«Облачное хранилище файлов с доступом через Телеграм бота»

[**ВВЕДЕНИЕ 3**](#_z59jo9p8f9lk)

[**1. ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ 3**](#_a5ug49so4dy)

[1.1 Обзор литературы и документации, необходимой для разработки 3](#_4n49jpj8avw7)

[1.2 Программные средства разработки 4](#_wskh2qcxbwre)

[1.3. Описание архитектуры системы 5](#_m9md9gcusjgz)

[1.4. Выбор технологий для разработки архитектуры веб-сервиса 6](#_9jtbfbjl5xx)

[1.5. Проектирование базы данных 11](#_5fxegjrmzes4)

[1.6.1. Структура и описание таблиц 12](#_y7aiwobegw7k)

[1.6.2. ER-диаграмма 15](#_9bxa10xvis2q)

[**2. РЕАЛИЗАЦИЯ ПРОГРАММНЫХ МОДУЛЕЙ 17**](#_9coemzu9zcdc)

[2.1. Разработка базы данных 17](#_r2hz5ujm7k64)

[2.2.1. Описание конечных точек для HTTP запросов к серверу 18](#_byl7ldo0ya44)

[2.3. Развертывание API 22](#_d7ws4gg37j2f)

[**ЗАКЛЮЧЕНИЕ 30**](#_pryg57faqnr7)

[**Список источников литературы 31**](#_yy7dadw85jc4)

# ВВЕДЕНИЕ

Разработка облачного хранилища с доступом через Телеграм-бота становится все более актуальной в современном мире цифровых технологий и удаленной работы. С ростом объема данных и необходимостью их безопасного хранения и совместного использования, такие хранилища представляют собой эффективное решение для управления файлами, обмена данными и упрощения взаимодействия пользователей.

Облачное хранилище с доступом через Телеграм-бота предоставляет пользователям возможность легкого доступа к своим файлам, их совместного использования и организации. Такое решение особенно полезно для работы в командах, когда требуется быстрая передача данных, а также для индивидуального использования, где важна простота работы с системой.

Подобные решения часто обеспечивают не только базовые функции хранения, но и широкий спектр возможностей, таких как автоматическая синхронизация данных, безопасность через шифрование и токены, а также аналитика использования для улучшения пользовательского опыта.

В условиях активного перехода на удаленную работу и необходимость эффективного управления данными облачное хранилище, интегрированное с популярным мессенджером, становится неотъемлемой частью современного рабочего процесса. Такое направление разработки программного обеспечения способствует повышению производительности, удобства работы и созданию безопасной цифровой среды для различных сфер деятельности.

# 1. ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ

## 1.1 Обзор литературы и документации, необходимой для разработки

Перед началом разработки мы изучили следующую литературу и документацию, необходимую для создания нашего программного продукта:

1. "Основы тестирования и верификации программного обеспечения" (Старолетов С. М.): Книга представляет основные принципы и методы тестирования и верификации программного обеспечения, включая техники тестирования, планирование тестирования, автоматизацию тестирования и анализ результатов.
2. "Проектирование информационных систем" (Вейцман В. ): Эта книга является подробным руководством по проектированию информационных систем, охватывая все этапы процесса от сбора требований до создания дизайна системы и ее реализации.
3. "SQL. Библия пользователя, 2-е издание" (Кригель Алекс, Трухнов Борис): Руководство предлагает исчерпывающую информацию по использованию языка SQL, включая основы запросов, создание баз данных, управление данными и оптимизацию запросов.
4. "Проектирование веб-API: практическое руководство" (Лоре, А. ): В данном руководстве автор представляет методы проектирования эффективных веб-API, описывая основные принципы RESTful API, форматирование данных, безопасность API и многое другое.
5. "Pro Spring 5: An In-Depth Guide to the Spring Framework and Its Tools" (Iuliana Cosmina, Rob Harrop, Chris Schaefer, Clarence Ho) - эта книга предоставляет обширное понимание Spring Framework, включая основы и продвинутые темы. Она также охватывает инструменты Spring, которые могут быть полезны при разработке приложений на Java.
6. "PostgreSQL High Performance Cookbook" (Chitij Chauhan) - эта книга предоставляет рецепты и советы по оптимизации и улучшению производительности PostgreSQL, что может быть полезно при проектировании и оптимизации базы данных.
7. "Clean Code: A Handbook of Agile Software Craftsmanship" (Robert C. Martin) - эта книга предоставляет практические рекомендации по написанию чистого, понятного и поддерживаемого кода, что является важным аспектом разработки программного обеспечения.
8. Spring Framework документация - (<https://docs.spring.io/spring-framework/docs/current/reference/html/>) - официальная документация Spring Framework содержит подробные объяснения основных компонентов и функциональности фреймворка, что поможет разработчикам эффективно использовать Spring в своих проектах.
9. PostgreSQL документация (https://www.postgresql.org/docs/) - официальная документация PostgreSQL содержит информацию о различных аспектах работы с этой базой данных, начиная от установки и настройки до оптимизации запросов.

## 1.2 Программные средства разработки

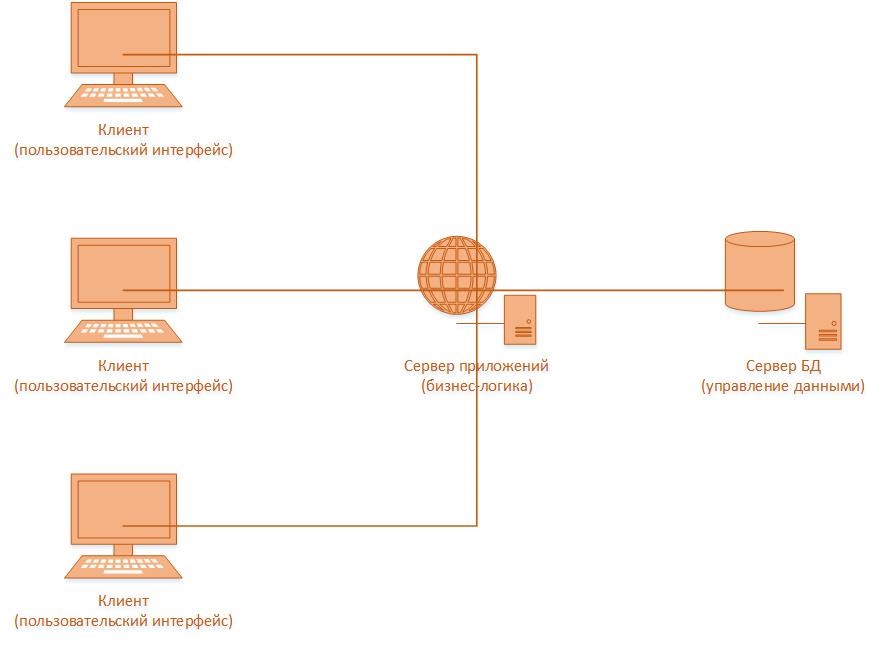
После изучения необходимой документации мы установили следующие программные средства для разработки и проектирования системы:

