

Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

ПУБЛИЧНЫЙ БЛОГ
БГУИР КП 1-40 04 01 324 ПЗ

Студент
Руководитель

Дорофеев В. И.
Удовин И. А.

Минск 2020

Оглавление

Вступление	3
1. Анализ аналогов и прототипов	4
1.1. Хабр	4
1.2. Пикабу	5
1.3. Reddit	6
2. Предназначение и формулировка целей	7
3. Используемые технологии.....	8
3.1. ASP.NET Core	9
3.2. Entity Framework	10
3.3. React	12
3.4. CSS	14
3.5. xUnit	15
3.6. Git	17
4. Логическая модель данных приложения.....	18
5. Функциональная модель приложения.....	20
6. Структура приложения	21
7. Демонстрация использования разработанного программного средства	25
Заключение.....	37
Список использованных источников.....	38
Приложение 1. Исходный код основных файлов проекта	39

Вступление

В качестве темы своего курсового проекта я выбрал создание публичного текстового блока.

Блог – это онлайн-ресурс с регулярно обновляемым контентом (текстовые записи, изображения, мультимедиа). Пользователя, который этот контент публикует, называют блогером.

Большую популярность блоги получили с развитием интернет-технологий, когда у каждого человека появился свободный доступ к интернету. Сегодня это не просто возможность делиться своими мыслями с другими людьми, это возможность зарабатывать на любимом деле и обрести большую популярность.

Какие бывают форматы:

- текстовые блоги;
- микроблоги (немного текста + картинка или видео);
- фотоблоги;
- видеоблоги;
- стриминг;
- ведение странички в социальных сетях;
- создание собственного сайта.

К текстовому формату блогов, которым является и курсовой проект, относятся различные площадки, где любой автор может опубликовать свою статью. Наиболее известными являются «Живой Журнал», «Яндекс.Дзен», «Пикабу», «Хабрахабр».

Такие ресурсы позволяют вести собственные блоги без надобности создания вебсайта, профиля в соцсетях и т. д. Все, что требуется от автора, – написать качественный и интересный текст.

1. Анализ аналогов и прототипов

1.1. Хабр

Хабр (бывший Хабрахабр) — русскоязычный веб-сайт в формате системы тематических коллективных блогов (именуемых хабами) с элементами новостного сайта, созданный для публикации новостей, аналитических статей, мыслей, связанных с информационными технологиями, бизнесом и интернетом. Основан Денисом Крючковым в июне 2006 года.

Контент сайта формируется пользователями-добровольцами, которые пишут в коллективные и персональные блоги, публикуют подкасты, переводят иностранные статьи, проводят опросы (голосования) и общаются с другими пользователями. С 2011 года на Хабре действует ППА — Программа Поощрения Авторов, в рамках которой владельцы портала выплачивают вознаграждение авторам хороших публикаций.

В рейтинге «Медиалогии» среди тематических изданий (IT и телеком СМИ — III квартал 2014) «Хабрахабр» занимал 3-е место. Входил в 1000 самых посещаемых сайтов в мире на март 2015 по рейтингу Alexa Internet.



Рисунок 1.1 - Логотип “Habrahabr”.

1.2. Пикабу

Пикабу́ (Pikabu) — русскоязычное информационно-развлекательное сообщество. Сайт работает исключительно по принципу пользовательского контента, его владельцы не занимаются написанием и продвижением постов. Для добавления и оценки (лайк, дизлайк) постов и комментариев требуется регистрация на сайте, просмотр сайта (кроме постов эротического характера) доступен и для незарегистрированных посетителей. Комментарии на Пикабу представляют собой «бесконечно ветвящиеся» древовидные обсуждения.

Посты и комментарии могут содержать в себе форматированный текст, а также прикрепленные картинки, гифки и видео. Контент как создаётся самими пользователями, так и копируется ими с других сайтов. Значительную часть контента составляют интернет-мемы, некоторые из которых зародились или получили широкую известность непосредственно на Пикабу. Одним из наиболее популярных мемов, появившихся именно благодаря публикации на Пикабу, является Ждун.



Рисунок 1.2 - Логотип “Пикабу”.

1.3. Reddit

Reddit - социальный новостной сайт, на котором зарегистрированные пользователи могут размещать ссылки на какую-либо понравившуюся информацию в интернете. Как и многие другие подобные сайты, Reddit поддерживает систему голосования за понравившиеся сообщения — наиболее популярные из них оказываются на заглавной странице сайта. Один из наиболее популярных сайтов в мире — 19-е место по посещаемости по данным Alexa Internet и SimilarWeb.

54 % пользователей сайта являются мужчинами, 54 % — проживают в США. По состоянию на 2013 год 6 % взрослых пользователей интернета в США являлись пользователями Reddit, при этом большинство пользователей сайта относилось к возрастной группе от 18 до 29 лет.



Рисунок 1.3 - Логотип “Reddit”.

2. Предназначение и формулировка целей

Назначение разработки:

В рамках курсового проекта необходимо создать веб-приложение, представляющую собой публичный текстовый блог.

Целевая аудитория:

Активные интернет-пользователи в возрасте от 18 до 29 лет.

Риски:

Создание данного продукта может быть связано со следующими рисками:

- Риск не учесть особенности правового поля;
- Риск не учесть некоторые аспекты предметной области;
- Риск некорректно рассчитать сроки создания продукта;
- Риск создать неконкурентоспособный продукт.

Перечень основных выполняемых функций:

Приложение должно давать возможность зарегистрироваться, изменить информацию о себе, сменить пароль; создавать, изменять, удалять и просматривать посты; оставлять и удалять комментарии к постам. Также для администрации должны быть доступны функции блокировки пользователей, изменения роли пользователей, а также создание, изменение и удаление категорий.

3. Используемые технологии

Для разработки серверной части приложения был выбран фреймворк ASP.NET Core 3.1. Для работы с данными используется технология Entity Framework 6 - Code-First. Пользовательский интерфейс разработан с помощью React. Для придания интерфейсу аккуратного внешнего вида была использована библиотека стилей Bootstrap, а также CSS для более детальной настройки стилей. Для юнит-тестирования сервисов приложения применяется среда xUnit.NET. Чтобы имитировать объекты зависимостей при проведении юнит-тестов, используется фреймворк Moq. В качестве системы контроля версий был выбран Git. Далее рассмотрим основные технологии из вышеназванных более детально.

3.1. ASP.NET Core

ASP.NET Core — свободно-распространяемый кросс-платформенный фреймворк для создания веб-приложений с открытым исходным кодом. Данная платформа разрабатывается компанией Майкрософт совместно с сообществом и имеет большую производительность по сравнению с ASP.NET. Имеет модульную структуру и совместима с такими операционными системами как Windows, Linux и macOS.

Несмотря на то, что это новый фреймворк, построенный на новом веб-стеке, он обладает высокой степенью совместимости концепций с ASP.NET. Приложения ASP.NET Core поддерживают параллельное управление версиями, при котором разные приложения, работающие на одном компьютере, могут ориентироваться на разные версии ASP.NET Core. Это было невозможно в предыдущих версиях ASP.NET.

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 версиях 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 C#.

ASP.NET Core характеризуется расширяемостью. Фреймворк построен из набора относительно независимых компонентов. Благодаря чему возможно 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.

3.2. Entity Framework

Entity Framework представляет из себя специальную объектно-ориентированную технологию на базе фреймворка .NET для работы с данными. Если традиционные средства ADO.NET позволяют создавать подключения, команды и прочие объекты для взаимодействия с базами данных, то Entity Framework представляет собой более высокий уровень абстракции, который позволяет абстрагироваться от самой базы данных и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, то на концептуальном уровне, который нам предлагает Entity Framework, мы уже работаем с объектами.

Первая версия Entity Framework - 1.0 вышла еще в 2008 году и представляла очень ограниченную функциональность, базовую поддержку ORM (object-relational mapping - отображения данных на реальные объекты) и один единственный подход к взаимодействию с БД - Database First. С выходом версии 4.0 в 2010 году многое изменилось - с этого времени Entity Framework стал рекомендуемой технологией для доступа к данным, а в сам фреймворк были введены новые возможности взаимодействия с БД - подходы Model First и Code First.

Дополнительные улучшения функционала последовали с выходом версии 5.0 в 2012 году. И наконец, в 2013 году был выпущен Entity Framework 6.0, обладающий возможностью асинхронного доступа к данным.

Центральной концепцией Entity Framework является понятие сущности или entity. Сущность представляет набор данных, ассоциированных с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их наборами.

Любая сущность, как и любой объект из реального мира, обладает рядом свойств. Например, если сущность описывает человека, то в ней можно выделить такие свойства, как имя, фамилия, рост, возраст, вес. Свойства необязательно представляют простые данные типа int, но и могут представлять более комплексные структуры данных. И у каждой сущности может быть одно или несколько свойств, которые будут отличать эту сущность от других и будут уникально определять эту сущность. Подобные свойства называют ключами.

При этом сущности могут быть связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим, подобно тому, как в реальной базе данных происходит связь через внешние ключи.

Отличительной чертой Entity Framework является использование запросов LINQ для выборки данных из БД. С помощью LINQ возможно не только извлекать определенные строки, хранящие объекты, из БД, но и получать объекты, связанные различными ассоциативными связями.

Entity Framework предполагает три возможных способа взаимодействия с базой данных:

- **Database first:** Entity Framework создает набор классов, которые отражают модель конкретной базы данных
- **Model first:** сначала разработчик создает модель базы данных, по которой затем Entity Framework создает реальную базу данных на сервере.
- **Code first:** разработчик создает класс модели данных, которые будут храниться в базе данных, а затем Entity Framework по этой модели генерирует базу данных и ее таблицы.

3.3. React

React — JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов.

React разрабатывается и поддерживается Facebook, Instagram и сообществом отдельных разработчиков и корпораций.

React может использоваться для разработки одностраничных и мобильных приложений. Его цель — предоставить высокую скорость, простоту и масштабируемость. В качестве библиотеки для разработки пользовательских интерфейсов React часто используется с другими библиотеками, такими как MobX, Redux и GraphQL.

Особенности:

- Однонаправленная передача данных

Свойства передаются от родительских компонентов к дочерним. Компоненты получают свойства как множество неизменяемых значений, поэтому компонент не может напрямую изменять свойства, но может вызывать изменения через callback-функции. Такой механизм называют «свойства вниз, события вверх».

- Виртуальный DOM

React использует виртуальный DOM. React создаёт кэш-структуру в памяти, что позволяет вычислять разницу между предыдущим и текущим состояниями интерфейса для оптимального обновления DOM браузера. Таким образом программист может работать со страницей, считая, что она обновляется вся, но библиотека самостоятельно решает, какие компоненты страницы необходимо обновить.

- JSX

JavaScript XML (JSX) — это расширение синтаксиса JavaScript, которое позволяет использовать HTML-подобный синтаксис для описания структуры интерфейса. Как правило, компоненты написаны с использованием JSX, но также есть возможность использования обычного JavaScript. JSX напоминает другой язык, созданный в компании Фейсбук для расширения PHP, XHP.

