

Лабораторная работа №4

ASP.NET Core Identity, работа с файлами

1. Цель работы.

Знакомство с механизмом аутентификации и авторизации.

Знакомство механизмом обмена файлами между клиентом и сервером

Знакомство с файловыми провайдерами ASP.Net Core.

2. Общие сведения.

2.1. ASP.NET Core Identity

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();
```

2.2. Работа с файлами

Статические файлы, такие как HTML, CSS, изображения и JavaScript, приложение ASP.NET Core может предоставлять непосредственно клиенту. Статические файлы как правило располагаются по пути Web Root:

<content-root>/wwwroot

В качестве **Content-root** обычно выбирается папка, в которой размещены все файлы проекта

Возможность работы со статическими файлами задается в методе Configure класса Startup:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment
env, ILoggerFactory loggerFactory)
{
    app.UseStaticFiles();
    app.UseIdentity();
    app.UseMvcWithDefaultRoute();
}
```

Пример доступа к файлу *<content-root>/wwwroot/css/site.css*:

```
<link asp-href-include="/css/site.css" rel="stylesheet" />
```

Для работы с файлами в ASP.NET Core используются поставщики файлов (file providers).

Поставщики файлов - это абстракция над файловыми системами. Основным интерфейсом является **IFileProvider**.

IFileProvider предоставляет методы для получения информации о файлах (**IFileInfo**), информации о каталоге (**IDirectoryContents**) и настройке уведомлений об изменениях (с использованием **IChangeToken**).

IFileInfo предоставляет методы и свойства отдельных файлов или каталогов. Он имеет два булевых свойства: **Exists** и **IsDirectory**, а также свойства, описывающие имя файла, длину (в байтах) и дату **LastModified**.

Читать из файла можно с помощью метода **CreateReadStream**.

Конкретной реализации интерфейса **IFileProvider** является **PhysicalFileProvider**, который обеспечивает доступ к физической файловой системе. Он оборачивает тип **System.IO.File** (для физического поставщика), просматривая все пути к каталогу и его дочерним элементам. Это ограничивает доступ только к определенному каталогу и его дочерним элементам, предотвращая доступ к файловой системе за пределами этой границы.

Готовые провайдеры для **Content_Root** и **Web_Root** можно получить из объекта **IHostingEnvironment**:

```
public SomeController(IHostingEnvironment env)
{
    var webFileProvider = env.WebRootFileProvider;
    var contentFileProvider = env.ContentRootFileProvider;
}
```

2.3. Передача файлов на сервер

Чтобы поддерживать загрузку файлов, HTML-форма должна иметь атрибут

```
enctype="multipart/form-data"
```

Доступ к отдельным файлам, загруженным на сервер, можно получить через привязку модели с использованием интерфейса **IFormFile**.

Интерфейс IFormFile описывает следующие методы и свойства:

```
public interface IFormFile
{
    string ContentType { get; }
    string ContentDisposition { get; }
    IDictionary Headers { get; }
    long Length { get; }
    string Name { get; }
    string FileName { get; }
    Stream OpenReadStream();
    void CopyTo(Stream target);
    Task CopyToAsync(Stream target,
        CancellationToken cancellationToken = null);
}
```

Пример сохранения файла в папке «wwwroot/Files»:

```
[HttpPost]
public async Task<IActionResult> Upload(
    [FromServices]IHostingEnvironment env,
    [FromForm]IFormFile uploadedFile)
{
    var path = env.WebRootPath + "/Files/" + uploadedFile.FileName;
    using (var stream = new FileStream(path, FileMode.Create))
    {
        await uploadedFile.CopyToAsync(stream);
    };
    return RedirectToAction("Index");
}
```

Пример сохранения файла в байтовый массив «byte[] AvatarImage»:

```
await uploadedFile
    .OpenReadStream()
    .ReadAsync(AvatarImage, 0, (int)uploadedFile.Length);
```

2.4. Передача файлов клиенту методом контроллера

Для отправки клиенту файлов предназначен абстрактный класс **FileResult**, который реализуется в классах:

- **FileContentResult**: отправляет клиенту массив байтов, считанный из файла;
- **VirtualFileResult**: представляет простую отправку файла напрямую с сервера по виртуальному пути;
- **FileStreamResult**: создает поток - объект System.IO.Stream, с помощью которого считывает и отправляет файл клиенту;
- **PhysicalFileResult**: для отправки используется реальный физический путь;

