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.................................................................................................5

РАЗДЕЛ 1 ОБЗОР АНАЛОГОВ

* 1. Конфигурационные файлы
  2. Конфигурационные классы

РАЗДЕЛ 2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1 Обзор технологий, применяемых при разработке

1.2.1 Технологии серверного приложения

* + 1. Технологии клиентского приложения

2.2 Структура клиентского приложения

* + 1. Интерфейсный слой
    2. Слой логики компонентов

2.2.3 Слой отправки запросов

* 1. Структура серверного приложения

2.3.1 Слой обработки запросов

2.3.2 Слой бизнес-логики

2.3.3 Слой базы данных

РАЗДЕЛ 3 ОПИСАНИЕ КОНФИГУРАТОРА

ЗАКЛЮЧЕНИЕ..........................................................................................19

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ..................................20

ПРИЛОЖЕНИЕ А………………………………………………………21

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

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

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

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

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

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

DTO –

Reader-класс –

Service-класс –

Repository-класс –

Controller-класс –

Entity-класс –

Транзакция ­–

Аннотация –

Бизнес-логика –

QEntity –

Компонент Angular –

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящем отчете о НИР применяют следующие сокращения и обозначения: АЛФАВИТНЫЙ ПОРЯДОК

БД – база данных.

Web-приложение – приложение, встраиваемое в браузер.

ВВЕДЕНИЕ

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

«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.

Так как заказчиком является не только ВНИИА, но и множество предприятий Госкорпорации «Росатом», в каждом из которых свои перечни документов и требования к ним, то при создании приложения было необходимо решить проблему настройки приложения под каждое отдельное предприятие.

ЦЕЛЬ

ЗАДАЧИ

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.

**1 Обзор аналогов**

Аналоги конфигурации бла бла бла

**1.1 Конфигурационные файлы**

**1.2 Конфигурационные классы**

2 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.

**2.1 Обзор технологий, применяемых при разработке**

**2.1.1 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.

**2.1.2 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.

**2.2 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.

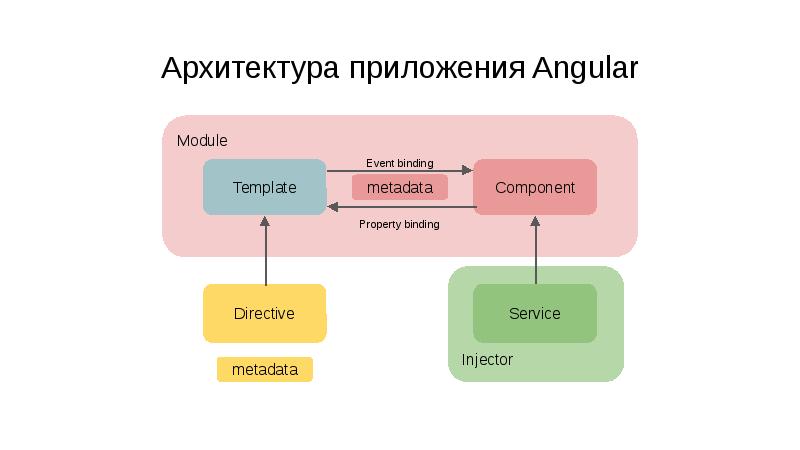


Рисунок 7 – Архитектура Angular приложения

**2.2.1 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.

**2.2.2 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.

**2.2.3 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.

**2.3 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 (DTO).

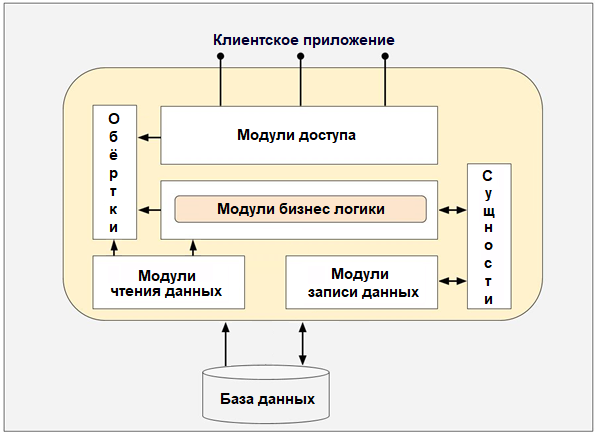


Рисунок 6 – Типичная структура серверного Spring приложения

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.

**2.3.1 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.

**2.3.2 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.

**2.3.3 Слой базы 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».

**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.

Данные 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;

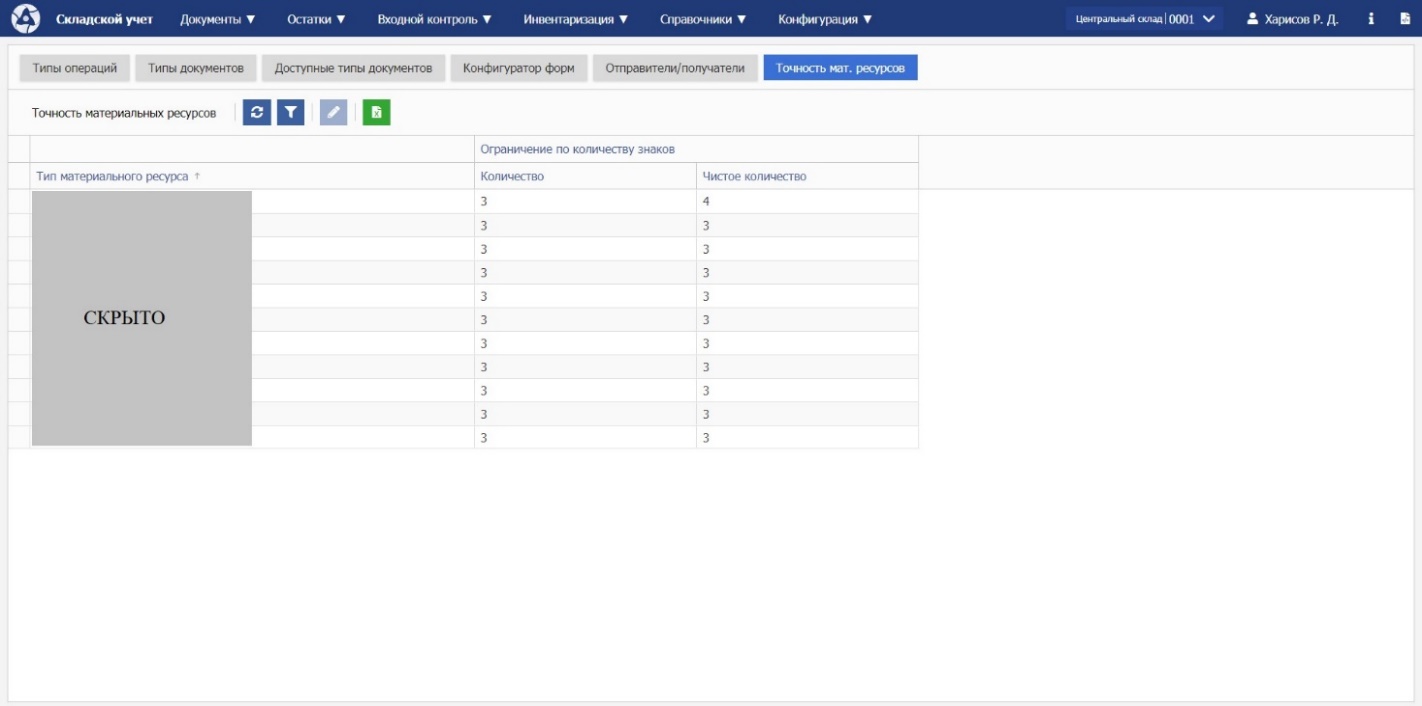


Рисунок 4 – Интерфейс конфигуратора точности материальных ресурсов

* 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;

