**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**ИНСТИТУТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ СИСТЕМ**

**ФАКУЛЬТЕТ АВТОМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ**

**КАФЕДРА ЭЛЕКТРОННЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН**

**Направление** 09.03.01 - Информатика и вычислительная техника

*(код и наименование направления)*

Профиль – Программное и аппаратное обеспечение вычислительной техники

Допускаю к защите

Заведующая кафедрой ЭВМ

\_\_\_\_\_\_\_\_\_ / Долженкова М.Л. /

(подпись) (Ф.И.О.)

**Разработка модуля миграции данных при изменении архитектуры BPM платформы**

Пояснительная записка выпускной квалификационной работы

ТПЖА 09.03.01.723 ПЗ

|  |  |  |  |
| --- | --- | --- | --- |
| Разработал: студент гр.ИВТб-4301-01-00 | \_\_\_\_\_\_\_\_\_\_\_ | / Кудяшев Я.Ю. / | \_\_\_\_\_\_\_\_\_\_ |
|  |  |  |  |
| Руководитель: к.т.н., доцент | \_\_\_\_\_\_\_\_\_\_\_ | / Долженкова М.Л. / | \_\_\_\_\_\_\_\_\_\_ |
|  |  |  |  |
| Нормоконтролёр: к.т.н., доцент | \_\_\_\_\_\_\_\_\_\_\_ | / Скворцов А.А./ | \_\_\_\_\_\_\_\_\_\_ |
|  | *(подпись)* | *(Ф.И.О.)* | *(дата)* |

Киров 2023

Реферат

Кудяшев Я.Ю. Разработка модуля миграции данных при изменении архитектуры BPM платформ: ТПЖА.090301.04.723 ПЗ: ВКР / ВятГУ, каф. ЭВМ; рук. Долженкова М.Л. – Киров, 2023. – Гр.ч. 8 л. ф.А1; ПЗ 95 с., 41 рис., 14 табл., 14 источников, 3 прил.

ЗАПРОСЫ REST API, МАППИНГ СУЩНОСТЕЙ, НАБОР ДАННЫХ, АНАЛИЗ ДАННЫХ, РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ, МОДУЛЬ, МИГРАЦИЯ ДАННЫХ, FRONTEND, JAVASCRIPT.

Объект выпускной квалификационной работы – программное обеспечение для миграции данных при изменении архитектуры BPM платформы.

Целью данной выпускной квалификационной работы является сокращение временных затрат на экспорт данных при переходе с одной платформы на другую.

В данной работе представлены анализ архитектур BPM платформ, выбор наиболее оптимального пути реализации решения, разработка модуля миграции данных, разработка алгоритмов миграции данных и его апробация.

Программное обеспечение разрабатывается в компании ELMA для автоматизированного переноса данных клиентов с платформы, имеющей монолитную архитектуры, на платформу с микросервисной архитектурой.

**СОДЕРЖАНИЕ**

Изм.

Лист

№ докум.

Подпись

Дата

Лист

2

ТПЖА 09.03.01.723 ПЗ

Разраб.

Провер.

*Долженкова М.Л.*

Реценз.

Кудяшев Я.Ю.

Н. Контр.

*Скворцов А.А.*

Утверд.

*Долженкова М.Л.*

Разработка модуля миграции данных при изменении архитектуры BPM платформы

Лит.

Листов

95

Кафедра ЭВМ

Группа ИВТ-41

[ВВЕДЕНИЕ 3](#_Toc138840505)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 4](#_Toc138840506)

[1.1 Проблемы миграции данных 4](#_Toc138840507)

[1.2 Модуль миграции данных 5](#_Toc138840508)

[1.3 Обзор аналогов 6](#_Toc138840509)

[1.4 Актуальность разработки 10](#_Toc138840510)

[1.5 Анализ состояния проблемы 10](#_Toc138840511)

[1.6 Проектирование подходов решения 11](#_Toc138840512)

[1.7 Расширенное техническое задание 23](#_Toc138840515)

[2 РАЗРАБОТКА СТРУКТУРЫ МОДУЛЯ 28](#_Toc138840523)

[2.1 Разработка этапов миграции данных 28](#_Toc138840524)

[2.2 Алгоритмы функционирования для миграции данных 41](#_Toc138840525)

[2.3 Регулярные выражения для маппинга сущностей 45](#_Toc138840526)

[3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 54](#_Toc138840527)

[3.1 Выбор инструментов разработки 54](#_Toc138840528)

[3.2 Реализация этапов миграции данных 58](#_Toc138840529)

[3.3 Апробация модуля 65](#_Toc138840530)

[ЗАКЛЮЧЕНИЕ 71](#_Toc138840531)

[ПРИЛОЖЕНИЕ А. Авторская справка 71](#_Toc138840532)

[ПРИЛОЖЕНИЕ Б. Листинг программного кода 74](#_Toc138840533)

[ПРИЛОЖЕНИЕ В. Библиографический справочник 94](#_Toc138840534)

# ВВЕДЕНИЕ

В последние годы всё больше теряет спрос ручной труд и растёт спрос на автоматизацию. Больше всего данная тенденция затронула бизнес: люди стремятся свести ручной труд к нулю и автоматизировать процессы как внутри компании, так и снаружи. Это, в первую очередь, необходимо для экономии времени и, как следствие, получения большей прибыли.

На сегодняшний день существуют компании по автоматизации бизнеса и бизнес-процессов, которые помогают компаниям сэкономить время, облегчить коммуникацию внутри компании и получить большую прибыль. Лидеров в данной отрасли среди стран СНГ является компания ELMA. Она занимается разработкой систем для управления бизнес-процессами. На данный момент компания имеет в распоряжении несколько платформ: ELMA 3, ELMA4 и ELMA365. Последняя является наиболее ходовым продуктом компании, которая обгоняет своих предшественников в функционале и гибкости настройки. В связи с этим, у многих компаний появилась потребность в переходе с платформ ELMA3/4 на ELMA365. Основная проблема при переходе с одной платформы на другую – различные архитектуры. Помимо этого, компаниям необходимо перенести свои старые данные на новую платформу, где возникают такие же проблемы.

Решением данной проблемы может послужить модуль для внедрения в систему, который будет переносить данные с одной площадки на другую вне зависимости от структуры решения, т.е. выполнять автоматизированный экспорт данных. Перед реализацией необходимо подробно изучить архитектуры обоих площадок, найти их различия и придумать пути решения проблемы по переносу данных, чему и посвящён данный дипломный проект.

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

На данном этапе работы необходимо рассмотреть функции модуля миграции данных для BPM платформ с разными архитектурами, провести обзор аналогов, существующих на сегодняшний день, их возможностей по экспорту данных, выяснить их недостатки и обосновать актуальность разработки текущего модуля.

## Проблемы миграции данных

Проблема миграции данных в глобальном масштабе является важным вопросом, стоящим перед организациями и компаниями, особенно в современной цифровой эпохе. Миграция данных означает перемещение информации из одной системы хранения или местоположения в другую, часто с целью повышения эффективности, сокращения затрат или обеспечения совместимости.

Существует проблема различных стандартов и форматов данных, которые используются в разных странах и организациях. Это может создавать сложности при обмене и совмещении данных, требуя дополнительных усилий для преобразования и согласования форматов.

Также необходимо учитывать факторы пропускной способности сети и доступности сервисов. Передача больших объёмов данных между удалёнными местоположениями может быть затруднена из-за ограниченной скорости соединения или нестабильного интернет-соединения.

Другой значимой проблемой является сохранение целостности данных во время миграции. В процессе передачи и преобразования данных возникает риск их повреждения или потери. Поэтому необходимо предусмотреть механизмы резервного копирования и проверки целостности данных.

Для решения проблем миграции данных в глобальном плане требуется тщательное планирование, анализ и подготовка. Организации должны учитывать затраты как по времени, так и по ресурсам на разработку специализированных приложений для осуществления миграции данных своей компании

В целом, проблема миграции данных в глобальном масштабе требует комплексного подхода и внимательного рассмотрения различных аспектов. Именно с этой целью и был создан модуль, представленный в дипломном проекте.

## Модуль миграции данных

Функциями модуля миграции данных для BPM платформ с разными архитектурами являются:

* авторизация на платформе ELMA3 и получение токена пользователя с правами администратора;
* получение структуры элементов, хранящихся на платформе ELMA3, и их значений;
* получение и просмотр структуры элементов платформы ELMA365;
* возможность вручную соотносить свойства экспортируемых элементов с платформы ELMA3 со свойствами структуры в ELMA365;
* возможность автоматически соотнести свойства экспортируемых элементов платформы ELMA3 со свойствами структуры в ELMA365 на основе типов данных с дальнейшим их ручным изменением;
* создание новых элементов в ELMA365 с соответствующими данными и структурой элементов из ELMA3;
* присвоение признака экспортированного элемента новым на платформе ELMA36.

## Обзор аналогов

На данном этапе необходимо рассмотреть текущие возможности и инструменты для миграции данных между платформами, проанализировать их функционал, составить список основных недостатков и ограниченностей моделей и сделать выводы по данной части.

* + 1. Стандартный импорт данных

На сегодняшний день на любую площадку платформы ELMA365 можно без особого труда стандартными средствами перенести данные в любое приложение раздела путём подготовки, структурированной заранее таблицы Excel или файла формата .csv. На рисунке 1 приведён пример формы для приёма импортируемого файла в ELMA365. В область загрузки необходимо загрузить файл в описанном выше формате и далее начнётся автоматический процесс импорта данных.

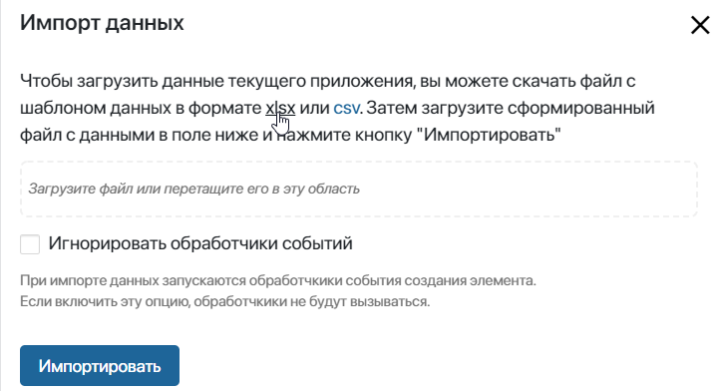


Рисунок 1 - Форма импорта данных в ELMA365

Общий процесс импорта данных в систему таким путём состоит из следующих шагов:

Создание приложения, в котором будут храниться данные. Данный этап подразумеваем подготовку структуры, которая должна удовлетворять потребностям как заказчика, так и импортируемых объектов;

Генерация шаблона документа. На этом этапе система предложит пользователя ознакомиться с шаблоном, которому должен удовлетворять файл с импортируемыми данными в формате .xlsx или .csv;

Приведение импортируемого файла в соответствие с шаблоном. Этап, на выполнение которого требуется больше всего времени, т.к. данные должны быть получены и приведены в строго соответствующий шаблону формат;

Загрузка подготовленного файла с данными в систему. На последнем этапе происходит перенос данных в систему и занесение их в таблицы в базе данных PostgreSQL.

Необходимо обратить внимание, что индексы создаются в БД для каждого свойства приложения, что значительно увеличивает время импорта данных. Наличие любого несоответствия типов данных мгновенно прерываем процесс импорта. На рисунке 2 приведён пример неудачного импорта данных, с наличием несоответствия типов данных.

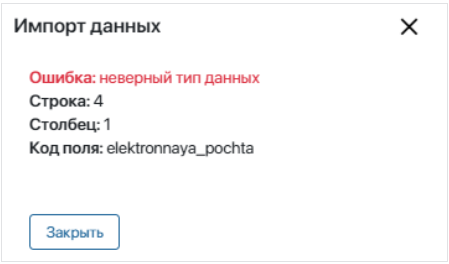


Рисунок 2 - Ошибка при импорте данных

Главными особенностями такого импорта является отсутствие возможности редактирования структуры файла прямо на платформе, что увеличиваем время для обработки данных, а также необходимость указания ссылок на приложения в виде идентификаторов уже существующих в системе элементов, что делает данный способ практически нелегитимным для импорта данных из ELMA3, т.к. элементов, на которые должна ссылаться система, ещё нет.

* + 1. Интеграционное приложение с ELMA365

Относительно недавно в ELMA365 Store появилось решение, позволяющее осуществлять миграцию данных с платформы ELMA3/4 на платформу ELMA365. Данное решение представляет из себя модуль для платформы ELMA3/4.

К ключевым особенностям и ограничениям данного модуля относится:

* привязка к платформе ELMA3/4. Для взаимодействия и контроля импорта данных;
* платная основа. Модуль является платным решением и находится в магазине ELMA Store;
* отсутствие интерфейса. Нет как такого интерфейса для взаимодействия с пользователем. Модуль работает в виде консольного приложения;
* миграция данных для системных разделов. Импорт данных возможен только для системных разделов, которые идут в стандартной поставке ELMA365;
* платформа разработки – C#. Основная часть модуля реализована на языке C#, который связан с платформой ELMA3, но никак не привязан к платформе ELMA365.

После анализа функционала и особенностей можно сделать вывод, что данный модуль подойдёт для компаний, которые пользуются стандартными решениями компании ELMA и не имеют в своём распоряжении решений, подготовленных специально для их компании: со своей структурой, свойствами и особенностями.

* + 1. Недостатки аналогов

Рассмотрев аналоги и обозрев их функционал можно точно сказать, что они имеют большое количество недостатков и ограниченного функционала. К основным относятся:

* отсутствие интерфейса для управления процессом экспорта данных. Для экспорта данных необходимо вручную приводить экспортируемые сущности к виду, соответствующему структуре на платформе ELMA365;
* отсутствие массового импорта данных на уровне всех разделов и приложений. Одновременно можно импортировать данные лишь в одно приложение раздела, что заставляет тратить на данный процесс огромное количество времени;
* отсутствие явных признаков импортированных объектов. Нет чётких индикаторов в системе, говорящих о принадлежности объектов другой платформе, что иногда затрудняет работу с ними;
* неявный формат ссылок на элементы при импорте данных. Как было сказано выше, для импорта связей элементов с другими элементами системы, необходимо прописывать идентификаторы этих объектов. При этом, нельзя сослаться на элемент, который ещё отсутствует в системе;
* дублирование сущностей. При импорте сущностей в систему нет проверки на дубли, что значительно осложняет работу в систему в виду одинаковых элементов;
* ненадёжность инструментов импорта. При возникновении технических проблем во время импорта сущностей, нет возможности продолжить импорт с того элемента, на котором он был прерван;

## Актуальность разработки

Модуль, позволяющий осуществить плавный перенос данных со старой платформы на новую на сегодняшний день имеет максимальный приоритет в разработке. Это, в первую очередь, связано с потребностью клиентов в совершенствовании и плавным переходом со старого продукта компании ELMA3, архитектура которого представляет из себя монолит, на новую платформу ELMA365, являющуюся микросервисным решением. На данный момент все больше клиентов уже проектируют решения на новой платформе и после завершения данной процедуры необходим плавный перенос данных компаний со старой платформы на новую с учётом новой архитектуры и свойств системы, для чего и предназначен разрабатываемый модуль.

## Анализ состояния проблемы

Совершенствование и плавный переход клиентов с монолитной архитектуры старого продукта ELMA3 на новую микросервисную платформу ELMA365 является актуальной задачей для компании. Старая платформа, представляющая из себя монолит, становится устаревшей и ограничивает возможности клиентов в области автоматизации и управления бизнес-процессами.

Основными проблемами, которые требуется решить, являются:

* необходимость плавного переноса данных: При переходе на новую платформу ELMA365 необходимо осуществить перенос данных компаний, использующих старую платформу ELMA3. Этот процесс требует специального модуля, который позволит осуществить плавный перенос данных, учитывая новую архитектуру и свойства системы. Плавный перенос данных важен, чтобы минимизировать простои и сбои в работе компаний, которые уже проектируют свои решения на новой платформе.
* разница в архитектуре: Старая платформа ELMA3 является монолитным решением, в то время как новая платформа ELMA365 основана на микросервисной архитектуре. Это приводит к различиям в структуре, функциональности и способе взаимодействия компонентов системы. Перенос данных с учётом новой архитектуры требует специального подхода и модуля, который обеспечит совместимость данных и их корректное размещение на новой платформе.
* потребности клиентов: Клиенты компании выражают потребность в совершенствовании и переходе на новую платформу ELMA365. Большое количество клиентов уже начинают проектировать свои решения на новой платформе, поэтому важно предоставить им возможность безопасно и эффективно перенести свои данные с предыдущей платформы. Модуль, разрабатываемый в данный момент, предназначен именно для удовлетворения этих потребностей клиентов.

Анализ состояния проблемы показывает, что необходимость плавного переноса данных со старой платформы ELMA3 на новую платформу ELMA365 является критической для компании. Разработка специального модуля, который позволит осуществить данный перенос, будет решением проблемы и поможет клиентам безопасно и эффективно перейти на новую платформу, с учётом её микросервисной архитектуры и свойств системы.

## Проектирование подходов решения

В данном разделе представлен процесс проектирования и анализа архитектур обоих платформ для понимания и поиска наиболее оптимальных путей решения. Анализ платформ, приведённый в данном разделе, необходим для проектирования дальнейшего решения, описанного в разделе программной реализации.

* + 1. Исследование архитектуры ELMA3

Работа системы ELMA 3/4 предполагает наличие необходимой и достаточной серверной инфраструктуры на стороне заказчика. Развертывание системы возможно в двух вариантах: Single Instance и Web Farm.

Single Instance предполагает, что для работы системы достаточно одного сервера приложений. Он является безальтернативным вариантом для конфигураций ниже Enterprise. Данную конфигурацию используют небольшие компании, для работы которых хватает мощности одного сервера. На рисунке 3 представлена схема Single Instance.

Web Farm предназначен для более крупных компаний, которым необходима мощность более 1 сервера и наличие резервного сервера для обеспечения надежности. Данный вид компоновки предполагает, что для работы системы недостаточно одного сервера приложений и требуется кластеризация серверов для повышения отказоустойчивости при работе системы. Доступен только в конфигурации Enterprise, наряду с Single Instance. Схема Web Farm представлена на рисунке 4.

В зависимости от потребностей и масштаба бизнеса, выбор между Single Instance и Web Farm может быть осуществлен, обеспечивая оптимальную инфраструктуру для работы системы ELMA 3/4 и удовлетворение потребностей компании.

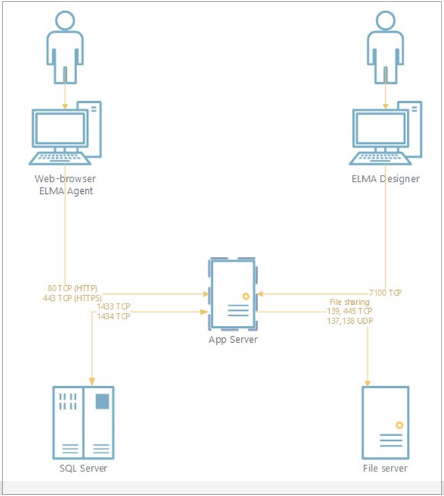


Рисунок 3 - Схема Single Instance

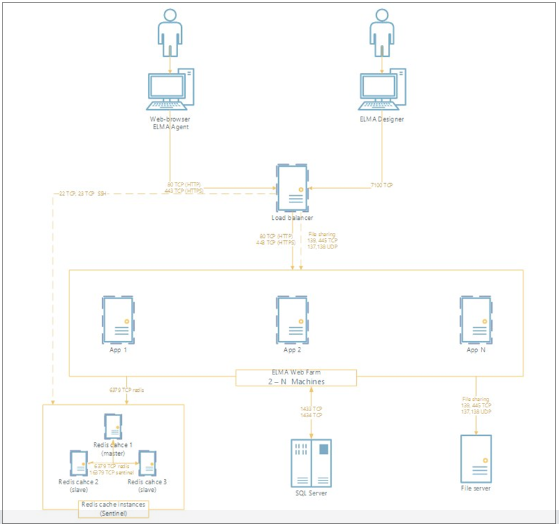


Рисунок 4 – Схема Web Farm

Безусловно, основными компонентами системы являются сервер приложений и сервер базы данных, однако есть множество дополнительных компонентов системы, которые влияют на её работу. На рисунке 5 представлена схема взаимодействия компонентов системы.

