



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
НА ТЕМУ:
«Программно-алгоритмический комплекс с
многоцелевой и масштабируемой архитектурой для
совместной работы и управления проектами»

Студент ИУ7-86Б
(Группа)

(Подпись, дата)

Шматко К. М.
(И. О. Фамилия)

Руководитель ВКР

(Подпись, дата)

Никульшин А. М.
(И. О. Фамилия)

2025 г.

РЕФЕРАТ

Расчетно-пояснительная записка 53 с., 12 рис., 1 табл., 16 источн., 1 прил.

В данной работе рассматривается проектирование и разработка программно-алгоритмического комплекса с многоцелевой и масштабируемой архитектурой, предназначенного для организации совместной работы пользователей и управления проектами.

Проведен анализ предметной области, включающий обзор и сравнение современных реляционных и нереляционных систем хранения данных. Рассмотрены характеристики распределенных систем хранения и методы организации сетевого многопользовательского взаимодействия.

Разработаны основные положения и структура предлагаемого программно-алгоритмического комплекса. Описана его модульная архитектура, ключевые компоненты и принципы их взаимодействия. Определены основные структуры данных, используемые для хранения пользовательской информации, проектных данных и создаваемого контента.

Обоснован выбор средств программной реализации. Выполнено тестирование разработанного комплекса для проверки корректности его работы. Описаны основные сценарии взаимодействия пользователя с программным обеспечением.

Проведено исследование потенциальной применимости разработанного комплекса в различных сценариях управления проектами и командной работы. Исследованы ключевые характеристики системы.

Ключевые слова: совместная работа, управление проектами, программный комплекс, масштабируемая архитектура, распределенное хранение данных, MongoDB, PostgreSQL, WebSockets, многопользовательское взаимодействие, гибридная база данных.

СОДЕРЖАНИЕ

РЕФЕРАТ	2
ОПРЕДЕЛЕНИЯ	5
ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ	6
ВВЕДЕНИЕ	7
1 Аналитический раздел	8
1.1 Обзор современных систем хранения данных	8
1.1.1 Определение и классификация систем управления базами данных	8
1.1.2 Неоднородные данные и их особенности	10
1.2 Анализ предметной области систем распределенного хранения данных	11
1.2.1 Характеристики современных систем хранения	11
1.2.2 Влияние типов данных на выбор системы хранения . . .	12
1.2.3 Особенности хранения текстов и структурированных дан- ных на примере структуры <i>Пользователь</i>	15
1.3 Сравнение систем распределенного хранения данных	19
1.3.1 Сравнение систем хранения	19
1.3.2 Реализация распределённых баз данных на примере MongoDB и PostgreSQL	21
1.3.3 Особенности сочетания MongoDB и PostgreSQL	23
1.4 Методы сетевого многопользовательского взаимодействия . . .	24
1.4.1 Архитектуры взаимодействия	24
1.4.2 Протоколы и подходы к обмену данными	25
1.5 Формализация задачи создания комплекса	26
1.5.1 Исходные задачи программно–алгоритмического комплекса	26
1.5.2 Диаграмма вариантов использования	28
1.5.3 Анализ требований пользователей и ролей	35

2	Конструкторский раздел	38
2.1	Основные положения предлагаемого программно- алгоритмического комплекса	38
2.2	Структура и компоненты программного приложения	39
2.3	Сценарий совместного редактирования	41
2.4	Ключевые структуры данных	43
2.4.1	Реляционная модель данных	43
2.4.2	Документная модель данных (MongoDB)	44
3	Технологический раздел	48
4	Исследовательский раздел	49
	ЗАКЛЮЧЕНИЕ	50
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	52
	ПРИЛОЖЕНИЕ А	53

ОПРЕДЕЛЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие термины с соответствующими определениями.

Системы хранения данных — совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Распределенная система хранения данных — система хранения, в которой данные физически расположены на нескольких узлах (серверах), связанных сетью, но представляются пользователю или приложению как единое целое.

Масштабируемость — способность системы, сети или процесса увеличивать свою производительность и пропускную способность пропорционально увеличению нагрузки (количества пользователей, объема данных, транзакций) путем добавления ресурсов (аппаратных или программных).

PostgreSQL — объектно-реляционная система управления базами данных, соответствующая стандарту SQL, расширяемая и поддерживающая сложные запросы и транзакции (ACID).

MongoDB — документоориентированная NoSQL система управления базами данных, использующая для хранения данных JSON-подобные документы (BSON).

WebSocket — сетевой протокол, обеспечивающий постоянное полнодуплексное (двустороннее) соединение между клиентом и сервером поверх одного TCP-соединения, предназначенный для интерактивного обмена данными в реальном времени.

Mind map (интеллект-карта, диаграмма связей) — метод структуризации и визуализации концепций с использованием графической записи в виде диаграммы.

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие сокращения и обозначения.

БД — База Данных

СУБД — Система Управления Базами Данных

ACID — Atomicity, Consistency, Isolation, Durability

API — Application Programming Interface

GraphQL — Graph Query Language

HDFS — Hadoop Distributed File System

HTTP — HyperText Transfer Protocol

HTTPS — HyperText Transfer Protocol Secure

JSON — JavaScript Object Notation

NoSQL — Not only SQL

P2P — Peer-to-Peer

REST — Representational State Transfer

S3 — Simple Storage Service

SQL — Structured Query Language

TCP — Transmission Control Protocol

UDP — User Datagram Protocol

WebRTC — Web Real-Time Communication

XML — Extensible Markup Language

ФИО — Фамилия, Имя, Отчество

ВВЕДЕНИЕ

В условиях динамично развивающейся цифровой экономики и роста популярности удаленных и гибридных форматов работы, организация совместной деятельности и управления проектами становится критически важной для успеха команд и организаций. Существующие инструменты часто либо специализированы на узком круге задач, либо представляют собой разрозненные сервисы, что усложняет интеграцию рабочих процессов и управление информацией. Возникает потребность в универсальных платформах, обеспечивающих комплексную поддержку командной работы. Более того, данные, генерируемые пользователями в рамках работы с подобным комплексом (текстовые записки, созданные схемы, файлы проектов, история коммуникаций в чатах и др.), могут послужить основой для формирования уникальных наборов данных (датасетов). Эти датасеты в перспективе могут быть использованы для проведения исследований и выполнения выпускных квалификационных работ студентами последующих курсов.

Целью данной работы является разработка программно-алгоритмического комплекса с многоцелевой и масштабируемой архитектурой для совместной работы и управления проектами.

Для достижения поставленной цели необходимо решить следующие задачи:

- 1) Провести анализ предметной области, обзор и сравнение существующих систем хранения данных и методов сетевого многопользовательского взаимодействия, а также формализовать требования к разрабатываемому комплексу.
- 2) Разработать архитектуру программно-алгоритмического комплекса, структуру приложения, описание его компонентов, их взаимодействия и ключевых структур данных.
- 3) Обосновать выбор средств программной реализации, разработать программное обеспечение, реализующее основные функции комплекса, и выполнить его тестирование для проверки корректности работы.
- 4) Провести исследование применимости и оценить ключевые характеристики разработанного программно-алгоритмического комплекса.

1 Аналитический раздел

1.1 Обзор современных систем хранения данных

1.1.1 Определение и классификация систем управления базами данных

Системы хранения данных, рассматриваемые в данном контексте, представляют собой совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями [1]. Эти системы, известные также как системы управления базами данных (СУБД), обеспечивают структурированное хранение данных и поддерживают операции над ними, такие как манипуляции, индексация, выполнение сложных запросов и поддержание целостности данных.

Существуют три основные группы моделей баз данных, различающихся структурой организации данных: дореляционные, реляционные и постреляционные базы данных.

Дореляционные базы данных

Основной особенностью дореляционных моделей является то, что взаимосвязи между данными управляются явно, и структура данных часто определяется физическим расположением данных, а не логическими ассоциациями.

Дореляционные модели включают в себя:

- 1) иерархическую модель, где структура данных представлена в виде дерева, информация в котором разбита на сегменты, имеющие строгое родитель–дочернее отношение;
- 2) сетевую модель, позволяющую создавать множество связей между узлами, что делает возможным иметь несколько родителей у одного узла;
- 3) инвертированные списки, строящиеся на основе терминов (слов или фраз), которые встречаются в документах или текстовых полях базы данных.

Реляционные базы данных

Основной особенностью реляционных баз данных является использование структурированных таблиц, где данные организованы в столбцы и строки [2]. Каждая таблица представляет собой коллекцию кортежей, общая структура которых описана в схеме таблицы. Таблицы могут быть связаны друг с другом через идентификаторы и ключи, что позволяет структурировать и интегрировать данные из различных источников.

Реляционные базы данных обладают набором свойств ACID:

- 1) атомарность – транзакция будет выполнена как одно целое.
- 2) согласованность – транзакция переводит базу данных из одного согласованного состояния в другое согласованное состояние.
- 3) изолированность – транзакция выполняется изолированно от других транзакций.
- 4) устойчивость – результаты успешно завершенной транзакции будут устойчивы к будущим сбоям.

Постреляционные базы данных

Основной особенностью постреляционных баз данных является их гибкость в обработке разноплановых и менее структурированных данных, а также поддержка более производительных способов распределения данных и масштабирования [2].

Виды постреляционных моделей:

- 1) ключ-значение хранилища;
- 2) графы;
- 3) документоориентированные базы данных.

1.1.2 Неоднородные данные и их особенности

Неоднородные данные представляют собой различные типы данных, которые могут отличаться по структуре, формату и источнику. Среди основных характеристик неоднородных данных можно выделить следующие:

- 1) форматы – неоднородные данные включают разнообразные форматы, такие как текст, изображения, аудио, видео, JSON, XML и таблицы SQL.
- 2) источники – неоднородные данные поступают из различных источников, включая социальные сети, системы транзакций, веб-логи и корпоративные информационные системы.
- 3) структурированность – неоднородные данные подразделяются на структурированные, полуструктурированные и неструктурированные, в зависимости от степени их организации.

