Лабораторная работа №4

ASP.NET Core Identity

1. Цель работы.

Знакомство с механизмом аутентификации и авторизации.

2. Общие сведения.

ASP.NET Core Identity – это система членства, позволяющая регистрировать учетные записи пользователей, регистрировать роли и назначать роли пользователям для реализации механизма аутентификации и авторизации.

Базовый набор интерфейсов, используемых в системе аутентификации находятся в пространстве имен Microsoft.AspNetCore.Identity.

Конкретная реализация интерфейсов Identity на базе Entity Framework Core находится в пространстве имен Microsoft.AspNetCore.Identity.EntityFrameworkCore.

Компоненты авторизации, находятся в пространстве имен Microsoft.AspNetCore.Authorization.

Основные классы, описанные в пространстве имен Microsoft.AspNetCore.Identity:

- IdentityUser – описывает пользователя;
- IdentityRole – описывает роль;
- UserManager – управляет пользователями (добавление, удаление, поиск, назначение роли и т.д.);
- RoleManager - управляет пользователями (добавление, удаление, поиск, роли и т.д.);
- SignInManager – реализует функции входа в/выхода из системы пользователя

Для использования механизма аутентификации необходимо добавить сервис аутентификации в метод ConfigureServices() и добавить аутентификацию в конвейер MiddleWare в методе Configure():

```
services.AddDbContext<ApplicationDbContext>(options =>
                options.UseSqlServer(

Configuration.GetConnectionString("DefaultConnection")));
            services.AddIdentity<ApplicationUser, IdentityRole>(opt=>
            {
                opt.Password.RequireLowercase = false;
                opt.Password.RequireNonAlphanumeric = false;
                opt.Password.RequireUppercase = false;
                opt.Password.RequireDigit = false;
            })
                .AddDefaultUI(UIFramework.Bootstrap4)
                .AddEntityFrameworkStores<ApplicationDbContext>()
                .AddDefaultTokenProviders();
```

3. Выполнение работы

3.1. Исходные данные

Используйте проект из лабораторной работы №3.

Из проекта удалите папку «Data».

3.2. Задание №1

Реализовать систему аутентификации с использованием Identity.

Приложение должно позволять:

- регистрироваться новому пользователю
- логиниться зарегистрированному пользователю.
- использовать роли для ограничения доступа
- использовать простые пароли

При создании базы данных предусмотреть:

- наличие роли «admin»
- наличие как минимум двух зарегистрированных пользователей, одному из которых назначена роль «admin»

Вместо класса **IdentityUser** используйте свой класс **ApplicationUser**, который должен наследоваться от класса IdentityUser.

Класс ApplicationUser и класс контекста должны быть описаны в отдельном проекте (библиотека классов .NET Core).

Выполните миграцию базы данных.

3.2.1. Рекомендации к заданию №1

Добавьте в решение новый проект – библиотеку классов .NET Core. Назовите проект XXX.DAL, где XXX – название вашего основного проекта. В основном проекте добавьте ссылку на созданный проект.

Добавьте в созданный проект следующие пакеты NuGet:

- Microsoft.AspNetCore.Identity.EntityFrameworkCore
- Microsoft.EntityFrameworkCore.SqlServer (для работы с SQL-сервером)

Поскольку в проекте уже используется имя ApplicationDbContext для контекста базы данных (см. файл Startup.cs), используйте это же имя для своего контекста.

В файле appsettings.json измените имя базы данных на любое удобное для вас имя.

Для заполнения базы начальными данными создайте класс DbInitializer со статическим методом, который и выполнит нужные действия. Вызывайте данный метод в методе Configure класса Startup, перед вызовом `app.UseEndpoints`.

В классе DbInitializer вам понадобятся объекты ApplicationDbContext, UserManager и RoleManager. Воспользуйтесь механизмом dependency injection для получения этих объектов. Предлагается два варианта.

Вариант 1. Передать эти объекты в метод Configure класса Startup, а затем передать в метод инициализации БД.

Вариант 2. В метод инициализации БД передать объект класса IApplicationBuilder. С его помощью можно получить нужные сервисы. Этот вариант более сложный, т.к. нужные сервисы – это «scoped» сервисы, т.е. они

могут быть извлечены при обработке http-запроса. Поэтому сначала нужно создать «scope» и получить объект ServiceProvider следующим образом:

```
var serviceProvider = app.ApplicationServices
                                .CreateScope()
                                .ServiceProvider),
```

а уже из него получить нужные сервисы:

