Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Саратовский государственный технический университет

имени Гагарина Ю.А.»

Институт электронной техники и приборостроения

Кафедра Информационная безопасность автоматизированных систем

Специальность 10.05.03 Информационная безопасность автоматизированных систем

**РАСЧЕТНО-ГРАФИЧЕСКАЯ РАБОТА**

по дисциплине «Технологии и методы программирования»

**Тема: “Персональный блог”**

|  |  |
| --- | --- |
|  | Выполнил: студент 2 курса  учебной группы с2-ИБС-22  очной формы обучения  Коршиков И.А. Руководитель работы:  доцент каф. ИБС  Кочедаев А. В. |

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, подпись преподавателя)

Саратов 2023

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Саратовский государственный технический университет

имени Гагарина Ю.А.»

Кафедра Информационная безопасность автоматизированных систем

**Задание**

**на расчетно-графическую работу**

студенту 2 курса учебной группы с2-ИБС-22 Института электронной техники и приборостроения

Коршикову И.А.

**Тема “Персональный блог”**

Формулировка задания: создать веб-приложение персонального блога. Требуемый функционал: Регистрация пользователей. Создание блога (блогов). Создание и редактирование статей блога. Тэги статей. Поиск по тэгам, Комментирование статей.

Сроки выполнения работы

Дата сдачи

**Руководитель работы Кочедаев Алексей Викторович**

**Задание принял к исполнению Коршиков Игорь Андреевич**

Оглавление

[Введение 4](#_Toc137133636)

[Описание технологий 5](#_Toc137133637)

[ASP.NET MVC 5](#_Toc137133638)

[Microsoft SQL Server 6](#_Toc137133639)

[Entity Framework 8](#_Toc137133640)

[Архитектура MVC 9](#_Toc137133641)

[Базы данных 11](#_Toc137133642)

[Контроллеры приложения 13](#_Toc137133643)

[Результат 19](#_Toc137133644)

[Вывод 21](#_Toc137133645)

# Введение

Задание состоит в написании веб-приложения персонального блога. Требуемый функционал: Регистрация пользователей. Создание блога (блогов). Создание и редактирование статей блога. Тэги статей. Поиск по тэгам, Комментирование статей.

# Описание технологий

## ASP.NET MVC

ASP.NET MVC (Model-View-Controller) является одним из популярных фреймворков для разработки веб-приложений. Вот некоторые преимущества ASP.NET MVC:

1. Разделение логики приложения: ASP.NET MVC позволяет разделять код на модели, представления и контроллеры. Модель отвечает за обработку данных и бизнес-логику, представление отображает информацию пользователю, а контроллер управляет взаимодействием между моделью и представлением. Это разделение упрощает разработку, тестирование и поддержку приложений.
2. Гибкость и контроль над HTML-кодом: В отличие от других популярных веб-фреймворков, ASP.NET MVC позволяет разработчикам иметь полный контроль над генерируемым HTML-кодом. Это дает возможность создавать более гибкие и настраиваемые пользовательские интерфейсы.
3. Чистота URL-адресов: ASP.NET MVC поддерживает использование чистых и понятных URL-адресов. Это делает приложение более понятным для пользователей и поисковых систем, а также улучшает SEO-оптимизацию.
4. Поддержка тестирования: Фреймворк ASP.NET MVC обеспечивает хорошую поддержку для модульного и автоматизированного тестирования. Это позволяет разработчикам создавать надежные и стабильные приложения, что упрощает обнаружение и исправление ошибок.
5. Возможность работы с различными клиентскими технологиями: ASP.NET MVC позволяет разработчикам использовать различные клиентские технологии, такие как JavaScript-библиотеки и фреймворки (например, Angular, React или Vue.js), для создания богатого пользовательского интерфейса. Это обеспечивает большую гибкость в выборе клиентской части приложения.
6. Интеграция с другими инструментами и фреймворками Microsoft: ASP.NET MVC тесно интегрируется с другими инструментами и фреймворками Microsoft, такими как Entity Framework для работы с базами данных, ASP.NET Identity для аутентификации и авторизации пользователей, а также средствами развертывания и масштабирования приложений, предоставляемыми платформой Azure.

В целом, ASP.NET MVC предлагает разработчикам гибкий и мощный фреймворк для создания веб-приложений с отделением логики, хорошей поддержкой тестирования и возможностью интеграции с различными инструментами и технологиями Microsoft.