1. GitHub Desktop: Это графический интерфейс для работы с Git, который упрощает работу с версионным контролем и совместной разработкой программного обеспечения.
2. Visual Studio Code: Это легкий и мощный текстовый редактор, который удобен для разработки программного обеспечения. Он поддерживает множество языков программирования и имеет обширную экосистему расширений. Visual Studio Code будет использован для разработки графической части приложения на React.
3. Postman: Это инструмент для тестирования API. Он позволяет отправлять запросы к API, тестировать и отлаживать взаимодействие с сервером, а также автоматизировать тестирование API.
4. IntelliJ IDEA Ultimate: Это интегрированная среда разработки (IDE) для различных языков программирования, включая Java, Kotlin, Groovy и другие. IntelliJ IDEA Ultimate предоставляет мощные инструменты для написания, отладки и тестирования кода, а также поддерживает множество фреймворков и технологий. Эта IDE будет использована для разработки серверной части приложения на Spring Framework.
5. DbVisualizer: Это универсальный инструмент для управления базами данных. DbVisualizer поддерживает работу с различными СУБД и предоставляет удобный интерфейс для выполнения SQL-запросов, анализа схем баз данных, мониторинга производительности и управления данными. Он будет использован для визуализации базы данных.

## 1.3. Описание архитектуры системы

Веб-сервис будет построен на трехуровневой клиент-серверной архитектуре. Трехуровневая архитектура предполагает разделение на три логических и физических уровня: уровень представления (пользовательский интерфейс), сервер-приложений (бизнес-логика), и уровень данных, предназначенный для хранения и управления данными.

Схематически такую архитектуру можно представить, как показано на рис. 1.1:



*Рис. 1.1 Трехуровневая клиент-серверная архитектура*

Уровень представления

На этом уровне реализован пользовательский интерфейс, который отображает и получает информацию от клиентов. Если говорить о сайтах, то уровень представления работает в веб-браузерах пользователей. Интерфейс веб-приложений разрабатывается с помощью HTML, CSS и JavaScript. Если говорить от тг-ботах, то уровень представления работает в приложении Telegram.

Уровень приложений

На этом уровне обрабатывается информация, собранная на уровне представления с помощью бизнес-логики, которая представляет собой набор бизнес-правил. Уровень приложения может добавлять, изменять и удалять данные, расположенные на уровне данных.  
 Уровень приложения разработан с помощью Java и взаимодействует с уровнем данных посредством вызовов API.

Уровень данных

Этот уровень также еще называют уровнем базы данных. Он предназначен для хранения и управления информацией, которая обрабатывается приложением. Как правило, роль уровня данных выполняет реляционная СУБД в нашем случае это PostgreSQL и хранилище файлов MinIO. В трехуровневой архитектуре обмен данными происходит только через уровень приложения. Уровень представления и уровень данных не могут взаимодействовать друг с другом напрямую.

## 1.4. Выбор технологий для разработки архитектуры веб-сервиса

Уровень представления

Создание графического интерфейса для Telegram-бота, реализованного на Java, требует выбора эффективных инструментов и технологий. В данном случае Telegram-бот выступает в роли уровня представления (front-end), взаимодействующего с пользователем через текстовый интерфейс. Такой подход позволяет избежать сложностей традиционного веб-разработки и сосредоточиться на логике взаимодействия.

Описание технологии:

Telegram-бот, реализованный на Java, представляет собой интерфейс для взаимодействия с пользователями через платформу Telegram. Java используется как надежный и масштабируемый язык для реализации логики работы бота и обработки пользовательских запросов.

Достоинства Теlegram Бота:

1.Простота реализации пользовательского интерфейса:

Telegram предоставляет готовую платформу для взаимодействия с пользователем, что освобождает разработчиков от необходимости создавать сложные графические интерфейсы.

2.Гибкость и функциональность:

С помощью Telegram Bot API можно реализовать разнообразные элементы интерфейса: кнопки, клавиатуры, инлайн-меню и другие интерактивные компоненты. Это позволяет создать удобный и функциональный интерфейс.

3.Масштабируемость:

Java как язык разработки обеспечивает стабильную работу бота даже при большом количестве пользователей, благодаря встроенным средствам многопоточности и управлению ресурсами.

4.Экосистема Telegram:

Telegram предоставляет готовую инфраструктуру для доставки сообщений, что упрощает разработку и ускоряет процесс создания приложения.

5.Кроссплатформенность и доступность:

Telegram-бот доступен пользователям на любой платформе, где установлен Telegram (iOS, Android, Windows, macOS, веб). Это значительно расширяет аудиторию приложения.

6.Расширяемость через Java-библиотеки:

Использование Java позволяет подключать сторонние библиотеки для обработки данных, интеграции с базами данных, API других сервисов и реализации бизнес-логики.

Таким образом, реализация уровня представления через Telegram-бот на Java позволяет сосредоточиться на функциональности приложения, избегая лишних затрат на создание сложного пользовательского интерфейса. Такой подход сочетает в себе надежность, удобство и высокую производительность.

Уровень приложений

Уровень приложения также называют back-end. Для разработки серверной части веб-сервиса был выбран язык программирования Java в связке с фреймворком Spring.

При разработке веб-приложений критически важно выбрать подходящий фреймворк для создания надежной, масштабируемой и производительной серверной части. Spring Framework является одним из самых популярных и широко используемых фреймворков для разработки back-end на языке Java. Этот выбор был обусловлен несколькими ключевыми факторами, которые мы подробно рассмотрим ниже.

1. Модульность и Комплексная Поддержка

Spring предоставляет богатый набор модулей, позволяющих решать разнообразные задачи в разработке веб-приложений. Среди наиболее значимых модулей можно выделить:

* Spring Core: обеспечивает основу для управления зависимостями и инверсии управления (IoC).
* Spring MVC: предоставляет мощный и гибкий фреймворк для разработки веб-приложений на основе архитектуры Model-View-Controller (MVC).
* Spring Data: упрощает взаимодействие с различными базами данных, предоставляя унифицированные интерфейсы и автоматическую реализацию репозиториев.
* Spring Security: обеспечивает комплексную поддержку аутентификации и авторизации, что критично для обеспечения безопасности пользовательских данных и самого приложения.

2. Инверсия Управления и Управление Зависимостями

Одной из основных концепций Spring является инверсия управления (IoC), которая позволяет отделить конфигурацию и внедрение зависимостей от бизнес-логики приложения. Это достигается с помощью контейнера IoC, который управляет созданием и жизненным циклом объектов. Это делает код более модульным, тестируемым и легко изменяемым.

