

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление 09.03.01 «Информатика и вычислительная техника»

подготовки:

Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

БАКАЛАВРА

на тему

**Микросервис синхронизации данных для
компонентов информационной системы**

Студент Калашников В.А. 03.06.2020
(подпись, дата)

Руководитель Синев М.П. 03.06.2020
(подпись, дата)

Нормоконтролёр А.В. Кучин
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 03.06.2020 г. № 12)

Заведующий кафедрой М.А. Митрохин
(подпись)

Работа защищена с отметкой Отлично (протокол заседания ГЭК от 17.06.2020 г. № 8)

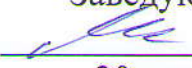
Секретарь ГЭК Е.И. Гурин
(подпись)

Пенза 2020

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«30» апреля 2020 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВАРА

1. Студент Калашников Виталий Александрович гр.16BV1 факультета вычислительной техники.

2. Тема работы: «Микросервис синхронизации данных для компонентов информационной системы»

Тема утверждена приказом ПГУ №0335а/о от "30" апреля 2020 г.

3. Руководитель бакалаврской работы к.т.н., доцент каф. ВТ ПГУ Синев М.П.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать приложение «Микросервис синхронизации данных для компонентов информационной системы», выполняющее двунаправленную передачу данных между клиентами и сервером. Система должна обеспечивать целостность данных, передавать клиентам только актуальную информацию, обеспечить безопасность от порчи данных извне, выдерживать высокие нагрузки, иметь возможность масштабироваться и беспрепятственно расширяться. Разрабатываемое приложение должно выполнять задачи проектирования, общего анализа и моделирования вычислительных систем.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметных областей: изучить актуальные на сегодняшний день решения, обеспечивающие двустороннюю передачу информации. Необходимо изучить методологии проектирования объектно-ориентированных приложений, а также изучить функционал фреймворка Spring.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	10	30.04.2020 – 07.05.2020	
2.	Анализ требований	10	08.05.2020 – 10.05.2020	
3.	Выбор технологий, применяемых при проектировании и разработке	15	11.05.2020 – 13.05.2020	
4.	Описание алгоритмов	15	14.05.2020 – 16.04.2020	
5.	Разработка приложения «Микросервис синхронизации данных для компонентов информационной системы»	30	17.04.2020 – 27.05.2020	
6.	Оформление пояснительной записки	10	28.05.2020 - 01.06.2020	

Дата выдачи задания "30" апреля 2020 г.

Руководитель бакалаврской работы  30.04.2020 Синев М.П.
(подпись, дата)

Задание к исполнению принял студент  30.04.2020 Калашников В.А.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 150 листа, 17 рисунков, 17 таблиц, 11 источников, 4 приложения.

ИНФОРМАЦИОННЫЕ СИСТЕМЫ, БД, СУБД, JAVA.

Цель работы – разработать приложение, обеспечивающий двусторонний обмен данными между клиентскими системами и сервером.

Объектом разработки является приложение, обеспечивающее выполнение следующих функций: обработка запросов от клиентов, передача только актуальных данных клиентам с обеспечением защиты от «ложных клиентов», обеспечение работоспособности системы под большими нагрузками. Хранение информации, полученное от клиентов должно осуществляться в БД. Так же необходимо решить задачу обеспечения высокой производительности за счет оптимизации процесса обмена информации.

В результате проведенной работы было разработано приложение, удовлетворяющее поставленной задаче, а именно: разработана структура БД, спроектированы классы, реализующий необходимый функционал. Приложение было отлажено и успешно интегрировано в целевую информационную систему.

					ПГУ 2.090301.18.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	<i>Микросервис синхронизации данных для компонентов информационной системы Пояснительная записка</i>	Лит.	Лист	Листов
Разраб.		<i>Калашиников В.А.</i>						
Провер.		<i>Синев М.П.</i>					2	150
Н. Контр		<i>Кучин А.В.</i>				ФВТ, каф. ВТ гр. 16ВВ1		
Утверд.		<i>Митрохин М.А.</i>						

Перечень принятых сокращений

API (Application Programming Interface) – программный интерфейс приложения;

HTTP (HyperText Transfer Protocol) – протокол передачи гипертекста;

ID (Identifier) – идентификатор;

REST (Representational State Transfer) – передача состояния представления;

SQL (Structured Query Language) – язык структурированных запросов;

UML (Unified Modeling Language) – унифицированный язык моделирования;

URI (Uniform Resource Identifier) – унифицированный (единообразный) идентификатор ресурса;

JDBC (Java DataBase Connectivity) – соединение с базами данных на Java;

XML (eXtensible Markup Language) – расширяемый язык разметки;

UUID (Universally Unique Identifier) – универсальный уникальный идентификатор;

JDBC (англ. Java DataBase Connectivity) – соединение с базами данных на Java;

БД – База данных;

СУБД – система управления базами данных;

ИС – информационная система;

КПС – клиентская подсистема;

ЭЦП – электронно-цифровая подпись;

ПЭВМ – персональная электронно-вычислительная система;

ОО – обслуживаемая организация.

					ПГУ 2.090301.18.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		3

Содержание

Перечень принятых сокращений	3
Введение.....	6
1 Анализ предметной области	7
1.1 Описание целевой системы.....	7
1.2 Постановка задачи.....	9
1.3 Обзор готовых решений	10
1.4 Выбор языка программирования.....	12
1.5 Выбор СУБД.....	17
1.6 Вывод.....	20
2 Реализация.....	21
2.1 Описание протокола	21
2.2 Структура БД.....	26
2.3 Описание классов.....	32
2.4 Программная реализация	36
2.5 Вывод.....	37
3 Описание результата.....	38
3.1 Описание микросервиса	38
3.2 Описание сервиса автораспределения.	41
3.3 Результат работы микросервиса синхронизации и влияние модуля автораспределения.	47
3.4 Вывод.....	49
4 Безопасные условия труда оператора ЭВМ	50
4.1 Анализ условий труда оператора ЭВМ.....	50
4.2 Меры борьбы с вредными воздействиями.....	55
Заключение	59
Список используемых источников.....	60
Приложение А. Листинги программы на Java	61
Приложение В. Листинг XML-файла.....	135

Приложение С. Листинг SQL-файла	138
Приложение D. Презентация	141

					ПГУ 2.090301.18.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

Наиболее характерная черта нынешнего состояния мировой экономики - колоссальные успехи и достижения в области вычислительной техники и информационных технологий, а также развитие наукоемких производств. Высокие темпы развития науки и технологий обработки информации, а главное масштабы и темпы их внедрения в производство и в общественную жизнь, превратили научно-техническую революцию в естественный процесс, она стала перманентной. В условиях динамичного развития рынка, усложнения его инфраструктуры, информация становится таким же стратегическим ресурсом, как материалы и энергия.

Современные технологии, позволяющие создавать, хранить, перерабатывать информацию, обеспечивать эффективные способы её представления, стали важным фактором обеспечения конкурентоспособности и средством повышения эффективности управления всеми сферами общественной жизнедеятельности. Уровень информатизации является сегодня одним из главных факторов успешного развития любого предприятия.

На фоне повсеместной информатизации общества и производства стали появляться компании, занимающиеся разработкой комплексных ИС. Однако для каждого подобного предприятия встает вопрос организации передачи данных между компонентами внутри ИС.

1 Анализ предметной области

1.1 Описание целевой системы

Архитектурно целевая ИС состоит из сервера балансировки нагрузки, центрального кластера серверов обработки данных, мастер-БД и двух репликации для быстрого доступа к данным для чтения, а также из множества КПС, установленных в ОО. Целевая ИС осуществляет обмен данными между клиентскими подсистемами и с смежными сторонними ИС. Общая схема ИС представлена на рисунке 1.

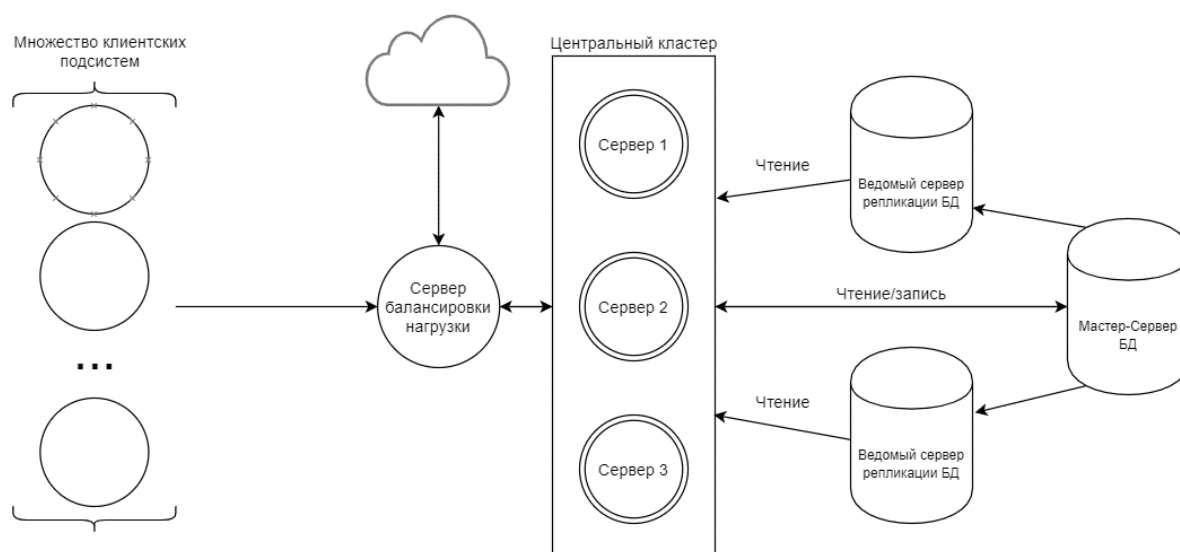


Рисунок 1 — Графического представление архитектуры ИС.

КПС состоит из внутреннего центрального сервера, локальной БД и множества вспомогательного ПО, которое устанавливается в зависимости от потребностей конечного потребителя услуг. Схема КПС представлен на рисунке 2.

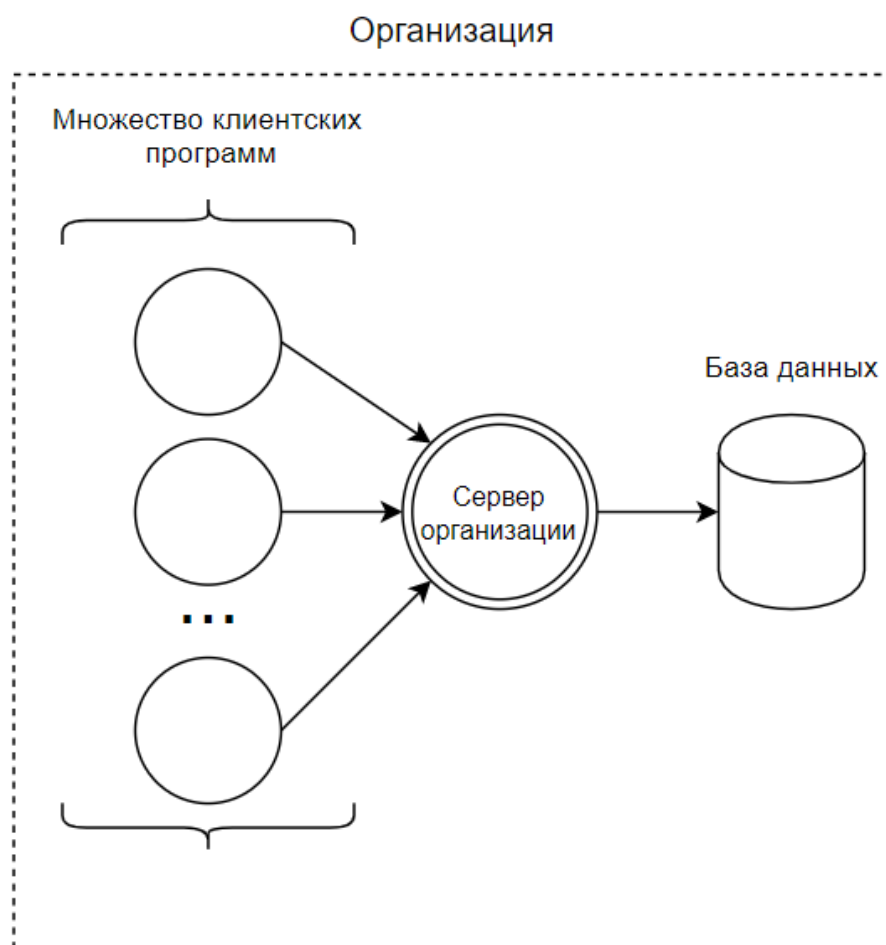


Рисунок 2 —графического представление архитектуры клиентской подсистемы.

Каждая клиентская программа в ходе работы фиксирует различные события. События передаются на сервер организации для централизованного хранения и управления. Примером таких событий являются: регистрация нового клиента в ОО или изменение данных у старого, событие прохода на территорию ОО, продажа товаров на территории в том числе вендинговой продукции.

Согласно требованию к ИС, КПС должны отправлять некоторые события в режиме реального времени, иные могут быть отправлены по итогу дня.

1.2 Постановка задачи

Требуется обеспечить двустороннюю передачу данных между центральным кластером ИС и множеством КПС.

Модуль синхронизации данных должен соответствовать следующим требованиям:

1. Актуальность. Пакеты данных от КПС должны приходит своевременно и обязательно попадать после обработки в центральный сервер БД на хранение. Так же КПС должны получать все необходимые данные от центрального кластера.

2. Производительность. Модуль должен выдерживать высокие нагрузки, создаваемые наплывом от КПС большого числа запросов на синхронизацию данных.

3. Масштабируемость. Модуль должен легко масштабироваться для распределения нагрузки на сервера и не конфликтовать между собой при проведении сеанса синхронизации.

4. Отказоустойчивость. В случае ошибок в какой-либо из секций, сеанс синхронизации не должен прерываться и целевая КПС должна получить ответ с обработанными данными.

5. Расширяемость. Разработчик должен иметь возможность расширять функциональность модуля и добавлять обработку новых сущностей без риска вывести из строя старый функционал.

6. Безопасность. Требуется проводить валидацию данных, а так цифровую подпись в пакете для аутентификации КПС.

7. Низкие требования к системе. Модуль не должен быть требователен к ресурсам ИС. Модуль должен иметь возможность устанавливаться на той же ЭВМ, что и программный сервер ИС.

Исходя из требований к модулю синхронизации, требуется проанализировать готовые решения.

1.3 Обзор готовых решений

На рынке представлено множество решений, использующих разные способы обмена данных между участниками. Это передача данных через брокеров сообщений, репликация БД и прочие коммерческие решения. Каждый представленный метод будет разобран далее.

1.3.1 Репликация БД. SymmetricDS.

SymmetricDS – это программное обеспечение с открытым исходным кодом для синхронизации баз данных с репликацией Multi-master. Он предназначен для масштабирования большого количества узлов, работы через соединения с низкой пропускной.

Синхронизация данных происходит асинхронно, при этом изменения данных отправляются через операцию «push» или «pull». Для передачи данных использует протоколы HTTP. Ключевой чертой *SymmetricDS* является возможность производить синхронизация с множеством СУБД за счет использования JDBC. В итоге SymmetricDS обеспечивает поддержку Oracle, MySQL, MariaDB, PostgreSQL, Greenplum, SQL Server, SQL Server Azure, HSQLDB, H2, Derby, DB2, Firebird, Informix, Interbase, SQLite, Sybase ASE, Sybase ASA, MongoDB, Amazon_Redshift и VoltDB и позволяет синхронизировать данные между разными СУБД.

Однако данное решение не подходит из-за особенности архитектуры целевой ИС. Если репликацию центральной БД можно сделать в КПС, то репликации БД всех имеющихся КПС на центральном сервере потребует очень много ресурсов. Данный метод подходит под задачи масштабирования и слияния нескольких ИС.

1.3.2 Брокеры сообщений. Apache Kafka и RabbitMQ.

Apache Kafka и RabbitMQ – распределенные брокеры сообщений, предназначенные для передачи данных между компонентами ИС. Данные

брокеры являются отказоустойчивыми и поддерживают вертикальное и горизонтальное масштабирование.

В Apache Kafka приложения-клиенты посылают сообщения на узел Kafka, и указанные сообщения обрабатываются другими приложениями, так называемыми потребителями. Указанные сообщения сохраняются, а потребители подписываются для получения новых сообщений. Kafka гарантирует, что все сообщения будут упорядочены именно в той последовательности, в которой поступили. Kafka не отслеживает, какие записи считываются потребителем и после этого удаляются, а просто хранит их в течение заданного периода времени. Потребители сами опрашивают Kafka, не появилось ли у него новых сообщений, и указывают, какие записи им нужно прочесть.

В RabbitMQ приложения-источник посылают сообщения на узел RabbitMQ, при этом RabbitMQ отсылает обратно источникам подтверждение, что сообщение получено. Клиенты постоянно соединены с RabbitMQ по TCP и ждут, когда RabbitMQ пришлет им сообщения. Получатели подтверждают получение сообщения или сообщают о неудаче. Если доставка неудачна, то RabbitMQ повторяет отправку сообщения до тех пор, пока оно не будет доставлено. После успешной доставки сообщение удаляется из очереди.

Apache Kafka и RabbitMQ не подходят потому что требователен к ресурсам ИС из-за сложной обработки очереди сообщений, так же очереди сообщений для каждой КПС могут занять очень много места на диске.

1.3.3 Прочие коммерческие решения.

Большинство решений для обеспечения двунаправленного обмена информации между компонентами ИС так или иначе справляются с задачей, но сильно уступают в отказоустойчивости и гибкости в угоду производительности.

Отличительной чертой всех решений заключается в зависимости от разработчиков продукта. В случае если в продукте будет найдена ошибка, критически сказывающийся на работе модуля синхронизации, то ликвидировать ее никак не удастся, пока это не сделает сторонняя команда разработчиков, что очень важно.

1.3.4 Итог.

Все рассмотренные решения так или иначе отвечают требованиям поставленной задачи, но не перекрывают абсолютно все пункты. В связи с чем было принято решение разработать собственный модуль синхронизации данных между компонентами для целевой ИС.

1.4 Выбор языка программирования

Важной частью разработки решения является выбор языка программирования и СУБД для хранения данных. В данном разделе будут разобраны основные языки программирования, используемые для разработки коммерческих проектов.

1.4.1 Язык программирования Go

Go — компилируемый язык программирования, разработанный компанией Google. Синтаксис и семантика языка является что-то среднее между языком C и Python. Разработчики языка Go пытались решить проблемы, которые возникли в ходе развития у других языков.

Первую проблему, которую успешно решили, это работа с многопоточностью. Большинство современных языков программирования были разработаны в года, когда процессоры обладали лишь одним ядром. С появлением могоядерности и, как следствия, многопоточности, языки смогли работать с новыми возможностями процессорами через специальные библиотеки и системные вызовы к операционной системе. Однако от разработчика программного обеспечения теперь требовалось решать проблемы взаимной блокировки нитей управления и избегания состояния

гонки. Разработчики языка абстрагировались от нитей управления, введя сущность «Горутин» и обеспечив работу с многопоточностью на уровне языка, а не на уровне системных вызовов и библиотек.

Вторая проблема, которую пытались решить разработчики языка, это обеспечение самодостаточности языка путем включения необходимых функций в стандартную библиотеку языка, а не через получение необходимых пакетов из вне. Используемые ныне языки программирования со временем эволюционировали и заняли ниши, к которым они изначально не были приспособлены. В рядовых языках расширение возможности обеспечивается через менеджеров управления зависимостями или просто через менеджеров пакетов. Так, например, для быстрого создания RESTful приложения на Java необходимо использовать Maven и включить в проект Spring, тогда как на Go достаточно подключить в коде проекта библиотеку net/http.

Но язык Go не является объектно-ориентированным в полном смысле, так как в язык сознательно не была включена концепция наследования для решения проблем сильной связности производных классов от класс-родителя. Взамен этого предлагается использовать композицию, что обеспечивает гораздо высокую гибкость и устойчивость к изменениям.

1.4.2 Язык программирования Python

Python – высокоуровневый язык программирования, который ориентирован на повышение производительности разработчика и читаемости кода. Этот язык очень гибкий и обладает широким функционалом, а код высокой степенью читаемости. Но у данного языка есть существенный недостаток – язык является интерпретируемым, из-за чего он обладает низким быстродействием. Задачу по повышению быстродействия языка пытаются решить путем частичной компиляции компонентов стандартной

библиотеки и совершенствованием Just-In-Time компилятора, который входит в состав интерпретатора.

Есть 3 основных направления, где используется Python:

1. WEB-разработка. Существуют два популярных Python-фреймворка, позволяющих быстро разработать WEB-приложение и это Django и Flask. Так, например, на Django реализованы такие сайты, как DropBox, Instagram, Reddit, Spotify, Pinterest и некоторые сервисы Google, например, YouTube и Google Search.

2. Data Science и Machine Learning. В индексе пакетов Python (Python Package Index, PyPi) хранятся все пакеты, разработанные для языка Python. В состав PyPi входят такие мощные инструменты для решения задач машинного обучения, как scikit-learn и TensorFlow. Благодаря простому синтаксису языка и высокой производительности представленных пакетов, Python стал ключевым языком в сфере Data Science.

3. Вспомогательные функции и автоматизация процессов. Одна из сфер применения Python – это написание небольших скриптов для автоматизации различных рабочих операций и процессов. В качестве примера можно привести систему генерации SQL-инъекций с обработкой входных данных, например, создание COMMENT ON выражений по документу с описанием таблиц.

1.4.3 Язык программирования JavaScript

JavaScript – язык программирования с слабой динамической типизацией. Наиболее часто используется в качестве языка сценариев веб-страниц. JavaScript поддерживает императивный, декларативный и объектно-ориентированный стили программирования. JavaScript часто используется в таких программных продуктах, которые в свою очередь используются в веб-разработке. Так как язык является интерпретируемым, то

производительность и безопасность зависит от конкретной реализации интерпретатора («движка»).

Данный язык имеет ряд недостатков и противоречий. Например, в языке нет многопоточности, ее эмулирует очередь задач, который так же имеет свои особенности. Но основным недостатком является недетерминированное поведение переменных при различных операциях, вызванное слабой типизацией.

Так на рисунке 3 представлен слияние двух массивов а и b, массив а состоит из элементов {1, 2, 3, 4, 5}, когда b состоит из элементов {10, 20, 30, 40, 50}. Результат слияние записывается в переменную с. Ожидается, что с будет содержать массив элементов {1, 2, 3, 4, 5, 10, 20, 30, 40, 50}, но результатом стала строка, содержащая текст «1,2,3,4,510,20,30,40,50».

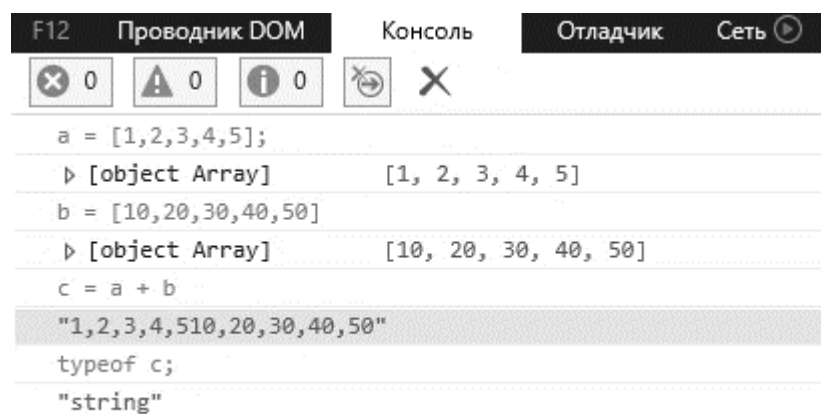


Рисунок 3 — слияние массивов в JavaScript.

Так как массив в JavaScript является объектом класса, то у него есть ряд встроенных методов, в том числе сортировки. Однако во время процесса сортировки элементы преобразовываются в строковый тип, что приводит к неверному результату. Пример такого представлен на рисунке 4.

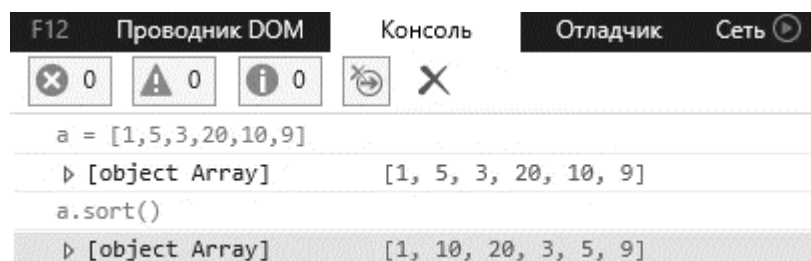


Рисунок 4 —сортировка массива в JavaScript.

1.4.4 Язык программирования Java

Java – сильно типизированный объектно-ориентированный язык программирования. В настоящее время разработкой занимается компания Oracle. Основной целью создания языка является обеспечения Кроссплатформенность между разными устройствами с разными операционными системами, тем самым исключив дорогостоящий процесс переноса одной и той же программы на разные платформы. Обеспечивается это компиляции кода программы в байт-код, который выполняется на виртуальной машине Java, установленной на целевой платформе. Виртуальная машина Java в свою очередь является интерпретатором, транслирующий байт-код программы в машинный код целевой системы. В настоящее время Java является одним из самых популярных языков программирования.

1.4.5 Итог

В качестве языка программирования был выбран язык Java. Интерпретируемые языки программирования не обеспечивают должный уровень производительности, поэтому Python и JavaScript не подходят. В отличии от Java, язык Go находится на стадии активного развития, а данным языком владеют малое количество разработчиков, что усложняет поиск сотрудников для будущей поддержки и расширения функционала разрабатываемого модуля.

1.5 Выбор СУБД

В данном разделе будут разобраны самые популярные СУБД, будут представлены их достоинства и недостатки, исходя из которых будет выбран оптимальная.

1.5.1 MS SQL Server

Microsoft SQL Server – разработанная корпорацией Майкрософт. Основным языком для написания запросов является Transact-SQL. Данная СУБД подходит как для создания персональных баз данных, так и для реализации БД масштаба компании. Хорошо подходит для проектов, использующие другие продукты Майкрософт.

Достоинства СУБД MS SQL Server:

- возможность установить разные версии СУБД на одной машине;
- возможность регулировки и отслеживания уровня производительности в целях оптимизации;
- возможность интеграции с другими продуктами Microsoft;
- возможность визуализации на мобильных устройствах.

Недостатки СУБД MS SQL Server:

- очень высокая стоимость продукта для юридических лиц;
- возможны проблемы в работе служб интеграции импорта файлов;
- высокая ресурсоемкость SQL Server (даже при настройке производительности);
- зависимость от платформы.

Данная СУБД не подходит по причине высокой цены коммерческой версии и малого количества поддерживаемых платформ.

1.5.2 Oracle Database

Oracle Database – объектно-реляционная система управления базами данных, разработанная компанией Oracle. Система популярна у разработчиков, проста в использовании, у нее понятная документация,

поддержка длинных наименований, JSON, улучшенный тег списка и Oracle Cloud.

Достоинства СУБД Oracle Database:

- высокий показатель надежности;
- современный функционал и новые разработки, доступные в последней версии;

Недостатки СУБД Oracle Database:

- высокая стоимость продукта для юридических лиц;
- для системы необходимы аппаратные ресурсы, обладающие достаточной мощностью и производительностью;

Как и с MS SQL, стоимость коммерческой версии не позволяет использовать данную СУБД, бесплатная версия позволяет развертывать небольшие по объему БД.

1.5.3 MySQL

MySQL – это самая популярная из всех крупных серверных БД. СУБД проста в установке и эксплуатации, в сети есть вся необходимая информация для работы с ней. MySQL и не пытается полностью реализовать SQL-стандарты, однако предлагает широкий функционал. Приложения общаются с базой данных через процесс-демон.

Достоинства MySQL:

- бесплатная для личного использования;
- простая установка и обладает огромным числом сторонних инструментов для работы с СУБД;
- безопасность;
- возможность использования множества функций в бесплатной версии;
- высокая производительность.

Недостатки MySQL:

- проблемы с надежностью;
- платная для коммерческого использования;
- некоторые ограничения функционала;
- проект поддерживается сообществом open-source, что сказывается на скорости выхода стабильной версии и скорости разработки в целом.

Данная СУБД обладает рядом ограничений функциональности, а также в случае нахождения критической ошибки в СУБД новая стабильная версия с исправлениями может выйти не скоро.

1.5.4 PostgreSQL

PostgreSQL – это самая продвинутая СУБД, ориентирующаяся в первую очередь на полное соответствие стандартам и расширяемость. PostgreSQL пытается полностью соответствовать SQL-стандартам ANSI/ISO. PostgreSQL отличается от других СУБД тем, что обладает объектно-ориентированным функционалом, в том числе полной поддержкой концепта ACID.

Достоинства PostgreSQL:

- бесплатная версия обладает достаточным функционалом для коммерческого использования;
- хорошо масштабируется;
- расширяемость функционала за счет создания хранимых процедур;
- высокая надёжность;
- существует множество готовых встроенных функций;
- подробная документация.

Недостатки PostgreSQL:

- требователен к ресурсам;

- бесплатная версия не имеет ряда функций, которые доступны в коммерческой, например, выполнение задач по расписанию;
- медленнее выполняет конкурентных СУБД.

1.5.5 Итог

Из всех представленных ранее СУБД для решения задачи больше всего подходит PostgreSQL. Данная СУБД обладает высокой степенью надёжности и поддерживает концепт ACID, бесплатная версия СУБД имеет необходимые функции для работы микросервиса.

В проекте используется версия 12, на момент начала разработки она была выбрана как наиболее стабильная.

1.6 Вывод

Основываясь на результатах проведенного анализа, было принято решение разработать собственное приложение на микросервисной архитектуре, которая может решить поставленные ранее задачи.

В качестве основного языка программирования был выбран язык Java, причиной того стало достаточно высокая производительность и большое количество фреймворков с широким функционалом, такие как Spring, Hibernate и т.д..

В качестве СУБД будет использоваться PostgreSQL т.к. бесплатная версия СУБД обеспечивает высокую степень надёжности и поддерживает концепт ACID.

2 Реализация

В данной работе приведены результаты разработки микросервиса синхронизации данных между центральным кластером ИС и множества КПС. Функциональное назначение данного модуля – обеспечения эффективного двунаправленного обмена данными в рамках поставленной задачи.

Разработку модуля можно разделить на следующие этапы:

- Разработка протокола обмена;
- разработка структуры БД;
- разработка программы.

2.1 Описание протокола

Взаимодействие осуществляется по схеме запрос-ответ. Инициатором запроса является КПС. Протокол взаимодействия – HTTPS (с предоставлением только серверного сертификата). Спецификация запроса:

- HTTP method: POST;
- request: транспортный конверт (MIME: text/xml);
- response: транспортный конверт (MIME: text/xml);
- timeout: 3 минуты.

Протокол предусматривает целостность данных в случае потери/частичной обработки пакета на прикладном уровне с помощью процедур подтверждения обработки каждой стороной.

2.1.1 Структура пакета.

