МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК
Кафедра программного обеспечения

РЕКОМЕНДОВАНО К ЗАЩИТЕ В ГЭК
И ПРОВЕРЕНО НА ОБЪЕМ
ЗАИМСТВОВАНИЯ
Заведующий кафедрой
к.т.н., доцент
_____М. С. Воробьева
24.06 2019 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(магистерская диссертация)

РАЗРАБОТКА ПОДСИСТЕМЫ ПРИНЯТИЯ РЕШЕНИЙ ОБ ОКАЗАНИИ УСЛУГ НА ОСНОВЕ ДАННЫХ О ЗАЯВИТЕЛЕ ДЛЯ ГАУ ТО «МФЦ»

02.04.03. Математическое обеспечение и администрирование
информационных систем
Магистерская программа «Разработка, администрирование и защита
вычислительных систем»

Выполнил работу
Студент 2 курса
очной формы обучения — Дементьев Илья Олегович

Руководитель работы
к.п.н., доцент кафедры
программного
обеспечения — Плотоненко Юрий Анатольевич

Рецензент
Заместитель директора
ГАУ ТО «МФЦ» — Сальников Дмитрий Евгеньевич

Тюмень – 2019

Оглавление

Введение

С ростом популярности МФЦ и количества людей, обращающихся для получения услуги в МФЦ, растет количество заявлений, которые необходимо обрабатывать. Обмен и получение необходимых данных из органов государственной власти и передача им информации для оказания услуги происходит через специализированные интерфейсы взаимодействия информационных систем. При этом большое количество времени при оказании услуги занимает ручная проверка данных, формирование запросов на получение информации, ввод полученных данных в систему и дальнейшее управление процессом оказания услуги. На данный момент большинство органов государственной власти имеют собственные сервисы электронного взаимодействия, что позволяет упростить задачу обмена информации с ними. При этом уровень вовлеченности человека в процесс оказания услуги все еще довольно высок, ему все еще приходится управлять процессом оказания услуги, который, в большинстве случаев, строго регламентирован. Поэтому можно минимизировать участие человека, передав всю возможную работу программному комплексу, и облегчить работу человека на тех этапах, где его участие необходимо.

Целью работы является: разработка подсистемы принятия решений о предоставлении услуги услуг на основе данных о заявителе для ГАУ ТО «МФЦ» на примере услуги выдачи электронной транспортной карты (далее ЭТК). Эта подсистема позволит автоматизировать процесс оказания услуги, уменьшив участие человека в процессе, а в некоторых случаях и вовсе обойтись без него.

Глава 1. Системы принятия решений

1.1 Описание деятельности предприятия

Многофункциональный центр предоставления государственных и муниципальных услуг (далее - МФЦ) - государственное учреждение, осуществляющее функции по взаимодействию с органами государственной власти, органами местного самоуправления и организациями, участвующими в предоставлении государственных (муниципальных) услуг, информированию граждан и организаций, приему и выдаче документов, обработке персональных данных, связанных с предоставлением указанных услуг.

Услуги в МФЦ оказываются по принципу «одного окна», т.е. исключение или максимально возможное ограничение участия заявителя в процессах сбора различных справок и документов, необходимых для получения той или иной государственной услуги, а также прозрачное и контролируемое прохождение документов на всех этапах предоставления государственных услуг.

Цели создания:

- Повышение качества и доступности государственных услуг;
- Сокращение сроков предоставления услуг;
- Повышение эффективности деятельности органов исполнительной власти и межведомственной координации;
- Повышение открытости и прозрачности для общества;

Основными функциями МФЦ являются:

- Прием запросов заявителей о предоставлении государственных или муниципальных услуг;
- Представление интересов заявителей при взаимодействии с государственными органами, органами местного самоуправления,

а также с организациями, участвующими в предоставлении государственных и муниципальных услуг;

- Представление интересов государственных органов, органов местного самоуправления при взаимодействии с заявителями;
- Информирование заявителей о порядке предоставления государственных и муниципальных услуг в МФЦ, о ходе выполнения запросов о предоставлении государственных и муниципальных услуг, а также по иным вопросам, связанным с предоставлением государственных и муниципальных услуг;
- Взаимодействие с государственными органами и органами местного самоуправления по вопросам предоставления государственных и муниципальных услуг, а также с организациями, участвующими в предоставлении государственных и муниципальных услуг;
- Выдача заявителям документов по результатам предоставления государственных и муниципальных услуг, если иное не предусмотрено законодательством Российской Федерации;
- Прием, обработка информации из информационных систем государственных органов, органов местного самоуправления, а также выдача заявителям на основании такой информации документов, если иное не предусмотрено федеральным законом; [5]

1.2 Описание систем принятия решений

В современном обществе круг задач, решаемых человеком, изменился, они стали более сложные и непривычные. Раньше темп изменения окружающей среды в жизни людей был невелик, и новые задачи возникали постепенно, поэтому решения принимались, на основе на одного-двух

главных факторов, не учитывая многие другие. В большинстве современных задач приходится учитывать огромное число взаимосвязанных факторов, определяющие варианты решений.

Система принятия решений – это информационная система, позволяющая принимать решения используя данные, знания, опыт и различные математические модели.

Одним из видов систем принятия решений является система поддержки принятия решений. Система поддержки принятия решений (СППР) – это система, исполняющая некоторую совокупность процедур обработки данных, предоставляющая доступ к данным и помогающая человеку в принятии решения. Задачи, решаемые с помощью СППР, делятся на два типа:

- Задачи, имеющие объективное решение. Такие задачи характеризуются наличием критерия достижения цели, четко определенными правилами принятия решения и наличием аналитической модели
- Задачи, не имеющие объективного решения. Решением таких задач будет являться получение экспертной оценки, на основе эвристических предпочтений. К ним относятся задачи, связанные с прием политических или экономических решений, а также задачи диагностики. [6]

Экспертная система (ЭС) – это система позволяющая на основе заложенных данных принимать решение по определенным типам задач, заменяя тем самым специалистов в этой области.

На основе функционального назначения экспертные системы делятся на следующие типы:

- Мощные экспертные системы, узконаправленные системы управления сложным оборудованием;
- Экспертные системы, рассчитанные для использования широким кругом специалистов;
- Экспертные системы, для массового потребителя с небольшим числом правил Применение таких систем позволяет, упростить поиск и устранение неисправностей, обходясь без высококвалифицированного персонала;
- Простые экспертные системы индивидуального использования, на основе небольшого набора правил, создаваемые для облегчения рутинной работы. [7]

Построение экспертной системы для решения, задач, требующих выполнения четкого выполнения модели, не является рациональным решением, поскольку в этом случае они строго детерминированы. ЭС же предназначена для решения неформализуемых задач на основе накопленных знаний в определенной области.

Сейчас по мере внедрения автоматизированных систем управления (АСУ) математические методы приобретают большую роль. АСУ— это система, которая состоит из специально обученного персонала, совокупности оборудования и набора программного обеспечения, использующегося для автоматизации процессов. [8] Создание АСУ, применяемой для управления, сбора и обработки информации, невозможно без исследования управляемого процесса методами математического моделирования. Например, работа небольшого аэродрома может быть обеспечена силами одного диспетчера, но работа крупного аэропорта требует АСУ, работающей согласно четкому алгоритму.

Система исполнения регламентов (СИР) - автоматизированная информационная система, обеспечивающая сбор, учет и обработку заявлений на предоставление государственных и муниципальных услуг в электронной форме, осуществление межведомственного электронного взаимодействия с органами государственной власти.

Функциональными задачами СИР являются:

1) обеспечение информационного взаимодействия с Федеральной государственной информационной системой "Единый портал государственных и муниципальных услуг" (ЕПГУ);
2) обеспечение информационного взаимодействия с региональной государственной информационной системой "Портал государственных и муниципальных услуг" (далее - РПГУ);
3) обеспечение межведомственного информационного взаимодействия в целях представления и получения документов и информации, в электронной форме с использованием единой системы межведомственного электронного взаимодействия и подключаемых к ней региональных систем межведомственного электронного взаимодействия.

СИР – является неким симбиозом СППР, экспертных систем и АСУ. В ней мы имеем строго определенный регламент, определяющий последовательность действий, условия разрешения проблемных ситуаций и возможные варианты решения. В случае СИР регламент заменяет математическую модель и определяет алгоритм работы системы. В спорных и предусмотренных регламентом ситуациях решение должно приниматься ответственным лицом, а система должна предоставить всю необходимую информацию для помощи в принятии этого решения. В тоже время процессы, четко следующие заданному алгоритму, должны исполняться без привлечения

ответственных лиц, если не предусмотрено обратное, и получать ожидаемый результат. [9]

Глава 2. Разработка программного решения

2.1 Постановка задачи

Конечным результатом должна быть система, обладающая следующим необходимым функционалом:

- Прием заявлений из АИС МФЦ и с РПГУ;
- Хранение исходных данных заявлений, для возможности их проверки;
- Сохранение приложенных фотографий и документов в файловое хранилище и получение из него, уже сохраненных файлов;
- Определение типа заявителя по данным заявителя;
- Формирование списка опрашиваемых внешних сервисов на основе типа заявителя;
- Формирование запросов в соответствии с форматом, требуемым каждым сервисом;
- Отправка и получение результатов синхронных и асинхронных запросов к внешним сервисам;
- Сохранение данных запроса и ответа, для отслеживания ошибок и разрешения спорных ситуаций;
- Подготовка ответов от внешних сервисов для дальнейшего принятия решения о возможности оказания услуги;
- Анализ ответов и принятие решения о возможности оказания услуги;
- Формирование запроса на выпуск ЭТК, в случае положительного решения, и генерация файла уведомления об отказе в оказании услуги, в противном случае.

Помимо описанных требований, также необходимо, чтобы система обладала возможностями:

- Отслеживание процесса обработки дела;
- Отображение списка всех дел;
- Проверка приложенных документов и фотографий ответственными лицами;
- Приложение к делу результатов запросов, отправленных в бумажном варианте;
- Сбор статистики о процессе обработки дела.

Архитектура системы должна соответствовать следующим требованиям:

- Гибкость – архитектура должна допускать простое развитие и изменение функционала, изменение конфигураций в соответствии с требованиями к системе;
- Независимость элементов – отдельные элементы могут быть использованы другими системами;
- Отказоустойчивость – работоспособность всей системы на должна завесить от работоспособности отдельных модулей, и система должна обладать возможностью восстановления после ошибок без прерывания процессов;
- Поддержка одновременного исполнения множества процессов – исполнение одного процесса не должно влиять на выполнение другого процесса (замедлять, приостанавливать, отменять, изменять данные).

2.2 Описание процесса оказания услуги

Услуга выдачи электронной транспортной карты позволяет учащимся школ, ВУЗов, СУЗов и пенсионерам города Тюмени карту Тюменской Транспортной Системы, предоставляющей льготный проезд на общественном транспорте в городе Тюмени.

Процесс оказания услуги разделен на три этапа:

1. Прием заявления и проверка корректности введенных данных;
2. Сбор данных из внешних источников;
3. Принятие решения о возможности оказания услуги.

Прием заявления происходит при личном обращении заявителя в МФЦ или через РПГУ. При личном обращении оператор принимает от заявителя необходимые документы (документ удостоверяющий личность, справку с места учебы, пенсионное удостоверение) и вносит данные в автоматизированную информационную систему МФЦ (АИС МФЦ), делается фото заявителя для печати его на транспортной карте. В этом случае верность данных проверяется оператором, и дополнительные проверки будут излишни. При подаче заявления через РПГУ, заявитель заполняет на сайте форму заявки на оказание услуги и прикрепляет отсканированные документы, подтверждающие правильность введенных данных, а также фотографию заданном формате. В данном случае необходима дополнительная проверка корректности введенных данных заявителя и соответствия формата фотографии. Проверка введенных данных производится ответственными лицами, поэтому необходимо реализовать модуль, позволяющий сверить данные заявления с данными в приложенных документах и проверить приложенную фотографию на соответствие формату (разрешение файла должно быть не менее 300 DPI, соотношение сторон должно быть 3x4, черно-белый снимок должен быть монохромным (8 бит), цветной снимок должен

быть 24-х битным). Заявления из АИС МФЦ и с РПГУ отправляется на сервис приема заявлений и переходит на следующий этап обработки.

