

Санкт-Петербургский государственный университет

Разработка высоконагруженной системы формирования бизнес-отчётности формата XBRL

КИРЕЕВ Андрей Андреевич

Отчёт по преддипломной практике
в форме «Производственное задание»

Научный руководитель:

доцент кафедры Системного Программирования Луцив Д.В.

Консультант:

руководитель отдела «Смарт-прайсинг» Гришин В.С.

Место работы: ООО «Озон Технологии»

Санкт-Петербург

2025

Оглавление

Введение	3
1. Постановка задачи	5
2. Обзор	6
2.1. Требования	6
2.2. Аналоги	8
2.3. Технологии реализации	9
2.4. Условия нагрузочного тестирования	11
3. Архитектура	14
4. Разработка	25
4.1. Серверная часть	25
4.2. Клиентская часть	36
5. Тестирование	43
5.1. Нагрузочное тестирование	43
5.2. Апробация системы	48
Заключение	58
Список источников	60
Приложение 1	63
Приложение 2	64
Приложение 3	66
Приложение 4	67
Приложение 5	69

Введение

Организациям и банкам Российской Федерации необходимо каждый квартал отчитываться в Центральный Банк (ЦБ), сообщая историю финансовых взаимодействий и сведения по кредитным источникам и контрагентам. Оформление этих сведений сводится к формированию общего для всех XBRL-отчёта, где XBRL (Extensible Business Reporting Language) – это расширяемый язык деловой отчётности [1]. Соответственно, XBRL-отчёт является файлом, передаваемым отчитывающейся финансовой организацией, в котором указывается деятельность этой организации в формате установленным правилами и требованиями ЦБ [2]. Так как данные для передачи, процесс их хранения и взаимодействия индивидуальны для каждой бизнес-организации, то нет готового универсального решения. Это является проблемой, которая должна быть решена.

Компания “Озон Технологии” выполняет техническое обслуживание как основного бизнеса в сфере интернет-маркетинга, так и внутренних процессов компании. Внедрение автоматизированного решения в процесс сбора внутренней бизнес-отчётности – является одним из проектов, над которым работают сотрудники организации. Проект реализуется для внутреннего отдела организации, отвечающего за ведение бизнес-отчётности, и призван повысить скорость и надёжность в сборе XBRL-отчётов. На текущий момент подготовка бизнес-отчётности в организации проводится с помощью компетентных и обученных сотрудников, выделенных специально для работы над данной задачей. Эта специфика является одним из недостатков. Выполнение работниками однообразных рутинных операций, необходимость обучать сотрудников правилам создания конкретного отчёта, затрачиваемое на подготовку отчёта время – всё это можно решить с помощью внедрения автоматизации.

Организацией были предприняты попытки написать простые программные скрипты для манипуляции с данными, используемыми в формировании отчётов. Однако такая полумера не стала полноценным и универсальным средством решения проблемы. В данном случае обновление схем сбора отчётов сильно затрудняется, поскольку ручное распространение новых версий отчётов между сотрудниками влечёт лишние действия и контроль за донесённой информацией. Этот негативный фактор приводит к необходимости в реализации единой точки сбора отчётов, просмотр и генерация которых доступна всем заинтересованным сотрудникам пользовательский интерфейс. Единая точка формирования отчётов должна помочь ускорить процесс обновления схем сбора, а клиентоориентированный интерфейс увеличит скорость доступа до нужных данных.

Поскольку потенциальное решение будет доступно всем сотрудникам отдела подготовки бизнес-отчётности и планируется к нему неоднократно обращаться – стоит уделить особое внимание его работоспособности под соответствующей нагрузкой. В конце каждого квартала года, когда готовые XBRL-отчёты необходимо передавать в вышестоящие инстанции, планируется рост запросов к решению. Как правило количество запросов зависит от количества потенциальных пользователей и частоты их обращения к информационной системе. Следовательно, для того, чтобы создаваемое решение сохраняло стабильную работу во время повышенной нагрузки, стоит провести нагрузочные тесты и соотнести их с целевым значением RPS¹.

Таким образом, все вышеописанные факторы свидетельствуют о критических недостатках текущего подхода к выполнению излишних однотипных задач. Появляется особая необходимость в создании автоматической информационной системы.

¹ RPS - количество запросов в секунду

1. Постановка задачи

Таким образом, целью работы является создание системы автоматического формирования бизнес-отчётности формата XBRL, способной работать в условиях повышенной нагрузки. Для достижения цели были поставлены следующие задачи.

1. Провести обзор предметной области, полученных требований и выбранных технологий реализации.
2. Спроектировать архитектуру системы.
3. Разработать структуру моделей данных. Реализовать серверную и клиентскую часть приложения.
4. Провести апробацию системы, нагрузочное тестирование, сделать вывод о пиковых (предельных) возможностях системы.

2. Обзор

Перед приёмом проекта на исполнение был получен ряд сформулированных требований от бизнес-заказчиков. Типы отчётов, входные и выходные данные, особенности работы и среды запуска системы были оговорены до старта процесса разработки. Также исходя из внутренней специфики внутренней инфраструктуры компании были рекомендованы нижеописанные технологии реализации системы. Для уточнения деталей разрабатываемого решения стоит по пунктам разобрать предлагаемые условия и ограничения.

2.1. Требования

Количество и типы потенциальных XBRL-отчётов, которые в перспективе должны будут формироваться создаваемой системой исчисляется десятками. Однако для первого запуска испытаний системы организацией были определены для формирования следующие 8 типов отчётов.

А) 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.

Б) 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.

В) Форма 0420762 «Реестр контрагентов» [5].

- Справочник для всех вышеописанных форм кроме формы 0420755 раздел V.
- Опциональный справочник для формы 0420755 раздел V.

Требования к создаваемой системе также продиктованы бизнес-заказчиками и обусловлены спецификой внутренней политики компании. Таким образом были выдвинуты следующие бизнес-требования к разработке.

1. Выходным результатом работы системы должны являться 8 файлов формата CSV². Формат разрешён ЦБ РФ и удобен для проверки компетентными сотрудниками при проведении первых тестовых запусках системы.
2. Входными данными являются данные из нижеописанных источников:
 - хранилище данных «Ceph S3» [8], содержит 3 типа квитанций в виде файлов формата JSON³;
 - база данных «PostgreSQL» [9], 1 таблица-источник;
 - база данных «Oracle» [10], 4 таблицы-источника.

² CSV (Comma-Separated Values) – текстовый формат файлов для хранения табличных данных.

³ JSON (JavaScript Object Notation) – формат файлов хранения данных со структурой «ключ-значение».

Те данные из источников, которые старше пяти лет – можно считать неактуальными и их специфика не должна учитываться при разработке.

3. Система в штатном режиме должна работать автономно, без необходимости привлечения пользователя. Система должна автоматически раз в квартал собирать набор отчётов за прошедший период.
4. Система должна иметь необходимые элементы управления для проверки системы на первоначальных этапах и сборки отчётов за иные промежутки времени.
5. Поскольку система должна иметь элементы взаимодействия с пользователями, то она должна поддерживать одновременную работу для всех сотрудников департамента организации.
6. Разрабатываемой системе необходимо иметь небольшой удобный десктопный пользовательский интерфейс, который должен разрабатываться с учётом перспективы встраивания приложения в общую инфраструктуру компании.
7. Система должна собираться под «Alpine»⁴ версией ОС Linux, для последующей развёртки средствами «Kubernetes»⁵.
8. Система должна предусматривать георезервирование: запуск одного и того же экземпляра системы будет производиться на разных подах кластера и в разных ЦОД⁶, корректность работы системы не должна от этого зависеть.

2.2. Аналоги

Данная создаваемая система является узкоспециализированной, и ориентирована на задачи конкретной финансовой организации. Поэтому даже

⁴ Alpine – легковесный и безопасный дистрибутив ОС Linux.

⁵ Kubernetes – ПО для автоматизации развёртывания контейнеризированных приложений.

⁶ ЦОД – центр обработки данных.

при возможном уже реализованном схожем решении – детальная информация о нем будет закрытой.

При поиске аналогов можно найти как наиболее похожие варианты иностранных программ, например «UBPartner XPE & XBRL Toolkit» [6], так и российские организации, которые как раз предлагают платные услуги по созданию механизмов ведения XBRL-отчётов в организациях, например «Аванкор: XBRL-Портал» [7]. Всё это не является полноценным аналогом, который можно взять к сравнению.

2.3. Технологии реализации

Требования описанные ранее предполагают разделение системы на серверную часть и клиентскую интерфейсную часть. Серверная часть должна подключаться к разным видам баз данных и к хранилищу «Сeph S3». Сбор готовых сервисов наиболее удобен с помощью написания Dockerfile [11]. Также необходимо иметь подходящий инструмент для проведения нагрузочного тестирования, чтобы выявить возможности приложения во время работы под нагрузкой. Таким образом, был получен набор рекомендованных технологий, которые будут использоваться при разработке.

1. Golang (коротко: Go) – язык программирования, который поддерживает Backend⁷-разработку «из коробки»[12]. Преимущества языка описаны ниже.

- Все необходимое для разработки серверной части приложения есть в стандартном пакете Go.
- Go имеет достаточно простой синтаксис, схожий с C-языками.
- Имеются эффективные инструменты для работы с многопоточными приложениями. В Go массово используются

⁷ Backend – серверная часть приложения.

«горутин» — 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.

Также нужно уточнить используемые библиотеки в разрабатываемой системе, поскольку её специализация заключается в активном взаимодействии с источниками.

- “go-pg@v10” — последняя версия самой распространённой ORM⁸-библиотеки [13], предоставляющей возможность делать запросы к БД PostgreSQL.
- “goracle” — библиотека для взаимодействия с БД Oracle [14]. Не поддерживает ORM, поэтому для написания запросов кодом — следует применить библиотеки для SQL-генерации [10]. Имеется более свежая версия библиотеки — “godror” [10], но она не поддерживается Alpine версией ОС Linux.
- “aws-sdk-go/v2” — последняя версия библиотеки для взаимодействия с хранилищем данных, предоставляющая как базовые возможности загрузки-выгрузки данных, так и возможность мультипарт-загрузки больших файлов [15].

2. JavaScript — язык программирования, разработанный для добавления интерактивности веб-страниц, который будет использоваться в качестве написания клиентского интерфейса [16]. Является универсальным и основным языком, на котором реализуется большинство веб-страниц. Выбор конкретного фреймворка зависит от команды разработки.

⁸ ORM (Object Relational Mapping) — технология работы с базами данных без написания явных sql-скриптов.

3. “Goose” – при введении в эксплуатацию новых БД, потребуется легковесная утилита, позволяющая с помощью команд командной строки применять SQL-миграции в проект [17].
4. “Docker” – инструмент контейнеризации приложений, который используется для сбора сервисов системы.
5. “k6” – инструмент проведения нагрузочного тестирования, запускаемый из командной строки и требующий JavaScript-файл со сценарием тестирования [18]. Выбран к использованию благодаря визуализации результатов с помощью Grafana.

2.4. Условия нагрузочного тестирования

Поскольку одним из условий работы является получение корректного анализа способностей системы, то стоит для начала определиться с основными формулировками. Высоконагруженные приложения – это программные системы, способные эффективно обрабатывать и обслуживать большое количество одновременных запросов и пользователей, обеспечивая при этом стабильную работу [27]. Такие приложения часто используются в сферах, где критически важны высокая производительность и надежность, например, в онлайн-банкинге, социальных сетях, стриминговых сервисах и крупных интернет-магазинах.

Одними из показателей высокой нагрузки, по которым анализируют возможности приложения, являются следующие характеристики [18].

1. RPS (request per second) – количество запросов в секунду, которое способно выдержать приложение.
2. «Response Time» – время отклика приложения на входящие запросы.
3. «Failed requests» – количество ошибочных ответов от системы. При разработке необходимо минимизировать этот показатель.

Основной метрикой, которой оперируют разработчики высоконагруженных сервисов – это RPS. Определение количества запросов в

секунду, которое должно выдерживать приложение, зависит от множества факторов, включая внутреннюю специфику приложения, архитектуру, используемое оборудование и запланированные требования к нагрузке. Абсолютной градации RPS для определения какое приложение является низконагруженным, а какое высоконагруженным – не существует. Сервисы электронной коммерции или ведущих маркетплейсов могут стабильно выдерживать от нескольких десятков тысяч RPS до нескольких миллионов [28]. В то же время для сайтов небольших интернет-магазинов или образовательных порталов длительная стабильная нагрузка в 100 RPS может оказаться критичной. Поэтому для того, чтобы определить порог RPS для конкретной системы – необходимо знать количество одновременно взаимодействующих с системой пользователей.

Оценить стабильность сервиса под разным количеством RPS можно с помощью проведения нагрузочного тестирования. Нагрузочное тестирование – это тип тестирования производительности, при котором проверяется, как система ведёт себя при заданной нагрузке, то есть имитирует ситуацию, когда множество пользователей одновременно выполняют некоторые действия, а именно отправляют HTTP-запросы [29]. Нагрузочное тестирование отображает соотношение показателей RPS, «Response Time» и «Failed requests». Также нагрузочное тестирование может выявить ошибки и неточности в разработанной системе.

Наиболее предпочтительной утилитой для проведения нагрузочного теста является «k6», поскольку написание сценариев легко производится с помощью языка программирования JavaScript и средствами Grafana хорошо визуализирует итоговый результат. Для запуска k6 нужно создать сценарии нагрузки и запустить их следующей командой.

```
K6_WEB_DASHBOARD=true k6 run example.js
```

Также стоит определить характеристики тестирования.

- Каждый микросервис развёрнут на одном поде Kubernetes и работает на 1 ядре процессора. Это сделано для выявления поведения приложения в минимальных условиях.
- Процессор машины, на которой развёрнут Kubernetes: Intel Core i3 12100 (доступно 4 ядра).
- CPU Usage ограничен 90% для предотвращения падения машины.
- Параметры используемых баз данных и «Ceph S3» близки к «боевым» и поддерживают десятки тысяч RPS.

3. Архитектура

Данная глава посвящена описанию разрабатываемой системы в сборе и включает в себя описание компонентов системы, схемы их развёртки, сценарии выполнения основного функционала системы. Особое внимание уделяется таким процессам как: получение данных из квитанций, поскольку этим решается проблема неоднородности типов источников, а также формирование набора отчётов на основе собранных данных, так как это ключевой результат, в котором заинтересованы бизнес-пользователи.

Структурно исходя из сформулированных требований, можно выделить 3 основных отдельных процесса, а именно: обработка файлов квитанций из хранилища данных, сбор записей из баз-источников с последующим формированием набора отчётов и пользовательское взаимодействие с результатами. Поскольку каждый из этих процессов независим от другого, то их работу можно разбить на 3 отдельных микросервиса. Первые две функциональности реализуют бизнес-логику, а значит должны исполняться в зоне ответственности серверной части, т.е. каждый на отдельном Backend-сервере. Для их реализации будет использован язык Golang, как рекомендованный и эффективный инструмент создания серверных решений. Пользовательское взаимодействие – это клиентская часть, соответственно оно должно исполняться на Frontend-сервере и быть реализовано одним из фреймворков языка JavaScript, выбор которого описывается в этой главе.

Таким образом, было выявлено, что с точки зрения проектирования архитектуры для решения над задачей нужно ввести в эксплуатацию нижеописанные компоненты.

1. Frontend⁹-сервер, содержащий микросервис пользовательского интерфейса (здесь и далее кратко: МПИ) и обеспечивающий удобное взаимодействие пользователей с системой.

⁹ Frontend – клиентская часть приложения.

2. Backend-сервер №1, содержащий микросервис обработки квитанций (здесь и далее кратко: МОК) и отвечающий за обработку трёх различных типов файлов в формате JSON.
3. Базу данных МОК с СУБД¹⁰ «PostgreSQL», хранящую актуальные собранные данные из квитанций.
4. Backend-сервер №2, содержащий микросервис сбора отчётов (здесь и далее кратко: МСО) и отвечающий за сбор и агрегацию данных из разных баз-источников, а также за формирование готовых отчётов.
5. Базу данных МСО с СУБД «PostgreSQL», хранящую агрегированные и подготовленные к использованию данные из источников.
6. Дополнительное пространство в хранилище данных «Сeph S3» (отдельный бакет), в которое будут помещаться готовые отчёты.
7. Также должно быть настроено соединение со всеми и используемыми источниками и серверами работы с логами «Elasticsearch»¹¹ и серверами сбора метрик «Prometheus»¹².

Спроектированную архитектуру системы можно представить через диаграмму развёртывания на рисунке 1.

¹⁰ СУБД – система управления базами данных.

¹¹ Elasticsearch – система поиска по данным.

¹² Prometheus – система мониторинга метрик.

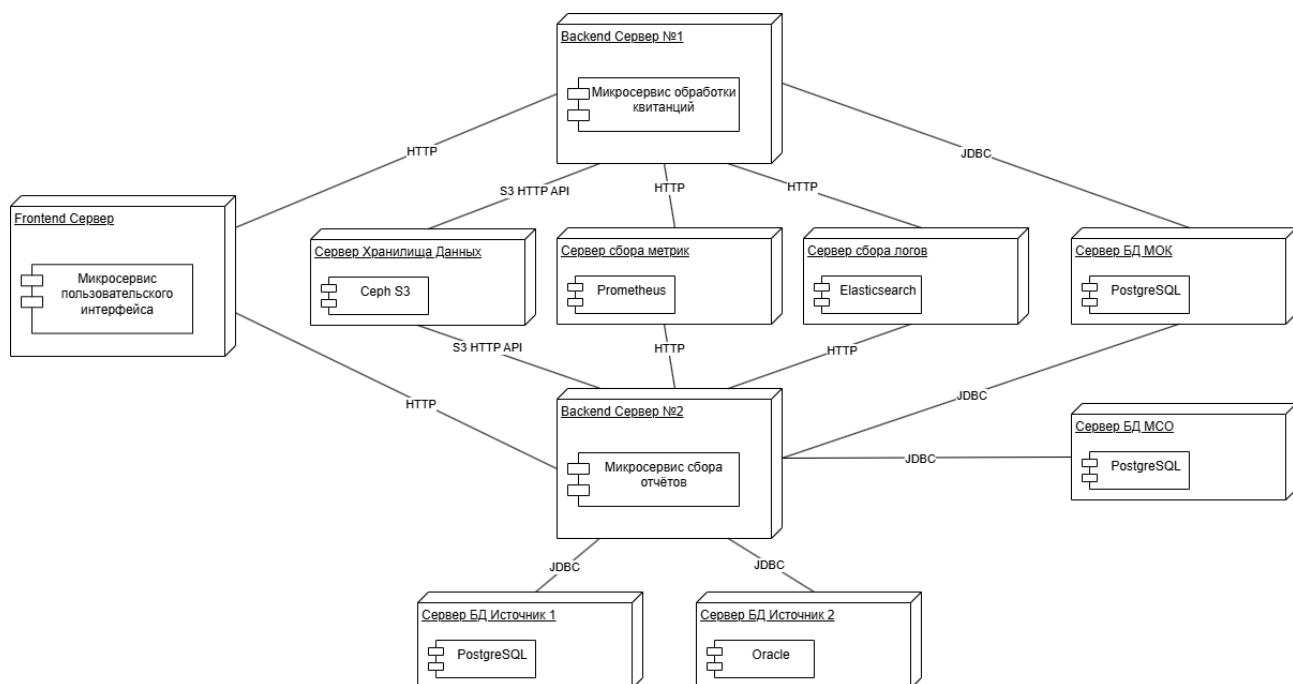


Рисунок 1 – Развёртка системы

Микросервис обработки квитанций. Задачей микросервиса обработки квитанций является перевод данных, полученных из квитанций JSON формата, в записи базы данных, которые подготовлены для дальнейших манипуляций. В микросервисе используется планировщик для автоматического запуска задач по расписанию, валидатор для определения подходящего типа файла и блок обработки квитанций. Наглядная визуализация микросервиса выполнена с помощью диаграммы компонентов на рисунке 2.



