РЕФЕРАТ

Отчёт 20 с., 11 рис., 3 источника.

JAVA-ПРИЛОЖЕНИЕ, СЕРВЕР, ANGULAR-ПРИЛОЖЕНИЕ, КЛИЕНТ, SPRING, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, АВТОМАТИЗАЦИЯ СКЛАДСКОГО УЧЁТА, АВТОМАТИЗАЦИЯ БИЗНЕСА, РАЗРАБОТКА, WEB-ПРИЛОЖЕНИЕ

Объектом исследования является клиент-серверное приложение автоматизации складского учёта «СКЛАД 2.0», разрабатываемое в рамках проекта «ПРИЗМА 2.0» на предприятии ФГУП «ВНИИА им. Духова».

Цель работы – разработка конфигуратора для приложения автоматизации складского учёта.

Результатом работы является программный конфигуратор, обладающий графическим интерфейсом, и позволяющий гибко настраивать приложение автоматизации складского учёта на различных предприятиях.

Область применения – автоматизация складского документооборота и учёта материальных ресурсов на множестве предприятий госкорпорации «Росатом».

Разработанный конфигуратор был успешно внедрён в клиент-серверное приложение.

Эффективность конфигуратора определяется гибкостью и последовательностью настройки бизнес-приложения

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 7](#_Toc167829480)

[1 Теоретическая часть 9](#_Toc167829481)

[1.1 Обзор технологий, применяемых при разработке 9](#_Toc167829482)

[1.1.1 Технологии серверного приложения 9](#_Toc167829483)

[1.1.2 Технологии клиентского приложения 11](#_Toc167829484)

[1.2 Структура клиентской части приложения 12](#_Toc167829485)

[1.2.1 Интерфейсный слой 12](#_Toc167829486)

[1.2.2 Слой логики компонентов 13](#_Toc167829487)

[1.2.3 Слой отправки запросов 14](#_Toc167829488)

[1.3 Структура серверной части приложения 15](#_Toc167829489)

[1.3.1 Слой обработки запросов 17](#_Toc167829490)

[1.3.2 Слой бизнес-логики 19](#_Toc167829491)

[1.3.3 Слой базы данных 20](#_Toc167829492)

[2 Подготовка к разработке конфигуратора 21](#_Toc167829493)

[2.1 Перечень требований, настраиваемых конфигуратором 21](#_Toc167829494)

[2.2 Выбор способа реализации конфигурации приложения 22](#_Toc167829495)

[2.2.1 Конфигурационные файлы 22](#_Toc167829496)

[2.2.2 Конфигурация при помощи аннотаций 24](#_Toc167829497)

[2.2.3 Конфигурация при помощи базы данных и пользовательского интерфейса 25](#_Toc167829498)

[2.3 Принятие решения по способу реализации конфигуратора и разбиение его на разделы 26](#_Toc167829499)

[3 Описание конфигуратора 27](#_Toc167829500)

[3.1 Конфигуратор точности материальных ресурсов 28](#_Toc167829501)

[3.2 Конфигуратор типов операций 31](#_Toc167829502)

[3.3 Конфигуратор типов документов 36](#_Toc167829503)

[3.4 Конфигуратор доступных типов документов 42](#_Toc167829504)

[3.5 Конфигуратор форм 44](#_Toc167829505)

[3.6 Конфигуратор отправителей и получателей 48](#_Toc167829506)

[Отладка, код-ревью и тестирование 49](#_Toc167829507)

[ЗАКЛЮЧЕНИЕ 50](#_Toc167829508)

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящем отчете о НИР применяют следующие термины с соответствующими определениями:

Аннотация – специальный механизм языка программирования Java, позволяющий настроить поведение аннотируемого класса, объекта, метода и т.п.

Бизнес-логика – набор правил поведения приложения при работе с данными.

Клиент – это интерфейс с набором функций, с которым взаимодействует пользователь

Компонент Angular – структурная единица клиентского приложения, состоящая из интерфейсного представления и внутренней логики работы компонента

Модуль бизнес-логики (сервисный класс) – Java-класс, используемый для проведения операций по изменению данных и определяющий поведение приложения при работе с запросами и данными

Модуль доступа серверного приложения (контроллер) – Java-класс, используемый для получения, перенаправления и обработки запросов от клиентского приложения

Модуль записи данных (репозиторий) – Java-класс, связанный с определенной сущностью и используемый для создания, изменения и удаления информации в базе данных

Модуль чтения данных – Java-класс, связанный с определенной сущностью и используемый для транзакционного чтения информации из базы данных

Объект-обёртка – Java-объект, используемый для передачи определенного набора данных между частями приложения или между приложениями

Отправитель (в рамках конфигуратора) – тип пользователя, обладающий соответствующей ролью и способный создавать и проводить расходные документы

Получатель (в рамках конфигуратора) – тип пользователя, обладающий соответствующей ролью и способный создавать и проводить приходные документы

Сервер – программно-аппаратная часть сервиса, которая хранится на сервере, обрабатывает полученные данные и отправляет ответ обратно

Стек технологий – набор технологий, используемых для разработки ПО, включающий языки программирования, фреймворки, библиотеки, сторонние программы и другие инструменты

Сущность – Java-объект, связанный с таблицей в базе данных и отображающий одну строку из неё

Транзакция ­– набор операций в базе данных, выполняемый атомарно

Фреймворк – программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта

Q-сущность – Java-объект, созданный фреймворком QueryDSL на основе сущностей и позволяющий составлять SQL-запросы при помощи программного кода на языке Java

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящем отчете о НИР применяют следующие сокращения и обозначения:

БД – база данных

ДМ – драгоценные металлы

МОЛ – материально ответственное лицо

ТМЦ – товарно-материальная ценность

ВВЕДЕНИЕ

Предметом разработки стал конфигуратор складского учёта, реализуемый в рамках проекта «СКЛАД 2.0» отделом «Призма» ФГУП «ВНИИА им. Духова».

Объектом разработки является приложение «СКЛАД 2.0» – Web-приложение для автоматизации­­­ складского учёта. Оно применяется:

* для выполнения операций с различными видами материальных ресурсов;
* для упрощения создания документов и работы с ними;
* для визуального отображения остатков;
* для интеграции с финансовыми сервисами.

Приложение состоит из серверной и клиентской части. Пользователи имеют доступ к клиенту, который отправляет запрос на сервер. Серверная часть выполняет обработку запроса и совершает действия, заложенные бизнес-логикой приложения. В рамках этих действий происходит работа с базой данных (чтение, создание, изменение и удаление записей), а также отправка запросов в различные сервисы и получение ответов от них.

Благодаря такой архитектуре, приложение «СКЛАД 2.0» намного проще расширять, изменять и интегрировать с другими сервисами, чем ранее разработанное приложение «СКЛАД», запускаемое локально у каждого пользователя.

В виду простоты масштабирования было принято решение распространить новое приложение на множество предприятий Госкорпорации «Росатом», в каждом из которых свой перечень документов и требований к ним, то при создании приложения было необходимо решить проблему настройки приложения под каждое отдельное предприятие.

Для этого был разработан конфигуратор складского учёта, позволяющий гибко настроить перечень операций с ТМЦ и материальными ресурсами, перечень документов, используемых на складах предприятия, а также их интерфейсно-программные и печатные формы.

Этот отчёт по выпускной квалификационной работе посвящен особенностям разработки вышеупомянутого конфигуратора и описанию его работы. Также в отчёте приведено теоретическое описание архитектуры приложения.

Основной целью работы является обеспечение возможности быстрой и гибкой настройки клиент-серверного приложения согласно требованиям предприятия без участия квалифицированных программистов.

В рамках выпускной работы были поставлены и решены следующие задачи:

1. Выявление перечня требований, отличающихся на предприятиях-заказчиках;
2. Выбор способа реализации конфигурации приложения;
3. Разбиение конфигуратора на логически завершённые разделы;
4. Программная реализация каждого раздела конфигуратора;
5. Внедрение конфигуратора в приложение;
6. Отладка и исправление ошибок, найденных командой тестирования.

Актуальность работы состоит в возможности развертывания всей системы складского учёта на любом предприятии-заказчике без необходимости изменения исходного кода программ.

1 Теоретическая часть

Приложение «СКЛАД 2.0» основано на клиент-серверной архитектуре, в которой пользователь взаимодействует с клиентской частью через графический интерфейс. При необходимости получить информацию из базы данных или изменить её клиентская часть приложения отправляет запрос серверной части, которая занимается обработкой запросов и проведением соответствующих транзакций в базе данных.

1.1 Обзор технологий, применяемых при разработке

1.1.1 Технологии серверного приложения

В этом пункте будет приведен перечень технологий, используемых при разработке серверной части приложения в том числе и технологий, применяемых при разработке конфигуратора приложения «СКЛАД 2.0»

Для разработки серверной части приложения использовались следующие технологии:

1. Java 17 – язык программирования строгой типизации, позволяющий обеспечить стабильность работы приложений. Зачастую применяется для автоматизации бизнеса;
2. Spring Boot – фреймворк для Java, позволяющий быстро запустить приложение и обеспечить гибкую и простую конфигурацию программной части серверного приложения. Включает в себя множество технологий для работы с базой данных, обработки HTTP-запросов, обеспечения безопасности и ролевого доступа, а также валидации данных;
3. Spring Core – фреймворк для Java, находящийся в составе Spring Boot и позволяющий упростить разработку и масштабирование приложения путём использования внедрения зависимостей, которое обеспечивает слабую связь между уровнями приложения;
4. Spring Security – фреймворк для Java, находящийся в составе Spring Boot, предоставляющий набор функционала для обеспечения безопасности, проведения авторизации и аутентификации пользователей при обработке их запросов и поддерживающий быструю и гибкую конфигурацию;
5. Реляционная база данных – БД, в которой информация представляет собой объект-строку, расположенную в соответствующей таблице. Таблицы связаны между собой при помощи ключей – ссылок на строки в других таблицах;
6. SQL – язык запросов к реляционным базам данных;
7. PostgreSQL – система управления базами данных, позволяющая расширить возможности SQL, а также оптимизировать работу БД;
8. Spring Data – фреймворк для Java, находящийся в составе Spring Boot и упрощающий обмен данными между приложением и базами данных при помощи технологии ORM. Основан на фреймворке Hibernate;
9. ORM Hibernate – фреймворк для Java, позволяющий произвести однозначное отображение объекта-строчки из таблицы базы данных в сущность (Java-объект) и наоборот;
10. QueryDSL – фреймворк для Java, позволяющий упростить обмен данными между приложением и базами данных при помощи создания запросов к БД на языке Java;
11. Flyway – фреймворк для Java, позволяющий управлять структурой баз данных из Java приложения путём создания, проведения миграций, а также наблюдения за их историей и защищающий приложение от внешних изменений структур баз данных;
12. HTTP-запросы с технологией REST – HTTP-запросы, состоящие из JSON-объектов и позволяющие запрашивать и передавать наборы данных между клиентской и серверной частями приложения;
13. Kerberos – протокол аутентификации компьютерной сети, применяемый для безопасного обмена запросами между клиентом и сервером;
14. Apache Kafka – распределенный программный брокер сообщений, используемый для обмена информацией между различными серверными приложениями.

1.1.2 Технологии клиентского приложения

В этом пункте будет приведен перечень технологий, используемых при разработке клиентской части приложения в том числе и технологий, применяемых при разработке конфигуратора приложения «СКЛАД 2.0»

Для разработки клиентской части приложения использовались следующие технологии:

1. TypeScript – язык программирования строгой типизации, основанный на языке JavaScript и позволяющий писать стабильный код для клиентской части приложения;
2. Angular – фреймворк для TypeScript, позволяющий разделить клиентское приложение на компоненты, имеющие собственный интерфейс. Благодаря Angular компоненты могут обмениваться информацией;
3. Prime NG – библиотека Angular-компонентов, включающая в себя множество решений, связанных с интерфейсом клиентского приложения;
4. AG Grid – библиотека Angular-компонентов, включающая в себя средства для работы с табличными данными;
5. Font Awesome – библиотека иконок для различных интерфейсных элементов;
6. HTTP-запросы с технологией REST.

1.2 Структура клиентской части приложения

Клиентское приложение занимается выводом информации на экран пользователя и отправкой запросов на сервер.

Оно состоит из трёх слоёв:

* Интерфейсный слой;
* Слой логики компонентов;
* Слой обработки запросов.

Данные между слоями приложения перемещаются классах-обёртках.

Фреймворк Angular предоставляет одностраничное отображение информации. За то, какая страница будет отображена на экране пользователя, отвечает маршрутизатор. В зависимости от адреса страницы маршрутизатор выбирает компонент, интерфейс которого будет виден пользователю.

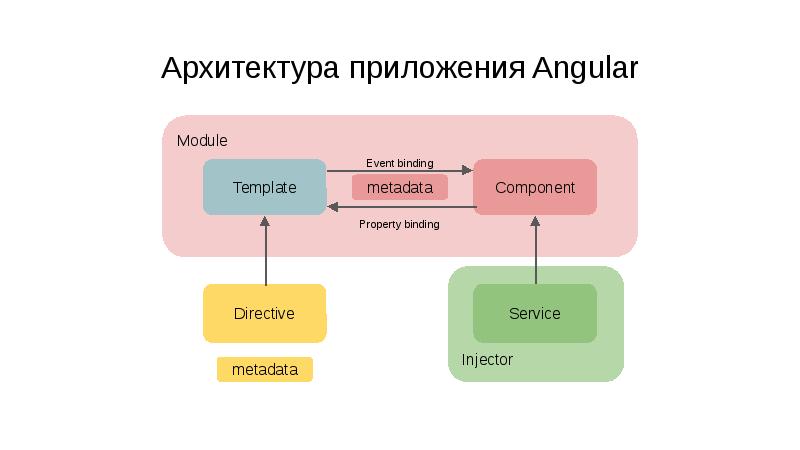


Рисунок 1 – Архитектура Angular приложения

1.2.1 Интерфейсный слой

Интерфейсный слой клиентского приложения состоит из HTML-страниц (шаблонов) и CSS-файлов стилей различных компонентов, которые могут иметь один из следующих типов:

* Табличный. Состоит из таблицы, основанной на AG Grid, и набора кнопок, связанных с ней;
* Регистрационная форма. Состоит из набора полей ввода различных видов (текстового, числового, календарного, выпадающего списка и т.д.);
* Меню. Состоит из набора кнопок для перехода между различными элементами приложения.

Шаблон одного компонента может включать в себя шаблоны других компонентов, что позволяет создавать цепочки вложенных шаблонов. При этом родительский и дочерний компоненты способны обмениваться между собой данными и событиями при помощи связок Angular.

Также Angular позволяет связать шаблон компонента с его логикой, что способствует динамическому отображению информации на странице.

Приложение использует однообразный стиль всех интерфейсных элементов. За это отвечает библиотека компонентов PrimeNG, однако при необходимости можно поменять стили объектов интерфейса при помощи   
CSS-файлов. Причем заданный стиль применяется лишь к компоненту, чей CSS-файл был изменен.

1.2.2 Слой логики компонентов

Слой логики компонентов состоит из файлов на языке Typescript. В основном они содержат один класс для данного компонента, а также набор параметров компонента.

Каждый компонент Angular имеет жизненный цикл. При создании компонента во время загрузки страницы выполняются несколько функций инициализации и проверки, например, функция «ngOnInit()» позволяет выполнить некоторые предварительные действия перед созданием компонента. При обновлении информации компонента или при закрытии страницы также выполняются несколько функций.

Компоненты табличного типа в основном содержат логику обработки нажатия на кнопки и построения запроса данных таблицы серверу. При нажатии на кнопки создания и редактирования происходит вызов компонента типа регистрационная форма. В нём в случае создания отображаются пустые и заполненные стандартными значениями поля ввода, в случае редактирования – поля ввода, заполненные данными с сервера, относящиеся к выбранной для редактирования строчки таблицы.

Регистрационная форма автоматически проверяет введенные данные и запрещает проводить сохранение в случае, если они некорректны. Также запрещается сохранять данные, если не заполнены обязательные поля.

Для выпадающих списков регистрационных форм зачастую требуются табличные данные с сервера, поэтому перед отображением такие компоненты запрашивают всю необходимую информацию.

Некоторые поля в форме могут быть скрыты или недоступны для редактирования, что связано с требованиями бизнес-логики.

Компоненты меню зачастую имеют лишь шаблон, так как они не нуждаются в сложной логике, которую необходимо выносить в отдельные методы класса.

