# СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

*СУБД* – система управления базами данных.

*БД* – база данных.

*ИС* – информационная система.

*DSL* – Domain Specific Language.

*SQL* – Structured Query Language.

*BPMN* – Business Process Model and Notation.

*URI* – Uniform Resource Identifier.

*API* – Application Programming Interface.

# ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

*Информационная система* – система, предназначенная для хранения, поиска и обработки информации, и соответствующие организационные ресурсы (человеческие, технические, финансовые и т. д.), которые обеспечивают и распространяют информацию (ISO/IEC 2382:2015[[1]](https://www.iso.org/obp/ui/#iso:std:iso-iec:2382:ed-1:v1:en).

*База* *данных* — совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств [моделирования данных](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) (ISO/IEC 10032:2003[[2]](https://www.iso.org/standard/38607.html).

*Индекс* – объект базы данных, создаваемый с целью повышения производительности поиска данных.

*Фреймворк* – набор функций и объектов, определяющий структуру программной системы, «каркас».

*Domain Specific Language* – предметно-ориентированный язык программирования, для использования в конкретной области применения.

*Structured Query Language* – язык структурированных запросов, для управления реляционной моделью данных в СУБД.

*Business Process Model and Notation* – система условных обозначений для моделирования бизнес-процесс. Также может применятся для описания взаимодействия компонентов в информационных системах[[3]](https://habr.com/ru/company/trinion/blog/331254/).

*Uniform Resource Identifier* – универсальный идентификатор ресурса, строка, позволяющая идентифицировать ресурс в сети интернет.

*Application Programming Interface* – спецификация взаимодействия с программой или программным компонентом.

# ВВЕДЕНИЕ

Всю свою историю, человечество изобретало новые способы хранения информации: наскальные рисунки, записки на папирусе, печатные книги, компьютеры… Чем сильнее развивался человек, тем больше информации ему необходимо было сохранить, получить и обработать, что повышало требования и к эффективности хранения, и к скорости доступа к ней (информации). Массовый приход вычислительной техники в нашу жизнь в конце двадцатого века в очередной раз повысил эти требования. Тогда и были изобретены базы данных и системы управления базами данных.

Современные информационные системы невозможно представить без баз данных, они используются повсеместно: интернет-магазины, встраиваемые устройства, приложения и игры, банки, корпоративные системы.

Когда некий клиент обращается к информационной системе, система отправляет запрос к базе данных, а база данных, в свою очередь, должна выполнить этот запрос и вернуть результат – от этой операции сильно зависит общее время, затраченное клиентом на действие в информационной системе.

С развитием технологий, растёт и активное число пользователей различных информационных систем, растёт количество запросов, объём передаваемых данных, сложность данных, что делает задачу ускорения обработки запросов к базам данных **актуальной** на любом этапе развития технологий.

**Целью** работы является автоматизация процесса ускорения база данных, что освободит администраторов и программистов от решения большей части задач по оптимизации, даст им дополнительное время на ускорении наиболее критичных и важных запросов вручную. Так как базы данных могут быть развёрнуты на всех актуальных на данный момент операционных системах (Windows, macOS, Linux) и архитектурах (x86-64, arm), то процесс автоматизации должен быть доступен на всех перечисленных конфигурациях.

Для реализации цели, передо мною были поставлены следующие **задачи**:

1. Обзор существующих решений.
2. Разработка требований, предъявляемых к создаваемому приложению.
3. Выбор средств реализации.
4. Проектирование и создание тестовой базы данных.
5. Проектирование и создание приложения.
6. Тестирование приложения.
7. Анализ полученных результатов.

**Результатом** работы является приложение, которое принимает на вход истории запросов к базе данных; производит действия, направленные на ускорение этих запросов; возвращает отчёт с результатами своей работы.

Работа поделена на четыре части: в первой содержится обзор существующих решений и обоснование используемых технологий. Во второй, описание этапа проектирования тестовой базы данных и приложения. В третьей части описание этапа разработки, а в последней, четвертой, обзор получившегося продукта и анализ результатов его работы.

