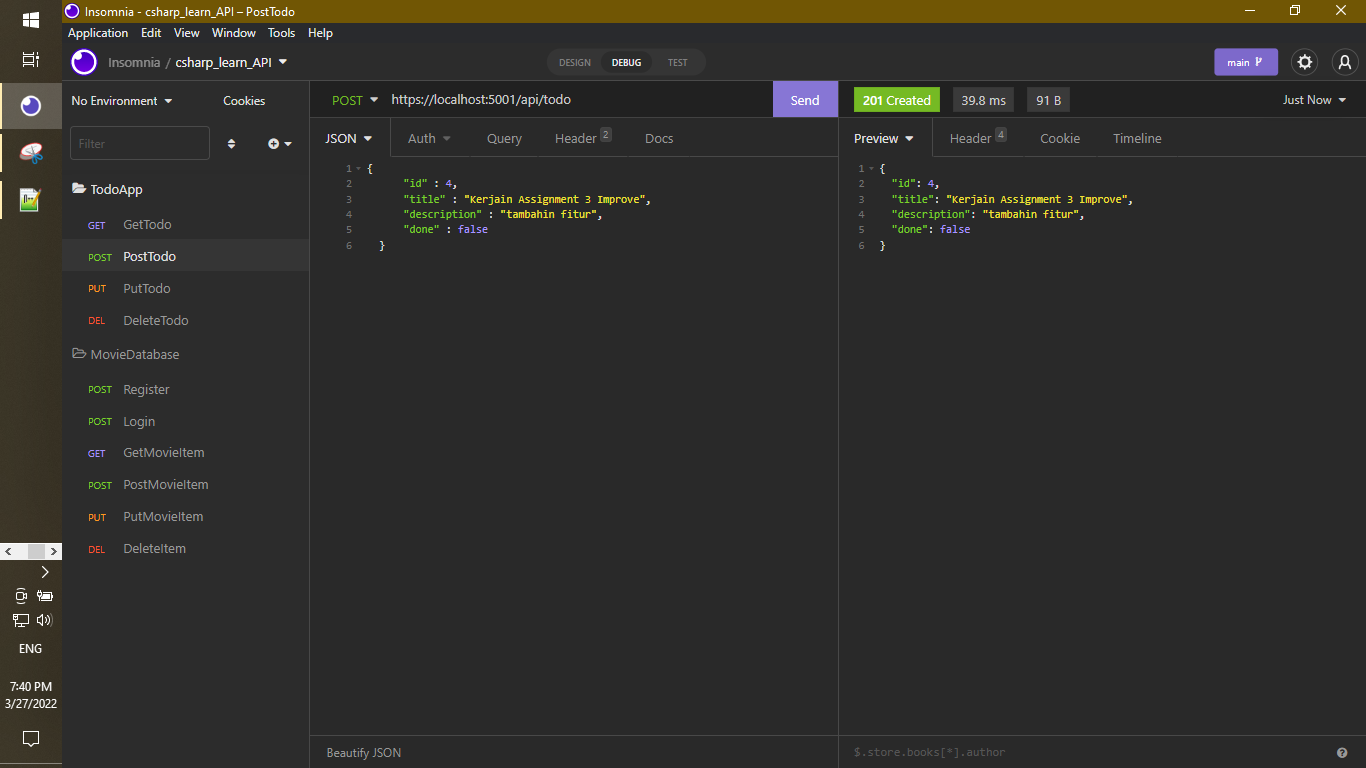
**Request Menggunakan Insomnia**

* GET

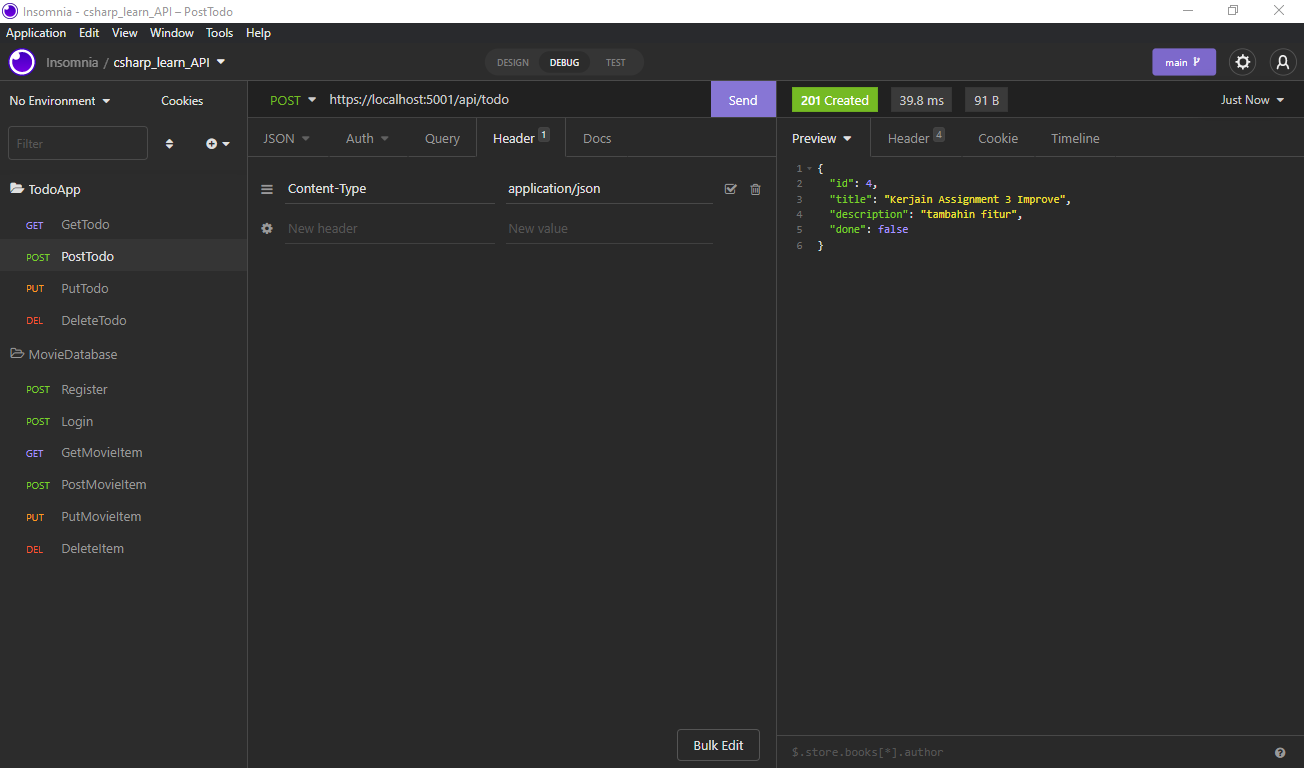


* POST

Body Input JSON:

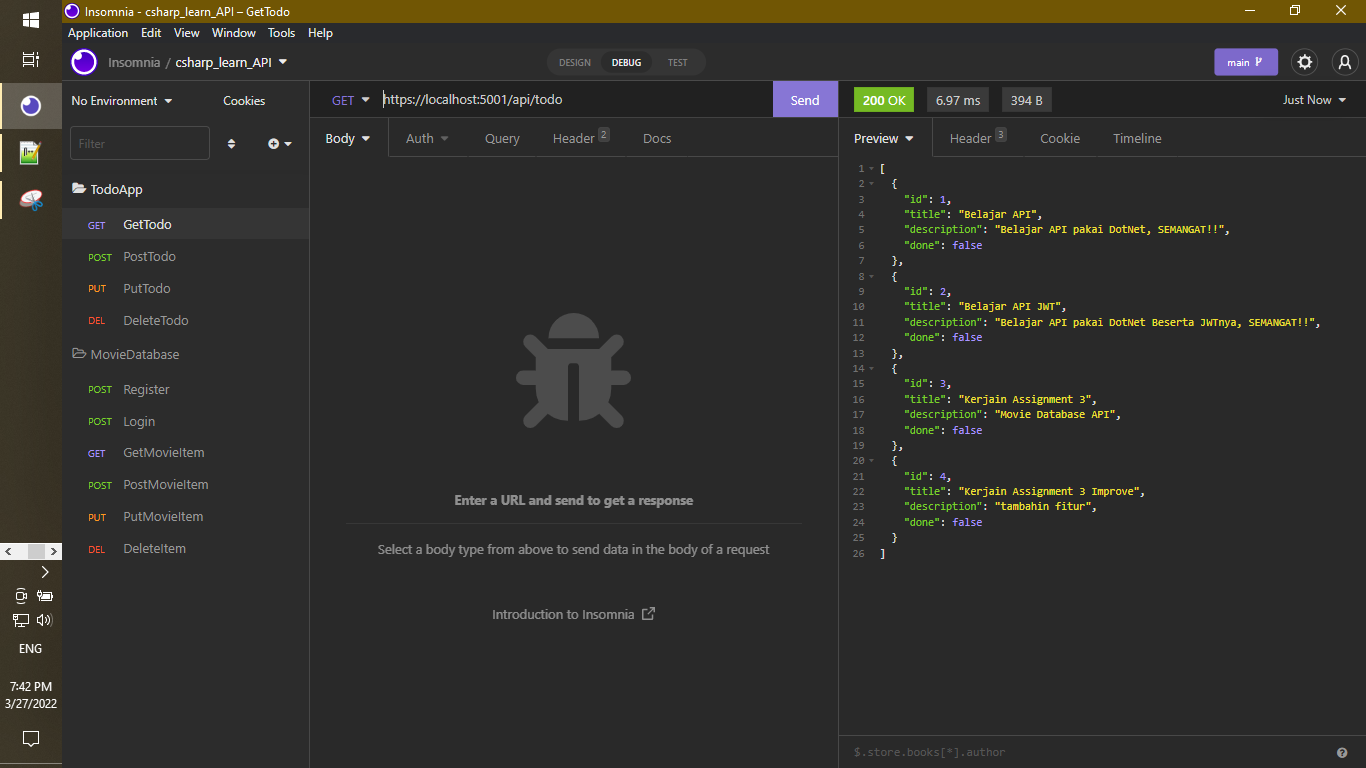


Header:

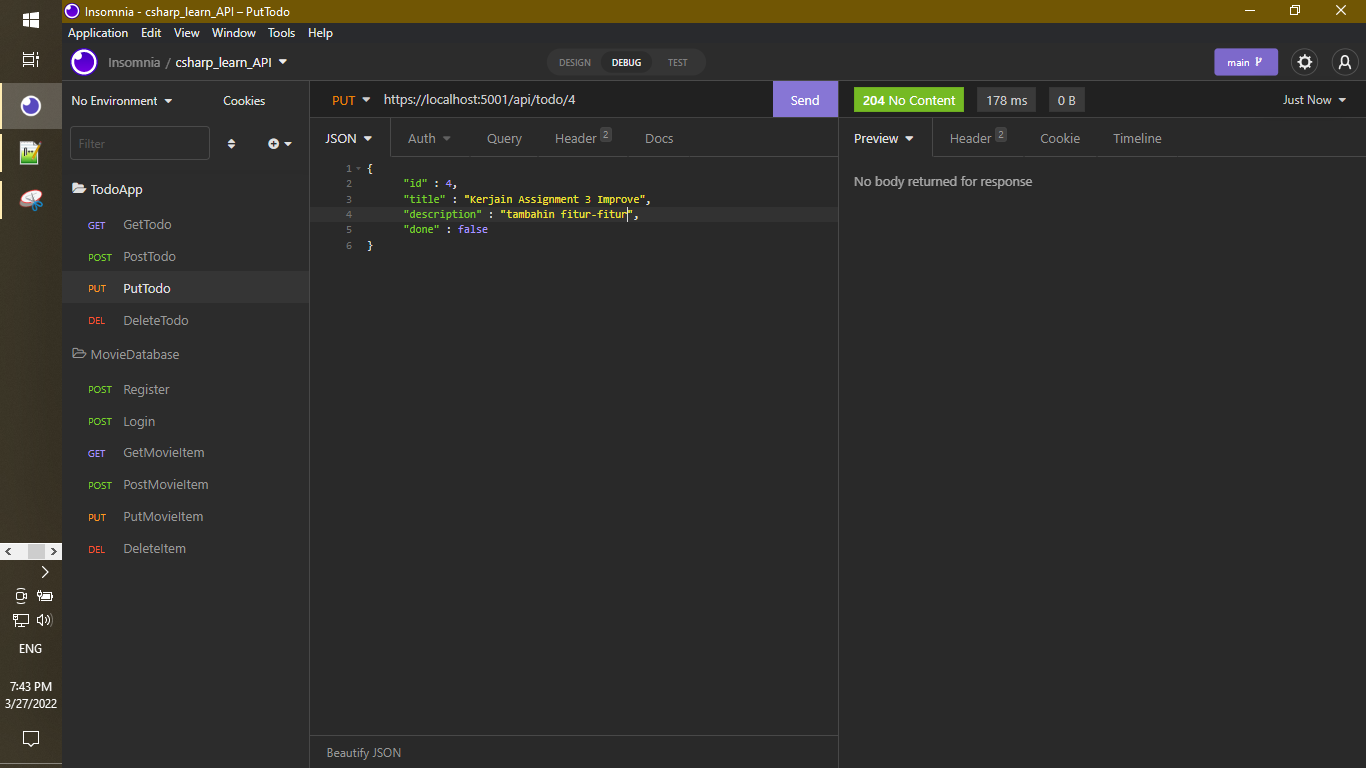


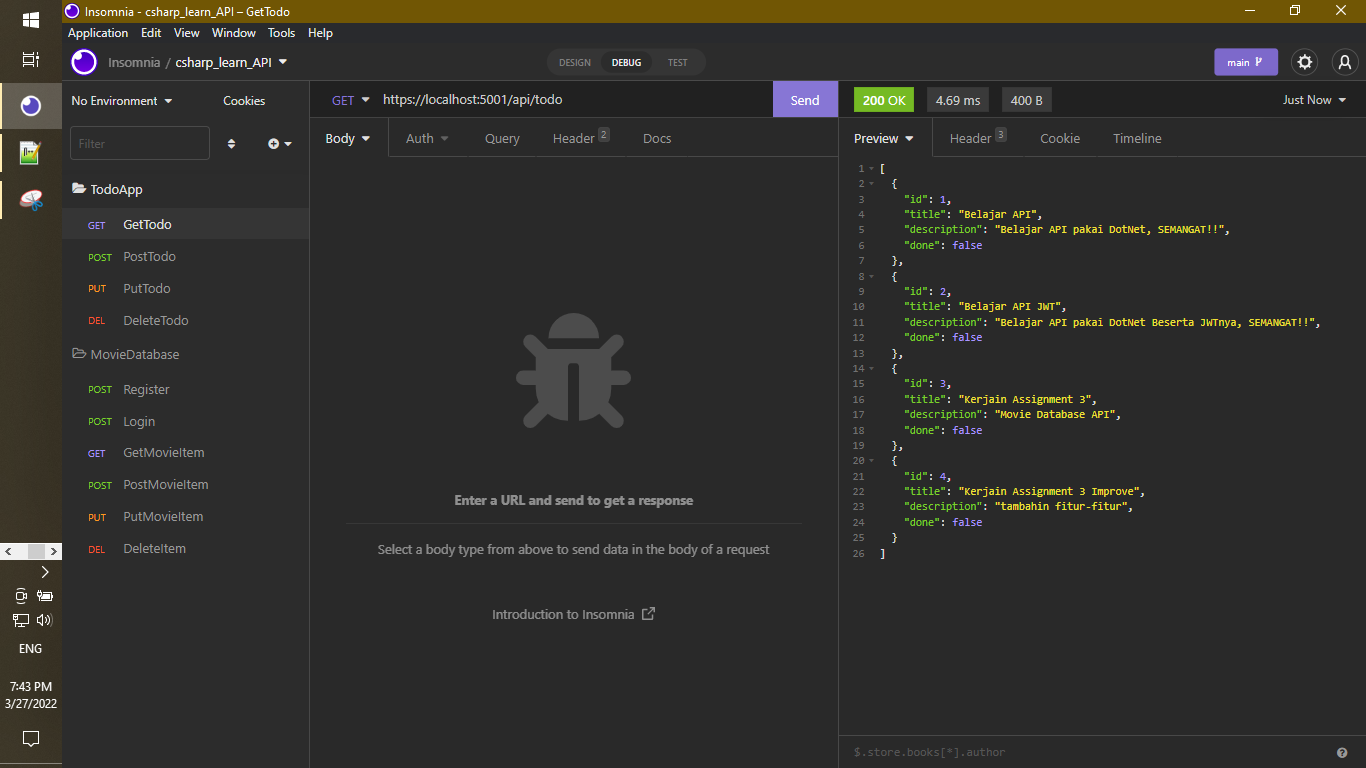
* PUT

GET sebelum PUT

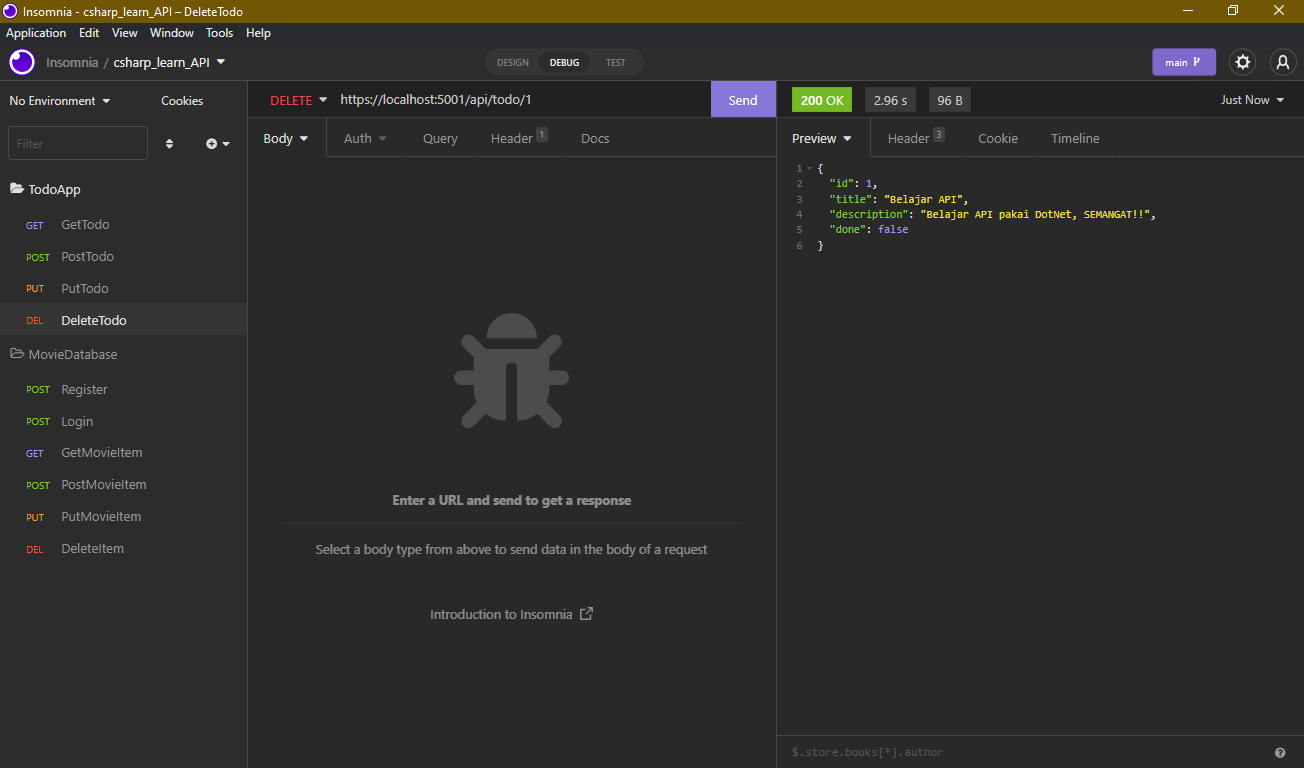


PUT

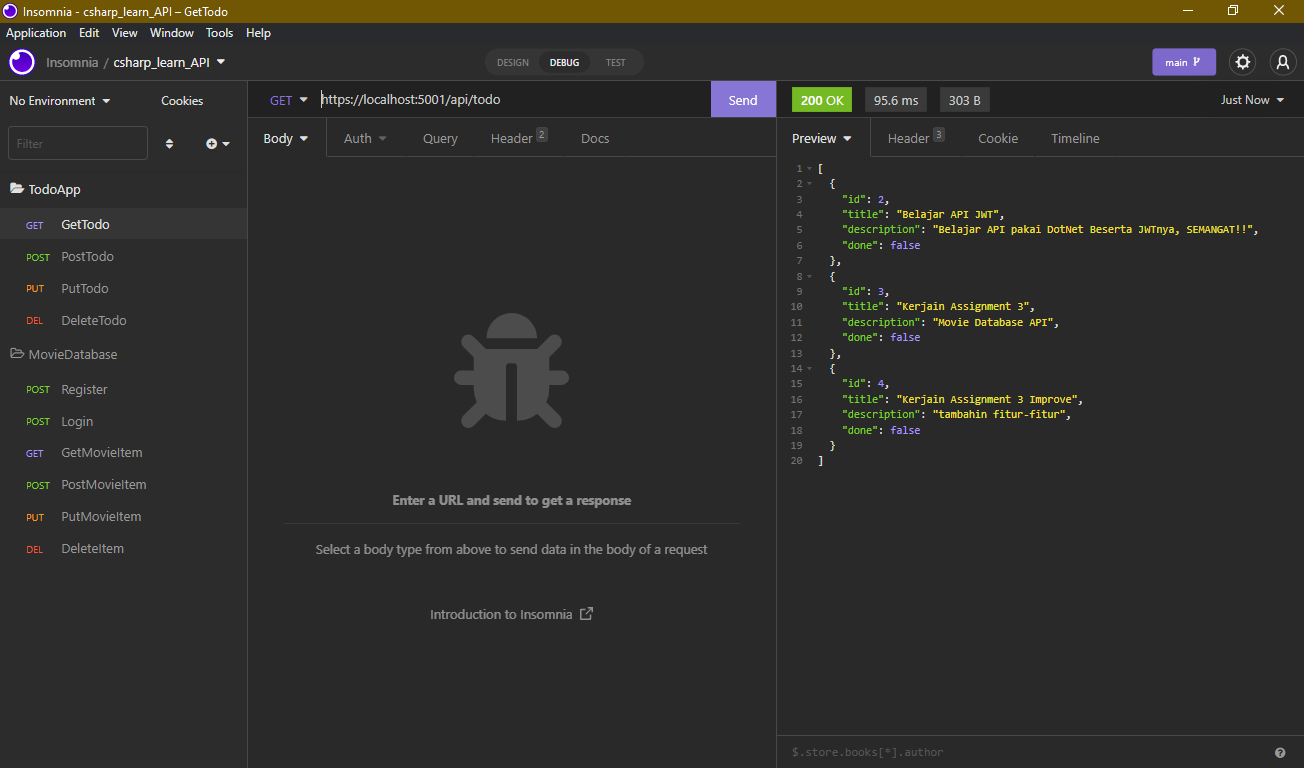
GET setelah PUT



* DELETE



GET setelah DELETE



1. Penambahan Authentication dengan JWT

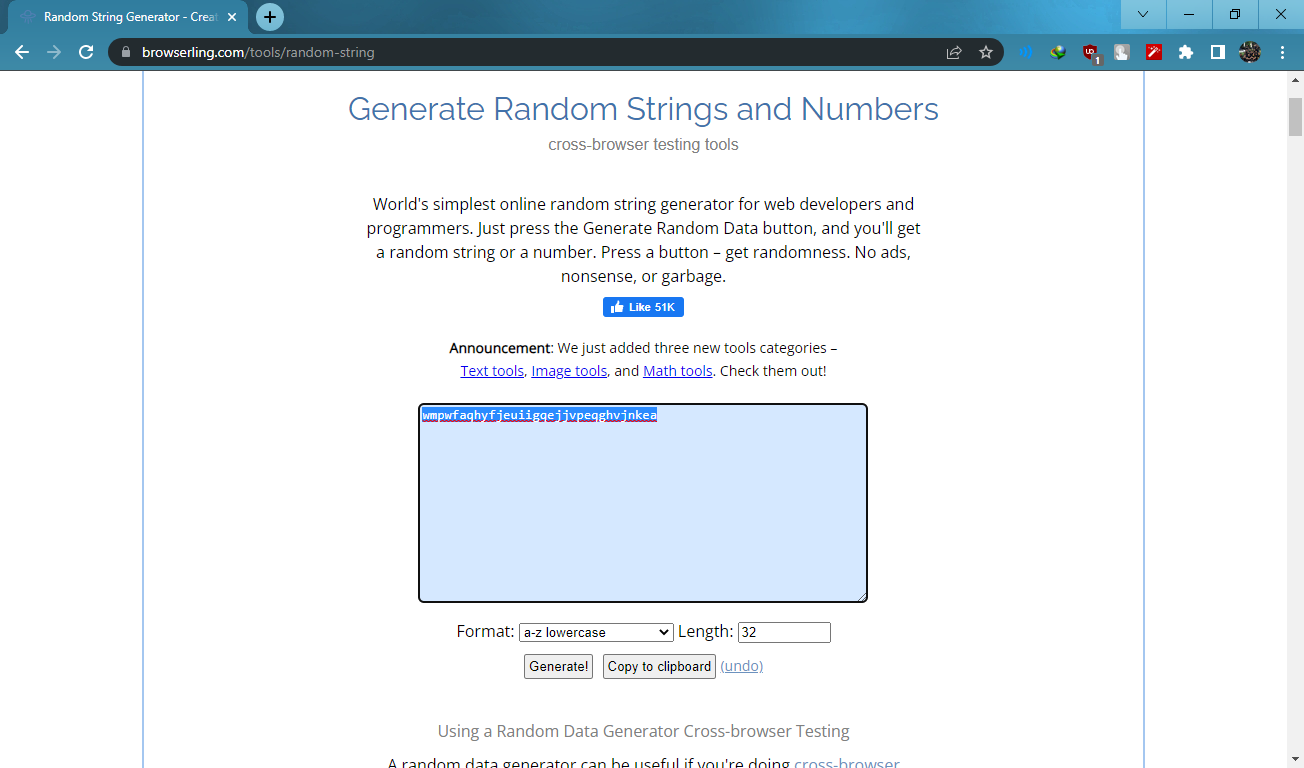
Tambahkan package:

* dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer
* dotnet add package Microsoft.AspNetCore.Identity
* dotnet add package System.IdentityModel.Tokens.Jwt
* dotnet add package Microsoft.AspNetCore.Identity.UI

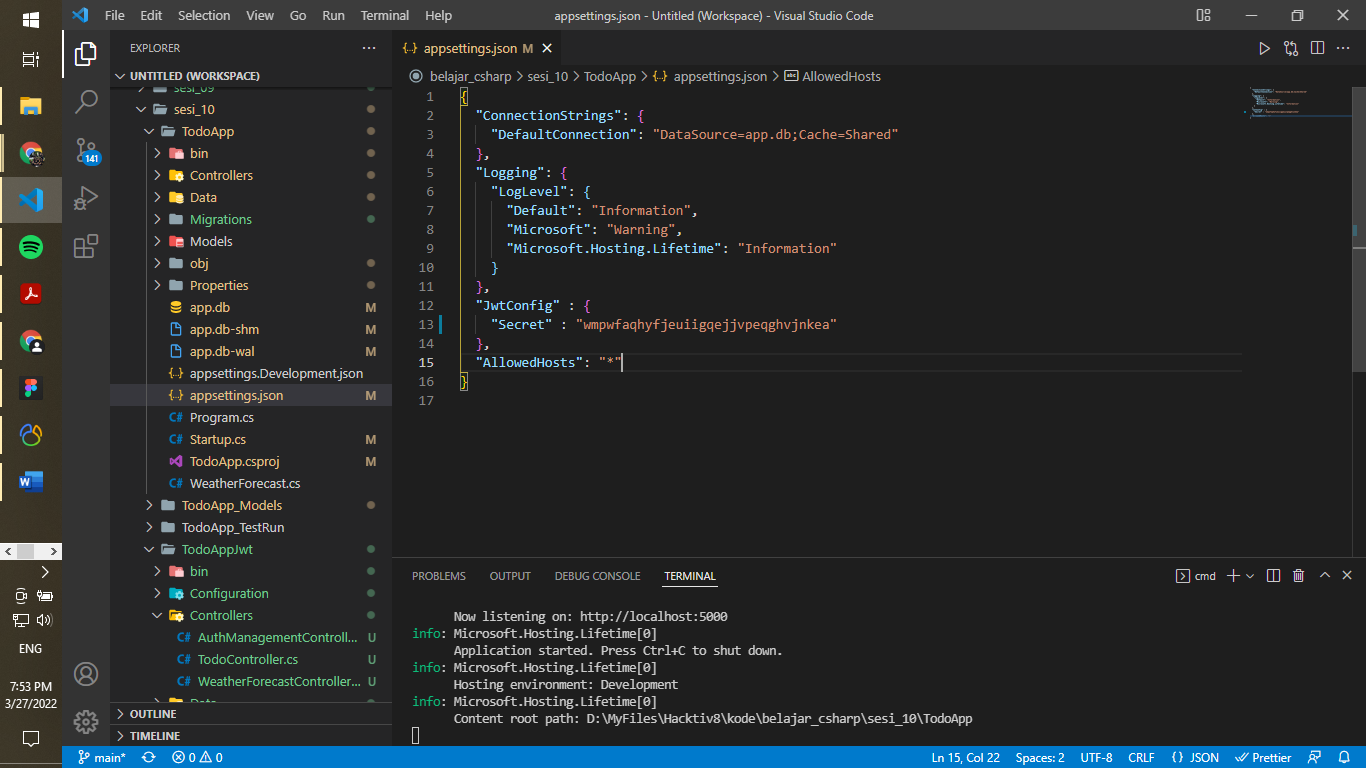
services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true).AddEntityFrameworkStores<ApiDbContext>();

1. Buka <https://www.browserling.com/tools/random-string>, Setting Length = 32, lalu klik ‘**generate’**

Hasil generate: wmpwfaqhyfjeuiigqejjvpeqghvjnkea



1. Lalu pada ‘appsetings.json’ tambahkan string hasil generate tersebut pada JwtConfig.Secret seperti dibawah ini



1. Pada ‘TodoApp’ Buat folder ‘Configuration’
2. Pada ‘TodoApp>Configuration’ tambahkan file JwtConfig.cs dengan isi:

namespace TodoApp.Configuration

{

    public class JwtConfig

    {

        public string Secret {get; set;}

    }

}

1. Lalu Pada ‘TodoApp>Startup.cs’ tambahkan di bagian atas file (include)

using System.Text;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using Microsoft.AspNetCore.Identity;

using TodoApp.Configuration;

1. Lalu Pada ‘TodoApp>Startup.cs’ tambahkan pada method ConfigurationServices:

services.Configure<JwtConfig>(Configuration.GetSection("JwtConfig"));

            services.AddAuthentication(options => {

                options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

                options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;

                options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

            } ).AddJwtBearer(jwt=>{

                var key = Encoding.ASCII.GetBytes(Configuration["JwtConfig:Secret"]);

                jwt.SaveToken = true;

                jwt.TokenValidationParameters = new TokenValidationParameters {

                    ValidateIssuerSigningKey = true,

                    IssuerSigningKey = new SymmetricSecurityKey(key),

                    ValidateIssuer = false,

                    ValidateAudience = false,

                    ValidateLifetime = true,

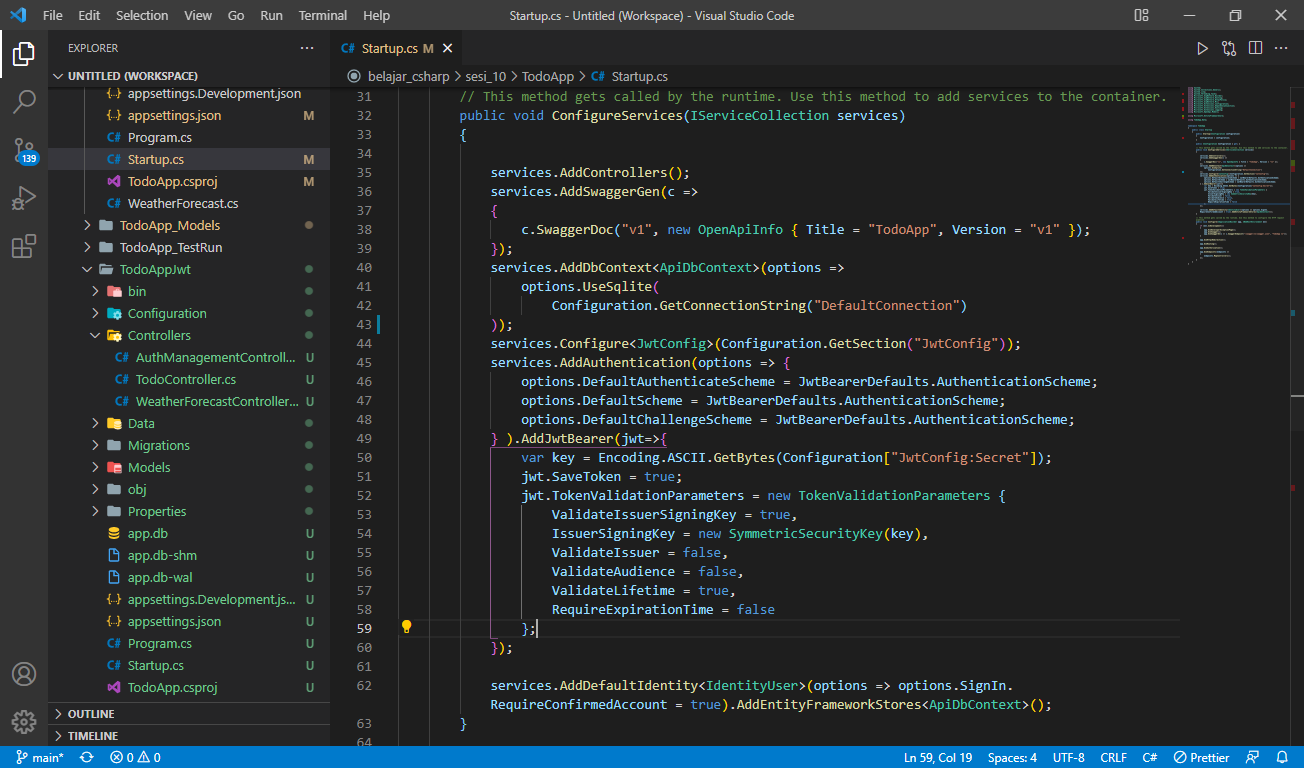
                    RequireExpirationTime = false

                };

            });

            services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true).AddEntityFrameworkStores<ApiDbContext>();