1.2.3 Слой отправки запросов

Слой отправки запросов состоит из сервисных классов (клиентских модулей доступа), которые полностью соответствуют точкам входа сервера. Данные сервисы автоматически генерируются при компиляции серверного приложения. Также при автоматической генерации создаются все необходимые для обмена данными объектов-обёртки. В методах сервисов создаётся базовый объект для HTTP-запроса, а затем отправляется по заданному HTTP-методу.

После отправки запроса и получения методы возвращают данные от сервера. Эта процедура является асинхронной, то есть главный поток приложения не блокируется на время ожидания данных от сервера.

1.3 Структура серверной части приложения

Серверная часть приложения используется для обработки запросов клиента и управления информацией в базе данных, а также отправки сообщений в случае ошибки.

Серверное приложение состоит из трёх слоёв:

* Слой обработки запросов;
* Слой бизнес-логики;
* Слой базы данных.

Между слоями и между серверной и клиентской частями данные перемещаются в виде объектов-обёрток.

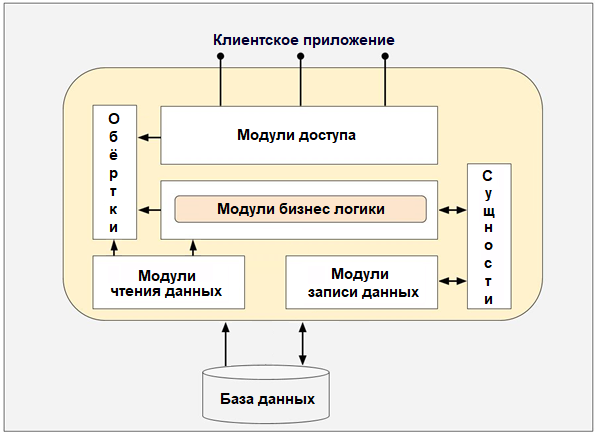


Рисунок 2 – Типичная структура серверного Spring приложения

Spring Core обеспечивает слабую связанность компонентов приложения при помощи инжекции зависимостей. Это механизм, позволяющий компонентам верхнего уровня не зависеть от реализации компонентов нижнего уровня, а лишь иметь ссылку на них для делегирования выполнения операций. Например, слой бизнес-логики не имеет информации о строении и реализации слоя базы данных, но способен делегировать ему операции сохранения, изменения и удаления информации в БД.

В этом подразделе будут рассмотрены структура и работа каждого слоя приложения.

1.3.1 Слой обработки запросов

Слой обработки запросов состоит из:

* Модулей доступа (контроллеров);
* Фильтров Spring Security;
* Технологий, используемых для получения и отправки запросов по протоколу HTTP.

Серверное приложение после развертывания подключается к некоторому порту сети и анализирует все запросы, отправленные на него.

Все запросы маршрутизируются при помощи точек входа – специальных URL-ссылок, составленных по REST и обеспечивающих однозначную идентификацию получателя запроса. С одной точкой входа может быть связано несколько маршрутов, отличающихся методами HTTP-запросов, часть из которых приведена ниже.

Методы HTTP-запросов:

1. GET – получение данных;
2. POST – отправка и сохранение данных;
3. PUT – отправка и изменение данных;
4. DELETE – удаление данных.

Запрос должен содержать в себе URL-ссылку на точку входа в приложении, а также метод HTTP-запроса. Также запрос может содержать тело в формате JSON-объекта – объекта типа «ключ-значение».

При разработке в классах модулей доступа также указываются ссылка и метод при помощи аннотаций.

При поступлении запроса Spring определяет, к какому модулю и к какому методу этого модуля запрос должен быть адресован.

Далее запрос проходит через каскады фильтров Spring Security, которые представляют собой последовательность следующих действий:

* Проверка правильности структуры данных в запросе;
* Авторизацию пользователя;
* Проверка доступности метода для пользователя;
* Дополнительные действия, заданные разработчиком.

При необходимости Spring Security позволяет изменять последовательность и набор фильтров, а также создавать пользовательские.

Доступность метода модуля доступа задаётся наличием у пользователя соответствующей роли. Необходимость определенной роли для метода контроллера указывается при помощи аннотации.

Если запрос не прошёл некоторый фильтр, то никаких действий с данными не происходит, в ответ на запрос отправляется сообщение об ошибке с описанием причины.

При успешном прохождении всех фильтров запрос перенаправляется в соответствующий метод контроллера. Контроллер обращается к слоям бизнес-логики и базы данных для проведения дальнейших операций с данными. В частности, он вызывает методы модулей бизнес-логики и модулей чтения данных.

После обработки запроса контроллер отправляет ответный запрос, содержащий необходимые данные. Например, при изменении информации в БД, обратно отправляется ответ, внутри которого находится объект, соответствующий измененной сущности в виде объекта-обёртки.

1.3.2 Слой бизнес-логики

Слой бизнес-логики состоит из модулей бизнес-логики (сервисов), выполняющих действия с данными путём их преобразования согласно правилам бизнес-логики. Сервисы – самая сложная и ответственная часть бизнес-приложения, так как от корректности их работы зависит корректность работы всего приложения.

Сервисы используют модули чтения данных, другие сервисы и модули записи данных для получения, сохранения, изменения и удаления данных.

Большинство методов сервисов являются транзакционными, то есть состоят из действий, выполняемых в рамках одной транзакции в БД. Это позволяет отменить изменения информации в БД при возникновении ошибки при выполнении метода.

В методах сервисов используются как сущности баз данных, так и объекты-обёртки. Изменение первых при завершении транзакции приводит к изменению информации в БД, а изменение вторых – нет.

Данные в сущностях поступают из модулей записи данных (репозиториев) или создаются внутри методов сервиса на основе объектов-обёрток и используются при сохранении и изменении информации в БД.

Данные в объектах-обёртках поступают в сервисы из компонентов верхнего уровня – контроллеров, а также из компонентов слоя базы данных –модулей чтения данных.

1.3.3 Слой базы данных

Слой базы данных состоит из:

* Конфигурации подключения к базе данных;
* Модулей записи данных;
* Модулей чтения данных.

В приложении применяется реляционная база данных вместе с PostgreSQL. Для соединения с ней используется Spring Data.

При настройке приложения производится конфигурация подключения к базе данных, выбор рабочей схемы и ввод данных пользователя, от лица которого будут выполняться все транзакции.

На основе таблицы в БД создается сущность (Java-объект), имеющая одноименные поля с соответствующими типами данных. При помощи технологии ORM фреймворка Hibernate сущность связывается с таблицей в БД.

На основе сущности создаются репозитории, предназначенные для чтения данных в виде сущностей, а также для их сохранения, изменения и удаления. Spring Data предоставляет базовый набор функций с сущностями в БД, который при необходимости можно расширить.

Также на основе сущности автоматически создаются Q-сущности – специальные классы QueryDSL, позволяющие создавать SQL-запросы на языке Java. На основе этих классов создаются модули чтения данных, используемые для быстрого чтения данных из нескольких таблиц БД и их гибкой упаковки в объекты-обёртки.

Главным отличием модулей чтения данных от репозиториев является запрет на изменение данных. Для этого все их методы объявляются транзакционными, но в режиме «только чтение».

В данном разделе ВКР была описана архитектура клиент-серверного приложения

2 Подготовка к разработке конфигуратора

В этом разделе отчёта описан процесс выполнения следующих задач:

1) Выявление перечня требований, отличающихся на предприятиях-заказчиках;

2) Выбор способа реализации конфигурации приложения;

3) Разбиение конфигуратора на логически завершённые разделы.

2.1 Перечень требований, настраиваемых конфигуратором

В этом подразделе приведен процесс выявления требований, настраиваемых конфигуратором.

Для составления перечня необходимо было провести анализ документов и их печатных форм, используемых на различных предприятиях, а также особенность их составления, заполнения и проведения.

Вышеупомянутая работа была выполнена с помощью коллег, бизнес-аналитиков. Они пояснили, в чём состоит отличие складов других предприятий. После анализа был получен следующий перечень правил:

1. Предприятия имеют разный набор документов;
2. В зависимости от того, с каким типом ТМЦ связан документ, он может иметь различную логику бизнес-процессов;
3. Предприятия могут иметь одинаково называемые документы, содержащие различные поля;
4. Существуют склады различных типов;
5. Доступность создания и проведения документа должна зависеть от типа склада;
6. Числовые поля документов могут иметь разное количество знаков после запятой, которое зависит от типа ТМЦ и предприятия;
7. Печатные формы одного и того же документа могут отличаться на различных предприятиях.

На основе этих правил было произведено разбиение конфигуратора на разделы, описанные в подразделе 2.3 текущего отчёта.

2.2 Выбор способа реализации конфигурации приложения

В этом подразделе описан процесс выбора способа реализации конфигурации клиент-серверного приложения.

Так как основным фреймворком серверной части приложения является Spring Boot, то все рассмотренные способы конфигурации основаны на нём.

Всего было рассмотрено три способа реализации конфигурации:

1. Конфигурационные файлы;
2. Конфигурация при помощи аннотаций Java и Spring;
3. Конфигурация при помощи базы данных и пользовательского интерфейса.

В следующих пункта текущего отчёта будут рассмотрены преимущества и недостатки упомянутых способов конфигурации.

2.2.1 Конфигурационные файлы

Фреймворк Spring Boot поддерживает конфигурацию при помощи конфигурационных файлов (файлов с расширениями «.yml», «.yaml», «.properties»).

Эти файлы хранят данные конфигурации в виде пары ключ-значение. Также они поддерживают вложенность ключей.

Преимуществами таких конфигурационных файлов являются:

1. Структурированность параметров;
2. Возможность разбиения конфигурации на отдельные файлы;
3. Хранение в папке приложения;
4. Поддержка изменения конфигурации без повторной компиляции – требуется лишь перезагрузка серверного приложения.

Недостатками являются:

1. Сложность использования для человека, не связанного с разработкой клиент-серверных приложений;
2. При неправильной конфигурации приложения его не получится запустить;
3. Настройка приложения проводится на стороне серверного приложения, следовательно, администратору требуется иметь постоянный доступ к файловой системе сервера;
4. Конфигурационные файлы поддерживают только английские названия параметров, что может вызывать трудности при настройке приложения.

2.2.2 Конфигурация при помощи аннотаций

Язык программирования Java поддерживает создание пользовательских аннотаций. По этой причине фреймворк Spring имеет в своём составе множество аннотаций, используемых при разработке. Однако их недостаточно для создания конфигурации, удовлетворяющей вышеупомянутым правилам, поэтому если выбрать этот способ конфигурации, то будет необходимо создать множество новых аннотаций, что более трудоёмко, чем создание конфигурационных файлов.

Преимуществами настройки при помощи аннотаций являются:

1. Возможность гибкой настройки поведения приложения;
2. Расширяемость при помощи наследования аннотаций;
3. Простота поддержки разработчиками.

Недостатками являются:

1. Для изменения конфигурации приложении необходимо изменять исходный код аннотаций, что требует от администратора навыков работы с аннотациями;
2. После изменения конфигурации необходимо перекомпилировать приложение;
3. При неправильной конфигурации приложение может: не запуститься; запуститься с нарушением работы некоторых функций и выдачей соответствующей ошибки; запуститься с нарушением работы некоторых функций без оповещения о неисправностях;
4. Перечень необходимых аннотаций может быть слишком большим, что негативно скажется на удобстве конфигурации.

2.2.3 Конфигурация при помощи базы данных и пользовательского интерфейса

Так как приложение имеет клиент-серверную архитектуру, то для конфигурации можно использовать пользовательский интерфейс приложения, связанный с таблицами в базе данных при помощи серверного приложения.

В этом случае разработка конфигуратора будет схожа с разработкой основной части приложения.

Преимуществами такой настройки являются:

1. Простое масштабирование конфигуратора, продиктованное архитектурой приложения;
2. Удобство использования даже для администратора, не являющегося квалифицированным специалистом по разработке клиент-серверных приложений;
3. Возможность вести историю изменений конфигурации;
4. Возможность структурирования конфигуратора;
5. Возможность создания более гибкой логики работы конфигуратора;
6. Администратор имеет доступ к конфигуратору через любой браузер, поддерживаемый приложением;
7. После конфигурации нет необходимости в перекомпиляции и перезапуске приложения;
8. При изменении конфигурации, например, при удалении устаревшего типа документа, приложение будет продолжать работать без ошибок.

Недостатками конфигурации через базу данных и пользовательский интерфейс являются:

1. Увеличенное время разработки конфигуратора, в виду необходимости создания пользовательского интерфейса;
2. После реализации интерфейсной и серверной части конфигуратора требуется также внести изменения в бизнес-логику приложения.

2.3 Принятие решения по способу реализации конфигуратора и разбиение его на разделы

После анализа способов конфигурации было принято решение использовать настройку через базу данных и пользовательский интерфейс, так как такое решение является наиболее гибким, понятным для потенциального пользователя, масштабируемым, а также соответствует концепции всего приложение, следовательно, проще поддерживается.

Для удобства использования интерфейсную часть конфигуратора было необходимо разбить на разделы, согласно процессу настройки приложения, подразумевающему следующие действия:

1. Создание типов операций – особых действий, определяющих поведение приложения при работе с документом;
2. Создания типа документа и его связи с операциями и печатной формой;
3. Настройка доступности документов для различных типов складов;
4. Настройка полей каждого документа;
5. Настройка отправителей и получателей для различных типов складов;
6. Настройка точности представления числовых данных в печатных формах.

Этот список действий в будущем может быть дополнен, что приведёт к созданию нового раздела конфигуратора.

Полученный список разделов конфигуратора представлен в следующем разделе.

В этом разделе ВКР было проведено исследование по выявлению требований, настраиваемых конфигуратором, описаны способы реализации конфигуратора, из которых впоследствии был выбран наиболее подходящий под требования.

3 Описание конфигуратора

В этом разделе размещено описание конфигуратора и его составных частей, а также иллюстрации с пользовательским интерфейсом. В виду конфиденциальности предприятия исходный код конфигуратора в отчёте о ВКР не будет приведён.

Приложение складского учёта – приложение, созданное командой «Призма» на замену старой системе складского учёта «СКЛАД», написанной на языке программирования Delphi. В данный момент оно находится в стадии разработки. Планируется использовать его на различных предприятиях госкорпорации «Росатом», поэтому необходимо обеспечить универсальность приложения. Эту функцию должен будет выполнять конфигуратор складского учёта.

Конфигуратор складского учёта – специальная вкладка приложения «СКЛАД 2.0», позволяющая быстро настроить переменные в приложении для использования приложения с учётом требований пользователя.

Для использования конфигуратора не обязательно знать языки программирования, а тем более программировать, поэтому с настройкой справится любой бизнес-аналитик. При необходимости конфигурацию приложения можно изменить без прерывания производственного процесса.

Для удобства использования функционал конфигуратора был разделён на части и размещён в разделы.

В рамках конфигуратора были реализованы следующие разделы:

* Конфигуратор точности материальных ресурсов;
* Конфигуратор типов операций;
* Конфигуратор типов документов;
* Конфигуратор доступных типов документов;
* Конфигуратор форм;
* Конфигуратор отправителей/получателей.

3.1 Конфигуратор точности материальных ресурсов

Конфигуратор точности материальных ресурсов ­– раздел конфигуратора, включающий в себя набор функций, позволяющий настраивать отображение числовых данных, представляемых в виде десятичных дробей, в печатных формах документов в зависимости от типа материального ресурса.

Формулировка проблемы, решаемой этим разделом конфигуратора: на складах каждого предприятия могут храниться различные материальные ресурсы (драгоценные металлы, химические вещества, оборудование и т.д.). При печати различных документов для каждого материального ресурса указывается его количество. Это количество должно иметь некоторое количество знаков после запятой, установленное нормами предприятия. Так как приложение по умолчанию хранило количество любого ресурса с точность в 6 знаков, то и выводилось оно с такой же точностью, что противоречило требованиям предприятий-заказчиков.

Было необходимо разработать функционал, позволяющий настроить точность каждого материального ресурса.

Типы материальных ресурсов хранились в отдельной таблице-справочнике в базе данных, который запрещено изменять. Для сохранений настроек конфигуратора точности была создана отдельная таблица в БД, одним из столбцов которой оказался ID типа материального ресурса. Двумя другими столбцами являются значения точности для количества ресурса и для количества чистого ресурса.

На основе спроектированной таблицы в БД были созданы соответствующие классы приложения:

* Сущность;
* Модуль чтения данных;
* Модуль записи данных;
* Модуль бизнес-логики;
* Обёртка;
* Модуль доступа.

Эти классы разделяют обязанности по обработке запросов и передаче/изменению данных.

Параллельно был реализован Angular-компонент для конфигуратора.