Рисунок 2 – Компоненты МОК

Основной процесс микросервиса обработки квитанций работает автономно и запускается планировщиком, который достаёт набор новых необработанных квитанций и, обрабатывая их, отправляет на запись в БД МОК. Верхнеуровнево визуализировать этот процесс можно с помощью диаграммы последовательности на рисунке 3, на котором актором иницирующим процесс выступает планировщик.

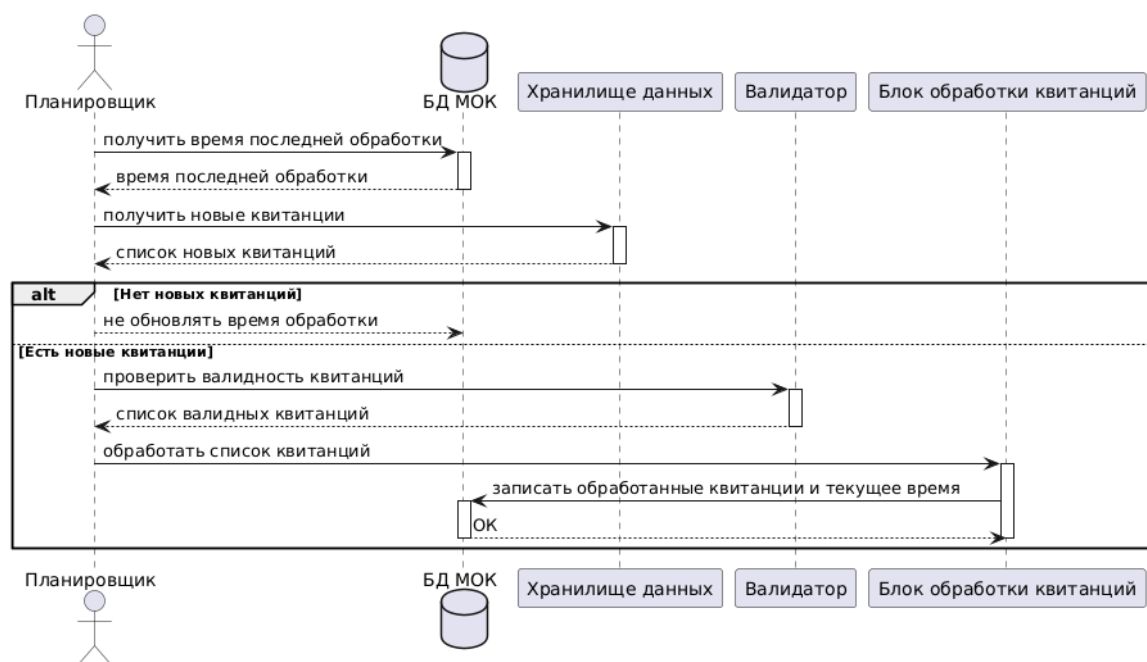


Рисунок 3 – Сценарий автоматической обработки квитанций

Аналогично этот процесс запускается ручной манипуляцией пользователя, который передаёт в HTTP-запросе наименование квитанции, которая будет обработана. Ручные манипуляции не являются главной задачей микросервиса, однако для понимания и этого процесса стоит привести диаграмму последовательности.

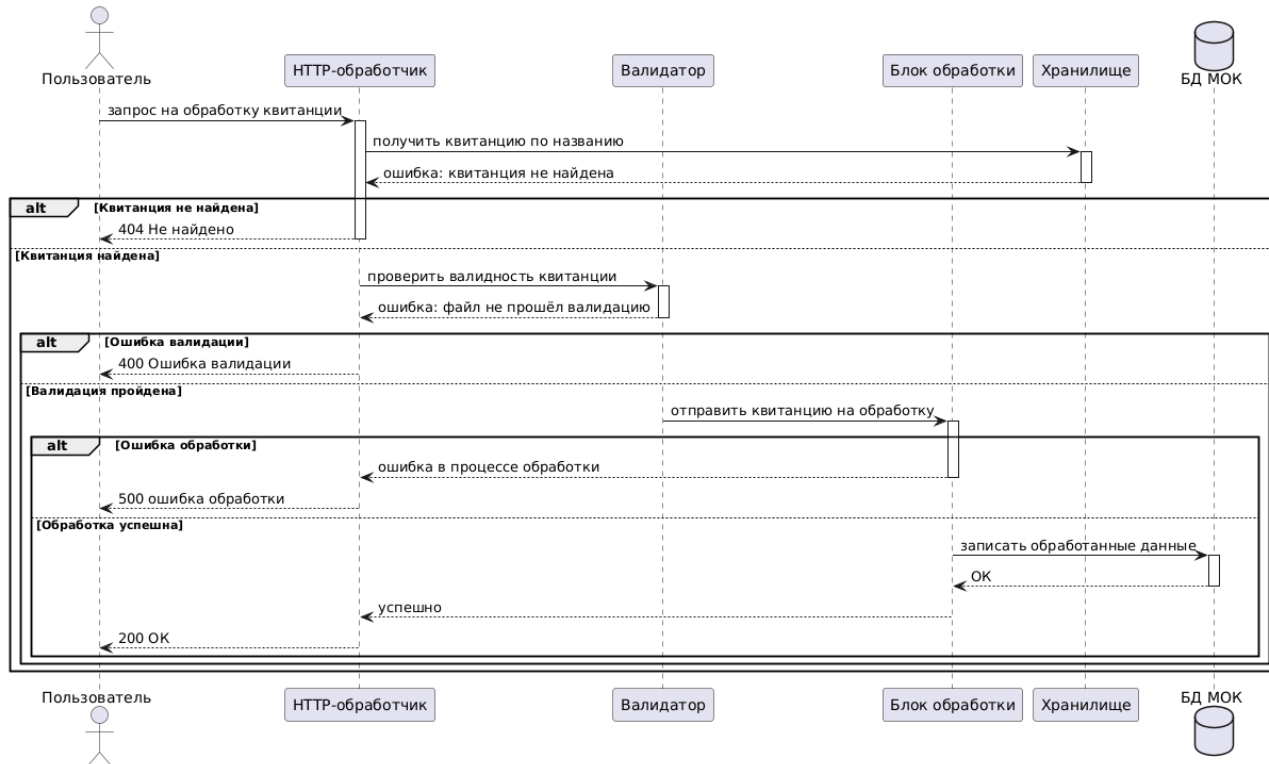


Рисунок 4 – Сценарий ручной обработки квитанции

Микросервис сбора отчётов. Задачи микросервиса сбора отчётов можно разделить на две основных: сбор данных из источников, где источниками являются базы данных, и формирование отчётов на основании собранных данных. Блок сбора данных преобразует, подсчитывает и агрегирует полученные значения, которые записываются в базу данных МСО в виде, готовом к использованию в отчёте. Блок формирования отчётов отвечает за получение данных из базы МСО, преобразования их в CSV формат и отправку в хранилище данных. Каждый из этих блоков создаёт задачу на исполнение либо по расписанию, либо по HTTP-запросу. Также имеется блок аудита задач, который вызывается по HTTP-запросу и получает детальную информацию о всех созданных задачах, в том числе и ссылки на созданные отчёты. Визуализация микросервиса представлена на рисунке 5.

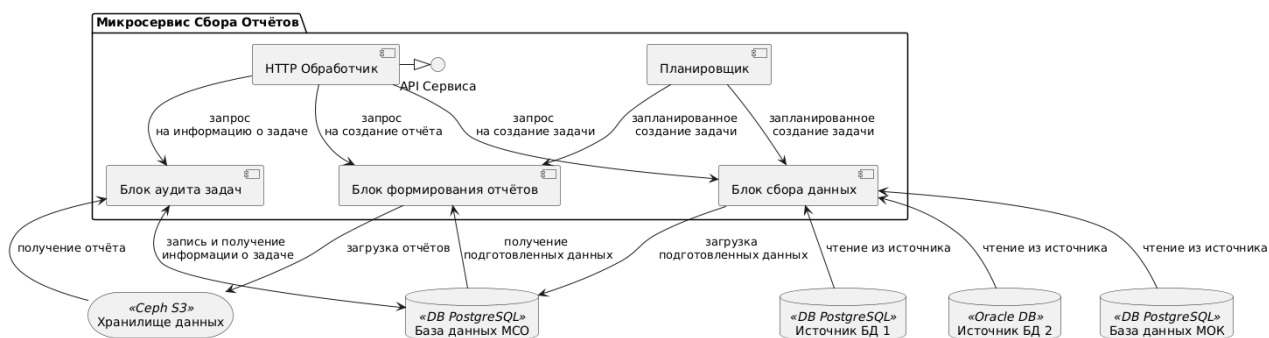


Рисунок 5 – Компоненты МСО

Для формирования готового отчёта необходимо для начала собрать данные из ряда источников. Планировщик по расписанию запускает процесс сбора данных, блок сбора данных агрегирует и обрабатывает данные из источников и отправляет на запись в собственную БД МСО. Абстрактно этот процесс можно визуализировать с помощью диаграммы последовательности на рисунке 6.

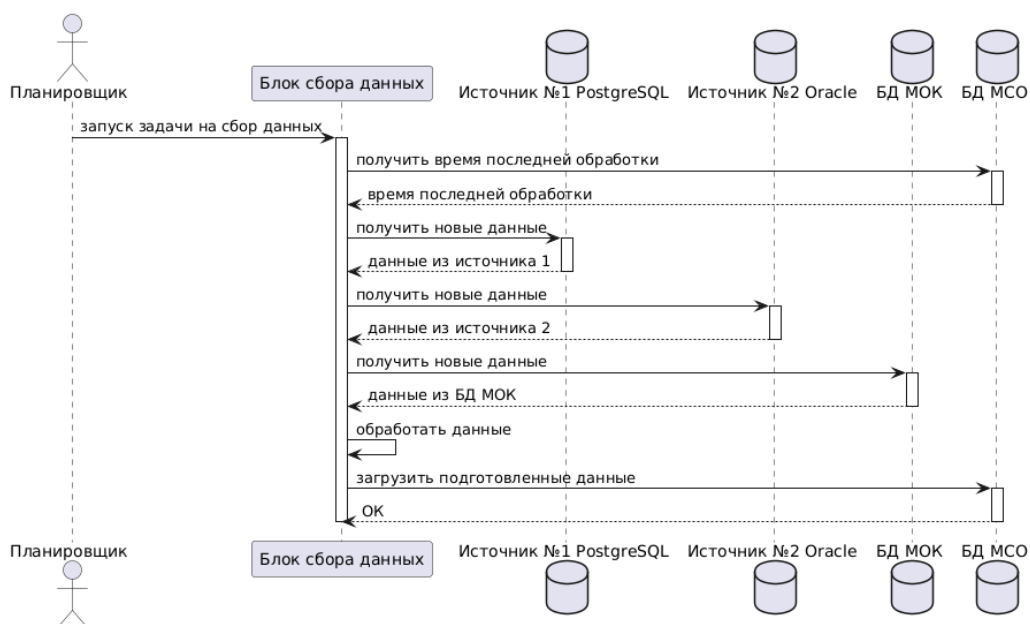


Рисунок 6 – Сценарий автоматического сбора данных

После того, как данные подготовлены, система готова к генерации отчётов. Запуск процесса генерации набора отчётов инициирует либо планировщик, либо HTTP-запрос пользователя. Непосредственно сам процесс генерации от метода запуска не зависит. Он заключается в сборе подготовленных данных из БД МСО, формировании файлов отчётов и отправке их в Хранилище данных.

Визуализировать процесс автоматического формирования отчётов и ручного можно наглядно на рисунках 7 и 8 соответственно.

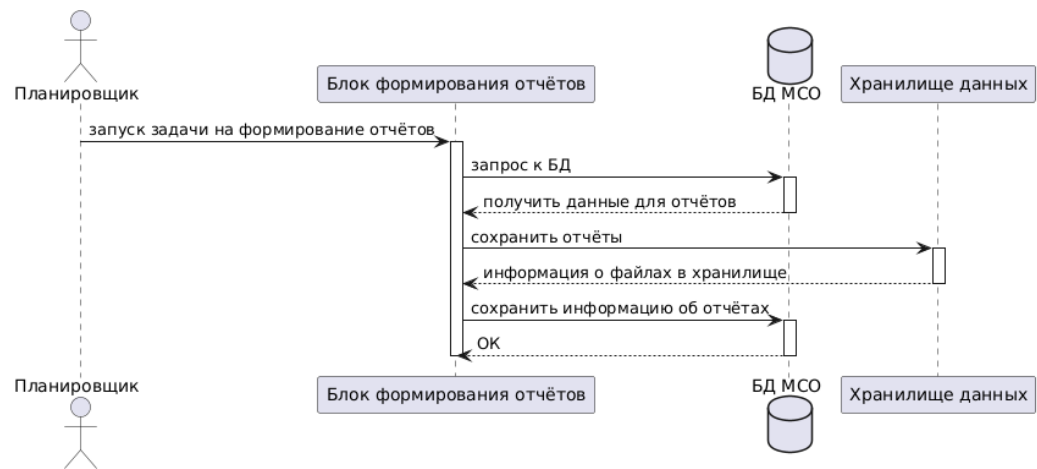


Рисунок 7 – Сценарий автоматического сбора отчётов

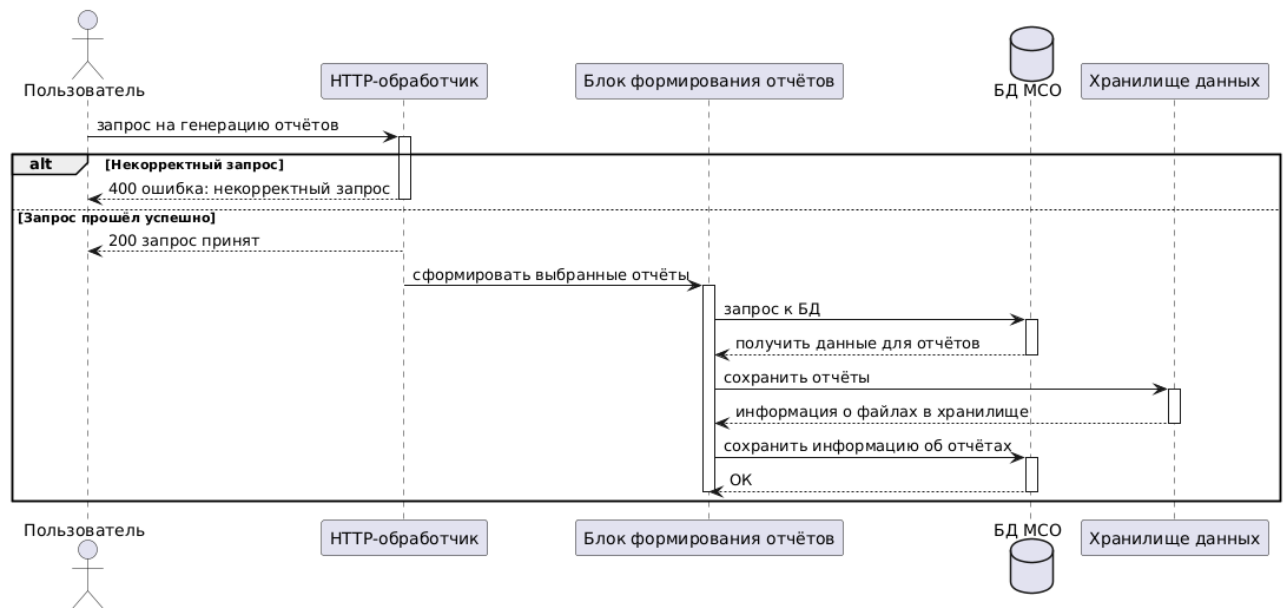


Рисунок 8 – Сценарий ручного сбора отчётов

Сгенерированные отчёты доступны для загрузки пользователем. Для это необходимо сделать запрос к МСО с набором названий отчётов, которые должны быть получены. Этот процесс представлен на рисунке 9.

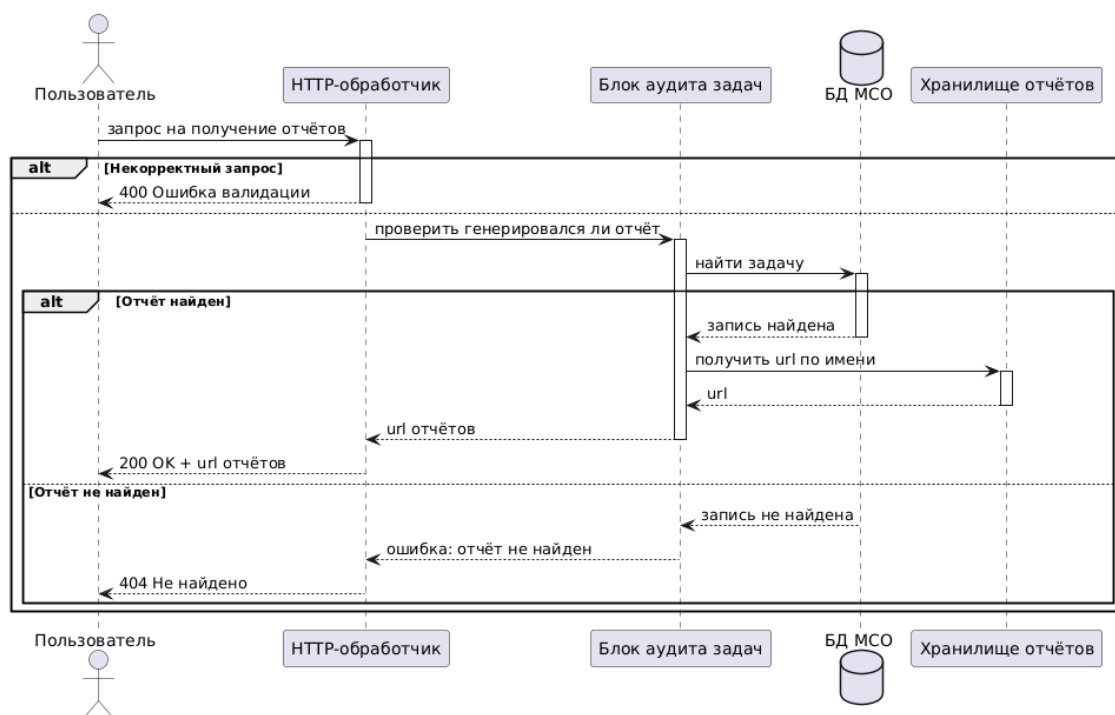


Рисунок 9 – Сценарий получения отчётов

Таким образом, была сформирована архитектура разрабатываемой системы, а также приведены диаграммы последовательности основных процессов, которые способствуют решения задачи автоматического сбора отчётов.

