**ВВЕДЕНИЕ**

В мире объем информации, с которой сталкиваются люди и организации, стремительно растет. Это касается как личных заметок и списков дел, так и сложных рабочих процессов, включающих управление знаниями, проектами и задачами. Эти данные представлены в разных форматах, хранятся в различных местах и зачастую неудобны для структурирования и запоминания. Для эффективной работы существуют различные решения, но каждое из них имеет свои ограничения.

На протяжении долгого времени люди использовали физические носители для ведения записей и организации информации. Например: бумажные блокноты, ежедневники, папки с документами. Однако такие методы имеют значительные недостатки.

С развитием технологий стали популярными цифровые средства организации информации, включая текстовые редакторы, облачные хранилища и специализированные приложения. Наиболее распространенные категории таких решений: приложения для заметок, системы управления проектами, облачные сервисы.

Несмотря на разнообразие решений, пользователи часто сталкиваются с необходимостью комбинировать несколько инструментов, что усложняет процесс работы с информацией и снижает продуктивность. Существующие инструменты либо узко специализированы, либо требуют сложной интеграции для выполнения различных задач.

Целью данной дипломной работы станет разработка платформы, которая решит многие проблемы неудобной работы с данными, поможет пользователям организовывать и структурировать информацию, улучшать продуктивность и оптимизировать рабочие процессы.

Для достижения цели данного дипломного проекта можно выделить следующие задачи:

* проанализировать аналоги и существующие решения;
* выбор и обоснование технологий и инструментов для реализации проекта;
* определение основных функциональных требований и пользовательских сценариев;
* спроектировать архитектуру приложения и взаимосвязей модулей;
* спроектировать базы данных и соответствующие сервисы;
* написать программное обеспечение;
* разработать методику испытаний и протестировать программное обеспечение;
* написать подробное руководство пользователя.

Данный дипломный проект выполнен мной лично, проверен на заимствования, процент оригинальности составляет ХХ% (отчет о проверке на заимствования прилагается).

**1 ОБЗОР ЛИТЕРАТУРЫ**

В данном разделе будут описаны существующие аналоги используемые технологии. Также будет изложен основной теоретический материал, который необходим для понимания данного дипломного проекта.

**1.1 Обзор аналогов и существующих решений**

В начале проектирования приложения хорошим решением будет начать с обзора и анализа существующих аналогов, чтобы выявить их преимущества и недостатки. На основе выводов можно выявить ключевой функционал и сделать свое приложение конкурентоспособным и актуальным.

**1.1.1**Приложение Obsidian.Obsidian − приложение персональной базы знаний, которое служит инструментом для создания и управления заметками [2]. Оно приобрело популярность благодаря своей гибкости, возможностям персонализации и поддержке взаимосвязанных заметок.

Ключевыми особенностями Obsidian являются:

* полный контроль над файлами;
* возможность работы офлайн;
* безопасность информации;
* быстрый поиск и навигация;
* поддержка плагинов;

Приложение Obsidian сохраняет всю информацию локально на устройстве и использует упрощённый язык разметки Markdown в качестве основного инструмента для работы с файлами. Такой подход обеспечивает полный контроль над данными и гарантирует, что информация не попадёт к третьим лицам. Однако это накладывает определённые требования на пользователя − необходимо умение работать с Markdown-файлами, что может сделать приложение менее удобным для новичков.

Кроме того, Obsidian не предоставляет собственного облачного хранилища. Для синхронизации заметок между несколькими устройствами пользователю придётся использовать сторонние сервисы.

Можно выделить основные недостатки данного приложения:

* для синхронизации данных между устройствами требуется ручная настройка;
* отсутствие продвинутых инструментов для работы с данными;
* новичкам может потребоваться время для освоения всех возможностей.

**1.1.2**Приложение Notion. На данный момент главным конкурентом является приложение Notion. Эта платформа для ведения заметок, управления знаниями проектами и задачами [3]. Она популярна среди свободных работников, малых бизнесов, крупных компаний и студентов.

Главной особенностью Notion является её богатый набор инструментов и возможность интеграции с такими сервисами, как Google Calendar, Slack, Trello и другие.

Платформа Notion позволяет создавать текстовые документы, списки задач, таблицы, календари, галереи изображений и другие элементы, которые можно комбинировать для создания гибких рабочих пространств. Также доступна возможность создания баз данных с фильтрацией, сортировкой и различными вариантами отображения. Для упрощения работы предоставляется множество готовых шаблонов, подходящих как для личных, так и для рабочих задач, таких как планирование, бюджетирование и управление проектами.

