OLIY VA O’RTA TA’LIM MAXSUS

VAZIRLIGI

MIRZO ULUG’BEK NOMIDAGI

O’ZBEKISTON MILLIY UNVERSITETI

AMALIY MATEMATIKA VA INTELEKTUAL TEXNALOGIYALAR FAKULTETI

AXBOROT XAVFSIZLIGI YO’NALISHI

3-KURS TALABASI

JO’RAYEV AHMADJONNING

MALAKAVIY AMALYOT ISHI



Tuzuvchi:Jo’rayev A.

Tekshiruvchi:Aloyev R

MUNDARIJA

1.Kichik bir nechta sahifadan iborat dinamik veb sayt yaratish.

2.Yuqoridagi veb saytingizga autentifikatsiya va indintifikatsiya qo’llash.

3.Yuqoridagi veb saytingizga avtorizatsiya qo’llash.

4.Veb saytlarni zaiflikka analiz qilish.

5.MSSQL bilan ishlaydigan SQL Injection zaiflikka ega veb sayt yaratish.

6.SQL Injection : MSSQL dan barcha jadvallarni nomini olish uchun sql kod yozish.

7.SQL Injection : MSSQL berilgan barcha jadvallarni nomini olish sql kod yozish.

8.SQL Injection : MSSQL da berilgan jadvallarni berilgan ustunlaridagi qiymatlarini olish uchun sql kod yozish.

1.KICHIK BIR NECHTA SAHIFADAN IBORAT DINAMIK VEB SAYT YARATISH.

Demak bir nechta sahifadan iborat bir kichik dinamik veb sahifa yaratishga urinib ko’ramiz.Ushbu veb saytni html css boostrap hamda javascript orqali tuzishga harakat qilamiz.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<link rel="stylesheet" href="./css/bootstrap.css">

</head>

<body>

<section class="navbar\_head">…</nav>

<div class="container my-5">…</div>

<script src="./js/jquery-3.3.1.slim.min.js"></script>

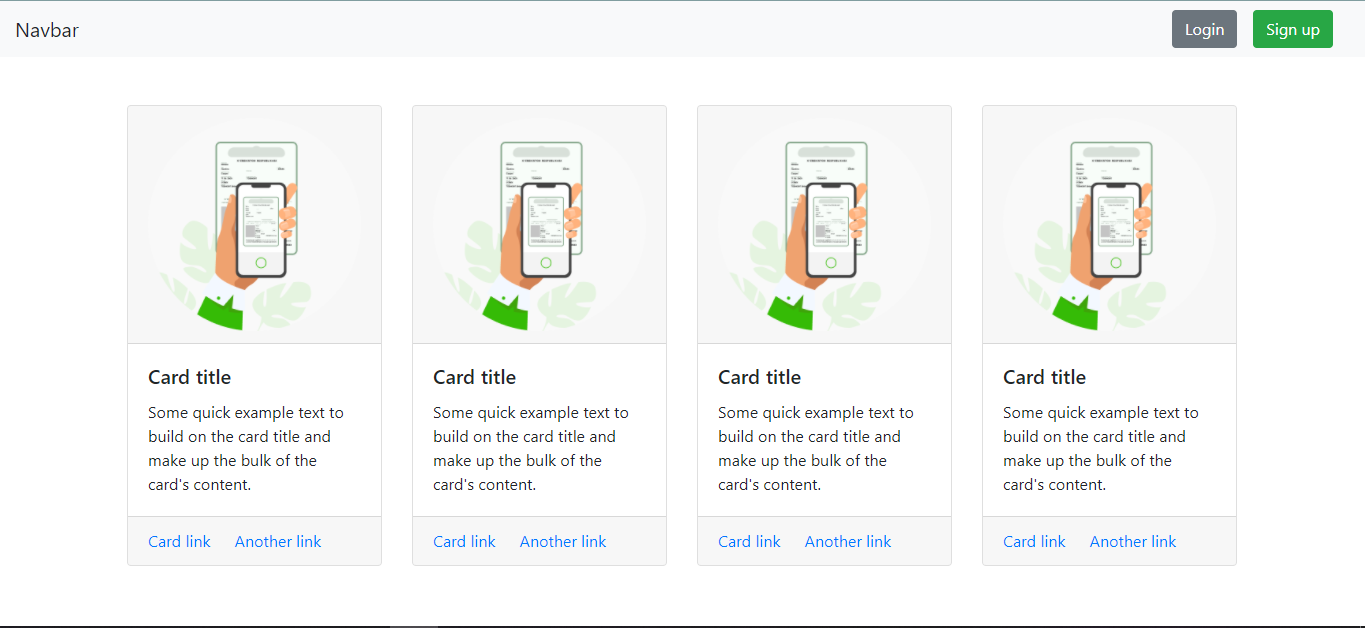
<script src="./js/popper.min.js"></script>

<script src="./js/bootstrap.min.js"></script>

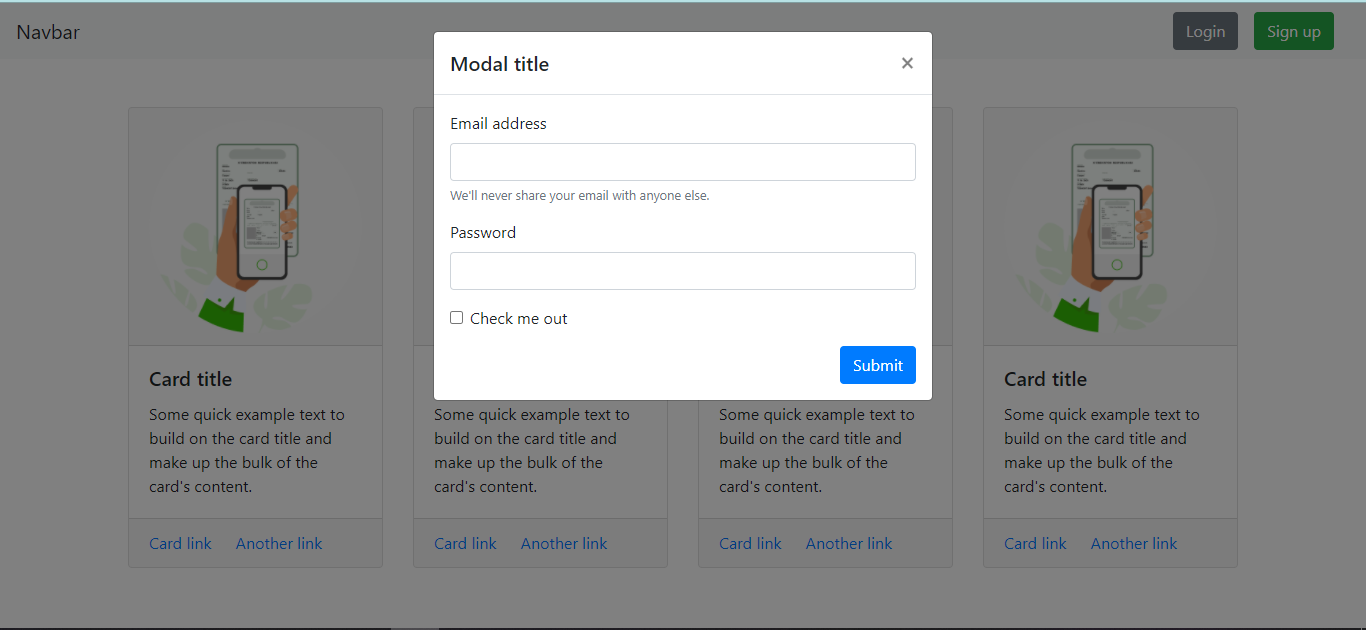
</body>

</html>

Mana bizning kichik veb sahifamiz ham tayyor bo’ldi ushbu veb sahifani tayyorlashda asosan bootstrap css ning kutubxonasidan foydalandik.