Микросервис пользовательского интерфейса. Поскольку от заказчиков было получено требование о реализации небольшого десктопного приложения, то в его реализации стоит опираться на успешную совместимость с Backend-решениями и удобством работы пользователей. На основе таблицы ниже стоит проанализировать основные инструменты для создания настольных приложений для компьютерных ОС.

Фреймворк / Библиотека	Преимущества	Недостатки
Electron.jsx[22]	<ul style="list-style-type: none"> - Лёгкая кроссплатформенность из коробки (можно перенести в браузерное решение) - Javascript поддерживает любые гибкости в построении интерфейса и визуализации данных - Простота и скорость написания скриптов 	<ul style="list-style-type: none"> - Большое потребление RAM/CPU чем у аналогов - Не малый размер итогового приложения засчёт движка Chromium

Фреймворк / Библиотека	Преимущества	Недостатки
	<ul style="list-style-type: none"> - DevTools для упрощения разработки - Можно использовать любую Javascript библиотеку напрямую 	
Qt (C++, PyQt)[23]	<ul style="list-style-type: none"> - Высокая производительность и нативный UI - Мощные инструменты собственные IDE (Qt Creator) - Большая коллекция UI-компонентов 	<ul style="list-style-type: none"> - Большая сложность языков разработки, необходимость знать специфику фреймворка - Меньше скорость разработки - Более сложная кроссплатформенность
JavaFX[24]	<ul style="list-style-type: none"> - Наличие кроссплатформенности - Хорошая производительность - FXML – декларативная разметка 	<ul style="list-style-type: none"> - Java не популярна среди UI-разработчиков - Устаревший WebView - Большой runtime, сложный деплой
.NET (WPF, WinForms)[25]	<ul style="list-style-type: none"> - Глубокая интеграция с Windows - Коллекция UI-компонентов - Инструменты от Microsoft (VS, XAML) 	<ul style="list-style-type: none"> - Ограничена Windows (WPF) - Сложности деплоя под Linux - Менее гибкий UI

Таблица 1 – Анализ инструментов разработки приложения

Проанализировав перечисленные инструменты, можно сказать, что наиболее подходящий вариант – это фреймворк «Electron.js». Его преимуществами можно считать важные для текущей системы – использование всех гибкостей построения интерфейсов средствами JavaScript, скорость разработки, кроссплатформенность и интуитивный GUI.

Для работы с фреймворком достаточно создать структуру приложения с помощью команд:

```
npm init -y
npm install electron --save-dev
```

Описать в секции скрипт «electron»:

```
"scripts": {
  "start": "electron ."
}
```

И запустить через стандартную команду: `npm start`

Клиентское приложение работает на предопisanном файле «main.js», в котором указываются все параметры программы как оконного приложения, а

далее оставшийся функционал работает через стандартную связку веб-инструментов: HTML, CSS и JavaScript.

Модели данных. Для двух вводимых баз данных сервисов МОК и МСО модели данных имеют свою некоторую специфику. Модель данных БД МОК структурно больше зависит по большей части от внутренней состава файлов. Каждая квитанция имеет свою специфику хранения данных, поэтому таблицы с содержанием основной информации повторяют их структуру. Аналогично таблицы основного содержания отчётов в БД МСО повторяют структуру выбранных к реализации XBRL-отчётов.

Однако стоит выделить некоторые технические таблицы, на которых построена работа системы. Часть из них одинакова для обеих баз данных, часть специфична для конкретной базы.

Стоит для начала привести общие специфические таблицы.

- «cronSchedules» – таблица хранения расписания планировщика. Поскольку роль системы заключается в автономной работе, то планировщик выполняющий запуск задач в назначенное время – имеет сохранённое расписание.
- «apps» – таблица регистрации экземпляров приложения и определения лидера. Система должна работать на нескольких ЦОД и обрабатывать данные из одних и тех же источников и записывать в одну и ту же базу микросервиса, поэтому логично ввести таблицу с записью информации о запущенных экземплярах.
- «syncTab» для МОК и «syncTab42r2» - ... - «syncTab55r5» для МСО – таблицы синхронизации данных с хранилищем-источником или базами-источниками соответственно. Поскольку на стороне разрабатываемых микросервисов необходимо держать наиболее актуальные данные, то необходимо учитывать сколько, каким экземпляром приложения и до каких даты и времени были собраны

данные из источников. Таблицы решают проблему дубликации данных и помогают восстановить работу в случае отказа системы.

Для МОК специальными таблицами являются следующие.

- «downloadJournalView» – представление, охватывающее все поля таблиц основного содержания с данными из квитанций. Является единой точкой, откуда МСО забирает данные обработанных квитанций.
- «debugFileNames» и «debugStatuses» – таблицы для отладки и тестирования, хранящие список встреченных из S3-хранилища файлов и статус их обработки.

Для МСО можно выделить следующие специальные таблицы.

- «taskAudits», «taskTypes», «taskStatuses» – таблицы для хранения информации о запуске задачи, типе задачи и её статусе. МСО более технически сложен чем МОК и выполняет основной функционал сбора отчётов, поэтому необходимо предусмотреть сохранение всех деталей задачи, запущенной планировщиком или пользователем.
- «debugNotFoundSins» – таблица для отладки и тестирования, хранящая информацию по идентификаторам контрагентов, отсутствующих в справочной информации.

Вышеописанную специфику моделей вводимых баз данных стоит учитывать и использовать в процессе работы над системой.

4. Разработка

В данной главе описываются детали реализации системы, разделённые на серверную и клиентскую части. Описывать каждый разработанный элемент, из которого состоит работоспособность системы – излишне. Поэтому в пунктах этой главы описываются только наиболее значимые практические решения. В пунктах серверной части – это реализация автоматизации, а именно принципы работы планировщика, системы приоритета и выбора лидера, без которых система не выполняла бы своё основное предназначение. Также в серверной части приводятся и параметры сценариев ручного вмешательства в работу системы. В то же время, в пунктах клиентской части – описывается создание страниц пользовательского интерфейса с примерами их работы.

4.1. Серверная часть

Автоматическая часть

Основной задачей разрабатываемой системы является фоновое формирование отчётов без необходимости взаимодействия с человеком. Создать ряд задач, которые смогут выполняться по расписанию и обрабатывать данные – несложная задача. Однако, в случаях автономной работы достаточно продолжительное время – могут случиться внештатные ситуации, такие как: обработка достаточно большого файла, отключение некоторых ЦОД, задержка в выполнении задачи и многие другие, к которым основная логика работы не будет готова. Поэтому стоит уделить особое внимание именно проектированию автоматизированной части системы, которая будет примерно одинакова для каждого микросервиса.

Планировщик. Для того, чтобы наше решение было базировалось на автоматизации не требовало присутствия человека для выполнения основной работы – было принято реализовать планировщик, запускающий по расписанию

набор задач, а именно: в МОК планировщик запускает задачи загрузки и парсинга квитанций в БД, в МСО планировщик запускает задачи загрузки данных из таблиц-источников и задачи сбора XBRL-отчетов. Также планировщик имеет собственную задачу, проверяющую актуальное расписание задач, хранящееся в таблице «cronSchedules». Расписание задаётся CRON¹³-строкой. При первом запуске – значением по умолчанию, а далее меняется пользователем по API¹⁴. Более детальную схему работы автоматизации на этапе планировщика можно на рисунке 10.

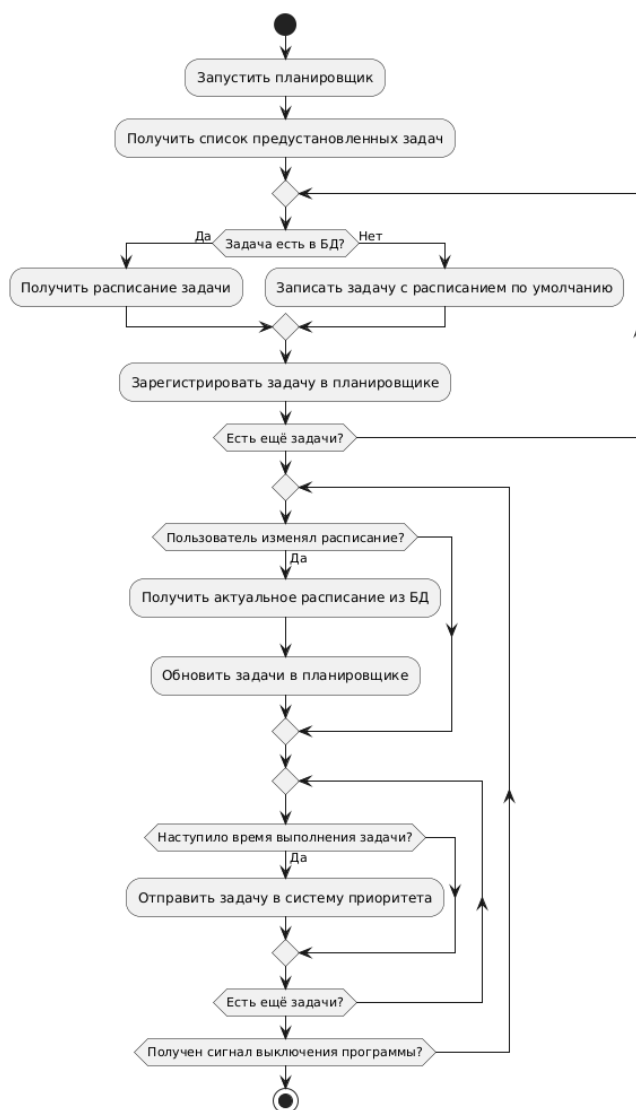


Рисунок 10 – Сценарий работы планировщика

¹³ CRON – планировщик задач в Unix и Linux системах.

¹⁴ API – средство взаимодействия с сервисом.

Пример хранения расписания представлен на рисунке 11.

	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 11 – Таблица хранения расписания задач планировщика

Система приоритета. Задача блока с планировщиком – запустить задачи по расписанию. Однако, в случаях, когда набор задач создан параллельно – не все задачи могут быть параллельно выполнены. Например: в таблице-источнике для данных отчёта 0420754 раздел 4 содержатся также и справочные данные для других отчётов, которые будут собираться в других параллельных задачи, а значит эта задача должна выполняться раньше остальных. Таким образом, был написан модуль, который называется «Система приоритета» и который принимает на вход параллельно запущенные планировщиком задачи для группировки их на наборы уровней для параллельного выполнения. Приоритет уровней 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 формирования XBRL-отчётов по основным отчётам и разделам.
4. Задачи формирования XBRL-отчётов, являющихся справочниками к собранным отчетам.

Так же стоит упомянуть, что в системе приоритета реализована очередь, которая следит за выполнением каждой задачи для того, чтобы не допустить выполнения нового экземпляра задачи, пока подобная ещё выполняется. Детальное отображение логики работы этого модуля представлено на рисунке 12.

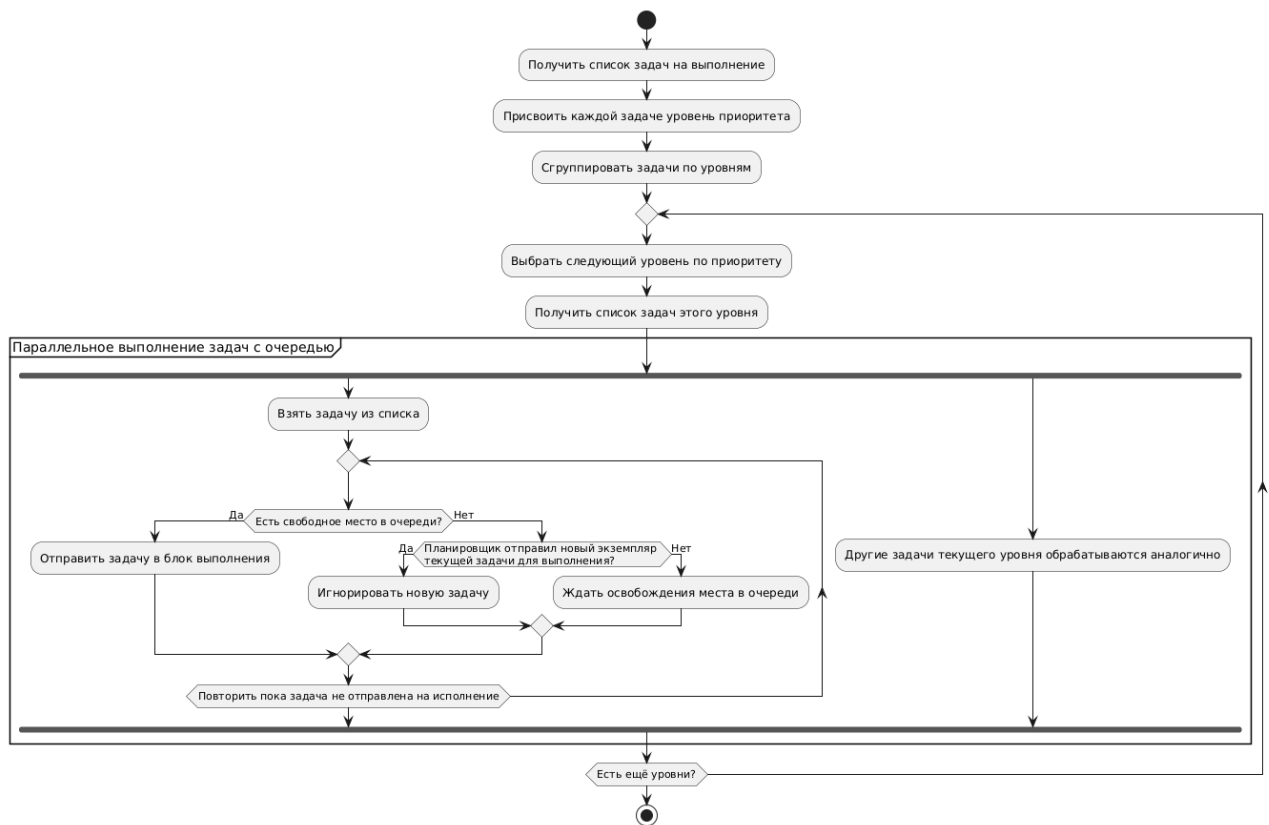


Рисунок 12 – Сценарий работы системы приоритета

Блок выполнения задач. Одно из требований, установленное организацией, является поддержка развёртки одного микросервиса на нескольких подах «Kubernetes», подключаемым к одним и тем же базам данных. Другое требование – соблюдать целостность и актуальность загружаемых данных. Отсюда вытекает несколько нюансов выполнения задач. В случае запуска одного единственного экземпляра приложения – синхронизация по загружаемым из таблиц-источников данным тривиальна. Но при одновременном включении множества экземпляров системы можно получить дублирование данных и некорректную сборку. Для решения этой проблемы был написан модуль выбора лидера. Он не является зависимостью блока выполнения задач, но используется в нём.

Блок выполнения задач работает с учётом того, что обрабатывать данные из одних и тех же источников и отправлять результаты в одну и ту же целевую таблицу могут сразу несколько экземпляров одного приложения.

Соответственно была придумана гибкая логика для этого процесса: экземпляр приложения получает из таблицы “taskAudits” левую границу доступного ему диапазона, делает выборку из источника, отправляет получившийся свой диапазон в таблицу и приступает к обработке данных. Таким образом, любой следующий экземпляр, готовый к загрузке данных, будет работать с собственным диапазоном данных. Детальная реализация логики работы этого блока представлена на рисунке 13.

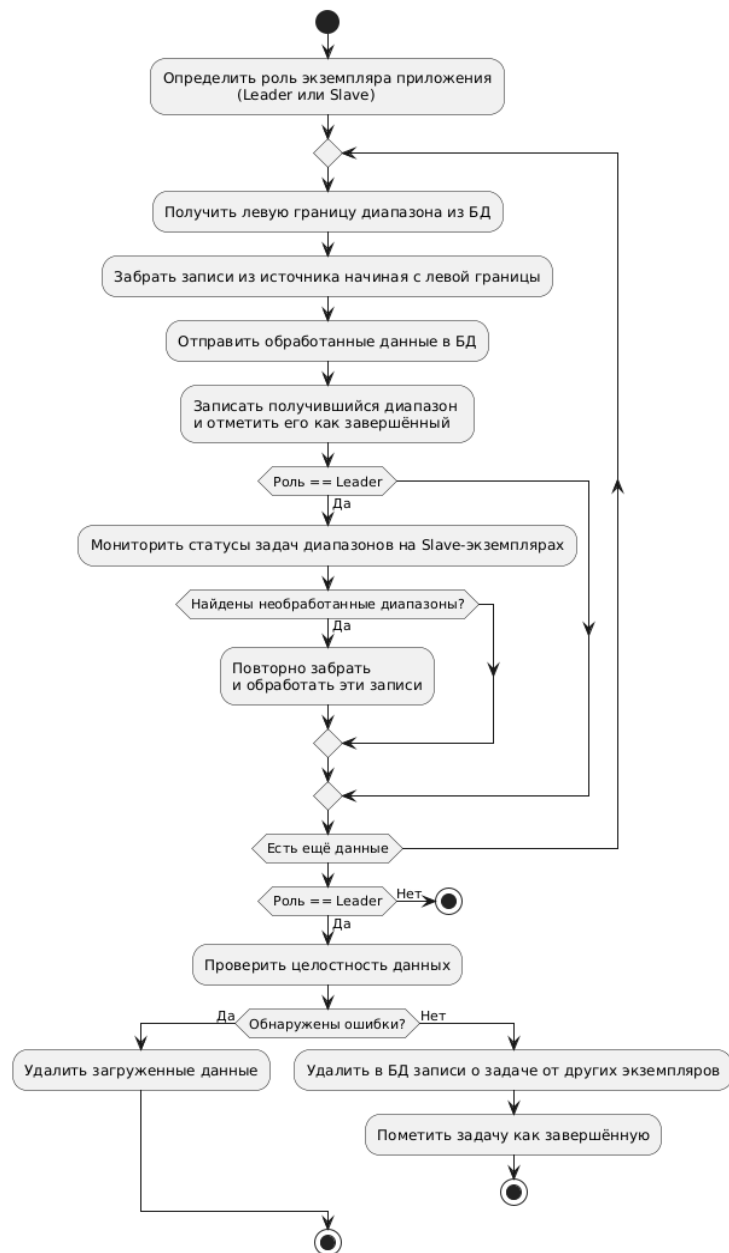


Рисунок 13 – Сценарий выполнения задачи

Выбор лидера. Как было изложено выше – микросервисы системы могут быть развёрнуты на нескольких подах, но при этом должны обеспечивать целостность данных при загрузке из источников. Это привело к созданию модуля выбора лидера среди экземпляров приложения. Алгоритм работает примерно следующим образом.

1. При включении приложения – приложение регистрируется в таблице синхронизации.
2. Если это единственное работающее на данный момент приложение – оно становится leader-приложением и соответствующим образом помечается в таблице, иначе – становится slave-приложением.
3. Каждое приложение при выполнении задач в таблице синхронизации отмечает диапазон данных с которыми будет работать.
4. Если приложение выключилось или завершилось из-за ошибки – оно остаётся записанным в таблице со статусом «in progress», а работу над его диапазоном данных будет проводить leader-приложение. Если упавшее приложение – это leader-приложение, то первое успешшее запросить права приложение – становится leader-приложением.
5. Leader-приложение также отвечает и за проверку успешной завершённости работы над диапазонами данных других экземпляров приложения.