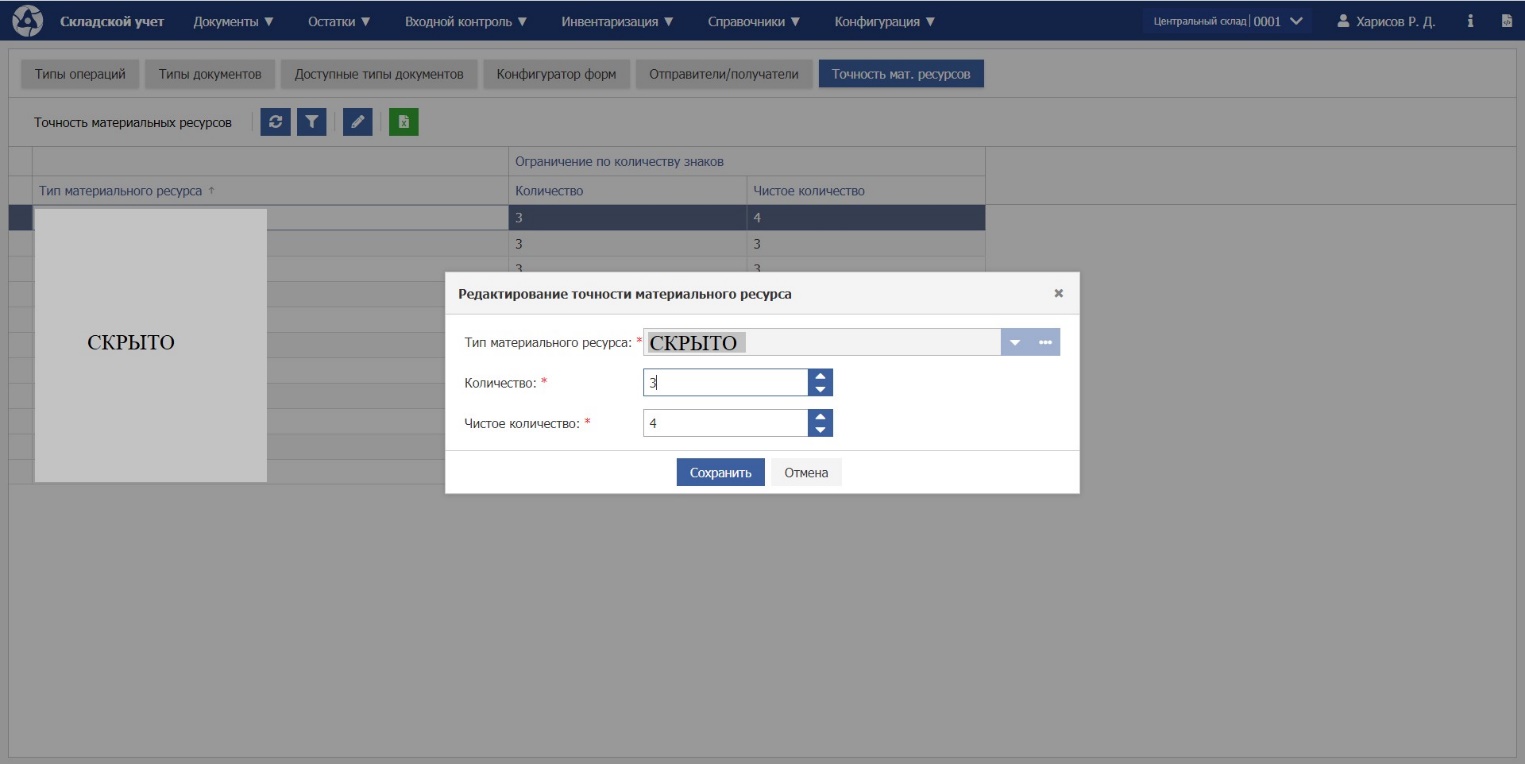


Рисунок 5 – Окно редактирования точности

* 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**

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.

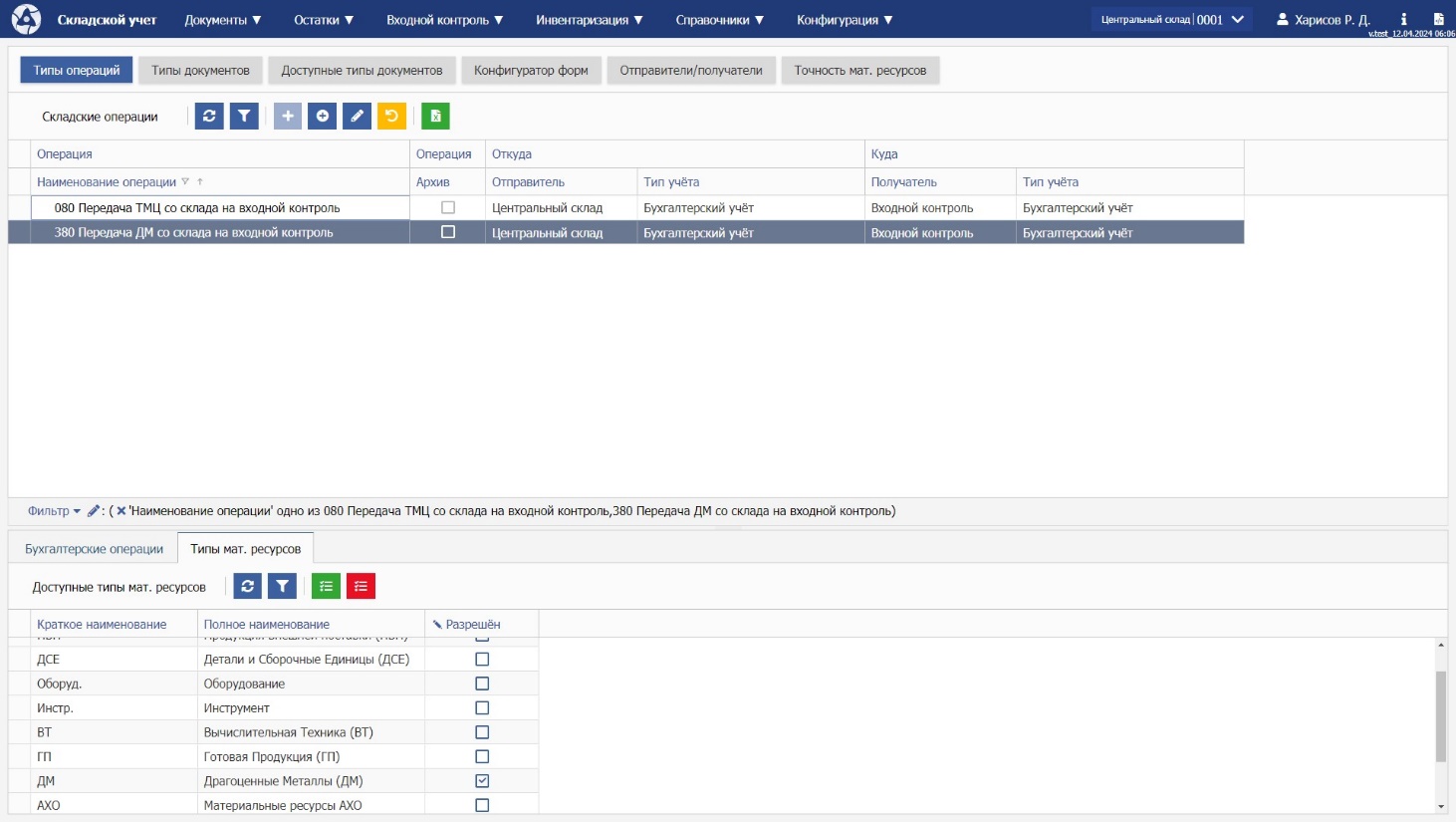


Рисунок 6 – Конфигуратор типов операций

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.