Весь функционал возможен с совместным редактированием документов в реальном времени.

У Notion мало недостатков, которые смогли бы сделать платформу неконкурентоспособной в каком-то направлении. Но можно выделить несколько минусов у приложения:

* ограничения оффлайн-доступа;
* сложность освоения;
* производительность при работе с большими базами данных;
* ограничения использования функционала.

**1.1.3**Приложение Miro.Miro − это цифровая платформа рабочего пространства, в котором команды управляют проектами, разрабатывают продукты и создают карты мыслей [4].

Это интересное решение, непохожее на остальные. Приложение представляет собой виртуальную интерактивную доску. Оно позволяет пользователям создавать визуальные схемы, диаграммы, текстовые блоки и многое другое. Самое главное отличие в том, что пользователи не привязаны строгой структуре страниц и могут использовать все неограниченное пространство доски.

Платформа является прекрасным решением для совещаний, для визуализации сложных схем, а также для совместной работы над проектами.

Несмотря на оригинальность, Miro имеет минусы, которые могут ограничить его использование:

* на больших досках с множеством объектов приложение может работать медленно;
* бесплатная версия ограничивает количество досок и функций;
* miro имеет достаточно большой функционал, и новому пользователю может понадобиться время;

**1.2 Файловые хранилища**

Обязательным элементом для приложений, которые работают с данными, являются файловые хранилища.

Системы хранения данных делят на три основных способа:

* файловые системы;
* блочные системы;
* объектные системы.

Для проектирования своего приложения необходимо понимать преимущества и недостатки каждого решения, а также основные принципы его функционирования.

Файловые системы управляют данными, организуя их в виде файлов и директорий, предоставляя пользователю привычный интерфейс работы с информацией. Доступ к файлам осуществляется по идентификатору, который включает имя сервера, путь к каталогу и имя файла. На низком уровне используется блочный метод представления информации.

Блочное хранилище предоставляет низкоуровневый доступ к данным, разбивая данные на блоки. Блок данных − это минимальная единица хранения, содержащая часть информации, которая может быть записана, прочитана или изменена независимо от других блоков. Блоки имеют фиксированный размер и хранятся в произвольных местах на диске. Блочное хранилище предоставляет приложениям доступ к этим блокам без информации о файловой системе, позволяя операционной системе или базе данных управлять их структурированием и организацией. Файлы состоят из конечного числа блоков. Серверная операционная система назначает каждому блоку данных уникальный идентификатор расположения, позволяющий быстро находить его. Благодаря этому блочные системы хранения обеспечивают высокую скорость доступа к данным.

Объектное хранилище организует данные в виде объектов. Объекты представляют собой самостоятельные единицы данных, которые сохраняются без строгой структуры или иерархии. Каждый объект содержит сами данные, метаданные с описательной информацией и уникальный идентификатор. Системное программное обеспечение использует эти характеристики для поиска и доступа к объектам. Благодаря такому подходу объектное хранилище обеспечивает высокую масштабируемость, так как данные могут распределяться по множеству серверов или узлов.

На рисунке 1.1 изображены основные типы хранилищ.

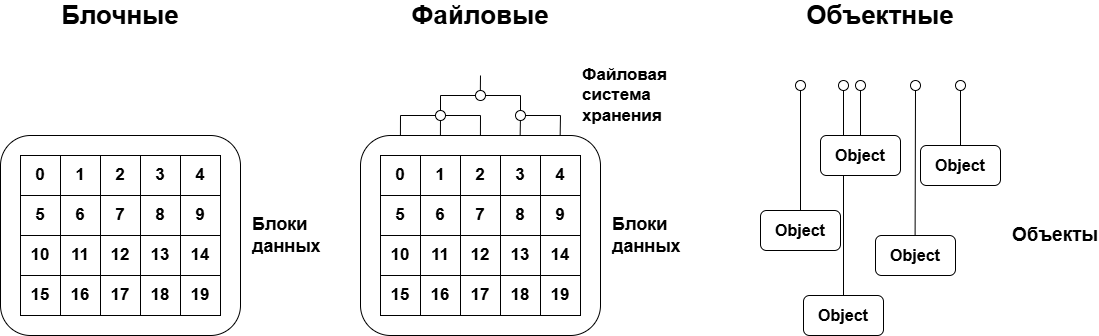


Рисунок 1.1 – Основные типы хранилищ

Для лучшего понимания следует проанализировать и сравнить хранилища. Также определить основные сценарии использования. Результат анализа представлен в таблице 1.1.