Таким образом, система может быть запущена одновременно на нескольких машинах, на нескольких подах «Kubernetes» и при этом обеспечивать целостность загружаемых данных.

Ручная часть

Система всегда работает преимущественно автономно от человека, однако пользователь должен иметь возможность провести ряд манипуляций над данными. Например, сформировать отчёты за иной временной диапазон, или

полностью пересобрать таблицу с загруженными данными в критической ситуации. Общение с пользователем реализовано через клиентскую часть приложения, которая взаимодействует с backend-приложением через HTTP-запросы. Для понимания ручной работы с системой стоит детально рассмотреть некоторые запросы, а именно POST- и GET-запросы к MCO.

1. 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
}
```

Логика работы метода представлена блок-схемой на рисунке 14.

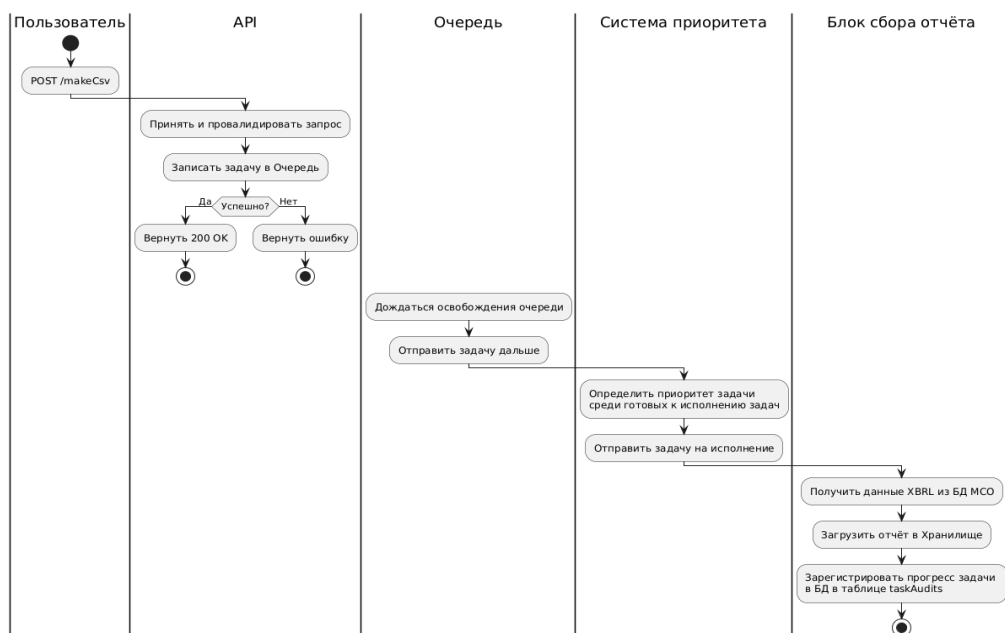


Рисунок 14 – Сценарий работы метода запуска формирования отчёта

2. GET /getCsv – метод получения статуса задачи формирования отчёта и URL-ссылки 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",
    "...",
  ]
}
```

Логика работы метода представлена блок-схемой на рисунке 15.

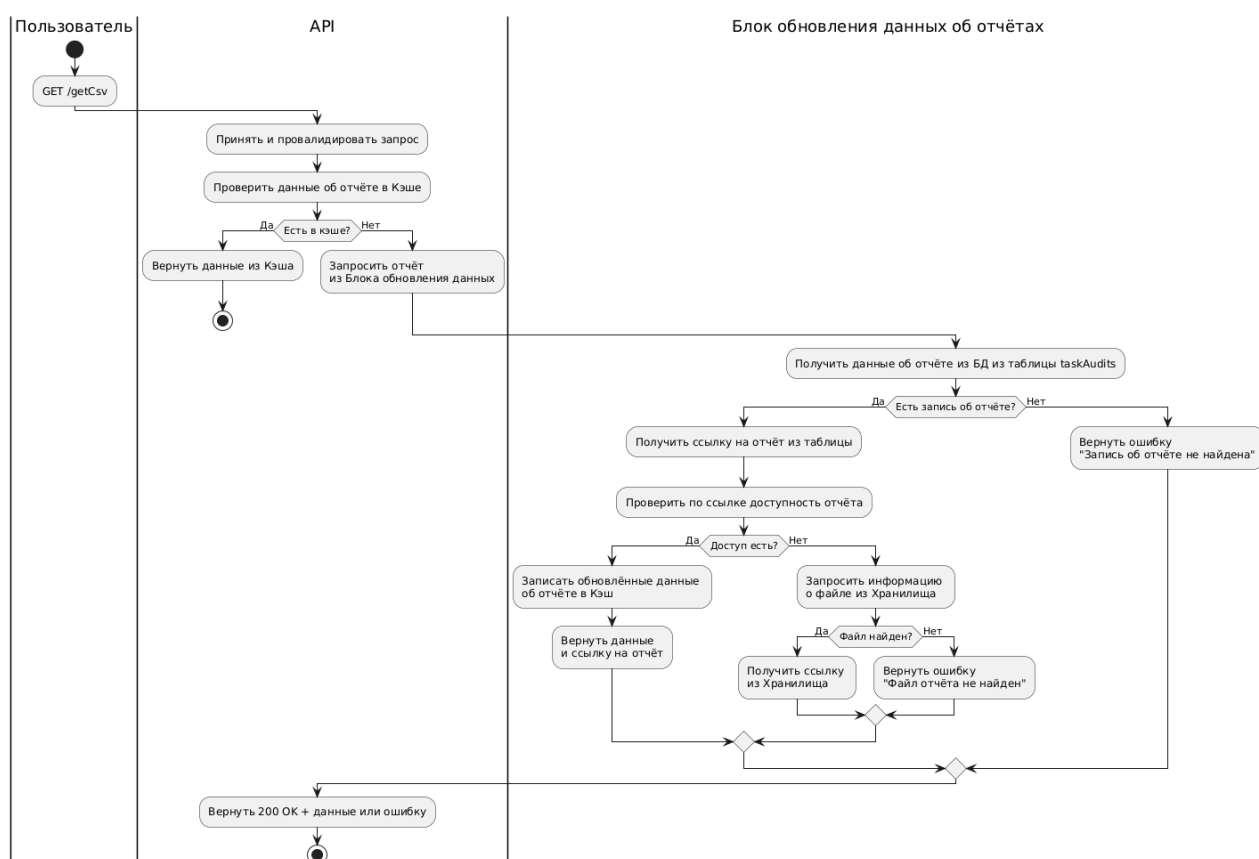


Рисунок 15 – Сценарий получения информации о готовом отчёте

3. POST /reMap – метод очищения таблицы с данными и запуска загрузки данных из источников заново.

Формат запроса – JSON:

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

Формат ответа – JSON:

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

Логика работы метода представлена блок-схемой на рисунке 16.

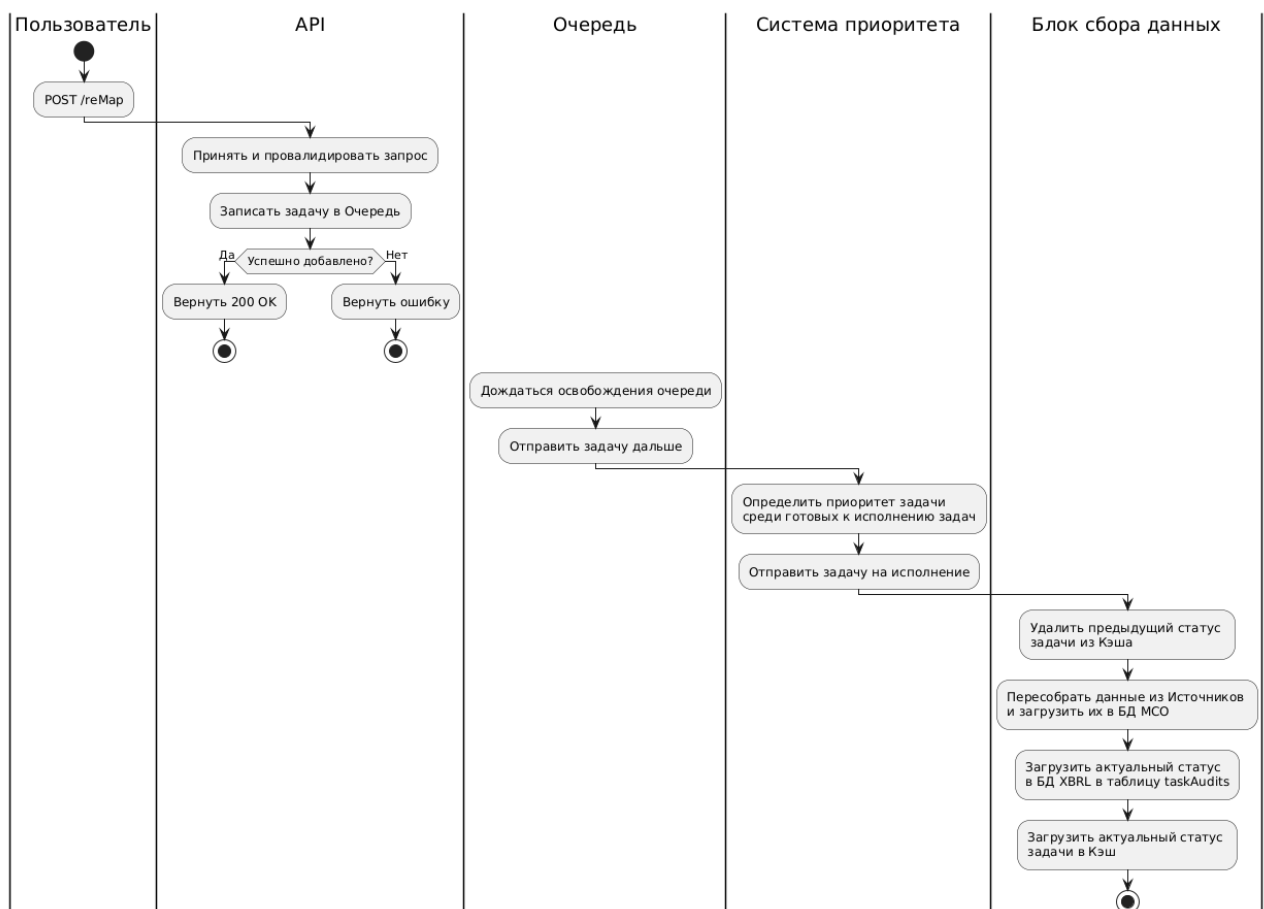


Рисунок 16 – Сценарий работы метода запуска перезагрузки данных

4. GET /mapStatus – метод получения статуса задачи загрузки данных.

Формат запроса – аргументы запроса в адресной строке:

/mapStatus?tableName=form0420754r3

Формат ответа – JSON:

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

Логика работы метода представлена блок-схемой на рисунке 17.

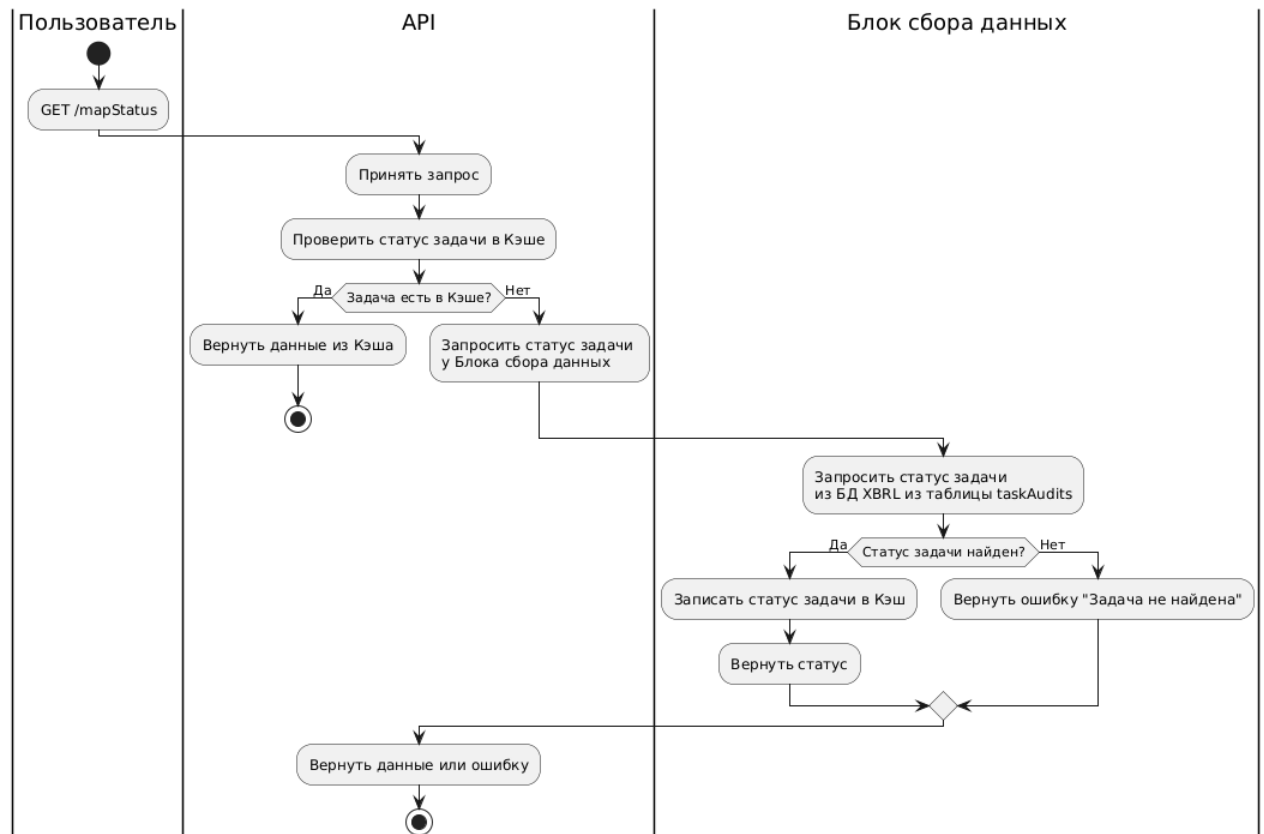


Рисунок 17 – Сценарий получения информации о статусе перезагрузки данных

В системе также реализованы и иные методы общения с backend-сервером, но их логика работы более проста и не нуждается в детальном разборе. В MCO описаны следующие HTTP-запросы.

- 1) /changeSchedule – изменение расписания конкретной задачи.
- 2) /getAllCsv – получение списка всех отчётов с пагинацией.
- 3) /getNoticesFromS3 – получение списка всех квитанций из хранилища данных с указанием типа квитанции и пагинацией.
- 4) /getNoticeFromS3 – получение конкретной квитанции из хранилища данных по имени файла.

5) /getRecords – получение записей собранных из источников с указанием таблицы, относящейся к отчёту, и пагинацией.

В МОК представлены следующие http-запросы.

- 1) /changeSchedule – изменение расписания конкретной задачи, аналогично поведению в МСО.
- 2) /fillDbByFirstNotice – загрузка конкретной квитанции первого типа в БД МСО.
- 3) /fillDbBySecondNotice – загрузка конкретной квитанции второго типа в БД МСО.
- 4) /fillDbByAlarm – загрузка конкретной квитанции третьего типа в БД МСО.
- 5) /listDownloadedNotices – получение информации об обработанных из Хранилища данных квитанций.

Сборка микросервисов

Процесс развёртки микросервисов на тестовом и рабочем контурах находится в зоне ответственности команды инфраструктуры организации. Предполагается, что приложение должно будет поднято на подах “Kubernetes” на базе образа “Linux Alpine”. Однако, для того, чтобы передать сервисы на публикацию и подключение ко всем компонентам – необходимо каким-либо образом их собрать. А также, поскольку были введены в эксплуатацию две базы данных, то нужно предусмотреть процесс применения актуальных миграций.

Подготовка приложения к эксплуатации будет заключаться в сборе docker-образа с помощью Dockerfile. Dockerfile микросервиса обработки квитанций не подразумевает сложных манипуляций и представлен с комментариями в приложении 1. Но Dockerfile микросервиса сбора отчётов, который представлен в приложении 2, немного усложнён необходимостью установки утилиты «Oracle Instant Client» [20], которая позволяет подключать приложение к базе данных «Oracle». Также из-за введённых баз данных были

написаны два аналогичных файла для утилиты «Goose», применяющей миграции в проект. Листинг такого файла представлен в приложении 3.

4.2. Клиентская часть

Необходимость в создании пользовательского интерфейса к разрабатываемой системе происходит из процесса первых запусков в реальных условиях. Проверка корректности и целостности загруженных данных усложняется такими проблемами как: согласование доступов в базы данных для отдельных проверяющих пользователей, лишние действия при просмотре квитанций в хранилище данных, необходимость вручную описывать каждый GET- и POST-запрос для отправки его в бэкенд и т.д. Исходя из этого появилась потребность в скором написании небольшого десктопного приложения, работающее только внутри локальной сети компании и доступ к которому будет только у пользователей, проверяющих корректность отчётов. Приложение должно выводить информацию о квитанциях до и после загрузки, отображать собранные и агрегированные записи в базе и готовые отчёты, а также иметь возможности создавать задачи на сбор отчётов и перезагрузку данных в таблицах. Доступ к источникам производится через подключение к Vault [21] хранилищу данных.

Реализация пользовательского интерфейса

Стоит привести страницы, которые должны быть внутри клиентского приложения.

- «Просмотр готовых отчетов» – функционал возможности получить ссылки на скачивание всех отчётов, которые собирались приложением.

- «Просмотр всех квитанций» – получение на просмотр всех доступных квитанций из «Сeph S3».
- «Статус обработки квитанций» – все встреченные приложением квитанции и иные файлы из «Сeph S3».
- «Просмотр таблиц» – просмотр записей в таблицах, на основании которых формируются отчёты.
- «Собрать отчёт» – функционал сбора отчёта.
- «Пересобрать таблицу» – функционал перезагрузки основных таблиц.

Стартовая страница приложения с данными пунктами меню выглядит следующим образом.

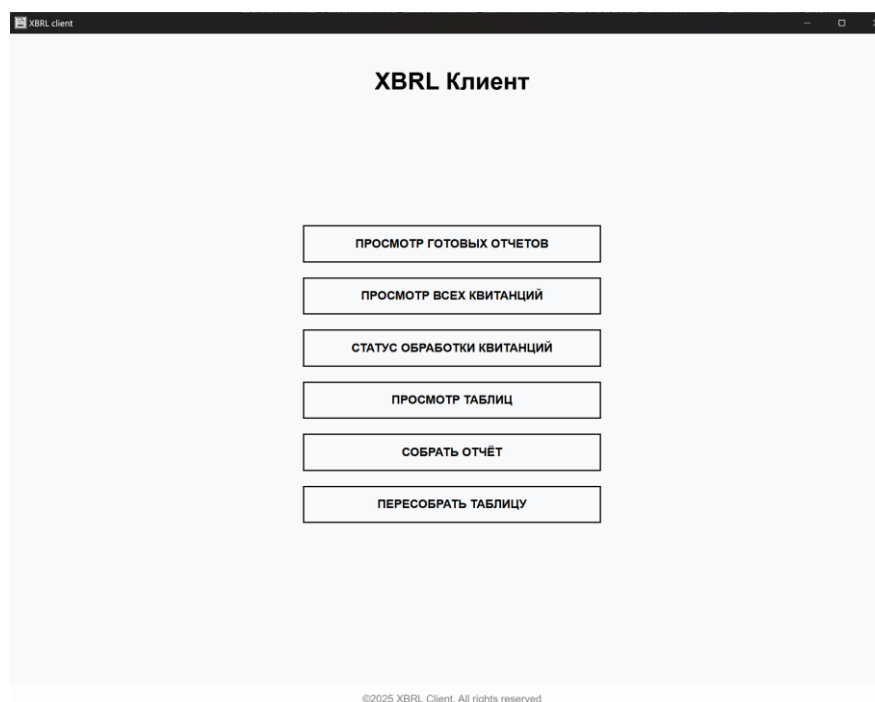


Рисунок 18 – Стартовая страница пользовательского интерфейса

На странице «Просмотр готовых отчетов», рисунок 19, представлен список ссылок на конкретные готовые отчёты, при нажатии на отчёт предлагается сохранить его на компьютер.

