

Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 23.М07-мм

Разработка системы формирования XBRL-отчетностей

Киреев Андрей Андреевич

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:

доцент кафедры СП Луцив Д.В.

Консультант:

руководитель проекта Хайзников Ф.М.

Место работы: ООО «ДИДЖИТАЛ СПИРИТ»

Санкт-Петербург

2024

Оглавление

Введение	3
1. Постановка задачи.....	5
2. Обзор.....	6
2.1. Условия	6
2.2. Аналоги	7
2.3. Технологии.....	7
3. Реализация	10
3.1. Архитектура	10
3.2. Модели данных.....	11
3.3. Реализация шедулера.....	14
3.4. Реализация системы приоритета.....	15
3.5. «Самооркестрация» инстансов системы.....	15
3.6. API-методы.....	17
3.7. Юнит-тесты.....	18
4. Расчеты компонентов.....	18
4.1. Расчет промежуточной базы данных ПМ.....	19
4.2. Расчет целевой базы данных ВМ и S3-хранилища.	20
Заключение	22

Введение

XBRL — это расширяемый язык деловой отчетности (от англ. extensible Business Reporting Language), формат передачи регуляторной, финансовой и иной отчетности [1]. XBRL-отчет — это файл, передаваемый в Центральный банк Российской Федерации (он же Банк России, Центробанк, ЦБ РФ) отчитывающейся финансовой организацией, в котором указывается деятельность этой организации в формате установленным правилами и требованиями ЦБ РФ [2]. Файл обычно имеет расширение “.xbrl” или “.xml”.

Данная работа написана в ходе производственной практики в компании, занимающейся созданием решения автоматизации процесса формирования отчетов для некоторой банковской организации. Этим обусловлены ограничения и меры принятые в ходе реализации системы.

Одно из таких ограничений это список отчетов, автоматическое формирование которых и было согласовано к разработке. А именно:

1. Форма 0420754 «Сведения об источниках формирования кредитных историй» [3].

- Раздел II. Сведения о записях кредитных историй и (или) иных данных, передаваемых источниками формирования кредитных историй.

- Раздел III. Сведения об источниках формирования кредитных историй, которым были уступлены права требования, не предоставляющих сведения в бюро кредитных историй.

- Раздел IV. Сведения о передаче источниками формирования кредитных историй недостоверных сведений в бюро кредитных историй.

2. Форма 0420755 «Сведения о пользователях кредитных историй» [4].

- Раздел II «Сведения о количестве запросов, полученных бюро кредитных историй».

- Раздел III. Сведения об отказах бюро кредитных историй в исполнении запросов пользователей кредитных историй, лиц, запросивших кредитный отчет.

- Раздел V. Сведения о запросах пользователей кредитных историй на получение кредитных отчетов субъектов кредитных историй.

3. Форма 0420762 «Реестр контрагентов» [5].

Создание XBRL-отчетов на текущий момент основано на ручных манипуляциях сотрудника конкретной организации. Данные для формирования находятся в разных типах источников: База Данных PostgreSQL, База Данных Oracle, S3-хранилище - соответственно человек, занимающийся формированием отчетов, вынужден собрать данные из всех этих источников за выбранный период даты, правильно агрегировать и, если это необходимо, провести вычисления над данными и после создать XBRL-отчет. Организацией-заказчиком были предприняты меры по ускорению этого процесса путем написания различных SQL- и Python-скриптов, но в данном случае имеется необходимость полного присутствия обученного сотрудника на протяжении всего формирования отчета. Таким образом, этот негативный фактор послужил сигналом к необходимости создать более автоматическое решение.

1. Постановка задачи

Целью работы является создание автоматической системы формирования XBRL-отчетов. Для её выполнения были поставлены следующие задачи:

1. Продумать и спроектировать архитектуру системы.
2. Выбрать и подобрать технологии и компоненты системы по необходимости. Для баз данных системы рассчитать оптимальный размер памяти.
3. Реализовать API-методы взаимодействия с системой.
4. Написать юнит-тесты, для тестирования фрагментов приложения.

