

ТИТУЛЬНИК

РЕФЕРАТ

ВЕБ-ПРИЛОЖЕНИЕ ДЛЯ УПРАВЛЕНИЯ И ХРАНЕНИЯ ЗАМЕТОК

Дипломный проект предоставлен следующим образом: 3 чертежа форматом А1, 3 плаката форматом А1, пояснительная записка, включающая: 102 страниц, 49 рисунков, 16 таблицы, 13 литературных источников, 1 приложение.

Цель работы – проектирование и разработка веб-приложения для управления и хранения пользовательских заметок

Разработка и использование описанного программного средства позволяет создавать заметки различного типа, для хранения онлайн с возможностью получить доступ при наличии интернета.

Результаты проектирования: функциональное назначение программного средства, описаны аналоги программного средства, разработаны алгоритмы работы, спроектирована логическая и физическая модель базы данных, описана структура программного средства с отображением диаграмм классов, уделено внимание вопросам технико-экономического обоснования, проведено тестирование работоспособности разработанного программного средства, разработана методика использования программного средства.

В разделе технико-экономического обоснования были произведены расчеты затрат, связанных с реализацией проекта, а также рентабельности разработки проекта. Проведенные расчеты показали экономическую целесообразность проекта.

СОДЕРЖАНИЕ

Введение.....	7
1 Анализ литературных источников и формирование требований к проектируемому программному средству.....	8
1.1 Описание предметной области	8
1.2 Анализ существующих аналогов.....	9
1.3 Постановка цели и задач	15
1.4 Входные данные	15
1.5 Выходные данные	16
2 Моделирование программного средства	17
2.1 Функциональные требования.....	17
2.2 Описание функциональности программного средства	17
2.3 Разработка информационной модели	19
2.4 Разработка модели взаимодействия пользователя с интерфейсом.....	21
2.5 Разработка технических требований к программному средству	22
3 Проектирование программного средства	23
3.1 Обоснование выбора среды разработки	23
3.2 Разработка структурной схемы программного средства	25
3.3 Разработка структуры классов.....	29
3.4 Разработка физической модели данных	31
3.5 Проектирование алгоритмов работы программного средства	34
4 Тестирование программного средства.....	38
5 Методика использования программного средства	45
5.1 Регистрация пользователя	45
5.2 Авторизация пользователя	46
5.3 Создание и редактирование заметки.....	48
5.4 Поиск заметок.....	55
5.5 Поделиться заметкой	56
5.6 Администратор	57
6 Техничко-экономическое обоснование разработки и использования программного средства.....	58
6.1 Описание функций, назначения и потенциальных пользователей программного средства.....	58
6.2 Расчет затрат на разработку программного средства.....	58
6.3 Оценка экономического эффекта при разработке программного средства для свободной реализации на ИТ–рынке.....	61
Заключение	63
Список использованных источников	64
Приложение А. Текст программы	65

ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке применяются следующие определения и сокращения.

Авторизация – предоставление определённому лицу или группе лиц прав на выполнение определённых действий; а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий.

Аутентификация – проверка подлинности предъявленного пользователем идентификатора.

БД – база данных.

ООП – объектно-ориентированное программирование.

ОС – операционная система.

Программа – данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определенного алгоритма.

Программное средство – совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ.

Программный модуль – программа или функционально завершённый фрагмент программы, предназначенный для хранения, трансляции, объединения с другими программными модулями и загрузки в оперативную память.

ПС – программное средство.

Фреймворк – программное обеспечение, облегчающее разработку и объединение различных компонентов большого программного проекта.

API – application programming interface (Интерфейс программирования приложений).

DOM – Document Object Model (Объектная модель документа).

HTTPS – HyperText Transfer Protocol Secure (Безопасный протокол передачи гипертекста).

JSON – Javascript Object Notation.

JWT – JSON Web Token (это JSON объект, который определен в открытом стандарте RFC 7519).

LINQ – Language-Integrated Query (Интегрированный язык запросов).

SPA – Single Page Application (Одностраничное приложение).

ВВЕДЕНИЕ

На сегодняшний день, мы все чаще записываем и сохраняем информацию, для дальнейшего изучения и использования. Это обусловлено большим количеством информации на просторах интернета и нашей жизни. И для большинства людей это является проблемой, так как требуется запоминать и записывать большое количество разного рода информацию на разных носителях.

Для того чтобы не потерять, не забыть самому и поделиться с другими информацией используются заметки.

Ведение заметок является неотъемлемой частью жизни современных людей. Однако не все могут позволить носить с собой записную книжку или не всегда она оказывается под рукой. Поиск нужных заметок в большом количестве данных занимает много времени, а также не всегда ты сможешь отыскать нужную тебе информацию. У записной книжки есть очень важный минус, что количество записей ограничено, также бумажную записную книжку можно забыть и потерять.

В связи с развитием интернета и создания большого количества веб-приложений, каждый пользователь может воспользоваться ими из любой части мира.

Одним из вариантов решения проблемы с ношением записных книжек и ограничением места, является использования специализированных веб-приложений, позволяющие оформлять и сохранять собственные заметки онлайн.

Существуют множество веб-приложений для ведения заметок, отличающиеся между собой: подходом формирования заметки, дизайном, удобством использования, а также ценой. Все они в каком-то виде пытаются заменить бумажные носители и предоставить удобства использования.

Целью данного дипломного проекта является сбор и анализ информации существующих аналогов в этой области и созданием собственного веб-приложения для управления и хранения пользовательских заметок. Данное программное обеспечение позволит сохранять свои заметки и легко искать информацию по заметкам, а также делиться этой информацией с другими людьми.

1 АНАЛИЗ ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ

1.1 Описание предметной области

В ходе увеличения объема информации и количества дел, появилась необходимость записывать всю эту информацию, для дальнейшего хранения упорядочивания и использования.

Для структурирования и хранения информации без использования интернета используются: меловые и маркерные доски, записные книжки.

Но с появлением интернета и его развития начали создавать различные программы для структурирования и записи информации.

Программы можно разделить на следующие виды по типу предоставляемого функционала:

- электронная доска;
- написание заметок;
- интеллект-карта.

К виду электронных досок можно отнести электронная интерактивная доска.

Интерактивная доска — это большой интерактивный экран в виде белой магнитно-маркерной доски. Интерактивная доска может быть представлена как автономным компьютером с большим сенсорным экраном, так и подключаемым к ноутбуку устройством, объединяющим проектор и сенсорную панель. Интерактивные доски используются в школьных кабинетах, переговорных, залах для групповых занятий, комнатах для дистанционного обучения и других помещениях [1].

К виду написание заметок можно отнести программные продукты, которые имитируют записную книгу и позволяют писать заметки и управлять ими.

К виду интеллект-карта можно отнести программные продукты, которые структурируют информацию в виде карты мыслей и методу интеллект-карт.

Интеллект-карта — это способ фиксации мыслей, наиболее похожий на то, как они рождаются и развиваются в нашей голове. Их также могут называть диаграммами связей, ментальными или ассоциативными картами и картами мыслей [2].

Программы для ведения заметок можно разделить на классы на основе предоставления сервисов в Интернет:

- offline-приложения – функционала – представляют собой приложения с частичным функционалом работающий через интернет;

– offline-приложения с элементами online-функционала – представляют собой приложения с частичным функционалом работающий через интернет;

– online-приложения – это приложения, функционал которого работает только с использованием интернета и представляют собой веб-приложения.

Веб-приложения – это интерактивные компьютерные приложения, разработанные для сети интернет, позволяющие пользователям вводить, получать и манипулировать данными с помощью взаимодействия. Такие программы обычно имеют очень тесную связь с сервером, отправляя на него множество запросов [3].

Также программы можно разделить на виды по их стоимости:

- бесплатные программы;
- бесплатные программы с дополнительными платными функциями;
- платные программы.

1.2 Анализ существующих аналогов

Сейчас создано много различных программных средств для создания пользовательских заметок, включающие простые и сложные, а также бесплатных и платных. Но каждая программа содержит различные функции и распространяется по-разному.

1.2.1 Google Keep

Одним из известных сервисов для создания быстрых заметок, является сервис от компании Google. Google Keep — онлайн-сервис для создания заметок и их хранения [4].

Сервис обладает следующими возможностями:

- создавать заметки в виде карточек;
- отправить заметку в архив;
- выбирать цвет карточки;
- поиск по заметкам;
- добавлять теги.

На рисунке 1.1 представлен интерфейс приложения Google Keep.

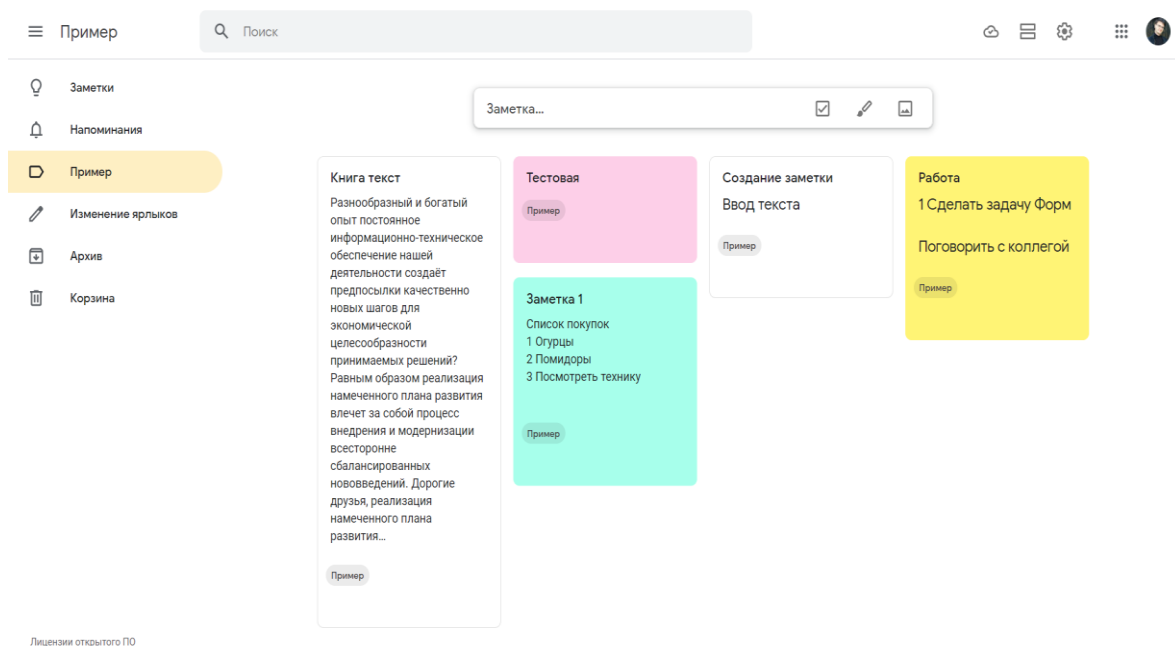


Рисунок 1.1 – Интерфейс программы Google Keep

Для создания заметки надо нажать поле заметка и тогда откроется небольшая форма для создания. Форма создания заметки представлена на рисунке 1.2.

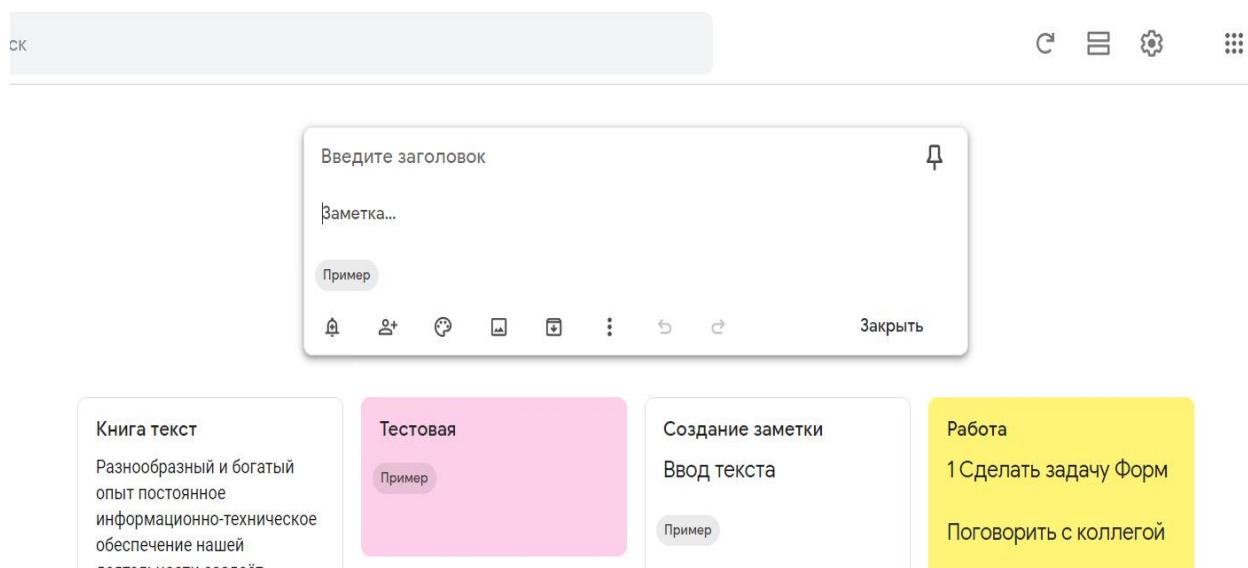


Рисунок 1.2 –Интерфейс формы по созданию заметки

При создании есть следующие возможности:

- изменить цвет карточки;
- добавить соавтора (только имеющего учетную запись Google);
- добавить тег;
- создание разного типа заметок.

Google keep поддерживает следующие виды заметок:

- текстовые
- в виде списка
- картинка (ссылка на картинку из интернета).

Заметки в виде списка позволяют отмечать выполнение данных задач. На рисунке 1.3 представлен пример создания заметки в виде списка с изменением цвета карточки.

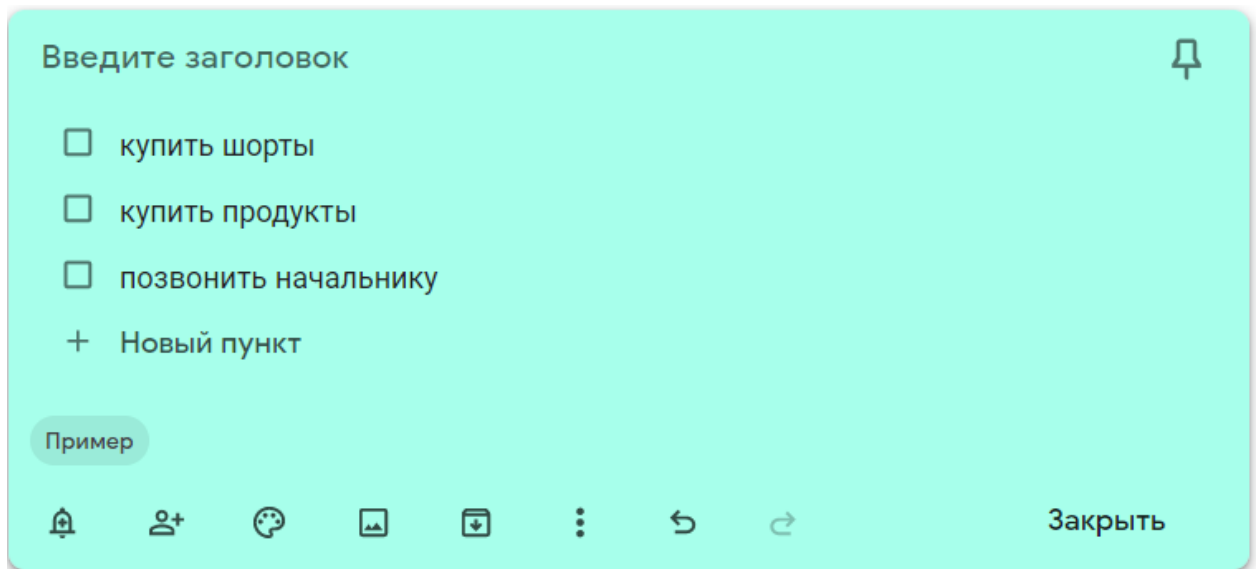


Рисунок 1.3 – Заметка в виде списка

Так как это сервис принадлежит к общей экосистеме Google, он предоставляется на бесплатной основе только пользователям, у которых есть учетная запись Google. Это накладывает ограничения на пользователя, так как он должен создать себе учетную запись и контролировать ее.

Проанализировав сервис можно выделить следующие преимущества:

- простой дизайн;
- создавать быстрые заметки, которые нужны на короткий срок;
- создавать разные типы заметок;
- поиск по заметкам;
- использование тегов;
- поддерживает русский язык;
- является частью экосистемы Google, на бесплатной основе.

Сервис обладает следующими недостатками:

- хаотичное расположение карточек и когда их большое количество сложность в поиске информации;
- привязан к экосистеме Google;

— возможность использовать в одно время один тип заметок, что не позволяет группировать информацию в одной заметке.

1.2.2 Simplenote

Simplenote — это менеджер заметок. Simplenote позволяет работать только с текстами — поддержки иллюстраций, вложенных файлов и данных прочих форматов здесь нет. Интерфейс программы представлен на рисунке 1.4 [5].

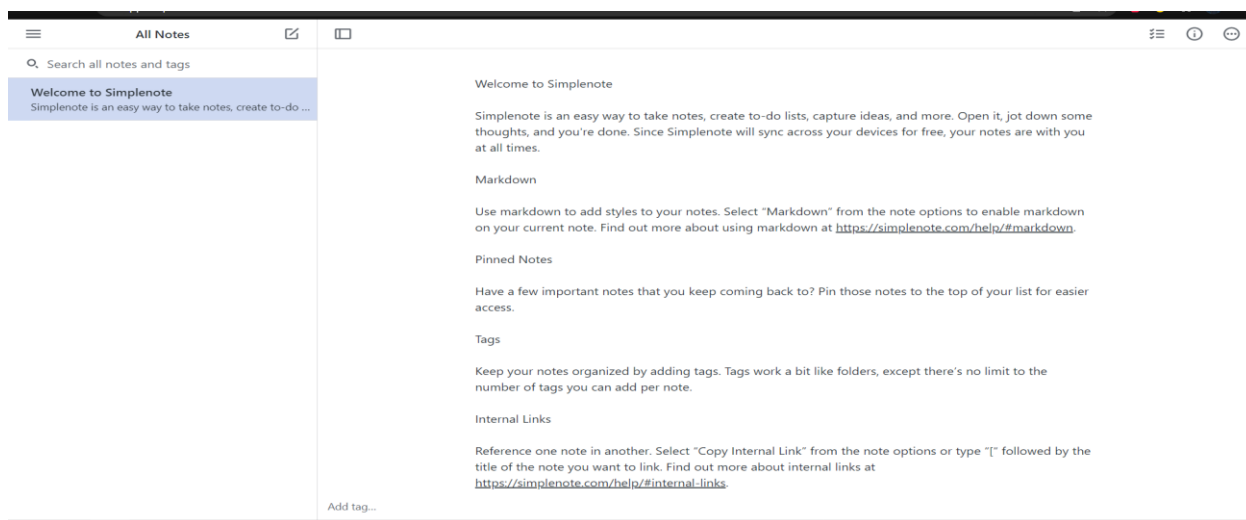


Рисунок 1.4 – Интерфейс программы

Программа предоставляет следующие возможности:

- создавать текстовые заметки, без возможности редактирования текста;
- искать заметки;
- добавлять теги.

Интерфейс позволяет добавлять заметки при нажатии на иконку редактирования и представлен на рисунке 1.5.

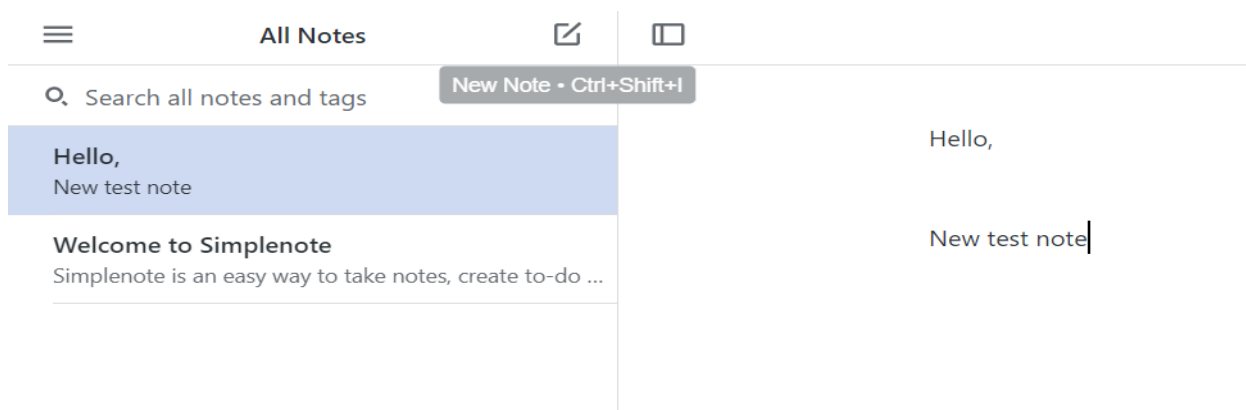


Рисунок 1.5 – Интерфейс создания заметки в программе SimpleNote

Для поиска используется поисковая строка, которая позволяет искать по названию, тегам и по содержанию заметки.

Это простое приложение имеет следующие преимущества:

- формировать текстовых заметок;
- добавление тегов;
- поиск по заметкам;
- бесплатное распространение.

Приложение обладает следующими недостатками:

– нет возможности оформлять другие типы заметок, только текстовые.

– интерфейс только на английском языке

По итогу можно сказать, что приложение позволяет, удобно не отвлекаясь записывать текстовые заметки, это удобно для студентов, которые ведут заметки лекций.

1.2.3 Notion

Notion — универсальная программа, которую в равной степени можно использовать и как приложение для хранения заметок, и в качестве платформы для создания баз знаний наподобие «Википедии», управления задачами и совместной работы с коллегами. Интерфейс программы Notion представлен на рисунке 1.6 [6].

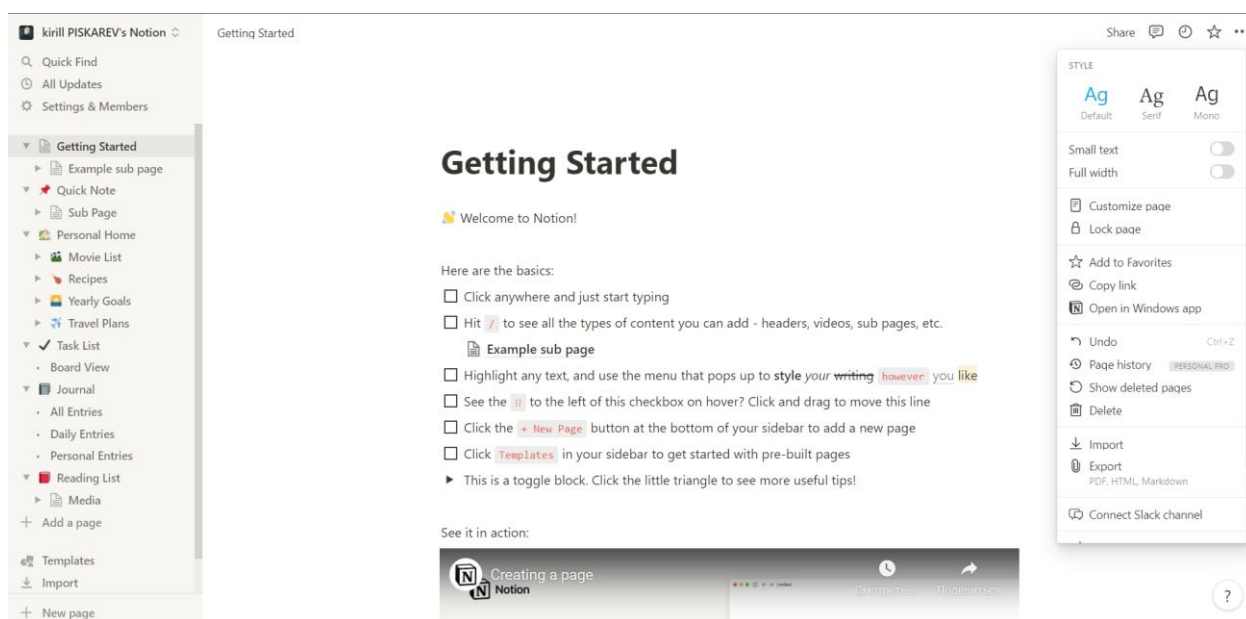


Рисунок 1.6 – Интерфейс программы Notion

В основной части функционал приложения предоставляет огромный выбор возможности подачи информации, начиная от форматируемых как угодно текстовых блоков, таблиц, чек-листов, канбан-досок или

многоуровневых списков и заканчивая дизайнерскими шаблонами всевозможных документов.

Интерфейс создание новой заметки представлен на рисунке 1.7.

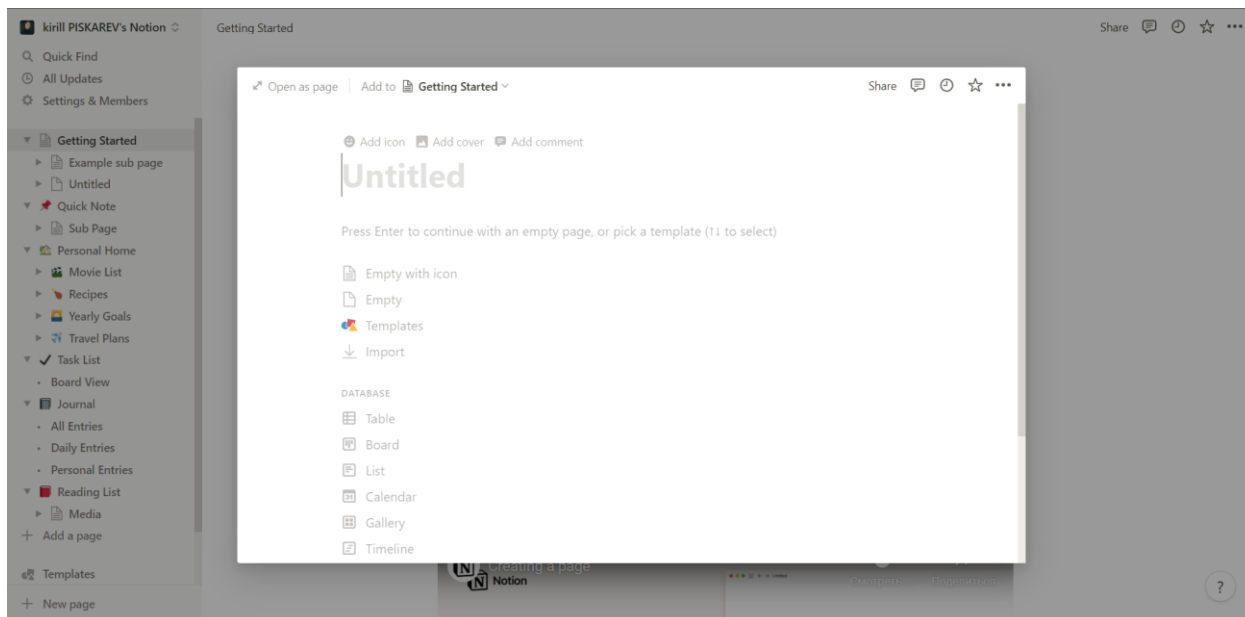


Рисунок 1.7 – Интерфейс создания заметки Notion

Урезанный функционал предоставляется бесплатно, и он больше предназначен для ознакомления, для полного использования, требуется оформлять подписку. Подписка составляет 4\$ в месяц для личного использования и 8\$ в месяц для командной работы.

