Администрирование информационных систем

02.03.03 -Математическое обеспечение и администрирование информационных систем, направленность (профиль) -разработка и администрирование информационных систем

https//vikchas.ru

Лабораторная работа «Управление пользователями информационной системы и их аутентификацией»

Часовских Виктор Петрович
доктор технических наук, профессор кафедры ШИиКМ, ФГБОУ ВО «Уральский государственный экономический университет

Екатеринбург 2026

Введение в ASP.NET Core Identity

ASP.NET Core Identity – это современная система управления пользователями и их аутентификацией.

ASP.NET Core Identity обеспечивает:

- Управление пользователями и их профилями
- Хранение учетных данных пользователей с безопасным хешированием паролей
- Реализацию входа/выхода пользователей
- Поддержку внешних провайдеров аутентификации (Google, Facebook, Twitter, Microsoft и др.)
- Многофакторную аутентификацию
- Управление ролями и политиками доступа
- Систему управления претензиями (claims)

Основные компоненты

IdentityUser

IdentityUser– это базовый класс, представляющий пользователя в системе. Он включает такие свойства как:

- UserName
- Email
- PasswordHash
- PhoneNumber
- TwoFactorEnabled
- LockoutEnabled

Вы можете расширить этот класс, добавив в него собственные свойства.

IdentityRole

IdentityRolepпредставляет роль в системе, что позволяет группировать пользователей по уровню доступа или функциональным обязанностям.

UserManager и RoleManager

UserManagerиRoleManager– это классы, предоставляющие API для работы с пользователями и ролями соответственно. Они инкапсулируют логику для:

- Создания, обновления и удаления пользователей/ролей
- Поиска пользователей
- Проверки паролей
- Назначения ролей пользователям
- Работы с претензиями и токенами

Подключение Identity к проекту

Шаг 1: Установка пакетов

C#

```
dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore

dotnet add package Microsoft.EntityFrameworkCore.SqlServer
```

Шаг 2: Создание контекста данных

C#

```
public class ApplicationDbContext : IdentityDbContext<IdentityUser>

{

   public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

       : base(options)

   {

   }

}
```

Шаг 3: Настройка в Startup.cs или Program.cs

C#

```
// Для .NET 6+

builder.Services.AddDbContext<ApplicationDbContext>(options =>


options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));
```

```
builder.Services.AddIdentity<IdentityUser, IdentityRole>(options => {
    // Настройки Identity
    options.Password.RequireDigit = true;
    options.Password.RequiredLength = 8;
    // Другие настройки...
})
.AddEntityFrameworkStores<ApplicationDbContext>()
.AddDefaultTokenProviders();
```

Шаг 4: Добавление middleware

C#

```
// Для .NET 6+
app.UseAuthentication();
app.UseAuthorization();
```

Базовые операции

Регистрация пользователя

C#

```
public async Task<IActionResult> Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = new IdentityUser { UserName = model.Email, Email = model.Email
};
        var result = await _userManager.CreateAsync(user, model.Password);

        if (result.Succeeded)
        {
            await _signInManager.SignInAsync(user, isPersistent: false);
            return RedirectToAction("Index", "Home");
        }
```

```
        foreach (var error in result.Errors)
        {
            ModelState.AddModelError(string.Empty, error.Description);
        }
    }

    return View(model);
}
```

Вход пользователя

C#

```
public async Task<IActionResult> Login(LoginViewModel model)
{
    if (ModelState.IsValid)
    {
        var result = await _signInManager.PasswordSignInAsync(model.Email,
            model.Password, model.RememberMe, lockoutOnFailure: false);

        if (result.Succeeded)
        {
            return RedirectToAction("Index", "Home");
        }

        ModelState.AddModelError(string.Empty, "Неверный логин или пароль.");
    }

    return View(model);
}
```

Выход из системы

C#

```
public async Task<IActionResult> Logout()
{
    await _signInManager.SignOutAsync();
    return RedirectToAction("Index", "Home");
}
```