2. Обзор

2.1. Условия

Для корректной реализации проекта необходимо учитывать все поставленные требования и условия от организации-заказчика. Стоит рассмотреть их подробнее:

- У организации есть потребность в ручной, удобной проверке отчётностей, которую должен выполнять компетентный сотрудник. Соответственно комфортный формат для сверки отчётностей для сотрудника — это любой, с которым можно взаимодействовать средствами Microsoft Excel. Также у компании есть скрипт, который переводит все отправляемые в ЦБ РФ отчёты автоматически из разрешения «.csv» в формат «.xml». Соответственно, на основании этих двух факторов, выходные данные, которые ожидается получить 7 XBRL-отчётов в формате CSV.
- Входными данными являются: JSON-файлы разного формата заполнения в 2-ух разных бакетах S3-хранилища, 4 таблицы в базе данных Oracle, 1 представление в базе данных PostgreSQL.
- Решение должно предоставлять отчет за каждый квартал года по его истечению.
- Должны быть настроены средствами Prometheus и Grafana метрики профилирования приложения, бизнес-метрики, алерты ошибок [6].
- По истечению 5 лет данные можно считать не актуальными.
- Приложение поднимается и работает на подах оркестратора Kubernetes.
- Приложение должно собираться под Alpine версией ОС Linux [7].
- Система должна учитывать георезервирование: запуск одного и того же приложения в разных ЦОД.

2.2. Аналоги

Данная создаваемая система является узкоспециализированной под конкретные задачи организации, поэтому даже при возможном уже реализованном схожем решении — детальная информация о нем будет закрытой.

Также стоит отметить, что организацией были написаны небольшие скрипты позволяющие формировать отчёты, но скорость формирования отчетов и необходимость наличия компетентного сотрудника для их подготовки — делает такое решение низкоэффективным и не конкурентным. Соответственно можно считать, что аналогов создаваемой системе нет.

2.3. Технологии

2.3.1. Golang

Из-за необходимости держать запущенное приложение удалённо, взаимодействовать с базами данных и хранилищем данных, а так же предоставлять возможность взаимодействия с системой, то данное решение было принято оформить как Backend-проект. Одним из языков поддерживающих Backend-разработку «из коробки» — является язык программирования Golang (кратко Go).

Достоинства языка можно выделить следующие [8]:

- Все необходимое для разработки серверной части приложения есть в стандартном пакете Go;
- Go имеет достаточно простой синтаксис, схожий с С-языками;
- Имеются эффективные инструменты для работы с многопоточными приложениями. В Go массово используются «горутины» — легковесные потоки, создаваемые и контролируемые Go-приложением.

- Из-за простоты и легковесности языка, возможности поддерживать C-библиотеки напрямую — приложения на Go работают быстрее многих конкурентов.

2.3.2. “go-pg@v10” и “goracle”

По условию работы — необходимо активное взаимодействие с двумя типами баз данных - PostgreSQL и Oracle. Соответственно под них были подобраны две библиотеки:

“go-pg@v10” — это последняя версия самой распространённой ORM-библиотеки, которая предоставляет возможность делать запросы к БД без явного написания SQL-скриптов [9]. Из достоинств можно выделить отличную реализацию ORM, с помощью которой можно формировать и запускать сложные SQL-скрипты. Из недостатков можно отметить: отсутствие метода переноса написанного кодом выражения в текст явного SQL-запроса.

“goracle” — библиотека для взаимодействия с базой данных Oracle; не поддерживает ORM, поэтому для написания запросов кодом — следует применить библиотеки для SQL-генерации [10]. У этой библиотеки есть более свежая и оптимизированная версия — “godror” [10], но она не поддерживается Alpine версией ОС Linux.

2.3.3. “aws-sdk-go”

Последняя версия библиотеки для взаимодействия с хранилищем данных, предоставляющая как базовые возможности загрузки-выгрузки данных, так и возможность мультипарт-загрузки больших файлов [11].