Регистр символов, если не указано иное, важен. Порядок следования элементов не имеет значения. Для строковых значений указана максимальная длина, гарантировано воспринимаемая участниками взаимодействия. Для числовых значений указана точная размерность в битах, гарантированно воспринимаемая всеми участниками взаимодействия. Суммы и количества, если не указано иное, должны быть неотрицательными значениями. Все поля, если не указано иное, обязательны. Если указано, что поле является

необязательным, то для его исключения необходимо целиком исключить поле и его значение, т.е. «пустое» значение не является указателем того, что поле не задано.

Для прозрачности работы модуля, вводится числовой идентификатор пакета PacketId, PacketId должен увеличиваться на единицу при создании каждого нового пакета. Он должен вестись отдельно для всех КПС. Значение PacketId для первого пакета – 0.

В транспортном пакете передается информация об ОО-источнике, с которой производится процедура синхронизации, и необходимые секции. Состав секций варьируется от типа синхронизации. В таблице 1 представлен состав пакета.

Таблица 1 — Структура транспортного пакета.

Название поля	Описание	Тип данных
OrgID	Идентификатор ОО (уникален в системе)	Int64
PacketID	Идентификатор пакета (уникален для КПС)	Int64
Version	Версия протокола	Int64
Date	Дата и время формирования пакета	DateTime
Org	Наименование организации	Str(256)
ClientVersion	Номер версии клиентской программы	Str(16)
Type	Тип синхронизации	Int
DirectorName	ФИО руководителя ОО	Str(256)
PaymentRegistry	Реестр платежей и покупок	Element
OrganizationStructure	Структура организации	Element
AccountsRegistry	Реестр учетных записей	Element
PassageEvents	Журнал событий системы доступа на территорию	Element
OrgSettings	Передаваемые параметры клиентской программы	Element
Log	Журнал ошибок и системных оповещений	Element
Signature	Цифровая подпись пакета	Element

Таблица 2 содержит описание структуры элемента «PaymentRegistry». В «PaymentRegistry» передается информация об платежах и покупках, совершаемые клиентами внутри ОО. Информация передается однонаправленно от КПС к серверу.

Таблица 2 — Структура элемента «PaymentRegistry».

Название поля	Описание поля	Тип данных
OrderID	Идентификатор платежа (уникален для учреждения)	Int64
ClientID	Идентификатор клиента (уникален в системе). Если указан номер карты, то это поле опционально	Int64
CardNum	Номер карты (уникален в системе). Если сумма, оплаченная картой, равна 0, то это поле опционально.	Int64
Date	Дата и время покупки	DateTime
OrderType	Тип покупки	Int32
Sum	Общая сумма заказа с учетом скидки (за минусом скидки)	Int32
Discount	Скидка	Int32
CashierID	Идентификатор кассира	Int64
State	Статус заказа	Int32
ProductList	Список купленных товаров	Element

В «State» задается флаг стонирования заказа. Параметр «OrderType» опционален и используется в некоторых ОО для уточнения типа покупки: вендинг, буфет, оплата регулярного обеда.

Таблица 3 содержит описание структуры элемента «ProductList», содержащийся в элементе «PaymentRegistry».

Таблица 3 — Структура элемента «ProductList».

Название поля	Описание поля	Тип данных
OrderDetailsID	Идентификатор элемента покупки (уникален для учреждения)	Int64
Qty	Количество	Int32
Price	Цена за единицу с учетом скидки (за минусом скидки)	Int32
Discount	Скидка (на единицу)	Int32
Name	Наименование товара	Str(256)
Code	Код товара	Str(32)
ProductUUID	UUID товара	Str(36)

Таблица 4 содержит описание структуры элемента «OrganizationStructure».

Таблица 4 — Структура элемента «OrganizationStructure».

Название поля	Описание поля	Тип данных
OrgId	Идентификатор ОО	Int64
OrgType	Тип ОО	Int32
Name	Название ОО	Str(256)
Address	Адрес	Str(256)
Version	Версия записи	Int64
State	Признак архивности	Int32

Таблица 5 содержит описание структуры элемента «AccountsRegistry».

Таблица 5 — Структура таблицы «AccountsRegistry».

Название поля	Описание поля	Тип данных
ClientID	Идентификатор записи	Int64
Version	Версия записи	Int64
FirstName	Имя	Str(256)
Patronymic	Отчество	Str(256)
Surname	Фамилия	Str(256)
GroupName	Название группы клиента (подразделение)	Str(256)
AccountType	Тип аккаунта	Int32
DocumentID	Идентификатор документа	Int64
Phone	Номер телефона	Int32
Email	E-Mail	Str(256)
ContractId	Номер договора	Int64
Gender	Пол	Int32
ClientUUID	UUID клиента	Str(36)

Таблица 6 содержит описание структуры элемента «PassageEvents».

Таблица 6 — Структура элемента «PassageEvents».

Название поля	Описание поля	Тип данных
PassageEventsID	Идентификатор записи	Int64
EnterName	Наименование входной группы	Str(256)
TurnstileUUID	UUID турникета	Str(36)
PassCode	Код события	Int32
CardID	Идентификатор карты	Int64
ClientID	Идентификатор клиента	Int64
EvtDate	Дата и время события	DateTime

В параметре «PassCode» элемента «PassageEvents» передается код события. Таблица 7 содержит расшифровки кодов.

Таблица 7 — Коды «PassCode».

Код	Рисшифровка
101	Проход на территорию ОО
102	Выход с территории ОО
201	Повторный проход
202	Повторный выход
301	Проход на территорию оператором пропускного пункта
302	Выход с территории оператором пропускного пункта
400	Обнаружен на территории при совершении операции по карте
500	Проход запрещен

В элементе «OrgSettings» передаются настройки, необходимые для наладки работы КПС, например расписание обращений, директиву на регистрацию определенных событий и прочее.

Таблица 8 содержит описание структуры элемента «OrgSettings».

Таблица 8 — Структура элемента «OrgSettings».

Название поля	Описание поля	Тип данных
OrgID	Идентификатор ОО	Int64
Version	Версия записи	Int64
SettingGroup	Группа настроек	Integer
GlobalUUID	Глобальный UUID	Str(36)
SettingCode	Код настройки	Int32
Value	Значение	Str(256)

Таблица 9 содержит описание структуры элемента «Log».

Таблица 9 — Структура таблицы «Log».

Название поля	Описание поля	Тип данных
EvtTime	Дата и время события	DateTime
Severity	Критичность	Int32
Code	Код события/группы событий	Int32
Text	Текст сообщения	Str(2048)

Листинг XML-файла транспортного пакета данных представлен в приложении В.

2.1.2 Обеспечение безопасности.

Для каждого пакета КПС генерирует ЭЦП и подписывает пакет по стандарту W3C XML signature. Алгоритм шифрования RSA-SHA256. Открытый ключ хранится у КПС и на центральном сервере. КПС с пакетом высылает SSL-сертификат клиента, выданный внутренним удостоверяющим центром ИС.

При обработки транспортного пакета, сервер по ID ОО и по SSL-сертификату проводит верификацию подписи. В случае успешной проверки начинается процедура синхронизации.

2.2 Структура БД

Далее представлены таблицы, с которыми работает ИС.

Таблица 10 содержит описание структуры таблицы «Payments». В таблицу «Payments» попадают покупки из секции «PaymentsRegistry» транспортного пакета, полученный от КПС. Таблица связана с таблицей «PaymentItems» по внешнему ключу, тип связи «один-ко-многим».

Таблица 10 — Структура таблицы «Payments».

Название поля	Описание поля	Тип данных
PaymentID	Идентификатор платежа	BigInteger
OrgID	Идентификатор ОО	BigInteger
ClientID	Идентификатор клиента	BigInteger
CardID	Идентификатор карты	BigInteger
Date	Дата и время покупки	Timestamp
OrderType	Тип покупки	Integer
Sum	Общая сумма заказа с учетом скидки	Integer
Discount	Скидка	Integer
CashierID	Идентификатор кассира	BigInteger
State	Статус заказа	Integer

Таблица 11 содержит описание структуры таблицы «PaymentItems», в данной таблице хранится список покупок по чеку из таблицы «Payments».

Таблица 11 — Структура таблицы «PaymentItems».

Название поля	Описание поля	Тип данных
ItemID	Идентификатор элемента покупки (уникален для учреждения)	BigInteger
PaymentID	Идентификатор платежа, к которому привязан элемент	BigInteger
Qty	Количество	Integer
Price	Цена за единицу с учетом скидки (за минусом скидки)	Integer
Discount	Скидка (на единицу)	Integer
Name	Наименование товара	VarChar(256)
Code	Код товара	VarChar(32)
ProductUUID	UUID товара	VarChar(36)

Таблица 12 содержит описание структуры таблицы «Organizations». Таблица «Organizations» содержит информацию об ОО, в которых установлен КПС. Большинство таблиц, в том числе «AccountsRegistry», ссылаются типом связи «Один-ко-многим».

Таблица 12 — Структура таблицы «Organizations».

Название поля	Описание поля	Тип данных
OrgId	Идентификатор ОО	BigInteger
DirectorID	Идентификатор клиента – руководителя ОО	BigInteger
OrgType	Тип ОО	Integer
Name	Название ОО	VarChar(256)
Address	Адрес	VarChar(256)
Version	Версия записи	BigInteger
State	Признак архивности	Integer
ContractID	Идентификатор договора на обслуживание	VarChar(40)
Phone	Номер телефона	VarChar(11)
City	Город	VarChar(256)

Таблица 13 содержит описание структуры элемента «AccountsRegistry». Данная таблица хранит информацию об всех клиентов, которые числятся за определенной ОО. Таблица хранит персональные данные и контактную информацию клиентов. Связана с таблицами покупок и проходов на территорию связью «Один-ко-многим».

Таблица 13 — Структура таблицы «AccountsRegistry».

Название поля	Описание поля	Тип данных
ClientID	Идентификатор записи	BigInteger
OrgID	Идентификатор ОО, к которому привязан клиент	BigInteger
Version	Версия записи	BigInteger
FirstName	Имя	VarChar(256)
Patronymic	Отчество	VarChar(256)
Surname	Фамилия	VarChar(256)
GroupName	Название группы клиента (подразделение)	VarChar(256)
AccountType	Тип аккаунта	Integer
DocumentID	Идентификатор документа	BigInteger
Phone	Номер телефона	Integer
Email	E-Mail	VarChar(256)
ContractId	Номер договора	BigInteger
ClientUUID	UUID клиента	VarChar(36)
Gender	Пол	Integer

Таблица 14 содержит описание структуры таблицы «PassageEvents».

Таблица 14 — Структура таблицы «PassageEvents».

Название поля	Описание поля	Тип данных
PassageEventsID	Идентификатор записи	BigInteger
OrgID	Идентификатор ОО - источника	BigInteger
EnterName	Наименование входной группы	VarChar(256)
TurnstileUUID	UUID турникета	VarChar(36)
PassCode	Код события	Integer
CardID	Идентификатор карты	BigInteger
ClientID	Идентификатор клиента	BigInteger
EvtDate	Дата и время события	Timestamp

В таблице «OrgSettings» и «OrgSettingsItems» хранятся настройки для КПС. КПС можно настроить непосредственно на территории ОО, права на изменение настроек имеет прикрепленный администратор, но также КПС может быть настроен инженером через WEB-интерфейсе центрального сервера ИС. Таблица «OrgSettings» является связующей для указания к какой ОО принадлежат настройки с указанием группы настроек. Сами значения хранятся в таблице «OrgSettingsItems». Так же таблица «OrgSettings»

используется для хранения настроек автоматического распределения обращений с указанием расписания обращений.

Таблица 15 содержит описание структуры таблицы группы настроек «OrgSettings».

Таблица 15 — Структура таблицы «OrgSettings».

Название поля	Описание поля	Тип данных
OrgSettingsID	Идентификатор записи	BigInteger
OrgID	Идентификатор ОО	BigInteger
SettingGroup	Группа настроек	Integer

Таблица 16 содержит описание структуры таблицы значений настроек «OrgSettingsItems».

Таблица 16 — Структура таблицы «OrgSettingsItems».

Название поля	Описание поля	Тип данных
OrgSettingsItemsID	Идентификатор записи	BigInteger
OrgSettingsID	Идентификатор группы настроек	BigInteger
GlobalUUID	Глобальный UUID	VarChar(36)
SettingCode	Код настройки	BigInteger
Value	Значение	VarChar(256)
Version	Версия записи	BigInteger

Таблица 17 содержит описание структуры таблицы «Logs». Данная таблица необходима для формирования отчета об работе КПС.

Таблица 17 — Структура таблицы «Logs».

Название поля	Описание поля	Тип данных
RecordID	Идентификатор записи	BigInteger
EvtTime	Дата и время события	Timestamp
OrgID	Идентификатор ОО	BigInteger
CSSVersion	Версия КПС	VarChar(32)
Severity	Критичность	Integer
Code	Код события/группы событий	Integer
Text	Текст сообщения	VarChar(2048)

Для прозрачности работы процедуры синхронизации введется запись таблицы «SyncHistory». Данная таблица содержит информацию о КПС-инициаторе и успешности проведения сеанса синхронизации.

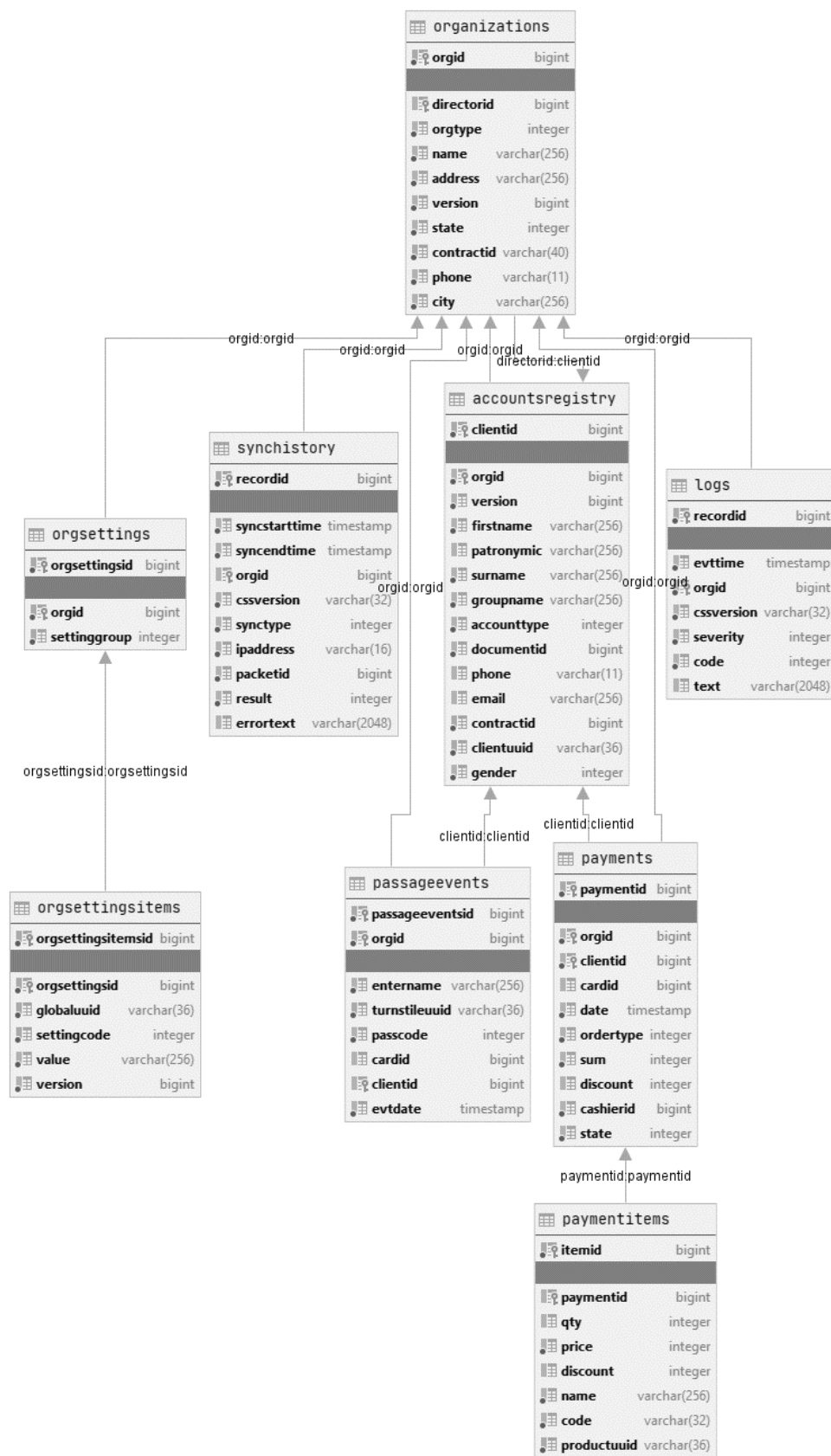
Таблица 18 содержит описание структуры таблицы «SyncHistory».

Таблица 18 — Структура таблицы «SyncHistory».

Название поля	Описание поля	Тип данных
RecordID	Идентификатор записи	BigInteger
SyncStartTime	Время начала процедуры синхронизации	Timestamp
SyncEndTime	Время конца процедуры синхронизации	Timestamp
OrgID	Идентификатор ОО	BigInteger
CSSVersion	Версия КПС	VarChar(32)
SyncType	Тип синхронизации	Integer
IPAddress	IP-адрес клиента	Integer
PacketID	ID обрабатываемого пакета	BigInteger
Result	Результат обработки	Integer
ErrorText	Текст ошибки	VarChar(2048)

Листинг SQL-файла для создания таблиц представлен в приложении С.

На рисунке 5 демонстрируется диаграмма таблиц в БД.



Powered by yFiles

Рисунок 5 – Схема БД

2.3 Описание классов

Для взаимодействия с КПС и проведения процесса синхронизации были созданы:

- класс-контроллеры – обрабатывает запросы от КПС;
- классы-сущности – отвечают за удобное взаимодействия с реляционными моделями путем представления их в виде объекта.
- транспортные классы – обеспечивают конвертацию данных из XML-документа в объект или обратно в XML-документ для передачи в КПС;
- классы-сервисы – выполняют основную часть функционала обработки пакета. Каждый класс-сервис обрабатывает свою секцию.

Для более удобной работы с данными, находящимся в XML-документе, был разработан транспортные класс, в полях которого находятся необходимые данные. Класс назван «SyncRequest», его UML диаграмма классов представлена на рисунке 6.

Точкой входа на обработку является класс-контроллер «SyncController». Данный класс осуществляет проверку данных в пакете и конвертацию XML-документ в экземпляр транспортного класса-запроса. UML диаграмма класса «SyncController» представлен на рисунке 7.

Обработка данных осуществляется через классы-сервисы. Классом, в котором сосредоточена основная логика, является «Processor». После всей обработки «Processor» возвращает объект класса-ответа для передачи данных в КПС. UML диаграмма класса «Processor» представлен на рисунке 8.

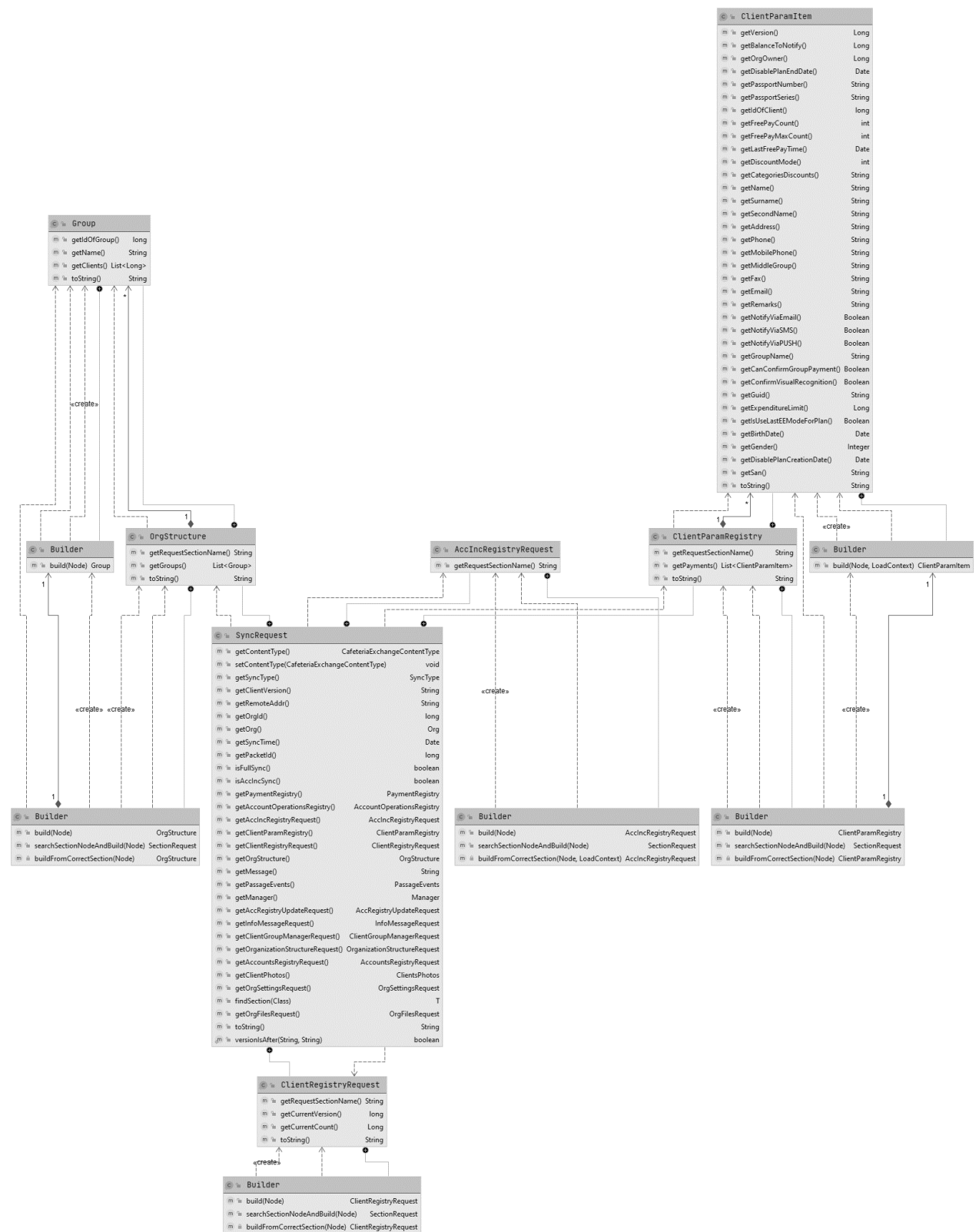


Рисунок 6 — UML диаграмма транспортных классов для упаковки запроса

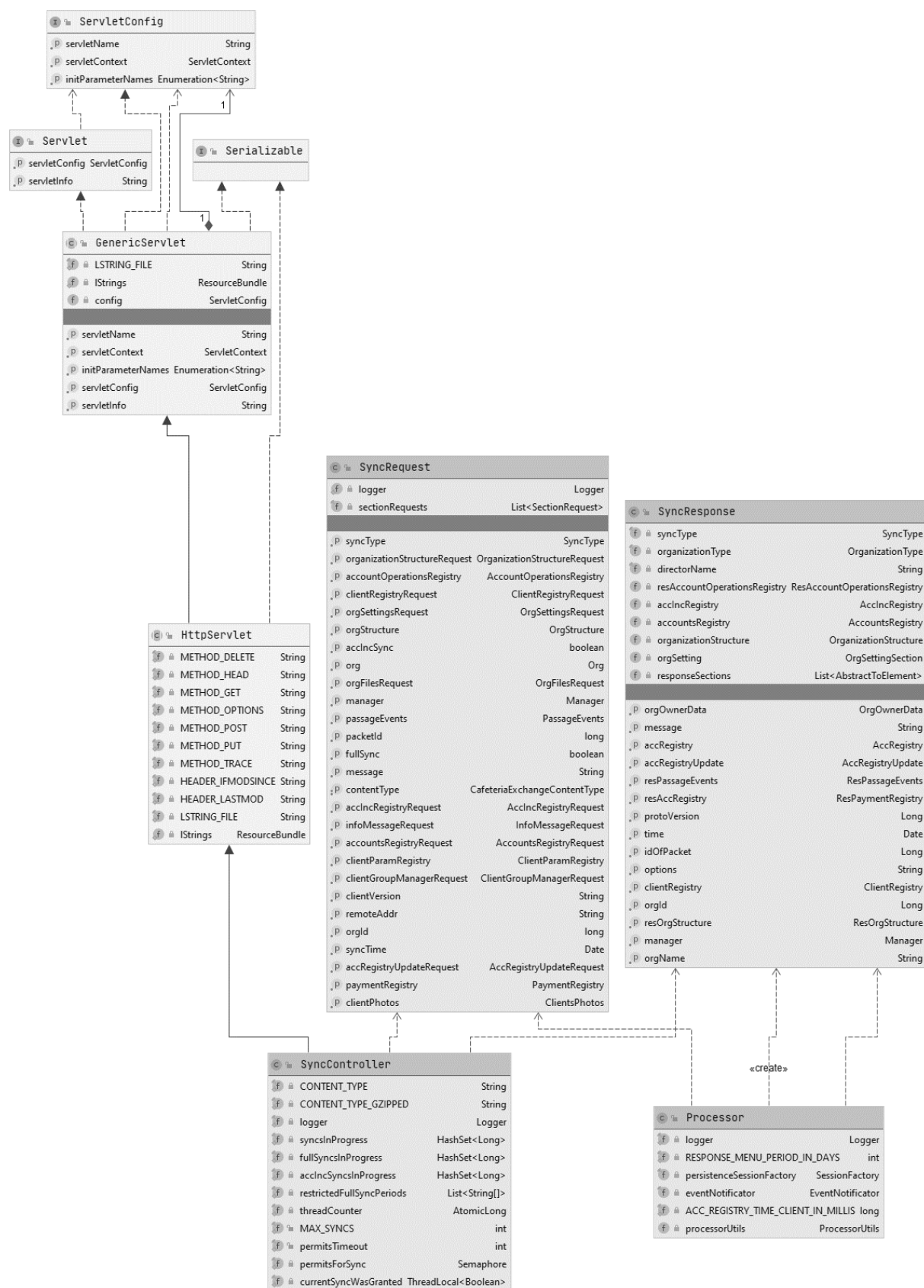


Рисунок 7 — UML диаграмма классов контроллера

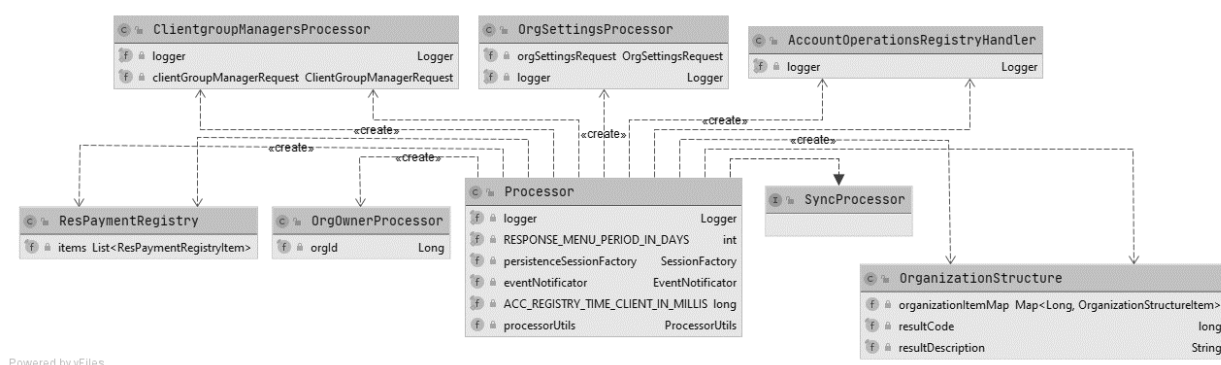


Рисунок 8 — UML диаграмма классов-сервисов

Как было сказано ранее, результатом обработки является транспортный класс-ответ, который будет конвертирован в XML-пакет для передачи в КПС. Класс был назван «SyncResponse», его UML-диаграмма представлена на рисунке 9.

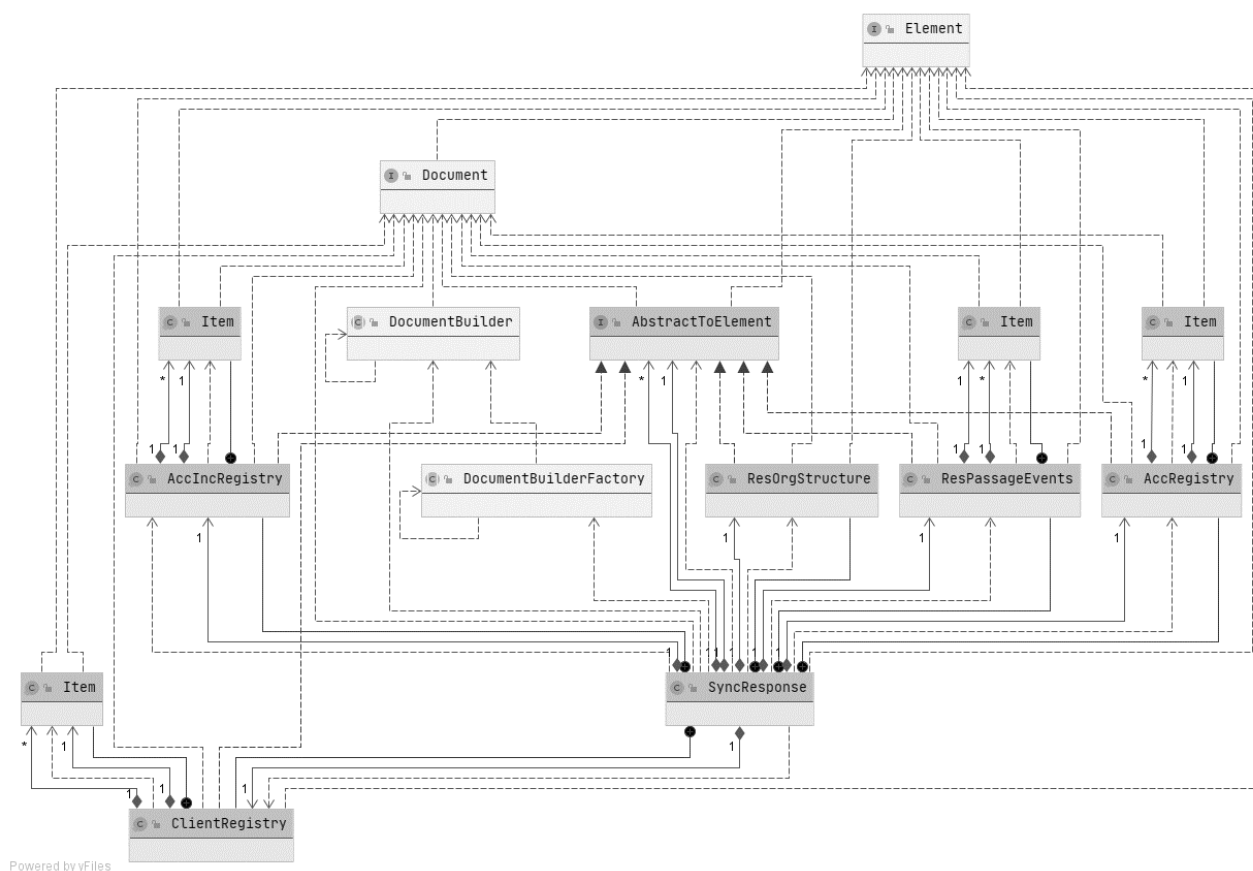


Рисунок 9 — UML диаграмма транспортных классов для хранения ответа

2.4 Программная реализация

Как упоминалось в предыдущих разделах, для реализации данного микросервиса используется язык программирования Java и фреймворк Spring. Процесс синхронизации начинается с обращения КПС к классу-контроллеру, далее происходит проверка наличия SSL-сертификата в запросе и верификация ЭЦП в транспортном пакете. Далее XML-документ обрабатывается и из него создается экземпляр транспортного класса-запроса, данные объект отправляется на обработку в классы-сервисы. Результатом обработки является экземпляр транспортного класса-ответа, поля которого преобразовываются в XML-документ, отправляемый в ответе.

