МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение   
высшего образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»**

Институт физико-технических интеллектуальных систем (ИФТИС)

Кафедра «Автоматика»

**Отчет**

**о производственной практике**

**(научно-исследовательской работе, на базовых предприятиях)**

**(8 семестр)**

на тему Автоматизация складского учёта

Студент, гр. Б20-603 Харисов Р.Д.

(подпись) (ФИО)

\*Руководитель практики инженер-программист 1 категории

(должность)

     Тонкошкуров А.Д.

(подпись) (ФИО)

М.П.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, ECTS \_\_\_\_\_\_ Дата защиты \_\_\_\_\_\_\_\_\_\_\_\_

\* если руководитель не является сотрудником НИЯУ МИФИ, его подпись заверяется печатью отдела кадров по месту работы

Москва 2024 г.

РЕФЕРАТ

Отчёт 27 с., 17 рис., 3 источника.

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

Объектом исследования является приложение для автоматизации производства и складского учёта.

Цель работы – разработка конфигуратора для приложения автоматизации складского учёта.

Область применения – множество предприятий госкорпорации «Росатом».

В процессе работы было изучено взаимодействие различный модулей приложения и был написан программный код для реализации функционала конфигуратора.

В итоге работы был создан конфигуратор, позволяющий гибко настраивать приложение автоматизации складского учёта на различных предприятиях.

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ...............................................................................3

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ.................................................4

ВВЕДЕНИЕ................................................................................................................5

ОСНОВНАЯ ЧАСТЬ................................................................................................6

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

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ...............................................20

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

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

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.

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящем отчете о НИР применяют следующие сокращения и обозначения:

БД – база данных.

Web-приложение – приложение, встраиваемое в браузер.

ВВЕДЕНИЕ

В ходе работы изучены основы программирования на Java и Typescript, изучены фреймворки Spring и Angular, а также дополнительные технологии, обеспечивающие стабильную работу приложений.

Предметом разработки стал конфигуратор складского учёта, реализуемый в рамках проекта «СКЛАД-2» отделом разработки «Призма» ФГУП «ВНИИА им. Духова».

ОСНОВНАЯ ЧАСТЬ

1. **Описание стека технологий, используемых для разработки приложения**

Для разработки серверной части приложения использовались следующие технологии:

* Java 17 – язык программирования строгой типизации, позволяющий обеспечить стабильность работы приложений. Зачастую применяется для автоматизации бизнеса.
* Spring Boot – фреймворк для Java, позволяющий быстро запустить приложение и обеспечить гибкую и простую конфигурацию приложения. Включает в себя множество технологий для работы с базой данных, обработки HTTP-запросов, обеспечения безопасности и ролевого доступа, а также валидации данных.
* 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.
* 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.
* QueryDSL – фреймворк для Java, позволяющий упростить обмен данными между приложением и базами данных при помощи создания запросов к БД на языке Java.
* Flyway – фреймворк для Java, позволяющий управлять структурой баз данных из Java приложения путём создания миграций и защищающий приложение от внешних изменений баз данных.
* HTTP-запросы с технологией REST – HTTP-запросы, состоящие из JSON-объектов и позволяющие запрашивать и передавать наборы данных между клиентской и серверной частями приложения.

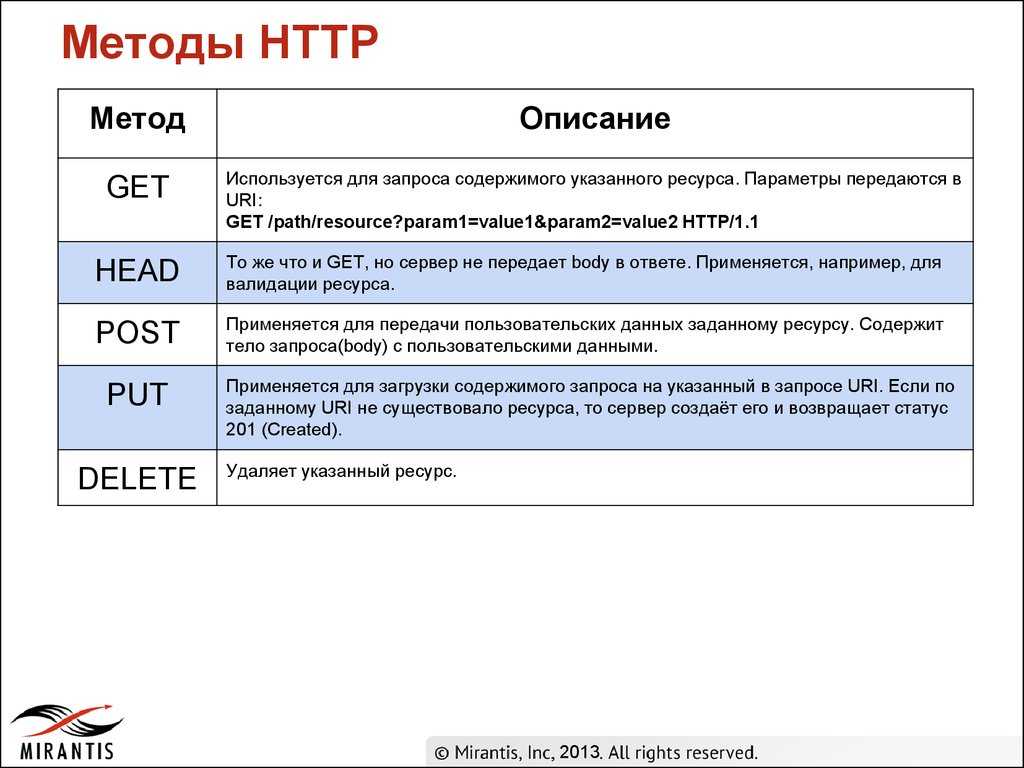


Рисунок 1 – Методы HTTP-запросов

* TypeScript – язык программирования строгой типизации, основанный на языке JavaScript и позволяющий писать безопасный код для клиентской части приложения.
* Angular – фреймворк для TypeScript, позволяющий разделить клиентское приложение на компоненты, имеющие собственный интерфейс. Благодаря Angular компоненты могут обмениваться информацией.

1. **Описание структуры серверной части приложения**

Серверная часть приложения используется для обмена данными с базой данных и клиентской частью, а также для обработки запросов и выдачи сообщений в случае ошибки.

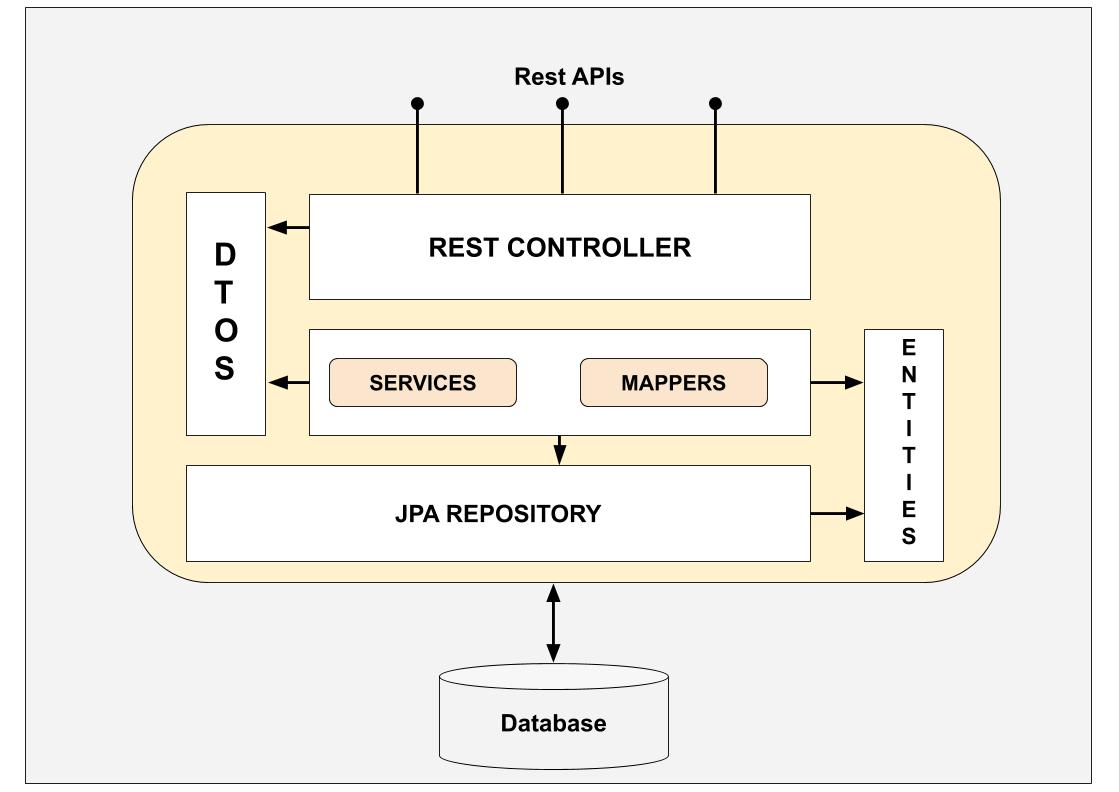


Рисунок 2 – Типичная структура серверного Spring приложения

Она принимает HTTP-запрос по REST от клиентской части. Spring Boot автоматически определяет от кого поступил данный запрос. Затем по URL-адресу запрос определяется REST-контроллер, который будет обрабатывать запрос.