Управление ролями

C#

```
// Создание роли
await _roleManager.CreateAsync(new IdentityRole("Admin"));

// Назначение роли пользователю
var user = await _userManager.FindByEmailAsync("user@example.com");
await _userManager.AddToRoleAsync(user, "Admin");

// Проверка принадлежности пользователя к роли
bool isInRole = await _userManager.IsInRoleAsync(user, "Admin");
```

Защита контроллеров и действий

C#

```
// Требует аутентификацию
[Authorize]
public IActionResult SecurePage()
{
    return View();
}

// Требует определенную роль
[Authorize(Roles = "Admin")]
public IActionResult AdminPanel()
```

```
{
    return View();
}

// Требует определенную политику
[Authorize(Policy = "RequireAdminRole")]
public IActionResult ManageUsers()
{
    return View();
}
```

Настройка политик авторизации

C#

```
builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("RequireAdminRole", policy => policy.RequireRole("Admin"));

    options.AddPolicy("CanManageUsers", policy =>
        policy.RequireClaim("Permission", "ManageUsers"));

    options.AddPolicy("AtLeast21", policy =>
        policy.RequireAssertion(context =>
            context.User.HasClaim(c => c.Type == "Age") &&
            int.Parse(context.User.FindFirst(c => c.Type == "Age").Value) >= 21));
});
```

ASP.NET Core Identity представляет собой мощное и гибкое решение для управления пользователями в веб-приложениях. Оно обеспечивает все необходимые функции для создания безопасной системы аутентификации и авторизации, при этом оставаясь достаточно гибким для адаптации к конкретным потребностям вашего проекта.

Использование ASP.NET Core Identity значительно упрощает разработку, так как вам не нужно создавать собственную систему аутентификации с нуля, а достаточно настроить существующее решение под свои требования.

При разработке серьезных приложений рекомендуется расширить базовый функционал Identity, добавив такие возможности как многофакторная аутентификация, подтверждение электронной почты, подробные журналы входов и блокировка аккаунтов при подозрительной активности.

Цель лабораторной работа “Управление пользователями информационной системы и их аутентификацией”

Изучить основные принципы работы ASP.NET Core Identity и реализовать систему аутентификации и авторизации в веб-приложении ASP.NET Core.

Задачи

1. Настроить ASP.NET Core Identity в проекте
2. Реализовать регистрацию и аутентификацию пользователей
3. Настроить управление ролями и политиками
4. Защитить разделы приложения с помощью авторизации

Теоретическая часть

ASP.NET Core Identity - это система членства, предоставляющая API для управления пользователями, паролями, ролями, токенами, подтверждением email и многим другим. Библиотека обеспечивает безопасное хранение учетных данных и интегрируется с Entity Framework Core для работы с базой данных.

Основные компоненты Identity:

1. IdentityUser- представляет пользователя
2. IdentityRole- представляет роль в системе
3. UserManager- управляет пользователями
4. SignInManager- обрабатывает вход/выход пользователей
5. RoleManager- управляет ролями

Практическая часть

Часть 1: Настройка проекта

1. Создайте новый проект ASP.NET Core MVC.
2. Добавьте необходимые пакеты NuGet:

3. Microsoft.AspNetCore.Identity.EntityFrameworkCore
4. Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

5. Создайте классApplicationUser, наследующийся отIdentityUser:

C#

```
public class ApplicationUser : IdentityUser
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime RegistrationDate { get; set; }
}
```

6. Создайте класс контекста данных ApplicationDbContext:

C#

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }

    // Можно добавить дополнительные DbSet для других сущностей
}
```

7. Настройте подключение к базе данных вappsettings.json:

JSON

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=IdentityLab;Trusted_Connection=True;MultipleActiveResultSets=true"
```

```
  }
}
```

8. Зарегистрируйте сервисы в методе Program.cs:

C#

```
var builder = WebApplication.CreateBuilder(args);

// Добавление контекста данных
builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

// Настройка Identity
builder.Services.AddIdentity<ApplicationUser, IdentityRole>(options => {
    options.Password.RequireDigit = true;
    options.Password.RequiredLength = 8;
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequireUppercase = true;
    options.Password.RequireLowercase = true;

    options.User.RequireUniqueEmail = true;

    options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(15);
    options.Lockout.MaxFailedAccessAttempts = 5;
})
.AddEntityFrameworkStores<ApplicationDbContext>()
.AddDefaultTokenProviders();

// Настройка cookies
```

```
builder.Services.ConfigureApplicationCookie(options => {
    options.LoginPath = "/Account/Login";
    options.AccessDeniedPath = "/Account/AccessDenied";
    options.SlidingExpiration = true;
    options.ExpireTimeSpan = TimeSpan.FromHours(1);
});

// Добавление MVC
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Middleware
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();

app.UseAuthentication();
app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

9. Создайте и примените миграцию:
10. dotnet ef migrations add InitialCreate

```
dotnet ef database update
```

Часть 2: Реализация регистрации и входа в систему

1. Создайте модели представления для входа и регистрации:

C#

```
// Models/AccountViewModels/RegisterViewModel.cs
public class RegisterViewModel
{
    [Required]
    [Display(Name = "Имя")]
    public string FirstName { get; set; }

    [Required]
    [Display(Name = "Фамилия")]
    public string LastName { get; set; }

    [Required]
    [EmailAddress]
    [Display(Name = "Email")]
    public string Email { get; set; }

    [Required]
    [StringLength(100, MinimumLength = 8)]
    [DataType(DataType.Password)]
    [Display(Name = "Пароль")]
    public string Password { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Подтверждение пароля")]
    [Compare("Password", ErrorMessage = "Пароль и его подтверждение не совпадают.")]
    public string ConfirmPassword { get; set; }
```

```
}

// Models/AccountViewModels/LoginViewModel.cs
public class LoginViewModel
{
    [Required]
    [EmailAddress]
    public string Email { get; set; }

    [Required]
    [DataType(DataType.Password)]
    public string Password { get; set; }

    [Display(Name = "Запомнить меня?")]
    public bool RememberMe { get; set; }
}
```

2. Создайте контроллерAccountController.cs:

C#

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using YourNamespace.Models.AccountViewModels;
using YourNamespace.Models;
using System.Threading.Tasks;

public class AccountController : Controller
{
    private readonly UserManager<ApplicationUser> _userManager;
    private readonly SignInManager<ApplicationUser> _signInManager;
```

```
public AccountController(
    UserManager<ApplicationUser> userManager,
    SignInManager<ApplicationUser> signInManager)
{
    _userManager = userManager;
    _signInManager = signInManager;
}

[HttpGet]
public IActionResult Register()
{
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser
        {
            UserName = model.Email,
            Email = model.Email,
            FirstName = model.FirstName,
            LastName = model.LastName,
            RegistrationDate = DateTime.Now
        };
```

```
            var result = await _userManager.CreateAsync(user, model.Password);

            if (result.Succeeded)
            {
                await _signInManager.SignInAsync(user, isPersistent: false);
                return RedirectToAction("Index", "Home");
            }

            foreach (var error in result.Errors)
            {
                ModelState.AddModelError(string.Empty, error.Description);
            }
        }
        return View(model);
    }

    [HttpGet]
    public IActionResult Login()
    {
        return View();
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null)
    {
        ViewData["ReturnUrl"] = returnUrl;
```

```
        if (ModelState.IsValid)
        {
            var result = await _signInManager.PasswordSignInAsync(
                model.Email, model.Password, model.RememberMe, lockoutOnFailure:
true);

            if (result.Succeeded)
            {
                if (!string.IsNullOrEmpty(returnUrl) && Url.IsLocalUrl(returnUrl))
                    return Redirect(returnUrl);
                else
                    return RedirectToAction("Index", "Home");
            }

            if (result.IsLockedOut)
            {
                return RedirectToAction("Lockout");
            }
            else
            {
                ModelState.AddModelError(string.Empty, "Неверная попытка входа.");
                return View(model);
            }
        }

        return View(model);
    }

    [HttpPost]
```

```
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Logout()
    {
        await _signInManager.SignOutAsync();
        return RedirectToAction("Index", "Home");
    }

    public IActionResult AccessDenied()
    {
        return View();
    }
}
```