Из-за большого набора функционала программа является сложной для освоения пользователя, без изучения предварительно документации по всем компонентам. Также большое количество функционала отвлекает от основной задачи записи заметок.

Программа имеет следующие преимущества:

- большое количество функций по созданию разных типов заметок;
- возможность добавлять теги;
- возможность работать с командой;
- создание разных видов досок.

Программа имеет следующие недостатки:

- не удобно использовать, из-за большого количества функций, требующее изучения документации перед использованием;
- интерфейс представлен только на английском, корейском.

1.3 Постановка цели и задач

Как показал анализ программных решений в области создания и хранения заметок они обладают следующими недостатками:

- нет возможности форматировать текст;
- не позволяют группировать разные типы заметок;
- не все позволяют вставлять ссылки на картинки из интернета с загрузкой картинки;
- у бесплатных решений не хватает функционала для полноценной работы с заметками;
- высокая стоимость;
- нет возможности поделиться заметкой с любым пользователем.

Целью данного дипломного проекта является разработка веб-приложения для управления и хранения пользовательских заметок, способных устранить вышеперечисленные недостатки и обладающими всеми необходимыми функциями, характерных для программных средств в этой области.

Программа должна обеспечивать выполнение следующих функций:

- регистрация пользователя;
- авторизация пользователя;
- создавать, редактировать и удалять заметки;
- добавлять теги;
- редактирование текстовых заметок;
- добавление таблиц;
- оформлять в виде списка текст;
- добавлять ссылки на медиа.
- хранение заметок;
- поиск по заметкам;
- возможность управления пользователями администратору, чтобы сбросить пароль или удалить пользователя из системы со всеми данными;
- надежное шифрование паролей;
- возможность поделиться заметкой.

1.4 Входные данные

К входной информации мы будем относить все вводимые пользователем данные, конфигурационные данные по заметкам.

Пользователи при регистрации будут указывать свои персональные данные, для регистрации, а также при авторизации.

При создании заметки пользователи должны указать наименование заметки, сам текст заметки, теги, относящиеся к заметкам. При поиске формировать поисковые запросы.

В качестве кода к входным данным мы отнесем запросы на сервер для работы с приложением, конфигурации и форматирование заметок.

Входной информацией будут является:

- данные для регистрации и авторизации;
- пользовательская информация для заметки;
- теги;
- добавление, редактирование и удаление пользовательских заметок;
- поисковые запросы;
- тип информации для формирования заметки (ссылки, текст, таблицы, код);
- методы запросов: GET, POST, PUT, DELETE.

1.5 Выходные данные

В качестве выходной информации будет выступать данные после сохранения заметок такие как: сама заметка в отформатированном виде, ссылки для возможности поделиться информацией.

Система будет отдавать в качестве выходной информации списки тегов, используемые в системе. Если пользователь добавляет тег, которого не было в системе, он автоматически добавляется для этого пользователя в список тегов.

К выходной информации также относятся системные данные: сессии и токены авторизации, ошибки системы.

К выходной информации мы относим:

- пользовательская информация;
- сессия и JWT;
- информация по заметкам;
- списки тегов;
- ссылки для просмотра пользовательских заметок;
- ошибки.

2 МОДЕЛИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

2.1 Функциональные требования

Функциональным назначением разрабатываемого веб-приложения является предоставление возможностей управления и хранения пользовательских заметок.

В качестве пользователя программного средства может выступать любой человек, который имеет доступ к персональному компьютеру с доступом к сети интернет. Для использования программного решения не требуется специальная подготовка или обучение пользователей.

Предполагается возможность одновременной эксплуатации разрабатываемого решения широким кругом пользователей.

Исходя из предполагаемого использования, можно заключить, что проектируемое программное решение должно реализовывать следующие группы функций:

- регистрация и авторизация пользователя;
- управление и хранение пользовательских заметок;
- поиск по заметкам;
- управление доступом к заметкам;
- управление пользователями.

2.2 Описание функциональности программного средства

Диаграмма вариантов использования (use case diagram) – это наиболее общее представление функционального назначения системы.

Диаграмма вариантов использования призвана ответить на главный вопрос моделирования: «что делает система во внешнем мире?». На диаграмме применяются два типа основных сущностей: варианты использования и действующие лица, между которыми устанавливаются следующие основные типы отношений:

- ассоциация между действующим лицом и вариантом использования;
- обобщение между действующими лицами;
- обобщение между вариантами использования;
- зависимости (различных типов) между вариантами использования [7].

В системе можно выделить следующие действующие лица:

- анонимный пользователь;
- пользователь;
- администратор;

На рисунке 2.1 представлена общая диаграмма вариантов использования.

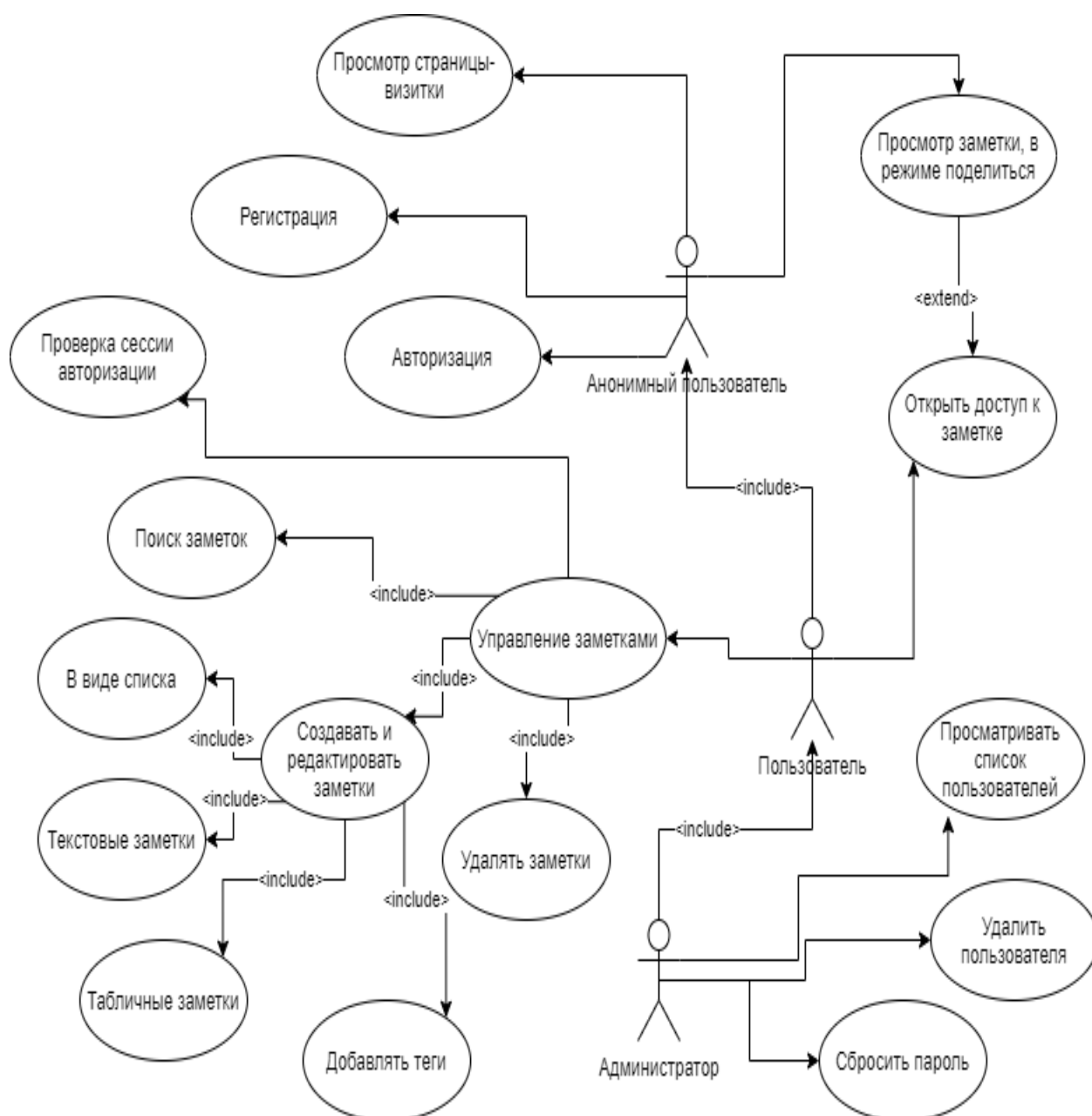


Рисунок 2.1 – Диаграмма вариантов использования

Анонимный пользователь – это пользователь, который не авторизован в системе. Данный пользователь может просмотреть главную страницу сайта, с описанием назначения веб-приложения. Анонимному пользователю доступны функции регистрации в приложении, а также авторизации.

Если Анонимному пользователю дали ссылку на доступ к режиму чтения заметки, ему доступна информация по заметке.

После авторизации анонимный пользователь получает роль «пользователь», после этого непосредственно перенаправляется в режим

управления заметками, но у него остаются все возможности и анонимного пользователя. В режиме управления заметками пользователю доступны следующие функции:

- создание заметки;
- редактирование заметки;
- удаление заметки;
- поиск заметок;
- открыть доступ к заметке;
- выход из системы.

Редактирование заметок включает в себя следующие возможности:

- работа с текстом (его форматирование и оформление);
- создание списка (числового, пунктами, переключателями);
- создание таблиц;
- добавление медийных ссылок (картинки и другие ссылки на источники из интернета).

В системе присутствует роль «Администратор», которая включает в себя возможности «Анонимного пользователя» и «Пользователя», а также расширяет возможности по управлению пользователями. У Администратора есть возможность просматривать пользователей, при необходимости сбрасывать пароль, а также удалять пользователя.

2.3 Разработка информационной модели

Информационный объект – это описание некоторой сущности предметной области реального объекта, процесса, явления или события. Информационный объект образуется совокупностью логически взаимосвязанных реквизитов, представляющих качественные и количественные характеристики сущности [8].

Проанализировав возможные пользовательские действия была спроектирована информационная модель БД, представленная на рисунке 2.2.

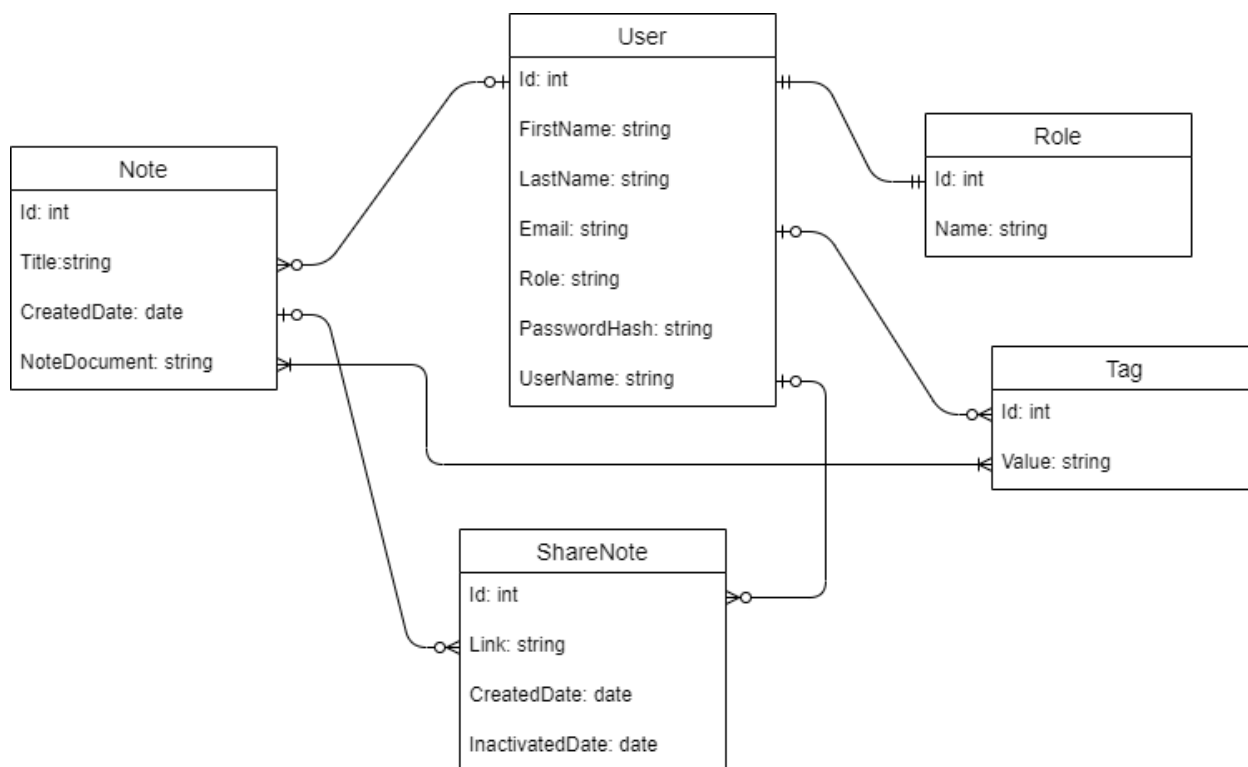


Рисунок 2.2 – Информационная модель БД

В процессе анализа предметной области были выделены следующие типы сущностей:

- **User** – сущность хранящая пользовательские учетные записи с требуемой пользовательской информацией. Атрибуты: Id – идентификатор пользователя; FirstName – имя пользователя; LastName – фамилия пользователя; Email – почтовый адрес пользователя; Role – роль пользователя в системе; PasswordHash – зашифрованный пароль пользователя; UserName – имя учетной записи в системе;
- **Role** – сущность для хранения ролей в системе. Атрибуты: Id – идентификатор роли; Name – наименование роли;
- **Note** – сущность для описание заметки. Атрибуты: Id – идентификатор заметки; Title – название заметки; CreatedDate – дата создания; NoteDocument – сами данные заметки, формат разметки html;
- **Tag** – сущность для описания тегов в системе. Атрибуты: Id – идентификатор тега; Value – значение тега;
- **ShareNote** – сущность для описания данных по режиму поделиться заметкой. Атрибуты: Id – идентификатор; Link – ссылка для возможности просмотреть заметку; CreateDate – дата создания ссылки; InactivedDate – дата инактивирования.

2.4 Разработка модели взаимодействия пользователя с интерфейсом

Проанализировав тенденции для реализации приложения будет использоваться подход одностраничных приложений.

SPA — это одностраничное веб-приложение, которое загружается на одну HTML-страницу. Благодаря динамическому обновлению с помощью JavaScript, во время использования не нужно перезагружать или подгружать дополнительные страницы. На практике это означает, что пользователь видит в браузере весь основной контент, а при прокрутке или переходах на другие страницы, вместо полной перезагрузки нужные элементы просто подгружаются.

В процессе работы пользователю может показаться, что он запустил не веб-сайт, а десктопное приложение, так как оно мгновенно реагирует на все его действия, без задержек [9].

К преимуществам одностраничным приложениям можно отнести:

- более быстрая загрузка страниц;
- улучшенное восприятие пользовательского интерфейса, поскольку загрузка данных с сервера происходит в фоновом режиме;
- нет необходимости писать код на сервере для визуализации страницы;
- разделение на Front-end и Back-end — разработку;
- упрощенная разработка под мобильные приложения; вы можете повторно использовать один и тот же серверный код для веб-приложения и мобильного приложения;

Данная технология также обладает следующими недостатками

- тяжелые клиентские фреймворки, которые нужно загружать на каждый клиент;
- веб-формы не компилируются, а потому их сложнее отлаживать, и они содержат больше уязвимостей, которыми могут воспользоваться злоумышленники;
- SEO — SPA усложняет оптимизацию сайта под поисковые движки. Поскольку большая часть веб-страницы строится на стороне клиента, боты поисковых систем видят страницу совершенно иначе, чем пользователь;

На рисунке 2.3 представлен пример сравнения жизненного цикла многостраничных приложений и одностраничных.

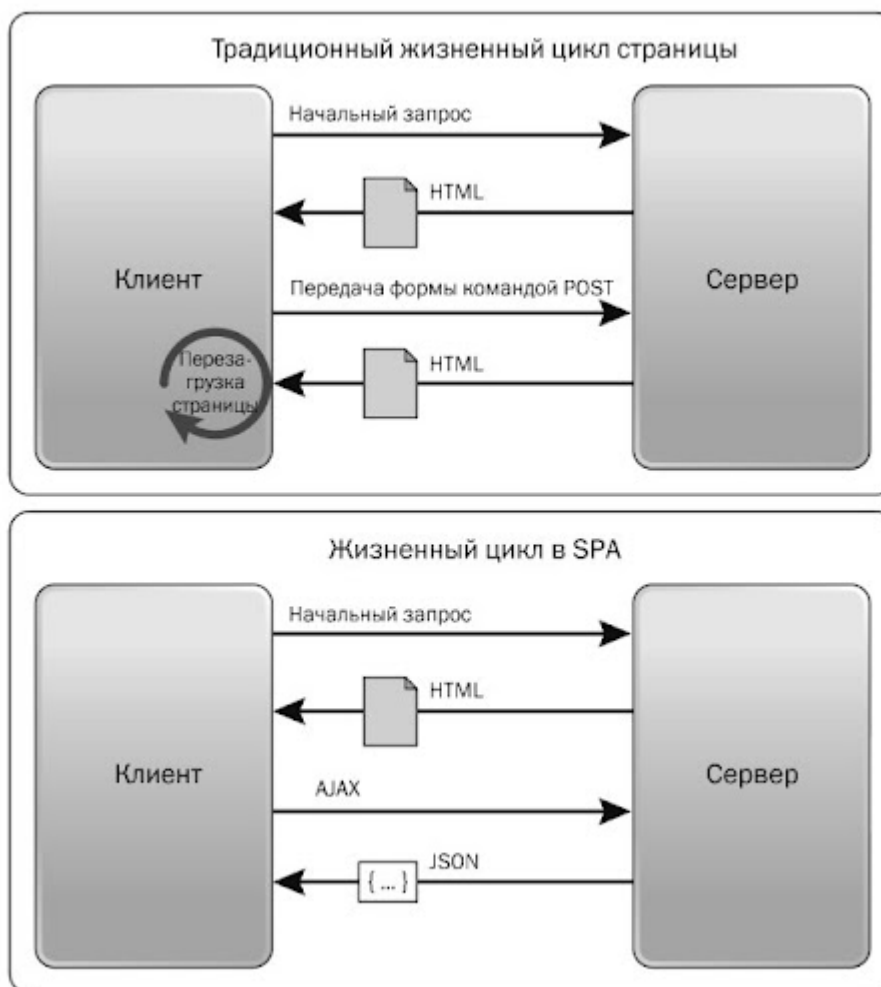


Рисунок 2.3 – Сравнение жизненного цикла многостраничных и одностраничных веб-приложений

2.5 Разработка технических требований к программному средству

Разрабатываемое программное решение должно обеспечивать корректное функционирование при развертывании программных модулей на сервере со следующими техническими характеристиками:

- Intel 2.2 ГГц или более быстродействующий процессор;
- оперативная память 4 Гбайт или более;
- доступный объем дискового пространства 100 Гбайт.

Для нормального функционирования клиентской части программного комплекса должны выполняться следующие технические требования:

- Pentium 2 ГГц или более быстродействующий процессор;
- оперативная память 2 Гбайт или более;
- 32- или 64-битная версия операционных систем Windows 10;
- браузер Google Chrome версии 95.x, Opera версии 81.x

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

3.1 Обоснование выбора среды разработки

Процесс разработки веб-приложения делиться на следующие этапы: разработка серверной части, разработка клиентской части, верстка. Для этого были использованы следующие инструменты и технологии: C#, ASP.NET Core, Vue3, JS, HTML, CSS, MS SQL Server 2019, Visual Studio 2019 Community.

C# — современный объектно-ориентированный и типобезопасный язык программирования. C# позволяет разработчикам создавать разные типы безопасных и надежных приложений, выполняющихся в .NET

ASP.NET Core является кроссплатформенной, высокопроизводительной средой с открытым исходным кодом для создания современных облачных приложений, подключенных к Интернету. ASP.NET Core позволяет выполнять следующие задачи:

- создавать веб-приложения и службы, приложения IoT и серверные части для мобильных приложений;
- использовать избранные средства разработки в Windows, macOS и Linux;
- выполнять развертывания в облаке или локальной среде;
- работать в .NET Core или .NET Framework.
- ASP.NET Core предоставляет следующие преимущества:
- единое решение для создания пользовательского веб-интерфейса и веб-API;
- интеграция современных клиентских платформ и рабочих процессов разработки;
- облачная система конфигурации на основе среды;
- встроенное введение зависимостей;
- упрощенный высокопроизводительный модульный конвейер HTTP-запросов;
- возможность размещения в IIS, Nginx, Apache, Docker или в собственном процессе;
- параллельное управление версиями приложения, ориентированное на .NET Core;
- инструментарий, упрощающий процесс современной веб-разработки;
- возможность сборки и запуска в ОС Windows, macOS и Linux;
- открытый исходный код и ориентация на сообщество [10].

Vue 3 представляет современный прогрессивный фреймворк, написанный на языке TypeScript и предназначенный для создания веб-

приложений на уровне клиента. Основная область применения данного фреймворка – это создание и организация пользовательского интерфейса.

Vue 3 имеет довольно небольшой размер и при этом обладает хорошей производительностью по сравнению с такими фреймворками как Angular или React, а также по сравнению с предыдущей версией - Vue.js 2.x. Поэтому неудивительно, что данный фреймворк в последнее время набирает обороты и становится все более популярным.

Одним из ключевых моментов в работе Vue 3 является виртуальный DOM. Структура веб-страницы, как правило, описывается с помощью DOM, которая представляет организацию элементов html на странице. Для взаимодействия с DOM (добавления, изменения, удаления html-элементов) применяется JavaScript. Но когда мы пытаемся манипулировать html-элементами с помощью JavaScript, то мы можем столкнуться со снижением производительности, особенно при изменении большого количества элементов. А операции над элементами могут занять некоторое время, что неизбежно скажется на пользовательском опыте. Однако если бы мы работали из кода js с объектами JavaScript, то операции производились бы быстрее.

Bootstrap – это инструментарий с открытым исходным кодом для разработки с помощью HTML (HyperText Markup Language), CSS (Cascading Style Sheets), JS (JavaScript). Он включает в себя множество разных компонентов для веб-сайта: типографику, веб-формы, кнопки, блоки навигации и другие. Основные преимущества Bootstrap:

- адаптивность – дизайн сайта будет корректно отображаться на экранах устройств разных размеров вне зависимости от диагонали;
- кросс-браузерность – одинаковое отображение во всех браузерах;
- легкость в использовании;
- понятный код – позволяет писать качественный и понятный код, что облегчает чтение кода для других разработчиков;

HTML – язык разметки (маркировки) гипертекста. HTML дает возможность производить переход от одной части текста к другой, и, что замечательно, эти части могут храниться на совершенно разных компьютерах. Он создан специально для разметки веб-страниц. Именно язык разметки дает браузеру необходимые инструкции о том, как отображать тексты и другие элементы страницы на мониторе.

CSS — язык таблиц стилей, который позволяет прикреплять стиль (например, шрифты и цвет) к структурированным документам (например, документам HTML и приложениям XML). Обычно CSS-стили используются для создания и изменения стиля элементов веб-страниц и пользовательских интерфейсов, написанных на языках HTML и XHTML, но также могут быть применены к любому виду XML-документа, в том числе XML, SVG и XUL. Отделяя стиль представления документов от содержимого документов, CSS упрощает создание веб-страниц и обслуживание сайтов.

JavaScript – это интерпретируемый язык программирования, разработанный для взаимодействия с веб-страницами. Он используется для описания сценариев для активных веб-страниц. Программа на JavaScript встраивается непосредственно в исходный текст HTML-документа и интерпретируется браузером по мере загрузки этого документа. С помощью JavaScript можно динамически изменять текст загружаемого HTML документа и реагировать на события, связанные с действиями посетителя или изменениями состояния документа или окна.

3.2 Разработка структурной схемы программного средства

Для разработки веб-приложения используется одностраничная архитектура с использованием фреймворка Vue3 для разработки клиентской части и ASP.NET Core для разработки серверной части.

Разработка серверной части предполагает создание REST API.

REST API – это прикладной программный интерфейс, который использует HTTP-запросы для получения, извлечения, размещения и удаления данных. Аббревиатура REST в контексте API расшифровывается как «передача состояния представления» (Representational State Transfer).

REST API включает следующие принципы:

- единый интерфейс;
- разграничение клиента и сервера;
- нет сохранения состояния;
- кэширование всегда разрешено;
- многоуровневая система;
- код предоставляется по запросу.

Для передачи запросов используется протокол HTTPS.

HTTPS — расширение протокола HTTP для поддержки шифрования в целях повышения безопасности.

На рисунке 3.1 представлена модель передачи данных использую REST API. На примере авторизации в системе.

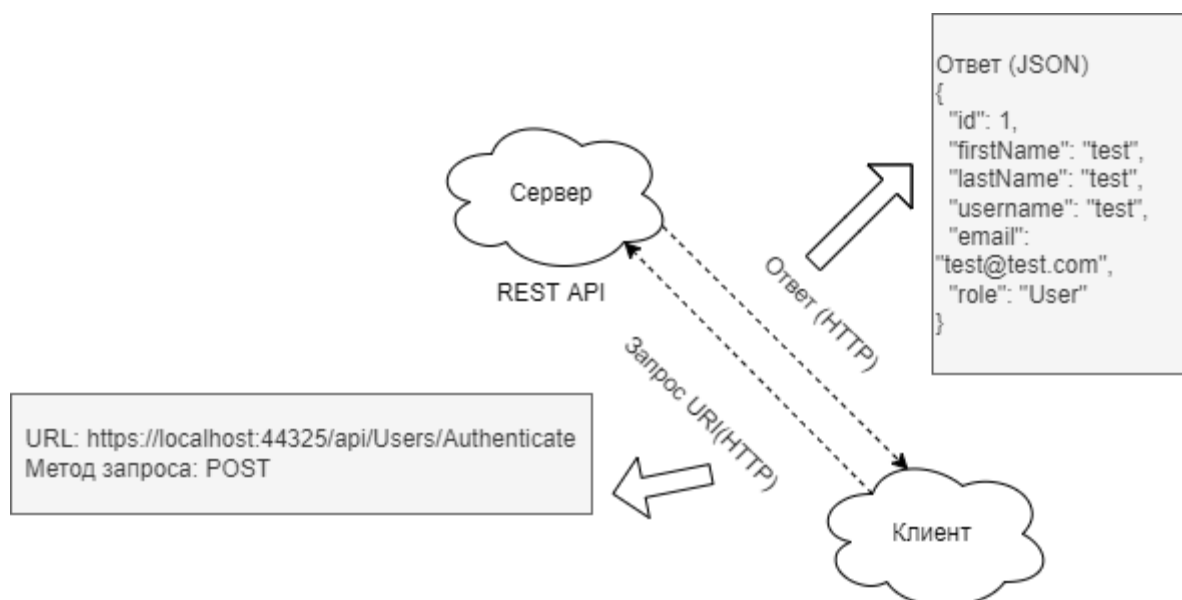


Рисунок 3.1 – Модель передачи данных с использованием REST API

Для разработки серверной части используется трехуровневая приложения. Данная модель архитектуры подразумевает разделение приложения на три уровня. На рисунке 3.2 представлена модель 3 уровневой архитектуры.

