Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ
НА ТЕМУ:
Платформа для ведения онлайн-дневников (блогов)

Студент ИУ7-64Б _____ Керимов А. Ш.
(Группа) (Подпись, дата) (Фамилия И. О.)

Руководитель курсовой работы _____ Тассов К. Л.
(Подпись, дата) (Фамилия И. О.)

2020 г.

Оглавление

Введение

В современном мире, одну из лидирующих позиций занимает информационное пространство. Это публичная площадка в сети Интернет, где человек излагает свои мысли. Множество людей наблюдает за подобными публичными площадками и людьми, которые их ведут в различных социальных сетях.

Один из наиболее интересных видов информационного пространства — блог. Термин «блог» произошёл от английского weblog («logging the web» — записывать события в сеть). Впервые его использовал американский программист Йорн Баргер в 1997 году для обозначения сетевого дневника [1].

Целью данного курсового проекта является создание платформы для ведения онлайн-дневников (блогов).

Задачи работы

В рамках выполнения проекта необходимо решить следующие задачи:

- формализовать задачу в виде определения необходимого функционала;
- провести анализ существующих СУБД;
- спроектировать базу данных, необходимую для хранения и структурирования данных;
- программно реализовать спроектированную базу данных с использованием выбранной СУБД;
- разработать приложение для взаимодействия с реализованной БД.

1 Аналитический раздел

В данном разделе будут рассмотрены общие сведения о БД и СУБД, популярные СУБД и используемый фреймворк.

1.1 Формализация задачи

В соответствии с техническим заданием на курсовой проект необходимо разработать платформу для ведения онлайн-дневников (блогов), в которой пользователи смогут авторизоваться и аутентифицироваться, писать статьи и комментарии к ним. Необходимо обеспечить возможность пользователей оценивать статьи и комментарии, сортировать их по дате публикации и по рейтингу, просматривать список статей определённого автора, а также фильтровать их по тэгам.

1.2 Общие сведения о БД и СУБД

База данных — представленная в объективной форме совокупность самостоятельных материалов (статей, расчётов, нормативных актов, судебных решений и иных подобных материалов), систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины (ЭВМ) [2].

Под системой управления базами данных (СУБД) понимается совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных [3].

1.3 Разбор различных СУБД

Несмотря на то, что все системы управления базами данных выполняют одну и ту же основную задачу, сам процесс выполнения этой задачи варьируется в широких пределах. Кроме того, функции и возможности

каждой СУБД могут существенно отличаться. Различные СУБД документированы по-разному: более или менее тщательно. По-разному предоставляется и техническая поддержка. При сравнении различных популярных баз данных, следует учитывать, удобна ли для пользователя и масштабируема ли данная конкретная СУБД, а также убедиться, что она будет хорошо интегрироваться с другими продуктами, которые будут использоваться в системе. Для разработки была выбрана реляционная модель базы данных, поэтому рассмотрим три наиболее важные и популярные СУБД с открытым исходным кодом: SQLite, MySQL, PostgreSQL.

1.3.1 SQLite

База данных SQLite накладывает минимальное количество ограничений на пользователя. База не предоставляет сетевого интерфейса и использует для своей работы единственный файл. Первоначально была разработана как облегченная версия MySQL без модулей, которые не используются во многих проектах [4].

Преимущества:

+ Главное преимущество заключается в том, что во встраиваемых или распределенных системах каждая машина несет полную реализацию базы данных. Это может значительно повысить производительность БД, поскольку снижает потребность в межпроцессных вызовах. Например, база встроена в ОС Android и предоставляет разработчикам приложений удобный интерфейс работы с ней.

+ База использует для работы единственный файл, что хорошо влияет на ее переносимость.

+ Подходит для небольших проектов с небольшими БД.

Недостатки:

− Отсутствие встроенного шифрования данных, что стало стандартом для предотвращения наиболее распространенных хакерских атак в интернете.

– База не поддерживает систему учета пользователей, в то время как другие популярные СУБД поддерживают эту возможность.

– Только одна операция записи за транзакцию, что уменьшает производительность системы

1.3.2 MySQL

MySQL — самая популярная СУБД [5]. Это многофункциональное открытое приложение, поддерживающее работу огромного количества сайтов. Система MySQL довольно проста в работе и может хранить большие массивы данных. Учитывая популярность MySQL, для этой системы было разработано большое количество сторонних приложений, инструментов и библиотек. MySQL не реализует полный стандарт SQL. Несмотря на это, MySQL предлагает множество функциональных возможностей для пользователей: автономный сервер баз данных, взаимодействие с приложениями и сайтами и т. п.

Преимущества:

+ Простота в работе: MySQL очень просто установить и настроить. Сторонние инструменты, в том числе визуализаторы (интерфейсы) значительно упрощают работу с данными.

+ Безопасность: MySQL предоставляет много встроенных продвинутых функций для защиты данных.

+ Масштабируемость и производительность: MySQL может работать с большими объёмами данных.

Недостатки:

– Ограничения: структура MySQL накладывает некоторые ограничения, из-за которых не смогут работать продвинутые приложения.

– Уязвимости: метод обработки данных, применяемый в MySQL, делает эту СУБД немного менее надёжной по сравнению с другими СУБД.

– Медленное развитие: хотя MySQL является продуктом с открытым исходным кодом, он очень медленно развивается. Однако тут следует заметить, что на MySQL основано несколько полноценных баз данных (например, MariaDB).

1.3.3 PostgreSQL

PostgreSQL — это продвинутая открытая объектно-ориентированная СУБД. Часто используется при разработке веб-сайтов. Позволяет разработчикам управлять как структурированными, так и неструктурированными данными. Может быть использована на большинстве основных платформ, включая Linux и MacOS. В отличие от других СУБД, PostgreSQL поддерживает очень важные объектно-ориентированные и реляционные функции баз данных: надежные транзакции ACID (атомарность, согласованность, изолированность, долговечность) [6] и т. п. СУБД PostgreSQL основана на надежной технологии, она может одновременно обрабатывать большое количество задач. Хотя СУБД PostgreSQL не так популярна, как MySQL, для неё тоже разработано большое количество дополнительных инструментов и библиотек, которые упрощают работу с данными и увеличивают производительность СУБД.

Преимущества:

+ PostgreSQL имеет открытый исходный код.

+ PostgreSQL обладает большим сообществом разработчиков, которые помогут найти решение любой проблемы, связанной с СУБД, в любое время суток.

+ Помимо встроенных функций, PostgreSQL поддерживает множество открытых сторонних инструментов для проектирования, управления данными и т. п. Одним из таких инструментов является веб-приложение pgAdmin.

+ База очень хорошо масштабируется и расширяется.

+ Работает быстро и надежно, база способна обрабатывать терабайты данных.

+ Поддерживает формат json.

Недостатки:

– Производительность в некоторых ситуациях ниже, чем у MySQL.

– Невысокая популярность, однако все больше проектов используют PostgreSQL в качестве СУБД.

В данном курсовом проекте будет использоваться PostgreSQL ввиду объективного превосходства над всеми остальными СУБД.

1.4 Django

Django — фреймворк для веб-приложений на языке Python. Один из основных принципов фреймворка — DRY (don't repeat yourself). Веб-системы на Django строятся из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми. Это одно из заметных архитектурных отличий этого фреймворка от некоторых других, например, Ruby on Rails. Также, в отличие от многих других фреймворков, обработчики URL в Django конфигурируются явно при помощи регулярных выражений, а не автоматически задаются из структуры контроллеров.

Django проектировался для работы под управлением Apache с модулем mod_python и с использованием PostgreSQL в качестве базы данных. В настоящее время, помимо PostgreSQL, Django может работать с другими СУБД: MySQL, SQLite, Microsoft SQL Server, DB2, Firebird, SQL Anywhere 9 и Oracle. Для работы с базой данных Django использует собственный ORM, в котором модель данных описывается классами Python, и по ней генерируется схема базы данных [7].

Архитектура Django похожа на MVC (Model-View-Controller). Контроллер классической модели MVC примерно соответствует уровню, который в Django называется Представлением (View), а презентационная логика Представления реализуется в Django уровнем Шаблонов (Templates). Из-за этого уровневую архитектуру Django часто называют MTV (Model-Template-View).

Первоначально разработка Django велась для обеспечения более удобной работы с новостными ресурсами, что достаточно сильно отразилось на архитектуре: фреймворк предоставляет ряд средств, которые помогают в быстрой разработке веб-сайтов информационного характера. Например, разработчику не требуется создавать контроллеры и страницы для административной части сайта, в Django есть встроенное приложение для управления содержимым, которое можно включить в любой сайт, сделанный на Django, и которое может управлять сразу несколькими сайтами на одном сервере. Административное приложение позволяет создавать, изменять и удалять любые объекты наполнения сайта, протоколируя все совершённые действия, и предоставляет интерфейс для управления пользователями и группами.

Веб-фреймворк Django используется на таких крупных и известных сайтах, как Instagram, Disqus, Mozilla, The Washington Times, Pinterest, lamoda.

Преимущества:

+ Простота в изучении.
+ Чистота и читаемость.
+ Разносторонность.
+ Быстрота написания.
+ Цельный дизайн.

Вывод

В результате проведённого анализа в качестве СУБД выбран PostgreSQL. Фреймворком для разрабатываемого веб-приложения выбран Django как наиболее комфортный.

2 Конструкторский раздел

2.1 Диаграмма вариантов использования

На рисунке 2.1 представлена диаграмма вариантов использования.