- Методы жизненного цикла

Методы жизненного цикла позволяют разработчику запускать код на разных стадиях жизненного цикла компонента. Например:

- `shouldComponentUpdate` — позволяет предотвратить перерисовку компонента с помощью возврата `false`, если перерисовка не нужна.
- `componentDidMount` — вызывается после первой отрисовки компонента. Часто используется для запуска получения данных с удаленного источника через API.
- `render` — важнейший метод жизненного цикла. Каждый компонент должен иметь этот метод. Обычно вызывается при изменении данных компонента для перерисовки данных в интерфейсе.
- Не только отрисовка HTML в браузере

React используется не только для отрисовки HTML в браузере. Например, Facebook имеет динамические графики, которые отрисовываются в теги `<canvas>`. Netflix и PayPal используют изоморфные загрузки для отрисовки идентичного HTML на сервере и клиенте.
- React Hooks

Хуки позволяют использовать состояние и другие возможности React без написания классов.

Построение пользовательских хуков позволяет помещать логику компонента в повторно используемые функции.

3.4. CSS

CSS (Cascading Style Sheets — каскадные таблицы стилей) — формальный язык описания внешнего вида документа, написанного с использованием языка разметки.

Преимущественно используется как средство описания, оформления внешнего вида веб-страниц, написанных с помощью языков разметки HTML и XHTML, но может также применяться к любым XML-документам, например, к SVG или XUL.

CSS используется создателями веб-страниц для задания цветов, шрифтов, стилей, расположения отдельных блоков и других аспектов представления внешнего вида этих веб-страниц. Основной целью разработки CSS являлось отделение описания логической структуры веб-страницы (которое производится с помощью HTML или других языков разметки) от описания внешнего вида этой веб-страницы (которое теперь производится с помощью формального языка CSS). Такое разделение может увеличить доступность документа, предоставить большую гибкость и возможность управления его представлением, а также уменьшить сложность и повторяемость в структурном содержимом.

Кроме того, CSS позволяет представлять один и тот же документ в различных стилях или методах вывода, таких как экранное представление, печатное представление, чтение голосом (специальным голосовым браузером или программой чтения с экрана), или при выводе устройствами, использующими шрифт Брайля.

3.5. xUnit

xUnit — это собирательное название семейства фреймворков для модульного тестирования, структура и функциональность которых основана на SUnit, предназначавшегося для языка программирования Smalltalk. SUnit, разработанный Кентом Бекон в 1998 году, был написан в высоко структурном объектно-ориентированном стиле, получил широкую популярность и был адаптирован для множества других языков. Названия фреймворков этого семейства образованы аналогично "SUnit", обычно заменяется буква "S" на первую букву (или несколько первых) в названии предполагаемого языка ("JUnit" для Java, "NUnit" для программной платформы .NET и т. д.). Семейство таких фреймворков с общей архитектурой обычно и известно как "xUnit".

Архитектура xUnit

Все фреймворки из семейства xUnit имеют следующие базовые компоненты архитектуры, которые в различных реализациях могут слегка варьироваться.

- Модуль, выполняющий тестирование (Test runner)

Модуль представляет собой исполняемую программу, которая выполняет тесты, реализованные с помощью фреймворка, и отображает информацию о ходе их выполнения.

- Тестовые сценарии (Test cases)

Варианты тестирования (тестовые сценарии/случаи) являются базовыми элементами модульных тестов.

- Конфигурации тестирования (Test fixtures)

Конфигурация тестирования (также называемая контекстом) — это набор предварительно заданных условий или состояний объектов, необходимый для запуска теста. Разработчик должен задать заведомо корректную конфигурацию перед выполнением каждого теста, а затем вернуть оригинальную конфигурацию после завершения теста.

- Наборы тестов (Test suites)

Тестовый набор — это несколько тестов, имеющих общую конфигурацию. Очередность выполнения тестов не должна иметь значения.

- Выполнение тестов (Test execution)

Выполнение каждого теста происходит согласно следующей схеме:

```
setup(); /* Сначала подготавливается 'контекст' тестирования */  
  
...  
  
/* Тело теста - здесь указывается тестовый сценарий */  
  
...  
  
teardown(); /* После прохождения теста (независимо от его  
результата) контекст тестирования "очищается" */
```

- Форматирование результатов тестирования (Test result formatter)

Модуль, выполняющий тестирование, должен вывести результаты в одном или нескольких заданных форматах. В дополнение к обычному тексту, воспринимаемому человеком, часто результаты выводятся в формате XML.

- Утверждения (Assertions)

Утверждение в тесте — это функция или макрос, которая проверяет поведение или состояние тестируемого модуля. Часто утверждением является проверка равенства или неравенства некоторого параметра модуля ожидаемому результату. Неудачное прохождение проверки приводит к провалу всего тестового сценария и (если необходимо) к исключению, которое останавливает сценарий без перехода к следующему утверждению.

3.6. Git

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.

Примерами проектов, использующих Git, являются Ядро Linux, Swift, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, DokuWiki, Qt и некоторые дистрибутивы Linux.

Программа является свободной и выпущена под лицензией GNU GPL 2.

Система спроектирована как набор программ, специально разработанных с учётом их использования в скриптах. Это позволяет удобно создавать специализированные системы контроля версий на базе Git или пользовательские интерфейсы. Например, Cogito является именно таким примером оболочки к репозиториям Git, а StGit использует Git для управления коллекцией исправлений (патчей).

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.

Удалённый доступ к репозиториям Git обеспечивается git-daemon, SSH-или HTTP-сервером. TCP-сервис git-daemon входит в дистрибутив Git и является наряду с SSH наиболее распространённым и надёжным методом доступа. Метод доступа по HTTP, несмотря на ряд ограничений, очень популярен в контролируемых сетях, потому что позволяет использовать существующие конфигурации сетевых фильтров.

4. Логическая модель данных приложения

Проведённый анализ предметной области позволил выявить следующий набор основных сущностей и их атрибуты:

- Пользователь (User):
 - Id - идентификатор;
 - UserName - имя пользователя;
 - Login - логин;
 - Password - пароль;
 - RoleId - идентификатор роли;
 - UnblockTime - время разблокировки;
 - Email - адрес электронной почты;
 - RegistrationDate - дата регистрации;
 - HasPhoto - флаг наличия фото;
 - IsDeleted - флаг удаления.
- Роль (Role):
 - Id - идентификатор;
 - Name - название.
- Пост (Post):
 - Id - идентификатор;
 - Name - название;
 - CategoryId - идентификатор категории;
 - ShortDescription - краткое описание;
 - Description - текст поста;
 - AuthorId - идентификатор автора;
 - PublicationTime - время публикации;
 - HasImage - флаг наличия изображения;
 - IsDeleted - флаг удаления.
- Категория (Category):
 - Id - идентификатор;
 - Name - название;
 - IsDeleted - флаг удаления.
- Комментарий (Comment):
 - Id - идентификатор;
 - Value - текст комментария.

- Тег (Tag):
 - Id - идентификатор;
 - Name - название.
- Пост-Тег (PostTag):
 - PostId - идентификатор поста;
 - TagId - идентификатор тега.

ER-диаграмма для описанных выше сущностей продемонстрирована на рисунке 4.1.

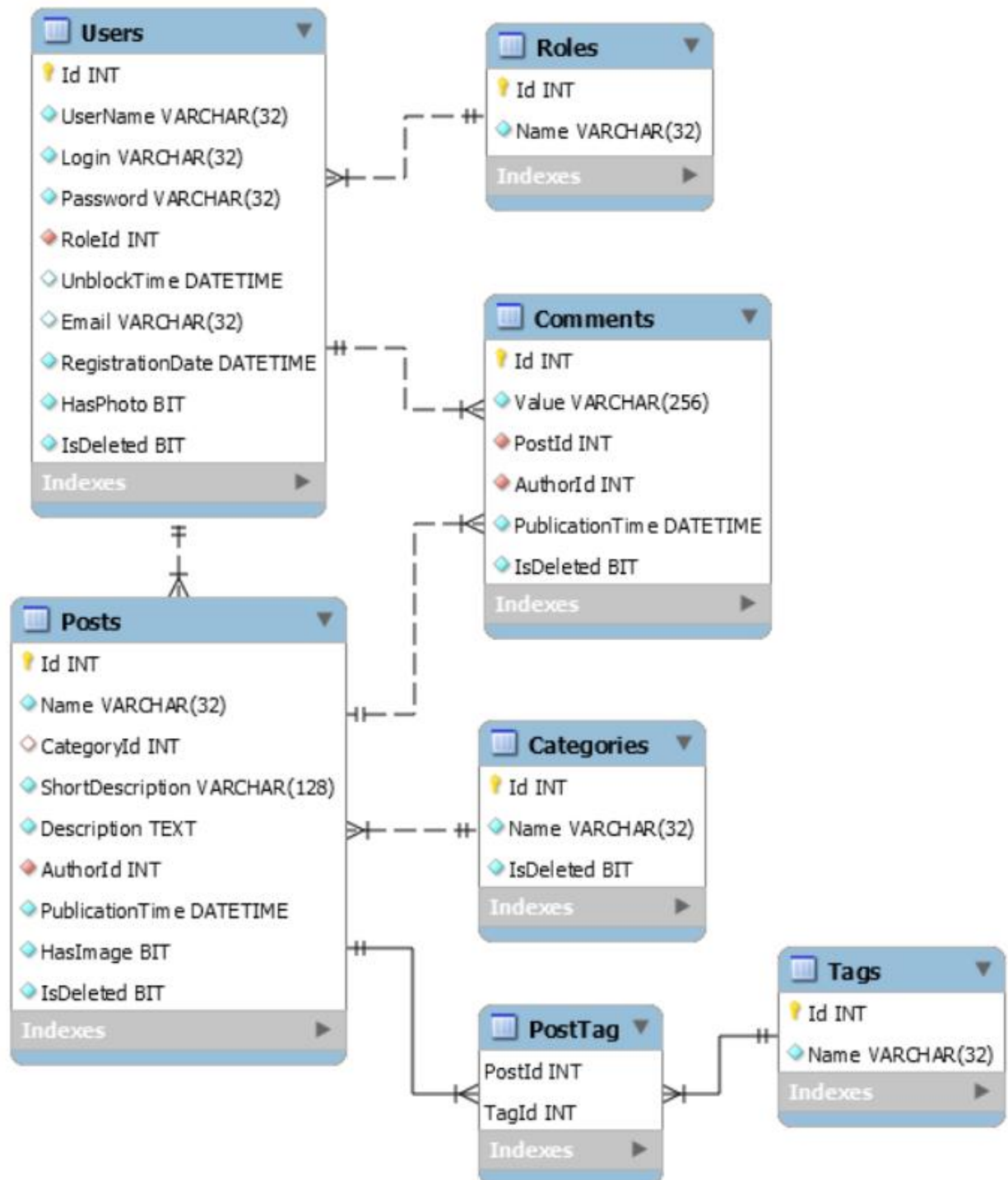


Рисунок 4.1 - ER-диаграмма.

5. Функциональная модель приложения

В рамках предметной области можно выделить четырёх акторов:

- Неавторизованный пользователь;
- Авторизованный пользователь;
- Администратор;
- Аккаунт-менеджер.

Рассмотрим функциональные возможности каждого актора.

Неавторизованный пользователь может:

- просматривать посты и комментарии к ним;
- просматривать профили других пользователей;
- зарегистрировать аккаунт;
- войти в аккаунт.

Авторизованный пользователь может:

- просматривать посты и комментарии к ним;
- создавать посты;
- редактировать свои посты;
- удалять свои посты;
- писать комментарии;
- удалять свои комментарии;
- просматривать профили других пользователей;
- редактировать информацию в своём профиле;
- сменить пароль от своей учётной записи.

Администратор помимо возможностей обычного авторизованного пользователя обладает следующими возможностями:

- удалять посты других пользователей;
- блокировать других пользователей;
- добавлять категории;
- изменять категории;
- удалять категории.