На рисунке 10 показана диаграмма последовательности процесса синхронизации.

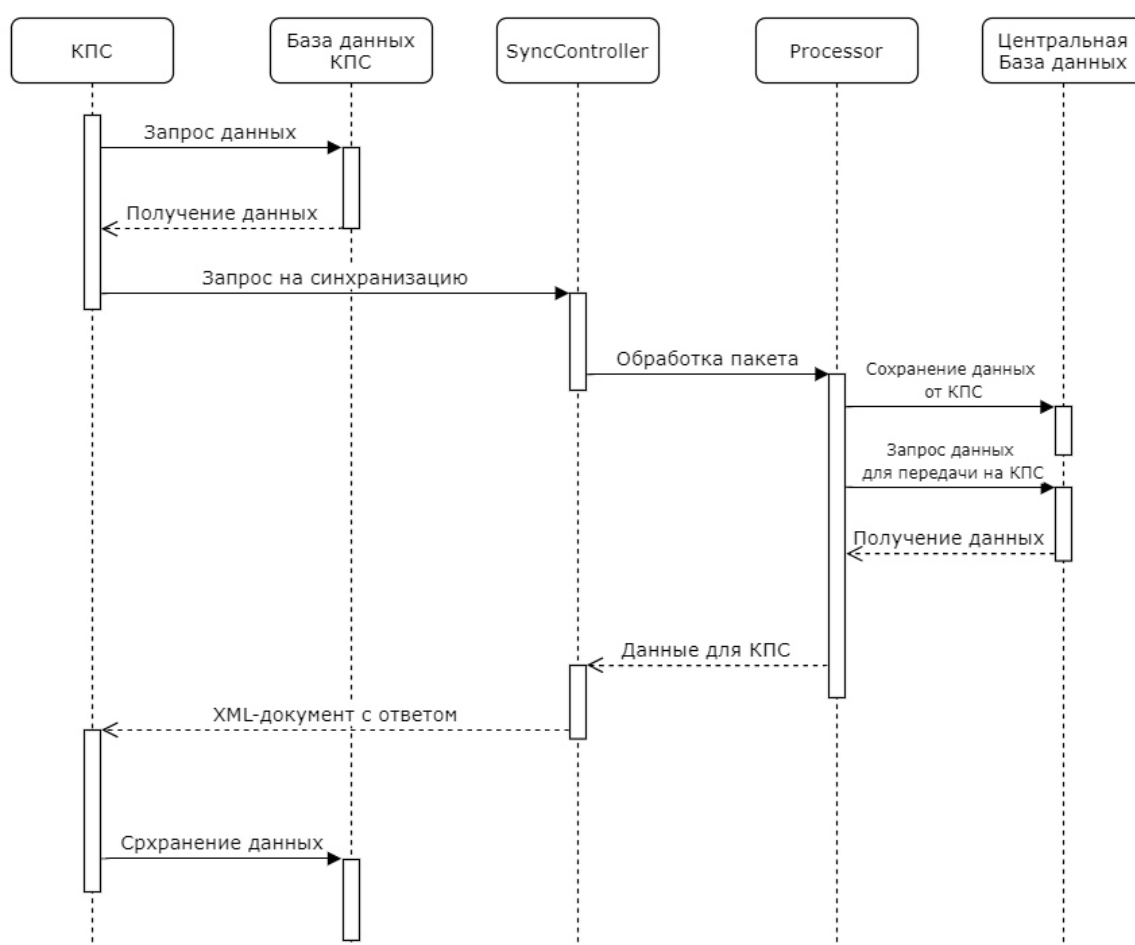


Рисунок 10 — Диаграмма последовательности процесса синхронизации

Для отслеживания изменений в сущностях было введено понятие версии. У каждой обрабатываемой сущности есть поле «Version», которая отображает актуальность объекта. Во время синхронизации в секции указывается максимальная версия сущности в данной КПС, в этом случае микросервис запросит у БД все сущности с версией, выше чем у КПС, и передаст их в ответе. Если версия совпадает с версией от КПС, то микросервис перезаписывает данную запись в БД. Инкрементацию версии может производить только центральный сервер, КПС изменяют сущности без подъема версии. Это сделано для обеспечения приоритета изменений у сервера перед КПС. Если запись была принята сервером, то КПС получает только ID записи и новую версию сущности, в ответе будет запись с изменёнными данными на стороне сервера.

Исходный код на языке Java класса-сущности «Пользователь», класса-контроллера, сервиса и репозитория находится в приложении А.

2.5 Вывод

В ходе проектирования микросервиса синхронизации был разработан протокол взаимодействия между сервером и КПС и был рассмотрен вопрос обеспечения безопасности целостности данных при взаимодействии.

Так же была разработана структура XML-документа, передаваемый в транспортном пакете, спроектированы сущности в БД для удобного хранения данных, а также спроектированы классы для взаимодействия с клиентом и обработки пакетов.

3 Описание результата

3.1 Описание микросервиса

В результате компиляции кода микросервиса был получен исполняемый файл «syncmodule.jar». Данный микросервис можно запустить любой машине, важным требованием является доступность данной машины для КПС и доступность БД для самого микросервиса. Данные для подключения к БД находятся в конфигурационном файле микросервиса. В случае необходимости возможно запустить сервис с новыми значениями для подключения к БД, указав параметры в аргументах командной строки:

- --DBaddress — указание нового адреса подключения к БД;
- --DBlogin — указание нового имени пользователя для идентификации в СУБД;
- --DBpass — указание нового пароля для аутентификации в СУБД;

Для отладки микросервиса синхронизации была разработана программа-эмулятор, позволяющая проводить тесты. Интерфейс клиентской программы представлен на рисунке 11.

Рисунок 11 — интерфейс программы для тестирования микросервиса.

Интерфейс программы можно разделить на 3 основные секции:

- секция для задания параметров динамического генерирования транспортных пакетов и количество экземпляров клиента (управляется отдельными нитями), а также общее количество сгенерированных пакетов;
- секция с адресом сервера для обращения, элементы управления;
- указание пути к готовому пакету и пути к SSL-сертификату.

Программа позволяет производить единовременный сеанс синхронизации с сервером, создав необходимое количество соединений и сгенерировав указанное количество пакетов.

Как было сказано ранее сервер после обработки формирует ответный пакет с XML-документом.

В документе прописывается дата формирования ответного документа и имя узла, который обработал пакет. Имя указывается для удобства поиска журналов для просмотра деталей ошибок и прочего.

Для сущностей-событий указывает ID последней обработанной записи, для распределенных объектов указывается новая версия с ID, если сервер принял изменения от КПС иначе сервер высылает новые данные с новой версией.

Пакет так же подписывается ЭЦП для обеспечения безопасности.

Пример листинга XML-документа, который является ответом на документ из приложения В, представлен ниже.

```
<?xml version="1.0" encoding="utf-8"?>
<Data>
  <Body>
    <DataSync Date="02.06.2020 12:24:21" OrgId="503"
PacketId="112" OrgType="1" NodeName="3" Type="1">
      <PaymentRegistry LastID="28494"/>
      <OrganizationStructure>
        <OS OrgId="503" Version="24"/>
        <OS OrgId="504" Version="25"/>
        <OS OrgId="505" OrgType="0" Name="Rename by Server"
Address="Some Address" Version="26" State="0"/>
      </OrganizationStructure>
      <AccountsRegistry>
        <AR ClientID="1269500" Version="44"/>
        <AR ClientID="1269501" Version="45"/>
      </AccountsRegistry>
      <PassEvents LastID="498"/>
      <OrgSettings>
        <OS OrgID="503" Version="1126"/>
        <OS OrgID="503" SettingGroup="9" Version="1127"
SettingCode="1301" Value="04:31;14:31" GlobalUUID="781fef0b-
efc3-440c-b96b-5a86402194bd"/>
      </OrgSettings>
    </DataSync>
```

```

</Body>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
      <SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha256" />
      <Reference URI="">
        <Transforms>
          <Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature" />
        </Transforms>
        <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <DigestValue>HV7uPN+CBCMdh3B8NOA4QFPvvm4=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>exOiZqHe09ph/XCXYx+ucnvgAXpbKZFD+HRlOlDLWAD
/zpX6/GIGMXQZbiXfl5lhMfo4JdXjg81BR4d02m86j46u7i2ytEvmF+SSjgl7/u
zEzVRRX5Z2Nn6F383PEkkaeMkhP/Xn4m5RwbMlmEFIUGykNLs+wTczsx2wbMqME=
</SignatureValue>
  </Signature>
</Data>

```

При сравнении данных из входного и выходного документа видно, что сервер принял все событийные данные об проходах и покупка. Отчасти принял данные по секции «OrganizationStructure», однако одна запись была изменена на стороне сервера. Та же ситуация и секцией «OrgSettings», одна из настроек расписания была изменена сервисом автораспределения.

3.2 Описание сервиса автораспределения.

В ходе тестирования микросервиса, было выявлено критическое значение обращений КПС в утренние и вечерние часы, которые вызывали высокие нагрузки на сервер и падение производительности всей ИС. Данная ситуация связано с регистрацией событий проходов клиентами на территорию обслуживаемого ОО и разовое обращение от КПС с отчетом информирования об продаже товаров.

Для решения данной проблемы, в первую очередь, сама процедура синхронизации была разделена на логические типы, где участвуют только

некоторые секции из имеющихся. Данный шаг позволил настроить периодичность передачи данных и оптимизировать время сеанса синхронизации для каждого из логических типов синхронизации. Однако всё еще существует необходимость полной синхронизации, где должны участвовать все секции. Как правило полная синхронизация производится при первичном заполнении БД КПС, при вводе новой сущности или нового атрибута у старых для установки значений по умолчанию или в случае ошибок со стороны оператора при ручном вводе. Разделение на синхронизации на типы не позволяет полностью снять проблему единовременного обращения.

Что бы избавиться от единовременного обращения КПС, было решено создать модуль автоматического распределения времени синхронизации, обеспечив тем самым гибкое расписание обращений.

Прежде всего были определены понятия разрешенного и запрещенного времени обращения. Разрешенный временной промежуток устанавливается для явного указания оптимального времени синхронизации. Иначе указывается только запрещенный промежуток для всех клиентов, в который не должен проводиться сеанс синхронизации. Разделение промежутков на запрещенные и разрешенные гарантирует выделение удобного времени для технического обслуживания ИС, например, создание бэкапа БД в ночное время.

На WEB-интерфейсе ИС инженером задается временной промежуток выражением, которое представляет собой последовательность временных интервалов. Выражение должно иметь вид НН:мм-НН:мм. Перечисление интервалов задается знаком точки с запятой (;). Для указания периода, когда синхронизация запрещена, перед интервалом необходимо поставить восклицательный знак (!). Количество разрешенных промежутков указывает

на количество необходимых сеансов синхронизации, время каждого сеанса должно укладываться в соответствующий временной интервал.

При указании запрещающих выражений, сеанс синхронизации будет распределен на все 24 часа за исключением указанных интервалов.

Процедура автоматического распределения начинается с расчета базового времени для каждой клиентской подсистемы, оно рассчитывается по формуле 1.

$$T_b = ID_o/k \quad (1)$$

T_b – базовое время;

ID_o – числовой идентификатор организации в центральном сервере;

k – коэффициент распределения базового времени.

Коэффициент распределения базового времени k свой для каждого типа синхронизации и задавался из допустимой плотности сеансов синхронизации по времени. Чем меньше коэффициент, тем больше сдвиг по времени получала каждая организация на проведение синхронизации.

Расчет итогового времени для разрешенных временных промежутков рассчитывается по формуле 2.

$$T_{si} = \left\lfloor \frac{P_{si} + T_b}{P_{ei}} \right\rfloor \quad (2)$$

T_{si} – итоговое время запуска сеанса синхронизации для i -того разрешенного интервала, указанное в минутах от начала дня;

P_{si} – время начала разрешенного периода для i -го сеанса;

P_{ei} – время конца разрешенного периода для i -го сеанса.

В случае указания запрещенных промежутков, происходит проверка вхождения T_b в эти самые промежутки. В случае, если T_b попадает в запрещённый промежуток, то происходит перерасчет по формуле 3.

$$T_s = \left\lceil \frac{P_{ei} + T_b}{24 * 60} \right\rceil \quad (3)$$

P_{ei} – время конца i -го запрещенного для синхронизации периода;

T_s – новое рассчитанное время для запуска синхронизации.

В случае, если T_s попадает в следующий запрещенный период, то происходит его повторный перерасчет по формуле 3.

На расчет T_s выделено ограниченное количество попыток, в случае если было задано некорректное выражение. По истечению количества попыток происходит сброс указанных выражений и установка выражений по умолчанию. Далее происходит регенерация T_s проблемного типа синхронизации уже по новому выражению.

В ходе ввода сервиса автораспределения, были собраны данные нагрузочного тестирования микросервиса для оценки эффективности метода.

Для сравнения далее приведены графики количества запросов на сеанс синхронизации до и после ввода модуля.

На рисунке 12 приведен графики работы ИС в один полный день. График, изображенный сплошной линией – график до ввода модуля. Весь массив запросов пришелся в временной промежуток от начала и до конца рабочего дня в ОО. Всего было обработано 33445 пакетов данных. Была промоделировано генерация событий проходов на территорию ОО, из-за чего пик пришелся на 7 часов утра.

График, изображённый пунктирной линией - график после ввода автоматического распределения. Было обработано 33835 пакетов данных. Наблюдается распределение нагрузки на ночное время, а также сниженные показатели пиков в дневные часы.

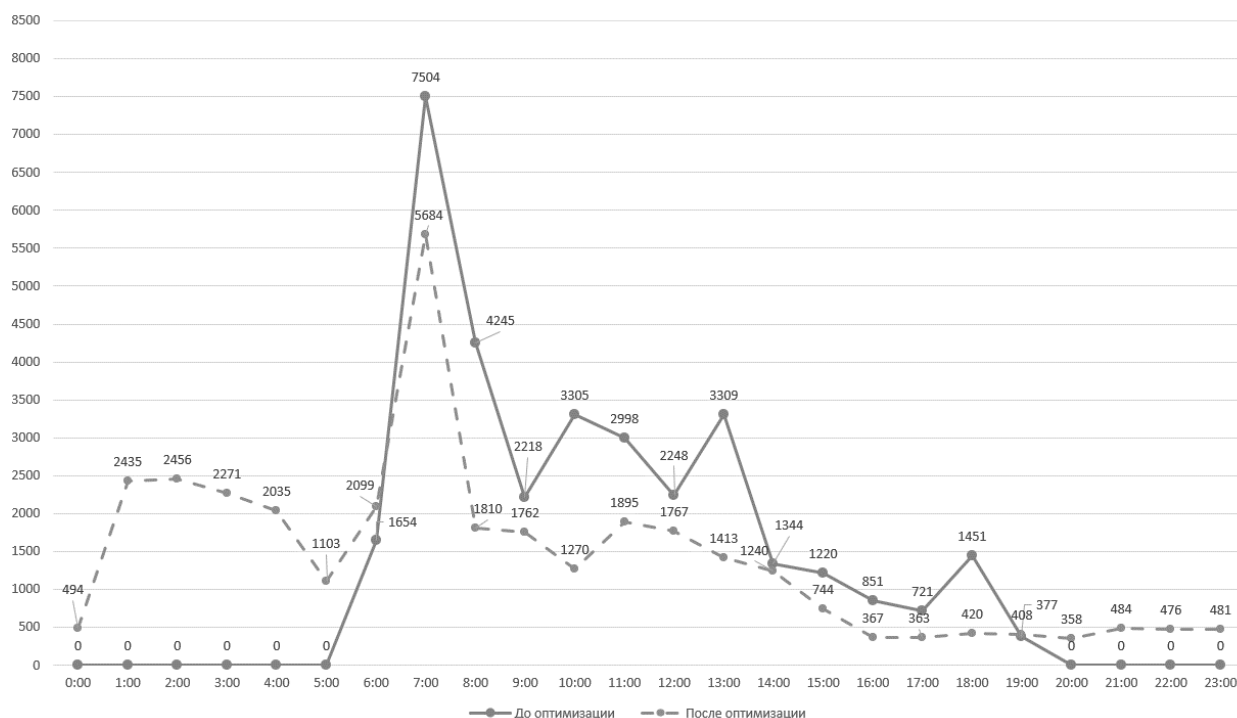


Рисунок 12 – График количества сеансов синхронизации по часам

Далее проведены графики количества запросов на обработку пакетов в интервале 10 минут. Данные брались из результатов тестов, что и для графиков на рисунке 12.

На рисунке 13 изображен сплошной линией график, по которому видно, что основной массив запросов за час укладывается в первые 2 минуты. Это вызвано установкой у всех КПС начинать передачу данных в строго определенное время.

После ввода автоматического распределения наблюдается равномерное распределение количества запросов в первые 4 минуты, не превышая критического значения. График представлен пунктиром.

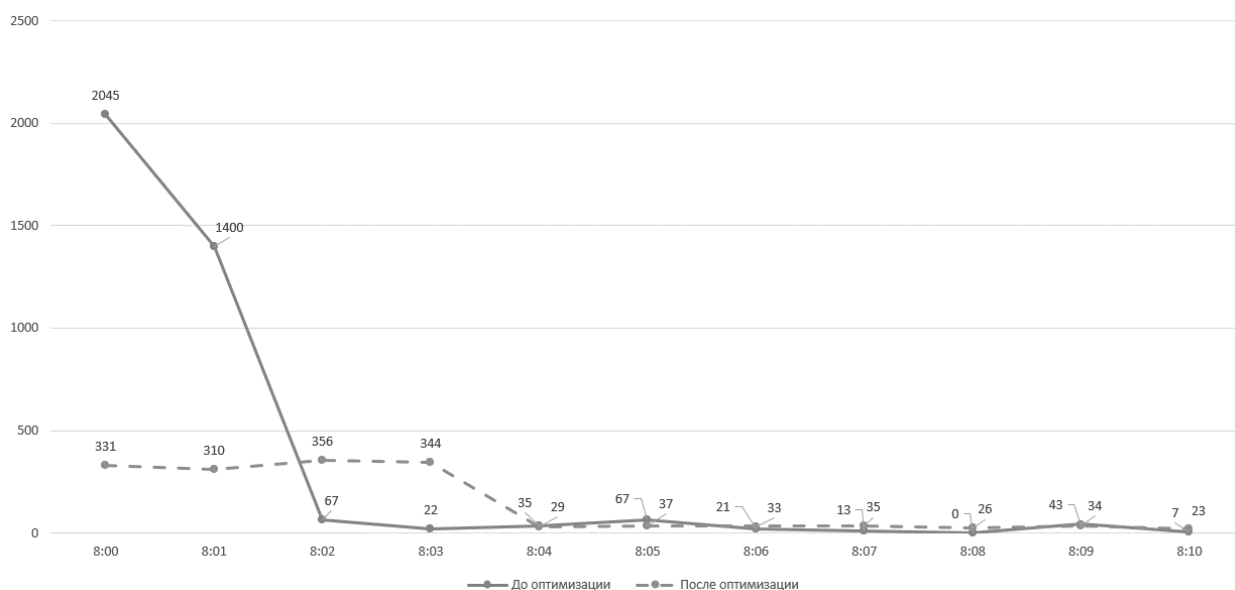


Рисунок 13 – График количества сеансов синхронизации на интервале 10 минут

Модуль сервиса автораспределения интегрирован в программный сервер. Настройка выражений по умолчанию выполняется инженером в специальном разделе WEB-интерфейса. Фрагмент WEB-интерфейса с выражениями представлен на рисунке 14.

Лимит записей в запросе к мастер базе	10000
Обработано до:	22.05.2020 10:13:14
Выражение для полной синхронизации	!22:00-05:00;!07:00-16:00
Выражение для синхронизации настроек ОО	04:00-11:00;11:00-18:00
Выражение для синхронизации данных по клиентам ОО	04:00-09:00;14:00-19:00
CRON-выражение для сервиса блокировки неиспользуемых карт	0 0 3 ? * 1
Сколько дней отсутствия активности по карте до блокировки	180
Сервер для запуска операции блокировки	1001
<input type="button" value="Сохранить"/> <input type="button" value="Отмена"/>	

Рисунок 14 — фрагмент интерфейса для настроек выражений автораспределения

3.3 Результат работы микросервиса синхронизации и влияние модуля автораспределения.

Для контроля потребляемых ресурсов и нагрузки сервера используется фреймворк Java Melody.

Java Melody — это фреймворк для мониторинга производительности и профайлинга, является частью фонда Apache Software. Позволяет в режиме реального времени следить за показателями потребляемых ресурсов сервера, а также формировать отчет мониторинга.

Далее представлены графики HTTP-запросов к серверу и потребляемых аппаратных ресурсов сервера в условиях обычной работы ИС. В графике отсутствуют данные за ночные часы в воскресенье, это вызвано профилактическими работами.

На рисунке 15 представлен график количества HTTP-запросов в минуту, большинство запросов – запросы на проведение сеанса синхронизации. Среднее значение составляет 46 запросов в минуту, пиковое значение 172 запроса в минуту. На ночное время пришлось примерно 23 запроса в минуту – эти запросы создавали исключительно КПС, выполняющие синхронизацию по расписанию после применения настроек модулем автораспределения.

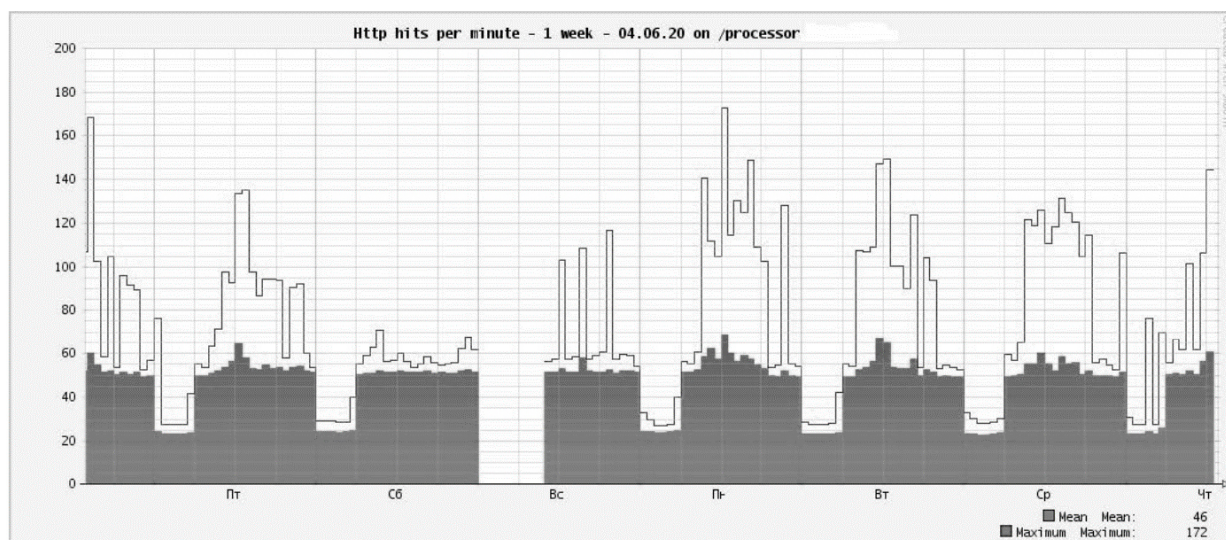


Рисунок 15 — График за неделю запросов на минуту

На рисунке 16 представлен график загрузки процессора в процентах. Пиковая нагрузка составила 45% и порождена другим сервисом, который формирует различные отчеты и рассылает их клиентам-подписчикам. Однако за всю неделю работы ИС процессор сервера не нагружался выше 50%, что говорит об отсутствии единновременного обращения от множества КПС. Данное обстоятельство говорит об эффективности работы модуля автораспределения и возможности расширить ИС, подключив новые организации к системе.

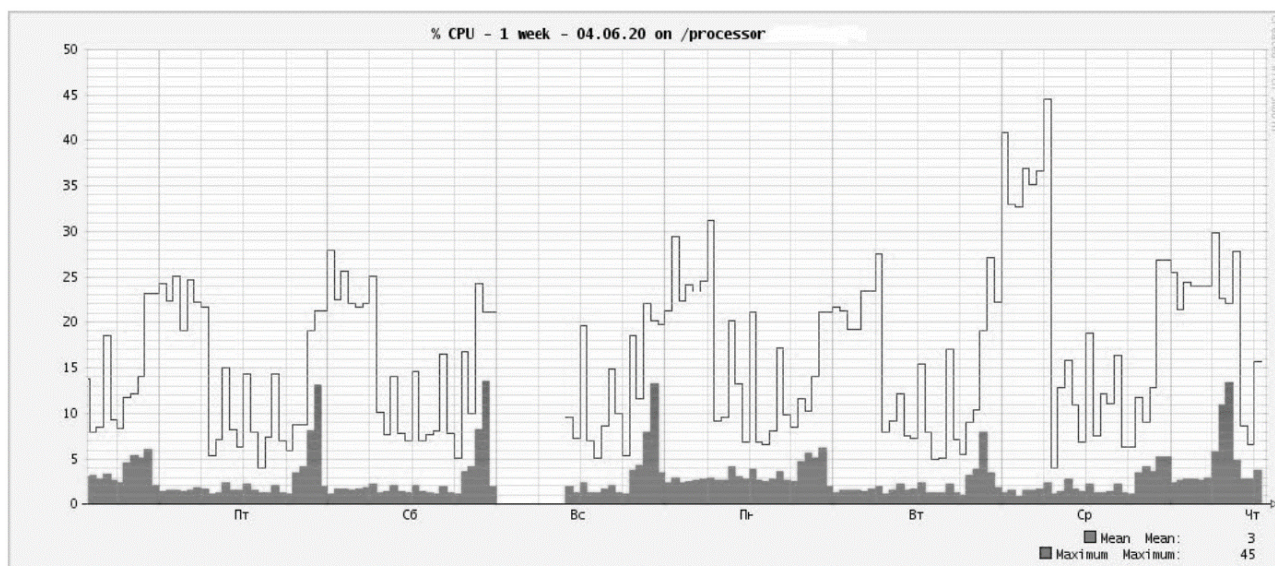


Рисунок 16 — График загрузки процессора за неделю

Далее на рисунке 17 представлен график использования JDBC-соединения. JDBC — стандарт предназначенный для взаимодействия Java-приложений с различными СУБД. Обычно используется как база для таких фреймворков, как Hibernate.

График отображает плотность потока данных между микросервисом и БД. Средняя загруженность составляет 13%. Максимальная загруженность составила 35% и так же вызвана сервисом формирования отчетов.

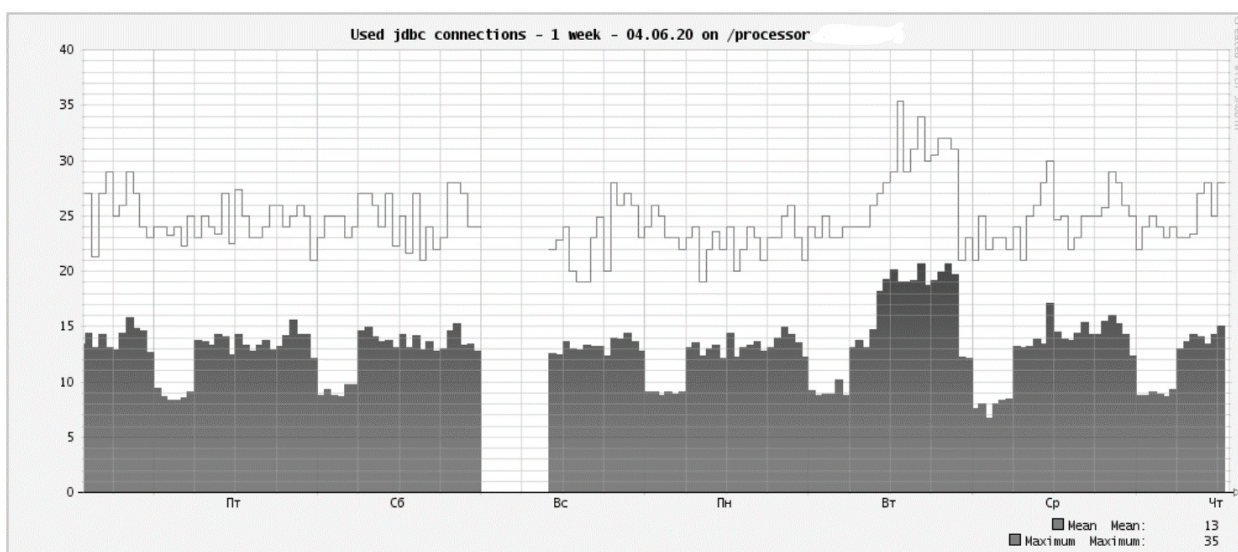


Рисунок 17 — График загруженности JDBC-соединения за неделю

3.4 Вывод

Результатом разработки стал микросервис двустороннего обмена информации между сервером и множеством КПС, который отвечает поставленным ранее требованиям.

Так же для повышения оптимизации был разработан и введен в эксплуатацию интегрированный в сервер ИС модуль автоматического распределения времени обращений, генерирующий индивидуально для каждого КПС расписание для взаимодействия. Данный модуль позволил распределить нагрузку на сервер и обеспечить возможность добавления новых участников взаимодействия.

4 Безопасные условия труда оператора ЭВМ

Комфортные и безопасные условия труда - один из основных факторов влияющих на производительность людей, работающих с персональным компьютером.

Изучение и решение проблем, связанных с обеспечением здоровых и безопасных условий, в которых протекает труд человека - одна из наиболее важных задач в разработке новых технологий и систем производства. Изучение и выявление возможных причин производственных несчастных случаев, профессиональных заболеваний, аварий, взрывов, пожаров, и разработка мероприятий и требований, направленных на устранение этих причин позволяют создать безопасные и благоприятные условия для труда человека. Комфортные и безопасные условия труда - один из основных факторов влияющих на производительность служащих инженерно-технического центра.

Работа сотрудников вычислительных центров непосредственно связана с компьютером, а соответственно, с дополнительным вредным воздействием целой группы факторов, что существенно снижает производительность их труда. К таким факторам можно отнести:

- воздействие вредных излучений от монитора и от компьютера;
- воздействие электромагнитных излучений;
- неправильная освещенность;
- не нормированный уровень шума;
- не комфортные метеорологические условия;
- прочие факторы.

4.1 Анализ условий труда оператора ЭВМ

Нормальная и безопасная работа оператора во многом зависит от того, в какой мере условия его работы соответствуют оптимальным. При этом под условиями работы подразумевают комплекс физических, химических,

биологических и психофизиологических факторов, установленных стандартами по безопасности труда.