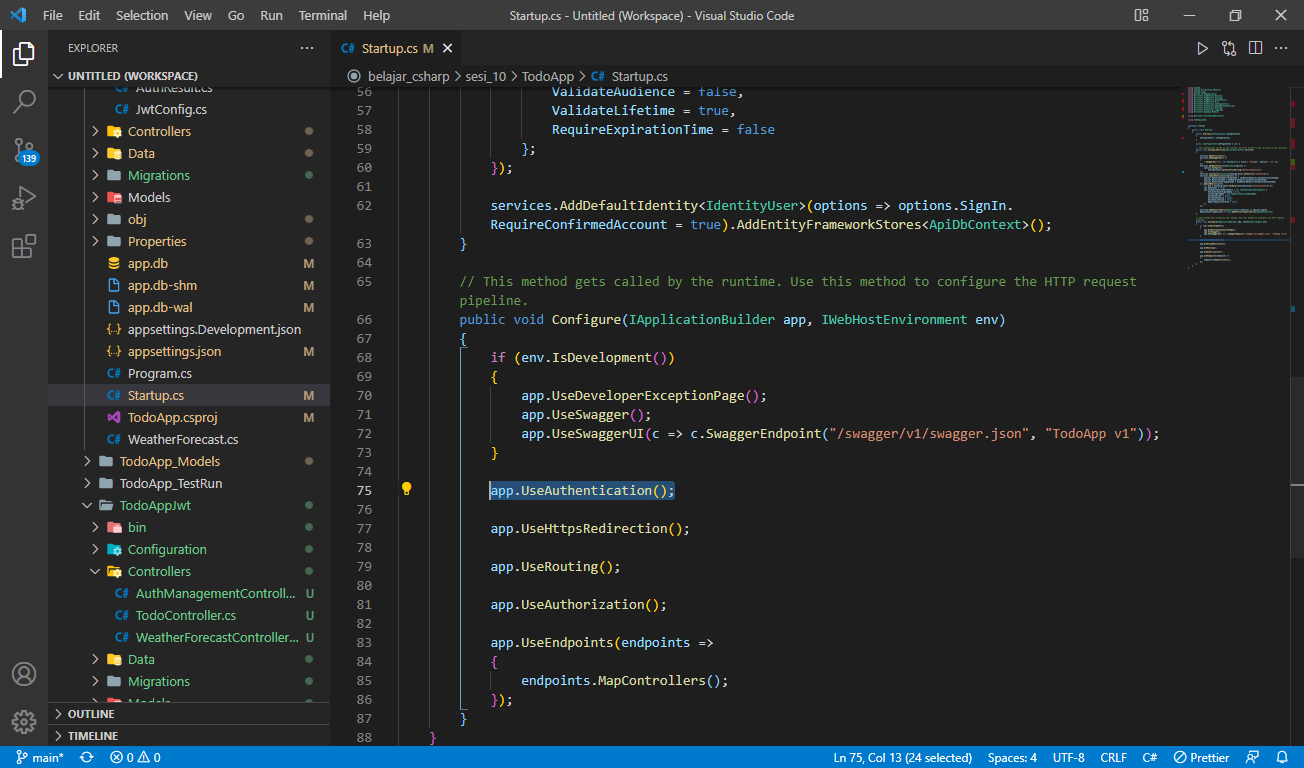
seperti ini:



1. Lalu Pada ‘TodoApp>Startup.cs’ tambahkan pada method Configure:

app.UseAuthentication();

Seperti ini:



1. Pada data ‘TodoApp>Data>ApiDbContext’ rubah

public class ApiDbContext : DbContext

menjadi

public class ApiDbContext : IdentityDbContext

berikut jadinya

using Microsoft.EntityFrameworkCore;

using TodoApp.Models;

namespace TodoApp.Data

{

    public class ApiDbContext : IdentityDbContext

    {

        public virtual DbSet<ItemData> Items {get;set;}

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

        {

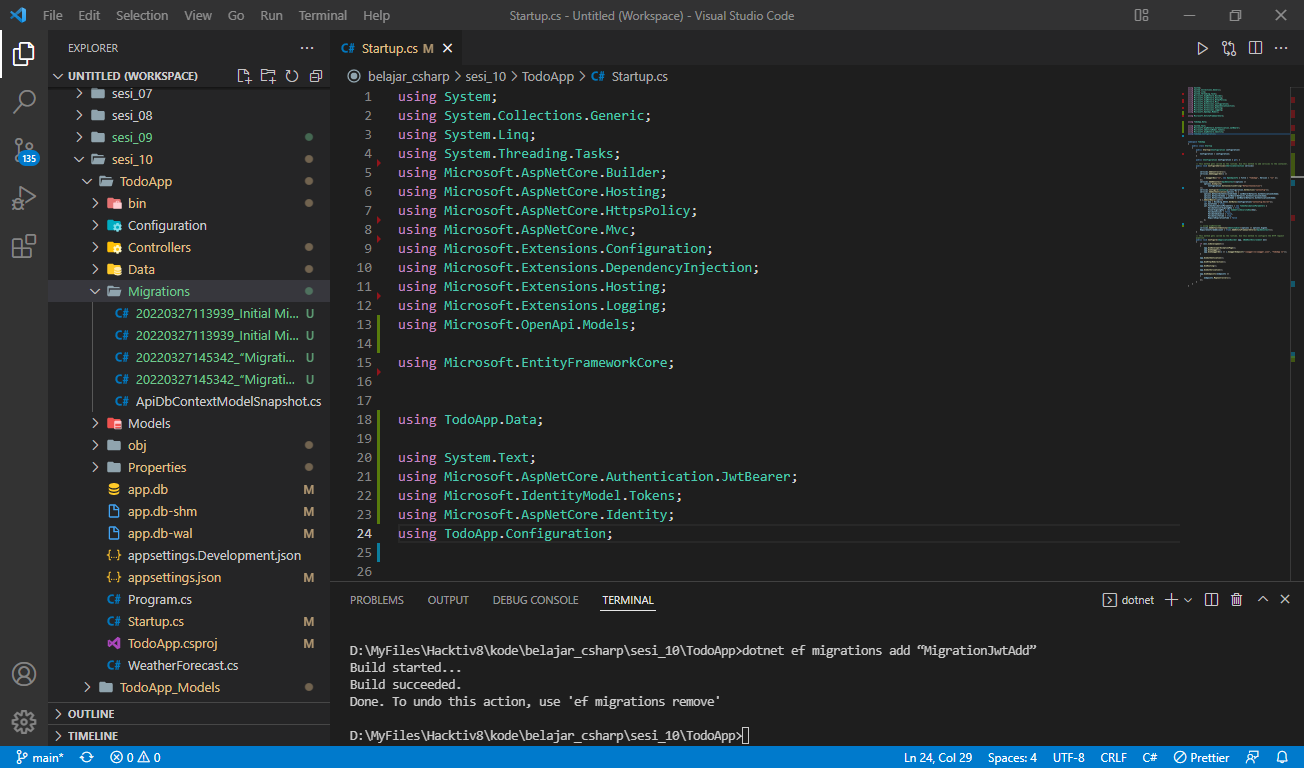
        }

    }

}

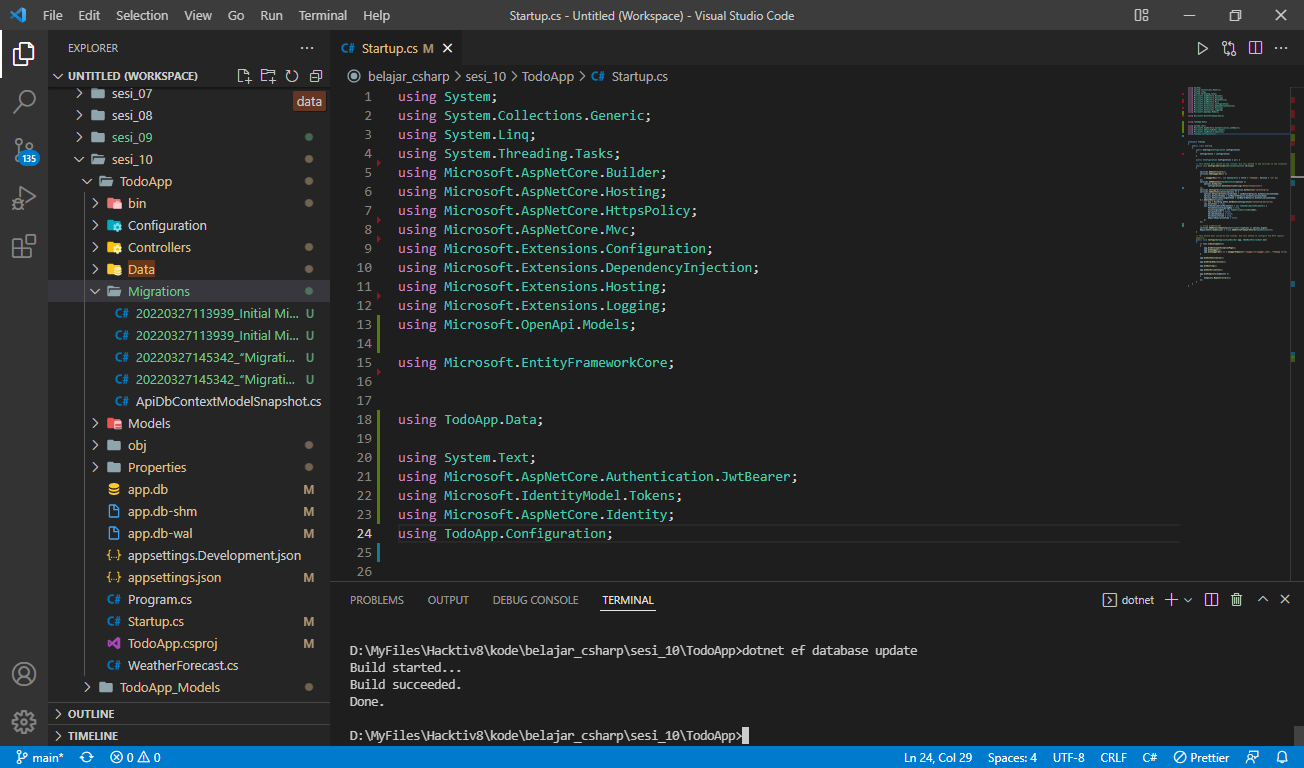
1. Menambahkan Migrasi

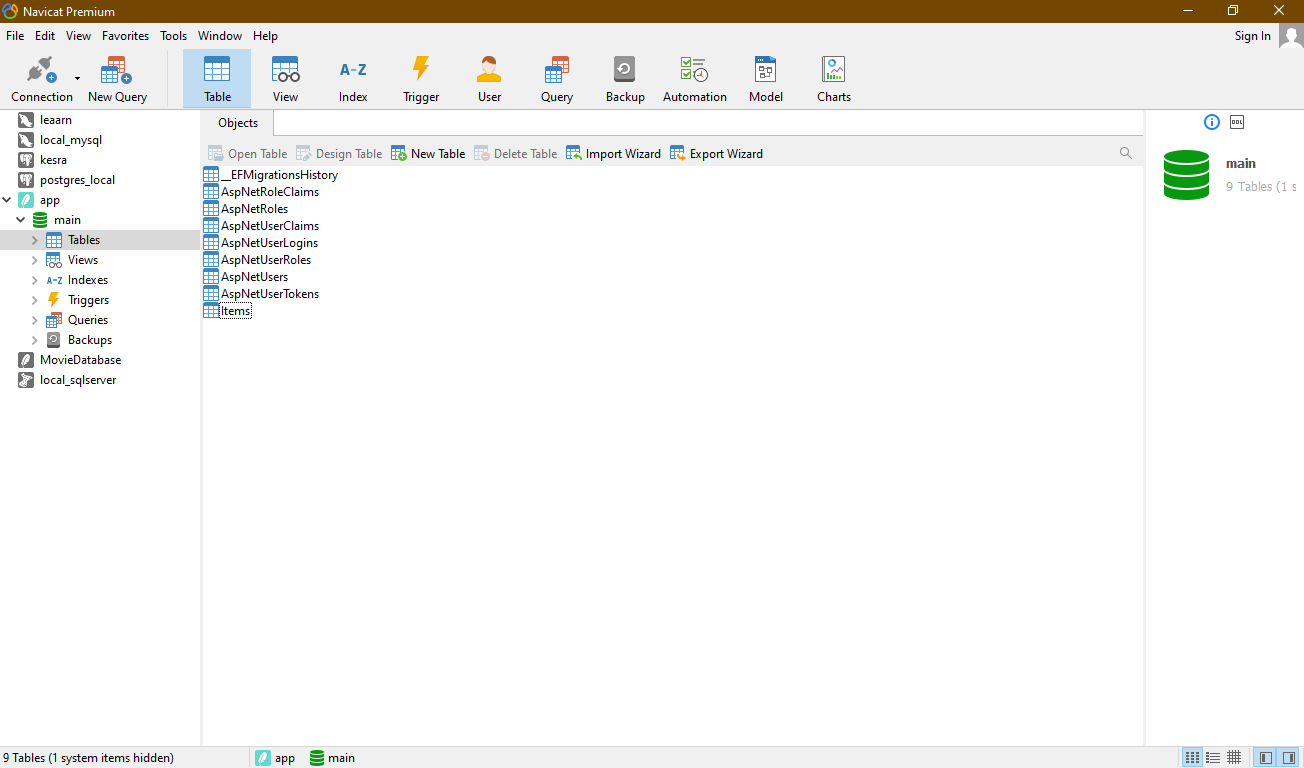
dotnet ef migrations add “MigrationJwtAdd”



1. Update database

dotnet ef database update





1. Penambahan Authentication dengan JWT II : Register
2. Pada ‘TodoApp>Configuration’ tambahkan file AuthResult.cs dengan isi:

using System.Collections.Generic;

namespace TodoApp.Configuration

{

    public class AuthResult

    {

        public string Token {get; set;}

        public bool Success {get; set;}

        public List<string> Errors {get; set;}

    }

}

1. Pada ‘TodoApp>Models’ Tambahkan Folder DTOs
2. Pada ‘TodoApp>Models>DTO’ Tambahkan Folder ‘Requests’ dan ‘Responses’
3. Pada ‘TodoApp>Models>DTO>Requests’ Tambahkan File ‘UserRegistrationDto.cs’, dengan isi file:

UserRegistrationDto.cs:

using System.ComponentModel.DataAnnotations;

namespace TodoApp.Models.DTOs.Requests

{

    public class UserRegistationDto

    {

        [Required]

        public string Username {get; set;}

        [Required]

        [EmailAddress]

        public string Email {get;set;}

        [Required]

        public string Password {get;set;}

    }

}

1. Pada ‘TodoApp>Models>DTO>Responses’ tambahkan file ‘RegistrationResponse.cs’ dengan isi:

using TodoApp.Configuration;

namespace TodoApp.Models.DTOs.Responses

{

    public class RegistrationResponse: AuthResult

    {

    }

}

1. Pada ‘TodoApp>Controllers’ tambahkan file ‘AuthManagementController.cs’

using System;

using System.Collections.Generic;

using System.IdentityModel.Tokens.Jwt;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

using System.Security.Claims;

using Microsoft.IdentityModel.Tokens;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Options;

using TodoApp.Models.DTOs.Requests;

using TodoApp.Models.DTOs.Responses;

using TodoApp.Configuration;

namespace TodoApp.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

    public class AuthManagementController : ControllerBase

    {

        private readonly UserManager<IdentityUser> \_userManager;

        private readonly JwtConfig \_jwtConfig;

        public class AuthManagementController : ControllerBase

    {

        private readonly UserManager<IdentityUser> \_userManager;

        private readonly JwtConfig \_jwtConfig;

        public AuthManagementController(UserManager<IdentityUser> userManager, IOptionsMonitor<JwtConfig> optionMonitor)

        {

            \_userManager = userManager;

            \_jwtConfig = optionMonitor.CurrentValue;

        }

        [HttpPost]

        [Route("Register")]

        public async Task<IActionResult> Register([FromBody] UserRegistationDto user)

        {

            if(ModelState.IsValid)

            {

                var existingUser = await \_userManager.FindByEmailAsync(user.Email);

                if(existingUser != null)

                {

                    return BadRequest(new RegistrationResponse(){

                        Errors = new List<string>(){

                            "Email already in use"

                        },

                        Success = false

                    });

                }

                var newUser = new IdentityUser() { Email = user.Email, UserName = user.Username};

                var isCreated = await \_userManager.CreateAsync(newUser, user.Password);

                if(isCreated.Succeeded)

                {

                    var jwtToken = GenerateJwtToken(newUser);

                    return Ok(new RegistrationResponse(){

                        Success = true,

                        Token = jwtToken

                    });

                } else

                {

                    return BadRequest(new RegistrationResponse(){

                        Errors = isCreated.Errors.Select(x=> x.Description).ToList(),

                        Success = false

                    });

                }

            }

            return BadRequest(new RegistrationResponse(){

                    Errors = new List<string>(){"Invalid Payload"},

                    Success = false

                });

        }

        [HttpPost]

        [Route("Login")]

        public async Task<IActionResult> Login([FromBody] UserLoginRequest user)

        {

            if(ModelState.IsValid)

            {

                var existingUser = await \_userManager.FindByEmailAsync(user.Email);

                if(existingUser == null)

                {

                    return BadRequest(new RegistrationResponse(){

                        Errors = new List<string>(){

                            $"Invalid login request"

                        },

                        Success = false

                    });

                }

                var isCorrect = await \_userManager.CheckPasswordAsync(existingUser, user.Password);

                if(!isCorrect)

                {

                    return BadRequest(new RegistrationResponse(){

                        Errors = new List<string>(){

                            "Invalid login request 2"

                        },

                        Success = false

                    });

                }

                var jwtToken = GenerateJwtToken(existingUser);

                return Ok(new RegistrationResponse(){

                        Success = true,

                        Token = jwtToken

                    });