На основе полученных данных определяется тип заявителя, в зависимости от которого формируется список сервисов, запросы к которым необходимы для принятия решения о правомерности оказания услуги. Для оказания услуги по выдаче ЭТК школьнику достаточно отправить запрос к сервису департамент образования Тюменской области. В ответе на этот запрос будут возвращены данные о наличии школьника в списке учащихся учебного заведения, класс, и дата окончания обучения. Для студентов, сначала нужно определить тип учебного заведения (СУЗ или ВУЗ). В случае, если заявитель является студентом СУЗа, необходимо выполнить запрос к другому сервису департамента образования, который похожий набор данных (подтверждение наличия заявителя в списке учащихся, курс и дату окончания обучения). Для студентов ВУЗов есть два сервиса для подтверждения наличия студента в учебном заведении: сервис предоставляющий данные о студентах Тюменского государственного университета и общий сервис ВУЗов Тюмени. Поскольку ТюмГУ предоставляет собственный сервис, он прекратил взаимодействие через общий сервис ВУЗов, при этом сервис ТюмГУ отвечает на запросы синхронно, в отличие от общего сервиса ВУЗов, запросы к которому обрабатываются вручную, что значительно ускоряет процесс оказания услуги. При выдаче транспортной карты пенсионеру необходимо выполнить запрос в Пенсионный фонд России (ПФР), и отправить заявку на получение подтверждения о регистрации пенсионера в Тюмени. Так как заявка на получение данных о регистрации отправляется в виде отсканированного напечатанного заявления, результат приходит в том же формате, каждая заявка обрабатывается вручную и все взаимодействие происходит через СЭД «Директум», наиболее простым способом передачи результатов запроса в разрабатываемую систему будет – ручной ввод, через пользовательский

интерфейс. Для всех типов заявителей выполняется запрос на проверку наличия у заявителя активных транспортных карт.

Последний этап, это проверка полученных данных и принятие решения об оказании услуги. Результаты выполнения всех запросов сохраняются в базу данных, а модуль принятия решений отслеживает получение ответов. После получения всех необходимых данных проводится проверка результатов и принятие решения об оказании услуги заявителю. В случае положительного решения формируется запрос к сервису Тюменской Транспортной Системы в, котором передаются данные из заявления и результатов запросов и фото заявителя. В противном случае формируется уведомление об отказе в оказании услуги, в котором должны быть перечислены все причины отказа, для дальнейшего их устранения заявителем. Уведомление об отказе распечатывается и выдается заявителю, поэтому необходимо реализовать модуль генерации печатных форм в соответствии с заданным форматом.

2.3 Построение архитектуры системы

На основе описанного процесса оказания услуги, а также с учетом требований к необходимо разработать архитектуру системы.

Первый этап – получение данных заявления. Для того, чтобы унифицировать процесс приема заявлений, был разработан микросервис, принимающий данные заявителя в формате XML. Этот сервис преобразует данные в «плоский» JSON-объект, т.е. JSON, представляющий собой Map, в котором в качестве ключа используется имя переменной, в формате: ***Объект_родитель.Имя_переменной***. Файлы, приложенные заявителем при обращении через сайт РПГУ, представлены в виде массива байт и закодированы в base64. Микросервис декодирует файл и сохраняет полученный массив байт в файловое хранилище, добавляя описание файла (имя файла, размер и тип). В результате будет получен уникальный

идентификатор файла, который сохраняется вместо него в JSON-объект, и в дальнейшем при необходимости получения файла, он будет загружаться из файлового хранилища по указанному идентификатору. Это позволяет уменьшить размер хранимых в базе данных объектов. Для заявлений, полученных из АИС МФЦ, выполнение этой процедуры не требуется, т.к. файлы сразу загружаются в файловое хранилище и вместо файла сразу получается его идентификатор. Использование очереди сообщений в качестве канала передачи, позволяет снизить нагрузку на базу данных при большом количестве запросов и изолировать сервис приема заявлений от остальной системы. В случае, если база данных или вся система будет не доступна, это позволит продолжить прием заявлений и обработать их позже, после восстановления работоспособности. Сервис регистрации заявлений получает сообщения из очереди, по мере их поступления. Так как в заявлении могут содержатся данные не используемые в процессе оказания услуги, например, данные представителя, сервис регистрации выделяет набор необходимых данных, формируя более лаконичный и более удобный для использования JSON-объект, который представлен в объектном формате (примеры таких объектов приведены в Приложении 1). Затем пара JSON-объектов сохраняется в базу данных заявлений. Упрощенный вариант будет использоваться в процессе обработки заявления, а исходный хранится для исключения потери какой-либо информации. Так же сервис регистрации создает заявление в АИС МФЦ для дальнейшей выдачи результатов оказания услуги в МФЦ. Схема движения данных и взаимодействия микросервисов представлена на схеме ниже (Рис.5)

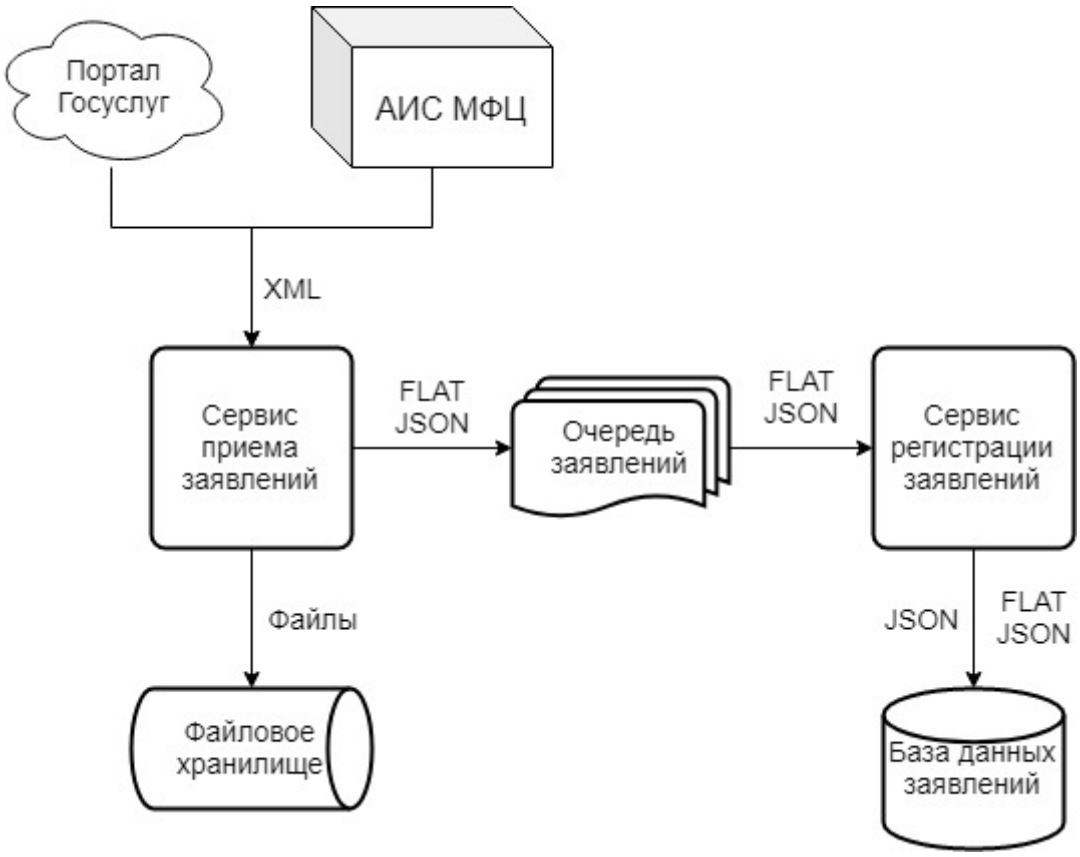


Рис. 1 Этап получения заявлений

В случае если заявление поступило с РПГУ, необходимо проверить корректность введенных данных. Такое заявление сохраняется в отдельную таблицу базы данных, и после его проверки ответственным лицом, с помощью сервиса для ввода пользовательских данных, попадают в общий список заявлений.

Сбор данных из внешних источников начинается с получения заявления из базы данных сервисом построения запросов, который периодически опрашивает базу данных на предмет наличия новых заявлений. Обнаружив новое заявление сервис формирует запросы к внешним источникам данных выбирая необходимую информацию и JSON-объекта и формирует новый JSON-запрос, который отправляется в сервис взаимодействия с внешними источниками. Сервис взаимодействия с внешними источниками использует внутри себя очередь сообщений (аналогичную той, что используется для получения заявлений). JSON-запрос отправляется соответствующий канал

очереди сообщений сервиса взаимодействия с внешними источниками и содержит данные необходимые для получения сведений из этого источника. Клиентская часть внешнего источника самостоятельно получает сообщения из очереди и сохраняет результаты в отдельный канал.

Внешние источники – это сервисы, реализованные по стандарту СМЭВ (Система межведомственного электронного взаимодействия).

Функции СМЭВ:

- Ведение реестра электронных сервисов;
- Ведение политик безопасности, применяемых к участникам взаимодействия;
- Маршрутизация сообщений к зарегистрированным электронным сервисам и их пользователям при синхронном и асинхронном взаимодействии;
- Протоколирование обращений (входящих и исходящий сообщений) к электронным сервисам;
- Гарантированная доставка сообщений, обеспечиваемая повторными вызовами электронных сервисов при сбоях;
- Обеспечение оповещения о сбоях в функционировании электронных сервисов;
- Передача информации о событиях на СМЭВ по подписке заинтересованным пользователям;
- Формирование динамически создаваемой статистики использования электронных сервисов;
- Подписание электронных сообщений электронной подписью;
- Форматно-логический контроль входящих сообщений;
- Контроль и мониторинг процессов межведомственного обмена с использованием СМЭВ.

Передача данных по этому стандарту происходит в виде сообщений в формате XML по протоколу SOAP. Набор и формат представления данных определяется поставщиком сервиса, однако стандарт СМЭВ накладывает условия на структуру сообщения. [1] Общая структура SOAP-запроса к системе электронного межведомственного взаимодействия приведена в приложении 2.

Сервис взаимодействия с внешними источниками преобразовывает полученные JSON-запросы в требуемый формат, отправляет их и ожидает ответ от сервиса, который преобразуется в JSON-ответ и сохраняется в базу данных для дальнейшего использования системой принятия решений. Запросы к сервисам СМЭВ сохраняются в базу данных в формате XML сообщений, это необходимо для разрешения возможных проблем, которые могут возникнуть при взаимодействии с сервисами.

Так как взаимодействие с внешним источником может происходить путем запроса данных с помощью бумажных заявлений, ответы на которые так же будут представлены на бумаге, наиболее оптимальным вариантом будет реализация ввода этих данных ответственными лицами через сервис ввода данных пользователя. Ниже представлена схема взаимодействия сервисов на этом этапе (Рис 6).

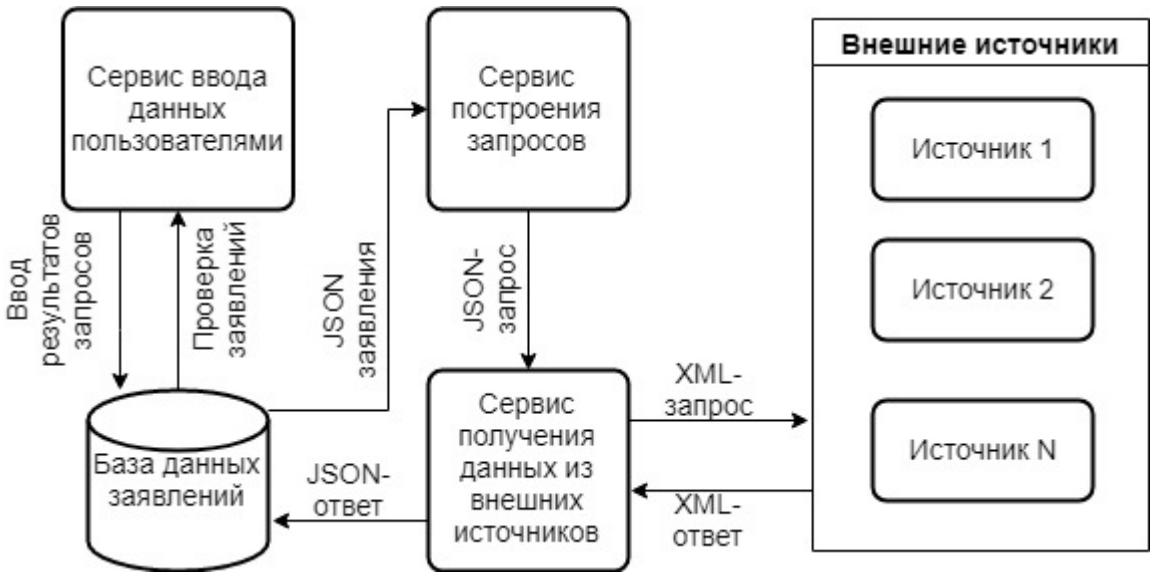


Рис. 2 Этап получения данных из внешних источников

На заключительном этапе, сервис обработки результатов периодически опрашивая базу отслеживает процесс получения необходимых данных для принятия решения о возможности оказания услуги из внешних источников. Набор данных определяется на основе данных заявления (тип получаемой карты, место учебы и других). Получив из базы все необходимые данные, сервис обработки результатов сопоставляет их эталонными данными, при которых оказание услуги возможно. Если все данные совпали с эталонами, формируется запрос на выпуск Электронной транспортной карты, который так же отправляется во внешнюю систему по стандарту СМЭВ. В противном случае список данных не совпавших с эталонами отправляется в сервис генерации уведомления об отказе, где формируется документ, содержащий список причин из-за, которых услуга не может быть оказана. Ниже приведена схема последнего этапа обработки заявления (Рис.7).