Аккаунт-менеджер помимо возможностей обычного авторизованного пользователя обладает следующими возможностями:

- блокировать других пользователей;
- изменять роли других пользователей.

6. Структура приложения

Решение состоит из 5 проектов (Рисунок 6.1). Рассмотрим каждый проект по отдельности.

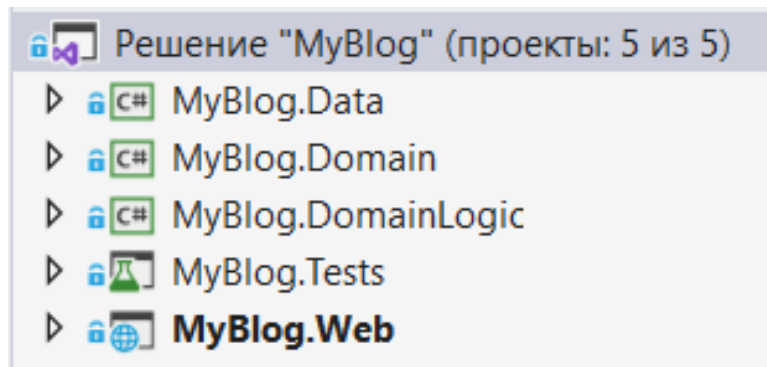


Рисунок 6.1 - Обозреватель решений (общий вид).

- **MyBlog.Data** - представляет собой уровень работы с данными. Содержит настройку контекста и инициализатор базы данных (рисунок 6.2).

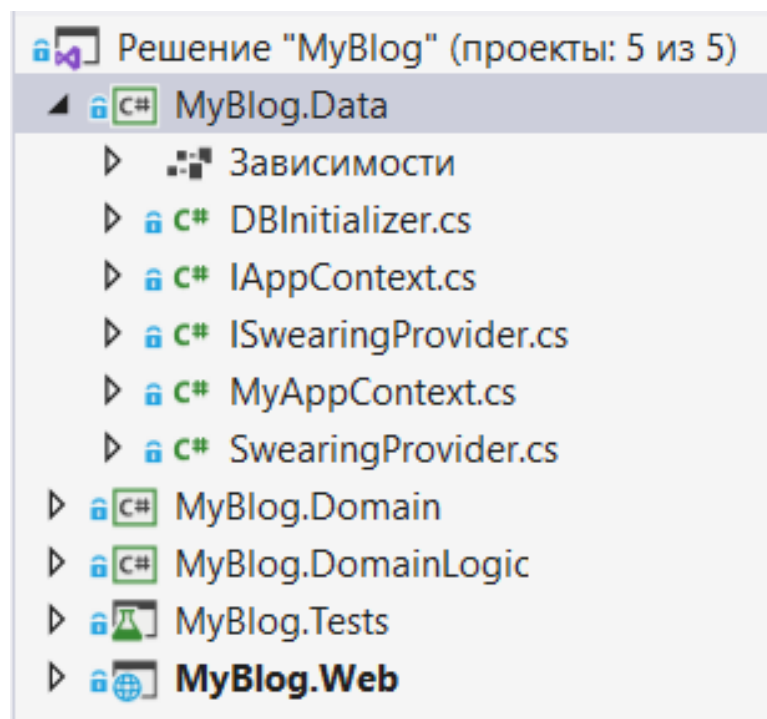


Рисунок 6.2 - Обозреватель решений (MyBlog.Data).

- **MyBlog.Domain** - содержит описание доменных моделей приложения (рисунок 6.3).

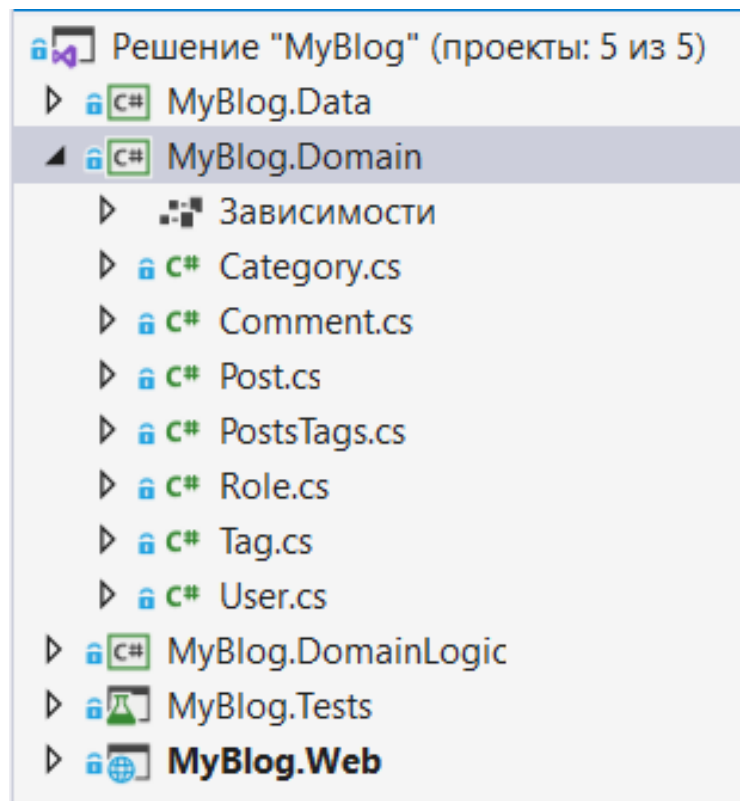


Рисунок 6.3 - Обзорщик решений (MyBlog.Domain).

- **MyBlog.Tests** - содержит юнит тесты приложения (рисунок 6.4).

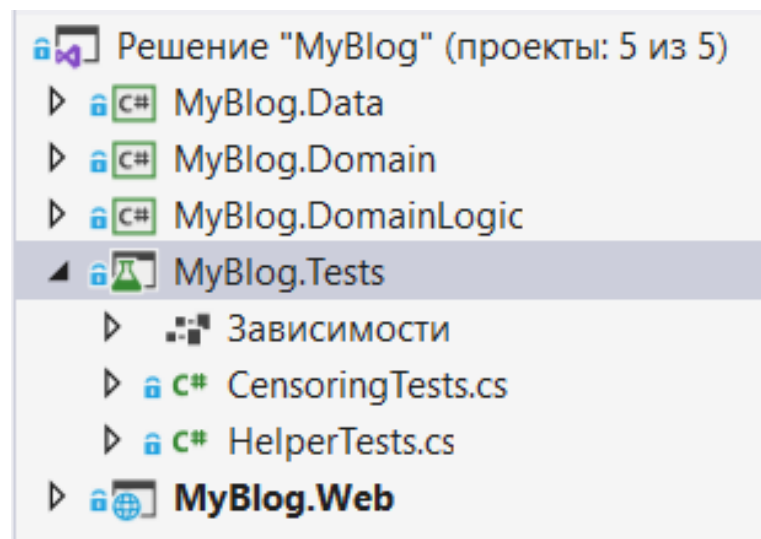


Рисунок 6.4 - Обзорщик решений (MyBlog.Tests).

- **MyBlog.DomainLogic** - содержит бизнес-логику проекта: фильтры, хелперы, менеджеры, DTO, всевозможные сервисы и прочее (рисунок 6.5).

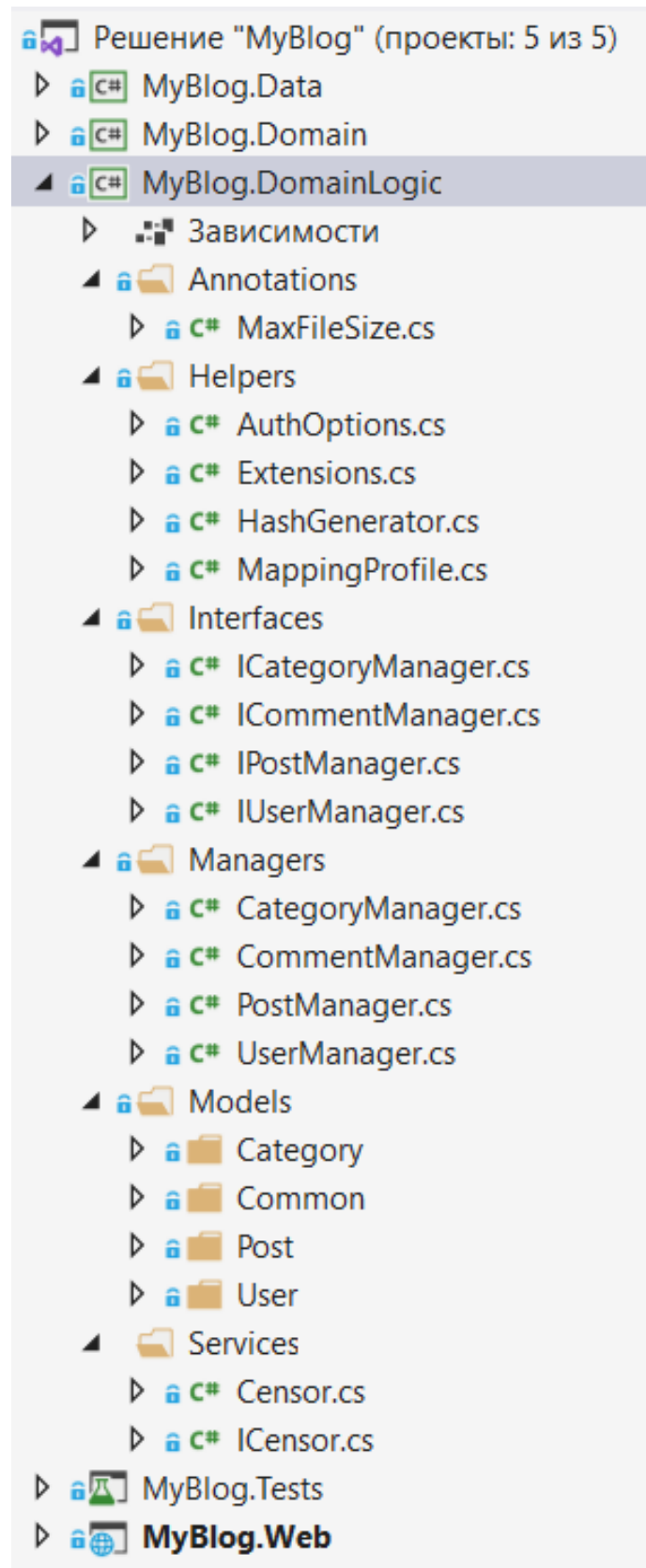


Рисунок 6.5 - Обзорщик решений (MyBlog.DomainLogic).

- **MyBlog.Web** - основной проект приложения. Содержит контроллеры, React компоненты, статические ресурсы, а также некоторые фильтры и сервисы серверного уровня (рисунок 6.6).