1.1-rasm kichik veb sahifaning ko’rinishi.



1.2-rasm veb sahifadagi modalning ko’rinishi

<div class="modal fade" id="exampleModal" tabindex="-1" aria-labelledby="exampleModalLabel" aria-hidden="true">

<div class="modal-dialog">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title" id="exampleModalLabel">Modal title</h5>

<button type="button" class="close" data-dismiss="modal" aria-label="Close">

<span aria-hidden="true">&times;</span>

</button>

</div>

<div class="modal-body">

<form>

<div class="form-group">

<label for="exampleInputEmail">Email address</label>

<input type="email" class="form-control" id="exampleInputEmail" aria-describedby="emailHelp">

<small id="emailHelp" class="form-text text-muted">We'll never share your email with anyone else.</small>

</div>

<div class="form-group">

<label for="exampleInputUserName">User Name</label>

<input type="text" class="form-control" id="exampleInputUserName">

</div>

<div class="form-group">

<label for="exampleInputPassword">Password</label>

<input type="password" class="form-control" id="exampleInputPassword">

</div>

<div class="form-group form-check">

<input type="checkbox" class="form-check-input" id="exampleCheck1">

<label class="form-check-label" for="exampleCheck1">Check me out</label>

</div>

<button type="submit" class="btn btn-primary float-right" onclick="AddUser()">Create User</button>

</form>

</div>

</div>

</div>

</div>

</section>

Web sahifada ishlatilgan modal ning html kodlari.

<div class="container my-5">

<div id="root" class="row">

<div class="col-md-3">

<div class="card">

<div class="card-header img-fluid">

<img src="./images/Group 55441.png" width="100%" height="100%" alt="">

</div>

<div class="card-body ">

<h5 class="card-title">Card title</h5>

<p class="card-text">Some quick example text to build on the card title and make up the bulk of the card's content.</p>

</div>

<div class="card-footer">

<a href="#" class="card-link">Card link</a>

<a href="#" class="card-link">Another link</a>

</div>

</div>

</div>

</div>

</div>

Bu esa veb sahifadagi cardning html kodlari.

2.YUQORIDAGI VEB SAYTGA AUTENTIFIKATSIYA VA IDENTIFIKATSIYA QO’LLASH.

Veb sahifaga identifikatsiya qo’shish uchun IdentityServer 4 dab foydalanamiz. Buning uchun nuget.org dan

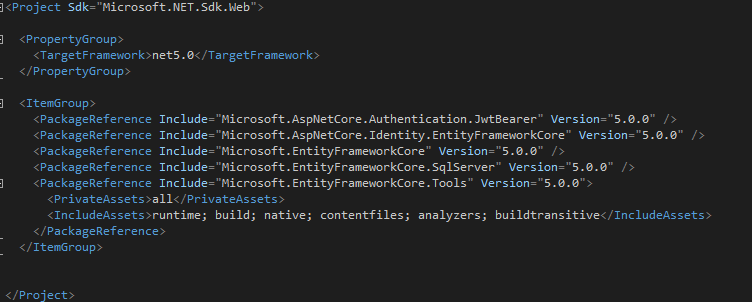
Install-Package Microsoft.EntityFrameworkCore -Version 5.0

Install-Package Microsoft.AspNetCore.Identity.EntityFrameworkCore -Version 5.0

Install-Package Microsoft.EntityFrameworkCore.SqlServer -Version 5.0

Install-Package Microsoft.EntityFrameworkCore.Tools -Version 5.0

Install-Package Microsoft.AspNetCore.Authentication.JwtBearer -Version 5.0 ushbu kutubxonalardan foydalanamiz.



2.1-rasm dasturning .csproj faylining ko’rinishi

Endilikda DbContext classidan voris olgan ma’lumotlar ombori bilan ishlash maqsadida class yaratamiz.

namespace SingleWebIdentityAplication.Models

{

public class ApplicationDbContext : IdentityDbContext

{

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

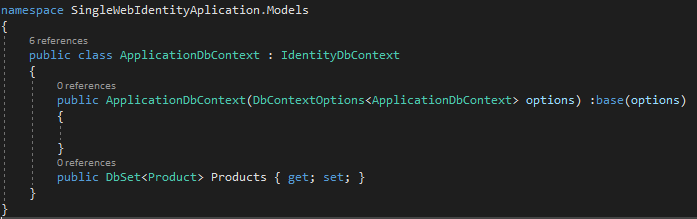
{

}

public DbSet<Product> Products { get; set; }

}

}



2.2-rasm ApplicationDbContext classining ko’rinishi

Bu yerda Dbset obyekti orqali ma’lumotlar omboridagi ma’lumotlarni obyekt ko’rinishida saqlashimiz yoki kiritishimiz mumkin.

Tuzayotgan veb sahifamiz online magazine bo’lgani sababli mahsulotlar haqidagi ma’lumotlarni saqlash maqsadida Product nomli classni yaratib olamiz.

namespace SingleWebIdentityAplication.Models

{

public class Product

{

[Key]

public int ProductId { get; set; }

public string ProductName { get; set; }

public int ProductCost { get; set; }

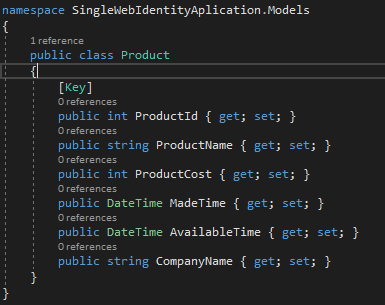
public DateTime MadeTime { get; set; }

public DateTime AvailableTime { get; set; }

public string CompanyName { get; set; }

}

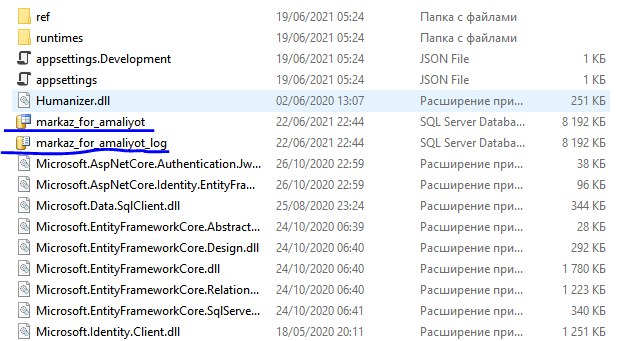
}



2.3-rasm Product classining ko’rinishi

Keyingi bosqichda biz LocalDb ma’lumotlar ombori bilan ishlaymiz.

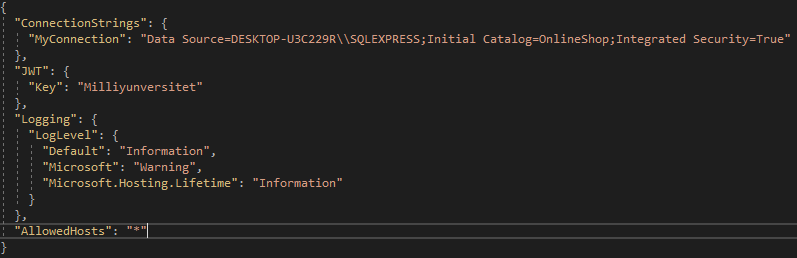
Dastlab .mdf file hosil qilib olamiz



2.4-rasm .mdf faylning joylashuvi

Endi LocalDb bilan connectionString yordamida bog’lanamiz.

Odatda connectionString appsetting.json faylda saqlanadi.



