

Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 23.M07-мм

Разработка высоконагруженной системы формирования бизнес-отчётности формата XBRL

Киреев Андрей Андреевич

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:

доцент кафедры СП Луцив Д.В.

Консультант:

руководитель отдела «Смарт-прайсинг» Малашин Р.В.

Место работы: ООО «ОЗОН Технологии»

Санкт-Петербург

2024

Оглавление

Введение	3
1. Постановка задачи	6
2. Технический обзор	7
2.1. Выбранные технологии	7
2.2. Архитектура	7
3. Реализация	14
3.1. Часть автоматизации	14
3.2. Взаимодействие с пользователем	16
3.3. Мониторинг приложения	19
Заключение	22
Список источников	24

Введение

Организациям и банкам необходимо каждый квартал отчитываться в Центральный банк Российской Федерации (он же Банк России, Центробанк, ЦБ РФ), сообщая историю финансовых взаимодействий, сведения по кредитным источникам и контрагентам. Оформление этих сведений сводится к формированию общего для всех XBRL-отчёта. XBRL — это расширяемый язык деловой отчётности (от англ. extensible Business Reporting Language), формат передачи регуляторной, финансовой и иной отчётности [1]. XBRL-отчёт — 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 установленными правилами и требованиями ЦБ РФ [2]. Т.к. данные для передачи, процесс их хранения и взаимодействия индивидуальны - нет готового универсального решения для каждой отдельной компании. Это является основной предпосылкой к созданию автоматизированной системы сбора XBRL-отчёта.

В работе прошлого семестра были уточнены конкретные отчёты, необходимые для формирования в системе:

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 отчёт.

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

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

Из-за того, что создание отчётов на текущий момент проводится с помощью обученных сотрудников, имеется необходимость в системе, описанной в этой работе, которая должна автоматизировать весь процесс. Необходимо позволить системе работать самостоятельно в фоне, а также реализовать возможность взаимодействия со специально обученными пользователями, которые могут манипулировать автоматизированной в нестандартных или экстренных случаях, например, внеплановый сбор отчёта или нарушение целостности передаваемых данных.

В работе второго семестра продолжается работа над системой, а именно: необходимо разработать API-методы взаимодействия с

сервисом, добавить метрики для наблюдения за приложением и проработать архитектуру автоматизированной части.

1. Постановка задачи

Целью работы является добавление в разрабатываемую систему автоматизации и взаимодействия с пользователем. Для достижения цели были поставлены следующие задачи:

1. Описать спроектированную архитектуру системы, разобрать внутреннее устройство приложения.
2. Привести концепцию работы созданной автоматической части системы.
3. Разработать API-методы к сервису для управления приложением вне автоматической части.
4. Реализовать сбор метрик для мониторинга приложения.

2. Технический обзор

2.1. Выбранные технологии

В работе прошлого семестра проводился сравнительный анализ технологий, с помощью которых может создаваться данная система. Стоит привести основные из них.

- 1) «Golang» - язык программирования, на котором создаётся система, имеет эффективные инструменты для работы с многопоточными приложениями и достаточный функционал «из коробки» [6].
- 2) «go-pg» версии 10 - последняя в момент написания приложения версия самой распространённой ORM-библиотеки для доступа и операций над базами данных «PostgreSQL».
- 3) «Goracle» — библиотека для взаимодействия с базой данных Oracle, напрямую выполняющая чистые SQL-скрипты.
- 4) «aws-sdk-go» - базовая библиотека для взаимодействия с хранилищем данных, поддерживающая возможность мультипарт-загрузки больших файлов.
- 5) «Docker» - инструмент контейнеризации для локального тестирования и сборки приложения на стендах.
- 6) «Goose» - легковесная утилита, позволяющая накатывать миграции в проект.

2.2. Архитектура

Как было указано в работе прошлого семестра, одно из условий для разработки системы - наличие трёх ресурсов-источников: база «PostgreSQL», база «Oracle» и хранилище данных

«Serph». Так как тип источника «хранилище данных» отличается от ряда других, требует дополнительных действий и иной обработки искомых JSON-файлов - было предложено ввести отдельный микросервис, задача которого сканировать «Serph» S3 на наличие новых файлов, обрабатывать их и записывать данные из них в собственную базу данных «PostgreSQL». А уже при наличии 3 баз-источников ввести второй микросервис, который взаимодействует с базами данных на удобном для агрегации языке SQL и формирует из их данных бизнес-отчёт.