3. Создайте представления для входа и регистрации:

Views/Account/Register.cshtml

HTML

```
@model RegisterViewModel

<h2>Регистрация нового пользователя</h2>

<div class="row">
    <div class="col-md-6">
        <form asp-controller="Account" asp-action="Register" method="post">
            <h4>Создайте новую учетную запись.</h4>
            <hr />
            <div asp-validation-summary="All" class="text-danger"></div>

            <div class="form-group">
                <label asp-for="FirstName"></label>
                <input asp-for="FirstName" class="form-control" />
```

```
            <span asp-validation-for="FirstName" class="text-danger"></span>
        </div>

        <div class="form-group">
            <label asp-for="LastName"></label>
            <input asp-for="LastName" class="form-control" />
            <span asp-validation-for="LastName" class="text-danger"></span>
        </div>

        <div class="form-group">
            <label asp-for="Email"></label>
            <input asp-for="Email" class="form-control" />
            <span asp-validation-for="Email" class="text-danger"></span>
        </div>

        <div class="form-group">
            <label asp-for="Password"></label>
            <input asp-for="Password" class="form-control" />
            <span asp-validation-for="Password" class="text-danger"></span>
        </div>

        <div class="form-group">
            <label asp-for="ConfirmPassword"></label>
            <input asp-for="ConfirmPassword" class="form-control" />
            <span asp-validation-for="ConfirmPassword" class="text-
danger"></span>
        </div>

        <button type="submit" class="btn btn-primary">Регистрация</button>
```

```
        </form>
    </div>
</div>
```

Views/Account/Login.cshtml

HTML

```
@model LoginViewModel

<h2>Вход в систему</h2>

<div class="row">
    <div class="col-md-6">
        <form asp-controller="Account" asp-action="Login"
              asp-route-returnUrl="@ViewData["ReturnUrl"]" method="post">
            <h4>Используйте свою учетную запись для входа.</h4>
            <hr />
            <div asp-validation-summary="All" class="text-danger"></div>

            <div class="form-group">
                <label asp-for="Email"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>

            <div class="form-group">
                <label asp-for="Password"></label>
                <input asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
```

```
        <div class="form-group">
            <div class="checkbox">
                <label>
                    <input asp-for="RememberMe" />
                    @Html.DisplayNameFor(m => m.RememberMe)
                </label>
            </div>
        </div>

        <button type="submit" class="btn btn-primary">Войти</button>
        <p>
            <a asp-action="Register">Зарегистрироваться как новый пользователь</a>
        </p>
    </form>
</div>
</div>
```

Часть 3: Настройка ролей и политик

1. Создайте класс инициализатора ролей:

C#

```
public static class RolesInitializer
{
    public static async Task InitializeAsync(
        IServiceProvider serviceProvider,
        IConfiguration configuration)
    {
        var roleManager = serviceProvider.GetRequiredService<RoleManager<IdentityRole>>();
        var userManager = serviceProvider.GetRequiredService<UserManager<ApplicationUser>>();
```

```
// Роли
string[] roles = { "Admin", "Manager", "User" };

foreach (var role in roles)
{
    if (!await roleManager.RoleExistsAsync(role))
    {
        await roleManager.CreateAsync(new IdentityRole(role));
    }
}

// Администратор
string adminEmail = configuration["AdminSettings:Email"];
string adminPassword = configuration["AdminSettings:Password"];

if (adminEmail == null || adminPassword == null)
{
    throw new Exception("Не указаны учетные данные администратора в конфигурации");
}

if (await userManager.FindByEmailAsync(adminEmail) == null)
{
    ApplicationUser admin = new ApplicationUser
    {
        Email = adminEmail,
        UserName = adminEmail,
        FirstName = "Admin",
```

```
            LastName = "Admin",
            RegistrationDate = DateTime.Now
        };

        var result = await userManager.CreateAsync(admin, adminPassword);

        if (result.Succeeded)
        {
            await userManager.AddToRoleAsync(admin, "Admin");
        }
    }
  }
}
```

