МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра информационных технологий в атомной энергетике

ОТЧЕТ ПО ПРОЕКТУ

по дисциплине «Программирование на языке Котлин»

Студент группы ИКБО-25-22 Ракитин В.А.

_____

(подпись студента)

Руководитель проектной работы Золотухин С.А.

_____

(подпись руководителя)

Работа представлена «____»_____ 2024 г.

Допущен к работе «___» _____ 2024 г.

Москва 2024

Оглавление

ВВЕДЕНИЕ

В современном цифровом пространстве обмен мнениями и публикация актуальных материалов играют ключевую роль в формировании общественного мнения и обсуждении важных вопросов. Однако многие существующие платформы для обсуждений и публикации статей устарели, не предлагая пользователям удобного интерфейса, возможности быстро находить актуальную информацию или участвовать в интерактивных дискуссиях. В ответ на эти вызовы мы разработали **веб-сервис для создания обсуждений и публикации статей**, который решает проблему нехватки качественных и современных платформ.

Существует распространённый стереотип, что Kotlin в первую очередь предназначен для разработки под Android. Однако это мнение является устаревшим! Язык Kotlin поддерживает множество платформ, включая JVM, JavaScript и Native, что открывает широкие возможности для кроссплатформенной разработки. Благодаря данной мультиплатформенной поддержке разработчики могут не только писать приложения в единой кодовой базе, но и эффективно переиспользовать код, что значительно ускоряет процесс разработки и снижает количество ошибок.

Проект представляет собой веб-сервис для публикации статей и создания обсуждений, который призван решить проблемы актуальности существующих платформ. Наша цель - предоставить пользователям удобный инструмент для обмена информацией и мнениями по важным вопросам.

Наш проект использует **Kotlin** — современный язык программирования, который благодаря своей кроссплатформенной природе предоставляет разработчикам возможность создавать приложения, работающие как на сервере, так и на клиенте. С помощью фреймворка **Ktor** для бэкенда и **Kotlin/JS** для фронтенда, мы можем эффективно построить гибкое и быстрое приложение с богатым пользовательским интерфейсом. При этом использование Kotlin значительно ускоряет процесс разработки и уменьшает количество ошибок, что является важным преимуществом для такого масштабного проекта.

Цели и задачи проекта

Основной целью проекта является создание платформы для обмена информацией и публикации актуальных материалов, где пользователи могут не только читать статьи, но и активно участвовать в обсуждениях. Мы стремимся предоставить простой и интуитивно понятный интерфейс, который позволяет пользователям легко создавать, редактировать и публиковать статьи, а также оставлять комментарии и вступать в обсуждения.

Для достижения поставленных целей проект включает в себя следующие задачи:

1. **Разработка пользовательского интерфейса (UI/UX)**: создание удобного и современного интерфейса для взаимодействия с контентом. Интерфейс должен быть адаптивным и доступным на различных устройствах — от десктопов до мобильных телефонов.
2. **Создание функционала для публикации и редактирования статей**: разработка системы для создания и редактирования текстовых публикаций, добавления медиа-ресурсов, таких как изображения и видео, а также возможность группировки контента с помощью категорий и тегов.
3. **Система обсуждений и комментариев**: обеспечение возможности пользователей обсуждать статьи, оставлять комментарии, а также вести диалоги в рамках обсуждений. Включение функции уведомлений и модерации контента для предотвращения появления нежелательных сообщений.
4. **Интеграция с REST API**: разработка API для взаимодействия с бэкендом, что позволит управлять статьями, комментариями и пользователями. Также будет реализована система аутентификации и авторизации пользователей.
5. **Обеспечение безопасности и защиты данных**: внедрение надежных механизмов защиты личных данных пользователей, безопасное хранение паролей и шифрование чувствительной информации. Разработка системы антиспама и защита от вредоносной активности.
6. **Деплой и настройка сервиса**: реализация стратегии развертывания сервиса на облачной платформе для обеспечения его доступности. Также планируется настройка автоматизации процессов развертывания и тестирования (CI/CD).
7. **Пользовательская документация и поддержка**: создание документации для пользователей, включая руководство по использованию платформы и раздел часто задаваемых вопросов. Также предусмотрена система поддержки, которая позволит пользователям обращаться за помощью.

Особенности проекта

1. **Модернизация платформы**: Наш сервис предоставляет современную альтернативу устаревшим платформам, имеющим громоздкий и неудобный

интерфейс. Мы делаем акцент на скорости работы и удобстве для конечного пользователя.

2. **Интерактивность и вовлеченность**: Благодаря системе комментариев и обсуждений, пользователи могут активно взаимодействовать друг с другом, обсуждать различные вопросы, делиться мнениями и получать уведомления о новых комментариях.
3. **Кроссплатформенность и масштабируемость**: Проект написан с использованием Kotlin, что позволяет обеспечить поддержку различных платформ и устройств. Это открывает возможности для дальнейшей масштабируемости и добавления новых функций.
4. **Безопасность и конфиденциальность**: Мы заботимся о безопасности пользовательских данных и обеспечиваем надежную защиту от внешних угроз, используя современные методы защиты информации и безопасные протоколы связи.

В рамках данного проекта будут изучены и применены современные технологии, такие как **Kotlin/JS**, **React**, **Ktor**, **REST API**, а также лучшие практики безопасности и оптимизации. Мы уверены, что наш сервис заполнит нишу, предоставив пользователям платформу для активных обсуждений и публикации материалов, соответствующую всем требованиям современности.

Наша команда:

1. Project Manager, Frontend Developer, UI/UX Designer - **Петрушенко Александр**
2. Бекенд-разработчик, Специалист по DevOps – **Костин Михаил**
3. Тестировщик, Технический писатель **- Ракитин Владимир**

1. ПЛАНИРОВАНИЕ И ПОДГОТОВКА

1.1 Техническое задание

1.1.1 Общие сведения

1.1 Полное наименование системы: Система для создания открытых обсуждений и статей. (далее – Система).

1.2 Организации, участвующие в создании Системы:
- Заказчик;
- Исполнитель.

1.3 Основанием для создания Системы: работы по созданию Системы проводятся в рамках Указания о начале работ и создании РГ по Проекту.

1.4 Плановые сроки начала и окончания работ определяются согласно Плана Проекта.

1.1.2 Цели и назначение автоматизированной системы

Целью создания Системы является создание платформы для открытых обсуждений по определенным категориям, чтобы способствовать сотрудничеству, обмену знаниями и информированному принятию решений. Это достигается за счет:

- Реализации панели создания документа для будущего обсуждения;
- Реализации панели конструктора заполнения документов и форм для хранения данных с возможностью заполнения всех необходимых полей по заданным критериям;
- Реализации хранения истории создания/изменения документов и форм хранения данных в разрезе пользователь/дата/внесенные изменения;
- Реализации разграничения прав доступа пользователей.

1.1.3 Характеристика объектов автоматизации

Объектом автоматизации Системы является деятельность сотрудников Отдела, связанная с информационным взаимодействием посредством Системы.

Деятельность сотрудников Отдела в Системе предполагает следующие взаимосвязанные действия:

- Просмотр и управление информацией;
- Участие в профессиональных сообществах;
- Работа с документами;
- Доступ к информационному блоку, посвященному публикации новостей, мероприятий, фото и видео материалов;
- Просмотр и публикация информации;
- Автоматизация и оперативный доступ к личному кабинету;
- Удобный и быстрый поиск информации.

1.1.4 Требования к системе

1.1.4.1 Перечень подсистем

Система должна состоять из следующих функциональных подсистем и модулей, обеспечивающих работу автоматизированных рабочих мест (АРМ) пользователей:

- Личный кабинет
  - Личный кабинет
  - Личный кабинет администратора
  - Сброс/восстановление пароля
  - Область публикации обсуждений
    - Публикация комментариев
  - Статистика профиля
  - Область с инструкциями пользования системы
    - Методические указания
  - Центр уведомлений
    - Рассылка опросов по улучшению Системы
    - Публикация информаций
  - Обратная связь

1.1.4.2 Требования к способам и средствам связи для информационного обмена между компонентами Системы

Компоненты Системы должны поддерживать информационный обмен посредством вычислительных сетей, объединенных посредством сертифицированных средств шифрования и функционирующих на базе протокола TCP/IP. Основными протоколами при передаче информации должны являться HTTP и HTTPS.

Взаимодействие пользователей с Системой происходит через веб-интерфейс и обращение к веб-серверу с помощью приложения Google Chrome версии 129.0.6668.71 и выше, по протоколу HTTP/HTTPS.

Взаимодействие Системы со смежными системами осуществляется с использованием дополнительно разработанных программных компонент – подробнее см. документ «Концепция по интеграции» и «Проектные решения по интеграции».

1.1.4.3 Требования к режимам функционирования Системы

Система должна функционировать в следующих режимах:

– Штатном;
– Профилактическом.

К указанным режимам предъявляются следующие требования:
– К штатному режиму:
  - Обеспечение функциональности приведенной в настоящем ТЗ;
– К профилактическому режиму:
  - Обеспечение проведения необходимых работ для восстановления штатного режима.

1.1.4.4 Требования к численности и квалификации персонала Системы и режиму его работы

1.1.4.4.1 Требования к составу персонала Системы

В состав персонала, необходимого для обеспечения эксплуатации, должны входить:

- Системный администратор;
- Администратор информационной безопасности;
- Прикладной администратор.

Допускается выполнение одним должностным лицом функций нескольких типов пользователей.

1.1.4.4.2 Требования к квалификации персонала

К квалификации персонала Системы предъявляются следующие требования, функциональные обязанности:

К Системному администратору:

Системный администратор должен обеспечивать функционирование в штатном режиме аппаратных и программных средств. В его функциональные обязанности должны входить:

- Настройка, диагностирование, оперативный контроль и оптимизация загрузки аппаратных средств;
- Контроль целостности системных баз данных;
- Резервное копирование наборов системных баз данных;
- Восстановление данных в системных базах данных;
- Регистрация пользователей;
- Координация деятельности администраторов информационных ресурсов и администраторов информационной безопасности;
- Обеспечение качества общесистемных сервисов (поиск, персонализация, сбор статистической информации и т.д.);
- Обобщение потребности пользователей в части общесистемных сервисов и выдача рекомендаций по их совершенствованию;
- Оперативный контроль и администрирование Системы, восстановление ее функционирования при неработоспособности;
- Сбор и подготовка статистических данных о качестве предоставляемых сервисов и использовании информационных ресурсов и их предоставление администраторам информационных ресурсов;
- Сопровождение проблем, возникших у пользователей, в части функционирования общесистемных сервисов.

- Реализация политики формирования информационных ресурсов;
- Обеспечение целостности и непротиворечивости информационного ресурса;
- Разработка механизмов изучения информационных потребностей пользователей Системы;
- Анализ качества состояния и использования информационного ресурса;
- Сопровождение проблем, возникших у пользователей, в части функционирования информационных ресурсов.