Таким образом, 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.

Архитектурные схемы ПМ и ВМ представлены на рисунках 1 и 2 соответственно.

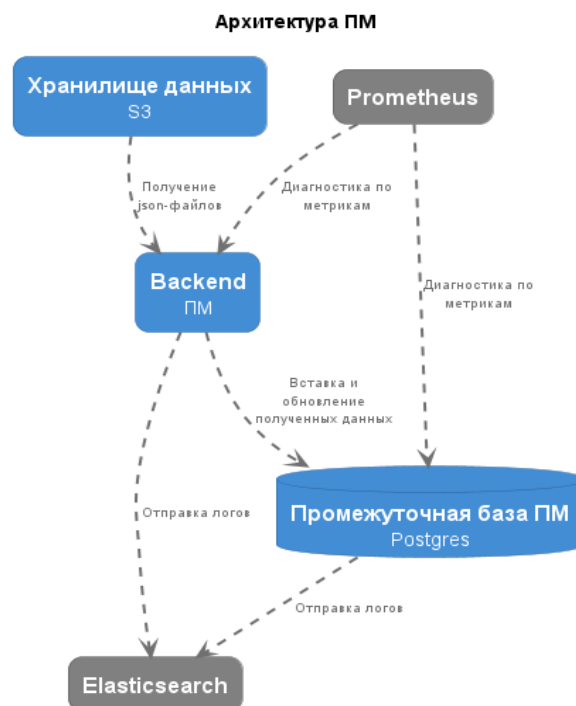


Рисунок 1 — Архитектура первого микросервиса

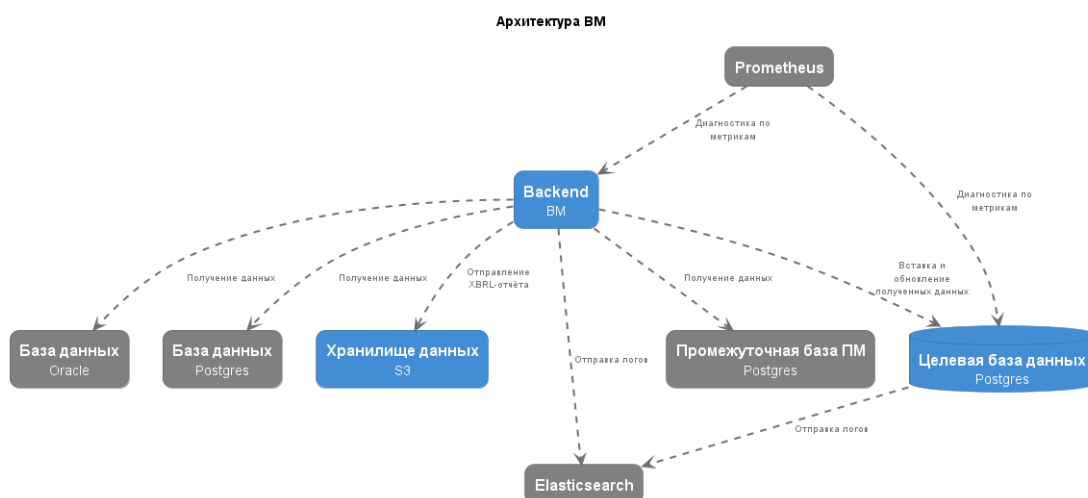


Рисунок 2 — Архитектура второго микросервиса

На архитектурной схеме первого микросервиса изображено взаимодействие backend-части сервиса с хранилищем данных, выступающем в роли источника, из которого ПМ забирает искомые JSON-файлы. Далее внутри себя сервис выполняет парсинг файла, маппинг его в структуры и отправляет на запись в промежуточную

базу ПМ. Также ПМ и база ПМ отдают собственные логи в «Elasticsearch» и метрики в «Prometheus». Для «Elasticsearch» логи пишутся в JSON-формате [7], за которыми сервис ходит в разрабатываемую систему. А для «Prometheus» написан пакет «monitoring» с интересующими функциональными метриками, которые «Prometheus» забирает из сервиса[8].

Описанную логику можно кратко представить с помощью диаграммы компонентов первого микросервиса на рисунке 3.