После этого была настроена следующая бизнес-логика:

* По умолчанию точность каждого материального ресурса выставляется на 3 знака после запятой;
* Пользователь при загрузке страницы получает таблицу всех материальных ресурсов доступных на предприятии и точность каждого из них;

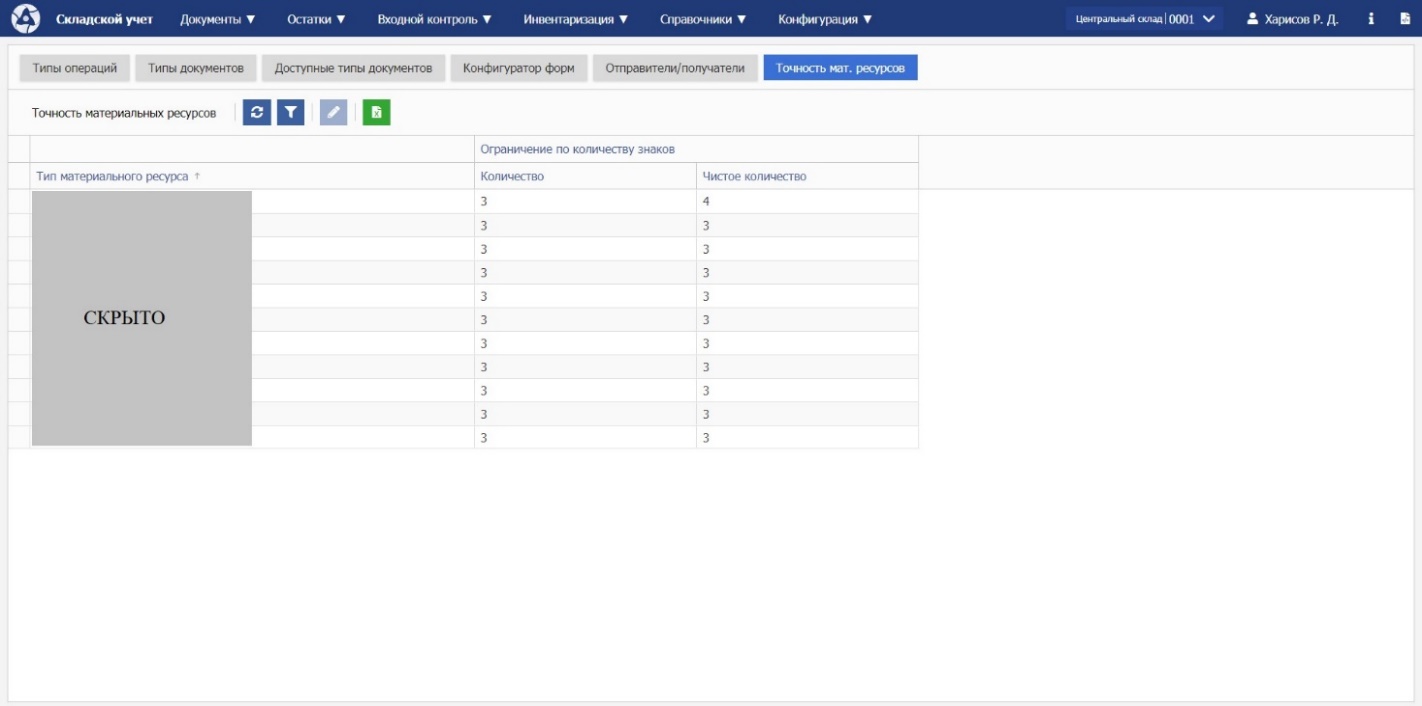


Рисунок 3 – Интерфейс конфигуратора точности материальных ресурсов

* При необходимости пользователь может обновить таблицу, включить фильтры, экспортировать таблицу в Microsoft Excel файл или изменить значение в таблице;
* Для редактирования необходимо предварительно выбрать строчку, а затем нажать на «карандаш» сверху таблицы;

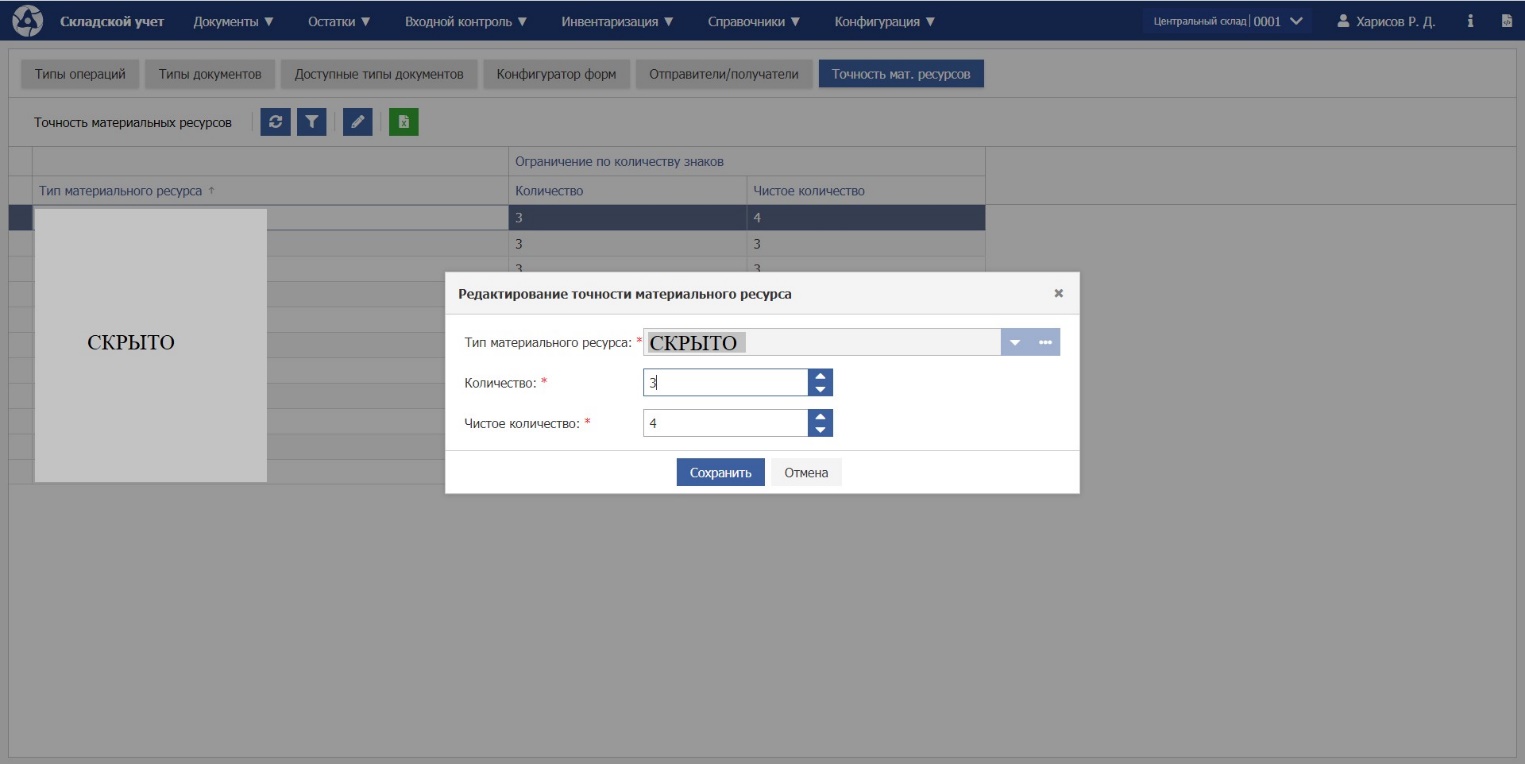


Рисунок 4 – Окно редактирования точности

* Интерфейс не позволяет выбрать значения, превышающие 6 и меньше 1, это было продиктовано бизнес-логикой;
* При сохранении изменений таблица обновляется и автоматически выбирается отредактированная строка.

Разработанный интерфейс полностью удовлетворяет поставленной задаче.

3.2 Конфигуратор типов операций

В этом и последующих подразделах будет рассмотрен процесс создания и настройки документа «Накладная на передачу ТМЦ на входной контроль».

Конфигуратор типов операций – раздел конфигуратора, включающий в себя набор функций, позволяющий создавать, редактировать типы операций, применяемых в приложении, а также редактировать бухгалтерские операции и типы материальных ресурсов, связанных с ними.

Формулировка проблемы, решаемой этим разделом конфигуратора: существует перечень операций, которые можно проводить с материальными ценностями склада. При помощи операций кладовщик способен чётко определить поведение приложения. При этом каждую операцию разрешено проводить только с определенными типами материальных ресурсов. При выполнении некоторых операций должен создаваться соответствующий документ. Однако перечень документов, а, следовательно, и перечень операций на каждом предприятии может отличаться.

Для решения данной проблемы был создан конфигуратор типов операций.

Он представляет из себя отдельную вкладку конфигуратора, на которой в верхней таблице указывается список созданных операций и кнопки для создания, редактирования или создания дочерней операции. Над таблицей также располагаются стандартные кнопки для обновления, фильтрации/сортировки или экспорта таблицы в Excel-файл.

Например, для создания вышеупомянутого документа требуется создать две операции:

* Передача ТМЦ со склада на входной контроль;
* Передача ДМ со склада на входной контроль.

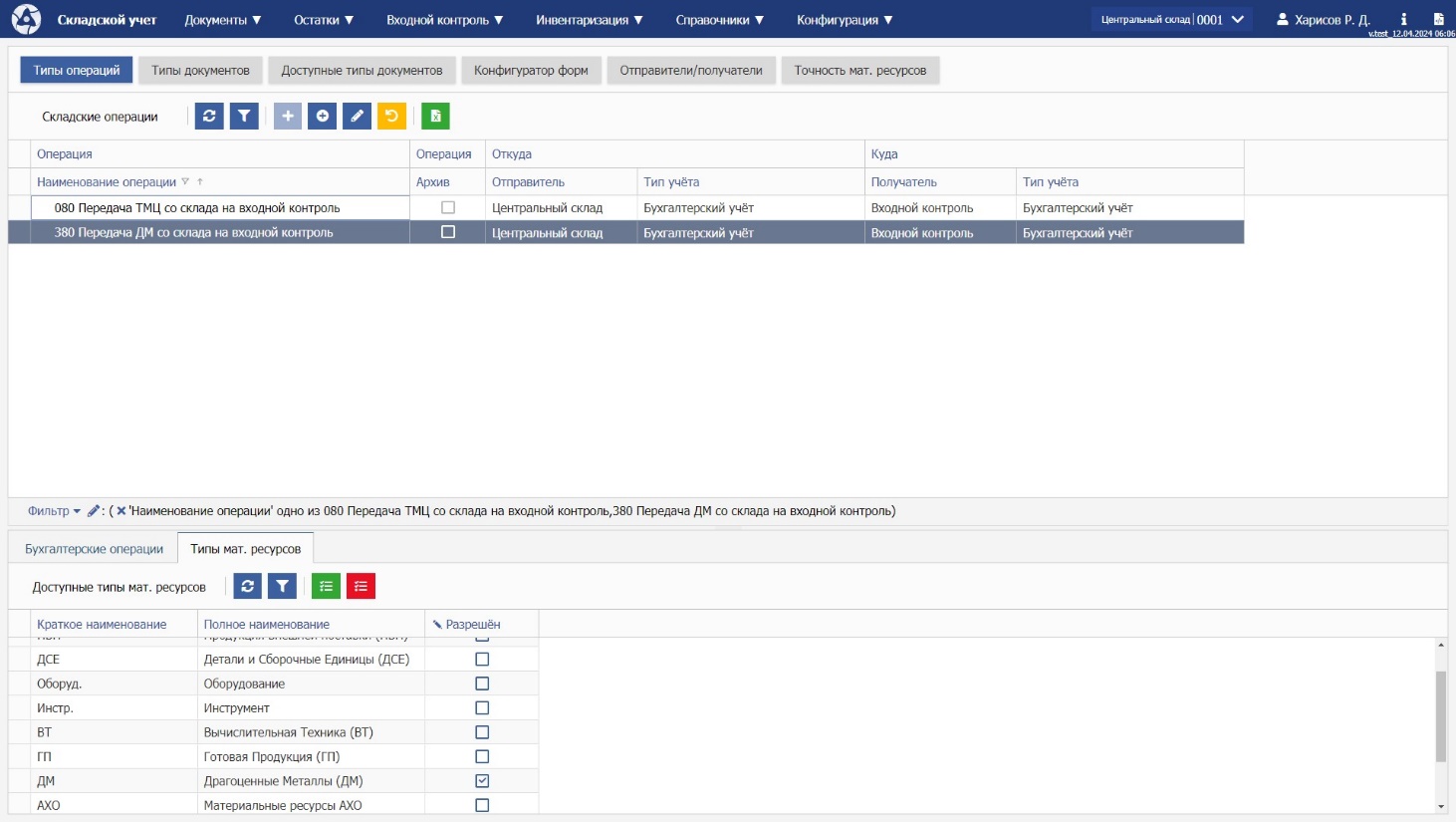


Рисунок 5 – Конфигуратор типов операций

При редактировании операции есть возможность изменять все поля сущности, кроме названия.

Перечень полей, приведенных в окне редактирования свойств операций

1. Отправитель – тип склада-отправителя для заданной операции;
2. Получатель – тип склада-получателя для заданной операции;
3. Тип учёта (откуда/куда);
4. Архив – признак архива.

Признак архива используется для отключения возможности создания документа с этой операцией. Устаревшие операции не удаляются из базы данных, так как это может привести к ошибкам приложения при работе со старыми документами.

При нажатии на кнопку добавления дочерней операции появляется аналогичное окно, однако для сохранения дочерней операции необходимо, чтобы её название отличалось от родительского.

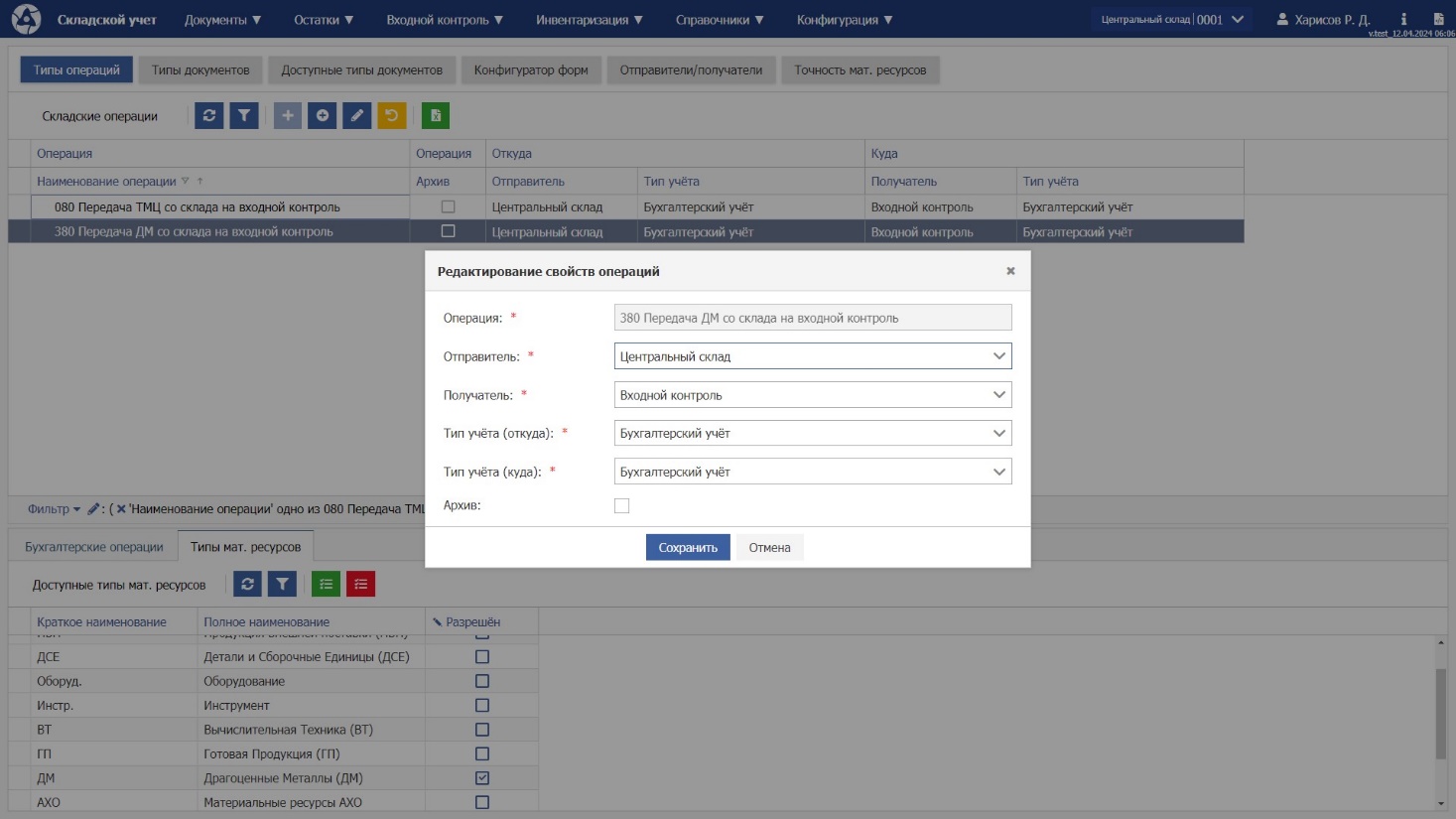


Рисунок 6 – Окно редактирования операции

Отправители и получатели указаны в справочнике приложения. Их перечень приведён на рисунке 7.