К физическим факторам относятся: вибрация и шум, запыленность и загазованность воздуха, температура поверхностей оборудования, материалов и воздуха; плотность воздуха, ее резкое изменение, подвижность и ионизация воздуха; ионизирующие и электромагнитные излучения, статические заряды и повышение напряжения в цепи, электрические и магнитные поля; отсутствие или недостаток естественного света, повышенная или пониженная освещенность, яркость и контрастность, пульсация светового потока; ультрафиолетовая или инфракрасная радиация.

К химическим факторам относятся: общетоксические, раздражающие, сенсibiliзирующие, канцерогенные, мутагенные; действующие через дыхательные пути, пищеварительную систему, кожный покров.

К биологическим факторам относятся: микроорганизмы (бактерии, вирусы, грибы и т. п.); макроорганизмы (растения и животные).

К психофизиологическим факторам относятся перегрузки: физические (статические, динамические, гиподинамия); нервно-психические (умственное перенапряжение, монотонность труда, эмоциональные перегрузки). Рассмотрим некоторые из указанных факторов.

4.1.1 Освещение.

Хорошее освещение необходимо для выполнения большинства задач, поставленных перед оператором. Чтобы правильно спланировать рациональную систему освещения, необходимо учитывать яркость источников света, их расположение в помещении, яркостной контраст между устройствами ЭВМ и фоном, блёскость поверхностей, качество и цвет светильников и поверхностей. Для малой и средней контрастности поверхности ЭВМ при темном фоне наименьший уровень освещенности должен быть 150 лк. Для большой контрастности при светлом или темном

фоне наименьший уровень освещенности 100 лк. ЭВМ эксплуатируют в помещениях, где необходимо предусматривать систему искусственного освещения из люминесцентных ламп дневного света или ламп накаливания.

Существуют прямая, отраженная и диффузная системы искусственного освещения. При прямом освещении свет попадает на объекты непосредственно от источников света. При этом 90—100% мощности светильника направлено на рабочую поверхность, что вызывает яркостные контрасты и резкие тени. При освещении отраженным светом 90—100% света направляется на потолок и верхнюю часть стен, от которых свет более или менее равномерно отражается по всему помещению. При этом достигается ровная освещенность без теней и блескости. Диффузное освещение обеспечивает рассеянный свет, одинаково распределенный по всем направлениям.

Такая система освещения требует меньшей мощности, чем две предыдущие, но вызывает частичное образование теней и блескости. Оптимальные условия освещения могут быть созданы при применении светящегося потолка, когда источники света прикрываются полупрозрачной поверхностью. Мягкий свет, льющийся с такого потолка, почти не создает теней и блескости. При установке светильников следует избегать направления света на оператора. Небольшое количество света, направленное в глаза наблюдателю, так изменяет уровень адаптации его глаз, что он лишается способности видеть освещаемый объект.

С освещением тесно связано требование оптимального сочетания цветов потолка, стен и пола помещения и установленного в нем оборудования, которое должно основываться на учете как физических, так и психологических факторов. При выборе цвета нужно исходить из требования создания условий для хорошего отражения общего освещения, что улучшает условия зрительного восприятия. Светлые оттенки всех цветов уменьшают

поглощение света. Как правило, наибольшие коэффициенты отражения требуются для потолков, несколько меньшие – для стен и полов. Рекомендуется выбирать цвета бледных, «пастельных» оттенков. При этом следует использовать совместимые группы цветов, чтобы тот или иной цвет не казался неуместным, «не выпирал» из общей цветовой гаммы. В этом случае потолок сможет отражать 80–90% света, стены 50–60%, пол – 15–30%. Примером совместимой гаммы цветов служит следующая окраска помещения: потолок – белого цвета, верхняя часть стен – бледно-зеленого цвета, остальная часть стен – зеленого цвета средней насыщенности, пол и драпировка – зеленого цвета средней насыщенности, мебель – зеленовато-серого или светло-коричневого цвета. При окраске различных помещений следует помнить, что цвета теплых тонов (красный, желтый, коричневый) стимулируют активность человека, создают впечатление бодрости и возбуждения. Цвета холодных тонов (синий, зеленый, фиолетовый) создают впечатление «холода».

4.1.2 Температура.

Температура окружающей среды существенно сказывается на работоспособности человека, особенно на его умственной деятельности. Установлено, что температуру +50 °С, намного превышающую допустимый уровень, человек может выдержать в течение 1 ч. При +30 °С умственная деятельность человека ухудшается, замедляется его реакция, что ведет к появлению ошибок. При +11°С начинается окоченение конечностей, такая температура минимально допустима. Наиболее благоприятный диапазон температур в летнее время от +18 до +24°С, в зимнее время от +14-17 до +22 °С.

Движение воздуха позволяет увеличить рабочий диапазон температур. Так, при скорости движения воздуха 0,1; 0,5; 0,9 м/с верхняя допустимая

граница рабочего диапазона сдвигается соответственно до +22, 24, 26 °С при интенсивном расходе энергии человеком порядка 1000 Дж/ч.

Как правило, значение относительной влажности в рабочем помещении не должно быть выше 70% и ниже 30%.

Применение вентиляции в рабочих помещениях еще более улучшает условия деятельности человека. Рекомендуемая интенсивность вентиляции для помещений с установленными ЭВМ 0,5 – 1,0 м³ свежего воздуха в минуту на каждый 1 м² пола.

4.1.2 Акустические шумы

Акустические шумы и борьба с ними приобрели большое значение в последние годы со все возрастающим количеством различных механизмов, машин, самолетов и т.д., являющихся источниками звуковых колебаний различной интенсивности. Шум резко снижает производительность труда и увеличивает травматизм. Физиологически шум воздействует на органы зрения и слуха, повышается кровяное давление, притупляется внимание.

Внешняя зона соответствует неблагоприятным уровням шума. Внутренняя зона – зона комфорта. Цифры обозначают процент звуков речи, которые будут правильно услышаны человеком с нормальными органами слуха при нормальном разговоре. Снижение шума достигается: правильной планировкой, направленной на изолирование источников звука; установкой специальных средств, преграждающих путь звуку; изоляцией вибрирующих машин с помощью соответствующих звукопоглощающих приспособлений, использованием звукопоглощающих материалов.

Механические вибрации. Механические вибрации не только вредно воздействуют на организм, но и мешают человеку выполнять как мыслительные, так и двигательные операции. Зрительное восприятие ухудшается под действием вибрации, в особенности на частотах между 25 и 40 Гц и между 60 и 90 Гц. Вертикальная вибрация в большей степени влияет

на операторов, работающих сидя, а горизонтальная — на работающих стоя. Вибрации с параметрами выше внешней кривой вызывают болевые ощущения. Человеческое тело реагирует на вибрацию в основном так же, как механические системы. Когда частота внешних колебаний приближается к частоте собственных колебаний человеческого тела, равной примерно 5 Гц, действие вибраций на человека неприятно.

4.2 Меры борьбы с вредными воздействиями

4.2.1 Санитарные нормы и стандарты безопасности

Нормальная и безопасная работа оператора во многом зависит от того, в какой мере условия его работы соответствуют оптимальным. При этом под условиями работы подразумевают комплекс физических, химических, биологических и психофизиологических факторов, установленных стандартами по безопасности труда.

В целях обеспечения безопасности здоровья пользователей в Российской Федерации действуют Санитарные нормы и правила «Гигиенические требования к терминалам, персональным электронно-вычислительным машинам и организации работ» СанПиН 2.2.2.542-96. Цель Санитарных норм — определить такие нормированные величины факторов воздействия, чтобы их вред был минимальным, а условия труда — комфортными. Предельно допустимые уровни, генерируемого монитором электромагнитного поля и поверхностного электростатического потенциала установлены СанПиН 2.2.2.542-96 и приведены в таблице 19.

Таблица 19 — Предельно допустимые уровни, установленные СанПиН 2.2.2.542-96.

Вид поля	Диапазон частот	Единица измерения	ПДУ
Магнитное поле	5 Гц- 2 кГц	нТл	250
Магнитное поле	2 - 400 кГц	нТл	25
Электрическое поле	5 Гц- 2 кГц	В/м	25
Электрическое поле	2 - 400 кГц	В/м	2,5
Эквивалентный (поверхностный) электростатический потенциал		В	500

Наиболее жесткие требования к монитору предъявляет ТСО 95. Он ограничивает параметры излучения монитора, потребления электроэнергии, визуальные параметры, так что делает монитор наиболее лояльным к здоровью пользователя. В части излучательных параметров ему соответствует и ТСО 92.

Стандарт MPR II менее жесткий устанавливает предельные уровни электромагнитного поля примерно в 2,5 раза выше. Разработан Институтом защиты от излучений (Швеция) и рядом организаций, в том числе крупнейших производителей мониторов.

В части электромагнитных полей стандарту MPR II соответствует российские санитарные нормы СанПиН 2.2.2.542-96 Гигиенические требования к терминалам, персональным электронно-вычислительным машинам и организации работ.

4.2.2 Требования к помещениям с ЭВМ

Согласно СН 512-78 и СанПиН 2.2.2.542-96 здания ЭВМ следует помещать с наветренной стороны ветров преобладающего направления по отношению к соседним предприятиям, являющихся источниками выделений вредных веществ и пыли.

Площадь на одно рабочее место с ВДТ или ПЭВМ должна составлять не менее 6 м², а объем - не менее 20 м³, а в учебных помещениях соответственно не менее 6 м² и 24 м³. В залах ЭВМ должно предусматриваться автоматическое пожаротушение.

Применение для тушения пожара воды, порошковых огнетушителей недопустимо - выводится из строя ЭВМ.

Здания и помещения для ЭВМ должны быть оборудованы системами центрального отопления, приточно-вытяжной вентиляции, хозяйственно-питьевого водопровода.

4.2.3 Требования к шуму

При выполнении основной работы на мониторах и ПЭВМ (диспетчерские, операторские, расчетные, кабины и посты управления, залы вычислительной техники и др.), где работают инженерно-технические работники, осуществляющие лабораторный, аналитический или измерительный контроль, уровень шума не должен превышать 60 дБА.

На рабочих местах в помещениях для размещения шумных агрегатов вычислительных машин (АЦПУ, принтеры, сканеры и др.) уровень шума не должен превышать 75 дБА.

Шумящее оборудование (АЦПУ, принтеры и др.), уровни шума которого превышают нормированные, должно находиться вне помещения с монитором и ПЭВМ.

Снизить уровень шума в помещениях с мониторами и ПЭВМ можно использованием звукопоглощающих материалов с максимальными коэффициентами звукопоглощения в области частот 63 - 8000 Гц для отделки помещений (разрешенных органами и учреждениями Госсанэпиднадзора России), подтвержденных специальными акустическими расчетами.

Дополнительным звукопоглощением служат однотонные занавеси из плотной ткани, гармонирующие с окраской стен и подвешенные в складку на расстоянии 15 - 20 см от ограждения. Ширина занавеси должна быть в 2 раза больше ширины окна.

4.2.3 Требования к шуму

При выполнении основной работы на мониторах и ПЭВМ (диспетчерские, операторские, расчетные, кабины и посты управления, залы вычислительной техники и др.), где работают инженерно-технические работники, осуществляющие лабораторный, аналитический или измерительный контроль, уровень шума не должен превышать 60 дБА.

На рабочих местах в помещениях для размещения шумных агрегатов вычислительных машин (АЦПУ, принтеры, сканеры и др.) уровень шума не должен превышать 75 дБА.

Шумящее оборудование (АЦПУ, принтеры и др.), уровни шума которого превышают нормированные, должно находиться вне помещения с монитором и ПЭВМ.

Снизить уровень шума в помещениях с мониторами и ПЭВМ можно использованием звукопоглощающих материалов с максимальными коэффициентами звукопоглощения в области частот 63 - 8000 Гц для отделки помещений (разрешенных органами и учреждениями Госсанэпиднадзора России), подтвержденных специальными акустическими расчетами.

Дополнительным звукопоглощением служат однотонные занавеси из плотной ткани, гармонирующие с окраской стен и подвешенные в складку на расстоянии 15 - 20 см от ограждения. Ширина занавеси должна быть в 2 раза больше ширины окна.

Заключение

В ходе выполнения выпускной квалификационной работой был разработан микросервис двусторонней синхронизации данных между сервером и КПС на языке Java. Был получен опыт разработки масштабируемых микросервисов и оптимизации высоконагруженных систем.

При реализации поставленной задачи были получены следующие результаты:

- разработан протокол взаимодействия для процесса синхронизации;
- разработан модуль автоматического распределения времени обращения для оптимизации процесса синхронизации;
- разработана программа для тестирования микросервиса синхронизации.

Данный микросервис полностью соответствует выдвинутым требованиям, обеспечивая актуальность передаваемых данных, микросервисная архитектура обеспечивает масштабируемость и производительность, архитектура приложения обеспечивает достаточный уровень отказоустойчивости и возможность расширять функционал. Безопасность обеспечивается за счет протокола обмена.

Список используемых источников

1. Авдошин С. М., Песоцкая Е. Ю. Информатизация бизнеса. Управление рисками // М.: ДМК Пресс, 2011. – 176 с.
2. Грофф Д., Вайнберг П., Оппель Э. SQL. Полное руководство, Вильямс Издательский дом //-2015. -№3. -С. 87.
3. Трутнев Д. Р. Архитектуры информационных систем. Основы проектирования. // СПб.: НИУ ИТМО, 2012. – 66 с.
4. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы. Построение и анализ, Вильямс Издательский дом -2013. -№3. -С. 140-173.
5. Дж. Уорсли, Дж. Дрейк. PostgreSQL. Для профессионалов — СПб.: «Питер», 2003. — С. 23-320.
6. Леон Шкляр, Рич Розен. Архитектура веб-приложений. — М.: «Эксмо», 2010. — С.35-150.
7. Макконнелл С. Совершенный код: Практическое руководство по разработке программного обеспечения // Русская редакция/БХВ -2017. -С. 126-129.
8. Ньюмен С. Создание микросервисов = Building Microservices. — СПб.: «Питер», 2016. — С. 56-246.
9. Шилдт Г. Java 8: полное руководство. — М.: «Вильямс», 2018. — С. 149- 348.
10. Кнут Д. Искусство программирования т.1. Основные алгоритмы //Вильямс Издательский дом 2000. -С. 27.
11. Стивенс Р. Алгоритмы. Теория и практическое применение // John Wiley & Sons.-2013. -С. 72.

Приложение А. Листинги программы на Java

Файл «Processor.java»

```
package core.logic;

import com.google.common.base.Joiner;
import core.RuntimeContext;
import core.client.ContractIdFormat;
import core.event.EventNotificator;
import core.event.SyncEvent;
import core.persistence.*;
import core.persistence.dao.org.OrgSyncWritableRepository;
import core.persistence.distributedobjects.products.Good;
import core.persistence.utils.DAOReadonlyService;
import core.persistence.utils.DAOService;
import core.persistence.utils.DAOUtils;
import core.persistence.utils.MigrantsUtils;
import core.service.EventNotificationService;
import core.sync.*;
import core.sync.handlers.groups.*;
import core.sync.handlers.org.owners.OrgOwnerData;
import core.sync.handlers.org.owners.OrgOwnerProcessor;
import core.sync.handlers.orgsetting.request.OrgSettingSection;
import core.sync.handlers.payment.registry.*;
import core.sync.manager.Manager;
import core.sync.request.*;
import core.sync.response.*;
import core.sync.response.registry.accounts.AccountsRegistry;
import core.utils.CalendarUtils;
import core.utils.CurrencyStringUtils;
import core.utils.HibernateUtils;
import org.apache.commons.lang.StringUtils;
import org.hibernate.*;
import org.hibernate.criterion.Restrictions;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.IOException;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.*;
import static core.persistence.utils.DAOUtils.*;
```

```

public class Processor implements SyncProcessor {
    private static final Logger logger =
LoggerFactory.getLogger(Processor.class);
    private final SessionFactory persistenceSessionFactory;
    private final EventNotificator eventNotificator;
    private static final long ACC_REGISTRY_TIME_CLIENT_IN_MILLIS
=
RuntimeContext.getInstance().getPropertiesValue("accRegistryUpda
te.timeClient", 7) * 24 * 60*60 * 1000;
    private ProcessorUtils processorUtils =
RuntimeContext.getAppContext().getBean(ProcessorUtils.class);
    public Processor(SessionFactory persistenceSessionFactory,
EventNotificator eventNotificator) {
        this.persistenceSessionFactory =
persistenceSessionFactory;
        this.eventNotificator = eventNotificator;
    }

    @Override
    public SyncResponse processSyncRequest(SyncRequest request)
throws Exception {
        Date syncStartTime = new Date();
        int syncResult = 0;
        SyncResponse response = null;
        try {
            switch (request.getSyncType()) {
                case TYPE_FULL: {
                    response = buildFullSyncResponse(request,
syncStartTime, syncResult);
                    break;
                }
                case TYPE_GET_ACC_INC: {
                    response = buildAccIncSyncResponse(request);
                    break;
                }
                case TYPE_GET_CLIENTS_PARAMS: {
                    response =
buildClientsParamsSyncResponse(request);
                    break;
                }
                case TYPE_GET_GET_ACC_REGISGTRY_UPDATE: {
                    response = buildAccRegisgtryUpdate(request);
                    break;
                }
            }
        }
    }

```

```

        case TYPE_COMMODITY_ACCOUNTING: {
            response =
buildCommodityAccountingSyncResponse(request);
            break;
        }
        case TYPE_CONSTRUCTED: {
            response =
buildUniversalConstructedSectionsSyncResponse(request,
syncStartTime, syncResult);
            break;
        }
        case TYPE_ORG_SETTINGS: {
            response =
buildOrgSettingsSectionsResponse(request, syncStartTime,
syncResult);
            break;
        }
    }
} catch (Exception e) {
    logger.error(String.format("Failed to perform
synchronization, orgId == %s", request.getOrgId()), e);
    syncResult = 1;
}
if (request.getSyncType() == SyncType.TYPE_FULL) {
    eventNotificator.fire(new
SyncEvent.RawEvent(syncStartTime, request, response));
}
return response;
}

public void disableClientCardsIfChangeOrg(Client client,
Set<Org> oldOrgs, long neworgId) throws Exception {
    if (client == null) {
        return;
    }
    Boolean isReplaceOrg = !client.getOrg().getOrgId()
.equals(neworgId);
    for (Org o : oldOrgs) {
        if (o.getOrgId().equals(neworgId)) {
            isReplaceOrg = false;
            break;
        }
    }
}
}

```

```

        public Client getClientInfo(long idOfContract) throws
Exception {
            Session persistenceSession = null;
            Transaction persistenceTransaction = null;
            Client client = null;
            try {
                persistenceSession =
persistenceSessionFactory.openSession();
                persistenceTransaction =
persistenceSession.beginTransaction();

                client = findClientByContractId(persistenceSession,
idOfContract);
                persistenceTransaction.commit();
                persistenceTransaction = null;
            } finally {
                HibernateUtils.rollback(persistenceTransaction,
logger);
                HibernateUtils.close(persistenceSession, logger);
            }
            return client;
        }

        private void saveLastProcessSectionDateSmart(SessionFactory
sessionFactory, Long orgId, SectionType sectionType) {
            if (!(sectionType.equals(SectionType.ACC_INC_REGISTRY)
|| sectionType.equals(SectionType.LAST_TRANSACTION))) {
                return;
            }
            processorUtils.saveLastProcessSectionCustomDate(sessionFactory,
orgId, sectionType);
        }

        private SyncResponse buildFullSyncResponse(SyncRequest
request, Date syncStartTime, int syncResult)
            throws Exception {
            Long packetId = null;
            SyncHistory syncHistory = null;
            ResPaymentRegistry resPaymentRegistry = null;
            ResAccountOperationsRegistry
resAccountOperationsRegistry = null;
            SyncResponse.AccRegistry accRegistry = null;
            SyncResponse.AccIncRegistry accIncRegistry = null;
            SyncResponse.ClientRegistry clientRegistry = null;

```

```

SyncResponse.ResOrgStructure resOrgStructure = null;
SyncResponse.ResPassageEvents resPassageEvents = null;
Manager manager = null;
OrgOwnerData orgOwnerData = null;
List<Long> errorClientIds = new ArrayList<>();
AccRegistryUpdate accRegistryUpdate = null;
OrganizationStructure organizationStructure = null;
AccountsRegistry accountsRegistry = null;
OrgSettingSection orgSettingSection = null;
List<AbstractToElement> responseSections = new
ArrayList<AbstractToElement>();
    StringBuilder performanceLogger = new StringBuilder();
    Long timeForDelta = System.currentTimeMillis();
    boolean bError = false;
    packetId = generatePacketId(request.getOrgId());
    syncHistory = createSyncHistory(request.getOrgId(),
packetId, syncStartTime, request.getClientVersion(),
        request.getRemoteAddr(),
request.getSyncType().getValue());
        addClientVersionAndRemoteAddressByOrg(request.getOrgId(),
request.getClientVersion(), request.getRemoteAddr(),
            null, null);
        processAccountOperationsRegistrySections(request,
syncHistory, responseSections, null);
        try {
            if
(request.getPaymentRegistry().getPayments().hasNext()) {
                if
(!RuntimeContext.getInstance().isPermitted(request.getOrgId(),
RuntimeContext.TYPE_P)) {
                    processorUtils
.createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory,
                                "no license slots
available");
                                throw new Exception("no license slots
available");
                }
            }
        }
        saveLastProcessSectionDateSmart(persistenceSessionFactory,
request.getOrgId(),
            SectionType.PAYMENT_REGISTRY);

```

```

        resPaymentRegistry =
processSyncPaymentRegistry(syncHistory.getIdOfSync(),
request.getOrgId(),
        request.getPaymentRegistry(),
errorClientIds);
    } catch (Exception e) {
        String message = String.format("Failed to process
PaymentRegistry, orgId == %s", request.getOrgId());
        processorUtils
        .createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory, message);
        logger.error(message, e);
        bError = true;
    }
    List<Long> clientsWithWrongVersion = new ArrayList<>();
    try {
        processSyncClientParamRegistry(syncHistory,
request.getOrgId(), request.getClientParamRegistry(),
        errorClientIds, clientsWithWrongVersion);
    } catch (Exception e) {
        String message = String.format("Failed to process
ClientParamRegistry, orgId == %s", request.getOrgId());
        processorUtils.createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    }
    try {
        if (request.getOrgStructure() != null) {
            resOrgStructure =
processSyncOrgStructure(request.getOrgId(),
request.getOrgStructure(), syncHistory);
        }
        if (resOrgStructure != null &&
resOrgStructure.getResult() > 0) {
            processorUtils.createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory,
resOrgStructure.getError());
        }
    } catch (Exception e) {
        resOrgStructure = new
SyncResponse.ResOrgStructure(1, "Unexpected error");
        String message = String.format("Failed to process
OrgStructure, orgId == %s", request.getOrgId());
        processorUtils

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.18.001 ПЗ

Лист

66

```

.createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    }
    try {
saveLastProcessSectionDateSmart(persistenceSessionFactory,
request.getOrgId(),
        SectionType.CLIENT_REGISTRY);
        clientRegistry =
processSyncClientRegistry(request.getOrgId(),
request.getClientRegistryRequest(),
        errorClientIds, clientsWithWrongVersion);
    } catch (Exception e) {
        String message = String.format("Failed to build
ClientRegistry, orgId == %s", request.getOrgId());
        processorUtils

.createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    }
    try {
        final OrganizationStructureRequest
organizationStructureRequest =
request.getOrganizationStructureRequest();
        if (organizationStructureRequest != null) {
saveLastProcessSectionDateSmart(persistenceSessionFactory,
request.getOrgId(),
        SectionType.ORGANIZATIONS_STRUCTURE);
        organizationStructure =
getOrganizationStructureData(request.getOrg(),

organizationStructureRequest.getMaxVersion(),
organizationStructureRequest.isAllOrgs());
        }
    } catch (Exception e) {
        String message = String
            .format("Failed to build organization
structure, orgId == %s", request.getOrgId());
        processorUtils

.createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory, message);
    }

```

```

        organizationStructure = new
OrganizationStructure(100, String.format("Internal error: %s",
e.getMessage()));
        logger.error(message, e);
    }
    try {
        accRegistry = getAccRegistry(request.getOrgId(),
null, request.getClientVersion());
    } catch (Exception e) {
        accRegistry = new SyncResponse.AccRegistry();
        String message = String.format("Failed to build
AccRegistry, orgId == %s", request.getOrgId());
        processorUtils
        .createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    }

    try {
        if (request.getPassageEvents() != null) {
            if
(request.getPassageEvents().getEvents().size() > 0) {
                if
(!RuntimeContext.getInstance().isPermitted(request.getOrgId(),
RuntimeContext.TYPE_S)) {
                    processorUtils.createSyncHistoryException(persistenceSessionFact
ory, request.getOrgId(),
                                syncHistory, "no license slots
available");
                    throw new Exception("no license slots
available");
                }
            }
        }
        saveLastProcessSectionDateSmart(persistenceSessionFactory,
request.getOrgId(),
                                SectionType.ENTER_EVENTS);
        resPassageEvents =
processSyncPassageEvents(request.getPassageEvents(),
request.getOrg());
    }
    } catch (Exception e) {
        logger.error(String.format("Failed to process enter
events, orgId == %s", request.getOrgId()), e);
        bError = true;
    }

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.18.001 ПЗ

Лист

68

```

    }
    try {
        orgOwnerData =
processOrgOwnerData(request.getOrgId());
    } catch (Exception e) {
        String message = String.format("Failed to process
org owner data, orgId == %s", request.getOrgId());
        processorUtils
        .createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    }
    try {
        if (request.getManager() != null) {
            manager = request.getManager();
            manager.setSyncHistory(syncHistory);
            manager.process(persistenceSessionFactory);
        }
    } catch (Exception e) {
        logger.error(String.format("Failed to process of
Distribution Manager, orgId == %s", request.getOrgId()), e);
    }
    Date syncEndTime = new Date();
    updateSyncHistory(syncHistory.getIdOfSync(), syncResult,
syncEndTime);
    updateFullSyncParam(request.getOrgId());
    runRegularPaymentsIfEnabled(request);
    String fullName =
DAOService.getInstance().getPersonNameByOrg(request.getOrg());
    try {
saveLastProcessSectionDateSmart(persistenceSessionFactory,
request.getOrgId(), SectionType.ACCOUNTS_REGISTRY);
        accountsRegistry =
RuntimeContext.getAppContext().getBean(AccountsRegistryHandler.c
lass).handlerFull(request, request.getOrgId());
    } catch (Exception e) {
        logger.error(String.format("Failed to build
AccountsRegistry, orgId == %s", request.getOrgId()), e);
    }
    try {
        ClientGroupManagerRequest clientGroupManagerRequest
= request.getClientGroupManagerRequest();
        if (clientGroupManagerRequest != null) {

```

```

        ClientgroupManagersProcessor processor = new
ClientgroupManagersProcessor(persistenceSessionFactory,
clientGroupManagerRequest);
        ResClientgroupManagers resClientgroupManagers =
processor.process();
        ClientgroupManagerData clientgroupManagerData =
processor.processData(request.getOrgId());
        responseSections.add(resClientgroupManagers);
        responseSections.add(clientgroupManagerData);
    }
    } catch (Exception e) {
        String message = String.format("Failed to process
GroupManagers, orgId == %s", request.getOrgId());
processorUtils.createSyncHistoryException(persistenceSessionFact
ory, request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    }
    fullProcessingGroupsOrganization(request, syncHistory,
responseSections);
    processInfoMessageSections(request, responseSections);
    try {
        OrgSettingsRequest orgSettingsRequest =
request.getOrgSettingsRequest();
        if (orgSettingsRequest != null) {
orgSettingsRequest.setorgIdSource(request.getOrgId());
            orgSettingSection =
processOrgSettings(orgSettingsRequest);
        }
    } catch (Exception e) {
        String message = String.format("Error when process
OrgSettings: %s", e.getMessage());
processorUtils.createSyncHistoryException(persistenceSessionFact
ory, request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    }
    logger.info("Full sync performance info: " +
performanceLogger.toString());
    return new SyncResponse(request.getSyncType(),
request.getOrgId(), request.getOrg().getShortName(),
        request.getOrg().getType(), fullName, packetId,
null, syncEndTime, "",
        accRegistry, resPaymentRegistry,
resAccountOperationsRegistry, accIncRegistry, clientRegistry,

```

```

        resOrgStructure, "", resPassageEvents, manager,
orgOwnerData, accRegistryUpdate, accountsRegistry,
        organizationStructure, responseSections,
orgSettingSection);
    }

    private SyncResponse
buildUniversalConstructedSectionsSyncResponse(SyncRequest
request, Date syncStartTime, int syncResult) throws Exception {
        Long packetId = null;
        SyncHistory syncHistory = null;
        List<AbstractToElement> responseSections = new
ArrayList<AbstractToElement>();
        List<Long> errorClientIds = new ArrayList<Long>();
        List<Long> clientsWithWrongVersion = new
ArrayList<Long>();
        packetId = generatePacketId(request.getOrgId());
        syncHistory = createSyncHistory(request.getOrgId(),
packetId, syncStartTime, request.getClientVersion(),
        request.getRemoteAddr(),
request.getSyncType().getValue());
        addClientVersionAndRemoteAddressByOrg(request.getOrgId(),
request.getClientVersion(), request.getRemoteAddr(), null, null);
        fullProcessingAccountOperationsRegistry(request,
responseSections);
        Boolean wasErrorProcessedPaymentRegistry = false;
        processPaymentRegistrySections(request, syncHistory,
responseSections, wasErrorProcessedPaymentRegistry, packetId,
errorClientIds);
        boolean wasErrorProcessedAccInc =
fullProcessingAccIncRegistryOrAccIncUpdate(request,
responseSections);
        fullProcessingClientParamsRegistry(request, syncHistory,
errorClientIds, clientsWithWrongVersion);
        fullProcessingOrgStructure(request, syncHistory,
responseSections);
        fullProcessingClientsRegistry(request, syncHistory,
responseSections, errorClientIds, clientsWithWrongVersion);
        fullProcessingOrganizationStructure(request,
syncHistory, responseSections);
        fullProcessingAccRegistry(request, syncHistory,
responseSections);
        fullProcessingPassageEvents(request, syncHistory,
responseSections);

```

```

        fullProcessingOrgOwnerData(request, syncHistory,
responseSections);
        fullProcessingRO(request, syncHistory,
responseSections);
        if (request.isFullSync() || request.isAccIncSync()) {
            runRegularPaymentsIfEnabled(request);
        }
        fullProcessingAccountsRegistry(request,
responseSections);
        fullProcessingClientGroupManagers(request, syncHistory,
responseSections);
        fullProcessingGroupsOrganization(request, syncHistory,
responseSections);
        processInfoMessageSections(request, responseSections);
        fullProcessingOrgSettings(request, syncHistory,
responseSections);
        Date syncEndTime = new Date();
        if (request.isFullSync() && syncHistory != null) {
            updateSyncHistory(syncHistory.getIdOfSync(),
syncResult, syncEndTime);
            updateFullSyncParam(request.getOrgId());
        }
        String fullName =
DAOService.getInstance().getPersonNameByOrg(request.getOrg());
        return new SyncResponse(request.getSyncType(),
request.getOrgId(), request.getOrg().getShortName(),
            request.getOrg().getType(), fullName, packetId,
null, syncEndTime,
            responseSections);
    }

    private void fullProcessingGroupsOrganization(SyncRequest
request, SyncHistory syncHistory,
        List<AbstractToElement> responseSections) {
        GroupsOrganizationRequest requestSection =
request.findSection(GroupsOrganizationRequest.class);
        if (requestSection == null) {
            return;
        }
        try {
            ResProcessGroupsOrganization resGroupsOrganization =
processResGroupsOrganization(requestSection);
            addToResponseSections(resGroupsOrganization,
responseSections);
        } catch (Exception e) {