Работа с неоднородными данными требует специального подхода, учитывающего их разнородность и сложность. Рассмотрим основные задачи, связанные с обработкой и управлением такими данными [3].

- 1) **Интеграция данных.** Объединение данных из различных источников требует нормализации, очистки и преобразования данных для обеспечения их согласованности и целостности.
- 2) **Хранение данных.** Разнообразие форматов может требовать использования гибридных решений для хранения, включая в себя как реляционные, так и NoSQL системы для обработки различных типов данных.
- 3) **Обработка и анализ.** Обработка разнообразных типов данных требует применения различных аналитических методов, таких как обработка естественного языка (Natural Language Processing) для текстов и алгоритмы компьютерного зрения для изображений.
- 4) **Управление метаданными.** Метаданные играют важную роль в интерпретации данных и обеспечении их доступности, предоставляя информацию о происхождении, структуре и содержании данных.

1.2 Анализ предметной области систем распределенного хранения данных

1.2.1 Характеристики современных систем хранения

Современные системы хранения данных являются фундаментальной основой для построения масштабируемых приложений, особенно в условиях растущих объемов информации и разнообразия типов данных. Они должны обеспечивать высокую производительность, надежность и гибкость для удовлетворения потребностей как разработчиков, так и конечных пользователей.

Ключевые характеристики современных систем хранения данных включают [4]:

- 1) масштабируемость – способность системы увеличивать свои ресурсы (объем хранения, производительность) по мере роста данных и нагрузки без значительной деградации производительности. Масштабируемость может быть вертикальной (увеличение ресурсов на одном сервере) и горизонтальной (добавление новых серверов в кластер);
- 2) устойчивость к отказам – обеспечение непрерывной работы системы даже при сбоях отдельных компонентов. Это достигается посредством репликации данных и избыточности серверов;
- 3) производительность – высокая скорость операций чтения и записи данных, низкая задержка при обработке запросов, что критически важно для приложений в реальном времени;
- 4) гибкость модели данных – поддержка различных моделей данных (реляционная, документоориентированная, графовая и др.) позволяет оптимально хранить и обрабатывать разные типы данных;
- 5) совместимость – возможность интеграции с другими системами и сервисами, поддержка стандартных протоколов и API;
- 6) безопасность – механизмы аутентификации, авторизации, шифрования данных и управления доступом для защиты данных от несанкционированного доступа;

- 7) консистентность данных – обеспечение целостности и согласованности данных во всех узлах распределенной системы, особенно в условиях одновременных операций чтения и записи;
- 8) скалируемая архитектура – возможность распределения нагрузки и хранения данных в масштабируемой архитектуре, такой как кластер или облако;
- 9) управление транзакциями – поддержка атомарности, согласованности, изоляции и долговечности (ACID-свойства) для гарантии правильности и надёжности операций с данными;
- 10) поддержка неоднородных данных – способность хранить и обрабатывать структурированные, полуструктурированные и неструктурированные данные.

Современные системы хранения данных, такие как NoSQL-базы (MongoDB, Cassandra) и реляционные базы данных (PostgreSQL, MySQL), обладают различными комбинациями этих характеристик, что позволяет выбирать оптимальную систему для конкретных задач [5–8].

1.2.2 Влияние типов данных на выбор системы хранения

Типы данных, с которыми предстоит работать системе, играют ключевую роль при выборе подходящей системы хранения. Каждая система оптимизирована для определённых видов данных и операций над ними [4].

Структурированные данные

Структурированные данные имеют строгую и фиксированную структуру. Они организованы в виде таблиц, где каждая запись соответствует определённому набору полей с предопределёнными типами данных. Примеры включают транзакционные данные, инвентаризационные списки, финансовые записи.

Для работы со структурированными данными оптимальны реляционные СУБД, такие как PostgreSQL или MySQL. Они обеспечивают:

- 1) Строгую схему данных и целостность при использовании ограничений и связей между таблицами.
- 2) Возможность выполнения сложных запросов с использованием языка SQL.
- 3) Поддержку транзакций с ACID-свойствами.

Полуструктурированные данные

Полуструктурированные данные не обладают строгой схемой, но имеют внутреннюю организацию и метаданные. Примеры включают XML, JSON-файлы, логи приложений.

Документоориентированные СУБД, такие как MongoDB или Cassandra, подходят для работы с полуструктурированными данными, предоставляя:

- 1) Гибкую схему данных, позволяющую хранить документы с различной структурой.
- 2) Высокую производительность при операциях с большими объёмами документов.
- 3) Масштабируемость и отказоустойчивость в распределённых средах.

Неструктурированные данные

Неструктурированные данные не имеют predetermined модели и включают текстовые документы, изображения, аудио и видео файлы. Их обработка требует специального подхода.

Системы хранения данных для неструктурированных данных, такие как Hadoop HDFS или Amazon S3, позволяют:

- 1) Хранить большие объёмы данных в распределённой среде.
- 2) Обрабатывать данные с использованием параллельных вычислений (например, MapReduce).
- 3) Использовать специализированные инструменты для анализа и поиска в данных.

Графовые данные

Графовые данные состоят из узлов и связей между ними, что позволяет моделировать сложные взаимоотношения. Применяются в социальных сетях, рекомендационных системах, анализе связей.

Графовые СУБД, такие как Neo4j или OrientDB, предлагают:

- 1) Хранение и обработку графовых структур данных.
- 2) Быстрый доступ к связанным данным без необходимости сложных JOIN-операций.
- 3) Специфичные языки запросов для работы с графами (например, Cypher).

Выбор системы хранения

Таким образом, при выборе системы хранения необходимо учитывать:

- 1) Типы данных и их структуру.
- 2) Объём данных и прогнозируемый рост.
- 3) Требования к производительности и масштабируемости.
- 4) Необходимость в транзакционной целостности и консистентности данных.
- 5) Возможность интеграции с существующими системами и инструментами.

Понимание типов данных и их особенностей позволяет выбрать оптимальную систему хранения для конкретных задач.

1.2.3 Особенности хранения текстов и структурированных данных на примере структуры *Пользователь*

В системе совместной работы над проектами структура *Пользователь* является одной из ключевых и включает в себя как структурированные данные, так и связи с другими сущностями системы.

Структура *Пользователь* может включать следующие поля:

- 1) **ФИО** – строковые поля для хранения имени, фамилии и отчества пользователя.
- 2) **Логин** – уникальный идентификатор пользователя в системе.
- 3) **Пароль** – хеш пароля для аутентификации пользователя.
- 4) **Фото** – ссылка на изображение профиля пользователя.
- 5) **Связи с проектами** – информация о проектах, к которым пользователь имеет доступ, а также о проектах, созданных пользователем.

Кроме того, система хранит неструктурированные данные, которые включают такие элементы как тексты и другие мультимедийные данные, представленные в формате JSON.

Хранение структурированных данных

Реляционная база данных является оптимальным выбором для хранения структурированных данных пользователей по следующим причинам:

- 1) **Строгая схема данных.** Реляционные СУБД позволяют задать фиксированную структуру данных, обеспечивая строгую организацию информации о пользователях. Это важно для обеспечения целостности данных, что критично для таких атрибутов, как идентификатор, логин, и хеш пароля.
- 2) **Связи между таблицами.** Возможность моделирования сложных связей между таблицами позволяет управлять отношениями между

пользователями и другими сущностями, такими как проекты. Это упрощает реализацию функциональности, связанной с ролями доступа и управления проектами.

3) **Безопасность и транзакционность.** Реляционные базы данных поддерживают ACID-транзакции, что гарантирует надёжность и консистентность данных, особенно важных в операциях, таких как регистрация новых пользователей и управление их аутентификацией.

Пример создания таблицы пользователей показывает, как можно определить строгую схему для хранения ключевой информации о пользователях 1.1:

Листинг 1.1 – Создание таблицы пользователей

```
CREATE TABLE users (  
    user_id SERIAL PRIMARY KEY,  
    full_name VARCHAR(255) NOT NULL,  
    login VARCHAR(100) UNIQUE NOT NULL,  
    password_hash VARCHAR(255) NOT NULL,  
    photo_url VARCHAR(255),  
    -- Дополнительные поля  
);
```

Для управления связями пользователей с проектами используется дополнительная таблица, что позволяет организовать реляцию «многие ко многим» и управлять ролями и правами доступа 1.2:

Листинг 1.2 – Создание таблицы связей пользователей с проектами

```
CREATE TABLE user_projects (  
    user_id INTEGER REFERENCES users(user_id),  
    project_id INTEGER REFERENCES projects(project_id),  
    role VARCHAR(50),  
    permission VARCHAR(50),  
    PRIMARY KEY (user_id, project_id)  
);
```


Хранение текстов в формате JSON

Для хранения текстов, представленных в формате JSON, выбрана нереляционная база данных по следующим причинам:

- 1) **Отсутствие фиксированной схемы.** Нереляционные базы данных позволяют работать с динамически изменяющимися данными, такими как текстовые блоки, и их метаданные. Это позволяет адаптировать структуру хранения в соответствии с изменяющимися требованиями без необходимости изменения схемы.
- 2) **Гибкость и масштабируемость.** Такие системы плавно подстраиваются под изменяющиеся объёмы данных и требования, позволяя легко хранить и манипулировать информацией в формате JSON.
- 3) **Высокая производительность при работе с документами.** Оптимизация под работу с JSON-документами обеспечивает быструю обработку и выборку данных, что важно при хранении и доступе к текстовым описаниям и метаданным.