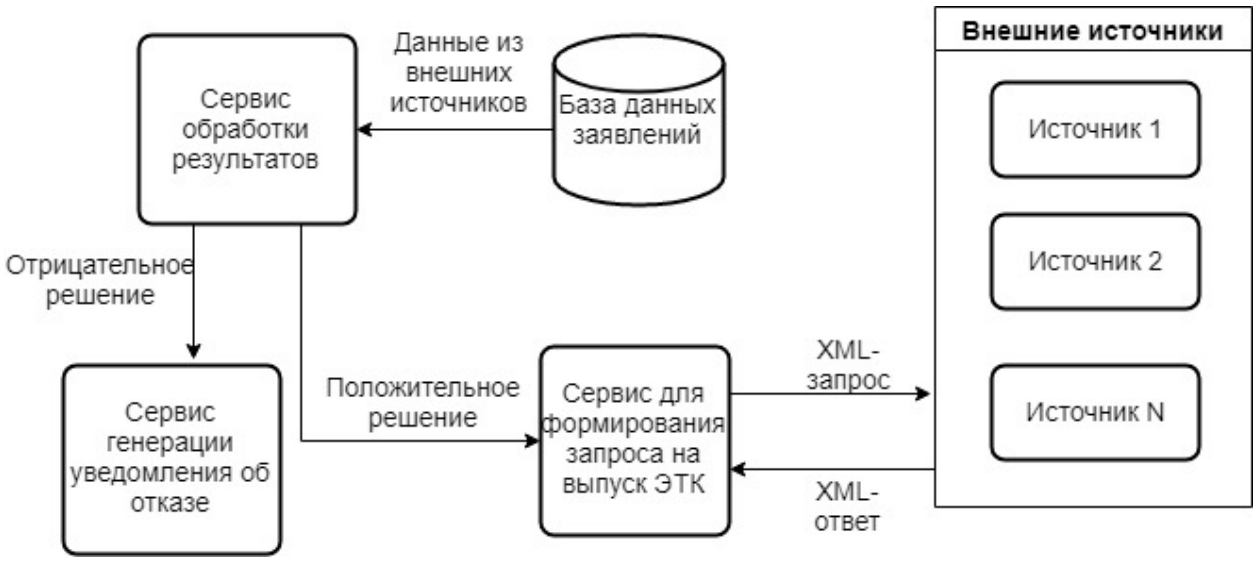


Рис. 3 Этап принятия решения об оказании услуги

Полученная архитектура приложения (Рис. 4), соответствует заявленным требованиям. Гибкость и независимость отдельных элементов достигается за счет использования микросервисного подхода, поскольку он позволяет легко изменять отдельные части системы внося изменения только в реализацию микросервиса, а их распределенность позволяет использовать их извне при необходимости. Отказоустойчивость так же обеспечивается за счет использования микросервисов, поскольку выход и строя одного не скажется на работе остальных, и за счет использования очередей сообщений, которые позволяют избежать потери данных при передаче их между сервисами. Упрощение сервисов позволяет быстро выполнять стоящие перед ними задачи, за счет чего время выполнения процесса складывается по большей части из времени ожидания ответов из внешних источников. Использование микросервисов реализует фабричную обработку заявления, т.е. после обработки одного заявления сервисом он готов к обработке следующего, а это значит, что одновременно в работе может находится несколько процессов обработки заявлений.

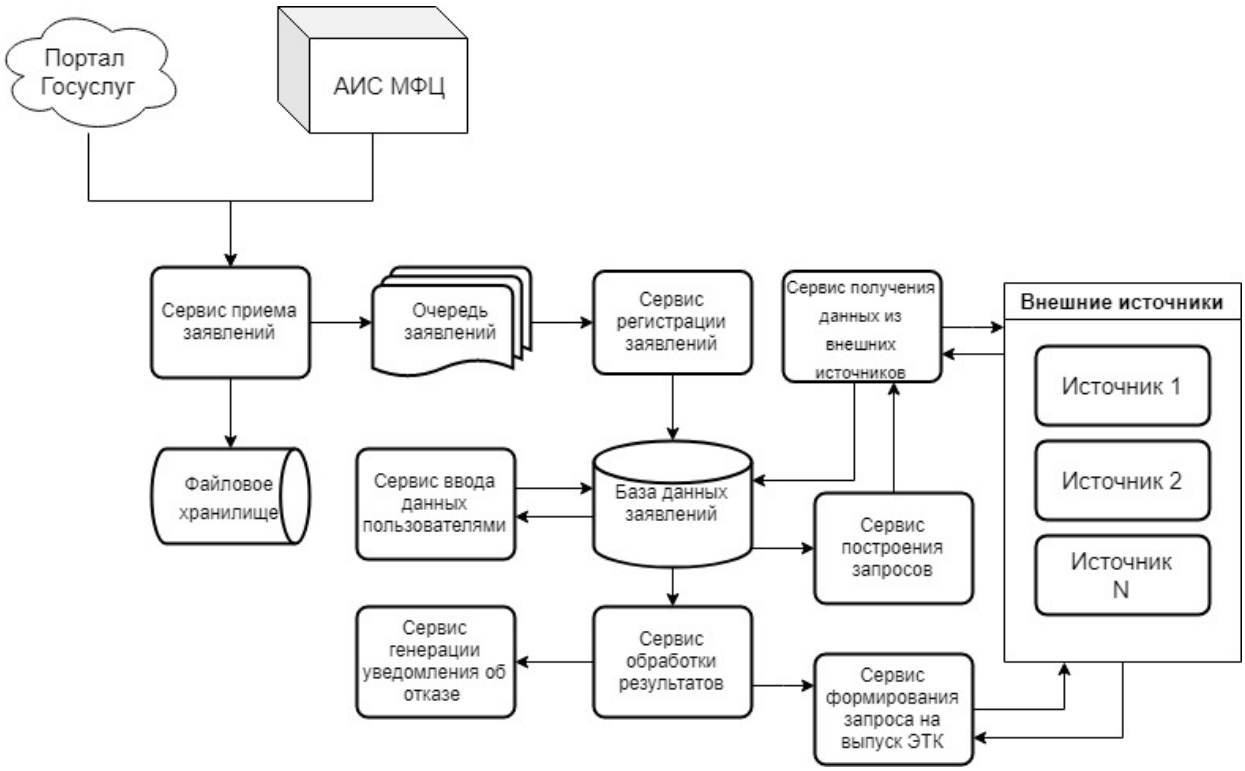


Рис. 4 Общая схема взаимодействия сервисов в системе

2.3 Обзор инструментов разработки

Для реализации системы был выбран язык программирования Kotlin. Это молодой статически типизированный язык программирования, компилируемый в JVM (Java Virtual Machine) байт-код, и поддерживающие как объектно-ориентированный, так и процедурный стили программирования. Выбор языка обусловлен следующими причинами:

- используется для разработки в ГАУ ТО МФЦ;
- полностью совместим с языком Java;
- поддерживает большое количество библиотек и фреймворков для Java;
- хорошо подходит для реализации серверных приложений;
- язык NULL-безопасен, что позволяет исключить возникновение ошибок типа NullPointerException еще на этапе написания кода;

- легкочитаемый и приятный синтаксис;
- автоматическая конвертация JAVA-кода в Kotlin и обратно в IDE InteleJ Idea;
- развивающийся язык, который перенимает лучшие практики у других. [13]

В качестве базы данных используется PostgreSQL 11, т.к. эта СУБД используется в ГАУ ТО МФЦ во многих проектах, где требуется реляционная база данных, а также обладает рядом преимуществ.

Одним из них является ее архитектура. Как и многие другие СУБД, PostgreSQL может применяться в среде клиент-сервер, что дает массу преимуществ пользователям и разработчикам.

Основу PostgreSQL составляет серверный процесс. Он выполняется на одном сервере. Доступ из приложений к данным базы осуществляется посредством процесса базы данных. Клиентские программы не могут получить доступ к данным самостоятельно, даже если они работают на том же компьютере, на котором выполняется серверный процесс.

Такое разделение позволяет построить распределенную систему. Можно отделить клиентов от сервера посредством сети и разрабатывать клиентские приложения в среде, удобной для пользователя.

PostgreSQL обладает поддержкой UUID, денежного, перечисляемого, геометрического, бинарного типов, сетевых адресов, битовых строк, текстового поиска, XML, JSON, массивов, композитных типов и диапазонов, а также некоторых внутренних типов для идентификации объектов и местоположения логов. Стоит сказать, что другие СУБД тоже имеют некоторые из этих типов данных, но только PostgreSQL поддерживает их все.

Наиболее важным в данном случае будет поддержка типов JSON и UUID, поскольку для генерации идентификаторов в таблицах используется UUID, а в формате JSON хранятся исходные данные заявлений, которые в дальнейшем преобразуются в удобный для использования формат.

Тип данных JSON обеспечивает проверку корректности JSON, который позволяет использовать специализированные JSON операторы и функции, встроенные в PostgreSQL для выполнения запросов и манипулирования данными. Также доступен тип JSONB — двоичная разновидность формата JSON, у которой пробелы удаляются, сортировка объектов не сохраняется, вместо этого они хранятся наиболее оптимальным образом, и сохраняется только последнее значение для ключей-дубликатов. JSONB обычно является предпочтительным форматом, поскольку требует меньше места для объектов, может быть проиндексирован и обрабатывается быстрее, так как не требует повторного синтаксического анализа. [12]

В качестве очереди сообщений используется – RabbitMQ. Это легковесный, легко разворачиваемый брокер сообщений, поддерживающий множество почтовых протоколов. Помимо этого, приложение реализовано на платформе Erlang/OTP, что гарантирует максимальную стабильность и масштабируемость очереди, как ключевого узла системы. Так же стоит учитывать тот факт, что приложение распространяется по лицензии Mozilla Public License и реализует открытый протокол AMQP, библиотеки для которого существуют во всех основных языках и платформах программирования.

Используемые библиотеки и фреймворки:

Spring Framework – универсальный и популярный фреймворк для Java-платформы, содержащий почти все модули необходимы для реализации приложений:

- Spring Core – поддержка аспектно-ориентированного программирования, инъекции зависимостей, data binding, и другие возможности;
- Spring Testing – модуль тестирования, поддерживающий корректное использование инъекции зависимостей и data binding, а так же сочетающийся с другими модулями Spring Framework;
- Spring Data Access – модуль для взаимодействия с базами данных содержащий реализации стандартов JDBC, ORM и поддержку транзакций;
- Spring MVC и Spring WebFlux – модули для реализации веб приложений. [4,11]

Axon Framework – фреймворк поддерживающий: CQRS, Event Sourcing, Domain Driven Design и Event Driven Microservices.

CQRS – разделяет обработку запросов на чтение и запись данных, т.е. создаются отдельные представления только для чтения и обновляя их при записи данных в таблицы. Таким образом уменьшается вероятность появления «тупиков» (deadlock) связанных с блокировки частей таблиц при записи и одновременной попыткой чтения данных из этой части.

Event Sourcing – это подход к реализации приложений, при котором состояние приложения хранится как цепочка событий. Основным плюсом этого подхода является возможность восстановить состояние процесса на любом этапе по хранимым событиям, так же это позволяет восстанавливать данные при их утере или переносе.

DDD Aggregate – доменный объект (domain object), над которым производятся все действия в процессе. Он реагирует на команды, получаемые от управляющих им объектов, и события на отслеживание, которых он

«подписан». Именно агрегат определяет состояния исполняемого процесса и именно он обрабатывает цепочку событий при использовании Event Sourcing.

Driven Microservices – позволяет использовать команды и события для управления системой на основе микросервисной архитектуры, что упрощает возможность ее масштабирования. Весь процесс строится на последовательности команд и событий, вызванных выполнением этих команд. Все обработчики событий реагируют на появление события конкретного типа и выполняют некоторые действия.

XDocReport – открытая библиотека позволяющая работать с документами в формате XML и генерировать отчеты в форматах PDF, XHTML и других. В системе она используется для генерации уведомлений об отказе в формате PDF, для дальнейшей печати и выдачи заявителю.

Jackson – библиотека для работы с JSON-объектами. Используется для отображения JSON’а на некоторый класс, для удобства использования.

Apache CXF – фреймворк обеспечивающий работу протокола SOAP. Так как внешние источники передают данные по стандарту СМЭВ, который в свою очередь передает данные по протоколу SOAP, данный фреймворк не обходим для реализации клиентской части сервисов СМЭВ.

Hibernate – библиотека реализующая стандарт Java Persistence API (JPA), который представляет собой спецификацию обеспечивающую объектно-реляционное отображение простых Java объектов и предоставляющая API для управления такими объектами. JPA ни сохраняет, ни управлять объектами, JPA только определяет правила: как что-то будет действовать. JPA также определяет интерфейсы, которые должны будут быть реализованы провайдерами. Также он определяет описание метаданных отображения и работу провайдеров. Каждый провайдер, реализуя JPA самостоятельно определяет реализацию получения, сохранения и управления объектами.