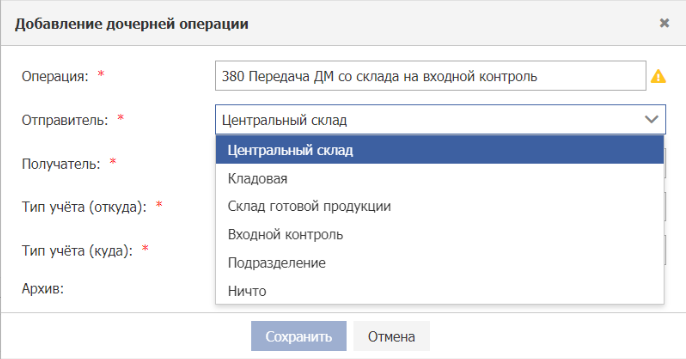


Рисунок 7 – Перечень отправителей и получателей.

В нижней таблице располагаются две вкладки: бухгалтерские операции и типы материальных ресурсов.

Первая вкладка позволяет связать операцию склада с операцией финансового сервиса. В ней находится список бухгалтерских операций, кнопки создания и удаления, а также стандартные для таблиц кнопки.

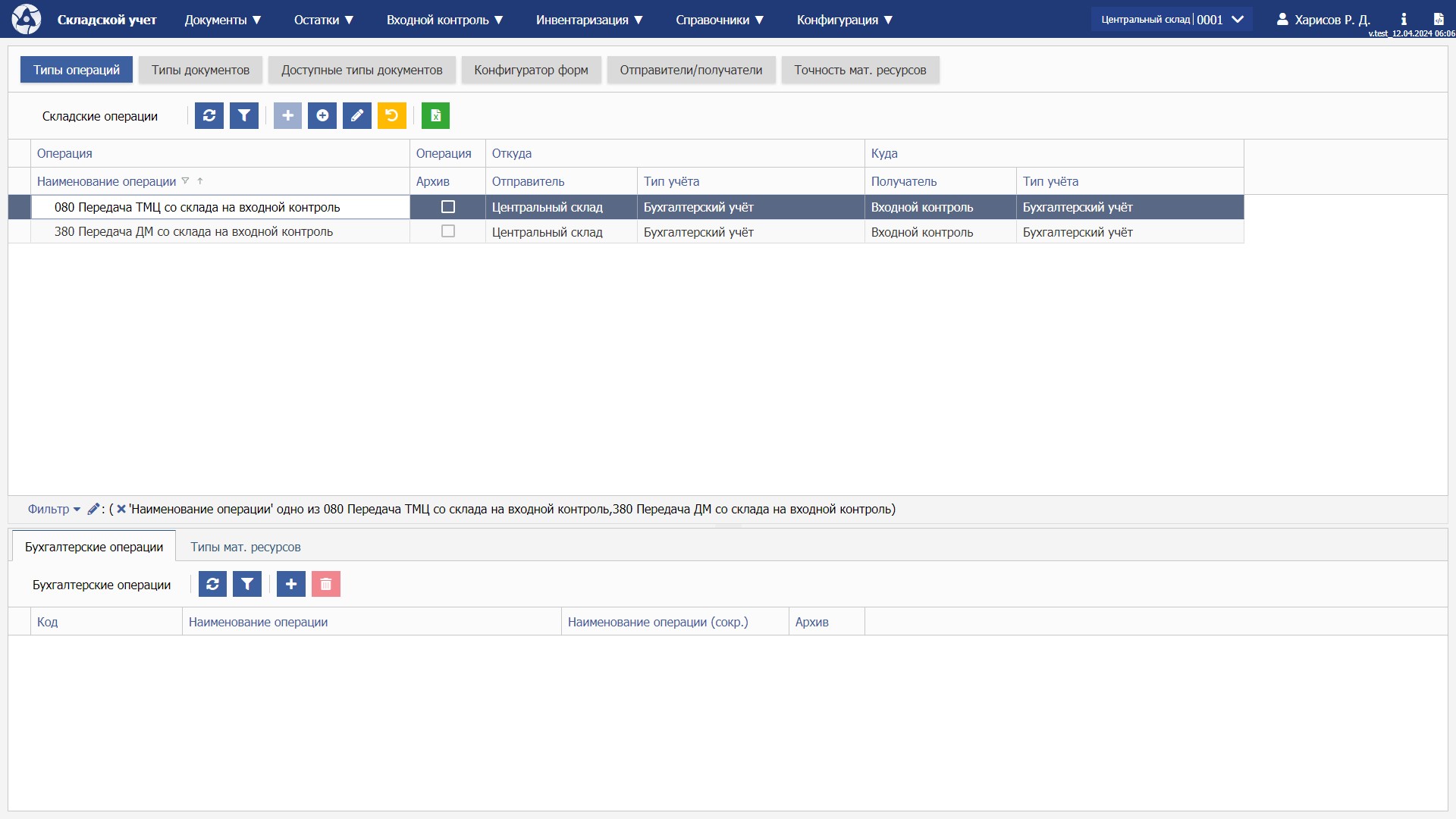


Рисунок 8 – Вкладка «Бухгалтерские операции»

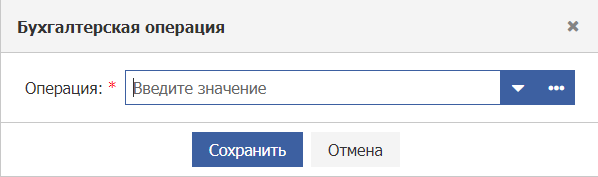


Рисунок 9 – Окно создания связи бухгалтерской операции и складской операции

В этом отчёте не будет приведён перечень бухгалтерских операций, так как это конфиденциальная информация.

Вторая вкладка позволяет указать, с какими типами материальных ресурсов можно проводить данную операцию. Это необходимо для установки ограничений действий работников склада согласно бизнес-процессу, а также для упрощения поиска материальных ценностей для проведения операции путём автоматической фильтрации их списка.

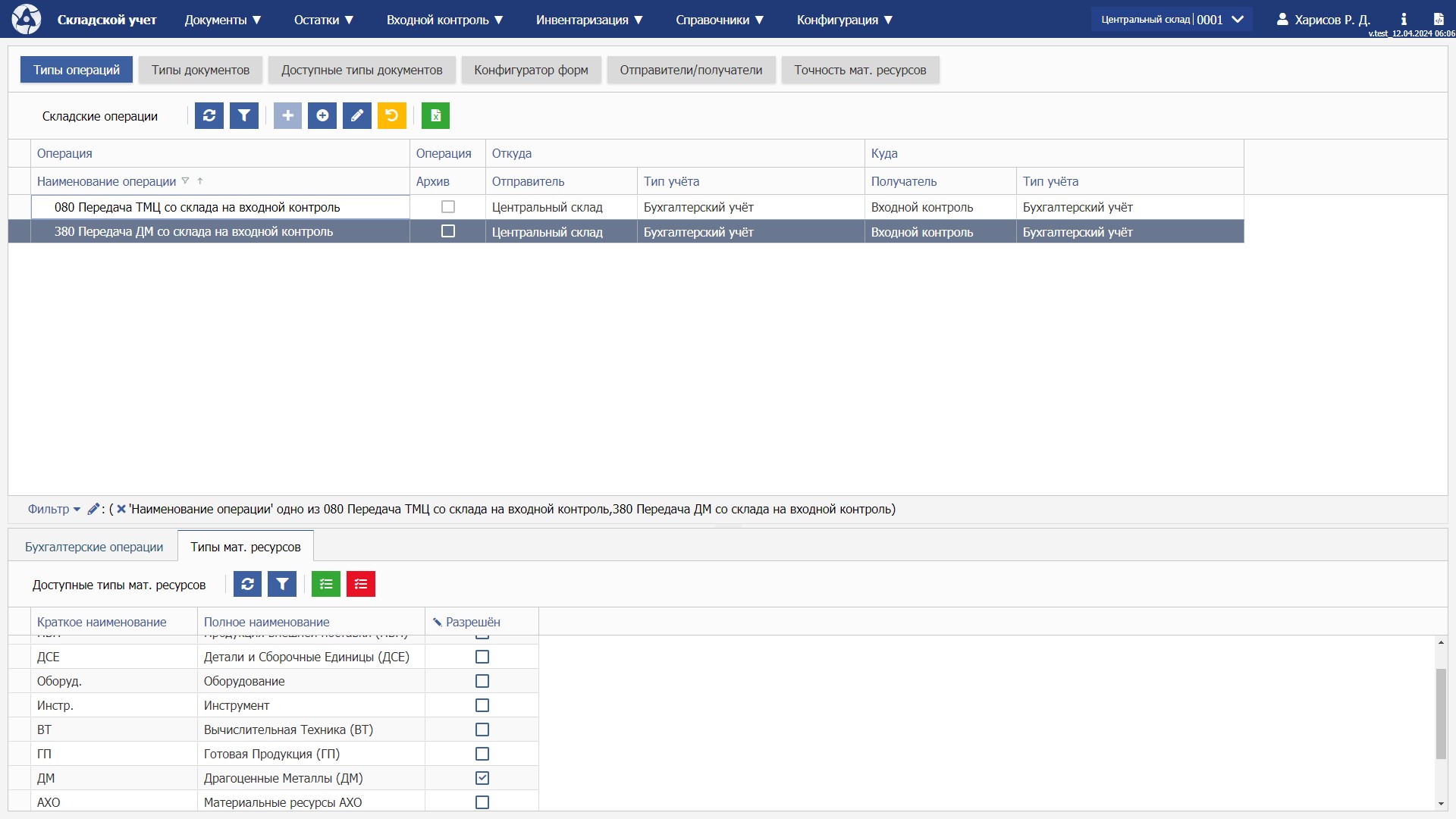


Рисунок 10 – Вкладка «Типы мат. ресурсов»

Список материальных ресурсов взят из справочника.

Сверху расположены кнопки, позволяющие выбрать все материальные ресурсы или наоборот убрать все, а также стандартные для таблиц кнопки.

3.3 Конфигуратор типов документов

Конфигуратор типов документов – раздел конфигуратора, включающий в себя набор функций, позволяющий создавать и редактировать типы документов, а также их связи с типами операций и шаблонами печатных формам, используемых приложением.

Формулировка проблемы, решаемой этим разделом конфигуратора: после создания типа операции, необходимо создать тип документа, который будет отвечать данной операции. У одного документа, может быть, несколько различных операций, а также своя печатная форма.

Для решения данной проблемы был создан конфигуратор типов документов.

В верхней таблице располагается список созданных типов документов и кнопки для их создания, редактирования, а также набор стандартных для всех таблиц кнопок. На рисунке 11 показан интерфейс конфигуратора с созданным вышеупомянутым документом.

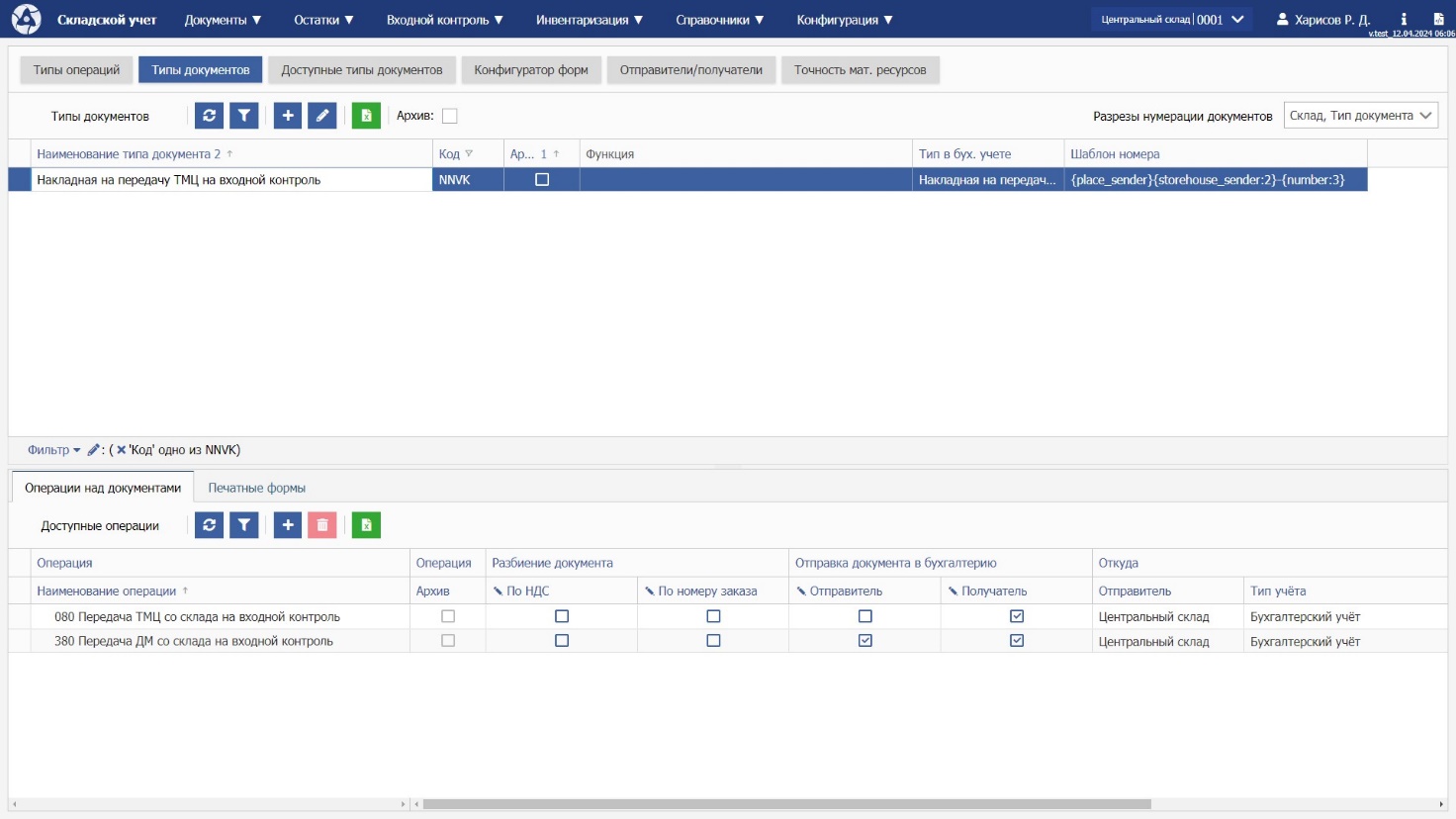


Рисунок 11 – Конфигуратор типов документов

При создании или редактировании документа открываются одинаковые окна (только при редактировании поля формы заполнены текущими значениями).

В форме есть возможность изменить следующие поля:

1. Код – специальный код документа, использующийся в приложении;
2. Наименование – название документа, используемое в пользовательском интерфейсе приложения;
3. Архив – признак архива;
4. Тип в бух. учёте – тип документа финансового сервиса;
5. Функция – специальное назначение документа, используемое для точной настройки поведения приложения при работе с создаваемым/редактируемым документом;
6. Шаблон номера – поле для настройки автоматической генерации номера документа.