Пример документа в нереляционной базе данных представлен в листинге 1.3:

Листинг 1.3 – Пример документа в нереляционной базе данных

```
{
  "_id": ObjectId("..."),
  "user_id": 123,
  "content": "Текстовый контент...",
  "metadata": {
    "tags": ["example", "text"],
    "created_at": ISODate("2024-12-01T12:00:00Z")
  },
  "additional_fields": {
    "field1": "value1",
    "field2": "value2"
  }
}
```

Поле `user_id` связывает текст с конкретным пользователем из реляционной базы данных, что позволяет осуществлять интеграцию между реляционной и нереляционной системами хранения на уровне приложения.

Хранение схем объектов для расширяемости системы

Чтобы обеспечить возможность расширения системы и добавления новых типов объектов без необходимости изменения существующего кода или структуры базы данных, необходимо хранить схемы объектов в базе данных. Это касается таких объектов, как тексты, изображения, графики и другие пользовательские сущности.

Нереляционная база данных, благодаря своей гибкой схеме и документно-ориентированной структуре, подходит для хранения как данных, так и соответствующих им схем. Схемы объектов могут быть сохранены в специальной коллекции, например, `objectSchemas`, где каждый документ описывает структуру определённого типа объекта.

Пример документа-схемы показан в листинге 1.4:

Листинг 1.4 – Пример документа-схемы

```
{
  "_id": ObjectId("..."),
  "object_type": "text",
  "schema": {
    "fields": [
      "title": "string",
      "required": "boolean",
      "type": "string"
    ],
    "properties": "array",
    "handlers": "array"
  },
  "created_at": ISODate("2024-12-01T12:00:00Z")
}
```

1.3 Сравнение систем распределенного хранения данных

1.3.1 Сравнение систем хранения

Для сравнения MongoDB, PostgreSQL, MySQL и Cassandra использованы следующие критерии:

- 1) **Горизонтальная масштабируемость:** возможность расширения системы путём добавления новых серверов.
- 2) **Вертикальная масштабируемость:** возможность улучшения производительности через увеличение мощности существующих серверов.
- 3) **Изменяемость схемы данных:** простота адаптации к изменяющейся структуре данных.
- 4) **Поддержка различных форматов данных:** работа с JSON и другими современными форматами.
- 5) **Консистенция данных:** гарантии целостности данных.
- 6) **Поддержка транзакций:** наличие и применение транзакций с ACID-свойствами.
- 7) **Поддержка полуструктурированных и неструктурированных данных:** возможность работы с менее структурированными данными.
- 8) **Поддержка структурированных данных:** работа с классической реляционной моделью.
- 9) **Активность и поддержка сообщества:** включая доступность документации, форумов и специалистов.
- 10) **Инструменты и интеграции:** доступность инструментов для разработки и интеграции с другими технологиями.

На основе этих критериев составлена таблица сравнения MongoDB, PostgreSQL, MySQL и Cassandra (таблица 1.1) [9; 10]. При оценке критериев

был применен весовой коэффициент, где «плюс» оценивается в 1 балл, «минус» в 0 баллов, а «плюс-минус» в 0.5 балла.

Таблица 1.1 – Сравнение MongoDB, PostgreSQL, MySQL и Cassandra

Критерий сравнения	MongoDB	PostgreSQL	MySQL	Cassandra
Горизонтальная масштабируемость	+	+-	-	+
Вертикальная масштабируемость	-	+	+	-
Изменяемость схемы данных	+	+-	-	+
Поддержка различных форматов данных	+	-	-	+
Консистенция данных	-	+	+	-
Поддержка транзакций	-	+	+	-
Поддержка полуструктурированных и неструктурированных данных	+	-	-	+
Поддержка структурированных данных	-	+	+	-
Активность и поддержка сообщества	+	+	+	+-
Инструменты и интеграции	+	+	+	+-
Сумма баллов	6	7	6	5

Из таблицы видно, что MongoDB и PostgreSQL лучше удовлетворяют ключевые критерии по сравнению с Cassandra и MySQL, однако ни одна из баз данных не охватывает все требования полностью. В связи с этим для выполнения всех задач системы необходимо их совместное использование. Это позволит:

- 1) Использовать **MongoDB** для хранения полуструктурированных и неструктурированных данных, таких как тексты, документы, мультимедиа и схемы объектов.
- 2) Применять **PostgreSQL** для хранения структурированных данных, требующих строгой консистентности и поддержки транзакций, таких как информация о пользователях, правах доступа, связях между объектами системы.
- 3) Объединить сильные стороны обеих систем, создавая гибкую, масштабируемую и надежную систему хранения, способную эффективно по времени и памяти обрабатывать неоднородные данные.

1.3.2 Реализация распределённых баз данных на примере MongoDB и PostgreSQL

Распределённые базы данных позволяют хранить и обрабатывать данные на нескольких серверах, обеспечивая масштабируемость, отказоустойчивость и высокую доступность системы.

MongoDB

MongoDB изначально спроектирована как распределённая документно-ориентированная база данных. Её основные механизмы реализации распределённости включают:

- 1) **Шардинг** – горизонтальное разделение данных по нескольким узлам (шардам). Позволяет масштабировать базу данных горизонтально, добавляя новые серверы для обработки увеличивающихся объёмов данных и нагрузки [11].

- 2) **Репликация** – процесс копирования и поддержания синхронных копий данных на нескольких узлах. Обеспечивает отказоустойчивость и высокую доступность системы, позволяя автоматически переключаться на резервные узлы в случае сбоя основного [11].
- 3) **Балансировка нагрузки** – распределение запросов между узлами кластера для обеспечения оптимальной производительности.

PostgreSQL

PostgreSQL изначально не является распределённой СУБД, однако существуют инструменты и расширения, позволяющие реализовать распределённость:

- 1) **Репликация.** PostgreSQL поддерживает потоковую репликацию на уровне основного и репликантных серверов, что обеспечивает отказоустойчивость и балансировку нагрузки на чтение [12].
- 2) **Логическая репликация.** Позволяет реплицировать отдельные таблицы и данные, что обеспечивает большую гибкость при настройке репликации [12].
- 3) **Расширения для горизонтального масштабирования:**
 - 1) **Citus** —расширение для PostgreSQL, которое превращает его в распределённую систему путем горизонтального масштабирования таблиц по узлам кластера [13]. Citus позволяет автоматически распределять данные по узлам и параллельно обрабатывать запросы, увеличивая производительность и масштабируемость.
 - 2) **Postgres-XL** – масштабируемая распределённая база данных на основе PostgreSQL, обеспечивающая параллельную обработку запросов и распределение данных [14]. Предоставляет как масштабирование на чтение, так и на запись, поддерживает транзакции и обеспечивает согласованность данных в кластере.
 - 3) **pg_pool** и **pg_bouncer** – инструменты для управления пулом подключений и балансировки нагрузки, которые помогают улучшить производительность и масштабируемость системы [15][16].

Позволяют распределять запросы между основным сервером и репликами для оптимального использования ресурсов.

Эти инструменты позволяют масштабировать PostgreSQL **горизонтально**, то есть увеличивать производительность системы путем добавления новых серверов или узлов. Горизонтальное масштабирование позволяет обрабатывать больший объем данных и нагрузку благодаря распределению данных и запросов по нескольким узлам. Это делает PostgreSQL пригодным для использования в распределённых системах.

1.3.3 Особенности сочетания MongoDB и PostgreSQL

Совместное использование MongoDB и PostgreSQL позволяет работать с разными типами данных и соблюдать требования к ним.

Основные преимущества совместного использования:

- 1) **Оптимизация по типу данных.** Каждая СУБД используется для тех типов данных, с которыми она работает наиболее эффективно.
- 2) **Масштабируемость.** Возможность масштабировать компоненты системы независимо друг от друга.
- 3) **Гибкость разработки.** Ускорение разработки благодаря использованию гибких инструментов для разных задач.

Однако при совместном использовании стоит учитывать следующие сложности:

- 1) **Интеграция данных.** Требуется обеспечить согласованность данных между двумя СУБД, что может быть реализовано через уровень приложения и чёткое разделение ответственности.
- 2) **Управление системой.** Необходимость поддержки и администрирования двух различных систем.
- 3) **Безопасность и авторизация.** Разработка единой системы аутентификации и авторизации для доступа к данным в обеих базах данных.

1.4 Методы сетевого многопользовательского взаимодействия

При создании системы для совместной работы нужно помнить, что пользователи могут одновременно редактировать одни и те же данные (заметки, задачи, схемы), поэтому система должна обеспечивать согласованность данных и актуальность их отображения для всех участников. Рассмотрим ключевые аспекты и методы, обеспечивающие такое взаимодействие.

1.4.1 Архитектуры взаимодействия

Существуют две основные архитектуры для построения сетевого взаимодействия.

Централизованная (Клиент-Серверная)

Наиболее распространенная архитектура для веб-приложений. Все клиенты (браузеры пользователей) подключаются к центральному серверу или кластеру серверов. Сервер хранит основные данные, обрабатывает запросы клиентов, управляет логикой приложения и обеспечивает синхронизацию данных между пользователями.

Из преимуществ можно выделить относительную простоту управления состоянием и согласованностью данных, так как сервер является единым источником истины, и простоту реализации контроля доступа и безопасности. Но сервер может стать узким местом по производительности при отсутствии масштабирования. Еще одним недостатком является единая точка отказа.

Децентрализованная (Peer-to-Peer, P2P)

Узлы (пользователи) взаимодействуют напрямую друг с другом, без центрального сервера, или с минимальным его участием, например, для обнаружения узлов. Каждый узел хранит копию данных или ее часть и обменивается изменениями с другими узлами.