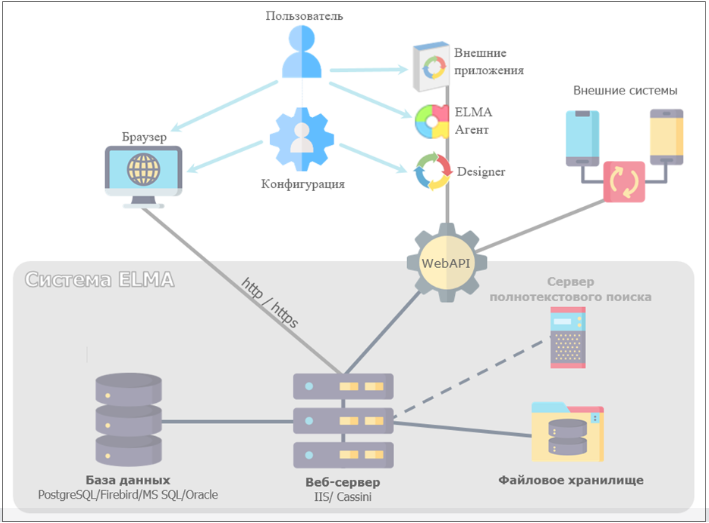


Рисунок 5 – Схема взаимодействия компонентов системы

По умолчанию, компонент в ELMA3/4 состоит из двух модулей:

* серверная часть;
* веб часть.

Серверная часть представляет собой подключаемую к приложению библиотеку и содержит бизнес-модель и связанную с ней логику.

Веб часть представляет собой полноценное ASP .NET MVC 4.0 приложение, которое полностью реализует паттерн MVC в своей реализации архитектуры.

Структура проекта подобного приложения состоит из следующих рекомендованных директорий:

* App\_Data – Место хранения данных для приложения. Это могут быть как простейшие базы данных, так и XML файлы и прочие ресурсы;
* Content – Рекомендуемое место для хранения статичного содержимого приложения (стили, изображения и т.п.);
* Scripts – Рекомендуемое место для хранения скриптов;
* Controllers – место для хранения классов контроллеров приложения. Каждый контроллер должен заканчиваться словом «Controller» так требует Фреймворк MVC;
* Models – Место для хранения классов моделей с которыми будет работать приложение и взаимодействовать контроллеры и представление;
* Views – Место расположения представлений. Для Razor разметки используются файлы с расширением «.cshtml».

Файлы представлений являются шаблоном HTML страницы. Для преобразования шаблона страницы в готовую для передачи HTML разметку ASP.NET MVC использует движки представлений, такие как WebForms или Razor. Так же можно подключить другие движки. В ELMA используется Razor.

Объектная модель системы – это набор классов, сущностей и перечислений, которые хранят данные конфигурации и используется в бизнес-логике. Под сущностью понимается объект способный хранить данные в базе данных.

Для описание объектной модели в системе ELMA BPM используются метаданные. Метаданные – это специализированные данные, предназначенные для описания данных объектной модели. Для их создания используется Дизайнер ELMA или плагин к Visual Studio, позволяющий визуально описывать объектную модель системы.

В файловой системе все метаданные хранятся в виде файлов с расширением .md, которые в действительности представляют собой xml-файлы. На основе метаданных с использованием механизма кодогенерации генерируется код C#, который описывает объект, хранящийся в базе данных. На рисунке 6 представлена оболочка редактора сущностей, в которой видны свойства, задаваемые при создании какой-либо сущности.

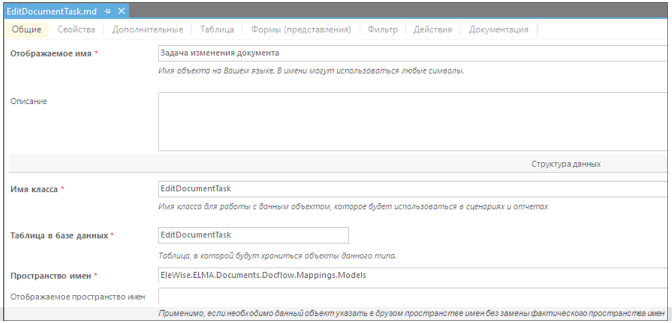


Рисунок 6 – Информация при создании сущности

На данной вкладке вводится общая информация о создаваемой сущности. Приведены следующие поля:

* отображаемое имя – используется для отображения названия типа в списке объектов. Данное поле является локализуемым;
* описание – описание типа создаваемой сущности. Поле является локализуемым;
* имя класса – название класса, описывающего тип создаваемой сущности;
* таблица в базе данных – название таблицы в базе данных, в которую будут сохранятся экземпляры создаваемой сущности;
* пространство имён – название пространства имён, в котором будет расположен создаваемый тип сущности;
* отображаемое пространство имён – используется для указания пространства имён.

В базе данных объектная модель представляет собой множество взаимосвязанных таблиц, соответствующих классам сущностей системы.

Важно учитывать определённую особенность архитектуры данных, которая называется «Наследование». Некоторые объекты системы имеют общего «родителя», который реализует общую для «потомков» составляющую: атрибуты, методы, логику работы.

Помимо этой особенности системы есть ещё одна: промежуточные таблицы связей. Данные таблицы, как правило, состоят из двух столбцов: Parent и Child и служат для организации хранения данных со связью N-N. Parent хранит Id основного объекта, а Child – Id связанных объектов.

Для наглядности будет рассмотрен пример связей сущностей при помощи MS SQL Server Manager Studio. На рисунке 7 приведён пример связи между двумя таблицами. Его трактовка с точки зрения БД будет пониматься как «Таблица Absence связана с таблицей GrounsForAbsence через атрибут Grounds, в котором хранится Id записи таблицы GrounsForAbsence. Хранение реляционное, а обработка объектная.

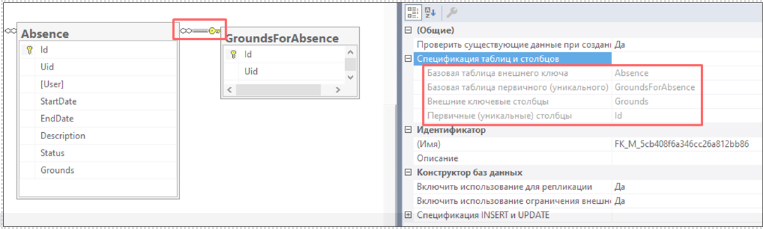


Рисунок 7 – Описание связи между таблицами

С точки зрения ELMA 3/4, эта информация обозначает, что объект с классом Absence (Отсутствие пользователя) содержит атрибут Grounds (Основание для отсутствия на рабочем месте) с типом атрибута GroundsForAbsence (Основание для отсутствия на рабочем месте) и одиночной связью.

* + 1. Исследование архитектуры ELMA365

Платформой-преемником данных будет выступать ELMA365 – Low-code платформа для автоматизации внутренних бизнес-процессов и CRM, являющаяся более новым продуктом компании и лидером среди BPM-систем в СНГ.

Система ELMA365 является микросервисным веб-приложением, построенным на базе следующих технологий: GoLang, Node.js, Angular 2. Используемые СУБД: PostgreSQL, MongoDB. Кэш-сервис – Redis. В качестве шины данных используется RabbitMQ.

PostgreSQL, MongoDB, Redis, RabbitMQ и S3 (minio, ceph, и другие S3-совместимые хранилища) могут быть включены в поставку ELMA365 или развёрнуты заказчиком самостоятельно.

Приложение ELMA 365 можно разделить на клиентскую часть (фронт) и серверную (бэк). На сервере хранятся все данные, а также там расположены микросервисы и подсистемы, отвечающие за работу приложения и взаимодействие его компонентов. Клиентская часть предоставляет пользователям интерфейс для работы с системой из браузера. Работа пользователей в системе осуществляется при помощи любого современного веб-браузера с поддержкой НТМL5, CSS, JavaScript, JQuery. Рекомендуемым браузером для работы с ELMA365 является Google Chrome. Поддерживаются браузеры Firefox, Орега, Safari.

Сервер веб-приложения использует контейнеризацию на базе Docker, а также оркестратор – Kubernetes. Кластер веб-приложения ELMA365 может быть развернут на серверных мощностях заказчика и быть доступен как в Internet, так и в Intranet.  Используется объектное хранилище на базе S3 протокола.

Компоненты системы выстроены в 3 слоя, а их взаимодействие основано на концепции MVC (Model-View-Controller). Данная концепция разделяет функционал приложения на 3 отдельные подсистемы: Model (хранение данных), View (визуализация данных) и Controller (обработка данных). В реалиях ELMA365 хранение данных обеспечивается несколькими СУБД, обработка выполняется набором микросервисов, а за отображение отвечает фронт. Данный подход позволяет изолировать данные подсистемы, что упрощает разработку и дальнейшую поддержку.

При проектировании приложений используются два различных подхода: «монолитная» архитектура и микросервисная. ELMA 365 построена на микросервисной архитектуре: система состоит из множества изолированных микросервисов со своей логикой и данными, каждый из которых отвечает за свой функционал. Так, например, processor управляет движением бизнес-процессов, worker отвечает за работу пользовательских сценариев, а mailer – за отправку писем по электронной почте. На рисунке 8 представлена схема подходов проектирования в ELMA365, где МС – микросервис.



Рисунок 8 – Подходы проектирования в ELMA365

Микросервисная архитектура обладает рядом достоинств по сравнению с монолитной. В первую очередь она позволяет изолировать функционал в отдельные компоненты, что упрощает их разработку и поддержку. Это также повышает стабильность работы системы: при отказе одного из микросервисов остальные продолжают работать в штатном режиме. Кроме того, выделение функционала в отдельные изолированные компоненты позволяет экономить ресурсы в случае, когда необходимо запустить несколько экземпляров микросервисов, т.к. не приходится запускать экземпляр приложения целиком.

К сожалению, минусы у микросервисной архитектуры также присутствуют. Изолированность микросервисов друг от друга приводит к дополнительным накладным расходам для организации их взаимодействия, а также вызывает дополнительные временные задержки на обмен данными между компонентами. Также может усложниться локализация ошибки в случае, когда в обработке запроса участвовали несколько микросервисов.

Несмотря на то, что функционал приложения очень напоминает классы ООП (имеются свои поля – атрибуты, свои методы – бизнес процессы и другие действия, каждый элемент приложения – это отдельный экземпляр), приравнивать их друг к другу не совсем корректно. Приложение – некая абстрактная сущность, которая, помимо полей и методов также имеет обширный ряд своих настроек, статусов, доступных фильтров и т.д.

В таблице AppViews в PostgreSQL хранятся приложения и их описания. В данной таблице присутствует множество колонок, но основные данные связаны с тремя основными структурами: Fields, Settings и Filters.

Структура Fields служит описанием полей контекста приложения. Она содержит информацию о каждом поле, такую как его название, тип данных, формат отображения и другие параметры. Например, на рисунке 9 представлена структура поля типа "Категория", где указаны соответствующие атрибуты и настройки для этого типа поля.

Для полей типа "приложение" используется структура SYS\_COLLECTION. Этот тип поля позволяет связывать данные с другими полями. Функционал связанных полей определяется с помощью поля linkedFieldCode. На рисунке 10 приведён фрагмент описания поля типа "Приложение", где указаны соответствующие атрибуты и настройки для этого типа поля.

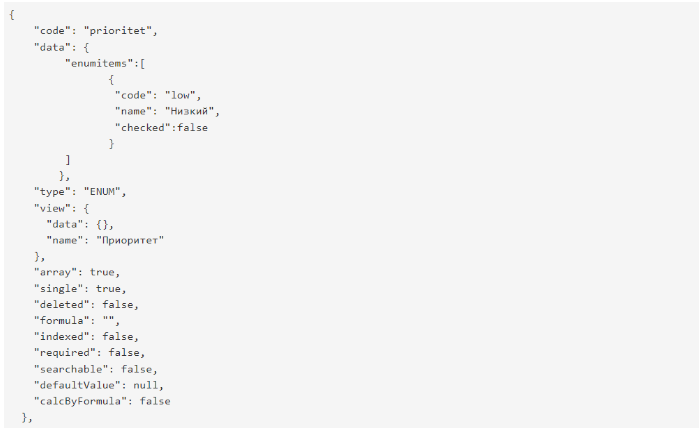


Рисунок 9 – Структура поля типа «Категория»



Рисунок 10 – Структура поля типа «Приложения»

Postgre SQL является основной СУБД, используемой в ELMA365. В ней хранятся элементы приложения и другие основные сущности платформы.

PostgreSQL – основное хранилище данных и структур конфигурации. Хранит:

* элементы приложений;
* экземпляры процессов;
* группы;
* пользователи (ФИО, дата приёма на работу и другие данные объекта типа «Пользователь»);
* метаданные файлов (значения атрибутов: кем создан, кем изменён и т.д.);
* описания приложений (поля, фильтры, фильтр по умолчанию, настройки, статусы и т.д.);
* шаблоны бизнес-процессов;
* описания виджетов;
* описания модулей;
* описание организационной структуры;
* исполнительская дисциплина и т.д.

В PostgreSQL лежат не все данные пользователей, а только данные объекта типа Пользователь, такие, как ФИО, дата приёма на работу, и т.д. На уровне системы Пользователи являются коллекцией, поэтому удобно хранить их в Postgre, тогда как данные аутентификации хранятся в Mongo.

Каждая из вышеописанных сущностей хранится в своей таблице. Данные, отправляемые в ELMA (например, описание полей приложения) хранятся в формате JSON. На рисунке 11 приведён пример структуры JSON-файла.



Рисунок 11 - Структура JSON-файла

## Расширенное техническое задание

В данном разделе представлено техническое задание на разработку модуля миграции данных для BPM платформ с разными архитектурами.

* + 1. Краткая характеристика области применения

Модуль предназначен для клиентов компании ELMA, которые ранее использовали для работы монолитную платформу ELMA3 и решили перейти на новую платформу с микросервисной архитектурой ELMA365.

* + 1. Назначение разработки

Функциональным назначением модуля является предоставление возможности клиентам, пользовавшимся платформой ELMA3, перенести данные с площадки своей компании на новую площадку, развёрнутую на основе микросервисной и более современной платформы ELMA365.

Модуль должен эксплуатироваться на ПК клиентов, вне зависимости от наличия решения на платформе ELMA3. Модуль доступен для загрузки как на облачную платформу ELMA365, так и на персональную, развёрнутую на сервере клиента. Особые требования к конечному пользователю не предъявляются.

* + 1. Требования к модулю

Модуль должен обладать следующими функциональными возможностями:

1. отображение структуры сущностей платформы ELMA3;
2. отображение структуры сущностей платформы ELMA365;
3. возможность вручную соотносить свойства сущностей двух платформ с соответствующей валидацией;
4. автоматическое соотношение свойств сущностей на основе регулярных выражений;
5. сохранение текущей сессии экспорта данных в БД платформы;
6. подсчёт потраченного на экспорт времени;
7. шкала прогресса процесса импорта данных;
   * 1. Требования к надёжности

Надёжное (устойчивое) выполнение программы должно быть обеспечено выполнением пользователем совокупности организационно-технических мероприятий, перечень которых приведён ниже:

1. организацией бесперебойного питания технических средств;
2. использованием лицензионного программного обеспечения.

Отказы модуля возможны вследствие некорректных действий пользователя при взаимодействии с браузером или вмешательством в код страницы другими пользователями. Во избежание возникновения отказов программы по указанным выше причинам следует обеспечить работу конечного пользователя без предоставления ему возможности вмешиваться в код страницы.

* + 1. Требования к составу и параметрам технических средств

В состав технических средств должен входить IBM-совместимый персональный компьютер, включающий в себя:

1. дисплей с разрешением не меньше, чем 1024x768;
2. не менее 4-х гигабайт оперативной памяти;
3. клавиатура, мышь.

Помимо вышеуказанных требований к техническим средствам и программным средствам, важно также обеспечить соответствующую поддержку и обслуживание системы, включая регулярные обновления операционной системы и браузера, а также резервное копирование данных. Это поможет поддерживать стабильную работу модуля и обеспечить надёжность и безопасность в процессе его использования.

Системные программные средства, используемые программой, должны быть представлены следующими операционными системами:

* 64-разрядная OC Windows 7 или более поздней версии;
* Mac OS X 10.10 или более поздняя версия;
* Linux ARM 64 bit.
  + 1. Требования к интерфейсу

Модуль должен обеспечивать взаимодействие с пользователем посредством графического пользовательского интерфейса и предоставлять возможность осуществлять ввод данных путём использования физической клавиатуры.

Предполагается использовать организацию интерфейса, выполненную в стилистике платформы ELMA365, в характерных для платформы цветах. Это необходимо для предотвращения ощущения несвязности контекста модуля.

Также необходимо обеспечить поддержку экранов с высоким разрешением.

* + 1. Требования к программной документации

Состав программной документации должен включать в себя:

1. техническое задание;
2. руководство пользователя;
3. техническую документацию;
4. исходный код.

Вывод

Проведя анализ предметной области можно сделать следующие выводы:

* текущие инструменты и модули для миграции данных имеют ряд недостатков, которые мешают каждой компании мигрировать данные на новую платформу, главные из которых: отсутствие возможности экспорта данных с платформы ELMA3 и невозможность экспорта кастомных разделов платформы;
* данная тематика и разработка модуля актуальна на текущий момент, т.к. большая часть клиентов стремится к переходу на новую платформу;
* наиболее продуктивной разработка видится как модуль к платформе ELMA365, а не ELMA3, в связи с её устаревшей архитектурой.

Также в данном разделе был выполнен анализ архитектур двух платформ: ELMA3 и ELMA365. Основной задачей было понять, то, как они устроены, как организовано хранение данных на платформах и решить – какая платформа подходит более для разработки модуля миграции данных.

Результаты анализа показали, что платформа ELMA365 имеет больший функционал и возможности для создания модуля, отвечающего требованиям, поставленным в разделе анализа предметной области.

Помимо этого, было рассмотрено расширенное техническое задание, приведены минимальные системные требования для работы модуля и проведён обзор требуемой программной документации.

На основе этих данных можно приступить к реализации самого модуля алгоритмов функционирования, необходимым для плавного и полноценного переноса данных с BPM платформ с разными архитектурами.

# 

# РАЗРАБОТКА СТРУКТУРЫ МОДУЛЯ

В данном разделе в соответствии с требованиями, поставленными в техническом задание, описана структура модуля, алгоритмы функционирования, а также определён механизм валидации атрибутов сущностей для обеих платформ для избегания конфликтных ситуаций.

* 1. Разработка этапов миграции данных

В данном разделе будет рассмотрен основной алгоритм, необходимый пройти пользователю для проведения экспорта данных при установке модуля к себе на платформу.

* + 1. Получение токена пользователя

После получения токена пользователя платформы ELMA3 через внешний API, следующим шагом будет использование полученного токена для выполнения операций с данными. Взаимодействие с данными осуществляется с помощью методов и запросов, предоставляемых сервисами платформы ELMA3.

Один из важных сервисов, доступных на сервере, это сервис метаданных системы. Он предоставляет информацию о структуре данных, сущностях, полях и их связях. Это позволяет получить полное представление о доступных объектах и их взаимосвязи, что необходимо для правильной обработки и экспорта данных.