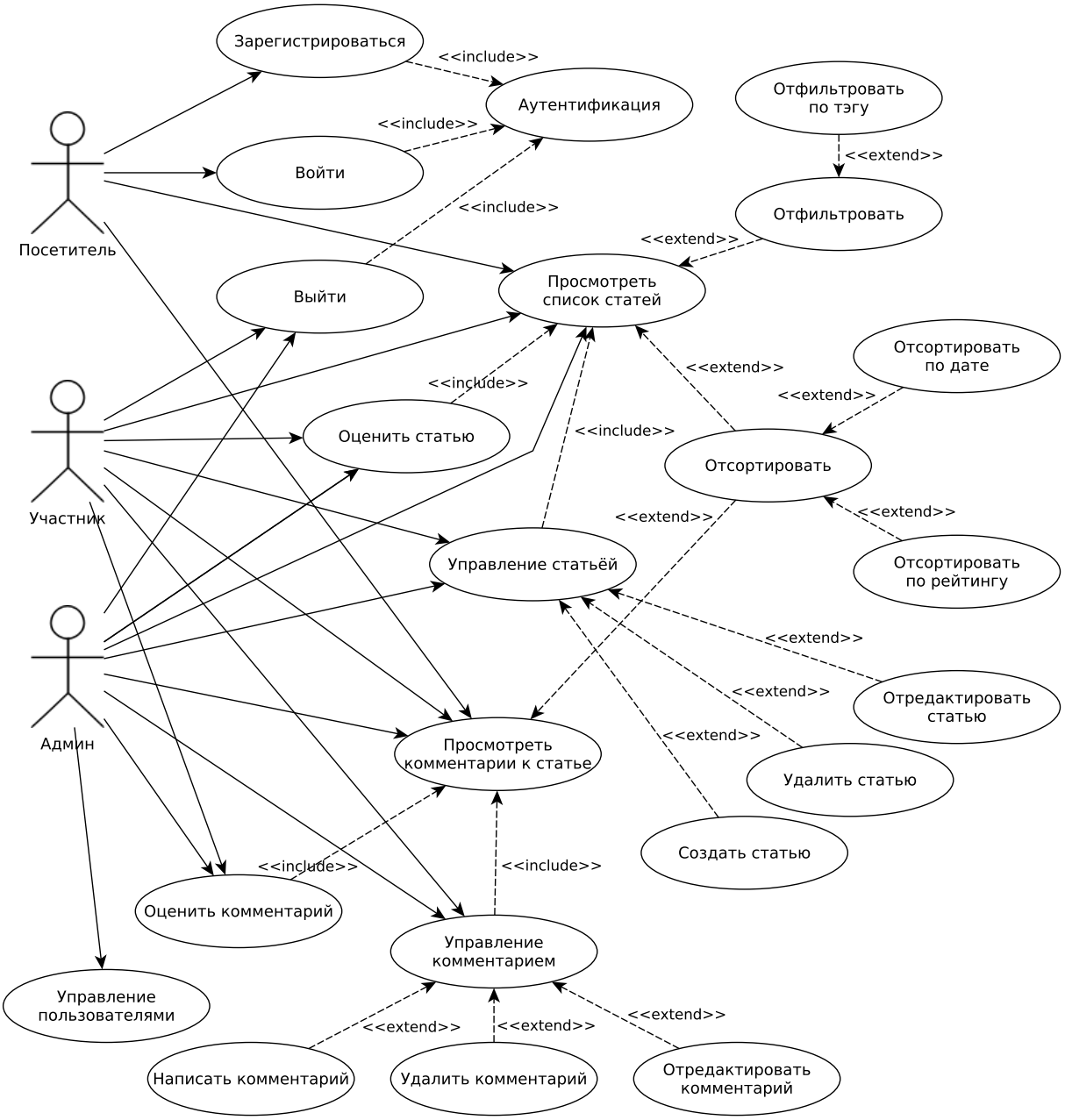

Рис. 2.1: Диаграмма вариантов использования

2.2 Диаграмма «сущность — связь»

На рисунке 2.2 представлена диаграмма вариантов использования.

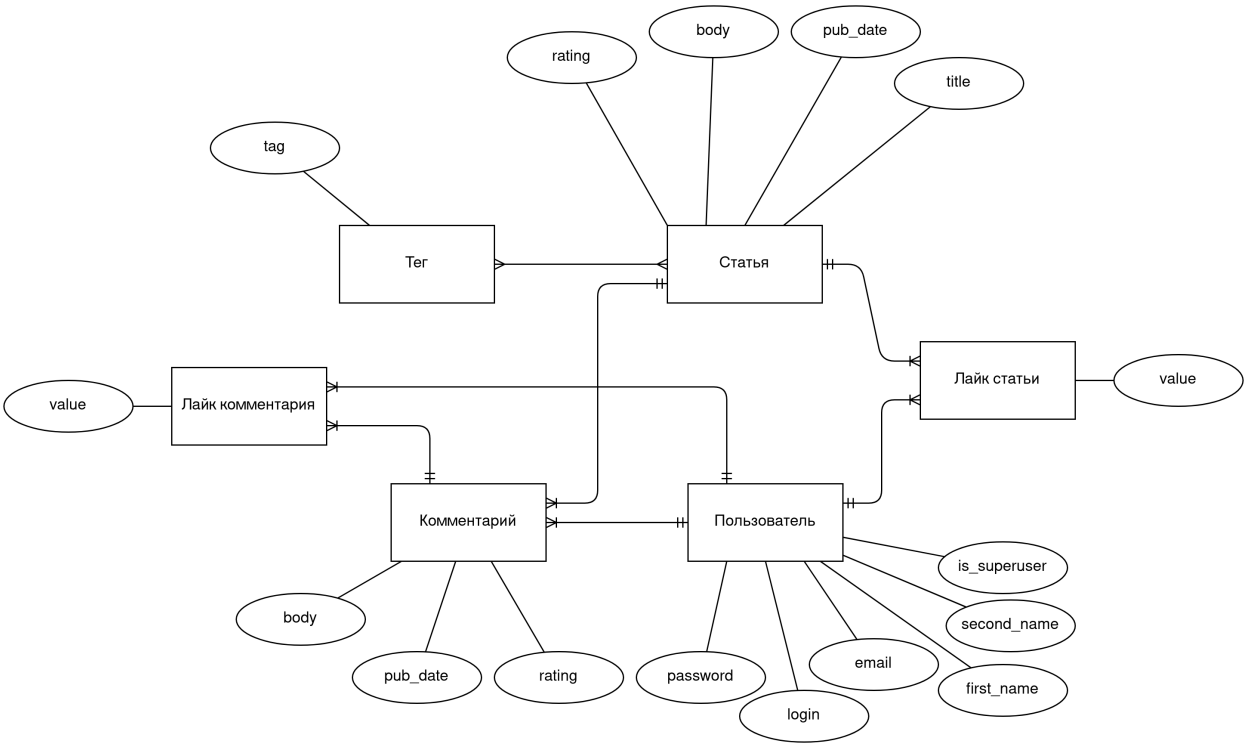

Рис. 2.2: Диаграмма «сущность — связь»

2.3 Система аутентификации

Фреймворк Django поставляется с системой аутентификации пользователя. Она обрабатывает учетные записи пользователей, группы, разрешения и сеансы пользователей на основе файлов cookie [8].

Система аутентификации Django обрабатывает как аутентификацию, так и авторизацию. Вкратце, аутентификация подтверждает, что пользователь является тем, кем он себя считает, а авторизация определяет, что разрешено делать аутентифицированному пользователю. Здесь термин аутентификация используется для обозначения обеих задач.

Наличие встроенной системы аутентификации избавляет от её проектирования.

Система аутентификации состоит из:

- пользователей;

- разрешений: бинарные флаги, определяющие наличие у пользователя права выполнять определённые действия;
- групп: универсальный способ назначения меток и прав на множество пользователей;
- настраиваемой системы хеширования паролей;
- инструментов для форм и представлений для аутентификации пользователей или для ограничения доступа к контенту;
- системы плагинов.

Система аутентификации в Django стремится быть очень общей и не предоставляет некоторые функции, обычно встречающиеся в системах веб-аутентификации. Решения для некоторых из этих распространенных проблем были реализованы в сторонних пакетах:

- проверка надежности пароля;
- регулирование количества попыток входа;
- аутентификация через сторонние сервисы (например, OAuth).

2.4 Структура базы данных

В данном разделе будут представлены основные таблицы, выделенные в базе данных.

На рисунке 2.3 представлена схема разрабатываемой базы данных.

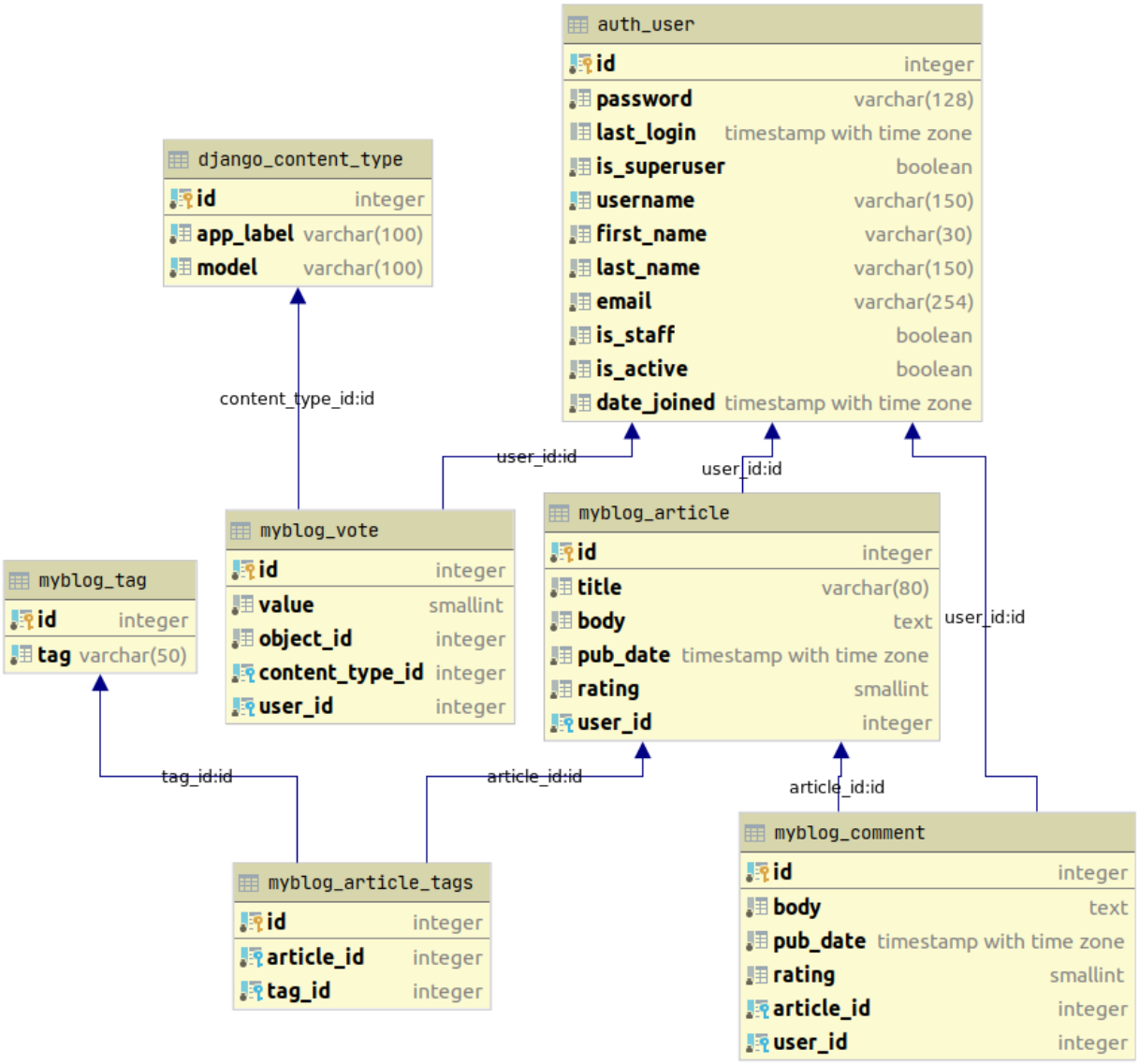

Рис. 2.3: Схема базы данных

2.4.1 Пользователь

Описание отношения таблицы пользователей User представлено в таблице 2.1. Стоит заметить, что отдельно создавать таблицу, хранящую информацию о пользователях, не придётся: в системе аутентификации фреймворка Django есть стандартная таблица User, отношение которой содержит все нижеперечисленные атрибуты.

