Министерство цифрового развития, связи и
массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Отчет
по курсовой работе
по дисциплине «**Web-разработка**»

Выполнил:
студент гр. ИС-142
«__» декабря 2024 г. _____ /Григорьев Ю.В./

Проверил:
преподаватель
«__» декабря 2024 г. _____ /Курзин А.С./

Оценка « _____ »

Новосибирск 2024

ЗАДАНИЕ

Целью работы было разработать веб-приложение блог на платформе ASP.NET Core с использованием технологии Razor Pages. Приложение должно обеспечивать базовую функциональность для управления постами, включая их создание, редактирование и удаление. Все посты должны отображаться на главной странице, а доступ к функциям редактирования и удаления осуществляется через отдельные страницы. В качестве хранилища данных используется реляционная база данных, управляемая системой управления базами данных (СУБД) SQLite. Реализация включает использование Entity Framework Core для взаимодействия с базой данных, а также создание структуры и логики приложения с применением стандартных паттернов разработки.

ВЫПОЛНЕНИЕ РАБОТЫ

1. Модели

В рамках выполнения курсового проекта была создана модель Post, представляющая структуру данных для публикаций блога. Она включает свойства для идентификатора, заголовка, содержимого и даты создания. Данная модель соответствует таблице в базе данных, обеспечивая хранение информации о постах.

Для аутентификации и управления пользователями использовалась встроенная функциональность Identity в ASP.NET Core, которая включает модели для управления пользователями, ролями, паролями и другими аспектами аутентификации. Настройка моделей была выполнена для интеграции с SQLite через Entity Framework Core, включая конфигурацию базы данных в классе ApplicationDbContext.

2. Контроллеры и обработчики страниц

Проект построен на основе Razor Pages, которые предоставляют обработчики страниц для выполнения действий, таких как создание, редактирование, удаление и просмотр постов. Для каждой операции были созданы отдельные классы в папке Pages/Posts:

1. Create: Обрабатывает создание нового поста. Проверяется корректность введенных данных и осуществляется их сохранение в базе данных.
2. Edit: Позволяет редактировать существующие посты. Загружается информация поста для редактирования, а затем изменения сохраняются в базе.
3. Delete: Обеспечивает удаление постов. Реализован метод подтверждения перед удалением.
4. Index: Отображает список всех постов на главной странице. Учет прав доступа реализован через Razor Pages, чтобы элементы редактирования и удаления отображались только для администратора.

Дополнительно была добавлена логика проверки прав администратора через сравнение email текущего пользователя с настроенным email администратора. Это позволяет ограничить доступ к действиям редактирования и удаления только администратору.

3. Представления

Для проекта были созданы Razor-представления, соответствующие каждой странице:

1. Index: Главная страница с отображением всех постов в формате списка. Посты представлены с заголовками, датой публикации и содержимым. Для авторизованных пользователей-администраторов предусмотрены кнопки для редактирования и удаления.
2. Create: Страница с формой для создания нового поста. Реализована проверка на заполнение обязательных полей.
3. Edit: Форма редактирования, которая предварительно заполняется данными поста. Позволяет администратору изменить заголовок и содержимое.
4. Delete: Страница подтверждения удаления поста с выводом его заголовка и содержания.

Для обеспечения аутентификации были созданы представления для страниц входа, регистрации и выхода из учетной записи. Они включают базовые формы для ввода email и пароля, а также ссылки на переключение между страницами входа и регистрации.

