МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет
имени К.И. Сатпаева

Институт автоматики и информационных технологий

Кафедра «Программная инженерия»

Сон Вячеслав Владимирович

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к дипломному проекту

Образовательная программа 6B06102 - Computer Science

Алматы 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет имени К.И.Сатпаева

Институт автоматики и информационных технологий

Кафедра «Программная инженерия»

ДОПУЩЕН К ЗАЩИТЕ
Заведующий кафедрой ПИ
канд. тех. наук, ассоц.профессор
_____ Ф.Н. Абдолдина
"30" 05 _____ 2025 г.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к дипломному проекту

На тему: «Приложение для обмена мыслями и эмоциями»

Образовательная программа: 6B06102 Computer Science

Выполнил	ФИО
Рецензент PhD, ассоциированный профессор _____Г.Б.Кашаганова "28" 05 _____ 2025 г.	Научный руководитель PhD, ассоциированный профессор _____Ж.М. Алибиева "28" 05 _____ 2025 г.

Алматы 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет имени К.И.Сатпаева

Институт автоматики и информационных технологий

Кафедра «Программная инженерия»

УТВЕРЖДАЮ
Заведующий кафедрой ПИ
канд. тех. наук,
ассоц.профессор
_____ Ф.Н.Абдолдина
"30" 05 2025 г.

ЗАДАНИЕ
на выполнение дипломного проекта

Обучающемуся: *Сон Вячеслав Владимирович*
Тема: Создание приложения для обмена мыслями и эмоциями
Утверждена приказом проректора по академической работе:

Срок сдачи законченного проекта	«_____»_____2025 г.

Исходные данные к дипломному проекту:
А) Проведение анализа предметной области: исследование современных социальных сетей и платформ;
Б) Реализация серверной части (Backend) платформы;
В) Реализация клиентской части (Frontend) платформы;
Г) Реализация функций взаимодействия пользователей с контентом;
Д) Тестирование разработанной веб-платформы и анализ результатов;
Перечень подлежащих разработке в дипломном проекте вопросов: (с точным указанием обязательных чертежей): *представлены 19 слайда презентации.*
Рекомендуемая основная литература: *из19 наименований.*

ГРАФИК

подготовки дипломного проекта

Наименование разделов, перечень разрабатываемых вопросов	**Сроки представления научному руководителю и консультантам**	**Примечание**
1. Анализ предметной области и существующих решений	04.10.2024 – 02.11.2024	Выполнено
2. Проектирование веб-платформы	11.11.2024 – 19.02.2025	Выполнено
3. Реализация серверной части (Backend)	24.02.2025 – 05.03.2025	Выполнено
4. Реализация клиентской части (Frontend)	06.03.2025 – 29.03.2025	Выполнено
5. Реализация дополнительного функционала	01.04.2025 – 24.04.2025	Выполнено
6. Тестирование и подготовка документации	02.05.2025 – 10.05.2025	Выполнено

Подписи

консультантов и нормоконтролера на законченную дипломный проект
с указанием относящихся к ним разделов проекта

Наименования разделов	**Консультанты, И.О.Ф. (уч. степень, звание)**	**Дата подписания**	**Подпись**	**Оценка**
Программное обеспечение	Б.Б.Досанов ст. преподаватель, магистр.	23.05.25		
Нормоконтролер	Джунусова.С.М ст. преподаватель, магистр.	23.05.25		

Научный
руководитель _____ Алибиева.Ж.М.

Задание принял к исполнению обучающийся _____ Сон В.В.

Дата «24» 09 2024г.

АҢДАТПА

Бұл дипломдық жоба минимализмге, пайдалану ыңғайлылығына және деректердің құпиялылығына бағытталған мәтіндік өзін-өзі көрсетуге арналған веб-платформаны әзірлеуге және іске асыруға арналған. Жобаның өзектілігі қазіргі заманғы әлеуметтік желілердің контенттің шамадан тыс жүктелуі, мультимедиялық мазмұнның көптігі, жарнаманың тым көп болуы және пайдаланушылардың өз жарияланымдарын жеткіліксіз бақылауы сияқты мәселелеріне байланысты.

Жобаның мақсаты - көрсетілген кемшіліктерден ада, пайдаланушылар ойларымен ыңғайлы бөлісе алатын, жеке жазбалар жүргізе алатын және тақырыптық мазмұнды таба алатын балама онлайн-кеңістік құру болды.

Жобаны орындау барысында Go тілінде Gin фреймворкы және PostgreSQL дерекқорымен өзара әрекеттесу үшін GORM ORM пайдаланылған бэкенд, сондай-ақ TypeScript және Vite қолданылған React-тағы фронтенд қамтитын толық функционалды жүйе жобаланып, іске асырылды. Іске асырылған негізгі функцияларға мыналар жатады: JWT арқылы пайдаланушыларды қауіпсіз тіркеу және аутентификациялау, мәтіндік посттарды құру, өңдеу және жою, олардың құпиялылық деңгейін басқару мүмкіндігі, мазмұнды ұйымдастыруға арналған тегтеу жүйесі, тегтер мен кілт сөздер бойынша посттарды іздеу, мазмұнды сүзу және сұрыптау, лайктар жүйесі және посттарды "Таңдаулыларға" қосу, сондай-ақ пайдаланушы деректерін JSON форматында экспорттау және безендіру тақырыбын өзгерту (ашық/күңгірт) сияқты аккаунтты басқару функциялары.

Әзірленген платформа қолмен тестілеу кезеңінен сәтті өтіп, оның жұмысқа қабілеттілігін және мәлімделген талаптарға сәйкестігін растады. Жоба қарапайымдылыққа, қауіпсіздікке және мазмұнға бағытталған пайдаланушылардың заманауи сұраныстарына жауап беретін сұранысқа ие мәтіндік сервисті құру мүмкіндігін көрсетеді.

АННОТАЦИЯ

. Данный дипломный проект посвящен разработке и реализации веб-платформы, предназначенной для текстового самовыражения пользователей, с акцентом на минимализм интерфейса, удобство использования и обеспечение приватности данных. Актуальность проекта обусловлена проблемами современных социальных сетей, такими как информационная перегруженность, избыток мультимедийного контента, навязчивая реклама и недостаточный контроль пользователей над своими публикациями.

Целью проекта являлось создание альтернативного онлайн-пространства, свободного от указанных недостатков, где пользователи могли бы комфортно делиться мыслями, вести личные записи и находить тематический контент.

В ходе выполнения проекта была спроектирована и реализована полнофункциональная система, включающая бэкенд на языке Go с использованием фреймворка Gin и ORM GORM для взаимодействия с базой данных PostgreSQL, а также фронтенд на React с применением TypeScript и Vite. Ключевые реализованные функции включают: безопасную регистрацию и аутентификацию пользователей с помощью JWT, создание, редактирование и удаление текстовых постов с возможностью управления их приватностью, систему тегирования для организации контента, поиск постов по тегам и ключевым словам, фильтрацию и сортировку контента, систему лайков и добавления постов в "Избранное", а также функции управления аккаунтом, такие как смена пароля, экспорт пользовательских данных в формате JSON и персонализация интерфейса через смену тем оформления (светлая/темная).

Разработанная платформа успешно прошла этап ручного тестирования, подтвердив свою работоспособность и соответствие заявленным требованиям. Проект демонстрирует возможность создания востребованного текстового сервиса, отвечающего современным запросам пользователей на простоту, безопасность и сфокусированность на контенте.

ABSTRACT

This diploma project is dedicated to the development and implementation of a web platform for text-based self-expression, focusing on interface minimalism, usability, and data privacy. The relevance of the project stems from the issues of modern social networks, such as information overload, excessive multimedia content, intrusive advertising, and insufficient user control over their publications.

The aim of the project was to create an alternative online space, free from these shortcomings, where users could comfortably share thoughts, maintain personal records, and find thematic content.

During the project, a fully functional system was designed and implemented, including a backend in Go using the Gin framework and GORM ORM for interaction with a PostgreSQL database, as well as a frontend in React using TypeScript and Vite. Key implemented functions include: secure user registration and authentication using JWT, creation, editing, and deletion of text posts with the ability to manage their privacy, a tagging system for content organization, searching posts by tags and keywords, content filtering and sorting, a system of likes and adding posts to "Favorites", as well as account management functions such as password change, export of user data in JSON format, and interface theme customization (light/dark).

The developed platform has successfully passed the manual testing phase, confirming its functionality and compliance with the stated requirements. The project demonstrates the feasibility of creating a sought-after text-based service that meets modern user demands for simplicity, security, and content focus.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире цифровые технологии и социальные медиа играют неотъемлемую роль в коммуникации и обмене информацией. Ежедневно миллионы пользователей обращаются к различным онлайн-платформам для общения, получения новостей, публикации собственного контента и самовыражения. Однако, несмотря на предоставляемые ими широкие возможности, большинство популярных социальных сетей и блоговых платформ сталкиваются с рядом существенных недостатков. К ним относятся избыточное количество мультимедийного контента (видео, изображения), которое часто отвлекает от текстовой информации, навязчивая реклама, интегрированная в ленты новостей и контент, а также ограниченный контроль пользователей над приватностью своих публикаций и использованием их персональных данных. Многие крупные платформы в первую очередь ориентированы на удержание внимания пользователей и монетизацию за счет рекламы, что не всегда совпадает с интересами пользователей в удобном, безопасном и сфокусированном взаимодействии.

Существующие решения часто страдают от следующих проблем:

- перегруженность контента: В лентах социальных сетей преобладают визуальные материалы и рекламные вставки, что затрудняет восприятие текстовой информации и приводит к информационной усталости.

- сложность ведения личных записей: Большинство платформ нацелены на публичные взаимодействия и создание контента для широкой аудитории, в то время как инструменты для приватного самовыражения, ведения личного дневника или заметок часто вторичны или неудобны.

- недостаточный контроль над анонимностью и данными: Пользовательские данные активно собираются и могут быть использованы в коммерческих или иных целях без полного и прозрачного контроля со стороны пользователя.

- влияние рекламы и монетизации: Крупные сервисы строят свою бизнес-модель на показе рекламы, что неизбежно влияет на пользовательский опыт, отвлекая и снижая качество взаимодействия с платформой.

В связи с этим, актуальность разработки альтернативных платформ, которые предлагают иной подход к онлайн-общению и публикации контента, значительно возрастает. Наблюдается запрос пользователей на более простые, минималистичные и контролируемые пространства, где можно было бы сосредоточиться на текстовом самовыражении, обмене мыслями и идеями без отвлекающих факторов.

Целью данного дипломного проекта является разработка удобной и минималистичной текстовой платформы для публикации мыслей и идей, где пользователи смогут:

- делать публичные и приватные записи, управляя видимостью своего контента.

- фильтровать и находить контент по тегам и ключевым словам.

- читать публичную ленту без обязательной регистрации, снижая барьер входа.

- использовать платформу без рекламы и избыточного мультимедийного контента, фокусируясь на тексте.

Для достижения поставленной цели в ходе дипломного проектирования были решены следующие основные задачи:

- проведен анализ предметной области, изучены существующие аналогичные платформы, выявлены их преимущества и недостатки для определения уникальных особенностей и конкурентных преимуществ разрабатываемого решения.

- разработано развернутое техническое задание, определяющее функциональные и нефункциональные требования к программной системе, включая требования к пользовательскому интерфейсу, безопасности и производительности.

- обоснован выбор стека технологий для реализации бэкенд-части (Go, Gin, GORM), фронтенд-части (React, Vite) и системы управления базами данных (PostgreSQL), а также механизма аутентификации (JWT).

- спроектирована архитектура программной системы, включая логическую схему базы данных (ERD), диаграмму классов (Go-структур), диаграмму компонентов и диаграмму вариантов использования для визуализации структуры и функциональности.