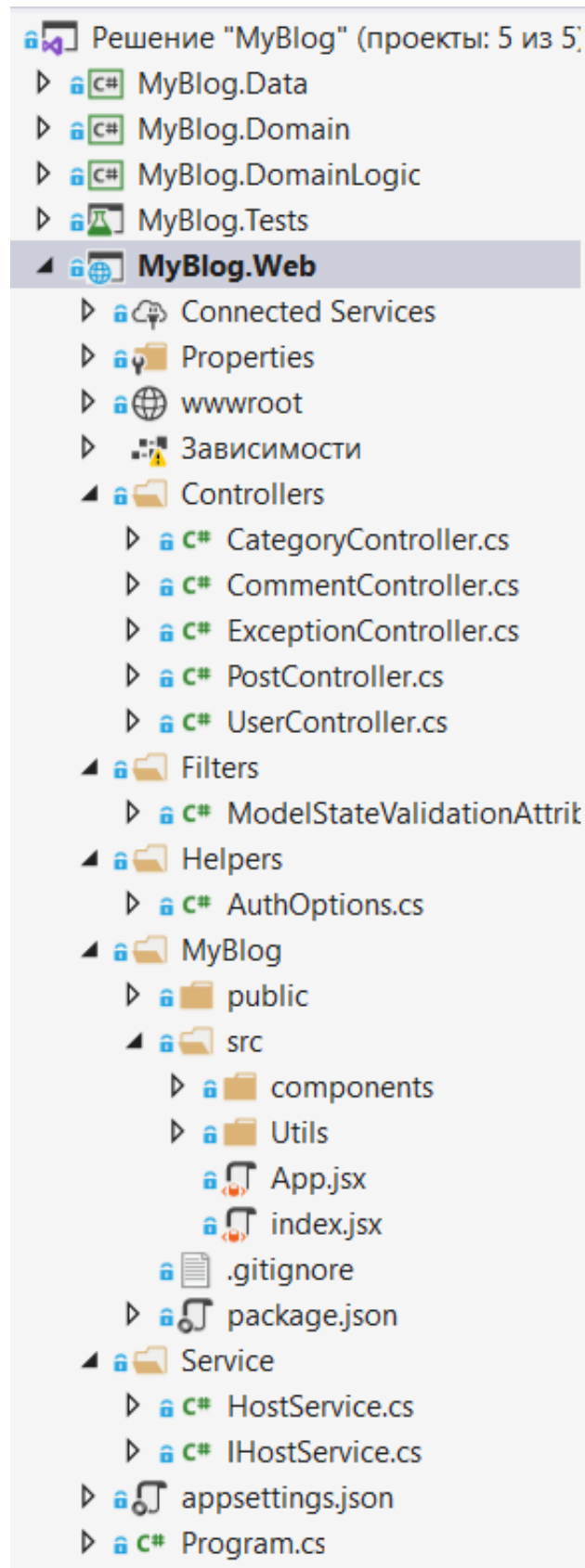


Рисунок 6.6 - Обзорщик решений (MyBlog.Web).

7. Демонстрация использования разработанного программного средства

Рассмотрим основные страницы приложения.

На рисунке 7.1 показана главная страница приложения.

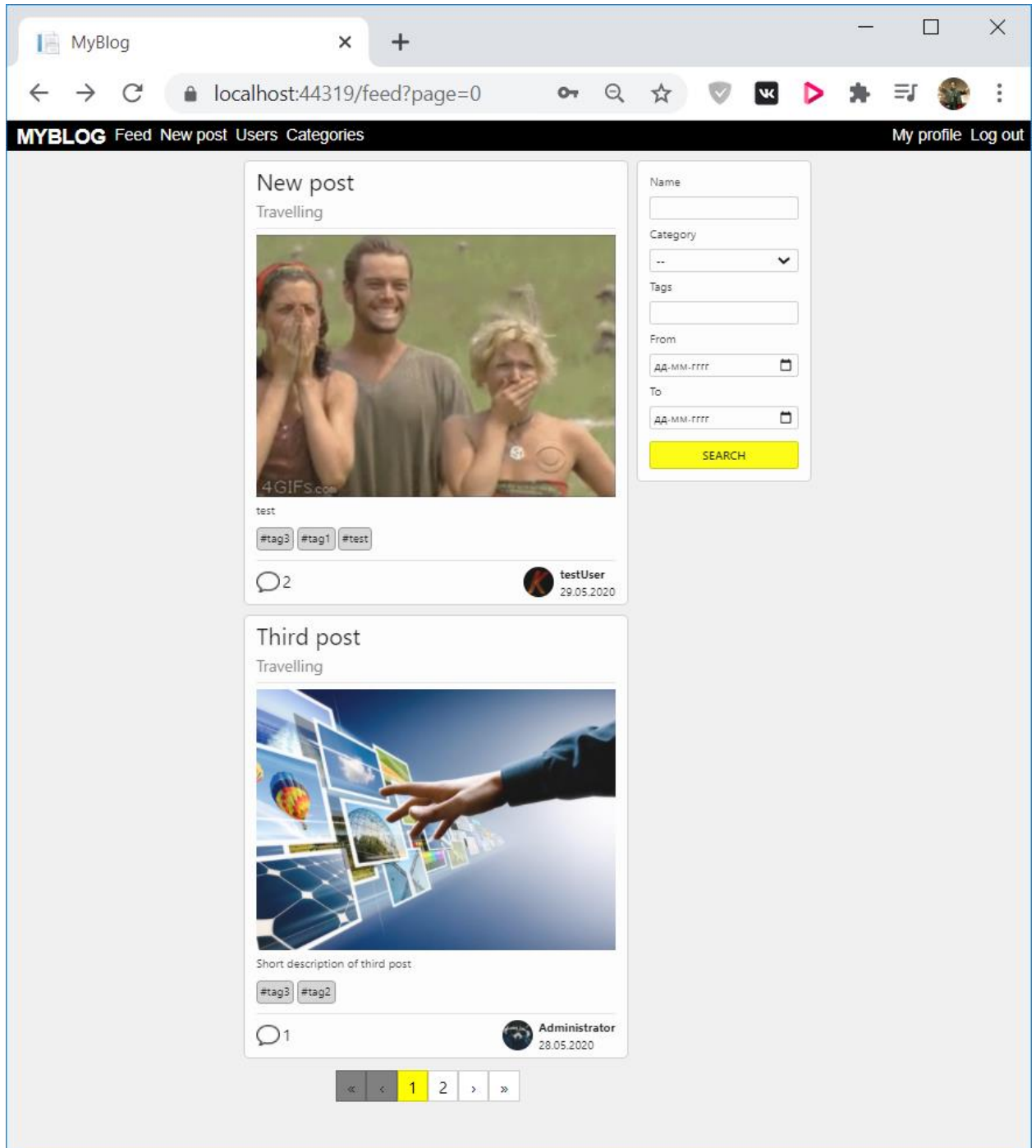


Рисунок 7.1 - Главная страница приложения.

На ней мы можем видеть ленту постов с краткой информацией. Справа находится панель фильтра для поиска постов по определённым критериям. Снизу находится пагинатор.

При нажатии на название или картинку поста мы попадём на страницу данного поста (рисунок 7.2). Здесь содержится подробная информация о посте, а также список комментариев.

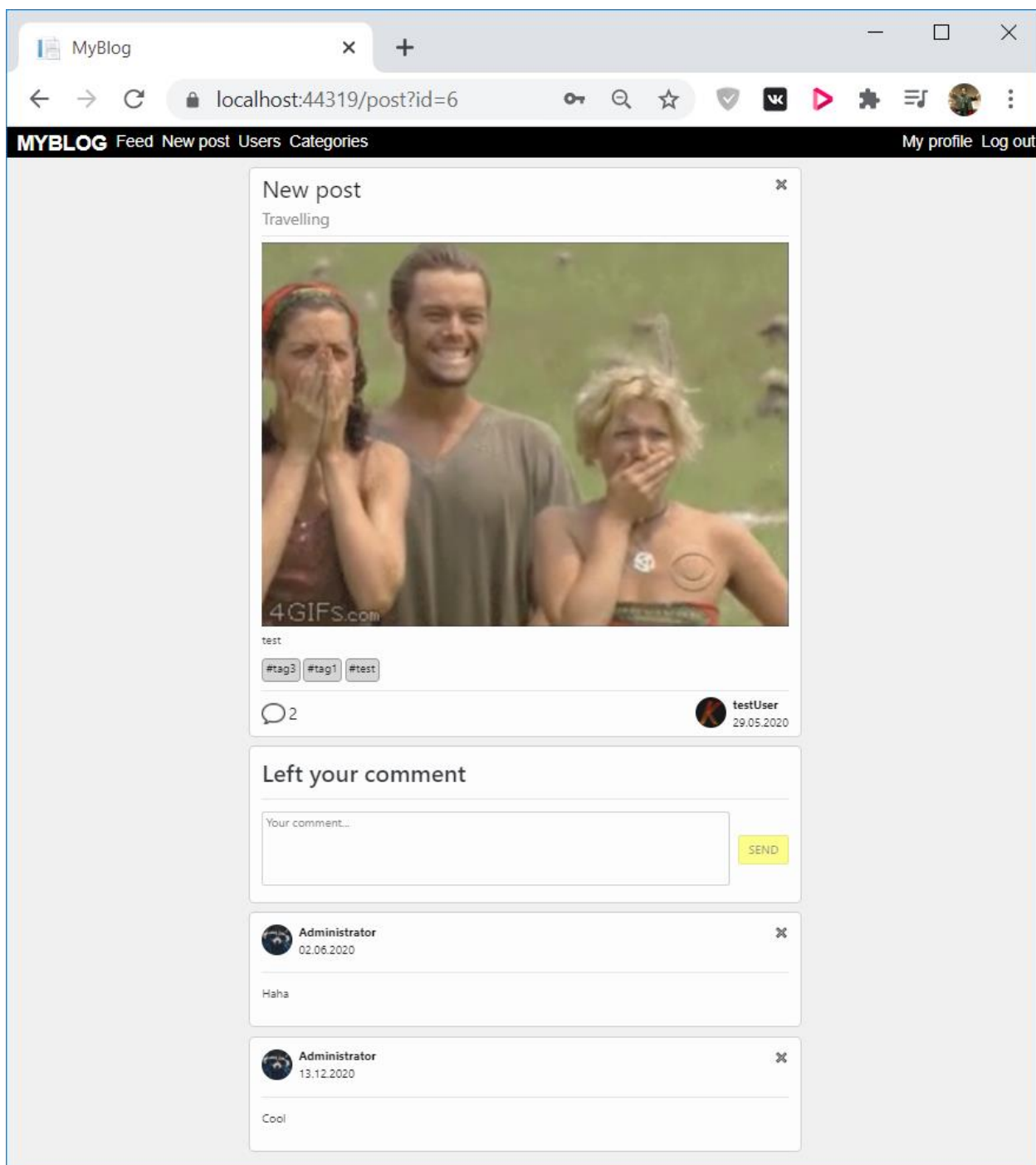


Рисунок 7.2 - Страница поста.

Для того, чтобы получить доступ к остальному функционалу, необходимо сперва зарегистрироваться. На рисунке 7.3 показана форма регистрации нового пользователя. Введённые данные предварительно валидируются на стороне клиента.

The image shows a web browser window with the address bar displaying 'localhost:44319/signUp'. The page title is 'MyBlog'. The header is black with 'MYBLOG Feed' on the left and 'Sign up Sign in' on the right. The main content area is light gray and contains a white registration form. The form has the title 'Registration' and five input fields: 'Username', 'Email', 'Login', 'Password', and 'Confirm password'. Below the fields is a yellow button labeled 'SIGN UP'.

Рисунок 7.3 - Страница регистрации нового пользователя.

После регистрации пользователь может войти в систему. На рисунке 7.4 продемонстрирована страница аутентификации.