## Microsoft SQL Server

Microsoft SQL Server (MS SQL) - это реляционная система управления базами данных (СУБД), разработанная компанией Microsoft. Вот некоторые преимущества MS SQL:

1. Производительность: MS SQL обладает высокой производительностью и способностью обрабатывать большие объемы данных. Он предлагает оптимизированные механизмы обработки запросов, индексацию и кэширование данных, что позволяет значительно улучшить скорость выполнения запросов и обработку транзакций.
2. Надежность и безопасность: MS SQL обеспечивает высокую надежность и защиту данных. Он предлагает механизмы резервного копирования и восстановления данных, а также функции транзакционной целостности, которые обеспечивают сохранность данных при возникновении сбоев системы. Кроме того, MS SQL предлагает мощные средства безопасности для защиты данных от несанкционированного доступа и атак.
3. Широкий функционал: MS SQL предлагает множество функций и возможностей для разработки сложных баз данных и приложений. Он поддерживает различные типы данных, включая географические и геометрические данные, и предоставляет богатый набор функций для манипуляции данными, создания хранимых процедур, триггеров и представлений. MS SQL также поддерживает распределенные транзакции и взаимодействие с другими приложениями и технологиями Microsoft.
4. Масштабируемость и доступность: MS SQL предлагает возможности масштабирования и высокой доступности для обработки растущих объемов данных и обеспечения непрерывной работы системы. Он поддерживает репликацию, кластеризацию, масштабирование горизонтально и вертикально, а также предлагает инструменты для мониторинга и управления производительностью базы данных.
5. Интеграция с экосистемой Microsoft: MS SQL тесно интегрирован с другими продуктами и технологиями Microsoft, такими как .NET Framework, Visual Studio, Azure и другими серверными продуктами. Это позволяет упростить разработку, развертывание и управление приложениями, а также обеспечить совместимость и согласованность между различными компонентами системы.
6. Поддержка и сообщество: MS SQL имеет большое сообщество пользователей и разработчиков, которое предоставляет поддержку, обучающие материалы, форумы и ресурсы для обмена опытом и решения проблем. Это делает MS SQL привлекательным выбором для разработчиков, которым требуется поддержка и доступ к ресурсам сообщества.

В целом, MS SQL предлагает мощную, надежную и масштабируемую СУБД с широким функционалом и интеграцией в экосистему Microsoft. Это делает его популярным выбором для разработки и управления базами данных в различных сферах.

## Entity Framework

Entity Framework (EF) является объектно-реляционным отображением (ORM) для платформы .NET, разработанным компанией Microsoft. Вот некоторые преимущества Entity Framework:

1. Упрощение доступа к данным: Entity Framework предоставляет высокоуровневый и декларативный подход к работе с данными. Он позволяет разработчикам работать с объектами и классами, а не с прямыми SQL-запросами и таблицами базы данных. Это упрощает доступ к данным и позволяет сосредоточиться на бизнес-логике приложения.

2. Разделение между логикой приложения и базой данных: EF позволяет разработчикам разделять логику приложения и структуру базы данных. Он предоставляет возможность определения моделей данных, которые автоматически отображаются на соответствующие таблицы базы данных. Это позволяет изменять структуру базы данных, не затрагивая логику приложения, и наоборот.

3. Автоматическое управление отношениями и связями: EF обеспечивает автоматическое управление отношениями и связями между объектами. Он позволяет определить связи между таблицами базы данных и автоматически обрабатывает операции вставки, обновления и удаления данных, сохраняя консистентность и целостность данных.

4. Поддержка различных провайдеров баз данных: EF поддерживает различные провайдеры баз данных, включая Microsoft SQL Server, Oracle, MySQL, PostgreSQL и другие. Это означает, что вы можете использовать EF с вашей предпочитаемой базой данных без необходимости изучать специфические детали каждого провайдера.

5. Повышение производительности: EF включает в себя различные оптимизации производительности, такие как отложенная загрузка данных, кэширование запросов и предварительная компиляция запросов. Это позволяет снизить количество обращений к базе данных и улучшить производительность приложения.

6. Интеграция с другими технологиями Microsoft: EF тесно интегрирован с другими технологиями и инструментами Microsoft, такими как ASP.NET, WPF, WCF и Azure. Это обеспечивает более простую и гармоничную разработку и взаимодействие с другими компонентами приложения.

В целом, Entity Framework предлагает удобный и гибкий подход к работе с данными в приложениях .NET, упрощает доступ к данным, улучшает производительность и обеспечивает интеграцию с другими технологиями Microsoft.

