					ОКЭИ 09.02.07 9025 77 П		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Скавренюк Н.Н.			Отчёт по производственной [1]практике	Лит.	Лист 2 / Листов 31
Провер.							
Реценз.						Отделение — очное гр. 4ВБ3	
Н. Контр.							
Утверд.							

Содержание

Введение

Образование является одной из важнейших сфер в жизни каждого человека. Оно позволяет людям не только расширять кругозор и приобретать новые знания, но и формировать профессиональные компетенции, делать научные и технические открытия, делиться знаниями с другими участниками сообщества. Особенно важно это для технических и инженерных дисциплин, где непрерывное обучение и доступ к актуальной информации играют ключевую роль в профессиональном развитии.

С учетом активного роста населения, урбанизации, цифровизации отрасли и внедрения «умных» технологий возрастает потребность в создании доступных, понятных и надежных информационных источников по электроэнергетике. Учитывая быстро меняющийся рынок, обновление стандартов и рост объемов информации, интерактивные цифровые платформы становятся основой современной системы образования в энергетической отрасли.Разработка автоматизированной системы для блога об электроэнергетике является одной из актуальных задач для сферы образования. В современном мире просто необходимо создать систему, которая позволит быстрее и удобнее изучать информацию об электроэнергетике и работе заводов, поставляющих электроэнергию.

Автоматизированная система для блога об электроэнергетике представляет собой программное обеспечение, способное дать пользователям просматривать блоги об электроэнергетике. Она устраняет необходимость постоянного участия ассистента и обеспечивает высокую степень автоматизации процесса.

Одним из важнейших преимуществ автоматизированной системы является высокий уровень независимости и доступности информации. Пользователь может самостоятельно находить нужные материалы, взаимодействовать с системой без постоянного участия администратора, а благодаря внедрению поисковых алгоритмов и системы фильтрации, время поиска информации значительно сокращается.В современном мире, где развитие технологий и автоматизация занимают все более важное место, разработка автоматизированной системы для блога об электроэнергетике становится неотъемлемой задачей в сфере образования. Она повысит качество блогов об электроэнергетике, а так же предоставит более удобный интерфейс.

Цель производственной практики состоит в изучении и разработке информационной системы для блога об электроэнергетике. Главной целью является создание эффективной, удобной и безопасной системы, которая удовлетворит потребности и ожидания пользователей. Так же необходимо поддерживать обратную связь с пользователей, это позволит разработать программное обеспечение более удобным и идеальным для пользователей и сделает программное обеспечение популярнее.

Основными задачами в разработке программного обеспечения для блога об электроэнергетике являются:

– провести анализ требований современной аудитории;
– разработать дизайн будущего веб–приложения;
– разработать веб–приложение на основе созданного дизайна;
– создать и подключить базу данных к веб–приложению;
– разработать руководство разработчику и пользователю;
– изучить ревьюирование и рефакторинг програмного кода;
– выбрать нужный способ ревьюирования и рефакторинга програмного кода;
– провести тестирование веб-приложения.

Разработка автоматизированной системы для блога об электроэнергетике предполагает интеграцию различных технологий для создания эффективного и удобного программного обеспечения. Это включает в себя разработку интуитивно понятного интерфейса, внедрение полезных функций и так далее .

Создание веб–сайта для блога об электроэнергетике является основной задачей для продвижения популяризации сферы образования об электроэнергетике. Конкуренция в сфере образования постоянно растёт, и каждый сайт по электроэнергетике демонстрирует свои решения для пользователей. Создание веб–страницы позволит пользователям не только ознакомиться с блогами об электроэнергетике, но и новостями, связанные с сайтом для блога об электроэнергетике.

Также необходимо учитывать, для какой аудитории разрабатывается программное обеспечение для блога об электроэнергетике. Это необходимо, так как у всех пользователей свои желания и предпочтения в образовании. Необходимо предоставить пользователям удобный и понятный интерфейс программного обеспечения, чтобы пользователям было понятно ориентироваться на веб–странице. Ни в коем случае нельзя делать программное обеспечение с большим количеством элементов, это может запутать пользователя и он не сможет разобраться в системе и просто уйдёт со страницы.

Разработка автоматизированной системы для блога об электроэнергетике имеет большой потенциал для улучшения процесса обучения, снижения затрат на обслуживание и повышения удовлетворенности пользователей. Продолжающееся развитие технологий и инноваций в этой области предоставляет возможности для создания все более совершенных и эффективных систем, отвечающих потребностям современных пользователей.

В рамках работы планируется изучить основные принципы работы автоматизированных систем для блога об электроэнергетике, а также провести обзор существующих технологий и применяемых решений в данной области. После этого будет проведён анализ требований потенциальных пользователей и определение функциональных и нефункциональных требований.

Объект исследования – блог об электроэнергетике.

Предмет исследования – сайт для блога об электроэнергетике и предоставление актуальной информации о нем.

1 Технико–экономическое обоснование

Современное общество находится в условиях стремительного развития информационных технологий и глобальной цифровизации. В этой среде образование приобретает не только важнейшее значение, но и новые формы реализации – гибкие, доступные и интерактивные. Одной из таких форм является образовательный блог – веб–приложение, предназначенное для размещения и распространения обучающих материалов, организации взаимодействия между автором и аудиторией, а также стимулирования самостоятельного изучения информации.

В последние годы наблюдается существенный рост популярности онлайн–обучения. Люди активно используют блоги, платформы и каналы для изучения новых тем, профессионального развития, освоения навыков и подготовки к экзаменам. Образовательный блог – это не только способ передачи знаний, но и форма мотивации и вовлечения через личное общение, обсуждение и практические рекомендации.

Блог, в отличие от традиционных форм обучения, доступен пользователям в любое время и из любого места. Это особенно актуально для людей с ограниченным временем (например, работающих специалистов или студентов), а также для тех, кто проживает в удалённых регионах.

В отличие от учебников, блог позволяет подавать информацию в живой, неформальной манере, сочетая текст, изображения, видео, тесты и комментарии. Такой подход увеличивает вовлеченность и способствует лучшему усвоению материала.

В блоге пользователи могут оставлять комментарии, задавать вопросы, участвовать в обсуждениях и оценивать материалы. Это создаёт интерактивную образовательную среду, позволяющую авторам улучшать контент на основе отзывов, а пользователям – чувствовать своё участие и значимость.

Сегодня особой ценностью обладает личный опыт – люди охотно читают и доверяют блогерам, которые делятся собственными знаниями, опытом и рекомендациями. Автор образовательного блога может построить доверительные отношения с аудиторией и сформировать вокруг себя сообщество единомышленников.

Филиал ПАО «Россети Волга» – «Оренбургэнерго» является крупнейшей электросетевой компанией Оренбургской области, обеспечивающей транспортировку электроэнергии и эксплуатацию распределительных сетей. Организация ПАО «Россети Волга» – «Оренбургэнерго» расположен по адрессу город Оренбург, Центральный район, улица Маржала Жукова, дом 44.

Компания «Оренбургэнерго» является одним из подразделением ПАО «Россети Волга» и осуществляет деятельность на территории Оренбургской области. Основная деятельность филиала «Оренбургэнерго» является:

– обеспечение бесперебойного электроснабжения потребителей;

– модернизация и ремонт электрических сетей;

– обслуживание объектов распределительных сетей 0,4–110 кВ;

– обучение и повышение квалификации персонала.

Филиал ПАО «Россети Волга» – «Оренбургэнерго» отвечает за надежное энергоснабжение потребителей Оренбургской области, а также за подготовку специалистов в области энергетики. Однако на данный момент процесс обучения новых сотрудников, повышения квалификации и популяризации профессии электрика среди молодежи осуществляется традиционными методами – лекциями, печатными пособиями и очными курсами. Такой подход имеет ряд недостатков:

– ограниченная доступность материалов – обучение возможно только в рамках специальных курсов, что требует временных и финансовых затрат;

– ограниченная интерактивность – традиционные методы не дают возможности вовлечь молодежь и потенциальных сотрудников в изучение темы;

– отсутствие цифровой образовательной среды – на текущий момент предприятие не использует единую платформу для дистанционного обучения сотрудников и популяризации профессии среди студентов.

Разработка образовательного блога по электроэнергетике решает указанные проблемы, предоставляя свободный доступ к качественным учебным материалам и статьям. Это позволит не только обучать персонал, но и привлекать новых специалистов в отрасль.

У каждой организации присутствует своя организационная структура предприятия. Организационная структура – это формальная система, которая определяет, как управляются и координируются различные функциональные направления, подразделения и индивиды в организации. Организационная структура определяет иерархические отношения между сотрудниками, структуру управления, потоки коммуникации, полномочия и обязанности сотрудников.

Организационная структура описывает, как различные части компании связаны друг с другом, как они взаимодействуют и какие роли они играют в достижении целей компании. Организационная структура может быть представлена в виде диаграммы, графика или другой визуальной формы.

Организационная структура может быть децентрализованной, когда полномочия распределены между разными отделами и менеджерами, или централизованной, когда управление и контроль находятся в одних руках. Также существует множество различных видов организационной структуры, каждая из которых имеет свои преимущества и недостатки, в зависимости от размеров и особенностей компании.

Организационная структура филиала «Оренбургэнерго» состоит из следующих подразделений:

– производительно–технический отдел;
– служба технологического обеспечения;
– служба оперативной эксплуатации средств технологического управления;
– служба радиосвязи;
– служба телекоммуникационных сетей;
– линейно–кабельный участок;
– служба автоматизированных систем диспетчерского управления;
– отдел внедрения автоматизированных систем управления.

При разработке информационной системы образовательного блога об электроэнергетике необходимо учитывать, для какой аудитории разрабатывается данный продукт. Образовательный блог об электроэнергетике предназначен для нескольких категорий пользователей:

– сотрудники Оренбургэнерго могут использовать данную платформу для повышения квалификации, изучения новых технологий и обмена опытом;

– студенты и абитуриенты получат доступ к образовательным материалам, что способствует профориентации и популяризации энергетических специальностей;

– инженеры и электрики могут находить полезные статьи и практические руководства;

– любой пользователь также может изучать контент про электроэнергетику.

Также необходимо выделить, какой основной функционал будет у программного обеспечения для образовательного блога об электроэнергетике. Для этого можно выделить следующий функционал:

– публикация стаей и учебных материалов. Администратор программного обеспечения образовательного блога об электроэнергетике сможет публиковать статьи и учебные материалы В общий доступ, что позволит пользователям ознакамливаться с данными учебными материалами;

– комментирование и оценка материалов. Пользователям будет дана возможность прокомментировать и оценить учебный материал, что позволит другим пользователям определить на качество данных учебных материалов.

– встроенный чат для общения. Пользователя будет дана возможность общаться с друг с другом с помощью встроенного чата;

Таким образом, разработка образовательного блога по электроэнергетике позволит решить эти проблемы, предоставив сотрудникам, студентам и всем заинтересованным пользователям удобный доступ к учебным материалам, возможность обмена опытом и повышения квалификации в дистанционном формате. Внедрение такой платформы не только оптимизирует процесс обучения внутри компании, но и будет способствовать привлечению молодежи в энергетическую отрасль.

Учет организационной структуры «Оренбургэнерго» и потребностей целевой аудитории при разработке блога обеспечит его эффективное внедрение и востребованность среди пользователей. Таким образом, создание цифрового образовательного ресурса станет важным шагом в модернизации системы подготовки кадров и популяризации профессии электрика в регионе.

2 Техническое задание

Настоящее Техническое задание определяет требования и порядок разработки системы по размещению и оценки блогов об электроэнергетике.

Заказчик: ГАПОУ «ОКЭИ».

Исполнитель: Скавренюк Никита Николаевич.

Начало работ: 21.03.2025.

Окончание работ: 17.04.2025.

Цель проекта: разработка системы по размещению и оценки блогов об электроэнергетике, аналог РН–ЭНЕРГО.

Назначение сайта: сайт для блога об электроэнергетике предназначен для публикации постов с информацией о работе электроэнергии, заводов, поставляющих электричество.

Цель сайта: Сервис должен предоставлять пользователям возможность чтения постов, оценку постов, а так же их комментирование.

Целевая аудитория ссайт для блога об электроэнергетике включает в себя простых пользователей, которые хотят изучить электроэнергетику.

Требования к структуре и функционированию сайта: сайт должен состоять из взаимосвязанных разделов с четко разделенными функциями.

Требования к персоналу: в большинстве случаев, использование сервиса не требует специфических знаний или навыков, однако для более эффективного использования всех возможностей сервиса рекомендуется ознакомиться с пользовательским интерфейсом и функционалом.

Требования к сохранности информации: для сохранности информации, предоставляемой пользователями, сайт для блога об электроэнергетике должен обеспечивать безопасное хранение данных на серверах с использованием современных методов шифрования. Кроме того, должна быть обеспечена возможность контроля доступа к информации на основе ролей и прав пользователей.

Требования к разделению пользователей: для того, чтобы обычные пользователи не смогли писать посты, а также сделать так, чтобы только зарегистрированные пользователи могли оставлять комментарии и оценивать блок, нужно разделить пользователей на классы. Всего есть 3 категории: гость, зарегистрированный пользователь и администратор.

Структура сайта:

– страница авторизации;

– страница регистрации;

– главная страница: на главной странице сайта отображается актуальная информация о блогах и обновлений;

– блоги: на странице блогов отображается все доступные блоги;

– личный кабинет клиента: в личном кабинете отображается информация о пользователе.

Блоги являются основным элементом сайта для блога об электроэнергетике, где представлены в виде карточек. Каждый блог имеет название, описание и фото представление о блоге, а так же дата публикации.

Карточки представляют собой описание, как выполнить тот или иной проект. Каждая карточка содержит название, дату создания, описание и фотографию самоделки.

Навигация:

– в левом верхнем углу находятся кнопки «главная», «блоги», «ЧАВо» и «личный кабинет». Кнопка «главная» позволяет перейти на главную страницу. При нажатии на кнопку «блоги» открывается страница, где пользователю предоставляется возможность просмотреть доступные блоги. Кнопка «ЧАВо» направит пользователя на блок с вопросами и ответами на них. А кнопка «личный кабинет» позволит пользователю зарегистрироваться в системе или просмотреть информацию об аккаунте;

– на главной странице отображаются актуальная информация, такая как новости об обновлениях сайта, популярные блоги;

– внутри блога обычный пользователь может просматривать инфтрукции, как сделать ту или иную самоделку, оставлять комментарий и оценивать с помощью звездного рейтинга;

Наполнение сайта (контент). Сайт для блога об электроэнергетике не предъявляет специфических требований к наполнению сайта, однако для обеспечения удобного и понятного интерфейса, рекомендуется использовать следующую структуру информации (для блога):

– название;

– описание;

– дата публикации.

Для наполнения блога:

– название;

– описание;

– дата создания;

– текст блога;

– оценочный блок.

Страницы всех разделов сайта должны формироваться программным путем на основании информации из базы данных на сервере. Модификация содержимого разделов должна осуществляться посредством администраторского веб–интерфейса, который без применения специальных навыков программирования (без использования программирования и специального кодирования или форматирования) должен предусматривать возможность редактирования информационного содержимого страниц сайта. Наполнение информацией должно проводиться с использованием шаблонов страниц сайта.

Типовыми элементами являются блоги.

При создании типовых элементов заполняются следующие поля для блога: название, описание, фото.

Функциональные возможности разделов:

– регистрация. Возможность регистрации для новых пользователей;

– авторизация. Возможность авторизации через существующий аккаунт;

– главная страница. озможность просмотра рекомендуемых и популярных блогов, актуальной информации.

Блоги:

– возможность просмотра блогов;

– возможность комментирования блогов;

– возможность оценивания блогов.

В верхнем левом углу каждого открытого блога должна быть кнопка «назад» синего цвета, предназначенная для возвращения на страницу выбора блога.

Дизайн и визуальное оформление:

– цветовая гамма: использовать преимущественно синюю и зеленую цветовую гамму, при этом цвета должны быть умеренно–контрастными, но спокойными, не вызывающими раздражения или усталости глаз;

– скругленные углы: все элементы интерфейса должны иметь скругленные углы, чтобы придать дизайну мягкость и сгладить острые углы;

– шрифт: для заголовков и основного текста использовать шрифт Montserrat, который обеспечивает хороший контраст и читаемость текста;

– контрастность: необходимо соблюдать баланс между контрастом текста и фона, чтобы избежать проблем с читаемостью и нагрузкой на зрение пользователей;

– адаптивность: дизайн должен быть адаптивным для корректного отображения на различных устройствах и разрешениях экрана.;

Требования к производительности и скорости работы:

– скорость загрузки сайта должна быть быстрой на всех устройствах;

– время отклика сайта должно быть минимальным;

– сайт должен быть оптимизирован под поисковые системы;

– должна быть реализована система кэширования;

– должны быть предусмотрены меры по защите от DDoS атак;

– должна быть возможность масштабирования сайта.

Требования к безопасности сайта:

– шифрование данных пользователей;

– ограничение доступа к конфиденциальной информации;

– регулярное обновление программного обеспечения и устранение уязвимостей;

– защита от атак на уязвимости программного обеспечения;

– мониторинг и блокировка подозрительной активности;

– обучение пользователей по безопасности.

База данных должна состоять из следующих сущностей:

– users (пользователи);

– admin (администратор);

– blogs (блоги);

– comments (комментарии);

– forum_questions (вопросы);

– forum_answers (ответы);

– raiting (рейтинг).

Сущность «users» должна состоять из следующих атрибутов:

– id;

– email;

– username;

– password.

Сущность «blogs» должна состоять из следующих атрибутов:

– id;

– title;

– description;

– content;

– created_at.

Сущность «admin» должна состоять из следующих атрибутов:

– id;

– email;

– adminUsername;

– adminSecretKey;

– adminPassword.

Сущность «comments» должна состоять из следующих атрибутов:

– id;

– blog_id;

– user_id;

– comment.

Сущность «raiting» должна состоять из следующих атрибутов:

– id;

– blog_id;

– user_id;

– raiting.

Сущность «forum_questions» должна состоять из следующих атрибутов:

– id;

– title;

– description;

– tags;

– user_id.

Сущность «forum_answers» должна состоять из следующих атрибутов:

– id;

– question_id;

– user_id;

– answer.

Разработка архитектуры системы и дизайна пользовательского интерфейса:

– анализ требований и определение основных сценариев использования системы;

– проектирование архитектуры системы, определение основных компонентов и их взаимодействия

– разработка прототипов пользовательского интерфейса с использованием инструментов прототипирования (Figma);

– создание дизайн–системы для обеспечения единообразия элементов интерфейса и удобства использования.

Создание базы данных для хранения информации о постах, комментариев и пользователях:

Выбор подходящей системы управления базами данных (SQLite);

– определение структуры базы данных, создание таблиц и определение связей между ними;

– написание кода для работы с базой данных на языке программирования (JavaScript);

– обеспечение безопасности данных, включая шифрование информации и ограничение доступа к данным.

Создание функциональных модулей, обеспечивающих основные функции управления постами:

– создание постов: возможность добавлять название, описание, изображения и ссылки на внешние источники;

– редактирование постов: добавление и удаление;

– настройки: изменение данных пользователя.

Тестирование системы:

– проведение функционального тестирования для проверки корректности работы системы;

– проведение нагрузочного тестирования для определения производительности системы при больших нагрузках;

– тестирование безопасности для выявления уязвимостей и возможности взлома системы.

Подготовка документации:

– подготовка пользовательской документации, описывающей основные функции и возможности системы;

– подготовка административной документации для управления системой и настройки прав доступа;

– создание обучающих материалов для быстрого освоения системы пользователями.

После завершения разработки сайта для блога об электроэнергетике, необходимо провести контроль и приемку сайта. Для этого необходимо выполнить следующие шаги:

– разработать тестовые сценарии: составить список действий, которые должны быть выполнены на сайте для проверки его работоспособности и соответствия требованиям;

– провести функциональное тестирование: выполнить тестовые сценарии и проверить корректность работы всех функций сайта, а также соответствие дизайна и оформления требованиям;

– выполнить нагрузочное тестирование: проверить работу сайта при большой нагрузке, чтобы убедиться в его стабильности и производительности;

– протестировать безопасность: проверить сайт на уязвимости и возможность взлома, а также убедиться в надежности системы защиты информации;

– подготовить отчет о тестировании: обобщить результаты тестирования и составить отчет с описанием всех обнаруженных проблем и несоответствий требованиям;

– провести приемку сайта: на основе отчета о тестировании принять решение о готовности сайта к эксплуатации, устранить выявленные проблемы и подготовить сайт к запуску;

– запустить сайт: после успешного завершения контроля и приемки, осуществить запуск сайта в эксплуатацию, начать работу с пользователями и продолжить развитие и поддержку системы.

3 Технический проект

3.1 Документация функциональной части

3.1.1 Описание постановки задачи

Цель этой производственной практике заключается в исследовании и разработке современной информационной системы, предназначенной для ведения блога, посвящённого тематике электроэнергетики. Основное внимание уделяется созданию функциональной, удобной в использовании и надёжной системы, способной удовлетворить запросы и ожидания целевой аудитории. Важной частью проекта является обеспечение постоянной и эффективной обратной связи с пользователями. Это позволит не только адаптировать интерфейс и функционал под реальные потребности пользователей, но и существенно повысит востребованность и популярность программного продукта.

Ключевая задача при разработке программного обеспечения для такого блога состоит в создании интуитивно понятного и удобного интерфейса, который позволит пользователям легко ориентироваться в материалах, публикуемых на платформе. Также необходимо упростить доступ к информации об электроэнергетике как для специалистов, так и для широкой аудитории, интересующейся данной сферой. Для этого проводится анализ как традиционных, так и современных методов подачи информации, оцениваются их сильные и слабые стороны, а также осуществляется их интеграция в итоговую систему с последующим тестированием.

Кроме того, значительная часть работы посвящена проектированию и реализации визуальной составляющей сайта. Это включает в себя анализ и подбор дизайна, верстку страниц, создание и настройку базы данных, а также реализацию серверной логики. Неотъемлемым компонентом является внедрение системы взаимодействия с пользователями – такой функционал, как форма обратной связи, возможность комментирования и оценки материалов. Это не только улучшает пользовательский опыт, но и даёт разработчику ценные сведения для последующей оптимизации системы.

В целом, данный курсовой проект ориентирован на внедрение передовых решений, направленных на повышение эффективности работы блога в сфере электроэнергетики. Реализация такого ресурса позволит сократить издержки, оптимизировать информационные потоки и внести вклад в развитие цифровых технологий в энергетической отрасли. Проект также нацелен на создание устойчивой и масштабируемой платформы, способной адаптироваться под изменяющиеся условия и запросы пользователей.