Spring также предоставляет механизмы для управления зависимостями (Dependency Injection), которые упрощают настройку и замену компонентов приложения. Это позволяет легко адаптировать приложение к изменяющимся требованиям и облегчает тестирование компонентов в изоляции.

3. Обработка данных и интеграция с базами данных

Spring Data является одним из самых мощных инструментов для работы с базами данных. Он поддерживает множество баз данных, включая реляционные (например, PostgreSQL, MySQL) и NoSQL (например, MongoDB). Spring Data JPA, в частности, позволяет легко работать с реляционными базами данных, автоматически генерируя SQL-запросы и упрощая реализацию CRUD-операций.

Это позволяет разработчикам сосредоточиться на бизнес-логике приложения, не тратя много времени на написание шаблонного кода для работы с базой данных.

4. Масштабируемость и Производительность

Spring Framework изначально спроектирован с учетом требований к масштабируемости и производительности. Он поддерживает многопоточность и асинхронное выполнение задач, что позволяет эффективно использовать ресурсы сервера и обеспечивать высокую производительность приложения.

Благодаря модульной архитектуре Spring, можно легко добавлять и конфигурировать необходимые компоненты по мере роста и изменения требований к приложению. Это делает Spring отличным выбором для проектов, которые планируются к масштабированию.

5. Безопасность

Безопасность является ключевым аспектом любого веб-приложения. Spring Security предоставляет мощные и гибкие средства для реализации аутентификации и авторизации пользователей. Он поддерживает различные механизмы аутентификации, включая базовую аутентификацию, OAuth2, JWT, и многие другие.

Spring Security также позволяет легко настраивать правила доступа к различным частям приложения, обеспечивая тем самым защиту пользовательских данных и предотвращение несанкционированного доступа.

6. Активная поддержка сообщества и документация

Spring имеет одно из самых активных сообществ среди фреймворков для разработки на Java. Это означает, что разработчики всегда могут рассчитывать на помощь сообщества, множество доступных библиотек и плагинов, а также регулярные обновления и улучшения фреймворка.

Официальная документация Spring является одной из лучших в отрасли, предоставляя исчерпывающие и детальные руководства по всем аспектам использования фреймворка. Это значительно упрощает процесс обучения и освоения Spring для новых разработчиков.

Выбор фреймворка Spring для разработки back-end нашего веб-приложения обусловлен его мощными возможностями, модульностью, поддержкой масштабируемости и безопасности, а также активным сообществом и качественной документацией. Эти факторы делают Spring идеальным выбором для создания надежного, производительного и легко расширяемого приложения для образовательных курсов по программированию.

Уровень данных

Для хранения и управления данными в нашем проекте было выбрано сочетание двух мощных технологий: PostgreSQL для работы с реляционными данными и MinIO для хранения неструктурированных данных (объектов). Такое решение позволяет эффективно управлять разнородной информацией, необходимой для работы образовательного веб-приложения.

1. Открытый исходный код и активное сообщество

PostgreSQL является СУБД с открытым исходным кодом, что означает отсутствие затрат на лицензирование и возможность гибкой настройки под нужды конкретного проекта. Активное сообщество разработчиков постоянно работает над улучшением и развитием PostgreSQL, обеспечивая регулярные обновления, исправления ошибок и внедрение новых функций. Это гарантирует, что СУБД будет поддерживаться и развиваться в долгосрочной перспективе.

2. Надежность и соответствие стандартам

PostgreSQL известна своей надежностью и соответствием стандартам ACID (Atomicity, Consistency, Isolation, Durability), что критично для обеспечения целостности и сохранности данных. Механизмы журналирования (WAL - Write-Ahead Logging) и восстановления после сбоев позволяют минимизировать потерю данных и обеспечить высокую степень готовности системы.

3. Богатый функционал и расширяемость

PostgreSQL предлагает широкий набор встроенных функций, включая поддержку сложных запросов, триггеров, хранимых процедур, индексов различных типов (B-tree, Hash, GiST, GIN и др.), а также полнотекстового поиска. Возможности расширяемости PostgreSQL позволяют добавлять новые типы данных, функции и операторы через плагины и модули, что делает её очень гибкой и настраиваемой.

4. Поддержка различных типов данных и структур данных

PostgreSQL поддерживает множество типов данных, включая примитивные (integer, numeric, string, boolean), структурированные (arrays, hstore, JSON, JSONB), геометрические (PostGIS) и другие. Это позволяет эффективно хранить и обрабатывать разнообразные данные, необходимые для работы нашего образовательного приложения.

5. Масштабируемость и производительность

PostgreSQL предоставляет инструменты и возможности для горизонтального и вертикального масштабирования, что позволяет справляться с увеличением нагрузки и объемов данных. Поддержка шардинга, репликации и балансировки нагрузки делает PostgreSQL подходящей для больших и динамически развивающихся проектов. Индексы, параллельное выполнение запросов и оптимизация планов выполнения запросов способствуют высокой производительности системы.

6. Безопасность

PostgreSQL предлагает развитые механизмы безопасности, включая аутентификацию пользователей, контроль доступа на уровне рядов и столбцов, шифрование данных и SSL/TLS для защиты данных в процессе передачи. Эти функции помогают обеспечить защиту конфиденциальных данных и соответствие стандартам безопасности.

7. Интеграция и совместимость

PostgreSQL легко интегрируется с другими инструментами и технологиями, используемыми в разработке веб-приложений. Она поддерживает стандарты SQL и обладает совместимостью с различными языками программирования (Java, Python, PHP, Node.js и др.), что упрощает разработку и поддержку приложения.

MinIO

Для работы с объектным хранилищем данных в проекте используется MinIO. Это высокопроизводительное решение с открытым исходным кодом, идеально подходящее для хранения больших объемов данных, таких как медиафайлы, документы и резервные копии.

Преимущества использования MinIO:

1.Объектное хранилище S3-совместимое

MinIO поддерживает протоколы Amazon S3, что позволяет легко интегрироваться с существующими решениями и библиотеками для работы с S3-хранилищами.

2.Масштабируемость и высокая производительность

MinIO разработан для работы с огромными объемами данных, обеспечивая низкую задержку и высокую скорость доступа даже при интенсивной нагрузке.

3.Простота развертывания

MinIO легко разворачивается на локальных серверах, в облачных средах или гибридных конфигурациях. Это дает возможность быстро внедрить объектное хранилище в инфраструктуру приложения.

4.Поддержка хранения неструктурированных данных

MinIO идеально подходит для хранения мультимедийных файлов (видео, изображений), пользовательских загрузок, логов и резервных копий, что дополняет возможности PostgreSQL для работы с реляционными данными.