# Архитектура MVC

Архитектура MVC (Model-View-Controller) является популярным паттерном проектирования, широко используемым во многих фреймворках разработки веб-приложений. Она разделяет логику приложения на три основных компонента: модель (Model), представление (View) и контроллер (Controller). Каждый из этих компонентов выполняет определенную роль и отвечает за свою часть функциональности приложения.

1. Модель (Model): Модель представляет бизнес-логику, данные и состояние приложения. Она отвечает за обработку данных, включая чтение, запись, валидацию и манипуляции с данными. Модель обычно состоит из классов или объектов, которые отражают структуру данных приложения и методы для работы с этими данными. Модель не зависит от представления или контроллера, что позволяет ей быть независимой и переиспользуемой.
2. Представление (View): Представление отображает данные пользователю и отвечает за отображение пользовательского интерфейса. Оно получает данные от модели и форматирует их для вывода пользователю в нужном виде (например, HTML, XML или JSON). Представление также может обрабатывать пользовательский ввод, например, формы или ссылки. Представление должно быть независимым от модели и контроллера, и его задача заключается только в отображении данных и взаимодействии с пользователем.
3. Контроллер (Controller): Контроллер является посредником между моделью и представлением. Он принимает входные запросы от пользователя (например, HTTP-запросы) и обрабатывает их, выполняя соответствующие операции в модели. Контроллер также выбирает соответствующее представление для отображения результатов операций модели и передает ему необходимые данные. Контроллер обычно содержит методы действий (action methods), которые связываются с определенными URL-маршрутами или событиями пользовательского взаимодействия.

Взаимодействие между компонентами архитектуры MVC происходит следующим образом: пользователь взаимодействует с интерфейсом, представление передает данные и действия контроллеру, контроллер обрабатывает эти данные и вызывает соответствующие методы модели для выполнения операций, а затем контроллер выбирает и передает данные представлению для отображения результатов пользователю.

Архитектура MVC позволяет разделить ответственность между компонентами, что облегчает разработку, тестирование и поддержку приложений. Она способствует повышению модульности, переиспользованию кода и разделению логики приложения от визуализации, что делает приложения более гибкими и расширяемыми.

# Базы данных

ASP.NET Identity — это система управления аутентификацией и авторизацией, предоставляемая платформой ASP.NET. При использовании ASP.NET Identity база данных играет важную роль в хранении информации о пользователях, идентификационных данных и настройках безопасности. Вот общий обзор того, как работает база данных в ASP.NET Identity:

1. Структура базы данных:

ASP.NET Identity использует реляционные базы данных для хранения данных. По умолчанию, создается база данных с таблицами, такими как "AspNetUsers", "AspNetRoles", "AspNetUserRoles", "AspNetUserClaims" и другими, которые предоставляют структуру для хранения информации о пользователях, ролях, привязках ролей к пользователям и прочих данных, связанных с аутентификацией и авторизацией.

2. Поставщики данных:

ASP.NET Identity использует понятие "поставщиков данных" для обеспечения гибкости и возможности использования различных источников данных. Встроенные поставщики данных для ASP.NET Identity предоставляют реализации для работы с различными СУБД, такими как Microsoft SQL Server, MySQL и другими.

3. Конфигурация базы данных:

Для настройки базы данных в ASP.NET Identity необходимо определить строку подключения к базе данных в файле конфигурации приложения (например, web.config). Строка подключения указывает на используемую базу данных и ее параметры, такие как сервер, имя базы данных, учетные данные и протоколы подключения.

4. Миграции базы данных:

ASP.NET Identity также предоставляет механизм миграций базы данных. Миграции позволяют изменять схему базы данных, вносить изменения в таблицы или добавлять новые таблицы без необходимости выполнять вручную SQL-скрипты. Механизм миграций автоматически применяет эти изменения при запуске приложения или вручную через команды управления миграциями.

5. Работа с данными:

ASP.NET Identity предоставляет API для взаимодействия с базой данных. Через этот API можно выполнять операции, такие как создание, чтение, обновление и удаление пользователей, ролей, привязок и других данных, связанных с аутентификацией и авторизацией. Методы и классы API предоставляют удобные средства для работы с базой данных без необходимости написания ручного SQL-кода.

В целом, база данных в ASP.NET Identity служит для хранения и управления информацией о пользователях, ролях и настройках безопасности. ASP.NET Identity обеспечивает гибкость в выборе и настройке базы данных, предоставляет удобный API для работы с данными и поддерживает механизм миграций для изменения схемы базы данных.

В таблице со статьями хранится информация о статье: название, текст, дата создания и т. д.

Статьи с владельцами соотносятся по полю UserID у статьи.

Схема базы данных представлена на рисунке 1.

