

Министерство образования Иркутской области

Государственное бюджетное профессиональное образовательное учреждение

Иркутской области

«Иркутский авиационный техникум»

(ГБПОУИО «ИАТ»)

КП.09.02.07–1.25.221.20 ПЗ

ВЕБ-ПРИЛОЖЕНИЕ «ФОРУМ ВЕДЬМАК»

Председатель ВЦК:

(подпись, дата)

(Н.Р. Огородникова)

Руководитель:

(подпись, дата)

(Е.С. Кубата)

Студент:

(подпись, дата)

(Ю.К. Терентьев)

Иркутск 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	4
1 Описание предметной области.....	6
2 Анализ инструментальных средств разработки, используемых при реализации веб приложения	9
3 Техническое задание	17
4 Проектирование веб-приложения.....	18
4.1 Структурная схема веб-приложения.....	18
4.2 Функциональная схема веб-приложения.....	23
4.3 Проектирование базы данных.....	30
4.4 Проектирование интерфейса.....	37
5 Разработка веб-приложения	44
5.1 Разработка интерфейса веб-приложения	44
5.2 Разработка базы данных веб-приложения	47
5.3 Разработка веб-приложения	52
6 Тестирование веб-приложения	55
6.1 Тестовые сценарии.....	55
6.2 Методы тестирования	59
7 Документирование веб-приложения	63
7.1 Руководство по установке веб-приложения	63
7.2 Руководство гостя	64
7.3 Руководство авторизованного пользователя.....	68
7.4 Руководство администратора.....	72
ЗАКЛЮЧЕНИЕ	80

					КП.09.02.07–1.25.221.20.ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	ВЕБ-ПРИЛОЖЕНИЕ «ФОРУМ» пояснительная записка	Лит.	Лист	Листов
Разраб.		Терентьев Ю.К.					2	99
Провер.		Кубата Е.С.						
Реценз.								
Н. Контр.								
Утверд.						ГБПОУИО «ИАТ» БД-22-1		

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ.....	83
Приложение А – Техническое задание	85
Приложение Б – Листинг Urls.....	95

					КП.09.02.07–1.25.221.20.ПЗ	Лист
						3
Изм.	Лист	№ докум.	Подпись	Дата		

ВВЕДЕНИЕ

Веб-приложение Форум по вселенной «Ведьмака» – это информационная платформа, предоставляющая пользователям доступ к структурированной базе знаний о мире, созданном Анджеем Сапковским. Приложение позволяет изучать информацию о персонажах, локациях, монстрах, магии и ключевых событиях, а также взаимодействовать с контентом через удобный интерфейс. В современном мире, где фэнтези-вселенные становятся частью массовой культуры, подобные ресурсы пользуются большим спросом среди фанатов, исследователей и новых зрителей, желающих глубже погрузиться в историю «Ведьмака».

Разработка качественного форума – это сложная задача, требующая внимания к нескольким ключевым аспектам:

- обширный и структурированный контент: успешное веб-приложение должно содержать детальную информацию по всем элементам вселенной, что требует продуманной системы хранения, категоризации и поиска данных.
- Удобный и интуитивный интерфейс: приложение должно быть простым в использовании как для обычных читателей, так и для редакторов, позволяя легко находить нужные материалы и при необходимости вносить правки.
- Гибкость и масштабируемость: поскольку вселенная «Ведьмака» постоянно расширяется (книги, игры, сериал), веб-приложение должен позволять добавлять новый контент без нарушения существующей структуры.

Преимущества веб-приложения:

- централизованный доступ к информации: пользователи могут быстро находить проверенные данные без необходимости поиска по разрозненным источникам.
- Возможность совместного редактирования: зарегистрированные пользователи могут дополнять и уточнять статьи, поддерживая актуальность базы знаний.

					КП.09.02.07–1.25.221.20.ПЗ	Лист 4
Изм.	Лист	№ докум.	Подпись	Дата		

– Мультимедийность: интеграция изображений, карт, цитат и ссылок на первоисточники делает изучение вселенной более наглядным и увлекательным.

Цель курсового проекта: разработка функционального веб-приложения на Django, которое обеспечит удобный доступ к информации о вселенной «Ведьмака» и позволит пользователям взаимодействовать с контентом.

Основные задачи:

– анализ предметной области: изучение структуры вселенной «ведьмака» и определение ключевых сущностей для базы данных (персонажи, локации, магия).

– Выбор инструментов разработки: обоснование использования django, postgresql (или sqlite), html/css/javascript и дополнительных библиотек.

– Проектирование базы данных: создание er-диаграммы и схемы таблиц с учетом связей между сущностями.

– Разработка backend-части:

– Настройка моделей django для хранения данных.

– Реализация CRUD – функционала для статей.

– Создание системы аутентификации и разграничения прав (например, модераторы/обычные пользователи).

– Разработка frontend – части:

– Верстка шаблонов с адаптивным дизайном.

– Добавление интерактивных элементов (поиск, фильтры, пагинация).

– Тестирование: проверка работоспособности всех функций, включая обработку пользовательского ввода и отображение контента.

– Документирование: описание архитектуры проекта, руководство для пользователей и разработчиков.

1 Описание предметной области

Предметной областью веб-приложения «Форум по вселенной Ведьмака» является создание централизованной онлайн-платформы для общения, обсуждения и систематизации знаний о мире, созданном Анджеем Сапковским. Форум – это комплексная интерактивная среда, основная деятельность которой заключается в организации публичных и тематических обсуждений для сообщества фанатов. Деятельность форума включает не только обмен мнениями, но и управление пользовательским контентом, модерацию, категоризацию информации, а также непосредственное взаимодействие между участниками.

Данное веб-приложение предназначено для автоматизации ключевых процессов функционирования форума, сфокусированных на пользовательском взаимодействии. Цель веб-приложения – предоставить участникам сообщества удобный и современный инструмент для поиска информации, создания обсуждений и комментирования, а также предоставить администрации и модераторам эффективные средства для управления контентом и поддержания порядка.

Веб-приложение оперирует следующими основными сущностями:

1. пользователь:

– учетная запись участника форума. Характеризуется уникальными логином и электронной почтой, именем, фамилией, датой регистрации и статусом аккаунта. Ключевым атрибутом является ссылка на роль, определяющая уровень доступа в системе.

2. Роль:

– определяет права и полномочия пользователя. Система поддерживает четыре основные роли: читатель, редактор, модератор и администратор. Роль характеризуется названием, описанием и набором разрешений в формате JSON.

3. Статья:

					КП.09.02.07–1.25.221.20.ПЗ	Лист 6
Изм.	Лист	№ докум.	Подпись	Дата		

– основная содержательная единица форума, энциклопедическая запись. Содержит заголовок, текст статьи, аннотацию, даты создания и публикации, а также статус. Связана с автором и категорией.

4. Категория:

– тематический раздел для группировки статей. Имеет древовидную структуру благодаря ссылке на родительскую категорию, что позволяет создавать сложные иерархии.

5. Медиафайл:

– файл, загруженный в систему. Хранит техническую информацию (имя, путь, размер, тип), а также данные для SEO (альтернативный текст).

6. Комментарий:

– отзыв или вопрос пользователя к статье. Поддерживает древовидную структуру (ответ на комментарий). Имеет статус модерации.

7. Тег:

– ключевое слово для гибкой категоризации и связывания статей (например, #Геральт, #Ведьмак, #Законы_Удивления). Характеризуется названием, цветом для визуального выделения и описанием.

В контексте функционала веб-приложения основными участниками и процессами являются:

1. Читатель:

– Просмотр контента: ознакомление с каталогом статей, фильтрация по категориям и тегам. Просмотр содержимого статей и связанных медиафайлов.

– Поиск информации: использование системы поиска для быстрого нахождения нужных материалов по вселенной.

– Комментирование: написание комментариев и вопросов к статьям, участие в обсуждениях.

– Регистрация и авторизация: Создание учетной записи для получения возможности комментирования.

2. Редактор:

					КП.09.02.07–1.25.221.20.ПЗ	Лист 7
Изм.	Лист	№ докум.	Подпись	Дата		

- создание и редактирование статей: наполнение энциклопедии: написание новых статей, а также улучшение и актуализация существующих материалов.

- Работа с медиа: загрузка и прикрепление изображений, карт и других файлов к статьям.

- Структурирование контента: присвоение статей категориям, добавление тегов и создание связей между статьями для улучшения навигации.

3. Модератор:

- контроль содержания: проверка новых статей и комментариев перед публикацией (пре-модерация) или после (пост-модерация).

- Поддержание порядка: удаление или редактирование некорректного контента, нарушающего правила форума.

- Управление дискуссиями: слежение за обсуждениями в комментариях.

4. Администратор:

- управление пользователями: назначение ролей (редактор, модератор), блокировка нарушителей.

- Управление структурой: создание и редактирование категорий, управление тегами.

- Настройка системы: конфигурация прав доступа для различных ролей.

- Аналитика: просмотр статистики по популярности статей и активности пользователей.

Таким образом, разрабатываемое веб-приложение «Форум по вселенной Ведьмака» предназначено для создания централизованной, структурированной и интерактивной энциклопедии. Веб-приложение предоставляет пользователям удобный инструмент для изучения мира, а редакторам и модераторам – эффективные средства для наполнения и поддержания порядка, что способствует формированию активного и знающего сообщества фанатов.

					КП.09.02.07–1.25.221.20.ПЗ	Лист 8
Изм.	Лист	№ докум.	Подпись	Дата		

2 Анализ инструментальных средств разработки, используемых при реализации веб приложения

Инструменты разработки программного продукта определяют будущий результат.

Проектировать структуру веб-приложения более удобно через инструменты разработки такие как: MySQL Workbench и Draw.io.

Веб-приложения будет состоять из двух частей – клиентская и серверная. Для реализации клиентской части отлично подойдет PyCharm Community Edition.

Серверная часть будет со стандартной базой данных PostgreSQL.

MySQL Workbench – Визуальный инструмент для проектирования баз данных. Хотя в проекте была использована СУБД PostgreSQL, MySQL Workbench был применен на этапе проектирования для создания детальной ER-модели предметной области из-за удобного и наглядного интерфейса.

Draw.io – это онлайн-инструмент для создания схем, диаграмм и блок-схем. Это бесплатное приложение, которое широко используется для визуализации процессов, архитектуры систем, ER-диаграмм баз данных и других видов диаграмм.

PyCharm Professional Edition – это интегрированная среда разработки (IDE) для языка программирования Python. IDE разработана компанией JetBrains и предоставляет все необходимые базовые инструменты для написания, отладки и запуска кода.

Python – это универсальный язык программирования, широко используемый в различных областях, таких как веб-разработка, анализ данных, машинное обучение, разработка программного обеспечения и многое другое. Python известен своей простотой и читабельностью, что делает данный язык отличным выбором как для начинающих, так и для опытных разработчиков. Веб-приложение будет содержать в себе информацию, которую необходимо хранить,

изменять, структурировать и использовать. Это реализуется благодаря базе данных. Сравнение средств реализации базы данных представлены в таблице 1.

Таблица 1 - Сравнение средств реализации базы данных

Критерий	PostgreSQL	MySQL	SQLite
Тип	Объектно-реляционная СУБД (ORDBMS)	Реляционная СУБД (RDBMS)	Встраиваемая реляционная СУБД (на основе файлов)
Производительность	Очень высокая на сложных запросах, с одновременным доступом.	Высокая на операциях чтения.	Очень высокая для однопользовательского доступа, не предназначена для высоких нагрузок.
Масштабируемость	Поддержка сложных репликаций, горизонтальное и вертикальное масштабирование.	Поддерживает репликацию и шардинг.	Отсутствует, так как это встраиваемая БД.
Функции и стандарты	Наиболее строгое соответствие стандарту SQL, поддержка оконных функций, JSON.	Хорошая поддержка SQL, но с некоторыми отклонениями от стандарта.	Поддерживает большинство SQL-запросов, но с ограничениями (напр., ограниченный ALTER TABLE).
Использование	Крупные и сложные проекты, где важна целостность данных и богатый функционал.	Веб-приложения, где преобладают операции чтения.	Мобильные приложения, простые десктопные приложения, прототипирование.

Выбор для проекта: для данного веб-приложения была выбрана PostgreSQL. Основными причинами стали:

- надежность и целостность данных. PostgreSQL строго следует принципу ACID, это критически важно для форума, где необходимо гарантировать, что ни один комментарий или правка статьи не будут потеряны из-за сбоя, а данные пользователей останутся консистентными.
- Поддержка сложных структур данных и запросов. Форум подразумевает не только простые выборки, но и сложные операции:

- рекурсивные запросы: идеально подходят для построения древовидных структур комментариев и вложенных категорий.
- Полнотекстовый поиск: позволяет реализовать мощный и точный поиск по текстам статей и комментариев.
- Работа с JSON (JSONB): поле permissions в таблице roles эффективно хранить и запрашивать в формате JSON, а высокая производительность типа JSONB идеально для этого подходит.
- Масштабируемость. по мере роста популярности форума и увеличения объема данных, PostgreSQL предоставляет проверенные инструменты для масштабирования, обеспечивая стабильную работу под нагрузкой.
- Богатая экосистема и соответствие стандартам. строгое следование стандарту SQL.

Серверная часть веб-приложения требует языка с хорошей экосистемой для веб-разработки. Сравнение языков программирования представлено в таблице 2.

Таблица 2 – Сравнение языков программирования для разработки веб-приложения

Критерий	Python (Django)	PHP (Laravel)	Java (Spring Boot)
Синтаксис	Простой и читаемый синтаксис.	Простой, но может быть непоследовательным.	Строгий, вербальный синтаксис
Фреймворки	Мощные фреймворки (Django), множество библиотек для любых задач.	Огромное количество готовых решений и фреймворков (Laravel, Symfony).	Огромная enterprise-экосистема (Spring).
Безопасность	Django предоставляет встроенную защиту.	Требует большей внимательности к безопасности.	Высокий уровень безопасности
Производительность	Вполне достаточна для веб-приложений среднего масштаба. Выше, чем у PHP.	Ниже, чем у Python и Java.	Очень высокая.
Использование	Быстрая разработка комплексных приложений, веб-сервисы.	Традиционно веб-сайты.	Крупные корпоративные, высоконагруженные и распределенные системы.

Выбор для проекта: в качестве основного языка программирования был выбран Python вместе с фреймворком Django. Это решение обусловлено следующими факторами:

- скорость и простота разработки. Принцип «батарейки в комплекте» фреймворка Django позволяет быстро создать каркас приложения. Встроенная панель администратора, ORM система аутентификации и маршрутизация экономят сотни часов разработки. Это идеально для проекта с четкой структурой (статьи, категории, пользователи).

- читаемость кода и низкий порог входа. Четкий и лаконичный синтаксис Python облегчает командную работу, написание чистого кода и будущую поддержку проекта. Новым разработчикам проще вникнуть в проект, написанный на Python/Django.

- Мощный ORM (Object-Relational Mapping). Django ORM позволяет работать с базой данных PostgreSQL на высоком уровне абстракции, практически не написания SQL-запросов вручную. Это ускоряет разработку и снижает количество ошибок. Легко описать модели Article, Category, Comment, и Django сгенерирует необходимые таблицы.

- Безопасность «из коробки». Django предоставляет встроенные и надежные механизмы защиты от распространенных уязвимостей (CSRF, XSS, SQL-инъекции, кликинг).

- Идеальное соответствие задачам проекта. Форум не требует экстремальной производительности Java-приложений, но нуждается в быстрой реализации сложной логики и структуры данных. Python и Django идеально балансируют между простотой, скоростью разработки и мощностью для решения задач проекта.

Интегрированная среда разработки – ключевой инструмент, влияющий на продуктивность программиста. Сравнение IDE представлено в таблице 3.

					КП.09.02.07–1.25.221.20.ПЗ	Лист 12
Изм.	Лист	№ докум.	Подпись	Дата		

Таблица 3 – Сравнение IDE

Критерий	PyCharm (Professional)	Visual Studio Code	Sublime Text
Тип	Полнофункциональная IDE	Редактор кода с возможностями IDE	Высокопроизводительный текстовый редактор с плагинами
Поддержка Python/Django	Нативная, глубокая поддержка «из коробки».	Требуется установка расширений.	Требуется установка множества плагинов для достижения функционала IDE.
Производительность	Высокая, оптимизирована для больших проектов.	Очень высокая, быстрый запуск.	Чрезвычайно высокая, самый быстрый запуск и отклик.
Отладка	Мощный встроенный дебаггер.	Дебаггер через расширение.	Только через сторонние плагины.
Стоимость	Платная, есть бесплатный Community-вариант.	Полностью бесплатная.	Условно-бесплатная.

Выбор для проекта: В качестве основной IDE был выбран PyCharm Professional. Ключевые причины:

- исключительная поддержка Django. PyCharm предлагает лучшую на рынке интеграцию с Django: понимание структуры проекта, автодополнение для шаблонов, моделей и представлений, удобная навигация между файлами.
- Встроенный набор инструментов. Наличие встроенного дебаггера, базы данных, терминала и инструментов для работы с Git значительно ускоряет процесс разработки и отладки.
- Умный код-ассистент. Высококачественное автодополнение кода и анализ кода помогают избегать ошибок и писать код в соответствии со стандартами.

Обеспечение качества и надежности программного продукта является неотъемлемой частью процесса разработки. Сравнение инструментов тестирования представлено в таблице 4.

Таблица 4 – Сравнение инструментов тестирования

Критерий	pytest	unittest (стандартный)	Django Test Case (на базе unittest)
----------	--------	------------------------	-------------------------------------

Продолжение таблицы 4

Синтаксис и простота	Очень простой и лаконичный синтаксис.	Более многословный синтаксис. Требуется создания классов, наследования от TestCase.	Аналогичен unittest по многословности, но добавляет специализированные assertion-методы для Django,
Фикстуры (Fixtures)	Мощная, гибкая и модульная система фикстур с широкими возможностями.	Базовые возможности через методы setUp() и tearDown().	Использует механизм setUp()/tearDown().
Интеграция с Django	Требуется установки и настройки дополнительного плагина pytest-django для полноценной интеграции.	Интегрируется на базовом уровне.	Прямая и нативная интеграция со всеми компонентами Django.
Производительность и возможности	Поддержка параметризации тестов, параллельного запуска.	Базовый функционал. Параллельный запуск и параметризация требуют усилий или отсутствуют.	Хорошая интеграция с системой Django (создание/сброс БД).

Выбор тестирования для веб-приложения «Форум по вселенной Ведьмака» был выбран комбинированный подход с использованием Django Test Case в качестве основы и pytest для расширенного функционала. Основными причинами стали:

- django test case выбран как основной фреймворк для тестирования потому-что:
- глубокая нативная интеграция с Django. Это критически важно для тестирования специфичных компонентов форума: моделей статей и комментариев, представлений для работы с энциклопедией, системы прав доступа для разных ролей пользователей. Фреймворк автоматически создает тестовую базу данных и предоставляет специализированные методы проверки, что значительно упрощает тестирование сложной бизнес-логики форума.

– Оптимизация для тестирования Django-приложений. Встроенная поддержка тестового клиента позволяет легко эмулировать поведение пользователей с разными ролями (Читатель, Редактор, Модератор), проверять доступ к защищенным страницам и тестировать workflow публикации статей и модерации комментариев, что составляет основу функционала форума.

– Гарантированная совместимость и надежность. Как часть ядра Django, этот инструмент обеспечивает полную совместимость со всеми версиями фреймворка и корректно обрабатывает все Django-специфичные аспекты (сигналы, middleware, работу ORM), что особенно важно для проекта с долгосрочной перспективой поддержки.

Pytest добавлен в стек для расширения возможностей тестирования:

– гибкость создания тестовых данных. Мощная система фикстур pytest идеально подходит для создания сложных наборов тестовых данных, что особенно ценно для наполнения энциклопедии тестовым контентом.

– Параметризация тестов. Возможность запуска одинаковых тестов с разными наборами входных данных эффективна для проверки граничных случаев в формах комментариев, поисковых запросов и валидации данных статей.

Для создания программного продукта было решено использовать следующие инструментальные средства:

1. для проектирования архитектуры и баз данных использовались CASE-средства:

– draw.io – для создания структурных схем, диаграмм потоков данных (DFD), диаграмм вариантов использования (Use Case) и других UML-диаграмм.

– MySQL Workbench – для наглядного проектирования структуры реляционной базы данных и построения ER-диаграммы.

2. Для разработки пользовательского интерфейса (UI/UX) использовался современный онлайн-сервис:

– draw.io – для создания структурированных и полностью отражающих функционал приложения прототипов.

					КП.09.02.07–1.25.221.20.ПЗ	Лист 15
Изм.	Лист	№ докум.	Подпись	Дата		

3. Для реализации серверной части использовался следующий стек технологий:

- язык программирования Python – в качестве основного языка разработки благодаря чистоте синтаксиса, богатой экосистеме и высокой скорости создания приложений.

- Веб-фреймворк Django – в качестве основы backend`а для быстрой разработки безопасного и поддерживаемого кода по паттерну MVC (MVT).

- СУБД PostgreSQL – в качестве надежной и мощной системы управления базами данных для хранения всей информации о сеансах, билетах, пользователях и фильмах.

4. Для реализации клиентской части (Frontend) применялись стандартные веб-технологии:

- html5, css3 – для создания семантической разметки и стилизации веб-страниц.

5. В качестве основной среды разработки (IDE) был выбран:

- pycharm professional – как наиболее мощная и удобная среда, обеспечивающая глубочайшую поддержку как Python, так и фреймворка Django, включая отладку, рефакторинг и работу с базой данных прямо из IDE.

6. Для тестирования веб-приложения был выбран:

- django test case и pytest, которые максимально ориентированы на предметную область и технологический стек разработки, также Pytest сможет расширить функционал возможностей для тестирования.

3 Техническое задание

В начале разработки создавалось техническое задание, в котором указывались основные требования.

Для создания технического задания использовался стандарт ГОСТ 34.602–2020 «Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы».

Согласно ГОСТ 34.602–2020 техническое задание должно включать следующие разделы:

- 1 Введение.
- 2 Основания для разработки.
- 3 Назначение приложения.
- 4 Требования к веб-приложению.
- 5 Требования к техническому обеспечению.
- 6 Требования к программному обеспечению.
- 7 Требования к тестированию.
- 8 Организационно - технические требования.

Техническое задание на разработку веб-приложения представлено в приложении А.

4 Проектирование веб-приложения

4.1 Структурная схема веб-приложения

Диаграмма прецедентов описывает высокоуровневые функциональные возможности системы с точки зрения пользователей. Диаграмма изображена на рисунке 1.