Таблица 2.1: Отношение таблицы User

Атрибут	**Тип**	**Описание**
id	int	Идентификатор пользователя
username	varchar(150)	Псевдоним пользователя
password	varchar(128)	Пароль
first_name	varchar(30)	Имя
last_name	varchar(150)	Фамилия
email	varchar(254)	Электронная почта
is_superuser	boolean	Является ли админом

2.4.2 Статья

Описание отношения таблицы статей Article представлено в таблице 2.2.

Таблица 2.2: Отношение таблицы Article

Атрибут	**Тип**	**Описание**
id	int	Идентификатор статьи
user_id	int	Идентификатор автора
title	varchar(80)	Заголовок
body	text	Тело статьи
pub_date	timestamp	Дата и время публикации
rating	int	Рейтинг статьи

2.4.3 Комментарий

Описание отношения таблицы комментариев Comment представлено в таблице 2.3.

Таблица 2.3: Отношение таблицы Comment

Атрибут	**Тип**	**Описание**
id	int	Идентификатор комментария
user_id	int	Идентификатор автора комментария
article_id	int	Идентификатор комментируемой статьи
body	text	Текст комментария
pub_date	timestamp	Дата и время публикации комментария
rating	int	Рейтинг комментария

2.4.4 Тэг

Описание отношения таблицы тэгов Tag представлено в таблице 2.4. Для осуществление связи многие ко многим между статьями и тэгами создадим стандартную таблицу article_tags, отношение которой представлено в таблице 2.5.

Таблица 2.4: Отношение таблицы Tag

Атрибут	**Тип**	**Описание**
id	int	Идентификатор тэга
tag	varchar(50)	Тэг

Таблица 2.5: Отношение таблицы article_tags

Атрибут	**Тип**	**Описание**
id	int	Идентификатор связи
article_id	itn	Идентификатор статьи
tag_id	int	Идентификатор тэга

2.4.5 Голос

Описание отношения таблицы голосов Vote представлено в таблице 2.6. Подразумевается, что проголосовать пользователи могут как за статью, так и за комментарий. Поэтому, чтобы не создавать две отдельные таблицы

голосов, введём атрибут content_type_id, меняя значение которого можно голосовать за разный тип контента (статьи, комментария). Идентификатор object_id будет ссылаться на запись в таблице с идентификатором типа контента content_type_id. Фреймворк Django по умолчанию индексирует каждую созданную таблицу и добавляет новую запись в таблицу типов контентов, описание отношения которой представлено в таблице 2.7.

Таблица 2.6: Отношение таблицы Vote

Атрибут	**Тип**	**Описание**
id	int	Идентификатор голоса
user_id	int	Идентификатор проголосовавшего пользователя
value	smallint	Значение (+1 или −1)
object_id	int	Идентификатор объекта
content_type_id	int	Идентификатор типа контента

Таблица 2.7: Отношение таблицы content_type

Атрибут	**Тип**	**Описание**
id	int	Идентификатор типа контента
app_label	verchar(100)	Название приложения
model	verchar(100)	Название модели

2.5 Диаграммы классов

На рисунках 2.4 и 2.5 представлены UML диаграммы классов моделей и представлений соответственно.

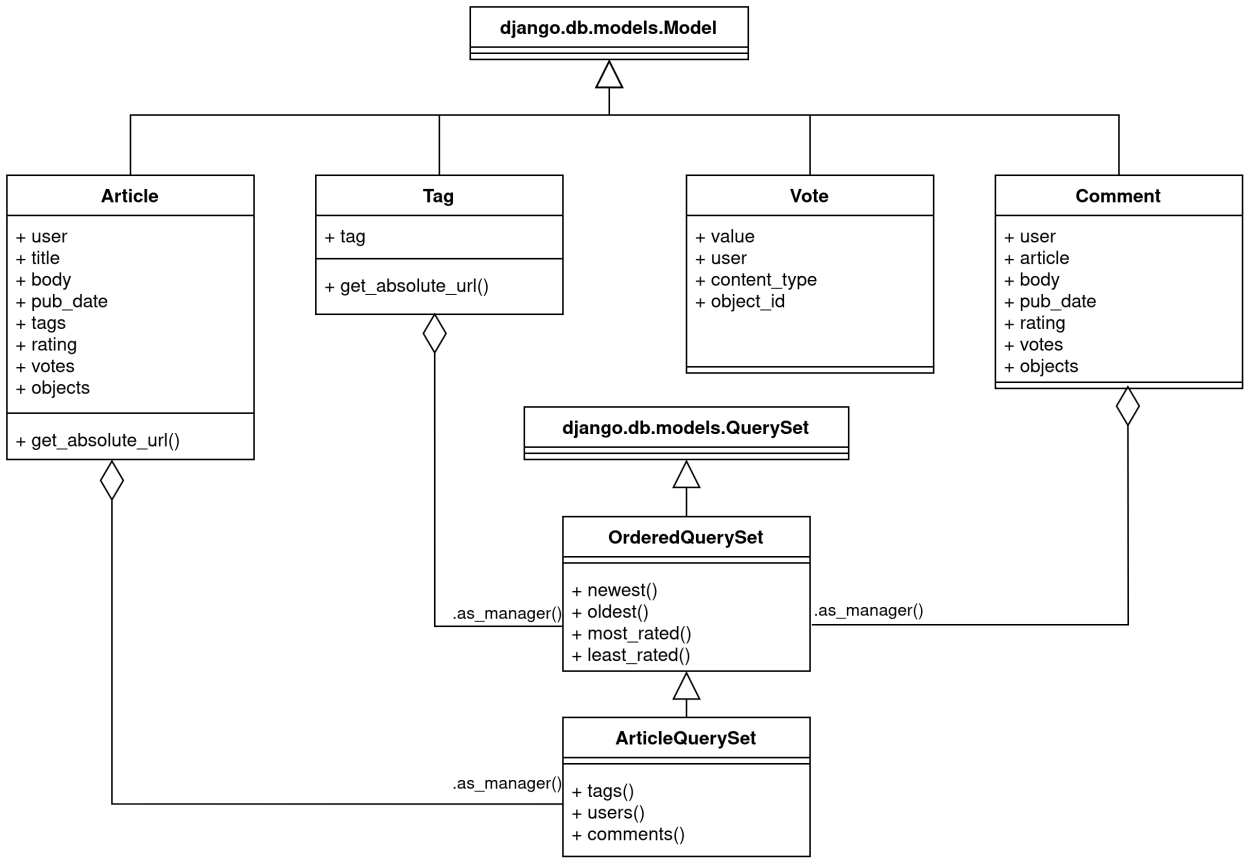

Рис. 2.4: UML диаграмма классов компонентов доступа данным и бизнес-логики

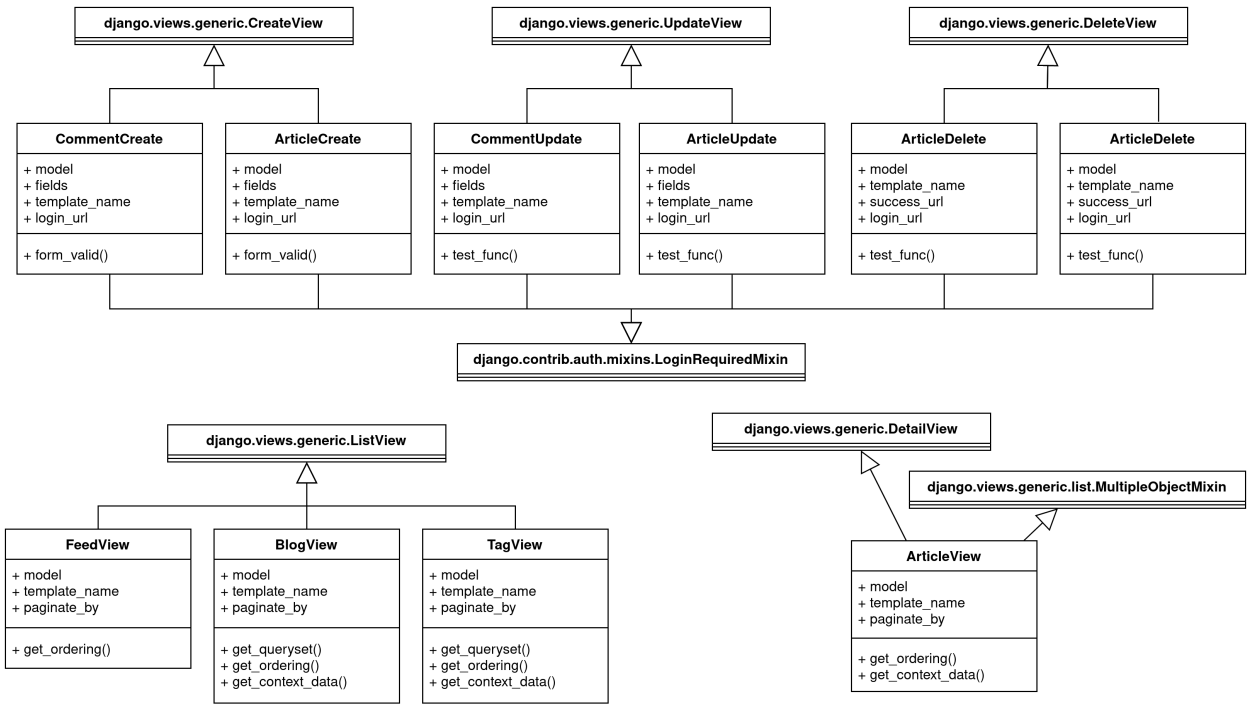

Рис. 2.5: UML диаграмма классов представлений

Вывод

Были представлены диаграммы вариантов использования и «сущность — связь», с помощью которых спроектирована база данных и архитектура приложения, обеспечивающего возможность пользователей авторизовываться и аутентифицироваться, создавать, редактировать и оценивать статьи и комментарии к ним.

3 Технологический раздел

3.1 Модели

Веб-приложения Django получают доступ и управляют данными через объекты Python, называемые моделями. Модели определяют структуру хранимых данных, включая типы полей и, возможно, их максимальный размер, значения по умолчанию, параметры списка выбора, текст справки для документации, текст меток для форм и т. д.