2.4 Реализация системы

Использование Axon Framework накладывает некоторые условия на реализацию процесса оказания услуги. Согласно концепции Domain Driven Development (DDD) при проектировании описывается единственная модель, описывающая всю предметную область. Основным элементом модели является агрегат – сущность, к которой обращаются потребители. Агрегат содержит необходимые данные, а также обработчики команд и событий, которые связаны с этим агрегатом.

Главным агрегатом в системе является заявление (класс Order код, которого приведен в Приложении 3), именно вокруг него строится весь процесс. Сам агрегат хранит только идентификатор заявления в базе данных, данные по которому получаются из базы и изменяются внешними сервисами в процессе обработки заявления. Агрегат способен реагировать на команды, получаемые им через шину команд (CommandBus), которые добавляются туда другими классами по мере необходимости. Методы обработки команд помечены аннотацией @CommandHandler. Команды могут обрабатываться только агрегатами, поэтому для взаимодействия с другими классами используются события (Event), которые могут быть получены и обработаны как агрегатами, так и другими классами. Так как подход Event Sourcing используемый при работе с Axon Framework, предполагает хранение не конечных данных, а последовательности событий, которая привела к получению этих данных, обработчики событий в агрегатах предназначены для изменения данных в самом агрегате и восстановления состояния агрегата по набору связанных с ним событий.

Так как агрегат всего лишь сущность с данными, обладающая возможностью реагировать на команды изменением данных внутри агрегата и отправлять события об их изменениях, он ничего не знает о процессе. Поэтому

необходимо, чтобы был объект знающий последовательность действий в процессе и управляющий им. В Axon Framework такие объекты называются сагами (Saga). Сага начинает процесс, выполнив метод помеченный аннотацией @StartSaga и отправкой команду на создание агрегата. Далее она реагирует на события, полученные от агрегатов, других сервисов и саг. Таким образом все действия, которые необходимо выполнить по ходу процесса определены в саге, а она лишь отправляет агрегату команду на изменение. Сага управляющая процессом обработки дела представлена в Приложении 4.

Каждый микросервис в системе представляет собой пару вида сага-агрегат. Передача управления и данных между ними осуществляется через EventBus – шину, которая хранит все события в базе данных и оповещает всех «слушателей» о появлении события.

2.5 Разбор данных из внешних источников и принятие решения об оказании услуги

Для принятия решения об оказании услуги необходимо, чтобы все данные из внешних источников соответствовали определенным условиям, описанным в регламенте об оказании услуги. Для каждого источника установлен, свой набор правил, поэтому более рационально вынести разбор данных в клиентскую часть сервисов для запроса данных из внешних источников.

После получения ответа на запрос клиентская часть проецирует полученное сообщение на класс удобный для разбора дальнейшей работы для принятия проверки сообщения. Затем перед передачей ответа в систему производится проверка всех необходимых данных. Например, в ответе сервиса о наличии школьника в учебном заведении будет содержатся поле определяющее соответствие данных переданных в запросе с данными хранящимися в базе Департамента образования Тюменской области. Другими

словами, на запрос является ли школьник N учащимся школы M будет дан однозначный ответ – да или нет. Ответы других сервисов не так однозначны, поэтому для каждого сервиса необходимо сформировать модель, на основе которой будет получен однозначный результат. Этот результат записывается в объект ответа, который будет передан в систему для дальнейшей обработки. Поскольку кроме простого ответа для оказания услуги могут потребоваться дополнительные сведения, предоставляемые сервисами, (например, дата окончания обучения) после применения модели для проверки результата исходный ответ сервиса преобразовывается в формат JSON, для унификации передаваемых результатов, и так же записывается в объект ответа.

Сага, отслеживающая процесс получения данных из внешних источников, сначала формирует список опрашиваемых источников. Полученный список передается в сервис формирования запросов к внешним источникам, который формирует и отправляет запросы к каждому источнику, находящемуся в списке. После получения всех результатов формирует список источников, данные из которых были не подтверждены или не получены. Если этот список пуст, то оснований для отказа в оказании услуги – нет, и управление передается сервису формирования запроса на выпуск ЭТК, в противном случае список отправляется в сервис генерации уведомления об отказе. Сервис генерации уведомлений об отказе хранит шаблон документа в формате ODT и, на основе полученного списка, заполняет его, описывая для заявителя проблемы возникшие при обработке заявления, чтобы он мог устранить все недочеты и подать заявление повторно.

Данный подход позволяет легко изменять набор опрашиваемых внешних источников, добавляя или убирая их из списка опрашиваемых. При этом следует лишь реализовать проверку всех необходимых условий на клиентской стороне сервиса и привести ответ к общему формату.

2.6 Описание интерфейса пользователя

Пользователи системы должны иметь возможность:

- Просматривать список всех заявлений;
- Отслеживать процесс обработки дел;
- Прикладывать результаты межведомственных запросов, отправляемых в бумажном формате;
- Проверять формат фотографий и правильность данных, при получении заявлений через РПГУ.

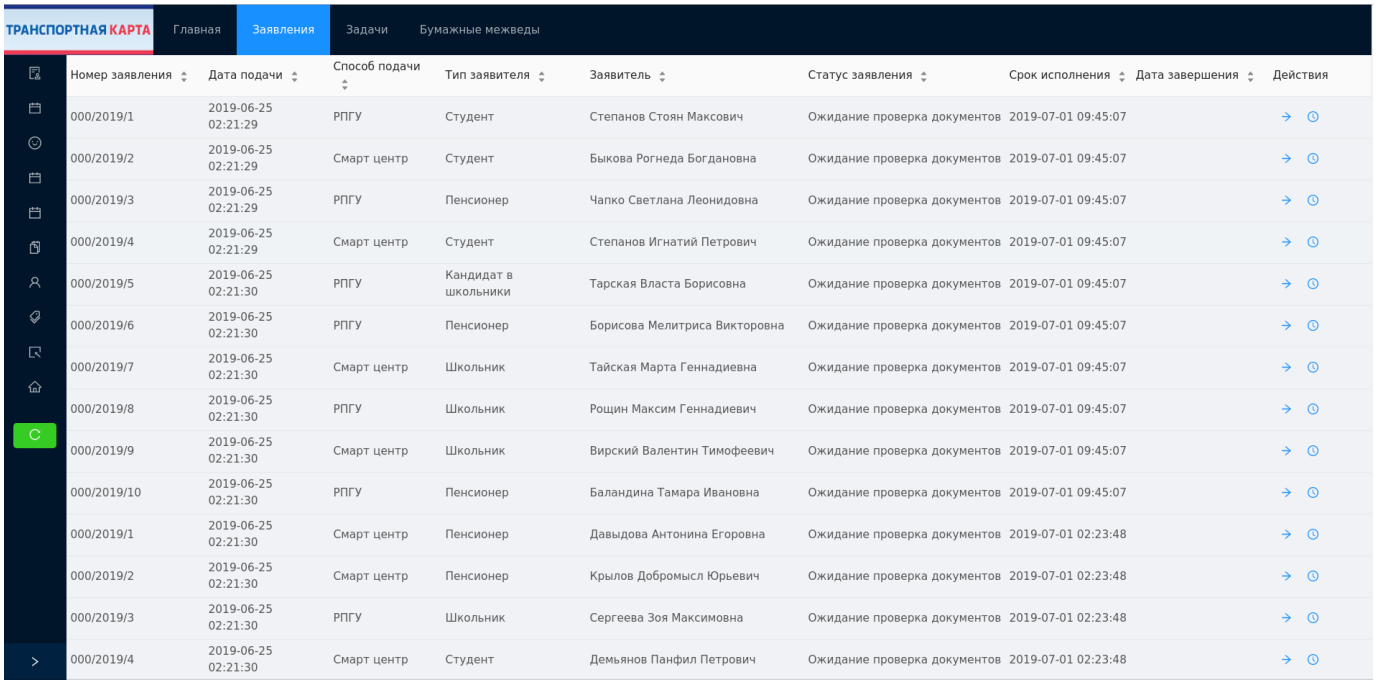
Для этого реализован сервис моделей представления. Этот сервис получает REST – запрос с набором условий, который формирует набор спецификаций JPA. Спецификация (JPA Specification) – представляет собой предикат над классом-сущностью (Entity), который описывает используемую таблицу базы данных. Спецификации реализуют интерфейс критериев (Criteria API), и интерпретируют предикаты в условия SQL-запроса, благодаря чему можно строить запросы к базе данных с помощью программного кода, не используя шаблонизацию запросов. По полученному набору спецификаций, формируется SQL-запрос к базе данных.

Все данные для отображения на web-формах находятся в специальных таблицах-представлениях, содержащих только те данные, которые необходимы пользователям при работе с интерфейсом.

Данные в таблице обновляются при получении в обработчике событий сервиса моделей представления. Axon Framework хранит данные в собственной таблице базы данных, а шина для получения событий (Event Bus) подключается к ней, и отслеживает изменения в таблице. В каждом микросервисе есть свой экземпляр Event Bus, который и передает информацию

о событиях сервису. Такой подход позволяет использовать микросервисную архитектуру, используя для управления и передачи данных события.

Согласно описанным требованиям был разработан web-интерфейс приложения. Ниже (Рис. 5) представлена главная форма системы. На ней отображается список заявлений. По каждому заявлению предоставляется информация о: номере заявления в системе АИС «МФЦ» или на РПГУ, дате подаче заявления, источнике заявления, типе и ФИО заявителя, статусе дела, планируемой и фактической дате завершения. С помощью панели слева можно настраивать фильтры для отображения списка заявлений, а также искать конкретное заявление по номеру.



Номер заявления	Дата подачи	Способ подачи	Тип заявителя	Заявитель	Статус заявления	Срок исполнения	Дата завершения	Действия
000/2019/1	2019-06-25 02:21:29	РПГУ	Студент	Степанов Стоян Максович	Ожидание проверка документов	2019-07-01 09:45:07		
000/2019/2	2019-06-25 02:21:29	Смарт центр	Студент	Быкова Рогнеда Богдановна	Ожидание проверка документов	2019-07-01 09:45:07		
000/2019/3	2019-06-25 02:21:29	РПГУ	Пенсионер	Чапко Светлана Леонидовна	Ожидание проверка документов	2019-07-01 09:45:07		
000/2019/4	2019-06-25 02:21:29	Смарт центр	Студент	Степанов Игнатий Петрович	Ожидание проверка документов	2019-07-01 09:45:07		
000/2019/5	2019-06-25 02:21:30	РПГУ	Кандидат в школьники	Тарская Власта Борисовна	Ожидание проверка документов	2019-07-01 09:45:07		
000/2019/6	2019-06-25 02:21:30	РПГУ	Пенсионер	Борисова Мелитриса Викторовна	Ожидание проверка документов	2019-07-01 09:45:07		
000/2019/7	2019-06-25 02:21:30	Смарт центр	Школьник	Тайская Марта Геннадиевна	Ожидание проверка документов	2019-07-01 09:45:07		
000/2019/8	2019-06-25 02:21:30	РПГУ	Школьник	Рощин Максим Геннадиевич	Ожидание проверка документов	2019-07-01 09:45:07		
000/2019/9	2019-06-25 02:21:30	Смарт центр	Школьник	Вирский Валентин Тимофеевич	Ожидание проверка документов	2019-07-01 09:45:07		
000/2019/10	2019-06-25 02:21:30	РПГУ	Пенсионер	Баландина Тамара Ивановна	Ожидание проверка документов	2019-07-01 09:45:07		
000/2019/1	2019-06-25 02:21:30	Смарт центр	Пенсионер	Давыдова Антонина Егоровна	Ожидание проверка документов	2019-07-01 02:23:48		
000/2019/2	2019-06-25 02:21:30	Смарт центр	Пенсионер	Крылов Добромысл Юрьевич	Ожидание проверка документов	2019-07-01 02:23:48		
000/2019/3	2019-06-25 02:21:30	РПГУ	Школьник	Сергеева Зоя Максимовна	Ожидание проверка документов	2019-07-01 02:23:48		
000/2019/4	2019-06-25 02:21:30	Смарт центр	Студент	Демьянов Панфил Петрович	Ожидание проверка документов	2019-07-01 02:23:48		

Рис. 5 Форма списка заявлений

Для внесения данных межведомственных запросов, отправляемых в бумажном формате, используется форма, представленная на ниже (Рис.6).