# ПОДГОТОВИТЕЛЬНЫЙ ЭТАП

## Введение в предметную область

В широком смысле, базой данных можно назвать любое существующее средство хранения информации, будь то книга или молекула ДНК. В контексте компьютерных наук, первые базы данных появились в 50-х годах прошлого столетия, когда появилось программируемое оборудование для записи.

В 70-х, когда данных стало много, остро стала задача манипулирования этими данными, был разработан SQL (Structured Query Language, язык структурированных запросов) – язык программирования для управления, изменения, создания и удаления данных, основанный на реляционной модели данных. Реляционная модель данных – прикладная теория, которая определяет задачи обработки данных на трёх аспектах:

1. Структурный аспект – данные представляют собой набор отношений (часто их называют таблицами).
2. Аспект целостности – отношения отвечают условия целостности, которые уникальны для каждой модели данных.
3. Аспект обработки – модель должна поддерживать операторы манипулирования отношениями, которые называются реляционной алгеброй.

Язык SQL, хотя и не был Тьюринг-полным языком (не позволял реализовать любую вычислимую функцию), но оказался чрезвычайно удобен для задач манипулирования данными. В течение десятилетия, появилось несколько СУБД, которые использовали SQL, к примеру: IBM System R, Ingres (будущая PostgreSQL), Oracle V2 (будущая Oracle Database). Популярность языка привела к его стандартизации комитетом ANSI в 1986 году, и к появлению класса SQL-совместимых СУБД, которые следовали стандарту ANSI-SQL, а также расширяли его, с целью получения конкурентных преимуществ. Это способствовало дальнейшему развитию языка: ANSI стандартизировало лучшие практики из той или иной СУБД, которые не были защищены патентным правом. Также, свою роль в массовости SQL сыграл декларативный характер языка (пользователями необходимо было задавать спецификацию решения задачи, а не способ её получения) и развитие средств визуального построения запросов.

Параллельно с этим, развивался класс NoSQL СУБД, которые предлагали альтернативный язык для работы с реляционной моделью данных. NoSQL базы не были хуже, но не предлагали такой легкости и гибкости как SQL, поэтому закрепились в определённых сферах, где использование того или иного языка или СУБД давало значительные преимущества.

Так как класс SQL-совместимых баз данных самый крупный и популярный на данный момент[[4]](https://db-engines.com/en/ranking), а многие вещи стандартизированы, то задачу автоматизации стоит решать именно для них.

## Обзор существующих решений

Перед разработкой стоит рассмотреть существующие решения. 7 из 10 самых популярных баз данных используют SQL как язык запросов. Рассмотрим их на наличие необходимой функциональности, данные будут представлены в таблице.

Таблица 1 - Сравнение аналогов

|  |  |
| --- | --- |
| **СУБД** | **Функциональность** |
| Oracle | Умеет автоматически строить индексы на основе истории запроса, но:   1. СУБД платная, цена даже за самую простую редакцию составляет несколько тысяч долларов[[5]](https://www.oracle.com/assets/technology-price-list-070617.pdf). 2. Алгоритм и правила построения индексов представляют из себя «черный ящик» для пользователя. |
| MySQL | Автоматически строит индексы только для полей, представляющих из себя ключи, и только во время создания таблицы или внесения в неё нового ключа. |
| Microsoft SQL Server | Умеет автоматически строить индексы на основе истории запросов, но:   1. Данная функциональность доступна только обладателями платной подписки cлужбы облачных вычислений Microsoft Azure. 2. Алгоритм и правила построения индексов представляют из себя «черный ящик» для пользователя. |
| PostgreSQL | Автоматически строит индексы только для полей, представляющих из себя ключи, и только во время создания таблицы или внесения в неё нового ключа. |
| IBM Db2 | Автоматически строит индексы только для полей, представляющих из себя ключи, и только во время создания таблицы или внесения в неё нового ключа. |
| Microsoft Access | Автоматически строит индексы только для полей, представляющих из себя ключи, и только во время создания таблицы или внесения в неё нового ключа. |
| SQLite | Автоматически строит индексы только для полей, представляющих из себя ключи, и только во время создания таблицы или внесения в неё нового ключа. |

Из таблицы можно сделать выводы о том, что существует потребность в создании приложения, которые позволит реализовать функциональность автоиндексирования для каждой из СУБД, с понятным и открытым принципом работы, а также с более доступной или вовсе бесплатной моделью использования.