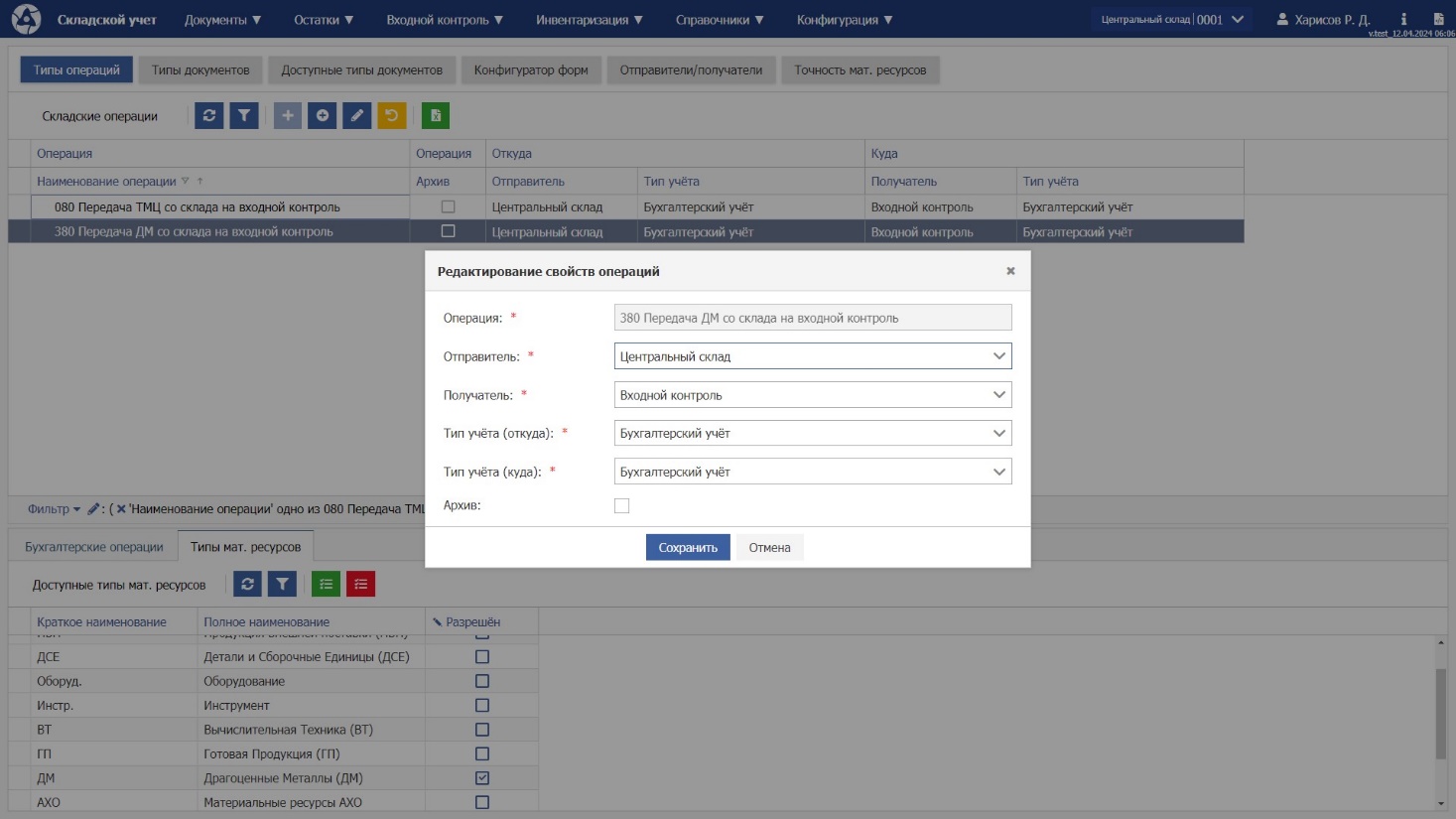


Рисунок 7 – Окно редактирования операции

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.

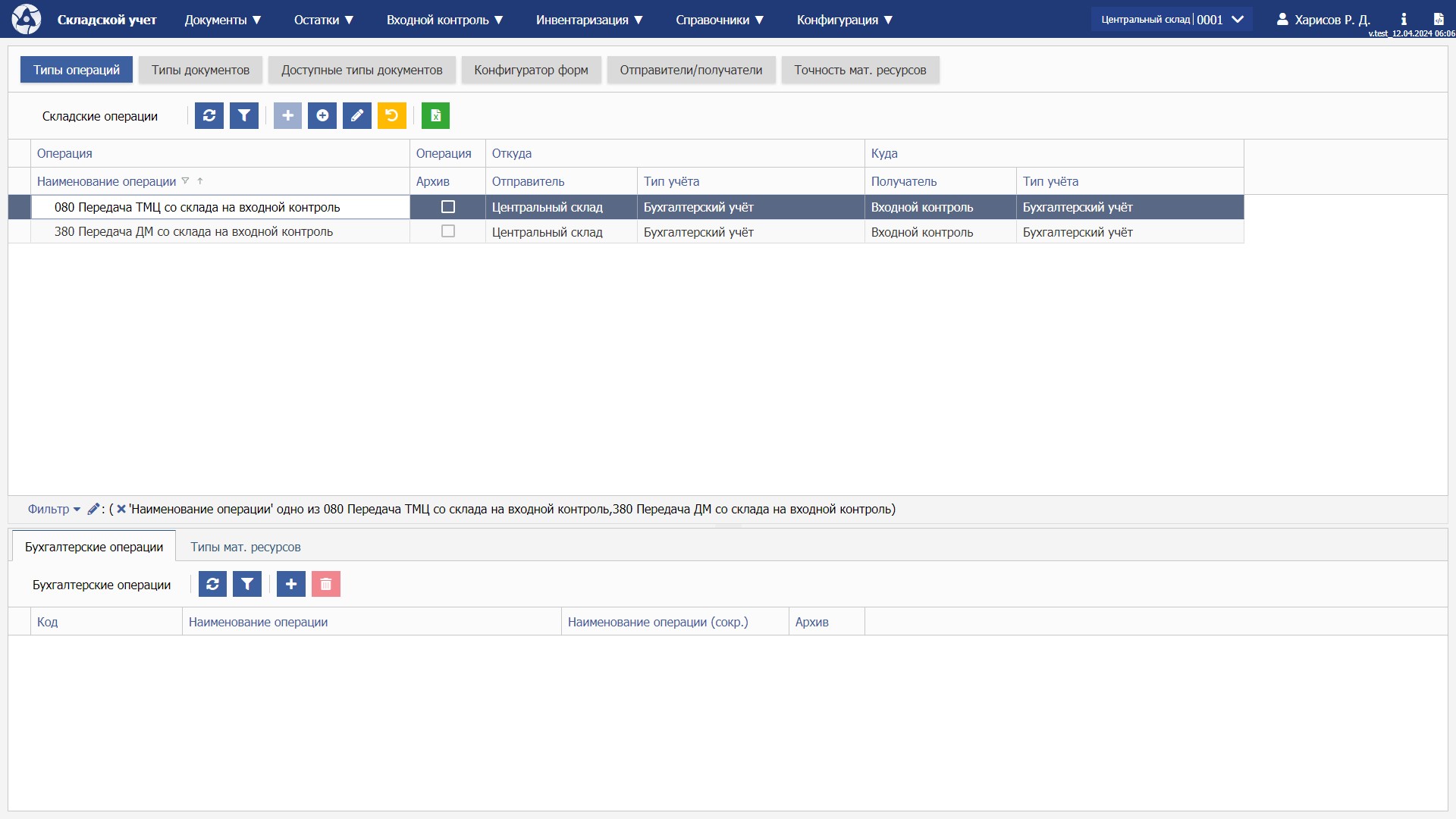


Рисунок 7 – Вкладка «Бухгалтерские операции»

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.