Дополнительно к корневым и функциональным сервисам, платформа ELMA3 также предоставляет возможность настройки и расширения функциональности через создание собственных сервисов. Это позволяет разработчикам адаптировать платформу под конкретные потребности и интегрировать её с другими системами.

В целом, взаимодействие с данными и функциональностью платформы ELMA3 осуществляется через соответствующие сервисы, которые предоставляют необходимые методы и запросы. Это позволяет эффективно управлять данными, выполнять операции и осуществлять экспорт данных для последующего использования в модуле ELMA365.

Для каждого типа данных и сущностей присущи свои уникальные свойства и признаки. Так, к примеру, у каждой сущности есть свой id, позволяющий идентифицировать сущность как элемент того или иного приложения. На рисунке 11 представлен макет интерфейса для взаимодействия пользователя и системы при вводе данных от учётной записи для дальнейшего получения токена пользователя и экспорта информации.

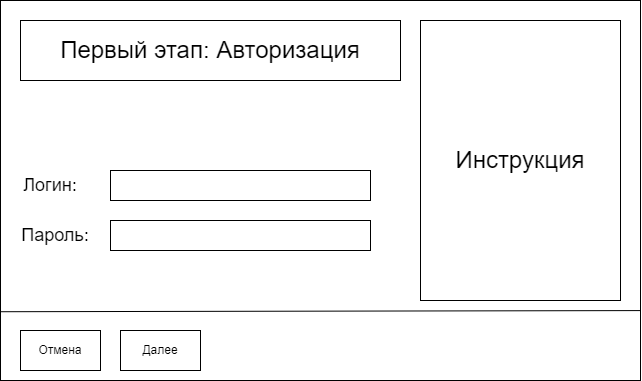


Рисунок 11 – Форма запроса логина и пароля пользователя

На рисунке 12 представлен основной алгоритм первой итерации работы модуля.

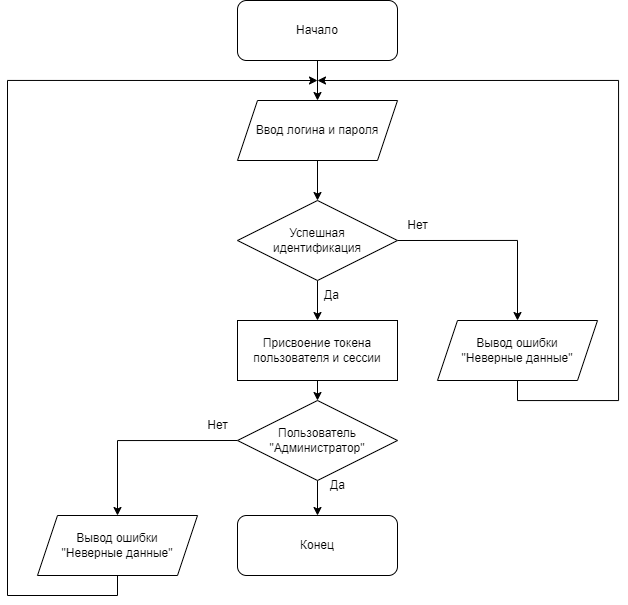


Рисунок 12 – Алгоритм получения токена пользователя

Запросы в ELMA3 осуществляются из оболочки модуля в ELMA365 путём создания запрос в формате JSON. Подробнее процесс и средства получения токена пользователя описаны в разделе программной реализации.

Также необходимо помнить, что данные должны быть верными и соответствовать пользователю с правами «Администратора». Администратор – пользователь, имеющий доступ ко всем разделам системы. Для предотвращения того или иного события предусмотрена валидация и обработка исключительных ситуаций со своими проверками и методами.

* + 1. Получение сущностей из ELMA3

Для успешного экспорта данных из разделов площадки ELMA3 в модуль ELMA365, процесс формирования структурированных массивов объектов является ключевым шагом. Каждый объект в массиве представляет определённый раздел и содержит свойства, соответствующие этому разделу. Такой подход позволяет организовать систематизированное хранение и передачу данных между системами.

В процессе экспорта данных модуль взаимодействует с внешним WEB API платформы ELMA3. Это API предоставляет доступ к различным сущностям и объектам системы, что обеспечивает гибкость и позволяет работать с данными независимо от выбранного решения.

Использование WEB API позволяет модулю получать и обрабатывать данные из разных разделов платформы ELMA3. Это означает, что модуль может получать информацию о задачах, процессах, пользователях и других сущностях, а также выполнять различные операции с этими данными.

WEB API платформы ELMA3 обеспечивает стандартизированный способ взаимодействия с системой, что упрощает разработку модулей и интеграцию с другими системами. Модуль может использовать API для получения актуальной информации, обновления данных или выполнения специфических действий в системе.

Процесс экспорта данных разделов площадки ELMA3 заключается в формировании структурированных массивов объектов, каждый из которых отвечает за свой раздел и имеет присущие ему свойства.

На рисунке 13 представлен алгоритм получения элементов из ELMA3 и запись их в хранилище для дальнейшего формирования элементов в ELMA365.

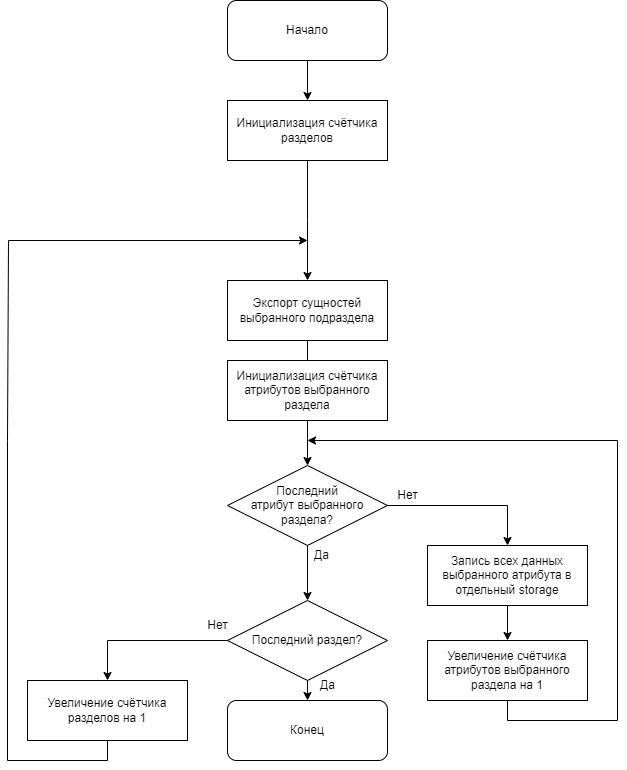


Рисунок 13 – Алгоритм получения элементов из ELMA3

Получение сущностей из ELMA3 также связано с формированием объектов JSON и дальнейшей передачей их из ELMA3 в ELMA365. На рисунке 14 представлен макет интерфейса загрузки объектов из оболочки платформы ELMA365. Процесс считается завершённым только тогда, когда все объекты всех заданных сущностей будут экспортированы в ELMA365, в промежуточное хранилище. Интерфейс шкалы переноса информации динамически отрисовывается на форме авторизации пользователя на платформе ELMA3. После успешного экспорта сущностей, выполняется переход на вторую форму модуля, где осуществляется ручной или автоматический маппинг типов сущностей.

Таким образом, процесс получения и экспорта сущностей из ELMA3 в ELMA365 включает формирование JSON-объектов, их загрузку в промежуточное хранилище, отслеживание прогресса передачи данных и дальнейший маппинг типов сущностей для обеспечения правильной обработки данных в модуле ELMA365.

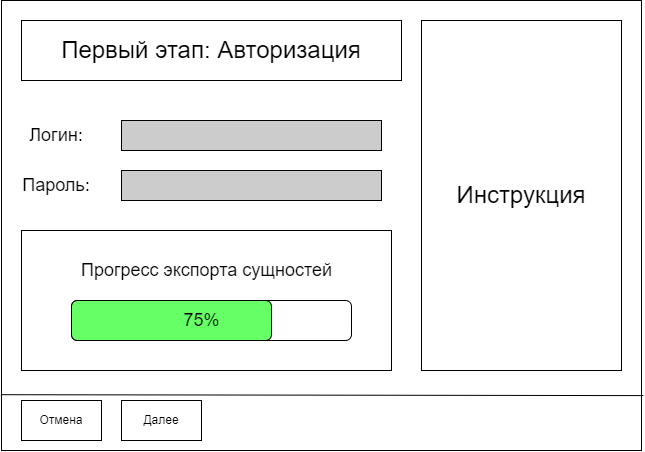


Рисунок 14 – Интерфейс экспорта данных

Помимо основного алгоритма загрузки данных с платформы, функционал программы подразумевает наличие шкалы, отображающей процесс экспорта данных с платформы ELMA3. Данная шкала подсчитывает процент экспортированных разделов в ELMA365, т.к. каждый запрос к системе ELMA3 экспортирует данные в виде массива объектов конкретного раздела. На рисунке 15 приведён алгоритм работы шкалы экспорта данных.

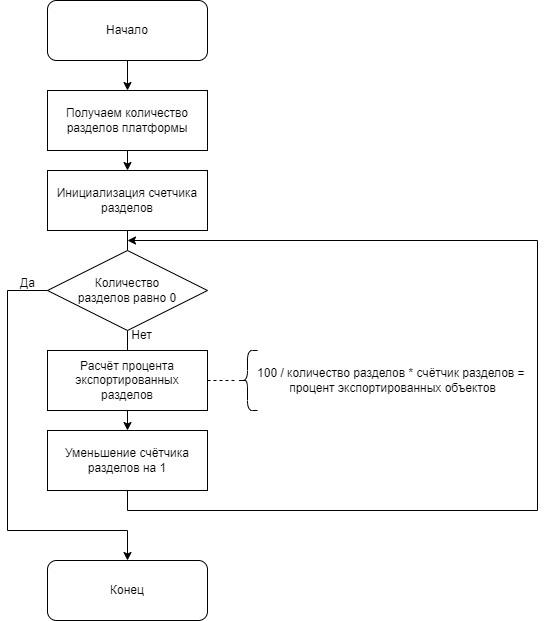


Рисунок 15 – Алгоритм заполнения шкалы прогресса

В ELMA3, как монолитной платформе, элементы хранятся в централизованной базе данных, которая является основным хранилищем информации системы. В виду этого, во время экспорта данных элементов, каждый из них представлен в виде объекта. На рисунке 16 приведён пример полученного объекта с платформы ELMA3, который в дальнейшем будет преобразован в объект на платформе ELMA365.



Рисунок 16 – Пример массива объектов с платформы ELMA3

Основное действие на данном этапе заключается в приведении полученных сущностей к каноничному для виду, где не будет системных атрибутов, системных разделов и прочего. Для этого был заведён отдельный интерфейс для представления сущностей с платформы ELMA3, рисунок 17.

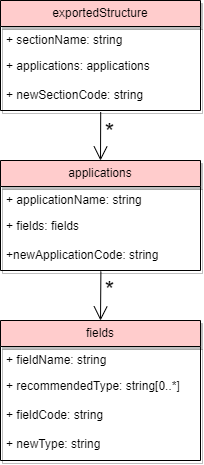


Рисунок 17 – Интерфейс для сущностей с платформы ELMA3

Интерфейс содержит наименование раздела, откуда была экспортирована сущность, его подраздел и список его атрибутов, которые и будут в дальнейшем сопоставляться с атрибутами сущностей платформы ELMA365. Поле recommendedType предназначено для сохранения типов данных их ELMA365 для каждого атрибута сущности с платформы ELMA3, чтобы пользователь мог понять: какие типы могут быть использованы для конкретного атрибута.

На рисунке 18 приведён алгоритм обработки сущностей с платформы ELMA3 и их атрибутов для выявления типов данных, которым соответствуют атрибуты сущностей. Его суть заключается в том, чтобы обработать все сущности платформы ELMA3, все их атрибуты и каждый атрибут проверить на соответствие каждому типу с платформы ELMA365. Данная возможность реализована при помощи описания типов ELMA365 и регулярных выражений для каждого из них, подробнее в разделе «Регулярные выражения для маппинга сущностей».

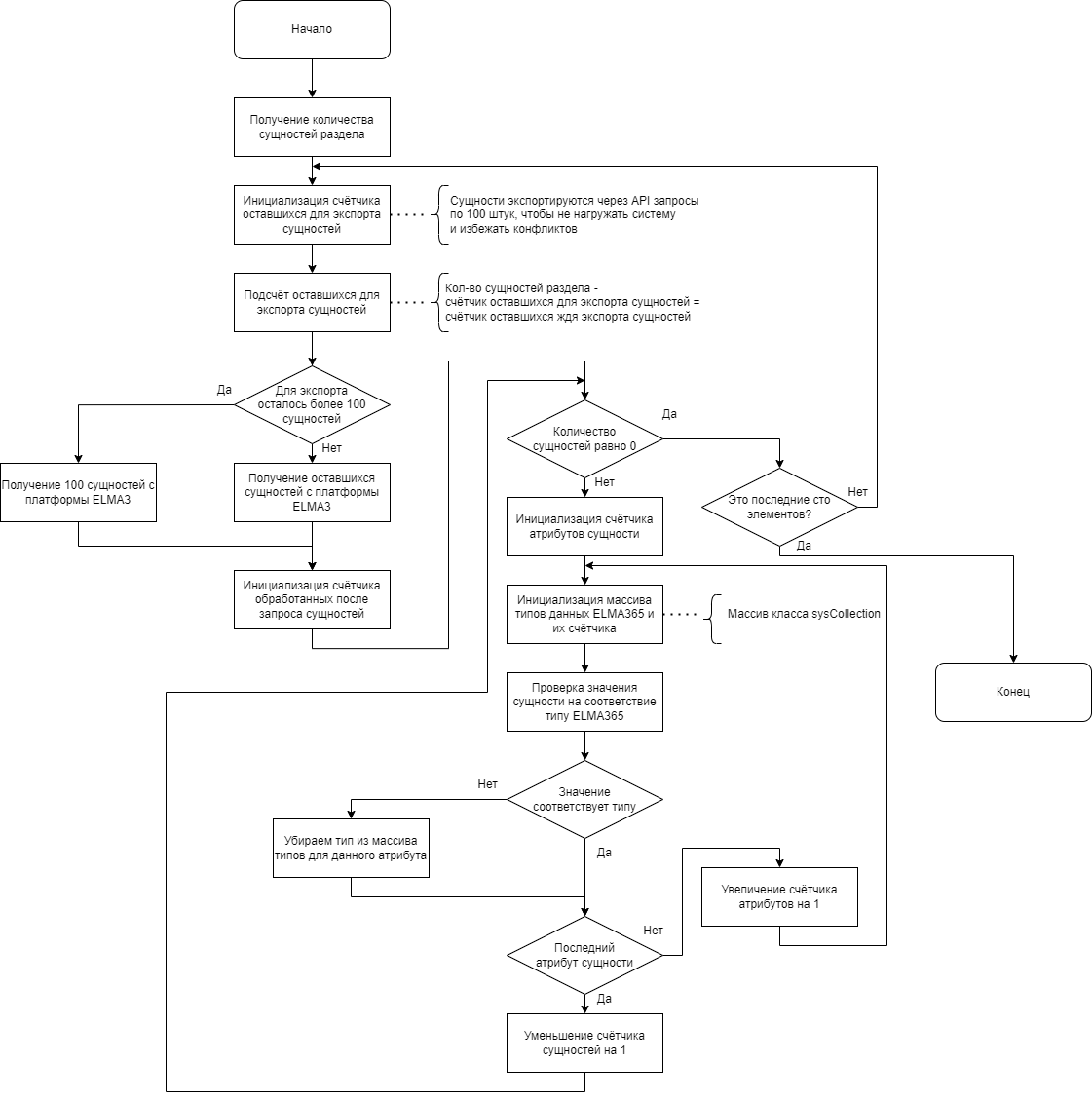


Рисунок 18 – Алгоритм присвоения рекомендованных типов

* + 1. Формирование элементов в ELMA365

Заключительной частью при экспорте данных с платформы ELMA3 на ELMA365 является формирование и создание элемента. Данный процесс осуществляется путём формирования объектов из полученных данных.

Для каждого типа сущностей написаны отдельные конструкторы и сделан маппинг. На рисунке 19 представлен обобщённый алгоритм для формирования элементов, суть которого заключается в следующем:

* Получение данных из исходной системы в формате, понятном для импорта в ELMA3;
* Разбор полученных данных на элементы, соответствующие типам сущностей в ELMA3;
* Формирование объектов для каждого типа сущностей с помощью соответствующих конструкторов;
* Маппинг данных из исходной системы на поля объектов с помощью соответствующих мапперов;
* Создание элемента в ELMA365 на основе сформированных объектов и заполнение его полями, соответствующими данным из исходной системы;
* Повторение шагов 2-5 для каждого элемента, который нужно создать в ELMA365;
* Проверка созданных элементов на наличие ошибок и отображение их пользователю.

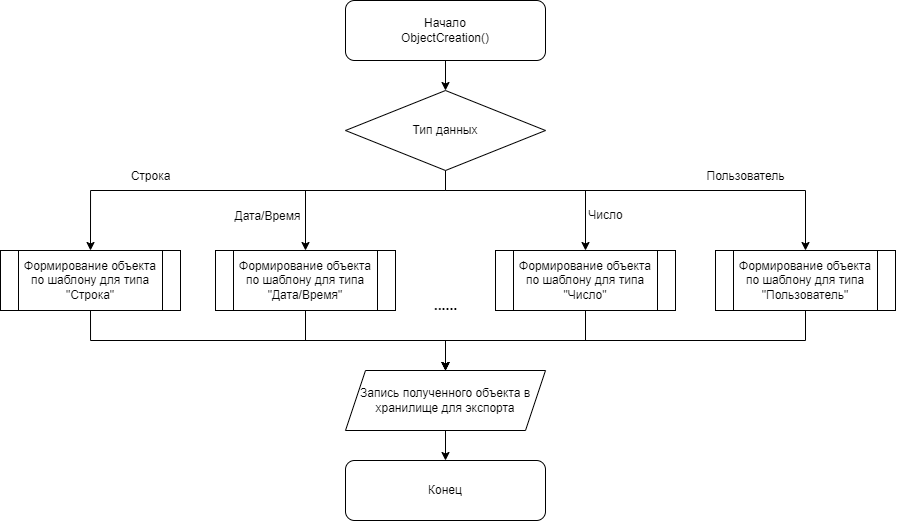


Рисунок 19 – Алгоритм формирования элементов разных типов

На рисунке 20 представлен пример конечного объекта, записываемого в базу данных платформы ELMA365.

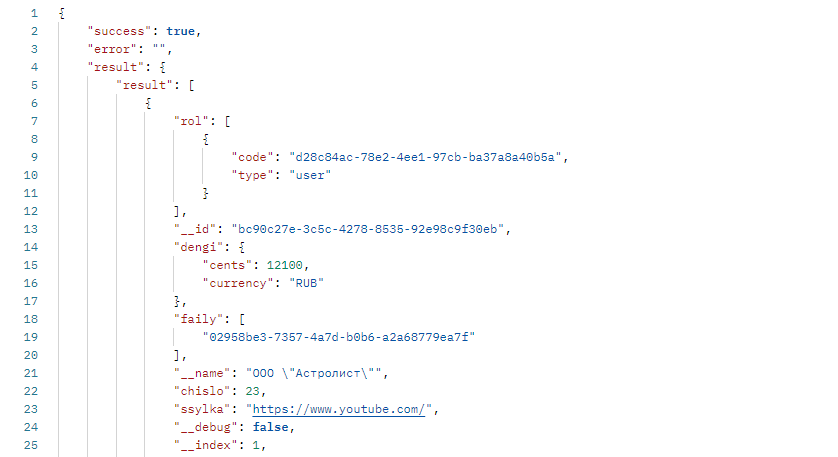


Рисунок 20 – Объект на платформе ELMA365

Для получения структуры платформы, к которой подключён модуль: разделов, приложений, атрибутов приложений, их типов и названий, - был заведён отдельный интерфейс, который отображает структуру системы и не содержит системных атрибутов и прочих свойств. Его диаграмма приведена на рисунке 21. Эта структура необходима для дальнейшего маппинга сущностей, т.к. система имеет множество ненужных пользователю атрибутов, классов и сущностей, которые, к тому же, не участвуют при их создании.