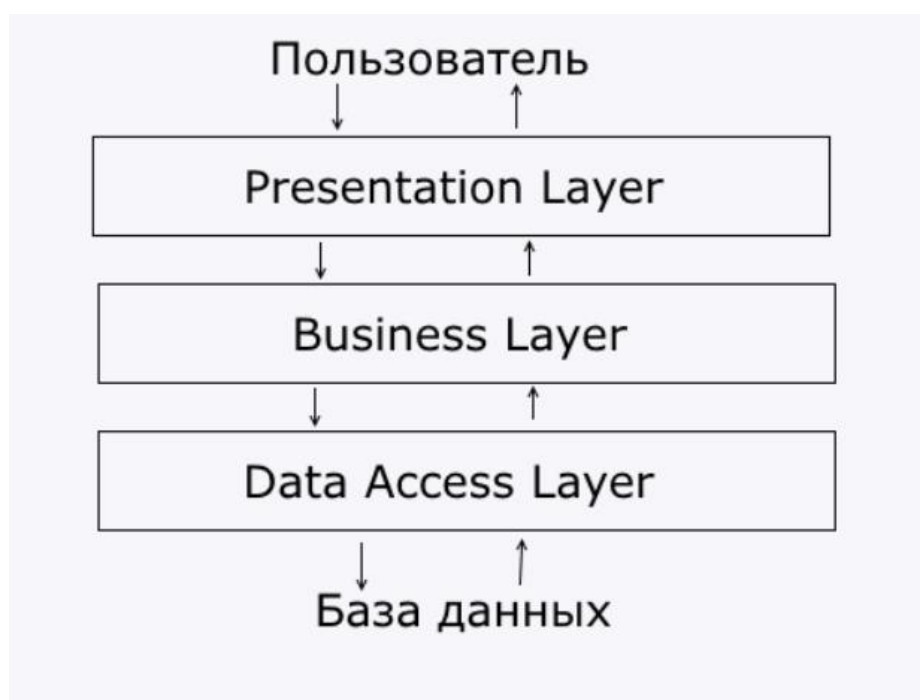


Рисунок 3.2 – Модель трехуровневой архитектуры

Presentation layer (уровень представления) – это тот уровень, с которым непосредственно взаимодействует пользователь. Этот уровень включает

компоненты пользовательского интерфейса, механизм получения ввода от пользователя.

Business layer (уровень бизнес-логики) – содержит набор компонентов, которые отвечают за обработку полученных от уровня представлений данных, реализует всю необходимую логику приложения, все вычисления, взаимодействует с базой данных и передает уровню представления результат обработки.

Data Access layer (уровень доступа к данным) – хранит модели, описывающие используемые сущности, также здесь размещаются специфичные классы для работы с разными технологиями доступа к данным, например, класс контекста данных Entity Framework.

На рисунке 3.3 представлена трехуровневая архитектура приложения.

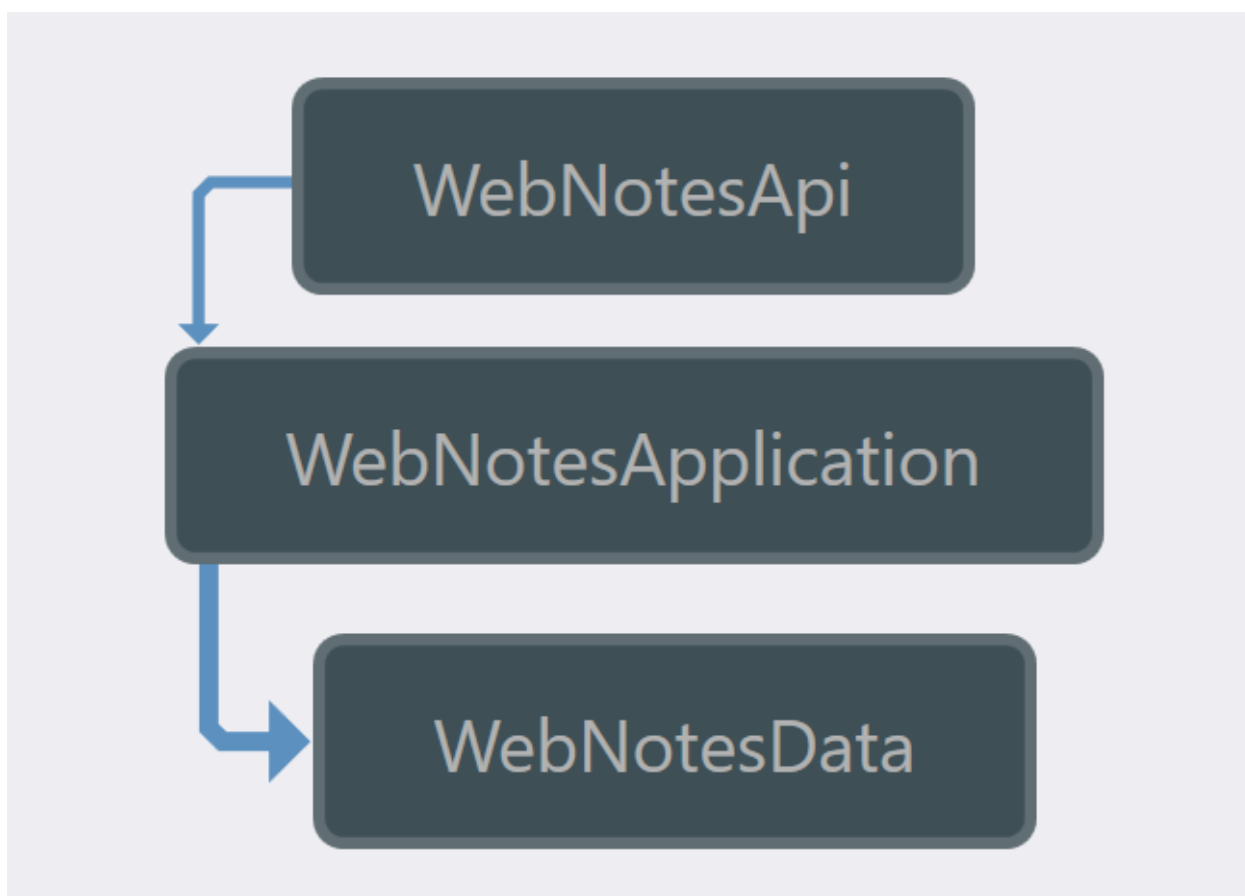


Рисунок 3.3 – Трехуровневая архитектура приложения

Для разработки клиентской части с использованием Vue3 подключены следующие модули:

- vue-router – официальная библиотека маршрутизации для. Она глубоко интегрируется с Vue и позволяет легко создавать SPA-приложения;
- vuex – паттерн управления состоянием и библиотека для приложений на Vue. Он служит централизованным хранилищем данных для

всех компонентов приложения с правилами, гарантирующими, что состояние может быть изменено только предсказуемым образом.

На рисунке 3.4 представлена структурная схема работы Vuex.

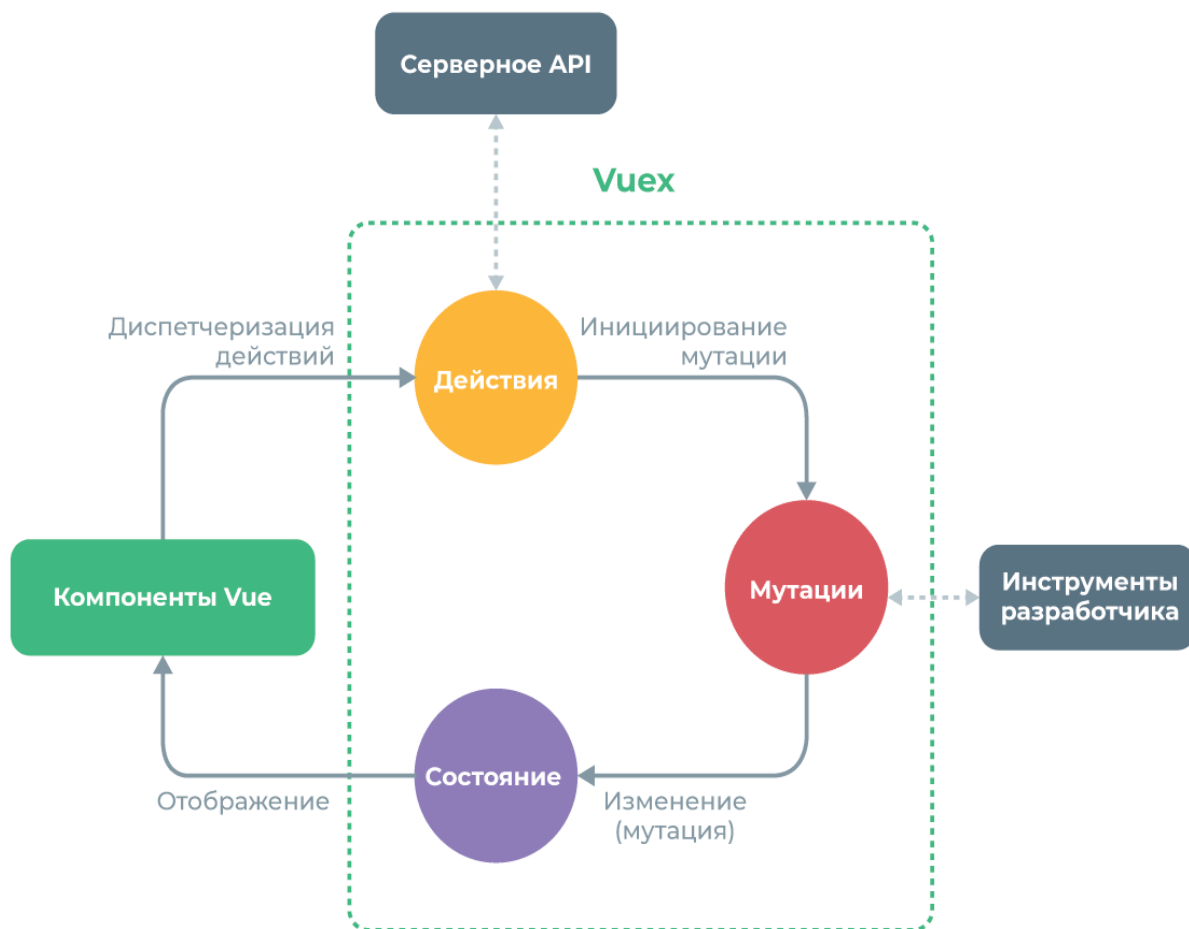


Рисунок 3.4 – Структурная схема работы Vuex

Структура клиентского приложения разделяется на следующие модули:

- хранилище состояний – в данном модуле будут храниться данные для использования глобально всеми компонентами;
- маршрутизация – данный модуль позволяет переходить и регистрировать различные страницы в системе и позволять управлять перенаправлением между страницами;
- представления – данный модуль, описывает сами страницы, которые будут представлены пользователю, данные страницы могут использовать компоненты, которые описывает небольшие функциональные блоки;
- провайдеры – сервисные модули, которые используются для переадресации данных между клиентской и серверной частью. Для этого используется библиотека Axios.

На рисунке 3.5 представлена клиентская структура взаимодействия компонентов.

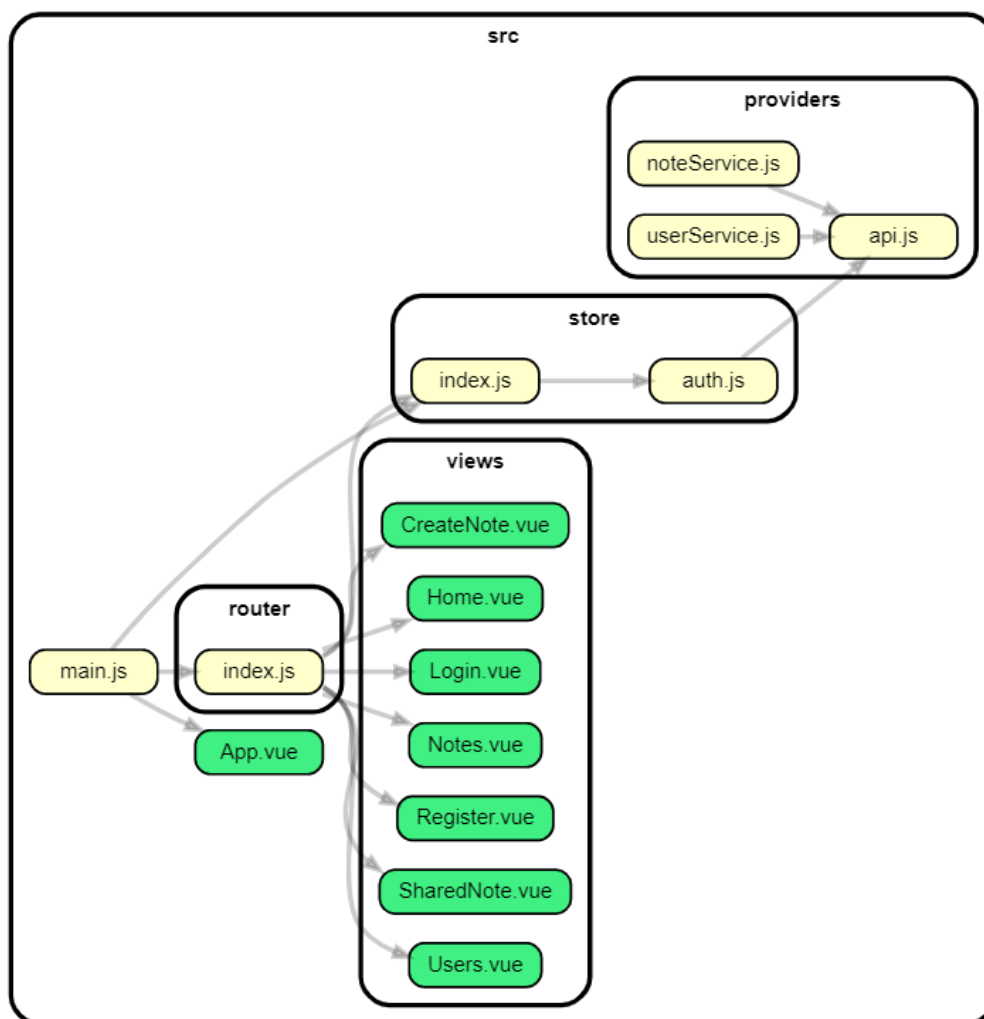


Рисунок 3.5 – Структура взаимодействия модулей клиентской части приложения

3.3 Разработка структуры классов

Диаграмма классов (class diagram) – основной способ описания структуры системы. На диаграмме классов применяется один основной тип сущностей: классы (включая многочисленные частные случаи классов: интерфейсы, примитивные типы, классы-ассоциации и многие другие), между которыми устанавливаются следующие основные типы отношений:

- ассоциация между классами;
- обобщение между классами;
- зависимости (различных типов) между классами и между классами и интерфейсами.

На рисунке 3.6 представлена диаграмма классов уровня представления.

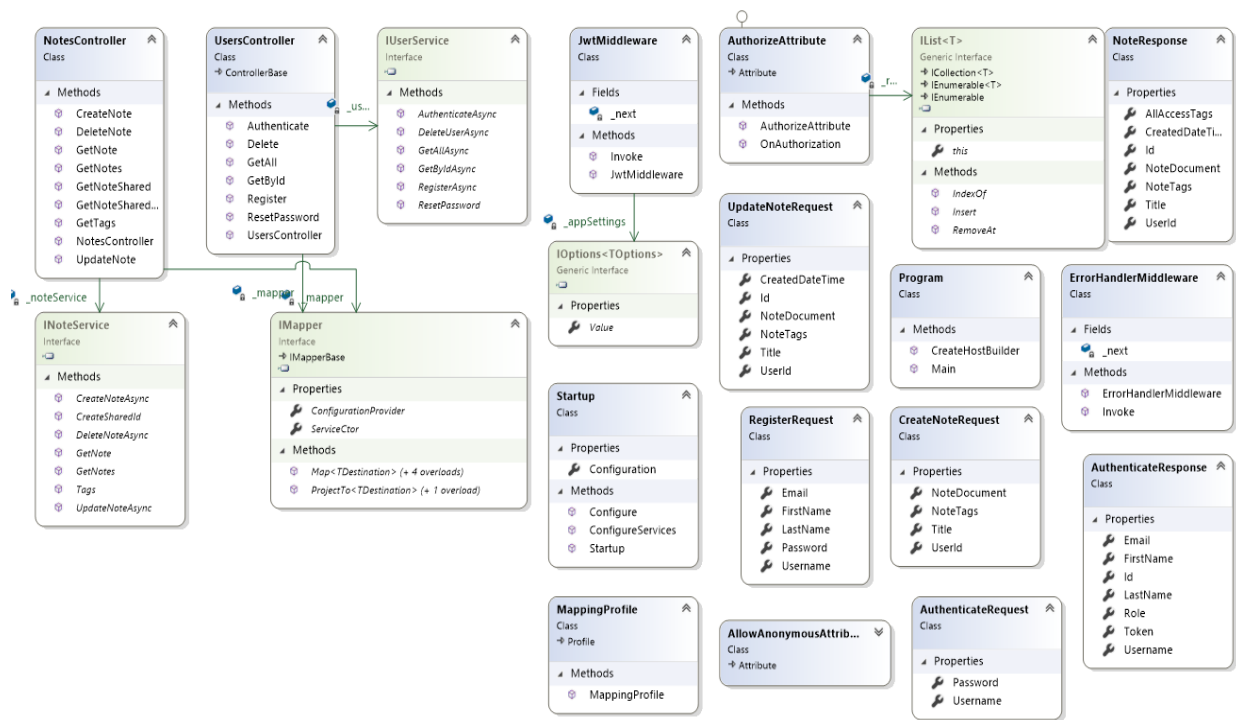


Рисунок 3.6 – Диаграмма классов уровня представления

Диаграмма классов уровня бизнес-логики представлена на рисунке 3.7.

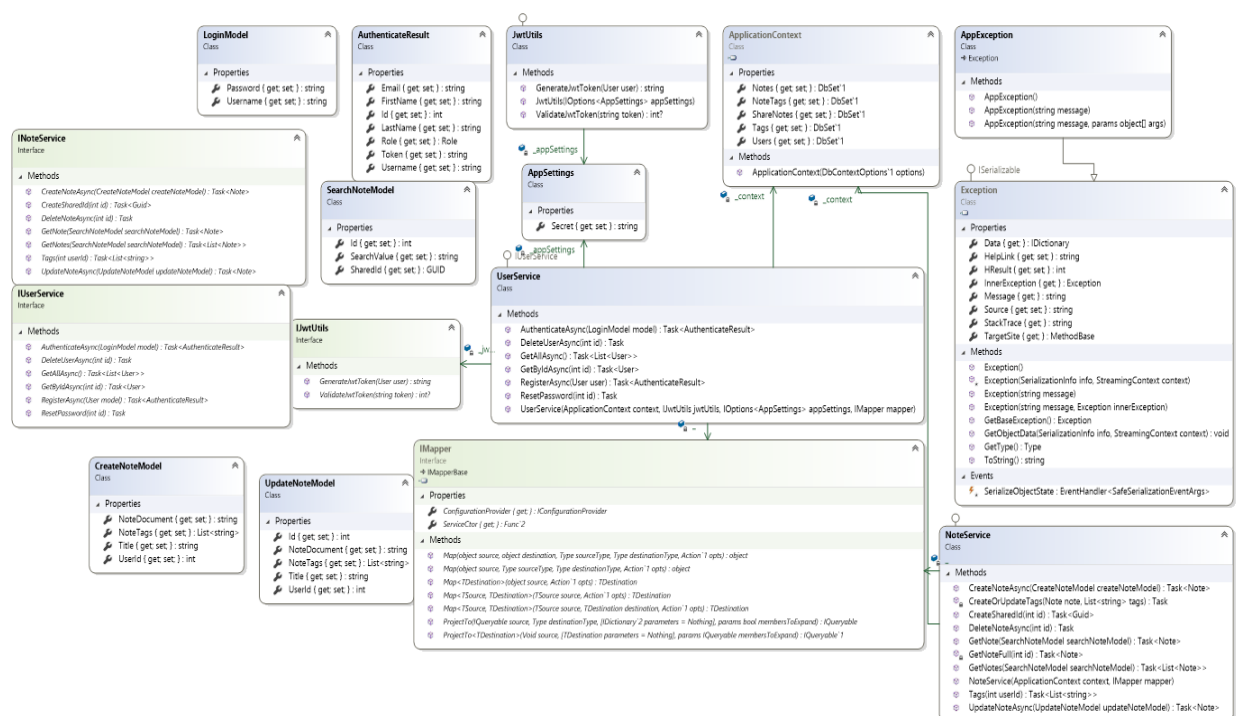


Рисунок 3.7 – Диаграмма классов уровня бизнес-логики

Диаграмма классов уровня доступа к данным представлена на рисунке 3.8.

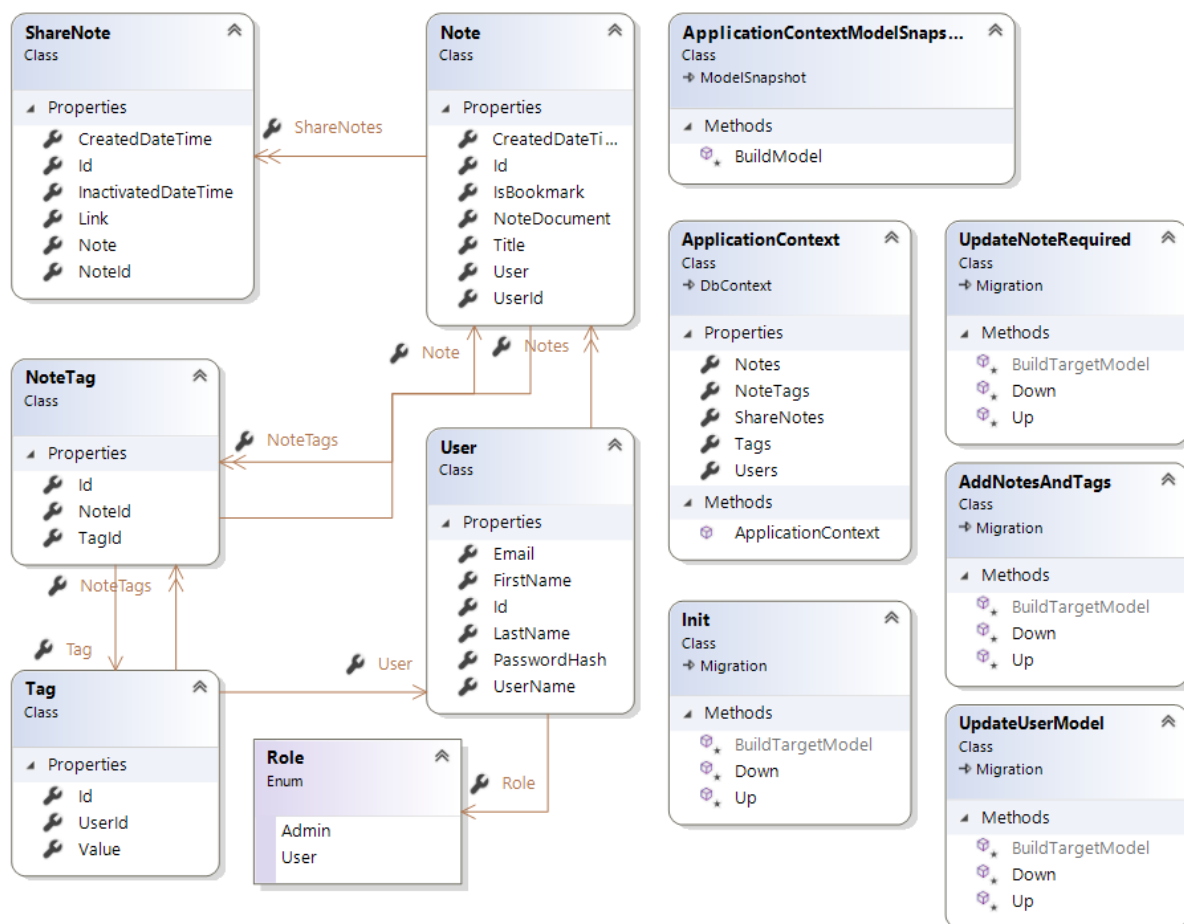


Рисунок 3.8 – Диаграмма классов уровня доступа к данным

3.4 Разработка физической модели данных

После проектирования информационной модели данных, она была реализована с использованием реляционных баз данных MS SQL. И в ходе проектирования была проведена нормализация и не все сущности необходимы на уровне БД.

Достоинства реляционного подхода:

- наличие небольшого набора абстракций, которые позволяют сравнительно просто моделировать большую часть распространенных предметных областей и допускают точные формальные определения, оставаясь интуитивно понятными;
- наличие простого и в то же время мощного математического аппарата, опирающегося главным образом на теорию множеств и математическую логику и обеспечивающего теоретический базис реляционного подхода к организации БД;
- возможность ненавигационного манипулирования данными без необходимости знания конкретной физической организации баз данных во внешней памяти [11].

На рисунке 3.9 представлена физическая диаграмма базы данных.