4. Аутентификация

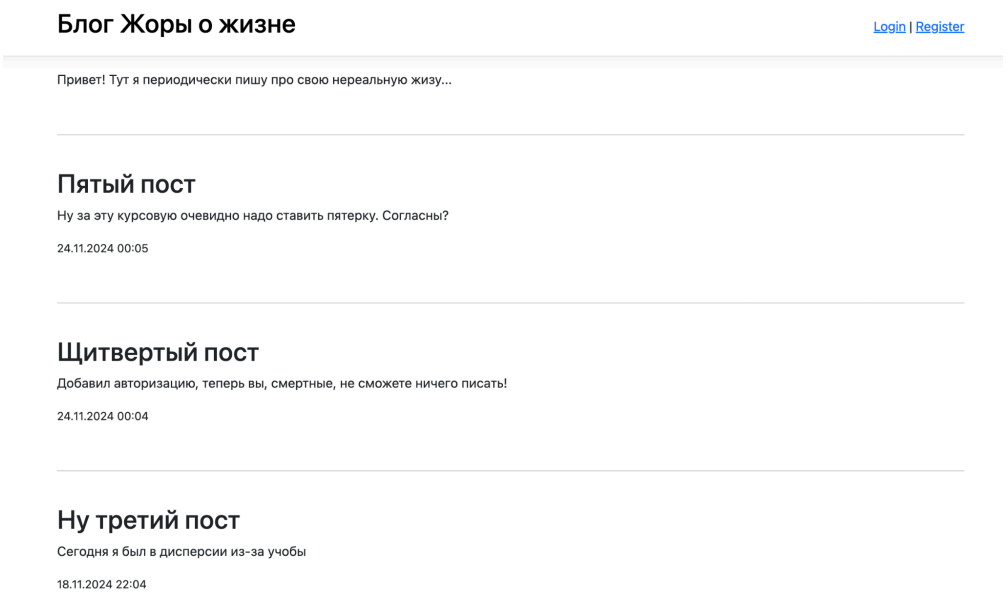
В проекте была интегрирована система аутентификации с использованием ASP.NET Core Identity. Это позволило:

1. Реализовать регистрацию и вход пользователей.
2. Проверять авторизацию для доступа к страницам создания, редактирования и удаления постов.
3. Добавить проверку на роль администратора, используя заданный в конфигурации email. Только администратору доступны кнопки для редактирования и удаления постов.

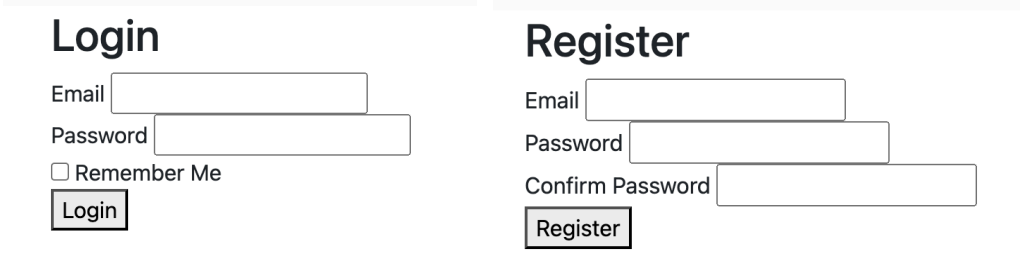
Для хранения пользователей Identity использует таблицы в базе данных, которые автоматически создаются с помощью миграций Entity Framework Core. Настройка выполнена так, чтобы взаимодействие с пользователями было полностью интегрировано с остальной частью проекта.

ДЕМОНСТРАЦИЯ РАБОТЫ

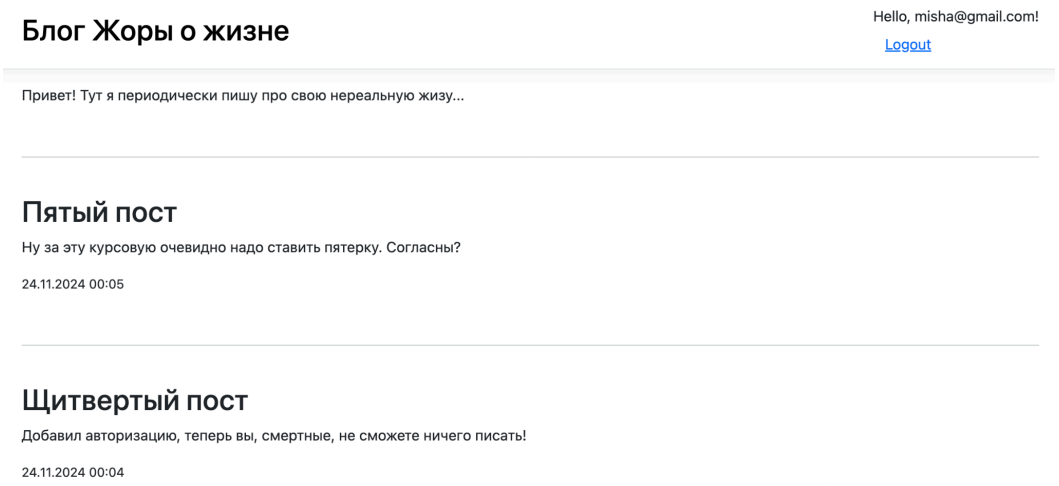
Простое отображение постов для неавторизованных пользователей



