Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет инженерно-экономический

Кафедра экономической информатики

Дисциплина «Программирование сетевых приложений»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  старший преподаватель  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Т.М. Унучек |
|  | \_\_\_.\_\_\_\_.2021 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«Программа складского учета и регистрации заказов»**

БГУИР КП 1-40 01 02-08 015 ПЗ

|  |  |
| --- | --- |
|  | Выполнил студент группы 914301  БОГОМАЗ Дмитрий Леонидович  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовой проект представлен на проверку \_\_\_.\_\_\_\_.2021  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2021

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc90239349)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ РЕДАКТИРОВАНИЯ ЗАМЕТОК 6](#_Toc90239350)

[2 ПОСТАНОВКА ЗАДАЧИ И ОБЗОР МЕТОДОВ ЕЕ РЕШЕНИЯ ПО РЕДАКТИРОВАНИЮ ЗАМЕТОК 14](#_Toc90239351)

[3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ НА ОСНОВЕ СТАНДАРТА IDEF0 23](#_Toc90239352)

[4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЁ ОПИСАНИЕ 26](#_Toc90239353)

[5 ОПИСАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ СЕРВЕРНОЙ ЧАСТИ 28](#_Toc90239354)

[6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 35](#_Toc90239355)

[7 РЕЗУЛЬТАТ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ 40](#_Toc90239356)

[ЗАКЛЮЧЕНИЕ 42](#_Toc90239357)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 43](#_Toc90239358)

[ПРИЛОЖЕНИЕ А 44](#_Toc90239359)

[ПРИЛОЖЕНИЕ В 57](#_Toc90239360)

# ВВЕДЕНИЕ

 Приложения для хранения и совместного редактирования заметок является не новой коммерческой идей. Примеров работающих и поддерживаемых веб-сервисов на данный момент достаточно, однако на примере данного приложения легко продемонстрировать основные аспекты коммерческой разработки, спроектировать работающую информационную систему, выбрать верные архитектурные решения, подобрать технологии для решения поставленных задач.

Структурирование самих заметок будет предполагать создание удобного для пользователя порядка самих заметок в сетке. Также для более понятного просмотра всех заметок предусмотрен просмотр заметок в виде таблицы.

Сервис также должен предоставлять пользователям возможность создания своего личного кабинета.

Также сервис должен предлагать пользователю фильтрацию, сортировку и поиск заметок по всем существующим полям самой заметки.

Целью является создание приложения для обеспечения хранения заметок с возможностью удобного просмотра и совместного редактирования самих заметок.

Главными задачами курсового проекта является:

* исследование предметной области;
* реализация клиент-серверного приложения;
* создание понятного интерфейса для взаимодействия с заметками;
* реализация базового функционала для работы с заметками;
* создания удобной среды для совместного редактирования заметок.

# ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ РЕДАКТИРОВАНИЯ ЗАМЕТОК

Приложения заметок широко распространены в сети, и миллионы людей, которые их используют, похоже, хотят разных вещей от каждого приложения. В отличие от почтовых приложений, в которых вашей целью является чтение, сортировка и удаление почты или погодных приложений для проверки ежедневного прогноза, поиск подходящего приложения для заметок представляет большую проблему при определении того, какое из них является лучшим.

Некоторым пользователям нужен минималистичный подход к заметкам, в то время как другие ищут приложение для заметок с богатым набором функций и различными инструментами и настройками. Разрабатываемое приложение будет схоже с Google Keep. Она будет реализовывать минимальный функционал с удобным интерфейсом.

Независимо от того, насколько простым или надежным должно быть ваше приложение для заметок, мы искали определенный набор качеств при тестировании каждого кандидата:

* Особенности – В идеале наши любимые приложения для создания заметок могут надежно синхронизироваться с облачными службами, обрабатывать конфликты копий, иметь простой импорт / экспорт, хранить большое количество заметок без замедления и предлагать варианты совместного использования для других приложений. Кроме того, огромным фактором является поиск; нужно, чтобы приложение могло быстро находить заметку независимо от того, сколько у нас есть.
* Дизайн. Отличный дизайн пользовательского интерфейса и пользовательского интерфейса имеет первостепенное значение при использовании приложения для заметок. Среда чтения и письма должна быть простой, но функциональной, а навигация по приложению должна казаться естественной.
* Простота использования. Отличное приложение для заметок будет достаточно интуитивно понятным, чтобы создавать, находить и систематизировать наши заметки при первом запуске.

Также можно категоризировать самих пользователей по стилю ведения заметок:

* Архитектор. Им нравится планировать, разрабатывать процессы и структуры, и им нужен инструмент для заметок, который позволит им легко структурировать свои идеи.
* Садовник. Им нравится исследовать, объединять различные мысли воедино, и им нужен инструмент для заметок, который позволит им легко развивать свои идеи.
* Библиотекарь. Им нравится собирать и составлять каталог ресурсов, и им нужен инструмент для заметок, который позволит им легко находить свои идеи.

Однако есть и четвертая категория. Есть приложения для быстрого сохранения и быстрого извлечения заметок. Не все в значительной степени полагаются на записи, чтобы фиксировать, систематизировать и извлекать из памяти свои мысли. Если просто нужно легкое решение для моментов, связанных с идеями, или для быстрого составления списка дел, также можно использовать приложение для быстрого создания заметок.

Фактически, многие люди используют приложение для быстрого захвата в дополнение к полноценному инструменту для создания заметок. Проблема с такими приложениями заключается в том, что многие люди в конечном итоге создают кучу непонятных заметок, которые сложно искать. Поэтому решением нашего в нашем приложении будет являться сортировка и фильтрация заметок для удобного использования.

Так как было выявлено, что приложения является аналогом Google Keep, то следует выделить сильные и слабые стороны данного приложения.

Сильные стороны:

* Функциональность поиска отличная.
* Чистый, простой, но привлекательный интерфейс.
* Легко использовать.
* Позволяет изображению быть частью заметки.
* Примечания могут быть довольно длинными.
* Надежное место для сохранения информации, которая вам понадобится позже.
* Прикреплен к учетной записи Google.
* Возможность добавлять соавторов к заметке.
* Синхронизация со всеми устройствами.
* Возможность работы с различными видами ресурсов.
* Координация цвета.
* Живые плитки.

Слабыми сторонами являются:

* Невозможно изменить формат текста.
* Пользователи не могут отменить изменения.
* Уведомления не отправляются при добавлении задач.
* Отсутствие папок для организации.
* Недостаточная скорость обновления при совместном редактировании заметок.

Многие слабые стороны являются также и особенностями, поэтому можно говорить о том, что Google Keep практически идеален для быстрого сохранения заметок. Единственным существенным недостатком является скорость обновления при совместном редактировании заметок.

Большинство функционала Google Keep перенесено в приложение, поэтому можно говорить о том, что имеются практически идентичные сильные и слабые стороны приложения.

Информационная система хранения заметок должна удовлетворять требованиям по быстродействию, точности хранения информации, надежно защищено, так как в заметках могут храниться данные пользователей.

Данные требования позволят улучшить стабильность системы и долгосрочно поддерживать систему без сбоев.

В дальнейшем будет требоваться разработка приложения на различные платформы, так как это является важным аспектом для успешного приложения. Это может позволить синхронизировать данные удобно.

Также в будущем можно использовать базу данных на стороне клиента и синхронизировать с базой данных сервера. Это позволит создать офлайн сервис по редактированию заметок с помощью технологии PWA.

# ПОСТАНОВКА ЗАДАЧИ И ОБЗОР МЕТОДОВ ЕЕ РЕШЕНИЯ ПО РЕДАКТИРОВАНИЮ ЗАМЕТОК

Задачей данного раздела является обзор методов решения, по созданию приложения для хранения и редактирования заметок.

Для реализации поставленных целей данного курсового проекта необходимо выполнить следующие задачи:

* исследование предметной области;
* реализация клиент-серверного приложения;
* создание понятного интерфейса для взаимодействия с заметками;
* реализация базового функционала для работы с заметками;
* создания удобной среды для совместного редактирования заметок.

Основным методом решения будет являться создание многоуровневой архитектуры с использованием базовых паттернов для веб-проектирования.

**Выбор подходящего программного обеспечения для редактирования заметок**

В качестве сервера для веб-приложения был выбран фреймворк Java – Spring Boot. Он будет выполнять роль API для клиентского приложения. Java представляет собой язык программирования и платформу вычислений, которая была впервые выпущена Sun Microsystems в 1995 г. Существует множество приложений и веб-сайтов, которые не работают при отсутствии установленной Java, и с каждым днем число таких веб-сайтов и приложений увеличивается. Java отличается быстротой, высоким уровнем защиты и надежностью. Java — язык программирования общего назначения. Относится к объектно-ориентированным языкам программирования, к языкам с сильной типизацией. Java является объектно-ориентированным языком, относится к языкам программирования с сильной типизацией. Эта задача решается благодаря компиляции написанного на Java кода в байт-код. Этот формат исполняет JVM или виртуальная машина Java. JVM — часть среды исполнения Java (JRE). Виртуальная машина не зависит от платформы.

Spring Boot — это среда на основе Java с открытым исходным кодом, используемая для создания микросервиса. Он разработан Pivotal Team и используется для создания автономных и готовых к использованию пружин. В этой главе вы познакомитесь с Spring Boot и познакомитесь с его основными понятиями. Spring Boot предоставляет хорошую платформу для разработчиков Java для разработки автономного и производительного весеннего приложения, которое вы можете просто запустить . Вы можете начать работу с минимальными настройками без необходимости полной настройки конфигурации Spring.

Для интерфейса использовался Angular. Angular представляет фреймворк от компании Google для создания клиентских приложений. Прежде всего он нацелен на разработку SPA-решений (Single Page Application), то есть одностраничных приложений. В этом плане Angular является наследником другого фреймворка AngularJS. В то же время Angular это не новая версия AngularJS, а принципиально новый фреймворк.

Angular предоставляет такую функциональность, как двустороннее связывание, позволяющее динамически изменять данные в одном месте интерфейса при изменении данных модели в другом, шаблоны, маршрутизация и так далее.

Одной из ключевых особенностей Angular является то, что он использует в качестве языка программирования TypeScript. Поэтому перед началом работы рекомендуется ознакомиться с основами данного языка, про которые можно прочитать здесь.

Но мы не ограничены языком TypeScript. При желании можем писать приложения на Angular с помощью таких языков как Dart или JavaScript. Однако TypeScript все-таки является основным языком для Angular.

Последняя версия Angular - Angular 13 вышла в ноябре 2021 года.

В качестве базы данных использовалась MySQL. MySQL — это реляционная система управления базами данных с открытым исходным кодом. В настоящее время эта СУБД одна из наиболее популярных в веб-приложениях — подавляющее большинство CMS использует именно MySQL (часто только её, без альтернатив), а почти все веб-фреймворки поддерживают MySQL уже на уровне базовой конфигурации (без дополнительных модулей).

Из преимуществ СУБД MySQL стоит отметить простоту использования, гибкость, низкую стоимость владения (относительно платных СУБД), а также масштабируемость и производительность.

Для создания серверного приложения использовался паттерн MVC. MVC — это паттерн проектирования веб-приложений, который включает в себя несколько более мелких шаблонов. При использовании MVC на три отдельных компонента разделены модель данных приложения, пользовательский интерфейс и логика взаимодействия пользователя с системой, благодаря чему модификация одного из этих компонентов оказывает минимальное воздействие на остальные или не оказывает его вовсе.

Основная цель применения MVC состоит в разделении данных и бизнес-логики от визуализации. За счет такого разделения повышается возможность повторного использования программного кода: например, добавить представление данных какого-либо существующего маршрута не только в виде HTML, но и в форматах JSON, XML, PDF, XLSX становится очень просто и не требует изменений слоя бизнес-логики исходного маршрута. Также упрощается и сопровождение программного кода: внесение изменений во внешний вид, например, не отражаются на бизнес-логике, а изменения бизнес-логики не затрагивают визуализацию.

