11. Авторизация  
www.youtube.com/watch?v=8ZtInClXe1Q

Зачем нужна авторизация в данном проекте? Чтобы люди могли добавлять на сайт статьи от своего имени.

Какие основные компоненты авторизации?

1. База данных. Тут всё понятно: нужна таблица, которая будет нести имя пользователя и хэш пароля, а также соль (т.н. Salt, см. первый источник). Ей будет являться основная таблица с пользователями. EtityFramework’у будет указано, что email стоит использовать как ALTERNATE KEY поскольку нельзя создать несколько аккаунтов на один адрес электронной почты – это ускорит поиск по таблице, т.к. будет создан хэш-мап.

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Website.Models.DocumentModel.DbDocument>()

.HasGeneratedTsVectorColumn(p => p.TitleTsVector, "english", p => new { p.Title })

.HasIndex(p => p.TitleTsVector)

.HasMethod("GIN");

modelBuilder.Entity<Website.Models.UserModel.User>()

.HasAlternateKey(u => u.EmailAdress);

}

2. Микросервис проверки и создания паролей. Проще говоря, программный класс, который занимается хешированием паролей по случайному ключу (т.н. Salt, см. первый источник) а также их проверкой, он уже написан на момент написания документа, несет в себе следующее (часть кода удалена):

public class PasswordsService

{

private const int KeySizeBits = 512;

private Func<byte[], byte[]> HashAlg = SHA512.HashData;

private static byte[] GenerateSalt()

{

var Bits = new byte[KeySizeBits / 8]; // /8 = bytes

System.Security.Cryptography.RandomNumberGenerator.Create().GetBytes(Bits);

return Bits;

}

public byte[] HashPassword(string PlainTextPassword, out byte[] GeneratedSalt)

{

var Salt = GenerateSalt();

var SaltedPassword = new byte[PlainTextPassword.Length + Salt.Length];

for (int i = 0; i < PlainTextPassword.Length; i++)

SaltedPassword[i] = (byte)(PlainTextPassword[i]);

for (int i = 0; i < Salt.Length; i++)

SaltedPassword[PlainTextPassword.Length + i] = Salt[i];

GeneratedSalt = Salt;

return HashAlg(SaltedPassword); // should better use PBKDF2 ?

}

public bool ConfirmPassword(string PlainTextPassword, byte[] HashedPassword, byte[] Salt)

{

var SaltedPossiblePassword = new byte[PlainTextPassword.Length + Salt.Length];

for (int i = 0; i < PlainTextPassword.Length; i++)

SaltedPossiblePassword[i] = (byte)(PlainTextPassword[i]);

for (int i = 0; i < Salt.Length; i++)

SaltedPossiblePassword[PlainTextPassword.Length + i] = Salt[i];

return HashAlg(SaltedPossiblePassword).SequenceEqual(HashedPassword);

}  
}

3. Менеджер сессий. Пожалуй, я упрощу себе задачу и не буду создавать реалистичного (т.е. такого, какого я бы писал в коммерческом проекте) менеджера, который не будет убивать старые сессии, а будет сохранять их в БД. Я буду просто убивать каждую сессию после суток неактивности. Для небольшого сайта этого может быть достаточно, сущность сессии достаточно легковесна, скорее всего она будет нести в себе только идентификатор и id пользователя, которому она принадлежит. Даже если на сайт ежедневно заходит 10 тысяч человек, пусть идентификатор сессии будет 100-символьной строкой – 100 байт, а id пользователя – int – 4 байта, (100+4)\*10000 /1024 /1024 = 1 Мб – ничто для современных объемов ОЗУ. Также, ASP, естественно, имеет встроенного менеджера сессий, но его я применю в следующем своём проекте, в этом же я хочу удостовериться в своём понимании. Код следующий:

public class SessionInfo

{

public int UserId { get; set; }

public DateTime ValidThrough { get; set; }

public SessionInfo(int UserId, DateTime ValidThrough)

{

this.UserId = UserId;

this.ValidThrough = ValidThrough;

}

public bool IsValidNow => this.ValidThrough > DateTime.UtcNow;