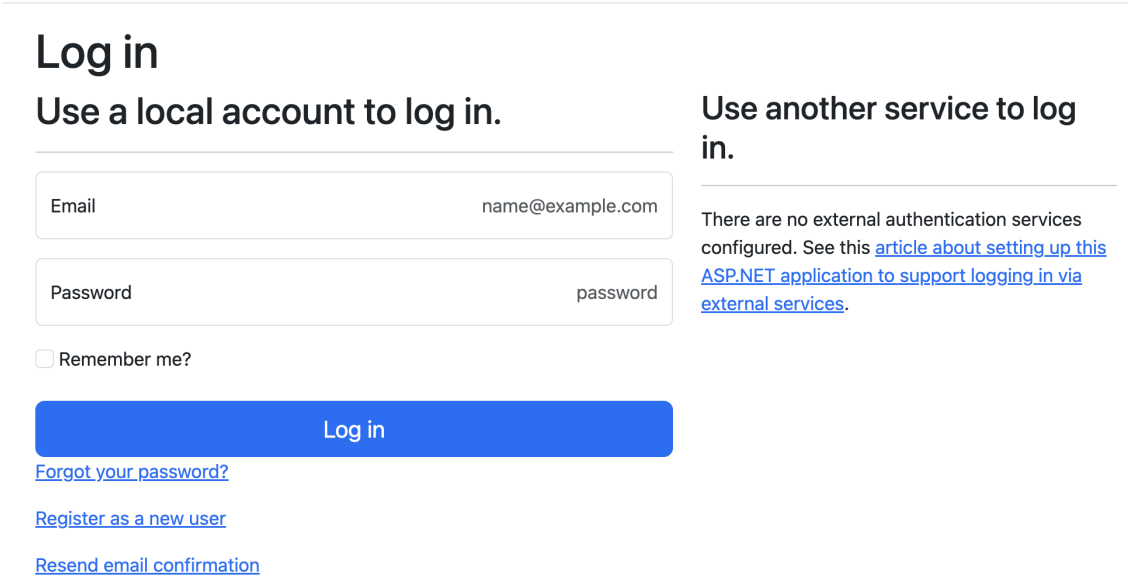
Страницы входа / регистрации



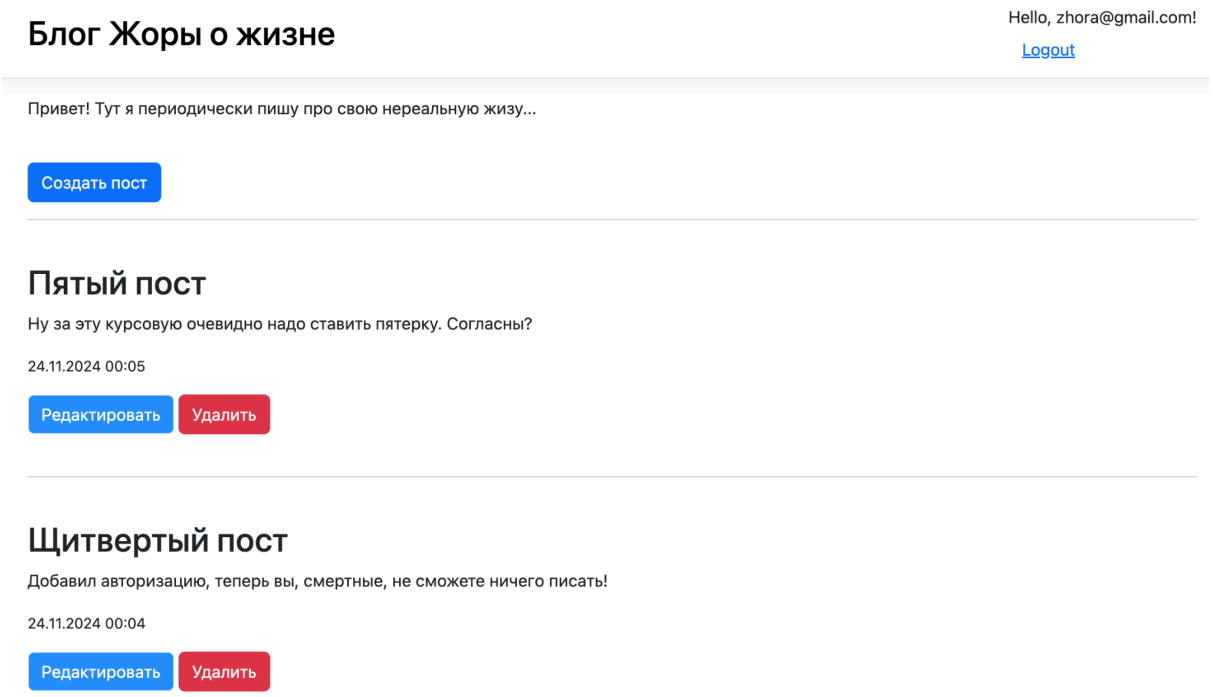
Главная страница после авторизации не от имени администратора



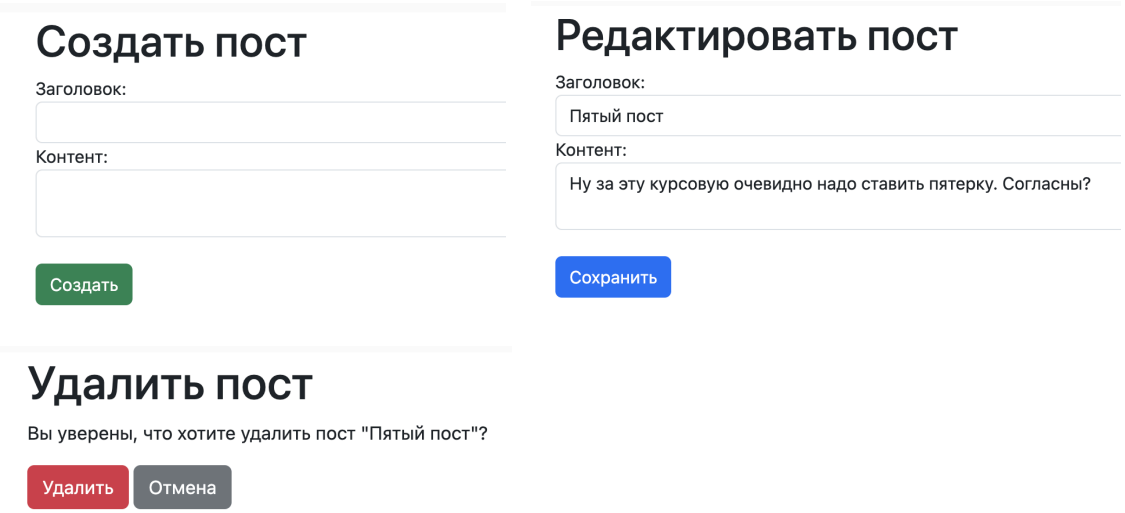
Страницы создания/редактирования/удаления постов для неавторизованного пользователя (просит авторизоваться)



Главная страница после авторизации от имени администратора



Страницы создания / редактирования / удаления постов от имени администратора



Страницы создания / редактирования / удаления постов не от имени администратора

ТЕСТИРОВАНИЕ

Для обеспечения корректной работы разработанного веб-приложения был реализован и проведен процесс модульного тестирования. Тестирование охватывало ключевые аспекты функциональности приложения, включая создание, редактирование и удаление записей блога, а также проверку авторизации пользователей. Все тесты были выполнены с использованием популярного фреймворка тестирования **xUnit**, который обеспечивает гибкость и надежность при написании модульных тестов.

