Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра Информатики

Специальность «ИиТП»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

«Блогхост»

БГУИР КП 1–40 04 01 ПЗ

Выполнил: Руководитель:

студент группы 753503 Удовин И. А.

Попов В.А.

Минск 2020

Оглавление

[**Введение** 3](#_Toc58636918)

[**1.** **Инструменты разработки** 4](#_Toc58636919)

[**1.1.** **Назначение разработки** 4](#_Toc58636920)

[**1.2.** **Перечень основных выполняемых функций** 4](#_Toc58636921)

[**1.3.** **Входные и выходные данные пользователя** 4](#_Toc58636922)

[**1.4.** **Требования к надежности** 4](#_Toc58636923)

[**1.5.** **Требования к эргономике** 5](#_Toc58636924)

[**1.6.** **Требования к организации системы** 6](#_Toc58636925)

[**1.7.** **Обоснование выбора языка программирования и среды разработки** 7](#_Toc58636926)

[**2.** **Моделирование предметной области** 8](#_Toc58636927)

[**2.1.** **Обзор используемых технологий** 8](#_Toc58636928)

[**2.1.1.** **C#** 8](#_Toc58636929)

[**2.1.2.** **ASP.Net Core 2.2** 11](#_Toc58636930)

[**2.1.3.** **RazorPages** 13](#_Toc58636931)

[**2.1.4.** **Git / GitHub** 14](#_Toc58636932)

[**2.1.5.** **SQLite** 18](#_Toc58636933)

[**2.2.** **Информация о базе данных** 19](#_Toc58636934)

[**3.** **Реализация приложения** 21](#_Toc58636935)

[**Заключение** 22](#_Toc58636936)

[**Список используемых источников** 23](#_Toc58636937)

[**Приложение 1. Код программы** 24](#_Toc58636938)

# **Введение**

Блог (англ. blog, от web log — интернет-журнал событий, интернет-дневник, онлайн-дневник) — веб-сайт, основное содержимое которого — регулярно добавляемые человеком записи, содержащие текст, изображения или мультимедиа. Для блогов характерна возможность публикации отзывов (комментариев) посетителями; она делает блоги средой сетевого общения, имеющей ряд преимуществ перед электронной почтой, группами новостей и чатами.

Большинство блогеров говорят, что ведут или читают блоги ради общения с интересными им людьми. В первую очередь это возможность сказать что-то один раз так, чтобы это услышали многие. Какой смысл рассказывать десяти, двадцати, тридцати знакомым о поездке в пригородный парк, если это можно описать в своём блоге, украсив запись фотографиями? Каждый прочтёт об этом в удобное ему время или не будет читать вовсе, решив, что это ему неинтересно.

1. **Инструменты разработки**
   1. **Назначение разработки**

Блогхост – это проект, позволяющий людям создавать и вести собственные блоги, просматривать и оценивать записи других людей, а также общаться в лайвчате.

* 1. **Перечень основных выполняемых функций**

Блогхост разделяет пользователей на три группы:

* Обычный пользователей (user) – может авторизоваться, просматривать чужие посты и комментировать их
* Модератор (moder) – может авторизоваться, создавать посты и комментировать их
* Админ (admin) – может авторизоваться, создавать посты, комментировать их, а также менять роли остальных пользователей

Интегрирован чат с использованием технологии SignalR, который позволяет прогружать сообщения без обновления страницы.

Интегрирован rich text editor TinyMce в содержание поста.

При публикации поста, а также если на этот пост был оставлен комментарий, то автору на электронную почту будет выслано уведомление.

* 1. **Входные и выходные данные пользователя**

Пользователь должен иметь возможность зайти в систему используя уникальный никнейм и пароль, который должен соответствовать определенным требованиям (длина более 8 символов, символы английского алфавита и цифры из которых обязательно должно быть: хотя бы 1 буква в верхнем регистре, 1 в нижнем, 1 цифра, 1 не буквенно-циферный символ)

* 1. **Требования к надежности**

Обычный пользователь (user) не должен иметь возможность напрямую влиять на содержание поста, а также не должен иметь возможность изменять свою роль, роли других пользователей, а также создавать новые посты.

* 1. **Требования к эргономике**

Пользовательский интерфейс сайта должен отвечать современным  
требованиям к эргономике и технической эстетике.  
Внутренний пользовательский интерфейс сайта должен позволять  
администратору относительно свободно ориентироваться в  
информационном и функциональном пространстве системы и  
удовлетворять следующим требованиям:

1. Однозначно понимаемое назначение названий пунктов меню  
(функциональных элементов интерфейса) или их графических  
изображений.

2. Группировка элементов интерфейса по функциональному  
признаку.

3. Минимизация вертикальной и горизонтальной прокрутки.

4. Четко сформулированные и понятные пользователю сообщения  
об ошибках.

5. Цветовая гамма, выдержанная в спокойных тонах (или, как  
минимум, не должна идти кровь из глаз при просмотре  
интерфейса), не раздражающих пользователя.

Внешний пользовательский интерфейс сайта должен удовлетворять следующим требованиям:

1. Адекватно отображаться в зависимости от типа подключения

пользователя (пользователи, работающие через модем;

пользователи, работающие через высокоскоростной канал

доступа).

2. Обеспечивать большую скорость загрузки страниц, достигаемую

в результате оптимизации графических элементов.

3. Обеспечивать минимум усилий и временных затрат пользователя

для навигации по страницам сайта.

4. Корректно отображаться при различных разрешениях и

количестве одновременно отображаемых цветов монитора.

5. Сохранять идентичность отображения на большинстве

современных ОС и Web-браузерах (Mozilla Firefox, начиная с

первой версии Quantum и до текущей версии; Opera, начиная с

версии 9 и до текущей версии; Google Chrome начиная с версии

70 и до текущей версии)

* 1. **Требования к организации системы**

Система должна состоять из мало зависимых подсистем (отдельных приложений), которые обеспечивают работу всей системы в целом. Данные подсистемы должны реализовывать свой функционал и развиться как самостоятельные подсистемы.

Можно использовать классический паттерн проектирование MVC(Model-View-Controller).

Статическая страница на HTML не умеет реагировать на действия пользователя. Для двухстороннего взаимодействия нужны динамические веб-страницы. MVC — ключ к пониманию разработки динамических веб-приложений, поэтому разработчику нужно знать эту модель.

MVC расшифровывается как модель-представление-контроллер. Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения. Компоненты MVC:

* Модель — этот компонент отвечает за данные, а также определяет структуру приложения. Например, если вы создаете To-Do приложение, код компонента model будет определять список задач и отдельные задачи.
* Представление — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента view определяет внешний вид приложения и способы его использования.
* Контроллер — этот компонент отвечает за связь между model и view. Код компонента controller определяет, как сайт реагирует на действия пользователя. По сути, это мозг MVC-приложения.
  1. **Обоснование выбора языка программирования и среды разработки**

В качестве языка программирования был выбран C#, в качестве среды разработки – Visual Studio Community 2019. Эта среда разработки, как и сам язык создана компанией Microsoft, следовательно, писать код получается максимально нативно.

1. **Моделирование предметной области**
   1. **Обзор используемых технологий**
      1. **C#**

C# (произносится си шарп) — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров компании Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумота как язык разработки приложений для платформы Microsoft .NET Framework.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Pascal, Модула, Smalltalk и, в особенности, Java — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

Из-за технических ограничений на отображение (стандартные шрифты, браузеры и т. д.) и того, что знак диеза ♯ не представлен на стандартной клавиатуре компьютера, при записи имени языка программирования используют знак решётки (#). Это соглашение отражено в Спецификации языка C# ECMA-334. Тем не менее, на практике (например, при размещении рекламы и коробочном дизайне), «Майкрософт» использует знак диеза.

Названия языков программирования не принято переводить, поэтому язык называют, используя транскрипцию, — «Си шарп».