Первоначальная структура системы имеет системные разделы, которые не имеют элементов и, как следствие, не могут быть изменены пользователем, подразделы, которые скрыты для пользователя, и другие ненужные атрибуты. На рисунке 22 приведена первоначальная структура платформы, которую необходимо преобразовать в понятную для пользователя.

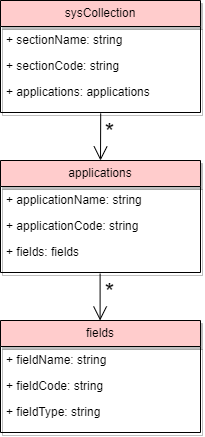


Рисунок 21 – Диаграмма интерфейса sysCollection

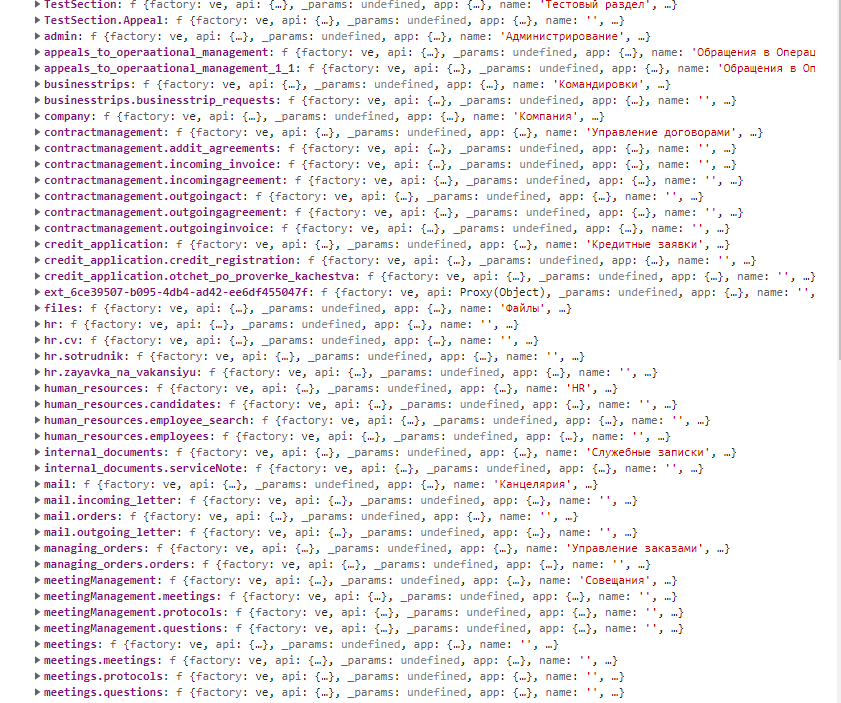


Рисунок 22 – Первоначальная структура платформы

В результате создания пользовательского интерфейса sysCollection и преобразований системной структуры с последующим удалением системных разделов, скрытых приложений, не редактируемых атрибутов подразделов удалось получить понятную пользователю структуру платформы, которая в дальнейшем используется для маппинга и дальнейшего создания элементов на платформе. На рисунке 23 изображён пример конечной структуры платформы. Данные поля использовались также для отображения в пользовательском интерфейсе, что облегчает порог вхождения пользователя.



Рисунок 23 – Пример структуры sysCollection

* 1. Алгоритмы функционирования для миграции данных

Система ELMA3 и ELMA365 включает в себя несколько разновидностей типов данных и сущностей. В частности, выделяются 2 главные разновидности типов объектов: простые и составные.

Как было сказано ранее – каждый тип сущности в ELMA3 имеет свой идентификатор, который, соответственно, может повторяться. К простым типам данных относятся:

* строка;
* текст;
* целое число;
* дробное число;
* дата / время;
* да / нет;
* выпадающий список;
* деньги;
* URL.

Отличительным признаком всех простых объектов является то, что они не содержат дополнительных вложенностей и ссылок на другие типы данных. Простые объекты представляют собой самодостаточные сущности, содержащие только простые атрибуты и значения, которые можно легко сериализовать и передавать между системами или хранить в базе данных. Это делает их удобными для обработки и анализа данных, а также позволяет эффективно использовать их в программной реализации и бизнес-логике модуля.

Для соотношения полей приложения из ELMA3 и полей в приложении в ELMA365 предусмотрен пользовательский интерфейс, который даёт возможность пользователю выбрать и соотнести тот или иной атрибут старого приложения с новым, чтобы не возникало конфликтов и неточностей при экспорте. Этот интерфейс позволяет визуально представить соотношение полей и установить связь между ними. Пользователь может просматривать список полей старого и нового приложений, выбирать соответствующие поля и устанавливать связи между ними.

Такой подход обеспечивает гибкость и контроль при соотнесении полей, а также минимизирует возможность ошибок и неточностей при экспорте данных. Пользователь может легко просмотреть все доступные поля, выбрать нужные соответствия и установить их взаимосвязь. Это упрощает процесс миграции данных и обеспечивает точность и надёжность в результате.

Таким образом, использование пользовательского интерфейса для соотношения полей приложений предоставляет удобный и интуитивно понятный способ управления данными и обеспечивает точность и контроль при переходе между различными версиями приложений.

Основными требованиями к пользователю при выборе и соотношении полей является равенство типа поля в ELMA3 и ELMA365. В противном случае пользователю будет запрещён выбор несоответствующего типа данных.

Под составными объектами понимаются приложения, которые имеют несколько вложенных атрибутов других типов.

При экспорте составных объектов важно учитывать связи между ними. Возникает ситуация, когда некоторые объекты, на которые ссылается элемент, ещё не были экспортированы или вообще не будут экспортированы. Для таких случаев был разработан алгоритм, который позволяет устанавливать связи между элементами после их создания.

На рисунке 26 представлена схема данного алгоритма. Он предусматривает следующий подход: при экспорте элемента, который ссылается на другие объекты, вместо прямой связи создается временная ссылка, которая указывает на идентификатор или другой уникальный идентификатор объекта, с которым должна быть установлена связь. После создания всех элементов происходит этап связывания, на котором происходит поиск и установка фактических ссылок между объектами на основе временных ссылок.

Этот алгоритм позволяет успешно обрабатывать сложные структуры с множеством взаимосвязей и устанавливать связи между объектами даже в случаях, когда некоторые из них ещё не доступны.

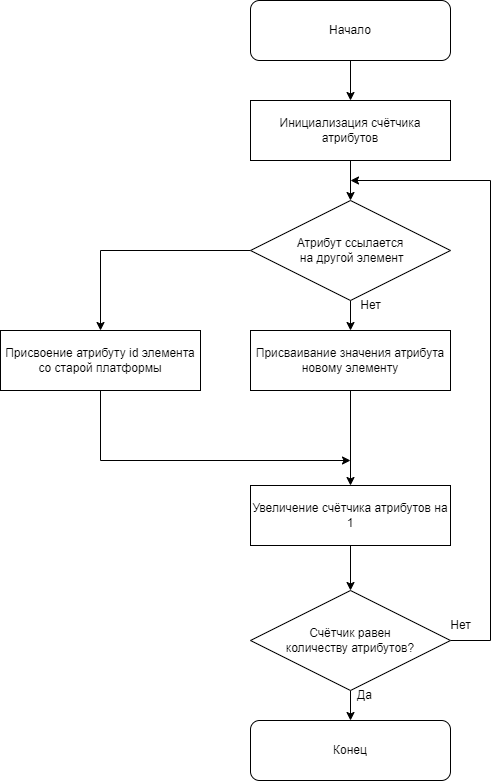


Рисунок 26 – Алгоритм работы с составными атрибутами

Отличительной чертой новых элементов является атрибут, хранящий признак элемента, относящий его к экспортированному, а не коренному элементу. В качестве данного атрибута выступает переменная типа «Строка», хранящая id элемента со старой платформы. Для однозначного идентифицирования элемента, как экспортированного, было добавлено графическое отображение со старой платформы. На рисунке 27 приведён пример такого отображения, явно говорящего о том, что элемент не является коренным и был экспортирован из ELMA3.

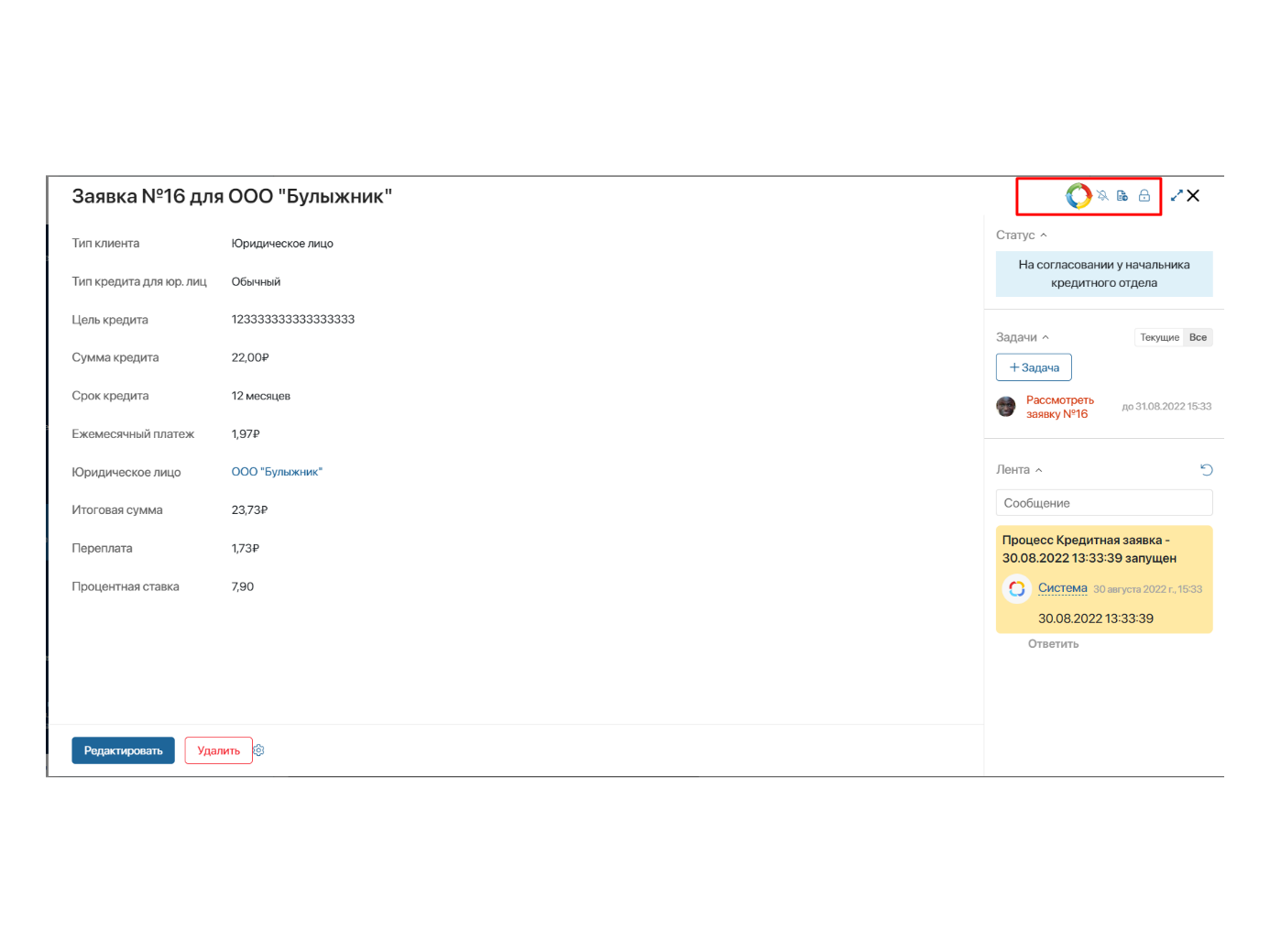


Рисунок 27 – Отображение признака экспортированного элемента

* 1. Регулярные выражения для маппинга сущностей

В качестве основного инструмента для валидации свойств сущностей при их соотношении было принято решение использовать регулярные выражения.

Перед тем как приступить к написанию регулярных выражений для каждого типа данных платформы ELMA365, необходимо проанализировать данные, определить ограничения и шаблоны.

Процесс написания регулярных выражений для каждого типа сущностей состоит из нескольких этапов:

* определить доступные типы данных на платформе ELMA3. На этом этапе необходимо проанализировать простые и составные типы данных на платформе ELMA3: какой набор символов входит в них, их структуру и значение в системе. Для наглядности была разработана диаграмма классов, отражающая структуру сущностей системы и их связи. Классовая диаграмма сущностей платформы ELMA3 изображена на рисунке 28;

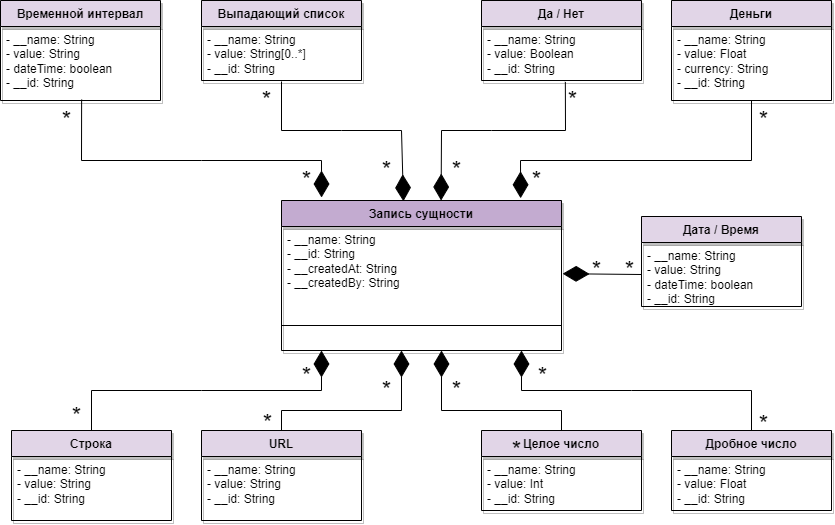


Рисунок 28 – Диаграмма классов сущностей платформы ELMA3

* определить доступные типы сущностей платформы ELMA365. Одним из самых важных моментов является определить список доступных на платформе-преемнике типов данных. В дальнейшем именно под эти типы будут соотноситься типы сущностей платформы ELMA3. Особенностью платформы ELMA365 является то, что она имеет большее количество новых типов, которые способны более конкретно описать некоторые данные. Для большего понимания была составлена диаграмма классов, отражающая типы сущностей платформы ELMA365, изображённая на рисунке 29. Благодаря наличию этих диаграмм можно увидеть особенности в организации типов данных в двух системах, которые необходимы для будущего маппинга сущностей, описанного в разделе программной реализации;

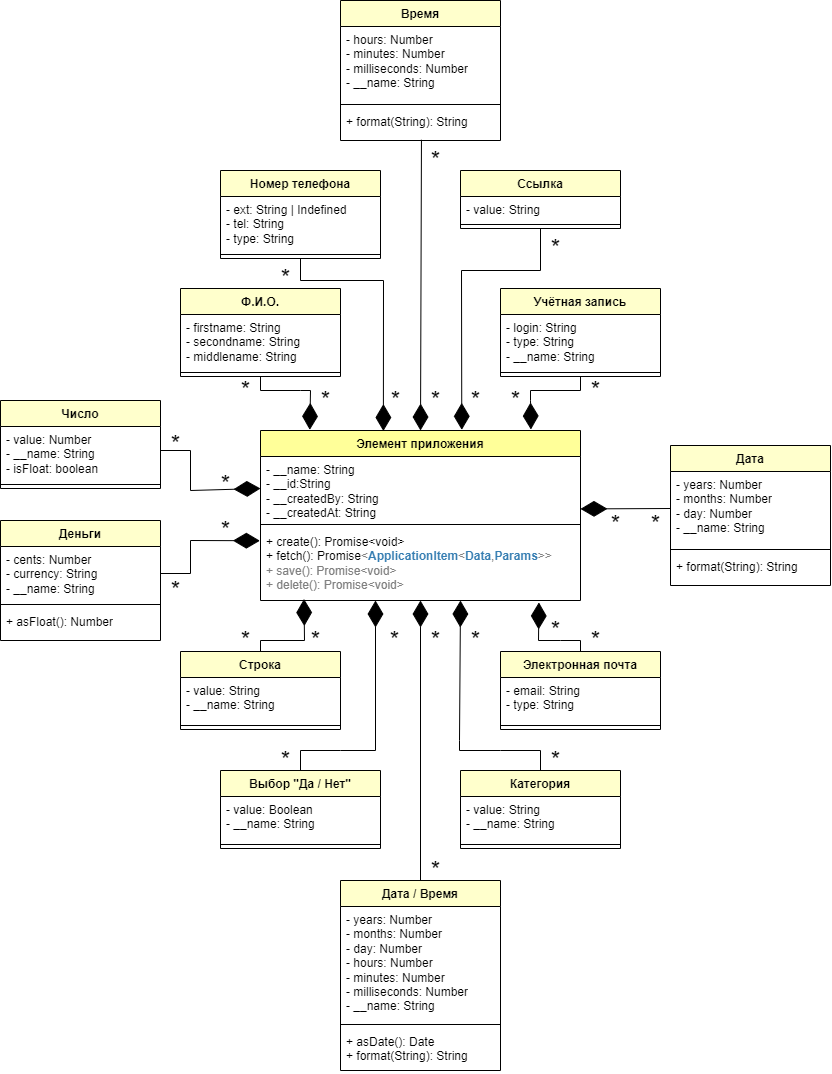


Рисунок 29 – Диаграмма классов сущностей платформы ELMA365

* подготовить регулярные выражения для каждого типа данных платформы ELMA3 и интерпретировать их в соответствии с типами на платформе ELMA365. Учитывая, что объекты системы передаются в модуль посредством API-запросов в виде строковых значений, подробнее в разделе программной реализации, возникла потребность в разборе и присвоении соответствующих типов через регулярные выражения. Далее описан процесс написания выражений для типов данных.

Для описания типов платформы ELMA365 был создан класс, описывающий особенности каждого типа. Диаграмма класса приведена на рисунке 30. В атрибуте «name» хранится название класса, атрибут «regularExpression» описывает допустимый набор символов и последовательностей символов для каждого типа путём регулярных выражений и атрибут «weight» задаёт вес каждого типа, который, в большинстве случаем, полезен как для ручного режима сопоставления типов, так и для автоматического. Веса помогают выбрать наиболее подходящий для массива атрибутов какого-либо раздела платформы ELMA3 тип данных наиболее подходящий в платформе ELMA365. Далее будут приведены примеры.

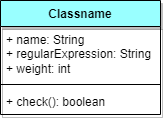


Рисунок 30 – Диаграмма класса сопоставления типов ELMA3 с типами ELMA365

Далее, в таблицах 1-10, приведены данные для каждого типа сущностей платформы ELMA365: возможный аналог типа из ELMA3, пример характерной для типа строки и регулярное выражение, задающее содержимое типа.