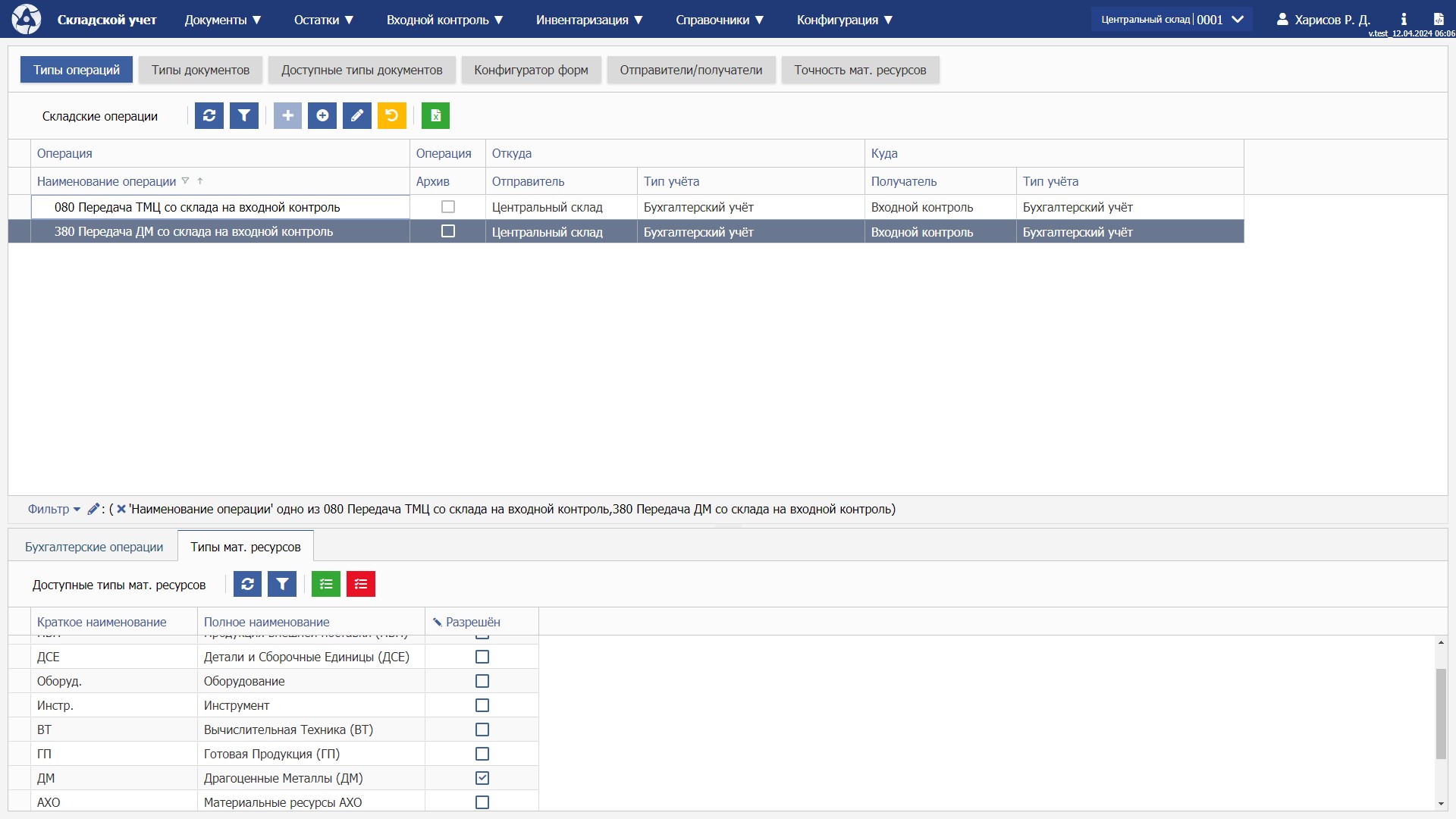


Рисунок 8 – Вкладка «Типы мат. ресурсов»

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.

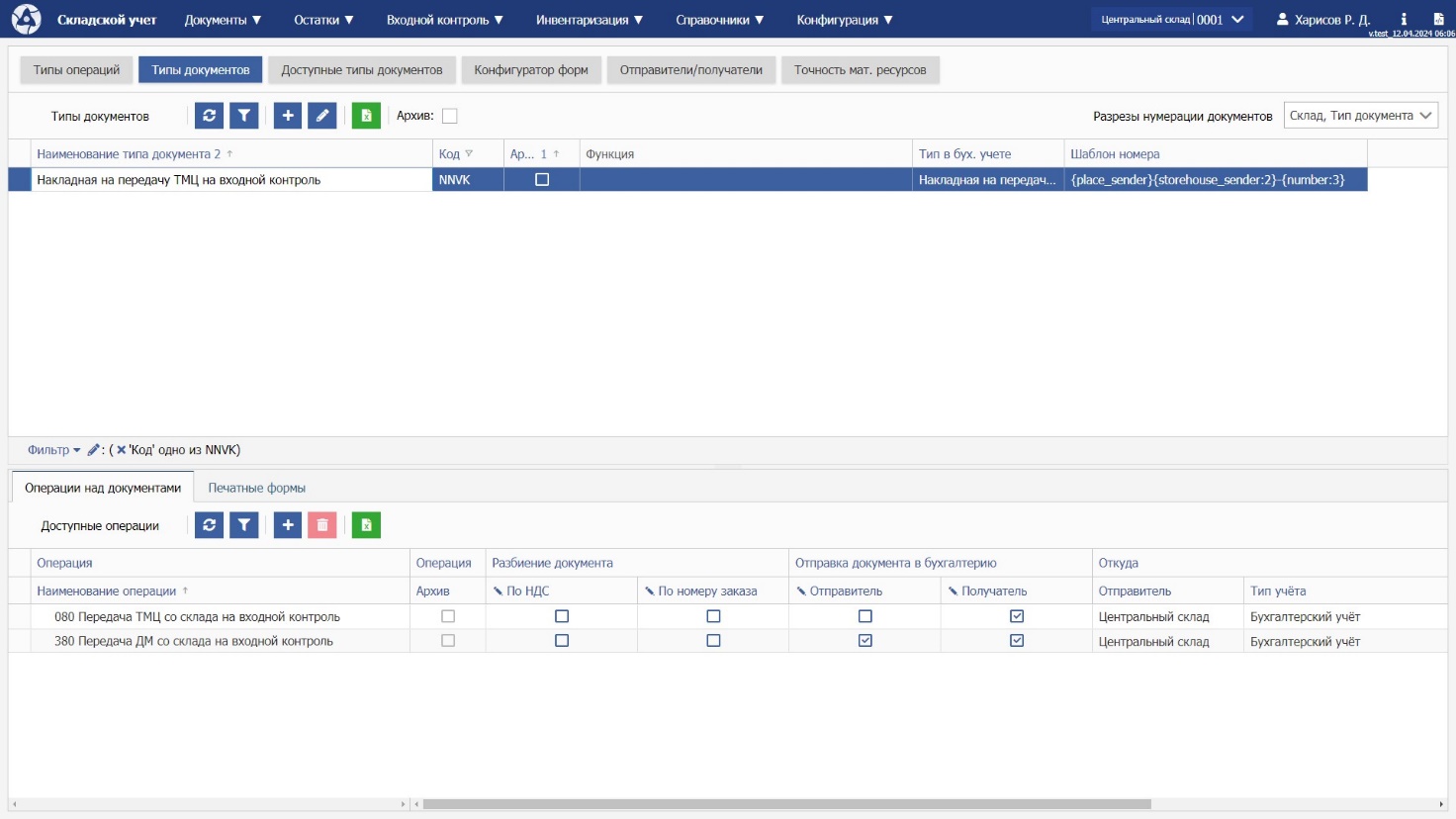


Рисунок 9 – Конфигуратор типов документов

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.