## Обоснование используемых технологий

Ключевым требованием к разрабатываемому приложению является поддержка как можно большего числа конфигураций «операционная система – архитектура компьютера – СУБД», то в качестве языка программирования следует выбрать тот, код которого будет ожидаемо и единообразно исполняться везде, где может запускаться. Таким языком является Java, благодаря Java Virtual Machine (JVM) *–* среде выполнения, доступной на трёх миллиардах устройств по заверениям разработчиков[[6]](https://www.oracle.com/java/moved-by-java/timeline/). Созданный в 1990 году Java – крайне распространённый язык, ежегодно занимающий лидирующие позиции[[7]](https://www.tiobe.com/tiobe-index/) в рейтингах популярность и любви от программистов за удачную реализацию принципов объектно-ориентированного программирования (ООП), наличия удобных средств разработки, пакетных менеджеров и многих других преимуществ. Тем не менее, за свою более чем тридцатилетнюю историю, как и любой другой язык, Javа успел накопить проблем, в первую очередь связанных с слишком строгом следовании правилам ООП и громоздким синтаксисом. Со временем, стали появляется другие языки, которые исправляли накопившиеся проблемы, а их код мог выполняться в JVM, что обеспечивало полную совместимость с существующим кодом, написанным на Java.

Одним из таких языков является Kotlin, созданный в Санкт-Петербурге в 2011 году фирмой JetBrains, специализирующейся на разработке средств программирования, преимущественно на Java. Имея богатый опыт, контакт с сообществом, под вдохновением от книги «Java. Эффективное программирование», они создали язык, который поддерживал мультипарадигменное программирование и значительно упростил, и укоротил оригинальный синтаксис. Многие из тех вещей, которые были введены в Kotlin, в данный момент заимствуются в новых редакциях Java. Именно Kotlin будет использоваться при разработке приложения.

Другой причиной, почему была выбрана именно технология JVM, помимо языка, является существование протокола JDBC. Протокол обеспечивает общий интерфейс для взаимодействия с любой SQL СУБД, для который разработчики СУБД написали JDBC-драйвер, что они обычно и делают, ввиду широкого распространения Java/JVM в корпоративной разработке. Таким образом, программистам, решившим реализовать в своём приложении работу с новый СУБД, не нужно изучать библиотеку от разработчиков этой самой СУБД, нужно лишь один раз научится работать с протоколом JDBC. Это также позволит мне удобно добавлять поддержку новых СУБД, так как интерфейс выполнения SQL выражений будет общий, сложная задача будет состоять в том, чтобы научить программу формировать выражения создания индексов, специфичные для каждый СУБД.

Приложению также нужен интерфейс взаимодействия с пользователем, графический или консольный. Стоит отметить, что, если реализовать взаимодействие приложения через HTTP-протокол, появится возможность независимо от ядра приложения (той части, которая будет отвечать за разбор истории запросов и формирование индексов), обновлять графический интерфейс. Или создать консольный интерфейс ввода. Или создать другой клиент на любой вкус. Было решено так и поступить. Клиент я решил реализовать в виде веб-приложения, как это часто делают другие авторы ПО, используемого в качестве инфраструктурного, к примеру, Jenkins, WildFly или pgAdmin. Выбор фронтенд-фреймворка по большей части дело вкуса: каждый из трёх самых популярный фрейворков (Vue, React, Angular) имеет подробную документацию, активное сообщество, библиотеки компонентов. С первыми двумя у меня был опыт, в обоих случаях положительный, а потому я решил использовать React: мне он нравится своим декларативным подходом.

Выбор библиотеки компонентов зависит от внешнего вида, так как правила работы с компонентами определяет фреймворк, а потому они везде будут одинаковы. Я выбрал Ant Desing, стиль компонентов мне показалась наиболее аккуратным, а документация очень удобной, ввиду наличия множества примеров использования компонентов, практически на каждый параметр присутствует свой пример.