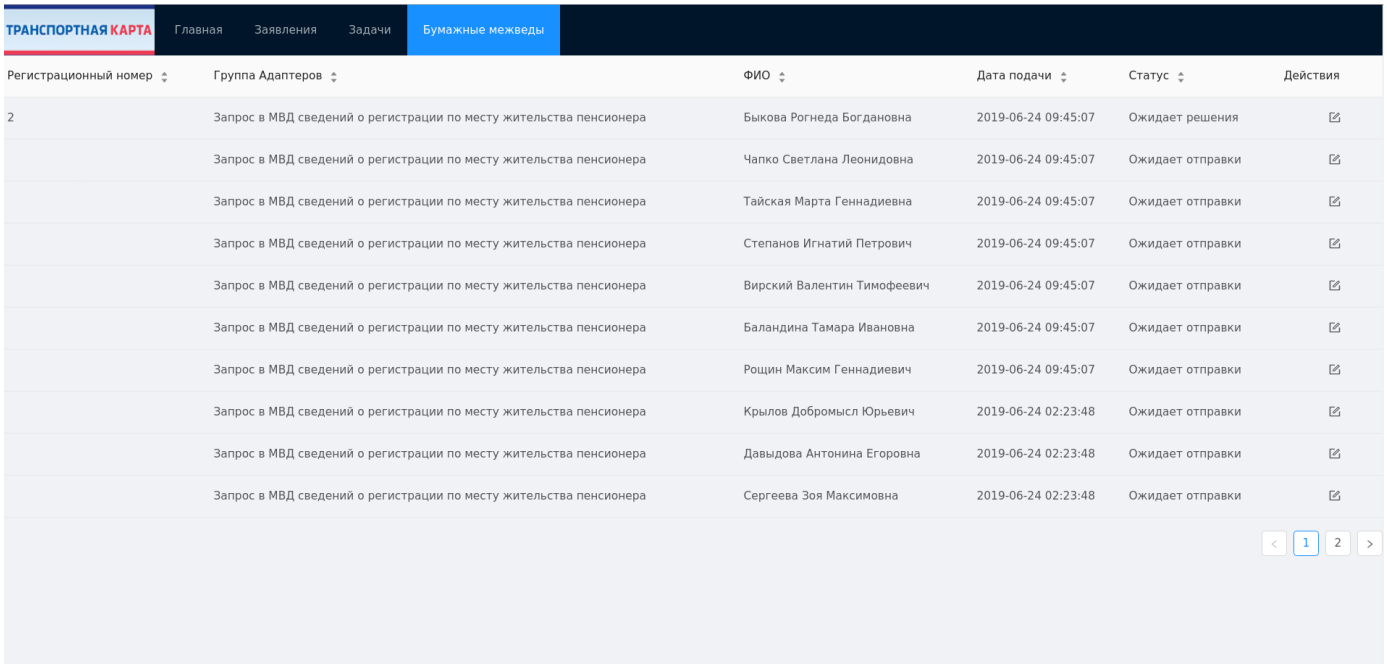


Рис. 6 Форма обработки бумажных межведомственных запросов

На форме отображается список незавершенных запросов. По каждому запросу предоставляются: регистрационный номер запроса, полученный от органа государственной власти, тип запроса, ФИО заявителя, дата подачи заявления и статус обработки запроса. С помощью кнопки действий можно задать регистрационный номер запроса после его получения от ОГВ (Рис.7), если он не был задан ранее, тем самым зарегистрировать его в системе.

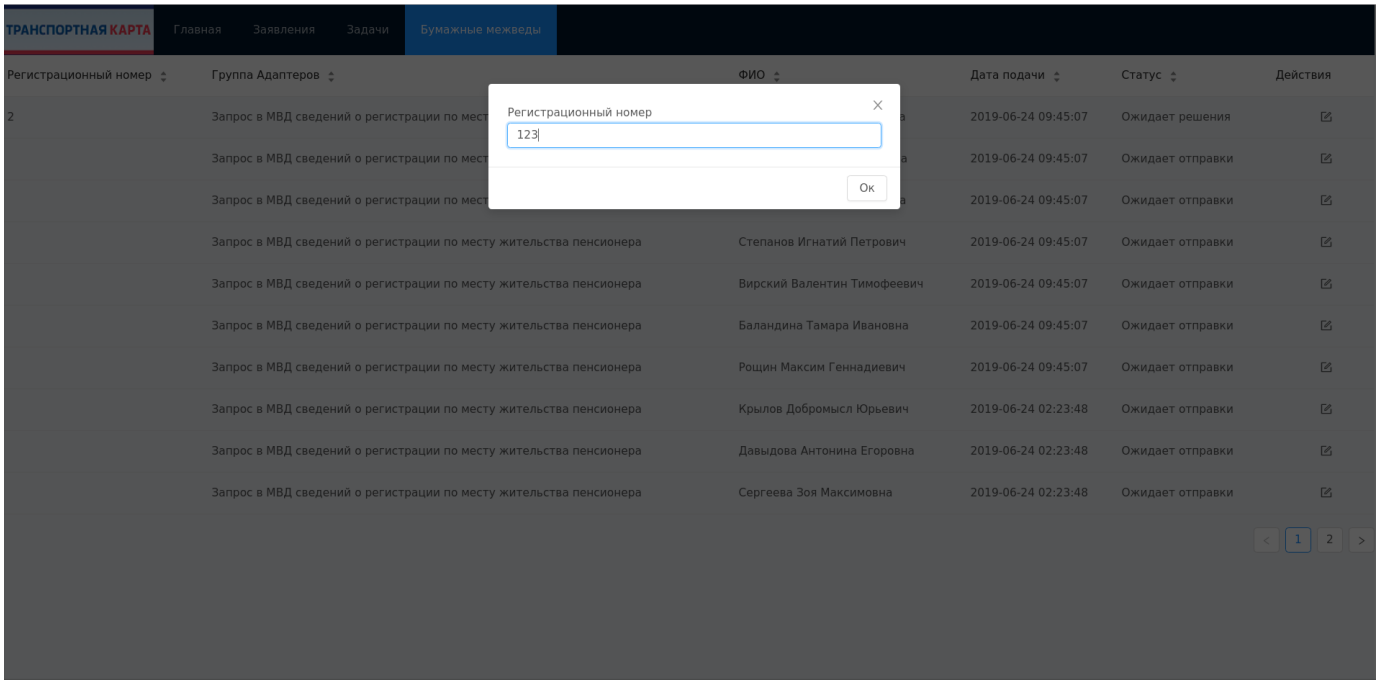


Рис. 7 Регистрация запроса в системе

Если запрос уже зарегистрирован, то кнопка действий позволит записать результат выполнения запроса (Рис.8).

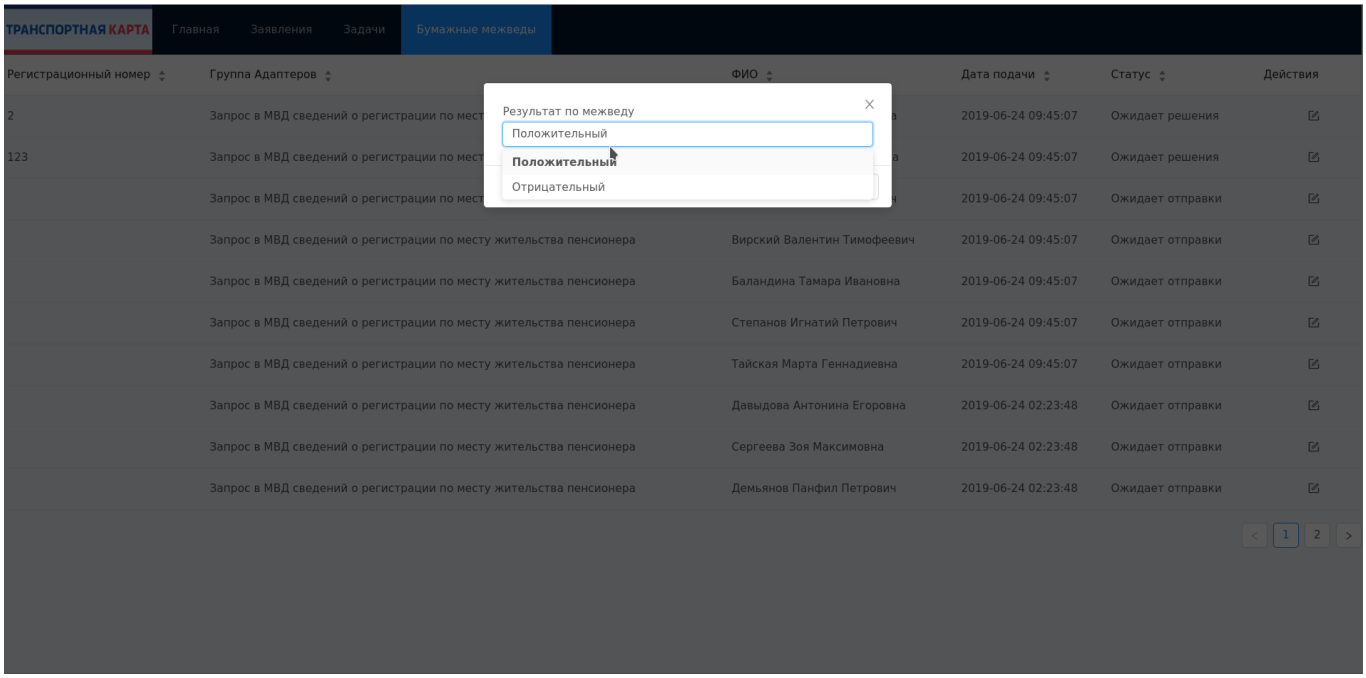


Рис. 8 Занесение результата запроса

Так же интерфейс пользователя позволяет брать задания на проверку данных пользователя. Есть два типа заданий:

- Проверка фотографии пользователя на соответствие требуемому формату
- Сверка данных пользователя с отсканированным экземпляром документа

На изображении (Рис.9) представлено задание на проверку фотографии. Рядом с фото приведен список критериев, по которым необходимо проверить фото, для облегчения работы ответственных лиц.

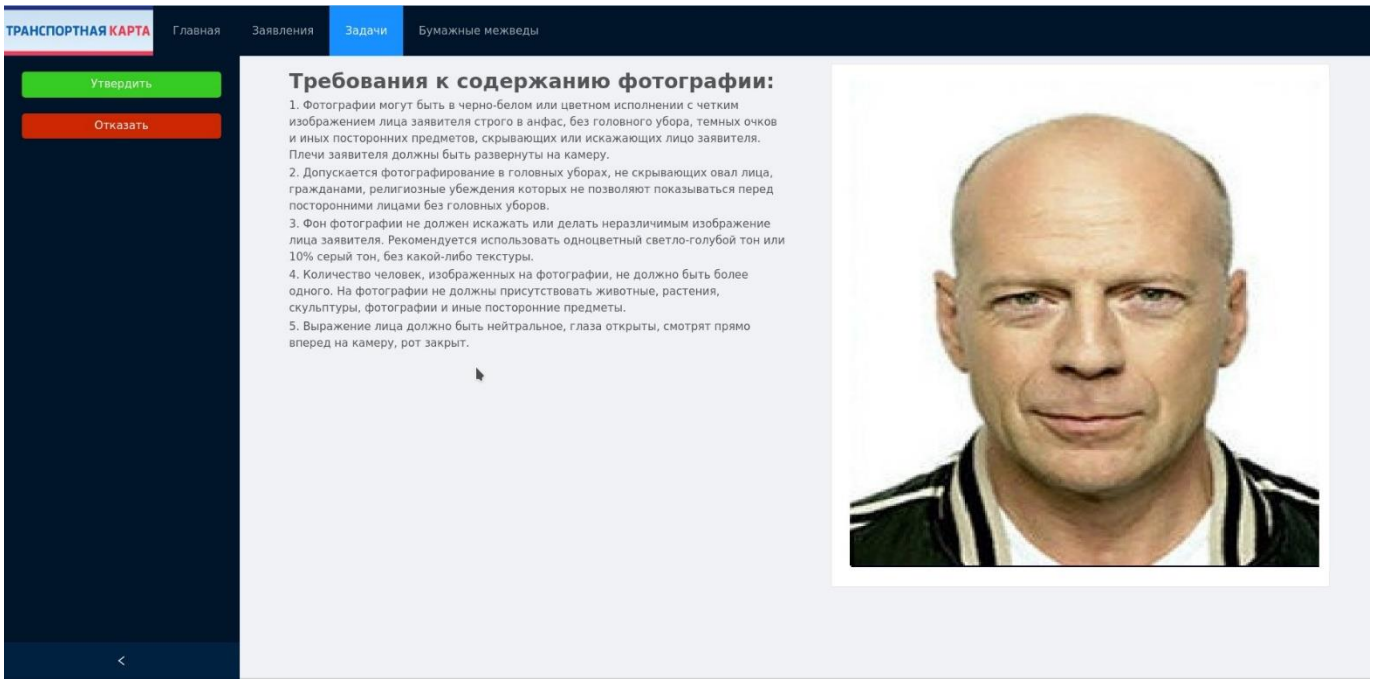


Рис. 9 Форма проверки формата фотографии

Второй тип заданий – это задания на проверку правильности ввода данных заявителя по документу, подтверждающему эти данные. На форме (Рис. 10) приведены необходимые данные для сверки и ссылка на скачивание документа для сверки.