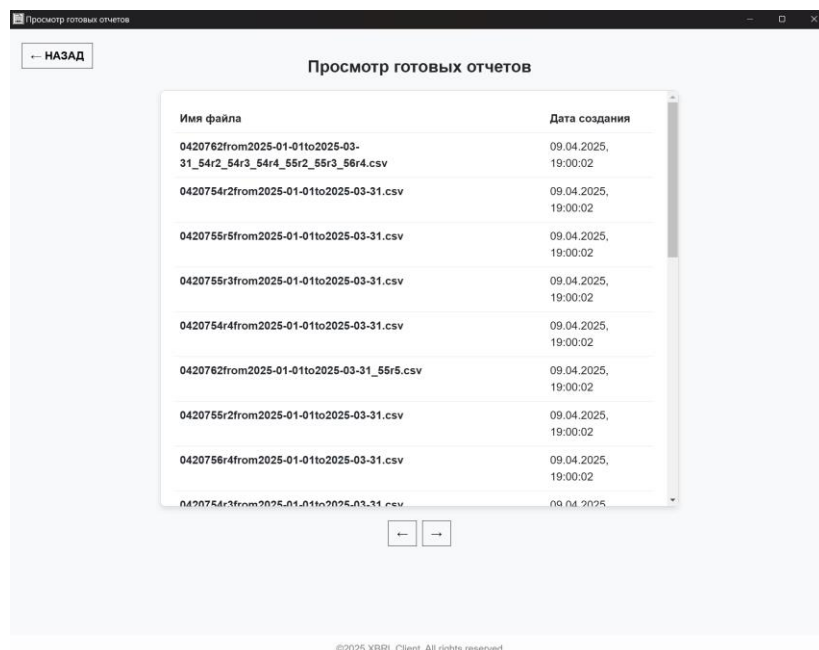


Рисунок 19 – Страница «Просмотр готовых отчетов»

На странице «Просмотр всех квитанций», отображённой на рисунке 20, представлен функционал доступа к 3-ём разным типам квитанций и возможность просмотреть их контент, как на рисунке 21.

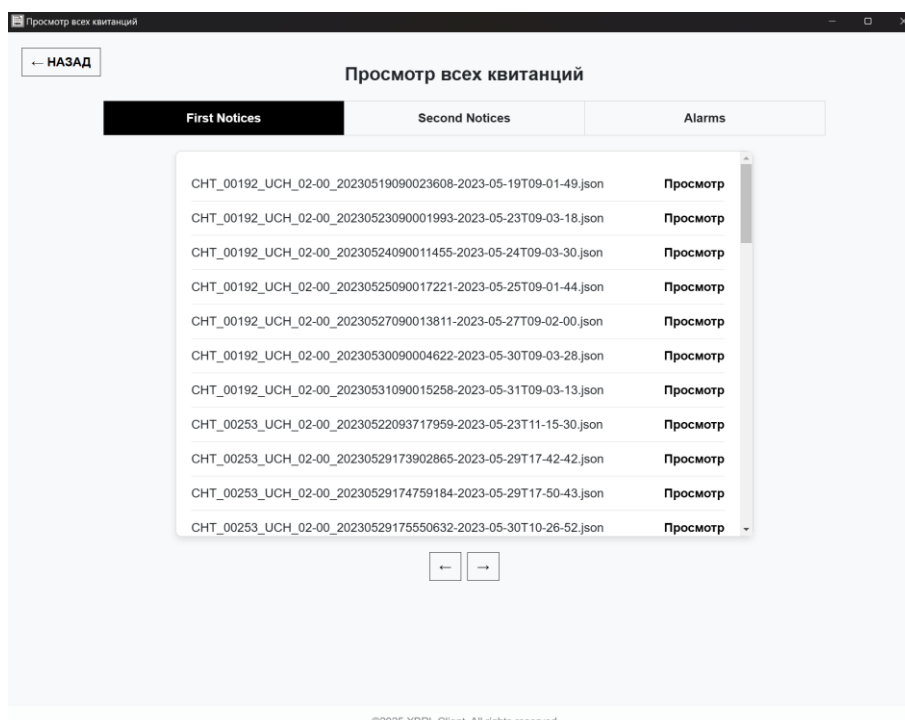


Рисунок 20 – Страница «Просмотр всех квитанций»

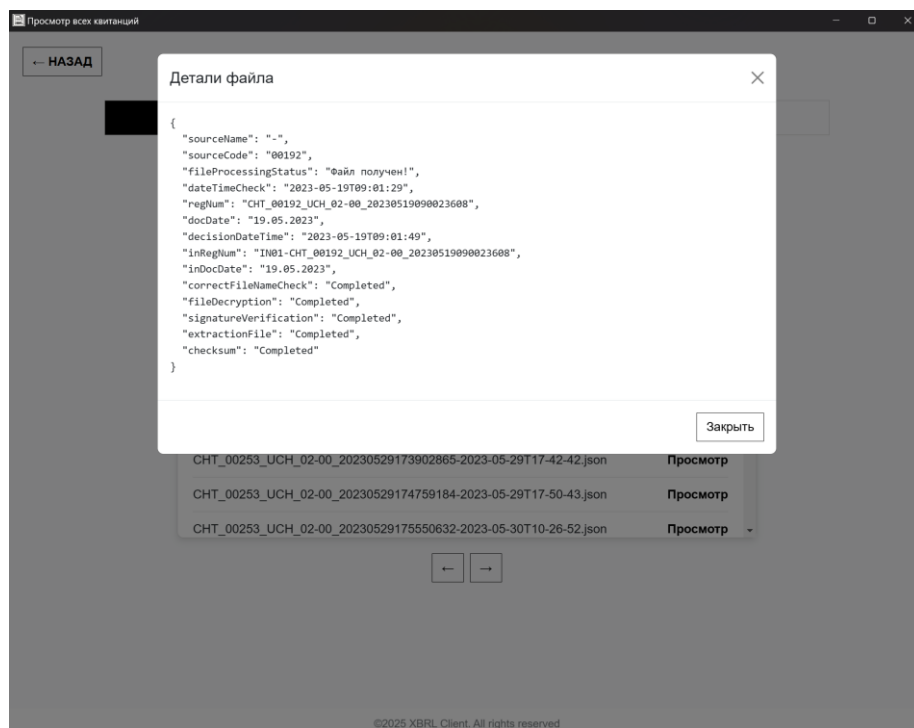


Рисунок 21 – Просмотр контента квитанции

На странице «Статус обработки квитанций», показанной на рисунке 22, располагается таблица с информацией о каждом файле: имя, статус загрузки, размер в байтах. Также на ней реализован поиск по интересующему названию файла, рисунок 23.

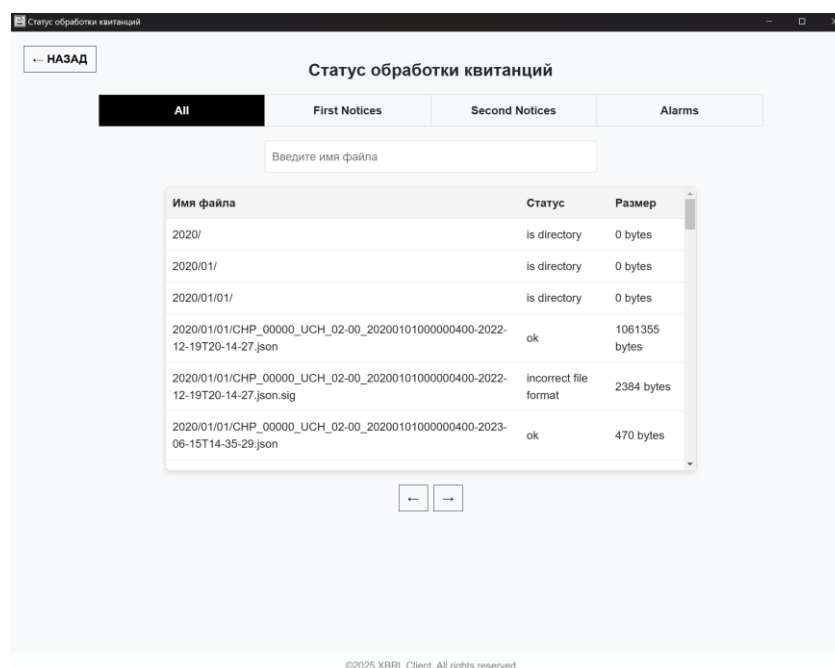


Рисунок 22 – Страница «Статус обработки квитанций»

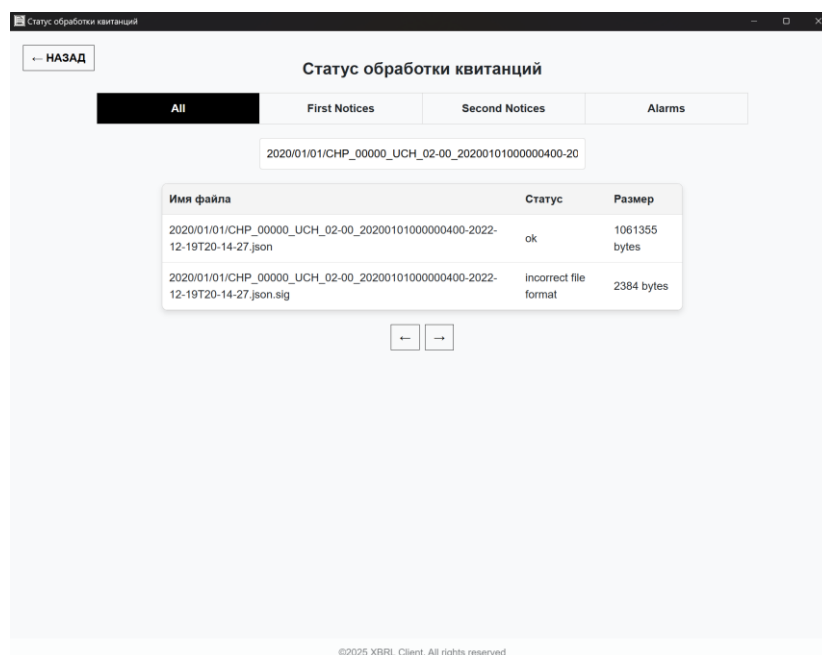


Рисунок 23 – Страница «Поиск по квитанциям»

На странице «Просмотр таблиц» располагаются на выбор все таблицы, которые относятся к сборке отчёта, рисунок 24. Здесь реализован поиск по разным типам контрагентов и фильтрация по старым и новым записям.

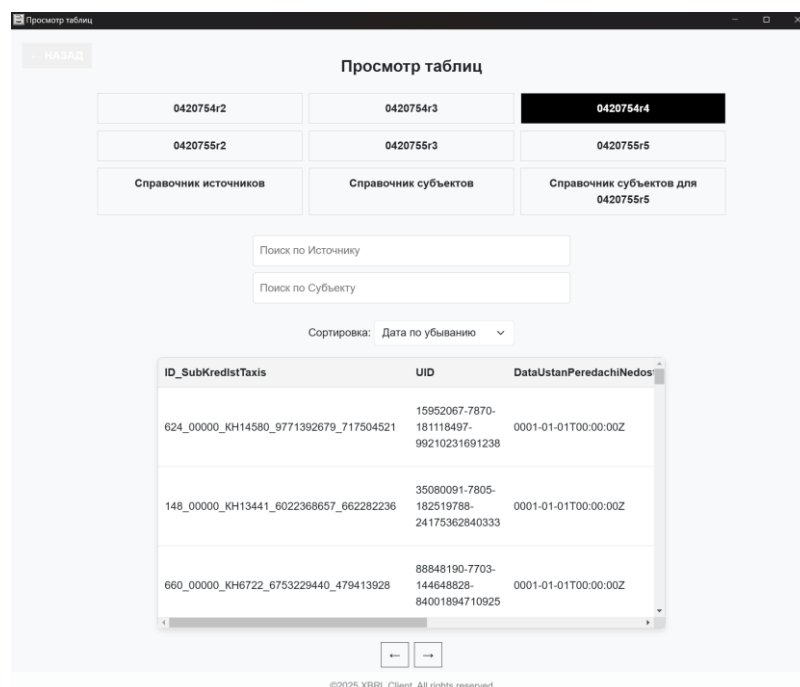


Рисунок 24 – Страница «Просмотр таблиц»

На странице «Собрать отчёт» представлен стандартный функционал сбора отчёта через вышеописанный запрос «/makeCsv», рисунок 25. На рисунке 26 представлен результат выполнения запроса: вывод ссылок на готовые отчёты (3 выбранных отчёта и справочник к ним).

Собрать отчёт

0420754r2

0420754r3

0420754r4

0420755r2

0420755r3

0420755r5

01.01.2025 09.04.2025

ПОДТВЕРДИТЬ

©2025 XBRL Client. All rights reserved

Рисунок 25 – Страница «Собрать отчёт»

Собрать отчёт

0420754r2

0420754r3

0420754r4

0420755r2

0420755r3

0420755r5

01.01.2025 09.04.2025

ПОДТВЕРДИТЬ

1. Скачать файл
2. Скачать файл
3. Скачать файл
4. Скачать файл

©2025 XBRL Client. All rights reserved

Рисунок 26 – Результат сбора отчётов.

На странице «Пересобрать таблицу» располагается возможность выбрать одну из таблиц отчётов, вывести статус загрузки данных в неё и запустить перезагрузку, рисунок 27.

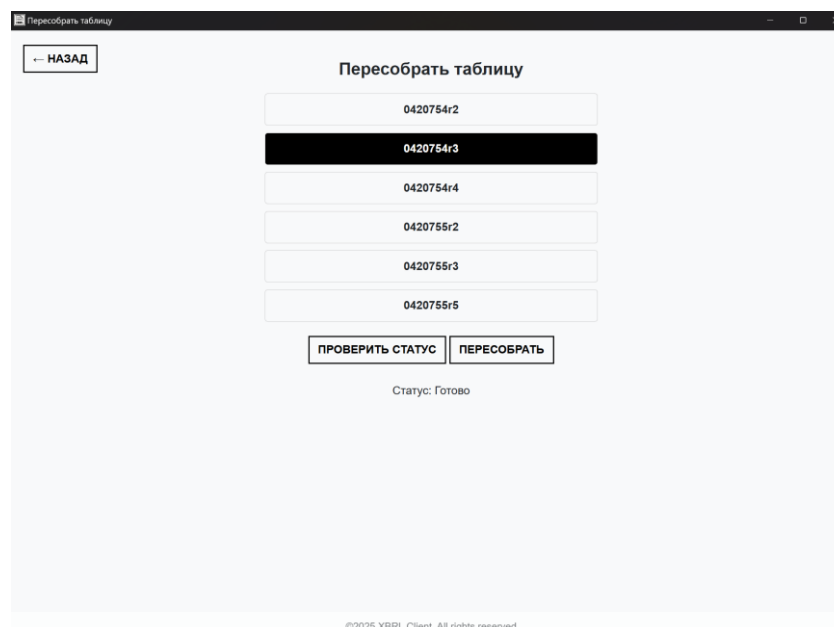


Рисунок 27 – Результат сбора отчётов.

Готовое приложение можно собрать через утилиту «electron/packager» [26], указав любую настольную ОС.

```
npx @electron/packager <sourcedir> xbrl_client --platform=<platform> --arch=<arch>
```

Таким образом, было реализовано небольшое и эффективное клиентское приложение с пользовательским интерфейсом, что поможет проверяющим отчёты сотрудникам ускорить процесс своей работы.

5. Тестирование

5.1. Нагрузочное тестирование

Проведение нагрузочного тестирования МОК. Как упоминалось ранее, у МОК предусмотрены POST-запросы для загрузки и обработки конкретных файлов в базу. Например, для анализа ошибки загрузки файла или загрузки в базу пропущенного файла. Несмотря на то, что при стандартной эксплуатации сервиса эти запросы будут использоваться единично и крайне редко – стоит определить пиковые возможности микросервиса.

Запросы, которые будут протестированы («fillDbByFirstNotice», «fillDbBySecondNotice», «fillDbByAlarm») – не имеют сложной логики: каждый из них получает в теле запроса параметр «jsonFileName» – наименование JSON-файла, скачивает этот файл из «Ceph S3», парсит согласно внутренней логике и вставляет записи в нужные таблицы в БД. Поэтому эти запросы избегают сложных оптимизаций.

Максимальное количество потенциальных пользователей, которые в момент разбора внештатной ситуации могут начать пользоваться МОК, ограничивается числом менее 20 человек. Это примерная численность сотрудников отдела разработки и отдела бизнес-заказчиков. Будем использовать худший сценарий, когда каждый пользователь делает запрос каждую секунду на каждую ручку, что в реальности практически невозможно. Соответственно, преодоление порога в 60 RPS будет свидетельствовать о высоконагруженности МОК.

Далее разберём код нагрузочного сценария, листинг которого представлен в приложении 1. В нём есть основной сценарий плавного поэтапного увеличения RPS до заданного значения.

```
firstNoticesRequests: {  
    executor: 'ramping-vus',  
    startVUs: 1,  
    stages: [  
        { duration: '10s', target: 10 },  
        { duration: '30s', target: 50 },  
    ]  
}
```

```

        { duration: '30s', target: 100 },
        { duration: '18m', target: 100 },
        { duration: '20s', target: 0 },
    ],
    exec: 'firstNoticesRequests',
}

```

Также в коде есть 3 функции, каждая из которых описывает логику запроса к МОК и проверку статуса его ответа. Каждая из них обращается к банку данных наименований квитанций, берёт случайное название файла и отправляет его к бэкенду. Файлы для проведения тестирования выбраны так, чтобы МОК отвечал на них статусом 200 ОК и не искажалась результирующая статистика, т.к. проверка идёт на нагрузку, а не на проверку парсинга. Всего файлов квитанций первого типа: 373 штуки, файлов квитанций второго типа 9646 штук, файлов квитанций третьего типа: 11729 штук, этого достаточно для рандомизации запросов, все они разного имеют разный объём. Предварительно база данных МОК очищена и готова к записи квитанций. Выбран целевой RPS в размере 300 req/s, по 100 req/s на каждую ручку, что значительно превышает запланированный порог. Рассмотрим результаты двадцатиминутного нагрузочного теста, который по итоговому проценту можно считать успешным.