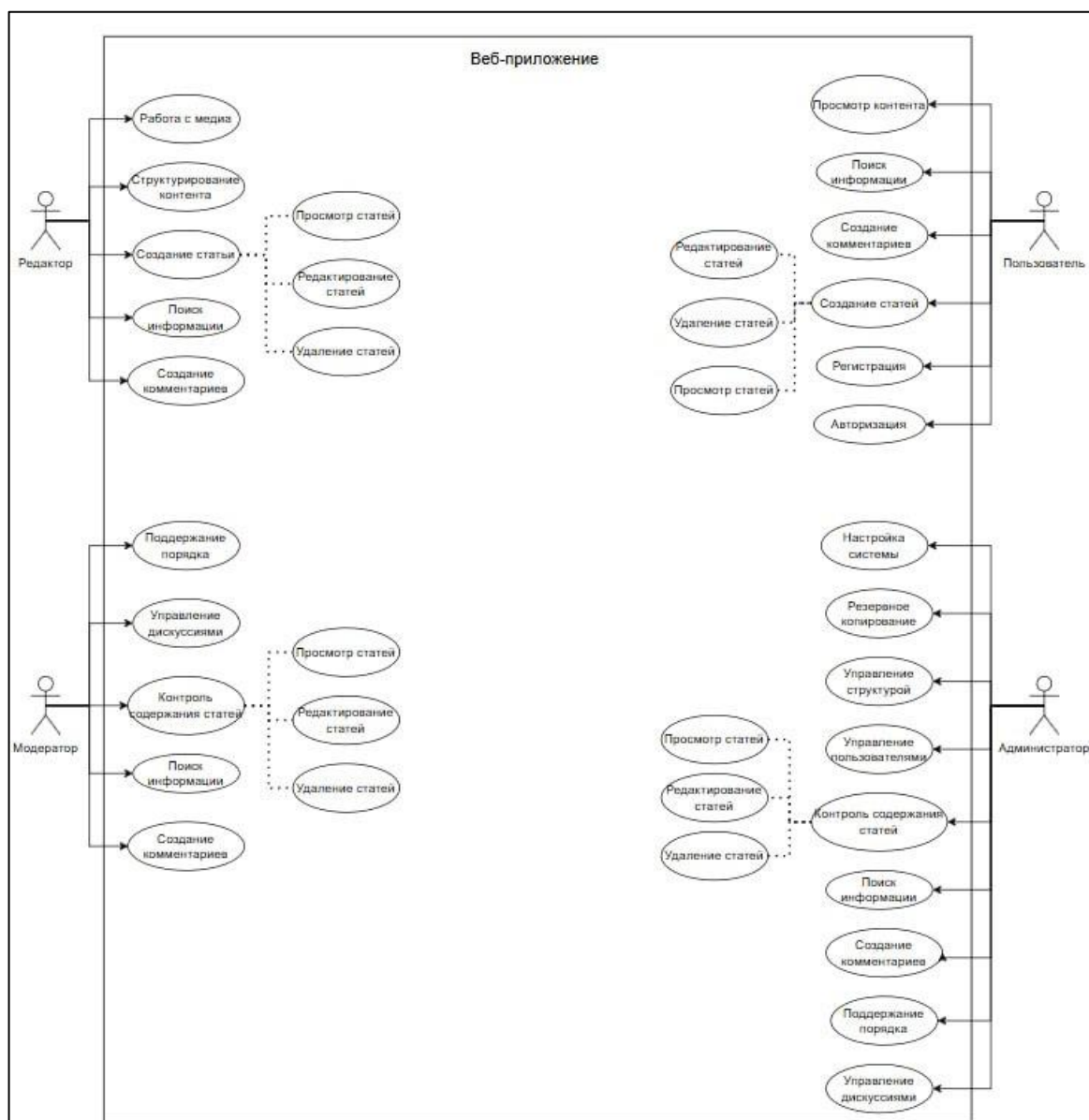


Рисунок 1 – Диаграмма прецедентов

На диаграмме представлены четыре основных прецедента:

Изм.	Лист	№ докум.	Подпись	Дата

– пользователь: может просматривать статью (медиа контент, общую информацию), может выполнять основные процессы: поиск информации, создание статьи: редактирование и удаление, написание комментариев к статьям, добавление статьи в избранное.

– Администратор: имеет полный доступ к системе, включая управление контентом (статьи, комментарии) и управление пользователями.

– Модератор: имеет ограниченный доступ к статьям, может редактировать статьи, просматривать статьи комментарии к ним, удаление комментариев, нарушающие правила форума.

– Редактор: может редактировать статью, просматривать медиа контент, структуру статьи, также может просматривать и удалять комментарии.

Диаграмма наглядно показывает, что система разделена на клиентскую часть (для пользователей) и административную часть. Все ключевые функции, описанные в техническом задании и реализованные в коде (просмотр статьи и медиа, поиск информации, создание/редактирование статьи), отражены в виде прецедентов.

Диаграмма деятельности показывает поток выполнения операций в системе, в данном случае – процесс создания статьи. Диаграмма изображена на рисунке 2.

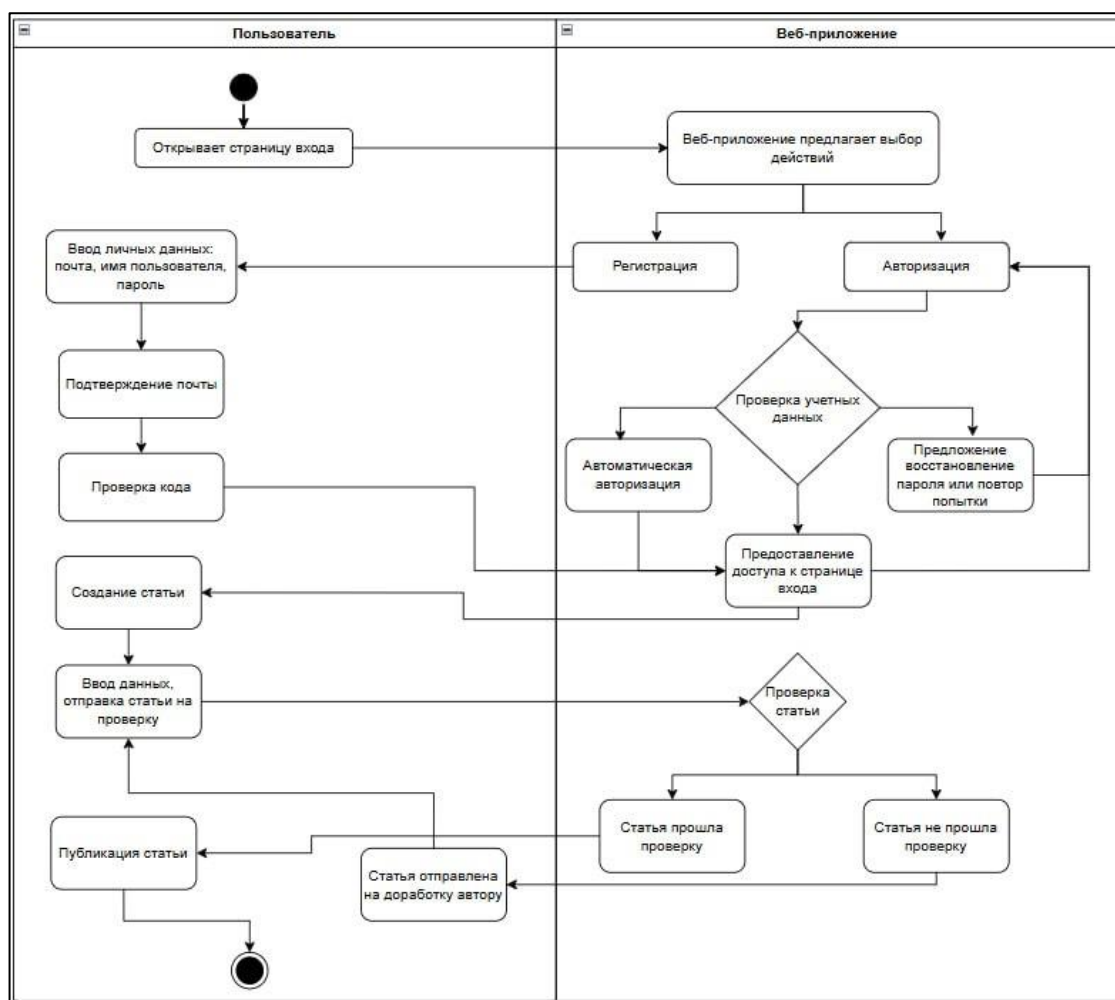


Рисунок 2 – Диаграмма деятельности

Процесс начинается с регистрации или авторизации. После успешной аутентификации пользователь может создать статью. После заполнения информации, статья автоматически отправляется на проверку модератору. Модератор может опубликовать статью, если статья корректна написана и не имеет ошибок, или же модератор может выделить ошибки в статье, написать примечания и отправить редактору для редактирования. Редактор после внесения нужных изменений отправляет статью пользователю на согласование публикации, если пользователь согласен с внесенными изменениями – статья опубликуется, также пользователь может изменить статью и отправить на повторную проверку, также пользователь может отказаться от внесенных изменений и статья удалиться.

Диаграмма детализирует основной бизнес-процесс приложения – создание статьи. Диаграмма демонстрирует последовательность шагов, точки принятия решений (проверка авторизации, редактирование статьи) и ключевые операции, написание комментария, добавление статьи в избранное, что соответствует реализованному функционалу.

Диаграмма показывает структурные компоненты веб-приложения и связи между компонентами в виде информационных объектов: файлов, модулей, библиотек, пакетов. Диаграмма изображена на рисунке 3.

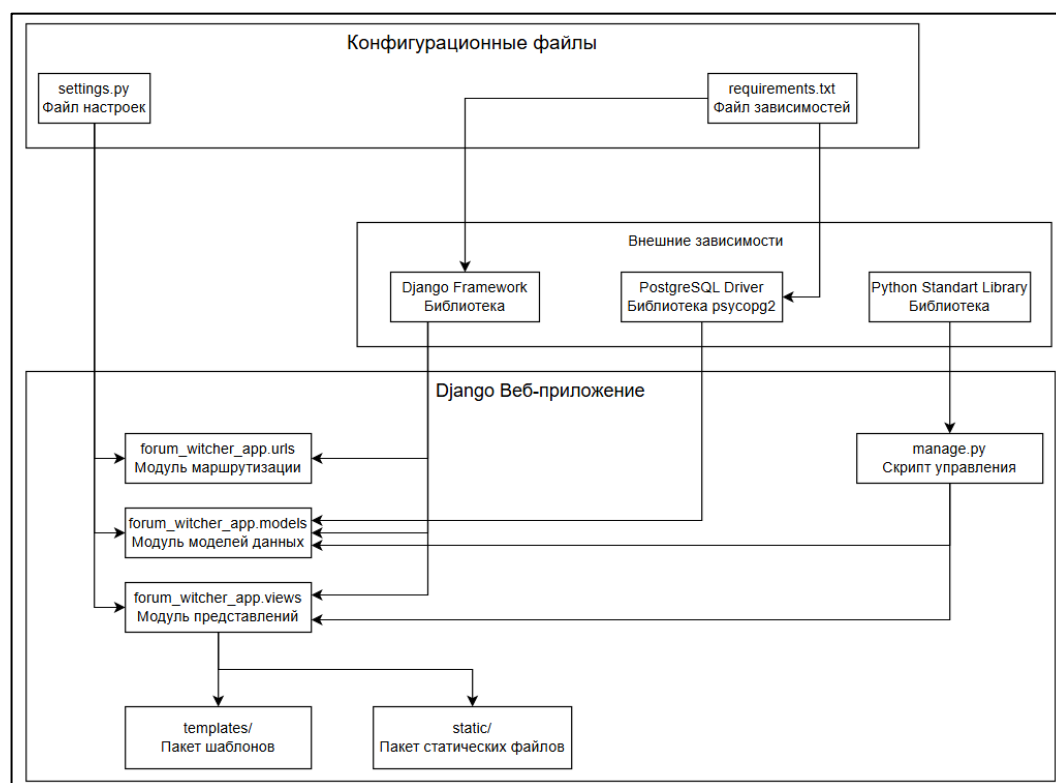


Рисунок 3 – Диаграмма компонентов

Описание компонентов:

- forum_witcher_app.models – модуль, содержащий модели данных (Article, Category, Comment, UserProfile, SearchQuery, User)
- forum_witcher_app.views – модуль бизнес-логики, обработчики запросов
- forum_witcher_app.urls – модуль маршрутизации URL

- templates/ – пакет HTML-шаблонов
- static/ – пакет статических файлов (CSS, JS, изображения)
- manage.py – скрипт управления Django-приложением
- settings.py – конфигурационный файл настроек приложения
- requirements.txt – файл зависимостей проекта
- Django Framework – основная библиотека веб-фреймворка
- PostgreSQL Driver – библиотека для работы с PostgreSQL
- Python Standard Library – стандартная библиотека Python

Диаграмма компонентов демонстрирует модульную архитектуру системы, где каждый компонент выполняет строго определенную функцию. Четкое разделение на модели, представления, шаблоны и статические файлы соответствует принципам MVC/MVT. Зависимости от внешних библиотек явно указаны, что облегчает развертывание и сопровождение системы.

Диаграмма показывает аппаратные компоненты «узлы» и программные компоненты «артефакты», работающие на каждом узле. Диаграмма изображена на рисунке 4.

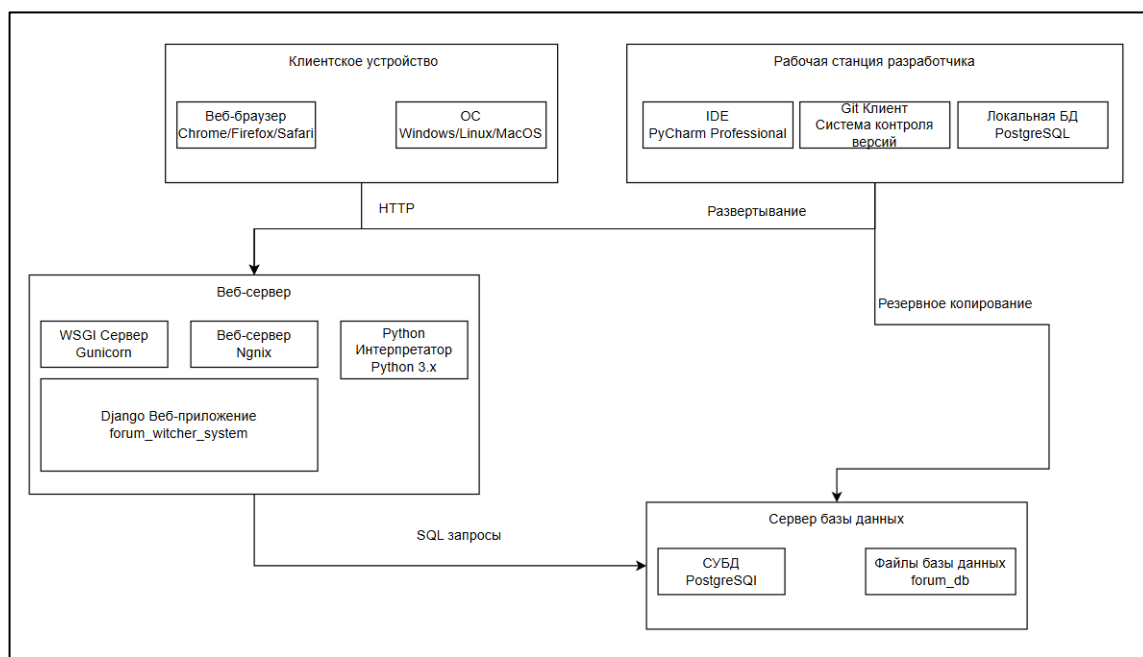


Рисунок 4 – Диаграмма развертывания

Изм.	Лист	№ докум.	Подпись	Дата

Описание узлов и артефактов:

Узел: Клиентское устройство

- Артефакт: Веб-браузер – пользовательский интерфейс системы,
- Артефакт: Операционная система – платформа для работы браузера.

Узел: Веб-сервер

- Артефакт: WSGI Сервер (Gunicorn) – обработчик Python-приложений,
- Артефакт: Веб-сервер (Nginx) – статические файлы и прокси,
- Артефакт: Django Приложение – основная бизнес-логика,
- Артефакт: Python Интерпретатор – среда выполнения приложения.

Узел: Сервер базы данных

- Артефакт: СУБД PostgreSQL – система управления базами данных,
- Артефакт: Файлы базы данных – физическое хранилище данных.

Узел: Рабочая станция разработчика

- Артефакт: IDE PyCharm – среда разработки,
- Артефакт: Git Клиент – контроль версий кода,
- Артефакт: Локальная БД – база данных для разработки.

Диаграмма развертывания показывает трехзвенную архитектуру системы (клиент-сервер-БД), что обеспечивает масштабируемость и надежность. Разделение веб-сервера и сервера базы данных на разные узлы позволяет независимо масштабировать вычислительную мощность и ресурсы хранения данных. Использование Nginx в качестве фронтенд-сервера и Gunicorn в качестве бэкенд-сервера соответствует лучшим практикам развертывания Django-приложений.

4.2 Функциональная схема веб-приложения

Контекстная диаграмма – это высокоуровневое визуальное представление, которое показывает взаимодействия между разрабатываемой системой и внешними объектами. Контекстная диаграмма изображена на рисунке 5.

					КП.09.02.07–1.25.221.20.ПЗ	Лист 23
Изм.	Лист	№ докум.	Подпись	Дата		

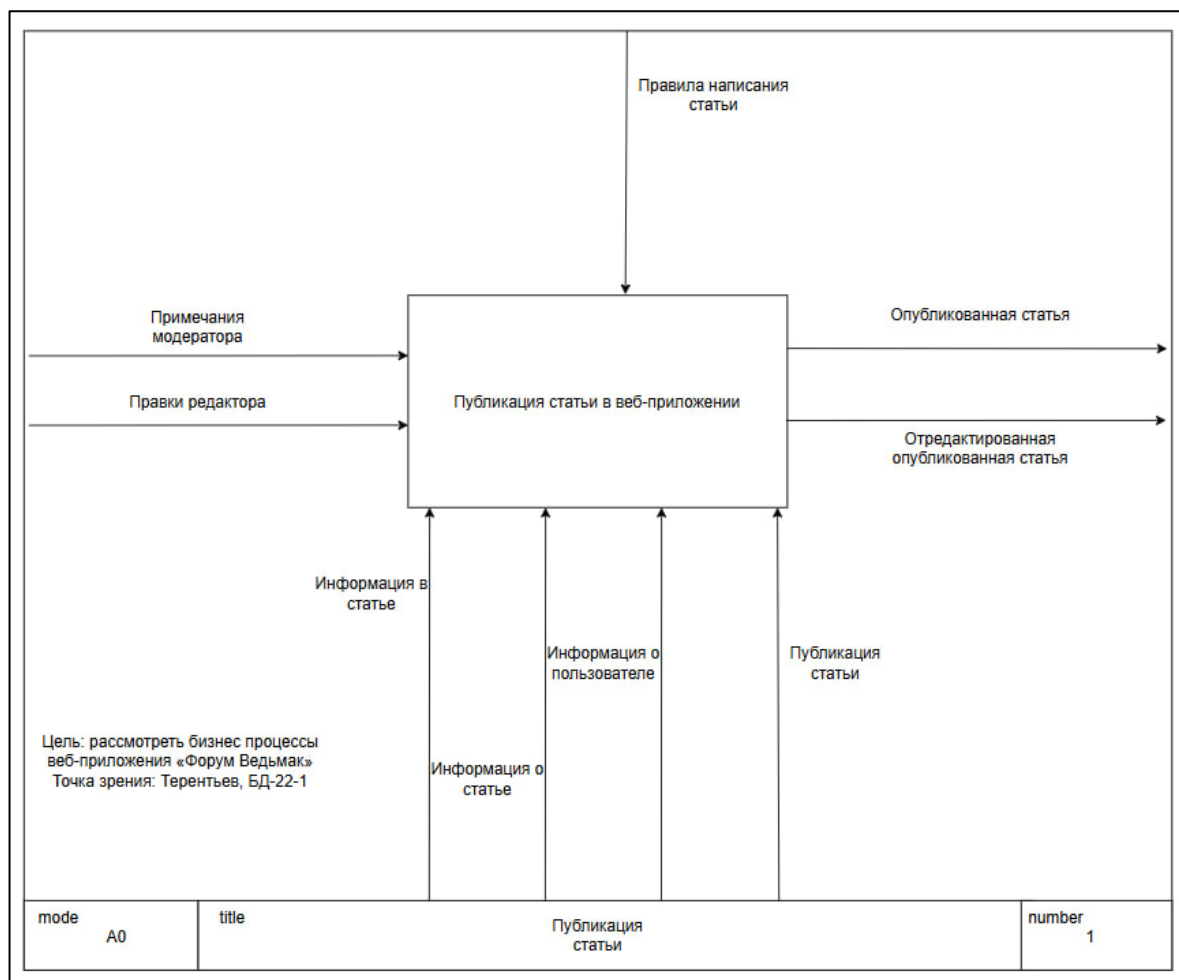


Рисунок 5 – Контекстная диаграмма

Диаграмма показывает, что система получает информацию о статье, обрабатывает согласно установленным правилам написания статьи и публикует статью.

Диаграмма декомпозиций (A1) предназначена для детализации работы, то есть декомпозицией называется разделение бизнес-процессов на более мелкие составляющие. Диаграмма изображена на рисунке 6.

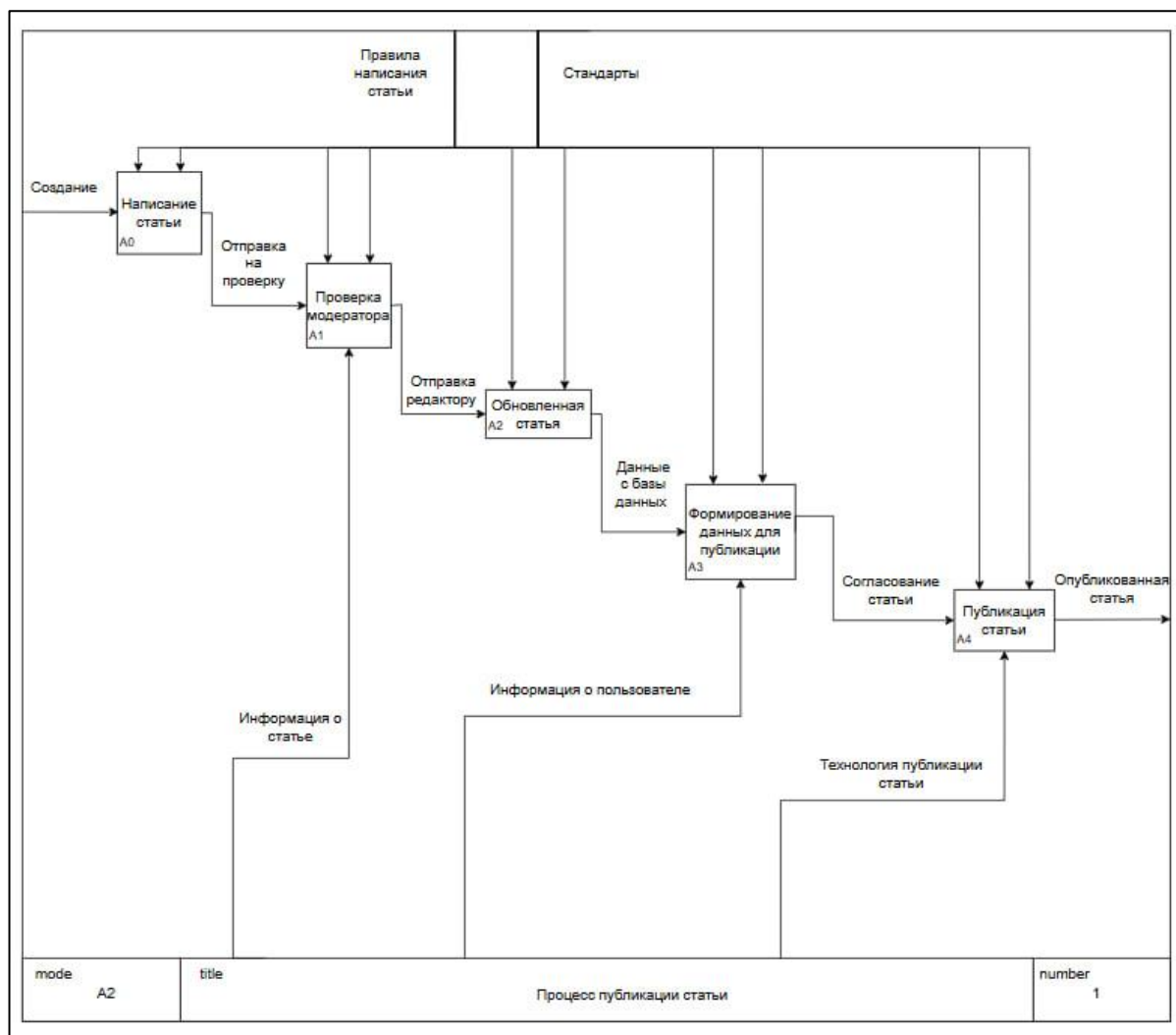


Рисунок 7 – Диаграмма декомпозиций A2

Диаграмма A2 демонстрирует детальную последовательность операций при публикации статьи. Четко прослеживается разделение на логические этапы: написание статьи, проверка, формирование обновленной статьи и публикация. Такая декомпозиция позволяет эффективно распределить ответственность между модулями системы и обеспечивает прозрачность бизнес-процесса.

Диаграмма классов показывает статичную структуру системы, демонстрируя классы, атрибуты, поведение и связь друг с другом. Диаграмма изображена на рисунке 8.