Обработка информации в системе представляет собой многоэтапный процесс, включающий взаимодействие пользователя с интерфейсом, обработку запросов сервером, хранение и обновление данных в базе, а также динамическое отображение изменений на странице. Процесс обработки информации в системе выглядит следующим образом:

– просмотр списка блогов. Когда пользователь заходит на сайт, система отправляет запрос к базе данных для получения списка доступных блогов. Данные могут включать заголовки и краткое описание, миниатюры изображений, рейтинги и количество комментариев. Сервер обрабатывает запрос, после чего формирует HTML–страницу с динамически подгружаемыми данными;

– переход на страницу блога. При клике на конкретный блог система отправляет запрос к БД, чтобы получить полный контент, загружает связанные данные: комментарии, оценки, информацию об авторе;

– добавление комментариев и оценок. Если пользователь авторизован, он может оставить комментарий, после чего сохраняется в БД с привязкой к ID блога и пользователя, поставить оценку, значение записывается в таблицу ratings, где также фиксируются ID пользователя и оценка, которую он выставил;

– расчет и отображение рейтинга. Система регулярно пересчитывает средний рейтинг блога. Агрегация данных – все оценки по конкретному блогу суммируются и делятся на их количество.

Таким образом, система обеспечивает интерактивность, персонификацию и актуальность данных за счет комплексной обработки информации на каждом этапе взаимодействия пользователя.

Обработка данных в системе строится на четком разделени входной информации, что поступает в систему и выходной информации, что система возвращает пользователю. Рассмотрим каждый аспект подробно.

Входная информация – это данные, которые система получает от пользователя или извлекает из базы данных для дальнейшей обработки. Рассмотрим, какаие данные входят в входную иформацию:

– данные, введённые пользователем. Пользователь взаимодействует с системой через формы, передавая следующие данные: логин для идентификации, пароль;

– комментарии. Текст комментария, ID блога, чтобы привязать комментарий к правильной записи;

– оценки. Оценка сохраняется в связанной таблице ratings, ID пользователя, для предотвращения повторного голосования.

Справочная информация из базы данных. Система использует дополнительные данные для корректной работы:

– список блогов;

– заголовки, краткие описания;

– предыдущие оценки и комментарии.

Идентификаторы для связи данных. Чтобы данные не терялись, система использует ключи:

– ID блога – связывает контент, комментарии и оценки.

– ID пользователя – определяет автора комментария или оценку.

– сессионные токены – для поддержания авторизации.

Выходная информация – это результат обработки входных данных, который видит пользователь или сохраняется в системе.

При открытии страницы со всеми блогами, пользователь получает следующие данные:

– название;
– краткое описание (например, первые 200 символов текста).
При открытии конкретной записи, пользователь увидит следующие данные:
– заголовок и основной контент;
– блок оценки статьи и средний рейтинг;
– список комментариев.
Каждый оставленный комментарий может включать в себя:
– автора;
– текст комментария;
– форма добавления комментария.

3.1.2 Описание функций

Для того, чтобы пользователь мог взаимодействовать с информационной системой для блогов об электроэнергетике есть различные функции.

Модуль авторизации и регистрации пользователя, позволяет клиенту зарегистрироваться и войти на сайт. Это сделано для того, чтобы, только авторизированные пользователе смогли оставлять комментарии и делиться отзывом.

Входная информация может включать в себя:

– логин и пароль (для регистрации и входа);

– проверка пользователя из базы данных (существует ли такой пользователь или нет).

Процесс выполнения задачи:

– пользователь вводит данные через форму;

– приложение проверяет соответствие данных в базе;

– в случае успеха создаётся сессия пользователя.

Результат выполнения работы может включать в себя уведомления об успешном входе или регистрации, переход в личный кабинет.

Модуль списков блогов включает в себя, список всех доступных блогов, с которыми пользователь может ознакомиться.

Процесс выполнения задачи выглядит следующим образом, система запрашивает данные из базы данных для отображения всех существующих блогов. Затем, информационная система отображает все доступные блоги на отдельной странице.

При успешной получения данных о всех блогов сайт отображает их с примененными стилями, названием блога, описанием.

Входная информация включает в себя основное наполнение блога, такое как название и описание блога.

Модуль взаимодействие с блогом включает себя страницу открытого блога который открыл пользователь. Он включает себя название блога, описание, сам текст блога, блок для оценки статьи и блок для оставления отзыва в виде комментария.

Входная информация включает в себя ID открытого блога, комментарии и оценка блога.

Процесс выполнения задачи выглядит следующим образом:

– система получает запрос на открытие определенного блога с помощью его ID;

– система загружает наполнение блога, а также форму для оставления комментария и блок для оценки статьи.

После выполнения работы пользователь получает полную статью с названием, описанием и самим содержимым блога, форму для написания отзыва об этой статье и блок для оценки с помощью звездного рейтинга.

Модуль комментирования блога позволяет клиенту написать отзыв об этой статье.

Входная информация включает в себя ID пользователя и текст комментария, который оставил клиент.

Процесс выполнения задач выглядит следующим образом авторизованный пользователь вводит текст комментариев специальную форму, которая находится после блога. Затем, система записывает в базу данных ID пользователя, который оставил данный отзыв и текст комментария.

Результат работы выглядит следующим образом: новый комментарий отображается на странице блога, пользователь получает уведомление о добавлении комментария.

Модуль оценивания блогов даст пользователю возможность поставить свою оценку блогу.

Результат работы:

– обновление средней оценки;

– уведомление о сохранении оценки.

Входная информация:

– ID пользователя и блога;

– оценка блога.

Процесс выполнения:

– проверка существования оценки от пользователя;

– обновление или вставка новой оценки в базу;

– перерасчёт средней оценки.

Модуль управления блогами доступен только администратору. Позволяет добавлять и удалять блоги, а также автоматически создавать базы данных для комментариев и оценок.

Результат работы выглядит следующим образом после добавления нового блога, созданный блог отображается в общем списке

Входная информация включает в себя данные блога: заголовок, текст, описание.

Процесс выполнения:

– администратор вводит данные блога;

– система добавляет его в основную базу;

– создаёт новую БД для комментариев и оценок.

3.2 Документация обеспечивающей части

3.2.1 Информационное обеспечение

Информационное обеспечение включает в себя описание всех данных, участвующих в процессе функционирования веб–приложения, включая входную и выходную информацию, принципы построения и структуру базы данных, а также макеты и структуру выходных документов.

Входная информация – это данные, которые пользователь вводит в веб–приложение через соответствующие формы. К ним относятся:

– форма регистрации. Чтобы пользователь зарегистрироваться на сайте, ему необходимо ввести логин и пароль;

– фора авторизации. Для входа в личный кабинет, пользователю необходимо ввести свой логин и пароль

– форма комментария и оценка блога. Если пользователь захочет опубликовать отзыв и оставить оценку для конкретного блога, он может это сделать, введя текст комментария в специальную форму, а так же оценить блог с помощью блока звездного рейтинга;

– форма добавления блога. Даная форма доступна только администраторам веб–приложения. С помощью этой формы, администраторы смогут опубликовать статью, введя название статьи, описание и само содержание темы.

Для хранения информации используется реляционная база данных SQLite 3, обеспечивающая простую и удобную работу с данными без необходимости отдельного сервера. Структура базы данных построена по принципу логического разделения сущностей и их связей.

Всего в информационной системе для блога об электроэнергетике содержит 4 таблицы:

– users (пользователи);

– blogs (блоги);

– comments (комментарии);

– raitings (рейтинг).

Таблица users представляет собой небольшую таблицу, состоящую из 3 колонок, где код пользователя создается автоматически, а логин и пароль вносится при регистрации нового пользователя. Структура Users представлена в таблице 1.

Таблица 1 – Структура таблицы «Users»

Идентификатор	Наименование поля	Тип данных	Размер
1	2	3	4
Код пользователя	id	Числовой	Целое число
Логин	username	Текстовый	100
Пароль	password	Текстовый	20
Почта	email	Текстовый	100

Таблица блок имеет 5 колонок, где код блога создается автоматически, а название, описание и контент добавляется администратором. Дата создания блога также фиксируется автоматически. Структура blogs представлена в таблице 2.

Таблица 2 – Структура таблицы «Blogs»

Идентификатор	Наименование поля	Тип данных	Размер
1	2	3	4
Код блога	id	Числовой	Целое число
Название	title	Текстовый	100
Описание	description	Текстовый	200
Контент	content	Текстовый	Большой текст
Дата создания	created_at	Дата	Дата

Таблица комментариев имеет 3 колонки, где код комментария создается автоматически, а код пользователя добавляется в таблицу вместе с комментарием самого пользователя. Структура comments представлена в таблице 3.

Таблица 3 – Структура таблицы «Comments»

Идентификатор	Наименование поля	Тип данных	Размер
1	2	3	4
Код комментария	id	Числовой	Целое число
Код пользователя	user_id	Числовой	Целое число
Комментарий	comment	Текстовый	500

Таблица комментариев имеет 3 колонки, где код комментария создается автоматически, а код пользователя добавляется в таблицу вместе с комментарием самого пользователя. Структура comments представлена в таблице 4.

Таблица 4 – Структура таблицы «Rating»

Идентификатор	Наименование поля	Тип данных	Размер
1	2	3	4
Код комментария	id	Числовой	Целое число
Код пользователя	user_id	Числовой	Целое число
Отзыв	Raiting	Числовой	5

Таблица администраторов имеет 5 колонок, где секретный ключ для входа создается автоматически, а данные администратора, такие как почта, пароль и логин, добавляются за счет регистрации администратора. Структура admins представлена в таблице 5.

Таблица 5 – Структура таблицы «admins»

Идентификатор	Наименование поля	Тип данных	Размер
1	2	3	4

Продолжение таблицы 5

Код адмиина	id	Числовой	Целое число
Логин админа	username	Текстовый	100
Почта	email	Текстовый	100
Пароль	password	Текстовый	20
Секретный ключ	secret_key	Числовой	6

Таблица для вопросов имеет 5 колонок, где код вопроса создается автоматически, а название, описание и теги добавляется пользователем. Структура forum_questions представлена в таблице 6.

Таблица 6 – Структура таблицы «forum_questions»

Идентификатор	Наименование поля	Тип данных	Размер
1	2	3	4
Код вопроса	id	Числовой	Целое число
Название	title	Текстовый	100
Описание	description	Текстовый	500
Теги	tags	Текстовый	Небольшой текст
Код пользователя	user_id	Числовой	Целое число

Таблица для ответов имеет 4 колонки, где код ответа создается автоматически, а сам ответ добавляется пользователем. Структура forum_answers представлена в таблице 7.

Таблица 7 – Структура таблицы «forum_answers»

Идентификатор	Наименование поля	Тип данных	Размер
1	2	3	4
Код ответа	id	Числовой	Целое число
Код вопроса	question_id	Числовой	Целое число
Код пользователя	user_id	Числовой	Целое число
Ответ	answer	Текстовый	500

Обработка информации в веб–приложении осуществляется в несколько этапов. Пользователь вводит данные в форму (например, комментарий и оценку блога). На стороне клиента срабатывает базовая валидация. После отправки форма передаёт данные на сервер. Сервер получает данные, проверяет сессию пользователя. Выполняется повторная валидация и приведение типов. Если это оценка: проверяется, оставлял ли пользователь ранее оценку, если да – обновляется; если нет – создаётся новая запись. Если это комментарий – создаётся новая запись в таблице comments. Данные записываются в базу с помощью SQL–запросов. В случае успеха возвращается статус 200 и обновляется интерфейс. При загрузке страницы блога из базы данных извлекаются комментарии и оценки. Формируется HTML и отправляется пользователю.

К выходной информации относятся:

– личный кабинет – имя пользователя, дата регистрации;
– список блогов – заголовки, краткое описание, средняя оценка;
– страница блога – текст, комментарии.

3.2.2 Техническое обеспечение

Техническое обеспечение разрабатываемого веб–приложения «Образовательный блог» включает в себя подбор оптимальной серверной инфраструктуры, среды хостинга, средств сетевого взаимодействия и пользовательского оборудования, обеспечивающего корректную и стабильную работу системы.

Хостинг и серверная инфраструктура

Для размещения веб–приложения выбран виртуальный выделенный сервер, предоставляющий возможность гибкой настройки серверной части, масштабирования проекта и высокой отказоустойчивости. Это особенно важно с учетом того, что приложение использует модульную архитектуру на базе Node.js и может требовать отдельного управления зависимостями и окружением. Для этот можно использовать Render.

Выбранная конфигурация VPS:
– процессор: 2 виртуальных ядра (vCPU, Intel Xeon);
– оперативная память (RAM): 4–8 ГБ;
– дисковое пространство: 60 ГБ SSD NVMe;
– сетевое подключение: 100 Мбит/с с возможностью расширения;
– серверное ПО: Node.js, SQLite.

Обоснование выбора VPS:
– полный контроль над окружением;
– возможность установки дополнительных библиотек и зависимостей через npm;
– поддержка работы в режиме backend + frontend;
– надежность и масштабируемость (в случае роста проекта).

Разработка проекта выполняется на пользовательском компьютере, соответствующем следующим требованиям:
– процессор: Intel Core i5 / AMD Ryzen 5;
– оперативная память: не менее 8 ГБ;
– диск: SSD 256 ГБ;
– рперационная система: Windows 10/11;

Программное обеспечение:
– среда разработки – Visual Studio Code;
– менеджер пакетов – npm / yarn;
– веб–браузеры – Google Chrome, Firefox;
– тестирование – Postman.

Использование локальной среды позволяет производить быстрое тестирование и отладку, без риска нарушения стабильности размещенной версии.

Все клиентские запросы обрабатываются сервером, развернутым на VPS. Обмен данными между клиентом и сервером осуществляется по защищённому протоколу HTTPS. Сервер также взаимодействует с базой данных, хранящей информацию о пользователях, комментариях, оценках и контенте блога.

Для защиты приложения от несанкционированного доступа и обеспечения целостности данных используются:

– брандмауэр на стороне сервера;

– защита от DDoS–атак (через средства провайдера или Cloudflare);

– регулярные обновления Node.js и зависимостей из node_modules.

Контроль за стабильностью и доступностью приложения осуществляется с помощью:

– логирования;

– внешнего мониторинга;

– ручного и автоматического тестирования.

3.2.3 Программное обеспечение

Разработка программного обеспечения для блога об электроэнергетике–это сложный и многогранный процесс, включающий проектирование и реализацию программных компонентов, обеспечивающих удобное взаимодействие пользователей с сайтом, управление блогами и процессами. Это приложение должно учитывать разнообразные потребности пользователей, такие как поиск и просмотр блогов, получение рекомендаций, участие в различных активностях и многое другое.

Во время разработки информационной системы для блога об электроэнергетике необходимо провести анализ и проектирование системы, определить основные функциональные возможности, учесть специфические требования и потребности пользователей.

Помимо этого, важно продумать удобную и интуитивно понятную навигацию по сайту. Пользователи должны легко находить интересующие их разделы, переходить между статьями.

Кроме того, интеграция с социальными сетями может способствовать увеличению аудитории и улучшению взаимодействия пользователей. Возможность делиться публикациями, авторизоваться через популярные сервисы и комментировать с использованием существующих аккаунтов упростит процесс вовлечения новых участников. Это также может положительно сказаться на популярности ресурса за счет органического распространения контента.

Также необходимо уделить внимание вопросам безопасности и защиты данных. Поскольку информационная система для блога ою электроэнергетике хранит данные о пользователях, их комментариях и других персонализированных настройках, важно обеспечить защиту от возможных атак злоумышленников. Безопасность включает защиту пользовательских аккаунтов, предотвращение утечек данных и защиту от спама и вредоносных действий в комментариях.

Тестирование является важной частью процесса разработки программного обеспечения. Тщательное тестирование помогает выявить ошибки, уязвимости и недоработки, что гарантирует стабильную и безопасную работу системы. Это также улучшает пользовательский опыт, делая сайт более удобным и плавным в использовании.

После запуска системы необходимо обеспечить ее поддержку и регулярное обновление. Это включает оптимизацию сайта, исправление багов, добавление новых функций и улучшение производительности. Регулярные обновления позволят платформе оставаться актуальной, привлекать новых пользователей и удерживать аудиторию.

Технологический стек разработки

Для создания удобного и визуально привлекательного интерфейса блога о DIY необходимо использовать современные веб–технологии. К ним относятся:

– HTML, CSS, JavaScript – для верстки и стилизации страниц, а также для обеспечения динамических элементов интерфейса;

– адаптивная верстка – обеспечение корректного отображения сайта на мобильных устройствах, планшетах и компьютерах.

Бэкенд–часть системы отвечает за обработку пользовательских запросов, управление данными и обеспечение логики работы блога. Для этой части можно использовать:

– Node.js – для построения серверной части приложения. Node.js обеспечивает возможность разработки приложений с использованием JavaScript. Совместимость с JavaScript делает процесс разработки более удобным и доступным, позволяя разработчикам использовать в новых проектах уже сформированные навыки и опыт работы;

– SQLite – для хранения данных о пользователях, блогах, комментариях и рейтингах. Благодаря особенностям архитектуры SQLite работает быстро, особенно на чтение. Компоненты СУБД встроены в приложение и вызываются в том же процессе. Поэтому доступ к ним быстрее, чем при взаимодействии между разными процессами. Так же база данных состоит из табличных записей, связей между ними, индексов и других компонентов. В SQLite они хранятся в едином файле, который находится на том же устройстве, что и программа. Чтобы при работе не возникало ошибок, файл блокируется для сторонних процессов перед записью. Раньше это приводило к тому, что записывать данные в базу мог только один процесс единовременно. Но в новых версиях это решается перенастройкой режима работы СУБД.

Тестирование и мониторинг

Существует несколько видов тестирования, которые помогут поддерживать сайт в стабильном состоянии:

– ручное тестирование – проверка функциональности вручную перед обновлениями;

– автоматизированное тестирование – тестирование UI, комментариев и авторизации;

– нагрузочное тестирование – проверка работы сайта при высокой посещаемости.

Для мониторинга производительности и сбора аналитики можно использовать:

– Google Analytics – для анализа поведения пользователей;

– Prometheus, Datadog – для отслеживания работы сервера и базы данных.

Взаимодействие с пользователями

Эффективное взаимодействие с пользователями играет ключевую роль в развитии блога. Для этого можно внедрить:

– систему комментариев с возможностью оставлять отзывы и обсуждать проекты;

– форум или сообщество для обмена идеями и советами;

– интеграцию с социальными сетями, чтобы пользователи могли делиться своими работами.

Разработка и поддержка блога об электроэнергетике – это непрерывный процесс, включающий в себя проектирование, разработку, тестирование, запуск и дальнейшее улучшение платформы. Важно не только создать удобный и безопасный сайт, но и поддерживать активное сообщество пользователей, добавлять новые функции и следить за тенденциями в образовательной сфере.

4 Рабочий проект

4.1 Разработка веб–приложения

Создание веб–приложения – это комплексный процесс, включающий проектирование пользовательского интерфейса, разработку функциональных страниц и настройку механизмов динамического взаимодействия с данными.

Разработка веб–приложения включает реализацию визуального дизайна, структуры и логики страниц, а также настройку механизмов динамического вывода контента. Веб–приложение предназначено для пользователей, желающих получать образовательную информацию, оставлять комментарии, а также оценивать записи блога. В этом разделе подробно описаны визуальные и функциональные компоненты проекта.

Веб–дизайн образовательного блога выполнен в соответствии с принципами современного UX/UI–дизайна: минимализм, интуитивность, высокая читаемость и адаптивность для разных устройств. Основные элементы дизайна были разработаны с использованием CSS–фреймворка Tailwind CSS и JavaScript–библиотек для интерактивности.

Основные элементы интерфейса:

– шапка сайта: логотип, название блога, кнопки навигации («Главная», «Блоги», «Регистрация», «Вход»/«Личный кабинет»);

– меню навигации: представлено в виде горизонтального списка; на мобильных устройствах преобразуется в бургер–меню;

– основная часть (контент): карточки блогов с кратким описанием, рейтингом, кнопкой перехода к полному просмотру;

– форма комментирования: реализована в виде всплывающей или встроенной формы с полем ввода и кнопкой отправки;

– подвал (footer): ссылки на политику конфиденциальности, контакты, социальные сети.

Цветовая палитра выбрана спокойная оттенки синего и белого, шрифты – читаемые, без засечек.

Веб–приложение состоит из следующих основных страниц:

Главная страница отображает приветствие, краткое описание сайта и переход к последним добавленным блогам. Основной функционал – просмотр последних записей, переход к регистрационной форме.

Список блогов представляет собой галерею блогов с кратким описанием, средней оценкой, кнопками «Читать», «Оценить». Динамически заполняется на основе данных из БД.

Страница конкретного блога отображает полное содержимое статьи, комментарии и форму добавления комментария. Динамически формируется на основе параметра ID.

Регистрация и вход:

– формы с валидацией: логин, пароль, повтор пароля.

– обработчики отправки и проверок реализованы в auth.js.

Личный кабинет отображает личную информацию. Форма взаимодействия с пользователем на каждой странице реализована через простые HTML–формы с обработкой через JavaScript.

Дизайн веб–приложения представлен в приложении Г.

Чтобы пользователь смог войти или зарегистрироваться на сайте блогов об электроэнергетике, для этот есть формы входа и регистрации. Что форма входа регистрации работали для этого используются запросы на маршруты /register и /login. Пользователь для регистрации вводит свои данные такие как логин и пароль. Система проверяет, если этот пользователь с таким же логином или нет. Если его нет, то система вносит в базу данных логин и пароль пользователя. После чего, пользователь сможет войти в свой личный кабинет.

Загрузка списка блогов на главной и в на стрaгнице блогов осуществляется с помощью запроса GET /api/blogs. Затем получаем ответ: массив объектов блогов, отображаемых с помощью шаблонов.

Загрузка содержимого конкретного блога осуществляется с помощью запроса: GET /api/blog/:id. Таким образом, получаем ответ: JSON с содержимым статьи, комментариями, рейтингом.

Пользователь может оставить свой отзыв на любом блоге с помощью запроса POST /api/comment. Затем передаются следующие необходимые данные в базу данных: ID блога, текст комментария, ID пользователя. После успешного добавления данных, на сайте появляется тот отзыв, который оставил пользователь.

Также, чтобы пользователь смог оценить блок с помощью звездного рейтинга, для этого есть запрос POST api/raitings. При отправке оценки система проверяет, есть ли эта оценка или нет и кто отправляет эту оценку. Если этой оценки нет и его отправляет новый пользователь, тогда он вносит эту оценку в базу данных.