*Общая информация по версиям*

|  |  |
| --- | --- |
| **Версия** | **Нововведения** |
| **C# 2.0** | * Частичные типы * Обобщённые типы (*generics*) * Итераторы и ключевое слово yield * Анонимные методы * Оператор *null*-объединения * *Nullable*-типы |
| **C# 3.0** | * Запросы, интегрированные в язык (*LINQ*) * Инициализаторы объектов и коллекций * Лямбда-выражения * Деревья выражений * Неявная типизация и ключевое слово var * Анонимные типы * Методы расширения * Автоматические свойства |
| **C# 4.0** | * Динамическое связывание и ключевое слово dynamic * Именованные и опциональные аргументы * Обобщенная [ковариантность и контрвариантность](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B2%D0%B0%D1%80%D0%B8%D0%B0%D0%BD%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B8_%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%B0%D0%B2%D0%B0%D1%80%D0%B8%D0%B0%D0%BD%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) * Библиотека *TPL*, концепция задач и классы Task, Parallel * Класс MemoryCache * Классы параллельных коллекций |
| **C# 5.0** | * Шаблон *TAP* * Асинхронные методы async и await * Сведения о вызывающем объекте |
| **C# 6.0** | * Компилятор как сервис * Импорт членов статических типов в пространство имён * Фильтры исключений * await в блоках catch/finally * Инициализаторы автосвойств * Автосвойства только для чтения * *null*-условные операции (?. и ?[]) * Интерполяция строк * Оператор nameof * Инициализатор словаря * Функции сжатые до выражений |
| **C# 7.0** | * out-переменные * Сопоставление с шаблоном * Шаблоны с is * Шаблоны и выражение switch * Кортежи * Распаковка кортежей (деконструкторы) * Локальные функции * Улучшения литералов * Локальные переменные и возвращаемые значения по ссылке * Расширение списка типов, возвращаемых асинхронными методами * Больше членов класса в виде выражений * throw выражения |
| **C# 8.0** | * Члены только для чтения * Члены интерфейса по умолчанию * Улучшения сопоставления шаблонов * Объявления using * Статические локальные функции * Удаляемые ссылочные структуры * Ссылочные типы, допускающие значение NULL * Асинхронные потоки * Индексы и диапазоны * Присваивание объединения со значением NULL * Неуправляемые сконструированные типы * Выражения stackalloc во вложенных выражениях * Больше членов класса в виде выражений * Улучшения интерполированных строк |

* + 1. **ASP.Net Core 2.2**

Платформа ASP.NET Core представляет технологию от компании Microsoft, предназначенную для создания различного рода веб-приложений: от небольших веб-сайтов до крупных веб-порталов и веб-сервисов.

С одной стороны, ASP.NET Core является продолжением развития платформы ASP.NET. Но с другой стороны, это не просто очередной релиз. Выход ASP.NET Core фактически означает революцию всей платформы, ее качественное изменение.

Разработка над платформой началась еще в 2014 году. Тогда платформа условно называлась ASP.NET vNext. В июне 2016 года вышел первый релиз платформы. А в декабре 2019 года вышла версия ASP.NET Core 3.1, которая собственно и будет охвачена в текущем руководстве.

ASP.NET Core теперь полностью является opensource-фреймворком. Все исходные файлы фреймворка доступны на GitHub.

ASP.NET Core может работать поверх кросс-платформенной среды .NET Core, которая может быть развернута на основных популярных операционных системах: Windows, Mac OS, Linux. И таким образом, с помощью ASP.NET Core мы можем создавать кросс-платформенные приложения. И хотя Windows в качестве среды для разработки и развертывания приложения до сих пор превалирует, но теперь уже мы не ограничены только этой операционной системой. То есть мы можем запускать веб-приложения не только на ОС Windows, но и на Linux и Mac OS. А для развертывания веб-приложения можно использовать традиционный IIS, либо кросс-платформенный веб-сервер Kestrel.

Благодаря модульности фреймворка все необходимые компоненты веб-приложения могут загружаться как отдельные модули через пакетный менеджер Nuget. Кроме того, в отличие от предыдущих версий платформы нет необходимости использовать библиотеку System.Web.dll.

ASP.NET Core включает в себя фреймворк MVC, который объединяет функциональность MVC, Web API и Web Pages. В предыдущих версии платформы данные технологии реализовались отдельно и поэтому содержали много дублирующей функциональности. Сейчас же они объединены в одну программную модель ASP.NET Core MVC. А Web Forms полностью ушли в прошлое.

Кроме объединения вышеупомянутых технологий в одну модель в MVC был добавлен ряд дополнительных функций.

Одной из таких функций являются тэг-хелперы (tag helper), которые позволяют более органично соединять синтаксис html с кодом С#.

ASP.NET Core характеризуется расширяемостью. Фреймворк построен из набора относительно независимых компонентов. И мы можем либо использовать встроенную реализацию этих компонентов, либо расширить их с помощью механизма наследования, либо вовсе создать и применять свои компоненты со своим функционалом.

Также было упрощено управление зависимостями и конфигурирование проекта. Фреймворк теперь имеет свой легковесный контейнер для внедрения зависимостей, и больше нет необходимости применять сторонние контейнеры, такие как Autofac, Ninject. Хотя при желании их также можно продолжать использовать.

В качестве инструментария разработки мы можем использовать последние выпуски Visual Studio, начиная с версии Visual Studio 2015. Кроме того, мы можем создавать приложения в среде Visual Studio Code, которая является кросс-платформенной и может работать как на Windows, так и на Mac OS X и Linux.

Если суммировать, то можно выделить следующие ключевые отличия ASP.NET Core от предыдущих версий ASP.NET:

* Новый легковесный и модульный конвейер HTTP-запросов
* Возможность развертывать приложение как на IIS, так и в рамках своего собственного процесса
* Использование платформы .NET Core и ее функциональности
* Распространение пакетов платформы через NuGet
* Интегрированная поддержка для создания и использования пакетов NuGet
* Единый стек веб-разработки, сочетающий Web UI и Web API
* Конфигурация для упрощенного использования в облаке
* Встроенная поддержка для внедрения зависимостей
* Расширяемость
* Кроссплатформенность: возможность разработки и развертывания приложений ASP.NET на Windows, Mac и Linux
* Развитие как open source, открытость к изменениям

Эти и другие особенности, и возможности стали основой для новой модели программирования.

Новое в версии ASP.NET Core 2.2:

* улучшение работы с Web API
* микросервисы и Azure
* улучшение производительности
* поддержка многоэтапной JIT-компиляции
* поддержка расширений SQL Server и SQLite для EF Core
* Обновленные шаблоны Bootstrap и Angular, версий 4 и 6 соответственно
* добавление поддержки API Security + включая HTTP/2 для всех соединений ASP.NET Core.
  + 1. **RazorPages**

Razor Pages —фича, появившаяся в Core.Net 2.0. Razor Page — это страница, состоящая из стандартной разметки (View) и бэкенд класса. В каком-то смысле напоминает Web Forms только без поддержки сохранения состояния. Преимущество такого решения очевидно — мы избавляемся от ненужной прослойки — модели страницы (модель данных в виде например Entity это само собой). Бэкенд страницы является и контроллером и моделью — классика ООП — инкапсуляция данных и методов работы с ними в одном объекте. В конце концов модель страницы — это просто класс, нет никаких причин почему этим классом не может быть контроллер.

Иными словами, Razor Pages — более вменяемое решение для веба чем MVC, теперь мы имеем дело с традиционным и логичным понятием «страница» а не с контролерами и моделями разбросанными по всему проекту. Но поскольку .NET будет развиваться по направлению Core.Net то вряд ли Razor Page появятся в стандартном фреймворке, несмотря на то что ближайшие годы большинство проектов будет оставаться на стандартном .NET. Тем не менее можно изобразить функциональность Razor Pages и на стандартном фреймворке.

* + 1. **Git / GitHub**

Git — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. На сегодняшний день его поддерживает Джунио Хамано.

Среди проектов, использующих Git — ядро Linux, Swift, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, DokuWiki, Qt, ряд дистрибутивов Linux.

Программа является свободной и выпущена под лицензией GNU GPL версии 2. По умолчанию используется TCP порт 9418.

Git поддерживает быстрое разделение и слияние версий, включает инструменты для визуализации и навигации по нелинейной истории разработки. Как и Darcs, BitKeeper, Mercurial, Bazaar и Monotone, Git предоставляет каждому разработчику локальную копию всей истории разработки, изменения копируются из одного репозитория в другой.

Удаленный доступ к репозиториям Git обеспечивается git-демоном, SSH- или HTTP-сервером. TCP-сервис git-daemon входит в дистрибутив Git и является наряду с SSH наиболее распространённым и надёжным методом доступа.