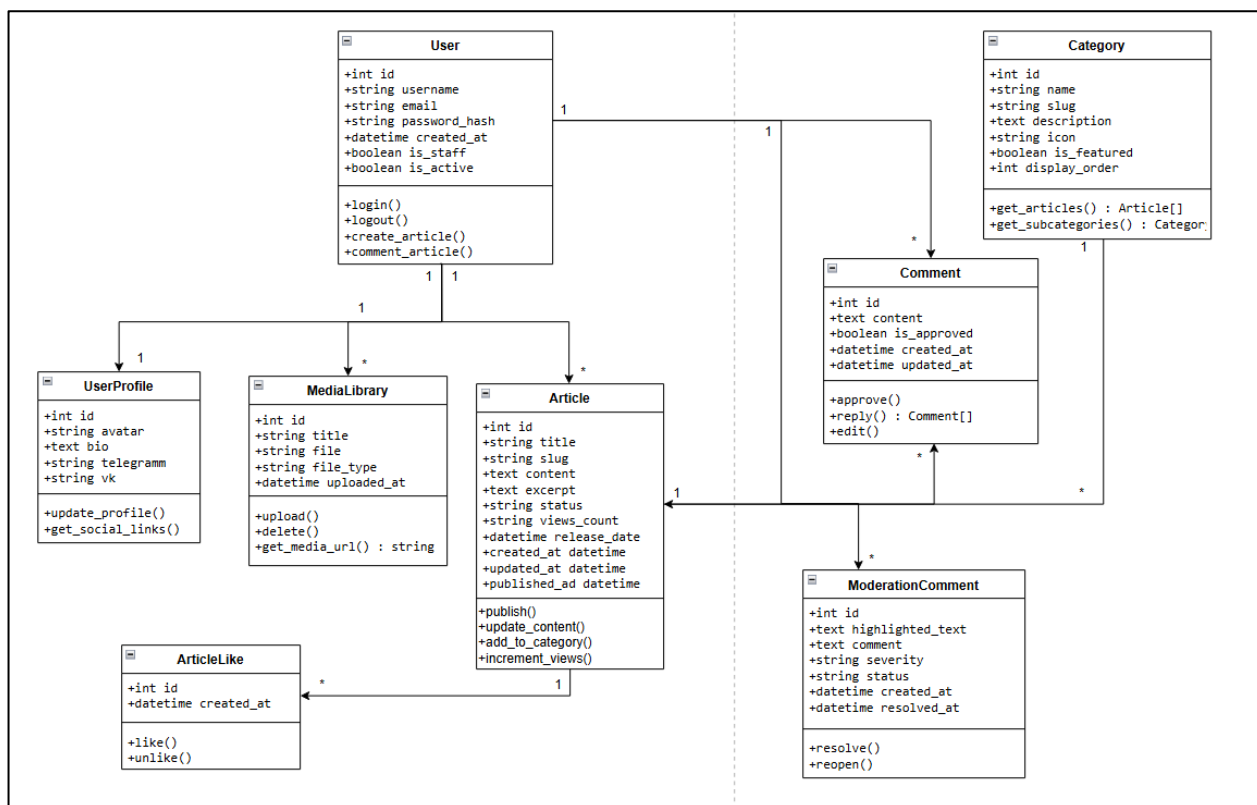


Рисунок 8 – Диаграмма классов

Диаграмма классов представляет статическую структуру данных системы:

- User – модель пользователя с аутентификацией и управлением контентом.
- UserProfile – расширенный профиль пользователя с социальными связями и аватаром.
- Article – сущность статьи с метаданными, контентом и системой версий.
- Category – категоризация контента по темам вселенной Ведьмака.
- Comment – система комментариев с древовидной структурой и модерацией.
- ArticleLike – механизм оценки контента через лайки.
- ModerationComment – система модерации с выделением текста и отслеживанием статуса.
- MediaLibrary – медиа-библиотека для хранения изображений и файлов.

Связи отображают бизнес-логику: пользователь создает статьи и комментарии, статьи принадлежат категориям, комментарии образуют древовидную структуру, система лайков и модерации обеспечивает взаимодействие и контроль качества контента, медиа–библиотека поддерживает визуальное оформление статей.

Диаграмма классов демонстрирует хорошо продуманную объектную модель, соответствующую требованиям тематического форума. Четкое разделение ответственности между классами, наличие необходимых атрибутов и методов, а также логичные связи между сущностями обеспечивают основу для устойчивой и расширяемой архитектуры системы.

Диаграмма потоков данных показывает движение данных в информационной системе: откуда данные, поступают, как обрабатываются, где хранятся и передаются. Диаграмма изображена на рисунке 9.

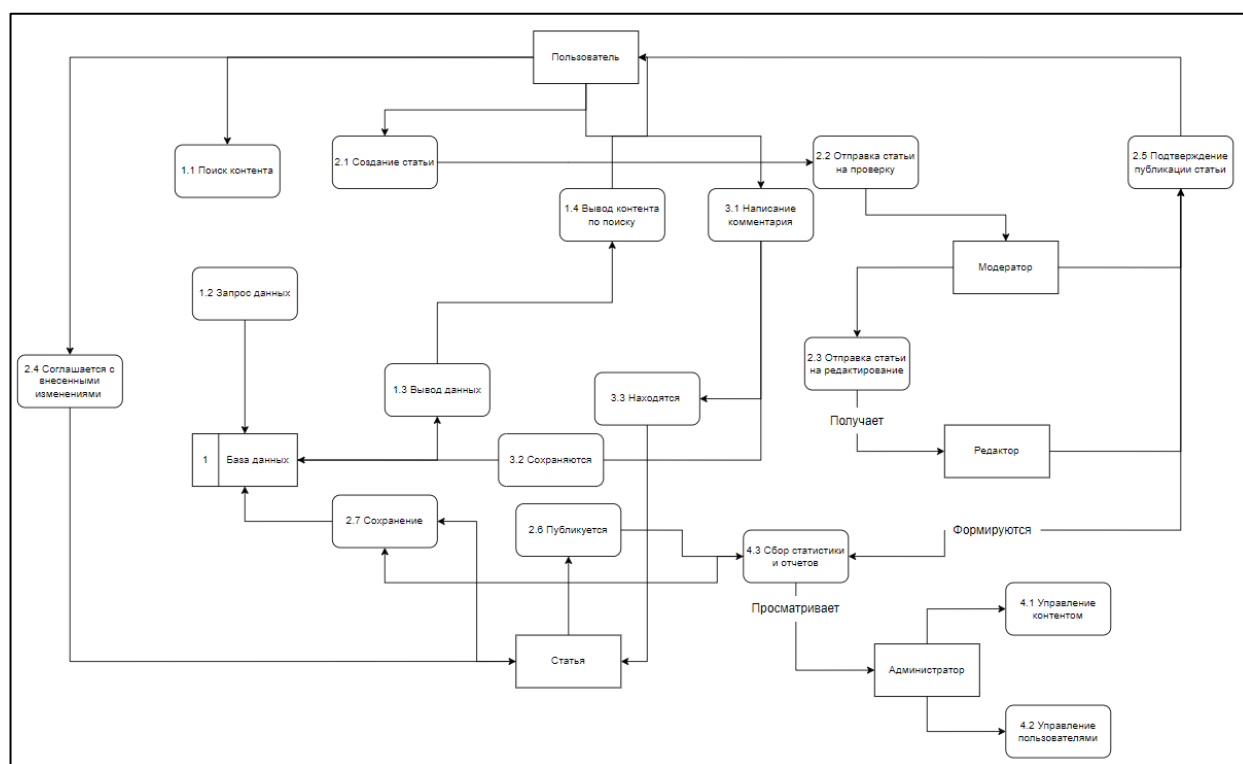


Рисунок 9 – Диаграмма потоков данных DFD

Диаграмма потоков данных (DFD) показывает движение информации в системе:

- Внешние сущности: пользователь, модератор, редактор, администратор.
- Процессы: Веб-приложение форум (центральный обработчик всех операций)
- Потоки данных: поисковые запросы и результаты поиска, создание и публикация статей, комментарии и обратная связь.
- Хранилища данных: База данных PostgreSQL.

Диаграмма иллюстрирует полный жизненный цикл контента: от создания пользователем, через процессы модерации и редактирования, до финальной публикации и хранения в базе данных.

Разработанный комплект диаграмм полностью охватывает функциональную схему информационной системы:

- IDEF0 диаграммы (A0, A1, A2) эффективно отображают иерархию бизнес-процессов, начиная от контекстного представления и заканчивая детализированными операциями управления контентом, модерации и взаимодействия с пользователями.
- Диаграмма классов четко определяет структуру данных системы, показывая сущности (статьи, пользователи, комментарии, категории), атрибуты, методы и взаимосвязи, что полностью соответствует реализованным в проекте моделям Django и таблицам базы данных.
- DFD диаграмма наглядно демонстрирует потоки информации между внешними сущностями и системой, подчеркивая роль каждого участника в процессе создания и управления контентом.

Все диаграммы взаимосвязаны и непротиворечивы: сущности из диаграммы классов соответствуют данным, обрабатываемым в DFD, а бизнес-процессы IDEF0 отражают функциональность, реализованную в методах классов веб-приложения. Такой комплексный подход к проектированию обеспечивает полное понимание

архитектуры системы форума и облегчает дальнейшую разработку, масштабирование и сопровождение проекта.

4.3 Проектирование базы данных

Перед началом разработки программного обеспечения важно спроектировать базу данных, определив, с какими данными будут работать пользователи системы и как данные взаимосвязаны. Инфологическая модель базы данных изображена на рисунке 10.

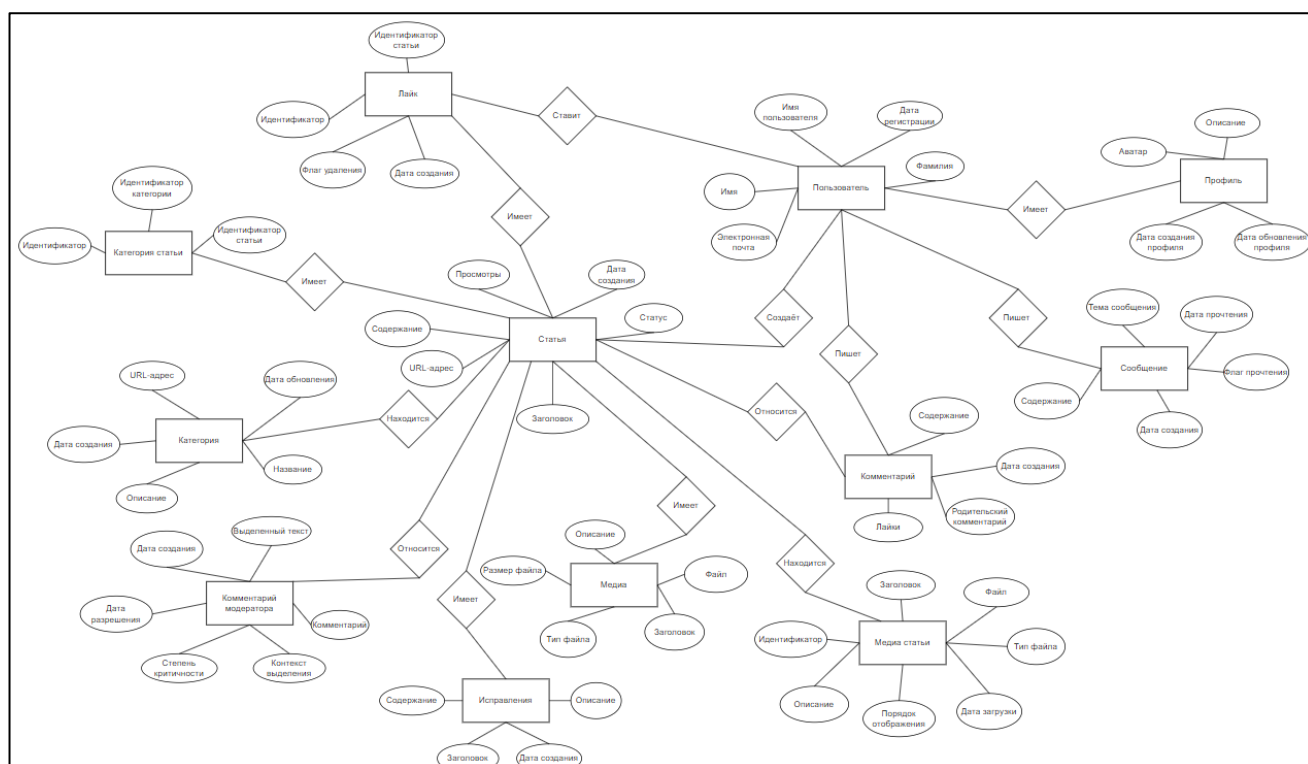


Рисунок 10 – Инфологическая модель базы данных

Инфологическая модель представляет семантическое описание предметной области в терминах сущностей и отношений:

- USER – сущность «Пользователь» с персональными данными и учетной информацией.
- ARTICLE – сущность «Статья» с основным контентом форума.

– PROFILE – сущность «Профиль» с дополнительными данными пользователя.

– CATEGORY – сущность «Категория» для тематической организации контента.

– COMMENT – сущность «Комментарий» для обсуждения статей.

– MEDIA – сущность «Медиа» для хранения файлов и изображений.

– LIKE – сущность «Лайк» для добавления статьи в избранное.

– PROFILE – сущность «Профиль» для отображения личной информации пользователя.

– ARTICLE_MEDIA – сущность «Медиа статьи» для сохранения файлов.

– MESSAGE – сущность «Сообщение» для личного общения между пользователями.

– REVISIONS – сущность «Исправления» для удобства модерации.

– MODERATION_COMMENTS – сущность «Комментарии модератора» для удобства редактирования статьи.

Связи между сущностями:

– Пользователь имеет один профиль (1:1).

– Пользователь создает несколько статей (1:M).

– Пользователь пишет несколько комментариев (1:M).

– Пользователь загружает несколько медиафайлов (1:M).

– Статья относится к нескольким категориям (M:M).

– Статья содержит несколько комментариев (1:M).

– Статья включает несколько медиафайлов (1:M).

– Категория содержит несколько статей (M:M).

– Комментарий может иметь ответы (1:M).

– Категория может иметь подкатегории (1:M).

– Статья может иметь множество лайков (1:M).

– Пользователи могут отправлять множество сообщений (M:M).

– Статья может иметь множество исправлений (1:M).

- Статья может иметь множество комментариев модератора (1:M).

Датологическая диаграммы изображена на рисунке 11.

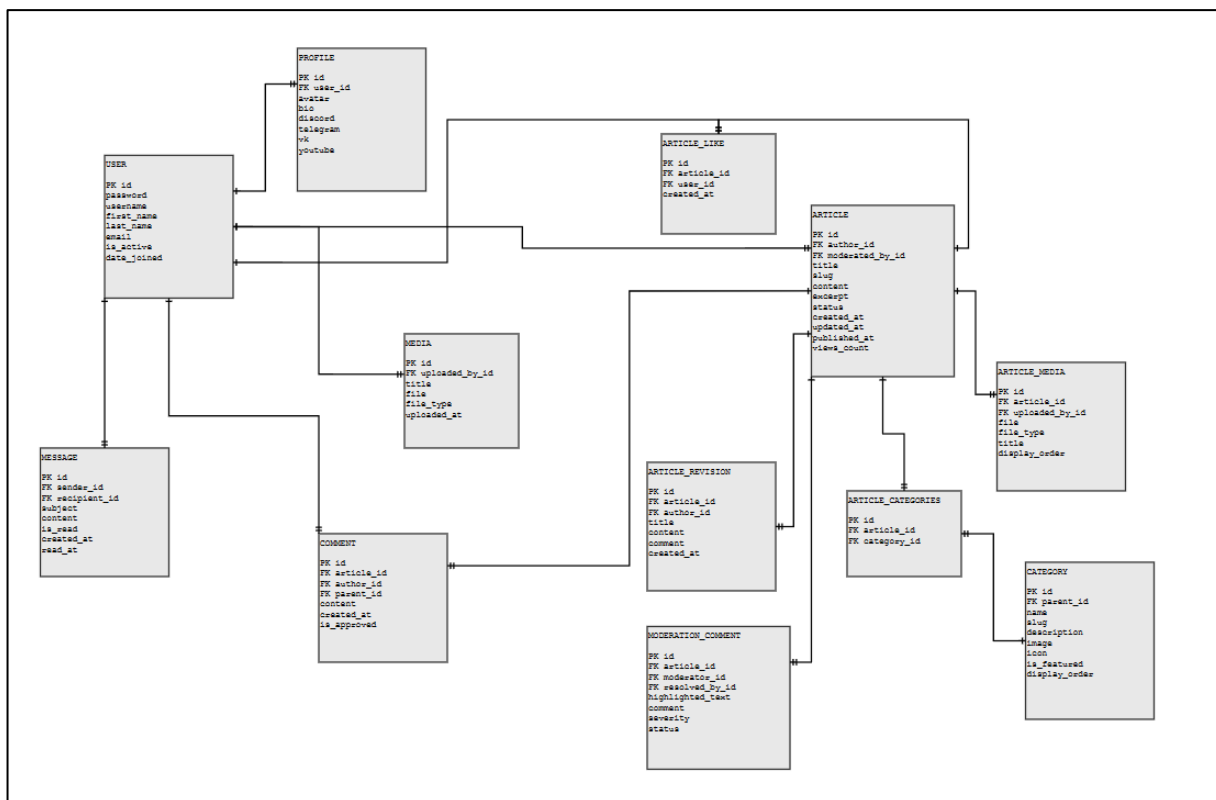


Рисунок 11 – Датологическая модель базы данных

- Типы данных: определены конкретные SQL-типы (VARCHAR, INT, DECIMAL, DATETIME, BLOB).
- Ограничения: добавлены CHECK-ограничения для валидации данных.
- Индексы: определены уникальные индексы и первичные ключи.
- Внешние ключи: явно указаны FOREIGN KEY ограничения.
- Значения по умолчанию: установлены DEFAULT значения для временных меток.

ER-модель изображена на рисунке 12.

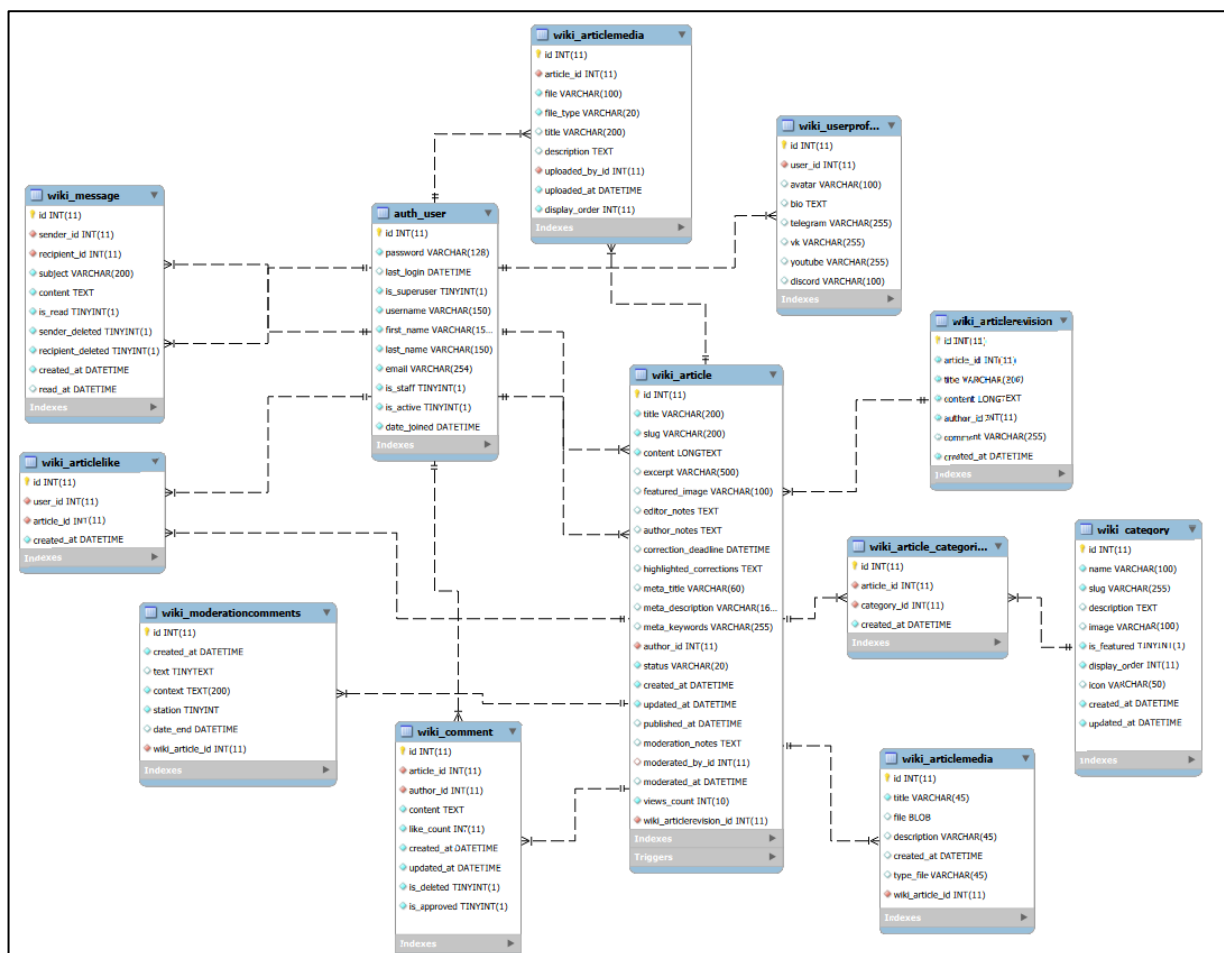


Рисунок 12 – ER-модель базы данных

Представленная ER-модель соответствует третьей нормальной форме:

1. Первая нормальная форма: все атрибуты атомарны, нет повторяющихся групп.
2. Вторая нормальная форма: нет частичных зависимостей от составного ключа (все сущности имеют простой первичный ключ).
3. Третья нормальная форма: нет транзитивных зависимостей - все неключевые атрибуты зависят только от первичного ключа.

Обоснование выбора 3NF:

- Устраняет избыточность данных.
- Обеспечивает целостность данных.
- Упрощает поддержку и модификацию БД.

Изм.	Лист	№ докум.	Подпись	Дата

База данных веб-приложения создана и реализована в СУБД PostgreSQL и состоит из 12 таблиц.