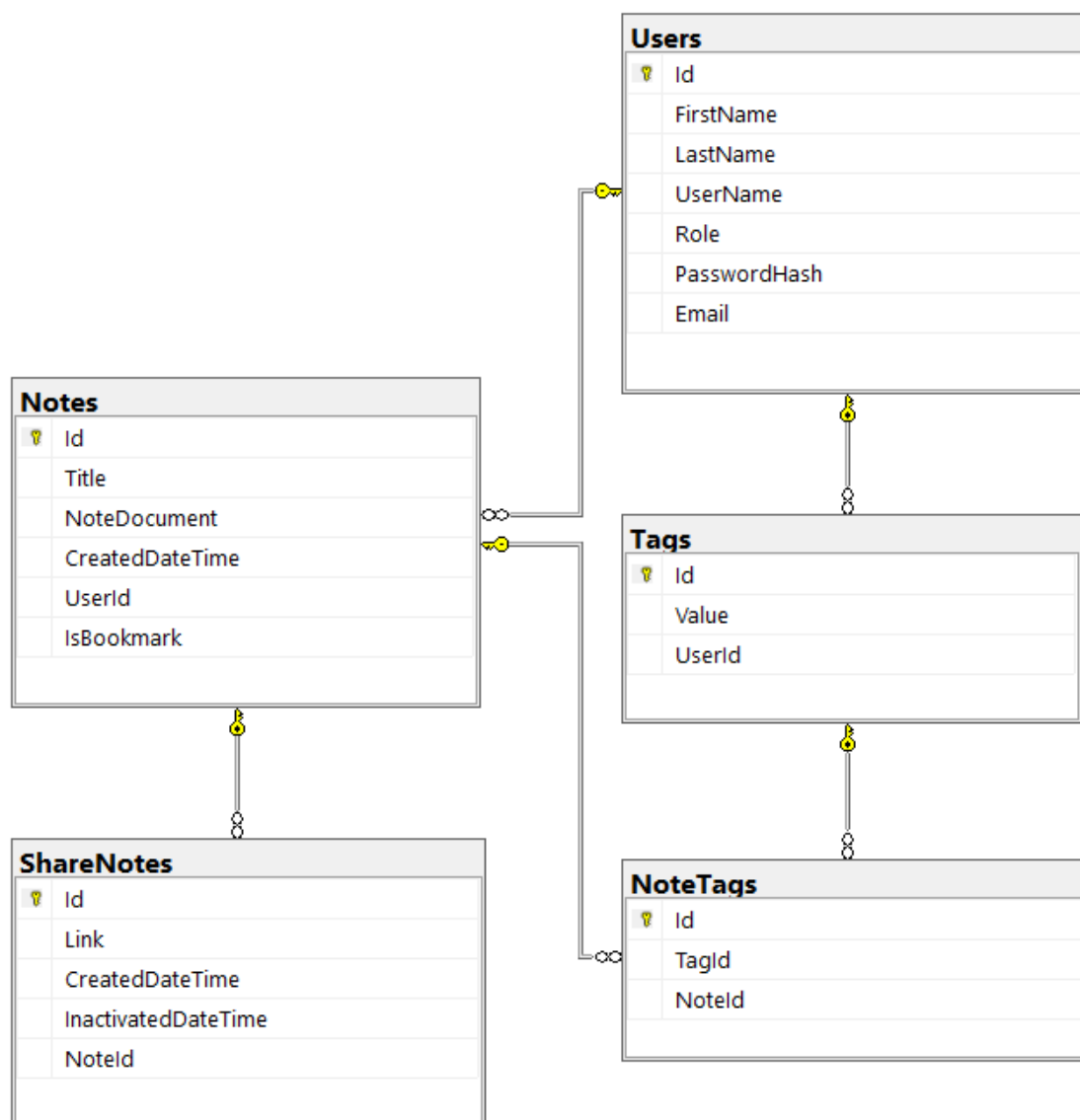


Рисунок 3.9 – Физическая модель данных БД

Формализованное описание объектов представлено в таблицах 3.1-3.5. Описание таблицы «Users» представлено в таблице 3.1.

Таблица 3.1 – Описание таблицы «Users»

Поле	Тип данных	Описание
Id	INT	Идентификатор пользователя
FirstName	NVARCHAR(MAX)	Имя пользователя
LastName	NVARCHAR(MAX)	Фамилия пользователя
UserName	NVARCHAR(MAX)	Имя учетной записи в системе

Продолжение таблицы 3.1

Поле	Тип данных	Описание
Role	INT	Роль пользователя
PasswordHash	NVARCHAR(MAX)	Хеш пароля
Email	NVARCHAR(MAX)	Почтовый адрес пользователя

Описание таблицы «Notes» представлено в таблице 3.1.

Таблица 3.2 – Описание таблицы «Notes»

Поле	Тип данных	Описание
Id	INT	Идентификатор заметки
Title	NVARCHAR(MAX)	Название заметки
NoteDocument	NVARCHAR(MAX)	Отформатированный текст заметки(в виде HTML
CreatedDateTime	DATETIME2(7)	Дата создания заметки
UserId	INT	Внешний ключ: Идентификатор пользователя
IsBookmark	BIT	Флаг для важных заметок

Описание таблицы «Tags» представлено в таблице 3.3.

Таблица 3.3 – Описание таблицы «Tags»

Поле	Тип данных	Описание
Id	INT	Идентификатор категории
Value	NVARCHAR(MAX)	Значение категории
UserId	INT	Внешний ключ: Идентификатор пользователя

Описание таблицы «NoteTags» представлено в таблице 3.4. Данная таблицы выступает связующей между таблицами «Notes» и «Tags», для образования связи многие ко многим

Таблица 3.4 – Описание таблицы «NoteTags»

Поле	Тип данных	Описание
Id	INT	Идентификатор
TagId	INT	Внешний ключ: Идентификатор категории
NoteId	INT	Внешний ключ: Идентификатор заметки

Описание таблицы «ShareNotes» представлено в таблице 3.4.

Таблица 3.5 – Описание таблицы «ShareNotes»

Поле	Тип данных	Описание
Id	INT	Идентификатор записи поделиться заметкой
Link	NVARCHAR(MAX)	Ссылка на просмотр заметки
CreatedDateTime	DATETIME2(7)	Дата создания
InactivatedDateTime	DATETIME2(7)	Дата отмены работы ссылки
NoteId	INT	Внешний ключ: Идентификатор заметки

3.5 Проектирование алгоритмов работы программного средства

Для определения логики написания программы были спроектированы следующие алгоритмы работы программного средства.

3.5.1 Алгоритм регистрации на стороне сервера

Схема работы алгоритма регистрации на стороне сервера представлена на рисунке 3.10.

Данный алгоритм отображает схему работу регистрации на сервере приложения, после того как веб-приложение отправляет данные пользователя для регистрации. Результатом выполнения обработки запроса будет или ошибка, или успешный ответ с данными пользователями и JWT токеном.

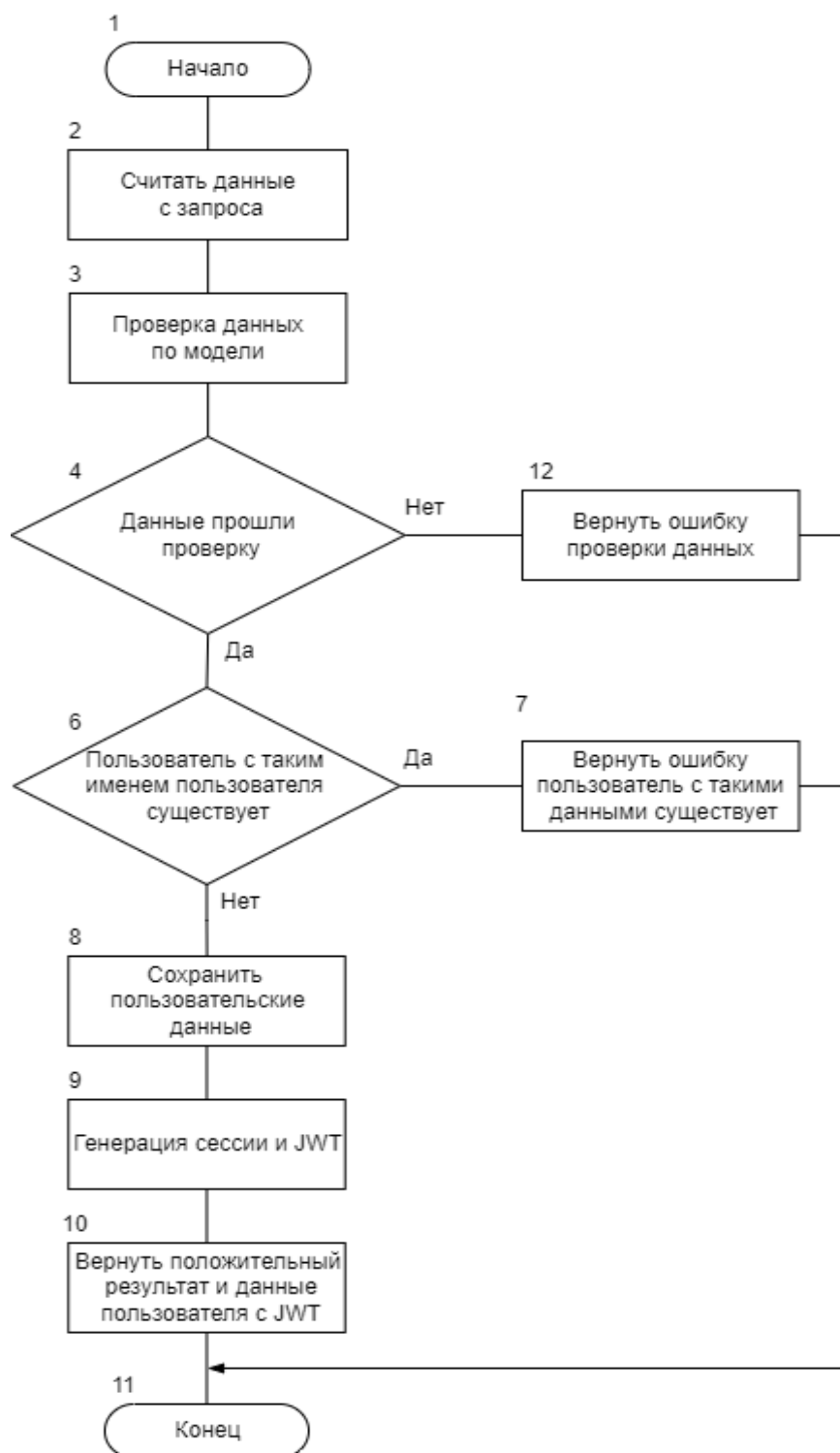


Рисунок 3.10 – Блок-схема работы алгоритма регистрации на стороне сервера

3.5.2 Алгоритм проверки уровня доступа на стороне сервера

Схема работы алгоритма регистрации на стороне сервера представлена на рисунке 3.11. Данный алгоритм демонстрирует как происходит проверка уровня доступа пользователя при отправке запроса на сторону сервера.

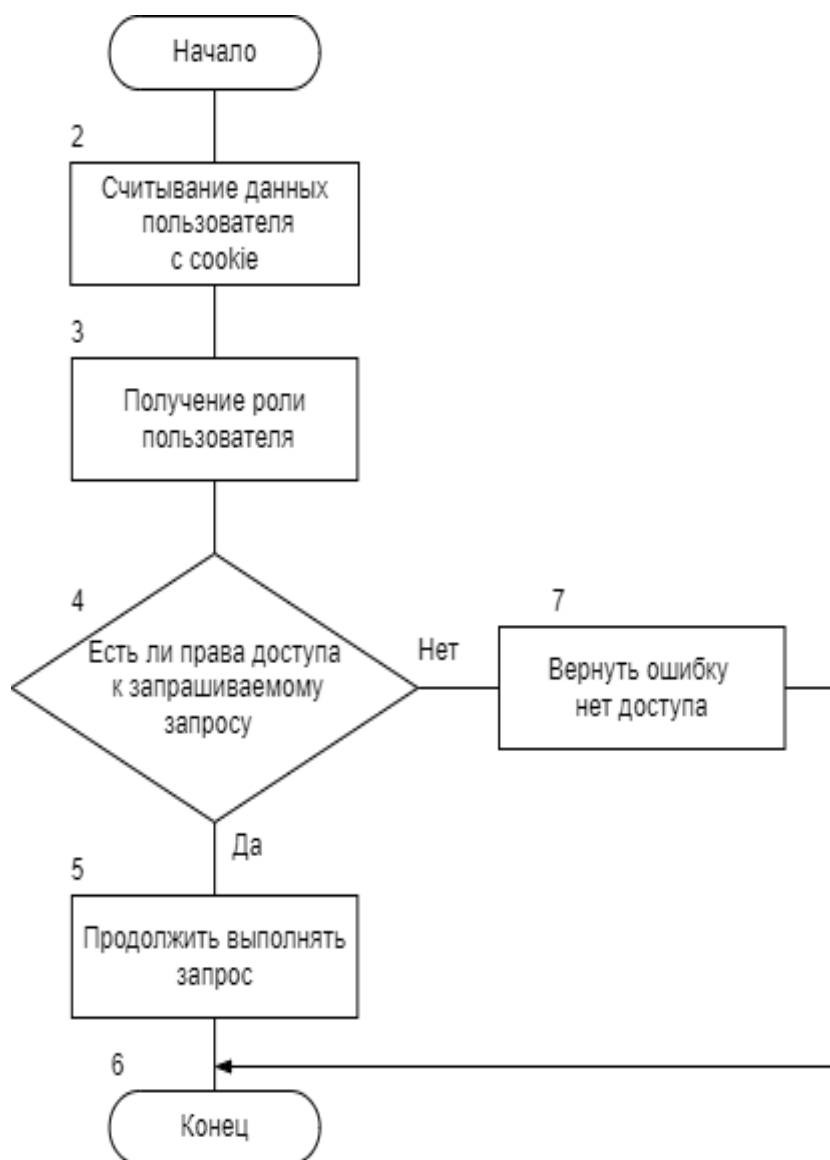


Рисунок 3.11 – Блок-схема алгоритма проверки уровня доступа на стороне сервера

3.5.3 Алгоритм от лица пользователя на стороне клиента

На рисунке 3.12 представлена блок схема алгоритма от лица пользователя на клиентской стороне программы. Данный алгоритм демонстрирует поведение системы на основе поведения пользователя, позволяя выдавать возможности работы с заметками после авторизации в системе.

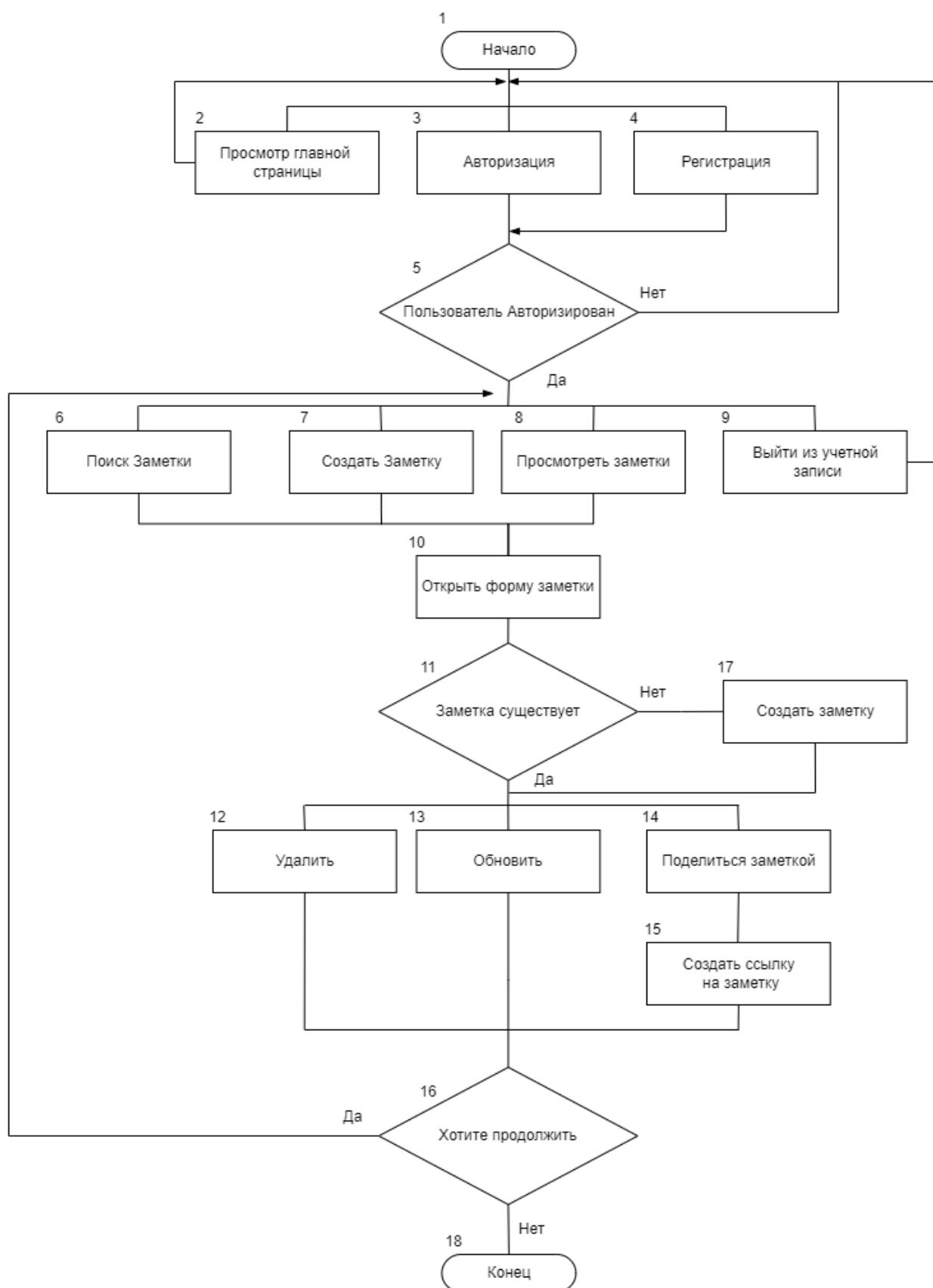


Рисунок 3.12 – Блок-схема алгоритма программы от лица пользователя на клиентской стороне

4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Тестирование программного средства – процесс исследования, испытания ПС с целью получения информации о качестве продукта.

Существующие на сегодня методы тестирования ПС не позволяют однозначно и полностью выявить все дефекты и установить корректность функционирования анализируемой программы, поэтому все существующие методы тестирования действуют в рамках формального процесса проверки исследуемого или разрабатываемого ПС.

Такой процесс формальной проверки, или верификации, может доказать, что дефекты отсутствуют с точки зрения используемого метода. То есть нет никакой возможности точно установить или гарантировать отсутствие дефектов в программном продукте с учётом человеческого фактора, присутствующего на всех этапах жизненного цикла ПС.

Существуют следующие виды тестирования:

- функциональные;
- нефункциональные;
- связанные с изменениями;
- модульное.

Функциональное тестирование – это тестирование программного средства в целях проверки реализуемости функциональных требований, то есть способности программного средства в определённых условиях решать задачи, нужные пользователям.

Нефункциональное тестирование описывает тесты, необходимые для определения характеристик программного средства, которые могут быть измерены различными величинами.

Модульное тестирование – процесс в программировании, позволяющий проверить на корректность единицы исходного кода, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки.

В рамках разработки дипломного проекта функциональное тестирование реализовано с помощью набора тест-кейсов.

Тест-кейс – это профессиональная документация тестировщика, последовательность действий, направленная на проверку какого-либо функционала, описывающая как прийти к фактическому результату.

Тест-кейс №1 – Проверка работы сайта на разных ОС. Тест кейс представлен в таблице 4.1.

Таблица 4.1 – Тест-кейс №1

Название	Шаги воспроизведения	Ожидаемый результат	Фактический результат
Проверка работы сайта на разных ОС	1 Открыть веб-приложение 2 Проверить отображение приложения в браузере Opera версии 81.x 3 Проверить отображение приложения в браузере Google Chrome версии 95.x	Отображается все корректно	Отображается все корректно

Тест-кейс №2 – Проверка отображения главной страницы. Тест кейс представлен в таблице 4.2.

Таблица 4.2 – Тест-кейс №2

Название	Шаги воспроизведения	Ожидаемый результат	Фактический результат
Проверка отображения главной страницы	1 Открыть веб-приложение 2 Открыть главную страницу	Отображается главная страница с описанием приложения	Отображается главная страница с описанием приложения

Тест-кейс №3 – Регистрация пользователя: успешное создание пользователя. Тест кейс представлен в таблице 4.3.

Таблица 4.3 – Тест-кейс №3

Название	Шаги воспроизведения	Ожидаемый результат	Фактический результат
Регистрация пользователя: успешное создание пользователя	1 Открыть веб-приложение 2 Нажать на панели навигации «Регистрация» 3 Заполнить все поля корректно 4 Нажать кнопку «Регистрация»	Пользователь успешно зарегистрирован	Пользователь успешно зарегистрирован

Тест-кейс №4 – Регистрации пользователя: ошибка при регистрации.
Тест кейс представлен в таблице 4.4.

Таблица 4.4 – Тест-кейс №4

Название	Шаги воспроизведения	Ожидаемый результат	Фактический результат
Регистрации пользователя: ошибка при регистрации	1 Открыть веб-приложение 2 Нажать на панели навигации «Регистрация» 3 Заполнить все поля корректно и написать существующие имя пользователя 4 Нажать кнопку «Регистрация»	Получить ошибку, что пользователь с таким именем существует	Получить ошибку, что пользователь с таким именем существует

Тест-кейс №5 – Вход в систему. Тест кейс представлен в таблице 4.5.

Таблица 4.5 – Тест-кейс №5

Название	Шаги воспроизведения	Ожидаемый результат	Фактический результат
Вход в систему	1 Открыть веб-приложение 2 Нажать на панели навигации «Вход» 3 Заполнить все поля корректно 4 Нажать кнопку «Вход»	Пользователь успешно вошел в систему	Пользователь успешно вошел в систему

Тест-кейс №6 – Открыть форму создания заметки. Тест кейс представлен в таблице 4.6.

Таблица 4.6 – Тест-кейс №6

Название	Шаги воспроизведения	Ожидаемый результат	Фактический результат
Открыть форму создания заметки	1 Открыть веб-приложение 2 Войти в систему	Откроется форма создания заметки	Откроется форма создания заметки

Продолжение таблицы 4.6

Название	Шаги воспроизведения	Ожидаемый результат	Фактический результат
	3 В навигационной панели нажать «Создать заметку»		

Тест-кейс №7 – Создание заметки. Тест кейс представлен в таблице 4.7.

Таблица 4.7 – Тест-кейс №7

Название	Шаги воспроизведения	Ожидаемый результат	Фактический результат
Создание заметки через форму создания	1 Открыть веб-приложение 2 Войти в систему 3 В навигационной панели нажать «Создать заметку» 4 Ввести все корректные данные 5 Нажать на кнопку «Создать»	Заметка успешно создалась	Заметка успешно создалась

Тест-кейс №8 – Обновить заметку через форму создания. Тест кейс представлен в таблице 4.8.

Таблица 4.8 – Тест-кейс №8

Название	Шаги воспроизведения	Ожидаемый результат	Фактический результат
Обновить заметку через форму создания	1 Открыть веб-приложение 2 Войти в систему 3 В навигационной панели нажать «Создать заметку» 4 Ввести все корректные данные 5 Нажать на кнопку «Создать» 6 Обновить данные 7 Нажать на кнопку «Обновить»	Заметка успешно обновилась	Заметка успешно обновилась

Тест-кейс №9 – Просмотр списка заметок и поиск. Тест кейс представлен в таблице 4.9.

Таблица 4.9 – Тест-кейс №9

Название	Шаги воспроизведения	Ожидаемый результат	Фактический результат
Просмотр списка заметок и поиск	1 Открыть веб-приложение 2 Войти в систему 3 В навигационной панели нажать «Заметки» 4 Ввести в поисковую строку существующее значение тега или текста заметки 5 Нажать на кнопку «Поиска» 6 Нажать на карточку заметки	Страница с результатом отображается и при нажатии на заметку заметка подгружается	Страница с результатом отображается и при нажатии на заметку заметка подгружается

Как видно из результатов функциональное тестирование тестирования, приложение работает исправно.

Было проведено модульное тестирование. Результаты первичного тестирования представлена на рисунке 4.1. После запуска первичного модульного тестирования были выявлены следующие ошибки:

- строка подключения к базе данных была не корректна;
- ошибки в передачи данных между контроллером и представлением;
- ошибки в формировании LINQ запросов;
- не правильная передача данных для создания тестов;
- отсутствует валидация на модели;
- проблемы с доступом к контроллерам;
- синтаксические ошибки в коде.

✖ WeNotesTest (20)	938 ms	
✖ WeNotesTest (20)	938 ms	
✔ NoteControllerTests	130 ms	
✔ CreateNote_Corr...	122 ms	
✔ CreateNote_Co...	< 1 ms	
✔ CreateNote_Co...	< 1 ms	
✔ CreateNote_Co...	122 ms	
✔ CreateNote_Corr...	1 ms	
✔ CreateNote_Co...	< 1 ms	
✔ CreateNote_Co...	< 1 ms	
✔ CreateNote_Co...	1 ms	
✔ DeleteNote_Corr...	4 ms	
✔ DeleteNote_Co...	4 ms	
✔ DeleteNote_Co...	< 1 ms	
✔ UpdateNote_Cor...	3 ms	
✔ UpdateNote_C...	< 1 ms	
✔ UpdateNote_C...	< 1 ms	
✔ UpdateNote_C...	3 ms	
✖ NoteServiceTests (9)	808 ms	
✖ CreateNote_Not...	806 ms	
✖ CreateNote_N...	6 ms	...
✖ CreateNote_N...	16 ms	...
✖ CreateNote_N...	784 ms	...
✖ DeleteNote_Not...	1 ms	
✖ DeleteNote_N...	< 1 ms	...
✖ DeleteNote_N...	< 1 ms	...
✖ DeleteNote_N...	1 ms	...
✖ UpdateNote_No...	1 ms	
✖ UpdateNote_N...	< 1 ms	...
✖ UpdateNote_N...	< 1 ms	...
✖ UpdateNote_N...	1 ms	...

Рисунок 4.1 – Первичный результат модульного тестирования

После получения результатов, были проведены исправления кода и выполнено повторное тестирование после проведения изменений. Результат итогового выполнения модульного тестирования представлен на рисунке 4.2.

✓ WeNotesTest (20)	875 ms
✓ WeNotesTest (20)	875 ms
✓ NoteControllerTests	122 ms
✓ CreateNote_Corr...	114 ms
✓ CreateNote_Co...	< 1 ms
✓ CreateNote_Co...	< 1 ms
✓ CreateNote_Co...	114 ms
✓ CreateNote_Corr...	1 ms
✓ CreateNote_Co...	< 1 ms
✓ CreateNote_Co...	< 1 ms
✓ CreateNote_Co...	1 ms
✓ DeleteNote_Corr...	3 ms
✓ DeleteNote_Co...	3 ms
✓ DeleteNote_Co...	< 1 ms
✓ UpdateNote_Cor...	4 ms
✓ UpdateNote_C...	< 1 ms
✓ UpdateNote_C...	< 1 ms
✓ UpdateNote_C...	4 ms
✓ NoteServiceTests (9)	753 ms
✓ CreateNote_Not...	752 ms
✓ CreateNote_N...	5 ms
✓ CreateNote_N...	19 ms
✓ CreateNote_N...	728 ms
✓ DeleteNote_Not...	< 1 ms
✓ DeleteNote_N...	< 1 ms
✓ DeleteNote_N...	< 1 ms
✓ DeleteNote_N...	< 1 ms
✓ UpdateNote_No...	1 ms
✓ UpdateNote_N...	< 1 ms
✓ UpdateNote_N...	< 1 ms
✓ UpdateNote_N...	1 ms

Рисунок 4.2 – Результат итогового выполнения модульного тестирования

5 МЕТОДИКА ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО СРЕДСТВА

После входа в программное средство открывается главная страница с описанием основных функций приложения. Данная страница доступна для всех типов пользователей. Главная страница приложения представлена на рисунке 5.1.