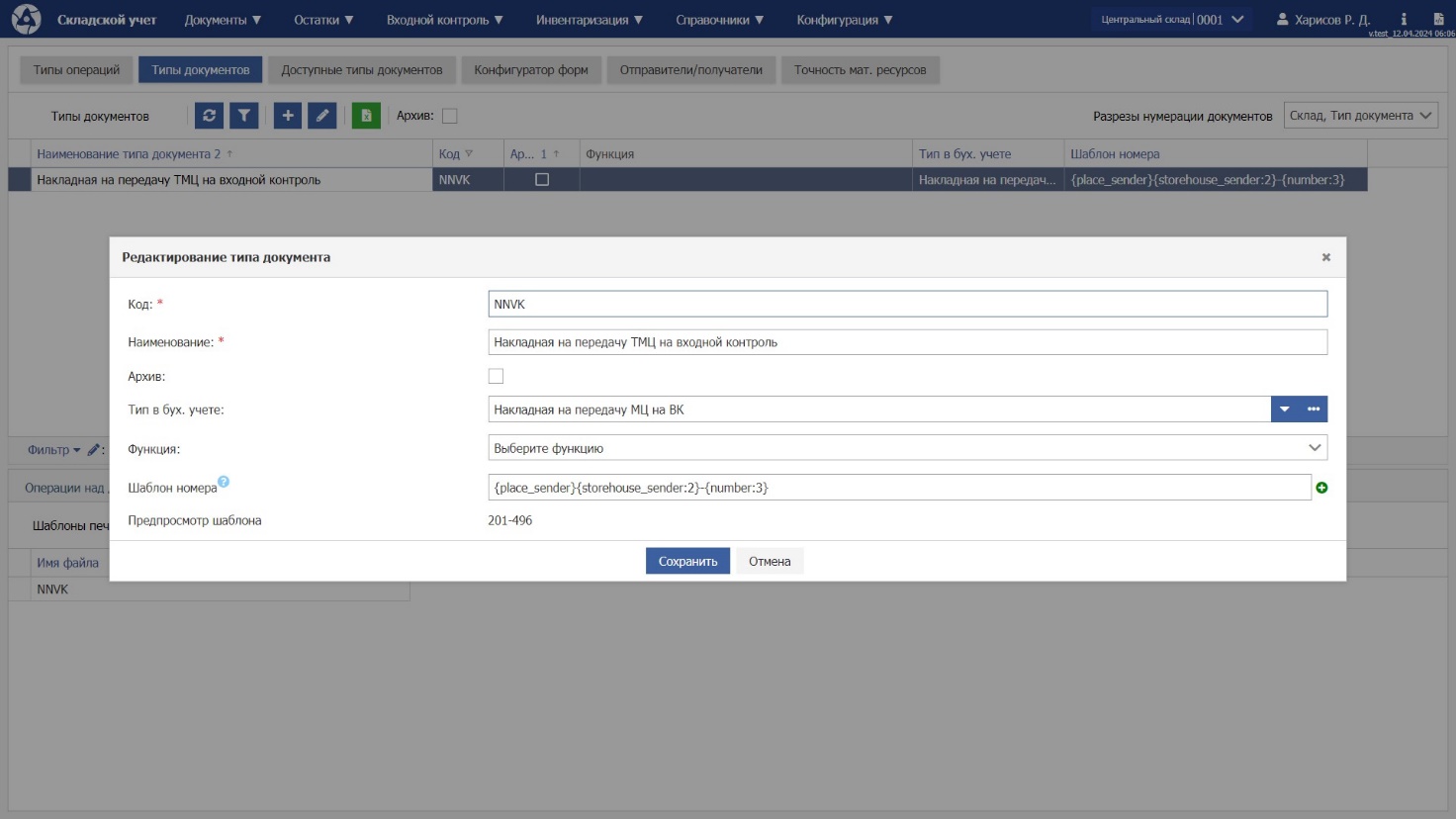


Рисунок 10 – Окно редактирования типа документа

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

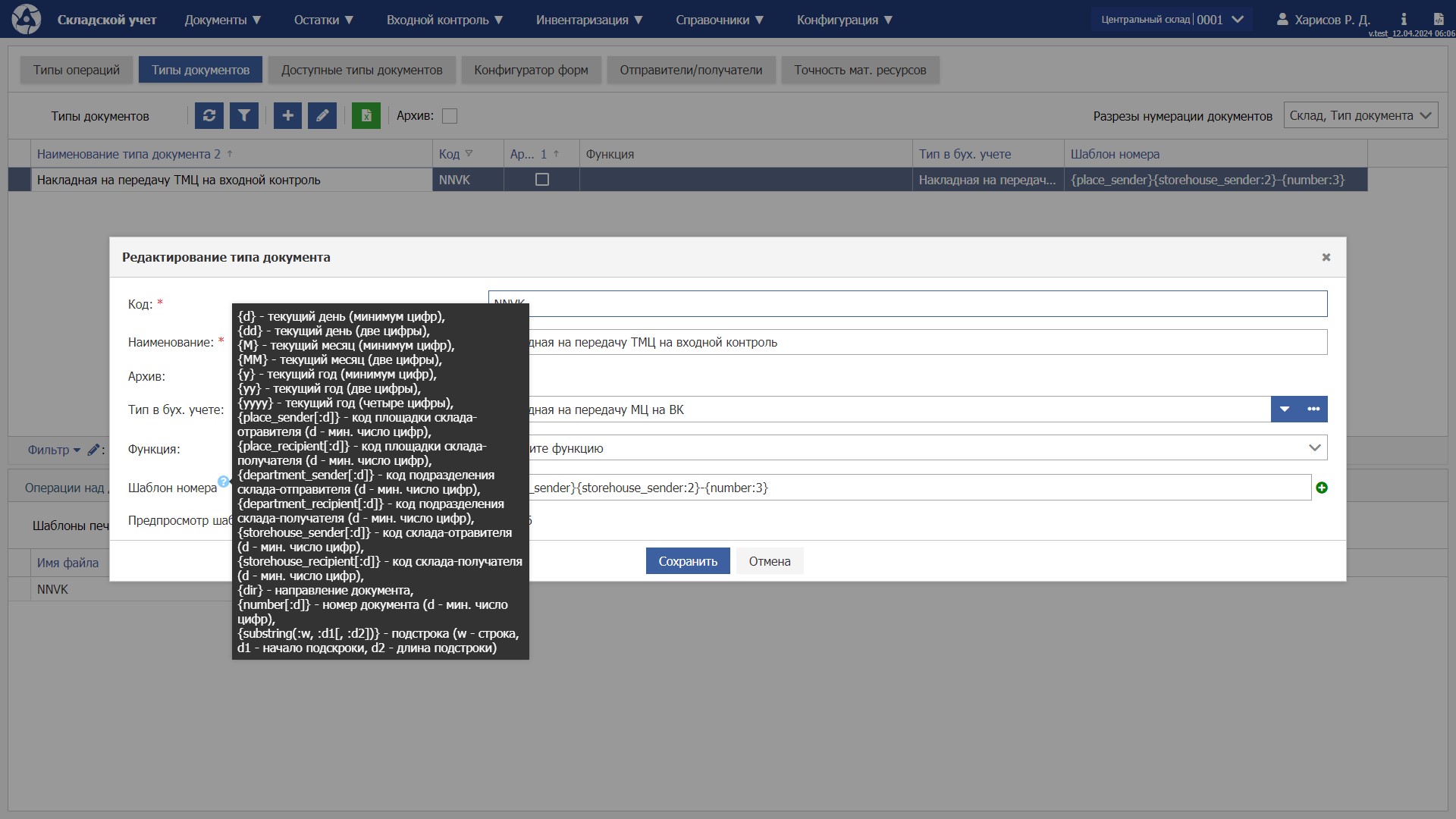


Рисунок 11 – Легенда для шаблон-строки шаблона номера документа

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