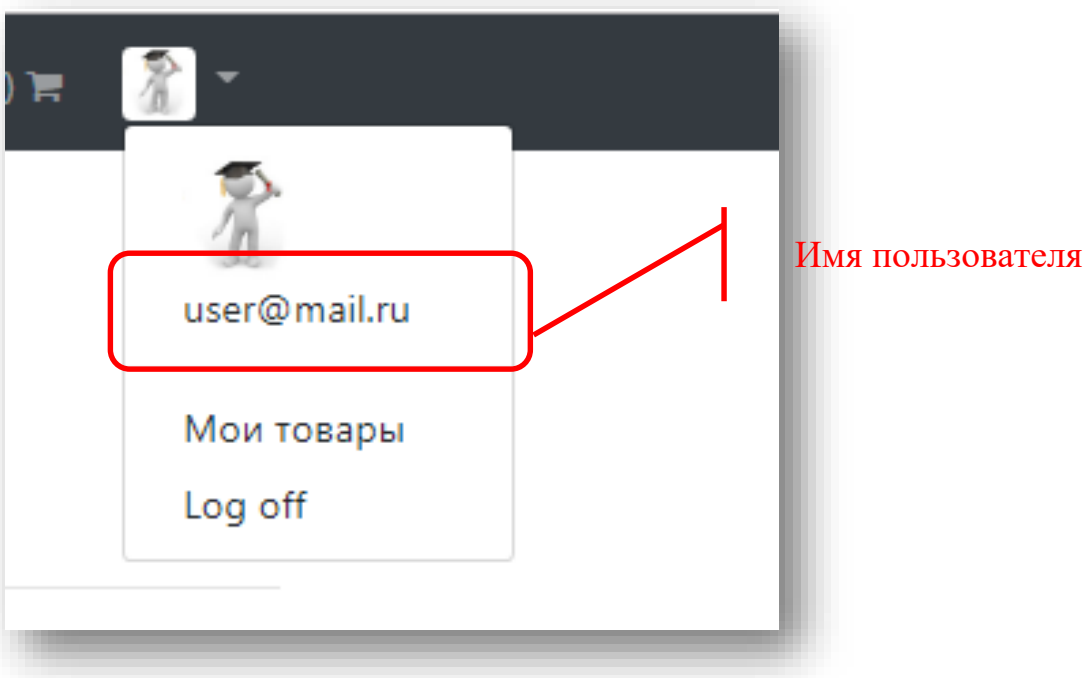
serviceProvider.GetService или serviceProvider.GetRequiredService.

Для создания страниц регистрации и входа в систему используйте scaffolding

3.3. Задание №2

Информация пользователя на странице приложения должна выводиться только если пользователь прошел аутентификацию. В противном случае должны выводиться ссылки «Войти» и «Зарегистрироваться»

В информации пользователя должно выводиться реальное имя пользователя:



Пункт меню «Log off» должен ссылаться на страницу «LogOut» (расположена по пути «Areas/Identity/Pages/Account»)

3.3.1. Рекомендации к заданию №2

Выполните внедрение (injection) в представление _UserPartial.cshtml класса SignInManager для проверки регистрации пользователя. В качестве примера можно использовать представление _LoginPartial.schtml, которое есть в проекте. Используйте `SignInManager<ApplicationUser>` вместо `SignInManager<IdentityUser>`

Имя пользователя можно извлечь из свойства User:

```
@User.Identity.Name
```

Откройте модель страницы Logout.cshtml. Выход происходит при запросе по методу Post. Поэтому пункт меню «Log off» нужно оформить в виде форма html (тэг <form>).

Для передачи параметра «returnurl» (адреса для возврата) используйте тэг-хелпер:

```
asp-route-returnurl="@ViewContext.HttpContext.Request.Path"
```

4. Контрольные вопросы

Что такое шаблон проектирования «Внедрение зависимостей»?

Как шаблон проектирования «Внедрение зависимостей» реализуется в ASP.NET MVC?

Как работает механизм привязки модели к параметрам методов контроллера?

Для чего используется библиотека Moq?

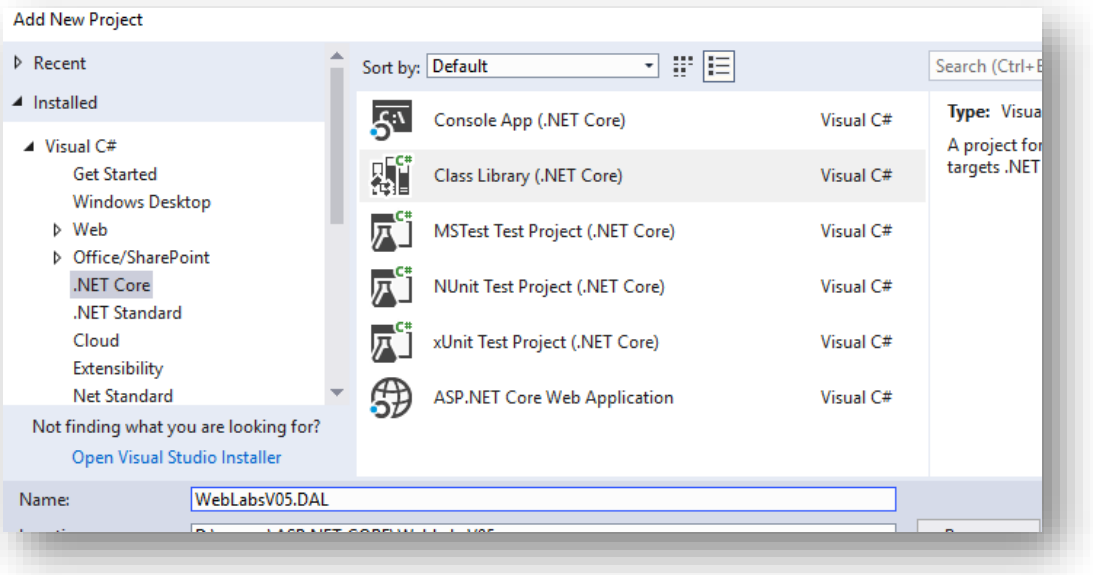
5. Пример выполнения работы

ВНИМАНИЕ:

- Проект, используемый в качестве примера, имеет название **WebLabsV05**. Следовательно, все пространства имен в примерах начинаются с WebLabsV05, например: WebLabsV05.DAL.
- Удалите из основного проекта папку Data