4.2 Логика работы веб–приложения

Модули и функциональные файлы – это части кода, которые помогают структурировать веб–приложение и упростить его разработку, поддержку и масштабирование.

Модуль – это отдельный файл с кодом, в котором собраны функции, классы или переменные, относящиеся к одной конкретной задаче. Модуль можно подключить и использовать в других частях программы. Модули необходимы в разработке веб–приложения, так как они упрощают навигацию по проекту, облегчают поддержку и отладку, упрощают тестирование и позволяют переиспользовать код.

Веб–приложение блога об электроэнергетике включает в себя следующие модули:

– главная основная страница (index.html). Представляет из себя страницу с приветствием, описанием назначения веб–приложения и списком популярных на данный момент блогов;

– страница с блогами (bloglist.html). Эта страница представляет собой список с блогами в виде карточек с названием и описанием блога. С помощью этой страницы пользователь сможет ознакомиться со всеми блогами, которые ему доступны;

– блог (blog.html). Эта страница создается с помощью администратора. Она представляет собой страницу, на которой пользователь сможет ознакомиться с статьей, а так же оценить блог с помощью звездного рейтинга и/или написать отзыв с помощью формы ввода комментария;

– регистрация (register.html). Представляет собой страницу с формой для регистрации. С помощью этой формы, пользователь сможет зарегистрироваться;

– вход (login.html). Представляет собой страницу с формой для входа в личный кабинет. С помощью этой формы, пользователь сможет войти в свой аккаунт;

– личный кабинет (profile.html). Эта страница представляет собой небольшую страницу, где пользователь сможет выйти из своего аккаунта. Если пользователь не войдет в личный кабинет, тогда он не сможет оставлять комментарии и оценивать блоги с помощью звездного рейтинга;

– админ–панель (admin.html). Представляет собой страницу, где администратор сайта сможет добавлять новые блоги.

Также веб–приложения блога об электроэнергетике, имеет функциональные файлы и скрипты. Функциональные файлы и скрипты активно участвуют в работе веб–приложения. На сайте присутствуют следующие функциональные файлы и скрипты:

– серверная часть веб–приложения (server.js). Это один из важных функциональных файлов для работы веб–приложения. Он обрабатывает важные события, такие как создание базы данных, запуск сервера, вход и регистрация пользователя, добавление новых комментариев, создание новых блогов и так далее;

– регистрационная часть веб–приложения (reg.js). Этот файл связан с server.js и помогает для регистрации нового пользователя;

– вход (log.js). Этот файл также связан с файлом server.js. Этот файл помогает войти пользователю в личный кабинет;

– личный кабинет(account.js). Этот файл позволяет сохранять сессию, а также отображать имя пользователя в личном кабинете. Также, если пользователь вошел в личный кабинет, он сможет составлять комментарии и оценивать блоги с помощью звездного рейтинга;

– форма комментария (comments.js). Этот файл позволяет пользователю оставлять комментарий. Также этот файл позволяет отображать все добавленные комментарии в блоге;

– основа для блога (blog.js). Этот функциональный файл показывает системе, как должен выглядеть блок. В этом файле встроен макет, помощью которого система адаптируют блог, чтобы все шло по порядку;

– работа комментариев и звездного рейтинга (script.js). С помощью этого функционального файла система может выстроить работу системы комментариев

и звездного рейтинга. Без этого файла не будет работать система комментариев и звездного рейтинга соответственно;

– админ панель (admin.js). Это функциональный файл, позволяет admin.html создавать новые блоги, редактировать их и удалять;

Описание логики работы модуля регистрации выглядит следующим образом:

– пользователь вводит логин и пароль;

– скрипт reg.js проверяет корректность (валидация длины и формата);

– отправка POST–запроса на сервер /api/register;

– сервер проверяет уникальность логина, хэширует пароль;

– запись нового пользователя в таблицу users;

– ответ пользователю: успех – редирект на login.html.

Описание логики работы модуля входа выглядит следующим образом:

– пользователь вводит логин и пароль;

– отправка POST–запроса /api/login;

– сравнение логина и пароля в БД;

– В случае успеха – установка сессии/токена, редирект на account.html.

Описание логики работы модуля страницы входа выглядит следующим образом:

– при загрузке страницы по ID блога отправляется запрос к GET /api/blog/:id;

– отображается содержимое, комментарии и средняя оценка;

– пользователь может оставить комментарий с помощью POST–запроса к api/comment и оценить блог с помощью звездного рейтинга, POST–запроса api/raitnig;

– после отправки новый комментарий появляется в в блоке с комментариями, а средняя оценка обновляется.

Описание логики работы модуля личного кабинета выглядит следующим образом:

– при загрузке проверяется наличие активной сессии;

– получается информация о пользователе и его активности.

Описание логики работы модуля проверки сессии выглядит следующим образом:

– используется на страницах account.html, blog.html;

– проверяет, есть ли активная сессия;

– Если нет, пользователя перенаправляют на страницу входа.

Все скрипты изображены в приложении Д.

4.3 Руководство системному программисту

Информационная система для образовательного блога об электроэнергетике представляет собой сайт, который позволяет пользователям просматривать блоги, комментировать и оценивать их. Администраторам представлена возможность добавления блогов об электроэнергетике, изменения блогов, удаления, а так же

модерации комментариев. Она состоит из нескольких компонентов, таких как главная страница, страница с блогами, личный кабинет, выставление оценки и комментариев, а так же админ панель, где администраторы смогут работать с блогами и модерировать комментарии. Так же есть система, позволяющая добавлять и удалять администраторов.

Для создания удобного и визуально привлекательного интерфейса образовательного блога об электроэнергетике необходимо использовать современные веб–технологии. К ним относятся:

– HTML, CSS, JavaScript – для верстки и стилизации страниц, а также для обеспечения динамических элементов интерфейса;

– адаптивная верстка – обеспечение корректного отображения сайта на мобильных устройствах, планшетах и компьютерах.

Бэкенд–часть системы отвечает за обработку пользовательских запросов, управление данными и обеспечение логики работы блога. Для этой части можно использовать:

– Node.js – для построения серверной части приложения. Node.js обеспечивает возможность разработки приложений с использованием JavaScript. Совместимость с JavaScript делает процесс разработки более удобным и доступным, позволяя разработчикам использовать в новых проектах уже сформированные навыки и опыт работы;

– SQLite – для хранения данных о пользователях, блогах, комментариях и рейтингах. Благодаря особенностям архитектуры SQLite работает быстро, особенно на чтение. Компоненты СУБД встроены в приложение и вызываются в том же процессе. Поэтому доступ к ним быстрее, чем при взаимодействии между разными процессами. Так же база данных состоит из табличных записей, связей между ними, индексов и других компонентов. В SQLite они хранятся в едином файле, который находится на том же устройстве, что и программа. Чтобы при работе не возникало ошибок, файл блокируется для сторонних процессов перед записью. Раньше это приводило к тому, что записывать данные в базу мог только один процесс единовременно. Но в новых версиях это решается перенастройкой режима работы СУБД.

Информационная система для блога об электроэнергетике имеет свою большую структуру. Структура данного проекта включает в себя следующие элементы:

– server.js – основной серверный файл, который помогает запускать приложение;

– ratings.db – файл базы данных, который хранит в себе все необходимые данные об пользователях, администраторов, блогахи так далее;

– public – папка, которая хранит в себе html, css и js файлы, а так же изображения на сайте;

– node_modules – папка, которая хранит в себе все файлы и компоненты для работы node.js.

Для размещения веб–приложения выбран виртуальный выделенный сервер, предоставляющий возможность гибкой настройки серверной части, масштабирования проекта и высокой отказоустойчивости. Это особенно важно с

учетом того, что приложение использует модульную архитектуру на базе Node.js и может требовать отдельного управления зависимостями и окружением. Для этот можно использовать Render.

Выбранная конфигурация VPS:

– процессор: 2 виртуальных ядра (vCPU, Intel Xeon);

– оперативная память (RAM): 4–8 ГБ;

– дисковое пространство: 60 ГБ SSD NVMe;

– сетевое подключение: 100 Мбит/с с возможностью расширения;

– серверное ПО: Node.js, SQLite.

Разработка проекта выполняется на пользовательском компьютере, соответствующем следующим требованиям:

– процессор: Intel Core i5 / AMD Ryzen 5;

– оперативная память: не менее 8 ГБ;

– диск: SSD 256 ГБ;

– рперационная система: Windows 10/11;

Программное обеспечение:

– среда разработки – Visual Studio Code;

– менеджер пакетов – npm / yarn;

– веб–браузеры – Google Chrome, Firefox.

Для запуска сервера необходимы следующие этапы:

– проверить на установленные зависимости. Для это необходимо в командной строке пропистаь команду npm intsall;

– убедиться в наличии файла базы данных ratings.db. Если его нет – создать файл и выполнить начальную инициализацию таблиц;

– настроить почтовый сервис в server.js. Для этого нужно указать логин и пароль для nodemailer;

– запуск сервера. Для этого необходимо ввести команду node server.js;

– проверить доступность веб-приложения по адресу http://localhost:3000

Если решили переименовать файл server.js, то в этом случае, перед запуска сервера, введите команду node новое название файла.js.

Информационная система для блога об электроэнергетике и имеет базу данных под названием ratings.db. Таблица включает в себя следующие данные:

– admins (администратор) – данная таблица хранит данные об администраторах;

– users (пользователи) – данная таблица хранит данные о пользователях;

– blogs (блоги) – данная таблица хранит данные о блогах;

– comments и ratings (комментарии, звездный рейтинг) – данная таблица хранит данные о комментариях и звездном рейтинге.

Также при работе с сайтом у администратора могут возникнуть ошибки. Ошибки, причина и их решение указаны в таблице 1

Таблица 8 – ошибки, причины и их решения

Ошибка	Причина	Решение
Пароль слишком простой. Используйте	Пароль не соответствует	Придумать более сложный пароль,

заглавные, строчные буквы, цифры и спецсимволы.	требованиям (длина ≥ 8, есть заглавная, строчная буква, цифра и спецсимвол)	например: Admin@2024
"Неверные данные" при входе администратора	Неверный логин, пароль или секретный ключ	Убедиться, что логин, пароль и 6-значный ключ указаны верно (без пробелов). Проверить раскладку клавиатуры
"Ошибка отправки письма" при регистрации администратора	Неправильный email или проблемы со связью с почтовым сервером	Убедиться, что email указан верно. Убедиться, что пароль приложения действителен и указана верная почта отправителя (например: user: 'example@mail.com', pass: 'aaaa bbbb cccc dddd')

4.4 Руководство пользователя

Руководство пользователя – это документ, который предоставляет подробную информацию о том, как использовать определенное оборудование, программу или сайт. В руководстве предоставляются инструкции по установке, порядок выполнения последовательности действий по настройке и использованию оборудования или ИТ-продукта, а также описание функций, возможностей и рекомендации по их использованию.

Руководство пользователя помогает новым пользователям быстрее освоиться с программой или сайтом, избежать ошибок и максимально эффективно использовать их возможности. В случае опытных пользователей руководство пользователя необходимо для обеспечения подробной информации о всех возможностях и функциях программы. Оно помогает пользователям освоить более сложные, «продвинутые» функции, узнать о возможных проблемах и их решениях, а также повысить производительность и эффективность работы. Руководство пользователя также может включать в себя советы по эксплуатационной безопасности и рекомендации по оптимальным настройкам для получения наилучших результатов.

Информационная система образовательного блога об электроэнергетике представляет собой современное веб- приложение, созданное для обмена знаниями, повышения профессиональной компетенции и формирования

сообщества специалистов и энтузиастов отрасли. Архитектура сайта разделена на несколько логических модулей, каждый из которых отвечает за определённый набор функций и гарантирует гибкость дальнейшего развития.

В информационной системе для блога об электроэнергетики существует три пользователя:

– зарегистрированный пользователь. Этот тип пользователя может не только просматривать блоги, он может оставлять отзыв с помощью комментария, а также оценивать блок с помощью звездного рейтинга;

– гость. Этот тип пользователя может просматривать блоги, но не может оставлять комментарии и оценивать блоги с помощью звездного рейтинга;

– администратор. Этот тип пользователя может создавать новые блоги, изменять их, а так же удалять блоги.

Для объяснения работы пользователю информационной системе для блога о электроэнергии для этого необходимо расписать, как работает каждый элемент в данной системе.

Регистрация и вход.

Для начала пользователь может зарегистрироваться в веб-приложении. Регистрация позволит пользователю оставлять комментарии и оценивать блоки с помощью звездного рейтинга. Для регистрации необходимо перейти на страницу личного кабинета и нажать кнопку «Зарегистрироваться», после этого, откроется форма для регистрации. Форма регистрации изображена рисунке 1.

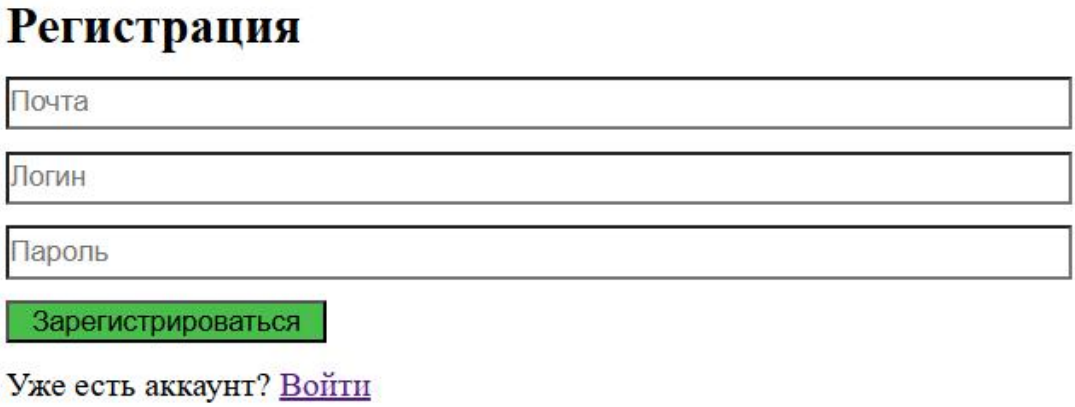


Рисунок 1 – Форма регистрации

После регистрации, пользователя перекинет в личный кабинет и сможет оставлять комментарии и оценивать блог с помощью звездного рейтинга. Личный кабинет изображен на рисунке 2

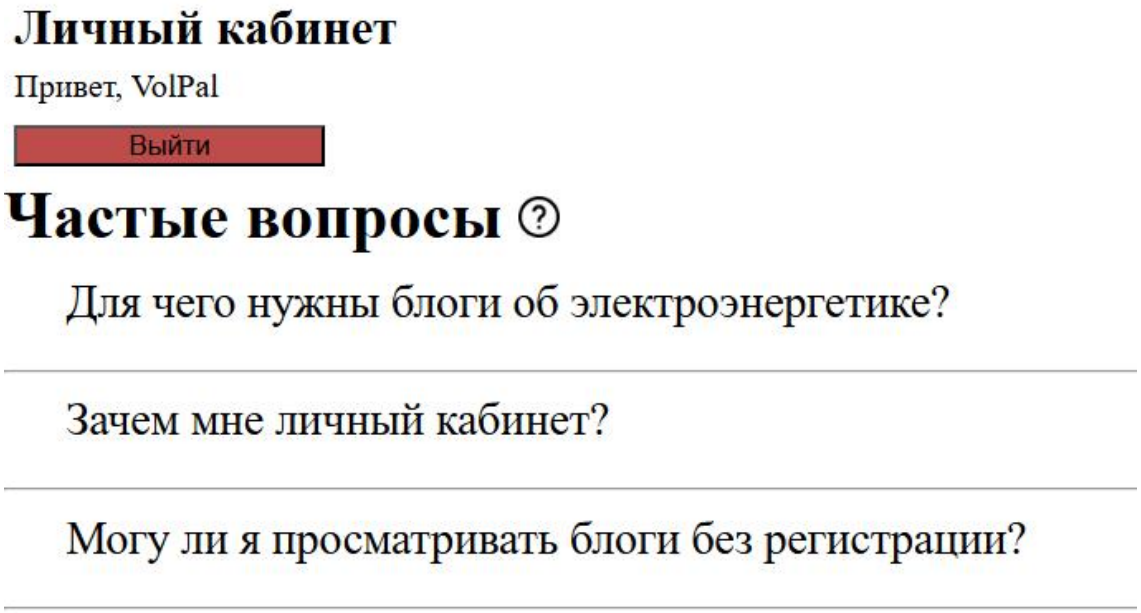


Рисунок 2 – Личный кабинет

Просмотр статей.

После регистрации пользователь сможет просмотреть все доступные блоги на главной странице, либо на странице «Блог». Все доступные статьи отображаются на главной странице или на странице «Блоги». Страница с блогами изображена на рисунке 3.

Рисунок 3 – Страница с блогами

При нажатии на «Читать полностью», откроется выбранный блог и пользователь сможет просмотреть блог. Ниже статьи отображаются комментарии пользователей и средняя оценка. Тестовый формат блога изображен на рисунке 4.

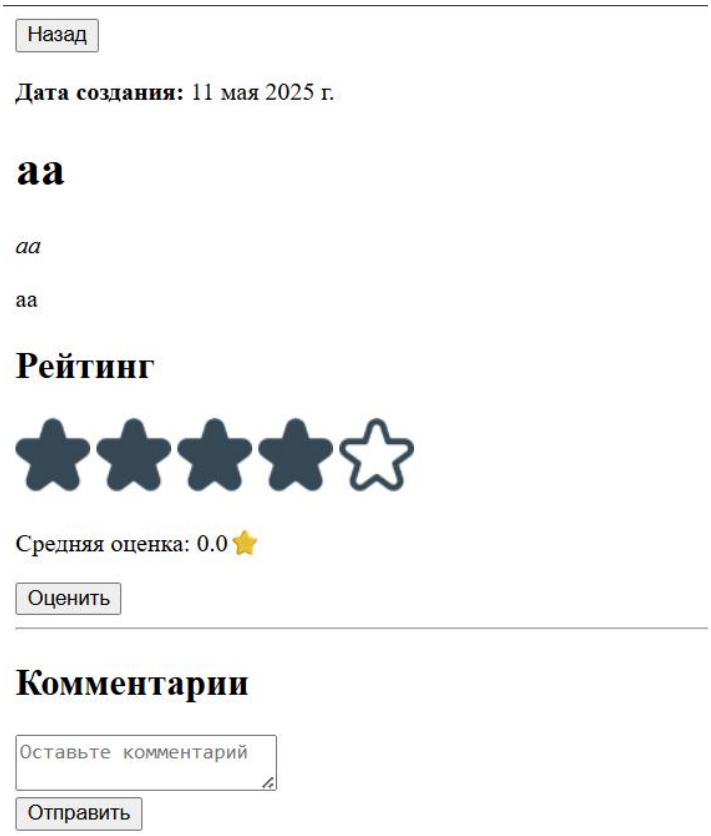


Рисунок 4 – Тестовый формат блога

Комментирование блога.

После прочтения статьи, пользователь сможет оставить свой отзыв о нем, написав комментарий специальное выведенном блоке. Чтобы оставить комментарий, пользователю необходимо авторизоваться. Внизу страницы с блогом находится форма ввода комментария. После ввода отзыва нажмите кнопку «Отправить». Комментарий появится сразу после отправки. Блок с комментарием изображен на рисунке 5.

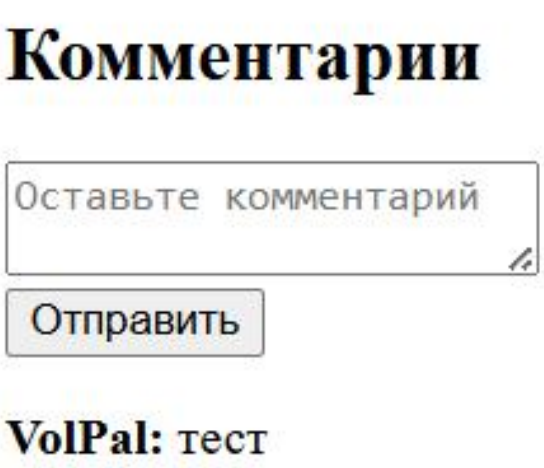


Рисунок 5 – Блок с комментариями

Оценка статей.

Пользователь может оставить оценку с помощью звездного рейтинга. Оценки выставляются в виде звёздочек от 1 до 5. Пользователь может оценить статью один раз, при этом он может изменить свою оценку позже. После

выставления оценки, необходимо нажать кнопку «Сохранить». После отправки оценки, средний рейтинг меняется. Блок с звездным рейтингом изображен на рисунке 6

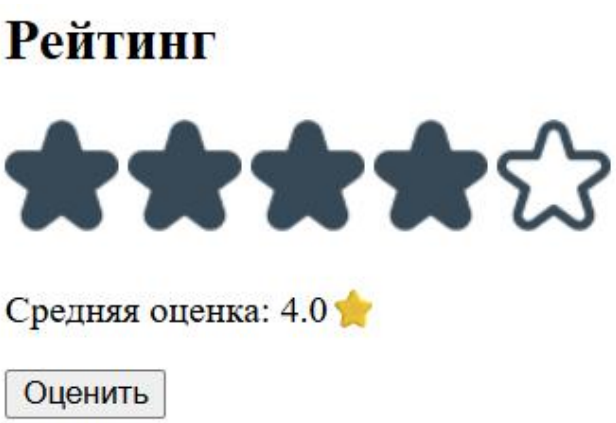


Рисунок 6 – Блок с звездным рейтингом

Ограничения и требования

Комментирование и оценка недоступны без регистрации. Не допускаются оскорбительные или спам-сообщения – они будут удаляться модератором. Добавление, редактирование и удаление блогов недоступны пользователю, это доступно только администратору.

Также при работе с сайтом у пользователя могут возникнуть ошибки. Ошибки, причина и их решение указаны в таблице 1

Таблица 9 – ошибки, причины и их решения

Ошибка	Причина	Решение
"Ошибка регистрации" при создании пользователя	Такой логин уже существует	Выбрать другое имя пользователя
"Неверные учетные данные" при входе обычного пользователя	Введены неправильные имя пользователя или пароль	Убедиться, что логин и пароль правильные. Зарегистрироваться, если ещё не создана учётная запись
Ошибка при добавлении комментария: Комментарий не может быть пустым	Поле ввода комментария остался не заполненым	Заполнить поле комментария перед отправкой

4.5 Ревьюирование программного кода

Ревьюрирование кода – практика, при которой разработчики смотрят и оценивают код, написанный другими. Это важная часть создания программного обеспечения, которая помогает повысить качество, улучшить читаемость и обнаружить потенциальные проблемы.