Ядро Git представляет собой набор утилит командной строки с параметрами. Все настройки хранятся в текстовых файлах конфигурации. Такая реализация делает Git легко портируемым на любую платформу и дает возможность легко интегрировать Git в другие системы (в частности, создавать графические git-клиенты с любым желаемым интерфейсом).

Репозиторий Git представляет собой каталог файловой системы, в котором находятся файлы конфигурации репозитория, файлы журналов, хранятся операции, выполняемые над репозиторием, индекс, описывающий расположение файлов, и хранилище, содержащее собственно файлы.

Структура хранилища файлов не отражает реальную структуру хранящегося в репозитории файлового дерева, она ориентирована на повышение скорости выполнения операций с репозиторием. Когда ядро обрабатывает команду изменения (неважно, при локальных изменениях или при получении патча от другого узла), оно создает в хранилище новые файлы, соответствующие новым состояниям изменённых файлов. Существенно, что никакие операции не изменяют содержимого уже существующих в хранилище файлов.

По умолчанию репозиторий хранится в подкаталоге с названием «.git» в корневом каталоге рабочей копии дерева файлов, хранящегося в репозитории. Любое файловое дерево в системе можно превратить в репозиторий git, отдав команду создания репозитория из корневого каталога этого дерева (или указав корневой каталог в параметрах программы). Репозиторий может быть импортирован с другого узла, доступного по сети. При импорте нового репозитория автоматически создается рабочая копия, соответствующая последнему зафиксированному состоянию импортируемого репозитория (то есть не копируются изменения в рабочей копии исходного узла, для которых на том узле не была выполнена команда commit).

Нижний уровень git является так называемой контентно-адресуемой файловой системой. Инструмент командной строки git содержит ряд команд по непосредственной манипуляции этим репозиторием на низком уровне. Эти команды не нужны при нормальной работе с git как с системой контроля версий, но нужны для реализации сложных операций (ремонт повреждённого репозитория и так далее), а также дают возможность создать на базе репозитория git свое приложение.

Для каждого объекта в репозитории вычисляется SHA-1-хеш, и именно он становится именем файла, содержащего данный объект в каталоге .git/objects. Для оптимизации работы с файловыми системами, не использующими деревья для каталогов, первый байт хеша становится именем подкаталога, а остальные — именем файла в нём, что снижает количество файлов в одном каталоге (ограничивающий фактор производительности на таких устаревших файловых системах).

В классическом обычном сценарии в репозитории git есть три типа объектов — файл, дерево и «коммит». Файл есть какая-то версия какого-то пользовательского файла, дерево — совокупность файлов из разных подкаталогов, «коммит» — дерево и некая дополнительная информация (например, родительские коммиты, а также комментарий).

Репозиторий Git бывает локальный и удаленный. Локальный репозиторий — это подкаталог .git, создаётся (в пустом виде) командой git init и (в непустом виде с немедленным копированием содержимого родительского удалённого репозитория и простановкой ссылки на родителя) командой git clone.

Практически все обычные операции с системой контроля версий, такие, как коммит и слияние, производятся только с локальным репозиторием. Удалённый репозиторий можно только синхронизировать с локальным как «вверх» (push), так и «вниз» (pull).

Наличие полностью всего репозитория проекта локально у каждого разработчика даёт Git ряд преимуществ перед SVN. Так, например, все операции, кроме push и pull, можно осуществлять без наличия интернет-соединения.

Очень мощной возможностью git являются ветви, реализованные куда более полно, чем в SVN: по сути, ветвь git есть не более чем именованная ссылка, указывающая на некий коммит в репозитории (используется подкаталог refs). Коммит без создания новой ветви всего лишь передвигает эту ссылку на себя, а коммит с созданием ветви — оставляет старую ссылку на месте, но создает новую на новый коммит, и объявляет её текущей. Заменить локальные девелоперские файлы на набор файлов из иной ветви, тем самым перейдя к работе с ней — так же тривиально.

Команда push передает все новые данные (те, которых еще нет в удалённом репозитории) из локального репозитория в репозиторий удаленный. Для исполнения этой команды необходимо, чтобы удалённый репозиторий не имел новых коммитов в себя от других клиентов, иначе push завершается ошибкой, и придётся делать pull и слияние.

Команда pull — обратна команде push. В случае, если одна и та же ветвь имеет независимую историю в локальной и в удаленной копии, pull немедленно переходит к слиянию.

Слияние в пределах разных файлов осуществляется автоматически (всё это поведение настраивается), а в пределах одного файла — стандартным двухпанельным сравнением файлов. После слияния нужно объявить конфликты как разрешенные.

Результатом всего этого является новое состояние в локальных файлах у того разработчика, что осуществил слияние. Ему нужно немедленно сделать коммит, при этом в данном объекте коммита в репозитории окажется информация о том, что коммит есть результат слияния двух ветвей и имеет два родительских коммита.

Также Git имеет временный локальный индекс файлов. Это — промежуточное хранилище между собственно файлами и очередным коммитом (коммит делается только из этого индекса). С помощью этого индекса осуществляется добавление новых файлов (git add добавляет их в индекс, они попадут в следующий коммит), а также коммит не всех измененных файлов (коммит делается только тем файлам, которым был сделан git add). После git add можно редактировать файл далее, получатся три копии одного и того же файла — последняя, в индексе (та, что была на момент git add), и в последнем коммите.

Имя ветви по умолчанию: master. Имя удалённого репозитория по умолчанию, создаваемое git clone во время типичной операции «взять имеющийся проект с сервера себе на машину»: origin.

Таким образом, в локальном репозитории всегда есть ветвь master, которая есть последний локальный коммит, и ветвь origin/master, которая есть последнее состояние удаленного репозитория на момент завершения и исполнения последней команды pull или push.

Команда fetch (частичный pull) — берёт с удалённого сервера все изменения в origin/master, и переписывает их в локальный репозиторий, продвигая метку origin/master.

GitHub — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.

* Прямо на сайте можно просмотреть файлы проектов с подсветкой синтаксиса для большинства языков программирования.
* Можно создавать приватные репозитории, которые будут видны только вам и выбранным вами людям. Раньше возможность создавать приватные репозитории была платной.
* Есть возможность прямого добавления новых файлов в свой репозиторий через веб-интерфейс сервиса.
* Код проектов можно не только скопировать через Git, но и скачать в виде обычных архивов с сайта.
* Кроме Git, сервис поддерживает получение и редактирование кода через SVN и Mercurial.
  + 1. **SQLite**

SQLite - это встроенная библиотека, которая реализует автономный, безсерверный, нулевой конфигурации, транзакционный механизм СУБД SQL. Это база данных, которая настроена на нуль, что означает, как и другие базы данных, которые вам не нужно настраивать в вашей системе.

SQLite не является автономным процессом, как другие базы данных, вы можете связать его статически или динамически в соответствии с вашим требованием с вашим приложением. SQLite напрямую обращается к своим файлам хранения.

*Особенности SQLite*

* SQLite не требует отдельного процесса сервера или системы для работы (без сервера).
* SQLite поставляется с нулевой конфигурацией, что означает отсутствие необходимости в настройке или администрировании.
* Полная база данных SQLite хранится в одном кросс-платформенном диске.
* SQLite очень маленький и легкий, менее 400KiB полностью сконфигурированный или менее 250KiB с дополнительными функциями, опущенными.
* SQLite является автономным, что означает отсутствие внешних зависимостей.
* SQLite-транзакции полностью совместимы с ACID, обеспечивая безопасный доступ к нескольким процессам или потокам.
* SQLite поддерживает большинство функций языка запросов, найденных в стандарте SQL92 (SQL2).
* SQLite написан на ANSI-C и предоставляет простой и простой в использовании API.
* SQLite доступен в UNIX (Linux, Mac OS-X, Android, iOS) и Windows (Win32, WinCE, WinRT).

Стандартные команды SQLite для взаимодействия с реляционными базами данных аналогичны SQL. Это CREATE, SELECT, INSERT, UPDATE, DELETE и DROP.