К достоинствам можно отнести высокую отказоустойчивость, так как нет единой точки отказа, потенциально лучшую масштабируемость для некоторых задач и меньшую нагрузку на центральную инфраструктуру. Однако в такой системе значительно сложнее обеспечить согласованность данных, разрешать

конфликты и гарантировать безопасность.

Для большинства систем совместной работы клиент-серверная архитектура является более предпочтительной из-за упрощения задач синхронизации и управления. Однако элементы P2P могут использоваться для специфических функций, например, WebRTC для видеозвонков в чатах.

1.4.2 Протоколы и подходы к обмену данными

На прикладном уровне для взаимодействия клиента и сервера используются различные протоколы и подходы:

- 1) **HTTP/S с REST API или GraphQL.** Клиент отправляет запросы (GET, POST, PUT, DELETE и т.д.) на сервер для получения или изменения данных. REST является стандартным архитектурным стилем. GraphQL предлагает более гибкий подход к запросу данных клиентом. Подходит для операций, не требующих немедленного отклика у других пользователей, например, сохранение профиля, создание нового проекта.
- 2) **WebSockets.** Протокол, обеспечивающий постоянное двунаправленное соединение между клиентом и сервером. Подходит для функций, требующих обмена данными в реальном времени, таких как совместное редактирование документов, мгновенные уведомления, чаты, отображение статуса присутствия пользователей. Сервер может отправлять данные клиенту по своей инициативе, без запроса от клиента, что критично для real-time функциональности.
- 3) **Базовый транспортный протокол.** Как правило, для надежной передачи данных в клиент-серверных веб-приложениях используется TCP, который гарантирует доставку пакетов в правильном порядке и контроль целостности, что важно для синхронизации данных. UDP используется реже, в основном там, где допустима потеря пакетов ради скорости, например, в некоторых играх или потоковом видео.

Учитывая необходимость обеспечения функций совместной работы, чата и мгновенных уведомлений в реальном времени, для разрабатываемого комплекса в качестве основного механизма синхронизации состояния между клиентами и сервером выбран протокол WebSockets. Для стандартных

операций с данными, не требующих немедленной синхронизации, будет использоваться HTTP/S с применением REST-подхода.

1.5 Формализация задачи создания комплекса

1.5.1 Исходные задачи

программно–алгоритмического комплекса

Исходные задачи программно–алгоритмического комплекса включают:

- 1) **Регистрация и аутентификация пользователей** – предоставление возможности пользователям создавать учетные записи, входить в систему и управлять своими профилями.
- 2) **Создание и управление проектами** – пользователи могут создавать проекты, настраивать их параметры, а также управлять доступом к ним.
- 3) **Добавление пользователей в проекты** – возможность приглашать других пользователей в проекты, назначать роли и права доступа.
- 4) **Совместное редактирование документов** – предоставление инструментов для создания и редактирования различных типов документов внутри проектов:
 - 1) **Записки** – создание текстовых документов с возможностью форматирования и вставки элементов.
 - 2) **Схемы** – создание и редактирование диаграмм и схем в режиме онлайн.
 - 3) **Онлайн-презентации** – создание и демонстрация презентаций внутри проекта.
 - 4) **База знаний** – формирование и поддержка базы знаний проекта.
 - 5) **Графики и диаграммы** – визуализация данных в виде графиков и диаграмм.
 - 6) **Mind map** – создание интеллектуальных карт для структурирования идей.

5) **Дополнительные сервисы проекта:**

- 1) **Календарь** – интеграция календаря для планирования мероприятий и событий проекта.
- 2) **Трекер задач** – управление задачами и отслеживание их выполнения.
- 3) **Чат** – реализация коммуникации между участниками проекта в режиме реального времени.

6) **Обеспечение безопасности данных** – защита данных пользователей и проектов от несанкционированного доступа.

7) **Масштабируемость системы** – возможность системы эффективно по времени работать при увеличении количества пользователей и проектов.

8) **Модульная система для добавления новых сущностей и сервисов** – разработка и интеграция модульного подхода, который позволит расширять функциональность системы. Данный подход обеспечивает возможность:

- 1) **Добавления новых сущностей в базу данных** – гибкое изменение схемы данных для интеграции новых типов данных и сущностей без необходимости модификации основной структуры базы данных.
- 2) **Создания новых сервисов** – разработка дополнительных инструментов и служб, что позволяет адаптировать систему под изменяющиеся требования пользователей и проектов.

1.5.2 Диаграмма вариантов использования

Диаграмма вариантов использования представляет основные функции системы и взаимодействия пользователей с ней. Опишем основные варианты использования программно-алгоритмического комплекса:

- 1) Регистрация в системе.
- 2) Вход в систему (аутентификация).
- 3) Просмотр и редактирование своего профиля.
- 4) Создание проекта.
- 5) Приглашение других пользователей в проект.
- 6) Создание и редактирование:
 - 1) Записок.
 - 2) Схем.
 - 3) Презентаций.
 - 4) Mind map.
- 7) Использование дополнительных сервисов:
 - 1) Календарь.
 - 2) Трекер задач.
 - 3) Чат.
- 8) Управление правами доступа участников проекта.
- 9) Просмотр и участие в проектах, в которые он добавлен.

На рисунке 1.1 представлена общая диаграмма вариантов использования сервиса.

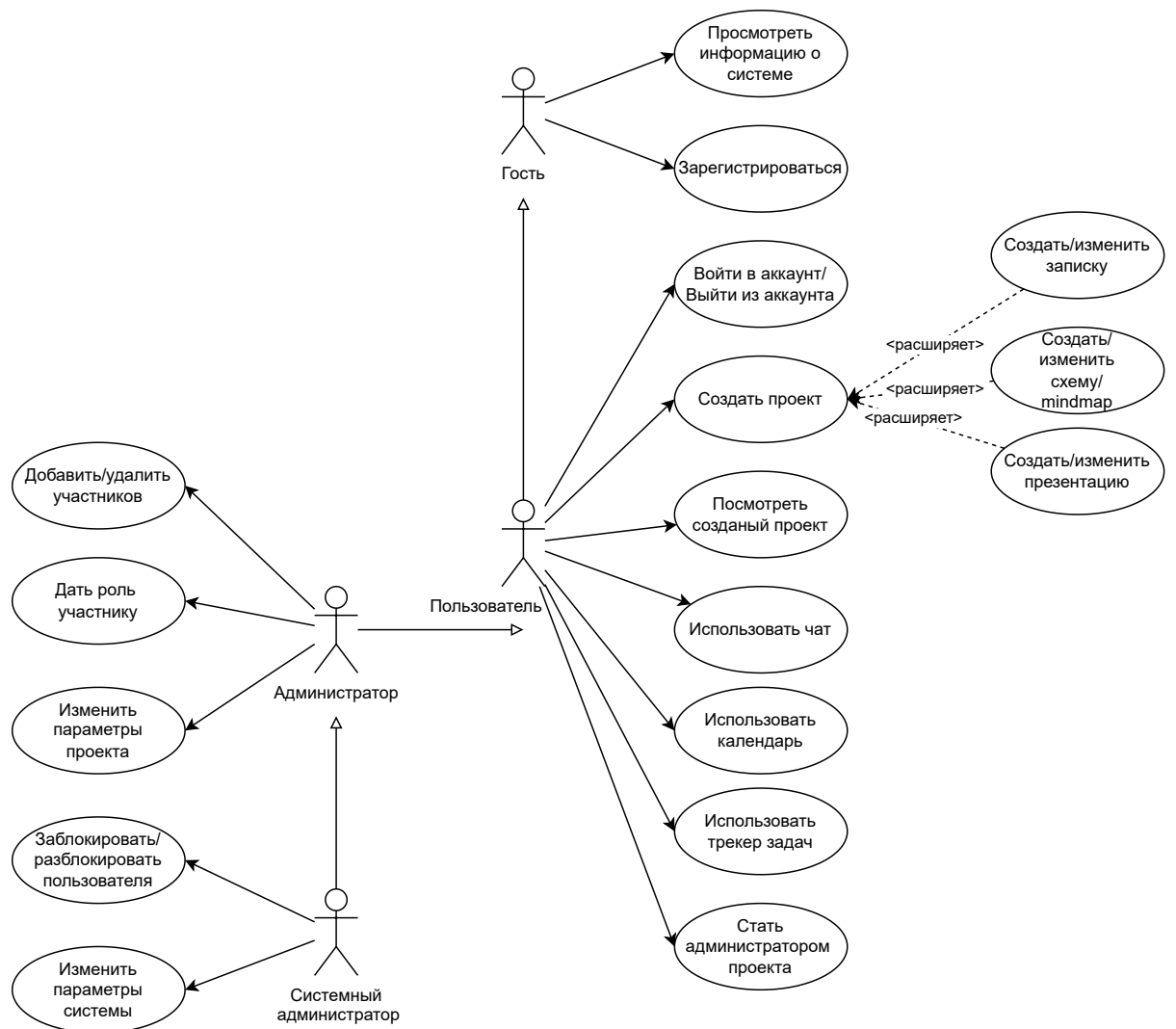


Рисунок 1.1 – Диаграмма вариантов использования сервиса

На рисунке 1.2 представлена диаграмма вариантов использования записок.



Рисунок 1.2 – Диаграмма вариантов использования записок

На рисунке 1.3 представлена диаграмма вариантов использования схем и mind map.