Основная цель процесса – обнаружение ошибок и уязвимостей, а также улучшение качества кода. Для эффективного код-ревью необходимо установить ясные критерии оценки, определить роли и ответственности разработчиков, а также использовать инструменты и системы управления версиями, которые упрощают процесс.

Существуют различные подходы:

– парное программирование. Оно предполагает работу двух разработчиков над одним и тем же кодом, что позволяет обнаруживать ошибки на ранней стадии и обмениваться знаниями. Коллеги сотрудничают в режиме реального времени – один пишет код (драйвер), а другой просматривает его (навигатор). Такой способ часто используют команды разработчиков, потому что товарищи по команде совместно находят наиболее эффективное решение проблемы;

– обзоры «из-за плеча». Два разработчика – автор и рецензент – сотрудничают лично или удаленно через общий экран, и автор объясняет и аргументирует выбранные решения. Рецензент задает вопросы и вносит предложения. Автор может внести небольшие изменения во время обзора, а более крупные исправления отложить на потом;

– инструментальные обзоры. Использование инструментов позволяет сократить время и обеспечить создание кода самого высокого качества. Рецензии с помощью инструментов могут автоматически собирать измененные файлы и вычленять различия, упрощать получение отзывов и обсуждение с помощью комментариев. Также они могут включать статическое тестирование безопасности приложений (SAST), чтобы помочь выявить и устранить уязвимости;

– электронная почта. Передача по электронной почте часто используется для решения незначительных проблем и проверки небольших фрагментов.

Для ревьюрирование кода я использовал инструментальный обзор, Потому что данный метод быстрее, удобнее и этот способ дает рекомендации по исправлению кода.

Первая проблема – повторное использование middleware-ов. Код представлен на рисунке 7

```
app.use(session({ secret: 'adminsecret', resave: false, saveUninitialized: true }));

app.use(session({
    secret: 'secret-key',
    resave: false,
    saveUninitialized: true
}));
```

Рисунок 7 – повторное использование middleware-ов

Решение данной проблемы заключается в сокращении кода в один middleware. Исправленный код изображен на рисунке 8.

```
app.use(session({
  secret: 'adminsecret secret-key',
  resave: false,
  saveUninitialized: true
}));
```

Рисунок 8 – исправленный middleware

Вторая проблема заключается в том, то, что мы использовали app.use(bodyParser.json()); два раза. Проблема изображена на рисунке 9.

```
app.use(cors());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded
});
app.use(bodyParser.json());
```

Рисунок 9 – повторяющийся app.use(bodyParser.json());

Решением данной проблемы является удалением одного из app.use(bodyParser.json());, а именно второго.

Проблема номер три – использование res.redirect в API. Это не характерно для REST, оно характерно для серверных маршрутов. Проблема изображена на рисунке 10.

```
function (err) {
  if (err) return res.status(500).s
  res.redirect('/blog/' + blogId);
}
```

Рисунок 10 – использование redirect в REST

Решением данной проблемы является замена redirect на json.

Также была найдена еще одна проблема с повторением функции, а именно повторялось функция выхода из личного кабинета. Функция выхода из акккаунта изображена на рисунке 11.

```
// Выход из аккаунта
app.post('/api/logout', (req, res) => {
  req.session.destroy(() => {
    res.json({ success: true });
  });
});
```

Рисунок 11 – функция выхода из аккаунта

Чтобы немного улучшить скорость работы системы, необходимо было убрать одну лишнюю функцию выхода из аккаунта.

Во время ревьюрирования программного кода информационной системы для блога об электроэнергетики была обнаружена уязвимость, а именно уязвимость к XSS-атакам.

XSS – довольно распространенная уязвимость, которую можно обнаружить на множестве веб-приложений. Ее суть довольно проста, злоумышленнику удается внедрить на страницу JavaScript-код, который не был предусмотрен разработчиками. Этот код будет выполняться каждый раз, когда жертвы будут заходить на страницу приложения, куда этот код был добавлен.

Данная уязвимость была обнаружена в функции для загрузки комментария. Функция для загрузки комментария изображена на рисунке 12.

```
// Загрузка комментариев
async function loadComments() {
    try {
        const res = await fetch(`/api/blogs/${blogId}/comments`);
        const comments = await res.json();
        const container = document.getElementById('comments');
        container.innerHTML = comments.map(c => `<p><b>${c.username}:</b> ${c.text}</p>`).join('');
    } catch (error) {
        console.error('Ошибка при загрузке комментариев:', error);
    }
}
```

Рисунок 12 – функция для загрузки комментариев

Для исправления данной уязвимости необходимо исправить код, сделать в более защищенном, используя функции textContent и createTextNode. Исправленный код изображен на рисунке 13.

```
async function loadComments() {
    try {
        const res = await fetch(`/api/blogs/${blogId}/comments`);
        const comments = await res.json();
        const container = document.getElementById('comments');

        // Очистка контейнера перед вставкой новых комментариев
        container.innerHTML = '';

        comments.forEach(c => {
            const p = document.createElement('p');

            const b = document.createElement('b');
            b.textContent = `${c.username}: `;
            p.appendChild(b);

            const text = document.createTextNode(c.text);
            p.appendChild(text);

            container.appendChild(p);
        });

    } catch (error) {
        console.error('Ошибка при загрузке комментариев:', error);
    }
}
```

Рисунок 13 – исправленный код загрузки комментариев

Благодаря textContent и createTextNode, они автоматически экранируют любые HTML-теги, введённые пользователями. Это предотвращает внедрение <script> и других опасных конструкций.

4.6 Рефакторинг программного кода

Рефакторинг – изменение во внутренней структуре ПО, имеющее целью облегчить понимание его работы и упростить модификацию, не затрагивая наблюдаемого поведения. Рефакторинг позволяет нам улучшить читаемость, поддерживаемость и эффективность кода, делая его более гибким и легким для внесения будущих изменений.

Существует несколько причин, почему возникает необходимость. проведения рефакторинга программного кода:

– снижение технического долга. Технический долг – это накопленные проблемы и недостатки в коде, которые могут замедлять разработку и усложнять поддержку продукта. Рефакторинг помогает устранить эти проблемы и снизить технический долг, что способствует более плавному и эффективному процессу разработки;

– улучшение читаемости и тестируемости кода. Улучшение читаемости не только сократит время на чтение кода, но и поспособствует более легкому развитию и поддержке программного обеспечения в долгосрочной перспективе, а также облегчит написание автоматических тестов даже тем людям, у которых разработка не является основным направлением;

– повышение эффективности разработчиков. Когда код унифицирован, структурирован и понятен, можно с легкостью внедрять новые функции или поддерживать существующие. Кроме того, благодаря этому разработчики могут быстрее ориентироваться в коде и вносить изменения без риска нежелательного влияния на другие части программного обеспечения, что в конечном итоге снижает количество ошибок.

Перед началом рефакторинга необходимо иметь обширный набор тестов, который покрывает все аспекты функциональности приложения. Тесты должны запускаться автоматически после каждого изменения, чтобы убедиться в отсутствии регрессии.

Изменения в коде выполняются таким образом, чтобы не нарушить его работоспособность. Даже если в коде много «некрасивых» решений, приоритет отдается стабильности работы приложения. Также изменения, выполняемые в процессе рефакторинга, не должны видоизменять внешний интерфейс программы и ее поведение с точки зрения пользователя.

Выполнение преобразований небольшими последовательными шагами позволяет избежать ошибок, которые могут возникнуть при крупных и радикальных переработках. Эта практика также облегчает отслеживание внесенных изменений и откат нежелательных эффектов.

Рефакторинг кода проходит в следующих случаях:

– код сложен и труден для понимания. Плохая читаемость кода свидетельствует о высокой сложности его конструкций и отсутствии четкой структуры. Рефакторинг в этом случае направлен на упрощение структуры кода и на улучшение его логичности и организации. Таким образом код становится понятнее, что ускоряет разработку и облегчает обучение новых членов команды;

– в коде много дублирования. Частые повторения одинаковых или похожих фрагментов кода увеличивают вероятность ошибок при внесении изменений, так как каждый случай дублирования требует одинаковых корректировок. Это не только утяжеляет программу, но и влечет риски для ее поддержки и расширения. Рефакторинг с целью избавления от дублирования часто включает создание общих методов или классов. Сокращение дублированных частей упрощает процессы тестирования и сопровождения кода;

– производительность системы снижена. Анализ показывает, что медленная работа часто связана не с инфраструктурными ограничениями, а с неоптимальным кодом. Рефакторинг с акцентом на производительность может включать оптимизацию алгоритмов, устранение избыточных вычислений и уменьшение числа операций ввода-вывода. Улучшение эффективности кода часто ведет к лучшему пользовательскому опыту и сокращению ресурсов, необходимых для поддержки приложения.

– код содержит устаревшие решения. Устаревший код может стать причиной проблем с безопасностью, производительностью и интеграцией с другими системами. Рефакторинг с учетом последних достижений в области технологий дает возможность модернизировать приложение. Обновление кодовой базы обеспечивает лучшую поддержку, совместимость и предоставляет возможность использования нового функционала.

В рефакторинге участвовало два файла server.js и script.js. В server.js было много повторяющихся ссылок на кокретные файлы, что могло повлиять на скорость работы системы. Код до рефакторинга изображенен на рисунке 14.

```
app.get('/register', (req, res) => res.sendFile(path.join(__dirname, 'public/register.html')));
app.get('/login', (req, res) => res.sendFile(path.join(__dirname, 'public/login.html')));
app.get('/profile', (req, res) => res.sendFile(path.join(__dirname, 'public/profile.html')));
app.get('/comments', (req, res) => res.sendFile(path.join(__dirname, 'public/bloglist.html')));
app.get('/admin', (req, res) => res.sendFile(path.join(__dirname, 'public/admin.html')));
```

Рисунок 14 – код до рефакторинга

После рефакторинга код стал короче и проще в развитии. Все повторяющиеся маршруты были объединены в один цикл forEach. forEach используется для итерации элементов массива и выполнения указанной функции один раз для каждого элемента. Чтобы добавить новый маршрут, достаточно просто добавить его в массив. Код после рефакторинга изображен на рисунке 15.

```
['register', 'login', 'profile', 'comments', 'admin'].forEach(route => {
  app.get(`/${route}`, (req, res) => res.sendFile(path.join(__dirname, `public/${route}.html`)));
});
```

Рисунок 15 – код после рефакторинга

До рефакторинга программного кода таблиц базы данных информационные системы для блога об электроэнергетике, все таблицы были прописаны раздельно. Это может повлиять на скорость создания и подключения к таблицам в базе данных. На рисунке 16 продемонстрирован код базы данных до рефакторинга.

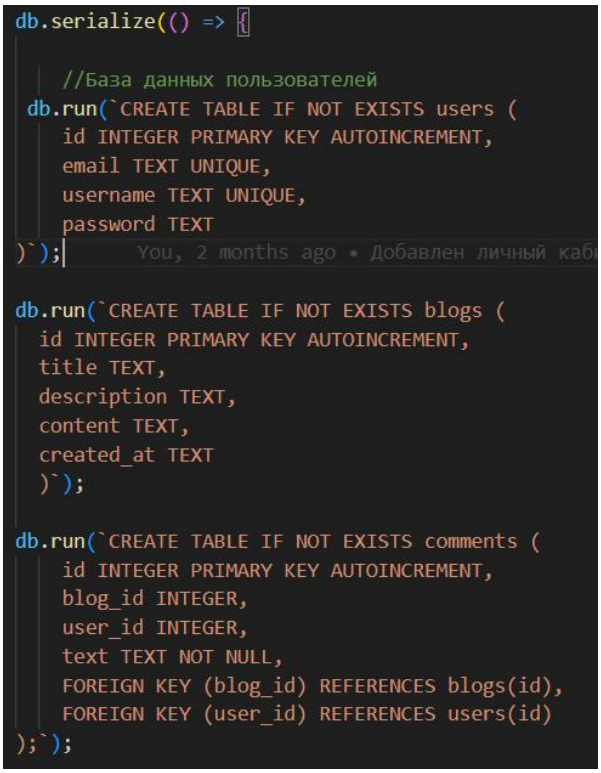


Рисунок 16 – код до рефакторинга

После рефакторинга, создание всех таблиц было объединено в одну функцию const tabels, к которой подключена функция db.serialize. Благодаря этому создание и подключение таблиц станет быстрее. Код создание таблиц после рефакторинга изображен на рисунках 17 и 18 .

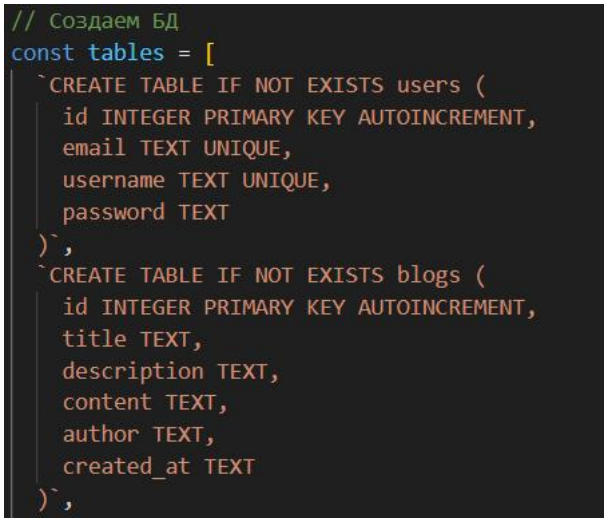


Рисунок 17 – код для добавления таблиц после рефакторинга

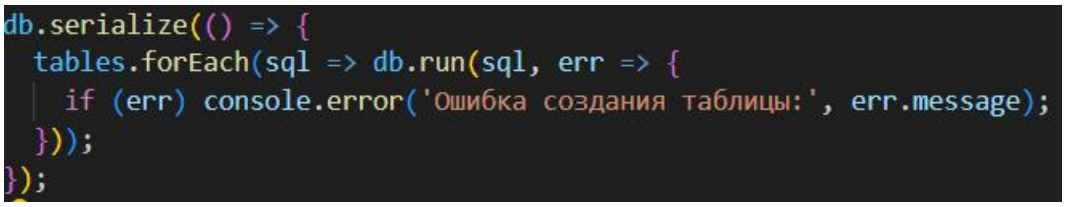


Рисунок 18 – код создания таблиц после рефакторинга

В script.js каждая функция делала fetch, res.json и так далее. Это не практично, так как это многократное дублирование, которое усложняет поддержку и увеличивает вероятность ошибок. На рисунке 19 изображен один из примеров, где в функциях использовались fetch и res.json.

```
// Загрузка комментариев
async function loadComments() {
    try {
        const res = await fetch(`/api/blogs/${blogId}/comments`);
        const comments = await res.json();
        const container = document.getElementById('comments');
```

Рисунок 19 – код с fetch и res.json

Поэтому для облегчения поддержки и уменьшения вероятности ошибок была создана отдельная fetchJSON. Это позволит системе обрабатывать все ошибки, а также выводить понятные сообщения об ошибке. Обработчик ошибок представлен на рисунке 20.

```
// Универсальный fetch с обработкой ошибок
const fetchJSON = async (url, options = {}) => {
    try {
        const res = await fetch(url, options);
        if (!res.ok) throw new Error(await res.text());
        return await res.json();
    } catch (err) {
        console.error(`Ошибка запроса к ${url}:`, err);
        throw err;
    }
};
```

Рисунок 20 – обработчик ошибок

В исходной реализации функции deleteRating присутствует дублирующий запрос к /api/session, несмотря на наличие функции checkLogin(), которая уже выполняет аналогичную проверку авторизации. Данная функция представлена на рисунке 21.

```
// Проверка логина
const checkLogin = async () => {
    const res = await fetch('/api/session');
    const data = await res.json();
    return data.loggedIn;
};
```

Рисунок 21 – проверка логина

Вместо дублирования кода используется повторное применение функции checkLogin(), что делает реализацию короче, чище и проще в поддержке: при изменении логики авторизации потребуется внести правки только в одном месте - в checkLogin(). Обновленная функция проверки логина продемонстрирована на рисунке 22. Также пример использования функции проверки логина в других функциях представлен рисунке 23.

```
const checkLogin = async () => {
    const data = await fetchJSON('/api/session');
    return data.loggedIn;
};
```

Рисунок 22 – обновленная проверка логина

```
async function sendComment() {
    if (!await checkLogin()) {
        alert('Пожалуйста, войдите в аккаунт, чтобы оставить комментарий.');
        window.location.href = "/login.html";
        return;
    }
```

Рисунок 23 – пример использование функции проверки логина в функции отправки комментария

Изначально выполнение DOM-дерева не является асинхронным, в результате чего вложенные в него функции запускаются одновременно, не ожидая завершения друг друга. Это может привести к сбоям в случае зависимости одной функции от другой, визуальным ошибкам из-за несвоевременной загрузки данных, а также к трудноотлавливаемым сбоям выполнения. Первоначальный год изображен на рисунке 24.

```
document.addEventListener('DOMContentLoaded', () => {
    if (!blogId) {
        console.error("blogId не найден в URL");
        return;
    }
    fetchAverageRating();
    loadComments();
    loadUserRating();
});
```

Рисунок 24 – первоначальный вид DOM-дерева

После рефакторинга DOM-дерево использует функцию асинхронности, что позволяет использовать функцию await. Это гарантирует, что все функции в DOM-дереве будут загружаться по очереди. Это позволит избежать меньше ошибок, а также поможет легче обновлять DOM-дерево. DOM-дерево после рефакторинга изображено на рисунке 25.

```
document.addEventListener('DOMContentLoaded', async () => {
    if (!blogId) {
        console.error("blogId не найден в URL");
        return;
    }

    await fetchAverageRating();
    await loadComments();
    await loadUserRating();
});
```

Рисунок 25 – DOM-дерево после рефакторинга

4.7 Программа и методика испытания веб-приложения

Испытания веб-приложения – это процесс всесторонней проверки работоспособности, стабильности, функциональности, безопасности и удобства использования веб-системы с целью выявления и устранения ошибок до внедрения в эксплуатацию.

Существует несколько видов тестирования, а именно:

– функциональное тестирование – проверка корректности работы функционала сайта или веб-приложения;

– тестирование производительности – измерение скорости загрузки страниц, а также способности сайта или приложения выдерживать большое количество пользователей и запросов;

– тестирование безопасности – проверка наличия уязвимостей и устойчивости к потенциальным атакам;

– тестирование совместимости – обеспечение корректной работы веб-приложения на разных браузерах, устройствах и операционных системах;

– тестирование юзабилити – оценка удобства и простоты использования сайта или приложения для пользователей;

– ручное тестирование – процесс поиска ошибок в программе без использования специальных ПО, силами человека.

Объектом тестирования является веб-приложение «Блог об электроэнергетике».

Комплектность системы:

– серверная часть: server.js, база данных ratings.db, папка public (HTML, CSS, JS);

– клиентская часть: HTML-страницы (index.html, blog.html, login.html, admin.html и др.);

– база данных: SQLite-файл с таблицами пользователей, блогов, комментариев, рейтингов;

– документация: руководство пользователя, руководство системному программисту, инструкция по установке.

Оценка результатов проведения тестирования разработки информационной системы для блога об электроэнергетике включает анализ результатов тестов и их соответствие ожиданиям и требованиям. Так же, оценка результатов проведения тестирования помогает понять текущее состояние сайта. Он включает в себя результаты тестирования, выявленные дефекты и рекомендации по их устранению. Такой отчет позволяет:

– оценить качество сайта;

– определить области, требующие улучшений;

– обеспечить прозрачность процесса тестирования;

– сформировать план действий для дальнейшего развития проекта.

Отчет о тестировании также служит важным инструментом для коммуникации между различными участниками проекта. Он помогает менеджерам проектов, разработчикам, тестировщикам и другим

заинтересованным сторонам быть в курсе текущего состояния сайта и принимать обоснованные решения. Кроме того, отчет может быть использован для документирования процесса тестирования и хранения информации для будущих проектов. Вот несколько важных аспектов, которые следует учитывать при оценке результатов.

Успешность прохождения тестов. Необходимо проанализировать количество и типы ошибок, выявленных в процессе тестирования. Если тесты успешно проходят без ошибок, это может указывать на то, что информационная система по садоводству работает стабильно и соответствует своим требованиям. Если же при тестировании были обнаружены те или иные ошибки и уязвимости, то ошибки и уязвимости нужно исправить

Покрытие функциональности. Важно проверить, насколько полностью тестирование охватывает все функциональные возможности системы. Если тесты покрывают большую часть функций информационной системы по садоводству, это говорит о хорошем покрытии тестами.

Отклонения от требований. Если есть отклонения или несоответствия, важно документировать их и обратить внимание на их серьезность. Некоторые отклонения могут быть незначительными и легко устранимы, в то время как другие могут требовать значительных изменений в системе. Если не проверить на отклонения от требований, это может повлиять на будущее работы информационной системы.

Производительность и надежность. Бывают случаи, когда на информационную систему заходит много людей, из-за чего, может произойти понижение производительности или же полная неработоспособность сайта. Поэтому необходимо оценить результаты тестирования, чтобы определить производительность и надежность информационной системы по садоводству. Важно убедиться, что система работает эффективно и надежно в различных условиях и нагрузках.

Реакция на ошибки. Адекватная реакция на ошибки – критически важный аспект надежности программного обеспечения. Система должна не только обнаруживать нештатные ситуации, но и корректно реагировать на них, минимизируя негативное воздействие на пользователя и другие компоненты. Грамотная обработка ошибок подразумевает предоставление информативных сообщений, позволяющих пользователю понять проблему и предпринять необходимые действия, а также логирование для последующего анализа причин сбоев.

Соответствие пользовательскому интерфейсу. Оценить, насколько интуитивно понятен и удобен пользовательский интерфейс информационной системы по садоводству. Если пользователи успешно выполняют необходимые действия без затруднений, это может свидетельствовать о высокой эффективности разработки. Если же пользователю сложно найти какие-либо элементы или функции на сайте, то следует пересмотреть дизайн сайта и/или удалить не нужные элементы и функции

Надежность безопасности. Так же необходимо уделить внимание проверке системы на наличие уязвимостей и недостатков в области безопасности. Важно

убедиться, что информационная система по садоводству защищена от несанкционированного доступа и не представляет рисков для пользователей и окружающей среды. Если же у системы есть какие-то уязвимости безопасности, то может произойти утечка данных клиентов или злоумышленники смогут взломать систему и использовать ее в своих целях и интересах.

Документация и отчётность. Так же нужно проверить качество документации, которая сопровождает информационную систему по садоводству. Она должна быть четкой, понятной и обеспечивать достаточно информации для поддержки и эксплуатации системы. Также убедиться, что созданы отчеты о результатах тестирования, которые содержат достаточно информации для оценки.