Так как ядро приложения будет является сервером, доступ к которому могут получить несколько клиентов в сети, или может быть развернуто несколько экземпляров приложения параллельно, нам нужно где-то хранить информацию, какая из баз данных сейчас индексируется приложением, чтобы не допустить параллельного построения одних и тех же индексов несколькими экземплярами. Хранить достаточно лишь строку, url занятой базы данных. Для таких целей отлично подойдут хранилища типа «ключ-значение», самые известные из которых Redis и Memcached. Оба продукта представляют одинаковую функциональность, но Redis имеет чуть более удобную в использовании библиотеку для JVM, поэтому был выбран он.

Kotlin поддерживает те же серверные фреймворки, что и Java, в том числе Spring Boot, давно зарекомендовавший себя как лучший фреймворк для серверной разработки на JVM. Тем не менее, он создавался для Java, и его использование в паре с Kotlin приведёт с некоторым ограничением в отношении использования разных возможностей языка. Для Kolin существует свой фреймворк – Ktor. Он написал с учётом всех плюсов языка, декларативен, основан на замыканиях, а не классах, по сути, представляет из себя DSL, что очень удобно.

Невозможно будет проверить работу программы без существования сложной базы данных. Модель данных будет описана в следующей части, здесь стоит определиться с СУБД, в который эта база будет создана. Для тестирования, стоит выбрать ту, которая имеет несколько разных индексов, позволит создать сложную модель данных, а опыт работы в ней достаточен, чтобы в случае проблем быть уверенным, что проблемы именно в приложении, а не в неправильно сконфигурированной СУБД. Для меня такой является PostgreSQL.

# 2. ЭТАП ПРОЕКТИРОВАНИЯ

## 2.1 Проектирование структуры и процессов

Определившись с используемыми технологиями, нужно понять, когда и как эти технологии будут между собой взаимодействовать. При проектировании стоит учитывать, что процесс автоиндексирования базы данных может занимать длительное время, поэтому передача клиенту итогового отчёта будет асинхронным процессом. В таком случае, наилучшим вариантов будет дать возможность клиенту самому выбрать, куда получить отчёт, а не дожидаться HTTP-ответа от серверной части приложения. В связи с этим, я вижу этот процесс так:

1. Клиент создаёт задание на автоиндексацию, предоставляя историю запросов, данные для подключения к БД и данные для получения отчёта.
2. Сервер, получив запрос, выполнить необходимые проверки, среди которых, к примеру, проверка на доступность БД, над который будет проводиться работа и разбор SQL-выражений на корректность, не занята ли БД индексацией в данный момент.
3. Клиент получит сообщение, в случае успеха: его задание принято в работу, и отчёт он получит, как только работа будет завершена; в случае проблем на этапе валидации, сообщение об ошибке придёт сразу в теле HTTP-ответа; в случае проблем на этапе автоиндексации, сообщение об ошибке придёт туда, куда он указал получать отчёты.

Наглядно этот процесс представлен на BPMN-схеме (Рисунок 1). Рисунок 2 показывает взаимодействие всех компонентов системы.