К Администратору информационной безопасности:

Администратор информационной безопасности должен иметь полномочия по контролю за действиями системных администраторов, администраторов информационных ресурсов и пользователей (без вмешательства в их действия), а также права по настройке, позволяющие задавать полномочия пользователей по доступу к информационным ресурсам и сервисам. В функции администратора информационной безопасности должны входить следующие функции:

- Реализация политики информационной безопасности Системы;
- Классификация пользователей в соответствии с выработанной политикой информационной безопасности;
- Формирование профилей групп пользователей;
- Формирование индивидуальных профилей пользователей;
- Периодический контроль соответствия прав доступа пользователей к информационным ресурсам и сервисам установленному регламенту;

– Анализ журналов регистрации штатных компонентов, а также средств защиты, регистрация попыток несанкционированного доступа и оперативное реагирование на такие попытки;
– Участие в расследовании нештатных ситуаций и принятие решения об имевшем месте несанкционированном доступе;
– Расследование случаев несанкционированного доступа к Системе;
– Администрирование средств защиты информации.

1.1.4.4.3 Требования к режиму работы персонала Системы

Функционирование Системы в целом должно определяться соответствующим законодательством, внутренними приказами и инструкциями. Функции по регламентному обслуживанию и администрированию Системы должны выполняться в нерабочее время. В случае возникновения аварийных ситуаций, ведущих к отказу функционирования Системы, восстановление Системы должно осуществляться в рабочее время.

1.1.4.5 Требования к надежности

Надежность Системы оценивается на основании времени работы Системы в штатном режиме.

Время работы Системы в штатном режиме должно составлять не менее 95 процентов времени в год. Время недоступности Системы не должно превышать показатели, приведенные в таблице 3:

Таблица 3 – Требования к устойчивости

Показатель	**Процент времени работы в штатном режиме**	**Время недоступности Системы, в день**	**Время недоступности Системы, в месяц**	**Время недоступности Системы, в год**
Работа Системы в штатном режиме функционирования	95%	72.00 минуты	36 часов	18,26 дней

1.1.4.5.1 Перечень аварийных ситуаций

Ниже приводится перечень возможных аварийных ситуаций с указанием требований к мероприятиям и средствам восстановления работоспособности Системы:

– Отказ технических средств (оборудования):
  - Отказ электрооборудования;
  - Отказ сервера;
– Отказ программных средств, не входящих в состав Системы;
– Отказ компонентов системы:
  - Сбой базы данных (без потери работоспособности);
– Нарушение канала связи с серверами Системы.
– Ошибка в работе персонала

Перечень возможных фатальных ошибок и реакция программного обеспечения и персонала на них должны быть определены на этапе «Ввода Системы в постоянную

эксплуатацию» и отражены в документах «Руководство пользователя» и «Руководство контент менеджера и модератора Системы»;

- Отказ Сервера

1.1.4.5.2 Требования к надежности технических средств и программного обеспечения.

Надежность Системы в части технического обеспечения должна обеспечиваться:

- Использованием аппаратного обеспечения повышенной отказоустойчивости и его структурным резервированием;
- Защитой аппаратного обеспечения по электропитанию путем использования источников бесперебойного питания;
- Дублированием носителей информационных массивов и организацией процесса резервного копирования данных. При разработке Системы должен быть определен перечень информационных ресурсов, подлежащих резервному копированию, их ожидаемый объем, а также периодичность проведения резервного копирования каждого информационного ресурса.

Назначенные сроки службы технических средств Системы, среднее время работы на отказ не устанавливается и определяется требованиями к техническим средствам.

На стадии ввода в действие Системы должен производиться анализ отказов и неисправностей, а также должны приниматься меры по их предупреждению и устранению.

1.1.4.6 Требования безопасности

Подсистема информационной безопасности должна обеспечивать выполнение следующих функций:

- Идентификация, аутентификация и контроль доступа пользователей к информации;
- Авторизация пользователей;
- Протоколирование и учет обращений;
- Обеспечение целостности программных средств и обрабатываемой информации;
- Резервирование и хранение копий данных.

1.1.5 Состав и содержание работ

Перечень работ, проводимых в рамках настоящего Проекта, приведен в Таблице 1.

Таблица 1 – Перечень работ по созданию Системы

№ Этап	Работы	Срок
Этап 1. Обследование и проектирование	Определение границ прототипа Системы; Обследование инфраструктуры Заказчика методом интервьюирования; Разработка технического задания;	Сентябрь 2024 – октябрь 2024

№ Этап	Работы	Срок
	Проектирование логической и физической архитектуры Системы; Проектирование функциональных модулей прототипа Системы; Разработка программы и методики испытаний прототипа Системы; Разработка функциональных модулей прототипа Системы; Проведение испытаний прототипа Системы;	
Этап 2. Разработка	Проектирование функциональных модулей Системы; Разработка программы и методики испытаний; Разработка функциональных модулей Системы; Разработка эксплуатационной документации; Проведение предварительных испытаний. Ввод опытную эксплуатацию Системы.	Октябрь 2024 – декабрь 2024
Этап 3. Ввод Системы в постоянную эксплуатацию	Анализ результатов опытной эксплуатации Устранение замечаний, выявленных во время опытной эксплуатации Системы; Доработка Системы; Доработка проектной и эксплуатационной документации на Систему; Первоначальная загрузка данных в Систему; Проведение приемочных испытаний; Ввод Системы в постоянную (промышленную) эксплуатацию; Передача исходного кода заказчику.	Декабрь 2024 – декабрь 2024

1.1.6 Порядок разработки и приемки

Порядок организации разработки Системы

Порядок организации разработки Системы должен осуществляться в соответствии с требованиями ГОСТ Р 59793–2021 «Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания».

Перечень документов и исходных данных для разработки Системы

Исходные данные для разработки Системы описаны в Техническом задании на разработку Системы.

Перечень документов, предъявляемых по окончании соответствующих этапов работ

Этап	Перечень документов
Этап 1. Обследование и проектирование	– Детальный план-график работ по проекту; – Техническое задание на разработку Системы; – Программа и методика испытаний прототипа Системы; – Протокол проведения испытаний прототипа Системы.
Этап 2. Разработка	– Программа и методика предварительных испытаний Системы, включая предварительные нагрузочные испытания; – Руководство администратора по настройке и сопровождению Системы; – Руководство пользователя; – Инструкция по резервному копированию и восстановлению Системы после сбоев; – Материалы по обучению пользователей; – Регламент эксплуатации Системы; – Регламент сопровождения Системы; – Протокол проведения предварительных испытаний, включая заключение о производительности Системы.
Этап 3. Ввод Системы в постоянную эксплуатацию	– План устранения замечаний; – Актуализированная проектная и эксплуатационная документация; – Программа и методика приемочных испытаний Системы; – Исходный код;

Требования к гарантийным обязательствам разработчика

Гарантия и техническая поддержка должна осуществляться Исполнителем на срок не менее 12 месяцев.

1.1.7 Требования к подготовке объекта к вводу системы в действие

Виды, состав, объем и методы испытаний Системы

В состав испытаний Системы входят:

- Предварительные испытания (прототип Системы);
- Опытная эксплуатация;
- Приемочные испытания.

Предварительные испытания должны включать проверку:

- Полнота и качество реализуемых функций при штатных, предельных, критических значениях параметров объекта автоматизации и в других условиях функционирования подсистемы, указанных в настоящем документе;
- Выполнения каждого требования, относящегося к интерфейсу подсистемы;
- Работы персонала в диалоговом режиме;
- Комплектности и качества эксплуатационной документации.

Опытная эксплуатация должна включать проверку:

- Полноты и качества реализуемых функций при штатных, предельных, критических значениях параметров объекта автоматизации и в других условиях функционирования подсистемы, указанных в настоящем документе;
- Выполнения каждого требования, относящегося к интерфейсу подсистемы;
- Работы персонала в диалоговом режиме;
- Комплектности и качества эксплуатационной документации.

Приемочные испытания должны включать проверку:

- Полноты и качества реализуемых функций при штатных, предельных, критических значениях параметров объекта автоматизации и в других условиях функционирования подсистемы, указанных в настоящем документе;
- Выполнения каждого требования, относящегося к интерфейсу подсистемы;
- Работы персонала в диалоговом режиме;
- Комплектности и качества эксплуатационной документации.

Перечень участвующих предприятий и организаций, место и сроки проведения

Для проведения приемочных испытаний Системы создается Приемочная комиссия, в состав которой входят представители следующих организаций:

- АО «Концерн Росэнергоатом» (Заказчик);
- ООО «KPR project» (Исполнитель).

Место проведения:

г. Москва, ул. Ордынка Б., 24.

О готовности к проведению приемочных испытаний Исполнитель оповещает Заказчика отдельно

1.1.8 Требования к документированию

Комплект документации предоставляется Заказчику Исполнителем в одном экземпляре в печатном виде, а также в электронном виде на машинных носителях.

В перечень документов, разрабатываемых в рамках Проекта, должны входить:

- Техническое задание;
- Пояснительная записка;
- Программа и методика предварительных испытаний;
- Программа и методика приемо-сдаточных испытаний;
- Инструкция администратора;
- Регламент штатного обслуживания;
- Регламент внештатного обслуживания;
- Паспорт Системы;
- Исходный код.

1.1.9 Источники разработки

ГОСТ 34.201-2020 «Информационные технологии. Комплекс стандартов на автоматизированные системы. Виды, комплектность и обозначение документов при создании автоматизированных систем».

ГОСТ Р 59793–2021 «Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания»

Лекции «Программирование на языке Котлин» РТУ МИРЭА

1.2 Диаграмма вариантов использования

1.2.1 Обоснование сценариев использования

1. Регистрация и авторизация пользователей

Цель: Обеспечить доступ к функционалу приложения только для зарегистрированных пользователей, а также собрать базовую информацию о пользователе (например, имя, электронная почта, предпочтения).

Описание:

- **Регистрация**: Новый пользователь может зарегистрироваться через форму с обязательными полями (имя, email, пароль). Для повышения безопасности пароль должен быть защищен.
- **Авторизация**: Пользователь должен войти в систему с помощью логина и пароля. Важно, чтобы система предлагала функционал восстановления пароля в случае его утери.

Роль в приложении:

- Пользователь становится частью системы и получает доступ к дополнительным функциям (например, созданию статей и обсуждений).

Причина необходимости: Регистрация и авторизация важны для управления контентом и обеспечения безопасности. Система может хранить личную информацию пользователя и его активность (например, какие статьи он публиковал или в каких обсуждениях участвовал).

2. Создание и публикация статей

Цель: Предоставить пользователям возможность делиться своими знаниями и опытом, создавая статьи на платформе.