Rates		Gauges	
metric	rate	metric	value
checks	1/s	vus	11
http_req_failed	0/s	vus_max	300

Рисунок 28 – Второй запуск нагрузки МОК

Также стоит проанализировать график RPS, «Response Time» и «Failed requests» отображённый на рисунке 29. На нём видно, что со временем на 300 req/s сервис начинает «тормозить» во времени ответа, RPS падает в среднем до 60-100 req/s, а время ответа до 10 секунд.

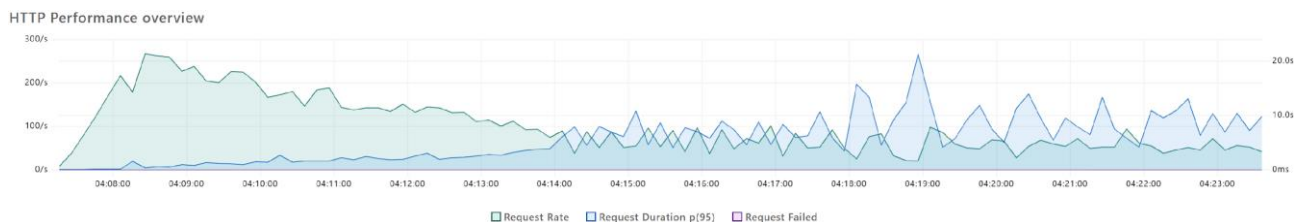


Рисунок 29 – График RPS, «Response Time» и «Failed requests» МОК

Этот тест показывает, что если снизить нагрузку до 100 req/s, то МОК способен на одном поде это значение RPS, имея приемлемое время ответа. Поэтому порог в заявленные 60 RPS можно считать пройденным. Однако для обеспечения меньшей нагрузки на инстанс приложения и понижения времени ответа до минимального – стоит добавить еще 2-3 пода и выделить столько же ядер процессора, при учёте доступности этих ресурсов.

Проведение нагрузочного тестирования МСО. Микросервис сбора отчёта также создан преимущественно для автоматизированной работы, где взаимодействие с пользователем должно быть сведено к минимуму. Однако МСО имеет более широкий функционал и запросы на генерацию отчётов используются не редко. Поскольку у нас есть два ключевых POST-запроса: «/makeCsv» - сбор отчёта и «/reMar» - загрузка из источников, то была произведена оптимизация логики их работы под высокую нагрузку. Она заключалась в том, чтобы запросы не собирали отчёты напрямую, а создавали задачи на сбор отчётов, помещая их в общую очередь. Такой подход позволил избежать долгого ожидания ответа от сервера в случае работы системы с большим количеством данных. Также для GET-запросов была придумана система кэширования, описанная в реализации ручной части. Она позволяет возвращать результаты завершённых задач, без необходимости нагружать базу данных.

Данный микросервис имеет потенциал встраивания в развивающуюся платформу организации, поэтому потенциальное количество одновременных пользователей при худшем сценарии – это все сотрудники организации со стороны бизнес-заказчиков. Их примерное количество – около 300 человек,

соответственно при худшем сценарии, когда каждый сотрудник делает запрос раз в секунду – сервис должен выдерживать 300RPS. Поскольку GET-запросы на просмотр готовых отчётов или таблиц происходят чаще чем POST-запросы на их сбор, то примем, что POST-запросы на загрузку будут составлять 25% относительно числа GET-запросов на просмотр. Это один из неблагоприятных сценариев, поскольку при штатной эксплуатации приложение должно собирать отчёты раз в квартал (7 POST-запросов в 4 месяца), а запрашивать их гораздо чаще.

Далее разберём код нагрузочного сценария, листинг которого представлен в приложении 2. В нём есть основной сценарий плавного поэтапного увеличения RPS до заданного значения для GET- и POST-запросов.

```
getCsv: {
  executor: 'ramping-vus',
  startVUs: 1,
  stages: [
    { duration: '10s', target: 10 },
    { duration: '30s', target: 80 },
    { duration: '30s', target: 400 },
    { duration: '20m', target: 400 },
    { duration: '20s', target: 0 },
  ],
  exec: 'getCsv',
},
makeCsv: {
  executor: 'ramping-vus',
  startVUs: 1,
  stages: [
    { duration: '10s', target: 10 },
    { duration: '30s', target: 50 },
    { duration: '30s', target: 100 },
    { duration: '20m', target: 100 },
    { duration: '20s', target: 0 },
  ],
  exec: 'makeCsv',
},
```

Аналогично нагрузочному тесту для МОК в коде данного скрипта также есть функции для отправки запросов к бэкенду. Единственное допущение для данного нагрузочного теста будет заключаться в том, бэкенд не будет отдавать ошибки по запросам «/mapStatus» и «/getCsv», если данных о сборе отчёта или таблицы ещё не было загружено в базу. Но если они появились – поддерживается стандартная работа. Это происходит потому, что эти GET-запросы штатно идут

после выполнения соответствующих POST-запросов, а иное поведение портит статистику результата и не является показательным.

Как и в прошлом нагрузочном тесте, выбрано значение RPS сильно большее, чем целевое – 1000req/s, для проверки критических мощностей приложения. Как видно на рисунке 6 проблем и ошибок у сервиса не обнаружено. А вывод из рисунка 7 следует, что под нагрузкой в 1000req/s, приложение держит нагрузку стабильно, а показатель времени ответа располагается в пределах 1мс и 1сек, что является приемлемым значением.

Rates		Gauges	
metric	rate	metric	value
checks	1/s	vus	2
http_req_failed	0/s	vus_max	1k

Рисунок 30 – Результат нагрузки в 1000RPS на BM

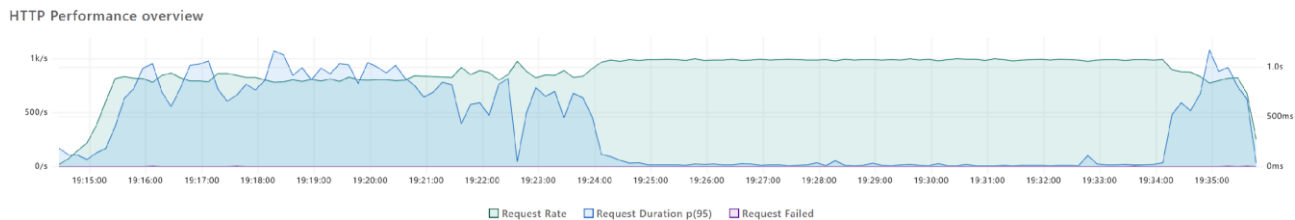


Рисунок 31 – График RPS, «Response Time», «Failed requests» для 1000 RPS на MCO

Поскольку под нагрузкой в 1000 req/s приложение работает довольно стабильно – было принято решение увеличить количество RPS. Однако, хоть тесты на 1600 req/s и 1300 req/s завершились успешно и без ошибок, но анализ графика RPS, «Response Time» и «Failed requests» показал, что со временем вырисовывается тренд на деградацию времени ответа, что видно на рисунках 8 и 9 для 1600 req/s и 1300 req/s соответственно.

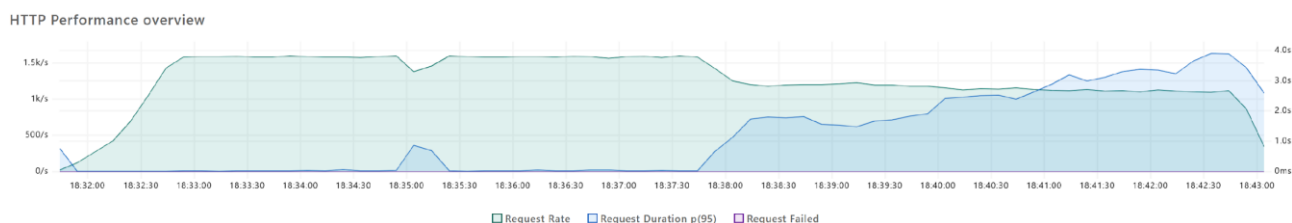


Рисунок 32 – График RPS, «Response Time», «Failed requests» для 1600 RPS на MCO

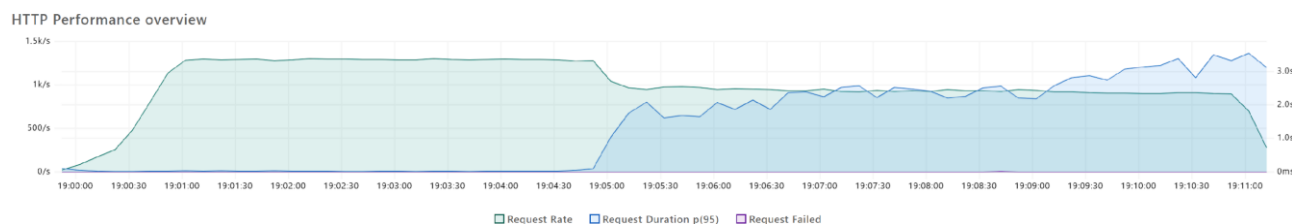


Рисунок 33 – График RPS, «Response Time», «Failed requests» для 1300 RPS на MCO

Следовательно можно сделать вывод о том, что оптимальное пиковое RPS, которое приложение может стабильно держать равно примерно 1000RPS. Однако добавление одного дополнительного пода и выделения ему отдельного ядра процессора – снизит время ответа и просадки сервиса. Таким образом, можно сказать, что оба микросервиса готовы к высоконагруженности и выдерживают бóльшую нагрузку, чем требуется.

5.2. Апробация системы

Для проверки того, что система выполняет свой основной функционал, необходимо провести ряд тестовых сценариев, которые показывают каким образом и что происходит в системе. Разобраны следующие главные тестовые сценарии.

1. Тест «из квитанций в отчёты»

Загрузка по одной квитанций каждого типа в хранилище данных.

out-file-invoice-load-uch	1	1.4 KiB
out-preprocessing-invoice-ch	1	544.0 B
out-report-uch	1	1.8 KiB

Рисунок 34 – Количество записей квитанций в хранилище данных

Ожидание обработки квитанции средствами МОК.

```
[2025-05-25T23:15:00+03:00][DEBUG][Service Layer]: Insert pack of SecondNotices;
[2025-05-25T23:15:00+03:00][DEBUG][Service Layer]: Insert pack of Alarms;
[2025-05-25T23:15:00+03:00][DEBUG][Service Layer]: Insert pack of FirstNotices;
[2025-05-25T23:15:00+03:00][DEBUG][Service Layer]: AlarmsTask successful;
[2025-05-25T23:15:00+03:00][DEBUG][Service Layer]: SecondNoticeTask successful;
[2025-05-25T23:15:00+03:00][DEBUG][Service Layer]: FirstNoticeTask successful;
```

Рисунок 35 – Лог работы МОК по загрузке квитанций

Просмотр информации о квитанциях средствами МПИ.

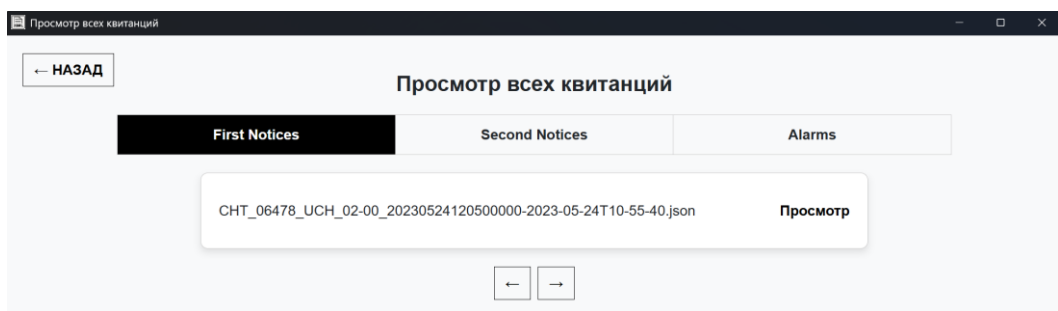


Рисунок 36 – Просмотр обработанной квитанции первого типа

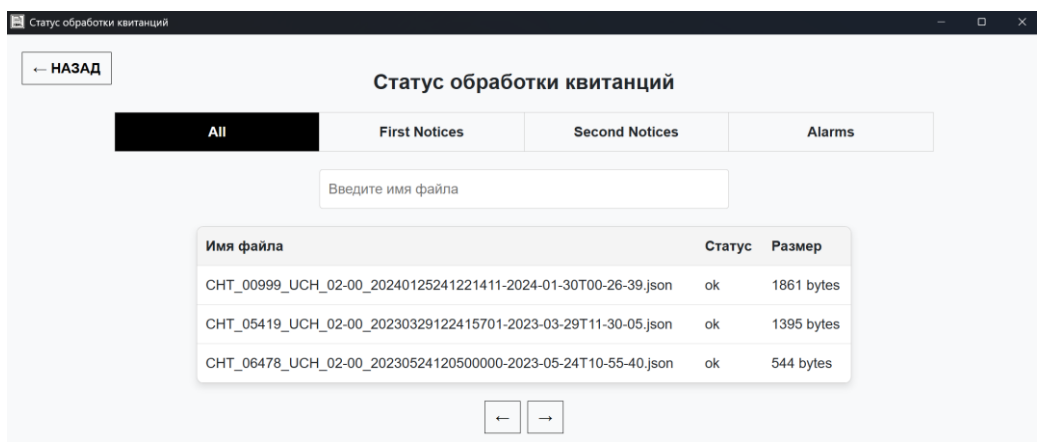


Рисунок 37 – Просмотр статуса встреченных квитанций

Добавление ещё по 1 квитанции каждого типа в хранилище данных.



 out-file-invoice-load-uch	2	2.7 KiB
 out-preprocessing-invoice-ch	2	1.1 KiB
 out-report-uch	2	2.3 KiB

Рисунок 38 – Количество записей квитанций в хранилище данных после добавления новых

Видно, что планировщик запустил новую задачу и обработал новые квитанции.

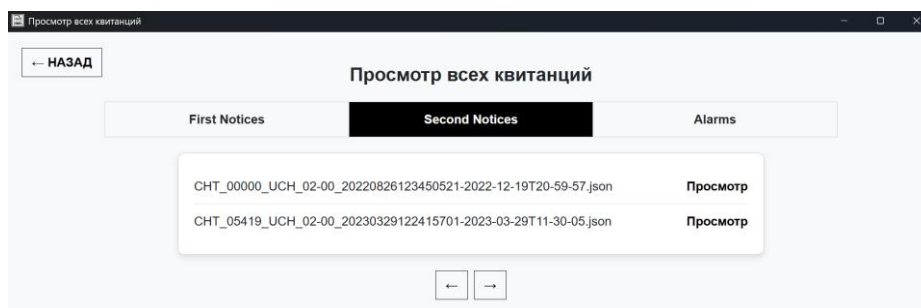


Рисунок 39 – Просмотр обработанных квитанций второго типа

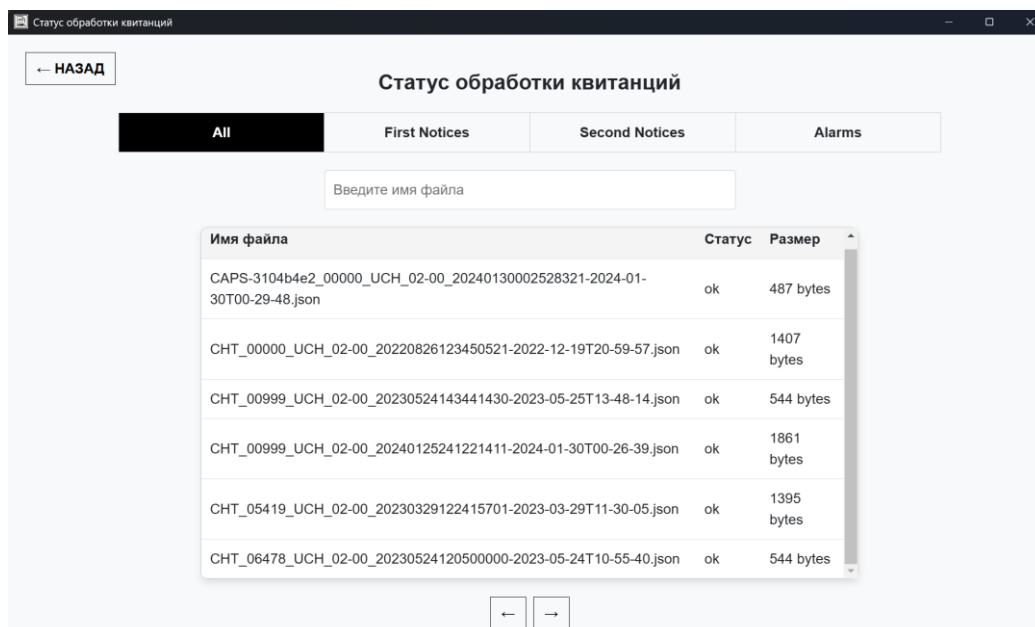


Рисунок 40 – Просмотр статуса встреченных квитанций после добавления

После обработки можно создать запрос на генерацию отчёта «0420754r2», собираемого из данных квитанций.

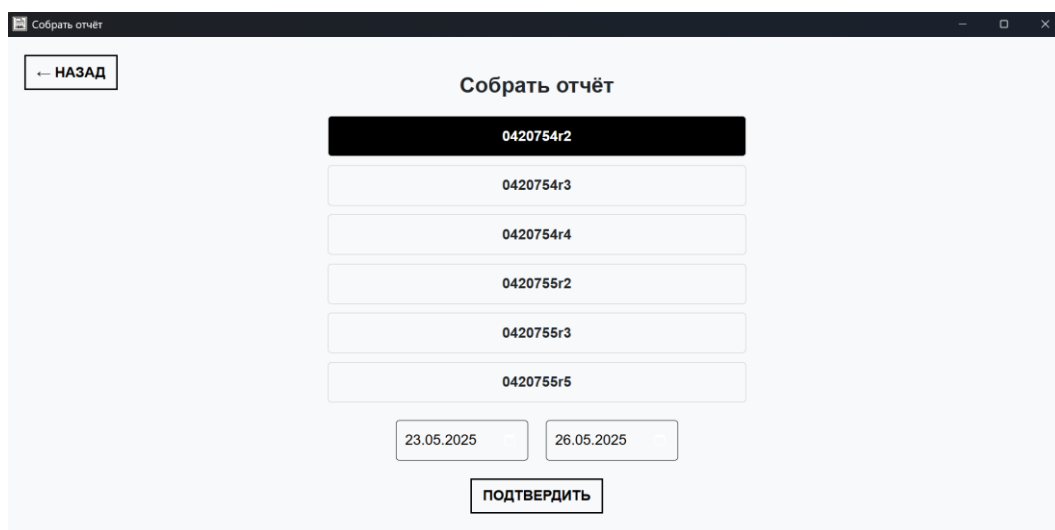


Рисунок 41 – Запрос на генерацию отчёта «0420754r2»

После скачивания файла с отчётом можно увидеть, что после подготовки данных прошла проверку только одна квитанция.

	A
1	ID_IstochnikFormirKred
2	643_7725784700_1137
3	

Рисунок 42 – Записи в готовом отчёте

Проверив данный ID источника можно убедиться, что он получен из идентификатора квитанции SIN=2084, которая соответствует одной из загруженных квитанций.

2. Тест «загрузка данных из источников».