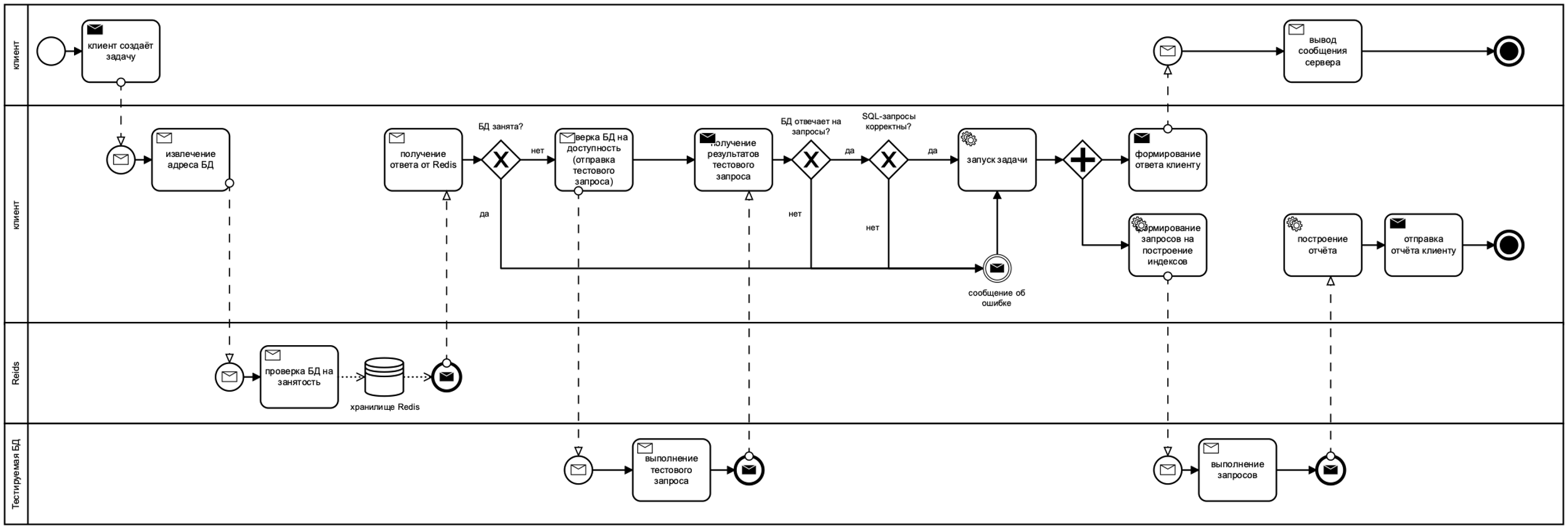


Рисунок 1 - BPMN-схема

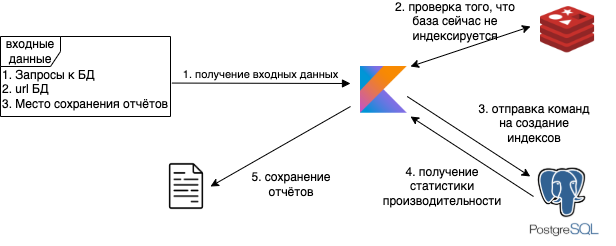


Рисунок 2 - Компонентная схема

## Проектирование API

Так как количество поддерживаемых приложение СУБД можно увеличить, путём подключения нового JDBC-драйвера, необходимо также отдавать клиентским приложениям информацию о том, какие СУБД поддерживаются на данный момент, чтобы они, в свою очередь, не пытались слать бессмысленные задачи на оптимизацию. Также, динамическое получение клиентами списка доступных СУБД позволит оставлять их код неизменным при добавлении нового драйвера к серверу. В качестве поддерживаемой изначально я выбрал PostgreSQL, причины были описаны на этапе 1.3. Таким образом будет сформирован первый путь к серверу: support/instances/.

Отчёты, формируемые по результатам автоиндексации, должны содержать информацию о том, какой индекс использовался при осуществлении запроса к БД и как изменилась скорость выполнения по сравнению с оригинальным запросом, выполненным без индекса. Изначально, в качестве формата отчёта я выбрал CSV: этот формат, ввиду небольшого количества служебных символов (запятые и символы переноса строк), легко читаемый человеком. Его просмотр доступен в средствах форматирования электронных таблиц, таких как Microsoft Excel. Тем не менее, помимо человека, отчёт может понадобится другой клиентской программе, поэтому также необходимо предоставлять его в форме, которую также хорошо и быстро воспринимает вычислительная машина. Самым популярным таким форматов является JSON, чуть ли не повсеместно применяемый при передаче данных по протоколу HTTP. На выбор пользователю будут доступны оба формата. Если же, как и в случае с СУБД, отдавать список поддерживаемых форматов с сервера, то новые форматы можно будет добавлять, не внося изменений в клиент. Создадим путь support/formats/.