В случае GET-запросов (запросов на получение информации) контроллер запрашивает данные у Reader-классов, которые обращаются к базе данных, в основном с помощью QueryDSL, и получают необходимые данные. Затем эти данные отправляются контроллеру, который собирает JSON файл с данными и отсылает их клиентской части приложения.

В случае POST-, PUT-, DELETE-запросов (запросов на изменение данных) контроллер отправляет запрос Service-классам, которые обрабатывают данные в соответствие с бизнес-логикой и выдают команду Repository-классам, которые управляют базой данных при помощи Spring Data (Hibernate).

Обмен данными между сервисами и БД происходит с помощью Entity-объектами (сущностями), между контроллерами и ридерами – с помощью DTO – классами обёртками над сущностями. DTO также используется для обменами данными с клиентскими приложениями и другими приложениями. Они позволяют стандартизировать структуры данных.

1. **Описание структуры клиентского приложения.**

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

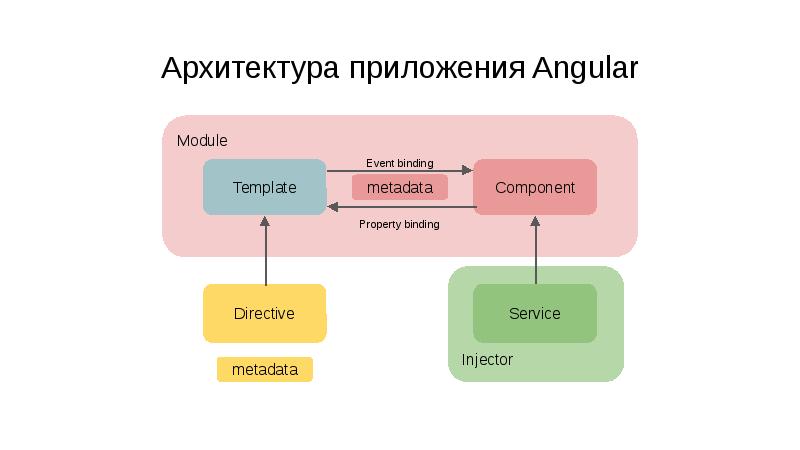


Рисунок 3 – Архитектура Angular приложения

В клиенте также есть сервисы (Services), однако они занимаются отправкой запросов. Каждый сервис клиента связан с контроллером сервера.

Angular приложение разбито на множество компонентов (Components), каждый из которых отвечает за свою страницу и имеет собственный интерфейс (Template). За выбор текущей страницы отвечает класс Router, встроенный в фреймворк Angular.

Каждый компонент состоит из исполняемого файл TypeScript, HTML разметки его страницы (интерфейса) и файла CSS, определяющего стили HTML страницы.

Именно компоненты запрашивают через сервисы данные с сервера. Это происходит при нажатии соответствующих кнопок в интерфейсе или при переходе на конкретную страницу.

1. **Описание конфигуратора**

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

Конфигуратор складского учёта – специальная вкладка, 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.

В рамках конфигуратора мною были реализованы следующие его части:

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

1. **Конфигуратор точности материальных ресурсов**

Формулировка проблемы: на складах каждого предприятия могут храниться различные материальные ресурсы (драгоценные металлы, химические вещества, оборудование и т.д.). При составлении различных документов (этим занимается приложение) для каждого материального ресурса указывается его количество. Это количество должно иметь некоторое количество знаков после запятой (точность), 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 предприятия.

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

К сожалению, по причине конфиденциальности предприятия, запрещено выкладывать исходный код приложения, поэтому далее будет кратко описан процесс разработки конфигуратора.

Типы материальных ресурсов хранились в отдельной таблице в базе данных (в так называемом справочнике, 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 ресурса и для чистого ресурса.

После проектирования таблицы в БД были созданы следующие классы, о функционале которых было рассказано ранее:

* MaterialResourceTypeQuantityPrecision – Entity-класс
* MaterialResourceTypeQuantityPrecisionDto – DTO-класс
* MaterialResourceTypeQuantityPrecisionRepository – Repository-класс
* MaterialResourceTypeQuantityPrecisionReader – Reader-класс
* MaterialResourceTypeQuantityPrecisionController – RestController-класс

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 (список) всех материальных ресурсов доступных на предприятии и точность каждого из них.

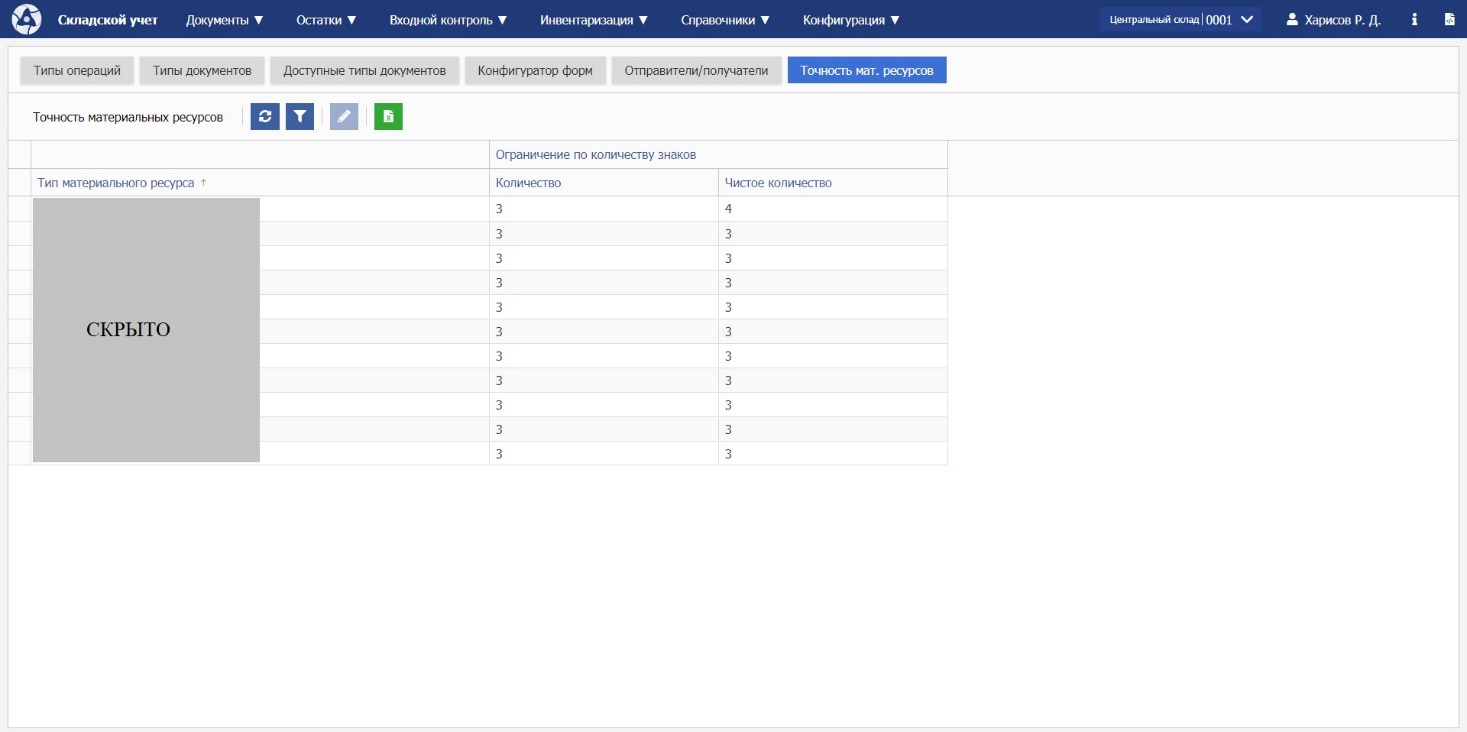


Рисунок 4 – Интерфейс конфигуратора (конфиденциальная информация скрыта)