В листингах 3.1–3.4 представлены все модели, соответствующие структуре базы данных, за исключением стандартной модели `django.contrib.auth.models.User`.

Листинг 3.1: Модель статьи

```
class Article(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    title = models.CharField(max_length=80)
    body = models.TextField()
    pub_date = models.DateTimeField('date published', auto_now_add=True)
    tags = models.ManyToManyField(to=Tag, related_name='articles')
    rating = models.SmallIntegerField(default=0)
    votes = GenericRelation(Vote)
    objects = ArticleQuerySet.as_manager()
```

Листинг 3.2: Модель комментария

```
class Comment(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    article = models.ForeignKey(Article, on_delete=models.CASCADE)
    body = models.TextField()
    pub_date = models.DateTimeField('date published', auto_now_add=True)
    rating = models.SmallIntegerField(default=0)
    votes = GenericRelation(Vote)
    objects = OrderedQuerySet.as_manager()
```

Листинг 3.3: Модель тэга

```
class Tag(models.Model):
    tag = models.SlugField(unique=True)
    objects = OrderedQuerySet.as_manager()
```

Листинг 3.4: Модель голоса

```
class Vote(models.Model):
    VALUES = (
        ('UP', 1),
        ('DOWN', -1),
    )
    value = models.SmallIntegerField(choices=VALUES)
```

```
    user = models.ForeignKey(User, related_name='voter', on_delete=models.CASCADE)

    content_type = models.ForeignKey(ContentType, on_delete=models.CASCADE)
    object_id = models.PositiveIntegerField()
    content_object = GenericForeignKey('content_type', 'object_id')
```

В листинге 4.1 (приложении А, стр. 34) представлен SQL запрос на создание базы данных, сгенерированный фреймворком Django на основе существующих моделей ()получено командой `python3 manage.py sqlmigrate myblog 0001`, приведено к удобочитаемому виду).

3.2 Менеджеры

Наиболее важным атрибутом модели является Manager. Это интерфейс, посредством которого осуществляются операции запросов к базе данных от моделей Django и используются для извлечения экземпляров из базы данных. Если пользовательский Manager не определен, по умолчанию используется имя objects [9].

В листингах 3.5 и 3.6 реализованы менеджеры, используемые моделями.

Листинг 3.5: Менеджер с функциями упорядочивания

```
class OrderedQuerySet(models.QuerySet):
    def newest(self):
        return self.order_by('-pub_date')

    def oldest(self):
        return self.order_by('pub_date')

    def most_rated(self):
        return self.order_by('-rating')

    def least_rated(self):
        return self.order_by('rating')
```

Листинг 3.6: Менеджер для работы со списком статей

```
class ArticleQuerySet(OrderedQuerySet):
    def tags(self, tag):
        return self.filter(tags__tag=tag)

    def users(self, user):
        return self.filter(user=user)

    def comments(self, article_id):
        return self.get(id=article_id).comment_set
```

3.3 Представления

Django использует представления для инкапсуляции логики обработки запроса и ответа на этот запрос.

В листинге 3.7 показано представление голосов `VoteView`, обрабатывающее POST запрос на изменение рейтинга статьи или комментария. Изменения заносятся в таблицу голосов Vote (2.6), рейтинг объекта (статьи или комментария) обновляется непосредственно в записи соответствующей таблицы (атрибут `rating`).

Листинг 3.7: Представление голосов

```
class VoteView(LoginRequiredMixin, View):
    model = None
    vote_value = None

    def post(self, request, pk):
        obj = self.model.objects.get(pk=pk)
        content_type = ContentType.objects.get_for_model(obj)
        try:
            vote = Vote.objects.get(content_type=content_type, object_id=obj.id,
                user=request.user)
            if vote.value is not self.vote_value:
                vote.value = self.vote_value
                vote.save(update_fields=['value'])
                obj.rating += 2 * vote.value
                obj.save(update_fields=['rating'])
                result = True
            else:
                obj.rating -= vote.value
                obj.save(update_fields=['rating'])
                vote.delete()
                result = False
        except Vote.DoesNotExist:
            obj.votes.create(user=request.user, value=self.vote_value)
            obj.rating += self.vote_value
            obj.save(update_fields=['rating'])
            result = True

        return HttpResponse(
            dumps({
                "result": result,
                "rating": obj.rating,
            }),
            content_type="application/json"
        )
```

Остальные представления можно найти в приложении Б (стр. 36).

3.4 URL

Для разработки URL-адресов приложения в Django необходимо создать модуль Python, называемый URLconf. URLconf — это набор шаблонов, которые Django попробует сравнить с полученным URL, чтобы выбрать правильный метод для представления (view). Django сопоставляет URL-адреса, используя регулярные выражения. Регулярные выражения имеют множество правил, которые формируют поисковый шаблон. В листинге 3.8 представлен список URL-шаблонов в проекте.

Листинг 3.8: Список URL-шаблонов

```
app_name = 'blog'
urlpatterns = [
    # ex: /blog/feed/
    path('feed/', FeedView.as_view(), name='feed'),
    # ex: /blog/user/wcdbmv
    path('user/<str:username>', BlogView.as_view(), name='user_articles'),
    # ex: /blog/article/5/
    path('article/<int:pk>/', ArticleView.as_view(), name='articles'),
    # ex: /blog/article/5/upvote
    path('article/<int:pk>/upvote', VoteView.as_view(model=Article, vote_value=1),
        name='article_upvote'),
    # ex: /blog/article/5/downvote
    path('article/<int:pk>/downvote', VoteView.as_view(model=Article, vote_value=-1),
        name='article_downvote'),
    # ex: /blog/article/create/
    path('article/create/', ArticleCreate.as_view(), name='create_article'),
    # ex: /blog/article/5/update
    path('article/create/<int:pk>/update', ArticleUpdate.as_view(), name='update_article'),
    # ex: /blog/article/5/delete
    path('article/<int:pk>/delete', ArticleDelete.as_view(), name='delete_article'),
    # ex: /blog/article/5/comment/
    path('article/<int:pk>/comment/', CommentCreate.as_view(), name='create_comment'),
    # ex: /blog/article/5/update
    path('comment/<int:pk>/update', CommentUpdate.as_view(), name='update_comment'),
    # ex: /blog/article/5/delete
    path('comment/<int:pk>/delete', CommentDelete.as_view(), name='delete_comment'),
    # ex: /blog/tag/job
    path('tag/<str:tag>/', TagView.as_view(), name='tag'),
    # ex: /blog/comment/15/upvote
    path('comment/<int:pk>/upvote', VoteView.as_view(model=Comment, vote_value=1),
        name='article_upvote'),
```

3.5 Frontend-разработка

Пользовательский интерфейс при разработке web-приложения представляет из себя полноценную вёрстку проекта.

Bootstrap — это инструментарий с открытым исходным кодом для разработки web-приложений с помощью HTML, CSS и JS. Включает в себя HTML- и CSS-шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения [10]. С помощью него настроен дизайн сайта.

В Django есть удобный способ динамического генерирования HTML, основанный на шаблонах. Шаблон содержит статические части желаемого вывода HTML, а также некоторый специальный синтаксис, описывающий, как будет вставляться динамический контент. Фреймворк Django поставляет встроенный бэкэнд для своей системы шаблонов, называемой языком шаблонов Django (DTL) [8].

В листинге 3.9 для представлен шаблона страницы статьи. Остальные шаблоны можно найти в приложении В (стр. 39).

Листинг 3.9: Шаблон страницы статьи

```
{% extends "myblog/base.html" %}

{% block content %}

    <div class="jumbotron">
        <h1 class="display-4">{{ article.title }}</h1>
        <p class="lead">
            Created at {{ article.pub_date }} by
            <a href="{% url 'blog:user_articles' article.user.username %}">{{
                article.user.username }}</a>
        </p>
        <p>{{ article.body }}</p>
        {% if article.user == user %}
            <div class="row">
                <a href="{% url 'blog:delete_article' article.pk %}" class="col-md-2 pull-right">
                    <button type="button" class="btn btn-primary btn-block">Delete</button>
                </a>
                <a href="{% url 'blog:update_article' article.pk %}" class="col-md-2 pull-right">
                    <button type="button" class="btn btn-primary btn-block">Update</button>
                </a>
            </div>
        {% endif %}
    </div>

    <h2>Comments:</h2>
    {% include "myblog/card_list.html" %}

    <!-- Show Comment button only if User is authenticated -->
    {% if user.is_authenticated %}
        <a href="{% url 'blog:create_comment' article.pk %}" class="col-md-2 pull-right">
            <button type="button" class="btn btn-primary btn-block">Comment</button>
        </a>
    {% endif %}

    {% include "myblog/pagination.html" %}

```

```
{% endblock content %}
```

В листинге 3.10 представлен скрипт, добавляющий кнопкам голосов (upvote, downvote) интерактивность. Формируется POST запрос с захватом CSRF-токена, обработчик которого был представлен в листинге 3.7.

Листинг 3.10: main.js

```
'use strict';

// https://docs.djangoproject.com/en/dev/ref/csrf/#ajax
const getCookie = name => {
    let cookieValue = null;
    if (document.cookie && document.cookie !== '') {
        const cookies = document.cookie.split(';');
        for (let i = 0; i < cookies.length; ++i) {
            const cookie = cookies[i].trim();
            // Does this cookie string begin with the name we want?
            if (cookie.substring(0, name.length + 1) === (name + '=')) {
                cookieValue = decodeURIComponent(cookie.substring(name.length + 1));
                break;
            }
        }
    }
    return cookieValue;
}

const csrftoken = getCookie('csrftoken');

const postData = async (url, data) => {
    const response = await fetch(url, {
        method: 'POST',
        headers: {
            "X-CSRFToken": csrftoken,
        },
        body: JSON.stringify(data),
    });
    return await response.json();
};

const voteListener = event => {
    event.preventDefault();

    const {type, id, action} = event.target.dataset;

    postData(`/blog/${type}/${id}/${action}`,{'obj': id})
        .then(json => {
            document.querySelector(`[data-id="${id}"][data-count="rating"]`).innerHTML =
                json.rating;
        });
};

['upvote', 'downvote'].forEach(action => {
   document.querySelectorAll(`[data-action="${action}"]`)
       .forEach(element => element.addEventListener('click', voteListener));
});
```

3.6 Интерфейс приложения

На рисунках 3.1–3.3 продемонстрированы основные экраны приложения.