Концепция MVC разделяет данные, представление и обработку действий пользователя на компоненты:

Модель — предоставляет собой объектную модель некой предметной области, включает в себя данные и методы работы с этими данными, реагирует на запросы из контроллера, возвращая данные и/или изменяя своё состояние. При этом модель не содержит в себе информации о способах визуализации данных или форматах их представления, а также не взаимодействует с пользователем напрямую.

Представление — отвечает за отображение информации (визуализацию). Одни и те же данные могут представляться различными способами и в различных форматах. Например, коллекцию объектов при помощи разных представлений можно представить на уровне пользовательского интерфейса как в табличном виде, так и списком; на уровне API можно экспортировать данные как в JSON, так в XML или XSLX.

Контроллер — обеспечивает связь между пользователем и системой, использует модель и представление для реализации необходимой реакции на действия пользователя. Как правило, на уровне контроллера осуществляется фильтрация полученных данных и авторизация — проверяются права пользователя на выполнение действий или получение информации.

# 3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ НА ОСНОВЕ СТАНДАРТА IDEF0

Методология IDEF0 предписывает построение иерархической системы диаграмм – единичных описаний фрагментов системы. Сначала проводит­ся описание системы в целом и ее взаимодействия с окружающим миром (контекстная диаграмма), после чего проводится функциональная деком­позиция - система разбивается на подсистемы и каждая подсистема опи­сывается отдельно (диаграммы декомпозиции). Затем каждая подсистема разбивается на более мелкие и так далее до достижения нужной степени подробности. Контекстная диаграмма представлена на рисунке 3.1.

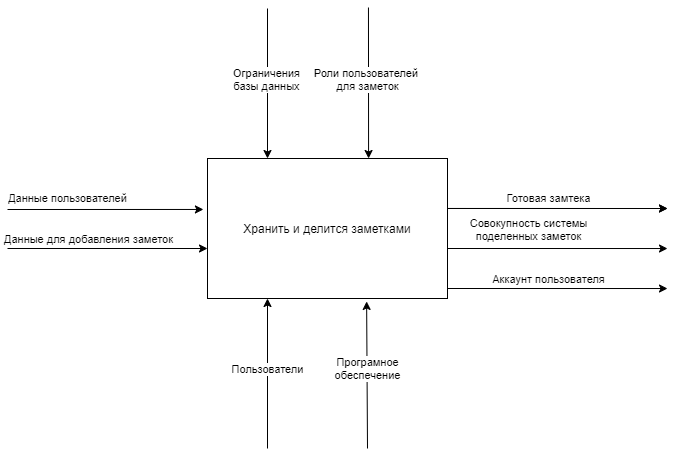


Рисунок 3.1 – Контекстная диаграмма

На основе данных пользователя и данных для добавления заметок, пользователь с помощью программного обеспечения выходит готовая заметка, аккаунт самого пользователя, совокупность системы поделенных заметок. Все это регулируется с помощью ограничений базы данных и ролей пользователя для каждой конкретной заметки.

Проведя декомпозицию контекстной диаграммы, получилось три блока: авторизировать, создать заметку, поделиться с пользователем. В качестве управления использованы те же правила, как и на контекстной диаграмме. После авторизации возвращается аккаунт пользователя, после создания заметки – готовая заметка, а после того, как пользователь поделился заметкой с другими пользователями, появляется система поделенных сообщений.

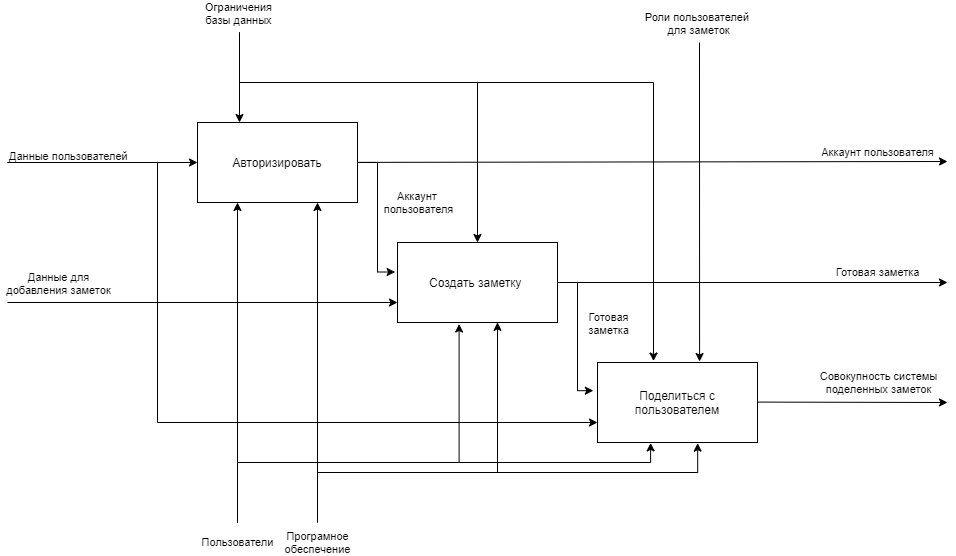


Рисунок 3.2 – Диаграмма декомпозиции контекстной диаграммы

Перейдем на 3 уровень декомпозиции блока «Авторизировать» (рис. 3.3). Диаграмма представлена двумя блоками: «Регистрировать» и «Войти».

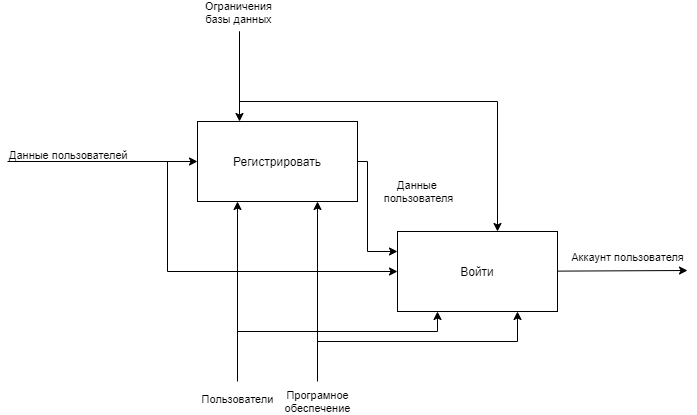


Рисунок 3.3 – Декомпозиция блока «Авторизировать»

На рисунке 3.4 представлена декомпозиция блока «Создать заметку». Она представлена тремя блоками: «Создать пустую заметку», «Изменить текст заметки», «Изменить дизайн заметки».

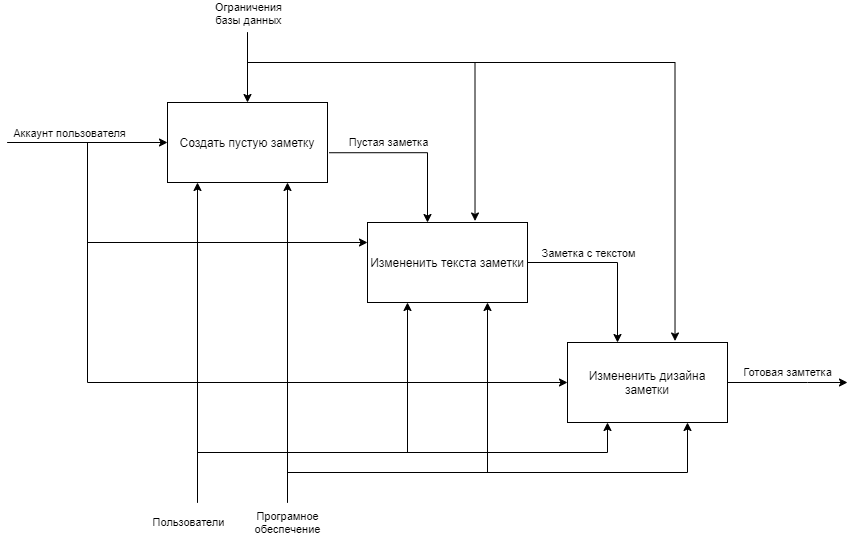


Рисунок 3.4 – Декомпозиция блока «Создать заметку»

На рисунке 3.5 представлена декомпозиция блока «Поделиться с пользователем». Она представлена двумя блоками: «Отослать вопрос на поделиться к пользователю», «Дождаться ответа пользователя».



Рисунок 3.5 – Декомпозиция блока «Поделиться с пользователем»

**4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЁ ОПИСАНИЕ**

MySQL — это реляционная система управления базами данных с открытым исходным кодом. В настоящее время эта СУБД одна из наиболее популярных в веб-приложениях — подавляющее большинство CMS использует именно MySQL (часто только её, без альтернатив), а почти все веб-фреймворки поддерживают MySQL уже на уровне базовой конфигурации (без дополнительных модулей). Hibernate – это ORM фреймворк для Java с открытым исходным кодом. Эта технология является крайне мощной и имеет высокие показатели производительности.Hibernate создаёт связь между таблицами в базе данных (далее – БД) и Java-классами и наоборот.

Для описания информационной модели курсовой работы были созданы 6 сущностей: Пользователь, Уведомление, Дизайн заметки, История заметки, Заметка, Текст заметки.

Сущность «Пользователь» хранит в себе логин, имя пользователя и пароля пользователя.

Сущность «Уведомление» хранит в себе тип заметки, содержание уведомления, дата создания.

Сущность «Дизайн заметки» хранит в себе цвет самой заметки.

Сущность «История заметки» хранит в себе дату создания и дату изменения.

Сущность «Заметка» является одной из ключевой, потому что она хранит связана с большим количеством других сущностей и хранит в себе: роль пользователя, порядок заметки, является ли она свободной в доступе.

Сущность «Текст заметки» хранит в себе текст заголовка и текст содержимого.

«Пользователь» к «Заметка» относится как один ко многим. «Текст заметки» относится к «Заметка» как один ко многим. «Пользователь» к «Уведомление» один ко многим.

Выявив существующие связи между сущностями, строится логический уровень информационной модели, представленную на рисунке 4.1.

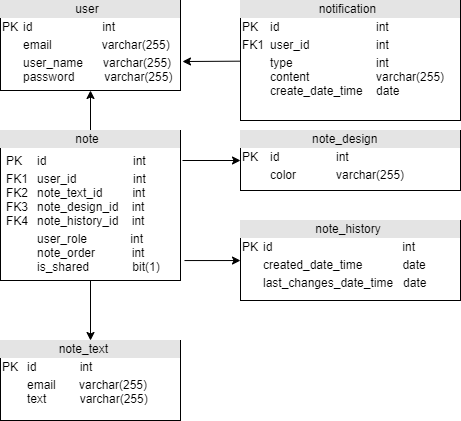


Рисунок 4.1 – Логический уровень информационной модели

Целью построения логической модели является получение графического представления логической структуры исследуемой предметной области. Логическая модель предметной области иллюстрирует сущности, а также их взаимоотношения между собой.

Сущности описывают объекты, являющиеся предметом деятельности предметной области, и субъекты, осуществляющие деятельность в рамках предметной области. Свойства объектов и субъектов реального мира описываются с помощью атрибутов.

База данных была приведена к третей нормальной форме, так как значения в записи, которая не входит в ключ этой записи, не относятся к таблице. В общем, в любое время содержимое группы полей может применяться к более чем одной записи в таблице, рассмотрите возможность размещения этих полей в отдельной таблице.

Для доступа к базе данных используется паттерн «Репозиорий». Эрик Эванс дал точное описание в своей книге: «Репозиорий представляет собой все объекты определенного типа в виде концептуального множества. Его поведение похоже на поведение коллекции, за исключением более развитых возможностей для построения запросов».