Внесение корректировок и улучшений. Результаты тестирования могут выявить слабые места или недочеты в информационной системе по садоводству. Важно, чтобы эти результаты использовались для улучшения разработки. Оцените, насколько эффективно команда разработчиков реагирует на обнаруженные проблемы, вносит исправления и выпускает обновления.

В ходе тестирования была проверена на выявление проблем с производительностью сайта. Это позволило нам определить, какие есть проблемы со скоростью загрузки веб-приложения и что нужно для этого сделать. Для тестирования на производительность использовался онлайн сервис PageSpeed Insights. Результат тестирования изображен на рисунке 26.

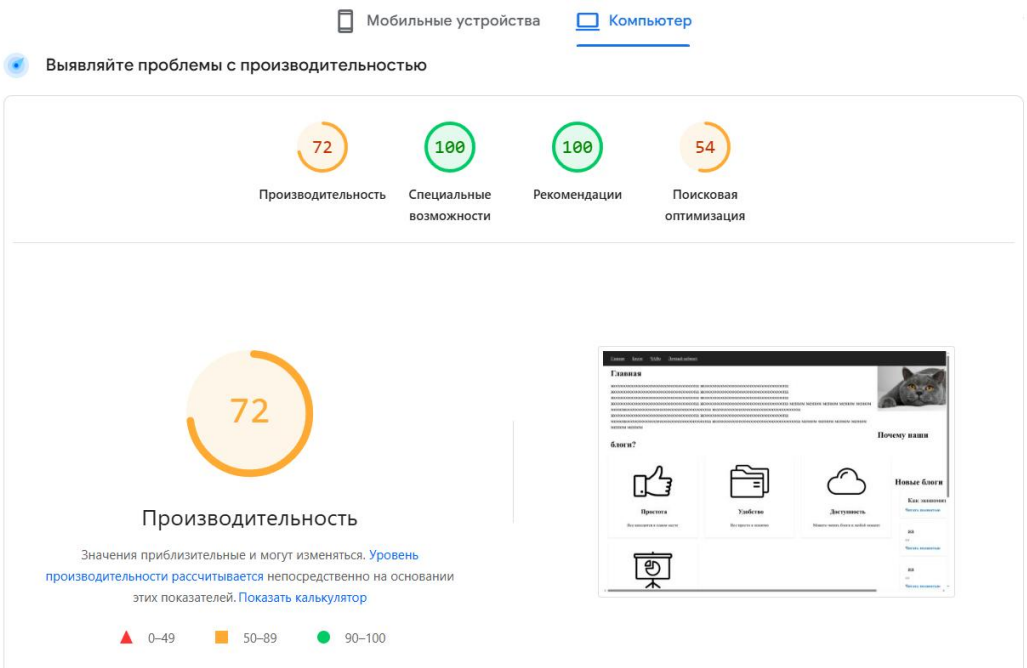


Рисунок 26 – результат тестирования веб-приложения.

Это тестирование показало, что веб-приложения «Блог об электроэнергетике» есть проблемы с производительностью. Проблемы возникли у двух параметров, а именно у Cumulative Layout Shift и у Speed Index.

Cumulative Layout Shift (совокупное смещение макета) – это величина, на которую смещаются видимые элементы области просмотра при загрузке. Чем ниже показатель, тем лучше.

Speed Index (индекс скорости загрузки) показывает, как быстро на странице появляется контент. Так же, чем ниже, тем лучше. Показатели после тестирования изображены на рисунке 27.

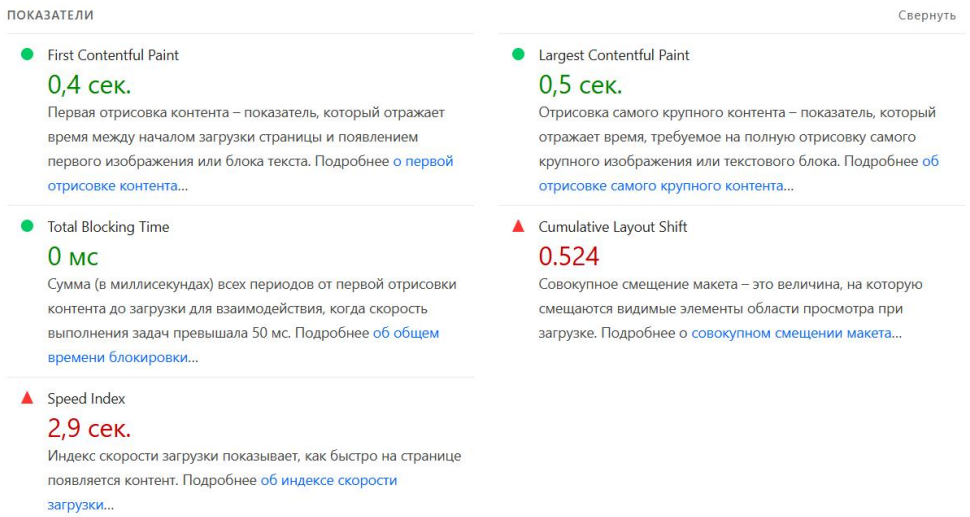


Рисунок 27 – итоговые показатели после тестирования

Промежуточном итоге можно сказать, что в основном веб-приложение прошло все тесты единственное, что необходимо сделать это повысит производительность сайта.

Так же проводилось ручное тестирование, чтобы определить, работают ли все функции в информационной системе для блога об электроэнергетике. Это было необходимо, чтобы убедиться, что все функции системы и работа с базой данных работает корректно и без перебоев. Так же, без ручного тестирования, мы не смогли бы понять, где могут находиться ошибки в информационной системе и это повлияло бы на стабильность работы и привлечение новых пользователей. В тестировании,участвовали регистрация и удаление нового администратора, добавление блога и так далее.

Первым, что необходимо проверить в информационной системе для блога об электроэнергетике – регистрация и удаление нового администратора, работает ли система проверки полей ввода данных. Форма регистрации нового администратора изображена на рисунке 28.

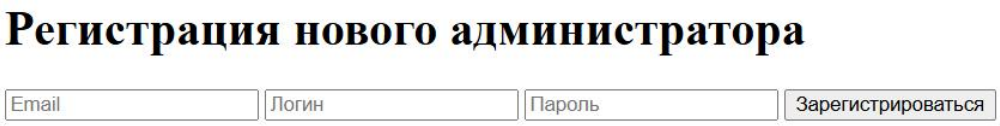


Рисунок 28 – форма регистрации нового администратора

Информационная система для блога об электроэнергетике предоставляет компании возможность зарегистрировать нового администратора в системе. Это позволит новому администратору добавлять, редактировать и удалять блоги Для регистрации в системе, необходимо ввести следующие данные:

– имя пользователя;

– электронная почта;

– придумать пароль.

Если попробовать нажать кнопку «Зарегистрироваться» с пустыми полями ввода, то сайт запросит ввести соответствующие данные. Это сделано для того, чтобы в базе данных не создавались пустые регистрационные данные. Ошибка изображена на рисунке 29.

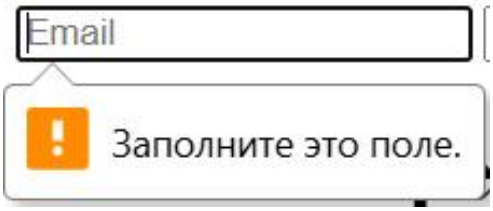


Рисунок 28 – ошибка при вводе пустых значений

Далее мы вводим недостающие данные. Если мы попробуем зарегистрировать пользователя с паролем меньше, чем 8 символов, выдается соответствующая ошибка. На рисунке 29 изображена ошибка с паролем.

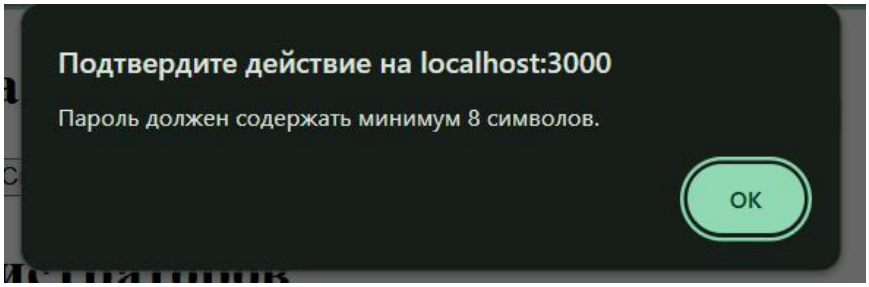


Рисунок 29 – ошибка с паролем

После ввода всех необходимых данных нажимаем кнопку зарегистрироваться. После регистрации нам система скажет, что администратор зарегистрирован, и он уведомлен по почте. В этом сообщении прикреплен файл, в котором указаны все регистрационные данные, такие как логин, пароль и секретный ключ, который генерируется автоматически. На рисунке 30 изображено сообщение после регистрации.

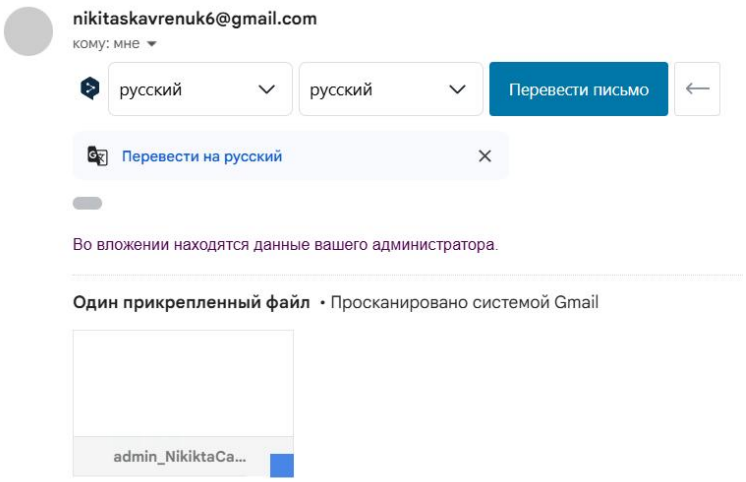


Рисунок 30 – сообщение после регистрации

Теперь мы должны проверить функцию удаления администратора. Эта функция необходима так и примеру, если сотрудник был уволен, то его личный

кабинет администратора также необходимо удалить. Для этого на странице регистрации нового администратора есть блок со всеми зарегистрированными администраторами, который изображен на рисунке 31.

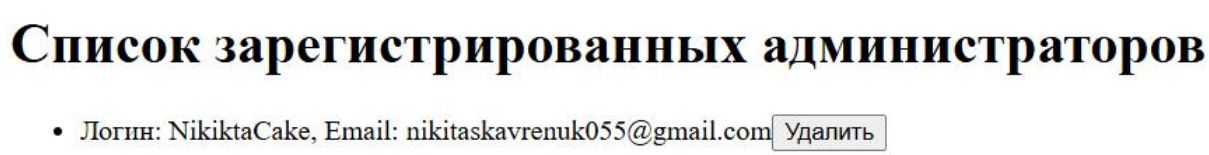


Рисунок 31 – блок со всеми зарегистрированными администраторами

Чтобы утолить администратора, необходимо нажать кнопку удалить. После чего нас спросят, действительно ли хотите удалить администратора, подтверждаем это действие. После удаления администратора, придет на почту сообщения о том, то, что он больше не является администратором на сайте. Данное сообщение изображено на рисунке 32.

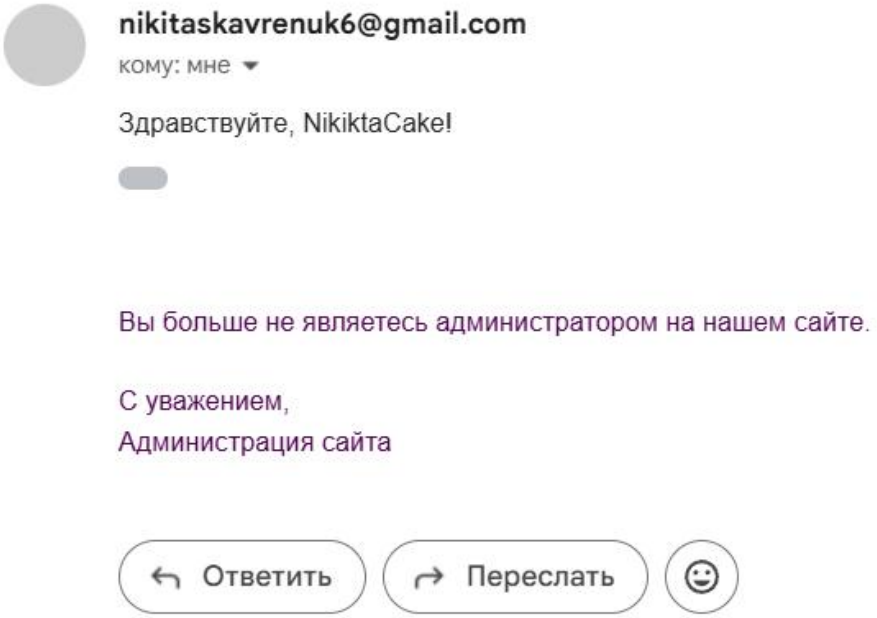


Рисунок 32 – сообщение о том то, что не являетесь администратором на сайте.

В промежуточном итоге, мы можем выделить, что форма регистрации нового аккаунта работает корректно, система не регистрирует аккаунт с пустыми данными, так же система не регистрирует пользователя без электронной почты и пароля, удаление работает так же корректно. Новый пользователь был зарегистрирован и внесен в базу данных администраторов блога об электроэнергетике

Далее необходимо проверить добавление нового блога, Для этого необходимо войти с учетной записи администратора. После входа администратора встретит админ панель, Которая поможет в дальнейшем администратору добавлять, редактировать и удалять блоги. Данная админ панель продемонстрирована на рисунке 33.

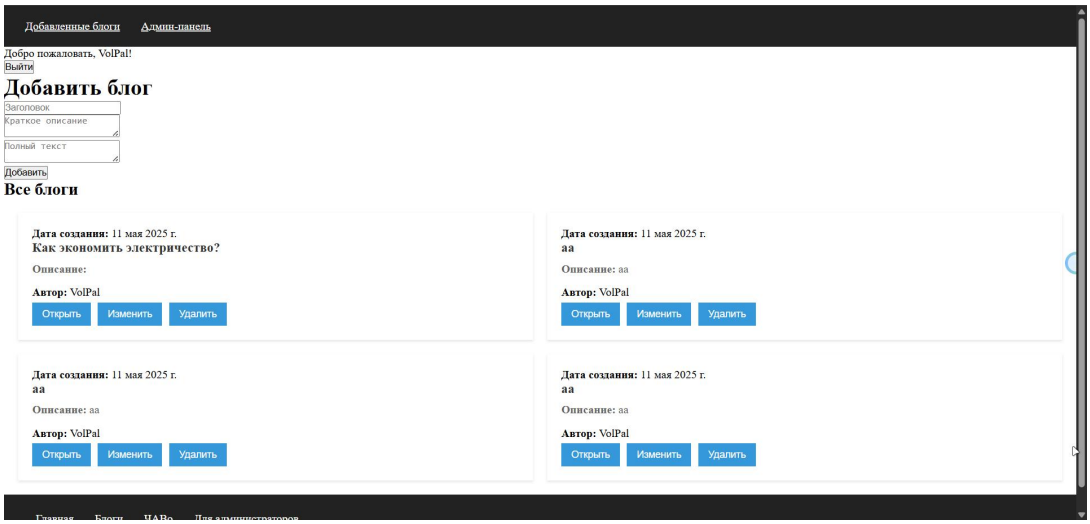


Рисунок 33 – админ панель

Чтобы добавить новый блок, необходимо ввести данные в небольшую форму. Форма имеет 3 пункта заголовок, краткое описание и полный текст. Форма добавления блога изображена на рисунке 34.

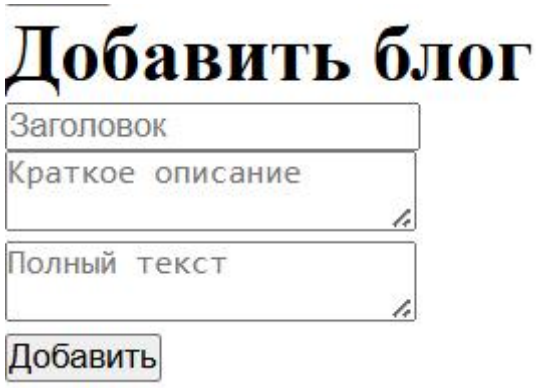


Рисунок 34 – форма добавления блога

После ввода всех необходимых данных форму нажимаем кнопку добавить, после чего новый блог появится ниже.

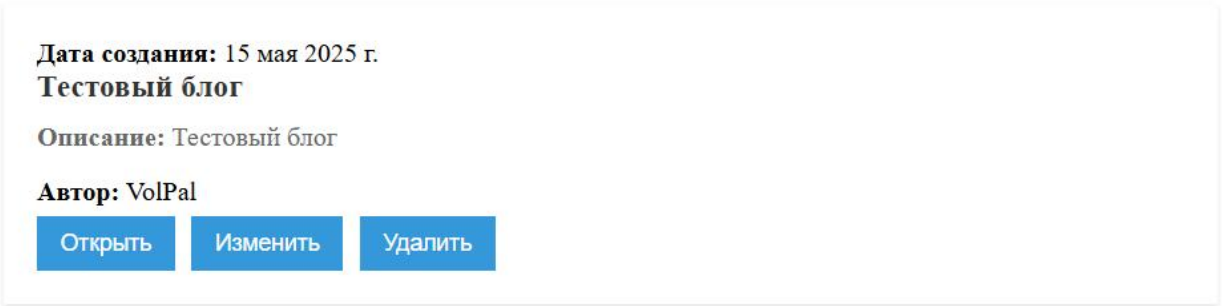


Рисунок 35 – новый блог

В общем итоге, можно выделить следующие выводы, все тесты прошли успешно и без проблем. Регистрация администратора проходит успешно и новый администратор добавляется в базу данных, администратор может может войти в свою учетную запись, блог добавляются в систему. Все тесты завершились успешно, продемонстрировав корректность и стабильность работы системы.

5 Техника безопасности и пожарная безопасность

Во время прохождения производственной практики в ПАО «Россети Волга» – филиал «Оренбургэнерго», особое внимание уделялось соблюдению требований техники безопасности и пожарной безопасности при работе с компьютерной техникой и ИТ–инфраструктурой. Безопасная организация труда – одна из приоритетных задач компании, особенно в подразделениях, связанных с эксплуатацией электрооборудования и информационных технологий.

Раздел содержит описание основных мер, направленных на защиту сотрудников от вредных и опасных факторов производственной среды, возникающих при установке, эксплуатации, обслуживании и ремонте вычислительной техники, серверного оборудования и при использовании ИТ–систем.

В компании «Оренбургэнерго» все работы, связанные с подключением, техническим обслуживанием и ремонтом вычислительной и электронной техники, проводятся с соблюдением правил электробезопасности.

Основные меры:

– все ИТ–устройства подключаются через заземлённые розетки с защитой от короткого замыкания и перенапряжений;

– ипользуются источники бесперебойного питания (ИБП) для серверного и критически важного оборудования;

– при техническом обслуживании компьютеров или другой техники обязательно полное отключение питания;

– лица, допускаемые к работам, проходят инструктаж по электробезопасности и имеют соответствующую группу допуска (например, I или II);

– все соединительные кабели, удлинители, блоки питания проходят регулярный осмотр на предмет повреждений и перегрева.

В помещениях, где размещено чувствительное оборудование, особенно серверное и коммутационное, соблюдаются меры защиты от ЭМИ (электромагнитных излучений) и накопления статического электричества:

– серверные помещения экранируются и размещаются в изолированных зонах;

– используются антистатические коврики и заземляющие браслеты при работе с компонентами компьютеров;

– применяются экранированные сетевые кабели для минимизации помех.

ИТ–оборудование может создавать акустические шумы, особенно серверы, источники питания, вентиляционные установки:

– серверные комнаты изолированы от основных рабочих помещений и оснащены системой шумопоглощения;

– рабочие места сотрудников соответствуют санитарным нормам по уровню шума – до 50 дБ в течение 8–часового рабочего дня;

– регулярно проводится техническое обслуживание вентиляторов, кулеров и других подвижных компонентов для исключения избыточных шумов и вибраций.

Оптимальный уровень освещённости оказывает важное влияние на здоровье сотрудников и эффективность работы:

– в помещениях организации установлено комбинированное освещение (естественное и искусственное);

– освещённость составляет не менее 300 люкс;

– светильники расположены таким образом, чтобы исключить блики на мониторах и снизить утомляемость глаз.

В условиях эксплуатации большого количества электрических и вычислительных устройств повышается риск возгораний, что требует строгого соблюдения норм пожарной безопасности:

– все помещения оснащены противопожарной сигнализацией и системами автоматического оповещения;

– в серверных и кабинетах установлены углекислотные и порошковые огнетушители;

– используются негорючие кабели с двойной изоляцией;

– запрещено использование неисправных электрических приборов, перегрузка розеток и применение самодельных удлинителей;

– регулярно проводится проверка состояния кабельной инфраструктуры и электрооборудования;

– в компании разработан и утверждён план эвакуации при пожаре, размещённый на каждом этаже здания.

Работы по ремонту и техническому обслуживанию компьютерной и серверной техники выполняются только специалистами ИТ–отдела:

– перед началом работ оборудование полностью обесточивается.

– при замене компонентов (память, процессор, видеокарта и т.д.) применяются антистатические средства защиты.

– используется специализированный инструмент и диагностическое программное обеспечение.

– в случае работ в серверной зоне требуется наличие второго специалиста и регистрация в журнале посещений.

Заключение

Разработка программного обеспечения для образовательного блога об электроэнергетике позволила создать удобную, функциональную и адаптивную веб–платформу для обмена знаниями, публикации аналитических и обучающих материалов, а также взаимодействия между пользователями, интересующимися энергетикой и смежными инженерными дисциплинами.

В процессе создания проекта были последовательно реализованы этапы:

– анализа требований целевой аудитории и особенностей отрасли;

– проектирования архитектуры системы, включая базу данных, модульную структуру и пользовательский интерфейс;

– непосредственной разработки и программной реализации веб–приложения с применением HTML, CSS, JavaScript и SQLite.

В результате была разработана информационная система, обеспечивающая:

– публикацию статей, инструкций и обзоров, связанных с темами электроэнергетики, электробезопасности, распределительных сетей, работы с высоковольтным оборудованием, энергосбережением и т.д.;

– возможность для зарегистрированных пользователей комментировать материалы, ставить оценки и подписываться на авторов, что способствует вовлеченности и развитию сообщества;

– реализацию адаптивного интерфейса, корректно работающего на любых устройствах – от настольных ПК до мобильных телефонов и планшетов.