5.1. Добавление проекта

5.1.1. Добавьте в решение новый проект – библиотеку классов:



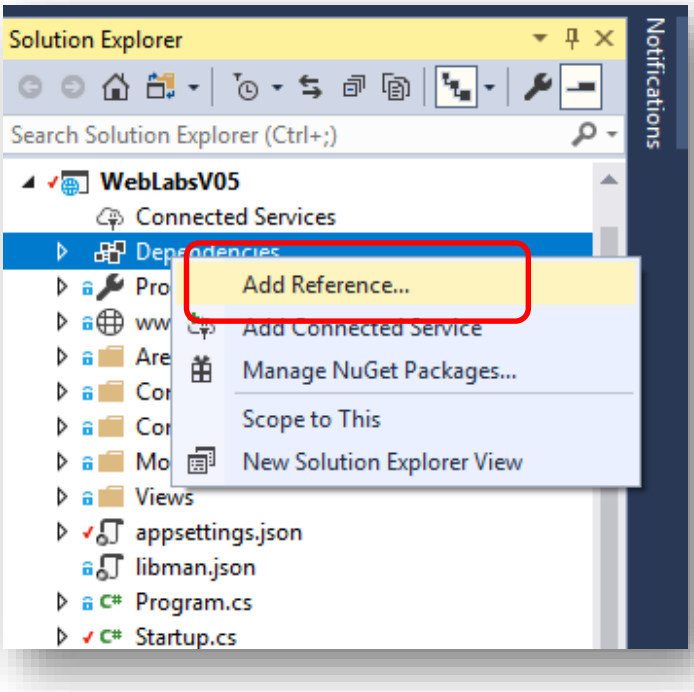
5.1.2. Добавьте в проект пакеты NuGet

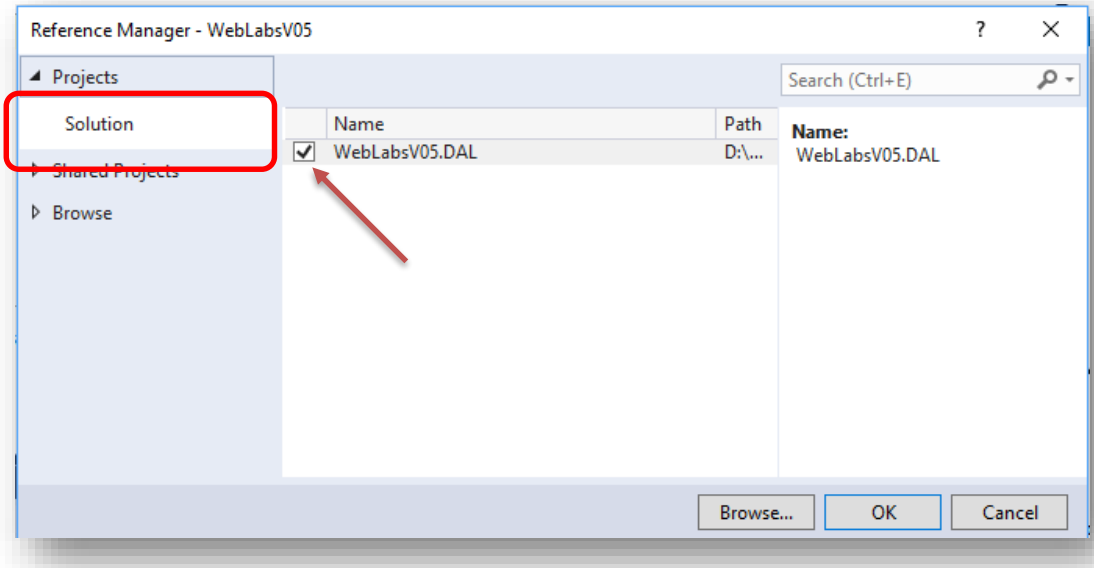
- Microsoft.AspNetCore.Identity.EntityFrameworkCore
- Microsoft.EntityFrameworkCore.SqlServer

5.1.3. Добавьте в проект папку Data

5.1.4. Добавьте в проект папку Entities

5.1.5. Сделайте ссылку в основном проекте на созданный проект:





5.2. Описание классов Identity

5.2.1. В папку Entities добавьте класс ApplicationUser:

```
using Microsoft.AspNetCore.Identity;
using System;
using System.Collections.Generic;
```

```
using System.Text;

namespace WebLabsV05.DAL.Entities
{
    public class ApplicationUser:IdentityUser
    {
    }
}
```

5.2.2. В папку Data добавьте класс ApplicationDbContext:

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace WebLabsV05.DAL.Data
{
public class ApplicationDbContext:IdentityDbContext<ApplicationUser>
    {
        public
ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {

        }
    }
}
```

5.3. Использование классов Identity в основном проекте

В файле Startup.cs добавьте ссылки на пространства имен XXX.DAL.Data и XXX.DAL.Entities:

```
using WebLabsV05.DAL.Data;
using WebLabsV05.DAL.Entities;
```

Измените настройку Identity для использования класса ApplicationUser, для возможности использования ролей и для возможности простых паролей:

```
services.AddIdentity<ApplicationUser, IdentityRole>(options =>
            {
                options.SignIn.RequireConfirmedAccount = false;
                options.Password.RequireLowercase = false;
                options.Password.RequireNonAlphanumeric = false;
                options.Password.RequireUppercase = false;
                options.Password.RequireDigit = false;
            })
      .AddEntityFrameworkStores<ApplicationDbContext>();
      .AddDefaultTokenProviders();
```

5.4. Изменение строки подключения

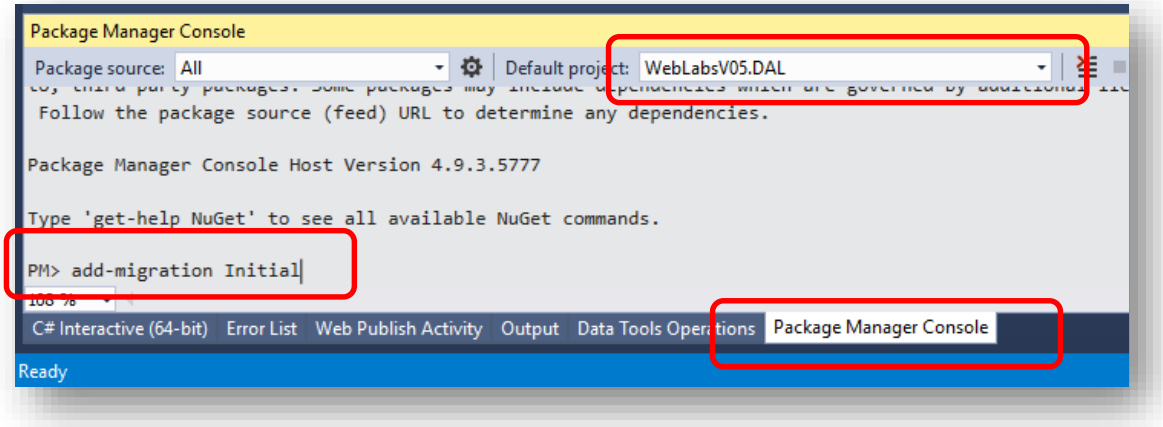
Измените строку подключения в файле appsettings.json основного проекта:

```
"ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;
Database=aspnet-WebLabsV05-Data;
Trusted_Connection=True;MultipleActiveResultSets=true"}
```

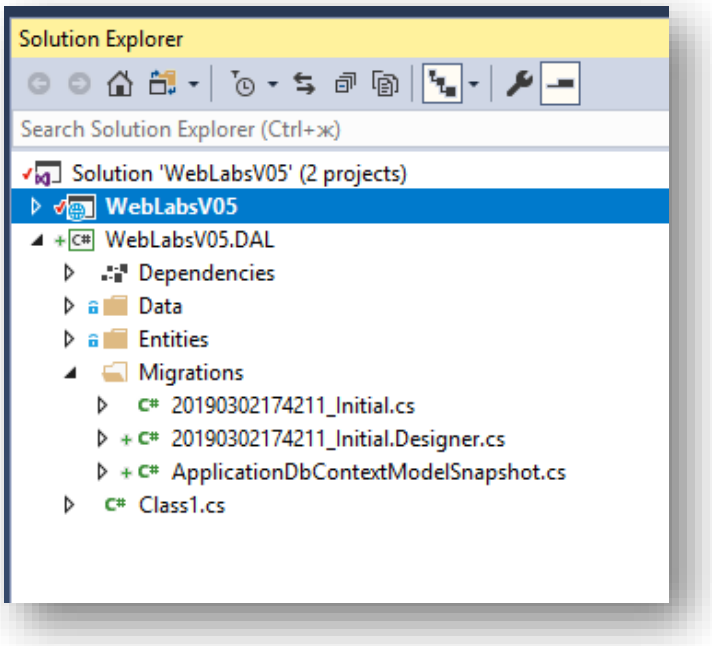
Строка подключения не должна иметь переводов строки. В приведенном примере строка разбита, т.к. не умещается по ширине страницы

5.5. Миграция базы данных

В консоли диспетчера пакетов выберите проект XXX.DAL и введите команду «add-migration Initial»

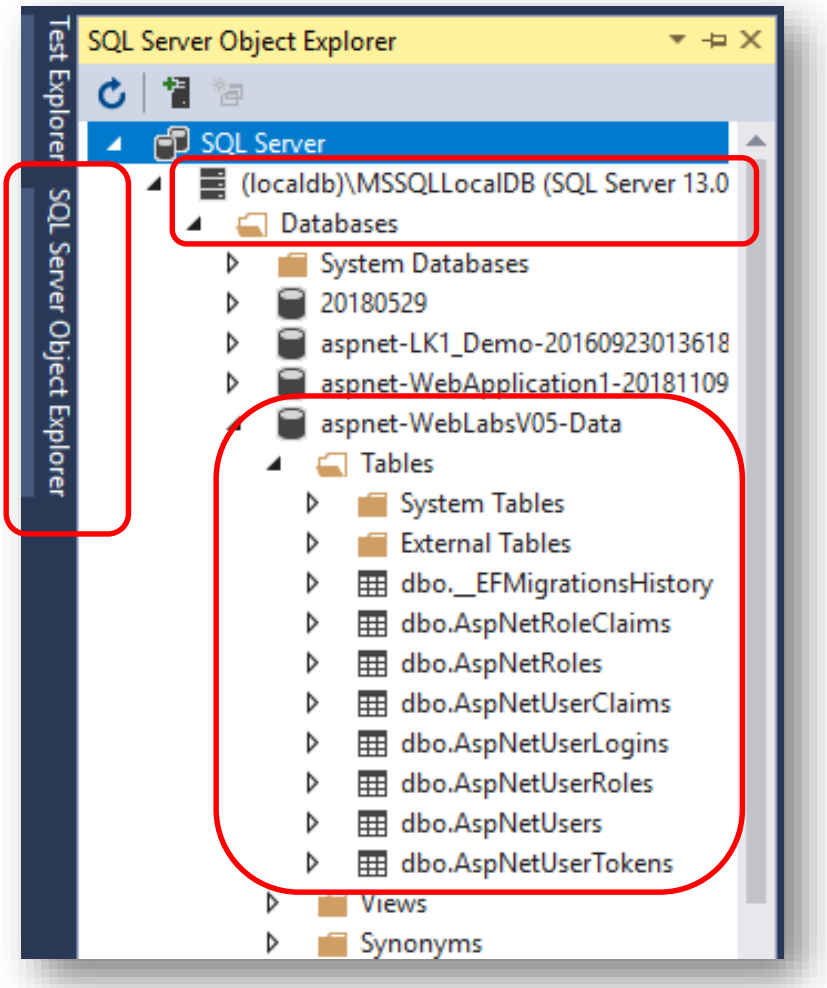


В проекте появляется папка миграций:



В консоли диспетчера пакетов введите команду «update-database».

В окне «SQL Server Object Explorer» убедитесь, что база данных создана:



5.6. Создание класса инициализации базы данных

В основной проект добавьте папку Services.

В полученную папку добавьте класс DbInitializer.

Опишите метод Seed(), который наполнит базу начальными данными:

```
public static async Task Seed(ApplicationDbContext context,
                              UserManager<ApplicationUser> userManager,
                              RoleManager<IdentityRole> roleManager)
{
      // создать БД, если она еще не создана
      context.Database.EnsureCreated();

      // проверка наличия ролей
      if (!context.Roles.Any())
      {
            var roleAdmin = new IdentityRole
            {
                  Name = "admin",
                  NormalizedName = "admin"
            };
            // создать роль admin
            await roleManager.CreateAsync(roleAdmin);
      }

      // проверка наличия пользователей
      if (!context.Users.Any())
      {
            // создать пользователя user@mail.ru
            var user = new ApplicationUser
            {
                  Email = "user@mail.ru",
                  UserName = "user@mail.ru"
            };
            await userManager.CreateAsync(user, "123456");
            // создать пользователя admin@mail.ru
            var admin = new ApplicationUser
            {
                  Email = "admin@mail.ru",
                  UserName = "admin@mail.ru"
            };
            await userManager.CreateAsync(admin, "123456");
            // назначить роль admin
            admin = await userManager.FindByEmailAsync("admin@mail.ru");
            await userManager.AddToRoleAsync(admin, "admin");
      }
}
```

Внедрите в метод Configure класса Startup объекты классов ApplicationDbContext, UserManager и RoleManager:

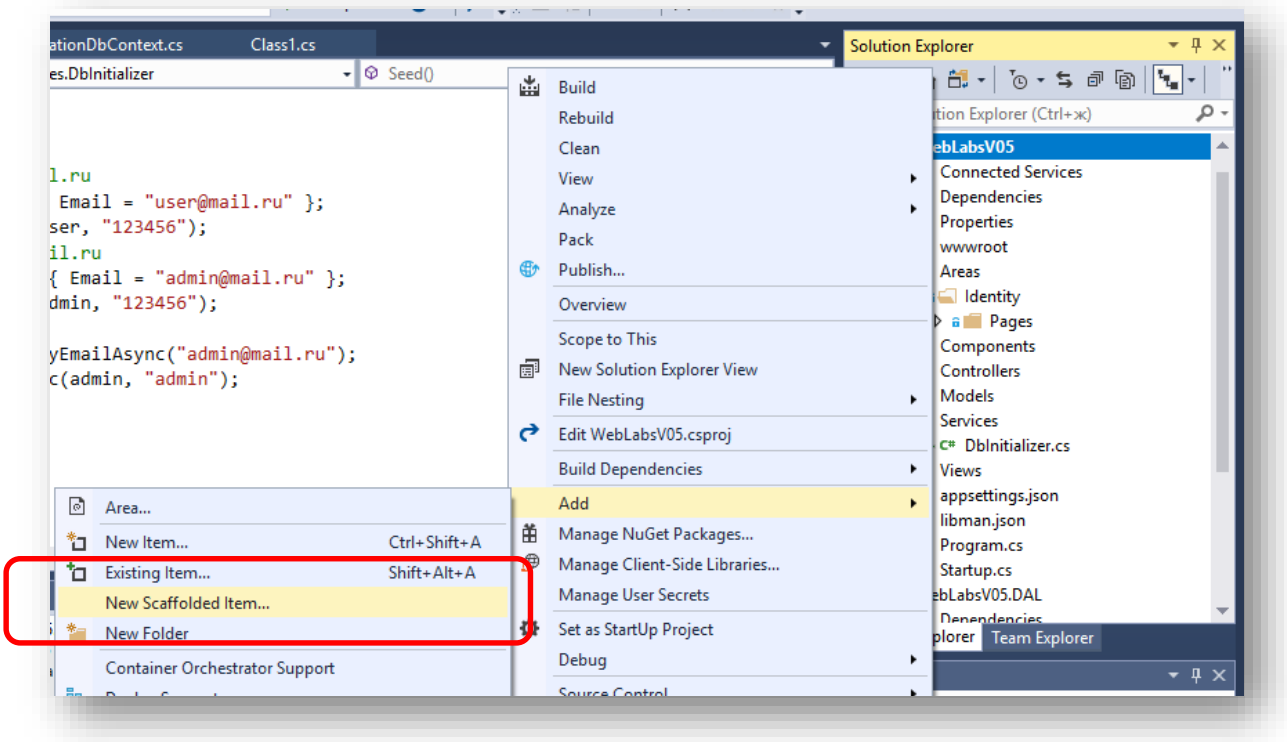
```
public void Configure(IApplicationBuilder app,
                      IWebHostEnvironment env,
                      ApplicationDbContext context,
                      UserManager<ApplicationUser> userManager,
                      RoleManager<IdentityRole> roleManager)
{
. . .
}
```

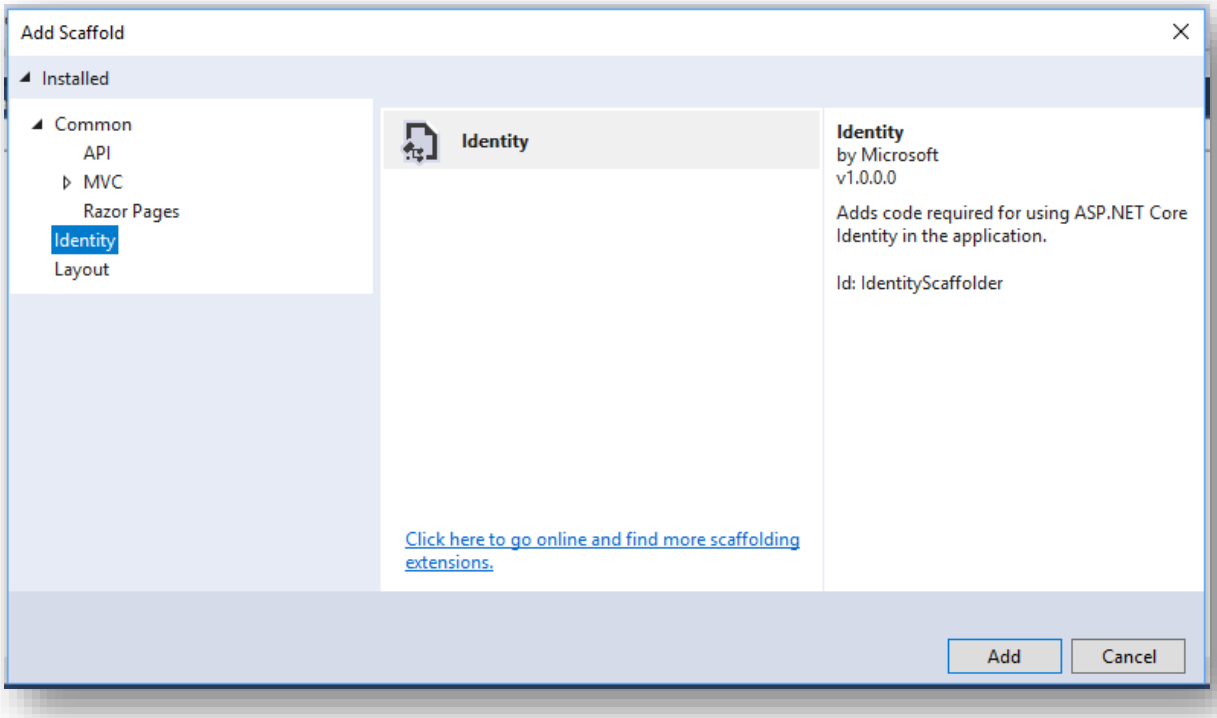
Вызовите метод Seed в методе Configure класса Startup:

```
DbInitializer.Seed(context, userManager, roleManager)
                .GetAwaiter()
                .GetResult();
```

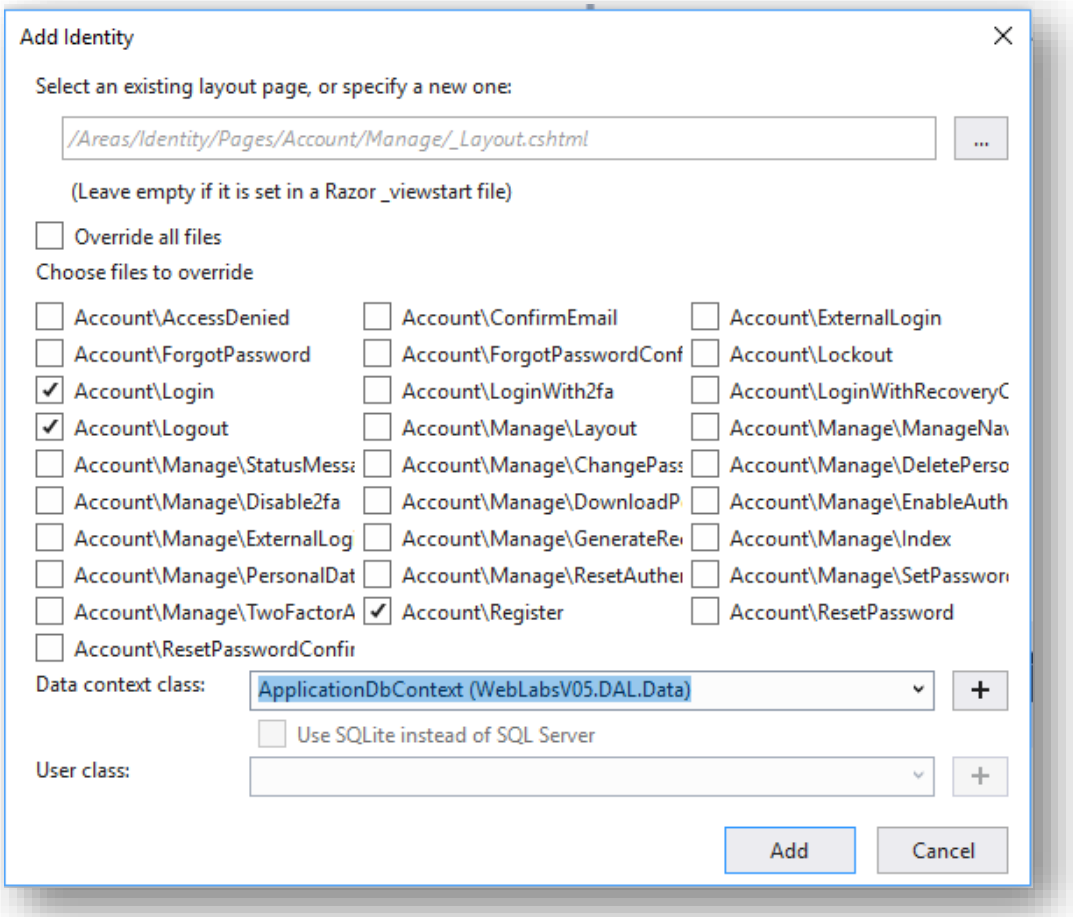
5.7. Создание страниц аутентификации

Добавьте в основной проект «Scaffolded item» - страницы Identity:

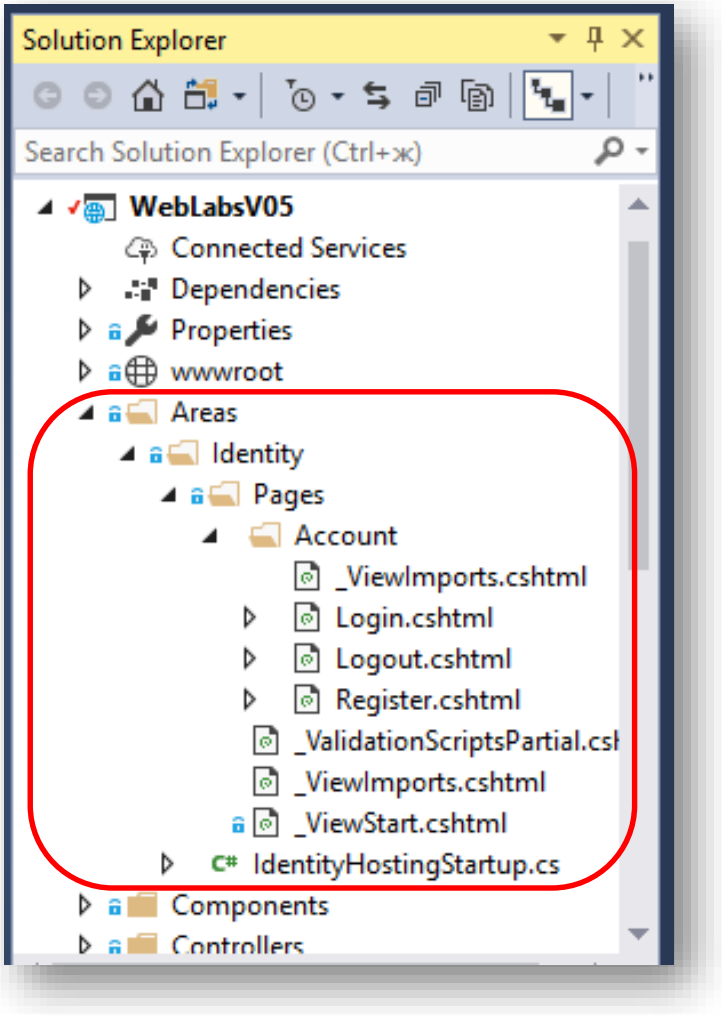




Выберите страницы Login, Logout и Register. Укажите ваш контекст базы данных:



Убедитесь, что в проекте появились сгенерированные страницы:

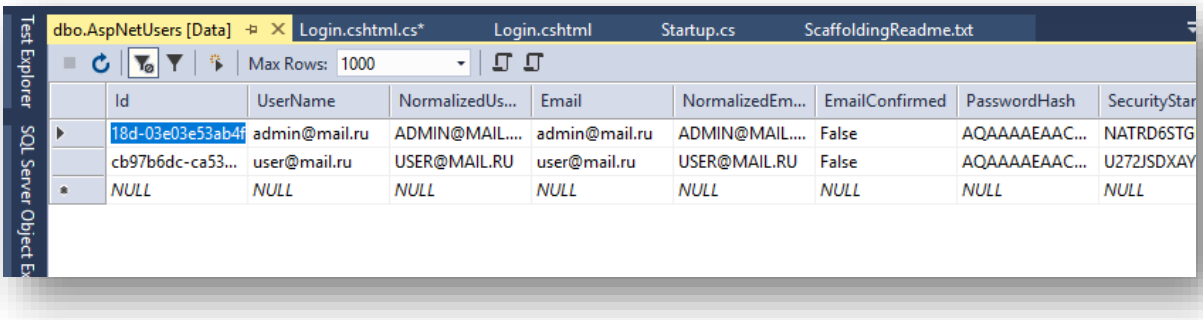


Удалите или закомментируйте в коде модели страницы Register.cshtml.cs строки кода, в которых используется IEmailSender.