2.3.4. Docker

Для локального тестирования и сборки на стендах применялся Docker [12], а в частности задача написания корректных сценариев в Dockerfile.

2.3.5. Goose

Легковесная утилита, написанная на Go и позволяющая с помощью команд командной строки накатывать миграции в проект [13].

3. Реализация

3.1. Архитектура

Для решения данной задачи были проанализированы возможные механики действий и на их основании было принято разделить систему на два микросервиса: первый микросервис (здесь и далее ПМ) реализует парсинг 2-ух типов JSON-файлов и заносит информацию о данных в них в некоторую промежуточную базу, второй микросервис (здесь и далее ВМ) реализует логику перенесения данных из всех таблиц источников (в том числе и из промежуточной таблицы) в файлы формата CSV.

Также в системе должны использоваться: Prometheus — для анализа и диагностики метрик, Elasticsearch — для лёгкого поиска по логам приложения, целевой и промежуточной баз данных.

Упрощённые архитектурные схемы ПМ и ВМ представлены на рисунках 1 и 2 соответственно.



Рисунок 1 — Архитектура первого микросервиса

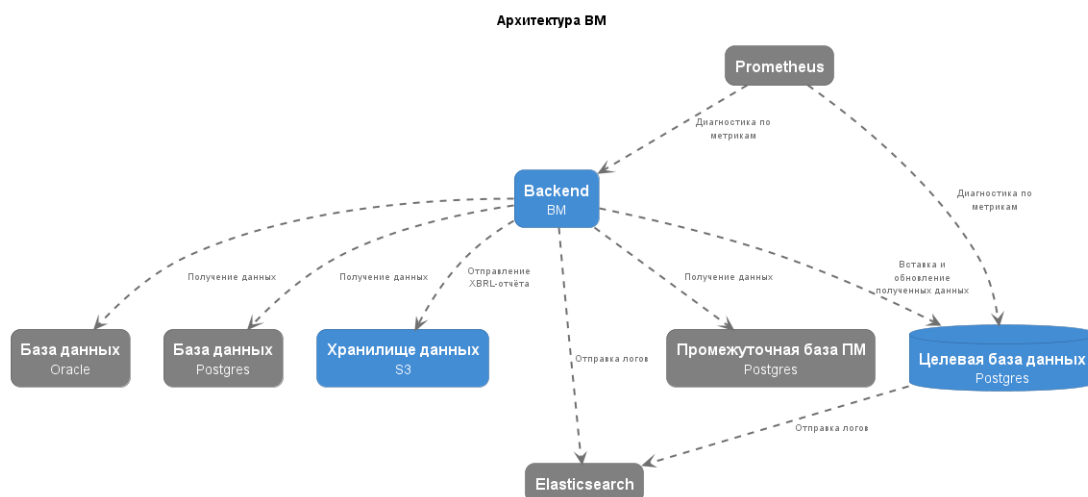


Рисунок 2 — Архитектура второго микросервиса

3.2. Модели данных

Для целевой и промежуточной баз данных были спроектированы модели данных с учётом всех требований. Упрощённая схема модели данных (без указания конкретных полей таблиц) для ПМ представлена на рисунке 3.