* 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

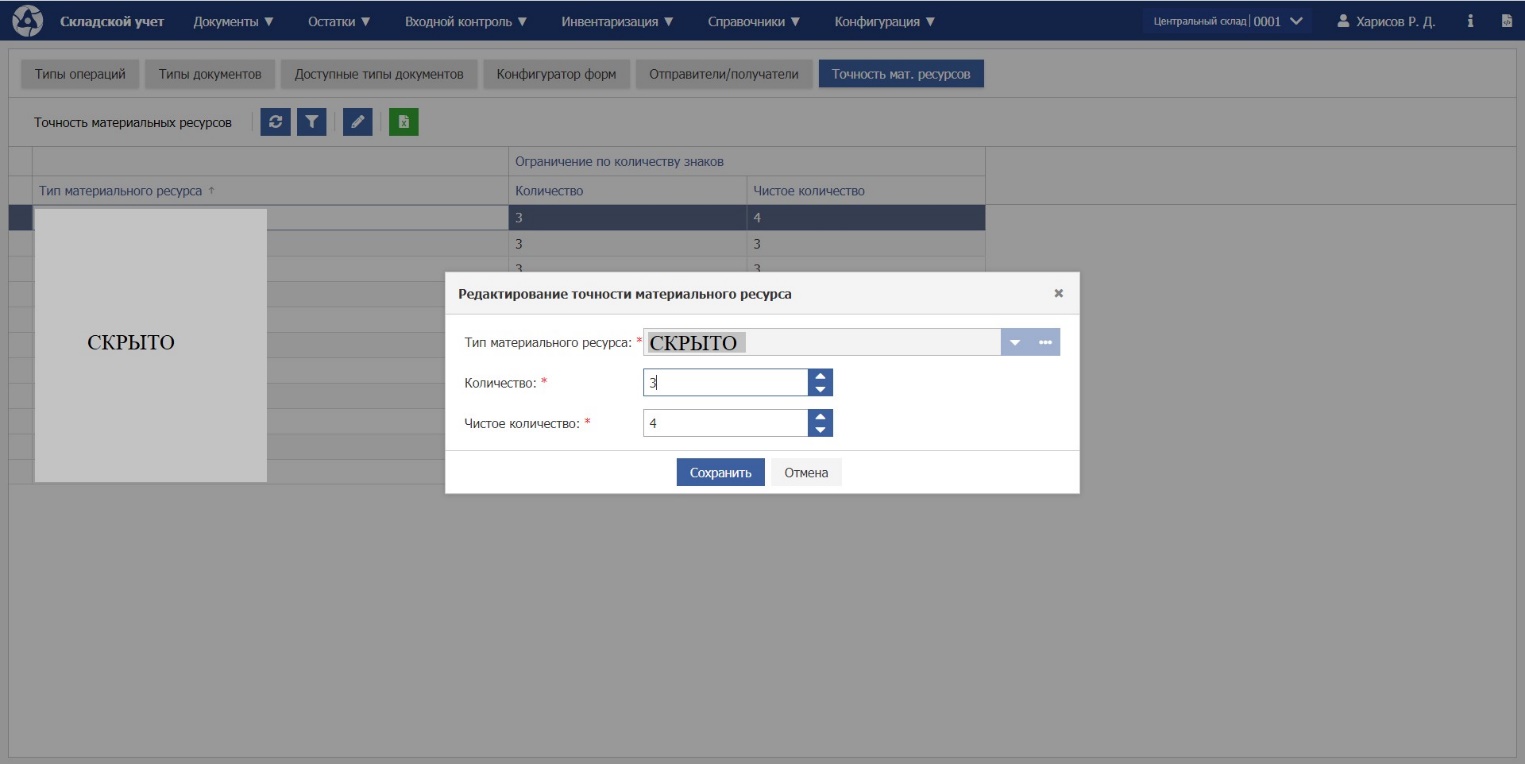


Рисунок 5 – Окно редактирования точности

* 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.

1. **Конфигуратор типов операций**

Формулировка проблемы: существует перечень операций, которые можно проводить с материальными ценностями склада. 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.

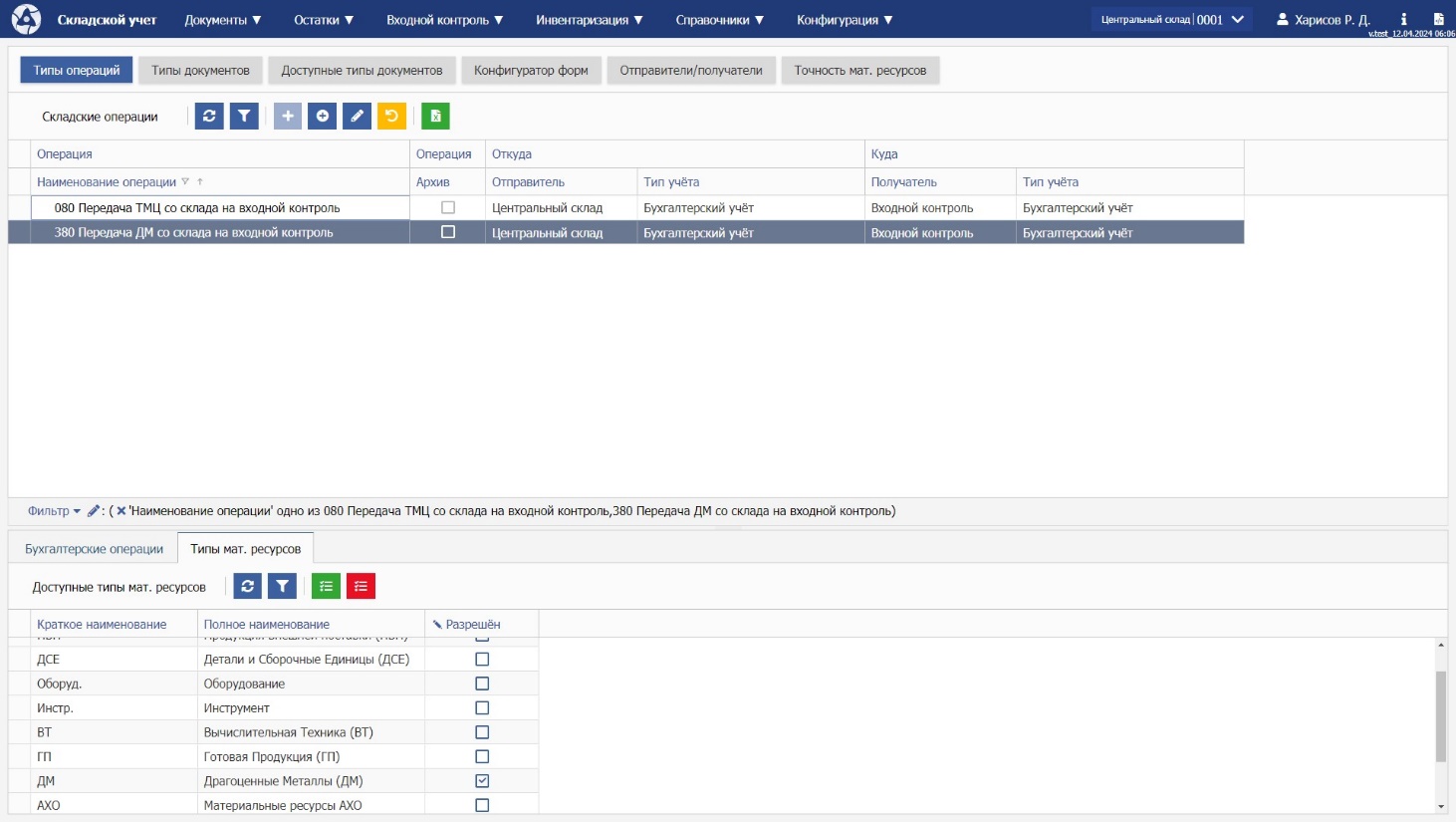


Рисунок 6 – Конфигуратор типов операций

При редактировании операции есть возможность изменять все поля сущности, кроме названия. Также есть возможность архивации устаревших операций.