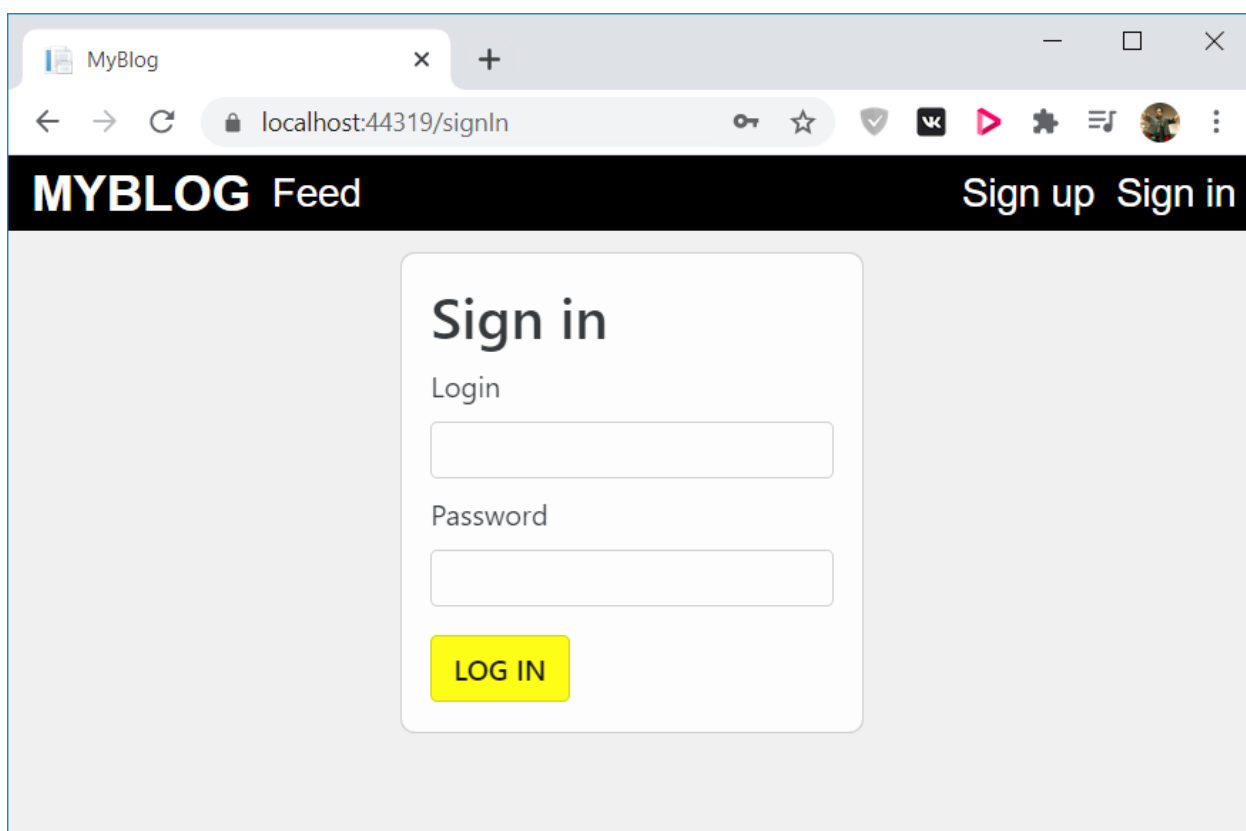
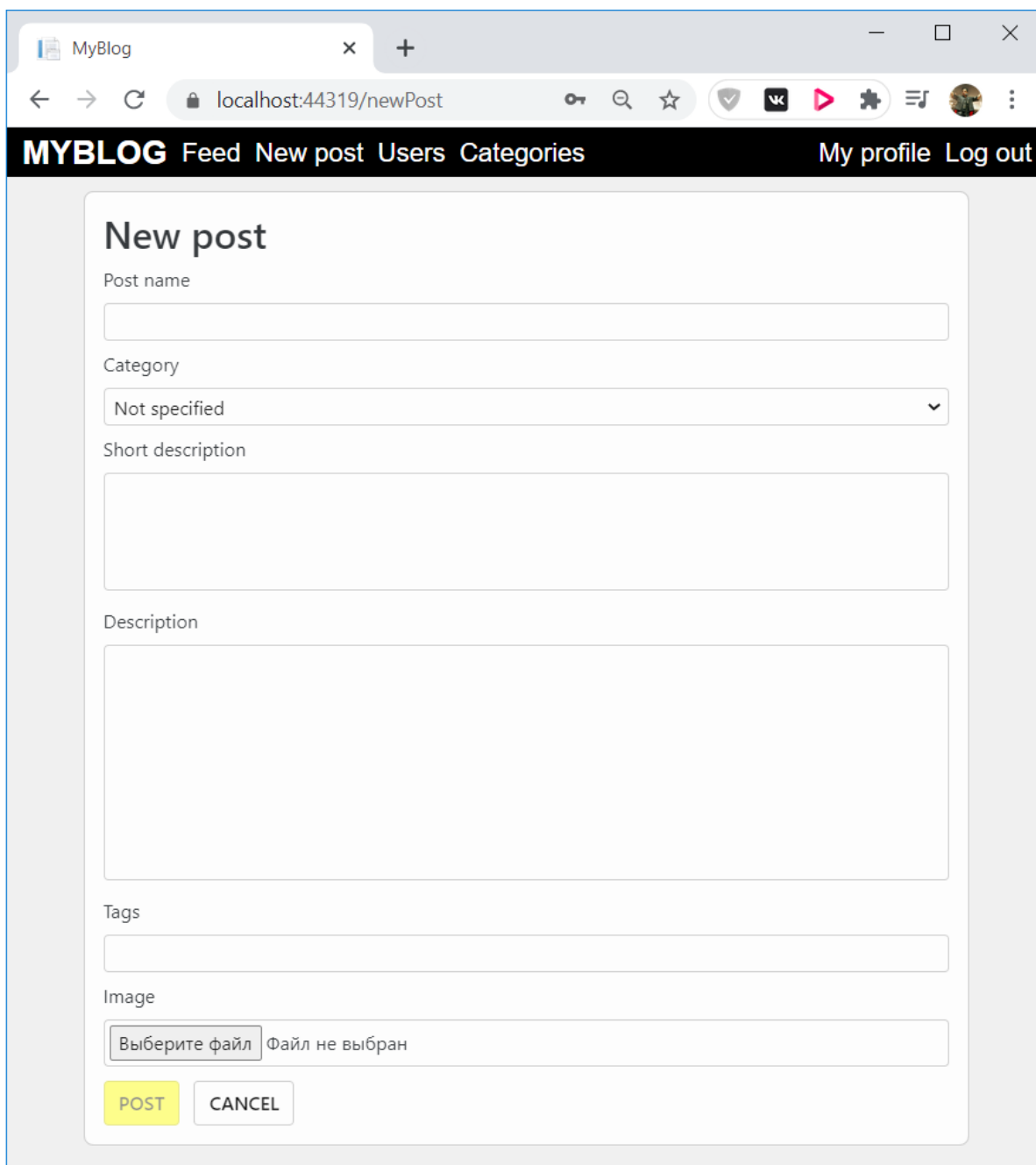


Рисунок 7.4 - Страница аутентификации.

Войдя в систему и нажав на кнопку “New post”, пользователь попадает на форму создания нового поста (рисунок 7.5). Введённые данные предварительно валидируются на стороне клиента.



MYBLOG Feed New post Users Categories My profile Log out

New post

Post name

Category

Not specified

Short description

Description

Tags

Image

Выберите файл Файл не выбран

POST CANCEL

Рисунок 7.5 - Страница создания нового поста.

Свой пост после публикации можно отредактировать. На рисунке 7.6 показана форма редактирования поста.

MyBlog

localhost:44319/editPost?id=1

MYBLOG Feed New post Users Categories My profile Log out

Edit post

Post name

Third post

Category

Travelling

Short description

Short description of third post

Description

Suspendisse porta nisi augue, quis mattis odio iaculis nec. Pellentesque tristique tempus turpis. Cras orci mauris, fermentum non tempus vitae, feugiat quis turpis. Integer sed turpis id magna egestas dignissim. Cras eros ipsum, dictum vehicula dictum ut, sagittis at erat. Vivamus id posuere lorem. Donec aliquet massa ac nibh consequat vulputate.

Tags

tag3, tag2

Image

Выберите файл Файл не выбран

SAVE CANCEL

Рисунок 7.6 - Страница редактирования поста.

Нажав на кнопку “My profile”, пользователь попадает на страницу своего профиля (рисунок 7.7). Аналогичная страница. Профили других пользователей будут иметь аналогичный вид. Здесь содержится информация о пользователе и панель с кнопками, предоставляющими определённый функционал, в зависимости от типа вашей учётной записи и отношения к данному пользователю.