Таблица 1.1 – Сравнения типов хранилищ

|  |  |  |  |
| --- | --- | --- | --- |
| Характеристики | Файловые | Блочные | Объектные |
| Относительная стоимость | Средняя | Высокая | Низкая |
| Быстродействие | Высокое | Очень высокое | Среднее |
| Масштабируемость | Ограниченная | Средняя | Высокая |
| Совместимость с облачными технологиями | Средняя | Низкая | Высокая |
| Гибкость управления данными | Высокая | Низкая | Высокая |

По таблице 1.1 можно сказать, к каким сценариям подходят данные типы хранилищ.

Файловые хранилища удобны для традиционных систем хранения данных, но обладают ограниченной масштабируемостью. Они подходят для хранения общего пользовательского контента и веб-контента.

Блочные хранилища оптимальны для высокопроизводительных задач, таких как базы данных и виртуализация. Однако это дорогостоящее решение, которое также слабо совместимо с облачными технологиями.

Объектные хранилища являются лучшим выбором для облачных и распределенных систем, но они менее производительны при частом доступе к данным.

**1.3 Серверная разработка**

Для разработки приложения необходимо понимать современные подходы к проектированию и советующие методы.

**1.3.1**Асинхронные и синхронные операции. Популярным подходом к построению систем являются использование асинхронной модели ввода-вывода.

В синхронных операциях каждая задача выполняется последовательно. Это означает, что система ждет завершения текущей задачи, прежде чем перейти к следующей. Например, если сервер обрабатывает запросы синхронно, он будет ждать, пока не завершит обработку одного запроса, прежде чем начать обработку следующего. При увеличении нагрузки система может быстро исчерпать ресурсы, что приведет к замедлению или сбоям.

В асинхронных операциях задачи выполняются независимо друг от друга. Система не ждет завершения одной задачи, чтобы начать другую. Вместо этого она может переключаться между задачами, выполняя их параллельно или в фоновом режиме.

На рисунке 1.2 изображен рабочий процесс асинхронной модели.

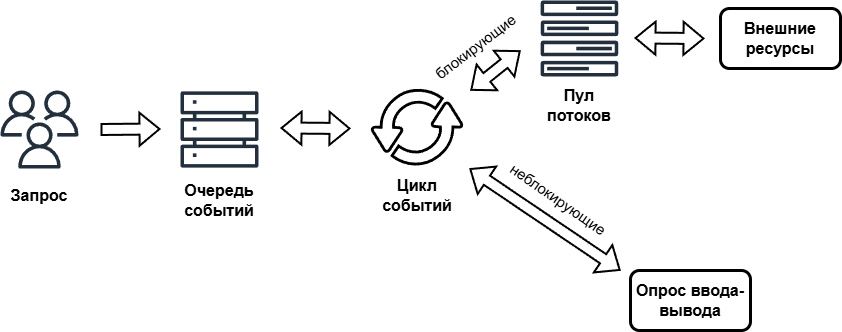


Рисунок 1.2 – Рабочий процесс асинхронной модели

Асинхронные операции не обязательно делают каждую отдельную задачу быстрее, но они позволяют системе эффективнее использовать ресурсы и обрабатывать больше задач одновременно. Это повышает стабильность и предсказуемость системы, особенно под высокой нагрузкой.

**1.3.2**Современная архитектура серверных приложений.Доминирующей архитектурой является его многоуровневая архитектура, которая позволяет разработчикам структурировать код в виде модулей, контроллеров и провайдеров. Это способствует поддержанию чистоты кода, его повторному использованию и легкости тестирования. На рисунке 1.3 изображена упрощённая многоуровневая архитектура.