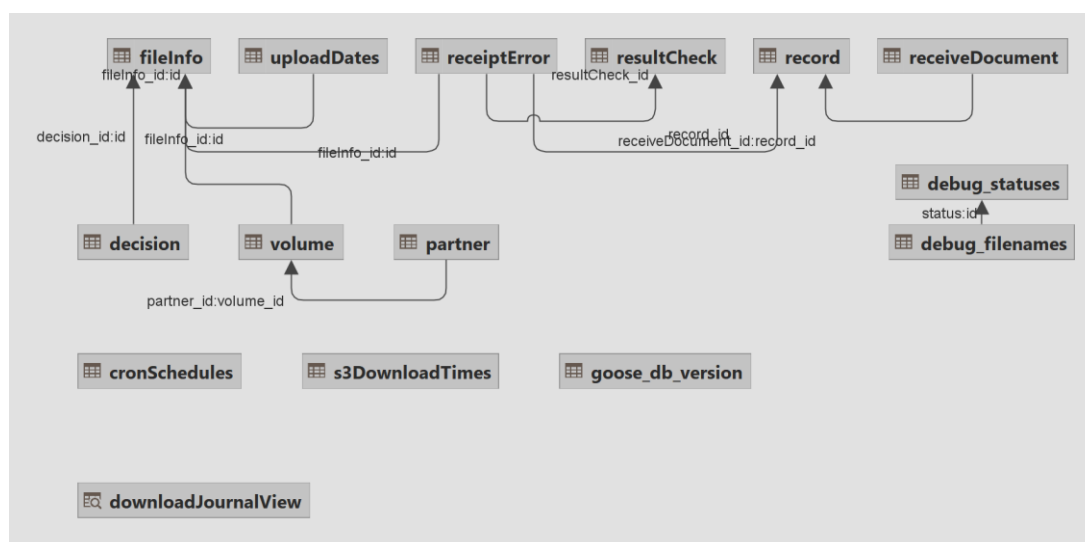


Рисунок 3 — Модель данных первого микросервиса

Рассмотрим таблицы данной модели:

- «fileInfo», «uploadDates», «receiptError», «resultCheck», «record», «receiveDocument», «decision», «volume» и «partner» — это таблицы соответствующие структуре JSON-файлов, связи между которыми так же повторяют эту структуру;
- «s3DownloadTimes» — это таблица синхронизации ПМ с S3 хранилищем. Синхронизация в системе устроена таким образом, что в этой таблице сохраняется дата, до которой все JSON-файлы из конкретного бакета считаются встреченными системой и занесёнными в БД. Дата эта сверяется с полем «LastModified» каждого объекта в хранилище;
- «cronSchedules» — таблица хранения расписания шедулера (п. 3.3);
- «goose_db_version» — таблица хранящая информацию о примененных миграциях в данную БД;
- «debugFileNames» и «debugStatuses» — таблицы для отладки и тестирования, хранящие список встреченных из S3-хранилища файлов и статус их обработки;
- «downloadJournalView» — представление, охватывающее все поля всех таблиц, хранящих данные JSON-файлов.

Так же приведём упрощённую схему модели данных (без указания конкретных полей таблиц) для ВМ на рисунке 4.



Рисунок 4 — Модель данных второго микросервиса

Рассмотрим таблицы данной модели:

«form0420754r2» и «form0420754r2_tmp_ids», «form0420754r3» и «form0420754r3_tmp_ids», «form0420754r4» и «form0420754r4_tmp_ids», «form0420755r2» и «form0420755r2_tmp_ids», «form0420755r3» и «form0420755r3_tmp_ids», «form0420755r5» и «form0420755r5_tmp_ids» — таблицы для хранения подготовленных данных для конкретных CSV-отчетов и таблицы для хранения временных данных для отделения уже загруженных по времени;

«cronSchedules» — таблица хранения расписания шедулера (п. 3.3);

«goldenCards», «goldenCardsSubjects», «goldenCardsSubjectsFor55r5» — справочные данные для отчёта 762 для разных типов контрагентов в XBRL-отчете.

«taskAudits», «taskTypes», «taskStatuses» — таблицы для хранения информации о всех запускаемых тасках любого типа: от задачи маппинга в целевую таблицу до задачи сборки самого CSV-отчета;

«errorDictionary» — таблица справочной информации по расшифровкам кодов ошибок в принимаемых сервисом данных;

«notFoundSins» — таблица для отладки и тестирования, хранящая информацию по контрагентам, отсутствующим в справочной информации;

«lastCheckDates» — это таблица синхронизации ВМ с базами-источниками, из которых ВМ забирает данные. Синхронизация происходит по полям даты в каждой конкретной таблице-источнике;

«goose_db_version» — таблица хранящая информацию о применённых миграциях в данную БД.