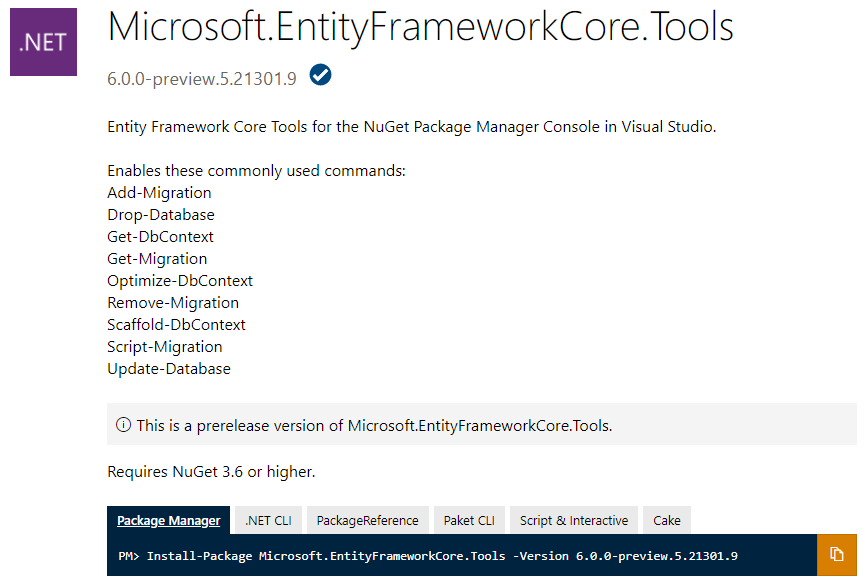
2.5-rasm connectionString ning ko’rinishi

Endilikda migration orqali ma’lumotlar omborida jadvallar yaratamiz,

Undan oldin esa migration haqida qisqacha to’xtalib o’tsak.Migration – EF Core-dagi migratsiya xususiyati ma'lumotlar bazasida mavjud bo'lgan ma'lumotlarni saqlab, ma'lumotlar bazasi sxemasini dasturning ma'lumotlar modeli bilan sinxronlashtirish uchun uni bosqichma-bosqich yangilashga imkon beradi.  loyihalarda ma'lumotlar modellari funktsiyalarni amalga oshirishda o'zgaradi yangi ob'ektlar yoki xususiyatlar qo'shiladi va o'chiriladi va ma'lumotlar bazasi sxemalari mos ravishda dastur bilan hamohang bo'lishi uchun o'zgartirilishi kerak. Yuqori darajada migratsiya quyidagi tarzda ishlaydi:

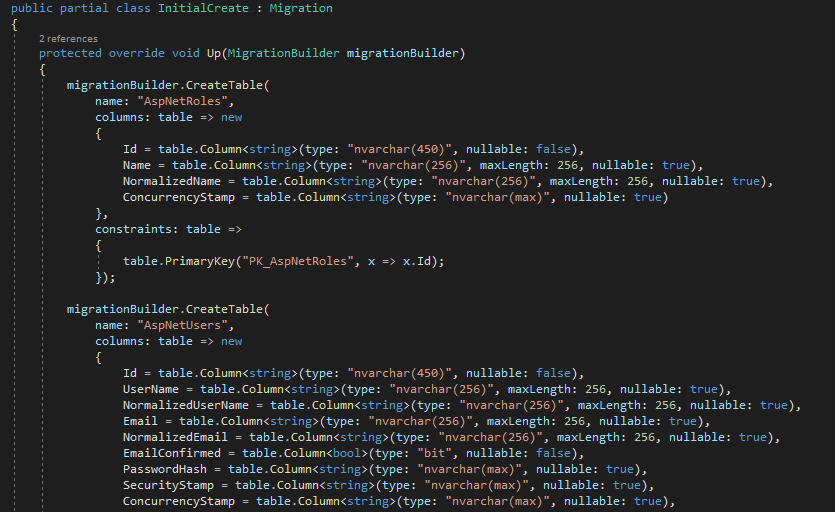
* Ma'lumotlar modelini o'zgartirish kiritilganda, ishlab chiquvchi ma'lumotlar bazasi sxemasini sinxronlashtirish uchun zarur bo'lgan yangilanishlarni tavsiflovchi mos keladigan migratsiyani qo'shish uchun EF Core vositalaridan foydalanadi. EF Core farqni aniqlash uchun amaldagi modelni eski modelning surati bilan taqqoslaydi va ko'chirish manbalari fayllarini yaratadi; fayllarni boshqa har qanday manba fayllari kabi loyihangizning manba boshqaruvida kuzatish mumkin.
* Yangi migratsiya hosil bo'lgandan so'ng, ma'lumotlar bazasiga turli usullar bilan qo'llanilishi mumkin. EF Core barcha ko'chib o'tishni maxsus tarix jadvalida qayd etib, qaysi migratsiya qo'llanilgan va qaysisi qo'llanilmaganligini bilishga imkon beradi.

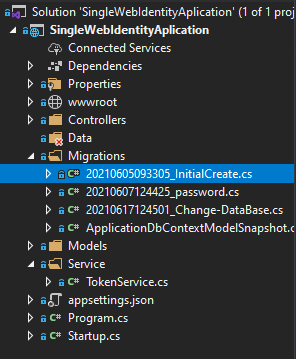
Shuni yodda tutingki migration extantion method bo’lgani sababdan nuget.org sayti orqali : Microsoft.EntityFrameworkCore.Tools package ni yuklab olishimiz kerak.



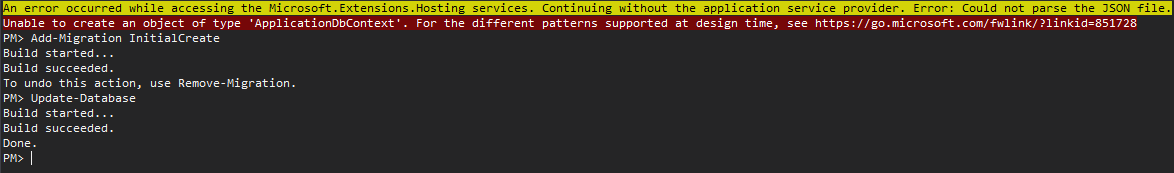
2.6-rasm nuget package ning ko’rinishi

Keyingi boshqichda nuget package console oynasi orqali migrationga buyruq beramiz: Add-Migration “InitialCreate”.

Ushbu buyruqdan so’ng migration migration fayllarni yartadi. 

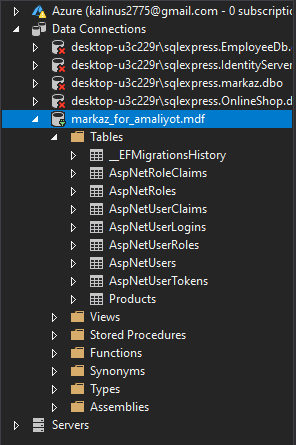


2.7- 2.8 -rasmlar migration faylining ko’rinishi



2.9-rasm migration buyruqlarining bajarilishi

Update-Database buyrug’idan so’ng ma’lumotlar omborida bir qancha jadvallar yaratilganini ko’rishimiz mumkin.



2.10-rasm migration dan so’ng jadvallarning ko’rinishi

Ko’rib turganimiz dek migration orqali bir qancha qo’shimcha jadvallar hosil bo’ldi buning asosiy jsababi biz EntityFrameworkCore da IdentityDbContext dan foydalanganimiz uchun (2.2-rasm).