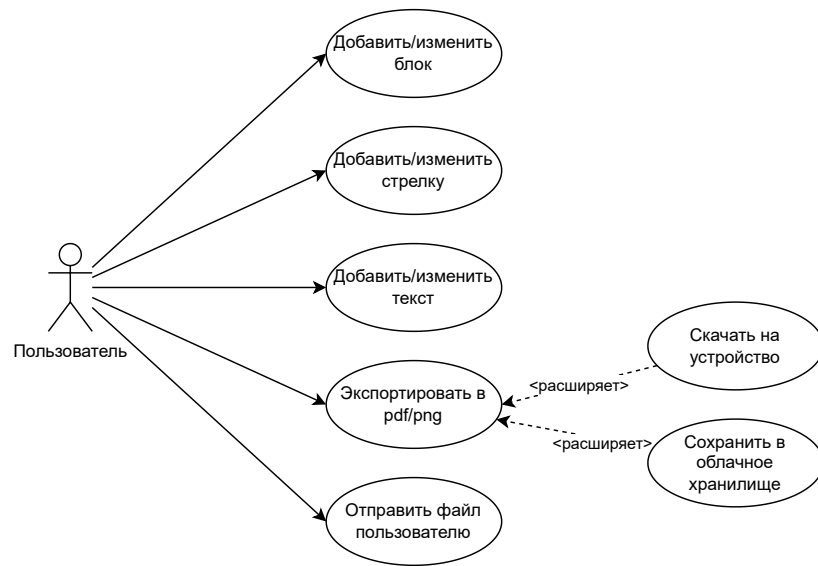


Рисунок 1.3 – Диаграмма вариантов использования схем и mind map

На рисунке 1.4 представлена диаграмма вариантов использования календаря.

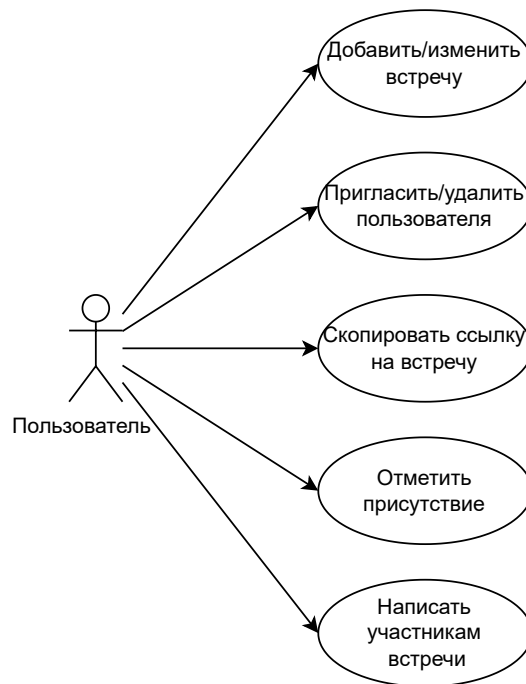


Рисунок 1.4 – Диаграмма вариантов использования календаря

На рисунке 1.5 представлена диаграмма вариантов использования трекера задач.

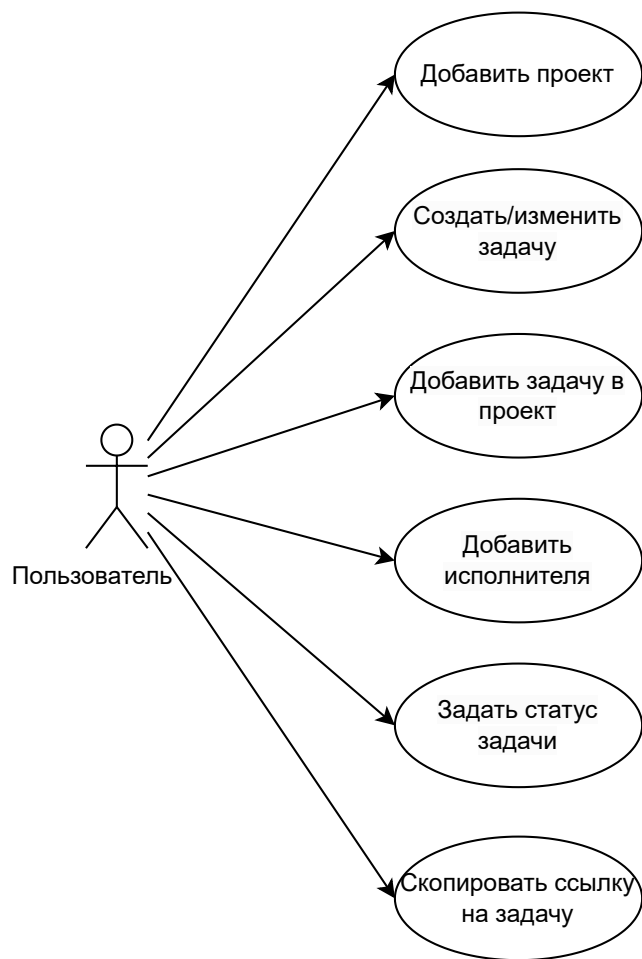


Рисунок 1.5 – Диаграмма вариантов использования трекера задач

На рисунке 1.6 представлена диаграмма вариантов использования чата.



Рисунок 1.6 – Диаграмма вариантов использования чата

На рисунке 1.7 представлена диаграмма вариантов использования презентаций.

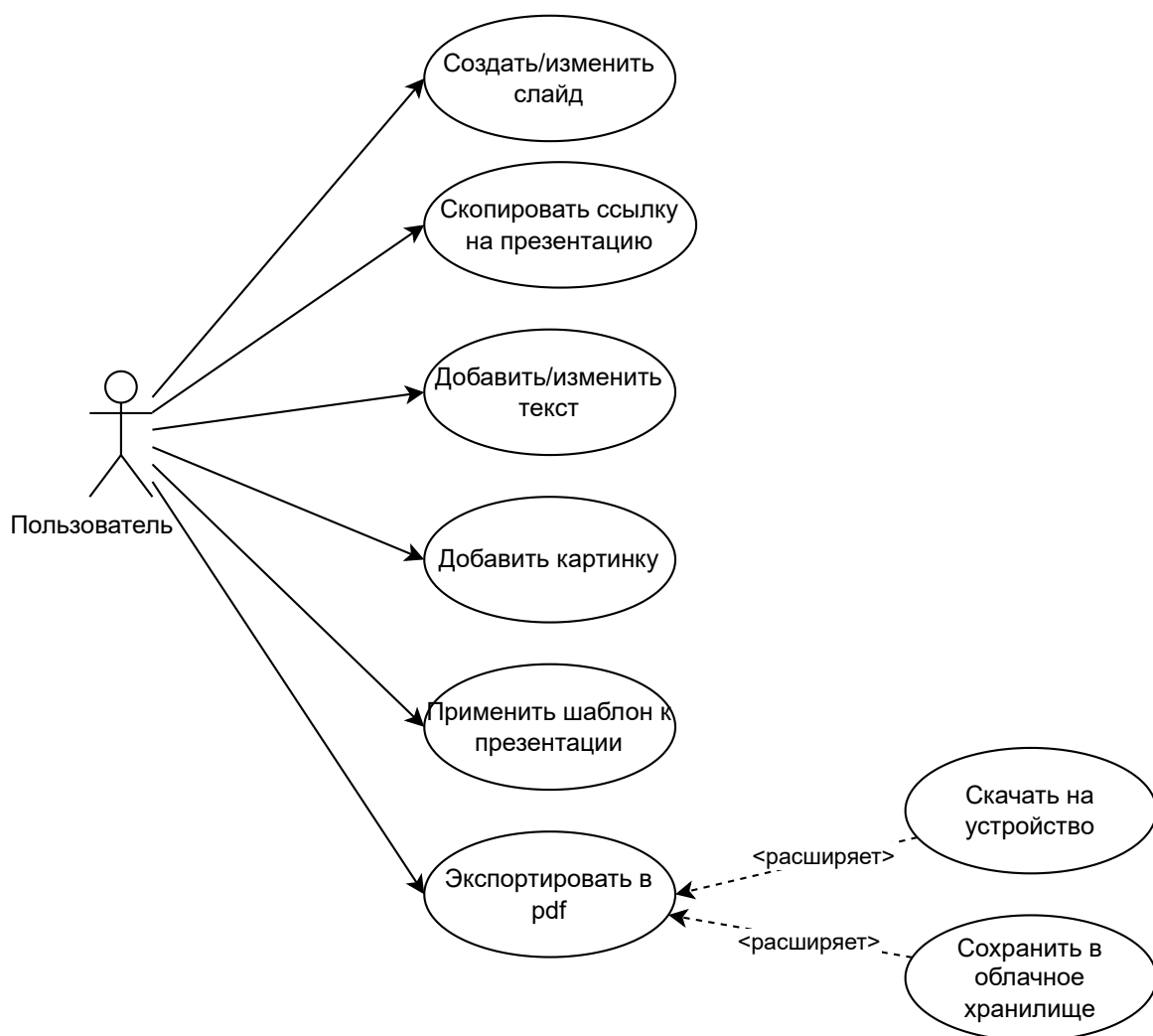


Рисунок 1.7 – Диаграмма вариантов использования презентаций

1.5.3 Анализ требований пользователей и ролей

Для реализации системы необходимо определить основные роли пользователей и их требования к системе.

Роли пользователей

Первично предполагаются следующие роли в системе:

- 1) **Гость** — пользователь, не зарегистрированный в системе. Имеет ограниченный доступ, может просматривать публичную информацию о системе.
- 2) **Зарегистрированный пользователь** имеет учетную запись в системе. Может создавать проекты, присоединяться к проектам, использовать доступные инструменты.
- 3) **Администратор проекта** — пользователь, создавший проект или назначенный администратором. Имеет расширенные права на управление проектом и его участниками.
- 4) **Системный администратор** отвечает за общую поддержку и администрирование системы, управление пользователями, настройку прав доступа.

Требования пользователей

1. Гость

- 1) Возможность просматривать информацию о системе и её возможностях.
- 2) Возможность зарегистрироваться в системе.

2. Зарегистрированный пользователь

- 1) Возможность входа в систему с использованием учетных данных.
- 2) Создание новых проектов.