}

public class SessionManager : BackgroundService

{

public bool ValidateSession(string SessionId)

{

SessionInfo FoundSession;

// if session does not exist

if (!this.Sessions.TryGetValue(SessionId, out FoundSession)) return false;

// if session is outdated

if ((FoundSession.ValidThrough.Ticks - DateTime.UtcNow.Ticks) < 0) return false;

// if session is valid, validate it for the next 24 hours (by default)

FoundSession.ValidThrough = DateTime.UtcNow.AddSeconds(SessionLifeTimeSecs);

return true;

}

public void CreateSession(int UsedId, out string CreatedSessionId)

{

var SessionId = new string(System.Security.Cryptography.RandomNumberGenerator

.GetBytes(SessionIdStringLength).Select(x => (char)x).ToArray());

var ValidThrough = DateTime.UtcNow.AddSeconds(this.SessionLifeTimeSecs);

CreatedSessionId = SessionId;

this.Sessions.Add(SessionId, new SessionInfo(UsedId, ValidThrough));

}

protected override async Task ExecuteAsync(System.Threading.CancellationToken ct)

{

while (!ct.IsCancellationRequested)

{

await Task.Delay(SessionsCleanUpIntervalSecs \* 1000);

this.Sessions = this.Sessions.Where(s => s.Value.IsValidNow)

.ToDictionary(key => key.Key, value => value.Value);

}

}

}

4. Добавление сессии в куки. Напишем абстрактный контроллер, от которого наследуем все контроллеры, которые так или иначе работают с авторизацией.

public abstract class AControllerWithAuth : Controller

{

private WebsiteContext DbCtx;

protected readonly SessionManager SM;

protected Website.Models.UserModel.User? AuthedUser { get; set; } = null;

public AControllerWithAuth(SessionManager SM, WebsiteContext ctx)

{

this.DbCtx = ctx;

this.SM = SM;

ViewBag.AuthedUser = null;

}

public override void OnActionExecuting(ActionExecutingContext context)

{

base.OnActionExecuting(context);

if (Request.Cookies.ContainsKey(SessionManager.SessionIdCoockieName))

{

SessionInfo? Info;

#pragma warning disable CS8604

SM.ValidateSession(Request.Cookies[SessionManager.SessionIdCoockieName], out Info);

#pragma warning restore CS8604

this.AuthedUser = DbCtx.Users.Find(Info?.UserId) ?? null;