* 1. **Информация о базе данных**

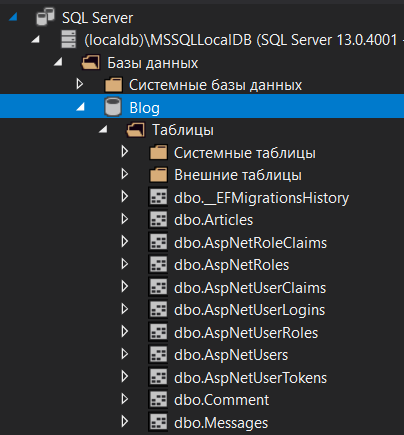


Рисунок 1 - список таблиц базы данных

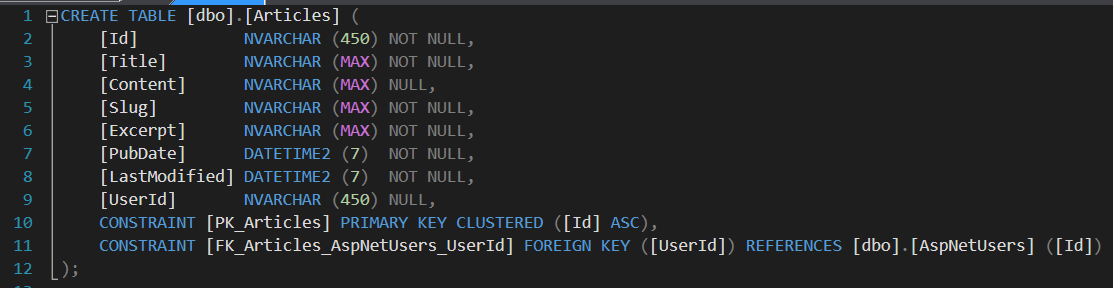


Рисунок 2 - таблица постов

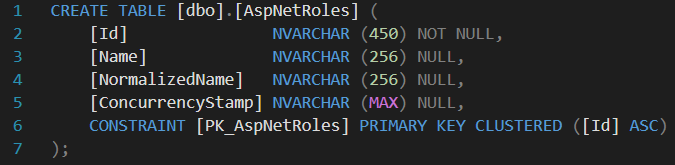


Рисунок 3 - таблица ролей

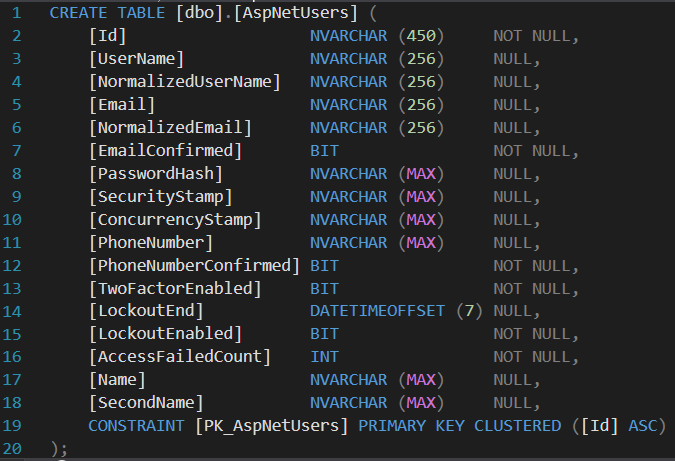


Рисунок 4 - таблица пользователей

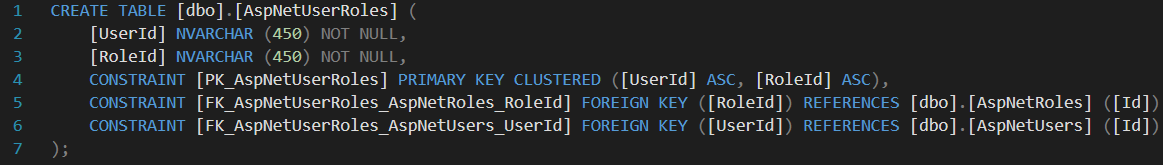


Рисунок 5 - таблица ролей пользователей

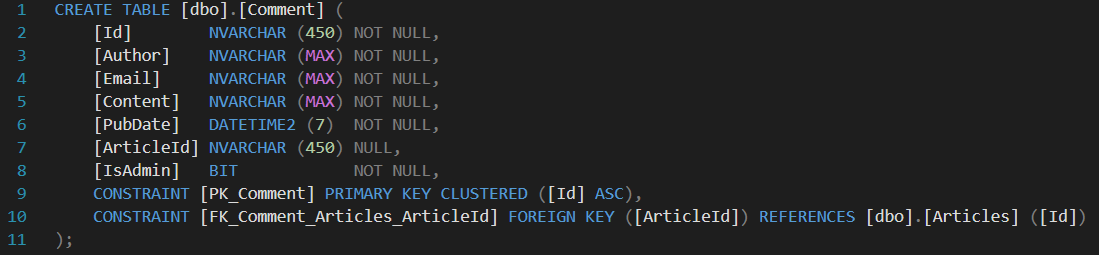


Рисунок 6 - таблица комментариев

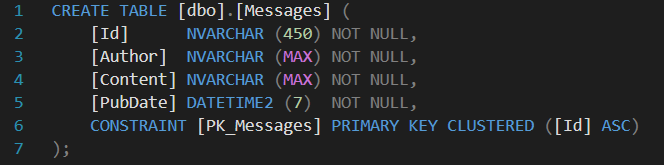


Рисунок 7 - таблица сообщений

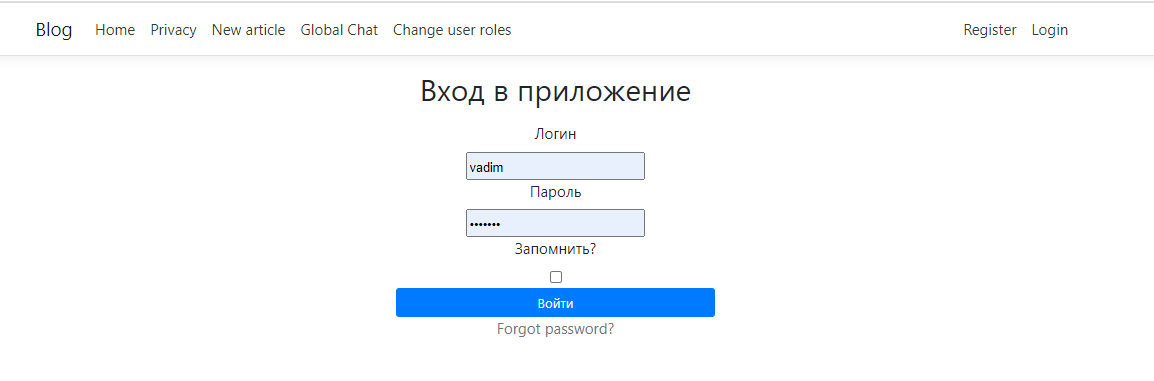
1. **Реализация приложения**

Рисунок 8 - вход в аккаунт

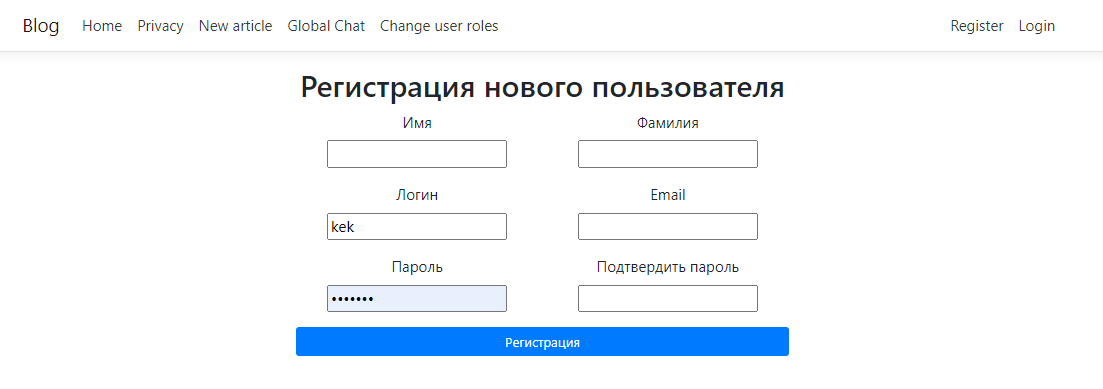


Рисунок 9 - регистрация нового пользователя

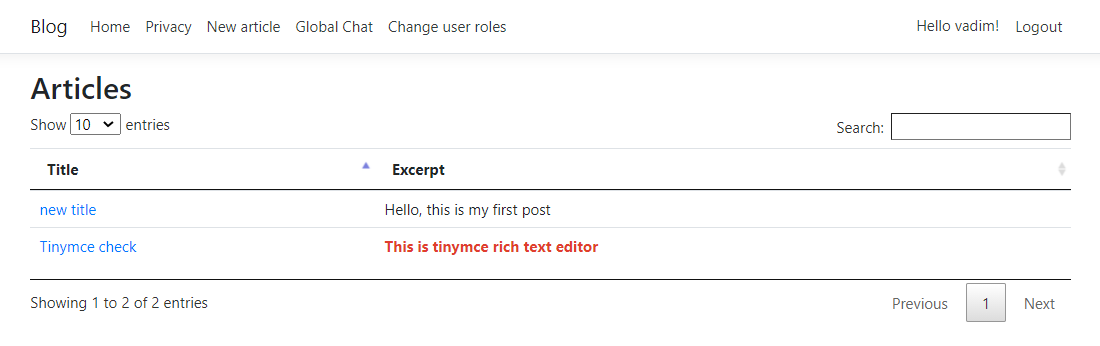


Рисунок 10 - список статей (постов)