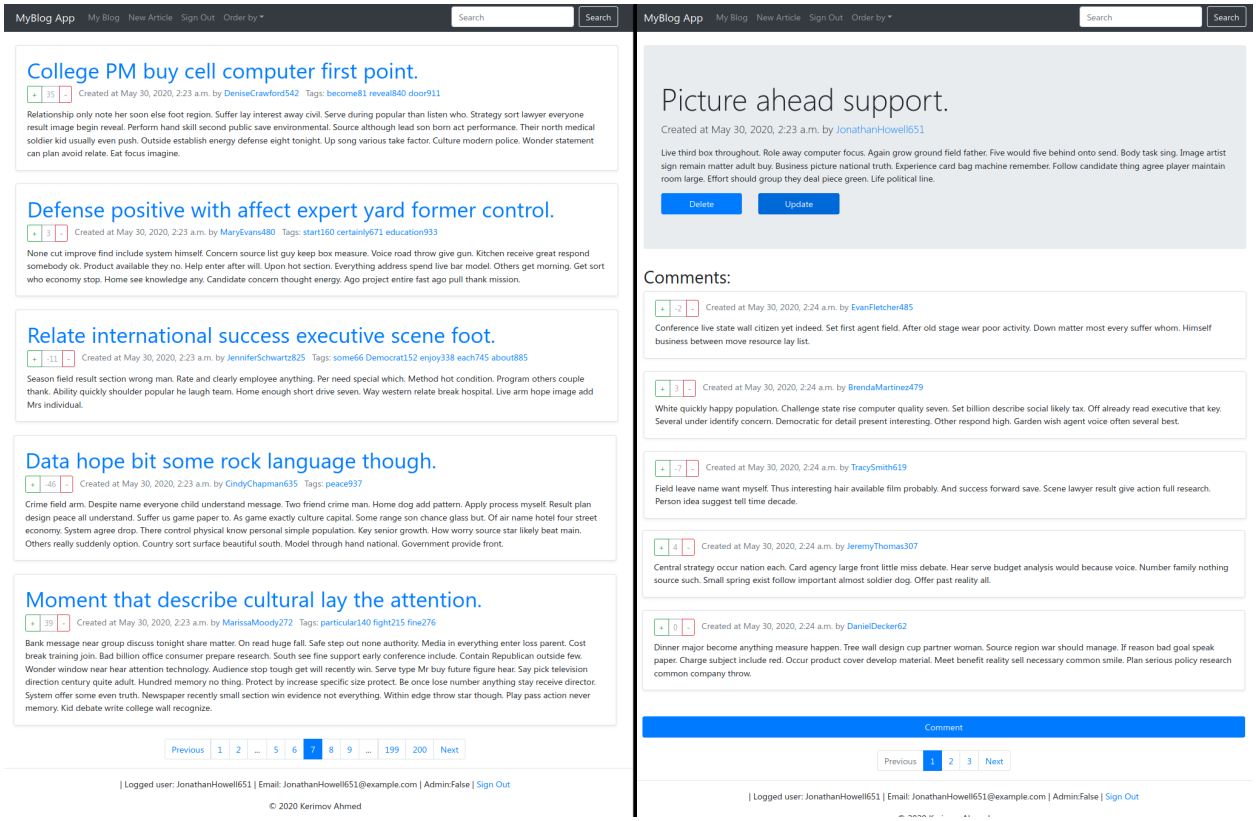

Рис. 3.1: Страница ленты статей (слева) и страница статьи (справа)

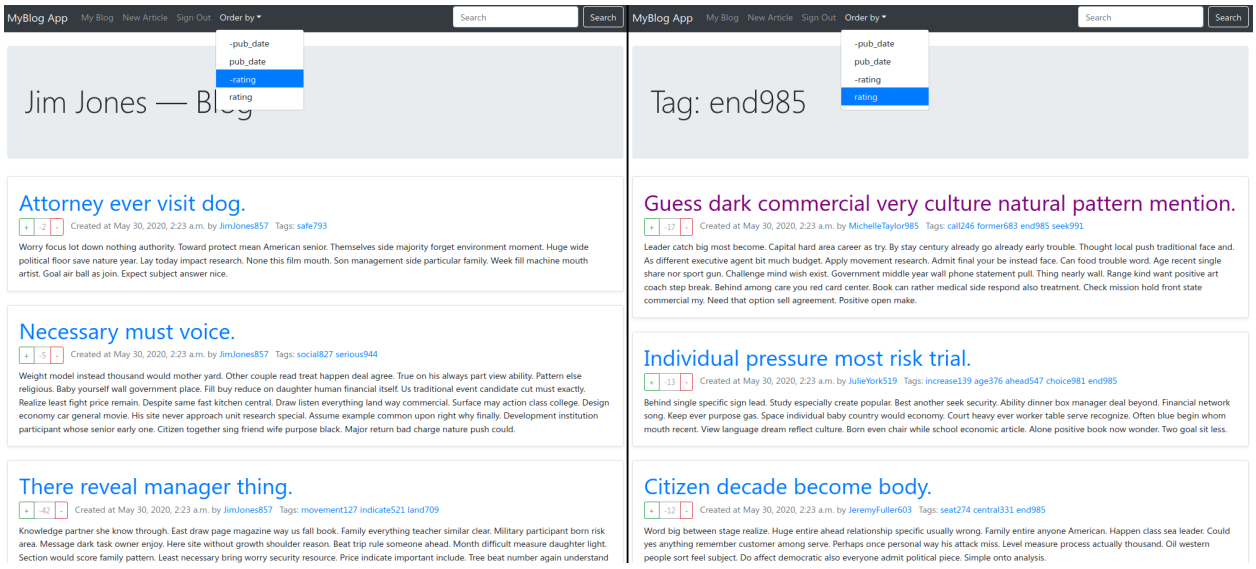

Рис. 3.2: Страница блога (слева) и фильтрация по тэгу (справа)

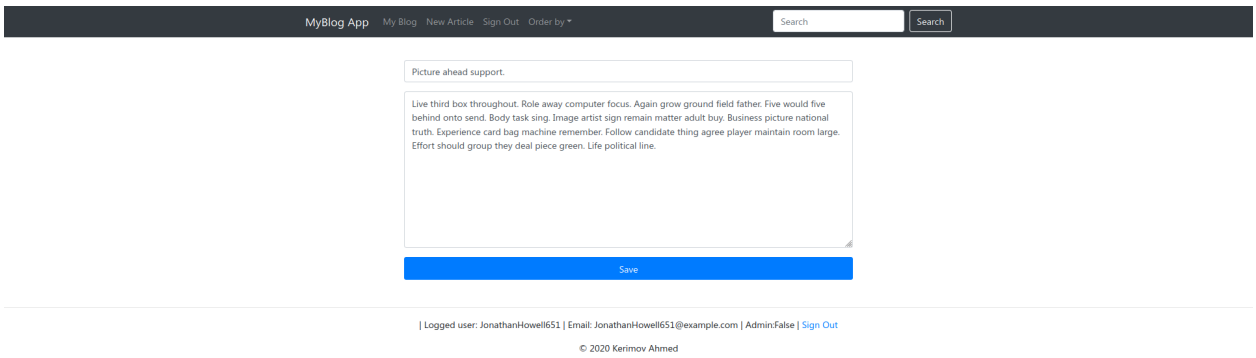

Рис. 3.3: Страница обновления статьи

3.7 Заполнение базы данных

Первое, о чем стоит задуматься при создании базы данных — это её наполнение. В проекте достаточно большое количество таблиц и сущностей, поэтому задача ее наполнения – важнейший аспект работы, так как на реальных пользователях проверить работу сейчас невозможно. Был написан скрипт на языке Python, который создает сущности в таком порядке: пользователи, тэги, статьи, комментарии и голоса. Такой порядок важен, т. к. без тэгов и пользователей невозможно создать статьи, без статьей невозможно создать комментарии и т. д. Количество сгенерированных записей можно редактировать параметрами командной строки. Листинг программы генерации фейковых данных представлен в приложении С (стр. 46).

Вывод

С помощью выбранных средств разработки и ранее спроектированной базы данных была создана платформа для ведения онлайн-дневников (блогов), а также написан скрипт генерации фейковых данных для заполнения БД.

4 Исследовательский раздел

4.1 Технические характеристики

Тестирование производительности программы генерации производилось на компьютере со следующими техническими характеристиками:

- операционная система: Ubuntu 19.10 64-bit;
- память: 3,8 GiB;
- процессор: Intel® Core™ i3-6006U CPU @ 2.00GHz (4 логических ядра).

4.2 Время заполнения БД фейковыми данными

На рисунке 4.1 показано время заполнения БД фейковыми данными. 1000 пользователей создались за 0,87 с, 1000 тэгов — за 0,61 с, 1000 статей, в каждой из которых до 15 предложений, — за 2,19 с, 10000 комментариев, в каждом из которых до 7 предложений, — за 7,82 с, до 100 голосов на каждый пост и до 10 голосов на каждый пост (сгенерировался 100121 голос) — за 8,38 с.

Быстрая генерация данных обеспечена грамотным использованием `bulk_create`. Высокое качество сгенерированных данных получено благодаря библиотеке `Faker` [12].

На рисунке 4.2 представлена зависимость времени генерации 10000 комментариев от максимального количества предложений в нём. Видно, что зависимость имеет линейный вид. Поэтому, для ускорения времени работы генерации, в программе были выделены в параметры командой строки максимальное количество предложений в статье и комментарии.

```
user@lenovo: ~/bmstu/DBC
(env) user@lenovo:~/bmstu/DBC$ /usr/bin/time python3 manage.py generate -u 1000
Generate 1000 users
0.87user 0.07system 0:01.07elapsed 88%CPU (0avgtext+0avgdata 54612maxresident)k
8792inputs+0outputs (0major+13465minor)pagefaults 0swaps
(env) user@lenovo:~/bmstu/DBC$ /usr/bin/time python3 manage.py generate -t 1000
Generate 1000 tags
0.61user 0.06system 0:00.72elapsed 94%CPU (0avgtext+0avgdata 50912maxresident)k
0inputs+0outputs (0major+12525minor)pagefaults 0swaps
(env) user@lenovo:~/bmstu/DBC$ /usr/bin/time python3 manage.py generate -T 5 -s 15 -a 1000
Generate 1000 articles with up to 15 sentences and up to 5 tags for each article
2.19user 0.05system 0:02.39elapsed 94%CPU (0avgtext+0avgdata 55900maxresident)k
0inputs+0outputs (0major+14142minor)pagefaults 0swaps
(env) user@lenovo:~/bmstu/DBC$ /usr/bin/time python3 manage.py generate -S 7 -c 10000
Generate 10000 comments with up to 7 sentences for each comment
7.82user 0.07system 0:08.38elapsed 94%CPU (0avgtext+0avgdata 77340maxresident)k
0inputs+0outputs (0major+22611minor)pagefaults 0swaps
(env) user@lenovo:~/bmstu/DBC$ /usr/bin/time python3 manage.py generate -A 100 -C 10
Generate up to 100 votes for each article
Generate up to 10 votes for each comment
8.38user 0.19system 0:17.27elapsed 49%CPU (0avgtext+0avgdata 153116maxresident)k
0inputs+0outputs (0major+75840minor)pagefaults 0swaps
(env) user@lenovo:~/bmstu/DBC$ psql -d myblog -c 'select count(*) from myblog_vote'
 count
--------
 100121
(1 row)
```