```

```

        String message = String.format("Failed to process
ResGroupsOrganization, %s", e.getMessage());
processorUtils.createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    } try {
        ProcessGroupsOrganizationData groupsOrganizationData
= processGroupsOrganizationData(requestSection);
        addToResponseSections(groupsOrganizationData,
responseSections);
    } catch (Exception e) {
        String message = String.format("Failed to process
GroupsOrganization, %s", e.getMessage());
processorUtils.createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    }
}

private void fullProcessingRO(SyncRequest request,
SyncHistory syncHistory,
List<AbstractToElement> responseSections) {
    try {
        if (request.getManager() != null) {
            Manager manager = request.getManager();
            manager.setSyncHistory(syncHistory);
            manager.process(persistenceSessionFactory);
            addToResponseSections(manager,
responseSections);
        }
    } catch (Exception e) {
        logger.error(
            String.format("Failed to process of
Distribution Manager, orgId == %s", request.getOrgId()), e);
    }
}

private void fullProcessingAccountsRegistry(SyncRequest
request, List<AbstractToElement> responseSections) {
    try {
        AccountsRegistryRequest requestSection =
request.findSection(AccountsRegistryRequest.class);
        if (requestSection != null) {

```

```

        AccountsRegistryHandler accountsRegistryHandler
=
RuntimeContext.getAppContext().getBean(AccountsRegistryHandler.c
lass);

        AccountsRegistry result = null;
        switch (requestSection.getContentType()) {
            case ForAll: {
saveLastProcessSectionDateSmart(persistenceSessionFactory,
request.getOrgId(), SectionType.ACCOUNTS_REGISTRY);
                result =
accountsRegistryHandler.handlerFull(request,
request.getOrgId());
                break;
            }
            case ForCardsAndClients: {
                result =
accountsRegistryHandler.accRegistryUpdateHandler(request);
                break;
            }
            case ForMigrants: {
                result =
accountsRegistryHandler.handlerMigrants(request.getOrgId());
            }
        }
        if (result != null) {
            addToResponseSections(result,
responseSections);
        }
    } catch (Exception e) {
        logger.error(String.format("Failed to build
AccountsRegistry, orgId == %s", request.getOrgId()), e);
    }
}

private void fullProcessingClientGroupManagers(SyncRequest
request, SyncHistory syncHistory, List<AbstractToElement>
responseSections) {
    try {
        ClientGroupManagerRequest clientGroupManagerRequest
= request.getClientGroupManagerRequest();
        if (clientGroupManagerRequest != null) {
            ClientgroupManagersProcessor processor = new
ClientgroupManagersProcessor(persistenceSessionFactory,

```

```

        clientGroupManagerRequest);
        ResClientgroupManagers resClientgroupManagers =
processor.process();
        addToResponseSections(resClientgroupManagers,
responseSections);
        ClientgroupManagerData clientgroupManagerData =
processor.processData(request.getOrgId());
        addToResponseSections(clientgroupManagerData,
responseSections);
    }
    } catch (Exception e) {
        String message = String.format("Failed to process
GroupManagers, orgId == %s", request.getOrgId());
        processorUtils

.createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    }
}

private void fullProcessingOrgOwnerData(SyncRequest request,
SyncHistory syncHistory,
List<AbstractToElement> responseSections) {
    try {
        SectionRequest requestSection =
request.findSection(OrgOwnerDataRequest.class);
        if (requestSection != null) {
            OrgOwnerData orgOwnerData =
processOrgOwnerData(request.getOrgId());
            addToResponseSections(orgOwnerData,
responseSections);
        }
    } catch (Exception e) {
        String message = String.format("Failed to process
org owner data, orgId == %s", request.getOrgId());
        processorUtils.createSyncHistoryException(persistenceSessionFact
ory, request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    }
}

private void fullProcessingPassageEvents(SyncRequest
request, SyncHistory syncHistory,

```

```

        List<AbstractToElement> responseSections) {
    try {
        if (request.getPassageEvents() != null) {
            if
(request.getPassageEvents().getEvents().size() > 0) {
                if
(!RuntimeContext.getInstance().isPermitted(request.getOrgId(),
RuntimeContext.TYPE_S)) {

processorUtils.createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory, "no license slots
available");
throw new Exception("no license slots available");
}

}

saveLastProcessSectionDateSmart(persistenceSessionFactory,
request.getOrgId(), SectionType.ENTER_EVENTS);
        SyncResponse.ResPassageEvents resPassageEvents =
processSyncPassageEvents(request.getPassageEvents(), request.getOrg());

        addToResponseSections(resPassageEvents,
responseSections);
    }
    } catch (Exception e) {
        logger.error(String.format("Failed to process enter
events, orgId == %s", request.getOrgId()), e);
    }
}

private void fullProcessingAccRegistry(SyncRequest request,
SyncHistory syncHistory,
List<AbstractToElement> responseSections) {
    SyncResponse.AccRegistry accRegistry = null;
    try {
        accRegistry = getAccRegistry(request.getOrgId(),
null, request.getClientVersion());
    } catch (Exception e) {
        accRegistry = new SyncResponse.AccRegistry();
        String message = String.format("Failed to build
AccRegistry, orgId == %s", request.getOrgId());
processorUtils.createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    }
}

```

```

        addToResponseSections(accRegistry, responseSections);
    }
    private boolean
fullProcessingAccIncRegistryOrAccIncUpdate(SyncRequest request,
        List<AbstractToElement> responseSections) {
        SyncRequest.AccIncRegistryRequest accIncRegistryRequest
= request.getAccIncRegistryRequest();
        if (accIncRegistryRequest == null) {
            return false;
        }
        boolean wasError = false;
        try {
            AccRegistryUpdate accRegistryUpdate =
getAccRegistryUpdate(request.getOrg(),
                accIncRegistryRequest.dateTime);
            addToResponseSections(accRegistryUpdate,
responseSections);
            saveLastProcessSectionDateSmart(persistenceSessionFactory,
request.getOrgId(),
                SectionType.ACC_INC_REGISTRY);
        } catch (Exception e) {
            logger.error(String.format("Failed to build
AccIncRegistry, orgId == %s", request.getOrgId()), e);
            AccRegistryUpdate accRegistryUpdate = new
AccRegistryUpdate();
            addToResponseSections(accRegistryUpdate,
responseSections);
            wasError = true;
        }
        return wasError;
    }

    private void fullProcessingOrganizationStructure(SyncRequest
request, SyncHistory syncHistory,
        List<AbstractToElement> responseSections) {
        final OrganizationStructureRequest
organizationStructureRequest = request

.findSection(OrganizationStructureRequest.class);
        if (organizationStructureRequest == null) {
            return;
        }
        OrganizationStructure organizationStructureData = null;
        try {

```

```

saveLastProcessSectionDateSmart(persistenceSessionFactory,
request.getOrgId(),
        SectionType.ORGANIZATIONS_STRUCTURE);
        organizationStructureData =
getOrganizationStructureData(request.getOrg(),
        organizationStructureRequest.getMaxVersion(),
organizationStructureRequest.isAllOrgs());
    } catch (Exception e) {
        String message = String
            .format("Failed to build organization
structure, orgId == %s", request.getOrgId());
processorUtils.createSyncHistoryException(persistenceSessionFact
ory, request.getOrgId(), syncHistory, message);
        organizationStructureData = new
OrganizationStructure(100,
            String.format("Internal error: %s",
e.getMessage()));
        logger.error(message, e);
    }
    addToResponseSections(organizationStructureData,
responseSections);
}

private void fullProcessingClientsRegistry(SyncRequest
request, SyncHistory syncHistory, List<AbstractToElement>
responseSections, List<Long> errorClientIds, List<Long>
clientsWithWrongVersion) {
    try {
        SyncRequest.ClientRegistryRequest
clientRegistryRequest = request.getClientRegistryRequest();
        if (clientRegistryRequest != null) {
saveLastProcessSectionDateSmart(persistenceSessionFactory,
request.getOrgId(),SectionType.CLIENT_REGISTRY);
            SyncResponse.ClientRegistry clientRegistry =
processSyncClientRegistry(request.getOrgId(),
                clientRegistryRequest, errorClientIds,
clientsWithWrongVersion);
            addToResponseSections(clientRegistry,
responseSections);
        }
    } catch (Exception e) {
        String message = String.format("Failed to build
ClientRegistry, orgId == %s", request.getOrgId());

```

```

processorUtils.createSyncHistoryException(persistenceSessionFactory, request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    }
}

private boolean fullProcessingPaymentRegistry(SyncRequest request, SyncHistory syncHistory,
        List<AbstractToElement> responseSections, List<Long> errorClientIds, Long packetId) {
    boolean wasError = false;
    try {
        PaymentRegistry paymentRegistryRequest = request.getPaymentRegistry();
        if (paymentRegistryRequest == null) {
            return wasError;
        }
        if (paymentRegistryRequest.getPayments() != null && paymentRegistryRequest.getPayments().hasNext()) {
            if
(!RuntimeContext.getInstance().isPermitted(request.getOrgId(), RuntimeContext.TYPE_P)) {
                SyncHistory localSyncHistory = syncHistory;
                if (localSyncHistory == null) {
String clientVersion = (request.getClientVersion() == null ? "" : request.getClientVersion());
                Long packet = (packetId == null ? -1L : packetId);
                localSyncHistory = createSyncHistory(request.getOrgId(), packet, new Date(), clientVersion, request.getRemoteAddr(), request.getSyncType().getValue());
                }
                final String s = String
                    .format("Failed to process PaymentRegistry, orgId == %s, no license slots available", request.getOrgId());
processorUtils.createSyncHistoryException(persistenceSessionFactory, request.getOrgId(),
                    localSyncHistory, s);
                throw new Exception("no license slots available");
            }
        }
    }
}

saveLastProcessSectionDateSmart(persistenceSessionFactory, request.getOrgId(),

```

```

        SectionType.PAYMENT_REGISTRY);
        ResPaymentRegistry resPaymentRegistry =
processSyncPaymentRegistry(
            syncHistory != null ?
syncHistory.getIdOfSync() : null, request.getOrgId(),
            paymentRegistryRequest, errorClientIds);
        addToResponseSections(resPaymentRegistry,
responseSections);
    } catch (Exception e) {
        wasError = true;
        String message = String.format("Failed to process
PaymentRegistry, orgId == %s", request.getOrgId());
processorUtils.createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    }
    return wasError;
}

private void fullProcessingClientParamsRegistry(SyncRequest
request, SyncHistory syncHistory,
        List<Long> errorClientIds, List<Long>
clientsWithWrongVersion) {
    try {
        SyncRequest.ClientParamRegistry clientParamRegistry
= request.getClientParamRegistry();
        if (clientParamRegistry != null) {
            processSyncClientParamRegistry(syncHistory,
request.getOrgId(), clientParamRegistry, errorClientIds,
                clientsWithWrongVersion);
        }
    } catch (Exception e) {
        String message = String.format("Failed to process
ClientParamRegistry, orgId == %s", request.getOrgId());
processorUtils.createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    }
}

private void fullProcessingOrgStructure(SyncRequest request,
SyncHistory syncHistory, List<AbstractToElement>
responseSections) {

```

```

        SyncRequest.OrgStructure orgStructureRequest =
request.getOrgStructure();
        if (orgStructureRequest == null) {
            return;
        }
        SyncResponse.ResOrgStructure resOrgStructure = null;
        try {
            resOrgStructure =
processSyncOrgStructure(request.getOrgId(), orgStructureRequest,
syncHistory);
            if (resOrgStructure != null &&
resOrgStructure.getResult() > 0) {
processorUtils.createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory,
resOrgStructure.getError());
        }
    } catch (Exception e) {
        resOrgStructure = new
SyncResponse.ResOrgStructure(1, "Unexpected error");
        String message = String.format("Failed to process
OrgStructure, orgId == %s", request.getOrgId());
processorUtils.createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    }
    addToResponseSections(resOrgStructure,
responseSections);
}

private void
fullProcessingAccountOperationsRegistry(SyncRequest request,
List<AbstractToElement> responseSections) {
    try {
saveLastProcessSectionDateSmart(persistenceSessionFactory,
request.getOrgId(), SectionType.ACCOUNT_OPERATIONS_REGISTRY);
        if (request.getAccountOperationsRegistry() != null)
        {
            AccountOperationsRegistryHandler
accountOperationsRegistryHandler = new
AccountOperationsRegistryHandler();
            ResAccountOperationsRegistry
resAccountOperationsRegistry = accountOperationsRegistryHandler
                .process(request);
        }
    }
}

```

```

addToResponseSections(resAccountOperationsRegistry,
responseSections);
    }
    } catch (Exception e) {
        logger.error("Ошибка при обработке
AccountOperationsRegistry: ", e);
    }
}

private void
processAccountRegistrySectionsForMigrants(SyncRequest request,
SyncHistory syncHistory, List<AbstractToElement>
responseSections) {
    try {
        AccountsRegistryRequest requestSection =
request.findSection(AccountsRegistryRequest.class);
        if (requestSection == null ||
AccountsRegistryRequest.ContentType.ForCardsAndClients
.equals(requestSection.getContentType())) {
            AccountsRegistry accountsRegistry =
RuntimeContext.getAppContext().getBean(AccountsRegistryHandler.c
lass).handlerMigrants(request.getOrgId());
            addToResponseSections(accountsRegistry,
responseSections);
        }
    } catch (Exception e) {
        String message = String.format("Failed to build
AccountsRegistry, orgId == %s", request.getOrgId());
        processorUtils.createSyncHistoryException(persistenceSessionFact
ory, request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    }
}

private void
processAccRegistrySectionsForMigrants(SyncRequest request,
SyncHistory syncHistory, List<AbstractToElement>
responseSections) {
    SyncResponse.AccRegistry accRegistryForMigrants;
    try {
        accRegistryForMigrants =
getAccRegistryForMigrants(request.getOrgId());
        addToResponseSections(accRegistryForMigrants,
responseSections);
    }
}

```

```

        } catch (Exception e) {
            String message = String.format("Failed to build
AccRegistry, orgId == %s", request.getOrgId());
            if (syncHistory != null) {
processorUtils.createSyncHistoryException(persistenceSessionFactory, request.getOrgId(), syncHistory, message);
            }
            logger.error(message, e);
        }
    }

    private void
processClientRegistrySectionsForMigrants(SyncRequest request,
SyncHistory syncHistory,
List<AbstractToElement> responseSections) {
        try {
            SyncResponse.ClientRegistry clientRegistry =
processSyncClientRegistryForMigrants(request.getOrgId());
            addToResponseSections(clientRegistry,
responseSections);
        } catch (Exception e) {
            String message = String.format("Failed to build
ClientRegistry, orgId == %s", request.getOrgId());
            if (syncHistory != null) {
processorUtils.createSyncHistoryException(persistenceSessionFactory, request.getOrgId(), syncHistory, message);
            }
            logger.error(message, e);
        }
    }

    private void
processAccountOperationsRegistrySections(SyncRequest request,
SyncHistory syncHistory,
List<AbstractToElement> responseSections, Boolean
error) {
        try {
            if (request.getAccountOperationsRegistry() != null)
            {
saveLastProcessSectionDateSmart(persistenceSessionFactory,
request.getOrgId(),
SectionType.ACCOUNT_OPERATIONS_REGISTRY);
            }
        }
    }

```

```

        AccountOperationsRegistryHandler
accountOperationsRegistryHandler = new
AccountOperationsRegistryHandler();
        ResAccountOperationsRegistry
resAccountOperationsRegistry = accountOperationsRegistryHandler
        .process(request);
addToResponseSections(resAccountOperationsRegistry,
responseSections);
    }
    } catch (Exception e) {
        if (error != null) {
            error = true;
        }
        String message = String.format("Ошибка при обработке
AccountOperationsRegistry: %s", e.getMessage());
        if (syncHistory != null) {
processorUtils.createSyncHistoryException(persistenceSessionFactory,
request.getOrgId(), syncHistory,
            message);
        }
        logger.error(message, e);
    }
}

private void processPaymentRegistrySections(SyncRequest
request, SyncHistory syncHistory,
        List<AbstractToElement> responseSections, Boolean
error, Long packetId, List<Long> errorClientIds) {
    try {
        if (request.getPaymentRegistry() != null) {

saveLastProcessSectionDateSmart(persistenceSessionFactory,
request.getOrgId(), SectionType.PAYMENT_REGISTRY);
if (request.getPaymentRegistry().getPayments() != null) {
            String clientVersion =
(request.getClientVersion() == null ? "" :
request.getClientVersion());
Long packet = (packetId == null ? -1L : packetId);
            if (syncHistory == null &&
!request.getSyncType().equals(SyncType.TYPE_GET_ACC_INC)) {
                syncHistory =
createSyncHistory(request.getOrgId(), packet, new Date(),
clientVersion,
request.getRemoteAddr(), request.getSyncType().getValue());
            }
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        }
        if
(request.getPaymentRegistry().getPayments().hasNext()) {
            if
(!RuntimeContext.getInstance().isPermitted(request.getOrgId(),
RuntimeContext.TYPE_P)) {
                final String s = String
                    .format("Failed to process
PaymentRegistry, orgId == %s, no license slots available",
                        request.getOrgId());
processorUtils.createSyncHistoryException(persistenceSessionFactory, request.getOrgId(), syncHistory, s);
                throw new Exception("no license
slots available");
            }
        }
        ResPaymentRegistry resPaymentRegistry =
processSyncPaymentRegistry(null,
request.getOrgId(), request.getPaymentRegistry(), errorClientIds);
        addToResponseSections(resPaymentRegistry,
responseSections);
    }
} catch (Exception e) {
    logger.error(String.format("Failed to process
Payment Registry, orgId == %s", request.getOrgId()), e);
    if (error != null) {
        error = true;
    }
}
}

private SyncResponse
buildCommodityAccountingSyncResponse(SyncRequest request) throws
Exception {
    Long packetId = null;
    SyncHistory syncHistory = null;
    ResPaymentRegistry resPaymentRegistry = null;
    ResAccountOperationsRegistry
resAccountOperationsRegistry = null;
    SyncResponse.AccRegistry accRegistry = null;
    SyncResponse.AccIncRegistry accIncRegistry = null;
    SyncResponse.ClientRegistry clientRegistry = null;
    SyncResponse.ResOrgStructure resOrgStructure = null;

```

```

SyncResponse.ResPassageEvents resPassageEvents = null;
Manager manager = null;
OrgOwnerData orgOwnerData = null;
AccRegistryUpdate accRegistryUpdate = null;
OrganizationStructure organizationStructure = null;
AccountsRegistry accountsRegistry = null;
OrgSettingSection orgSettingSection = null;
List<AbstractToElement> responseSections = new
ArrayList<AbstractToElement>();
boolean bError = false;
packetId = generatePacketId(request.getOrgId());
try {
    orgOwnerData =
processOrgOwnerData(request.getOrgId());
} catch (Exception e) {
    String message = String.format("Failed to process
org owner data, orgId == %s", request.getOrgId());
processorUtils.createSyncHistoryException(persistenceSessionFact
ory, request.getOrgId(), syncHistory, message);
    logger.error(message, e);
}
try {
    if (request.getManager() != null) {
        manager = request.getManager();
        manager.setSyncHistory(syncHistory);
        manager.process(persistenceSessionFactory);
    }
} catch (Exception e) {
    logger.error(
        String.format("Failed to process of
Distribution Manager, orgId == %s", request.getOrgId()), e);
}
Date syncEndTime = new Date();
String fullName =
DAOService.getInstance().getPersonNameByOrg(request.getOrg());
return new SyncResponse(request.getSyncType(),
request.getOrgId(), request.getOrg().getShortName(),
    request.getOrg().getType(), fullName, packetId,
null, syncEndTime, "", accRegistry, resPaymentRegistry,
resAccountOperationsRegistry, accIncRegistry, clientRegistry,
    resOrgStructure, "", resPassageEvents, manager,
orgOwnerData, accRegistryUpdate, accountsRegistry,
    organizationStructure, responseSections,
orgSettingSection);

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.18.001 ПЗ

Лист

86

```

    }

    private void processInfoMessageSections(SyncRequest request,
List<AbstractToElement> responseSections) {
        InfoMessageRequest infoMessageRequest =
request.getInfoMessageRequest();
        if (infoMessageRequest == null) {
            return;
        }

        InfoMessageData infoMessageData = null;
        try {
            saveLastProcessSectionDateSmart(persistenceSessionFactory,
request.getOrgId(), SectionType.INFO_MESSAGE);
            infoMessageData =
getInfoMessageData(request.getOrg(),
infoMessageRequest.getMaxVersion());
        } catch (Exception e) {
            String message = String
                .format("Failed to build organization
structure, orgId == %s", request.getOrgId());
            infoMessageData = new InfoMessageData(
                new ResultOperation(100,
String.format("Internal error: %s", e.getMessage())));
            logger.error(message, e);
        }
        addToResponseSections(infoMessageData,
responseSections);
    }

    private SyncResponse
buildClientsParamsSyncResponse(SyncRequest request) {
        Long packetId = null;
        SyncHistory idOfSync = null;
        ResPaymentRegistry resPaymentRegistry = null;
        ResAccountOperationsRegistry
resAccountOperationsRegistry = null;
        SyncResponse.AccRegistry accRegistry = null;
        SyncResponse.AccIncRegistry accIncRegistry = null;
        SyncResponse.ClientRegistry clientRegistry = null;
        SyncResponse.ResOrgStructure resOrgStructure = null;
        SyncResponse.ResPassageEvents resPassageEvents = null;
        Manager manager = null;
        OrgOwnerData orgOwnerData = null;
    }

```

```

        AccRegistryUpdateRequest accRegistryUpdateRequest =
null;
        List<Long> errorClientIds = new ArrayList<Long>();
        AccRegistryUpdate accRegistryUpdate = null;
        OrganizationStructure organizationStructure = null;
        AccountsRegistry accountsRegistry = null;
        OrgSettingSection orgSettingSection = null;
        List<AbstractToElement> responseSections = new
ArrayList<AbstractToElement>();
        Boolean bError = false;
        processAccountOperationsRegistrySections(request, null,
responseSections, null);
        processPaymentRegistrySections(request, null,
responseSections, bError, packetId, errorClientIds);
        try {
            accRegistryUpdateRequest =
request.getAccRegistryUpdateRequest();
        } catch (Exception e) {
            accRegistry = new SyncResponse.AccRegistry();
            String message = String.format("Failed to build
AccRegistry, orgId == %s", request.getOrgId());
            logger.error(message, e);
        }
        try {
            if (accRegistryUpdateRequest != null) {
                accRegistry = getAccRegistry(request.getOrgId(),
accRegistryUpdateRequest.getClientIds(),
request.getClientVersion());
            } else {
                accRegistry = getAccRegistry(request.getOrgId(),
null, request.getClientVersion());
            }
        } catch (Exception e) {
            accRegistry = new SyncResponse.AccRegistry();
            String message = String.format("Failed to build
AccRegistry, orgId == %s", request.getOrgId());
            logger.error(message, e);
        }
        List<Long> clientsWithWrongVersion = new
ArrayList<Long>();
        try {
            processSyncClientParamRegistry(idOfSync,
request.getOrgId(), request.getClientParamRegistry(),
errorClientIds, clientsWithWrongVersion);

```

```

        } catch (Exception e) {
            logger.error(String.format("Failed to process
ClientParamRegistry, orgId == %s", request.getOrgId()),e);
        }
        try {
            saveLastProcessSectionDateSmart(persistenceSessionFactory,
            request.getOrgId(),SectionType.CLIENT_REGISTRY);
            clientRegistry =
            processSyncClientRegistry(request.getOrgId(),
            request.getClientRegistryRequest(),errorClientIds,
            clientsWithWrongVersion);
        } catch (Exception e) {
            logger.error(String.format("Failed to build
ClientRegistry, orgId == %s", request.getOrgId()), e);
        }
        Date syncEndTime = new Date();
        return new SyncResponse(request.getSyncType(),
            request.getOrgId(), request.getOrg().getShortName(),
            request.getOrg().getType(), "", packetId, null,
            syncEndTime, "",accRegistry, resPaymentRegistry,
            resAccountOperationsRegistry, accIncRegistry, clientRegistry,
            resOrgStructure, "", resPassageEvents, manager,
            orgOwnerData, accRegistryUpdate, accountsRegistry,
            organizationStructure, responseSections, orgSettingSection);
    }

    private SyncResponse buildAccRegisgtryUpdate(SyncRequest
request) {
        Long packetId = null;
        SyncHistory idOfSync = null;
        ResPaymentRegistry resPaymentRegistry = null;
        ResAccountOperationsRegistry
resAccountOperationsRegistry = null;
        SyncResponse.AccRegistry accRegistry = null;
        SyncResponse.AccIncRegistry accIncRegistry = null;
        SyncResponse.ClientRegistry clientRegistry = null;
        SyncResponse.ResOrgStructure resOrgStructure = null;
        SyncResponse.ResPassageEvents resPassageEvents = null;
        Manager manager = null;
        OrgOwnerData orgOwnerData = null;
        AccRegistryUpdateRequest accRegistryUpdateRequest =
null;
        List<Long> errorClientIds = new ArrayList<Long>();
        AccRegistryUpdate accRegistryUpdate = null;

```

```

        OrganizationStructure organizationStructure = null;
        AccountsRegistry accountsRegistry = null;
        OrgSettingSection orgSettingSection = null;
        List<AbstractToElement> responseSections = new
ArrayList<>();
        processAccountOperationsRegistrySections(request, null,
responseSections, null);
        processPaymentRegistrySections(request, null,
responseSections, null, packetId, errorClientIds);
        try {
            accRegistryUpdateRequest =
request.getAccRegistryUpdateRequest();
        } catch (Exception e) {
            accRegistry = new SyncResponse.AccRegistry();
            String message = String.format("Failed to build
AccRegistry, orgId == %s", request.getOrgId());
            logger.error(message, e);
        } try {
            if (accRegistryUpdateRequest != null) {
                accRegistry = getAccRegistry(request.getOrgId(),
accRegistryUpdateRequest.getClientIds(), request.getClientVersion
());
            } else {
                accRegistry = getAccRegistry(request.getOrgId(),
null, request.getClientVersion());
            }
        } catch (Exception e) {
            accRegistry = new SyncResponse.AccRegistry();
            String message = String.format("Failed to build
AccRegistry, orgId == %s", request.getOrgId());
            logger.error(message, e);
        }
        try {
            accountsRegistry = new
AccountsRegistryHandler().accRegistryUpdateHandler(request);
        } catch (Exception e) {
            logger.error(String.format("Failed to build
AccountsRegistry, orgId == %s", request.getOrgId()), e);
        }
        Date syncEndTime = new Date();
        return new SyncResponse(request.getSyncType(),
request.getOrgId(), request.getOrg().getShortName(),

```

```

        request.getOrg().getType(), "", packetId, null,
syncEndTime, "", accRegistry, resPaymentRegistry,
resAccountOperationsRegistry, accIncRegistry, clientRegistry,
        resOrgStructure, "", resPassageEvents, manager,
orgOwnerData, accRegistryUpdate, accountsRegistry,
        organizationStructure, responseSections,
orgSettingSection);
    }

```

```

    private SyncResponse buildAccIncSyncResponse(SyncRequest
request) {
        Long packetId = request.getPacketId();
        ResAccountOperationsRegistry
resAccountOperationsRegistry = null;
        ResPaymentRegistry resPaymentRegistry = null;
        SyncResponse.AccRegistry accRegistry = null;
        SyncResponse.AccIncRegistry accIncRegistry = null;
        SyncResponse.ClientRegistry clientRegistry = null;
        SyncResponse.ResOrgStructure resOrgStructure = null;
        SyncResponse.ResPassageEvents resPassageEvents = null;
        Manager manager = null;
        OrgOwnerData orgOwnerData = null;
        List<Long> errorClientIds = new ArrayList<Long>();
        AccRegistryUpdate accRegistryUpdate = null;
        OrganizationStructure organizationStructure = null;
        AccountsRegistry accountsRegistry = null;
        OrgSettingSection orgSettingSection = null;
        List<AbstractToElement> responseSections = new
ArrayList<AbstractToElement>();
        Boolean bError = false;
        processAccountOperationsRegistrySections(request, null,
responseSections, bError);
        processPaymentRegistrySections(request, null,
responseSections, bError, packetId, errorClientIds);
        processInfoMessageSections(request, responseSections);
        try {
            accRegistryUpdate =
getAccRegistryUpdate(request.getOrg(),
request.getAccIncRegistryRequest().dateTime);
            saveLastProcessSectionDateSmart(persistenceSessionFactory,
request.getOrgId(), SectionType.ACC_INC_REGISTRY);
        } catch (Exception e) {
            logger.error(String.format("Failed to build
AccIncRegistry, orgId == %s", request.getOrgId()), e);

```

```

        accRegistryUpdate = new AccRegistryUpdate();
        bError = true;
    }
    try {
        if (request.getPassageEvents() != null) {
            saveLastProcessSectionDateSmart(persistenceSessionFactory,
            request.getOrgId(), SectionType.ENTER_EVENTS);
            resPassageEvents =
            processSyncPassageEvents(request.getPassageEvents(),
            request.getOrg());
        }
    } catch (Exception e) {
        logger.error(String.format("Failed to process Enter
Events, orgId == %s", request.getOrgId()), e);
        bError = true;
    }
    Date syncEndTime = new Date();
    runRegularPaymentsIfEnabled(request);
    try {
        accountsRegistry =
        RuntimeContext.getAppContext().getBean(AccountsRegistryHandler.c
lass).accRegistryHandler(request, request.getOrgId());
    } catch (Exception e) {
        logger.error(String.format("Failed to build
AccountsRegistry, orgId == %s", request.getOrgId()), e);
    }
    updateOrgSyncDate(request.getOrgId());
    return new SyncResponse(request.getSyncType(),
request.getOrgId(), request.getOrg().getShortName(),
        request.getOrg().getType(), "", packetId, null,
syncEndTime, "", accRegistry, resPaymentRegistry,
resAccountOperationsRegistry, accIncRegistry, clientRegistry,
        resOrgStructure, "", resPassageEvents, manager,
orgOwnerData, accRegistryUpdate, accountsRegistry,
        organizationStructure, responseSections,
orgSettingSection);
    }

    private void updateOrgSyncDate(long orgId) {
        try {
            OrgSyncWritableRepository orgSyncWritableRepository
= OrgSyncWritableRepository.getInstance();
            orgSyncWritableRepository.updateAccRegistryDate(orgId);
        } catch (Exception e) {

```