**5 ОПИСАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ СЕРВЕРНОЙ ЧАСТИ**

Unified Modeling Language (UML) — унифицированный язык моделирования. Расшифровка: modeling подразумевает создание модели, описывающей объект. Unified (универсальный, единый) — подходит для широкого класса проектируемых программных систем, различных областей приложений, типов организаций, уровней компетентности, размеров проектов. UML описывает объект в едином заданном синтаксисе, поэтому где бы не была нарисована диаграмму, ее правила будут понятны для всех, кто знаком с этим графическим языком — даже в другой стране.

Одна из задач UML — служить средством коммуникации внутри команды и при общении с заказчиком. Давайте рассмотрим возможные варианты использования диаграмм.

У кaждого языкa моделировaния cущеcтвует cвой cловaрь. Cловaрь UML cоcтоит из cледующих элементов. Сущноcти — это aбcтрaкции, которые являютcя оcновными элементaми моделей. Отношения — это cвязи между cущноcтями. Диaгрaммы — это отрaжение взaимодейcтвия cущноcтей и отношений.

**Диаграмма вариантов использования.**

Диаграммы вариантов использования рассматриваются для анализа требований системы высокого уровня. Когда требования системы анализируются, функциональные возможности фиксируются в случаях использования.

Можно сказать, что варианты использования — это не что иное, как системные функции, написанные организованным образом. Второе, что имеет отношение к сценариям использования, — это актеры. Актеры могут быть определены как то, что взаимодействует с системой.

Актерами могут быть пользователи, некоторые внутренние приложения или некоторые внешние приложения. Для составления диаграммы вариантов использования должны быть определены функциональные возможности должны быть представлены как вариант использования, актеры, отношения между вариантами использования и актерами.

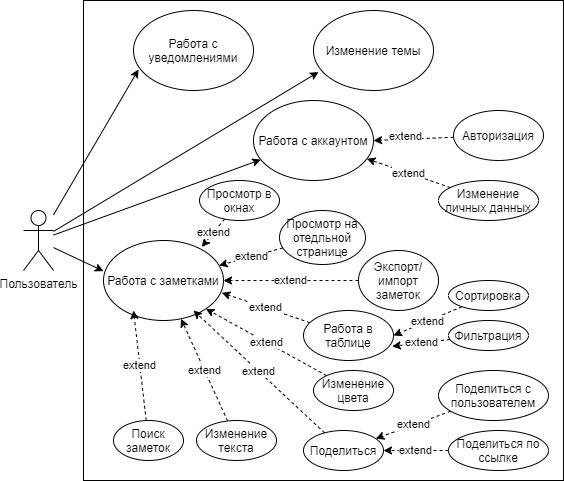


Рисунок 5.1 – Диаграмма вариантов использования

Главными частями использования приложения есть работа с заметками, аккаунтом, уведомлениями и изменение темы. Данные категории уже подразделяются на остальные. Большая часть рабочего функционала связана с самими заметками, их представлением и изменением.

Данная диаграмма описывает возможности самого приложения, рассматривая его с точки зрения пользователя, который мог бы использовать само приложение.

**Диаграмма последовательности.**

Диаграммы последовательностей используются для уточнения диаграмм прецедентов, более детального описания логики сценариев использования. Диаграммы последовательностей обычно содержат объекты, которые взаимодействуют в рамках сценария, сообщения, которыми они обмениваются, и возвращаемые результаты, связанные с сообщениями. Впрочем, часто возвращаемые результаты обозначают лишь в том случае, если это не очевидно из контекста. На рисунке 5.2 представлена диаграмма деятельности работы с заметкой.

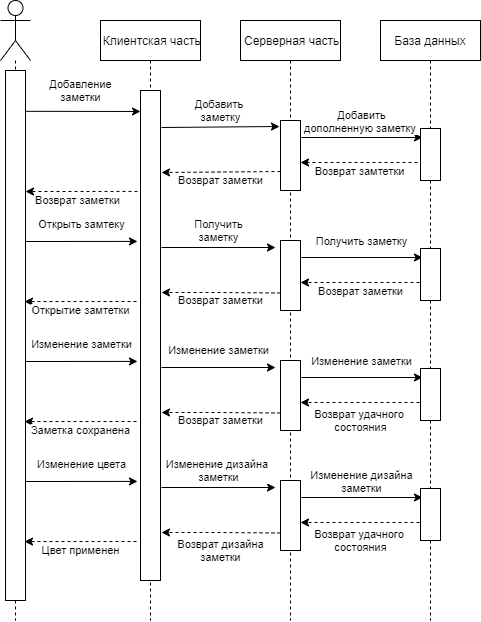


Рисунок 5.2 – Диаграмма последовательности работы с заметкой

Практически все действия в приложении будут выглядеть как обращения к серверу, к базе данных и ответ от них. Это связано с тем, что все происходит через HTTP запросы и логика работы с заметками не является сложной, поэтому чаще всего последовательность действий будут для каждого варианта использования не будут сильно разниться.

**Диаграмма развертывания.**

диаграмма развертывания (deployment diagram) предназначена для представления общей конфигурации или топологии распределенной программной системы и содержит.

Физическое представление программной системы не может быть полным, если отсутствует информация о том, на какой платформе и на каких вычислительных средствах она реализована. Если разрабатывается программа, выполняющаяся локально на компьютере пользователя и не использующая периферийных устройств и ресурсов, то в разработке дополнительных диаграмм нет необходимости. При разработке же корпоративных приложений наличие таких диаграмм может быть крайне полезным для решения задач рационального размещения компонентов в целях эффективного использования распределенных вычислительных и коммуникационных ресурсов сети, обеспечения безопасности и других.

Для представления общей конфигурации и топологии распределенной программной системы в UML предназначены диаграммы развертывания.

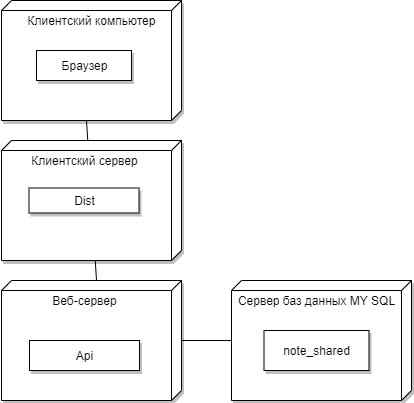


Рисунок 5.3 – Диаграмма развертывания

**Диаграмма компонентов.**

Диаграмма компонентов, в отличие от ранее рассмотренных диаграмм, описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу. Пунктирные стрелки, соединяющие модули, показывают отношения взаимозависимости, аналогичные тем, которые имеют место при компиляции исходных текстов программ. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними.

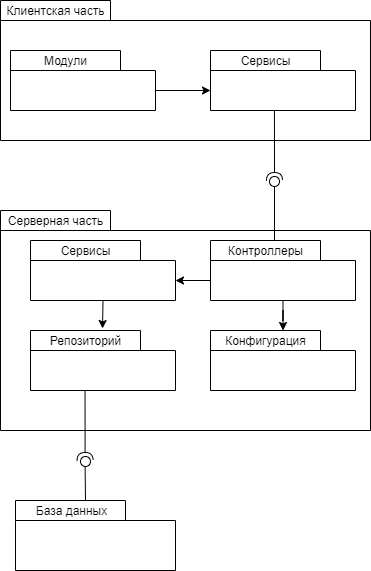


Рисунок 5.4 – Диаграмма компонентов

**Диаграмма классов.**

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами. Вид и интерпретация диаграммы классов существенно зависит от точки зрения (уровня абстракции): классы могут представлять сущности предметной области (в процессе анализа) или элементы программной системы (в процессах проектирования и реализации).

На данной диаграмме представлена диаграмма классов пакета controllers. В данном пакете все контроллеры унаследованы от классы BaseController.

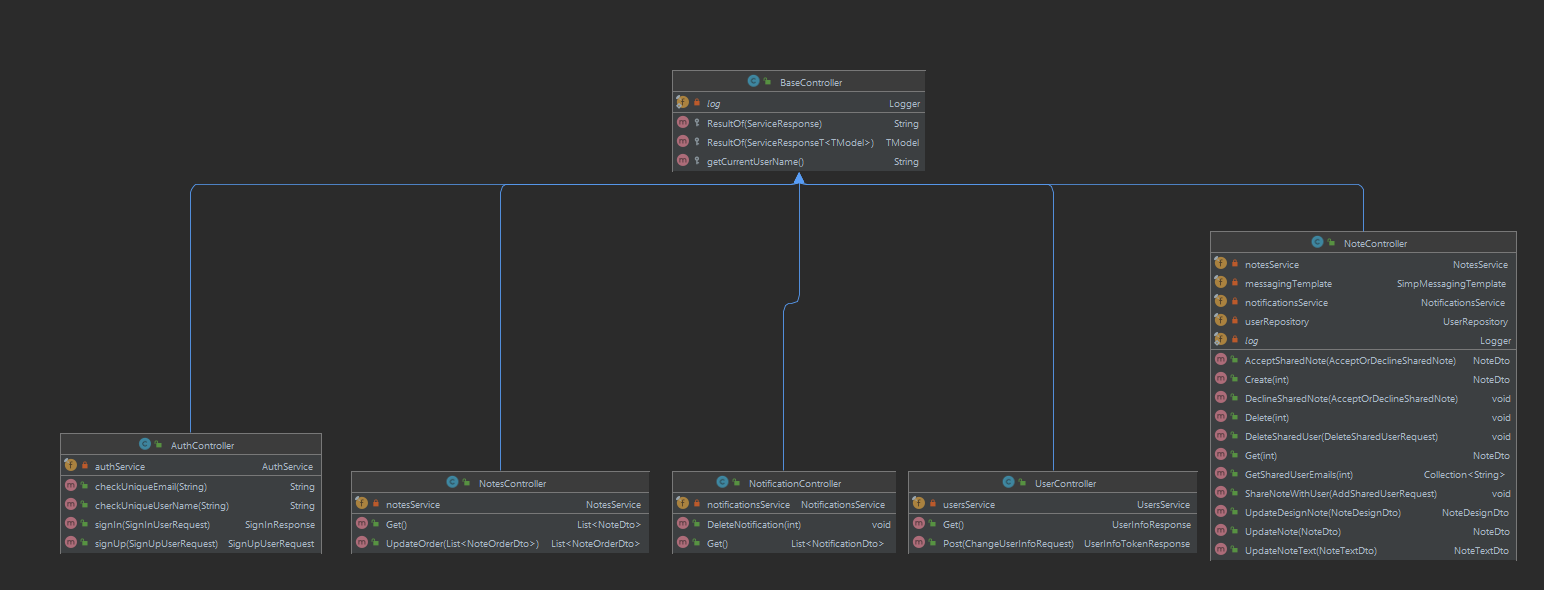
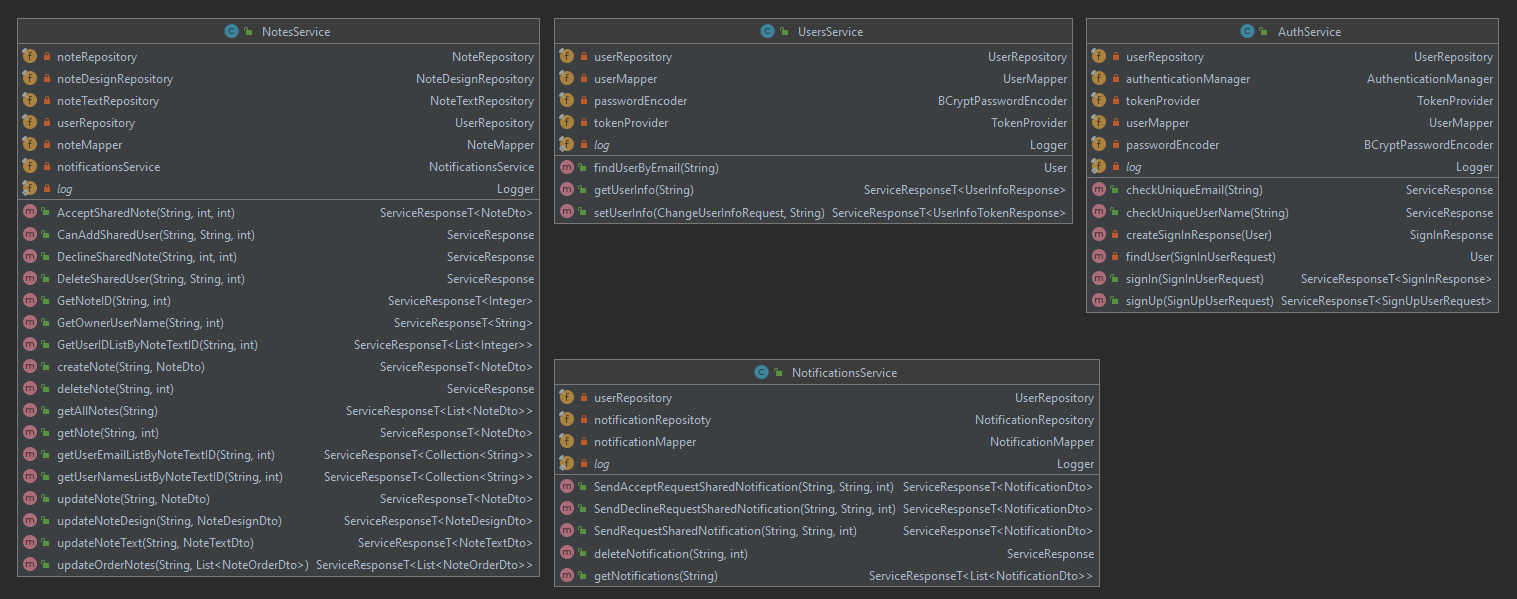
****