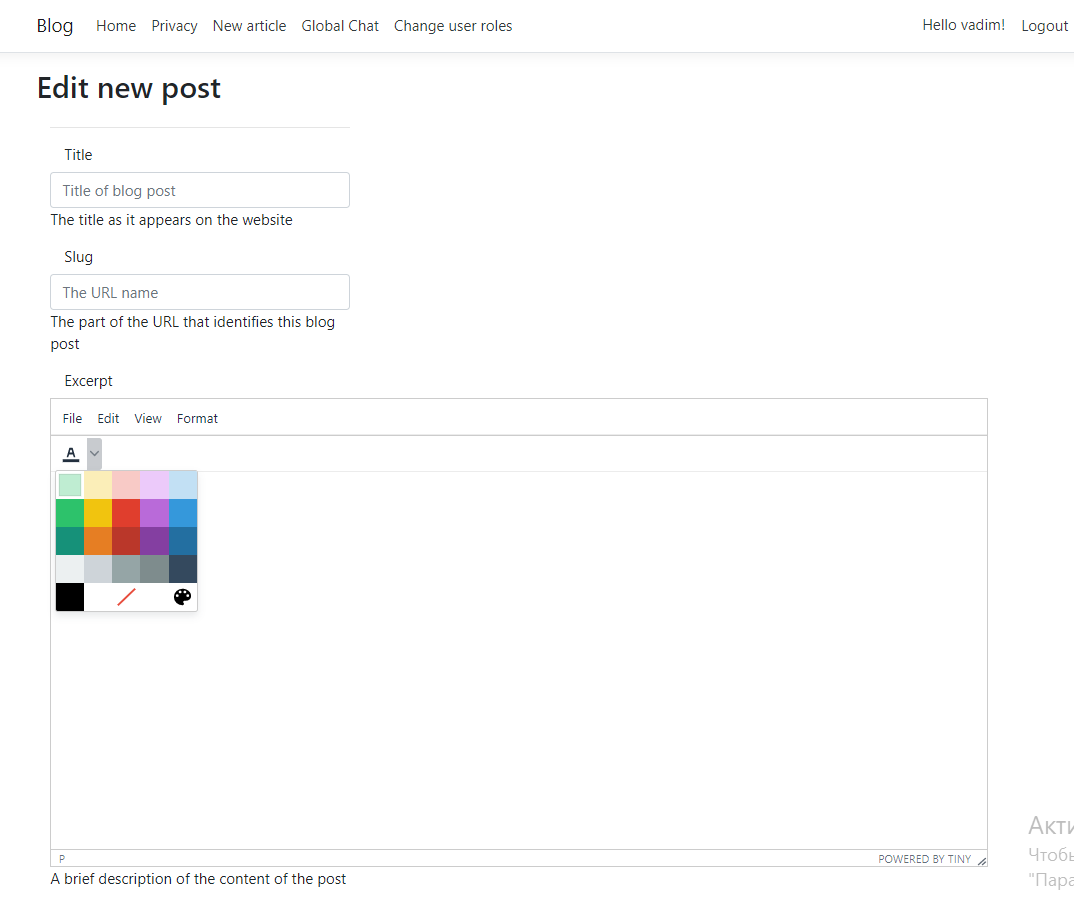


Рисунок 11- редактирование/создание нового поста

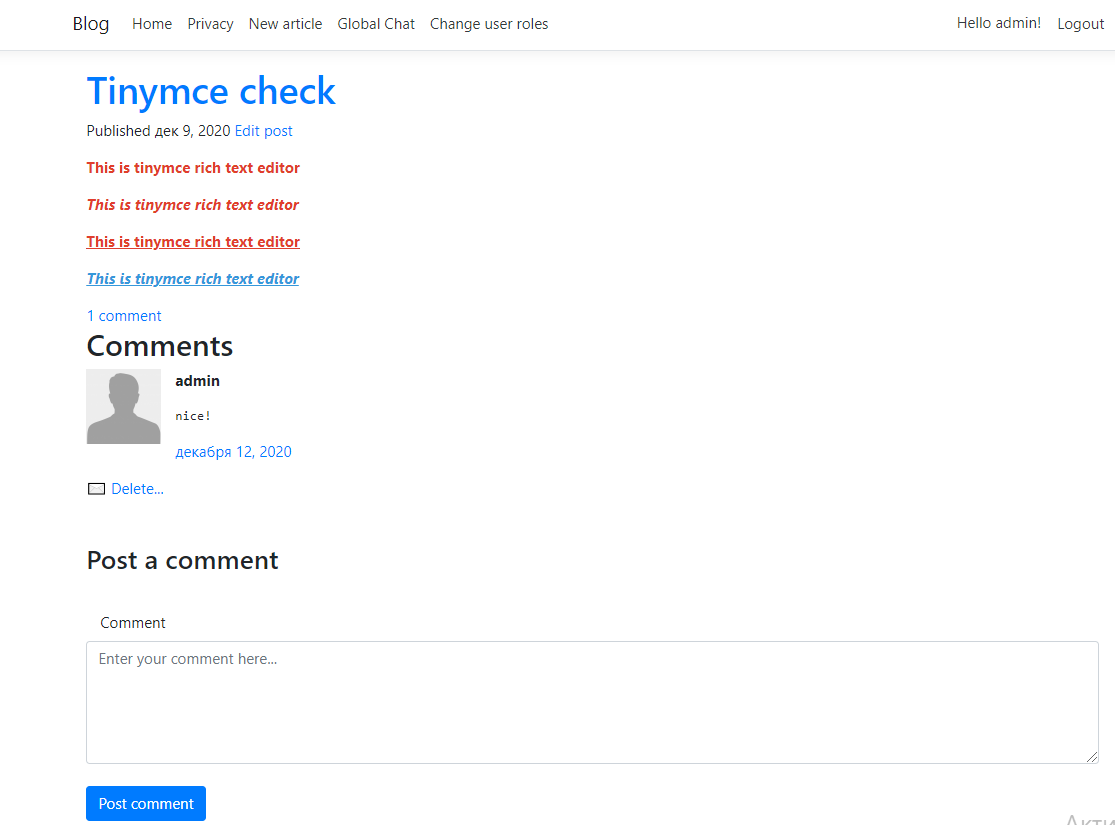


Рисунок 12 - просмотр поста/статьи

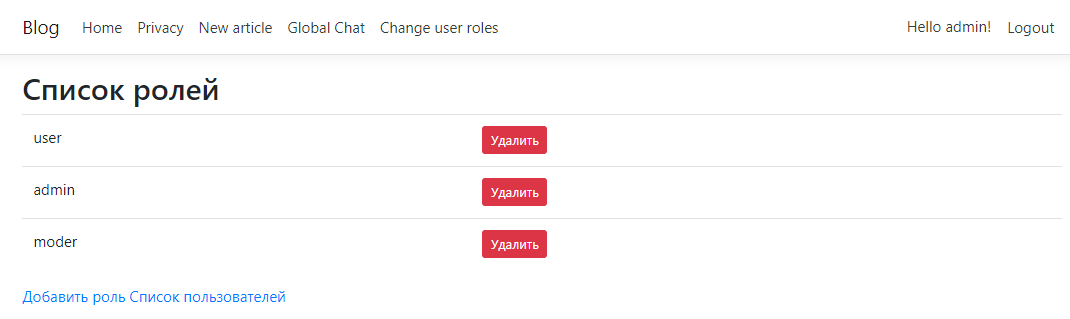


Рисунок 13 - список ролей

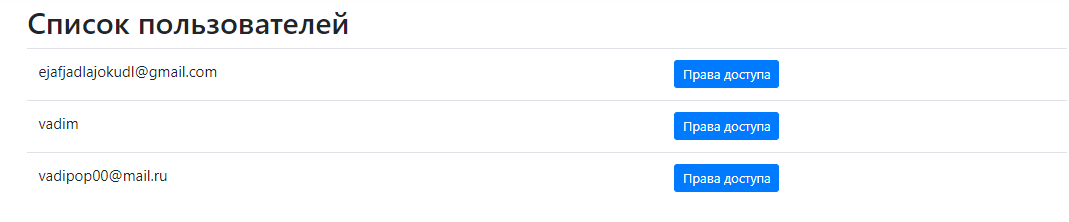


Рисунок 14 - список пользователей

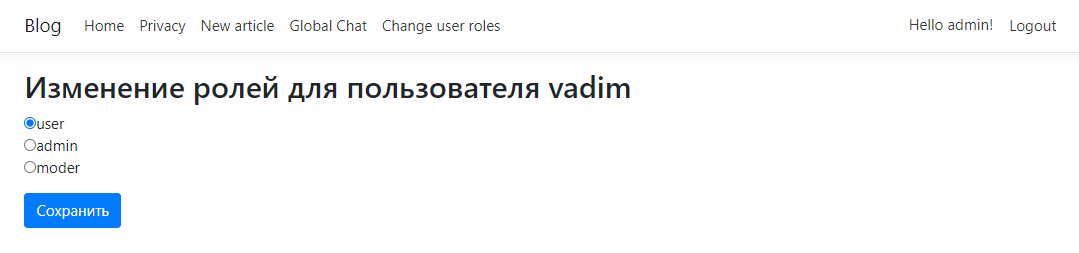


Рисунок 15 - изменение роли пользователя

# **Заключение**

В ходе курсового проекта был разработан блогхост, позволяющий пользователям делиться постами, комментировать их, общаться в живом чате.

# **Список используемых источников**

ASP.NET Core documentation - <https://metanit.com/sharp/aspnet5/29.1.php>

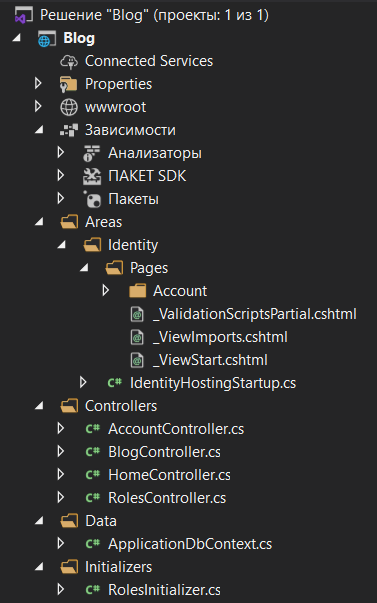
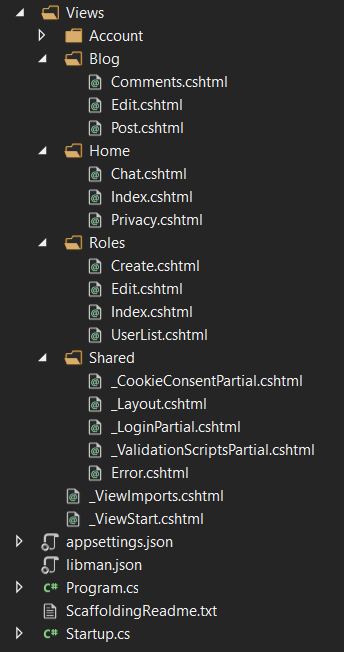
RazorPages documentation - <https://www.learnrazorpages.com/>

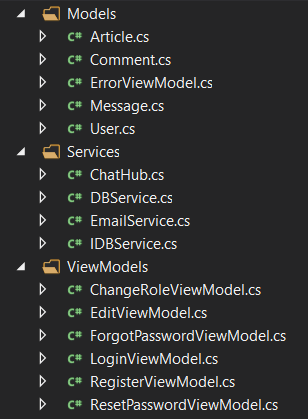
TinyMce documentation - <https://www.tiny.cloud/blog/bootstrap-wysiwyg-editor/>

Entity Framework documentation - <https://www.learnentityframeworkcore.com/>

# **Приложение 1. Код программы**

Код и структура проекта - <https://github.com/vdmppv/bloghost-bsuir>





*Рисунки 16 – 18 – структура проекта*

**Contollers/AccountController.cs**

using System.Collections.Generic;

using System.Linq;

using System.Security.Claims;

using System.Threading.Tasks;

using Blog.Models;

using Blog.ViewModels;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Configuration;

using Microsoft.AspNetCore.Authentication;

using Microsoft.AspNetCore.Authentication.Cookies;

using Blog.Data;

namespace Blog.Controllers

{

public class AccountController : Controller

{

private readonly UserManager<User> \_userManager;

private readonly SignInManager<User> \_signInManager;

private readonly RoleManager<IdentityRole> \_roleManager;

public IConfiguration \_config;

private readonly ApplicationDbContext \_ctx;

public AccountController(UserManager<User> userManager, SignInManager<User> signInManager, RoleManager<IdentityRole> roleManager, IConfiguration configuration, ApplicationDbContext ctx)

{

\_userManager = userManager;

\_signInManager = signInManager;

\_roleManager = roleManager;

\_config = configuration;

\_ctx = ctx;

}

[HttpGet]

public IActionResult Register()

{

return View();

}

[HttpPost]

public async Task<IActionResult> Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

User user = new User

{

Email = model.Email,

UserName = model.UserName,

Name = model.Name,

SecondName = model.SecondName

};

// добавляем пользователя

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

if (result.Succeeded)

{

User registeredUser = \_ctx.Users.FirstOrDefault(c => c.UserName == model.UserName);

