

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.

Время выполнения работы: 4 часа

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 обортывает 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 файлах проекта замените IdentityUser на ApplicationUser.

Измените настройку Identity (класс Startup, или для версии от 6 – класс Program) 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();
```

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

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

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

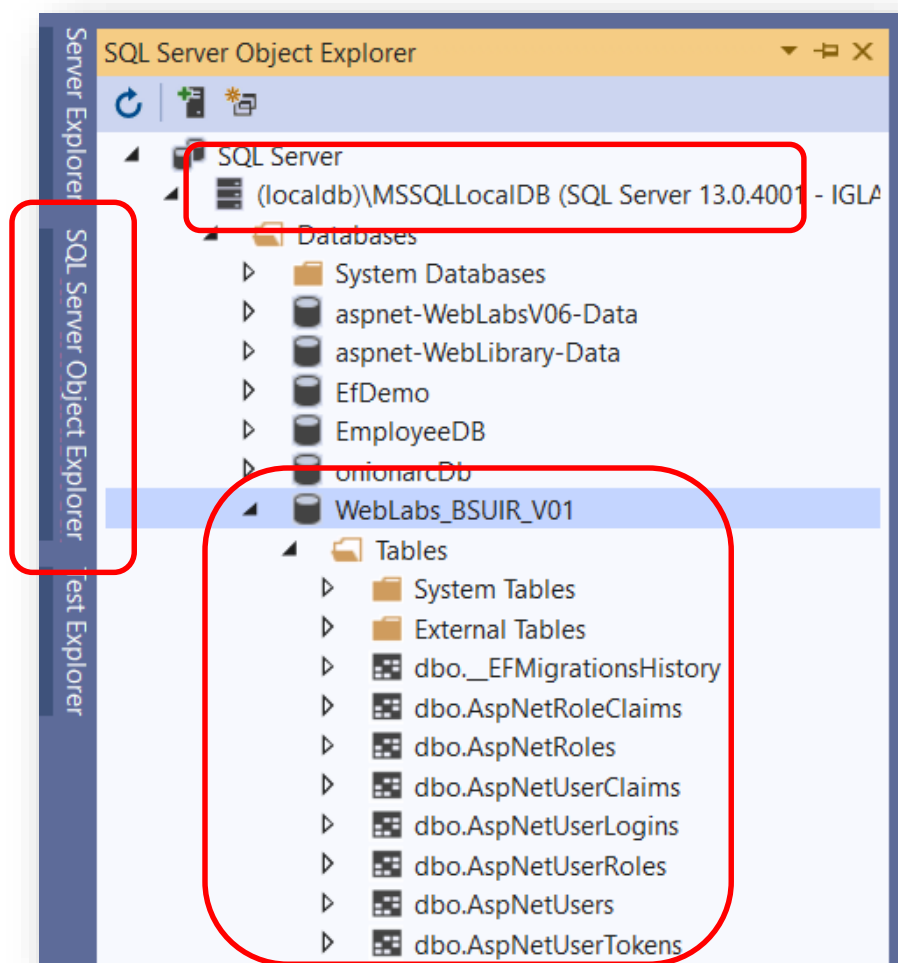


Рисунок. Создание класса инициализации базы данных

Примечание: при использовании базы данных SQLite воспользуйтесь сторонними программами, например, SQLite Studio.

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` (для версии 6 и выше – в классе `Program`, перед вызовом `app.Run()`);)

В классе `DbInitializer` вам понадобятся объекты `ApplicationDbContext`, `UserManager` и `RoleManager`. Воспользуйтесь механизмом `dependency injection` для получения этих объектов. Для этого в метод инициализации БД передать

объект класса `IApplicationBuilder` (для версии 6 и выше - объект `WebApplication - app`). С его помощью можно получить нужные сервисы.

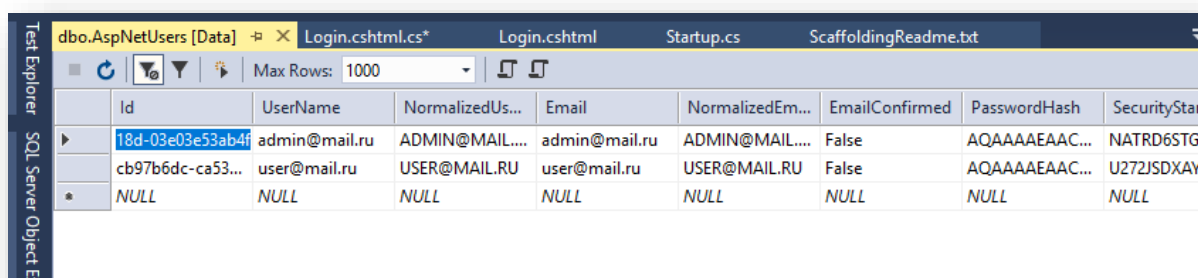
Нужные сервисы – это «scoped» сервисы, т.е. они могут быть извлечены при обработке http-запроса. Поэтому сначала нужно создать «scope» и получить объект `ServiceProvider` следующим образом:

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

а уже из него получить нужные сервисы:

`serviceProvider.GetService` или `serviceProvider.GetRequiredService`.

Проверьте, что база заполняется данными:



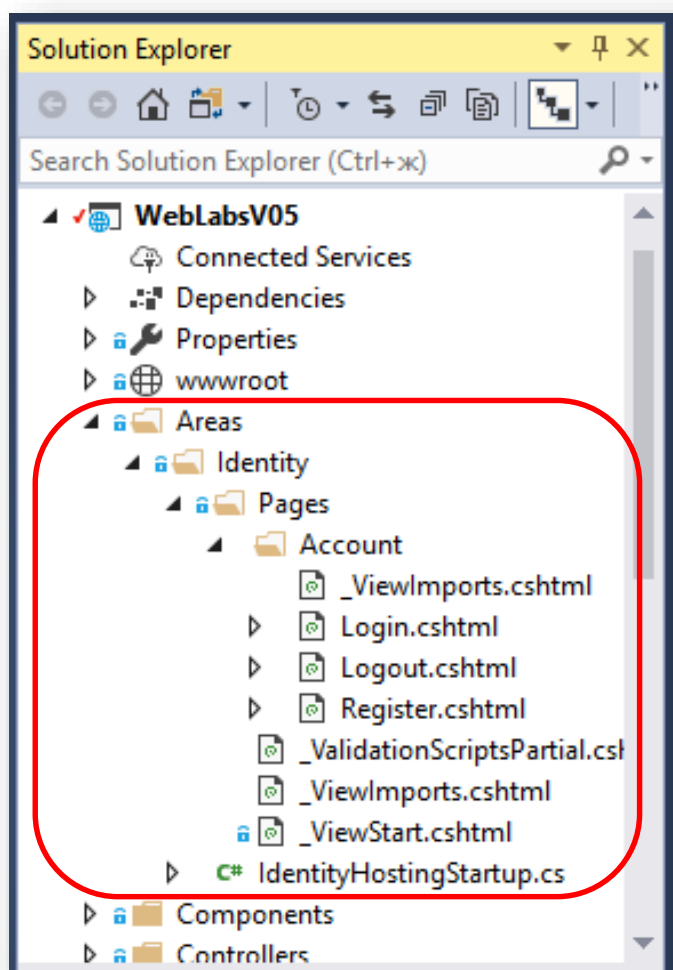
The screenshot shows the SQL Server Enterprise Manager interface. The 'Test Explorer' pane on the left shows 'SQL Server Object Explorer'. The main pane displays the 'dbo.AspNetUsers [Data]' table. The table has columns: Id, UserName, NormalizedUs..., Email, NormalizedEm..., EmailConfirmed, PasswordHash, and SecurityStar. The first two rows contain data for 'admin@mail.ru' and 'user@mail.ru', while the third row is NULL.

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

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

Scaffold для VisualStudio Code – см. [ссылку](https://docs.microsoft.com/en-us/aspnet/core/security/authentication/scaffold-identity?view=aspnetcore-6.0&tabs=netcore-cli#scaffold-identity-into-a-razor-project-with-authorization)
<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/scaffold-identity?view=aspnetcore-6.0&tabs=netcore-cli#scaffold-identity-into-a-razor-project-with-authorization>

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";
});
```