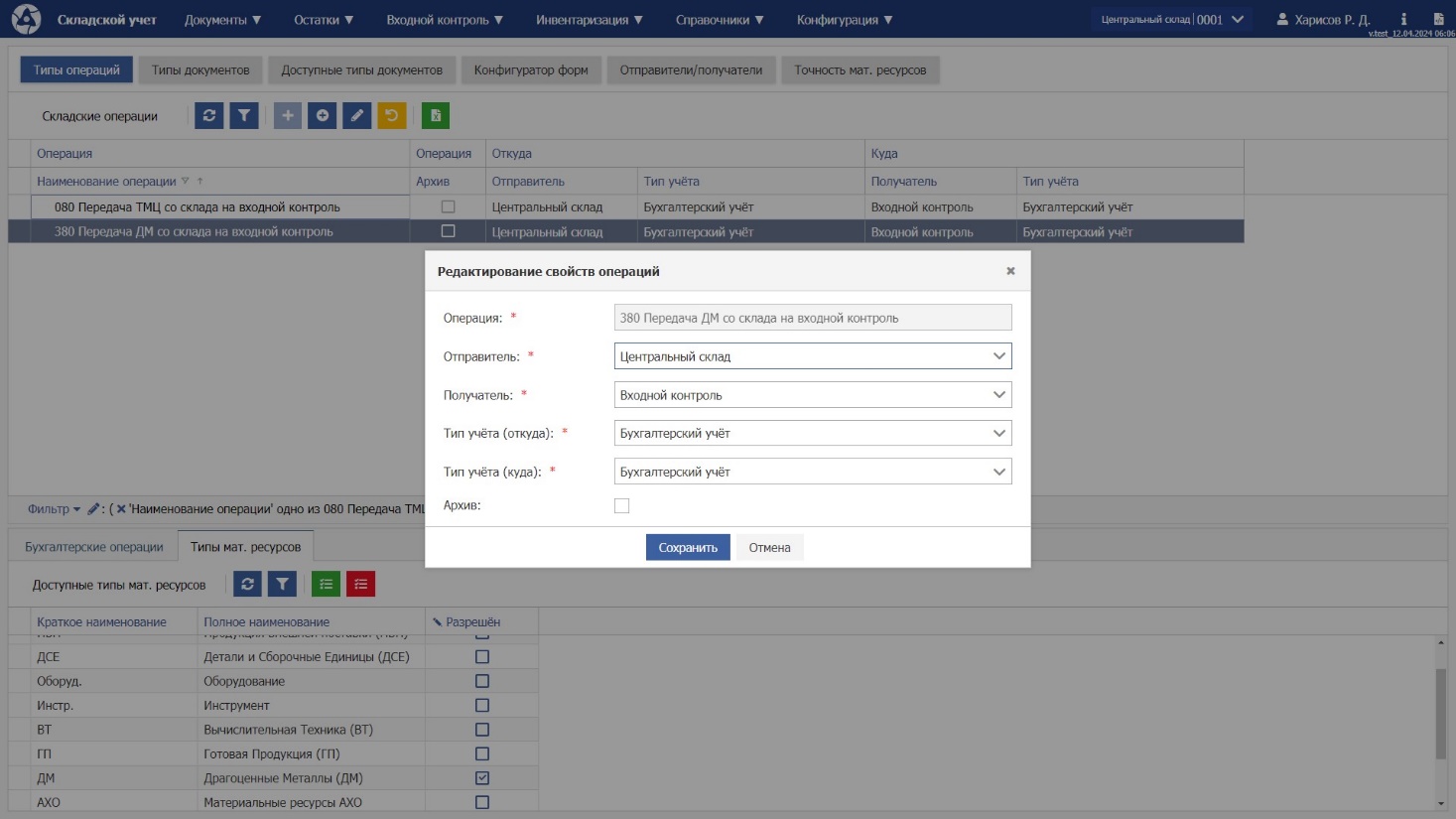


Рисунок 7 – Окно редактирования операции

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

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

Первая вкладка позволяет связать операцию склада с операцией финансового сервиса «АС Финансы», входящего в состав проекта «Призма 2.0». В ней находится список бухгалтерских операций, кнопки создания и удаления, а также стандартные для таблиц кнопки.

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.

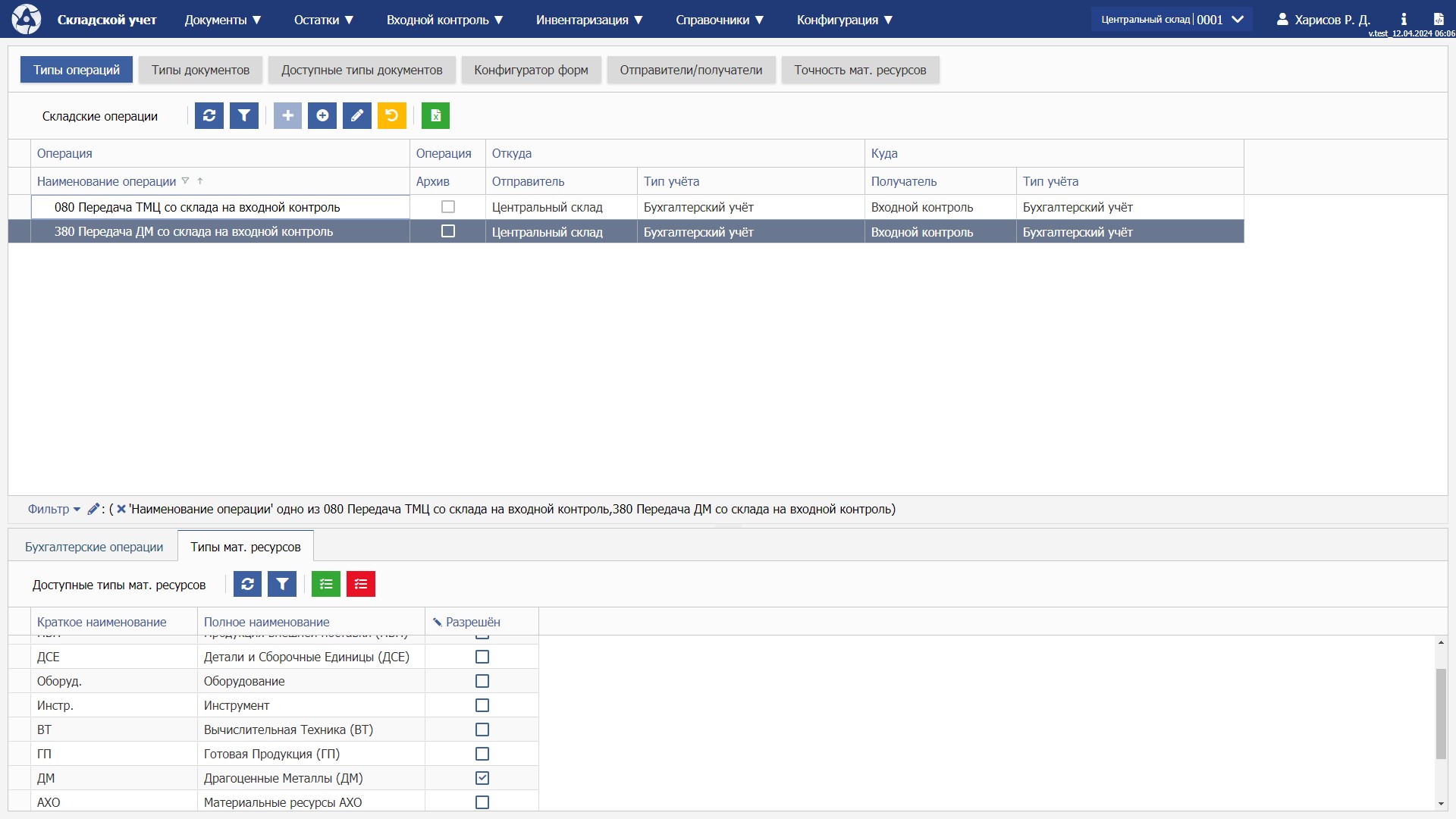


Рисунок 8 – Вкладка «Типы мат. ресурсов»

Список материальных ресурсов взят из справочника, который можно в любой момент обновить.

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

1. **Конфигуратор типов документов**

Формулировка проблемы: после создания типа операции, необходимо создать тип документа, который будет отвечать данной операции. У одного документа, может быть, несколько различных операций, а также печатные формы.

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.