Таблица 1 – Таблица «User»

Поле	Тип данных	Описание
id	Int	Идентификатор
password	Char(128)	Пароль
last_login	DateTime	Дата последнего входа
is_superuser	Boolean	Флаг суперпользователя
username	Char(150)	Имя пользователя
first_name	Char(150)	Имя
last_name	Char(150)	Фамилия
email	Char(120)	Электронная почта
is_staff	Boolean	Флаг персонала
is_active	Boolean	Флаг активности
date_joined	DateTime	Дата регистрации

Таблица 6 – Таблица «Profile»

Поле	Тип данных	Описание
id	Int	Идентификатор
avatar	Blob	Аватар
bio	Text	Биография
telegram	Char(100)	Telegram
vk	Char(100)	ВКонтакте
user_id	Int	Идентификатор
user_id	Int	Идентификатор пользователя

Таблица 7 – Таблица «Category»

Поле	Тип данных	Описание
id	Int	Идентификатор
name	Char(100)	Название
slug	Char(50)	URL-адрес
description	Text	Описание
created_at	DateTime	Дата создания
image	Char(100)	Изображение
updated_at	DateTime	Дата обновления

Таблица 8 – Таблица «Article»

Поле	Тип данных	Описание
id	Int	Идентификатор
title	Char(200)	Заголовок
slug	Char(200)	URL-адрес
content	Text	Содержание
excerpt	Text	Краткое описание
featured_image	Char(100)	Главное изображение

Продолжение таблицы 8

meta_title	Char(60)	Мета-заголовок
meta_description	Char(160)	Мета-описание
meta_keywords	Char(255)	Мета-ключевые слова
status	Char(20)	Статус
created_at	DateTime	Дата создания
updated_at	DateTime	Дата обновления
published_at	DateTime	Дата публикации
views_count	Int	Количество просмотров
author_id	Int	Идентификатор автора
moderated_at	DateTime	Дата модерации
moderated_by_id	Int	Идентификатор модератора
moderation_notes	Text	Заметки модерации
author_notes	Text	Заметки автора
correction_deadline	DateTime	Срок исправлений
editor_notes	Text	Заметки редактора
highlighted_corrections	JSONB	Выделенные исправления

Таблица 9 – Таблица «Comment»

Поле	Тип данных	Описание
id	Int	Идентификатор
content	Text	Содержание
created_at	DateTime	Дата создания
is_approved	Boolean	Флаг одобрения
article_id	Int	Идентификатор статьи
author_id	Int	Идентификатор автора
updated_at	DateTime	Дата обновления
article_id	Int	Идентификатор статьи
author_id	Int	Идентификатор автора
parent_id	Int	Идентификатор родительского комментария

Таблица 10 – Таблица «Article_Media»

Поле	Тип данных	Описание
id	Int	Идентификатор медиа
file	Char(100)	Файл
file_type	Char(20)	Тип файла
title	Char(200)	Заголовок
description	Text	Описание
uploaded_at	DateTime	Дата загрузки
display_order	Int	Порядок отображения
article_id	Int	Идентификатор статьи
uploaded_by_id	Int	Идентификатор загрузившего

Таблица 11 – Таблица «Article_Revision»

Поле	Тип данных	Описание
------	------------	----------

Продолжение таблицы 11

id	Int	Идентификатор ревизии
title	Char(200)	Заголовок
content	Text	Содержание
comment	Char(255)	Комментарий к изменению
created_at	DateTime	Дата создания
article_id	Int	Идентификатор статьи
author_id	Int	Идентификатор автора

Таблица 12 – Таблица «Media»

Поле	Тип данных	Описание
id	Int	Идентификатор медиа
title	Char(200)	Заголовок
file	Char(100)	Файл
file_type	Char(50)	Тип файла
uploaded_at	DateTime	Дата загрузки
uploaded_by_id	Int	Идентификатор загрузившего

Таблица 13 – Таблица «Moderation_Comment»

Поле	Тип данных	Описание
id	Int	Идентификатор
highlighted_text	Text	Выделенный текст
comment	Text	Комментарий
start_position	Int	Начальная позиция
end_position	Int	Конечная позиция
created_at	DateTime	Дата создания
resolved_at	DateTime	Дата разрешения
selection_context	Text	Контекст выделения
severity	Char(20)	Серьезность
status	Char(20)	Статус
updated_at	DateTime	Дата обновления
article_id	Int	Идентификатор статьи
moderator_id	Int	Идентификатор модератора
resolved_by_id	Int	Идентификатор разрешившего

Таблица 14 – Таблица «Message»

Поле	Тип данных	Описание
id	Int	Идентификатор
subject	Char(200)	Тема
content	Text	Содержание
is_read	Boolean	Флаг прочтения
sender_deleted	Boolean	Флаг удаления отправителем
recipient_deleted	Boolean	Флаг удаления получателем
created_at	DateTime	Дата создания
read_at	DateTime	Дата прочтения

Продолжение таблицы 14

recipient_id	Int	Идентификатор получателя
sender_id	Int	Идентификатор отправителя

Таблица 15 – Таблица «Article_Like»

Поле	Тип данных	Описание
id	Int	Идентификатор
created_at	DateTime	Дата создания
article_id	Int	Идентификатор статьи
user_id	Int	Идентификатор пользователя

Таблица 16 – Таблица «Article_Categories»

Поле	Тип данных	Описание
id	Int	Идентификатор
article_id	Int	Идентификатор статьи
category_id	Int	Идентификатор категории

Спроектированная база данных полностью соответствует требованиям предметной области вики-форума. ER-модель в третьей нормальной форме обеспечивает оптимальную структуру для хранения данных, минимизируя избыточность и обеспечивая целостность. Четко определенные связи между сущностями позволяют эффективно реализовать все бизнес-процессы системы, от публикации статей до модерации контента и взаимодействия пользователей. Выбранные типы данных и ограничения гарантируют корректное хранение информации и предотвращают возникновение противоречивых состояний в базе данных.

4.4 Проектирование интерфейса

Для проектирования пользовательского интерфейса веб-приложения «Форум» использовался профессиональный онлайн-инструмент Draw.io. Данный инструмент был выбран за кроссплатформенность, простоту использования, широкий набор элементов для построения схем и удобство совместной работы, что позволило быстро и наглядно создать прототипы всех основных экранов системы.

Были разработаны прототипы всех ключевых окон системы, соответствующих функциональным требованиям, предъявляемым к клиентской и административной частям приложения. Страница Login изображена на рисунке 13.

The diagram shows a wireframe for a login page. It consists of a large outer rectangle containing a smaller rounded rectangle. Inside the rounded rectangle, the elements are arranged vertically: a title, a label followed by a text input field, another label followed by a text input field, a wide button, and finally a text link followed by a button.

```
graph TD
    Title[title]
    Label1[label]
    FormName[form.name]
    Label2[label]
    FormPassword[form.password]
    Button[button]
    HelpText[help_text]
    RegisterButton[button_to_register]

    Title --- Label1
    Label1 --- FormName
    FormName --- Label2
    Label2 --- FormPassword
    FormPassword --- Button
    Button --- HelpText
    HelpText --- RegisterButton
```

Рисунок 13 – Страница Login

Рисунок 13 – Страница авторизации (Login). Содержит стандартную форму для ввода учетных данных (имя пользователя и пароль), кнопку для входа в систему, а также ссылки для восстановления пароля и регистрации нового пользователя. Дизайн выполнен в минималистичном стиле, фокусирующем внимание на процессе аутентификации. Страница Register изображена на рисунке 14.

logo

title

label

form.name

label

form.email

label

form.password1

label

form.password1.help_text

form.password2

label

button

help_text

button_to_register

Рисунок 14 – Страница Register

Рисунок 14 – Страница регистрации (Register). Прототип формы создания новой учетной записи. Пользователю предлагается ввести основные данные для регистрации: имя пользователя, адрес электронной почты, пароль. После успешного заполнения формы пользователь создает учетную запись и получает возможность войти в систему. Страница Home изображена на рисунке 15.

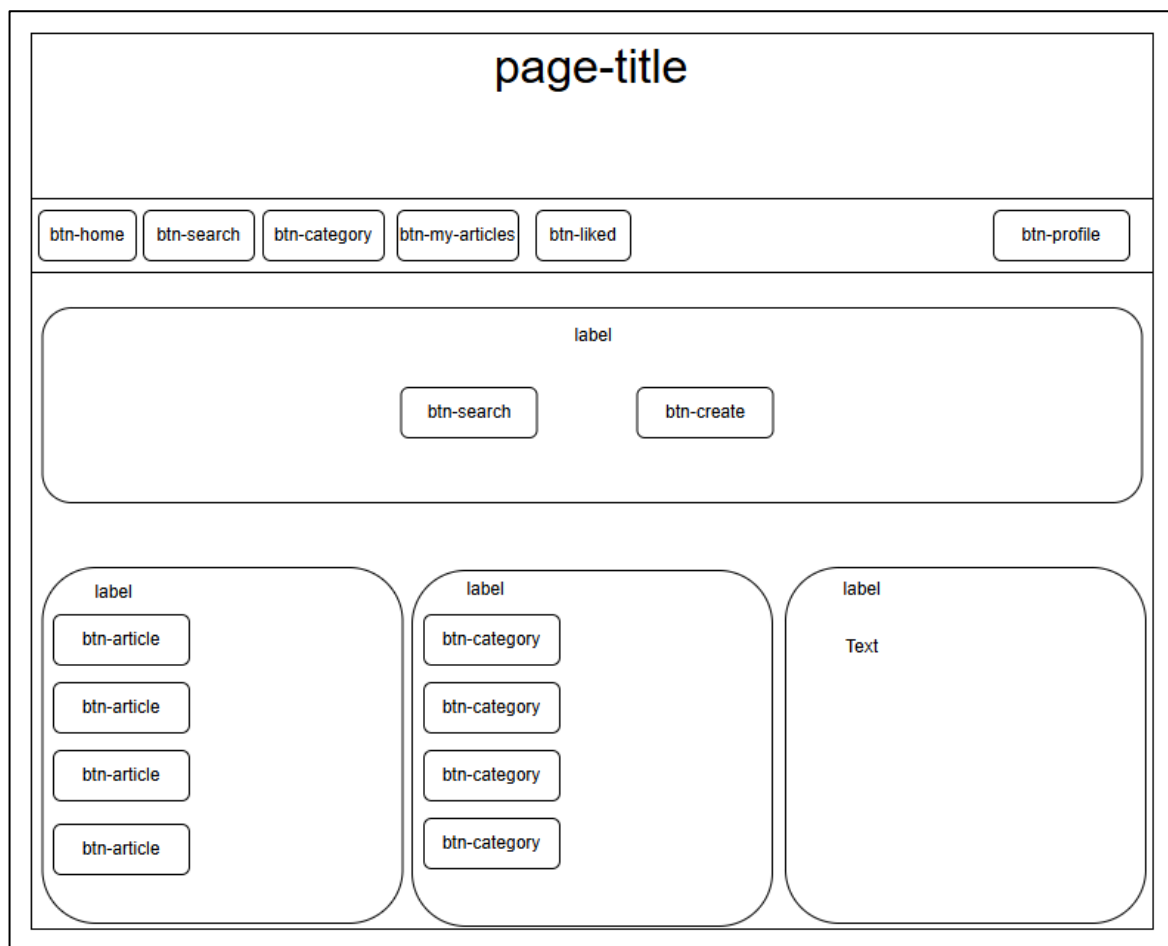


Рисунок 15 – Страница Home

Рисунок 15 – Главная страница (Home). Является центральным узлом для клиента после авторизации. На странице представлены последние статьи, категории, статьи пользователя и понравившиеся статьи. Каждая статья находится в подходящей категории. Страница Profile изображена на рисунке 16.

profile

page-title

profile-avatar

btn-select-file

Text

btn-save

name

email

date-of-registration

social-network

telegram

vkontakte

settings-profile

telegram-link

text

vkontakte-link

text

about-you

text

save-settings

Рисунок 16 – Страница Profile

Рисунок 16 – Страница профиля пользователя (Profile). Представляет личный кабинет зарегистрированного пользователя. Здесь отображается персональная информация (имя, email), также опубликованные статьи пользователя, понравившиеся статьи и общее количество просмотров. Страница Create-Article изображена на рисунке 17.

page-title

article_title

text

description

text

content_article

text

categories

text

text

text

text

mediafiles

btn-select-files

btn-publish-article

Рисунок 17 – Страница Create-Article

Рисунок 18 – Страница создания статьи (Create-Article): После регистрации пользователь может создать свою собственную статью, пользователь открывает страницу создания, заполняет статью согласно правилам, отправляет на проверку и, если статья полностью соответствует правилам, то публикуется на главной странице, с указанием даты создания и автора данной статьи.

Разработанный в Draw.io прототип полностью охватывает все запланированные сценарии использования веб-приложения «Форум» как для

клиентов, так и для администраторов. Интерфейс спроектирован интуитивно понятным и удобным для конечного пользователя, с четкой последовательностью шагов от просмотра статьи до создания статьи. Прототип наглядно демонстрирует строгое разделение интерфейсов и функционала для клиента и администратора, что является ключевым требованием к системе.

Использование прототипа на раннем этапе позволило утвердить логику взаимодействия, визуальную структуру и навигацию до начала этапа фронтенд-разработки, что способствовало снижению количества доработок на поздних стадиях проекта.

					КП.09.02.07–1.25.221.20.ПЗ	Лист
						43
Изм.	Лист	№ докум.	Подпись	Дата		

5 Разработка веб-приложения

5.1 Разработка интерфейса веб-приложения

Веб-приложение «Форум» было разработано с использованием фреймворка Django. Реализация интерфейса выполнена с акцентом на пользовательский опыт, безопасность и производительность. Структура шаблонов соответствует архитектуре MVT.

Все функции, указанные в техническом задании, были реализованы полностью, создав целостную экосистему для онлайн-бронирования:

1. Система аутентификации и авторизации:

- кастомная модель пользователя.
- Email-верификация при регистрации.
- Восстановление пароля.
- Разграничение ролей.
- Авторизация по логину и паролю.

2. Основной функционал для пользователей:

- просмотр списка статей с пагинацией и сортировкой.
- Фильтрация статей по тегам, категориям, дате публикации.
- Детальный просмотр статьи с полным содержимым и изображениями.
- Комментирование статей с поддержкой форматирования.
- Система лайков/дизлайков для статей и комментариев.
- Поиск по форуму (полнотекстовый поиск по заголовкам и содержимому).
- Личный кабинет с историей активности (статьи, комментарии, лайки).
- Создание и редактирование своих статей через редактор.

3. Административный функционал:

- crud операции для статей (создание, чтение, обновление, удаление).
- Управление пользователями через Django admin-панель.
- Управление комментариями (модерация, удаление, скрытие).

- Система отчетности по активности пользователей.
- Логирование действий в системе (кто, что и когда сделал).
- Управление тегами и категориями контента.
- Экспорт данных в PDF/Excel форматах.

4. Дополнительные функции:

- смена email с подтверждением через код верификации.
- Смена пароля с проверкой старого пароля.
- Email-уведомления о новых комментариях к статьям.
- Поддержка загрузки изображений в статьи.
- Система предпросмотра статей перед публикацией.
- Пагинация для списков статей и комментариев.

На рисунке 18 изображён код просмотра статей.

```
<div class="article-container">
  <div class="article-header">
    <h1 class="article-title">{{ article.title }}</h1>

    <div class="meta-info">
      <div class="meta-item">
        <div class="meta-label">Автор</div>
        <div class="meta-value">
          <a href="{% url 'wiki:user_public_profile' article.author.username %}" class="author-link">
            {{ article.author.username }}
          </a>
        </div>
      </div>
      <div class="meta-item">
        <div class="meta-label">Опубликовано</div>
        <div class="meta-value">{{ article.created_at|date:"d.m.Y H:i" }}</div>
      </div>
      {% if article.views_count %}
      <div class="meta-item">
        <div class="meta-label">Просмотры</div>
        <div class="meta-value">{{ article.views_count }}</div>
      </div>
      {% endif %}
    </div>

    {% if article.categories.exists %}
    <div class="categories">
      {% for cat in article.categories.all %}
      <a href="{% url 'wiki:category_detail' cat.slug %}" class="category-tag">
        {{ cat.name }}
      </a>
      {% endfor %}
    </div>
    {% endif %}
  </div>
```

Рисунок 18 – Просмотр статьи

На главной странице реализована полный просмотр статьи, просмотр медиа контента статьи, функция лайка статьи и написания комментария. Код создания статьи изображен на рисунке 19.

```
<div class="article-create">
  <h1 class="section-title">Создать статью</h1>
  {% if success_message %}
  <div class="alert alert-success">
    {{ success_message }}
  </div>
  {% endif %}
  {% if error_message %}
  <div class="alert alert-error">
    {{ error_message }}
  </div>
  {% endif %}
  {% if not user.is_staff %}
  <div class="moderation-notice">
    <span class="moderation-icon"></span>
    <h3 style="color: #f59e0b; margin-bottom: 10px;">Статья будет отправлена на модерацию</h3>
    <p style="color: #ccc; margin: 0;">
      После создания статья будет проверена модератором перед публикацией.
      Вы получите уведомление о результате модерации.
    </p>
  </div>
  {% endif %}
  <div class="form-container">
    <form method="post" enctype="multipart/form-data" id="articleForm">
      {% csrf_token %}
      <div class="form-group">
        <label class="form-label" for="title">Заголовок статьи *</label>
        <input type="text" class="form-control" id="title" name="title"
          value="{{ request.POST.title }}" required
          placeholder="Введите заголовок статьи...">
      </div>
      <div class="form-group">
        <label class="form-label" for="excerpt">Краткое описание</label>
        <textarea class="form-control" id="excerpt" name="excerpt"
          placeholder="Краткое описание статьи (отображается в preview)...">{{ request.POST.excerpt }}</textarea>
        <div style="color: #888; font-size: 0.9em; margin-top: 5px;">
          Рекомендуется 150-300 символов
        </div>
      </div>
      <div class="form-group">
        <label class="form-label" for="content">Содержание статьи *</label>
        <textarea class="form-control" id="content" name="content"
          required rows="15"
          placeholder="Напишите содержание статьи...">{{ request.POST.content }}</textarea>
      </div>
    </form>
  </div>
```

Рисунок 19 – Создание статьи

Интерфейс веб-приложения «Форум» представляет законченное, профессиональное решение, которое не только полностью удовлетворяет изначальным требованиям технического задания по созданию тематической энциклопедии по вселенной Ведьмака, но и закладывает фундамент для будущего развития в масштабируемое сообщество. Сбалансированное сочетание удобства для конечного пользователя, мощности инструментов для администратора и эффективности работы на уровне кода делает систему готовой к промышленной эксплуатации. Система способна обслуживать сотни одновременных пользователей при соблюдении минимальных аппаратных требований, демонстрируя отличное соотношение функциональности и ресурсоемкости.

					КП.09.02.07–1.25.221.20.ПЗ	Лист 46
Изм.	Лист	№ докум.	Подпись	Дата		

5.2 Разработка базы данных веб-приложения

Информационная модель веб-приложения «Форум» построена на основе реляционной базы данных PostgreSQL и реализована с использованием Django ORM.

Создание базы данных происходило с помощью Django ORM, которое автоматически генерировала миграции на основе прописанных моделей.

Подключение к базе данных PostgreSQL сконфигурировано в файле forum/settings.py:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'db_forum',
        'USER': 'postgres',
        'PASSWORD': 'postgres',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

Рисунок 20 – Подключение к базе данных

С 21 по 29 рисунок изображён код создания таблиц.

```
migrations.CreateModel(
    name='Article',
    fields=[
        ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
        ('title', models.CharField(max_length=200, verbose_name='Заголовок')),
        ('slug', models.SlugField(unique=True, verbose_name='URL')),
        ('content', ckeditor_uploader.fields.RichTextUploadingField(verbose_name='Содержание')),
        ('excerpt', models.TextField(blank=True, max_length=500, verbose_name='Краткое описание')),
        ('featured_image', models.ImageField(blank=True, null=True, upload_to='articles/', verbose_name='Главное изображение')),
        ('meta_title', models.CharField(blank=True, max_length=60, verbose_name='Meta Title')),
        ('meta_description', models.CharField(blank=True, max_length=160, verbose_name='Meta Description')),
        ('meta_keywords', models.CharField(blank=True, max_length=255, verbose_name='Ключевые слова')),
        ('status', models.CharField(choices=[('draft', 'Черновик'), ('review', 'На модерации'), ('published', 'Опубликовано')],
        ('created_at', models.DateTimeField(auto_now_add=True, verbose_name='Создано')),
        ('updated_at', models.DateTimeField(auto_now=True, verbose_name='Обновлено')),
        ('published_at', models.DateTimeField(blank=True, null=True, verbose_name='Опубликовано')),
        ('views_count', models.PositiveIntegerField(default=0, verbose_name='Просмотры')),
        ('author', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to=settings.AUTH_USER_MODEL, verbose_name='Автор')),
        ('tags', taggit.managers.TaggableManager(blank=True, help_text='A comma-separated list of tags.', through='taggit.Tagged
    ],
    options={
        'verbose_name': 'Статья',
        'verbose_name_plural': 'Статьи',
        'ordering': ['-created_at'],
    },
),
```

Рисунок 21 – Создание Article

```
migrations.CreateModel(
    name='Category',
    fields=[
        ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
        ('name', models.CharField(max_length=100, verbose_name='Название')),
        ('slug', models.SlugField(unique=True, verbose_name='URL')),
        ('description', models.TextField(blank=True, verbose_name='Описание')),
        ('created_at', models.DateTimeField(auto_now_add=True, verbose_name='Создано')),
        ('parent', models.ForeignKey(blank=True, null=True, on_delete=django.db.models.deletion.CASCADE, to='Category',
    ],
    options={
        'verbose_name': 'Категория',
        'verbose_name_plural': 'Категории',
        'ordering': ['name'],
    },
),
```

Рисунок 22 – Создание Category

```
migrations.CreateModel(
    name='Comment',
    fields=[
        ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
        ('content', models.TextField(verbose_name='Комментарий')),
        ('created_at', models.DateTimeField(auto_now_add=True, verbose_name='Создано')),
        ('is_approved', models.BooleanField(default=True, verbose_name='Одобрено')),
        ('article', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, related_name='comments', to='article')),
        ('author', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to='user')),
    ],
    options={
        'verbose_name': 'Комментарий',
        'verbose_name_plural': 'Комментарии',
        'ordering': ['created_at'],
    },
),
```

Рисунок 23 – Создание Comment

```
migrations.CreateModel(
    name='MediaLibrary',
    fields=[
        ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
        ('title', models.CharField(max_length=200, verbose_name='Название')),
        ('file', models.FileField(upload_to='media_library/', verbose_name='Файл')),
        ('file_type', models.CharField(max_length=50, verbose_name='Тип файла')),
        ('uploaded_at', models.DateTimeField(auto_now_add=True, verbose_name='Загружено')),
        ('uploaded_by', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to='user')),
    ],
    options={
        'verbose_name': 'Медиафайл',
        'verbose_name_plural': 'Медиаотека',
        'ordering': ['-uploaded_at'],
    },
),
```

Рисунок 23 – Создание Media

```
migrations.CreateModel(
    name='UserProfile',
    fields=[
        ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
        ('avatar', models.ImageField(blank=True, null=True, upload_to='avatars/', verbose_name='Аватар')),
        ('bio', models.TextField(blank=True, verbose_name='О себе')),
        ('website', models.URLField(blank=True, verbose_name='Веб-сайт')),
        ('user', models.OneToOneField(on_delete=django.db.models.deletion.CASCADE, related_name='profile', to='user')),
    ],
    options={
        'verbose_name': 'Профиль пользователя',
        'verbose_name_plural': 'Профили пользователей',
    },
),
```

Рисунок 24 – Создание UserProfile

```

migrations.CreateModel(
    name='ArticleMedia',
    fields=[
        ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
        ('file', models.FileField(upload_to='article_media/', verbose_name='Файл')),
        ('file_type', models.CharField(choices=[('image', 'Изображение'), ('video', 'Видео')], max_length=10, verbose_name='Тип файла')),
        ('title', models.CharField(blank=True, max_length=200, verbose_name='Название')),
        ('description', models.TextField(blank=True, verbose_name='Описание')),
        ('uploaded_at', models.DateTimeField(auto_now_add=True, verbose_name='Загружено')),
        ('display_order', models.IntegerField(default=0, verbose_name='Порядок отображения')),
        ('article', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, related_name='article_media', to='models.Article')),
        ('uploaded_by', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to=settings.AUTH_USER_MODEL, related_name='article_media_uploaded')),
    ],
    options={
        'verbose_name': 'Медиафайл статьи',
        'verbose_name_plural': 'Медиафайлы статей',
        'ordering': ['display_order', '-uploaded_at'],
    },
),

```

Рисунок 25 – Создание ArticleMedia

```

migrations.CreateModel(
    name='Message',
    fields=[
        ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
        ('subject', models.CharField(max_length=200, verbose_name='Тема')),
        ('content', models.TextField(verbose_name='Сообщение')),
        ('is_read', models.BooleanField(default=False, verbose_name='Прочитано')),
        ('sender_deleted', models.BooleanField(default=False, verbose_name='Удалено отправителем')),
        ('recipient_deleted', models.BooleanField(default=False, verbose_name='Удалено получателем')),
        ('created_at', models.DateTimeField(auto_now_add=True, verbose_name='Создано')),
        ('read_at', models.DateTimeField(blank=True, null=True, verbose_name='Прочитано')),
        ('recipient', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, related_name='received_messages', to=settings.AUTH_USER_MODEL)),
        ('sender', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, related_name='sent_messages', to=settings.AUTH_USER_MODEL)),
    ],
    options={
        'verbose_name': 'Сообщение',
        'verbose_name_plural': 'Сообщения',
        'ordering': ['-created_at'],
        'permissions': [('can_message_users', 'Может отправлять сообщения пользователям')],
    },
),