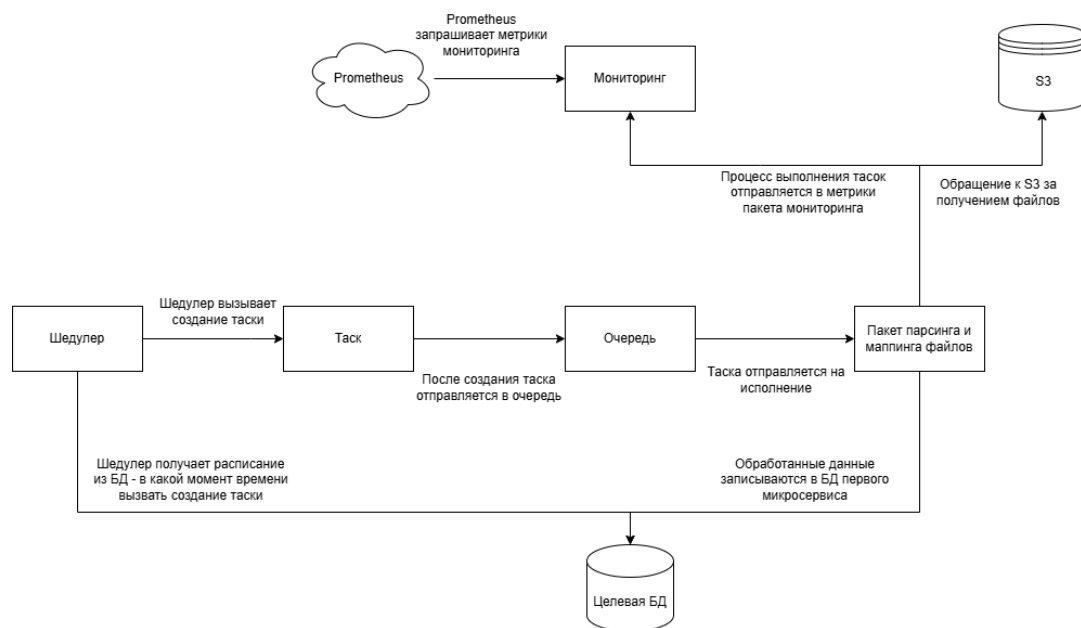


Рисунок 3 — Диаграмма компонентов ПМ

Кодовая реализация первого микросервиса сосредоточена в директории «internal» (рисунок 4). Описание имеющихся в директории пакетов - следующее:

- config - пакет для парсинга и применения конфига приложения;
- domain - пакет основных структур и помощников в маппинге;
- handler - пакет маршрутизации API-методов;
- monitoring - пакет с описанными метриками приложения, отдаваемыми в «Prometheus»;
- service - пакет с целевой логикой приложения;

- core - пакет с инициализацией объектов структур, регистрацией хендлера и метрик мониторинга;
- logic - логика основных действий приложения;
- mapping - пакет с функциями загрузки файлов, правилами агрегирования данных и маппинга в структуры БД;
- task - пакет автоматизации работы приложения, автоматическое создание тасок по указанному расписанию или добавление их в очередь по API.

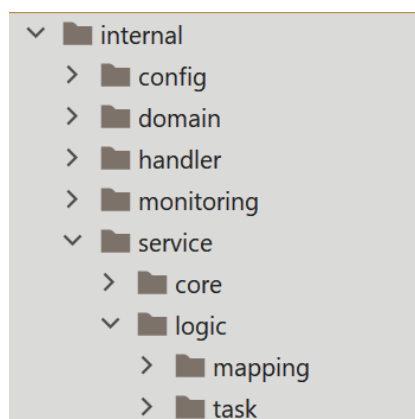


Рисунок 4 — Каталог “internal” первого микросервиса

На архитектурной схеме второго микросервиса изображено взаимодействие backend-части сервиса с двумя базами данных «PostgreSQL» и базой «Oracle», выступающих в роли источников, из которых ВМ забирает данные из указанных таблиц. Внутри себя сервис более детально агрегирует данные из таблиц-источников с учётом даты и времени, количества записей, проведённых вычислений и наличия уникальных строк согласно бизнес-требованиям. Далее подготовленные строки ВМ аналогично собирает и загружает в собственную базу данных. А уже в процессе создания отчёта, вызванного автоматически или с помощью API - ВМ собирает необходимые строки из таблиц собственной БД, создаёт CSV-отчёт на их основе и отправляет его в S3-хранилище

данных, откуда пользователь по ссылке сможет его получить. Также ПМ и база ПМ отдают собственные логи в «Elasticsearch» и метрики в «Prometheus».