- реализована ключевая функциональность бэкенда, включая систему регистрации и аутентификации пользователей, управление созданием, редактированием, удалением и просмотром постов, систему тегирования, поиска по тегам и ключевым словам, а также функционал лайков и добавления постов в избранное.

- разработан пользовательский интерфейс фронтенда, обеспечивающий интуитивно понятное взаимодействие с реализованной функциональностью бэкенда, включая формы ввода, отображение списков постов, навигацию и управление пользовательским профилем (смена пароля, экспорт данных, смена темы оформления).

- проведено ручное тестирование основных пользовательских сценариев для проверки работоспособности и соответствия системы заявленным требованиям.

Структура данной пояснительной записки отражает последовательность выполнения указанных задач. Теоретический раздел (Раздел 1) представляет анализ предметной области, сравнение с аналогами и обоснование выбора технологического стека. Проектный раздел (Раздел 2) подробно описывает проектирование архитектуры, базы данных, API, а также ключевые аспекты

реализации бэкенда и фронтенда [1]. Прикладной раздел (Раздел 3) посвящен результатам тестирования и оценке достижения поставленных задач. В заключении подводятся итоги проделанной работы и намечаются перспективы дальнейшего развития проекта. Приложения содержат графические материалы, включая диаграммы и листинги кода.

1 Теоретический раздел (Исследовательско-технологический)

1.1 Общее описание предметной области и анализ существующих решений

Современное общество характеризуется глубокой интеграцией цифровых технологий во все сферы жизни, и онлайн-коммуникации занимают в этом процессе центральное место. Платформы для обмена информацией, социального взаимодействия и публикации контента стали неотъемлемой частью повседневности, предоставляя пользователям беспрецедентные возможности для самовыражения, обучения и поддержания связей. Однако, по мере развития этих платформ, выявляется ряд системных проблем и неудовлетворенных потребностей пользователей, что открывает пространство для создания новых, более специализированных и ориентированных на конкретные нужды сервисов.

1.1.1 Современные тенденции в онлайн-коммуникациях и публикации контента: проблемы и вызовы

Эволюция онлайн-платформ привела к формированию сложных экосистем, где доминируют несколько ключевых тенденций, оказывающих значительное влияние на пользовательский опыт:

- доминирование визуального контента и клиповое мышление: наблюдается явный сдвиг в сторону визуальных форматов - изображений, коротких и длинных видео (TikTok, Instagram Reels, YouTube). Это приводит к тому, что текстовая информация часто отходит на второй план, а у пользователей формируется привычка к быстрому потреблению легко усваиваемого, но не всегда глубокого контента. Платформы, перегруженные медиа, могут вызывать информационную усталость и затруднять концентрацию на смысле.

- влияние алгоритмических лент: Большинство крупных платформ используют сложные алгоритмы для формирования персонализированных лент новостей и рекомендаций. Хотя это может повысить релевантность предлагаемого контента, такой подход также создает "фильтрационные пузыри", ограничивая пользователя рамками его предполагаемых интересов и снижая его контроль над информационным потоком. Хронологический или тематический порядок часто уступает место непрозрачным механизмам ранжирования.

- "экономика внимания" и информационная перегрузка: Платформы активно борются за время и внимание пользователей, применяя различные механики удержания: постоянные уведомления, геймификацию, бесконечные ленты прокрутки. Это ведет к избыточному потреблению информации,

цифровому стрессу и снижению продуктивности.

- монетизация через рекламу и вопросы конфиденциальности: Основной бизнес-моделью многих популярных сервисов является таргетированная реклама, основанная на сборе и анализе больших объемов пользовательских данных. Это вызывает закономерные опасения относительно конфиденциальности, безопасности персональной информации и ее возможного нецелевого использования. Навязчивая реклама также напрямую ухудшает пользовательский опыт.

- поверхностность взаимодействия: Форматы, ориентированные на быструю реакцию (лайки, репосты, короткие комментарии), не всегда способствуют глубокому и осмысленному диалогу или рефлексии. Это может приводить к упрощению дискуссий и снижению качества коммуникации.

- сложность и избыточность функционала: В погоне за удержанием аудитории многие платформы постоянно расширяют свой функционал, что приводит к усложнению интерфейсов и отходу от первоначальных, более простых целей.

На фоне этих тенденций формируется запрос на альтернативные пространства, которые предлагают иной пользовательский опыт: более спокойный, сфокусированный, контролируемый и уважающий приватность пользователя. Возрастает ценность платформ, где можно избежать информационного шума и сосредоточиться на качественном контенте или глубоком самовыражении, особенно в текстовом формате, который остается ключевым инструментом для передачи сложных идей и нюансированных эмоций.

1.1.2 Анализ аналогов и их характеристики

Для определения рыночной ниши и уникальных преимуществ разрабатываемой текстовой платформы был проведен анализ ряда существующих популярных решений, предназначенных для публикации контента и социального взаимодействия. В качестве основных объектов для сравнения были выбраны Twitter (X), Reddit и Medium, как платформы с различным фокусом, но так или иначе работающие с текстовым контентом.

– twitter (X)

– описание: Глобальная платформа для микроблогинга, позволяющая пользователям публиковать короткие сообщения ("твиты"), обмениваться новостями и мнениями в режиме реального времени.

- сильные стороны: Оперативность, широкий охват, возможность следить за событиями и лидерами мнений, простота публикации коротких сообщений.

- недостатки с точки зрения целей проекта:

- перегруженность медиа и рекламой: Лента насыщена визуальным контентом и рекламными вставками, что отвлекает от текстовой информации.

- ориентация на публичность и краткость: Формат коротких сообщений и публичный характер платформы не всегда подходят для развернутого самовыражения или ведения приватных записей.

- алгоритмическая лента: снижает контроль пользователя над информационным потоком.

- высокий уровень "шума": Большое количество информационного мусора и токсичных взаимодействий.

- reddit

- описание: Социальная новостная платформа и форум, организованная вокруг тематических сообществ ("сабреддитов"), где пользователи делятся контентом и участвуют в обсуждениях.

- сильные стороны: Огромное разнообразие тематических сообществ, возможность глубоких и специализированных дискуссий, система пользовательского голосования за контент.

- недостатки с точки зрения целей проекта:

- сложность интерфейса: может быть высоким порог входа для новых пользователей.

- фокус на сообществах: Платформа в большей степени ориентирована на взаимодействие внутри сабреддитов, а не на индивидуальное ведение блога или дневника.

- зависимость от модерации и алгоритмов: Видимость контента сильно зависит от правил конкретных сообществ и алгоритмов платформы.

- наличие медиа и рекламы: также присутствует значительное количество нетекстового контента и рекламы.

- medium

- описание: Платформа для публикации и чтения статей, эссе и историй, часто более длинного и аналитического формата.

- сильные стороны: Высокое качество контента, удобный редактор, фокус на содержательных текстах, возможность для авторов монетизировать свой труд.

- недостатки с точки зрения целей проекта:

- paywall (платный доступ): Значительная часть контента доступна только по платной подписке, что противоречит идее свободного доступа.

- ориентация на длинные форматы: менее подходит для коротких заметок, мыслей или дневниковых записей.

- отсутствие явной функции приватных записей: Платформа в первую очередь публичная.

- сложность для простого самовыражения: может показаться избыточной для пользователей, ищущих простое пространство без фокуса на профессиональной публикации.

1.1.3 Обоснование необходимости разработки собственного решения

На основе выявленных проблем существующих платформ и анализа потребностей определенной части аудитории, разработка собственной минималистичной текстовой платформы представляется целесообразной и востребованной. Необходимость такого решения обусловлена следующими факторами:

1. Запрос на информационный детокс и концентрацию: В условиях постоянной информационной перегрузки растет число пользователей, ищущих "тихие гавани" в интернете - места, где можно сосредоточиться на чтении и письме без отвлекающих уведомлений, рекламы и бесконечных лент мультимедийного контента. Разрабатываемая платформа предлагает именно такое пространство, ставя во главу угла исключительно текстовый контент.

2. Потребность в простоте и минимализме: Сложные интерфейсы и избыточный функционал многих современных платформ отталкивают пользователей, ценящих простоту и интуитивность. Проект нацелен на создание максимально простого и понятного инструмента, который не требует длительного освоения и позволяет сразу перейти к сути - созданию и чтению текстовых записей.

3. Важность приватности и личного пространства: Не все мысли и идеи предназначены для публичного обозрения. Существует устойчивая потребность в удобном и безопасном инструменте для ведения личного дневника, заметок или рефлексий. Разрабатываемая платформа предоставляет пользователю полный контроль над видимостью каждой своей записи, позволяя легко переключаться между публичным и приватным режимами.

4. Альтернатива коммерциализированным платформам: Отсутствие рекламы и фокуса на монетизации пользовательских данных является ключевым преимуществом, создающим более доверительную и комфортную среду для самовыражения.

5. Низкий порог входа для потребления контента: Возможность читать публичные посты без обязательной регистрации делает платформу более открытой и доступной для широкой аудитории.

Таким образом, разрабатываемый проект занимает нишу, которая недостаточно покрыта существующими крупными игроками рынка. Он предлагает уникальное сочетание простоты, фокуса на тексте, контроля над приватностью и отсутствия коммерческого давления.

Ключевые отличительные особенности и преимущества проектируемой платформы:

- исключительно текстовый формат: обеспечивает среду, свободную от визуального шума, способствуя глубокому погружению в чтение и письмо.

- минималистичный дизайн и интуитивный интерфейс: снижает когнитивную нагрузку и делает платформу доступной для пользователей с любым уровнем технической подготовки.

- гибкое управление приватностью: позволяет пользователям легко определять видимость каждой своей записи (публичная или приватная).

- отсутствие рекламы и трекинга: гарантирует чистый пользовательский опыт и уважение к данным пользователя.

- свободный доступ к публичному контенту: Чтение публичных постов не требует регистрации.

- организация контента: Возможность использования тегов и поиска по ключевым словам и тегам для удобной навигации.

- интерактивность: Реализованы функции лайков и добавления постов в "Избранное" для базового социального взаимодействия.

Целевая аудитория проекта:

- люди, сознательно выбирающие текстовый формат общения и ценящие минимализм в цифровых продуктах.

- авторы, блогеры, писатели, ищущие простую и не отвлекающую площадку для публикации своих мыслей, заметок, эссе или коротких историй.

- пользователи, ведущие личные дневники, журналы рефлексии или просто нуждающиеся в удобном инструменте для фиксации идей.

- индивиды, ищущие альтернативу крупным социальным сетям, с большим контролем над своими данными и потребляемым контентом.

Разработка данной платформы направлена на удовлетворение потребностей этой аудитории, предлагая им ценный и уникальный инструмент для текстового самовыражения и взаимодействия.

1.2 Обоснование выбора технологий и методов разработки

Выбор технологического стека и методологии разработки является фундаментальным этапом, определяющим эффективность процесса создания программного продукта, его качество, производительность, масштабируемость и удобство дальнейшей поддержки. Для реализации минималистичной текстовой платформы были выбраны современные и проверенные технологии, обеспечивающие оптимальное сочетание указанных характеристик.

1.2.1 Выбор архитектурного подхода: Клиент-серверная архитектура и REST API

Для разрабатываемой веб-платформы была выбрана классическая клиент-серверная архитектура. Данный подход предполагает разделение системы на две основные взаимодействующие части:

- клиентская часть (Frontend): реализуется как одностраничное приложение (SPA - Single Page Application), работающее в браузере пользователя. Отвечает за пользовательский интерфейс (UI), взаимодействие с пользователем и отправку запросов на сервер.
- серверная часть (Backend): отвечает за обработку запросов от клиента, реализацию бизнес-логики, аутентификацию и авторизацию пользователей, а также за взаимодействие с системой управления базами данных.

Взаимодействие между клиентской и серверной частями осуществляется по протоколу HTTP/HTTPS с использованием архитектурного стиля REST (Representational State Transfer) [2]. Данные передаются в формате JSON (JavaScript Object Notation).