Рисунок 5.5 – Диаграмма классов пакета Controllers

На данной диаграмме все классы пакета Services.

Рисунок 5.6 – Диаграмма классов пакета Services

**Диаграмма состояний.**

Для большинства физических систем, кроме самых простых и тривиальных, статических представлений совершенно недостаточно для моделирования процессов функционирования подобных систем как в целом, так и их отдельных подсистем и элементов.

Диаграммы состояний являются хорошо известным средством описа­ния поведения систем. Они определяют все возможные состояния, в кото­рых может находиться конкретный объект, а также процесс смены состояний объекта в результате влияния некоторых событий.

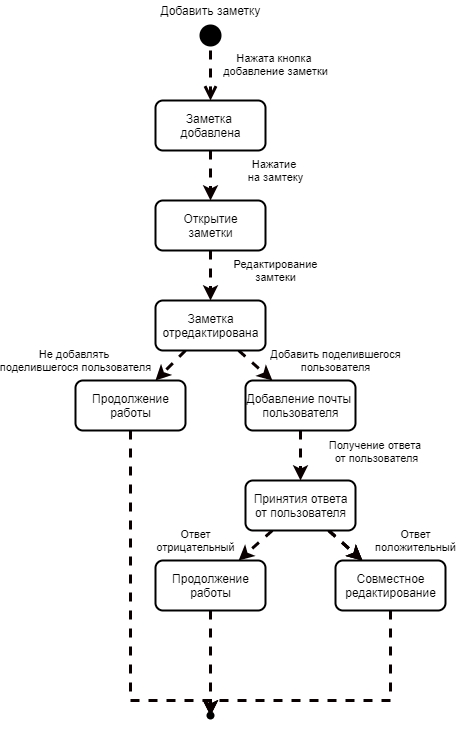
****

Рисунок 5.67 – Диаграмма состояний

Диаграмма состояние в данном случае показывает как работает последовательность работы с заметкой. Так как это веб-приложение, диаграмма не может в полной мере описать работу с приложением, так как приложение модульное и есть возможность переходить из одной части в другое без определенной последовательности.

1. **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

При запуске приложения автоматически происходит перенаправление на страницу с авторизацией. По стандарту используется светлая тема (рисунок 6.1).

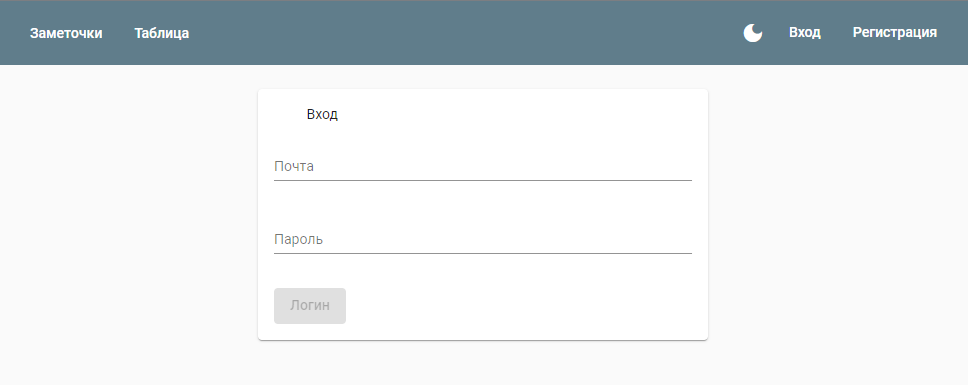


Рисунок 6.1 – Вход

Если же пользователь не зарегистрирован, то можно нажать на кнопку регистрации, где будет форма регистрации (рисунок 6.2). После регистрации пользователю нужно продублировать данные в форме входа.

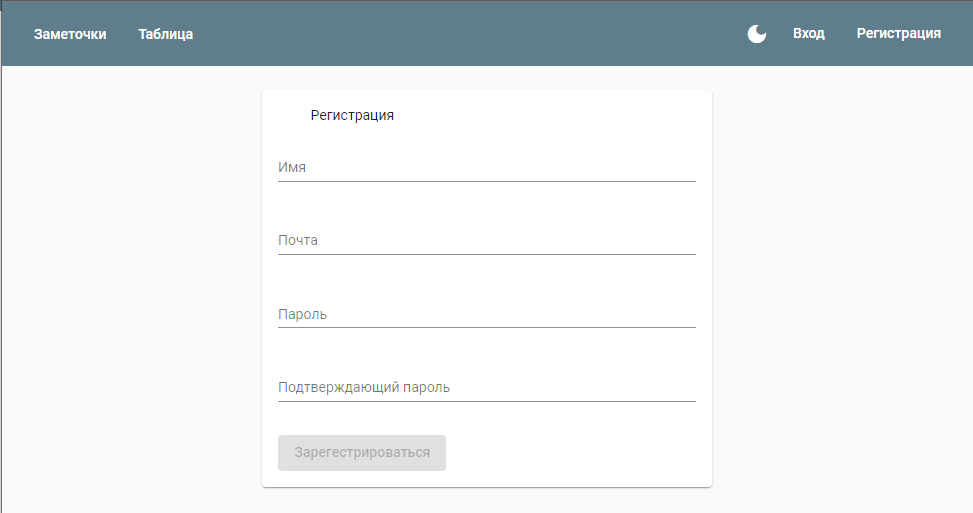


Рисунок 6.2 – Регистрация

После успешного входа в приложение пользователь попадает на главную страницу, на которой есть сетка заметок (рисунок 6.3). На данной странице доступен импорт заметок и их поиск.

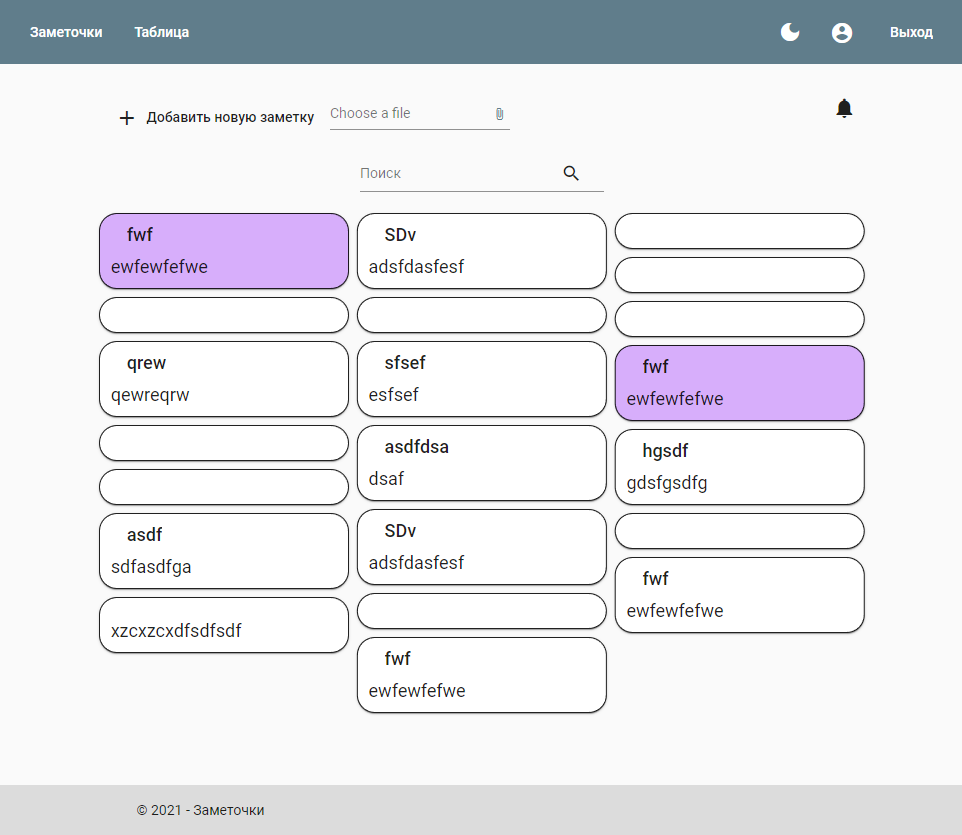


Рисунок 6.3 – Главная страница

После нажатия на несколько заметок пользователь увидит окна с этими заметками (рисунок 6.4). Функциональность в окне представлена изменением цвета, удаление, сохранением, редактированием, добавлением пользователя, просмотром на полный экран.

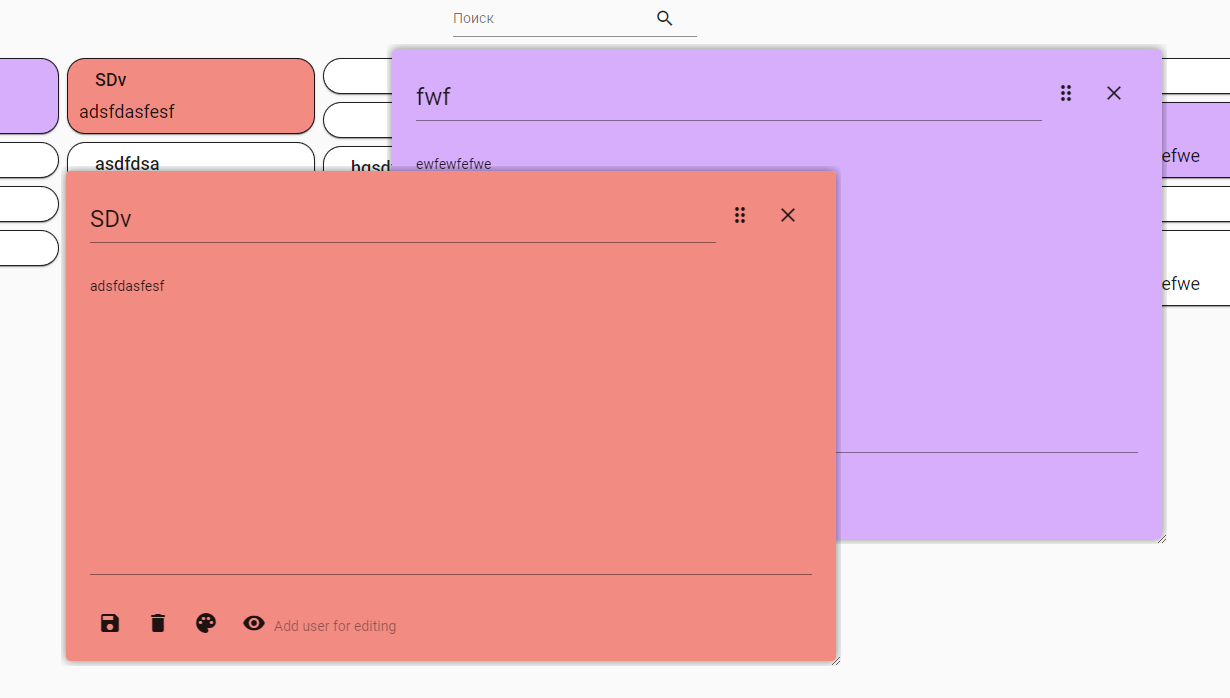


Рисунок 6.5 – Окна заметок

Также можно сменить тему на темную (рисунок 6.6).