Описание:

- После успешной авторизации, пользователи могут перейти в раздел для создания статьи. Для этого предусмотрены поля для ввода заголовка, текста статьи и добавления тегов или категорий, чтобы пользователи могли легко находить интересующие их темы.
- Веб-редактор статьи должен поддерживать базовое форматирование (жирный, курсив, список, вставка изображений и ссылок).
- После завершения написания статьи, пользователь может сохранить её как черновик или сразу опубликовать.

Роль в приложении:

- Создание статей — основной функционал платформы, который помогает пользователям делиться знаниями и опытом.
- Статья может стать основой для обсуждения, привлекая других пользователей для комментирования и взаимодействия.

Причина необходимости: Возможность создания и публикации статей помогает платформе стать ценным ресурсом для распространения знаний, улучшает взаимодействие и создает основу для создания контента.

3. Чтение и комментирование статей

Цель: Обеспечить пользователям возможность активно взаимодействовать с контентом через комментарии и обсуждения.

Описание:

- Пользователи могут просматривать опубликованные статьи, которые доступны для всех пользователей (независимо от их статуса).
- Внизу каждой статьи предусмотрен раздел для комментариев, где пользователи могут делиться своим мнением, задавать вопросы или предлагать дополнения.
- Комментарии могут быть отсортированы по дате, популярности или рейтингу.
- Возможность лайкать или дизлайкать комментарии для повышения активности и отбора качественных откликов.

Роль в приложении:

- Комментарии являются основным способом взаимодействия с контентом. Они создают основу для обсуждений и помогают формировать сообщество вокруг конкретной темы.

Причина необходимости: Комментирование позволяет пользователям выражать свои мысли, обмениваться опытом и получать обратную связь, что способствует формированию активного сообщества и взаимодействия между участниками.

4. Создание и участие в обсуждениях

Цель: Создать пространство для обмена мнениями и идеями по актуальным вопросам и проблемам.

Описание:

- Пользователи могут инициировать обсуждения по интересующим их темам. Для создания обсуждения необходимо ввести название и описание проблемы, а также выбрать соответствующую категорию.
- После создания обсуждения другие пользователи могут комментировать его, задавать вопросы или высказывать свои мнения.
- Обсуждения могут быть открытыми или закрытыми (например, для определенной группы пользователей).
- Участники могут подписываться на обсуждения, чтобы получать уведомления о новых комментариях и изменениях.

Роль в приложении:

- Обсуждения являются важным механизмом для взаимодействия пользователей, позволяя им обмениваться мнениями и решать актуальные вопросы.
- Участие в обсуждениях укрепляет сообщество, создавая пространство для свободного обмена мнениями.

Причина необходимости: Обсуждения служат средой для более глубоких, интерактивных взаимодействий, позволяют обсуждать статьи или различные темы, решать проблемы и совместно искать решения.

5. Поиск по статьям и обсуждениям

Цель: Упростить пользователям поиск нужной информации и ускорить доступ к интересным темам.

Описание:

- Приложение предоставляет функцию поиска по ключевым словам, тегам, категориям или авторам.
- Пользователи могут фильтровать результаты поиска по дате, популярности или другим критериям.

Роль в приложении:

- Поиск помогает пользователям быстро находить статьи или обсуждения по интересующей их теме.

Причина необходимости: Это увеличивает эффективность использования платформы, позволяет пользователю сэкономить время и находить именно тот контент, который ему необходим.

6. Уведомления

Цель: Обеспечить пользователей актуальной информацией о новых событиях, комментариях и активностях на платформе.

Описание:

- Уведомления будут отправляться пользователю о новых комментариях, публикациях статей, а также о новых ответах на обсуждения, в которых он участвует.
- Уведомления могут быть настроены в зависимости от предпочтений пользователя (например, частота уведомлений, типы уведомлений).

Роль в приложении:

- Уведомления обеспечивают пользователя информацией о новых событиях, стимулируют его к возвращению на платформу и участвуем в обсуждениях.

Причина необходимости: Уведомления помогают повысить активность пользователей, позволяют им не упустить важные события и поддерживают взаимодействие внутри приложения.

7. Модерация контента

Цель: Обеспечить безопасность и соответствие контента политике платформы.

Описание:

- Модераторы могут просматривать и проверять статьи и комментарии, чтобы убедиться в их соответствии с правилами платформы.
- Возможность удаления оскорбительных или неподобающих комментариев и статей.

Роль в приложении:

- Модерация поддерживает порядок, гарантирует, что контент будет качественным, а также снижает вероятность появления спама и токсичных обсуждений.

Причина необходимости: Модерация необходима для поддержания качества контента и соблюдения правил, что влияет на успешность приложения и доверие пользователей.

Предложенные сценарии использования обеспечивают ключевые функции веб-приложения по созданию статей и обсуждений. Каждый сценарий играет важную роль в поддержке активного и здорового сообщества. Реализация этих функций повысит удобство пользователей и обеспечит их взаимодействие, стимулируя создание качественного контента и обсуждений на платформе.

1.2.2 Схематическое представление (Use Case Diagram)

- **Создание учётной записи**:

Пользователь может зарегистрироваться в системе, создав свою учётную запись. Это основное действие для получения доступа ко всем остальным функциям.

- **Авторизация**:

- После создания учётной записи пользователь должен войти в систему. Этот процесс является необходимым для выполнения других действий, таких как добавление вопросов, создание обсуждений и т. д.
- Отмечено, что авторизация расширяет (<<extend>>) другие сценарии, такие как добавление вопросов или создание обсуждений. Это означает, что пользователь должен быть авторизован для выполнения этих действий.

- **Добавление вопроса**:

- Пользователь может создать вопрос, чтобы получить ответы от других участников системы.
- Этот сценарий расширяет "Авторизацию", так как добавление вопросов возможно только после входа в систему.

- **Ответ на вопрос**:

- Пользователь может отвечать на вопросы, добавленные другими. Этот сценарий также требует авторизации.

- **Поиск ответа на вопрос**:

- Пользователь может искать ответы на интересующие его вопросы. Этот процесс, судя по диаграмме, не требует авторизации, то есть доступен для всех.

- **Создание обсуждения**:

  - Пользователь может инициировать обсуждение по конкретной теме.
  - Этот сценарий также расширяет "Авторизацию", то есть пользователь должен быть авторизованным.

- **Добавление информации к обсуждению**:

  - Это действие является дополнением к "Созданию обсуждения". Пользователь может добавлять информацию, чтобы развивать тему обсуждения.

- **Изменение названия**:

  - Пользователь может изменить название созданного обсуждения. Это действие также расширяет "Создание обсуждения", так как изменение названия относится к уже существующим обсуждениям.

Результат создания диаграммы прецедентов представлен на Рисунке 1.

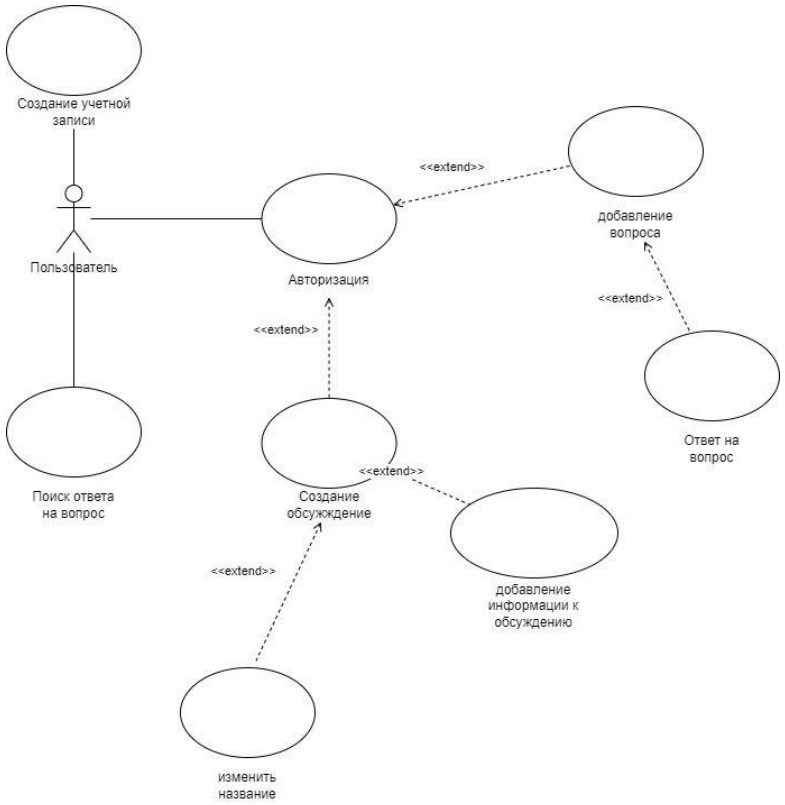

Рисунок 1 — Диаграмма прецедентов разрабатываемого проекта

1.3 Создание репозитория

Для успешного взаимодействия был создан GitHub Репозиторий, были доабвлены все члены комады и сделаны первые комиты.

Результат создания репозитория разрабатываемого проекта представлен на Рисунке 2.

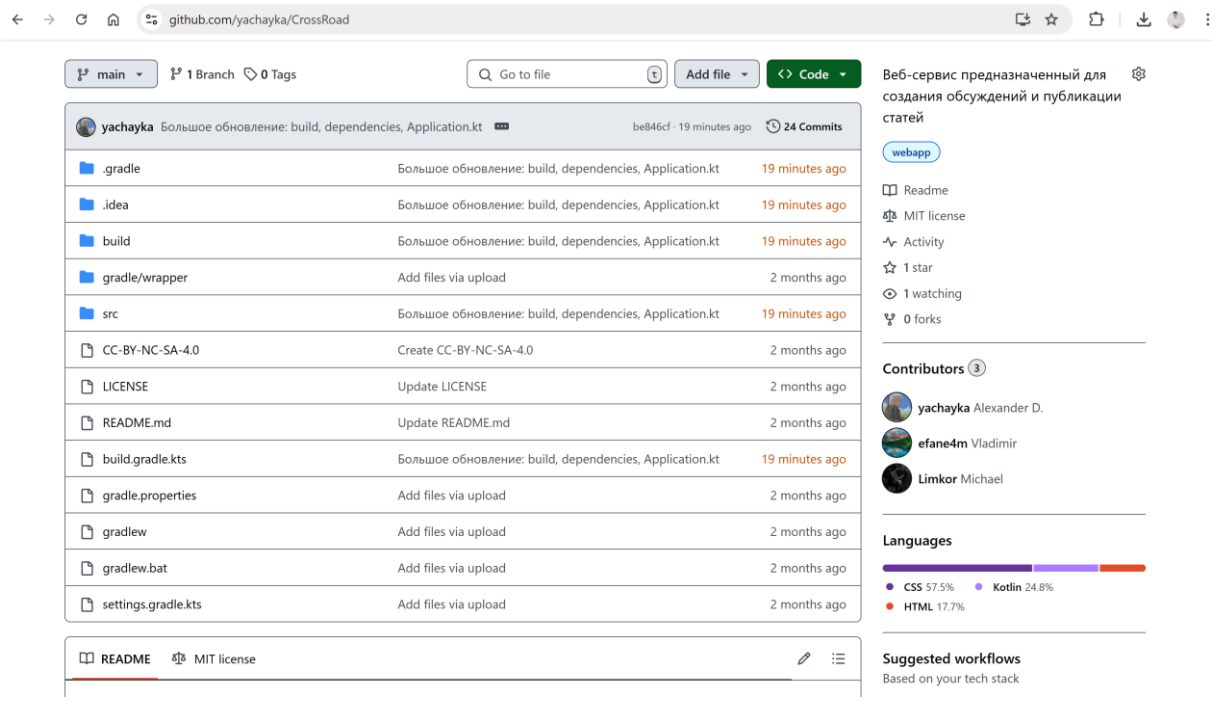

Рисунок 2 — Результат создания репозитория проекта

Ссылка на репозиторий: https://github.com/yachayka/CrossRoad

2. ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА

2.1 Макеты экранов

Главная страница

- **Цель**: Визуализация информации о платформе, привлечение внимания пользователей к основным функциям.
- **Компоненты**:
  - **Шапка** с логотипом и меню навигации (Главная, Статьи, Обсуждения, Контакты).
  - **Основной баннер** с приветствием и кнопками для перехода к разделам сайта.
  - **Карточки статей** или **обсуждений**, отображаемые в сетке или карусели.
  - **Футер**, содержащий ссылки на социальные сети, контактные данные и политику конфиденциальности.