Описанную логику можно кратко представить с помощью диаграммы компонентов первого микросервиса на рисунке 5.

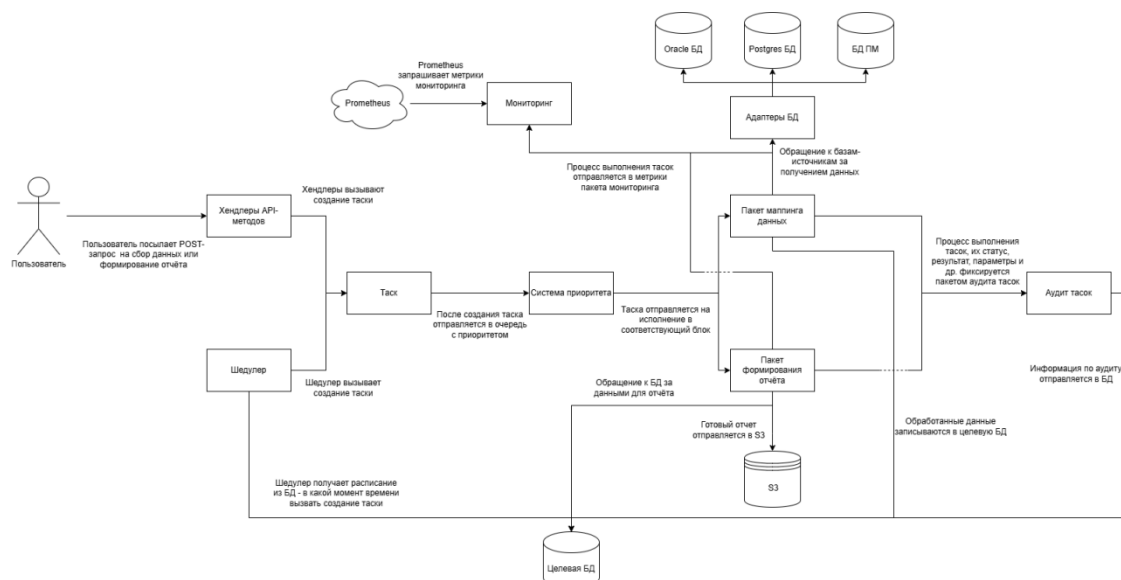


Рисунок 5 — Диаграмма компонентов ПМ

Кодовая реализация второго микросервиса также сосредоточена в директории «internal» (рисунок 6). Описание имеющихся в директории пакетов - следующее:

- config - пакет для парсинга и применения конфига приложения;
- domain - пакет основных структур и помощников в маппинге;
- handler - пакет маршрутизации API-методов;
- monitoring - пакет с описанными метриками приложения, отдаваемыми в «Prometheus»;
- service - пакет с целевой логикой приложения;
- adapters/oracleAdapter - пакет с подключением к базе-источнику данных «Oracle» и основными к ней запросами;

- adapters/pg1Adapter - пакет с подключением к базе-источнику данных “PostgreSQL” и основными к ней запросами;
- adapters/pg2Adapter - пакет с подключением к базе данных “PostgreSQL”, относящейся к ПМ, и основными к ней запросами;
- core - пакет с инициализацией объектов структур, регистрацией хендлера и метрик мониторинга;
- logic - логика основных действий приложения;
- audit - пакет, ведущий историю запросов к приложению на выполнение основных функций, их статус и введённые параметры;
- csvMakers - пакет, отвечающий за сбор бизнес-отчётов;
- mappers - пакет, отвечающий за загрузку и агрегацию над данными, получаемыми из баз-источников, а также проводящий необходимые дополнительные вычисления;
- priority - пакет с внутренней системой приоритета выполнения тасок;
- task - пакет автоматизации работы приложения, автоматическое создание тасок по указанному расписанию или добавление их в очередь по API.