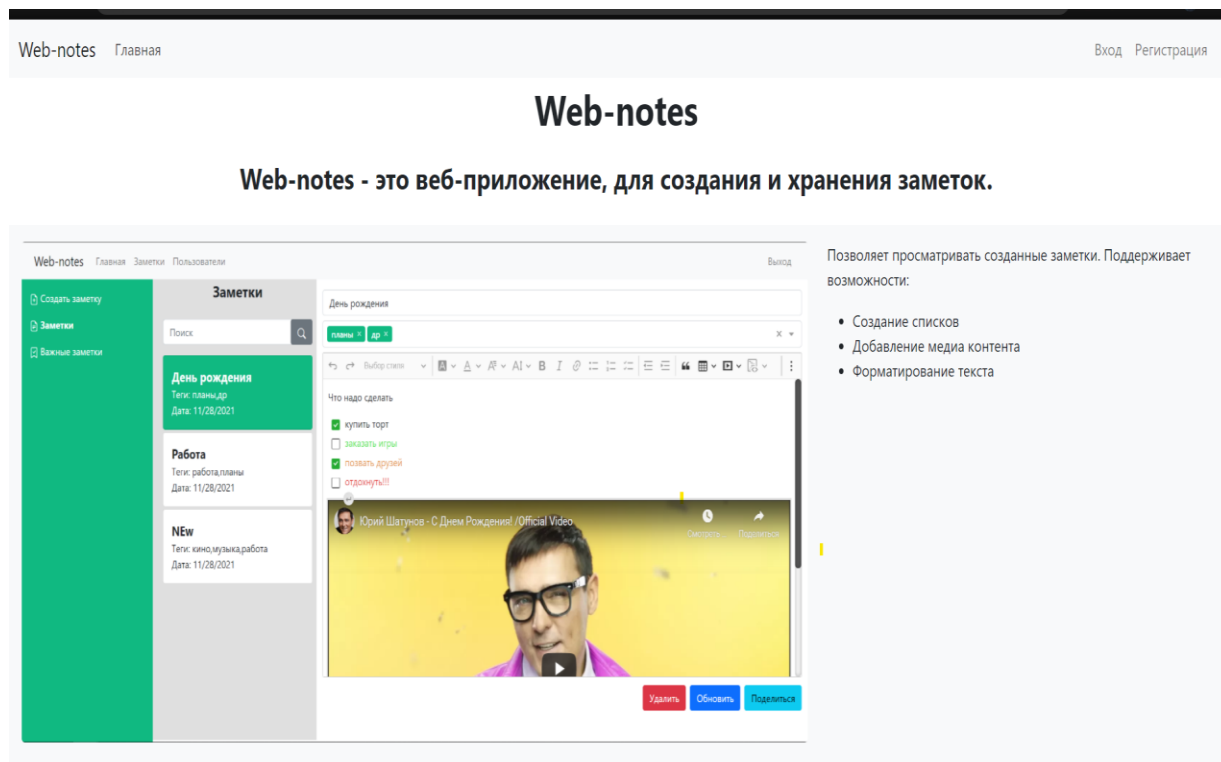
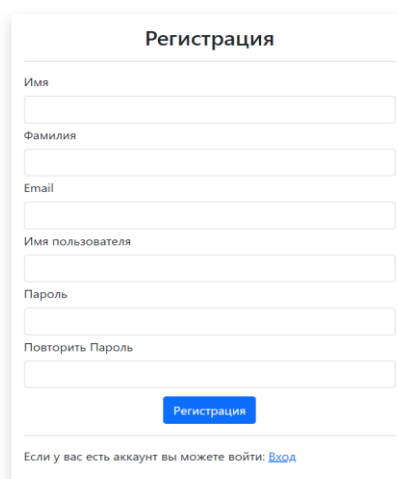


Рисунок 5.1 – Главная страница веб-приложения

5.1 Регистрация пользователя

Для того чтобы зарегистрироваться в системе требуется на панели навигации нажать на элемент управления «Регистрация». После нажатия открывается форма регистрации, где пользователю нужно ввести все обязательные поля. Данная функция доступна только не авторизованным пользователям. Форма регистрации представлена на рисунке 5.2.



Регистрация

Имя

Фамилия

Email

Имя пользователя

Пароль

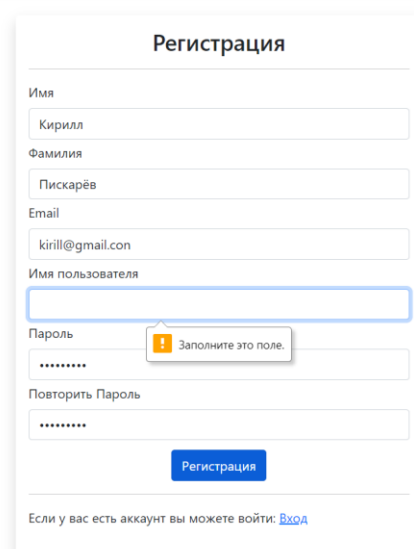
Повторить Пароль

[Регистрация](#)

Если у вас есть аккаунт вы можете войти: [Вход](#)

Рисунок 5.2 – Форма регистрации веб-приложения

Если пользователь заполнил не все обязательные поля в форме регистрации и нажал на кнопку «Регистрация», то увидит сообщение о том, что требуется заполнить пропущенное поле. Пример заполнения формы и с указанием пропущенного поля представлен на рисунке 5.3.



Регистрация

Имя

Фамилия

Email

Имя пользователя

Пароль

Повторить Пароль

[Регистрация](#)

Если у вас есть аккаунт вы можете войти: [Вход](#)

Заполните это поле.

Рисунок 5.3 – Форма регистрации веб-приложения с указанием ошибки

5.2 Авторизация пользователя

Для того чтобы пользователь авторизовался, требуется наличие учетной записи в системе. Если пользователь имеет учетную запись требуется в панели навигации нажать на элемент управления «Вход». Форма входа представлена на рисунке 5.4.

Вход

Имя Пользователя

Пароль

Вход

Если у вас нет аккаунта вы можете зарегистрироваться:
[Регистрация](#)

Рисунок 5.4 – Форма входа в веб-приложения

На форме присутствует проверка ввода данных. Если пользователь не введет данные в одно из полей, появится уведомление, что требуется ввести данные. Пример проверки ввода данных представлен на рисунке 5.5.

Вход

Имя Пользователя

admin

Пароль

Заполните это поле.

Если у вас нет аккаунта вы можете зарегистрироваться:
[Регистрация](#)

Рисунок 5.5 – Форма входа веб-приложения с указанием ошибки

На форму входа в приложении можно попасть с формы регистрации веб-приложения. Для этого при переходе на форму регистрации требуется внизу формы нажать на ссылку «Вход». Форма с ссылкой представлена на рисунке 5.2.

После успешной авторизации в веб-приложении, пользователя перенаправляет на страницу со списком заметок. После входа навигационная

панель меняет свои элементы. Главная страница с измененной навигационной панелью для авторизованного пользователя представлена на рисунке 5.6.

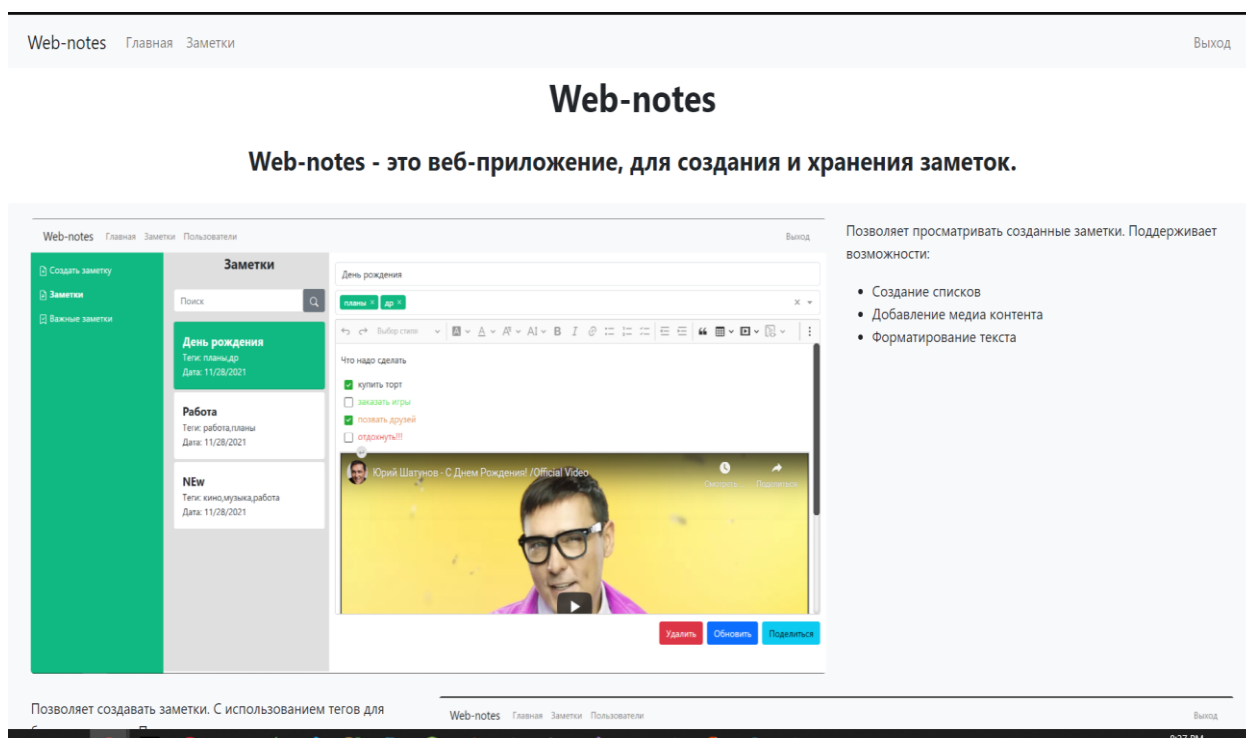


Рисунок 5.6 – Главная страница веб-приложения для авторизованного пользователя

Для выхода из учетной записи требуется нажать на элемент управления «Выход». После нажатия на элемент управления, закончится авторизационная сессия и пользователя перенаправит на главную страницу.

5.3 Создание и редактирование заметки

После авторизации в системе пользователю будут доступны возможности для управления заметками. Для создания заметки требуется в навигационной панели нажать на элемент управления «Заметки», после чего откроется страница для управления заметками. На странице управления будут доступны возможности: создания, редактирования, просмотра и поиска заметок.

Для создания заметки в левой части страницы в навигационной панели требуется нажать на элемент «Создать заметку». Форма создания заметки представлена на рисунке 5.7.

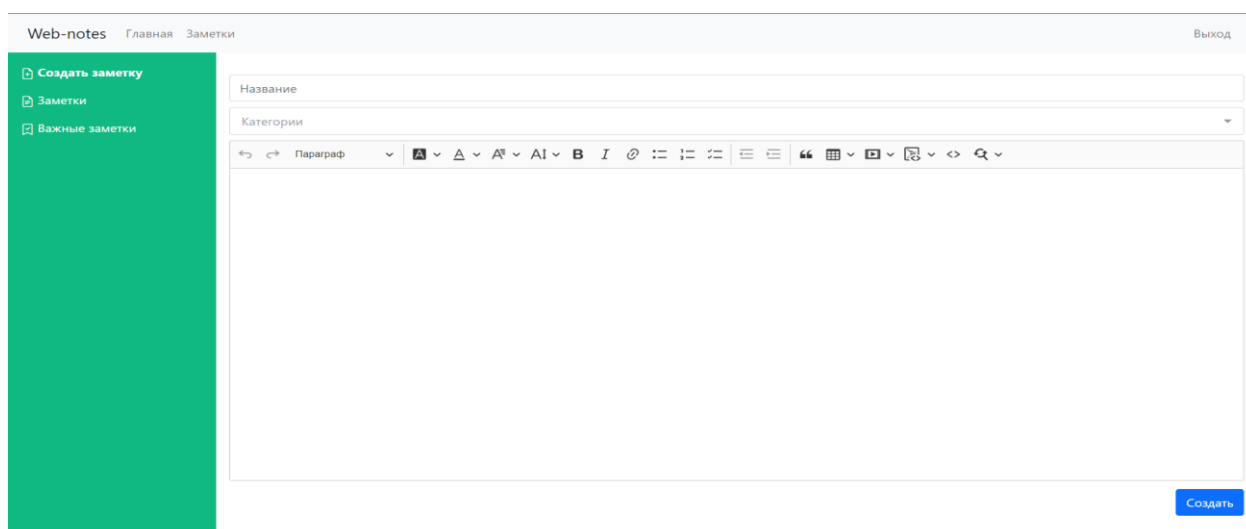


Рисунок 5.7 – Форма создания заметки

На форме расположены следующие элементы страницы: название, категории и элемент для формирования заметки. Название служит для обозначения краткого наименования заметки, для более удобного поиска. Элемент категории представляет собой поле для ввода нескольких категорий заметки, используется также для поиска и краткой характеристики типа заметки. Элемент для создания заметки представляет собой многофункциональный редактор для формирования различного типа и содержания заметки.

Категории включают в себя преопределенный базовый список категорий, который можно выбрать из выпадающего списка при нажатии на элемент категории. Также можно создать новую категорию, написав ее и нажав на кнопку «Ввода». Пример заполнения и списка категорий представлен на рисунке 5.8.

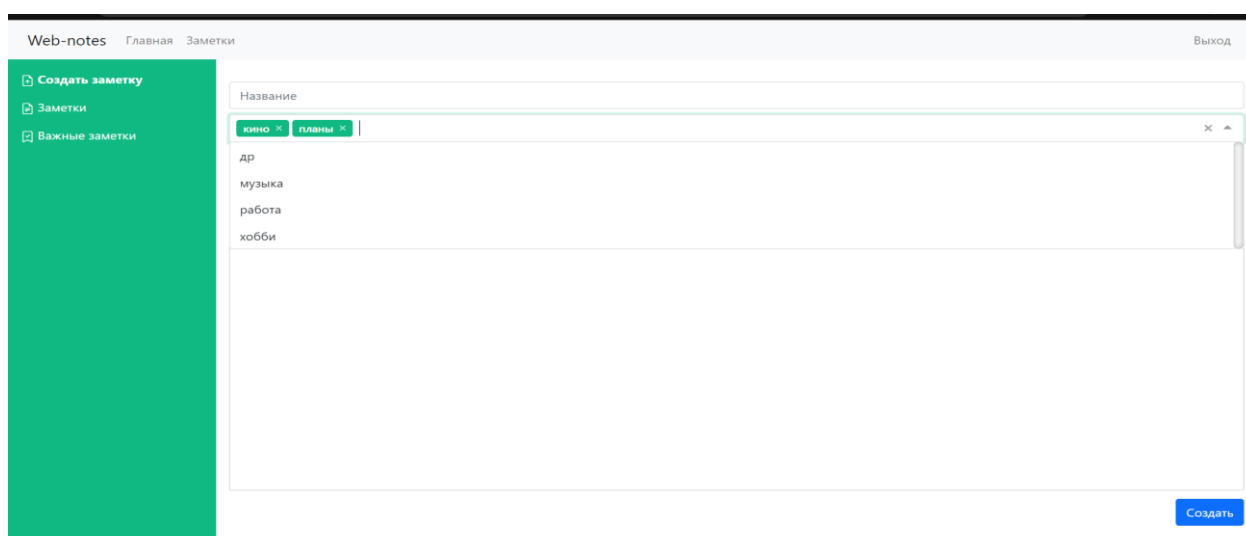


Рисунок 5.8 – Указание категорий в заметке

Элемент для формирования заметки имеет следующие возможности:

- написание текста и его форматирование: изменения стиля, изменения цвета фона, цвета текста, семейство шрифта, жирность и курсив;
- создания ссылок;
- формирования списка;
- создания листа задания;
- выравнивание текста;
- цитирования;
- создание таблиц;
- добавление медиа ссылок;
- формирования блока кода;
- поиск по тексту и замена.

Пункт меню для изменения стиля текста представлен на рисунке 5.9. изменение стиля позволяет выбрать тип заголовка или использовать обычный текст.

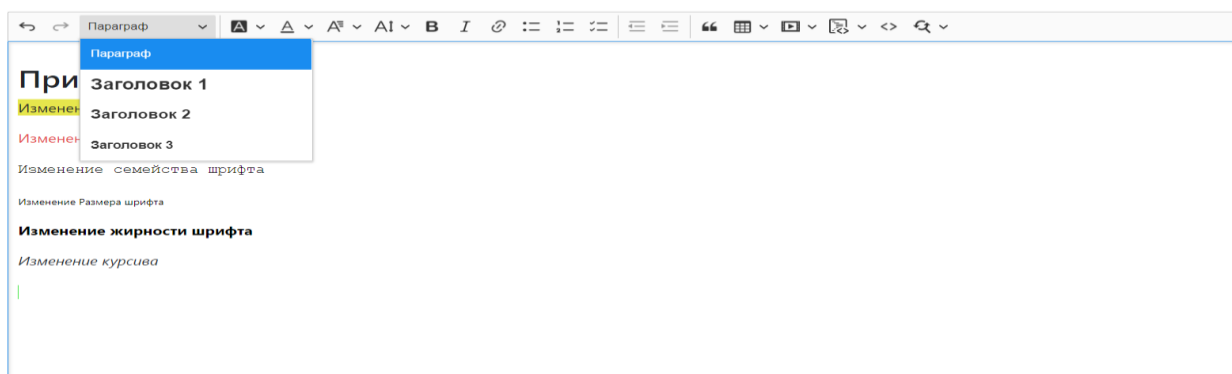


Рисунок 5.9 – Меню для изменения стиля сообщения

Пункт меню для изменения цвета фона представлен на рисунке 5.10. Позволяет задать цвет фона выделенного текста.

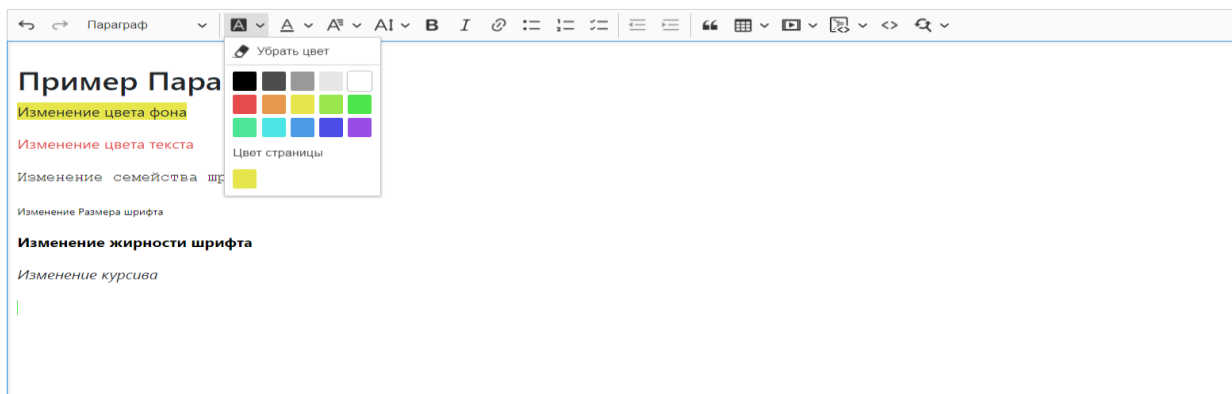


Рисунок 5.10 – Меню для изменения цвета фона

Для изменения цвета текста используется такого же типа пункт меню. Пункт меню для изменения семейства шрифта представлен на рисунке 5.11.

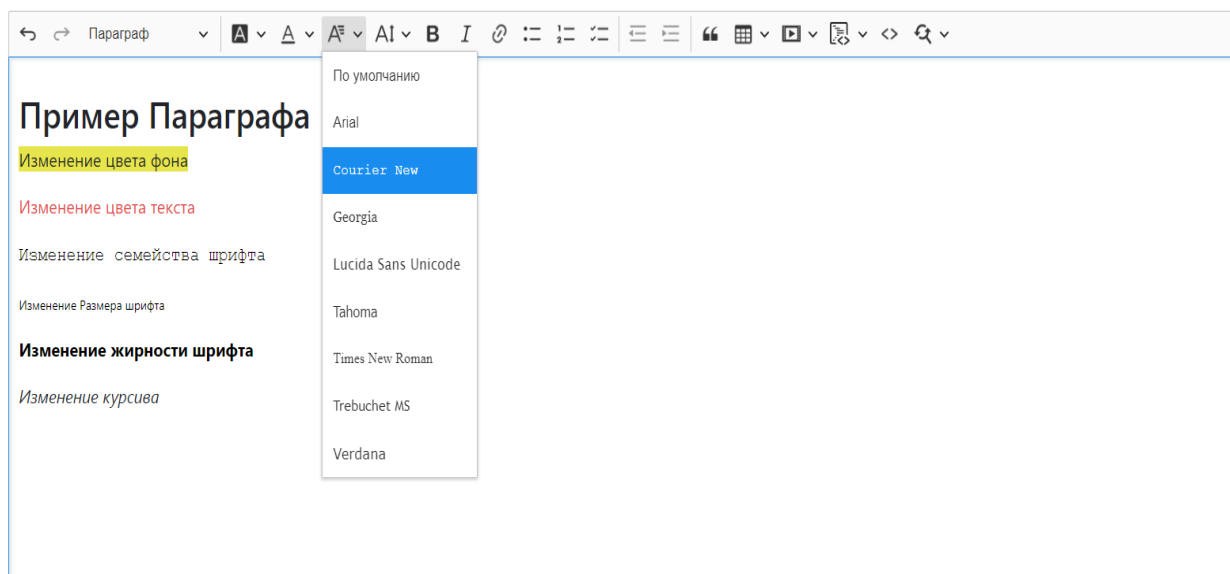


Рисунок 5.11 – Меню для изменения семейства шрифта

Пункт меню для изменения размера шрифта представлен на рисунке 5.12.

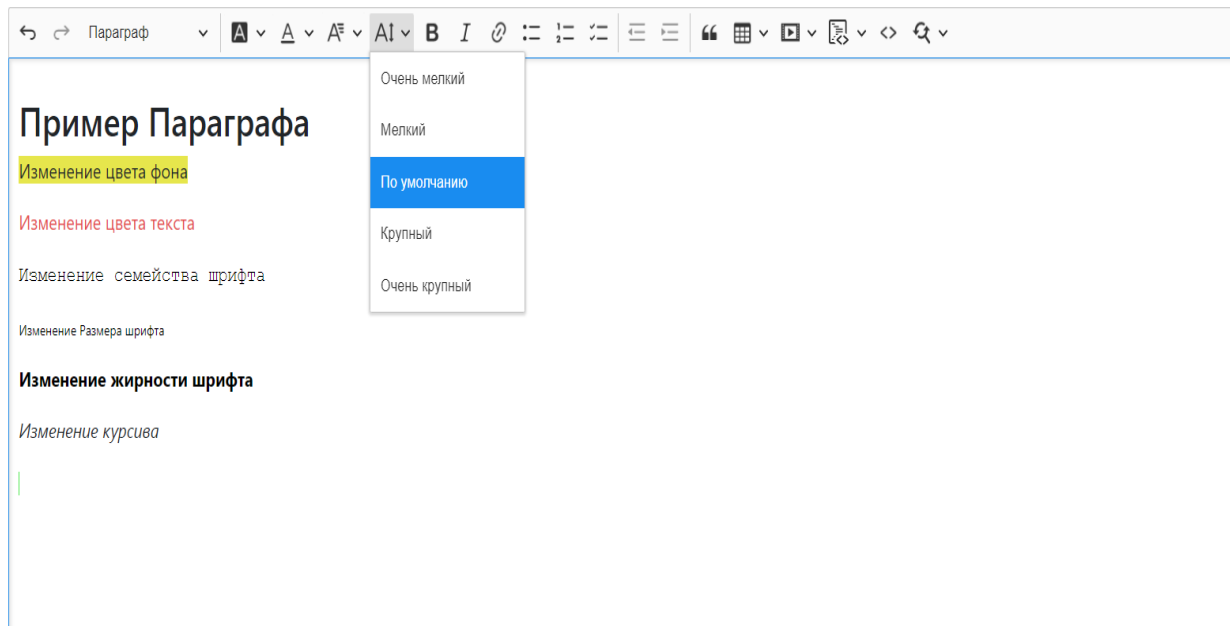


Рисунок 5.12 – Меню для изменения размера шрифта

Пример с использованием перечисленных возможностей форматирования текста представлен на рисунке 5.13.

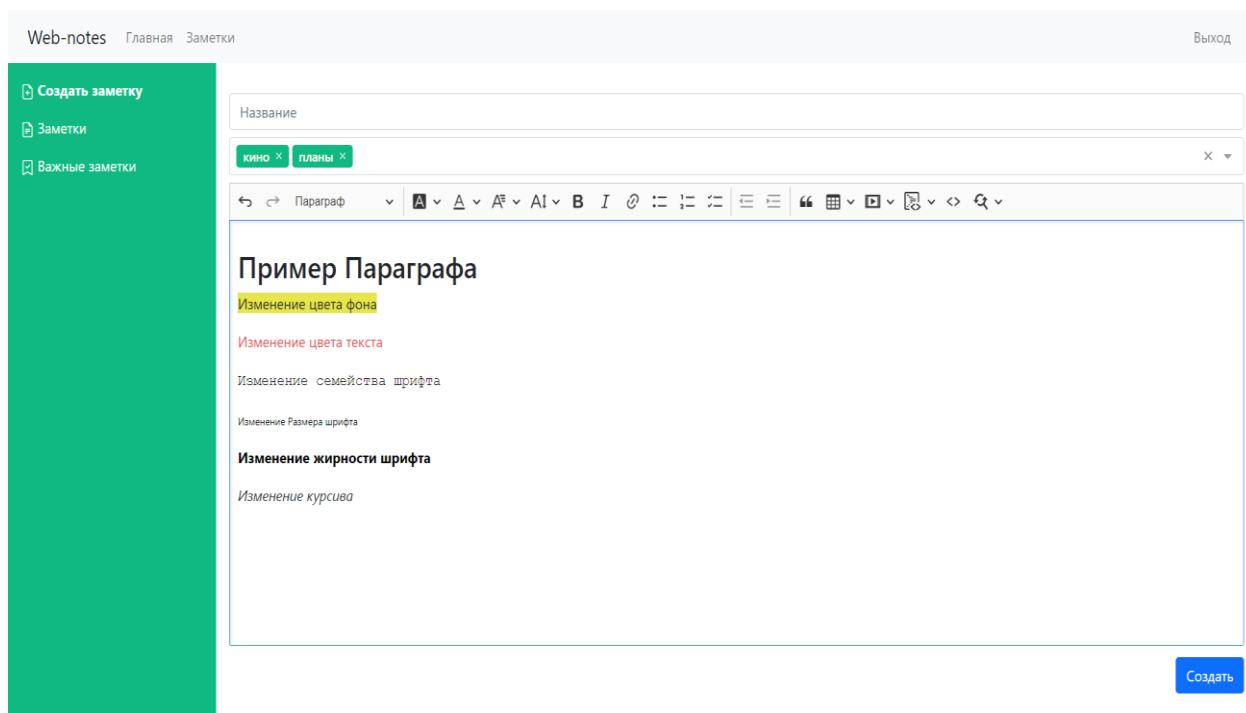


Рисунок 5.13 – Пример форматирования текста

Для создания ссылок есть пункт меню «Ссылка». Результат формирования и пример формирования ссылки представлен на рисунке 5.14.