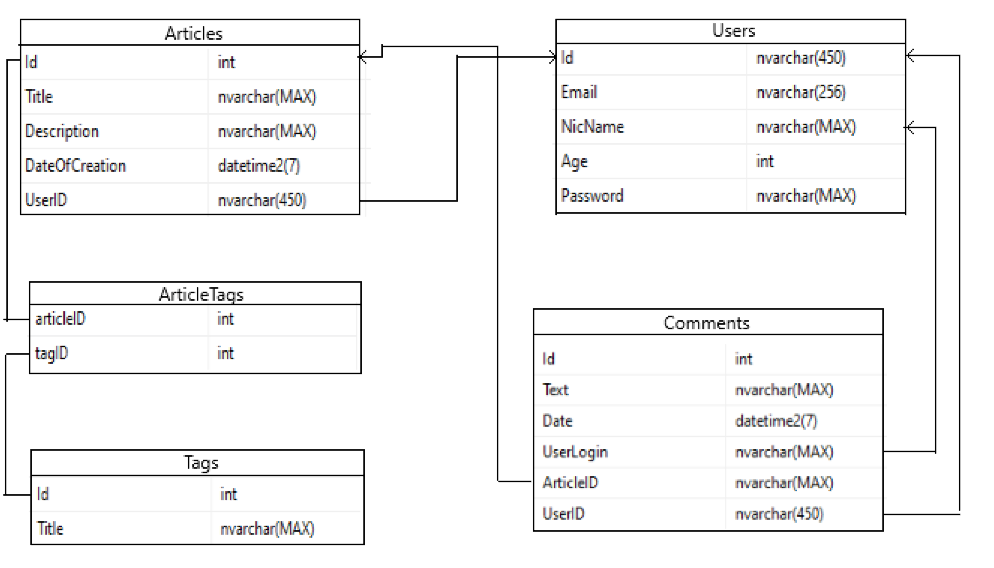


Рисунок 1. Схема базы данных

# Контроллеры приложения

Это код контроллера "AccountController" в ASP.NET-приложении, отвечающий за обработку операций регистрации и авторизации пользователей. В данном коде реализованы следующие функциональности:

* В конструкторе класса определены приватные поля \_signInManager и \_userManager, которые представляют менеджеры аутентификации и управления пользователями соответственно.
* Контроллер содержит действия для обработки GET- и POST-запросов на аутентификацию и регистрацию пользователей, а также выхода из системы (Logout).
* Действие Login отображает страницу входа пользователя и обрабатывает POST-запрос на аутентификацию. Оно использует SignInManager.PasswordSignInAsync для проверки правильности введенного пароля и аутентификации пользователя.
* Действие Register отображает страницу регистрации пользователя и обрабатывает POST-запрос на создание нового пользователя. Оно использует UserManager.CreateAsync для создания нового пользователя.
* Действие Logout обрабатывает POST-запрос на выход из системы. Оно использует SignInManager.SignOutAsync для удаления аутентификационных куки и перенаправляет пользователя на домашнюю страницу.

Этот код позволяет реализовать функциональность аутентификации и авторизации пользователей в ASP.NET-приложении, обеспечивая безопасный доступ к защищенным разделам приложения.

namespace BlogWebApp.Controllers

{

public class AccountController : Controller

{

private readonly UserManager<ArticleUser> \_userManager;

private readonly SignInManager<ArticleUser> \_signInManager;

public AccountController(UserManager<ArticleUser> userManager, SignInManager<ArticleUser> signInManager)

{

\_userManager = userManager;

\_signInManager = signInManager;

}

[HttpGet]

public IActionResult Register()

{

return View();

}

[HttpPost]

public async Task<IActionResult> Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

ArticleUser user = new ArticleUser { Email = model.Email, Login = model.NicName, UserName = model.NicName, Age = model.Age };

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

if (result.Succeeded)

{

await \_signInManager.SignInAsync(user, false);

return RedirectToAction("Index", "Home");

}

else

{

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

}

return View(model);

}

[HttpGet]

public IActionResult Login(string returnUrl = null)

{

return View(new LoginViewModel { ReturnUrl = returnUrl });

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Login(LoginViewModel model)

{

if (ModelState.IsValid)

{

var result = await \_signInManager.PasswordSignInAsync(model.Login, model.Password, model.RememberMe, false);

if (result.Succeeded)

{

if (!string.IsNullOrEmpty(model.ReturnUrl) && Url.IsLocalUrl(model.ReturnUrl))

{

return Redirect(model.ReturnUrl);

}

else

{

return RedirectToAction("Index", "Home");