Одним из ключевых преимуществ является интуитивно понятная навигация и простота подачи информации, что делает ресурс доступным как для специалистов отрасли, так и для студентов энергетических направлений, преподавателей и любителей технических дисциплин. Также предусмотрены механизмы поиска по тематикам и авторам, фильтрации по рейтингу и дате публикации, что облегчает доступ к нужным материалам.

Разработанная система обладает высокой степенью масштабируемости. В будущем возможно интеграция с социальными сетями и мессенджерами для расширения охвата аудитории, внедрение расширенной статистики активности пользователей, просмотров, кликов и вовлеченности, реализация системы уведомлений, информирующих пользователей о новых комментариях, реакциях, публикациях подписанных авторов и других событиях.

Цель проекта – разработка удобной, безопасной и эффективной среды для обмена знаниями в области электроэнергетики – была успешно достигнута. Внедрение такого ресурса особенно ценно для формирования и поддержки профессионального сообщества, повышения технической грамотности и распространения современных знаний в одной из ключевых отраслей экономики.

Сайт может использоваться как вспомогательный образовательный инструмент для энергетических учебных заведений, как внутренняя корпоративная платформа для обмена знаниями между инженерами, или как публичный информационный ресурс для широкой технической аудитории.

Список используемых источников

1 Филиал «Оренбургэнерго». — Текст : электронный // energybase.ru : [сайт]. — URL: https://energybase.ru/distribution/orenburgenergo. – 300 с.

2 Trending icons. — Текст : электронный // Uicons : [сайт]. — URL: https://www.flaticon.com/uicons/interface–icons.– 560 с.

3 Форум. — Текст : электронный // Cropas : [сайт]. — URL: https://cropas.by/seo–slovar/forum/. – 340 с.

4 Введение в HTML и CSS. — Текст : электронный // Hexlet : [сайт]. — URL: https://ru.hexlet.io/programs/html–css–basics–free. – 410 с.

5 Введение в JavaScript. — Текст : электронный // Hexlet : [сайт]. — URL: https://ru.hexlet.io/programs/javascript–basics–free. – 263 с.

6 Основы HTML и CSS. — Текст : электронный // Stepik : [сайт]. — URL: https://stepik.org/course/2621/promo. – 462 с.

7 Основы JavaScript. — Текст : электронный // Code Basics : [сайт]. — URL: https://code–basics.com/ru/languages/javascript. – 415 с.

8 Модальное окно Modal. — Текст : электронный // КотурГайды : [сайт]. — URL: https://guides.kontur.ru/components/popup–elements/modal/. – 440 с.

9 Как создать базу данных и связать с JAVASCRIPT?. — Текст : электронный // Хабр : [сайт]. — URL: https://qna.habr.com/q/1282298. – 725 с.

10 — Текст : электронный // Яндекс Картинки : [сайт]. — URL: https://yandex.ru/images/ – 340 с.

11 Делаем сами: адаптивный сайт — КОД. журнал Яндекс Практикума URL: https://thecode.media/bootstrap/.– 448 с.

12 Основные операции с данными — METANIT.COM URL: http://metanit.com/web/nodejs/8.4.php. – 520 с

13 О компании. — Текст : электронный // Россети Волга : [сайт]. — URL: https://www.rossetivolga.ru/ru/o_kompanii/. – 260 с.

14 Организационная структура. — Текст : электронный // Wikipedia : [сайт]. — URL: https://ru.wikipedia.org/wiki/Организационная_структура – 345 с.

15 Блог об электроэнергетике. — Текст : электронный // РН–ЭНЕРГО : [сайт]. — URL: https://www.rn–energo.ru/company/blog/ – 280 с.

16 Особенности, характеристики и области применения Node.js. — Текст : электронный // scand : [сайт]. — URL: https://scand.com/ru/company/blog/node–js–features–uses–and–benefits–of–development/ – 590 с.

17 9 бесплатных хостингов для сайтов: разбираемся, какой использовать. — Текст : электронный // SkillBox : [сайт]. — URL: https://skillbox.ru/media/marketing/9–besplatnykh–khostingov–dlya–saytov–razbiraemsya–kakoy–ispolzovat/ – 305 с.

18 Настройка Node.js. — Текст : электронный // beget : [сайт]. — URL: https://beget.com/ru/kb/how–to/web–apps/node–js – 200 с.

19 Работа с JSON в JavaScript. — Текст : электронный // Learn JavaScript : [сайт]. — URL: https://learn.javascript.ru/json. – 530 с.

20 Ошибки при использовании localStorage. — Текст : электронный // Stack Overflow : [сайт]. — URL: https://ru.stackoverflow.com/questions/542028/. – 450 с.

21 Современные подходы к верстке. — Текст : электронный // WebDev Notes : [сайт]. — URL: https://webdevnotes.com/css–layout–modern/. – 380 с.

22 Энергетическая безопасность России. — Текст : электронный // Energy Today : [сайт]. — URL: https://energytoday.ru/analytics/energy–security/. – 490 с.

23 Протокол HTTPS: зачем он нужен? — Текст : электронный // Хабр : [сайт]. — URL: https://habr.com/ru/articles/https–intro/. – 410 с.

24 Асинхронность в JavaScript. — Текст : электронный // Learn JS : [сайт]. — URL: https://learn.javascript.ru/async. – 512 с.

25 Как работает DOM. — Текст : электронный // DevDocs : [сайт]. — URL: https://devdocs.io/dom/. – 365 с.

26 Облачные технологии в энергетике. — Текст : электронный // Energy Cloud : [сайт]. — URL: https://energycloud.ru/articles/cloud–tech–power/. – 475 с.

27 Основы Python. — Текст : электронный // Stepik : [сайт]. — URL: https://stepik.org/course/67/promo. – 500 с.

28 Понятие API и его применение. — Текст : электронный // WebAPI Guide : [сайт]. — URL: https://webapiguide.ru/api–intro/. – 460 с.

29 JavaScript и безопасность. — Текст : электронный // OWASP : [сайт]. — URL: https://owasp.org/www–community/attacks/xss/. – 525 с.

30 Энергетические тренды 2025. — Текст : электронный // Аналитика Энерго : [сайт]. — URL: https://energoanalytics.ru/trends/2025/. – 355 с.

Приложение А
(обязательно)

Информационная модель

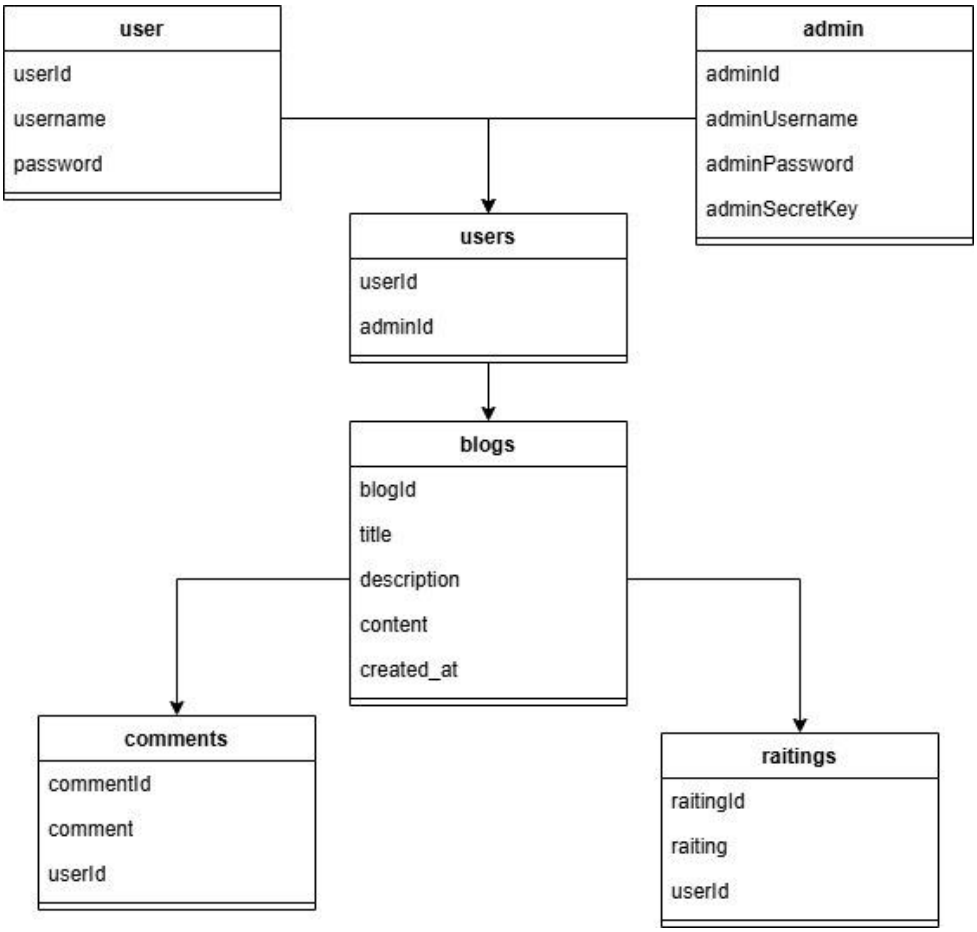


Рисунок А.1 – Информационная модель 3-й нормальной формы

Приложение Б
(обязательно)
Функциональная модель

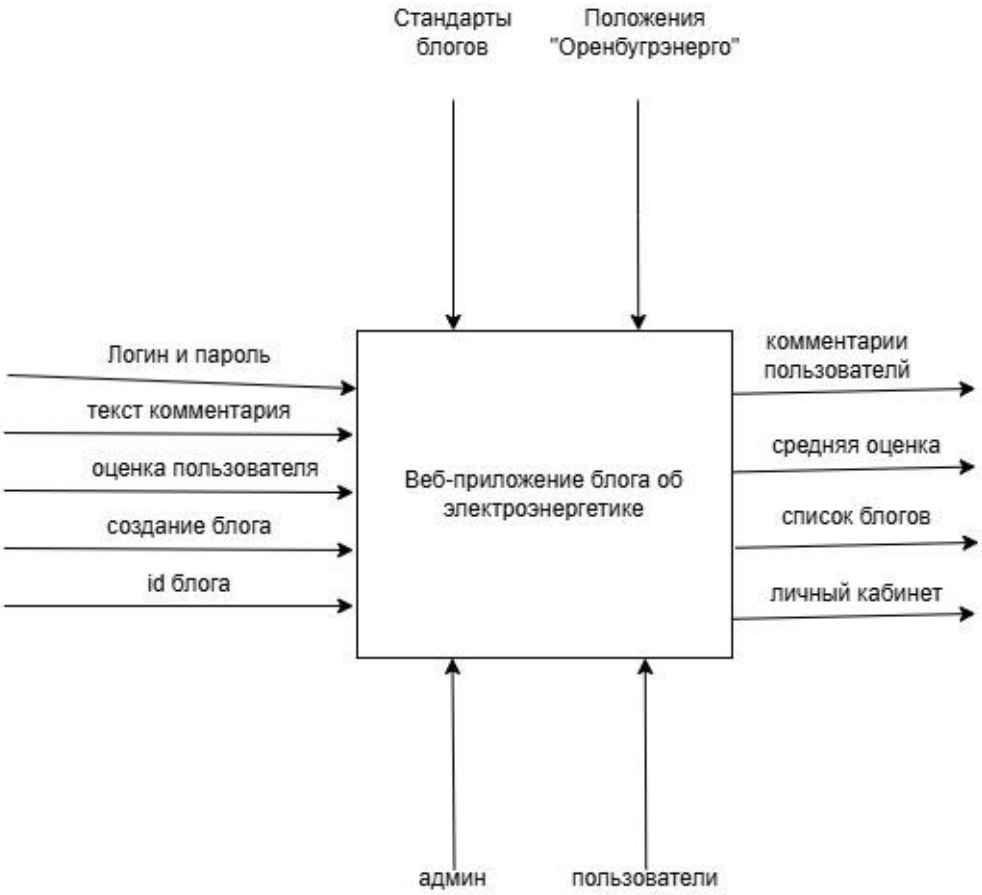


Рисунок Б.1 – Модель «Черный ящик»

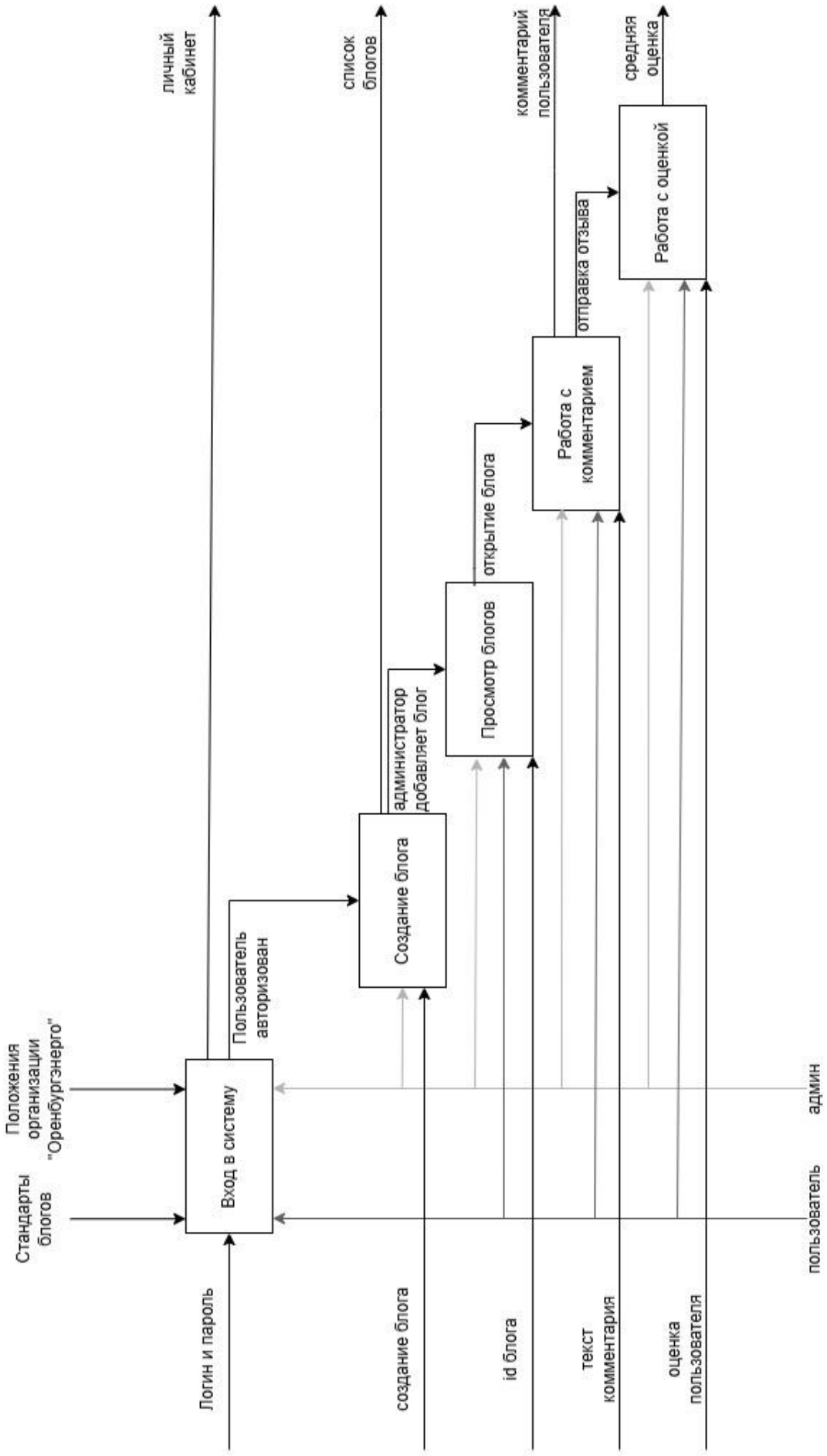


Рисунок Б.2 – Модель декомпозиции первого уровня.