Keying bosqichda biz startup faylida identifikatsiya va autentifikastisani ro’yhatdan o’tkazamiz.

services.AddAuthentication(x =>

{

x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

}).AddJwtBearer(x =>

{

x.RequireHttpsMetadata = true;

x.SaveToken = true;

x.TokenValidationParameters = new Microsoft.IdentityModel.Tokens.TokenValidationParameters

{

ValidateIssuerSigningKey = true,

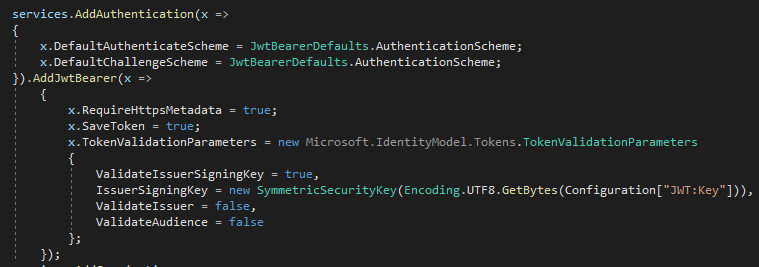
IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["JWT:Key"])),

ValidateIssuer = false,

ValidateAudience = false

};

});



2.11-rasm authentification ni qo’llanishi

Biz ushbu kichik veb sahifamizda autentification ni JWT token orqali amalga oshirmoqdamiz.Shuning uchun endilikda dasturimizga JWT tokenni hosil qilish uchun service qo’shamiz.

public string GenerateToken(IModel model)

{

var tokenHandler = new JwtSecurityTokenHandler();

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

var tokenDiscriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(new Claim[]

{

new Claim(ClaimsType.UserName,model.UserName),

new Claim(ClaimsType.UserRole, RolesBase.User)

}),

Expires = DateTime.UtcNow.AddHours(1),

SigningCredentials = new SigningCredentials(

new SymmetricSecurityKey(key),

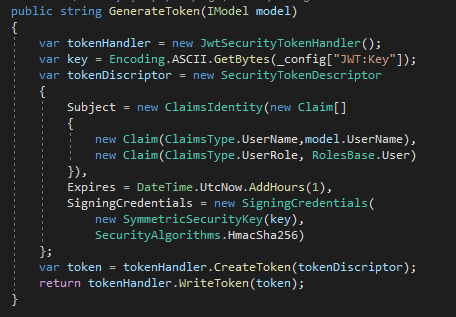
SecurityAlgorithms.HmacSha256)

};

var token = tokenHandler.CreateToken(tokenDiscriptor);

return tokenHandler.WriteToken(token);

}



2.12-rasm JWT tokenni hosil qilinishi

GenerateToken() methodi ko’rib turganimiz dek parameter sifatida IModel interface sidan voris olgan class larni qabul qiladi.