Пример отправки файла «Picture.jpg» из папки «wwwroot/images»:

```
public IActionResult GetImage([FromServices] IHostingEnvironment env)
{
    var provider = env.WebRootFileProvider;
    var path = Path.Combine("images", "Picture1.jpg");
    var fInfo = provider.GetFileInfo(path);
    var ext = Path.GetExtension(fInfo.Name);
    var extProvider = new FileExtensionContentTypeProvider();
    return File(fInfo.CreateReadStream(),
                extProvider.Mappings[ext]);
}
```

3. Выполнение работы

3.1. Исходные данные

Используйте проект из лабораторной работы №3.

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.

Выполните миграцию базы данных.

3.2.1. Рекомендации к заданию №1

Добавьте в проект папку «Entities». В ней опишите класс ApplicationUser

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

Удалите папку «Migrations» из папки «Data». Создайте новую миграцию.

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:

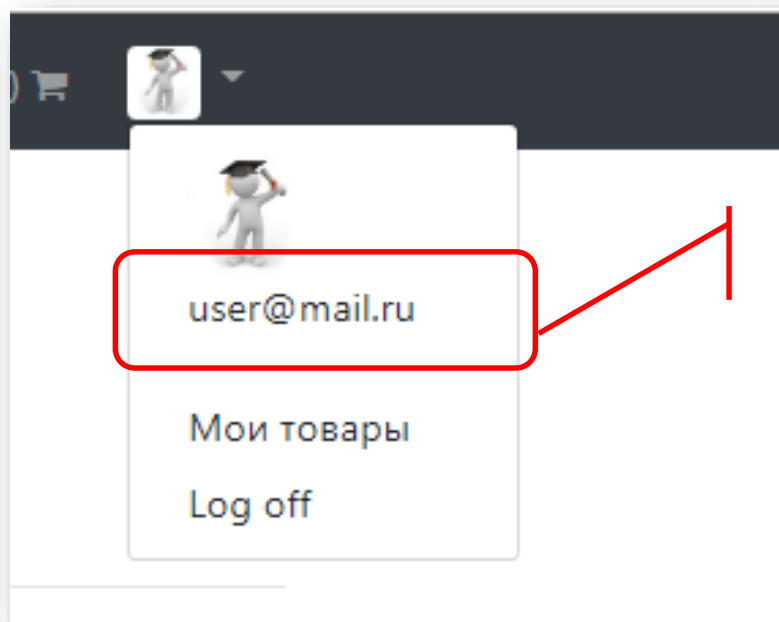
serviceProvider.GetService или serviceProvider.GetRequiredService.

Для создания страниц регистрации и входа в систему используйте scaffolding. В рамках лабораторных работ достаточно создать страницы Register, LogIn и LogOut.

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:



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.shtml`, 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 формы `html` (тэг `<form>`).

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

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

3.4. Задание №3

Добавьте в проект возможность загрузки аватара пользователя. Изображение аватара должно храниться в базе данных.

Выполните миграцию базы данных.

3.4.1. Рекомендации к заданию №3

Для хранения изображения в классе `ApplicationUser` добавьте свойство типа `byte[]`. Также можно добавить свойство, описывающее MIME-тип изображения.

Для получения MIME-типа изображения можно воспользоваться классом `FileExtensionContentTypeProvider`:

```
var extProvider = new FileExtensionContentTypeProvider();  
var mimeType = extProvider.Mappings[".png"];
```

3.5. Задание №4

На панели навигации, в меню пользователя должен отображаться аватар, сохраненный при регистрации пользователя. Если аватар отсутствует, то должен выводиться общий аватар из папки «`wwwroot/images`»

3.5.1. Рекомендации к заданию №4

Для передачи изображения создайте контроллер, метод `GetAvatar()` которого будет передавать аватар клиенту, а при его отсутствии – файл из

папки «images». Для получения данных пользователя понадобится внедрить в контроллер класс UserManager, а для доступа к папке wwwroot – объект IHostingEnvironment.

Для получения изображения в разметке в качестве значения атрибута *src* тэга *img* нужно указать адрес «Имя контроллера/GetImage». Для получения адреса воспользуйтесь вспомогательным методом @Url.Action.

Изображение общего аватара поместите в папку wwwroot/Images.