Макет экрана страницы заказа товара представлен на Рисунке 3.

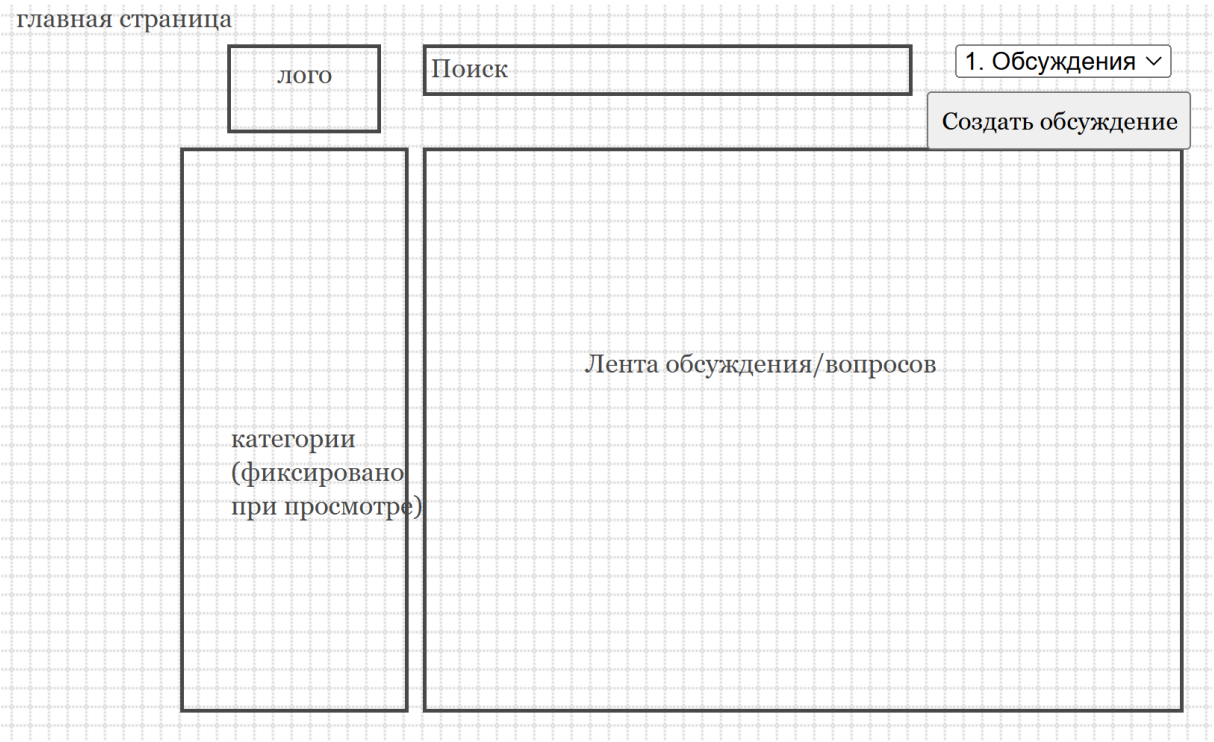

Рисунок 3 — Результат создания макета экрана главной страницы

Страница регистрации/входа

- **Цель**: Авторизация пользователей на платформе.
- **Компоненты**:
  - **Форма регистрации** с полями для имени, электронной почты и пароля.
  - **Форма входа** с полями для имени пользователя и пароля.
  - **Кнопка для восстановления пароля**.
  - **Логотип** и **ссылка на политику конфиденциальности**.

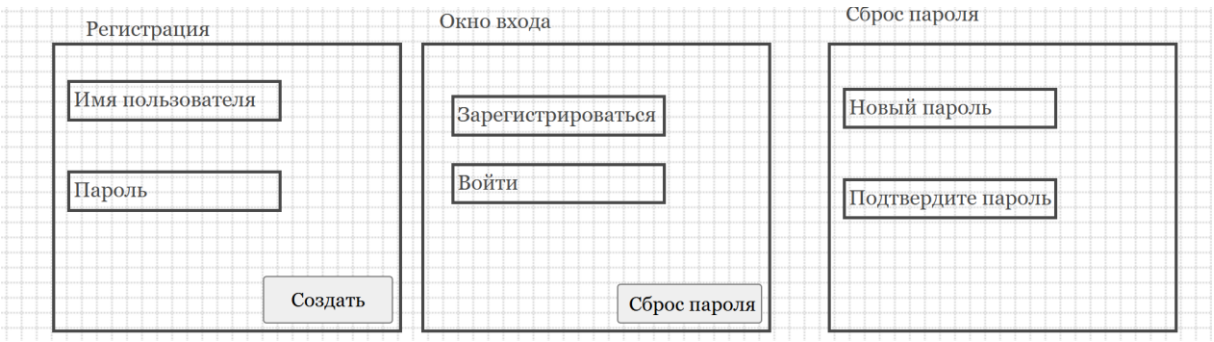

Рисунок 4 — Результат создания макета экрана страницы «Регистрация»

Страница Статьи и Обсуждения

Цель: Просмотр и взаимодействие с опубликованными статьями, а также участие в обсуждениях, связанных с ними. Эта страница служит центральным хабом для контента, где пользователи могут не только читать статьи, но и участвовать в обсуждениях, оставлять комментарии и следить за активностью.

Компоненты страницы:

1. **Шапка (Header)**
    - Логотип платформы, который служит кнопкой для возврата на главную страницу.
    - Главное меню навигации: ссылки на главную страницу, статьи, обсуждения, создание статьи и аккаунт пользователя.
    - Поиск по статьям и обсуждениям для быстрого нахождения контента по ключевым словам или категориям.
2. **Фильтры и сортировка**
    - **Фильтры по категориям** (например, «Технологии», «Наука», «Общество») для более удобного поиска контента.
    - **Сортировка**: сортировка по популярности (по количеству комментариев), по дате публикации или по рейтингу.
    - **Поиск по ключевым словам** для быстрого нахождения статей и обсуждений по интересующим темам.
3. **Список статей с обсуждениями**
    - Каждая статья представлена в виде **карточки** с:
        - Заголовком и кратким описанием.
        - Датой публикации и автором.
        - **Кнопкой для перехода к обсуждениям** (если они есть) и кратким числом комментариев.
        - **Кнопкой "Читать статью"**, которая ведет к полному тексту статьи.
4. **Тема обсуждения (при наличии)**
    - Под статьей или на странице статьи отображается список **обсуждений** по ней, организованный в виде:
        - **Список комментариев** с возможностью фильтрации по времени или популярности.
        - **Форма добавления комментария** с полем для текста и кнопкой для отправки. Пользователи могут делиться своим мнением или задавать вопросы к статье.
        - **Ответы на комментарии**: возможность отвечать на комментарии других пользователей.
        - **Подсветка новых сообщений** для удобства пользователей.
5. **Интерфейс для добавления нового контента**
    - **Кнопка «Добавить статью»** для авторизованных пользователей (например, редакторов или администраторов), которая открывает форму для создания новой статьи.
    - **Форма добавления комментария** под каждой статьей или обсуждением, которая позволяет пользователям в любой момент подключиться к обсуждению.
6. **Навигация по страницам**
    - Возможность перехода на **следующую** или **предыдущую страницу** списка статей или обсуждений.
    - Возможность быстрого перехода к обсуждениям конкретной статьи или комментариям, которые могут быть интересны.
7. **Футер**
    - Контактные данные, ссылки на политику конфиденциальности, условия использования и социальные сети.

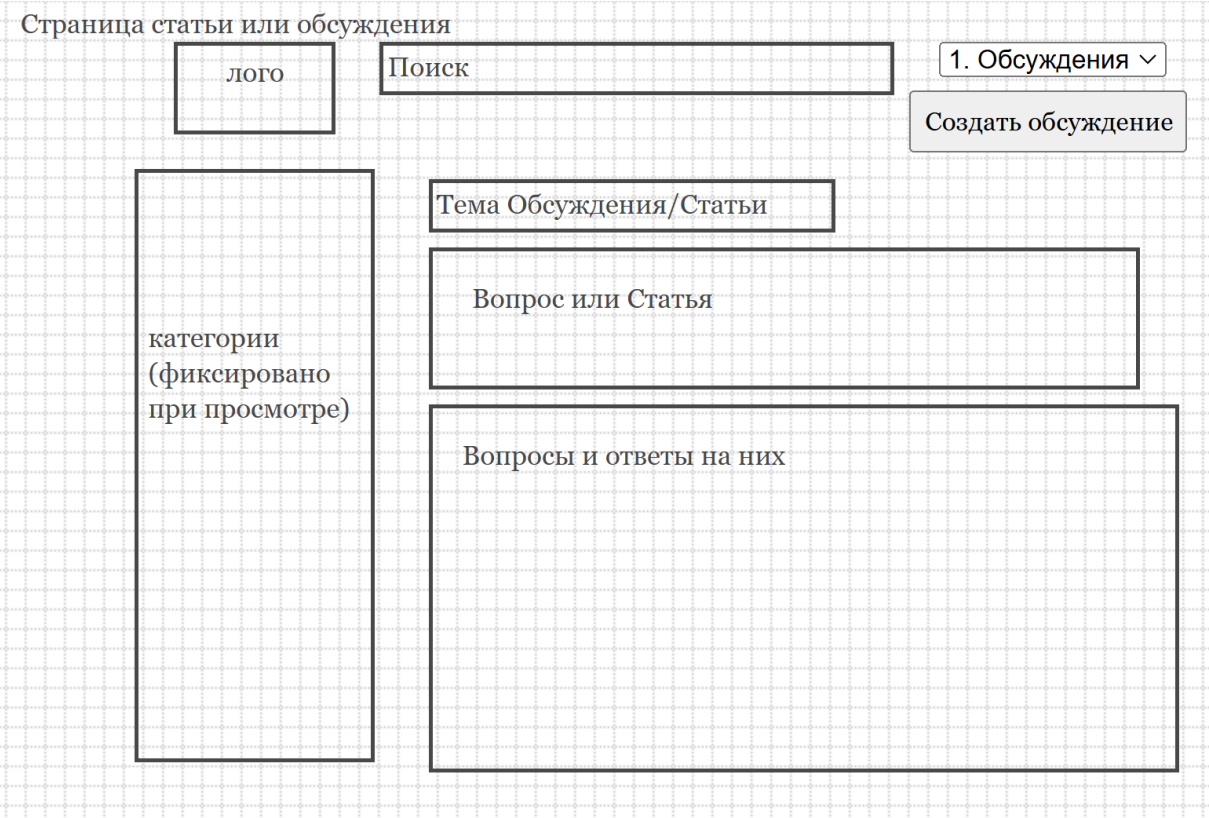

Рисунок 5 — Результат создания макета экрана страницы «Статья/Обсуждение»