5.Безопасность и защита данных

MinIO поддерживает шифрование на уровне сервера и клиента, а также инструменты контроля доступа и аудита, что гарантирует безопасность данных.

6.Кроссплатформенность и совместимость

MinIO можно использовать с различными языками программирования, включая Java, Python, и другие. Простые API для интеграции делают его универсальным решением.

7.Эффективное резервное копирование и восстановление

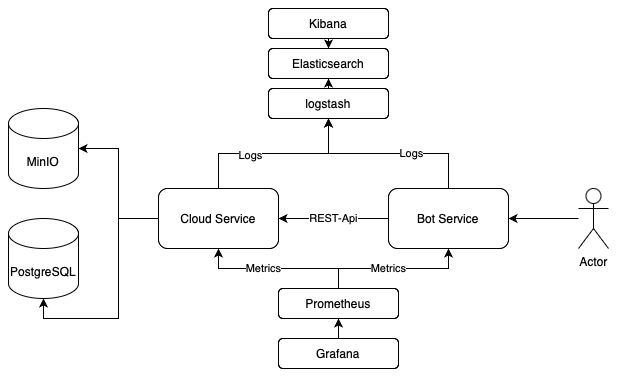
С помощью MinIO можно легко организовать систему резервного копирования, что критично для образовательного проекта, где требуется защита данных от потерь.

Использование PostgreSQL и MinIO в рамках одного проекта позволяет разделить хранение данных на два уровня:

• Реляционные данные (PostgreSQL): для обработки структурированных данных, таких как учетная информация пользователей, статистика, метаданные.

• Объектное хранилище (MinIO): для хранения файлов, документов, медиа и других неструктурированных данных.

На диаграмме компонентов (рис. 1.2) отображено общее взаимодействие компонентов системы:



*Рис. 1.2 Схема окружения*

В данной схеме окружения уровень приложения (Cloud Service) и уровень данных являются сервером, а уровень представления (Bot Service) является клиентом.

## 1.5. Проектирование базы данных

Информация, отображаемая и обрабатываемая программой, хранится в базе данных PostgreSQL. Для проектирования таблиц была использована программа pgAdmin, средство для администрирования PostgreSQL.

Создание различных таблиц в базе данных имеет ключевое значение для обеспечения функциональности и эффективности платформы для хранения файлов. Каждая таблица выполняет свою уникальную роль в организации и хранении данных, необходимых для работы системы.

Таблица Users содержит информацию о зарегистрированных пользователях системы. Она позволяет управлять учетными записями пользователей, их аутентификацией и доступом к системе. В этой таблице хранятся имя пользователя, пароль и данные о хранилище (bucket\_name), используемые для интеграции с объектным хранилищем.

Таблица Workspaces предназначена для хранения рабочих пространств пользователей. Она позволяет создавать отдельные пространства для организации файлов и контента, а также связывать их с владельцами через поле owner\_id. Это обеспечивает персонализацию рабочих областей и распределение прав доступа.

Таблица Files хранит информацию о загруженных файлах в системе. Она позволяет сохранять путь к файлам, их названия, расширения и даты загрузки. Каждая запись в этой таблице связана с конкретным рабочим пространством, что обеспечивает

логическую организацию файлов и их привязку к определенным проектам.

Таблица User\_Workspace фиксирует связи между пользователями и рабочими пространствами, реализуя отношение “многие-ко-многим”. Это позволяет нескольким пользователям работать в одном рабочем пространстве, обеспечивая совместный доступ и управление данными.

Эти таблицы формируют целостную структуру базы данных, обеспечивая эффективное хранение и управление информацией о пользователях, рабочих пространствах и файлах. Такой подход позволяет системе масштабироваться, обеспечивать совместную работу пользователей и поддерживать интеграцию с внешними хранилищами данных.

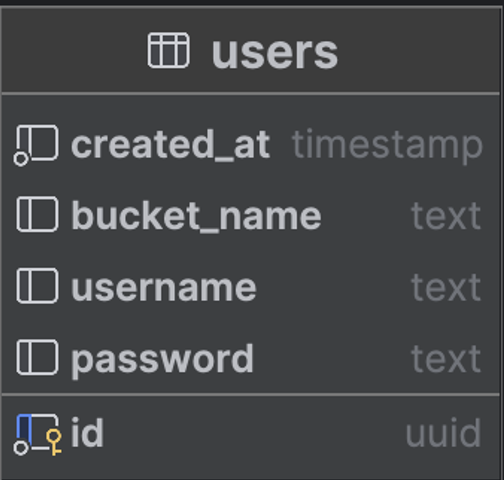
Эти таблицы формируют целостную структуру базы данных, обеспечивая хранение, организацию и управление всей необходимой информацией для эффективной работы платформы.

#### 1.6.1. Структура и описание таблиц

1.6.1.1. Таблица Users

Хранит информацию о пользователях системы.

Структура таблицы Users:



*Рис. 1.4 Структура таблицы Users*

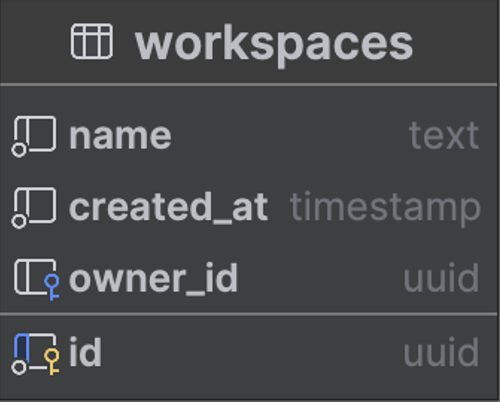
Назначение полей:

* id (uuid): уникальный идентификатор пользователя.
* created\_at (timestamp): дата и время создания записи пользователя.
* bucket\_name (text): имя хранилища для данных пользователя (для интеграции с MinIO).
* username (text): имя пользователя.
* password (text): пароль пользователя.

1.6.1.2. Таблица workspaces

Представляет рабочие пространства, создаваемые пользователями.

Структура таблицы workspaces:



*Рис. 1.5 Структура таблицы workspaces*

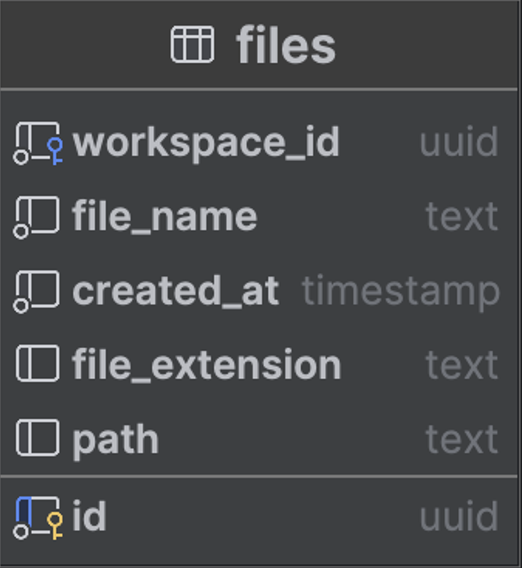
Назначение полей:

* id (uuid): уникальный идентификатор рабочего пространства.
* name (text): название рабочего пространства.
* created\_at (timestamp): дата и время создания рабочего пространства.
* owner\_id (uuid): идентификатор пользователя-владельца (ссылка на таблицу users).