4. Контрольные вопросы

5. Пример выполнения работы

ВНИМАНИЕ:

- Проект, используемый в качестве примера, имеет название WebLabs_BSUIR_V01. Следовательно, все пространства имен в примерах начинаются с WebLabs_BSUIR_V01, например: WebLabs_BSUIR_V01.Data.

5.1. Описание классов Identity

5.1.1. Добавьте в проект папку Entities

5.1.2. В папку Entities добавьте класс ApplicationUser:

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

namespace WebLabs_BSUIR_V01.Entities
{
    public class ApplicationUser:IdentityUser
    {
    }
}
```

5.1.3. В папке Data измените класс ApplicationDbContext:

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

```

using Microsoft.EntityFrameworkCore;

namespace WebLabs_BSUIR_V01.Data
{
    public class ApplicationDbContext:IdentityDbContext<ApplicationUser>
    {
        public
        ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }
    }
}

```

5.2. Использование классов Identity в основном проекте

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

```

using WebLabs_BSUIR_V01.Data;
using WebLabs_BSUIR_V01.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();
services.AddAuthorization();

```

5.3. Изменение строки подключения

Измените строку подключения в файле appsettings.json основного проекта:

```

"ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;
    Database=WebLabs_BSUIR_V01-Data;
    Trusted_Connection=True;MultipleActiveResultSets=true"
}