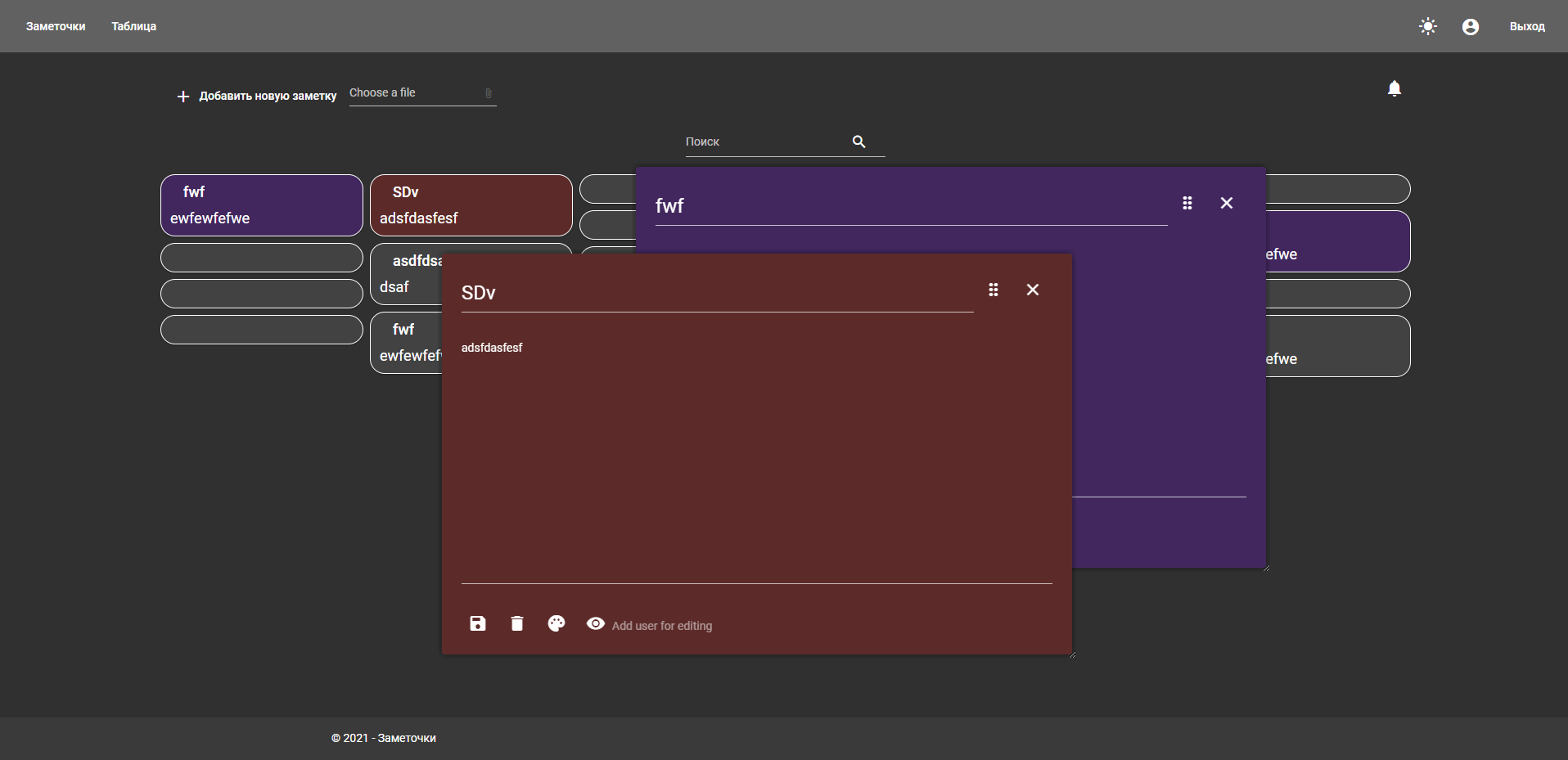


Рисунок 6.6 – Темная тема

На разных дисплеях сетка заметок будет меняться. Например, на более малых устройствах будет выглядеть как список (рисунок 6.7).

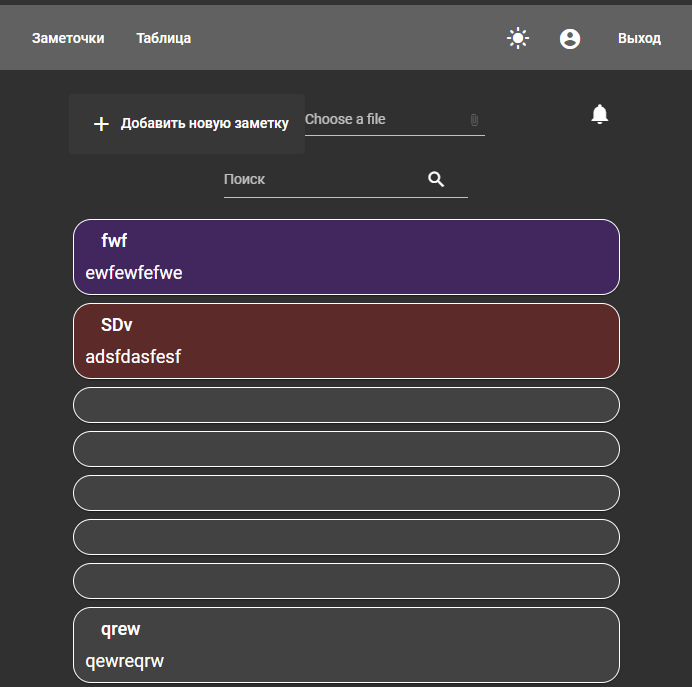


Рисунок 6.7 – Список заметок

Также для малых дисплеев заметка будет растягиваться на весь экран (рисунок 6.8).

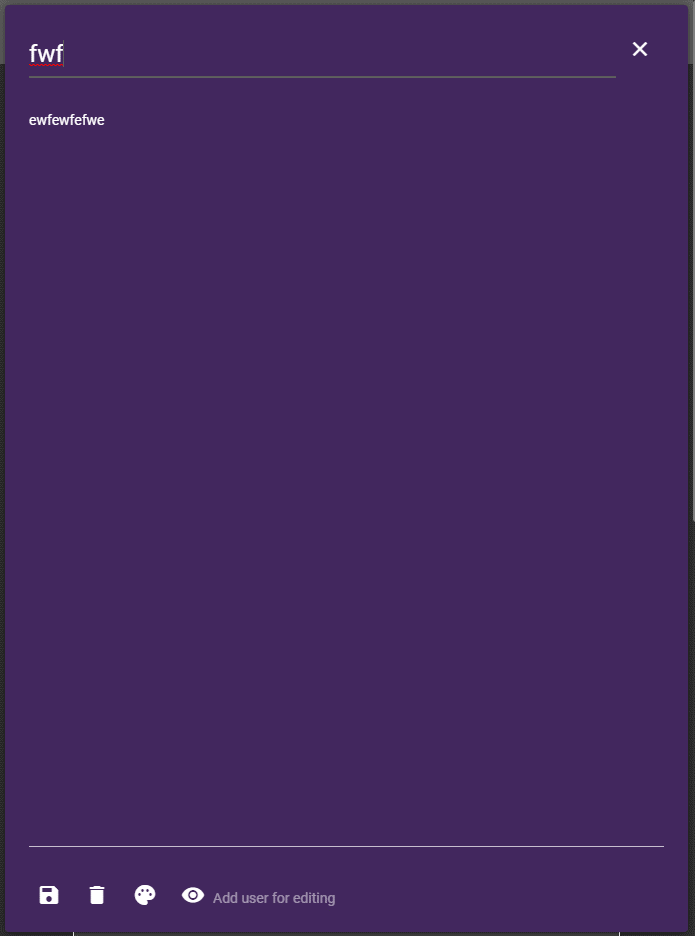


Рисунок 6.8 – Заметка на малых устройствах

На главной старице также доступна вкладка с личной информацией пользователя (рисунок 6.9).

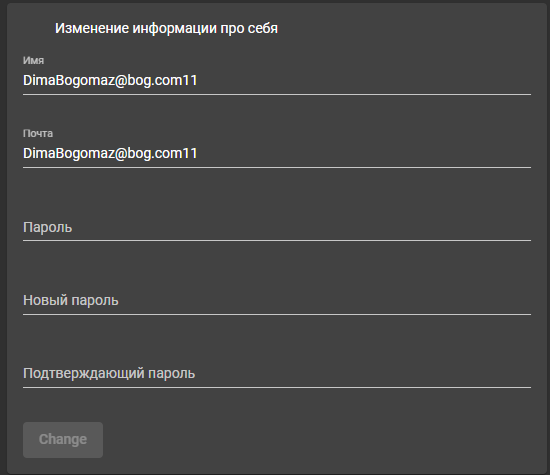


Рисунок 6.8 – Заметка на малых устройствах

Есть также в навигационной панели есть кнопка «Таблица», по нажатия на которую открывается страница со списком всех заметок, в которой предусмотрена фильтрация и сортировка (рисунок 6.9).

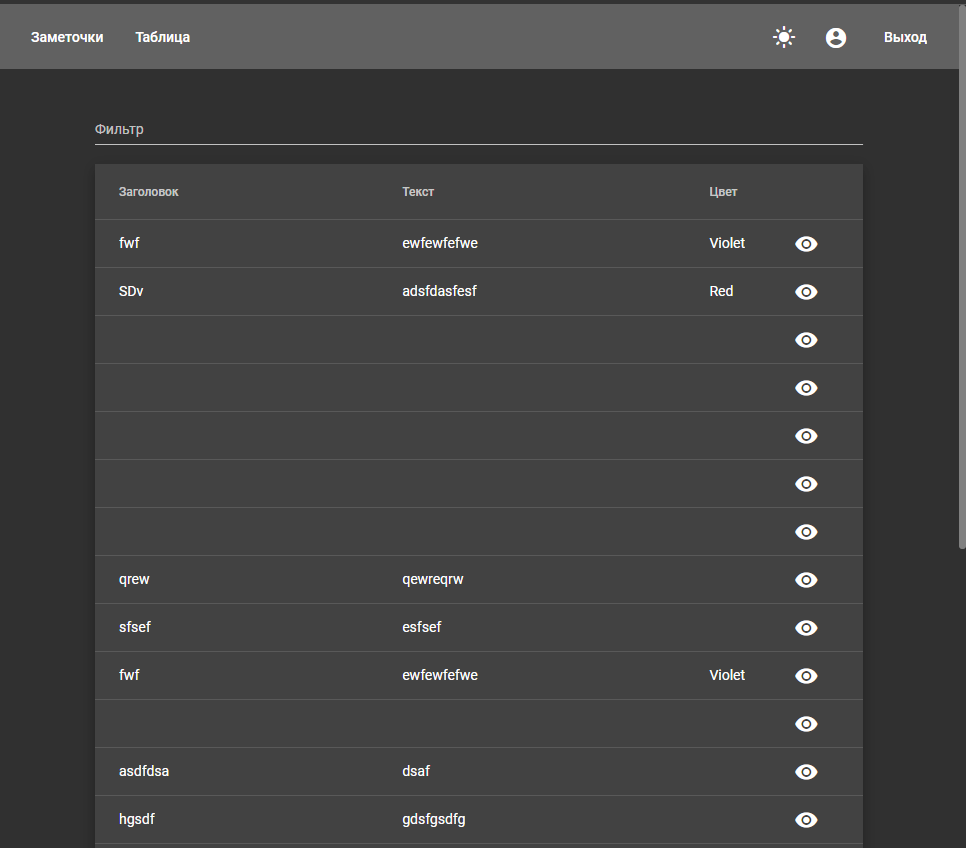


Рисунок 6.9 – Таблица с заметками

Из таблицы можно открыть конкретную заметку на весь экран, где можно поделиться заметкой со всеми пользователями, сохранить в файл саму заметку (рисунок 6.9).