```

        logger.error("Не удалось обновить время
синхронизации, orgId: " + orgId, e);
    }
}

private void updateFullSyncParam(long orgId) {
    Session persistenceSession = null;
    Transaction persistenceTransaction = null;
    try {
        persistenceSession =
persistenceSessionFactory.openSession();
        persistenceTransaction =
persistenceSession.beginTransaction();
        falseFullSyncByOrg(persistenceSession, orgId);
        setValueForClientsSyncByOrg(persistenceSession,
orgId, Boolean.FALSE);
        setValueForOrgSettingsSyncByOrg(persistenceSession,
orgId, Boolean.FALSE);
        persistenceTransaction.commit();
        persistenceTransaction = null;
    } finally {
        HibernateUtils.rollback(persistenceTransaction,
logger);
        HibernateUtils.close(persistenceSession, logger);
    }
}

private void discardClientSyncParam(Long orgId) {
    Session persistenceSession = null;
    Transaction persistenceTransaction = null;
    try {
        persistenceSession =
persistenceSessionFactory.openSession();
        persistenceTransaction =
persistenceSession.beginTransaction();
        setValueForClientsSyncByOrg(persistenceSession,
orgId, Boolean.FALSE);
        persistenceTransaction.commit();
        persistenceTransaction = null;
    } finally {
        HibernateUtils.rollback(persistenceTransaction,
logger);
        HibernateUtils.close(persistenceSession, logger);
    }
}

```

```

    }

    private void discardOrgSettingsSyncParam(Long orgId) {
        Session persistenceSession = null;
        Transaction persistenceTransaction = null;
        try {
            persistenceSession =
persistenceSessionFactory.openSession();
            persistenceTransaction =
persistenceSession.beginTransaction();
            setValueForOrgSettingsSyncByOrg(persistenceSession,
orgId, Boolean.FALSE);
            persistenceTransaction.commit();
            persistenceTransaction = null;
        } finally {
            HibernateUtils.rollback(persistenceTransaction,
logger);
            HibernateUtils.close(persistenceSession, logger);
        }
    }

    private ResPaymentRegistry processSyncPaymentRegistry(Long
idOfSync, Long orgId, PaymentRegistry paymentRegistry,
        List<Long> errorClientIds) throws Exception {
        ResPaymentRegistry resPaymentRegistry = new
ResPaymentRegistry();
        List<Long> allocatedClients =
ClientManager.getAllocatedClientsIds(orgId);
        Iterator<Payment> payments =
paymentRegistry.getPayments();
        while (payments.hasNext()) {
            Payment Payment = payments.next();
            ResPaymentRegistryItem resAcc;
            try {
                resAcc =
processSyncPaymentRegistryPayment(idOfSync, orgId, Payment,
errorClientIds, allocatedClients);
                if (resAcc.getResult() != 0) {
                    logger.error("Failure in response payment
registry: " + resAcc);
                }
            } catch (Exception e) {
                logger.error(String.format("Failed to process
payment == %s", Payment), e);
            }
        }
    }

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.18.001 ПЗ

Лист

94

```

        resAcc = new
ResPaymentRegistryItem(Payment.getOrderId(), 100, "Internal
error");
    }
    resPaymentRegistry.addItem(resAcc);
}
return resPaymentRegistry;
}

private OrgOwnerData processOrgOwnerData(Long orgId) throws
Exception {
    Session persistenceSession = null;
    Transaction persistenceTransaction = null;
    OrgOwnerData orgOwnerData = null;
    try {
        persistenceSession =
persistenceSessionFactory.openSession();
        persistenceTransaction =
persistenceSession.beginTransaction();
        AbstractProcessor processor = new
OrgOwnerProcessor(persistenceSession, orgId);
        orgOwnerData = (OrgOwnerData) processor.process();
        persistenceTransaction.commit();
        persistenceTransaction = null;
    } finally {
        HibernateUtils.rollback(persistenceTransaction,
logger);
        HibernateUtils.close(persistenceSession, logger);
    }
    return orgOwnerData;
}

public static List<Client> getMigrants(Long orgId) {
    List<Client> clients = null;
    try {
        Session persistenceSession = null;
        Transaction persistenceTransaction = null;
        try {
            persistenceSession =
RuntimeContext.getInstance().createReportPersistenceSession();
            persistenceTransaction =
persistenceSession.beginTransaction();

```

```

        clients =
MigrantsUtils.getActiveMigrantsForOrg(persistenceSession,
orgId);

        persistenceTransaction.commit();
        persistenceTransaction = null;
    } finally {
        HibernateUtils.rollback(persistenceTransaction,
logger);

        HibernateUtils.close(persistenceSession,
logger);
    }
    } catch (Exception e) {
        String message = String.format("processMigrants:
%s", e.getMessage());
        logger.error(message, e);
    }
    return clients;
}

public ResPaymentRegistryItem
processSyncPaymentRegistryPayment(Long idOfSync, Long orgId,
Payment payment,
List<Long> errorClientIds, List<Long>
allocatedClients) throws Exception {
    Session persistenceSession = null;
    Transaction persistenceTransaction = null;
    try {
        persistenceSession =
persistenceSessionFactory.openSession();
        persistenceTransaction =
persistenceSession.beginTransaction();
        orgId = DAOService.getInstance()
            .receiveorgIdByAccessory(orgId,
Accessory.BANK_ACCESSORY_TYPE, payment.getIdOfPOS());
        Long orgIdanization = getorgId(persistenceSession,
orgId);

        if (null == orgIdanization) {
            return new
ResPaymentRegistryItem(payment.getOrderId(), 130,
                String.format("Organization not found,
orgId == %s, OrderId == %s", orgId, payment.getOrderId()));
        }
        if (payment.getIdOfCashier() == null) {

```

```

        return new
ResPaymentRegistryItem(payment.getOrderId(), 201,
                        String.format("Cashier attribute not
found, orgId == %s, OrderId == %s", orgId,
payment.getOrderId())));
    }
    Long orgIdPayment = payment.getorgId();
    boolean isFromFriendlyOrg = false;
    if (null != orgIdPayment) {
        Long orgIdanizationPayment =
getorgId(persistenceSession, orgIdPayment);
        if (null == orgIdanizationPayment) {
            return new
ResPaymentRegistryItem(payment.getOrderId(), 130,
                        String.format("Organization not
found, orgId == %s, OrderId == %s", orgIdPayment,
payment.getOrderId())));
        }
        if
(!DAOService.getInstance().isOrgFriendly(orgId, orgIdPayment)) {
            return new
ResPaymentRegistryItem(payment.getOrderId(), 150,
                        String.format("Organization is not
friendly, orgId == %s, OrderId == %s",orgIdPayment,
payment.getOrderId())));
        }
        isFromFriendlyOrg = true;
        long temp = orgId;
        orgId = orgIdPayment;
        orgIdPayment = temp;
    }
    CompositeOrderId compositeOrderId = new
CompositeOrderId(orgId, payment.getOrderId());
    Order order = findOrder(persistenceSession,
compositeOrderId);
    if (payment.isCommit()) {
        if (order != null) {
            Long orderCardNo = order.getCard() == null ?
null : order.getCard().getCardNo();
            if ((("" + orderCardNo).equals("" +
payment.getCardNo())) && (order.getCreateTime()
.equals(payment.getTime())) &&
(order.getSumByCard().equals(payment.getSumByCard())) {

```

```

        return new
ResPaymentRegistryItem(payment.getOrderId(), 0, "Order is
already registered");
    } else {
        return new
ResPaymentRegistryItem(payment.getOrderId(), 110,
String.format("Order already registered but attributes differ,
orgId == %s, OrderId == %s",orgId, payment.getOrderId()));
    }
}
Card card = null;
Long cardNo = payment.getCardNo();
if (null != cardNo) {
    card =
findCardByCardNoExtended(persistenceSession, cardNo,
payment.getIdOfClient(), null, null);
    if (null == card) {
        return new
ResPaymentRegistryItem(payment.getOrderId(), 200,
String.format("Unknown card,
orgId == %s, OrderId == %s, CardNo == %s", orgId,
payment.getOrderId(),
cardNo));
    }
}
Client client = null;
Long idOfClient = payment.getIdOfClient();
if (null != idOfClient) {
    client = findClient(persistenceSession,
idOfClient);
    if (null == client) {
        return new
ResPaymentRegistryItem(payment.getOrderId(), 210,
String.format("Unknown client,
orgId == %s, OrderId == %s, IdOfClient == %s",orgId,
payment.getOrderId(), idOfClient));
    }
}
if (null != card) {
    if (null == client) {
        client = card.getClient();
    }
}
}

```

```

        Set<Long> idOfFriendlyOrgSet =
getIdOfFriendlyOrg(persistenceSession, orgId);
        if (null != client) {
            Org clientOrg = client.getOrg();
            if (!clientOrg.getOrgId().equals(orgId) &&
!idOfFriendlyOrgSet.contains(clientOrg.getOrgId())) {
                if
(! (MigrantsUtils.getActiveMigrantsByIdOfClient(persistenceSessio
n, client.getIdOfClient()).size() > 0)) {
                    if
(!allocatedClients.contains(client.getIdOfClient())) {
                        errorClientIds.add(idOfClient);
                    }
                }
            }
        }
        if (payment.getSumByCard() < 0 ||
payment.getSumByCash() < 0 || payment.getRSum() < 0 ||
payment.getGrant() < 0) {
            return new
ResPaymentRegistryItem(payment.getOrderId(), 250,
                        String.format("Negative sum(s) are
specified, orgId == %s, OrderId == %s", orgId,
                        payment.getOrderId()));
        }
        long totalPurchaseDiscount = 0;
        long totalPurchaseRSum = 0;
        long totalLunchRSum = 0;
        Set<String> rations = new HashSet<>();
        PreorderComplex preorderComplex =
findPreorderComplexByPayment(persistenceSession, payment);
        boolean saveAllPreorderDetails =
((preorderComplex != null) && preorderComplex.getModeOfAdd()
.equals(PreorderComplex.COMPLEX_MODE_4));
        for (Purchase purchase : payment.getPurchases())
        {
            if (null !=
findOrderDetail(persistenceSession, new
CompositeOrderIdDetail(orgId, purchase.getOrderIdDetail())) {
                return new
ResPaymentRegistryItem(payment.getOrderId(), 120,
String.format("Order detail is already registered, orgId == %s,
OrderId == %s, OrderIdDetail == %s",orgId, payment.getOrderId(),
purchase.getOrderIdDetail()));
            }
        }
    }

```

```

        }
        if (purchase.getDiscount() < 0 ||
purchase.getrPrice() < 0 /*|| purchase.getQty() < 0*/) {
            return new
ResPaymentRegistryItem(payment.getOrderId(), 250,
String.format("Negative Discount or rPrice are specified, orgId
== %s, OrderId == %s, Discount == %s, Discount == %s",
                orgId, payment.getOrderId(),
purchase.getDiscount(), purchase.getrPrice()));
        }
        OrderDetail orderDetail = new OrderDetail(
            new CompositeOrderIdDetail(orgId,
purchase.getOrderIdDetail()), payment.getOrderId(),
purchase.getQty(), purchase.getDiscount(), purchase.getrPrice(),
purchase.getName());

            payment.getIdOfClient() == null,
            purchase.getIdOfRule(),
OrderDetailFRationType.fromInteger(purchase.getfRation()));
            if (purchase.getGuidOfGoods() != null) {
                Good good =
findGoodByGuid(persistenceSession, purchase.getGuidOfGoods());
                if (good != null) {
                    orderDetail.setGood(good);
                }
            }
            if (saveAllPreorderDetails ||
purchase.getGuidPreOrderDetail() != null) {
                savePreorderGuidFromOrderDetail(persistenceSession,
purchase.getGuidPreOrderDetail(), orderDetail, false,
preorderComplex, purchase.getItemCode());
            }
            persistenceSession.save(orderDetail);
            totalPurchaseDiscount +=
purchase.getDiscount() * Math.abs(purchase.getQty());
            totalPurchaseRSum += purchase.getrPrice() *
Math.abs(purchase.getQty());
            if (orderDetail.isComplex() ||
orderDetail.isComplexItem()) {
                totalLunchRSum += purchase.getrPrice() *
Math.abs(purchase.getQty());
            }
            if (purchase.getfRation() != null &&
OrderDetailFRationType.fromInteger(purchase.getfRation()) !=
OrderDetailFRationType.NOT_SPECIFIED) {

```

```

rations.add(OrderDetailFRationType.fromInteger(purchase.getfRati
on()).toString());
        }
    }
    if (totalPurchaseRSum != payment.getRSum()
        || totalPurchaseDiscount !=
payment.getTrdDiscount() + payment
        .getGrant()) {
        return new
ResPaymentRegistryItem(payment.getOrderId(), 300,
                        String.format("Invalid total sum by
order, orgId == %s, OrderId == %s", orgId,
payment.getOrderId()));
    }
    if (payment.getRSum() != payment.getSumByCard()
+ payment.getSumByCash() + payment.getSummFromCBHR()) {
        return new
ResPaymentRegistryItem(payment.getOrderId(), 310,
                        String.format("Invalid sum of order
card and cash payments, orgId == %s, OrderId == %s",orgId,
payment.getOrderId()));
    }
    persistenceSession.flush();
    persistenceTransaction.commit();
    persistenceTransaction = null;
    String[] values =
generatePaymentNotificationParams(persistenceSession, client,
payment);
    if
(payment.getOrderType().equals(OrderTypeEnumType.UNKNOWN) ||
payment.getOrderType().equals(OrderTypeEnumType.DEFAULT) ||
payment.getOrderType().equals(OrderTypeEnumType.VENDING)) {
        values =
EventNotificationService.attachToValues("isBarOrder", "true",
values);
    } else if
(payment.getOrderType().equals(OrderTypeEnumType.PAY_PLAN) ||
payment.getOrderType()

.equals(OrderTypeEnumType.SUBSCRIPTION_FEEDING)) {
        values =
EventNotificationService.attachToValues("isPayOrder", "true",
values);
    }
}

```

```

        } else if
(payment.getOrderType().equals(OrderTypeEnumType.REDUCED_PRICE_PL
LAN) || payment

.getOrderType().equals(OrderTypeEnumType.DAILY_SAMPLE) ||
payment.getOrderType().equals(OrderTypeEnumType.REDUCED_PRICE_PL
AN_RESERVE) ||
payment.getOrderType().equals(OrderTypeEnumType.CORRECTION_TYPE)
||
payment.getOrderType().equals(OrderTypeEnumType.WATER_ACCOUNTING
) ||
payment.getOrderType().equals(OrderTypeEnumType.RECYCLING_RETION
S)) {
            values =
EventNotificationService.attachToValues("isFreeOrder", "true",
values);
        }
        if (rations.size() > 0) {
            values = EventNotificationService
.attachToValues(EventNotificationService.PARAM_FRATION,
StringUtils.join(rations, ","), values);
        }
        String date = new
SimpleDateFormat("dd.MM.yyyy HH:mm").format(payment.getTime());
        values =
EventNotificationService.attachToValues(EventNotificationService
.PARAM_ORDER_EVENT_TIME, date, values);
        values = EventNotificationService
.attachToValues(EventNotificationService.PARAM_COMPLEX_NAME,
getComplexName(payment), values);
        values =
EventNotificationService.attachTargetIdToValues(payment.getOrder
Id(), values);
        values = EventNotificationService
.attachSourceOrgIdToValues(orgId,
values);
        long totalBuffetRSum = totalPurchaseRSum -
totalLunchRSum;
        long totalRSum = totalBuffetRSum +
totalLunchRSum;
        long totalAmountBuyAll = totalBuffetRSum +
totalLunchRSum;
        values =
EventNotificationService.attachToValues(EventNotificationService

```

					ПГУ 2.090301.18.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		102

```

.PARAM_AMOUNT_PRICE, Long.toString(totalBuffetRSum / 100) + ',' +
+ Long.toString(totalBuffetRSum % 100), values);
        values =
EventNotificationService.attachToValues(EventNotificationService
.PARAM_AMOUNT_LUNCH, Long.toString(totalLunchRSum / 100) + ',' +
Long.toString(totalLunchRSum % 100), values);
        values =
EventNotificationService.attachToValues(EventNotificationService
.PARAM_AMOUNT, Long.toString(totalRSum / 100) + ',' +
Long.toString(totalRSum % 100), values);
        values =
EventNotificationService.attachAmountBuyAllToValues(totalAmountBuyAll, values);
    }
}
} else {
    if (null != order) {
        if (payment.getIdOfClient() != null) {
            Client client =
DAOService.getInstance().findClientById(payment.getIdOfClient())
;
        }
RuntimeContext.getFinancialOpsManager().cancelOrder(persistenceSession, order, payment);
        persistenceSession.flush();
        persistenceTransaction.commit();
        persistenceTransaction = null;
    } else {
        return new
ResPaymentRegistryItem(payment.getOrderId(), 0,
                        String.format("Unknown order, orgId
== %s, OrderId == %s", orgId, payment.getOrderId()));
    }
}
return new
ResPaymentRegistryItem(payment.getOrderId(), 0, null);
} finally {
    HibernateUtils.rollback(persistenceTransaction,
logger);
    HibernateUtils.close(persistenceSession, logger);
}

}

private String getComplexName(Payment payment) {
    if (payment.getPurchases() == null) {

```

```

        return "";
    }
    for (Purchase purchase : payment.getPurchases()) {
        if (purchase.getType() != null && purchase.getType()
> 0 && purchase.getType() < 100) {
            return purchase.getName();
        }
    }
    return "";
}

```

```

private Long getOrderNotificationType(String[] values)
throws Exception {
    if
(EventNotificationService.findBooleanValueInParams(new
String[]{"isBarOrder"}, values)) {
        return;
    } else {
        throw new Exception("Не определен тип события");
    }
}

```

```

private void processSyncClientParamRegistry(SyncHistory
syncHistory, Long orgId,
SyncRequest.ClientParamRegistry clientParamRegistry,
List<Long> errorClientIds,
List<Long> clientsWithWrongVersion) throws Exception
{
    Session persistenceSession = null;
    Transaction persistenceTransaction = null;
    try {
        persistenceSession =
persistenceSessionFactory.openSession();
        persistenceTransaction =
persistenceSession.beginTransaction();
        Iterator<SyncRequest.ClientParamRegistry.ClientParamItem>
clientParamItems = clientParamRegistry.getPayments().iterator();
        List<Long> allocatedClients =
ClientManager.getAllocatedClientsIds(persistenceSession, orgId);
        HashMap<Long, HashMap<String, ClientGroup>> orgMap =
new HashMap<Long, HashMap<String, ClientGroup>>();
        Org org = (Org) persistenceSession.get(Org.class,
orgId);
        Set<Org> orgSet = org.getFriendlyOrg();
    }
}

```

```

        orgSet.add(org);
        for (Org o : orgSet) {
            List clientGroups =
getClientGroupsByorgId(persistenceSession, o.getOrgId());
            HashMap<String, ClientGroup> nameIdGroupMap =
new HashMap<String, ClientGroup>();
            for (Object object : clientGroups) {
                ClientGroup clientGroup = (ClientGroup)
object;
nameIdGroupMap.put(clientGroup.getGroupName(), clientGroup);

orgMap.put(clientGroup.getCompositeIdOfClientGroup().getorgId(),
nameIdGroupMap);
            }
            orgMap.put(o.getOrgId(), nameIdGroupMap);
        }
        Long version = null;
        if (clientParamItems.hasNext()) {
            version =
updateClientRegistryVersion(persistenceSession);
        }
        persistenceTransaction.commit();
        persistenceTransaction = null;
        while (clientParamItems.hasNext()) {
            SyncRequest.ClientParamRegistry.ClientParamItem
clientParamItem = clientParamItems.next();
            Client client =
DAOUtils.findClient(persistenceSession,
clientParamItem.getIdOfClient());
            disableClientCardsIfChangeOrg(client, orgSet,
orgId);

            try {
processSyncClientParamRegistryItem(clientParamItem, orgMap,
version, errorClientIds, orgId, allocatedClients, orgSet,
clientsWithWrongVersion);
            } catch (Exception e) {
                String message = String.format("Failed to
process clientParamItem == %s", orgId);
                if (syncHistory != null) {
processorUtils.createSyncHistoryException(persistenceSessionFact
ory, orgId, syncHistory, message);
                }
                logger.error(message, e);
            }
        }
    }

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.18.001 ПЗ

Лист

105

```

        }
    } finally {
        HibernateUtils.rollback(persistenceTransaction,
logger);
        HibernateUtils.close(persistenceSession, logger);
    }
}

private void
processSyncClientParamRegistryItem(SyncRequest.ClientParamRegist
ry.ClientParamItem clientParamItem, HashMap<Long,
HashMap<String, ClientGroup>> orgMap, Long version, List<Long>
errorClientIds, Long orgId,
List<Long> allocatedClients, Set<Org> orgSet,
List<Long> clientsWithWrongVersion) throws Exception {
boolean ignoreNotifyFlags =
RuntimeContext.getInstance().getSmsService().ignoreNotifyFlags()
;

    Session persistenceSession = null;
    Transaction persistenceTransaction = null;
    try {
        persistenceSession =
persistenceSessionFactory.openSession();
        persistenceTransaction =
persistenceSession.beginTransaction();
        Client client = findClient(persistenceSession,
clientParamItem.getIdOfClient());
        if (clientParamItem.getVersion() != null &&
clientParamItem.getVersion() <
client.getClientRegistryVersion()) {
clientsWithWrongVersion.add(client.getIdOfClient());
            return;
        }

        if
(!orgMap.keySet().contains(client.getOrg().getOrgId())) {
            if
(! (MigrantsUtils.getActiveMigrantsByIdOfClient(persistenceSessio
n, clientParamItem.getIdOfClient())
.size() > 0)) {
                if
(!allocatedClients.contains(clientParamItem.getIdOfClient())) {
                    errorClientIds.add(client.getIdOfClient());
                    throw new IllegalArgumentException(

```

```

        "Client from another
organization. idOfClient=" + client.getIdOfClient().toString()
        + ", orgId=" +
orgId.toString() + ", clientParamItem=" + clientParamItem
        .toString());
    }
}
} else {
    Long orgOwner = clientParamItem.getOrgOwner();
    boolean changeOrg = false;
    if (orgOwner != null) {
        Org org = (Org)
persistenceSession.get(Org.class, orgOwner);
        changeOrg =
!client.getOrg().getOrgId().equals(org.getOrgId());
        client.setOrg(org);
    }
    client.setFreePayCount(clientParamItem.getFreePayCount());
    client.setFreePayMaxCount(clientParamItem.getFreePayMaxCount());
    client.setLastFreePayTime(clientParamItem.getLastFreePayTime());
    client.setDisablePlanCreationDate(clientParamItem.getDisablePlan
CreationDate());
    client.setDisablePlanEndDate(clientParamItem.getDisablePlanEndDa
te());

        if (clientParamItem.getExpenditureLimit() !=
null) {

client.setExpenditureLimit(clientParamItem.getExpenditureLimit()
);
        }
        if (clientParamItem.getAddress() != null) {
client.setAddress(clientParamItem.getAddress());
        }
        if
(!RuntimeContext.getInstance().getOptionValueBool(Option.OPTION_
DISABLE_EMAIL_EDIT) && clientParamItem.getEmail() != null) {
            String email = clientParamItem.getEmail();
            if (client != null && client.getEmail() !=
null && !client.getEmail().equals(email)) {
                client.setSsoid("");
            }
            client.setEmail(email);
            if
(!StringUtils.isEmpty(clientParamItem.getEmail()))

```

```

        &&
clientParamItem.getNotifyViaEmail() == null) {
            client.setNotifyViaEmail(true);
        }
    }
    if (clientParamItem.getMobilePhone() != null) {
        String mobile =
Client.checkAndConvertMobile(clientParamItem.getMobilePhone());
        if (client != null && client.getMobile() !=
null && !client.getMobile().equals(mobile)) {
            client.setSsoid("");
        }
        client.setMobile(mobile);
        logger.info("class : ClientManager, method :
modifyClientTransactionFree line : 344, idOfClient : "
+ client.getIdOfClient() + " mobile
: " + client.getPhone());
        if (!StringUtils.isEmpty(mobile)) {
            if (clientParamItem.getNotifyViaSMS() ==
null) {
                client.setNotifyViaSMS(true);
            }
            if (clientParamItem.getNotifyViaPUSH()
== null) {
                client.setNotifyViaPUSH(false);
            }
        }
    }
    if (clientParamItem.getMiddleGroup() != null) {
client.setMiddleGroup(clientParamItem.getMiddleGroup());
    }
    if (clientParamItem.getGender() != null) {
client.setGender(clientParamItem.getGender());
    }
    if (clientParamItem.getBirthDate() != null) {
client.setBirthDate(clientParamItem.getBirthDate());
    }
    if (clientParamItem.getName() != null) {
client.getPerson().setFirstName(clientParamItem.getName());
    }
    if (clientParamItem.getPhone() != null) {
        client.setPhone(clientParamItem.getPhone());
    }
}

```

```

        if (clientParamItem.getSecondName() != null) {
client.getPerson().setSecondName(clientParamItem.getSecondName()
);
        }
        if (clientParamItem.getSurname() != null) {
client.getPerson().setSurname(clientParamItem.getSurname());
        }
        if (clientParamItem.getRemarks() != null) {
client.setRemarks(clientParamItem.getRemarks());
        }
        if (clientParamItem.getSan() != null) {
            client.setSan(clientParamItem.getSan());
        }
        if (!ignoreNotifyFlags) {
            if (clientParamItem.getNotifyViaEmail() !=
null) {
client.setNotifyViaEmail(clientParamItem.getNotifyViaEmail());
            }
            if (clientParamItem.getNotifyViaSMS() !=
null) {

client.setNotifyViaSMS(clientParamItem.getNotifyViaSMS());
            }
            if (clientParamItem.getNotifyViaPUSH() !=
null) {
client.setNotifyViaPUSH(clientParamItem.getNotifyViaPUSH());
            }
        }
        if (clientParamItem.getFax() != null) {
            client.setFax(clientParamItem.getFax());
        }

        if (clientParamItem.getCanConfirmGroupPayment()
!= null) {
client.setCanConfirmGroupPayment(clientParamItem.getCanConfirmGr
oupPayment());
        }
        if
(clientParamItem.getConfirmVisualRecognition() != null) {
client.setConfirmVisualRecognition(clientParamItem.getConfirmVis
ualRecognition());
        }
        if
(StringUtils.isEmpty(clientParamItem.getGroupName())) {

```

```

        ClientGroup clientGroup;
        if (changeOrg) {
            clientGroup =
findClientGroupByGroupNameAndorgId(persistenceSession,
                                     client.getOrg().getOrgId(),
clientParamItem.getGroupName());
        } else {
            clientGroup =
orgMap.get(client.getOrg().getOrgId()).get(clientParamItem.getGr
oupName());
        }
        if (clientGroup == null) {
            clientGroup =
createClientGroup(persistenceSession,
client.getOrg().getOrgId(),clientParamItem.getGroupName());
orgMap.get(client.getOrg().getOrgId()).put(clientGroup.getGroupN
ame(), clientGroup);
        }
        if (client.getClientGroup() == null ||
!clientGroup.equals(client.getClientGroup())) {
ClientManager.createClientGroupMigrationHistory(persistenceSessi
on, client,
client.getOrg(),clientGroup.getCompositeIdOfClientGroup().getIdO
fClientGroup(), clientGroup.getGroupName(),

ClientGroupMigrationHistory.MODIFY_IN_CSS.concat(String.format("
(ид. 00=%s)", orgId)));
        }
        client.setClientGroup(clientGroup);
client.setIdOfClientGroup(clientGroup.getCompositeIdOfClientGrou
p().getIdOfClientGroup());
    }

    if (clientParamItem.getIsUseLastEEModeForPlan()
!= null) {
client.setUseLastEEModeForPlan(clientParamItem.getIsUseLastEEMod
eForPlan());
    }
    if (clientParamItem.getPassportNumber() != null)
{
client.setPassportNumber(clientParamItem.getPassportNumber());
    }
    if (clientParamItem.getPassportSeries() != null)
{

```

```

client.setPassportSeries(clientParamItem.getPassportSeries());
    }
    }
    client.setClientRegistryVersion(version);
    persistenceSession.update(client);
    persistenceSession.flush();
    persistenceTransaction.commit();
    persistenceTransaction = null;
} catch (Exception e) {
    logger.debug(e.getMessage(), e);
} finally {
    HibernateUtils.rollback(persistenceTransaction,
logger);
    HibernateUtils.close(persistenceSession, logger);
}
}

private SyncResponse.ResOrgStructure
processSyncOrgStructure(Long orgId, SyncRequest.OrgStructure
reqStructure, SyncHistory syncHistory) throws Exception {
    Session persistenceSession = null;
    Transaction persistenceTransaction = null;
    try {
        persistenceSession =
persistenceSessionFactory.openSession();
        persistenceTransaction =
persistenceSession.beginTransaction();
        Org organization = findOrg(persistenceSession,
orgId);
        List<ClientGroup> superfluousClientGroups = new
ArrayList<ClientGroup>();
        for (ClientGroup clientGroup :
organization.getClientGroups()) {
            if (clientGroup.isTemporaryGroup()) {
                if
(findClientGroupByName(clientGroup.getGroupName(),
reqStructure)) {
                    superfluousClientGroups.add(clientGroup);
                }
            } else {
                if
(!findClientGroupById(clientGroup.getCompositeIdOfClientGroup().
getIdOfClientGroup(), reqStructure)) {
                    superfluousClientGroups.add(clientGroup);
                }
            }
        }
    }
}

```

```

        }
    }
}
for (ClientGroup clientGroup :
superfluousClientGroups) {
    organization.removeClientGroup(clientGroup);
    for (Client client : clientGroup.getClients()) {
        client.setIdOfClientGroup(null);
        client.setClientGroup(null);
        client.setUpdateTime(new Date());
        persistenceSession.update(client);
    }
    persistenceSession.delete(clientGroup);
}
for (SyncRequest.OrgStructure.Group reqGroup :
reqStructure.getGroups()) {
    try {
processSyncOrgStructureGroup(persistenceSession, organization,
reqGroup);
    } catch (Exception e) {
        String message = String.format("Failed to
process: %s", reqGroup);
        logger.error(message, e);
        return new SyncResponse.ResOrgStructure(2,
message);
    }
}
persistenceTransaction.commit();
persistenceTransaction = null;
return new SyncResponse.ResOrgStructure();
} finally {
    HibernateUtils.rollback(persistenceTransaction,
logger);
    HibernateUtils.close(persistenceSession, logger);
}
}

private static boolean findClientGroupById(long
idOfClientGroup, SyncRequest.OrgStructure reqStructure)
throws Exception {
    for (SyncRequest.OrgStructure.Group group :
reqStructure.getGroups()) {
        if (idOfClientGroup == group.getIdOfGroup()) {
            return true;
        }
    }
}

```

```

        }
    }
    return false;
}

private static boolean findClientGroupByName(String
groupName, SyncRequest.OrgStructure reqStructure)
throws Exception {
    for (SyncRequest.OrgStructure.Group group :
reqStructure.getGroups()) {
        if (groupName.equals(group.getName())) {
            return true;
        }
    }
    return false;
}

private void processSyncOrgStructureGroup(Session
persistenceSession, Org organization,
SyncRequest.OrgStructure.Group reqGroup) throws
Exception {
    CompositeIdOfClientGroup compositeIdOfClientGroup = new
CompositeIdOfClientGroup(organization.getOrgId(), reqGroup.getIdO
fGroup());
    ClientGroup clientGroup =
findClientGroup(persistenceSession, compositeIdOfClientGroup);
    if (null != clientGroup) {
        if (!StringUtils.equals(reqGroup.getName(),
clientGroup.getGroupName())) {
            clientGroup.setGroupName(reqGroup.getName());
            persistenceSession.update(clientGroup);
        }
    } else {
        clientGroup = new
ClientGroup(compositeIdOfClientGroup, reqGroup.getName());
        persistenceSession.save(clientGroup);
    }
    for (Long idOfClient : reqGroup.getClients()) {
        Long idOfClientGroup =
getClientGroup(persistenceSession, idOfClient,
organization.getOrgId());
        if (idOfClientGroup != null &&
(idOfClientGroup.longValue() ==

```