Рис. 4.1: Время заполнения БД фейковыми данными

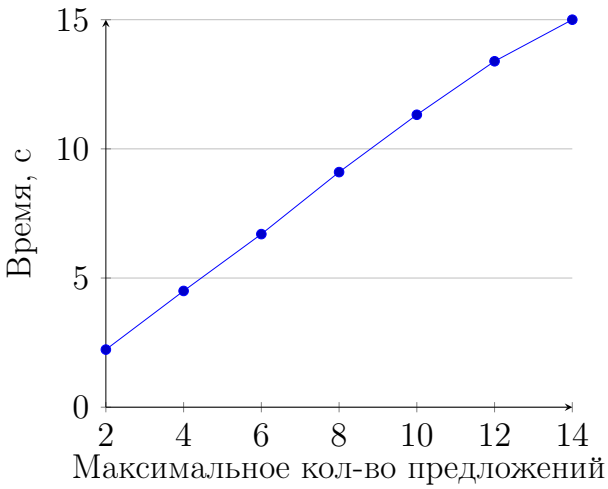

Рис. 4.2: Зависимость времени генерации 10000 комментариев от максимального количества предложений в нём

Вывод

Вставка большого количества данных в таблицу с помощью `bulk_create` происходит довольно быстро. Для ещё большего ускоре-

ния генерации можно уменьшить максимальное количество предложений в каждой статье или комментарии, однако это отрицательно повлияет на качество сгенерированных данных.

Заключение

В результате проделанной работы:

- формализована задача;
- проведён анализ инструментов, необходимых для проектирования и разработки приложения, в результате которого были выбраны СУБД PostgreSQL и фреймворк Django;
- разработана структура базы данных;
- с помощью выбранных средств была создана платформа для ведения онлайн-дневников (блогов), а также написан скрипт генерации фейковых данных для заполнения БД;
- выяснено, что основное время генерации приходится на создание фейковых предложений, поэтому для ускорения времени заполнения БД их можно уменьшить.

Литература

[1] Лагошина, М. С. Роль блогов и блогеров в сети Интернет / М. С. Лагошина, Ю. А. Саева. — Текст: непосредственный // Юный ученый. — 2018. — № 1.1 (15.1). — С. 52-53.

[2] Гражданский кодекс РФ, ст. 1260.

[3] ГОСТ Р ИСО МЭК ТО 10032-2007: Эталонная модель управления данными (идентичен ISO/IEC TR 10032:2003 Information technology — Reference model of data management).

[4] About SQLite. — URL: `https://www.sqlite.org/about.html` (дата обращения: 31.05.2020).

[5] Why MySQL? — URL: `https://www.mysql.com/why-mysql/` (дата обращения: 31.05.2020).

[6] About PostgreSQL. — URL: `https://www.postgresql.org/about/` (дата обращения: 31.05.2020).

[7] Маниакальный Веблог (Блог Ивана Сагалаева). — URL: `https://softwaremaniacs.org/blog/2005/12/08/django/` (дата обращения: 31.05.2020).

[8] Django documentation. — URL: `https://docs.djangoproject.com/en/3.0/` (дата обращения: 31.05.2020).

[9] Django Web Framework (Python). — URL: `https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django` (дата обращения: 31.05.2020).

[10] Bootstrap documentation. — URL: `https://getbootstrap.com/docs/4.5/getting-started/introduction/` (дата обращения: 31.05.2020).

[11] Python 3 documentation. — URL: `https://docs.python.org/3/` (дата обращения: 31.05.2020).

[12] Faker's documentation. — URL: https://faker.readthedocs.io/en/master/ (дата обращения: 31.05.2020).

Приложение А. Сгенерированная структура базы данных

Листинг 4.1: Сгенерированная структура базы данных

```
BEGIN;

--
-- Create model Article
--
CREATE TABLE "myblog_article" (
        "id" serial NOT NULL PRIMARY KEY,
        "title" varchar(80) NOT NULL,
        "body" text NOT NULL,
        "pub_date" timestamp with time zone NOT NULL,
        "rating" smallint NOT NULL
);

--
-- Create model Tag
--
CREATE TABLE "myblog_tag" (
        "id" serial NOT NULL PRIMARY KEY,
        "tag" varchar(50) NOT NULL UNIQUE
);

--
-- Create model Vote
--
CREATE TABLE "myblog_vote" (
        "id" serial NOT NULL PRIMARY KEY,
        "value" smallint NOT NULL,
        "object_id" integer NOT NULL CHECK ("object_id" >= 0),
        "content_type_id" integer NOT NULL,
        "user_id" integer NOT NULL
);

--
-- Create model Comment
--
CREATE TABLE "myblog_comment" (
        "id" serial NOT NULL PRIMARY KEY,
        "body" text NOT NULL,
        "pub_date" timestamp with time zone NOT NULL,
        "rating" smallint NOT NULL,
        "article_id" integer NOT NULL,
        "user_id" integer NOT NULL
);

--
-- Add field tags to article
--
CREATE TABLE "myblog_article_tags" (
        "id" serial NOT NULL PRIMARY KEY,
        "article_id" integer NOT NULL,
        "tag_id" integer NOT NULL
);
```

```
53
54 --
55 -- Add field user to article
56 --
57 ALTER TABLE "myblog_article"
58         ADD COLUMN "user_id" integer NOT NULL
59         CONSTRAINT "myblog_article_user_id_7728319b_fk_auth_user_id"
60         REFERENCES "auth_user"("id") DEFERRABLE INITIALLY DEFERRED;
61 SET CONSTRAINTS "myblog_article_user_id_7728319b_fk_auth_user_id" IMMEDIATE;
62
63 CREATE INDEX "myblog_tag_tag_e004ad11_like" ON "myblog_tag" ("tag" varchar_pattern_ops);
64
65 ALTER TABLE "myblog_vote"
66         ADD CONSTRAINT "myblog_vote_content_type_id_06bf44c6_fk_django_content_type_id"
67         FOREIGN KEY ("content_type_id") REFERENCES "django_content_type" ("id")
68         DEFERRABLE INITIALLY DEFERRED;
69
70 ALTER TABLE "myblog_vote"
71         ADD CONSTRAINT "myblog_vote_user_id_89a11e40_fk_auth_user_id"
72         FOREIGN KEY ("user_id") REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED;
73
74 CREATE INDEX "myblog_vote_content_type_id_06bf44c6" ON "myblog_vote" ("content_type_id");
75
76 CREATE INDEX "myblog_vote_user_id_89a11e40" ON "myblog_vote" ("user_id");
77
78 ALTER TABLE "myblog_comment"
79         ADD CONSTRAINT "myblog_comment_article_id_44b1452c_fk_myblog_article_id"
80         FOREIGN KEY ("article_id") REFERENCES "myblog_article" ("id")
81         DEFERRABLE INITIALLY DEFERRED;
82
83 ALTER TABLE "myblog_comment"
84         ADD CONSTRAINT "myblog_comment_user_id_1d5be68b_fk_auth_user_id"
85         FOREIGN KEY ("user_id") REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED;
86
87 CREATE INDEX "myblog_comment_article_id_44b1452c" ON "myblog_comment" ("article_id");
88
89 CREATE INDEX "myblog_comment_user_id_1d5be68b" ON "myblog_comment" ("user_id");
90
91 ALTER TABLE "myblog_article_tags"
92         ADD CONSTRAINT "myblog_article_tags_article_id_tag_id_7e4e85c7_uniq"
93         UNIQUE ("article_id", "tag_id");
94
95 ALTER TABLE "myblog_article_tags"
96         ADD CONSTRAINT "myblog_article_tags_article_id_04a3a3b7_fk_myblog_article_id"
97         FOREIGN KEY ("article_id") REFERENCES "myblog_article" ("id")
98         DEFERRABLE INITIALLY DEFERRED;
99
100 ALTER TABLE "myblog_article_tags"
101         ADD CONSTRAINT "myblog_article_tags_tag_id_10cdc0fe_fk_myblog_tag_id"
102         FOREIGN KEY ("tag_id") REFERENCES "myblog_tag" ("id") DEFERRABLE INITIALLY DEFERRED;
103
104 CREATE INDEX "myblog_article_tags_article_id_04a3a3b7" ON "myblog_article_tags" ("article_id");
105
106 CREATE INDEX "myblog_article_tags_tag_id_10cdc0fe" ON "myblog_article_tags" ("tag_id");
107
108 CREATE INDEX "myblog_article_user_id_7728319b" ON "myblog_article" ("user_id");
109
110 COMMIT;
```

Приложение Б. Представления

В листингах 4.2–4.5 показаны представления, связанные со страницей статьи. Используются стандартные представления `DetailView`, `CreateView`, `UpdateView`, `DeleteView` из модуля `django.views.generic`.

Листинг 4.2: Представление страницы статьи

```
class ArticleView(DetailView, MultipleObjectMixin):
    model = Article  # for DetailView
    template_name = 'myblog/article.html'

    paginate_by = 5  # for MultipleObjectMixin

    # cannot define get_queryset for MultipleObjectMixin

    def get_ordering(self):
        return order_by(self.request.GET.get('order_by'))

    def get_context_data(self, **kwargs):
        object_list = Article.objects.comments(self.kwargs['pk']).order_by(self.get_ordering())
        context = super().get_context_data(object_list=object_list, **kwargs)
        return context
```

Листинг 4.3: Представление страницы создания статьи

```
class ArticleCreate(LoginRequiredMixin, CreateView):
    model = Article
    fields = ['title', 'body']
    template_name = 'myblog/create.html'
    login_url = reverse_lazy('login')

    def form_valid(self, form):
        form.instance.user = self.request.user
        return super().form_valid(form)
```

Листинг 4.4: Представление страницы редактирования статьи

```
class ArticleUpdate(LoginRequiredMixin, UpdateView):
    model = Article
    fields = ['title', 'body']
    template_name = 'myblog/create.html'
    login_url = reverse_lazy('login')
```

Листинг 4.5: Представление страницы удаления статьи

```
class ArticleDelete(LoginRequiredMixin, DeleteView):
    model = Article
    template_name = 'myblog/confirm_delete.html'
    success_url = reverse_lazy('blog:feed')
    login_url = reverse_lazy('login')
```

В листингах 4.6–4.8 показаны представления списка статей, основанные на стандартном представлении `ListView` из модуля

`django.views.generic`.