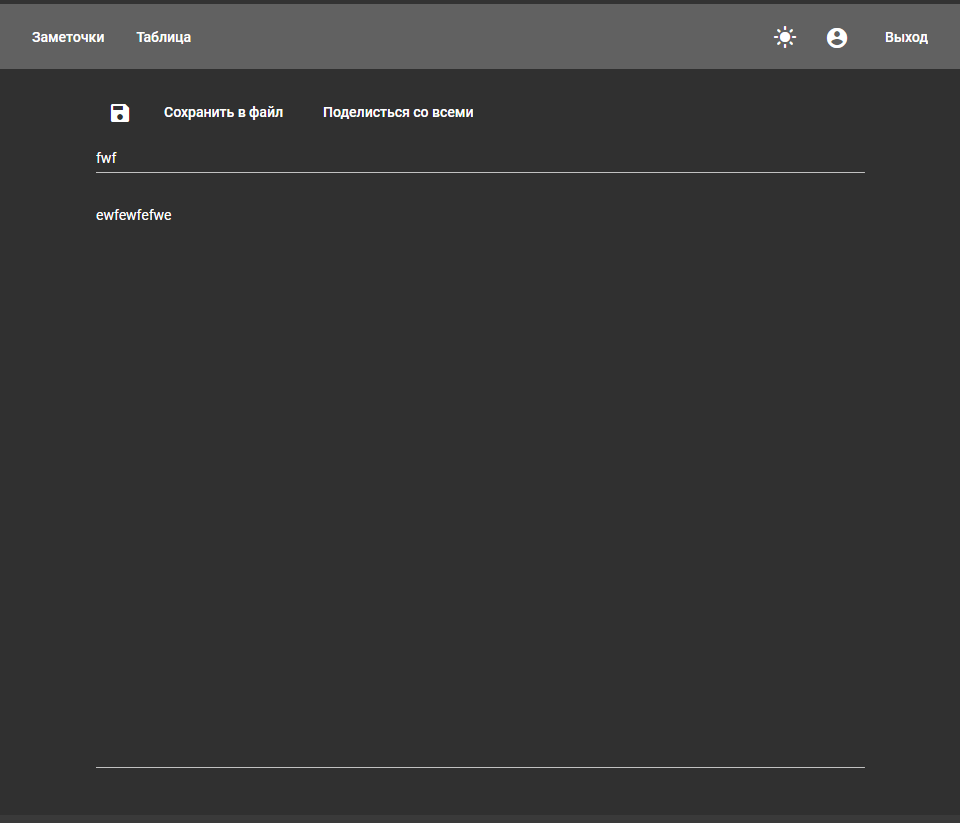


Рисунок 6.9 – Заметка на весь экран

**7 РЕЗУЛЬТАТ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ**

В системе предусмотрена обработка неправильного ввода с помощью реактивных форм. Валидация предусмотрена на формах входа, регистрации и смены данных. Предусмотрена валидация как на клиентской части приложения, так и на серверной. Кнопки в каждой форме при каких-либо ошибках блокируются. Также предусмотрен глобальный обработчик ошибок.

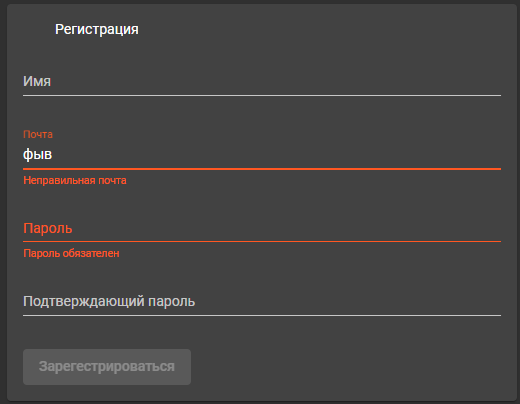


Рисунок 7.1 – Попытка регистрации

Валидация пароля также сделала таким образом, чтобы говорить клиенту о произошедших ошибках во введении данных.

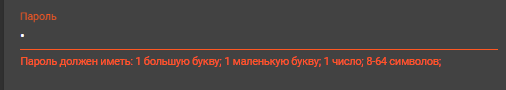


Рисунок 7.2 – Ошибки ввода пароля

Валидация уникальности почты и имени пользователя также есть на всех формах, которые есть в приложении. Во время печати происходит отправление запроса на сервер с проверкой на уникальность. После этого происходит обработка ответа сервера на стороне клиента.

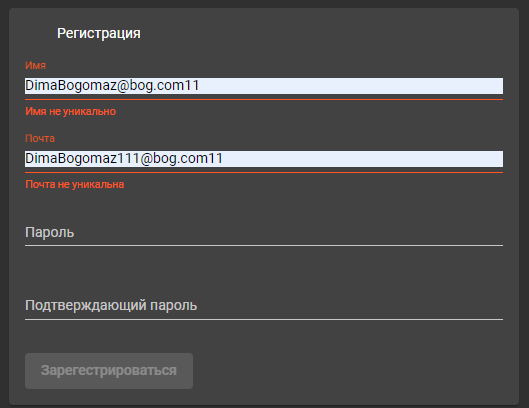


Рисунок 7.3 – Валидация уникальности имени и почты

**ЗАКЛЮЧЕНИЕ**

В ходе данной работы было создано приложение для публикации и хранения заметок. Она не является аналогом некого коммерческого проекта, так как не реализует некий уникальный функционал.

Были изучены основные принципы построения приложений, архитектурные подходы для формирования веб-приложений, учитывая особенности используемых фреймворков. Изучен функционал базы данных MySQL, подход к построению обращения к базе данных по средствам паттернов.

В данной курсовой были проведены исследования в предметной области построение систем, хранящих заметки в виде веб-приложений.

Проведен синтез и анализ структурных, функциональных и логических схем разрабатываемой системы и отдельных их элементов.

Для удобства использования нужной информации база данных разбита на несколько взаимосвязанных таблиц, что обеспечивает быстрое и эффективное их использование.

Результатом работы является рабочая система по работе с заметками.

В дальнейшем планируется усовершенствования системы в виде добавления более сложного алгоритма поиска, добавления форматирования текста, усовершенствования системы уведомления приложения.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[10] Изучение MySQL для начинающих [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://itproger.com/course/php-mysql/33>

[10] Изучение MySQL для начинающих [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://itproger.com/course/php-mysql/33>

**ПРИЛОЖЕНИЕ А**

**( обязательное )**

**Листинг основных элементов**

package com.noteshared.services;  
  
import com.noteshared.domain.entities.users.User;  
import com.noteshared.domain.entities.users.UserRepository;  
import com.noteshared.mappers.UserMapper;  
import com.noteshared.models.requests.SignInUserRequest;  
import com.noteshared.models.requests.SignUpUserRequest;  
import com.noteshared.models.responses.ServiceResponse;  
import com.noteshared.models.responses.ServiceResponseT;  
import com.noteshared.models.responses.SignInResponse;  
import com.noteshared.security.jwt.TokenProvider;  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication.BadCredentialsException;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.AuthenticationException;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.stereotype.Service;  
  
@Service  
@Slf4j  
@RequiredArgsConstructor  
public class AuthService {  
 private final UserRepository userRepository;  
 private final AuthenticationManager authenticationManager;  
 private final TokenProvider tokenProvider;  
 private final UserMapper userMapper;  
 private final BCryptPasswordEncoder passwordEncoder;  
  
 public ServiceResponseT<SignInResponse> signIn(SignInUserRequest authenticationDTO) {  
 if (authenticationDTO.getEmail() == null || authenticationDTO.getPassword() == null) {  
 return new ServiceResponseT<>("Bad credential");  
 }  
 try {  
 User user = findUser(authenticationDTO);  
 return new ServiceResponseT<>(createSignInResponse(user));  
 } catch (AuthenticationException e) {  
 return new ServiceResponseT<>("Invalid email or password");  
 }  
 }  
  
 public ServiceResponseT<SignUpUserRequest> signUp(SignUpUserRequest signUpUserRequest) {  
 if (!signUpUserRequest.getPassword().equals(signUpUserRequest.getConfirmPassword())) {  
 return new ServiceResponseT<>("password mustMatch");  
 }  
 if (signUpUserRequest.getEmail() == null || signUpUserRequest.getPassword() == null ||  
 signUpUserRequest.getConfirmPassword() == null || signUpUserRequest.getUserName() == null) {  
 throw new BadCredentialsException("Fill in required fields");  
 }  
  
 var emailUnique = checkUniqueEmail(signUpUserRequest.getEmail());  
 if (!emailUnique.isSuccess())  
 {  
 return new ServiceResponseT<>(emailUnique);  
 }  
 var userNameUnique = checkUniqueUserName(signUpUserRequest.getUserName());  
 if (!userNameUnique.isSuccess())  
 {  
 return new ServiceResponseT<>(userNameUnique);  
 }  
  
 User user = userMapper.signUpDtoToUser(signUpUserRequest);  
 user.setPassword(passwordEncoder.encode(user.getPassword()));  
 userRepository.save(user);  
 return new ServiceResponseT<>(signUpUserRequest);  
 }  
  
 public ServiceResponse checkUniqueEmail(String email)  
 {  
 if (userRepository.existsByEmail(email))  
 {  
 return new ServiceResponse("email notUnique");  
 }  
 return new ServiceResponse();  
 }  
  
 public ServiceResponse checkUniqueUserName(String userName)  
 {  
 if (userRepository.existsByUserName(userName))  
 {  
 return new ServiceResponse("userName notUnique");  
 }  
 return new ServiceResponse();  
 }  
  
 private User findUser(SignInUserRequest authenticationDto) {  
 String email = authenticationDto.getEmail();  
 String password = authenticationDto.getPassword();  
 var user = userRepository.findByEmail(email).get();  
 authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(user.getUserName(), password));  
 return user;  
 }  
  
 private SignInResponse createSignInResponse(User user) {  
 String token = tokenProvider.createToken(user.getUserName());  
 SignInResponse signInResponse = new SignInResponse();  
 signInResponse.setToken(token);  
 signInResponse.setEmail(user.getEmail());  
 return signInResponse;  
 }  
}

package com.noteshared.services;  
  
import com.noteshared.domain.entities.notedesigns.NoteDesignRepository;  
import com.noteshared.domain.entities.notes.Note;  
import com.noteshared.domain.entities.notes.NoteRepository;  
import com.noteshared.domain.entities.notes.UserRoleForNote;  
import com.noteshared.domain.entities.notetexts.NoteTextRepository;  
import com.noteshared.domain.entities.users.User;  
import com.noteshared.domain.entities.users.UserRepository;  
import com.noteshared.models.DTO.NoteDesignDto;  
import com.noteshared.models.DTO.NoteDto;  
import com.noteshared.mappers.NoteMapper;  
import com.noteshared.models.DTO.NoteOrderDto;  
import com.noteshared.models.DTO.NoteTextDto;  
import com.noteshared.models.responses.ServiceResponse;  
import com.noteshared.models.responses.ServiceResponseT;  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.messaging.simp.SimpMessagingTemplate;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.ArrayList;  
import java.util.Collection;  
import java.util.Comparator;  
import java.util.List;  
import java.util.stream.Collectors;  
  