namespace SingleWebIdentityAplication.Models

{

public class SignInViewModel : LoginViewModel

{

public string EmailAddress { get; set; }

}

public class LoginViewModel : IModel

{

public string UserName { get; set; }

public string Password { get; set; }

}

public interface IModel

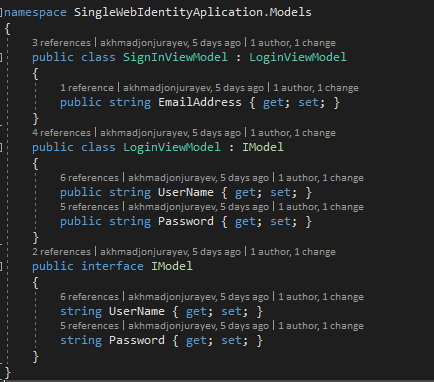
{

string UserName { get; set; }

string Password { get; set; }

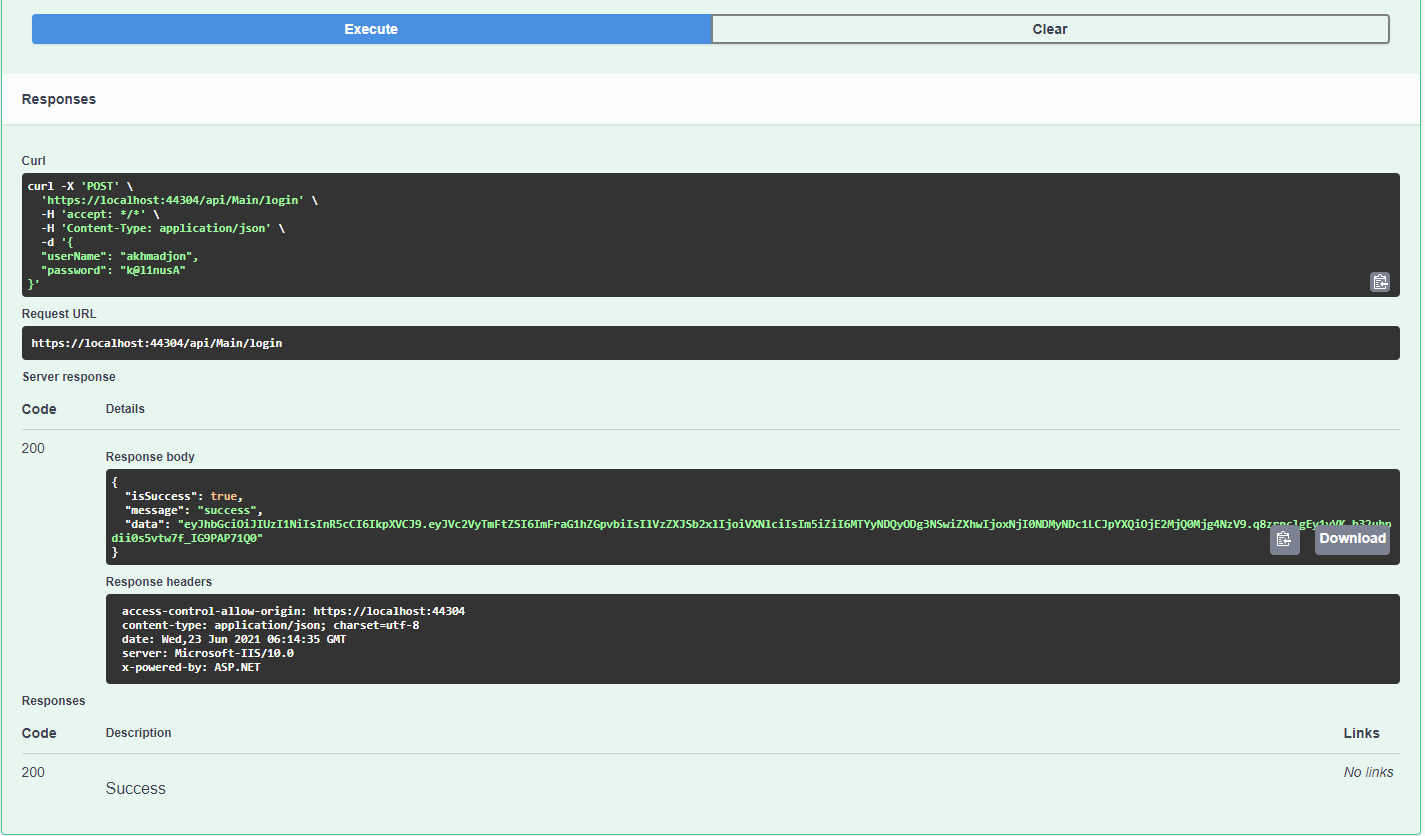
}

}

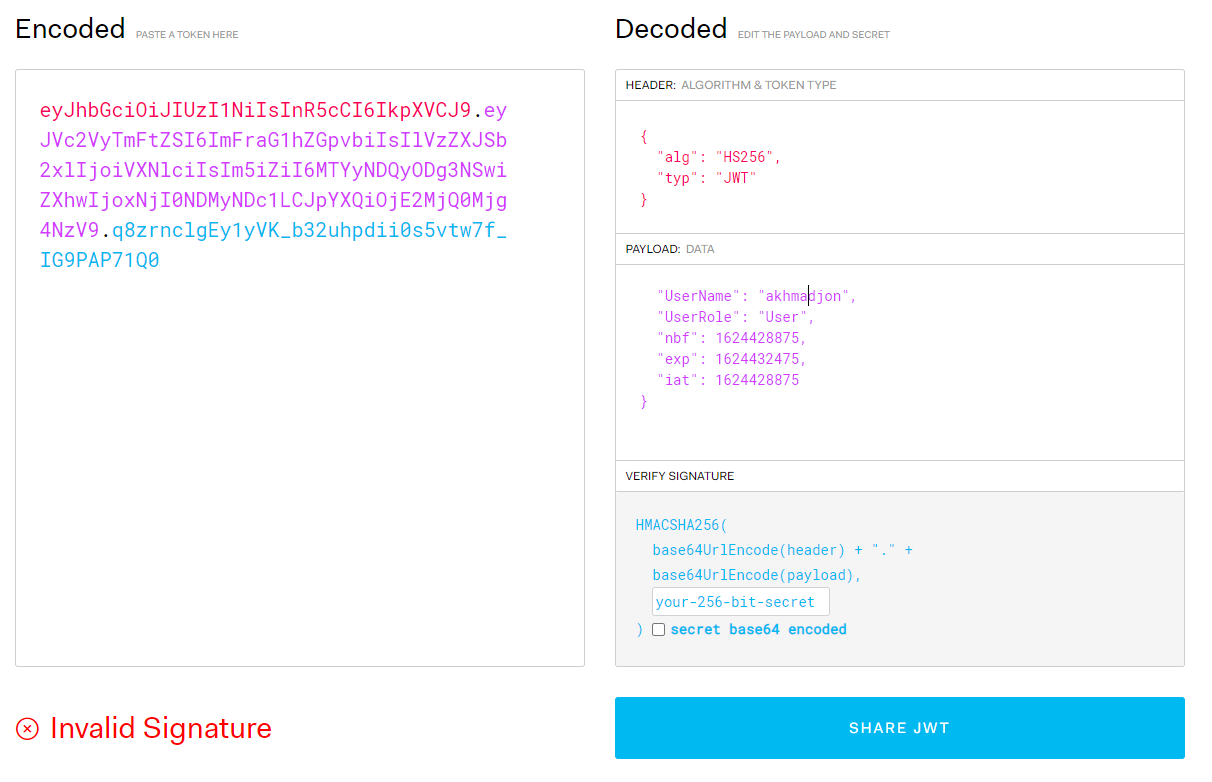


2.13-rasm ViewModellarning ko’rinishi

Keying bosqichda biz GenerateToken() methodining ishlashi tekshirish maqsadida Swagger orqali tekshirib chiqishimiz mumkin.



2.14-rasm Swaggerdan methodni tekshirish



2.15-rasm Jwt.io saytidan tokenni tekshirilishi

3.YUQORIDAGI VEB SAHIFAGA AVTORIZATSIYANING QO’LLANISHI

Bizning veb sahifamizga avtorizatsiyani qo’llash uchun html fayllarimizda ham biroz o’zgartirishlarni amalga oshirishimiz talab qilinadi.Undan oldin esa veb sahifamizning backend qismida ham bir oz o’zgarishlarni amalga oshiramiz.

services.AddIdentity<IdentityUser, IdentityRole>(p =>

{

p.Password.RequireDigit = true;

p.Password.RequiredLength = 8;

p.Password.RequireLowercase = true;

p.Password.RequireNonAlphanumeric = false;

}).AddEntityFrameworkStores<ApplicationDbContext>().

AddDefaultTokenProviders();

Starup faylida avtorizatsiyani ro’yhatdan o’tkazamiz.Middeleware da

app.UseAuthorization();

Endilikda service qismida avtorizatsiya uchun qo’shimcha methodlarni yozamiz.

public interface ITokenService

{

Task<ResponseData<string>> SignInToken(SignInViewModel model);

Task<ResponseData<string>> LogInToken(LoginViewModel login);

Task<ResponseData<List<Product>>> GetAllProduct();

Task<ResponseData<string>> AddRangeProduct(IEnumerable<Product> products);

}

public class TokenService : ITokenService

{

private readonly ApplicationDbContext \_db;

private readonly IConfiguration \_config;

private readonly UserManager<IdentityUser> \_userManager;

private readonly SignInManager<IdentityUser> \_signIn;

public TokenService(IConfiguration config,

UserManager<IdentityUser> userManager,

SignInManager<IdentityUser> signIn,

ApplicationDbContext db)

{

\_db = db;

\_config = config;

\_userManager = userManager;

\_signIn = signIn;

}

public async Task<ResponseData<string>> SignInToken(SignInViewModel model)

{

if (model.UserName == default || model.Password == default)

return new ResponseData<string>() { IsSuccess = false, Message = "error-invalid-data" };

var user = new IdentityUser

{

UserName = model.UserName,

Email = model.EmailAddress

};

var signIn =await \_userManager.CreateAsync(user);

if(signIn.Succeeded)

{

var result = await \_userManager.AddPasswordAsync(user, model.Password);

if(result.Succeeded)

{

return new ResponseData<string>() { IsSuccess = true, Message = "success-add-data", Data = GenerateToken(model) };

}

}

return new ResponseData<string>() { IsSuccess = false, Message = signIn.Errors.ToString() };

}

public string GenerateToken(IModel model)

{

var tokenHandler = new JwtSecurityTokenHandler();

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

var tokenDiscriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(new Claim[]

{

new Claim(ClaimsType.UserName,model.UserName),

new Claim(ClaimsType.UserRole, RolesBase.User)

}),

Expires = DateTime.UtcNow.AddHours(1),

SigningCredentials = new SigningCredentials(

new SymmetricSecurityKey(key),

SecurityAlgorithms.HmacSha256)

};

var token = tokenHandler.CreateToken(tokenDiscriptor);

return tokenHandler.WriteToken(token);

}

public async Task<ResponseData<string>> LogInToken(LoginViewModel login)

{

if (login.Password == default || login.UserName == default)

return new ResponseData<string>() { IsSuccess = false, Message = "error-invalid-data" };

var user = await \_userManager.FindByNameAsync(login.UserName);

if (user == null)

return new ResponseData<string>() { IsSuccess = false, Message = "error-not-found-data" };

var isSuccess = await \_signIn.CheckPasswordSignInAsync(user, login.Password, false);

if (isSuccess.Succeeded)

{

return new ResponseData<string>() { IsSuccess = true, Message = "success", Data = GenerateToken(login) };

}

return new ResponseData<string>() { IsSuccess = false, Message = "error-not-found-data" };

}

public async Task<ResponseData<List<Product>>> GetAllProduct()

{

return new ResponseData<List<Product>>() { IsSuccess = true, Data = await \_db.Products.Take(4).AsNoTracking().ToListAsync() };

}

public async Task<ResponseData<string>> AddRangeProduct(IEnumerable<Product> products)

{

if(products.Any())

{

\_db.ChangeTracker.AutoDetectChangesEnabled = false;

await \_db.Products.AddRangeAsync(products);

\_db.ChangeTracker.AutoDetectChangesEnabled = true;

await \_db.SaveChangesAsync();

return new ResponseData<string>() { IsSuccess = true, Message = "success-add-data" };

}

return new ResponseData<string>() { IsSuccess = false, Message = "error-invalid-data" };

}

}

Dendency Injection dan foydalanish maqsadida service simizga interface dan foydalanamiz hamda stratup faylida ro’yhatdan o’tkizamiz.

services.AddScoped<ITokenService, TokenService>();

endilikda sahifamizga kirishi uchun login.html faylini hosil qilamiz.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Login Page</title>

<link rel="stylesheet" href="./css/bootstrap.css">

</head>

<body>

<div class="container">

<div class="row" style="display: flex;

justify-content: center;

align-items: center;

height: 100vh;">

<div class="col-md-4">

<form>

<div class="form-group">

<label>User Name :</label>

<input class="form-control" type="text" name="username" id="name">

</div>

<div class="form-group">

<label>Password :</label>

<input class="form-control" type="password" name="password" id="password">

</div>

<div class="form-check">

<input type="checkbox" class="form-check-input" id="exampleCheck1">

<label class="form-check-label" for="exampleCheck1">Check me out</label>

</div>

<button class="btn btn-info" type="button" onclick="ChekUser()">Submit</button>

</form>

</div>

<div class="col-md-8">

<img src="./images/rasm4.jpg" width="100%" height="100%">

</div>

</div>

</div>

<script>

function ChekUser() {

var myHeaders = new Headers();

myHeaders.append("Content-Type", "application/json");

myHeaders.append("charset", "utf-8");

var raw = JSON.stringify({

"UserName": document.getElementById("name").value,

"Password": document.getElementById("password").value

});

var requestOptions = {

method: 'POST',

headers: myHeaders,

body: raw,

redirect: 'follow'