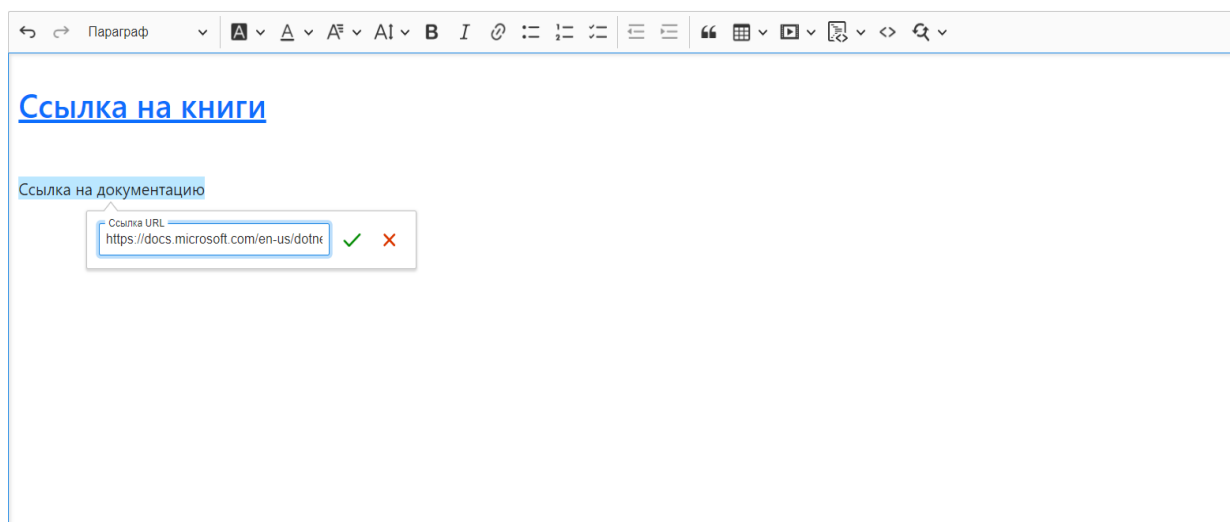


Рисунок 5.14 – Формирование ссылки

Элемент позволяет создавать списки трех видов:

- маркированный список;
- нумерованный список;
- список задач.

Пример с использованием всех типов списка представлен на рисунке 5.15.

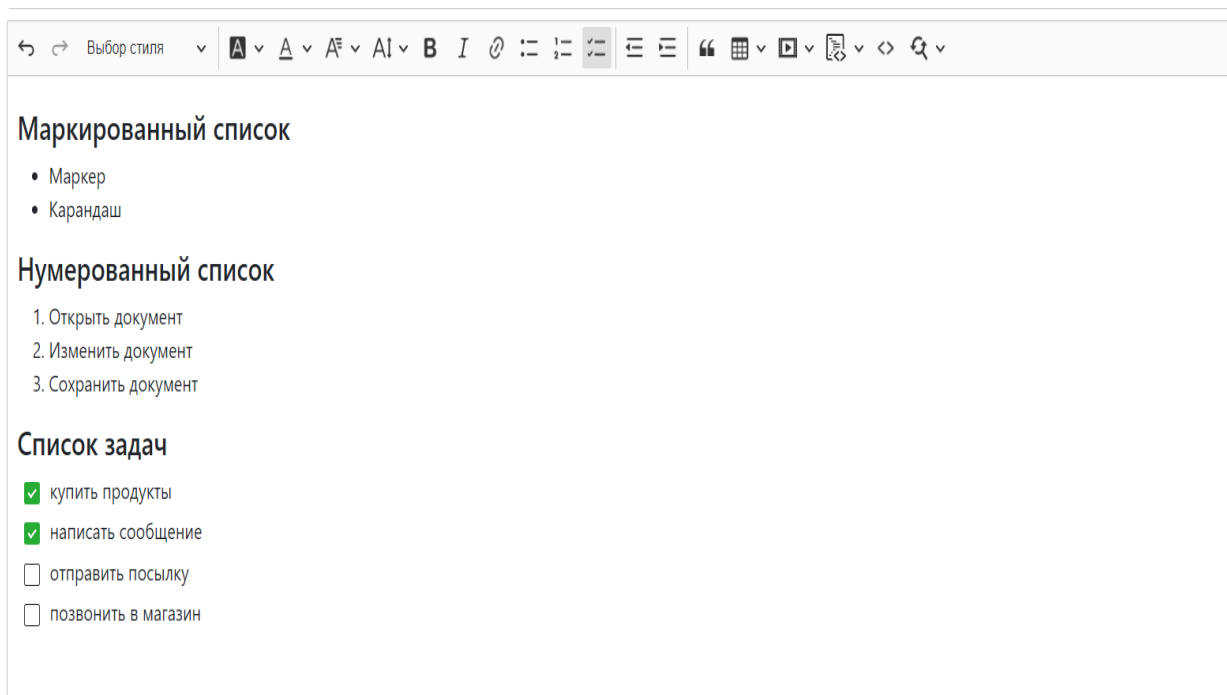


Рисунок 5.15 – Создание списков

Также элемент для формирования заметки позволяет делать цитаты. Для этого есть пункт меню «Цитаты». Пример текста с использованием цитат представлен на рисунке 5.16.

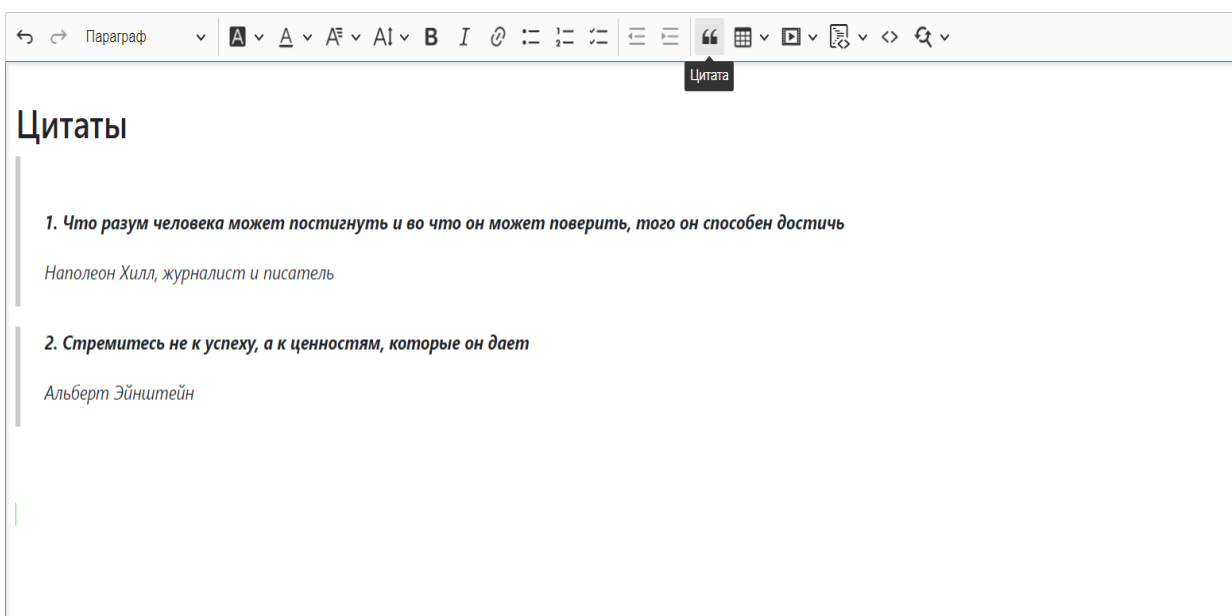


Рисунок 5.16 – Пример формирования цитат

Для добавления медиа ссылок требуется выбрать пункт меню «Вставить медиа» и вставить ссылку на видео. После чего в заметку добавиться превью к видео. Пример использования представлен на рисунке 5.17.

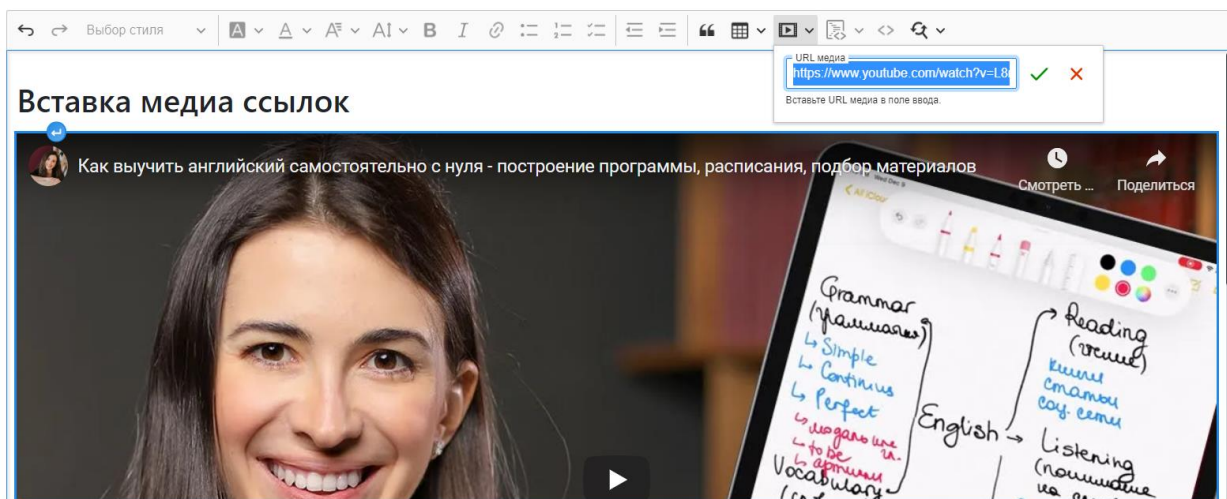


Рисунок 5.18 – Меню для вставки медиа ссылок

Для пользователей, которые хотят вставить в заметку код, есть пункт меню для вставки кода с выбором языка программирования. Пример вставки кода представлен на рисунке 5.19.

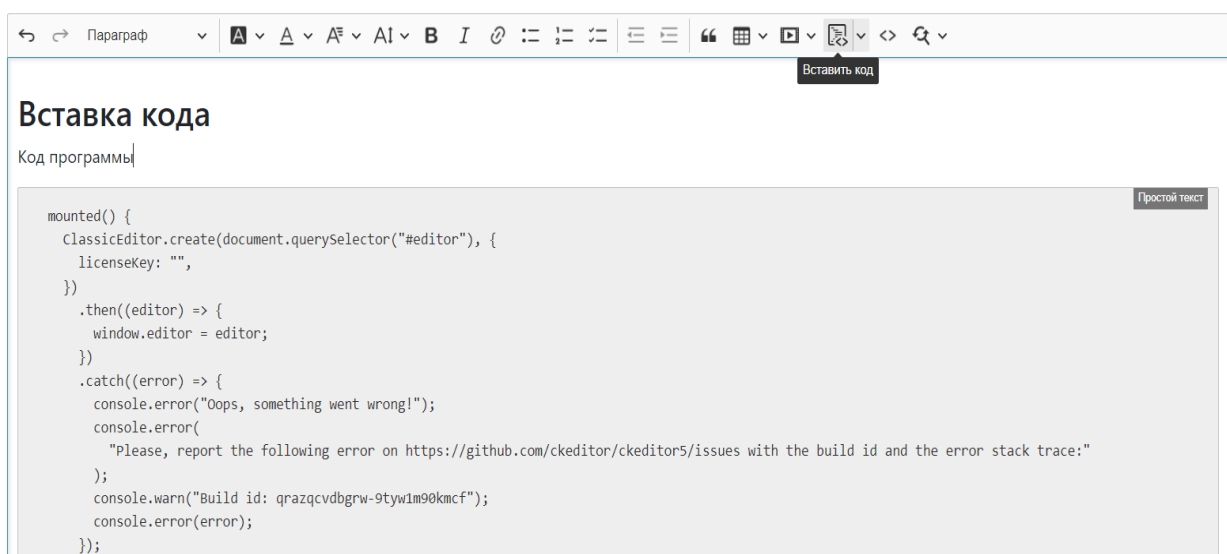


Рисунок 5.19 – Пример вставки кода

Пункт меню поиска позволяет найти фрагмент текста и заменить. Пример поиска представлен на рисунке 5.20.

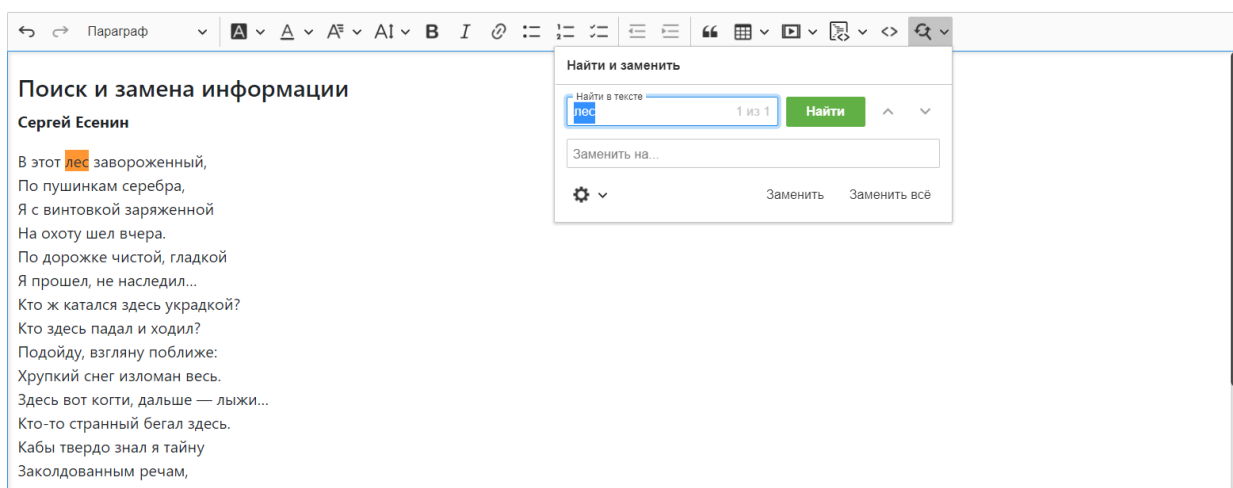


Рисунок 5.20 – Пример использования функций поиска по тексту

После заполнения заметки требуется нажать на кнопку «Создать». Появятся кнопки для обновления и удаления заметки. Результат создания заметки представлен на рисунке 5.21.

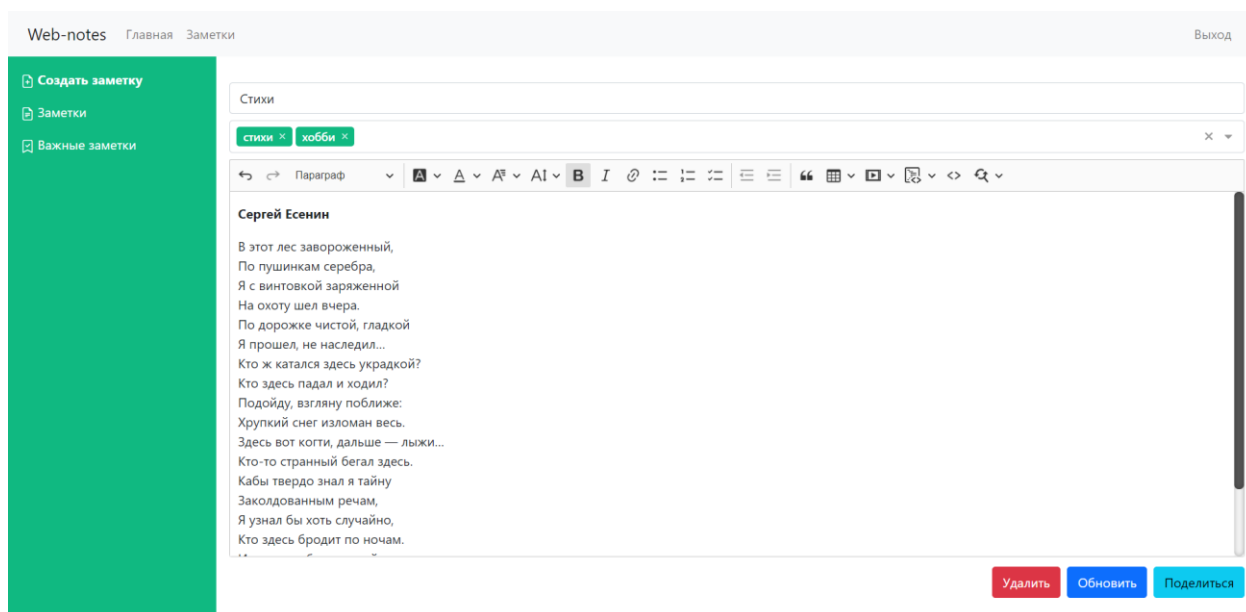


Рисунок 5.21 – Результат создания заметки

5.4 Поиск заметок

Для поиска заметки требуется в левой части навигации выбрать пункт «Заметки», после нажатия откроется форма со списком заметок и возможностью поиска по заметкам. На рисунке 5.22 представлена страница со списком заметок.

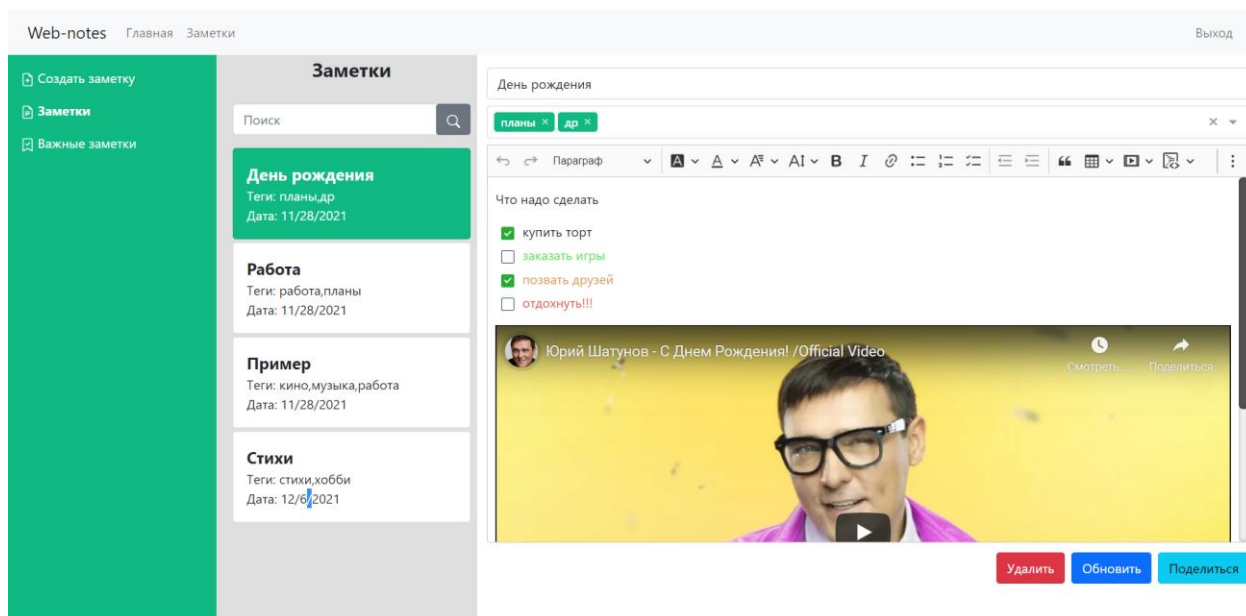


Рисунок 5.22 – Страница поиска заметок

Поиск по заметкам позволяет искать по полям: название, категория, текст заметки. Пример поиска по категории представлен на рисунке 5.23.

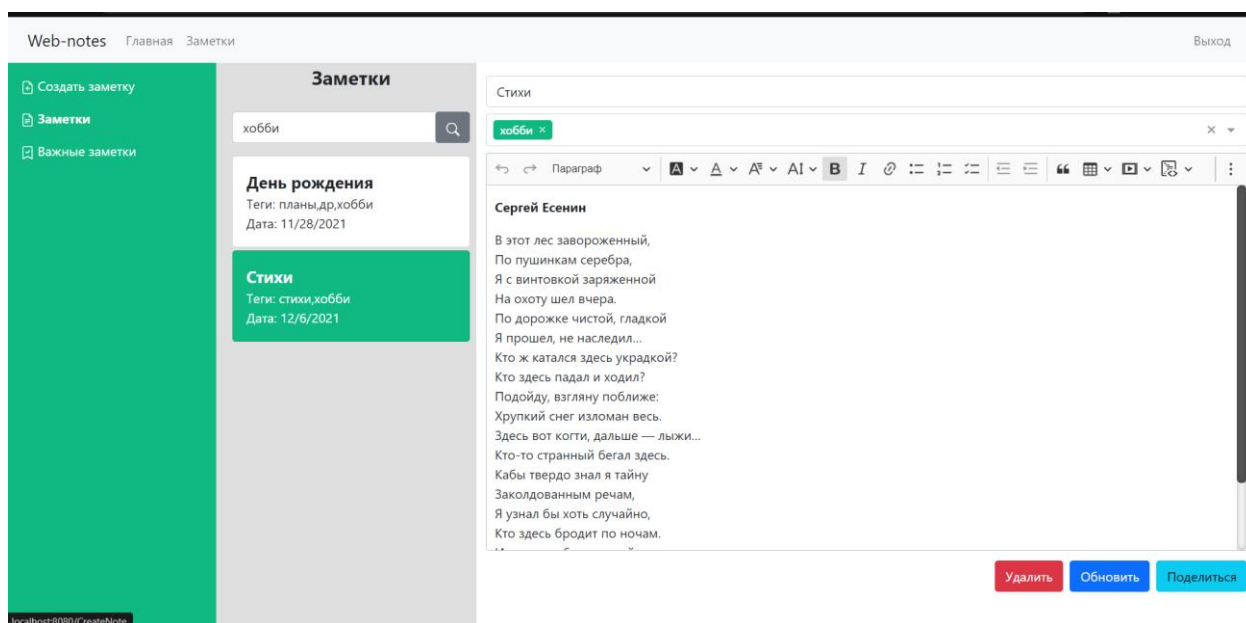


Рисунок 5.23 – Поиск по заметкам с использованием критерия категории

5.5 Поделиться заметкой

После создания заметки на форме появляется кнопка для возможности поделиться заметкой с неавторизованным пользователем. Для этого требуется нажать на кнопку «Поделиться» и скопировать ссылку. После того как пользователь для просмотра содержимого заметки откроет страницу по

ссылке, ему будет доступна заметка в режиме чтения. Страница для просмотра заметки представлена на рисунке 5.24.

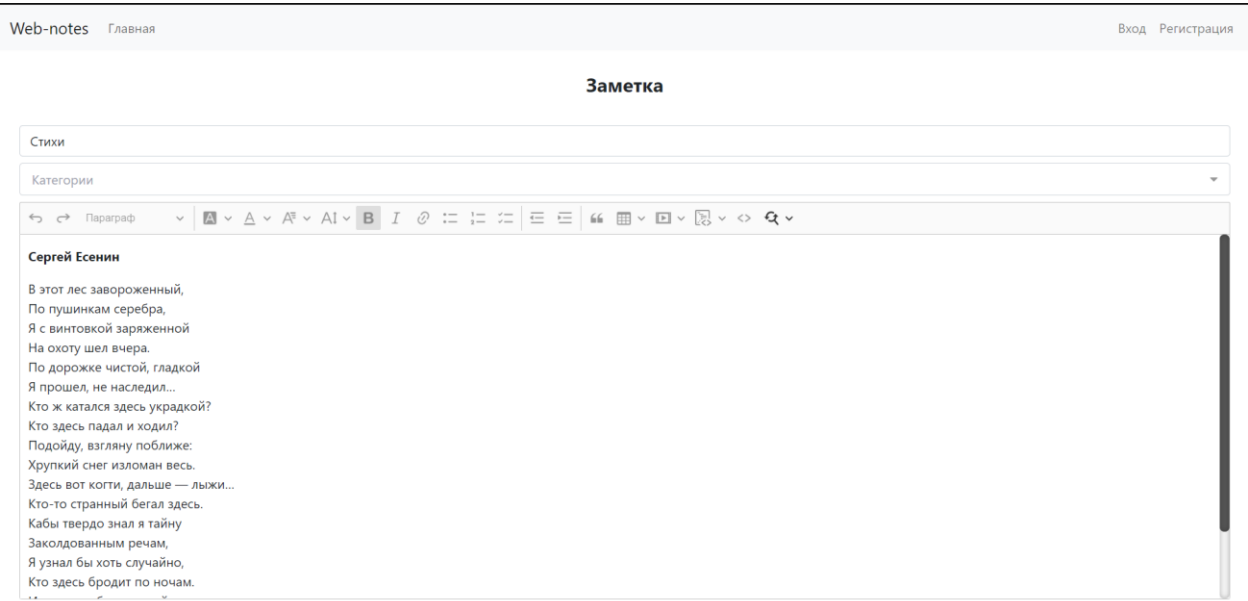


Рисунок 5.24 – Заметка в режиме чтения

5.6 Администратор

В веб-приложении существует пользователь с ролью администратор, ему доступны все вышеперечисленные возможности системы, а также возможность просматривать список пользователей, удалять пользователей и сбрасывать пароль по заявке пользователя. Страница со списком пользователей представлена на рисунке 5.25.

Web-notes Главная Заметки Пользователи Выход

Пользователи

Имя	Имя учетной записи	Email	Роль	
Admin User	admin		Admin	Удалить Сбросить пароль
Normal User	user		User	Удалить Сбросить пароль

Рисунок 5.25 – Страница со списком пользователей

6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ РАЗРАБОТКИ И ПРОГРАММНОГО СРЕДСТВА

ОБОСНОВАНИЕ ИСПОЛЬЗОВАНИЯ

6.1 Описание функций, назначения и потенциальных пользователей программного средства

Целью данного дипломного проекта является создание веб-приложение для управления и хранения пользовательских заметок.

Разрабатываемое веб-приложение предназначено для применения широким кругом пользователей. Данное веб-приложение позволяет создавать и редактировать заметки с использованием браузера, это позволяет иметь доступ к собственным заметкам везде и с любого устройства. Это дает большую гибкость для хранения важной информации. Приложение автоматизирует некоторые процессы работы с заметками, позволяет искать информацию и относить к различной категории.

Программное средство разрабатывается организацией для свободной реализации на рынке IT [12].

6.2 Расчет затрат на разработку программного средства

Затраты на разработку программного средства включают в себя следующие статьи:

- затраты на основную заработную плату разработчиков;
- затраты на дополнительную заработную плату разработчиков;
- отчисления на социальные нужды;
- прочие затраты (амортизационные отчисления, расходы на электроэнергию, командировочные расходы, арендная плата за офисные помещения и оборудование, расходы на управление и реализацию и т.п.).