Таблица 1 – Данные для типа строка

|  |  |
| --- | --- |
| Тип ELMA365 | Строка |
| Пример строки | «Доброе утро!!!» |
| Регулярное выражение | Отсутствует |
| Особенности | Данный тип является стандартным для каждого типа данных, т.к. изначально все данные приходят в виде строки. Строка имеет наименьший приоритет (вес) и в автоматическом режиме вряд ли будет сопоставлена с другим типом, если имеется соответствие с любым другим типом. |

Таблица 2 – Данные для типа число (1)

|  |  |
| --- | --- |
| Тип ELMA365 | Целое число |
| Пример строки | «123456789» |
| Регулярное выражение | /^(?!0\d)\d{1,32}$/ |
| Особенности | На вход принимается любое целое число, которое не превышает 32 символов. Если число начинается с 0, то длина строки не должна превышать 1 символ. Это необходимо для валидации входных параметров типа «Число». |

Таблица 3 – Данные для типа число (2)

|  |  |
| --- | --- |
| Тип ELMA365 | Дробное число |
| Пример строки | «123.45678» |
| Регулярное выражение | /^(?!0\d)\d{1,32}(\.\d{1,8})?$/ |
| Особенности | На вход принимается любое дробное число, целая часть которого отделена точкой. Целая часть не должна превышать 32 символа, а дробная – 8. Если целая часть начинается с 0, то длина строки не должна превышать 1 символ. |

Таблица 4 – Данные для типа «Выбор Да /Нет»

|  |  |
| --- | --- |
| Тип ELMA365 | Выбор «Да / Нет» |
| Пример строки | «false» |
| Регулярное выражение | /^(true|false)$/ |
| Особенности | На вход принимается строка, содержащая значение «true» или «false». Пробелы не допускаются и входящая строка должна быть приведена к нижнему регистру, иначе это тип «Строка». |

Таблица 5 – Данные для типа «Дата / Время»

|  |  |
| --- | --- |
| Тип ELMA365 | Дата / Время |
| Пример строки | «2022-05-18T12:30:00.000Z» |
| Регулярное выражение | /^\d{4}-(0[1-9]|1[0-2])-([0-2][1-9]|3[01])T([01][0-9]|2[0-3]):([0-5][0-9]):([0-5][0-9])\.\d{3}Z$/ |
| Особенности | Учитывая то, что данные приходят в виде объектов через API запросы, они имеют свою особенную структуру. Так тип «Дата / Время» с платформы ELMA3 пересылается в формате «1970-01-01T00:30:00.000Z». проверка осуществляется на соответствие заданному формату, корректным временным диапазонам и длине чисел. Для остальных типов «Дата» и «Время» данные приходят в таком же формате. |

Таблица 6 – Данные для типа «Дата»

|  |  |
| --- | --- |
| Тип ELMA365 | Дата |
| Пример строки | «2022-12-31T00:00:00.000Z» |
| Регулярное выражение | /^(\d{4})-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])T00:00:00\.000Z$/ |
| Особенности | Данные типа «Дата / Время» с платформы приходят в общем формате, описанном для типа «Дата / Время». В таком случае для данного типа выполняется проверка на нулевые значения полей времени. Строка соответствует как типу «Дата / Время», так и типу «Дата», но приоритет типа «Дата» выше. |

Таблица 7 – Данные для типа «Время»

|  |  |
| --- | --- |
| Тип ELMA365 | Время |
| Пример строки | «1970-01-01T12:30:00.000Z» |
| Регулярное выражение | /^1970-01-01T(0[0-9]|1[0-9]|2[0-3]):([0-5][0-9]):([0-5][0-9]).000Z$/ |
| Особенности | Для данного типа характерны особенности вышеописанных типов «Дата / Время» и «Дата». Отличительной чертой является то, что при указании только времени, дата всегда приходит в запросе в виде константы равной «1970-01-01». |

Таблица 8 – Данные для типа «Электронная почта»

|  |  |
| --- | --- |
| Тип ELMA365 | Электронная почта |
| Пример строки | «example@example.com» |
| Регулярное выражение | /^[^\s@]+@[^\s@]+\.[^\s@]+$/ |
| Особенности | Данный тип характеризуется наличием символа «@» после адреса и наличием домена. Таким образом, регулярное выражение проверяет, что строка соответствует паттерну "что-то@что-то.что-то", где "что-то" - любая непустая последовательность символов, кроме пробела или символа "@". |

Таблица 9 – Данные для типа «Учётная запись»

|  |  |
| --- | --- |
| Тип ELMA365 | Учётная запись |
| Пример строки | «@example» |
| Регулярное выражение | /@\S+/ |
| Особенности | Учётная запись должна начинаться с символа «@» и не иметь пробельных символов. Данный тип будет иметь более высокий приоритет, чем стркоа. |

Таблица 10 – Данные для типа «Деньги»

|  |  |
| --- | --- |
| Тип ELMA365 | Деньги |
| Пример строки | «1000RUB» |
| Регулярное выражение | (\d+)RUB |
| Особенности | Данные типа «Деньги» приходят в формате объекта, который в дальнейшем преобразуется в строку типа «сумма валюта». Число – это «копейки», которые в дальнейшем в конструкторе преобразуются в рубли и копейки путём деления на 100. |

Таблица 11 – Данные для типа «Номер телефона»

|  |  |
| --- | --- |
| Тип ELMA365 | Номер телефона |
| Пример строки | «+7-123-456-78-90», «+71234567890», «81234567890» |
| Регулярное выражение | /^(\+7|7|8)[-]?\d{3}[-]?\d{3}[-]?\d{2}[-]?\d{2}$/ |
| Особенности | На вход принимается строка, которая может начинаться на знак «+», 7 или 8. Далее следует 10 цифр. Также учитывается возможное наличие символов-разделителей в виде дефиса «-». |

Таблица 12 – Данные для типа «Ф.И.О.»

|  |  |
| --- | --- |
| Тип ELMA365 | Ф.И.О. |
| Пример строки | «Иванов Иван Иванович», «Петров Пётр» |
| Регулярное выражение | ^[А-ЯЁ][а-яё]+(\s[А-ЯЁ][а-яё]+)?\s[А-ЯЁ][а-яё]+$ |
| Особенности | Данные должны быть представлены тремя словами, начинающимися с верхнего регистра. Отчество может отсутствовать. Имена должны быть представлены на русском языке. |

Таблица 13 – Данные для типа «Ссылка»

|  |  |
| --- | --- |
| Тип ELMA365 | Ссылка |
| Пример строки | «https://www.example.com», «ftp://ftp.example.com/files» |
| Регулярное выражение | ^(https?|ftp):\/\/[^\s/$.?#].[^\s]\*$ |
| Особенности | Отличительной особенностью данного типа является наличие характерного «http://», «https://» или «ftp://» за которым следует доменное имя или IP-адрес. Затем могут следовать путь и параметры запроса. Знак ^ указывает на начало строки, а $ - на конец строки, чтобы убедиться, что вся строка соответствует этому шаблону. |

Вывод

В данном разделе были рассмотрены основные алгоритмы и приёмы, которые использовались при создании модуля. Был спроектирован интерфейс для взаимодействия с пользователем. Удалось чётко понять: какие типы есть на обоих платформах и сопоставить их между друг другом.

Были спроектированы основные подходы к валидации типов данных при их сопоставлении в пользовательском интерфейсе. Описан способ применения регулярных выражений для модуля: определён набор символов, соответствующий каждому типу данных, установлены алгоритм проверки свойств сущностей на соответствие этим выражениям и приведены примеры их работы.

# ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

В данном разделе описаны выбранные инструменты для реализации модуля (язык программирования, библиотеки), разработан графический интерфейс пользователя, а также предоставлены детали реализации и визуализации реализации применённых ранее алгоритмов.

* 1. Выбор инструментов разработки

Т.к. было принято решение реализовать миграцию данных для BPM платформ с разными архитектурами путём создания модуля, то и языки программирования необходимо было выбирать из числа тех, которые поддерживаются платформой. Помимо вышесказанного, необходимо реализовать пользовательский интерфейс, который можно, при возможности, наращивать и добавлять новый функционал без особого изменения остальной структуры кода.

Среди доступных языков реализации логики имеются:

* JavaScript. JavaScript является широко используемым языком программирования, который поддерживается практически всеми современными браузерами. Он идеально подходит для разработки пользовательского интерфейса (UI) и взаимодействия с веб-страницами. JavaScript обеспечивает динамическое изменение содержимого страницы, обработку событий и взаимодействие с сервером.
* TypeScript. TypeScript является надмножеством JavaScript, которое добавляет статическую типизацию и другие возможности, улучшающие разработку и поддержку кода. Он компилируется в JavaScript и обладает расширенными возможностями для структурирования и поддержки проекта. TypeScript позволяет обнаруживать ошибки на этапе разработки и упрощает работу с большими кодовыми базами.
* C#. C# является языком программирования, разработанным Microsoft, и часто используется для создания приложений на платформе .NET. Он обладает широким набором инструментов и библиотек для разработки мощных и масштабируемых приложений. C# подходит для реализации бизнес-логики и взаимодействия с базой данных.
* Golang. Golang, также известный как Go, является языком программирования, разработанным Google, с фокусом на производительность, простоту и масштабируемость. Он подходит для создания эффективных и надёжных системных приложений. Golang имеет нативную поддержку параллельных вычислений и хорошую производительность.
* C++. C++ - это язык программирования общего назначения, который обеспечивает высокую производительность и низкоуровневый доступ к системным ресурсам. Он широко используется для разработки системного и высокопроизводительного программного обеспечения. C++ предлагает богатые возможности, но может быть более сложным в освоении и требовать более тщательного управления памятью.

Рассмотрев основные языки программирования, приведённые выше, был проведён анализ и учтены требования к функциональности, производительности и доступности инструментов разработки конкретного языка.

Первым требованием к реализации является реализация модуля, а не отдельно взятой программы по ряду причин:

* разделение функциональности. Модуль позволяет разделить функциональность проекта на отдельные части. Это способствует лучшей организации и структурированию кода, делает его более понятным и легко поддерживаемым. Каждый модуль может быть ответственным за определённую задачу или функцию, что упрощает разработку и обслуживание проекта;
* модульность и повторное использование. Модульность позволяет создавать независимые модули, которые можно повторно использовать в разных проектах или внедрять в другие модули. Это сокращает время разработки и способствует повышению производительности. Учитывая архитектуру платформы – модуль можно внедрить на площадку любого клиента без особых усилий;
* гибкость и масштабируемость. Модульный подход обеспечивает гибкость и масштабируемость проекта. Модули могут быть легко добавлены или удалены, а новый функционал может быть внедрён без влияния на другие модули или функции проекта. Это позволяет проекту быть более адаптивным к изменениям требований и обеспечивает более простое добавление нового функционала;
* тестирование и отладка. Модульный подход облегчает тестирование и отладку проекта. Модули могут быть независимо тестированы, что облегчает обнаружение и исправление ошибок. Также модули могут быть заменены заглушками или имитациями для более эффективного тестирования;
* расширяемость. Модули могут быть легко расширены для добавления нового функционала или взаимодействия с другими системами или модулями. Это позволяет проекту быть более гибким и способным адаптироваться к новым требованиям;
* нативный доступ к данным. Благодаря прямому подключению модуля к платформе, данные берутся напрямую, не через запрос к серверу, что значительно ускоряет процесс обработки информации и экономит значительное количество времени.

В целом, модульный подход позволяет создавать более организованные, гибкие и расширяемые проекты.

На следующем этапе необходимо было выбрать родительскую платформу для модуля: ELMA3 или ELMA365. Обе эти платформы имеют свои точки расширения благодаря модульной структуре. Был проведён анализ возможностей платформ и выбран наиболее оптимальный вариант.

В виду наличия значительных преимуществ было принято решение реализовывать модуль для платформы ELMA365. В числе преимуществ модуля для ELMA365 над ELMA3 следующее:

Гибкая настройка модуля. Платформа ELMA365 имеет гибкую настройку модуля: возможность написания методов API, создание виджетов на языке JavaScript, настройка точек расширения, создание BPMn –процессов, для настройки бизнес-логики. Пример интерфейса настроек модуля на платформе ELMA365 представлена на рисунке 31.

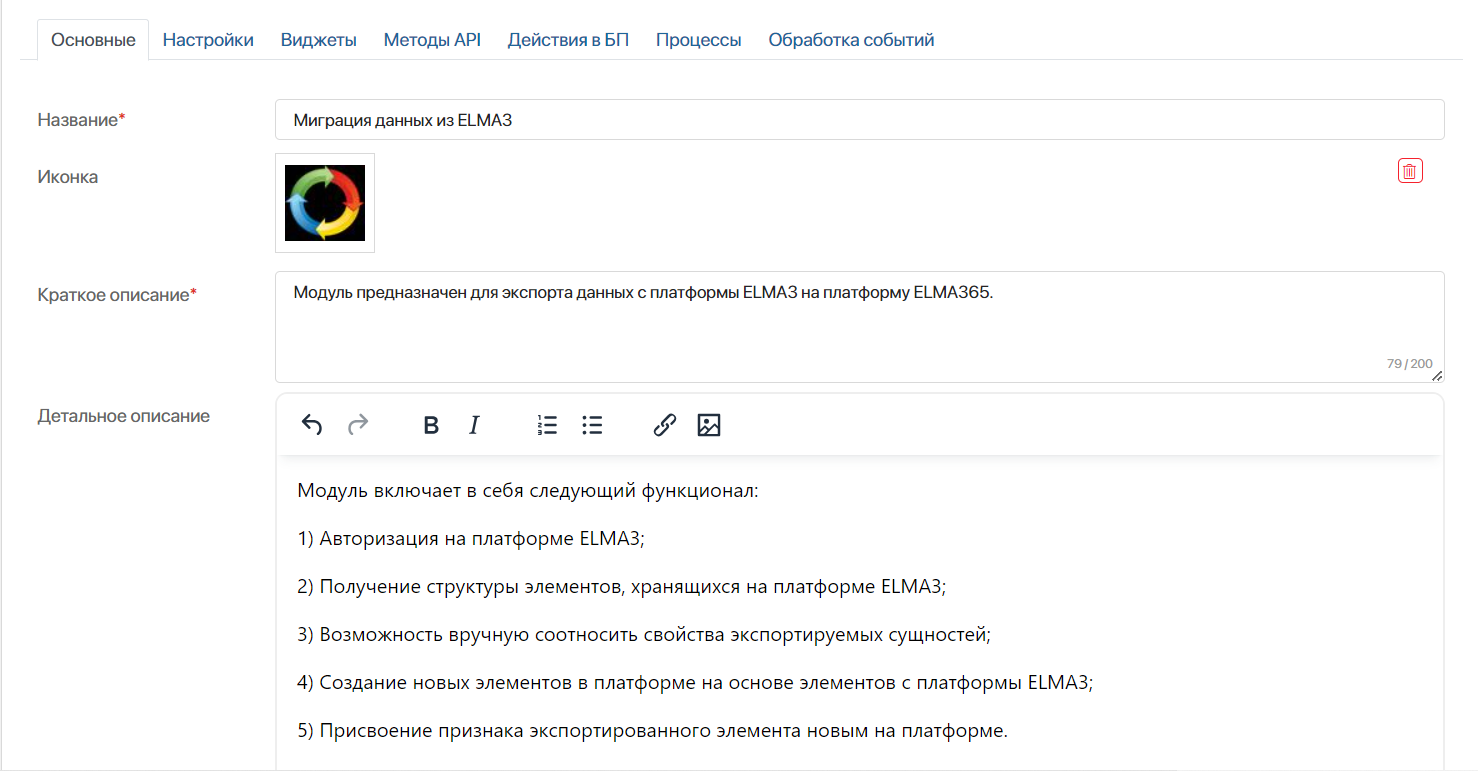


Рисунок 31 – Интерфейс настроек модуля ELMA365

Для реализации интерфейса было принято решение использовать динамическую отрисовку при помощи задавания разметки JavaScript. Данная реализация была выбрана по ряду причин.

Во-первых, использование JavaScript для создания и изменения разметки позволяет нам гибко и динамически управлять интерфейсом на основе событий и данных. Мы можем легко реагировать на действия пользователя, изменять содержимое и структуру элементов интерфейса в реальном времени.

Во-вторых, такой подход позволяет нам полностью разделить логику приложения и представление. Мы можем сосредоточиться на написании логики и бизнес-логики приложения в JavaScript, в то время как разметка HTML остаётся чистой и содержит только статическую структуру интерфейса. Это способствует лучшей организации кода и повышает его поддерживаемость.

Также, при использовании JavaScript для задания разметки, мы получаем более гибкое управление динамическими данными. Мы можем получать данные из сервера, обрабатывать их и вставлять в разметку, обновлять интерфейс при изменении данных без необходимости перезагрузки страницы.

Кроме того, использование JavaScript для отрисовки интерфейса позволяет нам легко создавать переиспользуемые компоненты и модули, что ускоряет разработку и облегчает поддержку проекта.

В итоге, выбор динамической отрисовки интерфейса при помощи задавания разметки в JavaScript даёт нам больше гибкости, улучшает организацию кода, облегчает работу с динамическими данными и повышает поддерживаемость проекта. Этот подход является современным и эффективным способом разработки веб-приложений.

* 1. Реализация этапов миграции данных

В данном разделе описаны основные инструменты и средства программирования, которые использовались для реализации основных этапов миграции данных с платформы ELMA3 на платформу ELMA365.

* + 1. Получение доступа к площадке для экспорта

Процесс получения доступа к площадке преемнику осуществляется путём получения токена пользователя платформы ELMA3. Как было описано в разделе разработки структуры модуля. Пользователю необходимо ввести действительные логин и пароль пользователя от площадки ELMA3. Затем эти данные передаются через REST API на стенд клиента. Также подразумевается, что URL площадки изначально введён в контекстную переменную модуля, которая располагается в настройках, рисунок 32.

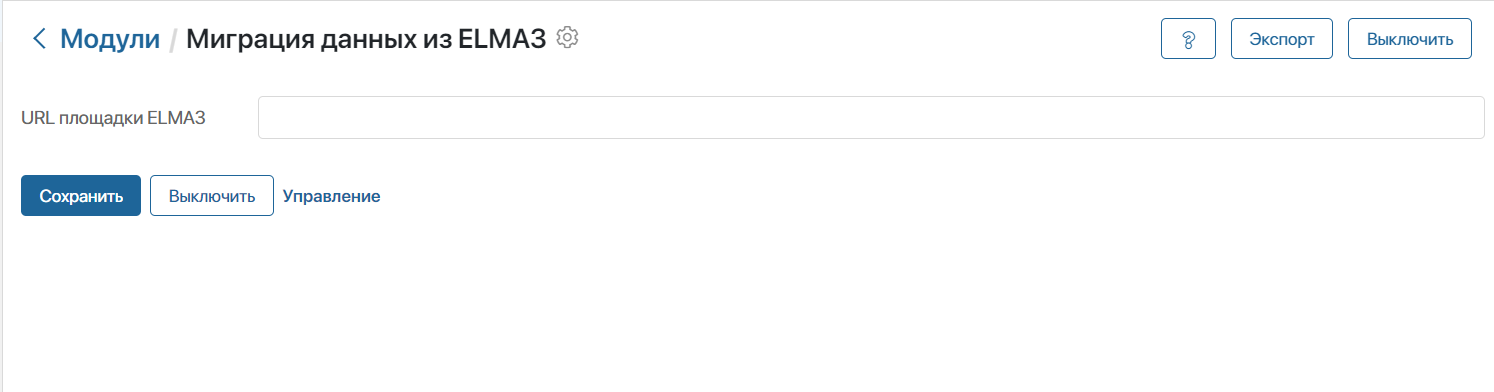


Рисунок 32 – Поле URL площадки в настройках модуля

Дальнейшее получение токена пользователя и авторизация осуществляется путём взаимодействия с сервисом IAutorization. Данный сервис имеет свои методы и запросы. Так для текущего модуля был использован метод POST с Uri /LoginWith для авторизации и получения токена пользователя по логину и паролю.