Преимущества выбранного архитектурного подхода:

- четкое разделение ответственностей: Позволяет независимо разрабатывать, тестировать и масштабировать клиентскую и серверную части.
- гибкость и расширяемость: Изменения в одной части системы (например, редизайн пользовательского интерфейса) могут производиться с минимальным влиянием на другую часть, при условии сохранения стабильности API.
- возможность использования различных технологий: Для фронтенда и бэкенда могут быть выбраны наиболее подходящие для их задач языки программирования и фреймворки.
- поддержка различных клиентов: хорошо спроектированный REST API может в будущем использоваться не только веб-приложением, но и другими клиентами (например, мобильными приложениями).

1.2.2 Обоснование выбора технологий для бэкенда: Go, Gin, GORM, PostgreSQL

Для реализации серверной части (бэкенда) платформы был выбран следующий стек технологий: язык программирования Go, веб-фреймворк Gin, ORM-библиотека GORM и система управления базами данных PostgreSQL.

- язык программирования Go (Golang)
- обоснование: Go, разработанный компанией Google, является компилируемым, статически типизированным языком программирования, который отлично подходит для создания высокопроизводительных и масштабируемых сетевых приложений и веб-сервисов [3]. Ключевыми

преимуществами Go, обусловившими его выбор для данного проекта, являются: высокая производительность благодаря компиляции в машинный код; эффективный параллелизм за счет горутин и каналов, что важно для обработки множества одновременных запросов; простой и лаконичный синтаксис, ускоряющий разработку и поддержку; статическая типизация, повышающая надежность; обширная стандартная библиотека и быстрая компиляция [4].

- сравнение с альтернативами: По сравнению с Python (Django/Flask), Go выигрывает в производительности при высоких нагрузках. В отличие от Node.js (Express.js), Go эффективнее справляется с CPU-bound операциями и предлагает преимущества статической типизации. Платформы Java (Spring) или C# (.NET) являются более ресурсоемкими и могут быть избыточными для данного проекта.

- веб-фреймворк Gin

- обоснование: Gin - это высокопроизводительный HTTP-веб-фреймворк для Go. Он выбран за свою скорость, минималистичность (не навязывает излишнюю структуру) и удобный API для маршрутизации, обработки middleware и рендеринга JSON [5].

- альтернативы: Стандартный пакет net/http в Go требует больше шаблонного кода. Gin предлагает хороший баланс производительности и простоты по сравнению с другими Go-фреймворками, такими как Echo или Fiber.

- ORM (Object-Relational Mapper) GORM

- обоснование: GORM значительно упрощает взаимодействие с реляционными базами данных, позволяя работать с объектами Go вместо написания SQL-запросов. Он поддерживает удобное определение связей, автоматические миграции и хуки (например, для хеширования паролей) [6].

- альтернативы: Прямое использование database/sql требует больше кода и более подвержено ошибкам. GORM был выбран за популярность и набор функций.

- система Управления Базами Данных (СУБД) PostgreSQL

- обоснование: PostgreSQL - это мощная, надежная и функциональная объектно-реляционная СУБД. Ее преимущества: поддержка транзакций ACID, расширяемость, поддержка сложных запросов и различных типов данных, хорошая масштабируемость и соответствие стандартам SQL [7].

- альтернативы: MySQL, хотя и популярен, часто уступает PostgreSQL в функциональности и строгости. SQLite не подходит для многопользовательских веб-приложений. NoSQL базы данных (например, MongoDB) менее подходят для структурированных реляционных данных данного проекта [8].

выбранный стек технологий для бэкенда обеспечивает высокую производительность, надежность, удобство разработки и хорошие перспективы для масштабирования приложения.

1.2.3 Обоснование выбора технологий для фронтенда: React, TypeScript, Vite, CSS-переменные

Для разработки клиентской части (фронтенда) платформы был выбран следующий набор технологий: библиотека React, язык TypeScript, инструмент сборки Vite и подход к стилизации с использованием CSS-переменных и инлайн-стилей.

- библиотека React

- обоснование: React является одной из самых популярных JavaScript-библиотек для создания пользовательских интерфейсов. Его ключевые преимущества: компонентный подход, упрощающий разработку и поддержку; Virtual DOM для оптимизации обновлений и повышения производительности; однонаправленный поток данных; большое сообщество и обширная экосистема библиотек (для управления состоянием, маршрутизации, работы с формами) [9].

- альтернативы: Angular (более комплексный и с высоким порогом входа), Vue.js (хорошая альтернатива, но React выбран из-за большей распространенности), Svelte (компилятор с меньшим сообществом).

- язык TypeScript

- обоснование: TypeScript, добавляя статическую типизацию к JavaScript, позволяет выявлять ошибки на этапе компиляции, улучшает читаемость и поддержку кода, делает рефакторинг более надежным и обеспечивает лучшую поддержку в IDE [10].

- инструмент сборки Vite

- обоснование: Vite обеспечивает значительно более быстрый сервер разработки (благодаря нативным ES-модулям и HMR) и оптимизированную сборку для продакшена (с использованием Rollup) по сравнению с традиционными сборщиками, такими как Webpack в его стандартной конфигурации [11].

- альтернативы: Webpack (более сложная настройка), Create React App (менее быстрый в разработке).

- подход к стилизации (CSS-переменные и инлайн-стили)

- обоснование: Использование глобальных CSS-переменных (в index.css) позволяет легко реализовывать и переключать темы оформления (светлая/темная) путем изменения значений переменных. Инлайн-стили (объекты React.CSSProperties) применяются для специфической стилизации отдельных компонентов. Такой подход обеспечивает гибкость и простоту управления темами.

- альтернативы: CSS Modules, CSS-in-JS (Styled Components, Emotion), Utility-first CSS (Tailwind CSS, UnoCSS). Выбранный подход был сочтен достаточным для данного проекта.

Выбранный стек технологий для фронтенда позволяет создавать современные, интерактивные и производительные пользовательские интерфейсы.

1.2.4 Выбор механизма аутентификации: JWT

Для обеспечения безопасности и идентификации пользователей в разрабатываемой платформе был выбран механизм аутентификации на основе JSON Web Tokens (JWT) [16].

- принцип работы и процесс аутентификации: JWT представляет собой компактный, самодостаточный и подписанный JSON-объект, содержащий информацию о пользователе (claims) и срок действия. После успешной проверки учетных данных бэкенд генерирует и подписывает JWT секретным ключом, отправляя его клиенту. Клиент сохраняет токен (в localStorage) и прикрепляет его к заголовку Authorization (схема Bearer) при каждом запросе к защищенным API-ресурсам. Сервер валидирует токен по подписи и сроку действия.

- обоснование выбора JWT:

- stateless: Не требует хранения состояния сессии на сервере, что упрощает масштабирование.

- подходит для REST API: легко передается в HTTP-заголовках.

- гибкость: Полезная нагрузка может содержать необходимые данные.

- безопасность (при правильном использовании): Цифровая подпись гарантирует целостность; передача по HTTPS защищает от перехвата.

- стандарт и поддержка: является отраслевым стандартом (RFC 7519) с хорошей поддержкой библиотеками (например, github.com/golang-jwt/jwt/v5 для Go [17]).

- альтернативы: Сессии на основе cookie (stateful, сложнее в масштабировании); OAuth 2.0 / OpenID Connect (избыточны для простой внутренней аутентификации). JWT был выбран как оптимальное решение для данного проекта.

1.2.5 Технология программирования и тестирования

- технология программирования: Разработка велась с использованием модульного подхода [18], клиент-серверной архитектуры с REST API, ORM GORM для работы с БД, React Context API для управления глобальным состоянием на фронтенде. Использовалась система контроля версий Git. При написании кода уделялось внимание его читаемости и простоте [19].

- технология тестирования: применялась стратегия многоуровневого тестирования. На бэкенде планировались (и частично могли быть реализованы) модульные тесты (Go testing, testify) и интеграционные тесты (с net/http/httptest и тестовой БД) для проверки API эндпоинтов. На фронтенде возможно написание модульных тестов для компонентов (Jest, React Testing Library). Основным

методом проверки функциональности в рамках проекта являлось ручное End-to-End тестирование всех ключевых пользовательских сценариев, включая проверку UI/UX.

2 Проектный раздел (Проектно-экспериментальный)

2.1 Проектирование архитектуры и функциональности системы

В основе проекта лежит клиент-серверная архитектура, выбранная для четкого разделения логики представления и бизнес-логики, что способствует модульности и масштабируемости. Клиентская часть (Frontend), разработанная на React, отвечает за пользовательский интерфейс и взаимодействие с пользователем. Серверная часть (Backend), реализованная на Go с использованием фреймворка Gin и ORM GORM, обрабатывает бизнес-логику, управляет аутентификацией и взаимодействует с базой данных PostgreSQL. Обмен данными между клиентом и сервером осуществляется через RESTful API в формате JSON.

2.1.1 Диаграмма вариантов использования (Use Case Diagram)

Диаграмма вариантов использования (Рисунок 2.1) наглядно представляет основные функциональные возможности системы и то, как различные типы пользователей (акторы) взаимодействуют с ней. Она определяет границы системы и ключевые сценарии использования, показывая взаимодействие пользовательских действий с серверной логикой.

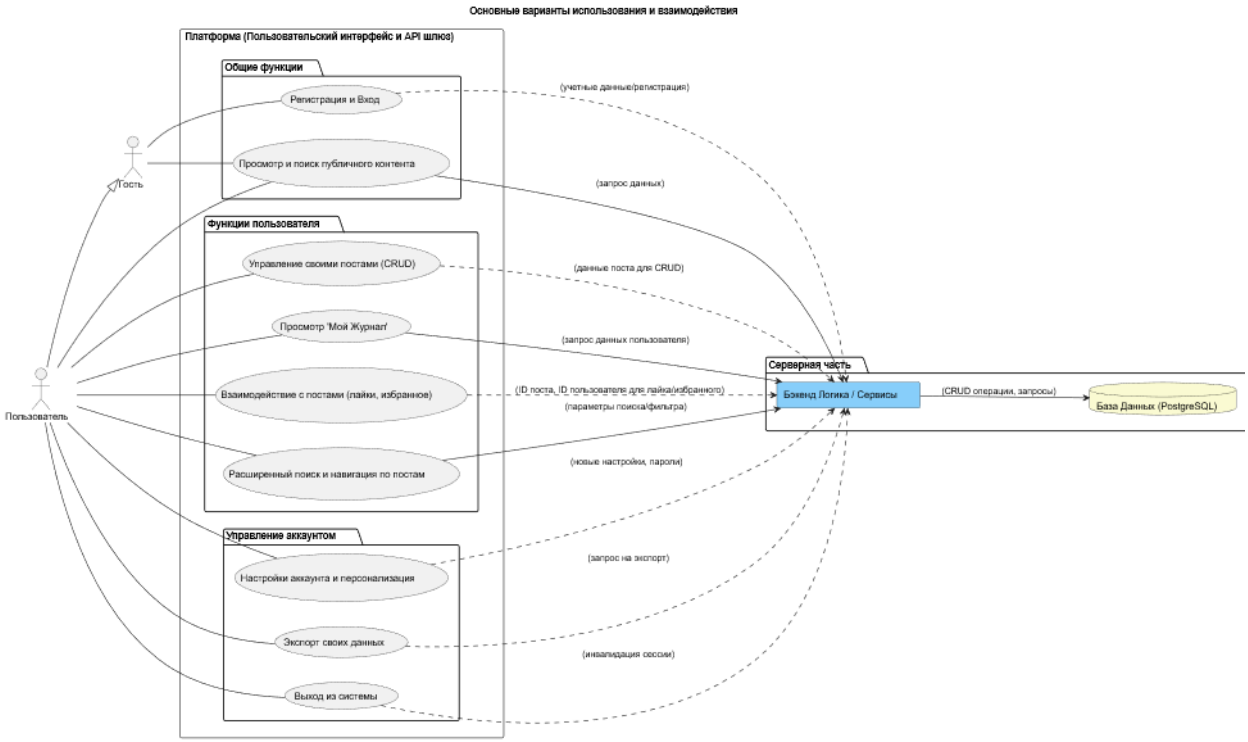


Рисунок 2.1 - Диаграмма вариантов использования

На диаграмме выделены два основных актора: Гость и Зарегистрированный пользователь. Диаграмма иллюстрирует, что Зарегистрированный пользователь наследует все возможности Гостя и обладает дополнительными правами, такими как управление контентом и своим аккаунтом. Варианты использования сгруппированы по функциональным блокам и показывают их связь с серверной логикой.