6.2.1 Расчет затрат на основную заработную плату команды разработки

Затраты на основную заработную плату определяются составом команды, которая занимается разработкой программного средства, месячным окладом специалистов и трудоемкостью процесса разработки и рассчитываются по формуле:

$$Z_o = K_{пр} * \sum_{i=1}^n T_{чi} * t_i \quad (6.1)$$

где n – количество исполнителей, занятых разработкой конкретного ПО;

$K_{пр}$ – коэффициент премий;

$T_{чi}$ – часовая заработная плата i -го исполнителя, руб.;

t_i – трудоемкость работ, выполняемых i -м исполнителем, ч.

В разработке веб-приложения будет участвовать три исполнителя.

Расчет затрат на основную заработную плату разработчика представлено в таблице 6.1. Данные по заработной плате команды разработки предоставлены организацией на 01.11.2021.

Таблица 6.1 – Расчет затрат на основную заработную плату команды разработки

№	Участник команды	Вид выполняемой работы	Месячная заработная плата, руб	Часовая заработная плата, руб	Трудовое количество работ, ч	Зарплата по тарифу, руб
1	2	3	4	5	6	7
1	Инженер-программист	Разработка ПО	1596	9.50	270	2565.00
2	Бизнес-аналитик	Анализ требований	1100	6.55	60	392.86
3	Тестировщик	Тестирование программного средства	900	5.36	70	375.00
Премия (75% от основной заработной платы)						2499.64
Итого затраты на основную заработную плату						5832.50

6.2.2 Расчет затрат на дополнительную заработную плату

Затраты на дополнительную заработную плату разработчика включает выплаты, предусмотренные законодательством о труде (оплата трудовых отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей), и определяется по формуле:

$$З_{д} = \frac{З_{о} * Н_{д}}{100} \quad (6.2)$$

где $З_{о}$ – затраты на основную заработную плату, руб.;

$Н_{д}$ – норматив дополнительной заработной платы (17%);

Затраты на дополнительную заработную плату составит:

$$З_{д} = \frac{5832.50 * 17\%}{100} = 991.53 \text{ руб}$$

6.2.3 Расчет отчислений на социальные нужды

Отчисления на социальные нужды (в фонд социальной защиты населения и на обязательное страхование) определяются в соответствии с действующими законодательными актами по формуле:

$$P_{\text{соц}} = \frac{(З_о + З_д) * H_{\text{соц}}}{100} \quad (6.3)$$

где $H_{\text{соц}}$ – норматив отчислений на социальные нужды (34.6%);

Отчисления на социальные нужду составит:

$$P_{\text{соц}} = \frac{(5832.50 + 991.53) * 34.6}{100} = 2361.11 \text{ руб}$$

6.2.4 Расчет прочих затрат

Прочие затраты включают затраты, связанные с разработкой конкретного программного средства напрямую, а также связанные с функционированием организации-разработчика в целом. Расчет прочих затрат выполняется в процентах от затрат на основную заработную плату команды разработчиков с учетом премии по формуле:

$$З_{\text{пз}} = \frac{З_о * H_{\text{пз}}}{100} \quad (6.4)$$

где $H_{\text{пз}}$ – норматив прочих затрат, 135%;

Рассчитаем сумму прочих затрат:

$$З_{\text{пз}} = \frac{5832.50 * 135}{100} = 7873.88 \text{ руб}$$

Полная сумма затрат на разработку программного средства находится путем суммирования всех рассчитанных статей затрат. Расчет приведен в таблице 6.2.

Таблица 6.2 – Затраты на разработку программного средства

Статья затрат	Сумма, руб
Основная заработная плата разработчика	5832.50
Дополнительная заработная плата разработчика	991.53
Отчисления на социальные нужды	2361.11
Прочие затраты	7873.88
Общая сумма затрат на разработку	17059.01

Общая сумма затрат на разработку составила 17059.01 руб.

6.3 Оценка экономического эффекта при разработке программного средства для свободной реализации на ИТ–рынке

6.3.1 Экономический эффект у разработчика

Экономический эффект организации-разработчика программного средства в данном случае представляет собой прибыли от его продажи множеству потребителей (подписок). Прибыль от реализации в данном случае напрямую зависит от объемов продаж, цены реализации и затрат на разработку данного ПО. Цена формируется на рынке под воздействием спроса и предложения. Прибыль, полученная разработчиком от реализации ПО на рынке, осуществляется по формуле:

$$П = Ц * N - НДС - З_p \quad (7.5)$$

где Ц – цена реализации ПО заказчику, руб;
N – количество копий (лицензий) ПО, которое будет куплено клиентами;
НДС – сумма налога на добавленную стоимость, руб;
З_р – сумма расходов на разработку и реализацию ПО, руб.

Проанализировав схожие программные решения на рынке, было принято решение взять среднюю стоимость в размере 25 руб., количество реализуемых в год подписок 1450 в 2022, 1000 в 2023, 600 в 2024. Общее количество реализуемых подписок равно 3050. Расчеты будем производить за первый год реализации подписок.

Налог на добавленную стоимость определяется по формуле:

$$НДС = \frac{Ц * N * Н_{дс}}{100\% + Н_{дс}} \quad (6.6)$$

где Н_{дс} – ставка налога на добавленную стоимость, (20%);

Рассчитаем налог на добавленную стоимость:

$$НДС = \frac{25 * 1450 * 20}{100 + 20} = 6041.67 \text{ руб}$$

Рассчитаем прибыль, полученная разработчиком от реализации ПО на рынке:

$$П = 25 * 1450 - 6041.67 - 17059.01 = 13149.32 \text{ руб}$$

Далее рассчитываем рентабельности затрат на разработку программного средства по следующей формуле:

$$P = \frac{\Pi}{З_p} * 100\% \quad (6.7)$$

$$P = \frac{13149.32}{17059.01} * 100\% = 77.08 \%$$

Таким образом, при реализации 1450 подписок программного средства за первый год по цене 25 руб, организация получит экономический эффект в размере 13149.32 руб. Рентабельность затрат на разработку программного средства равна 77.08 %.

ЗАКЛЮЧЕНИЕ

В ходе дипломного проектирования проведен анализ литературы по теме дипломного проекта, изучены наиболее известные программные решения для создания и управления заметок. В процессе работы были исследованы основные направления по формированию заметок.

Для каждого программного средства были выделены его достоинства и недостатки использования, которые учитывались при формировании требований к программному средству.

В ходе моделирования программного средства были разработаны функциональные требования и описана функциональность к программному средству. На основе требований была разработана информационная модель и схема вариантов использования.

Исходя из полученных на этапе моделирования данных, была спроектирована архитектура программного решения и разработаны структуры классов и спроектированы алгоритмы работы программного средства. На основании информационной модели была спроектирована модель базы данных.

После разработки программного средства было проведено тестирование и исправление недочетов программы.

Таким образом, задачи, поставленные в рамках индивидуального задания, были выполнены. Знания и опыт полученные в процессе прохождения дипломного проектирования будут полезны при дальнейшей работе по специальности

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Интерактивная доска [Электронный ресурс]. – Электронные данные. – Режим доступа: https://hrwiki.ru/wiki/Interactive_whiteboard – Дата доступа: 01.11.2021
- [2] Интеллект карты [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://externat.foxford.ru/polezno-znat/kak-ispolzovat> – Дата доступа: 01.11.2021
- [3] Основные различия между веб-сайтом и веб-приложением [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://artismedia.by/blog/osnovnye-razlichiya-mezhdu-veb-sajtom> – Дата доступа: 01.11.2021
- [4] Google Keep – что это такое и как им пользоваться? [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://impression.ua/google-keep-chto-eto-takoe-i-kak-im-polzovatsya> – Дата доступа: 05.11.2021
- [5] Simplenote [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://simplenote.com> – Дата доступа: 10.11.2021
- [6] Notion. One workspace. Every team. [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.notion.so> – Дата доступа: 11.11.2021
- [7] Моделирование на UML. Общие диаграммы [Электронный ресурс]. – Электронные данные. – Режим доступа: http://book.uml3.ru/sec_1_5 – Дата доступа: 17.11.2021
- [8] Проектирование реляционной базы данных [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://itteach.ru/bazi-dannich/proektirovanie-relyatsionnoy-bazi-dannich> – Дата доступа: 20.11.2021
- [9] Одностраничные (spa) и многостраничные (pwa) веб-приложения [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://vc.ru/seo/odnostranichnye-spa-i-mnogostranichnye-pwa-veb-prilozheniya> – Дата доступа: 21.11.2021
- [10] Введение в ASP.NET Core [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.microsoft.com/ru-ru/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0> – Дата доступа: 23.11.2021
- [11] Бураков, П.В. Введение в системы баз данных: учебное пособие / П.В. Бураков, В.Ю. Петров – СПб: СПбГУ ИТМО, 2010. – 128 с.
- [12] Горовой, В. Г. Экономическое обоснование проекта по разработке программного обеспечения : метод. пособие / В. Г. Горовой, А. В. Грицай, В.А. Пархименко. – Минск : БГУИР, 2018. – 12 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Текст программы

```
import { createApp } from "vue";
import App from "../App.vue";
import router from "../router";
import store from "../store";
import "bootstrap/dist/css/bootstrap.min.css";
import "@vueform/multiselect/themes/default.css";
import "bootstrap";
createApp(App).use(store).use(router).mount("#app");

<template>
  <template v-if="!isAuthorized">
    <NavBar />

    <router-view />
  </template>
  <template v-else>
    <div class="container-fluid p-0 m-0 min-vh-100 d-flex flex-
column">
      <div class="row p-0 m-0">
        <div class="col p-0 m-0">
          <NavBar />
        </div>
      </div>
      <div class="row flex-grow-1 p-0 m-0">
        <router-view />
      </div>
    </div>
  </template>
</template>

<script>
import { mapGetters } from "vuex";
import NavBar from "@components/NavBar";

export default {
  name: "App",
  components: {
    NavBar,
  },
  created() {
    this.$store.dispatch("Auth/checkAuth");
  },
  computed: {
    ...mapGetters("Auth", ["isAuthorized"]),
  },
};
```

```

</script>

<style>
html,
body,
#app {
  height: 100%;
  overflow: hidden;
}

::-webkit-scrollbar-track {
  -webkit-box-shadow: inset 0 0 6px rgba(0, 0, 0, 0.3);
  box-shadow: inset 0 0 6px rgba(0, 0, 0, 0.3);
  border-radius: 10px;
  background-color: #f5f5f5;
}

::-webkit-scrollbar {
  width: 12px;
  background-color: #f5f5f5;
}

::-webkit-scrollbar-thumb {
  border-radius: 10px;
  -webkit-box-shadow: inset 0 0 6px rgba(0, 0, 0, 0.3);
  box-shadow: inset 0 0 6px rgba(0, 0, 0, 0.3);
  background-color: #555;
}

.sidebar {
  background-color: #10b981;
  color: white;
}
</style>
<template>
  <div class="users-container container-fluid">
    <p class="fs-1 fw-bold text-center">Пользователи</p>
    <table class="table">
      <thead>
        <tr>
          <th scope="col">Имя</th>
          <th scope="col">Имя учетной записи</th>
          <th scope="col">Email</th>
          <th scope="col">Роль</th>
          <th scope="col"></th>
        </tr>
      </thead>
      <tbody>
        <UserCard
          v-for="user in users"
          :key="user.id"
          :user="user"
          @delete="onDelete"

```

```

        @reset="onReset"
      />
    </tbody>
  </table>
</div>
</template>

<script>
import UserCard from "@/components/UserCard.vue";
import { getUsers } from "../providers/userService.js";
export default {
  name: "Users",
  components: {
    UserCard,
  },
  data() {
    return {
      users: [],
    };
  },
  created() {
    getUsers(this.$store.state.Auth.user).then(({ data }) => {
      this.users = data;
    });
  },
  methods: {
    onDelete(id) {},
    onReset(id) {},
  },
};
</script>
<template>
  <div class="p-4">
    <p class="text-center fw-bold fs-4">Заметка</p>
    <div class="form-group pt-3 pb-2">
      <input
        class="form-control"
        type="text"
        id="title"
        v-model="note.title"
        placeholder="Название"
      />
    </div>
    <div class="form-group pb-2">
      <Multiselect
        id="noteTags"
        v-model="note.noteTags"
        mode="tags"
        placeholder="Категории"
        :closeOnSelect="false"
        :searchable="true"

```

```

        :createTag="true"
        :options="[
          { value: 'кино', label: 'кино' },
          { value: 'музыка', label: 'музыка' },
          { value: 'работа', label: 'работа' },
        ]"
      />
    </div>
    <div class="form-group pb-1">
      <div id="editor">{{ note.noteDocument }}</div>
    </div>
  </div>
</template>

<script>
import { getNoteShared } from "../providers/noteService.js";
import Multiselect from "@vueform/multiselect";
export default {
  name: "SharedNote",
  components: {
    Multiselect,
  },
  data() {
    return {
      isNew: true,
      note: {},
    };
  },
  mounted() {
    ClassicEditor.create(document.querySelector("#editor"), {
      licenseKey: "",
    })
      .then((editor) => {
        window.editor = editor;
        editor.isReadOnly = true;
        debugger;
        getNoteShared(this.$route.query.id,
this.$store.state.Auth.user).then(
          ({ data }) => {
            this.note = { ...data };
            window.editor.setData(this.note.noteDocument);
          }
        );
      })
      .catch((error) => {
        console.error("Oops, something went wrong!");
        console.error(
          "Please, report the following error on
https://github.com/ckeditor/ckeditor5/issues with the build id
and the error stack trace:"
        );
        console.warn("Build id: grazqcvdbgrw-9tywlm90kmcf");
      });
  },
};

```

```

        console.error(error);
    });
},
};
</script>
<template>
    <div>
        <div class="div-center shadow p-3 bg-body rounded">
            <div class="content">
                <h3 class="text-center">Регистрация</h3>
                <hr />

                <form @submit.prevent="register">
                    <div class="form-group pb-1">
                        <label for="userName" class="form-label">Имя</label>
                        <input
                            class="form-control"
                            type="text"
                            id="userName"
                            v-model="registerModel.firstName"
                            required
                        />
                    </div>
                    <div class="form-group pb-1">
                        <label for="userName" class="form-
label">Фамилия</label>
                        <input
                            class="form-control"
                            type="text"
                            id="userName"
                            v-model="registerModel.lastName"
                            required
                        />
                    </div>
                    <div class="form-group pb-1">
                        <label for="userName" class="form-
label">Email</label>
                        <input
                            class="form-control"
                            type="email"
                            id="userName"
                            v-model="registerModel.email"
                            required
                        />
                    </div>
                    <div class="form-group pb-1">
                        <label for="userName" class="form-label">Имя
пользователя</label>
                        <input
                            class="form-control"
                            type="text"
                            id="userName"

```

```

        v-model="registerModel.userName"
        required
    />
</div>

<div class="form-group pb-1">
    <label for="password" class="form-label">Пароль</label>
    <input
        class="form-control"
        type="password"
        id="password"
        v-model="registerModel.password"
        required
    />
</div>
<div class="form-group pb-1">
    <label for="comparePassword" class="form-label">
        Повторить Пароль</label>
    <input
        class="form-control"
        type="password"
        id="comparePassword"
        v-model="registerModel.comparePassword"
        required
    />
</div>
<div class="text-center pt-2">
    <button class="btn btn-primary"
type="submit">Регистрация</button>
</div>
<hr />
<p>
    Если у вас есть аккаунт вы можете войти:
    <router-link to="/login">Вход</router-link>
</p>
</form>
</div>
</div>
</div>
</template>

<script>
export default {
    name: "Register",
    data() {
        return {
            registerModel: {
                userName: "",
                comparePassword: "",
                password: "",

```

```

        firstName: "",
        lastName: "",
    },
    };
},
methods: {
    register() {
        this.$store.dispatch("Auth/register", this.registerModel);
    },
},
};
</script>
<template>
    <div class="col-md-2 p-0 m-0 sidebar">
        <SideBar />
    </div>

    <div class="col-md-10 p-0 m-0">
        <div class="row p-0 m-0 h-100">
            <div class="col-md-3 list-container">
                <p class="fw-bold fs-4 text-center">Заметки</p>
                <SearchNotes @search="onSearch" />
                <div class="overflow-auto list-notes">
                    <ListNotes :notes="notes" @openNote="onOpenNote" />
                </div>
            </div>
            <div class="col-md-9">
                <NoteForm
                    :isNew="isNew"
                    :note="note"
                    @update="onUpdate"
                    @delete="onDelete"
                />
            </div>
        </div>
    </div>
</template>

<script>
import NoteForm from "@components/NoteForm.vue";
import ListNotes from "@components/ListNotes.vue";

import SideBar from "@components/SideBar";
import SearchNotes from "@components/SearchNotes.vue";
import { getNotes, deleteNote, updateNote } from
"../providers/noteService.js";

export default {
    name: "Notes",
    components: {
        NoteForm,
        ListNotes,

```

```

        SearchNotes,
        SideBar,
    },
    data() {
        return {
            notes: [],
            note: null,
            isNew: false,
            searchValue: "",
        };
    },
    created() {
        getNotes(this.$store.state.Auth.user,
this.searchValue).then(({ data }) => {
            this.notes = data;
            this.note = data[0];
            this.note.isCurrent = true;
        });
    },
    methods: {
        onOpenNote(id) {
            this.note.isCurrent = false;
            this.note = this.notes.find((x) => {
                return x.id === id;
            });

            this.note.isCurrent = true;
        },

        onSearch(searchValue) {
            this.searchValue = searchValue;
            this.updateListNotes();
        },
        onUpdate(note) {
            updateNote(this.$store.state.Auth.user, note).then(({ data
}) => {
                console.log(data);

                this.updateListNotes().then(() => {
                    this.onOpenNote(note.id);
                });
            });
        },
        onDelete(id) {
            deleteNote(this.$store.state.Auth.user, id).then(() => {
                getNotes(this.$store.state.Auth.user,
this.searchValue).then(
                    ({ data }) => {
                        this.notes = data;
                        this.note.isCurrent = false;
                        this.note = data[0];
                        this.note.isCurrent = true;
                    }
                );
            });
        },
    },

```



```

        }
      );
    });
  },
  updateListNotes() {
    return getNotes(this.$store.state.Auth.user,
this.searchValue).then(
      ({ data }) => {
        this.notes = data;
        return data;
      }
    );
  },
},
};
</script>

<style scoped>
.list-container {
  background-color: rgb(223, 223, 223);
  height: 92vh;
  padding-right: 0px;
}
.list-notes {
  height: 77vh;
}
</style>
<template>
  <div>
    <div class="div-center shadow p-3 bg-body rounded">
      <div class="content">
        <h3 class="text-center">Вход</h3>
        <hr />
        <form @submit.prevent="login">
          <div class="form-group pb-1">
            <label for="userName" class="form-label">Имя
Пользователя</label>
            <input
              class="form-control"
              type="text"
              id="userName"
              v-model="loginModel.userName"
              required
            />
          </div>
          <div class="form-group pb-1">
            <label for="password" class="form-
label">Пароль</label>
            <input
              class="form-control"
              type="password"
              id="password"

```

```

        v-model="loginModel.password"
        required
      />
    </div>
    <div class="pt-2 text-center">
      <button type="submit" class="btn btn-
primary">Вход</button>
    </div>
    <hr />

    <p>
      Если у вас нет аккаунта вы можете
      зарегистрироваться:
      <router-link to="/register">Регистрация</router-
link>
    </p>
  </form>
</div>
</div>
</div>
</template>

<script>
export default {
  name: "Login",
  data() {
    return {
      loginModel: {
        userName: "",
        password: "",
      },
    };
  },
  methods: {
    login() {
      this.$store.dispatch("Auth/login",
this.loginModel).then(() => {
        debugger;
        this.$router.push("Notes");
      });
    },
  },
};
</script>

<style>
.div-center {
  width: 500px;
  height: 400px;
  background-color: #fff;
  position: absolute;
  left: 0;

```

```

    right: 0;
    top: 60px;
    bottom: 0;
    margin: auto;
    max-width: 100%;
    max-height: 100%;
    overflow: auto;
    padding: 1em 2em;
    border-bottom: 2px solid #ccc;
    display: table;
}

div.content {
  display: table-cell;
  vertical-align: middle;
}
</style>
import api from "../providers/api.js";

export const createNote = (user, note) => {
  return api.post(
    "/Notes/CreateNote",
    {
      ...note,
      userId: user.id,
    },
    headers(user)
  );
};

export const updateNote = (user, note) => {
  return api.post(
    "/Notes/UpdateNote",
    {
      ...note,
      userId: user.id,
    },
    headers(user)
  );
};

export const deleteNote = (user, id) => {
  return api.delete(`/Notes/DeleteNote/${id}`, headers(user));
};

export const getNotes = (user, searchValue = "") => {
  return api.get(`/Notes/GetNotes?SearchValue=${searchValue}`,
    headers(user));
};

export const getNote = (id, user) => {
  return api.get(`/Notes/GetNote/${id}`, headers(user));
};

```

```

export const getNoteSharedId = (id, user) => {
  return api.get(`/Notes/GetNoteSharedId/${id}`, headers(user));
};

export const getTags = ( user) => {
  return api.get(`/Notes/GetTags/${user.id}`, headers(user));
};

export const getNoteShared= (id, user) => {
  return api.get(`/Notes/GetNoteShared/${id}`, headers(user));
};

function headers(user) {
  const authHeader = user.token
    ? { Authorization: "Bearer " + user.token }
    : {};
  return {
    headers: {
      ...authHeader,
      "Content-Type": "application/json",
    },
  };
}
import api from "../providers/api.js";

export const getUsers = (user) => {
  return api.get("/Users/GetAll", headers(user));
};

function headers(user) {
  const authHeader = user.token
    ? { Authorization: "Bearer " + user.token }
    : {};
  return {
    headers: {
      ...authHeader,
      "Content-Type": "application/json",
    },
  };
}
import axios from "axios";

const apiProvider = axios.create({
  baseURL: "https://localhost:44325/api/",
});

apiProvider.interceptors.response.use(
  (response) => response,
  (error) => {
    let title = "Error";

```

```

    let errorMessage = error.response.data.message;

    if (errorMessage) {
      alert("Error message: " + errorMessage);
    }

    return Promise.reject(error);
  }
);

export default apiProvider;
import api from "../providers/api.js";

export default {
  namespaced: true,
  state: {
    user: {},
  },
  getters: {
    isAuthorized: (state) => {
      return !!state.user.token;
    },
    role: (state) => {
      return state.user.role;
    },
    isAdmin: (state) => {
      return state.user.role === "Admin";
    }
  },
  mutations: {
    setUser(state, user) {
      state.user = user;
    },
  },
  actions: {
    checkAuth(context) {
      context.commit("setUser",
JSON.parse(localStorage.getItem("auth")) || {});
    },
    login(context, loginData) {
      return api.post("/Users/Authenticate", loginData).then(({
data }) => {
        console.log(data);
        context.commit("setUser", data);
        localStorage.setItem("auth", JSON.stringify(data));
      });
    },
    register(context, registerData) {
      return api.post("/Users/Register", registerData).then(({
data }) => {
        console.log(data);
        context.commit("setUser", data);
      });
    },
  },
};

```

```

        localStorage.setItem("auth", JSON.stringify(data));
    });
},
logout(context) {
    context.commit("setUser", {});
    localStorage.setItem("auth", JSON.stringify({}));
},
},
});
import { createStore } from "vuex";
import AuthStoreModule from "../store/auth.js";

export default createStore({
    modules: {
        Auth: AuthStoreModule
    },
});
<template>
    <div class="h-100 me-2 sidebar pt-2">
        <ul class="nav nav-pills flex-column mb-auto">
            <li class="nav-item">
                <router-link class="nav-link text-white" :class="{ 'fw-
bold': $route.name === 'CreateNote' }" to="/CreateNote"
                ><i class="bi bi-file-earmark-plus"></i> Создать
заметку</router-link
                >
            </li>
            <li class="nav-item">
                <router-link class="nav-link text-white" :class="{ 'fw-
bold': $route.name === 'Notes' }" to="/Notes"
                ><i class="bi bi-file-earmark-text"></i>
Заметки</router-link
                >
            </li>
            <li class="nav-item">
                <router-link class="nav-link text-white" :class="{ 'fw-
bold': $route.name === 'ImportantNotes' }" to="/ImportantNotes"
                ><i class="bi bi-bookmark-check"></i> Важные
заметки</router-link
                >
            </li>
        </ul>
    </div>
</template>
<template>
    <div class="form-group pt-3 pb-2">
        <!-- <label for="title" class="form-label">Название</label>
-->
        <input
            class="form-control"
            type="text"
            id="title"

```

```

        v-model="noteModel.title"
        placeholder="Название"
    />
</div>
<div class="form-group pb-2">
    <Multiselect
        id="noteTags"
        v-model="noteModel.noteTags"
        mode="tags"
        placeholder="Категории"
        :closeOnSelect="false"
        :searchable="true"
        :createTag="true"
        :options="categoryOptions"
    />
</div>
<div class="form-group pb-1">
    <div id="editor">{{ noteModel.noteDocument }}</div>
</div>

<div class="d-md-flex justify-content-md-end py-2">
    <div v-if="isNew">
        <button class="btn btn-primary" v-
on:click="onCreateNote()">
            Создать
        </button>
    </div>
    <div v-else>
        <button class="btn btn-danger me-md-2" v-
on:click="onDeleteNote()">
            Удалить
        </button>
        <button class="btn btn-primary me-md-2" v-
on:click="onUpdateNote()">
            Обновить
        </button>
        <button class="btn btn-info" v-on:click="onShareNote()">
            Поделиться
        </button>
    </div>
</div>
</template>

<script>
import Multiselect from "@vueform/multiselect";
import { getNoteSharedId, getTags } from
"../providers/noteService.js";

export default {
    name: "NoteForm",
    props: {
        isNew: Boolean,

```

```

    note: Object,
  },
  emits: ["create"],
  components: {
    Multiselect,
  },
  data() {
    return {
      categoryOptions: [],
      editor: null,
      noteModel: {
        title: "",
        noteTags: [],
        noteDocument: "",
        options: [],
      },
      options: ["Batman", "Robin", "Joker"],
    };
  },
  created() {
    getTags(this.$store.state.Auth.user).then(({ data }) => {
      this.categoryOptions = data.map((value) => {
        return { value: value, label: value };
      });
    });
  },
  watch: {
    note() {
      if (this.note) {
        this.noteModel = { ...this.note };
        window.editor.setData(this.noteModel.noteDocument);
      }
    },
  },
  mounted() {
    ClassicEditor.create(document.querySelector("#editor"), {
      licenseKey: "",
    })
      .then((editor) => {
        window.editor = editor;
      })
      .catch((error) => {
        console.error("Oops, something went wrong!");
        console.error(
          "Please, report the following error on  

https://github.com/ckeditor/ckeditor5/issues with the build id  

          and the error stack trace:"
        );
        console.warn("Build id: grazqcvdbgrw-9tywlm90kmcf");
        console.error(error);
      });
  },