2. Добавьте учетные данные администратора вappsettings.json:

JSON

```
"AdminSettings": {
  "Email": "admin@example.com",
  "Password": "Admin123!"
}
```

3. Вызовите инициализатор при запуске приложения вProgram.cs:

C#

```
// После app.Build()
using (var scope = app.Services.CreateScope())
{
   var services = scope.ServiceProvider;
   try
   {
      await RolesInitializer.InitializeAsync(services, builder.Configuration);
   }
```

```
    catch (Exception ex)
    {
        var logger = services.GetRequiredService<ILogger<Program>>();
        logger.LogError(ex, "Произошла ошибка при инициализации ролей.");
    }
}
```

4. Создайте контроллер для управления пользователями:

C#

```
[Authorize(Roles = "Admin")]
public class UserManagementController : Controller
{
    private readonly UserManager<ApplicationUser> _userManager;
    private readonly RoleManager<IdentityRole> _roleManager;

    public UserManagementController(
        UserManager<ApplicationUser> userManager,
        RoleManager<IdentityRole> roleManager)
    {
        _userManager = userManager;
        _roleManager = roleManager;
    }

    public async Task<IActionResult> Index()
    {
        var users = _userManager.Users.ToList();
        var userViewModels = new List<UserViewModel>();

        foreach (var user in users)
        {
```

```
            var roles = await _userManager.GetRolesAsync(user);

            userViewModels.Add(new UserViewModel
            {
                Id = user.Id,
                Email = user.Email,
                FirstName = user.FirstName,
                LastName = user.LastName,
                RegistrationDate = user.RegistrationDate,
                Roles = roles.ToList()
            });
        }

        return View(userViewModels);
    }

    [HttpGet]
    public async Task<IActionResult> Edit(string id)
    {
        var user = await _userManager.FindByIdAsync(id);
        if (user == null)
        {
            return NotFound();
        }

        var userRoles = await _userManager.GetRolesAsync(user);
        var allRoles = _roleManager.Roles.Select(r => r.Name).ToList();

        var model = new EditUserViewModel
```

```
        {
            Id = user.Id,
            Email = user.Email,
            FirstName = user.FirstName,
            LastName = user.LastName,
            UserRoles = userRoles.ToList(),
            AllRoles = allRoles
        };

        return View(model);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(EditUserViewModel model, string[] selectedRoles)
    {
        if (!ModelState.IsValid)
        {
            return View(model);
        }

        var user = await _userManager.FindByIdAsync(model.Id);
        if (user == null)
        {
            return NotFound();
        }

        user.Email = model.Email;
```

```
        user.UserName = model.Email;
        user.FirstName = model.FirstName;
        user.LastName = model.LastName;

        var result = await _userManager.UpdateAsync(user);
        if (!result.Succeeded)
        {
            foreach (var error in result.Errors)
            {
                ModelState.AddModelError("", error.Description);
            }
            return View(model);
        }

        // Обновление ролей
        var userRoles = await _userManager.GetRolesAsync(user);
        selectedRoles = selectedRoles ?? new string[] { };

        // Удаление ролей
        result = await _userManager.RemoveFromRolesAsync(user,
userRoles.Except(selectedRoles));
        if (!result.Succeeded)
        {
            ModelState.AddModelError("", "Не удалось удалить текущие роли");
            return View(model);
        }

        // Добавление новых ролей
```

```
        result = await _userManager.AddToRolesAsync(user, selectedRoles.Except(userRoles));
        if (!result.Succeeded)
        {
            ModelState.AddModelError("", "Не удалось добавить выбранные роли");
            return View(model);
        }

        return RedirectToAction("Index");
    }
}
```

5. Создайте модели представления:

C#

```
public class UserViewModel
{
    public string Id { get; set; }
    public string Email { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime RegistrationDate { get; set; }
    public List<string> Roles { get; set; }
}

public class EditUserViewModel
{
    public string Id { get; set; }

    [Required]
    [EmailAddress]
```

```
    public string Email { get; set; }

    [Required]
    public string FirstName { get; set; }

    [Required]
    public string LastName { get; set; }

    public List<string> UserRoles { get; set; }
    public List<string> AllRoles { get; set; }
}
```

Часть 4: Защита разделов приложения

1. Создайте защищенный контроллер:

C#

```
public class SecureController : Controller
{
    [Authorize]
    public IActionResult UserArea()
    {
        return View();
    }

    [Authorize(Roles = "Manager,Admin")]
    public IActionResult ManagerArea()
    {
        return View();
    }

    [Authorize(Roles = "Admin")]
```

```
    public IActionResult AdminArea()
    {
        return View();
    }
}
```

2. Создайте простые представления для этих действий.
3. Обновите _Layout.cshtml, чтобы показывать разные меню в зависимости от роли пользователя:

HTML

```
<nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-dark bg-dark border-bottom box-shadow mb-3">
    <div class="container">
        <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">Identity Lab</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
            <ul class="navbar-nav flex-grow-1">
                <li class="nav-item">
                    <a class="nav-link text-light" asp-controller="Home" asp-action="Index">Главная</a>
                </li>

                @if (User.Identity.IsAuthenticated)
                {
                    <li class="nav-item">
                        <a class="nav-link text-light" asp-controller="Secure" asp-action="UserArea">Личный кабинет</a>
```

```
            </li>

            @if (User.IsInRole("Manager") || User.IsInRole("Admin"))
            {
                <li class="nav-item">
                    <a class="nav-link text-light" asp-controller="Secure" asp-action="ManagerArea">Управление</a>
                </li>
            }

            @if (User.IsInRole("Admin"))
            {
                <li class="nav-item">
                    <a class="nav-link text-light" asp-controller="Secure" asp-action="AdminArea">Администрирование</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link text-light" asp-controller="UserManagement" asp-action="Index">Пользователи</a>
                </li>
            }
        }
    </ul>

    <partial name="_LoginPartial" />
  </div>
 </div>
</nav>
```

4. Создайте частичное представление _LoginPartial.cshtml:

HTML

```
@if (User.Identity.IsAuthenticated)
{
  <form method="post" asp-controller="Account" asp-action="Logout" id="logoutForm" class="navbar-right">
    <ul class="nav navbar-nav navbar-right">
      <li class="nav-item">
        <span class="nav-link text-light">Привет, @User.Identity.Name!</span>
      </li>
      <li class="nav-item">
        <button type="submit" class="btn btn-link navbar-btn nav-link text-light">Выйти</button>
      </li>
    </ul>
  </form>
}
else
{
  <ul class="nav navbar-nav navbar-right">
    <li class="nav-item">
      <a class="nav-link text-light" asp-controller="Account" asp-action="Register">Регистрация</a>
    </li>
    <li class="nav-item">
      <a class="nav-link text-light" asp-controller="Account" asp-action="Login">Вход</a>
    </li>
  </ul>
}
```

Задания для самостоятельной работы

1. Реализуйте функцию подтверждения адреса электронной почты при регистрации.
2. Добавьте возможность сброса пароля.
3. Реализуйте интеграцию с внешним провайдером аутентификации (Google, Facebook).
4. Разработайте систему управления пользовательскими претензиями (claims) для более гибкой авторизации.
5. Добавьте политику, требующую сложный пароль и двухфакторную аутентификацию для администраторов.

Реферат ответов на контрольные вопросы

1. Что такое ASP.NET Core Identity и какие проблемы она решает?
2. Объясните различие между аутентификацией и авторизацией.
3. Какие компоненты входят в состав ASP.NET Core Identity?
4. Как защитить определенный участок приложения (контроллер или его метод)?
5. В чем разница между Claims и Roles в контексте авторизации?
6. Как работает хранение паролей в ASP.NET Core Identity?
7. Что такое политики (Policies) и как их использовать для авторизации?
8. Как добавить пользовательские данные в стандартную модель IdentityUser?