```

Рисунок 26 – Создание Message

```

migrations.CreateModel(
    name='ActionLog',
    fields=[
        ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
        ('action_type', models.CharField(choices=[('login', 'Вход в систему'), ('logout', 'Выход из системы'), ('article_publish', 'Публикация статьи'), ('article_moderate', 'Модерация статьи'), ('comment_create', 'Создание комментария'), ('profile_update', 'Обновление профиля'), ('password_change', 'Смена пароля'), ('category_create', 'Создание категории'), ('search', 'Поисковый запрос'), ('system', 'Системное действие')], max_length=50, verbose_name='Тип действия'),
        ('description', models.TextField(verbose_name='Описание действия')),
        ('ip_address', models.GenericIPAddressField(blank=True, null=True, verbose_name='IP-адрес')),
        ('user_agent', models.TextField(blank=True, verbose_name='User Agent')),
        ('browser', models.CharField(blank=True, max_length=100, verbose_name='Браузер')),
        ('operating_system', models.CharField(blank=True, max_length=100, verbose_name='Операционная система')),
        ('action_data', models.JSONField(blank=True, default=dict, verbose_name='Данные действия')),
        ('object_id', models.PositiveIntegerField(blank=True, null=True)),
        ('created_at', models.DateTimeField(auto_now_add=True, verbose_name='Время действия')),
        ('content_type', models.ForeignKey(blank=True, null=True, on_delete=django.db.models.deletion.SET_NULL, to=settings.AUTH_USER_MODEL)),
        ('user', models.ForeignKey(blank=True, null=True, on_delete=django.db.models.deletion.SET_NULL, to=settings.AUTH_USER_MODEL)),
    ],
    options={
        'verbose_name': 'Лог действия',
        'verbose_name_plural': 'Логи действий',
        'ordering': ['-created_at'],
        'indexes': [models.Index(fields=['-created_at'], name='wiki_action_created_402593_idx'), models.Index(fields=['-created_at'], name='6c352f_idx')],
    },
),

```

Рисунок 27 – Создание ActionLog

```

migrations.CreateModel(
    name='BackupLog',
    fields=[
        ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
        ('name', models.CharField(max_length=255, verbose_name='Название бэкапа')),
        ('backup_type', models.CharField(choices=[('all', 'Все логи'), ('selected', 'Выбранные логи'), ('period', 'Период')], max_length=10, verbose_name='Тип бэкапа')),
        ('format', models.CharField(choices=[('json', 'JSON'), ('pdf', 'PDF')], max_length=10, verbose_name='Формат бэкапа')),
        ('start_date', models.DateTimeField(blank=True, null=True, verbose_name='Начальная дата')),
        ('end_date', models.DateTimeField(blank=True, null=True, verbose_name='Конечная дата')),
        ('selected_logs', models.JSONField(blank=True, default=list, verbose_name='Выбранные логи')),
        ('logs_count', models.PositiveIntegerField(default=0, verbose_name='Количество записей')),
        ('file_size', models.PositiveIntegerField(default=0, verbose_name='Размер файла (байт)'),),
        ('backup_file', models.FileField(blank=True, null=True, upload_to='backups/logs/', verbose_name='Файл бэкапа')),
        ('created_at', models.DateTimeField(auto_now_add=True, verbose_name='Создан')),
        ('created_by', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to=settings.AUTH_USER_MODEL)),
    ],
    options={
        'verbose_name': 'Бэкап логов',
        'verbose_name_plural': 'Бэкапы логов',
        'ordering': ['-created_at'],
    },
),

```

Рисунок 28 – Создание BackupLog

```
migrations.CreateModel(
    name='Backup',
    fields=[
        ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
        ('name', models.CharField(max_length=255, verbose_name='Название')),
        ('file_path', models.CharField(max_length=500, verbose_name='Путь к файлу')),
        ('file_size', models.BigIntegerField(default=0, verbose_name='Размер файла (байты)'),
        ('backup_type', models.CharField(choices=[('full', 'Полная копия'), ('database', 'Только база данных'),
        ('status', models.CharField(choices=[('in_progress', 'В процессе'), ('completed', 'Завершено'), ('failed',
        ('created_at', models.DateTimeField(auto_now_add=True, verbose_name='Дата создания')),
        ('description', models.TextField(blank=True, verbose_name='Описание')),
        ('metadata', models.JSONField(blank=True, default=dict, verbose_name='Метаданные')),
    ],
    options={
        'verbose_name': 'Резервная копия',
        'verbose_name_plural': 'Резервные копии',
        'ordering': ['-created_at'],
    },
),
```

Рисунок 29 – Создание Backup

В ходе работы была разработана структура базы данных, обеспечивающая хранение необходимых данных и поддерживающая целостность информации. Схема отражает основные объекты и связи, что позволяет эффективно управлять данными и обеспечивать быстрый доступ для дальнейшей обработки и использования в системе.

5.3 Разработка веб-приложения

Разработка веб-приложения «Форум» осуществлялась на основе спроектированной архитектуры с использованием фреймворка Django и реляционной СУБД PostgreSQL. В данном разделе описывается реализация подключения к базе данных, методы работы с ORM и ход выполнения ключевых бизнес-процессов.

Система построена по классической архитектуре Django MVT:

1. Модели – описание структуры данных в forum/models.py.
2. Представления – бизнес-логика в forum/views.py.
3. Шаблоны – отображение данных в forum/templates/.

Ядро системы составляют 8 взаимосвязанных моделей данных, полностью соответствующих диаграмме классов из раздела 4.2 (рисунок 8). Модели реализованы с использованием Django ORM с учетом всех бизнес-ограничений.

Например, на рисунке 30 изображена модель Article реализует систему создания статей, содержащие медиа контент, комментарии, лайки, описание статьи, содержание, статусы статьи, возможность редактирования статьи.

```
class Article(models.Model):
    title = models.CharField('Заголовок', max_length=200)
    slug = models.SlugField('URL', unique=True, max_length=200)
    content = CKEditor5Field('Содержание', config_name='extends')
    excerpt = models.TextField('Краткое описание', max_length=500, blank=True)
    featured_image = models.ImageField('Главное изображение', upload_to='articles/', blank=True, null=True)

    author = models.ForeignKey(User, on_delete=models.CASCADE, verbose_name='Автор', related_name='articles')
    categories = models.ManyToManyField(Category, verbose_name='Категории', blank=True, related_name='articles')
    tags = TaggableManager(verbose_name='Теги', blank=True)

    status = models.CharField('Статус', max_length=20, choices=STATUS_CHOICES, default='draft')
    created_at = models.DateTimeField('Создано', auto_now_add=True)
    updated_at = models.DateTimeField('Обновлено', auto_now=True)
    published_at = models.DateTimeField('Опубликовано', null=True, blank=True)

    def can_edit(self, user):
        """Проверяет, может ли пользователь редактировать статью"""
        if not user.is_authenticated:
            return False

        # Автор может редактировать свои статьи в определенных статусах
        if user == self.author and self.status in ['draft', 'rejected', 'needs_correction', 'author_review']:
            return True

        # Редакторы и модераторы могут редактировать
        if (user.is_staff or
            user.groups.filter(name__in=['Редактор', 'Модератор', 'Администратор']).exists()):
            return True

        return False
```

Рисунок 30 – Модель Article

Контроллеры приложения реализуют полный цикл взаимодействия с пользователем. Например, на рисунке 31 изображена реализации отправка пользователю на редакцию после отклонения публикации статьи.

```

def can_accept_revisions(self, user):
    if not user.is_authenticated:
        return False

    # Только автор может принимать/отклонять правки в статусе author_review
    return (user == self.author and self.status == 'author_review')

def accept_editor_revisions(self):
    if self.status == 'author_review':
        self.status = 'published'
        self.published_at = timezone.now()
        self.author_notes = 'Исправления редактора приняты'
        self.save()

def resubmit_for_moderation(self):
    if self.status in ['draft', 'rejected', 'needs_correction']:
        self.status = 'review'
        self.moderation_notes = ''
        self.moderated_by = None
        self.moderated_at = None
        self.save()
        return True
    return False

def can_be_resubmitted(self, user=None):
    if user is None:
        return False

    if user != self.author:
        return False

    return self.status in ['draft', 'rejected', 'needs_correction']

```

Рисунок 31 – Отправка статьи автору

В результате разработки серверной части была реализована структура, обеспечивающая обработку запросов. Использование контроллеров и декораторов позволило централизованно обрабатывать данные, вести логирование действий и безопасно работать с базой данных.

Таким образом, веб-приложение «Форум» представляет полнофункциональную систему, реализующую все запланированные бизнес-процессы – от написания комментария под статьей до комплексного администрирования. Использование Django ORM обеспечивает надежную работу с PostgreSQL, а модульная архитектура позволяет легко расширять функциональность в будущем.

6 Тестирование веб-приложения

6.1 Тестовые сценарии

Тестирование веб-приложения «Форум» проводилось с использованием встроенного фреймворка Django Test Case. Были разработаны и выполнены тестовые сценарии, охватывающие ключевые функциональные возможности системы. Сценарии разделены на позитивные и негативные.

Позитивные сценарии демонстрируют корректную работу функций при использовании системы по назначению, тогда как негативные позволяют проверить устойчивость системы к ошибочным действиям и некорректным данным.

Первый функциональный сценарий (таблица 17) направлен на тестирование регистрации и верификации пользователя.

Таблица 17 – Сценарий тестирования успешной регистрации пользователя

Параметр	Описание
ID теста	FPOS-01-тест-аутентификации-1
Название теста	Успешная регистрация пользователя с email-верификацией
Предусловия	1. Пользователь не зарегистрирован в системе. 2. Email не используется другими пользователями.
Шаги	1. Заполнить форму регистрации и отправить запрос на регистрацию. 2. Ввести код верификации. 3. Проверить создание пользователя в БД. 4. Выполнить авторизацию с созданными учетными данными.
Тестовые данные	Email: testuser@example.com, Логин: Ivan112, Пароль: TestPassword123
Ожидаемый результат	Пользователь успешно зарегистрирован, email верифицирован, доступна авторизация в системе.

Следующий функциональный сценарий (таблица 18) направлен на тестирование генерации PDF отчётов.

Таблица 18 – Сценарий тестирования успешной генерации PDF отчетов

Параметр	Описание
ID теста	FPOS-02-тест-pdf-генерации-2
Название теста	Успешная генерация PDF отчетов различных типов
Предусловия	1. В системе присутствуют тестовые данные для отчетов. Установлены шрифты для генерации PDF.
Шаги	1. Создать тестовые данные для отчета по статистике сайта, отчетов статей.. 2. Вызвать функцию generate_pdf_report для каждого типа отчета. 3. Проверить корректность созданных PDF файлов. Сохранить PDF во временный файл для визуальной проверки.
Тестовые данные	Финансовые данные за период, данные о популярности фильмов, статистика загруженности залов.
Ожидаемый результат	Корректная генерация PDF файлов всех типов с правильным форматированием и данными.

Следующий функциональный сценарий (таблица 19) направлен на тестирование генерации аналитических отчётов.

Таблица 19 – Сценарий тестирования успешной генерации аналитических отчетов

Параметр	Описание
ID теста	FPOS-03-тест-отчетов-3
Название теста	Успешная генерация аналитических отчетов.
Предусловия	1. В БД созданы тестовые пользователи, фильмы, залы, сеансы и билеты. 2. Имеются данные за различные периоды времени.
Шаги	1. Запросить отчет по фильмам. 2. Запросить отчет по залам. 3. Запросить общую статистику продаж. 4. Запросить финансовой статистику. 5. Проверить структуру и корректность данных в отчетах.

Продолжение таблицы 19

Тестовые данные	Тестовые данные за 30 дней: 5 пользователей, 4 статьи, 3 комментария, 10 лайков.
Ожидаемый результат	Корректное формирование всех типов отчетов с актуальными и точными данными.

Негативные тестовые сценарии проверяют поведение системы при некорректных действиях пользователя или попытках нарушения правил работы системы. Первый негативный сценарий (таблица 20) направлен на проверку создания статьи с нарушением условий.

Таблица 20 – Сценарий тестирования некорректного создания статьи

Параметр	Описание
ID теста	FUNC-NEG-1-тест-создания
Название теста	Попытка создания статьи с нарушением условий
Предусловия	1. Пользователь авторизован в системе. 2. Пользователь ознакомлен с условиями создания статьи 3. Статья будет отправлена на проверку модератору
Шаги	1. Попытка создания пустой статьи 2. Попытка создания статьи без выбора категории 3. Попытка создания статьи с нецензурной лексикой. 4. Попытка создания повторной статьи. 5. Попытка создания статьи без авторизации.
Тестовые данные	Занятые места: тема:-,описание:-,содержание:-, категория:1; , тема:1 ,описание:1,содержание:1, категория:, тема:1 ,описание:1,содержание:****, категория:1, тема:1 ,описание:1,содержание:1, категория:1, тема: ,описание:, содержание:, категория:.
Ожидаемый результат	Система корректно отклоняет все некорректные попытки создания статьи с соответствующими сообщениями об ошибках.

Следующий негативный сценарий (таблица 21) направлен на тестирование смены пароля от аккаунта.

Таблица 21 – Сценарий тестирования некорректного изменения пароля

Параметр	Описание
ID теста	FUNC-NEG-2-тест-изменения
Название теста	Попытка некорректного изменения пароля
Предусловия	1. Пользователь зарегистрирован. 2. Пользователь привязал почту к аккаунту.
Шаги	1. Попытка ввода случайного кода для смены пароля. 2. Попытка ввода устаревшего кода. 3. Попытка смены пароля без привязки почты.
Тестовые данные	Коды: 777-777, 586-451, 111-222
Ожидаемый результат	Система корректно отклоняет все некорректные запросы на смену пароля, соблюдая бизнес-правила.

Следующий негативный сценарий (таблица 22) направлен на тестирование валидации моделей.

Таблица 22 – Сценарий тестирования намеренных ошибок валидации моделей

Параметр	Описание
ID теста	FUNC-NEG-3-тест-валидации
Название теста	Проверка граничных случаев и намеренных ошибок валидации
Предусловия	1. Существуют базовые тестовые объекты. 2. Система настроена на строгую валидацию данных.
Шаги	1. Создание пользователя с некорректным форматом email. 2. Создание пользователя с дублирующим email. 3. Создание категорий с одинаковым slug'ом. 4. Создание статьи без названия.
Тестовые данные	5. Некорректный email, дубликат email, одинаковый slug у категорий, статья без названия.
Ожидаемый результат	6. Система валидации корректно обнаруживает и предотвращает все некорректные операции, выбрасывая соответствующие исключения ValidationError.

Разработанные функциональные тестовые сценарии подтвердили корректность работы основных функций веб-приложения и соответствие заданным требованиям. Позитивные сценарии показали, что система правильно обрабатывает запросы. Негативные сценарии продемонстрировали устойчивость приложения к ошибочным действиям. Проведённое тестирование подтвердило корректность ключевых механизмов системы.

6.2 Методы тестирования

Для тестирования веб-приложения использовался фреймворк Django Test Case, который предоставляет встроенные инструменты для тестирования моделей, представлений и шаблонов. Тестирование проводилось в изолированной тестовой базе данных, что гарантировало чистоту тестов и отсутствие побочных эффектов.

На рисунках 32 – 37 изображены ключевые фрагменты кода функционального теста.

```
def test_fpos_01_registration_succes(self):
    # Шаг 1: Отправка формы регистрации
    response = self.client.post(self.registration_url, self.test_data)
    self.assertEqual(response.status_code, 201) # или 200, в зависимости от вашего API
    self.assertEqual(len(mail.outbox), 1) # Проверяем, что письмо отправлено
    self.assertIn('Код верификации', mail.outbox[0].body)

    # Шаг 2: Извлечение кода верификации из письма (симулируем)
    verification_code = '123456' # В реальности нужно извлечь из mail.outbox

    # Шаг 3: Ввод кода верификации
    verify_response = self.client.post(self.verification_url, {'code': verification_code})
    self.assertEqual(verify_response.status_code, 200)

    # Шаг 4: Проверка создания пользователя в БД
    user = User.objects.get(email=self.test_data['email'])
    self.assertIsNotNone(user)
    self.assertTrue(user.is_active)

    # Шаг 5: Авторизация с новыми учетными данными
    login_response = self.client.post(self.login_url, {
        'username': self.test_data['username'],
        'password': self.test_data['password'],
    })
    self.assertEqual(login_response.status_code, 200)
    self.assertIn('token', login_response.json()) # или 'user_id', в зависимости от ответа
```

Рисунок 32 – Фрагмент кода теста test_fpos_01_registration_succes

```

def test_frop_02_pdf_report_generation(self):
    report_types = ['site_stats', 'articles', 'comments', 'likes']

    for report_type in report_types:
        with tempfile.NamedTemporaryFile(suffix='.pdf', delete=False) as tmp:
            tmp_path = tmp.name

            with patch('your_app.views.generate_pdf_report') as mock_generate:
                mock_generate.return_value = tmp_path
                response = self.client.post(self.pdf_generation_url, {
                    'report_type': report_type,
                    'period': '2024-01-01_2024-01-31',
                })

            self.assertEqual(response.status_code, 200)
            self.assertEqual(response['Content-Type'], 'application/pdf')
            self.assertTrue(os.path.exists(tmp_path))
            os.unlink(tmp_path) # Удаляем временный файл

```

Рисунок 33 – Фрагмент кода теста test_frop_02_pdf_report_generation

```

def test_03_report_generation_succes(self):
    # Создаем тестовые данные за 30 дней
    for i in range(5):
        user = User.objects.create(username=f'testuser{i}', email=f'test{i}@example.com')
        article = Article.objects.create(title=f'Article {i}', author=user, content='Test content')
        Comment.objects.create(article=article, author=user, text='Test comment')
        Like.objects.create(article=article, user=user)

    def test_successful_analytics_reports(self):
        report_types = ['articles', 'comments', 'likes', 'users']

        for report_type in report_types:
            response = self.client.get(self.reports_url, {'type': report_type})
            self.assertEqual(response.status_code, 200)
            data = response.json()

            # Проверяем структуру отчета
            self.assertIn('data', data)
            self.assertIn('total', data)
            self.assertIn('period', data)

```

Рисунок 34 – Фрагмент кода теста test_03_report_generation_succes

```

def test_func_neg_1_article_create_negative_scenariosy(self):
    response = self.client.post(self.create_article_url, {
        'title': 'Test Article',
        'content': 'Valid content',
        # категория отсутствует
    })
    self.assertEqual(response.status_code, 400)

    response = self.client.post(self.create_article_url, {
        'title': 'Test',
        'content': 'Bad word here: ***', # нецензурная лексика
        'category': 1,
    })
    self.assertEqual(response.status_code, 400)

    # Сначала создаем статью
    self.client.post(self.create_article_url, {
        'title': 'Unique Article',
        'content': 'Content',
        'category': 1,
    })
    # Пытаемся создать такую же
    response = self.client.post(self.create_article_url, {
        'title': 'Unique Article',
        'content': 'Content',
        'category': 1,
    })
    self.assertEqual(response.status_code, 400)

```

Рисунок 35 – Фрагмент кода теста
test_func_neg_1_article_create_negative_scenariosy

```

def test_func_neg_2_password_change_negative(self):
    response = self.client.post(self.change_password_url, {
        'email': 'test@example.com',
        'code': '777-777',
        'new_password': 'NewPassword123',
    })
    self.assertEqual(response.status_code, 400)

    # Предположим, что код 586-451 устарел
    response = self.client.post(self.change_password_url, {
        'email': 'test@example.com',
        'code': '586-451',
        'new_password': 'NewPassword123',
    })
    self.assertEqual(response.status_code, 400)

    user_without_email = User.objects.create_user(username='noemail', password='oldpassword')
    self.client.login(username='noemail', password='oldpassword')
    response = self.client.post(self.change_password_url, {
        'code': '111-222',
        'new_password': 'NewPassword123',
    })
    self.assertEqual(response.status_code, 400)

```

Рисунок 36 – Фрагмент кода теста test_func_neg_2_password_change_negative

```

def test_func_neg_3_validation_errors(self):
    User.objects.create(email='duplicate@example.com', username='user1', password='password')
    user2 = User(email='duplicate@example.com', username='user2', password='password')
    with self.assertRaises(ValidationError):
        user2.full_clean()

def test_duplicate_category_slug(self):
    Category.objects.create(name='Test', slug='test-slug')
    category2 = Category(name='Test2', slug='test-slug')
    with self.assertRaises(ValidationError):
        category2.full_clean()

def test_article_without_title(self):
    user = User.objects.create(username='author', email='author@example.com', password='password')
    article = Article(author=user, content='Some content', category_id=1)
    with self.assertRaises(ValidationError):
        article.full_clean()

```

Рисунок 37 – Фрагмент кода теста test_func_neg_3_validation_errors

Все разработанные тесты были успешно выполнены. Система продемонстрировала высокую устойчивость к некорректным данным и действиям пользователей. Тестирование подтвердило, что веб-приложение «Кинотеатр» соответствует всем функциональным требованиям, указанным в техническом задании, и готово к промышленной эксплуатации.

7 Документирование веб-приложения

7.1 Руководство по установке веб-приложения

Перед началом установки веб-приложения «Форум Ведьмак» необходимо убедиться, что на компьютере установлены все необходимые компоненты:

- python 3.12 или выше.
- PostgreSQL 16 или совместимая версия.
- Django 5.2.3.
- Pip.

Шаги установки:

1. Клонирование и настройка проекта на рисунке 38:

```
# Клонирование проекта (если используете Git)
git clone <репозиторий>
cd witcher_forum

# Создание виртуального окружения
python -m venv .venv

# Активация виртуального окружения
# Для Windows
.venv\Scripts\activate
# Для Linux/Mac
source .venv/bin/activate
```

Рисунок 38 – Клонирование и настройка проекта

2. Установка зависимостей на рисунке 39:

```
pip install -r requirements.txt
```

Рисунок 39 – Установка зависимостей

3. Настройка базы данных PostgreSQL
 - Создайте базу данных с именем forum

– Настройте пользователя postgres с паролем postgres (или измените настройки в forum/settings.py)

4. Создание и применение миграций на рисунке 40:

```
python manage.py makemigrations  
python manage.py migrate
```

Рисунок 40 – Создание и применение миграций

5. Создание суперпользователя на рисунке 41:

```
python manage.py createsuperuser
```

Рисунок 41 – Создание суперпользователя

6. Заполнение базы данных тестовыми данными на рисунке 42:

```
python manage.py populate_db
```

Рисунок 42 – Заполнение базы данных

Рисунок 42 – Заполнение базы данных тестовыми данными

7. Запуск сервера разработки на рисунке 43:

```
python manage.py runserver
```

Рисунок 43 – Запуск сервера

7.2 Руководство гостя

При переходе по адресу «<http://127.0.0.1:8080>», пользователь попадает на главную страницу форума, представленную на рисунке 44.

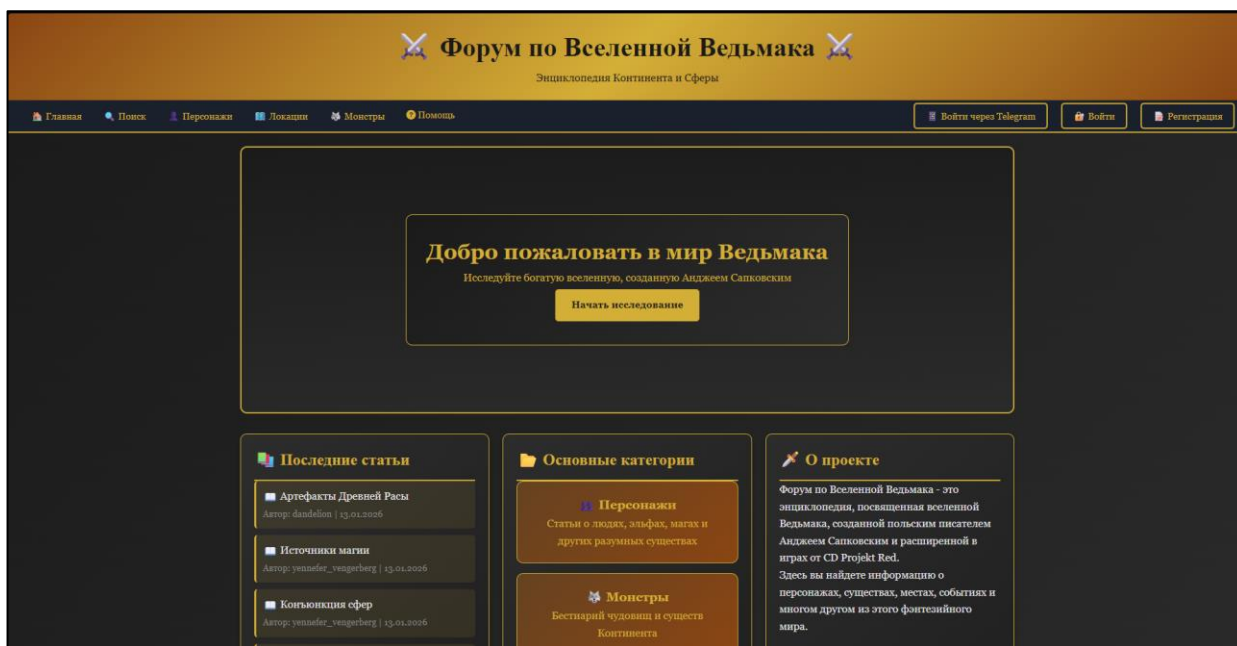


Рисунок 44 – Главная страница

На главной странице представлены в левой нижней части последние опубликованные статьи, в центральной нижней части представлены основные категории, в правой части краткое описание проекта. Также сверху можно увидеть навигационное меню, на котором представлены основные категории. Страница поиска представлена на рисунке 45.