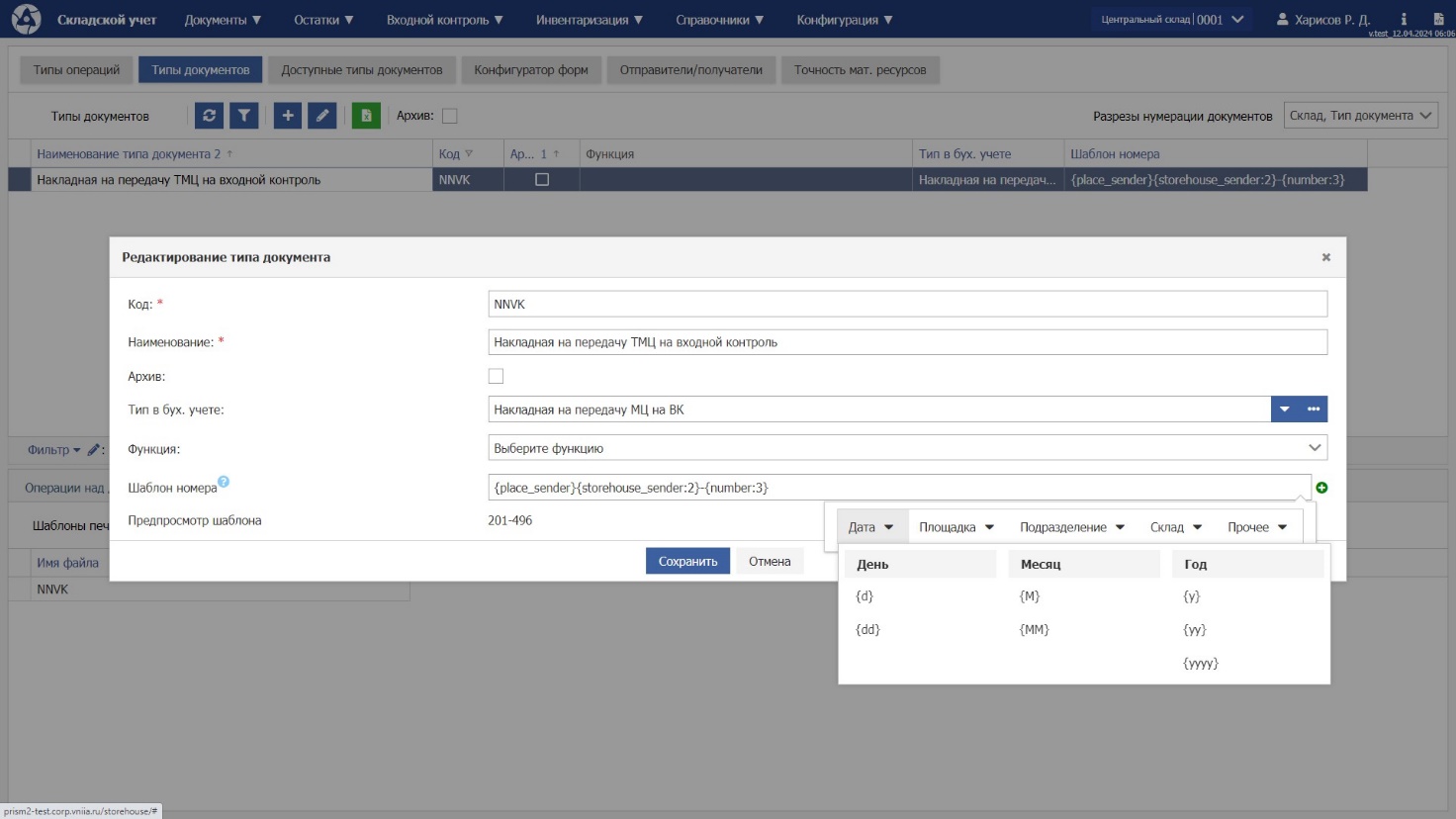


Рисунок 12 – Добавление элемента шаблон-строки шаблона номера документа

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.

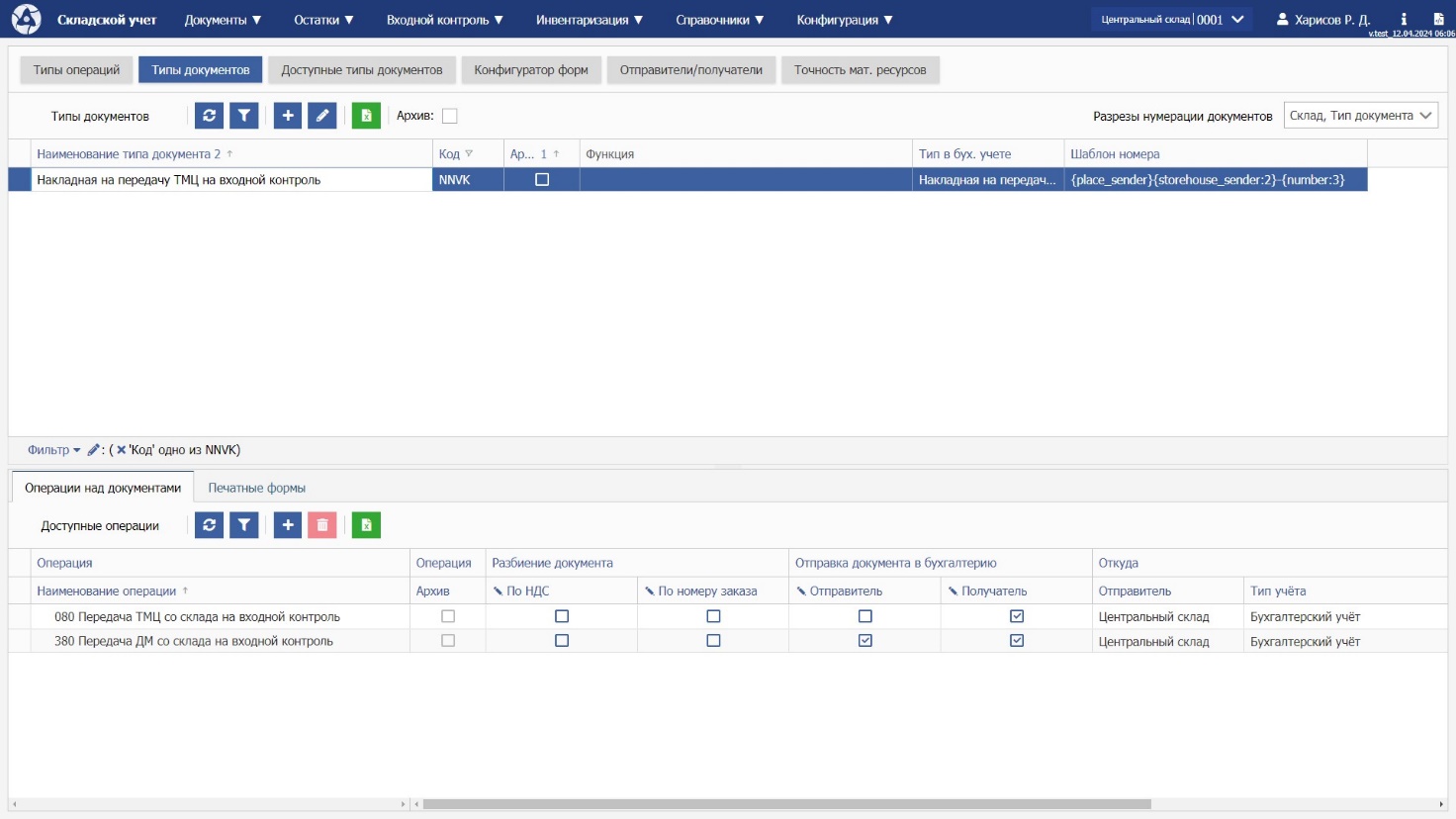


Рисунок 13 – Вкладка «Операции над документами»

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.