};

fetch("https://localhost:44304/api/main/login", requestOptions)

.then(response => response.text())

.then(result => {

var data = JSON.parse(result);

console.log(data);

if (data.isSuccess) {

window.location.href = "./index.html";

}

})

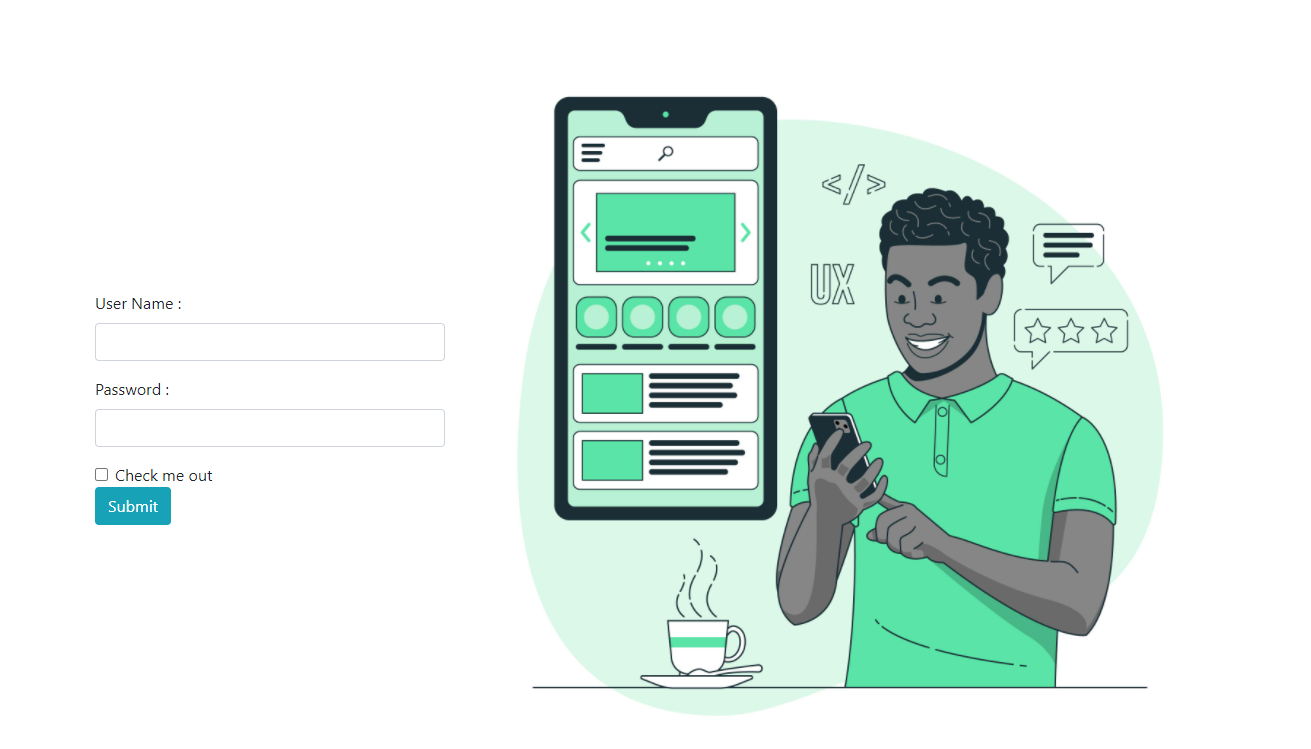
.catch(error => console.log('error', error));