```

clientGroup.getCompositeIdOfClientGroup().getIdOfClientGroup().l
ongValue())) {
    continue;
}
Client client = findClient(persistenceSession,
idOfClient);
Set<Long> idOfFriendlyOrgSet =
getIdOfFriendlyOrg(persistenceSession,
client.getOrg().getOrgId());
if (null == client) {
    logger.info(String.format("Client with
IdOfClient == %s not found", idOfClient));
} else if
(!client.getOrg().getOrgId().equals(organization.getOrgId()) &&
!idOfFriendlyOrgSet.contains(client.getOrg().getOrgId())) {
    logger.error(String.format(
        "Client with IdOfClient == %s belongs to
other organization. Client: %s, orgId by request: %s",
        idOfClient, client,
organization.getOrgId()));
} else if
(client.getOrg().getOrgId().equals(organization.getOrgId())) {
    client.setClientGroup(clientGroup);
client.setIdOfClientGroup(clientGroup.getCompositeIdOfClientGrou
p().getIdOfClientGroup());
    client.setUpdateTime(new Date());
    persistenceSession.update(client);
}
}
}

private void addClientVersionAndRemoteAddressByOrg(Long
orgId, String clientVersion, String remoteAddress,
String sqlServerVersion, Double databaseSize) {
    Session persistenceSession = null;
    Transaction persistenceTransaction = null;
    try {
        persistenceSession =
persistenceSessionFactory.openSession();
        persistenceTransaction =
persistenceSession.beginTransaction();
        updateClientVersionAndRemoteAddressByOrg(persistenceSession,
orgId, clientVersion, remoteAddress,
sqlServerVersion, databaseSize);
    }
}

```

```

        persistenceTransaction.commit();
        persistenceTransaction = null;
    } catch (Exception e) {
        logger.error("Can't update ClientVersion,
RemoteAddress and sqlServerVersion for ID of Org: " + orgId, e);
    } finally {
        HibernateUtils.rollback(persistenceTransaction,
logger);
        HibernateUtils.close(persistenceSession, logger);
    }
}

    public SyncHistory createSyncHistory(Long orgId, Long
packetId, Date startTime, String clientVersion,
        String remoteAddress, Integer syncType) throws
Exception {
        Session persistenceSession = null;
        Transaction persistenceTransaction = null;
        try {
            persistenceSession =
persistenceSessionFactory.openSession();
            persistenceTransaction =
persistenceSession.beginTransaction();
            Org organization =
getOrgReference(persistenceSession, orgId);
            SyncHistory syncHistory = new
SyncHistory(organization, startTime, packetId, clientVersion,
remoteAddress, syncType);
            persistenceSession.save(syncHistory);
            persistenceSession.flush();
            persistenceTransaction.commit();
            persistenceTransaction = null;
            return syncHistory;
        } finally {
            HibernateUtils.rollback(persistenceTransaction,
logger);
            HibernateUtils.close(persistenceSession, logger);
        }
    }

    private SyncResponse.AccRegistry getAccRegistry(Long orgId,
List<Long> clientIds, String clientVerision)
        throws Exception {
        if (SyncRequest.versionIsAfter(clientVerision, "2.7")) {

```

```

        return null;
    }
    SyncResponse.AccRegistry accRegistry = new
SyncResponse.AccRegistry();
    Session persistenceSession = null;
    Transaction persistenceTransaction = null;
    try {
        persistenceSession =
persistenceSessionFactory.openSession();
        persistenceTransaction =
persistenceSession.beginTransaction();
        Set<Long> orgIdSet = new HashSet<Long>();
        Org org = (Org) persistenceSession.get(Org.class,
orgId);
        Set<Org> orgSet = org.getFriendlyOrg();
        for (Org o : orgSet) {
            orgIdSet.add(o.getOrgId());
        }
        orgIdSet.add(orgId);
        List<Card> cards =
getClientsAndCardsForOrgs(persistenceSession, orgIdSet,
clientIds);
        for (Card card : cards) {
            accRegistry.addItem(new
SyncResponse.AccRegistry.Item(card));
        }
        if (clientIds == null || clientIds.isEmpty()) {
            List<Client> allocClients =
ClientManager.findAllAllocatedClients(persistenceSession, org);
            for (Client client : allocClients) {
                for (Card card : client.getCards()) {
                    accRegistry.addItem(new
SyncResponse.AccRegistry.Item(client, card));
                }
            }
        }
        List<Client> migrantClients =
MigrantsUtils.getActiveMigrantsForOrg(persistenceSession,
org.getOrgId());
        for (Client client : migrantClients) {
            for (Card card : client.getCards()) {
                accRegistry.addItem(new
SyncResponse.AccRegistry.Item(client, card));
            }
        }
    }

```

```

        }
        persistenceTransaction.commit();
        persistenceTransaction = null;
    } finally {
        HibernateUtils.rollback(persistenceTransaction,
logger);
        HibernateUtils.close(persistenceSession, logger);
    }
    return accRegistry;
}

private SyncResponse.AccRegistry
getAccRegistryForMigrants(Long orgId) throws Exception {
    SyncResponse.AccRegistry accRegistry = new
SyncResponse.AccRegistry();
    Session persistenceSession = null;
    Transaction persistenceTransaction = null;
    try {
        persistenceSession =
persistenceSessionFactory.openSession();
        persistenceTransaction =
persistenceSession.beginTransaction();
        Org org = (Org) persistenceSession.get(Org.class,
orgId);

        List<Client> migrantClients =
MigrantsUtils.getActiveMigrantsForOrg(persistenceSession,
org.getOrgId());
        for (Client client : migrantClients) {
            for (Card card : client.getCards()) {
                accRegistry.addItem(new
SyncResponse.AccRegistry.Item(client, card));
            }
        }
        persistenceTransaction.commit();
        persistenceTransaction = null;
    } finally {
        HibernateUtils.rollback(persistenceTransaction,
logger);
        HibernateUtils.close(persistenceSession, logger);
    }
    return accRegistry;
}

```

```

        private AccRegistryUpdate getAccRegistryUpdate(Org org, Date
fromDateTime) throws Exception {
            AccRegistryUpdate accRegistryUpdate = new
AccRegistryUpdate();
            final Date currentDate = new Date();
            List<AccountTransactionExtended> accountTransactionList
= null;
            try {
                Date dateStartDate =
getQueryStartDate(org.getOrgId(), fromDateTime);
                long time_delta = System.currentTimeMillis();
                accountTransactionList =
DAOReadonlyService.getInstance().getAccountTransactionsForOrgSin
ceTimeV2(org, dateStartDate, currentDate);
                time_delta = System.currentTimeMillis() -
time_delta;
                if (time_delta > 10L * 1000L) {
                    logger.error(String.format(
                        "Transactions query time = %s ms. orgId
= %s. Period = %3$td.%3$tm.%3$tY %3$tT - %4$td.%4$tm.%4$tY %4$tT
(date from packet = %5$td.%5$tm.%5$tY %5$tT)",
                        time_delta, org.getOrgId(),
dateStartDate, currentDate, fromDateTime));
                }
                for (AccountTransactionExtended accountTransaction :
accountTransactionList) {
                    accRegistryUpdate.addAccountTransactionInfoV2(accountTransaction
);
                }
            } catch (Exception e) {
                logger.error("AccRegistryUpdate section failed", e);
                accRegistryUpdate.setResult(new ResultOperation(500,
e.getMessage()));
            }
            return accRegistryUpdate;
        }

```

```

        private SyncResponse.AccIncRegistry getAccIncRegistry(Org
org, Date fromDateTime) throws Exception {
            SyncResponse.AccIncRegistry accIncRegistry = new
SyncResponse.AccIncRegistry();
            Session persistenceSession = null;
            Transaction persistenceTransaction = null;
            try {

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.18.001 ПЗ

Лист

118

```

        persistenceSession =
persistenceSessionFactory.openSession();
        persistenceTransaction =
persistenceSession.beginTransaction();
        Date currentDate = new Date();
        List<Integer> transactionSourceTypes = Arrays
.asList(AccountTransaction.PAYMENT_SYSTEM_TRANSACTION_SOURCE_TYP
E, AccountTransaction.CASHBOX_TRANSACTION_SOURCE_TYPE,
AccountTransaction.ACCOUNT_TRANSFER_TRANSACTION_SOURCE_TYPE,
AccountTransaction.CANCEL_TRANSACTION_SOURCE_TYPE);
        persistenceSession.refresh(org);
        List<AccountTransaction> accountTransactionList =
getAccountTransactionsForOrgSinceTime(persistenceSession,
org, fromDateTime, currentDate,
transactionSourceTypes);
        for (AccountTransaction accountTransaction :
accountTransactionList) {
            SyncResponse.AccIncRegistry.Item accIncItem =
new SyncResponse.AccIncRegistry.Item(
                accountTransaction.getIdOfTransaction(),
accountTransaction.getClient().getIdOfClient(),
                accountTransaction.getTransactionTime(),
accountTransaction.getTransactionSum());
            accIncRegistry.addItem(accIncItem);
        }
        accIncRegistry.setDate(currentDate);
        persistenceTransaction.commit();
        persistenceTransaction = null;
    } finally {
        HibernateUtils.rollback(persistenceTransaction,
logger);
        HibernateUtils.close(persistenceSession, logger);
    }
    return accIncRegistry;
}

private SyncResponse.ClientRegistry
processSyncClientRegistry(Long orgId,
SyncRequest.ClientRegistryRequest
clientRegistryRequest, List<Long> errorClientIds,
List<Long> clientsWithWrongVersion) throws Exception
{
    SyncResponse.ClientRegistry clientRegistry = new
SyncResponse.ClientRegistry();

```

```

        List<Client> clients;
        Org organization;
        List<Org> orgList;
        List<Long> activeClientsId;
        Session persistenceSession = null;
        Transaction persistenceTransaction = null;
        try {
            persistenceSession =
RuntimeContext.getInstance().createReportPersistenceSession();
            persistenceTransaction =
persistenceSession.beginTransaction();
            organization = getOrgReference(persistenceSession,
orgId);
            orgList = new
ArrayList<Org>(organization.getFriendlyOrg());
            orgList.add(organization);
            clients = findNewerClients(persistenceSession,
orgList, clientRegistryRequest.getCurrentVersion());
            List<Client> migrants =
MigrantsUtils.getActiveMigrantsForOrg(persistenceSession,
orgId);

            clients.addAll(migrants);
            for (Long idOfClientWithWrongVersion :
clientsWithWrongVersion) {
                Client c = (Client)
persistenceSession.load(Client.class,
idOfClientWithWrongVersion);
                if (!clients.contains(c)) {
                    clients.add(c);
                }
            }
            for (Client client : clients) {
                if (client.getOrg().getOrgId().equals(orgId)) {
                    clientRegistry.addItem(new
SyncResponse.ClientRegistry.Item(client, 0));
                } else {
                    clientRegistry.addItem(new
SyncResponse.ClientRegistry.Item(client, 1));
                }
            }
            activeClientsId =
findActiveClientsId(persistenceSession, orgList);

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.18.001 ПЗ

Лист

120

```

        Map<String, Set<Client>> alienClients =
ClientManager.findAllocatedClients(persistenceSession,
organization);
        for (Map.Entry<String, Set<Client>> entry :
alienClients.entrySet()) {
            boolean isTempClient =
entry.getKey().equals("TemporaryClients");
            for (Client cl : entry.getValue()) {
                if (cl.getClientRegistryVersion() >
clientRegistryRequest.getCurrentVersion()) {
                    clientRegistry.addItem(new
SyncResponse.ClientRegistry.Item(cl, 2, isTempClient));
                }
                activeClientsId.add(cl.getIdOfClient());
            }
        }
        for (Client migrant : migrants) {
            activeClientsId.add(migrant.getIdOfClient());
        }
        persistenceTransaction.commit();
        persistenceTransaction = null;
    } finally {
        HibernateUtils.rollback(persistenceTransaction,
logger);
        HibernateUtils.close(persistenceSession, logger);
    }
    try {
        persistenceSession =
persistenceSessionFactory.openSession();
        persistenceTransaction =
persistenceSession.beginTransaction();
        if (clientRegistryRequest.getCurrentCount() != null
&& activeClientsId.size() != clientRegistryRequest
.getCurrentCount()) {
            for (Long id : activeClientsId) {
                clientRegistry.addActiveClientId(id);
            }
        }
        if (!errorClientIds.isEmpty()) {
            List errorClients =
fetchErrorClientsWithoutFriendlyOrg(persistenceSession,
organization.getFriendlyOrg(), errorClientIds);
            ClientGroup clientGroup =
findClientGroupByGroupNameAndorgId(persistenceSession,

```

```

        organization.getOrgId(),
ClientGroup.Predefined.CLIENT_LEAVING.getNameOfGroup());
        if (clientGroup == null) {
            clientGroup =
createClientGroup(persistenceSession, organization.getOrgId(),
        ClientGroup.Predefined.CLIENT_LEAVING);
        }
        for (Object object : errorClients) {
            Client client = (Client) object;
            client.setClientGroup(clientGroup);
            if
(client.getOrg().getOrgId().equals(orgId)) {
                clientRegistry.addItem(new
SyncResponse.ClientRegistry.Item(client, 0));
            } else {
                clientRegistry.addItem(new
SyncResponse.ClientRegistry.Item(client, 1));
            }
        }
        persistenceTransaction.commit();
        persistenceTransaction = null;
    } finally {
        HibernateUtils.rollback(persistenceTransaction,
logger);
        HibernateUtils.close(persistenceSession, logger);
    }
    return clientRegistry;
}

```

```

private SyncResponse.ClientRegistry
processSyncClientRegistryForMigrants(Long orgId) throws
Exception {
    SyncResponse.ClientRegistry clientRegistry = new
SyncResponse.ClientRegistry();
    Session persistenceSession = null;
    Transaction persistenceTransaction = null;
    try {
        persistenceSession =
RuntimeContext.getInstance().createReportPersistenceSession();
        persistenceTransaction =
persistenceSession.beginTransaction();

```

```

        List<Client> clients =
MigrantsUtils.getActiveMigrantsForOrg(persistenceSession,
orgId);
        for (Client client : clients) {
            clientRegistry.addItem(new
SyncResponse.ClientRegistry.Item(client, 0));
        }
        persistenceTransaction.commit();
        persistenceTransaction = null;
    } finally {
        HibernateUtils.rollback(persistenceTransaction,
logger);
        HibernateUtils.close(persistenceSession, logger);
    }
    return clientRegistry;
}

private OrganizationStructure
getOrganizationStructureData(Org org, long version, boolean
isAllOrgs) throws Exception {
    OrganizationStructure organizationStructure = new
OrganizationStructure();
    Session persistenceSession = null;
    Transaction persistenceTransaction = null;
    try {
        persistenceSession =
persistenceSessionFactory.openSession();
        persistenceTransaction =
persistenceSession.beginTransaction();
        List<Org> list;
        if (isAllOrgs) {
            list =
DAOUtils.getOrgsSinceVersion(persistenceSession, version);
        } else {
            list =
DAOUtils.findAllFriendlyOrgs(persistenceSession,
org.getOrgId());
        }
        organizationStructure.addOrganizationStructureInfo(persistenceSe
ssion, org, list, isAllOrgs);
        persistenceTransaction.commit();
        persistenceTransaction = null;
    } finally {

```

```

        HibernateUtils.rollback(persistenceTransaction,
logger);
        HibernateUtils.close(persistenceSession, logger);
    }
    return organizationStructure;
}

private ProcessGroupsOrganizationData
processGroupsOrganizationData (GroupsOrganizationRequest
sectionRequest)
    throws Exception {
    Session persistenceSession = null;
    Transaction persistenceTransaction = null;
    ProcessGroupsOrganizationData result = null;
    try {
        persistenceSession =
persistenceSessionFactory.openSession();
        persistenceTransaction =
persistenceSession.beginTransaction();
        AbstractProcessor processor = new
GroupsOrganizationDataProcessor(persistenceSession,
sectionRequest);
        result = (ProcessGroupsOrganizationData)
processor.process();
        persistenceTransaction.commit();
        persistenceTransaction = null;
    } finally {
        HibernateUtils.rollback(persistenceTransaction,
logger);
        HibernateUtils.close(persistenceSession, logger);
    }
    return result;
}

private ResProcessGroupsOrganization
processResGroupsOrganization (GroupsOrganizationRequest
sectionRequest)
    throws Exception {
    Session persistenceSession = null;
    Transaction persistenceTransaction = null;
    ResProcessGroupsOrganization result = null;
    try {
        persistenceSession =
persistenceSessionFactory.openSession();

```

```

        persistenceTransaction =
persistenceSession.beginTransaction();
        AbstractProcessor processor = new
ResGroupsOrganizationProcessor(persistenceSession,
sectionRequest);
        result = (ResProcessGroupsOrganization)
processor.process();
        persistenceTransaction.commit();
        persistenceTransaction = null;
    } finally {
        HibernateUtils.rollback(persistenceTransaction,
logger);
        HibernateUtils.close(persistenceSession, logger);
    }
    return result;
}

private SyncResponse.ResPassageEvents
processSyncPassageEvents(PassageEvents PassageEvents, Org org) {
    Session persistenceSession = null;
    Transaction persistenceTransaction = null;
    SyncResponse.ResPassageEvents resPassageEvents = new
SyncResponse.ResPassageEvents();
    Long orgId;
    Map<String, Long> accessories = new HashMap<String,
Long>();
    for (PassageEventItem e : PassageEvents.getEvents()) {
        try {
            persistenceSession =
persistenceSessionFactory.openSession();
            persistenceTransaction =
persistenceSession.beginTransaction();
            final Long idOfClient = e.getIdOfClient();
            orgId = accessories.get(e.getTurnstileAddr());
            if (orgId == null) {
                orgId = org.getOrgId();
                orgId = DAOService.getInstance()
.receiveorgIdByAccessory(orgId,
Accessory.GATE_ACCESSORY_TYPE, e.getTurnstileAddr());
                accessories.put(e.getTurnstileAddr(), orgId);
            }
            if (existPassageEvent(persistenceSession, orgId,
e.getpassageEventId())) {

```

```

        PassageEvent ee =
findPassageEvent(persistenceSession,
                new
CompositePassageEventId(e.getpassageEventId(), orgId));
        final boolean checkClient =
                (ee.getClient() == null &&
idOfClient == null) || (ee.getClient() != null && ee.getClient()
                .getIdOfClient().equals(idOfClient));
        if (checkClient &&
ee.getEvtDateTime().equals(e.getEvtDateTime())) {
                SyncResponse.ResPassageEvents.Item item
= new SyncResponse.ResPassageEvents.Item(
                e.getpassageEventId(),
SyncResponse.ResPassageEvents.Item.RC_OK,
                "Enter event already
registered");
                resPassageEvents.addItem(item);
        } else {
                SyncResponse.ResPassageEvents.Item item
= new SyncResponse.ResPassageEvents.Item(
                e.getpassageEventId(),
SyncResponse.ResPassageEvents.Item.RC_OK, String.format(
                "Enter event already registered
but attributes differ, orgId == %d, passageEventId == %d",
                orgId, e.getpassageEventId()));
                resPassageEvents.addItem(item);
        }
    } else {
        Client clientFromPassageEvent = null;
        if (idOfClient != null) {
            clientFromPassageEvent = (Client)
persistenceSession.get(Client.class, idOfClient);
            if (clientFromPassageEvent == null) {
                SyncResponse.ResPassageEvents.Item
item = new SyncResponse.ResPassageEvents.Item(
                e.getpassageEventId(),
SyncResponse.ResPassageEvents.Item.RC_CLIENT_NOT_FOUND,
                String.format("Client not
found: %d", idOfClient));
                resPassageEvents.addItem(item);
                continue;
            }
        }
    }
}

```

```

        if (e.getPassDirection() ==
        PassageEvent.CHECKED_BY_TEACHER_EXT) {
            String qstr = "delete from
        PassageEventManual where idOfClient = :idOfClient and
        evtDateTime <= :evtDateTime";
            Query query =
        persistenceSession.createQuery(qstr);
            query.setParameter("idOfClient",
        e.getIdOfClient());
            query.setParameter("evtDateTime",
        e.getEvtDateTime());
            query.executeUpdate();
        }
        PassageEvent passageEvent = new
        PassageEvent();
        passageEvent.setCompositepassageEventId(new
        CompositePassageEventId(e.getpassageEventId(), orgId));
        passageEvent.setEnterName(e.getEnterName());
        passageEvent.setTurnstileAddr(e.getTurnstileAddr());
        passageEvent.setPassDirection(e.getPassDirection());
        passageEvent.setEventCode(e.getEventCode());
        passageEvent.setCardId(e.getIdOfCard());
        passageEvent.setClient(clientFromPassageEvent);
        passageEvent.setEvtDateTime(e.getEvtDateTime());
        passageEvent.setIssueDocDate(e.getIssueDocDate());
        SyncResponse.ResPassageEvents.Item item =
        new SyncResponse.ResPassageEvents.Item(e.getpassageEventId(),
        0, null);
        resPassageEvents.addItem(item);
        if
        (CalendarUtils.isDateToday(e.getEvtDateTime()) && idOfClient !=
        null && (
            e.getPassDirection() ==
        PassageEvent.ENTRY || e.getPassDirection() == PassageEvent.EXIT
            || e.getPassDirection() == PassageEvent.RE_ENTRY
            || e.getPassDirection() == PassageEvent.RE_EXIT)) {
            final EventNotificationService
        notificationService = RuntimeContext.getAppContext()
            .getBean(EventNotificationService.class);
            String[] values =
        generateNotificationParams(persistenceSession,
        clientFromPassageEvent, e);

```

```

        values =
EventNotificationService.attachTargetIdToValues(e.getpassageEven
tId(), values);
        values = EventNotificationService
.attachSourceOrgIdToValues(orgId, values);
        values =
EventNotificationService.attachOrgAddressToValues(org.getAddress
()), values);
        values =
EventNotificationService.attachOrgShortNameToValues(org.getShort
NameInfoService(), values);
        values = EventNotificationService
.attachGenderToValues(clientFromPassageEvent.getGender(),
values);

        switch (org.getType()) {
            case PROFESSIONAL:
                break;
        }
    }
    if
(RuntimeContext.getInstance().getOptionValueBool(Option.OPTION_J
OURNAL_TRANSACTIONS) && (e.getPassDirection() ==
PassageEvent.ENTRY || e.getPassDirection() == PassageEvent.EXIT
|| e.getPassDirection() == PassageEvent.RE_ENTRY ||
e.getPassDirection() == PassageEvent.RE_EXIT) && e.getIdOfCard()
!= null) {
        final CompositePassageEventId compositepassageEventId =
passageEvent.getCompositepassageEventId();
        if (card == null) {
            final String message = "Не найдена
карта по событию прохода: orgId=%d, passageEventId=%d,
idOfCard=%d";

            logger.error(String.format(message,
compositepassageEventId.getorgId(),
compositepassageEventId.getpassageEventId(), e.getIdOfCard()));
        }
        if (card != null && card.getCardType()
== Card.TYPE_UEC) {
            String OGRN =
extraxtORGNFromOrgByorgId(persistenceSession, orgId);
            String transCode;
            switch (e.getPassDirection()) {
                case PassageEvent.ENTRY:
                case PassageEvent.RE_ENTRY:

```

```

        transCode = "IN";
        break;
    case PassageEvent.EXIT:
    case PassageEvent.RE_EXIT:
        transCode = "OUT";
        break;
    default:
        transCode = null;
    }
    if (transCode != null) {
        TransactionJournal
transactionJournal = new TransactionJournal(
        compositepassageEventId.getorgId(),
        compositepassageEventId.getpassageEventId(), new
Date(), OGRN, TransactionJournal.SERVICE_CODE_SCHL_ACC,
transCode, TransactionJournal.CARD_TYPE_CODE_UEC,
        TransactionJournal.CARD_TYPE_ID_CODE_MUID,
Card.TYPE_NAMES[card.getCardType()],
        Long.toHexString(card.getCardNo()),
clientFromPassageEvent.getSan(), clientFromPassageEvent.getContra
ctId(),
clientFromPassageEvent.getClientGroupTypeAsString(),
e.getEnterName());
        persistenceSession.save(transactionJournal);
    }
}

}

}
persistenceTransaction.commit();
persistenceTransaction = null;
} catch (Exception ex) {
    logger.error("Save enter event to database
error: ", ex);
    resPassageEvents = new
SyncResponse.ResPassageEvents();
    for (PassageEventItem ee :
PassageEvents.getEvents()) {
        SyncResponse.ResPassageEvents.Item item;
        item = new
SyncResponse.ResPassageEvents.Item(ee.getpassageEventId(), 1,
"Save to data base error");
        resPassageEvents.addItem(item);
    }
} finally {

```

```

        HibernateUtils.rollback(persistenceTransaction,
logger);
        HibernateUtils.close(persistenceSession,
logger);
    }
}
return resPassageEvents;
}

private boolean PassageEventOwnerHaveSmartWatch(Session
session, PassageEvent PassageEvent) throws Exception {
    if (PassageEvent.getClient() != null) {
        return
PassageEvent.getClient().clientHasActiveSmartWatch();
    } else if (PassageEvent.getCardId() != null) {
        return
DAOUtils.findClientByCardNoAndHeHaveActiveSW(session,
PassageEvent.getCardId(),
PassageEvent.getCompositepassageEventId().getorgId());
    }
    return false;
}

private Long generatePacketId(Long orgId) throws Exception {
    Session persistenceSession = null;
    Transaction persistenceTransaction = null;
    try {
        persistenceSession =
persistenceSessionFactory.openSession();
        persistenceTransaction =
persistenceSession.beginTransaction();
        Org organization =
getOrgReference(persistenceSession, orgId);
        OrgSync orgSync = (OrgSync)
persistenceSession.load(OrgSync.class, orgId);
        Long result = orgSync.getPacketId();
        orgSync.setPacketId(++result);
        organization.setUpdateTime(new
java.util.Date(java.lang.System.currentTimeMillis()));
        persistenceSession.update(organization);
        persistenceSession.update(orgSync);
        persistenceSession.flush();
        persistenceTransaction.commit();
        persistenceTransaction = null;
    }
}

```

```

        return result;
    } finally {
        HibernateUtils.rollback(persistenceTransaction,
logger);
        HibernateUtils.close(persistenceSession, logger);
    }
}

private String[] generateNotificationParams(Session session,
Client client, PassageEventItem event) throws Exception {
    final int passDirection = event.getPassDirection();
    final Date eventDate = event.getEvtDateTime();
    final String passageEvent = "Вход";
    final String exitEvent = "Выход";
    String eventName = "";
    if (passDirection == PassageEvent.ENTRY) {
        eventName = passageEvent;
    } else if (passDirection == PassageEvent.EXIT) {
        eventName = exitEvent;
    } else if (passDirection == PassageEvent.RE_ENTRY) {
        eventName = passageEvent;
    } else if (passDirection == PassageEvent.RE_EXIT) {
        eventName = exitEvent;
    }
    Calendar calendar = Calendar.getInstance();
calendar.setTimeZone(RuntimeContext.getInstance().getDefaultLocalTimezone(null));
    calendar.setTime(eventDate);
    int hour = calendar.get(Calendar.HOUR_OF_DAY);
    int minute = calendar.get(Calendar.MINUTE);
    String time = (hour < 10 ? "0" + hour : hour) + ":" +
(minute < 10 ? "0" + minute : minute);
    DateFormat df =
DateFormat.getDateTimeInstance(DateFormat.FULL,
DateFormat.FULL);
    String empTime = df.format(eventDate);
    String enterWithChecker = "0";
    if (event.getIdOfClient() != null &&
event.getEventCode() == 112) {
        enterWithChecker = "1";
    }
    return new String[]{
}

```

```

        private String[] generatePaymentNotificationParams(Session
session, Client client, Payment payment) {
            long complexes = 0L;
            long others = 0L;
            String date = new SimpleDateFormat("dd.MM.yy
HH:mm").format(payment.getTime());
            String contractId =
String.valueOf(client.getContractId());
            DateFormat df =
DateFormat.getDateTimeInstance(DateFormat.FULL,
DateFormat.FULL);
            String empTime = df.format(payment.getTime());
            return new String[]{
                "date", date, "contractId", contractId,
"others", CurrencyStringUtils.copecksToRubles(others),
                "complexes",
CurrencyStringUtils.copecksToRubles(complexes), "empTime",
empTime};
        }

        public void runRegularPaymentsIfEnabled(SyncRequest request)
{
            if (RuntimeContext.getInstance().isMainNode() &&
RuntimeContext.getInstance().getSettingsConfig()
                .isAutopaymentBkEnabled()) {
                processorUtils.runRegularPayments(request);
            }
        }

        private void fullProcessingOrgSettings(SyncRequest request,
SyncHistory syncHistory,
            List<AbstractToElement> responseSections) {
            try {
                OrgSettingsRequest orgSettingsRequest =
request.getOrgSettingsRequest();
                if (orgSettingsRequest != null) {
                    orgSettingsRequest.setorgIdSource(request.getOrgId());
                    OrgSettingSection orgSettingSection =
processOrgSettings(orgSettingsRequest);
                    addToResponseSections(orgSettingSection,
responseSections);
                    discardOrgSettingsSyncParam(request.getOrgId());
                }
            } catch (Exception e) {

```

```

        String message = String.format("Error when process
OrgSettingSetting: %s", e.getMessage());
processorUtils.createSyncHistoryException(persistenceSessionFact
ory, request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    }
}

private SyncResponse
buildOrgSettingsSectionsResponse(SyncRequest request, Date
syncStartTime, int syncResult) {
    SyncHistory syncHistory = null;
    Long packetId = null, idOfSync = null;
    ResAccountOperationsRegistry
resAccountOperationsRegistry = null;
    ResPaymentRegistry resPaymentRegistry = null;
    SyncResponse.AccRegistry accRegistry = null;
    SyncResponse.AccIncRegistry accIncRegistry = null;
    SyncResponse.ClientRegistry clientRegistry = null;
    SyncResponse.ResOrgStructure resOrgStructure = null;
    SyncResponse.ResPassageEvents resPassageEvents = null;
    Manager manager = null;
    OrgOwnerData orgOwnerData = null;
    AccRegistryUpdate accRegistryUpdate = null;
    OrganizationStructure organizationStructure = null;
    AccountsRegistry accountsRegistry = null;
    OrgSettingSection orgSetting = null;
    List<AbstractToElement> responseSections = new
ArrayList<AbstractToElement>();
    try {
        OrgSettingsRequest orgSettingsRequest =
request.getOrgSettingsRequest();
        if (orgSettingsRequest != null) {
            orgSettingsRequest.setOrgIdSource(request.getOrgId());
            orgSetting =
processOrgSettings(orgSettingsRequest);
        }
    } catch (Exception e) {
        String message = String.format("Error when process
OrgSettingSetting: %s", e.getMessage());
        processorUtils.createSyncHistoryException(persistenceSessionFact
ory, request.getOrgId(), syncHistory, message);
        logger.error(message, e);
    }
}

```