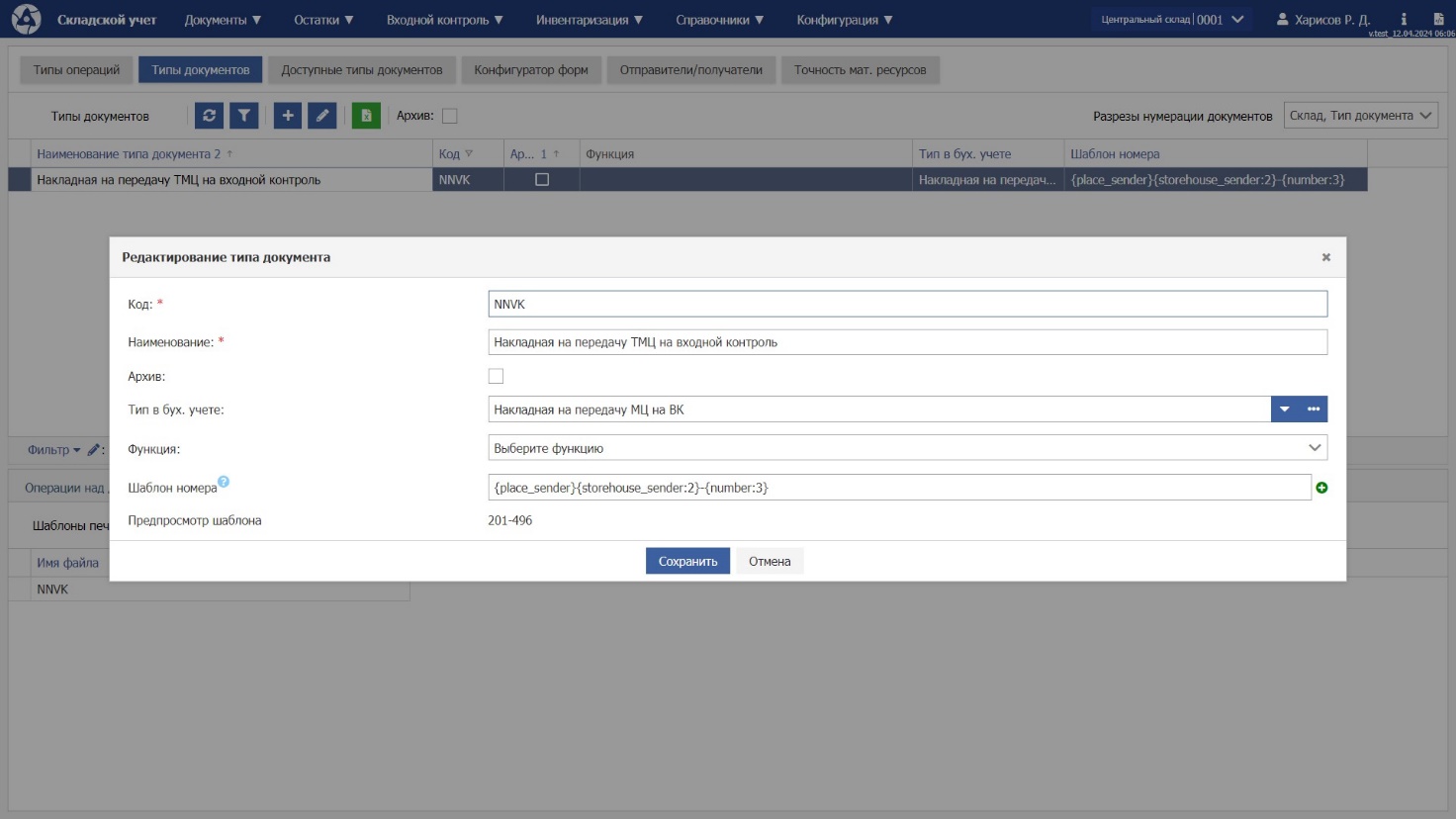


Рисунок 12 – Окно редактирования типа документа

Для генерации шаблона номера существует возможность написать шаблон-строку согласно следующему синтаксису, размещенному в легенде (легенда появляется при наведении на «?» рядом с шаблоном номера):

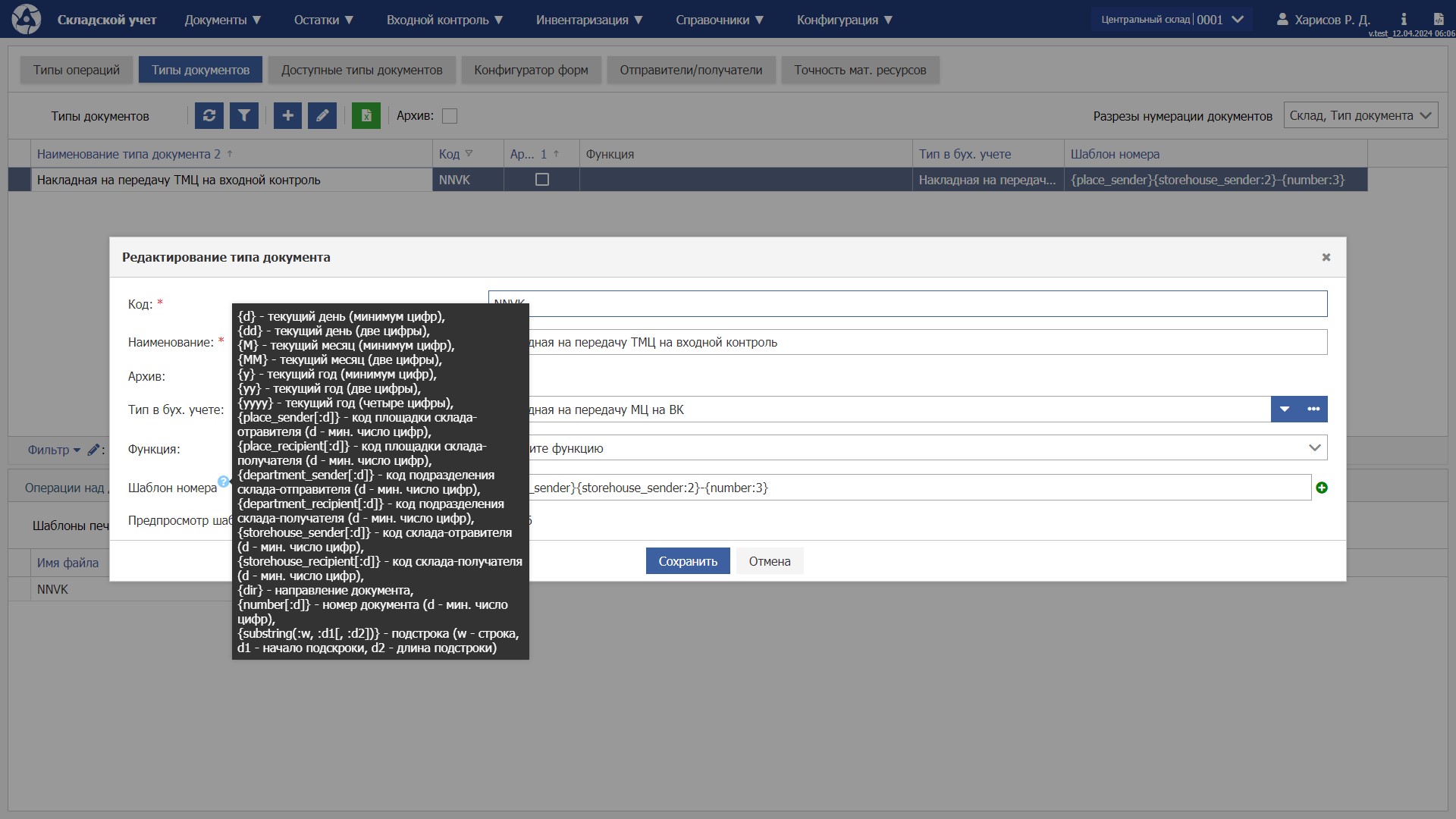


Рисунок 13 – Легенда для шаблон-строки шаблона номера документа

Также есть возможность создания шаблона при помощи кнопки добавления элементов строки (при нажатии на «+» справа от поля ввода). Пользователь имеет возможность выбрать тип добавляемого элемента шаблона из следующего списка:

1. Дата
2. Площадка
3. Подразделение
4. Склад
5. Прочее

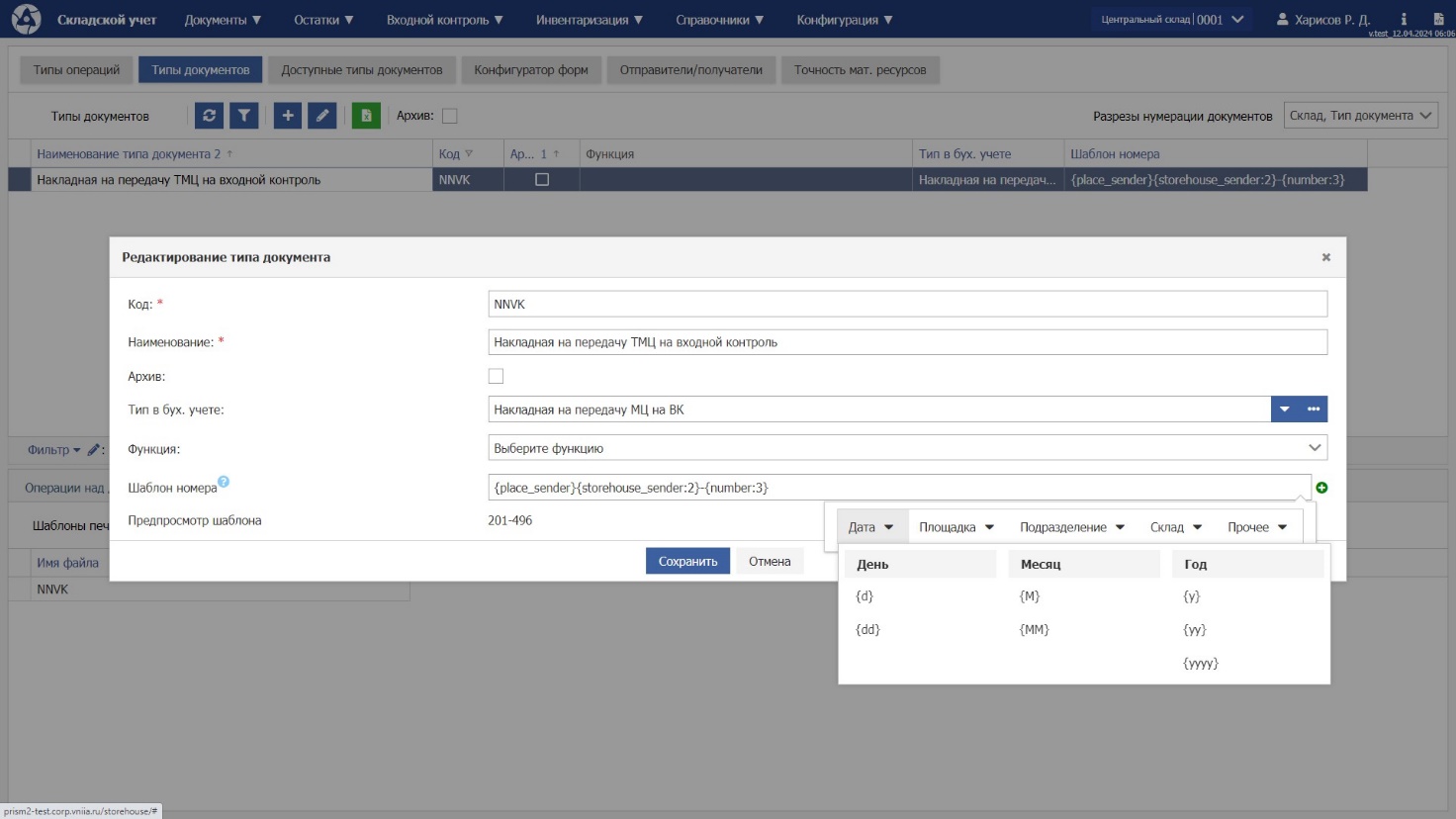


Рисунок 14 – Добавление элемента шаблон-строки шаблона номера документа

В нижней таблице расположено две вкладки: операции над документами и печатные формы.

В первой вкладке существует возможность связать документ с ранее созданными операциями. Во вкладке располагается список связанных с выбранным документом операций и кнопки добавления/удаления, а также стандартные для табличных компонентов кнопки. При необходимости можно указать тип разбиения документа: по НДС или по номеру заказа, а также того, кто может отправить (отправитель и получатель) документ в бухгалтерию.

На рисунке 15 созданный документ был связан с ранее созданными операциями.

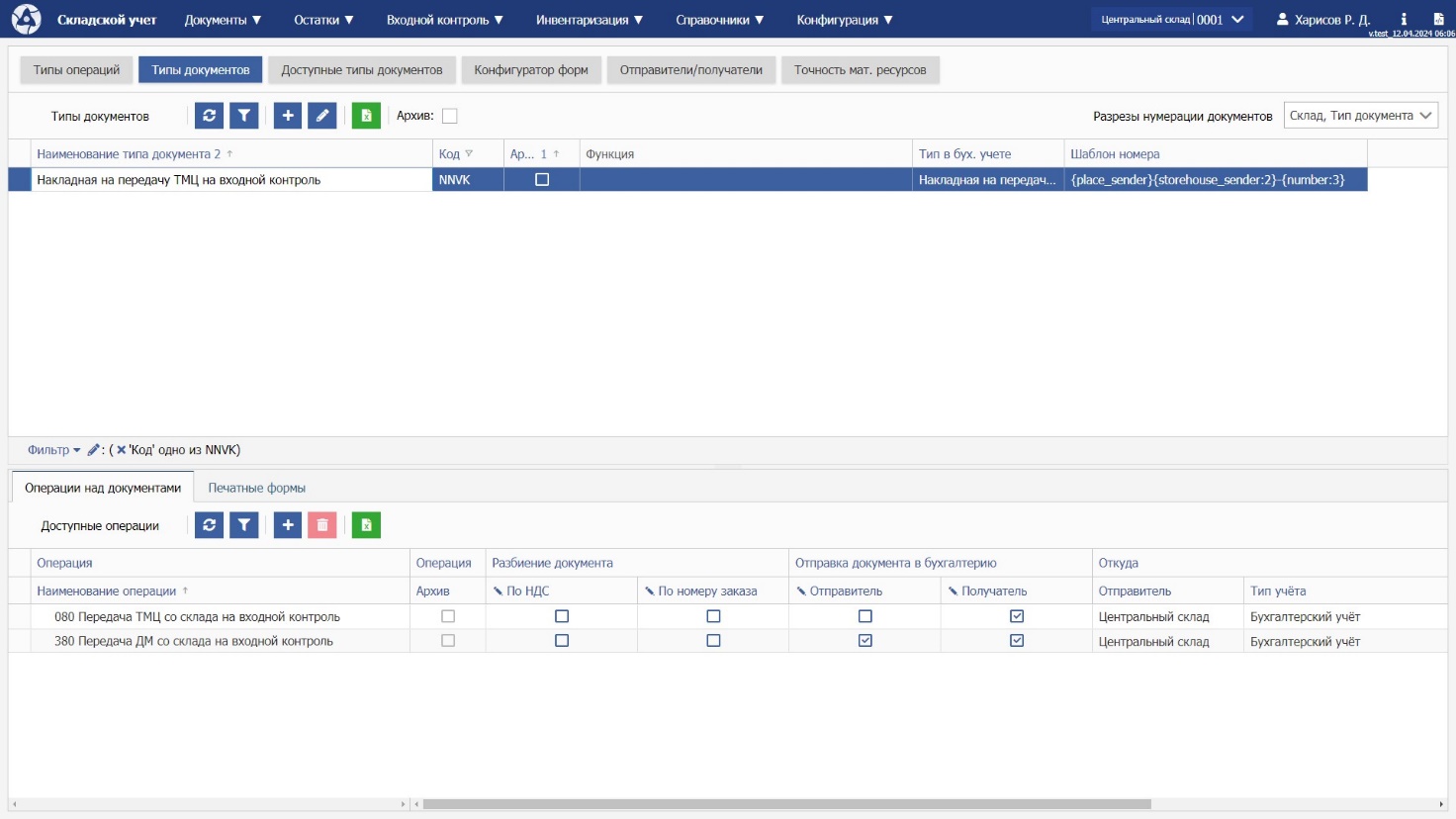


Рисунок 15 – Вкладка «Операции над документами»

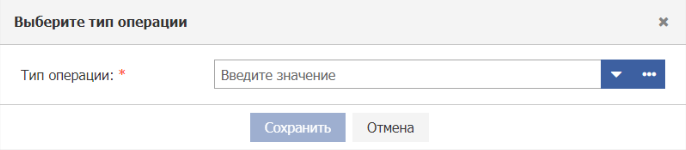


Рисунок 16 – Окно создания связи операции с документом

При нажатии кнопки «…» открывается окно, содержащее таблицу для выбора ранее созданных операций и позволяющее использовать фильтрацию для поиска.