- 3) Просмотр и участие в проектах, в которые пользователь добавлен.
- 4) Использование инструментов для создания и редактирования контента (записки, схемы, презентации и т.д.).
- 5) Участие в коммуникациях проекта (чат).
- 6) Просмотр календаря и задач проекта.

3. Администратор проекта

- 1) Все возможности зарегистрированного пользователя.
- 2) Управление участниками проекта (добавление/удаление пользователей).
- 3) Назначение ролей и прав доступа участникам проекта.
- 4) Настройка параметров проекта.

4. Системный администратор

- 1) Управление пользователями системы (блокировка/разблокировка учетных записей, восстановление доступа).
- 2) Настройка глобальных параметров системы.

Вывод

Современные системы управления базами данных (СУБД) подразделяются на дореляционные, реляционные и постреляционные, каждая из которых отвечает различным требованиям к хранению данных — от строгих структурированных до более гибких моделей. Для хранения неоднородных данных, таких как в системах совместной работы над проектами, выбор конкретной модели определяется специфическими потребностями. Например, реляционные базы данных подходят, когда требуются строгая схема и сложные связи между сущностями, как в случае данных о пользователях. В то же время, нереляционные базы, такие как MongoDB, обеспечивают гибкость и масштабируемость при работе с полуструктурированными данными в формате JSON.

Сравнительный анализ показал, что сочетание MongoDB и PostgreSQL позволяет использовать преимущества каждой модели, обеспечивая надежность и целостность структурированных данных, что является важным для адаптации системы к нуждам проекта. Однако это требует тщательного планирования, интеграции и управления сложностями.

Для разрабатываемого комплекса была выбрана клиент-серверная архитектура как обеспечивающая большую простоту управления состоянием и безопасностью. Были определены основные протоколы взаимодействия: HTTP/S с REST для стандартных запросов и WebSockets для обеспечения функциональности в реальном времени (совместное редактирование, чаты, уведомления).

В результате формализации задачи были определены исходные задачи программно-алгоритмического комплекса, охватывающие основные функциональные возможности системы совместной работы над проектами, а также был проведен анализ ролей пользователей.

2 Конструкторский раздел

2.1 Основные положения предлагаемого программно-алгоритмического комплекса

При проектировании программно-алгоритмического комплекса были заложены следующие основные положения, направленные на обеспечение его функциональности, масштабируемости и многоцелевого использования:

1) **Клиент-серверная архитектура:** Комплекс реализуется в рамках клиент-серверной архитектуры, где пользователь взаимодействует с системой через клиентское приложение (веб-браузер), а основная бизнес-логика, управление данными и синхронизация выполняются на серверной стороне.

2) **Гибридная модель хранения данных:** Учитывая необходимость работы с неоднородными данными, применяется гибридный подход к хранению:

- **PostgreSQL** используется для хранения структурированных, реляционных данных, требующих строгой схемы, целостности и поддержки транзакций. Сюда относятся данные о пользователях, проектах, файлах (метаданные), типах файлов, а также связи между ними (участие пользователей в проектах, принадлежность файлов проектам, роли и права доступа).
- **MongoDB** используется для хранения полуструктурированных и неструктурированных данных, требующих гибкости схемы и горизонтальной масштабируемости. В MongoDB хранится фактическое содержимое файлов, состоящее из различных блоков данных, а также информация о стилях.

3) **Комбинированное использование REST API и WebSockets:** Для взаимодействия между клиентом и сервером применяются два основных механизма:

- **REST API** используется для выполнения стандартных CRUD-операций (создание, чтение, обновление, удаление) над основными

сущностями системы (пользователи, проекты, файлы и т.д.), а также для запроса или модификации данных, не требующих немедленной синхронизации у других пользователей.

- **WebSocket** используется для обеспечения взаимодействия в реальном времени важного для функций совместной работы: одновременное редактирование документов, обмен сообщениями в чате, доставка мгновенных уведомлений, отображение статуса присутствия пользователей.

4) **Модульность серверного приложения:** Серверное приложение проектируется с учетом модульности, где каждая основная функциональная область (управление пользователями, проектами, контентом и т.д.) выделяется в логический компонент, что упрощает разработку, тестирование и дальнейшее развитие системы.

5) **Масштабируемость:** Архитектура и выбор технологий (в частности, использование MongoDB и возможность репликации/масштабирования PostgreSQL) закладывают основу для потенциальной горизонтальной и вертикальной масштабируемости системы при росте нагрузки.

2.2 Структура и компоненты программного приложения

Программно-алгоритмический комплекс строится на основе многоуровневой архитектуры, близкой к принципам чистой или луковой архитектуры, с четким разделением ответственности между слоями. Общая структура представлена на рисунке 2.1.

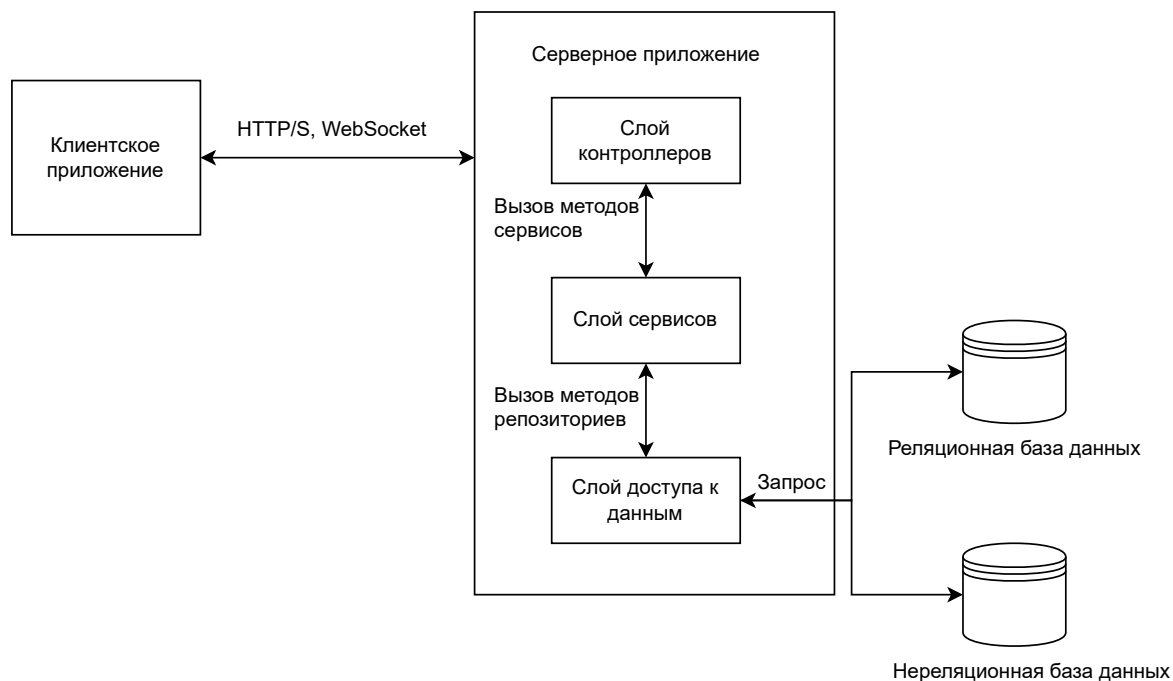


Рисунок 2.1 – Общая архитектура программно-алгоритмического комплекса

Основные компоненты и слои архитектуры:

- 1) **Клиентское приложение** будет реализовано как одностраничное веб-приложение (SPA), отвечающее за пользовательский интерфейс, визуализацию данных и взаимодействие с пользователем. Обмен данными с серверной частью будет осуществляться через REST API и WebSocket.
- 2) **Серверное приложение** реализует бизнес-логику и будет состоять из следующих слоев:
 - **Слой Контроллеров**, который отвечает за обработку входящих HTTP-запросов и вызов методов соответствующего сервиса. В этот же слой логически входит обработка WebSocket-соединений для real-time взаимодействия.
 - **Слой Сервисов**, который инкапсулирует основную бизнес-логику приложения. Сервисы координируют работу репозиторий, выполняют преобразование данных, реализуют правила предметной области и управляют транзакциями.
 - **Слой Доступа к Данным**, включающий интерфейсы репозиторий и абстрагирующий детали взаимодействия с базами данных,

предоставляя методы для CRUD-операций.

- 3) **Система хранения реляционных данных** хранит метаданные и связи между основными сущностями системы.
- 4) **Система хранения документных данных:** хранит гибкое содержимое файлов и связанные с ним стили.

На рисунке 2.2 представлена более подробная диаграмма компонентов серверного приложения.