            }

            return BadRequest(new RegistrationResponse(){

                    Errors = new List<string>(){"Invalid Payload"},

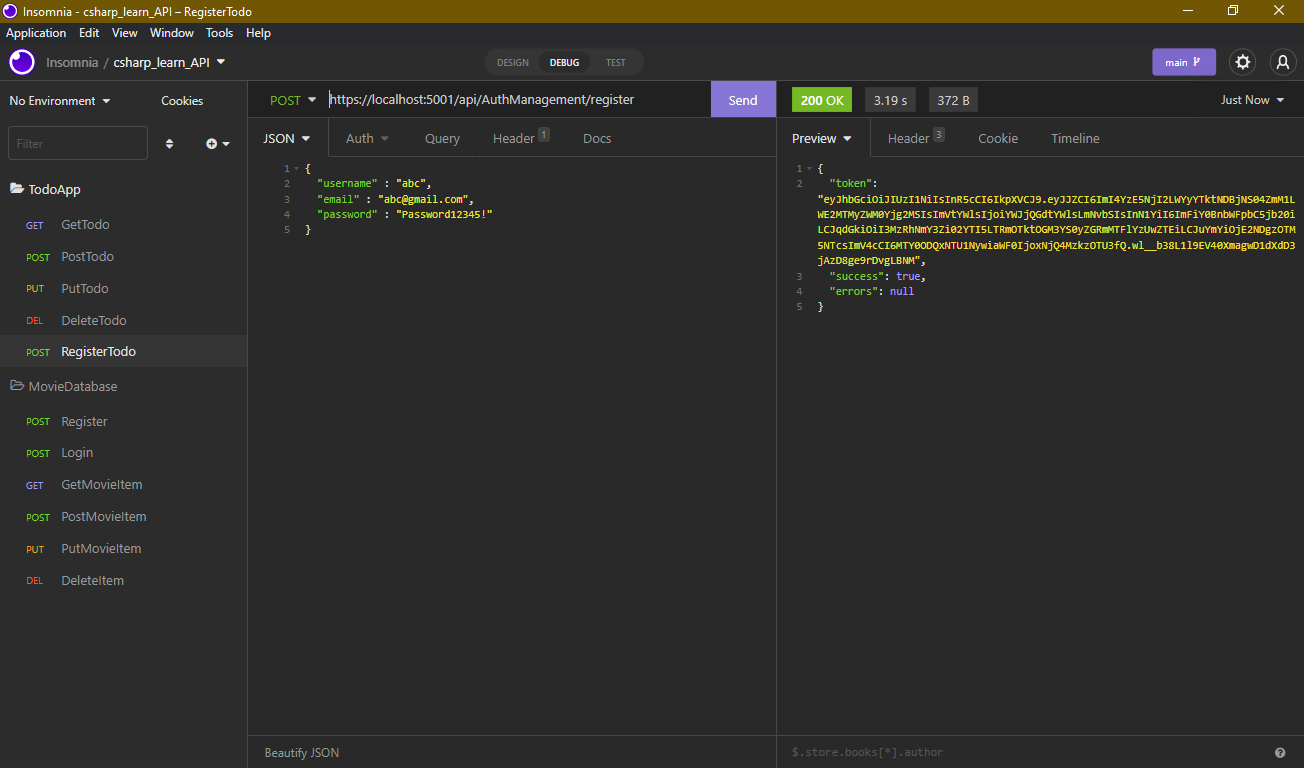
                    Success = false

                });

        }

}

1. Testing Register



1. Penambahan Authentication dengan JWT III : Login
2. Pada ‘TodoApp>Models>DTOs>Request’ tambahkan file ‘UserLoginRequest.cs’

UserLogin.cs:

using System.ComponentModel.DataAnnotations;

namespace TodoApp.Models.DTOs.Requests

{

    public class UserLoginRequest

    {

        [Required]

        [EmailAddress]

        public string Email {get;set;}

        [Required]

        public string Password {get;set;}

    }

}

1. Pada ‘TodoApp>Controller>AuthManagementController’ Tambahkan pada bagian bawah register:

[HttpPost]

        [Route("Login")]

        public async Task<IActionResult> Login([FromBody] UserLoginRequest user)

        {

            if(ModelState.IsValid)

            {

                var existingUser = await \_userManager.FindByEmailAsync(user.Email);

                if(existingUser == null)

                {

                    return BadRequest(new RegistrationResponse(){

                        Errors = new List<string>(){

                            $"Invalid login request"

                        },

                        Success = false

                    });

                }

                var isCorrect = await \_userManager.CheckPasswordAsync(existingUser, user.Password);

                if(!isCorrect)

                {

                    return BadRequest(new RegistrationResponse(){

                        Errors = new List<string>(){

                            "Invalid login request 2"

                        },

                        Success = false

                    });

                }

                var jwtToken = GenerateJwtToken(existingUser);

                return Ok(new RegistrationResponse(){

                        Success = true,

                        Token = jwtToken

                    });

            }

            return BadRequest(new RegistrationResponse(){

                    Errors = new List<string>(){"Invalid Payload"},

                    Success = false

                });

        }

        private string GenerateJwtToken(IdentityUser user)

        {

            var jwtTokenHandler = new JwtSecurityTokenHandler();

            var key = Encoding.ASCII.GetBytes(\_jwtConfig.Secret);

            var tokenDescriptor = new SecurityTokenDescriptor

            {

                Subject = new ClaimsIdentity( new []

                {

                    new Claim("Id", user.Id),

                    new Claim(JwtRegisteredClaimNames.Email, user.Email),

                    new Claim(JwtRegisteredClaimNames.Sub, user.Email),

                    new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())

                }),

                Expires = DateTime.UtcNow.AddHours(6),

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

            };

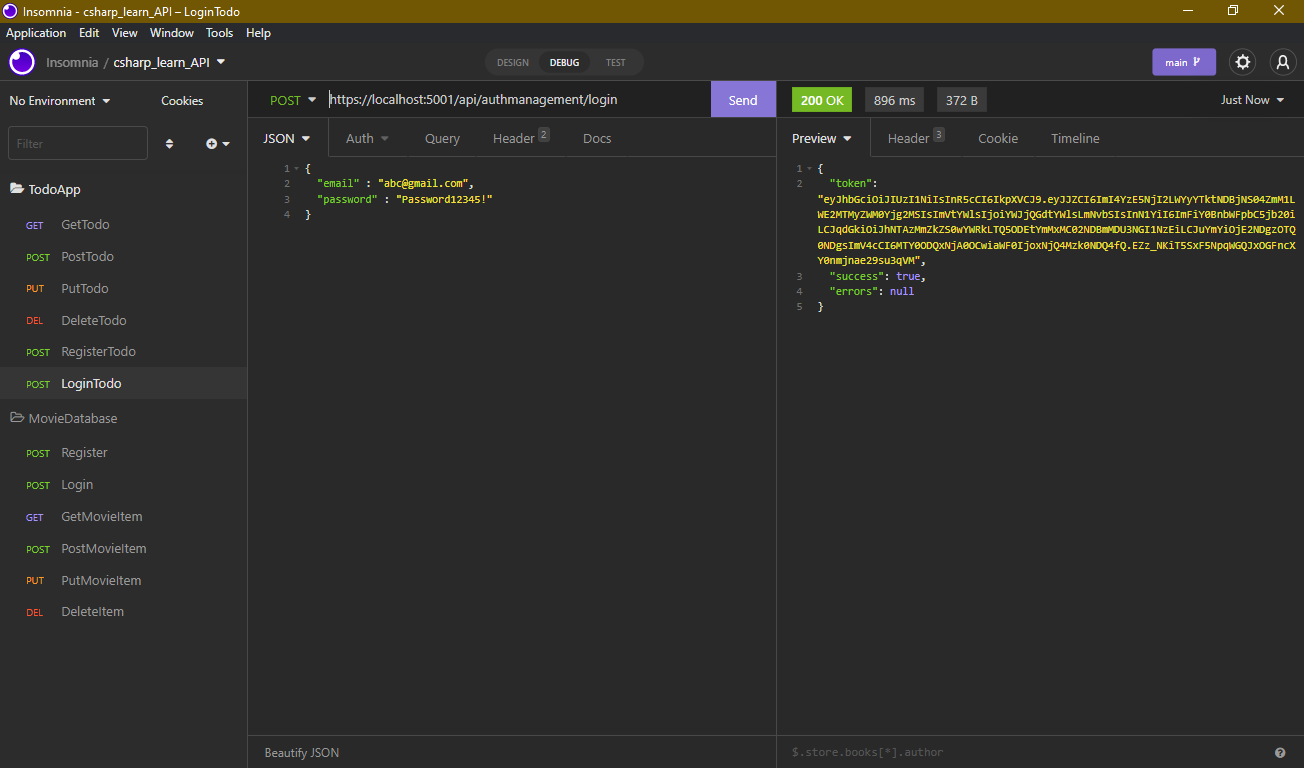
            var token = jwtTokenHandler.CreateToken(tokenDescriptor);

            var jwtToken = jwtTokenHandler.WriteToken(token);

            return jwtToken;

        }

1. Testing Login



1. Pada ‘TodoApp>Controller>TodoController.cs’ dibagian atas (include) tambahkan:

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

1. Pada ‘TodoApp>Controller>TodoController.cs’ di bawah

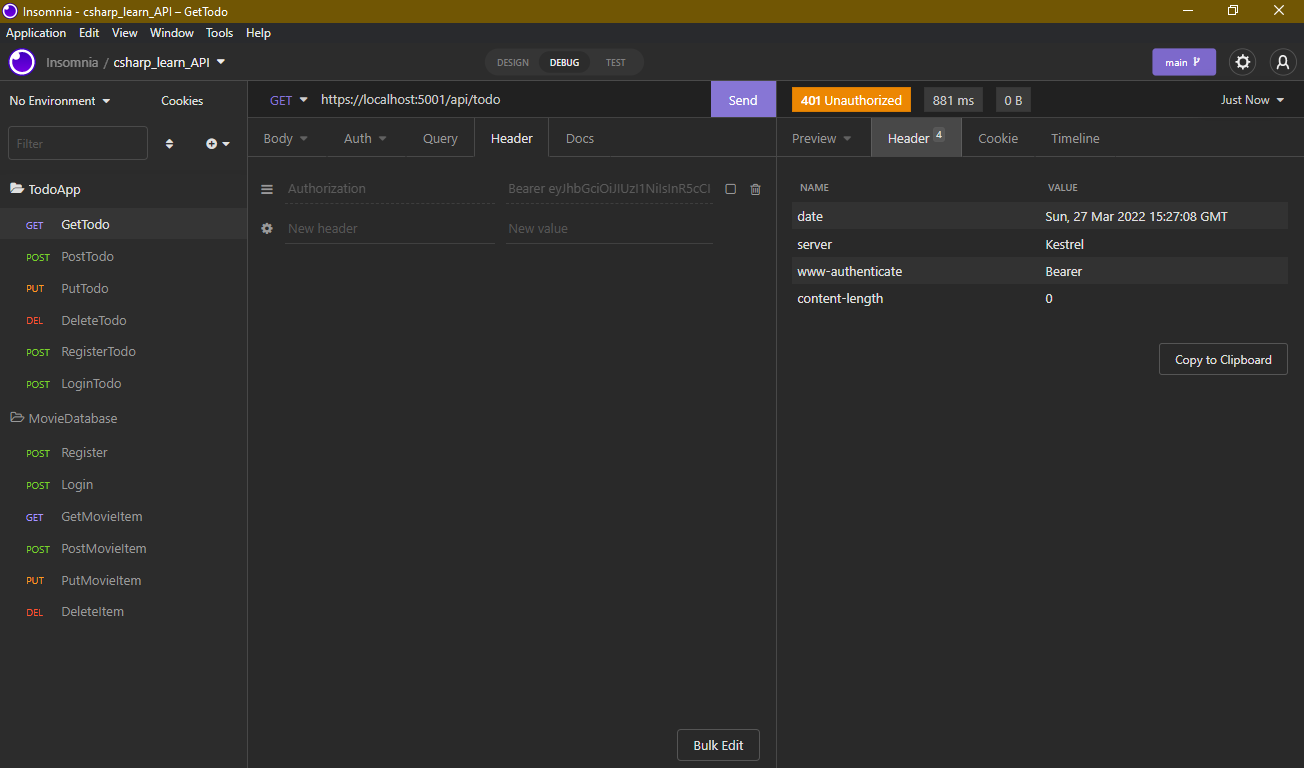
[ApiController]

tambahkan:

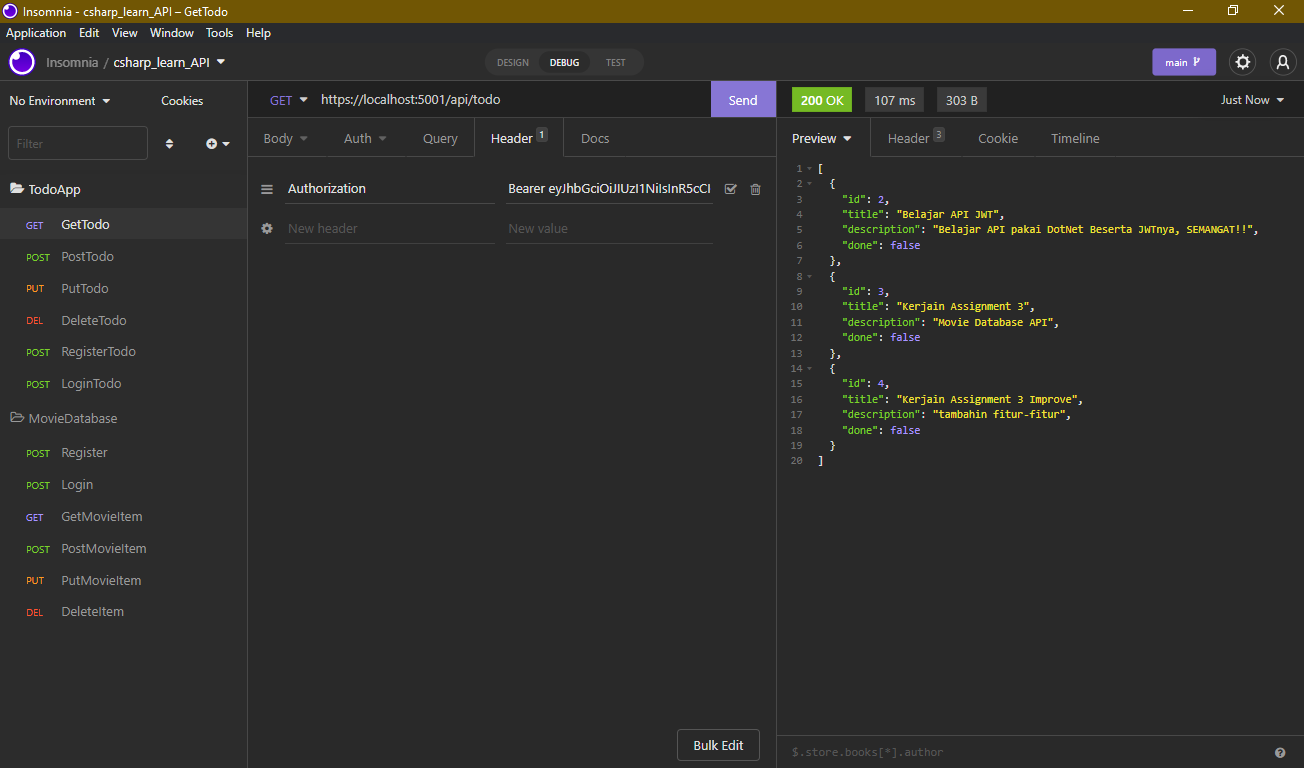
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

1. Hasil testing

Saat tanpa token:



Memakai token:



Best Practice “Create Web API: RESTFUL API”

ASSIGNMENT 3 – Movie Database

Membuat Web API Movie Database dengan melakukan setiap langkah yang sama dengan Langka pembuatan TodoApp diatas namun dilakukan penggantian nama:

1. Project: MovieDatabase

dotnet new webapi -n “MovieDatabase” -lang “C#” -au none

1. Models:

* ItemData.cs

using System;

namespace MovieDatabaseApi.Models

{

    public class ItemData

    {

        public int Id {get; set;}

        public string Name {get; set;}

        public string Genre {get; set;}

        public string Duration {get; set;}

        public DateTime ReleaseDate {get; set;}

    }

}

1. Data:

* ApiDBContext.cs

using Microsoft.EntityFrameworkCore;

using TodoApp.Models;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.AspNetCore.Identity;

namespace TodoApp.Data

{

    public class ApiDbContext : IdentityDbContext

    {

        public virtual DbSet<ItemData> Items {get;set;}

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

        {

        }

    }

}

1. Drop Database

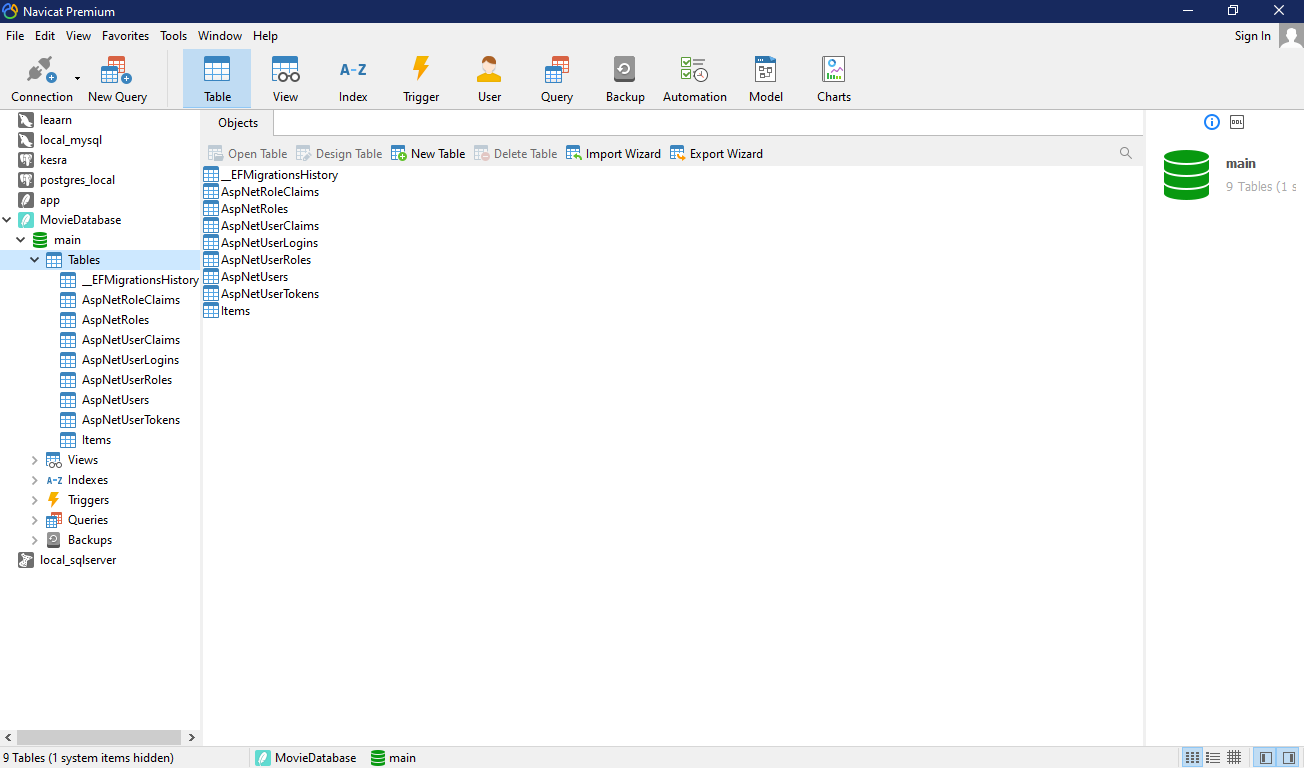
* dotnet ef database drop

1. Remove Migration and re add migrations

* dotnet ef migrations remove
* dotnet ef migrations add "Initial Migrations"