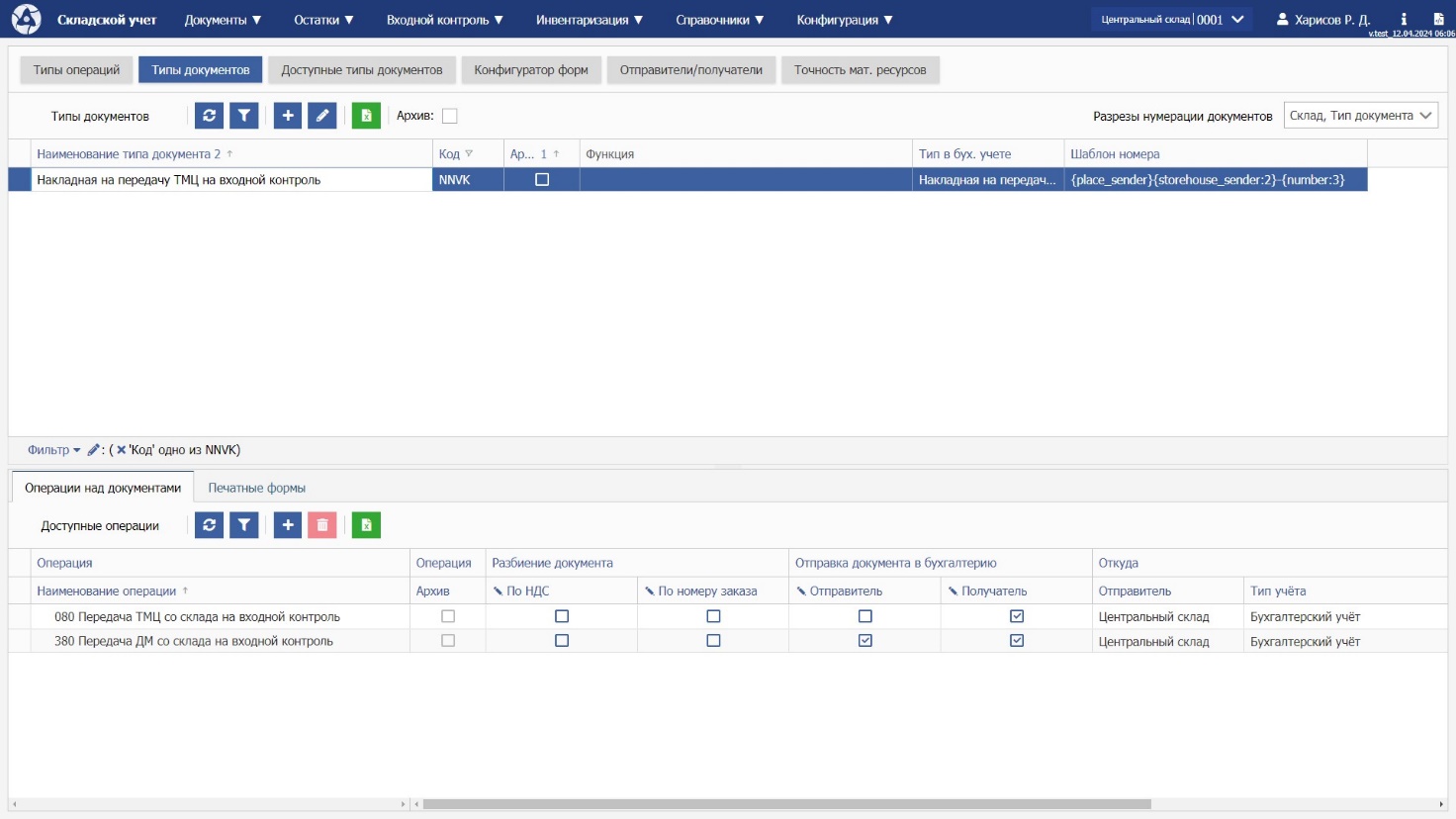


Рисунок 9 – Вкладка «Типы документов»

При создании или редактировании документа открываются одинаковые окна (только при редактировании поля формы заполнены текущими значениями). В окне создания/редактирования есть возможность указать специальный код документа, наименование в системе, признак архива, тип в бухгалтерском учёте (для связи с сервисом «АС Финансы»), функцию документа, а также шаблон его номера.

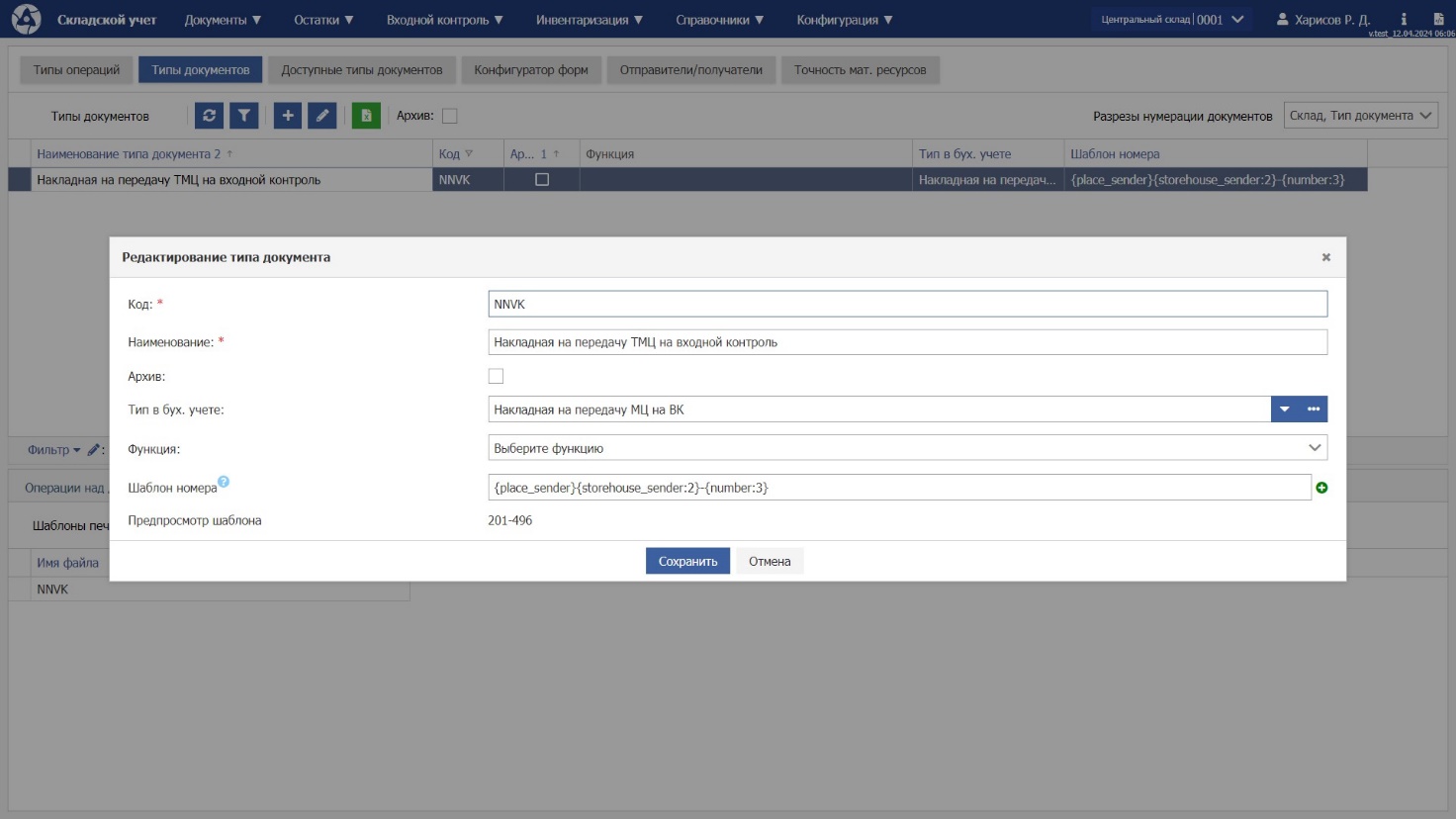


Рисунок 10 – Окно редактирования типа документа

Для генерации шаблона номера можно написать шаблон-строку согласно следующему синтаксису (легенда появляется при наведении на «?» рядом с шаблоном номера):