Приложение В
(обязательно)
Схема работы системы

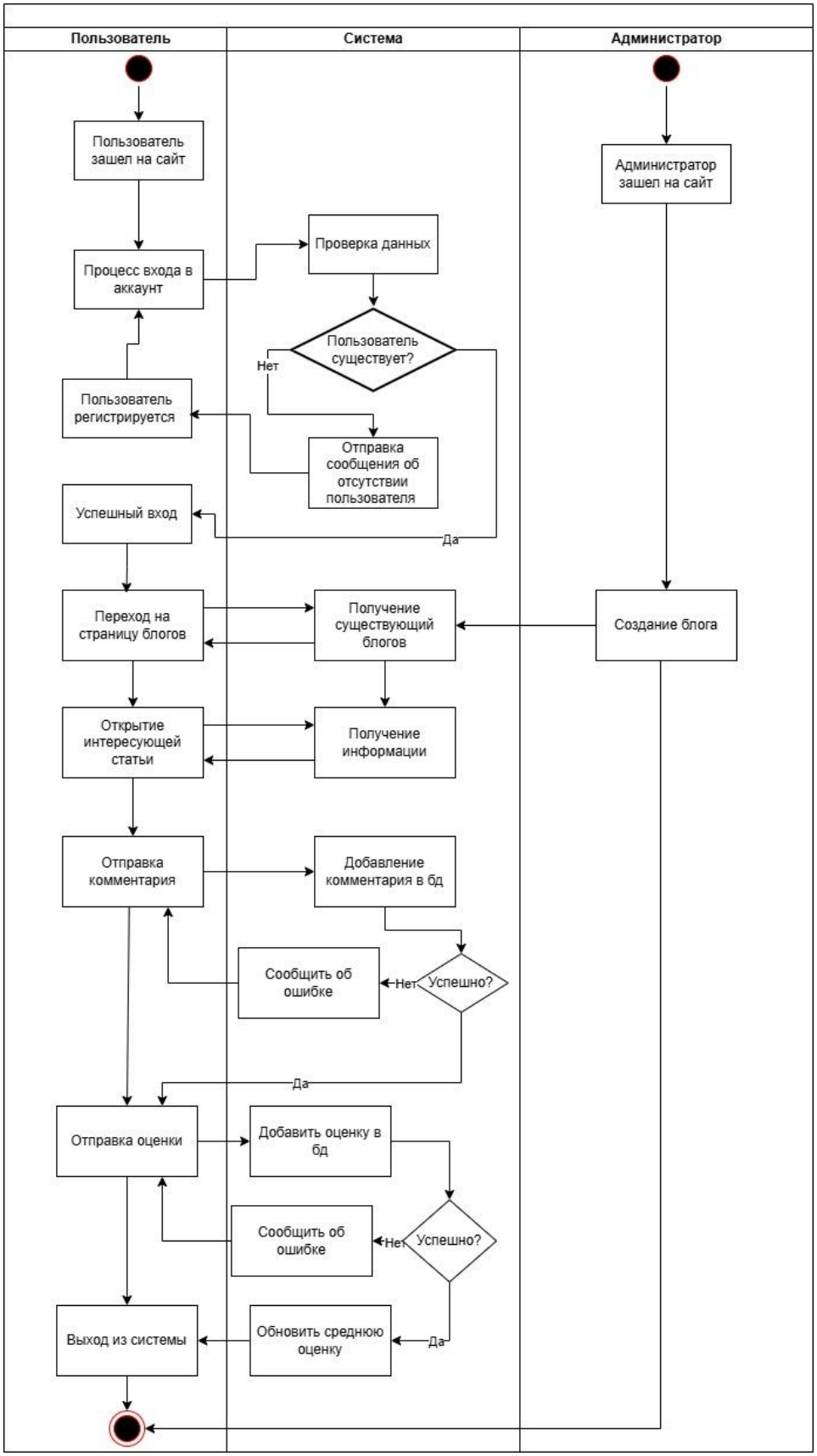


Рисунок В.1 – Диаграмма деятельности

Приложение Г
(не обязательно)
Макет сайта

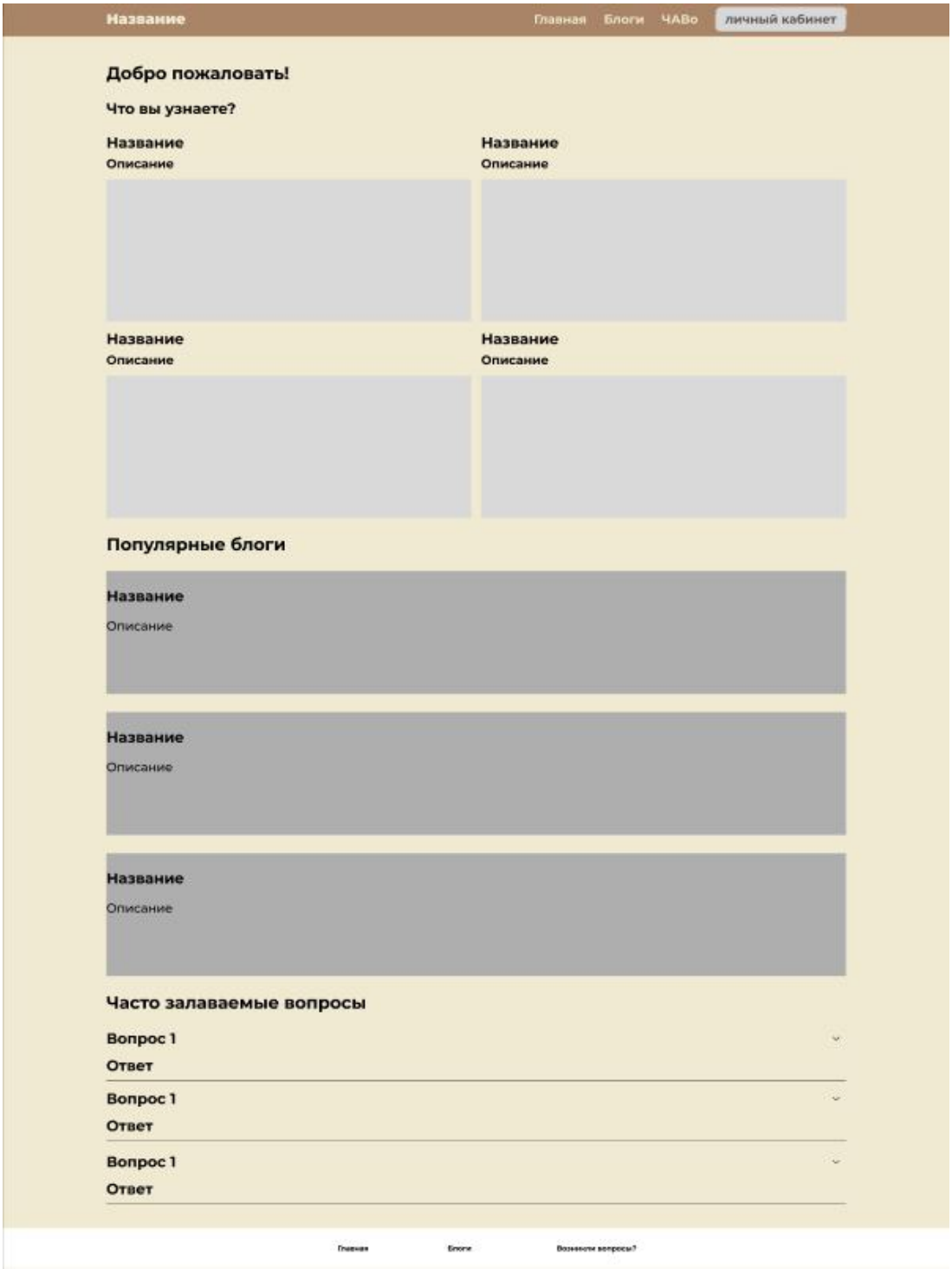


Рисунок Г.1 – Главная страница

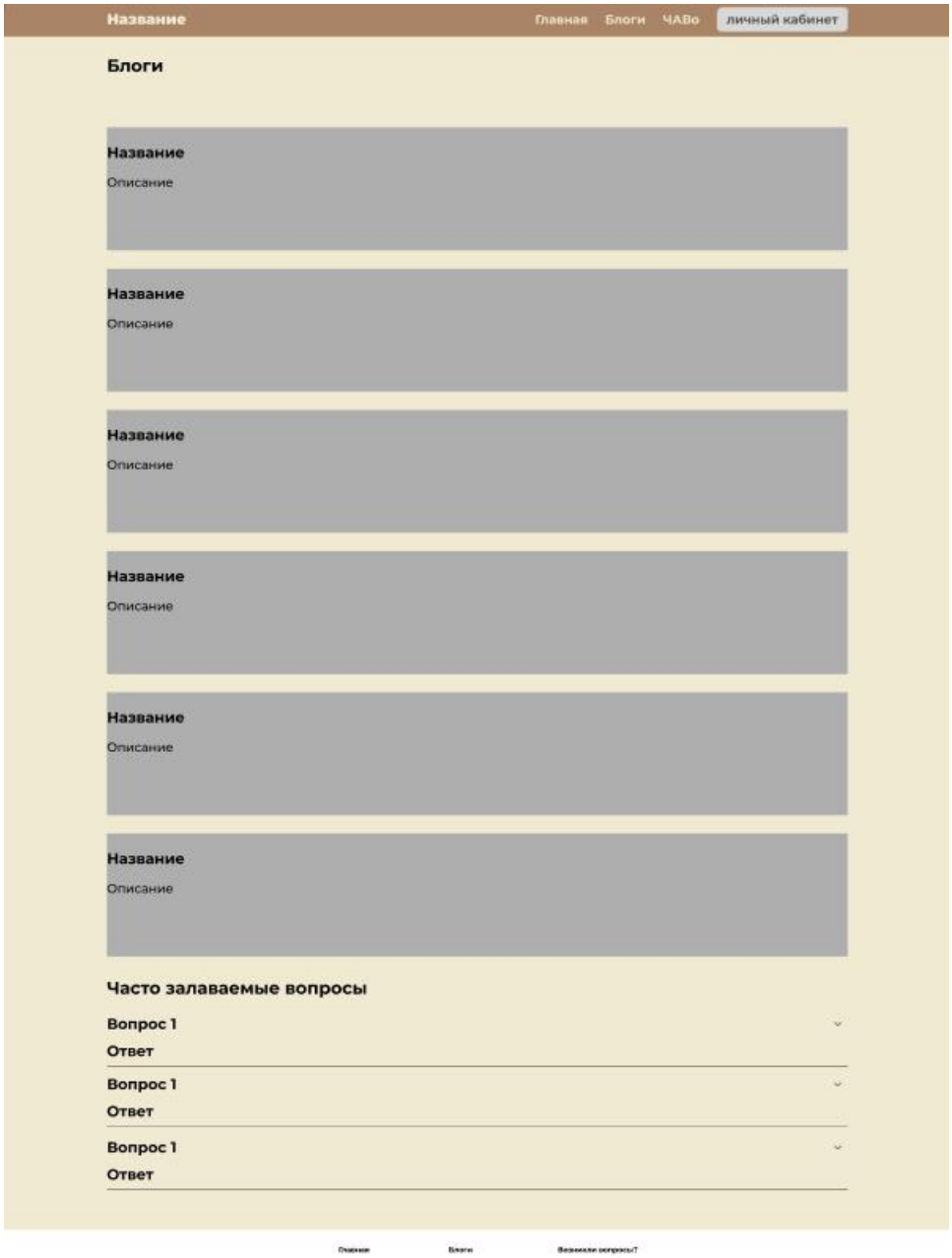


Рисунок Г.2 – Страница блогов

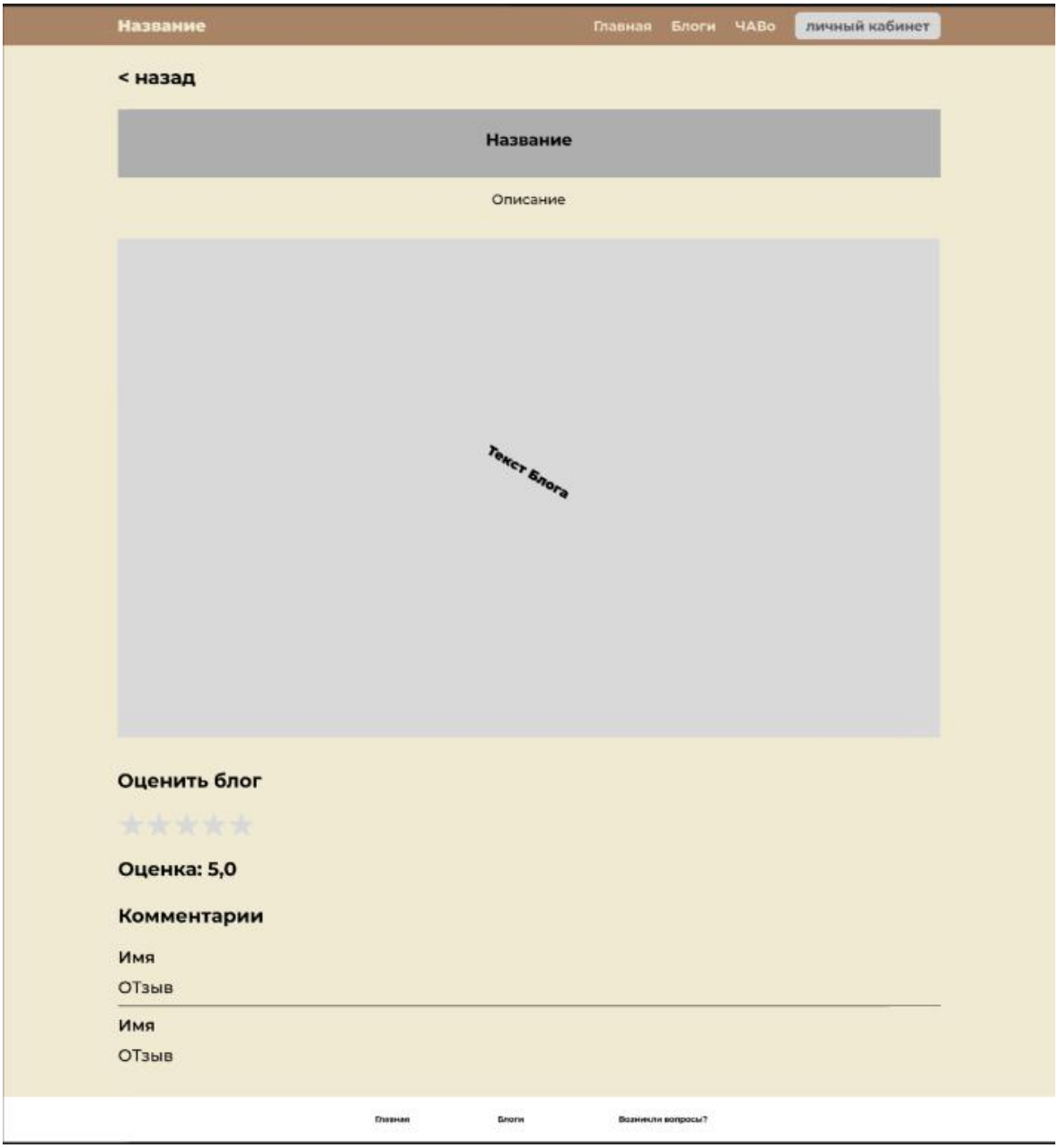


Рисунок Г.2 – Страница блога

Приложение Д
(не обязательно)

Листинг программы

```
// Регистрация
app.post('/api/register', (req, res) => {
    const { username, password } = req.body;
    db.run('INSERT INTO users (username, password) VALUES (?, ?)', [username, password], function(err) {
        if (err) return res.json({ success: false, message: 'Ошибка регистрации' });
        res.json({ success: true, message: 'Новый пользователь зарегистрирован' });
    });
});

// Вход
app.post('/api/login', (req, res) => {
    const { username, password } = req.body;
    db.get('SELECT id, username FROM users WHERE username = ? AND password = ?', [username, password], (err, user) => {
        if (err || !user) return res.json({ success: false, message: 'Неверные учетные данные' });
        req.session.user = user;
        res.json({ success: true, message: 'Добро пожаловать!' });
    });
});
```

Рисунок Д.1 – Код для входа и регистрации (файл server.js)

```
// Добавление комментария
app.post('/api/comments', (req, res) => {
    if (!req.session.user) return res.json({ success: false, message: 'Необходимо войти в аккаунт' });

    const { comment } = req.body;
    db.run('INSERT INTO comments (user_id, comment) VALUES (?, ?)', [req.session.user.id, comment], function(err) {
        if (err) return res.json({ success: false, message: 'Ошибка при добавлении комментария' });
        res.json({ success: true, message: 'Спасибо за ваш отзыв!' });
    });
});

// Получение всех комментариев
app.get('/api/comments', (req, res) => {
    db.all('SELECT comments.comment, users.username FROM comments JOIN users ON comments.user_id = users.id', [], (err, rows) => {
        if (err) return res.json({ success: false, message: 'Ошибка загрузки комментариев' });
        res.json(rows);
    });
});
```

Рисунок Д.2 – Код для комментариев (файл server.js)

```
app.post('/api/blogs', (req, res) => {
  const { title, description, content } = req.body;

  db.run('INSERT INTO blogs (title, description, content) VALUES (?, ?, ?)', [title, description, content], function(err) {
    if (err) return res.status(500).json({ error: err.message });

    const blogId = this.lastID;
    const dbDir = path.join(__dirname, 'db');

    // Создаем папку, если ее нет
    if (!fs.existsSync(dbDir)) {
      fs.mkdirSync(dbDir);
    }

    const blogDbPath = path.join(dbDir, `blog_${blogId}.db`);
    const blogDb = new sqlite3.Database(blogDbPath, (err) => {
      if (err) {
        console.error('Ошибка при создании блога:', err);
        return res.status(500).json({ error: 'Не удалось создать базу для блога' });
      }

      blogDb.serialize(() => {
        blogDb.run('CREATE TABLE IF NOT EXISTS ratings (id INTEGER PRIMARY KEY AUTOINCREMENT, value INTEGER)');
        blogDb.run('CREATE TABLE IF NOT EXISTS comments (id INTEGER PRIMARY KEY AUTOINCREMENT, user_id INTEGER, text TEXT)');
      });

      res.json({ success: true, id: blogId });
    });
  });
});
```

Рисунок Д.3 – Код для создания блога (файл server.js)

```
app.put('/api/blogs/:id', (req, res) => {
  const { id } = req.params;
  const { title, description, content} = req.body;

  if (title !== undefined)
    db.run('UPDATE blogs SET title = ? WHERE id = ?', [title, id]);

  if (content !== undefined)
    db.run('UPDATE blogs SET content = ? WHERE id = ?', [content, id]);

  if (description !== undefined)
    db.run('UPDATE blogs SET description = ? WHERE id = ?', [description, id]);

  res.json({ success: true });
});


app.delete('/api/blogs/:id', (req, res) => {
  const { id } = req.params;
  db.run('DELETE FROM blogs WHERE id = ?', id, err => {
    if (err) return res.status(500).json({ error: err.message });


      res.json({ success: true });
    });      You, 3 hours ago • Добавлена админ панель, обновленны некторые ф…
  });
```

Рисунок Д.4 – Код для обновления и удаления блога (файл server.js)

```
app.post('/api/blogs/:id/ratings', (req, res) => {
  const blogId = req.params.id;
  const rating = req.body.rating;
  const blogDb = new sqlite3.Database(`db/blog_${blogId}.db`);
  blogDb.run('INSERT INTO ratings (value) VALUES (?)', [rating], function(err) {
    if (err) return res.status(500).json({ error: err.message });
    res.json({ success: true });
  });
});

app.get('/api/blogs/:id/average_rating', (req, res) => {
  const blogId = req.params.id;
  const blogDb = new sqlite3.Database(`db/blog_${blogId}.db`);
  blogDb.get('SELECT AVG(value) as average_rating FROM ratings', [], (err, row) => {
    if (err) return res.status(500).json({ error: err.message });
    res.json(row);
  });
});
```

Рисунок Д.5 – Код для добавления и обновления рейтинга (файл server.js)

```
async function register() {
    const email = document.getElementById('email').value;
    const username = document.getElementById('username').value;
    const password = document.getElementById('password').value;

    const res = await fetch('/api/register', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ email, username, password })
    });

    const text = await res.text();  // Получаем текст вместо JSON
    try {
        const data = JSON.parse(text);  // Попробуем распарсить текст как JSON
        alert(data.message);
        if (data.success) window.location.href = '/login';
    } catch (e) {
        console.error('Ошибка парсинга JSON:', text);
    }
}       You, last month • Добавлен личный кабинет
```

Рисунок Д.6 – Код для регистрации пользователя (файл reg.js)

```
async function login() {       You, last month • Добавлен личный
    const username = document.getElementById('username').value;
    const password = document.getElementById('password').value;

    const res = await fetch('/api/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username, password })
    });

    const data = await res.json();
    alert(data.message);
    if (data.success) window.location.href = '/profile';
}
```

Рисунок Д.7 – Код для входа пользователя (файл log.js)

```
async function checkSession() {      You, last month • Добавлен личный кабинет …
    const res = await fetch('/api/session');
    const data = await res.json();
    if (!data.loggedIn) {
        window.location.href = '/login';
    }
}

async function sendComment() {
    const comment = document.getElementById('comment').value;
    const res = await fetch('/api/comments', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ comment })
    });
    alert((await res.json()).message);
    loadComments();
}

async function loadComments() {
    const res = await fetch('/api/comments');
    const comments = await res.json();
    document.getElementById('comments').innerHTML = comments.map(c => `<p><b>${c.username}:</b> ${c.comment}</p>`).join('');
}

document.addEventListener('DOMContentLoaded', () => {
    checkSession();
    loadComments();
});
```

Рисунок Д.8 – Код для добавления комментария (файл comment.js)

```
async function loadBlog() {
    const id = new URLSearchParams(window.location.search).get('id');
    const res = await fetch(`/api/blogs/${id}`);
    const blog = await res.json();

    document.getElementById('blog').innerHTML = `
    <a href="bloglist.html"><button>Назад</button></a>
      <h1>${blog.title}</h1>
      <p><i>${blog.description}</i></p>
      <div>${blog.content}</div>
      <h2>Рейтинг</h2>
  <div class="rating">
    <input type="hidden" id="star1_hidden" value="1">
    <img src="IMG-TEST/fi-rr-star.png" onmouseover="change(this.id)" id="star1" class="rating-star">

    <input type="hidden" id="star2_hidden" value="2">
    <img src="IMG-TEST/fi-rr-star.png" onmouseover="change(this.id)" id="star2" class="rating-star">

    <input type="hidden" id="star3_hidden" value="3">
    <img src="IMG-TEST/fi-rr-star.png" onmouseover="change(this.id)" id="star3" class="rating-star">

    <input type="hidden" id="star4_hidden" value="4">
    <img src="IMG-TEST/fi-rr-star.png" onmouseover="change(this.id)" id="star4" class="rating-star">

    <input type="hidden" id="star5_hidden" value="5">
    <img src="IMG-TEST/fi-rr-star.png" onmouseover="change(this.id)" id="star5" class="rating-star">
</div>

<input type="hidden" id="ratingValue" value="0">
<p>Средняя оценка: <span id="averageRating">0</span>⭐</p>
<button onclick="saveRating()">Сохранить оценку</button>
<hr>
<h2>Комментарии</h2>
<textarea id="comment" placeholder="Оставьте комментарий"></textarea><br>
<button onclick="sendComment()">Отправить</button>
<div id="comments"></div>
    `;}loadBlog();      You, 1 minute ago • Uncommitted changes
```

Рисунок Д.9 – Код для отображения блогов (файл blog.js)

```
async function fetchBlogs() {
  const res = await fetch('/api/blogs');
  const blogs = await res.json();
  const container = document.getElementById('blogs');
  container.innerHTML = '';

  blogs.forEach(blog => {
    const div = document.createElement('div');
    div.className = 'blog-card';
    div.innerHTML = `
      <h3>${blog.title}</h3>
      <p><b>Описание:</b> ${blog.description || ''}</p>
      <button onclick="openBlog(${blog.id})">Открыть</button>
      <button onclick="openEdit(${blog.id}, \`${blog.title}\`, \`${blog.description || ''}\`, \`${blog.content.replace(/`/g, '\\`')}\`)">Изменить</button>
      <button onclick="deleteBlog(${blog.id})">Удалить</button>
    `;
    container.appendChild(div);
  });
}
```

Рисунок Д.10 – Код для отображения блогов в админ-панели (файл admin.js)

```
async function addBlog() {
  const title = document.getElementById('title').value;
  const description = document.getElementById('description').value;
  const content = document.getElementById('content').value;

  await fetch('/api/blogs', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ title, content, description })
  });

  document.getElementById('title').value = '';
  document.getElementById('description').value = '';
  document.getElementById('content').value = '';
  fetchBlogs();
}

function openEdit(id, title, description, content) {
  editingId = id;
  document.getElementById('editTitle').value = title;
  document.getElementById('editDescription').value = description;
  document.getElementById('editContent').value = content;
  document.getElementById('editModal').style.display = 'block';
}

async function saveEdit() {
  const title = document.getElementById('editTitle').value;
  const description = document.getElementById('editDescription').value;
  const content = document.getElementById('editContent').value;

  await fetch(`/api/blogs/${editingId}`, {
    method: 'PUT',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ title, description, content })
  });

  closeEdit();
  fetchBlogs();
}
```

Рисунок Д.11 – Код для добавления и изменеия блогов (файл admin.js)

```
  function closeEdit() {
    editingId = null;
    document.getElementById('editModal').style.display = 'none';
  }

  function deleteBlog(id) {
if (confirm('Вы уверены, что хотите удалить этот блог? Это действие необратимо.')) {
  fetch(`/api/blogs/${id}`, { method: 'DELETE' })      You, 4 hours ago • Добавлена а
    .then(res => res.json())
    .then(data => {
      if (data.success) {
        alert('Блог удален');
        location.reload();
      }
    });
  }
}
```

Рисунок Д.12 – Код для удаления блогов (файл admin.js)

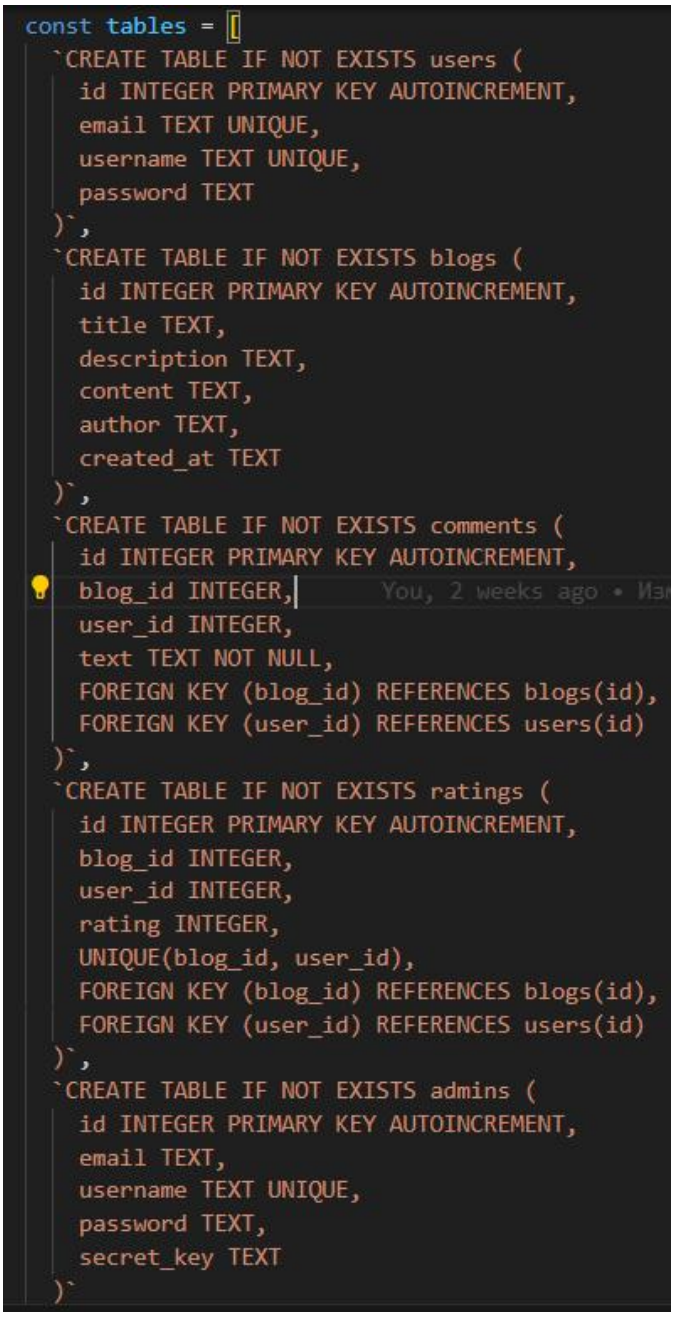
```
async function checkSession() {
    const res = await fetch('/api/session');
    const data = await res.json();
    if (!data.loggedIn) {
        window.location.href = '/login';
        return;
    }
    document.getElementById('welcome').innerText = `Привет, ${data.user.username}`;
}

async function logout() {
    await fetch('/api/logout', { method: 'POST' });
    window.location.href = '/login';
}

document.addEventListener('DOMContentLoaded', checkSession);      You, last month • 
```