2.1.2 Диаграмма последовательности: Создание нового поста

Для иллюстрации динамического взаимодействия между различными компонентами системы при выполнении конкретного пользовательского сценария используется диаграмма последовательности. На Рисунке 2.2 представлена диаграмма для процесса создания нового поста авторизованным пользователем, демонстрирующая пошаговый обмен сообщениями между участниками взаимодействия.

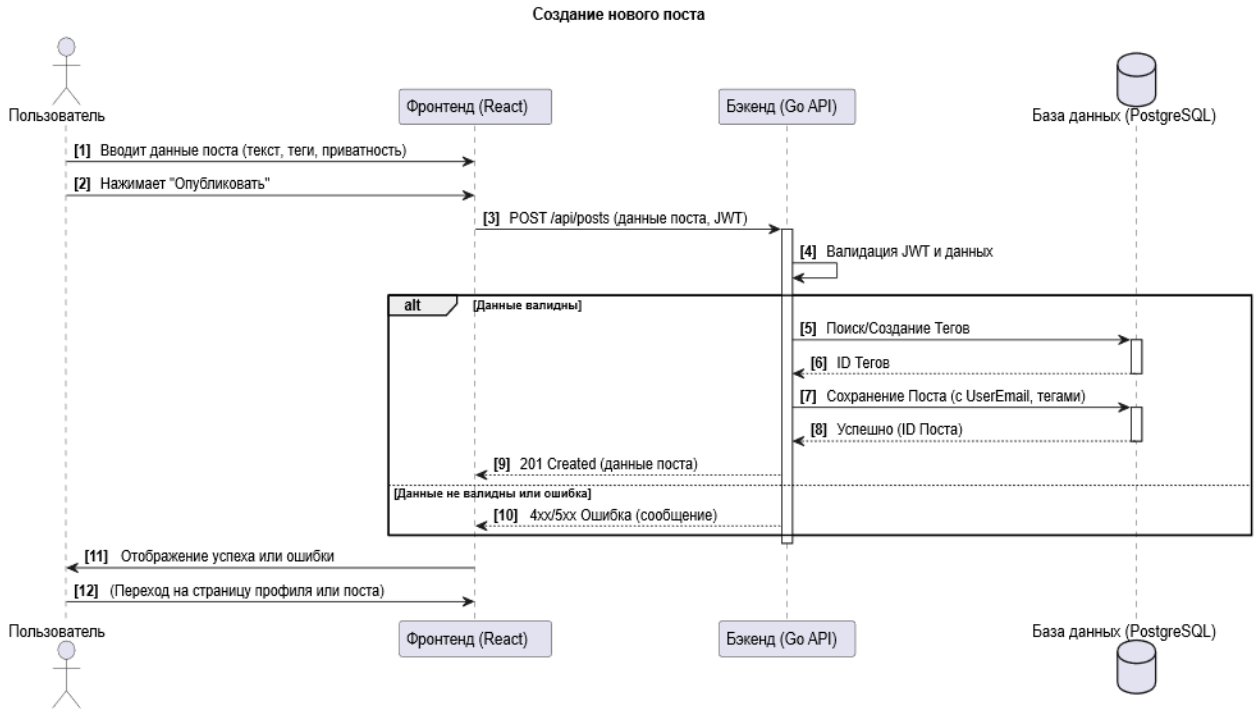


Рисунок 2.2 - Диаграмма последовательности: Создание нового поста

Рисунок 2.2 показывает, как пользователь через Фронтенд инициирует создание поста. Фронтенд отправляет данные и JWT на Бэкенд, который валидирует информацию, обрабатывает теги, сохраняет пост в Базу данных и возвращает Фронтенду результат. Эта диаграмма помогает понять временной порядок и зависимости вызовов.

2.1.3 Диаграмма активности: Регистрация нового пользователя

Диаграмма активности (Рисунок 2.3) используется для моделирования потока работ или операций внутри системы для конкретного процесса. На Рисунке 2.3 показан детализированный поток действий для процесса

"Регистрация нового пользователя".

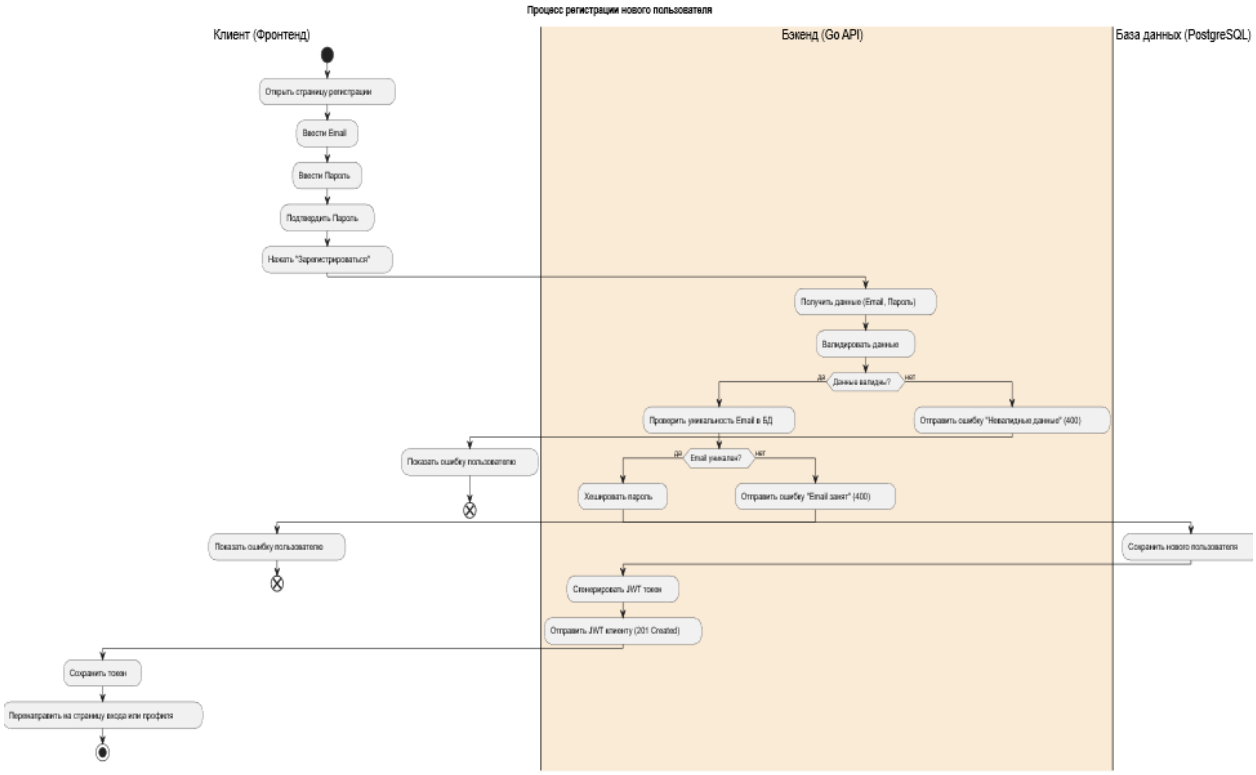


Рисунок 2.3 - Диаграмма активности: Регистрация нового пользователя

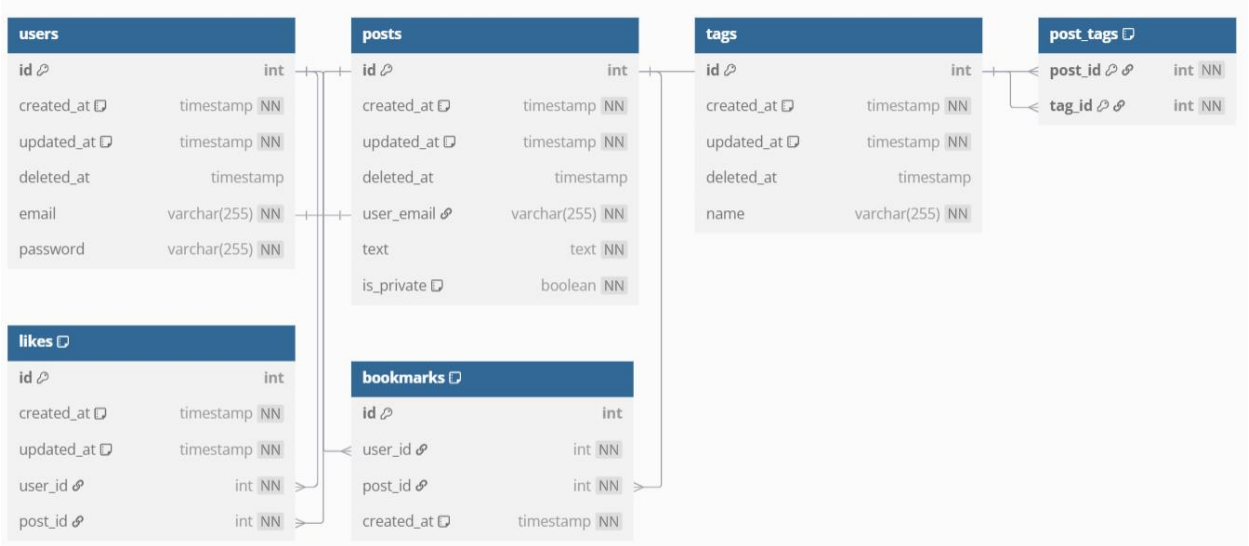
Диаграмма на Рисунке 2.3 иллюстрирует шаги от ввода данных на Фронтенде, их отправку на Бэкенд для валидации и проверки уникальности email, до хеширования пароля, сохранения пользователя в Базе Данных, генерации JWT и информирования пользователя об итоге операции. Эта диаграмма наглядно показывает логику процесса и возможные пути его выполнения.

2.2 Проектирование базы данных и моделей данных

Центральным элементом хранения информации в платформе является реляционная база данных PostgreSQL. Структура базы данных была тщательно спроектирована для эффективного хранения данных о пользователях, их текстовых записях, тегах, а также о взаимодействиях пользователей с контентом, таких как лайки и добавление в избранное.

2.2.1 Структура базы данных (ERD)

Логическая схема данных (ERD) определяет основные сущности системы и связи между ними. Ключевыми таблицами являются: users (информация о пользователях), posts (текстовые записи), tags (справочник тегов), likes (информация о лайках) и bookmarks (записи об избранных постах). Для реализации связи "многие-ко-многим" между постами и тегами используется промежуточная таблица post_tags. Внешние ключи и уникальные индексы обеспечивают целостность данных и определяют отношения между таблицами, такие как принадлежность поста пользователю, связь лайка с пользователем и постом, и аналогично для закладок. Временные метки (created_at, updated_at, deleted_at) используются для отслеживания жизненного цикла записей.