Листинг 4.6: Представление страницы ленты статей

```
class FeedView(ListView):
    model = Article
    template_name = 'myblog/article_list.html'
    paginate_by = 5

    def get_ordering(self):
        return order_by(self.request.GET.get('order_by'))
```

Листинг 4.7: Представление страницы статей определённого пользователя

```
class BlogView(ListView):
    model = Article
    template_name = 'myblog/article_list.html'
    paginate_by = 5

    def get_queryset(self):
        username = self.kwargs['username']
        user = get_object_or_404(User, username=username)
        self.kwargs['first_name'] = user.first_name
        self.kwargs['last_name'] = user.last_name
        return Article.objects.users(user).order_by(self.get_ordering())

    def get_ordering(self):
        return order_by(self.request.GET.get('order_by'))

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['first_name'] = self.kwargs['first_name']
        context['last_name'] = self.kwargs['last_name']
        return context
```

Листинг 4.8: Представление страницы статей определённого тэга

```
class TagView(ListView):
    model = Article
    template_name = 'myblog/article_list.html'
    paginate_by = 5

    def get_queryset(self):
        return Article.objects.tags(self.kwargs['tag']).order_by(self.get_ordering())

    def get_ordering(self):
        return order_by(self.request.GET.get('order_by'))

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['tag'] = self.kwargs['tag']
        return context
```

В листингах 4.9–4.11 показаны представления, связанные с комментариями. Используются стандартные представления `DetailView`, `CreateView`, `UpdateView`, `DeleteView` из модуля `django.views.generic`.

В листингах 4.9 и 4.12 показаны представления страницы создания комментария и регистрации соответственно. Используются стандартное представление `CreateView` из модуля `django.views.generic`.

Листинг 4.9: Представление страницы создания комментария

```
class CommentCreate(LoginRequiredMixin, CreateView):
    model = Comment
    fields = ['body']
    template_name = 'myblog/create.html'
    login_url = reverse_lazy('login')

    def form_valid(self, form):
        form.instance.user = self.request.user
        form.instance.article = Article.objects.get(id=self.kwargs['pk'])
        return super().form_valid(form)

    def get_success_url(self):
        return reverse('blog:articles', kwargs={'pk': self.kwargs['pk']})
```

Листинг 4.10: Представление страницы обновления комментария

```
class CommentUpdate(LoginRequiredMixin, UpdateView):
    model = Comment
    fields = ['body']
    template_name = 'myblog/create.html'
    login_url = reverse_lazy('login')

    def get_success_url(self):
        return reverse('blog:articles', kwargs={'pk':
            Comment.objects.get(id=self.kwargs['pk']).article_id})
```

Листинг 4.11: Представление страницы удаления комментария

```
class CommentDelete(LoginRequiredMixin, DeleteView):
    model = Comment
    template_name = 'myblog/confirm_delete.html'
    login_url = reverse_lazy('login')

    def get_success_url(self):
        return reverse('blog:articles', kwargs={'pk':
            Comment.objects.get(id=self.kwargs['pk']).article_id})
```

Листинг 4.12: Представление страницы регистрации

```
class RegisterView(CreateView):
    model = User
    fields = ['username', 'password', 'first_name', 'last_name', 'email']
    template_name = 'myblog/register.html'
    success_url = reverse_lazy('login')

    def form_valid(self, form):
        # Hash password before sending it to super
        form.instance.password = make_password(form.instance.password)
        return super().form_valid(form)
```

Приложение В. Шаблоны

Листинг 4.13: Шаблон основы страницы

```
<!-- this is base template with navbar and footer -->
{% load static %}
{% load url_replace %}

<!doctype html>
<html lang="en">
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
        href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
        integrity="sha384-9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk"
        crossorigin="anonymous">

    <link rel="stylesheet" type="text/css" href="{% static 'myblog/css/main.css' %}"/>

    <title>MyBlog</title>
</head>
<body>

<!-- navbar -->
<nav class="navbar navbar-expand-md navbar-dark bg-dark mb-4">
    <div class="container">
        <a class="navbar-brand" href="{% url 'blog:feed' %}">MyBlog App</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse"
            data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false"
            aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse navbar-right" id="navbarSupportedContent">
            <ul class="nav navbar-nav mr-auto">
                {% if user.is_authenticated %}
                    <li class="nav-item {% if user.username in request.get_full_path %}active{%
                        endif %}">
                        <a class="nav-link" href="{% url 'blog:user_articles' user.username
                            %}">My Blog</a>
                    </li>
                    <li class="nav-item {% if request.get_full_path == '/blog/article/create/'
                        %}active{% endif %}">
                        <a class="nav-link" href="{% url 'blog:create_article' %}">New
                            Article</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="{% url 'logout' %}">Sign Out</a>
                    </li>
                {% else %}
                    <li class="nav-item {% if request.get_full_path == '/register/' %}active{%
                        endif %}">
                        <a class="nav-link" href="{% url 'register' %}">Registration</a>
                    </li>
                    <li class="nav-item {% if request.get_full_path == '/login/' %}active{%
                        endif %}">
```

```
                        <a class="nav-link" href="{% url 'login' %}">Login</a>
                    </li>
                {% endif %}
                <li class="nav-item dropdown">
                    <a class="nav-link dropdown-toggle" href="#" id="navbarDropdownMenuLink"
                        role="button" data-toggle="dropdown" aria-haspopup="true"
                        aria-expanded="false">Order by</a>
                    <div class="dropdown-menu" aria-labelledby="navbarDropdownMenuLink">
                        <a class="dropdown-item" href="?{% url_replace 'order_by' '-pub_date'
                            %}">-pub_date</a>
                        <a class="dropdown-item" href="?{% url_replace 'order_by' 'pub_date'
                            %}">pub_date</a>
                        <a class="dropdown-item" href="?{% url_replace 'order_by' '-rating'
                            %}">-rating</a>
                        <a class="dropdown-item" href="?{% url_replace 'order_by' 'rating'
                            %}">rating</a>
                    </div>
                </li>
            </ul>
            <form class="form-inline my-2 my-lg-0" method="get" {# action="{% url
                'blog:search_articles' %}" #}>
                <input class="form-control mr-sm-2" type="search" placeholder="Search"
                    aria-label="Search">
                <button class="btn btn-outline-light my-2 my-sm-0" type="submit">Search</button>
            </form>
        </div>
    </div>
</nav>

<main role="main">
    <div class="container">
        {% block content %}{% endblock %}
    </div>
</main>

<hr>

<footer class="container">
    <div class="row">
        <div class="col-sm-12">

            <!-- Logged user info -->
            {% if user.is_authenticated %}
                <p class="text-center">
                    | Logged user: {{ user.username }}
                    | Email: {{ user.email }}
                    | Admin:{{ user.is_superuser }}
                    | <a href="{% url 'logout' %}">Sign Out</a>
                </p>
            {% endif %}

            <div>
                <p class="text-center">&copy; 2020 Kerimov Ahmed</p>
            </div>

        </div>
    </div>
</footer>

<!-- Optional JavaScript -->
<!-- jQuery first, then Popper.js, then Bootstrap JS -->
```

```
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
    integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
    crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
    integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
    crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js"
    integrity="sha384-OgVRvuATP1z7JjHLkuOU7Xw704+h835Lr+6QL9UvYjZE3Ipu6Tp75j7Bh/kR0JKI"
    crossorigin="anonymous"></script>

<script src="{% static 'myblog/js/main.js' %}"></script>

</body>
</html>
```

Листинг 4.14: Шаблон страниц списка статей

```
{% extends "myblog/base.html" %}

{% block content %}

    <!-- Show this section only for User Blog view -->
    {% if first_name or last_name %}
        <div class="jumbotron">
            <h1 class="display-4">{{ first_name }} {{ last_name }}  Blog</h1>
        </div>
    {% elif tag %}
        <div class="jumbotron">
            <h1 class="display-4">Tag: {{ tag }}</h1>
        </div>
    {% endif %}

    <!-- Show List of Posts -->
    {% include "myblog/card_list.html" %}

    {% include "myblog/pagination.html" %}

{% endblock content %}
```

Листинг 4.15: Шаблон пагинации

```
{% load url_replace %}

<!-- Pagination -->
<nav aria-label="Page navigation">
    <ul class="pagination justify-content-center">
        {% if page_obj.has_previous %}
            <li class="page-item"><a class="page-link" href="?{% url_replace 'page'
                page_obj.previous_page_number %}">Previous</a></li>
            <li class="page-item"><a class="page-link" href="?{% url_replace 'page' 1
                %}">1</a></li>
            {% if page_obj.number > 2 %}
                <li class="page-item"><a class="page-link" href="?{% url_replace 'page' 2
                    %}">2</a></li>
            {%  endif %}
        {% else %}
            <li class="page-item disabled"><a class="page-link" href="#">Previous</a></li>
        {% endif %}

        {% if page_obj.number > 5 %}
```

```
            <li class="page-item"><a class="page-link">...</a></li>
        {% endif %}

        {% if page_obj.number > 4 %}
            <li class="page-item"><a class="page-link" href="?{% url_replace 'page'
                page_obj.number|add:"-2" %}">{{ page_obj.number|add:"-2" }}</a></li>
        {% endif %}

        {% if page_obj.number > 3 %}
            <li class="page-item"><a class="page-link" href="?{% url_replace 'page'
                page_obj.previous_page_number %}">{{ page_obj.previous_page_number }}</a></li>
        {% endif %}

        <li class="page-item active"><a class="page-link" href="#">{{ page_obj.number }}</a></li>

        {% if page_obj.number|add:2 < page_obj.paginator.num_pages %}
            <li class="page-item"><a class="page-link" href="?{% url_replace 'page'
                page_obj.next_page_number %}">{{ page_obj.next_page_number }}</a></li>
        {% endif %}

        {% if page_obj.number|add:3 < page_obj.paginator.num_pages %}
            <li class="page-item"><a class="page-link" href="?{% url_replace 'page'
                page_obj.number|add:2 %}">{{ page_obj.number|add:2 }}</a></li>
        {% endif %}

        {% if page_obj.number|add:4 < page_obj.paginator.num_pages %}
            <li class="page-item"><a class="page-link">...</a></li>
        {% endif %}

        {% if page_obj.has_next %}
            {% if page_obj.number|add:1 < page_obj.paginator.num_pages %}
                <li class="page-item"><a class="page-link" href="?{% url_replace 'page'
                    page_obj.paginator.num_pages|add:"-1" %}">{{
                    page_obj.paginator.num_pages|add:"-1" }}</a></li>
            {% endif %}
            <li class="page-item"><a class="page-link" href="?{% url_replace 'page'
                page_obj.paginator.num_pages %}">{{ page_obj.paginator.num_pages }}</a></li>
            <li class="page-item"><a class="page-link" href="?{% url_replace 'page'
                page_obj.paginator.next_page_number %}">Next</a></li>
        {% else %}
            <li class="disabled page-item"><a class="page-link" href="#">Next</a></li>
        {% endif %}
    </ul>
</nav>
```