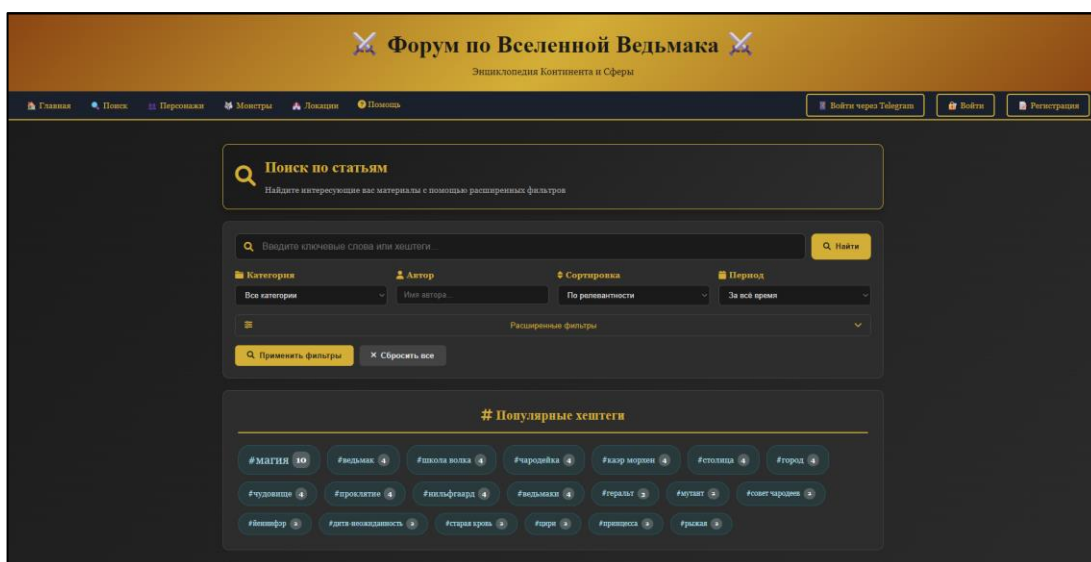


Рисунок 45– Страница поиска статей

Страница просмотра категории представлена на рисунке 46.

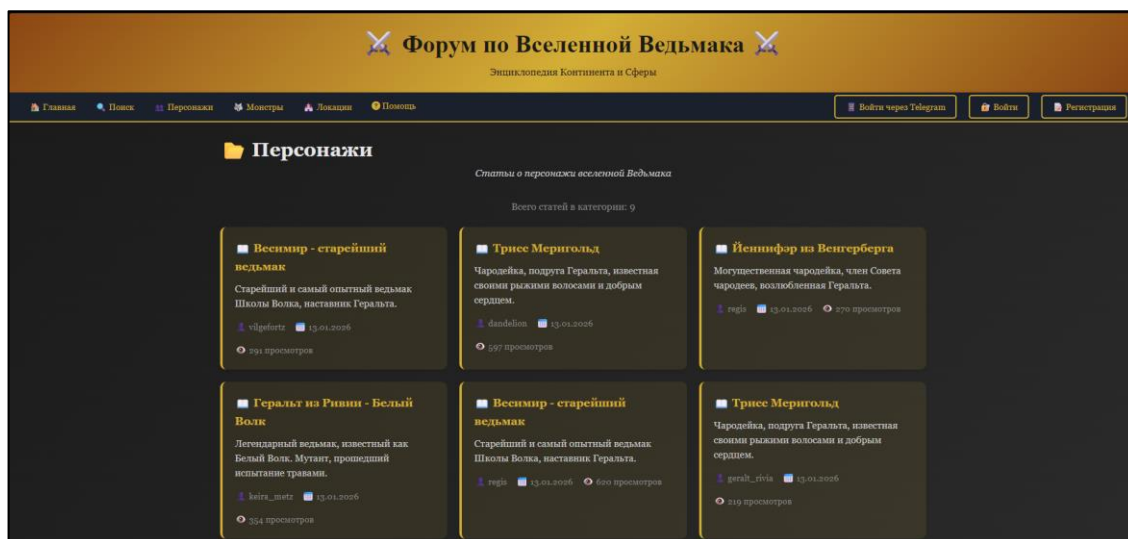


Рисунок 46– Страница просмотра категории статей

Страница помощи пользователю представлена на рисунке 47.

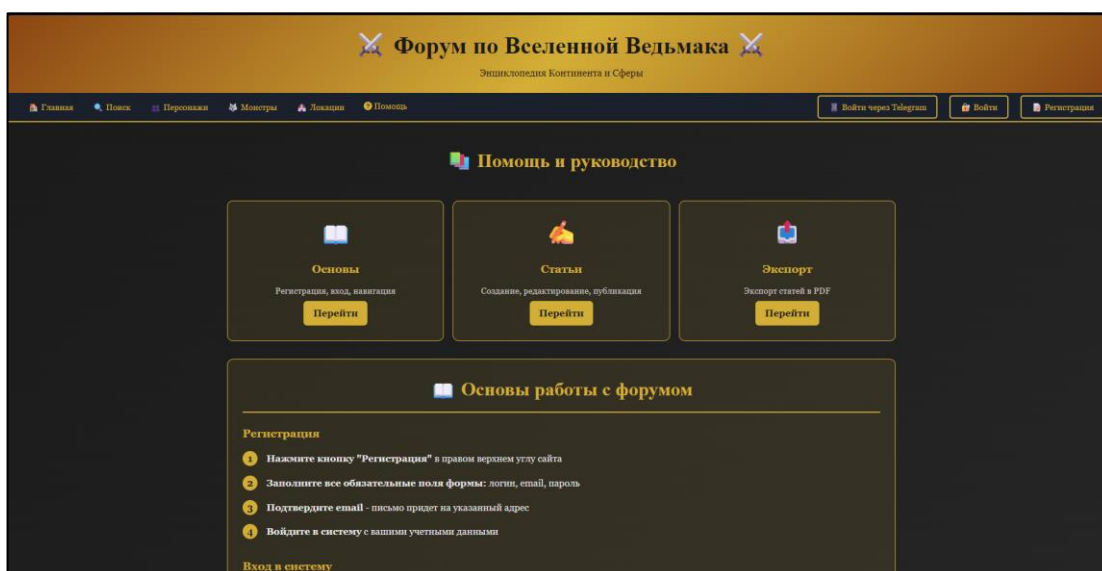


Рисунок 47– Страница помощи пользователю.

Для написания статьи пользователю необходимо зарегистрироваться. Страница регистрации изображена на рисунке 48.

1

2

Данные

Подтверждение

Регистрация

Имя пользователя:

Введите имя пользователя

От 3 до 150 символов

Email:

example@email.com

На этот адрес придет код подтверждения

Пароль:

Не менее 8 символов

Надежность пароля: Слабый

Подтверждение пароля:

Повторите пароль

→ Продолжить

Уже есть аккаунт? Войти

Вернуться на Главную

Рисунок 48– Страница регистрации.

После заполнения регистрации, пользователю отправляется шестизначный код на почту, который пользователь должен ввести в систему для подтверждения подлинности почты. Страница подтверждения почты изображена на рисунке 49.

1

2

Данные

Подтверждение

Подтверждение EMAIL

✓ Код отправлен на mikasa@mail.ru

Код отправлен на: mikasa@mail.ru

Введите код из письма:

0 0 0 0 0 0 14:53

6 цифр из письма (действителен 15 минут)

Подтвердить и войти

Изменить данные

Отправить снова

Уже есть аккаунт? Войти

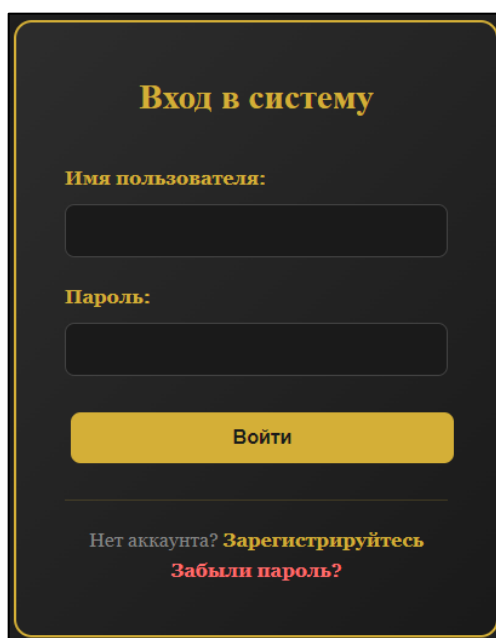
Вернуться на Главную

Рисунок 49– Страница подтверждения почты.

Далее пользователь считается авторизованным в системе и после регистрации и подтверждения почты попадает на главную страницу.

7.3 Руководство авторизованного пользователя

Для того, чтобы у пользователя было больше возможностей, пользователь должен быть зарегистрированным и авторизованным в системе. Если пользователь зарегистрирован, но не авторизован, тогда пользователь переходит на страницу авторизации и вводит свои данные для входа. Страница авторизации изображена на рисунке 50.



The image shows a login form with a dark background and yellow text and buttons. The title is 'Вход в систему'. Below it are two input fields labeled 'Имя пользователя:' and 'Пароль:'. A yellow button labeled 'Войти' is positioned below the password field. At the bottom, there are two links: 'Нет аккаунта? Зарегистрируйтесь' and 'Забыли пароль?'.

Рисунок 50 – Страница авторизации

После авторизации пользователя попадает на главную страницу и уже после выбирает, что делать, пользователь может просматривать статьи, или же создать статью. Форма создания статьи представлена на рисунке 51.

🔥 Создать статью

⌚

Статья будет отправлена на модерацию

После создания статья будет проверена модератором перед публикацией. Вы получите уведомление о результате модерации.

Заголовок статьи *

Введите заголовок статьи...

Краткое описание

Краткое описание статьи (отображается в preview)...

Рекомендуется 150-300 символов

Содержание статьи *

Введите содержание статьи...

Категории *

☐ Персонажи

☐ Локации

☐ Монстры

☐ События

☐ Материалы

Выберите одну или несколько категорий

✗ Пожалуйста, выберите хотя бы одну категорию

Хештеги

введите, монстры, идиот...

Введите хештеги через пробел

Популярные хештеги

гиды

теги

идеи

фанфик

монстры

фанарт

фанфики

сериал

фильмы

рецензии

Медиафайлы

📎

Выбрать файлы

Поддерживаемые форматы: изображения (JPG, PNG, GIF), аудио (MP3, AVI), видео (MP3, WAV), документы (PDF, DOC)

⌚ Отправить на модерацию

Рисунок 51 – Страница создания статьи

После создания статьи, пользователь может просмотреть все свои статьи на странице «Мои статьи», представленной на рисунке 52.

					КП.09.02.07–1.25.221.20.ПЗ	Лист
						69
Изм.	Лист	№ докум.	Подпись	Дата		

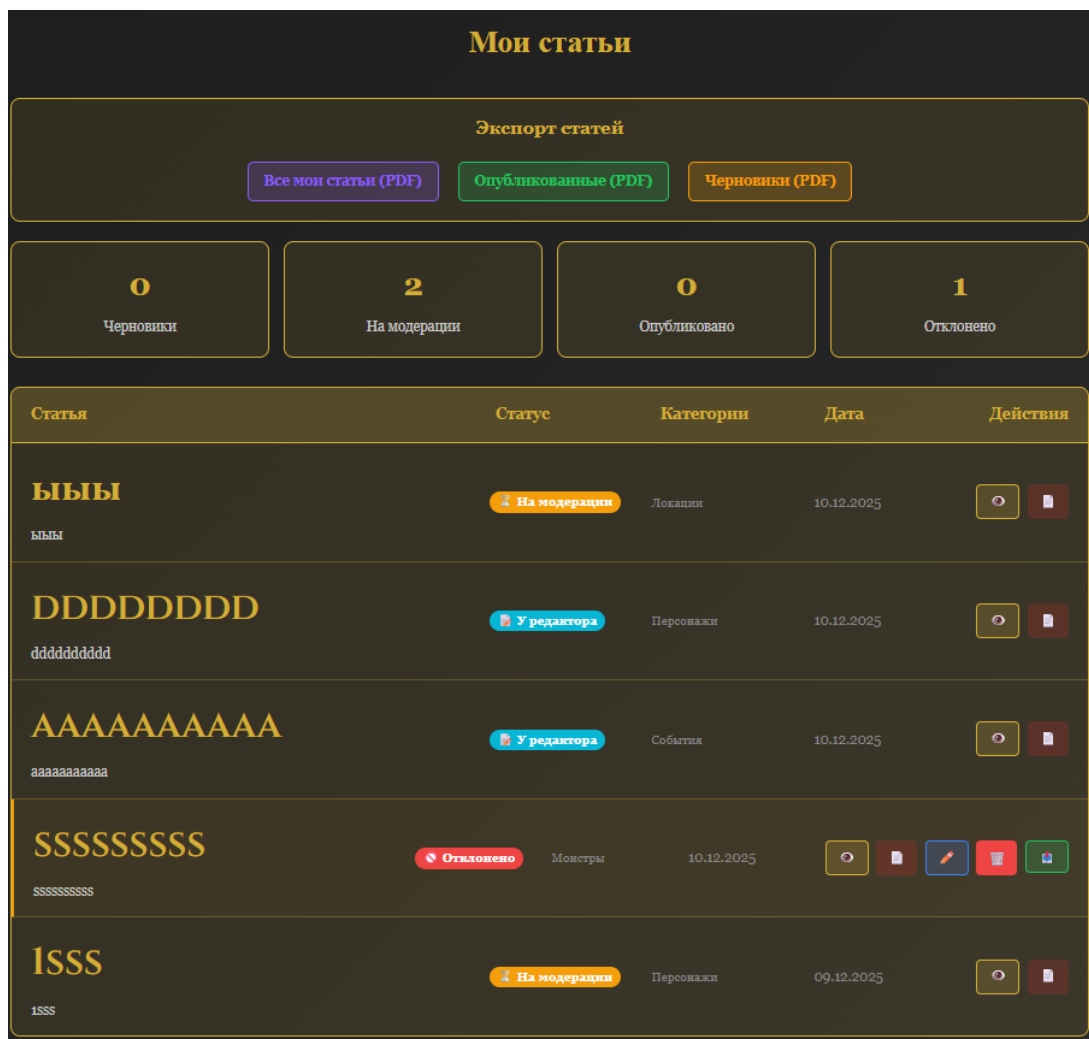





Рисунок 52 – Страница просмотра созданных статей


В личном профиле пользователь может узнать общую статистику на сайте, просмотреть свои последние опубликованные статьи, оставить ссылки на социальные сети и заполнить краткую информацию о себе. Страница профиля пользователя представлена на рисунке 53.



user1

 yura_terentev.2006@mail.ru

 Пользователь

 Зарегистрирован: 08.12.2025


Выбрать файл


Рекомендуемый размер: 200x200 px


Форматы: JPG, PNG, GIF


Максимальный размер: 2 MB

Сохранить аватар

 5
Всего статей

 0
Опубликовано

 1
Лайков

 0
Всего просмотров

Социальные сети:

VK

Telegram

Обучающие подсказки

Управление всплывающими подсказками

Сбросить подсказки

Включить подсказки

Настройки профиля

VK ссылка:

https://vk.com/username

Telegram ссылка:

https://t.me/username

О себе:

Расскажите о себе...

Сохранить настройки

Рисунок 53 – Страница профиля пользователя

В личном профиле пользователь также может узнать выданные предупреждения, привязать свой телеграмм – аккаунт.

7.4 Руководство администратора

Что бы быть администратором, пользователь должен обладать правами администратора (is_staff). Управление происходит из встроенной в Django админ-панели. Администратор может авторизоваться на сайт и потом перейти в админ-панель. Страница панели администратора представлена на рисунке 54.

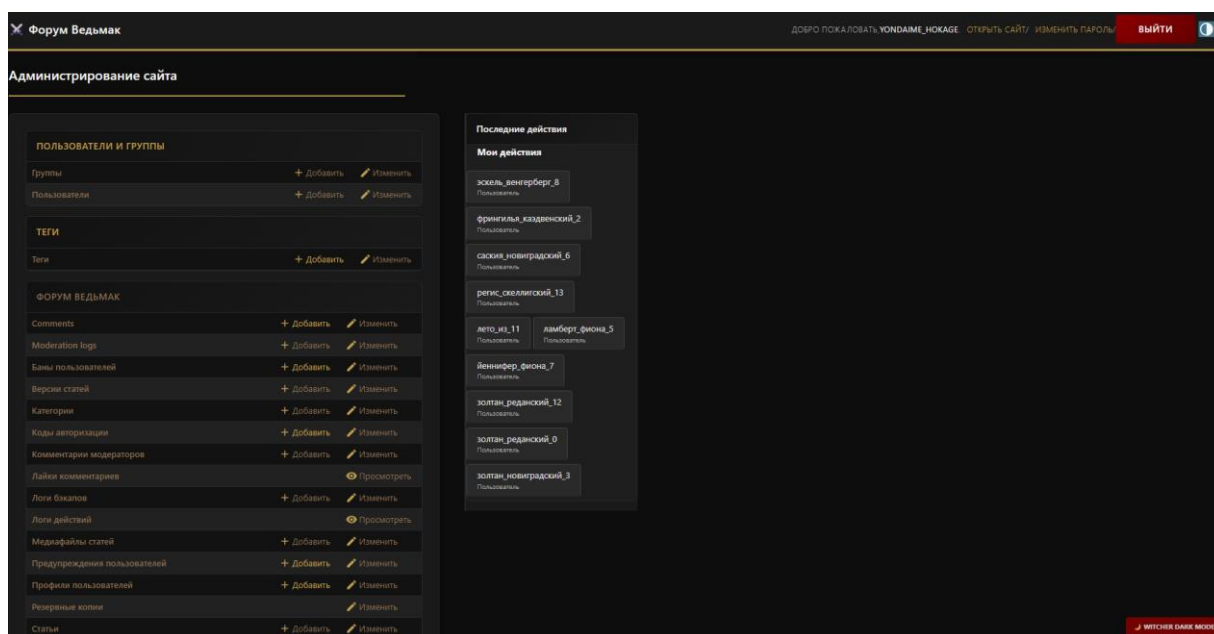


Рисунок 54 – Страница панели администратора

Администратор может управлять бэкапами. Также администратор может просматривать, создавать и восстанавливать базу данных с помощью бэкапов. Страница модели Veeam и страница управления бэкапами изображены на рисунках 55 и 56.

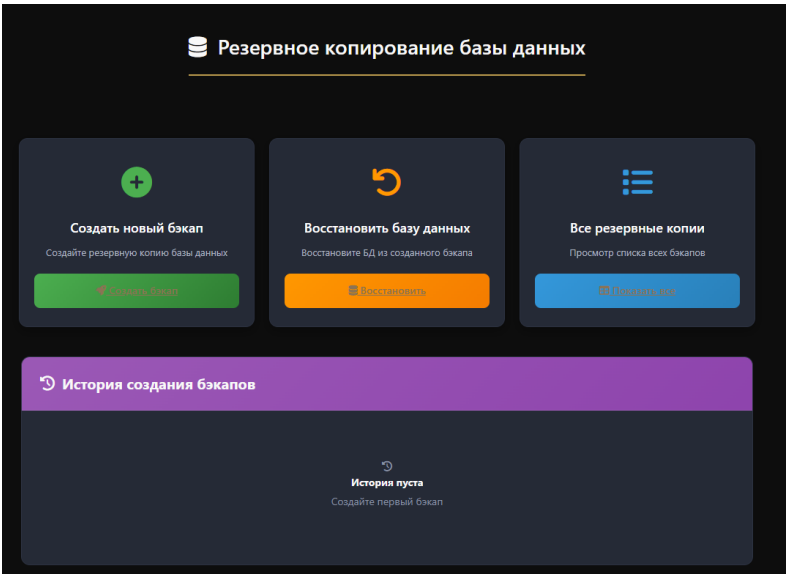


Рисунок 55 – Страница управления резервными копиями

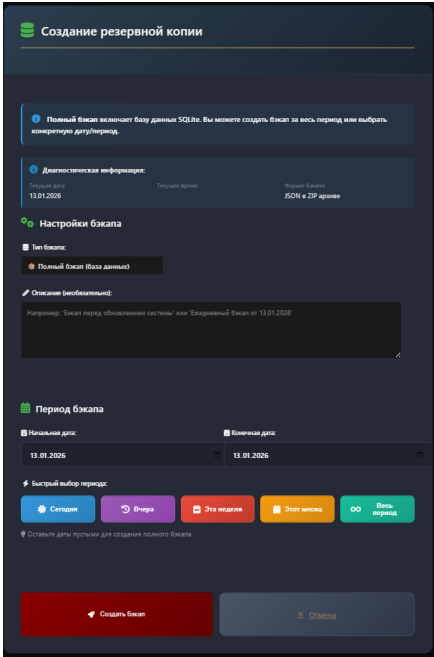


Рисунок 56 – Страница создания резервной копии базы данных

Администратор может просматривать, изменять содержимое статьи, добавлять или удалять категорию статьи, менять теги, изменять пользователя, который опубликовал статью, удалять статьи. Страница модели «Статьи» изображена на рисунке 57.

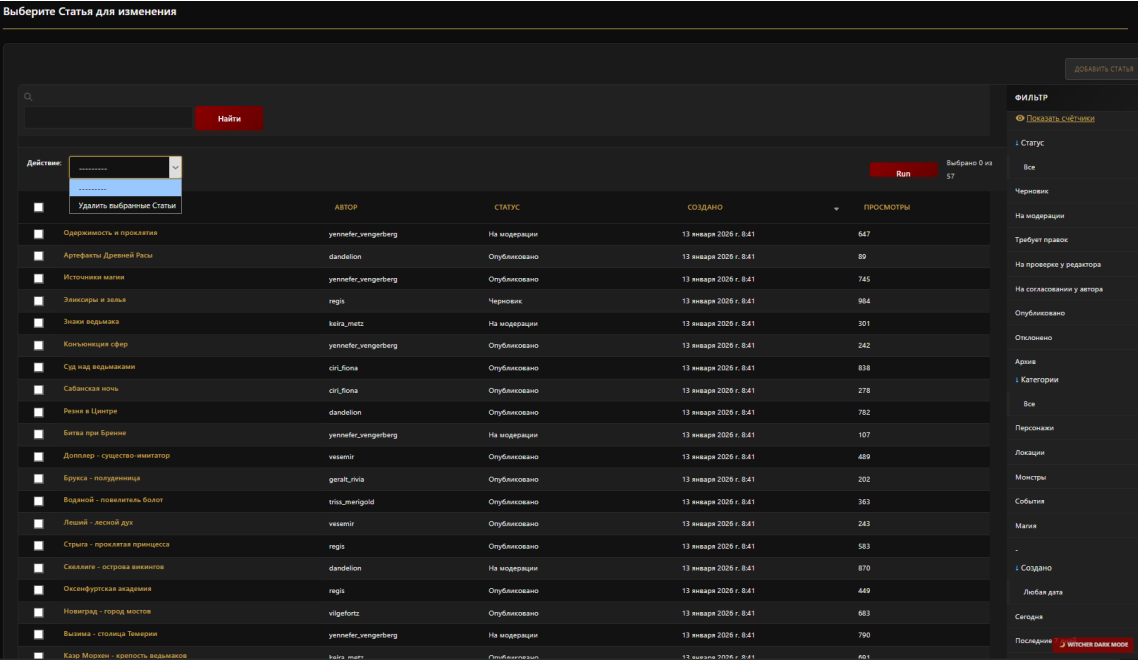


Рисунок 57 – Страница модели «Статьи»

Администратор имеет полный доступ к взаимодействиям с комментариями статьи. Страница модели «Комментарии» изображена на рисунке 58.