}

}

else

{

ModelState.AddModelError("", "Неправильный логин и (или) пароль");

}

}

return View(model);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Logout()

{

await \_signInManager.SignOutAsync();

return RedirectToAction("Index", "Home");

}

}

}

Этот код представляет собой контроллер "HomeController" в ASP.NET-приложении. В данном коде реализованы следующие функциональности:

1. В начале контроллера создается экземпляр контекста данных "ApplicationContext", который представляет доступ к базе данных с помощью Entity Framework.
2. Метод Index отвечает за отображение главной страницы блога. Он и осуществляет отображение списка статей. Метод использует LINQ-запросы для выборки данных из базы данных.
3. Метод CreateArticle обрабатывает GET- и POST-запросы для создания новой статьи в базе данных. Если модель данных Article проходит валидацию, создается новая статья и сохраняется в базе данных.
4. Метод Details отображает подробную информацию о выбранной статье на основе ее идентификатора.
5. Методы Edit, Delete и ConfirmDelete обрабатывают действия связанные с редактированием, удалением и подтверждением удаления соответственно. Эти методы требуют аутентификации пользователя.
6. Метод SearchArticle осуществляет поиск статьи по выбранному тегу.
7. Методы CreateTag создают новый тег и связывают его со статьёй в базе данных.
8. Методы CreateComment осуществляют создание комментария к выбранной статье в базе данных.
9. Метод SeeComment отображает список коментариев к выбранной статье.

Этот контроллер обеспечивает логику обработки запросов и взаимодействие с базой данных для страницы аукциона и других функциональностей приложения.

namespace BlogWebApp.Controllers

{

public class HomeController : Controller

{

private ApplicationContext db;

private readonly UserManager<ArticleUser> \_userManager;

public HomeController(UserManager<ArticleUser> userManager, ApplicationContext context)

{

\_userManager = userManager;

db = context;

}

public async Task<IActionResult> Index()

{

string id = \_userManager.GetUserId(HttpContext.User);

return View(db.Articles.Where(p => p.UserID == id));

}

[HttpGet]

public IActionResult CreateArticle()

{

return View();

}

[HttpPost]

public async Task<IActionResult> CreateArticle(Article article)

{

article.DateOfCreation = DateTime.Now;

article.UserID = \_userManager.GetUserId(HttpContext.User);

db.Articles.Add(article);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

public async Task<IActionResult> Details(int? id)

{

if (id != null)

{

Article article = await db.Articles.FirstOrDefaultAsync(p => p.Id == id);

if (article != null)

return View(article);

}

return NotFound();

}

public async Task<IActionResult> Edit(int? id)

{

if (id != null)

{

Article article = await db.Articles.FirstOrDefaultAsync(p => p.Id == id);

if (article != null)

return View(article);

}

return NotFound();

}

[HttpPost]

public async Task<IActionResult> Edit(Article article)

{

article.UserID = \_userManager.GetUserId(HttpContext.User);

article.DateOfCreation = DateTime.Now;

db.Articles.Update(article);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

[HttpGet]

[ActionName("Delete")]

public async Task<IActionResult> ConfirmDelete(int? id)

{

if (id != null)

{

Article article = await db.Articles.FirstOrDefaultAsync(p => p.Id == id);

if (article != null)

return View(article);

}

return NotFound();

}

[HttpPost]

public async Task<IActionResult> Delete(int? id)

{

if (id != null)

{

Article article = await db.Articles.FirstOrDefaultAsync(p => p.Id == id);

if (article != null)

{

db.Articles.Remove(article);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

}

return NotFound();

}

public IActionResult CreateTag()

{

string id = \_userManager.GetUserId(HttpContext.User);

ViewData["ArticleID"] = new SelectList(db.Articles.Where(p => p.UserID == id), "Id", "Title");

ViewData["TagID"] = new SelectList(db.Tags, "Id", "Title");

return View();

}

[HttpPost]

public async Task<IActionResult> CreateTag(int? articleid, [Bind("articleID,tagID")] ArticleTag articleTag)

{

if (ModelState.IsValid)

{

db.Add(articleTag);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

return View();

}

public IActionResult SearchArticl()

{

ViewData["TagID"] = new SelectList(db.Tags, "Id", "Title");

return View();

}

[HttpPost]

public IActionResult SearchArticle(int? TagId)

{

return View(db.ArticleTags.Include(p => p.article).Where(s => s.tagID == TagId).Select(p=>p.article).ToList());

}

public IActionResult CreateComment()

{

return View();

}

[HttpPost]

public async Task <IActionResult> CreateComment(Comment comment)

{

comment.Date = DateTime.Now;

comment.UserID = \_userManager.GetUserId(HttpContext.User);

comment.UserLogin = User.Identity.Name;

db.Comments.Add(comment);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

public IActionResult SeeComment()

{

return View(db.Comments);

}

}

}

# Результат

На рисунке 2 изображена страница входа в аккаунт.