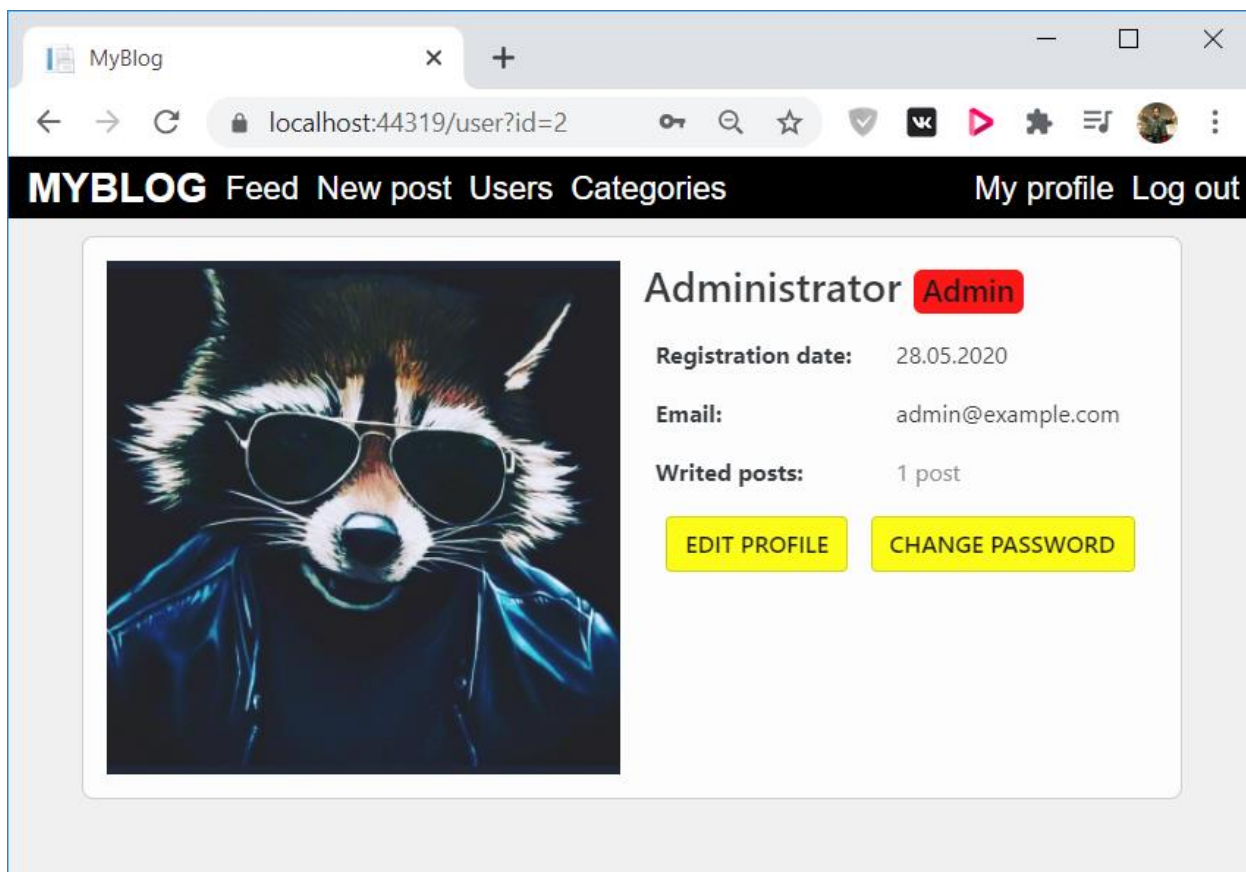
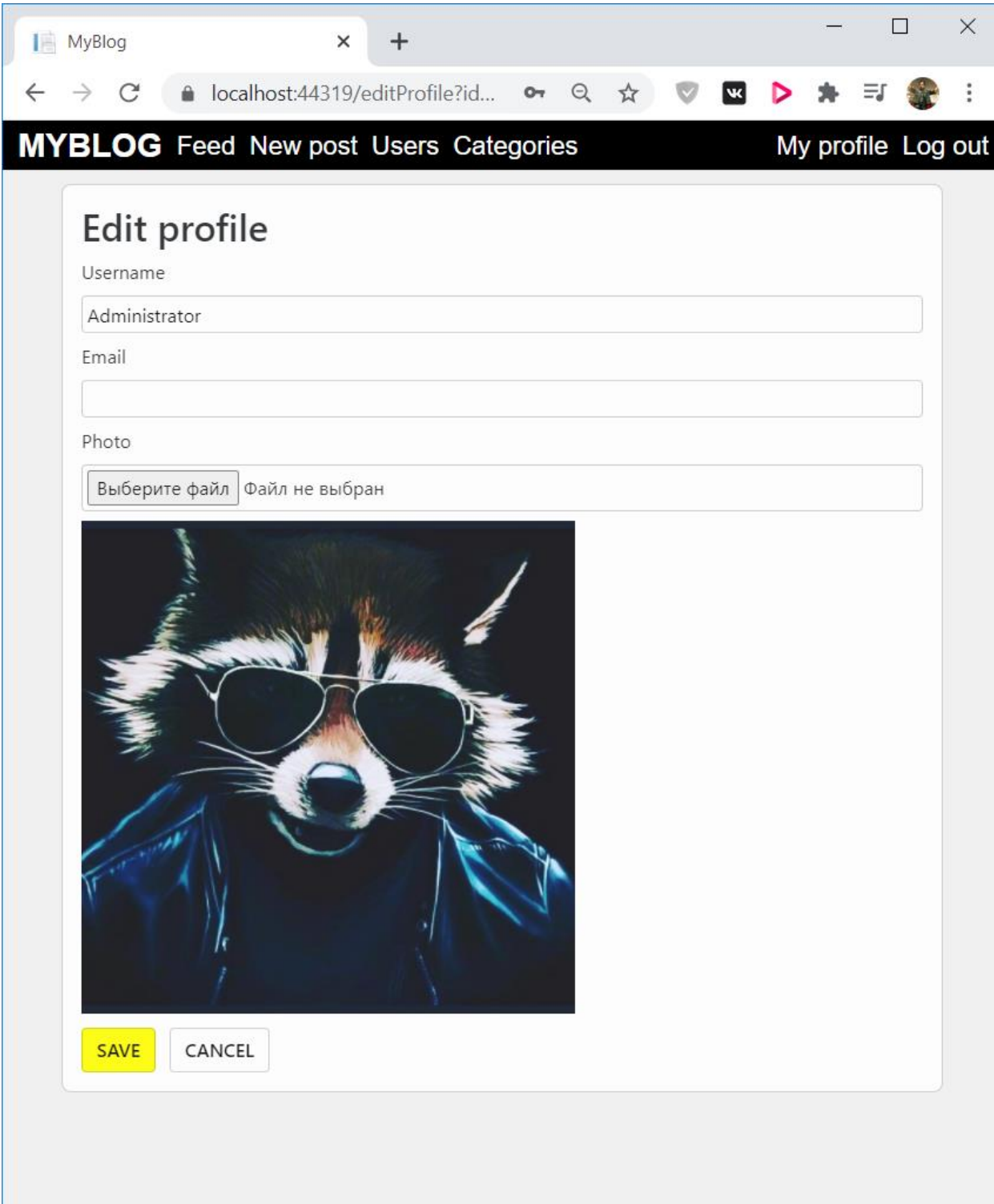


Рисунок 7.7 - Страница профиля пользователя.

Свой профиль можно отредактировать. На рисунке 7.8 показана форма редактирования профиля.



MyBlog

localhost:44319/editProfile?id...

MYBLOG Feed New post Users Categories My profile Log out

Edit profile


Username

Administrator

Email

Photo

Выберите файл Файл не выбран



SAVE CANCEL

Рисунок 7.8 - Страница редактирования профиля.

Также со страницы своего профиля можно попасть на страницу смены пароля (рисунок 7.9).

MyBlog

localhost:44319/changePassword?id=2

MYBLOG Feed New post Users Categories My profile Log out

Password changing

Old password

New password

Confirm new password

SAVE CANCEL

Рисунок 7.9 - Страница смены пароля.

На рисунке 7.10 продемонстрирована страница со списком пользователей. Данная страница доступна только для администраторов и аккаунт-менеджеров.

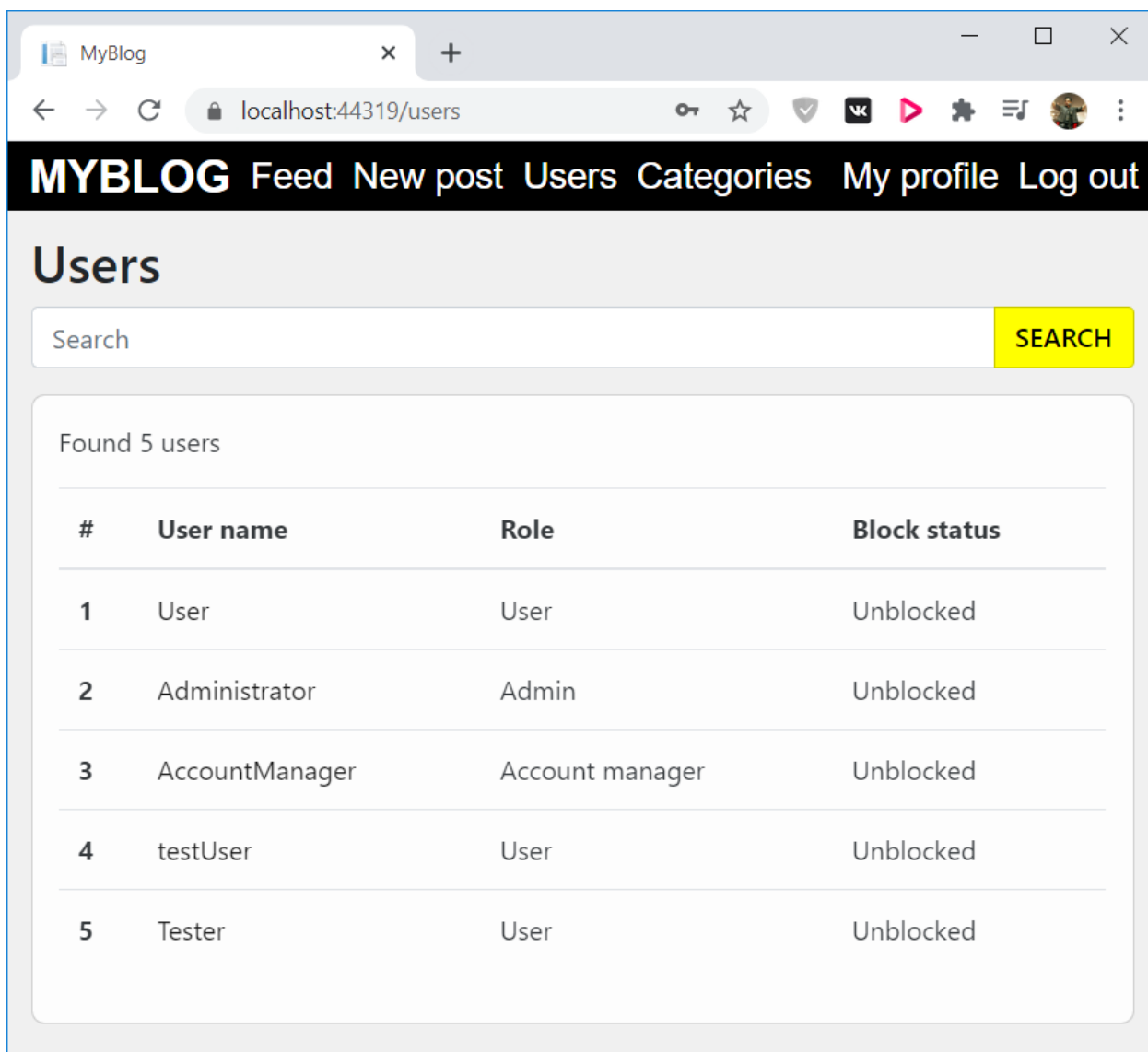


Рисунок 7.10 - Страница пользователей.

На рисунке 7.11 показана форма блокировки пользователя. Данная возможность доступна только для администраторов и аккаунт-менеджеров.

The screenshot shows a web browser window with the address bar displaying 'localhost:44319/blocking?id=1'. The browser's tab is labeled 'MyBlog'. The page features a black navigation bar with the text 'MYBLOG' in white, followed by links: 'Feed', 'New post', 'Users', 'Categories', 'My profile', and 'Log out'. The main content area is light gray and contains a white modal form titled 'Blocking of User'. Inside the form, there are two input fields: 'Number of days' and 'Number of hours', both containing the value '0'. At the bottom of the form are two buttons: a red 'BAN' button and a white 'CANCEL' button with a gray border.

Рисунок 7.11 - Страница блокировки пользователя.

На рисунке 7.12 продемонстрирована страница со списком категорий. Здесь можно отредактировать, удалить или создать новую категорию. Эти возможности доступны только администратору.