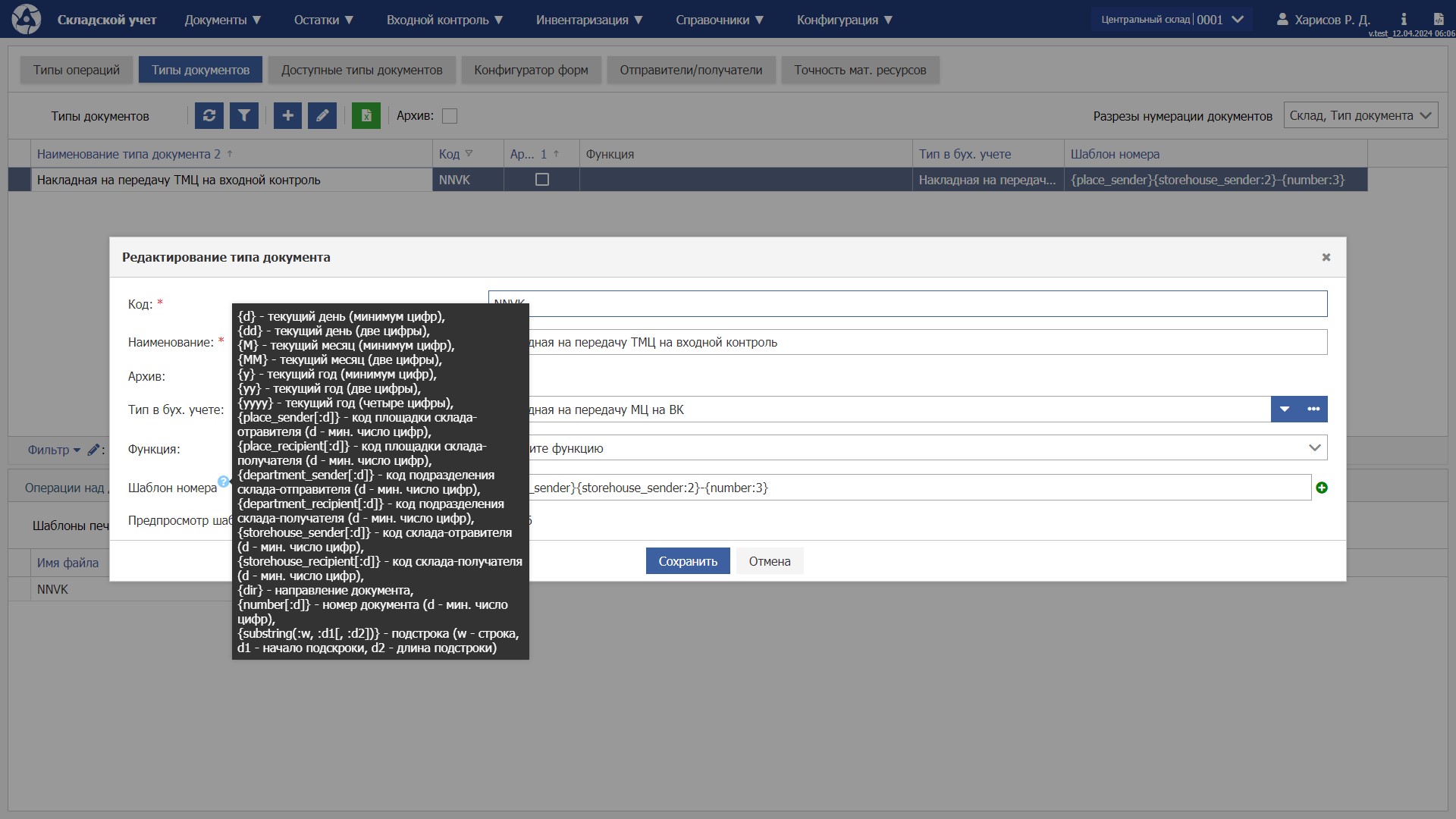


Рисунок 11 – Легенда для шаблон-строки шаблона номера документа

Но также есть возможность создания шаблона при помощи кнопки добавления элементов строки (при нажатии на «+» справа от поля ввода):

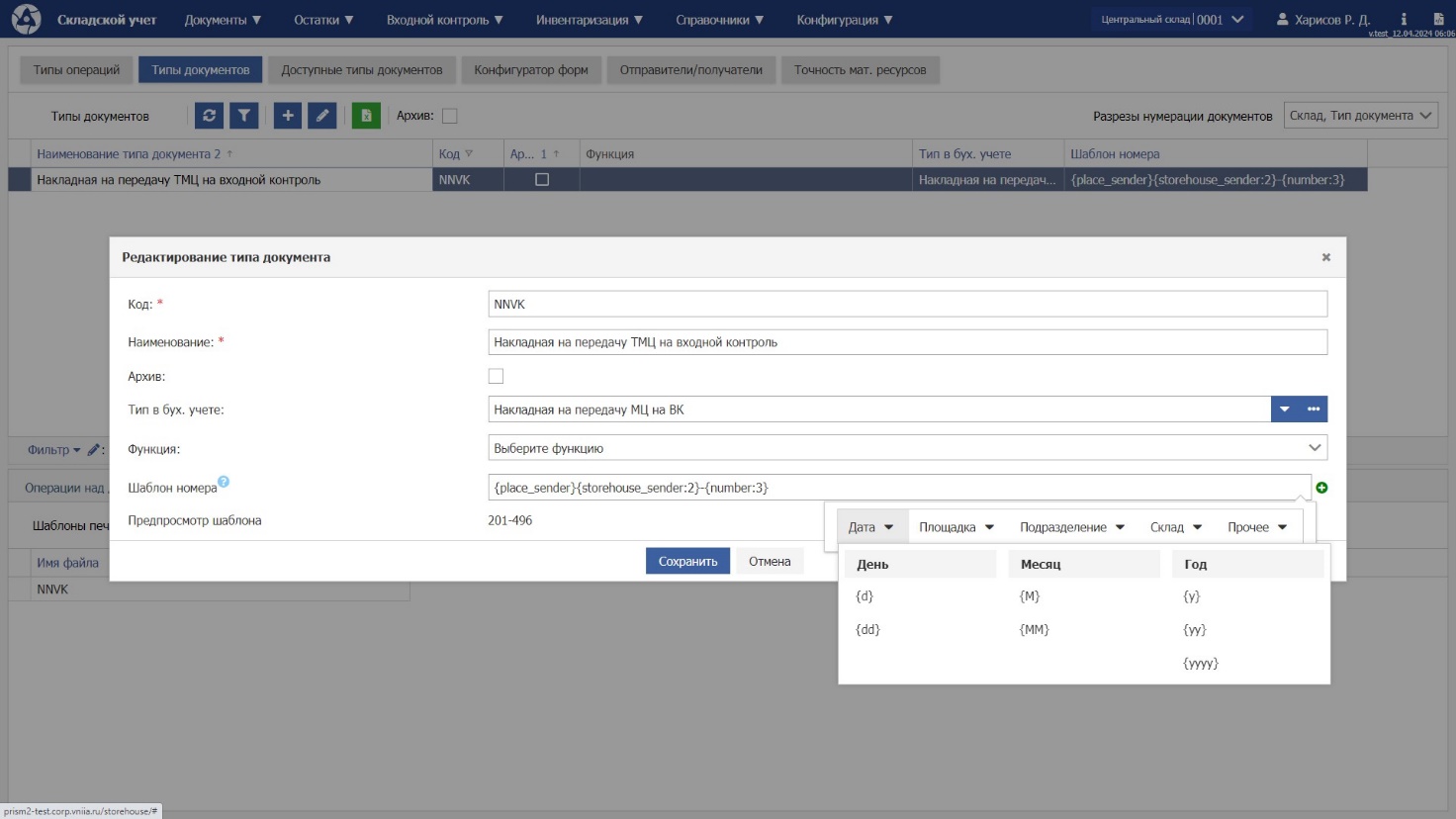


Рисунок 12 – Добавление элемента шаблон-строки шаблона номера документа

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

В первой вкладке можно связать документ с ранее созданными операциями. Во вкладке располагается список связанных с выбранным документом операций и кнопки добавления/удаления, а также стандартные 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.