После отправки запроса на авторизацию, сервис IAutorization проверяет предоставленные учётные данные пользователя. Если логин и пароль верны, сервис генерирует и возвращает уникальный токен пользователя. Этот токен представляет собой специальную строку, которая используется для идентификации и аутентификации пользователя в дальнейших запросах к системе.

Запрос направляется по адресу: http://bpm-demo.elma-bpm.ru/API/REST/Authorization/LoginWith?username={USERNAME}, где в контекст URL выполняется подстановка логина пользователя. В результате получаем ответ в формате JSON, рисунок 33.

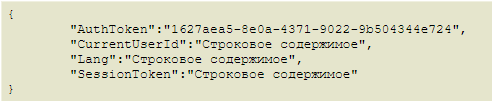


Рисунок 33 – Пример ответа сервиса IAutorization

После получения ответа и токена идёт обращение через REST API к каждому разделу платформы с целью проверки прав доступа. Если все успешно, то начинается процесс экспорта сущностей платформы ELMA3.

* + 1. Получение структуры элементов платформы

Второй этап работы модуля заключается в получении сущностей платформы ELMA3. За получение записей сущности отвечает сервис IBatchOperationService. Данный сервис имеет свои методы и запросы. В разработке модуля использовался HTTP метод POST, позволяющий получить элементы определённого раздела для их дальнейшей обработки. Элементы передаются в размере 100 штук. Это сделано с целью оптимизации процесса и уменьшения нагрузки на сервер.

При запросе на получение сущностей, клиентское приложение отправляет запрос к сервису IBatchOperationService с указанием нужного раздела и количества элементов для получения. Сервис IBatchOperationService обрабатывает запрос и возвращает запрошенные элементы в формате JSON. Далее при помощи методов JavaScript для работы с объектами типа JSON происходит парсинг этих данных и приведение объектов к нужному виду, который был описан в разделе разработки структуры модуля.

На рисунке 34 приведён пример ответа сервера в формате JSON.

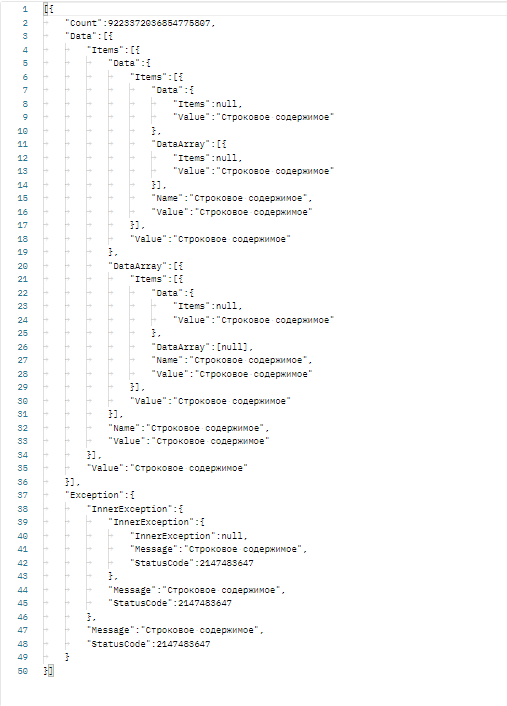


Рисунок 34 – Ответ сервера в формате JSON

Использование метода POST и разделение элементов на пакеты помогает оптимизировать процесс получения данных и улучшить производительность модуля. Это позволяет эффективно обрабатывать большие объёмы данных и управлять загрузкой на сервер.

Таким образом, второй этап работы модуля заключается в получении сущностей платформы ELMA3 с помощью сервиса IBatchOperationService. Это позволяет получить необходимые данные для дальнейшей обработки и использования в рамках функциональности модуля.

* + 1. Маппинг сущностей платформы

Для маппинга сущностей был разработан специальный интерфейс, позволяющий пользователю вручную или автоматически соотносить атрибуты сущностей различной вложенности системы. На первоначальном этапе пользователь выбирает разделы с элементами на платформе ELMA3 и соотносит их с разделами на платформе ELMA365. Макет интерфейса представлен на рисунке 35.

Платформа ELMA365 имеет встроенный виджет, называющийся «Код». Его ключевой особенностью является то, что он поддерживает разметку HTML, стили CSS и логику JavaScript. Основной механизм маппинга сущностей, благодаря которому осуществляется подтягивание названий разделов и их атрибутов осуществляется в нём.

Благодаря тому, что модуль напрямую подключён к площадке платформы ELMA365, есть возможность обращаться напрямую к системным коллекциям платформы, которая у каждой компании своя.

Названия разделов и его приложений, а также название типов атрибутов этого приложения преобразуются в строку путём методов, написанных на встроенном в ELMA365 компиляторе TypeScript. Так для получения всей системной коллекции можно обратиться к объекту Global, который на платформе ELMA365 содержит наименования всех разделов и их приложений.

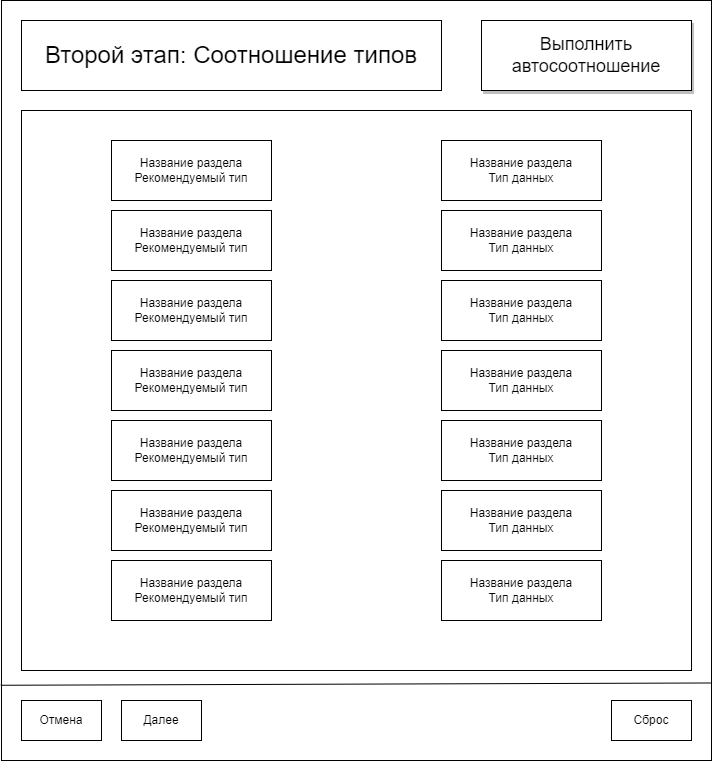


Рисунок 35 – Макет интерфейса для маппинга сущностей

Кроме ручного маппинга сущностей была реализована возможность автоматического маппинга при помощи нажатия на кнопку «Выполнить автосоотношение». В таком случае соотношение сущностей и их атрибутов выполняется при помощи весов, которые подробно описаны в разделе разработка структуры модуля.

* + 1. Создание элементов в ELMA365

Конечный этап заключается в переносе элементов непосредственно на платформу, к которой подключён модуль. Элементы записываются в базу данных PosgreSQL.

Благодаря наличию встроенной среды разработки и компилятора на базе языка программирования TypeScript, логика прописывается напрямую на платформе. Это необходимо для оптимизации и уменьшения времени обращения к платформе.

За создание элементов отвечают методы для работы с объектами встроенные в систему:

* create(). Позволяет создавать новую строку в базе данных в конкретной таблице и изменять её атрибуты;
* save(). Позволяет сохранять внесённые изменения в элемент при занесении новых значений.

Данные методы применяются для конструктора элементов. Под конструкторами понимается метод, динамически формирующий элементы платформы ELMA365 для каждого раздела и приложения этого раздела. Правила, по которым формируются эти элементы задаются на этапе маппинга и соотношения типов. Данный этап был описан ранее.

Динамическое формирование кода допускается в языке TypeScript путём использования метода eval(). Метод eval() позволяет выполнять произвольный код, переданный в виде строки, во время выполнения программы. Это дает возможность создавать и выполнять код динамически на основе определённых условий или входных данных.

Однако, необходимо быть осторожным при использовании метода eval(), так как он может представлять потенциальные уязвимости безопасности, особенно если входные данные поступают от ненадёжных источников. При динамическом формировании кода рекомендуется применять соответствующие меры безопасности, например, валидацию и фильтрацию входных данных.

Вместо метода eval(), в языке TypeScript также доступны другие способы динамического формирования кода, такие как использование шаблонных строк (template strings) и функций высшего порядка (higher-order functions). Эти подходы позволяют генерировать и комбинировать код на основе данных и шаблонов, обеспечивая гибкость и удобство в разработке.

Таким образом, при необходимости динамического формирования кода в TypeScript, можно использовать методы, такие как eval(), шаблонные строки и функции высшего порядка. Однако, необходимо соблюдать осторожность и применять соответствующие меры безопасности при работе с динамически сгенерированным кодом.

Также некоторые типы данных изначально необходимо привести к каноничному для системы ELMA365 виду. Для составных типов, таких как «Дата», «Дата / Время» и т.д., были написаны методы для их формирования, входными атрибутами которых являются строки, которые получаем модуль из платформы ELMA3 путём формирования API запросов, описанных в разделе получение структуры элементов платформы. Конструкторы приведены в приложении С.

Конструкторы элементов позволяют динамически создавать различные компоненты, такие как текстовые поля, кнопки, списки выбора и другие, в зависимости от заданных правил маппинга. Это позволяет модулю быть гибким и адаптироваться под разные разделы и приложения платформы ELMA365.

* 1. Апробация модуля

В данном разделе представлены полный список шагов, которые необходимо сделать пользователю для успешного подключения модуля к своей платформе и миграции данных со старой платформы.

Процесс состоит из следующих шагов:

1. Установка модуля к себе на платформу. На данном этапе пользователю необходимо самостоятельно загрузить модуль на платформу ELMA365 при помощи стандартных средств для импорта, приведённых на рисунке 36.

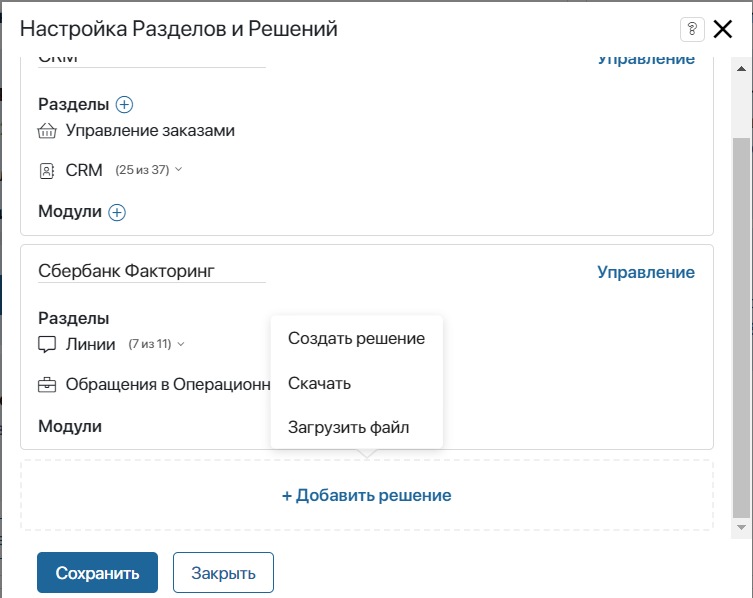


Рисунок 36 – Импорт модуля на платформу

1. Ввод URL платформы ELMA3. На данном этапе пользователю необходимо ввести URL платформы ELMA3, с которой будет осуществляться миграция данных на новую платформу ELMA365. Ввод осуществляется в поле контекстной переменной «URL стенда», расположенной в настройках модуля. Ранее этот этап описывался в разделе получения доступа к площадке для экспорта.
2. Инициализировать запуск интерфейса модуля. Для этого необходимо перейти на главную страницу платформы и нажать на кнопку с логотипом ELMA3, рисунок 37.

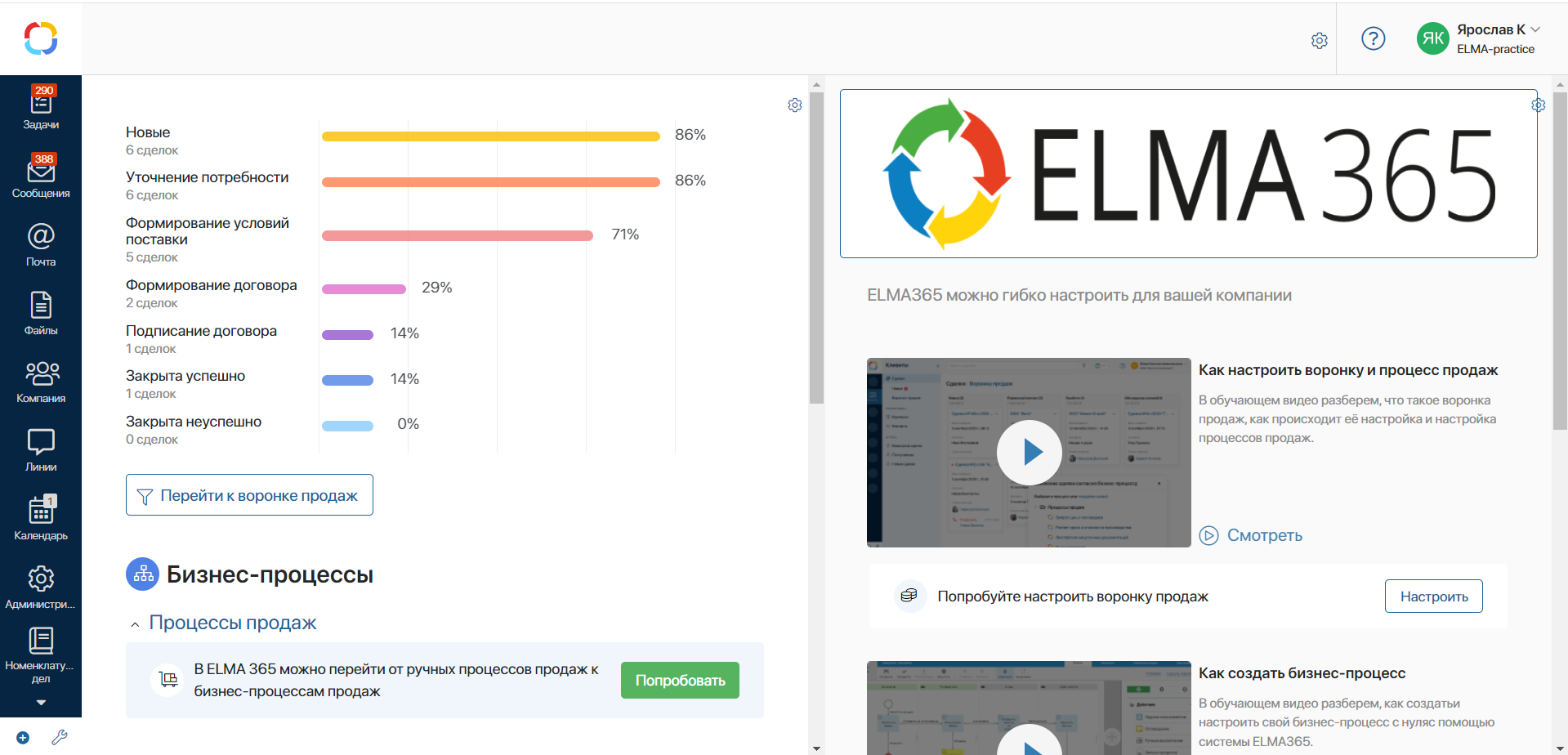


Рисунок 37 – Кнопка миграции данных на главной странице

1. Ввод логина и пароля пользователя с ELMA3. Текущий этап подразумевает ввод логина и пароля пользователя с правами администратора на платформе ELMA3. Для успешного получения токена необходимо пройти авторизацию и нажать на кнопку «Далее». Пример экранной формы приведён на рисунке 38.
2. Маппинг сущностей. На текущем этапу перед пользователем располагается окно со всеми разделами, подразделами и атрибутами с платформы ELMA3 и ELMA365, рисунок 39. Ему необходимо на основании рекомендаций, которые отображаются снизу блоков сущностей, сопоставить атрибуты объектов платформы ELMA3 и ELMA365. Нельзя сопоставить неподходящие по рекомендациям типы, рисунок 40. Также пользователь может нажать на кнопку «Автоматическое сопоставление», что подразумевает автоматический маппинг сущностей разных платформ на основании их весов. После успешного полного или частичного маппинга сущностей, необходимо нажать на кнопку «Далее».