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.

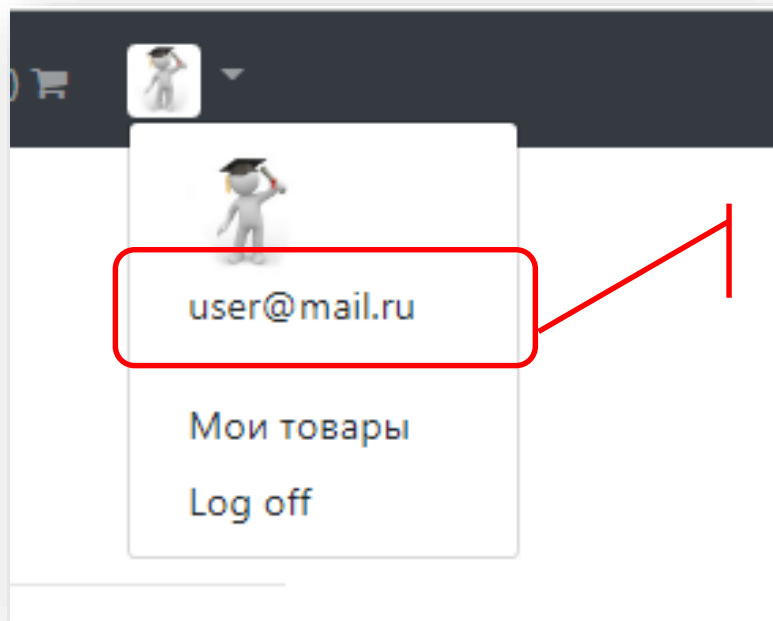
There are no external authentication services configured. See [this article](#) for details on setting up this ASP.NET application to support logging in via external services.

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 (на страницу Register) возможность загрузки аватара пользователя.

Изображение аватара должно храниться в базе данных. Для этого добавьте соответствующее свойство в класс ApplicationUser.

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` (`IWebHostEnvironment` для версии 5 и выше).

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