Для начала теста стоит заполнить источники тестовыми данными. Далее необходимо дождаться запуска задачи на загрузку данных планировщиком.

```

[2025-05-26T00:11:00+03:00][DEBUG][Service Layer]: Start Prioritizer #6ed46696;
[2025-05-26T00:11:01+03:00][DEBUG][Service Layer]: Map54r4Task requested;
[2025-05-26T00:11:02+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP54R4 COUNT: 100000;
[2025-05-26T00:11:17+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP54R4 COUNT: 43424;
[2025-05-26T00:11:23+03:00][DEBUG][Service Layer]: Map54r4Task successful done;
[2025-05-26T00:11:23+03:00][DEBUG][Service Layer]: Map54r2Task requested;
[2025-05-26T00:11:23+03:00][DEBUG][Service Layer]: Map55r3Task requested;
[2025-05-26T00:11:23+03:00][DEBUG][Service Layer]: Map55r5Task requested;
[2025-05-26T00:11:23+03:00][DEBUG][Service Layer]: Map56r4Task requested;
[2025-05-26T00:11:23+03:00][DEBUG][Service Layer]: Map55r2Task requested;
[2025-05-26T00:11:23+03:00][DEBUG][Service Layer]: Map54r3Task requested;
[2025-05-26T00:11:23+03:00][DEBUG][Service Layer]: Map56r4Task successful done;
[2025-05-26T00:11:23+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP55R3 COUNT: 635;
[2025-05-26T00:11:23+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP54R2 COUNT: 1;
[2025-05-26T00:11:23+03:00][DEBUG][Service Layer]: Map55r3Task successful done;
[2025-05-26T00:11:23+03:00][DEBUG][Service Layer]: Map54r2Task successful done;
QUERY 55r5 Time: 0.422180
[2025-05-26T00:11:24+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP55R2 COUNT: 100000;
[2025-05-26T00:11:24+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP54R3 COUNT: 100000;
ROWS SCAN 55r5 Time: 1.495103
[2025-05-26T00:11:25+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP55R5 COUNT: 143371;
[2025-05-26T00:11:30+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP55R2 COUNT: 100000;
[2025-05-26T00:11:31+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP54R3 COUNT: 100000;
[2025-05-26T00:11:37+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP55R2 COUNT: 100000;
[2025-05-26T00:11:38+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP54R3 COUNT: 100000;
[2025-05-26T00:11:45+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP55R2 COUNT: 100000;
[2025-05-26T00:11:45+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP54R3 COUNT: 31873;
[2025-05-26T00:11:47+03:00][DEBUG][Service Layer]: Map54r3Task successful done;
[2025-05-26T00:11:51+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP55R2 COUNT: 100000;
[2025-05-26T00:11:57+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP55R2 COUNT: 100000;
[2025-05-26T00:11:59+03:00][DEBUG][Service Layer]: Map55r5Task successful done;
[2025-05-26T00:12:00+03:00][DEBUG][Service Layer]: Start Prioritizer #3bede6d3;
[2025-05-26T00:12:01+03:00][DEBUG][Service Layer]: Map54r4Task requested;
[2025-05-26T00:12:01+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP54R4 COUNT: 412;
[2025-05-26T00:12:01+03:00][DEBUG][Service Layer]: Map54r4Task successful done;
[2025-05-26T00:12:01+03:00][DEBUG][Service Layer]: Map54r2Task requested;
[2025-05-26T00:12:01+03:00][DEBUG][Service Layer]: Map56r4Task requested;
[2025-05-26T00:12:01+03:00][DEBUG][Service Layer]: Map54r3Task requested;
[2025-05-26T00:12:01+03:00][DEBUG][Service Layer]: Map55r3Task requested;
[2025-05-26T00:12:01+03:00][DEBUG][Service Layer]: Map55r5Task requested;
[2025-05-26T00:12:01+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP54R2 COUNT: 1;
[2025-05-26T00:12:01+03:00][DEBUG][Service Layer]: Map56r4Task successful done;
[2025-05-26T00:12:01+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP55R3 COUNT: 635;
[2025-05-26T00:12:01+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP54R3 COUNT: 922;
[2025-05-26T00:12:01+03:00][DEBUG][Service Layer]: Map54r3Task successful done;
QUERY 55r5 Time: 0.064734
ROWS SCAN 55r5 Time: 0.006847
[2025-05-26T00:12:01+03:00][DEBUG][Report Mapper Module (Mapping)]: MAP55R5 COUNT: 419;
[2025-05-26T00:12:01+03:00][DEBUG][Service Layer]: Map55r3Task successful done;

```

Рисунок 43 – Лог МСО с информацией о запущенных задачах

Далее стоит проверить средствами ПМИ загрузку данных из источников. Просмотрев частично таблицы, видно, что они были заполнены.

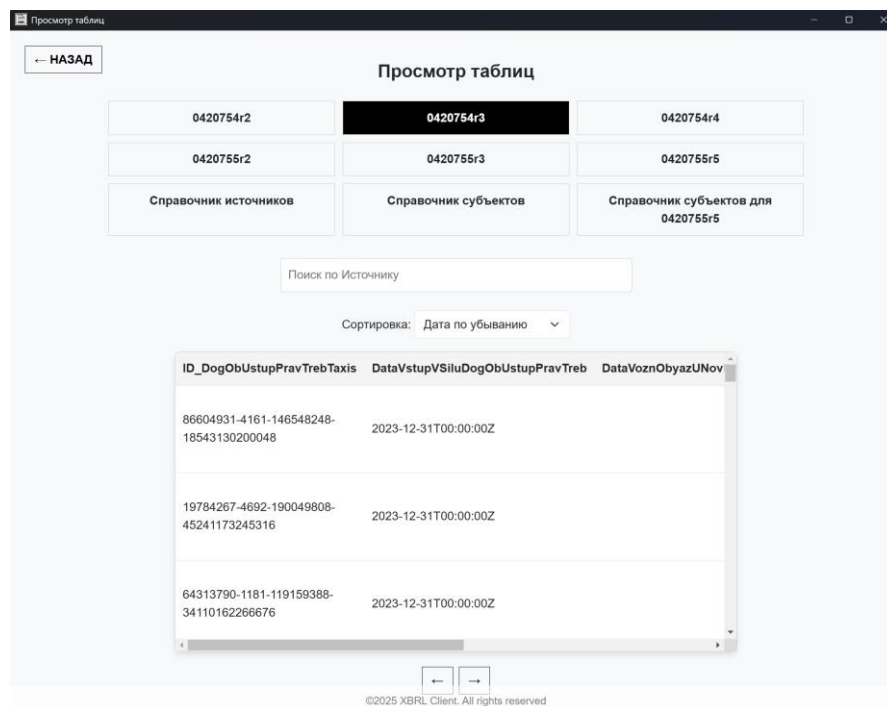


Рисунок 44 – Таблица для отчёта «0420754r3»

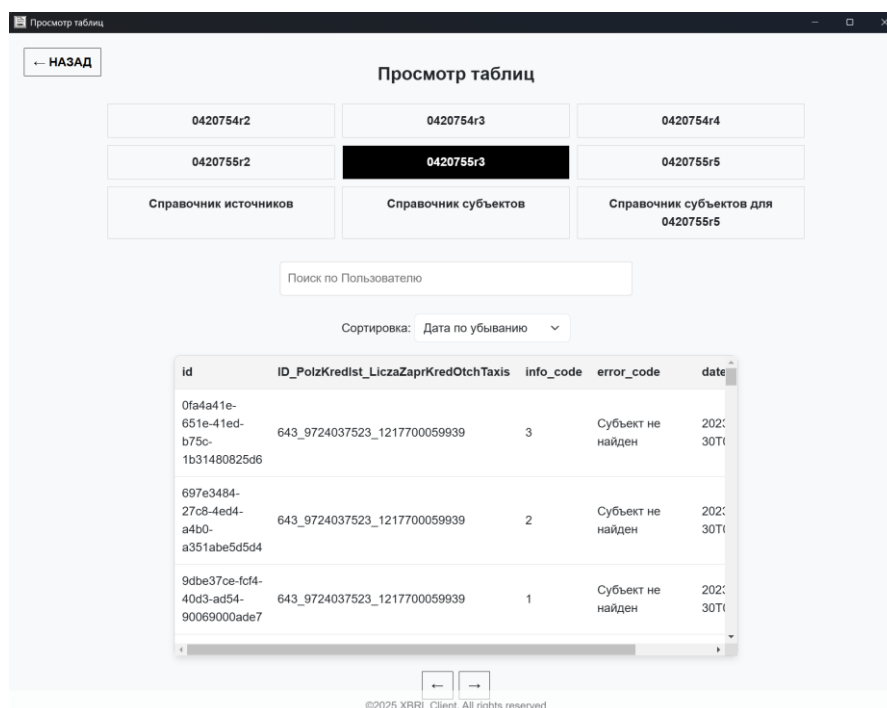


Рисунок 45 – Таблица для отчёта «0420755r3»

Также стоит провести выборочный поиск по ID контрагентов из источников в загруженных данных. Например, для таблицы отчёта «0420754r3» найдены записи.

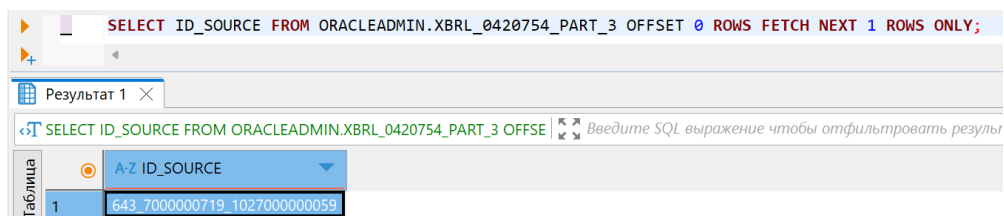


Рисунок 46 – Получение случайного ID контрагента из таблицы-источника

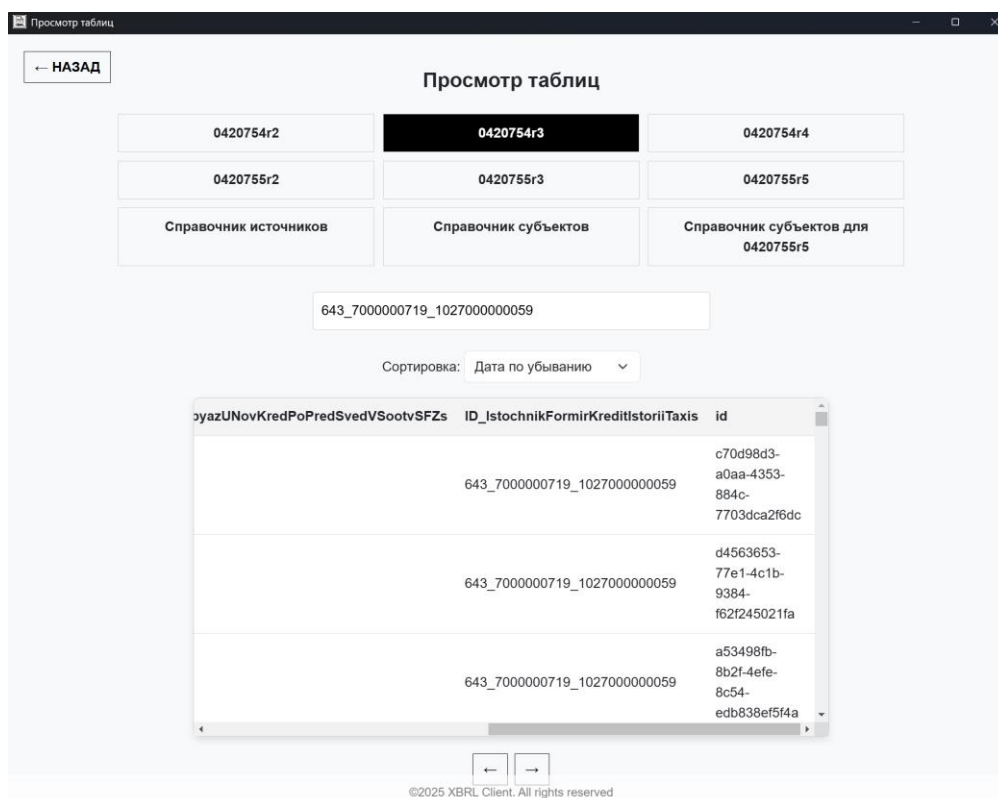


Рисунок 47 – Поиск по выбранному ID контрагента в подготовленной таблице

Аналогичный тест для таблицы отчёта «0420755r3», где найдены следующие записи.

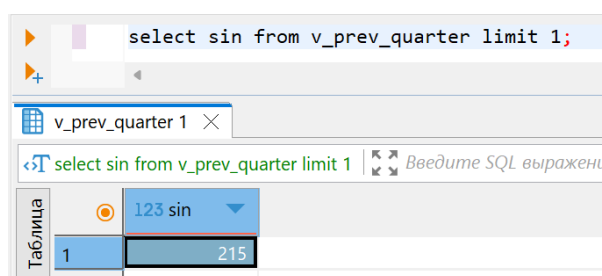


Рисунок 48 – Получение случайного идентификатора SIN

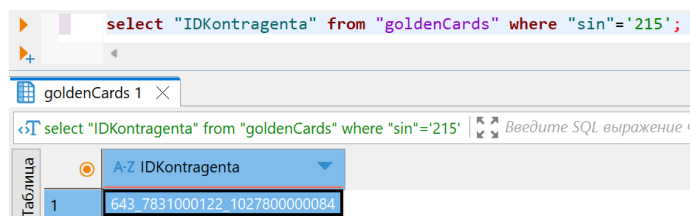


Рисунок 49 – Подбор соответствия ID контрагента по идентификатору SIN

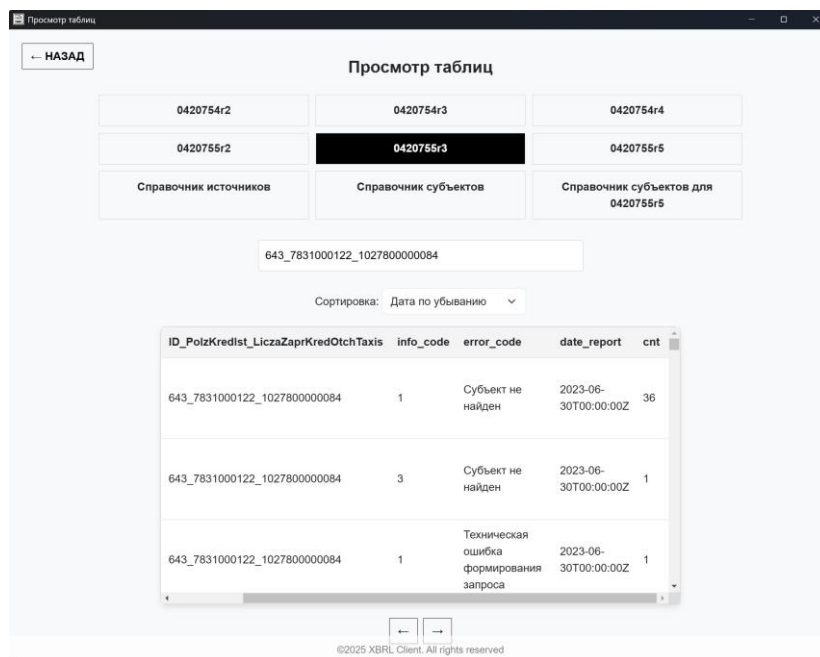


Рисунок 50 – Поиск по выбранному ID контрагента в подготовленной таблице

3. Тест «формирование отчётов»

Для получения сформированных отчётов стоит для начала сделать запрос средствами ПМИ на их генерацию.

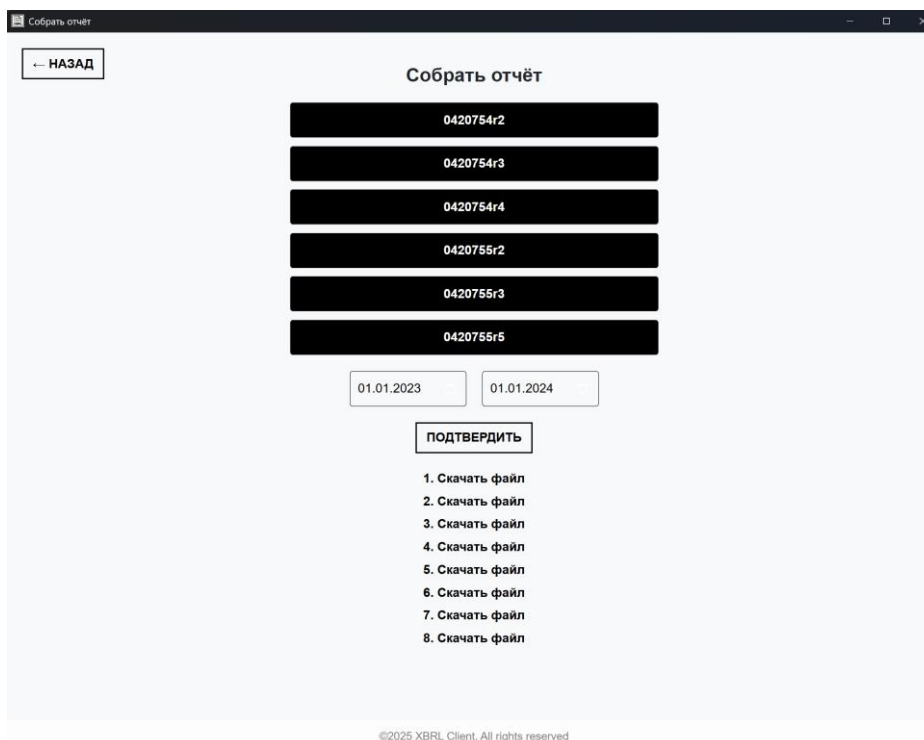


Рисунок 51 – Запрос на генерацию отчётов за год

Сегодня			
	0420755r5from2023-01-01to2024-01-01.csv	26.05.2025 0:45	Файл Microsoft Excel... 19 941 КБ
	0420755r3from2023-01-01to2024-01-01.csv	26.05.2025 0:45	Файл Microsoft Excel... 167 КБ
	0420755r2from2023-01-01to2024-01-01.csv	26.05.2025 0:45	Файл Microsoft Excel... 94 КБ
	0420754r4from2023-01-01to2024-01-01.csv	26.05.2025 0:45	Файл Microsoft Excel... 17 728 КБ
	0420754r3from2023-01-01to2024-01-01.csv	26.05.2025 0:45	Файл Microsoft Excel... 25 825 КБ
	0420754r2from2023-01-01to2024-01-01.csv	26.05.2025 0:44	Файл Microsoft Excel... 1 КБ

Рисунок 52– Информация по скачанным отчётам

Для корректности отчётов стоит проверить те из них, которые собираются один к одному в соответствии с подготовленной таблицей. И из них видно, что количество записей в таблице соответствует количеству записей в отчёте.