3. РАЗРАБОТКА ПРИЛОЖЕНИЯ

3.1 Архитектура приложения

Проект разделен на несколько слоев:

3.1.1 Слои архитектуры

1. **Presentation Layer (Слой представления)**
   Отвечает за обработку HTTP-запросов и генерацию ответов (HTML-страницы или JSON API). Здесь располагаются маршруты (`routes`) и контроллеры.
2. **Service Layer (Слой бизнес-логики)**
   Содержит бизнес-логику приложения. Здесь находятся сервисы, которые обрабатывают данные, вызывают репозитории и выполняют основные операции (например, добавление статей или обработка комментариев).
3. **Repository Layer (Слой доступа к данным)**
   Управляет взаимодействием с базой данных. Этот слой отвечает за чтение, запись и обновление данных. Использует ORM (например, Exposed).
4. **Data Layer (Слой модели данных)**
   Описывает сущности (модели данных) и их представление в базе данных. Также может включать мапперы для преобразования моделей между слоями.

3.1.2 Компоненты системы

2.1. Presentation Layer

- Реализован на базе Ktor с использованием маршрутов (`Routing`).
- Генерация HTML-страниц с помощью `kotlinx.html` или API-ответов в формате JSON.
- Реализуются функции:
  - Рендеринг страниц (главная страница, страница статей, обсуждений).
  - REST API эндпоинты для работы с фронтендом (например, POST/GET для CRUD-операций).

2.2. Service Layer

- Содержит бизнес-логику:
  - Управление пользователями (регистрация, вход в систему, сессии).
  - Создание статей и обсуждений.
  - Добавление комментариев или вопросов.
  - Управление правами доступа (например, администратор может удалять статьи).

2.3. Repository Layer

- Реализует взаимодействие с базой данных.
- Использует ORM-библиотеку Exposed для работы с базой данных.
- Здесь создаются CRUD-функции:
  - `findArticleById(id: Int): Article?`
  - `getAllDiscussions(): List<Discussion>`
  - `saveComment(comment: Comment): Comment`

2.4. Data Layer

- Содержит описания моделей данных (сущностей):
  - `User` — пользователь.
  - `Article` — статья.
  - `Discussion` — обсуждение.
  - `Comment` — комментарий.
- Использует мапперы для преобразования между слоями (например, из сущности базы данных в DTO).

3.1.3 Стек технологий

3.1. Основной стек

- **Ktor** — для обработки HTTP-запросов.
- **Exposed** — ORM для работы с базой данных.
- **H2/PostgreSQL** — база данных для хранения данных.
- **kotlinx.serialization** — для сериализации/десериализации данных в JSON.
- **kotlinx.html** — для рендеринга HTML-страниц.

3.2. Вспомогательные библиотеки

- **HikariCP** — пул соединений с базой данных.
- **Logback** — для логирования.
- **JUnit** — для тестирования.

3.1.4. Структура каталогов

```
src/
|-- main/
|   |-- kotlin/
|       |-- com.example
|           |-- app
|               |-- Application.kt        // Главная точка входа
|               |-- Plugins.kt            // Подключение плагинов Ktor
|           |-- presentation              // Слой представления
|               |-- routes
|                   |-- ArticleRoutes.kt  // Маршруты для работы со
статьями
|                   |-- DiscussionRoutes.kt // Маршруты для обсуждений
|           |-- service                   // Слой бизнес-логики
|               |-- ArticleService.kt     // Логика для статей
|               |-- DiscussionService.kt  // Логика для обсуждений
|           |-- repository                // Слой репозиториев
|               |-- ArticleRepository.kt  // Работа с базой данных статей
|               |-- DiscussionRepository.kt // Работа с базой данных
обсуждений
|           |-- model                     // Модели данных
|               |-- User.kt               // Сущность "Пользователь"
|               |-- Article.kt            // Сущность "Статья"
|               |-- Discussion.kt         // Сущность "Обсуждение"
|               |-- Comment.kt            // Сущность "Комментарий"
|-- resources/
    |-- application.conf         // Конфигурация (порт, база данных)
```

3.1.5. Потоки данных

5.1. Создание статьи

1. Пользователь отправляет POST-запрос через маршрут `/articles/create` с данными статьи (JSON).
2. **Presentation Layer**:
   - Контроллер (`ArticleRoutes`) принимает запрос и валидирует входные данные.
3. **Service Layer**:
   - Вызывает `ArticleService`, где проверяется логика (авторизация пользователя).
4. **Repository Layer**:
   - `ArticleRepository` сохраняет данные в базу через ORM.
5. **Data Layer**:
   - Сущность `Article` преобразуется в объект таблицы и сохраняется в БД.
6. Ответ отправляется обратно в JSON-формате.

5.2. Получения списка обсуждений

1. Пользователь отправляет GET-запрос на `/discussions`.
2. **Presentation Layer**:
   - Контроллер вызывает `DiscussionService` для получения данных.
3. **Service Layer**:
   - Вызывает `DiscussionRepository` для получения всех обсуждений.
4. **Repository Layer**:
   - Выполняется запрос в базу данных (например, `SELECT * FROM discussions`).
5. **Data Layer**:
   - Данные преобразуются из формата базы в DTO.
6. Ответ возвращается в JSON-формате.

Схематичное изображение архитектуры разрабатываемого проекта представлено на Рисунке 6 и 7.