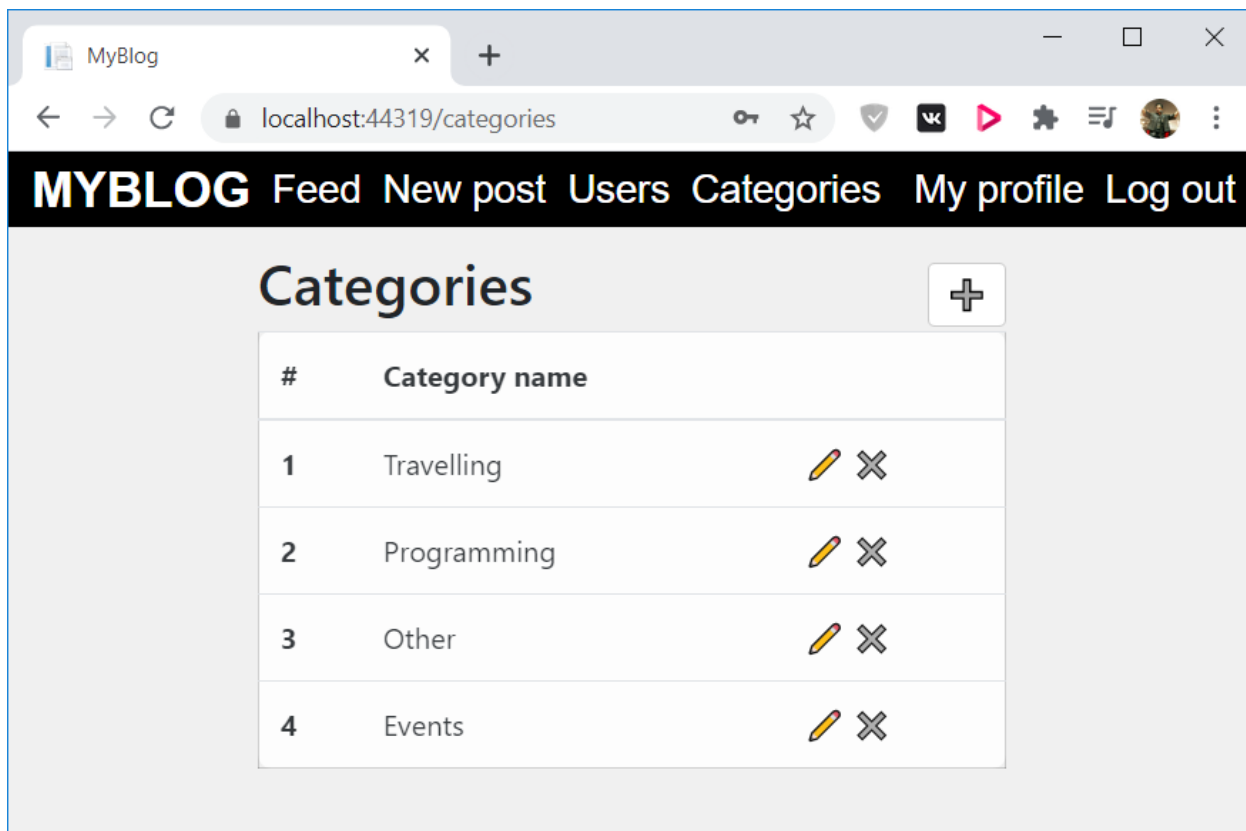


Рисунок 7.12 - Страница категорий.

Заключение

В ходе работы над курсовым проектом было разработано программное средство, представляющее собой публичный текстовый блог. Были получены или углублены знания по таким темам, как ASP.NET Core 3.1, Entity Framework 6, React, CSS, xUnit, git, HTTP.

Также у данного проекта есть перспективы расширения. Например, система рейтинга и подписки на понравившихся авторов.

Список использованных источников

[1] Адам Фримен. ASP.NET Core MVC 2 с примерами на С# для профессионалов Foundation / Адам Фримен. — СПб.: Диалектика, 2019. — 1008 с.

[2] ReactJS [Интернет-ресурс]. - Документация. - Режим доступа: <https://ru.reactjs.org/>

[3] Wikipedia [Интернет-ресурс]. - ASP.NET Core. - Режим доступа: https://ru.wikipedia.org/wiki/ASP.NET_Core

[4] Wikipedia [Интернет-ресурс]. - xUnit. - Режим доступа: <https://ru.wikipedia.org/wiki/XUnit>

Приложение 1. Исходный код основных файлов проекта

AppContext.cs

```
public sealed class MyAppContext : DbContext, IAppContext
{
    public MyAppContext(DbContextOptions<MyAppContext> options) : base(options)
    {
        Database.EnsureCreated();
        this.ChangeTracker.LazyLoadingEnabled = false;
    }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) =>
        optionsBuilder.EnableSensitiveDataLogging();

    public Microsoft.EntityFrameworkCore.DbSet<Post> Posts { get; set; }
    public Microsoft.EntityFrameworkCore.DbSet<Category> Categories { get; set; }
    public Microsoft.EntityFrameworkCore.DbSet<Tag> Tags { get; set; }
    public Microsoft.EntityFrameworkCore.DbSet<User> Users { get; set; }
    public Microsoft.EntityFrameworkCore.DbSet<Role> Roles { get; set; }
    public Microsoft.EntityFrameworkCore.DbSet<PostsTags> PostsTags { get; set; }
    public Microsoft.EntityFrameworkCore.DbSet<Comment> Comments { get; set; }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        builder.Entity<Tag>()
            .HasMany(t => t.Posts)
            .WithMany(p => p.Tags)
            .UsingEntity<PostsTags>(
                pt => pt
                    .HasOne(pt => pt.Post)
                    .WithMany()
                    .HasForeignKey("PostId"),
                pt => pt
                    .HasOne(pt => pt.Tag)
                    .WithMany()
                    .HasForeignKey("TagId"))
            .ToTable("PostsTags")
            .HasKey(et => new { et.PostId, et.TagId });

        builder.Entity<User>()
            .HasKey(u => u.Id);
        builder.Entity<User>()
            .Property(u => u.Id)
            .ValueGeneratedOnAdd();
        builder.Entity<User>()
            .HasMany(u => u.Posts)
            .WithOne(p => p.Author)
            .OnDelete(DeleteBehavior.SetNull);
        builder.Entity<User>()
            .HasMany(u => u.Comments)
            .WithOne(c => c.Author)
            .OnDelete(DeleteBehavior.SetNull);
        builder.Entity<User>()
            .HasQueryFilter(u => !u.IsDeleted);

        builder.Entity<Role>()
            .HasKey(r => r.Id);
        builder.Entity<Role>()
            .Property(r => r.Id)
            .ValueGeneratedOnAdd();

        builder.Entity<Post>()
            .HasKey(e => e.Id);
    }
}
```

```

builder.Entity<Post>()
    .Property(e => e.Id)
    .ValueGeneratedOnAdd();
builder.Entity<Post>()
    .HasQueryFilter(e => !e.IsDeleted);
builder.Entity<Post>()
    .HasMany(p => p.Comments)
    .WithOne(c => c.Post)
    .OnDelete(DeleteBehavior.SetNull);

builder.Entity<Category>()
    .HasKey(c => c.Id);
builder.Entity<Category>()
    .Property(c => c.Id)
    .ValueGeneratedOnAdd();
builder.Entity<Category>()
    .HasQueryFilter(c => !c.IsDeleted);

builder.Entity<Tag>()
    .HasKey(t => t.Id);
builder.Entity<Tag>()
    .Property(t => t.Id)
    .ValueGeneratedOnAdd();

builder.Entity<Comment>()
    .HasKey(c => c.Id);
builder.Entity<Comment>()
    .Property(c => c.Id)
    .ValueGeneratedOnAdd();
builder.Entity<Comment>()
    .HasQueryFilter(c => !c.IsDeleted);
}
}

```

PostManager.cs

```

public class PostManager : IPostManager
{
    private readonly IAppContext _appContext;
    private readonly IMapper _mapper;

    public PostManager(IAppContext appContext, IMapper mapper)
    {
        _appContext = appContext;
        _mapper = mapper;
    }

    public async Task AddPostAsync(PostCreateDto post, string hostRoot)
    {
        var newPost = _mapper.Map<PostCreateDto, Post>(post);
        await _appContext.Posts.AddAsync(newPost);
        await _appContext.SaveChangesAsync(default);

        if (post.Image != null)
        {
            string path = $"{hostRoot}\\wwwroot\\img\\posts\\{newPost.Id}.jpg";
            await using var fileStream = new FileStream(path, FileMode.Create);
            await post.Image.CopyToAsync(fileStream);
        }

        if (post.Tags != null)
        {
            var tags = post.Tags.ParseSubstrings(",");
            foreach (var tagName in tags)
            {

```



```

        var tag = await _appContext.Tags.FirstOrDefaultAsync(t =>
string.Equals(t.Name.ToLower(), tagName.ToLower()));
        if (tag == null)
        {
            tag = _mapper.Map<string, Tag>(tagName);
            await _appContext.Tags.AddAsync(tag);
            await _appContext.SaveChangesAsync(default);
        }
        await _appContext.PostsTags.AddAsync(new PostsTags { PostId = newPost.Id, TagId =
tag.Id });
    }

    await _appContext.SaveChangesAsync(default);
}

public async Task UpdatePostAsync(PostUpdateDto post, string hostRoot)
{
    var update = await _appContext.Posts.FirstOrDefaultAsync(e => e.Id == post.Id);
    if (update == null)
    {
        throw new NullReferenceException($"Post with id={post.Id} not found");
    }

    _mapper.Map(post, update);
    await _appContext.SaveChangesAsync(default);

    _appContext.PostsTags.RemoveRange(_appContext.PostsTags.Where(et => et.PostId == post.Id));
    if (post.Tags != null)
    {
        var tags = post.Tags.ParseSubstrings(",");
        foreach (var tagName in tags)
        {
            var tag = await _appContext.Tags.FirstOrDefaultAsync(t =>
string.Equals(t.Name.ToLower(), tagName.ToLower()));
            if (tag == null)
            {
                tag = _mapper.Map<string, Tag>(tagName);
                await _appContext.Tags.AddAsync(tag);
                await _appContext.SaveChangesAsync(default);
            }
            await _appContext.PostsTags.AddAsync(new PostsTags { PostId = update.Id, TagId =
tag.Id });
        }

        if (post.Image != null)
        {
            string path = $"{hostRoot}\\wwwroot\\img\\posts\\{update.Id}.jpg";
            await using var fileStream = new FileStream(path, FileMode.Create);
            await post.Image.CopyToAsync(fileStream);
        }

        await _appContext.SaveChangesAsync(default);
    }

    public async Task<Page<PostLiteDto>> GetPostsAsync(int index, int pageSize, string name, int?
categoryId, string tags, string from, string to, int? author)
    {
        List<string> stringTags = tags?.ParseSubstrings(",").ToList();
        List<int> postIdByTags = new List<int>();
        if (!string.IsNullOrEmpty(tags))
        {
            postIdByTags = _appContext.PostsTags
                .Include(pt => pt.Tag)
                .Where(pt => stringTags.Contains(pt.Tag.Name))

```

```

        .AsEnumerable()
        .GroupBy(pt => pt.PostId)
        .Where(g => g.Count() == stringTags.Count)
        .Select(g => g.Key)
        .ToList();
    }

    var result = new Page<PostLiteDto>() { CurrentPage = index, PageSize = pageSize };
    var query = _appContext.Posts.Include(p => p.Category).Include(p=>p.Author).AsQueryable();
    if (name != null)
    {
        query = query.Where(p => p.Name.ToLower().Contains(name.ToLower()));
    }
    if (categoryId.HasValue)
    {
        query = query.Where(p => p.CategoryId == categoryId);
    }
    if (tags != null)
    {
        query = query.Where(p => postIdByTags.Contains(p.Id));
    }
    if (from != null)
    {
        query = query.Where(p => DateTime.ParseExact(from, "d/M/yyyy",
CultureInfo.InvariantCulture) <= p.PublicationTime.Date);
    }
    if (to != null)
    {
        query = query.Where(p => DateTime.ParseExact(to, "d/M/yyyy",
CultureInfo.InvariantCulture) >= p.PublicationTime.Date);
    }
    if (author.HasValue)
    {
        query = query.Where(p => p.AuthorId == author);
    }
    result.TotalRecords = await query.CountAsync();
    query = query.OrderByDescending(p => p.PublicationTime).Skip(index *
pageSize).Take(pageSize);
    result.Records = await _mapper.ProjectTo<PostLiteDto>(query).ToListAsync(default);

    for (int i = 0; i < result.Records.Count; i++)
    {
        var postTags = _appContext.PostsTags.Include(pt => pt.Tag).Where(pt => pt.PostId ==
result.Records[i].Id).Select(pt => pt.Tag).ToHashSet();
        foreach (var tag in postTags)
        {
            result.Records[i].Tags.Add(tag.Id.ToString(), tag.Name);
        }

        result.Records[i].Comments = _appContext.Comments.Count(c => c.PostId ==
result.Records[i].Id);

        if (result.Records[i].HasImage)
        {
            string path = $"img\\posts\\{result.Records[i].Id}.jpg";
            result.Records[i].Image = path;
        }

        result.Records[i].AuthorPhoto = (await _appContext.Users.FirstAsync(u => u.Id ==
result.Records[i].AuthorId)).HasPhoto ?
            $"img\\users\\{result.Records[i].AuthorId}.jpg" :
            "img\\users\\default.jpg";
    }
    return result;
}