В классе Startup настройте пути к созданным страницам в куки аутентификации:

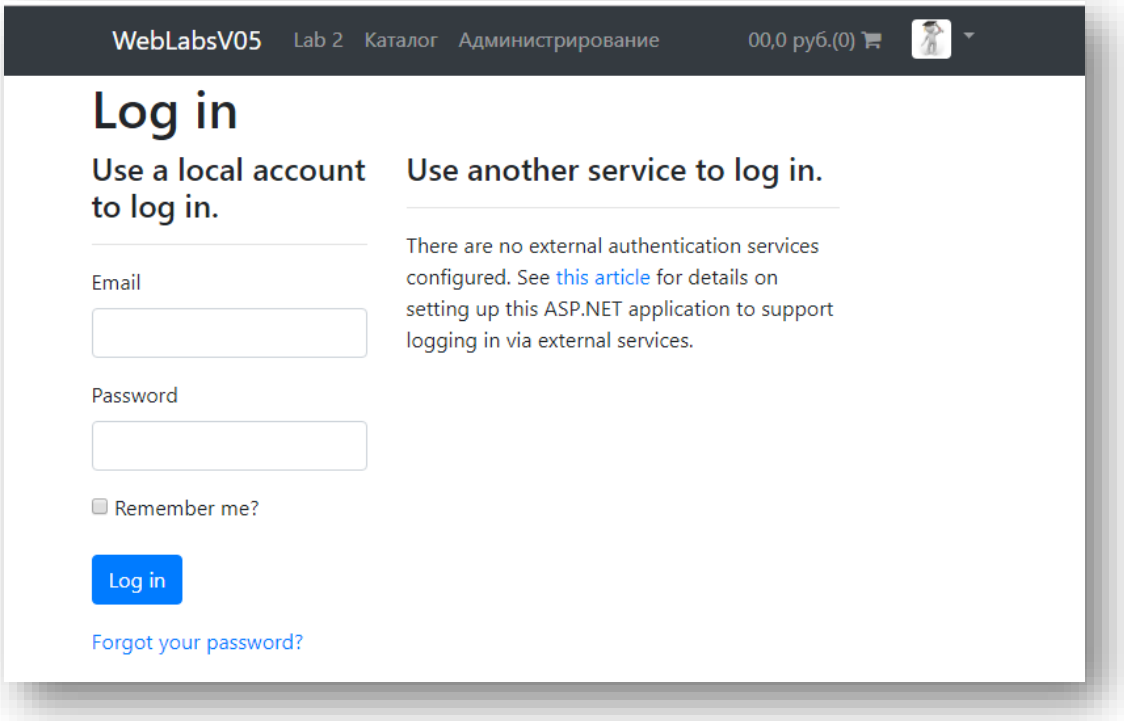
```
services.ConfigureApplicationCookie(options =>
            {
                options.LoginPath = $"/Identity/Account/Login";
                options.LogoutPath = $"/Identity/Account/Logout";
            });
```

Запустите проект. Убедитесь, что в базе данных появились записи в таблицах:



Введите в адресной строке https://XXX/identity/account/login.

Убедитесь, что отобразилась страница ввода логина и пароля:



5.8. Изменение панели информации пользователя

В начале представления _UserPartial добавьте нужные библиотеки:

```
@using Microsoft.AspNetCore.Identity;
@using WebLabsV05.DAL.Entities;
@inject SignInManager<ApplicationUser> signInManager
```

В разметке добавьте проверку регистрации пользователя, вывод реального имени пользователя, переход на страницу Logout и вывод меню «Войти – Зарегистрироваться», если пользователь не прошел проверку:

```
@if (signInManager.IsSignedIn(User))
{
    @await Component.InvokeAsync("Cart")
```

```
    <div class="dropdown ml-4 nav-color">
        <div class="dropdown-toggle" id="dropdownMenuButton" data-
toggle="dropdown" aria-haspopup="true" aria-expanded="false">
            <img src="~/Images/avatar.jpg" width="30" alt="User"
class="rounded" />
        </div>
        <div class="dropdown-menu" aria-
labelledby="dropdownMenuButton">
            <div class="dropdown-item-text">
                <img src="~/images/avatar.jpg" width="50" alt="User"
class="rounded" />
                @User.Identity.Name
            </div>
            <div class="dropdown-divider"></div>
            <a class="dropdown-item" asp-controller="Product"
                                     asp-action="UserProducts">Мои
товары</a>
            <form asp-area="Identity"
                  asp-page="/Account/Logout"
                  asp-route-returnurl=
                        "@ViewContext.HttpContext.Request.Path">
                <input type="submit"
                     value="Log off"
                     class="dropdown-item" />
            </form>

        </div>
    </div>
}
else
{
    <ul class="nav navbar-nav ml-auto">
        <li><a class="nav-item nav-link"
                asp-area="Identity"
                asp-page="/Account/Login">
                    Войти
            </a></li>
        <li><a class="nav-item nav-link"
                asp-area="Identity"
                asp-page="/Account/Register">
                    Зарегистрироваться
            </a></li>
    </ul>
}
```

Вид меню для пользователя, не прошедшего проверку, приведен на рисунке:

Попробуйте войти в систему как user@mail.ru , пароль «123456» (такой пользователь зарегистрирован при инициализации базы данных (см. п. 5.6), а затем выйти из системы.

Зарегистрируйте нового пользователя.