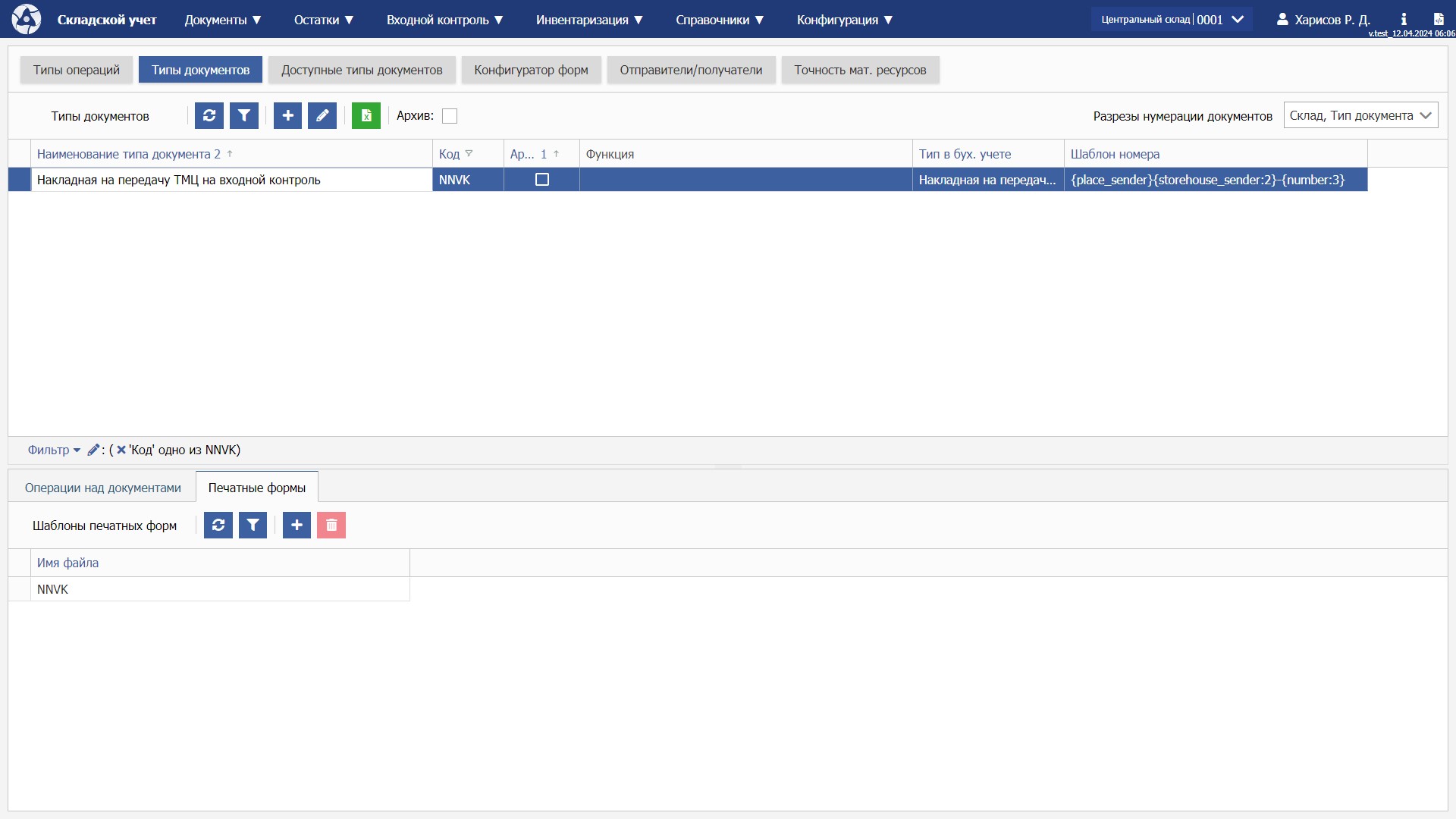


Рисунок 14 – Вкладка «Печатные формы»

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.

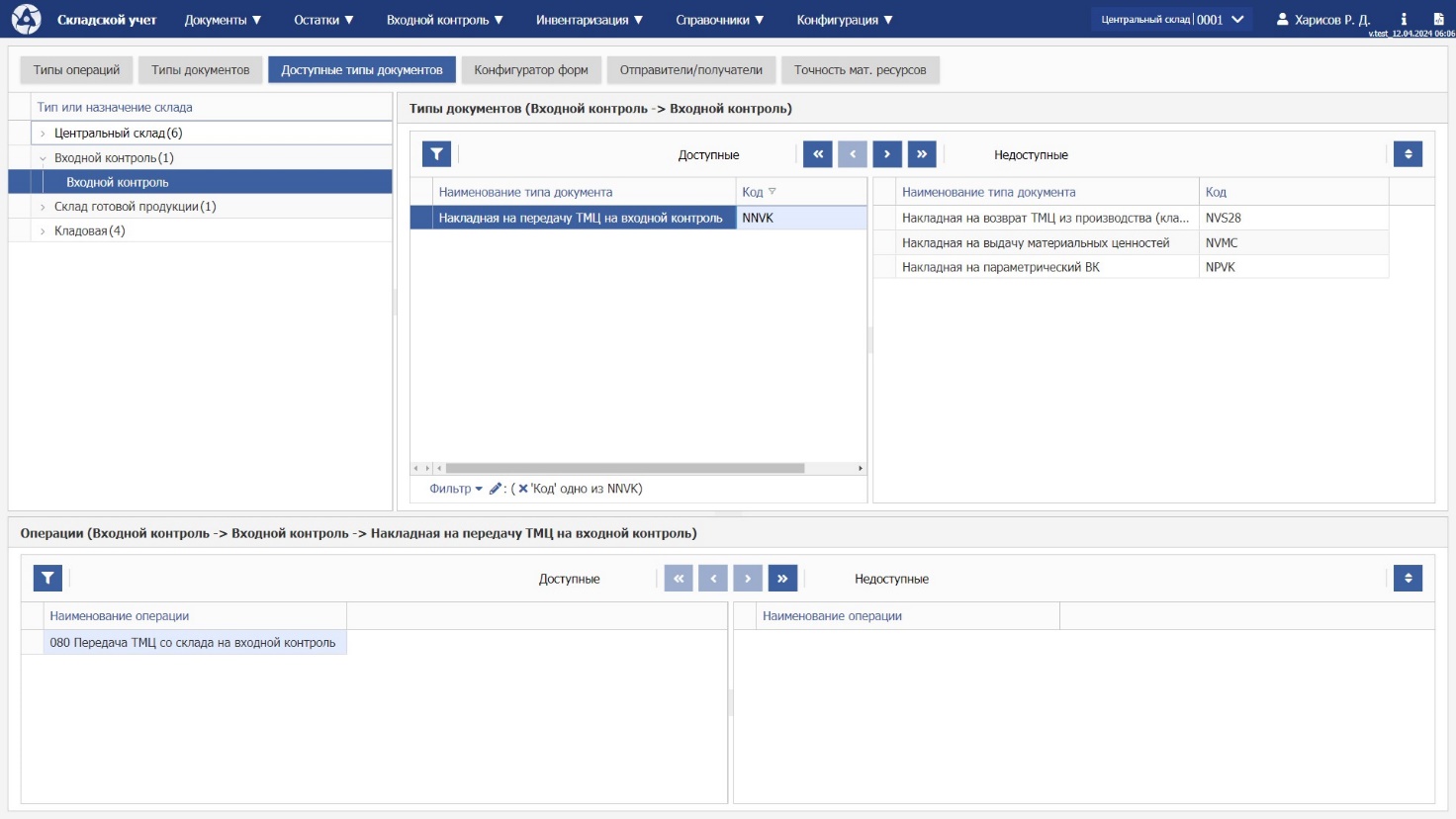


Рисунок 15 – Вкладка «Доступные типы документов»

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

Далее в разделе «Типы документов» находятся две таблицы: доступные и недоступные документы. Путем нажатия кнопок «<<», «<», «>», «>>» над таблицами можно перемещать все документы или один выбранный между ними.

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

**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. Так как приложение поставляется на предприятие, а не на конкретный склад, то необходимо обеспечить возможность настроить формы для создания электронных документов согласно каждому отдельному складу.

Для решения этой проблемы был создан конфигуратор форм.