1. Update database

* dotnet ef database update



1. Controller:

* MovieItemController.cs, dengan syntax yang telah disesuaikan pada bagian PUT, karena ada perbedaan field data:

        [HttpPut("{id}")]

        public async Task<IActionResult> UpdateItem(int id, ItemData item)

        {

            if(id!=item.Id){

                return BadRequest();

            }

            var existItem = await \_context.Items.FirstOrDefaultAsync(x=> x.Id == id);

            if(existItem == null)

                return NotFound();

            existItem.Name = item.Name;

            existItem.Genre = item.Genre;

            existItem.Duration = item.Duration;

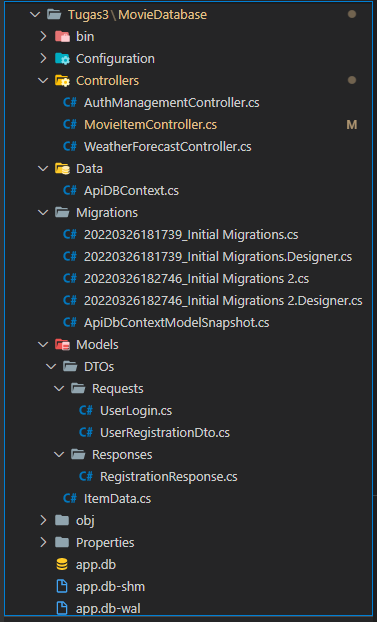
            existItem.ReleaseDate = item.ReleaseDate;

            await \_context.SaveChangesAsync();

            // return NoContent(existItem);

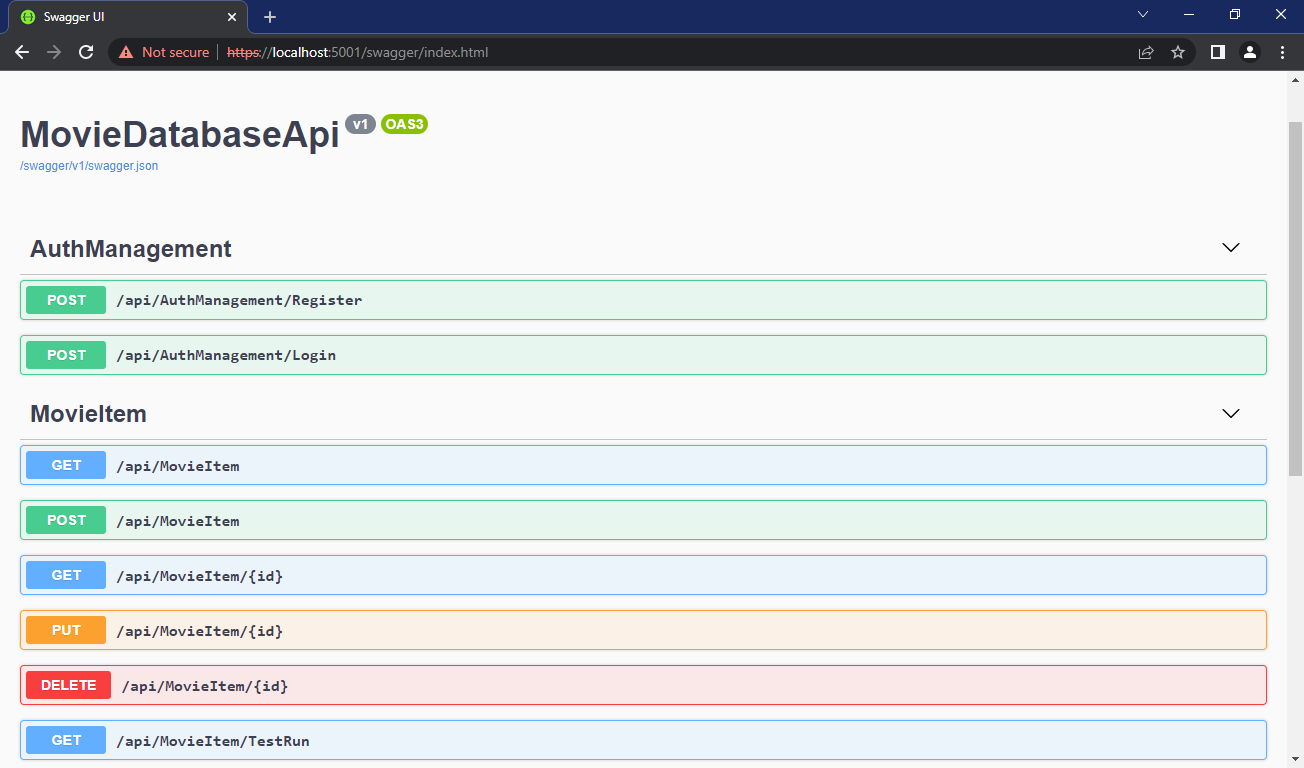
            return Ok(existItem);

        }

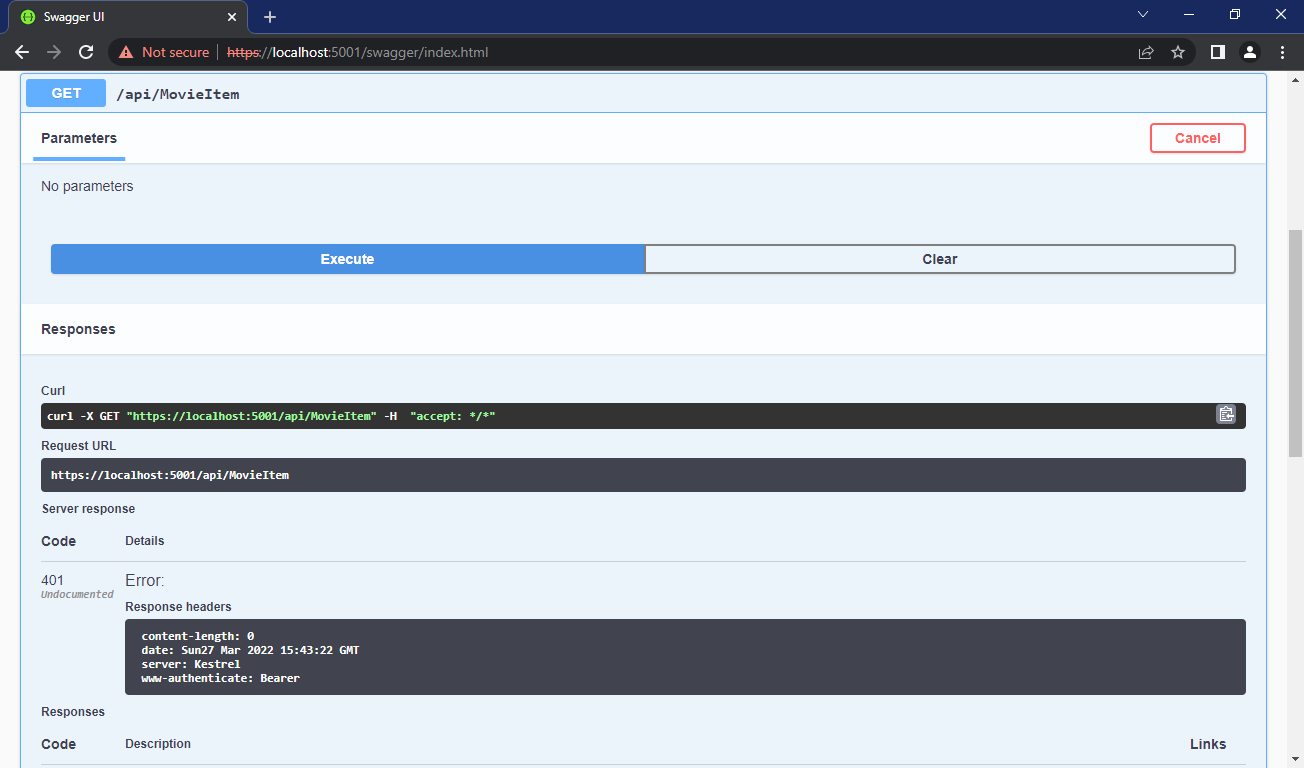


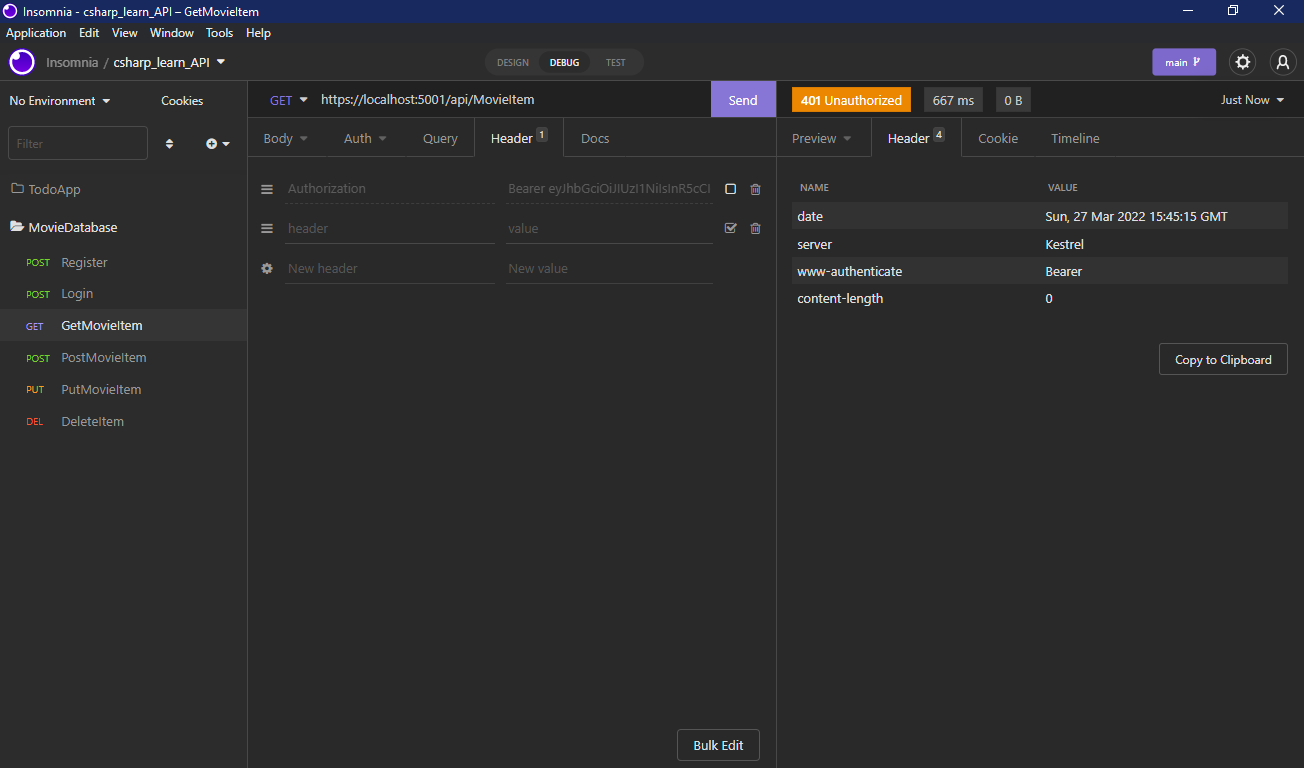
1. Hasil Running:

* Running Awal memakai swagger

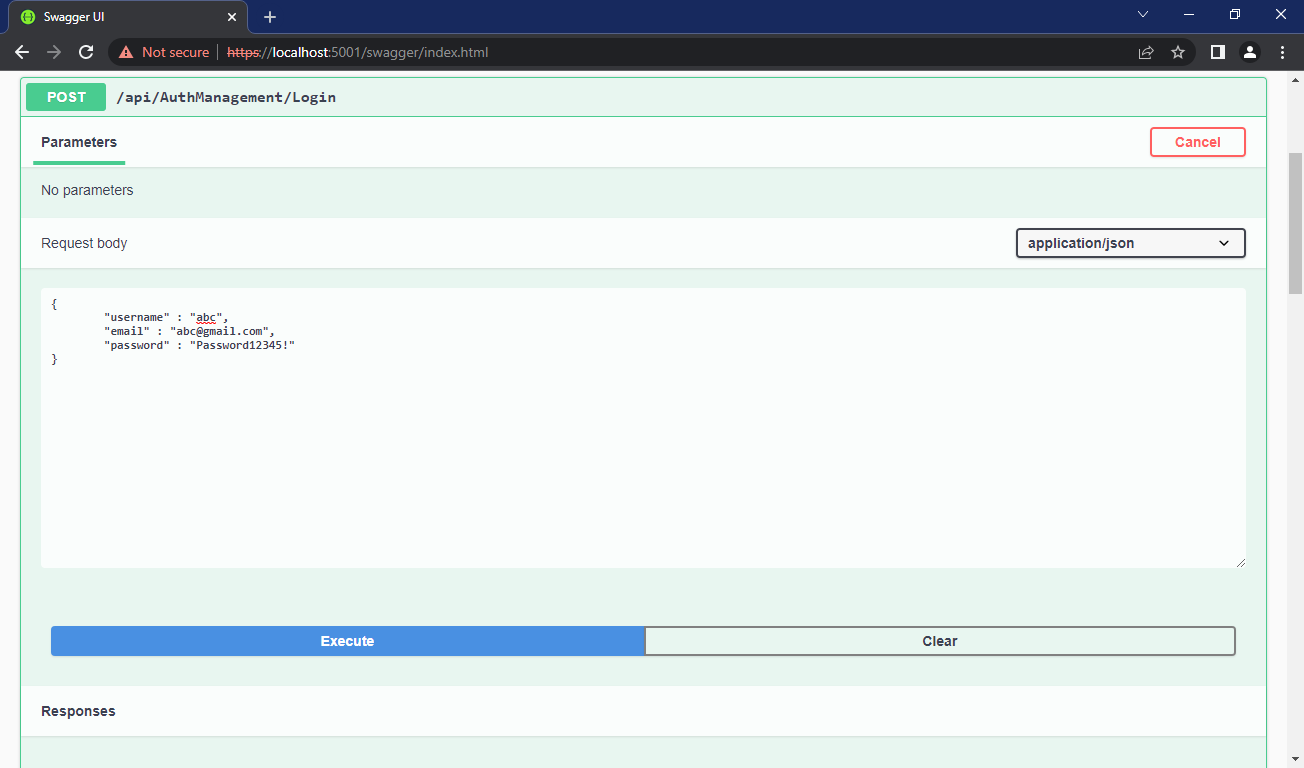


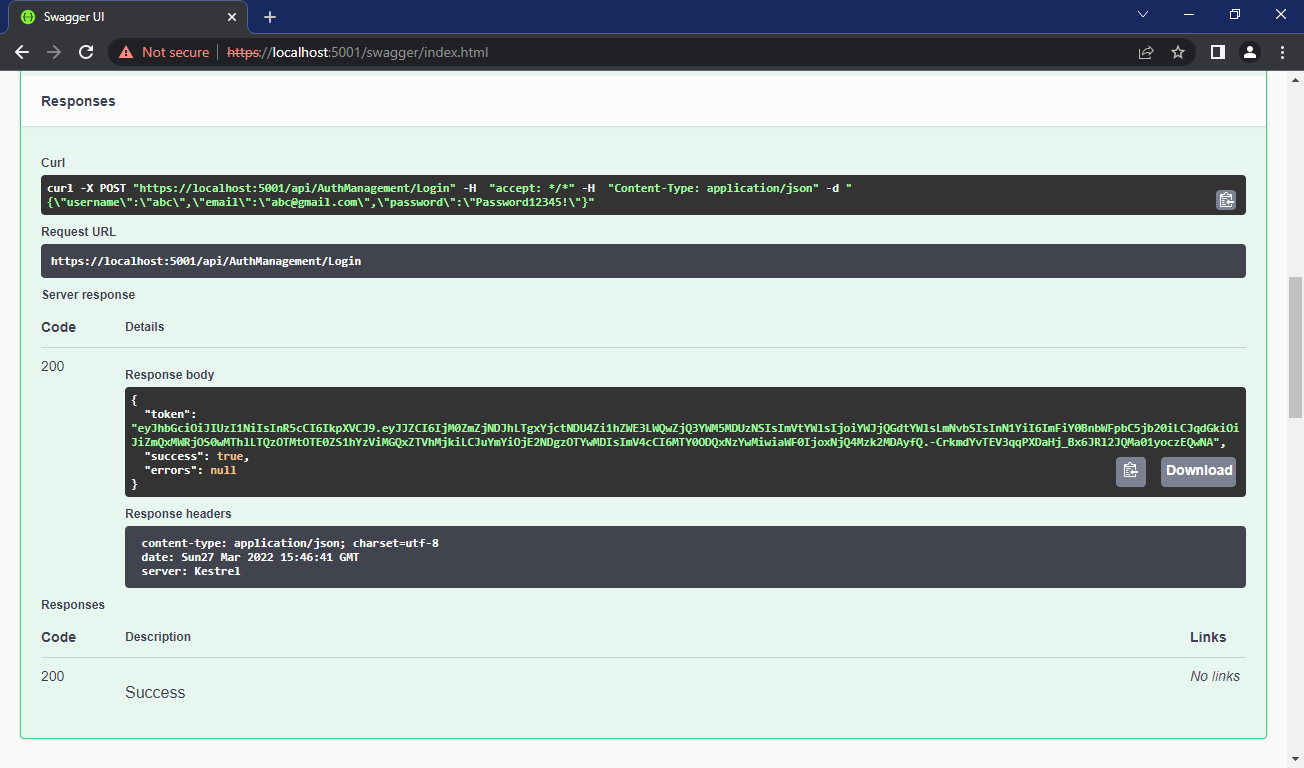
* GET tanpa token

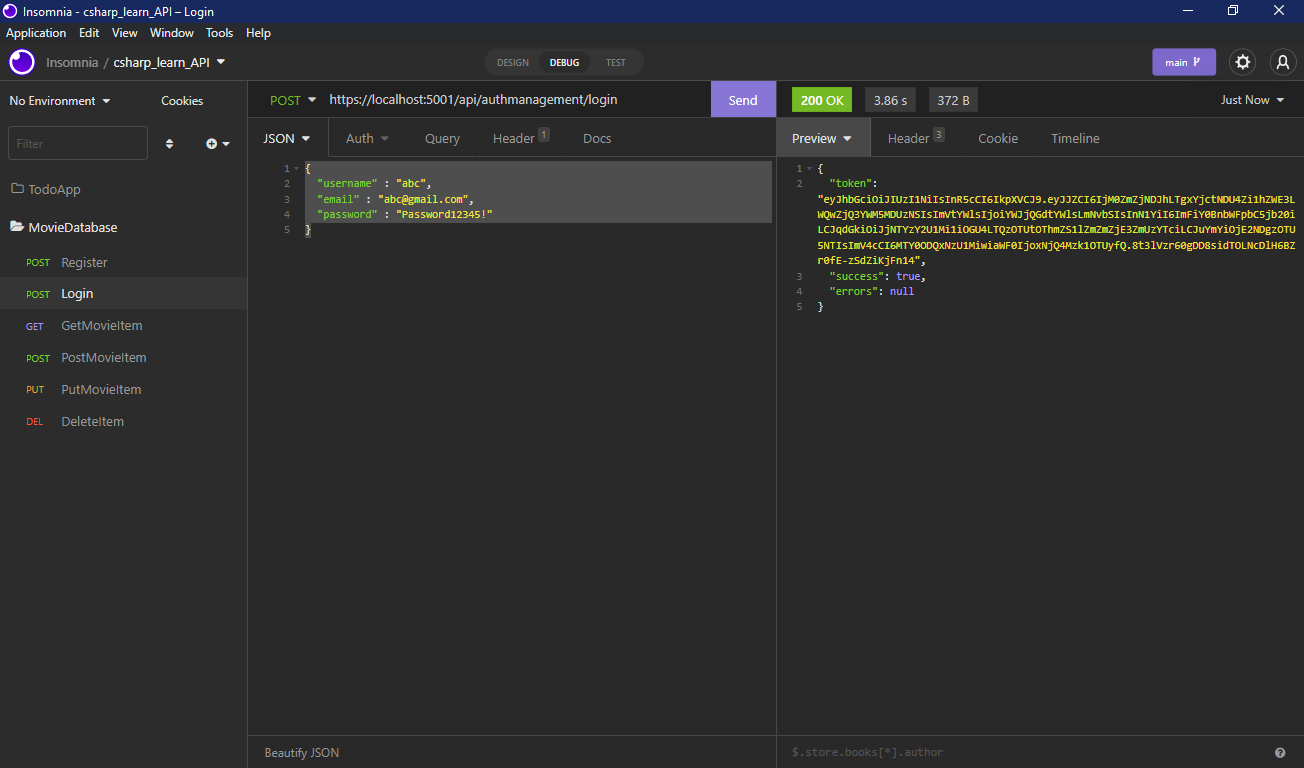




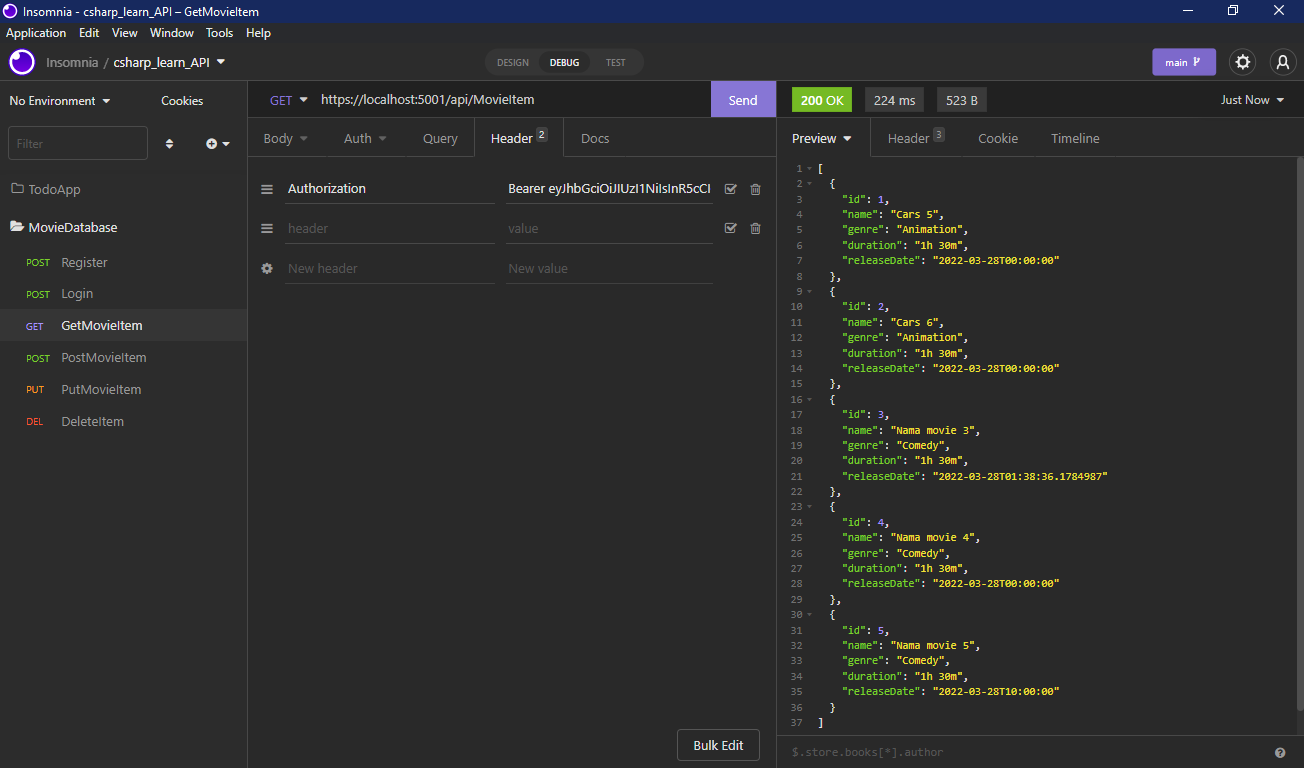
* Login







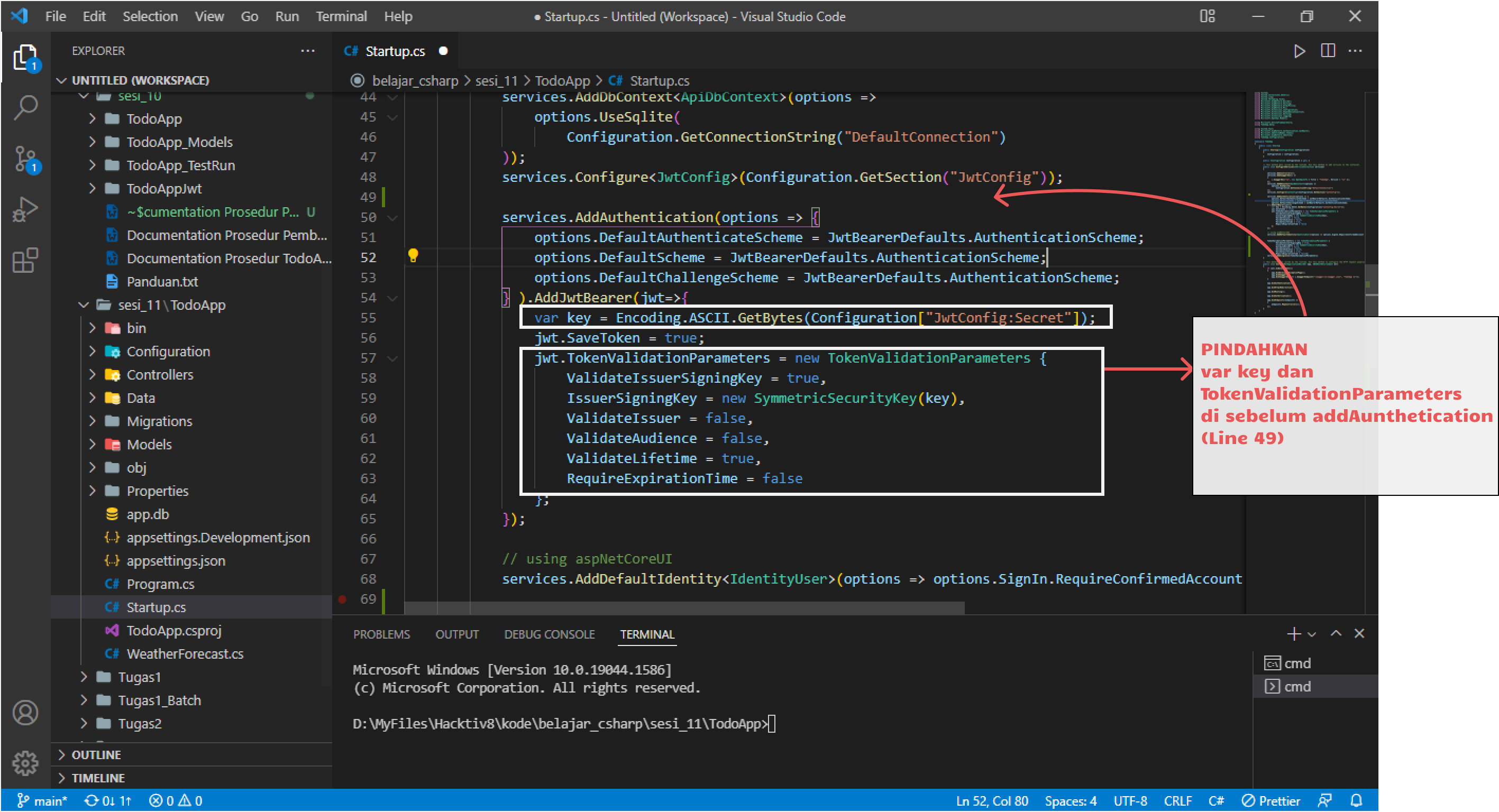
* GET dengan token



Best Practice “Create Web API: RESTFUL API”

SESI 11 – Refresh JWT Token

1. Penambahan TokenValidationParameters
2. **Startup.cs**
3. Buka ‘TodoApp/Startup.cs’.
4. Pindahkan var Key dan TokenValidationParameters keluar dari Authentichation



1. Selain itu Didalam parameter TokenvalidationParameters

Tambahkan:

ClockSkew = TimeSpan.Zero

1. Lalu setelah TokenValidationParameter tambahkan Pula

services.AddSingleton(TokenValidationParameters);

1. Pada AddJwtBearer dibagian jwt.TokenValidationParameters isikan dengan TokenValidationParameters:

jwt.TokenValidationParameters = TokenValidationParameters;

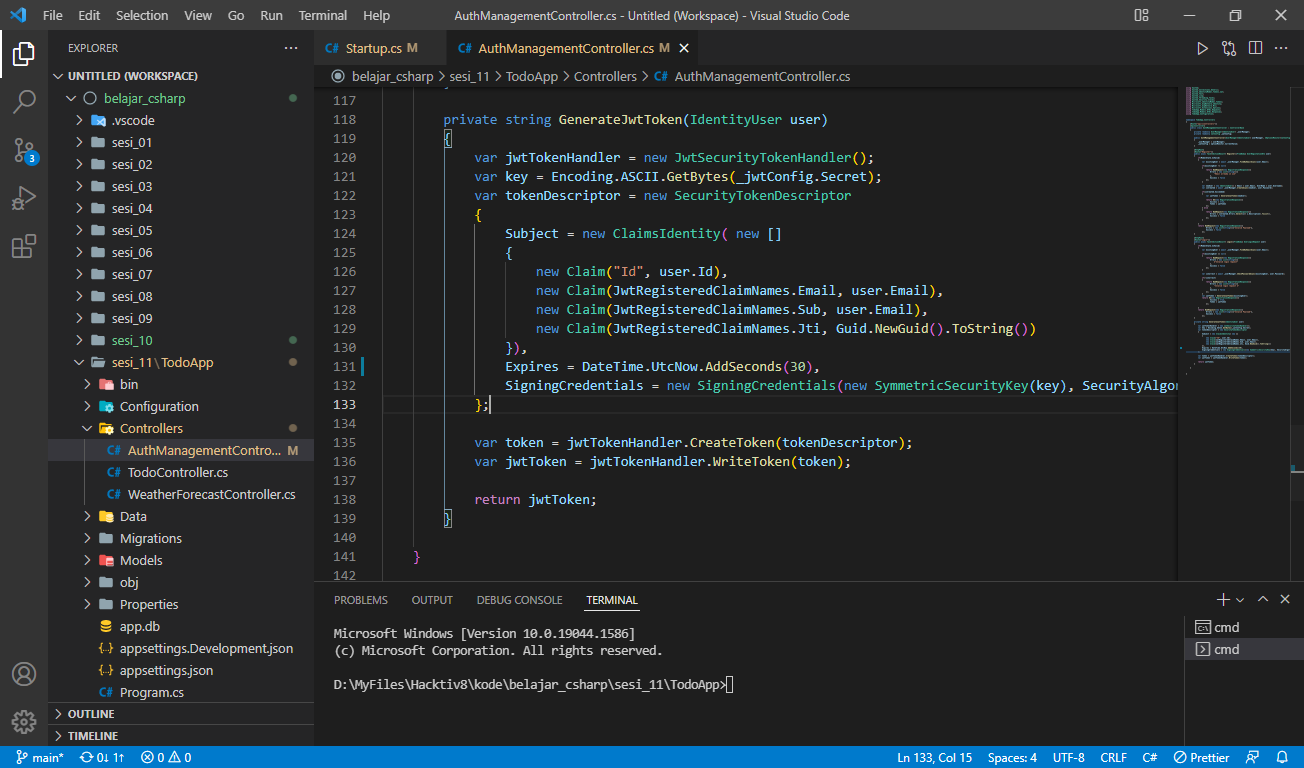


1. **‘TodoApp/Controllers/AuthManagementController.cs’**
2. Pada Method GenerateJwtToken di bagian var tokenDescriptor.Expires, ubah menjadi seperti ini:

Expires = DateTime.UtcNow.AddSecond(30),

[Hal ini membuat **Token** akan **Expire** dalam **30 detik** setelah di generate]

1. Seperti ini



1. **‘TodoApp/Configuration/AuthResult.cs’**
2. Tambahkan propertiesstring RefreshTokenpada ObjectAuthResult,

public class AuthResult

    {

        public string Token {get; set;}

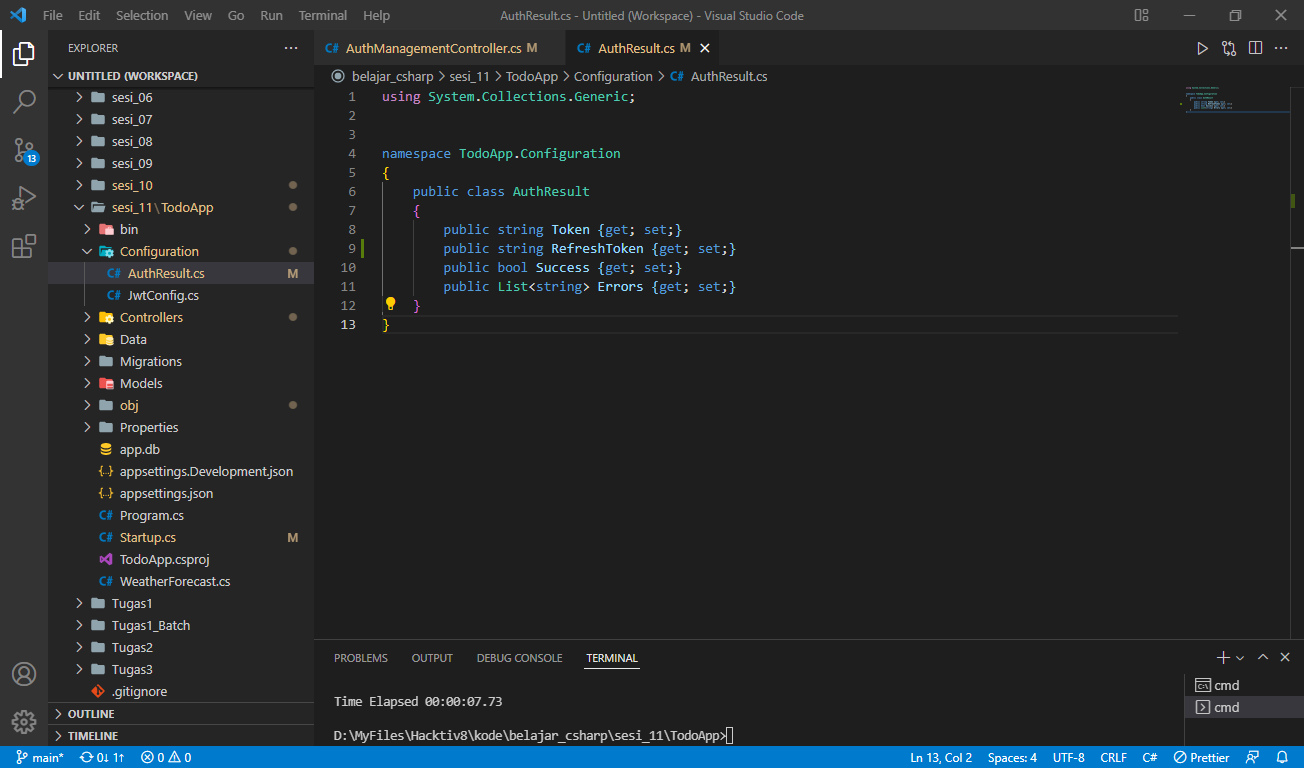
        public string RefreshToken {get; set;}

        public bool Success {get; set;}

        public List<string> Errors {get; set;}

    }

1. maka akan jadi seperti ini:



1. **‘TodoApp/Models’**
2. Tambahkan File RefrehToken.cs, dan isikan kode:

using System;

using System.ComponentModel.DataAnnotations.Schema;

using Microsoft.AspNetCore.Identity;

namespace TodoApp.Models

{

    public class RefreshToken

    {

        public int Id {get; set;}

        public string UserId {get; set;}

        public string Token {get; set;}

        public string JwtId {get; set;}

        public bool IsUsed {get; set;}

        public bool IsRevoked {get; set;}

        public DateTime AddedDate{get; set;}

        public DateTime ExpiryDate{get; set;}

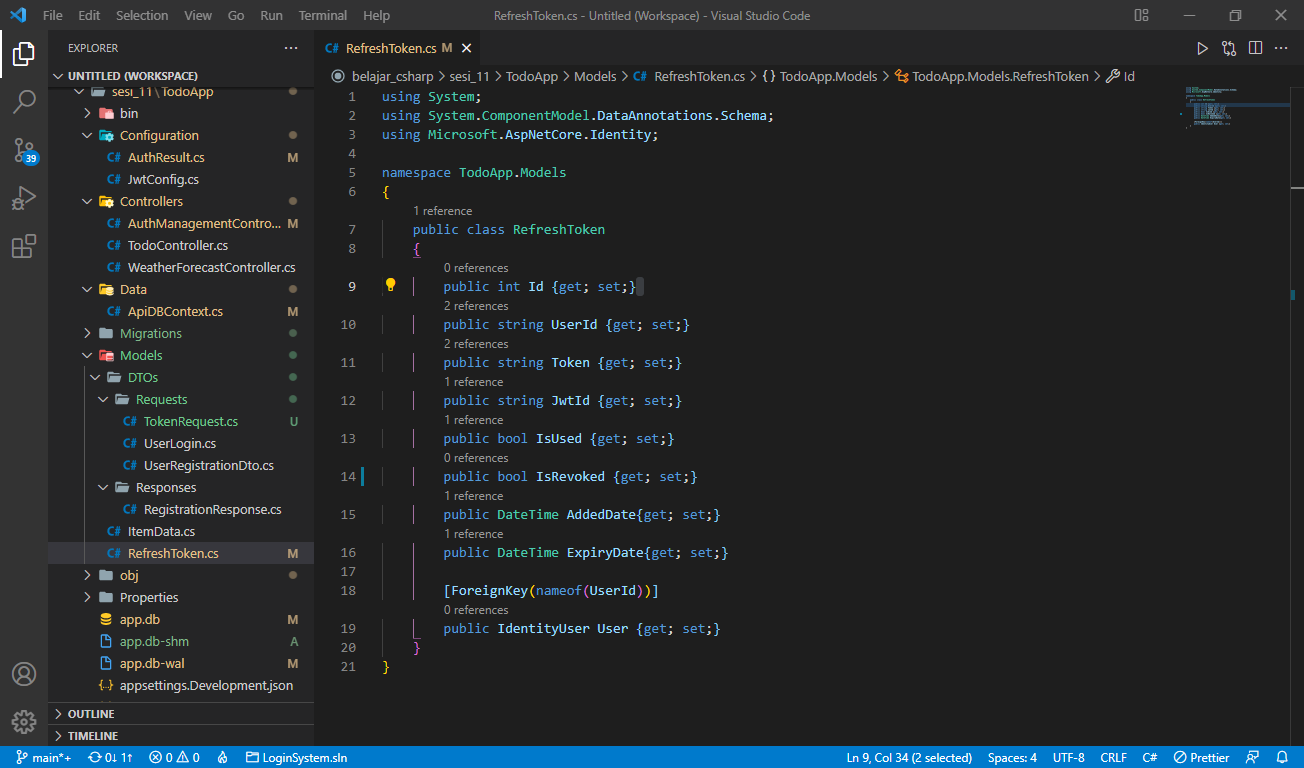
        [ForeignKey(nameof(UserId))]

        public IdentityUser User {get; set;}

    }

}

1. Hasil File seperti ini



**Note:**

using System.ComponentModel.DataAnnotations.Schema;

digunakan untuk

[ForeignKey(nameof(UserId))]

1. **Pada ‘TodoApp/Models/DTO/Request’ tambahkan file TokenRequest.cs dengan isi file:**

using System.ComponentModel.DataAnnotations;

namespace TodoApp.Models.DTOs.Requests

{

    public class TokenRequest

    {

        [Required]

        public string Token {get;set;}

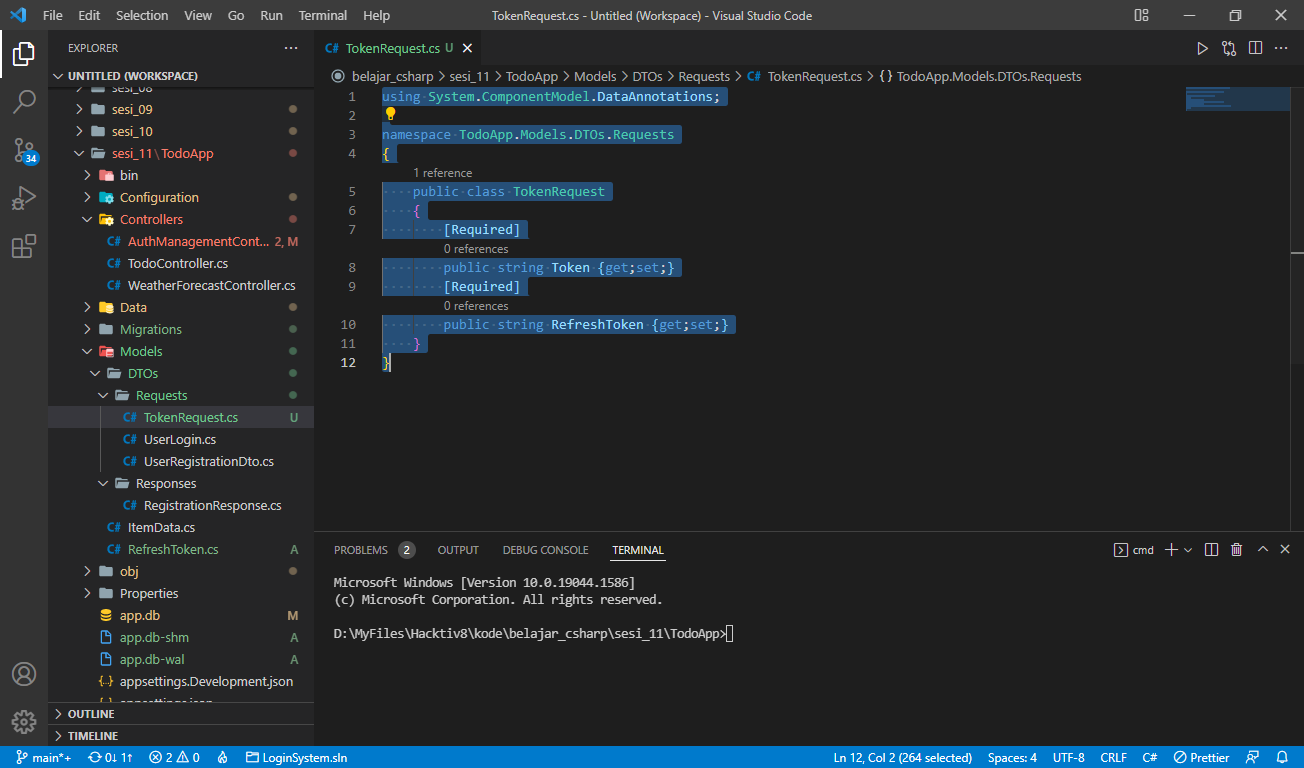
        [Required]

        public string RefreshToken {get;set;}

    }

}

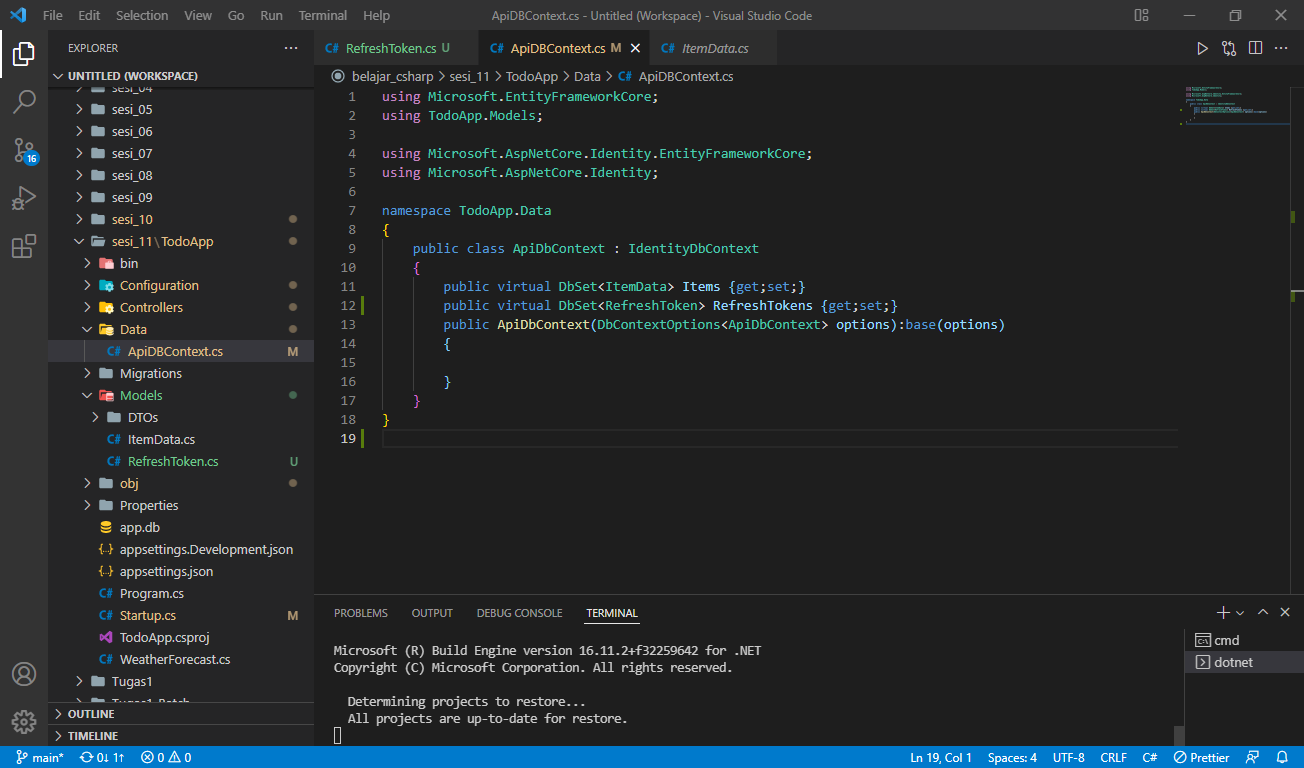
**Seperti ini:**



1. **‘TodoApp/Data/ApiDbContext.cs’**
2. public class ApiDbContext : IdentityDbContext
3. Pada file ‘**ApiDbContext.cs’, bagian** class ApiDbContext : IdentityDbContext Tambahkan:

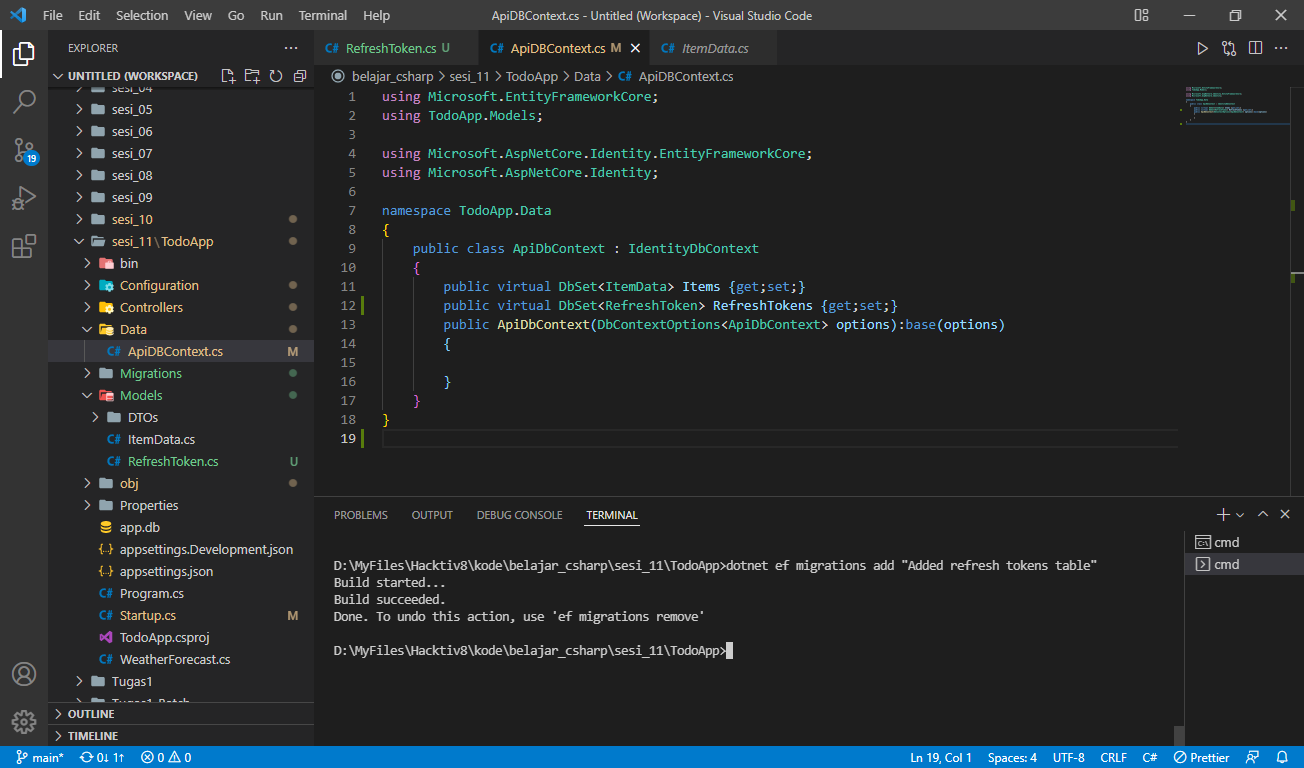
public virtual DbSet<RefreshToken> RefreshTokens {get;set;}

1. Akan jadi seperti ini:



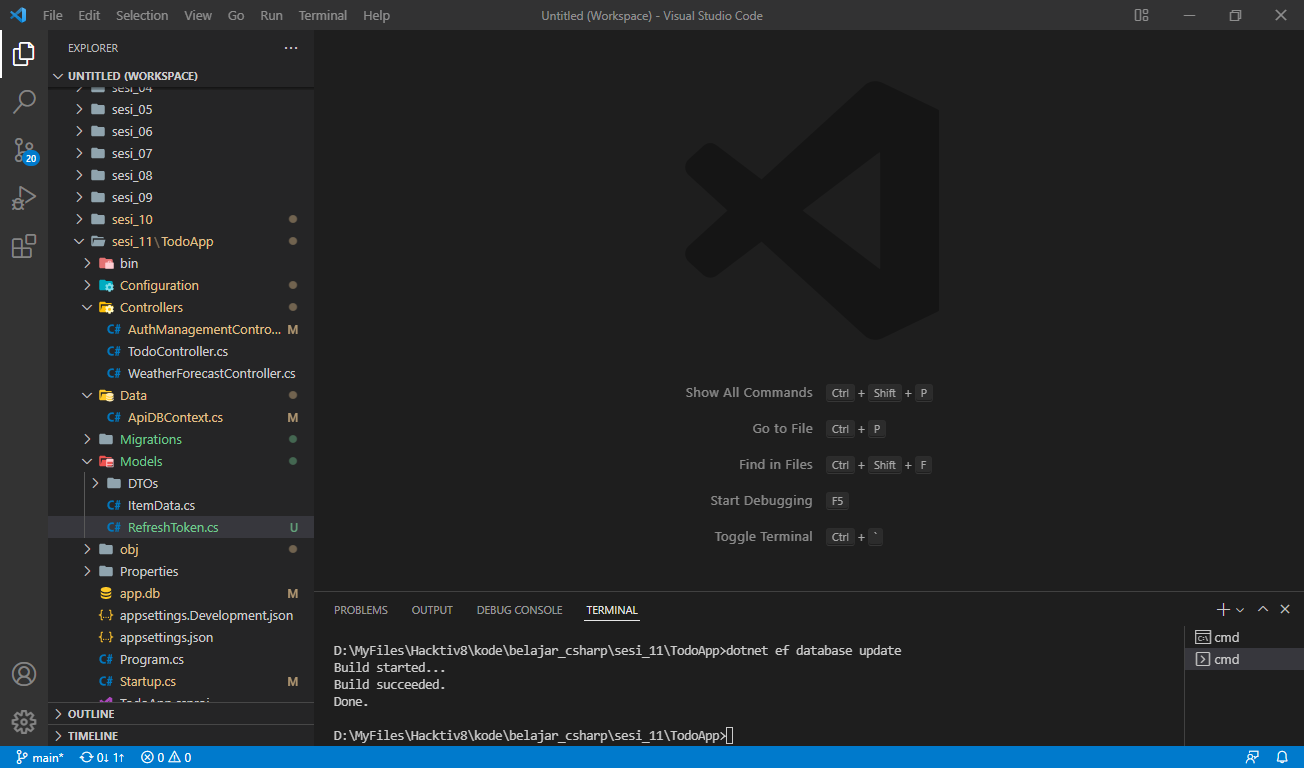
1. **Add Migrations and Update Database**
2. Add migration dengan menjalakan command pada terminal