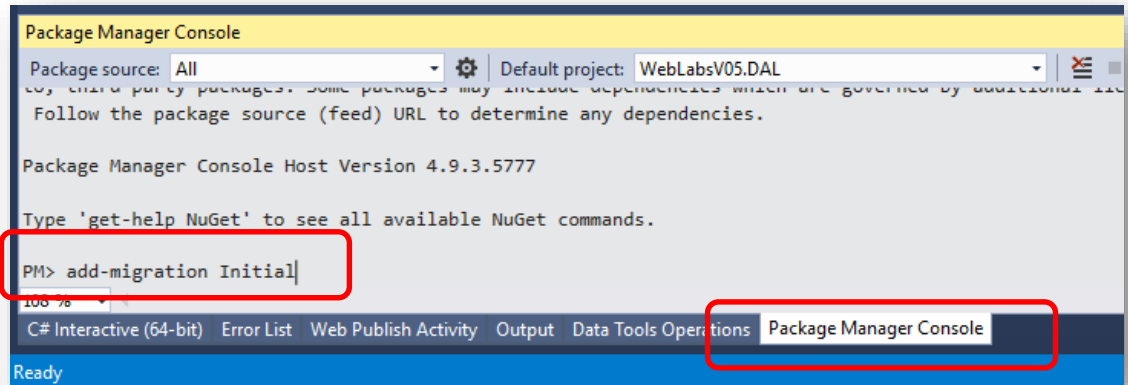
```

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

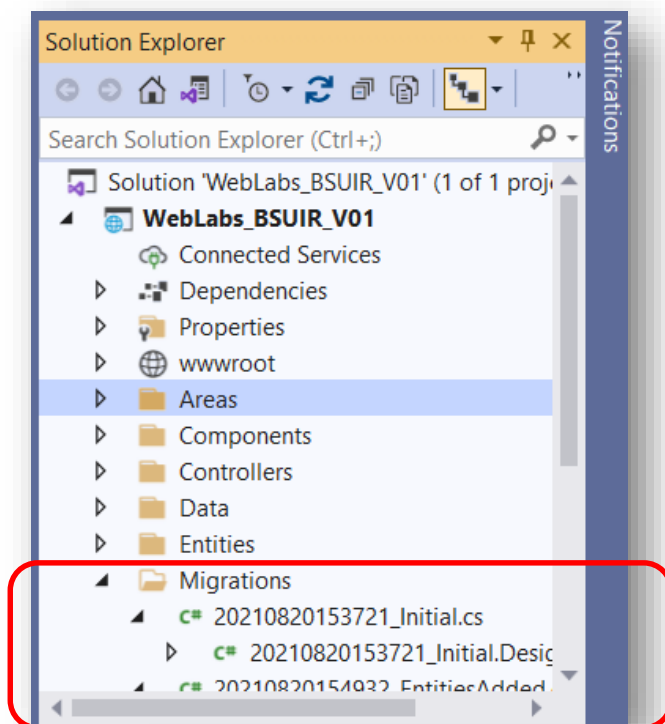
5.4. Миграция базы данных

Удалите папку «Migrations» из папки «Data».

В консоли диспетчера пакетов введите команду «add-migration Initial»

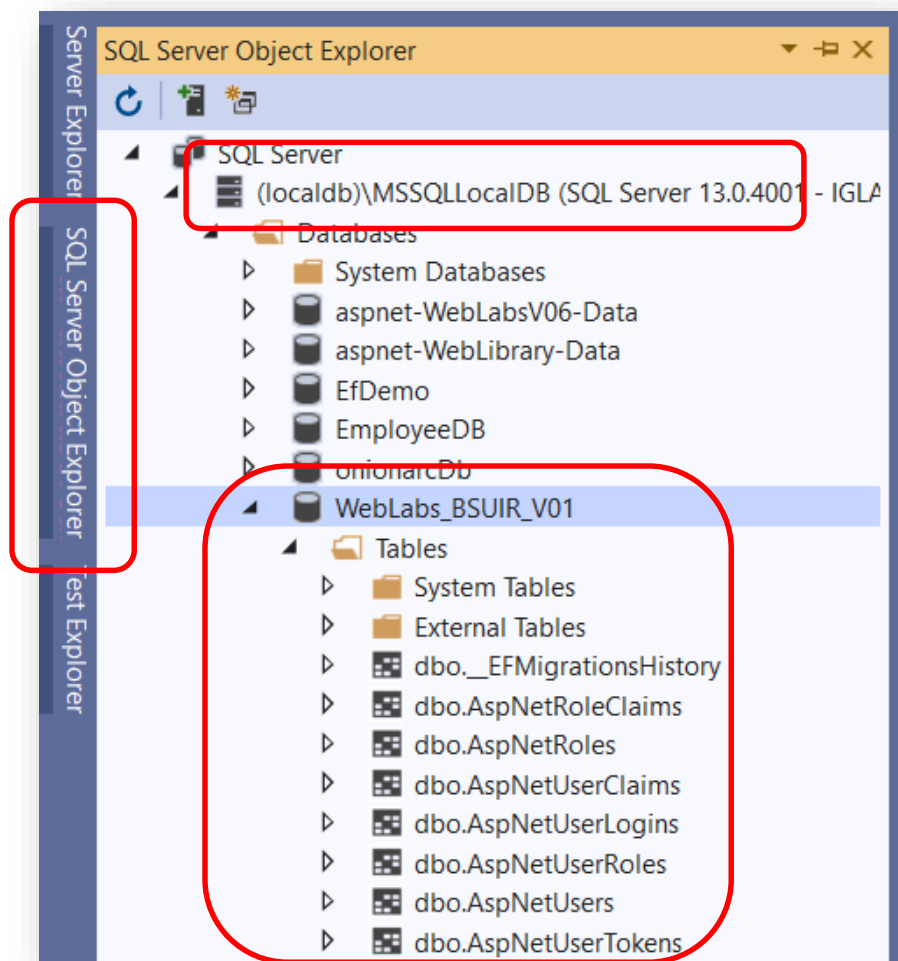


В проекте появляется папка миграций:



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

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



5.5. Создание класса инициализации базы данных

В папку «Data» 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, перед вызовом app.UseEndpoints:

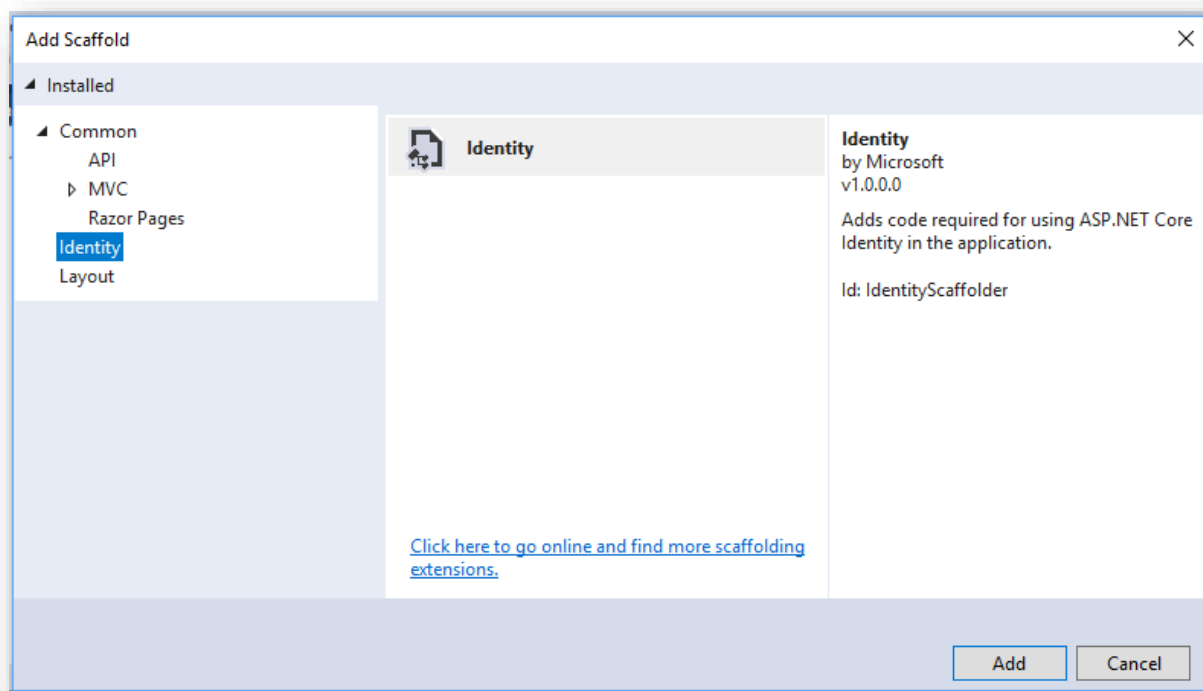
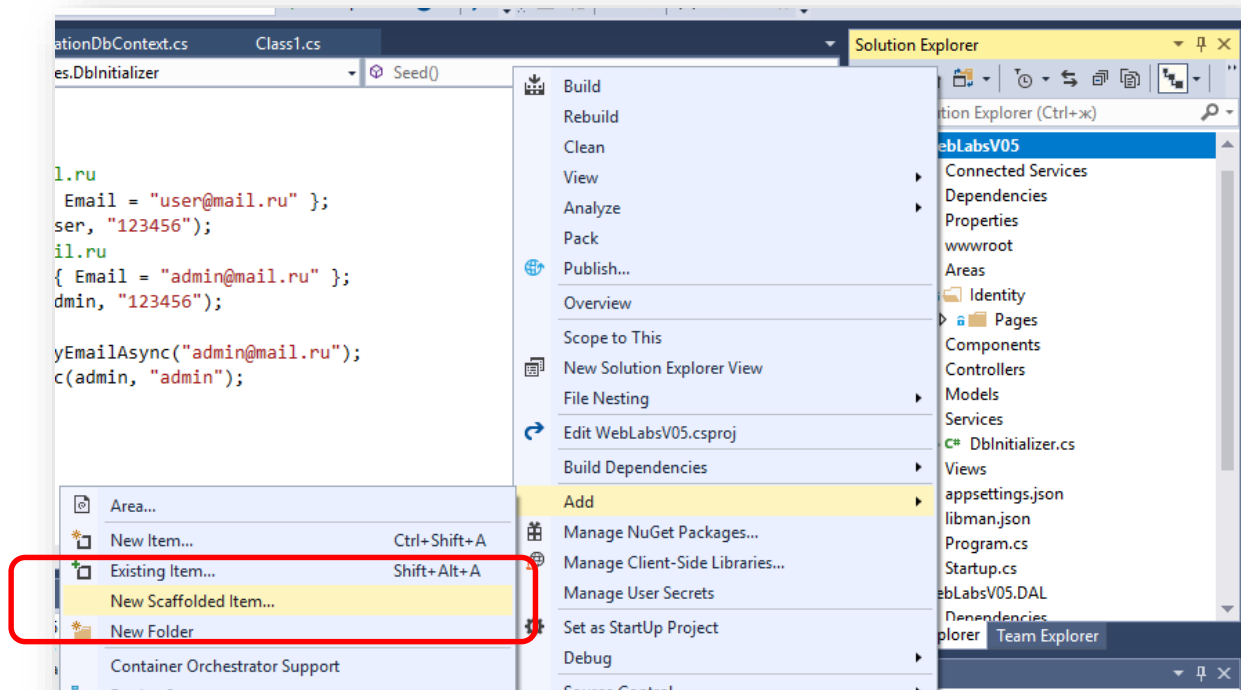
```