await \_userManager.AddToRoleAsync(registeredUser, "user");

}

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 });

}

[HttpGet]

[AllowAnonymous]

public async Task<IActionResult> ConfirmEmail(string userId, string code)

{

if (userId == null || code == null)

{

return View("Error");

}

var user = await \_userManager.FindByIdAsync(userId);

if (user == null)

{

return View("Error");

}

var result = await \_userManager.ConfirmEmailAsync(user, code);

if (result.Succeeded)

{

await \_signInManager.SignInAsync(user, false);

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

}

else

return View("Error");

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Login(LoginViewModel model)

{

if (ModelState.IsValid)

{

var hash = model.Password.GetHashCode();

var user = await \_userManager.FindByNameAsync(model.UserName);

var isPasswordValid = user.PasswordHash;

PasswordVerificationResult result = \_userManager.PasswordHasher.VerifyHashedPassword(user, user.PasswordHash, model.Password);

if (user == null)

{

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

}

else

{

//await Authenticate(user);

await \_signInManager.PasswordSignInAsync(model.UserName, model.Password, model.RememberMe, false);

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

}

}

return View(model);

}

[HttpGet]

[AllowAnonymous]

public IActionResult ForgotPassword()

{

return View();

}

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<IActionResult> ForgotPassword(ForgotPasswordViewModel model)

{

if (ModelState.IsValid)

{

var user = await \_userManager.FindByNameAsync(model.Email);

if (user == null || !(await \_userManager.IsEmailConfirmedAsync(user)))

{

// пользователь с данным email может отсутствовать в бд

// тем не менее мы выводим стандартное сообщение, чтобы скрыть

// наличие или отсутствие пользователя в бд

return View("ForgotPasswordConfirmation");

}

}

return View(model);

}

[HttpGet]

[AllowAnonymous]

public IActionResult ResetPassword(string code = null)

{

return code == null ? View("Error") : View();

}

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<IActionResult> ResetPassword(ResetPasswordViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

var user = await \_userManager.FindByNameAsync(model.Email);

if (user == null)

{

return View("ResetPasswordConfirmation");

}

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

if (result.Succeeded)

{

return View("ResetPasswordConfirmation");

}

foreach (var error in result.Errors)

{

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

}

return View(model);

}

[AllowAnonymous]

public async Task<IActionResult> Logout()

