private string GenerateAccessToken(int userId)

{

var tokenHandler = new JwtSecurityTokenHandler();

var key = Encoding.UTF8.GetBytes(\_jwtsettings["Jwt:Key"]);

//var key = Encoding.UTF8.GetBytes(\_jwtsettings["[Jwt:Key"]);

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(new Claim[]

{

// Added UserID Here

new Claim(ClaimTypes.Name, Convert.ToString(userId))

}),

Expires = DateTime.UtcNow.AddDays(1),

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key),

SecurityAlgorithms.HmacSha256Signature)

};

var token = tokenHandler.CreateToken(tokenDescriptor);

return tokenHandler.WriteToken(token);

}

Add a RefreshToken Model Class

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using WebApi.Models;

namespace BookStoresWebAPI.Models

{

public partial class RefreshToken

{

[Key]

public int TokenId { get; set; }

public int UserId { get; set; }

public string Token { get; set; }

public DateTime ExpiryDate { get; set; }

public virtual User User { get; set; }

}

}

In User Class

using System;

using System.Collections.Generic;

using WebApi.Models;

namespace BookStoresWebAPI.Models

{

public partial class User

{

public User()

{

RefreshTokens = new HashSet<RefreshToken>();

}

public int UserId { get; set; }

public string EmailAddress { get; set; }

public string Password { get; set; }

public string? FirstName { get; set; }

public string? LastName { get; set; }

public short RoleId { get; set; }

public DateTime? HireDate { get; set; }

public virtual Role? Role { get; set; }

public virtual ICollection<RefreshToken>? RefreshTokens { get; set; }

}

}

Now when the user logins, we have to return both jwtToken and RefreshToken. RefreshToken will be saved in database

private RefreshToken GenerateRefreshToken()

{

RefreshToken refreshToken = new RefreshToken();

var randomNumber = new byte[32];

using (var rng = RandomNumberGenerator.Create())

{

rng.GetBytes(randomNumber);

refreshToken.Token = Convert.ToBase64String(randomNumber);

}

refreshToken.ExpiryDate = DateTime.UtcNow.AddMonths(6);

return refreshToken;

}

If user is found,

if (user != null)

{

RefreshToken refreshToken = GenerateRefreshToken();

user.RefreshTokens.Add(refreshToken);

await \_context.SaveChangesAsync();

userWithToken = new UserWithToken(user);

userWithToken.RefreshToken = refreshToken.Token;

}

//sign your token here here..

userWithToken.AccessToken = GenerateAccessToken(user.UserId);

return userWithToken;

private string GenerateAccessToken(int userId)

{

var tokenHandler = new JwtSecurityTokenHandler();

var key = Encoding.UTF8.GetBytes(\_jwtsettings["Jwt:Key"]);

//var key = Encoding.UTF8.GetBytes(\_jwtsettings["[Jwt:Key"]);

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(new Claim[]

{

// Added UserID Here

new Claim(ClaimTypes.Name, Convert.ToString(userId))

}),

Expires = DateTime.UtcNow.AddDays(1),

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key),

SecurityAlgorithms.HmacSha256Signature)

};

var token = tokenHandler.CreateToken(tokenDescriptor);

return tokenHandler.WriteToken(token);

}

UserWIthToken Class

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace BookStoresWebAPI.Models

{

public class UserWithToken : User

{

public string AccessToken { get; set; }

public string RefreshToken { get; set; }

public UserWithToken(User user)

{

this.UserId = user.UserId;

this.EmailAddress = user.EmailAddress;

this.FirstName = user.FirstName;

this.LastName = user.LastName;

this.HireDate = user.HireDate;

this.Role = user.Role;

}

}

}

Add RefreshToken Api

[HttpPost("RefreshToken")]

public async Task<ActionResult<UserWithToken>> RefreshToken([FromBody] RefreshRequest refreshRequest)

{

User user = await GetUserFromAccessToken(refreshRequest.AccessToken);

if (user != null && ValidateRefreshToken(user, refreshRequest.RefreshToken))

{

UserWithToken userWithToken = new UserWithToken(user);

userWithToken.AccessToken = GenerateAccessToken(user.UserId);

return userWithToken;

}

return null;

}

Add a class RefreshRequest

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace BookStoresWebAPI.Models

{

public class RefreshRequest

{

public string AccessToken { get; set; }

public string RefreshToken { get; set; }

}

}

This Api will do 2 things

1. First get user from GetUserFromAccessToken, here pass the Req
2. If the user is not null, validat it which takes user & refreshToken

private async Task<User> GetUserFromAccessToken(string accessToken)

1. {
2. try
3. {
4. var tokenHandler = new JwtSecurityTokenHandler();
5. //var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_config["Jwt:Key"]));
6. var key = Encoding.UTF8.GetBytes(\_jwtsettings["Jwt:Key"]);
7. var tokenValidationParameters = new TokenValidationParameters
8. {
9. ValidateIssuerSigningKey = true,
10. IssuerSigningKey = new SymmetricSecurityKey(key),
11. ValidateIssuer = false,
12. ValidateAudience = false
13. };
14. SecurityToken securityToken;
15. var principle = tokenHandler.ValidateToken(accessToken, tokenValidationParameters, out securityToken);
16. JwtSecurityToken jwtSecurityToken = securityToken as JwtSecurityToken;
17. if (jwtSecurityToken != null && jwtSecurityToken.Header.Alg.Equals(SecurityAlgorithms.HmacSha256, StringComparison.InvariantCultureIgnoreCase))
18. {
19. var userId = principle.FindFirst(ClaimTypes.Name)?.Value;
20. return await \_context.Users.Where(u => u.UserId == Convert.ToInt32(userId)).FirstOrDefaultAsync();
21. }
22. }
23. catch (Exception)
24. {
25. return new User();
26. }
27. return new User();
28. }

private bool ValidateRefreshToken(User user, string refreshToken)

{

RefreshToken refreshTokenUser = \_context.RefreshTokens.Where(rt => rt.Token == refreshToken)

.OrderByDescending(rt => rt.ExpiryDate)

.FirstOrDefault();

if (refreshTokenUser != null && refreshTokenUser.UserId == user.UserId

&& refreshTokenUser.ExpiryDate > DateTime.UtcNow)

{

return true;

}

return false;

}