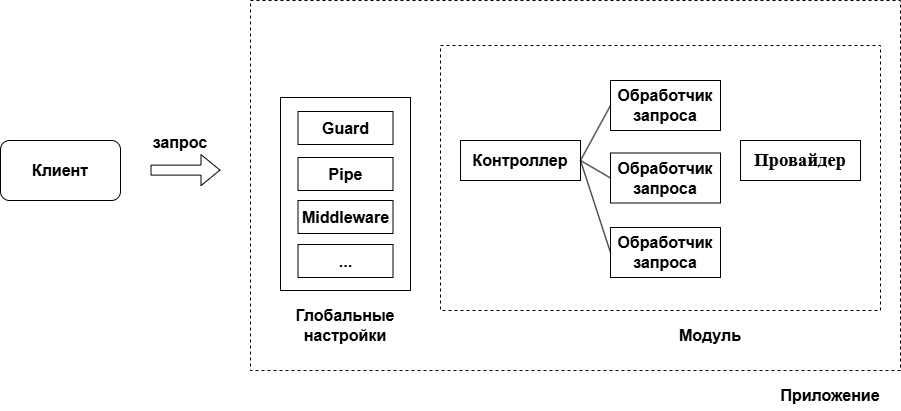


Рисунок 1.3 – Многоуровневая архитектура

**1.3.3**Серверный рендеринг.Это решение необходимо для создания производительных приложений. Суть в том, чтобы рендерить страницы на сервере и генерировать статические файлы на этапе сборки. Эти подходы делают платформу очень быстрой и улучшают SEO, так как поисковые системы могут индексировать контент, который уже отрендерен на сервере. Схематичное отображение работы рендеринга на стороне сервера изображено на рисунке 1.4.

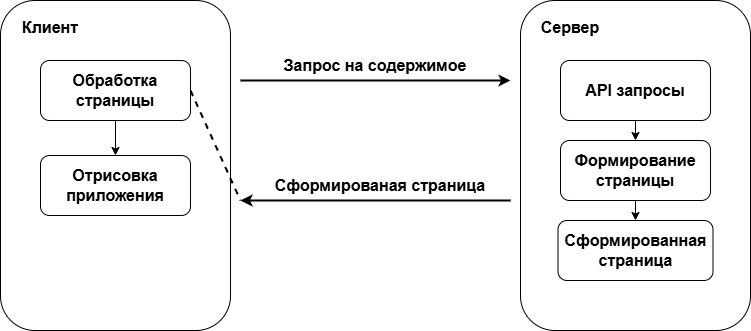


Рисунок 1.3 – Рендеринг на стороне сервера

Ещё одна мощная функция − инкрементальная статическая регенерация. Она сочетает преимущества статической генерации и динамических обновлений: страницы генерируются на этапе сборки, но при необходимости могут обновляться без полной пересборки проекта.

**1.4 Безопасность современных приложений**

Современные приложения должны соответствовать принципам конфиденциальности, доступности и целостности данных. Информация должна быть защищена от несанкционированного доступа, а платформа − обеспечивать стабильный и непрерывный доступ к данным и сервисам. Кроме того, необходимы меры для предотвращения несанкционированного изменения или повреждения информации. Несоблюдение хотя бы одного из этих принципов может привести к утрате доверия пользователей, отказу от продукта или юридической ответственности.

Конфиденциальность в современных приложениях включает в себя несколько ключевых аспектов:

1. Контроль доступа − ограничение прав пользователей и системных компонентов на основе ролевой модели или других механизмов аутентификации и авторизации.
2. Шифрование данных − использование шифрования при передаче и хранении данных для защиты от утечек и перехвата.
3. Минимизация сбора данных − сбор и хранение только тех данных, которые необходимы для работы системы.
4. Защита от угроз − использование разных механизмов безопасности.
5. Безопасное удаление данных − гарантия полного удаления данных пользователей по их запросу или по истечении срока хранения.

Для проектирования системы необходимо понимать типы атак и их потенциальные последствия. Для определения актуальных угроз можно использовать открытый проект по безопасности приложений OWASP (Open Web Application Security Project) [5]. Эксперты организации каждый год обновляют список критических уязвимостей приложений.

SQL-инъекция – это тип атаки, при котором злоумышленник вставляет вредоносные SQL-запросы в поля ввода приложения с целью манипулирования базой данных. Эта атака эксплуатирует уязвимости в обработке входных данных и может привести к краже, модификации или удалению данных. Причиной данной уязвимости является недостаточная фильтрация ввода и ошибки, которые могут быть выведены сервером и дать злоумышленнику подсказки о том, как правильно сформировать вредоносный код для выполнения.

Для защиты от SQL-инъекций следует использовать следующие методы:

1. Использование ORM. Многие современные фреймворки предлагают ORM, которые автоматически экранируют вводимые данные и защищают от SQL-инъекций.
2. Валидировать и фильтровать данные. Валидация ввода данных пользователя перед их использованием в запросах.
3. Использовать подготовленные выражения. Вместо прямой вставки данных пользователя в SQL-запросы, использовать подготовленные выражения.