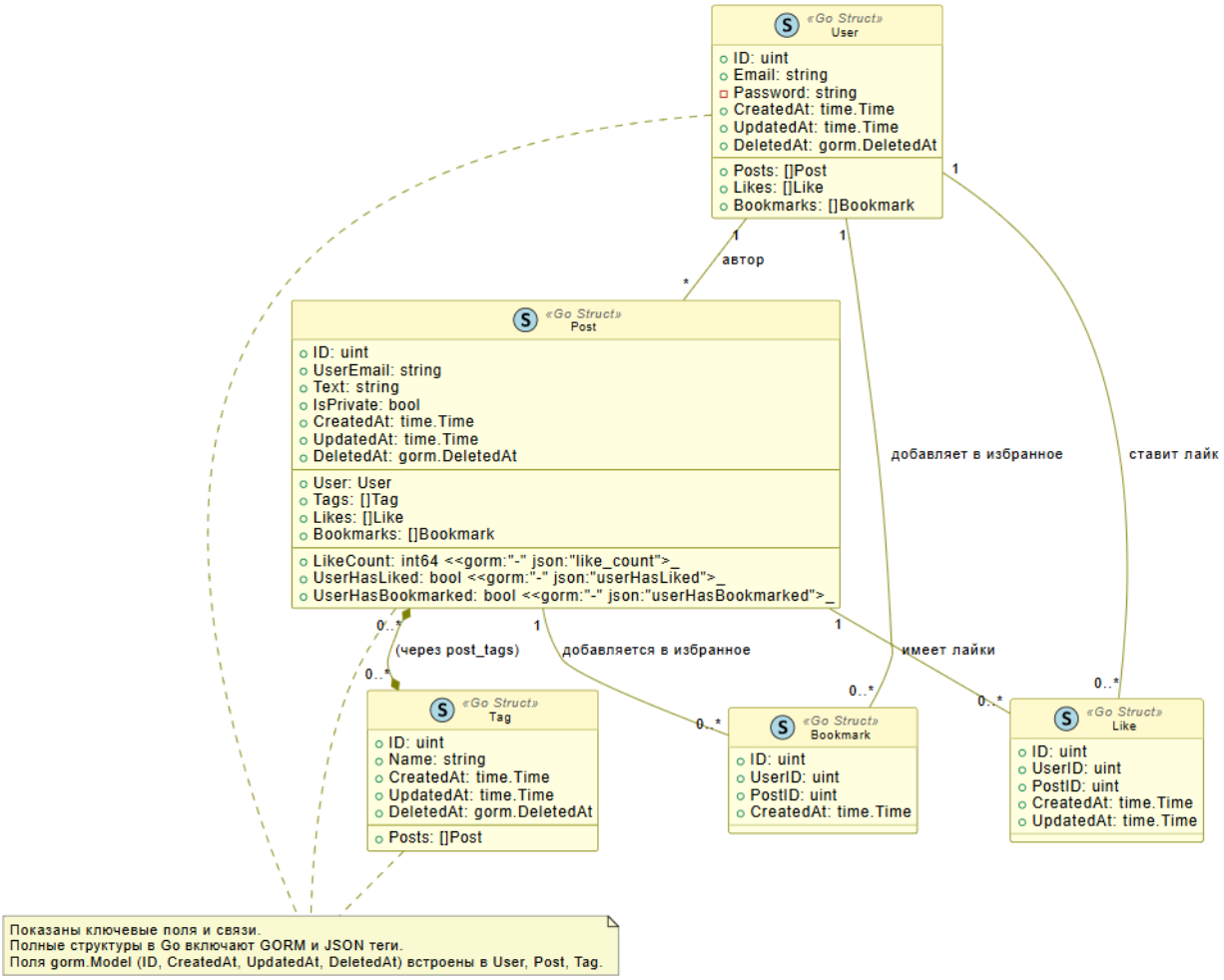


2.2.2 Модели данных на стороне сервера (Go-структуры)

На стороне бэкенда таблицы базы данных отображаются на Go-структуры, определенные в пакете models. Эти структуры используются ORM GORM для всех операций с базой данных. Основные модели включают User, Post, Tag, Like и Bookmark. Каждая структура содержит поля, соответствующие колонкам таблицы, с указанием типов данных и GORM-тегами для определения первичных ключей, внешних ключей, ограничений и связей. JSON-теги используются для управления сериализацией данных при обмене с фронтендом.

Например, структура Post включает поля для текста, приватности, связи с автором, а также срезы для связанных тегов, лайков и закладок. Дополнительно она содержит вычисляемые на сервере поля LikeCount, UserHasLiked, UserHasBookmarked, которые добавляются в JSON-ответ для удобства фронтенда. GORM-теги в коде этих структур определяют маппинг на таблицы

БД и типы связей.



2.3 Реализация серверной части (Backend)

Серверная часть приложения, разработанная на Go с использованием фреймворка Gin и ORM GORM, является ядром системы, отвечающим за реализацию бизнес-логики и предоставление RESTful API для взаимодействия с клиентской частью.

2.3.1 Маршрутизация и структура API эндпоинтов

Фреймворк Gin используется для определения маршрутов API. Все эндпоинты сгруппированы под общим префиксом /api для обеспечения единой точки входа. Основные группы маршрутов включают /api/register и /api/login для процессов аутентификации, группу /api/profile для операций, связанных с профилем текущего пользователя (получение данных профиля, смена пароля, экспорт постов, получение списков понравившихся и избранных записей), и

группу /api/posts для всех CRUD-операций над постами, а также для поиска, фильтрации и интерактивных действий (лайки, добавление в избранное). Каждый маршрут привязан к соответствующей функции-контроллеру, расположенной в пакете controllers.

2.3.2 Механизм аутентификации и авторизации

Аутентификация пользователей реализована на основе JSON Web Tokens (JWT). После успешной проверки учетных данных (email и пароль) сервер генерирует JWT, который содержит идентификатор пользователя, email и срок действия. Токен подписывается секретным ключом (JWT_SECRET_KEY, хранящимся в переменных окружения), что гарантирует его целостность. Для защиты маршрутов, требующих аутентификации, используется промежуточное ПО (AuthMiddleware). Данное middleware перехватывает входящие запросы, извлекает JWT из заголовка Authorization (по схеме Bearer), проверяет его подпись и срок действия. В случае валидного токена, информация о пользователе (UserID и Email) извлекается из утверждений (claims) токена и помещается в контекст текущего запроса Gin (c.Set("userID", ...)). Это делает данные пользователя доступными для последующих контроллеров, которые могут использовать их для реализации бизнес-логики и проверки прав доступа (авторизации). Например, при редактировании или удалении поста система проверяет, совпадает ли UserEmail поста с email аутентифицированного пользователя. Если токен невалиден, отсутствует или просрочен, AuthMiddleware прерывает запрос и возвращает клиенту HTTP-статус 401 Unauthorized.

2.3.3 Взаимодействие с базой данных и реализация бизнес-логики

Все операции с базой данных PostgreSQL осуществляются через ORM-библиотеку GORM, что позволяет работать с данными в терминах Go-структур и абстрагироваться от написания "сырых" SQL-запросов для большинства стандартных операций. Ключевые аспекты бизнес-логики включают:

- управление пользователями: Процесс регистрации включает проверку уникальности email и безопасное хеширование паролей с использованием алгоритма bcrypt перед сохранением в базу данных. При входе в систему введенный пароль сравнивается с сохраненным хешем.

- управление постами: Создание постов включает привязку к автору (на основе данных из JWT) и обработку тегов: система ищет существующие теги по имени или создает новые, после чего GORM автоматически управляет записями в связующей таблице post_tags. Редактирование и удаление постов доступны

только их авторам. Удаление постов реализовано как "мягкое" (soft delete), путем установки временной метки в поле deleted_at.

- лайки и Избранное: Функционал добавления/удаления лайка или добавления/удаления из избранного реализован по принципу "toggle". При соответствующем запросе от пользователя система проверяет наличие существующей связи (лайка или закладки) и либо создает новую запись в таблицах likes или bookmarks, либо удаляет существующую (физическое удаление с помощью Unscoped().Delete()). Эти операции выполняются в рамках транзакций базы данных для обеспечения атомарности и консистентности данных.

- поиск, фильтрация и сортировка: для эндпоинтов, возвращающих списки постов, реализована возможность динамической фильтрации по дате (год, месяц) и сортировки по различным критериям (дата создания, количество лайков). Это достигается путем анализа query-параметров HTTP-запроса и конструирования соответствующих запросов к базе данных с помощью GORM, включая использование WHERE, JOIN (например, с таблицей likes для сортировки по популярности) и ORDER BY. Также учитывается статус аутентификации пользователя для корректного отображения публичных и приватных постов. Связанные данные, такие как информация об авторе и теги, загружаются с помощью метода Preload GORM. Для каждого поста в API-ответах формируются вычисляемые поля LikeCount, UserHasLiked и UserHasBookmarked путем выполнения дополнительных запросов к соответствующим таблицам.

2.4 Реализация клиентской части (Frontend)

Пользовательский интерфейс (UI) разработан как одностраничное приложение (SPA) с использованием библиотеки React и языка TypeScript. Инструмент сборки Vite обеспечивает быструю разработку и оптимизированную сборку для продакшена.

2.4.1 Структура проекта, навигация и управление состоянием

Фронтенд-приложение имеет модульную структуру, организованную по функциональному признаку. Основные директории включают src/pages (компоненты, представляющие отдельные страницы приложения, такие как HomePage, LoginPage, ProfilePage), src/components (переиспользуемые UI-компоненты, например, Layout для общего макета, Post для отображения карточки поста, PrivateRoute для защиты маршрутов), src/api (функции для взаимодействия с REST API бэкенда), src/context (реализации React Context API для глобального управления состоянием) и src/types (определения TypeScript-

интерфейсов для данных). Навигация между страницами приложения реализована с использованием библиотеки react-router-dom. Корневой компонент App.tsx конфигурирует маршруты и оборачивает приложение в глобальные провайдеры контекста: AuthProvider для управления состоянием аутентификации пользователя (включая хранение JWT в localStorage и информацию о текущем пользователе) и ThemeContext для управления темой оформления (светлая/темная) и ее сохранения.

2.4.2 Взаимодействие с API и отображение данных

Компоненты страниц инициируют запросы к бэкенд API через специализированные асинхронные функции, определенные в модулях директории src/api (например, api/posts.ts). Эти функции используют библиотеку axios для выполнения HTTP-запросов, автоматически прикрепляя JWT к защищенным запросам. Полученные от сервера данные (в формате JSON) обрабатываются, и состояние соответствующих React-компонентов обновляется с помощью хука useState, что приводит к перерисовке пользовательского интерфейса. Обработка ошибок API-запросов также реализована для информирования пользователя.

2.4.3 Ключевые пользовательские интерфейсы и их функциональность

Пользовательский интерфейс спроектирован с акцентом на минимализм, интуитивность и удобство использования. Стилизация выполнена с использованием инлайн-стилей (React.CSSProperties) и глобальных CSS-переменных, определенных в index.css, что обеспечивает гибкую смену тем оформления.

- страницы аутентификации (LoginPage.tsx, RegisterPage.tsx): Предоставляют стандартные формы для ввода email и пароля. Валидация полей осуществляется на стороне клиента с использованием библиотек react-hook-form и yup перед отправкой данных на сервер.