}

</script>

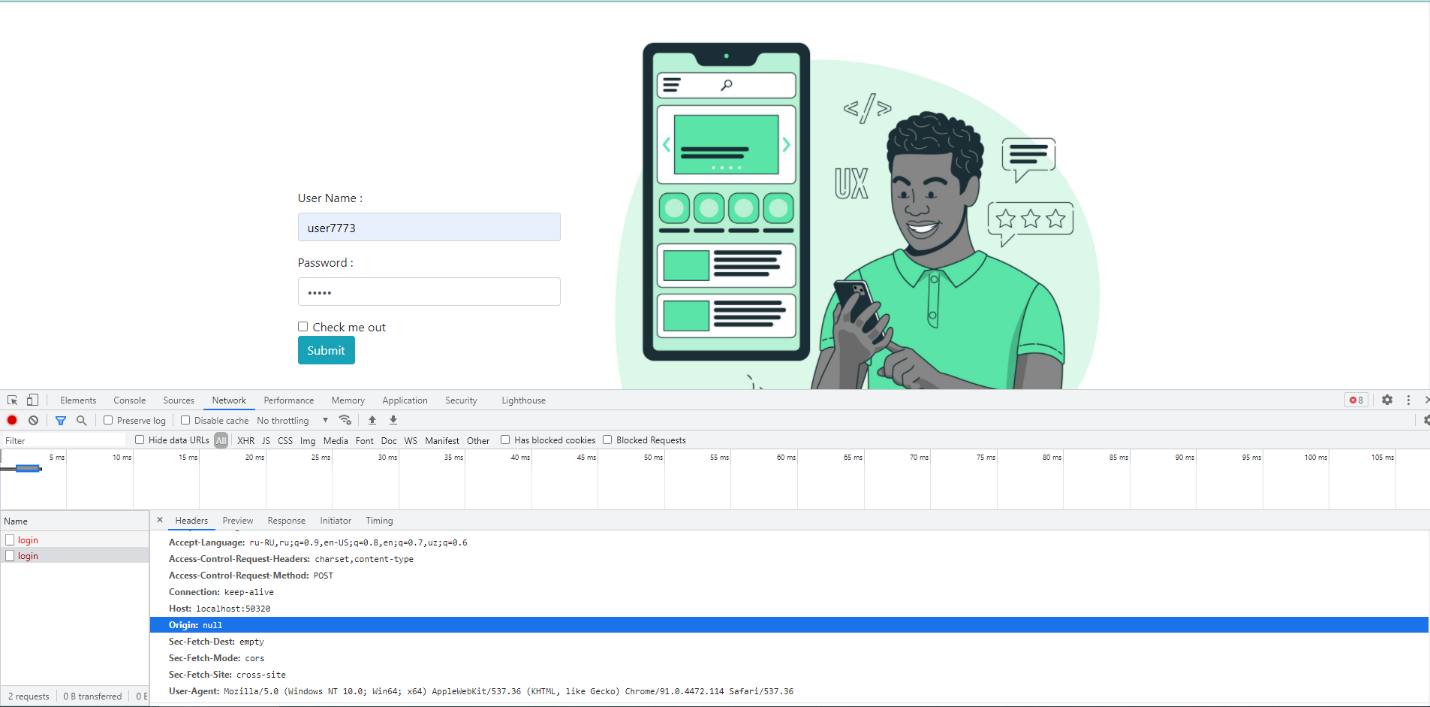
</body>

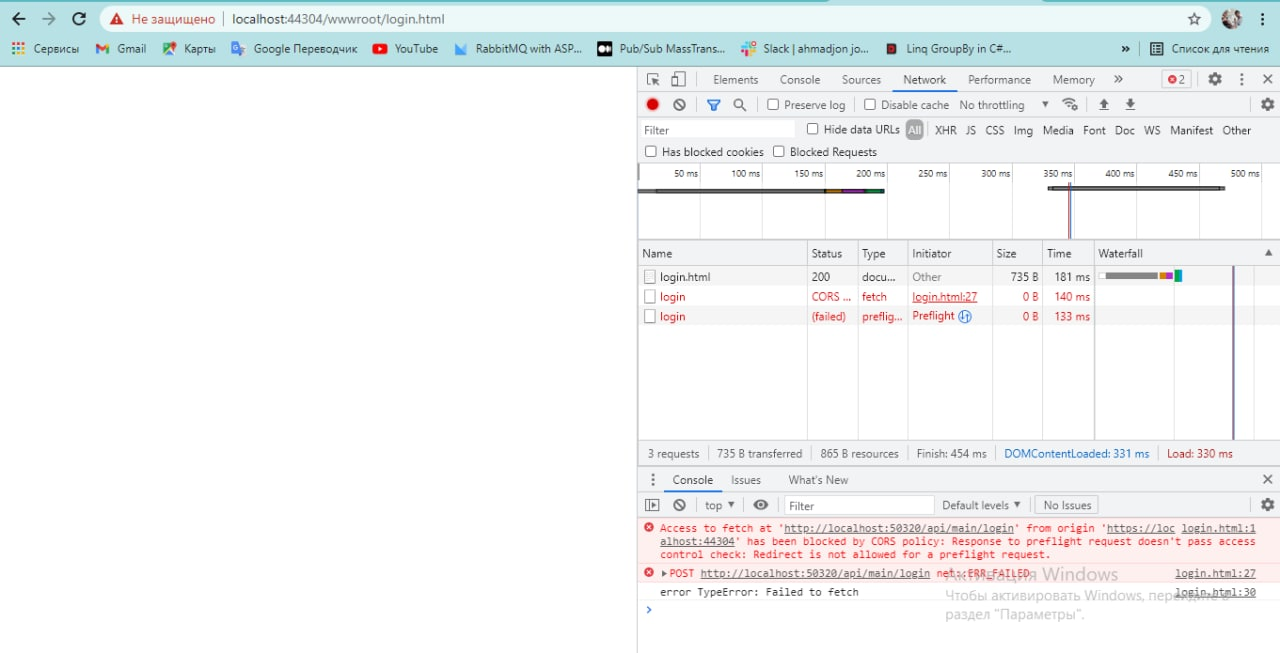
</html>



3.1-rasm login.html ning ko’rinishi.

Endilida dasturni ishlashini tekshirib ko’ramiz.





3.2 3.3 -rasmlar login.html faylidagi haolikning ko’rinishi

Ko’rib turganimiz kabi dasturimiz ishlashi jarayonida hatolik yuzaga keldi.Buning asosiy sabbai Cors origins bilan muomo hisoblanadi veb sahifa bir bir Restful Api dan foydalanishi uchun saytda Cors Origins siyosati qo’lanishi kerak bo’ladi.Agar Cors origins hizmati yoqilmagan bo’lsa u holda tashqaridan kelgan so’rovlarga dastur javob qaytara olmaydi. Undan tashqari veb sahifani cors originsi ham null bo’lmasligi kerak.Bu muommoni yechish uchun startup faylida cors origins ni hizmatlarga qo’shib qo’yamiz.

services.AddCors(options =>

{

options.AddPolicy("default", builder =>

{

builder.WithOrigins("https://localhost:44304");

builder.AllowAnyHeader();

builder.AllowAnyMethod();

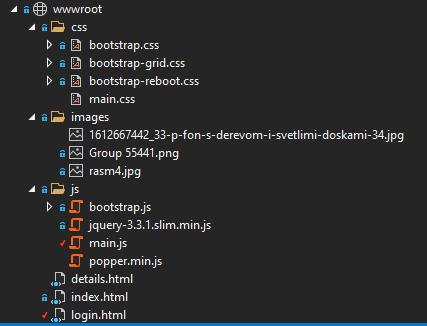
});

});

app.UseCors("default");

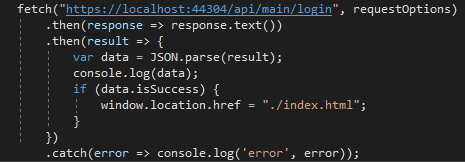
Keyingi boshqichda login.html faylimizni wwwroot faylini ichiga ko’chiramiz hamda middleware da o’zgarishlarni hosil qilamiz.

app.UseStaticFiles("/wwwroot");



3.4-rasm wwwroot faylining ko’rinishi

Endilikda login.html faylini ishga tushirsak localhost orqali ishga tushadi shu sababidan cors origin ni qiymati sifatida localhostni oladi.



3.5-rasm Promise ning ko’rinishi

Ko’rib turganimizdek agar foydalanuvchi autentification dan muvafiqiyatli ravishda o’tsa u holda index.html sahifasiga o’tishini amalga oshirmoqdamiz.

4.VEB SAYTLARNI ZAIFLIKKA ANALIZ QILISH

Ilgari ko'plab mashhur veb-saytlar buzilgan.Hackerlar faol va har doim veb-saytlarni buzishga va ma'lumotlarni tarqatishga harakat qilishadi.Shuning uchun veb-ilovalarning xavfsizligini sinovdan o'tkazish juda muhimdir. Veb-dastur xavfsizligi skaneri - bu veb-ilovada qora qutilarni avtomatik sinovdan o'tkazadigan va xavfsizlik nuqsonlarini aniqlaydigan dasturiy ta'minot.Skanerlar manba kodiga kira olmaydi ular faqat funktsional sinovlarni amalga oshiradilar va xavfsizlikning zaif tomonlarini topishga harakat qilishadi.Turli pullik va bepul veb-dasturlarning zaiflik skanerlari mavjud.Hozir esa ochiq kodli veb dasturlarni zaiflikka tekshiradigan dasturlar bilan tanishamiz.

1. Grabber

Grabber - bu veb-ilovalardagi ko'plab xavfsizlik zaifliklarini aniqlay oladigan veb-dastur skaneri.U skanerlashni amalga oshiradi va zaiflik qayerda ekanligini aytadi.U quyidagi zaifliklarni aniqlay oladi:

* Saytlararo senariy
* SQL in'ektsiyasi
* Ajax sinovi
* Fayl qo'shilishi
* JS manba kodi analizatori
* Zaxira nusxasini tekshirish

Boshqa xavfsizlik brauzerlari bilan taqqoslaganda tez emas, lekin sodda va ko'chma.Bu faqat kichik veb-dasturlarni sinash uchun ishlatilishi kerak, chunki katta dasturlarni skanerlash uchun juda ko'p vaqt talab etiladi.

Ushbu vosita hech qanday GUI interfeysini taklif qilmaydi.Bundan tashqari, u PDF hisobotlarini yarata olmaydi.Ushbu vosita oddiy va shaxsiy foydalanish uchun mo'ljallangan.Ushbu vosita Python-da ishlab chiqilgan va ochiq kodi mavzud : <https://github.com/neuroo/grabber>.

## 2. Vega

Vega yana bir bepul ochiq manbali veb-brauzer va sinov platformasi. Ushbu vosita yordamida siz veb-dastur xavfsizligini sinovdan o'tkazishingiz mumkin. Ushbu vosita Java-da yozilgan va GUI-ga asoslangan muhitni taklif qiladi. Bu OS, Linux va Windows uchun mavjud.Undan SQL in'ektsiyasi, sarlavha kiritish, kataloglar ro'yxati, qobiq in'ektsiyasi, saytlararo skriptlar, fayllarni qo'shish va boshqa veb-dasturlarning zaif tomonlarini topish uchun foydalanish mumkin. Ushbu vosita JavaScript-da yozilgan kuchli API yordamida ham kengaytirilishi mumkin.Dastur bilan ishlashda u sizga so’rovlarning umumiy soni, soniyada so'rovlarning og’irligi va maksimal soni kabi bir nechta parametrlarni o'rnatishga imkon beradi. Siz Vega Scanner, Vega Proxy va Proxy Scanner-dan foydalanishingiz va shuningdek hisob ma'lumotlari bilan skanerlashingiz mumkin.