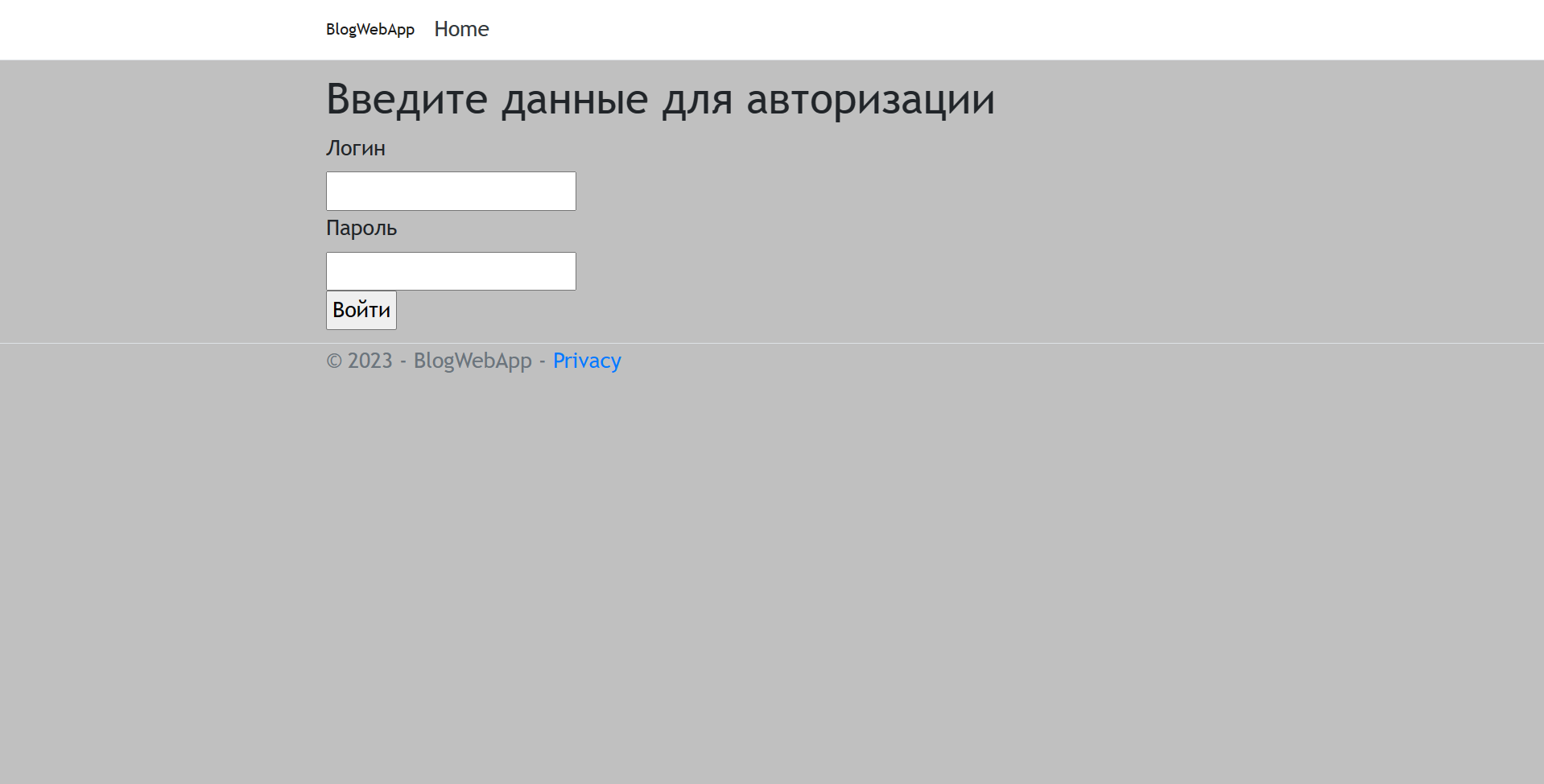


Рисунок 2. Страница входа в аккаунт

На рисунке 3 изображена страница добавления лота.