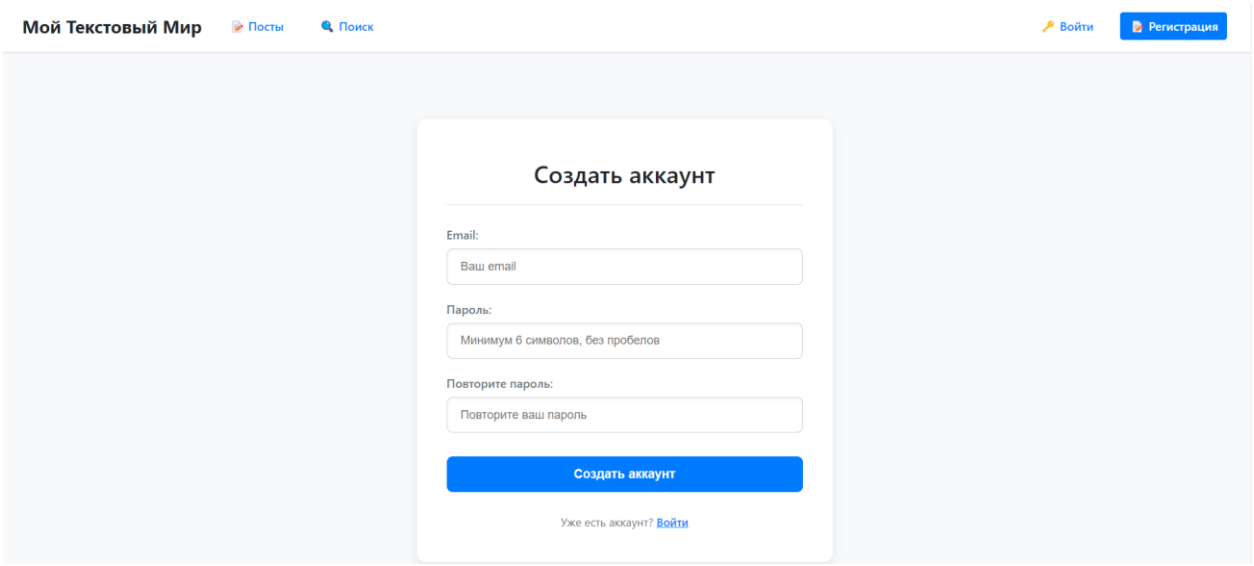


Рисунок 2.12 - Интерфейс страницы регистрации

- на Рисунке 2.12 представлен интерфейс страницы регистрации, демонстрирующий поля для ввода данных и элементы управления. Аналогичный интерфейс реализован для страницы входа.

- создание и Редактирование постов (NewPost.tsx, EditPost.tsx): Эти страницы содержат формы для ввода основного текста поста, строки тегов (которые на клиенте преобразуются в массив перед отправкой на бэкенд) и чекбокс для установки флага приватности. При редактировании существующего поста форма предварительно заполняется его текущими данными, которые загружаются с сервера.

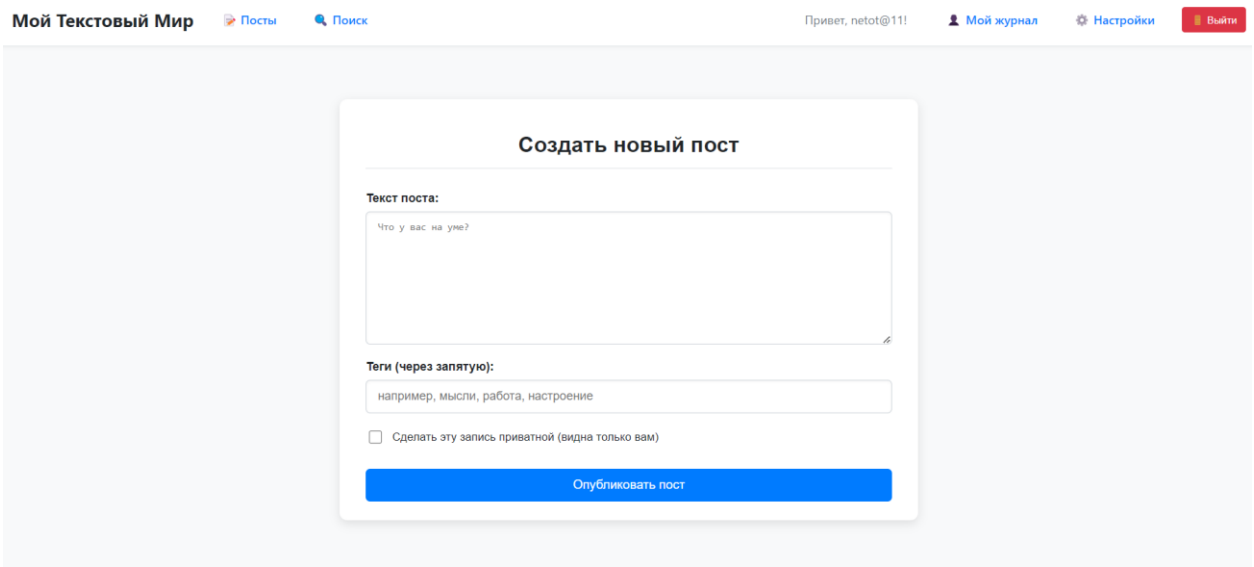


Рисунок 2.13 - Интерфейс страницы создания нового поста

- рисунок 2.13 иллюстрирует форму создания нового поста, включая поля для текста, тегов и выбора приватности. Страница редактирования имеет схожий

интерфейс с предзаполненными данными.

- отображение постов (в списках и детально):

- компонент Post.tsx отвечает за отображение отдельного поста в виде карточки в списках (например, на главной странице, в профиле, в результатах поиска). Карточка содержит превью текста, информацию об авторе, дате публикации, тегах, количестве лайков, а также интерактивные кнопки для лайка, добавления в избранное и (для автора поста) удаления.

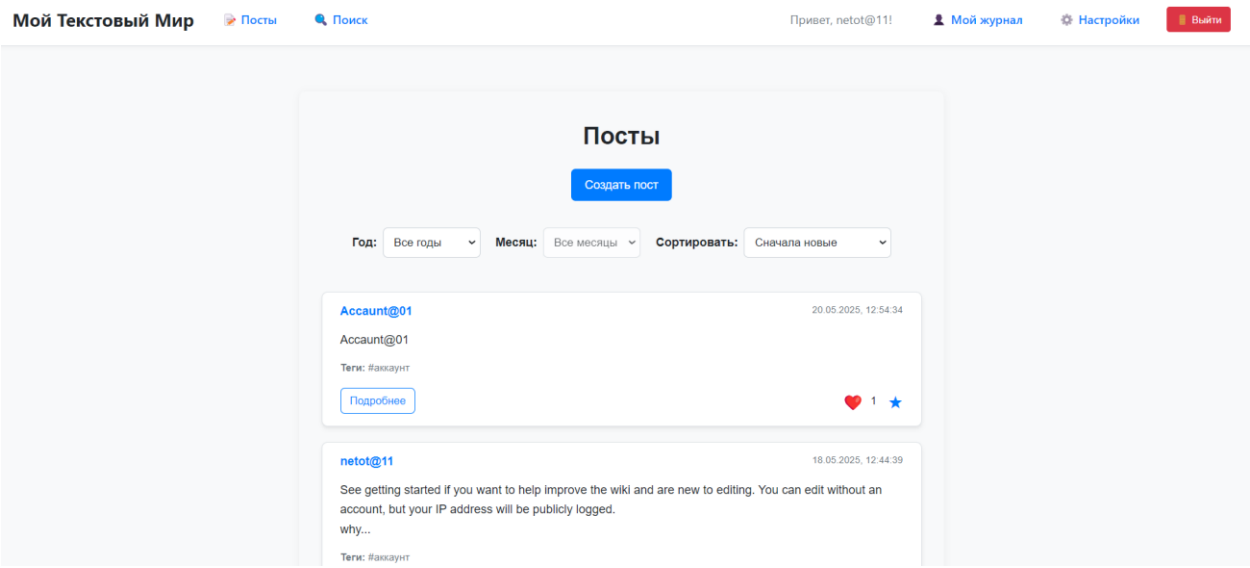


Рисунок 2.14 - Пример отображения карточки поста в списке

- на Рисунке 2.14 показан внешний вид карточки поста, используемой для отображения записей в различных списках, с элементами для взаимодействия.

- страница PostView.tsx отображает полную информацию о выбранном посте, включая весь текст, данные об авторе, теги, актуальное количество лайков и статус лайка/избранного для текущего пользователя. Здесь также присутствуют кнопки для лайка и добавления в избранное.

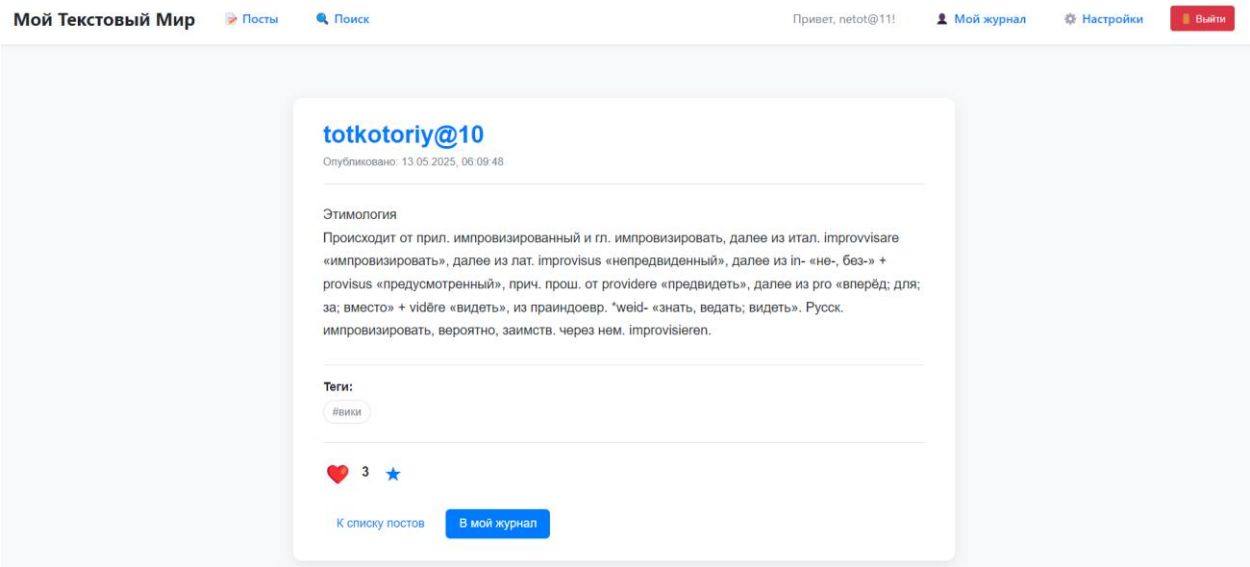


Рисунок 2.15 - Интерфейс детального просмотра поста

- рисунок 2.15 демонстрирует страницу детального просмотра поста с его полным содержимым и элементами управления.

- страница профиля пользователя (Profile.tsx): Является основным разделом для авторизованного пользователя и содержит несколько вкладок для удобной навигации по личному контенту.

- "мои записи": По умолчанию отображает все посты, созданные текущим пользователем (как публичные, так и приватные). Для этого списка реализована возможность фильтрации по дате (год, месяц) и сортировки (по дате, по популярности).

- "понравившиеся": загружает и отображает список постов, которые пользователь ранее лайкнул.

- "избранное": загружает и отображает список постов, которые пользователь добавил в закладки. Состояние активной вкладки, а также параметры фильтрации и сортировки для "Моих записей", синхронизируются с URL-параметрами с помощью хука useSearchParams. На странице также присутствуют кнопки для экспорта всех своих записей в формате JSON, перехода к настройкам аккаунта и выхода из системы. (ПОМЕТКА: Вставить Рисунок 2.16 - Интерфейс страницы профиля пользователя (вкладка "Мои записи")) На Рисунке 2.16 представлен интерфейс страницы профиля с активной вкладкой "Мои записи" и доступными элементами управления и фильтрации.

- страница настроек (SettingsPage.tsx): предоставляет пользователю возможность изменить свой текущий пароль (с проверкой старого пароля и подтверждением нового) и переключить тему оформления интерфейса между светлой и темной. Выбор темы сохраняется в localStorage и глобально применяется ко всему приложению через ThemeContext и CSS-переменные.