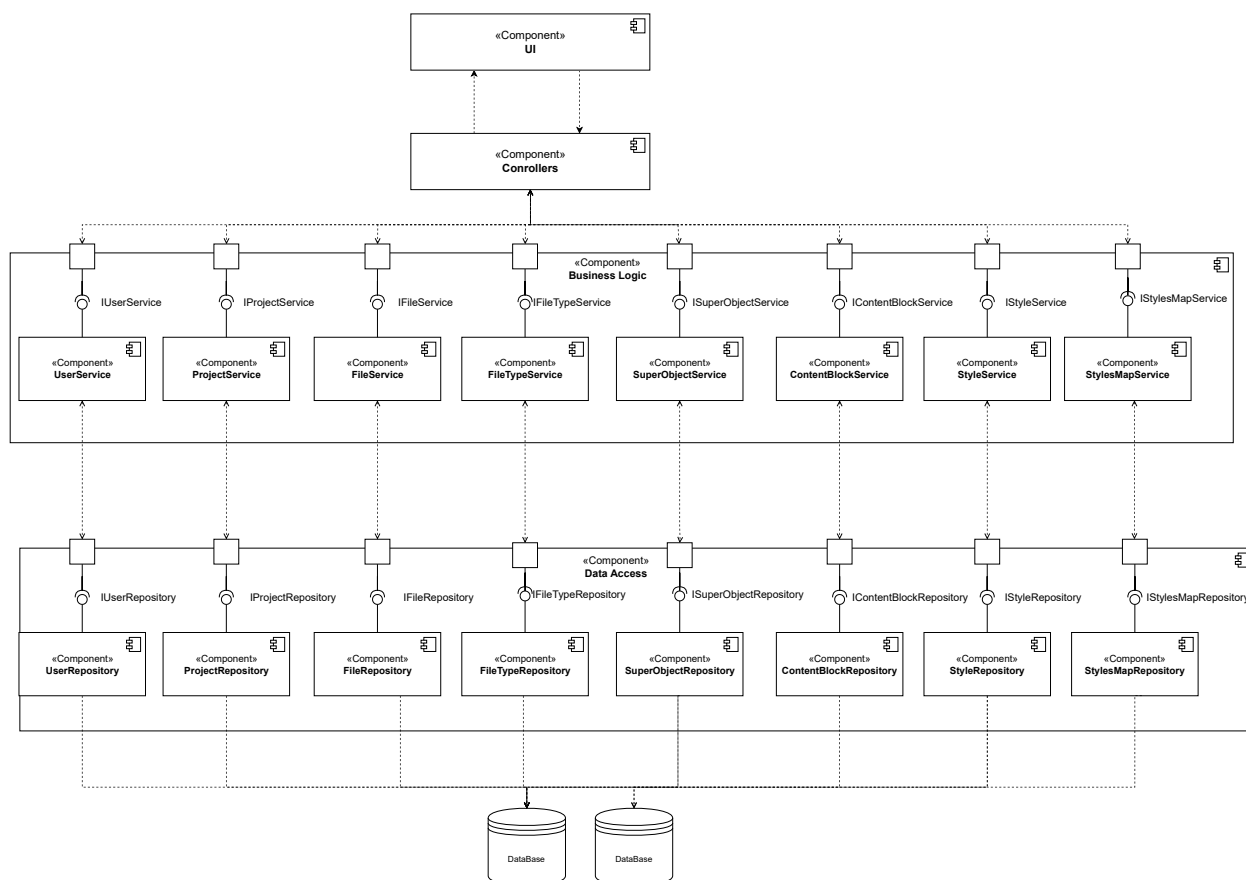


Рисунок 2.2 – Диаграмма компонентов серверного приложения

2.3 Сценарий совместного редактирования

Для иллюстрации взаимодействия компонентов системы при выполнении операций в реальном времени рассмотрим сценарий совместного редактирования документа. Процесс представлен диаграммой последовательности на рисунке 2.3.

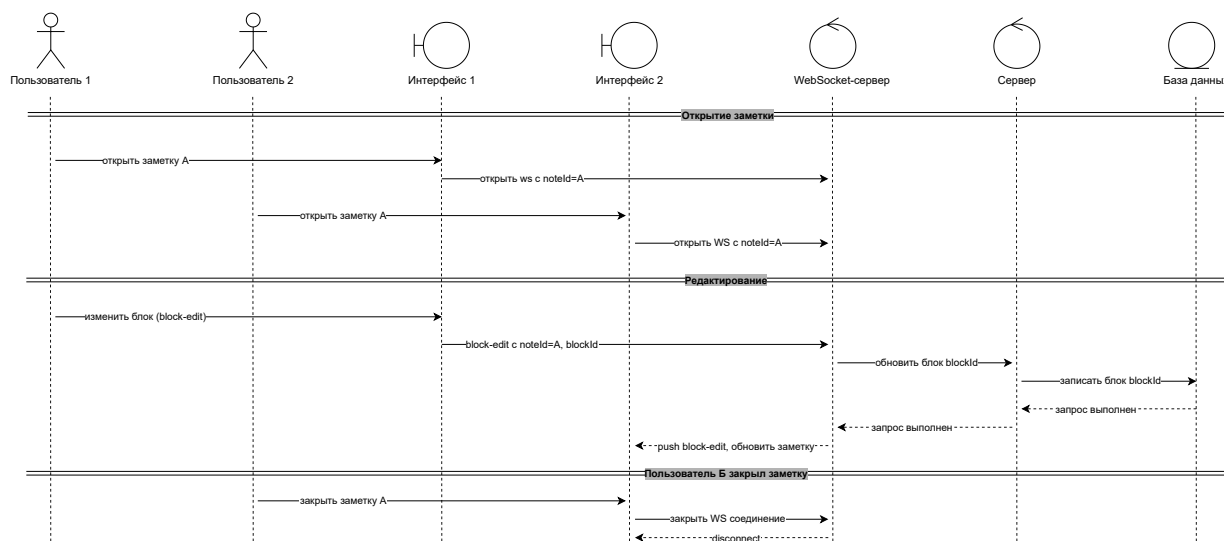


Рисунок 2.3 – Диаграмма последовательности: Совместное редактирование документа

Основные шаги взаимодействия при внесении изменения пользователем 1 в документ, который также просматривает пользователь 2:

- 1) Пользователь 1 выполняет действие редактирования (например, вводит символ) в клиентском приложении 1.
- 2) Клиентское приложение 1 формирует сообщение об операции редактирования, указывая тип операции, позицию, измененные данные и идентификатор документа/файла, и отправляет его на сервер через установленное WebSocket-соединение.
- 3) Серверное приложение, в частности его компонент, отвечающий за обработку WebSocket-сообщений, принимает данное сообщение.
- 4) Обработчик WebSocket вызывает соответствующий метод сервисного слоя, ответственного за управление контентом, передавая ему детали операции редактирования.
- 5) Сервис контента проверяет полученную операцию и взаимодействует со слоем доступа к данным для сохранения изменений в соответствующей системе хранения.
- 6) После подтверждения успешного сохранения данных от слоя доступа к данным, компонент обработки WebSocket-сообщений получает уведомление об успешном применении операции.

- 7) Компонент обработки WebSocket-сообщений рассылает информацию о выполненной операции всем остальным пользователям, включая пользователя 2, которые в данный момент работают с этим же документом и подключены к соответствующей WebSocket-сессии.
- 8) Клиентское приложение 2 получает сообщение об операции через WebSocket и применяет соответствующие изменения к своему локальному представлению документа, отображая актуальное состояние пользователя 2.

Данный механизм обеспечивает синхронизацию состояния документа у всех активных пользователей в режиме реального времени.

2.4 Ключевые структуры данных

Для хранения информации выбраны две модели данных: реляционная модель в PostgreSQL и документная модель в MongoDB.

2.4.1 Реляционная модель данных

Структура реляционной базы данных предназначена для хранения основной метаинформации о сущностях системы и связей между ними. Основные таблицы:

- **users:** Информация о пользователях (id, name, surname, login, password(hash), photo).
- **projects:** Информация о проектах (id, author_id, date, name).
- **file_types:** Справочник типов файлов (id, name).
- **files:** Метаинформация о файлах (id, name, type_id, author_id, date).
- **projects_users:** Связь пользователей с проектами, включая роли и разрешения (id, project_id, user_id, role, permission).
- **projects_files:** Связь файлов с проектами (id, project_id, file_id).

Эта структура обеспечивает ссылочную целостность, транзакционность и возможности для сложных запросов с объединением данных. Полная реляционная схема базы данных представлена на рисунке 2.4.