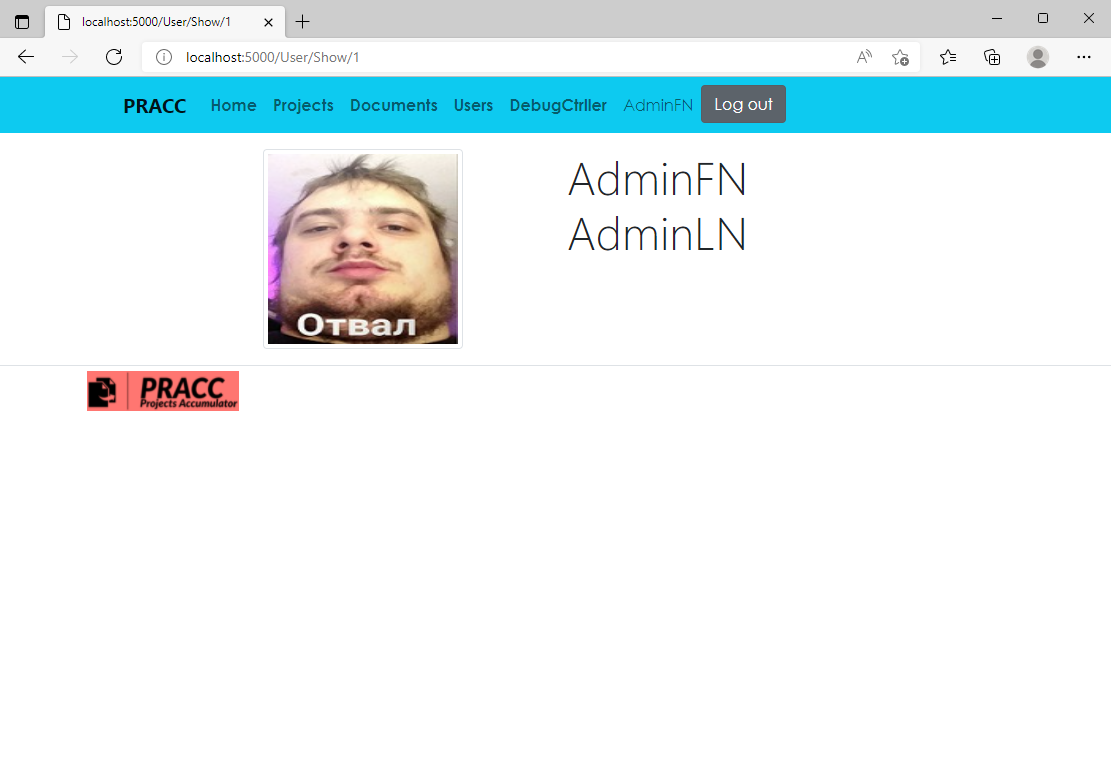
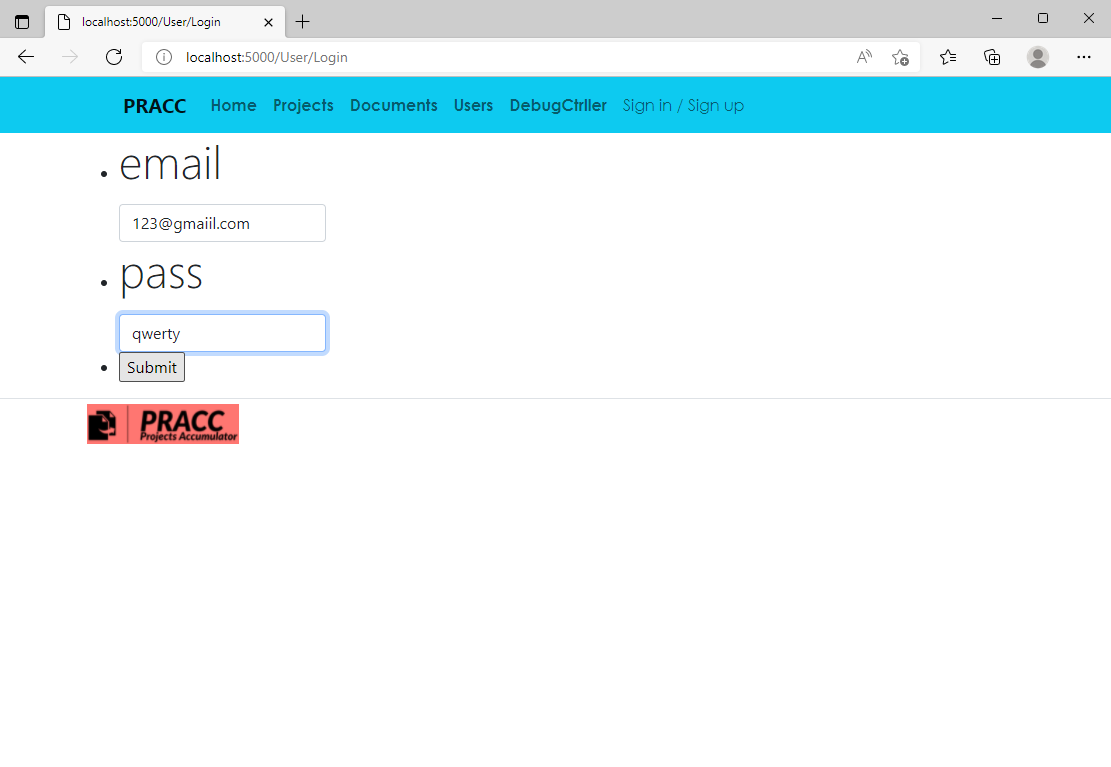
ViewBag.AuthedUser = this.AuthedUser;

}

}

}

Визуал на данный момент выглядит далеко не приятно, но он и был написан только с целью произвести ручное тестирование:



Также реализована регистрация, но здесь она не будет освещаться в связи с её простотой.