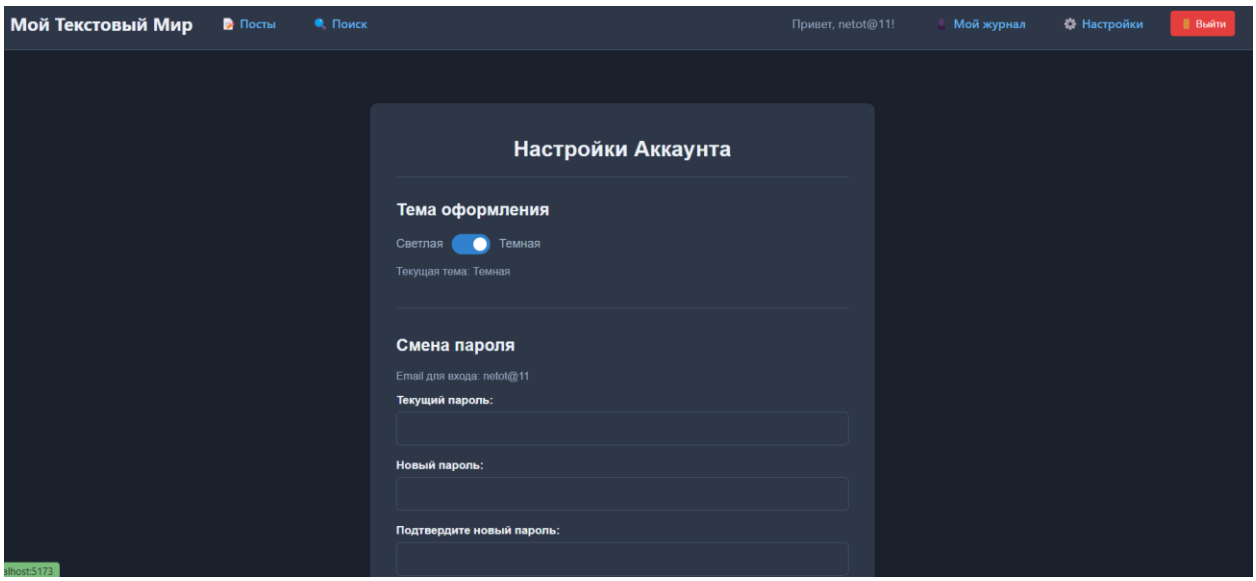


Рисунок 2.17 - Интерфейс страницы настроек аккаунта

- рисунок 2.17 показывает форму смены пароля и переключатель темы оформления на странице настроек.

- страница поиска (SearchPage.tsx): содержит форму для ввода тега. После выполнения поиска отображается список найденных постов, к которому также можно применить фильтры по дате и сортировку. Поисковый запрос и параметры фильтрации/сортировки отражаются в URL.

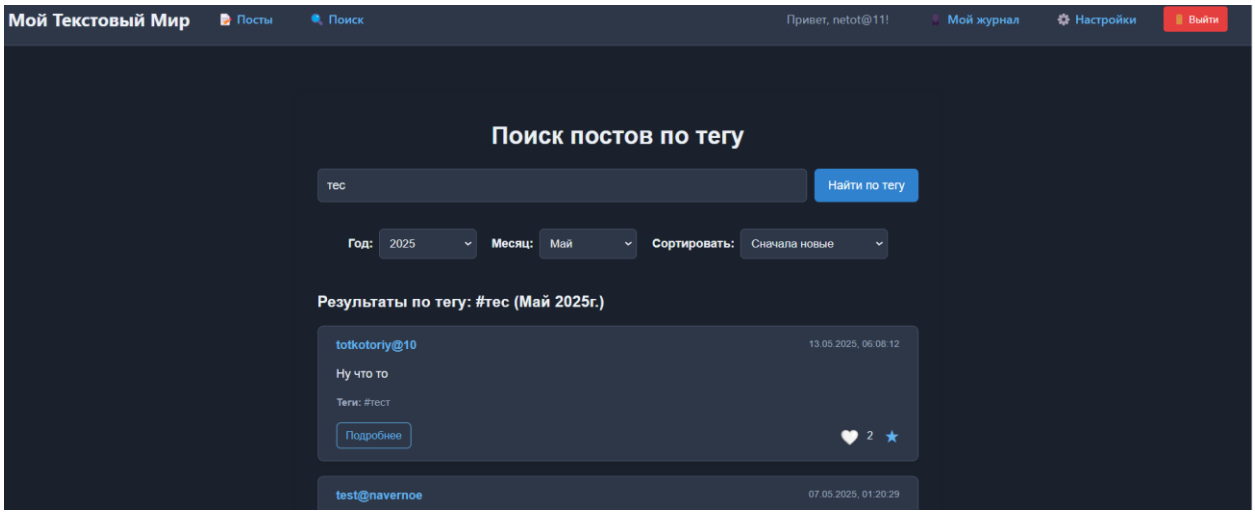


Рисунок 2.18 - Интерфейс страницы результатов поиска по тегу

- на Рисунке 2.18 представлен интерфейс страницы поиска с полем для ввода тега и областью отображения результатов.

3 Прикладной раздел (Практическая реализация, тестирование и результаты)

3.1 Результаты тестирования

Для проверки работоспособности и соответствия разработанной платформы заявленным требованиям было проведено комплексное ручное тестирование. Тестирование охватывало все основные пользовательские сценарии и функциональные модули системы. В ходе тестирования проверялись:

- корректность процессов регистрации новых пользователей и авторизации существующих.
- функционал создания, редактирования, просмотра (публичных и приватных) и удаления постов.
- правильность работы системы тегирования, включая добавление тегов к постам и поиск по ним.
- эффективность поиска постов по ключевым словам.
- корректность применения фильтров по дате (год, месяц) и различных видов сортировки списков постов.
- работоспособность механизма лайков (постановка/снятие, отображение счетчика и статуса).
- функционал добавления постов в "Избранное" и их удаления оттуда, а также отображение статуса.
- просмотр списков "Мои записи", "Понравившиеся" и "Избранное" на странице профиля.
- процедуры смены пароля и экспорта пользовательских данных.
- корректность переключения и сохранения темы оформления.
- обработка некорректного ввода данных и отображение сообщений об ошибках.
- проверка прав доступа к приватным данным и защищенным функциям (например, невозможность редактировать или удалять чужие посты, доступ к приватным записям только для автора).

В процессе тестирования особое внимание уделялось проверке прав доступа, корректности обработки ошибок и валидации вводимых данных. Были выявлены и устранены незначительные недочеты в пользовательском интерфейсе и логике обработки некоторых граничных условий. По результатам тестирования можно утверждать, что основные функции платформы работают корректно и в соответствии с техническим заданием. Отказов в работе критически важных функций зафиксировано не было.

3.2 Оценка достижения поставленных задач

Все основные задачи, сформулированные во введении данного дипломного проекта, были успешно решены:

1. Проведен детальный анализ предметной области и существующих аналогов, что позволило выявить свободную нишу и определить ключевые конкурентные преимущества разрабатываемой платформы. Результаты анализа представлены в Разделе 1.

2. Разработано подробное техническое задание, которое послужило основой для проектирования и реализации системы. Основные требования ТЗ отражены в описании функциональности в Разделе 2.

3. Обоснован и успешно применен выбранный стек технологий (Go, Gin, GORM, PostgreSQL для бэкенда; React, Vite, TypeScript для фронтенда; JWT для аутентификации), который обеспечил необходимую производительность, безопасность и удобство разработки. Обоснование представлено в Разделе 1.

4. Спроектирована и задокументирована архитектура системы, включая структуру базы данных, API и основные компоненты пользовательского интерфейса, с использованием соответствующих диаграмм (вариантов использования, последовательности, активности), представленных в Разделе 2.

5. Реализована вся запланированная ключевая функциональность бэкенда и фронтенда, включая регистрацию, авторизацию, управление постами, тегами, лайками, избранным, поиск, фильтрацию, экспорт данных и смену темы. Детали реализации описаны в Разделе 2.

6. Разработан минималистичный и интуитивно понятный пользовательский интерфейс, обеспечивающий комфортное взаимодействие с платформой, что подтверждается описанием ключевых экранов в Разделе 2.

7. Проведено тестирование основных пользовательских сценариев, подтвердившее работоспособность и соответствие системы заявленным требованиям, как описано в предыдущем подразделе (3.1).

Таким образом, цель проекта - создание удобной и минималистичной текстовой платформы для публикации мыслей и идей с контролем приватности и без рекламы - полностью достигнута. Разработанное программное обеспечение представляет собой готовый к использованию продукт, отвечающий всем основным заявленным требованиям.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данного дипломного проекта была успешно разработана минималистичная текстовая платформа, предназначенная для обмена мыслями и эмоциями. Проект был нацелен на решение актуальных проблем современных социальных сетей, таких как информационная перегруженность, недостаточный контроль над приватностью и навязчивая коммерциализация.

Основные достигнутые результаты включают:

– создание полнофункционального бэкенда на Go с использованием Gin и GORM, обеспечивающего надежную работу с данными в PostgreSQL.

– реализацию безопасной системы аутентификации пользователей на основе JWT.

– разработку современного и отзывчивого фронтенда на React (с TypeScript и Vite), предоставляющего интуитивно понятный пользовательский интерфейс.

– внедрение полного цикла управления постами (создание, редактирование, удаление, просмотр публичных и приватных записей).

– реализацию системы тегирования, поиска по тегам и ключевым словам, а также фильтрации и сортировки контента.

– добавление интерактивных элементов, таких как система лайков и возможность добавления постов в "Избранное".

– обеспечение функций управления аккаунтом, включая смену пароля, экспорт пользовательских данных и персонализацию интерфейса через смену тем оформления.

Разработанная платформа соответствует всем основным требованиям, изложенным в техническом задании, и успешно прошла этап ручного тестирования ключевых пользовательских сценариев. Проект демонстрирует возможность создания востребованного и конкурентоспособного продукта в нише текстовых платформ, ориентированных на простоту, удобство и уважение к личному пространству пользователя.

Перспективы дальнейшего развития проекта могут включать внедрение системы комментариев, разработку системы уведомлений, расширение настроек профиля, а также дальнейшую оптимизацию производительности и возможное внедрение автоматизированных тестов для повышения надежности при последующих доработках.

В целом, дипломный проект успешно завершен, поставленные цели достигнуты, а разработанное программное обеспечение готово к практическому применению и дальнейшему развитию.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Гамма, Э., Хелм, Р., Джонсон, Р., Влиссидес, Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. - СПб.: Питер, 2020. - 368 с.
2. Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures (Doctoral dissertation, University of California, Irvine). Chapter 5: Representational State Transfer (REST). [Электронный ресурс]. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (дата обращения: 16.02.2025).
3. Донован, А. А. П., Керниган, Б. В. Язык программирования Go. - М.: Вильямс, 2016. - 432 с.
4. Official Go Documentation. [Электронный ресурс]. URL: https://golang.org/doc/ (дата обращения: 18.05.2025).
5. Gin Web Framework: The official Gin Gonic documentation. [Электронный ресурс]. URL: https://gin-gonic.com/docs/ (дата обращения: 19.02.2025).
6. GORM - The fantastic ORM library for Golang. [Электронный ресурс]. URL: https://gorm.io/docs/ (дата обращения: 19.02.2025).
7. PostgreSQL: Documentation: Manuals. [Электронный ресурс]. URL: https://www.postgresql.org/docs/current/index.html (дата обращения: 01.03.2025).
8. Bcrypt package - golang.org/x/crypto/bcrypt. [Электронный ресурс]. URL: https://pkg.go.dev/golang.org/x/crypto/bcrypt (дата обращения: 01.03.2025).
9. React - A JavaScript library for building user interfaces: Official Documentation. [Электронный ресурс]. URL: https://reactjs.org/docs/getting-started.html (дата обращения: 03.03.2025).
10. TypeScript Handbook - Basic Types. [Электронный ресурс]. URL: https://www.typescriptlang.org/docs/handbook/basic-types.html (дата обращения: 20.03.2025).
11. Vite | Next Generation Frontend Tooling: Guide. [Электронный ресурс]. URL: https://vitejs.dev/guide/ (дата обращения: 20.03.2025).
12. Axios GitHub Repository. [Электронный ресурс]. URL: https://github.com/axios/axios (дата обращения: 22.03.2025).
13. React Router: Web Documentation. [Электронный ресурс]. URL: https://reactrouter.com/en/main (дата обращения: 28.03.2025).
14. React Hook Form - Performant, flexible and extensible forms with easy-to-use validation. [Электронный ресурс]. URL: https://react-hook-form.com/ (дата обращения: 28.03.2025).
15. Yup - JavaScript schema builder for value parsing and validation. [Электронный ресурс]. URL: https://github.com/jquense/yup (дата обращения: 11.04.2025).
16. JSON Web Tokens - Introduction. [Электронный ресурс]. URL: https://jwt.io/introduction (дата обращения: 14.04.2025).