Межсайтовая запросная подделка (CSRF) − это атака, при которой злоумышленник заставляет браузер жертвы выполнить нежелательные действия на доверенном сайте, на котором пользователь уже авторизован. CSRF использует авторизационные данные пользователя, такие как cookies, чтобы совершить действия от его имени.

Пример атаки:

1. Пользователь авторизуется на доверенном сайте.
2. Злоумышленник создает фальшивую веб-страницу, которая содержит скрытую форму, отправляющую запрос на сервер банка.
3. Злоумышленник размещает эту страницу на форуме или отправляет ссылку на эту страницу жертве.
4. Пользователь, будучи авторизованным на сайте банка, случайно или по ошибке посещает вредоносную страницу, не подозревая, что она отправляет запрос на сайт банка.
5. Браузер жертвы автоматически отправляет запрос с авторизационными данными (например, cookies), а сервер банка обрабатывает его как обычный запрос, потому что он приходит от авторизованного пользователя.

Для предотвращения этой атаки рекомендуется использовать уникальные ключи для каждой сессии, которые должны передаваться в теле запроса, или предотвращать автоматическую передачу данных с межсайтовыми запросами.

Одними из наиболее распространённых проблем безопасности в приложениях являются нарушение контроля доступа и небезопасный дизайн архитектуры. Эти уязвимости позволяют злоумышленникам получить доступ к ресурсам и функционалу, к которым они не должны иметь доступа. Ошибки в проектировании системы могут привести к серьёзным последствиям, включая утечку данных и компрометацию всей инфраструктуры, поэтому важно внедрить лучшие практики на этапах разработки приложения. Для платформ рекомендуется:

1. Разрабатывать систему контроля доступа, руководствуясь принципом минимизации привилегий, предоставляя пользователям только необходимые им права для выполнения их задач.
2. Реализовывать проверку аутентификации и авторизации на всех уровнях приложения.
3. Продумывать возможные уязвимости на этапе проектирования системы.
4. Проводить тесты контроля доступа и моделировать возможные сценарии атак.

Также важным и популярным недостатком в приложениях является пренебрежение криптографией. Это уязвимость, связанная с недостаточным или неправильным использованием методов шифрования. Отсутствие шифрования данных, слабые алгоритмы или неправильное управление ключами могут привести к утечке конфиденциальной информации.

Частыми ошибками являются:

1. Использование устаревших алгоритмов.
2. Хранение паролей и ключей в открытом виде или внутри кода приложения.
3. Использование слабых или предсказуемых ключей шифрования.
4. Отсутствие шифрования при передаче данных по сети.
5. Разрабатывать собственные криптографические алгоритмы, если разработчик не является экспертом в данной области.

Применение современных криптографических методов, использование надёжных библиотек и соблюдение рекомендаций по безопасности играет ключевую роль в защите данных. В таблице 1.2 представлено сравнение основных алгоритмов хеширования.

Таблица 1.2 – Cравнение основных алгоритмов хеширования

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Уровень безопасности | Возможность настройки сложности | Использование памяти |
| CRYPT | Не рекомендуется | Нет | Нет |
| MD5crypt | Не рекомендуется | Нет | Нет |
| BCRYPT | Высокая | Да | Нет |
| PBKDF2 | Средняя – высокая | Да | Нет |
| Scrypt | Высокая | Да | Да |
| Argon2 | Высокая | Да | Да |

Для надёжной защиты данных предпочтительно использовать Argon2 или Scrypt, так как эти алгоритмы обеспечивают высокий уровень безопасности. Они позволяют настраивать сложность вычислений и требуют значительных объёмов памяти, что делает их устойчивыми к атакам с использованием специализированного оборудования. Если применение этих алгоритмов невозможно, BCRYPT остаётся достойной альтернативой.

**2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ**

В данном разделе изложены основные сценарии взаимодействия системы с одним или несколькими действующими лицами, а также представлено высокоуровневое описание приложения. На основе системного подхода выявлена структура платформы.

**2.1 Ключевые сценарии**

Основной задачей при проектировании приложения является определение ключевых сценариев, что позволит принять оптимальное решение относительно архитектуры. Важно найти баланс между потребностями пользователя, бизнеса и техническими требованиями системы. Для этого нужно определить ключевые варианты использования со стороны пользователя и провести их анализ, чтобы выявить соответствующие задачи для разработки.