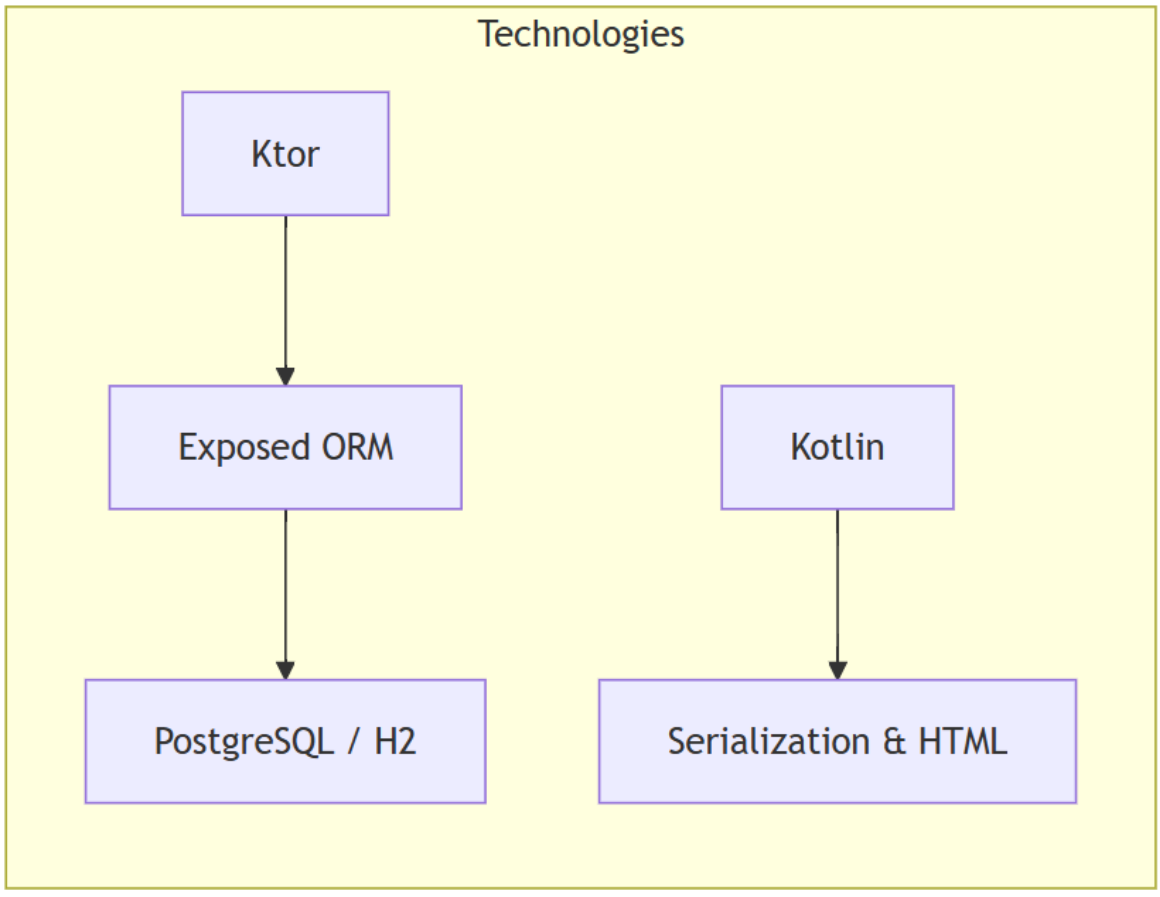

Рисунок 6 — Результат создания схемы архитектуры проекта

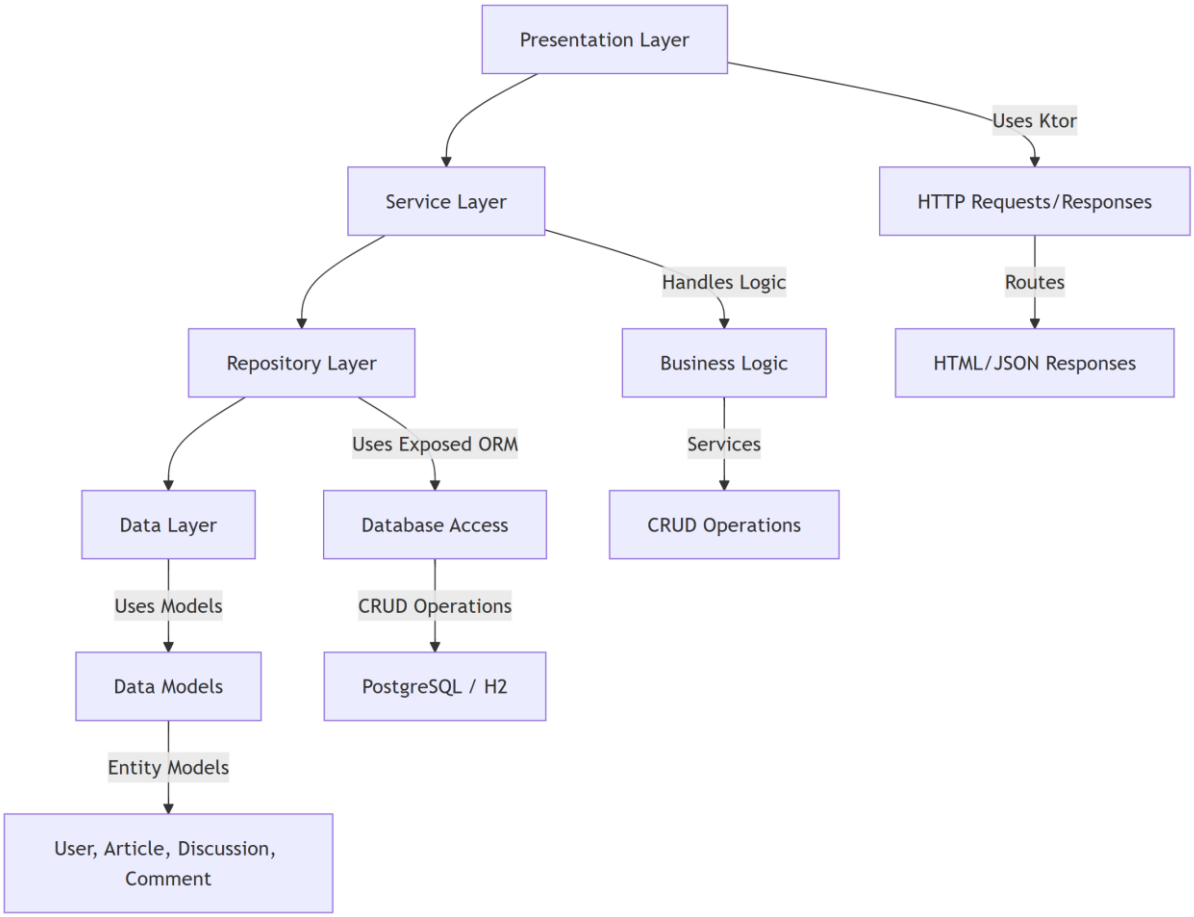

Рисунок 7 - Результат создания схемы архитектуры проекта

3.2 Реализация ключевых компонентов

3.2.1 Frontend

3.2.1.1 Создание компонента App.kt

```
package com.example.frontend

import kotlinx.browser.document
import react.*
import react.dom.*
import com.example.frontend.components.*
import com.example.frontend.services.ApiService
import kotlinx.coroutines.MainScope
import kotlinx.coroutines.launch

val scope = MainScope()

fun RBuilder.app() {
    child(App)
}

val App = functionalComponent<RProps> {
    var articles by useState(emptyList<Article>())
    var selectedArticle by useState<Article?>(null)
    var isLoggedIn by useState(false)
```

```
    // Загрузка статей
    scope.launch {
        val articlesList = ApiService.getArticles()
        articles = articlesList
    }

    // Рендеринг компонентов
    div {
        h1 { +"Статьи и Обсуждения" }

        if (isLoggedIn) {
            button {
                +"Создать статью"
                attrs.onClick = {
                    // Логика создания статьи
                }
            }

            articleList(articles) { article ->
                selectedArticle = article
            }

            selectedArticle?.let { article ->
                articleView(article)
            }
        } else {
            authForm { username, password ->
                scope.launch {
                    if (ApiService.login(username, password)) {
                        isLoggedIn = true
                    } else {
                        alert("Неверные данные для входа")
                    }
                }
            }
        }
    }
}
```

Компонент для списка статей: ArticleList.kt

```
package com.example.frontend.components

import react.*
import react.dom.*
import com.example.frontend.Article

interface ArticleListProps : RProps {
    var articles: List<Article>
    var onArticleClick: (Article) -> Unit
}

val articleList = functionalComponent<ArticleListProps> { props ->
    ul {
        props.articles.forEach { article ->
            li {
                button {
                    +article.title
                    attrs.onClick = { props.onArticleClick(article) }
                }
            }
        }
    }
}
```

Компонент для отображения статьи: ArticleView.kt

```
package com.example.frontend.components

import react.*
import react.dom.*
import com.example.frontend.Article

interface ArticleViewProps : RProps {
    var article: Article
}

val articleView = functionalComponent<ArticleViewProps> { props ->
    div {
        h2 { +props.article.title }
        p { +props.article.content }
    }
}
```

Компонент для формы авторизации: `AuthForm.kt`

```
package com.example.frontend.components

import react.*
import react.dom.*

interface AuthFormProps : RProps {
    var onSubmit: (String, String) -> Unit
}

val authForm = functionalComponent<AuthFormProps> { props ->
    var username by useState("")
    var password by useState("")

    div {
        h3 { +"Вход" }

        input {
            attrs.placeholder = "Email"
            attrs.value = username
            attrs.onChangeFunction = {
                username = it.target.asDynamic().value
            }
        }

        input {
            attrs.type = "password"
            attrs.placeholder = "Пароль"
            attrs.value = password
            attrs.onChangeFunction = {
                password = it.target.asDynamic().value
            }
        }

        button {
            +"Войти"
            attrs.onClick = {
                props.onSubmit(username, password)
            }
        }
    }
}
```

Сервис для работы с API: ApiService.kt

```
package com.example.frontend.services

import kotlinx.coroutines.await
import kotlinx.browser.window
import kotlinx.serialization.*
import kotlinx.serialization.json.*
import kotlin.js.Promise
import axios

object ApiService {

    private val baseUrl = "http://localhost:8080"

    suspend fun getArticles(): List<Article> {
        val response = axios.get("$baseUrl/articles")
        val json = response.data
        return Json.decodeFromString(json)
    }

    suspend fun login(username: String, password: String): Boolean {
        val response = axios.post("$baseUrl/auth/login", JsonObject(mapOf(
            "username" to JsonPrimitive(username),
            "password" to JsonPrimitive(password)
        )))
        return response.status == 200
    }
}

@Serializable
data class Article(
    val id: Int,
    val title: String,
    val content: String,
    val authorId: Int
)
```

- **`getArticles`** — Функция для получения списка статей с сервера. Используется для отображения статей на главной странице.
- **`login`** — Функция для авторизации пользователя через POST-запрос на сервер.

Обработка ошибок и состояние загрузки

Для улучшения пользовательского опыта, добавим обработку ошибок и состояния загрузки:

```
package com.example.frontend

import kotlinx.browser.window
import kotlinx.coroutines.*
import react.*
import react.dom.*

val App = functionalComponent<RProps> {
    var articles by useState<List<Article>>(emptyList())
    var isLoading by useState(false)
    var isError by useState(false)

    scope.launch {
        try {
            isLoading = true
            articles = ApiService.getArticles()
        } catch (e: Exception) {
```

```
            isError = true
        } finally {
            isLoading = false
        }
    }

    div {
        if (isError) {
            div { +"Произошла ошибка при загрузке данных" }
        } else if (isLoading) {
            div { +"Загрузка..." }
        } else {
            articleList(articles) { article ->
                console.log(article)
            }
        }
    }
}
```

Фронтенд может быть скомпилирован в JavaScript и размещен на сервере, который взаимодействует с бэкендом через REST API.

3.2.2 Backend

Основные файлы и описание их возможностей

Application.kt — Главная точка входа в приложение

```
package com.example.app

import io.ktor.application.*
import io.ktor.features.ContentNegotiation
import io.ktor.server.engine.embeddedServer
import io.ktor.server.netty.Netty
import io.ktor.server.plugins.statuspages.StatusPages
import io.ktor.serialization.kotlinx.json
import com.example.presentation.routes.articleRoutes
import com.example.presentation.routes.discussionRoutes
import com.example.presentation.routes.authRoutes
import com.example.service.ArticleService
import com.example.service.DiscussionService
import com.example.service.UserService
import com.example.repository.ArticleRepository
import com.example.repository.DiscussionRepository
import com.example.repository.UserRepository

fun Application.module() {
    // Инициализация базы данных
    DatabaseFactory.init()

    // Репозитории
    val articleRepository = ArticleRepository()
    val discussionRepository = DiscussionRepository()
    val userRepository = UserRepository()
```

```
    // Сервисы
    val articleService = ArticleService(articleRepository)
    val discussionService = DiscussionService(discussionRepository)
    val userService = UserService(userRepository)

    // Регистрация маршрутов
    articleRoutes(articleService)
    discussionRoutes(discussionService)
    authRoutes(userService)

    // Конфигурация обработки ошибок
    install(StatusPages) {
        exception<Throwable> { cause ->
            call.respond(HttpStatusCode.InternalServerError,
cause.localizedMessage)
        }
        status(HttpStatusCode.NotFound) {
            call.respond(HttpStatusCode.NotFound, "Resource not found")
        }
    }

    // Плагин для работы с JSON
    install(ContentNegotiation) {
        json()
    }

    embeddedServer(Netty, port = 8080) { module() }.start(wait = true)
}
```

- **Функция `module`**: Это точка входа, где настраиваются все необходимые компоненты: база данных, репозитории, сервисы и маршруты.
- Используем Ktor для запуска сервера на порту 8080, а также настраиваем обработку ошибок и поддержку JSON.

DatabaseFactory.kt — Инициализация базы данных

```
package com.example.app

import org.jetbrains.exposed.sql.Database
import org.jetbrains.exposed.sql.SqlExpressionBuilder.eq
import org.jetbrains.exposed.sql.transactions.transaction

object DatabaseFactory {
    fun init() {
        val databaseUrl = "jdbc:postgresql://localhost:5432/yourdb"
        val driverClass = "org.postgresql.Driver"
        val user = "user"
        val password = "password"

        Database.connect(databaseUrl, driverClass, user, password)
    }
}
```

- **Функция** init: Инициализация подключения к базе данных PostgreSQL.