1. Процесс тестирования

Для тестирования был создан отдельный проект, подключенный к основному проекту приложения. Были выполнены настройки тестовой среды, включая подключение к контексту базы данных в памяти (InMemory Database) и настройка мок-объектов для взаимодействия с сервисами, такими как UserManager, используемыми для проверки авторизации пользователей. Это позволило изолировать тесты и исключить влияние внешних факторов, таких как реальная база данных или служба управления пользователями.

2. Тесты для функциональности создания записей (Create)

Для страницы создания записей были написаны тесты, проверяющие два основных сценария:

1. **Создание записи администратором**: Тест проверяет, что при вводе корректных данных администратор может создать новую запись, которая затем сохраняется в базе данных.
2. **Доступ для неадминистратора**: Тест проверяет, что пользователи без прав администратора получают отказ в доступе (HTTP статус 403) при попытке открыть страницу создания записей или отправить данные на сервер.

Эти тесты подтвердили корректность проверки прав доступа и валидации данных при создании записей.

3. Тесты для функциональности редактирования записей (Edit)

Для страницы редактирования записей были реализованы следующие тесты:

1. **Редактирование записи администратором**: Проверяется успешное обновление записи в базе данных при корректных входных данных.
2. **Попытка редактирования несуществующей записи**: Тест проверяет, что при попытке отредактировать запись с несуществующим идентификатором возвращается статус "404 Not Found".
3. **Доступ для неадминистратора**: Проверяется, что неавторизованные пользователи получают отказ в доступе (HTTP статус 403) при попытке открыть или отправить данные на страницу редактирования.

Эти тесты обеспечили уверенность в правильной реализации проверки прав доступа, обработки ошибок и обновления данных.

4. Тесты для функциональности удаления записей (Delete)

Для страницы удаления записей были протестированы следующие сценарии:

1. **Удаление записи администратором**: Тест проверяет, что запись корректно удаляется из базы данных, если она существует.
2. **Попытка удаления несуществующей записи**: Проверяется, что при попытке удалить запись с несуществующим идентификатором возвращается статус "404 Not Found".
3. **Доступ для неадминистратора**: Проверяется, что пользователи без прав администратора получают отказ в доступе (HTTP статус 403) при попытке удалить запись.

Тесты подтвердили, что функциональность удаления записей работает корректно и безопасно, а также проверяется наличие необходимых прав доступа.

5. Результаты тестирования

В результате проведенного тестирования все разработанные тесты успешно прошли. Это подтверждает корректность реализации следующих аспектов:

- Проверки прав доступа для страниц создания, редактирования и удаления записей.

- Обработки ошибок при работе с несуществующими данными.
- Сохранения, обновления и удаления данных в базе.

Достижение полной успешности тестов позволяет уверенно утверждать, что функциональность приложения реализована в соответствии с заданными требованиями и корректно работает при различных сценариях использования.

```
[xUnit.net 00:00:00.15]   Starting:    SimpleBlog.Tests
[xUnit.net 00:00:01.20]   Finished:    SimpleBlog.Tests
  SimpleBlog.Tests test succeeded (2.2s)

Test summary: total: 17, failed: 0, succeeded: 17, skipped: 0, duration: 2.2s
```

Демонстрация успешного прохождения всех unit-тестов

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы был разработан веб-приложение **"Личный блог"**, представляющее собой упрощенный блог. Приложение реализовано на платформе **ASP.NET Core** с использованием технологии **Razor Pages** и базы данных **SQLite**, что позволило обеспечить удобство разработки, гибкость настройки и высокую производительность.

В процессе работы были выполнены следующие задачи:

- Спроектирована структура данных и создана модель **Post** для управления записями блога.
- Разработан набор страниц для создания, редактирования, удаления и просмотра постов с учетом прав доступа.
- Интегрирована система аутентификации на основе **ASP.NET Core Identity**, что позволило ограничить доступ к административным действиям только авторизованным пользователям.
- Реализован механизм проверки прав администратора на основе email, заданного в конфигурации приложения.
- Обеспечен пользовательский интерфейс с применением **Bootstrap**, включая адаптивную навигационную панель.