3.3. Реализация шедулера

Для того, чтобы наше решение было автоматическим и не требовало присутствия человека для запуска, было принято реализовать шедулер, запускающий по расписанию набор параллельных тасок, а именно: в ПМ шедулер запускает таски загрузки и парсинга JSON-файлов в БД, в ВМ шедулер запускает таски загрузки данных из таблиц-источников, таски сбора XBRL-отчетов и таску системы приоритета (п. 3.4). Так же шедулер имеет собственную таску, проверяющую расписание запуска тасок, хранящееся в таблице cronSchedules. Расписание задаётся CRON-строкой и меняется по API. Пример хранения расписания представлен на рисунке 5.

	id	tag	cronString	created_at
1	9164007b-b8f2-4b27-a79b-23d8ac31a124	CheckerTask	* / 1 * * * *	2023-12-20 22:22:24.025063 +00:00
2	2c5c1a2f-5af2-46f9-92f8-286e7d348ae1	FirstNoticeTask	* / 1 * * * *	2023-12-20 22:22:24.032491 +00:00
3	c417d43b-96cf-4967-98b9-7b44aea65d65	SecondNoticeTask	* / 1 * * * *	2023-12-20 22:22:24.035695 +00:00

Рисунок 5 — Таблица хранения расписания тасок шедулера

3.4. Реализация системы приоритета

Задача абстрактного слоя с шедулером — запустить параллельно работающие задачи. Однако, не все задачи должны быть параллельны с другими, например: в таблице-источнике для данных отчёта 0420754 раздел 4 содержится так же и справочные данные для других отчётов, которые будут собираться в других параллельных задачах, а значит эта задача должна выполняться раньше остальных. Таким образом был написан модуль, который принимает на вход параллельно запущенные шедулером задачи для конкретного времени в расписании и сортирует их запуск в порядке приоритета. Приоритет был определен следующий:

- 1) Задачи загрузки данных из таблиц-источников, содержащих справочные данные;
- 2) Задачи загрузки данных из таблиц-источников, не содержащих справочные данные;
- 3) Задачи сборки XBRL-отчетов по основным отчётам и разделам;
- 4) Задачи сборки XBRL-отчетов, являющихся справочниками к собранным отчетам.

Так же стоит упомянуть, что система приоритета следит за выполнением каждой задачи, чтобы не запустить новую подобную задачу, пока не завершилась предыдущая, и не допустить дублирования действий.

3.5. «Самооркестрация» инстансов системы

Оркестрация микросервисов — это управление и контроль процессов, происходящих между микросервисами. Но что если один и тот же микросервис запущен одновременно N-раз на нескольких разных виртуальных машинах, подах кластеров Kubernetes, компьютеров сотрудников и при этом собирать данные в одну и ту же целевую таблицу и помещать готовые отчеты в одно и то же хранилище данных? Не будет ли нарушена целостность данных? Модуль

системы, который следит за количеством экземпляров приложения, их синхронизацией и корректным завершением — было принято называть «самооркестрацией».

В случае запуска одного единственного экземпляра приложения — синхронизация по загружаемым из таблиц-источников данным тривиальна. Но при одновременном включении множества экземпляров системы (без необходимости знать где именно они запущены) можно получить дублирование данных и некорректную сборку. Для решения этой проблемы был написан модуль самооркестрации, имеющий данный алгоритм:

- 1) При включении приложения — приложение регистрируется в таблице синхронизации.

- 2) Если это единственное работающее на данный момент приложение - оно становится и помечается в таблице как master-приложение, иначе — как slave-приложение.

- 3) Каждое приложение при выполнении тасок в таблице синхронизации отмечает область данных с которыми будет работать.

- 4) Если приложение выключилось или завершилось из-за ошибки — оно помечается в таблице как завершённое, а работу над его областью данных будет проводить master-приложение. Если упавшее приложение — это master, то первое успешное запросить права приложение — становится master-ом.