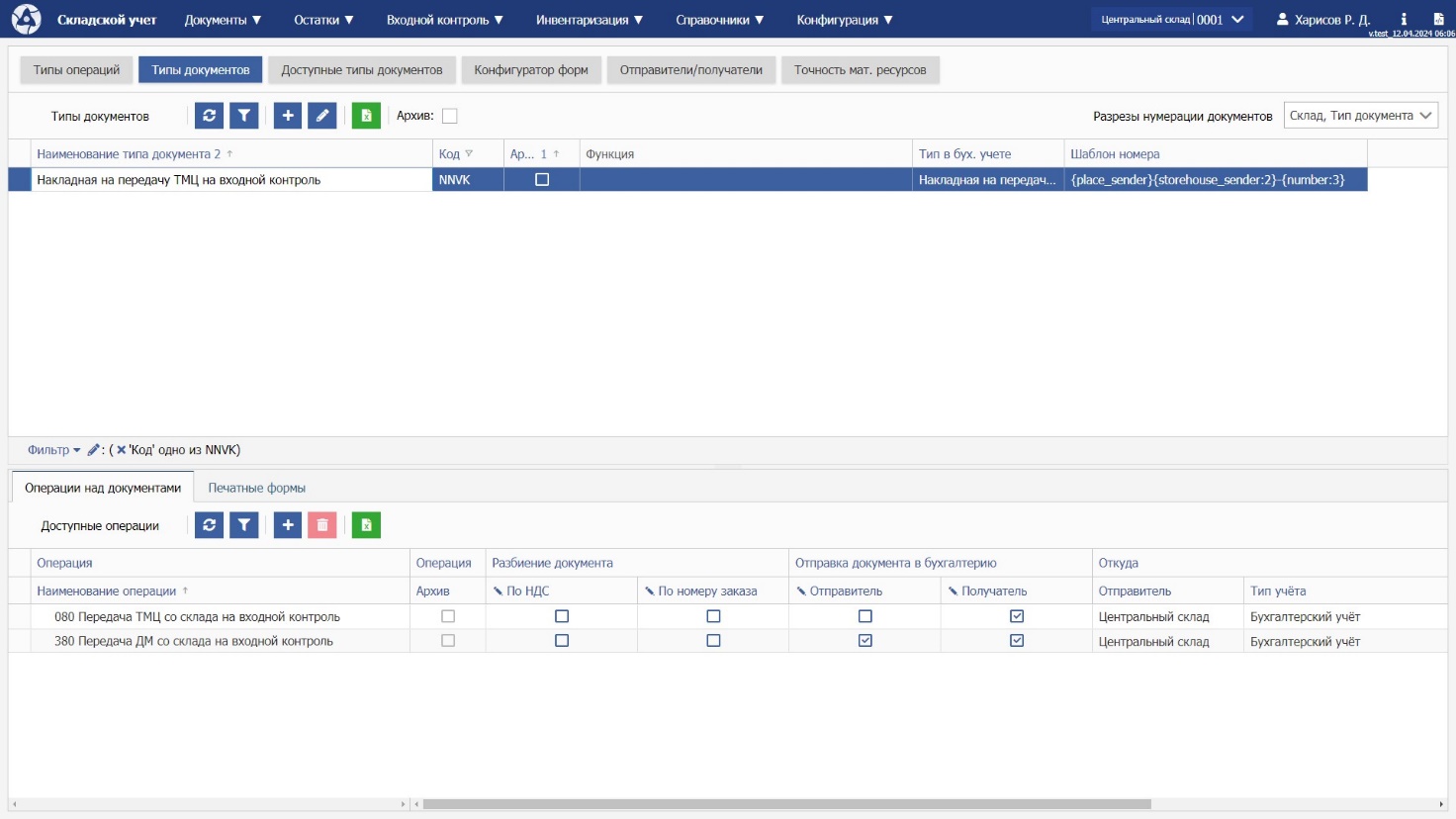


Рисунок 13 – Вкладка «Операции над документами»

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

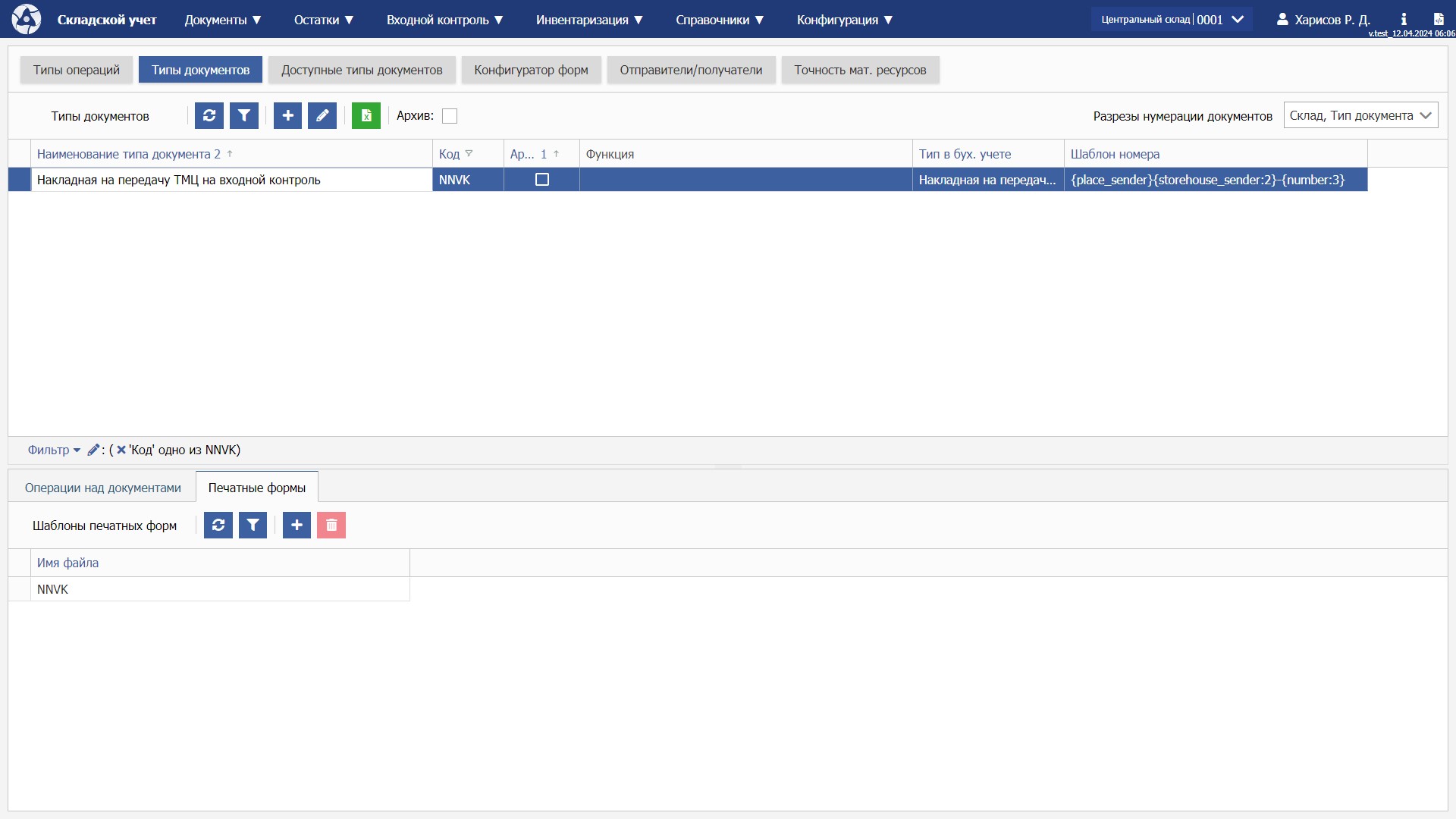


Рисунок 14 – Вкладка «Печатные формы»

1. **Конфигуратор доступных типов документов**

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.

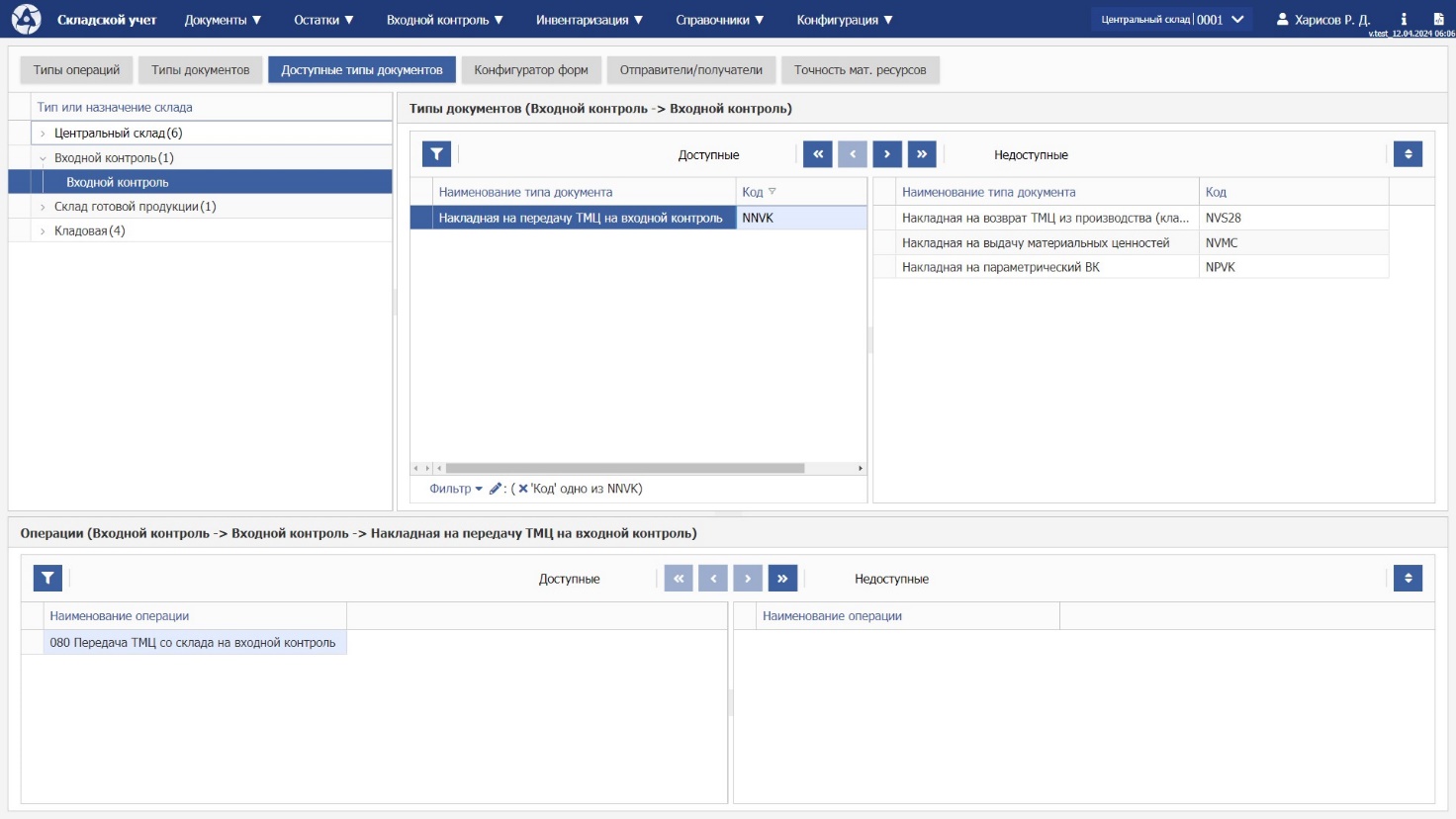


Рисунок 15 – Вкладка «Доступные типы документов»

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.