Приложение успешно реализует все поставленные задачи и соответствует требованиям курсовой работы. В результате выполнения проекта были закреплены навыки работы с ASP.NET Core, Razor Pages, базами данных и системой аутентификации. Разработанное приложение может быть легко расширено и дополнено новыми функциями, такими как категории постов, комментарии или система ролей для пользователей.

Проект демонстрирует возможности современной веб-разработки и может быть использован как основа для создания полнофункциональных блогов или других типов контент-ориентированных приложений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Andrew Troelsen. Pro C# 5.0 and the .NET 4.5 Framework. Sixth Edition. Apress – 2012, 1487 p.

2. Jeffrey Richter. CLR via CSharp. 3rd edition. Microsoft Press – 2010, 896 p.

3. М. МакДональд, А. Фримен, М. Шпушта. Microsoft ASP.NET 4 с примерами на C# 2010 для профессионалов/ Пер. с англ. – М.: ООО “И.Д. Вильямс”, 2011. – 1424 с.

4. Дж. Гленн, Н. Тонн. Разработка клиентских веб-приложений на основе платформы Microsoft .NET Framework. Учебный курс Microsoft/ Пер. с англ. – М.: “Русская редакция”, СПб.: Питер, 2007. – 768 с.

5. Дж. Шеперд. Microsoft ASP.NET: Step by Step/ Пер. с англ. – ЭКОМ Паблишерз, 2009. – 720 с.

ПРИЛОЖЕНИЕ

Исходный код проекта: примеры модели и контроллера, unit-тесты для него

Models/AccountViewModels.cs

```
namespace SimpleBlog.Models
{
    public class RegisterViewModel
    {
        public string Email { get; set; }
        public string Password { get; set; }
        public string ConfirmPassword { get; set; }
    }

    public class LoginViewModel
    {
        public string Email { get; set; }
        public string Password { get; set; }
        public bool RememberMe { get; set; }
    }
}
```

Models/Post.cs

```
using System.ComponentModel.DataAnnotations;

namespace SimpleBlog.Models
{
    public class Post
    {
        public int Id { get; set; }
```

```
        [Required]
        [StringLength(100)]
        public string Title { get; set; }

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

        public DateTime CreatedAt { get; set; } = DateTime.Now;
    }
}
```

Pages/Posts/Create.cshtml

```
@page
@model SimpleBlog.Pages.Posts.CreateModel

<h1>Создать пост</h1>

<form method="post">
    <label>Заголовок:</label>
    <input type="text" asp-for="Post.Title" class="form-control" />
    <span asp-validation-for="Post.Title" class="text-danger"></span>

    <label>Контент:</label>
    <textarea asp-for="Post.Content" class="form-control"></textarea>
    <span asp-validation-for="Post.Content" class="text-danger"></span>
    <br>
    <button type="submit" class="btn btn-success">Создать</button>
</form>
```

Pages/Posts/Create.cshtml.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Options;
using SimpleBlog.Data;
using SimpleBlog.Models;

namespace SimpleBlog.Pages.Posts
{
    [Authorize]
    public class CreateModel : PageModel
    {
        private readonly ApplicationDbContext _context;
        private readonly UserManager<IdentityUser> _userManager;
        private readonly AdminSettings _adminSettings;

        public CreateModel(
            ApplicationDbContext context,
            UserManager<IdentityUser> userManager,
            IOptions<AdminSettings> adminSettings)
        {
            _context = context;
            _userManager = userManager;
            _adminSettings = adminSettings.Value;
```

```
        }

        [BindProperty]
        public Post Post { get; set; }

        public IActionResult OnGet()
        {
            // Проверка: текущий пользователь — администратор?
            var currentUserEmail = _userManager.GetUserName(User);
            if (currentUserEmail != _adminSettings.AdminEmail)
            {
                return Forbid(); // Запретить доступ
            }

            return Page();
        }

        public IActionResult OnPost()
        {
            // Проверка: текущий пользователь — администратор?
            var currentUserEmail = _userManager.GetUserName(User);
            if (currentUserEmail != _adminSettings.AdminEmail)
            {
                return Forbid(); // Запретить доступ
            }

            if (!ModelState.IsValid)
            {
                return Page();
            }