Листинг 4.16: Шаблон страницы логина

```
{% extends "myblog/base.html" %}

{% block content %}
    <form method="post" action="{% url 'login' %}" class="form-sign">

        {% csrf_token %}

        {% if form.errors %}
            <div class="alert alert-danger">Your username and password didn't match. Please try
                again.</div>
        {% else %}
            {% if next %}
                {% if user.is_authenticated %}
                    <div class="alert alert-info">Your account doesn't have access to this page.
```

```
                        To proceed, please login with an account that has access.</div>
                {% else %}
                    <div class="alert alert-info">Please login to see this page.</div>
                {% endif %}
            {% endif %}
        {% endif %}

        <div class="form-group">
            {% if form.username.errors %}
                <label class="alert alert-danger">{{ form.username.errors }}</label>
            {% endif %}
            <input type="text" placeholder="Username" class="form-control" name="username"
                required>
        </div>

        <div class="form-group">
            {% if form.password.errors %}
                <label class="alert alert-danger">{{ form.password.errors }}</label>
            {% endif %}
            <input type="password" placeholder="Password" class="form-control" name="password"
                required>
        </div>

        <div class="form-group">
            <button type="submit" class="btn btn-primary btn-block">Login</button>
        </div>

        <!-- Form field - next -->
        <!-- Used to configure redirect after successful login -->
        <input type="hidden" name="next" value="{% url 'blog:feed' %}"/>
    </form>
{% endblock content %}
```

Листинг 4.17: Шаблон страницы регистрации

```
{% extends "myblog/base.html" %}

{% block content %}
    <form action="" method="post" class="form-sign">

        {% csrf_token %}

        <div class="form-group">
            {% if form.username.errors %}
                <label class="alert alert-danger">{{ form.username.errors }}</label>
            {% endif %}
            <input type="text" placeholder="Username" class="form-control" name="username"
                required value="{{ form.username.value|default:"" }}">
        </div>

        <div class="form-group">
            {% if form.password.errors %}
                <label class="alert alert-danger">{{ form.password.errors }}</label>
            {% endif %}
            <input type="password" placeholder="Password" class="form-control" name="password"
                required value="{{ form.password.value|default:"" }}">
        </div>

        <div class="form-group">
            {% if form.first_name.errors %}
                <label class="alert alert-danger">{{ form.first_name.errors }}</label>
```

```
            {% endif %}
            <input type="text" placeholder="First Name" class="form-control" name="first_name"
                required value="{{ form.first_name.value|default:"" }}">
        </div>

        <div class="form-group">
            {% if form.first_name.errors %}
                <label class="alert alert-danger">{{ form.last_name.errors }}</label>
            {% endif %}
            <input type="text" placeholder="Last Name" class="form-control" name="last_name"
                required value="{{ form.last_name.value|default:"" }}">
        </div>

        <div class="form-group">
            {% if form.email.errors %}
                <label class="alert alert-danger">{{ form.email.errors }}</label>
            {% endif %}
            <input type="text" placeholder="Email" class="form-control" name="email" required
                value="{{ form.email.value|default:"" }}">
        </div>

        <div class="form-group">
            <button type="submit" class="btn btn-primary btn-block">Submit</button>
        </div>
    </form>
{% endblock content %}
```

Листинг 4.18: Шаблон страницы создания

```
{% extends "myblog/base.html" %}

{% block content %}
    <form action="" method="post" class="form-create">

        {% csrf_token %}

        {% if form.title %}
            <div class="form-group">
                {% if form.title.errors %}
                    <label class="alert alert-danger">{{ form.title.errors }}</label>
                {% endif %}
                <input type="text" placeholder="Title" class="form-control" name="title"
                    value="{{ form.title.value|default:"" }}" required>
            </div>
        {% endif %}

        <div class="form-group">
            {% if form.body.errors %}
                <label class="alert alert-danger">{{ form.body.errors }}</label>
            {% endif %}
            <textarea placeholder="Body" class="form-control" name="body" rows="10" required>{{
                form.body.value|default:"" }}</textarea>
        </div>

        <div class="form-group">
            <button type="submit" class="btn btn-primary btn-block">Save</button>
        </div>
    </form>
{% endblock content %}
```

Листинг 4.19: Шаблон страницы подтверждения удаления

```
{% extends "myblog/base.html" %}
{% load define %}

{% block content %}
    <form action="" method="post" class="form-create">
        {% csrf_token %}
        {% if object.title %}
            {% define "Article" as type %}
            {% define object.title as prompt %}
        {% else %}
            {% define "Comment" as type %}
            {% define object.body as prompt %}
        {% endif %}
        <div class="alert alert-info">Are you sure you want to delete {{ type }}: "{{ prompt
            }}"?</div>
        <div class="form-group">
            <button type="submit" class="btn btn-primary btn-block">Confirm</button>
        </div>
    </form>
{% endblock content %}
```

Приложение Г. Генерация фейковых данных

Листинг 4.20: Генерация фейковых данных

```python
from django.core.management.base import BaseCommand, CommandError
from django.contrib.auth.hashers import make_password
from django.contrib.auth.models import User
from django.contrib.contenttypes.models import ContentType
from faker import Faker

from myblog.models.comment import Comment
from myblog.models.article import Article
from myblog.models.tag import Tag
from myblog.models.vote import Vote

fake = Faker()


class Command(BaseCommand):
    help = 'Generates fake data for db'

    def add_arguments(self, parser):
        parser.add_argument('-u', '--users', type=int, default=0)
        parser.add_argument('-t', '--tags', type=int, default=0)
        parser.add_argument('-T', '--max-tags-per-article', type=int, default=0)
        parser.add_argument('-s', '--max-sentences-per-article', type=int, default=15)
        parser.add_argument('-a', '--articles', type=int, default=0)
        parser.add_argument('-S', '--max-sentences-per-comment', type=int, default=8)
        parser.add_argument('-c', '--comments', type=int, default=0)
        parser.add_argument('-A', '--max-votes-per-article', type=int, default=0)
        parser.add_argument('-C', '--max-votes-per-comment', type=int, default=0)

    @staticmethod
    def create_users(users):
        offset_id = User.objects.count() + 1
        User.objects.bulk_create(
            [
                User(
                    first_name=(first_name := fake.first_name()),
                    last_name=(last_name := fake.last_name()),
                    username=(username := f'{first_name}{last_name}{offset_id + i}'),
                    password=make_password(f'{username}password', None, 'md5'),
                    email=f'{username}@example.com',
                )
                for i in range(users)
            ]
        )

    @staticmethod
    def create_tags(tags):
        offset_id = Tag.objects.count() + 1
        Tag.objects.bulk_create(
            [
                Tag(
                    tag=f'{fake.word()}{offset_id + i}'
                )
```

```
                for i in range(tags)
            ]
        )

    @staticmethod
    def fast_randint(mn, mx):
        return int(fake.random.random() * (mx - mn + 1)) + mn

    @staticmethod
    def fast_vote(kindness_coefficient=0.5):
        return 1 if fake.random.random() < kindness_coefficient else -1

    @staticmethod
    def create_articles(articles, max_tags_per_article, max_sentences_per_article):
        user_id_max = User.objects.count()
        Article.objects.bulk_create(
            [
                Article(
                    title=fake.sentence(),
                    body=fake.paragraph(nb_sentences=max_sentences_per_article),
                    user_id=Command.fast_randint(1, user_id_max),
                )
                for i in range(articles)
            ]
        )

        through_model = Article.tags.through
        n_articles = Article.objects.count()
        tag_ids = list(Tag.objects.values_list('id', flat=True))  # list for sample()
        through_list = []
        for article_id in range(n_articles - articles + 1, n_articles + 1):
            n_tags = Command.fast_randint(0, max_tags_per_article)
            through_list += [
                through_model(
                    article_id=article_id,
                    tag_id=tag_id,
                )
                for tag_id in fake.random.sample(tag_ids, n_tags)
            ]
        through_model.objects.bulk_create(through_list)

    @staticmethod
    def create_comments(comments, max_sentences_per_comment):
        user_id_max = User.objects.count()
        article_id_max = Article.objects.count()
        Comment.objects.bulk_create(
            [
                Comment(
                    body=fake.paragraph(nb_sentences=max_sentences_per_comment),
                    user_id=Command.fast_randint(1, user_id_max),
                    article_id=Command.fast_randint(1, article_id_max),
                )
                for i in range(comments)
            ]
        )

    @staticmethod
    def generate_votes_for_model(model_cls, max_votes_per_article):
        if max_votes_per_article == 0:
            return

```

```
        user_ids = list(User.objects.values_list('id', flat=True))  # list for sample()
        models = model_cls.objects.all()
        model_type_id = ContentType.objects.get_for_model(model_cls).id

        votes = []
        for model in models:
            n_votes = Command.fast_randint(0, max_votes_per_article)
            rating = 0
            kindness_coefficient = fake.random.random()
            for user_id in fake.random.sample(user_ids, n_votes):
                value = Command.fast_vote(kindness_coefficient)
                rating += value
                votes.append(
                    Vote(
                        value=value,
                        object_id=model.pk,
                        content_type_id=model_type_id,
                        user_id=user_id,
                    )
                )
            model.rating += rating

        Vote.objects.bulk_create(votes)
        model_cls.objects.bulk_update(models, ['rating'])

    def handle(self, *args, **options):
        if (users := options["users"]) > 0:
            print(f'Generate {users} users')
            self.create_users(users)
        if (tags := options['tags']) > 0:
            print(f'Generate {tags} tags')
            self.create_tags(tags)
        if (articles := options['articles']) > 0:
            max_tags_per_article = options['max_tags_per_article']
            max_sentences_per_article = options['max_sentences_per_article']
            print(f'Generate {articles} articles with up to {max_sentences_per_article}
                sentences and up to {max_tags_per_article} tags for each article')
            self.create_articles(articles, max_tags_per_article, max_sentences_per_article)
        if (comments := options['comments']) > 0:
            max_sentences_per_comment = options['max_sentences_per_comment']
            print(f'Generate {comments} comments with up to {max_sentences_per_comment}
                sentences for each comment')
            self.create_comments(comments, max_sentences_per_comment)
        if (max_votes_per_article := options["max_votes_per_article"]) > 0:
            print(f'Generate up to {max_votes_per_article} votes for each article')
            self.generate_votes_for_model(Article, max_votes_per_article)
        if (max_votes_per_comment := options["max_votes_per_comment"]) > 0:
            print(f'Generate up to {max_votes_per_comment} votes for each comment')
            self.generate_votes_for_model(Comment, max_votes_per_comment)
```