Основные сценарии взаимодействия с системой:

* авторизация пользователей;
* аутентификация пользователей;
* управление рабочим пространством;
* управление досками;
* совместная работа;
* экспорт и импорт данных.

Далее необходимо разбить основные сценарии на элементарные составляющие с точки зрения реализации.

**2.1.1**Авторизации и аутентификация пользователей.Когда пользователь открывает страницу регистрации, система отображает форму с полями почта, пароль и подтверждение пароля. Форма проводит проверку ввода на корректность в реальном времени.

После успешного заполнения формы данные отправляются на сервер с помощью запроса. Сервер проверяет: не зарегистрирован ли уже пользователь. Если данные уникальны, выполняет хеширование пароля. Затем данные записываются в базу данных.

После успешной регистрации система генерирует уникальный ключ подтверждения и отправляет пользователю письмо со ссылкой. При переходе по этой ссылке сервер проверяет ключ, активирует учётную запись и уведомляет пользователя об успешной верификации.

Когда пользователь вводит почту и пароль на странице входа, форма отправляет данные на сервер. Сервер выполняет поиск пользователя по почте в базе данных. Если учётная запись найдена, происходит сравнение введённого пароля с сохранённым хешем. Если пароль совпадает, система генерирует ключ доступа, который передаётся пользователю. Этот ключ хранится в локальном хранилище.

При последующих запросах на защищённые ресурсы клиент передаёт этот ключ, а сервер проверяет его достоверность, срок действия и соответствие пользователю. Если ключ истёк или недействителен, пользователь перенаправляется на страницу входа.

Этот подход обеспечивает защиту данных, безопасное хранение паролей и удобный механизм аутентификации

**2.1.2**Управление рабочим пространством и досками.Когда пользователь успешно авторизуется в системе и инициирует загрузку рабочего пространства, система начинает обработку запроса. Первым шагом сервер отправляет информацию о рабочем пространстве пользователя, включая его структуру, настройки и связанный контент.

После получения данных система применяет базовые параметры интерфейса, загружает пользовательские настройки и адаптирует отображение элементов в соответствии с предпочтениями пользователя.

Затем контент связывается с элементами интерфейса, и данные интегрируются в рабочее пространство. Текстовые записи, файлы, таблицы и другие компоненты привязываются к своим местам, обеспечивая целостность структуры. Как только все элементы загружены и отображены, система визуализирует рабочее пространство, предоставляя пользователю готовую среду для работы.

На каждой странице пользователь имеет специальную ячейку, в которую можно вставлять шаблонизированный компонент. Этот компонент поддерживает ввод данных, позволяя пользователю настраивать его содержимое в зависимости от текущих задач и потребностей.

Все изменения, внесённые в компонент, сохраняются автоматически и синхронизируются с сервером в реальном времени. Данные компонентов сохраняются в специальном формате, обеспечивающем их корректное восстановление и дальнейшую обработку при последующих загрузках рабочего пространства.

Также сессия локально хранит историю изменений, чтобы предотвратить ошибки случайного пользовательского ввода.

Страницы могут храниться внутри других страниц. Это позволяет формировать иерархическую структуру, где одна страница может выступать в роли контейнера для нескольких вложенных страниц. Такой подход создаёт гибкую систему досок.

**2.1.3**Совместная работа.Для реализации совместной работы владелец страницы должен иметь возможность делиться ею с другими, предоставляя доступ с разными уровнями прав. Для этого он может либо сформировать специальную ссылку, либо отправить приглашение через платформу, указав конкретных пользователей.

При создании ссылки владелец выбирает уровень доступа: просмотр, комментирование или полный доступ к редактированию. Аналогично, при приглашении пользователей он может задать индивидуальные права для каждого.

Когда владелец изменяет настройки доступа, система обновляет права на сервере, синхронизируя их в реальном времени. Формируется журнал активности, который фиксирует изменения и действия пользователей на странице, а также систему уведомлений, информирующую участников о новых приглашениях, изменениях прав или обновлениях в содержимом.

**2.1.4**Экспорт и импорт данных.Пользователь может запросить у сервера доступные ему данные в специальном формате, предназначенном для сохранения и последующего восстановления структуры информации. Сервер обрабатывает запрос, собирая все необходимые данные в единый пакет, включая содержимое страниц, вложенные элементы, файлы и настройки. После подготовки данные передаются пользователю в виде загружаемого файла.