@Service  
@Slf4j  
@RequiredArgsConstructor  
public class NotesService {  
 private final NoteRepository noteRepository;  
 private final NoteDesignRepository noteDesignRepository;  
 private final NoteTextRepository noteTextRepository;  
 private final UserRepository userRepository;  
 private final NoteMapper noteMapper;  
 private final NotificationsService notificationsService;  
  
 // OK  
 public ServiceResponseT<NoteDto> getNote(String currentUserName, int noteID) {  
 var user = userRepository.findByUserName(currentUserName).get();  
 var noteList = user.getNotes();  
 var count = noteList.stream().filter(n -> n.getId() == noteID).findFirst().stream().count();  
  
 var note = noteRepository.findById(noteID).get();  
 var noteDto = noteMapper.noteToNoteDto(note);  
 if(note.isShared() || count != 0) {  
 return new ServiceResponseT<>(noteDto);  
 }  
 else {  
 return new ServiceResponseT<>("Not allowed");  
 }  
 }  
  
 // OK  
 public ServiceResponseT<NoteDto> createNote(String currentUserName, NoteDto noteDto) {  
 var user = userRepository.findByUserName(currentUserName).get();  
 var note = noteMapper.noteDtoToNote(noteDto);  
 note.setUser(user);  
 noteRepository.save(note);  
 var newNoteDto = noteMapper.noteToNoteDto(note);  
 return new ServiceResponseT<>(newNoteDto);  
 }  
  
 @Transactional  
 public ServiceResponse deleteNote(String currentUserName, int noteID) {  
 var user = userRepository.findByUserName(currentUserName).get();  
 var noteList = user.getNotes().stream().filter(n -> n.getId() == noteID);  
 var note = noteList.findFirst().get();  
 if(note == null) {  
 return new ServiceResponse("Not allowed");  
 }  
 noteRepository.delete(note);  
 return new ServiceResponse();  
 }  
  
 public ServiceResponseT<NoteDto> updateNote(String currentUserName, NoteDto updateNoteDto) {  
 var user = userRepository.findByUserName(currentUserName).get();  
 var noteList = user.getNotes();  
  
 var note = (Note)noteList.stream().filter(n -> n.getId() == updateNoteDto.getId()).toArray()[0];  
 if(note == null) {  
 return new ServiceResponseT<>("Not allowed");  
 }  
 updateNoteDto.setUserRole(note.getUserRole());  
  
 note = noteMapper.noteDtoToNote(updateNoteDto);  
 var updatedNote = noteRepository.save(note);  
  
 var updatedNoteDto = noteMapper.noteToNoteDto(updatedNote);  
 return new ServiceResponseT<NoteDto>(updatedNoteDto);  
 }  
  
 // OK  
 public ServiceResponseT<NoteDesignDto> updateNoteDesign(String currentUserName, NoteDesignDto updateNoteDesignDto) {  
 var user = userRepository.findByUserName(currentUserName).get();  
 var noteList = user.getNotes();  
 var note = noteList.stream().filter(n -> n.getId() == updateNoteDesignDto.getNoteID()).findFirst().get();  
 if(note == null) {  
 return new ServiceResponseT<>("Not allowed");  
 }  
 var oldNoteDesign = note.getNoteDesign();  
 var newNoteDesign = noteMapper.noteDesignDtoToNoteDesign(updateNoteDesignDto);  
 if(oldNoteDesign != null) {  
 newNoteDesign.setId(oldNoteDesign.getId());  
 }  
  
 note.setNoteDesign(newNoteDesign);  
 noteRepository.save(note);  
  
 var newNoteDesignDto = noteMapper.noteDesignToNoteDesignDto(newNoteDesign);  
 return new ServiceResponseT<>(newNoteDesignDto);  
 }  
  
 // OK  
 public ServiceResponseT<Collection<String>> getUserEmailListByNoteTextID(String currentUserName, int noteTextID) {  
 var user = userRepository.findByUserName(currentUserName).get();  
 var noteList = user.getNotes();  
 var note = noteList.stream().filter(n ->n.getNoteText() != null && n.getNoteText().getId() == noteTextID)  
 .findFirst().get();  
 if(note == null) {  
 return new ServiceResponseT<>("Not allowed");  
 }  
 var noteText = noteTextRepository.findById(note.getNoteText().getId()).get();  
 var userEmails = noteRepository.findAllByNoteText(noteText).get().stream().map(n -> n.getUser().getEmail());  
 return new ServiceResponseT<>(userEmails.collect(Collectors.*toList*()));  
 }  
  
 // OK  
 public ServiceResponseT<Collection<String>> getUserNamesListByNoteTextID(String currentUserName, int noteTextID) {  
 var user = userRepository.findByUserName(currentUserName).get();  
 var noteList = user.getNotes();  
 var note = noteList.stream().filter(n ->n.getNoteText() != null && n.getNoteText().getId() == noteTextID)  
 .findFirst().get();  
 if(note == null) {  
 return new ServiceResponseT<>("Not allowed");  
 }  
 var noteText = noteTextRepository.findById(note.getNoteText().getId()).get();  
 var userNames = noteRepository.findAllByNoteText(noteText).get().stream().map(n -> n.getUser().getUserName());  
 return new ServiceResponseT<>(userNames.collect(Collectors.*toList*()));  
 }  
  
 // OK  
 public ServiceResponseT<List<NoteDto>> getAllNotes(String currentUserName) {  
 var user = userRepository.findByUserName(currentUserName).get();  
 var noteList = user.getNotes();  
 noteList.sort(Comparator.*comparing*(n1->n1.getOrder()));  
 var noteDto = noteList.stream().map(n -> noteMapper.noteToNoteDto(n));  
 return new ServiceResponseT<>(noteDto.collect(Collectors.*toList*()));  
 }  
  
 // OK  
 public ServiceResponseT<List<NoteOrderDto>> updateOrderNotes(String currentUserName, List<NoteOrderDto> notesOrder) {  
 var user = userRepository.findByUserName(currentUserName).get();  
  
 user.getNotes().stream().forEach(n -> {  
 n.setOrder(notesOrder.stream().filter(n1 -> n1.getId() == n.getId()).findFirst().get().getOrder());  
 });  
 userRepository.save(user);  
 return new ServiceResponseT<>(notesOrder);  
 }  
  
 public ServiceResponseT<NoteTextDto> updateNoteText(String currentUserName, NoteTextDto updateNoteTextDto)  
 {  
 var user = userRepository.findByUserName(currentUserName).get();  
 var noteList = user.getNotes();  
 var note = noteList.stream().filter(n -> n.getNoteText().getId() == updateNoteTextDto.getId()).findFirst().get();  
 if(note == null) {  
 return new ServiceResponseT<>("Not allowed");  
 }  
 var oldNoteText = note.getNoteText();  
 var newNoteText = noteMapper.noteTextDtoToNoteText(updateNoteTextDto);  
 newNoteText.setId(oldNoteText.getId());  
  
 note.setNoteText(newNoteText);  
 noteRepository.save(note);  
 var newNoteTextDto = noteMapper.noteTextToNoteTextDto(newNoteText);  
 return new ServiceResponseT<>(newNoteTextDto);  
 }  
  
 public ServiceResponseT<List<Integer>> GetUserIDListByNoteTextID(String currentUserName, int noteTextID)  
 {  
 var user = userRepository.findByUserName(currentUserName).get();  
 var noteList = user.getNotes();  
 var note = noteList.stream().filter(n -> n.getNoteText().getId() == noteTextID).findFirst().get();  
 if(note == null) {  
 return new ServiceResponseT<>("Not allowed");  
 }  
 var userIdList = note.getNoteText().getNotes().stream().map(n -> n.getUser().getId());  
 return new ServiceResponseT<>(userIdList.collect(Collectors.*toList*()));  
 }  
  
 public ServiceResponse CanAddSharedUser(String currentUserName, String sharedUserEmail, int noteTextID) {  
 var user = userRepository.findByUserName(currentUserName).get();  
 var noteText = noteTextRepository.findById(noteTextID).get();  
 var note = noteRepository.findAllByNoteTextAndAndUser(noteText, user).get().stream().findFirst().get();  
 if (note != null && note.getUserRole() != UserRoleForNote.*Owner*)  
 {  
 return new ServiceResponse("Don't has access");  
 }  
 var sharedUser = userRepository.findByEmail(sharedUserEmail);  
 if (sharedUser.isEmpty())  
 {  
 return new ServiceResponse("User is not found");  
 }  
 var dnote = noteText.getNotes()  
 .stream().filter(n -> n.getUser().getEmail() == sharedUserEmail)  
 .findAny();  
 if (!dnote.isEmpty())  
 {  
 return new ServiceResponse("Shared user also has access");  
 }  
 return new ServiceResponse();  
 }  
  
 public ServiceResponse DeleteSharedUser(String currentUserName, String sharedUserEmail, int noteTextID)  
 {  
 var note = userRepository.findByUserName(currentUserName).get()  
 .getNotes().stream()  
 .filter(n -> n.getNoteText().getId() == noteTextID)  
 .findFirst().get();  
 if (note.getUserRole() != UserRoleForNote.*Owner*)  
 {  
 return new ServiceResponse("Don't has access");  
 }  
  
 var sharedUser = userRepository.findByEmail(sharedUserEmail).get();  
 if (sharedUser == null)  
 {  
 return new ServiceResponse();  
 }  
  
 var deleteNote = sharedUser.getNotes().stream().filter(not -> not.getNoteText().getId() == noteTextID).findFirst().get();  
 noteRepository.delete(deleteNote);  
  
 return new ServiceResponse();  
 }  
  
 public ServiceResponseT<NoteDto> AcceptSharedNote(String userName, int noteTextID, int notificationID)  
 {  
 int order = -100;  
 var user = userRepository.findByUserName(userName).get();  
  
 var newNote = new Note();  
 newNote.setOrder(order); newNote.setUser(user); newNote.setNoteText(noteTextRepository.getById(noteTextID)); newNote.setUserRole(UserRoleForNote.*Editor*);  
 noteRepository.save(newNote);  
  
 notificationsService.deleteNotification(userName, notificationID);  
  
 return new ServiceResponseT<>(noteMapper.noteToNoteDto(newNote));  
 }  
  
 public ServiceResponse DeclineSharedNote(String userName, int noteTextID, int notificationID)  
 {  
 var result = notificationsService.deleteNotification(userName, notificationID);  
 return result;  
 }  
  
 public ServiceResponseT<String> GetOwnerUserName(String userName, int noteTextID)  
 {  
 var note = userRepository.findByUserName(userName).get()  
 .getNotes().stream()  
 .filter(n -> n.getNoteText().getId() == noteTextID)  
 .findFirst().get();  
 var noteText = noteTextRepository.findById(note.getNoteText().getId()).get();  
 var resultNoteID = GetNoteID(userName, noteTextID);  
 if (!resultNoteID.isSuccess())  
 {  
 return new ServiceResponseT<>(resultNoteID);  
 }  
 note = noteRepository.findAllByNoteTextAndAndUserRole(noteText, UserRoleForNote.*Owner*).get().stream().findFirst().get();  
 if (note == null)  
 {  
 return new ServiceResponseT<>("Note is not found", true);  
 }  
 return new ServiceResponseT<>(note.getUser().getUserName(), true);  
 }  
  
 public ServiceResponseT<Integer> GetNoteID(String userName, int noteTextID)  
 {  
 var note = userRepository.findByUserName(userName).get()  
 .getNotes().stream()  
 .filter(n -> n.getNoteText().getId() == noteTextID)  
 .findFirst().get();  
 if (note == null)  
 {  
 return new ServiceResponseT<>("Note is not found");  
 }  
 return new ServiceResponseT<>(note.getId());  
 }  
}

package com.noteshared.services;  
  
import com.fasterxml.jackson.core.JsonProcessingException;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.noteshared.domain.entities.notifications.Notification;  
import com.noteshared.domain.entities.notifications.NotificationRepository;  
import com.noteshared.domain.entities.notifications.NotificationType;  
import com.noteshared.domain.entities.users.UserRepository;  
import com.noteshared.mappers.NotificationMapper;  
import com.noteshared.models.DTO.AcceptedRequestSharedNoteNotificationContent;  
import com.noteshared.models.DTO.DeclinedRequestSharedNoteNotificationContent;  
import com.noteshared.models.DTO.NotificationDto;  
import com.noteshared.models.DTO.RequestSharedNoteNotificationContent;  
import com.noteshared.models.responses.ServiceResponse;  
import com.noteshared.models.responses.ServiceResponseT;  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.sql.Date;  
import java.util.List;  
import java.util.stream.Collectors;  
  