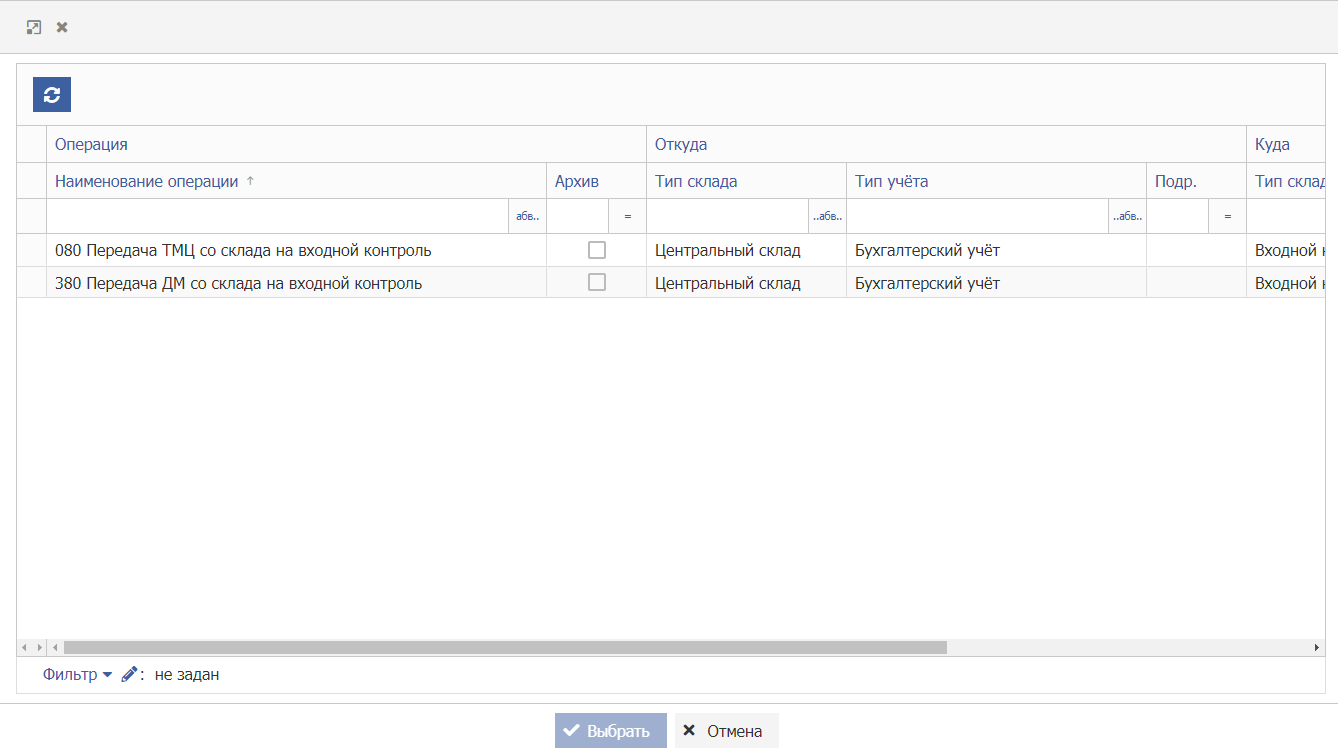


Рисунок 17 – Окно с таблицей операций

Во второй вкладке можно указать печатные формы, соответствующие выбранному документу. В данный момент не реализована печать документов, у которых есть несколько печатных форм, но если это понадобится, то в этой вкладке можно будет создать несколько записей.

На рисунке 18 созданный документ был связан с шаблоном печатной формы.

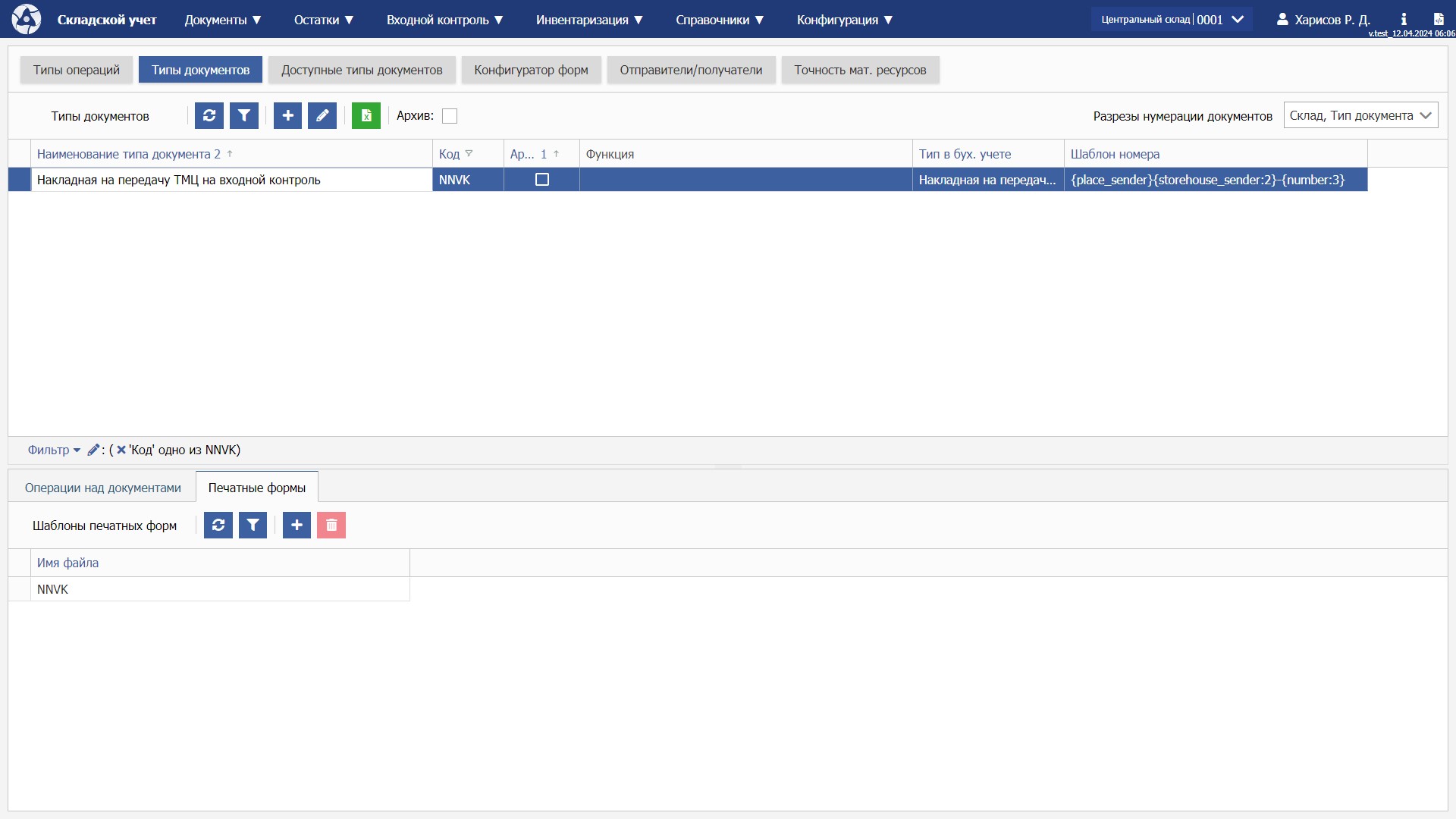


Рисунок 18 – Вкладка «Печатные формы»

Во вкладке располагаются список указанных печатных форм и кнопки создания/удаления, а также стандартные для всех таблиц кнопки.

После создания документа в приложении пользователи могут запросить его печатную форму. На основе полей и позиций документа на серверной части приложения строится PDF-файл документа, который загружается на клиентскую часть и отображается в окне пользовательского интерфейса. При необходимости работник склада имеет возможность скачать файл на свой персональный компьютер или отправить его на печать.

3.4 Конфигуратор доступных типов документов

Конфигуратор доступных типов документов – раздел конфигуратора, включающий в себя функциональность

Формулировка проблемы: в приложении окно создания документа стандартизировано, следовательно, любой работник может создать любой документ, что не удовлетворяет принципу разделения обязанностей.

Для решения этой проблемы был создан конфигуратор доступных типов документов.



Рисунок 19 – Вкладка «Доступные типы документов»

Сначала в левой таблице выбирается ответственная сущность склада, которой необходимо открыть доступ к документам склада. Это делается в таблице «Тип или назначение склада» в левом верхнем углу вкладки.

Так как в этом отчёте рассматривается процесс создания и настройки документа, связанного со входным контролем, то на рисунках в этом разделе будет выбран склад типа «Входной контроль».



Рисунок 20 – Таблица «Тип или назначение склада»

Вкладка «Входной контроль» содержит лишь одну строку, однако некоторые вкладки состоят из большего количества строк. Например, вкладка «Центральный склад» содержит 6 строк.

Далее в таблице «Типы документов» находятся две таблицы: доступные и недоступные документы. Перемещение документов между ними происходит путем нажатия кнопок «<<», «<», «>», «>>».



Рисунок 21 – Таблица «Типы документов»

При необходимости есть возможность открыть или закрыть доступ для конкретных операций документа в аналогичной таблице для операций в нижней части вкладки.

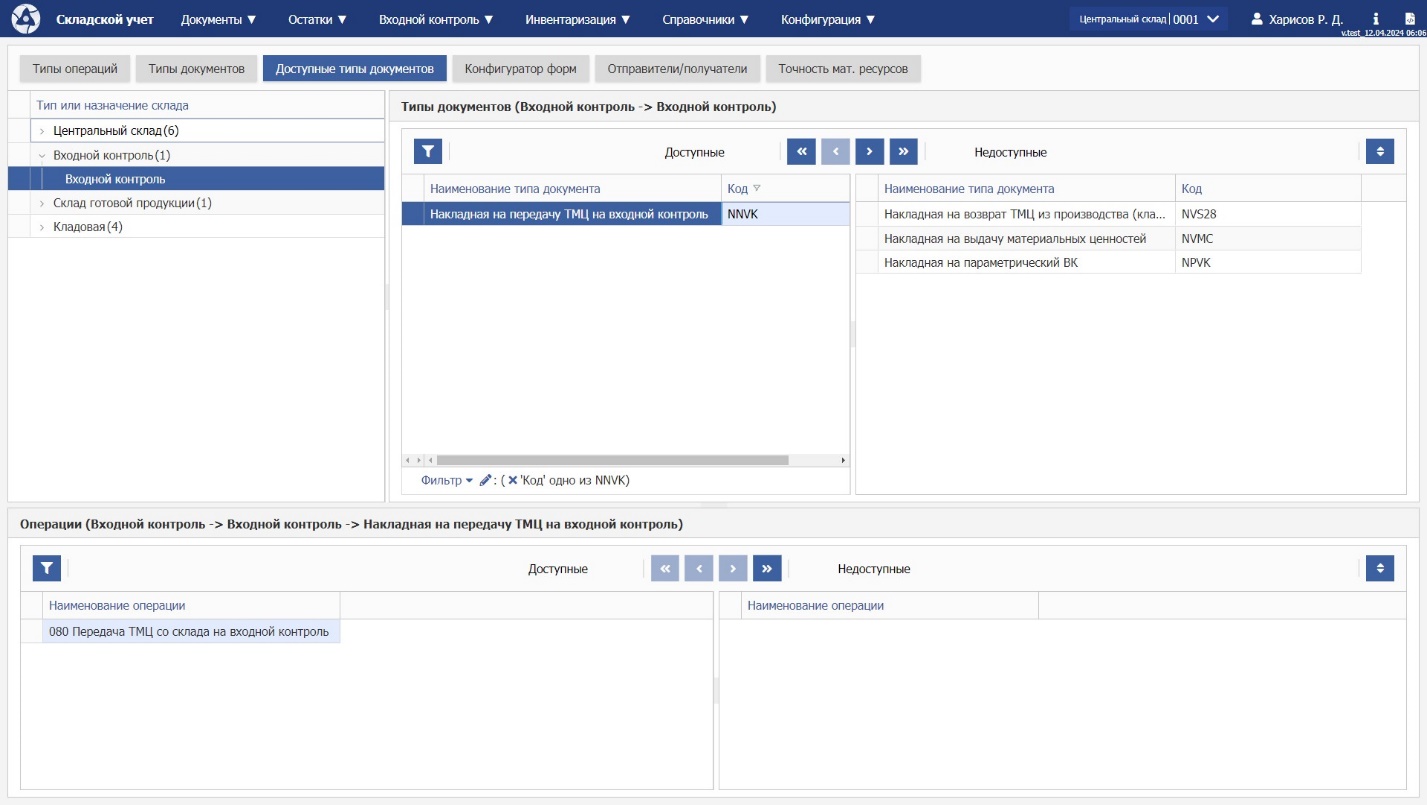


Рисунок 22 – Таблица «Операции»

Это может быть полезно в случае, если у одного документа есть несколько связанных с ним операций, однако бизнес-процесс запрещает создавать документы с одной из операций на одном из типов складов.

3.5 Конфигуратор форм

Конфигуратор форм – раздел конфигуратора, включающий в себя набор функций, позволяющий настроить поля форм создания и редактирования документов склада, используемых в приложении

Формулировка проблемы, решаемой этим разделом конфигуратора: одни и те же документы на складах различных предприятий могут иметь различные поля, в том числе различную доступность полей. Для предотвращения перегрузки пользовательского интерфейса и для правильной работы бизнес-логики некоторых документов необходимо ограничить набор видимых, обязательных и редактируемых полей документов.

Для решения этой проблемы был создан конфигуратор форм.

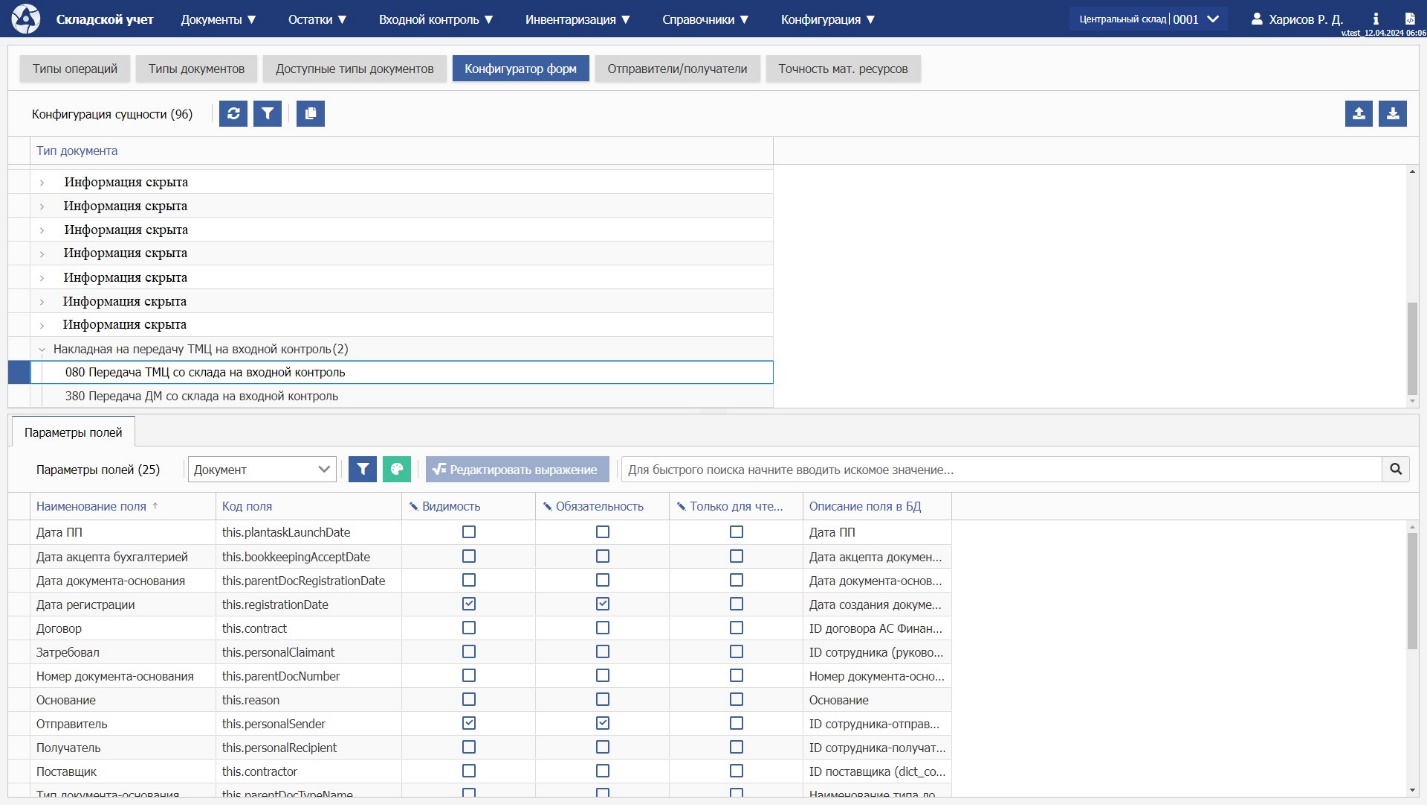


Рисунок 23 – Конфигуратор форм

В верхней таблице указаны ранее созданные типы документов, а также операции, связанные с ними. Для удобства использования операции сгруппированы по типам документов.