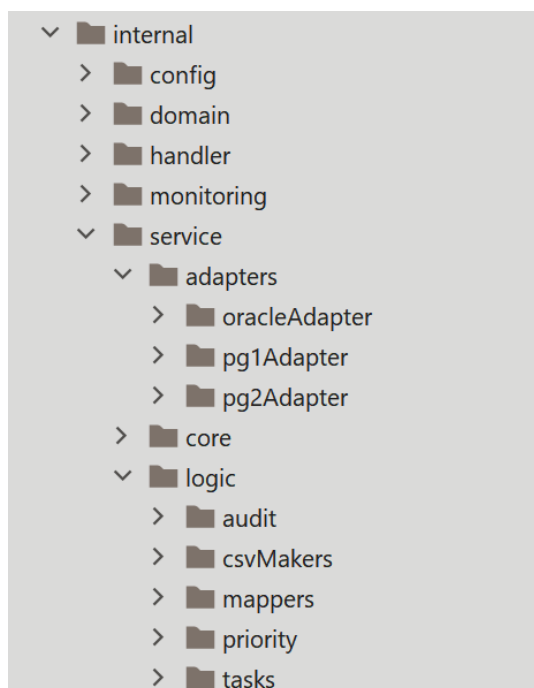


Рисунок 6 — Каталог “internal” второго микросервиса

3. Реализация

3.1. Часть автоматизации

Одна из основных проблем, которую наша система призвана решить - это проблема неавтоматизированного текущего решения. Поэтому для автоматизации нашего решения, исключающего необходимость участия человека при выгрузке данных и формировании отчёта - был написан и внедрён шедулер, который по расписанию запускает несколько параллельных задач. В частности, в ПМ шедулер активирует задачи загрузки и парсинга JSON-файлов в базу данных, а в ВМ — задачи загрузки данных из таблиц-источников, сбор XBRL-отчётов и задачу системы приоритетов. Также шедулер имеет свою задачу, которая проверяет расписание запуска задач, хранящееся в таблице «cronSchedules». Расписание

задаётся с помощью CRON-строки и может изменяться через API. Пример хранения расписания показан на рисунке 7.

	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 7 — Таблица хранения расписания тасок шедулера

Задача слоя с шедулером — организовать выполнение параллельных задач. Однако здесь можно столкнуться с тем, что по бизнес-требованиям не все задачи могут быть выполнены одновременно. Например, таблица-источник для отчёта 0420754, раздел 4, содержит справочные данные для других отчётов, которые обрабатываются в параллельных задачах. Этот факт требует, чтобы данная задача выполнялась первой. В связи с этим был разработан модуль, который получает параллельные задачи, запущенные шедулером в конкретное время, и сортирует их по приоритету. Модуль был условно назван системой приоритета. Он написан на каналах «горутин», с помощью которых параллельные таски могут блокировать ресурсы при своём исполнении и разблокировать по завершению. Модуль слушает эти каналы и выбирает, какая таска будет исполняться следующей. Система приоритета следит за тем, чтобы новые задачи не запускались до завершения предыдущих, предотвращая дублирование и имплементируя тем самым простейшую очередь.

Также стоит учесть требование, что приложение может быть развёрнуто одновременно и на нескольких Kubernetes-подах, и даже на отдельных компьютерах сотрудников. И при этом оно должно загружать данные в одну и ту же базу. Задача кажется тривиальной, когда нет необходимости контролировать целостность собираемых

данных, ежеминутную проверку источников на появление новых данных и малые размеры таблиц. Однако в реальной ситуации приходится учитывать проблемы обеспечения целостности данных, отсутствия дубликатов и сохранения таблиц «1 к 1». Для предотвращения этих ошибок был разработан модуль самооркестрации, который работает по следующему алгоритму:

- 1) При запуске приложение регистрируется в таблице синхронизации.
- 2) Если это единственный активный инстанс, он становится master-приложением и помечается как таковой в таблице; если нет — помечается как slave-приложение.
- 3) При выполнении задач каждое приложение отмечает в таблице синхронизации область данных, с которой оно работает.
- 4) Если приложение завершается или отключается из-за ошибки, оно помечается как завершенное, а его задачу берет на себя master-приложение. Если завершившимся было master, то первым запросившее права приложение становится новым master.
- 5) Master-приложение также контролирует завершение задач других инстансов.

Таким образом, система может запускаться на нескольких несвязанных машинах, при этом гарантируя целостность загружаемых данных.

3.2. Взаимодействие с пользователем

API-методы — это наборы функций и протоколов, которые позволяют различным программам взаимодействовать друг с другом[9]. Именно через них планируется общение пользователя с

программой, в будущем пользователя должна заменить клиентская часть приложения.

Стоит рассмотреть более детально некоторые API-методы, представленные в работе первого семестра:

1) POST /changeSchedule — метод изменения расписания шедулера, присутствует в ПМ и ВМ.