```



```

    });
  },

  methods: {
    onCreateNote() {
      this.$emit("create", {
        ...this.note,
        title: this.noteModel.title,
        noteTags: this.noteModel.noteTags,
        noteDocument: window.editor.getData(),
      });
    },
    onUpdateNote() {
      this.$emit("update", {
        ...this.note,
        title: this.noteModel.title,
        noteTags: this.noteModel.noteTags,
        noteDocument: window.editor.getData(),
      });
    },
    onDeleteNote() {
      if (window.confirm("Вы точно хотите удалить заметку?")) {
        this.$emit("delete", this.note.id);
      }
    },
    onShareNote() {
      getNoteSharedId(this.note.id,
this.$store.state.Auth.user).then(
      ({ data }) => {
        debugger;
        alert(`http://localhost:8080/Shared?id=${data}`);
      }
    );
  },
},
};
</script>

<style>
.ck-editor__editable {
  height: 450px;
}
</style>
<template>
  <div class="p-2">
    <div v-for="note in notes" :key="note.id">
      <ListCardNote :note="note"
@openNote="onOpenNote"></ListCardNote>
    </div>
  </div>
</template>

```

```

<script>
import ListCardNote from "@components/ListCardNote.vue";
export default {
  name: "ListNotes",
  components: {
    ListCardNote,
  },
  props: {
    notes: Array,
  },
  emits: ["openNote"],
  methods: {
    onOpenNote(id) {
      this.$emit("openNote", id);
    }
  }
};
</script>
<template>
  <div class="card m-0 mb-1" :class="{ 'current-note':
isCurrent}" v-on:click="cardClick">
    <div class="card-body">
      <p class="p-0 m-0 fw-bold fs-5">
        {{ note.title }}
      </p>
      <p class="p-0 m-0 fs-6">Тегі: {{ categories }}</p>
      <p class="p-0 m-0 fs-6">Дата: {{ date }}</p>
    </div>
  </div>
</template>

<script>
export default {
  name: "ListCardNote",
  props: {
    note: Object,
  },
  emits: ["openNote"],
  computed: {
    categories() {
      return this.note.noteTags.join();
    },
    date() {
      let date = new Date(this.note.createdDateTime);
      return date.toLocaleDateString("en-US");
    },
    isCurrent() {
      return this.note.isCurrent;
    }
  },
  methods: {
    cardClick() {

```

```

        this.$emit("openNote", this.note.id);
    }
}
};
</script>

<style scoped>
.current-note{
    background-color: #10b981;
    color: white;
}
.card:hover{
    border: solid 1px green;
}
</style>
using System;
using System.Collections.Generic;
using AutoMapper;
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;
using WebNotesApi.Authorization;
using WebNotesApi.Models.Notes;
using WebNotesApplication.Models;
using WebNotesApplication.Services;

namespace WebNotesApi.Controllers
{
    [Authorize]
    [ApiController]
    [Route("api/[controller]")]
    public class NotesController
    {
        private readonly INoteService _noteService;
        private readonly IMapper _mapper;

        public NotesController(INoteService noteService, IMapper
mapper)
        {
            _noteService = noteService;
            _mapper = mapper;
        }

        [HttpGet("{id:int}")]
        public async Task<ActionResult<NoteResponse>>
GetNote(int id)
        {
            var response = await _noteService.GetNote(new
SearchNoteModel()
            {
                Id = id
            });

```

```

        return _mapper.Map<NoteResponse>(response);
    }

    [HttpGet("[action]")]
    public async
Task<ActionResult<IEnumerable<NoteResponse>>>
GetNotes([FromQuery] SearchNoteModel searchNoteModel)
    {
        var response = await
_noteService.GetNotes(searchNoteModel);
        var lResponse =
_mapper.Map<IEnumerable<NoteResponse>>(response);
        foreach (var note in lResponse)
        {
            note.AllAccessTags = await _noteService.Tags(
note.UserId);
        }

        return new OkObjectResult(lResponse);
    }

    [HttpPost("[action]")]
    public async Task<ActionResult<NoteResponse>>
CreateNote(CreateNoteRequest request)
    {
        var response = await
_noteService.CreateNoteAsync(_mapper.Map<CreateNoteModel>(request));

        var lResponse = _mapper.Map<NoteResponse>(response);
        lResponse.AllAccessTags = await _noteService.Tags(
lResponse.UserId);

        return new OkObjectResult(lResponse);
    }

    [HttpPost("[action]")]
    public async Task<ActionResult<NoteResponse>>
UpdateNote(UpdateNoteRequest request)
    {
        var response = await
_noteService.UpdateNoteAsync(_mapper.Map<UpdateNoteModel>(request));

        var lResponse = _mapper.Map<NoteResponse>(response);
        lResponse.AllAccessTags = await _noteService.Tags(
lResponse.UserId);

        return new OkObjectResult(lResponse);
    }

```

```

        [HttpDelete("[action]/{id:int}")]
        public async Task<ActionResult> DeleteNote(int id)
        {
            await _noteService.DeleteNoteAsync(id);

            return new OkResult();
        }

        [AllowAnonymous]
        [HttpGet("[action]/{id}")]
        public async Task<ActionResult<NoteResponse>>
GetNoteShared(Guid id)
        {
            var response = await _noteService.GetNote(new
SearchNoteModel()
            {
                SharedId = id
            });

            var lResponse = _mapper.Map<NoteResponse>(response);
            lResponse.AllAccessTags = await _noteService.Tags(
lResponse.UserId);

            return new OkObjectResult(lResponse);
        }

        [AllowAnonymous]
        [HttpGet("[action]/{id}")]
        public async Task<Guid> GetNoteSharedId(int id)
        {
            var response = await
_noteService.CreateSharedId(id);

            return response;
        }

        [AllowAnonymous]
        [HttpGet("[action]/{id}")]
        public async Task<ActionResult<IEnumerable<string>>>
GetTags(int userId)
        {
            var response = await _noteService.Tags(userId);

            return response;
        }
    }

    }

using System.Collections.Generic;
using AutoMapper;
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;
using WebNotesApi.Authorization;

```

```

using WebNotesApi.Models.Users;
using WebNotesApplication.Models;
using WebNotesApplication.Services;
using WebNotesData.Entities;

namespace WebNotesApi.Controllers
{
    [Authorize]
    [ApiController]
    [Route("api/[controller]")]
    public class UsersController : ControllerBase
    {
        private readonly IUserService _userService;
        private readonly IMapper _mapper;

        public UsersController(IUserService userService, IMapper
mapper)
        {
            _userService = userService;
            _mapper = mapper;
        }

        [AllowAnonymous]
        [HttpPost("[action]")]
        public async Task<IActionResult>
Authenticate(AuthenticateRequest model)
        {
            var response = await
_userService.AuthenticateAsync(_mapper.Map<LoginModel>(model));
            HttpContext.Items["User"] =
_mapper.Map<User>(response);
            return Ok(response);
        }

        [AllowAnonymous]
        [HttpPost("[action]")]
        public async Task<IActionResult>
Register(RegisterRequest model)
        {
            var response = await
_userService.RegisterAsync(_mapper.Map<User>(model));
            return Ok(response);
        }

        [Authorize(Role.Admin)]
        [HttpGet("[action]")]
        public async Task<ActionResult<List<User>>> GetAll()
        {
            var users = await _userService.GetAllAsync();
            return users;
        }
    }
}

```

```

        [Authorize(Role.Admin)]
        [HttpDelete("[action]/{id:int}")]
        public async Task<ActionResult<List<User>>> Delete(int
id)
        {
            await _userService.DeleteUserAsync(id);
            return Ok();
        }

        [Authorize(Role.Admin)]
        [HttpPost("[action]/{id:int}")]
        public async Task<ActionResult<List<User>>>
ResetPassword(int id)
        {
            await _userService.ResetPassword(id);
            return Ok();
        }

        [HttpGet("{id:int}")]
        public async Task<IActionResult> GetById(int id)
        {
            // only admins can access other user records
            var currentUser = (User)HttpContext.Items["User"];
            if (id != currentUser.Id && currentUser.Role !=
Role.Admin)
            {
                return Unauthorized(new { message =
"Unauthorized" });
            }

            var user = await _userService.GetByIdAsync(id);
            return Ok(user);
        }
    }
}
using System.Collections.Generic;

namespace WebNotesApi.Models.Notes
{
    public class CreateNoteRequest
    {
        public string Title { get; set; }
        public string NoteDocument { get; set; }
        public int UserId { get; set; }

        public List<string> NoteTags { get; set; }
    }
}
using System.Linq;
using AutoMapper;
using BCrypt.Net;
using WebNotesApi.Models.Notes;

```

```

using WebNotesApi.Models.Users;
using WebNotesApplication.Models;
using WebNotesData.Entities;
using BCryptNet = BCrypt.Net.BCrypt;

namespace WebNotesApi.Profiles
{
    public class MappingProfile : Profile
    {
        public MappingProfile()
        {
            CreateMap<User, AuthenticateResult>().ReverseMap();
            CreateMap<AuthenticateResponse,
AuthenticateResult>().ReverseMap();
            CreateMap<AuthenticateRequest,
LoginModel>().ReverseMap();
            CreateMap<RegisterRequest, User>()
                .ForMember(user => user.PasswordHash, opt =>
opt.MapFrom(request => BCryptNet.HashPassword(request.Password,
SaltRevision.Revision2B)))
                .ForMember(user => user.Role, opt =>
opt.MapFrom(request => Role.User));
            CreateMap<CreateNoteModel, Note>()
                .ForMember(x => x.NoteTags, m => m.Ignore());
            CreateMap<CreateNoteRequest, CreateNoteModel>();
            CreateMap<UpdateNoteRequest, UpdateNoteModel>();
            CreateMap<UpdateNoteModel, Note>()
                .ForMember(x => x.NoteTags, m => m.Ignore());
            CreateMap<Note, NoteResponse>().ForMember(x =>
x.NoteTags, c => c.MapFrom(t => t.NoteTags.Select(x =>
x.Tag.Value)));
        }
    }
}

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using WebNotesApplication.Models;
using WebNotesData.Entities;

namespace WebNotesApplication.Services
{
    public interface INoteService
    {
        Task<Note> GetNote(SearchNoteModel searchNoteModel);
        Task<List<Note>> GetNotes(SearchNoteModel
searchNoteModel);
        Task<Note> CreateNoteAsync(CreateNoteModel
createNoteModel);
        Task<Note> UpdateNoteAsync(UpdateNoteModel
updateNoteModel);
    }
}

```



```

        Task DeleteNoteAsync(int id);
        Task<Guid> CreateSharedId(int id);
        Task<List<string>> Tags( int userId);
    }
}
using System.Collections.Generic;
using System.Threading.Tasks;
using WebNotesApplication.Models;
using WebNotesData.Entities;

namespace WebNotesApplication.Services
{
    public interface IUserService
    {
        Task<AuthenticateResult> AuthenticateAsync(LoginModel
model);
        Task<AuthenticateResult> RegisterAsync(User model);
        Task<List<User>> GetAllAsync();
        Task<User> GetByIdAsync(int id);
        Task DeleteUserAsync(int id);
        Task ResetPassword(int id);
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using Microsoft.EntityFrameworkCore;
using WebNotesApplication.Exceptions;
using WebNotesApplication.Models;
using WebNotesData.Context;
using WebNotesData.Entities;

namespace WebNotesApplication.Services
{
    public class NoteService : INoteService
    {
        private readonly ApplicationContext _context;

        private readonly IMapper _mapper;

        public NoteService(
            ApplicationContext context,
            IMapper mapper)
        {
            _context = context;
            _mapper = mapper;
        }

        public async Task<Note> GetNote(SearchNoteModel
searchNoteModel)

```

```

        {
            var note = await _context.Notes.AsNoTracking()
                .Include(x => x.NoteTags)
                .ThenInclude(x => x.Tag)
                .Include(x=>x.ShareNotes)
                .SingleOrDefaultAsync(n => n.Id ==
searchNoteModel.Id || n.ShareNotes.Any(s=>s.Link ==
searchNoteModel.SharedId.ToString()));
            if (note is null)
            {
                throw new ApplicationException($"Note with id =
{searchNoteModel.Id} not founded");
            }

            return note;
        }

        public async Task<List<Note>> GetNotes(SearchNoteModel
searchNoteModel)
        {
            var notes = await _context.Notes.AsNoTracking()
                .Include(x => x.NoteTags).ThenInclude(x =>
x.Tag)
                .Where(x =>

string.IsNullOrEmpty(searchNoteModel.SearchValue) ||

x.NoteDocument.Contains(searchNoteModel.SearchValue) ||

x.Title.Contains(searchNoteModel.SearchValue) ||
                x.NoteTags.Any(n =>
n.Tag.Value.Contains(searchNoteModel.SearchValue)))
                .ToListAsync();
            if (!notes.Any())
            {
                throw new ApplicationException($"Notes with search
criteria not founded");
            }

            return notes;
        }

        public async Task<Note> CreateNoteAsync(CreateNoteModel
createNoteModel)
        {
            var note = _mapper.Map<Note>(createNoteModel);
            note.CreatedDateTime = DateTime.Now;
            await _context.Notes.AddAsync(note);
            await _context.SaveChangesAsync();
            await CreateOrUpdateTags(note,
createNoteModel.NoteTags);
            // TODO Add NoteTags create and update

```

```

        return await GetNoteFull(note.Id); // include extend
table
    }

    public async Task<Note> UpdateNoteAsync(UpdateNoteModel
updateNoteModel)
    {
        var note = await
_context.Notes.SingleOrDefaultAsync(n => n.Id ==
updateNoteModel.Id);
        note.NoteDocument = updateNoteModel.NoteDocument;
        note.Title = updateNoteModel.Title;

        // TODO Add NoteTags create and update
_context.Notes.Update(note);
        await _context.SaveChangesAsync();
        await CreateOrUpdateTags(note,
updateNoteModel.NoteTags);

        return await GetNoteFull(note.Id); // include extend
table
    }

    public async Task DeleteNoteAsync(int id)
    {
        var note = await
_context.Notes.SingleOrDefaultAsync(n => n.Id == id);
        _context.Notes.Remove(note);

        await _context.SaveChangesAsync();
    }

    public async Task<Guid> CreateSharedId(int id)
    {
        var note = await
_context.Notes.SingleOrDefaultAsync(n => n.Id == id);
        var shared = new ShareNote()
        {
            CreatedDateTime = DateTime.Now,
            Link = Guid.NewGuid().ToString(),
            NoteId = note.Id
        };
        await _context.ShareNotes.AddAsync(shared);
        await _context.SaveChangesAsync();

        return Guid.Parse(shared.Link);
    }

    public async Task<List<string>> Tags(int userId)
    {

```

```

        var tags = _context.Tags.Where(x => x.UserId ==
userId || x.UserId == null);

        return await
tags.Select(t=>t.Value).Distinct().ToListAsync();
    }

    private async Task CreateOrUpdateTags(Note note,
List<string> tags)
    {
        var existedNoteTags = await _context.NoteTags
.AsNoTracking()
.Include(x => x.Tag)
.Where(t => t.NoteId == note.Id && (t.Tag.UserId
== null || t.Tag.UserId == t.Tag.UserId))
.ToListAsync();

        var removeNoteTags = existedNoteTags.Where(t =>
!tags.Contains(t.Tag.Value));
        var addTags = tags.Where(t => existedNoteTags.All(nt
=> nt.Tag.Value != t)).ToList();

        _context.NoteTags.RemoveRange(removeNoteTags);
        await _context.SaveChangesAsync();

        var tagsDb = await
_context.Tags.AsNoTracking().ToListAsync();

        var addNewNoteTags = tagsDb.Where(at =>
addTags.Exists(x => x == at.Value)).ToList();
        var addNewTags = addTags.Where(at =>
addNewNoteTags.All(x => x.Value != at)).ToList();

        foreach (var tag in addNewTags)
        {
            var newTag = new Tag()
            {
                Value = tag,
                UserId = note.UserId
            };
            await _context.Tags.AddAsync(newTag);

            var newNoteTag = new NoteTag()
            {
                NoteId = note.Id,
                Tag = newTag
            };
            await _context.NoteTags.AddAsync(newNoteTag);
            await _context.SaveChangesAsync();
        }

        foreach (var tag in addNewNoteTags)

```

```

        {
            var existTag = await
            _context.Tags.SingleOrDefaultAsync(t => t.Value == tag.Value);
            var newNoteTag = new NoteTag()
            {
                NoteId = note.Id,
                TagId = existTag.Id
            };
            await _context.NoteTags.AddAsync(newNoteTag);
            await _context.SaveChangesAsync();
        }
    }

    private async Task<Note> GetNoteFull(int id)
    {
        return await _context.Notes.AsNoTracking().Include(x
=> x.NoteTags).ThenInclude(x => x.Tag).SingleOrDefaultAsync(n =>
n.Id == id);
    }
}

using AutoMapper;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Options;
using System.Collections.Generic;
using System.Threading.Tasks;
using WebNotesApplication.Authorization;
using WebNotesApplication.Exceptions;
using WebNotesApplication.Models;
using WebNotesData.Context;
using WebNotesData.Entities;
using BCryptNet = BCrypt.Net.BCrypt;

namespace WebNotesApplication.Services
{
    public class UserService : IUserService
    {
        private readonly ApplicationContext _context;
        private readonly IJwtUtils _jwtUtils;
        private readonly AppSettings _appSettings;

        private readonly IMapper _mapper;

        public UserService(
            ApplicationContext context,
            IJwtUtils jwtUtils,
            IOptions<AppSettings> appSettings,
            IMapper mapper)
        {
            _context = context;
            _jwtUtils = jwtUtils;
            _appSettings = appSettings.Value;
        }
    }
}

```

```

        _mapper = mapper;
    }

    public async Task<AuthenticateResult>
AuthenticateAsync(LoginModel model)
    {
        var user = await
_context.Users.SingleOrDefaultAsync(x => x.UserName ==
model.Username);

        // validate
        if (user == null ||
!BCryptNet.Verify(model.Password, user.PasswordHash))
        {
            throw new AppException("Username or password is
incorrect");
        }

        // authentication successful so generate jwt token
        var jwtToken = _jwtUtils.GenerateJwtToken(user);

        var authResult =
_mapper.Map<AuthenticateResult>(user);
        authResult.Token = jwtToken;

        return authResult;
    }

    public async Task<List<User>> GetAllAsync()
    {
        var users = await
_context.Users.AsNoTracking().ToListAsync();
        return users;
    }

    public async Task<User> GetByIdAsync(int id)
    {
        var user = await _context.Users.FindAsync(id);
        if (user == null)
        {
            throw new KeyNotFoundException("User not
found");
        }

        return user;
    }

    public Task DeleteUserAsync(int id)
    {
        throw new System.NotImplementedException();
    }

```

```

        public Task ResetPassword(int id)
        {
            throw new System.NotImplementedException();
        }

        public async Task<AuthenticateResult> RegisterAsync(User
user)
        {
            var existUser = await
_context.Users.SingleOrDefaultAsync(x => x.UserName ==
user.UserName);
            if (existUser is not null)
            {
                throw new AppException($"User with user name
{user.UserName} exists");
            }

            await _context.Users.AddAsync(user);
            await _context.SaveChangesAsync();

            var jwtToken = _jwtUtils.GenerateJwtToken(user);

            var authResult =
_mapper.Map<AuthenticateResult>(user);
            authResult.Token = jwtToken;

            return authResult;
        }
    }
}
using WebNotesData.Entities;

namespace WebNotesApplication.Authorization
{
    public interface IJwtUtils
    {
        public string GenerateJwtToken(User user);
        public int? ValidateJwtToken(string token);
    }
}
using Microsoft.Extensions.Options;
using Microsoft.IdentityModel.Tokens;
using System;
using System.IdentityModel.Tokens.Jwt;
using System.Linq;
using System.Security.Claims;
using System.Text;
using WebNotesData.Entities;

namespace WebNotesApplication.Authorization
{

```

```

public class JwtUtils : IJwtUtils
{
    private readonly AppSettings _appSettings;

    public JwtUtils(IOptions<AppSettings> appSettings)
    {
        _appSettings = appSettings.Value;
    }

    public string GenerateJwtToken(User user)
    {
        // generate token that is valid for 7 days
        var tokenHandler = new JwtSecurityTokenHandler();
        var key =
Encoding.ASCII.GetBytes(_appSettings.Secret);
        var tokenDescriptor = new SecurityTokenDescriptor
        {
            Subject = new ClaimsIdentity(new Claim[]
            {
                new Claim("id", user.Id.ToString()),
                new Claim(ClaimTypes.Role,
((int)user.Role).ToString())
            },
            Expires = DateTime.UtcNow.AddDays(7),
            SigningCredentials = new SigningCredentials(new
SymmetricSecurityKey(key),
SecurityAlgorithms.HmacSha256Signature)
        };
        var token =
tokenHandler.CreateToken(tokenDescriptor);
        return tokenHandler.WriteToken(token);
    }

    public int? ValidateJwtToken(string token)
    {
        if (token == null)
        {
            return null;
        }

        var tokenHandler = new JwtSecurityTokenHandler();
        var key =
Encoding.ASCII.GetBytes(_appSettings.Secret);
        try
        {
            tokenHandler.ValidateToken(token, new
TokenValidationParameters
            {
                ValidateIssuerSigningKey = true,
                IssuerSigningKey = new
SymmetricSecurityKey(key),
                ValidateIssuer = false,

```



```

        ValidateAudience = false,
        // set clockskew to zero so tokens expire
exactly at token expiration time (instead of 5 minutes later)
        ClockSkew = TimeSpan.Zero
    }, out SecurityToken validatedToken);

    var jwtToken = (JwtSecurityToken)validatedToken;
    var userId = int.Parse(jwtToken.Claims.First(x
=> x.Type == "id").Value);

    // return user id from JWT token if validation
successful
    return userId;
}
catch
{
    // return null if validation fails
    return null;
}
}
}
using Microsoft.EntityFrameworkCore;
using WebNotesData.Entities;

namespace WebNotesData.Context
{
    public class ApplicationContext : DbContext
    {
        public DbSet<User> Users { get; set; }
        public DbSet<Note> Notes { get; set; }
        public DbSet<Tag> Tags { get; set; }
        public DbSet<NoteTag> NoteTags { get; set; }
        public DbSet<ShareNote> ShareNotes { get; set; }
        public
ApplicationContext(DbContextOptions<ApplicationContext> options)
            : base(options)
        {
        }
    }
}
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace WebNotesData.Entities
{
    public class Note
    {
        public int Id { get; set; }
    }
}

```

```

        [Column(TypeName = "NVARCHAR(MAX)")]
        public string Title { get; set; }

        [Column(TypeName = "NVARCHAR(MAX)")]
        [MaxLength]
        public string NoteDocument { get; set; }

        [Required]
        public DateTime CreatedDateTime { get; set; }

        public int UserId { get; set; }

        public User User { get; set; }

        public List<NoteTag> NoteTags { get; set; }

        public bool IsBookmark { get; set; }

        public List<ShareNote> ShareNotes { get; set; }
    }
}
namespace WebNotesData.Entities
{
    public class NoteTag
    {
        public int Id { get; set; }

        public int TagId { get; set; }
        public int NoteId { get; set; }

        public Tag Tag { get; set; }
        public Note Note { get; set; }
    }
}
namespace WebNotesData.Entities
{
    public enum Role
    {
        Admin,
        User
    }
}
using System;
using System.ComponentModel.DataAnnotations;

namespace WebNotesData.Entities
{
    public class ShareNote
    {
        public int Id { get; set; }
    }
}

```

```

        [Required]
        public string Link { get; set; }

        [Required]
        public DateTime CreatedDateTime { get; set; }
        public DateTime InactivatedDateTime { get; set; }

        public Note Note { get; set; }
        public int NoteId { get; set; }
    }
}
#nullable enable
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace WebNotesData.Entities
{
    public class Tag
    {
        public int Id { get; set; }

        [Required]
        public string Value { get; set; }

        public User? User { get; set; }

        public int? UserId { get; set; }

        public List<NoteTag> NoteTags { get; set; }
    }
}
using System.Collections.Generic;
using System.Text.Json.Serialization;

namespace WebNotesData.Entities
{
    public class User
    {
        public int Id { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Email { get; set; }
        public string UserName { get; set; }
        public Role Role { get; set; }

        [JsonIgnore]
        public string PasswordHash { get; set; }

        public List<Note> Notes { get; set; }
    }
}
using Microsoft.AspNetCore.Builder;

```

```

using Microsoft.AspNetCore.Hosting;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.IdentityModel.Logging;
using Microsoft.OpenApi.Models;
using System.Collections.Generic;
using System.Text;
using System.Text.Json.Serialization;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using WebNotesApi.Authorization;
using WebNotesApi.Helpers;
using WebNotesApplication;
using WebNotesApplication.Authorization;
using WebNotesApplication.Services;
using WebNotesData.Context;
using WebNotesData.Entities;
using BCryptNet = BCrypt.Net.BCrypt;

namespace web_notes_api
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this
        method to add services to the container.
        public void ConfigureServices(IServiceCollection
        services)
        {
            services.AddDbContext<ApplicationContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("DefaultC
onnection")), ServiceLifetime.Transient);
            services.AddAutoMapper(typeof(Startup));
            services.AddCors();
            services.AddControllers().AddJsonOptions(x =>
            {
                // serialize enums as strings in api responses
                (e.g. Role)
                x.JsonSerializerOptions.Converters.Add(new
JsonStringEnumConverter());
            });

            // configure strongly typed settings object

```

```

services.Configure<AppSettings>(Configuration.GetSection("AppSettings"));

// configure DI for application services
services.AddScoped<IJwtUtils, JwtUtils>();
services.AddScoped<IUserService, UserService>();
services.AddScoped<INoteService, NoteService>();
services.AddSwaggerGen(opt =>
{
    opt.SwaggerDoc("v1", new OpenApiInfo { Title =
"My Api", Version = "v1" });
    opt.AddSecurityDefinition("bearer", new
OpenApiSecurityScheme
    {
        Type = SecuritySchemeType.Http,
        BearerFormat = "JWT",
        In = ParameterLocation.Header,
        Scheme = "bearer"
    });
});

var key =
Encoding.ASCII.GetBytes(Configuration.GetSection("AppSettings:secret").Value);
services.AddAuthentication(x =>
{
    x.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
    x.DefaultChallengeScheme =
JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(x =>
{
    x.RequireHttpsMetadata = false;
    x.SaveToken = true;
    x.TokenValidationParameters = new
TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new
SymmetricSecurityKey(key),
        ValidateIssuer = false,
        ValidateAudience = false
    };
});
}

// This method gets called by the runtime. Use this
method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app,
IWebHostEnvironment env, ApplicationContext context)

```

```

    {
        if (env.IsDevelopment())
        {
            IdentityModelEventSource.ShowPII = true;
            app.UseDeveloperExceptionPage();
            app.UseSwagger();
            app.UseSwaggerUI(c =>
c.SwaggerEndpoint("/swagger/v1/swagger.json", "WebNotesApi
v1"));
        }

        app.UseHttpsRedirection();

        app.UseRouting();

        app.UseAuthentication();
        app.UseAuthorization();

        // global cors policy
        app.UseCors(x => x
            .AllowAnyOrigin()
            .AllowAnyMethod()
            .AllowAnyHeader());

        // global error handler
        app.UseMiddleware<ErrorHandlerMiddleware>();

        // custom jwt auth middleware
        app.UseMiddleware<JwtMiddleware>();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
        });
    }
}

```