17. JWT package for Go - github.com/golang-jwt/jwt/v5. [Электронный ресурс]. URL: https://pkg.go.dev/github.com/golang-jwt/jwt/v5 (дата обращения: 14.04.2025).

18. Фримен, Э., Фримен, Э., Сьерра, К., Бейтс, Б. Паттерны проектирования. - СПб.: Питер, 2021. - 656 с.

19. Мартин, Р. Чистый код: создание, анализ и рефакторинг. - СПб.: Питер, 2021. - 464 с.

Приложение А

Техническое задание

Настоящее техническое задание (ТЗ) распространяется на разработку программной системы - веб-платформы, предназначенной для публикации и обмена текстовым контентом с акцентом на минимализм, удобство использования и приватность данных пользователей. Предполагается, что данную систему будут использовать пользователи, заинтересованные в простом и сфокусированном пространстве для самовыражения, ведения личных записей и чтения тематического контента без избыточных функций и рекламы.

А1.5.1 Основание для разработки

Программная система разрабатывается в рамках выполнения дипломного проекта на тему "Разработка минималистичной текстовой платформы для обмена мыслями и эмоциями" на основании задания, утвержденного научным руководителем. Необходимость разработки обусловлена выявленными недостатками существующих социальных платформ и потребностью пользователей в альтернативных решениях.

А.1.5.2 Назначение

Система предназначена для предоставления пользователям следующих возможностей:
- публикация текстовых записей (постов) с возможностью управления их видимостью (публичные/приватные).
- организация контента с помощью тегов.
- поиск и фильтрация постов по тегам, ключевым словам и дате.
- взаимодействие с контентом через систему лайков и добавления в избранное.
- ведение личного журнала (просмотр своих приватных и публичных записей).
- управление своим аккаунтом, включая смену пароля и экспорт собственных данных.
- просмотр публичного контента без обязательной регистрации.
- использование платформы без рекламы и мультимедийного контента.

А.1.5.3 Требования к функциональным характеристикам

Программная система должна обеспечивать выполнение следующих функций:
- регистрация и Аутентификация Пользователей:
- возможность создания нового аккаунта с использованием уникального email и пароля.
- безопасное хранение паролей (хеширование).
- авторизация пользователей по email и паролю.
- управление сессиями с использованием JWT (JSON Web Tokens).
- возможность выхода из системы.
- управление Постами:
- создание нового текстового поста авторизованным пользователем.

- возможность указания текста поста, списка тегов и статуса приватности (публичный/приватный) при создании.

- редактирование собственных постов (текст, теги, приватность) авторизованным пользователем.

- удаление (мягкое) собственных постов авторизованным пользователем.

- просмотр и Навигация по Постам:

- отображение списка публичных постов для всех пользователей.

- отображение списка всех постов (публичных и приватных) для авторизованного пользователя в разделе "Мой журнал".

- просмотр полного текста отдельного поста с информацией об авторе и дате публикации.

- проверка прав доступа при просмотре приватных постов (только автор).

- поиск и Фильтрация:

- поиск постов по одному или нескольким тегам.

- поиск постов по ключевым словам в тексте.

- фильтрация списков постов по дате создания (год, месяц).

- сортировка списков постов по дате публикации (новые/старые) и по количеству лайков (популярные).

- взаимодействие с Постами:

- возможность для авторизованного пользователя поставить/убрать лайк посту.

- отображение общего количества лайков для каждого поста.

- возможность для авторизованного пользователя добавить/удалить пост из списка "Избранное".

- отображение статуса "лайкнут ли пост текущим пользователем" и "добавлен ли пост в избранное текущим пользователем".

- управление Аккаунтом:

- возможность смены пароля для авторизованного пользователя (с подтверждением старого пароля).

- возможность экспорта всех своих постов в формате JSON.

- возможность смены темы оформления интерфейса (светлая/темная) с сохранением выбора.

Приложение Б

Листинг (код) программы

```
package routes

import (
   "backend/controllers"

   "github.com/gin-gonic/gin"
)

func SetupRouter(r *gin.Engine) {
   api := r.Group("/api")
   {
      api.POST("/register", controllers.Register)
      api.POST("/login", controllers.Login)

      profileGroup := api.Group("/profile")
      profileGroup.Use(controllers.AuthMiddleware())
      {
         profileGroup.GET("", controllers.GetProfile)
         profileGroup.POST("/change-password", controllers.ChangePassword)
         profileGroup.GET("/export", controllers.ExportUserPosts)
         profileGroup.GET("/bookmarks", controllers.GetBookmarkedPosts)
         profileGroup.GET("/liked-posts", controllers.GetLikedPosts)
      }

      postsGroup := api.Group("/posts")
      {
         postsGroup.POST("", controllers.AuthMiddleware(), controllers.CreatePost)
         postsGroup.GET("", controllers.GetPosts)

         postByIdGroup := postsGroup.Group("/:id")
         postByIdGroup.Use(controllers.AuthMiddleware())
         {
            postByIdGroup.GET("", controllers.GetPostByID)
            postByIdGroup.PUT("", controllers.UpdatePost)
            postByIdGroup.DELETE("", controllers.DeletePost)
            postByIdGroup.POST("/like", controllers.ToggleLike)

            postByIdGroup.POST("/bookmark", controllers.ToggleBookmark)
            postByIdGroup.GET("/bookmark-status", controllers.GetBookmarkStatus)

         }

         postsGroup.GET("/:id/likes", controllers.GetPostLikes)


         postsGroup.GET("/my", controllers.AuthMiddleware(), controllers.GetUserPosts)
         postsGroup.GET("/tagged", controllers.GetPostsByTag)
```

```
        postsGroup.GET("/search", controllers.SearchPostsByKeyword)
    }
  }
}


package utils

import (
 "log"
 "os"
 "strings"
 "time"

 "github.com/golang-jwt/jwt/v5"
)

var secretKey = []byte(os.Getenv("JWT_SECRET_KEY"))

type Claims struct {
 Email  string `json:"email"`
 UserID uint   `json:"user_id"`
 jwt.RegisteredClaims
}

func GenerateToken(email string, userID uint) (string, error) {
 claims := Claims{
  Email:  email,
  UserID: userID,
  RegisteredClaims: jwt.RegisteredClaims{
   ExpiresAt: jwt.NewNumericDate(time.Now().Add(72 * time.Hour)),
  },
 }

 token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
 return token.SignedString(secretKey)
}

// Проверка JWT
func ValidateToken(tokenString string) (*Claims, error) {
 log.Println("Полученный токен:", tokenString)

 tokenString = strings.TrimPrefix(tokenString, "Bearer ")

 token, err := jwt.ParseWithClaims(tokenString, &Claims{}, func(token *jwt.Token) (interface{},
error) {

  return secretKey, nil
 })
```

```
 if err != nil {
  log.Println("Ошибка при парсинге токена:", err)
  return nil, err
 }

 claims, ok := token.Claims.(*Claims)
 if !ok || !token.Valid {
  log.Println("Токен не валиден")
  return nil, err
 }

 log.Printf("Извлеченные данные из токена - Email: %s, UserID: %d", claims.Email,
claims.UserID)
 return claims, nil
}

package main

import (
 "backend/config"
 "backend/middleware"
 "backend/models"
 "backend/routes"
 "log"

 "github.com/gin-gonic/gin"
)

func main() {
 config.ConnectDatabase()
 gin.SetMode(gin.ReleaseMode)

 config.InitDB()

 config.DB.AutoMigrate(&models.User{}, &models.Post{}, &models.Tag{},
&models.Comment{}, &models.Like{})

 r := gin.Default()

 r.Use(middleware.CORSMiddleware())

 routes.SetupRouter(r)

 for _, route := range r.Routes() {
  log.Printf("Маршрут: %s %s", route.Method, route.Path)
 }


 log.Println("Сервер запущен на порту 8080")
```

Продолжение приложения Б

```
 if err := r.Run(":8080"); err != nil {
  log.Fatal("Ошибка запуска сервера:", err)
 }
}
```

```
      <div>Loading...</div>
    ) : (
      <div>
        <div
          className="container"
          style={{ maxWidth: "1440px", padding: "0 15px", margin: "0 auto" }}
        >
          <label>Select Time Range: </label>
          <select value={selectedTimeRange} onChange={handleTimeRangeChange}>
            {timeRanges.map((time) => (
              <option value={time.measurementDateTime}>
                {time.formattedDateTime}
              </option>
            ))}
          </select>

          <label>Select City: </label>
          <select value={selectedCity} onChange={handleCityChange}>
            {cities.map((c) => (
              <option value={c.cityId}>{c.cityNameEn}</option>
            ))}
          </select>

          <label>Select Chemical Element: </label>
          <select
            value={selectedChemicalElement}
            onChange={handleChemicalElementChange}
          >
            {chemicalElements.map((ch) => (
              <option value={ch.id}>{ch.nameEn}</option>
            ))}
          </select>
        </div>
        <div id="map" style={{ width: "100%", height: "100vh" }}></div>
      </div>
    )}
  </div>
 );
};

export default MapWrapper;

import { Routes, Route, Navigate } from "react-router-dom";
import Layout from "./components/Layout";
```

Продолжение приложения Б

```
import Home from "./pages/Home";
import Posts from "./pages/Posts";
import NewPost from "./pages/NewPost";
import EditPost from "./pages/EditPost"
import SearchPage from "./pages/SearchPage";
import PostView from "./pages/PostView";
import Profile from "./pages/Profile";
import Login from "./pages/Login";
import Register from "./pages/Register";
import SettingsPage from "./pages/SettingsPage";
import { AuthProvider, useAuth, AuthContextType } from "./context/AuthContext";
import { ThemeProvider } from "./context/ThemeContext";
import React from "react";

const PrivateRoute = ({ element }: { element: React.ReactElement }) => {
  const authContext = useAuth() as AuthContextType;

  if (authContext.loading) {
    return <p style={{ textAlign: 'center', marginTop: '50px', fontSize: '1.2em' }}>Проверка
авторизации...</p>;
  }

  return authContext.isAuthenticated ? element : <Navigate to="/login" replace />;
};

function App() {
  return (
    <ThemeProvider>
      <AuthProvider>
        <Routes>
          <Route path="/" element={<Layout />}>
            <Route index element={<Home />} />
            <Route path="posts" element={<Posts />} />
            <Route path="posts/new" element={<PrivateRoute element={<NewPost />} />} />
            <Route path="posts/edit/:id" element={<PrivateRoute element={<EditPost />} />} />
            <Route path="posts/search" element={<SearchPage />} />
            <Route path="posts/:id" element={<PostView />} />
            <Route path="profile" element={<PrivateRoute element={<Profile />} />} />
            <Route path="settings" element={<PrivateRoute element={<SettingsPage />} />} />
            <Route path="login" element={<Login />} />
            <Route path="register" element={<Register />} />
          </Route>
        </Routes>
      </AuthProvider>
    </ThemeProvider>
  );
}

export default App;
```