1.6.1.3. Таблица files

Содержит информацию о файлах, связанных с рабочими пространствами.

Структура таблицы files:



*Рис. 1.6 Структура таблицы files*

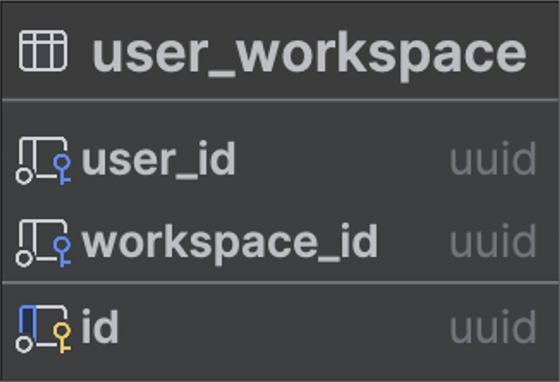
Назначение полей:

* id (uuid): уникальный идентификатор файла.
* workspace\_id (uuid): идентификатор рабочего пространства, к которому принадлежит файл (ссылка на workspaces).
* file\_name (text): имя файла.
* created\_at (timestamp): дата и время создания файла.
* file\_extension (text): расширение файла.
* path (text): путь к файлу.

1.6.1.4. Таблица user\_workspace

Мостовая таблица для связи пользователей и рабочих пространств (многие-ко-многим).

Структура таблицы user\_workspace:



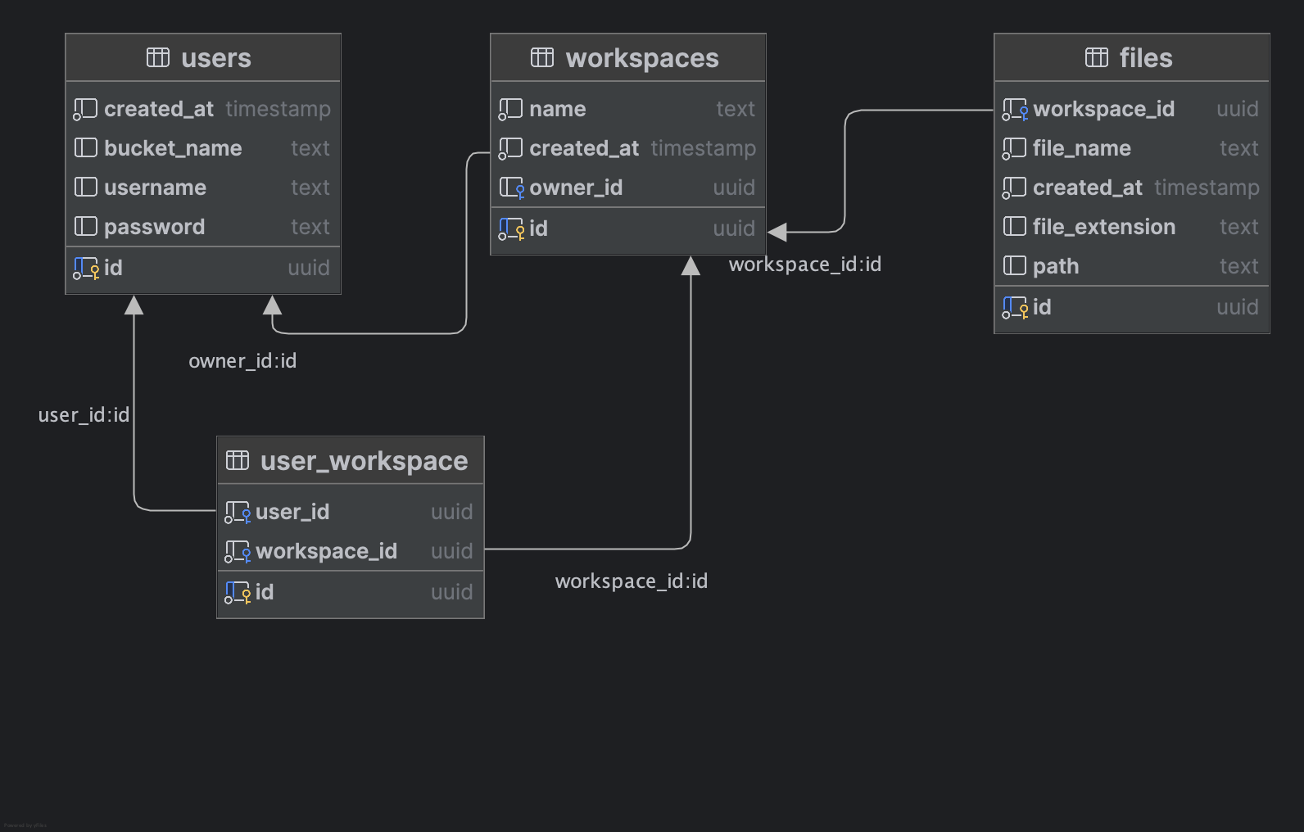
*Рис. 1.7 Структура таблицы user\_workspace*

Назначение полей:

* id (uuid): уникальный идентификатор записи.
* user\_id (uuid): идентификатор пользователя (ссылка на users).
* workspace\_id (uuid): идентификатор рабочего пространства (ссылка на workspaces).

### 1.6.2. ER-диаграмма

Связи между таблицами приведены на ER-диаграмме ниже (рис. x.xx):



*Рис. 1.14 ER-диаграмма*

При проектировании базы данных были реализованы следующие внешние ключи:

* FK\_workspaces\_owner – связь «Один-ко-многим» – для одного пользователя из таблицы users может существовать множество рабочих пространств из таблицы workspaces.
* FK\_files\_workspace – связь «Один-ко-многим» – для одного рабочего пространства из таблицы workspaces может существовать множество файлов из таблицы files.
* FK\_user\_workspace\_user и FK\_user\_workspace\_workspace – связь «Многие-ко-многим» – для одного пользователя из таблицы users и одного рабочего пространства из таблицы workspaces реализована промежуточная таблица user\_workspace, которая позволяет нескольким пользователям работать в одном рабочем пространстве.