Формат запроса - JSON:

```
{
  "cronStr": "*/1 * * * *", // CRON-строка с расписанием шедулера
  "tag": "Map54r2Task" // тег задачи, под которым она сохранена в шедулере
}
```

Формат ответа - JSON:

```
{
  "info": "", // информационное сообщение об ошибке
  "status": "Done" // статус вызванного метода
}
```

2) GET /getCsv - метод получения статуса задачи формирования отчёта и URL на готовый отчет, если задача успешно завершена.

Формат запроса - аргументы запроса в адресной строке:

```
/getCsv?formFiles=0420754r2.csv,0420754r3.csv&fromDate=2024-01-01&toDate=2024-09-01
```

Формат ответа - JSON:

```
{
  "info": "", // информационное сообщение об ошибке
  "status": "Done", // статус вызванного метода
  "urls": [ // ссылки на отчеты в s3-хранилище
    "http://some-url-on-bucket/1",
    "http://some-url-on-bucket/2",
    "...",
  ]
}
```

3) POST /makeCsv - метод запуска задачи формирования отчёта.

Формат запроса - JSON:

```
{
  "formFiles": [ // файлы к сборке
    "0420754r2.csv",
    "0420754r3.csv",
    "0420755r5.csv"
  ],
  "fromDate": "2000-01-01",
  "toDate": "2030-12-12"
}
```

Формат ответа - JSON:

```
{
  "info": "Please do GET-request", // информационное сообщение о
  // необходимости выполнить /getCsv
  "status": "Pending", // статус выполнения задачи
  "url": "http://some-host/getCsv?params..." //сгенерированная ссылка
  // для /getCsv
}
```

4) GET /mapStatus — метод получения статуса задачи загрузки данных для отчёта.

Формат запроса - аргументы запроса в адресной строке:

/mapStatus?tableName=form0420754r3

Формат ответа - JSON:

```
{
  "info": "", // информационное сообщение об ошибке
  "status": "Done" // статус задачи маппинга
}
```

5) POST /reMap — метод полного очищения таблицы с данными для отчёта и запуска загрузки данных с самого начала.

Формат запроса - JSON:

```
{
  "tableName": "form0420754r2"
}
```

```
}
```

Формат ответа - JSON:

```
{  
  "info": "Please do GET-request", // информационное сообщение о  
необходимости выполнить /mapStatus  
  "status": "Pending", // статус задачи маппинга  
  "url": "http://some-host/mapStatus?params..." // сгенерированная  
ссылка для /mapStatus  
}
```

3.3. Мониторинг приложения

Т.к. разрабатываемая система способствует автоматизации процесса и работает на параллельно запущенных задачах - за ходом их выполнения необходимо наблюдать. Для этого были продуманы и разработаны метрики, реализация которых находится в пакете «monitoring» и собирается из сервиса средствами «Prometheus».

Опишем разработанные метрики и то, какую информацию они собирают в первом микросервисе. Инициализация метрик в ПМ указана на рисунке 8.

1) «notice_time_interval_metric» - время, которое задача затратила на парсинг и обработку одного набора JSON-файлов.

2) «notice_records_gauge» - количество записей в одном наборе JSON-файлов.

3) «notice_average_time» - среднее время обработки N-числа записей, где N задано из конфига.

```

func newMetrics(noticeNValue int) *metrics { 1 usage  ⤵ Andrey Kireev *
    m := &metrics{
        noticeNValue: float64(noticeNValue),
        noticeTime: prometheus.NewGaugeVec(prometheus.GaugeOpts{
            Name: "notice_time_interval_metric",
            Help: "Time interval from start map from sources to end map",
        }, []string{noticeLabel}),
        noticeRecords: prometheus.NewGaugeVec(prometheus.GaugeOpts{
            Name: "notice_records_gauge",
            Help: "Gauge counter of mapped Records",
        }, []string{noticeLabel}),
        noticeAvg: prometheus.NewHistogramVec(prometheus.HistogramOpts{
            Name: "notice_average_time",
            Help: "Average Time of processing N Records",
            Buckets: prometheus.ExponentialBuckets(start: 0.01, factor: 2, count: 10),
        }, []string{noticeLabel}),
    }
    return m
}

```

Рисунок 8 - Инициализация метрик ПМ

Опишем также и разработанные метрики ВМ. Инициализация метрик в ПМ указана на рисунке 9.