{

// удаляем аутентификационные куки

await \_signInManager.SignOutAsync();

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

}

private async Task Authenticate(User user)

{

string roleId =\_ctx.UserRoles.FirstOrDefault(x => x.UserId == user.Id).RoleId;

var role = await \_roleManager.FindByIdAsync(roleId);

var claims = new List<Claim>

{

new Claim(ClaimsIdentity.DefaultNameClaimType, user.UserName),

new Claim(ClaimsIdentity.DefaultRoleClaimType, role.Name)

};

ClaimsIdentity id = new ClaimsIdentity(claims, "ApplicationCookie", ClaimsIdentity.DefaultNameClaimType, ClaimsIdentity.DefaultRoleClaimType);

await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new ClaimsPrincipal(id));

}

}

}

**Contollers/BlogController.cs**

using System;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Blog.Models;

using Blog.Services;

using Microsoft.Extensions.Configuration;

namespace Blog.Controllers

{

public class BlogController : Controller

{

//private readonly IBlogService \_blog;

private readonly IDBService \_repo;

private readonly IConfiguration \_config;

private readonly EmailService \_emailService;

public BlogController(IDBService repo, IConfiguration config, EmailService emailService)

{

\_repo = repo;

\_config = config;

\_emailService = emailService;

}

[HttpGet]

public IActionResult Index()

{

var articles = \_repo.GetAllPosts();

return View(articles);

}

[Route("/get")]

[HttpGet]

public IActionResult GetPosts()

{

var articles = \_repo.GetAllPosts();

return Ok(articles);

}

[Route("/blog/edit/")]

[Authorize(Roles = "admin, moder")]

public ActionResult New()

{

return View("Edit", new Article());

}

[Route("/blog/{slug?}")]

[Authorize(Roles = "admin, moder"), HttpPost, ValidateAntiForgeryToken]

public async Task<IActionResult> Save(Article article)

{

if (!ModelState.IsValid)

{

return View("Edit", new Article());

}

var existing = \_repo.GetPost(article.Id);

if (existing == null)

{

\_repo.AddPost(article);

// BackgroundJob.Enqueue(() => emailService.SendEmailAsync(\_config["user:email"], "You successfully added a post to your blog", "Article: " + article.Excerpt, \_config));

}

else

{

existing.Title = article.Title.Trim();

existing.Slug = !string.IsNullOrWhiteSpace(article.Slug) ? article.Slug.Trim() : Article.CreateSlug(article.Title);

existing.Content = article.Content.Trim();

existing.Excerpt = article.Excerpt.Trim();

\_repo.UpdatePost(existing);

}

await \_repo.SaveChangesAsync();

return Redirect(article.GetLink());

}

[Route("/blog/edit/{id?}")]

[HttpGet, Authorize(Roles = "admin, moder")]

public IActionResult Edit(string id)

{

if (string.IsNullOrEmpty(id))

{

return View(new Article());

}

var article = \_repo.GetPost(id);

if (article != null)

{

\_repo.SaveChangesAsync();

return View(article);

}

return NotFound();

}

[Route("/blog/{slug?}")]

public IActionResult Details(string slug)

{

var article = \_repo.GetPostBySlug(slug);

if (article == null)

return NotFound();

return View("Post", article);

}

[Route("/blog/deletepost/{id}")]

[HttpPost, Authorize(Roles = "admin, moder"), AutoValidateAntiforgeryToken]

public IActionResult DeletePost(string id)

{

var existing = \_repo.GetPost(id);

if (existing != null)

{

\_repo.RemovePost(existing.Id);

\_repo.SaveChangesAsync();

return Redirect("/");

}

return NotFound();

}

[Route("/blog/comment/{articleId}")]

[HttpPost]

public async Task<IActionResult> AddComment(string articleId, Comment comment)

{

var article = \_repo.GetPost(articleId);

if (!ModelState.IsValid)

{

return View("Post", new Article());

}

if (article == null)

{

return NotFound();

}

comment.ArticleId = articleId;

comment.IsAdmin = User.Identity.IsAuthenticated;

comment.Content = comment.Content.Trim();

if (User.Identity.IsAuthenticated)

{

if (User.Identity.Name.Trim() == "admin")

{

comment.Author = "Vadim Popov".Trim();

comment.Email = "vadim.popov00@bk.ru".Trim();

}

comment.Author = User.Identity.Name.Trim();

comment.Email = "vadasdfoasdfov@bk.ru".Trim();

}

else

{

comment.Author = comment.Author.Trim();

comment.Email = comment.Email.Trim();

}

await \_emailService.SendEmailAsync(\_config["user:email"], "New comment from: " + comment.Author,

"Comment: " + comment.Content + '\n' + "Article: " + articleId);

article.Comments.Add(comment);

\_repo.UpdatePost(article);

await \_repo.SaveChangesAsync();

return Redirect(article.GetLink() + "#" + comment.Id);

}

[Route("/blog/comment/{articleId}/{commentId}")]

[Authorize(Roles = "admin, moder")]

public IActionResult DeleteComment(string articleId, string commentId)

{

var article = \_repo.GetPost(articleId);

if (article == null)

{

return NotFound();

}

var comment = article.Comments.FirstOrDefault(c => c.Id.Equals(commentId, StringComparison.OrdinalIgnoreCase));

if (comment == null)

{

return NotFound();

}

article.Comments.Remove(comment);

\_repo.UpdatePost(article);

\_repo.SaveChangesAsync();

return Redirect(article.GetLink() + "#comments");

}

}

}

**Controllers/HomeController.cs**

using System.Diagnostics;

using Microsoft.AspNetCore.Mvc;

using Blog.Models;

using Microsoft.Extensions.Logging;

using Microsoft.AspNetCore.Authorization;

namespace Blog.Controllers

{

public class HomeController : Controller

{

private readonly ILogger \_logger;

public HomeController(ILogger<HomeController> logger)

{

\_logger = logger;

}

[Authorize]

public IActionResult Index()

{

\_logger.LogInformation("Log message on Index() method");

return View();

}

public IActionResult Privacy()

{

\_logger.LogInformation("Log message on Privacy() method");

return View();

}

public IActionResult Chat()

{

\_logger.LogInformation("Log message on Chat() method");

return View();

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

\_logger.LogError("Log message on Error() method");

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

}

**Controllers/RolesController.cs**

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Identity;

using Blog.Models;

using Blog.ViewModels;

using Microsoft.AspNetCore.Authorization;

namespace Blog.Controllers

{

public class RolesController : Controller

{

RoleManager<IdentityRole> \_roleManager;

UserManager<User> \_userManager;

public RolesController(RoleManager<IdentityRole> roleManager, UserManager<User> userManager)

{

\_roleManager = roleManager;

\_userManager = userManager;

}

[Authorize(Roles = "admin")]

public IActionResult Index() => View(\_roleManager.Roles.ToList());

public IActionResult Create() => View();

[HttpPost]

public async Task<IActionResult> Create(string name)

{

if (!string.IsNullOrEmpty(name))

{

IdentityResult result = await \_roleManager.CreateAsync(new IdentityRole(name));

if (result.Succeeded)

{

return RedirectToAction("Index");

}

else

{

foreach (var error in result.Errors)

{

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

}

}

}

return View(name);

}

[HttpPost]

public async Task<IActionResult> Delete(string id)

{

IdentityRole role = await \_roleManager.FindByIdAsync(id);

if (role != null)

{

IdentityResult result = await \_roleManager.DeleteAsync(role);

}

return RedirectToAction("Index");

}

public IActionResult UserList() => View(\_userManager.Users.ToList());

public async Task<IActionResult> Edit(string userId)

{

// получаем пользователя

User user = await \_userManager.FindByIdAsync(userId);

if (user != null)

{

// получем список ролей пользователя

var userRoles = await \_userManager.GetRolesAsync(user);

var allRoles = \_roleManager.Roles.ToList();

ChangeRoleViewModel model = new ChangeRoleViewModel

{

UserId = user.Id,

UserEmail = user.Email,

UserRoles = userRoles,

AllRoles = allRoles

};

return View(model);

}

return NotFound();

}

[HttpPost]

public async Task<IActionResult> Edit(string userId, List<string> roles)

{

// получаем пользователя

User user = await \_userManager.FindByIdAsync(userId);

if (user != null)