# 2. РЕАЛИЗАЦИЯ ПРОГРАММНЫХ МОДУЛЕЙ

## 2.1. Разработка базы данных

Для реализации базы данных нашего веб-приложения было решено использовать PostgreSQL, развернутый с помощью Docker. Это позволило создать контейнеризованную и изолированную среду, обеспечивающую легкость развертывания и масштабируемость базы данных.

Использование Docker для PostgreSQL

Docker обеспечивает удобство управления и развертывания баз данных с помощью контейнеров. В нашем случае был использован официальный Docker image PostgreSQL. Это позволило быстро настроить и запустить базу данных в контейнере с минимальными усилиями.

Использование Hibernate для создания таблиц

Для создания и управления таблицами базы данных мы использовали Hibernate - мощный ORM-фреймворк для Java. Hibernate позволяет работать с базой данных через объектно-ориентированный подход, избегая необходимости написания большого количества SQL-запросов вручную.

Каждая таблица в базе данных была представлена отдельной сущностью (Entity) в коде на Java. Это позволило удобно и структурировано описать структуру таблиц и связи между ними. Ниже представлен пример кода для создания таблицы *users*.

Пример кода сущности User:

@Getter

@Setter

@Accessors(chain = true)

@NoArgsConstructor

@Builder

@AllArgsConstructor

@Entity

@Table(name = "users")

public class User {

@Id

@Column(name = "id", nullable = false, updatable = false, unique = true)

private UUID id;

@Column(name = "created\_at", nullable = false)

private LocalDateTime createdAt;

@Column(name = "bucket\_name", nullable = false)

private String bucketName;

@Column(name = "username", nullable = false, unique = true)

private String username;

@Column(name = "password", nullable = false)

private String password;

@ManyToMany

@JoinTable(

name = "user\_workspace",

joinColumns = @JoinColumn(name = "user\_id"),

inverseJoinColumns = @JoinColumn(name = "workspace\_id")

)

private List<Workspace> workspaces = new ArrayList<>();

@PrePersist

void prePersist() {

var uuid = UUID.randomUUID();

this.id = uuid;

this.bucketName = uuid.toString();

this.createdAt = LocalDateTime.now();

}

}

Описание кода:

* Аннотация @Entity указывает, что данный класс является сущностью и будет отображаться в таблицу базы данных.
* Аннотация @Table(name = "users") указывает, что данная сущность будет отображаться в таблицу с именем *users*.
* Аннотация @Id указывает на первичный ключ таблицы.
* Аннотация @PrePersist указывает, что аннотированный метод должен быть вызван перед тем, как сущность будет сохранена в базу данных. В данном случае она используется для генерации нового UUID и установки createdAt перед сохраняем в базу.
* Связи @ManyToMany используются для определения отношений "один ко многим" между сущностями User и Workspace.

Этот подход позволяет эффективно управлять структурой базы данных и данными, обеспечивая высокую производительность и легкость в обслуживании приложения.

### 2.2.1. Описание конечных точек для HTTP запросов к серверу

В нашем проекте все запросы обрабатываются на сервере. Для обеспечения эффективного и структурированного взаимодействия между клиентом и сервером используются конечные точки (endpoints), которые представляют собой конкретные URL, к которым обращается клиент для выполнения определённых действий. Ниже опишем конечные точки, способ обращения к ним и их функциональность на примере класса .

* POST /api/v1/workspaces - эндпоинт для создания рабочей области в облаке.

request body:

{

"name": "string"

}

response body:

{

"workspaceId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",

"name": "string",

"userId": "3fa85f64-5717-4562-b3fc-2c963f66afa6"

}

* GET /api/v1/workspaces?pageNumber={pageNumber}&pageSize={pageSize} - эндпоинт получения пагинированного контента в рабочей области из облака. Принимает запрос с телом в JSON формате. Ниже пример тела с JSON объектом.

query params:

pageNumber: integer

pageSize: integer

request body:

{

"content": [],

"totalPages": 0,

"totalElements": 0

}

* DELETE /api/v1/workspaces/{workspaceId} - эндпоинт для удаления рабочей области в облаке.

path variable:

workspaceId: integer

request status: 200

* PATCH /api/v1/workspaces/{workspaceId} - эндпоинт для обновления рабочей области в облаке.

request body:

{

"name": "string"

}

response body:

{

"workspaceId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",

"name": "string",

"userId": "3fa85f64-5717-4562-b3fc-2c963f66afa6"

}

* GET /api/v1/workspaces/all - эндпоинт для получения всех рабочих областей в облаке.

response body:

[{

"workspaceId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",

"name": "string",

"userId": "3fa85f64-5717-4562-b3fc-2c963f66afa6"

}]

* GET /api/v1/users - эндпоинт для создания пользователя.

request body:

{

"username": "string",

"password": "string"

}

response body:

{

"id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",

"token": "string"

}

* GET /api/v1/auth - эндпоинт для создания пользователя.

request body:

{

"username": "string",

"password": "string"

}

response body:

{

"token": "string"

}

* POST /api/v1/workspaces/{workspaceId} - эндпоинт для создания рабочей области в облаке.

path variable:

workspaceId: string

response status: 200

* GET /api/v1/files/workspaces/{workspaceId} - эндпоинт для получение файлов из рабочей области в облаке.

query params:

pageNumber: integer

pageSize: integer

path variable:

workspaceId: string

request body:

{

"content": [],

"totalPages": 0,

"totalElements": 0

}

* POST /api/v1/files/workspaces/{workspaceId} - эндпоинт для создания файлов в рабочей области в облаке.

request body: file array

path variable:

workspaceId: string

request status: 200

* POST /api/v1/files/{fileId}/workspaces/{workspaceId} - эндпоинт для создания файлов в рабочей области в облаке.

request body: file array

path variable:

workspaceId: string

fileId: string

request status: string

* DELETE /api/v1/files/{fileId}/workspaces/{workspaceId} - эндпоинт для создания файлов в рабочей области в облаке.

path variable:

workspaceId: string

fileId: string

request status: string

## 2.3. Развертывание API

Для развертывания нашего API и базы данных был использован Docker Compose. Этот инструмент позволяет легко и удобно управлять несколькими контейнерами Docker, обеспечивая возможность быстро поднимать все необходимые сервисы с помощью одного конфигурационного файла. В данном проекте Docker Compose используется для развертывания контейнеров с главным сервисом cb-cloud-service, сервисом для взаимодействия с ботом cb-bot-service, базой данных PostgreSQL, облачным хранилищем MinIO, сервисом для хранения и поиска данных Elasticsearch, веб-интерфейсом для работы с Elasticsearch Kibana, инструментом для обработки и передачи логов в Elasticsearch Logstash, инструментом для мониторинга и алертинга Prometheus и с панелью визуализации данных Grafana.