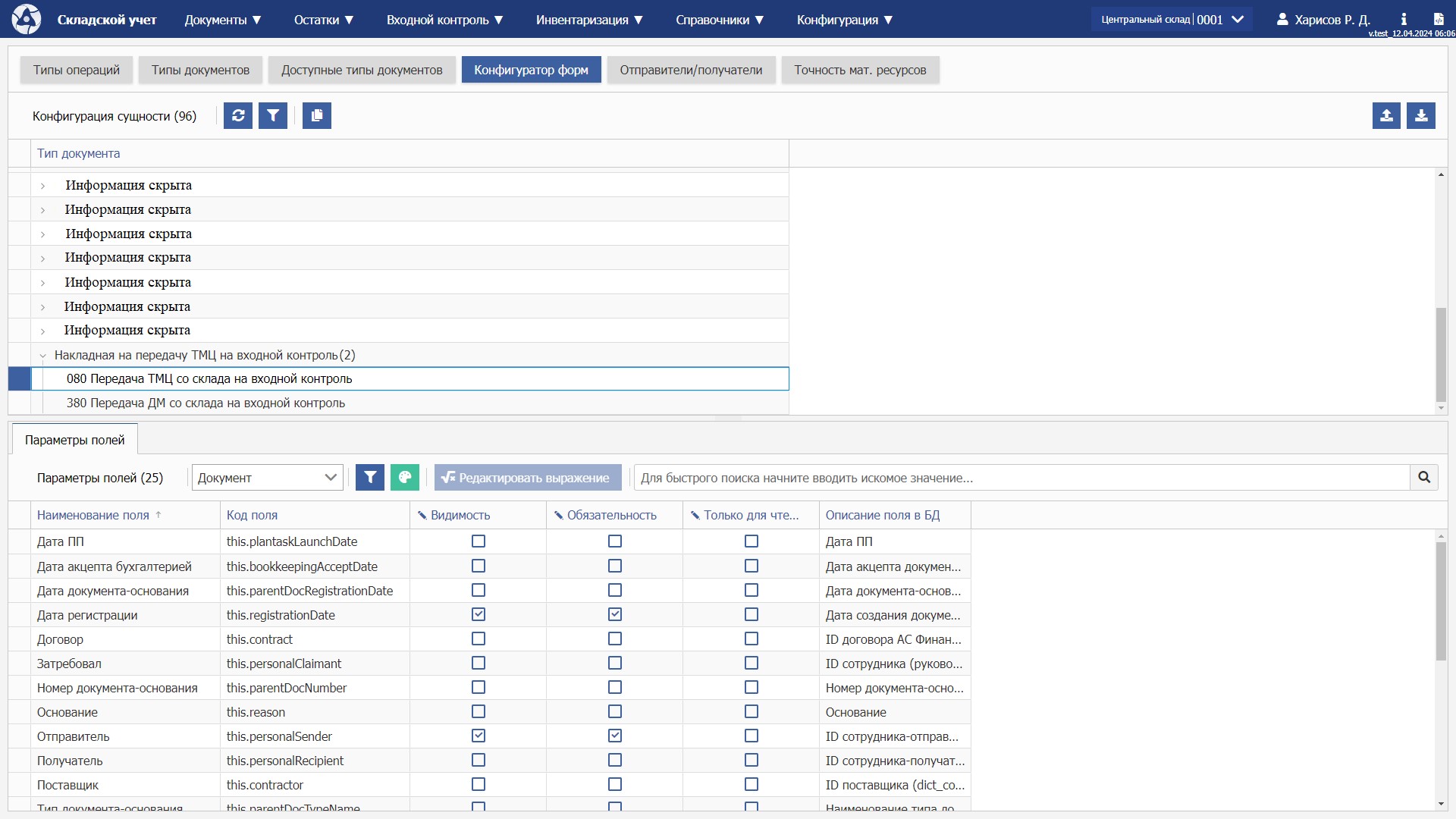


Рисунок 24 – Таблица с документами и операциями

После выбора операции существует возможность скопировать её конфигурацию. Для этого необходимо нажать правую верхнюю кнопку на рисунке 25. После копирования конфигурации в буфер обмена скопированная конфигурация станет выделяться штрихпунктирной линией, а также появятся кнопки вставки конфигурации из буфера и очистки буфера, показанные на рисунках 25, 26.

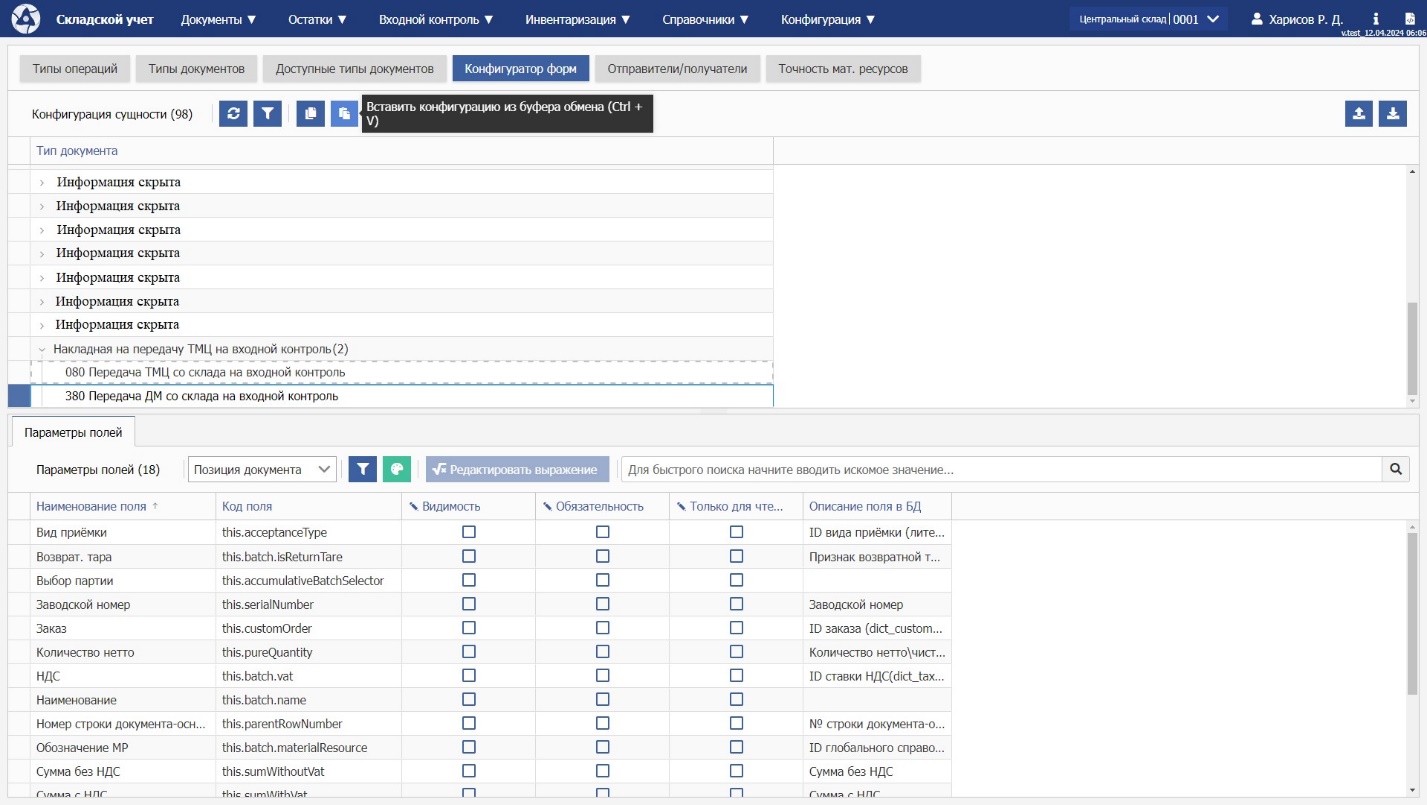


Рисунок 25 – Кнопка «Вставить конфигурацию из буфера обмена»

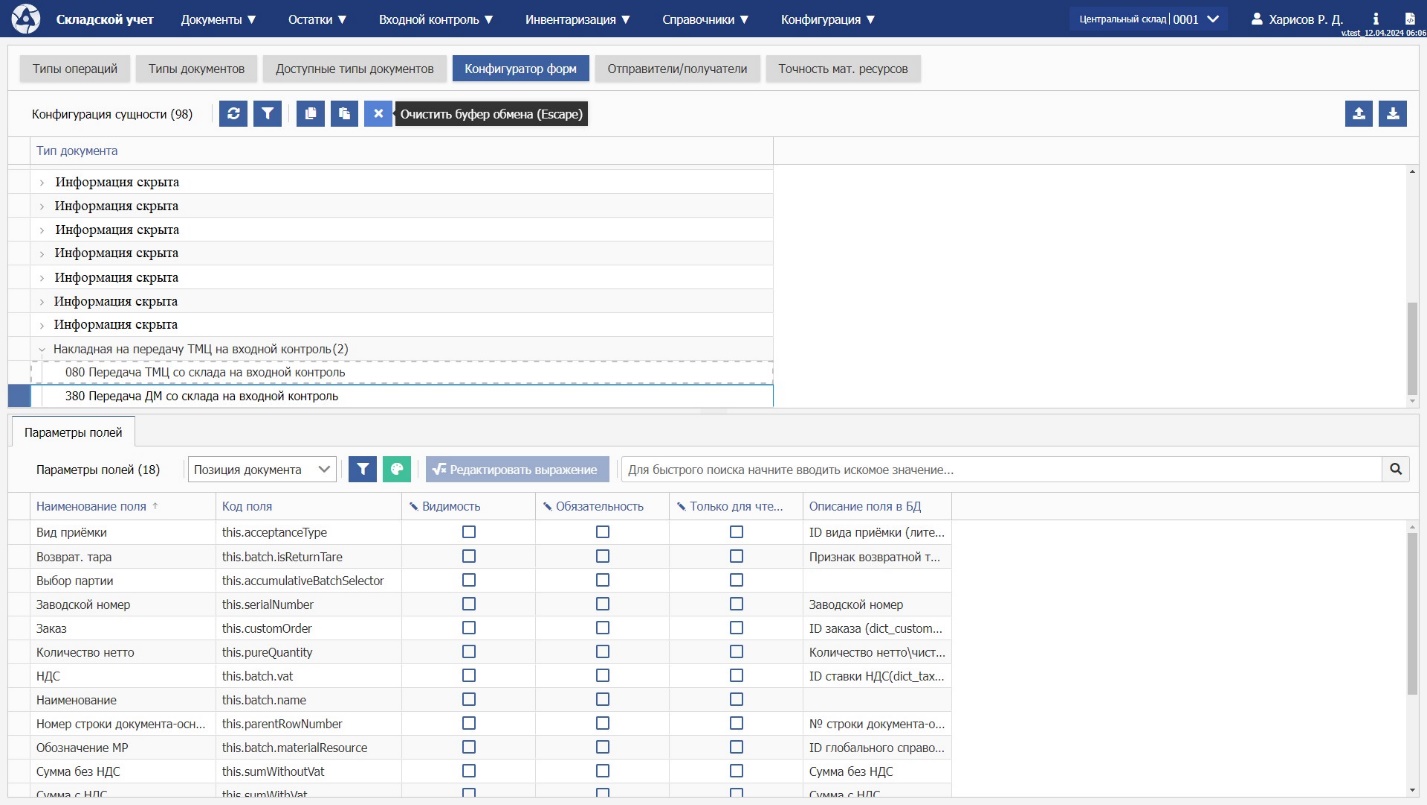


Рисунок 26 – Кнопка «Очистить буфер обмена»

Для каждой из вышеупомянутых кнопок, предназначенных для работы с буфером обмена, заданы привычные для пользователей персональных компьютеров сочетания клавиш:

* Для копирования из буфера: CTRL+C;
* Для вставки из буфера: CTRL+V;
* Для очистки буфера: ESC.

Помимо копирования конфигурации одной пары документ-операции в конфигураторе форм предусмотрена возможность экспортирования и импортирования конфигурации. Для этого необходимо использовать две кнопки в правом верхнем углу вкладки.

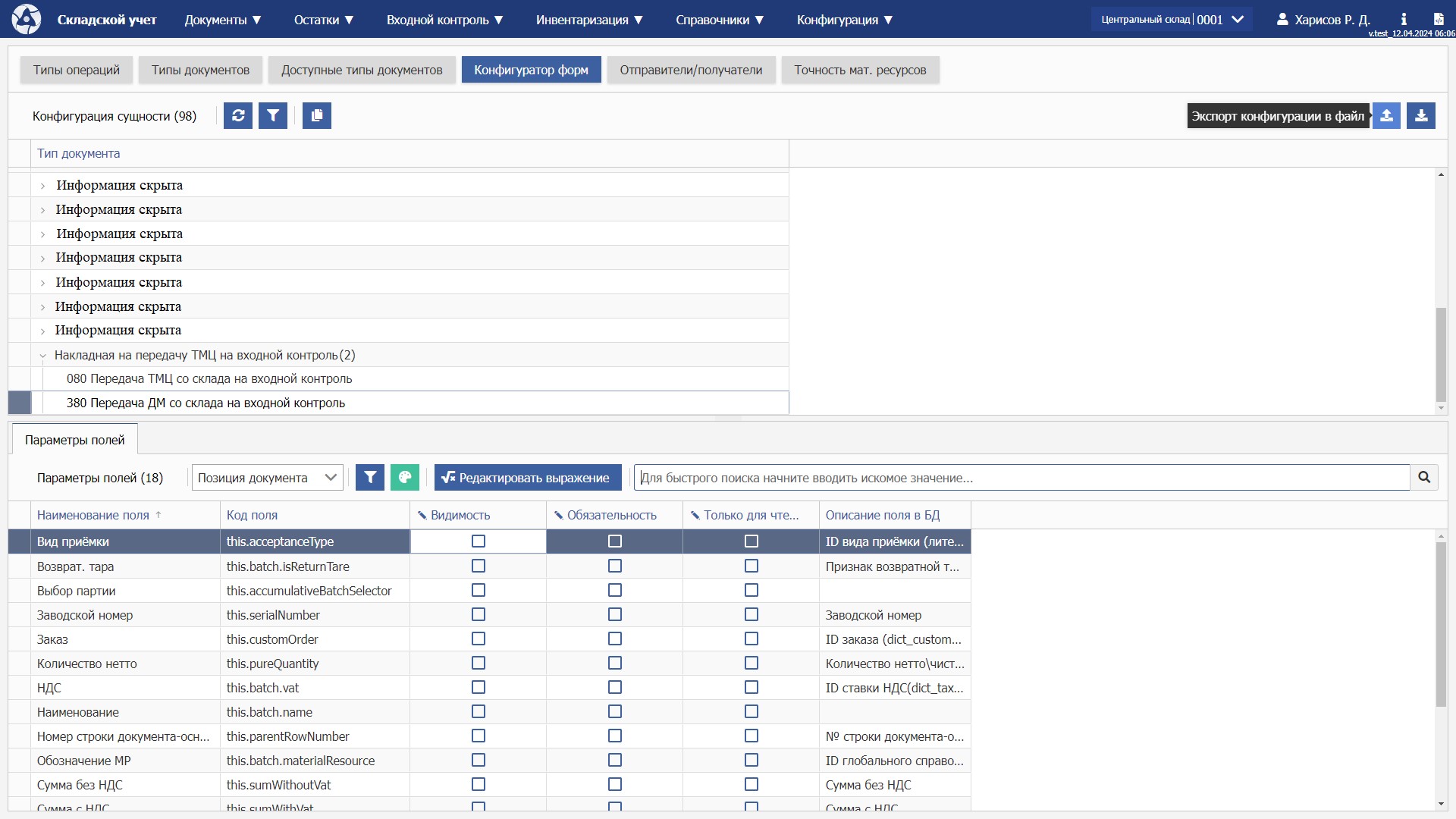


Рисунок 27 – Кнопки экспорта и импорта конфигурации

Далее в этом подразделе будет рассмотрена конфигурация ранее упомянутой накладной на входной контроль. После выбора соответствующей операции в нижней таблице будет отображаться перечень полей документов.