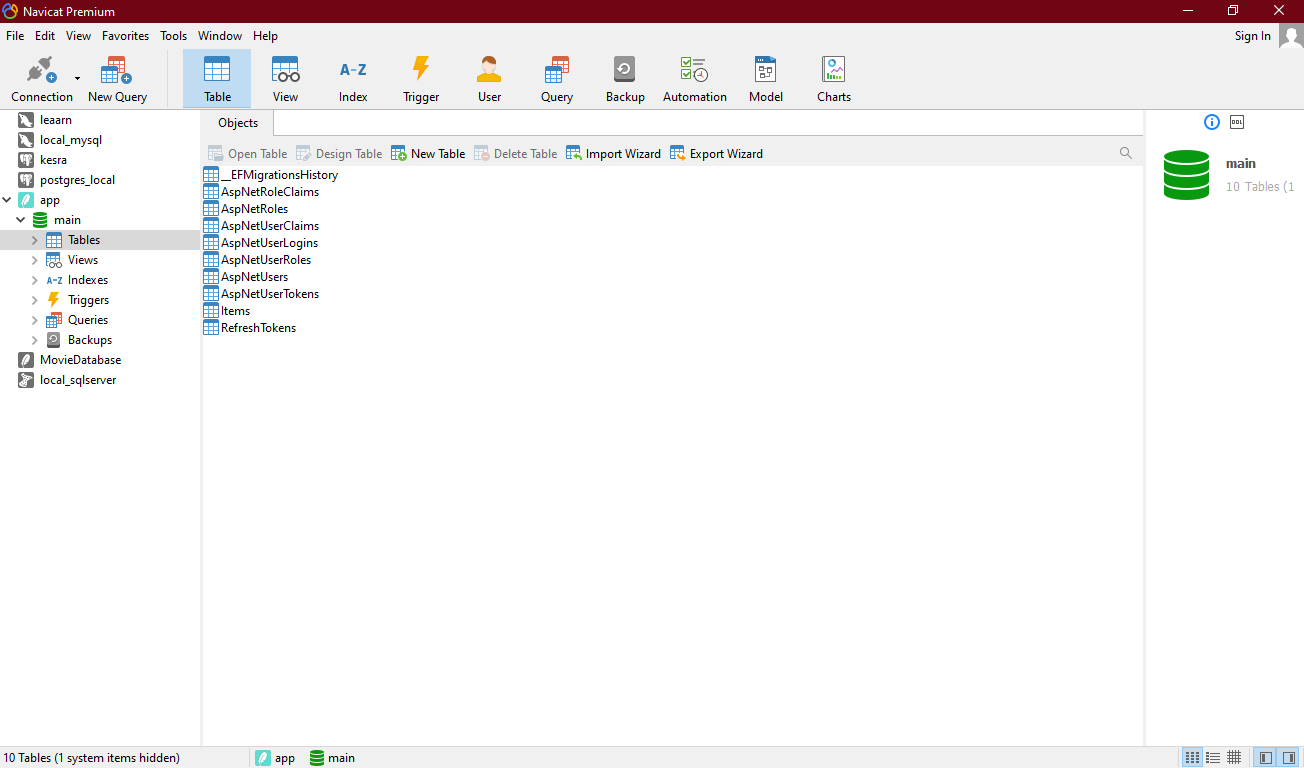
dotnet ef migrations add "Added refresh tokens table"



1. Update Database dengan menjalankan command pada terminal

dotnet ef database update



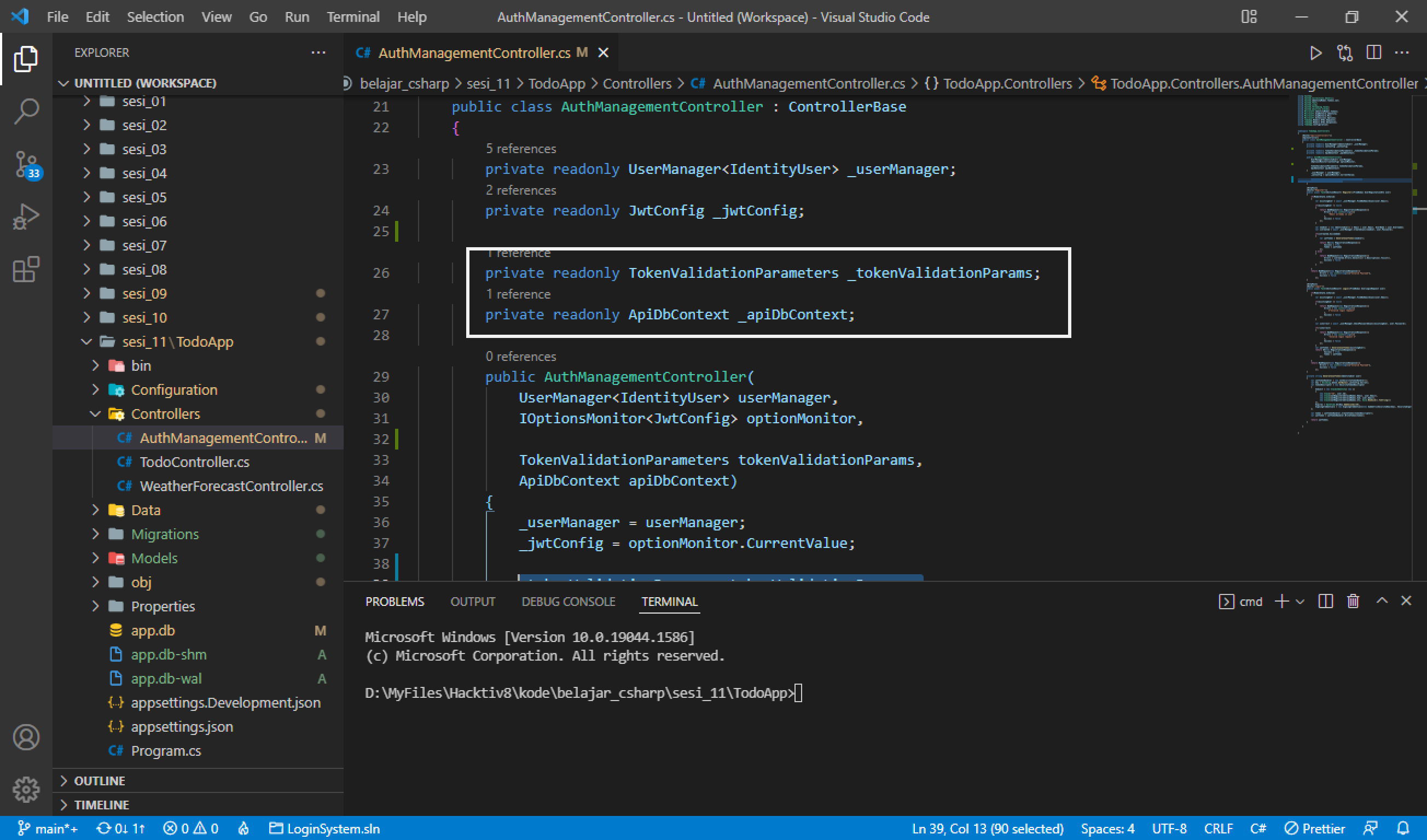


1. **Membuat EndPoint Baru ‘RefreshToken’ Pada ‘TodoApp/Controllers/AuthManagementControllers’**
2. Tambahkan 2 variable private readonly baru pada class AuthManagementController : ControllerBase

private readonly TokenValidationParameters \_tokenValidationParams;

private readonly ApiDbContext \_apiDbContext;

seperti ini



1. Pada public AuthManagementController, tambahkan 2 parameter baru

TokenValidationParameters tokenValidationParams,

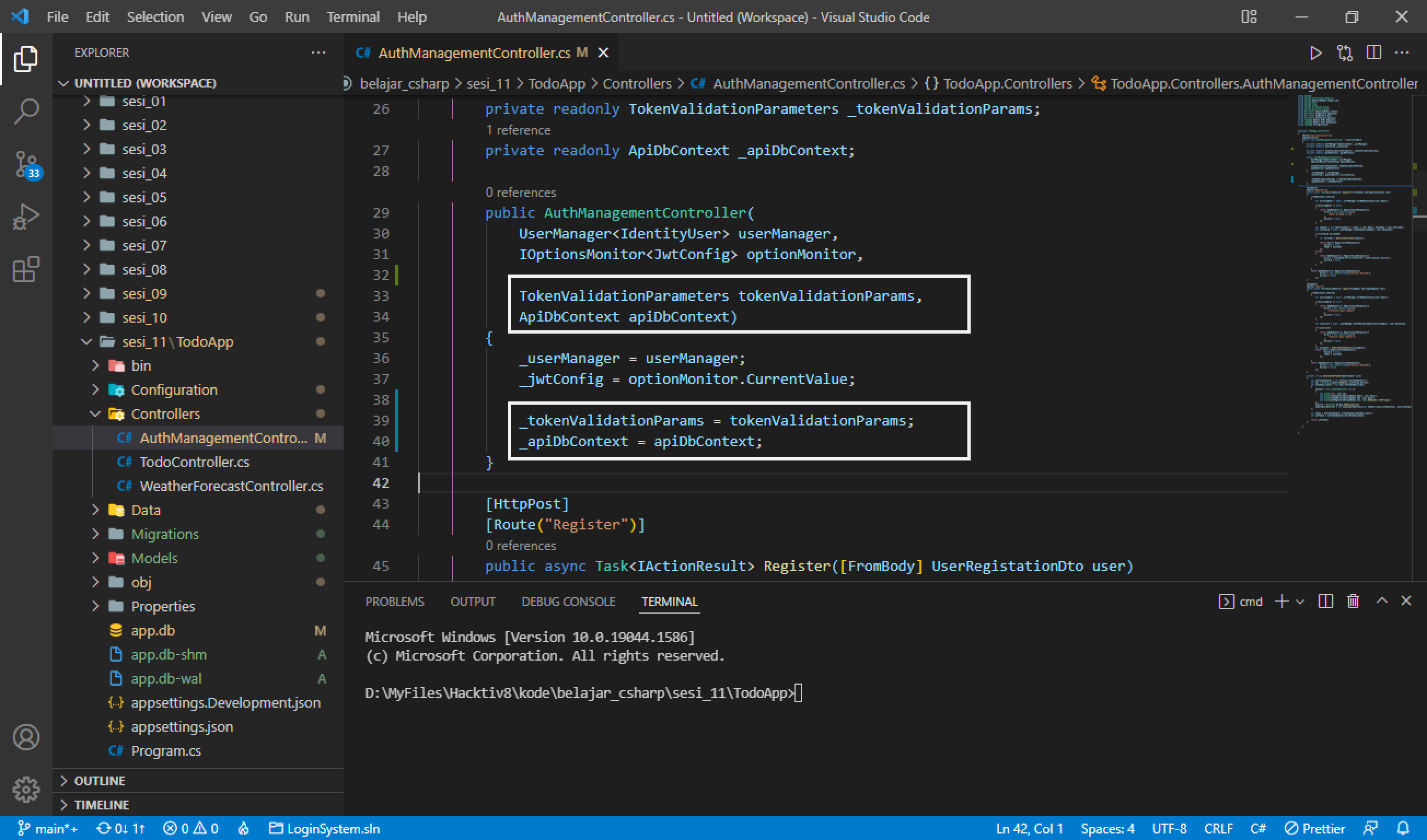
ApiDbContext apiDbContext

Lalu tambahkan di dalam method:

\_tokenValidationParams = tokenValidationParams;

\_apiDbContext = apiDbContext;

Seperti ini:

****

1. Pada **‘AuthManagementController.cs’**, Tambahkan method baru RandomString seperti ini:

private string RandomString(int length)

        {

            var random = new Random();

            var chars = "ABCDEFGHIJKLMNOPGRSTUVWXY0Z123456789";

            return new string(Enumerable.Repeat(chars, length).

                Select(x=> x[random.Next(x.Length).ToArray()]));

        }

1. Pada mehtod GenerateJwtToken tambahkan variable refreshToken disetelah variable jwtToken

var refreshToken = new RefreshToken()

            {

                JwtId = token.Id,

                IsUsed = false,

                IsRevoked = false,

                UserId = user.Id,

                AddedDate = DateTime.UtcNow,

                ExpiryDate = DateTime.UtcNow.AddHours(4),

                Token = RandomString(35) + Guid.NewGuid()

            };

Tambahkan pula:

await \_apiDbContext.RefreshToken.AddAsync(refreshToken);

await \_apiDbContext.SaveChangeAsync();

untuk penambahan dan penyimpana pada database Tabel RefreshToken,

dan ubah nilai return menjadi:

return new AuthResult(){

Token = jwtToken,

Success = true,

refreshToken = refreshToken.Token

};

**Isi Generate:**

private async Task<AuthResult> GenerateJwtToken(IdentityUser user)

        {

            var jwtTokenHandler = new JwtSecurityTokenHandler();

            var key = Encoding.ASCII.GetBytes(\_jwtConfig.Secret);

            var tokenDescriptor = new SecurityTokenDescriptor

            {

                Subject = new ClaimsIdentity( new []

                {

                    new Claim("Id", user.Id),

                    new Claim(JwtRegisteredClaimNames.Email, user.Email),

                    new Claim(JwtRegisteredClaimNames.Sub, user.Email),

                    new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())

                }),

                Expires = DateTime.UtcNow.AddSeconds(30),

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

            };

            var token = jwtTokenHandler.CreateToken(tokenDescriptor);

            var jwtToken = jwtTokenHandler.WriteToken(token);

            var refreshToken = new RefreshToken()

            {

                JwtId = token.Id,

                IsUsed = false,

                isRevoked = false,

                UserId = user.Id,

                AddedDate = DateTime.UtcNow,

                ExpiryDate = DateTime.UtcNow.AddHours(4),

                Token = RandomString(35) + Guid.NewGuid()

            };

            await \_apiDbContext.RefreshTokens.AddAsync(refreshToken);

            await \_apiDbContext.SaveChangesAsync();

            return new AuthResult(){

                Token = jwtToken,

                Success = true,

                RefreshToken = refreshToken.Token

            };

        }

1. Pada MethodLogin dan Register ganti isi jwtToken dan return menjadi:

**Login**

var jwtToken = await GenerateJwtToken(existingUser);

**Register**

var jwtToken = await GenerateJwtToken(newUser);

**return:**

return Ok(jwtToken);

1. **Tambahkan method baru** VerifyAndGenerateToken

private async Task<AuthResult> VerifyAndGenerateToken(TokenRequest tokenRequest)

        {

            var jwtTokenHandler = new JwtSecurityTokenHandler();

            try

            {

                var tokenInVerification = jwtTokenHandler.ValidateToken(tokenRequest.Token, \_tokenValidationParams, out var ValidateToken);

                if (ValidateToken is JwtSecurityToken jwtSecurityToken )

                {

                    var result = jwtSecurityToken.Header.Alg.Equals(SecurityAlgorithms.HmacSha256, StringComparison.InvariantCultureIgnoreCase);

                    if(result == false){

                        return null;

                    }

                }

                var utcExpiryDate = long.Parse(tokenInVerification.Claims.FirstOrDefault(x=> x.Type == JwtRegisteredClaimNames.Exp).Value);

                var expiryDate = UnixTimeStampToDateTime(utcExpiryDate);

                if(expiryDate > DateTime.UtcNow){

                    return new AuthResult(){

                        Success = false,

                        Errors = new List<string>(){

                            "Token has expired"

                        }

                    };

                }

                var storedToken = await \_apiDbContext.RefreshTokens.FirstOrDefaultAsync(x=> x.Token == tokenRequest.RefreshToken);

                if(storedToken == null){

                    return new AuthResult(){

                        Success = false,

                        Errors = new List<string>(){

                            "Token does not exist"

                        }

                    };

                }

                if(storedToken.isRevoked){

                    return new AuthResult(){

                        Success = false,

                        Errors = new List<string>(){

                            "Token has been revoked"

                        }

                    };

                }

                var jti = tokenInVerification.Claims.FirstOrDefault(x => x.Type == JwtRegisteredClaimNames.Jti).Value;

                if (storedToken.JwtId != jti)

                {

                    return new AuthResult(){

                        Success = false,

                        Errors = new List<string>(){

                            "Token doesn't match"

                        }

                    };

                }

                storedToken.IsUsed = true;

                \_apiDbContext.RefreshTokens.Update(storedToken);

                await \_apiDbContext.SaveChangesAsync();

                var dbUser = await \_userManager.FindByIdAsync(storedToken.UserId);

                return await GenerateJwtToken(dbUser);

            } catch (Exception err)

            {

                if(err.Message.Contains("Lifetime vaildation failed. The token is expired."))

                {

                    return new AuthResult(){

                        Success = false,

                        Errors = new List<string>(){

                            "Token has expired please re-login"

                        }

                    };

                } else

                {

                    return new AuthResult(){

                        Success = false,

                        Errors = new List<string>(){

                            "Something went wrong."

                        }

                    };

                }

            }

        }

**Note: fungsi dipangggil saat** RefreshToken()

1. **Tambahkan fungsi private baru** UnixTimeStampToDateTime

private DateTime UnixTimeStampToDateTime(long unixTimeStamp)

        {

            var dateTimeVal = new DateTime(1970,1,1,0,0,0,0, DateTimeKind.Utc);

            dateTimeVal = dateTimeVal.AddSeconds(unixTimeStamp);

            return dateTimeVal;

        }

**Note: fungsi dipangggil saat** VerifyAndGenerateToken()

1. **Tambahkan Action baru untuk RefreshToken**

[HttpPost]

[Route("RefreshToken")]

public async Task<IActionResult> RefreshToken([FromBody] TokenRequest tokenRequest)

        {

            if(ModelState.IsValid)

            {

                var result = await VerifyAndGenerateToken(tokenRequest);

                if(result == null){

                    return BadRequest(new RegistrationResponse(){

                        Errors = new List<string>() {"Invalid Token"},

                        Success = false

                    });

                }

                return Ok(result);

            }

            return BadRequest(new RegistrationResponse(){

                        Errors = new List<string>() {"Invalid Payload"},

                        Success = false

                    });

        }