Конфигурация Docker Compose

Ниже приведен файл docker-compose.yml, который используется для развертывания сервисов:

yaml

version: '3.7'

services:

cb-cloud-service:

image: ${CB\_CLOUD\_SERVICE\_DOCKER\_TAG}

container\_name: cb-cloud-service

depends\_on:

- postgres

- minio

ports:

- "8080:8080"

environment:

- POSTGRES\_HOST=postgres

- POSTGRES\_PORT=5432

- POSTGRES\_DB=${CS\_DB}

- POSTGRES\_USER=${POSTGRES\_USER}

- POSTGRES\_PASSWORD=${POSTGRES\_PASSWORD}

- MINIO\_URL=http://minio:9000

- MINIO\_ACCESS\_KEY=${MINIO\_USER}

- MINIO\_SECRET\_KEY=${MINIO\_PASSWORD}

- TOKEN\_TTL=${TOKEN\_TTL}

- TOKEN\_SECRET=${TOKEN\_SECRET}}

networks:

monitoring:

aliases:

- cb-cloud-service

cb-bot-service:

image: ${CB\_BOT\_SERVICE\_DOCKER\_TAG}

container\_name: cb-bot-service

depends\_on:

- cb-cloud-service

ports:

- "8081:8090"

environment:

- BOT\_TOKEN=${BOT\_TOKEN}

- CB\_URL=http://cb-cloud-service:8080

- TOKEN\_TTL=${TOKEN\_TTL}

networks:

monitoring:

aliases:

- cb-bot-service

prometheus:

image: prom/prometheus:latest

container\_name: prometheus

volumes:

- ./config/prometheus.yml:/etc/prometheus/prometheus.yml

ports:

- "9090:9090"

networks:

monitoring:

aliases:

- prometheus

grafana:

build: './config/grafana'

container\_name: grafana

ports:

- "3000:3000"

environment:

- GF\_SECURITY\_ADMIN\_USER=${GRAFANA\_USER}

- GF\_SECURITY\_ADMIN\_PASSWORD=${GRAFANA\_PASSWORD}

networks:

monitoring:

aliases:

- grafana

logstash:

container\_name: logstash

image: docker.elastic.co/logstash/logstash:8.10.2

volumes:

- logstashdata01:/usr/share/logstash/data

- ./config/logstash.conf:/usr/share/logstash/pipeline/logstash.conf:ro

environment:

- xpack.monitoring.enabled=false

- ELASTIC\_USER=elastic

- ELASTIC\_PASSWORD=${ELASTIC\_PASSWORD}

- ELASTIC\_HOSTS=https://es:9200

ports:

- 5000:5000

networks:

monitoring:

aliases:

- cb-cloud-service

kibana:

container\_name: kibana

image: docker.elastic.co/kibana/kibana:8.10.2

volumes:

- kibanadata:/usr/share/kibana/data

ports:

- 5601:5601

environment:

- SERVERNAME=kibana

- ELASTICSEARCH\_HOSTS=http://es:9200

- ELASTICSEARCH\_USERNAME=kibana\_system

- ELASTICSEARCH\_PASSWORD=${KIBANA\_PASSWORD}

networks:

monitoring:

aliases:

- cb-cloud-service

es:

image: docker.elastic.co/elasticsearch/elasticsearch:8.10.2

container\_name: es

volumes:

- esdata01:/usr/share/elasticsearch/data

ports:

- 9200:9200

environment:

- node.name=es01

- ELASTIC\_PASSWORD=${ELASTIC\_PASSWORD}

- discovery.type=single-node

- xpack.security.enabled=true

networks:

monitoring:

aliases:

- cb-cloud-service

minio:

image: minio/minio:latest

container\_name: minio

ports:

- "9000:9000"

- "9001:9001"

environment:

- MINIO\_ROOT\_USER=${MINIO\_USER}

- MINIO\_ROOT\_PASSWORD=${MINIO\_PASSWORD}

command: server /data --console-address ":9001"

volumes:

- minio-data:/data

restart: unless-stopped

networks:

monitoring:

aliases:

- minio

postgres:

image: postgres:15

container\_name: postgres

environment:

- POSTGRES\_USER=${POSTGRES\_USER}

- POSTGRES\_PASSWORD=${POSTGRES\_PASSWORD}

- POSTGRES\_DB=${CS\_DB}

ports:

- "${POSTGRES\_PORT}:5432"

volumes:

- postgres-data:/var/lib/postgresql/data

restart: unless-stopped

networks:

monitoring:

aliases:

- postgres

networks:

monitoring:

volumes:

esdata01:

driver: local

kibanadata:

driver: local

logstashdata01:

driver: local

minio-data:

driver: local

postgres-data:

driver: local

Описание конфигурации

#### **cb-cloud-service**

* **image: ${CB\_CLOUD\_SERVICE\_DOCKER\_TAG}**Использует образ сервиса API, тег которого задается переменной окружения.
* **container\_name: cb-cloud-service**Указывает имя контейнера для удобства управления.
* **depends\_on:**Указывает, что контейнер зависит от PostgreSQL и MinIO, которые должны быть запущены первыми.
* **ports: "8080:8080"**Прокидывает порт 8080 контейнера на хост-машину, что позволяет взаимодействовать с API.
* **environment**Настраивает переменные окружения для подключения к базе данных PostgreSQL, MinIO и работы с токенами.
* **networks:**Подключается к пользовательской сети monitoring для взаимодействия с другими сервисами.

#### **cb-bot-service**

* **image: ${CB\_BOT\_SERVICE\_DOCKER\_TAG}**Использует образ сервиса бота, тег которого задается переменной окружения.
* **container\_name: cb-bot-service**Указывает имя контейнера.
* **depends\_on:**Гарантирует, что cb-cloud-service будет запущен до запуска бота.
* **ports: "8081:8090"**Прокидывает порт 8090 контейнера на порт 8081 хост-машины.
* **environment**Устанавливает переменные окружения для работы бота и взаимодействия с cb-cloud-service.
* **networks:**Использует сеть monitoring.

#### 

#### **Prometheus**

* **image: prom/prometheus:latest**Официальный образ для мониторинга метрик.
* **container\_name: prometheus**Указывает имя контейнера.
* **volumes:**Связывает конфигурационный файл prometheus.yml с контейнером.
* **ports: "9090:9090"**Делает доступным интерфейс мониторинга Prometheus.
* **networks:**Работает в сети monitoring.

#### **Grafana**

* **build: './config/grafana'**Сборка контейнера Grafana из локальной директории.
* **container\_name: grafana**Имя контейнера для панели визуализации данных.
* **ports: "3000:3000"**Делает интерфейс Grafana доступным на порту 3000 хост-машины.
* **environment:**Настраивает логин и пароль для администратора через переменные окружения.
* **networks:**Использует сеть monitoring.