1. **Конфигуратор форм**

Постановка задачи: одни и те же документы на складах различных предприятий могут иметь различные поля. 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. Это реализуется следующем образом: пользователь заходит в конфигуратор форм, выбирает тип документа, затем выбирает нужный склад и расставляет какие поля будут появляться в форме при создании документа. Остальные поля DTO класса не будут заполняться. При печати документа все поля будут отображаться в необходимых местах.

Процесс реализации конфигуратора форм похож на процесс реализации конфигуратора точности материальных ресурсов, поэтому эта часть будет опущена.

В итоге был получен следующий функционал:

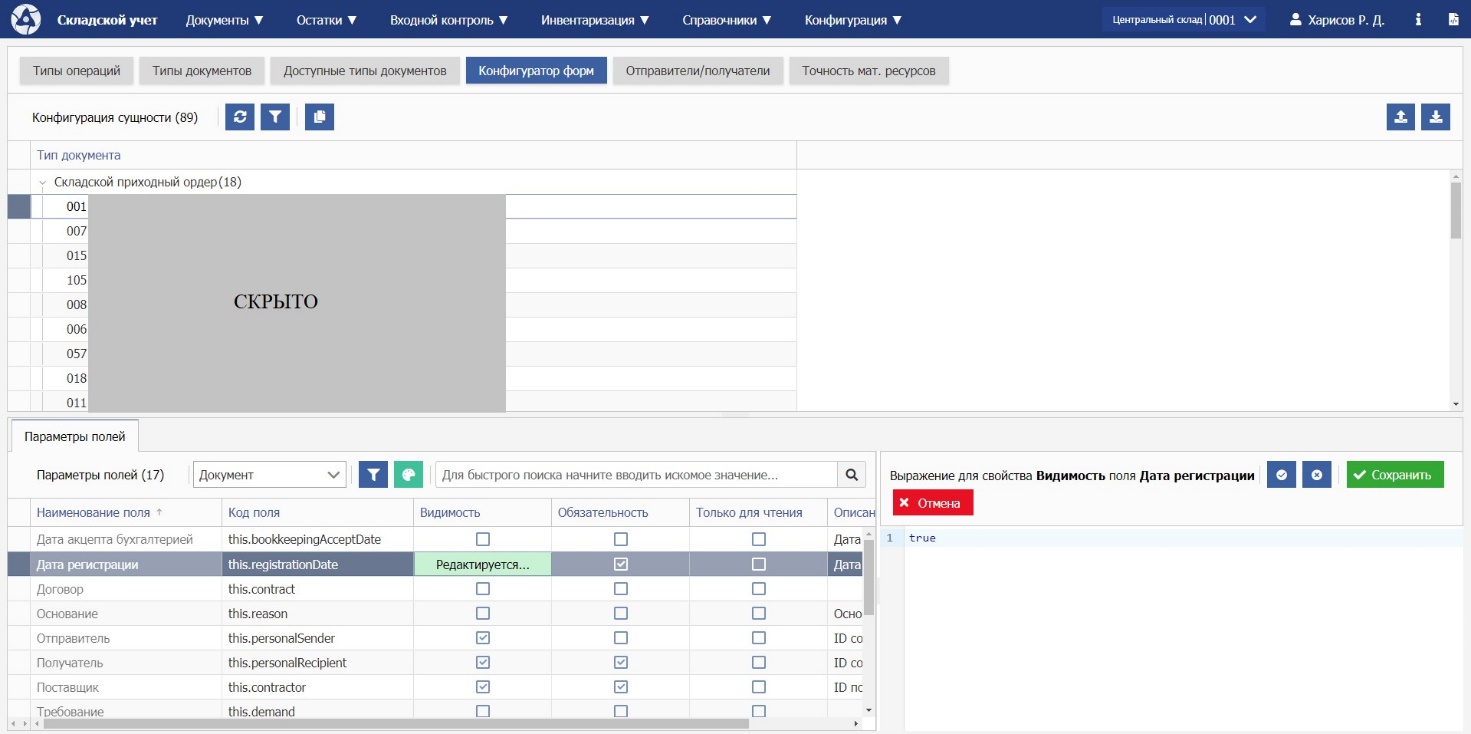


Рисунок 16 – Конфигуратор форм документа

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

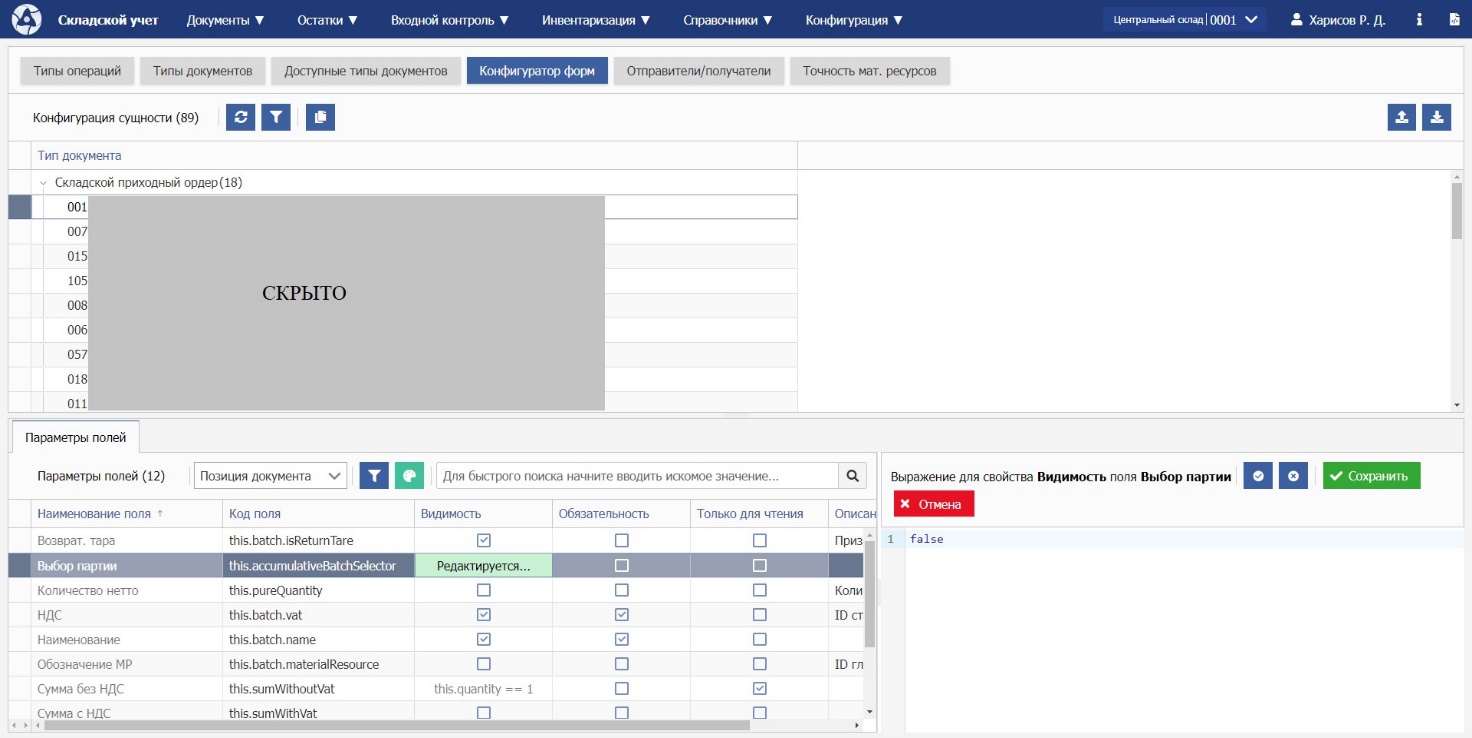


Рисунок 17 – Конфигуратор форм позиций документа

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

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Сьерра К., Бэйтс Б. Изучаем Java. - ЭКСМО, 2022. - 720 с.

2. Шилдт Г. Java. Руководство для начинающих. Современные методы создания, компиляции и выполнения программ на Java. - Диалектика-Вильямс, 2018. - 816 с.

3. Уоллс К. Spring в действии. - 6 изд. - Диалектика-Вильямс, 2022. - 544 с.