DbInitializer.Seed(context, userManager, roleManager).Wait();

```

5.6. Создание страниц аутентификации

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



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

Add Identity ✕

Select an existing layout page, or specify a new one:

...

(Leave empty if it is set in a Razor _viewstart file)

☐ Override all files

Choose files to override

<input type="checkbox"/> Account\AccessDenied	<input type="checkbox"/> Account\ConfirmEmail	<input type="checkbox"/> Account\ExternalLogin
<input type="checkbox"/> Account\ForgotPassword	<input type="checkbox"/> Account\ForgotPasswordConf	<input type="checkbox"/> Account\Lockout
<input checked="" type="checkbox"/> Account>Login	<input type="checkbox"/> Account>LoginWith2fa	<input type="checkbox"/> Account>LoginWithRecoveryC
<input checked="" type="checkbox"/> Account\Logout	<input type="checkbox"/> Account\Manage\Layout	<input type="checkbox"/> Account\Manage\ManageNav
<input type="checkbox"/> Account\Manage\StatusMess	<input type="checkbox"/> Account\Manage\ChangePass	<input type="checkbox"/> Account\Manage\DeletePerso
<input type="checkbox"/> Account\Manage\Disable2fa	<input type="checkbox"/> Account\Manage\DownloadP	<input type="checkbox"/> Account\Manage\EnableAuth
<input type="checkbox"/> Account\Manage\ExternalLog	<input type="checkbox"/> Account\Manage\GenerateRei	<input type="checkbox"/> Account\Manage\Index
<input type="checkbox"/> Account\Manage\PersonalDat	<input type="checkbox"/> Account\Manage\ResetAuther	<input type="checkbox"/> Account\Manage\SetPasswor
<input type="checkbox"/> Account\Manage\TwoFactorA	<input checked="" type="checkbox"/> Account\Register	<input type="checkbox"/> Account\ResetPassword
<input type="checkbox"/> Account\ResetPasswordConfir		

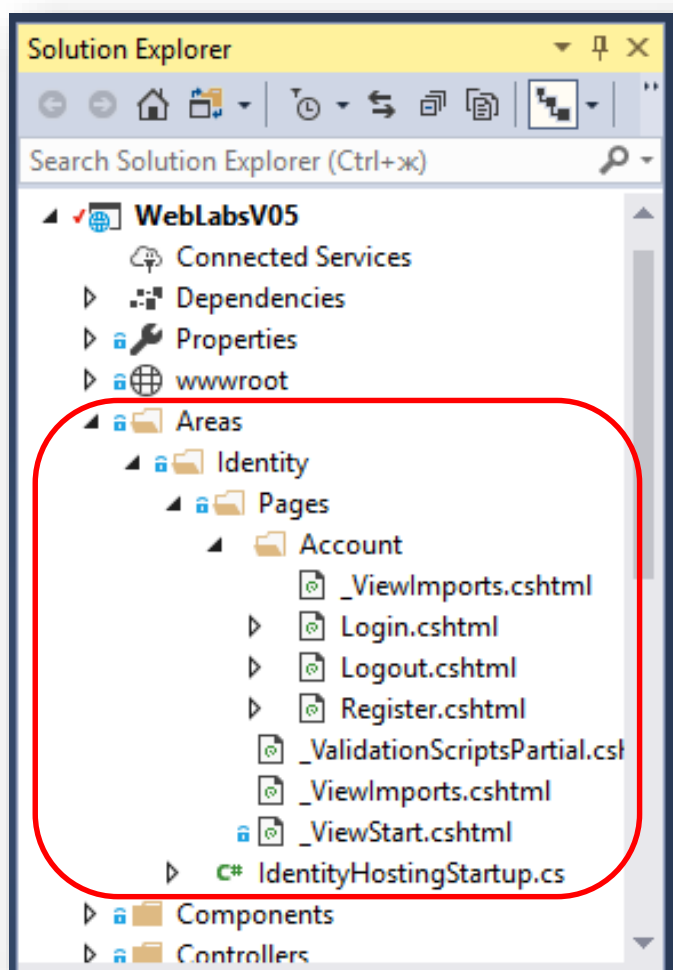
Data context class: +

☐ Use SQLite instead of SQL Server

User class: +

Add Cancel

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:

Test Explorer SQL Server Object Explorer


dbo.AspNetUsers [Data] Login.cshtml.cs* Login.cshtml Startup.cs ScaffoldingReadme.txt

Max Rows: 1000

	Id	UserName	NormalizedUs...	Email	NormalizedEm...	EmailConfirmed	PasswordHash	SecurityStar
	18d-03e03e53ab4f	admin@mail.ru	ADMIN@MAIL....	admin@mail.ru	ADMIN@MAIL....	False	AQAAAAEAAC...	NATRD6STG
	cb97b6dc-ca53...	user@mail.ru	USER@MAIL.RU	user@mail.ru	USER@MAIL.RU	False	AQAAAAEAAC...	U272JSDXAY
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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

Убедитесь, что отобразилась страница ввода логина и пароля:

WebLabsV05 Lab 2 Каталог Администрирование 00,0 руб.(0) 

Log in

Use a local account to log in.

Email

Password

☐ Remember me?

Log in

[Forgot your password?](#)

Use another service to log in.

There are no external authentication services configured. See [this article](#) for details on setting up this ASP.NET application to support logging in via external services.

5.7. Изменение панели информации пользователя

В начале представления `_UserPartial` добавьте нужные библиотеки:

```
@using Microsoft.AspNetCore.Identity;
@using WebLabs_BSUIR_V01.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">
                
            </div>
            <div class="dropdown-menu" aria-
labelledby="dropdownMenuButton">
                <div class="dropdown-item-text">
                    
                    @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. 5.5), а затем выйти из системы.

Зарегистрируйте нового пользователя.

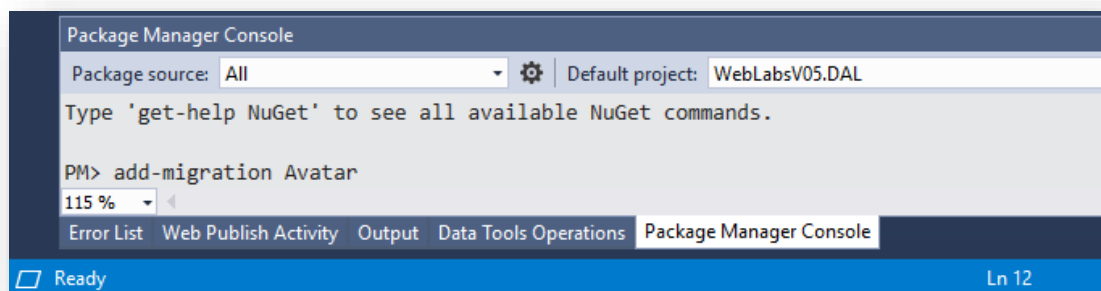
5.8. Добавление аватара пользователя

5.8.1. Изменение класса ApplicationUser

Добавьте класс ApplicationUser свойства, описывающие данные аватара и его Mime тип:

```
public class ApplicationUser:IdentityUser
{
    public byte[] AvatarImage { get; set; }
}
```

Выполните миграцию базы данных



Выполните команду «update-database»

Убедитесь, что в таблицеAspNetUsers появились новые поля.

5.8.2. Изменение страницы Register

В коде страницы внесите следующие изменения.

Добавьте в класс InputModel свойство:

```
public IFormFile Avatar { get; set; }
```

Измените метод OnPostAsync для сохранения изображения в базе данных:

```
var user = new ApplicationUser { UserName = Input.Email, Email = Input.Email };
```

```

if(Input.Avatar!=null)
{
    user.AvatarImage = new byte[(int)Input.Avatar.Length];
    await Input.Avatar
        .OpenReadStream()
        .ReadAsync(
            user.AvatarImage,
            0,
            (int)Input.Avatar.Length);
}

```

Измените страницу Register, чтобы пользователь при регистрации мог указать имя файла аватарки и отослать его на сервер:

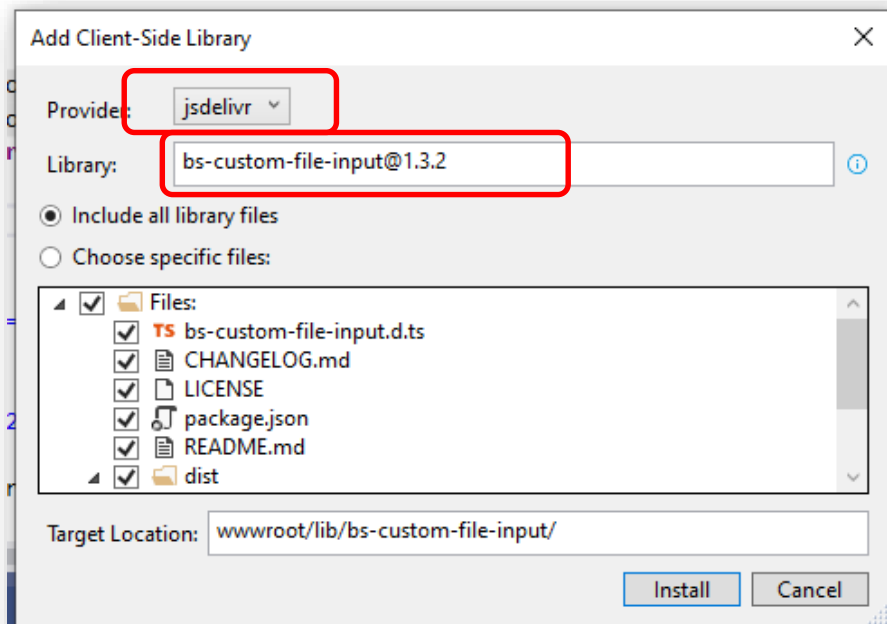
```

<form asp-route-returnUrl="@Model.ReturnUrl"
      method="post"
      enctype="multipart/form-data">
    <h4>Create a new account.</h4>
    . . .
    <div class="form-group">
        <label asp-for="Input.Avatar"></label>
        <input asp-for="Input.Avatar" type="file"
              class="form-control" />
    </div>
    <button type="submit"
            class="btn btn-primary">Register</button>
</form>

```

Вариант оформления с помощью библиотеки bs-custom-file-input (см. <https://getbootstrap.com/docs/4.4/components/forms/#file-browser>, <https://www.npmjs.com/package/bs-custom-file-input>)

Добавьте clien-side library:



В конце страницы, в секцию «Scripts», подключите загруженную библиотеку и активируйте ее:

```
@section Scripts {  
    <partial name=" ValidationScriptsPartial" />  
    <script src="~/lib/bs-custom-file-input/dist/bs-custom-file-  
input.js"></script>  
    <script>  
        $(document).ready(function() {  
            bsCustomFileInput.init()  
        })  
    </script>  
}
```

Оформите разметку для выбора аватарки:

```
<div class="form-group">  
    <label asp-for="Input.Avatar"></label>  
    <div class="custom-file">  
        <input asp-for="Input.Avatar" type="file" class="custom-  
file-input" id="customFile">  
        <label class="custom-file-label" for="customFile">Выберите  
файл</label>  
    </div>  
</div>
```

Зарегистрируйте пользователя с аватаром. Убедитесь, что данные сохраняются в базе данных.

5.9. Отображение аватара пользователя

5.9.1. Создание контроллера

В папке Controllers проекта создайте контроллер Image. Опишите метод GetAvatar(), который вернет изображение клиенту:

```
namespace WebLabsV06.Controllers
{
    public class ImageController : Controller
    {
        UserManager<ApplicationUser> _userManager;
        IWebHostEnvironment _env;

        public ImageController(UserManager<ApplicationUser>
userManager, IWebHostEnvironment env)
        {
            _userManager = userManager;
            _env = env;
        }

        public async Task<FileResult> GetAvatar()
        {
            var user = await _userManager.GetUserAsync(User);
            if (user.AvatarImage != null)
                return File(user.AvatarImage, "image/...");
            else
            {
                var avatarPath = "/Images/anonymous.png";

                return File(_env.WebRootFileProvider
                    .GetFileInfo(avatarPath)
                    .CreateReadStream(), "image/...");
            }
        }
    }
}
```

5.9.2. Изменение представления _UserPartial

```
<div class="dropdown ml-4 nav-color">
    <div class="dropdown-toggle"
        id="dropdownMenuButton"
        data-toggle="dropdown" aria-haspopup="true"
        aria-expanded="false">
        
    </div>
```

```
<div class="dropdown-menu"
    aria-labelledby="dropdownMenuButton">
  <div class="dropdown-item-text">
    
    @User.Identity.Name
  </div>
```

Запустите проект. Войдите в систему под разными учетными данными.
Убедитесь, что аватар отображается правильно.