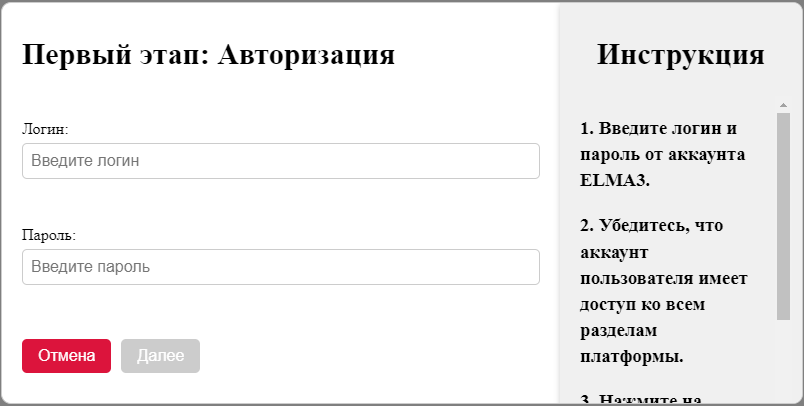


Рисунок 38 – Окно авторизации пользователя в ELMA3

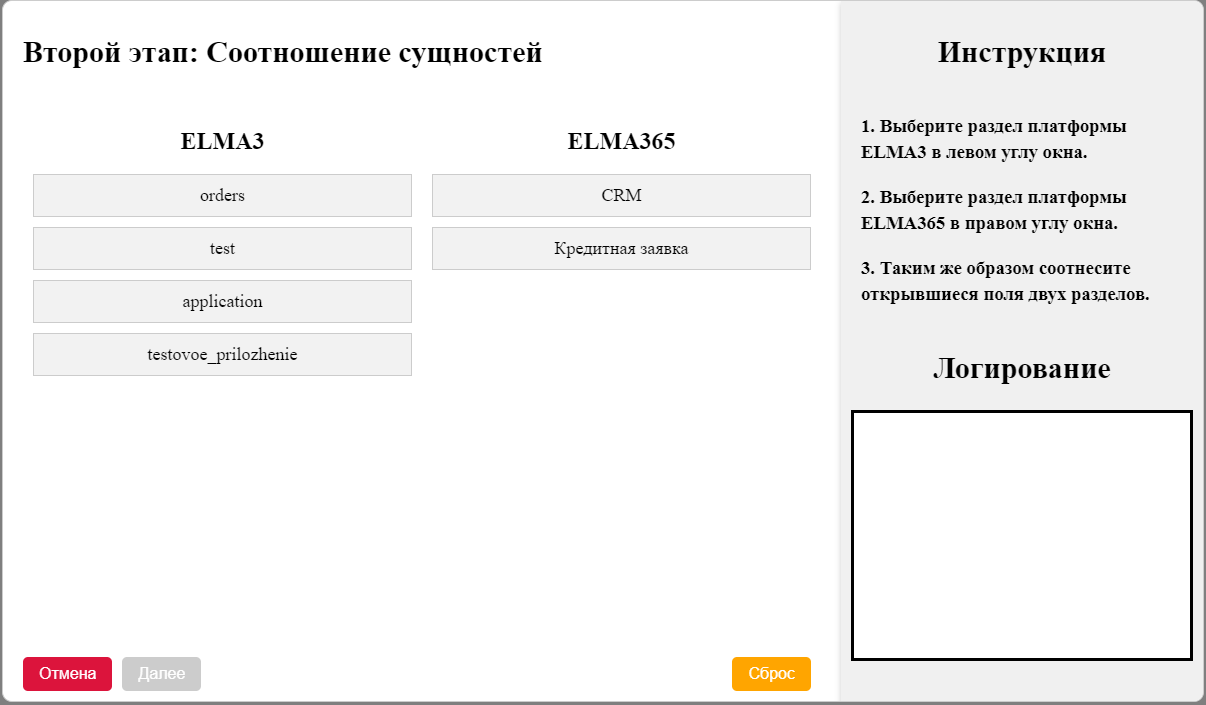


Рисунок 39 – Окно с разделами платформ

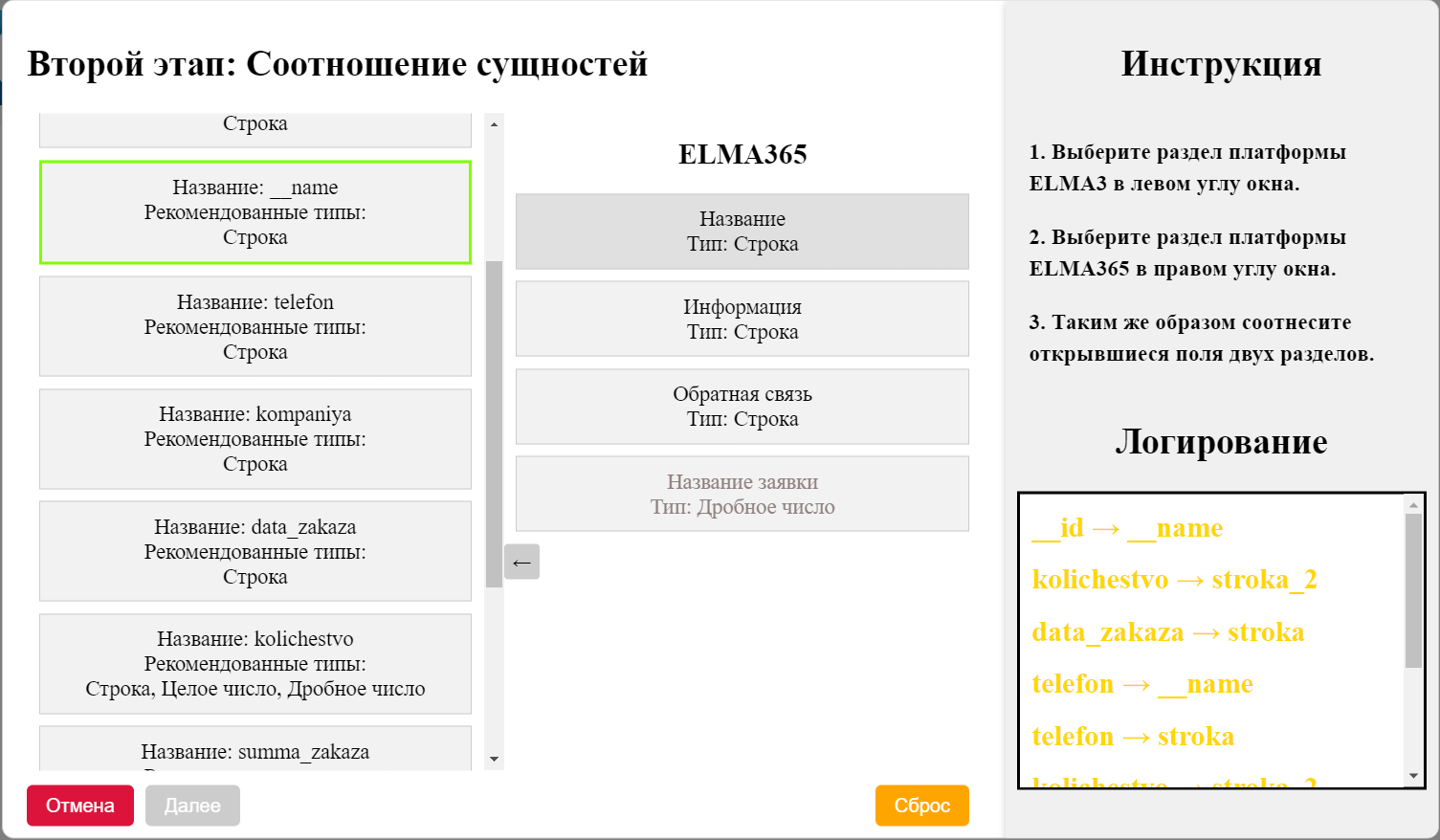


Рисунок 40 – Процесс выбора типа для атрибута

1. Создание элементов в ELMA365. После маппинга сущностей перед пользователем появляется окно, отображающее прогресс миграции текущих данных на платформу, т.е. заполнение её новыми элементами на основании данных со старой платформы, рисунок 41. После успешной миграции, окно закрывается.



Рисунок 41 – Окно прогресса экспорта сущностей на платформу ELMA365

Вывод

В данном разделе были рассмотрены основные инструменты и средства, используемые при создании модуля. Были рассмотрены аналоги и их плюсы и минусы.

Была проведена апробация модуля: показаны шаги, необходимые для запуска модуля и его дальнейшего использования.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта был проведён анализ предметной области и рассмотрены основные вопросы, касающиеся тематики разрабатываемого модуля. Затронут вопрос проблемы миграции данных в глобальном масштабе и проанализирована актуальность данной проблемы на текущий момент. Проблема миграции данных стала актуальной для многих современных компаний в силу постоянного совершенствования их программных решений.

После успешного проведения анализа предметной области были спроектированы основныеподходы к решению данной проблемы. Выяснилось, что модуль для миграции данных можно реализовать на нескольких платформах с применением разных инструментов и практик. В итоге, выбор пал на модуль на платформе ELMA365, что было обусловлено наличием более гибких инструментов для реализации интерфейса, прямого подключения к платформе-преемнику и более популярным языку для написания логики модуля.

В процессе работы над проектом было составлено расширенное техническое задание, которое имело следующие цели: описать требования, спецификации и функциональность проекта. В расширенном техническом задании были указаны минимальные системные требования для правильной работы модуля, определены требования к документации, а также указано основное назначение модуля и другие важные детали проекта.

Разработаны основные алгоритмы функционирования, которые использовались в модуле. Среди них: импорт объектов с платформы ELMA3, создание сущностей на платформе ELMA365, анализ данных импортируемых сущностей и т.п. Помимо вышеуказанных алгоритмов было принято ключевое решение: использовать регулярные выражения для маппинга импортируемых сущностей. Этот выбор был обусловлен форматом, в котором поступают данные с платформы ELMA3, и существующими типами данных на платформе ELMA365. Для каждого типа платформы ELMA365, рассматриваемого в рамках данного дипломного проекта, было подобрано своё уникальное регулярное выражение, способное описать его структуру. Затем на основе этих регулярных выражений данные записываются на платформу-преемник.

После реализации проекта необходимо было провести апробацию разработанного модуля. Данный процесс был описан в виде шагов воспроизведения и подключения модуля для успешной миграции данных с платформы ELMA3 на платформуELMA365.

В дальнейшем, проект может быть расширен и доработан для поддержки других платформ или добавления дополнительных функций, которые упростят процесс миграции данных и обеспечат ещё большую гибкость пользователей в работе с информацией.

В целом, выполнение дипломного проекта по разработке модуля миграции данных с платформы ELMA3 на платформу ELMA365 было успешным. Полученные результаты подтверждают его работоспособность и готовность к применению в реальных условиях. Проект открывает новые возможности для организаций, стремящихся совершенствовать свои бизнес-процессы и использовать современные технологии в управлении своей деятельностью.

# ПРИЛОЖЕНИЕ А

(обязательное)

Авторская справка

Я, Кудяшев Ярослав Юрьевич, автор выпускной квалификационной работы «Разработка модуля миграции данных при изменении архитектуры BPM платформы» сообщаю, что мне известно о персональной ответственности автора за разглашение сведений, подлежащих защите законами РФ о защите объектов интеллектуальной собственности.

Одновременно сообщаю, что:

1. При подготовке к защите выпускной квалификационной работы не использованы источники (документы, отчёты, диссертации, литература и т.п.), имеющие гриф секретности или «Для служебного пользования» ФГБОУ ВО «Вятский государственный университет» или другой организации.

2. Данная работа не связана с незавершёнными исследованиями или уже с завершёнными, но ещё официально не разрешёнными к опубликованию ФГБОУ ВО «Вятский государственный университет» или другими организациями.

3. Данная работа не содержит коммерческую информацию, способную нанести ущерб интеллектуальной собственности ФГБОУ ВО «Вятский государственный университет» или другой организации.

4. Данная работа не является результатом НИР или ОКР, выполняемой по договору с организацией.

5. В предлагаемом к опубликованию тексте нет данных по незащищённым объектам интеллектуальной собственности других авторов.

6. Использование моей дипломной работы в научных исследованиях оформляется в соответствии с законодательством РФ о защите интеллектуальной собственности отдельным договором.

Автор: Кудяшев Я.Ю. «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_2023 г. \_\_\_\_\_\_\_\_\_\_\_\_

подпись

Сведения по авторской справке подтверждаю: «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

Заведующая кафедрой ЭВМ: М.Л.Долженкова \_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

# ПРИЛОЖЕНИЕ Б

(обязательное)

Листинг программного кода

/\* Client scripts module \*/

declare const console: any

/\* Class for elma365 types \*/

class elma365Type {

    name: string;

    regularExpression: string;

    weight: number;

    constructor(name: string, regularExpression: string, weight: number) {

        this.name = name;

        this.regularExpression = regularExpression;

        this.weight = weight;

    }

    check(inputString: string): boolean {

        const regex = new RegExp(this.regularExpression);

        return regex.test(inputString);

    }

}

//ELMA365 types to be migrated

const elma365Types: elma365Type[] = [

    new elma365Type("Строка", ".\*", 1),

    new elma365Type("Целое число", "^(?!0\\d)\\d{1,32}$", 3),

    new elma365Type("Дробное число", "^(?!0\\d)\\d{1,32}(\\.\\d{1,8})?$", 2),

    new elma365Type("Выбор Да/Нет", "^(true|false)$", 2),

    new elma365Type("Дата/Время", "^\\d{4}-(0[1-9]|1[0-2])-([0-2][1-9]|3[01])T([01][0-9]|2[0-3]):([0-5][0-9]):([0-5][0-9])\\.\\d{3}Z$", 3),

    new elma365Type("Дата", "^(\\d{4})-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])T00:00:00\\.000Z$", 2),

    new elma365Type("Время", "^1970-01-01T(0[0-9]|1[0-9]|2[0-3]):([0-5][0-9]):([0-5][0-9]).000Z$", 2),

    new elma365Type("Электронная почта", "^[^\\s@]+@[^\\s@]+\\.[^\\s@]+$", 2),

    new elma365Type("Учётная запись", "@\\S+", 2),

    new elma365Type("Деньги", "(\\d+)RUB", 2),

    new elma365Type("Номер телефона", "^(\\+7|7|8)[-]?\\d{3}[-]?\\d{3}[-]?\\d{2}[-]?\\d{2}$", 2),

    new elma365Type("Ф.И.О.", "^[А-ЯЁ][а-яё]+(\\s[А-ЯЁ][а-яё]+)?\\s[А-ЯЁ][а-яё]+$", 2),

    new elma365Type("Ссылка", "^(https?|ftp):\\/\\/[^\\s/$.?#].[^\\s]\*$", 2)

];

const typeNames: string[] = elma365Types.map((type) => type.name);

//Interface for new elements

interface exportData {

    sectionCode: string;    //ELMA365 section code

    applicationCode: string;    //ELMA365 application code

    fields: {

        fieldCode: string;  //ELMA365 attribute code

        fieldType: string;

        fieldData: string;

        fieldDataId: string;

    }[]

}

//mainCollection ELMA3

interface elma3Collection {

    sectionCode: string;

    newSectionCode: string;

    applications: {

        applicationCode: string;

        applicatioName: string;

        newApplicationCode: string;

        fields: {

            fieldCode: string;

            fieldName: string;

            recommendedTypes: string[];

            newType: string;

        }[]

    }[]

}

const token = ' Bearer f3849368-7d6e-4d42-a7e2-9354650f366b'

const section = 'store\_\_'

const applications = [

    'orders',

    'test',

    'n23icukveanpgr',

    'testovoe\_prilozhenie'

]

let elma3Collection: elma3Collection = {

    sectionCode: section,

    newSectionCode: section,

    applications: [],

}

//Get sysCollection from ELMA3

async function getCollectionFromElma3(): Promise<void> {

    //Get all applications of the section

    for (let i = 0; i < applications.length; i++) {

        const url = 'https://4toea5on4zmik.elma365.ru/pub/v1/app/' + section + '/' + applications[i] + '/list'

        const data = {

        }

        const option = {

            method: 'POST',

            headers: {

                'Authorization': token

            },

            body: JSON.stringify(data)

        }

        const response = await fetch(url, option)

        //take data to the structure elma3Collection

        if (response.ok) {

            const answer = (await response.json()).result.result[0]

            //console.log(answer)

            const application: any = {

                applicationCode: applications[i],

                applicatioName: "",

                newApplicationCode: "",

                fields: [],

            };

            const keys = Object.keys(answer)

            //console.log(keys)

            for (let j = 0; j < keys.length; j++) {

                const fieldCode = keys[j]; // Получаем первый ключ (fieldCode) объекта

                application.fields.push({

                    fieldCode: fieldCode,

                    fieldName: "",

                    recommendedTypes: typeNames,

                    newType: "",

                });

            }

            elma3Collection.applications.push(application);

        } else {

            console.log('Ошибка при получении структуры платформы ELMA3')

        }

    }

    //SystemFields to be deleted

    const systemFields = [

        '\_\_debug',

        '\_\_index',

        '\_\_version',

        '\_\_createdAt',

        '\_\_createdBy',

        '\_\_updatedAt',

        '\_\_updatedBy',

        '\_\_externalId',

        '\_\_subscribers',

        '\_\_externalProcessMeta',

        '\_\_directory'

    ]

    // function for checking the occurrence of strings in an array

    function isSystemField(fieldCode: string) {

        return systemFields.indexOf(fieldCode) !== -1;

    }

    // Delete system fields from applications

    elma3Collection.applications.forEach((application) => {

        application.fields = application.fields.filter((field) => !isSystemField(field.fieldCode));

    });

    //console.log(elma3Collection)/\* Осталось убрать ненужные поля \*/

}

/\* Cheking data from ELMA3 with regular expression \*/

async function checkViaRegularExpressions(): Promise<void> {

    for (let applicationNumber = 0; applicationNumber < elma3Collection.applications.length; applicationNumber++) {

        const url = 'https://4toea5on4zmik.elma365.ru/pub/v1/app/' + section + '/' + applications[applicationNumber] + '/list'

        const data = {

        }

        const option = {

            method: 'POST',

            headers: {

                'Authorization': token

            },

            body: JSON.stringify(data)

        }

        const response = await fetch(url, option)

        //take data to the structure elma3Collection

        if (response.ok) {

            const answer = (await response.json()).result.result

            //console.log(answer)

            for (let fieldsNumber = 0; fieldsNumber < elma3Collection.applications[applicationNumber].fields.length; fieldsNumber++) {

                //Get array of on property data

                let dataForCheck: string[] = []

                for (let i = 0; i < answer.length; i++) {

                    const data = eval('answer[' + String(i) + '].' + elma3Collection.applications[applicationNumber].fields[fieldsNumber].fieldCode)

                    dataForCheck.push(data)

                }

                //Regular expression check

                let recommendedTypes: string[] = []

                for (let i = 0; i < dataForCheck.length; i++) {

                    for (let j = 0; j < elma365Types.length; j++) {

                        const data = elma365Types[j].check(dataForCheck[i])

                        if (data == false) {

                            recommendedTypes.push(elma365Types[j].name)

                        }

                    }

                }

                const uniqueStrings: string[] = recommendedTypes.filter((str, index) => recommendedTypes.indexOf(str) === index);

                const fullTypes = typeNames

                const filteredArray1: string[] = fullTypes.filter((str) => uniqueStrings.indexOf(str) === -1);

                elma3Collection.applications[applicationNumber].fields[fieldsNumber].recommendedTypes = filteredArray1

                //console.log(recommendedTypes)

            }

        }

        //console.log(dataForCheck)

        console.log(elma3Collection)

    }

}

async function onInit(): Promise<void> {

    const data: exportData = {

        sectionCode: "n3543",

        applicationCode: "ysvsy",

        fields: [

            {

                fieldCode: "stroka",

                fieldType: "STRING",

                fieldData: "",

                fieldDataId: "id1",

            },

            {

                fieldCode: "\_\_name",

                fieldType: "STRING",

                fieldData: "",

                fieldDataId: "id2",

            },

            {

                fieldCode: "stroka\_1",

                fieldType: "STRING",

                fieldData: "",

                fieldDataId: "",

            },

            {

                fieldCode: "stroka\_2",

                fieldType: "STRING",

                fieldData: "",

                fieldDataId: "id2",

            },

            // Дополнительные элементы полей

        ],

    };

    //await getCollectionFromElma3()

    //await checkViaRegularExpressions()

    await getSysCollection()

    //await createElement(data)

}

/\* Create element on the platform ELMA365 \*/

async function createElement(data: exportData): Promise<void> {

    if (data.applicationCode && data.sectionCode) {

        //create element

        const stringCreateElement = 'Global.ns.' + data.sectionCode + '.app.' + data.applicationCode + '.create()'

        const newElement = eval(stringCreateElement)

        data.fields.forEach((field) => {

            switch (field.fieldType) {

                case 'STRING':

                    const changeField = 'newElement.data.' + field.fieldCode + ' = "' + field.fieldData + '"'

                    eval(changeField)

                    break;

                case 'NUMBER':

                    break;

                default:

                    break;

            }

        })

        //save element

        const stringSaveElement = 'newElement.save()'

        eval(stringSaveElement)

    }

}

/\* Get system collections \*/

async function getSysCollection(): Promise<void> {

    //mainCollection ELMA365

    interface sysCollection {

        sectionName: string;

        sectionCode: string;

        applications: {

            applicationName: string;

            applicationCode: string;

            fields: {

                fieldName: string;

                fieldCode: string;

                fieldType: string;

            }[];

        }[];

    }

    //mainArray

    const resultArray: sysCollection[] = [];

    //get system structure

    const systemStructure = [Global.ns];

    //console.log(systemStructure);

    //all sections names and applications

    const objectNames: string[] = [];

    for (const obj of systemStructure) {

        const keys = Object.keys(obj);

        for (const key of keys) {

            objectNames.push(key);

        }

    }

    //console.log(objectNames);

    let correctName = ""

    //Приводим массив строк к нужному виду

    for (let i = 0; i < objectNames.length; i++) {

        const dotIndex = objectNames[i].indexOf(".")

        if (dotIndex !== -1) {

            correctName = objectNames[i].slice(0, dotIndex)

        } else {

            correctName = objectNames[i]

        }

        //Отсеиваем по признаку наличия имени

        const requestStringName = "Global.ns." + correctName + ".name"

        let isTrueName = false

        try {

            if (eval(requestStringName) !== "") {

                isTrueName = true

            }

        } catch (err) {

        }

        if (isTrueName == true) {

            const sysObj: sysCollection = {

                sectionName: "",

                sectionCode: correctName,

                applications: []

            };

            resultArray.push(sysObj);

        }

    }

    //console.log(resultArray);

    // Убираем дубликаты, пустые объекты

    const uniqueResultArray = resultArray.filter(

        (item, index, self) =>

            self.findIndex((i) => i.sectionCode === item.sectionCode) === index

    );