```

        discardOrgSettingsSyncParam(request.getOrgId());
        Date syncEndTime = new Date();
        return new SyncResponse(request.getSyncType(),
request.getOrgId(), request.getOrg().getShortName(),
            request.getOrg().getType(), "", packetId, null,
syncEndTime, "", accRegistry, resPaymentRegistry,
resAccountOperationsRegistry, accIncRegistry, clientRegistry,
resOrgStructure, "", resPassageEvents, manager, orgOwnerData,
accRegistryUpdate, accountsRegistry, organizationStructure,
            responseSections, orgSetting);
    }

    private OrgSettingSection
processOrgSettings(OrgSettingsRequest orgSettingsRequest) throws
Exception {
        Session persistenceSession = null;
        Transaction persistenceTransaction = null;
        OrgSettingSection orgSetting = null;
        try {
            persistenceSession =
persistenceSessionFactory.openSession();
            persistenceTransaction =
persistenceSession.beginTransaction();
            AbstractProcessor processor = new
OrgSettingsProcessor(persistenceSession, orgSettingsRequest);
            orgSetting = (OrgSettingSection)
processor.process();
            persistenceTransaction.commit();
            persistenceTransaction = null;
        } finally {
            HibernateUtils.rollback(persistenceTransaction,
logger);
            HibernateUtils.close(persistenceSession, logger);
        }
        return orgSetting;
    }
}

```

Приложение В. Листинг XML-файла

```
<?xml version="1.0" encoding="utf-8"?>
<Data>
  <Body>
    <DataSync OrgId="503" PacketId="112" Version="1"
Date="02.06.2020 12:23:43" Type="1" ClientVersion="2.7.93.1"
DirectorName="Никишин Владимир Петрович" Org="Some
Organization">
      <PaymentRegistry>
        <RP OrderID="28489" CashierID="0025000380"
OrderType="1" CardNum="1560" ClientID="1269473" Date="02.06.2020
11:04:54" State="0" Sum="11500">
          <PL OrderDetailsID="0600212" Qty="1"
Price="2500" Name="Кофе Капучино" Code="8708B30"
ProductUUID="cff7fb0f-ef5a-459a-9b61-31531a40f7d6"/>
          <PL OrderDetailsID="0600213" Qty="2"
Price="4500" Name="Печенье Орио" Code="1902A69"
ProductUUID="e8beeeef-ad9e-44e7-b363-elc46f6alafe"/>
        </RP>
        <RP OrderID="28493" CashierID="0025000378"
OrderType="2" CardNum="73089954" ClientID="1269473"
Date="02.06.2020 11:18:32" State="0" Sum="5900">
          <PL OrderDetailsID="0600214" Qty="1" Price="5900"
Name="Сок Яблочный" Code="3122L99" ProductUUID="da44ab2b-5ab6-
48a4-99e7-8547b7999bba"/>
        </RP>
        <RP OrderID="28494" CashierID="0025000378"
OrderType="1" CardNum="73089954" ClientID="1269473"
Date="02.06.2020 11:19:32" State="0" Sum="90000">
          <PL OrderDetailsID="0600216" Qty="20"
Price="4500" Name="Печенье Орио" Code="1902A69"
ProductUUID="e8beeeef-ad9e-44e7-b363-elc46f6alafe"/>
        </RP>
        <RP OrderID="28494" CashierID="0025000378"
OrderType="1" CardNum="73089954" ClientID="1269473"
Date="02.06.2020 11:21:26" State="1" Sum="90000">
          <PL OrderDetailsID="0600217" Qty="20" Price="4500"
Name="Печенье Орио" Code="1902A69" ProductUUID="e8beeeef-ad9e-
44e7-b363-elc46f6alafe"/>
        </RP>
      </PaymentRegistry>
      <OrganizationStructure MaxVersion="23">
        <OS OrgId="503" OrgType="1" Name="Main Building"
Address="Some Address" Version="23" State="0"/>
        <OS OrgId="504" OrgType="0" Name="Manufacture Building"
Address="Some Address" Version="23" State="0"/>
        <OS OrgId="505" OrgType="0" Name="Storehouse"
Address="Some Address" Version="23" State="0"/>
      </OrganizationStructure>
    </DataSync>
  </Body>
</Data>
```

```

</OrganizationStructure>
<AccountsRegistry MaxVersion="43">
  <AR ClientID="1269500" Version="43"
  FirstName="Александр" Patronymic="Петрович" Surname="Миронин"
  GroupName="Руководители производства" AccountType="2"
  DocumentID="600343501" Phone="Some PhoneNum"
  Email="SomeEmail@coopmail.com" ContractId="600343501" Gender="0"
  ClientUUID="15cd65dd-c847-4b72-bale-450bc0ecb967"/>
  <AR ClientID="1269501" Version="43" FirstName="Ирина"
  Patronymic="Дмитриевна" Surname="Юртаева" GroupName="Юристы"
  AccountType="2" DocumentID="600343502" Phone="Some PhoneNum"
  Email="SomeEmail@coopmail.com" ContractId="600343502" Gender="1"
  ClientUUID="e94edef3-1494-463c-97cb-7033457e4458"/>
</AccountsRegistry>
  <PassEvents>
    <PE PassageEventsID="489" EnterName="Терминал Охраны"
    TurnstileUUID="11617G7267B0" PassCode="301" ClientID="1269473"
    EvtDate="02.06.2020 10:58:54" />
    <PE PassageEventsID="493" EnterName="Вход №1"
    TurnstileUUID="00250B0037A8" PassCode="102" EventCode="17"
    CardID="43317908" ClientID="1269473" EvtDate="02.06.2020
    11:01:57" />
    <PE PassageEventsID="494" EnterName="Вход №1"
    TurnstileUUID="00250B0037A8" PassCode="201" EventCode="17"
    CardID="43317908" ClientID="1269473" EvtDate="02.06.2020
    11:02:01" />
    <PE PassageEventsID="495" EnterName="Вход №1"
    TurnstileUUID="00250B0037A8" PassCode="202" EventCode="17"
    CardID="43317908" ClientID="1269473" EvtDate="02.06.2020
    11:05:28" />
    <PE PassageEventsID="496" EnterName="Терминал Охраны"
    TurnstileUUID="11617G7267B0" PassCode="301" ClientID="1310510"
    EvtDate="02.06.2020 11:07:45" />
    <PE PassageEventsID="497" EnterName="Вход №1"
    TurnstileUUID="00250B0037A8" PassCode="500" EventCode="17"
    CardID="93509545" ClientID="1310510" EvtDate="02.06.2020
    11:11:05" />
    <PE PassageEventsID="498" EnterName="Вход №1"
    TurnstileUUID="00250B0037A8" PassCode="101" EventCode="17"
    CardID="93509545" ClientID="1310510" EvtDate="02.06.2020
    11:13:35" />
  </PassEvents>
  <OrgSettings MaxVersion="1125">
    <OS OrgID="503" SettingGroup="9" Version="1125"
    SettingCode="1300" Value="13:00;17:00;19:00"
    GlobalUUID="5bb802d3-94ef-4360-bad9-b60dc76076dd"/>
    <OS OrgID="503" SettingGroup="9" Version="1125"
    SettingCode="1301" Value="01:00;12:00" GlobalUUID="781fef0b-
    efc3-440c-b96b-5a86402194bd"/>
  </OrgSettings>

```

```

        <Log>
            <L EvtTime="02.06.2020 11:11:05" Severity="2"
Code="2001" Text="Проход запрещен из-за ошибки считывателя:
RES_QUIT (Предыдущая команда прерван). CardID=93509545
ClientID=1310510"/>
            <L EvtTime="02.06.2020 11:21:26" Severity="1"
Code="1005" Text="Стонирование покупки OrderID=28495, Клиент
ID=1269473"/>
        </Log>
    </DataSync>
</Body>
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
            <CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
            <SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha256" />
            <Reference URI="">
                <Transforms>
                    <Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature" />
                </Transforms>
                <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
                <DigestValue>nk05Y0Pk0mQeNICtrs9g0y81Y4I=</DigestValue>
            </Reference>
        </SignedInfo>

        <SignatureValue>j05aDZv+5vEzx35pp2slo2vxerumoHD1qWhRy++UYAVr
YtfQlov2h9x0mN6RLFiCZZmfQVIJSeX8LJEQS0uJikkeBIyXawJ9SL8OWjlAqCWo
0dcGkQinDVIrwx7P8HRXnrkPl9jFCbaOnpwq8bgGgSY9H+pDJv7je/664IoBBF0=
</SignatureValue>
    </Signature>
</Data>

```

Приложение С. Листинг SQL-файла

Создание пользователя.

```
CREATE TABLE Organizations
(
    OrgID          BIGSERIAL PRIMARY KEY,
    DirectorID    BIGINT,
    OrgType        INTEGER          NOT NULL,
    Name           VarChar(256) NOT NULL,
    Address        VarChar(256) NOT NULL,
    Version        BIGINT          NOT NULL DEFAULT 0,
    State          Integer          NOT NULL,
    ContractID     VarChar(40)     NOT NULL UNIQUE,
    Phone          VarChar(11)     NOT NULL,
    City           VarChar(256) NOT NULL
);

CREATE TABLE AccountsRegistry
(
    ClientID       BIGSERIAL PRIMARY KEY,
    OrgID          BIGINT          NOT NULL REFERENCES Organizations
(OrgID),
    Version        BIGINT          NOT NULL DEFAULT 0,
    FirstName      VarChar(256) NOT NULL,
    Patronymic     VarChar(256),
    Surname        VarChar(256) NOT NULL,
    GroupName      VarChar(256) NOT NULL,
    AccountType    Integer          NOT NULL,
    DocumentID     BIGINT          NOT NULL,
    Phone          VarChar(11),
    Email          VarChar(256),
    ContractId     BIGINT          NOT NULL UNIQUE,
    ClientUUID     VarChar(36)     NOT NULL UNIQUE,
    Gender         INTEGER          NOT NULL DEFAULT 0
);

ALTER TABLE Organizations
    ADD FOREIGN KEY (DirectorID) REFERENCES AccountsRegistry
(ClientID);

CREATE TABLE Payments
(
    PaymentID     BIGSERIAL PRIMARY KEY,
    OrgID         BIGINT          NOT NULL REFERENCES Organizations
(OrgID),
    ClientID      BIGINT          NOT NULL REFERENCES AccountsRegistry
(ClientID),
    CardID        BIGINT,
```

```

        Date          Timestamp NOT NULL,
        OrderType     Integer    NOT NULL,
        Sum            Integer    NOT NULL,
        Discount       Integer,
        CashierID     BIGINT      NOT NULL,
        State          Integer    NOT NULL
    );

CREATE TABLE PaymentItems
(
    ItemID            BIGSERIAL PRIMARY KEY,
    PaymentID         BIGINT REFERENCES Payments (PaymentID),
    Qty               Integer,
    Price              Integer      NOT NULL,
    Discount           Integer,
    Name               VarChar(256) NOT NULL,
    Code               VarChar(32)  NOT NULL,
    ProductUUID        VarChar(36)  NOT NULL UNIQUE
);

CREATE TABLE PassageEvents
(
    PassageEventsID   BIGINT,
    OrgID              BIGINT REFERENCES Organizations (OrgID),
    EnterName          VarChar(256) NOT NULL,
    TurnstileUUID      VarChar(36)  NOT NULL UNIQUE,
    PassCode           Integer      NOT NULL,
    CardID             BIGINT,
    ClientID           BIGINT REFERENCES AccountsRegistry
(ClientID),
    EvtDate             Timestamp    NOT NULL,
    PRIMARY KEY (PassageEventsID, OrgID)
);

CREATE TABLE OrgSettings
(
    OrgSettingsID     BIGSERIAL PRIMARY KEY,
    OrgID              BIGINT NOT NULL REFERENCES Organizations
(OrgID),
    SettingGroup       Integer NOT NULL,
    CONSTRAINT OrgSettings_UK UNIQUE (OrgID, SettingGroup)
);

CREATE TABLE OrgSettingsItems
(
    OrgSettingsItemsID BIGSERIAL PRIMARY KEY,
    OrgSettingsID       BIGINT NOT NULL REFERENCES
OrgSettings (OrgSettingsID),
    GlobalUUID          VarChar(36) NOT NULL UNIQUE,
    SettingCode          INTEGER NOT NULL,

```

```

        Value                VarChar(256) NOT NULL,
        Version               BIGINT      NOT NULL,
        CONSTRAINT OrgSettingsItems_UK UNIQUE (SettingCode,
OrgSettingsID)
);

CREATE TABLE Logs
(
    RecordID    BIGSERIAL PRIMARY KEY,
    EvtTime     Timestamp    NOT NULL,
    OrgID       BIGINT       NOT NULL REFERENCES
Organizations (OrgID),
    CSSVersion  VarChar(32)  NOT NULL,
    Severity    Integer      NOT NULL,
    Code        Integer      NOT NULL,
    Text        VarChar(2048)
);

CREATE TABLE SyncHistory
(
    RecordID        BIGSERIAL PRIMARY KEY,
    SyncStartTime   Timestamp    NOT NULL,
    SyncEndTime     Timestamp    NOT NULL,
    OrgID           BIGINT REFERENCES Organizations (OrgID),
    CSSVersion      VarChar(32)  NOT NULL,
    SyncType        Integer      NOT NULL,
    IPAddress       VarChar(16)  NOT NULL,
    PacketID        BIGINT       NOT NULL,
    Result          Integer      NOT NULL,
    ErrorText       VarChar(2048)
);

```

Приложение Д. Презентация

Микросервис синхронизации данных для компонентов информационной системы

Автор работы: студент 16ВВ1 Калашников Виталий Александрович

Научный руководитель: к.т.н., доцент Синев Михаил Петрович

Пенза, 2020

Цель и объект работы

Цель работы – разработать микросервис для двустороннего обмена информацией между компонентами информационной системы

Объект работы - приложение, обеспечивающее выполнение следующих функций:

- обработка запросов от клиентов;
- обеспечение работоспособности системы под большими нагрузками;
- передача только актуальных данных клиентам с обеспечением защиты от «ложных клиентов».

2

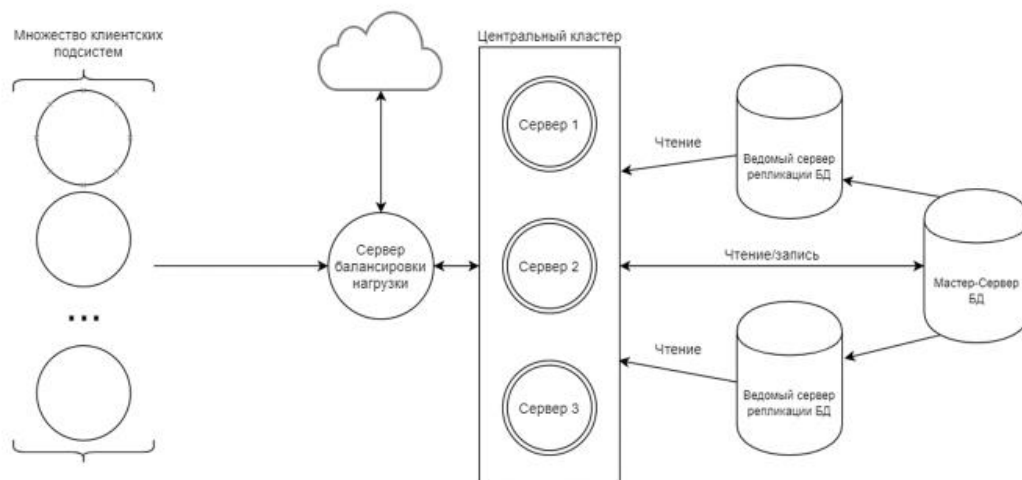
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.18.001 ПЗ

Лист

141

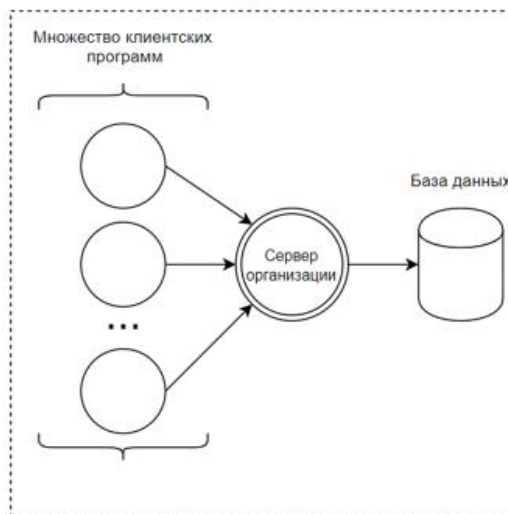
Описание целевой системы



3

Описание целевой системы

Организация



4

Изм.	Лист	№ докум.	Подпись	Дата

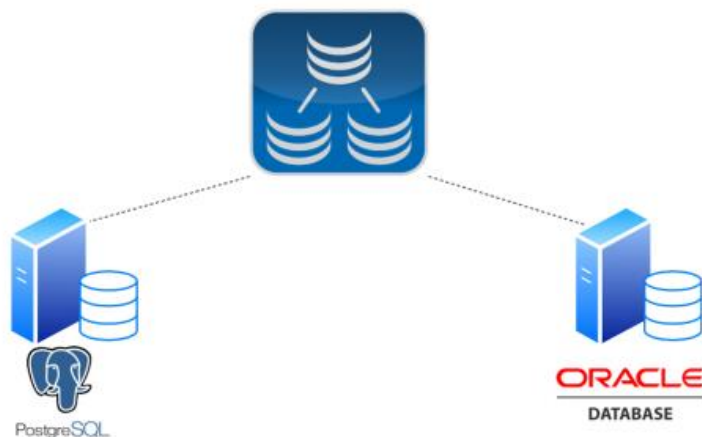
ПГУ 2.090301.18.001 ПЗ

Лист

142

Популярные решения задачи

Репликация таблиц базы данных «чужой» системы в базу данных целевой.



5

Популярные решения задачи

Обмен данными при помощи брокеров сообщений.



6

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.18.001 ПЗ

Лист

143

Стек технологий



javamelody
monitoring of JavaEE applications



HIBERNATE

7

Протокол взаимодействия

Взаимодействие осуществляется по схеме запрос-ответ.

Инициатором запроса является КПС.

Протокол взаимодействия – HTTPS.

Спецификация запроса:

- HTTP method: POST;
- request: транспортный конверт (MIME: text/xml);
- response: транспортный конверт (MIME: text/xml);
- timeout: 3 минуты.

8

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.18.001 ПЗ

Лист

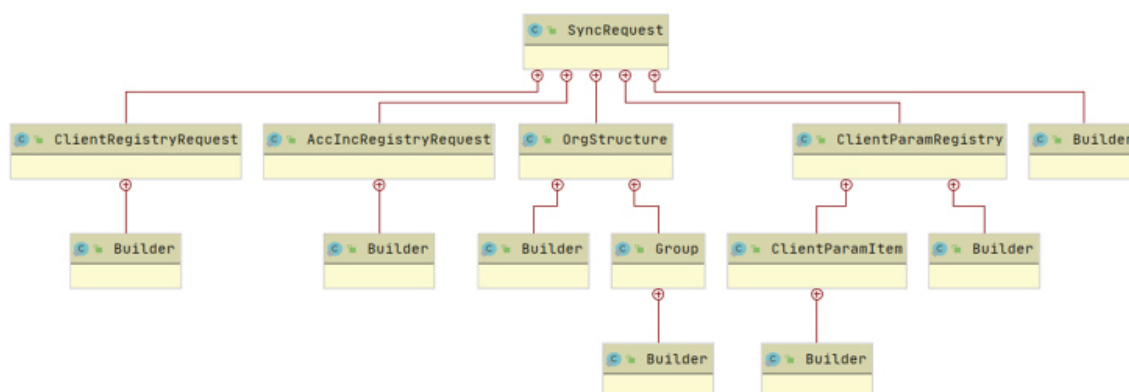
144

Состав пакета

Название поля	Описание	Тип данных
OrgID	Идентификатор ОО (уникален в системе)	Int64
PacketID	Идентификатор пакета (уникален для КПС)	Int64
Version	Версия протокола	Int64
Date	Дата и время формирования пакета	DateTime
Org	Наименование организации	Str(256)
ClientVersion	Номер версии клиентской программы	Str(16)
Type	Тип синхронизации	Int
DirectorName	ФИО руководителя ОО	Str(256)
PaymentRegistry	Реестр платежей и покупок	Element
OrganizationStructure	Структура организации	Element
AccountsRegistry	Реестр учетных записей	Element
PassageEvents	Журнал событий системы доступа на территорию	Element
OrgSettings	Передаваемые параметры клиентской программы	Element
Log	Журнал ошибок и системных оповещений	Element
Signature	Цифровая подпись пакета	Element

9

Диаграмма транспортных классов для упаковки запроса



10

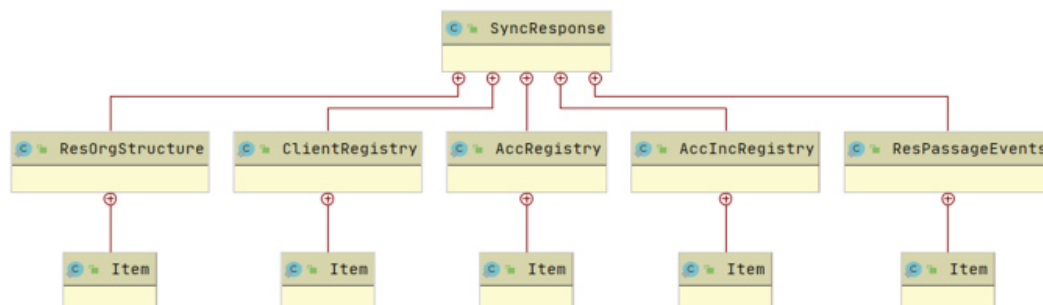
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.18.001 ПЗ

Лист

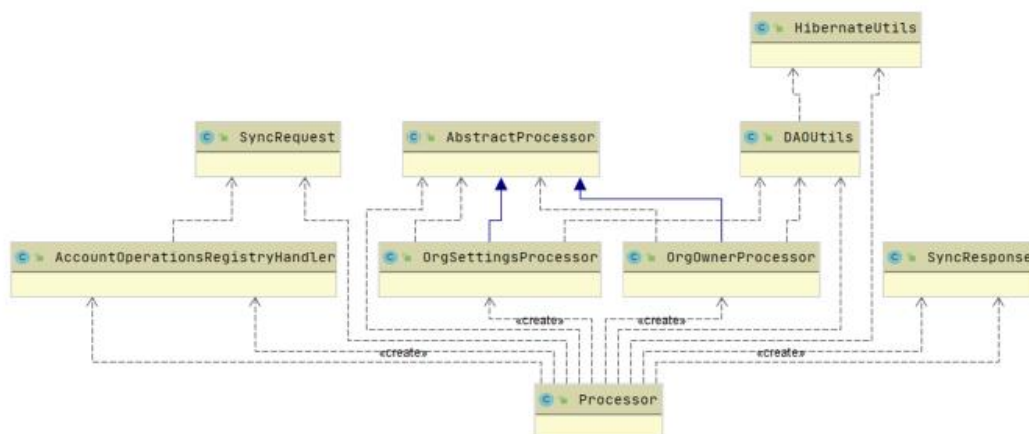
145

Диаграмма транспортных классов для упаковки ответа



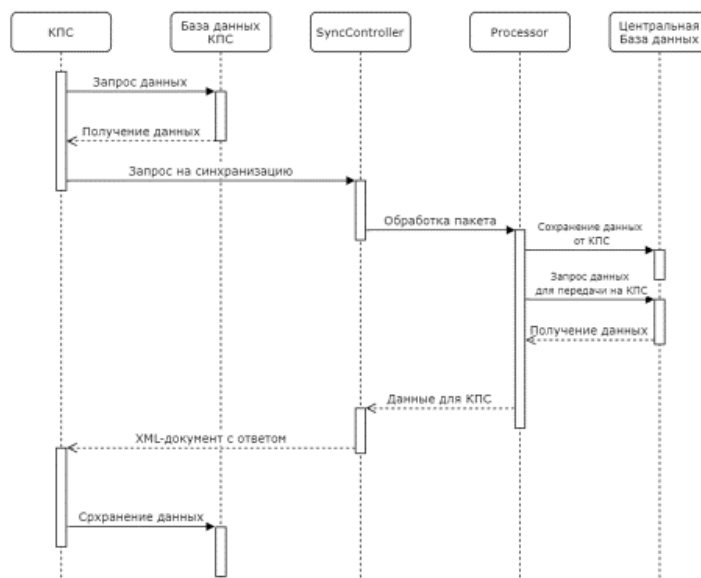
11

Диаграмма классов-обработчиков



12

Диаграмма последовательности



13

Модуль автораспределения обращений

Проблема:

Большое количество клиентских систем отправляют огромный массив данных на сервер в одно и тоже время, создавая нагрузку.

Решение:

На стороне сервера создать модуль для генерации расписания обращений сразу для всех клиентских систем, не нарушая требований сохранять актуальность данных.

14

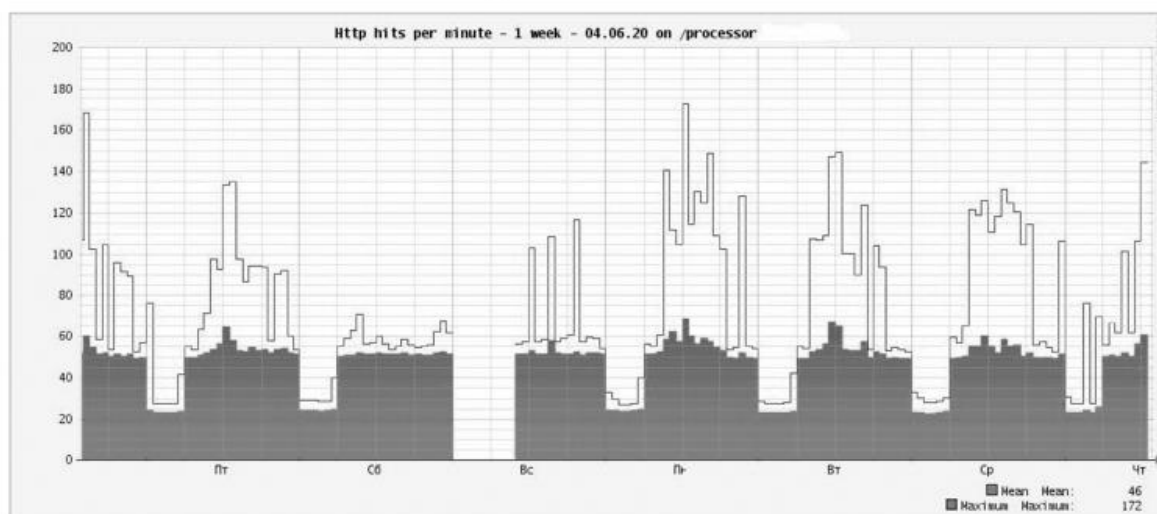
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.18.001 ПЗ

Лист

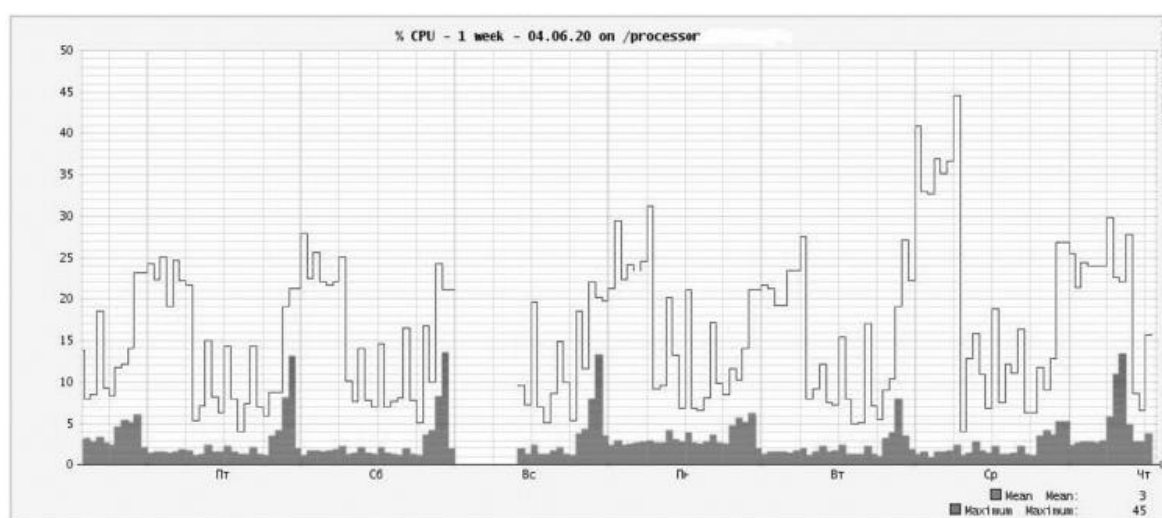
147

Показатели производительности



15

Показатели производительности



16

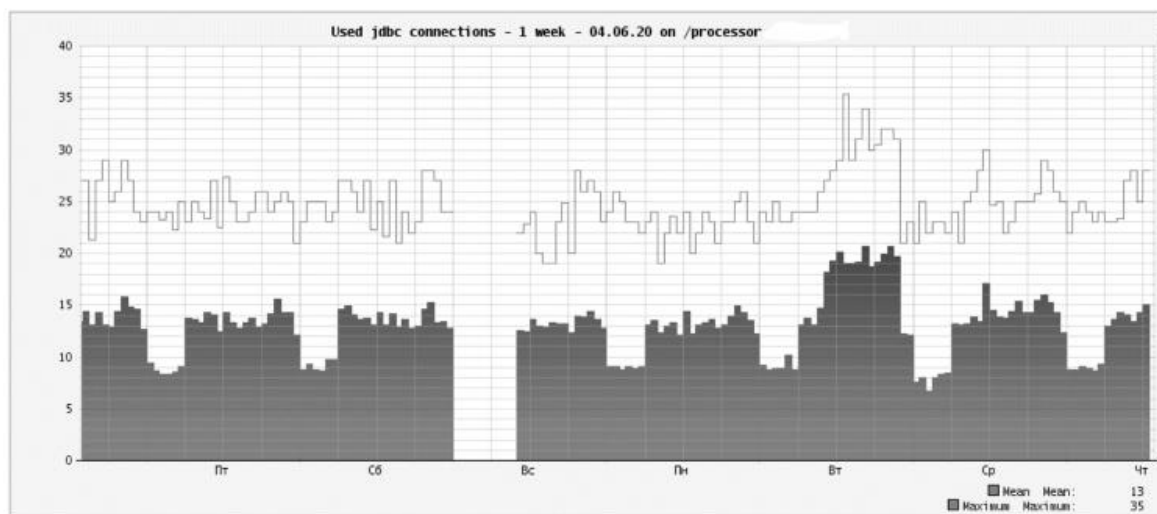
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.18.001 ПЗ

Лист

148

Показатели производительности



17

Результаты

- Результатом разработки стал микросервис двустороннего обмена информации между сервером и множеством клиентских систем, который отвечает поставленным ранее требованиям;
- Разработан протокол взаимодействия между клиентом и сервером;
- Так же для повышения оптимизации был разработан и введен в эксплуатацию модуль автоматического распределения времени обращений.

18

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.18.001 ПЗ

Лист

149

Благодарю за
внимание!

19

					ПГУ 2.090301.18.001 ПЗ	Лист
						150
Изм.	Лист	№ докум.	Подпись	Дата		