#### **Logstash**

* **image: docker.elastic.co/logstash/logstash:8.10.2**Использует официальный образ Logstash.
* **container\_name: logstash**Имя контейнера для обработки логов.
* **volumes:**Связывает локальную конфигурацию Logstash с контейнером.
* **ports: "5000:5000"**Прокидывает порт 5000 для приема логов.
* **environment:**Настраивает параметры подключения к Elasticsearch и отключает xpack.monitoring.
* **networks:**Взаимодействует с другими сервисами через сеть monitoring.

#### **Kibana**

* **image: docker.elastic.co/kibana/kibana:8.10.2**Использует официальный образ Kibana.
* **container\_name: kibana**Имя контейнера.
* **volumes:**Обеспечивает постоянное хранение данных Kibana.
* **ports: "5601:5601"**Делает доступным веб-интерфейс Kibana.
* **environment:**Настраивает параметры подключения к Elasticsearch.
* **networks:**Работает в сети monitoring.

#### **Elasticsearch (es)**

* **image: docker.elastic.co/elasticsearch/elasticsearch:8.10.2**Использует официальный образ Elasticsearch.
* **container\_name: es**Имя контейнера.
* **volumes:**Связывает директорию на хосте с контейнером для постоянного хранения данных.
* **ports: "9200:9200"**Делает Elasticsearch доступным на порту 9200 хост-машины.
* **environment:**Настраивает безопасность и параметры узла.
* **networks:**Работает в сети monitoring.

#### **MinIO**

* **image: minio/minio:latest**Использует официальный образ MinIO.
* **container\_name: minio**Имя контейнера.
* **ports: "9000:9000", "9001:9001"**Делает доступным интерфейс MinIO и консоль управления.
* **command:**Настраивает запуск сервера с адресом консоли :9001.
* **volumes:**Хранение данных осуществляется через volume minio-data.
* **networks:**Включен в сеть monitoring.

#### **PostgreSQL**

* **image: postgres:15**Использует официальный образ PostgreSQL.
* **container\_name: postgres**Имя контейнера для базы данных.
* **ports: "${POSTGRES\_PORT}:5432"**Делает PostgreSQL доступным на указанном порту хост-машины.
* **volumes:**Обеспечивает сохранность данных базы через volume postgres-data.
* **environment:**Настраивает пользователя, пароль и имя базы данных через переменные окружения.
* **networks:**Работает в сети monitoring.

#### 

#### **networks:**

* Определена сеть monitoring для взаимодействия всех сервисов.

#### **volumes:**

* Настроены локальные тома (esdata01, kibanadata, logstashdata01, minio-data, postgres-data) для постоянного хранения данных.

Запуск сервисов

Для запуска всех сервисов используется команда:

docker-compose up -d

* up: Команда запускает контейнеры, определенные в docker-compose.yml.
* -d: Флаг запускает контейнеры в фоновом режиме (detached mode).

После выполнения этой команды Docker Compose автоматически создаст и запустит все контейнеры, используя указанные настройки.

Преимущества использования Docker Compose

1. Упрощение развертывания: Docker Compose позволяет автоматически развернуть и настроить несколько контейнеров с одной команды. Это значительно упрощает процесс развертывания по сравнению с ручной настройкой каждого контейнера.
2. Изолированная среда: Каждый контейнер работает в своей изолированной среде, что минимизирует конфликты зависимостей и версий программного обеспечения.
3. Гибкость и масштабируемость: Легкость добавления новых сервисов и масштабирование существующих контейнеров обеспечивает гибкость при изменении или расширении проекта.
4. Консистентность среды: Docker Compose гарантирует, что все окружения (разработки, тестирования, продакшн) будут иметь идентичные настройки, что минимизирует проблемы, связанные с различиями в конфигурации.

Использование Docker Compose для развертывания компонентов проекта предоставляет множество преимуществ, включая простоту, гибкость и консистентность. Это решение позволяет сосредоточиться на разработке функционала приложения, а не на управлении инфраструктурой, обеспечивая надежную и воспроизводимую среду для развертывания нашего приложения.

# 

# ЗАКЛЮЧЕНИЕ

Таким образом, в результате работы над платформой для тестирования знаний были разработаны:

1. База данных на PostgreSQL;
2. Веб-сервер с помощью технологии Spring;
3. Интерфейс на основе Telegram API.

Разработка бота для файлового хранилища является важным шагом в сфере управления данными и повышения удобства их использования. Применение современных технологий позволяет создать надежный инструмент, обеспечивающий хранение, управление и доступ к файлам прямо через мессенджер.

Интеграция PostgreSQL в качестве реляционной базы данных обеспечивает безопасное и структурированное хранение информации о пользователях, загруженных файлах, их метаданных и настройках доступа. Это позволяет эффективно управлять данными и обеспечивать высокий уровень их защиты.

Применение Spring Framework гарантирует надежность, масштабируемость и удобство разработки серверной части бота. С помощью Spring можно создавать надежные API, которые обеспечивают обработку запросов от Telegram API, быструю работу с базой данных и стабильную производительность.

Использование Telegram API для разработки интерфейса бота делает взаимодействие пользователя с системой интуитивно понятным и удобным. Это позволяет легко загружать, сохранять и получать доступ к файлам через знакомый интерфейс Telegram.

В целом, бот для файлового хранилища, созданный с использованием PostgreSQL, Spring и Telegram API, представляет собой современное и удобное решение для управления файлами. Такой инструмент способствует повышению эффективности работы с данными, предоставляет простой и безопасный доступ к ним и отвечает требованиям современных пользователей.

# Список источников литературы

1. Основы тестирования и верификации программного обеспечения. Старолетов С. М. – Санкт-Петербург: Лань, 2020. – 344 с.
2. Проектирование информационных систем. Вейцман В. – Санкт-Петербург: Лань, 2019. – 316 с.
3. Проектирование информационных систем: метод. указания по выполнению лабораторных работ. Коваленко В. В. – Сочи: РИЦ ФГБОУ ВО «СГУ», 2020. – 40 с.
4. Разработка требований к программному обеспечению. 3-е изд., дополненное / Пер. с англ. Вигерс Карл, Битти Джой — М.: Издательство «Русская редакция»; СПб.: БХВ-Петербург, 2014. — 736 с.
5. Норенков И. П. Основы автоматизированного проектирования: Учеб. для вузов. 2-е изд., перераб. и доп. — М.: Изд-во МГТУ им. Н. Э. Баумана, 2018 — 336 с.;
6. Г.Н. Федорова «Разработка, внедрение и адаптация программного обеспечения отраслевой направленности»: Учебное пособие / Г.Н. Федорова. – Москва: КУРС: ИНФРА – М, 2021. – 336 c.;