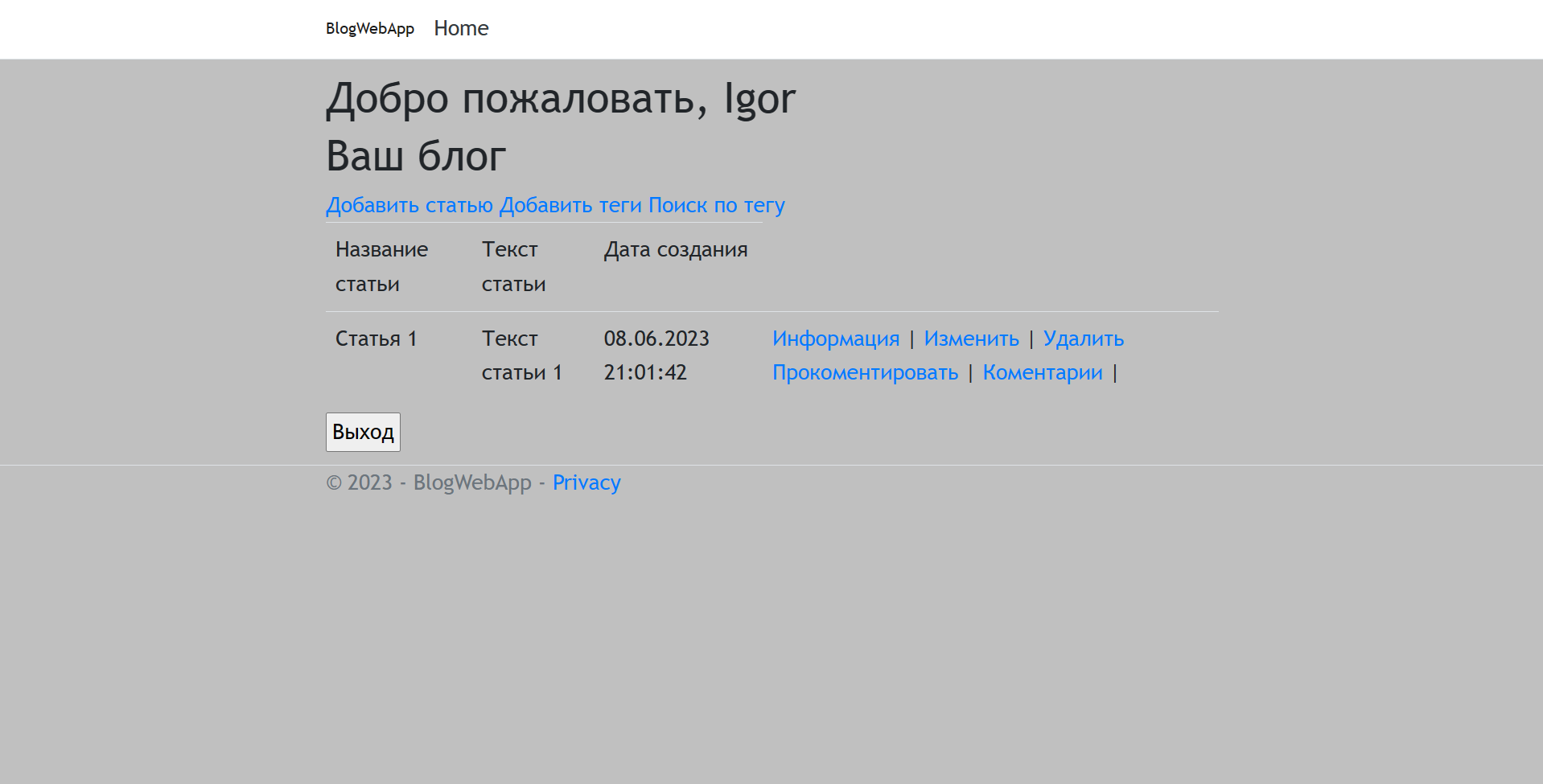


Рисунок 3. Страница блога

На рисунке 4 изображена страница поиска по тегу.