Было оговорено, что отчёты будут возвращаться пользователю асинхронно. Сущность, куда будет отправлен отчёт назовём «потребителем». Со списком потребителей можно организовать ту же схему, что и с список поддерживаемых СУБД и форматов – возвращать его клиенту, а клиент будет выбирать потребителя из этого списка. Нужно определится с потребителями, которые буду доступны в приложении с самого начала. Предполагается, что разворачивать программы пользователь будет в рамках своей локальной или корпоративной сети: никто не захочет передавать логин и пароль от своей базы в третьи руки. Приоритетным потребителем в таком случае будет выступать файловая система, пользователю в таком случае надо будет указать директорию, куда сохранять отчёт. Тем не менее, возможны ситуации, когда нельзя или не надо будет сохранять отчёт на машине, где развёрнуто приложение, надо обеспечить альтернативу. В локальных сетях часто используют смонтированный для общего доступа диск, доступ к которому возможен по разным протоколам. Одним из таких протоколов является SFTP: на машинах, доступ к которым есть по SSH, есть и доступ по SFTP. Альтернативой ему, часто используемой на устройствах с ОС Windows является SMB. Ещё может оказаться полезной отправка отчётов тем, у кого по тем или иным причинам нет доступа и к сети с машиной с развёрнутыми приложением вообще. В качестве такого потребителя будет выступать электронная почта. Итого, по умолчанию доступны четыре потребителя: FS (файловая система), SFTP, SMB, EMAIL. Актуальный список поддерживаемых потребителей будет доступен на support/consumers/.

Теперь надо определится с форматом запросов для создания задачи на автоиндексирование БД. В первую очередь, нужно передать информацию для подключения: протокол, адрес, порт, имя логической базы данных, логин и пароль. Всю эту информацию можно объединить в URI. Историю SQL-запросов можно передавать в виде массива строк. В виде можно передавать и требуемый формат отчёта, и наименование потребителя. Наиболее сложной задачей является каким-то образов стандартизировать информацию, необходимую для взаимодействия с потребителем. Серверу нужна разная информация для работы с потребителем: если потребитель является электронным почтовым ящиком, то необходим только адрес этого ящика. А если, к примеру, это SFTP хранилище, то нужны уже и адрес, и данные для авторизации. От потребителя к потребителю структура может становится как проще или сложнее. Решено было компоновать эту информацию в одну строку и разделять информацию символом «;». Информация, необходимую каждому поставщику будет описана в документации. В большинстве случаев, это всё же обычный URI. Запрос на новую задачу по автоиндексации будет располагаться по адресу: /bench/.

Полная спецификация API, оформленная в соответствии с стандартом OpenAPI 3[[8]](https://swagger.io/specification/) ниже (Рисунок 3).



Рисунок 3 - API серверной части

## Проектирование тестовой базы данных

# СПИСОК ИСТОЧНИКОВ

1. Международный стандарт ISO/IEC 2382:2015 Information technology – Vocabulary [Электронный ресурс] – URL: <https://www.iso.org/obp/ui/#iso:std:iso-iec:2382:ed-1:v1:en>. – Режим доступа: свободный. – Дата обращения: 02.05.2022.
2. Международный стандарт ISO/IEC TR 10032:2003 Reference Model of Data Managment [Электронный ресурс] – URL: <https://www.iso.org/standard/38607.html>. – Режим доступа: свободный. – Дата обращения: 02.05.2022.
3. Статистика использования СУБД [Электронный ресурс] – URL: <https://db-engines.com/en/ranking>. – Режим доступа: свободный. – Дата обращения: 01.05.2022.
4. Цена на редакции СУБД Oracle Database [Электронный ресурс] – URL: <https://www.oracle.com/assets/technology-price-list-070617.pdf>. – Режим доступа: свободный. – Дата обращения: 01.05.2022.
5. История Java от Oracle [Электронный ресурс] – URL: <https://www.oracle.com/java/moved-by-java/timeline/>. – Режим доступа: свободный. – Дата обращения: 03.05.2022.
6. Статистика использования языков программирования [Электронный ресурс] – URL: <https://www.tiobe.com/tiobe-index/>. – Режим доступа: свободный. – Дата обращения: 03.05.2022.
7. Описание компонентов BPMN на русском языке [Электронный ресурс] – URL: <https://habr.com/ru/company/trinion/blog/331254/>. – Режим доступа: свободный. – Дата обращения: 05.05.2022.
8. Правила спецификации OpenAPI 3 [Электронный ресурс] – URL: <https://swagger.io/specification/>. – Режим доступа: свободный. – Дата обращения: 06.05.2022.