Рисунок Д.13 – Код для личного кабинета (файл account.js)

Приложение А
(обязательно)
Код реализации базы данных



```
const tables = [
  `CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    email TEXT UNIQUE,
    username TEXT UNIQUE,
    password TEXT
  )`,
  `CREATE TABLE IF NOT EXISTS blogs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT,
    description TEXT,
    content TEXT,
    author TEXT,
    created_at TEXT
  )`,
  `CREATE TABLE IF NOT EXISTS comments (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    blog_id INTEGER,        You, 2 weeks ago • Из
    user_id INTEGER,
    text TEXT NOT NULL,
    FOREIGN KEY (blog_id) REFERENCES blogs(id),
    FOREIGN KEY (user_id) REFERENCES users(id)
  )`,
  `CREATE TABLE IF NOT EXISTS ratings (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    blog_id INTEGER,
    user_id INTEGER,
    rating INTEGER,
    UNIQUE(blog_id, user_id),
    FOREIGN KEY (blog_id) REFERENCES blogs(id),
    FOREIGN KEY (user_id) REFERENCES users(id)
  )`,
  `CREATE TABLE IF NOT EXISTS admins (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    email TEXT,
    username TEXT UNIQUE,
    password TEXT,
    secret_key TEXT
  )`
```

Рисунок А.1 – Код реализации базы данных

Приложение Б
(обязательно)
Программный код до / после рефакторинга

```
app.get('/register', (req, res) => res.sendFile(path.join(__dirname, 'public/register.html')));
app.get('/login', (req, res) => res.sendFile(path.join(__dirname, 'public/login.html')));
app.get('/profile', (req, res) => res.sendFile(path.join(__dirname, 'public/profile.html')));
app.get('/comments', (req, res) => res.sendFile(path.join(__dirname, 'public/bloglist.html')));
app.get('/admin', (req, res) => res.sendFile(path.join(__dirname, 'public/admin.html')));
```

Рисунок Б.1 – Ссылки на файлы до рефакторинга

```
['register', 'login', 'profile', 'comments', 'admin'].forEach(route => {
  app.get(`/${route}`, (req, res) => res.sendFile(path.join(__dirname, `public/${route}.html`)));
});
```

Рисунок Б.2 – Ссылки на файлы после рефакторинга

```
db.serialize(() => {

    //База данных пользователей
 db.run(`CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    email TEXT UNIQUE,
    username TEXT UNIQUE,
    password TEXT
)`);

db.run(`CREATE TABLE IF NOT EXISTS blogs (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  title TEXT,
  description TEXT,
  content TEXT,
  created_at TEXT
  )`);

db.run(`CREATE TABLE IF NOT EXISTS comments (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    blog_id INTEGER,
    user_id INTEGER,
    text TEXT NOT NULL,
    FOREIGN KEY (blog_id) REFERENCES blogs(id),
    FOREIGN KEY (user_id) REFERENCES users(id)
);`);
```

Рисунок Б.3 – Код таблиц до рефакторинга

```
// Создаем БД
const tables = [
  `CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    email TEXT UNIQUE,
    username TEXT UNIQUE,
    password TEXT
  )`,
  `CREATE TABLE IF NOT EXISTS blogs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT,
    description TEXT,
    content TEXT,
    author TEXT,
    created_at TEXT
  )`,
```

Рисунок Б.4 – Код таблиц после рефакторинга

```
// Загрузка комментариев
async function loadComments() {
    try {
        const res = await fetch(`/api/blogs/${blogId}/comments`);
        const comments = await res.json();
        const container = document.getElementById('comments');
```

Рисунок Б.5 – Код с fetch и res.json

```
// Универсальный fetch с обработкой ошибок
const fetchJSON = async (url, options = {}) => {
    try {
        const res = await fetch(url, options);
        if (!res.ok) throw new Error(await res.text());
        return await res.json();
    } catch (err) {
        console.error(`Ошибка запроса к ${url}:`, err);
        throw err;
    }
};
```

Рисунок Б.6 – Код с fetch и res.json после рефакторинга

```
// Проверка логина
const checkLogin = async () => {
    const res = await fetch('/api/session');
    const data = await res.json();
    return data.loggedIn;
};
```

Рисунок Б.7 – Проверка логина до рефакторинга

```
const checkLogin = async () => {
    const data = await fetchJSON('/api/session');
    return data.loggedIn;
};
```

Рисунок Б.8 – Проверка логина после рефакторинга

```
document.addEventListener('DOMContentLoaded', () => {
    if (!blogId) {
        console.error("blogId не найден в URL");
        return;
    }
    fetchAverageRating();
    loadComments();
    loadUserRating();
});
```

Рисунок Б.9 – DOM-дерево до рефакторинга

```
document.addEventListener('DOMContentLoaded', async () => {
    if (!blogId) {
        console.error("blogId не найден в URL");
        return;
    }

    await fetchAverageRating();
    await loadComments();
    await loadUserRating();
});
```

Рисунок Б.10 – DOM-дерево после рефакторинга

Приложение В
(обязательно)
Контрольный пример

Рисунок В.1 – главная страница

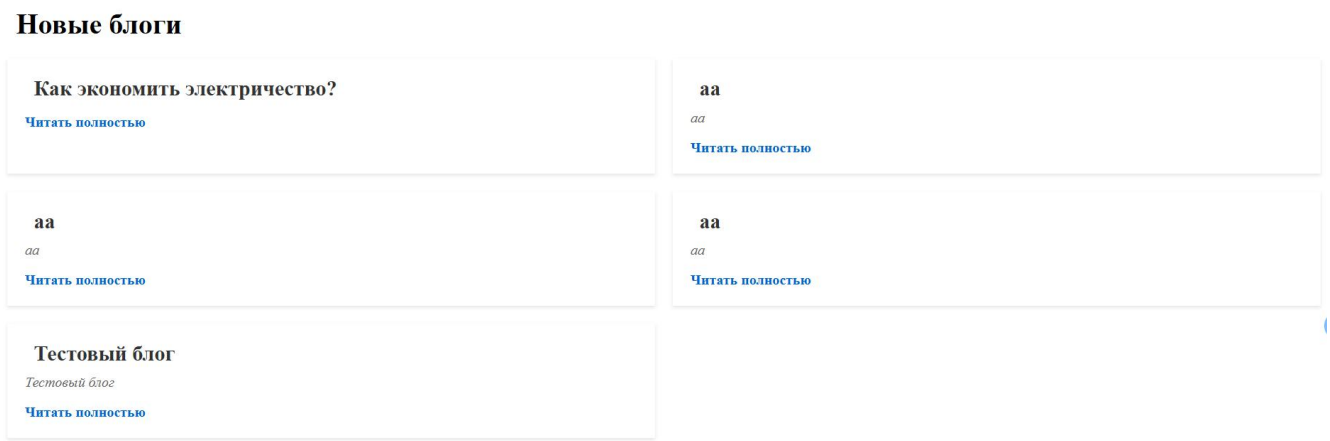


Рисунок В.2 – блоги

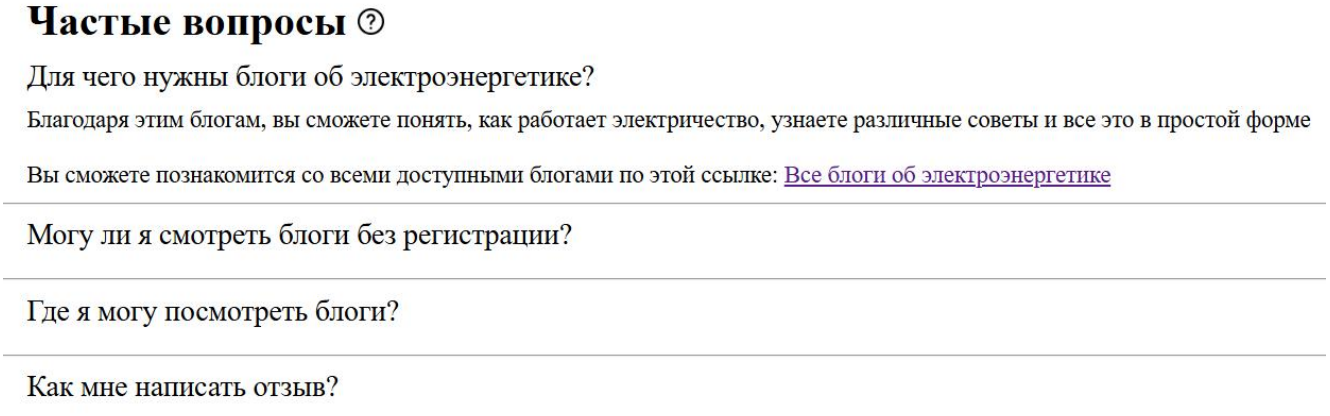


Рисунок В.2 – Блок с частыми вопросами

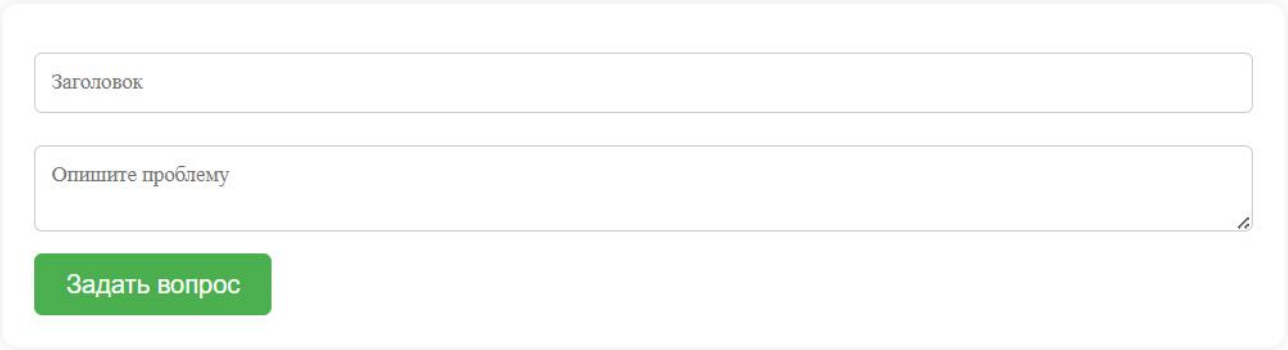


Рисунок В.3 – Блок для вопроса

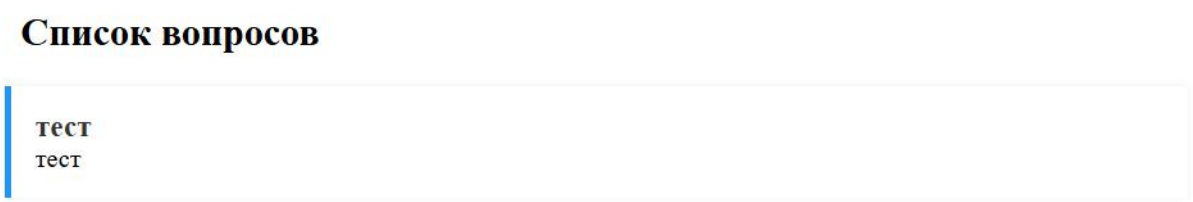


Рисунок В.4 – Блок с вопросами

Рисунок В.5 – Открытый блог

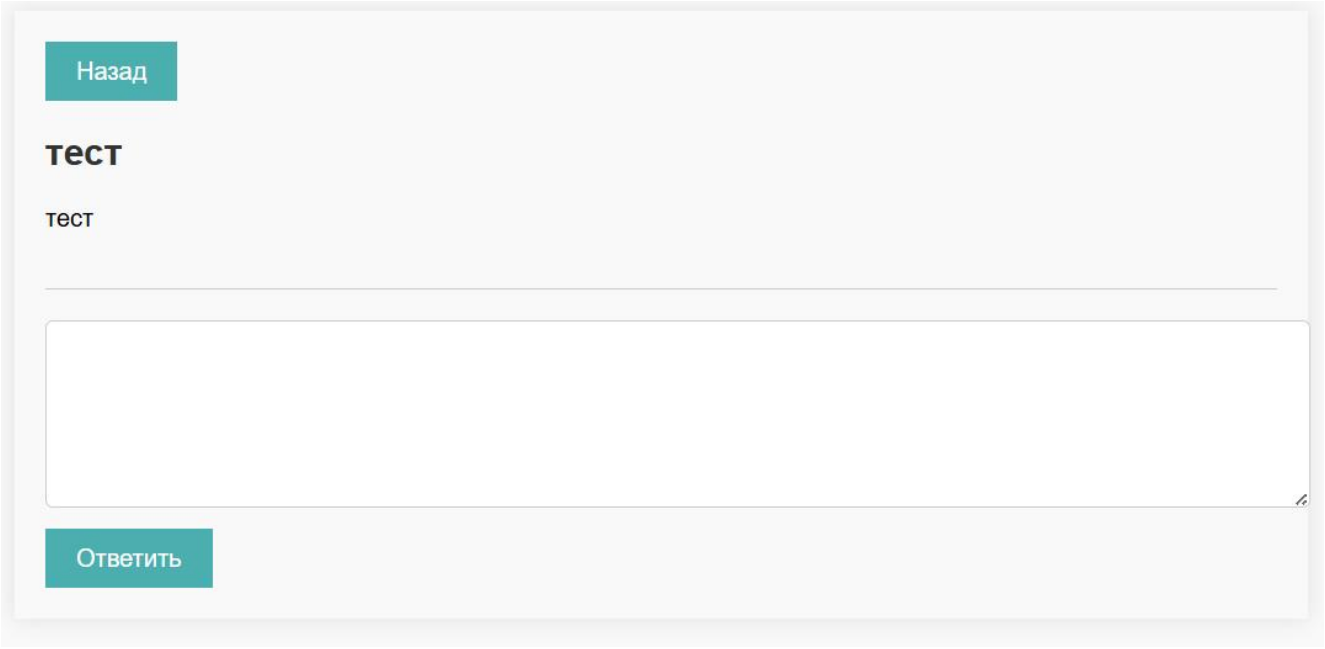


Рисунок В.6 – Открытый вопрос

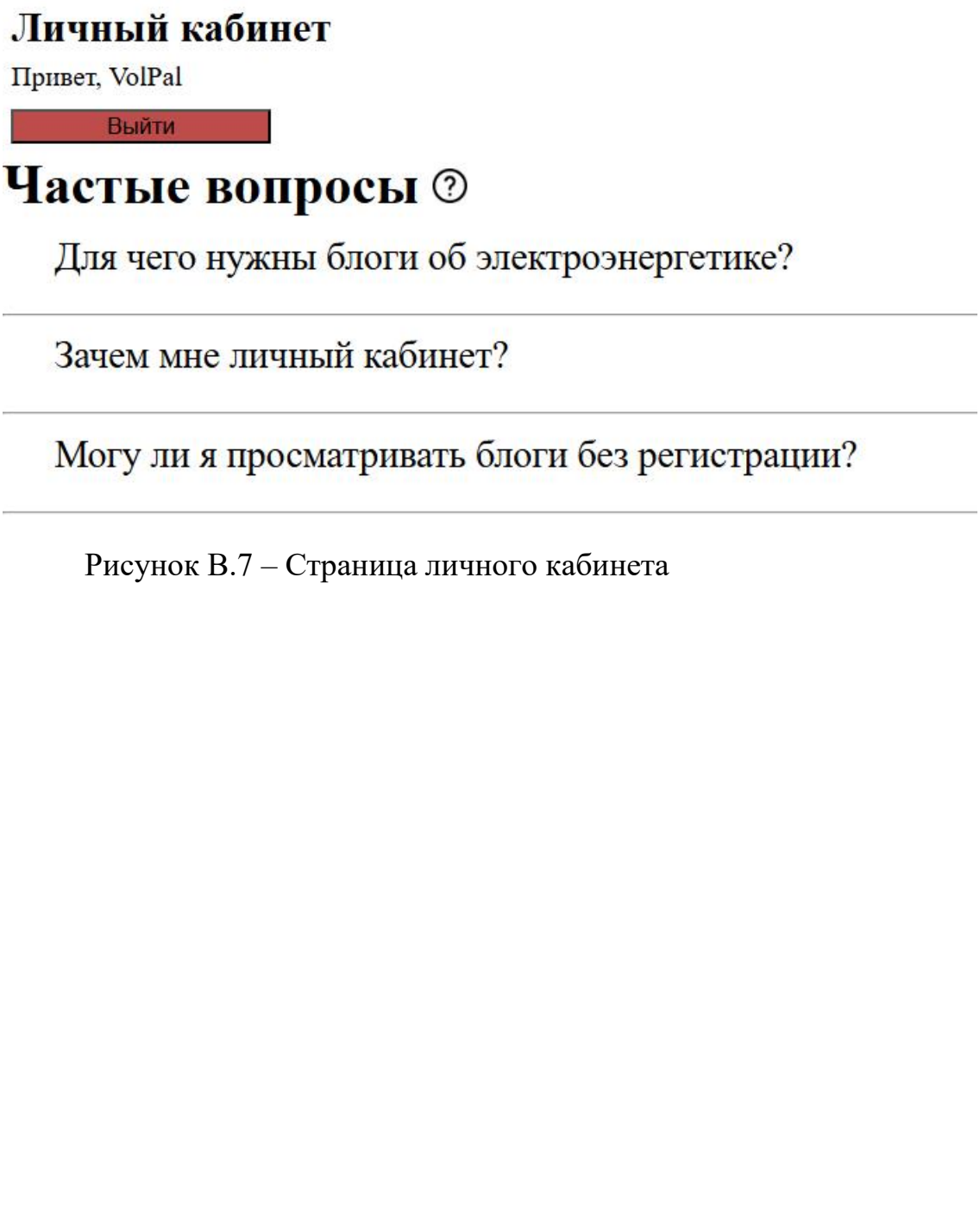


Рисунок В.7 – Страница личного кабинета

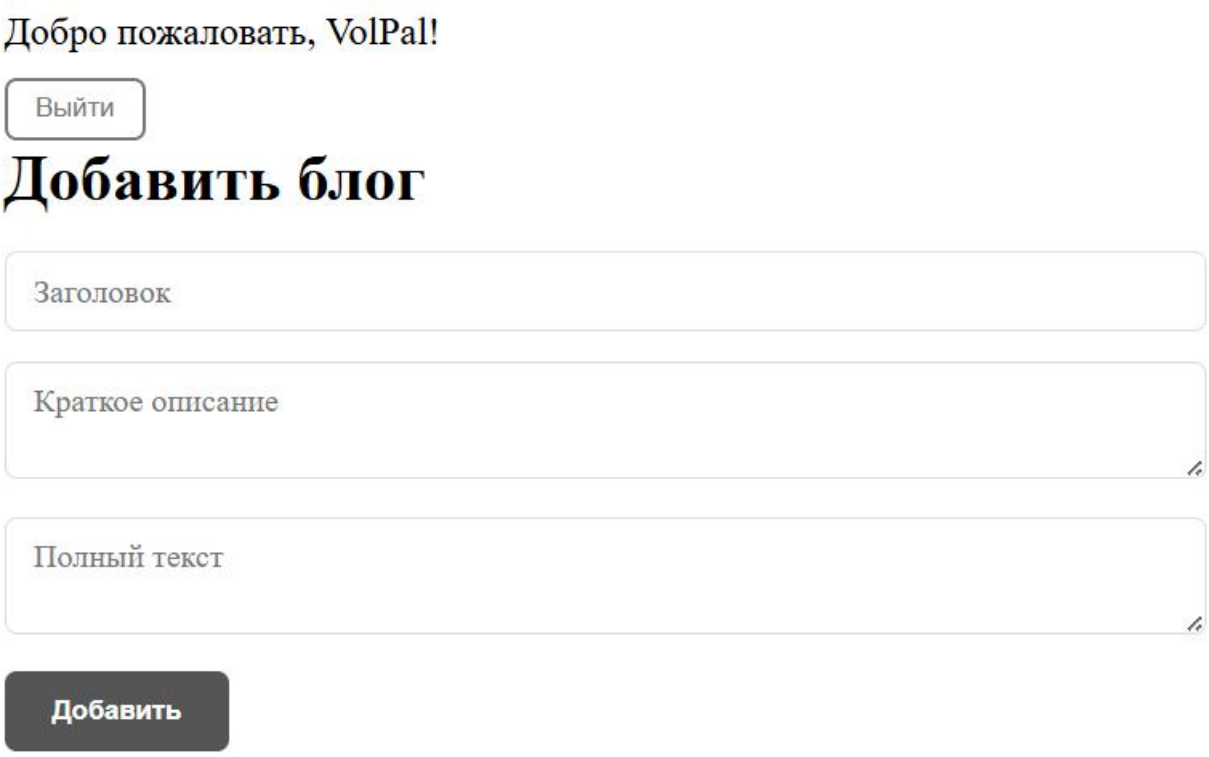


Рисунок В.8 – Страница добавления блога для администратора

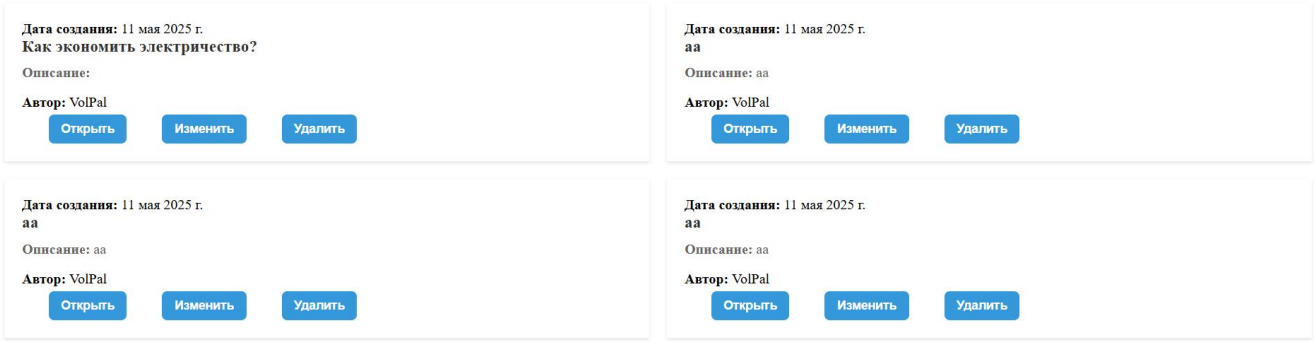


Рисунок В.9 – Страница добавленных блогов администраторами