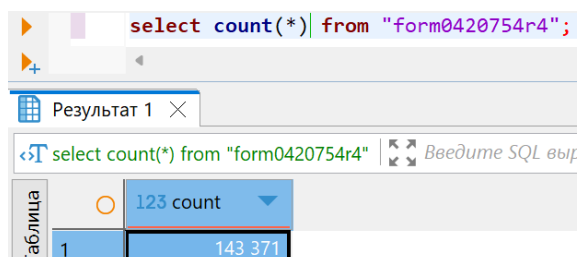


Рисунок 53 – Количество записей из таблицы отчёта «0420754r4»

143372	643_9909391439_1624_0000	0	15952067-	#####
143372				
<	>	0420754r4from2023-01-01to2024-0		+

Рисунок 54 – Количество записей в готовом отчёте «0420754r4»

<pre>select count(*) from "form0420754r3";</pre>	
Результат 1	
<pre>select count(*) from "form0420754r3"</pre>	
Таблица	123 count
1	330 575

Рисунок 55 – Количество записей из таблицы отчёта «0420754r3»

330576	643_9710091751335-	31.12.2023		
330577				
330578				
<	>	0420754r3from2023-01-01to2024-0		+

Рисунок 56 – Количество записей в готовом отчёте «0420754r3»

Таким образом, можно сделать вывод, что пользователь может отправить запрос на генерацию отчётов, и в отчёты собираются подготовленные данные за указанный период. Апробацию системы можно считать завершённой успешно.

Заключение

Была проделана работа по созданию системы автоматического формирования XBRL-отчетов. В ходе неё были получены следующие результаты.

1. Проведён обзор предметной области XBRL-отчётности, определены виды отчётов для реализации, а именно: форма 0420754 «Сведения об источниках формирования кредитных историй» части II, III, IV, форма 0420755 «Сведения о пользователях кредитных историй» части II, III, V, форма 0420762 «Реестр контрагентов». Изучены бизнес-требования, полученные от организации.
2. Разработана микросервисная архитектура системы: выделены сервисы сбора отчётов, обработки квитанций, пользовательского интерфейса, определено их взаимодействие. Выбран язык программирования Golang для реализации серверной части, а также JavaScript для клиентской части. Выбран инструмент k6 для проведения нагрузочного тестирования.
3. Выполнена реализация системы:
 - серверная часть: реализован основной блок автоматизации задач, блок ручного управления, блок аудита задач; созданы модули, повышающие обеспечение автономности работы системы: планировщик, выбор лидера, система определения приоритета и модуль исполнения задач;
 - клиентская часть: создан кросс-платформенный пользовательский интерфейс, позволяющий пользователям выполнять манипуляции с загруженными данными, обрабатывать квитанции и просматривать сформированные отчёты.
4. Выполнено тестирование и апробация системы:
 - проведено нагрузочное тестирование системы. Определены средние показатели возможности каждого микросервиса: более 1000 RPS на

1 “Kubernetes” под микросервиса МСО и более 300 RPS аналогично на микросервис МОК;

- проведена апробация системы по основным бизнес-кейсам, получен успешный результат по процессу формирования отчётов, по обработке квитанций.

Список источников

[1] Learn Microsoft — URL: <https://learn.microsoft.com/ru-ru/dynamics365/business-central/bi-create-reports-with-xbrl> (дата обращения 09.04.2025)

[2] Банк России. Открытый стандарт отчётности XBRL — URL: https://cbr.ru/projects_xbrl/ (дата обращения 09.04.2025)

[3] Банк России. Форма 0420754 «Сведения об источниках формирования кредитных историй» — URL: https://cbr.ru/explan/ot_bki/forma-0420754/ (дата обращения 09.04.2025)

[4] Банк России. Форма 0420755 «Сведения о пользователях кредитных историй» — URL: https://cbr.ru/explan/ot_bki/forma-0420755/ (дата обращения 09.04.2025)

[5] Банк России. Форма 0420762 «Реестр контрагентов» — URL: https://cbr.ru/explan/ot_bki/forma-0420762/ (дата обращения 09.04.2025)

[6] Официальный сайт «UBPARTNER». Описание инструментов «XBRL TOOLS» — URL: <https://www.ubpartner.com/xbrl-toolkit/> (дата обращения 21.05.2025)

[7] Официальный сайт «Avancore». Описание программного продукта «XBRL» — URL: <https://www.avancore.ru/services/avankor-xbrl/> (дата обращения 21.05.2025)

[8] Документация Ceph S3. Введение. — URL: <https://docs.ceph.com/en/latest/radosgw/s3/> (дата обращения 09.04.2025)

[9] Документация PostgreSQL. Обзор. — URL: <https://www.postgresql.org/> (дата обращения 09.04.2025)

[10] Документация Oracle. Базы данных. — URL: <https://www.oracle.com/database/> (дата обращения 09.04.2025)

[11] Документация Docker. Описание Dockerfile. — URL: <https://docs.docker.com/reference/dockerfile/> (дата обращения 21.05.2025)

- [12] Официальная документация языка Golang. Эффективный Go. — URL https://go.dev/doc/effective_go (дата обращения 09.04.2025)
- [13] Документация библиотеки «go-pg». Обзор. — URL: <https://pg.uptrace.dev/> (дата обращения 16.02.2023)
- [14] Github документация библиотеки «goracle». Обзор. — URL: <https://github.com/go-goracle/goracle> (дата обращения 16.02.2023)
- [15] Документация библиотеки «AWS SDK for Go v2». Введение. — URL: <https://aws.github.io/aws-sdk-go-v2/docs/> (дата обращения 16.02.2023)
- [16] Документация по JavaScript. Обзор. — URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (дата обращения 21.05.2025)
- [17] Официальный сайт утилиты «Goose». Обзор. — URL: <https://pressly.github.io/goose/> (дата обращения 21.05.2025)
- [18] Документация K6. Понимание результатов k6-тестирования. — URL: <https://github.com/grafana/k6-learn/blob/main/Modules/II-k6-Foundations/03-Understanding-k6-results.md> (дата обращения 09.04.2025)
- [19] Платформа Хабр. Просто о микросервисах. — URL: <https://habr.com/ru/companies/raiffeisenbank/articles/346380/> (дата обращения 21.05.2025)
- [20] Документация Oracle. Oracle Instant Client. — URL: <https://www.oracle.com/database/technologies/instant-client.html> (дата обращения 21.05.2025)
- [21] Github документация Vault. Обзор. — URL: <https://github.com/hashicorp/vault> (дата обращения 21.05.2025)
- [22] Документация Electron.js. Введение. — URL: <https://electronjs.org/docs/latest> (дата обращения 09.04.2025)
- [23] Документация Qt. Обзор. — URL: <https://doc.qt.io/> (дата обращения 09.04.2025)
- [24] Документация JavaFX. Обзор. — URL: <https://openjfx.io/> (дата обращения 09.04.2025)

[25] Документация Microsoft .NET WPF. Документация по Windows Presentation Foundation. — URL: <https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/?view=netdesktop-9.0> (дата обращения 09.04.2025)

[26] Документация Electron/packager. О программе. — URL: <https://electron.github.io/packager/main/> (дата обращения 09.04.2025)

[27] Платформа Хабр. Асинхронность как основной подход к разработке высоконагруженных приложений. — URL: <https://habr.com/ru/articles/733154/> (дата обращения 09.04.2025)

[28] Платформа Хабр. Как построить систему, способную выдерживать нагрузку в 5 млн RPS. — URL: <https://habr.com/ru/companies/ozontech/articles/749328/> (дата обращения 09.04.2025)

[29] Официальный портал Grafana. Тестирование при средней нагрузке: руководство для начинающих. — URL: <https://grafana.com/blog/2024/01/30/average-load-testing/> (дата обращения 09.04.2025)

Приложение 1

Листинг Dockerfile для сбора МОК.

```
# использование golang образа для alpine версии linux
FROM golang:1.22-alpine as dj

WORKDIR /app

# установка необходимых библиотек
RUN apk -U upgrade
RUN apk add build-base
RUN apk add shadow

# копирование файла с зависимостями в образ
COPY go.mod go.sum ./

# скачивание зависимостей внутрь образа
RUN go mod download

# копирование остальных файлов проекта
COPY . .

# создание отдельного пользователя для запуска системы
RUN groupadd -r dj && useradd -r -g dj download-journal

# смена пользователя
USER download-journal

# сбор приложения
RUN CGO_ENABLED=0 GOOS=linux go build -buildvcs=false -o /download-journal

# открытие доступа к порту приложения
EXPOSE 10223

# открытие доступа к порту метрик приложения
EXPOSE 2113

ENTRYPOINT ["/download-journal"]
```

Приложение 2

Листинг Dockerfile для сбора МСО.

```
# использование golang образа для alpine версии linux
FROM golang:1.18-alpine as xbrl

# установка необходимых библиотек
RUN apk -U upgrade
RUN apk add build-base
RUN apk add shadow
RUN apk add libaio

# создание отдельного пользователя для запуска системы
RUN groupadd -r xbrl && useradd -r -g xbrl xbrl-report

# установка директории для библиотек oracle
WORKDIR /opt/oracle

# скачивание библиотек и oracle instant client
RUN apk --no-cache add libaio libnsl libc6-compat curl && \
    cd /tmp && \
    curl -o instantclient-basiclite.zip \
    https://download.oracle.com/otn_software/linux/instantclient/instantclient-basiclite-linuxx64.zip \
    -SL && \
    unzip instantclient-basiclite.zip && \
    mv instantclient*/ /usr/lib/instantclient && \
    rm instantclient-basiclite.zip && \
    ln -s /usr/lib/instantclient/libclntsh.so.19.20 /usr/lib/libclntsh.so && \
    ln -s /usr/lib/instantclient/libocci.so.19.20 /usr/lib/libocci.so && \
    ln -s /usr/lib/instantclient/libociicus.so /usr/lib/libociicus.so && \
    ln -s /usr/lib/instantclient/libnnz19.so /usr/lib/libnnz19.so && \
    ln -s /usr/lib/libnsl.so.2 /usr/lib/libnsl.so.1 && \
    ln -s /lib/libc.so.6 /usr/lib/libresolv.so.2 && \
    ln -s /lib64/ld-linux-x86-64.so.2 /usr/lib/ld-linux-x86-64.so.2

# установка переменных окружения для oracle
ENV ORACLE_BASE /usr/lib/instantclient
```



```
ENV LD_LIBRARY_PATH /usr/lib/instantclient
ENV TNS_ADMIN /usr/lib/instantclient
ENV ORACLE_HOME /usr/lib/instantclient

WORKDIR /app

RUN chown -R xbrl-report /app
RUN chmod -R u=rwx /app

# копирование файла с зависимостями в образ
COPY go.mod go.sum ./
# скачивание зависимостей внутрь образа
RUN go mod download

# копирование остальных файлов проекта
COPY . .

# установка максимального доступа в директорию временного сбора отчётов
RUN chmod -R 777 /app/csvFiles

# сбор приложения
RUN GOOS=linux go build -buildvcs=false -o /xbrl-report

# смена пользователя
USER xbrl-report

# открытие доступа к порту приложения
EXPOSE 10222
# открытие доступа к порту метрик приложения
EXPOSE 2112

ENTRYPOINT ["/xbrl-report"]
```

Приложение 3

Листинг Dockerfile для сбора утилиты миграции.

```
# использование образа утилиты goose
FROM gomicro/goose as xbrl-migrations

WORKDIR /app

COPY ./migrations/ /app/migrations/
# файл с запуском команды применения миграции
COPY ./entrypoint.sh /app/entrypoint.sh

RUN chmod -R 777 /app

RUN apk -U upgrade
RUN apk add shadow
RUN groupadd -r xbrl && useradd -r -g xbrl xbrl-report
USER xbrl-report

ENTRYPOINT ["/app/entrypoint.sh"]
```

Приложение 4

Листинг файла нагрузочного тестирования “load_notices.js” МОК.

```
import http from 'k6/http';
import { check, sleep } from 'k6';
import { firstNotices, secondNotices, thirdNotices } from './bank.js';

export let options = {
  scenarios: {
    firstNoticesRequests: {
      executor: 'ramping-vus',
      startVUs: 1,
      stages: [
        { duration: '10s', target: 5 },
        { duration: '30s', target: 50 },
        { duration: '30s', target: 200 },
        { duration: '18m', target: 200 },
        { duration: '20s', target: 0 },
      ],
      exec: 'firstNoticesRequests',
    },
    secondNoticesRequests: {
      executor: 'ramping-vus',
      startVUs: 1,
      stages: [
        { duration: '10s', target: 5 },
        { duration: '30s', target: 50 },
        { duration: '30s', target: 200 },
        { duration: '18m', target: 200 },
        { duration: '20s', target: 0 },
      ],
      exec: 'secondNoticesRequests',
    },
    thirdNoticesRequests: {
      executor: 'ramping-vus',
      startVUs: 1,
      stages: [
        { duration: '10s', target: 5 },
        { duration: '30s', target: 50 },
        { duration: '30s', target: 200 },
        { duration: '18m', target: 200 },
        { duration: '20s', target: 0 },
      ],
      exec: 'thirdNoticesRequests',
    },
  },
};

function getRandomElement(arr) {
  return arr[Math.floor(Math.random() * arr.length)];
}

export function firstNoticesRequests() {
  let url = `${apiUrlNotices}/fillDbByFirstNotice`;
  let filename = getRandomElement(firstNotices)
  let payload = JSON.stringify({
    jsonFileName: filename,
  });

  let params = {
    headers: {
      'Content-Type': 'application/json',
    },
  },
```

```

    },
  };

  let res = http.post(url, payload, params);
  check(res, {
    'статус 200': (r) => r.status === 200,
  });
  sleep(1);
}

export function secondNoticesRequests() {
  let url = `${apiUrlNotices}/fillDbBySecondNotice`;
  let filename = getRandomElement(secondNotices)
  let payload = JSON.stringify({
    jsonFileName: filename,
  });

  let params = {
    headers: {
      'Content-Type': 'application/json',
    },
  };

  let res = http.post(url, payload, params);
  check(res, {
    'статус 200': (r) => r.status === 200,
  });
  sleep(1);
}

export function thirdNoticesRequests() {
  let url = ' ${apiUrlNotices}/fillDbByAlarm';
  let filename = getRandomElement(thirdNotices)
  let payload = JSON.stringify({
    jsonFileName: filename,
  });

  let params = {
    headers: {
      'Content-Type': 'application/json',
    },
  };

  let res = http.post(url, payload, params);
  check(res, {
    'статус 200': (r) => r.status === 200,
  });
  sleep(1);
}

```

Приложение 5

Листинг файла нагрузочного тестирования “load_xbrl.js” МСО.

```
import http from 'k6/http';
import { check, sleep } from 'k6';
import { formFiles, formNames } from './bank.js';

export let options = {
  scenarios: {
    getCsv: {
      executor: 'ramping-vus',
      startVUs: 1,
      stages: [
        { duration: '10s', target: 10 },
        { duration: '30s', target: 80 },
        { duration: '30s', target: 400 },
        { duration: '20m', target: 400 },
        { duration: '20s', target: 0 },
      ],
      exec: 'getCsv',
    },
    mapStatus: {
      executor: 'ramping-vus',
      startVUs: 1,
      stages: [
        { duration: '10s', target: 10 },
        { duration: '30s', target: 80 },
        { duration: '30s', target: 400 },
        { duration: '20m', target: 400 },
        { duration: '20s', target: 0 },
      ],
      exec: 'mapStatus',
    },
    reMap: {
      executor: 'ramping-vus',
      startVUs: 1,
      stages: [
        { duration: '10s', target: 10 },
        { duration: '30s', target: 50 },
        { duration: '30s', target: 100 },
        { duration: '20m', target: 100 },
        { duration: '20s', target: 0 },
      ],
      exec: 'reMap',
    },
    makeCsv: {
      executor: 'ramping-vus',
      startVUs: 1,
      stages: [
        { duration: '10s', target: 10 },
        { duration: '30s', target: 50 },
        { duration: '30s', target: 100 },
        { duration: '20m', target: 100 },
        { duration: '20s', target: 0 },
      ],
      exec: 'makeCsv',
    },
  },
};

function getRandomElement(arr) {
  return arr[Math.floor(Math.random() * arr.length)];
```

```

}

const startDate = new Date("2020-01-01T00:00:00.000Z")
const endDate = new Date("2026-01-01T00:00:00.000Z")

function getRandomDate(from, to) {
  const fromTime = from.getTime();
  const toTime = to.getTime();
  return new Date(fromTime + Math.random() * (toTime - fromTime));
}

export function getCsv() {
  let url = `${apiUrlXBRL}/getCsv?`;
  let filename = getRandomElement(formFiles)
  let fromDate = getRandomDate(startDate, endDate)
  let toDate = getRandomDate(fromDate, endDate)

  let fromDateMonth = (fromDate.getMonth()+1)
  if (fromDateMonth < 10) {
    fromDateMonth = "0"+fromDateMonth
  }
  let fromDateDay = (fromDate.getDay()+1)
  if (fromDateDay < 10) {
    fromDateDay = "0"+fromDateDay
  }

  let toDateMonth = (toDate.getMonth()+1)
  if (toDateMonth < 10) {
    toDateMonth = "0"+toDateMonth
  }
  let toDateDay = (toDate.getDay()+1)
  if (toDateDay < 10) {
    toDateDay = "0"+toDateDay
  }

  let fromDateStr = fromDate.getFullYear()+"-"+fromDateMonth+"-"+fromDateDay
  let toDateStr = toDate.getFullYear()+"-"+toDateMonth+"-"+toDateDay
  url = url+"formFiles="+filename+"&fromDate="+fromDateStr+"&toDate="+toDateStr

  let res = http.get(url);
  check(res, {
    'статус 200': (r) => r.status === 200,
  });
  sleep(1);
}

export function mapStatus() {
  let url = `${apiUrlXBRL}/mapStatus?`;
  let filename = getRandomElement(formNames)
  url = url+"tableName="+filename
  let res = http.get(url);
  check(res, {
    'статус 200': (r) => r.status === 200,
  });
  sleep(1);
}

export function reMap() {
  let url = `${apiUrlXBRL}/reMap`;
  let filename = getRandomElement(formNames)
  let payload = JSON.stringify({
    tableName: filename,
  });

  let params = {
    headers: {
      'Content-Type': 'application/json',
    },
  },

```

```

    });

    let res = http.post(url, payload, params);
    check(res, {
      'статус 200': (r) => r.status === 200,
    });
    sleep(1);
  }

export function makeCsv() {
  let url = `${apiUrlXBRL}/makeCsv`;
  let filename = getRandomElement(formFiles)
  let fromDate = getRandomDate(startDate, endDate)
  let toDate = getRandomDate(fromDate, endDate)

  let fromDateMonth = (fromDate.getMonth()+1)
  if (fromDateMonth < 10) {
    fromDateMonth = "0"+fromDateMonth
  }
  let fromDateDay = (fromDate.getDay()+1)
  if (fromDateDay < 10) {
    fromDateDay = "0"+fromDateDay
  }

  let toDateMonth = (toDate.getMonth()+1)
  if (toDateMonth < 10) {
    toDateMonth = "0"+toDateMonth
  }
  let toDateDay = (toDate.getDay()+1)
  if (toDateDay < 10) {
    toDateDay = "0"+toDateDay
  }

  let fromDateStr = fromDate.getFullYear()+"-"+fromDateMonth+"-"+fromDateDay
  let toDateStr = toDate.getFullYear()+"-"+toDateMonth+"-"+toDateDay

  let payload = JSON.stringify({
    toDate: toDateStr,
    fromDate: fromDateStr,
    formFiles: [filename],
  });

  let params = {
    headers: {
      'Content-Type': 'application/json',
    },
  };

  let res = http.post(url, payload, params);
  check(res, {
    'статус 200': (r) => r.status === 200,
  });
  sleep(1);
}

```