    //Присваивание названий разделов

    for (let app of uniqueResultArray) {

        const requestString = "Global.ns." + app.sectionCode + ".name"

        app.sectionName = eval(requestString)

    }

    //Присваиваем приложения разделов

    for (let app of uniqueResultArray) {

        const request = "Global.ns." + app.sectionCode + ".app"

        const appsObject = eval(request)

        const sectionApps = [appsObject];

        //all sections names and applications

        for (const obj of sectionApps) {

            const keys = Object.keys(obj);

            for (const key of keys) {

                if (key.indexOf('@') == -1) {

                    const appFields = {

                        applicationName: "",

                        applicationCode: key,

                        fields: []

                    }

                    app.applications.push(appFields)

                }

            }

        }

    }

    //Удаление элементов, где нет приложений

    const filteredArray = resultArray.filter(item => item.applications.length > 0)

    //Присвоение имен приложениям разделов

    filteredArray.forEach((item) => {

        item.applications.forEach((app) => {

            try {

                app.applicationName = eval(`Global.ns.${item.sectionCode}.app.${app.applicationCode}.name`)

            } catch (err) {

            }

        })

    })

    //Присвоение атрибутов приложениям

    //Названия системных атрибутов, которые нужно исключить

    const prohibitedArray = [

        '\_\_createdAt', '\_\_state',

        '\_\_createdBy', '\_\_target',

        '\_\_updatedAt', '\_\_targetData',

        '\_\_updatedBy', '\_\_subscribers',

        '\_\_deletedAt', '\_\_currentPerformers',

        '\_\_directory', '\_\_templateId',

        '\_\_externalProcessMeta', '\_\_logged',

        '\_\_externalId', '\_\_notifyOnStart',

        '\_\_index', '\_\_item',

        '\_\_tasks', '\_\_parentId',

        '\_\_tasks\_earliest\_duedate', '\_\_parent',

        '\_\_tasks\_performers', '\_\_branches',

        '\_\_debug', '\_\_template',

        '\_\_status',

        '\_\_id',

        '\_\_register\_name',

        '\_\_file'

    ]

    //Присвоение атрибутов приложениям

    for (let app of uniqueResultArray) {

        const sectionApps = app.applications

        //all sections names and applications

        for (const obj of sectionApps) {

            const request = "Global.ns." + app.sectionCode + ".app." + obj.applicationCode + ".fields"

            const appsObject = eval(request)

            const sectionApps = [appsObject];

            for (let fields of sectionApps) {

                const keys = Object.keys(fields)

                for (const key of keys) {

                    if (!prohibitedArray.some(item => key.indexOf(item) !== -1)) {

                        const appFields = {

                            fieldName: "",

                            fieldCode: key,

                            fieldType: ""

                        }

                        obj.fields.push(appFields)

                    }

                }

            }

        }

    }

    //Присвоение названий и типов атрибутам приложений

    filteredArray.forEach((item) => {

        item.applications.forEach((app) => {

            app.fields.forEach((field) => {

                try {

                    field.fieldName = eval(`Global.ns.${item.sectionCode}.app.${app.applicationCode}.fields.${field.fieldCode}.name`)

                    field.fieldType = eval(`Global.ns.${item.sectionCode}.app.${app.applicationCode}.fields.${field.fieldCode}.type`)

                } catch (err) {

                }

            })

        })

    })

    console.log(filteredArray)

}

<body>

    <!-- Main button -->

    <button type="button" id="main-button">

        <img class="main-image" src="https://izhevsk.ru/forums/icons/forum\_pictures/010224/10224890.png">

    </button>

    <!-- Window for authorization -->

    <div class="modal-overlay"></div>

    <div class="modal-window" id="main-window">

        <div class="modal">

            <div class="modal-window-title">

                <h2>Первый этап: Авторизация</h2>

            </div>

            <div class="modal-window-authorization">

                <div class="form-login">

                    <label for="login">Логин:</label>

                    <input type="text" id="login" placeholder="Введите логин">

                </div>

                <div class="form-password">

                    <label for="password">Пароль:</label>

                    <input type="password" id="password" placeholder="Введите пароль">

                </div>

            </div>

            <div class="modal-window-buttons">

                <button class="cancell-button">Отмена</button>

                <button class="next-button" disabled>Далее</button>

            </div>

        </div>

        <div class="modal-instruction">

            <div class="instruction-label">

                <h2>Инструкция</h2>

            </div>

            <div class="instruction-text-first-window">

                <h3>1. Введите логин и пароль от аккаунта ELMA3.</h3>

                <h3>2. Убедитесь, что аккаунт пользователя имеет доступ ко всем разделам платформы.</h3>

                <h3>3. Нажмите на кнопку "Далее" для перехода ко второму этапу миграции данных.</h3>

                <h3></h3>

            </div>

        </div>

    </div>

    <!-- Window for mapping -->

</body>

<script src="scripts.js"></script>

/\* Logic for first window \*/

const mainButton = document.getElementById("main-button")

const modalOverlay = document.querySelector(".modal-overlay")

const modal = document.querySelector(".modal-window")

const cancellButton = document.querySelector(".cancell-button")

const nextButton = document.querySelector(".next-button")

const loginInput = document.getElementById("login")

const passwordInput = document.getElementById("password")

let userToken = ''

/\* MainButton click \*/

mainButton.addEventListener("click", function () {

    modalOverlay.style.display = "block";

    modal.style.display = "flex";

})

/\* Validation for first window \*/

function handleInputChange() {

    if (passwordInput.value !== '' && loginInput.value !== '') {

        nextButton.disabled = false

    } else {

        nextButton.disabled = true

    }

}

/\* Login change \*/

loginInput.addEventListener("change", handleInputChange)

/\* Password change \*/

passwordInput.addEventListener("change", handleInputChange)

/\* CancellButton click \*/

cancellButton.addEventListener("click", function () {

    modalOverlay.style.display = "none";

    modal.style.display = "none";

    loginInput.value = "";

    passwordInput.value = "";

})

/\* Get User Token \*/

async function getUserToken(event) {

    const url = 'https://4toea5on4zmik.elma365.ru/admin/token'

    const data = {

        username: '',

        password: ''

    }

    const option = {

        method: 'POST',

        headers: {

            'Content-Type': 'application/json'

        },

        body: JSON.stringify(data)

    }

    try {

        const response = await fetch(url, option)

        const tokenData = await response.json()

        userToken = tokenData.token

    } catch {

        userToken = 'f3849368-7d6e-4d42-a7e2-9354650f366b'

        console.log(userToken)

    }

}

async function getElma3Entities() {

}

/\* NextButton click \*/

nextButton.addEventListener("click", function () {

    //Получить объекты с платформы ELMA3

    getUserToken()

    //Смена формы окна и переход ко второму этапу

    const mainWindow = document.getElementById("main-window")

    mainWindow.innerHTML = ""

    // Изменить дизайн окна

    modal.style.transition = "width 0.5s ease, height 0.5s ease"

    modal.style.width = "1200px"

    modal.style.height = "700px"

    const newMarkup = `

           <div class="modal">

               <div class="modal-window-title">

                   <h2>Второй этап: Соотношение сущностей</h2>

               </div>

               <div class="mapping-entities">

                   <div class="elma3-container">

                       <h2>ELMA3</h2>

                       <div class="elma3-entities"></div>

                    </div>

                   <div class="elma365-container">

                       <h2>ELMA365</h2>

                       <div class="elma365-entities"></div>

                    </div>

               </div>

               <div class="modal-window-buttons">

                   <button class="cancell-button">Отмена</button>

                   <button class="next-button" disabled>Далее</button>

                   <button class="reset-button">Сброс</button>

               </div>

           </div>

           <div class="modal-instruction">

               <div class="instruction-label">

                   <h2>Инструкция</h2>

               </div>

               <div class="instruction-text-second-window">

                   <h3>1. Выберите раздел платформы ELMA3 в левом углу окна.</h3>

                   <h3>2. Выберите раздел платформы ELMA365 в правом углу окна.</h3>

                   <h3>3. Таким же образом соотнесите открывшиеся поля двух разделов.</h3>

               </div>

                <div class="logging-label">

                <h2>Логирование</h2>

                </div>

                <div class="logging-text">

               </div>

           </div>

           `

    mainWindow.innerHTML = newMarkup

    /\* Logic for second window \*/

    const secondModalOverlay = document.querySelector(".modal-overlay")

    const secondModal = document.querySelector(".modal-window")

    const secondCancellButton = document.querySelector(".cancell-button")

    //Текущее приложение ELMA365

    let currentAppElma365 = ''

    let elma365FieldCods = []

    function clearELMA365Entities() {

        try {

            if (elma365FieldCods.length !== 0) {

                elma365FieldCods.forEach((id) => {

                    const divElements = elma365.querySelectorAll(`div#${id}`);

                    divElements.forEach((div) => {

                        // Изменение класса div

                        div.classList.add("field");

                        div.classList.remove("field-active-elma365");

                    });

                });

                console.log(elma365FieldCods)

            }

        }catch(err){

        }

    }

    // Get containers for entities

    const elma3 = document.querySelector(".elma3-entities")

    const elma365 = document.querySelector(".elma365-entities")

    const resetButton = document.querySelector(".reset-button")

    //Drawing entities in blocks for ELMA3

    function renderApplications() {

        elma3.innerHTML = ""

        clearELMA365Entities()

        applications.forEach((app) => {

            const block = document.createElement("div");

            block.className = "block";

            block.id = app.applicationCode

            block.textContent = app.applicationCode;

            block.addEventListener("click", () => {

                renderFields(app.fields, block);

            });

            elma3.appendChild(block);

        });

    }

    function renderFields(fields, block) {

        elma3.innerHTML = ""; // Очистить от блоков приложений

        //Очистка активных блоков ELMA365

        clearELMA365Entities()

        fields.forEach((field, blocks) => {

            const fieldDiv = document.createElement("div");

            fieldDiv.className = "field-active-elma3";

            fieldDiv.id = field.fieldCode

            fieldDiv.textContent = 'Название: ' + field.fieldCode;

            const fieldTypes = document.createElement("div")

            fieldTypes.textContent = "Рекомендованные типы: "

            const fieldsArray = document.createElement("div")

            fieldsArray.textContent = `${field.recommendedTypes.join(", ")}`

            fieldDiv.addEventListener("click", () => {

                getRightEntities(fieldDiv, block);

            })

            elma3.appendChild(fieldDiv);

            fieldDiv.appendChild(fieldTypes)

            fieldDiv.appendChild(fieldsArray)

        });

        //Отображаем только подходящие по типу сущности ELMA365

        function getRightEntities(fieldDiv, block) {

            clearELMA365Entities()

            //Проверка для сущностей ELMA3

            const allFieldDivs = elma3.getElementsByClassName("field-active-elma3");

            Array.from(allFieldDivs).forEach((div) => {

                if (div === fieldDiv) {

                    div.classList.add("field-touched-elma3");

                } else {

                    div.classList.remove("field-touched-elma3");

                }

            });

            //Проверка для сущностей ELMA365

            //Получение нужных рекомендуемых типов

            function findRecommendedTypesByFieldCode(applications, fieldCode, targetApplicationCode) {

                const foundApplication = applications.find((application) => {

                    return application.applicationCode === targetApplicationCode

                });

                if (foundApplication) {

                    const foundField = foundApplication.fields.find((field) => {

                        return field.fieldCode === fieldCode;

                    });

                    return foundField.recommendedTypes;

                } else {

                    return null; // Если не найдено совпадений, возвращаем null или другое значение по вашему усмотрению

                }

            }

            //Получение fieldCode сущностей ELMA365, в которых присутствует recomendedTypes

            function findFieldCodeInELMA365(applications, appCode, recommendedTypes) {

                const foundApplication = applications.find((application) => {

                    return application.applicationCode === appCode

                })

                //console.log(foundApplication)

                let foundField

                if (foundApplication) {

                    foundField = foundApplication.fields

                        .filter((field) => recommendedTypes.includes(field.fieldType))

                        .map((field) => field.fieldCode);

                } return foundField

            }

            //Рекоменжуемые типы у выбранного div

            const recommendedTypes = findRecommendedTypesByFieldCode(applications, fieldDiv.id, block.id);

            //console.log(recommendedTypes);

            //Получаем id тех div, у которых соответствуют типы. Остлось их сделать активными

            elma365FieldCods = findFieldCodeInELMA365(applicationsELMA365, currentAppElma365, recommendedTypes)

            //console.log(elma365FieldCods)

            elma365FieldCods.forEach((id) => {

                const divElements = elma365.querySelectorAll(`div#${id}`);

                divElements.forEach((div) => {

                    // Изменение класса div

                    div.classList.add("field-active-elma365");

                    div.classList.remove("field");

                });

            });

        }

        // Добавить кнопку-стрелку для возврата к приложениям

        const arrowButton = document.createElement("div");

        arrowButton.className = "arrow-button";

        arrowButton.textContent = "←";

        arrowButton.addEventListener("click", renderApplications);

        elma3.appendChild(arrowButton);

    }

    //Drawing entities for ELMA365

    function renderApplicationsELMA365() {

        elma365.innerHTML = ""

        currentAppElma365 = null

        applicationsELMA365.forEach((app) => {

            const block = document.createElement("div");

            block.className = "block";

            block.id = app.applicationCode

            block.textContent = app.applicationName;

            block.addEventListener("click", () => {

                renderFieldsELMA365(app.fields, block);

            });

            elma365.appendChild(block);

        });

    }

    function renderFieldsELMA365(fields, block) {

        elma365.innerHTML = ""; // Очистить от блоков приложений

        currentAppElma365 = block.id

        fields.forEach((field) => {

            const fieldDiv = document.createElement("div");

            fieldDiv.className = "field";

            fieldDiv.id = field.fieldCode

            fieldDiv.textContent = `${field.fieldName} `;

            const fieldType = document.createElement("div")

            fieldType.textContent = `Тип: ${field.fieldType}`

            elma365.appendChild(fieldDiv);

            fieldDiv.appendChild(fieldType)

        });

        // Добавить кнопку-стрелку для возврата к приложениям

        const arrowButton = document.createElement("div");

        arrowButton.className = "arrow-button";

        arrowButton.textContent = "←";

        arrowButton.addEventListener("click", renderApplicationsELMA365);

        elma365.appendChild(arrowButton);

    }

    resetButton.addEventListener("click", () => {

        renderApplications();

        renderApplicationsELMA365();

    })

    /\* CancellButton click \*/

    secondCancellButton.addEventListener("click", function () {

        secondModalOverlay.style.display = "none";

        secondModal.style.display = "none";

    })

})

/\* Login change \*/

loginInput.addEventListener("change", handleInputChange)

/\* Password change \*/

passwordInput.addEventListener("change", handleInputChange)

/\* CancellButton click \*/

cancellButton.addEventListener("click", function () {

    modalOverlay.style.display = "none";

    modal.style.display = "none";

    loginInput.value = "";

    passwordInput.value = "";

})

/\* Get User Token \*/

async function getUserToken(event) {

    const url = 'https://4toea5on4zmik.elma365.ru/admin/token'

    const data = {

        username: '',

        password: ''

    }

    const option = {

        method: 'POST',

        headers: {

            'Content-Type': 'application/json'

        },

        body: JSON.stringify(data)

    }

    try {

        const response = await fetch(url, option)

        const tokenData = await response.json()

        userToken = tokenData.token

    } catch {

        userToken = 'f3849368-7d6e-4d42-a7e2-9354650f366b'

        console.log(userToken)

    }

}

async function getElma3Entities() {

}

/\* NextButton click \*/

nextButton.addEventListener("click", function () {

    //Получить объекты с платформы ELMA3

    getUserToken()

    //Смена формы окна и переход ко второму этапу

    const mainWindow = document.getElementById("main-window")

    mainWindow.innerHTML = ""

    // Изменить дизайн окна

    modal.style.transition = "width 0.5s ease, height 0.5s ease"

    modal.style.width = "1200px"

    modal.style.height = "700px"

    const newMarkup = `

           <div class="modal">

               <div class="modal-window-title">

                   <h2>Второй этап: Соотношение сущностей</h2>

               </div>

               <div class="mapping-entities">

                   <div class="elma3-container">

                       <h2>ELMA3</h2>

                       <div class="elma3-entities"></div>

                    </div>

                   <div class="elma365-container">

                       <h2>ELMA365</h2>

                       <div class="elma365-entities"></div>

                    </div>

               </div>

               <div class="modal-window-buttons">

                   <button class="cancell-button">Отмена</button>

                   <button class="next-button" disabled>Далее</button>

                   <button class="reset-button">Сброс</button>

               </div>

           </div>

           <div class="modal-instruction">

               <div class="instruction-label">

                   <h2>Инструкция</h2>

               </div>

               <div class="instruction-text-second-window">

                   <h3>1. Выберите раздел платформы ELMA3 в левом углу окна.</h3>

                   <h3>2. Выберите раздел платформы ELMA365 в правом углу окна.</h3>

                   <h3>3. Таким же образом соотнесите открывшиеся поля двух разделов.</h3>

               </div>

                <div class="logging-label">

                <h2>Логирование</h2>

                </div>

                <div class="logging-text">

               </div>

           </div>

           `

    mainWindow.innerHTML = newMarkup

    /\* Logic for second window \*/

    const secondModalOverlay = document.querySelector(".modal-overlay")

    const secondModal = document.querySelector(".modal-window")

    const secondCancellButton = document.querySelector(".cancell-button")

    //Текущее приложение ELMA365

    let currentAppElma365 = ''

    let elma365FieldCods = []

    function clearELMA365Entities() {

        try {

            if (elma365FieldCods.length !== 0) {

                elma365FieldCods.forEach((id) => {

                    const divElements = elma365.querySelectorAll(`div#${id}`);

                    divElements.forEach((div) => {

                        // Изменение класса div

                        div.classList.add("field");

                        div.classList.remove("field-active-elma365");

                    });

                });

                console.log(elma365FieldCods)

            }

        }catch(err){

        }

    }

# ПРИЛОЖЕНИЕ В

(справочное)

Библиографический список

1. Справка по платформе ELMA365 [Электронный ресурс]. – Режим доступа: https://elma365.com/ru/help/.
2. Справка по внешнему API платформы ELMA3 [Электронный ресурс]. – Режим доступа: https://www.elma-bpm.ru/KB/article-6073.html.
3. Справочник CSS. Кратко, быстро, под рукой / Хрусталёв А.А. [Текст] – Санкт-Петербург: Изд-во НиТ, 2021. – 304 с.
4. Справочник по работе с JavaScript [Электронный ресурс] – Режим доступа https://javascript.ru/manual.
5. Справочник по работе с CSS [Электронный ресурс] – Режим доступа: https://hcdev.ru/css/.
6. Справочник по работе с HTML [Электронный ресурс]. – Режим доступа: http://htmlbook.ru/html.
7. mathJS [Электронный ресурс]. – Режим доступа: https://mathjs.org/.
8. Moment.js [Электронный ресурс]. – Режим доступа: https://momentjs.com/docs/.
9. REST API [Электронный ресурс]. – Режим доступа: https://www.redhat.com/en/topics/api/what-is-a-rest-api
10. Postman [Электронный ресурс]. – Режим доступа: https://learning.postman.com/docs/publishing-your-api/documenting-your-api/
11. Регулярные выражения в JavaScript [Электронный ресурс]. – Режим доступа: https://javascript.ru/basic/regular-expression
12. ELMA Academy [Электронный ресурс]. – Режим доступа: https://elma-academy.com/ru/
13. Справочник JavaScript. Кратко, быстро, под рукой / Никольский А.П. [Текст]. – Санкт-Петербург: Изд-во НиТ, 2021. – 304 с.
14. Справочник HTML. Кратко, быстро, под рукой / Кириченко А.В. [Текст]. – Санкт-Петербург: Изд-во НиТ, 2021. – 288 с.