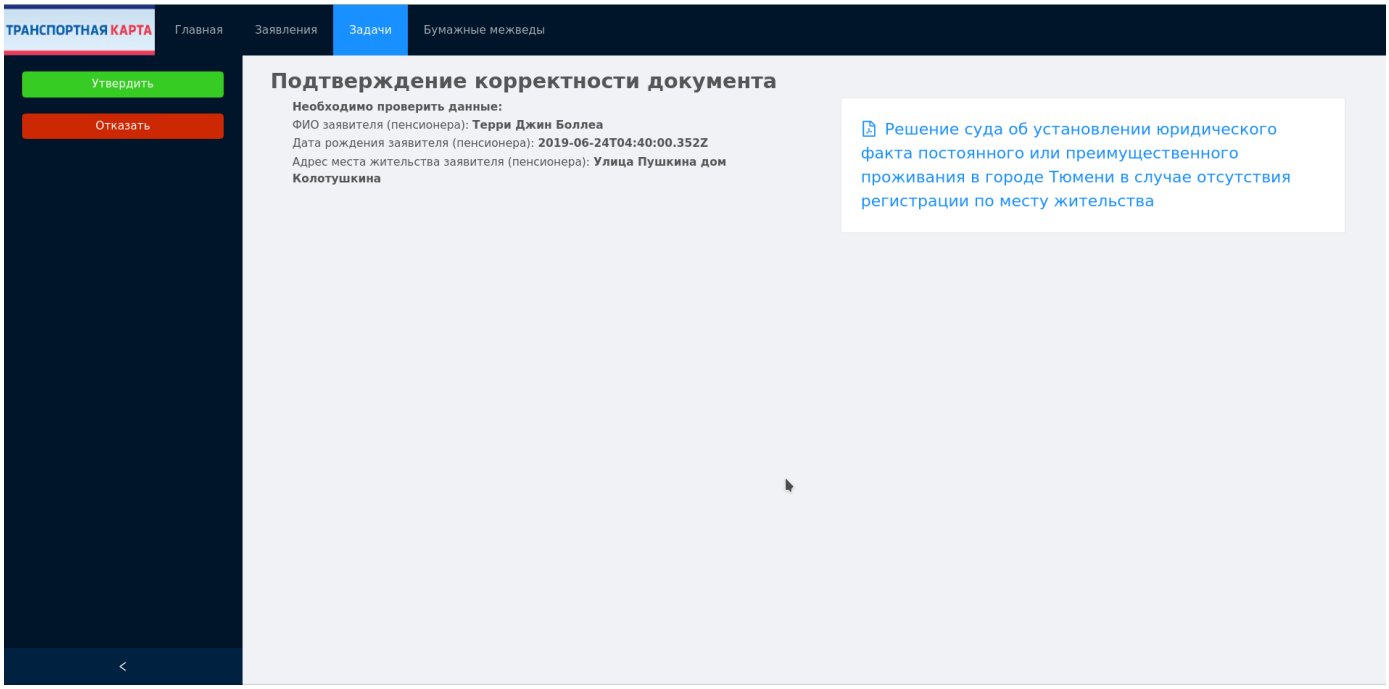


Рис. 10 Форма проверки корректности введенных данных

Кнопки «Утвердить» и «Отказать» сохраняют соответствующие результаты выполнения задания в базу данных. Далее эти результаты обрабатываются и передаются модулю принятия решения, который учитывает их при вынесении решения о возможности оказания услуги.

2.7 Описание схемы базы данных

В системе используется база данных PostgerSQL. В ней хранятся все события, генерируемые Axon Framework, данные о заданиях для пользователей, данные о заявления и так далее. На изображении, приведенном ниже представлена схема базы данных, используемая Axon Framewok (Рис.11).

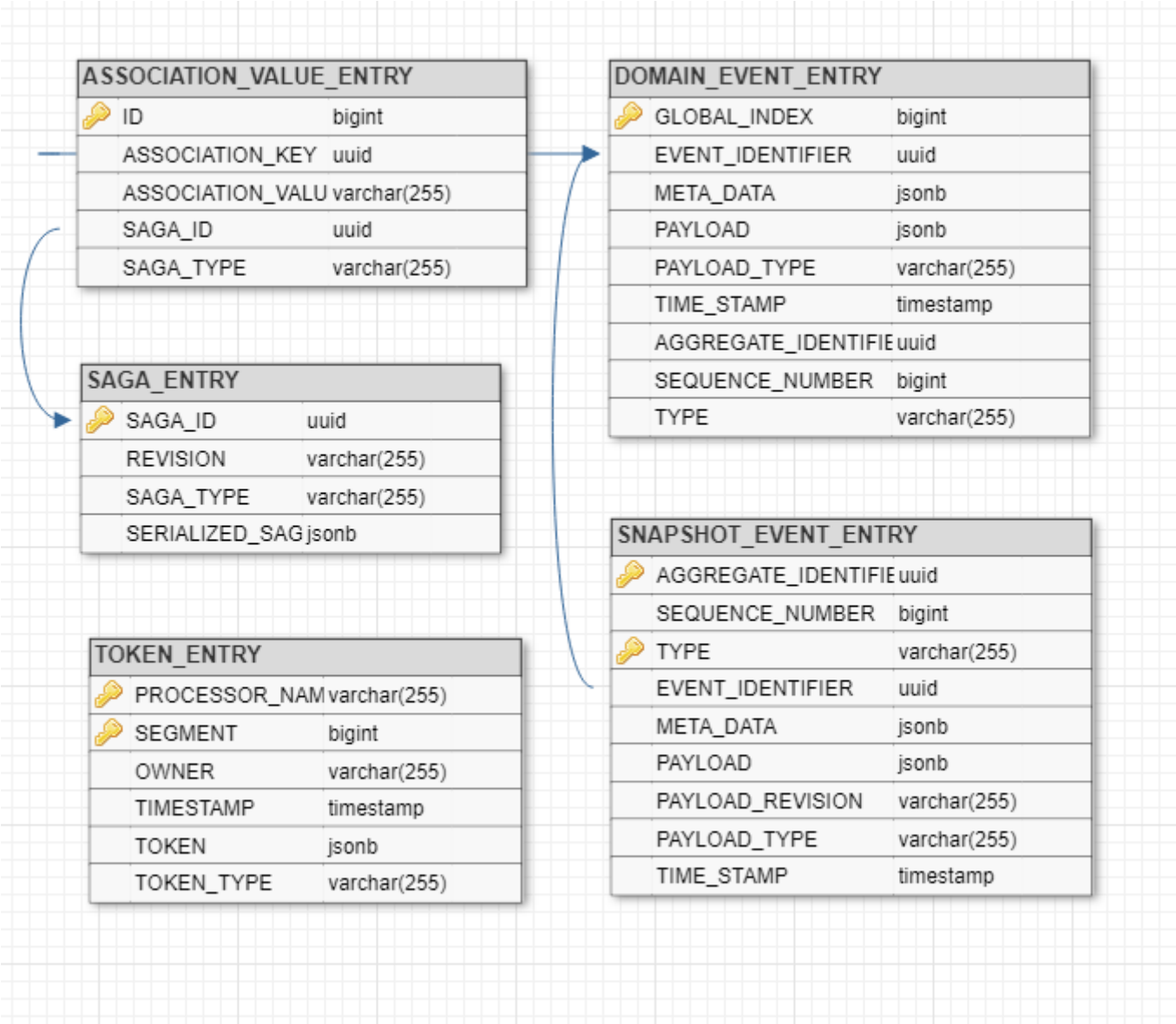


Рис. 11 Схема базы данных используемая Axon Framework

Таблица DOMAIN_EVENT_ENTRY используется для хранения всех событий. SAGA_ENTRY – хранит данные о саге (идентификатор, тип, данные о изменениях и сериализованное представление). ASSOCIATION_VALUE_ENTRY – позволяет создать связь многие ко многим с таблицами DOMAIN_EVENT_ENTRY и SAGA_ENTRY, связывая события с сагами через их идентификаторы. SNAPSHOT_EVENT_ENTRY – содержит сохраненные состояния агрегатов для более быстрого их восстановления из событий.

На следующем изображении (Рис.12) представлена схема хранящая основной набор данных используемых в процессе.

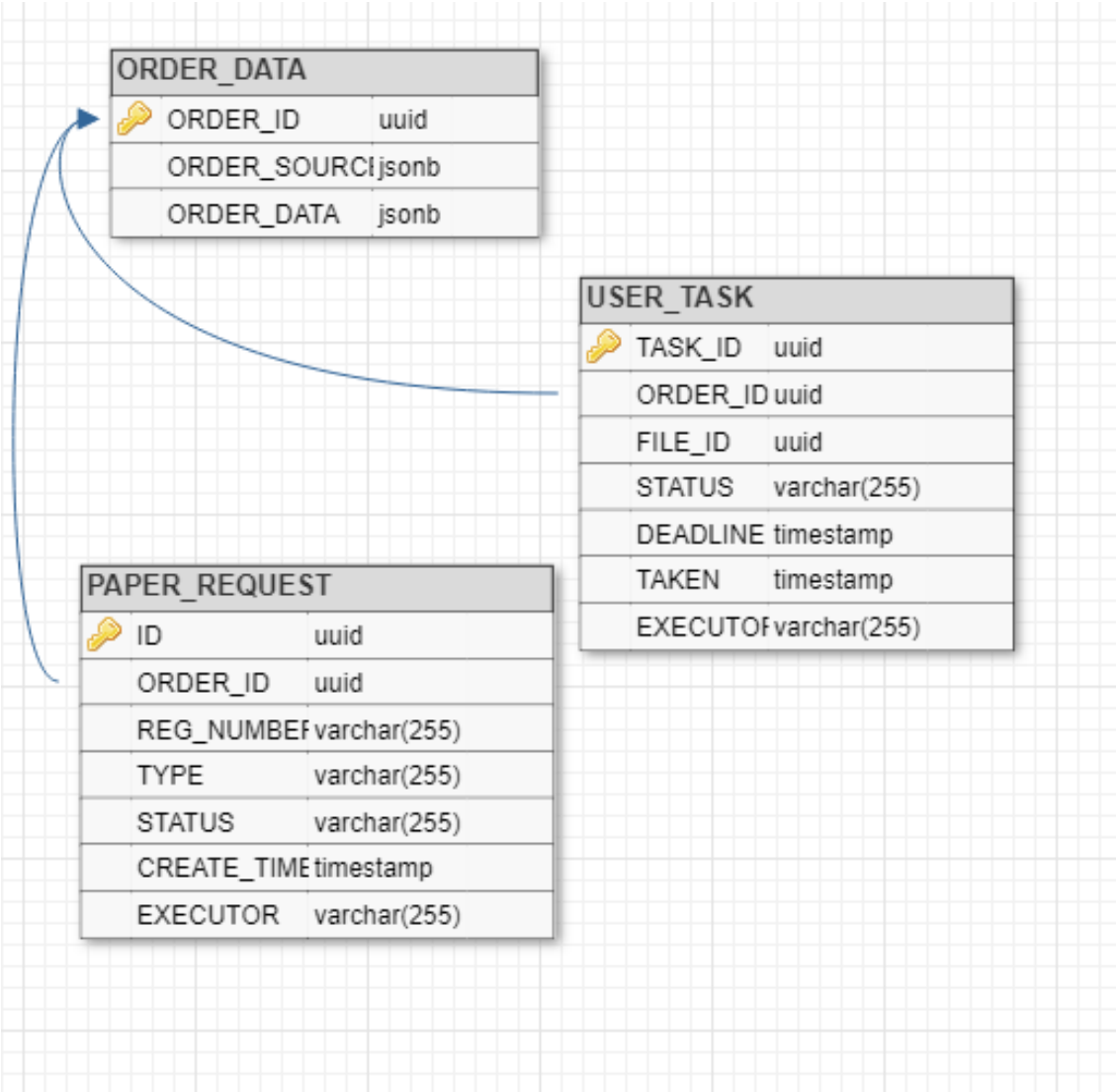


Рис. 12 Схема базы данных основного процесса

В таблице ORDER_DATA хранятся исходные данные заявления в формате JSON плоском виде (колонка ORDER_SOURCE) и в объектном представлении (колонка ORDER_DATA). Таблица USER_TASK содержит информацию о сформированных заданиях для пользователей:

- TASK_ID – идентификатор задания;
- ORDER_ID – идентификатор заявления;
- FILE_ID – идентификатор файла в файловом хранилище;

- TASK_TYPE – статус задания;
- DEADLINE – срок завершения задания;
- TAKEN – время выполнения;
- EXECUTOR – идентификатор пользователя, выполнившего задание.

Каждое задание связано с соответствующим заявлением через поле ORDER_ID. Аналогичный набор данные хранит таблица PAPER_REQUEST, но используется для хранения информации об межведомственных запросах, отправляемых в бумажном варианте.

Третьей схемой, используемой в системе, является схема представлений данных для отображений их в web-интерфейсе пользователей (Рис.13). Поскольку таблицы в ней заполняются сервисом представлений на основе данных полученных из событий, для удобства работы с таблицы представлений были вынесены в отдельную схему. На данном этапе в схеме находятся всего одна таблица, предоставляющая информацию для отображения на форме списка заявлений, но по мере развития системы их число будет расти.