```

```

public async Task<PostFullDto> GetPostAsync(int postId)
{
    var DbPost = await _appContext.Posts.Include(p => p.Author).Include(p =>
p.Category).Include(p=>p.Comments).ThenInclude(p=>p.Author).FirstOrDefaultAsync(e => e.Id ==
    postId)
    if (DbPost == null)
    {
        throw new NullReferenceException($"Post with id={postId} not found");
    }
    var postFullDto = _mapper.Map<PostFullDto>(DbPost);

    var tags = _appContext.PostsTags.Include(pt => pt.Tag).Where(pt => pt.PostId ==
postId).Select(pt => pt.Tag).ToHashSet();
    foreach (var tag in tags)
    {
        postFullDto.Tags.Add(tag.Id.ToString(), tag.Name);
    }

    if (DbPost.HasImage)
    {
        postFullDto.Image = $"img\\posts\\{postId}.jpg";
    }

    postFullDto.AuthorPhoto = (await _appContext.Users.FirstAsync(u => u.Id ==
postFullDto.AuthorId)).HasPhoto ?
        $"img\\users\\{postFullDto.AuthorId}.jpg" :
        "img\\users\\default.jpg";

    for (int i = 0; i < postFullDto.Comments.Count; i++)
    {
        postFullDto.Comments.ElementAt(i).AuthorPhoto = (await _appContext.Users.FirstAsync(u =>
u.Id == postFullDto.Comments.ElementAt(i).AuthorId)).HasPhoto ?
            $"img\\users\\{postFullDto.Comments.ElementAt(i).AuthorId}.jpg" :
            "img\\users\\default.jpg";
    }

    return postFullDto;
}

public async Task<PostToUpdateDto> GetPostToUpdateAsync(int postId)
{
    var post = await _appContext.Posts.FirstOrDefaultAsync(p => p.Id == postId);
    if (post == null)
    {
        throw new NullReferenceException($"Post with id={postId} not found");
    }
    PostToUpdateDto postToUpdate = _mapper.Map<PostToUpdateDto>(post);

    if (post.HasImage)
    {
        postToUpdate.Image = $"img\\posts\\{postId}.jpg";
    }
    var tags = _appContext.PostsTags.Include(pt => pt.Tag).Where(pt => pt.PostId ==
postId).Select(pt => pt.Tag.Name).ToHashSet();
    postToUpdate.Tags = String.Join(", ", tags);
    return postToUpdate;
}

public async Task<int> GetPostAuthorIdAsync(int postId)
{
    if (!await _appContext.Posts.AnyAsync(p => p.Id == postId))
    {
        throw new NullReferenceException($"Post with id={postId} not found");
    }
    return (int)(await _appContext.Posts.FirstOrDefaultAsync(p => p.Id == postId)).AuthorId;
}

```

```

public async Task DeletePostAsync(int postId, bool force, string hostRoot)
{
    var post = await _appContext.Posts.IgnoreQueryFilters()
        .FirstOrDefaultAsync(p => p.Id == postId);
    if (post == null)
    {
        throw new NullReferenceException($"Post with id={postId} not found");
    }
    if (force)
    {
        _appContext.Posts.Remove(post);
        string path = $"{hostRoot}\\wwwroot\\img\\posts\\{postId}.jpg";
        File.Delete(path);
    }
    else
    {
        post.IsDeleted = true;
    }
    await _appContext.SaveChangesAsync(default);
}
}

```

CommentManager.cs

```

public class CommentManager : ICommentManager
{
    private readonly IAppContext _appContext;
    private readonly IMapper _mapper;
    public CommentManager(IAppContext appContext, IMapper mapper)
    {
        _appContext = appContext;
        _mapper = mapper;
    }
    public async Task AddCommentAsync(NewCommentDto comment)
    {
        var newComment = _mapper.Map<Comment>(comment);
        await _appContext.Comments.AddAsync(newComment);
        await _appContext.SaveChangesAsync(default);
    }

    public async Task DeleteCommentAsync(int commentId, bool force = false)
    {
        var comment = await _appContext.Comments.IgnoreQueryFilters()
            .FirstOrDefaultAsync(c => c.Id == commentId);
        if (comment == null)
        {
            throw new NullReferenceException($"Comment with id={commentId} not found");
        }
        if (force)
        {
            _appContext.Comments.Remove(comment);
        }
        else
        {
            comment.IsDeleted = true;
        }
        await _appContext.SaveChangesAsync(default);
    }

    public async Task<int?> GetCommentAuthorIdAsync(int commentId)
    {
        if (!await _appContext.Comments.AnyAsync(c => c.Id == commentId))
        {
            throw new NullReferenceException($"Comment with id={commentId} not found");
        }
        return (await _appContext.Comments.FirstOrDefaultAsync(c => c.Id == commentId)).AuthorId;
    }
}

```

```

    }
}

```

CategoryManager.cs

```

public class CategoryManager : ICategoryManager
{
    private readonly IAppContext _appContext;
    private readonly IMapper _mapper;

    public CategoryManager(IAppContext appContext, IMapper mapper)
    {
        _appContext = appContext;
        _mapper = mapper;
    }

    public async Task<ICollection<Category>> GetCategoriesAsync()
    {
        return await _appContext.Categories.ToListAsync();
    }

    public async Task<Category> GetCategoryAsync(int categoryId)
    {
        var category = await _appContext.Categories.FirstOrDefaultAsync(c => c.Id == categoryId);
        if (category == null)
        {
            throw new NullReferenceException($"Category with id={categoryId} not found");
        }
        return category;
    }

    public async Task AddCategoryAsync(CategoryCreateDto category)
    {
        if (await _appContext.Categories.AnyAsync(c => Equals(c.Name.ToLower(),
category.Name.ToLower()))))
        {
            throw new ArgumentOutOfRangeException("Category with this name already exist");
        }
        await _appContext.Categories.AddAsync(_mapper.Map<Category>(category));
        await _appContext.SaveChangesAsync(default);
    }

    public async Task UpdateCategoryAsync(Category category)
    {
        var update = await _appContext.Categories.FirstOrDefaultAsync(c => c.Id == category.Id);
        if (update == null)
        {
            throw new KeyNotFoundException($"Category with id={category.Id} not found");
        }
        update.Name = category.Name;
        await _appContext.SaveChangesAsync(default);
    }

    public async Task DeleteCategoryAsync(int categoryId, bool force = false)
    {
        var category = await _appContext.Categories.IgnoreQueryFilters()
            .FirstOrDefaultAsync(c => c.Id == categoryId);
        if (category == null)
        {
            throw new KeyNotFoundException($"Category with id={categoryId} not found");
        }
        if (force)
        {
            _appContext.Categories.Remove(category);
        }
        else
    }
}

```

```

        {
            category.IsDeleted = true;
        }
        await _appContext.SaveChangesAsync(default);
    }
}

```

UserManager.cs

```

public class UserManager : IUserManager
{
    private readonly IAppContext _appContext;
    private readonly IMapper _mapper;
    public UserManager(IAppContext appContext, IMapper mapper)
    {
        _appContext = appContext;
        _mapper = mapper;
    }

    public async Task RegisterUserAsync(RegisterDto user)
    {
        if (await _appContext.Users.AnyAsync(u => string.Equals(u.Login.ToLower(),
user.Login.ToLower())))
        {
            throw new ArgumentException($"User with login \"{user.Login}\" already exist");
        }
        User newUser = _mapper.Map<User>(user);
        newUser.RoleId = (await _appContext.Roles.FirstOrDefaultAsync(r => r.Name == "User"))?.Id;
        await _appContext.Users.AddAsync(newUser);
        await _appContext.SaveChangesAsync(default);
    }

    public async Task UpdateUserAsync(UserUpdateDto user, string hostRoot)
    {
        User updatedUser = await _appContext.Users.FirstOrDefaultAsync(u => u.Id == user.Id);
        if (updatedUser == null)
        {
            throw new NullReferenceException($"User with id={user.Id} not found");
        }
        _mapper.Map(user, updatedUser);
        if (user.Photo != null)
        {
            string path = $"{hostRoot}\\wwwroot\\img\\users\\{updatedUser.Id}.jpg";
            await using var fileStream = new FileStream(path, FileMode.Create);
            await user.Photo.CopyToAsync(fileStream);
        }
        await _appContext.SaveChangesAsync(default);
    }

    public async Task<DateTime?> GetUnlockTimeAsync(int userId)
    {
        if (!await _appContext.Users.AnyAsync(u => u.Id == userId))
        {
            throw new NullReferenceException($"User with id={userId} not found");
        }
        return (await _appContext.Users.FirstAsync(u => u.Id == userId)).UnlockTime;
    }

    public async Task<LoginResponseDto> LoginAsync(LoginDto model)
    {
        var identity = await GetIdentityAsync(model.Login, model.Password);
        if (identity == null)
        {
            throw new NullReferenceException($"Wrong login or password");
        }
    }
}

```

```

var unlockTime = await GetUnlockTimeAsync(Int32.Parse(identity.Name));
if (unlockTime != null && ((unlockTime ?? DateTime.Now) > DateTime.Now))
{
    throw new UnauthorizedAccessException($"Banned until {unlockTime?.ToString("f")}");
}

var now = DateTime.UtcNow;
var jwt = new JwtSecurityToken(
    issuer: AuthOptions.ISSUER,
    audience: AuthOptions.AUDIENCE,
    notBefore: now,
    claims: identity.Claims,
    expires: now.Add(TimeSpan.FromMinutes(AuthOptions.LIFETIME)),
    signingCredentials: new SigningCredentials(AuthOptions.GetSymmetricSecurityKey(),
SecurityAlgorithms.HmacSha256));
var encodedJwt = new JwtSecurityTokenHandler().WriteToken(jwt);

var response = new LoginResponseDto
{
    AccessToken = encodedJwt,
    Name = identity.Name,
    Role = identity.Claims.Where(c => c.Type == ClaimTypes.Role).FirstOrDefault().Value,
};
return response;
}

private async Task<ClaimsIdentity> GetIdentityAsync(string login, string password)
{
    ClaimsIdentity identity = null;
    var user = await _appContext.Users.Include(u => u.Role).FirstOrDefaultAsync(u =>
string.Equals(u.Login, login));
    if (user == null)
    {
        return null;
    }
    var passwordHash = HashGenerator.Encrypt(password);
    if (passwordHash == user.Password)
    {
        var claims = new List<Claim>
        {
            new Claim(ClaimsIdentity.DefaultNameClaimType, user.Id.ToString()),
            new Claim(ClaimsIdentity.DefaultRoleClaimType, user.Role.Name)
        };
        identity = new ClaimsIdentity(claims, "Token", ClaimsIdentity.DefaultNameClaimType,
ClaimsIdentity.DefaultRoleClaimType);
    }
    return identity;
}

```