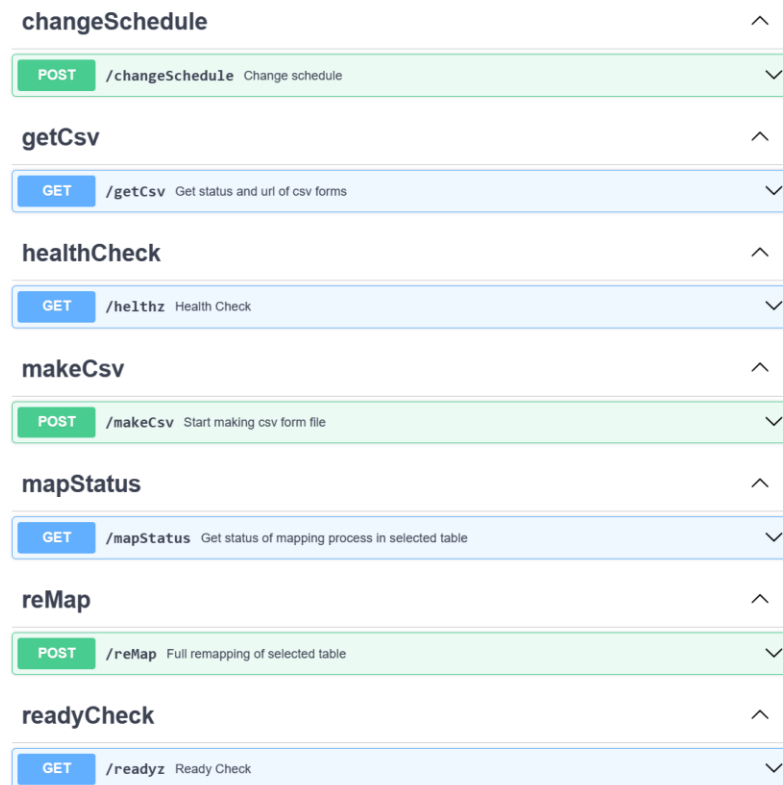
- 5) Master-приложение так же отвечает и за проверку завершённости работы над областями данных других экземпляров.

Таким образом, система может быть запущена одновременно на не связанных машинах и при этом обеспечивать целостность загружаемых данных.

3.6. API-методы

Для взаимодействия с системой были написаны следующие HTTP-методы:

1) Для ВМ:



changeSchedule	^
POST /changeSchedule Change schedule	▼
getCsv	^
GET /getCsv Get status and url of csv forms	▼
healthCheck	^
GET /helthz Health Check	▼
makeCsv	^
POST /makeCsv Start making csv form file	▼
mapStatus	^
GET /mapStatus Get status of mapping process in selected table	▼
reMap	^
POST /reMap Full remapping of selected table	▼
readyCheck	^
GET /readyz Ready Check	▼

Рисунок 6 — Пример Swagger-документации с API-методами для ВМ

/changeSchedule — метод изменения расписания шедулера;

/reMap — метод полного очищения таблицы с данными для отчёта и запуска загрузки данных с самого начала;

/mapStatus — метод получения статуса задачи загрузки данных для отчёта;

/makeCsv — метод запуска задачи формирования отчёта;

/getCsv — метод получения статуса задачи формирования отчёта и URL на готовый отчет, если задача успешно завершена;

/readyz и /helthz — это Healthcheck-методы, пингующие приложение и его подключённые компоненты соответственно.

Т.к. система задумывалась как автоматическая, то данные ручки скорее предназначены для решения проблемных ситуаций.

2) Для ПМ — система более автоматическая и просто парсит JSON-файлы, поэтому API-методы там следующие:

/changeSchedule — метод изменения расписания шедулера;

/readyz и /helthz — это Healthcheck-методы, пингующие приложение и его подключенные компоненты соответственно.

3.7. Юнит-тесты

Для тестирования работоспособности системы были проведены и интеграционные тесты, и нагрузочные тесты. Но, касаясь кода, были разработаны юнит-тесты, проверяющие корректность некоторых важных функций. Покрытие кода юнит-тестами в обоих микросервисах составляет около 40%.