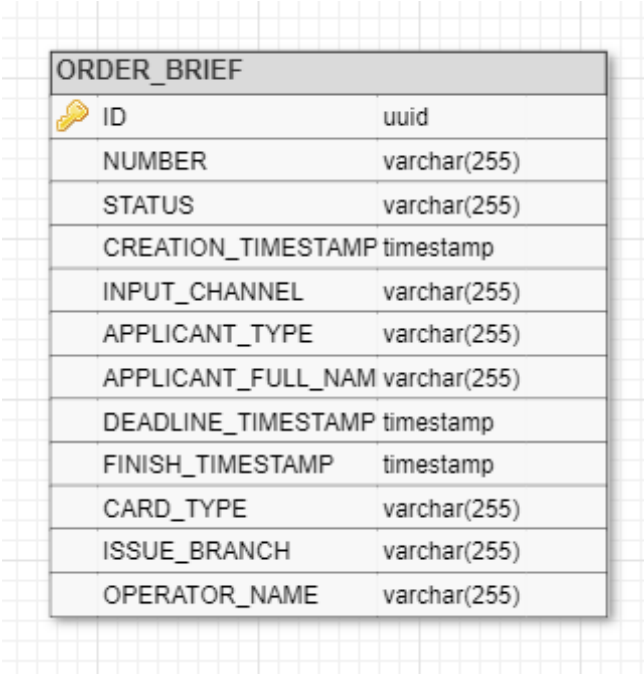


Рис. 13 Схема базы данных представлений

Единственная на данный момент таблица ORDER_BRIEF содержит следующие поля:

- ID – идентификатор заявления;
- NUMBER – номер заявления в АИС «МФЦ» или РПГУ;
- STATUS – статус процесса обработки заявления;
- CREATION_TIMESTAMP – дата получения заявления;
- INPUT_CHANNEL – источник получения заявления (АИС «МФЦ» или РПГУ);
- APPLICANT_TYPE – категория заявителя (школьник, студент, пенсионер);
- APPLICANT_FULL_NAME – ФИО заявителя;
- DEADLINE_TIMESTAMP – срок завершения обработки заявления;
- FINISH_TIMESTAMP – время завершения обработки заявления;
- CARD_TYPE – тип карты (электронная транспортная карта или кампусная карта);

- ISSUE_BRANCH – филиал для выдачи результата оказания услуги;
- OPERATOR_NAME – идентификатор оператора, обрабатывающего заявление

Заключение

В рамках выполненной работы был изучен процесс оказания услуги по выдаче электронной транспортной карты, определены этапы процесса, которые могут быть автоматизированы, сформулированы требования к функционалу и архитектуре приложения, была разработана архитектура системы принятия решения об оказании услуги по выдаче электронной транспортной карты, соответствующая этим требованиям, на основе построенной архитектуры было разработана система для соответствующая всем требованиям к функционалу. На данный момент система находится на стадии тестирования. Функционал системы в дальнейшем будет расширятся и улучшатся. Также планируется реализация подобных систем для оказания других услуг.

Список литературы

1. Методические рекомендации по работе с Единой системой межведомственного электронного взаимодействия Версия 2.5.6 https://smev.gosuslugi.ru/portal/api/files/get/80068 Дата обращения: 14.01.2019
2. Axon Reference Guide https://docs.axoniq.io/reference-guide/ Дата обращения: 20.03.2019
3. RabbitMQ Documentation https://www.rabbitmq.com/documentation.html Дата обращения: 20.05.2019
4. Spring Framework Documentation https://docs.spring.io/spring/docs/current/spring-framework-reference/index.html Дата обращения: 10.01.2019
5. Портал центров «Мои Документы» Тюменской области https://mfcto.ru Дата обращения: 10.01.2019
6. А.Д.Сараев, О.А.Щербина Труды Крымской Академии наук. - Симферополь: СОНАТ, 2006. - С. 47-59. СИСТЕМНЫЙ АНАЛИЗ И СОВРЕМЕННЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ
7. В.А.Кондратенко, С. В. Трубицына Проектирование и программная реализация экспертных систем на персональных ЭВМ: Пер. с англ./Предисл. Г.С. Осипова. - М.: Финансы и статистика, 1990. - 320 с: ил
8. Филимонова А.А., Боос Г.О. СОВРЕМЕННЫЕ АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ УПРАВЛЕНИЯ ТЕХНОЛОГИЧЕСКИМ ПРОЦЕССОМ // Инновации в науке: сб. ст. по матер. XXXVIII междунар. науч.-практ. конф. № 10(35). – Новосибирск: СибАК, 2014.

9. ПРИКАЗ от 1 апреля 2016 года N 93 ОБ УТВЕРЖДЕНИИ ПОРЯДКА ЭКСПЛУАТАЦИИ ИНФОРМАЦИОННОЙ СИСТЕМЫ "СИСТЕМА ИСПОЛНЕНИЯ РЕГЛАМЕНТОВ" http://docs.cntd.ru/document/429065470 Дата обращения: 10.05.2019
10. Eckel, Bruce. Thinking in Java / Bruce Eckel.—4th ed. 2006
11. Craig Walls Spring in Action, 5th Edition 2018
12. Documentation - PostgreSQL https://www.postgresql.org/docs/ Дата обращения: 06.02.2019
13. Reference - Kotlin Programming Language https://kotlinlang.org/docs/reference/ Дата обращения: 12.06.2019
14. Моуэт Э.Использование Docker / пер. с англ. А. В. Снастина; науч. ред. А. А. Маркелов. – М.: ДМК Пресс, 2017. – 354 с.: ил.
15. Э. Фримен, Э. Фримен, К. Сьерра, Б. Бейтс Паттерны проектирования. – СПб.: Питер, 2012. – 656с.
16. Apache CXF Documentation https://cxf.apache.org/docs/index.html Дата обращения: 10.01.2019

Приложение 1

Примеры JSON-объектов

JSON-файл в плоском представлении


```
{
  "CaseNumber": "000/2019/1",
  "CreateDate": "2019-06-10T12:44:48+00:00",
  "Files.Photo": "05358d2b-6c58-4aa5-b815-e49b3a87cca3",
  "MFCoperator": "Антонов Мокей Ильич",
  "filial.objectId": "1abd1c62-d20d-32be-ef84-84b02206e5e6",
  "ApplicantFL.Name": "Тимур",
  "ApplicantFL.SNILS": "123-456-789 0100000000",
  "FeedType.objectId": "88ab2bac-e2b3-b834-d411-dfd3658db3f4",
  "ApplicantFL.Surname": "Секунов",
  "ApplicantFL.BirthDate": "1995-07-10",
  "InputChannel.objectId": "db888db2-cae1-d9a9-f46e-5725d325efdf",
  "ApplicantFL.Patronymic": "Борисович",
  "SSUZ.SSUZName.objectId": "d345a35a-bbec-9122-b238-62c9e6f337b6",
  "TypeApplicant.objectId": "4c886c50-0ff3-4d0b-5363-61112074804b",
  "TypeUniversity.objectId": "452489fa-7052-88e2-89be-c3acbe71acc7",
  "WayOfRegistration.objectId": "6858672a-d8ad-5e73-609e-
bf7b421533e4",
  "ApplicantFL.Gender.objectId": "75ff79af-c535-8e02-93be-
d49650077eed",
  "ApplicantFL.FactAddress.city": "Чернышевское",
  "ApplicantFL.FactAddress.flat": "",
  "ApplicantFL.PermitDoc.Issuer": "SekunovTimur311",
  "ApplicantFL.PermitDoc.Number": "334659",
  "ApplicantFL.PermitDoc.Serial": "",
  "ApplicantFL.FactAddress.house": "51",
  "ApplicantFL.FactAddress.manual": "true",
  "ApplicantFL.FactAddress.region": "Тюменская область",
  "ApplicantFL.FactAddress.street": "Суворовский 1-й Переулок",
  "ApplicantFL.FactAddress.zipcode": "632323",
  "ApplicantFL.PermitDoc.IssueDate": "1995-07-10",
  "ApplicantFL.FactAddress.district": "",
  "ApplicantFL.FactAddress.fullAddress": "632323, Чернышевское, ул.
Суворовский 1-й Переулок, д. 51",
  "ApplicantFL.PermitDoc.Type.objectId": "1f83c9e1-2d51-072c-7f05-
e115c15c1919",
  "ApplicantFL.FactAddress.intracity_district": "",
  "ApplicantFL.PermitDoc.IdentifyDocAFC.objectId": "7cbc4de1-5b20-
0d7b-4ed7-c460a681f300"
}
```

JSON-файл в плоском представлении


```
{
  "orderId": null,
  "cardType": "UniversalElectronicCard",
  "feedType": "Applicant",
  "applicant": {
    "name": "Тимур",
    "type": "Student",
    "snils": "123-456-789 0100000000",
    "gender": null,
    "surname": "Секунов",
    "fullName": "Секунов Тимур Борисович",
    "birthDate": "1995-07-10",
    "permitDoc": {
      "type": "PassportRF",
      "issuer": "SekunovTimur311",
      "number": "334659",
      "series": "",
      "issueDate": "1995-07-10",
      "citizenship": null
    },
    "patronymic": "Борисович",
    "factAddress": {
      "city": "Чернышевское",
      "flat": "",
      "corps": null,
      "house": "51",
      "region": "Тюменская область",
      "street": "Суворовский 1-й Переулок",
      "zipCode": "632323",
      "building": null,
      "district": "",
      "fullAddress": "632323, Чернышевское, ул. Суворовский 1-й
Переулок, д. 51"
    }
  },
  "caseNumber": "000/2019/1",
  "properties": {
    "CaseNumber": "000/2019/1",
    "CreateDate": "2019-06-10T12:44:48+00:00",
    "Files.Photo": "05358d2b-6c58-4aa5-b815-e49b3a87cca3",
    "MFCoperator": "Антонов Мокей Ильич",
    "filial.objectId": "1abd1c62-d20d-32be-ef84-84b02206e5e6",
    "ApplicantFL.Name": "Тимур",
    "ApplicantFL.SNILS": "123-456-789 0100000000",
    "FeedType.objectId": "88ab2bac-e2b3-b834-d411-dfd3658db3f4",
    "ApplicantFL.Surname": "Секунов",
```

```
    "ApplicantFL.BirthDate": "1995-07-10",
    "InputChannel.objectId": "db888db2-cae1-d9a9-f46e-5725d325efdf",
    "ApplicantFL.Patronymic": "Борисович",
    "SSUZ.SSUZName.objectId": "d345a35a-bbec-9122-b238-62c9e6f337b6",
    "TypeApplicant.objectId": "4c886c50-0ff3-4d0b-5363-61112074804b",
    "TypeUniversity.objectId": "452489fa-7052-88e2-89be-c3acbe71acc7",
    "WayOfRegistration.objectId": "6858672a-d8ad-5e73-609e-
bf7b421533e4",
    "ApplicantFL.Gender.objectId": "75ff79af-c535-8e02-93be-
d49650077eed",
    "ApplicantFL.FactAddress.city": "Чернышевское",
    "ApplicantFL.FactAddress.flat": "",
    "ApplicantFL.PermitDoc.Issuer": "SekunovTimur311",
    "ApplicantFL.PermitDoc.Number": "334659",
    "ApplicantFL.PermitDoc.Serial": "",
    "ApplicantFL.FactAddress.house": "51",
    "ApplicantFL.FactAddress.manual": "true",
    "ApplicantFL.FactAddress.region": "Тюменская область",
    "ApplicantFL.FactAddress.street": "Суворовский 1-й Переулок",
    "ApplicantFL.FactAddress.zipcode": "632323",
    "ApplicantFL.PermitDoc.IssueDate": "1995-07-10",
    "ApplicantFL.FactAddress.district": "",
    "ApplicantFL.FactAddress.fullAddress": "632323, Чернышевское, ул.
Суворовский 1-й Переулок, д. 51",
    "ApplicantFL.PermitDoc.Type.objectId": "1f83c9e1-2d51-072c-7f05-
e115c15c1919",
    "ApplicantFL.FactAddress.intracity_district": "",
    "ApplicantFL.PermitDoc.IdentifyDocAFC.objectId": "7cbc4de1-5b20-
0d7b-4ed7-c460a681f300"
  },
  "issueBranch": "Неизвестный филиал: 1abd1c62-d20d-32be-ef84-
84b02206e5e6",
  "inputChannel": "SmartCenter",
  "providerCode": null,
  "registeredBy": "Антонов Мокей Ильич",
  "universityType": "College",
  "createTimestamp": "2019-06-10T12:44:48Z",
  "finishTimestamp": "2019-06-17T12:44:48Z",
  "attachedDocuments": {
    "photo": "05358d2b-6c58-4aa5-b815-e49b3a87cca3"
  }
}
```