{

// получем список ролей пользователя

var userRoles = await \_userManager.GetRolesAsync(user);

// получаем все роли

var allRoles = \_roleManager.Roles.ToList();

// получаем список ролей, которые были добавлены

var addedRoles = roles.Except(userRoles);

// получаем роли, которые были удалены

var removedRoles = userRoles.Except(roles);

await \_userManager.RemoveFromRolesAsync(user, removedRoles);

await \_userManager.AddToRolesAsync(user, addedRoles);

return RedirectToAction("UserList");

}

return NotFound();

}

}

}

**Startup.cs**

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Blog.Data;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Blog.Services;

using Microsoft.Extensions.Logging;

using Blog.Models;

namespace Blog

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(

Configuration.GetConnectionString("DefaultConnection")));

services.AddIdentity<User, IdentityRole>(opts =>

{

opts.User.RequireUniqueEmail = false;

})

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddRoles<IdentityRole>()

.AddDefaultTokenProviders();

services.AddAuthentication().AddCookie(options =>

{

options.LoginPath = new Microsoft.AspNetCore.Http.PathString("/Account/Login");

options.AccessDeniedPath = new Microsoft.AspNetCore.Http.PathString("/Account/Login");

});

services.AddTransient<EmailService>();

services.AddScoped<IDBService, DBService>();

services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();

//services.AddScoped<IBlogService, FileBlogService>();

services.Configure<CookiePolicyOptions>(options =>

{

// This lambda determines whether user consent for non-essential cookies is needed for a given request.

options.CheckConsentNeeded = context => true;

options.MinimumSameSitePolicy = SameSiteMode.None;

});

services.AddSignalR();

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_2);

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)

{

loggerFactory.AddDebug();

loggerFactory.AddConsole();

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

app.UseDatabaseErrorPage();

}

else

{

app.UseExceptionHandler("/Home/Error");

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseStatusCodePages();

app.UseStatusCodePagesWithReExecute("/error", "?code={0}");

app.Map("/error", ap => ap.Run(async context =>

{

await context.Response.WriteAsync($"Err: {context.Request.Query["code"]}");

}));

loggerFactory.AddFile("Logs/mylog-{Date}.txt");

app.UseSignalR(routes => { routes.MapHub<ChatHub>("/Home/Chatter"); });

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseCookiePolicy();

app.UseAuthentication();

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

template: "{controller=Home}/{action=Index}/{id?}");

});

}

}

}

**Services/DBService.cs**

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using Blog.Models;

using Blog.Data;

namespace Blog.Services

{

public class DBService : IDBService

{

private ApplicationDbContext \_ctx;

public DBService(ApplicationDbContext ctx)

{

\_ctx = ctx;

}

public void AddPost(Article article)

{

\_ctx.Articles.Add(article);

}

public List<Article> GetAllPosts()

{

return \_ctx.Articles.ToList();

}

public List<Message> GetAllMessages()

{

return \_ctx.Messages.ToList();

}

public Article GetPost(string id)

{

return \_ctx.Articles

.Include(p => p.Comments)

.FirstOrDefault(p => p.Id == id);

}

public Article GetPostBySlug(string slug)

{

return \_ctx.Articles

.Include(p => p.Comments)

.FirstOrDefault(p => p.Slug == slug);

}

public void RemovePost(string id)

{

\_ctx.Articles.Remove(GetPost(id));

}

public void UpdatePost(Article article)

{

\_ctx.Articles.Update(article);

}

public async Task<bool> SaveChangesAsync()

{

if (await \_ctx.SaveChangesAsync() > 0)

{

return true;

}

return false;

}

}

}

**Models/Article.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace Blog.Models

{

public class Article

{

[Required, Key]

public string Id { get; set; } = Guid.NewGuid().ToString();

[Required]

public string Title { get; set; }

public string Content { get; set; }

[Required]

public string Slug { get; set; }

[Required]

public string Excerpt { get; set; }

public DateTime PubDate { get; set; } = DateTime.UtcNow.ToLocalTime();

public DateTime LastModified { get; set; } = DateTime.UtcNow.ToLocalTime();

public IList<Comment> Comments { get; } = new List<Comment>();

public virtual User User { get; set; }

public string GetLink()

{

return $"/blog/{Slug}/";

}

public bool AreCommentsOpen(int commentsCloseAfterDays)

{

return PubDate.AddDays(commentsCloseAfterDays) >= DateTime.UtcNow.ToLocalTime();

}

public static string CreateSlug(string title)

{

title = title.ToLowerInvariant().Replace(" ", "-");

title = RemoveReservedUrlCharacters(title);

return title.ToLowerInvariant();

}

static string RemoveReservedUrlCharacters(string text)

{

var reservedCharacters = new List<string> { "!", "#", "$", "&", "'", "(", ")", "\*", ",", "/", ":", ";", "=", "?", "@", "[", "]", "\"", "%", ".", "<", ">", "\\", "^", "\_", "'", "{", "}", "|", "~", "`", "+" };

foreach (var chr in reservedCharacters)

{

text = text.Replace(chr, "");

}

return text;

}

public string RenderContent()

{

var result = Content;

return result;

}

}

}

**ViewModels/RegisterViewModel.cs**

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace Blog.ViewModels

{

public class RegisterViewModel

{

[Required]

[Display(Name = "Имя")]

public string Name { get; set; }

[Required]

[Display(Name = "Фамилия")]

public string SecondName { get; set; }

[Required]

[Display(Name = "Логин")]

public string UserName { get; set; }

[Required]

[Display(Name = "Email")]

public string Email { get; set; }

[Required]

[DataType(DataType.Password)]

[Display(Name = "Пароль")]

public string Password { get; set; }

[Required]

[Compare("Password", ErrorMessage = "Пароли не совпадают")]

[DataType(DataType.Password)]

[Display(Name = "Подтвердить пароль")]

public string PasswordConfirm { get; set; }

}

}

**Views/Blog/Edit.cshtml**

@model Article

@{

bool isNew = string.IsNullOrEmpty(Model.Title);

ViewData["Title"] = "Edit " + (Model.Title ?? "new post");

}

<link rel="stylesheet" href="~/css/admin.scss" />

<h2>@ViewData["Title"]</h2>

<div class="row">

<div class="col-md-4">

<section>

<form method="post" id="edit" class="container" **asp-controller**="Blog" **asp-action**="Save">

<hr />

<div **asp-validation-summary**="All" class="text-danger"></div>

<input **type**="hidden" **asp-for**="@Model.Id" />

<div class="form-group">

<label **asp-for**="Title" class="col-lg-1 control-label">Title</label>

<input **asp-for**="Title" required placeholder="Title of blog post" aria-describedby="desc\_title" class="form-control" />

<span **asp-validation-for**="Title" class="help-block">The title as it appears on the website</span>

</div>

<div class="form-group">

<label **asp-for**="Slug" class="col-lg-1 control-label">Slug</label>

<input **asp-for**="Slug" placeholder="The URL name" aria-describedby="desc\_slug" class="form-control" />

<span **asp-validation-for**="Slug" class="help-block">The part of the URL that identifies this blog post</span>

</div>

<div class="form-group">

<label for="textArea" class="col-lg-1 control-label">Excerpt</label>

<div class="col-lg-15">

<textarea **asp-for**="Excerpt" class="form-control" rows="5" id="textArea">Excerpt</textarea>

<span class="help-block">A brief description of the content of the post</span>

</div>

</div>

<br />

<div class="form-group">

<label for="textArea" class="col-lg-1 control-label">Post</label>

<textarea **asp-for**="Content" input type="text" class="form-control">@Model.Content</textarea>

<span class="help-block">A full post content</span>

</div>

<br />

<button type="submit" class="btn btn-success" title="Save the post">Save</button>

@if (!isNew)

{

<button type="submit" class="delete btn btn-danger" title="Delete the post..." **asp-action**="DeletePost" **asp-route-id**="Id">Delete</button>

}

</form>

</section>

</div>

<script src="~/js/admin.js" async></script>

</div>

@section Scripts {

<script type="text/javascript" src="https://cdn.tiny.cloud/1/no-api-key/tinymce/5/tinymce.min.js" referrerpolicy="origin"></script>

<script type="text/javascript">

tinymce.init({

selector: 'textarea', width: 1000, height: 500,

toolbar: "forecolor"

});

</script>

<script src="~/js/admin.js" async></script>

}