4. Расчеты компонентов

Произвести расчет памяти, необходимой для каждой системы, следует для того, чтобы определить параметры машины, на которой будет работать приложение. Вычислить память необходимо для всего, что вводится в эксплуатацию в ходе нашего решения, а именно: промежуточная база данных ПМ, целевая база данных ВМ и S3-хранилище, в которое будет выгружаться отчёт ВМ.

4.1. Расчёт промежуточной базы данных ПМ.

Для успешного расчёта стоит уточнить вводные данные:

- Все данные из JSON-файла заносятся в базу
- В год загружается около 8000 JSON-файлов разных размеров
- После 5 лет данные будут удаляться, т.к. считаются устаревшими.
- В JSON-файлах хранятся сведения, отправленные некоторой другой системой; JSON-файлы представляют из себя заголовки и массив ошибок некоторой операции.
- Имеется 2 типа JSON-файлов, но из себя они представляют одно и то же, просто с иными полями.

Путём перебора файлов в S3 - было проанализировано количество JSON-файлов и их вес. Было получено, что для первого типа JSON-файлов самый большой файл весит 50МБ, а самый маленький «безошибочный» файл — 500Б; для второго же типа - самый большой весит 10ГБ, а самый маленький «безошибочный» файл — 500Б. Поэтому далее в вычислениях было принято опираться на память, занимаемую вторым типом файлов. Стоит отметить, что «нормальное» (разовое и счётное) количество ошибок в файле приводит к его весу в ≈ 3 КБ. Так же оказалось, что файлы начали активно формироваться с 2021-ого года. Детализация приведена в таблице.

Первый тип JSON-файлов						
Год	Всего	≥ 10 ГБ	≥ 1 ГБ	≥ 1 МБ	≥ 3 КБ	≥ 500 Б
2023	8851	0	0	445	1328	7078
2022	7921	0	0	475	1030	6416
2021	8209	0	0	328	1149	6732
Второй тип JSON-файлов						
Год	Всего	≥ 10 ГБ	≥ 1 ГБ	≥ 1 МБ	≥ 3 КБ	≥ 500 Б
2023	8624	0	4	842	1382	6396
2022	7845	1	5	763	1098	5978
2021	8185	1	3	786	1228	6167

Таблица 1 — Статистика по JSON-файлам

Для первого типа файлов выведем средние показатели за все года:

Первый тип JSON-файлов					
Всего	≥10ГБ	≥1ГБ	≥1МБ	≥3КБ	≥500Б
≈8300	0	0	≈420	≈1170	≈6740

Таблица 2 — Усреднённая статистика по первому типу JSON-файлов

Для второго типа файлов выведем средние показатели за все года:

Второй тип JSON-файлов					
Всего	≥10ГБ	≥1ГБ	≥1МБ	≥3КБ	≥500Б
≈8200	≈1	≈4	≈800	≈1240	≈6180

Таблица 3 — Усреднённая статистика по второму типу JSON-файлов

Рассчитаем память, занимаемую обоими типами файлов за год:

$$V = (0 + 1) \times 10000 + (0 + 4) \times 1000 + (420 + 800) \times 1 + (1170 + 1240) \times 0,003 + (6740 + 6180) \times 0,0005 = 15234 \text{ МБ} = 15,2\text{ГБ}$$

Соответственно, за пять лет выходит выделение памяти равное:

$$V = 5 * 15,2\text{ГБ} = 76\text{ГБ}$$

Т.к. жёстких дисков на 76ГБ не существует, то минимально достаточный диск для хранения файлов за 5 лет - диск на 128ГБ.

4.2. Расчёт целевой базы данных ВМ и S3-хранилища.

Так как таблицы-источники для сбора отчетов содержат в себе персональные данные, то получить с продакшн-контура данные для анализа не представлялось возможным. Однако, организацией были предоставлены тестовые данные (за квартал), которые следовало считать приближенными к реальным. Так же стоит отметить, что таблицы-источники физически находятся в разных базах, но для вычислений это не будет учитываться. Детальное описание данных и их объёма представлено в таблице.