@Service  
@Slf4j  
@RequiredArgsConstructor  
public class NotificationsService {  
  
 private final UserRepository userRepository;  
 private final NotificationRepository notificationRepositoty;  
 private final NotificationMapper notificationMapper;  
  
 public ServiceResponseT<List<NotificationDto>> getNotifications(String currentUserName) {  
 var user = userRepository.findByUserName(currentUserName).get();  
  
 var notificationDtoList = user.getNotifications()  
 .stream()  
 .map(n -> notificationMapper.notificationToNotificationDto(n))  
 .collect(Collectors.*toList*());  
 return new ServiceResponseT<>(notificationDtoList);  
 }  
  
 @Transactional  
 public ServiceResponse deleteNotification(String currentUserName, int notificationID) {  
 var user = userRepository.findByUserName(currentUserName).get();  
  
 var notification = user.getNotifications().stream().filter(n -> n.getId() == notificationID).findFirst().get();  
 if(notification == null) {  
 return new ServiceResponse("Not allowed");  
 }  
 notificationRepositoty.delete(notification);  
 return new ServiceResponse();  
 }  
  
  
 public ServiceResponseT<NotificationDto> SendRequestSharedNotification(String currentUserName, String sharedUserEmail, int noteTextID) throws JsonProcessingException {  
 var sharedUser = userRepository.findByEmail(sharedUserEmail).get();  
 var currentUser = userRepository.findByUserName(currentUserName).get();  
  
 var notificationContent = new RequestSharedNoteNotificationContent();  
 notificationContent.setNoteTextID(noteTextID);  
 notificationContent.setFromUserEmail(currentUser.getEmail());  
  
 var notification = new Notification();  
 notification.setUser(sharedUser);  
 var objectMapper = new ObjectMapper();  
 notification.setContent(objectMapper.writeValueAsString(notificationContent));;  
 notification.setType(NotificationType.*RequestSharedNoteType*);  
 notification.setCreateDateTime(new Date(System.*currentTimeMillis*()));  
  
 notification = notificationRepositoty.save(notification);  
  
 var notificationDto = notificationMapper.notificationToNotificationDto(notification);  
 return new ServiceResponseT<NotificationDto>(notificationDto);  
 }  
  
 public ServiceResponseT<NotificationDto> SendAcceptRequestSharedNotification(String currentUserName, String ownerUserName, int noteTextID) throws JsonProcessingException {  
 var ownerUser = userRepository.findByUserName(ownerUserName).get();  
 var currentUser = userRepository.findByUserName(currentUserName).get();  
  
 var notificationContent = new AcceptedRequestSharedNoteNotificationContent();  
 notificationContent.setNoteTextID(noteTextID);  
 notificationContent.setFromUserEmail(currentUser.getEmail());  
  
 var notification = new Notification();  
 notification.setUser(ownerUser);  
 var objectMapper = new ObjectMapper();  
 notification.setContent(objectMapper.writeValueAsString(notificationContent));;  
 notification.setType(NotificationType.*AcceptedRequestSharedNoteType*);  
 notification.setCreateDateTime(new Date(System.*currentTimeMillis*()));  
  
 notification = notificationRepositoty.save(notification);  
  
 var notificationDto = notificationMapper.notificationToNotificationDto(notification);  
 return new ServiceResponseT<>(notificationDto);  
 }  
  
 public ServiceResponseT<NotificationDto> SendDeclineRequestSharedNotification(String currentUserName, String ownerUserName, int noteTextID) throws JsonProcessingException {  
 var ownerUser = userRepository.findByUserName(ownerUserName).get();  
 var currentUser = userRepository.findByUserName(currentUserName).get();  
  
 var notificationContent = new DeclinedRequestSharedNoteNotificationContent();  
 notificationContent.setNoteTextID(noteTextID);  
 notificationContent.setFromUserEmail(currentUser.getEmail());  
  
 var notification = new Notification();  
 notification.setUser(ownerUser);  
 var objectMapper = new ObjectMapper();  
 notification.setContent(objectMapper.writeValueAsString(notificationContent));;  
 notification.setType(NotificationType.*DeclinedRequestSharedNoteType*);  
 notification.setCreateDateTime(new Date(System.*currentTimeMillis*()));  
  
 notification = notificationRepositoty.save(notification);  
  
 var notificationDto = notificationMapper.notificationToNotificationDto(notification);  
 return new ServiceResponseT<NotificationDto>(notificationDto);  
 }  
}

package com.noteshared.services;  
  
import com.noteshared.domain.entities.users.User;  
import com.noteshared.domain.entities.users.UserRepository;  
import com.noteshared.mappers.UserMapper;  
import com.noteshared.models.requests.ChangeUserInfoRequest;  
import com.noteshared.models.responses.ServiceResponseT;  
import com.noteshared.models.responses.UserInfoResponse;  
import com.noteshared.models.responses.UserInfoTokenResponse;  
import com.noteshared.security.jwt.TokenProvider;  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.http.HttpStatus;  
import org.springframework.security.authentication.BadCredentialsException;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.Optional;  
  
@Service  
@Slf4j  
@RequiredArgsConstructor  
public class UsersService {  
 private final UserRepository userRepository;  
 private final UserMapper userMapper;  
 private final BCryptPasswordEncoder passwordEncoder;  
 private final TokenProvider tokenProvider;  
  
 @Transactional  
 public User findUserByEmail(String email) {  
 if (email == null || email.isBlank()) {  
 throw new BadCredentialsException(HttpStatus.*BAD\_REQUEST* +  
 String.*format*(" email cannot be null or empty"));  
 }  
 User result;  
 Optional<User> optionalUser = userRepository.findByEmail(email);  
 if (optionalUser.isPresent()) {  
 result = optionalUser.get();  
 } else {  
 throw new BadCredentialsException(HttpStatus.*BAD\_REQUEST* +  
 String.*format*(" User with email %s do not exist", email));  
 }  
 return result;  
 }  
  
 public ServiceResponseT<UserInfoResponse> getUserInfo(String userName)  
 {  
 var user = userRepository.findByUserName(userName);  
 if (user == null)  
 {  
 return new ServiceResponseT<UserInfoResponse>("user notFound");  
 }  
 UserInfoResponse userInfo = userMapper.userToUserInfoResponse(user.get());  
 return new ServiceResponseT<UserInfoResponse>(userInfo);  
 }  
  
 public ServiceResponseT<UserInfoTokenResponse> setUserInfo(ChangeUserInfoRequest userInfo, String userName) {  
 var userOptional = userRepository.findByUserName(userName);  
 if (userOptional == null)  
 {  
 return new ServiceResponseT<>("user notFound");  
 }  
 if (userInfo.getNewPassword() != null &&  
 userInfo.getConfirmNewPassword() != null &&  
 !userInfo.getNewPassword().equals(userInfo.getConfirmNewPassword())  
 ){  
 return new ServiceResponseT<>("newPassword mustMatch");  
 }  
 if(userInfo.getUserName() == null) {  
 return new ServiceResponseT<>("userName empty");  
 }  
 if(userInfo.getEmail() == null) {  
 return new ServiceResponseT<>("email empty");  
 }  
 var user = userOptional.get();  
 if(passwordEncoder.encode(userInfo.getOldPassword()).equals(user.getPassword())) {  
 return new ServiceResponseT<>("oldPassword mustMatch");  
 }  
 user.setUserName(userInfo.getUserName());  
 if(userInfo.getNewPassword() != null) {  
 user.setPassword(passwordEncoder.encode(userInfo.getNewPassword()));  
 }  
 user.setEmail(userInfo.getEmail());  
 userRepository.save(user);  
 String token = tokenProvider.createToken(user.getUserName());  
 var resultUserInfoToken = userMapper.userToUserInfoTokenResponse(user);  
 resultUserInfoToken.setToken(token);  
 return new ServiceResponseT<>(resultUserInfoToken);  
 }  
}

# ПРИЛОЖЕНИЕ В

**(обязательное)  
    Графический материал**

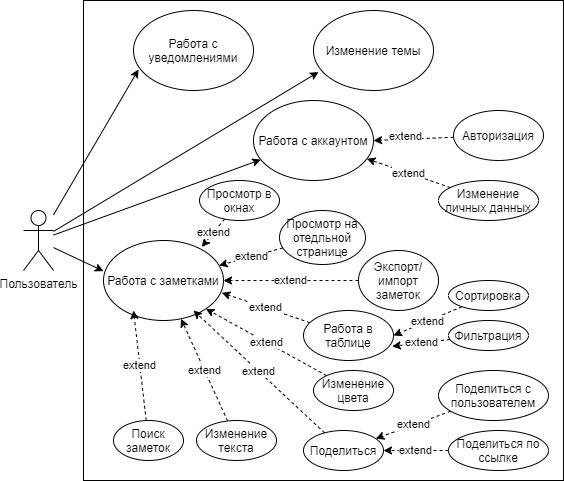
****

Рисунок В.1 – Диаграмма вариантов использования

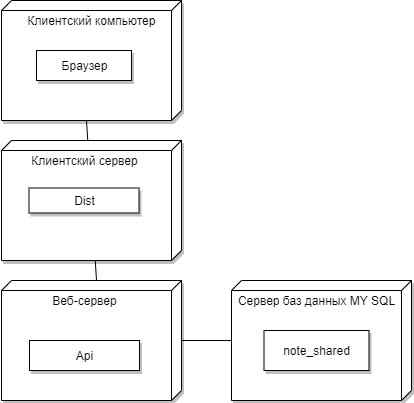
****

Рисунок В.2 – Диаграмма развертывания

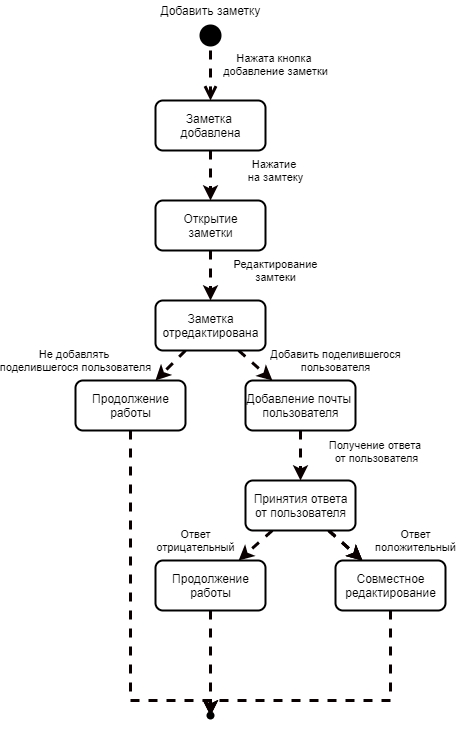
****

Рисунок В.3 – Диаграмма состояний