3. Zed Attack Proxy

Zed Attack Proxy ZAP nomi bilan ham tanilgan. Ushbu vosita ochiq manbali va OWASP tomonidan ishlab chiqilgan. Bu Windows, Unix / Linux va Macintosh platformalarida mavjud.Bu veb-ilovalardagi zaifliklarning keng doirasini topish uchun ishlatilishi mumkin. Asbob juda sodda va ulardan foydalanish oson. Agar siz penetratsion test bilan yangi tanishsangiz ham, ushbu vositadan osongina veb-ilovalarning penetratsion sinovlarini o'rganishni boshlashingiz mumkin.Bu ZAPning asosiy funktsiyalari:

* Proksi-serverni ushlab qolish
* Avtomatik skaner
* Fuzzer
* Veb-socket qo'llab-quvvatlash
* Plug-n-hack yordami
* Autentifikatsiyani qo'llab-quvvatlash
* REST-ga asoslangan API
* Dinamik SSL sertifikatlari
* Smartcard va mijozlarning raqamli sertifikatlarini qo'llab-quvvatlash

Siz ushbu vositani skanerlash uchun URL manzilini kiritish orqali skaner sifatida ishlatishingiz yoki ushbu sahifadan testlarni qo'lda bajarish uchun ushbu vositani ushlab turuvchi proksi sifatida foydalanishingiz mumkin.

## 9. SQLMap

SQLMap yana bir mashhur ochiq manbali penetratsion sinov vositasi. Bu veb-sayt ma'lumotlar bazasida SQL in'ektsiyasining zaif tomonlarini topish va ulardan foydalanish jarayonini avtomatlashtiradi. U kuchli aniqlash dvigateliga va ko'plab foydali xususiyatlarga ega. Shunday qilib, penetratsion sinovchi veb-saytda SQL in'ektsiyasini tekshirishni osongina amalga oshirishi mumkin. MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2, SQLite, Firebird, Sybase va SAP MaxDB kabi ma'lumotlar bazalari serverlarini qo'llab-quvvatlaydi. U oltita SQL injektsiya texnikasini to'liq qo'llab-quvvatlaydi: vaqtga asoslangan ko'r-ko’rona so’rovlar, mantiqqa asoslangan ko'r-ko’rona so’rovlar, xatolarga asoslangan so’rovlar, UNION so'rovlari, yig'ilgan so'rovlar va tarmoqdan tashqarida.

5.MSSQL bilan ishlaydigan SQL Injection zaiflikka ega veb sayt yaratish.

Sql injection uchun .net core mvc yordami kichik veb sahifa yaratamiz.

Dastlab ishni ma’lumotlar bazasida jadval yaratish bilan boshlaymiz.

Go Create Database IdentityServer4;

CREATE TABLE [dbo].[Employees] (

[EmployeeId] INT IDENTITY (1, 1) NOT NULL,

[FistName] NVARCHAR (MAX) NULL,

[LastName] NVARCHAR (MAX) NULL,

[Email] NVARCHAR (MAX) NULL,

[PhoneNumber] NVARCHAR (MAX) NULL,

CONSTRAINT [PK\_Employees] PRIMARY KEY CLUSTERED ([EmployeeId] ASC)

);

CREATE TABLE [dbo].[AspNetUsers] (

[Id] NVARCHAR (450) NOT NULL,

[UserName] NVARCHAR (256) NULL,

[NormalizedUserName] NVARCHAR (256) NULL,

[Email] NVARCHAR (256) NULL,

[NormalizedEmail] NVARCHAR (256) NULL,

[EmailConfirmed] BIT NOT NULL,

[PasswordHash] NVARCHAR (MAX) NULL,

[SecurityStamp] NVARCHAR (MAX) NULL,

[ConcurrencyStamp] NVARCHAR (MAX) NULL,

[PhoneNumber] NVARCHAR (MAX) NULL,

[PhoneNumberConfirmed] BIT NOT NULL,

[TwoFactorEnabled] BIT NOT NULL,

[LockoutEnd] DATETIMEOFFSET (7) NULL,

[LockoutEnabled] BIT NOT NULL,

[AccessFailedCount] INT NOT NULL,

CONSTRAINT [PK\_AspNetUsers] PRIMARY KEY CLUSTERED ([Id] ASC)

);

Endilikda connection stringni appsetting.json fayliga yozamiz. "ConnectionStrings": {

"MyConnection": "Data Source=DESKTOP-U3C229R\\SQLEXPRESS;Initial Catalog=IdentityServer4;Integrated Security=True"

},

Keyingi boshqichda controller da Action method yaratamiz.

[HttpGet("getdata/{id}")]

public IActionResult GetData(string id)

{

List<Authors> authors = new List<Authors>();

Authors auth;

string ConnectionString = \_con.GetConnectionString("MyConnection");

using(SqlConnection con=new SqlConnection(ConnectionString))

{

using(SqlCommand cmd=new SqlCommand())

{

string SqlQuery = $"select \* from Authors Where AId = {id}";

cmd.CommandText =SqlQuery;

cmd.Connection = con;

using(SqlDataAdapter sda=new SqlDataAdapter())

{

con.Open();

sda.SelectCommand = cmd;

using(SqlDataReader sdr=cmd.ExecuteReader())

{

while(sdr.Read())

{

auth = new Authors();

auth.AId = Convert.ToInt32(sdr["AId"]);

auth.FirstName = sdr["FirstName"].ToString();

auth.LastName = sdr["LastName"].ToString();

auth.PhoneNumber = sdr["PhoneNumber"].ToString();

auth.Emailaddress = sdr["Emailaddress"].ToString();

authors.Add(auth);

}

}

}

}

}

return Ok(authors);

}

}

Keyincha bosqichda .chtml faylini yaratib olamiz:

@model Sql\_injection\_Mvc.Models.Data

@{

ViewData["Title"] = "Home Page";

}

<div class="col-md-8">

<form method="post" asp-action="Search" class="text-center">

<div class="form-group">

<label asp-for="FirstName"></label>

<input asp-for="FirstName" class="form-control" />

<span asp-validation-for="FirstName" class="text-danger"></span>

</div>

<button type="submit" class="btn btn-info">Search</button>

</form>

@if(Model.emp!=null)

{

<ul>

@foreach(var item in Model.emp)

{

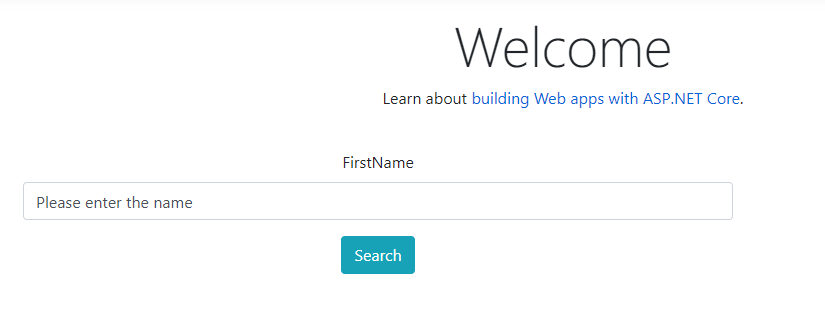
<li>@item.FirstName + @item.LastName</li>

}

</ul>

}

</div>



5.1-rasm veb dasturlarning html ko’rinishi

6.SQL Injection : MSSQL dan barcha jadvallarni nomini olish uchun sql kod yozish.

SELECT \* FROM

(SELECT table\_name FROM INFORMATION\_SCHEMA.TABLES)

7.SQL Injection : MSSQL berilgan barcha jadvallarni nomini olish sql kod yozish.

SELECT COLUMN\_NAME

FROM INFORMATION\_SCHEMA.COLUMNS

8.SQL Injection : MSSQL da berilgan jadvallarni berilgan ustunlaridagi qiymatlarini olish uchun sql kod yozish.

SELECT \* FROM (SELECT table\_name FROM information\_schema)