Таблица-источник	Отчёт	Количество записей, шт	Вес, МБ
1	0420754p2	19423	3,4
2	0420754p3	992400	196
3	0420754p4	430116	112
	0420762	430116	108
4	0420755p2	130680	19
5	0420755p3	138200	29
6	0420755p5	65000000	6500
	04207562	65000000	18000

Таблица 4 — Статистика по количеству записей и объему целевых таблиц

Рассчитаем память, занимаемую этими данными за 1 квартал:

$$V = 3,4 + 196 + 112 + 108 + 19 + 29 + 6500 + 18000 = 24967,4 \text{ МБ}$$

Т.к. в году 4 квартала, а хранить данные следует за 5 лет, то:

$$V = 4 * 5 * 24967,4 = 499348 \text{ МБ} = 499 \text{ ГБ}$$

Самый минимальный существующий жёсткий диск, удовлетворяющий требованию — это 512ГБ. Соответственно его стоит использовать для базы данных.

Так же стоит учесть, что по данным из этих таблиц каждый квартал будет формироваться XBRL-отчет, который будет весить столько же, сколько и данные в таблице за квартал. Значит вычисления для базы верны и для S3-хранилища с отчётами. Однако, стоит учесть и человеческий фактор, а именно: сборка отчётов возможна и при помощи API-запросов. Соответственно, сотрудники могут собирать отчёты и за другие периоды, поэтому для хранилища стоит рассматривать жёсткий диск размером от 1024ГБ.

Заключение

Таким образом, была проделана работа по созданию системы автоматического формирования XBRL-отчетов, а именно:

- подобраны оптимальные технологии для реализации системы;
- разработана архитектура с разделением на микросервисы;
- созданы модели данных для эффективного хранения данных в базах;
- рассчитаны параметры баз данных;
- учитаны требования и тонкости сборки отчётов;
- написаны API-методы для взаимодействия с микросервисами;
- протестированы отдельные функции с помощью юнит-тестов.

Также имеется и план дальнейшего развития приложения:

- добавления логики создания новых отчётов;
- обработка нового типа JSON-файлов первым микросервисом;
- добавление новых API-методов и интеграция с общей Frontend-

частью приложения организации.

Список источников

- [1] Learn Microsoft — URL: <https://learn.microsoft.com/ru-ru/dynamics365/business-central/bi-create-reports-with-xbrl> (дата обращения 16.02.2023)
- [2] Банк России. Открытый стандарт отчётности XBRL — URL: https://cbr.ru/projects_xbrl/ (дата обращения 16.02.2023)
- [3] Банк России. Форма 0420754 «Сведения об источниках формирования кредитных историй» — URL: https://cbr.ru/explan/ot_bki/forma-0420754/ (дата обращения 16.02.2023)
- [4] Банк России. Форма 0420755 «Сведения о пользователях кредитных историй» — URL: https://cbr.ru/explan/ot_bki/forma-0420755/ (дата обращения 16.02.2023)
- [5] Банк России. Форма 0420762 «Реестр контрагентов» — URL: https://cbr.ru/explan/ot_bki/forma-0420762/ (дата обращения 16.02.2023)
- [6] Prometheus — URL: https://prometheus.io/docs/tutorials/visualizing_metrics_using_grafana/ (дата обращения 16.02.2023)
- [7] Alpine Linux — URL: <https://www.alpinelinux.org/about/> (дата обращения 16.02.2023)
- [8] Go.Dev Blog — URL: https://go.dev/doc/effective_go (дата обращения 16.02.2023)
- [9] Go-pg Documentation — URL: <https://pg.uptrace.dev/> (дата обращения 16.02.2023)
- [10] Goracle Documentation — URL: <https://github.com/go-goracle/goracle> (дата обращения 16.02.2023)
- [11] AWS SDK for Go v2 — URL: <https://aws.github.io/aws-sdk-go-v2/docs/> (дата обращения 16.02.2023)
- [12] Docker Documentation — URL: <https://docs.docker.com/manuals/> (дата обращения 16.02.2023)

[13] Goose Official Page — URL: <https://github.com/pressly/goose> (дата обращения 16.02.2023)