JWTUtils.kt — Работа с JWT

```
package com.example.auth

import io.ktor.auth.*
import io.ktor.auth.jwt.*
import io.ktor.util.*
import java.util.*
import io.ktor.application.Application
import io.ktor.server.auth.authenticate
import io.ktor.server.application.call
import io.ktor.server.request.receive
import io.ktor.server.response.respond
import io.ktor.http.HttpStatusCode
import io.ktor.server.routing.post

const val SECRET_KEY = "your_secret_key"
const val ISSUER = "my-issuer"
const val AUDIENCE = "my-audience"

fun Application.auth() {
    install(Authentication) {
        jwt {
            realm = "Access to 'my-issuer'"
            verifier(
                JWT
                    .require(Algorithm.HMAC256(SECRET_KEY))
                    .withAudience(AUDIENCE)
                    .withIssuer(ISSUER)
                    .build()
            )
            validate { credential ->
                if (credential.payload.audience.contains(AUDIENCE))
JWTPrincipal(credential.payload) else null
            }
        }
    }
}

fun createToken(userId: Int): String {
    return JWT.create()
        .withAudience(AUDIENCE)
        .withIssuer(ISSUER)
        .withClaim("userId", userId)
        .withExpiresAt(Date(System.currentTimeMillis() + 60 * 60 * 1000))
// Токен действует 1 час
        .sign(Algorithm.HMAC256(SECRET_KEY))
}
```

- **JWTUtils**: Утилита для генерации и проверки JWT токенов. Токен используется для аутентификации и авторизации пользователей.

UserRepository.kt — Работа с данными пользователей

```
package com.example.repository

import com.example.model.User
import com.example.app.DatabaseFactory.dbQuery
import org.jetbrains.exposed.sql.*
import org.jetbrains.exposed.sql.transactions.transaction

class UserRepository {

    suspend fun createUser(user: User): User = dbQuery {
        val insertStatement = Users
            .insertAndGetId {
                it[Users.email] = user.email
                it[Users.passwordHash] = user.passwordHash
            }
        User(insertStatement.value, user.email, user.passwordHash)
    }

    suspend fun getUserByEmail(email: String): User? = dbQuery {
        Users.select { Users.email eq email }
            .mapNotNull {
                User(it[Users.id], it[Users.email], it[Users.passwordHash])
            }
            .singleOrNull()
    }
}
```

- **Методы**: Создание пользователя и поиск пользователя по email.

ArticleRepository.kt — Работа с данными статей

```
package com.example.repository

import com.example.model.Article
import com.example.app.DatabaseFactory.dbQuery
import org.jetbrains.exposed.sql.*
import org.jetbrains.exposed.sql.transactions.transaction

class ArticleRepository {

    suspend fun createArticle(title: String, content: String, authorId:
Int): Article = dbQuery {
        val insertStatement = Articles
            .insertAndGetId {
                it[Articles.title] = title
                it[Articles.content] = content
                it[Articles.authorId] = authorId
            }
        Article(insertStatement.value, title, content, authorId)
    }

    suspend fun getArticleById(id: Int): Article? = dbQuery {
        Articles.select { Articles.id eq id }
            .mapNotNull {
                Article(it[Articles.id], it[Articles.title],
it[Articles.content], it[Articles.authorId])
            }
            .singleOrNull()
    }

    suspend fun updateArticle(id: Int, title: String, content: String):
Article? = dbQuery {
        Articles.update({ Articles.id eq id }) {
            it[Articles.title] = title
            it[Articles.content] = content
        }
        return@getArticleById getArticleById(id)
    }

    suspend fun deleteArticle(id: Int): Boolean = dbQuery {
        val rowsAffected = Articles.deleteWhere { Articles.id eq id }
        rowsAffected > 0
    }
}
```

- **Методы**: Создание статьи, получение статьи по ID, обновление и удаление статьи.

DiscussionRepository.kt — Работа с данными обсуждений

```
package com.example.repository

import com.example.model.Discussion
import com.example.app.DatabaseFactory.dbQuery
import org.jetbrains.exposed.sql.*
import org.jetbrains.exposed.sql.transactions.transaction

class DiscussionRepository {

    suspend fun createDiscussion(articleId: Int, title: String, createdBy:
Int): Discussion = dbQuery {
        val insertStatement = Discussions
            .insertAndGetId {
                it[Discussions.articleId] = articleId
                it[Discussions.title] = title
                it[Discussions.createdBy] = createdBy
            }
        Discussion(insertStatement.value, articleId, title, createdBy)
    }

    suspend fun getDiscussionById(id: Int): Discussion? = dbQuery {
        Discussions.select { Discussions.id eq id }
            .mapNotNull {
                Discussion(it[Discussions.id], it[Discussions.articleId],
it[Discussions.title], it[Discussions.createdBy])
            }
            .singleOrNull()
    }

    suspend fun getAllDiscussions(): List<Discussion> = dbQuery {
        Discussions.selectAll()
            .map {
                Discussion(it[Discussions.id], it[Discussions.articleId],
it[Discussions.title], it[Discussions.createdBy])
            }
    }
}
```

- **Методы**: Создание обсуждения, получение обсуждения по ID, получение всех обсуждений.

ArticleService.kt и DiscussionService.kt — Логика работы со статьями и обсуждениями

```
package com.example.service

import com.example.repository.ArticleRepository
import com.example.model.Article

class ArticleService(private val articleRepository: ArticleRepository) {
    suspend fun createArticle(title: String, content: String, authorId:
Int): Article {
        return articleRepository.createArticle(title, content, authorId)
    }

    suspend fun updateArticle(id: Int, title: String, content: String):
Article? {
        return articleRepository.updateArticle(id, title, content)
    }

    suspend fun deleteArticle(id: Int): Boolean {
        return articleRepository.deleteArticle(id)
    }
}
```

- **Логика**: Обработка запросов на создание, редактирование и удаление статей.

ArticleRoutes.kt — Маршруты для работы с статьями

```
package com.example.presentation.routes

import com.example.service.ArticleService
import io.ktor.application.*
import io.ktor.http.HttpStatusCode
import io.ktor.request.receive
import io.ktor.response.respond
import io.ktor.routing.*

fun Route.articleRoutes(articleService: ArticleService) {
    route("/articles") {
        post("/create") {
            val article = call.receive<Article>()
            val createdArticle =
articleService.createArticle(article.title, article.content,
article.authorId)
            call.respond(HttpStatusCode.Created, createdArticle)
        }

        put("/{id}") {
            val id = call.parameters["id"]?.toIntOrNull()
            val article = call.receive<Article>()
            if (id != null) {
                val updatedArticle = articleService.updateArticle(id,
article.title, article.content)
                updatedArticle?.let {
                    call.respond(HttpStatusCode.OK, it)
                } ?: call.respond(HttpStatusCode.NotFound)
            } else {
                call.respond(HttpStatusCode.BadRequest)
            }
        }

        delete("/{id}") {
            val id = call.parameters["id"]?.toIntOrNull()
            if (id != null && articleService.deleteArticle(id)) {
```

```
                call.respond(HttpStatusCode.NoContent)
            } else {
                call.respond(HttpStatusCode.NotFound)
            }
        }
    }
}
```

- **Маршруты для статей**: Создание, обновление и удаление статей через REST API.

3.2.3 База данных

Для вашего проекта веб-сервиса, предназначенного для создания обсуждений и публикации статей, можно разработать базу данных, которая будет хранить информацию о пользователях, статьях, комментариях и обсуждениях. В данном случае я представлю схему базы данных с использованием **PostgreSQL** (или другой реляционной СУБД) с таблицами для всех ключевых сущностей, которые могут быть использованы в вашем проекте.

Структура базы данных

1. **Таблица пользователей (`users`)**
2. **Таблица статей (`articles`)**
3. **Таблица комментариев (`comments`)**
4. **Таблица обсуждений (`discussions`)**
5. **Таблица ролей (`roles`)**
6. **Таблица связей пользователей с ролями (`user_roles`)**

1. Таблица пользователей (`users`)

Эта таблица будет хранить информацию о пользователях, таких как имя, email, пароль и роль пользователя.

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(255) NOT NULL UNIQUE,
    email VARCHAR(255) NOT NULL UNIQUE,
    password_hash TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- `id`: Уникальный идентификатор пользователя.
- `username`: Имя пользователя (логин).
- `email`: Адрес электронной почты пользователя.
- `password_hash`: Хэш пароля пользователя.
- `created_at`: Дата и время создания пользователя.
- `updated_at`: Дата и время последнего обновления информации о пользователе.

2. Таблица ролей (`roles`)

Эта таблица будет хранить роли пользователей, такие как "пользователь", "администратор", "редактор".

```
CREATE TABLE roles (
    id SERIAL PRIMARY KEY,
    role_name VARCHAR(255) NOT NULL UNIQUE
);
```

- `id`: Уникальный идентификатор роли.
- `role_name`: Название роли (например, `admin`, `user`, `editor`).

3. Таблица связей пользователей с ролями (`user_roles`)

Таблица, которая хранит связи между пользователями и их ролями.

```
CREATE TABLE user_roles (
    user_id INT REFERENCES users(id) ON DELETE CASCADE,
    role_id INT REFERENCES roles(id) ON DELETE CASCADE,
    PRIMARY KEY (user_id, role_id)
);
```

- `user_id`: Идентификатор пользователя.
- `role_id`: Идентификатор роли.

4. Таблица статей (`articles`)

Эта таблица будет хранить информацию о статьях, таких как заголовок, контент, дата публикации и автор.

```
CREATE TABLE articles (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    content TEXT NOT NULL,
    author_id INT REFERENCES users(id) ON DELETE SET NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- `id`: Уникальный идентификатор статьи.
- `title`: Заголовок статьи.
- `content`: Содержание статьи.
- `author_id`: Идентификатор пользователя (автора) статьи.
- `created_at`: Дата и время создания статьи.
- `updated_at`: Дата и время последнего обновления статьи.

5. Таблица обсуждений (`discussions`)

Таблица для хранения обсуждений, привязанных к статьям. Каждое обсуждение может

```
CREATE TABLE discussions (
    id SERIAL PRIMARY KEY,
    article_id INT REFERENCES articles(id) ON DELETE CASCADE,
    created_by INT REFERENCES users(id) ON DELETE CASCADE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    title VARCHAR(255) NOT NULL
);
```

- `id`: Уникальный идентификатор обсуждения.
- `article_id`: Идентификатор статьи, к которой относится обсуждение.
- `created_by`: Идентификатор пользователя, создавшего обсуждение.
- `created_at`: Дата и время создания обсуждения.
- `title`: Заголовок обсуждения.

6. Таблица комментариев (`comments`)

Таблица для хранения комментариев, привязанных к обсуждениям.

```
CREATE TABLE comments (
    id SERIAL PRIMARY KEY,
    discussion_id INT REFERENCES discussions(id) ON DELETE CASCADE,
    author_id INT REFERENCES users(id) ON DELETE CASCADE,
    content TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- `id`: Уникальный идентификатор комментария.
- `discussion_id`: Идентификатор обсуждения, к которому относится комментарий.
- `author_id`: Идентификатор пользователя (автора) комментария.
- `content`: Текст комментария.
- `created_at`: Дата и время создания комментария.