Необходимо реализовать выборочные параметры экспорта, позволяя пользователю загружать только определённые данные, а также поддержку нескольких форматов.

Также пользователь может загрузить подготовленный файл с данными на сервер. При получении пакета система анализирует его содержимое и проверяет совместимость формата с платформой. Если файл поддерживается, сервер корректно связывает данные с учётной записью пользователя, воссоздавая структуру страниц, элементов и вложенных данных.

Чтобы минимизировать ошибки, необходимо добавить предварительный просмотр импортируемых данных перед их окончательной загрузкой.

**2.2 Структура приложения**

На основе ключевых сценариев и требований для приложения были выделены следующие блоки:

* блок представления;
* серверная часть;
* база данных
* файловое хранилище.

Блок представления в можно разбить на несколько ключевых частей, которые обеспечивают различные аспекты взаимодействия с пользователем. Можно выделить следующие модули:

* модуль интерфейса пользователя;
* модуль хранения состояний;
* модуль загрузки;
* модуль компонент интерфейса;
* модуль компонент пользователя;
* модуль контроля данных;
* модуль истории сессии;
* модуль локального хранилища.

Серверную часть для упрощения разработки можно разделить на следующие модули:

* модуль авторизации;
* модуль управления учетными записями;
* модуль управления рабочим пространством;
* модуль доступа к базе данных;
* модуль управления файлами;

**2.3 Блок представления**

Главная задача блока представления − предоставить интерфейс пользователю для взаимодействия с его данными. Это включает в себя организацию данных в понятном и доступном формате, а также обеспечение интуитивно понятных инструментов для их просмотра, редактирования и анализа. Блок представления отвечает за визуализацию информации.

**2.3.1**Модуль интерфейса пользователя. Модуль интерфейса пользователя является главной частью блока представления. Этот модуль отвечает за обеспечение взаимодействия пользователя с системой через визуальные элементы и обработку действий. Он выступает связующим звеном между пользователем и функционалом приложения, предоставляя интуитивно понятный способ управления данными и процессами.

Эта часть приложения будет управлять процессами других модулей блока. Также она будет содержать конфигурационные настройки для всех компонент.

**2.3.2**Модуль хранения состояний. Модуль хранения состояний отвечает за централизованное управление данными, которые определяют текущее состояние приложения. Для приложений с большим количеством компонент и связей необходимо хранилище, которое поможет контролировать поток информации. Данный блок предоставляет инструментарий модулю интерфейса пользователя для управления промежуточными данными.

Главная задача модуля − сохранять целостность данных и синхронизировать их с интерфейсом и бизнес-логикой.

**2.2.3** Модуль загрузки. Этот модуль отвечает за управление процессами загрузки данных, файлов или ресурсов. Он обеспечивает запросы к серверу для получения данных, которые необходимы компонентам интерфейса и пользователя.

Этот модуль напрямую связан с модулем хранения состояний.

**2.2.4** Модуль компонент интерфейса. Этот модуль отвечает за создание, управление и повторное использование стандартизированных элементов пользовательского интерфейса. Он содержит компоненты единообразного дизайна.

Данный блок связан напрямую с модулем пользовательского интерфейса.

**2.2.5** Модуль компонент пользователя. Этот модуль отвечает за реализацию элементов интерфейса, которые позволяют пользователю взаимодействовать с данными − просматривать, редактировать, структурировать и анализировать информацию. Он предоставляет стандартизированные шаблоны для отображения контента и включает как базовые, так и продвинутые компоненты, обеспечивая гибкость и удобство работы.

Базовые элементы представления данных будут включать: текст, списки, таблицы, медиа, файлы, код.

Также у модуля есть подмодуль, который будет отвечать за продвинутые компоненты. Подмодуль продвинутых компонентов будет включать: графики и визуализации, базы данных, сложные структуры.

Данный модуль связан с модулем пользовательского интерфейса.

**2.2.6** Модуль представления данных. Этот модуль отвечает за преобразование, форматирование и структурирование данных для их корректного и удобного отображения в интерфейсе.

Он выступает промежуточным звеном между «сырыми» данными и UI-компонентами, обеспечивая их совместимость и готовность к визуализации. Когда данные пользователя поступают в модуль хранения состояний, тогда происходит анализ в модуле представления.