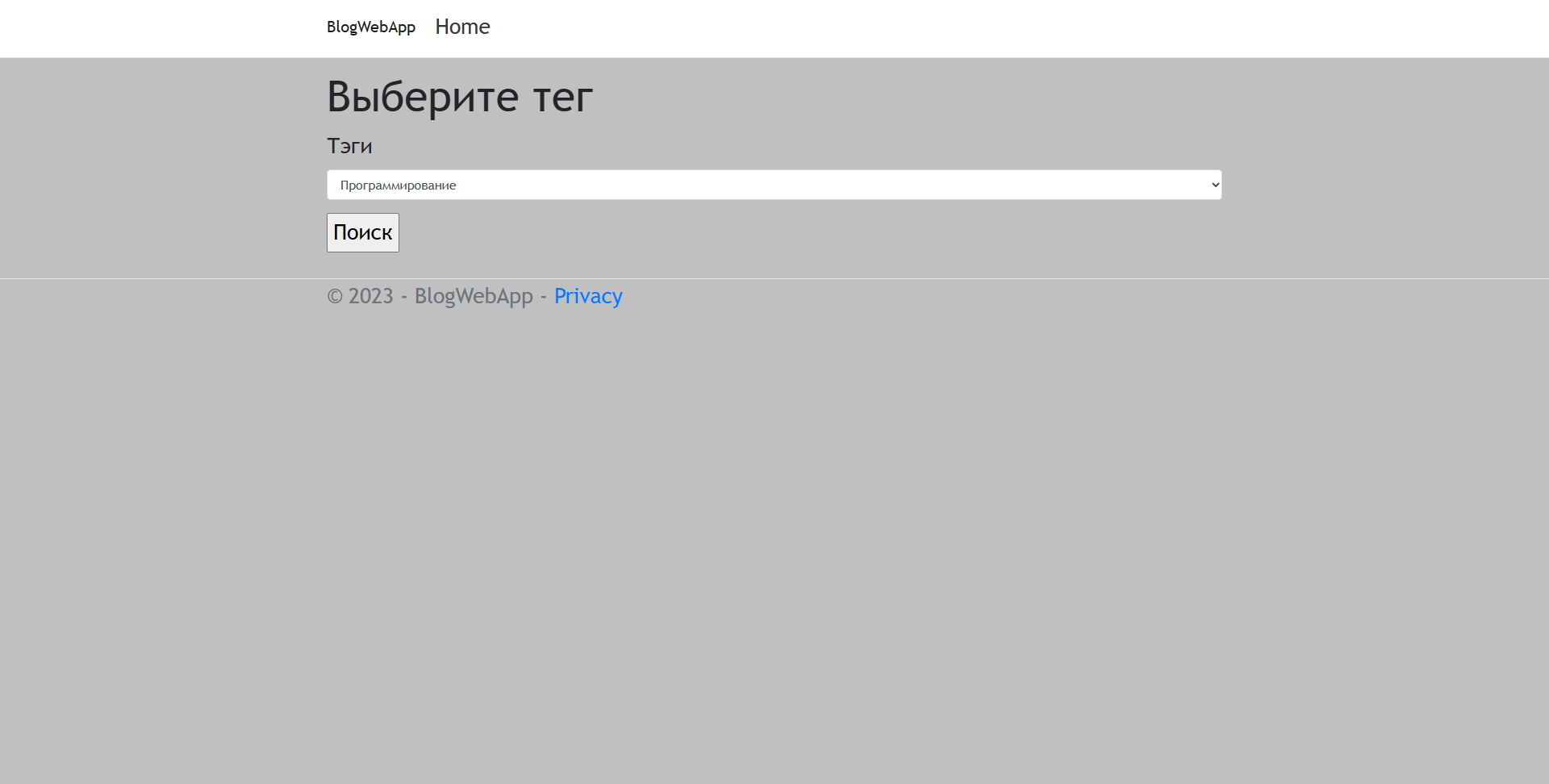


Рисунок 4. Страница поиска по тегу

# Вывод

Для создания веб-приложения аукциона с требуемым функционалом были использованы технологии и инструменты, такие как ASP.NET MVC, Microsoft SQL Server и ASP.NET Identity.

ASP.NET MVC позволяет эффективно разделить логику приложения на модель, представление и контроллер. Модель представляет данные, контроллер обрабатывает запросы и управляет взаимодействием с моделью и представлением, а представление отвечает за отображение данных пользователю.

Для реализации просмотра статей и поиска по тегу можно создать соответствующие методы в контроллере, которые будут извлекать данные из базы данных и передавать их в представление для отображения. Пользователь может просматривать список статей и использовать функцию поиска для нахождения статей по их тегам.

Добавление и редактирование статей также реализуется через соответствующие методы в контроллере. Пользователь может создавать новые статьи и редактировать уже существующие.

Для регистрации пользователей используется ASP.NET Identity. Пользователи могут создавать учетные записи, указывая свои данные.

В итоге, созданное веб-приложение персонального блога позволяет пользователям просматривать статьи, осуществлять поиск по тегам, добавлять и редактировать статьи, регистрироваться в системе редактировать список статей. Это приложение обладает необходимым функционалом для эффективного ведения персонального блога.