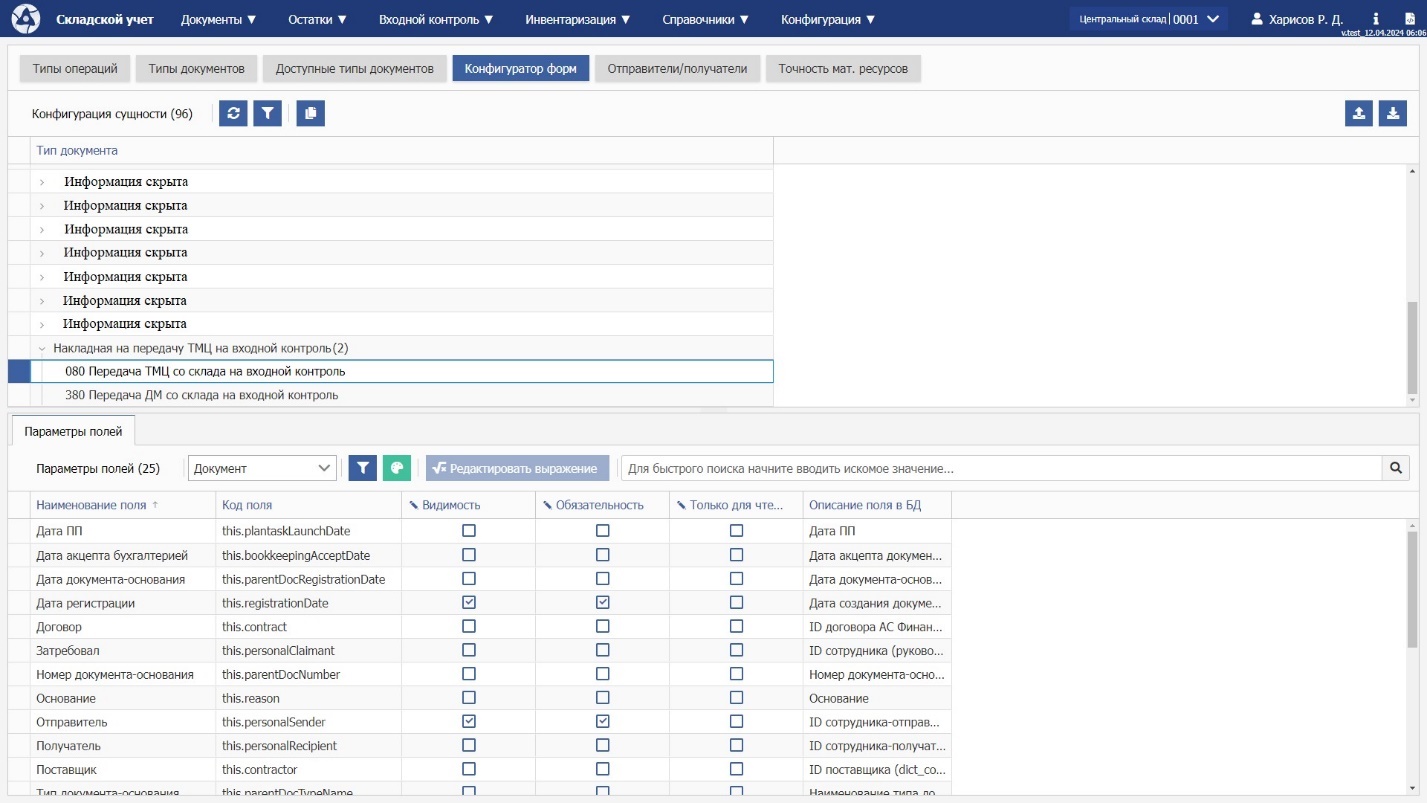


Рисунок 28 – Таблица «Параметры полей»

При помощи нажатия на квадратные поля для галочек выбирается состояние параметра поля. Например, на рисунке 28 для поля «Дата регистрации» настроены следующие параметры:

* Видимость – да;
* Обязательность – да;
* Только для чтения – нет.

Для каждого параметра поля существует возможность установки динамической настройки при помощи логических выражений на языке программирования JavaScript. После выбора параметра поля и нажатия кнопки «Редактировать выражение» открывается дополнительная вкладка, содержащая текстовый редактор для логического выражения.

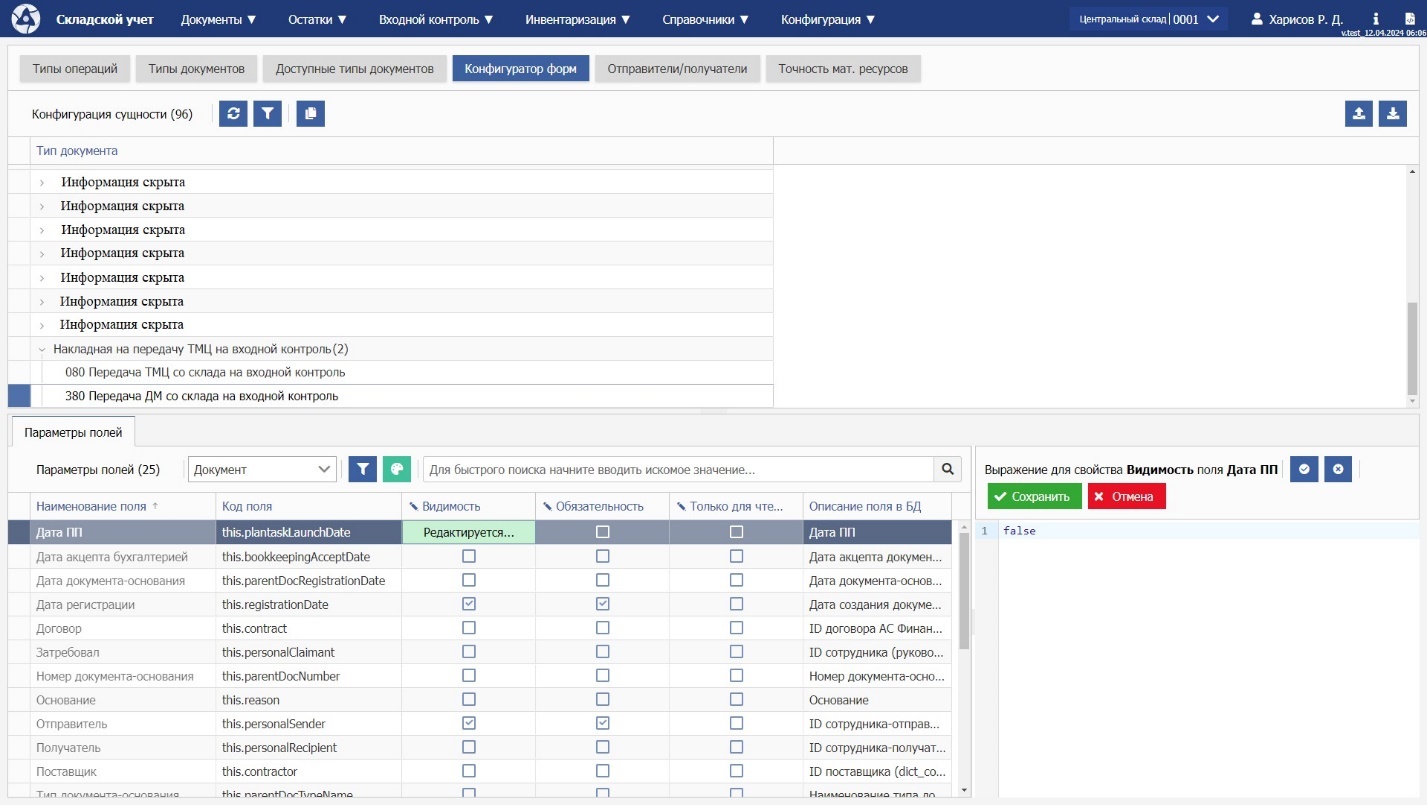


Рисунок 29 – Таблица «Параметры полей» с открытой вкладкой для редактирования выражения

Также существует возможность изменить параметры полей для позиций выбранного документа.

3.6 Конфигуратор отправителей и получателей

Конфигуратор отправителей и получателей – раздел конфигуратора, включающий в себя набор функций, позволяющий указать отправителей и получателей для каждого типа или назначения склада.

Формулировка проблемы, решаемой этим разделом конфигуратора: необходимо регулировать доступ пользователей к операциям, формирующим приходные и расходные документы, причём доступность создания документов зависит от типа или назначения склада. Также один тип работников может иметь доступ к формированию только приходных или только расходных документов.

Для решения этой проблемы был создан конфигуратор отправителей и получателей.

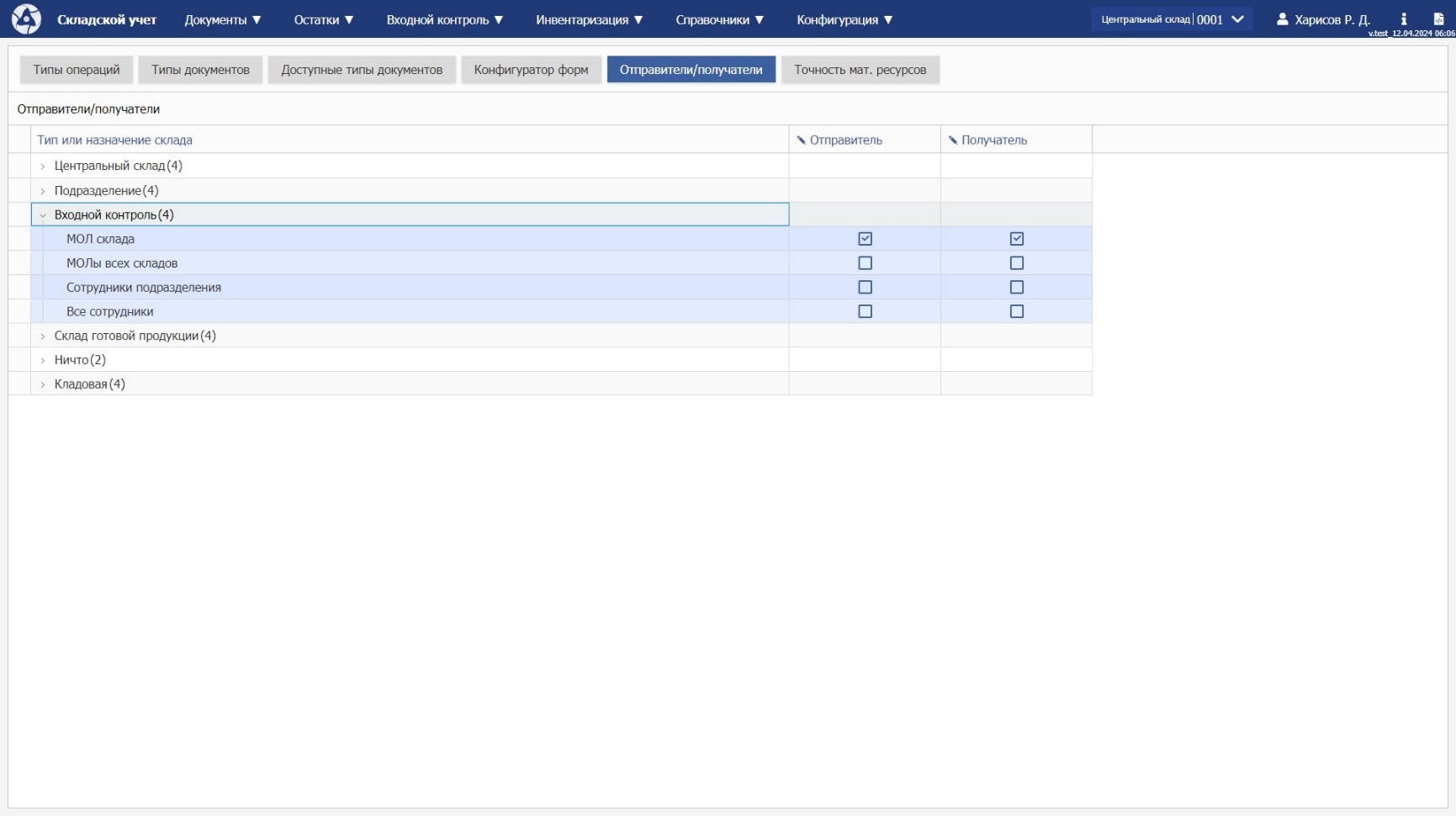


Рисунок 30 – Конфигуратор отправителей и получателей

В верхней таблице располагается таблица «Отправители/получатели», в которой указаны типы или назначения складов. При нажатии на раскрывающийся список появляются типы работников, связанные с выбранным типом склада. Для рассмотренного выше документа основным типом склада является «Входной контроль». На рисунке 30 указано, что отправителями и получателями для этого типа складов является МОЛ соответствующего склада.

В этом разделе отчёта была описана структура конфигуратора и принцип его работы. В итоге был реализованы клиентская и серверная части конфигуратора.

4 Внедрение конфигуратора в приложение

После разработки основной части конфигуратора было необходимо внедрить его в приложение. Для этого во всех местах, где от настроек конфигуратора могла зависеть бизнес-логика, были добавлены соответствующие проверки условий. На клиентской части также были изменены некоторые компоненты.

Например, при создании документа необходимо сделать запрос к базе данных (серверному приложению), чтобы узнать, какие поля формы необходимо отобразить в интерфейсе.

Этим в основном занималась оставшаяся часть команды разработчиков.

Также командой тестировщиков было проведено тестирование разделов конфигуратора и их влияния на бизнес-процесс в приложении. Все ошибки были оперативно исправлены. Для этого было необходимо проводить отладку программы.

4.1 Отладка конфигуратора

По мере создания разделов конфигуратора было множество случаев, когда необходимо было провести отладку программы.

Для отладки серверной части приложения применялись инструменты среды разработки IntelliJ IDEA, которые позволяют остановить выполнение программы в нужный момент времени и проверить значения различных переменных, используемых в данном фрагменте кода. Также при возникновении ошибки Java выводит в консоль исключение, который позволяет понять момент возникновения ошибки, краткое описание ошибки и все запущенные последовательности вызовов методов различных объектов.

Для отладки клиентской части приложения использовался отладчик, встроенный в браузер Google Chrome, позволяющий проанализировать запросы, отправленные на сервер, состояние переменных во время выполнения программы, а также HTML-разметку страницы и CSS-классы элементов разметки.

В случае сложных ситуаций приходилось обращаться к помощи наставника или сети Интернет.

После того, как мною была проверена работоспособность написанного программного кода, наставник провёл код-ревью – просмотр кода более квалифицированным программистом с целью исправления незаметных ошибок, приведения кода в более читаемый вид, а также упрощения кода. По итогам код-ревью были внесены некоторые изменения в структуру кода.

4.2 Перспективы использования конфигуратора

На момент написания отчёта конфигуратор успешно внедрён в приложение. Команда бизнес-аналитиков провела настройку приложения для складов ВНИИА. Также в текущем месяце планируется расширение перечня разделов.

Благодаря легкости расширения, конфигуратор будет активно дополняться, изменяться и корректироваться до окончания разработки приложения.

Также на момент написания отчёта командой разработчиков была создана новая технология, использующая и расширяющая конфигуратор форм. Благодаря этой технологии, документы имеющие особые поля, которые не нужно хранить в базе данных и которые должны быть заданы перед печатью, например, поля «Члены комиссии», имеют специальную форму, содержащую эти поля.

В этом разделе отчёта было описано внедрение конфигуратора в приложение, тестирование и отладка, а также перспективы его использования.

ЗАКЛЮЧЕНИЕ

В ходе работы выполнены следующие задачи:

* Изучена литература и интернет-ресурсы, необходимые для разработки клиент-серверного приложений;
* Выявлен перечень требований, отличающихся на предприятиях-заказчиках;
* Выбран способа реализации конфигурации приложения;
* Проведено разбиение конфигуратора на логически завершённые разделы;
* Реализованы разделы конфигуратора;
* Проведено внедрение конфигуратора в приложение;
* Проведены отладка и исправление ошибок, найденных командой тестирования;
* Для каждого раздела конфигуратора были выполнены следующие задачи:
  1. Были сформулированы требования к разделам конфигуратора;
  2. Были обсуждены тонкости работы разделов конфигуратора в различных ситуациях;
  3. Были спроектированы таблицы в базе данных;
  4. Были разработаны Java-классы, необходимые для работы серверного Spring приложения;
  5. Были разработаны Angular-компоненты, необходимые для работы клиентской части приложения;
  6. Была настроена связь между клиентом и сервером.