**2.2.7** Модуль истории сессии. Этот модуль отвечает за отслеживание, сохранение и восстановление действий пользователя в рамках одной сессии работы с приложением. Он позволяет реализовать функционал отмены и повтора операций, анализировать поведение пользователя и восстанавливать контекст работы после сбоев или перезагрузки. Модуль хранит историю с помощью модулей локального хранилища и состояний.

Также блок необходим для инициализации загрузки изменений на сервер.

**2.2.8** Модуль локального хранилища. Этот модуль отвечает за сохранение данных на стороне клиента, обеспечивая ускорение работы приложения и снижение нагрузки на сервер. Он предоставляет инструменты для безопасного и эффективного управления локальными данными, синхронизируя их с облаком при необходимости. Этот блок необходим для сохранения информации сессии пользователя. Данными данного блока пользуются модули загрузки и истории сессии.

**2.4 Серверная часть**

Серверная часть необходима для каждого продвинутого приложения. Данный блок содержит основную бизнес логику платформы. Это центральная часть программы.

**2.4.1**Модуль управления учетными записями. Модуль отвечает за редактирование и удаление учетных записей пользователей в системе. Этот модуль является центральным для администрирования пользовательских данных и обеспечивает функциональность, необходимую для поддержки жизненного цикла учетных записей.

Основные функции модуля включают управление ролями и правами пользователей, назначая им соответствующие уровни доступа в системе.

Модуль также будет вести журнал действий пользователей, что полезно для отслеживания изменений и статистики.

Также задачей модуля является регистрация новых пользователей, вход в систему, выход и восстановление пароля. Модуль ответственен за генерацию ключей доступа, которые используются для аутентификации в запросах пользователей.

Кроме того, модуль отвечает за управление сессиями пользователей, включая отслеживание активных сессий, их завершение по истечении срока действия ключей доступа или принудительно по запросу пользователя. Для повышения безопасности будет реализована многофакторная аутентификация, которая требует дополнительного подтверждения личности через электронную почту.

**2.4.2**Модуль авторизации. Модуль отвечает за управление правами доступа пользователей к ресурсам и функциям системы. В отличие от аутентификации, которая подтверждает личность пользователя, авторизация определяет, что именно пользователь может делать в системе. Этот модуль проверяет, имеет ли пользователь достаточные права для выполнения запрашиваемых действий, таких как просмотр, редактирование или удаление данных.

Модуль авторизации тесно интегрирован с модулем аутентификации, так как для проверки прав доступа необходимо знать, кто именно выполняет запрос. Например, после успешной аутентификации пользователь получает ключ доступа, который содержит информацию о его роли или правах. При каждом запросе к защищенному ресурсу модуль авторизации проверяет этот ключ и определяет, разрешено ли действие.

**2.4.3**Модуль управления рабочим пространством. Данный модуль является одним из ключевым в работе платформы.

Модуль отвечает за создание, настройку, организацию и удаление досок. Этот модуль предоставляет пользователям возможность создавать доски, настраивать их структуру. Также блок контролирует изменение в страницах.

**2.4.4**Модуль управления файловым хранилищем. Данный модуль является связующим звеном между приложением и системой хранения файлов.

Этот модуль отвечает за организацию и управление файлами, а также за поддержание их целостности и структуры. Он отслеживает состояние «озера данных», обеспечивая корректное размещение и доступ к файлам, а также контролирует их целостность, чтобы предотвратить потерю или повреждение данных. Кроме того, модуль создает и управляет метаданными файлов, такими как название, размер, тип, дата создания, автор и другие атрибуты, которые помогают быстро находить и идентифицировать нужные файлы.

Также модуль занимается экспортом и импортом данных. Он анализирует корректность и формат информации.

**2.4.5**Модуль доступа к базе данных. Данный модуль обеспечивает безопасное и эффективное взаимодействие между приложением и базой данных.

Этот модуль отвечает за выполнение запросов к базе данных, таких как чтение, запись, обновление и удаление данных, а также за управление соединениями с базой данных. Он предоставляет абстракцию над низкоуровневыми операциями, что упрощает работу с данными для других модулей системы.

**3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

Данный раздел дает исчерпывающие знания о реализации приложения. Здесь находится описание о структуре платформы, разработанной в предыдущем разделе (см. раздел 2 Системное проектирование), с точки зрения описания данных и обрабатывающих подпрограмм, функций процедур. В разделе

**3.1 Серверная часть**