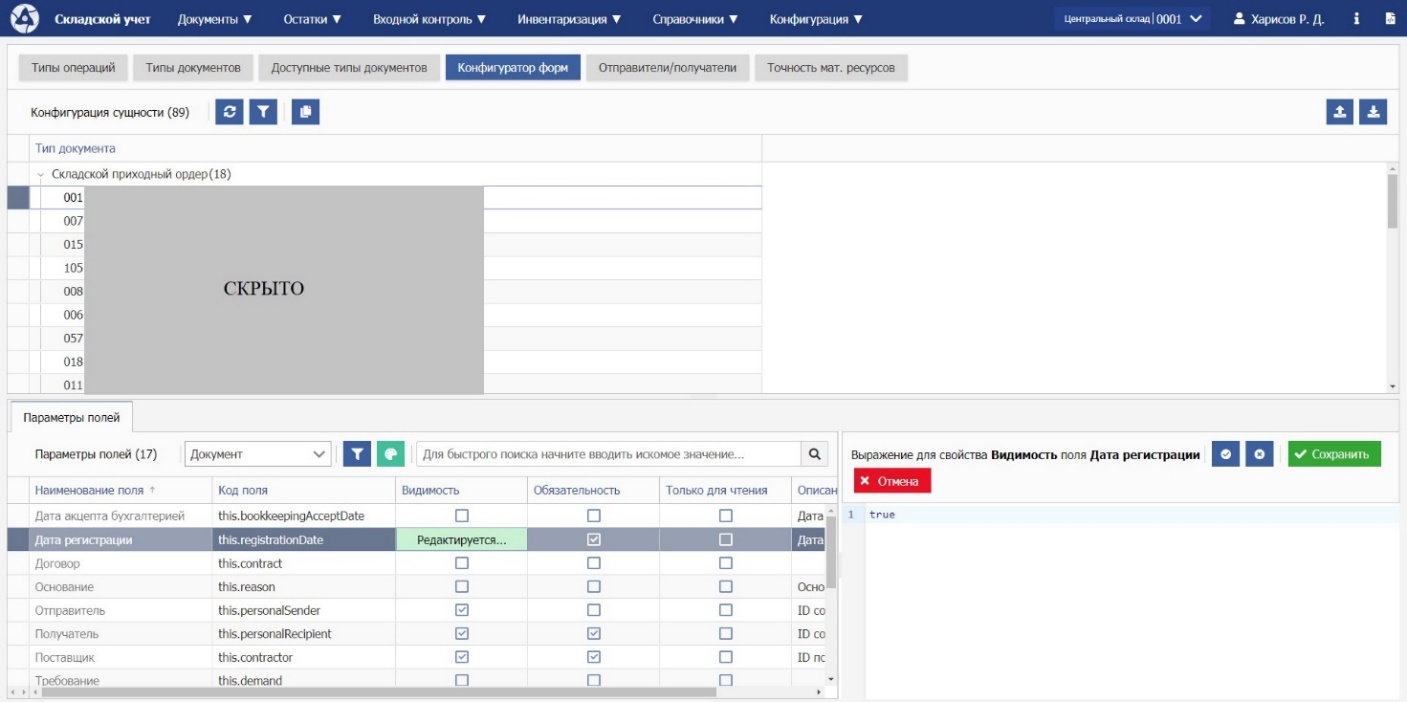


Рисунок 16 – Конфигуратор форм

В верхней таблице указаны типы документов и склады, которые могут создавать данные документы. В нижней таблице находится список полей, доступных для каждого документа (эти поля прописаны в DTO-классе DocumentDto) и описание к этим полям. Для каждого поля можно включить видимость, обязательность и возможность редактирования, причём настройку можно сделать как статической (true, false), так и динамической при помощи поля для ввода логических выражений на языке JavaScript (правый нижний угол).

В интерфейсе реализованы возможности обновления таблицы, установки фильтров, копирования, экспортирования и импортирования конфигурации для верхней таблицы и возможности фильтрации и быстрого поиска по полям.

Также в нижней таблице можно выбрать конфигурацию позиций документа. Это поля DTO-класса DocumentPositionDto, используемого для создания печатных форм документов. Функционал для позиций документа идентичен функционалу для документов

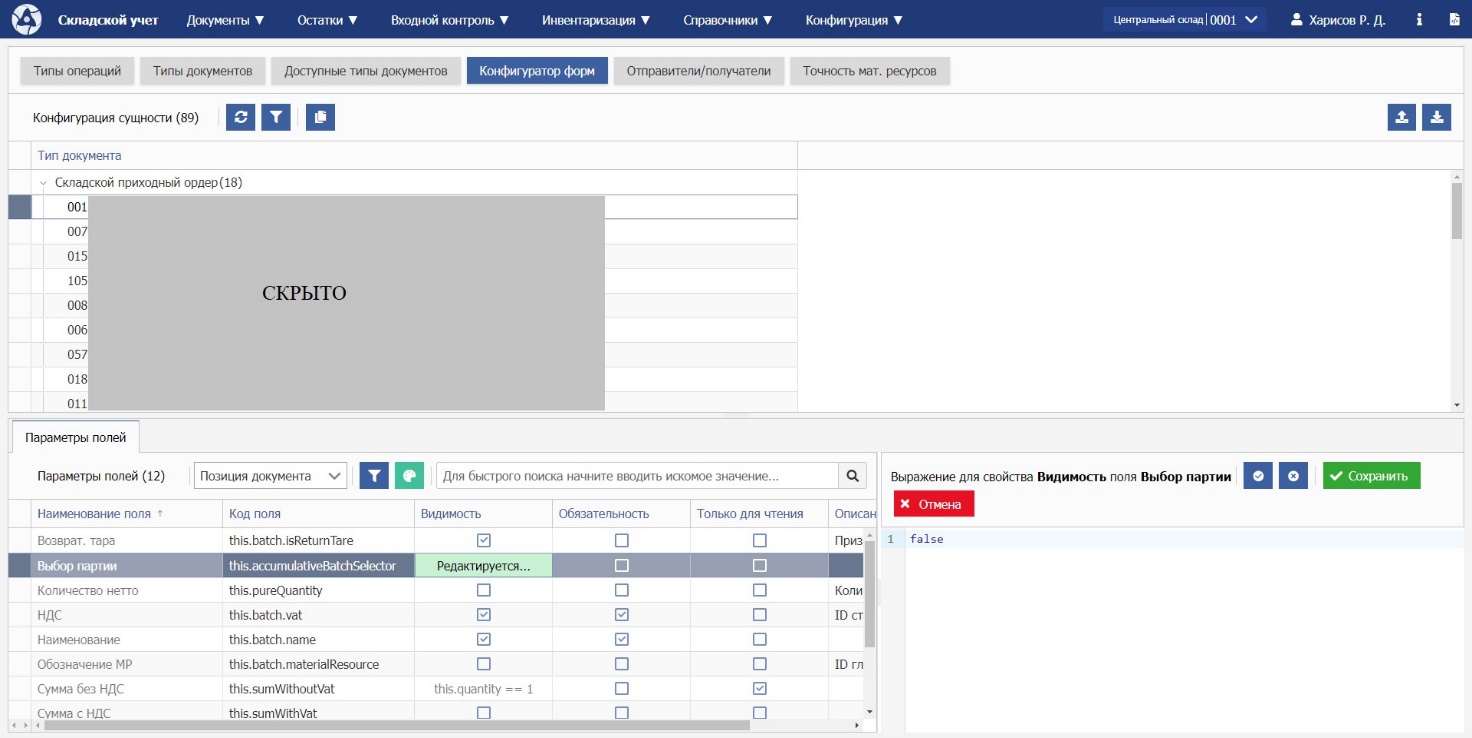


Рисунок 17 – Конфигуратор форм позиций документа

Полученный функционал приложения позволяет автоматически настраивать крупное приложение всего в несколько кликов, а самое главное – конфигуратор форм легко масштабируется при необходимости добавить новый документ.

1. **Отладка, код-ревью и тестирование**

По мере создания конфигураторов было множество случаев, когда необходимо было провести отладку программы. Для это применялись инструменты среды разработки IntelliJ IDEA, которые позволяют остановить выполнение программы в нужный момент времени и увидеть значения различных переменных, используемых в данном фрагменте кода. Также при возникновении ошибки Java выводит в консоль исключение (exception), который позволяет понять момент возникновения ошибки, краткое описание ошибки и все запущенные последовательности вызовов методов различных объектов.

После того, как мною была проверена работоспособность написанного программного кода, наставник (Тонкошкуров Алексей Дмитриевич) провёл код-ревью – просмотр кода более квалифицированным программистом с целью исправления незаметных ошибок, приведения кода в более читаемый вид, а также упрощение кода. По итогам код-ревью были внесены некоторые изменения в структуру кода, после чего конфигураторы были отправлены на тестирование.

Тестированием в подразделении занимается команда бизнес-аналитиков. Они проверяют все возможные сценарии работы программы, а также оценивают удобство, привлекательность интерфейса и соответствие результатов работы приложения и ожидаемого поведения программы. Обе работы были приняты без отправки на доработку.

После тестирования конфигураторы стали частью приложения для автоматизации складского учёта.

ЗАКЛЮЧЕНИЕ

В ходе работы выполнены следующие задачи:

* Была изучена литература и интернет-ресурсы, необходимые для разработки full-stack Spring-Angular приложений.
* Были разработаны конфигуратор точности материальных ресурсов и конфигуратор форм.
* Для каждого конфигуратора были выполнены следующие задачи:
  + Были сформулированы требования к конфигураторам
  + Были обсуждены тонкости работы конфигураторов в различных ситуациях
  + Были спроектированы таблицы в БД
  + Были разработаны Java-классы, необходимые для работы серверного Spring приложения
  + Были разработаны Angular-компоненты, необходимые для работы клиентской части приложения
  + Была настроена связь между клиентом и сервером
  + Были проведены отладка и код-ревью конфигураторов
* После тестирования конфигураторы стали частью приложения для автоматизации складского учёта