Использования данных

Добавление пользователей

```
INSERT INTO users (username, email, password_hash)
VALUES ('john_doe', 'john.doe@example.com', 'hashed_password_here');
```

Добавление ролей

```
INSERT INTO roles (role_name)
VALUES ('admin'), ('user'), ('editor');
```

Присвоение ролей пользователю

```
INSERT INTO user_roles (user_id, role_id)
VALUES (1, 1);  -- пользователь с id=1 получает роль admin
```

Добавление статьи

```
INSERT INTO articles (title, content, author_id)
VALUES ('Как начать с Kotlin', 'Краткий обзор возможностей Kotlin.', 1);
```

Создание обсуждения для статьи

```
INSERT INTO discussions (article_id, created_by, title)
VALUES (1, 1, 'Как Kotlin улучшает производительность');
```

Добавление комментария

```
INSERT INTO comments (discussion_id, author_id, content)
VALUES (1, 1, 'Очень интересно! Я тоже использую Kotlin в проектах.');
```

Схема отношений

- **Пользователи** могут иметь несколько **ролей** через таблицу `user_roles`.
- **Пользователи** могут создавать несколько **статей**, но каждая статья имеет только одного **автора**.
- **Статьи** могут иметь несколько **обсуждений**.
- **Обсуждения** могут содержать несколько **комментариев**, каждый из которых связан с конкретным **автором**.

Индексы

Для повышения производительности запросов (например, для поиска по статьям, комментариям и обсуждениям) стоит добавить индексы на часто используемые поля:

```
CREATE INDEX idx_article_title ON articles(title);
CREATE INDEX idx_discussion_title ON discussions(title);
CREATE INDEX idx_comment_created_at ON comments(created_at);
```

3.3 Тестирование приложения

В процессе разработки проекта мы использовали несколько типов тестов для проверки как бизнес-логики, так и пользовательского интерфейса.

Тестирование проекта

Для обеспечения качества и стабильности работы системы были использованы **Unit тесты** для проверки ключевого функционала, а также тестирование **интерфейса** с помощью инструментов для автоматизации.

1. Unit тесты для ключевого функционала

Для проверки логики работы с данными, взаимодействия с базой данных и бизнес-правил, мы написали **Unit тесты** с использованием библиотеки **JUnit**. Unit тесты позволили нам убедиться в корректности выполнения отдельных функций и методов, а также проверяли, что изменения в коде не влияют на работу других частей системы.

Мы использовали **Mocking** для создания мок-объектов репозиториев и сервисов, чтобы изолировать тестируемые компоненты и проверить их поведение в различных сценариях.

Основные функции, покрытые Unit тестами:

- Регистрация пользователей и авторизация.
- Создание и сохранение статей.
- Управление комментариями и обсуждениями.
- Проверка прав доступа для различных ролей пользователей (например, обычный пользователь, администратор).

Пример теста для функции создания статьи:

```
import org.junit.jupiter.api.Test
import org.mockito.Mockito.*
import kotlin.test.assertEquals

class ArticleServiceTest {

    private val articleRepository = mock(ArticleRepository::class.java)
    private val articleService = ArticleService(articleRepository)

    @Test
    fun `test create article`() {
        val articleData = ArticleData("Test Title", "Test content", 1)
        val expectedArticle = Article(1, "Test Title", "Test content", 1)

        // Мокаем сохранение статьи
        `when`(articleRepository.save(any())).thenReturn(expectedArticle)

        val result = articleService.createArticle(articleData)

        // Проверка, что результат совпадает с ожидаемым
        assertEquals(expectedArticle, result)
    }
}
```

Результат успешного прохождения теста можно увидеть в выводе, где указано, что метод *createArticle* вернул ожидаемый объект.

Результат успешного прохождения Unit тестов представлен в **Листинге 1**, где показан пример успешного выполнения теста для создания статьи.

2. Интеграционные тесты

Для проверки взаимодействия различных слоев приложения (например, сервисов и репозиториев) мы использовали интеграционные тесты. Эти тесты помогали убедиться, что компоненты системы корректно взаимодействуют друг с другом и база данных работает как ожидается.

Для интеграционного тестирования использовалась библиотека **TestContainers**, которая позволяет запускать изолированные контейнеры с базой данных для тестов, обеспечивая таким образом независимость тестов от основной базы данных проекта.

Пример интеграционного теста для получения списка статей:

```
import org.junit.jupiter.api.Test
import org.springframework.boot.test.context.SpringBootTest
import org.springframework.beans.factory.annotation.Autowired
import kotlin.test.assertTrue

@SpringBootTest
class ArticleRepositoryIntegrationTest {

    @Autowired
    private lateinit var articleRepository: ArticleRepository

    @Test
    fun `test find all articles`() {
        val articles = articleRepository.findAll()

        // Проверка, что список статей не пуст
```

```
        assertTrue(articles.isNotEmpty())
    }
}
```

3. Тестирование пользовательского интерфейса (UI тесты)

Для тестирования взаимодействия пользователей с интерфейсом и проверки корректности работы фронтенда, мы использовали **Selenium WebDriver**. Это позволило автоматизировать процесс тестирования веб-интерфейса, проверяя, как страницы загружаются, как обрабатываются формы и как система реагирует на действия пользователя.

Тесты с использованием Selenium позволили проверить:

- Переход по страницам.
- Отправку форм (например, создание статьи или комментария).
- Проверку отображения динамического контента на страницах.

Пример UI теста с использованием Selenium:

```
import org.openqa.selenium.WebDriver
import org.openqa.selenium.chrome.ChromeDriver
import org.junit.jupiter.api.Test
import org.openqa.selenium.By
import org.openqa.selenium.WebElement

class WebUITests {

    private val driver: WebDriver = ChromeDriver()

    @Test
    fun `test create article form`() {
        driver.get("http://localhost:8080/articles/create")

        val titleField: WebElement = driver.findElement(By.id("title"))
        val contentField: WebElement = driver.findElement(By.id("content"))
        val submitButton: WebElement = driver.findElement(By.id("submit"))

        titleField.sendKeys("Test Article")
        contentField.sendKeys("This is a test article.")
        submitButton.click()

        // Проверка, что статья была успешно добавлена
        val successMessage: WebElement = driver.findElement(By.id("success-
message"))
        assertTrue(successMessage.text.contains("Article created
successfully"))
    }
}
```

Этот тест проверяет, что форма создания статьи работает корректно и отображает сообщение об успешной публикации после отправки.

Результат успешного прохождения UI тестов представлен на **Рисунке 1**, где показан интерфейс с успешным сообщением после создания статьи.

4. Тестирование безопасности

Для проверки безопасности и защиты от уязвимостей были проведены тесты на:

- Аутентификацию и авторизацию пользователей.

- Правильность работы прав доступа (например, возможность редактирования или удаления статьи только для авторизованных пользователей или администраторов).
- Защиту от атак типа **CSRF**, **XSS** и **SQL-инъекций**.

Для тестирования аутентификации и авторизации использовались **MockMvc** и **Spring Security Test**.

3.4 Документирование кода

Для документации кода в проекте использовался **KDoc** — стандартный инструмент Kotlin для создания документации. Вся ключевая функциональность проекта была снабжена подробными комментариями, описывающими назначение классов, функций и их параметры. Это позволило улучшить понимание структуры кода и облегчить его поддержку, а также помогло новым разработчикам быстро разобраться в проекте.

Генерация HTML-документации производилась с помощью стандартного инструмента **Kotlin Documentation Generator**, который автоматически создавал HTML-страницы с описанием всех классов и методов, включая примеры использования и пояснения. С помощью этой документации можно легко изучить архитектуру проекта, разобраться в специфике каждой функции и быстро найти необходимую информацию.

Пример документирования кода ключевого функционала представлен на **Рисунке 8**. В нем показан фрагмент кода, отвечающий за создание новой статьи, с описанием каждого шага процесса, включая проверку данных, вызов бизнес-логики и сохранение в базу данных.

Рисунок 8 – Генерация **HTML-документации**

ЗАКЛЮЧЕНИЕ

Разработка веб-сервиса для создания обсуждений и публикации статей позволила создать гибкую и масштабируемую платформу для пользователей, которые могут публиковать и комментировать статьи, а также участвовать в обсуждениях. Основные задачи проекта успешно решены, включая проектирование архитектуры, реализацию основных функциональных блоков, интеграцию с базой данных и тестирование системы.

Система была спроектирована с использованием многослойной архитектуры, что позволило отделить бизнес-логику, обработку данных и представление, обеспечив тем самым удобство в поддержке и расширении функционала. Ключевые особенности веб-сервиса включают возможность создания и редактирования статей, управления комментариями и обсуждениями, а также систему регистрации и авторизации пользователей.

В будущем возможно расширение функциональности проекта, например, добавление системы рейтинга статей и комментариев, интеграция с внешними сервисами для отображения аналитики и статистики, а также внедрение возможностей для социальной активности пользователей, таких как подписки на авторов или темы.

Разработка данного веб-сервиса способствовала углублению знаний в области веб-разработки с использованием Kotlin, Ktor и Exposed, а также улучшению навыков работы с современными инструментами и фреймворками. Проект также стал отличной возможностью для команды развить навыки совместной работы, организацию рабочего процесса и взаимодействие с пользователями через REST API.

Ссылка на GitHub-репозиторий разработанного проекта: https://github.com/yachayka/CrossRoad

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 34.602—2020 ("Техническое задание на создание автоматизированной системы") является стандартом, который определяет общие требования к разработке автоматизированных систем (АС). Он используется в разных отраслях, включая медицину, и применим для разработки автоматизированных информационных систем (АИС).

2. ГОСТ 34.201—2020. Межгосударственный стандарт. Информационные технологии. Комплекс стандартов на автоматизированные системы. Виды, комплектность и обозначение документов при создании автоматизированных систем: Приказом Федерального агентства по техническому регулированию и метрологии № 1521-ст от 19 ноября 2021 г.: дата введения 2022-01-01. – М.: Российский институт стандартизации, 2021. – 10 с.

3. Блоштин А.В., Лаптев Д.В. Современные подходы к проектированию клиент-серверных приложений // Программные системы: теория и приложения. 2021. №2. URL: https://elibrary.ru/item.asp?id=46523678 (дата обращения: 26.11.2024)

4. Android Developers — официальный сайт документации по разработке на Kotlin: https://developer.android.com/kotlin (дата обращения: 26.11.2024).

5. Kotlin Documentation — полный справочник по языку Котлин: https://kotlinlang.org/docs (дата обращения: 26.11.2024).

6. Жемеров С., Исакова С. Kotlin в действии. 2-е издание. М.: Питер, 2022. URL: https://www.piter.com/product/kotlin-v-dejstvii (дата обращения: 26.11.2024).

7. Документация JUnit — для тестирования Kotlin-приложений: https://junit.org/junit5 (дата обращения: 26.11.2024).