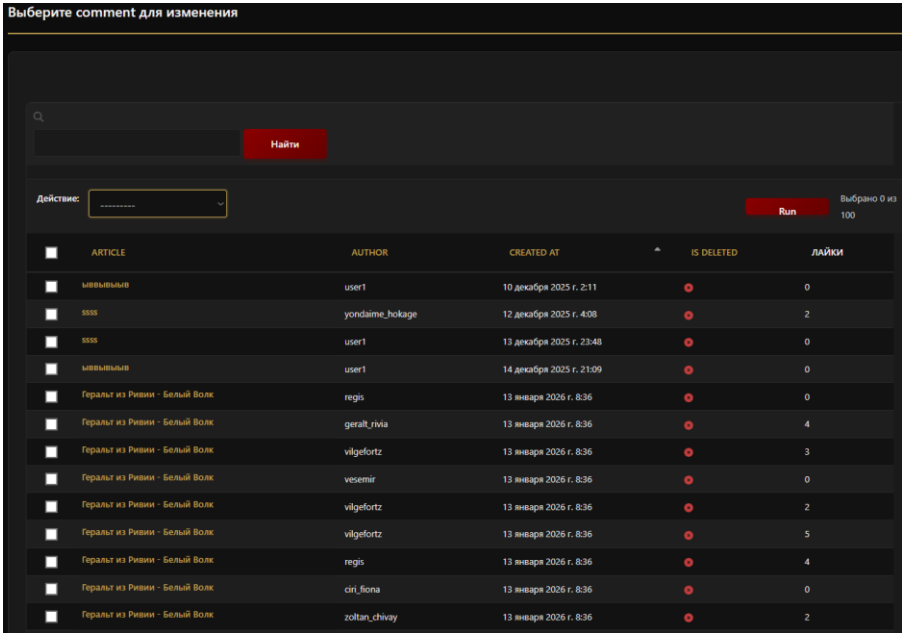


Рисунок 58 – Страница модели «Комментарии»

Администратор имеет полный доступ к взаимодействиям с категориями. Страница модели «Категории» изображена на рисунке 59.

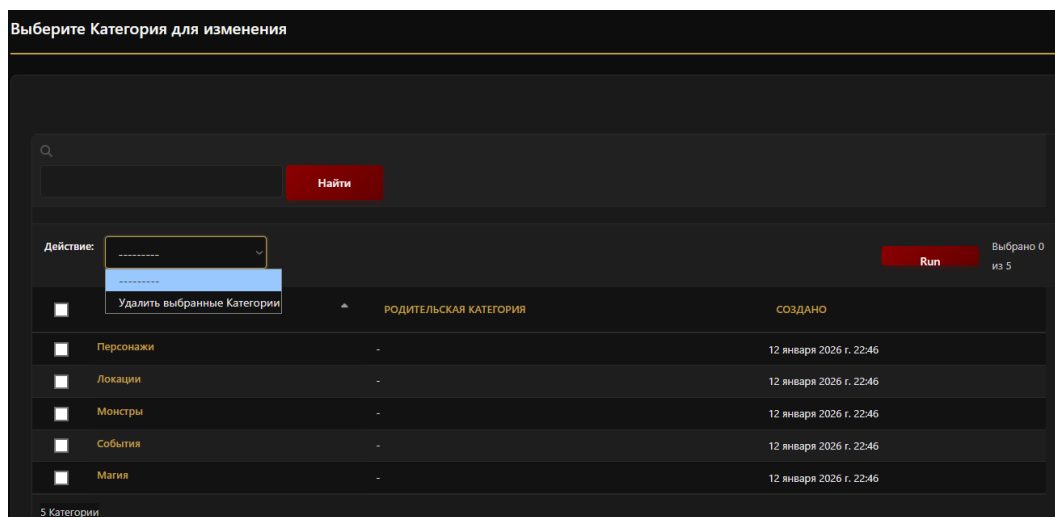


Рисунок 59 – Страница модели «Категории»

Администратор имеет полный доступ к взаимодействиям с профилями пользователей. Страница модели «Профили» изображена на рисунке 60.

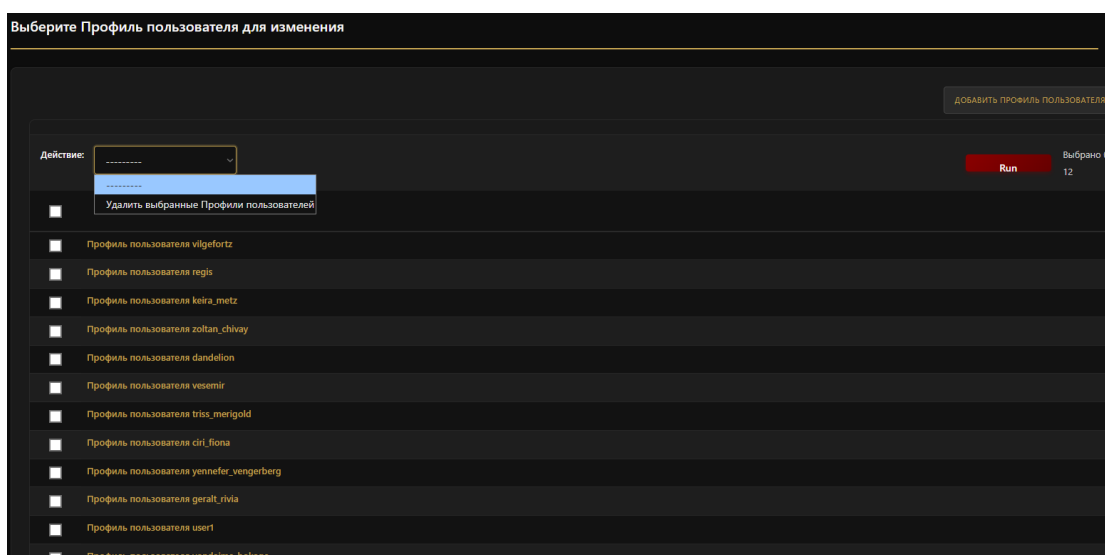


Рисунок 60 – Страница модели «Профили»

Администратор может просматривать баны и предупреждения пользователей, которые нарушили правила сайта. Страница «Предупреждения пользователей» и страница «Баны пользователей» изображены на рисунках 61 и 62.

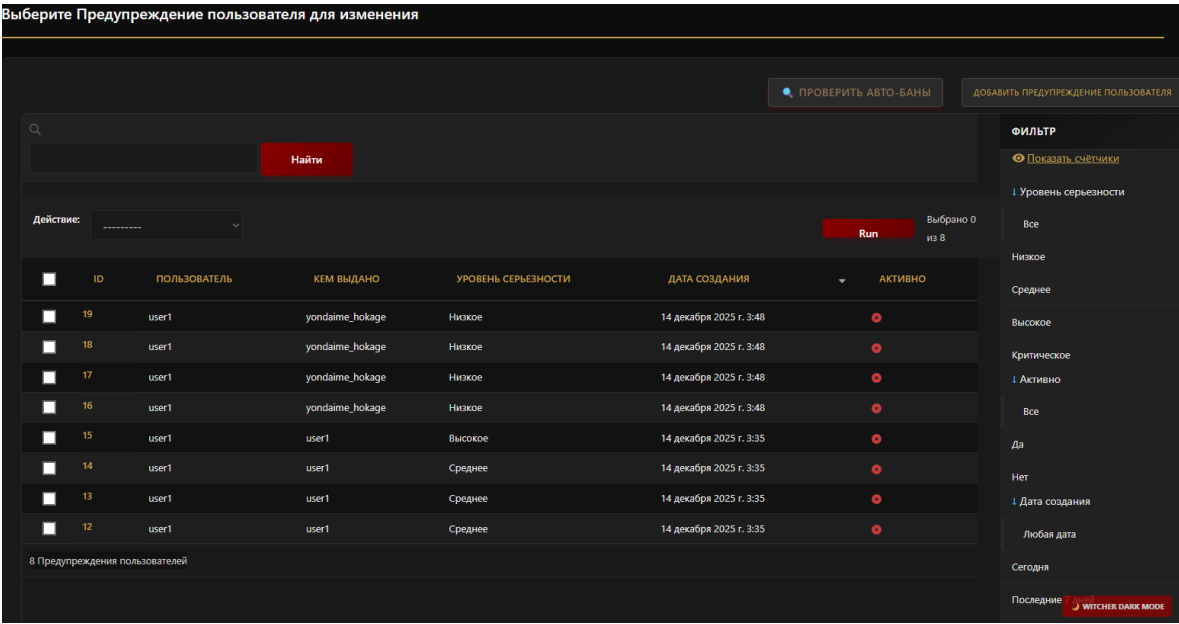


Рисунок 61 – Страница модели «Предупреждения пользователей»

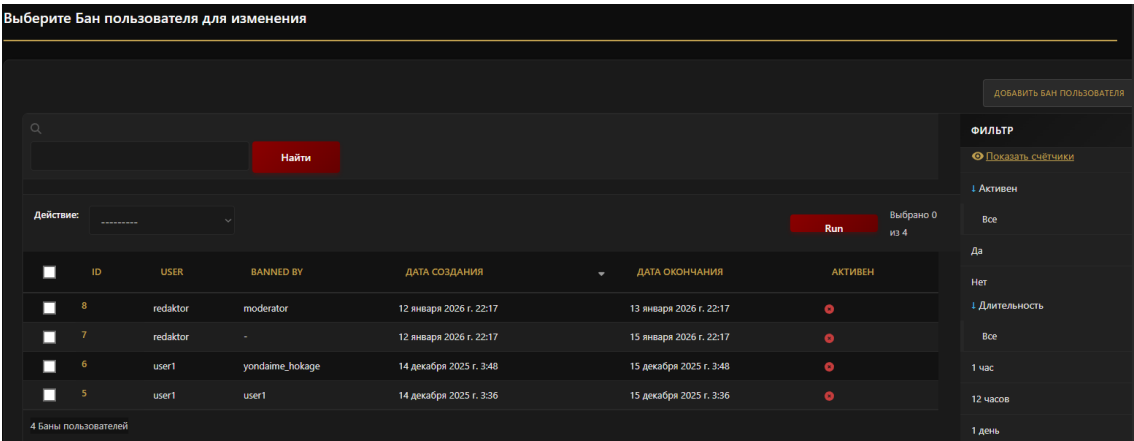


Рисунок 62 – Страница модели «Бан пользователей»

Администратор может просматривать логи операций, проходящих в системе, а также экспортировать логи в формате JSON и PDF с фильтрами. Страница модели «Логи действий» изображена на рисунке 63.

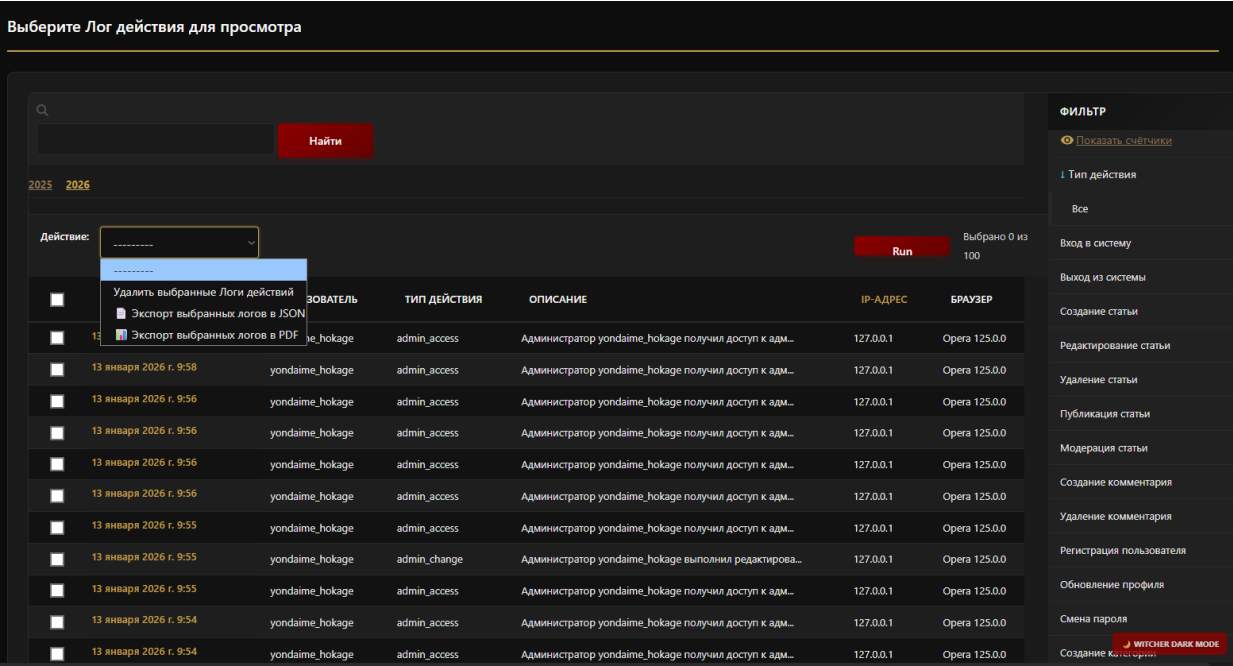


Рисунок 63 – Страница модели «Логи модерации пользователей»

Администратор может просматривать логи модерации пользователей, проходящих в системе. Страница модели «Логи модерации пользователей» изображена на рисунке 64.

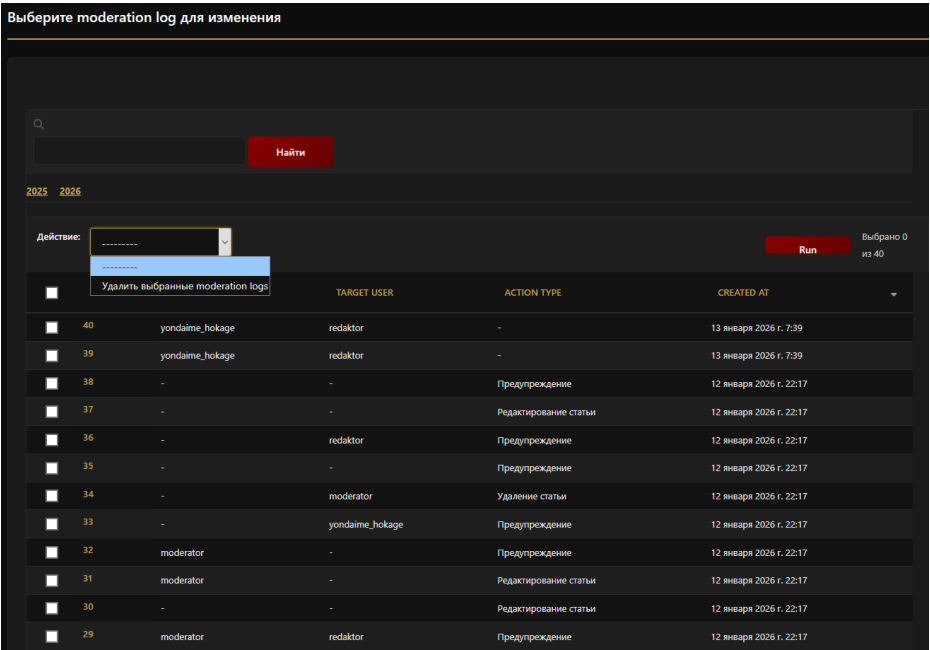


Рисунок 64 – Страница модели «Логи модерации пользователей»

Администратор может просматривать комментарии модератора к статье. Страница модели «Комментарии модератора» изображена на рисунке 65.

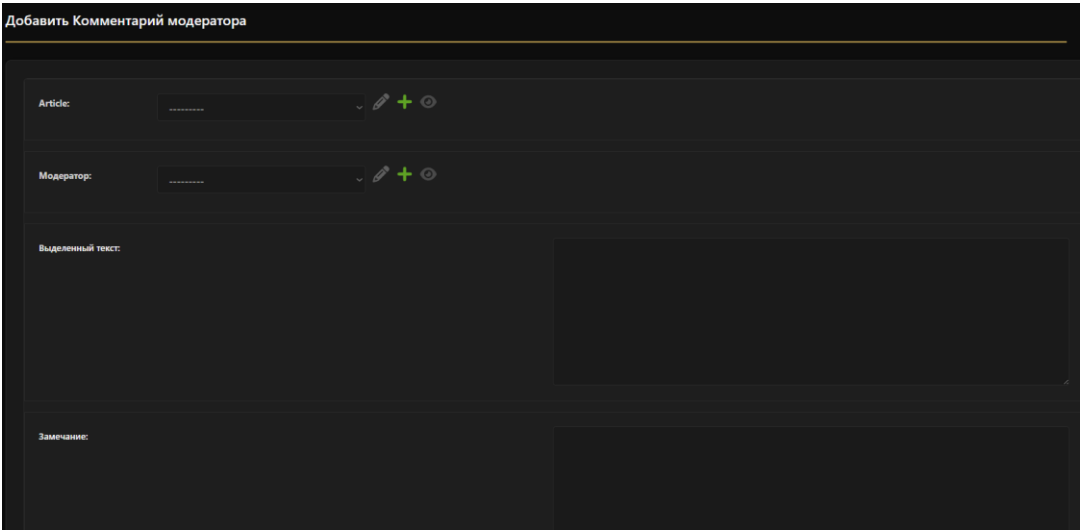


Рисунок 65 – Страница модели «Комментарии модератора»

Администратор может просматривать лайки пользователей к комментарию. Страница модели «Лайки комментариев» изображена на рисунке 66.

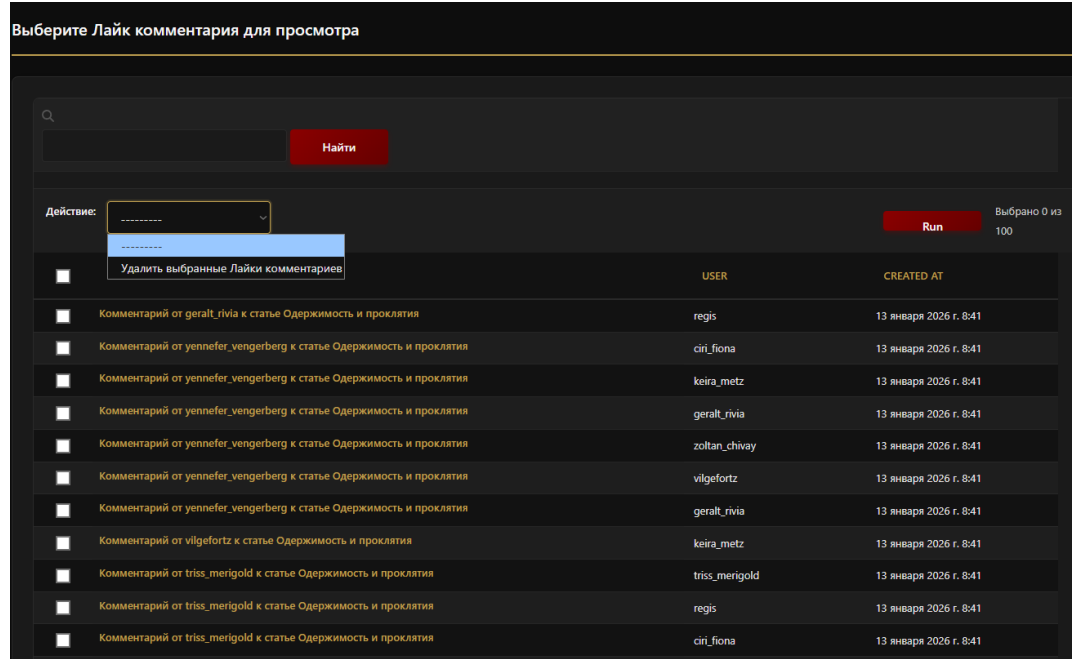


Рисунок 65 – Страница модели «Лайки комментариев»

Администратор может добавлять теги к статьям. Страница модели «Теги» изображена на рисунке 67.

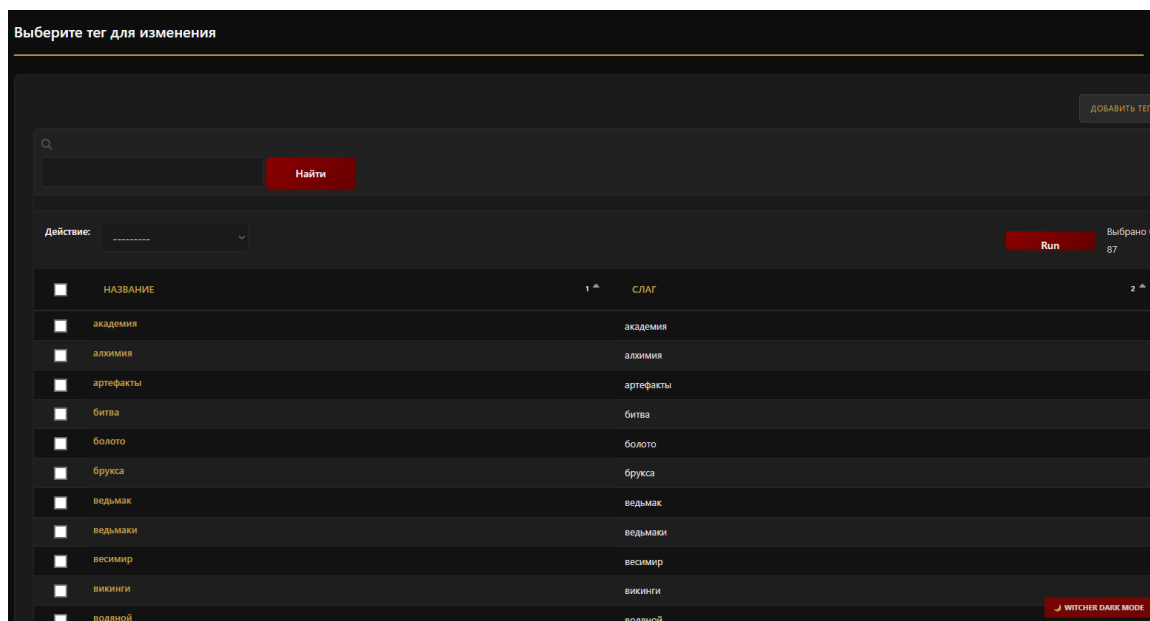


Рисунок 67 – Страница модели «Теги»

Документация охватывает полный цикл работы с системой форума от установки и настройки до ежедневного использования всеми типами пользователей. Представленные инструкции обеспечивают корректное развертывание системы и позволяют эффективно использовать функционал системы в повседневной работе.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта была разработана полнофункциональное веб–приложение «Форум» на базе фреймворка Django. Данный проект представляет законченное программное решение, направленное на автоматизацию ключевых бизнес-процессов современного форума, с фокусом на взаимодействие с конечным потребителем и внутреннее администрирование.

Итоги проекта в соответствии с поставленными целью и задачами:

1. Проведено предпроектное исследование и обоснован выбор технологического стека. В результате анализа современных инструментов был сформирован оптимальный стек технологий, полностью соответствующий требованиям проекта по производительности, безопасности, скорости разработки и удобству поддержки.

2. Выполнено комплексное проектирование архитектуры системы. На основе стандартов UML и IDEF0 была детально проработана структурная и функциональная схемы веб-приложения. Спроектирована и нормализована до третьей нормальной формы реляционная база данных, включающая 10 взаимосвязанных сущностей. Созданы детальные прототипы пользовательского интерфейса, утвердившие логику взаимодействия для всех ролей.

3. Разработано и формализовано техническое задание. ТЗ было составлено в соответствии с ГОСТ 34.602–2020.

4. Полностью реализована серверная и клиентская части системы.

5. Создан интуитивно понятный и отзывчивый пользовательский интерфейс. Интерфейс, реализованный с использованием HTML5 и CSS3, обеспечивает удобную навигацию для гостей, авторизованных пользователей и администраторов, полностью соответствуя разработанным прототипам.