1) «mapping_time_interval_metric» - время, которое задача затратила на маппинг набора строк из источника.

2) «mapping_records_gauge» - количество записей в одном наборе строк из источника.

3) «mapping_average_time» - среднее время обработки задачей N-числа записей в процессе маппинга, где N задано из конфига.

4) «csvMake_time_interval_metric» - время, которое задача затратила на набор сбора строк и построение части отчёта.

5) «mapping_records_gauge» - количество записей в одном наборе строк из БД.

6) «mapping_average_time» - среднее время обработки задачей N-числа записей в процессе сбора отчёта, где N задано из конфига.

```

func newMetrics(mappingNValue, csvMakeNValue int) *metrics {
    m := &metrics{
        mappingNValue: float64(mappingNValue),
        mappingTime: prometheus.NewGaugeVec(prometheus.GaugeOpts{
            Name: "mapping_time_interval_metric",
            Help: "Time interval from start map from sources to end map",
        }, []string{mapLabel}),
        mappingRecords: prometheus.NewGaugeVec(prometheus.GaugeOpts{
            Name: "mapping_records_gauge",
            Help: "Gauge counter of mapped Records",
        }, []string{mapLabel}),
        mappingAvg: prometheus.NewHistogramVec(prometheus.HistogramOpts{
            Name: "mapping_average_time",
            Help: "Average Time of processing N Records",
            Buckets: prometheus.ExponentialBuckets(0.01, 2, 10),
        }, []string{mapLabel}),

        csvMakeNValue: float64(csvMakeNValue),
        csvMakeTime: prometheus.NewGaugeVec(prometheus.GaugeOpts{
            Name: "csvMake_time_interval_metric",
            Help: "Time interval from start csv making from sources to end csv making",
        }, []string{csvLabel}),
        csvMakeRecords: prometheus.NewGaugeVec(prometheus.GaugeOpts{
            Name: "csvMake_records_gauge",
            Help: "Gauge counter of Records for csv make",
        }, []string{csvLabel}),
        csvMakeAvg: prometheus.NewHistogramVec(prometheus.HistogramOpts{
            Name: "csvMake_average_time",
            Help: "Average Time of processing N Records",
            Buckets: prometheus.ExponentialBuckets(start: 0.01, factor: 2, count: 10),
        }, []string{csvLabel}),
    }
    return m
}

```

Рисунок 9 - Инициализация метрик VM

Заключение

В ходе данной работы были достигнуты следующие результаты:

1. Была актуализирована база выбранных технологий и сформирована архитектура системы. Оптимальное разделение на два микросервиса позволит декомпозировать основную задачу формирования отчётности. А добавленные диаграммы компонентов упрощают понимание внутренней структуры системы.

2. Также была описана концепция внутренней логики работы автоматической части. Добавлены и проработаны шедюлер с созданием тасок и система приоритета с внутренней очередью. Описан блок динамического определения мастер-приложения, в случае когда система работает параллельно в множественном экземпляре и требуется сохранять целостность данных.

3. Разработаны и API-методы, позволяющие пользователям манипулировать приложением вне основной автоматической части. Это позволит решить аварийные ситуации и случаи запуска формирования отчётности не по расписанию. У перечисленных методов указан минимально необходимый набор параметров в запросах и ответах, его достаточно на начальном этапе работы с системой.

4. Также добавлен и слой мониторинг, в котором разработаны оптимальные метрики, которые позволят следить за приложением. Случаи серьёзного отклонения графиков и их некорректных показателей - могут стать поводом для анализа ошибок или изменения внутренних алгоритмов.

Стоит выделить возможности к дальнейшему росту приложения. В следующей работе стоит описать схему поддержки высокой нагрузки приложения и провести её тестирование. А также разработать функциональную клиентскую часть.

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] Официальная документация языка Golang. Введение. — URL https://go.dev/doc/effective_go (дата обращения 18.10.2024)

[7] Документация Elasticsearch. Структурирование логирования. — URL: <https://www.elastic.co/blog/structured-logging-filebeat> (дата обращения 18.10.2024)

[8] Документация Prometheus. Использование Prometheus в Golang приложениях. — URL: <https://prometheus.io/docs/guides/go-application> (дата обращения 18.10.2024)

[9] Платформа Хабр. API от А до Я. — URL: <https://habr.com/ru/articles/768752/> (дата обращения 08.10.2024)