            _context.Posts.Add(Post);
            _context.SaveChanges();
            return RedirectToPage("/Index");
        }
    }
}
```

CreateModelTests.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Options;
using Moq;
using SimpleBlog.Data;
using SimpleBlog.Models;
using SimpleBlog.Pages.Posts;
using System.Collections.Generic;
using System.Security.Claims;
using Xunit;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;

namespace SimpleBlog.Tests
{
    public class CreateModelTests
    {
```

```
        private ApplicationDbContext GetTestDbContext()
        {
            var options = new DbContextOptionsBuilder<ApplicationDbContext>()
                .UseInMemoryDatabase("TestDatabase")
                .Options;
            return new ApplicationDbContext(options);
        }

        private Mock<UserManager<IdentityUser>> GetMockUserManager()
        {
            var store = new Mock<IUserStore<IdentityUser>>();
            return new Mock<UserManager<IdentityUser>>(
                store.Object, null, null, null, null, null, null, null, null);
        }

        [Fact]
        public void OnGet_AdminUser_ReturnsPage()
        {
            // Arrange
            var dbContext = GetTestDbContext();
            var userManager = GetMockUserManager();
            var adminSettings = Options.Create(new AdminSettings { AdminEmail =
"admin@example.com" });

            userManager.Setup(um =>
um.GetUserName(It.IsAny<ClaimsPrincipal>())).Returns("admin@example.com");

            var createModel = new CreateModel(dbContext, userManager.Object,
adminSettings);

            // Act
            var result = createModel.OnGet();

            // Assert
            Assert.IsType<PageResult>(result);
        }

        [Fact]
        public void OnGet_NonAdminUser_ReturnsForbid()
        {
            // Arrange
            var dbContext = GetTestDbContext();
            var userManager = GetMockUserManager();
            var adminSettings = Options.Create(new AdminSettings { AdminEmail =
"admin@example.com" });

            userManager.Setup(um =>
um.GetUserName(It.IsAny<ClaimsPrincipal>())).Returns("user@example.com");

            var createModel = new CreateModel(dbContext, userManager.Object,
adminSettings);

            // Act
            var result = createModel.OnGet();

            // Assert
            Assert.IsType<ForbidResult>(result);
```

```
        }

        [Fact]
        public void OnPost_ValidAdminUser_AddsPostAndRedirects()
        {
            // Arrange
            var dbContext = GetTestDbContext();
            var userManager = GetMockUserManager();
            var adminSettings = Options.Create(new AdminSettings { AdminEmail =
"admin@example.com" });

            userManager.Setup(um =>
um.GetUserName(It.IsAny<ClaimsPrincipal>())).Returns("admin@example.com");

            var createModel = new CreateModel(dbContext, userManager.Object,
adminSettings)
            {
                Post = new Post { Title = "Test Post", Content = "This is a test
post." }
            };

            // Act
            var result = createModel.OnPost();

            // Assert
            Assert.IsType<RedirectToPageResult>(result);
            Assert.Equal(1, dbContext.Posts.CountAsync().Result); // Проверяем, что
пост был добавлен
        }

        [Fact]
        public void OnPost_InvalidModel_ReturnsPage()
        {
            // Arrange
            var dbContext = GetTestDbContext();
            var userManager = GetMockUserManager();
            var adminSettings = Options.Create(new AdminSettings { AdminEmail =
"admin@example.com" });

            userManager.Setup(um =>
um.GetUserName(It.IsAny<ClaimsPrincipal>())).Returns("admin@example.com");

            var createModel = new CreateModel(dbContext, userManager.Object,
adminSettings);
            createModel.ModelState.AddModelError("Title", "The Title field is
required."); // Добавляем ошибку модели

            // Act
            var result = createModel.OnPost();

            // Assert
            Assert.IsType<PageResult>(result);
        }
    }
}
```