6. Реализована и оптимизирована база данных. На основе спроектированной ER-модели с использованием Django ORM развернута надежная

					КП.09.02.07–1.25.221.20.ПЗ	Лист 80
Изм.	Лист	№ докум.	Подпись	Дата		

база данных PostgreSQL. Модели данных реализуют все необходимые бизнес-правила.

7. Проведено тестирование и составлена полная документация. Функциональное тестирование с использованием Django Test Case подтвердило корректность работы всех ключевых сценариев. Разработано подробное руководство пользователя, руководство по установке и настоящая пояснительная записка.

Вывод о проделанной работе:

Курсовой проект успешно реализован. Разработанная система является работоспособным, надежным и безопасным продуктом, который полностью удовлетворяет изначально сформулированным требованиям. В процессе работы были получены и закреплены практические навыки полного цикла разработки программного обеспечения: от анализа предметной области и проектирования архитектуры до реализации, тестирования и документирования. Проект демонстрирует глубокое понимание принципов веб-разработки на Django, работы с базами данных, построения API, обеспечения информационной безопасности и создания удобного пользовательского опыта.

Перспективные направления развития проекта:

Несмотря на завершенность, система обладает значительным потенциалом для масштабирования и улучшения:

1. Внедрение полноценной системы рейтинга.
2. Добавление службы поддержки, AI – ассистента для поддержки начинающих пользователей.
3. Повышение производительности и отказоустойчивости. Внедрение кеширования, использование асинхронных задач для отправки email и тяжелых отчетов, балансировка нагрузки и контейнеризация для упрощения развертывания.
4. Расширение интеграций. Подключение сервисов email-рассылок, смс-уведомлений.

Таким образом, данный курсовой проект не только представляет законченное учебное задание, но и служит качественной основой для создания коммерческого продукта, востребованного на рынке форумов.

					КП.09.02.07–1.25.221.20.ПЗ	Лист
						82
Изм.	Лист	№ докум.	Подпись	Дата		

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Python – Документация – URL: <https://habr.com/ru/hubs/python/articles/> (дата обращения: 05.09.2025) – Текст: электронный.
2. PythonWorld – Документация – URL: <https://pythonworld.ru/samouchitel-python> (дата обращения: 05.09.2025) – Текст: Электронный.
3. PythonTutorial – Документация – URL: <https://metanit.com/python/tutorial/> (дата обращения: 10.09.2025) – Текст: электронный.
4. Django – Документация – URL: <https://docs.djangoproject.com/en/5.2/> (дата обращения: 12.09.2025) – Текст: электронный.
5. PostgreSQL – Документация – URL: <https://www.postgresql.org/docs/> (дата обращения: 15.09.2025) – Текст: электронный.
6. W3Schools – Руководства – URL: <https://www.w3schools.com/> (дата обращения: 18.09.2025) – Текст: электронный.
7. MDN Web Docs – Документация – URL: <https://developer.mozilla.org/ru/> (дата обращения: 20.09.2025) – Текст: электронный.
8. Stack Overflow – Сообщество – URL: <https://stackoverflow.com/> (дата обращения: 22.09.2025) – Текст: электронный.
9. Draw.io – Инструмент – URL: <https://app.diagrams.net/> (дата обращения: 25.09.2025) – Текст: электронный.
10. ГОСТ 34.602-2020 – Стандарт – URL: <https://docs.cntd.ru/document/1200177846> (дата обращения: 28.09.2025) – Текст: электронный.
11. Django Girls – Учебник – URL: <https://tutorial.djangogirls.org/ru/> (дата обращения: 05.10.2025) – Текст: электронный.
12. PyCharm – Документация – URL: <https://www.jetbrains.com/help/pycharm/> (дата обращения: 10.10.2025) – Текст: электронный.

13. Bootstrap – Документация – URL: <https://getbootstrap.com/docs/5.3/> (дата обращения: 15.10.2025) – Текст: электронный.

14. Django REST framework – Документация – URL: <https://www.django-rest-framework.org/> (дата обращения: 20.10.2025) – Текст: электронный.

15. ReportLab – Документация – URL: <https://docs.reportlab.com/> (дата обращения: 25.10.2025) – Текст: электронный.

16. Pytest – Документация – URL: <https://docs.pytest.org/en/stable/> (дата обращения: 30.10.2025) – Текст: электронный.

17. Telegram Bot API – Документация – URL: <https://core.telegram.org/bots/api> (дата обращения: 05.11.2025) – Текст: электронный.

18. UML – Руководство – URL: <https://www.uml-diagrams.org/> (дата обращения: 10.11.2025) – Текст: электронный.

19. Реляционные базы данных – Статья – URL: <https://habr.com/ru/articles/725514/> (дата обращения: 15.11.2025) – Текст: электронный.

20. Git – Книга – URL: <https://git-scm.com/book/ru/v2> (дата обращения: 20.11.2025) – Текст: электронный.

21. PEP 8 – Руководство – URL: <https://peps.python.org/pep-0008/> (дата обращения: 25.11.2025) – Текст: электронный.

22. Two Scoops of Django – Книга – URL: <https://www.feldroy.com/books/two-scoops-of-django-3-x> (дата обращения: 01.12.2025) – Текст: электронный.

23. Django for Professionals – Книга – URL: <https://djangoforprofessionals.com/> (дата обращения: 05.12.2025) – Текст: электронный.

24. JetBrains Academy – Курс – URL: <https://www.jetbrains.com/academy/> (дата обращения: 10.12.2025) – Текст: электронный.

					КП.09.02.07–1.25.221.20.ПЗ	Лист 84
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А – Техническое задание
Министерство образования Иркутской области
Государственное бюджетное профессиональное
образовательное учреждение Иркутской области
«Иркутский авиационный техникум»
(ГБПОУИО «ИАТ»)

Техническое задание
ВЕБ-ПРИЛОЖЕНИЕ «ФОРУМ ВЕДЬМАК»

Руководитель: _____ (Е.С. Кубата)
(подпись, дата)

Студент: _____ (Ю.К. Терентьев)
(подпись, дата)

Иркутск, 2025

					КП.09.02.07–1.25.221.20.ПЗ	Лист
						85
Изм.	Лист	№ докум.	Подпись	Дата		

1. Введение

1.1 Общие сведения

Документ представляет техническое задание на создание веб-приложения «Форум», предназначенного для обсуждения тематики вселенной «Ведьмак», обмена знаниями, материалами и организации сообщества. Вид программного продукта: веб-приложение с серверной архитектурой.

1.2 Цели и задачи

Целью создания веб-приложения «Форум» является Создание тематического форума для фанатов вселенной «Ведьмак» с возможностью публикации, обсуждения и систематизации материалов.

Задачи веб-приложения включают:

- Реализация системы пользователей с регистрацией и авторизацией.
- Создание системы публикации статей с поддержкой текста, изображений и тегов.
- Реализация системы комментариев и оценок (лайков).
- Разработка панели администратора для модерации контента и управления пользователями.
- Обеспечение возможности поиска, фильтрации и экспорта данных.

2 Основания для разработки

2.1 Нормативные документы

Документ основывается на следующих нормативных документах:

- ГОСТ 34.602-2020 «Техническое задание на создание автоматизированной системы».

					КП.09.02.07–1.25.221.20.ПЗ	Лист 86
Изм.	Лист	№ докум.	Подпись	Дата		

- ГОСТ Р 56477-2015 «Проектирование и внедрение информационных систем».
- Методические указания к курсовому проекту по МДК 02.01.
- Приказ образовательного учреждения об утверждении тем курсовых проектов.

2.2 Проектные документы

Проектные документы включают:

- Задание на курсовой проект.
- Методические указания по выполнению дипломного проекта.
- Пояснительная записка.
- Руководство пользователя.

3 Назначение системы

3.1 Общее описание

Веб-приложение «Форум» предназначено для использования пользователя форума и административным персоналом. Приложение поддерживает процессы просмотра статьи, создания и редактирования статей, создание комментариев, а также управления статьями и анализа деятельности.

4. Требования к системе

4.1 Функциональные требования

- 1 Функционал роли «Пользователь»
 - Просмотр списка статей и сообщений.
 - Регистрация с подтверждением адреса почты.
 - Авторизация.
 - Поиск по форуму.

- Создание, редактирование, удаление своих статей и сообщений.
- Комментирование, лайки.
- Личный кабинет с историей активности.
- Смена пароля, адреса почты.
- Просмотр собственной статистики на сайте.
- Общение с другими пользователями и административным персоналом.

2 Функционал роли «Модератор»

- Все функции пользователя.
- Управление контентом: удаление комментариев, изменение и добавление категорий.

- Модерация статей.
- Предупреждение и блокировка пользователей.

3 Функционал роли «Редактор»

- Все функции пользователя.
- Редактирование статей.
- Управление контентом: удаление комментариев, изменение и добавление категорий.

4 Функционал роли «Администратор»

- Все функции модератора и редактора.
- Управление пользователями: блокировка, смена ролей.
- Управление разделами форума.
- Просмотр логов, статистики.
- Экспорт отчетов в PDF/Excel.
- Резервное копирование БД.

4.2 Технические требования

4.2.1 Производительность

- Количество одновременных пользователей: до 200 активных сессий;

					КП.09.02.07–1.25.221.20.ПЗ	Лист 88
Изм.	Лист	№ докум.	Подпись	Дата		

- Количество запросов в секунду (RPS/QPS): 100 RPS;
- Среднее время отклика: не более 15 мс.

4.2.2 Надежность

- Валидация пользовательского ввода на клиенте – все формы, поля, файлы и параметры должны проверяться перед отправкой запроса на сервер.
- Валидация пользовательского ввода на сервере – все входные данные, независимо от наличия клиентской проверки, проходят повторную строгую валидацию.
- Защита от SQL–инъекций за счёт применения параметризованных запросов или ORM с встроенной защитой.
- Логирование всех ключевых действий на клиентской части.

4.3 Эксплуатационные требования

1 Среда функционирования:

- Поддержка современных браузеров: Яндекс браузер, Chrome, Firefox, Edge, Safari.

2 Установка и развёртывание:

- Разработка инструкции по установке и настройке.

3 Резервирование и восстановление:

- Ручное резервное копирование базы данных.
- Восстановления базы данных из резервной копии.

4 Интеграция:

- Telegram API: интеграция с Telegram Bot API для уведомлений и управления.
- Email SMTP: интеграция с почтовыми сервисами для отправки писем.
- PDF Generation: интеграция с ReportLab для генерации PDF документов.

					КП.09.02.07–1.25.221.20.ПЗ	Лист 89
Изм.	Лист	№ докум.	Подпись	Дата		

4.4 Требования к информационной безопасности

Веб-приложение должно обеспечивать защиту данных и предотвращение несанкционированного доступа в соответствии со следующими требованиями:

1. Аутентификация и авторизация:
 - Аутентификация по имени пользователя и паролю.
 - Хранение паролей пользователей в зашифрованном виде с использованием встроенного механизма хэширования Django: PBKDF2 с SHA256.
 - Обязательная email-верификация при регистрации с временным кодом.
2. Защита от распространённых уязвимостей:
 - Защита от несанкционированного доступа к данным - проверка прав доступа к каждому ресурсу на серверной стороне.

5 Требования к техническому обеспечению

5.1 Сервер

- Количество процессоров: 1 шт.
- Количество ядер одного процессора: 2 ядра.
- Тактовая частота одного процессора: 2.5 ГГц.
- Оперативная память: 4 Гб.
- Пропускная способность сети: 100 Мбит/с.

5.2 Хранилище данных

- Тип накопителей: SSD.
- Общий объем хранилища: 100 Гб.
- IOPS на чтение: 1000.
- IOPS на запись: 500.

5.3 Клиентские устройства

					КП.09.02.07–1.25.221.20.ПЗ	Лист 90
Изм.	Лист	№ докум.	Подпись	Дата		

5.3.1 Минимальные требования для персонального компьютера (ноутбука)

- Количество ядер процессора: 2 ядра.
- Тактовая частота процессора: 1.6 ГГц.
- Оперативная память: 4 Гб.
- Тип диска: HDD или SSD.
- Свободного места на диске: не менее 1 Гб.
- Разрешение дисплея: Full HD и выше.
- Пропускная способность сети: 10 Мбит/с.

5.4 Сетевые требования

- Доступ к сети Интернет со скоростью не менее 10 Мбит/с.
- Поддержка следующих сетевых протоколов: HTTP/1.1, HTTPS, WebSocket.
- Поддержка IPv4 (обязательно), IPv6 (при наличии).
- Возможность работы через защищённые порты 443 (HTTPS) и 80 (HTTP).

6 Требования к программному обеспечению

6.1 Сервер

- Операционная система Ubuntu Server 24.04 LTS.
- Сервер базы данных: PostgreSQL 16.
- Веб-сервер: Nginx 1.24.
- Интерпретатор: Python 3.12.
- Менеджер пакетов: pip 24.0.
- Система контроля версий: Git 2.45.

6.2 Персональный компьютер (ноутбук)

1. Операционная система Windows 10 / 11 (64-bit.
2. Веб-браузеры (актуальные версии, не старше двух релизов):
Яндекс.Браузер 24.9+, Google Chrome 128+, Microsoft Edge 128.
3. Необходимые компоненты браузера:
 - Поддержка HTML5, CSS3.
 - Поддержка Web Storage API.
 - Поддержка WebSocket.
 - Включена поддержка JavaScript и cookies.
 - Поддержка PDF просмотра в браузере.

7 Требования к тестированию

7.1 Общие требования

Тестирование веб-приложения должно охватывать все ключевые аспекты функциональности, производительности и безопасности. Этап тестирования должен включать:

- Документирование.
- Функциональное тестирование.

7.1.1 Документирование

Тестовые сценарии – охватывающие ключевые функциональные и нефункциональные требования к системе для проверки работы:

- в типовых и исключительных ситуациях;
- при вводе корректных и некорректных данных;
- при различных ролях пользователей (администратор, пользователь, модератор и редактор).

Тестовая документация должна включать:

					КП.09.02.07–1.25.221.20.ПЗ	Лист 92
Изм.	Лист	№ докум.	Подпись	Дата		

- план тестирования;
- перечень тестовых сценариев;
- отчёт о результатах тестирования и выявленных дефектах.

7.1.2 Функциональное тестирование

- Проверка корректности работы веб-приложения в соответствии с функциональными требованиями.
- Проверка работы пользовательских сценариев.
- Проверка корректности взаимодействия с внешними системами.
- Проверка обработки ошибок и сообщений пользователю при некорректных действиях.

7.1.3 Нагрузочное тестирование

- Проверка производительности для подтверждения выполнения требований к производительности.

8 Организационно-технические требования

8.1 Этапы разработки

В таблице 1 представлены сроки и этапы разработки веб-приложения «Форум».

Таблица 1 – Сроки и этапы разработки веб-приложения

№	Этап	Срок выполнения
1	Предпроектное исследование предметной области и выбор технологического стека	20.09.2025
2	Разработка технического задания и проектной документации	30.09.2025

Продолжение таблицы 1

3	Разработка веб-приложения	02.11.2025
4	Разработка тестирования	05.12.2025
5	Документирование веб-приложения	20.12.2025

					КП.09.02.07–1.25.221.20.ПЗ	Лист
						94
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б – Листинг Urls

```

from .views import user_management, FAQView, HelpView, banned_page
from django.contrib.auth import views as auth_views
from .views import register
from django.contrib.auth import views as auth_views
from accounts.forms import CustomAuthenticationForm
from django.urls import path
from . import views
from django.contrib.admin.views.decorators import staff_member_required
from . import moderation_views
from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required
app_name = 'wiki'
urlpatterns = [
    # Основные страницы
    path("", views.home, name='home'),
    path('search/', views.search, name='search'),
    # Категории
    path('category/<slug:slug>/', views.category_detail, name='category_detail'),
    # Статьи
    path('article/create/', views.article_create, name='article_create'),
    path('article/<slug:slug>/', views.article_detail, name='article_detail'),
    path('article/<slug:slug>/edit/', views.article_edit, name='article_edit'),
    path('article/<slug:slug>/moderate/', views.article_moderate, name='article_moderate'),
    path('article/<slug:slug>/moderate/enhanced/', views.article_moderate_enhanced,
name='article_moderate_enhanced'),
    path('article/<slug:slug>/like/', views.toggle_article_like, name='toggle_article_like'),
    path('article/<slug:slug>/delete/', views.article_delete, name='article_delete'),
    path('article/<slug:slug>/resubmit/', views.article_resubmit, name='article_resubmit'),
    path('article/<slug:slug>/delete-by-author/', views.article_delete_by_author,
name='article_delete_by_author'),
    # Модерация
    path('moderation/', views.moderation_queue, name='moderation_queue'),
    path('my-articles/', views.my_articles, name='my_articles'),
    path('article/<slug:slug>/send-to-editor/', views.send_to_editor, name='send_to_editor'),
    # Модерация комментариев
    path('article/<slug:slug>/add-moderation-comment/', views.add_moderation_comment,
name='add_moderation_comment'),
    path('moderation-comment/<int:comment_id>/resolve/', views.resolve_moderation_comment,
name='resolve_moderation_comment'),
    path('moderation-comment/<int:comment_id>/delete/', views.delete_moderation_comment,
name='delete_moderation_comment'),
    # Редактура и ревью
    path('article/<slug:slug>/editor-review/', views.editor_review, name='editor_review'),
    path('article/<slug:slug>/author-review/', views.author_review, name='author_review'),
    # Медиа
    path('media/<int:media_id>/delete/', views.delete_media, name='delete_media'),
    # Управление категориями
    path('categories/', views.category_management, name='category_management'),

```

```

path('categories/create/', views.category_create, name='category_create'),
path('categories/<int:category_id>/edit/', views.category_edit, name='category_edit'),
path('categories/<int:category_id>/delete/', views.category_delete, name='category_delete'),
path('categories/<int:category_id>/toggle-featured/', views.category_toggle_featured,
    name='category_toggle_featured'),
path('categories/json/', views.get_categories_json, name='categories_json'),
# Пользовательские профили
path('user/<str:username>/', views.user_public_profile, name='user_public_profile'),
path('liked-articles/', views.liked_articles, name='liked_articles'),
path('accounts/profile/', views.profile, name='profile'),
# Отладка
path('debug/test-like/', views.debug_test_like, name='debug_test_like'),
path('debug/article-like/<slug:slug>/', views.debug_article_like, name='debug_article_like'),
# УТИЛИТЫ
path('clean-latex/', views.clean_all_articles_latex, name='clean_latex'),
path('editor/dashboard/', views.editor_dashboard, name='editor_dashboard'),
path('comment/<int:comment_id>/delete/', views.delete_comment, name='delete_comment'),
# Управление пользователями
path('user-management/', views.user_management, name='user_management'),
#Помощь
path('help/', HelpView.as_view(), name='help'),
path('help/faq/', FAQView.as_view(), name='faq'),
# Сообщения
path('messages/', views.messages_list, name='messages_list'),
path('messages/<str:folder>/', views.messages_list, name='messages_list'),
path('message/create/', views.message_create, name='message_create'),
path('message/create/<int:recipient_id>/', views.message_create, name='message_create'),
path('message/<int:message_id>/', views.message_detail, name='message_detail'),
path('message/<int:message_id>/delete/', views.message_delete, name='message_delete'),
path('message/send-quick/<int:user_id>/', views.send_quick_message,
name='send_quick_message'),
path('messages/unread-count/', views.get_unread_count, name='get_unread_count'),
# Аутентификация
path('login/', auth_views.LoginView.as_view(
    template_name='registration/login.html',
    authentication_form=CustomAuthenticationForm,# ИЗМЕНИТЬ путь
    redirect_authenticated_user=True
), name='login'),
path('logout/', auth_views.LogoutView.as_view(next_page='wiki:home'), name='logout'),
path('register/', register, name='register'),
# Восстановление пароля
path('password-reset/', views.password_reset_request, name='password_reset_request'),
path('password-reset/verify/', views.password_reset_verify, name='password_reset_verify'),
path('password-reset/complete/', views.password_reset_complete,
name='password_reset_complete'),
# Telegram Auth
path('auth/telegram/', views.telegram_auth, name='telegram_auth'),
path('auth/telegram/callback/', views.telegram_callback, name='telegram_callback'),
path('auth/telegram/disconnect/', views.telegram_disconnect, name='telegram_disconnect'),
path('auth/telegram/code/', views.telegram_auth_code, name='telegram_auth_code'),

```

```

    path('auth/telegram/generate-code/', views.telegram_generate_test_code,
name='telegram_generate_test_code'),
    path('auth/telegram/link/', views.telegram_link_with_code, name='telegram_link_with_code'),
    # Telegram Web App Auth
    path('auth/telegram/webapp/', views.telegram_webapp_login, name='telegram_webapp_login'),
    path('auth/telegram/webapp/callback/', views.telegram_webapp_callback,
name='telegram_webapp_callback'),
    path('auth/telegram/quick/', views.telegram_quick_login, name='telegram_quick_login'),
    path('admin/group-permissions/', views.group_permissions_info, name='group_permissions_info'),
    path('article/<slug:slug>/resubmit/', views.article_resubmit, name='article_resubmit'),
    path('article/<slug:slug>/delete-by-author/', views.article_delete_by_author,
name='article_delete_by_author'),
    path('article/<slug:slug>/return-to-draft/', views.article_return_to_draft,
name='article_return_to_draft'),
    path('admin/action-logs/', views.action_logs_view, name='action_logs'),
    path('admin/action-logs/export-json/', views.export_logs_json, name='export_logs_json'),
    path('debug/create-log/', views.debug_create_log, name='debug_create_log'),
    path('debug/test-logs/', views.debug_test_logs, name='debug_test_logs'),
    path('tutorial/mark-seen/<str:tutorial_type>', views.mark_tutorial_seen,
name='mark_tutorial_seen'),
    path('tutorial/disable/', views.disable_tutorials, name='disable_tutorials'),
    path('tutorial/reset/', views.reset_tutorials, name='reset_tutorials'),
    path('article/<slug:slug>/export-pdf/', views.export_article_pdf, name='export_article_pdf'),
    path('articles/export/', views.export_articles_list, name='export_articles_list'),
    path('help/', HelpView.as_view(), name='help'),
    path('help/faq/', FAQView.as_view(), name='faq'),
    path('statistics/', views.StatisticsView.as_view(), name='statistics'),
    path('statistics/export/', views.ExportStatsView.as_view(), name='export_statistics'),
    path('api/update_stats/', views.update_stats_api, name='update_stats_api'),
    path('articles/', views.article_list, name='article_list'),
    path('statistics/export/pdf/', views.export_statistics_pdf, name='export_statistics_pdf'),
    path('statistics/export/json/', views.export_statistics_json, name='export_statistics_json'),
    path('comment/<int:comment_id>/like/', views.comment_like, name='comment_like'),
    path('profile/', views.profile, name='profile'),
    path('statistics/', views.user_statistics, name='statistics'),
    path('statistics/<str:username>', views.user_statistics, name='user_statistics'),
    path('censorship-dashboard/', staff_member_required(views.censorship_dashboard),
name='censorship_dashboard'),
    path('my-censorship-warnings/', views.my_censorship_warnings, name='my_censorship_warnings'),
    path('admin/user-warnings/', staff_member_required(views.user_warnings_list),
name='user_warnings_list'),
    path('admin/reset-warnings/<int:user_id>', staff_member_required(views.reset_user_warnings),
        name='reset_warnings'),
    path('moderation/dashboard/', moderation_views.moderation_dashboard,
name='moderation_dashboard'),
    path('moderation/search/', moderation_views.user_search, name='user_search'),
    path('moderation/warned/', moderation_views.warned_users_list, name='warned_users_list'),
    path('moderation/banned/', moderation_views.banned_users_list, name='banned_users_list'),
    path('moderation/logs/', moderation_views.moderation_logs, name='moderation_logs'),
    path('moderation/user/<int:user_id>', moderation_views.user_detail, name='user_detail'),

```

```

path('moderation/warn/<int:user_id>/', moderation_views.warn_user, name='warn_user'),
path('moderation/ban/<int:user_id>/', moderation_views.ban_user, name='ban_user'),
path('moderation/unban/<int:user_id>/', moderation_views.unban_user, name='unban_user'),
path('banned/', banned_page, name='banned'),
]

```

					КП.09.02.07–1.25.221.20.ПЗ	Лист
						98
Изм.	Лист	№ докум.	Подпись	Дата		