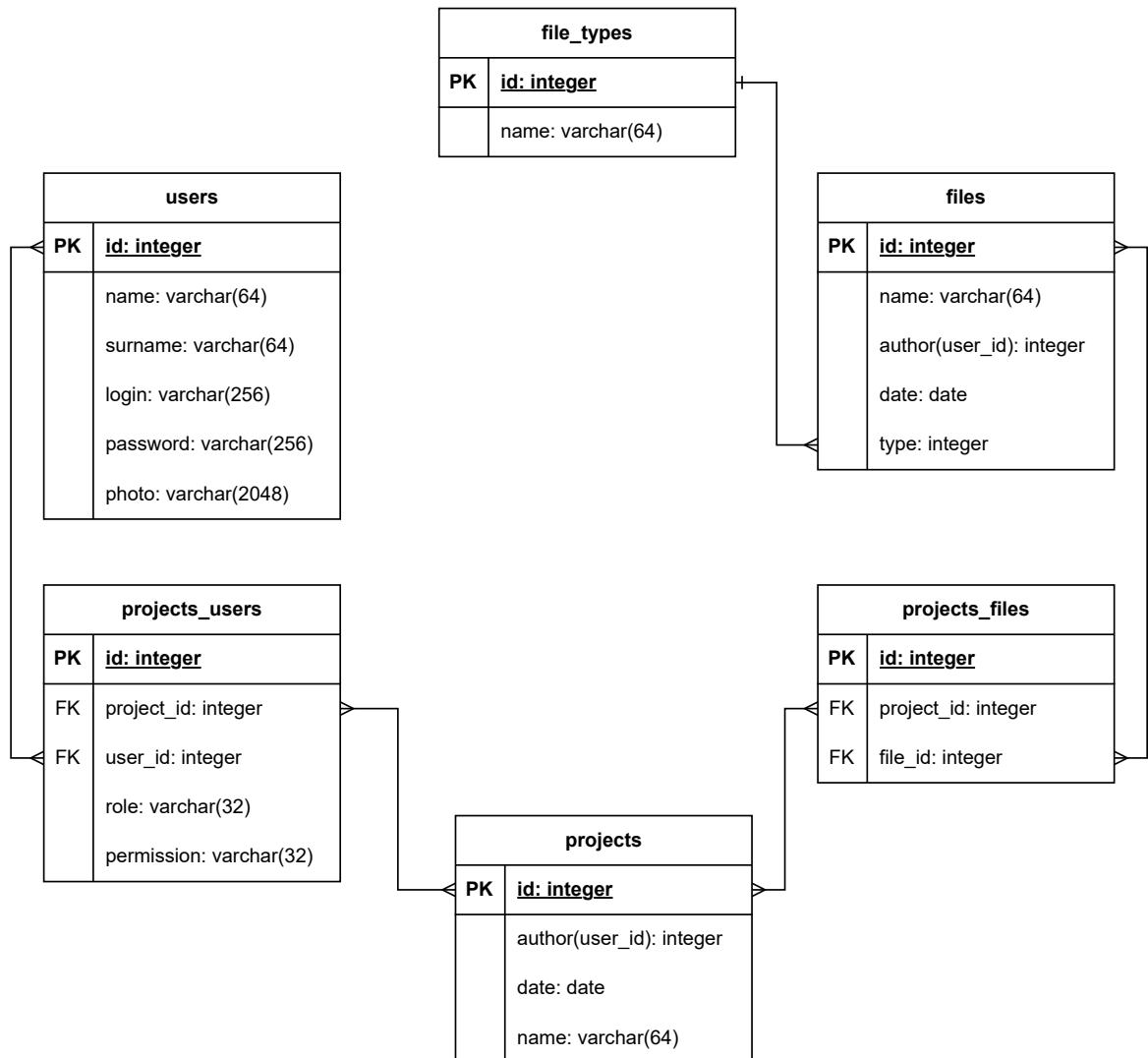


Рисунок 2.4 – Диаграмма реляционной базы данных

2.4.2 Документная модель данных (MongoDB)

Структура данных в нереляционной базе данных предназначена для хранения содержимого файлов и связанной с ним информации. Используются коллекции документов, структура которых может гибко изменяться.

Основные коллекции:

- **super_objects:** Корневой документ для содержимого файла. Содержит `fileId` (связь с реляционной базой), метаданные контента (`serviceType`, `lastChangeDate`, `name`), ссылки на первый и последний блоки (`firstItem`, `lastItem`), карту стилей (`stylesMapId`) и другую служебную информацию (`template`, `decoration`, `checksum`).
- **content_blocks:** Представляет собой отдельные блоки контента (текст, список, изображение и т.д.), связанные в последовательность через `prevItem` и `nextItem`. Хранит фактические данные блока (`data`, `items`, `label`) и его тип (`objectType`).
- **styles:** Определения стилей с различными атрибутами форматирования.
- **styles_maps:** Карты стилей, содержащие массив ссылок `links`, указывающих, какой стиль (`styleId`) применяется к какому блоку (`textId`) и в каком диапазоне (`start`, `end`).

Эта модель обеспечивает гибкость для хранения сложного и разнообразного контента.

Схема коллекций этой базы данных представлена на рисунке 2.5.

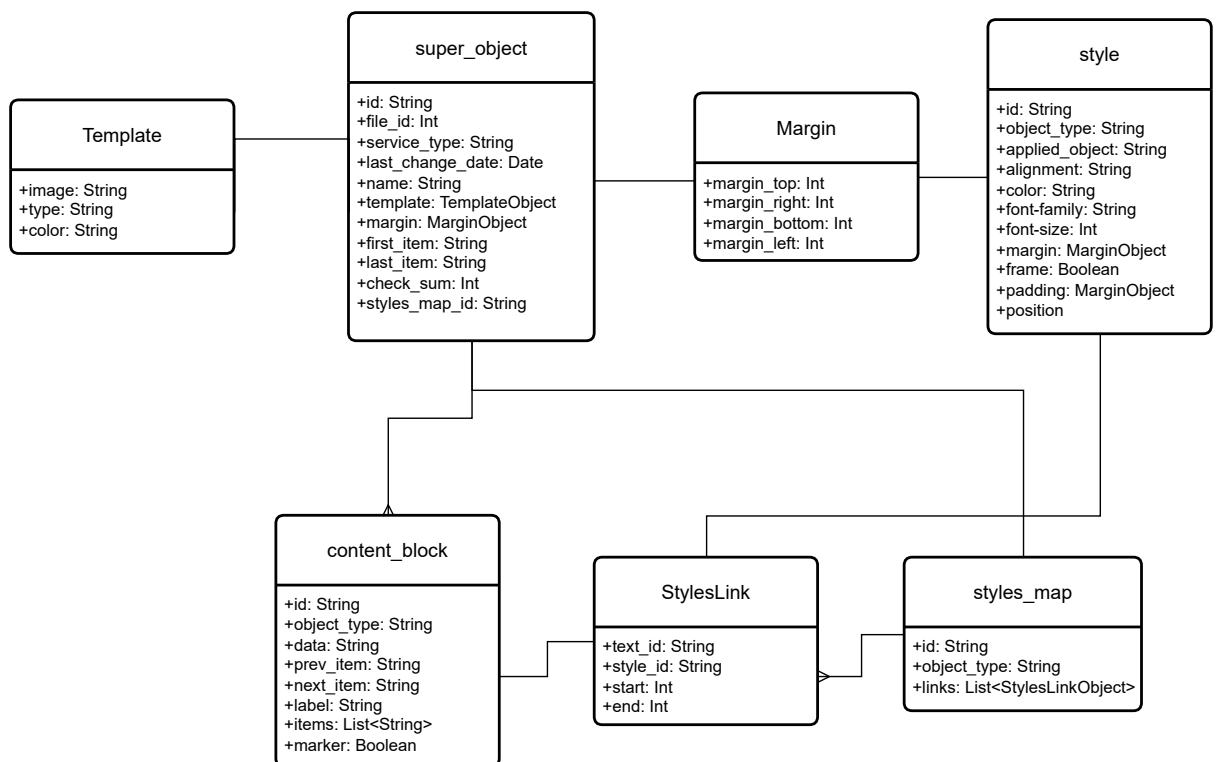


Рисунок 2.5 – Схема коллекций нереляционной базы данных

Вывод

В рамках конструкторского раздела были разработаны и описаны основные архитектурные решения для программно-алгоритмического комплекса совместной работы и управления проектами.

Определены ключевые положения, включая выбор клиент-серверной архитектуры, гибридной модели хранения данных с использованием PostgreSQL для структурированной информации и MongoDB для гибкого контента, а также комбинированного подхода к взаимодействию через REST API и WebSocket для поддержки реального времени. Представлена общая структура приложения, выделяющая клиентскую и серверную части, а также многоуровневую организацию серверного приложения (контроллеры, сервисы, доступ к данным), что способствует модульности и разделению ответственности.

Описаны основные компоненты серверного приложения (сервисы управления пользователями, проектами, файлами, контентом и др.) и принципы их взаимодействия, проиллюстрированные на примере сценария совместного редактирования с помощью диаграммы последовательности. Зафиксированы

ключевые структуры данных для реляционной (PostgreSQL) и документной (MongoDB) моделей, обеспечивающие хранение всей необходимой информации — от метаданных до сложного содержимого файлов и стилей.

3 Технологический раздел

4 Исследовательский раздел

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *С. А. Нестеров*. Базы данных: учеб. пособие. — СПб.: Издательство Политехнического университета, 2013. — С. 150.
2. *Н. Р. Бухараев*. Введение в реляционные базы данных и программирование на языке SQL. — Казань: Казанский университет, 2018. — С. 134.
3. *Ю. Л. Назаренко*. Обзор технологии "большие данные"(Big Data) и программно-аппаратных средств, применяемых для их анализа и обработки. — Донской государственный технический университет, 2016. — С. 7.
4. *Р. Э. Мамедли*. Системы управления базами данных: учебное пособие. — Нижневаторск : Издательство Нижневаторского государственного университета, 2021. — С. 214.
5. MongoDB Documentation. — [Электронный ресурс]. — Режим доступа: <https://www.mongodb.com/docs/manual/> (дата обращения: 03.11.2024).
6. Cassandra Documentation. — [Электронный ресурс]. — Режим доступа: <https://cassandra.apache.org/doc/latest/> (дата обращения: 03.11.2024).
7. PostgreSQL Documentation. — [Электронный ресурс]. — Режим доступа: <https://www.postgresql.org/docs/> (дата обращения: 03.11.2024).
8. MySQL Documentation. — [Электронный ресурс]. — Режим доступа: <https://dev.mysql.com/doc/> (дата обращения: 03.11.2024).
9. *Wen Tao*. Data Aggregation: Encyclopedia of Big Data. — Springer International Publishing, Cham, 2020.
10. *С. Д. Кузнецов, А. В. Посконин*. Распределенные горизонтально масштабируемые решения для управления данными. — Труды Института системного программирования РАН, 2013.
11. *A. Silberschatz, F. Korth Henry, S. Sudarshan*. DATABASE SYSTEM CONCEPTS, SEVENTH EDITION. — McGraw-Hill Education, 2020. — С. 1337.

12. Документация PostgreSQL: Replication. — [Электронный ресурс]. — Режим доступа: <https://www.postgresql.org/docs/current/runtime-config-replication.html> (дата обращения: 20.11.2024).
13. Citus Data Documentation. — [Электронный ресурс]. — Режим доступа: <https://docs.citusdata.com/en/stable/> (дата обращения: 20.11.2024).
14. Postgres-XL Documentation. — [Электронный ресурс]. — Режим доступа: https://omnidb.readthedocs.io/en/2.17.0/en/18_postgres-xl.html (дата обращения: 20.11.2024).
15. PgPool Documentation. — [Электронный ресурс]. — Режим доступа: https://www.pgpool.net/mediawiki/index.php/Main_Page (дата обращения: 20.11.2024).
16. PgBouncer Documentation. — [Электронный ресурс]. — Режим доступа: <https://www.pgouncer.org/config.html> (дата обращения: 20.11.2024).

ПРИЛОЖЕНИЕ А