Приложение 2

Общая структура электронного сообщения СМЭВ

```
<?xml version="1.0" encoding="utf-8"?>

<soapenv:Envelope xmlns:smev="http://smev.gosuslugi.ru/rev120315" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-

1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">

<!-- Заголовок электронного сообщения -->

<soapenv:Header>

<!— ЭП-ЕПГУ или ЭП-ОВ информационной системы, отправляющей электронное сообщение. Проверяется в СМЭВ -->

<wsse:Security soapenv:actor="http://smev.gosuslugi.ru/actors/smev">

<wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-messagesecurity-

1.0#Base64Binary" ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"

wsu:Id="SenderCertificate"><!-- Токен безопасности в Base64 --></wsse:BinarySecurityToken>

<ds:Signature Id="sender-wssec">

<ds:SignedInfo>

<ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>

<ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#gostr34102001-

gostr3411"/>

<ds:Reference URI="#sampleRequest">

<ds:Transforms>

<ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>

</ds:Transforms>
```

```
                                        <ds:DigestMethod
Algorithm="http://www.w3.org/2001/04/xmldsig-more#gostr3411"/>
                                        <ds:DigestValue><!-- Значение хеша в Base64 --
></ds:DigestValue>
                                </ds:Reference>
                        </ds:SignedInfo>
                        <ds:SignatureValue><!-- Значение подписи в Base64 --
></ds:SignatureValue>
                        <ds:KeyInfo>
                                <wsse:SecurityTokenReference>
                                <wsse:Reference URI="#SenderCertificate"
ValueType="http://docs.oasisopen.
                                org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509v3"/>
                                </wsse:SecurityTokenReference>
                        </ds:KeyInfo>
                </ds:Signature>
        </wsse:Security>
        <!-- ЭП-СМЭВ. Проверяется в информационной системе, получающей электронное
сообщение. -->
        <wsse:Security soapenv:actor="http://smev.gosuslugi.ru/actors/recipient">
                <wsse:BinarySecurityToken EncodingType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-soap-messagesecurity-
                1.0#Base64Binary" ValueType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
                wsu:Id="SMEVCertificate"><!-- Токен безопасности в Base64 --
></wsse:BinarySecurityToken>
                <ds:Signature Id="smev-wssec">
                        <ds:SignedInfo>
                                <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                                <ds:SignatureMethod
Algorithm="http://www.w3.org/2001/04/xmldsig-more#gostr34102001-
                                gostr3411"/>
```

```
                                    <ds:Reference URI="#sampleRequest">
                                        <ds:Transforms>
                                            <ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                                        </ds:Transforms>
                                    <ds:DigestMethod
Algorithm="http://www.w3.org/2001/04/xmldsig-more#gostr3411"/>
                                    <ds:DigestValue><!--    Значение    хеша    в    Base64    --
></ds:DigestValue>
                                    </ds:Reference>
                                    <ds:Reference URI="#smevHeader">
                                        <ds:Transforms>
                                            <ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                                        </ds:Transforms>
                                    <ds:DigestMethod
Algorithm="http://www.w3.org/2001/04/xmldsig-more#gostr3411"/>
                                    <ds:DigestValue><!--    Значение    хеша    в    Base64    --
></ds:DigestValue>
                                    </ds:Reference>
                                </ds:SignedInfo>
                                <ds:SignatureValue><!--    Значение    подписи    в    Base64    --
></ds:SignatureValue>
                                <ds:KeyInfo>
                                    <wsse:SecurityTokenReference>
                                        <wsse:Reference              URI="#SMEVCertificate"
ValueType="http://docs.oasisopen.
                                        org/wss/2004/01/oasis-200401-wss-x509-token-
profile-1.0#X509v3"/>
                                    </wsse:SecurityTokenReference>
                                </ds:KeyInfo>
                            <!-- Метка времени внутри ЭП-СМЭВ. -->
                                <ds:Object>
```

```
                                        <xds:QualifyingProperties
xmlns:xds="http://uri.etsi.org/01903/v1.1.1#">
                                        <xds:UnsignedProperties>
                                        <xds:UnsignedSignatureProperties>
                                        <xds:SignatureTimeStamp>
                                        <xds:HashDataInfo   uri="#signature-value-40ddb6ca-9ac1-
4026-a049-76901f3aa5d8"/>
                                        <xds:EncapsulatedTimeStamp>Метка        времени        в
Base64</xds:EncapsulatedTimeStamp>
                                        </xds:SignatureTimeStamp>
                                        </xds:UnsignedSignatureProperties>
                                        </xds:UnsignedProperties>
                                       </xds:QualifyingProperties>
                                       </ds:Object>
                            </ds:Signature>
                </wsse:Security>
                <!-- Унифицированный служебный заголовок СМЭВ. Подписывается ЭП СМЭВ. -->
                <smev:Header wsu:Id="smevHeader">
                        <smev:NodeId>Уникальный идентификатор узла СМЭВ</smev:NodeId>
                        <smev:MessageId>Уникальный код цепочки сообщений в СМЭВ</smev:MessageId>
                        <smev:TimeStamp>Дата получения сообщения СМЭВ</smev:TimeStamp>
                        <smev:MessageClass>REQUEST</smev:MessageClass>
                </smev:Header>
        </soapenv:Header>
        <!-- Тело электронного сообщения -->
        <soapenv:Body wsu:Id="sampleRequest">
                <smevSampleMsg:sampleRequest
xmlns:smevSampleMsg="http://smev.gosuslugi.ru/SampleMessage">
                        <!--   Унифицированный   служебный   блок   атрибутов   сообщения   СМЭВ.
Подписывается ЭП ОВ информационной
                        системы, отправляющей электронное сообщение -->
                        <smev:Message>
```

<smev:Sender><!--Данные о системе-инициаторе взаимодействия (Потребителе) --></smev:Sender>

<smev:Recipient><!--Данные о системе-получателе сообщения (Поставщике) --></smev:Recipient>

<smev:Originator><!--Данные о системе, инициировавшей цепочку из нескольких запросов-ответов,

объединенных единым процессом в рамках взаимодействия --></smev:Originator>

<smev:Service>

<smev:Code><!-Мнемоника сервиса--></smev:Code>

<smev:Version><!-Версия сервиса--></smev:Version>

</smev:Service>

<smev:TypeCode><!--Тип сообщения по классификатору сообщений в СМЭВ --></smev:TypeCode>

<smev:Status><!-- Статусе электронного сообщения по классификатору статусов --></smev:Status>

<smev:Date><!--Дата создания сообщения--></smev:Date>

<smev:RequestIdRef><!--Идентификатор сообщения-запроса, инициировавшего взаимодействие --></smev:RequestIdRef>

<smev:OriginRequestIdRef><!--Идентификатор сообщения-запроса, инициировавшего цепочку из

нескольких запросов-ответов, объединенных единым процессом в рамках взаимодействия --></smev:OriginRequestIdRef>

<smev:ServiceCode>< !--Код государственной услуги указывается в соответствии с правилами кодификации, установленными в ИС Сводного реестра государственных услуг --></smev:ServiceCode>

<smev:CaseNumber>< !--Номер дела указывается в соответствии с правилами, установленными в информационной системы-отправителя. --></smev:CaseNumber>

<smev:ExchangeType>< !-- Признак принадлежности электронного сообщения различным категориям взаимодействия. Указвается в соответствие с классификатором категорий взаимодействия--></smev:ExchangeType>

<smev:TestMsg>< !--Признак тестового электронного сообщения: запроса или ответа. Не указывается при продуктивном взаимодействии. --></smev:TestMsg>

```
                                                    <smev:OKTMO><!-- Код ОКТМО . --
></smev:OKTMO>
                        </smev:Message>
                        <!-- Унифицированный служебный блок-обертка передаваемых данных сообщения
СМЭВ. Данные электронного
                        сообщения -->
                        <smev:MessageData>
                                <!-- Унифицированный блок-обертка для передачи информации в
соответствии с требованиями
                                поставщика -->
                                <smev:AppData><!-- Данные из ОИВ --></smev:AppData>
                                <!-- Унифицированный блок передачи прикладных данных -->
                                <smev:AppDocument><!-- Передаваемый ZIP-архив в Base64 --
></smev:AppDocument>
                        </smev:MessageData>
                </smevSampleMsg:sampleRequest>
        </soapenv:Body>
        </soapenv:Envelope>
```

Приложение 3

Реализация класса Order

```
package ru.mfcto.kilterkit.core.order


import org.axonframework.commandhandling.CommandHandler
import org.axonframework.eventsourcing.EventSourcingHandler
import org.axonframework.modelling.command.AggregateIdentifier
import org.axonframework.modelling.command.AggregateLifecycle
import org.axonframework.spring.stereotype.Aggregate
import ru.mfcto.kilterkit.core.api.order.*
import
ru.mfcto.kilterkit.core.api.order.persistence.OrderPersistenceService
import ru.mfcto.kilterkit.core.api.validation.ValidationError
import ru.mfcto.kilterkit.core.order.data.OrderDataService
import java.util.*


@Suppress("UNUSED_PARAMETER")
@Aggregate
class Order {


    @AggregateIdentifier
    private lateinit var orderId: UUID
    private var validationErrors: Set<ValidationError> = setOf()


    constructor() // for Axon
```

```
        @CommandHandler
        constructor(cmd: RegisterOrder, orderPersistenceService:
OrderPersistenceService) {
            try {
                orderPersistenceService.saveOrderSource(cmd.orderId,
cmd.orderSource)

AggregateLifecycle.apply(OrderRegistered(cmd.orderId))
            } catch (e: Exception) {

AggregateLifecycle.apply(OrderRegistrationFailed(cmd.orderId,
e.localizedMessage))
            }
        }

        @CommandHandler
        fun handle(
            cmd: ParseOrder,
            orderPersistenceService: OrderPersistenceService,
            orderDataService: OrderDataService
        ) {
            try {
                val orderSource =
orderPersistenceService.loadOrderSource(orderId)
```

```
                val orderData =
orderDataService.readOrderData(orderSource)
                orderPersistenceService.saveOrderData(orderId,
orderData)
                AggregateLifecycle.apply(OrderParsed(orderId))
            } catch (e: Exception) {
                AggregateLifecycle.apply(OrderParsingFailed(orderId,
e.localizedMessage))
            }
        }

        @CommandHandler
        fun handle(cmd: ValidateOrder) {
            AggregateLifecycle.apply(OrderValidationStarted(orderId))
        }

        @CommandHandler
        fun handle(cmd: FinishOrderValidation) {
            if (cmd.validationErrors.isEmpty()) {
AggregateLifecycle.apply(OrderValidationSucceeded(orderId))
            } else {
AggregateLifecycle.apply(OrderValidationFailed(orderId,
cmd.validationErrors))
            }
```

```
}

@CommandHandler
fun handle(cmd: SetPupilInfo) {
    println("Set pupil info: ${cmd.pupilInfo}")
}

@EventSourcingHandler
fun on(event: OrderRegistered) {
    orderId = event.orderId
}

@EventSourcingHandler
fun on(event: OrderRegistrationFailed) {
    orderId = event.orderId
}

@EventSourcingHandler
fun on(event: OrderParsed) {
    // no op
}

@EventSourcingHandler
fun on(event: OrderValidationStarted) {
    // no op
```

```
    }

    @EventSourcingHandler
    fun on(event: OrderValidationSucceeded) {
        // no op
    }

    @EventSourcingHandler
    fun on(event: OrderValidationFailed) {
        validationErrors = event.validationErrors
    }
}
```

Приложение 4

Реализация OrderSaga

```
package ru.mfcto.kilterkit.core.order

import org.axonframework.commandhandling.gateway.CommandGateway
import org.axonframework.modelling.saga.EndSaga
import org.axonframework.modelling.saga.SagaEventHandler
import org.axonframework.modelling.saga.StartSaga
import org.axonframework.spring.stereotype.Saga
import org.springframework.beans.factory.annotation.Autowired
import ru.mfcto.kilterkit.core.api.order.*
import ru.mfcto.kilterkit.core.api.pdfgen.RefuseGenService
import ru.mfcto.kilterkit.core.api.pdfgen.dto.AppendDocDto
import ru.mfcto.kilterkit.view.orderlist.OrderBriefRepository
```

```
import java.util.*

@Saga
class OrderSaga @Autowired constructor(
    private val refuseGenService: RefuseGenService,
    private val orderBriefRepository: OrderBriefRepository
) {

    @Transient
    @Autowired
    private lateinit var commandGateway: CommandGateway

    @StartSaga
    @SagaEventHandler(associationProperty = "orderId")
    fun on(event: OrderRegistered) {
        commandGateway.send<Any>(ParseOrder(event.orderId))
    }

    @SagaEventHandler(associationProperty = "orderId")
    fun on(event: OrderParsed) {
        commandGateway.send<Any>(ValidateOrder(event.orderId))
    }

    @EndSaga
    @SagaEventHandler(associationProperty = "orderId")
```

```
    fun on(event: OrderParsingFailed) {
        // no op
    }

    @EndSaga
    @SagaEventHandler(associationProperty = "orderId")
    fun on(event: OrderValidationFailed) {
        val errors = mutableListOf<String>()
        for (e in event.validationErrors)
            errors += e.reasonText
        val fileUUID = refuseGenService.genRefuse(event.orderId,
errors)

        val orderData =
orderBriefRepository.findById(event.orderId).get()
        refuseGenService.appendRefuse(AppendDocDto().apply {
            orderNum = orderData.number
            docs = listOf(AppendDocDto.DocData().apply {
                fileID = UUID.fromString(fileUUID)
                fileName = "ref${event.orderId}.pdf"
                docName = "Уведомление об отказе
${orderData.number.replace("/", "")}"
            })
        })
    }
```

```
}
```